

CSPC702 - EMBEDDED SYSTEMS AND INTERNET OF THINGS(IOT)

UNIT – I Introduction to Embedded Systems

Syllabus- Unit I

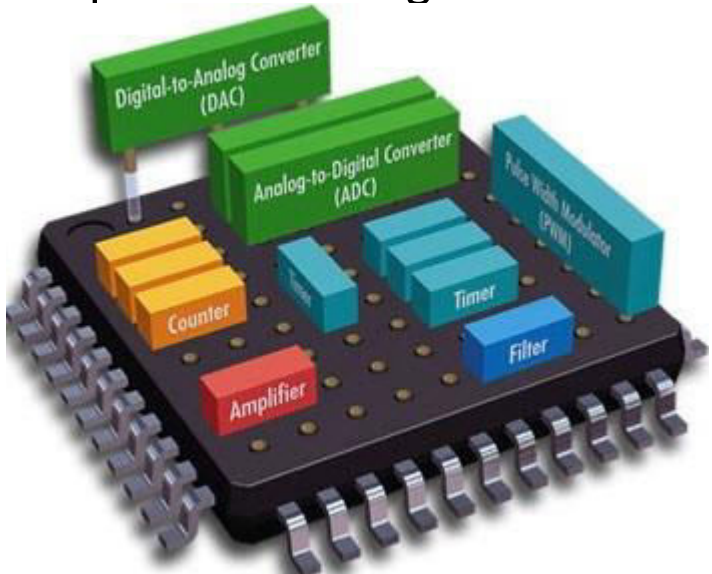
Introduction to Embedded Systems

- Introduction to Embedded Systems
- Applications of embedded system
- Features and Attributes of Embedded System
- Challenges in Embedded System
- Selection of Processors
- Recent trends in embedded system
- Embedded Firmware design approaches and development languages
- Embedded development life cycle.

Introduction to Embedded Systems

Definition

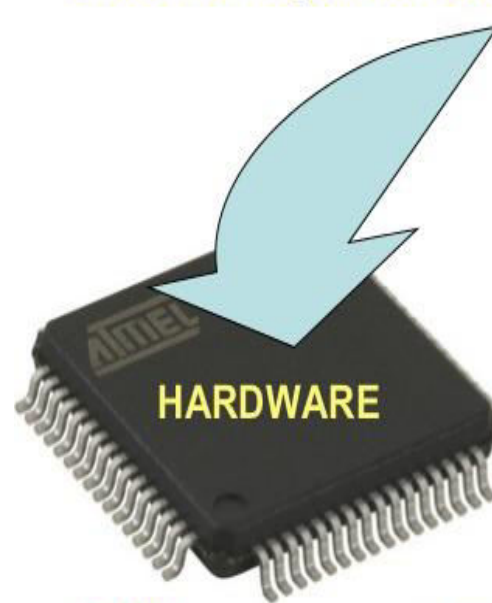
- It is an Electronic/Electro-mechanical system designed to perform a specific function and is a combination of both hardware & software. OR
- A combination of hardware and software which together form a component of a larger machine.



EMBEDDED SYSTEM

Definition: An Embedded System is one that has computer hardware with software embedded in it as one of its important components.

Its software embeds in ROM (Read Only Memory). It does not need secondary memories as in a computer



SOFTWARE PROGRAM

```
#include <16f876a.h>
#use delay (clock=2000000)
#byte PORTB=6
main()
{
    set_tris_b(0);
    portb=255; //decimal
    delay_ms(1000);
    portb=0x55; //hexadecimal
    delay_ms(1000);
    portb=0b10101010; //binary
    delay_ms(500);
}
```

03.01.09

murugan_m1@yahoo.com 996576

6

What is an Embedded system?

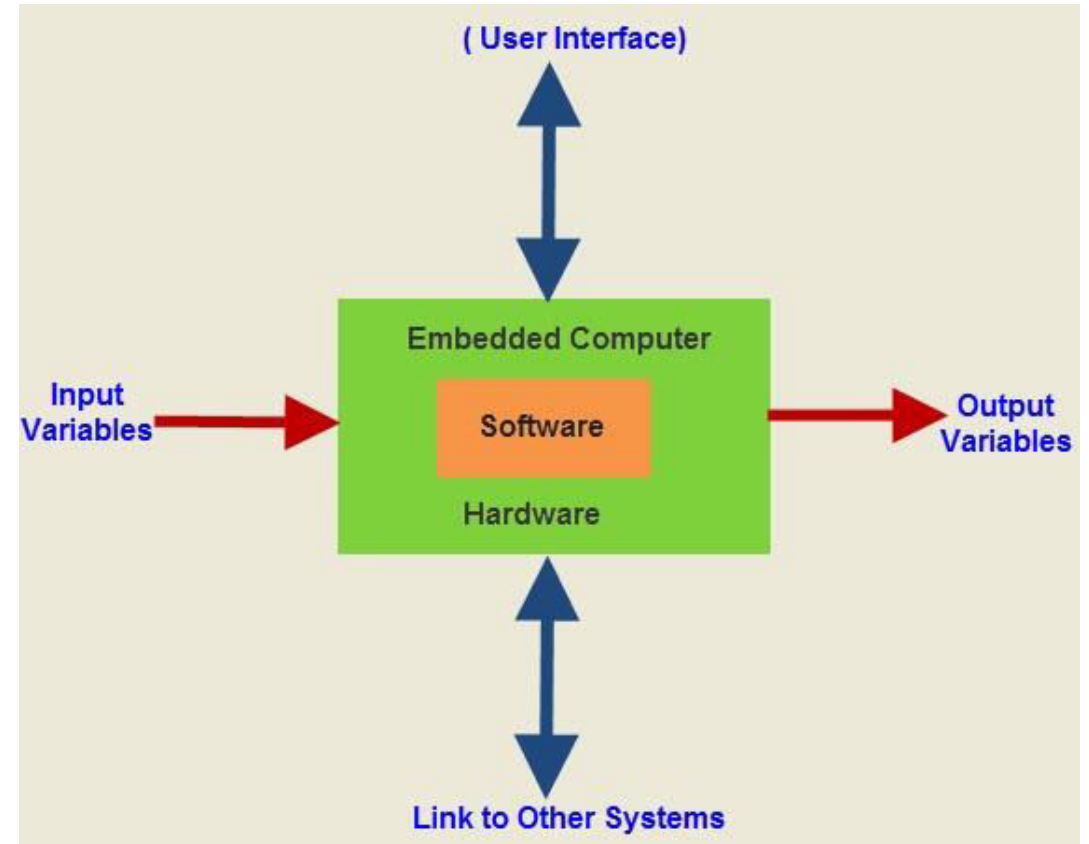
- An embedded system is one that has computer hardware with software embedded in it as one of its components.

Or

- We can define an embedded system as “A microprocessor based system that does not look like a computer”.

Or

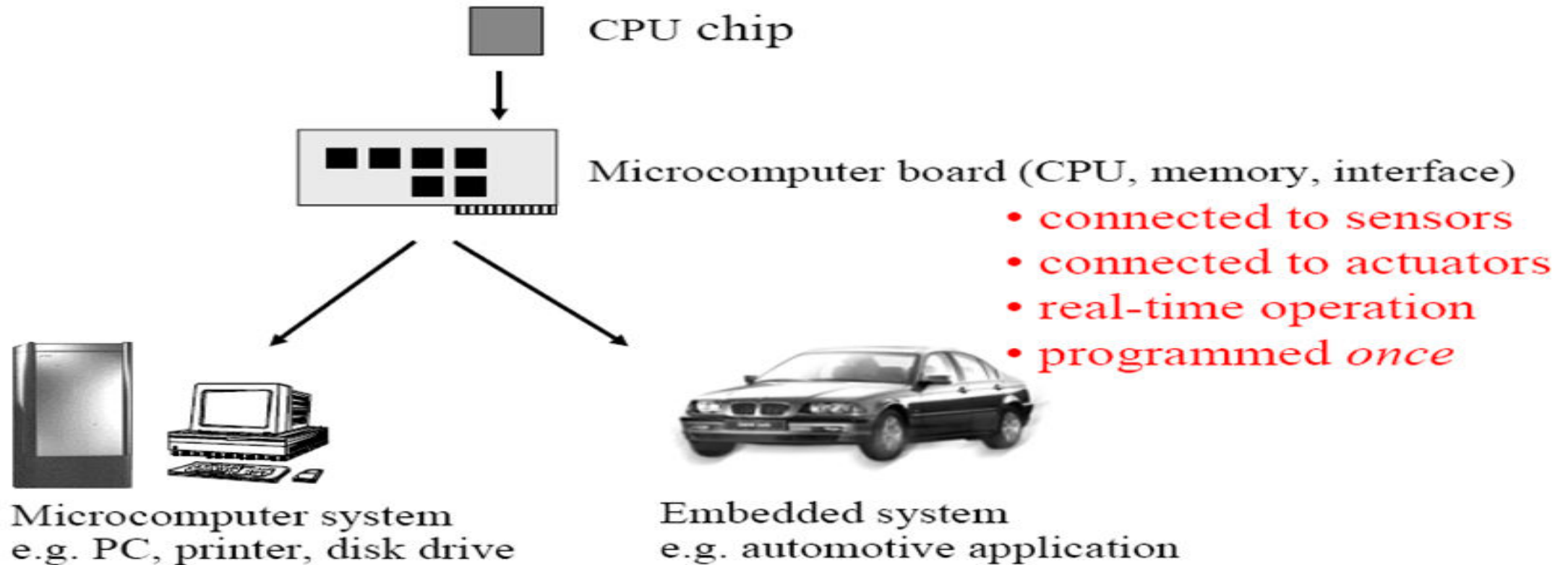
- we can say that it is “A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, as is the case of an antilock braking system in a car”.



What is an Embedded system?

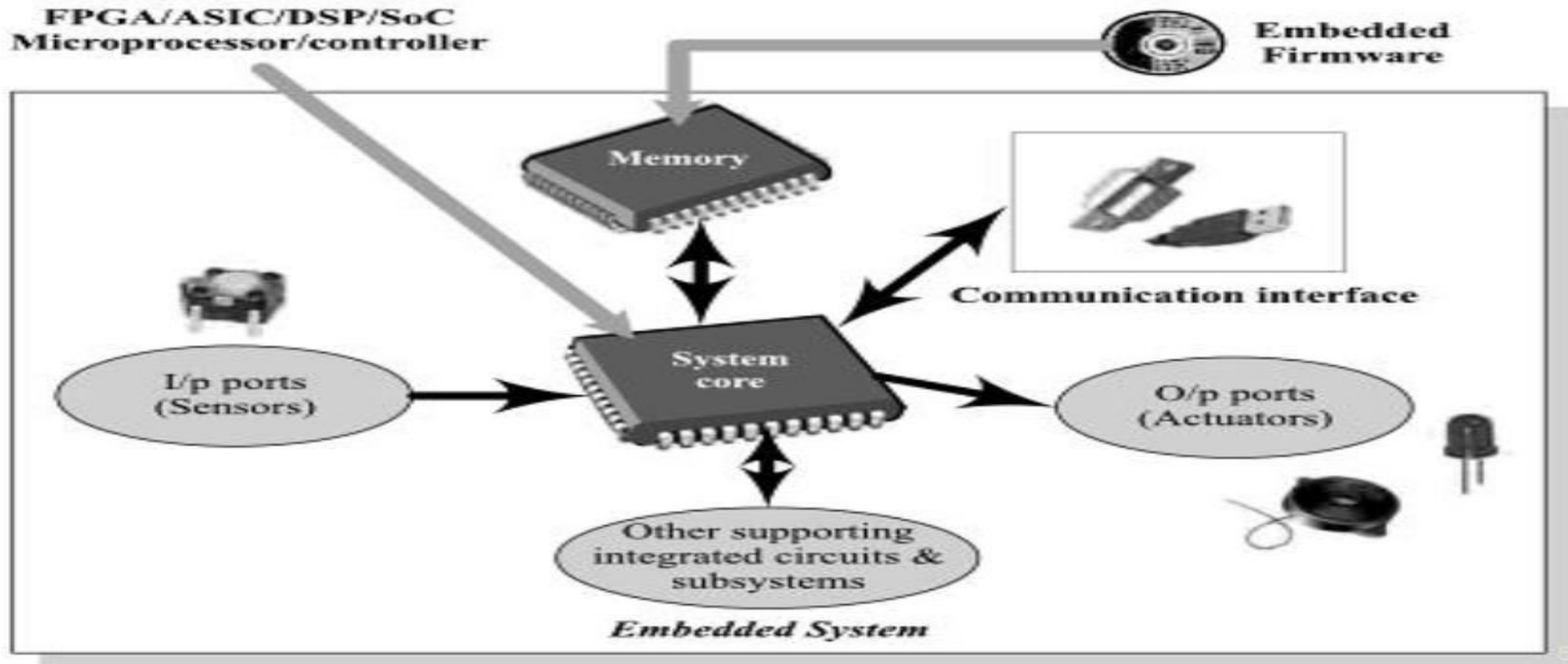
- An Embedded system is a combination of computer hardware and software. As with any electronic system, this system requires a hardware platform and that is built with a microprocessor or [microcontroller](#).
- The Embedded system hardware includes elements like
 - user interface,
 - Input/Output interfaces,
 - display and memory, etc.
- Generally, an embedded system comprises power supply, processor, memory, timers, serial communication ports and system application specific circuits.
- Embedded system software is written in a high-level language, and then compiled to achieve a specific function within a non-volatile memory in the hardware.
- Embedded system software is designed to keep in view of three limits. They are availability of system memory and processor speed.
- When the system runs endlessly, there is a need to limit the power dissipation for events like run, stop and wake up.

Introduction to Embedded Systems (contd.)



- An example of an embedded system is a microprocessor that controls an automobile engine.
- An embedded system is designed to run on its own without human intervention, and may be required to respond to events in realtime.

Introduction to Embedded Systems (contd.)



System Examples

SYSTEM EXAMPLES

WATCH

It is a time display **SYSTEM**

Parts: Hardware, Needles, Battery, Dial, Chassis and Strap



Rules

- 6. All needles move clockwise only
- 7. A thin needle rotates every second
- 8. A long needle rotates every minute
- 9. A short needle rotates every hour
- 10. All needles return to the original position after 12 hours



SYSTEM EXAMPLES

WASHING MACHINE

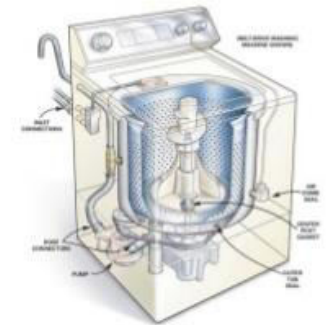
It is an automatic clothes washing **SYSTEM**

Parts: Status display panel, Switches & Dials, Motor, Power supply & control unit, Inner water level sensor and solenoid valve.



Rules

- 5. Wash by spinning
- 6. Rinse
- 7. Drying
- 8. Wash over by blinking
- 9. Each step display the process stage
- 10. In case interruption, execute only the remaining

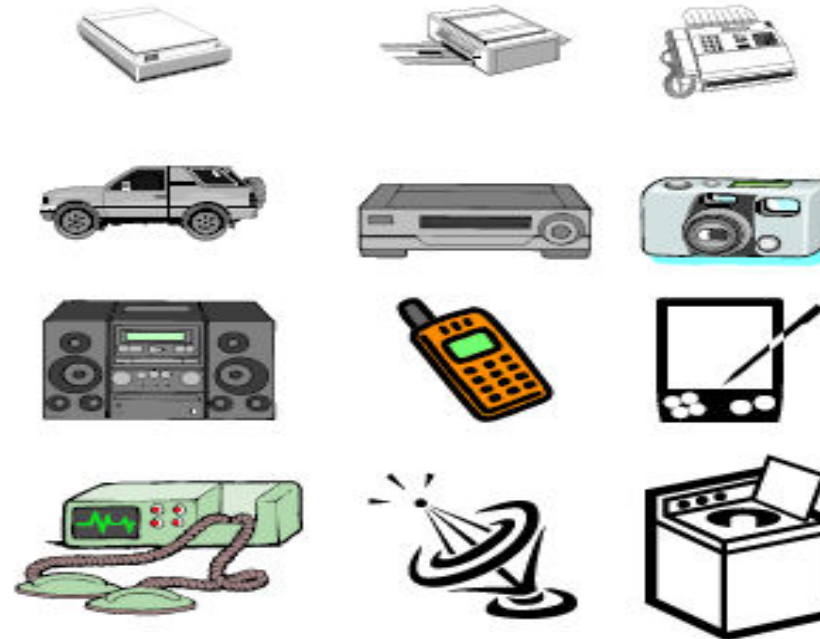


Introduction to Embedded Systems (contd.)

A “short list” of embedded systems

Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Automatic transmission
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Curbside check-in systems
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems

Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR's, DVD players
Video game consoles
Video phones
Washers and dryers

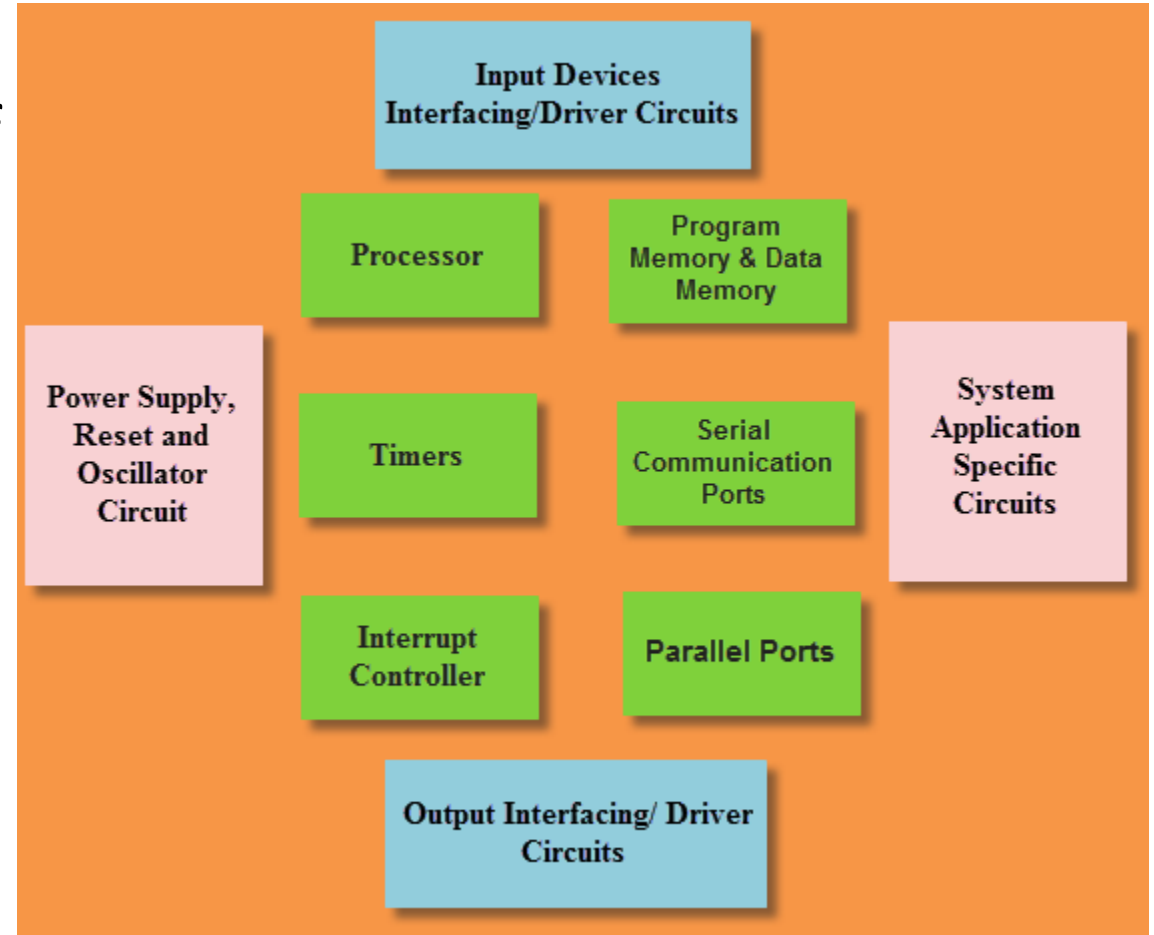


History of Embedded system

- Here, are important milestones from the history of embedded system:
- In 1960, embedded system was first used for developing Apollo Guidance System by Charles Stark Draper at MIT.
- In 1965, Autonetics, developed the D-17B, the computer used in the Minuteman missile guidance system.
- In 1968, the first embedded system for a vehicle was released.
- Texas Instruments developed the first microcontroller in 1971.
- In 1987, the first embedded OS, VxWorks, was released by Wind River.
- Microsoft's Windows embedded CE in 1996.
- By the late 1990s, the first embedded Linux system appeared.
- The embedded market reach \$140 billion in 2013.
- Analysts are projecting an Embedded market larger than \$40 billion by 2030.

Components of Embedded Systems

- The embedded systems basics include the components of embedded system hardware, embedded system types and several characteristics.
- An embedded system has three main components:
 - Embedded system hardware,
 - Embedded system software and
 - Operating system.



Components of Embedded Systems

Embedded System Hardware

- As with any electronic system, an embedded system requires a hardware platform on which it performs the operation. Embedded system hardware is built with a microprocessor or microcontroller. The embedded system hardware has elements like input output (I/O) interfaces, user interface, memory and the display. Usually, an embedded system consists of:
 - Power Supply
 - Processor
 - Memory
 - Timers
 - Serial communication ports
 - Output/Output circuits
 - System application specific circuits

Components of Embedded Systems

Embedded System Software

- The embedded system software is written to perform a specific function. It is typically written in a high level format and then compiled down to provide code that can be lodged within a non-volatile memory within the hardware.
- An embedded system software is designed to keep in view of the three limits:
 - Availability of system memory
 - Availability of processor's speed
 - When the system runs continuously, there is a need to limit power dissipation for events like stop, run and wake up.

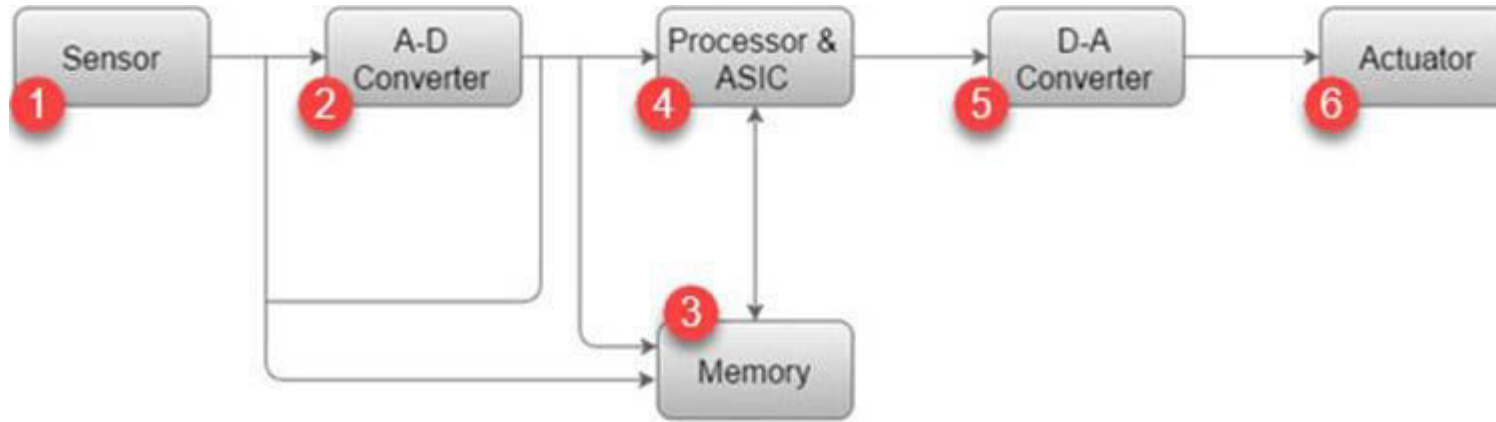
Components of Embedded Systems

Real Time Operating System

- A system is said to be real time, if it is essential to complete its work and deliver its service on time. Real time operating system manages the application software and affords a mechanism to let the processor run. The Real Time operating system is responsible for handling the hardware resources of a computer and host applications which run on the computer.
- An RTOS is specially designed to run applications with very precise timing and a high amount of reliability. Especially, this can be important in measurement and industrial automation systems wherein downtime is costly or a program delay could cause a safety hazard.

Architecture of the Embedded System

Below is basic architecture of the Embedded System:



1) Sensor:

Sensor helps you to measure the physical quantity and convert it to an electrical signal. It also stores the measured quantity to the memory. This signal can be read by an observer or by any electronic instrument such as A2D converter.

2) A-D Converter:

A-D converter (analog-to-digital converter) allows you to convert an analog signal sent by the sensor into a digital signal.

3) Memory:

Memory is used to store information. Embedded System majorly contains two memory cells 1) Volatile 2) Non volatile memory.

Architecture of the Embedded System

4) Processor & ASICs:

- This component processes the data to measure the output and store it to the memory.

5) D-A Converter:

- D-A converter (A digital-to-analog converter) helps you to convert the digital data fed by the processor to analog data.

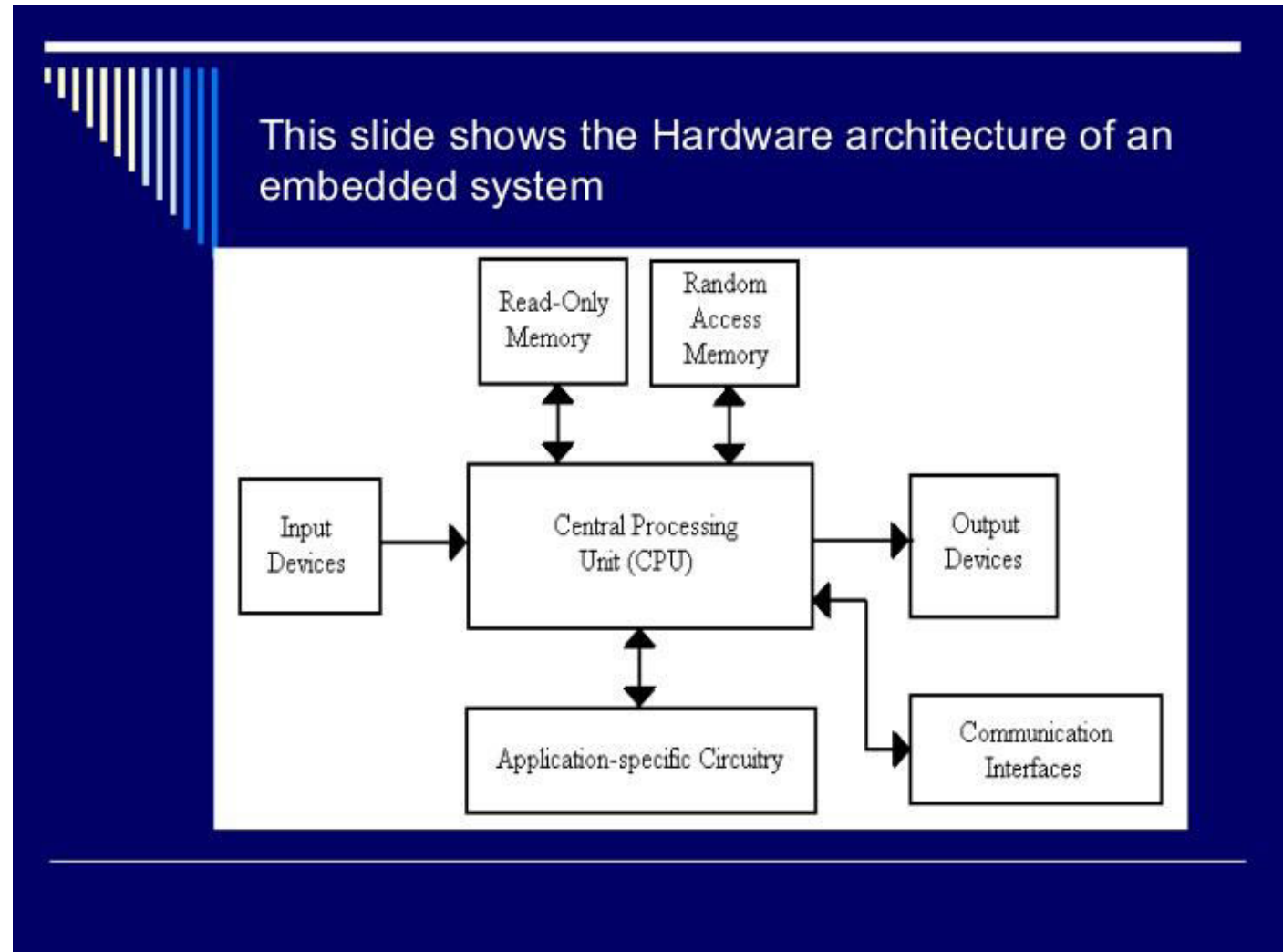
6) Actuator:

- An actuator allows you to compare the output given by the D-A converter to the actual output stored in it and stores the approved output in the memory.

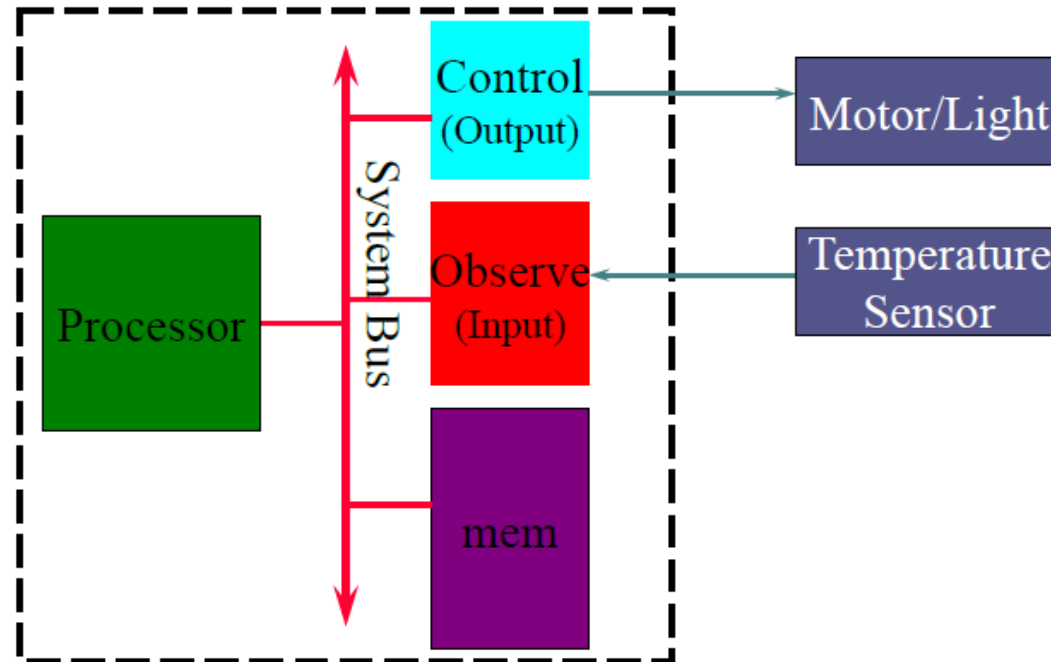
Architecture of an Embedded system

Now let us see the details of the various building blocks of the hardware of an embedded system.

- Central Processing Unit (CPU)
- Memory (Read only memory and Random access memory)
- Input Devices
- Output Devices
- Communication interfaces
- Application specific circuitry



Embedded System Block Diagram



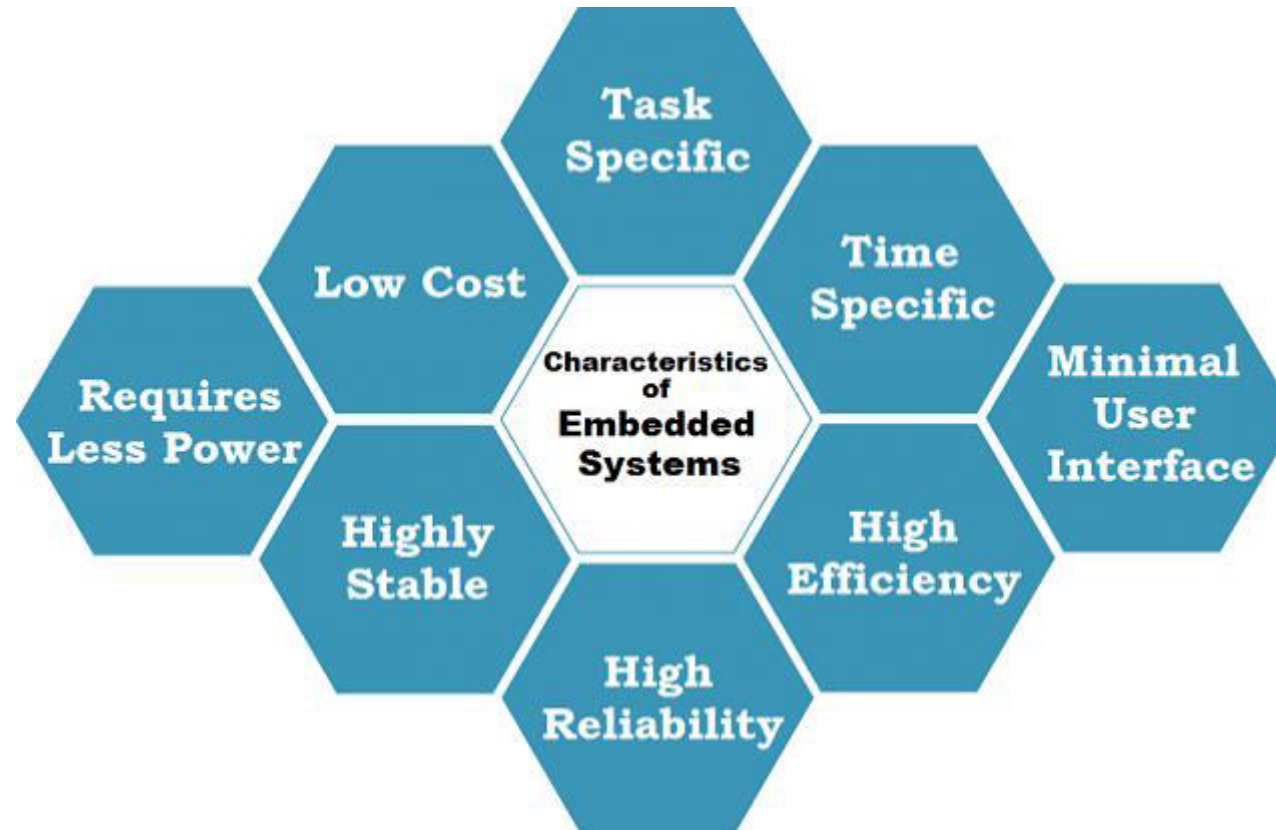
Significance of Embedded System

- Due to their compact size, low cost and simple design aspects made embedded systems very popular and encroached into human lives and have become indispensable.
- They are found everywhere from kitchen ware to space craft.
- **What makes embedded systems different?**
- Real-time operation
- size
- cost
- time
- reliability
- safety
- energy
- security

Important terminologies used in embedded system

- Now in this Embedded Systems tutorial, we will cover some important terms used in embedded system.
- **Reliability:** This measure of the survival probability of the system when the function is critical during the run time.
- **Fault-Tolerance:** Fault-Tolerance is the capability of a computer system to survive in the presence of faults.
- **Real-Time:**
 - Embedded system must meet various timing and other constraints. They are imposed on it by the real-time natural behavior of the external world.
 - For example, an airforce department which keeps track of incoming missile attacks must precisely calculate and plan their counter-attack due to hard real-time deadline. Otherwise, it'll get destroyed.
- **Flexibility:**
 - It's building systems with built-in debugging opportunities which allows remote maintenance.
 - For example, you are building a spacecraft which will land on another planter to collect various types of data and send collected detail back to us. If this spacecraft went insane and lost the control, we should be able to make some important diagnostic. So, flexibility is vital while designing an embedded system.
- **Portability:**
 - Portability is a measure of the ease of using the same embedded software in various environments. It requires generalized abstractions between the application program logic itself and the low-level system interfaces.

Characteristics of Embedded system



Characteristics of Embedded system

- Embedded Systems are task specific. They do the same task repeatedly /continuously over their lifetime. An mp3 player will function only as an mp3 player.
- Embedded systems are created to perform the task within a certain time frame. It must therefore perform fast enough. A car's brake system, if exceeds the time limit, may cause accidents.
- They have minimal or no user interface (UI). A fully automatic washing machine works on its own after the programme is set and stops once the task is over.
- Some embedded systems are designed to react to external stimuli and react accordingly. A thermometer, a GPS tracking device.
- Embedded systems are built to achieve certain efficiency levels. They are small sized, can work with less power and are not too expensive.
- Embedded systems cannot be changed or upgraded by the users. Hence, they must rank high on reliability and stability. They are expected to function for long durations without the user experiencing any difficulties.
- Microcontroller or microprocessors are used to design embedded systems.
- Embedded systems need connected peripherals to attach input & output devices.
- The hardware of an embedded-system is used for security and performance. The Software is used for features.

Characteristics of Embedded system

- **Single-functioned** – An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager.
- **Tightly constrained** – All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. It must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.
- **Reactive and Real time** – Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay. Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors. It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to control of the car.
- **Microprocessors based** – It must be microprocessor or microcontroller based.
- **Memory** – It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer.
- **Connected** – It must have connected peripherals to connect input and output devices.
- **HW-SW systems** – Software is used for more features and flexibility. Hardware is used for performance and security.

Characteristics of Embedded system

- Application-specific functionality – specialized for one or one class of applications
- Deadline constrained operation – system may have to perform its function(s) within specific time periods to achieve successful results
- Resource challenged – systems typically are configured with a modest set of resources to meet the performance objectives
- Power efficient – many systems are battery-powered and must conserve power to maximize the usable life of the system.
- Form factor – many systems are light weight and low volume to be used as components in host systems
- Manufacturable – usually small and inexpensive to manufacture based on the size and low complexity of the hardware.

Classification of Embedded Systems

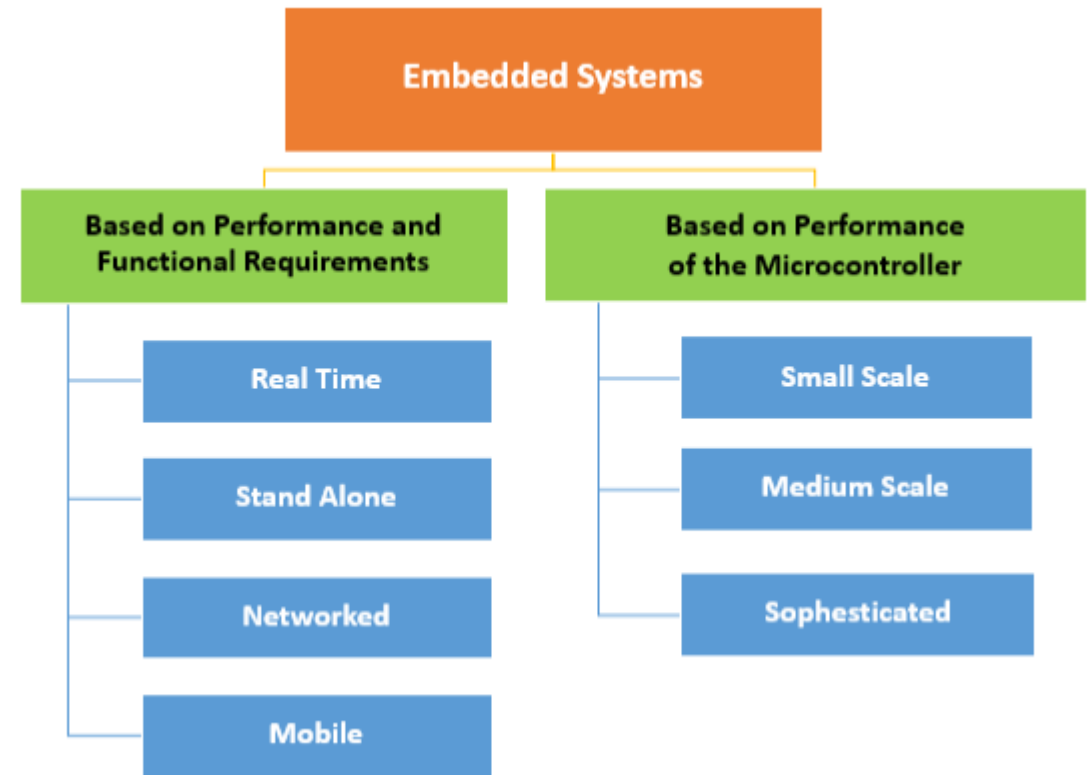
Embedded systems can be classified into different types based on performance, functional requirements and performance of the microcontroller.

Based on functionality and performance requirements, embedded systems are classified as :

- Stand-alone Embedded Systems
- Real-time Embedded Systems
- Networked Information Appliances
- Mobile Devices

Embedded Systems are classified into three types based on the performance of the microcontroller such as

- Small scale embedded systems
- Medium scale embedded systems
- Sophisticated embedded systems



Classification of Embedded Systems

Based on functionality

Stand Alone Embedded Systems

- Stand alone embedded systems do not require a host system like a computer, it works by itself. It takes the input from the input ports either analog or digital and processes, calculates and converts the data and gives the resulting data through the connected device-Which either controls, drives and displays the connected devices.
- Examples for the stand alone embedded systems are mp3 players, digital cameras, video game consoles, microwave ovens and temperature measurement systems.

Real Time Embedded Systems

- A real time embedded system is defined as, a system which gives a required o/p in a particular time. These types of embedded systems follow the time deadlines for completion of a task.
- Real time embedded systems are classified into two types such as soft and hard real time systems.

Classification of Embedded Systems

Based on functionality

Networked Embedded Systems

- These types of embedded systems are related to a network to access the resources. The connected network can be LAN, WAN or the internet. The connection can be any wired or wireless. This type of embedded system is the fastest growing area in embedded system applications. The embedded web server is a type of system wherein all embedded devices are connected to a web server and accessed and controlled by a web browser.
- Example for the LAN networked embedded system is a home security system wherein all sensors are connected and run on the protocol TCP/IP

• Mobile Embedded Systems

Mobile embedded systems are used in portable embedded devices like cell phones, mobiles, digital cameras, mp3 players and personal digital assistants, etc. The basic limitation of these devices is the other resources and limitation of memory.

Classification of Embedded Systems

Based on the performance

- **Small Scale Embedded Systems**

- These types of embedded systems are designed with a single 8 or 16-bit microcontroller, that may even be activated by a battery.
- Little hardware and software complexity.
- Usually “C” is used for developing these system.
- The need to limit power dissipation when system is running continuously.
- Programming tools: For developing embedded software for small scale embedded systems, the main programming tools are an editor, assembler, cross assembler and integrated development environment (IDE).



Classification of Embedded Systems

Based on the performance

- **Medium Scale Embedded Systems**

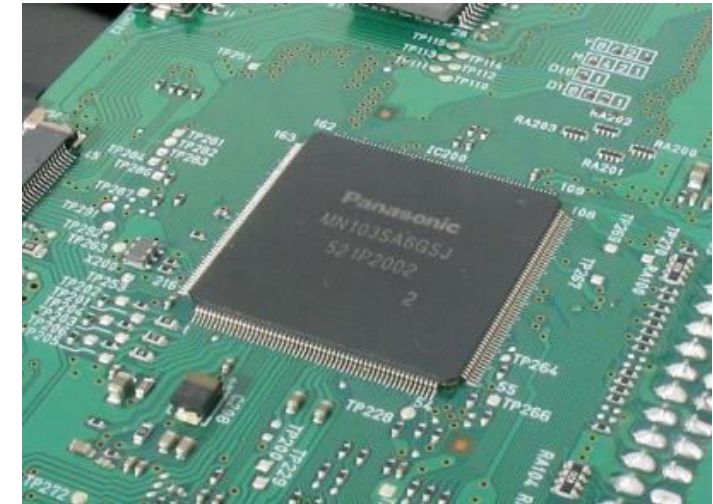
- These types of embedded systems design with a single or 16 or 32 bit microcontroller, (**Reduced Instructions Set Computers**)RISCs or DSPs.
- These types of embedded systems have both hardware and software complexities.
- **Programming tools:**For developing embedded software for medium scale embedded systems, the main programming tools are C, C++, JAVA, Visual C++, RTOS, debugger, source code engineering tool, simulator and IDE.



Classification of Embedded Systems

Based on the performance

- **Sophisticated Embedded Systems**
 - These types of embedded systems have enormous hardware and software complexities, that may need ASIPs, IPs, PLAs, scalable or configurable processors.
 - They are used for cutting-edge applications that need hardware and software Co-design and components which have to assemble in the final system.
- Constrained by the processing speed available in their hardware units.
- **Programming Tools:** For these systems may not be readily available at a reasonable cost or may not be available at all. A compiler or retargetable compiler might have to be developed for this.



Purpose of an Embedded System

Each Embedded system is designed to serve the purpose of any one or a combination of the following tasks.

1. Data Collection/ Storage/Representation
2. Data Communication
3. Data (Signal) Processing
4. Monitoring
5. Control
6. Application Specific user interface

Purpose of an Embedded System

1. Data Collection/ Storage/Representation

1. Data collection is usually done for storage, analysis, manipulation and transmission.
2. The term “Data” refers all kinds of information, viz, text, voice, Image, electrical signals and other measurable quantities.
3. Data can be either analog (continues) or Digital (discrete).
4. Embedded system with analog data capturing techniques collect data directly in the form of analog and converts the analog signal to digital signal by using A/D converters and then collect the binary equivalent of the analog data.
5. If the signal is digital it can be directly captured without any additional interface by digital embedded system.
6. The collected data may be stored directly in the system or may be transmitted to other systems or it may be progressed by the system or it may be deleted instantly after giving a meaningful representation.

Purpose of an Embedded System

- The digital camera is a typical example of an embedded system with data collection/storage/representation of data.
- Images are captured and the captured image may be stored with in the memory of the camera. The captured image can also be presented to the user through a LCD display unit.



Purpose of an Embedded System

2. Data Communication

- Embedded data communication systems are developed in applications ranging from complex satellite communication systems to simple home networking systems.

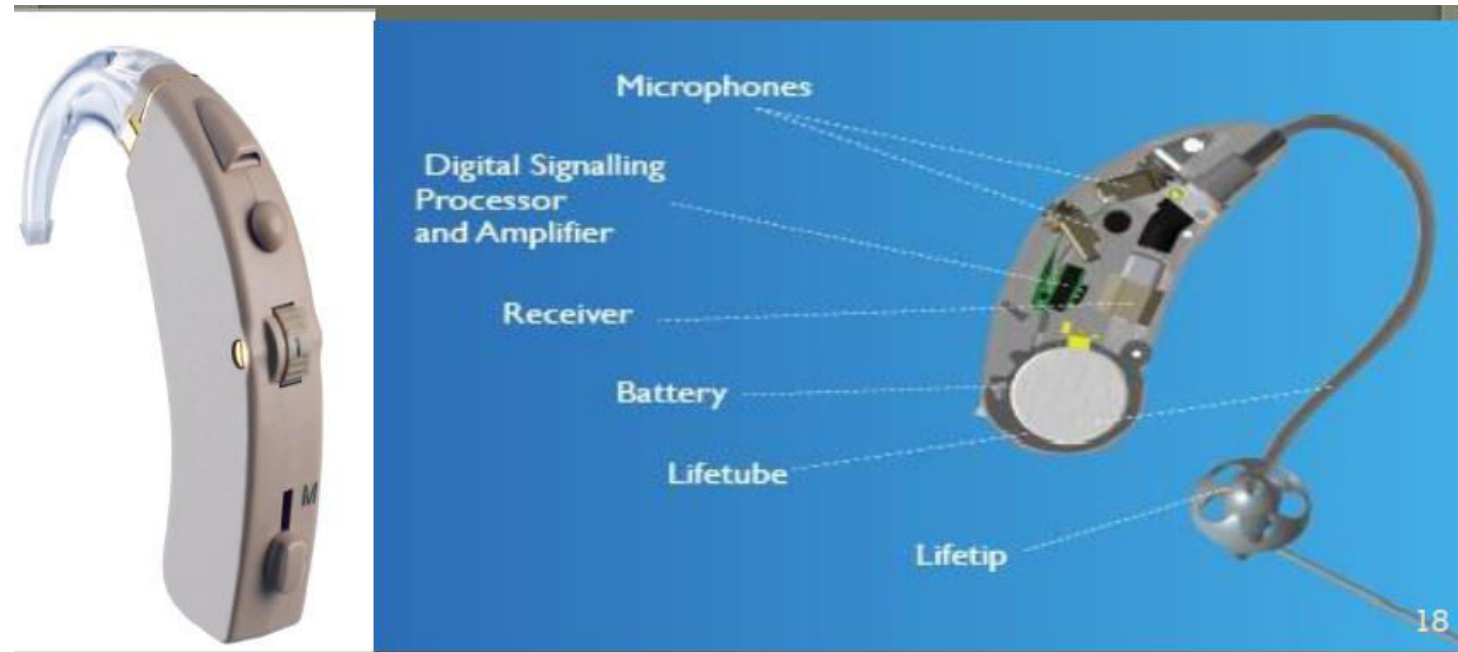


Figure: - A wireless network router for data communication

Purpose of an Embedded System

3. Data Processing

- The data collected by embedded system may be used for various kinds of signal processing.
- A digital hearing aid is a typical example of an embedded system employing data processing.



Purpose of an Embedded System

4. Monitoring

- All embedded products coming under the medical domain are with monitoring functions only. They are used for determining the state of some variables using input sensors.
- A very good example is the electro cardiogram (ECG) machine for monitoring the heartbeat of patient.



Figure:- A patient monitoring system for monitoring for heartbeat

Purpose of an Embedded System

5. Control

- Embedded system with control functionalities impose control over some variables according to the input variables.
- A system with control functionality contains both sensors and actuators.
- Sensors are inputs ports for capturing the changes in environment variables or measuring variables.
- Actuators are output ports are controlled according to the changes in input variable.



Purpose of an Embedded System

6. Application Specific user interface

- ❑ These are embedded systems with application specific user interfaces like buttons, switches, keypad, lights, bells, display units, etc.
- ❑ Mobile phone is an example for this, in mobile phone the user interface is provided through the keyboard, graphic LCD module, system speaker, vibration alert, etc....



Applications of embedded system

Some of the major Application Areas of Embedded System

1. Consumer Electronics

- ❖ Camcorders, Cameras, etc...

2. Household Appliances

- ❖ Television, DVD Player, Washing machine, fridge, microwave oven, etc.

3. Home automation and security system

- ❖ Air conditioners, Sprinkler, intruder detection alarms, fire alarms, closed circuit television cameras, etc

4. Automotive industry

- ❖ Anti-lock breaking system (ABS), engine control, ignition control, automatic navigation system, etc..

5. Telecommunication

- ❖ Cellular telephones, telephone switches, Router, etc...

Applications of embedded system

6. Computer peripherals

- ❖ Printers, scanners, fax machines, etc...

7. Computer Networking systems

- ❖ Network routers, switches, hubs, firewalls, etc...

8. Health care

- ❖ CT scanner, ECG, EEG, EMG, MRI, Glucose monitor, blood pressure monitor, medical diagnostic device, etc.

9. Measurement & Instrumentation

- ❖ Digital multi meters, digital CROs, logic analyzers PLC systems, etc...

10. Banking & Retail

- ❖ Automatic Teller Machine (ATM) and Currency counters, smart vendor machine, cash register, Share market, etc..

11. Card Readers

- ❖ Barcode, smart card readers, hand held devices, etc...

Applications of embedded system

Embedded systems are used in different applications like automobiles, telecommunications, smart cards, missiles, satellites, computer networking and digital consumer electronics.



Applications of embedded system

- **Embedded Systems in Automobiles and in telecommunications**
- Motor and cruise control system
- Body or Engine safety
- Entertainment and multimedia in car
- E-Com and Mobile access
- Robotics in assembly line
- Wireless communication
- Mobile computing and networking
- **Embedded Systems in Smart Cards, Missiles and Satellites**
- Security systems
- Telephone and banking
- Defense and aerospace
- Communication

Applications of embedded system

- **Embedded Systems in Peripherals & Computer Networking**
- Displays and Monitors
- Networking Systems
- Image Processing
- Network cards and printers
- **Embedded Systems in Consumer Electronics**
- Digital Cameras
- Set top Boxes
- High Definition TVs
- DVDs

Features and Attributes of Embedded System

Features of an Embedded system

- Embedded systems do a very specific task, they cannot be programmed to do different things.
- Embedded systems have very limited resources, particularly the memory. Generally, they do not have secondary storage devices such as the CDROM or the floppy disk.
- Embedded systems have to work against some deadlines. A specific job has to be completed within a specific time. In some embedded systems, called real-time systems, the deadlines are stringent. Missing a dead line may cause a catastrophe – loss of life or damage to property.
- Embedded systems are constrained for power, As many embedded systems operate through a battery, the power consumption has to be very low.
- Embedded systems need to be highly reliable. Once in a while, pressing ALT-CTRL-DEL is OK on your desktop, but you cannot afford to reset your embedded system.
- Some embedded systems have to operate in extreme environmental conditions such as very high temperatures and humidity.
- Embedded systems that address the consumer market (for example electronic toys) are very cost-effective. Even a reduction of Rs.10 is lot of cost saving, because thousands or millions systems may be sold.
- Unlike desktop computers in which the hardware platform is dominated by Intel and the operating system is dominated by Microsoft, there is a wide variety of processors and operating systems for the embedded systems. So, choosing the right platform is the most complex task .

Attributes of Embedded System

Quality Attributes of Embedded Systems

- Quality attributes are the non-functional requirements that need to be documented properly in any system design.
- If the quality attributes are more concrete and measurable, it will give a positive impact on the system development process and the end product.
- The various quality attributes that needs to be addressed in any embedded system development are broadly classified into two, namely
 - i. Operational Quality Attributes
 - ii. Non-Operational Quality Attributes

Quality Attributes of Embedded Systems

- **Operational Quality Attributes**

- The operational quality attributes represent the relevant quality attributes related to the embedded system when it is in the operational mode or 'online' mode.
- The important quality attributes coming under this category are listed below:
 - i. Response
 - ii. Throughput
 - iii. Reliability
 - iv. Maintainability
 - v. Security
 - vi. Safety

Operational Quality Attributes

i. Response

- Response is a measure of quickness of the system.
- It gives you an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.
- Ex. An embedded system deployed in flight control application should respond in a Real Time manner.
- Any response delay in the system will create potential damages to the safety of the flight as well as the passengers.
- It is not necessary that all embedded systems should be Real Time in response.
- For example, the response time requirement for an electronic toy is not at all time-critical.

Operational Quality Attributes

ii. Throughput

- Throughput deals with the efficiency of system.
- It can be defined as rate of production or process of a defined process over a stated period of time.
- The rates can be expressed in terms of units of products, batches produced, or any other meaningful measurements.
- In case of card reader like the ones used in buses, throughput means how much transactions the Reader can perform in a minute or hour or day.
- Throughput is generally measured in terms of 'Benchmark'. A 'Benchmark' is a reference point by which something can be measured.
- Benchmark can be a set of performance criteria that a product is expected to meet or a standard product that can be used for comparing other products of the same product line.

Operational Quality Attributes

iii. Reliability

- Reliability is a measure of how much percentage you rely upon the proper functioning of the system or what is the % susceptibility of the system to failure.
- Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) are the terms used in defining system reliability.
- MTBF gives the frequency of failures in hours/weeks/months.
- MTTR specifies how long the system is allowed to be out of order following a failure.
- For an embedded system with critical application need, it should be of the order of minutes.

Operational Quality Attributes

iv. Maintainability

- Maintainability deals with support and maintenance to the end user or client in case of technical issues and product failures or on the basis of a routine system checkup.
- Reliability and maintainability are considered as two complementary disciplines. A more reliable system means a system with less corrective maintainability requirements and vice versa.
- Maintainability can be classified into two types:
 - 1. Scheduled or Periodic Maintenance (Preventive Maintenance)
An inkjet printer uses ink cartridges, which are consumable components and as per the printer manufacturer the end use should replace the cartridge after each 'n' number of printouts to get quality prints.
 - 2. Maintenance to Unexpected Failures (Corrective Maintenance)
If the paper feeding part of the printer fails the printer fails to print and it requires immediate repairs to rectify this problem. Hence it is obvious that maintainability is simply an indication of the availability of the product for use.
In any embedded system design, the ideal value for availability is expressed as
$$A_i = \frac{MTBF}{(MTBF + MTTR)}$$
- Where A_i =Availability in the ideal condition, MTBF=Mean Time Between Failures, and MTTR= Mean Time To Repair

Operational Quality Attributes

- v. Security
- ‘Confidentially’, ‘Integrity’, and ‘Availability’ are three major measures of information security.
- ‘Confidentially’ deals with the protection of data and application from unauthorized disclosure.
- ‘Integrity’ deals with the protection of data and application from unauthorized modification.
- ‘Availability’ deals with protection of data and application from unauthorized users.
- Certain embedded systems have to make sure they conform to the security measures.
- Ex. An electronic safety Deposit Locker can be used only with a pin number like a password.

Operational Quality Attributes

- vi. Safety
- Safety deals with the possible damages that can happen to the operators, public and the environment due to the breakdown of an embedded system or due to the emission of radioactive or hazardous materials from the embedded products.
- The breakdown of an embedded system may occur due to a hardware failure or a firmware failure.
- Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of the damages to an acceptable level.

Non Operational Attributes

- Non Operational Attributes
- The quality attributes that needs to be addressed for the product 'not' on the basic of operational aspects are grouped under this category.
- The important quality attributes coming under this category are listed below:
 - i. Testability & Debug-ability
 - ii. Evolvability
 - iii. Portability
 - iv. Time to prototype and market
 - v. Per unit and total cost

Non Operational Attributes

i. Testability & Debug-ability

- Testability deals with how easily one can test his/her design, application and by which means he/she can test it.
- For an embedded product, testability is applicable to both the embedded hardware and firmware.
- Debug-ability is a means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system.
- Debug-ability has two aspects in the embedded system development context, namely, hardware level debugging and firmware level debugging.
- Hardware debugging is used for figuring out the issues created by hardware problems whereas firmware debugging is employed to figure out the probable errors that appear as a result of flaws in the firmware.

Non Operational Attributes

ii. **Evolvability**

- Evolvability is a term which is closely related to Biology.
- Evolvability is referred as the non-heritable variation.
- For an embedded system, the quality attribute 'Evolvability' refers to the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.

iii. **Portability**

- Portability is a measure of 'system independence'.
- An embedded product can be called portable if it is capable of functioning in various environments, target processors/controllers and embedded operating systems.
- A standard embedded product should always be flexible and portable.

Non Operational Attributes

iv. **Time-to-Prototype and Market**

- Time-to-market is the time elapsed between the conceptualization of a product and the time at which the product is ready for selling (for commercial product) or use (for non-commercial products).
- The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product.
- Product prototyping helps a lot in reducing time-to-market.

Non Operational Attributes

v. Per Unit Cost and Revenue

- Cost is a factor which is closely monitored by both end user (those who buy the product) and product manufacturer (those who build the product).
- Cost is a highly sensitive factor for commercial products.
- Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.
- When the product is introduced in the market, for the initial period the sales and revenue will be low.
- There won't be much competition when the product sales and revenue increase.
- During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.

Advantages of Embedded System

The advantages of Embedded Systems are:

- They are convenient for mass production. This results in low price per piece.
- These systems are highly stable and reliable.
- Embedded systems are made for specific tasks.
- The embedded systems are very small in size, hence can be carried and loaded anywhere.
- These systems are fast. They also use less power.
- The embedded systems optimize the use of resources available.
- They improve the product quality.
- Easily Customizable
- Low power consumption
- Low cost
- Enhanced performance

Disadvantages of Embedded System

The disadvantages of Embedded Systems are as follows:

- Once configured, these systems cannot be changed. Hence, no improvement or upgradation on the ones designed and created can be made.
- They are hard to maintain. It is also difficult to take a back-up of embedded files.
- Troubleshooting is difficult for embedded systems. Transferring data from one system to another is also quite problematic.
- Because these systems are made for specific tasks, hardware is limited.
- High development effort
- Larger time to market

Challenges in Embedded System

- Challenges of Embedded Software Development
- Embedded software is always a constituent of a larger system, for instance, a digital watch, a smartphone, a vehicle or automated industrial equipment. Such embedded systems must have real-time response under all circumstances within the time specified by design and operate under the condition of limited memory, processing power and energy supply. Moreover, embedded software must be immune to changes in its operating environment – processors, sensors, and hardware components may change over time. Other challenging requirements to embedded software are portability and autonomy.
- **Challenge #1: Stability**
- Stability is of paramount importance. Unexpected behavior from an embedded system is inadmissible and poses serious risks. End users demand that embedded systems must have uniform behavior under all circumstances and be able to operate durably without service.
- **Challenge #2: Safety**
- Safety is a special feature of embedded systems due to their primary application associated with lifesaving functionality in critical environments. Software Development Life Cycle (SDLC) for embedded software is characterized by more strict requirements and limitations in terms of quality, testing, and engineering expertise.

Challenges in Embedded System

- **Challenge #3: Security**

- [Security](#) became a burning issue in the digital world. The related risks grow exponentially, especially so for IoT devices gaining popularity worldwide and becoming more interconnected to each other. Because modern home appliances like electric cookers, refrigerators and washing machines have connectivity feature integrated by default, Internet of Things now is exposed to a serious risk of hacking attacks.

- **Challenge #4: Launch Phase**

- Time-to-market and time-to-revenue have always been tough indicators in [embedded system development](#), especially in the IoT segment. That is why the apps and platforms supposed to support zillions of IoT devices expected to appear by 2020 still are in their concept stage. Fabrication of hardware components housing embedded systems require extreme integration and flexibility due to very fast development of IoT industry. In addition, taking into account longer IoT device lifespan, future updates and releases become an issue for component designers.

Challenges in Embedded System

- **Challenge #5: Design Limitations**

- The challenges in design of embedded systems have always been in the same limiting requirements for decades:
- Small form factor;
- Low energy;
- Long-term stable performance without maintenance.
- The market demands from designers to pack more processing power and longer battery life into smaller spaces, which is often a tradeoff. Finally, depending on applications in IoT, there is a growing demand for manufacture of very scalable processor families ranging from cheap and ultra-low-power to maximum performance and highly configurable processors with forward-compatible instruction set. There is similar demand for increased performance of system buses and internal/external memory caches.

- **Challenge #6: Compatibility and Integrity**

- [Gartner Group estimation](#) shows that, presently, most of the apps in the market are launched by businesses younger than 3 years old. With all their probable expertise in software development, many of them lack hands-on experience in implementing and updating their applications in IoT environment, especially with regard to security implications.
- Further expansion of IoT devices on the background of their connectivity puts more pressure on their adaptability. Users must be capable of administering the app through a simple user interface via all available channels including [over-the-air firmware updates](#), which needs extreme compatibility across the entire ecosystem.
- Integrity becomes a function of security. To protect the IoT from malicious attacks or compromising, security must be implemented within each device at every level: the end node, gateway, cloud, etc.

Recent trends in embedded system

Artificial Intelligence

- The market growth for Artificial Intelligence (AI) technologies is unstoppable. We are in a virtual race to reach the end-line of this track. There is a huge upsurge in AI- embedded development with exuberant investments and wide-scale adoption by market giants and numerous startups. As per an estimate, [investment in artificial intelligence](#) was set to grow to a stratospheric 300% in 2017. Set to grow with a CAGR of 55.1% over 2016-2020, AI and cognitive systems will drive more than \$47B investments by 2020 says an IDC report.
- Hot-selling AI trends in action are: virtual assistants, natural language generation, speech recognition, biometrics, AI-optimized hardware and robotic automation. Like others, this road has its own share of speed breakers. For technology adopters, there is ambiguity about what virtual reality and artificial intelligence can be most used for, modernized data management platform, and lack of necessary skills and talent. One Solution- you can overcome with a highly experienced and war-tested partner to make a stomping difference.

Recent trends in embedded system

Wearable Electronics

- The bubble is about to burst. The hype around wearable technology has peaked. With an annual growth [rate of 18%](#) (at least till 2021), market players don't leverage its optimum value- as Wearable technology is yet to break boundaries of being a fad.
- Pebble, FitBit, or the recent Apple Watch- this industry needs to be revolutionized with superior battery performance, enhanced functionalities and diversified offerings.

Recent trends in embedded system

Retail

- Embedded application development is on haute couture of online and offline retail industry. Consumers and business owners will be hugely invested in embedded technology, especially IoT.
- Allowing laser-focused opportunities to reach a wider market, market leading companies will eagerly use 'tons' of data for sensor-based analytics, inventory management, manage losses from theft, and reach out customers in different ways.
- Move beyond drone-delivery systems and [magic mirrors](#). The new wave in embedded-sensor driven retail shopping will drastically change the way- consumers shop.

Recent trends in embedded system

Healthcare

- The era of tiny and smart wearables has just begun- Healthcare is on path to become patient-centric. There is a mad rush of patients to Big Data driven healthcare systems- and it potentially changes the way people interact, access and pay for services.
- Wearable gadgets are a common buy now as it brings immense abilities to monitor, track and alert users about various health-related symptoms.
- To keep buoyancy of the industry, hospitals and clinics are making the quickest sprint to embedded technology-based innovations like sensory tattoos, surgeon micro bots, medical tablets and bedside terminals.

Recent trends in embedded system

Internet of Things

- Intelligent Internet of Things is like The God of Small Things in embedded applications development, it reigns supreme. As per a mammoth prediction by Gartner, about 8.4 Billion connected “Things” will be in use by the end of this year. Though IoT-specific hardware drives many applications, the success is based on its software.
- In 2018, we expect tectonically shift in functionality, safety and security of IoT-enabled devices. Categorized into three wide domains: robots, drones and smart vehicles, IoT still has a lot of unexplored room and skills to capture newer markets with IoT exhibiting intelligence in homes, offices, factories and healthcare units.
- Make embedded application systems a driver for your business growth. A versatile partner with diverse domain expertise, superior technical proficiency, and cost effective engineering will save the day.

Recent trends in embedded system

Building Automation

- Automation systems for smart buildings and HVAC utilize embedded software and hardware and the sector should develop rapidly in the coming years.
- As we move into the era of smart buildings and smart cities, embedded intelligence will be an integral component of these smart systems. Building automation has been based primarily on monitoring and maintaining environmental conditions, lighting, and access control.
- As the systems become smarter, smart building functionality will likely extend into predictive and prescriptive systems that determine the optimal conditions. Ultimately, the goal is to move to completely autonomous and self-healing systems.
- These systems will be based on AI and machine learning, all predicated on embedded intelligence. The smart building industry has embraced the concept of the digital twin, using intelligent sensors to merge physical operations with virtual engineering models.

Recent trends in embedded system

Embedded Systems in the Industries

- Embedded systems have been a staple technology in industries like aerospace & defense, automotive, medical devices, communication, and industrial automation for decades.
- As processor architecture evolved and more computing power could be embedded in systems and devices, the intelligence and capabilities of these systems increased exponentially.
- This has allowed the products that have traditionally used embedded systems to become more intelligent and robust, and enabled products in other industries (consumer goods, appliances, sporting goods, etc.) to become smart and connected. Embedded systems are becoming an integral component of almost everything in our lives.

Recent trends in embedded system

- With the gadgets on the planet becoming more “connected and smart”, it is the instinctual for companies to identify matured tech partners to master this hardware groomed software engineered embedded world.
- Make embedded development a driver for your business growth. A versatile partner with diverse domain expertise, superior technical proficiency, and cost effective engineering will save the day.

Recent trends in embedded system

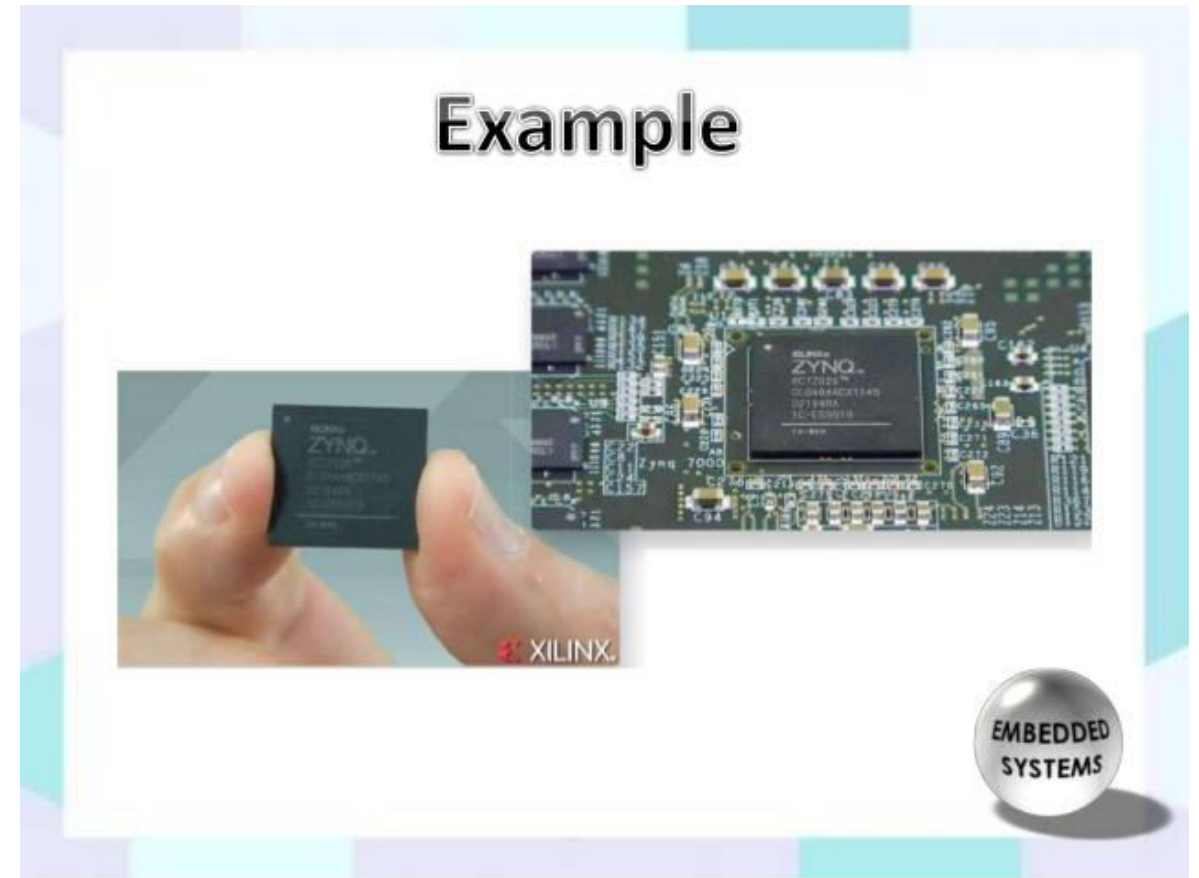
Other Various recent trends in embedded system are :-

- 1. SoC (System on a Chip)
- 2. Wireless Technology
- 3. Multi-core Processor
- 4. Multi-language support
- 5. User interface
- 6. Use of open Source Technology
- 7. Inter-operatability
- 8. Automation
- 9. Security
- 10. Power consumption

Recent trends in embedded system

1. SoC (System on a Chip)

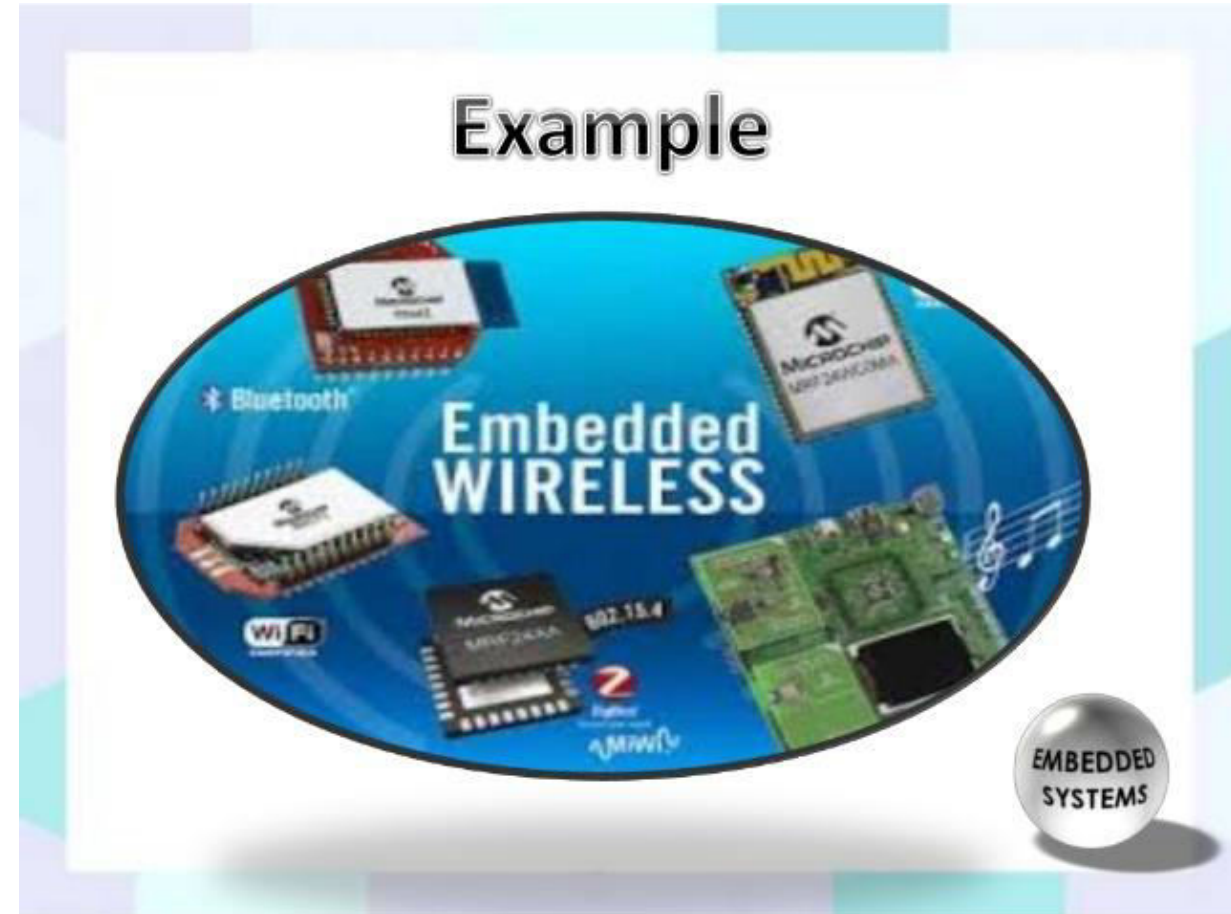
- A system on a chip (SoC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip.
- SoCs are very common in the mobile electronics market because of their low power consumption. A typical application is in the area of embedded systems.



Recent trends in embedded system

2. Wireless Technology

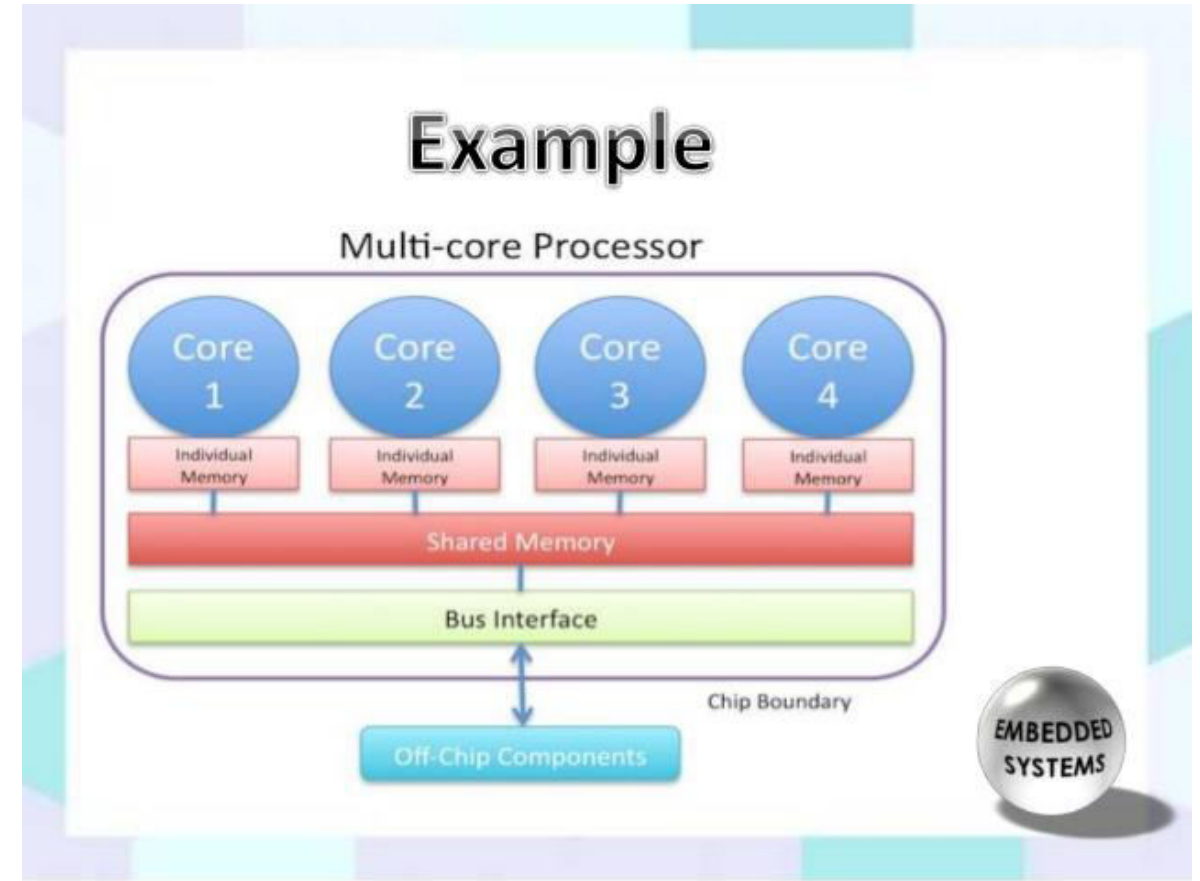
- The term "wireless" refers, in the most basic and obvious sense, to communications sent without wires or cables.
- It is a broad term that encompasses all sorts of wireless technologies and devices, including cellular communications, networking between computers with wireless adapters, and wireless computer accessories.
- Wireless communications travel over the air via electromagnetic waves (radio frequencies, infrared, satellite, etc).



Recent trends in embedded system

3. Multi-core Processor

- A multi-core processor is an integrated circuit (IC) to which two or more processors have been attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks.



Recent trends in embedded system

4. Multi-language support

- Embedded system provides us facility of multiple languages. For example ATM is a typical example of embedded system.
- In ATM there are three types of languages are used like Hindi, Punjabi, English.



Recent trends in embedded system

5. User interface

- Visual part of computer application or operating system through which a user interacts with a computer or a software.
- It determines how commands are given to the computer or the program and how information is displayed on the screen.



Recent trends in embedded system

6. Use of open Source Technology

- Open source technology refers to a program in which the source code is available to the general public for use and/or modification from its original design free of charge, i.e., open.
- Open source code is typically created as a collaborative effort in which programmers improve upon the code and share the changes within the community.



Recent trends in embedded system

7. Inter-operability

- Ability to work with each other. In the loosely coupled environment of a service-oriented architecture, separate resources don't need to know the details of how they each work, but they need to have enough common ground to reliably exchange messages without error or misunderstanding.
- Interoperability is when services can interact with each other without encountering such problems.



Recent trends in embedded system

8.Automation

- Automation or automatic control, is the use of various control systems for operating equipment such as machinery, processes in factories, switching in telephone networks, steering and stabilization of ships, aircraft and other applications with minimal or reduced human intervention. Some processes have been completely automated.
- The biggest benefit of automation is that it saves labor, however, it is also used to save energy and materials and to improve quality, accuracy and precision.

Example

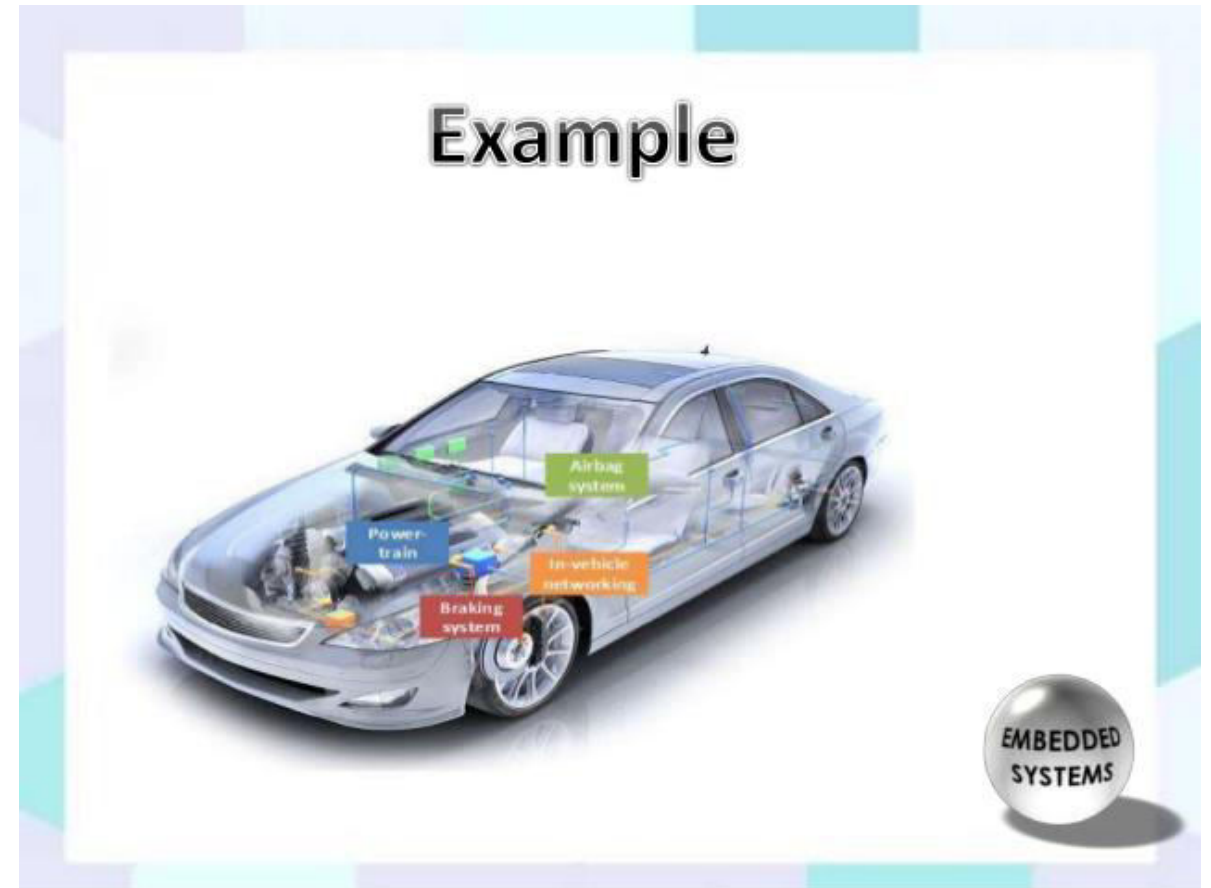


EMBEDDED
SYSTEMS

Recent trends in embedded system

9.Security

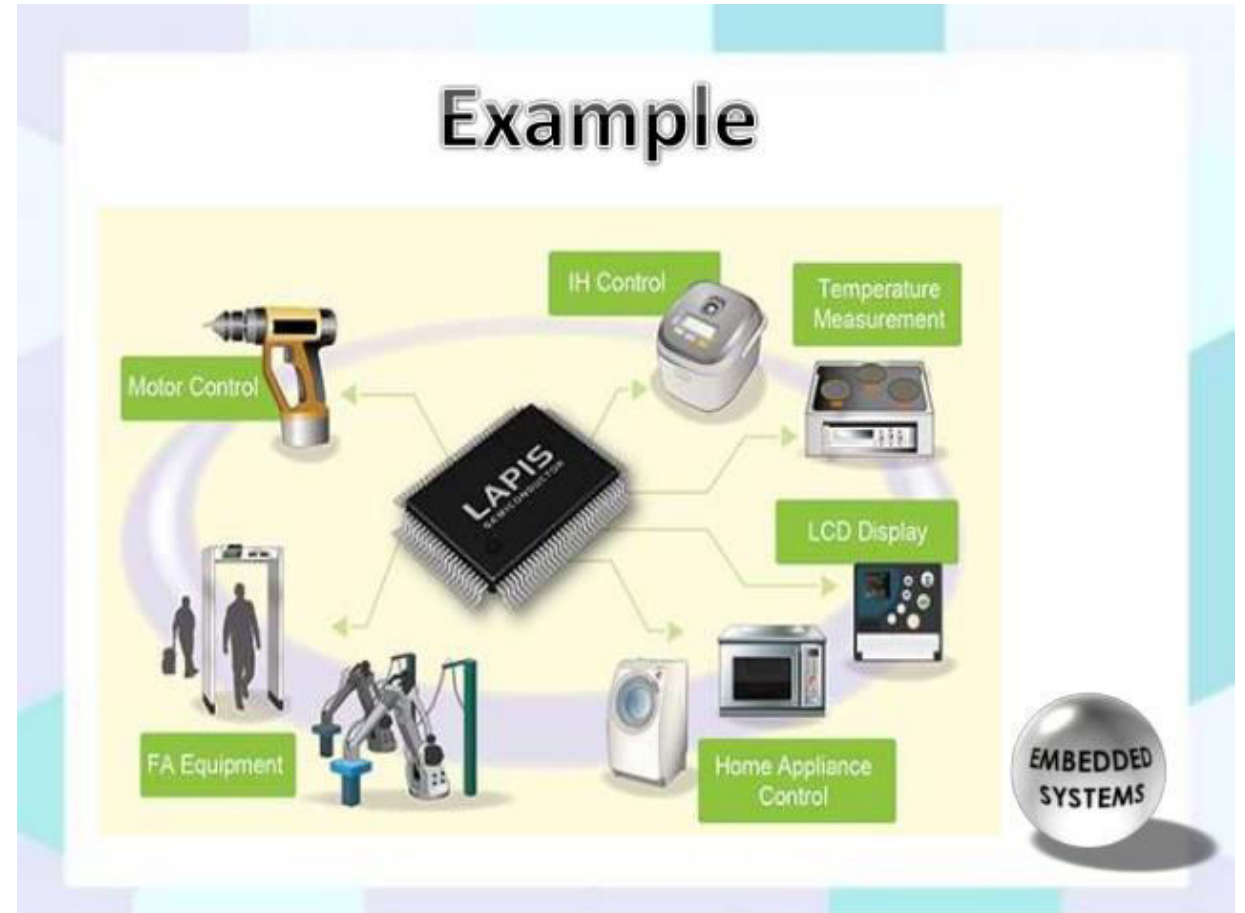
- Secure Design Capture system security and performance requirements and develop a secure architecture
- Threat Model Analysis Ensure your system is analyzed without bias and based on expert knowledge of the attack landscape
- Software & Code Testing Uncover serious problems early and avoid crash-causing defects in code
- Crypto Implementation Leverage our expertise to ensure crypto is implemented securely



Recent trends in embedded system

10.Power consumption

- Power consumption has traditionally been something influenced only by hardware developers. But power consumption depends not only on the hardware, but also on how it is used and how it is controlled by the system software.
- Through power consumption – it becomes possible to test and tune for power optimisation.



Selection of Processors

- Different systems require different processor features
- The processor is selected from the following considerations –
 - Instruction Set
 - Maximum bits in an operand in an operation
 - Processing Speed
 - Ability to solve the complex algorithms
 - A processor gives high computing performance when it has
 - Pipeline and Superscalar architecture
 - Pre-fetch cache unit, caches, register files and MMU
 - RISC core architecture
- A processor with register windows provides fast context switching in a multitasking system
- Processor has auto shut down features for its units
- A processor with burst mode accesses external memories fast

Selection of Processors

- With numerous kinds of processors with various design philosophies available at our disposal for using in our design, following considerations need to be factored during processor selection for an embedded system.
 - Performance Considerations
 - Power considerations
 - Peripheral Set
 - Operating Voltage
 - Specialized Processing Units
 - Memory
 - Cost

Selection of Processors

Performance considerations

- The first and foremost consideration in selecting the processor is its performance. The performance speed of a processor is dependent primarily on its architecture and its silicon design. Evolution of fabrication techniques helped packing more transistors in same area there by reducing the propagation delay. Also presence of cache reduces instruction/data fetch timing. Pipelining and super-scalar architectures further improves the performance of the processor. Branch prediction, speculative execution etc are some other techniques used for improving the execution rate. Multi-cores are the new direction in improving the performance.
- Rather than simply stating the clock frequency of the processor which has limited significance to its processing power, it makes more sense to describe the capability in a standard notation. MIPS (Million Instructions Per Second) or MIPS/MHz was an earlier notation followed by Dhrystones and latest EEMBC's **CoreMark**. CoreMark is one of the best ways to compare the performance of various processors.
- Processor architectures with support for extra instruction can help improving performance for specific applications. For example, SIMD (Single Instruction/Multiple Data) set and Jazelle – Java acceleration can help in improving multimedia and JVM execution speeds.
- So size of cache, processor architecture, instruction set etc has to be taken in to account when comparing the performance.

Selection of Processors

Power considerations

- Increasing the logic density and clock speed has adverse impact on power requirement of the processor. A higher clock implies faster charge and discharge cycles leading to more power consumption. More logic leads to higher power density there by making the heat dissipation difficult. Further with more emphasis on greener technologies and many systems becoming battery operated, it is important the design is for optimal power usage.
- Techniques like ***frequency scaling*** – reducing the clock frequency of the processor depending on the load, ***voltage scaling*** – varying the voltage based on load can help in achieving lower power usage. Further asymmetric multiprocessors, under near idle conditions, can effectively power off the more powerful core and load the less powerful core for performing the tasks. SoC comes with advanced power gating techniques that can shut down clocks and power to unused modules.

Selection of Processors

Peripheral Set

- Every system design needs, apart from the processor, many other peripherals for input and output operations. Since in an embedded system, almost all the processors used are SoCs, it is better if the necessary peripherals are available in the chip itself.
- This offers various benefits compared to peripherals in external IC's such as optimal power architecture, effective data communication using DMA, lower BoM etc. So it is important to have peripheral set in consideration when selecting the processor.

Selection of Processors

Operating Voltage

- Each and every processor will have its own operating voltage condition. The operating voltage maximum and minimum ratings will be provided in the respective data sheet or user manual.
- While higher end processors typically operate with 2 to 5 voltages including 1.8V for Cores/Analogue domains, 3.3V for IO lines, needs specialized PMIC devices, it is a deciding factor in low end micro-controllers based on the input voltage. For example it is cheaper to work with a 5V micro-controller when the input supply is 5V and a 3.3 micro-controllers when operated with Li-on batteries.

Selection of Processors

Specialized Processing Units

- Apart from the core, presence of various co-processors and specialized processing units can help achieving necessary processing performance. Co-processors execute the instructions fetched by the primary processor thereby reducing the load on the primary. Some of the popular co-processors include
 - ***Floating Point Co-processor:***
 - RISC cores supports primarily integer only instruction set. Hence presence of a FP co-processor can be very helpful in application involving complex mathematical operations including multimedia, imaging, codecs, signal processing etc.
 - ***Graphic Processing Unit:***
 - GPU(Graphic Processing Unit) also called as Visual processing unit is responsible for drawing images on the frame buffer memory to be displayed. Since human visual perception needed at-least 16 Frames per second for a smooth viewing, drawing for HD displays involves a lot of data bandwidth. Also with increasing graphic requirements such as textures, lighting shaders etc, GPU's have become a mandatory requirements for mobile phones, gaming consoles etc.
 - Various GPU's like ARM's MALI, PowerVX, OpenGL etc are increasing available in higher end processors. Choosing the right co-processor can enable smooth design of the embedded application.

Selection of Processors

Memory

- You will have to go for a processor that is able to support the entirety of your software program on its onboard memory. Remember: most microcontrollers have a limited amount of onboard address space, which is why you need to actively search for a solution that can accommodate your program.

Selection of Processors

Digital Signal Processors

- DSP is a processor designed specifically for signal processing applications. Its architecture supports processing of multiple data in parallel. It can manipulate real time signal and convert to other domains for processing. DSP's are either available as the part of the SoC or separate in an external package. DSP's are very helpful in multimedia applications. It is possible to use a DSP along with a processor or use the DSP as the main processor itself.

Price/Cost

- Various considerations discussed above can be taken in to account when a processor is being selected for an embedded design. It is better to have some extra buffer in processing capacities to enable enhancements in functionality without going for a major change in the design. While engineers (especially software/firmware engineers) will want to have all the functionalities, price will be the determining factor when designing the system and choosing the right processor.

Embedded Firmware design approaches and development languages

Introduction

- Embedded firmware is responsible for controlling various peripherals of the embedded hardware and generating responses in accordance with the functional requirements mentioned in the requirements for the particular product
- Firmware is considered as the master brain of the embedded systems
- Imparting intelligence to an embedded system is a one time process and it can happen at any stage of the design
- Once the intelligence is imparted to the embedded product, by embedding the firmware in the hardware, the product starts functioning properly and will continue serving the assigned task till hardware breakdown occurs or a corruption in embedded firmware occurs
- Designing an embedded firmware requires understanding of embedded product hardware like, various component interfacing, memory map details I/O port details, configuration and register details of various hardware chips used and some programming language.
- Embedded firmware development process starts with conversion of firmware requirements into a program model using modeling tools like UML or flow chart based representation
- UML diagram gives diagrammatic representation of the decision items to be taken and the task to be performed
- Once the program modeling is created, next step is the implementation of the task and actions by capturing the model using a language which is understandable by the target processor

Embedded Firmware design approaches

- Following gives an overview of the various steps involved in the embedded firmware design and development
 - Firmware design approaches depends on the
 - Complexity of the function to be performed
 - Speed of operation required .. –Etc
- Two basic approaches for firmware design
 - 1. Conventional Procedure based Firmware Design/Super Loop Design
 - 2. Embedded Operating System Based Design

Embedded Firmware design approaches

SUPER LOOP BASED APPROACH

- This approach is applied for the applications that are not time critical and the response time is not so important
- Similar to the conventional procedural programming where the code is executed task by task
- Task listed at the top of the program code is executed first and task below the first task are executed after completing the first task
- It is True procedural one
- In multiple task based systems, each task executed in serial

Embedded Firmware design approaches

SUPER LOOP BASED APPROACH

- Firmware execution flow of this will be as following
- 1.Configure the common parameter and perform initialization for various hardware components, memory, registers etc.
- 2.Start the first task and execute it
- 3.Execute the second task
- 4.Execute the next task
- 5.....
- 6.....
- 7.Execute the last defined task
- 8.Jump back to the first task and follow the same flow
- From the firmware execution sequence, it is obvious that the order in which the task to be executed are fixed and they are hard coded in the code itself

Embedded Firmware design approaches

SUPER LOOP BASED APPROACH

- Example of “ Super Loop Based Design” is –Electronic video game toy containing keypad and display unit

Drawback of Super Loop based Design

- Major drawback of this approach is that any failure in any part of a single task will affect the total system. If the program hang up at any point while executing a task, it will remain there forever and ultimately the product will stop functioning
 - Some remedial measures are there
 - Use of Hardware and software Watch Dog Timers (WDTs) helps in coming out from the loop when an unexpected failure occurs or when the processor hang up
 - May cause additional hardware cost and firmware overhead
- Another major drawback is lack of real timeliness
 - If the number of tasks to be executed within an application increases, the time at which each task is repeated also increases. This brings the probability of missing out some events
 - For example in a system with keypad, there will be task for monitoring the keypad connected I/O lines and this need not be the task running while you press the keys
 - That is key pressing event may not be in sync with the keypad press monitoring task within the firmware
 - To identify the key press, you may have to press the key for a sufficiently long time till the keypad status monitoring task is executed internally. – Lead to lack of real timeliness

Embedded Firmware design approaches

Embedded Operating System Based Approach

- Contains OS, which can be either a General purpose Operating System (GPOS) or real Time Operating System (RTOS).

- **General purpose Operating System (GPOS) based design**

- GPOS based design is very similar to the conventional PC based Application development where the device contain an operating system and you will be creating and running user applications on top of it.

- Examples of Microsoft Windows XP OS are PDAs, Handheld devices/ Portable Devices and point of Sale terminals.

Use of GPOS in embedded product merges the demarcation of Embedded systems and General Purpose systems in terms of OS

- For developing applications on the top of the OS , OS supported APIs are used

- OS based applications also requires 'Driver Software' for OS based applications also requires 'Driver Software' for different hardware present on the board to communicate with them

Embedded Firmware design approaches

Embedded Operating System Based Approach

- RTOS based design
 - RTOS based design approach is employed in embedded product demanding Real Time Responses
 - RTOS respond in a timely and predictable manner to events
 - RTOS contain a real time Kernel responsible for performing pre-emptive multi tasking scheduler for scheduling the task, multiple thread etc.
 - RTOS allows a flexible scheduling of system resources like the CPU and Memory and offer some way to communicate between tasks
 - Examples of RTOS are
Windows CE, pSOS, VxWorks, ThreadX, Micro C/OS II, Embedded Linux, Symbian etc...

EMBEDDED FIRMWARE DEVELOPMENT LANGUAGES

- For embedded firmware development you can use either
 - Target processor/controller specific language (Assembly language) or
 - Target processor/ controller independent language (High level languages) or
 - Combination of Assembly and high level language

Languages for Programming Embedded Systems

- Assembly language was the pioneer for programming embedded systems till recently.
- Nowadays there are many more languages to program these systems. Some of the languages are C, C++, Ada, Forth, and Java together with its new enhancement J2ME.
- The majority of software for embedded systems is still done in C language.
- Recent survey indicates that approximately 45% of the embedded software is still being done in C language.
- C++ is also increasing its presence in embedded systems. As C++ is based on C language, thus providing programmer the object oriented methodologies to reap the benefits of such an approach.
- C is very close to assembly programming and it allows very easy access to underlying hardware.
- A huge number of high quality compilers and debugging tools are available for the C language.
- Though C++ is theoretically more efficient than C, but some of its compilers have bugs due to the huge size of the language. ☹ These compilers may cause a buggy execution.

Embedded Firmware Design Languages

- Hopefully, the distinction between software and firmware doesn't become less clear once we bring up programming languages. Unfortunately, simply looking at languages doesn't help clear up the confusion as these languages can crossover into the general purpose software domain. Here's a short list of some popular languages for software and embedded firmware development.
- **C/C++:** C is something of a legacy language for embedded systems and tends to be preferred over C++.
- **Hardware Description Languages (HDLs):** The most popular are probably VHDL or Verilog. If you're programming an FPGA or designing an ASIC, use an HDL to define digital logic. SystemVerilog is a good choice if you are designing a system with extensive digital logic.
- **Python/MicroPython:** Python is great for software applications, but many do not prefer it due to the periodic garbage collection operations required, which may create some latency problems. However, there is such a huge developer community around Python that developers can access a huge range of functionality and powerful computing capabilities.

Embedded development life cycle

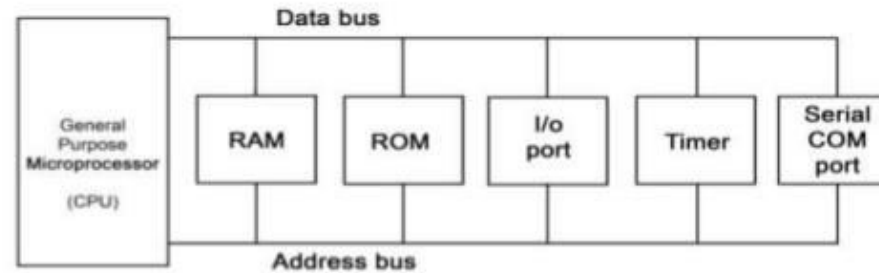
Separate PPT Enclosed

What is Microcontroller and Microprocessor?

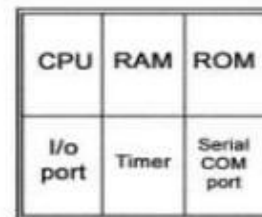
- **What is Microcontroller?**
- A microcontroller is a single-chip VLSI unit which is also called microcomputer. It contains all the memory and I/O interfaces needed, whereas a general-purpose microprocessor needs additional chips to offered by these necessary functions. Microcontrollers are widely used in embedded systems for real-time control applications.
- **What is a Microprocessor?**
- A microprocessor is a single chip semiconductor device. Its CPU contains a program counter, an ALU a stack pointer, working register, a clock timing circuit. It also includes ROM and RAM, memory decoder, and many serial and parallel ports.

Microprocessor and Microcontroller

(a). GENERAL PURPOSE MICROPROCESSOR SYSTEM



(b). MICROCONTROLLER



(Single chip)

Difference between Microprocessor and Microcontroller

Microprocessor	Microcontroller
It uses functional blocks like register, ALU, timing, and control units.	It uses functional blocks of microprocessors like RAM, timer, parallels I/O, ADC, and DAC.
In Microprocessor, bit handling instruction is less, One or two types only.	Microcontroller offers many kinds of bit handling instruction.
Offers rapid movements of code and data between external memory and microprocessor.	Offers rapid movements of code and data in the microcontroller.
Helps you to design general purpose digital computers system.	Helps you to design application-specific dedicated systems.
It allows you to do multitasking at a time.	It is a single task oriented system.
In Microprocessor system, you can decide the number of memory or I/O ports needed.	In Microcontroller system, the fixed number for memory or I/O makes a microcontroller ideal to complete the specific task.
Offers support for external memory and I/O ports, which makes it heavier and costlier system.	This type of system is lightweight and cheaper compares to the microprocessor.
External devices need more space, and their power consumption is quite higher.	This type of system consumes less amount of space, and power consumption is also very low.

How Embedded system Works?

- The processor may be a microprocessor or microcontroller. Microcontrollers are simply microprocessors with peripheral interfaces and integrated memory included. Microprocessors use separate integrated circuits for memory and peripherals instead of including them on the chip. Both can be used, but microprocessors typically require more support circuitry than microcontrollers because there is less integrated into the microprocessor. The term *system on a chip* ([SoC](#)) is often used. SoCs include multiple processors and interfaces on a single chip. They are often used for high-volume embedded systems. Some example SoC types are the application-specific integrated circuit ([ASIC](#)) and the field-programmable gate array (FPGA).

How Embedded system Works?

- Often, embedded systems are used in real-time operating environments and use a real-time operating system (RTOS) to communicate with the hardware. Near-real-time approaches are suitable at higher levels of chip capability, defined by designers who have increasingly decided the systems are generally fast enough and the tasks tolerant of slight variations in reaction. In these instances, stripped-down versions of the [Linux](#) operating system are commonly deployed, although other OSes have been pared down to run on embedded systems, including [Embedded Java](#) and Windows IoT (formerly Windows Embedded).

How does an embedded system work?

- Embedded systems always function as part of a complete device -- that's what's meant by the term *embedded*. They are low-cost, low-power-consuming, small computers that are embedded in other mechanical or electrical systems. Generally, they comprise a processor, power supply, and memory and communication ports. Embedded systems use the communication ports to transmit data between the processor and peripheral devices -- often, other embedded systems -- using a communication protocol. The processor interprets this data with the help of minimal software stored on the memory. The software is usually highly specific to the function that the embedded system serves.

References

- <https://www.watelectronics.com/classification-of-embedded-systems/>
- <https://www.slideshare.net/MoeMoeMyint/chapter-3-charateristics-and-quality-attributes-of-embedded-system>
- <https://electricalfundablog.com/embedded-system-characteristics-types-advantages-disadvantages/>
- https://www.tutorialspoint.com/embedded_systems/es_overview.htm
- <https://www.infopulse.com/blog/challenges-and-issues-of-embedded-software-development/>
- <https://radixweb.com/blog/emerging-trends-embedded-systems>
- <https://www.slideshare.net/ramankhipal3/trends-in-embedded-system-design>
- <https://www.arcweb.com/blog/embedded-systems-trends-technologies-0>

Embedded Development Life Cycle (EDLC)



Contents:

- Introduction
- What is EDLC
- Why EDLC
- Objectives of EDLC
- Different phases of EDLC
- EDLC approaches
- References

Introduction :

Software development company - SDLC

Product development company - EDLC

Example – Preparation of any food dish

- Dish selection and Ingredient list
- Procurement of the items in the list
- Preparation and initial taste testing
- Serving and final taste testing

Embedded Product development view

- Father - Overall management
- Mother - Developing and testing
- We - End user /client

What is EDLC?

- EDLC is an Analysis-Design-Implementation based problem solving approach for the product development.
 - Analysis – What product need to be developed
 - Design – Good approach for building it
 - Implementation – To develop it

Why edlc?

- Essential in understanding the scope and complexities involved in any Embedded product development.
- Defines interaction and activities among Various groups of product development sector.
 - Project management
 - System design and development
 - System testing
 - Release management and quality assurance

Objectives of EDLC

- Aim of any product development is the Marginal benefit
- Marginal benefit = Return on investment
- Product needs to be acceptable by the end user i.e. it has to meet the requirements of the end user in terms of quality, reliability & functionality.
- EDLC helps in ensuring all these requirements by following three objective
 - Ensuring that high quality products are delivered to user
 - Risk minimization and defect prevention in product development through project management
 - Maximize productivity

Ensuring high quality products

- The primary definition of quality in any embedded product development is return on investment achieved by the product.
- In order to survive in market, quality is very important factor to be taken care of while developing the product.
- Qualitative attributes depends on the budget of the product so budget allocation is very important.
- Budget allocation might have done after studying the market, trends & requirements of product, competition .etc.

Risk minimization & defect prevention through project management

- Project management (PM)
 - Adds an extra cost on budget
 - But essential for ensuring the development process is going in right direction
- Projects in EDLC requires Loose project management or tight project management.
- PM is required for
 - Predictability
 - Analyze the time to finish the product (PDS = no of person days)
 - Co-ordination
 - Resources (developers) needed to do the job
 - Risk management
 - Backup of resources to overcome critical situation
 - Ensuring defective product is not developed

Increased productivity

- Measure of efficiency as well as ROI

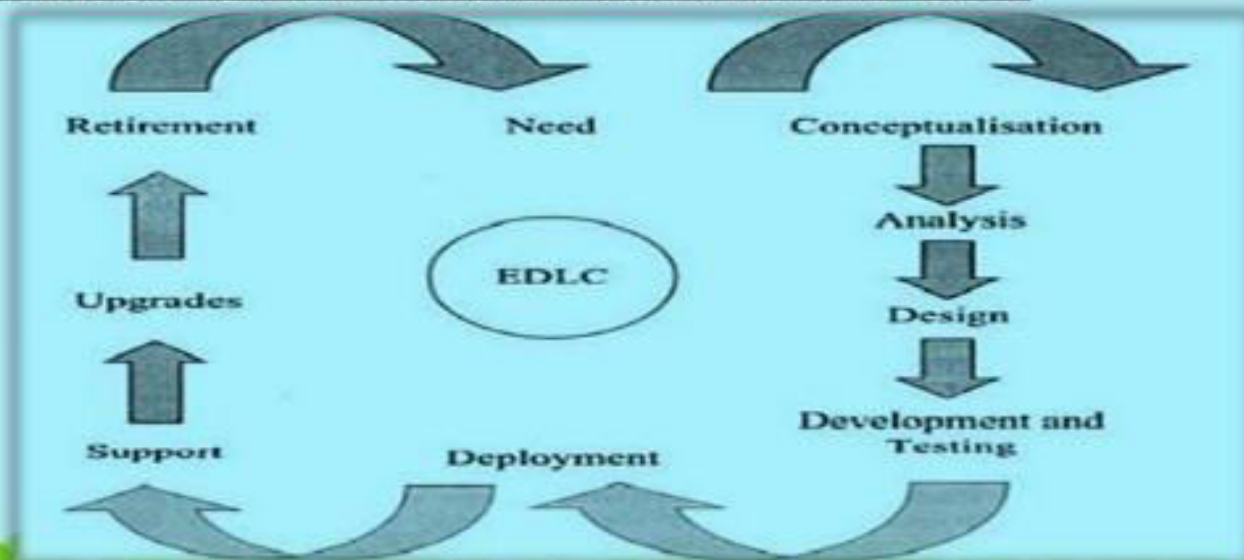
Different ways to improve the productivity are

- Saving the manpower
 - X members – X period
 - X/2 members – X period
- Use of automated tools where ever is required
- Re-usable effort – work which has been done for the previous product can be used if similarities present b/w previous and present product.
- Use of resources with specific set of skills which exactly matches the requirements of the product, which reduces the time in training the resource

Different phases of edlc

- A life cycle of product development is commonly referred as the “model”
- A simple model contains five phases
 - Requirement analysis
 - Design
 - Development and test
 - Deployment and maintenance
- The no of phases involved in EDLC model depends on the complexity of the product

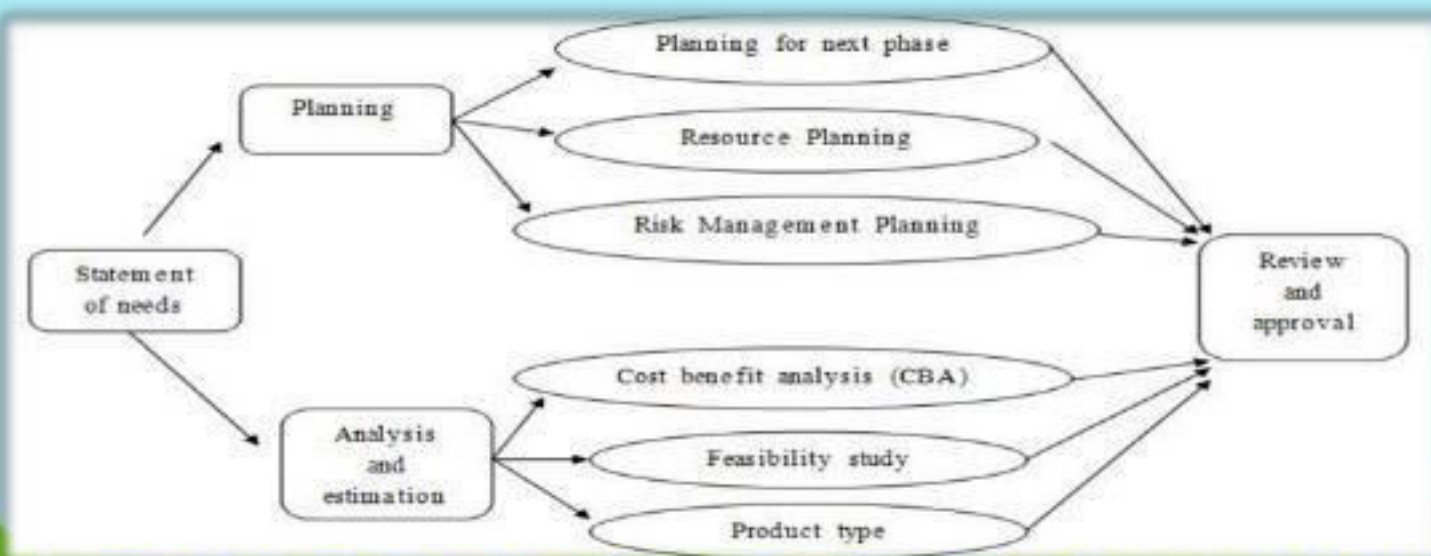
Classic Embedded product development life cycle model



NEED:

- Any embedded product may evolve as an output of a need.
- Need may come from an individual/from public/from company (generally speaking from an end user/client)
 - New/custom product development
 - Product re-engineering
 - Product maintenance

CONCEPTUALIZATION:



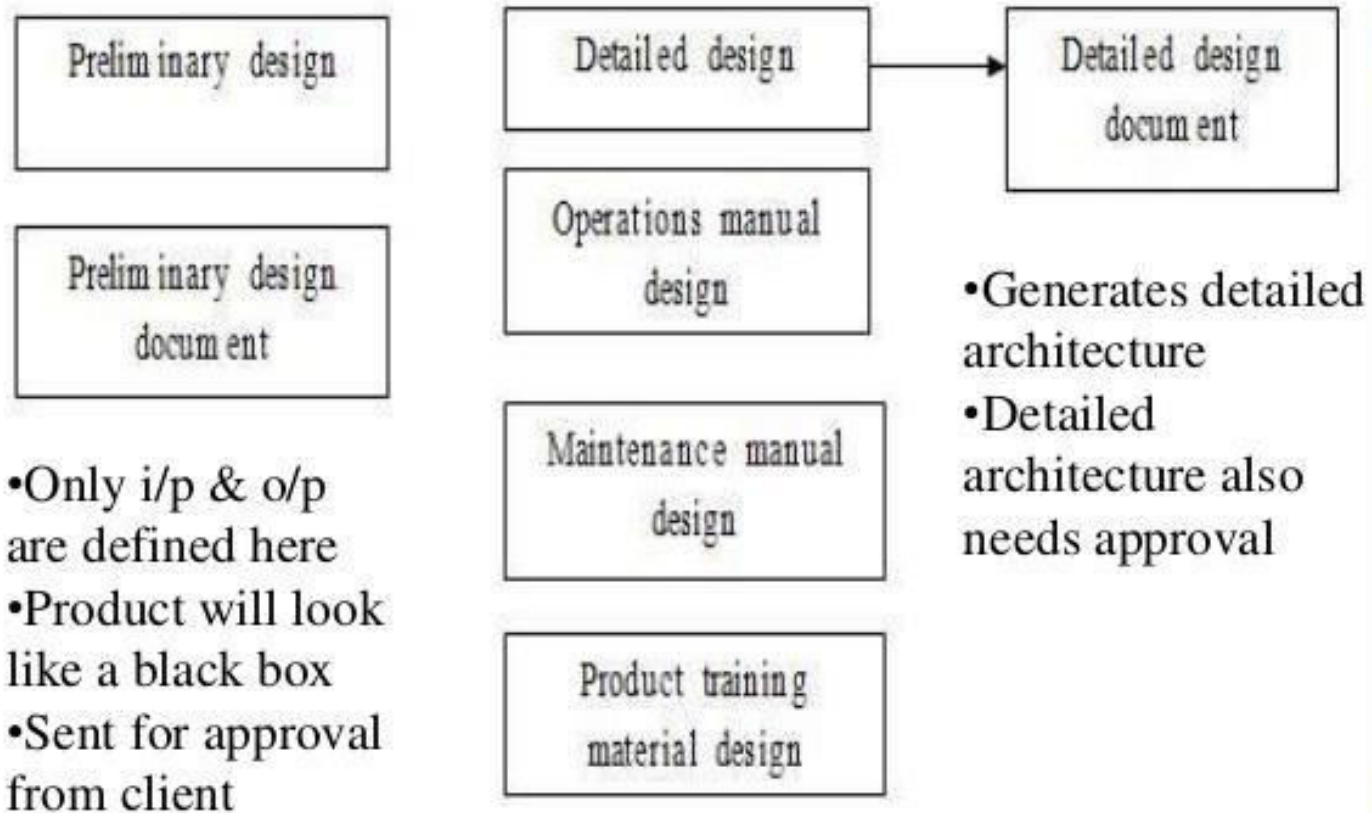
Conceptualization

The conceptualization phase of a project occurs in the initial design activity when the scope of the project is drafted and a list of the desired design features and requirements is created.

ANALYSIS:



DESIGN: Deals with the entire design of the product taking the requirements into consideration and focuses on how the functionalities can be delivered.



DEVELOPMENT AND TESTING:

- Development phase transforms the design into realizable product
- Design is transformed into hardware and firmware
- Look and feel of the device is very important

Testing phase can be divided into

- Unit testing – independent testing of hardware and firmware
- Integration testing – testing after integrating hardware and firmware
- System testing – testing of whole system on functionality and non-functionality basis
- User acceptance testing – testing of the product against the criteria mentioned by the end-user/client
- Test reports

DEPLOYMENT:

- A process of launching fully functional model into the market

SUPPORT:

- Deals with the operation and maintenance of the product
- Support should be provide to the end user/client to fix the bugs of the product

UPGRADES:

- Releasing of new version for the product which is already exists in the market
- Releasing of major bug fixes.

RETIREMENT/DISPOSAL:

- Everything changes, the technology you feel as the most advanced and best today may not be the same tomorrow
- Due to this the product cannot sustain in the market for long
- It has to be disposed on right time before it causes the loss.

EDLC APPROACHES:

Linear/Waterfall Model:



- Each phase of EDLC is executed in sequence
- Flow is unidirectional
- Output of one phase serving as input of other

Linear / Waterfall Model

The waterfall development model originates in the manufacturing and construction industries: highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

The steps are:

1. Specification.
2. Preliminary design.
3. Design review
4. Detailed design.
5. Design review.
6. Implementation.
7. Review.

Linear / Waterfall Model

Phases:

- 1) Requirement : In this phase we gather necessary information which will use for development of any project . For above example we gather information like which types of characteristics client wants. It also defines system requirement specification. This phase defines what to do.
- 2) Design: In design phase we then construct design to how to implement that requirements gathered into phase 1 .This phase define how to do .For this phase we then write algorithms
- 3) Coding: Now base on design phase we then write actual code to implement algorithms. This code should be efficient.
- 4) Testing : This phase use to test our coding part it checks all the validation...like our code should work for each and every possibilities of input if any bug occur then we have to report that bug to design phase or development phase.
- 5) Maintenance: In this phase we need keep updating information.
 1. The implementation process contains software preparation and transition activities, such as the conception and creation of the maintenance plan; the preparation for handling problems identified during development; and the follow up on product configuration management.

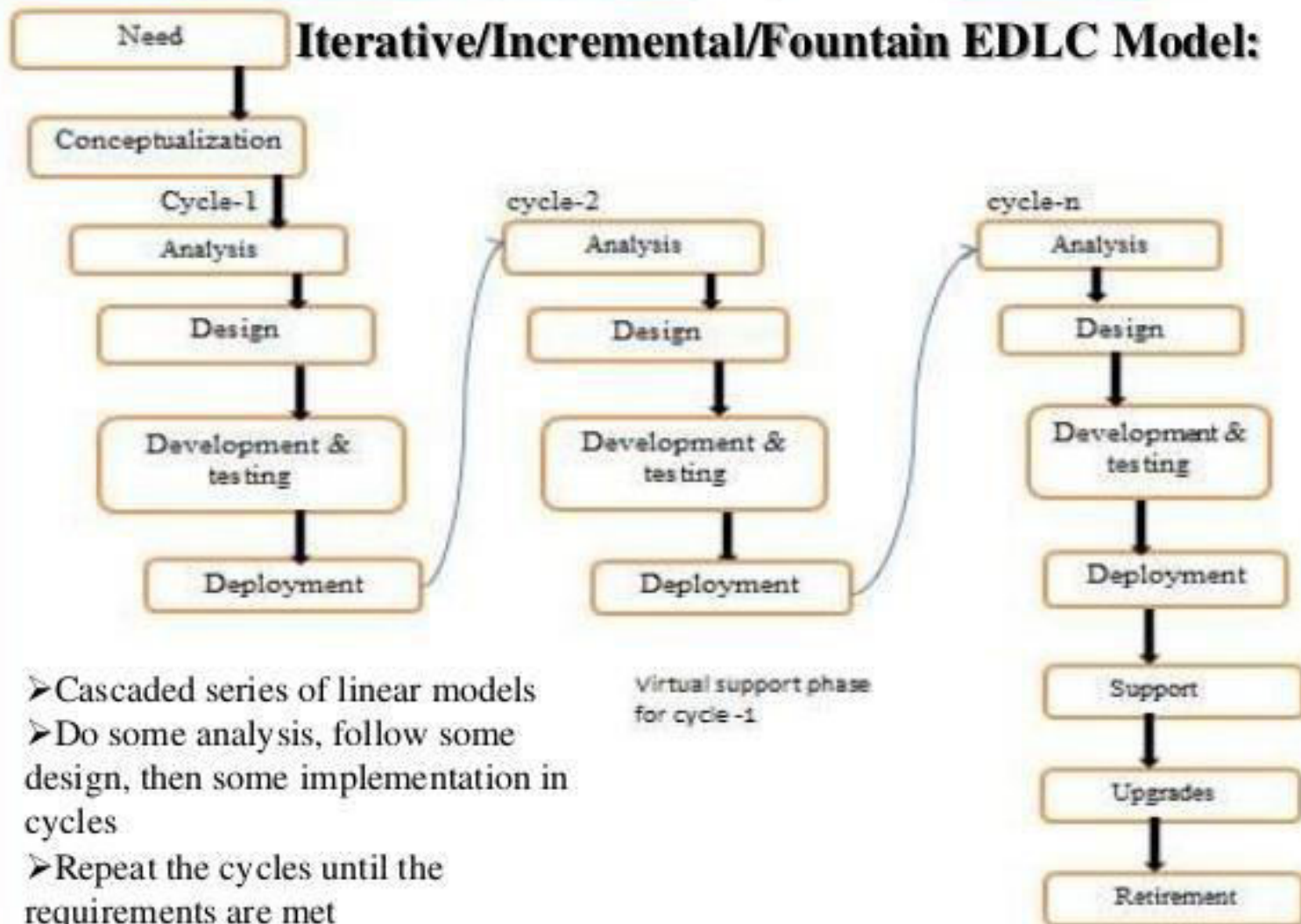
Linear / Waterfall Model

2. The problem and modification analysis process, which is executed once the application has become the responsibility of the maintenance group. The maintenance programmer must analyze each request, confirm it (by reproducing the situation) and check its validity, investigate it and propose a solution, document the request and the solution proposal, and, finally, obtain all the required authorizations to apply the modifications.
3. The process considering the implementation of the modification itself.
4. The process acceptance of the modification, by confirming the modified work with the individual who submitted the request in order to make sure the modification provided a solution.
5. The migration process is exceptional, and is not part of daily maintenance tasks. If the software must be ported to another platform without any change in functionality, this process will be used and a maintenance project team is likely to be assigned to this task.
6. Finally, the last maintenance process, also an event which does not occur on a daily basis, is the retirement of a piece

Strength and Weakness

Model/feature	Strengths	Weaknesses	When to Use
Waterfall	<ul style="list-style-type: none">• Easy to understand and implement.• Widely used and known.• Define before design, and design before coding.• Being a linear model, it is very simple to implement.• Works well on mature products and provides structure to inexperienced teams.• Minimizes planning overhead.• Phases are processed and completed one at a time.	<ul style="list-style-type: none">• All requirements must be known upfront• Inflexible.• Backing up to solve mistakes is difficult, once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.• A non-documentation deliverable only produced at the final phase.• Client may not be clear about what they want and what is needed.• Customers may have little opportunity to preview the system until it may be too late.• It is not a preferred model for complex and object-oriented projects.• High amounts of risk and uncertainty, thus, small changes or errors that arise in the completed software may cause a lot	<ul style="list-style-type: none">• When quality is more important than cost or schedule.• When requirements are very well known, clear, and fixed.• New version of existing product is needed.• Porting an existing product to a new platform

Iterative/Incremental/Fountain EDLC Model:

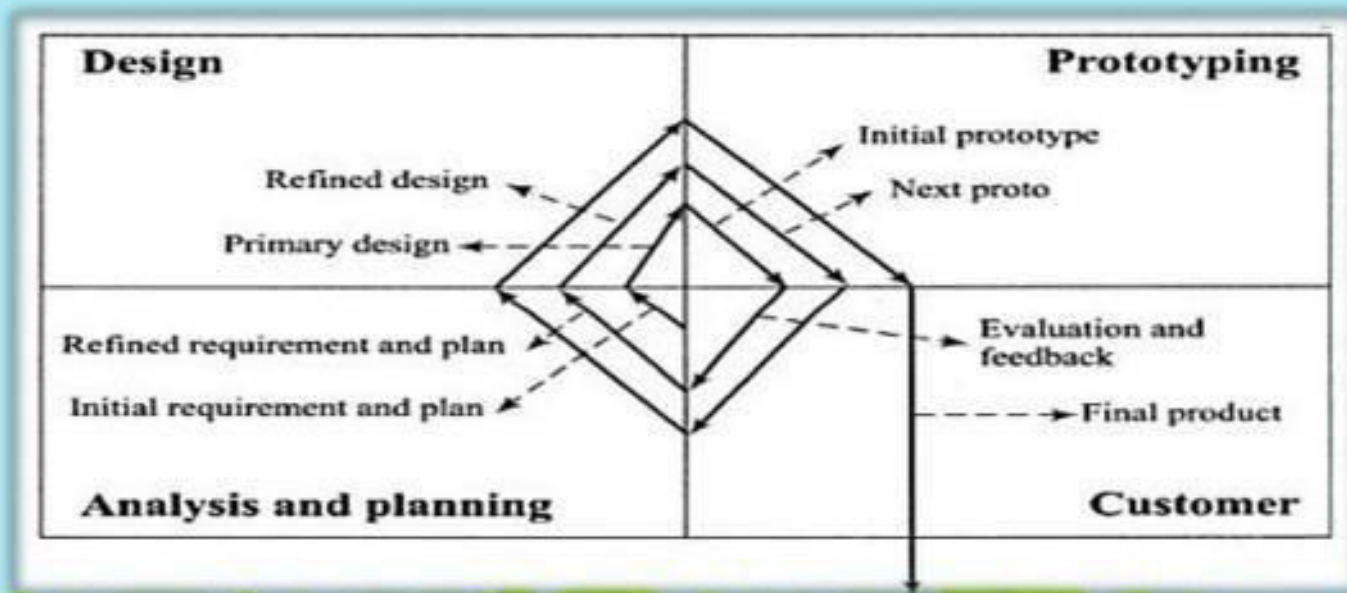


- Cascaded series of linear models
- Do some analysis, follow some design, then some implementation in cycles
- Repeat the cycles until the requirements are met

Model/feature	Strengths	Weaknesses	When to Use
Incremental/ Iterative.	<ul style="list-style-type: none"> • Develop high-risk or major functions first. • Risk is spread across smaller increments instead of concentrating in one large development. • Lessons learned at the end of each incremental delivery can result in positive revisions for the next increment. • Customers get important functionality early, and have an opportunity to respond to each build. • Each release delivers an operational product. • Initial product delivery is faster. • Reduces the risk of failure and changing the requirements. 	<ul style="list-style-type: none"> • Requires good planning and design. • Requires early definition of a complete and fully functional system to allow for the definition of increments. • The model does not allow for iterations within each increment. 	<ul style="list-style-type: none"> • On low to medium-risk projects. • A need to get basic functionality to the market early • On projects which have lengthy development schedules. • On a project with new technology, allowing the user to adjust to the system in smaller incremental steps rather than leaping to a major new product. • When it is high risky to develop the whole system at once.

Prototyping/evolutionary model:

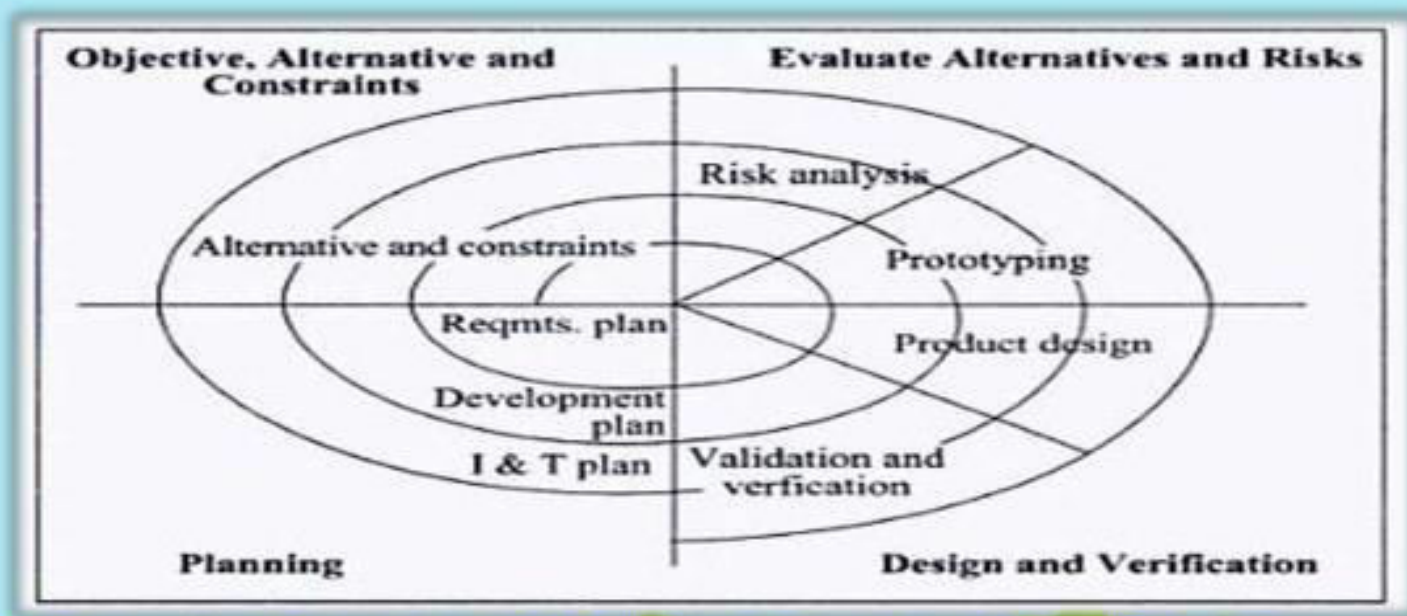
- Similar to iterative model, product is developed in multiple cycles
- The only difference is the model produces more refined prototype of the product at each cycle instead of just adding the functionality at each cycle like in iterative model.



The Rapid prototyping model is intended to provide a rapid implementation of high level portions of both the software and the hardware . The approach allows developers to construct working portion of hardware and software in incremental stages.Each stage through the cycle,one incorporates a little more of the intended functionality.The prototype is useful for both the designer and the customer. The prototype can be either evolutionary or throughway. It has the advantage of having a working system early in development process

Spiral model:

- Spiral model is best suited for the development of complex embedded products and situations where the requirements are changing from customer side.
- Risk evaluation in each stage helps in reducing risk



Spiral Model

The steps in spiral model life cycle are
Determine objective, alternatives, and constraints.

Identify and resolve risks.

Evaluate alternatives.

Develop deliverables-verify that they are correct.

Plan the next iteration.

Commit to an approach for the next iteration.

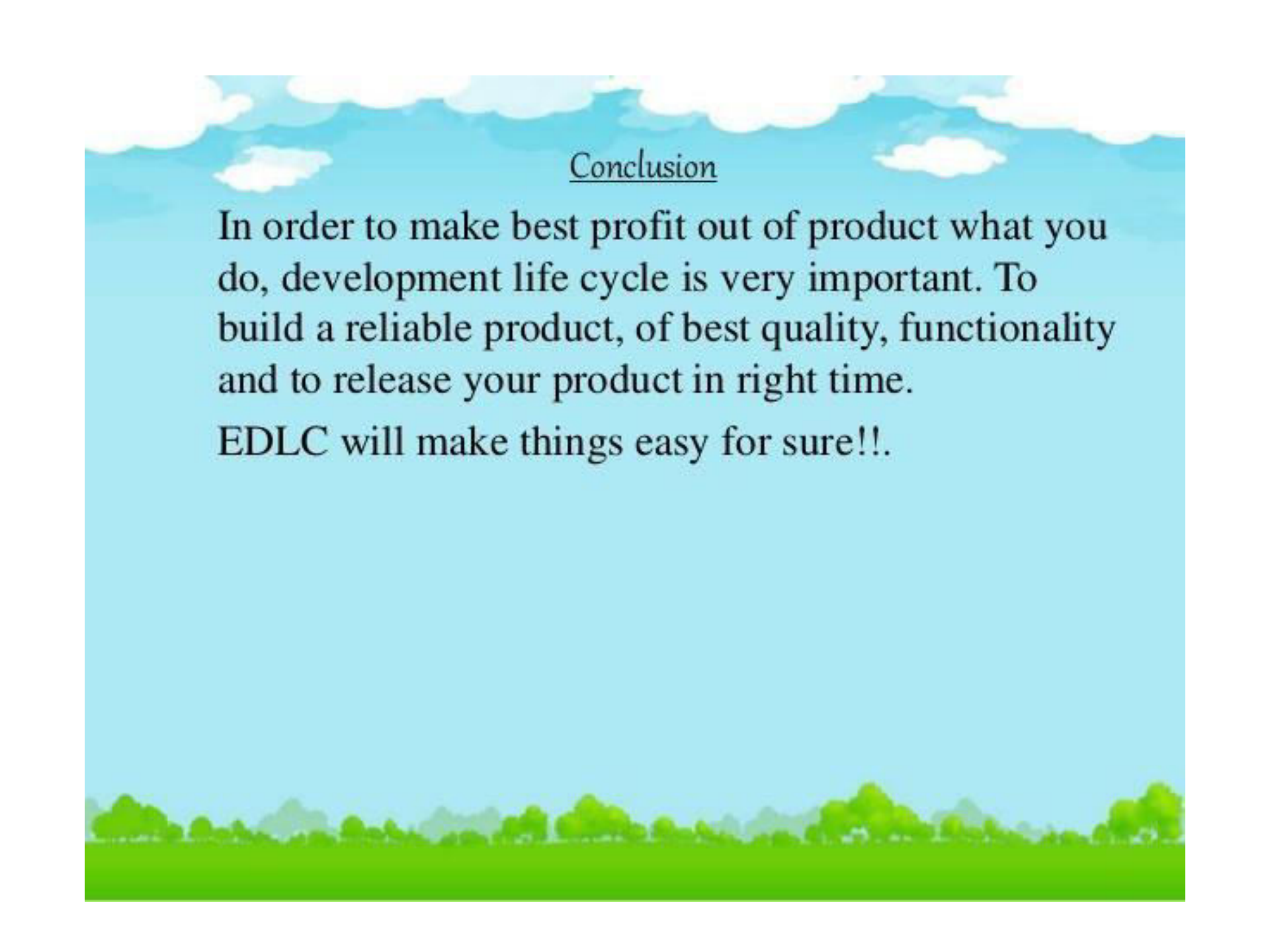
Spiral Model

Model/feature	Strengths	Weaknesses	When to Use
Spiral	<ul style="list-style-type: none"> • High amount of risk analysis. • Software is produced early in the software life cycle. • Strong approval and documentation control. • Additional functionality can be added at a later date. • Project monitoring is very easy and effective. • Concerned people of a project can early review each phase and each loop as well because of rapid prototyping tools. • Early and frequent feedback from users • Suitable to develop a highly customized product. • Provides early indication of insurmountable risks. 	<ul style="list-style-type: none"> • Cost involved in this model is usually high. • Risk assessment expertise is required. • Amount documentation required in intermediate stages makes management of a project very complex. • Time spent for evaluating risks for small or low-risk projects may be too large. • Time spent for planning, resetting objectives, doing risk analysis, and prototyping may be excessive. • Project's success is highly dependent on the risk analysis phase. 	<ul style="list-style-type: none"> • For medium to high-risk projects. • When risk evaluation and costs are important. • When significant changes are expected. • When users are not exactly sure what their needs.

Comparison

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rapid Model
Planning in early stage	Yes	Yes	Yes	No
Returning to an earlier phase	No	Yes	Yes	Yes
Handle Large-Project	Not Appropriate	Not Appropriate	Appropriate	Not Appropriate
Detailed Documentation	Necessary	Yes but not much	Yes	Limited
Cost	Low	Low	Expensive	Low
Requirement Specifications	Beginning	Beginning	Beginning	Time boxed release
Flexibility to change	Difficult	Easy	Easy	Easy
User Involvement	Only at beginning	Intermediate	High	Only at the beginning
Maintenance	Least	Promotes Maintainability	Typical	Easily Maintained
Duration	Long	Very long	Long	Short
Risk Involvement	High	Low	Medium to high risk	Low
Framework Type	Linear	Linear + Iterative	Linear + Iterative	Linear

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rapid Model
Testing	After completion of coding phase	After every iteration	At the end of the engineering phase	After completion of coding
Overlapping Phases	No	Yes (As parallel development is there)	No	Yes
Maintenance	Least Maintainable	Maintainable	Yes	Easily Maintainable
Re-usability	Least possible	To some extent	To some extent	Yes
Time-Frame	Very Long	Long	Long	Short
Working software availability	At the end of the life-cycle	At the end of every iteration	At the end of every iteration	At the end of the life cycle
Objective	High Assurance	Rapid Development	High Assurance	Rapid development
Team size	Large Team	Not Large Team	Large Team	Small Team
Customer control over administrator	Very Low	Yes	Yes	Yes



Conclusion

In order to make best profit out of product what you do, development life cycle is very important. To build a reliable product, of best quality, functionality and to release your product in right time.

EDLC will make things easy for sure!!.

Context Switching in OS

What is the context switching in the operating system?

- The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system.
- When switching perform in the system, it stores the old running process's status in the form of registers and assigns the CPU to a new process to execute its tasks. While a new process is running in the system, the previous process must wait in a ready queue. The execution of the old process starts at that point where another process stopped it.
- It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU to perform multiple tasks without the need for additional processors in the system.

When does context switching happen?

1. When a high-priority process comes to a ready state (i.e. with higher priority than the running process)
2. An Interrupt occurs
3. User and kernel-mode switch (It is not necessary though)
4. Preemptive CPU scheduling used.

The need for Context switching

- A context switching helps to share a single CPU across all processes to complete its execution and store the system's tasks status. When the process reloads in the system, the execution of the process starts at the same point where there is conflicting.
- **Following are the reasons that describe the need for context switching in the Operating system.**
- The switching of one process to another process is not directly in the system. A context switching helps the operating system that switches between the multiple processes to use the CPU's resource to accomplish its tasks and store its context. We can resume the service of the process at the same point later. If we do not store the currently running process's data or context, the stored data may be lost while switching between processes.
- If a high priority process falls into the ready queue, the currently running process will be shut down or stopped by a high priority process to complete its tasks in the system.
- If any running process requires I/O resources in the system, the current process will be switched by another process to use the CPUs. And when the I/O requirement is met, the old process goes into a ready state to wait for its execution in the CPU. Context switching stores the state of the process to resume its tasks in an operating system. Otherwise, the process needs to restart its execution from the initials level.
- If any interrupts occur while running a process in the operating system, the process status is saved as registers using context switching. After resolving the interrupts, the process switches from a wait state to a ready state to resume its execution at the same point later, where the operating system interrupted occurs.
- A context switching allows a single CPU to handle multiple process requests simultaneously without the need for any additional processors.

Example of Context Switching

- Suppose that multiple processes are stored in a Process Control Block (PCB). One process is running state to execute its task with the use of CPUs. As the process is running, another process arrives in the ready queue, which has a high priority of completing its task using CPU.
- Here we used context switching that switches the current process with the new process requiring the CPU to finish its tasks. While switching the process, a context switch saves the status of the old process in registers.
- When the process reloads into the CPU, it starts the execution of the process when the new process stops the old process. If we do not save the state of the process, we have to start its execution at the initial level.
- In this way, context switching helps the operating system to switch between the processes, store or reload the process when it requires executing its tasks.

Context switching triggers

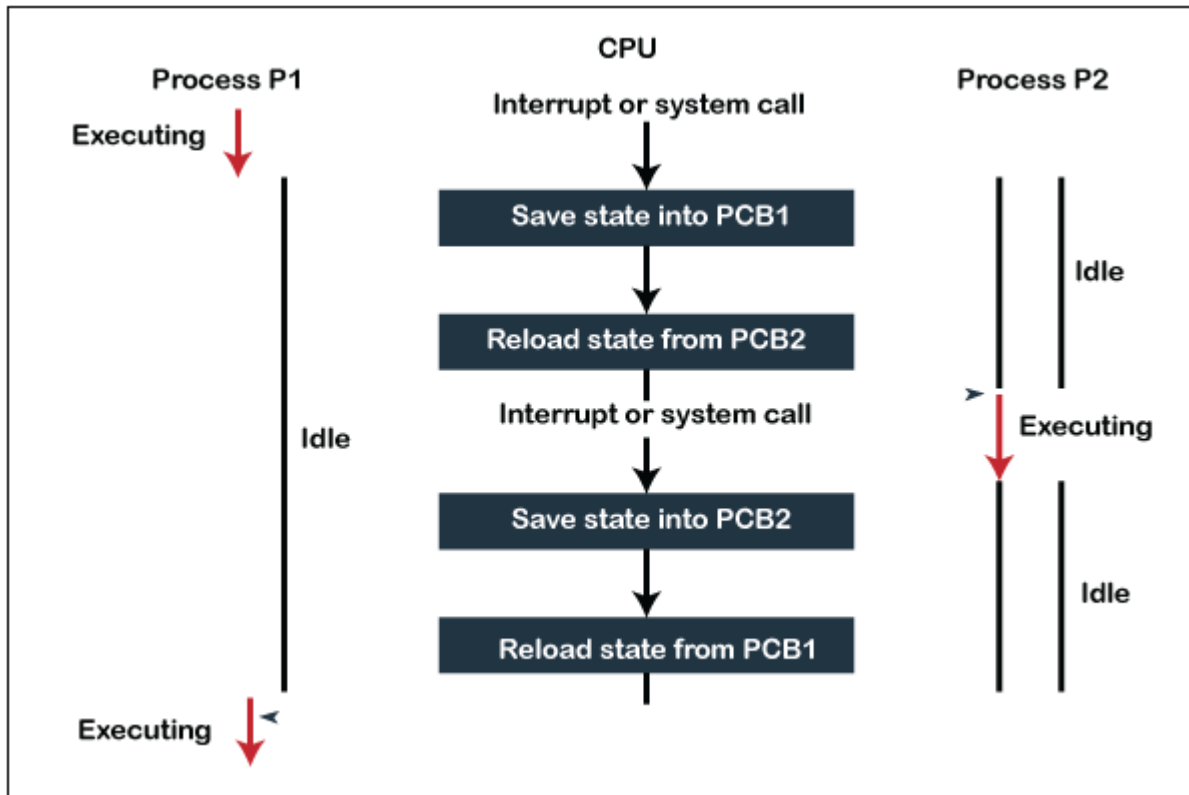
- Following are the three types of context switching triggers as follows.
 - Interrupts
 - Multitasking
 - Kernel/User switch
- **Interrupts:** A CPU requests for the data to read from a disk, and if there are any interrupts, the context switching automatic switches a part of the hardware that requires less time to handle the interrupts.
- **Multitasking:** A context switching is the characteristic of multitasking that allows the process to be switched from the CPU so that another process can be run. When switching the process, the old state is saved to resume the process's execution at the same point in the system.
- **Kernel/User Switch:** It is used in the operating systems when switching between the user mode, and the kernel/user mode is performed.

What is the PCB?

- A PCB (Process Control Block) is a data structure used in the operating system to store all data related information to the process. For example, when a process is created in the operating system, updated information of the process, switching information of the process, terminated process in the PCB.

Steps for Context Switching

- There are several steps involved in context switching of the processes. The following diagram represents the context switching of two processes, P1 to P2, when an interrupt, I/O needs, or priority-based process occurs in the ready queue of PCB.



- As we can see in the diagram, initially, the P1 process is running on the CPU to execute its task, and at the same time, another process, P2, is in the ready state.
- If an error or interruption has occurred or the process requires input/output, the P1 process switches its state from running to the waiting state. Before changing the state of the process P1, context switching saves the context of the process P1 in the form of registers and the program counter to the **PCB1**.
- After that, it loads the state of the P2 process from the ready state of the **PCB2** to the running state.

Context Switching

- **The following steps are taken when switching Process P1 to Process 2:**
- First, the context switching needs to save the state of process P1 in the form of the program counter and the registers to the PCB (Program Counter Block), which is in the running state.
- Now update PCB1 to process P1 and moves the process to the appropriate queue, such as the ready queue, I/O queue and waiting queue.
- After that, another process gets into the running state, or we can select a new process from the ready state, which is to be executed, or the process has a high priority to execute its task.
- Now, we have to update the PCB (Process Control Block) for the selected process P2. It includes switching the process state from ready to running state or from another state like blocked, exit, or suspend.
- If the CPU already executes process P2, we need to get the status of process P2 to resume its execution at the same time point where the system interrupt occurs.
- Similarly, process P2 is switched off from the CPU so that the process P1 can resume execution. P1 process is reloaded from PCB1 to the running state to resume its task at the same point. Otherwise, the information is lost, and when the process is executed again, it starts execution at the initial level.

Context Switching Cost

- Context Switching leads to an overhead cost because of TLB flushes, sharing the cache between multiple tasks, running the task scheduler etc. Context switching between two threads of the same process is faster than between two different processes as threads have the same virtual memory maps. Because of this TLB flushing is not required.

Advantage of Context Switching

- Context switching is used to achieve multitasking i.e. multiprogramming with time-sharing.
- Multitasking gives an illusion to the users that more than one process are being executed at the same time. But in reality, only one task is being executed at a particular instant of time by a processor. Here, the context switching is so fast that the user feels that the CPU is executing more than one task at the same time.

The disadvantage of Context Switching

- The disadvantage of context switching is that it requires some time for context switching i.e. the context switching time. Time is required to save the context of one process that is in the running state and then getting the context of another process that is about to come in the running state. During that time, there is no useful work done by the CPU from the user perspective. So, context switching is pure overhead in this condition.

- <https://www.javatpoint.com/what-is-the-context-switching-in-the-operating-system>
- <https://afteracademy.com/blog/what-is-context-switching-in-operating-system>

CSPC702 - EMBEDDED SYSTEMS AND INTERNET OF THINGS(IOT)

UNIT II

Real Time Operating Systems

Syllabus

UNIT – II Real Time Operating Systems

- Prime Movers: Real time without RTOS, Task states, Task table and data – Multitasking operating systems – Context switches – Kernels – Task swapping methods – Scheduler algorithms – Inter process communication mechanism-memory communication, Message passing, Signals. Overview of ARM Architecture, Programmer's model and Development Tools.

Introduction to RTOS (Real Time Operating Systems)

- **What is an Operating System (OS)?**
- Generally, an operating system, or OS, is the life support system of a computer.
- This is a piece of software that ensures the operation of the hardware and is responsible for the interaction between the hardware and the applications that run on the computer.
- An operating system fulfills a number of significant functions including the management of the CPU, memory, and peripheral devices of the computer. Also, an OS mediates between the user and the machine providing a user interface and services for all applications.
- An operating system is a computer program that supports a computer's basic functions, and provides services to other programs (or *applications*) that run on the computer. The applications provide the functionality that the user of the computer wants or needs. The services provided by the operating system make writing the applications faster, simpler, and more maintainable. If you are reading this web page, then you are using a web browser (the application program that provides the functionality you are interested in), which will itself be running in an environment provided by an operating system.

What is a Real-Time Operating System (RTOS)?

- **Real-time operating system (RTOS)** is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay. The full form of RTOS is Real time operating system.
- In a RTOS, Processing time requirement are calculated in tenths of seconds increments of time. It is time-bound system that can be defined as fixed time constraints. In this type of system, processing must be done inside the specified constraints. Otherwise, the system will fail. The Figure Shows Real-time system with RTOS

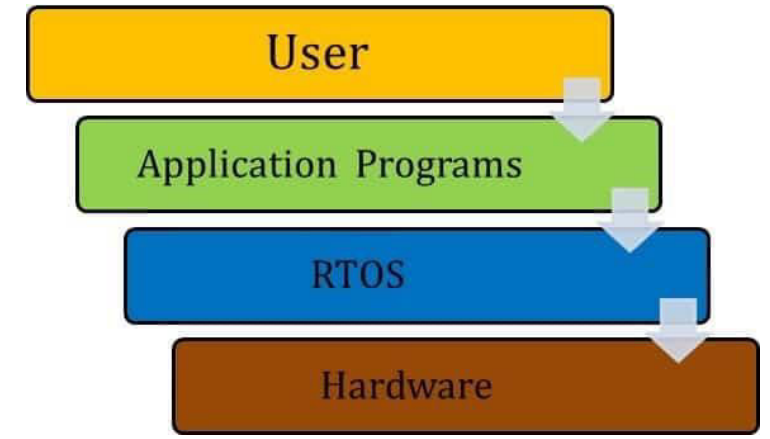


Fig. Realtime system with RTOS

OS or RTOS?

- The difference between an OS (Operating System) such as Windows or Unix and an RTOS (Real Time Operating System) found in embedded systems, is the response time to external events. OS's typically provide a non-deterministic, soft real time response, where there are no guarantees as to when each task will complete, but they will try to stay responsive to the user.
- An RTOS differs in that it typically provides a hard real time response, providing a fast, highly deterministic reaction to external events. The difference between the two can be highlighted through examples – compare, for example, the editing of a document on a PC to the operation of a precision motor control.

Why use an RTOS?

Here are important reasons for using RTOS:

- It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing.
- The Real time OS provides API functions that allow cleaner and smaller application code.
- Abstracting timing dependencies and the task-based design results in fewer interdependencies between modules.
- RTOS offers modular task-based development, which allows modular task-based testing.
- The task-based API encourages modular development as a task, will typically have a clearly defined role. It allows designers/teams to work independently on their parts of the project.
- An RTOS is event-driven with no time wastage on processing time for the event which is not occur

Why use an RTOS?

- There are well-established techniques for writing good embedded software without the use of an RTOS. In some cases, these techniques may provide the most appropriate solution; however as the solution becomes more complex, the benefits of an RTOS become more apparent. These include:
- **Priority Based Scheduling:** The ability to separate critical processing from non-critical is a powerful tool.
- **Abstracting Timing Information:** The RTOS is responsible for timing and provides API functions. This allows for cleaner (and smaller) application code.
- **Maintainability/Extensibility:** Abstracting timing dependencies and task based design results in fewer interdependencies between modules. This makes for easier maintenance.
- **Modularity:** The task based API naturally encourages modular development as a task will typically have a clearly defined role.
- **Promotes Team Development:** The task-based system allows separate designers/teams to work independently on their parts of the project.
- **Easier Testing:** Modular task based development allows for modular task based testing.
- **Code Reuse:** Another benefit of modularity is that similar applications on similar platforms will inevitably lead to the development of a library of standard tasks.
- **Improved Efficiency:** An RTOS can be entirely event driven; no processing time is wasted polling for events that have not occurred.
- **Idle Processing:** Background or idle processing is performed in the idle task. This ensures that things such as CPU load measurement, background CRC checking etc will not affect the main processing.

Components of RTOS

Here, are important Component of RTOS

The Scheduler: This component of RTOS tells that in which order, the tasks can be executed which is generally based on the priority.

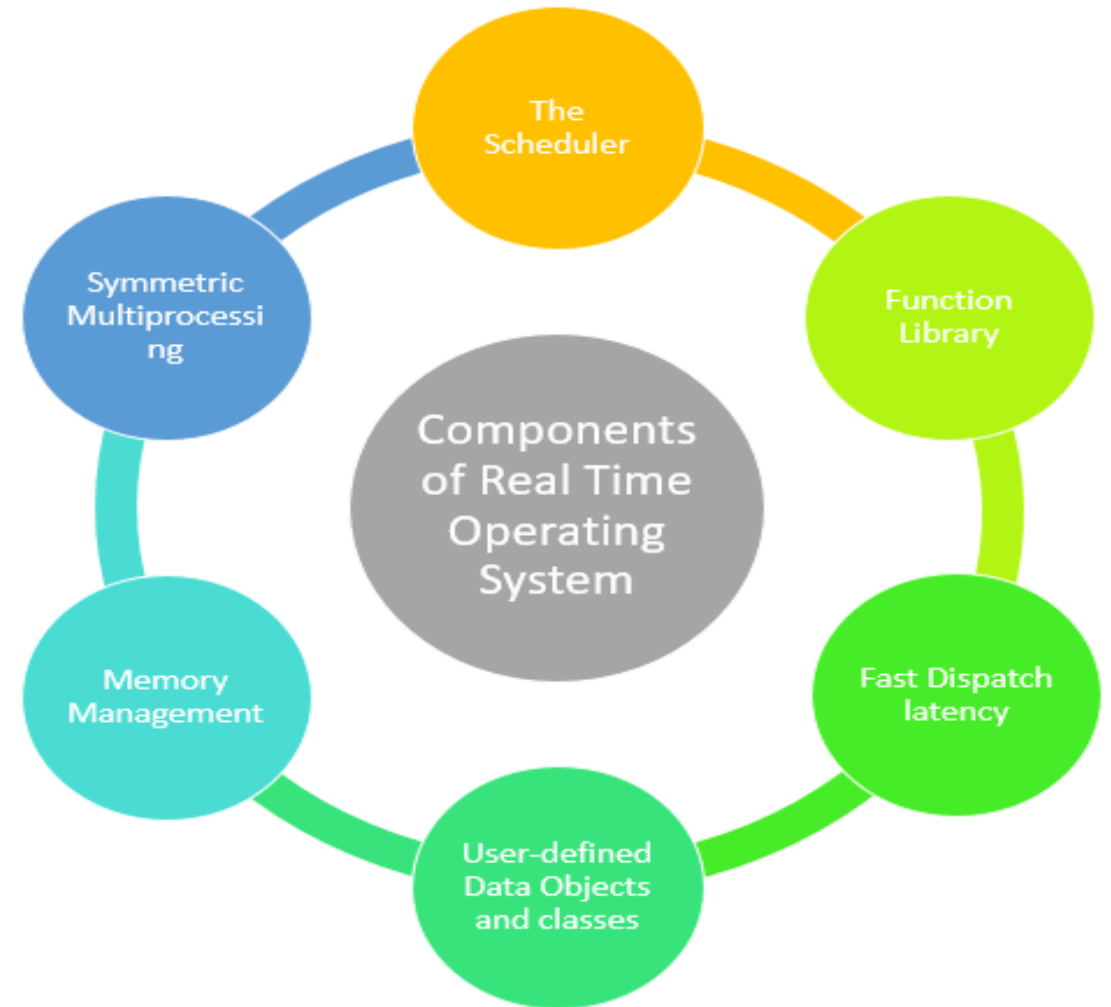
Symmetric Multiprocessing (SMP): It is a number of multiple different tasks that can be handled by the RTOS so that parallel processing can be done.

Function Library: It is an important element of RTOS that acts as an interface that helps you to connect kernel and application code. This application allows you to send the requests to the Kernel using a function library so that the application can give the desired results.

Memory Management: this element is needed in the system to allocate memory to every program, which is the most important element of the RTOS.

Fast dispatch latency: It is an interval between the termination of the task that can be identified by the OS and the actual time taken by the thread, which is in the ready queue, that has started processing.

User-defined data objects and classes: RTOS system makes use of programming languages like C or C++, which should be organized according to their operation.



Types of RTOS

Three types of RTOS systems are:

- **Soft Real Time:**

- Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS.
- Example: Online Transaction system and Livestock price quotation System.

- **Hard Real Time :**

- In Hard RTOS, the deadline is handled very strictly which means that given task must start executing on specified scheduled time, and must be completed within the assigned time duration.
- Example: Medical critical care system, Aircraft systems, etc.

- **Firm Real time:**

- These type of RTOS also need to follow the deadlines. However, missing a deadline may not have big impact but could cause undesired affects, like a huge reduction in quality of a product.
- Example: Various types of Multimedia applications.

Types of RTOS

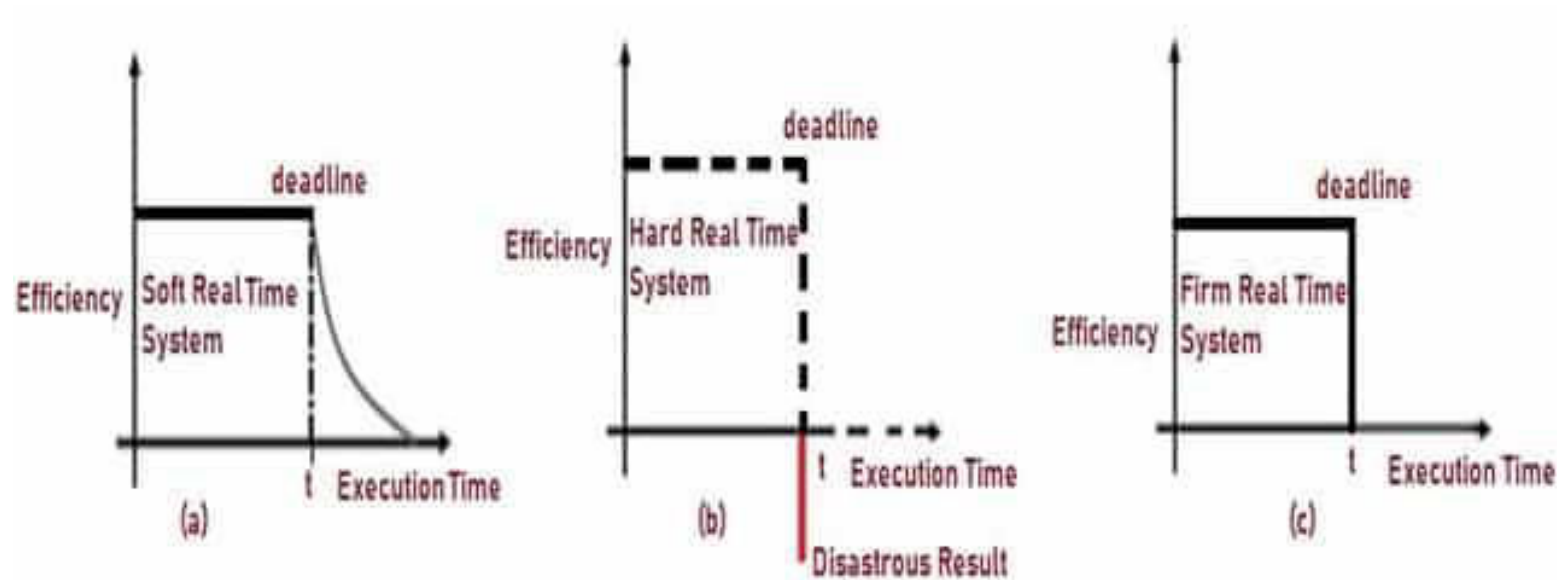


Fig. Time Efficiency Curves of different RTOS

Terms used in RTOS

Here, are essential terms used in RTOS:

- **Task** – A set of related tasks that are jointly able to provide some system functionality.
- **Job** – A job is a small piece of work that can be assigned to a processor, and that may or may not require resources.
- **Release time of a job** – It's a time of a job at which job becomes ready for execution.
- **Execution time of a job:** It is time taken by job to finish its execution.
- **Deadline of a job:** It's time by which a job should finish its execution.
- **Processors:** They are also known as active resources. They are important for the execution of a job.
- **Maximum It is the** allowable response time of a job is called its relative deadline.
- **Response time of a job:** It is a length of time from the release time of a job when the instant finishes.
- **Absolute deadline:** This is the relative deadline, which also includes its release time.

Goals of the RTOS

- **Small latency:** It is real-time after all!
- **Determinism:** Again, it is real-time. You need to know how long things take to process to make sure deadlines are met.
- **Structured Software:** With an RTOS, you are able to divide and conquer in a structured manner. It's straight-forward to add additional components into the application.
- **Scalability:** An RTOS must be able to scale from a simple application to a complex one with stacks, drivers, file systems, etc.
- **Offload development:** An RTOS manages many aspects of the system which allows a developer to focus on their application. For example an RTOS, along with scheduling, generally handles power management, interrupt table management, memory management, exception handling, etc.

Features of RTOS

Here are important features of RTOS:

- Occupy very less memory
- Consume fewer resources
- Response times are highly predictable
- Unpredictable environment
- The Kernel saves the state of the interrupted task and then determines which task it should run next.
- The Kernel restores the state of the task and passes control of the CPU for that task.

What is RTOS in Embedded System | Parameters for Selection

- RTOS is used in [Embedded Systems](#) as it requires real-time data. For a Real Time OS to be functional, following parameters are to be considered:
- ***Performance***
- ***Error-Free***
- ***Maximum Utilization***
- ***Task Shift***
- ***Middleware Support***
- **Embedded system usage**
- **Unique features**
- **24/7 performance**

Factors for selecting an RTOS

Here, are essential factors that you need to consider for selecting RTOS:

- **Performance:** Performance is the most important factor required to be considered while selecting for a RTOS.
- **Middleware:** if there is no middleware support in Real time operating system, then the issue of time-taken integration of processes occurs.
- **Error-free:** RTOS systems are error-free. Therefore, there is no chance of getting an error while performing the task.
- **Embedded system usage:** Programs of RTOS are of small size. So we widely use RTOS for embedded systems.
- **Maximum Consumption:** we can achieve maximum Consumption with the help of RTOS.
- **Task shifting:** Shifting time of the tasks is very less.
- **Unique features:** A good RTS should be capable, and it has some extra features like how it operates to execute a command, efficient protection of the memory of the system, etc.
- **24/7 performance:** RTOS is ideal for those applications which require to run 24/7.

What should be considered when choosing an RTOS?

- **Responsiveness:** The RTOS scheduling algorithm, interrupt latency and context switch times will significantly define the responsiveness and determinism of the system. The most important consideration is what type of response is desired – Is a hard real time response required? This means that there are precisely defined deadlines that, if not met, will cause the system to fail. Alternatively, would a non-deterministic, soft real time response be appropriate? In which case there are no guarantees as to when each task will complete.
- **Available system resources:** Micro kernels use minimum system resources and provide limited but essential task scheduling functionality. Micro kernels generally deliver a hard real time response, and are used extensively with embedded microprocessors with limited RAM/ROM capacity, but can also be appropriate for larger embedded processor systems.
- Alternatively, a full featured OS like Linux or WinCE could be used. These provide a feature rich operating system environment, normally supplied with drivers, GUI's and middleware components. Full featured OS's are generally less responsive, require more memory and more processing power than micro kernels, and are mainly used on powerful embedded processors where system resources are plentiful.

What should be considered when choosing an RTOS?

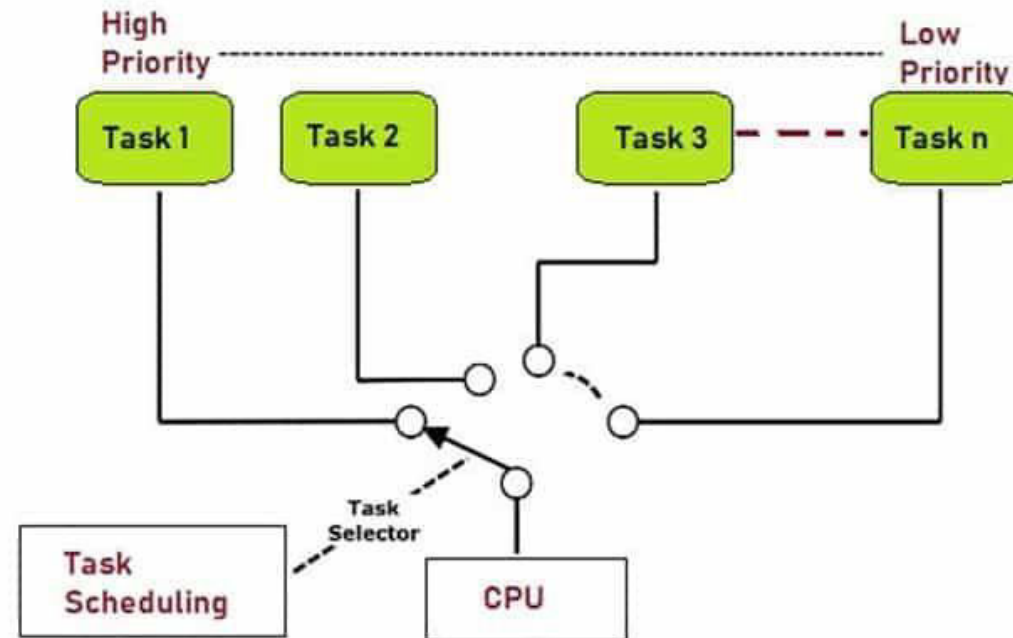
- **Open source or professionally licensed:** There are widely used, free open source RTOS's available, distributed under GPL or modified GPL licenses. However, these licenses may contain copy left restrictions and offer little protection. Professionally licensed RTOS products remove the copy left restrictions, offer full IP infringement indemnification and warranties. In addition, you have a single company providing support and taking responsibility for the quality of your product.
- **Quality:** What emphasis does the RTOS supplier place on quality within their organisation? Quality is more than just a coding standard. Are the correct procedures in place to guarantee the quality of future products and support? Well-managed companies that take quality seriously tend to be ISO 9001 certified.
- **Safety Certification:** Pre-certified and certifiable RTOS's are available for applications that require certification to international design standards such as DO-178C and IEC 61508. These RTOS's provide key safety features, and the design evidence required by certification bodies to confirm that the process used to develop the RTOS meets the relevant design standard.
- **Licensing:** It's not only the RTOS functionality and features that you'll need to consider, but the licensing model that will work best for your project budget and the company's "return on investment".
- **RTOS Vendor:** The company behind the RTOS is just as important as selecting the correct RTOS itself. Ideally you want to build a relationship with a supplier that can support not only your current product, but also your products of the future. To do this you need to select a proactive supplier with a good reputation, working with leading silicon manufacturers to ensure they can support the newest processors and tools.
- Trust, quality of product, and quality of support is everything.

The use of RTOS in embedded designs

- Many embedded programmers shy away from using an RTOS because they suspect that it adds too much complexity to their application, or it is simply unknown territory. An RTOS typically requires anything up to 5% of the CPU's resources to perform its duties. While there will always be some resource penalties, an RTOS can make up for it in areas such as simplified determinism, ease of use through HW abstraction, reduced development time and easier debugging.
- Using an RTOS means you can run multiple tasks concurrently, bringing in the basic connectivity, privacy, security, and so on as and when you need them. An RTOS allows you to create an optimized solution for the specific requirements of your project.

How does RTOS (Real Time Operating System) Work?

- Real Time Application requests are serviced by Real Time Operating System. The RTOS allows multiple tasks or programs to execute simultaneously based on its priority. Task scheduling Unit decides which thread is to be executed. The processor suspends the running task (if any) and executes the high priority task it receives.



Schematic Representation of Working of RTOS

How does RTOS (Real Time Operating System) Work?

- Let us say, a user is browsing on the net and after few seconds switches on YouTube Video and in no time, the user starts listening to a Podcast. Browsing on the net is considered as 'Task 1', Video on YouTube is Task 2 and listening to Podcast is assumed to be Task 3. Switching between these tasks is called as Multi-Tasking and RTOS provides efficient multitasking services.
- When the User shifts to Task 2, then Task 1 is terminated and executes Task 2 as its priority is High. Similarly, when the User shifts to Task 3, then Task 2 gets terminated and task 3 is executed. Task Scheduling Unit takes care of these functions. Inter task [communication](#), Synchronization, Time management is taken care of by RTOS Kernel. Schematic representation of the same is shown in the above fig(**Schematic Representation of Working of RTOS**).

Difference between in GPOS and RTOS

General-Purpose Operating System (GPOS)	Real-Time Operating System (RTOS)
It used for desktop PC and laptop.	It is only applied to the embedded application.
Process-based Scheduling.	Time-based scheduling used like round-robin scheduling.
Interrupt latency is not considered as important as in RTOS.	Interrupt lag is minimal, which is measured in a few microseconds.
No priority inversion mechanism is present in the system.	The priority inversion mechanism is current. So it can not modify by the system.
Kernel's operation may or may not be preempted.	Kernel's operation can be preempted.
Priority inversion remain unnoticed	No predictability guarantees

Comparison between GPOS and RTOS



RTOS and GPOS Differences

RTOS

Real-Time Operating System

- Deterministic: no random execution pattern
- Predictable Response Times
- Time Bound
- Preemptive Kernel

Examples:

Contiki source code, FreeRTOS™, Zephyr™ Project

Use Case:

Embedded Computing

GPOS

General-Purpose Operating System

- Dynamic memory mapping
- Random Execution Pattern
- Response Times not Guaranteed

Examples:

Microsoft® Windows® operating system, Apple® macOS® operating system, Red Hat® Enterprise Linux® operating system

Use Case:

Desktop, Laptop, Tablet computers

Applications of Real Time Operating System

Where are RTOS (Real Time Operating System) Used?

RTOS finds its application in [embedded systems](#) because of its accurate real-time data. Some of the major applications are listed below:

- | | |
|---|--|
| <ul style="list-style-type: none">➤ Airlines reservation system.➤ Air traffic control system.➤ Systems that provide immediate updating.➤ Used in any system that provides up to date and minute information on stock prices.➤ Defense application systems like RADAR.➤ Command Control Systems | <ul style="list-style-type: none">➤ Internet Telephony➤ Heart Pacemaker➤ Mobile applications.➤ Online transaction system.➤ Medical Critical Care systems.➤ Price quotation systems.➤ Network and multimedia systems.➤ Anti-Lock brake systems.➤ Online calling.➤ Ticket reservation systems.➤ Command control systems. |
|---|--|

Advantages of RTOS (Real Time Operating System)

- **The advantages of Real-Time Operating System include:**
- RTOS is event-driven with no processing time delay.
- Real Time OS offers task-based API development. This helps designers or testers to work independently on their parts of the project.
- It reduces the interdependencies between the modules by abstracting timing dependencies and task-based design.
- It offers cleaners and smaller application courses.
- Priority-based scheduling allows the user to separate analytical processing time and Critical processing time.

Disadvantages of RTOS

Here, are drawbacks/cons of using RTOS system:

- RTOS system can run minimal tasks together, and it concentrates only on those applications which contain an error so that it can avoid them.
- RTOS is the system that concentrates on a few tasks. Therefore, it is really hard for these systems to do multi-tasking.
- Specific drivers are required for the RTOS so that it can offer fast response time to interrupt signals, which helps to maintain its speed.
- Plenty of resources are used by RTOS, which makes this system expensive.
- The tasks which have a low priority need to wait for a long time as the RTOS maintains the accuracy of the program, which are under execution.
- Minimum switching of tasks is done in Real time operating systems.
- It uses complex algorithms which is difficult to understand.
- RTOS uses lot of resources, which sometimes not suitable for the system.

Prime Movers

- The **prime mover** behind a plan, idea, or situation is someone who has an important influence in starting it. (Or)
- It is a person who is chiefly responsible for the creation or execution of a plan.

(Or)

- an initial source of motive power.
- **What is an example of a prime mover?**

Windmills, waterwheels, turbines, steam engines, and internal-combustion engines are **prime movers**. In these machines the inputs vary; the outputs are usually rotating shafts capable of being used as inputs to other machines, such as electric generators, hydraulic pumps, or air compressors.

- **What are prime movers and its types?**

These technologies include different **prime mover types** such as steam turbines, gas turbines, reciprocating internal combustion engines, micro-gas turbines, micro-steam turbines, Stirling engines, fuel cells, and thermal photovoltaic systems.

- **What is prime mover concept?**

'that which moves without being moved') or **prime mover** (Latin: primum movens) is a **concept** advanced by Aristotle as a primary cause (or first uncaused cause) or "**mover**" of all the motion in the universe. As is implicit in the name, the unmoved **mover** moves other things, but is not itself moved by any prior action.

Real time without RTOS

Task states, Task table and data Task Management In RTOS

- The application is decomposed into small, schedulable, and sequential program units known as “Task”, a basic unit of execution and is governed by three time critical properties; release time, deadline and execution time. Release time refers to the point in time from which the task can be executed. Deadline is the point in time by which the task must complete. Execution time denotes the time the task takes to execute.

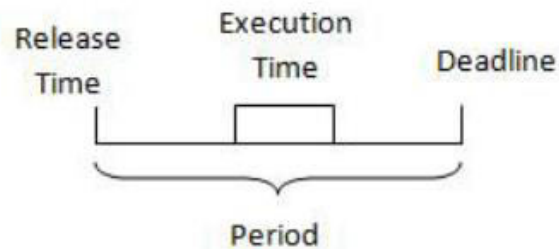
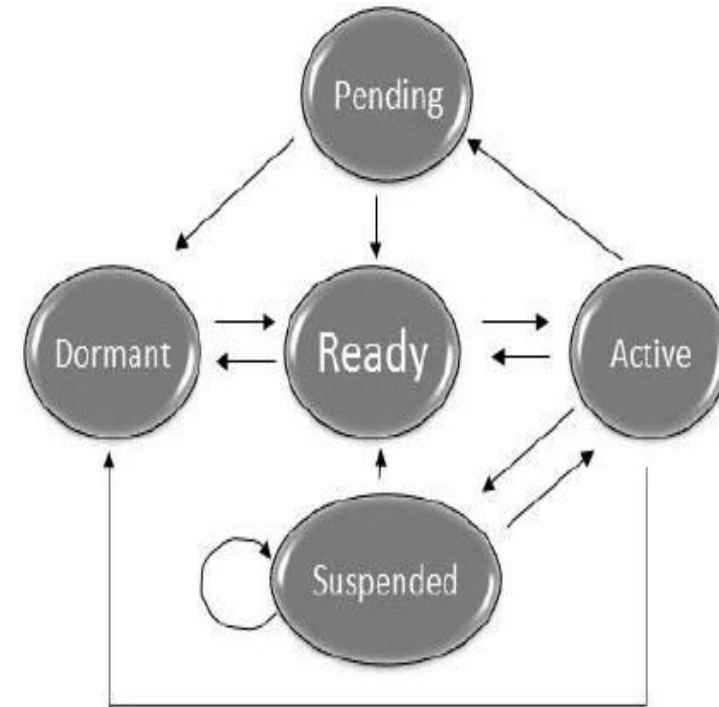


Fig:1.4: Use of RTOS for Time Management Application

- Fig: Use of RTOS for Time Management Application

Task states, Task table and data

- Dormant : Task doesn't require computer time
- Ready: Task is ready to go active state, waiting processor time
- Active: Task is running
- Suspended: Task put on hold temporarily
- Pending: Task waiting for resource.



Representation of Different Time Management Tasks Done by an RTOS

Task states, Task table and data

- During the execution of an application program, individual tasks are continuously changing from one state to another. However, only one task is in the running mode (i.e. given CPU control) at any point of the execution.
- In the process where CPU control is change from one task to another, context of the to-be-suspended task will be saved while context of the to-be-executed task will be retrieved, the process referred to as context switching.

Task states, Task table and data

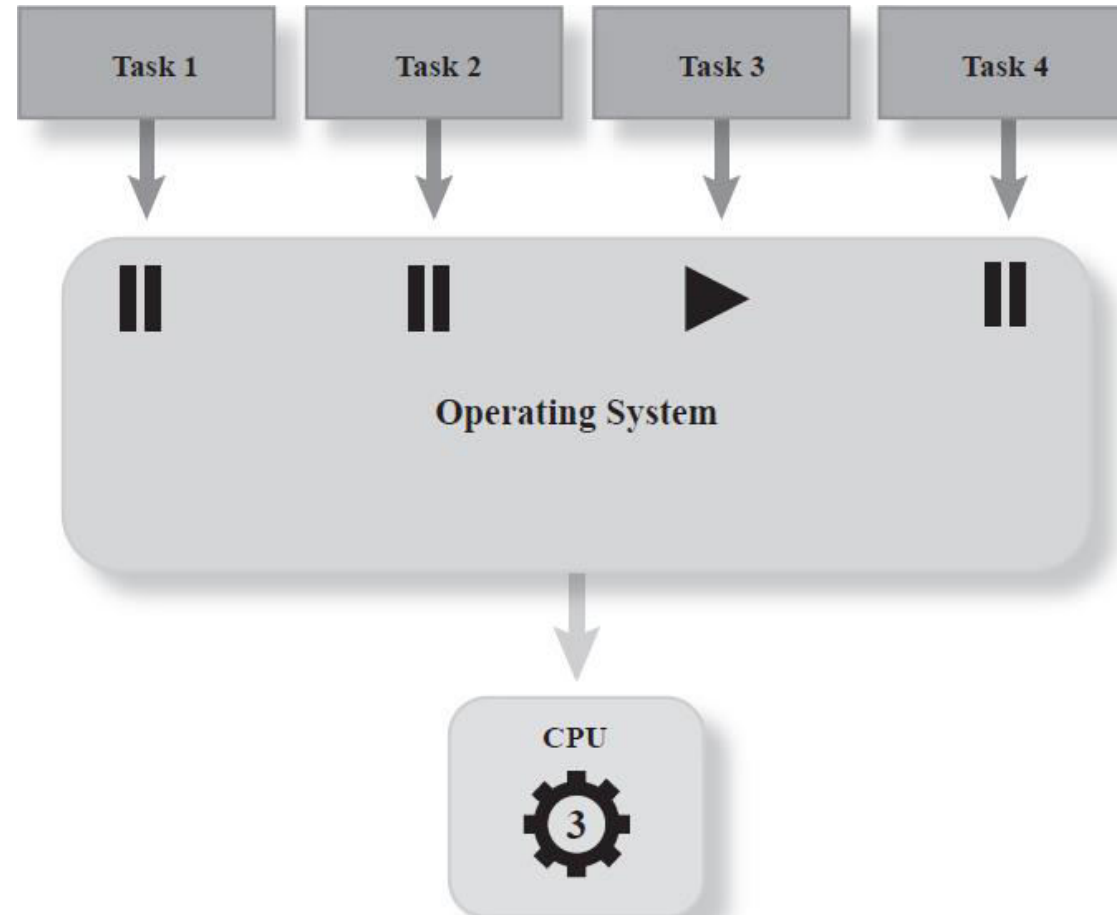
A task object is defined by the following set of components:

- Task Control block: Task uses TCBs to remember its context. TCBs are data structures residing in RAM, accessible only by RTOS
- Task Stack: These reside in RAM, accessible by stack pointer.
- Task Routine: Program code residing in ROM
- Scheduler: The scheduler keeps record of the state of each task and selects from among them that are ready to execute and allocates the CPU to one of them. Various scheduling algorithms are used in RTOS.
- Polled Loop: Sequentially determines if specific task requires time.

Multitasking operating systems

- **Definition** – Multitasking operating system provides the interface for executing the multiple program tasks by single user at a same time on the one computer system.
- **For example**, any editing task can be performed while other programs are executing concurrently. Other example, user can open Gmail and Power Point same time.
- **For example**, when you see someone in the car next to you eating a burrito, taking on his cell phone, and trying to drive at the same, that person is multitasking.
- (or) Multitasking Multitasking refers to term where multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. Operating system does the following activities related to multitasking.
- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- Operating System handles multitasking in the way that it can handle multiple operations / executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.
- A program that is loaded into memory and is executing is commonly referred to as a process.

Multitasking operating systems

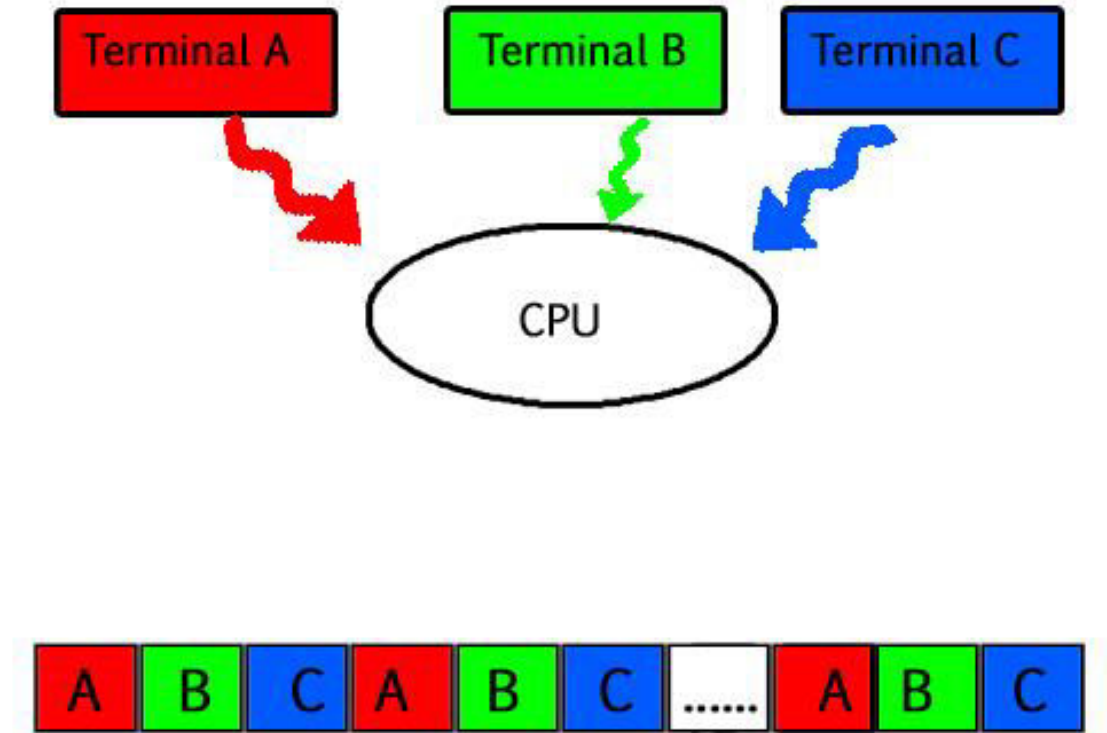


TERMS used in Multitasking operating systems

- **context switch:** a multitasking operating system shifting from one task to another; for example, after formatting a print job for one user, the computer might switch to resizing a graphic for another user.
- **cooperative multitasking:** an implementation of multitasking in which the operating system will not initiate a context switch while a process is running in order to allow the process to complete.
- **hardware interruption:** a device attached to a computer sending a message to the operating system to inform it that the device needs attention, thereby “interrupting” the other tasks that the operating system was performing.
- **multiprocessing:** the use of more than one central processing unit to handle system tasks; this requires an operating system capable of dividing tasks between multiple processors.
- **preemptive multitasking:** an implementation of multitasking in which the operating system will initiate a context switch while a process is running, usually on a schedule so that switches between tasks occur at precise time intervals.
- **time-sharing:** the use of a single computing resource by multiple users at the same time, made possible by the computer's ability to switch rapidly between users and their needs.

Multi tasking system's working

- In a time sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So each of them are assigned some time quantum for which they will execute e.g time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes for that quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.



Multi tasking system's working

- Thus the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.
- Here also basically a context switch is occurring but it is occurring so fast that the user is able to interact with each program separately while it is running. This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But actually only one process/ task is executing at a particular instant of time. In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time.
- In a more general sense, multitasking refers to having multiple programs, processes, tasks, threads running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e.g., CPU and Memory).
- As depicted in the above image, At any time the CPU is executing only one task while other tasks are waiting for their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task. i.e all the three tasks A, B and C are appearing to occur simultaneously because of time sharing.
- So for multitasking to take place, firstly there should be multiprogramming i.e. presence of multiple programs ready for execution. And secondly the concept of time sharing.

Types of Multitasking Operating System

- **True Multitasking:**
- True multitasking is the capable for executing and process multiple tasks concurrently without taking delay instead of switching tasks from one processor to other processor. It can perform couple of tasks in parallel with underlying the H/W or S/W.
- **Preemptive Multitasking**
- Preemptive multitasking is special task that is assigned to [computer operating system](#), in which it takes decision that how much time spent by one task before assigning other task for using the operating system. Operating system has control for completing this entire process, so it is known as “Preemptive”.
- **Cooperative Multitasking**
- Cooperative multitasking is known as “Non-Preemptive Multitasking”. Main goal of Cooperative multitasking is to run currently task, and to release the CPU to allow another task run. This task is performed by calling taskYIELD().Context-switch is executed when this function is called.

Advantages of Multitasking Operating System

- **Time Shareable**
 - In which, all tasks are allocated specific piece of time, so they do not need for waiting time for CPU.
- **Manage Several Users**
 - This [operating system](#) is more comfort for handling the multiple users concurrently, and several programs can run smoothly without degradation of system's performance.
- **Secured Memory**
 - Multitasking operating system has well defined memory management, because this operating system does not provide any types of permissions of unwanted programs to wasting the memory.
- **Great Virtual Memory**
 - Multitasking operating system contains the best virtual memory system. Due to virtual memory, any program do not need long waiting g time for completion their tasks, if this problem is occurred then those programs are transferred to virtual memory.

Advantages of Multitasking Operating System

- **Background Processing**
- Multitasking operating system creates the better environment to execute the background programs. These background programs are not [transparent](#) for normal users, but these programs help to run other programs smoothly such as firewall, antivirus software, and more.
- **Good Reliability**
- Multitasking operating system provides the several flexibilities for multiple users, and they are more satisfied to them. On which, every users can operate single or multiple programs with smoothly.
- **Use Multiple Programs**
- Users can operate multiple programs such as internet browser, PowerPoint, MS Excel, games, and other utilities concurrently.
- **Optimize Computer Resources**
- Multitasking operating system is able to handle smoothly multiple [computers' resources](#) such as RAM, [input/output devices](#), CPU, hard disk, and more.

Disadvantages of Multitasking Operating System

- **Memory Boundation**
- Computer can get slow performance, due to run multiple programs at a same time because main memory gets more load while loading multiple programs. CPU is not able to provide separate time for every program, and its response time gets increase. Main reason of occurring this problem is that it uses to less capacity RAM. So, for getting solution can be increased the RAM capacity.
- **Processor Boundation**
- Computer can run programs slowly due to slow speed of their processors, and its response time can increase while handling multiple programs. Need better processing power, to overcome this problem.
- **CPU Heat up**
- Multiple processors become busier at a time for executing any task in multitasking nature, So CPU produces more heat.

Examples of Multitasking Operating System

- There are some **examples of multi tasking OS** like as –
- Windows XP
- Windows Vista
- Windows 7
- Windows 8
- Windows 10
- Windows 2000
- IBM's OS/390
- Linux
- UNIX

Context switches

- Context switching is the mechanism that is changing the state of the processes.
- In computing, a **context switch** is the process of storing the state of a [process](#) or [thread](#), so that it can be restored and resume [execution](#) at a later point. This allows multiple processes to share a single [CPU](#), and is an essential feature of a [multitasking operating system](#).
- The precise meaning of the phrase “context switch” varies. In a multitasking context, it refers to the process of storing the system state for one task, so that task can be paused and another task resumed. A context switch can also occur as the result of an [interrupt](#), such as when a task needs to access [disk storage](#), freeing up CPU time for other tasks. Some operating systems also require a context switch to move between [user mode and kernel mode](#) tasks. The process of context switching can have a negative impact on system performance.

RTOS Context Switching

- An RTOS at its cores offers several basic primitives:
- A **scheduler** capable of **context switching** between different **tasks**. **Tasks** can usually be prioritized and at a bare minimum a **scheduler** is usually capable of alternating between **tasks** when new events arrive (i.e a new accelerometer sample is available) or when the **task yields** its slot.
- Very basic Operating System primitives (such as mutexes/semaphores and a way to pass messages between tasks)
- Configuration operations for sandboxing different code from one another by leveraging the privilege and access control features the hardware offers.
- **Schedulers usually come in two main varieties:**
- Preemptive - A context switch while a task is “running” if something more important comes up.
- Cooperative - A context switch will never occur while another task is “running”. A task must explicitly yield for another task to run. Tasks must “co-operate” for everyone to get a chance to run.
- When an RTOS scheduler decides a different task should be run than what is currently running, it will trigger a **context switch**. When switching from one task to another, the “state” of the current task needs to be preserved in some way. This includes information such as the execution state of the task (i.e blocked on a mutex, sleeping, etc) and the values of the active hardware registers.

RTOS Task Context Switching

- The [scheduling algorithms of real-time operating systems](#) (RTOS), stated that they can run tasks in such a way that leaves the impression of a multitasking behavior. This is achieved by giving the RTOS the capability to interrupt a currently executing task and to start executing another one. At some point in time, the interrupted task should resume its operation. When that occurs the microprocessor must be put in the same state as it was the last time the interrupted task was being executed. This is done using a mechanism called **context switching**.
- Each task uses a specific set of resources when it is executing. These include CPU registers, system status flags, access to memory (heap, stack), etc.. All these resources are what we call a **task state** (aka **task execution context**). **Context switching** is a process of saving the **task state** (with the intention for it to be restored at a later point in time) and switching it with another already saved task state.
- Task context switching guarantees that each task sees the CPU as its own. This mimics the behavior of real multitasking, where each task should have its own dedicated CPU.

Context Switching Basics

- Context switching is not a mechanism used only in real-time operating systems. Every microprocessor uses some form of context switching when an [exception](#) occurs and a service routine has to be executed. In most modern CPU architectures the exception context switching is usually handled partly by the hardware (some registers are automatically saved) and partly by the compiler-generated code.
- Task context switching in real-time operating systems is implemented as part of their source code. Although it is handled using software, context switching is hardware dependent, as the resources needed may differ from one microprocessor to another. This means that the code for task context switching must be ported for each CPU architecture.

Implementation Details

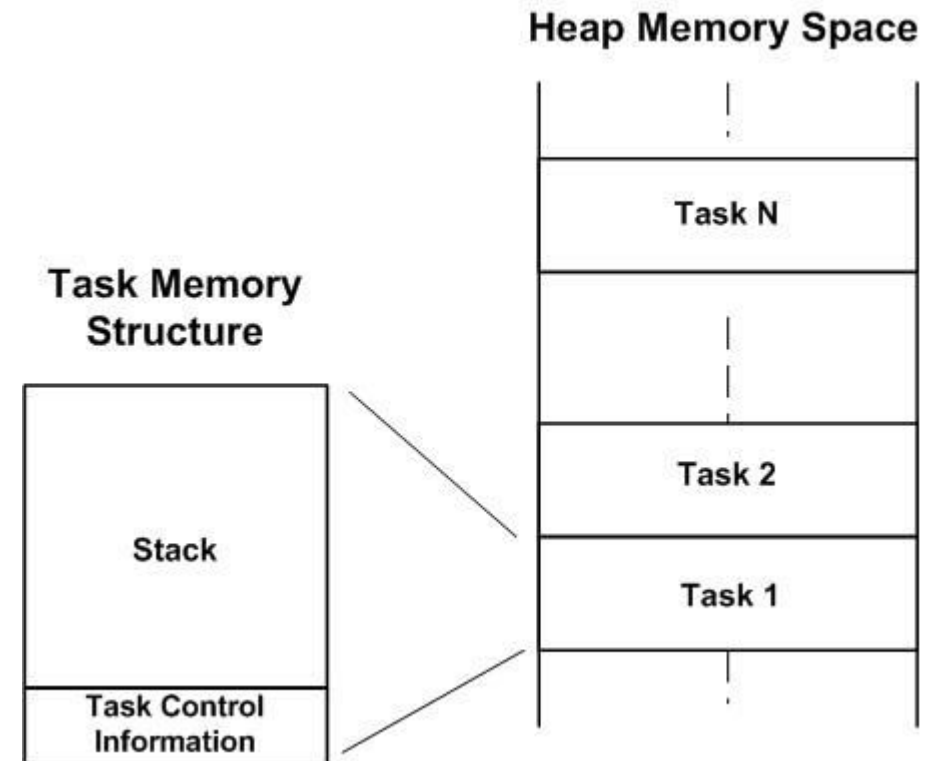
- Now analyze how task context switching can be implemented. As a start, we should make sure that each task, has its own private [stack](#).
- In addition to using it as a regular application stack, this private stack will also be used to store the task state (CPU registers, return address, stack pointer value, etc.). In the basic “bare-bone” applications we usually have only one stack for the whole program.
- The obvious question is how can we implement individual stacks for each task? This is accomplished by modifying the value of the stack pointer register.
- The stack is just a section from the volatile memory (RAM) that we “reserve” for stack operation. The location of this section is pointed by the stack pointer.
- The basic principle is that each task will have a specific area of the memory for its stack. The start address of this stack will be stored in a variable, so it can be loaded when the task is being activated.

Memory Allocation

- RTOS kernel objects such as tasks, semaphores, etc. can be allocated dynamically or statically (during compilation). For task dynamic allocation, the RTOS usually provides different schemes. For example:
- allocating the space for the task once, and never freeing it
- once a task has completed its operation the space allocated for it on the heap is freed
- The most suitable dynamic allocation scheme depends on the application complexity and the resource constraints of the embedded system.

Memory Allocation

- **Fig. Shows the RTOS task memory structure and placement in heap memory**
- Now let's see how much memory an RTOS task requires. We are not focusing on a specific RTOS distribution and we will try to cover the things that are common across all of them.
- Each created task should have a **task control information** memory area and a **stack** memory area. This is shown on the left side of Fig. 1. **Task control information** area has a fixed size and it may include:
 - task's name (a pointer to the C function implementing the task)
 - debug information
 - the size of the task's stack
 - top of the stack pointer (address)
 - task priority
- All tasks are placed in the [heap](#) memory.



Context Switching Flow

- As a final step let's analyze a context switching flow using an example involving two tasks – Task 1 and Task 2. Task 1 is currently running, while Task 2 which has a higher priority has just become ready to run. This situation will require a context switch, and the steps involved are the following: (from step 1 to step 9)

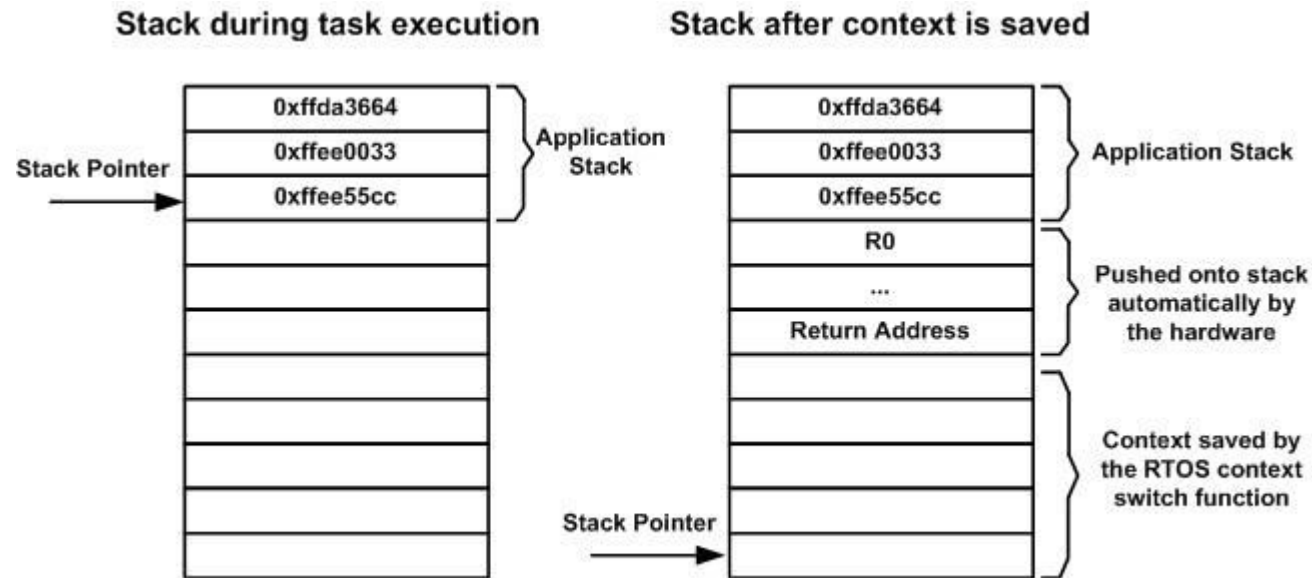


Fig. 2 Stack content before and after saving of task state (context)

Context Switching Flow

1. Task 1 is executing
2. RTOS tick exception is generated
3. The hardware automatically saves some registers onto the current task's stack. This depends on the CPU architecture.
 - ARM Cortex-M automatically saves R0-R3, R12, LR(R14), return address and xPSR.
4. An RTOS handler function for the tick exception stores any additional registers that are part of the current task state (see Fig. 2).
 - For ARM Cortex-M this handler function should save registers R4-R11, R14.
5. The stack pointer value (address of the last register pushed into the stack) is saved in the **task control information** area.
6. The handler checks if there is a higher priority task waiting to be run, in our case, this is Task 2. The CPU stack pointer register is loaded with the stack pointer value stored in Task 2's information control memory area.
7. We are still in the RTOS handler function for the tick exception, but now the stack pointer is pointing to the last entry in Task 2 stack area. The context that is saved by this handler upon entry (step 4) into the stack is now being restored. Note that the context for Task 1 was saved upon exception entry, but now we are restoring the context for Task 2.
8. When exiting the handler, the hardware automatically restores any data saved during step 3 of the flow. Note again that the hardware is restoring the context for Task 2, as the stack pointer is loaded with Task 2's stack address (step 6).
9. Now we are out of the exception handling routine. The program will continue regular execution but not on the task that was interrupted (Task 1), instead Task 2 will be executed. **The context switch is complete!**

All the “magic” of performing a task context switch is directly related to the manipulation of the stack pointer value for achieving an individual stack area for each task.

Kernels

- **RTOS Architecture – Kernel**
- **Kernel**
- RTOS kernel acts as an abstraction layer between the hardware and the applications. There are three broad categories of kernels
- **Monolithic kernel**
- Monolithic kernels are part of Unix-like operating systems like Linux, FreeBSD etc. A monolithic kernel is one single program that contains all of the code necessary to perform every kernel related task. It runs all basic system services (i.e. process and memory management, interrupt handling and I/O communication, file system, etc) and provides powerful abstractions of the underlying hardware. Amount of context switches and messaging involved are greatly reduced which makes it run faster than microkernel.

Kernels

- **Microkernel**

- It runs only basic process communication (messaging) and I/O control. It normally provides only the minimal services such as managing memory protection, Inter process communication and the process management. The other functions such as running the hardware processes are not handled directly by microkernels. Thus, micro kernels provide a smaller set of simple hardware abstractions. It is more stable than monolithic as the kernel is unaffected even if the servers failed (i.e. File System). Microkernels are part of the operating systems like AIX, BeOS, Mach, Mac OS X, MINIX, and QNX. Etc .

Kernels

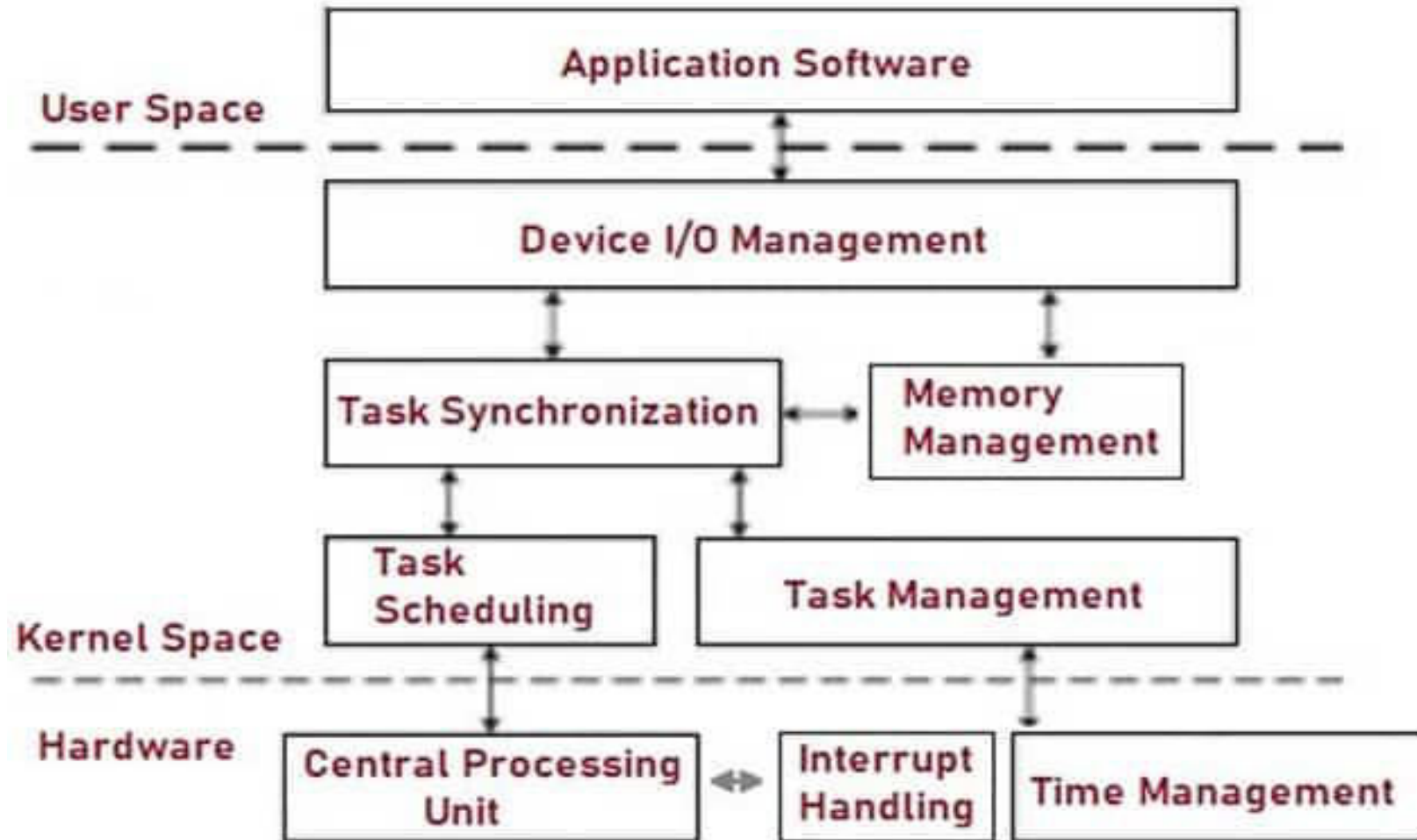
- **Hybrid Kernel**

- Hybrid kernels are extensions of microkernels with some properties of monolithic kernels. Hybrid kernels are similar to microkernels, except that they include additional code in kernel space so that such code can run more swiftly than it would were it in user space. These are part of the operating systems such as Microsoft Windows NT, 2000 and XP. DragonFly BSD, etc

- **Exokernel**

- Exokernels provides efficient control over hardware. It runs only services protecting the resources (i.e. tracking the ownership, guarding the usage, revoking access to resources, etc) by providing low-level interface for library operating systems and leaving the management to the application.

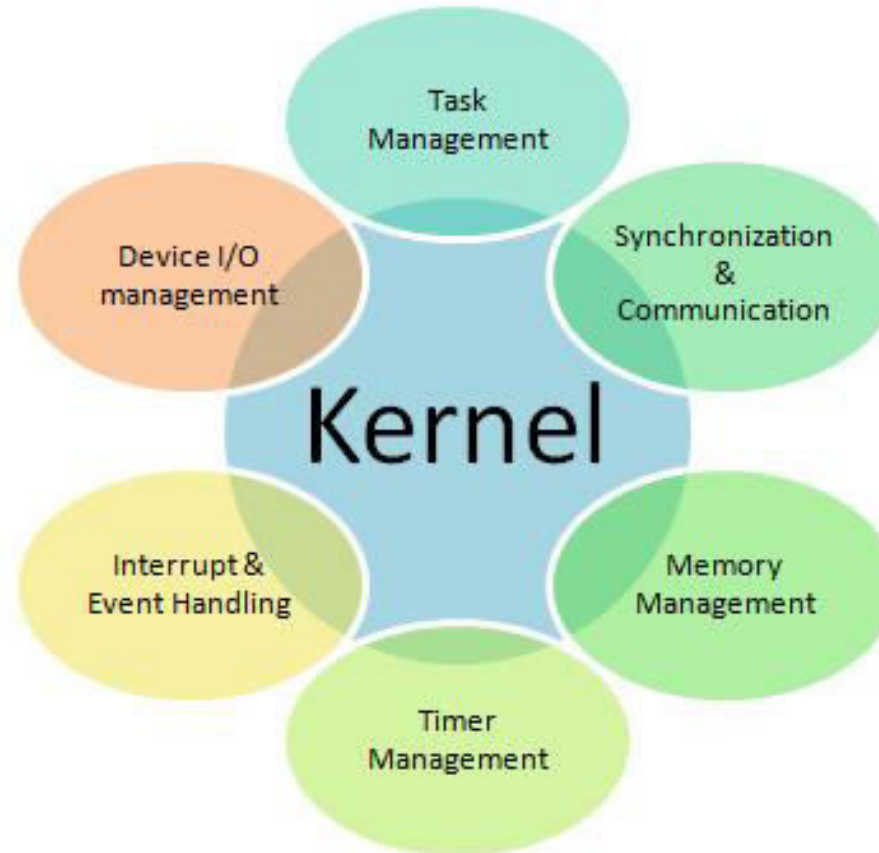
Relevance of Kernel in RTOS Architecture



Architecture of RTOS

Kernels

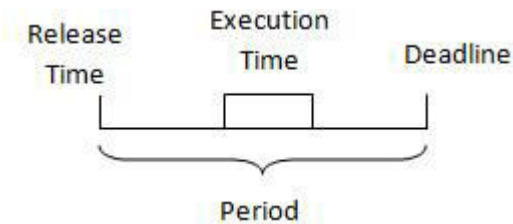
- Six types of common services are shown in the following figure below and explained in subsequent sections



A Figure Showing Common Services Offered By a RTOS System

RTOS Kernel Service- Task Management

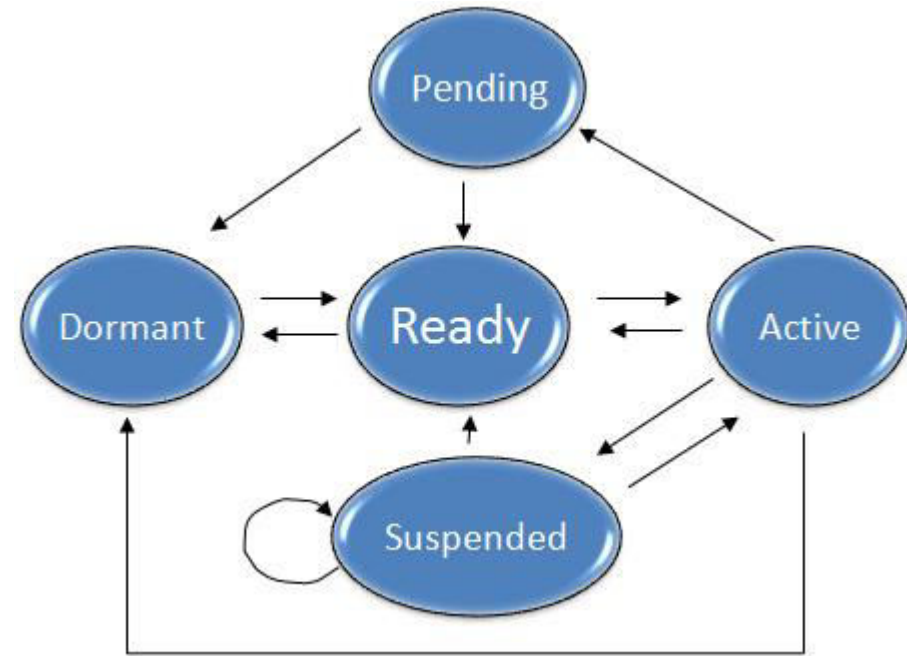
- **Task Management**
- **Task Object**
- In RTOS, The application is decomposed into small, schedulable, and sequential program units known as “Task”, a basic unit of execution and is governed by three time-critical properties; release time, deadline and execution time. Release time refers to the point in time from which the task can be executed. Deadline is the point in time by which the task must complete. Execution time denotes the time the task takes to execute.



A Diagram Illustrating Use of RTOS for Time Management Application

RTOS Kernel Service- Task Management

- Each task may exist in following states
- **Dormant** : Task doesn't require computer time
- **Ready**: Task is ready to go active state, waiting processor time
- **Active**: Task is running
- **Suspended**: Task put on hold temporarily
- **Pending**: Task waiting for resource.



A Figure Representing Different Time Management Tasks Done by an RTOS

RTOS Kernel Service- Task Management

- During the execution of an application program, individual tasks are continuously changing from one state to another. However, only one task is in the running mode (i.e. given CPU control) at any point of the execution. In the process where CPU control is change from one task to another, context of the to-be-suspended task will be saved while context of the to-be-executed task will be retrieved, the process referred to as context switching.
- A task object is defined by the following set of components:
- **Task Control block:** Task uses TCBs to remember its context. TCBs are data structures residing in RAM, accessible only by RTOS
- **Task Stack:** These reside in RAM, accessible by stack pointer.
- **Task Routine:** Program code residing in ROM

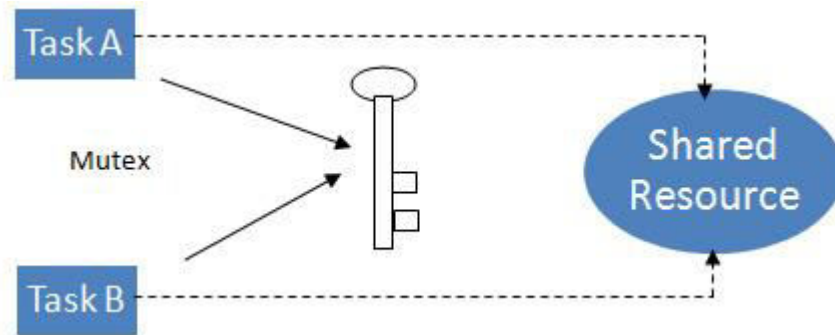
Task_ID
Task_State
Task_Priority
Task_Stack_Pointer
Task_Prog_Counter

RTOS Kernel Service- Synchronisation and communication

- Task Synchronisation & intertask communication serves to pass information amongst tasks.
- **Task Synchronisation**
- Synchronization is essential for tasks to share mutually exclusive resources (devices, buffers, etc) and/or allow multiple concurrent tasks to be executed (e.g. Task A needs a result from task B, so task A can only run till task B produces it).
- Task synchronization is achieved using two types of mechanisms:
- **Event Objects**
- Event objects are used when task synchronization is required without resource sharing. They allow one or more tasks to keep waiting for a specified event to occur. Event object can exist either in triggered or non-triggered state. Triggered state indicates resumption of the task.
- **Semaphores.**
- A semaphore has an associated resource count and a wait queue. The resource count indicates availability of resource. The wait queue manages the tasks waiting for resources from the semaphore. A semaphore functions like a key that define whether a task has the access to the resource. A task gets an access to the resource when it acquires the semaphore.

RTOS Kernel Service- Synchronisation and communication

- There are three types of semaphore:
 - Binary Semaphores
 - Counting Semaphores
 - **M**utually **E**xclusion(Mutex) Semaphores
- Semaphore functionality (Mutex) represented pictorially in the following figure



A Diagram Showing Architecture of Semaphore Functionality

RTOS Kernel Service

- **Memory Management**
 - Two types of memory managements are provided in RTOS – Stack and Heap.
 - Stack management is used during context switching for TCBs. Memory other than memory used for program code, program data and system stack is called heap memory and it is used for dynamic allocation of data space for tasks. Management of this memory is called heap management.
- **Timer Management**
 - Tasks need to be performed after scheduled durations. To keep track of the delays, timers- relative and absolute- are provided in RTOS.
- **Interrupt and event handling**
 - RTOS provides various functions for interrupt and event handling, viz., Defining interrupt handler, creation and deletion of ISR, referencing the state of an ISR, enabling and disabling of an interrupt, etc. It also restricts interrupts from occurring when modifying a data structure, minimise interrupt latencies due to disabling of interrupts when RTOS is performing critical operations, minimises interrupt response times.
- **Device I/O Management**
 - RTOS generally provides large number of APIs to support diverse hardware device drivers.

Task swapping methods

- The choice of scheduler algorithms can play an important part in the design of an embedded system and can dramatically affect the underlying design of the software. There are many different types of scheduler algorithm that can be used, each with either different characteristics or different approaches to solving the same problem of how to assign priorities to schedule tasks so that correct operation is assured.
- Time Slice
- Pre-emption
- *Co-operative multitasking*

Task swapping methods –Time Slice

- Time slicing has been previously mentioned in this chapter under the topic of multitasking and can be used within an embed-ded system where time critical operations are not essential. To be more accurate about its definition, it describes the task switching mechanism and not the algorithm behind it although its meaning has become synonymous with both.
- Time slicing works by making the task switching regular periodic points in time. This means that any task that needs to run next will have to wait until the current time slice is completed or until the current task suspends its operation. This technique can also be used as a scheduling method.

Task swapping methods –Time Slice

- The choice of which task to run next is determined by the scheduling algorithm and thus is nothing to do with the time slice mechanism itself. It just happens that many time slice-based systems use a round-robin or other fairness scheduler to distribute the time slices across all the tasks that need to run.
- For real-time systems where speed is of the essence, the time slice period plus the context switch time of the processor determines the context switch time of the system. With most time slice periods in the order of milliseconds, it is the dominant factor in the system response. While the time period can be reduced to improve the system context switch time, it will increase the number of task switches that will occur and this will reduce the efficiency of the system. The larger the number of switches, the less time there is available for processing.

Task swapping methods –Pre-emption

- The alternative to time slicing is to use pre-emption where a currently running task can be stopped and switched out — pre-empted — by a higher priority *active* task. The *active* qualifier is important as the example of pre-emption later in this section will show. The main difference is that the task switch does not need to wait for the end of a time slice and therefore the system context switch is now the same as the processor context switch.
- As an example of how pre-emption works, consider a system with two tasks A and B. A is a high priority task that acts as an ISR to service a peripheral and is activated by a processor interrupt from the peripheral. While it is not servicing the peripheral, the task remains dormant and stays in a suspended state. Task B is a low priority task that performs system housekeeping.

Task swapping methods-Pre-emption

- When the interrupt is recognised by the processor, the operating system will process it and activate task A. This task with its higher priority compared to task B will cause task B to be pre-empted and replaced by task A. Task A will continue processing until it has completed and then suspend itself. At this point, task B will context switch task A out because task A is no longer active.
- This can be done with a time slice mechanism provided the interrupt rate is less than the time slice rate. If it is higher, this can also be fine provided there is sufficient buffering available to store data without losing it while waiting for the next time slice point. The problem comes when the interrupt rate is higher or if there are multiple interrupts and associated tasks.

Task swapping methods-Pre-emption

- In this case, multiple tasks may compete for the same time slice point and the ability to run even though the total processing time needed to run all of them may be considerably less than the time provided within a single time slot. This can be solved by artificially creating more context switch points by getting each task to suspend after completion.
- This may offer only a partial solution because a higher priority task may still have to wait on a lower priority task to complete. With time slicing, the lower priority task cannot be pre-empted and therefore the higher priority task must wait for the end of the time slice or the lower priority task to complete. This is a form of priority inversion which is explained in more detail later.
- Most real-time operating systems support pre-emption in preference to time slicing although some can support both methodologies

Task swapping methods -*Co-operative multitasking*

- This is the mechanism behind Windows 3.1 and while not applicable to real-time operating systems for reasons which will become apparent, it has been included for reference.
- The idea of co-operative multitasking is that the tasks themselves co-operate between themselves to provide the illusion of multitasking. This is done by periodically allowing other tasks or applications the opportunity to execute.
- This requires programming within the application and the system can be destroyed by a single rogue program that hogs all the processing power. This method may be acceptable for a desktop personal computer but it is not reliable enough for most real-time embedded systems.

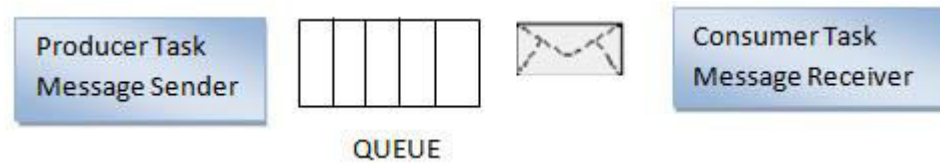
Scheduler algorithms

- Separate PPT

Inter process communication mechanism

- Intertask communication involves sharing of data among tasks through sharing of memory space, transmission of data, etc. Intertask communications is executed using following mechanisms
- Message queues – A message queue is an object used for intertask communication through which task send or receive messages placed in a shared memory. The queue may follow
 - 1) First In First Out (FIFO),
 - 2) Last in First Out(LIFO) or
 - 3) Priority (PRI) sequence.
- Usually, a message queue comprises of an associated queue control block (QCB), name, unique ID, memory buffers, queue length, maximum message length and one or more task waiting lists. A message queue with a length of 1 is commonly known as a mailbox.

Inter process communication mechanism



A Diagram Showing Flow of a Message Queue in a Mailbox

- Pipes – A pipe is an object that provide simple communication channel used for unstructured data exchange among tasks. A pipe does not store multiple messages but stream of bytes. Also, data flow from a pipe cannot be prioritized.
- Remote procedure call (RPC) – It permits distributed computing where task can invoke the execution of another task on a remote computer.

What is Inter Process Communication?

- **Inter process communication (IPC)** is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.
- It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.
- Since every single user request may result in multiple processes running in the operating system, the process may require to communicate with each other. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for a single program to use all of the IPC methods.

Approaches for Inter-Process Communication

Here, are few important methods for interprocess communication:



Inter-Process Communication

- **Pipes**
- Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.
- **Message Passing:**
- It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.
- IPC mechanism provides two operations:
 - Send (message)- message size fixed or variable
 - Received (message)

Inter-Process Communication

- **Message Queues:**
 - A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.
- **Direct Communication:**
 - In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

Inter-Process Communication

- **Indirect Communication:**
- Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.
- **Shared Memory:**
- Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.
- **FIFO:**
- Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.

Why IPC?

- Here, are the reasons for using the interprocess communication protocol for information sharing:
- It helps to speedup modularity
- Computational
- Privilege separation
- Convenience
- Helps operating system to communicate with each other and synchronize their actions.

Terms Used in IPC

- The following are a few important terms used in IPC:
- **Semaphores:** A semaphore is a signaling mechanism technique. This OS method either allows or disallows access to the resource, which depends on how it is set up.
- **Signals:** It is a method to communicate between multiple processes by way of signaling. The source process will send a signal which is recognized by number, and the destination process will handle it.

What is Like FIFOS and Unlike FIFOS

Like FIFOS	Unlike FIFOS
It follows FIFO method	Method to pull specific urgent messages before they reach the front
FIFO exists independently of both sending and receiving processes.	Always ready, so don't need to open or close.
Allows data transfer among unrelated processes.	Not have any synchronization problems between open & close.

Summary: Inter Process Communication

- Definition: Inter-process communication is used for exchanging data between multiple threads in one or more processes or programs.
- Pipe is widely used for communication between two related processes.
- Message passing is a mechanism for a process to communicate and synchronize.
- A message queue is a linked list of messages stored within the kernel
- Direct process is a type of inter-process communication process, should name each other explicitly.
- Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links.
- Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes.
- Inter Process Communication method helps to speedup modularity.
- A semaphore is a signaling mechanism technique.
- Signaling is a method to communicate between multiple processes by way of signaling.
- Like FIFO follows FIFO method whereas Unlike FIFO use method to pull specific urgent messages before they reach the front.

Memory communication

- The first thing that comes to mind as a mechanism for passing information between different tasks is using a shared memory location. It is important that the shared memory is protected either by a mutex or semaphore (see RTOS Mutex and Semaphore Basics).
- A very basic example of using shared memory location is a global variable. Although there is nothing stopping us from using such a variable, it is not recommended as there are far more sophisticated ways available as a means of communication between tasks.

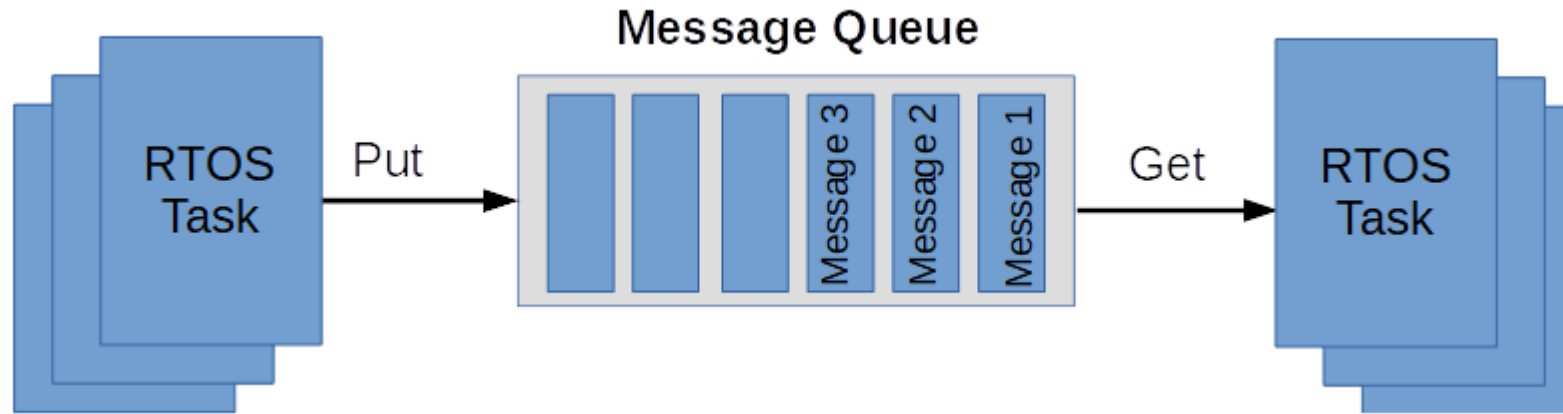
Message passing

- We can generally define two types of message passing:
- **Direct message passing** – The sender and the receiver of the messages are explicitly defined. As an example, the popular FreeRTOS has [Stream & Message Buffers](#) as primitives for direct message passing between a single writer task and a single reader task.
- **Indirect message passing** – The messages are placed in structures such as message queues or mailboxes and multiple tasks have read/write access.

Message Queues

- **Message queues** are data buffers with a finite amount of entries. Each entry can contain data of a certain size (e.g 32bits). Message queues can be used for passing data between tasks and between interrupt service routines and tasks. They are implemented as thread-safe FIFO buffers. Specific actions are defined in the RTOS in case a task tries to write to a full message queue or tries to read an empty one.
- The most common operations that can be performed on message queues are:
 - Create/delete a message queue
 - Get the number of messages currently stored in the queue
 - Put a message into the queue
 - Get a message from the queue

Message Queues



- *Example: Using a message queue for buffering data*
- *Let's have two tasks that are operating at different speeds. We can use a message queue as a buffer if one of the tasks produces a burst of data samples and the other task has to process each data sample individually at a fixed rate.*

Message Queues

- Mailboxes
- A mailbox can store a single data of specific size (e.g 32-bit variable) and can be implemented as a single-entry queue. A single mailbox can be accessed by many tasks. In some RTOS distributions, mailboxes can have more than one entry, which makes them very similar to what we described as a message queue in the previous chapter.
- Typical operation that can be performed on mailbox are:
 - Create/delete a mailbox
 - Write to a mailbox
 - Read a mailbox

Signals

- Signaling mechanisms are a basic form of communication. They indicate the occurrence of an event and can be used for synchronization purposes.
- The two additional signaling mechanisms that have wide use in real-time operating systems – **event flags** and **task flags**. As there are no strict naming conventions, you may find these mechanisms under slightly different names in different RTOS distributions, but the function they serve is pretty much the same.

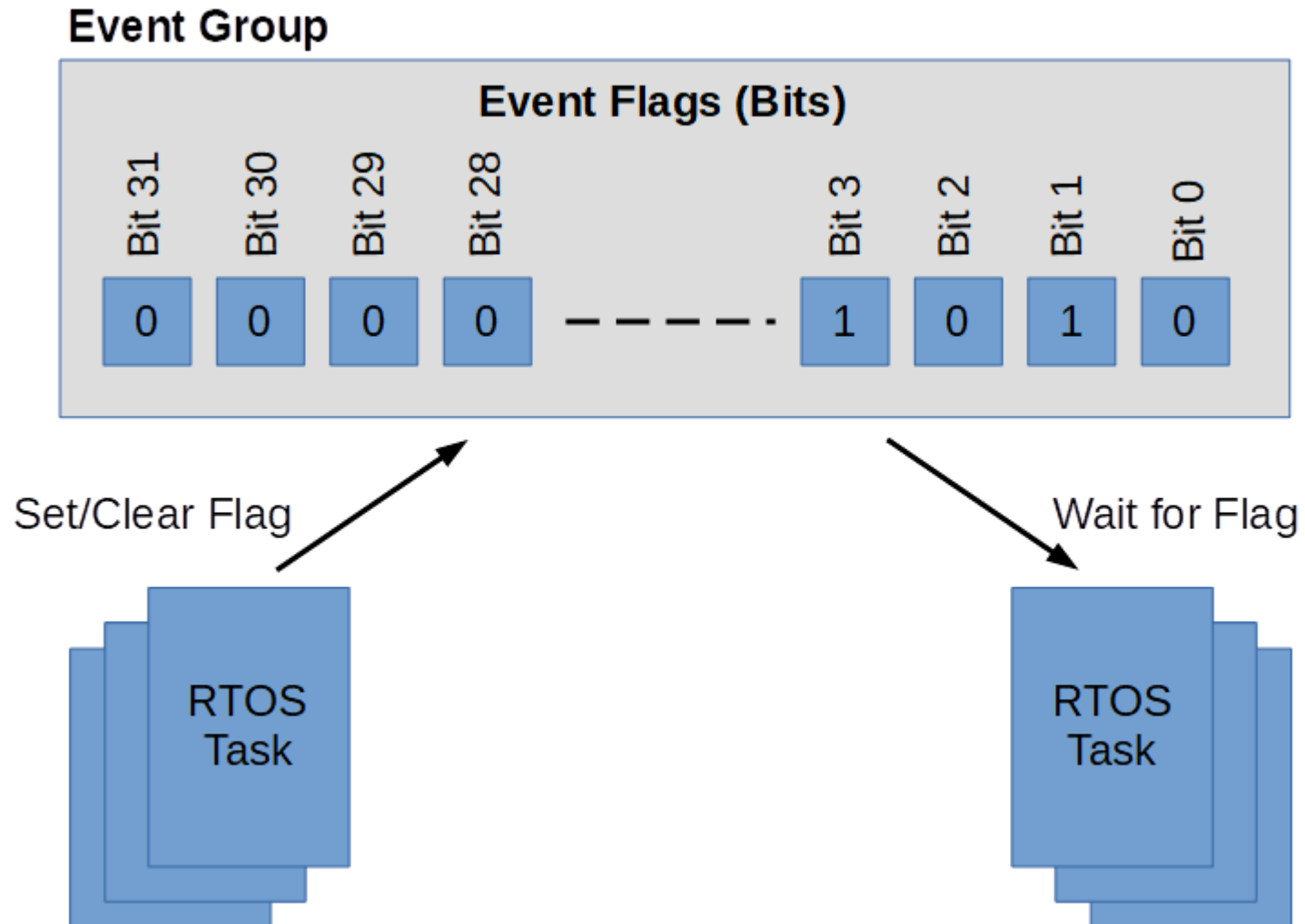
Signals

Event Flags

- **Event flags** are bits used to encode specific information. They are used for synchronizing tasks and communication. The grouping of individual event flags is called an **event group** or a **signal**.
- Events flags can be used by tasks and by interrupt service routines (ISR). A single event flag (or a group) can be accessed by many different tasks. The most common operations that can be performed on event flags are:
 - Create/delete event flags
 - Set/clear event flags
 - Read a flag's value
 - Wait on a flag to take a specific value
- An example API for using event flags can be found in the [CMSIS standard](#).

Signals

Event Flags



Signals

Task Flags

- **Task flags** are a special form of event flags. While event flags can be accessed by all tasks, the task flags are used for notifications to a single receiving task. The most common operations that can be performed on task flags are:
 - Set/clear flags of a specific task
 - Wait on a flag to take a specific value
- An example API for using task flags can be found in the [CMSIS](#) standard, the term used there is thread flags. In FreeRTOS these types of flags are defined as [task notifications](#).

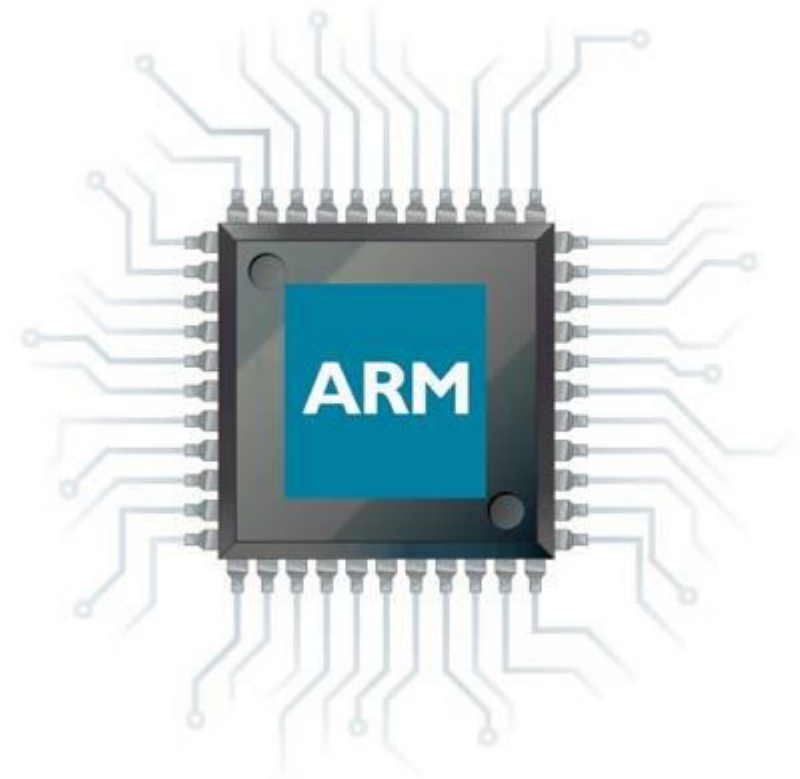
Signals

Task Flags



Overview of ARM Architecture

- ARM Holdings Inc. is a fabless semiconductor company that develops processors, system-on-chips, softwares etc. ARM was founded as Advanced RISC Machines in 1990 as RISC is the main CPU design strategy implemented in its processors.
- ARM is the world's leading provider of RISC based microprocessor solutions and other semiconductor IP's with more than 85 billion ARM based chips being shipped to date.
- Like other microprocessor companies like Intel, Freescale, Hitachi etc., ARM doesn't manufacture processors or other semiconductor devices but rather licenses the semiconductor cores as Intellectual Property (IP) to other semiconductor companies like ATMEL, Phillips (now NXP), Samsung etc. The important IP's of ARM include its low power, low cost, high efficient RISC microprocessors, system on chips and other peripherals.
- Apart from processors and IP Cores, ARM also provides comprehensive software development tools like Keil and DS-5 for developing complete systems based on ARM microcontrollers and system on chips.
- Today, ARM Processors are found in almost any domain like handheld devices, consumer electronics, robotics, automation, etc. Processors developed from ARM IP's are used in embedded systems like smart TV's, smart watches, smart phones, tablet computers, etc.

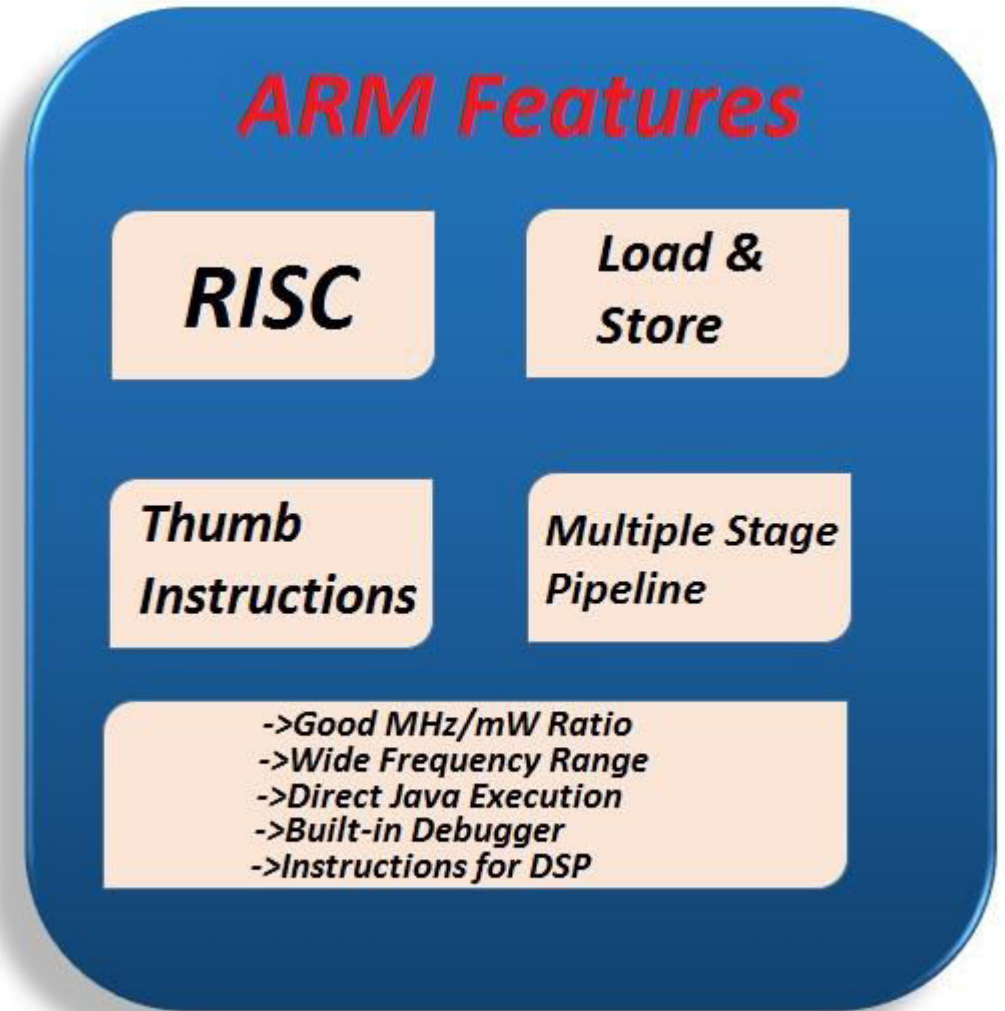


Features of ARM

- ARM Processors are based on reduced instruction set computing (RISC) architecture. But based on the requirements of the embedded systems, some amendments to the RISC architecture are made.
- ARM Processors follow Load and Store type architecture where the data processing is performed only on the contents of the registers rather than directly on the memory. The instructions for data processing on registers are different from that access the memory.
- The instruction set of ARM is uniform and fixed in length. 32-bit ARM Processors have two instruction sets: general 32-bit ARM Instruction Set and 16-bit Thumb Instruction Set.
- ARM supports multiple stages of pipeline to speed up the flow of instructions. In a simple three stage pipeline, the instructions follow three stages: fetch, decode and execute.

Features of ARM

- Some of the general features of ARM are listed here.
 - ARM Processors have a good speed of execution to power consumption ratio.
 - They have a wide range of clock frequency ranging from 1MHz to few GHz.
 - They support direct execution of Java bytecodes using ARM's Java Jazelle DBX.
 - ARM Processors have built in hardware for debugging.
 - Supports enhanced instructions for DSP operations.



ARM Processor Family

- ARM has several processors that are grouped into number of families based on the processor core they are implemented with.
- The architecture of ARM processors has continued to evolve with every family. Some of the famous ARM Processor families are ARM7, ARM9, ARM10 and ARM11.
- The following table shows some of the commonly found ARM Families along with their architectures.

ARM Family	Architecture
ARM7TDMI	ARMv4T
ARM9E	ARMv5TE
ARM11	ARMv6
Cortex-M	ARMv7-M
Cortex-R	ARMv7R
Cortex-A (32-bit)	ARMv7-A
Cortex-A (64-bit)	ARMv8-A

ARM Overview

- ARM follows the nomenclature shown in the below figure to describe the processor implementations. ARM Nomenclature



The letters or words after "ARM" are used to indicate the features of a processor.

- x – Family or series
- y – Memory Management/Protection Unit
- z – Cache
- T – 16 bit Thumb decoder
- D – JTAG Debugger
- M – Fast Multiplier
- I – Embedded In-circuit Emulator (ICE) Macrocell
- E – Enhanced Instructions for DSP (assumes TDMI)
- J – Jazelle (for accelerated JAVA execution)
- F – Vector Floating-point Unit
- S – Synthesizable Version

ARM Overview-Explanation of the features

- **Explanation of the features**
- *T – Thumb Instruction Set*
- ARM Processors support both the 32-bit ARM Instruction Set and 16-bit Thumb Instruction Set. The original 32-bit ARM Instructions consists of 32-bit opcodes which turns out to be a 4-byte binary pattern. 16-bit Thumb Instructions consists of 16-bit opcodes or 2-byte binary pattern to improve the code density.
- *D – JTAG Debug*
- JTAG is a serial protocol used by ARM to transfer the debug information between the processor and the test equipment.
- *M – Fast Multiplier*
- Older ARM Processors used a small and simple multiplier unit. This multiplier unit required more clock cycles to complete a single multiplication. With the introduction of Fast Multiplier unit, the clock cycles required for multiplication are significantly reduced and modern ARM Processors are capable of calculating a 32-bit product in a single cycle.
- *I – Embedded ICE Macrocell*
- ARM Processors have on-chip debug hardware that allows the processor to set breakpoints and watchpoints.

ARM Overview-Explanation of the features

- E – Enhanced Instructions
- ARM Processors with this mode will support the extended DSP Instruction Set for high performance DSP applications. With these extended DSP instructions, the DSP performance of the ARM Processors can be increased without high clock frequencies.
- J – Jazelle
- ARM Processors with Jazelle Technology can be used in accelerated execution of Java bytecodes. Jazelle DBX or Direct Bytecode eXecution is used in mobile phones and other consumer devices for high performance Java execution without affecting memory or battery.
- F – Vector Floating-point Unit
- The Floating Point Architecture in ARM Processors provide execution of floating point arithmetic operations. The Dynamic Range and Precision offered by the Floating Point Architecture in ARM Processors are used in many real time applications in the industrial and automotive areas.
- S – Synthesizable
- The ARM Processor Core is available as source code. This software core can be compiled into a format that can be easily understood by the EDA Tools. Using the processor source code, it is possible to modify the architecture of the ARM Processor.
- An example in ARM7 family of processors is the ARM7TDMI-S architecture based LPC2148 Processor.

ARM Processors

- ARM Processors can be divided into ARM Classic Processors, ARM Embedded Processors and ARM Application Processors.

Classic Processors

Embedded Processors

Application Processors

ARM Processors

- ARM Classic processors include ARM7, ARM9 and ARM11 families and ARM7TMDI is still the highest shipping 32-bit processor. ARM7 based processors are still used in many small and simple 32-bit devices.

Classic Processors

ARM11

ARM9

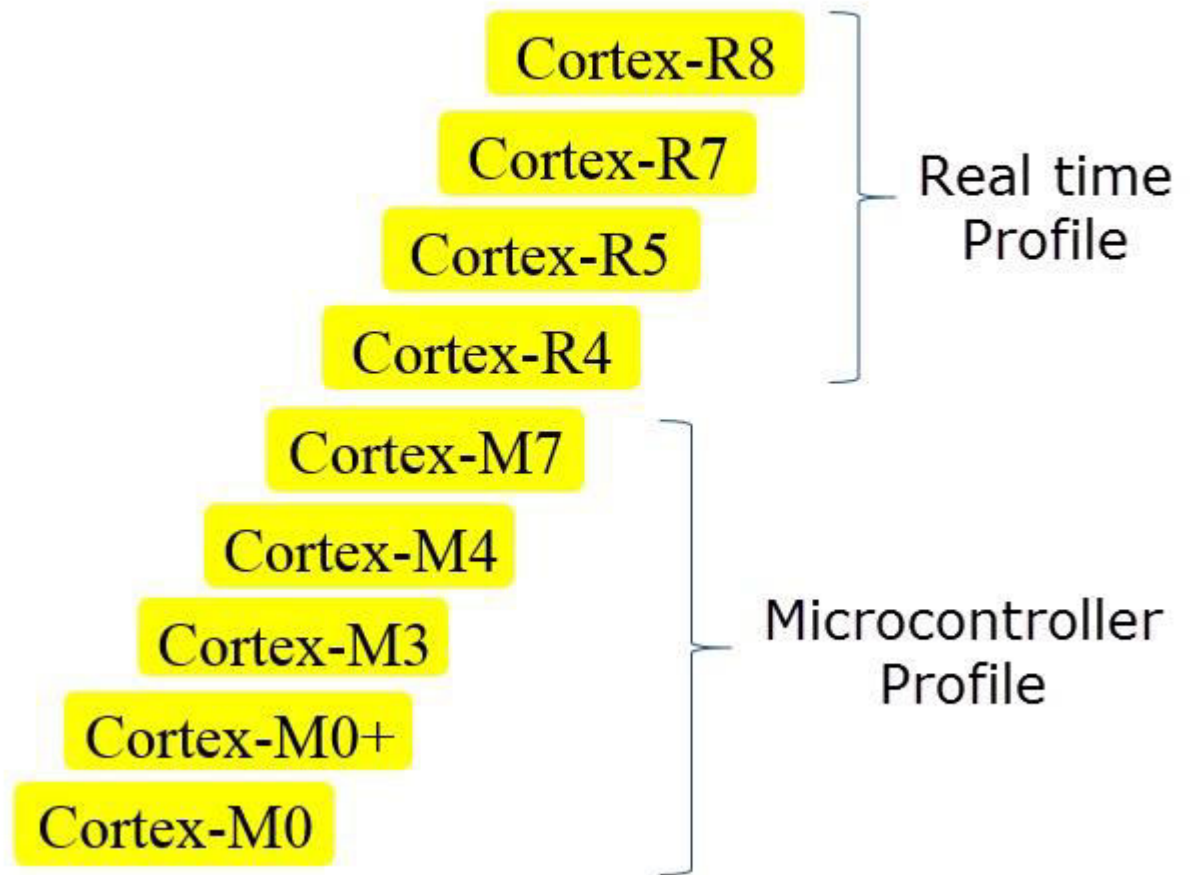
ARM7

Even though ARM7 or other classic ARM Processors can be used for small scale embedded systems, newer embedded systems are built using the advanced ARM embedded processors or the Cortex-M processors and Cortex-R Processors.

ARM Processors

- ARM Cortex-M Processors have a Microcontroller profile while the Cortex-R Processors have a Real time profile.
- ARM Cortex-M Processors are energy efficient, simple to implement and are mainly developed for advanced embedded applications. ARM Cortex-M Processors are further divided into several processor cores like Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7.

Embedded Processors



ARM Processors

- ARM Cortex-R Series of processors provide solution for real time embedded systems. They provide high reliability, high fault tolerance and real time responses. Cortex-R series of processors are used in systems where high performance is required and timing deadlines are important.
- The Cortex-R family includes the processor cores like Cortex-R4, Cortex-R5, Cortex-R7 and Cortex-R8.
- ARM Cortex-A Series of processors are the highest performance processors from ARM. They are used in powerful mobile devices, compelling technology products like network devices, consumer appliances, automation systems, automobiles and other embedded systems.

ARM Processors

Application Processors

High Performance	High Efficiency	Ultra-High Efficiency
Cortex-A73	Cortex-A53	Cortex-A35
Cortex-A72	Cortex-A9	Cortex-A32
Cortex-A57	Cortex-A8	Cortex-A7
Cortex-A17		Cortex-A5

- The Cortex-A Processors are again divided into high performance, high efficiency and ultra-high efficiency type processors. Each sub division has several types of processor cores.

Programmer's model and Development Tools

- Separate PPT

References

- <https://www.guru99.com/real-time-operating-system.html>
- <https://www.highintegritysystems.com/rtos/what-is-an-rtos/>
- <https://electricalfundablog.com/rtos-real-time-operating-system/>
- <https://www.engineersgarage.com/rtos-real-time-operating-system/>
- <https://digitalthinkerhelp.com/what-is-multitasking-operating-system-with-their-examples-types/>
- <https://www.geeksforgeeks.org/difference-between-multitasking-multithreading-and-multiprocessing/>
- [https://science.jrank.org/computer-science/Multitasking Operating Systems.html](https://science.jrank.org/computer-science/Multitasking_Operating_Systems.html)
- <https://open4tech.com/rtos-task-context-switching/>
- <https://interrupt.memfault.com/blog/cortex-m-rtos-context-switching>
- <https://www.electronicshub.org/arm-introduction/>
- <https://www.watelectronics.com/arm-processor-architecture-working/>
- <https://www.geeksforgeeks.org/arm-processor-and-its-features/>
- [https://www.cs.ccu.edu.tw/~pahsiung/courses/ese/notes/ESD 04 ARM Tools.pdf](https://www.cs.ccu.edu.tw/~pahsiung/courses/ese/notes/ESD_04_ARM_Tools.pdf)

References

- [https://ocw.aoc.ntua.gr/modules/document/file.php/ECE102/%CE%A3%CE%B7%CE%BC%CE%B5%CE%B9%CF%8E%CF%83%CE%B5%CE%B9%CF%82%20%CE%9C%CE%B1%CE%B8%CE%AE%CE%BC%CE%B1%CF%84%CE%BF%CF%82/ARM Programmer s Model.pdf](https://ocw.aoc.ntua.gr/modules/document/file.php/ECE102/%CE%A3%CE%B7%CE%BC%CE%B5%CE%B9%CF%8E%CF%83%CE%B5%CE%B9%CF%82%20%CE%9C%CE%B1%CE%B8%CE%AE%CE%BC%CE%B1%CF%84%CE%BF%CF%82/ARM%20Programmer%20s%20Model.pdf)
- <https://www.eetimes.com/intro-to-real-time-operating-systems/>
- <https://www.slideshare.net/phzope75/unit-4-rev>
- <https://www.guru99.com/inter-process-communication-ipc.html>
- <https://open4tech.com/communication-between-rtos-tasks/>

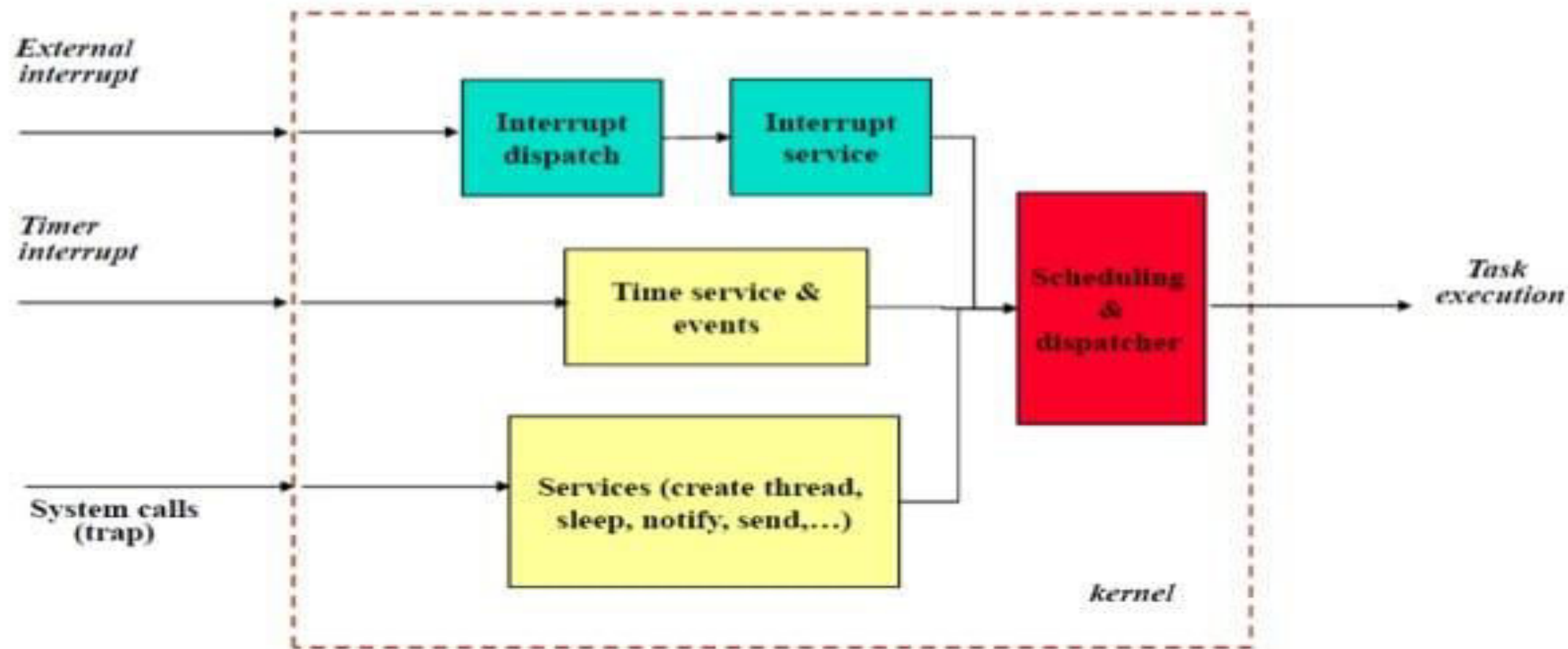
Unit-2
Scheduling Algorithms

Process scheduling in embedded systems

Research issues:

- low overhead memory protection,
- temporal protection of computing resources
- RTOSes for on-chip multiprocessors
- quality of service (QoS) control (besides real-time constraints)

Process Management Services



Process management - Three key requirements:

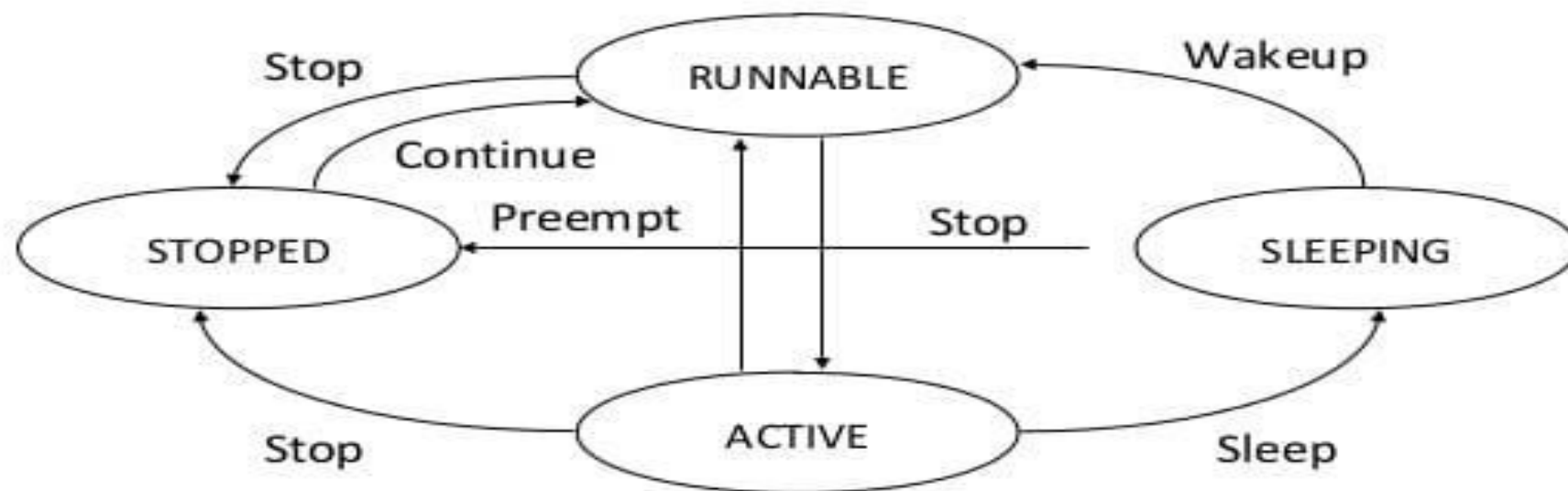
- 1. The timing behavior of the OS must be *predictable*** - services of the OS: Upper bound on the execution time!
- 2. OS must *manage the timing and scheduling***
OS possibly has to be aware of task deadlines;
(unless scheduling is done off-line).
- 3. The OS must be *fast***

Main Functionality of RTOS-Kernels

► *Process management:*

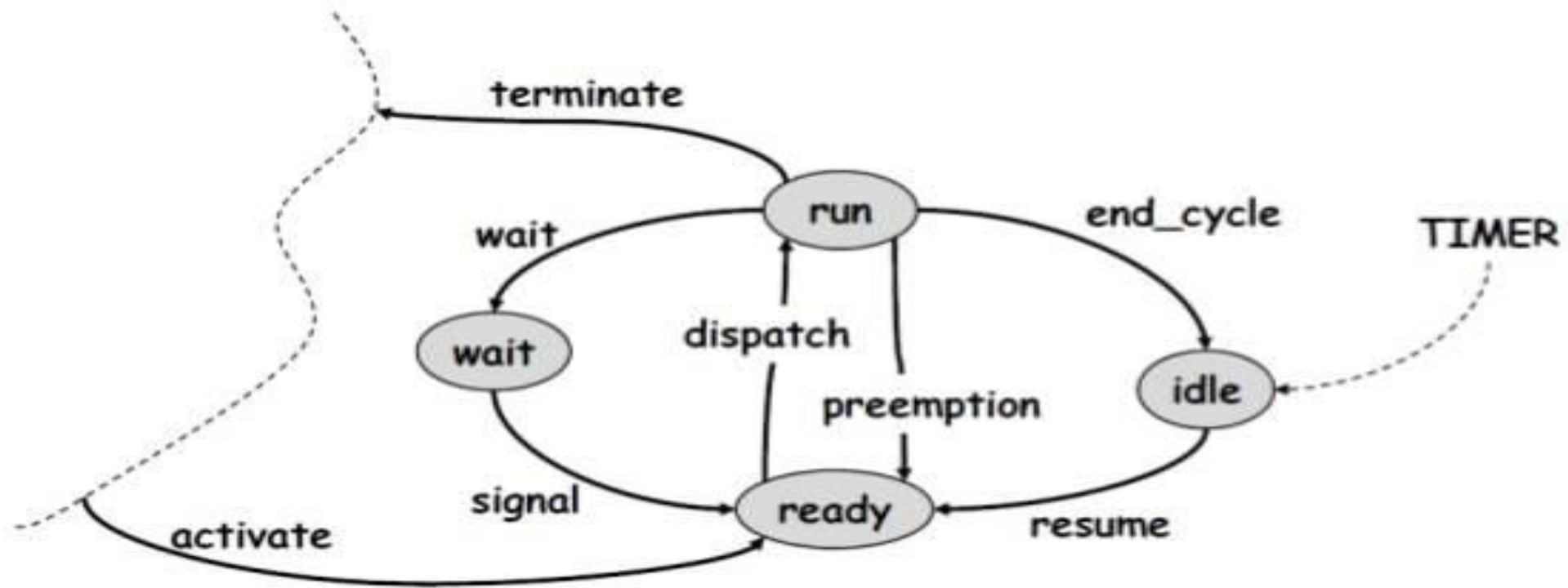
- Execution of *quasi-parallel tasks* on a processor using processes or threads (lightweight process) by
 - maintaining process states, process queuing,
 - preemptive tasks (fast context switching) and quick interrupt handling
- CPU *scheduling* (guaranteeing deadlines, minimizing process waiting times, fairness in granting resources such as computing power)
- Process *synchronization* (critical sections, semaphores, monitors, mutual exclusion)
- Inter-process *communication* (buffering)
- Support of a *real-time clock* as an internal time reference

Scheduling States: Simplified View of **Thread State Transitions**



Process States

► *Minimal Set of Process States:*



Process states

- ▶ *Run:*

- A task enters this state as it starts executing on the processor

- ▶ *Ready:*

- State of those tasks that are ready to execute but cannot be executed because the processor is assigned to another task.

- ▶ *Wait:*

- A task enters this state when it executes a synchronization primitive to wait for an event, e.g. a wait primitive on a semaphore. In this case, the task is inserted in a queue associated with the semaphore. The task at the head is resumed when the semaphore is unlocked by a signal primitive.

- ▶ *Idle:*

- A periodic job enters this state when it completes its execution and has to wait for the beginning of the next period.

Hard and Soft Real Time Systems

(Operational Definition)

- Hard Real Time System
 - Validation by provably correct procedures or extensive simulation that the system always meets the timings constraints
- Soft Real Time System
 - Demonstration of jobs meeting some statistical constraints suffices.
- Example – Multimedia System
 - 25 frames per second on an average

Real-Time System continue

- Soft RTS: meet timing constraints most of the time, **it is not necessary that every time constraint be met.** Some deadline miss is tolerated. (**tolerance is there**)
- Hard RTS: meet all time constraints exactly, Every resource management system must work in the correct order to meet time constraints. **No deadline miss is allowed.**
(**had when a print documentation was not available on time ,
teared the paper at mets**)

Role of an OS in Real Time Systems

1. Standalone Applications

- Often no OS involved
- Micro controller based Embedded Systems
- Some Real Time Applications are huge & complex
 - Multiple threads
 - Complicated Synchronization Requirements
 - Filesystem / Network / Windowing support
 - OS primitives reduce the software design time

Tasks Categories

- **Invocation = triggered operations**
 - Periodic (time-triggered)
 - Aperiodic (event-triggered)
- **Creation**
 - Static
 - Dynamic
- **Multi-Tasking System**
 - **Preemptive**: higher-priority process taking control of the processor from a lower-priority
 - **Non-Preemptive** : Each task can control the CPU for as long as it needs it.

Features of RTOS's

1. **Scheduling.**
2. **Resource Allocation.**
3. **Interrupt Handling.**
4. **Other issues like kernel size.**

1. Scheduling in RTOS

- More information about the tasks are known
 - No of tasks
 - Resource Requirements
 - Release Time
 - Execution time
 - Deadlines
- Being a more deterministic system better scheduling algorithms can be devised.

Why we need scheduling ?!

- each computation (task) we want to execute needs resources
- resources: processor, memory segments, communication, I/O devices etc.)
- the computation must be executed in particular order (relative to each other and/or relative to time)
- the possible ordering is either completely or statistically a priori known (described)
- scheduling: assignment of processor to computations;
- allocation: assignment of other resources to computations;

Scheduling Process

- **Scheduling** is the process of deciding which task should be executed at any point in time based on a predefined algorithm. The logic for the scheduling is implemented in a functional unit called the scheduler. The scheduling process is not present only in RTOS, it can be found in one form or another even in simple “bare-bone” applications.
- Different RTOS distributions may support a variety of scheduling algorithms. It is important that we choose the algorithm before the development of the user application starts. Like many things in the engineering field, there is not a universal algorithm that is suitable for every use case. There are always trade-offs. In this case, they are mainly related to speed (response times), implementation complexity, etc. The chosen algorithm should always enable the timing requirements of the tasks to be met.

Categories of Scheduling Algorithms

- **Offline Scheduling Algorithm**

- Offline scheduling algorithm selects a task to execute with reference to a predetermined schedule, which repeats itself after specific interval of time. For example, if we have three tasks T_a , T_b and T_c then T_a will always execute first, then T_b and after that T_c respectively.

- **Online Scheduling Algorithm**

- In Online scheduling a task executes with respect to its priority, which is determined in real time according to specific rule and priorities of tasks may change during execution. The online line scheduling algorithm has two types. They are more flexible because they can change the priority of tasks on run time according to the utilization factor of tasks.

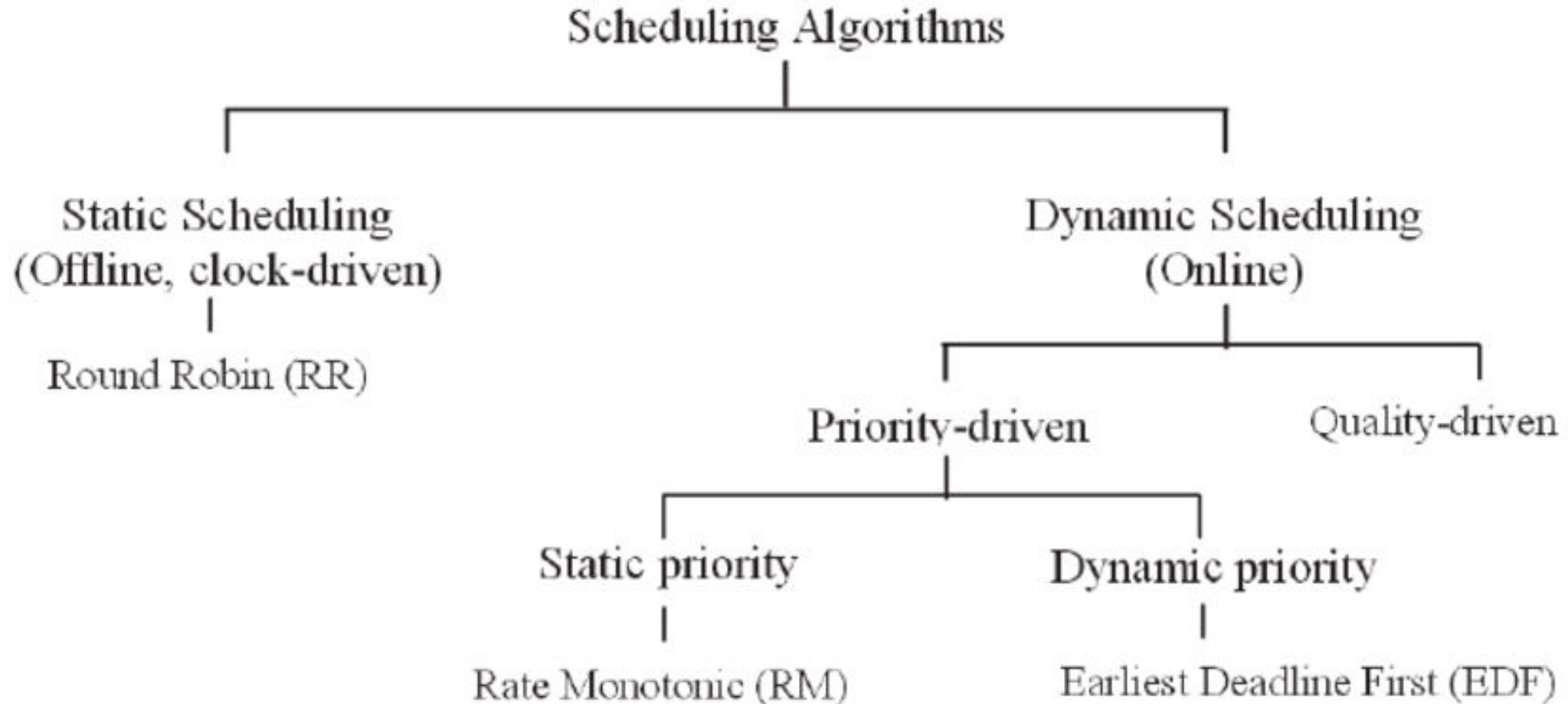
- **Fixed priority algorithms**

- In fixed priority if the k th job of a task T_1 has higher priority than the k th job of task T_2 according to some specified scheduling event, then every job of T_1 will always execute first then the job of T_2 i.e. on next occurrence priority does not change. More formally, if job $J(1,K)$ of task T_1 has higher priority than $J(2,K)$ of task T_2 then $J(1,K+1)$ will always has higher priority than of $J(2,K+1)$. One of best example of fixed priority algorithm is rate monotonic scheduling algorithm.

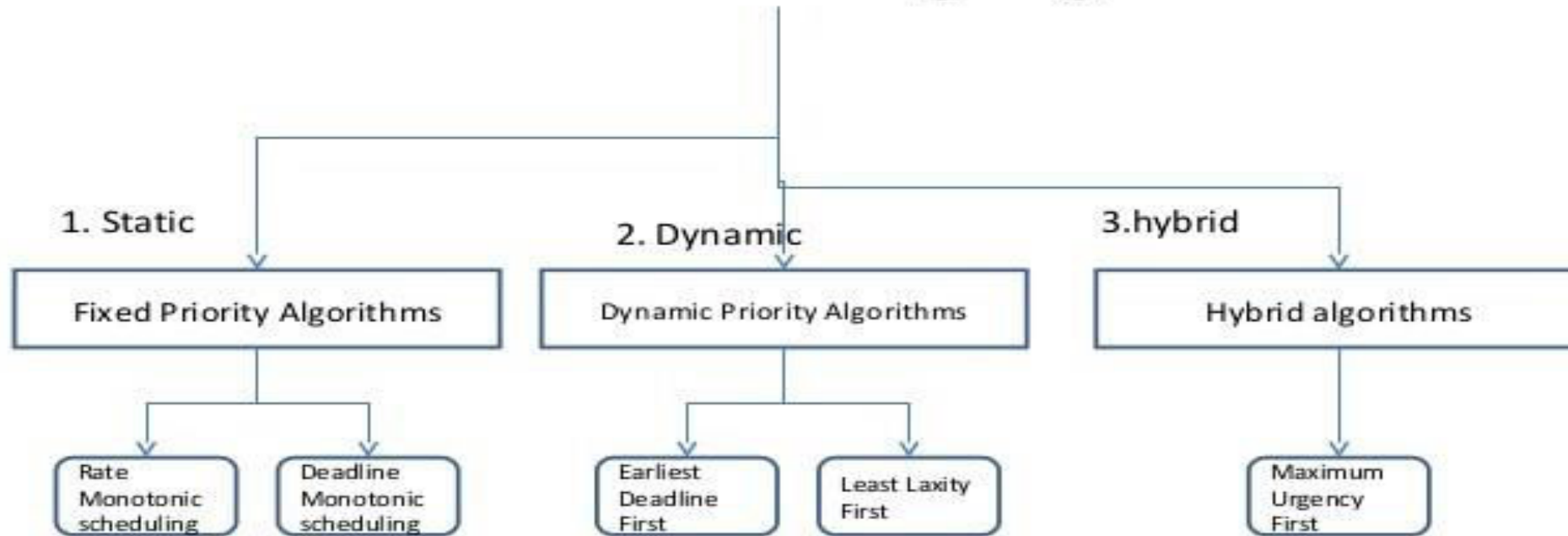
- **Dynamic priority algorithms**

- In dynamic priority algorithm, different jobs of a task may have different priority on next occurrence, it may be higher or it may be lower than the other tasks. One example of a dynamic priority algorithm is the earliest deadline first algorithm.

Scheduling Algorithms



Real-Time Scheduling Algorithms



Scheduling Algorithms

Static Vs. Dynamic

Static Scheduling :

- All Scheduling decisions at compile time.
- Temporal tasks structure fixed. Precedence and mutual exclusion satisfied by the schedule (implicit synchronization).
- One Solution is sufficient.
- Any solution is a sufficient schedulability test.

Benefits :

- Simplicity

Dynamic Scheduling :

- All Scheduling decisions at run time.
- Based upon set of ready tasks.
- Mutual Exclusion and synchronization enforced by explicit synchronization constructs.

Benefits :

- Flexibility.
- Only actually used resources are claimed.

Disadvantages :

- Guarantees difficult to support.
- Computational resources required for scheduling.

Scheduling Algorithms

Preemptive Vs. Non-Preemptive

Preemptive Scheduling :

- Event Driven
- Each event causes interruption of running tasks.
- Choice of running tasks reconsidered after each interruption.

Benefits :

- Can minimize response time to events.

Disadvantages :

- Requires considerable computational resources for scheduling.

Non-Preemptive Scheduling :

- Tasks remain active till completion.
 - Scheduling decisions only made after task completion.

Benefits :

- Reasonable when task execution times \approx task switching times.
- Less computational resources needed for scheduling.

Disadvantages :

- Can lead to starvation (not met the deadline). Especially for those real time tasks (or high priority tasks).

Scheduling Algorithms in RTOS

- 1. Clock Driven Scheduling**
- 2. Weighted Round Robin Scheduling**
- 3. Priority Scheduling (Greedy/List/Event Driven)**

Scheduling Algorithms in RTOS

1. Clock Driven Scheduling

- **All parameters about jobs (release time/execution time/deadline) known in advance.**
- **Schedule can be computed offline or at some regular time instances.**
- **Minimal runtime overhead.**
- **Not suitable for many applications.**

2. Weighted Round Robin Scheduling

- **Jobs scheduled in FIFO manner.**
- **Time quantum given to jobs is proportional to it's weight.**
- **Example use: High speed switching network**
 - **QoS guarantee.**
- **Not suitable for precedence constrained jobs.**
- **Job A can run only after Job B. No point in giving time quantum to Job B before Job A.**

Scheduling Algorithms in RTOS

3. Priority Scheduling (Greedy/List/Event Driven)

- Processor never left idle when there are ready tasks.
- Processor allocated to processes according to priorities.
- Priorities
 - Static (at design time)
 - Dynamic (at Run time)

Priority Scheduling

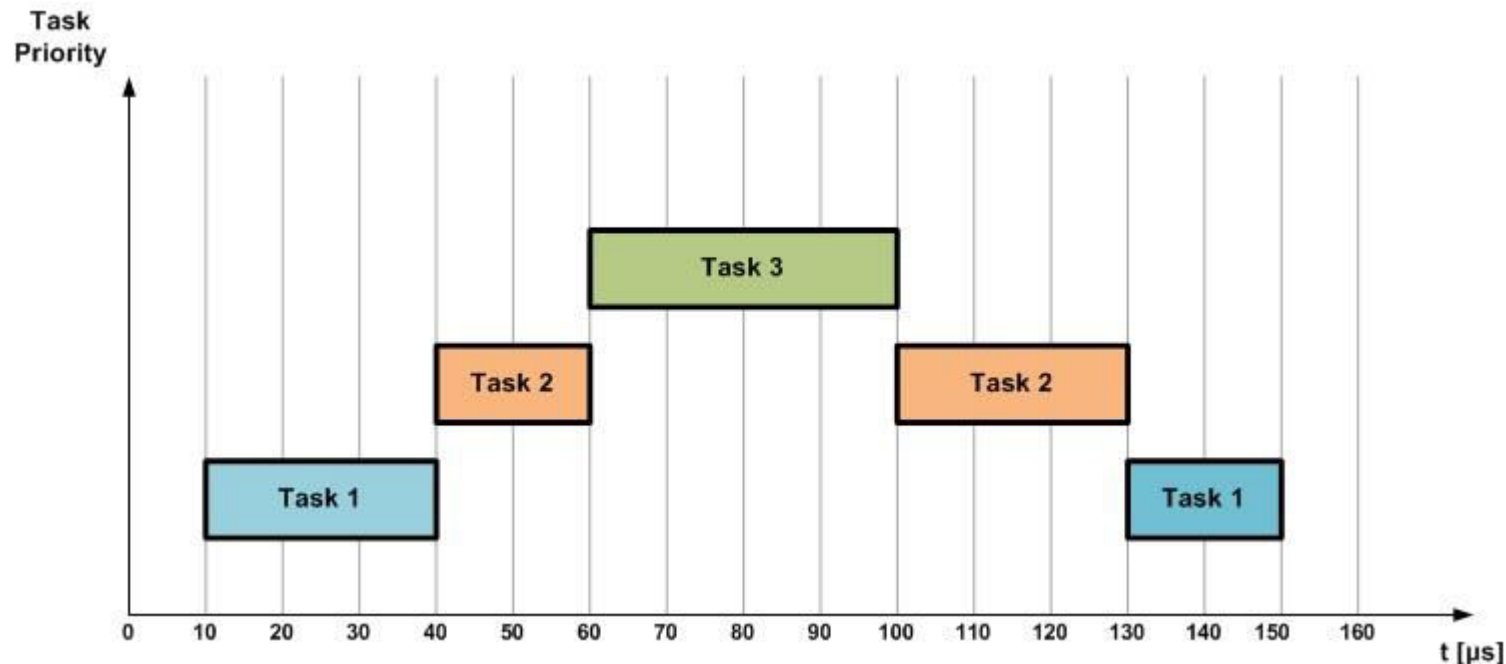
- Earliest Deadline First (EDF)
 - Process with earliest deadline given highest priority
- Least Slack Time First (LSF)
 - Slack =relative deadline-execution left
- Rate Monotonic Scheduling(RMS)
 - For periodic tasks
 - Tasks priority inversely proportional to it's period

Types of Scheduling Algorithms

- 1. Pre-Emption Scheduling Algorithm
- 2. Non-Preemption Scheduling Algorithms

Preemptive Scheduling

- **Preemptive scheduling** allows the interruption of a currently running task, so another one with more “urgent” status can be run.
- The interrupted task is involuntarily moved by the scheduler from running state to ready state. This dynamic switching between tasks that this algorithm employs is, in fact, a form of multitasking. It requires assigning a priority level for each task.
- A running task can be interrupted if a task with a higher priority enters the queue.



Preemptive Scheduling

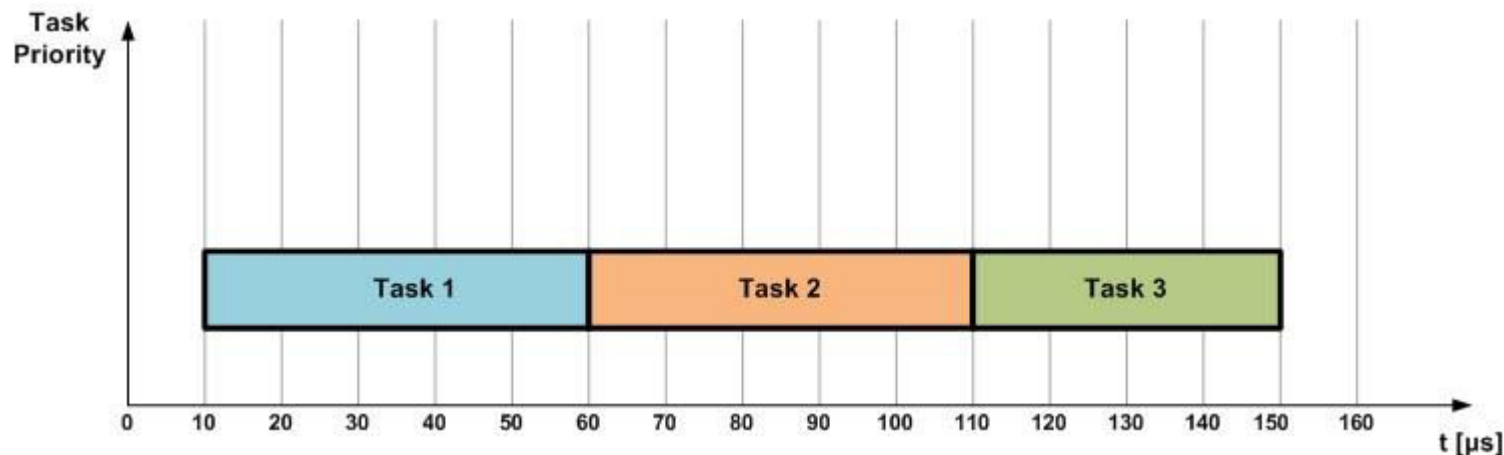
As an example let's have three tasks called Task 1, Task 2 and Task 3. Task 1 has the lowest priority and Task 3 has the highest priority. Their arrival times and execute times are listed in the table below.

k Name	Arrival Time [μ s]	Execute Time [μ s]
Task 1	10	50
Task 2	40	50
Task 3	60	40

In Fig. 1 we can see that Task 1 is the first to start executing, as it is the first one to arrive (at $t = 10 \mu$ s). Task 2 arrives at $t = 40 \mu$ s and since it has a higher priority, the scheduler interrupts the execution of Task 1 and puts Task 2 into running state. Task 3 which has the highest priority arrives at $t = 60 \mu$ s. At this moment Task 2 is interrupted and Task 3 is put into running state. As it is the highest priority task it runs until it completes at $t = 100 \mu$ s. Then Task 2 resumes its operation as the current highest priority task. Task 1 is the last to complete its operation.

Non-preemptive Scheduling (Co-Operative Scheduling)

In **non-preemptive scheduling**, the scheduler has more restricted control over the tasks. It can only start a task and then it has to wait for the task to finish or for the task to voluntarily return the control. A running task can't be stopped by the scheduler.



- If we take the three tasks specified in the table from the previous chapter and schedule them using a non-preemptive algorithm we get the behavior shown in Fig. 2. Once started, each task completes its operation and then the next one starts.
- The non-preemptive scheduling can simplify the synchronization of the tasks, but that is at the cost of increased response times to events. This reduces its practical use in complex real-time systems.

Difference between Pre-emptive and Non-Preemptive Scheduling

	Preemptive	Non-preemptive
Definition	Processes can use CPU for a limited time.	Once a process takes CPU it will only leave it after the completion of task.
Overheads	It has overheads	It does not have overheads
Flexibility	This type of scheduling is flexible.	This type of scheduling is non-flexible.
Interrupts	Processing of preemptively scheduled process can be interrupted.	Processing of non-preemptive scheduling cannot be interrupted.
Cost	Preemptive processes are directly related to the cost of the system.	Non-preemptive processes are not related to the cost of system in any ways.

Scheduling Algorithms

- **Processor utilization factor (U)**
- For a given task set of n periodic tasks, processor utilization factor U is the fraction of time that is spent for the execution of the task set. If S_i is a task from task set then C_i/T_i is the time spent by the processor for the execution of S_i . Similarly, for the task set of n periodic tasks

$$U = \sum_{i=1}^n \frac{C_i}{T_i}$$

- processor utilization is greater than one then that task set will not be schedulable by any algorithm. Processor utilization factor tells about the processor load on a single processor. $U=1$ means 100% processor utilization.

RATE MONOTONIC (RM) SCHEDULING ALGORITHM

- The Rate Monotonic scheduling algorithm is a simple rule that assigns priorities to different tasks according to their time period.
- That is task with smallest time period will have highest priority and a task with longest time period will have lowest priority for execution.
- As the time period of a task does not change so not its priority changes over time, therefore Rate monotonic is fixed priority algorithm. The priorities are decided before the start of execution and they does not change overtime.

Introduction to RATE MONOTONIC (RM) SCHEDULING ALGORITHM

- Rate monotonic scheduling Algorithm works on the principle of preemption. Preemption occurs on a given processor when higher priority task blocked lower priority task from execution.
- This blocking occurs due to priority level of different tasks in a given task set. rate monotonic is a preemptive algorithm which means if a task with shorter period comes during execution it will gain a higher priority and can block or preemptive currently running tasks.
- In RM priorities are assigned according to time period. Priority of a task is inversely proportional to its timer period. Task with lowest time period has highest priority and the task with highest period will have lowest priority. A given task is scheduled under rate monotonic scheduling Algorithm, if its satisfies the following equation:

$$\sum_{k=1}^n \frac{C_k}{T_k} \leq U_{RM} = n(2^{\frac{1}{n}} - 1)$$

where n is the number of tasks in a task set.

RATE MONOTONIC (RM) SCHEDULING ALGORITHM

- **Example of RATE MONOTONIC (RM) SCHEDULING ALGORITHM**
- For example, we have a task set that consists of three tasks as follows

Tasks	Release time(r_i)	Execution time(C_i)	Deadline (D_i)	Time period(T_i)
T1	0	0.5	3	3
T2	0	1	4	4
T3	0	2	6	6

Table 1. Task set

$$U = 0.5/3 + 1/4 + 2/6 = 0.167 + 0.25 + 0.333 = 0.75$$

RATE MONOTONIC (RM) SCHEDULING ALGORITHM

- As processor utilization is less than 1 or 100% so task set is schedulable and it also satisfies the above equation of rate monotonic scheduling algorithm.

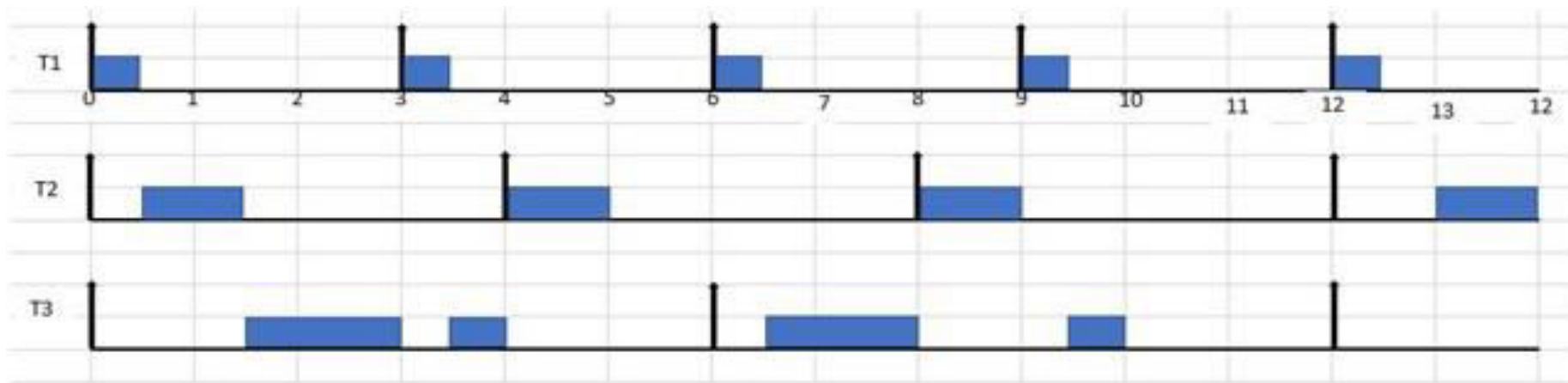


Figure 1. RM scheduling of Task set in table 1.

A task set given in table 1 its RM scheduling is given in figure 1. The explanation of above is as follows

RATE MONOTONIC (RM) SCHEDULING ALGORITHM

- According to RM scheduling algorithm task with shorter period has higher priority so T1 has high priority, T2 has intermediate priority and T3 has lowest priority. At $t=0$ all the tasks are released. Now T1 has highest priority so it executes first till $t=0.5$.
- At $t=0.5$ task T2 has higher priority than T3 so it executes first for one-time units till $t=1.5$. After its completion only one task is remained in the system that is T3, so it starts its execution and executes till $t=3$.
- At $t=3$ T1 releases, as it has higher priority than T3 so it preempts or blocks T3 and starts its execution till $t=3.5$. After that the remaining part of T3 executes.
- At $t=4$ T2 releases and completes its execution as there is no task running in the system at this time.
- At $t=6$ both T1 and T3 are released at the same time but T1 has higher priority due to shorter period so it preempts T3 and executes till $t=6.5$, after that T3 starts running and executes till $t=8$.
- At $t=8$ T2 with higher priority than T3 releases so it preempts T3 and starts its execution.
- At $t=9$ T1 is released again and it preempts T3 and executes first and at $t=9.5$ T3 executes its remaining part. Similarly, the execution goes on.

1.A Rate Monotonic scheduling

- Priority assignment based on rates of tasks
- Higher rate task assigned higher priority
- Schedulable utilization = 0.693 (Liu and Leyland)

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n(\sqrt[n]{2} - 1)$$

Where C_i is the computation time, and T_i is the release period

$$\lim_{n \rightarrow \infty} n(\sqrt[n]{2} - 1) = \ln 2 \approx 0.693147 \dots$$

- If $U < 0.693$, schedulability is guaranteed
- Tasks may be schedulable even if $U > 0.693$

RM example

Process	Execution Time	Period
P1	1	8
P2	2	5
P3	2	10

The utilization will be:

$$\frac{1}{8} + \frac{2}{5} + \frac{2}{10} = 0.725$$

The theoretical limit for 3 processes, under which we can conclude that the system is schedulable is:

$$U = 3(2^{\frac{1}{3}} - 1) = 0.77976 \dots$$

Since $0.725 < 0.77976 \dots$ **system is schedulable!**

1.B Deadline Monotonic scheduling

- Priority assignment based on relative deadlines of tasks
- Shorter the relative deadline, higher the priority

EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- Earliest deadline first (EDF) is dynamic priority scheduling algorithm for real time embedded systems.
- Earliest deadline first selects a task according to its deadline such that a task with earliest deadline has higher priority than others. It means priority of a task is inversely proportional to its absolute deadline.
- Since absolute deadline of a task depends on the current instant of time so every instant is a scheduling event in EDF as deadline of task changes with time.
- A task which has a higher priority due to earliest deadline at one instant it may have low priority at next instant due to early deadline of another task.
- EDF typically executes in preemptive mode i.e. currently executing task is preempted whenever another task with earliest deadline becomes active.

EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- EDF is an optimal algorithm which means if a task set is feasible then it is surely scheduled by EDF.
- Another thing is that EDF does not specifically take any assumption on periodicity of tasks so it is independent of Period of task and therefore can be used to schedule aperiodic tasks as well.
- If two tasks have same absolute deadline choose one of them randomly.

EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- **Example of EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM**
- An example of EDF is given below for task set of table-2.

Tas k	Release time(r_i)	Execution Time(C_i)	Deadline (D_i)	Time Period(T_i)
T1	0	1	4	4
T2	0	2	6	6
T3	0	3	8	8

Table 2. Task set

$U = 1/4 + 2/6 + 3/8 = 0.25 + 0.333 + 0.375 = 0.95 = 95\%$
As processor utilization is less than 1 or 100% so task set is surely schedulable by EDF.

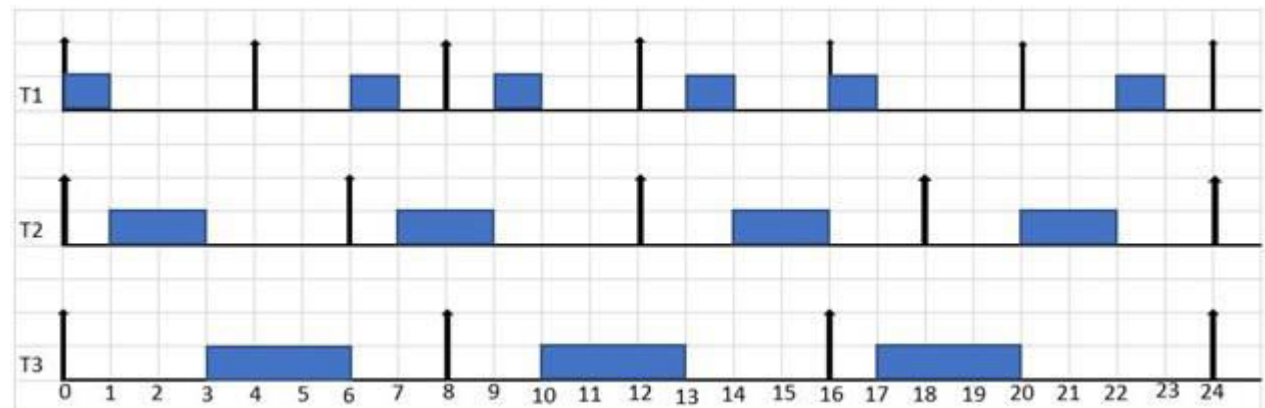


Figure 2. Earliest deadline first scheduling of task set in Table -2

EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- At $t=0$ all the tasks are released, but priorities are decided according to their absolute deadlines so T1 has higher priority as its deadline is 4 earlier than T2 whose deadline is 6 and T3 whose deadline is 8, that's why it executes first.
- At $t=1$ again absolute deadlines are compared and T2 has shorter deadline so it executes and after that T3 starts execution but at $t=4$ T1 comes in the system and deadlines are compared, at this instant both T1 and T3 has same deadlines so ties are broken randomly so we continue to execute T3.
- At $t=6$ T2 is released, now deadline of T1 is earliest than T2 so it starts execution and after that T2 begins to execute. At $t=8$ again T1 and T2 have same deadlines i.e. $t=16$, so ties are broken randomly and T2 continues its execution and then T1 completes. Now at $t=12$ T1 and T2 come in the system simultaneously so by comparing absolute deadlines, T1 and T2 has same deadlines therefore ties broken randomly and we continue to execute T3.

EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- At $t=13$ T1 begins its execution and ends at $t=14$. Now T2 is the only task in the system so it completes its execution.
- At $t=16$ T1 and T2 are released together, priorities are decided according to absolute deadlines so T1 executes first as its deadline is $t=20$ and T3's deadline is $t=24$. After T1 completion T3 starts and reaches at $t=17$ where T2 comes in the system now by deadline comparison both have same deadline $t=24$ so ties broken randomly and we continue to execute T3.
- At $t=20$ both T1 and T2 are in the system and both have same deadline $t=24$ so again ties broken randomly and T2 executes. After that T1 completes its execution. In the same way system continues to run without any problem by following EDF algorithm.

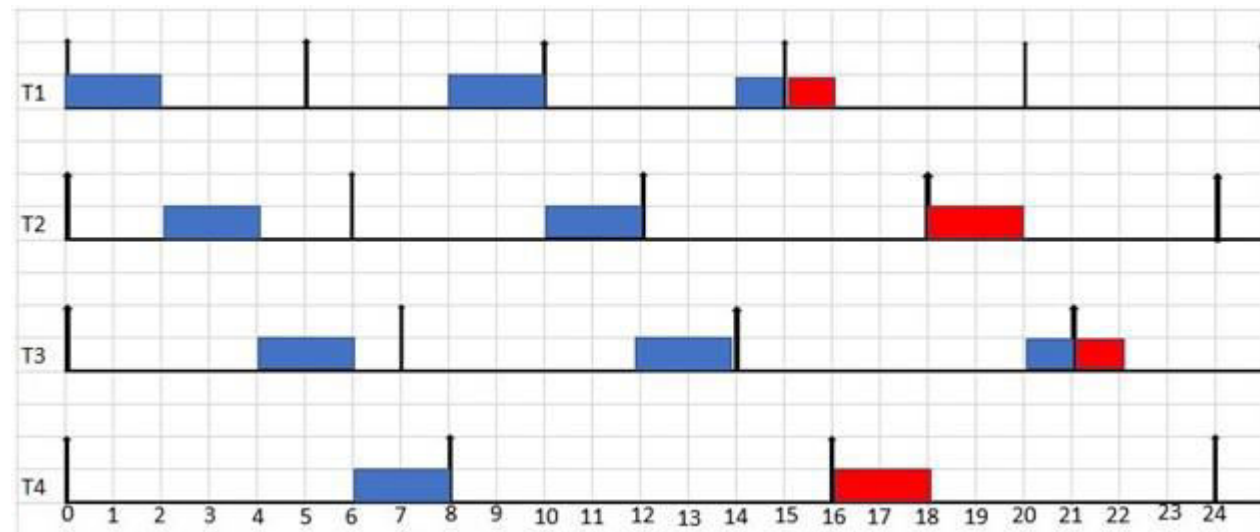
EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- **Transient Over Load Condition & Domino Effect in Earliest deadline first**
- Transient over load is a short time over load on the processor. Transient overload condition occurs when the computation time demand of a task set at an instant exceeds the processor timing capacity available at that instant.
- Due to transient over load tasks miss their deadline. This transient over load may occur due many reasons such as changes in the environment, simultaneous arrival of asynchronous jobs, system exception.
- operating systems under EDF, whenever a task in Transient overload condition miss its deadline and as result each of other tasks start missing their deadlines one after the other in sequence, such an effect is called domino effect. It jeopardizes the behavior of the whole system. An example of such condition is given below.

EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

Task	Release time(r_i)	Execution Time(C_i)	Deadline (D_i)	Period(T_i)
T1	0	2	5	5
T2	0	2	6	6
T3	0	2	7	7
T4	0	2	8	8

Figure 3. Domino effect under Earliest deadline first



EARLIEST DEADLINE FIRST (EDF) SCHEDULING ALGORITHM

- As in the above figure at $t=15$ T1 misses its deadline and after that at $t=16$ T4 is missing its deadline then T2 and finally T3 so the whole system is collapsed. It is clearly proved that EDF has a shortcoming due to domino effect and as a result critical tasks may miss their deadlines.
- The solution of this problem is another scheduling algorithm that is least laxity first (LLF). It is an optimal scheduling algorithm. Demand bound function and Demand bound analysis are also used for schedulability analysis of given set of tasks.

2.A Earliest Deadline First (EDF)

- Dynamic Priority Scheduling
- Priorities are assigned according to deadlines:
 - Earlier deadline, higher priority
 - Later deadline, lower priority
- The first and the most effectively widely used dynamic priority-driven scheduling algorithm.
- Effective for both preemptive and non-preemptive scheduling.

Two Periodic Tasks

- Execution profile of two periodic tasks

- Process A

• Arrives	0	20	40	...
• Execution Time	10	10	10	...
• End by	20	40	60	...

- Process B

• Arrives	0	50	100	...
• Execution Time	25	25	25	...
• End by	50	100	150	...

- Question: Is there enough time for the execution of two periodic tasks?

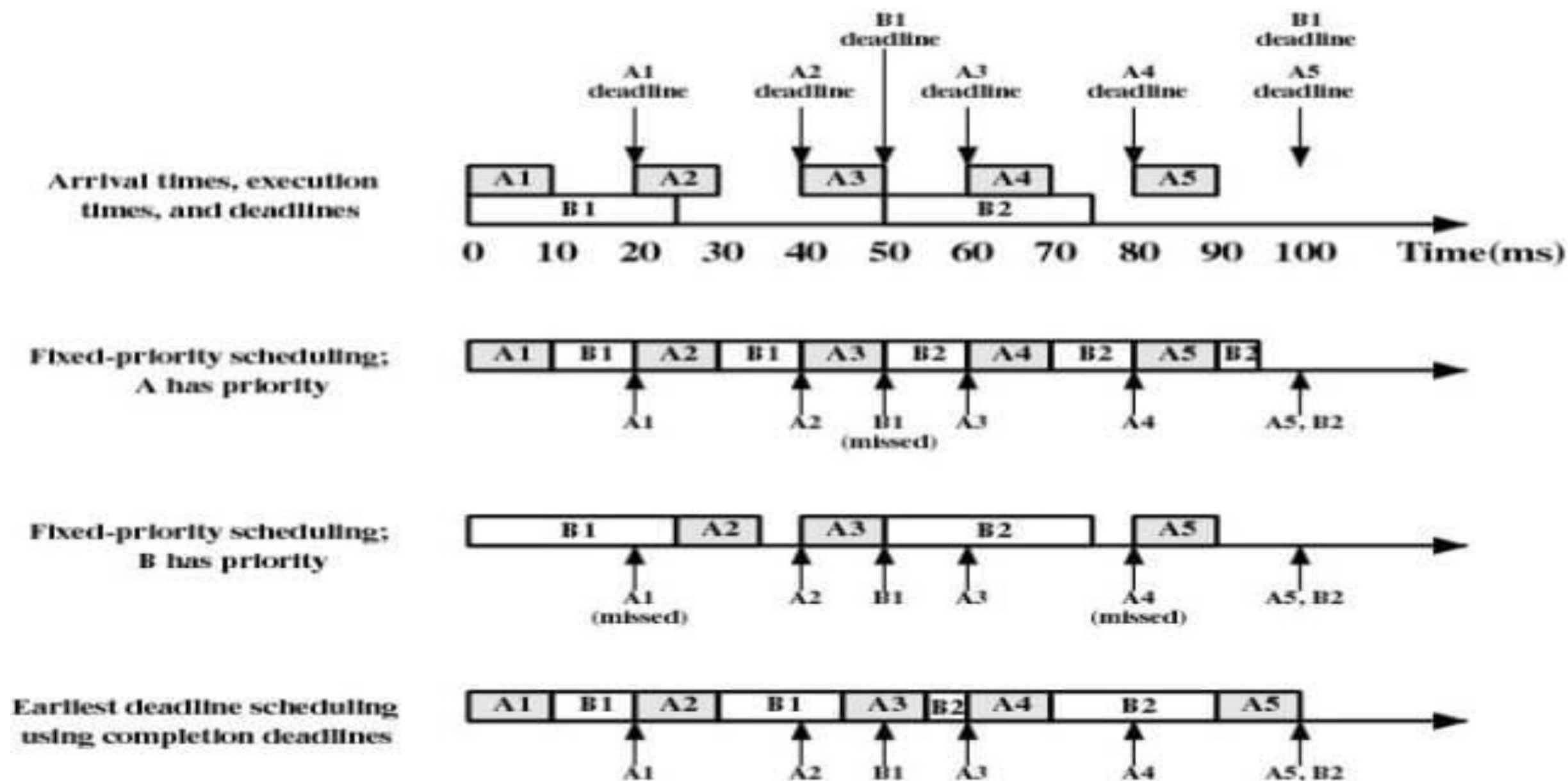
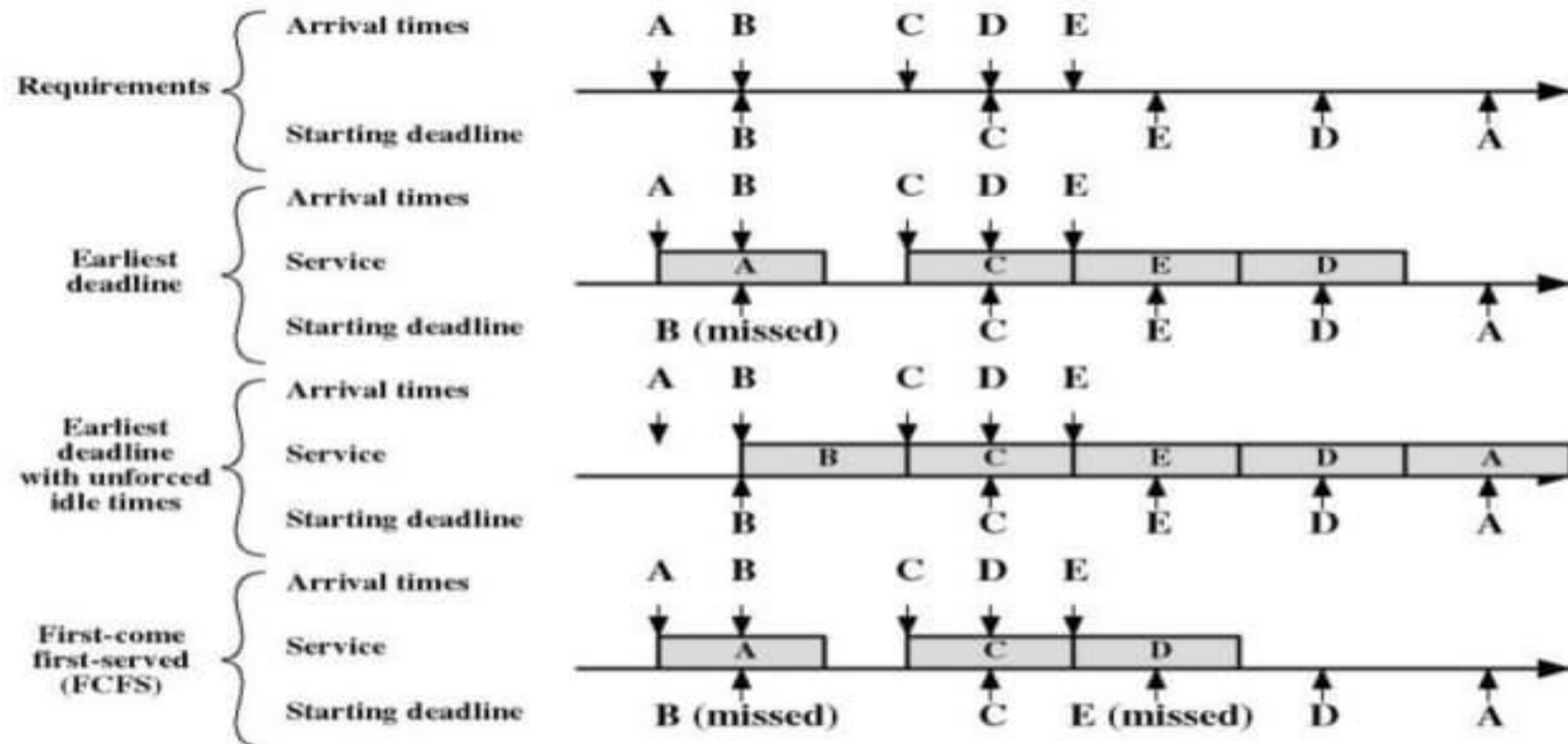
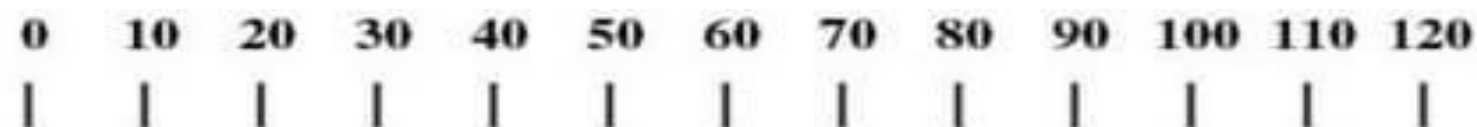


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines

Five Periodic Tasks

- Execution profile of five periodic tasks

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70



Advantages and Disadvantages of EDF over rate monotonic

- **Advantages of EDF over rate monotonic**
- No need to define priorities offline
- It has less context switching than rate monotonic
- It utilize the processor maximum up to 100% utilization factor as compared to rate monotonic
- **Disadvantages of EDF over rate monotonic**
- It is less predictable. Because response time of tasks are variable and response time of tasks are constant in case of rate monotonic or fixed priority algorithm.
- EDF provided less control over the execution
- It has high overheads

2.B Least Laxity First (LLF)

- Dynamic preemptive scheduling with dynamic priorities
- **Laxity** : The difference between the time until a tasks completion deadline and its remaining processing time requirement.
- a laxity is assigned to each task in the system and minimum laxity tasks are executed first.
- **Larger overhead than EDF due to higher number of context switches caused by laxity changes at run time**
 - Less studies than EDF due to this reason

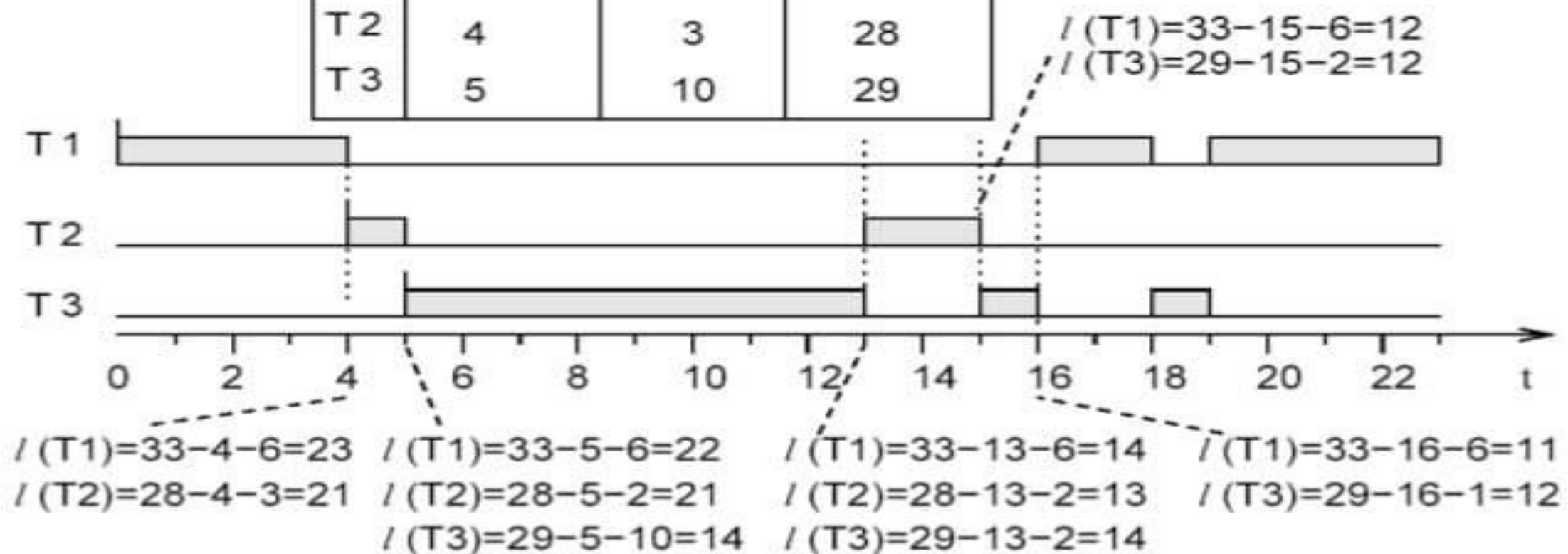
Least Laxity First Cont

- LLF considers the execution time of a task, which EDF does not.
- LLF assigns higher priority to a task with the least laxity.
- A task with zero laxity must be scheduled right away and executed without preemption or it will fail to meet its deadline.
- **The negative laxity indicates that the task will miss the deadline**, no matter when it is picked up for execution.

The **definition** of **laxity** is the quality or condition of being loose. Being loose and relaxed in the enforcement of a procedure is an example of **laxity**.

Least Laxity First Example

	arrival	duration	deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29



LEAST LAXITY FIRST (LLF) SCHEDULING ALGORITHM

- Least Laxity First (LLF) is a job level dynamic priority scheduling algorithm. It means that every instant is a scheduling event because laxity of each task changes on every instant of time.
- A task which has least laxity at an instant, it will have higher priority than others at this in
- More formally, priority of a task is inversely proportional to its run time laxity. As the laxity of a task is defined as its urgency to execute. Mathematically it is described as

$$L_i = D_i - C_i$$

LEAST LAXITY FIRST (LLF) SCHEDULING ALGORITHM

- Here d_i is the deadline of a task, C_i is the worst-case execution time(WCET) and L_i is laxity of a task. It means laxity is the time remaining after completing the WCET before the deadline arrives. For finding the laxity of a task in run time current instant of time also included in the above formula.

$$L_i = D_i - (t_i + C_i^R)$$

Here t_i is the current instant of time and C_i^R is the remaining WCET of the task. By using the above equation laxity of each task is calculated at every instant of time, then the priority is assigned. One important thing to note is that laxity of a running task does not change it remains same whereas the laxity of all other tasks is decreased by one after every one-time unit.

LEAST LAXITY FIRST (LLF) SCHEDULING ALGORITHM

- Example of Least Laxity first scheduling Algorithm
- An example of LLF is given below for a task set.

Task	Release time(r_i)	Execution Time(C_i)	Deadline (D_i)	Period(T_i)
T1	0	2	6	6
T2	0	2	8	8
T3	0	3	10	10

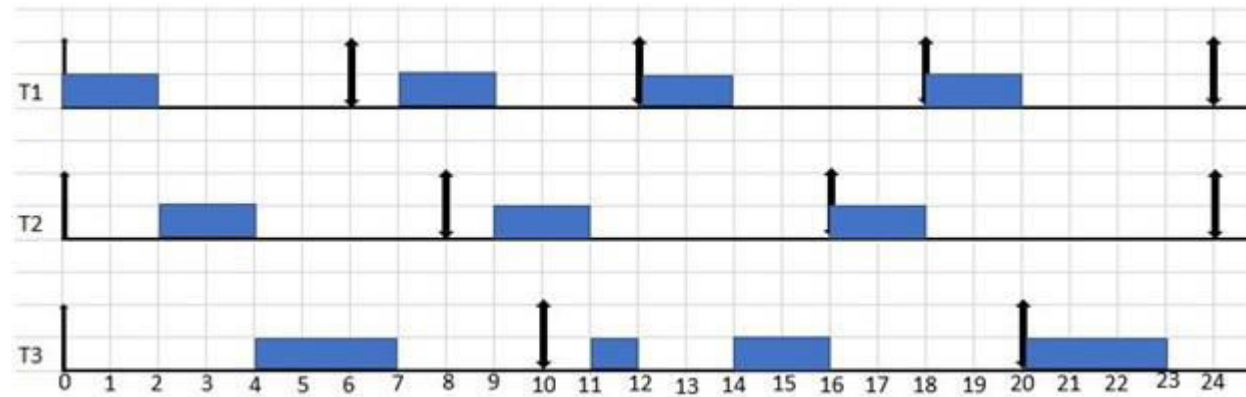


Figure 4. LLF scheduling algorithm

LEAST LAXITY FIRST (LLF) SCHEDULING ALGORITHM

- 1. At $t=0$ laxities of each task are calculated by using equation 4.2. as
 - $L1 = 6-(0+2) = 4$
 - $L2 = 8-(0+2) = 6$
 - $L3 = 10-(0+3) = 7$
- As task T1 has least laxity so it will execute with higher priority. Similarly, At $t=1$ its priority is calculated it is 4 and T2 has 5 and T3 has 6, so again due to least laxity T1 continue to execute.
- 2. At $t=2$ T1 is out of the system so Now we compare the laxities of T2 and T3 as following
 - $L2 = 8-(2+2) = 4$
 - $L3 = 10-(2+3) = 5$
- Clearly T2 starts execution due to less laxity than T3. At $t=3$ T2 has laxity 4 and T3 also has laxity 4, so ties are broken randomly so we continue to execute T2. At $t=4$ no task is remained in the system except T3 so it executes till $t=6$. At $t=6$ T1 comes in the system so laxities are calculated again
 - $L1 = 6-(0+2) = 4$
 - $L3 = 10-(6+1) = 3$

LEAST LAXITY FIRST (LLF) SCHEDULING ALGORITHM

- So T3 continues its execution.
- At $t=8$ T2 comes in the system where as T1 is running task. So at this instant laxities are calculated
 - $L1 = 12 - (8 + 1) = 3$
 - $L2 = 16 - (8 + 2) = 6$
- o T1 completes its execution. After that T2 starts running and at $t=10$ due to laxity comparison T2 has higher priority than T3 so it completes its execution.
 - $L2 = 16 - (10 + 1) = 5$
 - $L3 = 20 - (10 + 3) = 7$
- $t=11$ only T3 in the system so it starts its execution.
- At $t=12$ T1 comes in the system and due to laxity comparison at $t=12$ T1 wins the priority and starts its execution by preempting T3. T1 completes its execution and after that at $t=14$ due to alone task T3 starts running its remaining part. So, in this way this task set executes under LLF algorithm.

LEAST LAXITY FIRST (LLF) SCHEDULING ALGORITHM

- LLF is an optimal algorithm because if a task set will pass utilization test then it is surely schedulable by LLF. Another advantage of LLF is that it some advance knowledge about which task going to miss its deadline.
- On other hand it also has some disadvantages as well one is its enormous computation demand as each time instant is a scheduling event. It gives poor performance when more than one task has least laxity.

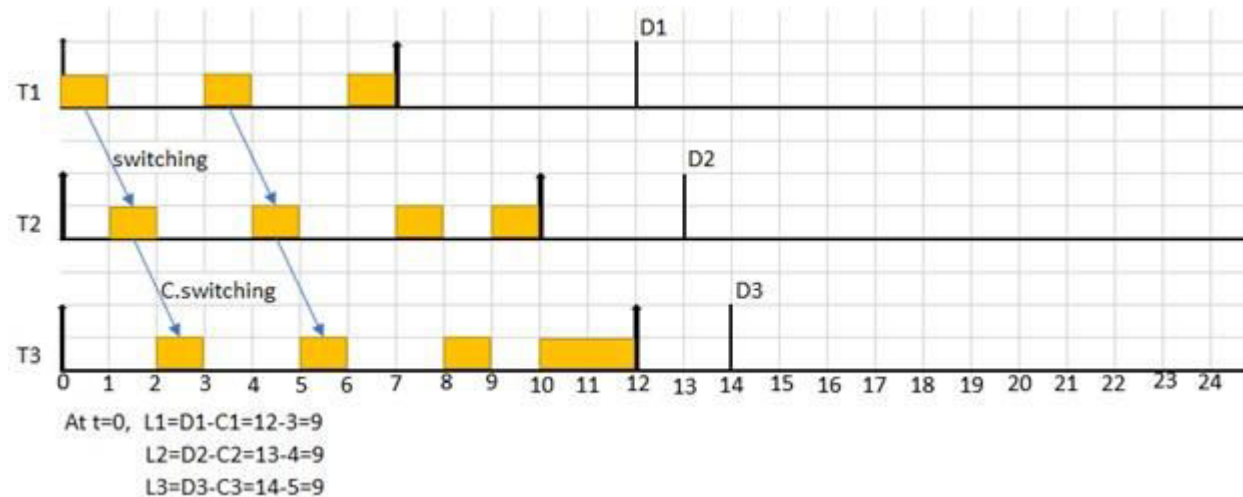
Enhanced Least Laxity First (LLF) Scheduling Algorithm

- LLF is good for avoiding transient overload and domino effect occurring in EDF. But it also has a shortcoming, that is thrashing. When more than one task has least laxity than this phenomenon of Thrashing occurs.
- Thrashing is the behavior of preemption by the tasks one another at each time tick. This kind of preemption results in context switch at each instant of time.
- This context switching results in more computation power consumption as switching from one task to another which result in saving and loading registers, memory mapping and updating many tables and lists. So, it results in loss of computation time as well.
- To overcome this short come of LLF an improved version of LLF is introduced that is Enhanced least laxity first scheduling algorithm (ELLF). IN ELF whenever more than one task has least laxity than they all are grouped together and EDF is applied within the group whereas taking this group as a single task LLF is applied to this group and other remaining tasks in the task set.

Enhanced Least Laxity First (LLF) Scheduling Algorithm

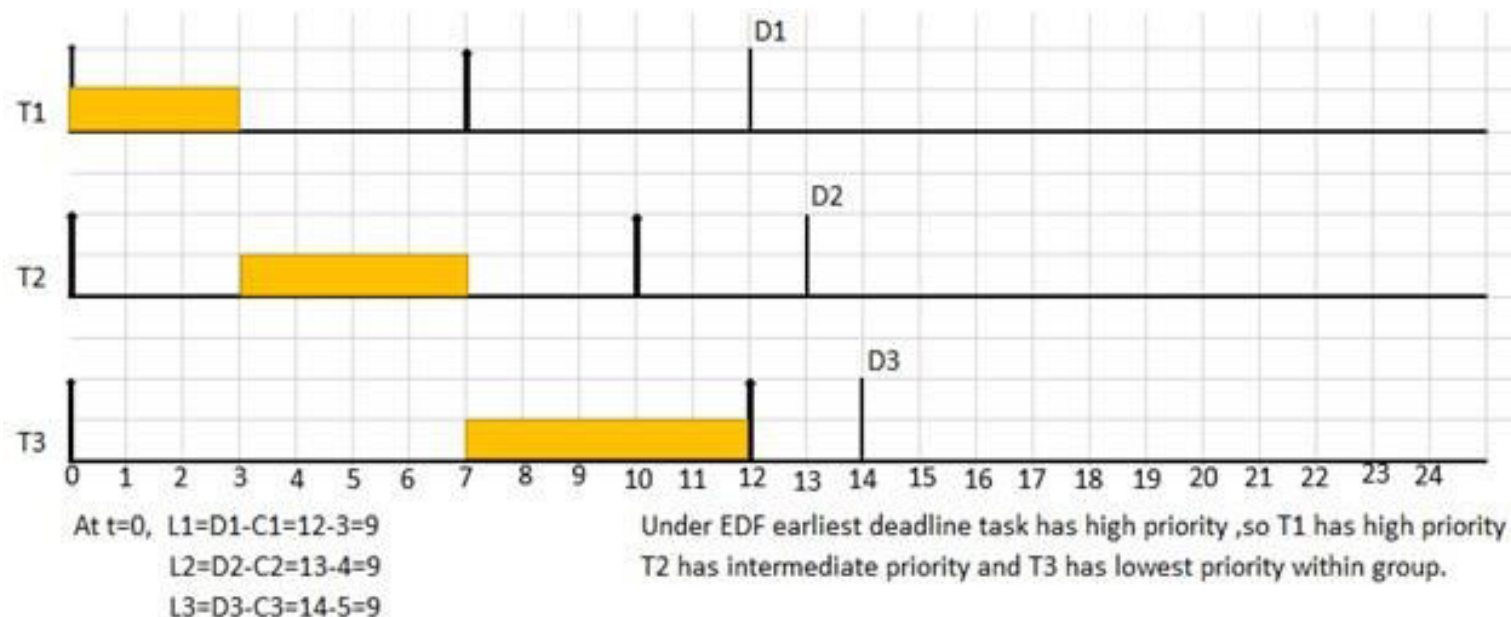
- An example of LLF with thrashing is given below

Task	Release time(r_i)	Execution Time(C_i)	Deadline (D_i)	Period(T_i)
T1	0	3	12	7
T2	0	4	13	10
T3	0	5	14	12



Enhanced Least Laxity First (LLF) Scheduling Algorithm

- So, to overcome this context switching due to thrashing We employ ELLF on the same example by grouping together the tasks having same least laxity, then apply EDF to avoid context switching.



Maximum Urgency First Algorithm

- This algorithm is a **combination of fixed and dynamic priority scheduling, also called mixed priority scheduling.**
- With this algorithm, each task is given an urgency which is defined as a combination of two fixed priorities (criticality and user priority) and a dynamic priority that is inversely proportional to the laxity.
- The MUF algorithm assigns priorities in two phases
- **Phase One** concerns the assignment of static priorities to tasks
- **Phase Two** deals with the run-time behavior of the MUF scheduler

3.A Maximum Urgency First Algorithm phase 1

The first phase consists of these steps :

- 1) It sorts the tasks from the shortest period to the longest period. Then it defines the critical set as the first N tasks such that the total CPU load factor does not exceed 100%. These tasks are guaranteed not to fail even during a transient overload.
- 2) All tasks in the critical set are assigned high criticality. The remaining tasks are considered to have low criticality.
- 3) Every task in the system is assigned an optional unique user priority

Maximum Urgency First Algorithm phase 2

- In the second phase, the MUF scheduler follows an algorithm to select a task for execution.
- This algorithm is executed whenever a new task is arrived to the ready queue.
- The algorithm is as follows:
 - 1) If there is only one highly critical task, pick it up and execute it.
 - 2) If there are more than one highly critical task, select the one with the highest dynamic priority. Here, the task with the least laxity is considered to be the one with the highest priority.
 - 3) If there is more than one task with the same laxity, select the one with the highest user priority.

Popular Scheduling Algorithms

- Some of the most popular scheduling algorithms that are used in CPU scheduling. Not all of them are suitable for use in real-time embedded systems. Currently, the most used algorithms in practical RTOS are **non-preemptive scheduling**, **round-robin scheduling**, and **preemptive priority scheduling**.
- **First Come, First Served (FCFS)**
- FCFS is a non-preemptive scheduling algorithm that has no priority levels assigned to the tasks. The task that arrives first into the scheduling queue (i.e enters ready state), gets put into the running state first and starts utilizing the CPU. It is a relatively simple scheduling algorithm where all the tasks will get executed eventually. The response time is high as this is a non-preemptive type of algorithm.

Popular Scheduling Algorithms

- **Shortest Job First (SJF)**
- In the shortest job first scheduling algorithm, the scheduler must obtain information about the execution time of each task and it then schedules the one with the shortest execution time to run next.
- SJF is a non-preemptive algorithm, but it also has a preemptive version. In the preemptive version of the algorithm (aka shortest remaining time) the parameter on which the scheduling is based is the remaining execution time of a task. If a task is running it can be interrupted if another task with shorter remaining execution time enters the queue.
- A disadvantage of this algorithm is that it requires the total execution time of a task to be known before it is run.

Popular Scheduling Algorithms

- **Priority Scheduling**

- Priority scheduling is one of the most popular scheduling algorithms. Each task is assigned a priority level. The basic principle is that the task with the highest priority will be given the opportunity to use the CPU.
- In the preemptive version of the algorithm, a running task can be stopped if a higher priority task enters the scheduling queue. In the non-preemptive version of the algorithm once a task is started it can't be interrupted by a higher priority task.
- Of course, not all tasks can have unique priority levels and there will always be tasks that have the same priority. Different approaches can be used for handling the scheduling of those tasks (e.g FCFS scheduling or round-robin scheduling).

Popular Scheduling Algorithms

- **Round-Robin Scheduling**
- Round-robin is a preemptive type of scheduling algorithm. There are no priorities assigned to the tasks. Each task is put into a running state for a fixed predefined time. This time is commonly referred to as time-slice (aka quantum). A task can not run longer than the time-slice. In case a task has not completed by the end of its dedicated time-slice, it is interrupted, so the next task from the scheduling queue can be run in the following time slice. A pre-empted task has an opportunity to complete its operation once it's again its turn to use a time-slice.
- An advantage of this type of scheduling is its simplicity and relatively easy implementation.

References

- <https://www.slideshare.net/ajal4u/realtime-scheduling-algorithms>
- <https://microcontrollerslab.com/scheduling-algorithm-real-time/>
- <https://open4tech.com/rtos-scheduling-algorithms/>

School of Electrical and Computer
Engineering N.T.U.A.



Embedded System Design

Ioannis Koutras
Dimitrios Soudris

Microprocessors and Digital Systems
Lab, ECE, NTUA



ARM Programmer's Model

Άδεια Χρήσης

Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άδεια χρήσης άλλου τύπου, αυτή πρέπει να αναφέρεται ρητώς.



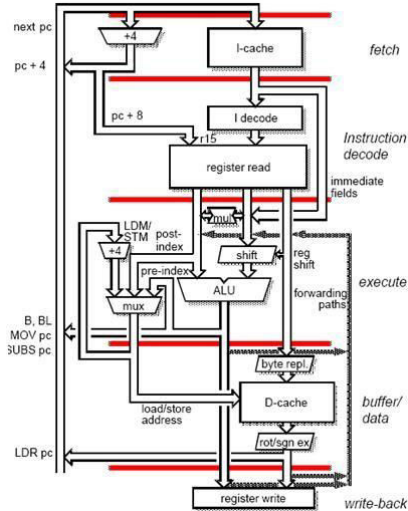


-) Founded in November 1990
-) Does not fabricate processors itself
-) Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers
-) Also develops technologies to assist with the designing of the ARM architecture
 - Software tools
 - Development boards, debug hardware
 - Bus architectures, peripherals

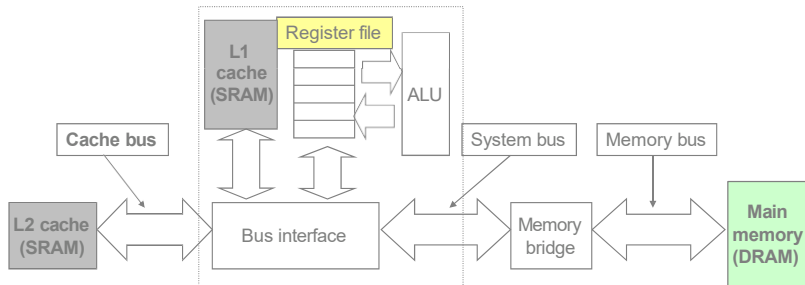
ARM as a RISC architecture

-) ARM confronts to the Reduced Instruction Set Computer (RISC) architecture.
-) A typical RISC system is defined:
 - Load / Store model
 - Operations on registers and not directly on memory
 - All data must be loaded into registers before they can be operated on.
 - Fixed instruction length
 - Small number of addressing modes
 - A large set of general-purpose registers that can hold either data or an address.
-) However, ARM is not a pure RISC architecture:
 - Conditional execution of most instructions
 - Arithmetic instructions alter condition codes only when desired.
 - Addition of a 32-bit barrel shifter before instruction execution

ARM Organization and Implementation



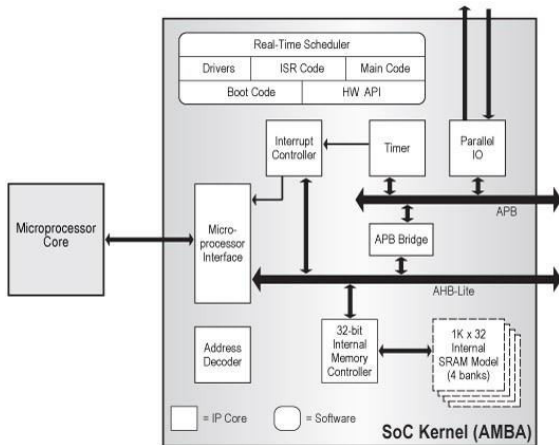
The Bigger Picture



Source: R. Bryant, D. O'Hallaron, Computer Systems: A Programmer's Perspective

ARM Organization and Implementation – AMBA

The important aspect of a SoC is not only which components or blocks it houses, but also how they are interconnected.



AMBA Specification

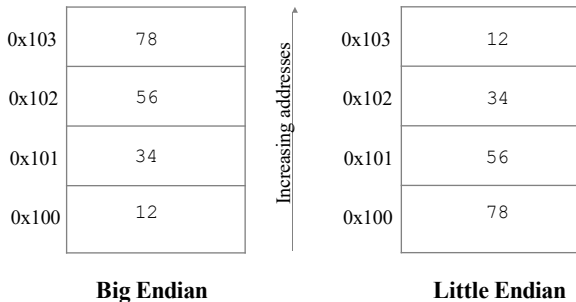
-) The **Advanced Microcontroller Bus Architecture** was introduced in 1996 and is widely used as the on-chip bus in System-on-a-chip (SoC) designs processors.
-) AMBA Goals:
 - Technology independence
 - To encourage modular system design
-) The AMBA 3.0 specification define five buses/interfaces:
 - Advanced eXtensible Interface (AXI)
 - Advanced High-performance Bus (AHB)
 - Advanced System Bus (ASB)
 - Advanced Peripheral Bus (APB)
 - Advanced Trace Bus (ATB)
-) ARM provides **ARMA Design Kit (ADK)** as a generic, stand-alone environment to enable the rapid creation of AMBA-based components and System-on-Chip (SoC) designs.

ARM Data Sizes and Instructions

-) The ARM is a 32-bit RISC architecture, so in relation to that:
 - **Byte** means 8 bits
 - **Halfword** means 16 bits (two bytes)
 - **Word** means 32 bits (four bytes)
-) Most ARM cores implement two instruction sets:
 - 32-bit ARM Instruction Set
 - 16-bit Thumb Instruction Set
-) ARM cores can be configured to view words stored in memory as either Big-Endian or Little-Endian format.

Big Endian vs. Little Endian

How 0x12345678 would be stored in a 32-bit memory?



The ARM programmer's model

What is a mode?

-) Characterized by specific behavior, privileges, associated registers
-) Triggered by some action (e.g. exception, interrupt)

The ARM has seven basic operating modes:

-) **User:** normal program execution mode
-) **FIQ:** used for handling a high priority (fast) interrupt
-) **IRQ:** used for handling a low priority (normal) interrupt
-) **Supervisor:** entered on reset and when a Software Interrupt instruction is executed
-) **Abort:** used for handling memory access violations
-) **Undefined:** used for handling undefined instructions
-) **System:** a privileged mode that uses the same registers as the user mode

ARM's Register Set

ARM has a total of 37 registers, all of which are 32-bit long

-) 30 general purpose registers
-) 1 dedicated program counter (*pc*)
-) 1 dedicated current program status register (*cpsr*)
-) 5 dedicated saved program status registers (*spsr*)

In any mode only a subset of the 37 registers are visible

-) The hidden registers are called *banked registers*.
-) The current processor mode governs which registers are accessible.

ARM's Register Set

Each mode can access

-) A particular set of *r0-r12* registers
-) A particular *r13* (the stack pointer, *sp*) and *r14* (the link register, *lr*)
-) The program counter, *r15* (*pc*)
-) The current program status register, *cpsr*

Privileged modes (except System) can also access

-) A particular saved program status register (*spsr*)

ARM Register Organization

Current Visible Registers

User Mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr

Banked out Registers

FIQ	IRQ	SVC	Undef	Abort
r8				
r9				
r10				
r11				
r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
spsr	spsr	spsr	spsr	spsr

ARM Register Organization

Current Visible Registers

FIQ Mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
spsr

User

r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)

Banked out Registers

IRQ

SVC

Undef

Abort

r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)

spsr	spsr	spsr	spsr
------	------	------	------





















Banked Registers

-) Banking of registers implies that the specific register depends not only on the number ($r0$, $r1$, $r2 \dots r15$) but also on the processor mode
-) The values stored in banked registers are preserved across mode changes.

Example

-) Assume that the processor is executing in user mode
-) Assume that the processor writes **0** in $r0$ and **8** in $r8$.
-) Processor changes to FIQ mode
 - In FIQ mode the value of $r0$ is
 - If processor overwrites both $r0$ and $r8$ with **1** in FIQ mode and changes back to user mode
 - o The new value stored in $r0$ (in user mode) is
 - o The new value stored in $r8$ (in user mode) is

ARM Register Organization

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	 R8-fiq	R8	R8	R8	R8
R9	 R9-fiq	R9	R9	R9	R9
R10	 R10-fiq	R10	R10	R10	R10
R11	 R11-fiq	R11	R11	R11	R11
R12	 R12-fiq	R12	R12	R12	R12
R13	 R13-fiq	 R13-svc	 R13-abt	 R13-irq	 R13-und
R14	 R14-fiq	 R14-svc	 R14-abt	 R14-irq	 R14-und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR-fiq	 SPSR-svc	 SPSR-abt	 SPSR-irq	 SPSR-und

 = banked register

SPSR = State Program Status Register

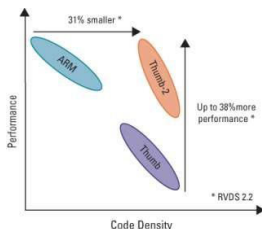
Thumb

Thumb is a 16-bit instruction set

-) Optimized for code density from C code
-) Improved performance from narrow memory
-) Subset of the functionality of the ARM instruction set

Core has additional execution state – Thumb

-) Switch between ARM and Thumb using **BX** instruction



For most instructions generated by compiler:

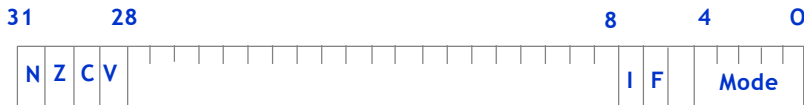
-) Conditional execution is not used
-) Source and destination registers identical
-) Only Low registers used
-) Constants are of limited size
-) Inline barrel shifter not used

Current Program Status Register

- Current Program Status Register (cpsr) is a dedicated register
- Holds information about the most recently performed ALU operation
- Controls the enabling and disabling of interrupts (both IRQ and FIQ)
- Sets the processor operating mode
- Sets the processor state



Current Program Status Register



-) *cpsr* has two important pieces of information:
 - **Flags:** contains the condition flags
 - **Control:** contains the processor mode, state and interrupt mask bits
-) All fields of the *cpsr* can be read/written in privileged modes
-) Only the flag field of *cpsr* can be written in User mode, all fields can be read in User mode

M[4:0]	Mode
10000	User
10001	FIQ
10010	IRQ
10011	SVC
10111	Abort
11011	Undefined
11111	System

Current Program Status Register



) Condition code flags

- N = **N**egative result from ALU
- Z = **Z**ero result from ALU
- C = ALU operation **C**arried out
- V = ALU operation **oV**erflowed

) Sticky Overflow flag – Q flag

- Architecture 5TE/J only
- Indicates if saturation has occurred

) Interrupt Disable bits

- I = 1: Disables the IRQ
- F = 1: Disables the FIQ

) T Bit

- Architecture xT only
- T = 0: Processor in ARM state
- T = 1: Processor in Thumb state

) Mode bits

- Specify the processor mode

The ARM Assembly Language

-) ARM instructions can be broadly classified as:
 - *Data Processing Instructions*: manipulate data within the registers
 - *Branch Instructions*: changes the flow of instructions or call a subroutine
 - *Load-Store Instructions*: transfer data between registers and memory
 - *Software Interrupt Instructions*: cause a software interrupt
 - *Program Status Instructions*: read / write the processor status registers
-) All instructions can access *r0-r14* directly.
-) Most instructions also allow use of the *pc*.
-) Specific instructions to allow access to *cpsr* and *spsr*.

Data Processing Instructions

Manipulate data within registers

-) Move operations
-) Arithmetic operations
-) Logical operations
-) Comparison operations
-) Multiply operations

Appending the S suffix for an instruction, e.g. *ADDS*

-) Signifies that the instruction's execution will update the flags in the *cpsr*

Typical ARM Data Processing Instructions



- **Operation** – Specifies the instruction to be performed

Almost all ARM instructions can be conditionally executed

- **Cond** – Specifies the optional conditional flags which have to be set under which to execute the instruction
- **S bit** – Signifies that the instruction updates the conditional flags
- **Rd** – Specifies the destination register
- **Rn** – Specifies the first source operand register
- **ShifterOperand2** – Specifies the second source operand
 - Could be a register, immediate value, or a shifted register / immediate value

Some data processing instructions may not specify the destination register or the source register

Data Processing Instructions

Consist of:

Arithmetic:	ADD	ADC	SUB	SBC	RSB	RSC
Logical:	AND	ORR	EOR	BIC		
Comparisons:	CMP	CMN	TST	TEQ		
Data movement:	MOV	MVN				

These instructions only work on registers, NOT memory

Move instructions

- *MOV* moves a 32-bit value into a register
- *MVN* moves the NOT of the 32-bit value into a register

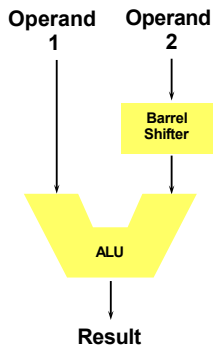
Example

BEFORE $r5 = 5$
 $r7 = 8$

MOV $r7, r5$

AFTER $r5 =$
 $r7 =$

The Barrel Shifter

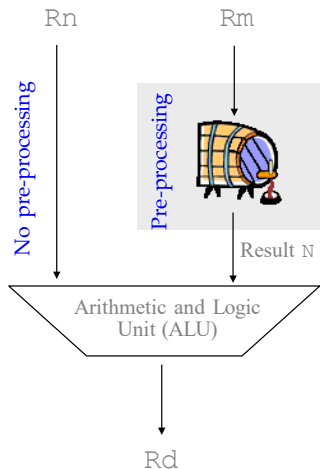


Operand 2 could be:

-) Register, optionally with shift operation
 - Shift value can be either be:
 - o Unsigned integer
 - o Specified in bottom byte of another register
 - Used for multiplication by constant
-) Immediate value
 - 8 bit number, with a range of 0-255
 - o Rotated right through even number of positions
 - Allows increased range of 32-bit constants to be loaded directly into registers

The ARM Barrel Shifter

- › Data processing instructions are processed within the ALU
- › ARM can shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before the value enters the ALU
- › Can achieve fast multiplies or division by a power of 2
- › Data-processing instructions that do not use the barrel shifter:
 - *MUL* (multiply)
 - *CLZ* (count leading zeros)



Using the Barrel Shifter

-) *LSL* shifts bits to the left.
-) It is similar to the C-language operator «

Example

BEFORE $r5 = 5$
 $r7 = 8$

MOV $r7, r5, LSL \#2$

AFTER $r5 =$
 $r7 =$

Arithmetic Instructions

Addition and subtraction of 32-bit signed and unsigned values

- Subtraction

Example

BEFORE r0 = 0x00000000

r1 = 0x00000002

r2 = 0x00000001

SUB r0, r1, r2

AFTER r0 =

r1 =

r2 =

- Addition

Example

BEFORE r0 = 0x00000000

r1 = 0x00000005

ADD r0, r1, r1, LSL #1

AFTER r0 =

r1 =

Logical Instructions

Bit-wise logical operations on two source registers

AND, ORR, EOR, BIC

- Logical OR

Example

BEFORE r0 = 0x00000000

r1 = 0x02040608

r2 = 0x10305070

ORR r0, r1, r2

AFTER r0 =

r1 =

r2 =

- Logical bit clear (BIC)

Example

BEFORE r1 = 0b1111

r2 = 0b0101

BIC r0, r1, r2

AFTER r0 =

r1 =

r2 =

Comparison Instructions

-) Compare or test a register with a 32-bit value
 - *CMP, CMN, TEQ, TST*
-) Registers under comparison are not affected; *cpsr* updated
-) Do not need the S suffix

Example

BEFORE *cpsr* = nzcvqiFt_USER
 r0 = 4
 r9 = 4

CMP *r0*, *r9*

AFTER *cpsr* =
 r0 =
 r9 =

Multiply Instructions

-) Multiply a pair of registers and optionally add (accumulate) the value stored in another register
 - *MUL, MLA*
-) Special instructions called long multiplies accumulate onto a pair of registers representing a 64-bit value
 - *SMLAL, SMULL, UMLAL, UMUL*

Example

BEFORE r0 = 0x00000000
 r1 = 0x00000002
 r2 = 0x00000002

MUL r0, r1, r2

AFTER r0 =
 r1 =
 r2 =

Branch Instructions

-) To change the flow of execution or to call a routine
 -) Supports subroutine calls, *if-then-else* structures, loops
 -) Change of execution forces the *pc* to point to a new address
-
-) Four different branch instructions on the ARM
 - *B<cond> label*
 - *BL<cond> label*
 - *BX<cond> Rm*
 - *BLX<cond> label | Rm*

Condition Mnemonics

Suffix/Mnemonic	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N!=V
AL	Always	

Conditional Execution

-) Most ARM instructions are conditionally executed
 - Instruction executes only if the condition-code flags satisfy a given test
-) Increases performance
 - Reduces the number of branches, which reduces the number of pipeline flushes
-) Improves code density
-) Two-letter mnemonic appended to the instruction mnemonic

Example usage

An add operation takes the form:

• *ADD r0, r1, r2; $r0 = r1 + r2$*

To execute this only if the zero flag is set:

• *ADDEQ r0, r1, r2; If zero flag set then $r0 = r1 + r2$*

Value of Conditional Execution

This improves code density and performance by reducing the number of forward branch instructions

if (x != 0)	CMP r3, #0	CMP r3, #0
a = b+c;	BEQ skip	ADDNE r0, r1, r2
else	ADD r0, r1, r2	SUBEQ r0, r1, r2
a = b-c;	B afterskip	
	skip	
	SUB r0, r1, r2	
	afterskip	

Can also be used to optimize a countdown loop (where the loop variable decrements from a positive number to zero)

loop	loop
.
SUB r1, r1, #1	SUBS r1, r1, #1
CMP r1, #0	BNE loop
BNE loop	

Single Register Data Transfer

-) ARM is based on a “load/store” architecture
 - All operands should be in registers
 - Load instructions are used to move data from memory into registers
 - Store instructions are used to move data from registers to memory
 - Flexible – allow transfer of a word or a half-word or a byte to and from memory

<i>LDR/STR</i>	Word
<i>LDRB/STRB</i>	Byte
<i>LDRH/STRH</i>	Half-word
<i>LDRSB</i>	Signed byte load
<i>LDRSH</i>	Signed half-word load

-) Syntax:
 - *LDR<cond><size> Rd, <address>*
 - *STR<cond><size> Rd, <address>*

LDR and STR

-) *LDR* and *STR* instructions can load and store data on a boundary alignment that is the same as the datatype size being loaded or stored.
-) *LDR* can only load 32-bit words on a memory address that is a multiple of 4 bytes – 0, 4, 8, and so on
-) *LDR r0, [r1]*
 - Loads register *r0* with the contents of the memory address pointed to by register *r1*
-) *STR r0, [r1]*
 - Stores the contents of register *r0* to the memory address pointed to by register *r1*
-) Register *r1* is called the **base address register**.

Addressing Modes

-) ARM provides three addressing modes:
 - Preindex with writeback
 - Preindex
 - Postindex
-) Preindex mode useful for accessing a single element in a data structure
-) Postindex and preindex with writeback useful for traversing an array

Addressing Modes

) Preindex with writeback

- Calculates address from a base register plus address offset
- Updates the address in the base register with the new address
- This is the address used to access memory
- Example: `LDR r0, [r1, #4]!`

) Preindex

- Same as preindex with writeback, but does not update the base register
- Example: `LDR r0, [r1, #4]`

) Postindex

- Only updates the base register *after* the address is used
- Example: `LDR r0, [r1], #4`

Examples on Addressing Modes

BEFORE

r0 = 0x00000000

r1 = 0x00009000

mem32[0x00009000] = 0x01010101

mem32[0x00009004] = 0x02020202

Preindexing with writeback

LDR r0, [r1, #4]!

AFTER

r0 =

r1 =

Preindexing

LDR r0, [r1, #4]

AFTER

r0 =

r1 =

Postindexing

LDR r0, [r1], #4

AFTER

r0 =

r1 =

More on Addressing Modes

-) Address <address> accessed by LDR/STR is specified by **a base register plus an offset**.
-) Offset takes one of the three formats:

Immediate Offset is a number that can be added to or subtracted from the base register.

Example: `LDR r0,[r1, #8];` `r0 = mem[r1+8]`

`LDR r0,[r1, #-8];` `r0 = mem[r1-8]`

Register Offset is a general-purpose register that can be added to or subtracted from the base register.

Example: `LDR r0,[r1, r2];` `r0 = mem[r1+r2]`

`LDR r0,[r1, -r2];` `r0 = mem[r1-r2]`

Scaled Register Offset is a general-purpose register shifted by an immediate value and then added to or subtracted from the base register.

Example: `LDR r0,[r1, r2, LSL #2];` `r0 = mem[r1+4*r2]`

`LDR r0,[r1, r2, RRX];` `r0 = mem[r1+RRX(r2)]`

Why assembly?

-) Break the conventions of your usual compiler, which might allow some optimizations
 - Example: Temporarily breaking rules about memory allocation, threading, calling conventions etc.
-) Build interfaces between code fragments using incompatible conventions
 - Example: Code produced by different compilers
-) Gain access to unusual programming modes of your processor
 - Example: 16 bit mode to interface startup
-) Produce reasonably fast code for tight loops to cope with a bad non-optimizing compiler
-) Produce hand-optimized code perfectly tuned for your particular hardware setup, though not to someone else's

Why Assembly sucks

-) Long and tedious to write initially
-) Bug-prone and bugs can be very difficult to chase
-) Code can be fairly difficult to understand, modify and maintain.
-) Code is non-portable to other architectures.
-) Code is optimized only for a certain implementation of a specific architecture.
-) You spend more time on a few details and don't focus on algorithmic design, which is where the most optimization opportunities often lie.
-) Small changes in algorithmic design can completely invalidate all your existing code.
-) Commercial optimizing compilers can outperform hand-coded assembly.
-) "Compilers make it a lot easier to use complex data structures, compilers don't get bored halfway through, and generate reliably pretty good code." John Levine

Middle Ground

From Charles Fiterman on comp.compilers about
Human vs. Computer-generated assembly code:

-) The human should always win and here is why.
-) **First** the human writes the whole thing in a high level language.
-) **Second** he profiles it to find the hot spots where it spends its time.
-) **Third** he has the compiler produce assembly for those small sections of code.
-) **Fourth** he hand-tunes them looking for tiny improvements over the machine generated code.
-) The human wins because he can use the machine.

Χρηματοδότηση

Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα» του ΕΜΠ έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του υλικού.

Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ARM Development Tools

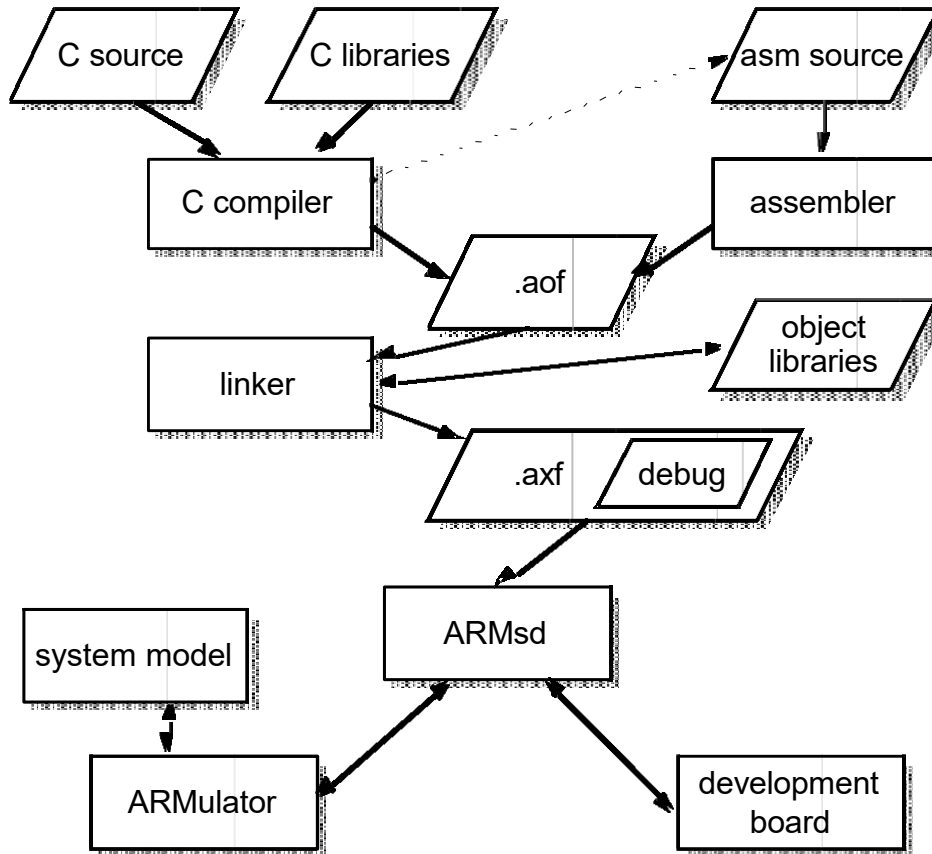
ARM Development Tools

- Cross-development
 - Windows PC
 - UNIX workstation
- Public domain compilers
 - gcc
- Commercial compilers
 - ARM C Compiler
 - IAR C Compiler

ARM Development Tools

- ARM C Compiler
- ARM Assembler
- Linker
- ARMsd
- ARMulator
- ARM Development Board
- Software Toolkit
- JumpStart

The structure of the ARM cross-development toolkit



ARM C Compiler

- ANSI C compliant
- ARM Procedure Call Standard
- Can produce
 - ARM object format
 - Assembly source output
 - Thumb code

ARM Assembler

- Produces ARM object format output
 - Can be linked with C compiler output
- Assembly source language
 - Near machine-level
 - Most assembly instruction → single ARM Thumb instruction

The Linker

- Resolves symbolic references between object files
- Extracts object modules from libraries as needed by program
- Different assembly for RAM, ROM, overlay, etc.
- Includes debug tables in output file

ARMsd

- ARM symbolic debugger
- For debugging ARM programs
 - Running under emulation (ARMuLator)
 - Running remotely on an ARM development board
 - Need JTAG test interface
 - Debugging embedded cores (difficult!)
 - Breakpoints / Watchpoints
 - Source level debugging
 - Original variable names from program

ARMulator

- ARMulator = ARM emulator
- A suite of programs that models the behavior of ARM processor core in software on a host system
- Different accuracy levels
 - Instruction-accurate
 - Cycle-accurate
 - Timing-accurate
- VHDL wrapper for interfacing into VHDL environment

ARM Development Board

- Components and interfaces to support development of ARM-based systems
- Includes
 - An ARM Core (e.g. ARM7TDMI)
 - Memory components (e.g. RAM, ROM, ...)
 - Programmable devices (e.g. FPGA)
- Supports
 - HW and SW development

Software Toolkit

- ARM Project Manager
 - Graphical front-end for building single library or executable image
 - From source files, object files, library files
 - Optimization for code size or execution time
 - Output in debug or release form
 - Target ARM processor (Thumb support?)

JumpStart

- VLSI Technology, Inc.
- X-Windows interface on workstation
- Same development tools

CSPC702-EMBEDDED SYSTEMS AND INTERNET OF THINGS(IOT)

UNIT – III Introduction to IoT

Syllabus

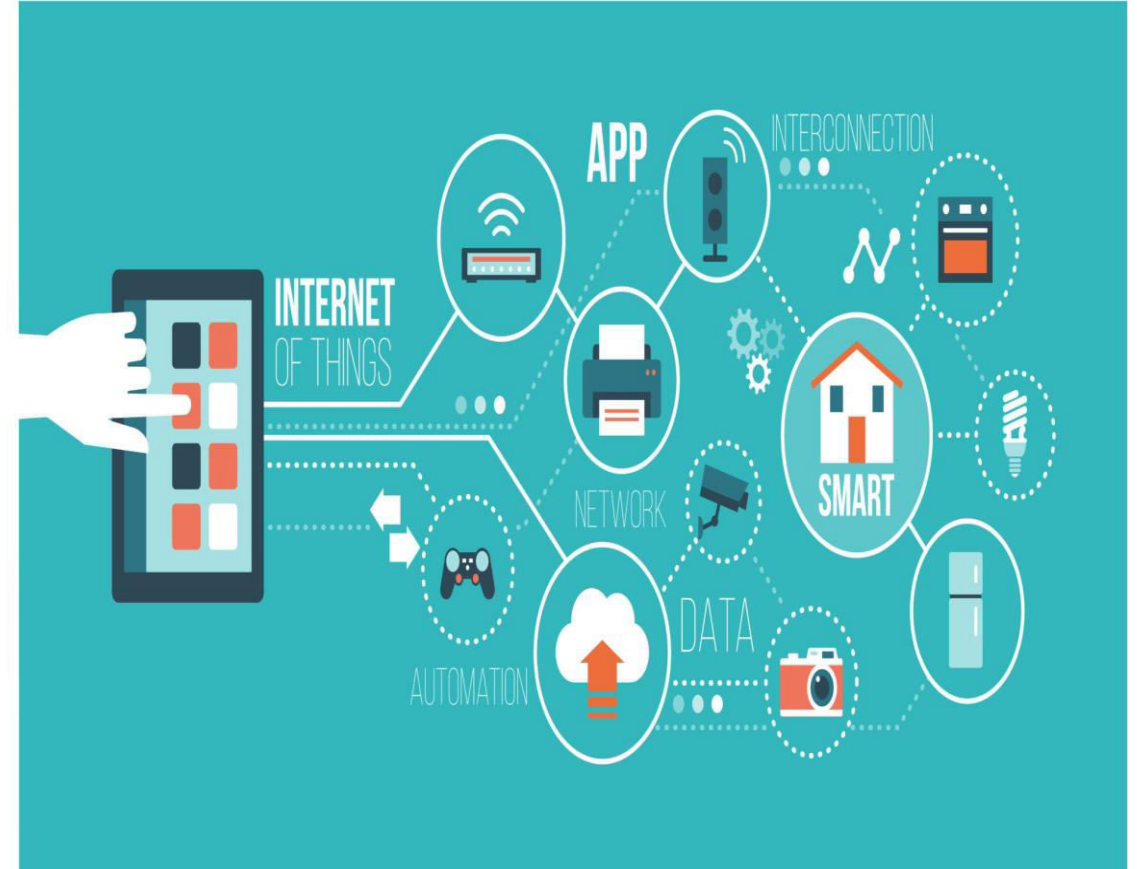
- **UNIT – III Introduction to IoT**
- Defining IoT, characteristics of IoT, physical design of IoT, logical design of IoT, functional blocks of IoT, communication models & APIs, machine to machine, difference between IoT and M2M, software defined network (SDN).

Introduction to IoT

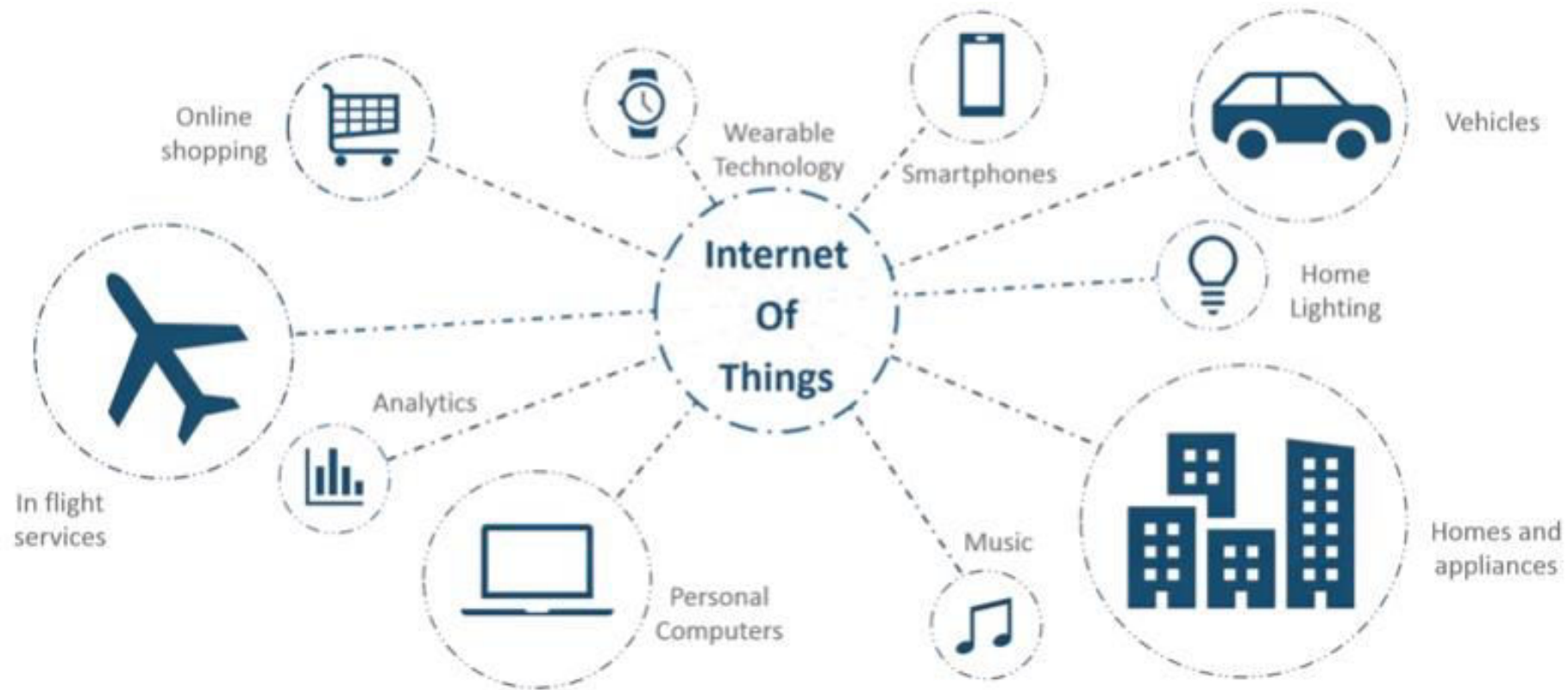
- **Internet of things (iot)** is a network of physical objects or people called "things" that are embedded with software, electronics, network, and sensors that allows these objects to collect and exchange data. The goal of IoT is to extend to internet connectivity from standard devices like computer, mobile, tablet to relatively dumb devices like a toaster.
- IoT makes virtually everything "smart," by improving aspects of our life with the power of data collection, AI algorithm, and networks. The thing in IoT can also be a person with a diabetes monitor implant, an animal with tracking devices, etc.
- Basically, **IoT is a network in which all physical objects are connected to the internet** through network devices or routers and exchange data. IoT allows objects to be controlled remotely across existing network infrastructure. IoT is a very good and intelligent technique which reduces human effort as well as easy access to physical devices. This technique also has autonomous control feature by which any device can control without any human interaction.

Introduction to IoT

- The 'Thing' in IoT can be any device with any kind of built-in-sensors with the ability to collect and transfer data over a network without manual intervention. The embedded technology in the object helps them to interact with internal states and the external environment, which in turn helps in decisions making process.



Introduction to IoT



- Connecting everyday things embedded with electronics, software, and sensors to internet enabling to collect and exchange data without human interaction called as the Internet of Things (IoT).
- The term "Things" in the Internet of Things refers to anything and everything in day to day life which is accessed or connected through the internet.

Introduction to IoT

- In a nutshell, IoT is a concept that connects all the devices to the internet and let them communicate with each other over the internet. IoT is a giant network of connected devices – all of which gather and share data about how they are used and the environments in which they are operated.
- By doing so, each of your devices will be learning from the experience of other devices, as humans do. IoT is trying to expand the interdependence in human- i.e *interact, contribute* and *collaborate* to things.
- A developer submits the application with a document containing the standards, logic, errors & exceptions handled by him to the tester. Again, if there are any issues Tester communicates it back to the Developer. It takes multiple iterations & in this manner a smart application is created.

Example

- Similarly, a room temperature sensor gathers the data and send it across the network, which is then used by multiple device sensors to adjust their temperatures accordingly. For example, refrigerator's sensor can gather the data regarding the outside temperature and accordingly adjust the refrigerator's temperature. Similarly, your air conditioners can also adjust its temperature accordingly. This is how devices can interact, contribute & collaborate.



Historical background of IoT

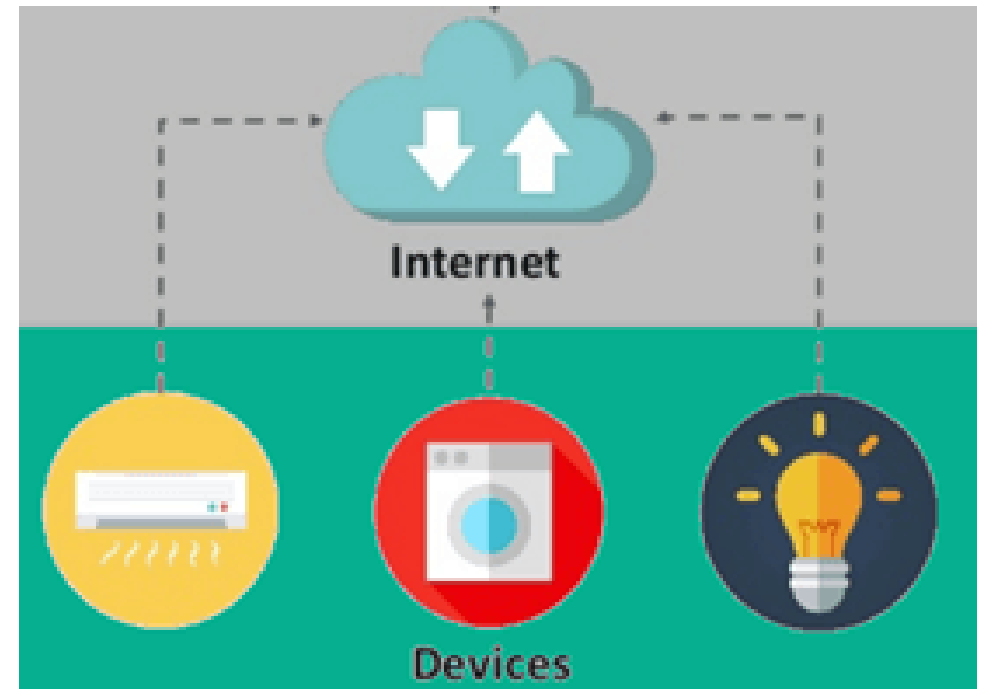
- 1970- the actual idea of connected devices was proposed
- 1990- John Romkey created a toaster which could be turned on/off over the internet
- 1995- Siemens introduced the first cellular module built for M2M
- 1999- the term "internet of things" was used by Kevin Ashton during his work at P&G which became widely accepted
- 2004 - the term was mentioned in famous publications like the guardian, Boston globe, and scientific American
- 2005-un's international telecommunications union (ITU) published its first report on this topic.
- 2008- the internet of things was born
- 2011- Gartner, the market research company, include "the internet of things" technology in their research

Historical background of IoT

- **In early 1982 the concept of the network of smart devices was discussed, with a modified coke machine.** This coke machine is modified at “Carnegie Mellon University” and becoming the first internet-connected appliance. This machine was able to report its inventory and whether newly loaded drinks were cold.
- **In 1994 Reza Raji explained the idea of IoT as “small packets of data to a large set of nodes, so as to integrate and automate everything from home appliances to entire factories”.** After that many companies proposed various solutions like microsoft’s at work or novell’s nest. Bill joy proposed device to device (D2D) communication as a part of his “six webs” frameworks at the world economic forum at davos in 1999.
- **The thought of internet of things first became popular in 1999.** British entrepreneur **Kevin Ashton** first used the term internet of things in 1999 while working at auto-id labs. Besides that near field communication, barcode scanners, QR code scanners and digital watermarking are the various devices which are working on IoT in the present scenario.

How IoT works?

- The entire IoT process starts with the devices themselves like smartphones, smartwatches, electronic appliances like TV, Washing Machine which helps you to communicate with the IoT platform.
- An IoT System have the following four fundamental components:
 - **Sensors/Devices**
 - **Connectivity**
 - **Data Processing**
 - **User Interface**



How IoT works?

- **1) Sensors/Devices:** Sensors or devices are a key component that helps you to collect live data from the surrounding environment. All this data may have various levels of complexities. It could be a simple temperature monitoring sensor, or it may be in the form of the video feed.
- A device may have various types of sensors which performs multiple tasks **apart** from sensing. Example, A mobile phone is a device which has multiple sensors like GPS, camera but your smartphone is not able to sense these things.
- **2) Connectivity:** All the collected data is sent to a cloud infrastructure. The sensors should be connected to the cloud using various mediums of communications. These communication mediums include mobile or satellite networks, Bluetooth, WI-FI, WAN, etc.

How IoT works?

- **3) Data Processing:** Once that data is collected, and it gets to the cloud, the software performs processing on the gathered data. This process can be just checking the temperature, reading on devices like AC or heaters. However, it can sometimes also be very complex like identifying objects, using computer vision on video.
- **4) User Interface:** The information needs to be available to the end-user in some way which can be achieved by triggering alarms on their phones or sending them notification through email or text message. The user sometimes might need an interface which actively checks their IoT system. For example, the user has a camera installed in his home. He wants to access video recording and all the feeds with the help of a web server.
- However, it's not always one-way communication. Depending on the IoT application and complexity of the system, the user may also be able to perform an action which may create cascading effects.
- For example, if a user detects any changes in the temperature of the refrigerator, with the help of IoT technology the user should be able to adjust the temperature with the help of their mobile phone.

Features of IOT

- The most important features of IoT on which it works are connectivity, analyzing, integrating, active engagement, and many more. Some of them are listed below:
- **Connectivity:** Connectivity refers to establish a proper connection between all the things of IoT to IoT platform it may be server or cloud. After connecting the IoT devices, it needs a high speed messaging between the devices and cloud to enable reliable, secure and bi-directional communication.
- **Analyzing:** After connecting all the relevant things, it comes to real-time analyzing the data collected and use them to build effective business intelligence. If we have a good insight into data gathered from all these things, then we call our system has a smart system.
- **Integrating:** IoT integrating the various models to improve the user experience as well.

Features of IOT

- **Artificial Intelligence:** IoT makes things smart and enhances life through the use of data. For example, if we have a coffee machine whose beans have going to end, then the coffee machine itself order the coffee beans of your choice from the retailer.
- **Sensing:** The sensor devices used in IoT technologies detect and measure any change in the environment and report on their status. IoT technology brings passive networks to active networks. Without sensors, there could not hold an effective or true IoT environment.
- **Active Engagement:** IoT makes the connected technology, product, or services to active engagement between each other.
- **Small Devices** – Devices, as predicted, have become smaller, cheaper, and more powerful over time. IoT exploits purpose-built small devices to deliver its precision, scalability, and versatility.
- **Endpoint Management:** It is important to be the endpoint management of all the IoT system otherwise, it makes the complete failure of the system. For example, if a coffee machine itself order the coffee beans when it goes to end but what happens when it orders the beans from a retailer and we are not present at home for a few days, it leads to the failure of the IoT system. So, there must be a need for endpoint management.

Benefits of IoT

- Since IoT allows devices to be controlled remotely across the internet, thus it created opportunities to directly connect & integrate the physical world to the computer-based systems using sensors and internet.
- The interconnection of these multiple embedded devices will be resulting in automation in nearly all fields and also enabling advanced applications.
- This is resulting in improved accuracy, efficiency and economic benefit with reduced human intervention.
- It encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities.

Benefits of IoT

The major benefits of IoT are:

- ***Improved Customer Engagement*** – IoT improves customer experience by automating the action. For e.g. any issue in the car will be automatically detected by the sensors. The driver, as well as the manufacturer, will be notified about it. Till the time driver reaches the service station, the manufacturer will make sure that the faulty part is available at the service station.
- ***Technical Optimization*** – IoT has helped a lot in improving technologies and making them better. The manufacturer can collect data from different car sensors and analyze them to improve their design and make them much more efficient.
- ***Reduced Waste*** – Our current insights are superficial, but IoT provides real-time information leading to effective decision making & management of resources. For example, if a manufacturer finds fault in multiple engines, he can track the manufacturing plant of those engines and can rectify the issue with manufacturing belt.

Applications of IoT

- **Energy Applications:** The energy rates have raised to a great extent. Individuals and organisations, both are searching ways to reduce and control the consumption. IoT provides a way to not only monitor the energy usage at the appliance-level but also at the house-level, grid level or could be at the distribution level. Smart Meters & Smart Grid are used to monitor energy consumption. It also detects threats to the system performance and stability, which protect appliances from downtime and damages.
- **Healthcare Application:** Smartwatches and fitness devices have changed the frequency of health monitoring. People can monitor their own health at regular intervals. Not only this, now if a patient is coming to the hospital by ambulance, by the time he or she reaches the hospital his health report is diagnosed by doctors and the hospital quickly starts the treatment. The data gathered from multiple healthcare applications are now collected and used to analyze different disease and find its cure.

Applications of IoT

- **Education:** IoT provides education aids which helps in fulfilling the gaps in the education industry. It not only improves the quality of education but also optimizes the cost and improves the management by taking into consideration students response and performance.
- **Government:** Governments are trying to build smart cities using IoT solutions. IoT enhances armed force systems and services. It provides better security across the borders through inexpensive & high-performance devices. IoT helps government agencies to monitor data in real-time and improve their services like healthcare, transportation, education etc.
- **Air and Water Pollution:** Through various sensors, we can detect the pollution in the air and water by frequent sampling. This helps in preventing substantial contamination and related disasters. IoT allows operations to minimize the human intervention in farming analysis and monitoring. Systems automatically detect changes in crops, soil, environment, and more.

Applications of IoT

- **Transportation:** IoT has changed the transportation sector. Now, we have self-driving cars with sensors, traffic lights that can sense the traffic and switch automatically, parking assistance, giving us the location of free parking space etc. Also, various sensors in your vehicle indicate you about the current status of your vehicle, so that you don't face any issues while travelling.
- **Marketing your product:** Using IoT, organizations can better analyze & respond to customer preferences by delivering relevant content and solutions. It helps in improving business strategies in the real-time.
- Now that we are aware of the powerful IoT solutions that have been astoundingly impacting various domains, let's take a deep dive and understand Raspberry Pi, which is commonly used to prepare IoT solutions. After understanding Raspberry Pi we will be creating an IoT application.

Application Domains

IoT is currently found in four different popular domains:

- 1) Manufacturing/Industrial business - 40.2%
- 2) Healthcare - 30.3%
- 3) Security - 7.7%
- 4) Retail - 8.3%

Modern Applications:

- Smart Grids
- Smart cities
- Smart homes
- Healthcare
- Earthquake detection
- Radiation detection/hazardous gas detection
- Smartphone detection
- Water flow monitoring

Challenges of Internet of Things (IoT)

At present IoT is faced with many challenges, such as:

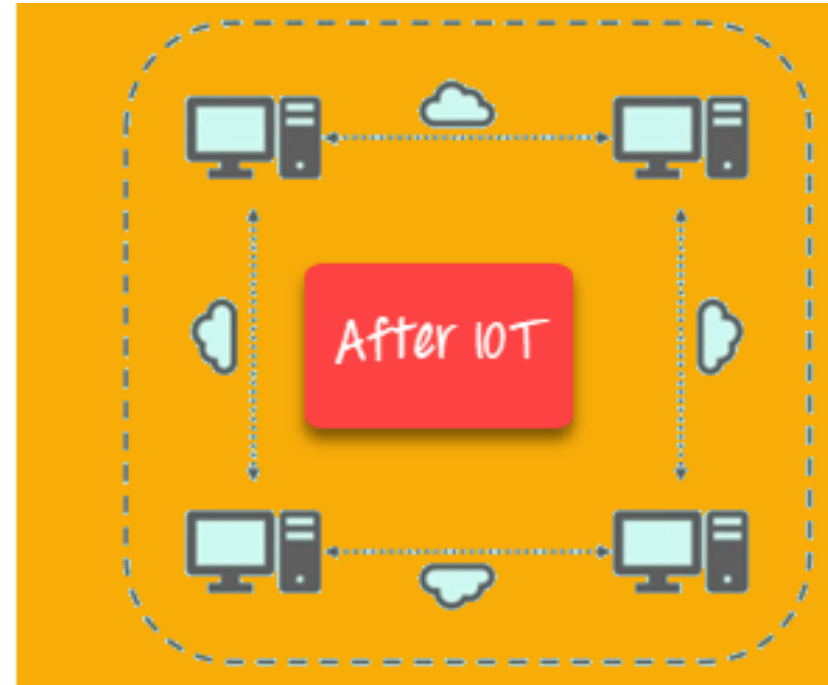
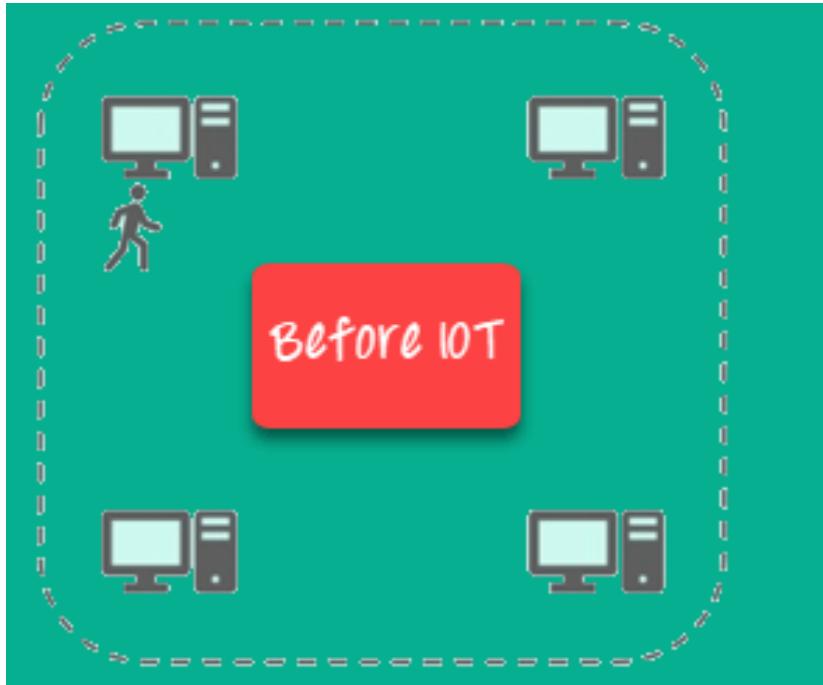
- Insufficient [testing](#) and updating
- Concern regarding data security and privacy
- Software complexity
- Data volumes and interpretation
- Integration with AI and automation
- Devices require a constant power supply which is difficult
- Interaction and short-range communication

Advantages of IoT

Key benefits of IoT technology are as follows:

- **Technical Optimization:** IoT technology helps a lot in improving technologies and making them better. Example, with IoT, a manufacturer is able to collect data from various car sensors. The manufacturer analyzes them to improve its design and make them more efficient.
- **Improved Data Collection:** Traditional data collection has its limitations and its design for passive use. IoT breaks it out of those spaces, and places it exactly where humans really want to go to analyze our world. It allows an accurate picture of everything. IoT facilitates immediate action on data.
- **Reduced Waste:** IoT offers real-time information leading to effective decision making & management of resources. For example, if a manufacturer finds an issue in multiple car engines, he can track the manufacturing plan of those engines and solves this issue with the manufacturing belt.
- **Improved Customer Engagement:** IoT allows you to improve customer experience by detecting problems and improving the process.

Advantages of IoT



Disadvantages of IoT

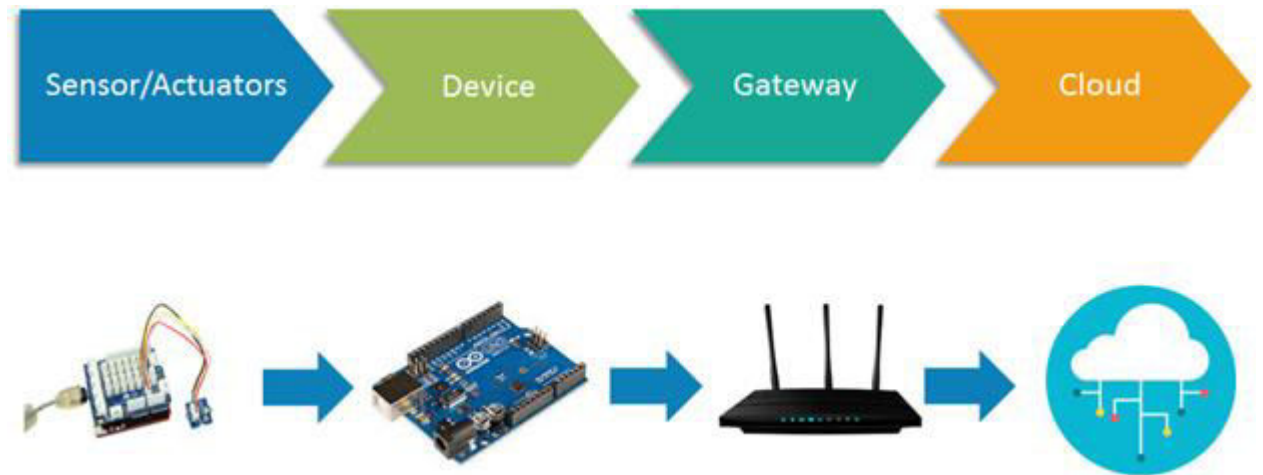
- **Security:** IoT technology creates an ecosystem of connected devices. However, during this process, the system may offer little authentication control despite sufficient security measures.
- **Privacy:** The use of IoT, exposes a substantial amount of personal data, in extreme detail, without the user's active participation. This creates lots of privacy issues.
- **Flexibility:** There is a huge concern regarding the flexibility of an IoT system. It is mainly regarding integrating with another system as there are many diverse systems involved in the process.
- **Complexity:** The design of the IoT system is also quite complicated. Moreover, it's deployment and maintenance also not very easy.
- **Compliance:** IoT has its own set of rules and regulations. However, because of its complexity, the task of compliance is quite challenging.

IoT Architecture

- Basically, there are three IoT architecture layers:
- 1. The client side (IoT Device Layer)
- 2. Operators on the server side (IoT Getaway Layer)
- 3. A pathway for connecting clients and operators (IoT Platform Layer)
- In fact, addressing the needs of all these layers is crucial on all the stages of IoT architecture. Being the basis of feasibility criterion, this consistency makes the result designed really work. In addition, the fundamental features of sustainable IoT architecture include functionality, scalability, availability, and maintainability. Without addressing these conditions, the result of IoT architecture is a failure.

IoT Architecture

- In the context of the Internet of Things, the architecture is a framework that defines the physical components, the functional organization and configuration of the network, operational procedures and the data formats to be used.
- The IoT architecture technology mainly consists of four major components:
 - Sensors/Devices
 - Gateways and Networks
 - Cloud/Management Service Layer
 - Application Layer

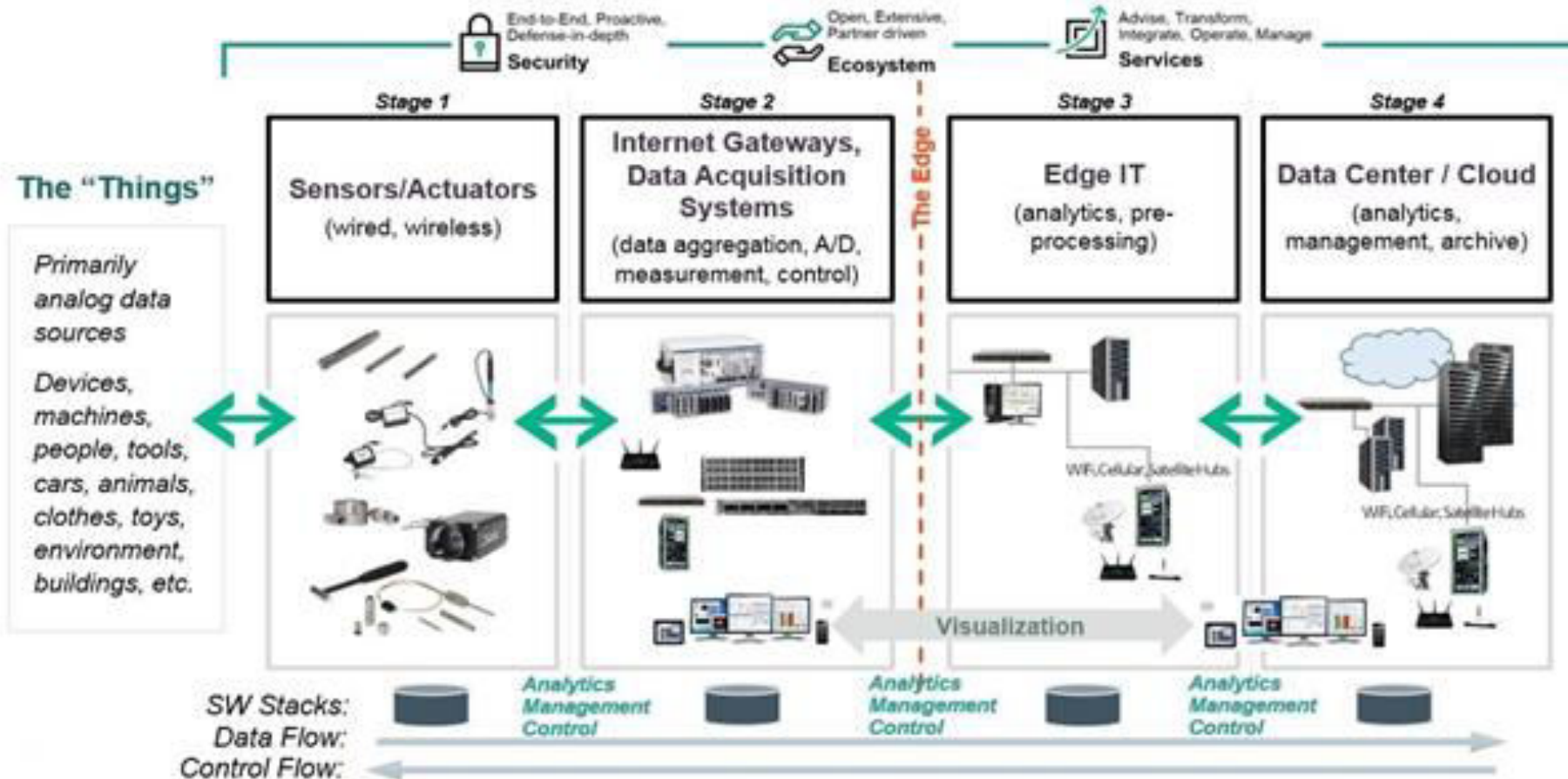


IoT Architecture

- **Stages of IoT Solutions Architecture**
- There are several layers of IoT built upon the capability and performance of IoT elements that provides the optimal solution to the business enterprises and end-users. The IoT architecture is a fundamental way to design the various elements of IoT, so that it can deliver services over the networks and serve the needs for the future.
- Following are the primary stages (layers) of IoT that provides the solution for IoT architecture.
- **Sensors/Actuators:** Sensors or Actuators are the devices that are able to emit, accept and process data over the network. These sensors or actuators may be connected either through wired or wireless. This contains GPS, Electrochemical, Gyroscope, RFID, etc. Most of the sensors need connectivity through sensors gateways. The connection of sensors or actuators can be through a Local Area Network (LAN) or Personal Area Network.
- **Gateways and Data Acquisition:** As the large numbers of data are produced by this sensors and actuators need the high-speed Gateways and Networks to transfer the data. This network can be of type Local Area Network (LAN such as WiFi, Ethernet, etc.), Wide Area Network (WAN such as GSM, 5G, etc.).
- **Edge IT:** Edge in the IoT Architecture is the hardware and software gateways that analyze and pre-process the data before transferring it to the cloud. If the data read from the sensors and gateways are not changed from its previous reading value then it does not transfer over the cloud, this saves the data used.
- **Data center/ Cloud:** The Data Center or Cloud comes under the Management Services which process the information through analytics, management of device and security controls. Beside this security controls and device management the cloud transfer the data to the end users application such as Retail, Healthcare, Emergency, Environment, and Energy, etc.

IoT Architecture

The 4 Stage IoT Solutions Architecture



IoT Architecture

- **Stage 1. Networked things (wireless sensors and actuators)**
- The outstanding feature about sensors is their ability to convert the information obtained in the outer world into data for analysis. In other words, it's important to start with the inclusion of sensors in the 4 stages of an IoT architecture framework to get information in an appearance that can be actually processed.
- For actuators, the process goes even further — these devices are able to intervene the physical reality. For example, they can switch off the light and adjust the temperature in a room.
- Because of this, sensing and actuating stage covers and adjusts everything needed in the physical world to gain the necessary insights for further analysis.

IoT Architecture

- **Stage 2. Sensor data aggregation systems and analog-to-digital data conversion**
- Even though this stage of IoT architecture still means working in a close proximity with sensors and actuators, Internet gateways and data acquisition systems (DAS) appear here too. Specifically, the latter connect to the sensor network and aggregate output, while Internet gateways work through Wi-Fi, wired LANs and perform further processing.
- The vital importance of this stage is to process the enormous amount of information collected on the previous stage and squeeze it to the optimal size for further analysis. Besides, the necessary conversion in terms of timing and structure happens here.
- In short, Stage 2 makes data both digitalized and aggregated.

IoT Architecture

- **Stage 3. The appearance of edge IT systems**
- During this moment among the stages of IoT architecture, the prepared data is transferred to the IT world. In particular, edge IT systems perform enhanced analytics and pre-processing here. For example, it refers to machine learning and visualization technologies. At the same time, some additional processing may happen here, prior to the stage of entering the data center.
- Likewise, Stage 3 is closely linked to the previous phases in the building of an architecture of IoT. Because of this, the location of edge IT systems is close to the one where sensors and actuators are situated, creating a wiring closet. At the same time, the residing in remote offices is also possible.

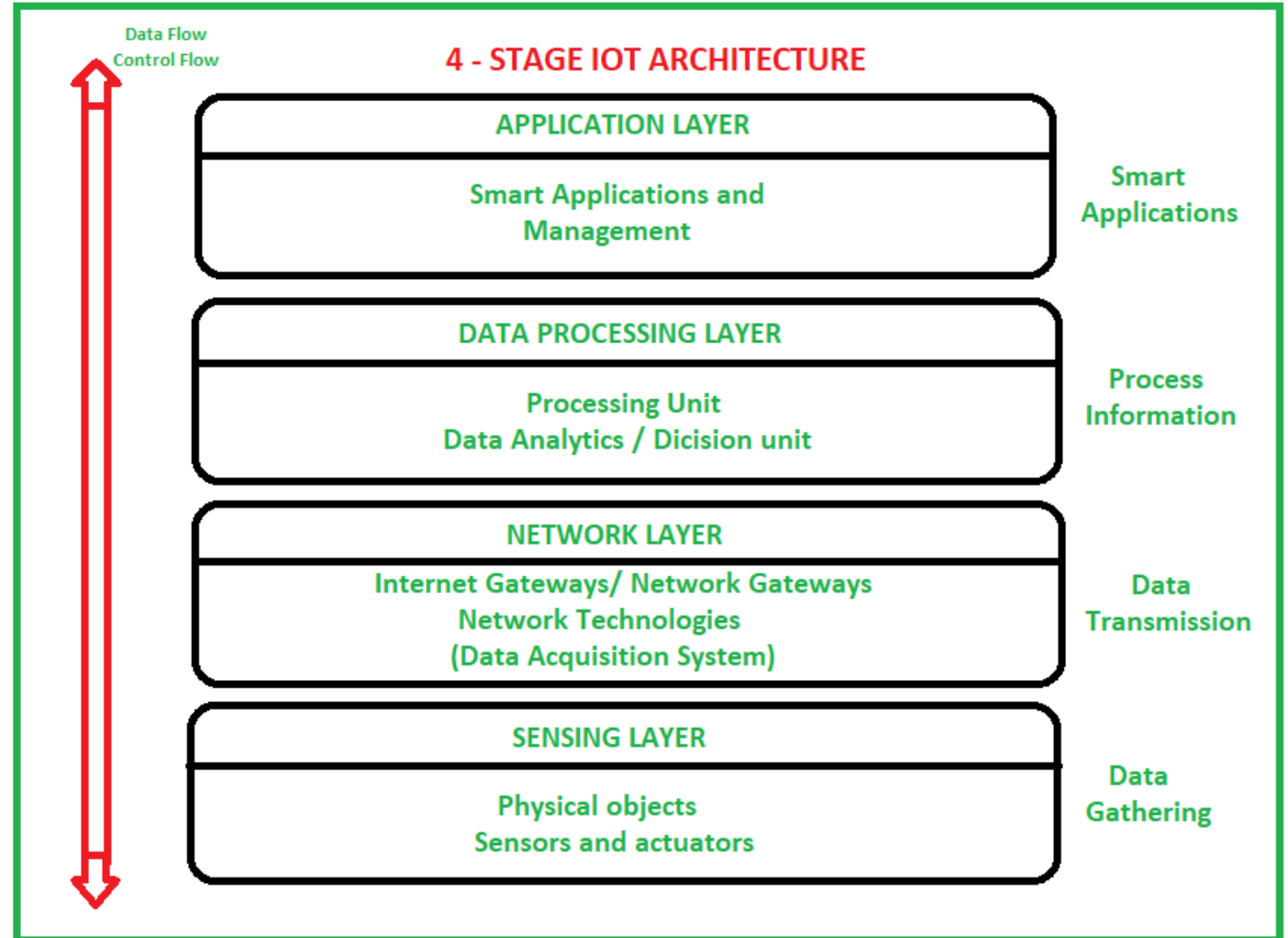
IoT Architecture

- **Stage 4. Analysis, management, and storage of data**
- The main processes on the last stage of IoT architecture happen in data center or cloud. Precisely, it enables in-depth processing, along with a follow-up revision for feedback. Here, the skills of both IT and OT (operational technology) professionals are needed. In other words, the phase already includes the analytical skills of the highest rank, both in digital and human worlds. Therefore, the data from other sources may be included here to ensure an in-depth analysis.
- After meeting all the quality standards and requirements, the information is brought back to the physical world — but in a processed and precisely analyzed appearance already.

Basic fundamental architecture of IoT i.e., 4 Stage IoT architecture

So, from the above image it is clear that there is 4 layers are present that can be divided as follows:

- Sensing Layer,
- Network Layer,
- Data processing Layer and
- Application Layer.



IoT Architecture

- These are explained as following below.
- **Sensing Layer** —
Sensors, actuators, devices are present in this Sensing layer. These Sensors or Actuators accepts data(physical/environmental parameters), processes data and emits data over network.
- **Network Layer** —
Internet/Network gateways, Data Acquisition System (DAS) are present in this layer. DAS performs data aggregation and conversion function (Collecting data and aggregating data then converting analog data of sensors to digital data etc). Advanced gateways which mainly opens up connection between Sensor networks and Internet also performs many basic gateway functionalities like malware protection, and filtering also some times decision making based on inputted data and data management services, etc.

IoT Architecture

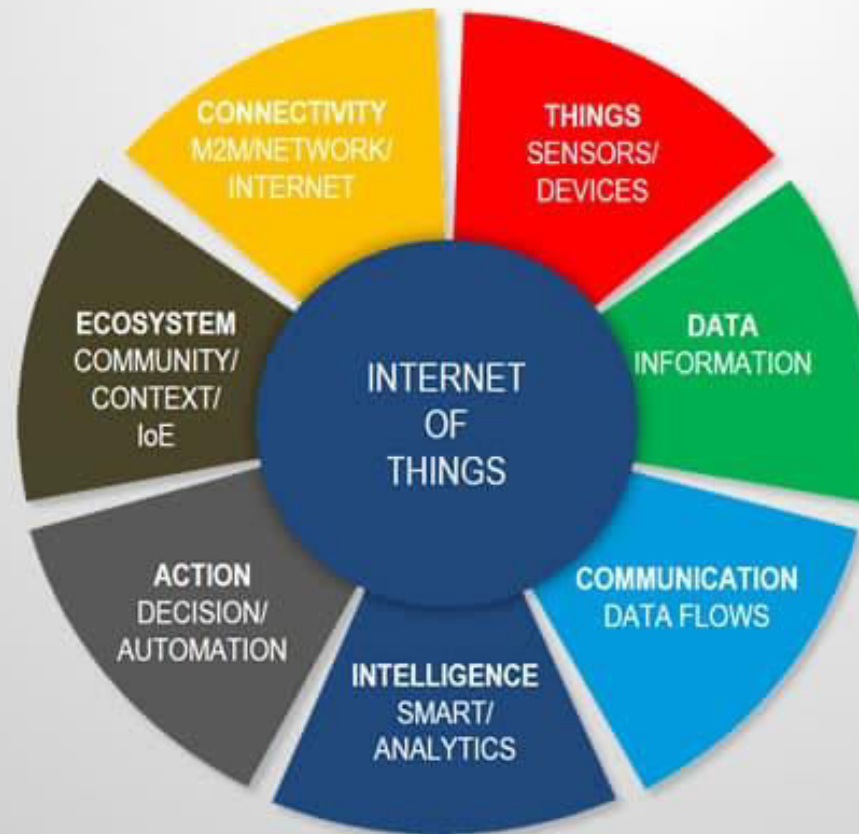
- **Data processing Layer** –
This is processing unit of IoT ecosystem. Here data is analyzed and pre-processed before sending it to data center from where data is accessed by software applications often termed as business applications where data is monitored and managed and further actions are also prepared. So here Edge IT or edge analytics comes into picture.
- **Application Layer** –
This is last layer of 4 stages of IoT architecture. Data centers or cloud is management stage of data where data is managed and is used by end-user applications like agriculture, health care, aerospace, farming, defense, etc.

Characteristics of IoT

- Massively scalable and efficient.
- IP-based addressing will no longer be suitable in the upcoming future.
- An abundance of physical objects is present that does not use IP, so IoT is made possible.
- Devices typically consume less power. When not in use, they should be automatically programmed to sleep.
- A device that is connected to another device right now may not be connected in another instant of time.
- Intermittent connectivity – IoT devices aren't always connected. In order to save bandwidth and battery consumption, devices will be powered off periodically when not in use. Otherwise, connections might turn unreliable and thus prove to be inefficient.

Characteristics of IoT

DEFINING IOT: 7 CHARACTERISTICS



Characteristics of IoT

- **There are 7 crucial IoT characteristics:**
- **Connectivity.** This doesn't need too much further explanation. With everything going on in IoT devices and hardware, with sensors and other electronics and connected hardware and control systems there needs to be a connection between various levels.
- **Things.** Anything that can be tagged or connected as such as it's designed to be connected. From sensors and household appliances to tagged livestock. Devices can contain sensors or sensing materials can be attached to devices and items.
- **Data.** Data is the glue of the Internet of Things, the first step towards action and intelligence.
- **Communication.** Devices get connected so they can communicate data and this data can be analyzed. Communication can occur over short distances or over a long range to very long range. Examples: Wi-Fi, [LPWA](#) network technologies such as [LoRa](#) or [NB-IoT](#).
- **Intelligence.** The aspect of intelligence as in the sensing capabilities in IoT devices and the intelligence gathered from big data analytics (also artificial intelligence).
- **Action.** The consequence of intelligence. This can be manual action, action based upon debates regarding phenomena (for instance in [smart factory](#) decisions) and automation, often the most important piece.
- **Ecosystem.** The place of the Internet of Things from a perspective of other technologies, communities, goals and the picture in which the Internet of Things fits. The Internet of Everything dimension, the platform dimension and the need for solid partnerships.

Physical design of IoT

- The **physical design** of an [IoT](#) system is referred to the **Things/Devices** and protocols that used to build an IoT system. all these things/Devices are called Node Devices and every device has a unique identity that performs remote sensing, actuating, and monitoring work. and the protocols that used to established communication between the Node devices and server over the internet.

Things

- Basically Things refers to IoT Devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities. Things are is main part of IoT Application.
- IoT Devices can be various type, Sensing Devices, Smart Watches, Smart Electronics appliances, Wearable Sensors, Automobiles, and industrial machines.
- These devices generate data in some forms or the other which when processed by data analytics systems leads to useful information to guide further actions locally or remotely.

Physical design of IoT

Physical Design of IoT

Things

Protocols

- Things/Devices are used to build a connection, process data, provide interfaces, provide storage, and provide graphics interfaces in an IoT system. all these generate data in a form that can be analyzed by an analytical system and program to perform operations and used to improve the system.
- for example temperature sensor that is used to analyze the temperature generates the data from a location and then determined by algorithms.

Physical design of IoT

Generic Block Diagram of IoT Devices

Connectivity

Devices like USB host and ETHERNET are used for connectivity between the devices and server.

Processor

A processor like a CPU and other units are used to process the data. these data are further used to improve the decision quality of an IoT system.

Audio/Video Interfaces

An interface like HDMI and RCA devices is used to record audio and videos in a system.

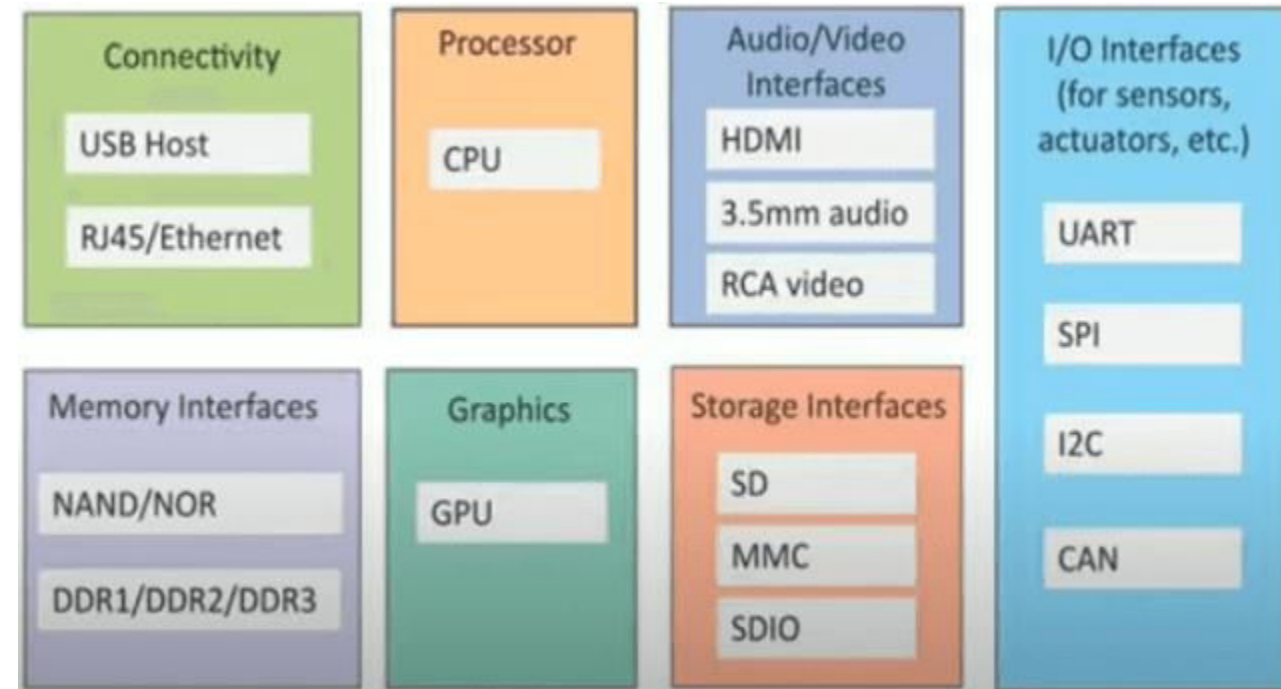
Input/Output interface

To giving input and output signals to sensors, and actuators we use things like UART, SPI, CAN, etc.

Storage Interfaces

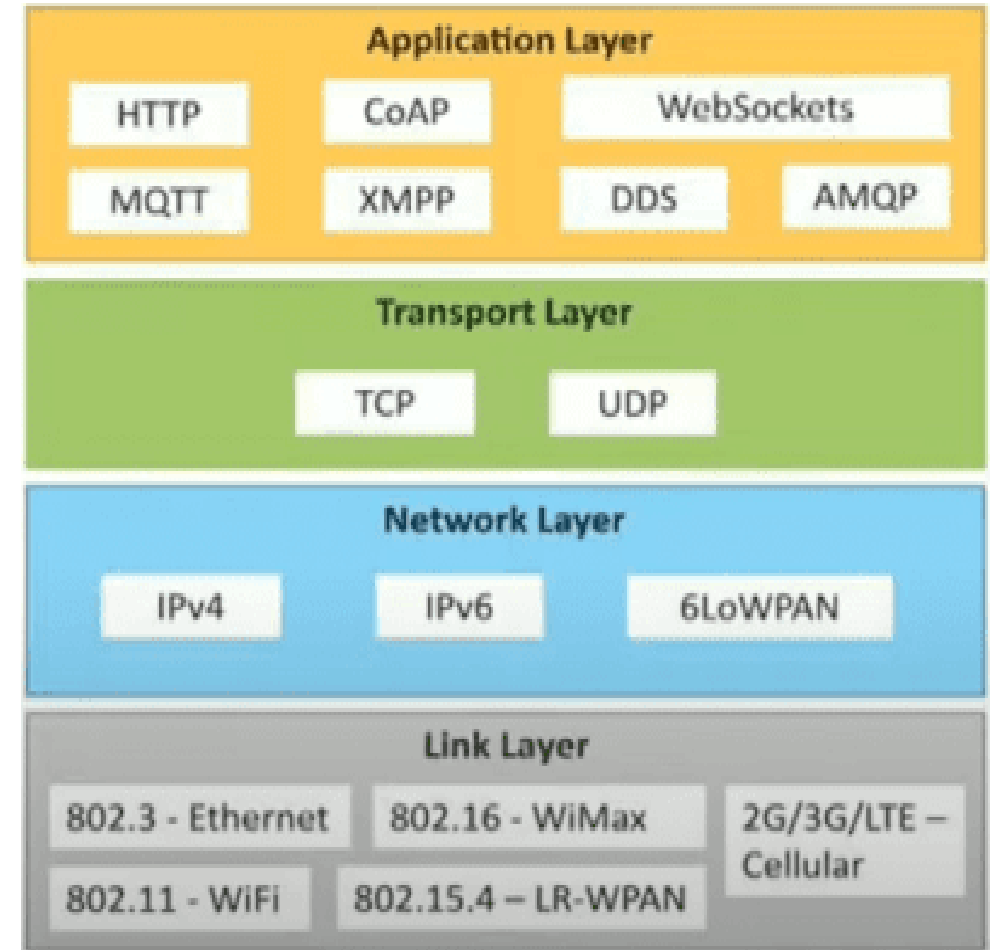
Things like SD, MMC, SDIO are used to store the data generated from an IoT device.

Other things like DDR, GPU are used to control the activity of an IoT system.



Physical design of IoT IoT Protocols

- These protocols are used to establish communication between a node device and server over the internet. it helps to send commands to an IoT device and receive data from an [IoT](#) device over the internet. we use different types of protocols that present on both the server and client-side and these protocols are managed by network layers like application, transport, network, and link layer.



Physical design of IoT

IoT Protocols (Application Layer)

- **Application Layer protocol**
- In this layer, protocols define how the data can be sent over the network with the lower layer protocols using the application interface. (or)
- Application layer protocols define how the applications interface with the lower layer protocols to send over their network.
- These protocols including
 - HTTP
 - CoAp
 - WebSocket
 - MQTT
 - XMPP
 - DDS
 - AMQP protocols.

Physical design of IoT

IoT Protocols (Application Layer)

- **HTTP** : *Hypertext Transfer Protocol (HTTP)* is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes.
- It makes a request to a server and then waits till it receives a response and in between the request server does not keep any data between two requests.
- HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response.
- HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests. Though often based on a TCP/IP layer, it can be used on any reliable transport layer, that is, a protocol that doesn't lose messages silently like UDP does. RUDP — the reliable update of UDP — is a suitable alternative.

Physical design of IoT

IoT Protocols (Application Layer)

- **CoAP** : CoAP-Constrained Application Protocol is a specialized Internet Application Protocol for constrained devices, as defined in RFC 7252. It enables devices to communicate over the Internet.
- It is defined as Constrained Application Protocol, and is a protocol intended to be used in very simple hardware. The protocol is especially targeted for constrained hardware such as 8-bits microcontrollers, low power sensors and similar devices that can't run on HTTP or TLS.
- It is a simplification of the HTTP protocol running on UDP, that helps save bandwidth. It is designed for use between devices on the same constrained network (e.g., low-power, lossy networks), between devices and general nodes on the Internet, and between devices on different constrained networks both joined by an internet.
- CoAP is also being used via other mechanisms, such as SMS on mobile communication networks.

Physical design of IoT

IoT Protocols (Application Layer)

- **WebSocket** : The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP.
- The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g., using XMLHttpRequest or <iframe>s and long polling).

Physical design of IoT

IoT Protocols (Application Layer)

- **MQTT** protocol runs on top of the TCP/IP networking stack. When clients connect and publish/subscribe, MQTT has different message types that help with the handshaking of that process.
- It is a machine-to-machine connectivity protocol that was designed as a publish/subscribe messaging transport and it is used for remote locations where a small code footprint is required.
- The MQTT header is two bytes and first byte is constant. In the first byte, you specify the type of message being sent as well as the QoS level, retain, and DUP (duplication) flags. The second byte is the remaining length field.

Physical design of IoT

IoT Protocols (Application Layer)

- **XMPP : Extensible Messaging and Presence Protocol (XMPP)** is a communication protocol for message-oriented middleware based on XML (Extensible Markup Language).
- It enables the near-real-time exchange of structured yet extensible data between any two or more network entities. Originally named **Jabber**, the protocol was developed by the eponymous open-source community in 1999 for near real-time instant messaging (IM), presence information, and contact list maintenance.
- Designed to be extensible, the protocol has been used also for publish-subscribe systems, signalling for VoIP, video, file transfer, gaming, the Internet of Things (IoT) applications such as the smart grid, and social networking services.

Physical design of IoT

IoT Protocols (Application Layer)

- **DDS** : The Data Distribution Service (DDS™) is a middleware protocol and API standard for data-centric connectivity from the Object Management Group® (OMG®). It integrates the components of a system together, providing low-latency data connectivity, extreme reliability, and a scalable architecture that business and mission-critical Internet of Things (IoT) applications need.
- In a distributed system, middleware is the software layer that lies between the operating system and applications. It enables the various components of a system to more easily communicate and share data. It simplifies the development of distributed systems by letting software developers focus on the specific purpose of their applications rather than the mechanics of passing information between applications and systems.

Physical design of IoT

IoT Protocols (Application Layer)

- **AMQP** : The AMQP – IoT protocols consist of a hard and fast of components that route and save messages within a broker carrier, with a set of policies for wiring the components together. The AMQP protocol enables patron programs to talk to the dealer and engage with the AMQP model. AMQP has the following three additives, which might link into processing chains in the server to create the favored capability.
 - **Exchange**: Receives messages from publisher primarily based programs and routes them to 'message queues'.
 - **Message Queue**: Stores messages until they may thoroughly process via the eating client software.
 - **Binding**: States the connection between the message queue and the change.

Physical design of IoT

IoT Protocols(Transport Layer)

- **Transport Layer**
- This layer provides functions such as error control, segmentation, flow control and congestion control.
- So this layer protocols provide end-to-end message transfer capability independent of the underlying network.
 - **TCP**
 - **UDP**

Physical design of IoT

IoT Protocols(Transport Layer)

- **TCP** : TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation through which application programs can exchange data.
- TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other. Together, TCP and IP are the basic rules defining the Internet. The Internet Engineering Task Force (IETF) defines TCP in the Request for Comment (RFC) standards document number 793.
- **UDP** : User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of Internet Protocol suite, referred as UDP/IP suite. Unlike TCP, it is unreliable and connectionless protocol. So, there is no need to establish connection prior to data transfer.

Physical design of IoT IoT Protocols(Network Layer)

- **Network Layer**
- This layer is used to send datagrams from the source network to the destination network. we use IPv4 and IPv6 protocols as a host identification that transfers data in packets.
 - **IPv4**
 - **IPv6**
 - **6LoWPAN**

Physical design of IoT IoT Protocols(Network Layer)

- **IPv4 :**
- An **Internet Protocol address (IP address)** is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. An IP address serves two main functions: host or network interface identification and location addressing.
- Internet Protocol version 4 (IPv4) defines an IP address as a 32-bit number. However, because of the growth of the Internet and the depletion of available IPv4 addresses, a new version of IP (IPv6), using 128 bits for the IP address, was standardized in 1998. IPv6 deployment has been ongoing since the mid-2000s.

Physical design of IoT

IoT Protocols(Network Layer)

- **IPv6 : Internet Protocol version 6 (IPv6)** is the most recent version of the Internet Protocol (IP), the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet. IPv6 was developed by the Internet Engineering Task Force (IETF) to deal with the long-anticipated problem of IPv4 address exhaustion. IPv6 is intended to replace IPv4. In December 1998, IPv6 became a Draft Standard for the IETF, who subsequently ratified it as an Internet Standard on 14 July 2017. IPv6 uses a 128-bit address, theoretically allowing 2^{128} , or approximately 3.4×10^{38} addresses.

Physical design of IoT

IoT Protocols(Network Layer)

- **6LoWPAN** : 6LoWPAN is an acronym of *IPv6 over Low-Power Wireless Personal Area Networks*. 6LoWPAN is the name of a concluded working group in the Internet area of the IETF. 6LoWPAN is a somewhat contorted acronym that combines the latest version of the Internet Protocol (IPv6) and Low-power Wireless Personal Area Networks (LoWPAN). 6LoWPAN, therefore, allows for the smallest devices with limited processing ability to transmit information wirelessly using an internet protocol. 6LoWPAN can communicate with 802.15.4 devices as well as other types of devices on an IP network link like WiFi.

Physical design of IoT IoT Protocols(Link Layer)

- **Link Layer**
- Link layer protocols determine how data is physically sent over the network's physical layer or medium (Coxial cable or other or radio wave). Link Layer determines how the packets are coded and signaled by the hardware device over the medium to which the host is attached (eg. coxial cable).
- **Ethernet**
- It is a set of technologies and protocols that are used primarily in LANs. it defines the physical layer and the medium access control for wired ethernet networks.
- **WiFi**
- It is a set of LAN protocols and specifies the set of media access control and physical layer protocols for implementing wireless local area networks.

Physical design of IoT IoT Protocols(Link Layer)

- Here we explain some Link Layer Protocols:
- **802.3 – Ethernet** : Ethernet is a set of technologies and protocols that are used primarily in LANs. It was first standardized in 1980s by IEEE 802.3 standard. IEEE 802.3 defines the physical layer and the medium access control (MAC) sub-layer of the data link layer for wired Ethernet networks. Ethernet is classified into two categories: classic Ethernet and switched Ethernet.
- **802.11 – WiFi** : IEEE 802.11 is part of the IEEE 802 set of LAN protocols, and specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) Wi-Fi computer communication in various frequencies, including but not limited to 2.4 GHz, 5 GHz, and 60 GHz frequency bands.

Physical design of IoT

IoT Protocols(Link Layer)

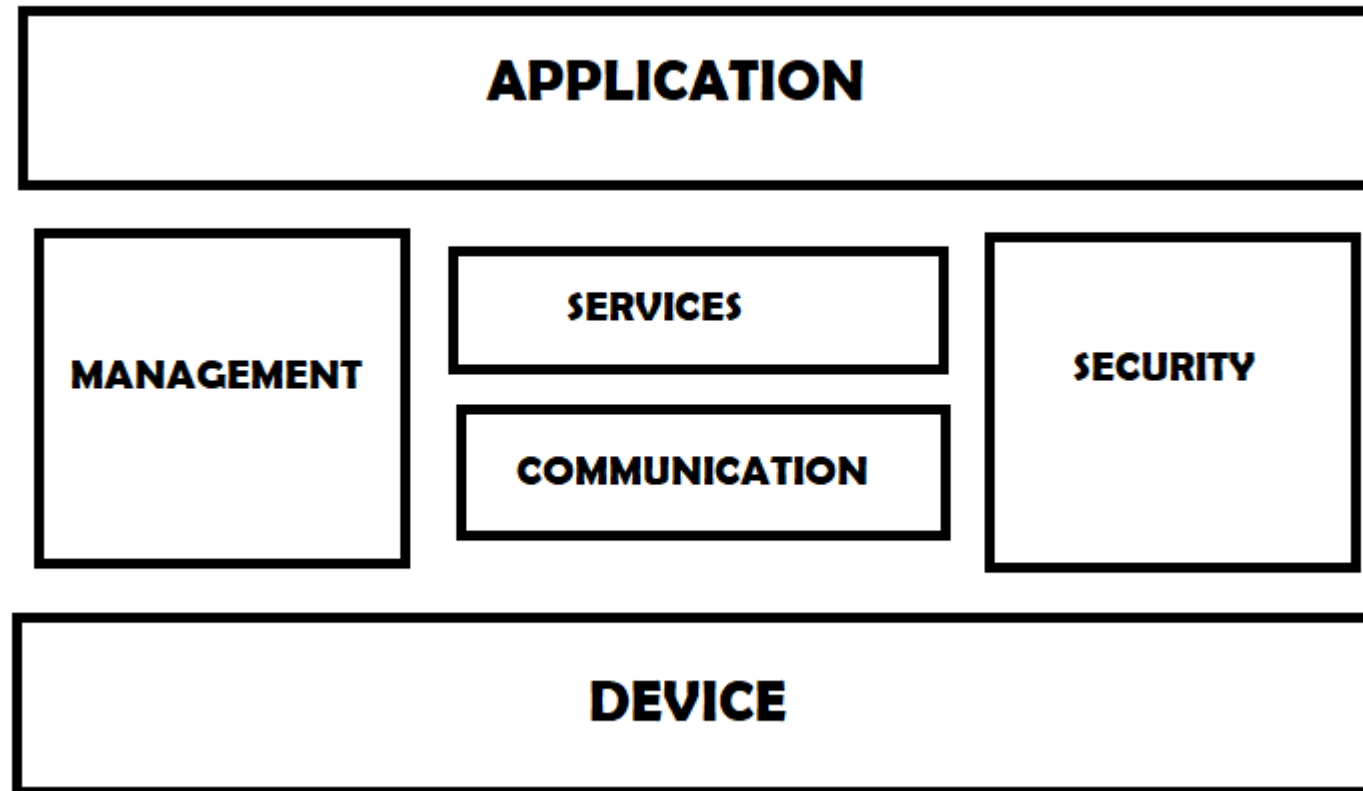
- **802.16 – Wi-Max** : The standard for WiMAX technology is a standard for Wireless Metropolitan Area Networks (WMANs) that has been developed by working group number 16 of IEEE 802, specializing in point-to-multipoint broadband wireless access. Initially 802.16a was developed and launched, but now it has been further refined. 802.16d or 802.16-2004 was released as a refined version of the 802.16a standard aimed at fixed applications. Another version of the standard, 802.16e or 802.16-2005 was also released and aimed at the roaming and mobile markets.
- **802.15.4 -LR-WPAN** : A collection of standards for Low-rate wireless personal area network. The IEEE's 802.15.4 standard defines the MAC and PHY layer used by, but not limited to, networking specifications such as Zigbee[®], 6LoWPAN, Thread, WiSUN and MiWi[™] protocols. The standards provide low-cost and low-speed communication for power constrained devices.
- **2G/3G/4G- Mobile Communication** : These are different types of telecommunication generations. IoT devices are based on these standards can communicate over the celluler networks.

Logical design of IoT

- Logical design of IoT system refers to an abstract representation of the entities & processes without going into the low-level specifics of the implementation.
- It uses **Functional Blocks**, **Communication Models**, and **Communication APIs** to implement a system.
- For understanding Logical Design of IoT, we describes given below terms.
 - IoT Functional Blocks
 - IoT Communication Models
 - IoT Communication APIs

Logical design of IoT- Functional blocks of IoT

- An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication and management.



Logical design of IoT- Functional blocks of IoT

- functional blocks are:
- **Device:** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
- **Communication:** Handles the communication for the IoT system.
- **Services:** services for device monitoring, device control service, data publishing services and services for device discovery.
- **Management:** this blocks provides various functions to govern the IoT system.
- **Security:** this block secures the IoT system and by providing functions such as authentication , authorization, message and content integrity, and data security.
- **Application:** This is an interface that the users can use to control and monitor various aspects of the IoT system. Application also allow users to view the system status and view or analyze the processed data.

These functional blocks consist of devices that provide monitoring control functions, handle communication between host and server, manage the transfer of data, secure the system using authentication and other functions, and interface to control and monitor various terms.

Logical design of IoT- IoT Communication Models

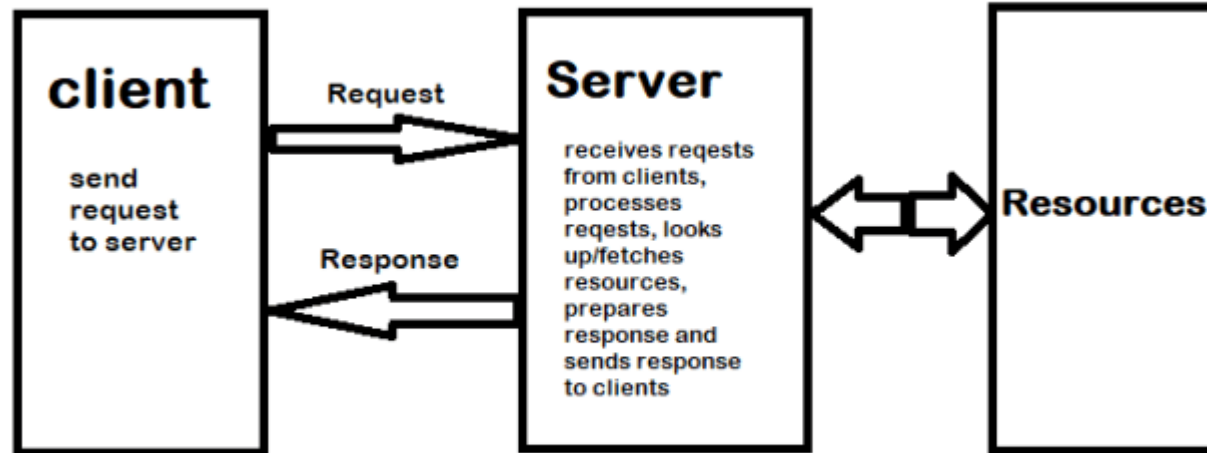
- There are several different types of models available in an IoT system that used to communicate between the system and server like the
 - Request-response model,
 - publish-subscribe model,
 - push-pull model and
 - exclusive pair model, etc.

Logical design of IoT- IoT Communication Models

- **Request-Response Model**

- Request-response model is communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves resource representation, prepares the response, and then sends the response to the client. Request-response is a stateless communication model and each request-response pair is independent of others.
- HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Logical design of IoT- IoT Communication Models



Request-Response Communication Model

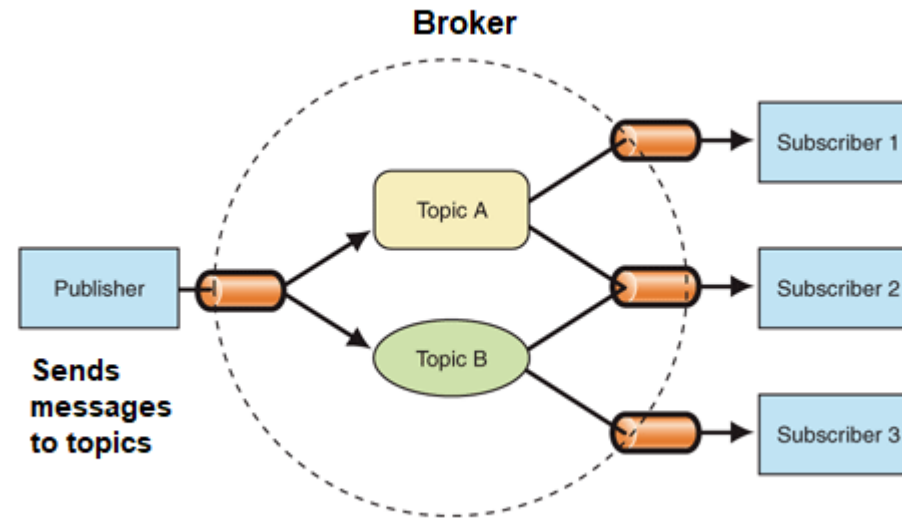
- **Example:** A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

Logical design of IoT- IoT Communication Models

- **Publish-Subscribe Model**
- Publish-Subscribe is a communication model that involves publishers, brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers.
- Consumers subscribe to the topics which are managed by the broker. When the broker receive data for a topic from the publisher, it sends the data to all the subscribed consumers.

Logical design of IoT- IoT Communication Models

- The Following figure shows the Publish-Subscribe Model

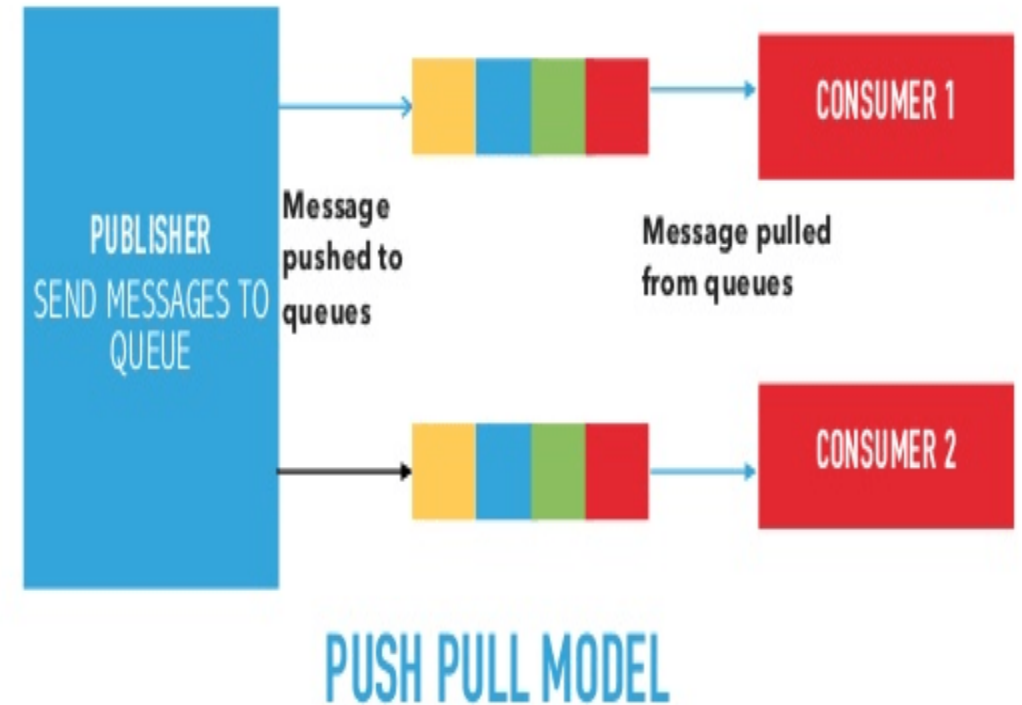


- **Example**
- On the website many times we subscribed to their newsletters using our email address. these email addresses managed by some third-party services and when a new article published on the website it directly sends to the broker and then the broker send these new data or post to all the subscribers.

Logical design of IoT- IoT Communication Models

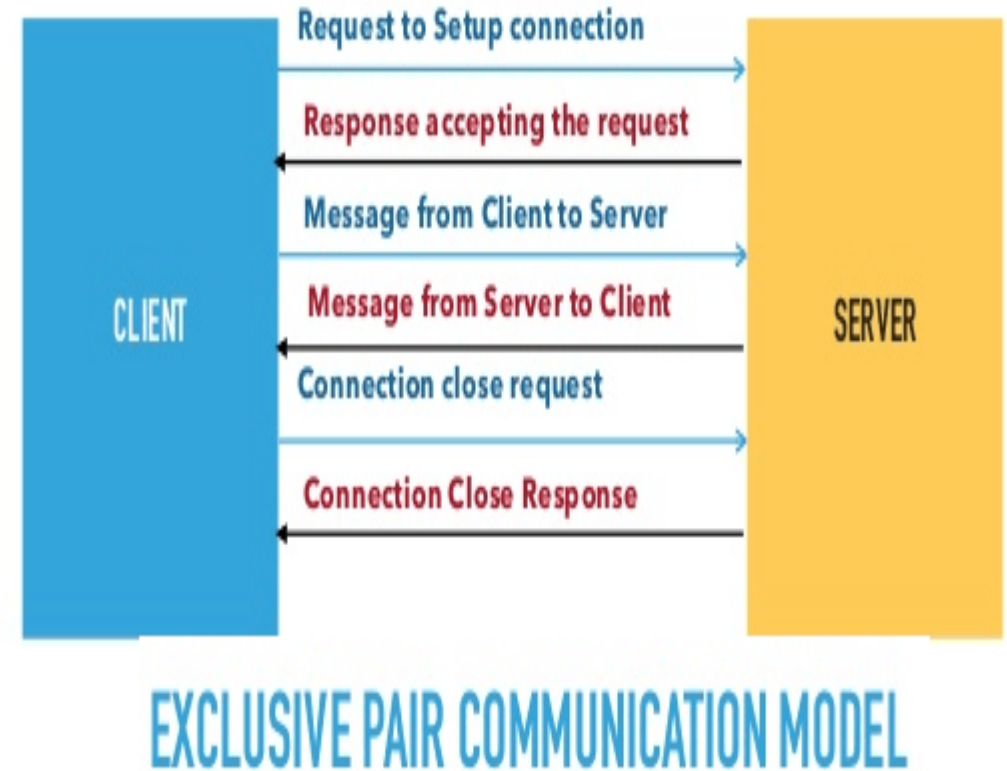
- **Push-Pull Model**

- Push-Pull is a communication model in which the data producers push the data to queues and the consumers Pull the data from the Queues.
- Producers do not need to be aware of the consumers. Queues help in decoupling the messaging between the Producers and Consumers.
- Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate rate at which the consumer pull data.
- **Example**
- When we visit a website we saw a number of posts that published in a queue and according to our requirements, we click on a post and start reading it.



Logical design of IoT- IoT Communication Models

- **Exclusive Pair Model**
- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server. Connection is setup it remains open until the client sends a request to close the connection. Client and server can send messages to each other after connection setup. Exclusive pair is stateful communication model and the server is aware of all the open connections.



Logical design of IoT- IoT communication APIs

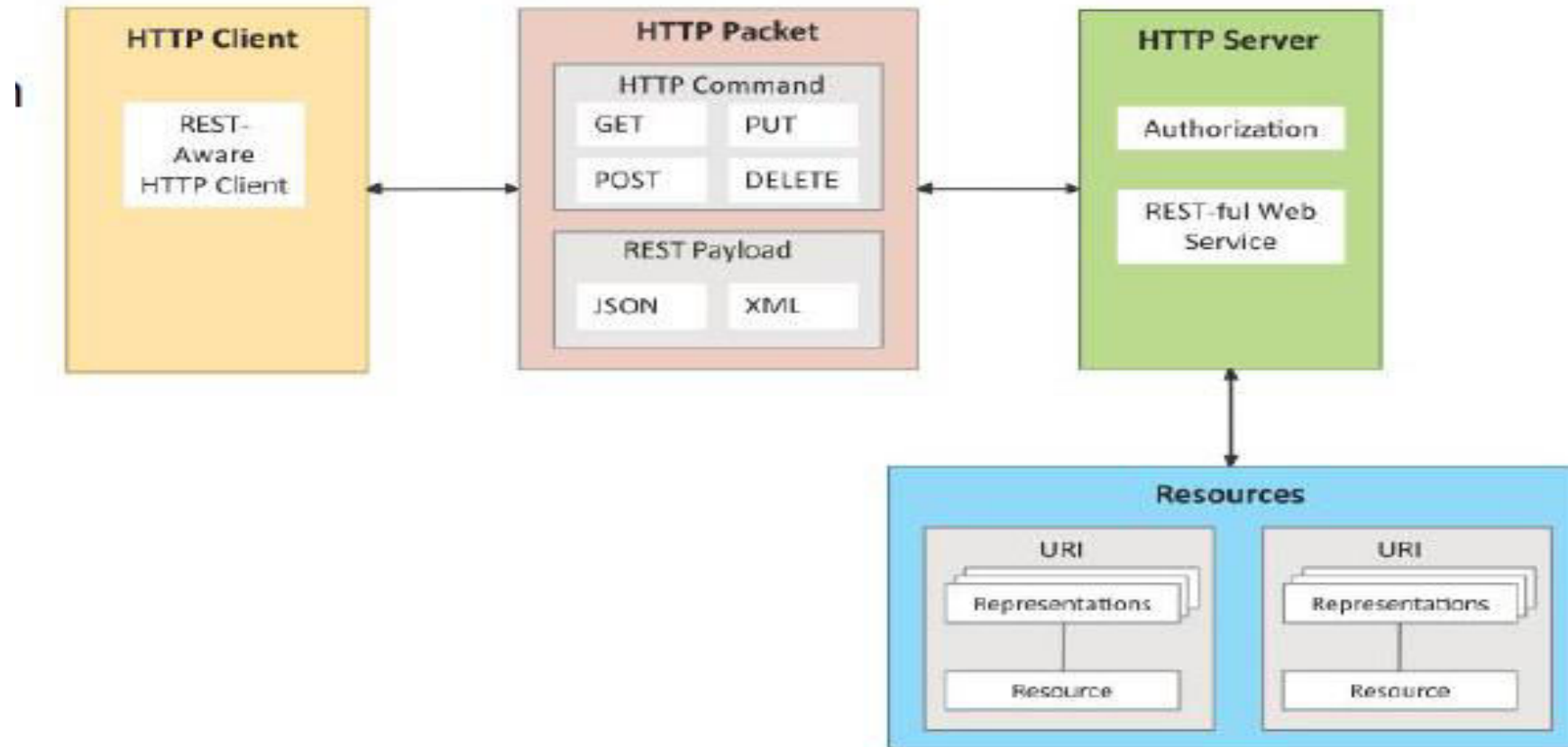
- Generally we used Two APIs For IoT Communication.
These IoT Communication APIs are:
 - REST-based Communication APIs
 - WebSocket-based Communication APIs

Logical design of IoT- IoT communication APIs

- **REST-based Communication APIs**
- Representational state transfer (REST) is a set of architectural principles by which you can design Web services the Web APIs that focus on systems's resources and how resource states are addressed and transferred.
- REST APIs that follow the request response communication model, the rest architectural constraint apply to the components, connector and data elements, within a distributed hypermedia system.

Logical design of IoT- IoT communication APIs

- The rest architectural constraint are as follows:



Logical design of IoT- IoT communication APIs

- **Client-server** – The principle behind the client-server constraint is the separation of concerns. for example clients should not be concerned with the storage of data which is concern of the serve.
- Similarly the server should not be concerned about the user interface, which is concern of the client.
- Separation allows client and server to be independently developed and updated.
- **Stateless** – Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- The session state is kept entirely on the client.

Logical design of IoT- IoT communication APIs

- **Cache-able** – Cache constraints requires that the data within a response to a request be implicitly or explicitly leveled as cache-able or non cache-able. If a response is cache-able, then a client cache is given the right to reuse that repsonse data for later, equivalent requests. caching can partially or completely eliminate some instructions and improve efficiency and scalability.
- **Layered system** – layered system constraints, constrains the behavior of components such that each component cannot see beyond the immediate layer with they are interacting. For example, the client cannot tell whether it is connected directly to the end server or two an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.

Logical design of IoT- IoT communication APIs

- **Uniform interface** – uniform interface constraints requires that the method of communication between client and server must be uniform. Resources are identified in the requests (by URIs in web based systems) and are themselves is separate from the representations of the resources data returned to the client. When a client holds a representation of resources it has all the information required to update or delete the resource you (provided the client has required permissions). Each message includes enough information to describe how to process the message.
- **Code on demand** – Servers can provide executable code or scripts for clients to execute in their context. this constraint is the only one that is optional.

Logical design of IoT- IoT communication APIs

A RESTful web service is a " Web API " implemented using HTTP and REST principles.
REST is most popular IoT Communication APIs.

The Following table represents the HTTP Methods used in REST

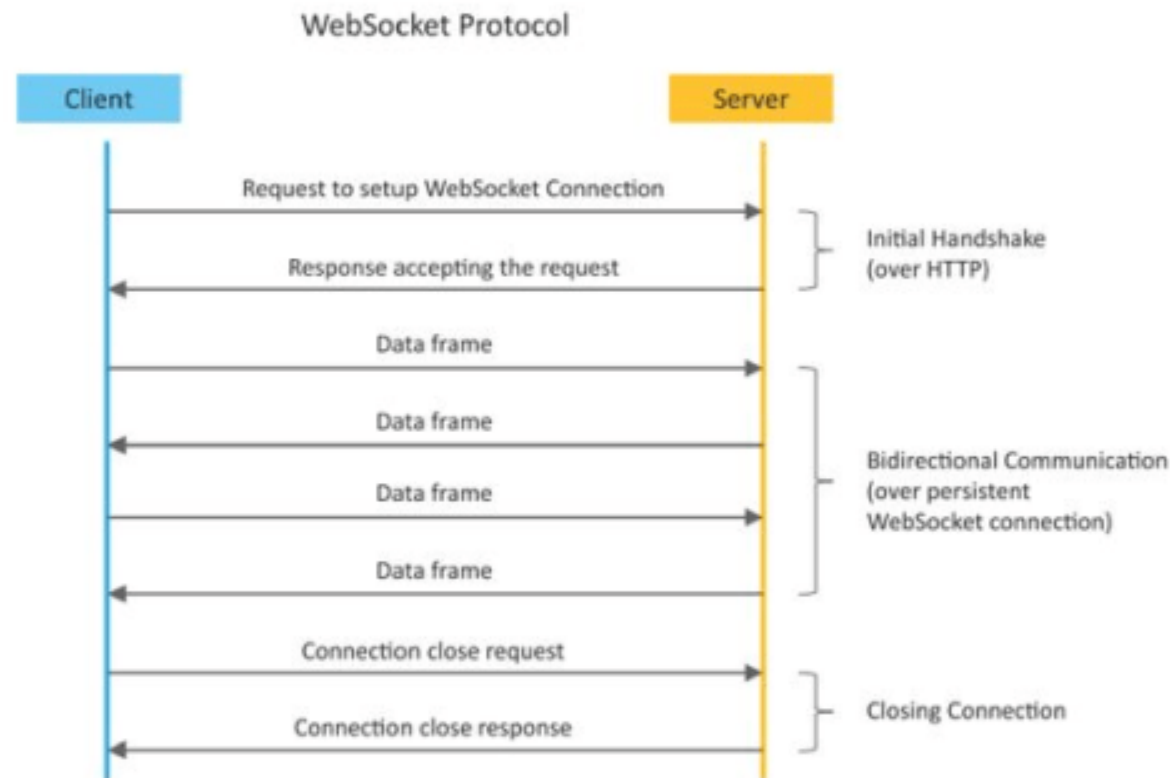
Uniform Resource Identifier (URI)	GET	PUT	PATCH	POST	DELETE
Collection, such as https://api.example.com/resources/	<i>List</i> the URIs and perhaps other details of the collection's members.	<i>Replace</i> the entire collection with another collection.	Not generally used	<i>Create</i> a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	<i>Delete</i> the entire collection.
Element, such as https://api.example.com/resources/item5	<i>Retrieve</i> a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	<i>Replace</i> the addressed member of the collection, or if it does not exist, create it.	<i>Update</i> the addressed member of the collection.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.	<i>Delete</i> the addressed member of the collection.

Logical design of IoT- IoT communication APIs

- **WebSocket based communication API**
- WebSocket APIs allow bi-directional, full duplex communication between clients and servers.
- WebSocket APIs follow the exclusive pair communication model. Unlike request-response model such as REST, the WebSocket APIs allow full duplex communication and do not require new connection to be setup for each message to be sent. WebSocket communication begins with a connection setup request sent by the client to the server.
- The request (called websocket handshake) is sent over HTTP and the server interprets it as an upgrade request. If the server supports websocket protocol, the server responds to the websocket handshake response. After the connection setup client and server can send data/messages to each other in full duplex mode.
- WebSocket API reduce the network traffic and latency as there is no overhead for connection setup and termination requests for each message.
- WebSocket suitable for IoT applications that have low latency or high throughput requirements. So Web socket is most suitable IoT Communication APIs for IoT System.

Logical design of IoT- IoT communication APIs

- The below given figure shows the WebSocket based communication API

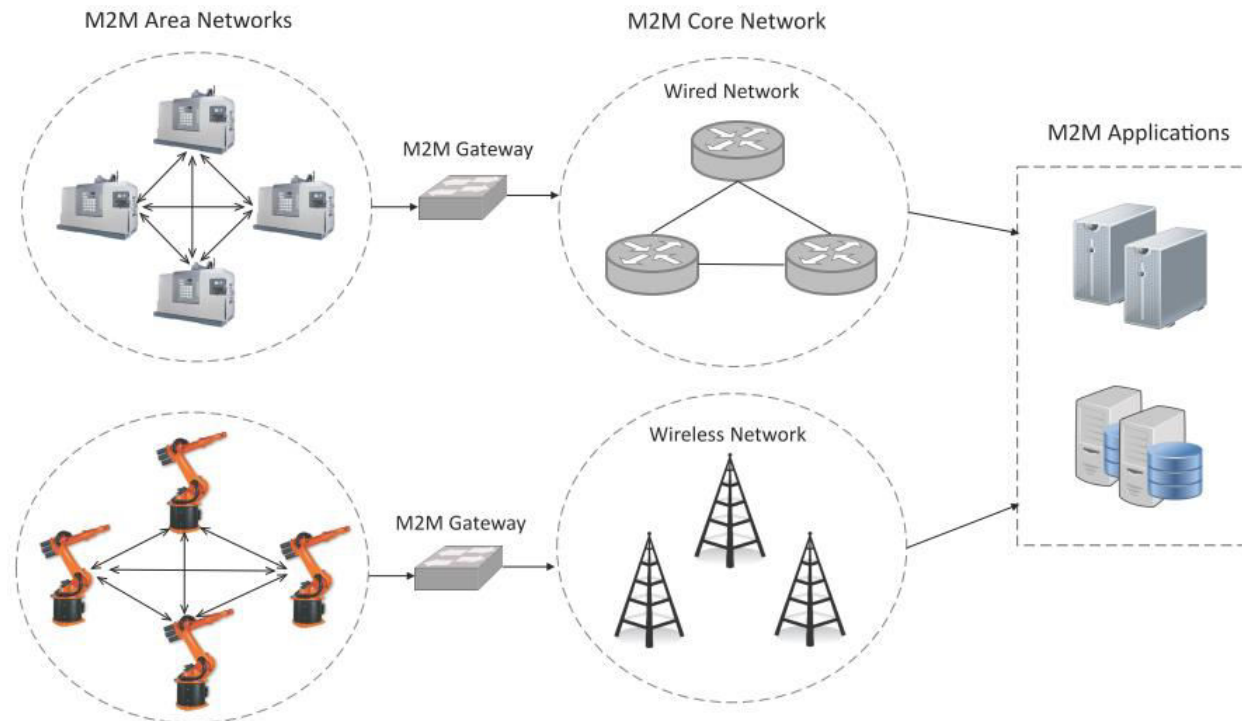


Machine to machine

- **What is the M2M Communication?**
- This is the M2M or Machine to Machine communication.
- The ***machine to machine*** concept represents any technology that allows two devices to exchange information with each other, for example, communicate and send data. The communication that occurs between the machines or devices is autonomous, there is no need for human intervention for this data exchange to take place.
- M2M connectivity is related to the ***Internet of Things (IoT)***. Both are part of the same concept and complement each other. Thanks to IoT, a system of machines or interrelated devices can be connected wirelessly, and exchange and analyze data automatically in the cloud. In short, IoT is enabled by integrating many M2M devices and using cloud web platforms to process all that data.

Machine-to-Machine (M2M)

- Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange.

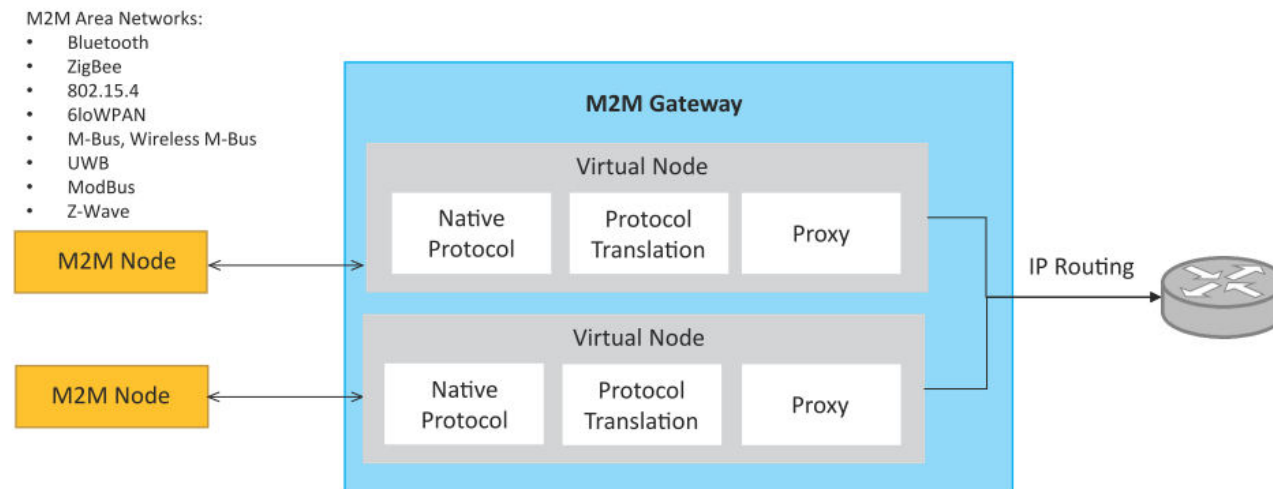


Machine-to-Machine (M2M)

- An M2M area network comprises machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.
- Various communication protocols can be used for M2M local area networks, such as ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.
- The communication network provides connectivity to remote M2M area networks.
- The communication network can use either wired or wireless networks (IP-based).
- While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks.

M2M Gateway

- Since non-IP-based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network.
- To enable communication between remote M2M area networks, M2M gateways are used.



Machine to machine

- [M2M, or machine-to-machine](#), is the foundation of the sophisticated device connectivity that we enjoy today. An M2M connection is a point-to-point connection between two network devices that allows them to transmit information via public networking technologies such as Ethernet and cellular networks.
- **Example:** Sensor telemetry is one of the original uses of M2M communication. For decades, businesses have used M2M to remotely monitor factors like temperature, energy consumption, moisture, pressure and more through sensors.
- **Example:** ATMs offer another great example of M2M technology. The ATM's internal computer is constantly communicating with a host processor that routes transactions to the appropriate banks and accounts. The banks then send back approval codes through the host processor, allowing transactions to be completed.
- What makes this an example of M2M technology is that the entire transaction happens remotely and without any need for a human operator on the bank's side. Machines communicate smoothly, efficiently and automatically, allowing transactions to be authorized in seconds. M2M technology has a decades-long track record of improving the world's ability to communicate and execute transactions effectively across long distances and in real time.

Machine to machine

- **What types of connectivity are used in M2M?**
- There are different types of connectivity between machines, and it is possible that you already know almost everyone. But, what are the most used? First, we have the **RFID**, or radiofrequency identification.
- The limitation of this type of connectivity is that it has a maximum range of 10 meters. On the other hand, there are **Bluetooth** and WiFi that also have a limited range, from 10 to 20 meters in the case of Bluetooth and 50 meters in the case of WiFi.
- These types of connectivity are short range if we compare them with the following. Connectivity using **low frequency** has a range of up to 1,000 km and the **GSM** network (using SIM cards) or the satellite is worldwide. As you can see, there are many different options that allow us adapting to each problem.

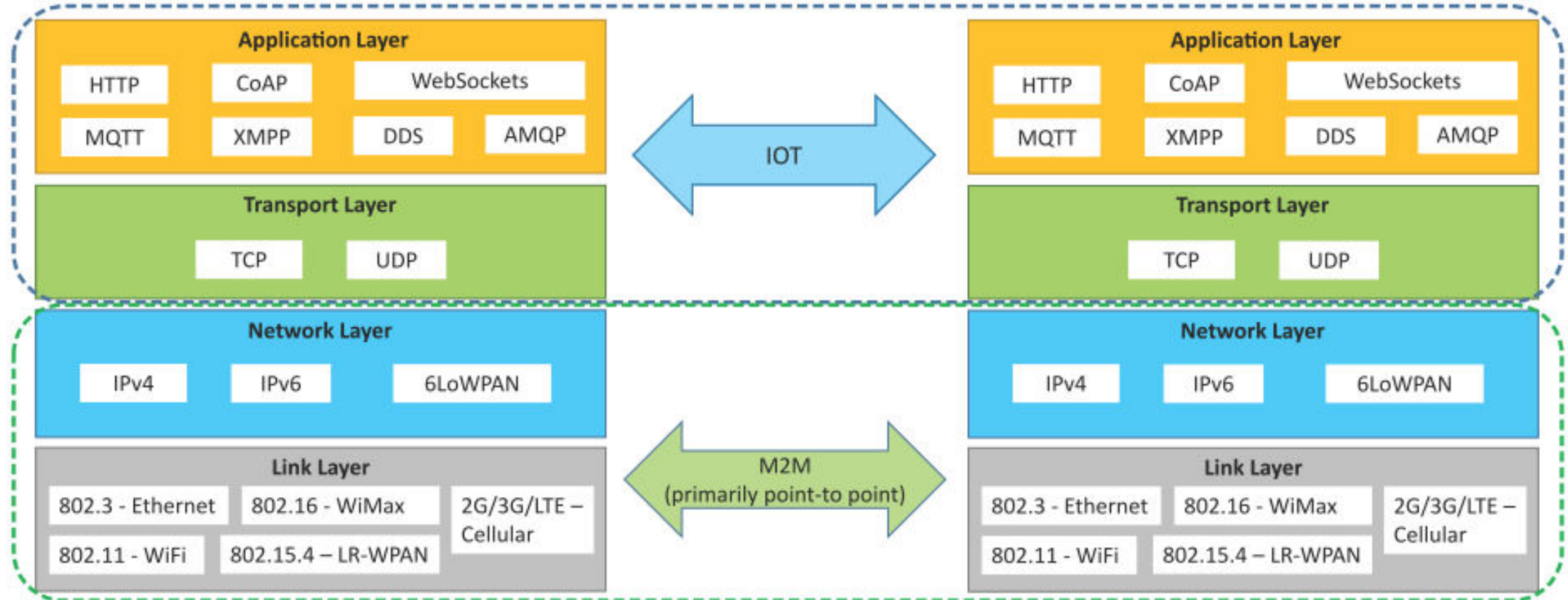
Machine to machine

- **Some interesting applications of M2M**
- The applications and areas in which connectivity between machines can be applied and used are very wide.
- For example, the connected *vending* machines allow the distributor to know their replacement status and to notify in cases which some product runs out.
- It is also very useful in the health area. Telemedicine is a concept already implemented in some places and has meant great improvements in this area. In hospitals, processes are automated to improve efficiency and safety, for example, using devices capable of reacting faster than humans. If a patient has a drop in vital signs and is connected to an M2M device, the machine can automatically administer extra oxygen before the hospital staff reaches it.

Machine to machine

- Likewise, it is also **used in the industry** , allowing machines to be connected to each other and sending data each other. With this data, they can optimize processes automatically, notify when a machine has a breakdown or even self-repair.
- In general we can establish the following industrial applications:
 - Automated maintenance
 - Procedure for requesting spare parts
 - End of process notice
 - Data collection for processing by other equipment
 - Intelligent stock control
 - Implementation of just-in-time systems
- More and more companies are starting to use this type of connectivity that improves the efficiency level and allows us to address great production challenges.

Communication in IoT vs M2M



Difference between IoT and M2M

- Communication Protocols
 - M2M and IoT can differ in how the communication between the machines or devices happens.
 - M2M uses either proprietary or non-IP-based communication protocols for communication within the M2M area networks.
- Machines in M2M vs Things in IoT
 - The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states.
 - M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.

Difference between IoT and M2M

- **Internet of Things :**
IOT is known as the Internet of Things where things are said to be the communicating devices that can interact with each other using a communication media. Usually every day some new devices are being integrated which uses IoT devices for its function.
- These devices use various sensors and actuators for sending and receiving data over the internet. It is an ecosystem where the devices share data through a communication media known as the internet.
- For example, an air conditioner's sensor may collect the data on outside temperatures and change its temperature to increase or decrease according to the outside environment's temperature. Likewise, the refrigerators may change their temperature accordingly, too.

Difference between IoT and M2M

- **Machine to Machine :**
This is commonly known as Machine to machine communication. It is a concept where two or more than two machines communicate with each other without human interaction using a wired or wireless mechanism.
- M2M is a technology that helps the devices to connect between devices without using internet.
- M2M communications offer several applications such as security, tracking and tracing, manufacturing and facility management.
- M2M technology may be present in offices, shopping malls, houses, and many other places. A common example of a machine to machine is controlling electrical devices like fans and bulbs using Bluetooth from the smartphone. Here, the smartphone and electrical devices are the two interacting devices with each other.

Main Differences between the IoT and M2M

1. IoT is a subset of M2M technology. In IoT, the communication between two machines without human instruction, making it a part of the M2M communication system.
2. The point-to-point communication of M2M is the main difference between M2M and IoT technology. Meanwhile, an IoT system usually locates its devices within a global cloud network that facilitates larger-scale automation and more advanced applications.
3. Another key difference between IoT and M2M is scalability. IoT is designed to be highly scalable because devices may also be included in the network and integrated into existing networks with minimal issues. In contrast, maintaining and setting up M2M networks could also be more labor-intensive, as new point-to-point connections must be built for each system.

Difference between IoT and M2M

- Hardware vs Software Emphasis
 - While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
- Data Collection & Analysis
 - M2M data is collected in point solutions and often in on-premises storage infrastructure.
 - In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud).
- Applications
 - M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications and on-premises enterprise applications.
 - IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

Difference between IoT and M2M

Sl.No	Basis of	IoT	M2M
1	Abbreviation	Internet of Things	Machine to Machine
2	Intelligence	Devices have objects that are responsible for decision making	Some degree of intelligence is observed in this
3	Connection type used	The connection is via Network and using various communication types.	The connection is a point to point
4	Communication protocol used	Internet protocols are used such as HTTP , FTP , and Telnet .	Traditional protocols and communication technology techniques are used
5	Data Sharing	Data is shared between other applications that are used to improve the end-user experience.	Data is shared with only

Difference between IoT and M2M

Sl.No	Basis of	IoT	M2M
6	Internet	Internet connection is required for communication	Devices are not dependent on the Internet.
7	Scope	A large number of devices yet scope is large.	Limited Scope for devices.
8	Business Type used	Business 2 Business(B2B) and Business 2 Consumer(B2C)	Business 2 Business (B2B)
9	Open API support	Supports Open API integrations.	There is no support for Open Api's
10	Examples	Smart wearables, Big Data and Cloud, etc.	Sensors, Data and Information, etc.

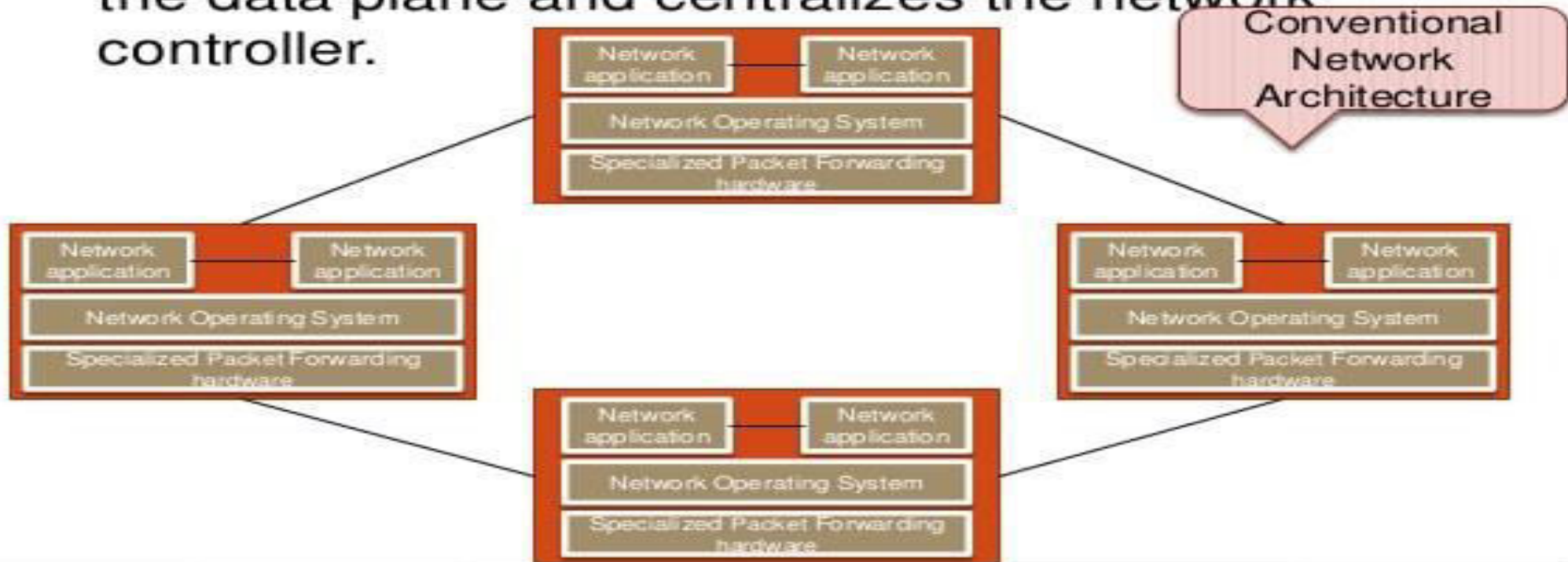
Software Defined Network (SDN)

- **What is software-defined networking (SDN)?**
- **Software-defined networking (SDN)** is an approach to networking that uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network.
- This model differs from that of traditional networks, which use dedicated hardware devices (i.e., routers and switches) to control network traffic. SDN can create and control a [virtual network](#) – or control a traditional hardware – via software.
- While [network virtualization](#) allows organizations to segment different virtual networks within a single physical network, or to connect devices on different physical networks to create a single virtual network, software-defined networking enables a new way of controlling the routing of data packets through a centralized server.

Software Defined Network (SDN)

What is Software Defined Networking?

- Software Defined Networking is a networking architecture that separates the control plane from the data plane and centralizes the network controller.



Key element of the SDN

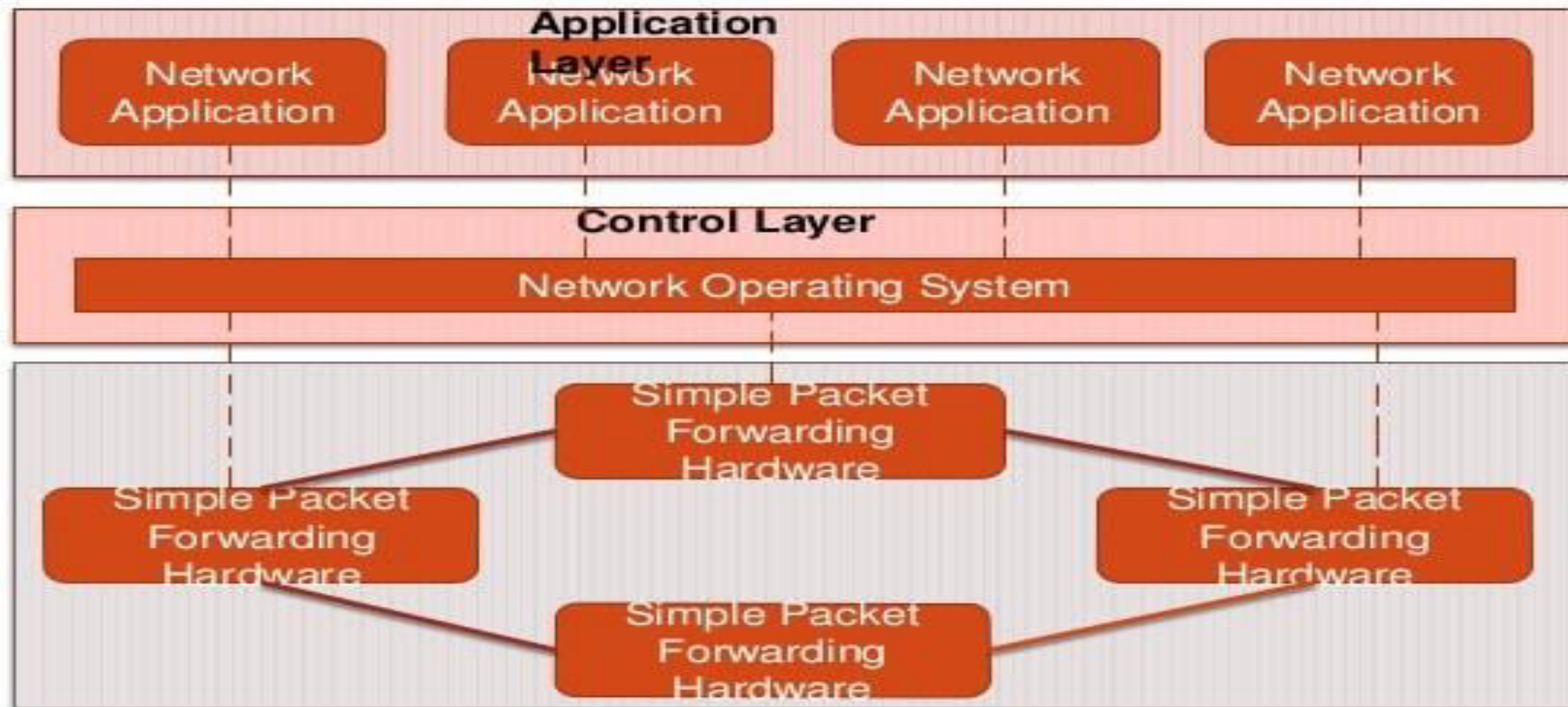
- Centralized Network Controller: With decoupled control and dataplanes and centralized network controller, the network administrators can rapidly configure the network.
- Programmable Open APIs: SDN architecture supports programmable open API for interface between the SDN application and control layers(Northbound interface).
- Standard Communication Interface(OpenFlow): SDN architecture uses a standard communication interface between the control and infrastructure layers(Southbound interface).

Software Defined Network (SDN)

- Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Software-based SDN controllers maintain a unified view of the network and make configuration, management and provisioning simpler.
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

Software Defined Network (SDN)

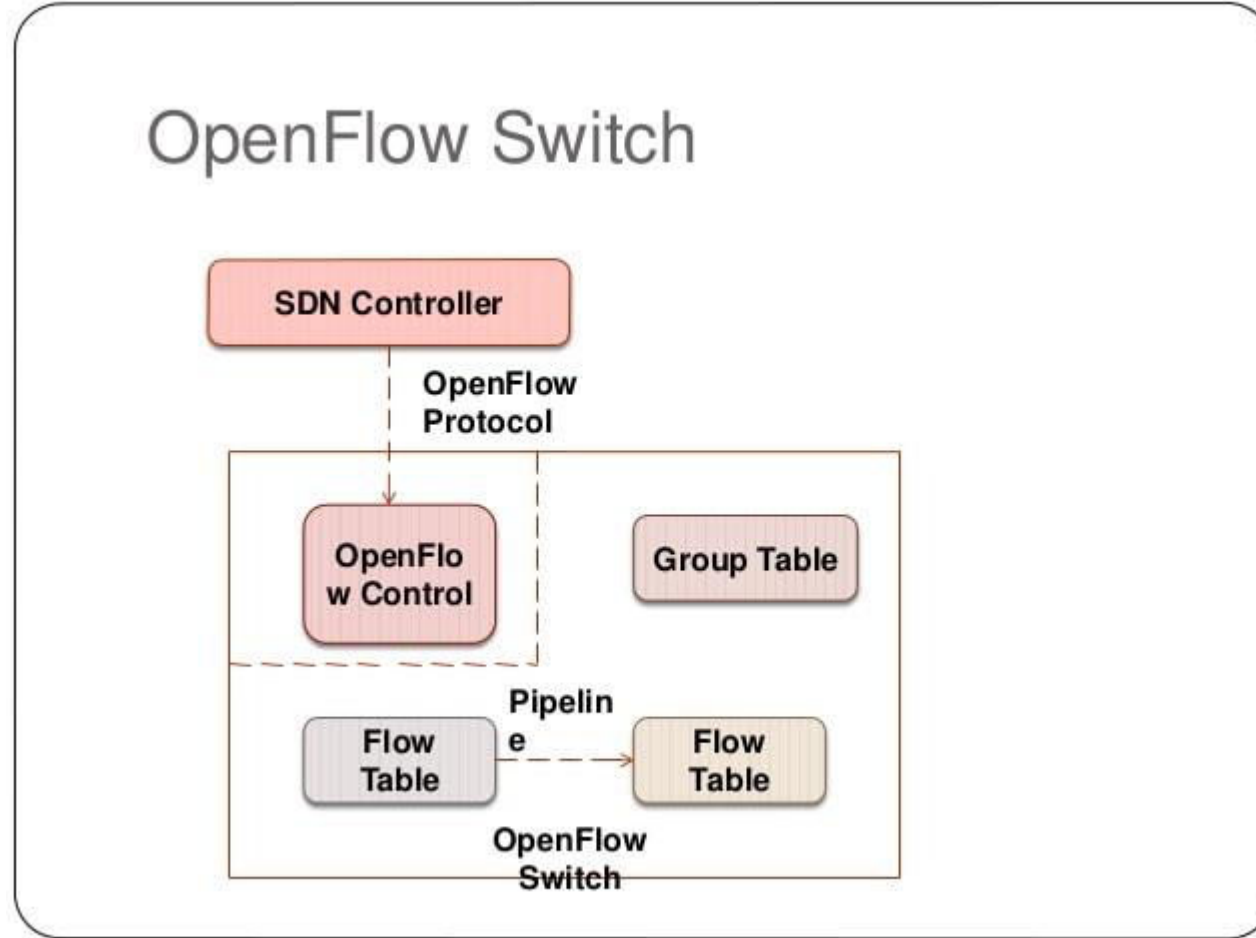
Software Defined Networking(SDN)



Software Defined Network (SDN)

- **Open flow**
- Which is defined by Open Networking Foundation(ONF) is broadly accepted by SDN protocol for the southbound interface.
- With Open Flow, the forwarding plane of the network devices can be directly accessed and manipulated.
- Open flow uses the concept of flows to identify network traffic based on predefined match rules.
- Flows can be programmed statically or dynamically by SDN control software.
- Openflow protocol is implemented on both sides of the interface between the controller and the network devices.
- The controller manages the switch via the openflow switch protocol.
- The controller can add, update and delete flow entries in the flow table.
- Each flow entries contains the match fields, counters and set of instructions to apply matching packets.

Software Defined Network (SDN)



Software Defined Network (SDN)

- **Why software-defined networking is important?**
- SDN represents a substantial step forward from traditional networking, in that it enables the following:
- **Increased control with greater speed and flexibility:** Instead of manually programming multiple vendor-specific hardware devices, developers can control the flow of traffic over a network simply by programming an open standard software-based controller. Networking administrators also have more flexibility in choosing networking equipment, since they can choose a single protocol to communicate with any number of hardware devices through a central controller.
- **Customizable network infrastructure:** With a software-defined network, administrators can configure network services and allocate virtual resources to change the network infrastructure in real time through one centralized location. This allows network administrators to optimize the flow of data through the network and prioritize applications that require more availability.

Software Defined Network (SDN)

- **Robust security:** A software-defined network delivers visibility into the entire network, providing a more holistic view of security threats. With the proliferation of smart devices that connect to the internet, SDN offers clear advantages over traditional networking. Operators can create separate zones for devices that require different levels of security, or immediately quarantine compromised devices so that they cannot infect the rest of the network.
- The key difference between SDN and traditional networking is infrastructure: SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking. It allows administrators to control the network, change configuration settings, provision resources, and increase network capacity — all from a centralized user interface, without the need for more hardware.
- There are also security differences between SDN and traditional networking. Thanks to greater visibility and the ability to define secure pathways, SDN offers better security in many ways. However, because software-defined networks use a centralized controller, securing the controller is crucial to maintaining a secure network.

Software Defined Network (SDN)

- **How does software-defined networking (SDN) work?**
- Here are the SDN basics: In SDN (like anything virtualized), the software is decoupled from the hardware. SDN moves the control plane that determines where to send traffic to software, and leaves the data plane that actually forwards the traffic in the hardware. This allows network administrators who use software-defined networking to program and control the entire network via a single pane of glass instead of on a device by device basis.
- There are three parts to a typical SDN architecture, which may be located in different physical locations:
 - **Applications**, which communicate resource requests or information about the network as a whole
 - **Controllers**, which use the information from applications to decide how to route a data packet
 - **Networking devices**, which receive information from the controller about where to move the data

Software Defined Network (SDN)

- Physical or [virtual networking](#) devices actually move the data through the network. In some cases, virtual switches, which may be embedded in either the software or the hardware, take over the responsibilities of physical switches and consolidate their functions into a single, intelligent switch. The switch checks the integrity of both the data packets and their virtual machine destinations and moves the packets along.

Software Defined Network (SDN)

- **Benefits of software-defined networking (SDN)**
- Many of today's services and applications, especially when they involve the cloud, could not function without SDN. SDN allows data to move easily between distributed locations, which is critical for cloud applications.
- Additionally, SDN supports moving workloads around a network quickly. For instance, dividing a virtual network into sections, using a technique called network functions virtualization (NFV), allows telecommunications providers to move customer services to less expensive servers or even to the customer's own servers. Service providers can use a virtual network infrastructure to shift workloads from private to public cloud infrastructures as necessary, and to make new customer services available instantly. SDN also makes it easier for any network to flex and scale as network administrators add or remove virtual machines, whether those machines are on-premises or in the cloud.
- Finally, because of the speed and flexibility offered by SDN, it is able to support emerging trends and technologies such as edge computing and the [Internet of Things](#), which require transferring data quickly and easily between remote sites.

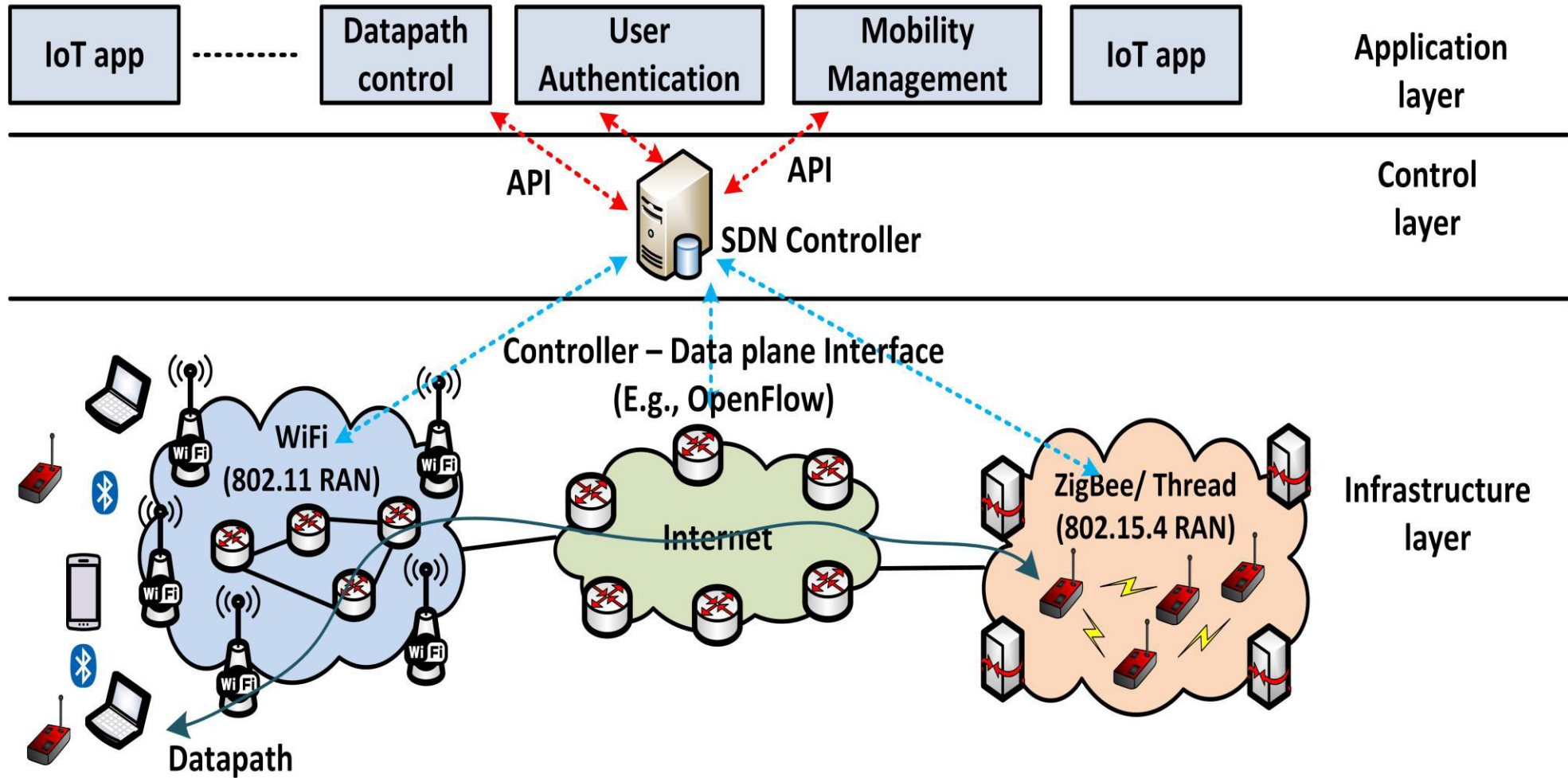
Software Defined Network (SDN)

- **How is SDN different from traditional networking?**
- The key difference between SDN and traditional networking is infrastructure: SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking. It allows administrators to control the network, change configuration settings, provision resources, and increase network capacity—all from a centralized user interface, without adding more hardware.
- There are also security differences between SDN and traditional networking. Thanks to greater visibility and the ability to define secure pathways, SDN offers better security in many ways. However, because software-defined networks use a centralized controller, securing the controller is crucial to maintaining a secure network, and this single point of failure represents a potential vulnerability of SDN.

Software Defined Network (SDN)

- **What are the different models of SDN?**
- While the premise of centralized software controlling the flow of data in switches and routers applies to all software-defined networking, there are different models of SDN.
- **Open SDN:** Network administrators use a protocol like OpenFlow to control the behavior of virtual and physical switches at the data plane level.
- **SDN by APIs:** Instead of using an open protocol, application programming interfaces control how data moves through the network on each device.
- **SDN Overlay Model:** Another type of software-defined networking runs a virtual network on top of an existing hardware infrastructure, creating dynamic tunnels to different on-premise and remote data centers. The virtual network allocates bandwidth over a variety of channels and assigns devices to each channel, leaving the physical network untouched.
- **Hybrid SDN:** This model combines software-defined networking with traditional networking protocols in one environment to support different functions on a network. Standard networking protocols continue to direct some traffic, while SDN takes on responsibility for other traffic, allowing network administrators to introduce SDN in stages to a legacy environment.

Software Defined Networking for IoT



Software Defined Networking for IoT

- **Internet of things (IoT)** poses challenges that are different from traditional Internet in different aspects — heterogeneous communication technologies, application-specific QoS requirements, massive influx of data, and unpredictable network conditions. On the other hand, software-defined networking (SDN) is a promising approach to control the network in a unified manner using rule-based management. The abstractions provided by SDN enable holistic control of the network using high-level policies, without being concerned about low-level configuration issues. Hence, it is advantageous to address the heterogeneity and application-specific requirements of IoT.
- We study the application and impact of softwarization on IoT networks from different perspectives: **access networks, edge networks, and wide area networks**. We also develop and analyze models to characterize the performance of softwarized networks.

Advantages and Disadvantages of SDN

- **Advantages of SDN:**

- Network is programmable hence can easily be modified via the controller rather than individual switches.
- Switch hardware becomes cheaper since each switch only needs a data plane.
- Hardware is abstracted, hence applications can be written on top of controller independent of switch vendor.
- Provides better security since the controller can monitor traffic and deploy security policies. For example, if the controller detects suspicious activity in network traffic, it can reroute or drop the packets.

- **Disadvantages of SDN:**

The central dependency of the network means single point of failure, i.e. if the controller gets corrupted, the entire network will be affected.

Features of software-defined networking

- There are [4 unique, defining features](#) of software-defined networking:
- **Agile.** As business and application needs change, administrators can adjust network configuration without
- **Centrally Managed.** SDN consolidates network intelligence, which provides a holistic view of the network configuration and activity
- **Programmable.** The ability to directly program network features and configure network resources quickly and easily through automated SDN services.
- **Open Connectivity.** SDN is based on and implemented via open standards. As a result, SDN streamlines network design and provides consistent networking in a vendor -neutral architecture.

Difference between SDN and Traditional Network

S.No.	SDN	TRADITIONAL NETWORK
01.	Software Defined Network is virtual networking approach.	Traditional network is the old conventional networking approach.
02.	Software Defined Network is centralized control.	Traditional Network is distributed control.
03.	This network is programmable.	This network is non programmable.
04.	Software Defined Network is open interface.	Traditional network is closed interface.
05.	In Software Defined Network data plane and control plane are decoupled by software.	In traditional network data plane and control plane are mounted on same plane
06.	It supports automatic configuration so it takes less time.	It supports static/manual configuration so it takes more time.
07.	It can prioritize and block specific network packets.	It leads all packets in the same way no prioritization support.

Difference between SDN and Traditional Network

S.No.	SDN	TRADITIONAL NETWORK
08.	It is easy to program as per need.	It is difficult to program again and to replace existing program as per use.
09.	Cost of Software Defined Network is low.	Cost of Traditional Network is high.
10.	Structural complexity is low in Software Defined Network.	Structural complexity is high in Traditional Network.
11.	Extensibility is high in Software Defined Network.	Extensibility is low in Traditional Network.
12.	In SDN it is easy to troubleshooting and reporting as it is centralized controlled.	In Traditional network it is difficult to troubleshoot and report as it is distributed controlled.
13.	Its maintenance cost is lower than traditional network.	Traditional network maintenance cost is higher than SDN.

References

- <https://www.edureka.co/blog/iot-tutorial/>
- <https://www.guru99.com/iot-tutorial.html>
- <https://www.javatpoint.com/iot-internet-of-things>
- <https://www.programmingoneonone.com/2021/04/physical-design-of-iot.html>
- <https://www.geeksforgeeks.org/difference-between-iot-and-m2m/>
- <https://www.javatpoint.com/iot-vs-m2m>
- [https://www.tutorialspoint.com/internet of things/internet of things common uses.htm](https://www.tutorialspoint.com/internet_of_things/internet_of_things_common_uses.htm)
- <https://iotbyhvm.ooo/physical-and-logical-design-of-iot/>
- <https://www.atriainnovation.com/en/m2m-communication-what-is-it/>
- <https://www.javatpoint.com/iot-architecture-models>
- <https://medium.datadriveninvestor.com/4-stages-of-iot-architecture-explained-in-simple-words-b2ea8b4f777f>
- <https://www.vmware.com/topics/glossary/content/software-defined-networking>
- https://cse.iitkgp.ac.in/~smisra/theme_pages/sdn/index.html#
- <https://www.slideshare.net/SagarRai14/sdn-software-defined-network-and-nfvnetwork-function-virtualization-for-internet-of-things>
- https://cse.iitkgp.ac.in/~smisra/theme_pages/sdn/index.html#
- <https://www.geeksforgeeks.org/difference-between-software-defined-network-and-traditional-network/>

CSPC702-EMBEDDED SYSTEMS AND INTERNET OF THINGS (IOT)

UNIT - IV Network and communication aspects

Syllabus

- **UNIT - IV Network and communication aspects**
- Wireless medium access issues, MAC protocol survey, Survey routing protocols, Sensor deployment & Node discovery, Data aggregation & dissemination- Applications of IoT: Home automation, Industry applications, Surveillance applications, Other IoT applications.

Wireless medium access issues

Introduction

- ☐ The medium access sublayer is the bottom part of data link layer. The medium access sublayer is also known as MAC (Medium Access Control) sublayer.
- ☐ When a common medium is shared by many stations, MAC layer plays a very important role. Without this control several stations transmits simultaneously could produce garbled messages.
- ☐ The media access control mechanism standardized by IEEE are implemented in the MAC sublayer of the Data link layer. It provides service to the LLC (Logical Link Control) sublayer and receives service from the physical layer below it.
- ☐ The basic functions of MAC sublayer is the media access control, error detection and station addressing. The physical transmission medium of a LAN is shared by the stations connected on the LAN.
- ☐ Media access control procedures are implemented to ensure that every station gets a fair chance to transmit and collision do not take place.
- ☐ There are several methods of media access control in LANs. Each method is applicable to specific LAN topology. In addition to the basic access procedures, the MAC layer also handles the frame delimiting, address recognition and error checking functions
- ☐ When number of user stations share a single transmission medium. This is called as multiple access communication. The transmission medium is broadcast in nature, so all other attached stations to the medium can receive the transmission from any given station.

Wireless medium access issues

- When it comes to communication using a wireless medium there is always a concern about the interference due to other present wireless communication technologies. Wireless means communication and message transfer without the use of physical medium i.e., wires.
- Let us understand how communication is done between them. Different Mobile stations(MS) are attached to a transmitter/receiver which communicates via a shared channel by other nodes. In this type of communication, it makes it difficult for the MAC design rather than the wireline networks.

Wireless medium access issues

- The very important issues which are observed are:

- Half Duplex operation,
- Time-varying channel, and
- Burst channel errors.

These are explained as following below.

- **Half Duplex operation:**

- Half-duplex transmission means when the sender and receiver both are capable of sharing data but one at a time.
- In wireless transmission, it is difficult to receive data when the transmitter is sending the data because during transmission a large amount or a large fraction of signal energy is leaked while broadcasting.
- The magnitude of the transferred signal and received signal differs a lot. Due to which collision detection is even not possible by the sender as the intensity of the transferred signal is large than the received one. Hence this causes the problem of collision and the prime focus should be to minimize the collision.

Wireless medium access issues

- **Time-varying channel :**

Time-varying channels include the three mechanisms for radio signal propagations they are Reflection, Diffraction, and Scattering.

- **Reflection**

This occurs when a propagating wave carrying information intrudes on an object that has very large dimensions than the wavelength of the wave.

- **Diffraction**

This occurs when the radio path between the transmitter and the receiver is collided by the surface with sharp edges. This is a phenomenon which causes the diffraction of the wave from the targeted position.

- **Scattering**

This occurs when the medium through from the wave is traveling consists of some objects which have dimensions smaller than the wavelength of the wave.

- While transmitting the signal by the node these are time shifted and this is called multipath propagation. While when this node signals intensity is dropped below a threshold value, then this is termed as fade. As a result Handshaking strategy is widely used so as a healthy communication can be set up.

MAC protocol survey

Introduction

- A typical node in the WSN consists of a sensor, embedded processor, moderate amount of memory and transmitter/receiver circuitry. The sensor node's radio in the WSNs consumes a significant amount of energy. Because of hardware limitations further energy efficiency can be achieved through the design of energy efficient communication protocols.
- Medium access control (MAC) is an important technique. One of the main functions of the MAC protocol is to avoid collisions from interfering nodes. The classical IEEE MAC protocol for wireless local area network wastes a lot of energy because of idle listening. Designing power efficient MAC protocol is one of the ways to prolong the life time of the network.

MAC protocol survey

MAC Protocol Design Challenges

- The medium access control protocols for the wireless sensor network have to achieve two objectives.
 - The first objective is the creation of the sensor network infrastructure.
 - The second objective is to share the communication medium fairly and efficient.
- **Attributes of a Good MAC Protocol**
 - Energy Efficiency
 - Latency
 - Throughput
 - Fairnessy.

MAC protocol survey

- **Major Sources of Energy Wastes**

- Collision
- Overhearing
- Packet Overhead
- Idle listening

- **MAC Performance Matrices**

- Energy Consumption per bit
- Average Delivery Ratio
- Average Packet Latency
- Network Throughput

MAC protocol survey

- **Responsibilities and Design issues of MAC Protocol**
- Ad hoc wireless networks are included portable nodes that trade packets by sharing a typical communicate radio channel. Because of the constraints of this channel, the data transmission to be shared among the nodes is constrained.
- In this manner, the point in these networks is to have the option to use the transmission capacity effectively, and ensure decency to all nodes. As we probably are aware, wireless networks contrast gigantically from wired networks moreover, ad hoc wireless networks have significantly progressively explicit attributes, for example, node versatility, power requirements.
- Thus, new protocols are required for controlling access to the physical medium. The special properties of the ad hoc network make the structure of a medium access control (MAC) protocol all the more testing.

MAC protocol survey

- **Responsibilities of MAC Protocol :**
 - Network overhead should be low.
 - Efficiently allocate the bandwidth.
 - Distributed MAC operation.
 - Power control mechanism should be present.
 - Maximum utilization of channel.
 - Hidden and Exposed problem should be removed.
 - Nodes should be sync with time.

MAC protocol survey

- **Design issues of MAC Protocol :**
- **Bandwidth Efficiency**
The shortage of data transfer capacity assets in these networks requires its proficient use. To evaluate this, we could state that bandwidth capacity is the proportion of the bandwidth used for data transmission to the complete accessible bandwidth capacity.
- **Quality of Service Support**
Quality of service support is difficult due to the mobility of the nodes. Once a node moves out of reach, the reservation in it is lost. In these networks, QoS is extremely important because if it is being used in military environments, the service support needed time to time.
- **Synchronization**
Some instruments must be found so as to give synchronization among the nodes. Synchronization is significant for directing the bandwidth reservation.
- **Hidden Terminal Problem**
When there are two nodes, both are outside of each other's range and try to communicate with same node within their range at the same time, then there must be packet collision.
- **Exposed Terminal Problem**
Uncovered nodes might be denied channel access pointlessly, which implies under usage of the bandwidth resources.

Survey routing protocols

- **Definition of Routing Protocol**
- Routing bears significant importance since nodes in a IoT network act as hosts and routers delivering **data to the gateways**. Many routing protocols have been proposed for sensor networks and are applicable within the IoTs. The routing of data from source to destination impacts the power consumption of forwarding nodes.

Survey routing protocols

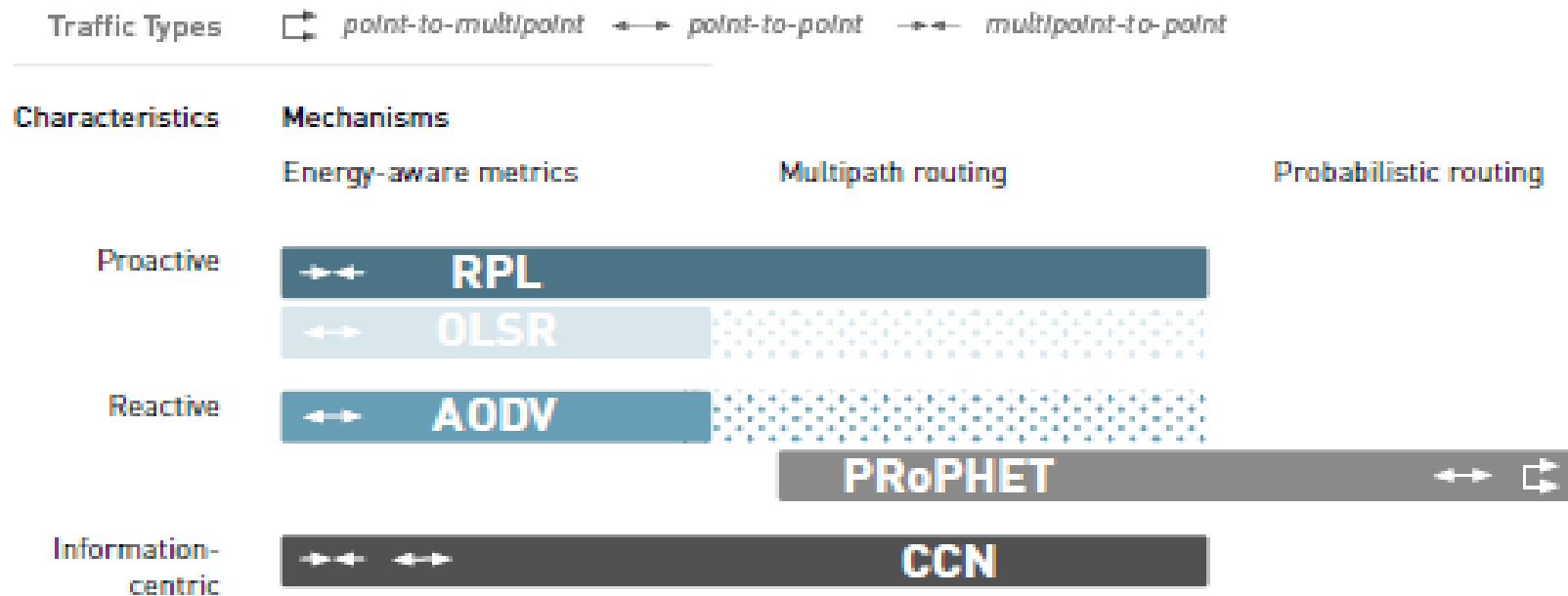


Figure 3.1: Overview over possible routing protocols for the IoT.

Survey routing protocols-RPL Protocol

- RPL is a distance-vector and a source routing protocol that is designed to operate on top of several link layer mechanisms including IEEE 802.15.4 PHY and MAC layers.
- These link layers could be constrained, potentially lossy, or typically utilized in conjunction with highly constrained host or router devices, such as but not limited to, low power wireless or PLC (Power Line Communication) technologies.
- RPL mainly targets collection-based networks, where nodes periodically send measurements to a collection point.
- A key feature of RPL is that it represents a specific routing solution for low power and lossy networks.
- The protocol was designed to be highly adaptive to network conditions and to provide alternate routes, whenever default routes are inaccessible.
- RPL provides a mechanism to disseminate information over the dynamically formed network topology. This mechanism uses Trickle to optimize the dissemination of control messages

Survey routing protocols- LOADng

- Lightweight on-demand ad hoc distance-vector routing protocol-next generation LOADng :
- The Lightweight on-demand ad hoc distance-vector routing protocol-next generation or LOADng is a lightweight variation of AODV for LLNs.
- It is designed based on the idea that LLNs are idle most of the time. Hence instead of adopting a proactive approach would generate unnecessary overhead, LOADng follows a reactive approach in which routes are established towards destinations only when there is some data to send.
- LOADng is a reactive routing protocol, and found suitable for a more general traffic pattern. It does not have any node that performs special functions like the root and is hence not subjected to the subsequent problems that arise due to such a consideration.
- Also, due to its compressed and flexible data format, there is no possibility of fragmentation. It does not impose any strict source routing rules, hence it can accommodate applications which require a fixed MTU.
- However, LOADng might have a higher delay in the route discovery phase and might have higher control traffic overhead if the traffic flows are predominantly P2P.

Survey routing protocols-CTP

- **Collection Tree Protocol (CTP) :**
- CTP is a distance vector routing algorithm that was developed as a solution to routing in WSNs.
- It stands as a predecessor to RPL and was considered the de-facto routing standard for Tiny OS.
- It builds a tree-based topology with the root at the sink of the network, CTP uses adaptive beaconing mechanism to broadcast routing control messages.
- Moreover, CTP relied on a specific link-layer technology for topology formation, CTP was earlier known for its efficient energy consumption and high Packet Reception Ratio (PRR).

Survey routing protocols

- **CORPL Routing Protocol:** CORPL will retain the Directed Acyclic Graph (DAG) based approach of RPL and at the same time introduce novel modifications to allow its application in Cognitive Radio environments.
- CORPL uses an opportunistic forwarding approach that consists of two key steps:
 - selection of a forwarder set i.e., each node in the network selects multiple next hop neighbors, and a coordination scheme to ensure that only the best receiver of each packet forwards it (unique forwarder selection).
- In CORPL, each node maintains a forwarder set such that the forwarding node (next hop) is opportunistically selected. The DAG construction process in CORPL follows a similar procedure as in RPL.
- After detecting a vacant channel, the gateway node transmits a Destination Information Object (DIO) message. is constructed in such a way that the forwarding nodes are within the transmission range of each other. During the DIO transmission, each node also reports some additional information using the Option field of the DIO message.
- Each node updates the neighborhood information through the DIO message transmission. Based upon the neighborhood information, each node dynamically prioritizes its neighbors in order to construct the forwarder list.

Survey routing protocols

- **CARP Routing Protocol:**

- Channel-aware Routing Protocol (CARP) is a multi-hop delivery of data to the sink for WSN. CARP obviates the drawbacks such as link quality is explicitly taken into account for selecting the next-hop node on a route to the sink.
- CARP quickly varying conditions of the underwater channel, the fact that two nodes can exchange short control packets correctly, may not be sufficient to guarantee that longer data packets are also going to be safely delivered”.
- Generally, CARP is a location is constructed in such a way that the forwarding nodes are within the transmission range of each other. During the DIO transmission, each node also reports some additional information using the Option field of the DIO message.
- Each node updates the neighborhood information through the DIO message transmission. Based upon the neighborhood information, each node dynamically prioritizes its neighbors in order to construct the forwarder list

Survey routing protocols

- free and greedy hop-by-hop routing protocol, whose performance is proved better than FBR, and of its enhanced version Flood. Link quality is explicitly considered when selecting a relay node for packet forwarding. The performance and applicability of CARP have been evaluated in the real ocean environment. However, there may have unnecessary control packets to be forwarded in CARP when selecting relay nodes for packet forwarding, and these control packets may be avoided in certain situations.
- Other characteristics that make CARP relay selection particularly suitable for implementing multi-hop routing in UWSNs include the following:
 - (i) The use of simple topology information (hop count) for routing around connectivity holes and shadow zones, thus avoiding the well-known pitfalls of geographic routing;
 - (ii) considering residual energy and buffer space, and
 - (iii) taking advantage of power control, if available, for selecting transmission powers so that shorter control packets experience a similar Packet Error Rate (PER) of longer data packets.

Survey routing protocols

- **E-CARP Routing Protocol:**
- E-CARP, which is an enhancement upon CARP, to develop a location-free and greedy hop-by-hop routing protocol for forwarding packets from sensor nodes to the sink node in an energy efficient manner.
- Generally, CARP does not consider the reusability of sensory data collected previously by domain applications in the following time points, which induces sensory data packets forwarding which may not be beneficial to certain applications.
- Therefore, E-CARP allows the caching of sensory data at the sink node, for avoiding these data packets forwarding in the network. CARP requires to reply a PONG control packet whenever receiving a PING control packet, when selecting the most appropriate relay node for packet forwarding. This PING-PONG strategy may not be mandatory when the network topology is relatively steady.
- This observation drives us to improve the relay node selection strategy in CARP, and the relay node adopted previously is given a higher priority to be reused at this moment. Simulation results validate that our E-CARP can decrease the communication cost and increase the network capability to a large extent, especially when the ratio of packet size between control packets and sensory data packets is relatively large.
- E-CARP does not differentiate the priority of different attributes. In fact, sensory data of attributes of more importance should be routed to SN with a higher priority. Besides, sensory data of a certain sensor node may vary following a spatial and/or temporal discipline.

Survey routing protocols

Routing Protocol	Main Results	Simulation Used
RPL	<ul style="list-style-type: none">➤ RPL showed better PRR and Energy consumption➤ RPL showed lesser churn➤ RPL showed high PRR➤ RPL had higher control-traffic overhead➤ RPL able to cater to variety of traffic patterns,➤ RPL is link-layer independent	Contiki/ Cooja
CTP	<ul style="list-style-type: none">➤ In smaller networks, CTP showed better PRR. In larger networks,➤ CTP showed high PRR➤ CTP is only collection-based	Contiki/ Cooja
LOADng	<ul style="list-style-type: none">➤ LOADng caters to more general traffic pattern➤ LOADng has flexible and compressible packet format➤ No single point of failure in LOADng➤ Longer route discovery phase in LOADng➤ More control traffic in LOADng if traffic is predominantly P2P	Contiki/ Cooja

Survey routing protocols

Routing Protocol	Main Results	Simulation Used
LOAD	<ul style="list-style-type: none">➤ In LOAD, control traffic / data traffic➤ LOAD routes longer than RPL routes➤ Higher delay in LOAD due to buffering during route-discovery➤ More collisions in LOAD due to flooding	NS2
CORPL	<ul style="list-style-type: none">➤ CORPL make use of Directed Acyclic Graph (DAG) like RPL➤ CORPL uses an opportunistic forwarding approach	Contiki/ Cooja
CARP	<ul style="list-style-type: none">➤ CARP is a multi-hop delivery of data to the sink for WSN.➤ CARP takes care of link quality while selecting the next-hop node on a route to the sink	Real Time Test-bed
E-CARP	<ul style="list-style-type: none">➤ E-CARP is an enhancement upon CARP,➤ E-CARP is a location-free and greedy hop-by-hop routing protocol for forwarding packets from sensor nodes to the sink node in an energy efficient manner➤ E-CARP does not differentiate the priority of different attributes	Test-bed

Sensor deployment & Node discovery

- What is sensor deployment?

Sensor deployment is **one of the major concerns in multisensor networks**. ... It avoids the unstable coverage caused by the large amount of computation, slow convergence speed, and easily falling into local optimum, which provides a new idea for multisens or deployment.

What is sensor deployment in IoT?

Wireless sensor networks (WSNs) are important components of smart cities. Deploying IoT sensors in WSNs is a challenging aspect of network design. Sensor deployment is performed to **achieve objectives like increasing coverage, strengthening connectivity**, improving robustness, or increasing the lifetime of a given WSN.

Sensor deployment & Node discovery

- **What are the types of sensors in IoT?**

IoT sensors have become critical to improving operational efficiency, reducing costs and enhancing worker safety.

- Temperature Sensors
- Humidity Sensors.
- Pressure Sensors.
- Proximity Sensors.
- Level Sensors.
- Accelerometers.
- Gyroscope.
- Gas Sensors.

Sensor deployment & Node discovery

- **How does sensors work in IoT?**
- An IoT system consists of sensors/devices which **“talk” to the cloud through some kind of connectivity**. Once the data gets to the cloud, software processes it and then might decide to perform an action, such as sending an alert or automatically adjusting the sensors/devices without the need for the user.
- How many sensors are there in IoT?
- **12 Sensor Types** for the Internet of Things. Here you have a list of IoT sensors sorted from most to least popular (according to Google Trends data), including temperature, proximity, pressure, and more.

Sensor deployment & Node discovery

- **What is node discovery in IoT?**
- Since the nodes can be stationary or mobile the network topology frequently changes, therefore node or neighbor discovery is a continuous process. ... The process of collaboration involves the **devices to discover each other and learn network topology.**

Node discovery

- **Challenges in Node Discovery in IoT**
- In order to efficiently do Neighbor discovery for opportunistic networking in IoT the following challenges must be dealt with:
 - **Recognising Presence of nodes:** This is an important challenge for the devices. The nodes need to recognise effectively the presence of other nodes within the communication range. The discovery of the visiting node should be done within a finite time window. Also, the nodes should be able to save resources by understanding that no node is present within the range of communication.
 - **Features of Mobility pattern:** Another challenge for the nodes is to understand the features of mobility pattern exhibited by the nodes. The Mobility models exploit the temporal and spatial pattern characteristics (for example, Levy nature of human walks).
 - **Acquisition of knowledge:** The knowledge acquisition of the node mobility pattern is important to predict the future arrival of nodes. This helps a node to save power by scheduling resources only when a contact is expected.

Node discovery

Neighbor discovery in IoT has the following requirements:

- Minimum technologies that are required to work in mesh-under and route-over configurations.
- To reduce the number of messages exchanged use of multicasting and flooding of multicast messages has to be avoided.
- Efficiency of the link between host and default router has to be optimized.
- To reduce the node active period, provide an option for sleeping of inactive nodes.
- 6LoWPAN header compression [RFC6282] requires propagation of context information to hosts.
- Disseminate context information and prefix information from the border to all routers in a LoWPAN.
- A multihop Duplicate Address Detection mechanism suitable for route-over LoWPANs is required.

Node discovery

- **Advantages of Neighbor discovery for opportunistic networking in IoT scenarios**
- **Extended Lifetime:** Extended lifetime is one of the most important advantages of performing neighbor node discovery. This is achieved by improving power management of both the static and dynamic nodes by efficiently scheduling resources only when the contacts are highly probable to occur.
- **Communication time:** By exploiting knowledge about node arrival time, resources can be tailored to application requirements in useful communication time after the discovery is done.
- **Planning of Communication:** It is possible to plan communication by learning the pattern of when the nodes and their contact time. Neighbor node discovery for IoT devices should be capable of the following:
 - Learning
 - Prediction
 - Low Latency
 - Energy Efficiency

Node Discovery

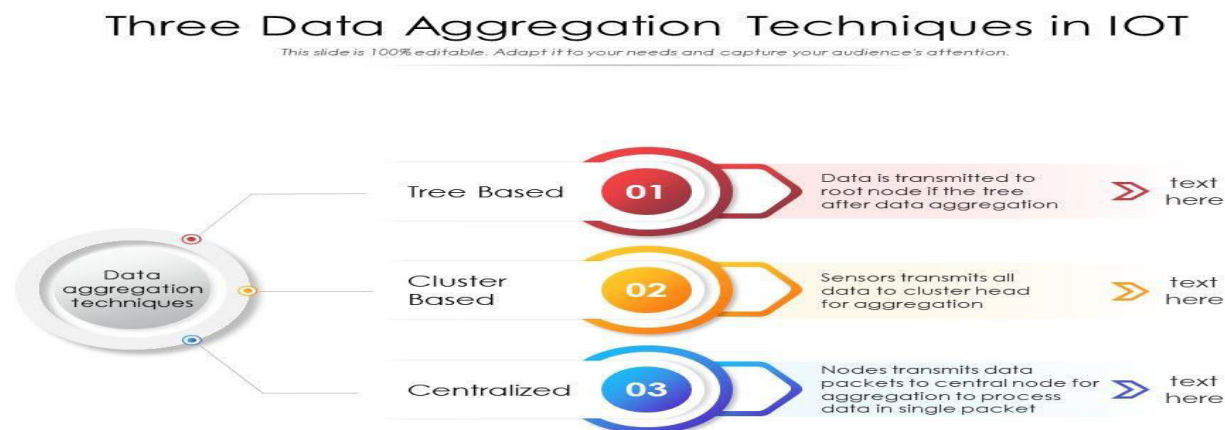
Sl.No	Name of the System	Technique used	Pros	Cons
1	ZebraNet-History based protocol	History based protocol	Data collection with energy efficiency.	Low performance with dynamic network changes.
2	RBTP: Low power mobile discovery protocol through recursive binary time partitioning.	Time synchronized protocol.	Decision taken based on a nodes own battery level.	Nodes does not consider arrival pattern of neighbours.
3	Wakeup scheduling in wireless sensor networks.	Even and odd shifting, ladder approach.	Forward propagation delay of messages is minimized.	Nodes will have to wait till the neighbour wakes up.
4	Context aware resource discovery framework	Arrival time based approach, Q learning used.	Learns higher and lower duty cycles effectively.	Learning is available on static nodes, not available on mobile elements.
5	Searchlight: won't you be my neigjbour.	Asynchronous deterministic approach.	Usage of probe nodes to discover with in shorter period.	Mobility agnostic method.
6	Context aware power management Discovery.	Colocation based approach.	Schedule of neighbours is accounted, making communication better.	More time wasted on data gather.
7	A fully distributed opportunistic approach	Limited flooding and trickle inspired algorithm	High discovery rates.	Time spent on node advertisement.

Data Aggregation & Dissemination

- **What is data aggregation in IoT?**
- Data Aggregation technique is **used to increase the lifetime of network** by collecting information in an energy efficient manner.
- Data aggregation is **any process in which information is gathered and expressed in a summary form for purposes** such as statistical analysis. A common aggregation purpose is to get more information about particular groups based on specific variables such as age, profession, or income.

Data Aggregation & Dissemination

- **What is the need of data aggregation in IoT?**
- The use of different IoT data aggregation methods **eliminates redundant data**. This reduces network traffic by significantly minimizing the number of sent data packages. IoT sensor nodes can also eliminate redundancies in the data received from neighboring nodes before transferring the final data packages.
- **What are the main types of data aggregation being used in the IoT?**
- The data aggregation mechanisms are categorized into three main groups, including **tree-based, cluster-based and centralized**.



Data Aggregation & Dissemination

- **Challenges to the implementation of wireless sensor networks in IoT scenarios**
- To improve our use of this complex technology, we need a closer look at some of [the typical WSN challenges](#). These include:
- **Data volumes:** This calls for re-conceptualizing of the very way things will be connected with one another in favor of [ways that reduce the generation of massive volumes of raw IoT data](#) as well as facing the challenge of processing and storing the data from connected IoT devices;

Data Aggregation & Dissemination

- **Data heterogeneity:** Facing millions of different connected and interconnected IoT devices, one is confronted with the need to organize the incoming data and reduce its complexity;
- **Highly dynamic IoT landscapes:** Facing constantly changing conditions such as night, day, working hours, disconnecting and reconnecting IoT devices, as well as highly heterogeneous device landscapes of ever-growing numbers of IoT devices connected to a network, there arises the need for managing IoT ecosystems in a systematic way.
- In view of these challenges to the use of WSN in IoT applications, different IoT data aggregation methods can overcome both energy-efficiency and data transmission hurdles. The [most common definition of data aggregation](#) is the process of fusing the data from multiple sensors to minimize redundant transmission. In this way, only the fused information is provided to the base station. Typically, data aggregation fuses data from multiple sensors at intermediate nodes and transmits the aggregated data to the base station.

Data Aggregation & Dissemination

- **Why data aggregation?**
- Data aggregation is the process of gathering and summarizing data from multiple sources. Aggregated data is normally found in a data warehouse. There, it can give answers to analytical questions and greatly reduce the time required to query large data sets.
- The main rationale behind data aggregation is that it minimizes energy depletion and the required network bandwidth.
- The use of different IoT data aggregation methods eliminates redundant data. This reduces network traffic by significantly minimizing the number of sent data packages.
- IoT sensor nodes can also eliminate redundancies in the data received from neighboring nodes before transferring the final data packages.

Data Aggregation & Dissemination

- Another aspect to consider is the tradeoff between bandwidth and S10 minutes but over very long distances.
- Since sensor nodes are powered by batteries, saving energy and extending battery life is essential to the IoT data collection effort. Data aggregation is considered an energy-aware data collection technique and is preferred in scenarios where extending battery life is crucial. It is even [known to increase the lifespan of WSNs](#).
- [Energy-aware data aggregation methods](#) include clustered aggregation, tree-based aggregation, in-network aggregation, as well as centralized data aggregation that specifically considers the energy consumption of sensor nodes.

Data aggregation methods in IoT sensor network settings

- a need to identify suitable [data aggregation techniques](#) to collect and analyze incoming data. Typically, at this level, we differentiate between flat IoT data aggregation methods and a hierarchical approach to data aggregation.
- In **flat wireless sensor networks**, all sensors play an equal role—there is no hierarchical arrangement. Every sensor node serves the same purpose and all IoT sensor nodes are peers. One disadvantage of flat wireless sensor networks is that data aggregation takes place only in the sink node area. As a result, network delay can be high. Also, if the sink node fails, this negatively impacts the entire network.

Data aggregation methods in IoT sensor network settings

- With the **hierarchical approach** to wireless sensor networks, there is a hierarchy among the individual nodes based on their capabilities. Roughly, these are divided into base stations, cluster heads, and sensor nodes. The sensor nodes within a given cluster communicate with each other and then communicate with the cluster head. More computing power and increased network transmission capabilities mean less battery life. So one of the main goals of this routing method is to achieve better energy efficiency for the sensors within a cluster.

Data aggregation methods in IoT sensor network settings

- **Cluster-based aggregation**
- This is a hierarchical method best suited for large-scale energy-constrained sensor environments. In such scenarios, it is not efficient for the sensors to transmit the IoT data directly to the sink node (base station). Rather, sensors transmit data to a local aggregator, also known as a cluster head. The cluster head aggregates data from all the sensors in its cluster and transmits it to the sink node. The cluster heads can communicate with the sink node directly via long-range transmissions.
- They can also do [multi-hopping through other cluster heads](#). The typical protocols here include clustered diffusion with dynamic data aggregation (CLUDDA), Low Energy Adaptive Clustering Hierarchy (LEACH), and Hybrid Energy-Efficient Distributed Clustering Approach (HEED)

Data aggregation methods in IoT sensor network settings

- **Chain-based aggregation or string-based aggregation**
- In some scenarios, if the cluster head is located too far from the sensors, communication between the sensors and the cluster head might consume excessive amounts of energy. On such occasions, it is more efficient for sensors to transmit data only to their closest neighbors in the network. Chain-based data aggregation is a hierarchical method whereby each sensor transmits only to its closest neighbor. Nodes are mostly organized into a linear data aggregation chain.
- The node that is located farthest from the base station initiates chain formation. At each step, a node's closest neighbor is selected as its successor in the chain. So a node receives data from one of its neighbors and merges the received data with its own data. It then transmits the fused data further down the chain to its next neighbor. A lead node, similar to the cluster head in cluster-based aggregation, transmits the aggregated data to the base station.
- An example of a chain-based data aggregation protocol is the so-called power-efficient data gathering protocol for sensor information systems (PEGASIS).

Data aggregation methods in IoT sensor network settings

- **Tree-based aggregation**
- In this scenario, data is aggregated through the creation of a data aggregation tree. Sensor nodes are organized in such a way that data aggregation takes place at intermediate nodes along the “tree”. The so-called “root node” only receives an already structured representation of the data. This aggregation technique is suited for applications that necessitate in-network data aggregation. One of the main challenges of tree-based aggregation is the creation of an energy-efficient data aggregation tree that optimizes the network lifespan and minimizes the number of transmissions.
- On average, tree-based methods are known to have high overhead, high energy uniformity, as well as greater strength, flexibility, and scalability as compared to cluster-based methods.

Data aggregation methods in IoT sensor network settings

- **Grid-based aggregation**
- This method is based on dividing the region of a sensor network into several grids. A set of sensors act as data aggregators in pre-defined regions of the sensor network. So we have a data aggregator (also known as an integrator) that is fixed in each grid. And the array of sensors acts as an aggregator/integrator within this particular region of the sensor network. The sensors in that particular grid transmit the data *directly* to the data aggregator that aggregates the data from all IoT sensors within the grid.
- In grid-based aggregation, the individual IoT sensors within a grid do not communicate with each other. Grid-based data aggregation is known to adapt to dynamic changes in the network.

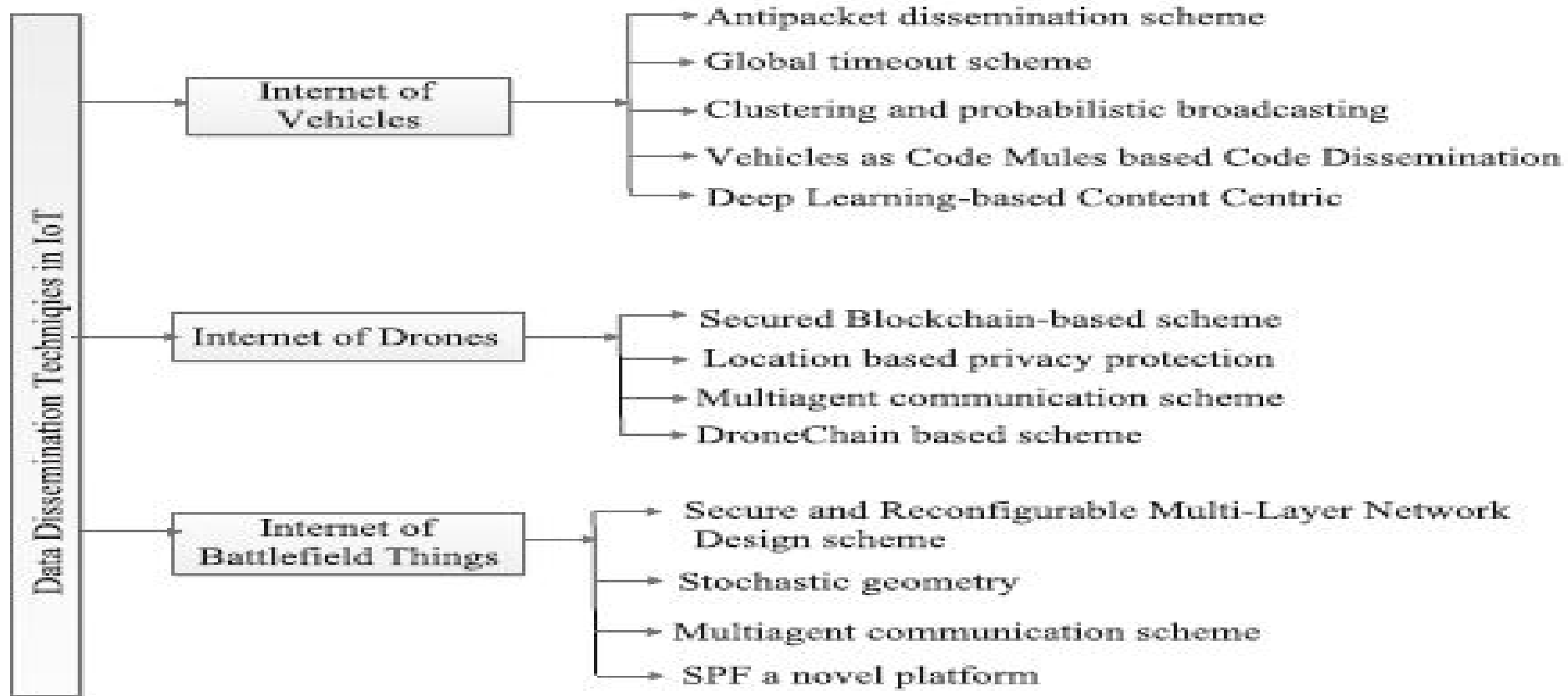
Data aggregation methods in IoT sensor network settings

- **Structureless aggregation**
- Structureless data aggregation does not involve any kind of architecture. Communication takes place from any node to any node within the network. In some cases, as for example in event-based applications that vary by event region, structureless aggregation is the preferred approach.

Data Aggregation & Dissemination

- **What is Data Dissemination?**
- Data Dissemination protocols are required to distribute the data and code between various sensor nodes and it provide periodic updates to sensor programs.
- Data dissemination is **the distribution or transmitting of statistical, or other, data to end users.** ... Proprietary data dissemination requires a specific piece of software in order for end users to view the data. The data will not open in common open formats.
- **What are the purpose of data dissemination?**
- **Advances in information and communication technologies** have made the global distribution of information and knowledge effortless and have made data available to multiple users the instant they are produced.
- **What are the principles of data dissemination?**
- Data should be disseminated in formats that are accessible and accompanied by documentation that is clear and complete. **Dissemination should be timely, and information should be made readily available on an equal basis to all users.**

Data dissemination for Internet of Things applications solution taxonomy



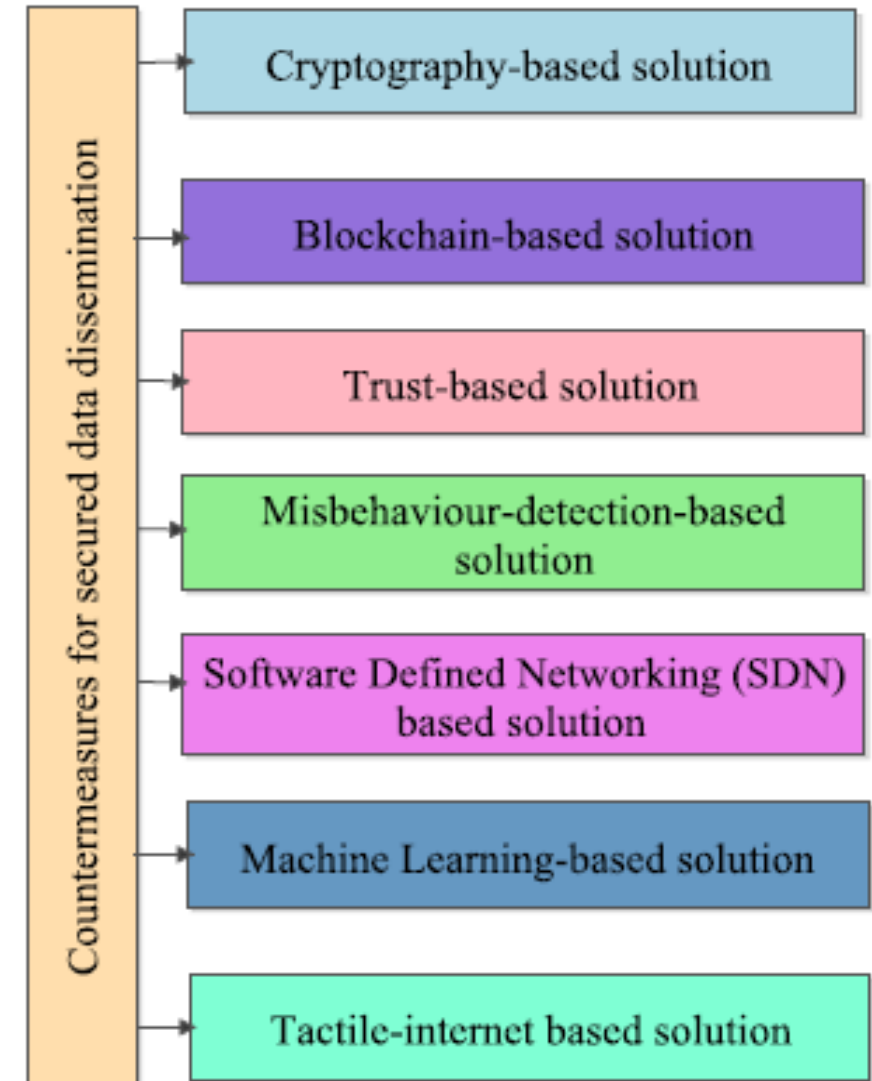
Classification of data dissemination protocols in Internet of Things environment based on structure of the network

y	Data dissemination protocols	Pros	Cons	Structure of the network
	Flooding	Simple	Energy Consumption	Flat networks
	SPIN	Energy aware	No assurance of data delivery	Flat networks
	Gossiping	Less energy Requirement	Slow Performance	Flat networks
	Directed diffusion	No implosion	—	Flat networks
	One-phase pull diffusion	Flexible to number of nodes	Excess Control Overhead	Flat networks
	Gradient-based routing	Enhanced Network lifespan	—	Flat networks
	Low Energy Adaptive Clustering Hierarchy(LEACH)	Data accuracy	Not suitable for Energy Constrained sensors	Cluster Network
	HEED	Network lifetime	Only Cluster to Cluster communication possible	Cluster Network
	CLUDDA	Data Aggregation	More Complex	Cluster Network
	PEGASIS	Energy efficient	Not Secured	Chain based network

Countermeasures for secured data dissemination in Internet of Things applications

There are a variety of countermeasures depending upon the type of attacks, such as malware attack, wormhole attack, and Sybil attack. The taxonomy of various countermeasures is shown in Figure

- **Cryptography-based solution:** It provides solutions that are key-based to securely communicate and exchange the data among different entities for any IoT applications. It includes symmetric searchable encryption, asymmetric searchable encryption, anonymous broadcast encryption, and homomorphic signature. Modern and classical cryptographic algorithms such as advanced encryption (AES), triple data encryption standard (DES), Rivest-Shamir-Adleman algorithm, and DES provides security with a minimum count of required resources. Hash functions are also used to create a summary of the message ($hM = \text{hash}(M)$) that can not be easily guessed by an untrusted party.



Countermeasures for secured data dissemination in Internet of Things applications

- **BC-based solution:** The existing cryptographic techniques such as Homomorphic Encryption Schemes, Secure Service Discovery, key Management Mechanism, and one-Way Trapdoor Permutation can also be used to secure data distribution between cloud and end user. But, it has some issues such as key validation, trust among entities involved in IoT. These issues can be resolved by using one of the emerging technology known as BC technology. It has already gained popularity among the research community because of its characteristics such as immutability, nonrepudiation, security, provenance, trust, and traceability.

Countermeasures for secured data dissemination in Internet of Things applications

- **Trust-based solution:** In the traditional dissemination schemes, the confidential information is passed through the unsecured cloud or infrastructure. Hence, there is a requirement of the third party to verify or authenticate the integrity of the information. So secure service discovery or BC can be a viable solution where the involvement of a trusted third party is expected.
- **Misbehaviour-detection-based solution:** Misbehavior of different entities such as sensors, base stations, administrator authority in the IoT environment can be considered as a malicious or unauthorized activity. These activities can not be detected in normal scenarios due to the absence of previous history, feedback, and set of specific procedures. In such cases, the BC technology can be a viable solution where all nodes/higher percentage of nodes agreed; then only the transaction is validated. It provides consensus in the network, and once a consensus is achieved, a block is mined and stored as an immutable ledger in the chain with all nodes having the same copy of the ledger.
- **Software-defined networking-based solution:** This platform is specially designed to provide secure and more flexible network solutions. A centralized software-defined network controller plays a vital role in managing the network resources in an easier way.^{85,86} Hence, this technology became more popular in the last decade.

Countermeasures for secured data dissemination in Internet of Things applications

- **Machine learning-based solution:** The countermeasures mentioned above can identify only those attacks which are in its database. If an attack is encountered during data distribution from cloud to end user other than a dictionary attack, then the methods mentioned above are not able to detect it. Various machine learning algorithms (such as classification, clustering, and Bayesian network) can be used in such scenarios to predict the attacks based on specific patterns. This is the new technological trend in detecting the cyberattacks and used to calculate the probability of the occurrence of attacks. The patterns are stored in a central node, and other nodes perform decision analytics on the stored data to find and match patterns and detect the signature of the attack. Hence, this technique can be suitable to detect failures during data dissemination in any IoT-based applications.
- **Tactile-Internet based solution:** In the traditional dissemination schemes, network latency is one of the major concern which affects on the performance of the network. Tactile internet is the new revolution in IoT. It combines ultra-low latency with extremely high availability, security, and reliability.

References

- <https://www.geeksforgeeks.org/wireless-media-access-issues-in-internet-of-things/>
- <https://www.geeksforgeeks.org/responsibilities-and-design-issues-of-mac-protocol/>
- <https://smartify.in/knowledgebase/iot-based-home-automation-system/>
- <https://www.simform.com/blog/home-automation-using-internet-of-things/>
- <https://premioinc.com/blogs/blog/five-useful-video-analytics-applications-for-iot-surveillance>
- <https://nexusintegra.io/7-industrial-iot-applications/>
- record-evolution.de/en/iot-data-aggregation-methods-in-wireless-sensor-landscapes-on-making-data-valuable/

Home Automation

UNIT - IV Network and communication aspects

What Is Home Automation?

- Home automation is the automatic control of electronic devices in your home. These devices are connected to the Internet, which allows them to be controlled remotely.
- With home automation, devices can trigger one another so you don't have to control them manually via an app or voice assistant. For example, you can put your lights on schedules so that they turn off when you normally go to sleep, or you can have your thermostat turn the A/C up about an hour before you return to work so you don't have to return to a stuffy house.
- Home automation makes life more convenient and can even save you money on heating, cooling and electricity bills.
- Home automation can also lead to greater safety with Internet of Things devices like security cameras and systems. But hold up; what's the Internet of Things?

Internet of Things vs. Home Automation

- The Internet of Things, commonly known as IoT, refers to any device that's connected to the Internet that isn't normally; for example, a smart light bulb that you can turn on and off via an app.
- All home automation devices are IoT devices, which can be automated to trigger one another.
- So while IoT refers to the devices themselves, home automation is what you can do with the IoT devices to make your life just a tad bit easier.

How Does Home Automation Work?

- Home automation works via a network of devices that are connected to the Internet through different communication protocols, i.e Wi-Fi, Bluetooth, ZigBee, and others.
- Through electronic interfaces, the devices can be managed remotely through controllers, either a voice assistant like Alexa or Google Assistant or an app.
- Many of these IoT devices have sensors that monitor changes in motion, temperature and light so the user can gain information about the device's surroundings.
- To make physical changes to the device, the user triggers actuators, the physical mechanisms like [smart light switches](#), motorized valves or motors that allows devices to be controlled remotely.

How Home Automation Works?

- Home automation works on three levels:

1. Monitoring: Monitoring means that users can check in on their devices remotely through an app. For example, someone could view their live feed from a smart security camera.



Nest Hello and Google Home Hub

How Home Automation Works?

- **2. Control:** Control means that the user can control these devices remotely, like panning a security camera to see more of a living space.

- **3. Automation:** Finally, automation means setting up devices to trigger one another, like having a smart siren go off whenever an armed security camera detects motion.



Amazon Echo Show and Box



Amazon Alexa and Amazon Cloud Cam

Home Automation System Components

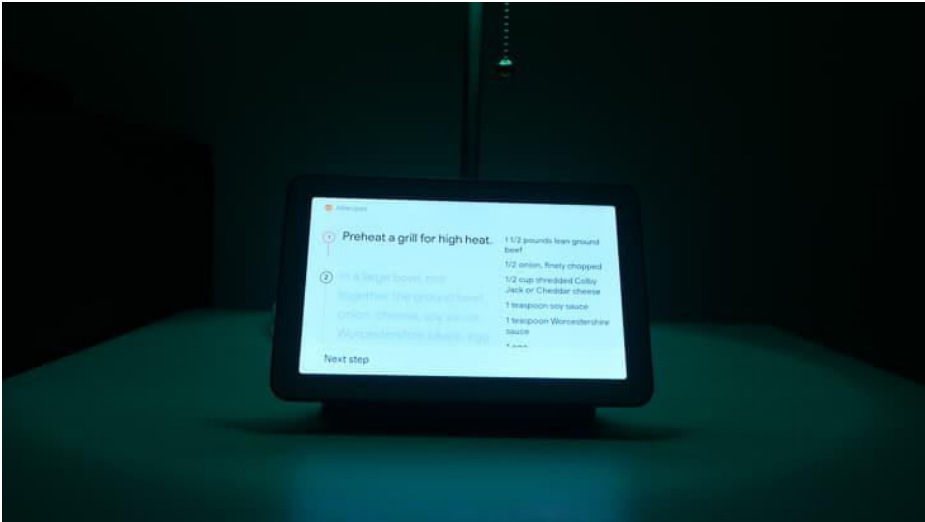
- While some home automation systems require hubs, some mobile applications connect directly to a router, which connects directly to an IoT device. Of course, it's preferable when there's no hub, as that's just an added cost on top of the cost of the IoT device itself.

Home Automation System Components

- **Remote Control**
- The hallmark of home automation is remote control, which is done through either a mobile application or through a voice assistant.
- **Mobile Application:** The mobile application allows users to control their devices in real-time, whether it's shutting off the outdoor lights or opening the [smart garage door](#) for a neighbor. The app is also where users set schedules, create scenes, groups of IoT devices, and customize device settings, like having your living room lights set to the perfect shade of blue. Most of the IoT devices we've reviewed have apps for Android and iOS devices, making them compatible with the majority of mobile devices and tablets.
- **Voice Assistants:** If home automation is the sundae, think of voice assistants as the cherry on top. With voice assistants, you can use your voice to control devices, whether that's disarming a security system as you walk in the front door, showing your video doorbell's footage on your Echo Show device, or setting a timer on a smart speaker while your hands are full of cooking utensils. Most IoT devices work with one of three voice assistants: Alexa, Google Assistant, and Siri.

Home Automation System Components

Recipes on the Google Nest Hub



Loup Ventures Annual Digital Assistant IQ Test Results²

Answered Correctly	Understood Query	Answered Correctly
Alexa	79.80%	99.90%
Google Assistant	92.90%	100%
Siri	83.10%	99.80%

Home Automation System Components

- **Alexa:** Alexa is Amazon's voice assistant that's built into Echo Show and Echo Dot devices (for more information, check out our [Echo Show review](#) and our [Echo Dot with Clock review](#)). Alexa is the voice assistant we see integrated into the highest number of smart home devices from companies like SimpliSafe, Ring Alarm and Vivint.



Amazon Alexa

Home Automation System Components

- **Google Assistant:** Google Assistant, is, as you can imagine, Google's voice assistant. Although Google Assistant has fewer "skills" or "actions" than Alexa, it has been proven to be the most accurate voice assistant in terms of understanding and answering queries correctly. To get Google Assistant, you'll need a smart speaker or a smart display; read our [Nest Mini review](#) or [Nest Hub review](#) to get started.



Google Nest Mini

Home Automation System Components

- **Siri:** Siri is Apple's voice assistant that's integrated into the iPhone. While Siri holds 35 percent of the global market share for voice assistants, compared to nine percent and four percent with Google Assistant and Alexa,³ respectively, there aren't too many IoT devices that work with Siri. Rather, the voice assistant is used mainly on iPhones and iPads in contrast to home automation devices, where Alexa and Google Assistant reign supreme.

Cloud Computing with Home Automation

- Rather than basing home automation systems off a dedicated IP address or high-end computer, many systems are based on a cloud, which is both more affordable and easier to use.
- For example, the Nest cameras don't have slots for micro-SD cards, which would have allowed footage to be stored locally. Rather, all recorded footage is automatically uploaded to a cloud server, only accessible through a Nest Aware subscription. In general, cloud computing is incredibly popular on the Internet, so IoT devices are no exception.

Cloud Computing with Home Automation

- **Control Protocols**
- The way that IoT devices connect to the Internet and each other is their control protocol; if IoT devices are people, think of the protocol as their common languages. Like on Earth, there are a few different languages, or protocols, that devices can speak, including:
- **WiFi:** WiFi is by far the most common control protocol; it means that your IoT device will use the regular Internet provided by your Internet Service Provider. While this doesn't require an additional hub, note that it can slow your web surfing speeds down, especially if you have a ton of different IoT devices set up at once.
- **Z-Wave:** Don't want to mess with your home's WiFi? Z-Wave is a wireless technology that won't interfere with your WiFi; rather, it operates on low power at 908.42 Mhz in the U.S and Canada.⁵
- **ZigBee:** Similar to Z-Wave, ZigBee is a mesh network and universal language that lets IoT devices communicate.
- **Thread:** Thread is another low-power, wireless mesh networking protocol based on an IP address open standard; it lets IoT devices connect to each other and the cloud.
- **Bluetooth:** Finally, Bluetooth is another mesh technology that lets people control and monitor IoT devices and automate systems.
- For most people, WiFi-connected devices will be sufficient, but for more advanced smart homes, you might want to switch to a mesh network like Z-Wave or ZigBee.

Advantages and Disadvantages of Home Automation

Advantages	Disadvantages
Energy Efficient	Cost
Hands-free convenience	Internet Reliance
Enhanced Security	Setup and Configuration
Save Time with Automated Tasks	More Technical Security Threats
Customization	Different Protocols (Z-Wave, Zigbee etc.)

Pros and Cons of Home Automation

Sl.No	Pros	Cons
1	Remote access: Being able to control devices remotely means things like unlocking the door for a plant sitter without having to leave a key under the mat.	Costs: IoT devices are certainly more expensive than their non-WiFi-connected counterparts. For example, the average smart bulb costs around \$32, while the average regular light bulb is about \$5. Of course, you have to factor in the additional features like remote control, dimming, 16 million different colors and voice integrations, to name a few, but overall, home automation isn't cheap, depending on where you shop.
2	Comfort: You know when you're all comfy in bed but realize you've left the bathroom light on? With smart light bulbs, you can turn them off from the comfort of your bed without having to leave those high thread count sheets.	Security issues: It's scary but true: anything that has to do with the Internet, whether it's browsing Etsy for a new bedspread or checking in on a motion notification from a smart security camera, can be hacked, and that includes IoT devices. Unfortunately, we've seen a fair share of hackings and security breaches from large tech companies that manufacture IoT devices; Ring's cameras, for example, were famously hacked, allowing the live feeds to be compromised. ⁷ Of course, this is an issue you wouldn't have with devices that aren't connected to the Internet, but if you want IoT devices, you'll have to adhere to some best digital security practices, detailed later on.

Pros and Cons of Home Automation

Sl.No	Pros	Cons
3	Energy efficiency: How many times have you left the heat on blast while you're out of the house for eight hours? With home automation, you can set things like thermostats on schedules to make sure you're not wasting energy. A study found that Nest thermostats in particular can save about 12 percent on heating and cooling costs, ⁶ for example. That means that over time, these smart thermostats can actually pay for themselves in savings.	New technology: Since IoT is a relatively new technology, you may run into some bugs, like devices having trouble connecting to the Internet or experiencing lag, depending on the device's make and model.
4	Convenience: Being able to control devices remotely or via voice commands, set them on schedules, and even sync them with the sunrise and sunset is nothing is not convenient. Imagine being able to come down in the morning to freshly made toast without you having to push a button!	Surveillance: If privacy is a huge concern, then smart security is probably not for you, as users can livestream footage from the camera's respective app. Instead, you might want to opt for a local alarm system; SimpliSafe has an option if you don't pay for the monthly plan, detailed in our SimpliSafe security review .

Pros and Cons of Home Automation

Sl.No	Pros	Cons
5	Safety: Finally, there are many smart security products that can increase your home's safety, like sensors for doors and windows, security cameras that can detect people, and video doorbells that let you greet whoever's knocking from anywhere with Internet.	Dependency on Internet: The basic requirement for the smart home system is the internet. Without a good and strong internet connection, you will not be able to take control of this. If there is no internet connection for some reason, there is no other way through which you can access and control your system.
6	Increased homes value: According to top real estate analysts, an investment in smart home technology can help increase your home's resale value. Many home-buying consumers are willing to pay for the features that are associated with smart homes. Check professional home reviews for further clarification.	Dependency of the Professionals: The basic requirement for the smart home system is the internet. Without a good and strong internet connection, you will not be able to take control of this. If there is no internet connection for some reason, there is no other way through which you can access and control your system. so it is very important to pick a good quality internet service provider here in Kozhikode. Airtel, Jio and Asianet are some good service providers. In case there is a problem with the smart home system, you cannot simply call a handyman or someone similar to repair or manage the bug. You will have to depend on the professionals. Only the company professionals can

What Is A Smart Home?

- A smart home is any home that includes automated, Internet of Things devices connected to mobile applications. Using these IoT devices, users can control many things in their home from lights to security systems to appliances.
- Increasingly, more and more homes are built with automation from the original construction, but technically, any home with an internet connection and IoT devices counts as a smart home.

How To Set Up Home Automation

- Setting up your home automation system is actually a lot less complicated than it may seem. You can either wing it and buy a smart home product that sounds like it's up your alley, or set up your smart home more strategically by following these easy steps:
- **Smart home ecosystem:** First, choose which “smart home ecosystem” you want to be a part of, and by that we mean Amazon or Google, most likely. This will determine which voice assistant you use, which will then determine which IoT devices will work with your system. Of course, you can use products that work with both Alexa and Google Assistant, but it might be a little confusing to have to remember which voice assistant to use with which IoT device, so we recommend sticking to one or the other. Once you've decided which voice assistant is for you, buy a compatible smart speaker or a smart display to kick off your home automation system.

How To Set Up Home Automation

- **Control protocol:** Next, decide which protocol you want your devices to communicate with, be it WiFi, Z-Wave, ZigBee or others. If you're just starting off with a smart home, WiFi will be your most straightforward option, as most IoT devices work with WiFi.
- **Types of products:** Next, go room by room and decide which types of products you'll need, be it security cameras, light bulbs, locks, coffee makers, and the like. Our buying guide below can help!
- **Brands:** Then, do some research as to the best smart home companies; the reviews on our website go over the most popular brands like Ring, Nest, SimpliSafe, Alder Security, and more.
- **Devices:** Now it's time to actually buy your IoT devices. We recommend buying in bulk, as many companies offer discounts for larger packages.

How To Set Up Home Automation

- **Installation:** Now, it's time to actually install the IoT devices where you want them. Most IoT devices have DIY installation, meaning you can do it yourself for free. However, some companies like Vivint and ADT require professional installation for their smart security systems, so be sure to factor in installation cost, if any, to your bottom line. To learn more, read our pages on [Vivint's costs](#) and [ADT's costs](#).
- **Customize settings:** So you've researched, purchased and installed your IoT devices in your home. What's next? Now, the fun can really start, as you can customize the devices to your liking, whether that means setting them on schedules, dimming lights, or having devices trigger one another. We'll talk more about these features in a bit, but first, let's talk about the IoT devices actually available on the market today.

Internet of Things Devices-Home Automation

- In 2015, there were 15 billion IoT devices. By 2020, that number had ballooned to 200 billion⁸ IoT devices and counting. While we can't list all of the IoT devices available, as that list is ever-growing, here are some of the most popular:
- **Lights:** Smart lights are one of the more affordable IoT devices out there, and they make adjusting your lighting more convenient and customizable than ever before. With most smart bulbs, we could change their color, dim them, set them onto schedules, or even have them blink to the beat of our music. That beats a \$3 bulb from the hardware store!
- **Thermostats:** Smart thermostats let us adjust our home's temperature remotely as well as set it onto schedules, saving our money on heating and cooling.
- **Locks:** Smart locks definitely upped our home's security. They locked automatically as soon as we exited the home, but if we needed to let someone in when we weren't home, we could either unlock them through the app or give our guests a temporary passcode, certainly safer than leaving a key under the welcome mat. Explore your options in our picks for the [best smart locks](#).
- **Video doorbells:** Video doorbells are essentially outdoor cameras that may or may not be hardwired into your existing doorbell setup, if you have one. We got notified whenever the doorbell was pressed or the camera detected motion or a person, depending on its artificial intelligence capabilities.

Internet of Things Devices-Home Automation

- **Security cameras:** Security cameras let you see what's going on at home from a mobile application; you'll also be notified of motion or people, again, depending on the camera's AI.
- **Security systems:** Smart security systems typically include motion, entry and glass break sensors, alerting you of motion, doors and windows opening and closing, and, you guessed it, glass breaking. For more information, check out our take on 2021's [best home security systems](#).
TVs and remotes: Google, turn the volume up 10 percent! In our home, we use smart TVs like Apple TV, Fire TV and Chromecast, which is either built into smart TVs or plugs into a TV's USB port.
- **Speakers:** Smart speakers are often the basis for a smart home ecosystem, allowing for voice commands through the voice assistants. For example, when we tell our Chromecast to pause, we're not speaking directly to the Chromecast device plugged into our TV, but instead our Nest Mini, which has the speaker and microphone necessary for us to communicate with Google Assistant.

Internet of Things Devices-Home Automation

- **Displays:** Smart displays work the exact same way as smart speakers, with voice assistants built-in; the major difference is that they have screens and often cameras, allowing for more entertainment and video chatting options. Smart displays tend to cost a lot more than smart speakers, so if you're on a budget, we'd recommend going with a smart speaker over a smart display.
- **Medical care:** If you have a senior in your life you'd like to care for and monitor remotely, there are a number of WiFi-connected medical alert systems available, many of which include detection for falls.
- **Other IoT products:** We've seen everything from Alexa-enabled microwaves to smart plugs, scales, [smoke detectors](#) and CO detectors. While our site focuses on smart home security like cameras and systems, home automation goes much further, with IoT devices in a number of different categories. Plus, you can [turn off your smoke alarm](#) from your phone, which is incredibly convenient.

IoT Features

- Once your IoT devices are bought and set up, it's time to create the home automation functionalities that sold you on the devices in the first place.
- **Remote control:** First and foremost, all home automation devices can be controlled remotely through a mobile application, whether that means disarming a security system for a neighbor, saying hi to a visitor through a video doorbell, or shining a light on an overnight guest who can't find the switch themselves.
- **Voice assistants:** Most IoT devices can also be controlled by voice commands via voice assistants, most commonly Alexa and Google Assistant. Schedules: Many IoT devices can also be put onto schedules so that they turn off and on automatically throughout the day. This is particularly useful for smart lights and thermostats, things that you may forget to adjust as you enter and exit your home each day.
- **Geofencing:** To make things even easier, you can connect the GPS onto your phone to certain IoT devices to make them turn off and on based on your location. An example? We had our doors unlock whenever we were nearby, which saved us the trouble of searching in our bag for our keys.

IoT Features

- **Triggers:** Devices of the same brand, or of different brands, can trigger one another, depending on their compatibility. For example, Ring devices can obviously work with each other, like having a security system trigger an outdoor light to go on. However, Ring products also work with third-party IoT devices from companies like Dome, First Alert, EcoLink and GE, allowing for more home automation opportunities. Learn more about Ring's smart home compatibility on our [Ring costs](#) page.
- **IFTTT:** Got two devices that can't connect directly on the app? Some devices work with IFTTT, which stands for If This Then That. IFTTT lets devices of different brands trigger one another; for example, Wyze cameras work with IFTTT, so they can work with Arlo cameras even though the companies don't have a direct partnership. To learn more, read our [Wyze cameras review](#) and our [Arlo cameras review](#).
- **App:** Finally, each IoT device has a corresponding app that allows for all of the above features, so it's important that it's user-friendly. As software updates can make improvements, be sure to check the app's current ratings from wherever you downloaded it.

IoT Features

- **Home and away modes:** This doesn't apply to all IoT devices, but some smart home products like bulbs can be set to what's called home and away modes. Consider this: many people keep their lights on all day to make it seem like they're home, supposedly preventing burglaries. However, this is pretty unrealistic, as even when you're home, you probably don't leave all the lights on all the time. With away mode, the lights will turn off and on at random, which more closely mimics real life. Home mode, on the other hand, may have some devices off and some devices on, customized to your liking so you can access it easily whenever you're home.
- **Scenes:** Scenes are groups of IoT devices that you can control at once rather than having to control each one individually. For example, we have all of the smart bulbs in our living room grouped together into a scene so we can dim them all at once.
- **Energy monitoring:** Want to see exactly how much energy your IoT device is using? Some bulbs and thermostats have energy monitoring so you can see how much you're saving.

Securing IoT Devices

- With some IoT devices, digital security can be more of an afterthought, not originally built into many first-generation models. However, as security breaches become more commonplace, many manufacturers are changing their ways, making their IoT devices less hackable. But ultimately, it's up to the user to take advantage of these [digital security](#) features. Here's how:
- **Secure router:** One of the most straightforward ways to secure your home automation system is to use a secure router from a company other than your Internet Service Provider; we recommend looking into routers from NETGEAR, Linksys, and TP-Link.
- **Privacy policy:** Sure, it may be time-consuming and more than a little bit boring, but it's important to actually read the company's privacy policy to see what customer data they keep and how they share it and sell it to third parties.
- **Name router:** Be sure to give your router a name other than the one that it came with.

Securing IoT Devices

- **Encryption:** Choose a strong encryption method for your WiFi, like WPA2.
- **WiFi password:** Make sure your WiFi network has a long, complicated, and [unique password](#). Of course, this makes it harder to add guests, but it also prevents others from hopping on (and hacking your IoT devices).
- **Separate WiFi network:** For the most security for your IoT devices, consider getting a separate WiFi network for IoT devices only. This will also create faster Internet speeds all around.
- **Password hygiene:** Aside from your WiFi network, your IoT account should have a password that's not repeated on any other account; no old, weak or repeated passwords allowed!
- **Device settings:** Often, devices have features enabled by default that aren't necessary, and that can make your IoT devices more hackable. Be sure to turn off these features when not in use, like WiFi, Bluetooth, and knowing your location.

Securing IoT Devices

- **Software updates:** Although change can be scary, software updates are a good thing! They often include updates specifically targeted at improving digital security, so be sure to perform all software updates as soon as they're available.
- **Authentication:** Some companies like Ring have added two-factor authentication to their accounts, meaning to sign in, we had to enter a passcode that was sent to our phones. This made sure that only us, the authorized users, could access our accounts. For even more authentication, look for accounts that allow for fingerprint or facial ID, known as multi-factor authentication. You can also add on either type of authentication through a password manager; for example, we use LastPass' Touch ID to access all accounts on our iPhones.
- **VPN:** Finally, if you're using an IoT device on a public Wi-Fi network, like a smart plug powering your laptop in a coffee shop, connect to a VPN, or Virtual Private Network, to encrypt your web traffic and hide your IP address.

References

- <https://www.security.org/home-automation/>

CSPC702-EMBEDDED SYSTEMS AND INTERNET OF THINGS (IOT)

UNIT - IV Network and communication aspects

Syllabus

- **UNIT - IV Network and communication aspects**
- Wireless medium access issues, MAC protocol survey, Survey routing protocols, Sensor deployment & Node discovery, Data aggregation & dissemination- Applications of IoT: **Home automation**, Industry applications, Surveillance applications, Other IoT applications.

Applications of IoT: Home automation

Smart home automation using IoT

- **What is home automation using IoT today?**
- Today, a smart home lives up to the consumer's expectations and sometimes even exceeds them, and using sensors, devices, appliances and the whole spaces in your house constantly collect data on how you use them.
- They learn about your habits and determine consumption patterns using complex algorithms.
- These insights then help personalize your experience at a granular level.
- Based on Internet of Things, smart home devices of the new generation use their sensor datum to automatically adjust regimes to your routines. They monitor your location in real-time and turn the heating on and off accordingly.
- The best part is that you really don't have to do anything. Smart thermostats rely on their algorithms to personalize home temperature to your preferences and save you good money on reduced energy use.

Applications of IoT: Home automation

Home automation has three major parts:

- Hardware
 - Software/apps
 - Communication protocols
- ✓ Each of these parts is equally important in building a truly smart home experience for your customers. having the right hardware enables the ability to develop your IoT prototype iteratively and respond to technology pivots with ease.
 - ✓ A protocol selected with the right testing and careful consideration helps you avoid performance bottlenecks that otherwise would restrict the technology and device integration capabilities with sensors and iot gateways.
 - ✓ Another important consideration is the firmware that resides in your hardware managing your data, managing data transfer, firmware ota updates, and performing other critical operations to make things talk.

Applications of Home Automation

- Rebuilding consumer expectations, home automation has been projected to target wide array applications for the new digital consumer. some of the areas where consumers can expect to see home automation led IoT-enabled connectivity are:
- lighting control
- hvac
- lawn/gardening management
- smart home appliances
- improved home safety and security
- home air quality and water quality monitoring
- natural language-based voice assistants
- better infotainment delivery
- ai-driven digital experiences
- smart switches
- smart locks
- smart energy meters

The list is still not exhaustive and will evolve over the time to accommodate new IoT use cases.

Home Automation Components

- A realistic model of a smart home developed by the following given major components. The major components can be broken into:
 - iot sensors
 - iot gateways
 - iot protocols
 - iot firmware
 - iot cloud and databases
 - iot middleware (if required)
- iot sensors involved in home automation are in thousands, and there are hundreds of home automation gateways as well. most of the firmware is either written in c, python, node.js, or any other programming language.
- the biggest players in iot cloud can be divided into a platform-as-a-service (paas) and infrastructure-as-a-service (iaas).

Major iot paas providers

- aws iot
- azure iot
- thingworx
- ubidots
- thingspeak
- carriots
- konekt
- tempoIQ
- xively
- ibm Bluemix

Characteristics of IoT Platforms

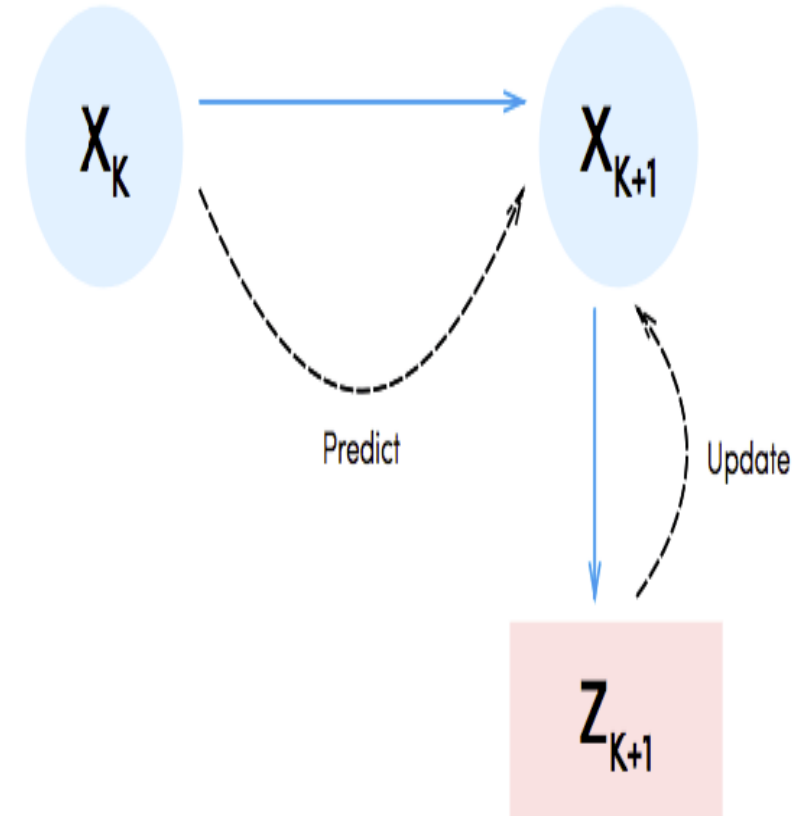
- Again, these platforms are extremely divided over the iot application and security-related features that they provide. a few of these platforms are open source.
- let's have a look at what you should expect from a typical iot platform:
- device security and authentication
- message brokers and message queuing
- device administration
- support towards protocols like coap, mqtt, and http
- data collection, visualization, and simple analysis capabilities
- integrability with other web services
- horizontal and vertical scalability
- websocket apis for real-time for real-time information flow
- apart from what we mentioned above, more and more platform builders are open sourcing their libraries to developers. take for example the dallas temperature library for [ds18b20 for arduino](#) was quickly ported because of open source development to a new version that helped developers to integrate [ds18b20 with linkit one](#). understanding these things become crucial as iot tends to evolve continuously and having an equally responsive platform makes it business safe to proceed.

Home Automation Sensors

- There are probably thousands of such sensors out there that can be a part of this list, but since this is an introduction towards smart home technology. IoT sensors for home automation by their sensing capabilities:
- temperature sensors
- lux sensors
- water level sensors
- air composition sensors
- video cameras for surveillance
- voice/sound sensors
- pressure sensors
- humidity sensors
- accelerometers
- infrared sensors
- vibrations sensors
- ultrasonic sensors
- depending upon what you need, you may use one or many of these to build a truly smart home iot product. let's have a look at some of the most commonly used home automation sensors.

Temperature Sensors

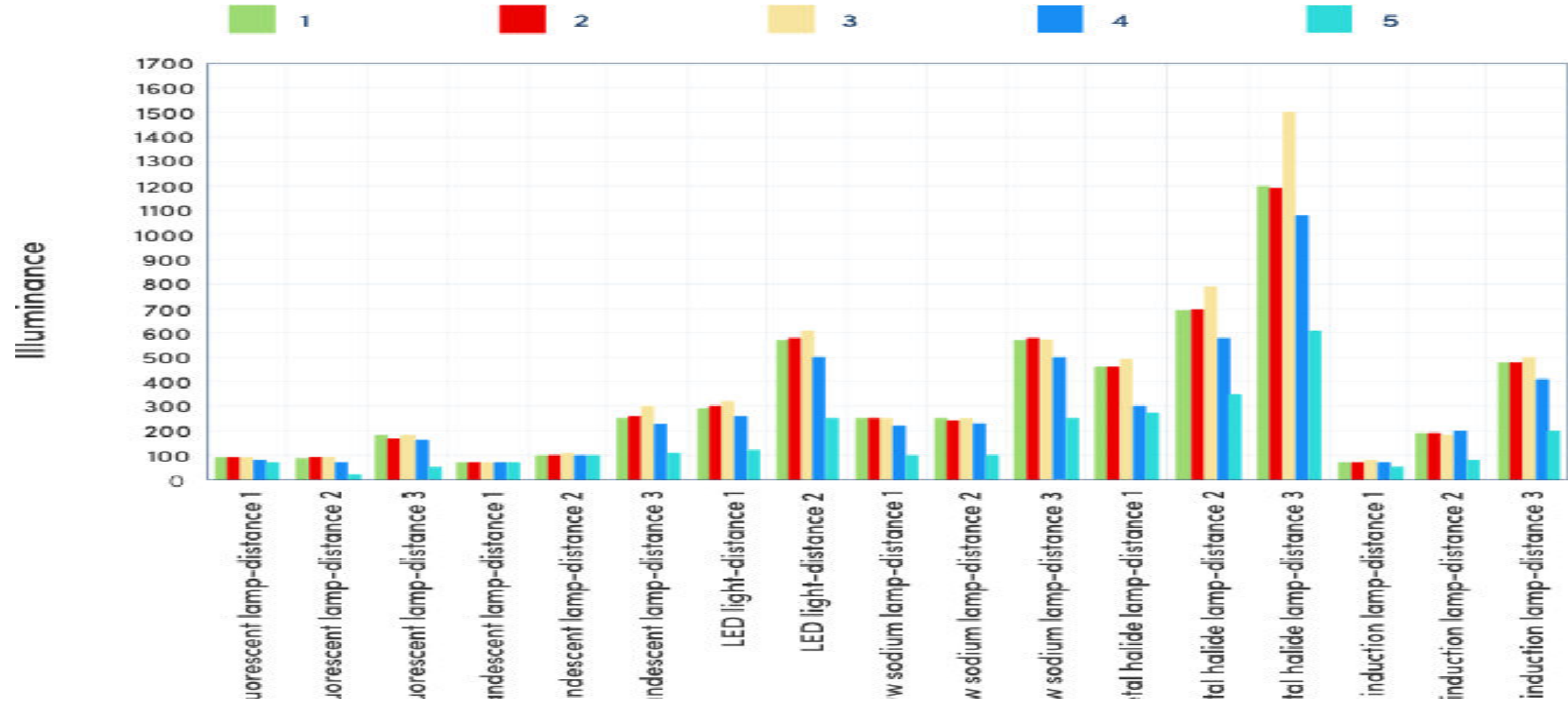
- The market is full of them, but the famous temperature sensors are dht11/22, ds18b20, lm35, and msp430 series from ti. the msp430 series is more accurate than the rest, but at the same time, it is one of the most expensive for prototyping or initial product testing purposes. msp430 tops all temperature sensors, as the precision and battery consumption is minimal with them.
- The dht11 has a very restricted temperature range and suffers from accuracy issues. dht22, on the other hand, is a little bit more accurate but still, doesn't make it as the preference.
- The ds18b20, on the other hand, is more accurate, as opposed to digital temperature sensors like the dht22 and 11. dallas temperature sensors are analog and can be extremely accurate down to 0.5 degrees.
- Take note that often, the temperatures that you directly sense from these sensors may not be very accurate, and you would occasionally see 1000 f or greater values no matter what you are doing.



Lux Sensors

- Lux sensors measure the luminosity and can be used to trigger various functions range from cross-validating movements to turn the lights on if it becomes too dark. Some of the most popular light sensors are TSL2591 and BH1750.
- Recent tests to include TSL2591 and BH1750 into low-powered IoT devices have found them to be working fairly good for most of the use cases.
- The study was done by [Robert and Tomas](#) that shows how these two compare against a spectrometer and a photodiode.
- To get a good idea of whether these two sensors would suffice your needs we would suggest illuminance tests followed by normalization of the data to observe deviations under various situations.

Lux Sensors

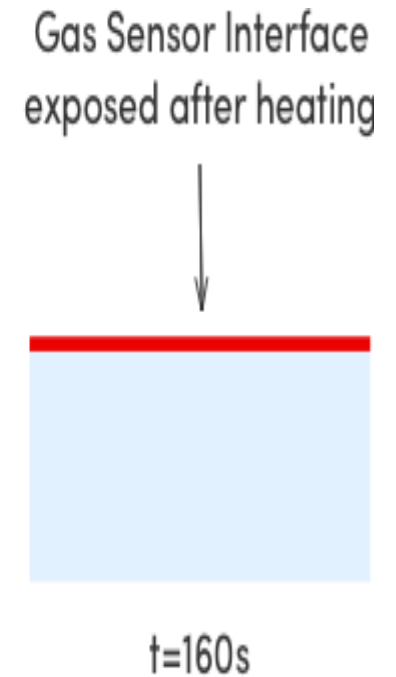
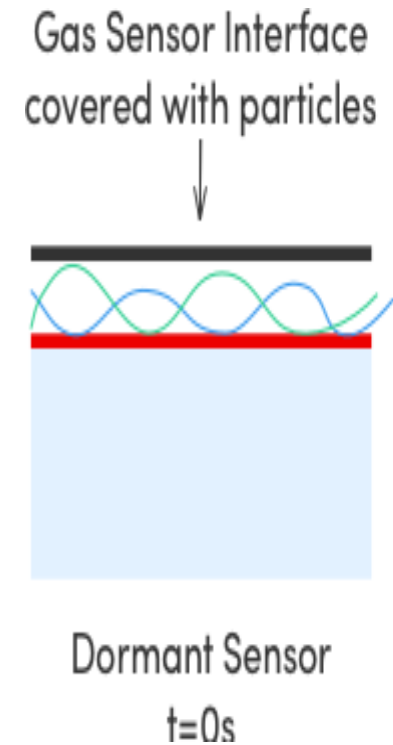


Water level sensors for Home Automation

- While building your prototype you may consider a solid state eTape liquid level sensor, or like others who just use an HC-SR04 ultrasonic sensor to measure the water level sensor.
- On the other hand, in other cases where those two don't suffice, one has to utilize something that can deliver a much higher performance.
- Float level sensors and other ICs like LM1830 offers a more precise measurement capability to IoT developers. Although, they are substantially much more expensive than others.

Air composition sensors

- There are a couple of specific sensors that are used by developers to measure specific components in the air:
- CO monitoring by MiCS-5525
- MQ-8 to measure Hydrogen gas levels
- MiCS-2714 to measure nitrogen oxide
- MQ135 to sense hazardous gas levels (NH₃, NO_x, Alcohol, Benzene, smoke, CO₂)
- Most of these sensors have a heating time, which also means that they require a certain time before they actually start delivering accurate values.
- These sensors mainly rely on their surface to detect gas components. When they initially start sensing, there's always something that's there on their surface, some sort of deposition that requires some heating to go away.
- Hence, after the surface gets heated enough true values start to show up.



Video cameras for surveillance and analytics

- A range of webcams and cameras specific to Hardware development kits are usually used in such scenarios. Hardware with USB ports offers to integrate and camera module to build functionalities.
- But, utilizing USB ports is not very efficient, especially in the case of real-time video transfer or any kind of video processing.
- Take RaspberryPi for example, it comes with a camera module (Pi cam) that connects using a flex connector directly to the board without using the USB port. This makes the Pi cam extremely efficient.

Sound detection for Home Automation

- Sound detection plays a vital role from monitoring babies to turning on and off lights automatically to automatically detecting your dog's sound at the door and opening it up for them.
- Some commonly used sensors for sound detection includes SEN-12462 and EasyVR Shield for rapid prototyping.
- These sensors aren't as good as industrial grade sensors like those from [3DSignals](#) which can detect even ultra-low levels of noise and fine tune between various noise levels to build even machine break up patterns.

Humidity sensors for Home Automation

- These sensors bring the capability of sensing humidity/RH levels in air for smart homes. The accuracy and sensing precision depends a lot on multiple factors including the overall sensor design and placement.
- But certain sensors like DHT22 and 11 built for rapid prototyping would always perform poorly when compared to high-quality sensors like HIH6100 and Dig RH.
- While building a product to sense humidity levels, ensure that there's no localized layer of humidity that is obscuring the actual results. Also, keep into consideration that in certain small spaces, the humidity might be too high at one end as compared to the others.
- When you look at free and open spaces where the air components can move much freely, the distribution around the sensor can be expected to be uniform and subsequently would require very less amount of corrective actions for the right calibration.

Home Automation Protocols

- One of the most important parts of building a home automation product is to think about protocols, protocols that your device would use to communicate to gateways, servers, and sensors. A few years ago, the only way to do so was by either using Bluetooth, wifi or GSM. But due to added expenses on cellular sim cards, and low performance of Wifi, most such solutions didn't work.
- A few years ago, the only way to do so was by either using Bluetooth, wifi or GSM. But due to added expenses on cellular sim cards, and low performance of Wifi, most such solutions didn't work.
- Bluetooth survived and later evolved as Bluetooth Smart or Bluetooth low energy. This helped bring a lot of connectivity in the “mobile server powered economy”, in this essentially your phone would act as a middleware to fetch data from BLE powered sensors and sent it over to the internet.

Home Automation Protocols

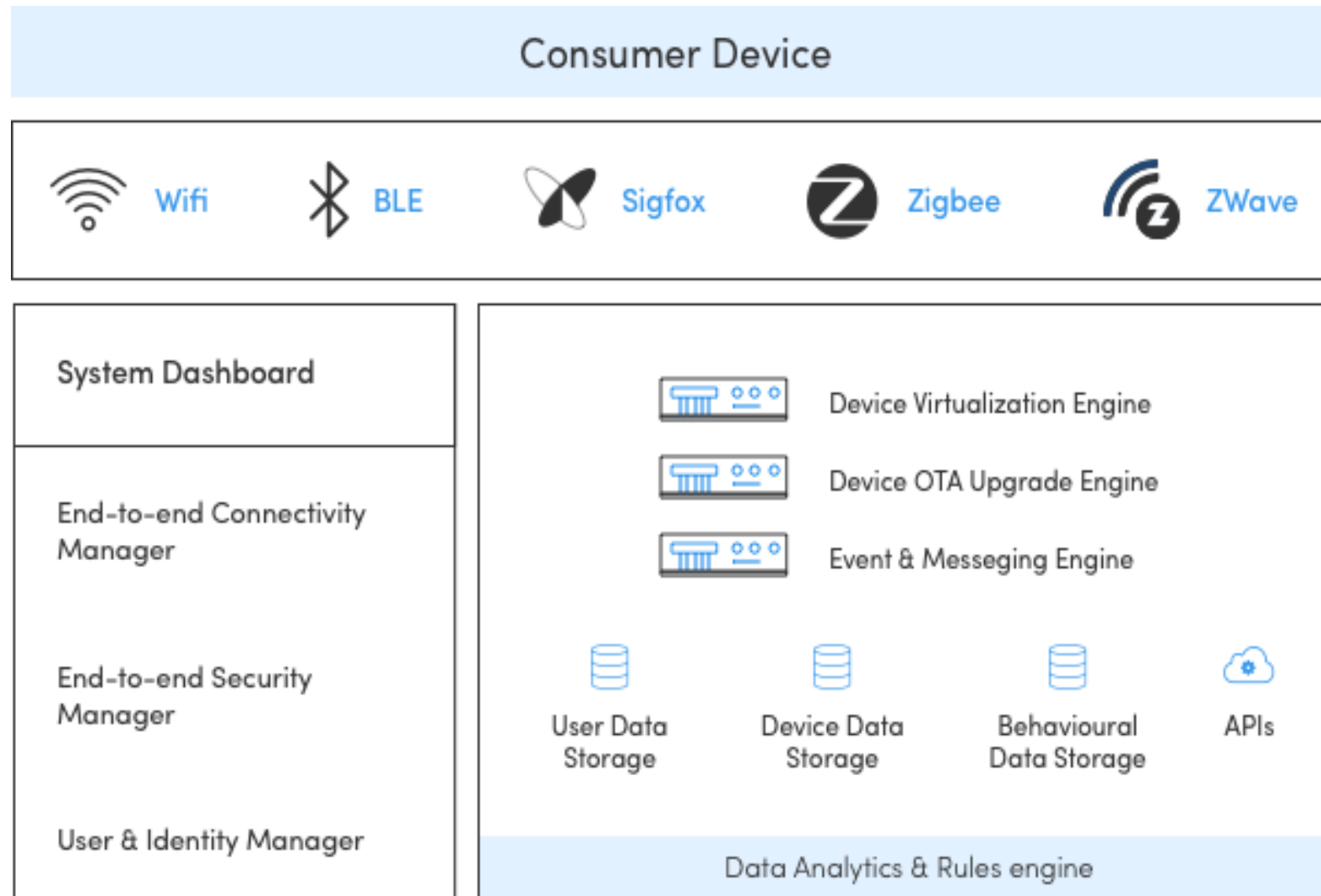
- When looking at the major home automation protocols, the following tops the list:
- Bluetooth low energy or Bluetooth Smart: Wireless protocol with mesh capabilities, security, data encryption algorithms and much more. Ideal for IoT-based products for smart homes.
- Zigbee: Low cost, mesh networked and low power radio frequency based protocol for IoT. Different Zigbee versions don't talk to each other.
- X10: A legacy protocol that utilizes powerline wiring for signaling and control
- Insteon: Communicates with devices both wirelessly and with wires
- Z-wave: Specializes in home automation with an emphasis on security
- Wifi: Needs no explanation
- UPB: Uses existing power lines installed in a home, reduces costs
- Thread: A royalty-free protocol for smart home automation, uses a 6lowpan
- ANT: An ultra low power protocol helping developers build low-powered sensors with a mesh distribution capabilities.
- 6lowpan

Home Automation Protocols-

Home Automation: Which protocol is the best?

- While there are some protocols that clearly offer much more than others, but it is always important to start from your smart home development needs and then move towards narrowing down the solutions.
- The commonly preferred protocols are Bluetooth low energy, Z-wave, Zigbee, and Thread. The protocol selection can now be narrowed down by the following factors:
 - Ability to perform identity verification
 - Quality of sensor networks
 - Data transfer rate
 - Security level
 - Network topology required
 - Density of objects around
 - Effective Distance to be covered

Home Automation Architecture



Home Automation Architecture

- This architecture supports the following considerations for home automation solutions:
- End to end security mechanisms involving multilevel authentication
- End to end data encryption, including the link layer
- Flexible and configurable access and authorization control
- Powerful cloud infrastructure
- Network agnostic with built-in feedback loops
- Configurable cloud-based rules engine
- API endpoints
- Data scalability
- NoSQL databases

Home Automation Gateways

- For developing a home automation product, often stand-alone product sending data to a server is not enough. Often due to battery and protocol limitations, the data from a sensor or sensors present in a home has been routed through an IoT gateway.
- To select the perfect gateway for your IoT home automation, consider some of the factors including:
 - Communication protocols supported
 - Real-time capabilities
 - MQTT, CoAP, HTTPS support
 - Security and configuration
 - Modularity

Home Automation Gateways

- When it comes to building IoT gateways, modularity and hybrid IoT protocol support top that list when a product is in the early stages of market introduction.
- To incorporate a gateway in your home automation stack you can consider the following options:
- Either create a Gateway from the ground up using existing hardware stacks for prototyping(using Raspberry Pi, Intel Edison, etc). Then when a PoC is validated you can create your own custom hardware.
- Or, you can use existing gateway modules like [Ingincs BLE gateway](#). These gateways are extremely easy to customize and connect with your cloud services and devices. However, they may or may not offer the same level of support that you need to build certain features.
- For example, a gateway with a bad networking queue may result in traffic congestion, or it may not support the required protocols that you wish to use.
- Further, pivoting with these gateways to some other technology stack may become very difficult. It should have been emphasized that they are extremely good for robust prototyping needs.

Home automation programming language for smart home developers

- The following programming languages dominated the home automation space:
- Python,
- Embedded C, C,
- Shell, Go,
- Javascript (node.js).
- This has mainly happened due to the sheer optimization of the languages for similar use cases.

Home Automation frameworks









- If you think you can build everything from home automation (protocols, hardware, software, etc) on your own, it is a bit unrealistic. Everyone starting from high growth startups to billion dollar consumer focused enterprises are now taking the help of home automation frameworks to build connected products to delight consumers.
- Everyone starting from high growth startups to billion dollar consumer focused enterprises are now taking the help of home automation frameworks to build connected products to delight consumers.
- There are more than 15 different smart home frameworks available for IoT developers to use and build their next generation of connected home products.
- Some of these frameworks are open source and some are closed-source. Let's have a look at some of them in the sections that follows.

Home Automation Framework

- **Open source IoT platforms and frameworks for Home Automation**
- Looking forward to doing a quick and dirty prototype? There's no need to write down everything from scratch. Thanks to a bunch of awesome contributions by people like we have open source platforms that can get your home automation products up and running in no time.
 - Home Assistant
 - Calaos
 - Domoticz
 - OpenHAB: Supports Raspberry Pi, written in Java and has design tools to build your own mobile apps by tweaking UI.
 - OpenMotics[Asked their developer, waiting for them to respond(dev confirmed)]
 - LinuxMCE
 - PiDome
 - MisterHouse
 - Smarthomatic

Home Assistant for smart home development:

- Supports RaspberryPi, uses Python with OS as Hassbian. It has simplified automation rules that developers can use to build their home automation product saving them thousands of lines of code.
- Home Assistant supports the following:

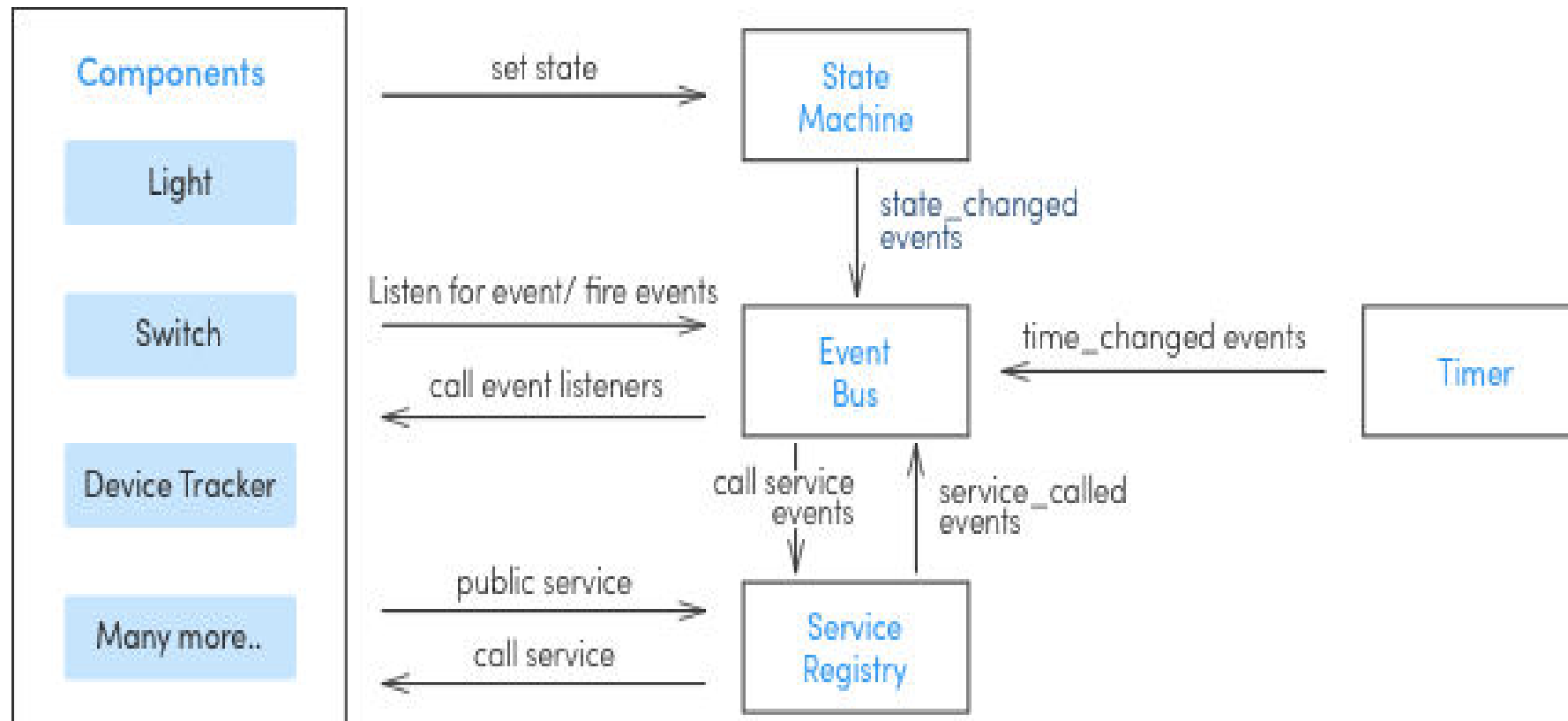
				
				
				
				

Home Assistant for smart home development:

- **How home assistant works involve the following:**
- Home control responsible for collecting information and storing devices
- Home automation triggers commands based on user configurations
- Smart home triggers based on past user behavior
- As developers, it is very important for us to understand the architecture of Home Assistant for us to build high-performing products on top of it.
- Let's have a look at Home control's architecture that makes control and information flow possible.
- **Home control consists of five components:**
- Components
- State machine
- Event bus
- Service registry
- Timer

Home Assistant for smart home development

- The core architecture of Home Assistant



Home Assistant for smart home development

- All of these components working together create a seamless asynchronous system for smart home IoT. In the earlier version of Home Assistant core, the core often had to stop while looking for new device information.
- But, with the new versions of home assistant, a backward compatible API, and ansyn core have been introduced making things a lot faster for IoT applications.
- The best part about home assistant's core architecture is how carefully it has been designed and developed to support IoT at home.

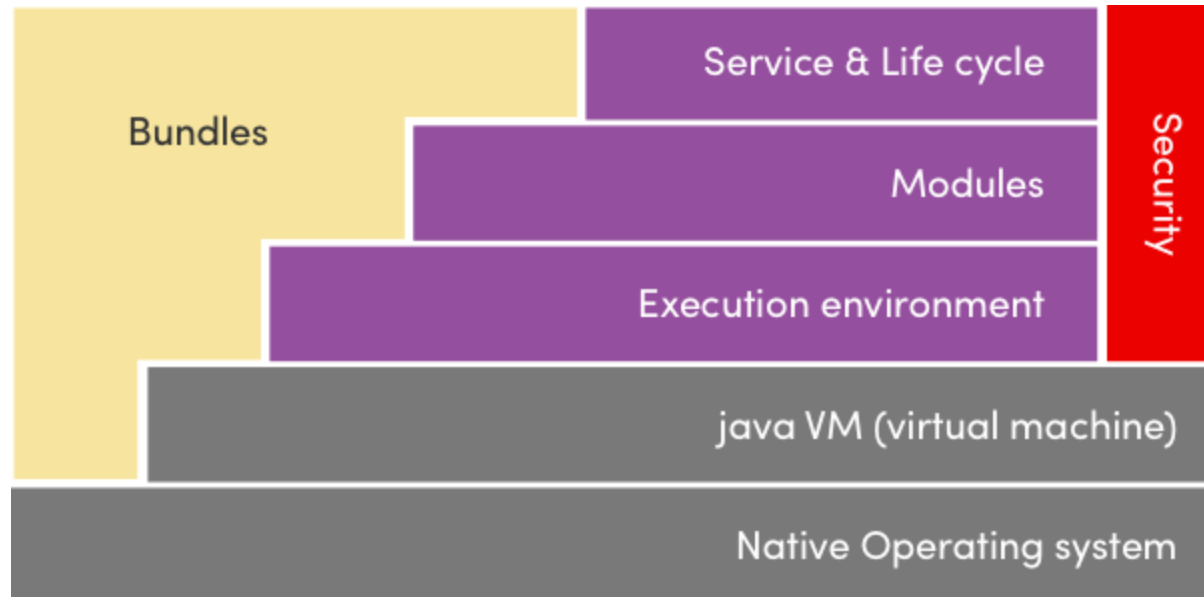
Home Assistant for smart home development

- **OpenHAB for Smart home automation**
- OpenHAB is a home automation and IoT gateway framework for smart homes. Similar to Home Assistant, OpenHAB works nicely with Raspberry Pi and comes with their own design tools to create a UI for your home automation product.
- An understanding architecture of OpenHAB:
- Modularity: It is realized with the bundle concept
- Runtime dynamics: so that software components can be managed at the runtime
- Service orientation: there are services for various components to speak with each other and exchange information

OpenHAB for Smart home automation

- Further relying on the OGSi framework, it leverages the following layers stacked together:
- Modular layer: Manages dependencies between bundles
- Life cycle layer: controls the life cycle of the bundles
- Service layers: defines a dynamic model of communication between various modules
- Actual services: this is the application layer, using all other layers
- Security layer: optional, leverages Java 2 security architecture and manages permissions from different modules

OpenHAB Architecture



OpenHAB features

- Plugin framework
- Rules engine
- Logging mechanism
- UI abstraction: A tree structure for UI Widgets, Item UI providers, and dynamic UI configuration
- UI implementations are available for the web, Android, and iOS
- Designer tools availability
- OpenHAB has been primarily only been observed as a project for the hobbyist programmer, even many parts of openhab.org convey the same. But, we have observed a different effort in the recent times from OpenHAB into building the developer economy for building IoT smart homes.

How Does a IoT based Home Automation System Work?

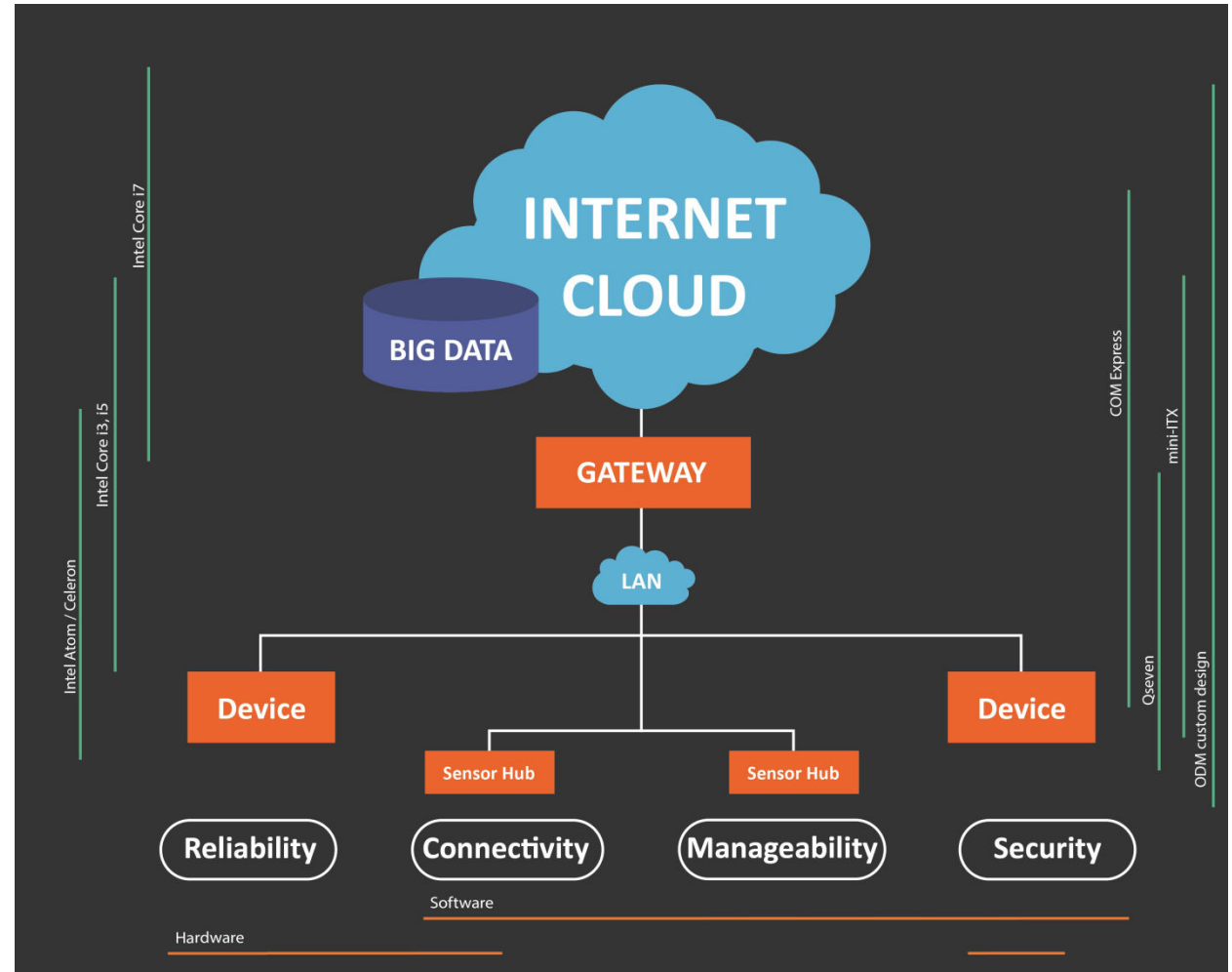
- The concept of Home Automation aims to bring the control of operating your every day home electrical appliances to the tip of your finger, thus giving user affordable lighting solutions, better energy conservation with optimum use of energy. Apart from just lighting solutions, the concept also further extends to have a overall control over your home security as well as build a centralised home entertainment system and much more. The **Internet of Things** (or commonly referred to as IoT) based Home Automation system, as the name suggests aims to control all the devices of your smart home through internet protocols or cloud based computing.
- The IoT based Home Automation system offer a lot of flexibility over the wired systems s it comes with various advantages like ease-of-use, ease-of-installation, avoid complexity of running through wires or loose electrical connections, easy fault detection and triggering and above and all it even offers easy mobility.

Home Automation System using IoT



Basic Setup-Home Automation

- Thus IoT based Home Automation system consist of a servers and sensors. These servers are remote servers located on Internet which help you to manage and process the data without the need of personalised computers.
- The internet based servers can be configured to control and monitor multiple sensors installed at the desired location.



Home Automation-Working Process

- **Controller: The Brain of Your System**
- The main controller or the hub is the most essential part of your Home Automation system irrespective of whether you connect single or multiple sensors in your home. The main controller or the hub is also referred to as gateway and is connected to your home router through the Ethernet cable. All the IoT based sensors transmits or receive commands through the centralised hub. The hub in turn receives the input or communicates the output to cloud network located over the internet.
- Due to this kind of architecture, it is possible to communicate with the centralised hub even from remote and distant locations through your smartphone. All you need is just a reliable internet connection at the hub location and the data package to your smartphone that helps you connect to the cloud network.
- Most of the smart home controllers available in the market from several manufacturers cater to all three widely used protocols of wireless communication for Home Automation: **ZigBee, Z-Wave and Wi-Fi.**

Home Automation-Working Process

- **Smart Devices: The Sensory Organs of Your Home**
- The IoT based home automation consist of several smart devices for different applications of lighting, security, home entertainment etc. All these devices are integrated over a common network established by gateway and connected in a mesh network. This means that it gives users the flexibility to operate one sensor based followed by the action of the other. For e.g. you can schedule to trigger the living room lights as soon as the door/windows sensor of your main door triggers after 7pm in the evening.
- Thus all the sensors within a common network can perform cross-talk via the main controller unit. As shown in the figure, some of the smart sensors in home automation acts as sensor hubs. These are basically the signal repeaters of signal bouncers which that are located in the midway between the hub installation location and the sensors that are at a distant location. For such long distances, these sensor hubs play an important role to allow easy transmission of signals to sensors that are far away from the main controller but in closer proximity to the sensor hub. The commonly used sensor hubs in IoT based Home Automation system are Smart Plugs.

Home Automation-Working Process

- **Wireless Connectivity: How the Internal Communication Occurs**
- Most of the IoT based Home Automation systems available today work on three widely used wireless communication protocols : Wi-Fi, ZigBee and Z-Wave
- The ZigBee and the Z-Wave controllers are assigned a network ID which is distributed over other sensors in the network. The communication amongst devices take place in a mesh topology where there is no fixed path for the signals transmitted from the controller to the sensors and vice versa. Depending on the availability of the shortest path the signal from the controller will travel to the target sensors either directly or through signal hops. If any intermediate sensor in the pathway is busy or occupied the signal will trace another path within the mesh network to reach the final destination. Note that sensors with different Network IDs cannot communicate with each other over common channel.

Home Automation-Working Process

- **Connected with the Cloud: Access Everything on the Go**
- The Cloud-based-Networking system involves storage and maintenance of data over the Internet location. This gives users the flexibility to have access to the data from any location on the planet.
- As a result of this, in IoT based Home Automation systems users over the cloud network can send commands to the hub even from a distant or remote location. The hub will further send the signal for the intended sensors to trigger and perform the user-requested action. Once the action is performed, the hub will update the status of the action taken to the cloud network and in this way users can control and monitor every aspect of their smart homes.

Home Automation-Working Process

- **Events and Notifications: Get Notified Instantly**
- Real-time monitoring and notifications is one of the key features of IoT based Home Automation systems. Since the hub is connected over the cloud network through the Internet, you can schedule various events as per your routine activities or daily schedules. The cloud network can receive and store all the user inputs and transfer them to the hub as per the scheduled events.
- Once the hub transfer the desired signals to the target sensor and the desired action takes places, it will quickly upload the new status over the cloud notifying user instantaneously. For e.g. the motion sensor will instantaneously notify the user wither through emails, SMS, calls or App notifications when it detects any unwanted motion or intrusion. After receiving such notification, the user can quickly turn on the IP based home security smart camera can check the status of your home even from remote location.

Home Automation-Working Process

- **IFTTT Integration: Put Internet to Work for You**
- It is not practically possible to trigger every action one by one in your day long busy schedule. This is where you can put the Internet to work for you. The IF This Then That (IFTTT) Integration helps you in this condition.
- This enables you to create cascading effect of actions where the target action will trigger only when the IF condition is satisfied. Some of the examples of IFTTT triggers are like “IF” day temperature above 25 degrees, turn the ACs on and roll-down the curtain blinds. IF Movie Mode is ON, then turn the lights to 10% brightness, IF soil moisture less than specific values, turn the water sprinklers in the garden ON.
- There are endless possibilities that you can create with IFTTT triggers and thus make the optimum use of your Home Automation system thereby making optimum use of energy and simultaneously enjoying a comfortable lifestyle.

Thank you

CSPC702- EMBEDDED SYSTEMS AND INTERNET OF THINGS (IOT)

UNIT - IV Network and communication aspects

Syllabus

- **UNIT - II Network Challenges and applications of IoT**
- Network and communication aspects: Wireless medium access issues, MAC protocol survey, Survey routing protocols, Sensor deployment & Node discovery, Data aggregation & dissemination- Design challenges- Development challenges-Security challenges. Applications of IoT: Home automation, **Industry applications, Surveillance applications, Other IoT applications.**

Applications of IoT: Industry applications

- The Following are some of the industrial applications in the current days.
- **Automated and remote equipment management and monitoring**
- **Predictive maintenance**
- **Faster implementation of improvements**
- **Pinpoint inventories**
- **Quality control**
- **Supply chain optimization**
- **Plant safety improvement**

Applications of IoT: Industry applications

- **Automated and remote equipment management and monitoring**
- One of the main IIoT applications is related to the automated management of equipment, allowing a centralized system to control and monitor all company processes.
- This ability to **remotely control** equipment via digital machines and software also implies that it is possible to control several plants located at different geographic locations.
- This gives companies an unprecedented ability to oversee advances in their production in real time, while also being able to analyze historical data that they obtain in relation to their processes. The objective of collecting and using that data is to support the improvement of processes and generating an environment where **information-based decisions** are a priority.

Applications of IoT: Industry applications

- **Predictive maintenance**
- Predictive maintenance consists of detecting the need for a machine to be **maintained before a crisis** takes place and production needs to be stopped urgently. It is therefore among the reasons to implement a data acquisition, analysis and management system.
- This system is one of the most effective Industrial IOT applications and works via sensors that, once installed on the machines and operating platforms, can **send alerts** when certain risk factors emerge. For example, the sensors that monitor robots or machines submit data to the platforms, which analyze the data received in real time and apply advanced algorithms that can issue warnings regarding high temperatures or vibrations that exceed normal parameters.

Applications of IoT: Industry applications

- **Faster implementation of improvements**
- IIoT generates valuable information so that those in charge of improving processes in an **industrial business model** (process, quality or manufacturing engineers) can access data and analyze it faster and automatically, and remotely perform the necessary processes adjustments. This also increases the speed in which changes and improvements are applied in [Operational Intelligence](#) and Business Intelligence – changes that are already offering competitive advantages to a myriad of industrial businesses.

Applications of IoT: Industry applications

- **Pinpoint inventories**
- The use of Industrial IoT systems allows for the automated **monitoring of inventory**, certifying whether plans are followed and issuing an alert in case of deviations. It is yet another essential **Industrial IOT application** to maintain a **constant and efficient workflow**.

Applications of IoT: Industry applications

- **Quality control**
- Another entry among the most important **IIoT applications** is the ability to monitor the **quality** of manufactured products at any stage: from the raw materials that are used in the process, to the way in which they are transported (via smart tracking applications), to the reactions of the end customer once the product is received.
- This information is vital when studying the efficiency of the company and applying the necessary changes in case **failures are detected**, with the purpose of optimizing the processes and promptly detect issues in the production chain. It has also been proven that it is essential to **prevent risks** in more delicate industries, such as pharmaceuticals or food.

Applications of IoT: Industry applications

- **Supply chain optimization**
- Among the Industrial IoT applications aimed at achieving a higher efficiency, we can find the ability to have **real time in-transit information** regarding the status of a company's supply chain.
- This allows for the detection of various hidden **opportunities for improvement** or pinpointing the issues that are hindering processes, making them inefficient or unprofitable.

Applications of IoT: Industry applications

- **Plant safety improvement**
- Machines that are part of IIoT can generate **real-time data** regarding the situation on the plant. Through the monitoring of equipment damages, plant air quality and the frequency of illnesses in a company, among other indicators, it is possible to **avoid hazardous scenarios** that imply a threat to the workers.
- This not only boosts safety in the facility, but also **productivity and employee motivation**. In addition, economic and reputation costs that result from poor management of company safety are minimized.

Applications of IoT: Surveillance applications

- **Five Useful Video Analytics Applications For IoT Surveillance**

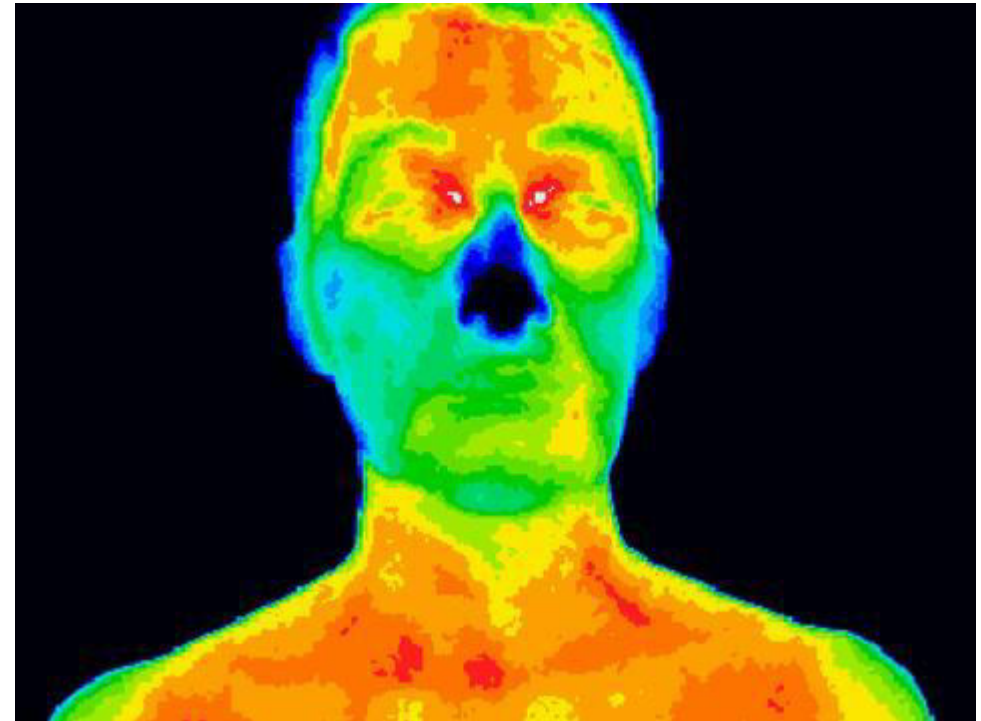
- 1. Facial Recognition**
- 2. Medical Intervention**
- 3. License Plate Recognition**
- 4. Behavioral Prediction**
- 5. Event Detection**

Applications of IoT: Surveillance applications

- **Facial Recognition**
- High performance GPUs use sequential and parallel compute for biometric identification deployments. The leading application for biometric processing — facial recognition — processes various features, textures and shapes of a subject's face for immediate cross-referencing with a secure database. AI and machine learning improve facial recognition speed and accuracy, far exceeding the capabilities of human monitoring by validating with a deep neural network for machine intelligence.
- A rugged NVR computer performing [facial recognition processes at an unmanned airport security touchpoint](#) can provide a finer screening with a highly accurate person-of-interest identification rate. Over time, system capabilities improve with the ability to identify subjects at greater distances with more data and information.
- Thermal cameras used for facial recognition can form additional layers of security at identification touchpoints. Long-wave thermal infrared (LWIR) cameras capture temperature data from subjects, providing accurate images of head and face shapes that may visually be obscured by glasses, facial hair or deceptive prosthetics, even in low-light environments. Rugged NVR computers processing a combination of images from standard visual and LWIR cameras offer stronger security than either input alone.

Applications of IoT: Surveillance applications

Facial recognition can also be [applied to customer service](#) in many applications. Management may be alerted when priority customers or exclusive club members enter the premises, allowing them to deliver personal attention to the patrons. Acute facial recognition identifying signs of agitation in customer waiting areas may alert management to redirect resources to alleviate congestion and reduce wait times. Retail establishments, casinos and large venues can all leverage facial recognition to strengthen customer loyalty through proactive attention.



Applications of IoT: Surveillance applications

- **Medical Intervention**
- Rugged NVR computers applying much of the same facial recognition technology used at security touchpoints can also leverage the applications toward [medical intervention](#). Visual and LWIR cameras can supply the rugged NVR computer with image data to be analyzed for subtle physical signs of illness or elevated temperatures.
- Rugged NVRs can concurrently process the same image data from camera systems positioned for security monitoring to passively screen subjects for illness symptoms. By identifying subjects for additional health screening prior to accessing densely populated areas like public centers, transportation hubs or large venues, rugged NVR computers provide a powerful tool in the fight to contain widespread illnesses.

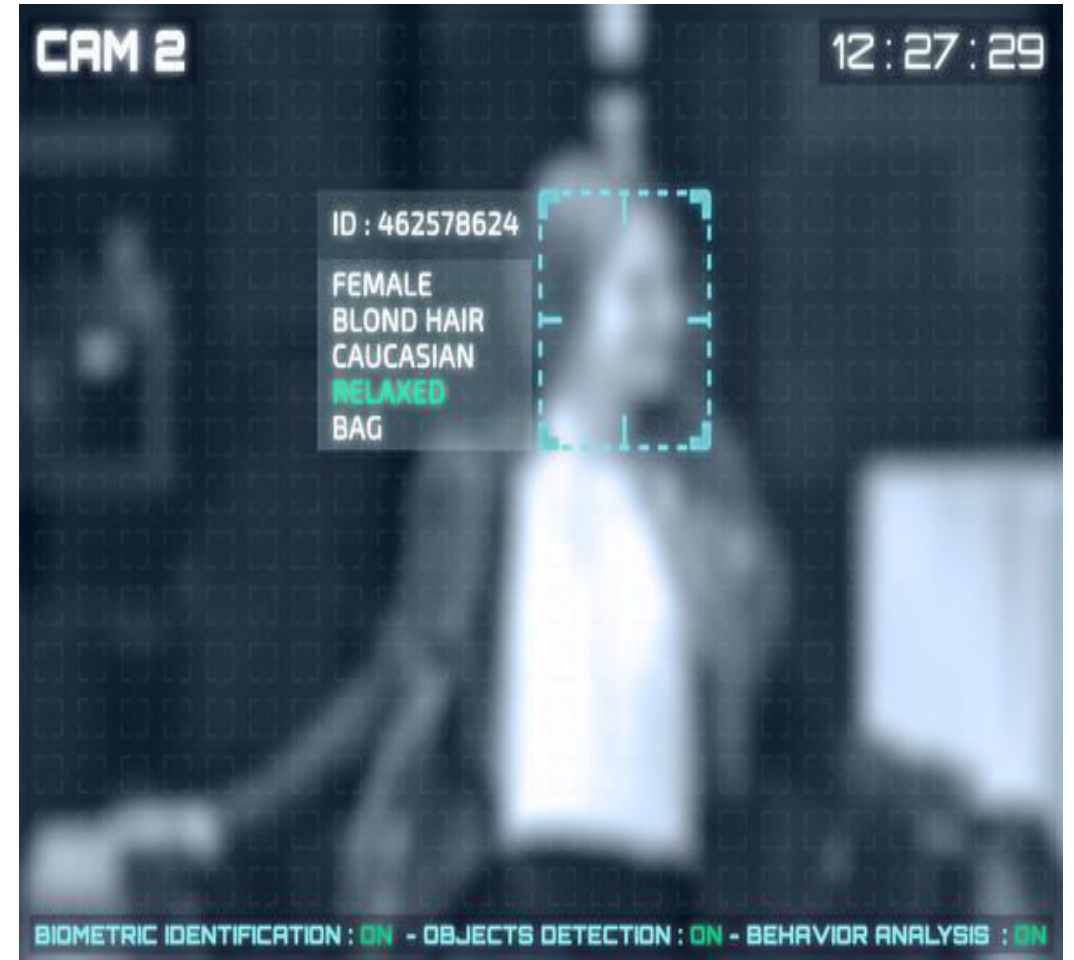


Applications of IoT: Surveillance applications

- **License Plate Recognition**
- High-resolution cameras deployed at facility parking locations and along roadways can instantly scan volumes of license plates to locate and track specific vehicles or maintain a record of vehicles entering or leaving an area. Rugged NVR computers running optical character reading algorithms accurately capture and process license plate digits automatically when the predetermined parameter is triggered, like a vehicle entering the camera field of view or a specific point within. The same rugged NVR computers are also integrated in first responder vehicles like police cars to process and store live camera footage. Police cars use these type of embedded computers as a control hub for cameras around the vehicles to wireless connected body cameras located on the police officer.
- Business facilities can deploy license plate recognition to automatically grant gate access to vehicles displaying a plate registered in an approved visitor database. In ungated areas, the system can passively maintain a record of all vehicles that enter or depart a facility, noting dates and times, for accountability purposes.

Applications of IoT: Surveillance applications

- K-12 educational facilities can use license plate recognition analytics to better safeguard students by logging vehicles entering the grounds and which students enter the vehicles, securing investigative leads in the event of abduction or missing person situation. For higher education, license plate recognition is a sound tool for parking enforcement when integrated with lot camera systems.
- In addition to improving security, smart cities deploying license plate recognition camera systems can factor traffic patterns based on vehicle volume and repeat commutes. Traffic signals can be optimized on license plate data to correspond with anticipated traffic increases based on regularity and concurrence of commutes.



Applications of IoT: Surveillance applications

- **Behavioral Prediction**

- The rugged NVR computer GPUs can power behavioral prediction analytics that allow for quicker response times. The ability to anticipate how other parties may act can be directed toward avoiding unfortunate or harmful scenarios, optimizing processes and personalizing customer experience.
- In-vehicle rugged NVR computers can alert truckers to signs that nearby vehicles may cut in front of them, brake suddenly or create any other dangerous situation requiring immediate attention. Machine learning refines traffic inference analysis, improving system response times. Behavioral predictive algorithms identify signs of distracted or impaired drivers on the road sooner, gradually allowing truck drivers more warning.
- In-cab cameras capturing driver behavior and rugged NVR computers implementing a robust mix of analytics can even collate vehicle telematics like speed, positioning and braking with images of the person behind the wheel to note signs of dangerous driving in the truck drivers themselves. Algorithms used to identify facial features can easily detect signs of drowsy or sleeping drivers, prompting an alert to rouse the driver and direct them to take corrective action.

Applications of IoT: Surveillance applications

- Behavioral predictive analytics have myriad security applications. Public transport, smart cities and venues can deploy rugged NVR computers in any surveillance environment to single out persons displaying very subtle signs of disruptive behavior. Airports applying behavioral predictive surveillance can identify traveler aggression that might be quickly de-escalated through prompt security intervention.



Applications of IoT: Surveillance applications

- **Event Detection**

- Rugged NVR computers deployed with event detection analytics use many of the same algorithms for the applications listed above. Event detection can alert smart city operators to numerous conditions that may lead to hazards for citizens, like unattended packages in public spaces, fallen items in roadways and weather-related complications.
- The same preventative insights smart cities glean from event detection can be applied in virtually any environment where specific programmed visual triggers alert operators to changes from normalcy. A person or object falling onto tracks at subway and rail terminals may alert traffic coordinators and response teams to redirect or suspend travel until the situation has been resolved by responding security teams.
- Schools, public buildings, large venues and shopping centers can all benefit greatly leveraging object-motion detection analytics to immediately identify firearms in the setting. Hypersensitive gun detection algorithms instantly alert security or law enforcement personnel upon the event of a weapon being drawn. This event detection analytic transforms a chain notification process relying on persons under stress to actively initiate a security process to a passive, real-time alert when a weapon is presented. By shaving valuable minutes or seconds off police response time, gun detection technology enables quicker intervention that can prevent disasters.

Other IoT applications

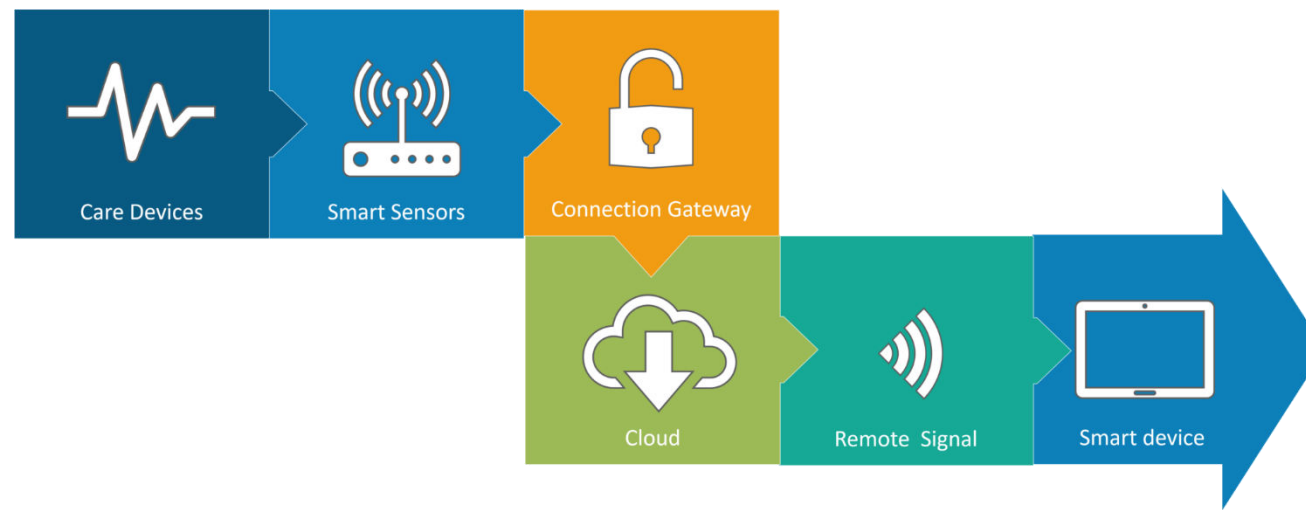
- **IoT Applications – Wearables**
- Wearable technology is a hallmark of IoT applications and probably is one of the earliest industries to have deployed the IoT at its service. We happen to see Fit Bits, heart rate monitors and smartwatches everywhere these days.
- One of the lesser-known wearables includes the Guardian glucose monitoring device. The device is developed to aid people suffering from diabetes. It detects glucose levels in the body, using a tiny electrode called glucose sensor placed under the skin and relays the information via Radio Frequency to a monitoring device.



Other IoT applications

- **IoT Applications – Health Care**

- IoT applications can turn reactive medical-based systems into proactive wellness-based systems.
- The resources that current medical research uses, lack critical real-world information. It mostly uses leftover data, controlled environments, and volunteers for medical examination. IoT opens ways to a sea of valuable data through analysis, real-time field data, and testing.
- The Internet of Things also improves the current devices in power, precision, and availability. IoT focuses on creating systems rather than just equipment.



Other IoT applications

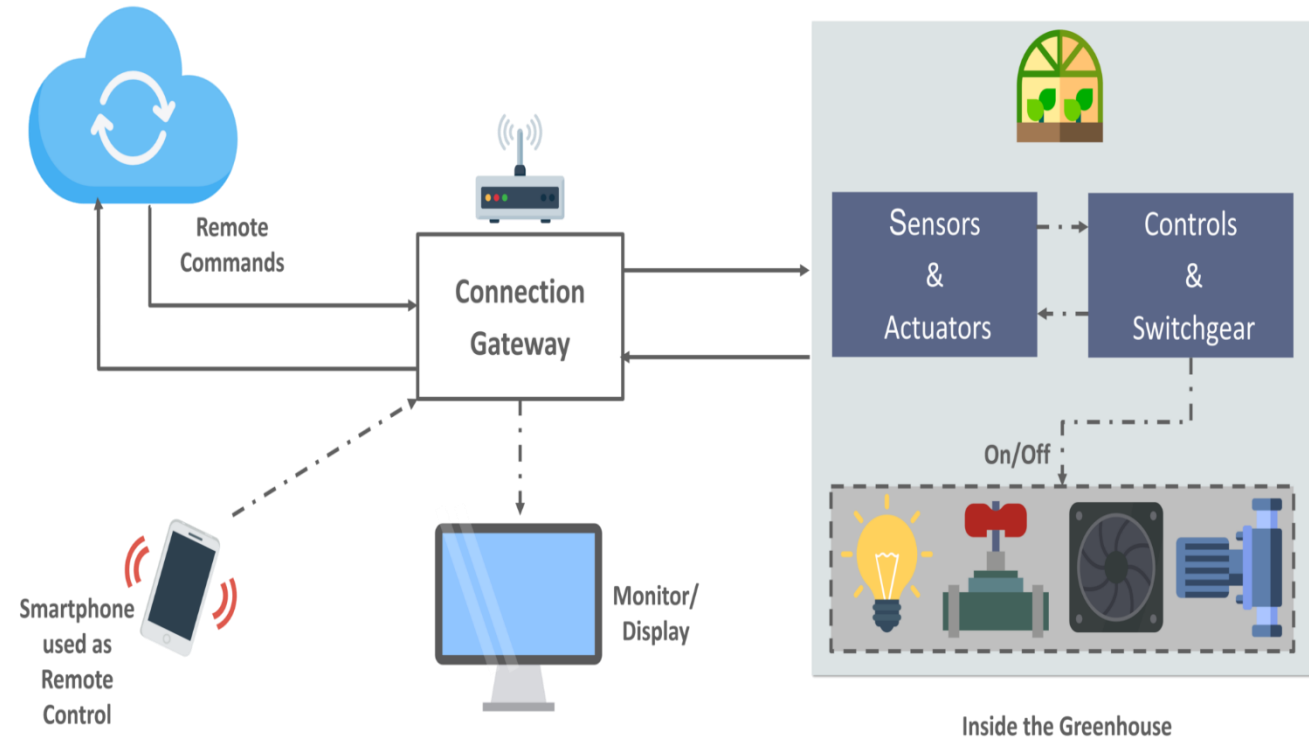
- **IoT Applications – Smart Cities**

- By now I assume, most of you must have heard about the term **Smart City**. The hypothesis of the optimized traffic system I mentioned earlier, is one of the many aspects that constitute a smart city.
- The thing about the smart city concept is that it's very specific to a city. The problems faced in Mumbai are very different than those in Delhi. The problems in Hong Kong are different from New York. Even global issues, like finite clean drinking water, deteriorating air quality and increasing urban density, occur in different intensities across cities. Hence, they affect each city differently.
- The Government and engineers can use IoT to analyze the often-complex factors of town planning specific to each city. The use of IoT applications can aid in areas like water management, waste control, and emergencies.

Other IoT applications

- **IoT Applications – Agriculture**

- Statistics estimate the ever-growing world population to reach nearly 10 billion by the year 2050. To feed such a massive population one needs to marry agriculture to technology and obtain best results. There are numerous possibilities in this field. One of them is the **Smart Greenhouse**.
- A greenhouse farming technique enhances the yield of crops by *controlling environmental parameters*. However, manual handling results in production loss, energy loss, and labor cost, making the process less effective.
- A greenhouse with embedded devices not only makes it easier to be monitored but also, enables us to control the climate inside it. Sensors measure different parameters according to the plant requirement and send it to the cloud. It, then, processes the data and applies a control action.



Other IoT applications

- **Self-driven Cars**

- We've seen a lot about self-driven cars. Google tried it out, Tesla tested it, and even Uber came up with a version of self-driven cars that it later shelved. Since it's human lives on the roads that we're dealing with, we need to ensure the technology has all that it takes to ensure better safety for the passenger and those on the roads.
- The cars use several sensors and embedded systems connected to the Cloud and the internet to keep generating data and sending them to the Cloud for informed decision-making through Machine Learning. Though it will take a few more years for the technology to evolve completely and for countries to amend laws and policies, what we're witnessing right now is one of the best applications of IoT.

Other IoT applications

- Traffic Management
- IoT Retail Shops
- Supply chain management
- Environmental monitoring
- Maintenance management.
- Water supply.
- E-Waste Management
- Smart grid and energy saving



Other IoT applications

Sl.No	Application Domain	Application
1	Smart mobility & smart tourism	Traffic management, multi-modal transport Road condition monitoring, parking system, waste collection Payment systems, tour guide services
2	Public safety & environmental monitoring	Environmental & territorial monitoring Video/radar/satellite surveillance Emergency site/rescue personal tracking, emergency plan
3	Smart Home	Plant maintenance, energy management Video surveillance, access management, children protection Entertainment, comfortable living
4	Smart Grid	Load management, storage service, entertainment services Sustainable mobility, booking charging slot Power generation/distribution/storage, energy management
5	Industrial processing	Real-time vehicle diagnostic, assistance driving Luggage management, boarding operation, mobile tickets Monitoring industrial plants

Other IoT applications

Sl.No	Application Domain	Application
6	Agriculture & breeding	Animal tracking, certification and trade control Irrigation, monitoring agricultural production & feed Farm registration management
7	Logistics & product lifetime management	Identification of materials/product deterioration Waterhouse management, retail, inventory Shopping operation, fast payment
8	Medical & healthcare	Remote monitoring medical parameters, diagnostics Medical equipment tracking, secure indoor environment management, Smart hospital services, entertainment services
9	Independent living	Elderly assistance, disabled assistance Personal home/mobile assistance, social inclusion Individual well-being, personal behavior impact on society

Thank you

CSPC702-Embedded Systems and Internet of Things

Unit V- Raspberry PI with Python and Arduino

Syllabus

- **UNIT - V Raspberry PI with Python and Arduino**
- Building IOT with RASBERRY PI- IoT Systems - Logical Design using Python – IoT Physical Devices & Endpoints - IoT Device -Building blocks -Raspberry Pi - Board - Linux on Raspberry Pi - Raspberry Pi Interfaces -Programming Raspberry Pi with Python - Other IoT Platforms – Arduino - Evolution of IOE and its benefits.

What is Raspberry pi?

- The Raspberry pi is a minimal effort master card measured PC that attachments into a PC screen or TV.
- Utilizations a standard console and mouse. It is a competent little gadget that empowers individuals of any age to investigate registering.

What Is Raspberry pi?

- *A Raspberry Pi is a credit card-sized computer originally designed for education.*
- *It is a motherboard containing different slots like USB port, Audio jack, memory slots, and Ethernet port etc.*
- *It is slower than a modern laptop or desktop but is still a complete Linux computer and can provide all the expected abilities that implies, at a low-power consumption level.*

Price of Raspberry pi is in between 5\$-35\$(350Rs-2500Rs)



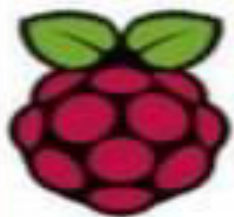
How did the Raspberry pi get its name?

How did the Raspberry Pi get its name

4

[fruit name + python]

A lot of computer companies were named after fruit. There's Tangerine Computer Systems, Apricot Computers, and the old British company Acorn, which is a family of fruit. Pi is because originally we were going to produce a computer that could only really run Python. So the Pi in there is for Python.



Raspberry Pi



History of Raspberry pi

History

5

- ❑ The Raspberry Pi was created in February 2012
- ❑ First generation Raspberry pi 1 (Model B) was launched
- ❑ Developed in the United Kingdom by the Raspberry pi Foundation
- ❑ Feature of Raspberry pi 1 model B

Memory- 512MB RAM

Connections -2 USB ports
- 100mb Ethernet port
-3.5mm jack for audio out
-HDMI

Processor - Broadcom BCM2835

Launched -Raspberry pi

- In February 2012 Raspberry pi 1 model B(Basic)
- In April 2014 Raspberry pi 1 model B+ (Credit card sized)
- Improved A+ and B+ models were released a year later.
- In February 2015 Raspberry pi 2 (added more RAM).
- In November 2015 Raspberry pi Zero (General Purpose Input/Output(GPIO))
- In February 2016 Raspberry pi 3 model B released (on-board wi-fi Bluetooth and USB boot capabilities)
- February 28,2017 the Raspberry pi Zero W(Identical to the Raspberry pi but has the wi-fi and bluetooth functionality)

Raspberry Pi Models



Model B



Model A



Compute Module



Model B+



Model A+



2 Model B



Zero

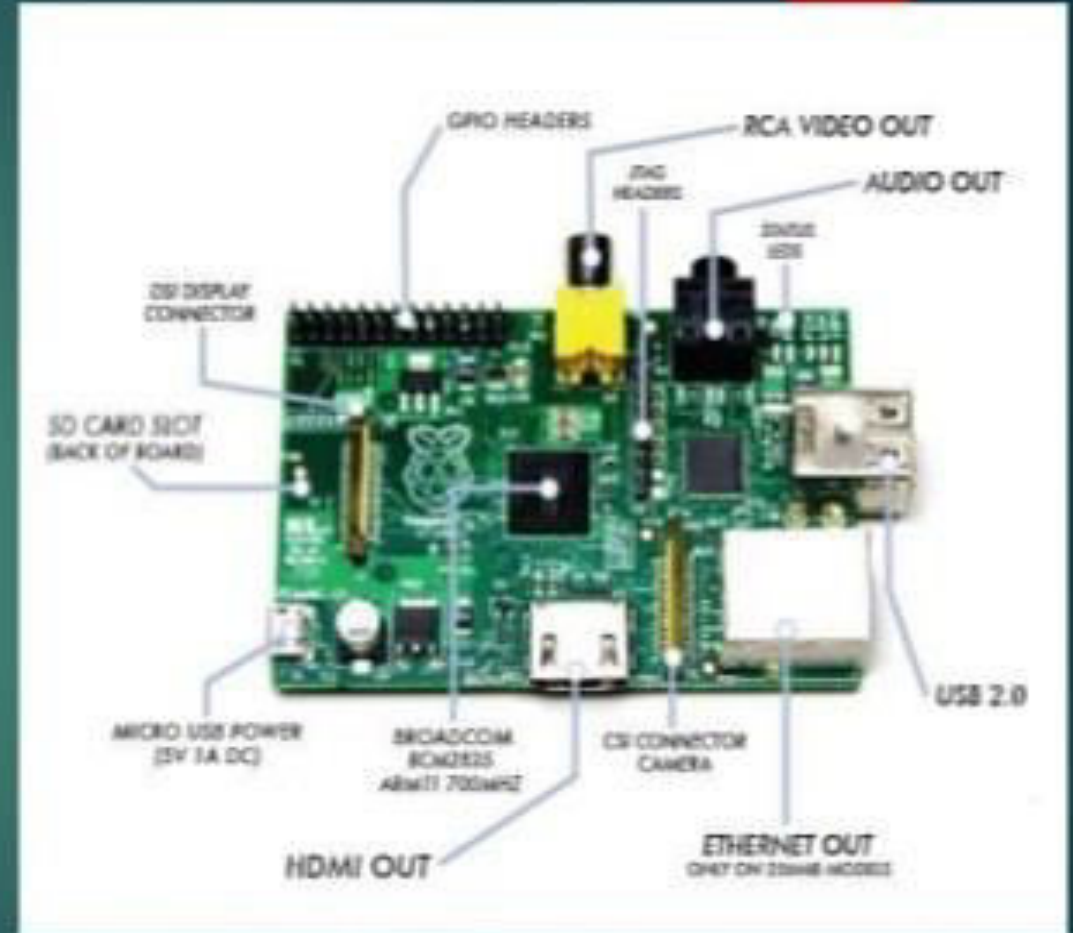


3 Model B

Components of Raspberry Pi

Component Of Raspberry Pi

- USB ports
- Audio Jack(3.5mm)
- HDMI (High-Definition Multimedia Interface)
- POWER Supply
- SD Card slot
- Ethernet
- LED(light emitting diode)
- Wi-Fi/Bluetooth
- CSI(Camera Serial Interface) Connector for camera
- Etc.



How Raspberry Pi Works?

- **An HDMI television or Monitor**
- You will need to connect your Raspberry Pi to a display, which means you will need an HDMI-enabled screen of some kind or HDMI supportable TV and connect it.
- **A USB keyboard and mouse**
- In order to control your Pi, you will need a keyboard and mouse. At this point, pretty much any USB keyboard and mouse will work.
- **An 8 GB micro SD card and card reader**
- 8GB card for this. The Samsung EVO + Class 10 cards like this one are best. If your computer does not have a card. You will want at least an memory card one like this will do.
- **A Power Supply**
- The Raspberry Pi is powered by a micro USB, much like the one you have likely used for your phone. Since the Pi 3 has four USB ports, it's best to use a good power supply that can provide at least 2.5A of current.



How do you use a Raspberry pi?

1.Reformat your microSD card

1.Download NOOBS onto the microSD card

1.Set up your Raspberry Pi

1.Download the Raspbian operating system on the Raspberry Pi

1.Configure your Raspberry Pi

Comparison of Raspberry Pi Versions

	Raspberry Pi	Raspberry Pi 2	Raspberry Pi 3
Released	February 2012	February 2015	February 2016
CPU	ARM1176JZF-S	ARM Cortex-A7	ARM Cortex-A53
CPU Speed	700 MHz Single Core	900 MHz Quad Core	1200 MHz Quad Core
RAM	512 MB 256 MB Rev 1	1GB	1 GB
GPU	Broadcom Video core IV	Broadcom Video core IV	Broadcom Video core IV
Storage	SDHC (Secure Digital High Capacity) Slot	Micro SDHC Slot	Micro SDHC Slot
	Micro SDHC Model A+ and B+		
USB Ports	2 on Model B	4	4
WiFi	No built-in Wifi	No built-in Wifi	802.11n and Bluetooth 4.1

Applications of Raspberry Pi

- **Simple Desktop**
- **Raspberry pi smart mirror**
- **Gaming Device**
- **Robot**
- **CC Tv Camera**

Project Developed with the help of Raspberry pi
Some of them are here!

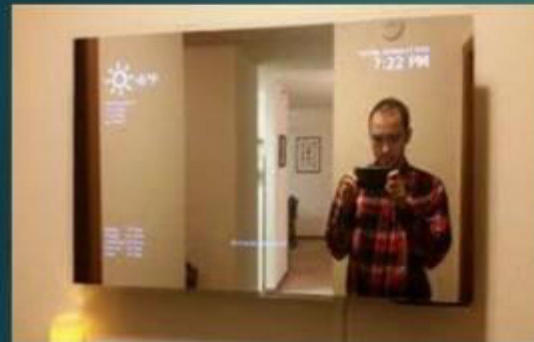
12

1. Simple desktop



2. Raspberry pi Smart Mirror (A mirror Which displays applications, other useful information)

13



Applications of Raspberry Pi

3. Gaming Device

14



4. Robot

15



5. Cc TV Camera

16



Benefit and Limit of Raspberry Pi

- **What is the benefit of Raspberry Pi?**
- The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that **enables people of all ages to explore computing**, and also easy to learn.
- **What are the limits of a Raspberry Pi?**
- The memory of the Raspberry Pi is more limited than you're probably used to, with just **512MB or 256MB available**. You can't expand that with extra memory in the way you can a desktop PC.

Raspberry Pi in IoT

Raspberry pi in IOT

17

Using Raspberry pi we can create various sensor of IOT

Examples

1. IoT DoorBell: This projects shows an automatic door bell which sends an sms or email. Thus the owner can easily monitor from anywhere if someone knock his door.
2. Review on Temperature & Humidity Sensing using IoT: Here is the temperature and humidity sensing using IOT. This is a novel method using raspberry pi scripting.
3. Home Automation Using Internet of Things: Internet of Things (IoT) allows us to implement home automation system that can be controlled remotely through internet. The proposed system can monitor different parameters like gas, light, motion detection, temperature, etc. using the sensor data and also trigger a process according to the requirement. The data from the sensors are uploaded to a cloud server and this data can be used to analyze the parameters.
4. **Water proof sensor**
5. **Etc.**

Advantages of Raspberry Pi

- This microcomputer is useful for small business that run on a lower budget to use their product or to invent new technology that embeds the product. Small business owners can use it to automate any small task, i.e.; such as using the Pi to run a website or use it as a small database and media server.
- The product does not require user to have extensive programming experience since it is aimed for the younger generation to learn about programming. Python, the programming language i.e.; Pi uses, is a smaller amount complex than other languages available. It has better code readability and allows the user to type concepts using fewer number of lines. Python also has a automatic memory management function.
- The product gives a lot of room to experiment and turn it into something else i.e.; entirely different. The SD cards on the board can be easily switched, i.e.; which allows to change the functions of the device without spending a lot of time re-installing the software.
- The Raspberry Pi is perfect for adaptive technology and it is able to display images or play videos i.e.; at high-definition resolution to building systems such as prototyping embedded systems. This product makes it possible to build complex and effective at a cheaper price.
- The product is efficient and i.e.; provides an ethical alternative to small businesses. This small card sized product i.e.; makes it easy to recycle and does not release as much carbon dioxide emissions into the environment, i.e.; unlike big servers that need lots of energy and extensive cooling systems.

Disadvantages of Raspberry Pi

- It does not replace the computer, and the processor is not as fast. It is a time consuming to download and install software i.e.; unable to do any complex multitasking.
- Not compatible with the other operating systems such as Windows.
- This is fit for those who want a gadget that they can tailor to their own needs and tastes, i.e.; not for those that just wants to urge a job done fast. Business owners need to consider the extra hassle if is worth it.
- This product not be useful for bigger business that already have big servers, i.e.; which would already do everything that the Raspberry Pi does. So, it would not be worth and it take time to get to put it together.

Advantages and Disadvantages of Raspberry Pi

- **Advantages of Raspberry Pi**

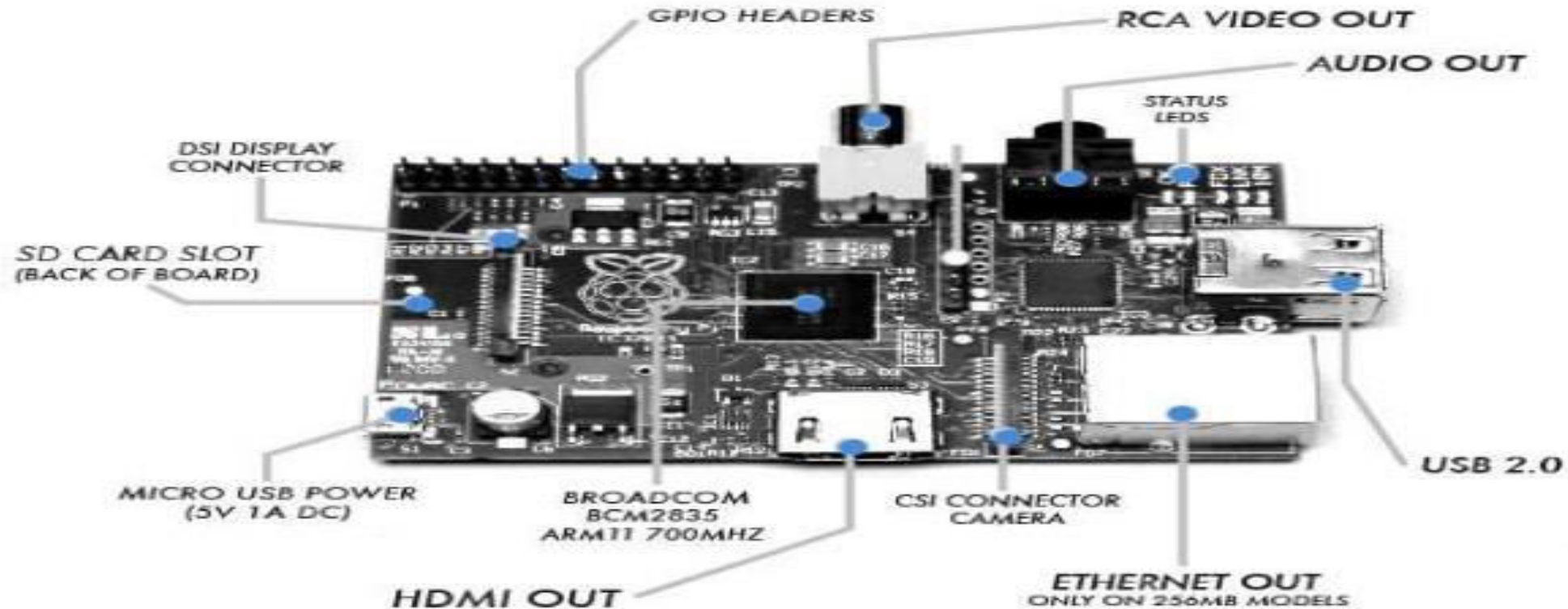
- Small in Size
- Cheap/Low cost
- Low cost server and hosting, ability to handle web traffic
- Open Source
- Can be used as super computer
- User Friendly
- Act as a small web browser
- Energy efficient

- **Disadvantages of Raspberry Pi**

- It cannot run X86 operating system
- Low Processor
- Unable to perform complex multitasking
- Compatibility
- Not useful forv bigger businesses
- Time Consuming

Specification of Raspberry Pi

Specification



Source:

<https://jeffskinnerbox.files.wordpress.com/2012/11/raspberry-pi.jpeg>

IoT Physical Devices & Endpoints

- **What are IoT physical devices and endpoints?**
- An Endpoint, from an IoT perspective, is a **physical computing device that performs a function or task as a part of an Internet connected product or service**. An Endpoint, for example, could be a wearable fitness device, an industrial control system, an automotive telematics unit or even a personal drone unit.
- **What are physical devices in IoT?**
- IoT refers to any **system of physical devices that receive and transfer data over wireless networks with limited human intervention**. This is made possible by integrating compute devices in all kinds of objects.

IoT Physical Devices & Endpoints

- **IoT Device**

- Thing in Internet of Things(IoT) can be any object that has a unique identifier and which can send/receive data over a network.
- IoT devices are connected to the internet and send information about themselves or about their surroundings over a network or allow actuation upon the physical entities or environment around them remotely.
- **Some example of IoT devices are:**
 - A Home Automation device that allows remotely control and monitor the appliances.
 - An Industrial Machine which sends information about its operation and health monitoring data to a server.
 - A car sends information about its location to a cloud based service.
 - A wireless enabled wearable device that measures data about a person such as number of steps walked and send the data to a cloud based service.

IoT Physical Devices & Endpoints

- **Basic Building block of an IoT device**

- IoT stands for "Internet of Things," which means using the Internet to connect different things. IoT is an intersection between the physical and virtual worlds. It essentially maps virtual operations onto real interactions.

- **IoT device consists of a number of modules based on functional attributes:**

- **Sensors:**

- It can be either on board the IoT device or attached to the device. IoT device can collect various types of information from the onboard or attached sensors such as temperature, humidity, light intensity etc.
- The sensed information can be communicated either to other devices or cloud based server/storage. These form the front end of the IoT devices. These are the so called "Things" of the system.
- Their main purpose is to collect data from its surrounding (sensors) or give out data to its surrounding (actuators). These have to be uniquely identifiable devices with a unique IP address so that they can be easily identifiable over a large network. It should be able to collect real time data. Examples of sensors are: gas sensor, water quality sensor, moisture sensor etc.

IoT Physical Devices & Endpoints

- **Actuation:**

- IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device. Example relay switch connected to an IoT device can turn appliances on/off based on the commands send to the device.

- **Processors:**

- Processors are the brain of the IoT system. Their main function is to process the data captured by the sensors and process them so as to extract the valuable data from the enormous amount of raw data collected. It gives intelligence to the data. Processors mostly work on real-time basis and can be easily controlled by applications. These are also responsible for securing the data – that is performing encryption and decryption of data.

IoT Physical Devices & Endpoints

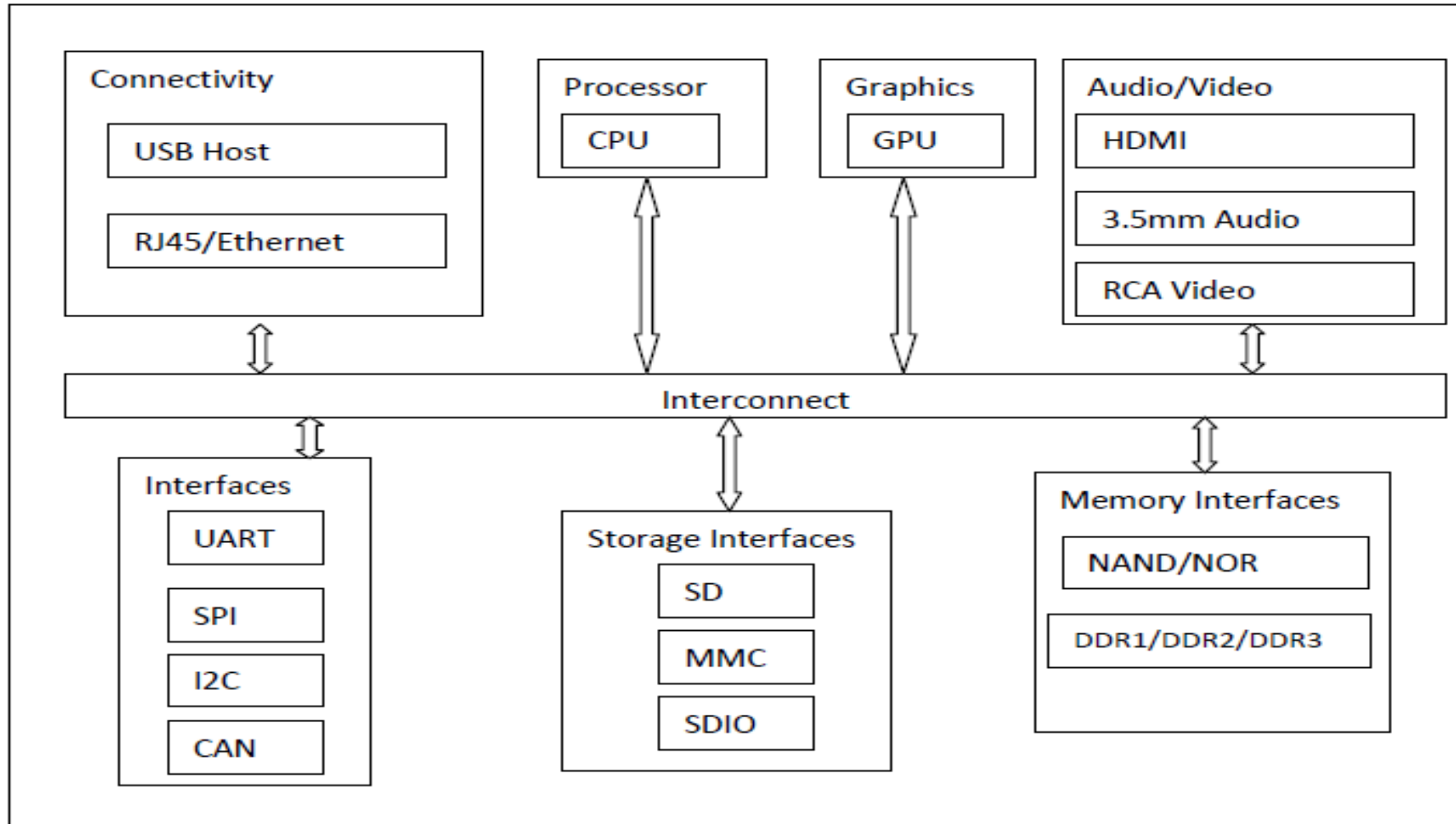
- **Communication:**

- It is responsible for sending collected data to other devices or cloud based servers/storage and receiving data from other devices and commands from remote applications.
- Gateways are responsible for routing the processed data and send it to proper locations for its (data) proper utilization. Gateway helps in to and fro communication of the data. It provides network connectivity to the data.
- Network connectivity is essential for any IoT system to communicate. LAN, WAN, PAN etc are examples of network gateways.

- **Analysis & Processing:**

- These are responsible for taking decision based upon the collected data.
- Given diagram shows the Single Board Computer(SBC) based IoT device that includes CPU, GPU, RAM, storage and various types of interfaces and peripherals.

IoT Physical Devices & Endpoints



IoT Device

- **What are IoT devices?**
- IoT devices are the nonstandard computing devices that connect wirelessly to a network and have the ability to transmit data, such as the many devices on the internet of things ([IoT](#)).
- IoT involves extending internet connectivity beyond standard devices, such as desktops, laptops, smartphones and tablets, to any range of traditionally "dumb" or non-internet-enabled physical devices and everyday objects. Embedded with technology, these devices can communicate and interact over the internet. They can also be [remotely monitored and controlled](#).
- **What are the IoT devices and examples of IoT devices?**
- IoT devices are pieces of hardware, such as sensors, actuators, gadgets, appliances, or machines, that are programmed for certain applications and can transmit data over the internet or other networks. They can be embedded into other mobile devices, industrial equipment, environmental sensors, medical devices, and more.
- There are several top devices in the market. **Smart Mobiles, smart refrigerators, smart watches, smart fire alarm, smart door lock, smart bicycle, medical sensors, fitness trackers, smart security system** etc., are few examples of IoT products.
- **Why IoT Devices?**
- Increasingly, IoT devices are using AI and machine learning to bring intelligence and autonomy to systems and processes, such as autonomous driving, industrial smart manufacturing, medical equipment, and home automation. Many of these devices are small, power- and cost-constrained microcontroller-based systems. Network bandwidth and consumer expectations around data privacy and user experience continue to demand more on-device processing, where data is processed on the IoT endpoint, rather than using cloud-based approaches.

IoT Device

- **How do IoT Devices Work?**
- Different IoT devices have different functions, but they all have similarities in terms of how they work. Firstly, IoT devices are physical objects that sense things going on in the physical world. They contain an integrated CPU, network adapter and firmware, and are usually connected to a Dynamic Host Configuration Protocol server. It also requires an IP address to function over the network.
- Most IoT devices are configured and managed through a software application. For example, an app on your smartphone to control the lights in your home. Some devices also have integrated web servers, which eliminates the need for external applications. For example, the lights switch on immediately when you enter a room.

Examples of IoT Devices

- **Home Security**

- The key driver behind smart and secure homes is IoT. A variety of sensors, lights, alarms and cameras (all of which can be controlled from a smartphone) are connected via IoT to provide 24x7 security.

- **Activity Trackers**

- Smart home security cameras provide alerts and peace of mind. Activity trackers are sensor devices that can monitor and transmit key health indicators in real-time. You can track and manage your blood pressure, appetite, physical movement and oxygen levels.

- **Industrial Security and Safety**

- IoT-enabled detection systems, sensors and cameras can be placed in restricted areas to detect trespassers. They can also identify pressure buildups and small leaks of hazardous chemicals and fix them before they become serious problems.

- **Augmented Reality Glasses**

- Augmented Reality (AR) glasses are wearable computer-enabled glasses that help you get extra information such as 3D animations and videos to the user's real-world scenes. The information is presented within the lenses of the glasses and can help users access Internet applications.

- **Motion Detection**

- Motion sensors can detect vibrations in buildings, bridges, dams and other large-scale structures. These devices can identify anomalies and disturbances in the structures that could lead to catastrophic failures. They can also be used in areas susceptible to floods, landslides, and earthquakes.

Most Popular IoT Devices in 2021

- **1. Google Home Voice Controller**
- [Google Home](#) voice controller is one of the most popular IoT devices out there today. It provides voice-enabled services like alarms, lights, thermostats, volume control and lots more.
- **2. Amazon Echo Plus Voice Controller**
- [Amazon Echo Plus](#) voice controller is another popular and reliable IoT device on the market. It provides voice-enabled services like answering phone calls, setting timers and alarms, checking the weather, and lots more.
- **3. August Doorbell Cam**
- [August Doorbell Cam](#) is an IoT device that allows you to answer your door from any remote location. It constantly captures motion changes and suspicious activity in your doorstep.
- **4. August Smart Lock**
- [August Smart Lock](#) is a proven and reliable security IoT device that helps users to manage their doors from any remote location. It helps keep thieves away and provides an extra layer of security for your home.
- **5. Foobot**
- [Foobot](#) is an IoT device that can accurately measure indoor pollution. It helps to improve the air quality in houses, cafes, workplaces, and other indoor public spaces.

Building Blocks

- Building IOT with RASPERRY PI
- Internet of Things The Internet of Things (IoT) is a scenario in which objects, animals or people are provided with single identifiers and the capability to automatically transfer and the capability to automatically transfer data more to a network without requiring human-to-human or human-to-computer communication.

Introduction to Raspberry Pi

- The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things (IoT).
- Raspberry Pi was basically introduced in 2006.
- It is particularly designed for educational use and intended for Python.
- A Raspberry Pi is of small size i.e., of a credit card sized single board computer, which is developed in the United Kingdom(U.K) by a foundation called Raspberry Pi.
- There have been three generations of Raspberry Pis: Pi 1, Pi 2, and Pi 3
- The first generation of Raspberry (Pi 1) was released in the year 2012, that has two types of models namely model A and model B.
- Raspberry Pi can be plugged into a TV, computer monitor, and it uses a standard keyboard and mouse. ☐ It is user friendly as can be handled by all the age groups

Introduction to Raspberry Pi

- It does everything you would expect a desktop computer to do like word-processing, browsing the internet spreadsheets, playing games to playing high definition videos.
- All models feature on a broadcom system on a chip (SOC), which includes chip graphics processing unit GPU(a Video Core IV), an ARM compatible and CPU.
- The CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM.
- An operating system is stored in the secured digital SD cards and program memory in either the MicroSDHC or SDHC sizes.
- Most boards have one to four USB slots, composite video output, HDMI and a 3.5 mm phone jack for audio. Some models have WiFi and Bluetooth.
- Several generations of Raspberry Pis have been released.
- All models feature a Broadcom system on a chip (SoC) with an integrated ARMcompatible central processing unit (CPU) and on-chip graphics processing unit (GPU).
- Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ or 1.5 GHz for the Pi 4; on-board memory ranges from 256 MB to 1 GB with up to 4 GB available on the Pi 4 random-access memory (RAM).

Introduction to Raspberry Pi

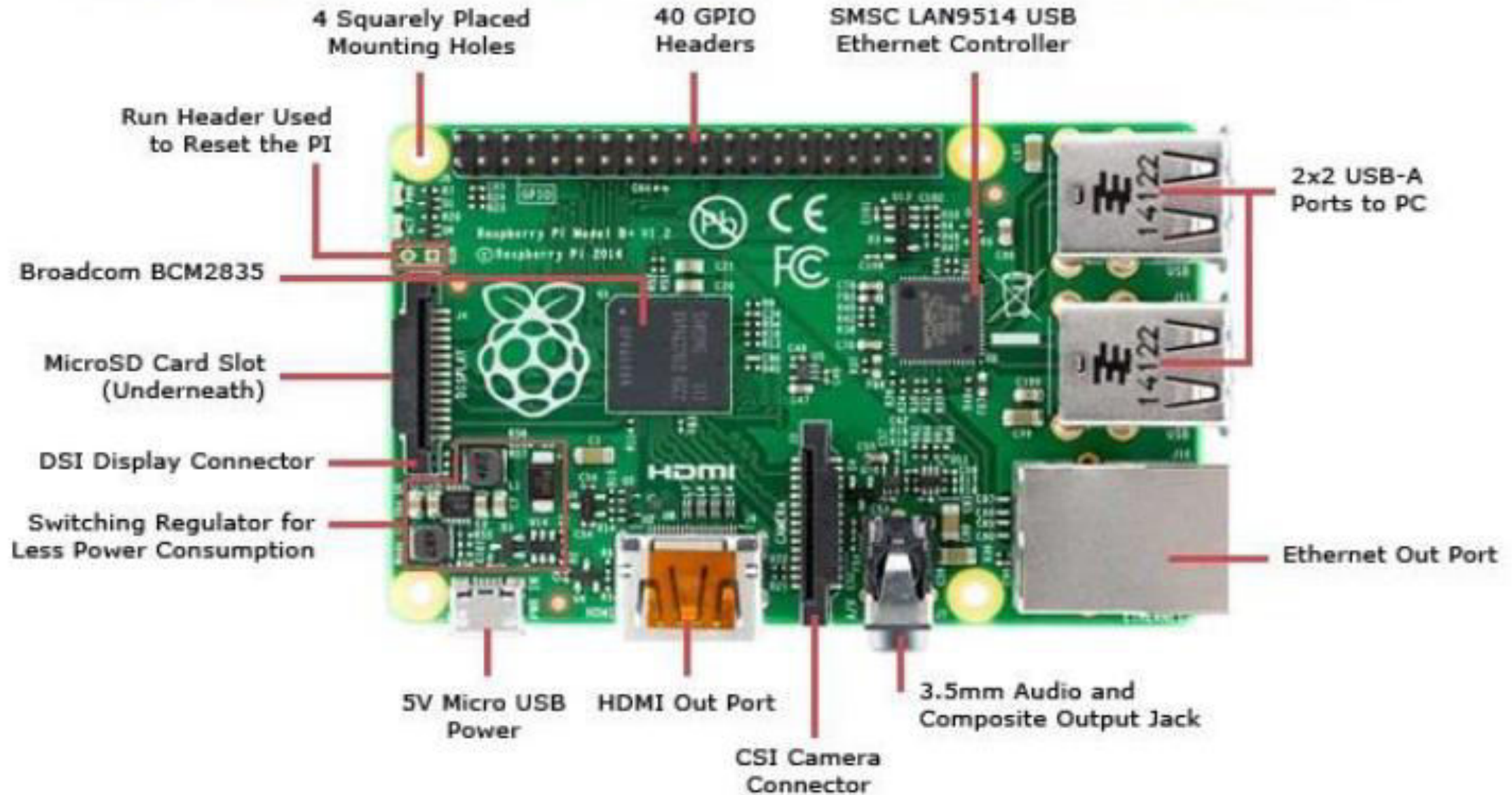
- Several generations of Raspberry Pis have been released.
- All models feature a Broadcom system on a chip (SoC) with an integrated ARMcompatible central processing unit (CPU) and on-chip graphics processing unit (GPU).
- Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ or 1.5 GHz for the Pi 4; on-board memory ranges from 256 MB to 1 GB with up to 4 GB available on the Pi 4 random-access memory (RAM).
- Secure Digital (SD) cards in MicroSDHC form factor (SDHC on early models) are used to store the operating system and program memory.
- The boards have one to five USB ports. For video output, HDMI and composite video are supported, with a standard 3.5 mm tip-ring-sleeve jack for audio output.
- Lower-level output is provided by a number of GPIO pins, which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3 and Pi Zero W have on-board Wi-Fi and Bluetooth.

GPIO Pin Diagram

GPIO Pinout Diagram



Raspberry Pi-Components



Raspberry Pi-Components

- **Components and Peripherals**
- **Voltages:** Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V). The remaining pins are all general purpose 3V3 pins
- A GPIO pin designated as an output pin can be set to high (3V3) or low (0V). A GPIO pin designated as an input pin can be read as high (3V3) or low (0V).
- **Processor & RAM:** Raspberry based on ARM11 processor. Latest version supports 700MHz processor and 512MB SDRAM. The Central processing unit is the brain of the raspberry pi board and that is responsible for carrying out the instructions of the computer through logical and mathematical operations.
- **Ethernet:** The Ethernet port of the raspberry pi is the main gateway for communicating with additional devices. The raspberry pi Ethernet port is used to plug your home router to access the internet.

Raspberry Pi-Components

- **USB Ports:** It has 2 USB ports. USB port provide current upto 100mA. For connecting devices that draw current more than 100mA, an external USB powered hub is required.
- **Ethernet Port:** It has standard RJ45 Ethernet port. Connect Ethernet cable or USB wifi adapter to provide internet connectivity.
- **HDMI Output:** It supports both audio and video output. Connect raspberry Pi to monitor using HDMI cable.
- **Composite video Output:** Raspberry comes with a composite video output with an RCA jack that supports both PAL and NTSC video output.
- **Audio Output:** It has 3.5mm audio output jack. This audio jack is used for providing audio output to old television along with RCA jack for video.
- **GPIO Pins:** It has a number of general purpose input/output pins. These pins are used to connect other electronic components. For example, you can connect it to the temperature sensor to transmit digital data.

Raspberry Pi-Components

- **Display Serial Interface (DSI):** DSI interface are used to connect an LCD panel to Raspberry Pi.
- **Cameral Serial Interface(CSI):** CSI interface are used to connect a camera module to Raspberry Pi.
- **SD Card slot:** Raspberry does not have built in OS and storage. Plug in an SD card loaded with Linux to SD card slot.
- **Power Input:** Raspberry has a micro USP connector for power input.
- **Memory:** The raspberry pi model A board is designed with 256MB of SDRAM and model B is designed with 512MB. Raspberry pi is a small size PC compare with other PCs. The normal PCs RAM memory is available in gigabytes. But in raspberry pi board, the RAM memory is available more than 256MB or 512MB
- **Status LEDs:** Raspberry has 5 status LEDs.

Status	LED Function
ACT	SD card Access
PWR	3.3V power is present
FDX	Full duplex LAN Connected
LNK	Link/Network Activity
100	100 Mbit LAN connected

Programming Raspberry Pi with Python

- **Raspberry Pi**
- The Raspberry Pi is a [single-board computer](#) developed by the [Raspberry Pi Foundation](#), a UK-based charity organization. Originally designed to provide young people with an affordable computing option to learn how to program, it has developed a massive following in the maker and DIY communities because of its compact size, full Linux environment, and general-purpose input–output (**GPIO**) pins.
- With all the features and capabilities that are packed into this small board, there's no shortage of projects and use cases for the Raspberry Pi.

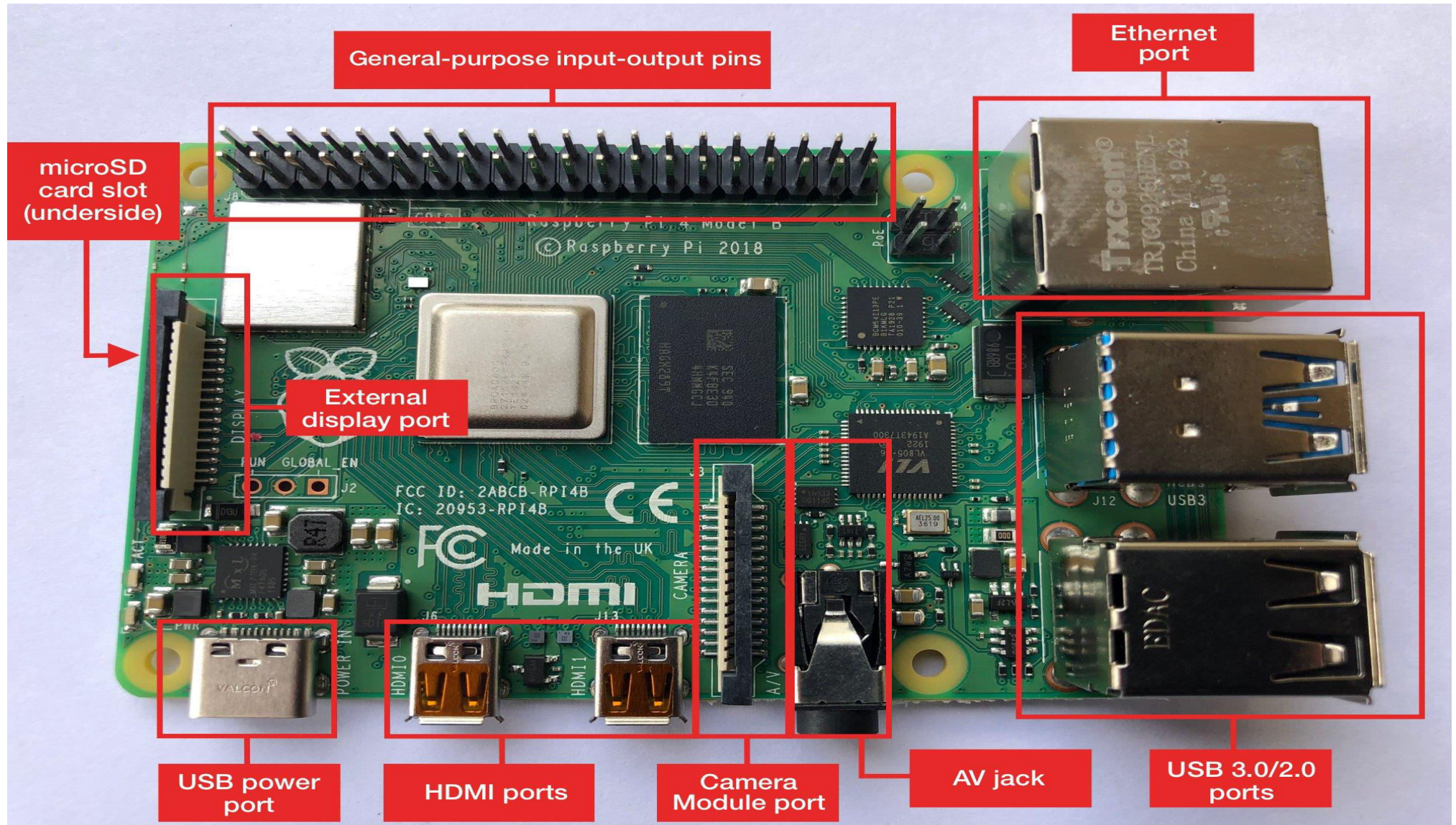
Programming Raspberry Pi with Python

- Some example projects include the following:
- [Line-following robot](#)
- [Home weather station](#)
- [Retro gaming machine](#)
- [Real-time object detection camera](#)
- [Minecraft server](#)
- [Button-controlled music box](#)
- [Media center](#)
- [Remote experiments on the International Space Station](#)
- If you can think of a project that would benefit from having a credit card-sized computer attached to it, then someone has probably used a Raspberry Pi to do it. The Raspberry Pi is a fantastic way to bring your Python project ideas to life.

Raspberry Pi Board Overview

- The Raspberry Pi comes in a variety of [form factors](#) for different use cases. In this tutorial, you'll be looking at the most recent version, the [Raspberry Pi 4](#).
- Below is the board layout of the Raspberry Pi 4. While this layout is slightly different from previous models of the Raspberry Pi, most of the connections are the same. The setup described in the next section should be the same for both a Raspberry Pi 3 and a Raspberry Pi 4:

Raspberry Pi Board Overview



Raspberry Pi Board Overview

- The Raspberry Pi 4 board contains the following components:
- **General-purpose input–output pins:** These pins are used to connect the Raspberry Pi to electronic components.
- **Ethernet port:** This port connects the Raspberry Pi to a wired network. The Raspberry Pi also has Wi-Fi and Bluetooth built in for wireless connections.
- **Two USB 3.0 and two USB 2.0 ports:** These USB ports are used to connect peripherals like a keyboard or mouse. The two black ports are USB 2.0 and the two blue ports are USB 3.0.
- **AV jack:** This AV jack allows you to connect speakers or headphones to the Raspberry Pi.

Raspberry Pi Board Overview

- **Camera Module port:** This port is used to connect the [official Raspberry Pi Camera Module](#), which enables the Raspberry Pi to capture images.
- **HDMI ports:** These HDMI ports connect the Raspberry Pi to external monitors. The Raspberry Pi 4 features two micro HDMI ports, allowing it to drive two separate monitors at the same time.
- **USB power port:** This USB port powers the Raspberry Pi. The Raspberry Pi 4 has a **USB Type-C** port, while older versions of the Pi have a **micro-USB** port.
- **External display port:** This port is used to connect the official seven-inch Raspberry Pi [touch display](#) for touch-based input on the Raspberry Pi.
- **microSD card slot (underside of the board):** This card slot is for the microSD card that contains the Raspberry Pi operating system and files.

Raspberry Pi vs Arduino

- People often wonder what the difference is between a Raspberry Pi and an Arduino. The Arduino is another device that is widely used in physical computing. While there is some overlap in the capabilities of the Arduino and the Raspberry Pi, there are some distinct differences.
- [The Arduino platform](#) provides a hardware and software interface for programming [microcontrollers](#). A microcontroller is an [integrated circuit](#) that allows you to read input from and send output to electronic components. Arduino boards generally have limited memory, so they're often used to repeatedly run a single program that interacts with electronics.
- The Raspberry Pi is a general-purpose, Linux-based computer. It has a full operating system with a GUI interface that is capable of running many different programs at the same time.
- The Raspberry Pi comes with a variety of software preinstalled, including a web browser, an office suite, a terminal, and even Minecraft. The Raspberry Pi also has built-in Wi-Fi and Bluetooth to connect to the Internet and external peripherals.
- For running Python, the Raspberry Pi is often the better choice, as you get a full-fledged Python installation out of the box without any configuration.

Setting Up the Raspberry Pi

- Unlike the Arduino, which requires only a USB cable and a computer to set up, the Raspberry Pi has more hardware requirements to get up and running. After the initial setup, though, some of these peripherals will no longer be required.
- **Required Hardware**
- The following hardware is required for the initial setup of your Raspberry Pi. If you end up connecting to your Raspberry Pi over SSH, then some of the hardware below will not be needed after the initial setup.
- **Monitor**
- You'll need a monitor during the initial setup and configuration of the operating system. If you'll be using SSH to connect to your Raspberry Pi, then you won't need the monitor after setup. Make sure your monitor has an HDMI input.
- **microSD Card**
- Raspberry Pi uses a microSD card to store the operating system and files. If you buy a [Raspberry Pi kit](#), then it will contain a preformatted microSD card for you to use. If you buy a microSD card separately, then you'll need to [format it yourself](#). Look for a microSD card with at least 16GB of capacity.

Setting Up the Raspberry Pi

- **Keyboard and Mouse**

- A USB keyboard and mouse are required during the initial setup of the Raspberry Pi. Once the setup is complete, you can switch to using Bluetooth versions of these peripherals if you prefer. Later in this tutorial, you'll see how to connect to the Raspberry Pi over SSH. If you choose to connect this way, then a physical keyboard and mouse are not required after the initial setup.

- **HDMI Cables**

- You'll need an HDMI cable to connect the Raspberry Pi to a monitor. Different Raspberry Pi models have different HDMI cable requirements:

	Raspberry Pi	Raspberry Pi
Raspberry Pi 4	3/2/1	Zero
micro HDMI	HDMI	mini HDMI
micro HDMI to HDMI	HDMI to HDMI	mini HDMI to HDMI

Depending on your model, you may need to purchase a special HDMI cable or adapter.

Setting Up the Raspberry Pi

- **Power Supply**
- The Raspberry Pi uses a USB connection to power the board. Again, different Raspberry Pi models have different USB connection and power requirements.
- Below are the connection and power requirements for the different models:

Raspberry Pi 4	Raspberry Pi 3/2/1/Zero
USB-C	Micro-USB
At least 3.0 amps	At least 2.5 amps

To avoid any confusion when selecting a power supply, it's recommended that you use the official power supply for your [Raspberry Pi 4](#) or [other model](#).

Setting Up the Raspberry Pi

- **Optional Hardware**

- You can use a whole range of additional hardware with the Raspberry Pi to extend its capabilities. The hardware items listed below are not required to use your Raspberry Pi but would be useful to have on hand.

- **Case**

- It's nice to have a case for your Raspberry Pi to keep its components from being damaged during normal use. When selecting a case, make sure that you purchase the correct type for your model of the Raspberry Pi.

- **Speakers**

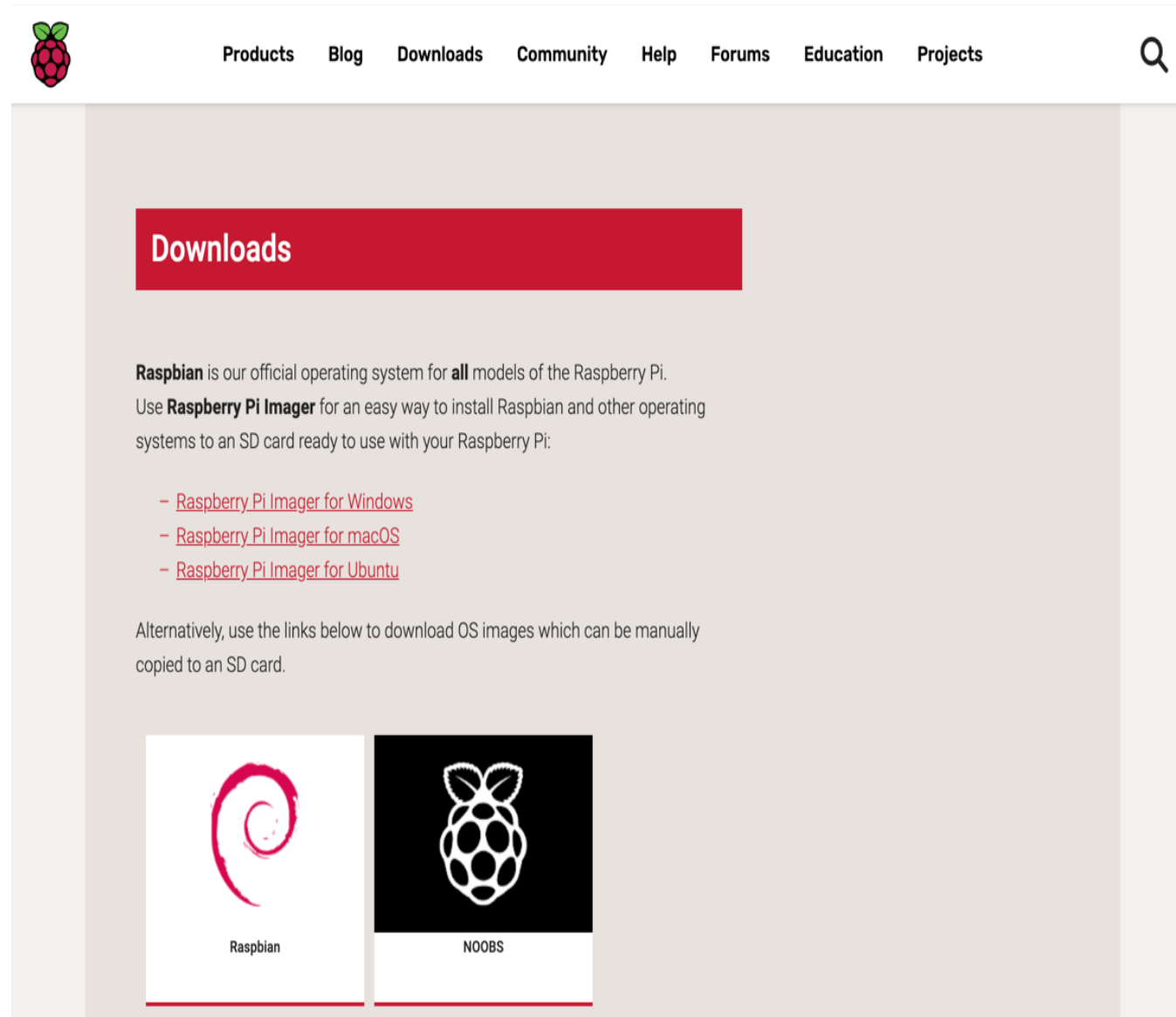
- If you want to [play music or sound](#) from your Raspberry Pi, then you'll need speakers. These can be any standard speakers that have a 3.5 mm jack. You can connect the speakers to the Raspberry Pi using the [AV jack](#) on the side of the board.

Setting Up the Raspberry Pi

- **Heat Sinks (Recommended)**
- The Raspberry Pi can do a lot of computing for a little board. This is one of the reasons it's so awesome! But this does mean that it can get a little hot sometimes. It's recommended that you purchase a [set of heatsinks](#) to prevent the Raspberry Pi from [throttling the CPU](#) when it gets too hot.
- **Software**
- The operating system for the Raspberry Pi is stored on a microSD card. If your card did not come from an official Raspberry Pi kit, then you'll need to install the operating system on it.
- There are multiple ways to set up the operating system on your Raspberry Pi. You can find out more about the different installation options on the [Raspberry Pi website](#).
- In this section, you'll look at two ways to install **Raspbian**, the officially supported Raspberry Pi operating system, which is based on Debian Linux.

Setting Up the Raspberry Pi

- **Raspberry Pi Imager (Recommended)**
- The Raspberry Pi foundation recommends that you use the **Raspberry Pi Imager** for the initial setup of your SD card. You can download the imager from the [Raspberry Pi Downloads page](#). Once on this page, download the appropriate version for your operating system:



Setting Up the Raspberry Pi

- After you've downloaded the Raspberry Pi Imager, start the application. You'll see a screen that allows you to select the operating system that you want to install along with the SD card you would like to format:

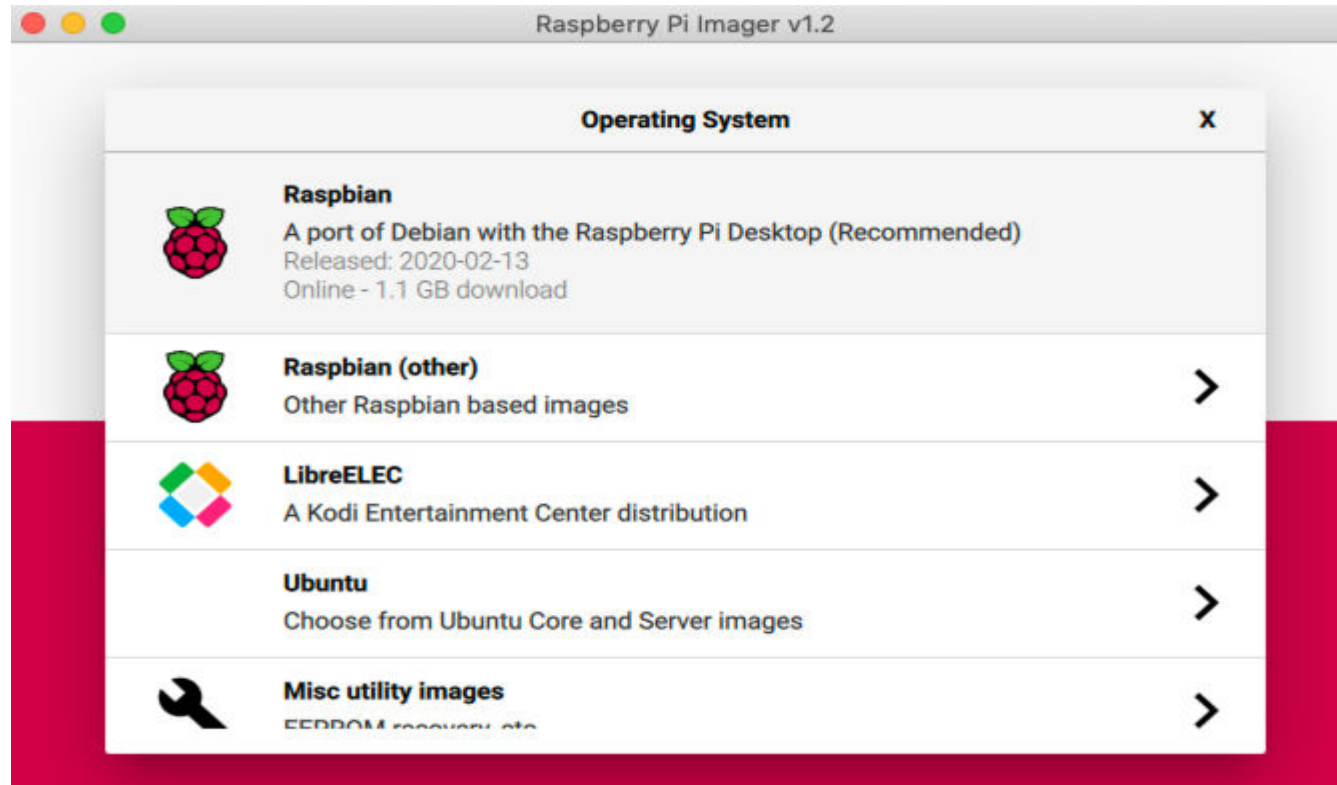


You'll be given two options when first loading the application: *Choose OS* and *Choose SD Card*. Select *Choose OS* first.

Note: There's a chance that Windows may prevent the Raspberry Pi Imager from starting because it's an unrecognized application. If you receive a pop-up that says *Windows protected your PC*, then you can still run the application by clicking *More info* and selecting *Run anyway*.

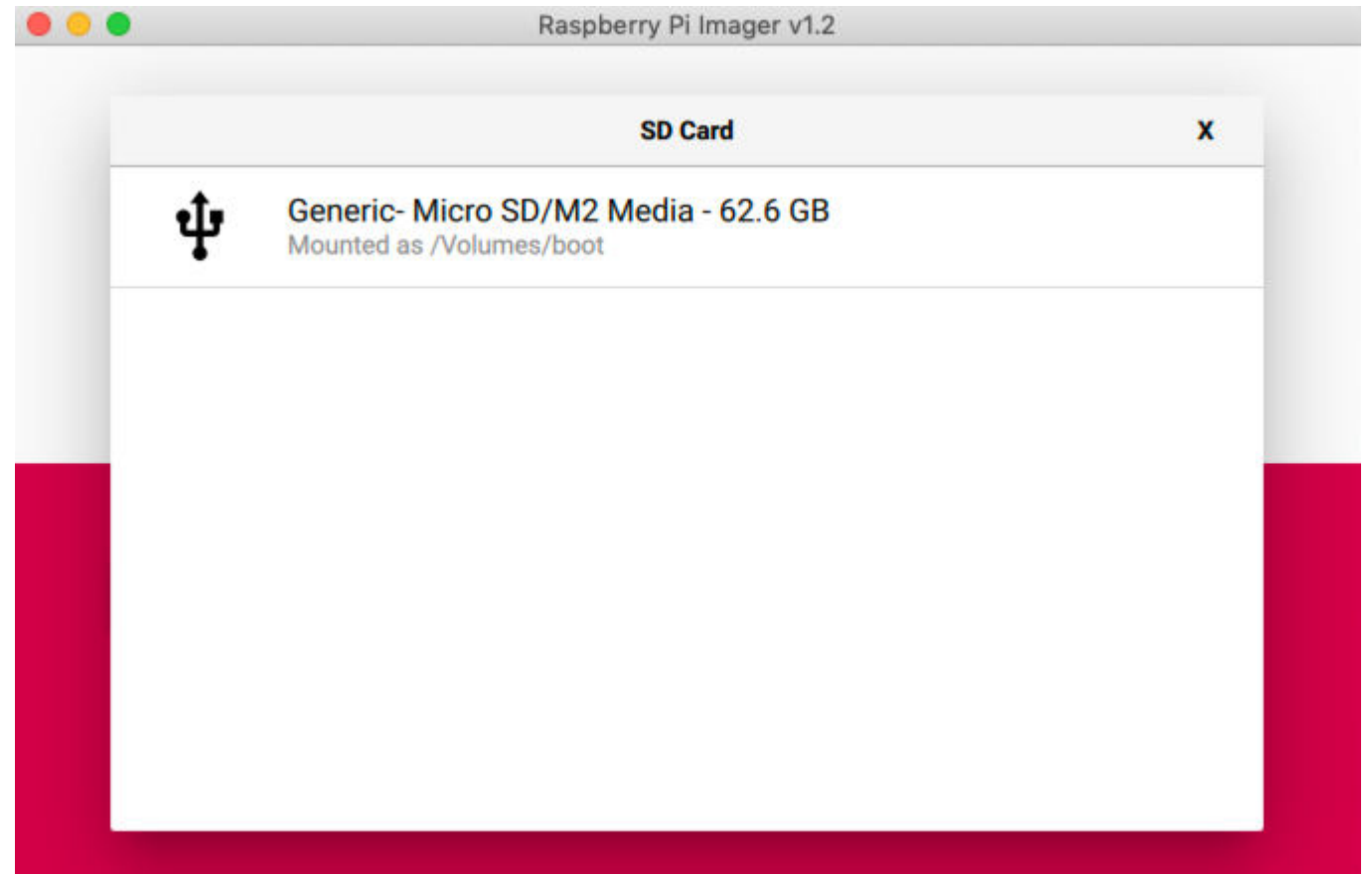
Setting Up the Raspberry Pi

- With the application running, click the *Choose OS* button and choose the first *Raspbian* option:



Setting Up the Raspberry Pi

- After selecting the Raspbian operating system, you need to select the SD card you're going to use. Make sure your microSD card is inserted into your computer and click *Choose SD Card*, then select the SD card from the menu:



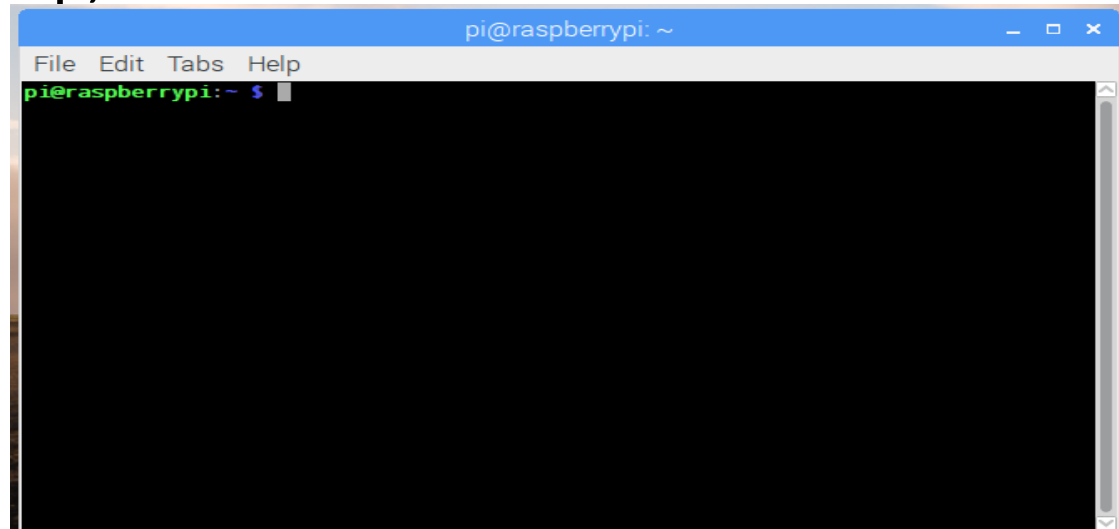
For Detail Study of Raspberry pi installation : go through the following link
<https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started>

Linux on Raspberry Pi

- **Terminal**
- The terminal (or 'command-line') on a computer allows a user a great deal of control over their system. Users of Windows may already have come across command Prompt or Powershell, while mac OS users may be familiar with Terminal. All of these tools allow a user to directly manipulate their system through the use of commands. These commands can be chained together and/or combined together into complex [scripts](#) that can potentially complete tasks more efficiently than much larger traditional software packages

Linux on Raspberry Pi

- On the Raspberry Pi OS, the default terminal application is called LXTerminal. This is known as a 'terminal emulator', this means that it emulates the old style video terminals — from before Windowing systems were developed — inside a graphical environment. The application can be found on the Raspberry Pi desktop, and when started will look something like this:



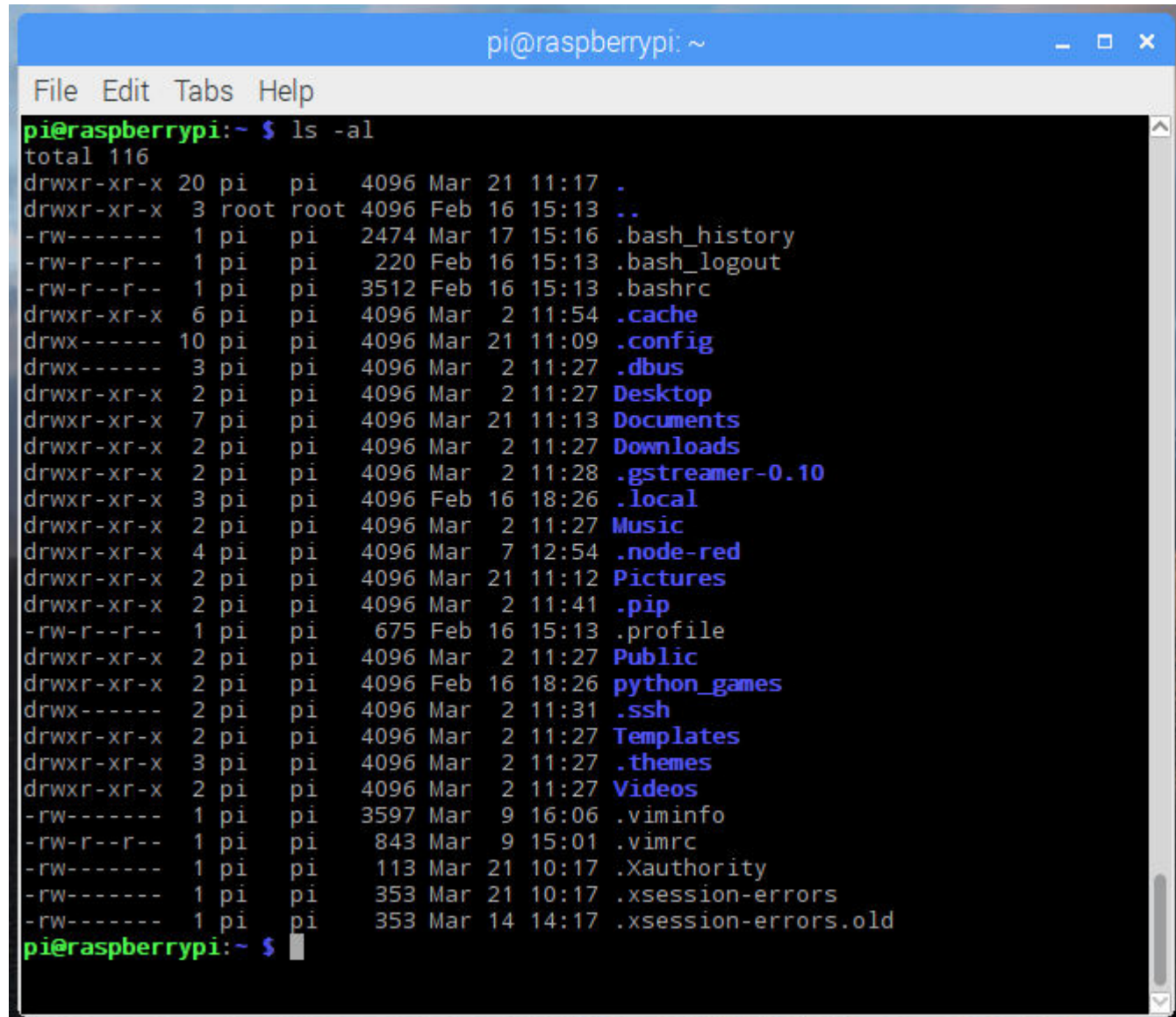
- In the terminal window you should be able to see the following prompt:

pi@raspberrypi ~\$

- This shows your username and the hostname of the Pi. Here the username is pi and the hostname is raspberrypi.

Linux on Raspberry Pi

- **Navigating and browsing your Pi**
- One of the key aspects of using a terminal is being able to navigate your file system. Go ahead and type `ls -la` into the Terminal window, and then hit the RETURN key. You should see something similar to:



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ ls -al  
total 116  
drwxr-xr-x 20 pi pi 4096 Mar 21 11:17 .  
drwxr-xr-x 3 root root 4096 Feb 16 15:13 ..  
-rw----- 1 pi pi 2474 Mar 17 15:16 .bash_history  
-rw-r--r-- 1 pi pi 220 Feb 16 15:13 .bash_logout  
-rw-r--r-- 1 pi pi 3512 Feb 16 15:13 .bashrc  
drwxr-xr-x 6 pi pi 4096 Mar 2 11:54 .cache  
drwx----- 10 pi pi 4096 Mar 21 11:09 .config  
drwx----- 3 pi pi 4096 Mar 2 11:27 .dbus  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:27 Desktop  
drwxr-xr-x 7 pi pi 4096 Mar 21 11:13 Documents  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:27 Downloads  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:28 .gststreamer-0.10  
drwxr-xr-x 3 pi pi 4096 Feb 16 18:26 .local  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:27 Music  
drwxr-xr-x 4 pi pi 4096 Mar 7 12:54 .node-red  
drwxr-xr-x 2 pi pi 4096 Mar 21 11:12 Pictures  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:41 .pip  
-rw-r--r-- 1 pi pi 675 Feb 16 15:13 .profile  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:27 Public  
drwxr-xr-x 2 pi pi 4096 Feb 16 18:26 python_games  
drwx----- 2 pi pi 4096 Mar 2 11:31 .ssh  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:27 Templates  
drwxr-xr-x 3 pi pi 4096 Mar 2 11:27 .themes  
drwxr-xr-x 2 pi pi 4096 Mar 2 11:27 Videos  
-rw----- 1 pi pi 3597 Mar 9 16:06 .viminfo  
-rw-r--r-- 1 pi pi 843 Mar 9 15:01 .vimrc  
-rw----- 1 pi pi 113 Mar 21 10:17 .Xauthority  
-rw----- 1 pi pi 353 Mar 21 10:17 .xsession-errors  
-rw----- 1 pi pi 353 Mar 14 14:17 .xsession-errors.old  
pi@raspberrypi:~ $
```


Linux on Raspberry Pi

- The **ls** command lists the contents of the directory that you are currently in (your present working directory).
- The **-la** component of the command is what's known as a 'flag'. Flags modify the command that's being run.
- In this case the **l** displays the contents of the directory in a list, showing data such as their sizes and when they were last edited, and the **a** displays all files, including those beginning with a. known as 'dotfiles'. Dotfiles usually act as configuration files for software and as they are written in text, they can be modified by simply editing them.

Linux on Raspberry Pi

- In order to navigate to other directories, the change directory command, `cd` can be used. You can specify the directory that you want to go to by either the 'absolute' or the 'relative' path.
- So if you wanted to navigate to the ***python_games*** directory, you could either do ***cd /home/pi/python_games*** or just ***cd python_games*** (if you are currently in ***home/pi***).
- There are some special cases that may be useful: `~` acts as an alias for your home directory, so ***~/python_games*** is the same as ***/home/pi/python_games***; `.` and `..` are aliases for the current directory and the parent directory respectively, e.g. if you were in ***home/pi/python_games***, ***cd..*** would take you to ***/home/pi***.

Linux on Raspberry Pi

- **History and auto-complete**
- Rather than type every command, the terminal allows you to scroll through previous commands that you've run by pressing the up or down keys on your keyboard.
- If you are writing the name of a file or directory as part of a command then pressing tab will attempt to auto-complete the name of what you are typing. For example, if you have a file in a directory called aLongFileName then pressing tab after typing a will allow you to choose from all file and directory names beginning with a in the current directory, allowing you to choose aLongFileName.

Linux on Raspberry Pi

- **The Sudo command**
- Some commands that make permanent changes to the state of your system require you to have root privileges to run. The command `sudo` temporarily gives your account (if you're not already logged in as root) the ability to run these commands, provided your user name is in a list of users ('`sudoers`'). When you append `sudo` to the start of a command and press enter, the command following `sudo` will be run using root privileges. Be very careful: commands requiring root privileges can irreparably damage your system! Note that on some systems you will be prompted to enter your password when you run a command with `sudo`.

Linux on Raspberry Pi

- **Installing software using apt**
- The [apt](#) command is used to install software in Raspberry Pi OS. This is the 'package manager' that is included with any Debian-based Linux distributions, including Raspberry Pi OS. It allows you to install and manage new software packages on your Raspberry Pi.
- In order to install a new package, you would type `sudo apt install <package-name>`, where `<package-name>` is the package that you want to install.
- Running `sudo apt update` will update a list of software packages that are available on your system. If a new version of a package is available, then `sudo apt full-upgrade` will update any old packages to the new version.
- Finally, `sudo apt remove <package-name>` removes or uninstalls a package from your system.

Linux on Raspberry Pi

- **Other useful commands**

- There are a few other commands that you may find useful, these are listed below:
- **cp** makes a copy of a file and places it at the specified location (essentially doing a 'copy-paste'), for example - `cp file_a /home/other_user/` would copy the file `file_a` from your home directory to that of the user `other_user` (assuming you have permission to copy it there). Note that if the target is a folder, the filename will remain the same, but if the target is a filename, it will give the file the new name.
- **mv** moves a file and places it at the specified location (so where `cp` performs a 'copy-paste', `mv` performs a 'cut-paste'). The usage is similar to `cp`, so `mv file_a /home/other_user/` would move the file `file_a` from your home directory to that of the specified user. `mv` is also used to rename a file, i.e. move it to a new location, e.g. `mv hello.txt story.txt`.
- **rm** removes the specified file (or directory when used with `-r`). **Warning:** Files deleted in this way are generally not restorable.
- **mkdir:** This makes a new directory, e.g. `mkdir new_dir` would create the directory `new_dir` in the present working directory.
- **cat** lists the contents of files, e.g. `cat some_file` will display the contents of `some_file`.

Linux on Raspberry Pi

- **Finding out about a command**
- To find out more information about a particular command then you can run the `man` followed by the command you want to know more about (e.g. `man ls`). The man-page (or manual page) for that command will be displayed, including information about the flags for that program and what effect they have. Some man-pages will give example usage.

Linux on Raspberry Pi

- **The Linux File System**

- It is important to have a basic understanding of the fundamentals of the Linux file system: where your files are kept, where software is installed, where the danger zones are, and so on.

- **Home**

- When you log into a Pi and open a terminal window, or you boot to the command line instead of the graphical user interface, you start in your home folder; this is located at `/home/pi`, assuming your username is `pi`.
- This is where the user's own files are kept. The contents of the user's desktop is in a directory here called `Desktop`, along with other files and folders.
- To navigate to your home folder on the command line, simply type `cd` and press Enter. This is the equivalent of typing `cd /home/pi`, where `pi` is your username. You can also use the tilde key (`~`), for example `cd ~`, which can be used to relatively link back to your home folder. For instance, `cd ~/Desktop/` is the same as `cd /home/pi/Desktop`.
- Navigate to `/home/` and run `ls`, and you'll see the home folders of each of the users on the system.
- Note that if logged in as the `root` user, typing `cd` or `cd ~` will take you to the [root](#) user's home directory; unlike normal users, this is located at `/root/` not `/home/root/`.

Linux on Raspberry Pi

- **Linux Commands**

Linux Command	Use
ls	The ls command lists the content of the current directory (or one that is specified). It can be used with the -l flag to display additional information (permissions, owner, group, size, date and timestamp of last edit) about each file and directory in a list format. The -a flag allows you to view files beginning with . (i.e. dotfiles).
cd	Using cd changes the current directory to the one specified. You can use relative (i.e. cd directoryA) or absolute (i.e. cd /home/pi/directoryA) paths.
pwd	The pwd command displays the name of the present working directory: on a Raspberry Pi, entering pwd will output something like /home/pi.
mkdir	You can use mkdir to create a new directory, e.g. mkdir newDir would create the directory newDir in the present working directory.
rmdir	To remove empty directories, use rmdir. So, for example, rmdir oldDir will remove the directory oldDir only if it is empty.
rm	The command rm removes the specified file (or recursively from a directory when used with -r). Be careful with this command: files deleted in this way are mostly gone for good!

Linux on Raspberry Pi

• Linux Commands

Linux Command	Use
cp	Using cp makes a copy of a file and places it at the specified location (this is similar to copying and pasting). For example, <code>cp ~/fileA /home/otherUser/</code> would copy the file fileA from your home directory to that of the user otherUser (assuming you have permission to copy it there). This command can either take FILE FILE (<code>cp fileA fileB</code>), FILE DIR (<code>cp fileA /directoryB/</code>) or -r DIR DIR (which recursively copies the contents of directories) as arguments.
mv	The mv command moves a file and places it at the specified location (so where cp performs a 'copy-paste', mv performs a 'cut-paste'). The usage is similar to cp. So <code>mv ~/fileA /home/otherUser/</code> would move the file fileA from your home directory to that of the user otherUser. This command can either take FILE FILE (<code>mv fileA fileB</code>), FILE DIR (<code>mv fileA /directoryB/</code>) or DIR DIR (<code>mv /directoryB /directoryC</code>) as arguments. This command is also useful as a method to rename files and directories after they've been created.
touch	The command touch sets the last modified time-stamp of the specified file(s) or creates it if it does not already exist.
cat	You can use cat to list the contents of file(s), e.g. <code>cat thisFile</code> will display the contents of thisFile. Can be used to list the contents of multiple files, i.e. <code>cat *.txt</code> will list the contents of all .txt files in the current directory.
head	The head command displays the beginning of a file. Can be used with -n to specify the number of lines to show (by default ten), or with -c to specify the number of bytes.
tail	The opposite of head, tail displays the end of a file. The starting point in the file can be specified either through -b for 512 byte blocks -c for bytes or -n for number of lines

Linux on Raspberry Pi

• Linux Commands

Linux Command	Use
chmod	You would normally use chmod to change the permissions for a file. The chmod command can use symbols u (user that owns the file), g (the files group) , and o (other users) and the permissions r (read), w (write), and x (execute). Using chmod u+x filename will add execute permission for the owner of the file.
chown	The chown command changes the user and/or group that owns a file. It normally needs to be run as root using sudo e.g. sudo chown pi:root filename will change the owner to pi and the group to root.
ssh	ssh denotes the secure shell. Connect to another computer using an encrypted network connection. For more details see SSH (secure shell)
scp	The scp command copies a file from one computer to another using ssh. For more details see SCP (secure copy)
sudo	The sudo command enables you to run a command as a superuser, or another user. Use sudo -s for a superuser shell. For more details see Root user / sudo
dd	The dd command copies a file converting the file as specified. It is often used to copy an entire disk to a single file or back again. So, for example, dd if=/dev/sdd of=backup.img will create a backup image from an SD card or USB disk drive at /dev/sdd. Make sure to use the correct drive when copying an image to the SD card as it can overwrite the entire disk

Linux on Raspberry Pi

• Linux Commands

Linux Command	Use
df	Use df to display the disk space available and used on the mounted filesystems. Use df -h to see the output in a human-readable format using M for MBs rather than showing number of bytes.
unzip	The unzip command extracts the files from a compressed zip file.
tar	Use tar to store or extract files from a tape archive file. It can also reduce the space required by compressing the file similar to a zip file. To create a compressed file, use tar -cvzf filename.tar.gz directory/ To extract the contents of a file, use tar -xvzf filename.tar.gz
pipes	A pipe allows the output from one command to be used as the input for another command. The pipe symbol is a vertical line . For example, to only show the first ten entries of the ls command it can be piped through the head command ls head
tree	Use the tree command to show a directory and all subdirectories and files indented as a tree structure.
&	Run a command in the background with &, freeing up the shell for future commands.

Linux on Raspberry Pi

- **Linux Commands**

Linux Command	Use
wget	Download a file from the web directly to the computer with wget. So wget https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-datasheet.pdf will download the Raspberry Pi 4 datasheet and save it as raspberry-pi-4-datasheet.pdf.
curl	Use curl to download or upload a file to/from a server. By default, it will output the file contents of the file to the screen.
man	Show the manual page for a file with man. To find out more, run man man to view the manual page of the man command.

Linux on Raspberry Pi

- **Search Commands**

Search Command	Use
grep	<p>Use grep to search inside files for certain search patterns. For example, <code>grep "search" *.txt</code> will look in all the files in the current directory ending with <code>.txt</code> for the string search.</p> <p>The grep command supports regular expressions which allows special letter combinations to be included in the search.</p>
awk	<p>awk is a programming language useful for searching and manipulating text files.</p>
find	<p>The find command searches a directory and subdirectories for files matching certain patterns.</p>
whereis	<p>Use whereis to find the location of a command. It looks through standard program locations until it finds the requested command.</p>

Linux on Raspberry Pi

• Networking Commands

Networking Command	Use
ping	The ping utility is usually used to check if communication can be made with another host. It can be used with default settings by just specifying a hostname (e.g. ping raspberrypi.org) or an IP address (e.g. ping 8.8.8.8). It can specify the number of packets to send with the -c flag..
nmap	nmap is a network exploration and scanning tool. It can return port and OS information about a host or a range of hosts. Running just nmap will display the options available as well as example usage.
hostname	The hostname command displays the current hostname of the system. A privileged (super) user can set the hostname to a new one by supplying it as an argument (e.g. hostname new-host).
ipconfig	Use ifconfig to display the network configuration details for the interfaces on the current system when run without any arguments (i.e. ifconfig). By supplying the command with the name of an interface (e.g. eth0 or lo) you can then alter the configuration: check the manual page for more details.

Linux on Raspberry Pi

- **Text Editors on Desktop**

- **Text Editor**

- When using Raspberry Pi OS Desktop, in the accessories menu there is an option to run a Text Editor. This is a simple editor which opens in a window like a normal application. It allows use of the mouse and keyboard, and has tabs and syntax highlighting.

- You can use keyboard shortcuts, such as Ctrl + S to save a file and Ctrl + X to exit.

- **Thonny**

- [Thonny](#) is a Python REPL and IDE, so you can write and edit Python code in a window and run it directly from the editor. Thonny has independent windows, and syntax highlighting, and uses Python 3.

- **Geany**

- A fast and lightweight IDE, supporting many different file types, including C/C++ and Python. It is installed by default on Raspberry Pi OS.

Linux on Raspberry Pi

- **Text Editors in the Terminal**
- **Nano**
- GNU Nano is at the easy-to-use end of command-line editors. It's installed by default, so use `nano somefile.txt` to edit a file, and keyboard shortcuts like `Ctrl + O` to save and `Ctrl + X` to exit.
- **Vi**
- Vi is a very old (c. 1976) command-line editor, which is available on most UNIX systems and is pre-installed on Raspberry Pi OS. It's succeeded by Vim (Vi Improved), which requires installation.
- Unlike most editors, Vi and Vim have a number of different modes. When you open Vi with `vi somefile.txt`, you start in command mode which doesn't directly permit text entry. Press `i` to switch to insert mode in order to edit the file, and type away. To save the file you must return to command mode, so press the Escape key and enter `:w` (followed by Enter), which is the command to write the file to disk.
- To search for the word 'raspberry' in a file, make sure you're in command mode (press Escape), then type `/raspberry` followed by `n` and `N` to flick forwards/backwards through the results.
- To save and exit, enter the command `:wq`. To exit without saving, enter the command `:q!`.
- Depending on your keyboard configuration, you may find your cursor keys don't work. In this case, you can use the H-J-K-L keys (which move left, down, up, and right respectively) to navigate the file in command mode.

Linux on Raspberry Pi

- ***Vim***
- Vim is an extension of Vi and works in much the same way, with a number of improvements. Only Vi is installed by default so to get the full features of Vim, install it with APT:

```
sudo apt install vim
```

- You can edit a file in Vim with `vim somefile.txt`.
- ***Emacs***
- Emacs is a GNU command-line text editor; it's powerful, extensible, and customisable. You can install it with APT:

```
sudo apt install emacs
```

- You can use keyboard combination commands, such as `Ctrl + X Ctrl + S` to save and `Ctrl + X Ctrl + C` to close.

Linux on Raspberry Pi

- **Linux Users**
- User management in Raspberry Pi OS is done on the command line. The default user is pi, and the password is raspberry. You can add users and change each user's password.
- **Changing your Password**
- Once you're logged in as the pi user, it is highly advisable to use the passwd command to change the default password to improve your Pi's security.
- Enter passwd on the command line and press Enter. You'll be prompted to enter your current password to authenticate, and then asked for a new password. Press Enter on completion and you'll be asked to confirm it. Note that no characters will be displayed while entering your password. Once you've correctly confirmed your password, you'll be shown a success message (passwd: password updated successfully), and the new password will apply immediately.
- If your user has sudo permissions, you can change another user's password with passwd followed by the user's username. For example, sudo passwd bob will allow you to set the user bob's password, and then some additional optional values for the user such as their name. Just press Enter to skip each of these options.
- ***Remove a User's Password***
- You can remove the password for the user bob with sudo passwd bob -d. Without a password the user will not be able to login to a Terminal.

Linux on Raspberry Pi

- **Linux Users**
- **Creating a New User**
- You can create additional users on your Raspberry Pi OS installation with the `adduser` command.
- Enter `sudo adduser bob` and you'll be prompted for a password for the new user bob. Leave this blank if you don't want a password.
- **Your Home Folder**
- When you create a new user, they will have a home folder in `/home/`. The pi user's home folder is at `/home/pi/`.
- **The skel Command**
- Upon creating a new user, the contents of `/etc/skel/` will be copied to the new user's home folder. You can add or modify dot-files such as the `.bashrc` in `/etc/skel/` to your requirements, and this version will be applied to new users.
- **Deleting a User**
- You can delete a user on your system with the command `userdel`. Apply the `-r` flag to remove their home folder too:
- `sudo userdel -r bob`

Linux on Raspberry Pi

- **The `.bashrc` File**

- In your home folder you will find a hidden file called `.bashrc` which contains some user configuration options. You can edit this file to suit your needs. Changes made in this file will be actioned the next time a terminal is opened, since that is when the `.bashrc` file is read.

- **Shell Scripts**

- Commands can be combined together in a file which can then be executed. As an example, copy the following into your favourite text editor:

```
#!/usr/bin/bash
while :
do
Echo Raspberry Pi!
done
```

Save this with the name `fun-script`.

Linux on Raspberry Pi

- Before you can run it you must first make it executable; this can be done by using the change mode command `chmod`. Each file and directory has its own set of permissions that dictate what a user can and can't do to it. In this case, by running the command `chmod +x fun-script`, the file `fun-script` will now be executable.
- You can then run it by typing `./fun-script` (assuming that it's in your current directory).
- This script infinitely loops and prints Raspberry Pi!; to stop it, press `Ctrl + C`. This kills any command that's currently being run in the terminal.

Raspberry PI Interfaces

- It supports SPI, serial and I2C interfaces for data transfer.
- Serial : Serial Interface on Raspberry has receive(Rx) and Transmit(Tx) pins for communication with serial peripherals.
- SPI: Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices. In an SPI connection, there is one master device and one or more peripheral devices. There are 5 pins Raspberry for SPI interface.
 - MISO(Master In Slave Out): Master line for sending data to the peripherals.
 - MOSI(Master Out Slave In): Slave Line for sending data to the master.
 - SCK(Serial Clock): Clock generated by master to synchronize data transmission.
 - CE0(Chip Enable 0): To enable or disable devices.
 - o CE1(Chip Enable 1): To enable or disable devices.
- I2C: I2C Interface pins are used to connect hardware modules.

I2C interface allows synchronous data transfer with two pins: SDA(data line) and SCL (Clock Line)

Raspberry PI Interfaces

- **Features of Raspberry PPI**

- 1. Where the system processing is huge. They can process high end programs for applications like Weather Station, Cloud server, gaming console etc. With 1.2GHz clock speed and 1 GB RAM RASPBERRY PI can perform all those advanced functions.
- 2. RASPBERRY PI 3 has wireless LAN and Bluetooth facility by which you can setup WIFI HOTSPOT for internet connectivity.
- 3. RASPBERRY PI had dedicated port for connecting touch LCD display which is a feature that completely omits the need of monitor.
- 4. RASPBERRY PI also has dedicated camera port so one can connect camera without any hassle to the PI board.
- 5. RASPBERRY PI also has PWM outputs for application use.
- 6. It supports HD steaming

Raspberry PI Interfaces

- **Applications**

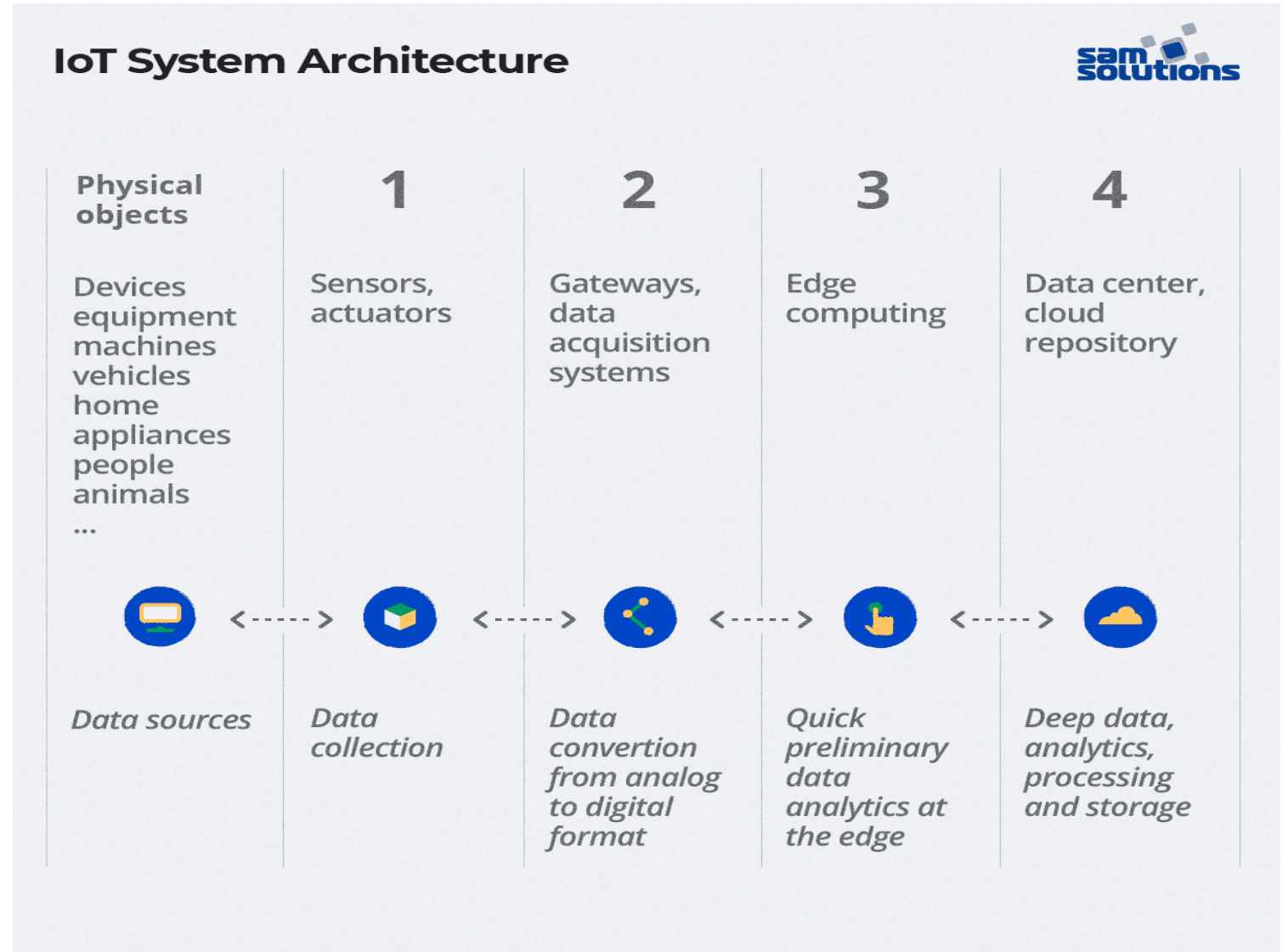
- Hobby projects.
- Low cost PC/tablet/laptop
- IoT applications
- Media center
- Robotics
- Industrial/Home automation
- Server/cloud server
- Print server
- Security monitoring
- Web camera
- Gaming
- Wireless access point

Programming Raspberry Pi with Python

- Separate PPt
- <https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/all>

Other IoT Platforms

- **The Internet of Things concept** implies the creation of a distributed network consisting of numerous physical objects equipped with embedded software, sensors and connectivity options that collect and share data with each other and with the central platform via the internet.



IoT Technology Overview

- **IoT system architecture** consists of four layers:
- *Sensors and actuators* collect data directly from physical objects (devices, equipment, machines, vehicles, home appliances, people, animals, etc.).
- *Gateways and data acquisition systems* convert gathered data from the analog to the digital format.
- *Edge computing* ensures there's immediate preliminary data analytics right on devices.
- *Data centers or cloud services* provide deep data analysis, processing and storage.

IoT Technology Overview

Examples of IoT systems:

- *Smart home systems* (security devices, intelligent lighting, conditioning, heating, connected home appliances)
- *Wearable health devices* both for self-tracking of health conditions (pulse oximeters, glucometers) and for vital sign monitoring in clinics
- *Logistics tracking systems* (GPS trackers, fuel level sensors, alert systems to monitor driver behavior)
- *Autonomous vehicles* (farming equipment, warehouse autonomous robots, passenger buses)
- *Smart factory equipment* (robotics, predictive maintenance solutions)

IoT Platform

- **What Is an IoT Platform?**
- **An IoT platform** serves as a mediator between the world of physical objects and the world of actionable insights. Combining numerous tools and functionalities, Internet of Things platforms enable you to build unique hardware and software products for collecting, storing, analyzing and managing the plethora of data generated by your connected devices and assets.
- **Types of Internet of Things Platforms**
- IoT products consist of numerous components:
 - Hardware
 - Software
 - Communication technologies
 - Central repository (cloud or local)
 - End-user applications

Types of IoT platforms

- To cover each aspect while developing an IoT product, there are several **types of IoT platforms**.
- **Hardware development platforms** provide physical development boards for creating IoT devices, including microcontrollers, microprocessors, Systems on Chip (SoC), Systems on Module (SoM).
- **App development platforms** serve as an integrated development environment (IDE) with tools and features for coding applications.
- **Connectivity platforms** provide communication technologies to connect physical objects with the data center (on-premise or cloud) and transmit information between them. Among popular connectivity protocols and standards for the Internet of Things are MQTT, DDS, AMQP, Bluetooth, ZigBee, WiFi, Cellular, LoRaWAN and more.
- **Analytics platforms** use intelligent algorithms to analyze collected information and transform it into actionable insights for customers.
- **End-to-end IoT platforms** cover all aspects of IoT products, from development and connectivity to data management and visualization.

Most Popular IoT Platforms in 2021

- To make it easier for you to decide which IoT platform to choose for your project, we've compiled a list of the most popular Internet of Things platforms for this year, with detailed descriptions of each one.
- Google Cloud IoT
- Cisco IoT Cloud Connect
- Salesforce IoT Cloud
- IRI Voracity
- Particle
- IBM Watson IoT
- ThingWorx
- Amazon AWS IoT Core
- Microsoft Azure IoT Hub
- Oracle IoT

IoT Platforms

1. Google Cloud IoT

- Google launched its platform for Internet of Things development on the basis of its end-to-end Google Cloud Platform. Currently, it's one of the world's top Internet of Things platforms. Google Cloud IoT is the integration of various services that add value to connected solutions.
- **Cloud IoT Core** allows you to capture and handle device data. A *device manager* component is used to register devices with the service, and monitor and configure them. *MQTT and HTTP protocol bridges* are used for device connection and communication with the Google Cloud Platform.
- **Cloud Pub/Sub** performs data ingestion and message routing for further data processing.
- **Google BigQuery** enables secure real-time data analytics.
- **AI Platform** applies machine learning features.
- **Google Data Studio** visualizes data by making reports and dashboards.
- **Google Maps Platform** helps visualize the location of connected assets.
- The platform automatically integrates with Internet of Things hardware producers such as Intel and Microchip. It supports various operating systems, including Debian Linux OS.



Google Cloud Platform

IoT Platforms

- **Core features of Google Cloud IoT:**
- AI and machine learning capabilities
- Real-time data analysis
- Strong data visualization
- Location tracking
- **Core use cases:**
- Predictive maintenance
- Real-time asset tracking
- Logistics and supply chain management
- Smart cities and buildings

IoT Platforms

2. Cisco IoT Cloud Platform

- Cisco IoT Cloud Connect is originally an offering for mobile operators. This mobility cloud-based software suite for industrial and individual use cases is on the list of the best Internet of Things cloud platforms. Cisco also provides reliable IoT hardware, including switches, access points, routers, gateways and more.
- Take a look at some examples of powerful Cisco Internet of Things products and solutions.
- **Cisco IoT Control Center** ensures impeccable cellular connectivity management, allowing you to integrate all your IoT devices in one SaaS solution.
- **Extended Enterprise Solution** allows for the development of IoT business applications at the edge and ensures rapid deployment and centralized network management.
- **Edge Intelligence** simplifies data processing by allocating data flows either to local or multi-cloud environments.
- **Industrial Asset Vision** utilizes sensors to monitor your assets continuously and deliver data for better decision-making.
- **Cisco IoT Threat Defense** protects sensible data and devices against cyberattacks, providing secure remote access, segmentation, visibility and analysis, and other security services.



IoT Platforms

- **Core features of Cisco IoT Cloud Connect:**
- Powerful industrial solutions
- High-level security
- Edge computing
- Centralized connectivity and data management
- **Core use cases:**
- Connected cars
- Fleet management
- Home security and automation
- Payment and POS solutions
- Predictive maintenance
- Industrial networking
- Smart meters
- Healthcare

IoT Platforms

3. Salesforce IoT Cloud

- Salesforce specializes in customer relations management and masterfully enhances this segment with the help of IoT solutions.
- The Salesforce IoT Cloud platform gathers valuable information from connected devices to deliver personalized experiences to and build stronger relationships with your customers. It works in tandem with Salesforce CRM: data from connected assets is delivered directly to the CRM system where context-based actions are initiated immediately.
- For example, if sensors detect an error in windmill performance, it is instantly reflected in the CRM dashboard and the system can either adjust parameters automatically or create a service ticket.



IoT Platforms

- **Core features of Salesforce IoT Cloud:**
- Full integration of customers, products and CRM
- No need for programming skills to create rules, conditions and events due to a simple point-and-click UI
- Compatibility with third-party websites, services and other products
- A proactive approach to customer issues and needs
- **Core use cases:**
- Government administration
- Machinery
- Financial services
- Marketing and advertising
- Chemicals
- By using Salesforce IoT Cloud, businesses get a holistic view of customer data, improve customer experience and increase sales.

IoT Platforms

4. IRI Voracity

- If you need an all-in-one data management platform that enables IoT data control at every stage of your business processes, IRI Voracity is the perfect fit.
- This platform uses two engines, IRI CoSort and Hadoop, to process Big Data. It can discover, govern, integrate, analyze, transform and migrate data from various sources and in various formats such as Unix, Linux or Windows file systems, ISAM, MongoDB, LDIF, HIVE, JSON, S3, PostgreSQL, MQTT, Kafka and more.



IoT Platforms

- **Core features of IRI Voracity:**
- **A Data Governance Portal** enables data search and classification in silos. It also provides encryption and anonymization to comply with data privacy regulations.
- **A Faster ETL and Analytic Alternative** performs extraction and transformation of large-sized data much faster than legacy ETL tools.
- **A DB Ops Environment** allows you to administer all your databases from one place.
- **Core use cases:**
 - Big Data analytics
 - ETL modernization
 - Data governance

IoT Platforms

5. Particle

- Particle offers an IoT edge-to-cloud platform for global connectivity and device management, as well as hardware solutions, including development kits, production modules and asset tracking devices. With Particle's team of IoT experts, who provide end-to-end professional services, you can develop your product from concept to production.
- **Core features of the Particle platform:**
 - Integration with third-party services via REST API
 - Firewall-protected cloud
 - Capability to work with data from Google Cloud or Microsoft Azure
 - No need for technical expertise in order to use the platform
- **Core use cases:**
 - Real-time asset monitoring
 - Live vehicle tracking
 - Predictive maintenance
 - Environmental monitoring
 - Compliance monitoring
 - Real-time order fulfillment



IoT Platforms

6. IBM Watson IoT

- An IoT platform built on IBM Cloud is a fully managed cloud service for device management, flexible and scalable connectivity options, secure communications and data lifecycle management. With IBM Watson IoT, you can collect insights from automobiles, buildings, equipment, assets and things.
- **Core features of IBM Watson IoT:**
 - Data ingestion from any source with the help of MQTT
 - Direct access to the latest data in the Cloudant NoSQL DB solution
 - Built-in monitoring dashboards to control your assets
 - Analytics Service to process raw metrics
 - The Cloud Object Storage solution for long-term data archiving
- **Core use cases:**
 - Supply chain management
 - Regulatory compliance
 - Building management
 - Energy consumption
 - Shipping and logistics

IBM **Watson IoT**[™]

IoT Platforms

7. ThingWorx

- The specialized Industrial Internet of Things (IIoT) platform ThingWorx is used in a variety of manufacturing, service and engineering scenarios. The platform addresses common challenges across industries, from remote monitoring and maintenance to workforce efficiency and asset optimization.
- **Core features of ThingWorx:**
 - Access to multiple data sources due to the extension of traditional industrial communications
 - Powerful ready-to-use tools and applications to create and scale IIoT solutions quickly
 - Real-time insights from complex industrial IoT data to proactively optimize operations and prevent issues
 - Total control over network devices, processes and systems
- **Core Use Cases:**
 - Remote asset monitoring
 - Remote maintenance/service
 - Predictive maintenance and asset management
 - Optimized equipment effectiveness



IoT Platforms

8. Amazon AWS IoT Core

- One of the leading players in the market, Amazon AWS IoT Core allows you to connect devices to AWS cloud services without the need to manage servers. The platform provides reliability and security for managing millions of devices.
- **Core features of Amazon AWS IoT Core:**
 - A wide choice of connection protocols, including MQTT, MQTT over WSS, HTTP and LoRaWAN
 - Ability to use with other AWS services such as AWS Lambda, Amazon Kinesis, Amazon DynamoDB, Amazon CloudWatch, Alexa Voice Service and more to build IoT applications
 - A high level of security provided by end-to-end encryption throughout all points of connection, automated configuration and authentication
 - Machine learning capabilities
 - A variety of services for edge computing
- **Core use cases:**
 - Connected vehicles
 - Connected homes
 - Asset tracking
 - Smart building
 - Industrial IoT



IoT Platforms

9. Microsoft Azure IoT Hub

- With the open-source Azure IoT platform from Microsoft, you can quickly build scalable and secure edge-to-cloud solutions. Utilizing ready-to-use tools, templates and services, you can develop flexible applications according to your company's needs.

Core features of Azure IoT Hub:

- Data protection all the way from the edge to the cloud
- The ability to operate even in offline mode with Azure IoT Edge
- Seamless integration with other Azure services
- Enhanced AI solutions
- Continuous cloud-scale analytics
- Fully managed databases
- Azure Industrial IoT solution



- **Core use cases:**
- Automotive industry
- Discrete manufacturing
- Energy sector
- Healthcare
- Transportation
- Retail

IoT Platforms

10. Oracle IoT

- The Internet of Things Cloud Service by Oracle is a managed Platform as a Service (PaaS) for connecting your devices to the cloud.
- **Core features of Oracle IoT:**
 - The ability to create applications and connect them to devices with JavaScript, [Java](#), Android, iOS, C POSIX and REST APIs
 - Integration with enterprise applications, web services and other Oracle Cloud Services
 - Real-time analysis tools to aggregate and filter incoming data streams
 - Automatic synchronization of data streams with Oracle Business Intelligence Cloud Service
 - Unique digital identity for each device to establish trust relationships among devices and applications
- **Core use cases:**
 - Connected logistics
 - Predictive maintenance
 - Smart manufacturing
 - Workplace safety

IoT Platforms

- **How to Choose the Best IoT Platform?**
- There's no definite answer to this question since there's no one best platform suitable for any digital project. The choice will always depend on the specific requirements of your business.
- Large enterprises are more likely to turn to giants such as Amazon or Microsoft. Their offerings are the best established, but also the most expensive. Smaller companies may find more cost-efficient options that will nevertheless perfectly meet their requirements.
- When choosing a provider, you should consider the technical capabilities of a platform, its partner ecosystem, industry-specific features and, in general, the provider's reputation. All these parameters should comply with your company strategy and budget.

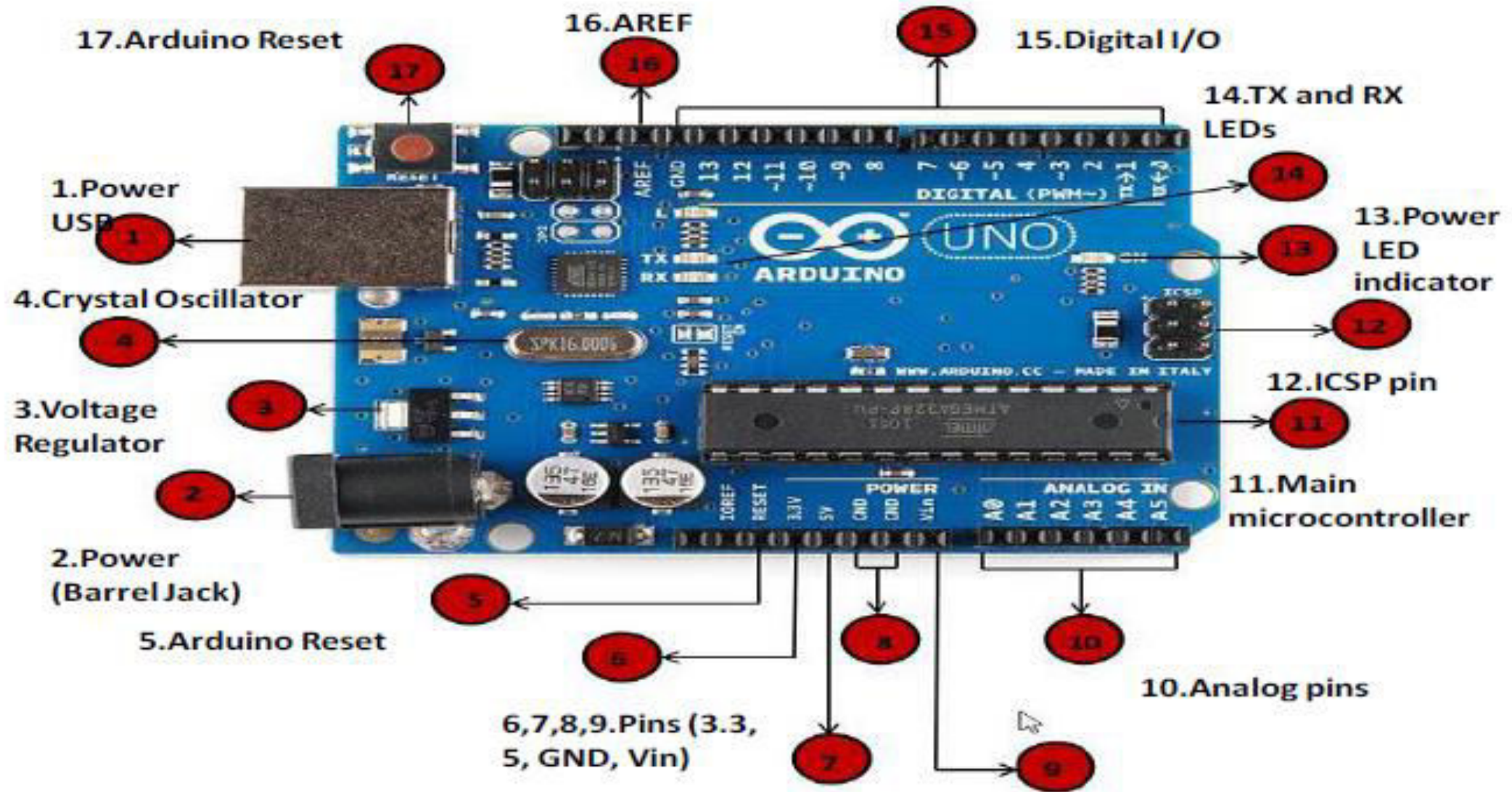
Arduino

- Building IOT with Arduino Internet of Things
- The Internet of Things (IoT) is a scenario in which objects, animals or people are provided with single identifiers and the capability to automatically transfer and the capability to automatically transfer data more to a network without requiring human-to-human or human-to-computer communication.

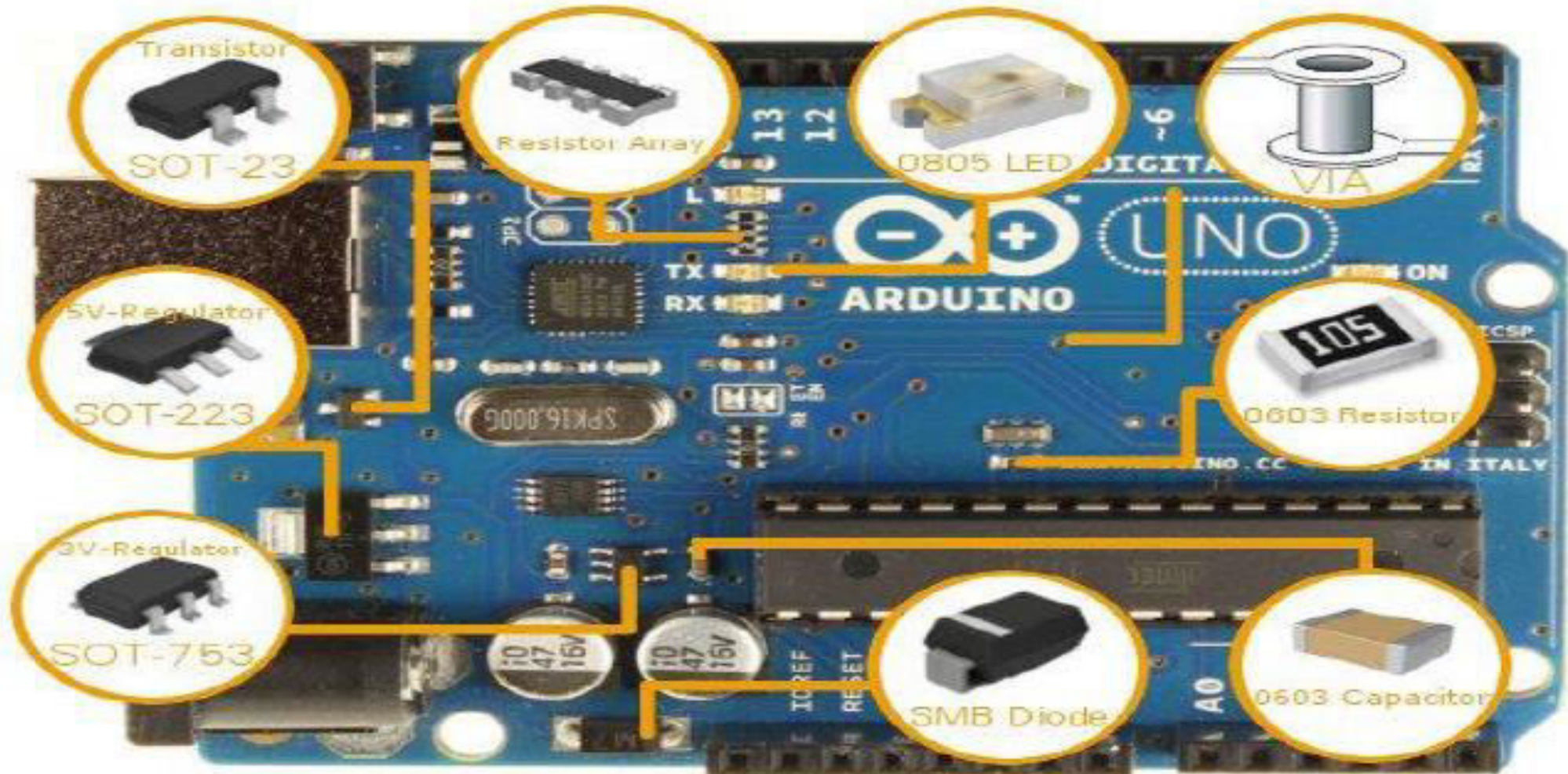
Arduino

- **Arduino Board:**
- An Arduino is actually a microcontroller based kit.
- It is basically used in communications and in controlling or operating many devices.
- Arduino UNO board is the most popular board in the Arduino board family.
- In addition, it is the best board to get started with electronics and coding.
- Some boards look a bit different from the one given below, but most Arduino's have majority of these components in common.
- It consists of two memories- Program memory and the data memory.
- The code is stored in the flash program memory, whereas the data is stored in the data memory.
- Arduino Uno consists of 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button

Arduino



Arduino



Arduino

- **1.Power USB** Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).
- **2.Power (Barrel Jack)** Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).
- **3.Voltage Regulator** The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.
- **4.Crystal Oscillator** The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.
- **5,17.Arduino Reset** You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).
 - 6,7,8,9.Pins (3.3, 5, GND, Vin)
 - 3.3V (6) – Supply 3.3 output volt
 - 5V (7) – Supply 5 output volt
 - Most of the components used with Arduino board works fine with 3.3 volt and 5 volt.
 - GND (8)(Ground) – There are several GND pins on the Arduino, any of which can be used to ground your circuit.
 - Vin (9) – This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.

Arduino

- **10.Analog pins** The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.
- **11.Main microcontroller** Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

Arduino

- **12.ICSP pin** Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the mas
- **13.Power LED indicator** This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection. ter of the SPI bus.

Arduino

- **14.TX and RX LEDs** On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.
- **15.Digital I/O**
 - The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.

Arduino

- 16.AREF
- AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Program an Arduino

- The most important advantage with Arduino is the programs can be directly loaded to the device without requiring any hardware programmer to burn the program.
- This is done because of the presence of the 0.5KB of Bootloader which allows the program to be burned into the circuit.
- All we have to do is to download the Arduino software and writing the code.
- The Arduino tool window consists of the toolbar with the buttons like verify, upload, new, open, save, serial monitor.
- It also consists of a text editor to write the code, a message area which displays the feedback like showing the errors, the text console which displays the output and a series of menus like the File, Edit, Tools menu.

Arduino

- **Steps to program an Arduino**
- Programs written in Arduino are known as sketches. A basic sketch consists of 3 parts
 - 1. Declaration of Variables
 - 2. Initialization: It is written in the setup () function.
 - 3. Control code: It is written in the loop () function.
- The sketch is saved with .ino extension. Any operations like verifying, opening a sketch, saving a sketch can be done using the buttons on the toolbar or using the tool menu.
- The sketch should be stored in the sketchbook directory.
- Chose the proper board from the tools menu and the serial port numbers.
- Click on the upload button or chose upload from the tools menu. Thus the code is uploaded by the bootloader onto the microcontroller.

Arduino

- **Basic Arduino functions are:**

- `digitalRead(pin)`: Reads the digital value at the given pin.
- `digitalWrite(pin, value)`: Writes the digital value to the given pin.
- `pinMode(pin, mode)`: Sets the pin to input or output mode.
- `analogRead(pin)`: Reads and returns the value.
- `analogWrite(pin, value)`: Write
- `serial.begin(baud rate)`: Sets the beginning of serial communication by setting the bit rate.s the value to that pin.

Arduino

- **Design your own Arduino**
- The following components are needed to design Arduino Board- A breadboard, a led, a power jack, a IC socket, a microcontroller, few resistors, 2 regulators, 2 capacitors.
 - The IC socket and the power jack are mounted on the board.
 - Add the 5v and 3.3v regulator circuits using the combinations of regulators and capacitors.
 - Add proper power connections to the microcontroller pins.
 - Connect the reset pin of the IC socket to a 10K resistor.
 - Connect the crystal oscillators to pins 9 and 10
 - Connect the led to the appropriate pin.
 - Mount the female headers onto the board and connect them to the respective pins on the chip.
 - Mount the row of 6 male headers, which can be used as an alternative to upload programs.
 - Upload the program on the Microcontroller of the readymade Arduino and then pry it off and place back on the user kit.

Arduino

- **Advantages of Arduino Board**

- 1. It is inexpensive
- 2. It comes with an open source hardware feature which enables users to develop their own kit using already available one as a reference source.
- 3. The Arduino software is compatible with all types of operating systems like Windows, Linux, and Macintosh etc.
- 4. It also comes with open source software feature which enables experienced software developers to use the Arduino code to merge with the existing programming language libraries and can be extended and modified.
- 5. It is easy to use for beginners.
- 6. We can develop an Arduino based project which can be completely stand alone or projects which involve direct communication with the software loaded in the computer.
- 7. It comes with an easy provision of connecting with the CPU of the computer using serial communication over USB as it contains built in power and reset circuitry.

Arduino

- **Interfaces UART Peripheral:**

- A UART (Universal Asynchronous Receiver/Transmitter) is a serial interface.
- It has only one UART module.
- The pins (RX, TX) of the UART are connected to a USB-to-UART converter circuit and also connected to pin0 and pin1 in the digital header.

- **SPI Peripheral:**

- The SPI (Serial Peripheral Interface) is another serial interface. It has only one SPI module.

- **TWI:**

- The I2C or Two Wire Interface is an interface consisting of only two wires, serial data, and a serial clock: SDA, SCL.
- You can reach these pins from the last two pins in the digital header or pin4 and pin5 in the analog header.

IoT Platform

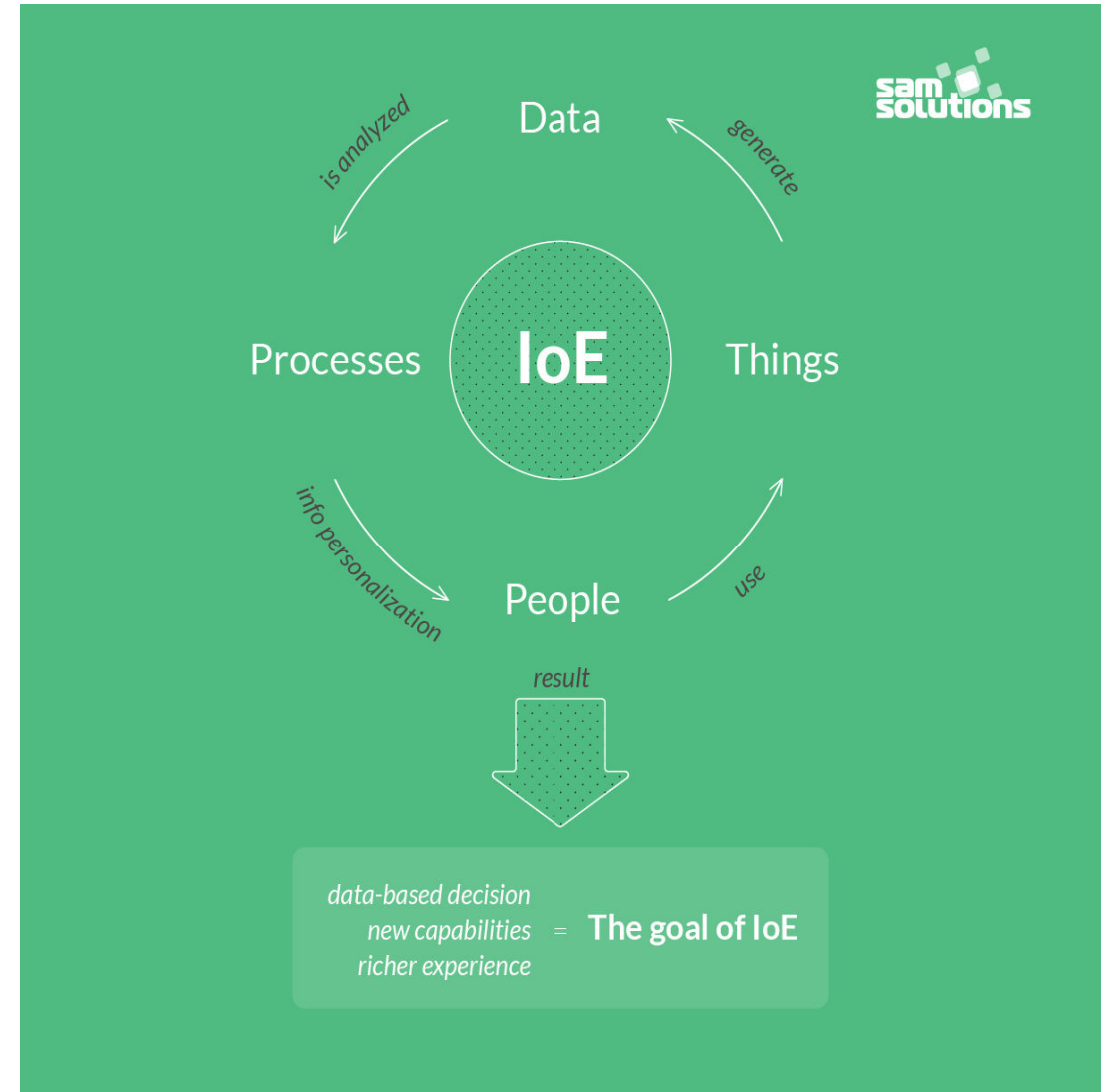
- **What Are the Top IoT App Development Platforms?**
- Currently, there are more than 600 publicly known Internet of Things platforms globally. However, the leaders — Amazon AWS IoT Core, Microsoft Azure IoT Hub and IBM Watson — still hold their positions as the top three Internet of Things app development platforms.
- **Why do I need an IoT platform?**
- An IoT platform is a unique tool that will provide continuous monitoring of all your assets, be it vehicles, manufacturing equipment, livestock, or anything else. It will help you as the owner of a business gain a comprehensive view of all processes seasoned with intelligent analytics of collected data. The result — quicker decisions, reduced issues and increased revenues.
- **What Is the Difference between IoT and Cloud Computing?**
- IoT is about gathering data from physical devices and transferring it to digital space for further analysis. Cloud computing is purely about data processing, delivery and storage. These are two different technologies that complement each other, resulting in efficient solutions.

Evolution of IOE and its benefits

- **What Is the Internet of Everything (IoE)?**
- The evolution of the global web has resulted in virtual connections ubiquitously penetrating real-world objects and activities. Today, everything can be connected with everything, creating a new distributed ecosystem that goes beyond the familiar [IoT \(Internet of Things\)](#) concept. Cisco has coined a special term — **the Internet of Everything (IoE)** — to describe this dynamically changing phenomenon. In this article, we will formulate the Internet of Everything definition and how it differs from the IoT.

Evolution of IOE and its benefits

- **Internet for Everything: Definition and Main Features**
- The IoE concept is based on the idea of all-round connectivity, intelligence and cognition. It means that intelligent internet connections are not restricted by computers, tablets and smartphones (as used to be the case for the last couple of decades). Any object can be outfitted with digital features and connected to the common network of other objects, people and processes in order to generate valuable information, exchange it and facilitate relevant decision-making.



Evolution of IOE and its benefits

- **What Is IoE?**
- The Internet of Everything is the connections between people, things, data and processes combined into a common interrelated system, the aim of which is to improve experiences and make smarter decisions.
- The IoE philosophy depicts the world in which billions of sensors are implanted into billions of devices, machines and ordinary objects, giving them expanded networking opportunities, thus making them smarter.
- **What does IoE mean for businesses, governments and individuals?**
The main goal of the IoE technology is to convert collected information into actions, facilitate data-based decision-making and provide new capabilities and richer experiences.

Evolution of IOE and its benefits

- **IoE Features**
- **Decentralization and moving to the edge** — data is processed not in a single center, but in numerous distributed nodes
- **Data input and output** — external data can be put into devices and given back to other components of the network
- **Relation to every technology in the process of digital transformation** — cloud computing, fog computing, AI, ML, IoT, Big Data, etc. Actually, a rise in Big Data and the IoE technology development are interconnected.

Evolution of IOE and its benefits

- **IoE Constituent Elements**

- Key components of the IoE market are hardware, software and services. As for constituent elements of the Internet of Everything, there are four of them:

- **People**

- People provide their personal insights via websites, applications or connected devices they use (such as social networks, healthcare sensors and fitness trackers); AI algorithms and other smart technologies analyze this data to “understand” human issues and deliver relevant content according to their personal or business needs that helps them quickly solve issues or make decisions.

- **Things**

- Here we encounter the pure IoT concept. Various physical items embedded with sensors and actuators generate data on their status and send it to the needed destination across the network.

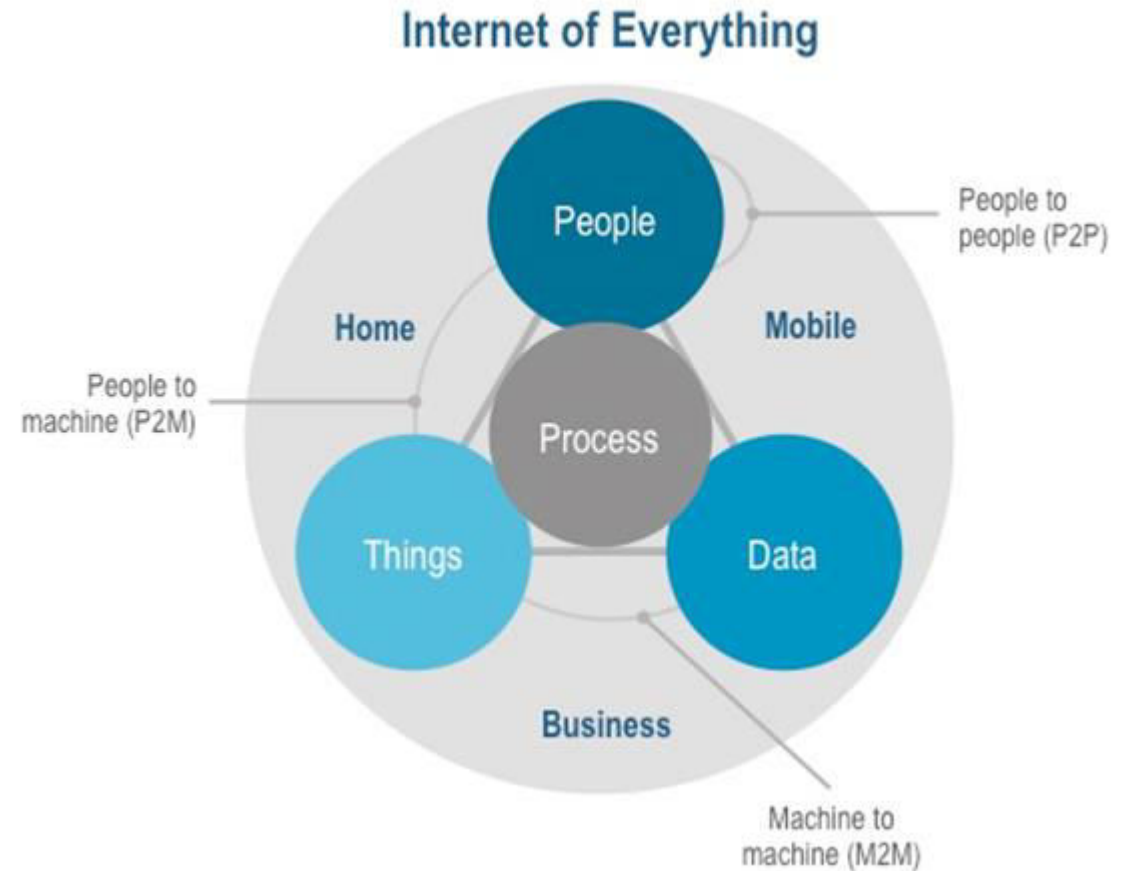
Evolution of IOE and its benefits

- **Data**

- The raw data generated by devices has no value. But once it is summarized, classified and analyzed, it turns into priceless information that can control various systems and empower intelligent solutions.

- **Processes**

- Different processes based on artificial intelligence, machine learning, social networks or other technologies ensure that the right information is sent to the right person at the right time. The goal of processes is to guarantee the best possible usage of Big Data.



Evolution of IOE and its benefits

- **Internet of Everything vs. Internet of Things**

- To avoid the confusion between the terms, IoT vs. IoE, let's figure out in what ways they differ.
- The core **difference** between the Internet of Things and the Internet of Everything is the number of pillars for these concepts:
- IoT focuses on physical objects only
- IoE encompasses four components (things, processes, data and people)
- The IoT, in essence, is the interconnectivity of physical objects that send and receive data, while the IoE is a wider term that includes, apart from IoT, numerous technologies and people as the end-nodes.

Evolution of IOE and its benefits

- Although IoT and IoE are different terms, there are also some **similarities** between them:
- **Decentralization** — both systems are distributed and don't have a single center; each node works as a small management center and is able to perform certain tasks independently
- **Security issues** — distributed systems are still highly vulnerable to penetration and cyberattacks; the more devices are connected to the network, the higher the susceptibility to breaches
- On the one hand, decentralization is one of the IoE and IoT advantages, since the whole system doesn't fail even if there are problems in a couple of nodes. On the other has, such a distribution causes disadvantages in the form of threats for data security and personal privacy.

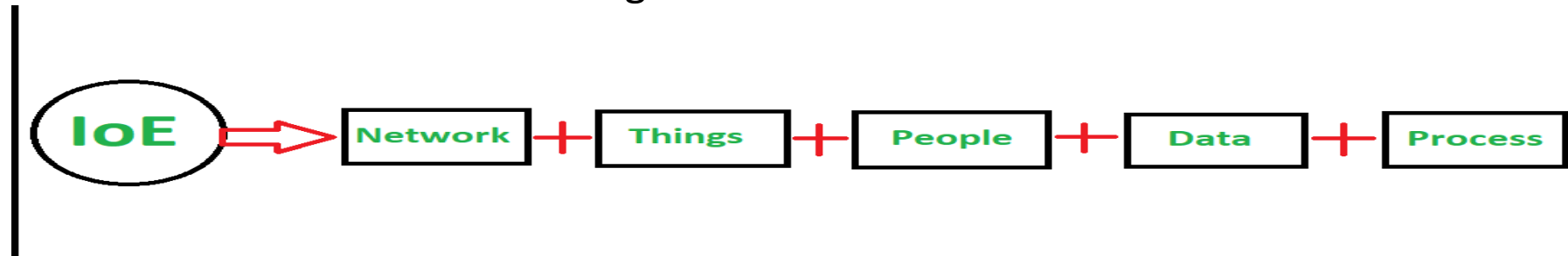
Evolution of IOE and its benefits

- **Internet of Everything Examples**

- Practically every industry can apply the Internet of Everything model into its processes and benefit from it. Here are some general examples:
- Municipality systems can implement smart water and electricity meters for residents and commercial organizations in order to monitor usage rates and make decisions concerning economy and cutting costs.
- The manufacturing industry can implement sensors for predictive maintenance into production to monitor equipment parts that need to be fixed or replaced. This helps eliminate downtime and reduce the fixing costs.
- Logistics and delivery companies can [introduce sensors and smart devices on trucks](#) to optimize delivery conditions and possible routing. Eventually, companies can improve end-user satisfaction.

IoT Vs IoE

- Internet of Everything (IoE) :
IoE is the intelligent connection between 4 key elements i.e people, process, data, and things. It is considered as superset of Internet of Things (IoT). IoE covers the wider concept of connectivity where network intelligence works as the foundation of Internet of Things. Internet of Everything acts as an extension of Internet of Things.



- Internet of Things (IoT) :
IoT is a network of interconnected physical devices/objects which collect and exchange data over wireless networks. Internet of Things has two main parts i.e 'Internet' which is the backbone of connectivity and 'Things' meaning to object/physical devices. It brings the power of the internet, data processing and analytics and decision making to the real world of physical objects. as an extension of Internet of Things.



IoT Vs IoE

Sl No.	Internet of Everything (IoE)	Internet of Things (IoT)
1.	The term IoE is coined by CISCO.	The term IoT coined by Kevin Ashton in 1999 during his work at Procter & Gamble.
2.	IoE is the intelligent connection between people, process, data and things by creating 'web of things' which is the next generation of internet.	IoT is the network of physical devices where collection and exchange of data occurs without human intervention.
3.	The goal of IoE is turning information into actions, providing data based decision making and provide new capabilities and richer experiences.	The goal of IoT is to form an ecosystem of connected objects/physical devices. Or to create an ecosystem connecting from Thing to Thing.
4.	In IoE, communication occurs between Machine to Machine, Machine to People and technology assisted People to People.	In IoT, communication occurs between Machine to Machine.
5.	It is more complex than IoT as IoE includes IoD (Internet of Digital), IoH (Internet of Human) and IoT (Internet of Things).	It is less complex than IoE as IoT(Internet of Things) is considered as a part of Bigger IoE ecosystem.
6.	It has four pillars people, process, data, and things.	It has one pillar things i.e it focuses on physical objects only.
7.	It is considered as the superset for Internet of Things(IoT), along with IoH, IoD, communication technologies and the internet itself and it is considered a generation after IoT.	It is considered as the subset of bigger Internet of Everything(IoE) and IoT is considered one generation before IoE.
8.	Example are Connecting roads with hospitals to save more lives, Connecting homes for comfort living, Connecting food and people in the supply chain, Elderly care monitoring.	Examples are Wearable health monitors, Connected appliances, Autonomous farming equipment, smarter energy management systems, Smart surveillance.

References

- <https://www.elprocus.com/building-the-internet-of-things-using-raspberry-pi/>
- <https://realpython.com/python-raspberry-pi/>
- <https://www.sam-solutions.com/blog/top-iot-platforms/>
- <https://www.sam-solutions.com/blog/what-is-internet-of-everything-ioe/>
- <https://www.geeksforgeeks.org/difference-between-ioe-and-iot/>
- https://www.raspberrypi.org/documentation/computers/using_linux.html

Chapter 6

IoT Systems – Logical Design using Python

INTERNET OF THINGS

A Hands-On Approach



Outline

- Introduction to Python
- Installing Python
- Python Data Types & Data Structures
- Control Flow
- Functions
- Modules
- Packages
- File Input/Output
- Date/Time Operations
- Classes

Python

- Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.
- The main characteristics of Python are:
 - Multi-paradigm programming language
 - Python supports more than one programming paradigms including object-oriented programming and structured programming
 - Interpreted Language
 - Python is an interpreted language and does not require an explicit compilation step. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.
 - Interactive Language
 - Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

Python - Benefits

- Easy-to-learn, read and maintain
 - Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages. Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.
- Object and Procedure Oriented
 - Python supports both procedure-oriented programming and object-oriented programming. Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.
- Extendable
 - Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.
- Scalable
 - Due to the minimalistic nature of Python, it provides a manageable structure for large programs.
- Portable
 - Since Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source
- Broad Library Support
 - Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.

Python - Setup

- Windows

- Python binaries for Windows can be downloaded from <http://www.python.org/getit> .
- For the examples and exercise in this book, you would require Python 2.7 which can be directly downloaded from: <http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi>
- Once the python binary is installed you can run the python shell at the command prompt using
 > python

- Linux

```
#Install Dependencies
sudo apt-get install build-essential
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev

#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5

#Install Python
./configure
make
sudo make install
```

Numbers

- Numbers
 - Number data type is used to store numeric values. Numbers are immutable data types, therefore changing the value of a number data type results in a newly allocated object.

```
#Integer
>>>a=5
>>>type(a)
<type 'int'>

#Floating Point
>>>b=2.5
>>>type(b)
<type 'float'>

#Long
>>>x=9898878787676L
>>>type(x)
<type 'long'>

#Complex
>>>y=2+5j
>>>y
(2+5j)
>>>type(y)
<type 'complex'>
>>>y.real
2
>>>y.imag
5
```

```
#Addition
>>>c=a+b
>>>c
7.5
>>>type(c)
<type 'float'>

#Subtraction
>>>d=a-b
>>>d
2.5
>>>type(d)
<type 'float'>
```

```
#Multiplication
>>>e=a*b
>>>e
12.5
>>>type(e)
<type 'float'>
```

```
#Division
>>>f=b/a
>>>f
0.5
>>>type(f)
<type float'>

#Power
>>>g=a**2
>>>g
25
```


Strings

- Strings

- A string is simply a list of characters in order. There are no limits to the number of characters you can have in a string.

#Create string

```
>>>s="Hello World!"
>>>type(s)
<type 'str'>
```

#String concatenation

```
>>>t="This is sample program."
>>>r = s+t
>>>r
'Hello World!This is sample program.'
```

#Get length of string

```
>>>len(s)
12
```

#Convert string to integer

```
>>>x="100"
>>>type(s)
<type 'str'>
>>>y=int(x)
>>>y
100
```

#Print string

```
>>>print s
Hello World!
```

#Formatting output

```
>>>print "The string (The string (Hello World!)
has 12 characters"
```

#Convert to upper/lower case

```
>>>s.upper()
'HELLO WORLD!'
>>>s.lower()
'hello world!'
```

#Accessing sub-strings

```
>>>s[0]
'H'
>>>s[6:]
'World!'
>>>s[6:-1]
'World'
```

#strip: Returns a copy of the string with the #leading and trailing characters removed.

```
>>>s.strip("!")
'Hello World'
```

Lists

- Lists
 - List a compound data type used to group together other values. List items need not all have the same type. A list contains items separated by commas and enclosed within square brackets.

#Create List

```
>>>fruits=['apple','orange','banana','mango']
>>>type(fruits)
<type 'list'>
```

#Get Length of List

```
>>>len(fruits)
4
```

#Access List Elements

```
>>>fruits[1]
'orange'
>>>fruits[1:3]
['orange', 'banana']
>>>fruits[1:]
['orange', 'banana', 'mango']
```

#Appending an item to a list

```
>>>fruits.append('pear')
>>>fruits
['apple', 'orange', 'banana', 'mango', 'pear']
```

#Removing an item from a list

```
>>>fruits.remove('mango')
>>>fruits
['apple', 'orange', 'banana', 'pear']
```

#Inserting an item to a list

```
>>>fruits.insert(1,'mango')
>>>fruits
['apple', 'mango', 'orange', 'banana', 'pear']
```

#Combining lists

```
>>>vegetables=['potato','carrot','onion','beans','radish']
>>>vegetables
['potato', 'carrot', 'onion', 'beans', 'radish']

>>>eatables=fruits+vegetables
>>>eatables
['apple', 'mango', 'orange', 'banana', 'pear', 'potato', 'carrot', 'onion', 'beans', 'radish']
```

#Mixed data types in a list

```
>>>mixed=['data',5,100.1,8287398L]
>>>type(mixed)
<type 'list'>
>>>type(mixed[0])
<type 'str'>
>>>type(mixed[1])
<type 'int'>
>>>type(mixed[2])
<type 'float'>
>>>type(mixed[3])
<type 'long'>
```

#Change individual elements of a list

```
>>>mixed[0]=mixed[0]+" items"
>>>mixed[1]=mixed[1]+1
>>>mixed[2]=mixed[2]+0.05
>>>mixed
['data items', 6, 100.14999999999999, 8287398L]
```

#Lists can be nested

```
>>>nested=[fruits,vegetables]
>>>nested
[['apple', 'mango', 'orange', 'banana', 'pear'],
 ['potato', 'carrot', 'onion', 'beans', 'radish']]
```

Tuples

- Tuples

- A tuple is a sequence data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed within parentheses. Unlike lists, the elements of tuples cannot be changed, so tuples can be thought of as read-only lists.

#Create a Tuple

```
>>>fruits=("apple","mango","banana","pineapple")
>>>fruits
('apple','mango','banana','pineapple')
```

```
>>>type(fruits)
<type 'tuple'>
```

#Get length of tuple

```
>>>len(fruits)
4
```

#Get an element from a tuple

```
>>>fruits[0]
'apple'
>>>fruits[:2]
('apple','mango')
```

#Combining tuples

```
>>>vegetables=('potato','carrot','onion','radish')
>>>eatables=fruits+vegetables
>>>eatables
('apple','mango','banana','pineapple','potato','carrot','onion','radish')
```

Dictionaries

- Dictionaries

- Dictionary is a mapping data type or a kind of hash table that maps keys to values. Keys in a dictionary can be of any data type, though numbers and strings are commonly used for keys. Values in a dictionary can be any data type or object.

#Create a dictionary

```
>>>student={'name':'Mary','id':'8776','major':'CS'}
>>>student
{'major':'CS', 'name':'Mary', 'id':'8776'}
>>>type(student)
<type 'dict'>
```

#Get length of a dictionary

```
>>>len(student)
3
```

#Get the value of a key in dictionary

```
>>>student['name']
'Mary'
```

#Get all items in a dictionary

```
>>>student.items()
[('gender', 'female'), ('major', 'CS'), ('name', 'Mary'), ('id', '8776')]
```

#Get all keys in a dictionary

```
>>>student.keys()
['gender', 'major', 'name', 'id']
```

#Get all values in a dictionary

```
>>>student.values()
['female', 'CS', 'Mary', '8776']
```

#Add new key-value pair

```
>>>student['gender']='female'
>>>student
{'gender': 'female', 'major': 'CS', 'name': 'Mary', 'id': '8776'}
```

#A value in a dictionary can be another dictionary

```
>>>student1={'name':'David','id':'9876','major':'ECE'}
>>>students={'1': student,'2':student1}
>>>students
{'1': {'gender': 'female', 'major': 'CS', 'name': 'Mary', 'id': '8776'}, '2': {'major': 'ECE', 'name': 'David', 'id': '9876'}}
```

#Check if dictionary has a key

```
>>>student.has_key('name')
True
>>>student.has_key('grade')
False
```

Type Conversions

- Type conversion examples

#Convert to string

```
>>>a=10000
```

```
>>>str(a)
```

```
'10000'
```

#Convert to int

```
>>>b="2013"
```

```
>>>int(b)
```

```
2013
```

#Convert to float

```
>>>float(b)
```

```
2013.0
```

#Convert to long

```
>>>long(b)
```

```
2013L
```

#Convert to list

```
>>>s="aeiou"
```

```
>>>list(s)
```

```
['a', 'e', 'i', 'o', 'u']
```

#Convert to set

```
>>>x=['mango','apple','banana','mango','banana']
```

```
>>>set(x)
```

```
set(['mango', 'apple', 'banana'])
```

Control Flow – if statement

- The *if* statement in Python is similar to the *if* statement in other languages.

```
>>>a = 25**5
>>>if a>10000:
        print "More"
    else:
        print "Less"

More
```

```
>>>if a>10000:
    if a<1000000:
        print "Between 10k and 100k"
    else:
        print "More than 100k"
    elif a==10000:
        print "Equal to 10k"
    else:
        print "Less than 10k"

More than 100k
```

```
>>>s="Hello World"
>>>if "World" in s:
        s=s+"!"
    print s

Hello World!
```

```
>>>student={'name':'Mary','id':'8776'}
>>>if not student.has_key('major'):
        student['major']='CS'

>>>student
{'major':'CS', 'name':'Mary', 'id':'8776'}
```

Control Flow – for statement

- The *for* statement in Python iterates over items of any sequence (list, string, etc.) in the order in which they appear in the sequence.
- This behavior is different from the *for* statement in other languages such as C in which an initialization, incrementing and stopping criteria are provided.

#Looping over characters in a string

```
helloString = "Hello World"

for c in helloString:
    print c
```

#Looping over items in a list

```
fruits=['apple','orange','banana','mango']

i=0
for item in fruits:
    print "Fruit-%d: %s" % (i,item)
    i=i+1
```

#Looping over keys in a dictionary

```
student
=
'nam
e':
'Mar
y', 'id': '8776', 'gender': 'female', 'major': 'CS'

for key in student:
    print "%s: %s" % (key,student[key])
```

Control Flow – while statement

- The while statement in Python executes the statements within the while loop as long as the while condition is true.

```
#Prints even numbers upto 100
```

```
>>> i = 0
```

```
>>> while i<=100:
```

```
    if i%2 == 0:
```

```
        print i
```

```
    i = i+1
```


Control Flow – range statement

- The range statement in Python generates a list of numbers in arithmetic progression.

#Generate a list of numbers from 0 – 9

```
>>>range (10)  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

#Generate a list of numbers from 10 - 100 with increments of 10

```
>>>range(10,110,10)  
[10, 20, 30, 40, 50, 60, 70, 80, 90,100]
```

Control Flow – break/continue statements

- The *break* and *continue* statements in Python are similar to the statements in C.

- Break

- Break statement breaks out of the for/while loop

#Break statement example

```
>>>y=1
>>>for x in range(4,256,4):
    y = y * x
    if y > 512:
        break
    print y

4
32
384
```

- Continue

- Continue statement continues with the next iteration.

#Continue statement example

```
>>>fruits=['apple','orange','banana','mango']
>>>for item in fruits:
    if item == "banana":
        continue
    else:
        print item

apple
orange
mango
```

Control Flow – pass statement

- The *pass* statement in Python is a null operation.
- The *pass* statement is used when a statement is required syntactically but you do not want any command or code to execute.

```
>fruits=['apple','orange','banana','mango']
>for item in fruits:
    if item == "banana":
        pass
    else:
        print item

apple
orange
mango
```

Functions

- A function is a block of code that takes information in (in the form of parameters), does some computation, and returns a new piece of information based on the parameter information.
- A function in Python is a block of code that begins with the keyword *def* followed by the function name and parentheses. The function parameters are enclosed within the parenthesis.
- The code block within a function begins after a colon that comes after the parenthesis enclosing the parameters.
- The first statement of the function body can optionally be a documentation string or docstring.

```
students = {'1': {'name': 'Bob', 'grade': 2.5},
            '2': {'name': 'Mary', 'grade': 3.5},
            '3': {'name': 'David', 'grade': 4.2},
            '4': {'name': 'John', 'grade': 4.1},
            '5': {'name': 'Alex', 'grade': 3.8}}
```

```
def averageGrade(students):
    "This function computes the average grade"
    sum = 0.0
    for key in students:
        sum = sum + students[key]['grade']
    average = sum/len(students)
    return average
```

```
avg = averageGrade(students)
print "The average grade is: %0.2f" % (avg)
```

Functions - Default Arguments

- Functions can have default values of the parameters.
- If a function with default values is called with fewer parameters or without any parameter, the default values of the parameters are used

```
>>>def displayFruits(fruits=['apple','orange']):  
    print "There are %d fruits in the list" % (len(fruits))  
    for item in fruits:  
        print item  
  
#Using default arguments  
>>>displayFruits()  
apple  
orange  
  
>>>fruits = ['banana','pear','mango']  
>>>displayFruits(fruits)  
banana  
pear  
mango
```

Functions - Passing by Reference

- All parameters in the Python functions are passed by reference.
- If a parameter is changed within a function the change also reflected back in the calling function.

```
>>>def displayFruits(fruits):  
    print "There are %d fruits in the list" % (len(fruits))  
    for item in fruits:  
        print item  
    print "Adding one more fruit"  
    fruits.append('mango')  
  
>>>fruits = ['banana', 'pear', 'apple']  
>>>displayFruits(fruits)  
There are 3 fruits in the list  
banana  
pear  
apple  
  
#Adding one more fruit  
>>>print "There are %d fruits in the list" % (len(fruits))  
There are 4 fruits in the list
```

Functions - Keyword Arguments

- Functions can also be called using keyword arguments that identifies the arguments by the parameter name when the function is called.

```
>>>def
printStudentRecords(name,age=20,major='CS'):
    print "Name: " + name
    print "Age: " + str(age)
    print "Major: " + major

#This will give error as name is required argument
>>>printStudentRecords()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: printStudentRecords() takes at least 1
argument (0 given)
```

```
#Correct use
>>>printStudentRecords(name='Alex')
Name: Alex
Age: 20
Major: CS

>>>printStudentRecords(name='Bob',age=22,major='ECE')
Name: Bob
Age: 22
Major: ECE

>>>printStudentRecords(name='Alan',major='ECE')
Name: Alan
Age: 20
Major: ECE
```

```
#name is a formal argument.
**kwargs is a keyword argument that receives all
arguments except the formal argument as a
dictionary.

>>>def student(name, **kwargs):
    print "Student Name: " + name
    for key in kwargs:
        print key + ': ' + kwargs[key]

>>>student(name='Bob', age='20', major = 'CS')
Student Name: Bob
age: 20
major: CS
```

Functions - Variable Length Arguments

- Python functions can have variable length arguments. The variable length arguments are passed to as a tuple to the function with an argument prefixed with asterix (*)

```
>>>def student(name, *varargs):  
    print "Student Name: " + name  
    for item in varargs:  
        print item  
  
>>>student('Nav')  
Student Name: Nav  
  
>>>student('Amy', 'Age: 24')  
Student Name: Amy  
Age: 24  
  
>>>student('Bob', 'Age: 20', 'Major: CS')  
Student Name: Bob  
Age: 20  
Major: CS
```


Modules

- Python allows organizing the program code into different modules which improves the code readability and management.
- A module is a Python file that defines some functionality in the form of functions or classes.
- Modules can be imported using the import keyword.
- Modules to be imported must be present in the search path.

```
#student module - saved as student.py
def averageGrade(students):
    sum = 0.0
    for key in students:
        sum = sum + students[key]['grade']
    average = sum/len(students)
    return average

def printRecords(students):
    print "There are %d students" %(len(students))
    i=1
    for key in students:
        print "Student-%d: " % (i)
        print "Name: " + students[key]['name']
        print "Grade: " + str(students[key]['grade'])
        i = i+1
```

```
# Importing a specific function from a module
>>>from student import averageGrade

# Listing all names defines in a module
>>>dir(student)
```

```
#Using student module
>>>import student
>>>students = {'1': {'name': 'Bob', 'grade': 2.5},
'2': {'name': 'Mary', 'grade': 3.5},
'3': {'name': 'David', 'grade': 4.2},
'4': {'name': 'John', 'grade': 4.1},
'5': {'name': 'Alex', 'grade': 3.8}}

>>>student.printRecords(students)
There are 5 students
Student-1:
Name: Bob
Grade: 2.5
Student-2:
Name: David
Grade: 4.2
Student-3:
Name: Mary
Grade: 3.5
Student-4:
Name: Alex
Grade: 3.8
Student-5:
Name: John
Grade: 4.1

>>>avg = student.averageGrade(students)
>>>print "The average grade is: %0.2f" % (avg)
3.62
```

Packages

- Python package is hierarchical file structure that consists of modules and subpackages.
- Packages allow better organization of modules related to a single application environment.

```
# skimage package listing

skimage/      Top level package
  __init__.py Treat directory as a package

  color/ color color subpackage
    __init__.py
    colorconv.py
    colorlabel.py
    rgb_colors.py

  draw/ draw draw subpackage
    __init__.py
    draw.py
    setup.py

  exposure/ exposure subpackage
    __init__.py
    _adapthist.py
    exposure.py

  feature/ feature subpackage
    __init__.py
    _brief.py
    _daisy.py

...
```

File Handling

- Python allows reading and writing to files using the file object.
- The `open(filename, mode)` function is used to get a file object.
- The mode can be read (r), write (w), append (a), read and write (r+ or w+), read-binary (rb), write-binary (wb), etc.
- After the file contents have been read the close function is called which closes the file object.

Example of reading an entire file

```
>>>fp = open('file.txt','r')
>>>content = fp.read()
>>>print content
This is a test file.
>>>fp.close()
```

Example of reading line by line

```
>>>fp = open('file1.txt','r')
>>>print "Line-1: " + fp.readline()
Line-1: Python supports more than one programming paradigms.
>>>print "Line-2: " + fp.readline()
Line-2: Python is an interpreted language.
>>>fp.close()
```

Example of reading lines in a loop

```
>>>fp = open('file1.txt','r')
>>>lines = fp.readlines()
>>>for line in lines:
    print line
```

Python supports more than one programming paradigms.
Python is an interpreted language.

File Handling

Example of reading a certain number of bytes

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>fp.close()
```

Example of seeking to a certain position

```
>>>fp = open('file.txt','r')
>>>fp.seek(10,0)
>>>content = fp.read(10)
>>>print content
ports more
>>>fp.close()
```

Example of getting the current position of read

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>currentpos = fp.tell()
>>>print currentpos
<built-in method tell of file object at 0x0000000002391390>
>>>fp.close()
```

Example of writing to a file

```
>>>fo = open('file1.txt','w')
>>>content='This is an example of writing to a file in
Python.'
>>>fo.write(content)
>>>fo.close()
```

Date/Time Operations

- Python provides several functions for date and time access and conversions.
- The datetime module allows manipulating date and time in several ways.
- The time module in Python provides various time-related functions.

Examples of manipulating with date

```
>>>from datetime import date

>>>now = date.today()
>>>print "Date: " + now.strftime("%m-%d-%y")
Date: 07-24-13

>>>print "Day of Week: " + now.strftime("%A")
Day of Week: Wednesday

>>>print "Month: " + now.strftime("%B")
Month: July

>>>then = date(2013, 6, 7)
>>>timediff = now - then
>>>timediff.days
47
```

Examples of manipulating with time

```
>>>import time
>>>nowtime = time.time()
>>>time.localtime(nowtime)
time.struct_time(tm_year=2013, tm_mon=7, tm_mday=24, tm_ec=51, tm_wday=2, tm_yday=205,
tm_isdst=0)

>>>time.asctime(time.localtime(nowtime))
'Wed Jul 24 16:14:51 2013'

>>>time.strftime("The date is %d-%m-%y. Today is a %A. It is %H hours, %M minutes and %S seconds now.")
'The date is 24-07-13. Today is a Wednesday. It is 16 hours, 15 minutes and 14 seconds now.'
```

Classes

- Python is an Object-Oriented Programming (OOP) language. Python provides all the standard features of Object Oriented Programming such as classes, class variables, class methods, inheritance, function overloading, and operator overloading.
- Class
 - A class is simply a representation of a type of object and user-defined prototype for an object that is composed of three things: a name, attributes, and operations/methods.
- Instance/Object
 - Object is an instance of the data structure defined by a class.
- Inheritance
 - Inheritance is the process of forming a new class from an existing class or base class.
- Function overloading
 - Function overloading is a form of polymorphism that allows a function to have different meanings, depending on its context.
- Operator overloading
 - Operator overloading is a form of polymorphism that allows assignment of more than one function to a particular operator.
- Function overriding
 - Function overriding allows a child class to provide a specific implementation of a function that is already provided by the base class. Child class implementation of the overridden function has the same name, parameters and return type as the function in the base class.

Class Example

- The variable *studentCount* is a class variable that is shared by all instances of the class *Student* and is accessed by *Student.studentCount*.
- The variables *name*, *id* and *grades* are instance variables which are specific to each instance of the class.
- There is a special method by the name `__init__()` which is the class constructor.
- The class constructor initializes a new instance when it is created. The function `__del__()` is the class destructor

```
# Examples of a class
class Student:
    studentCount = 0

    def __init__(self, name, id):
        print "Constructor called"
        self.name = name
        self.id = id
        Student.studentCount = Student.studentCount + 1
        self.grades={}

    def __del__(self):
        print "Destructor called"

    def getStudentCount(self):
        return Student.studentCount

    def addGrade(self,key,value):
        self.grades[key]=value
    def getGrade(self,key):
        return self.grades[key]

    def printGrades(self):
        for key in self.grades:
            print key + ": " + self.grades[key]
```

```
>>>s = Student('Steve','98928')
Constructor called

>>>s.addGrade('Math','90')
>>>s.addGrade('Physics','85')
>>>s.printGrades()
Physics: 85
Math: 90

>>>mathgrade = s.getGrade('Math')
>>>print mathgrade
90

>>>count = s.getStudentCount()
>>>print count
1

>>>del s
Destructor called
```

Class Inheritance

- In this example Shape is the base class and Circle is the derived class. The class Circle inherits the attributes of the Shape class.
- The child class Circle overrides the methods and attributes of the base class (eg. draw() function defined in the base class Shape is overridden in child class Circle).

Examples of class inheritance

class Shape:

```
def __init__(self):
    print "Base class constructor"
    self.color = 'Green'
    self.lineWeight = 10.0

def draw(self):
    print "Draw - to be implemented"
    def setColor(self, c):
        self.color = c
    def getColor(self):
        return self.color

def setLineWeight(self, lwt):
    self.lineWeight = lwt

def getLineWeight(self):
    return self.lineWeight
```

class Circle(Shape):

```
def __init__(self, c,r):
    print "Child class constructor"
    self.center = c
    self.radius = r
    self.color = 'Green'
    self.lineWeight = 10.0
    self.__label = 'Hidden circle label'

def setCenter(self,c):
    self.center = c
    def getCenter(self):
        return self.center

def setRadius(self,r):
    self.radius = r

def getRadius(self):
    return self.radius

def draw(self):
    print "Draw Circle (overridden function)"
```

class Point:

```
def __init__(self, x, y):
    self.xCoordinate = x
    self.yCoordinate = y

def setXCoordinate(self,x):
    self.xCoordinate = x

def getXCoordinate(self):
    return self.xCoordinate

def setYCoordinate(self,y):
    self.yCoordinate = y

def getYCoordinate(self):
    return self.yCoordinate
```

```
>>>p = Point(2,4)
>>>circ = Circle(p,7)
Child class constructor
>>>circ.getColor()
'Green'
>>>circ.setColor('Red')
>>>circ.getColor()
'Red'
>>>circ.getLineWeight()
10.0
>>>circ.getCenter().getXCoordinate()
2
>>>circ.getCenter().getYCoordinate()
4
>>>circ.draw()
Draw Circle (overridden function)
>>>circ.radius
7
```


Further Reading

- Code Academy Python Tutorial, <http://www.codecademy.com/tracks/python>
- Google's Python Class, <https://developers.google.com/edu/python/>
- Python Quick Reference Cheat Sheet, <http://www.addedbytes.com/cheat-sheets/python-cheat-sheet/>
- PyCharm Python IDE, <http://www.jetbrains.com/pycharm/>

Chapter 7

IoT Physical Devices & Endpoints

INTERNET OF THINGS

A Hands-On Approach



Outline

- Basic building blocks of an IoT Device
- Exemplary Device: Raspberry Pi
- Raspberry Pi interfaces
- Programming Raspberry Pi with Python
- Other IoT devices

What is an IoT Device

- A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smart TV, computer, refrigerator, car, etc.).
- IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely.

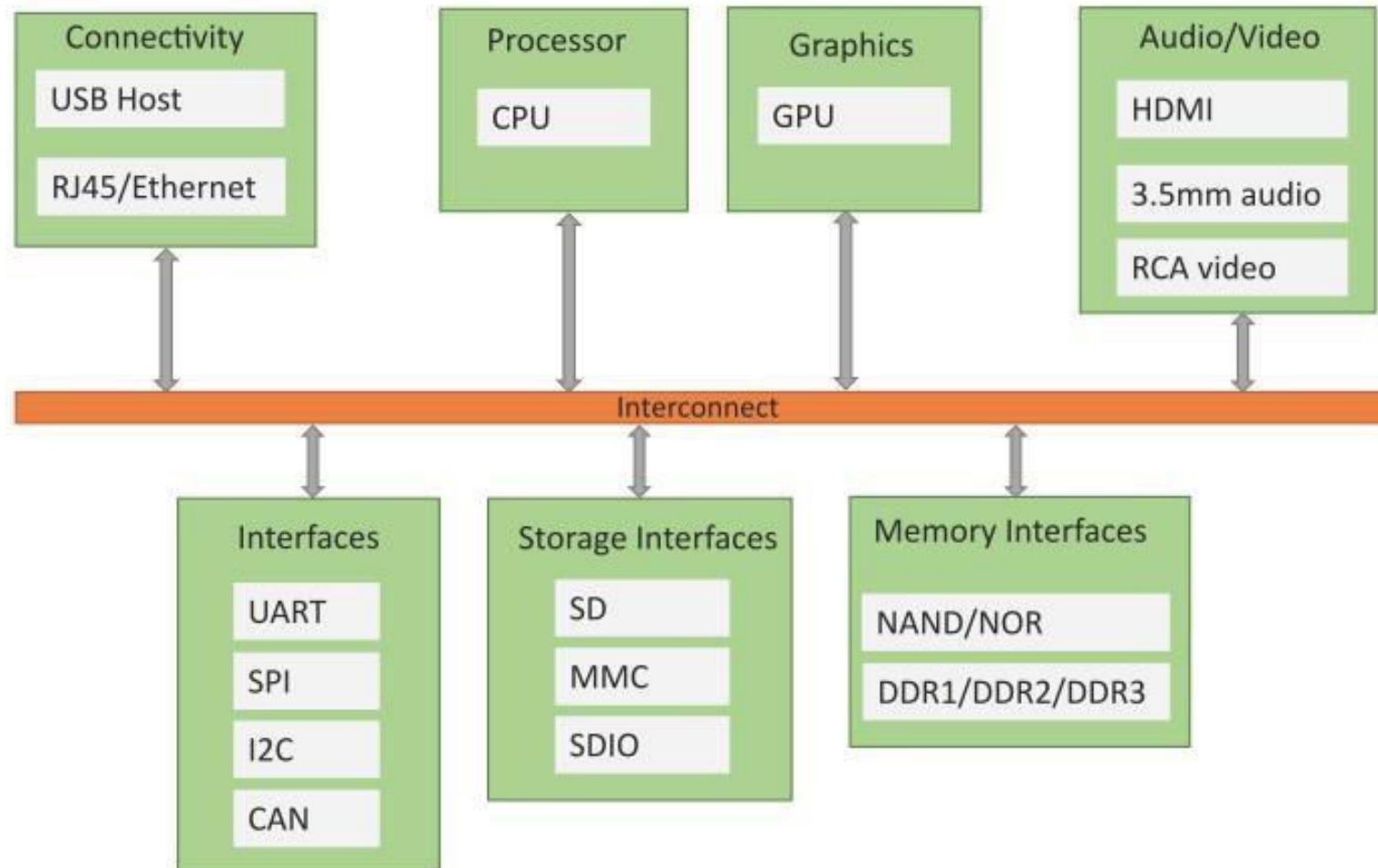
IoT Device Examples

- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information about its operation and health monitoring data to a server.
- A car which sends information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

Basic building blocks of an IoT Device

- Sensing
 - Sensors can be either on-board the IoT device or attached to the device.
- Actuation
 - IoT devices can have various types of actuators attached that allow taking
 - actions upon the physical entities in the vicinity of the device.
- Communication
 - Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications.
- Analysis & Processing
 - Analysis and processing modules are responsible for making sense of the collected data.

Block diagram of an IoT Device



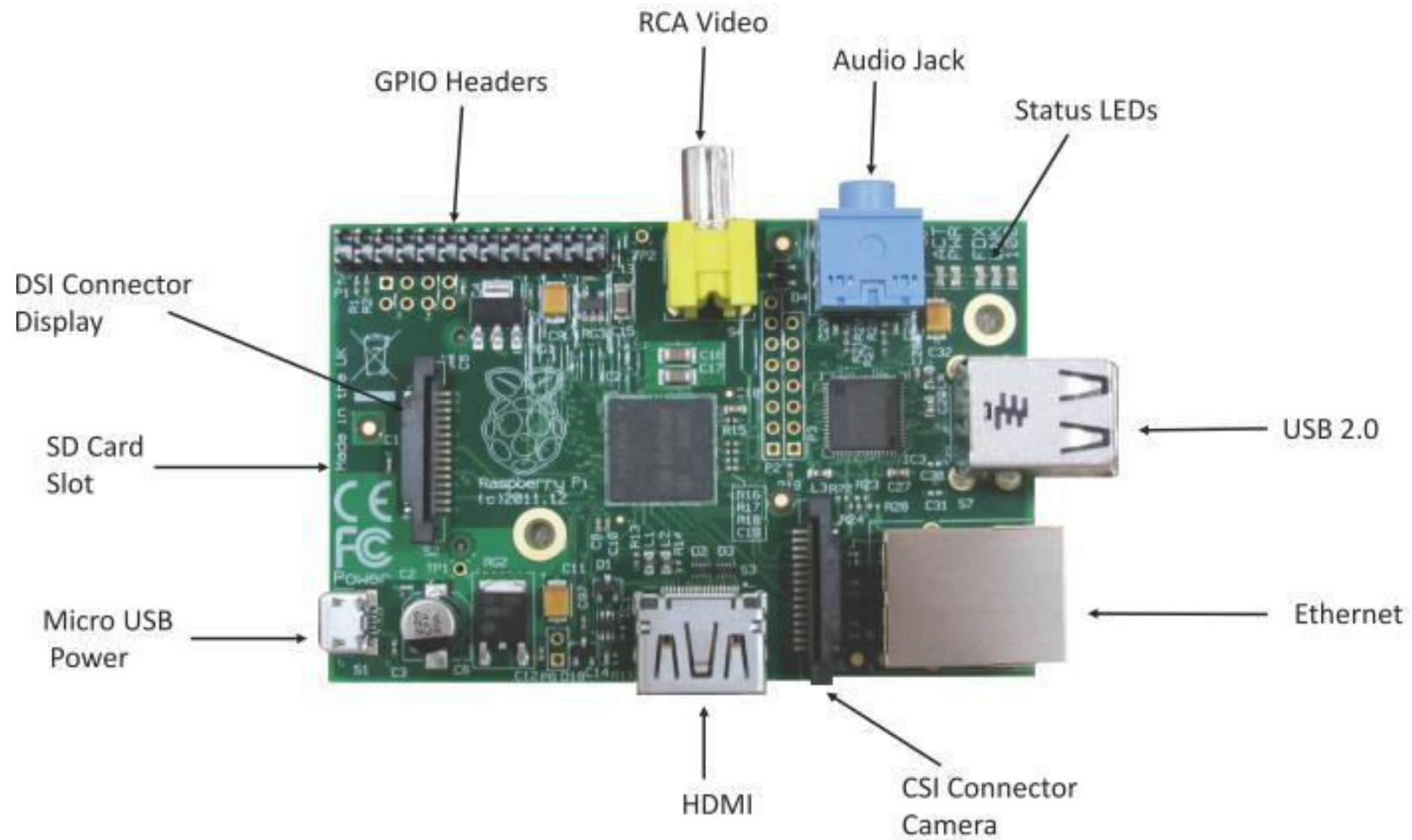
Exemplary Device: Raspberry Pi

- Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.
- Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.
- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins.
- Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".

Exemplary Device: Raspberry Pi

- Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.
- Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.
- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins.
- Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".

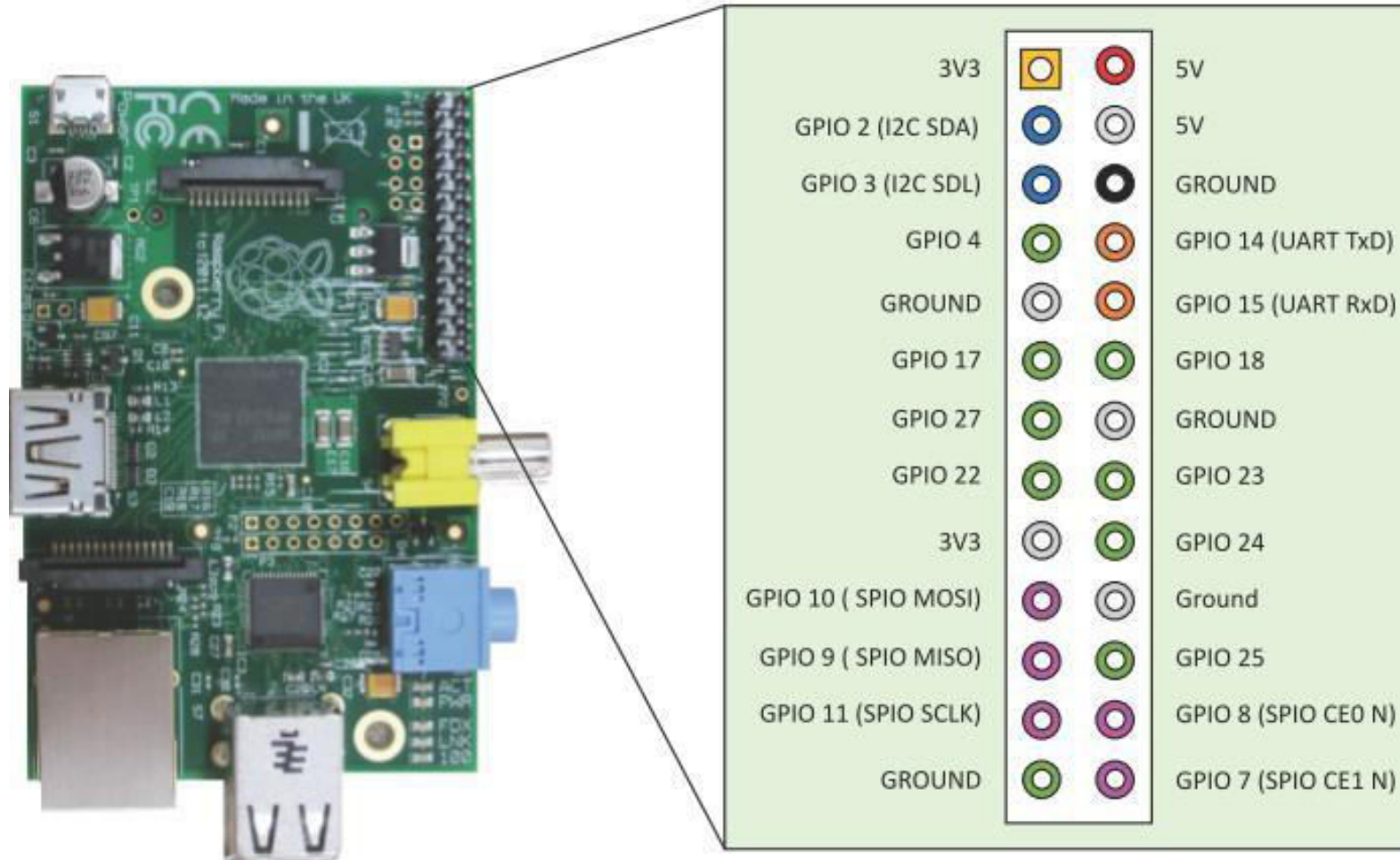
Raspberry Pi



Linux on Raspberry Pi

- Raspbian
 - Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.
- Arch
 - Arch is an Arch Linux port for AMD devices.
- Pidora
 - Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
- RaspBMC
 - RaspBMC is an XBMC media-center distribution for Raspberry Pi.
- OpenELEC
 - OpenELEC is a fast and user-friendly XBMC media-center distribution.
- RISC OS
 - RISC OS is a very fast and compact operating system.

Raspberry Pi GPIO



Raspberry Pi Interfaces

- Serial
 - The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.
- SPI
 - Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.
- I2C
 - The I2C interface pins on Raspberry Pi allow you to connect hardware modules. I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

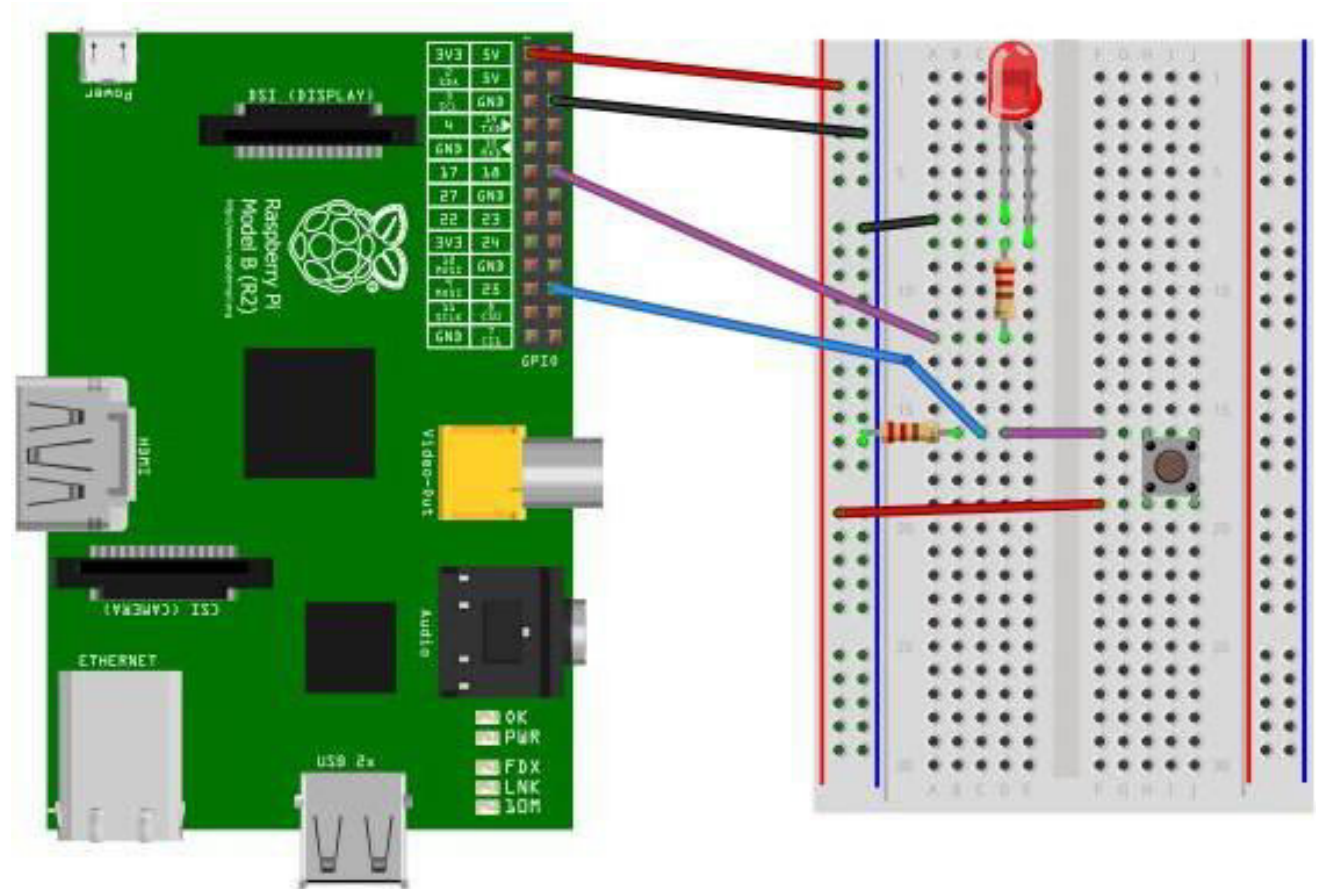
Raspberry Pi Example: Interfacing LED and switch with Raspberry Pi

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

#Switch Pin
GPIO.setup(25, GPIO.IN)
#LED Pin
GPIO.setup(18, GPIO.OUT)
state=False

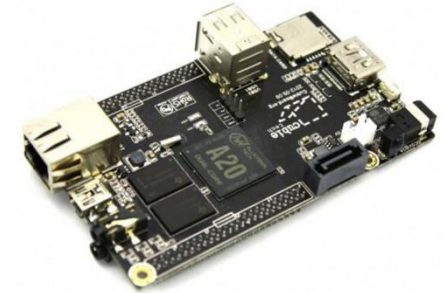
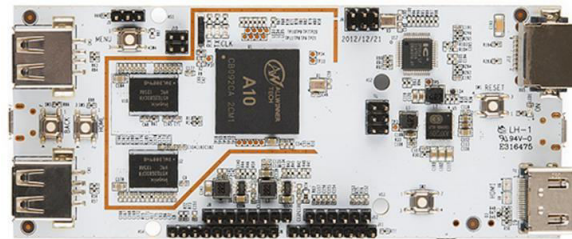
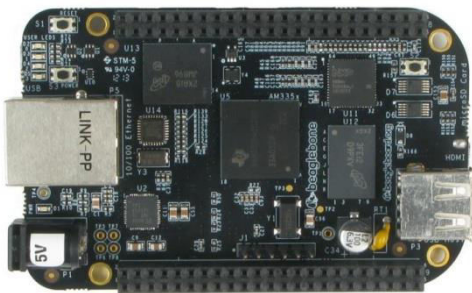
def toggleLED(pin):
    state = not state
    GPIO.output(pin, state)

while True:
    try:
        if (GPIO.input(25) == True):
            toggleLED(pin)
            sleep(.01)
    except KeyboardInterrupt:
        exit()
```



Other Devices

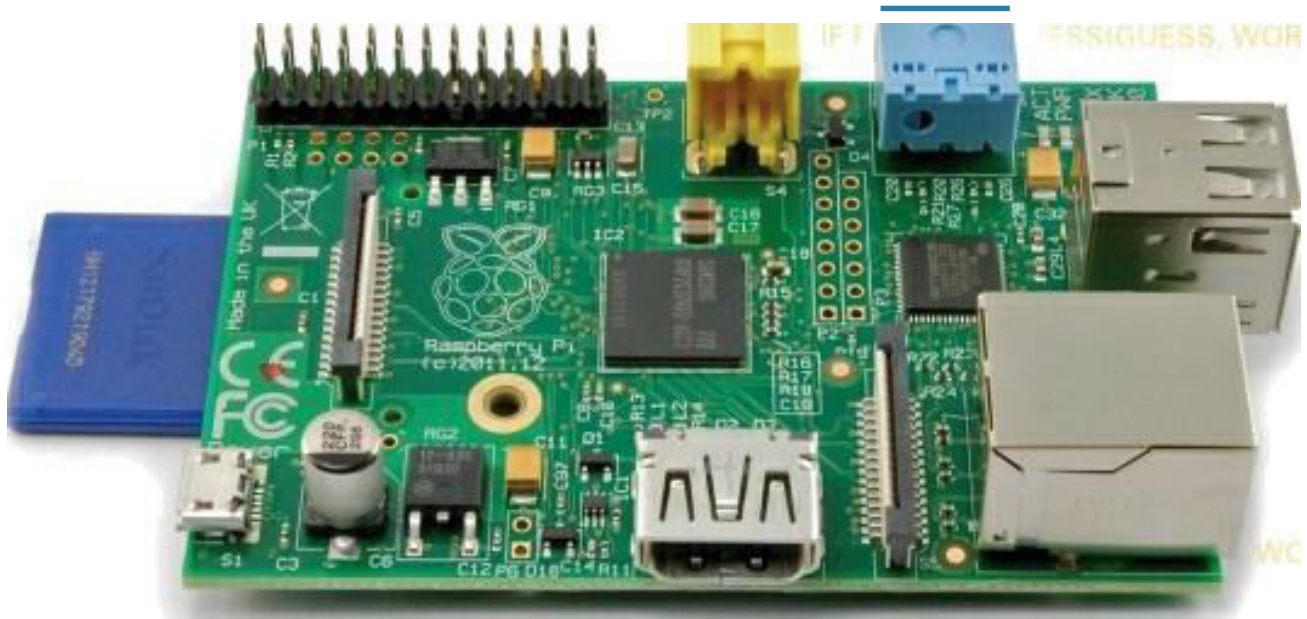
- pcDuino
- BeagleBone Black
- Cubieboard



TAB

Programming the Raspberry PiTM

Getting Started with Python



Simon Monk

About the Author

Dr. Simon Monk (Preston, UK) has a degree in cybernetics and computer science and a Ph.D. in software engineering. Simon spent several years as an academic before he returned to the industry, co-founding the mobile software company Momote Ltd. Simon is now a full-time author and has published three books in the McGraw-Hill *Evil Genius* series. He is also the author of *Programming Arduino* and has published books on IOIO and .NET Gadgeteer. You can follow Simon on Twitter @simonmonk2.

Programming the Raspberry Pi™

Getting Started with Python

Simon Monk



New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

Copyright © 2013 by The McGraw-Hill Companies. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

ISBN: 978-0-07-180784-5

MHID: 0-07-180784-5

The material in this eBook also appears in the print version of this title: ISBN: 978-0-07-180783-8, MHID: 0-07-180783-7

McGraw-Hill eBooks are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. To contact a representative please e-mail us at bulksales@mcgraw-hill.com.

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

Information has been obtained by McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, McGraw-Hill, or others, McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

TERMS OF USE

This is a copyrighted work and The McGraw-Hill Companies, Inc. ("McGrawHill") and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without McGraw-Hill's prior consent. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED "AS IS." MCGRAW-HILL AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

To my brothers, Andrew and Tim Monk, for their love and wisdom.

1	<u>Introduction</u>
2	<u>Getting Started</u>
3	<u>Python Basics</u>
4	<u>Strings, Lists, and Dictionaries</u>
5	<u>Modules, Classes, and Methods</u>
6	<u>Files and the Internet</u>
7	<u>Graphical User Interfaces</u>
8	<u>Games Programming</u>
9	<u>Interfacing Hardware</u>
10	<u>Prototyping Project (Clock)</u>
11	<u>The RaspiRobot</u>
12	<u>What Next</u>
	<u>Index</u>

[Acknowledgments](#)

[Introduction](#)

1 [Introduction](#)

[What Is the Raspberry Pi?](#)

[What Can You Do with a Raspberry Pi?](#)

[A Tour of the Raspberry Pi](#)

[Setting Up Your Raspberry Pi](#)

[Buying What You Need](#)

[Connecting Everything Together](#)

[Booting Up](#)

[Summary](#)

2 [Getting Started](#)

[Linux](#)

[The Desktop](#)

[The Internet](#)

[The Command Line](#)

[Navigating with the Terminal](#)

[sudo](#)

[Applications](#)

[Internet Resources](#)

[Summary](#)

3 [Python Basics](#)

[IDLE](#)

[Python Versions](#)

[Python Shell](#)

[Editor](#)

[Numbers](#)

[Variables](#)

[For Loops](#)

[Simulating Dice](#)

[If](#)

[Comparisons](#)

[Being Logical](#)

[Else](#)

[While](#)

[Summary](#)

4 [Strings, Lists, and Dictionaries](#)

[String Theory](#)

[Lists](#)

[Functions](#)

[Hangman](#)

[Dictionaries](#)

[Tuples](#)

[Multiple Assignment](#)

[Multiple Return Values](#)

[Exceptions](#)

[Summary of Functions](#)

[Numbers](#)

[Strings](#)

[Lists](#)

[Dictionaries](#)

[Type Conversions](#)

[Summary](#)

5 [Modules, Classes, and Methods](#)

[Modules](#)

[Using Modules](#)

[Useful Python Libraries](#)

[Installing New Modules](#)

[Object Orientation](#)

[Defining Classes](#)

[Inheritance](#)

[Summary](#)

6 [Files and the Internet](#)

[Files](#)

[Reading Files](#)

[Reading Big Files](#)

[Writing Files](#)

[The File System](#)

[Pickling](#)

[Internet](#)

[Summary](#)

7 [Graphical User Interfaces](#)

[Tkinter](#)

[Hello World](#)

[Temperature Converter](#)

[Other GUI Widgets](#)

[Checkbutton](#)

[Listbox](#)

[Spinbox](#)

[Layouts](#)

[Scrollbar](#)

[Dialogs](#)

[Color Chooser](#)

[File Chooser](#)

[Menus](#)

[The Canvas](#)

[Summary](#)

8 [Games Programming](#)

[What Is Pygame?](#)

[Hello Pygame](#)

[A Raspberry Game](#)

[Following the Mouse](#)

[One Raspberry](#)

[Catch Detection and Scoring](#)

[Timing](#)

[Lots of Raspberries](#)

[Summary](#)

9 [Interfacing Hardware](#)

[GPIO Pin Connections](#)

[Direct Connection to GPIO Pins](#)

[Expansion Boards](#)

[Pi Face](#)

[Slice of PI/O](#)

[RaspiRobotBoard](#)

[Gertboard](#)

[Prototyping Boards](#)

[Pi Cobbler](#)

[Pi Plate](#)

[Humble Pi](#)

[Arduino and the Pi](#)

[Arduino and Pi Talk](#)

[Summary](#)

10 [Prototyping Project \(Clock\)](#)

[What You Need](#)

[Hardware Assembly](#)

[Software](#)

[Phase Two](#)

[Summary](#)

11 [The RaspiRobot](#)

[What You Need](#)

[Phase 1: A Basic Rover](#)

[Hardware Assembly](#)

[About the Software](#)

[Phase 2: Adding a Range Finder and Screen](#)

[Step 1: Assemble the Range Finder Serial Adapter](#)

[Step 2: Attach the Screen](#)

[Step 3: Update the Software](#)

[Step 4: Run It](#)

[Revised Software](#)

[Summary](#)

12 [What Next](#)

[Linux Resources](#)

[Python Resources](#)

[Raspberry Pi Resources](#)

[Other Programming Languages](#)

[Scratch](#)

[C](#)

[Applications and Projects](#)

[Media Center \(Rasbmc\)](#)

[Home Automation](#)

[Summary](#)

[Index](#)

As always, I thank Linda for her patience and support.

I also thank Andrew Robinson and my son, Matthew Monk, for their technical review of much of the material in this book. Check out Andrew's Raspberry Pi project book. I'm sure it will be excellent.

From TAB/McGraw-Hill, my thanks go out to my patient and thoroughly nice editor Roger Stewart and the excellent project management of Vastavikta Sharma and Patty Mon. It is always a pleasure to work with such a great team.

The Raspberry Pi is rapidly becoming a worldwide phenomena. People are waking up to the possibility of a \$35 (U.S.) computer that can be put to use in all sorts of settings—from a desktop workstation to a media center to a controller for a home automation system.

This book explains in simple terms, to both nonprogrammers and programmers new to the Raspberry Pi, how to start writing programs for the Pi in the popular Python programming language. It then goes on to give you the basics of creating graphical user interfaces and simple games using the `pygame` module.

The software in the book mostly uses Python 3, with the occasional use of Python 2 where necessary for module availability. The Raspbian Wheezy distribution recommended by the Raspberry Pi Foundation is used throughout the book.

The book starts with an introduction to the Raspberry Pi and covers the topics of buying the necessary accessories and setting everything up. You then get an introduction to programming while you gradually work your way through the next few chapters. Concepts are illustrated with sample applications that will get you started programming your Raspberry Pi.

Three chapters are devoted to programming and using the Raspberry Pi's GPIO connector, which allows the device to be attached to external electronics. These chapters include two sample projects—one for making an LED clock and the other a Raspberry Pi controller robot, complete with ultrasonic rangefinder.

Here are the key topics covered in the book:

- Python numbers, variables, and other basic concepts
- Strings, lists, dictionaries, and other Python data structures
- Modules and object orientation
- Files and the Internet
- Graphical user interfaces using Tkinter
- Game programming using Pygame
- Interfacing with hardware via the GPIO connector
- Sample hardware projects

All the code listings in the book are available for download from the book's website at <http://www.raspberrypibook.com>, where you can also find other useful material relating to the book, including errata.

Introduction

The Raspberry Pi went on general sale at the end of February 2012 and immediately crashed the websites of the suppliers chosen to take orders for it. So what was so special about this little device and why has it created so much interest?

What Is the Raspberry Pi?

The Raspberry Pi, shown in [Figure 1-1](#), is a computer that runs the Linux operating system. It has USB sockets you can plug a keyboard and mouse into and HDMI (High-Definition Multimedia Interface) video output you can connect a TV or monitor into. Many monitors only have a VGA connector, and Raspberry Pi will not work with this. However, if your monitor has a DVI connector, cheap HDMI-to-DVI adapters are available.

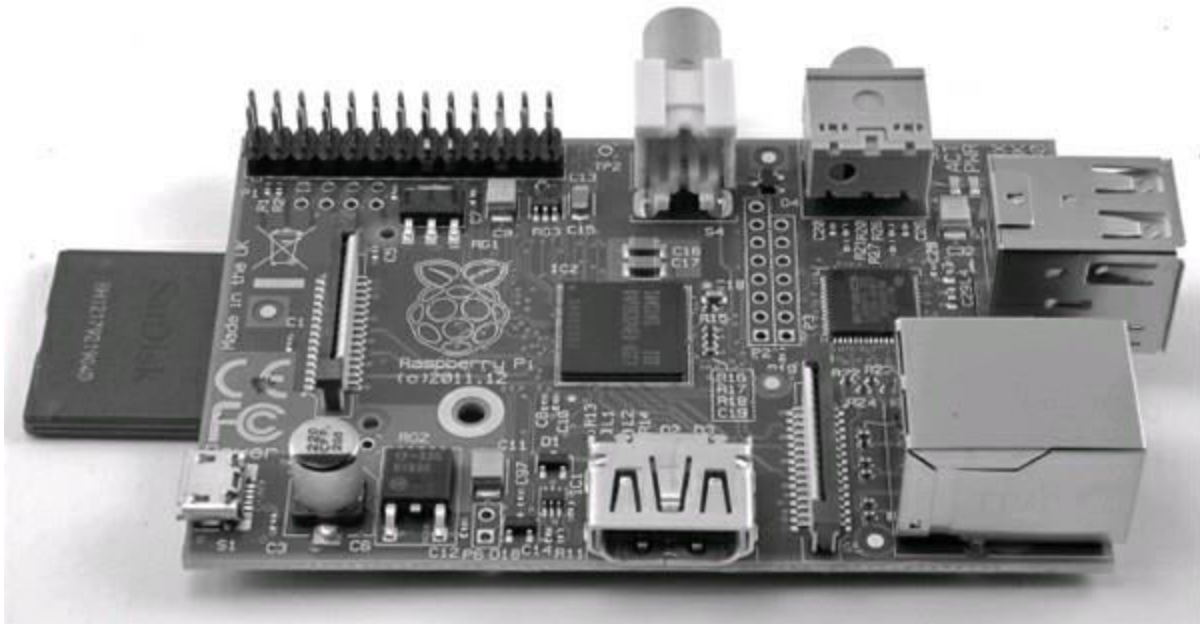


Figure 1-1 *The Raspberry Pi*

When Raspberry Pi boots up, you get the Linux desktop shown in [Figure 1-2](#). This really is a proper computer, complete with an office suite, video playback capabilities, games, and the lot. It's not Microsoft Windows; instead, it is Windows open source rival Linux (Debian Linux), and the windowing environment is called LXDE.

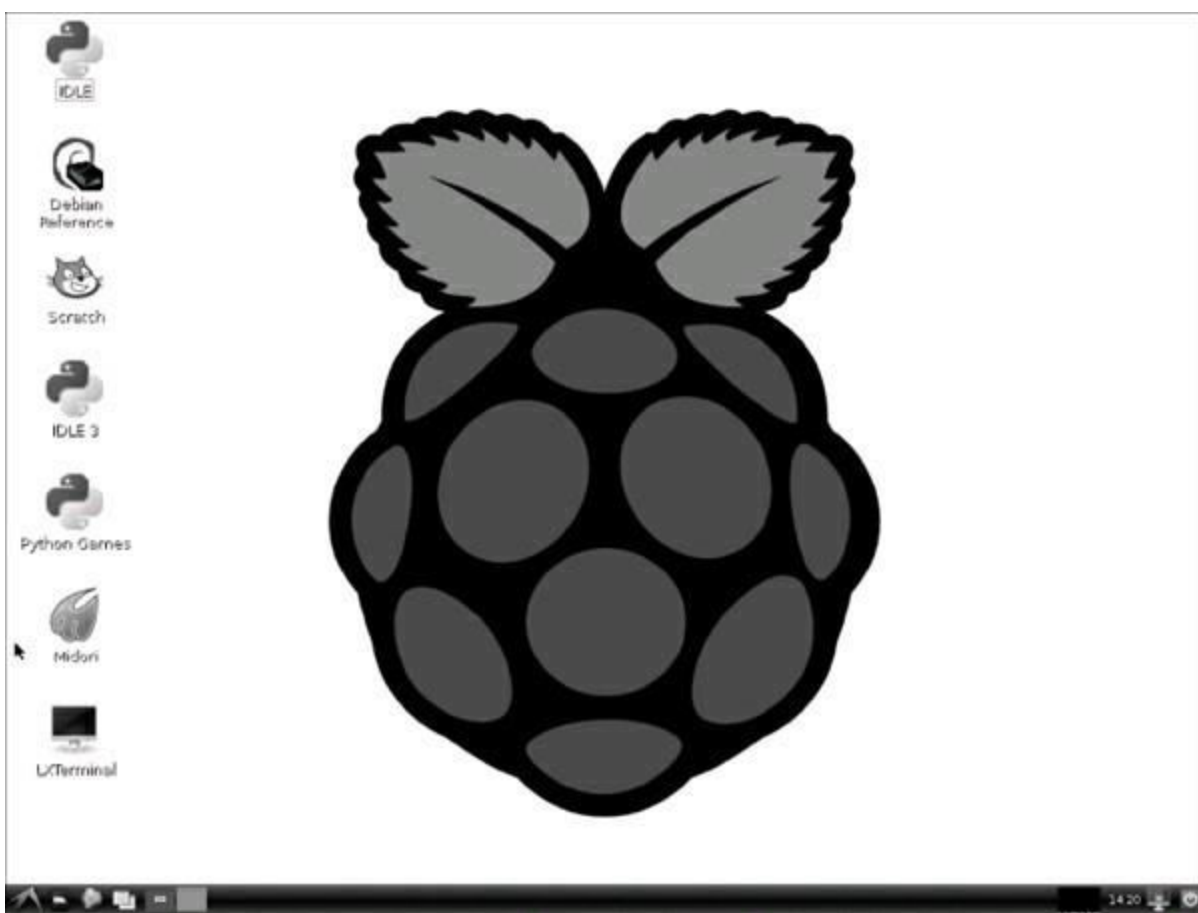


Figure 1-2 *The Raspberry Pi desktop*

Its small (the size of a credit card) and extremely affordable (starting at \$25). Part of the reason for this low cost is that some components are not included with the board or are optional extras. For instance, it does not come in a case to protect it—it is just a bare board. Nor does it come with a power supply, so you will need to find yourself a 5V micro-USB power supply, much like you would use to charge a phone (but probably with higher power). A USB power supply and a micro-USB lead are often used for this.

What Can You Do with a Raspberry Pi?

You can do pretty much anything on a Raspberry Pi that you can on any other Linux desktop computer, with a few limitations. The Raspberry Pi uses an SD card in place of a hard disk, although you can plug in a USB hard disk. You can edit office documents, browse the Internet, and play games (even games with quite intensive graphics, such as *Quake*).

The low price of the Raspberry Pi means that it is also a prime candidate for use as a media center. It can play video, and you can just about power it from the USB port you find on many TVs.

A Tour of the Raspberry Pi

[Figure 1-3](#) labels the various parts of a Raspberry Pi. This figure takes you on a tour of the Model B Raspberry Pi, which differs from the Model A by virtue of having an RJ-45 LAN connector, allowing it to be connected to a network.

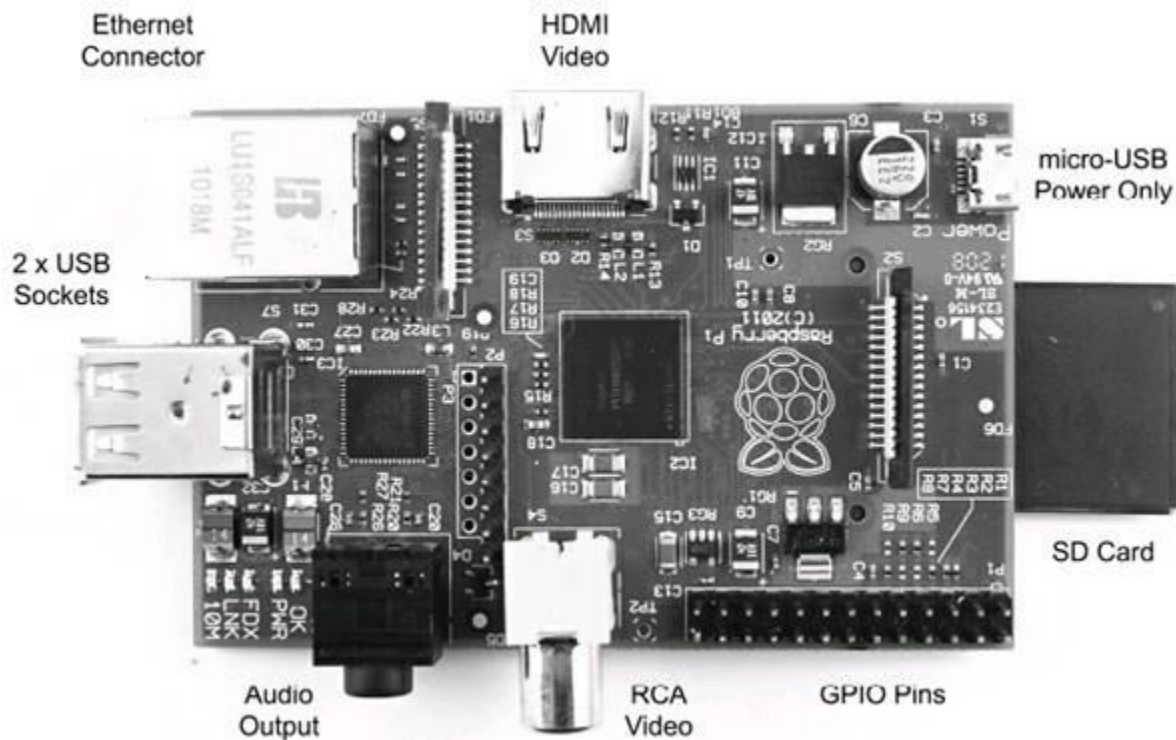


Figure 1-3 *The anatomy of a Raspberry Pi*

The RJ-45 Ethernet connector is shown in the top-left corner of the figure. If your home hub is handy, you can plug your Raspberry Pi directly into your local network. While we are on the subject, it is worth noting that the Raspberry Pi does not have Wi-Fi built in. For wireless networking, you will need to plug in a USB wireless adapter. This may then require some additional work installing drivers.

Immediately below the Ethernet socket you'll find a pair of USB sockets, one on top of the other. You can plug a keyboard, mouse, or external hard disks into the board, but you'll fairly rapidly run out of sockets. For this reason, many people use a USB hub to gain a few more USB sockets.

In the bottom-left corner of the figure you'll find an audio socket that provides a stereo analog signal for headphones or powered speakers. The HDMI connector is also sound capable.

Next to the audio socket is an RCA video connector. You are unlikely to use this connector unless you are using your Raspberry Pi with an older TV. You are far more likely to use the HDMI connector immediately opposite it, shown at the top of the figure. HDMI is higher quality, includes sound, and can be connected to DVI-equipped monitors with a cheap adapter.

To the right of the yellow RCA jack are two rows of pins. These are called GPIO (General Purpose Input/Output) pins, and they allow the Raspberry Pi to be connected to custom electronics. Users of the Arduino and other microcontroller boards will be used to the idea of GPIO pins. Later, in [Chapter 11](#), we will use these pins to enable our Raspberry Pi to be the “brain” of a little roving robot by controlling its motors. In [Chapter 10](#), we will use the Raspberry Pi to make an LED clock.

The Raspberry Pi has an SD card slot underneath the board. This SD card needs to be at least 2GB in size. It contains the computer's operating system as well as the file system in which you can store any documents you create. The SD card is an optional extra feature when buying your Raspberry Pi. Preparing your own SD card is a little complex to do, and suppliers such as SK Pang, Farnell, and RS Components all sell already-prepared SD cards. Because no disk is built into your Raspberry Pi, this card is effectively your computer, so you could take it out and put it in a different Raspberry Pi and all your stuff would be there.

Above the SD card is a micro-USB socket. This is only used to supply power to the Raspberry Pi.

Therefore, you will need a power supply with a micro-USB connector on the end. This is the same type of connector used by many mobile phones, including most Android phones. Do, however, check that it is capable of supplying at least 700mA; otherwise, your Raspberry Pi may behave erratically.

For those interested in technical specs, the big square chip in the center of the board is where all the action occurs. This is Broadcom's "System on a Chip" and includes 256MB of memory as well as the graphics and general-purpose processors that drive the Raspberry Pi.

You may also have noticed flat cable connectors next to the SD card and between the Ethernet and HDMI connectors. These are for LCD displays and a camera, respectively. Look for camera and LCD display modules becoming available for the Pi in the near future.

Setting Up Your Raspberry Pi

You can make your life easier by buying a prepared SD card and power supply when you buy your Raspberry Pi, and for that matter you may as well get a USB keyboard and mouse (unless you have them lurking around the house somewhere). Let's start the setup process by looking at what you will need and where to get it from.

Buying What You Need

[Table 1-1](#) shows you what you will need for a fully functioning Raspberry Pi system. At the time of writing, the Raspberry Pi itself is sold through two worldwide distributors based in the UK: Farnell (and the related U.S. company Newark) and RS Components, which is not to be confused with RadioShack.

Item	Source and Part Number	Additional Information
Raspberry Pi, Model A or B	Farnell (www.farnell.com) Newark (www.newark.com) RS Components (www.rs-components.com)	The difference between the two models is that the Model B has a network connection.
USB power supply (U.S. mains plug)	Newark: 39T2392 RadioShack: 55053163 Adafruit PID:501	5V USB power supply. Should be capable of supplying 700mA (3W), but 1A (5W) is better.
USB power supply (UK mains plug)	Farnell: 2100374 Maplins: N15GN	
USB power supply (European mains plug)	Farnell: 1734526	
Micro-USB lead	RadioShack: 55048949 Farnell: 2115733 Adafruit PID 592	
Keyboard and mouse	Any computer store	Any USB keyboard will do. Also, wireless keyboards and mice that come with their own USB adaptor will work, too.
TV/monitor with HDMI	Any computer/electrical store	
HDMI lead	Any computer/electrical store	
SD card (prepared)	SK Pang: RSP-2GBSD Newark: 96T7436 Farnell: 2113756	
Wi-Fi adapter*	http://elinux.org/RPi_VerifiedPeripherals#USB_WiFi_Adapters	Elinux.org provides an up-to-date list of Wi-Fi adapters.
USB hub*	Any computer store	
HDMI-to-DVI adapter*	Newark: 74M6204 Maplins: N24CJ Farnell: 1428271	
Ethernet patch cable*	Any computer store	
Case*	Adafruit, SK Pang, or Alliedelec.com	
* These items are optional.		

Table 1-1 *A Raspberry Pi Kit*

Power Supply

[Figure 1-4](#) show a typical USB power supply and USB-A-to-micro-USB lead.



Figure 1-4 *USB power supply*

You may be able to use a power supply from an old MP3 player or the like, as long as it is 5V and can supply enough current. It is important not to overload the power supply because it could get hot and fail (or even be a fire hazard). Therefore, the power supply should be able to supply at least 700mA, but 1A would give the Raspberry Pi a little extra when it comes to powering the devices attached to its USB ports.

If you look closely at the specs written on the power supply, you should be able to determine its current supply capabilities. Sometimes its power-handling capabilities will be expressed in watts (W); if that's the case, it should be at least 3W. If it indicates 5W, this is equivalent to 1A.

Keyboard and Mouse

The Raspberry Pi will work with pretty much any USB keyboard and mouse. You can also use most wireless USB keyboards and mice—the kind that come with their own dongle to plug into the USB port. This is quite a good idea, especially if they come as a pair. That way, you are only using up one of the USB ports. This will also come in quite handy in [Chapter 10](#) when we use a wireless keyboard to control our Raspberry Pi-based robot.

Display

Including an RCA video output on the Raspberry Pi is, frankly, a bit puzzling because most people are going to go straight to the more modern HDMI connector. A low-cost 22-inch LCD TV will make a perfectly adequate display for the Pi. Indeed, you may just decide to use the main family TV, just plugging the Pi into the TV when you need it.

If you have a computer monitor with just a VGA connector, you are not going to be able to use it without an expensive converter box. On the other hand, if your monitor has a DVI connector, an inexpensive adapter will do the job well.

SD Card

You can use your own SD card in the Raspberry Pi, but it will need to be prepared with an operating system disk image. This is a little fiddly, so you may just want to spend a dollar or two more and buy an SD card that is already prepared and ready to go.

You can also find people at Raspberry Pi meet-ups who will be happy to help you prepare an SD card. The prepared SD cards supplied by Farnell and RS Components are overpriced. Look around on the Internet to find suppliers (such as SK Pang) who sell prepared cards, with the latest operating system distribution, for less than you would pay for an SD card in a supermarket. If you indeed want to “roll your own” SD card, refer to the instructions found at www.raspberrypi.org/downloads.

To prepare your own card, you must have another computer with an SD card reader. The procedure

is different depending on whether your host computer is a Windows, Mac, or Linux machine. However, various people have produced useful tools that try to automate the process as much as possible.

If you decide to roll your own, be sure to follow the instructions carefully—with some tools, it is quite easy to accidentally reformat a hard disk attached to your computer if the tool mistakes it for the SD card! Fortunately, this process is getting better all the time as easier-to-use software tools become available.

A big advantage of making your own SD card is that you can actually choose from a range of operating system distributions. [Table 1-2](#) shows the most popular ones available at the time of writing. Check on the Raspberry Pi Foundation’s website for newer distributions.

Distribution	Notes
Raspbian Wheezy	This is the “standard” Raspberry Pi operating system and the one used in all the examples in this book. It uses the LXDE desktop.
Arch Linux ARM	This distribution is more suited to Linux experts.
QtonPi	This distribution is intended for people developing rich graphical programs using the Qt5 graphics framework.
Occidentalis	A distribution made by Adafruit and based on Raspbian Wheezy but with improvements intended for hardware hackers.

Table 1-2 *Raspberry Pi Linux Distributions*

Of course, nothing is stopping you from buying a few SD cards and trying out the different distributions to see which you prefer. However, if you are a Linux beginner, you should stick to the standard Raspbian Wheezy distribution.

Case

The Raspberry Pi does not come in any kind of enclosure. This helps to keep the price down, but also makes it rather vulnerable to breakage. Therefore, it is a good idea to either make or buy a case as soon as you can. [Figure 1-5](#) shows a few of the ready-made cases currently available.

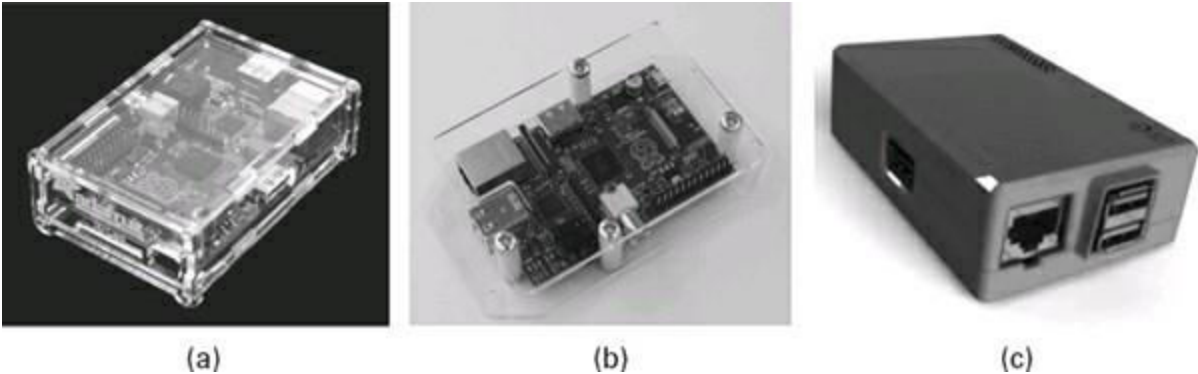


Figure 1-5 *Commercial Raspberry Pi cases*

The cases shown are supplied by (a) Adafruit (www.adafruit.com), (b) SK Pang (www.skpang.co.uk/), and (c) ModMyPi (www.modmypi.com). The case you choose will depend on what you plan to do with your Raspberry Pi. If you have access to a 3D printer, you can also use the following open source designs:

- www.thingiverse.com/thing:23446
- www.thingiverse.com/thing:24721

You can also find a folded card design called the Raspberry Punnet at www.raspberrypi.org/archives/1310.

People are having a lot of fun building their Raspberry Pi into all sorts of repurposed containers, such as vintage computers and games consoles. One could even build a case using Legos. My first

case for a Raspberry Pi was made by cutting holes in a plastic container that used to hold business cards (see [Figure 1-6](#)).

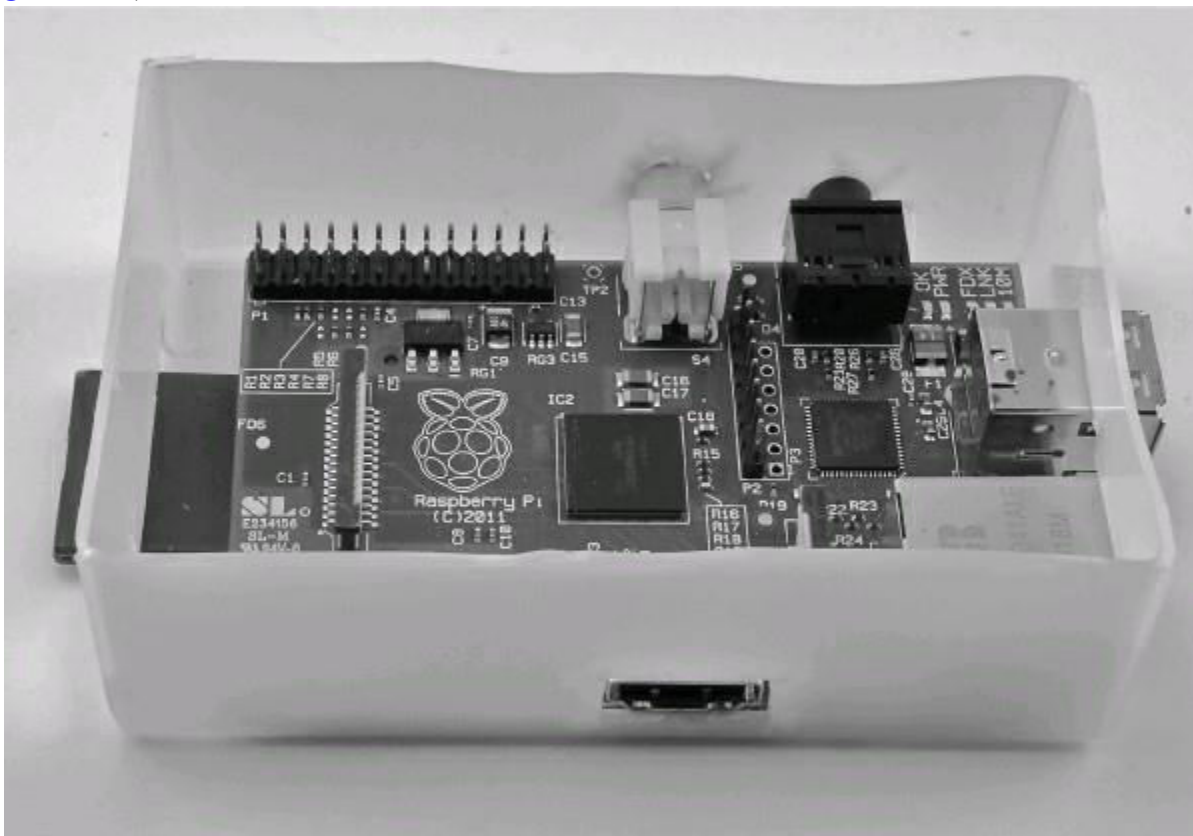


Figure 1-6 *A homemade Raspberry Pi case*
Wi-Fi

Neither of the Raspberry Pi models has support for Wi-Fi. Therefore, to wirelessly connect your Raspberry Pi to the network, you have just two options. The first is to use a USB wireless adapter that just plugs into a USB socket (see [Figure 1-7](#)). With any luck, Linux should recognize it and immediately allow you to connect (or show what you need to do to connect).

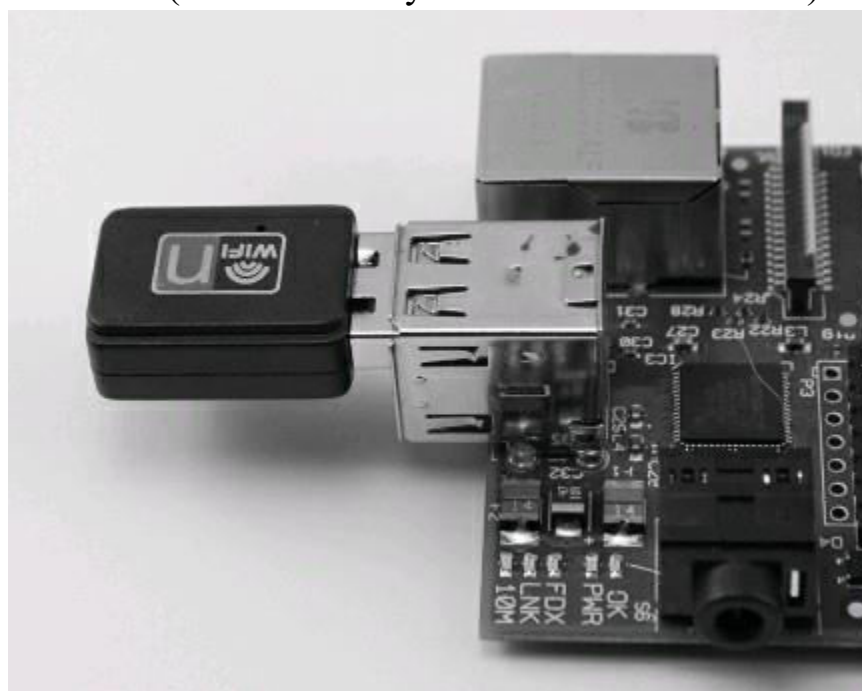


Figure 1-7 *Wi-Fi adapter*

The Wi-Fi adapters in the list referenced in [Table 1-1](#) are purported to work with the Raspberry Pi. However, there are sometimes problems with Wi-Fi drivers, so be sure to check the Raspberry Pi forum and wiki for up-to-date information on compatible devices.

The second option for Wi-Fi is to use a Wi-Fi bridge with a Model B Raspberry Pi. These devices are usually USB powered and plug into the Ethernet socket on the Raspberry Pi. They are often used by the owners of game consoles that have an Ethernet socket but no Wi-Fi. This setup has the advantage in that the Raspberry Pi does not require any special configuration.

USB Hub

Because the Raspberry Pi has just two USB ports available, you will rapidly run out of sockets. The way to obtain more USB ports is to use a USB hub (see [Figure 1-8](#)).



Figure 1-8 *A USB hub*

These hubs are available with anywhere from three to eight ports. Make sure that the port supports USB 2. It is also a good idea to use a “powered” USB hub so that you do not draw too much power from the Raspberry Pi.

Connecting Everything Together

Now that you have all the parts you need, let’s get it all plugged together and boot your Raspberry Pi for the first time. [Figure 1-9](#) shows how everything needs to be connected.

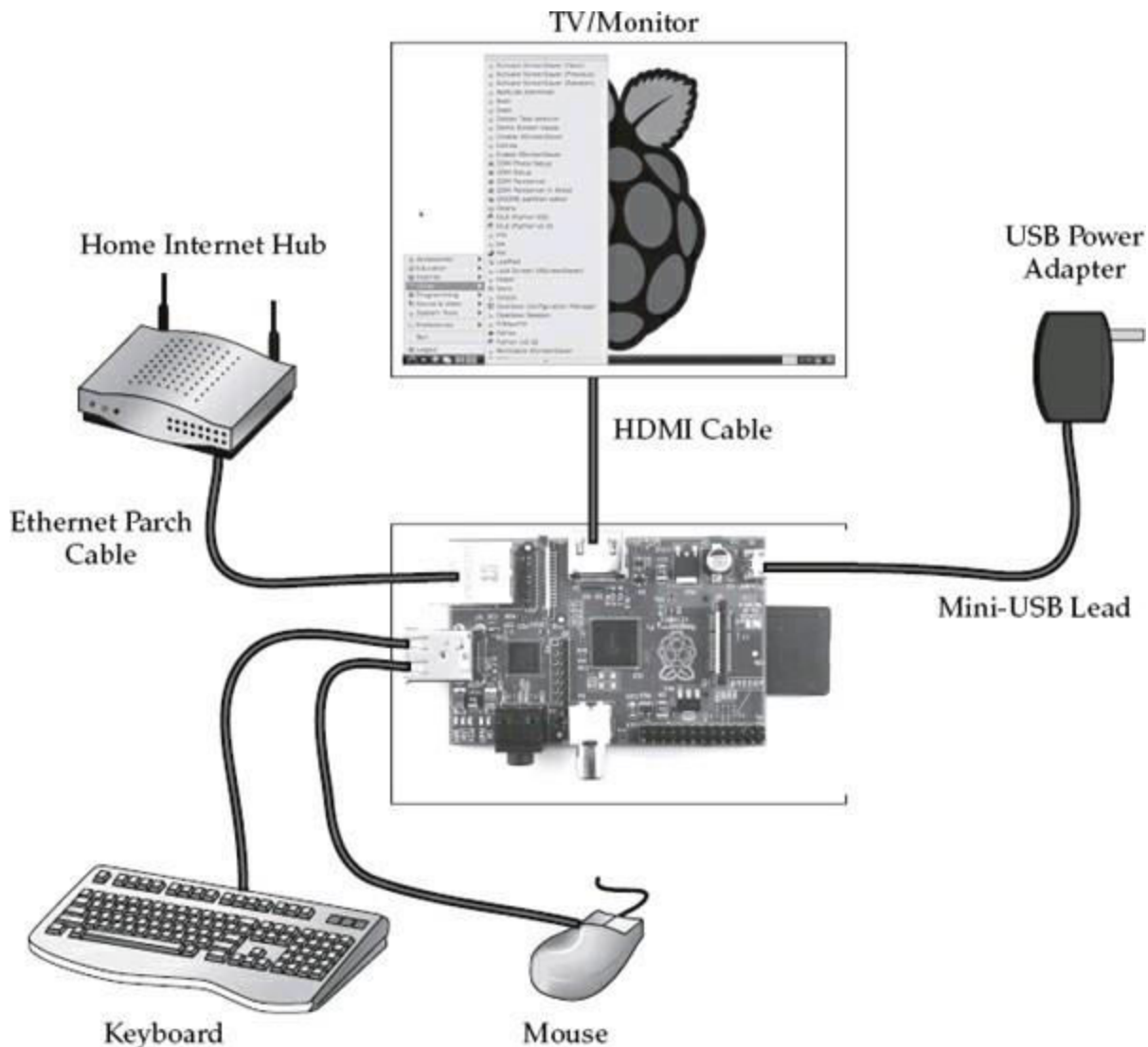


Figure 1-9 A Raspberry Pi system

Insert the SD card, connect the keyboard, mouse, and monitor to the Pi, attach the power supply, and you are ready to go.

Booting Up

The first time you boot your Raspberry Pi, it will not immediately boot into the kind of graphical environment you would normally see in, say, a Windows computer. Instead, it will stop to allow a first-time configuration (see [Figure 1-10](#)). It is a good idea to make a number of the configuration changes shown here.



Figure 1-10 *Configuration screen*

First, if your SD card is larger than 2GB, the Raspberry Pi will only make use of the first 2GB unless you select the option to `expand_rootfs`. Select this option using the UP and DOWN ARROW keys and ENTER.

Another change well worth making is the `boot_behaviour` option. If this is not set to Boot Straight to Desktop, you will be forced to log in and start the windowing environment manually each time you power up your Raspberry Pi (see [Figure 1-11](#)).

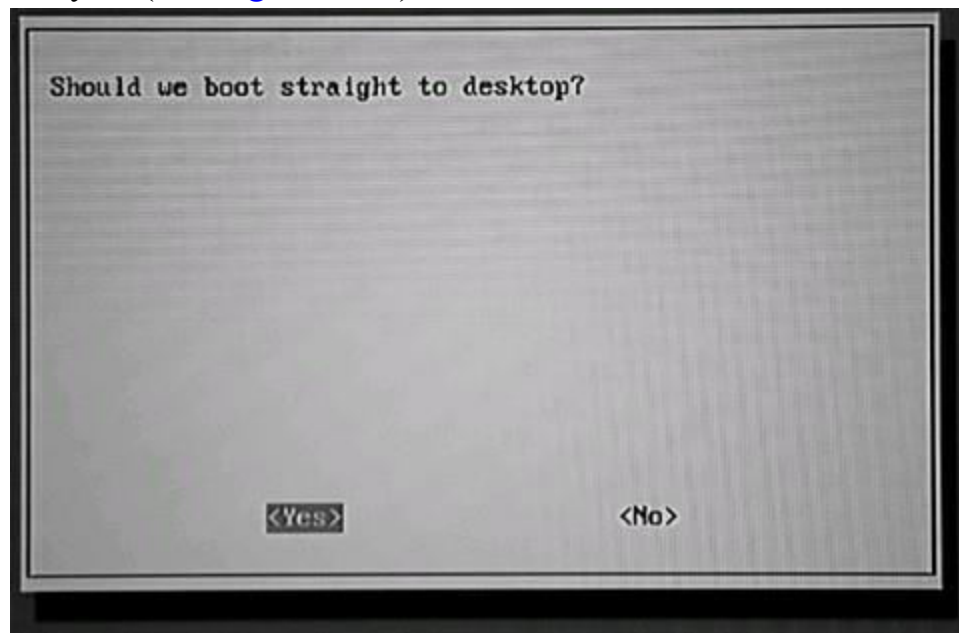


Figure 1-11 *Boot-to-desktop option*

Summary

Now that we have set up our Raspberry Pi and it is ready to use, we can start exploring some of its features and get a grip on the basics of Linux.

The Raspberry Pi uses Linux as its operating system. This chapter introduces Linux and shows you how to use the desktop and command line.

Linux

Linux is an open source operating system. This software has been written as a community project for those looking for an alternative to the duopoly of Microsoft Windows and Apple OS X. It is a fully featured operating system based on the same solid UNIX concepts that arose in the early days of computing. It has a loyal and helpful following and has matured into an operating system that is powerful and easy to use.

Although the operating system is called Linux, various Linux distributions (or *distros*) have been produced. These involve the same basic operating system, but are packaged with different bundles of applications or different windowing systems. Although many distros are available, the one recommended by the Raspberry Pi foundation is called Raspbian Wheezy.

If you are only used to some flavor of Microsoft Windows, expect to experience some frustration as you get used to a new operating system. Things work a little differently in Linux. Almost anything you may want to change about Linux can be changed. The system is open and completely under your control. However, as they say in *Spiderman*, with great power comes great responsibility. This means that if you are not careful, you could end up breaking your operating system.

The Desktop

At the end of [Chapter 1](#), we had just booted up our Raspberry Pi, logged in, and started up the windowing system. [Figure 2-1](#) serves to remind you of what the Raspberry Pi desktop looks like.

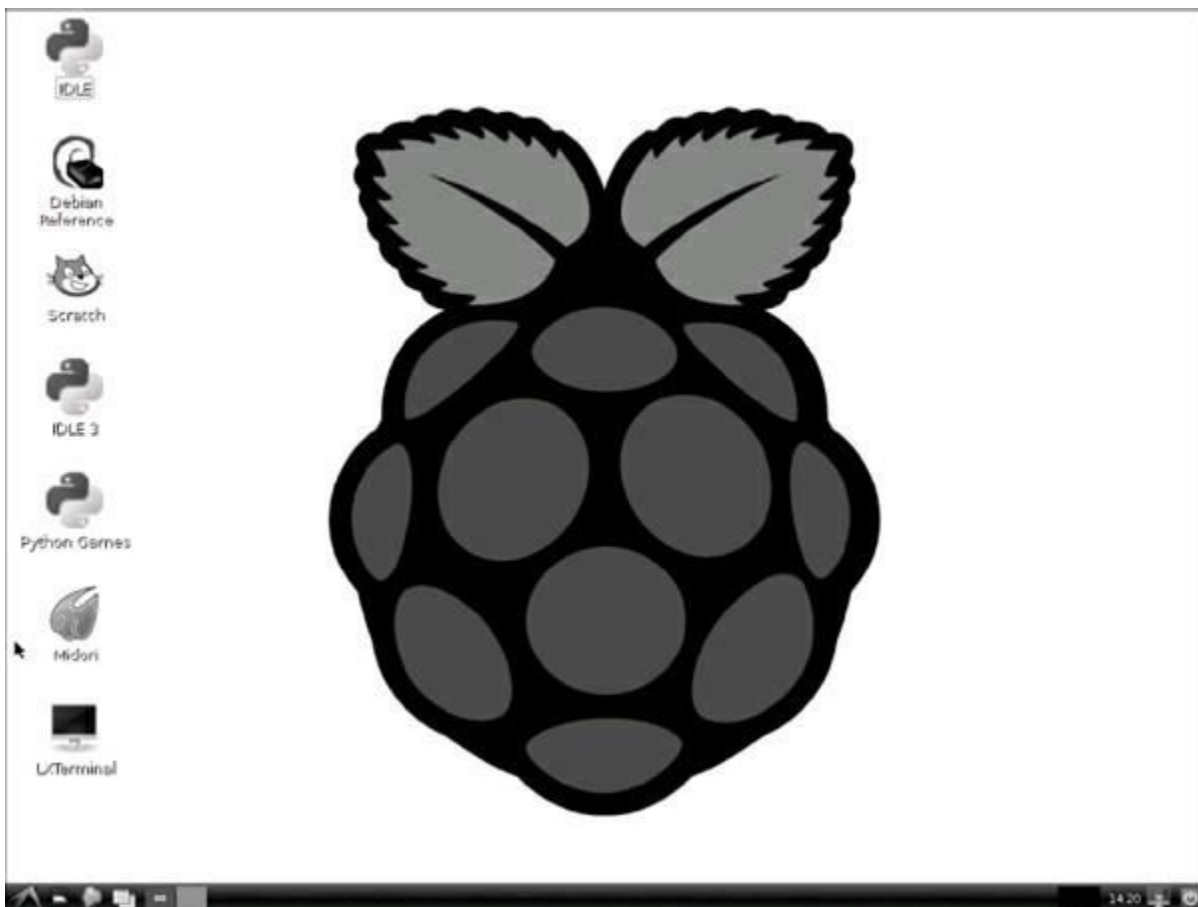


Figure 2-1 *Raspberry Pi desktop*

If you are a user of Windows or Mac computers, you will be familiar with the concept of a desktop

as a folder within the file system that acts as a sort of background to everything you do on the computer.

Along the left side of the desktop, you see some icons that launch applications. Clicking the left-most icon on the bar at the bottom of the screen will show us some of the applications and tools installed on the Raspberry Pi (rather like the Start menu in Microsoft Windows). We are going to start with the File Manager, which can be found under the Accessories.

The File Manager is just like the File Explorer in Windows or the Finder on a Mac. It allows you to explore the file system, copy and move files, as well as launch files that are executable (applications).

When it starts, the File Manager shows you the contents of your home directory. You may remember that when you logged in, you gave your login name as pi. The root to your home directory will be `/home/pi`. Note that like Mac's OS X, Linux uses slash (/) characters to separate the parts of a directory name. Therefore, / is called the *root* directory and `/home/` is a directory that contains other directories, one for each user. Our Raspberry Pi is just going to have one user (called pi), so this directory will only ever contain a directory called pi. The current directory is shown in the address bar at the top, and you can type directly into it to change the directory being viewed, or you can use the navigation bar at the side. The contents of the directory `/home/ pi` include just the directories Desktop and python_games.

Double-clicking Desktop will open the Desktop directory, but this is not of much interest because it just contains the shortcuts on the left side of the desktop. If you open python_games, you will see some games you can try out, as shown in [Figure 2-2](#).

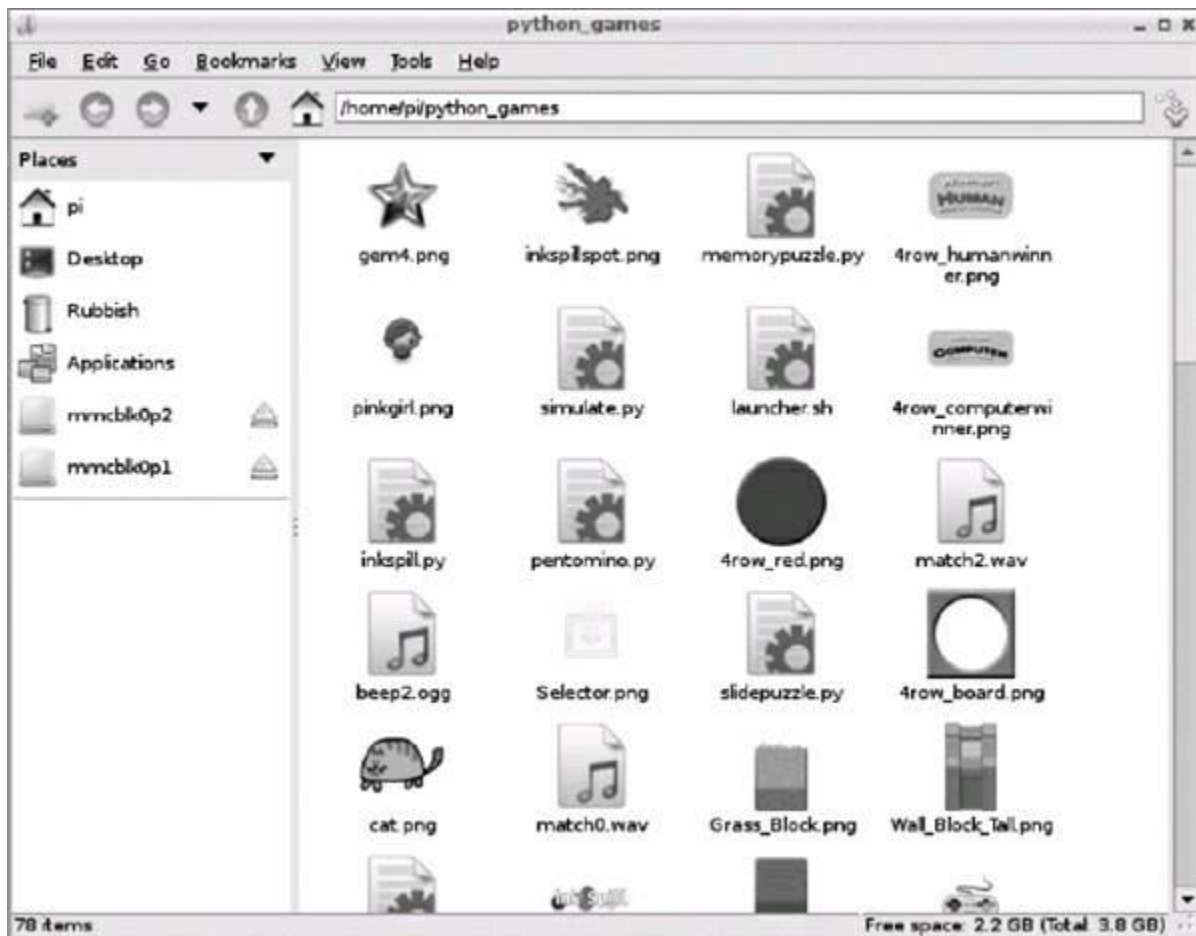


Figure 2-2 *The contents of python_games, as shown in File Manager*

You shouldn't often need to use any of the file system outside of your home directory. You should keep all documents, music files, and so on, housed within directories on your home folder or on an external USB flash drive.

The Internet

If you have a home hub and can normally plug in any Internet device using an Ethernet cable, you should have no problem getting your Raspberry Pi online. Your home hub should automatically assign the Raspberry Pi an IP address and allow it to connect to the network.

The Raspberry Pi comes with a web browser called Midori, which you will find under the Internet section of your start menu. You can check that your connection is okay by starting Midori and connecting to a website of your choice, as shown in [Figure 2-3](#).



Figure 2-3 *The Midori web browser*

The Command Line

If you are a Windows or Mac user, you may have never used the command line. If you are a Linux user, on the other hand, you almost certainly will have done so. In fact, if you are a Linux user, then about now you will have realized that you probably don't need this chapter because it's all a bit basic for you.

Although it is possible to use a Linux system completely via the graphical interface, in general you will need to type commands into the command line. You do this to install new applications and to configure the Raspberry Pi.

From the launcher button (bottom left), open the LXTerminal, which is shown in [Figure 2-4](#).

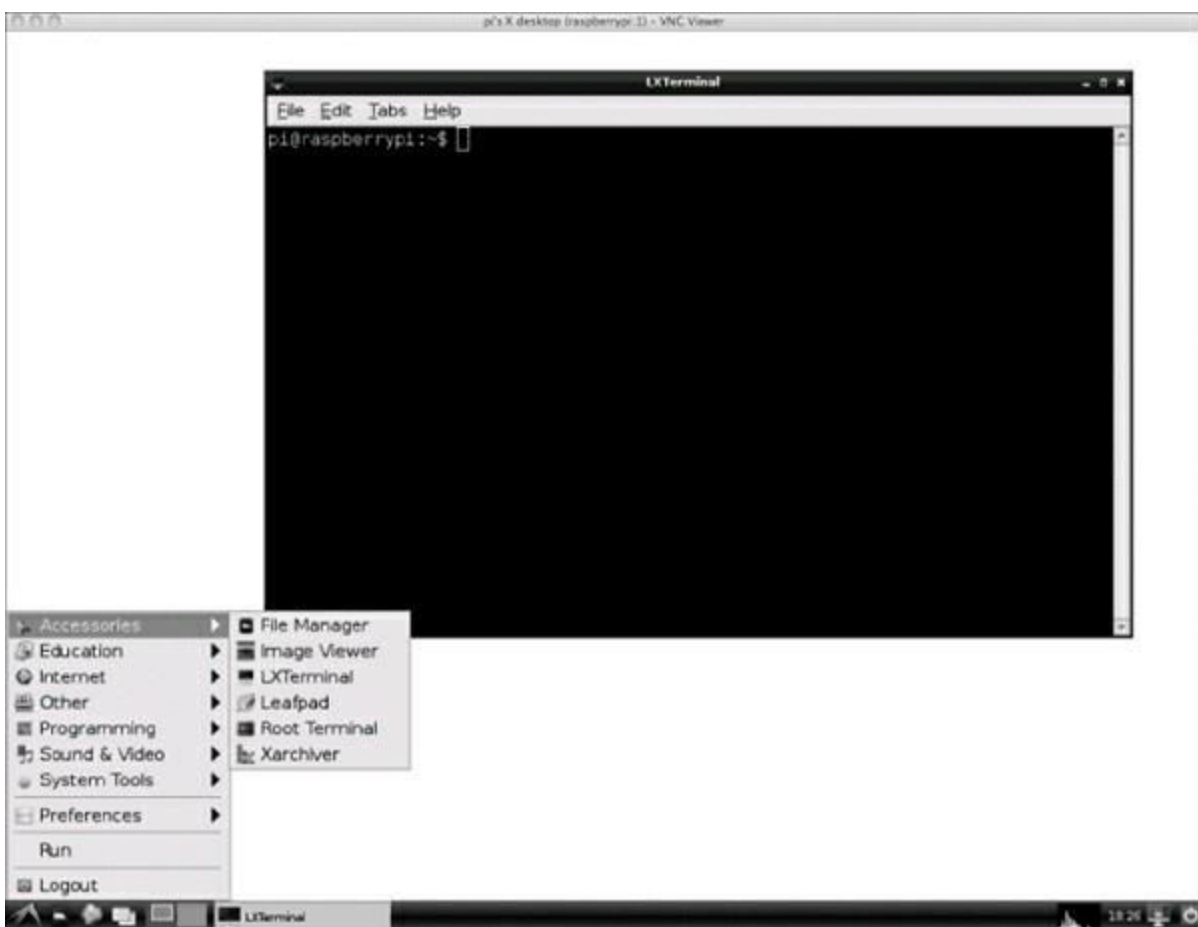


Figure 2-4 The LXTerminal command line

Navigating with the Terminal

You will find yourself using three commands a lot when you are using the command line. The first command is `pwd`, which is short for *print working directory* and shows you which directory you are currently in. Therefore, after the `$` sign in the terminal window, type `pwd` and press RETURN, as shown in [Figure 2-5](#).

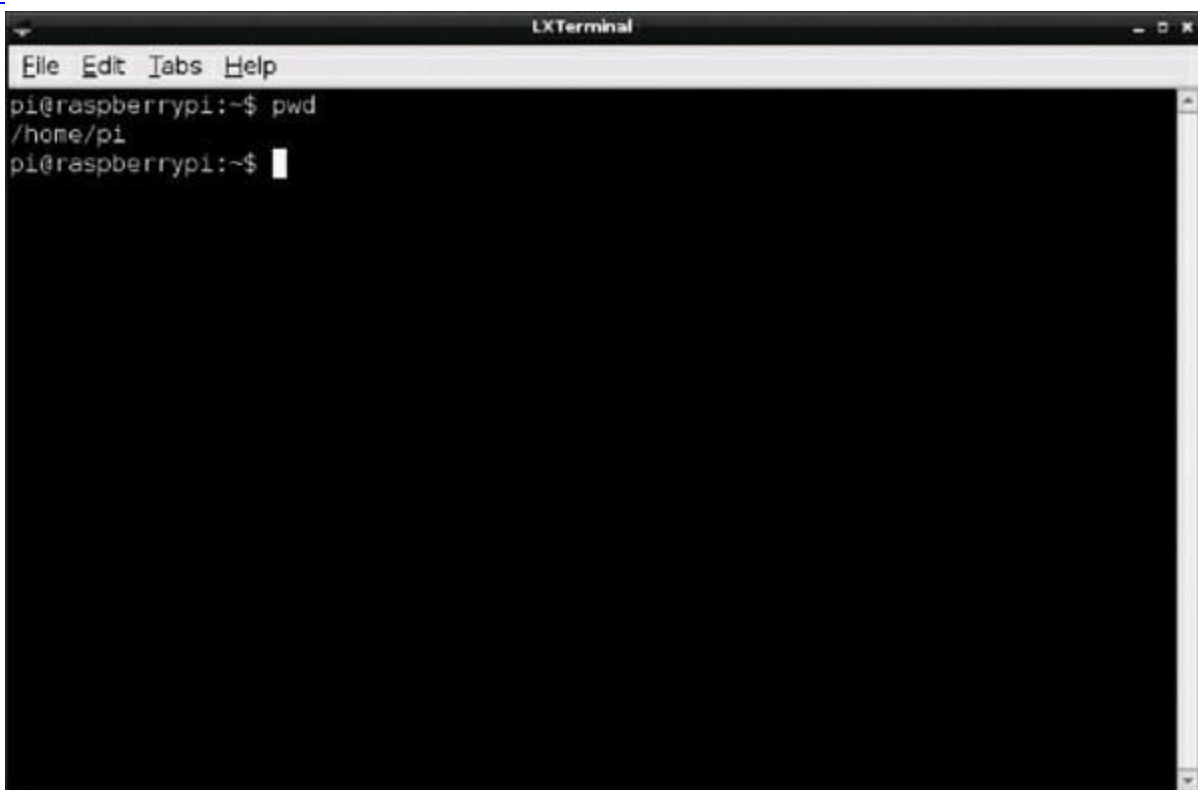


Figure 2-5 The `pwd` command

As you can see, we are currently in `/home/pi`. Rather than provide a screen shot for everything we

are going to type into the terminal, I will use the convention that anything I want you to type will be prefixed with a `$` sign, like this:

```
$pwd
```

Anything you should see as a response will not have `$` in front of it. Therefore, the whole process of running the `pwd` command would look something like this:

```
$pwd
/home/pi
```

The next common command we are going to discuss is `ls`, which is short for *list* and shows us a list of the files and directories within the working directory. Try the following:

```
$ls
Desktop
```

This tells us that the only thing in `/home/pi` is the directory `Desktop`.

The final command we are going to cover for navigating around is `cd` (which stands for *change directory*). This command changes the current working directory. It can change the directory relative either to the old working directory or to a completely different directory if you specify the whole directory, starting with `/`. So, for example, the following command will change the current working directory to `/home/pi/Desktop`:

```
$pwd
/home/pi
$cd Desktop
```

You could do the same thing by typing this:

```
cd /home/pi/Desktop
```

Note that when entering a directory or filename, you do not have to type all of it. Instead, at any time after you have typed some of the name, you can press the `TAB` key. If the filename is unique at that point, it will be automatically completed for you.

```
sudo
```

Another command that you will probably use a lot is `sudo` (for super-user do). This runs whatever command you type after it as if you were a super-user. You might be wondering why, as the sole user of this computer, you are not automatically a super-user. The answer is that, by default, your regular user account (username: `pi`, password: `raspberry`) does not have privileges that, say, allow you to go to some vital part of the operating system and start deleting files. Instead, to cause such mayhem, you have to prefix those commands with `sudo`. This just adds a bit of protection against accidents.

For the commands we have discussed so far, you will not need to prefix them with `sudo`. However, just for interest, try typing the following:

```
sudo ls
```

This will work the same way `ls` on its own works; you are still in the same working directory. The only difference is that you will be asked for your password the first time you use `sudo`.

Applications

The Raspbian Wheezy distribution for Raspberry Pi is fairly sparse. However, loads of applications can be installed. Installing new applications requires the command line again. The command `apt-get` is used to both install and uninstall applications. Because installing an application often requires super-user privileges, you should prefix `apt-get` commands with `sudo`.

The command `apt-get` uses a database of available packages that is updated over the Internet, so the first `apt-get` command you should use is `sudo apt-get update`

```
sudo apt-get update
```

which updates the database of packages. You will need to be connected to the Internet for it to work.

To install a particular package, all you need to know is the package manager name for it. For

example, to install the Abiword word processor application, all you need to type is the following:

```
sudo apt-get install abiword
```

It will take a while for everything that is needed to be downloaded and installed, but at the end of the process you will find that you have a new folder in your start menu called Office that contains the application Abiword (see [Figure 2-6](#)).

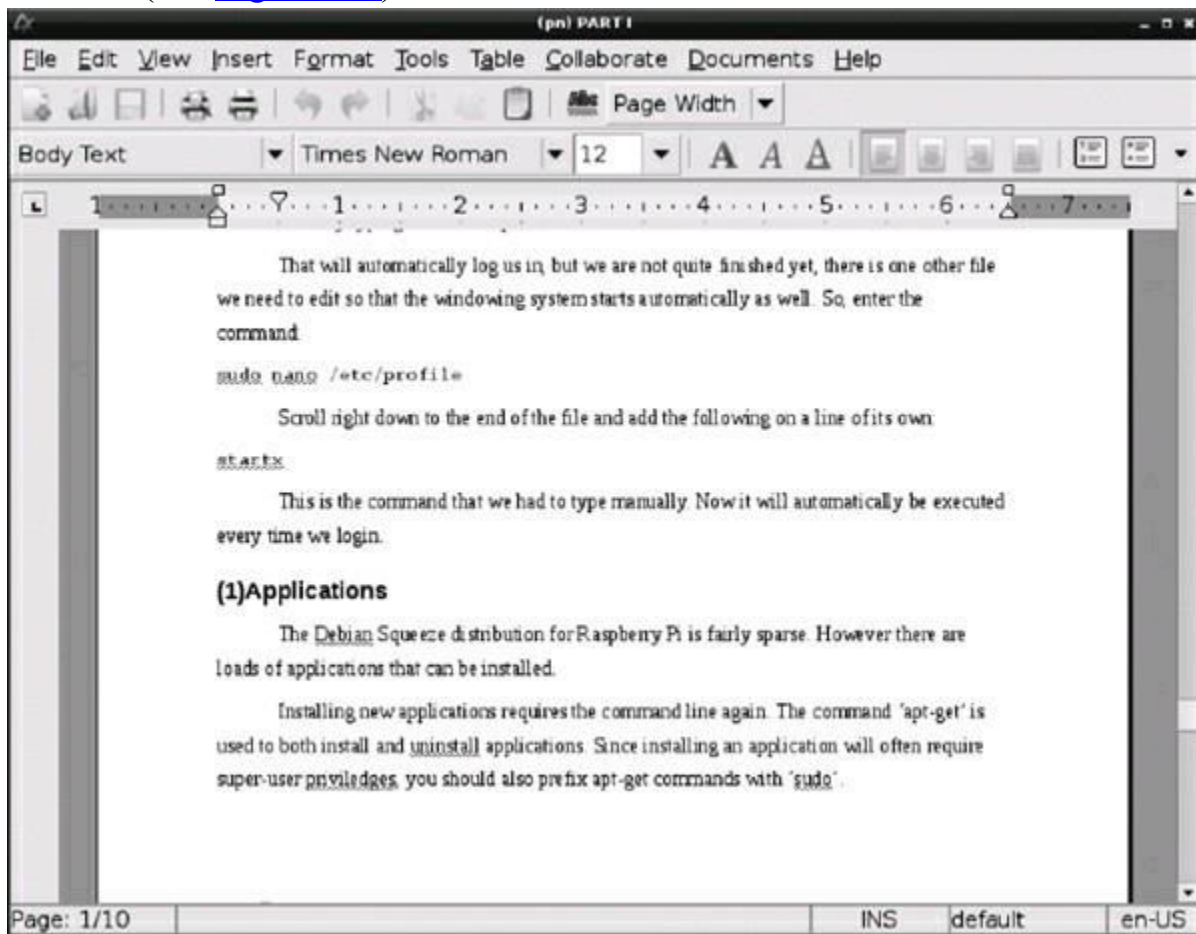


Figure 2-6 *Abiword screen*

You will notice that the text document in Abiword is actually part of this chapter. In fact, it is close to this part of this chapter, as I am writing it. (I can feel myself falling into a recursive hole. I may well vanish in a puff of logic.)

Abiword is a perfectly serviceable word processor. If I didn't love my Mac quite so much, I would be tempted to write this entire book on my Raspberry Pi.

While we are on the subject of office applications, the spreadsheet stable mate of Abiword is called Gnumeric. To install it, here is all you need to type:

```
sudo apt-get install gnumeric
```

Once this application is installed, another option will have appeared in your Office menu—this one for Gnumeric.

To find out about other packages you might want to install, look for recommendations on the Internet, especially on the Raspberry Pi forum (www.raspberrypi.org/phpBB3). You can also browse the list of packages available for Raspbian Wheezy at <http://packages.debian.org/stable/>.

Not all of these packages will work, because the Raspberry Pi does not have vast amounts of memory and storage available to it; however, many will.

If you want to remove a package, use the following command:

```
sudo apt-get remove --auto-remove --purge packagename
```

This removes both the package and any packages it depends on that are not used by something else that still needs them. Be sure to keep an eye on the bottom-right corner of your File Manager window; it will tell you how much free space is available.

Internet Resources

Aside from the business of programming the Raspberry Pi, you now have a functioning computer that you are probably keen to explore. To help you with this, many useful Internet sites are available where you can obtain advice and recommendations for getting the most out of your Raspberry Pi.

[Table 2-1](#) lists some of the more useful sites relating to the Raspberry Pi. Your search engine will happily show you many more.

Site	Description
www.raspberrypi.org	The home page of the Raspberry Pi Foundation. Check out the forum and FAQs.
www.raspberrypi-spy.co.uk	A blog site with useful how-to posts.
http://elinux.org/RaspberryPiBoard	The main Raspberry Pi wiki. Lots of information about the Raspberry Pi, especially a useful list of verified peripherals (http://elinux.org/RPi_VerifiedPeripherals).

Table 2-1 *Internet Resources for the Raspberry Pi*

Summary

Now that we have everything set up and ready to go on our Raspberry Pi, it is time to start programming in Python.

The time has come to start creating some of our own programs for the Raspberry Pi. The language we are going to use is called Python. It has the great benefit that it is easy to learn while at the same time being powerful enough to create some interesting programs, including some simple games and programs that use graphics.

As with most things in life, it is necessary to learn to walk before you can run, and so we will begin with the basics of the Python language.

Okay, so a programming language is a language for writing computer programs in. But why do we have to use a special language anyway? Why couldn't we just use a human language? How does the computer use the things that we write in this language?

The reason why we don't use English or some other human language is that human languages are vague and ambiguous. Computer languages use English words and symbols, but in a very structured way.

IDLE

The best way to learn a new language is to begin using it right away. So let's start up the program we are going to use to help us write Python. This program is called IDLE, and you will find it in the programming section of your start menu. In fact, you will find more than one entry for IDLE. Select the one labelled "IDLE 3" after it. [Figure 3-1](#) shows IDLE and the Python Shell.

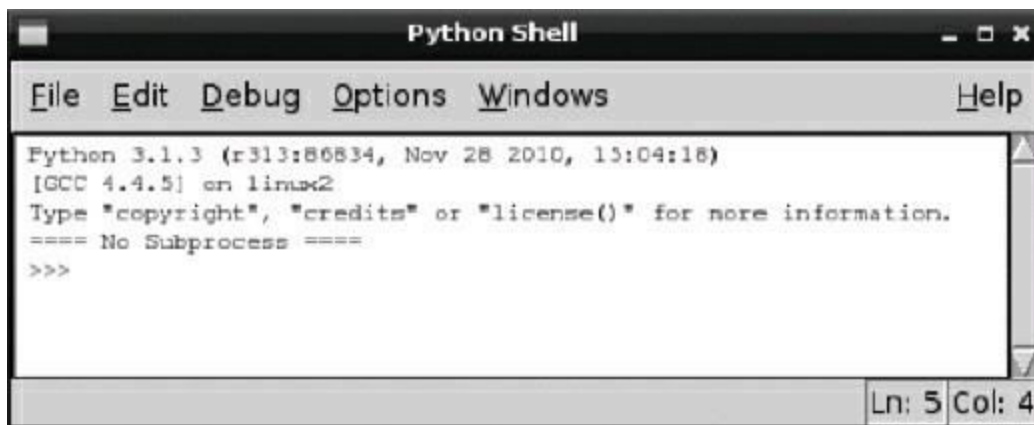


Figure 3-1 *IDLE and the Python Shell*

Python Versions

Python 3 was a major change over Python 2. This book is based on Python 3.1, but as you get further into Python you may find that some of the modules you want to use are not available for Python 3.

Python Shell

What you see in [Figure 3-1](#) is the Python Shell. This is the window where you type Python commands and see what they do. It is very useful for little experiments, especially while you're learning Python.

Rather like at the command prompt, you can type in commands after the prompt (in this case, `>>>`) and the Python console will show you what it has done on the line below.

Arithmetic is something that comes naturally to all programming languages, and Python is no exception. Therefore, type `2 + 2` after the prompt in the Python Shell and you should see the result `(4)` on the line below, as shown in [Figure 3-2](#).

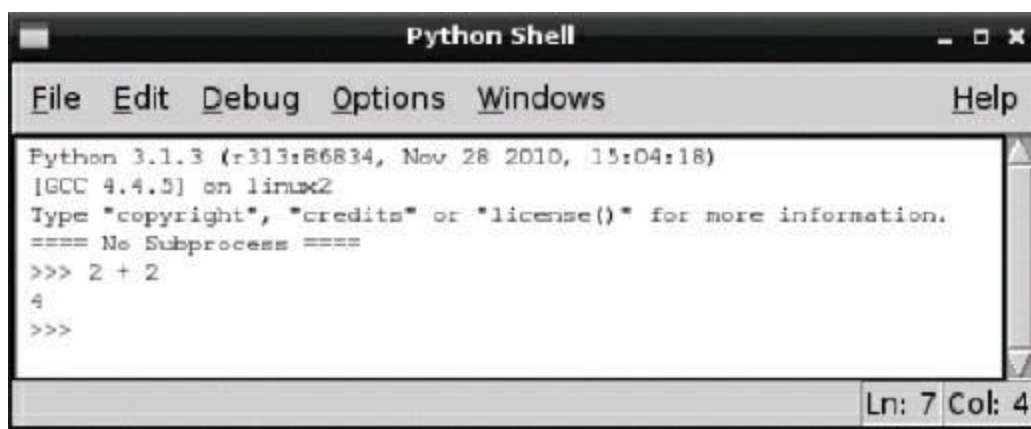


Figure 3-2 *Arithmetic in the Python Shell*

The Python Shell is a great place to experiment, but it is not the right place to write a program. Python programs are kept in files so that you do not have to retype them. A file may contain a long list of programming language commands, and when you want to run all the commands, what you actually do is run the file.

The menu bar at the top of IDLE allows us to create a new file. Therefore, select File and then New Window from the menu bar. [Figure 3-3](#) shows the IDLE Editor in a new window.

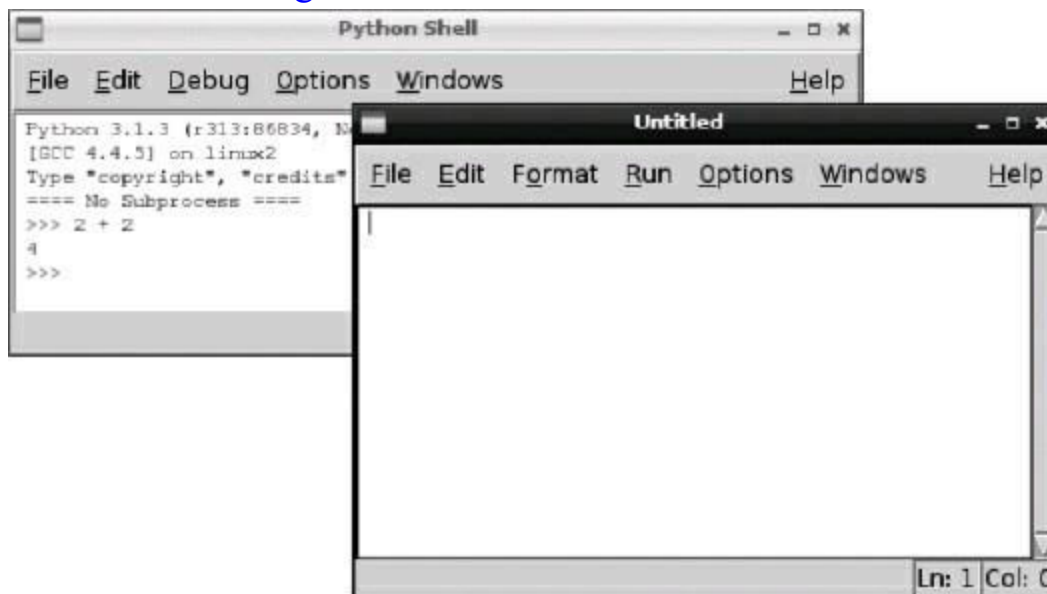


Figure 3-3 *The IDLE Editor*

Type the following two lines of code into IDLE:

```
print('Hello')
print('World')
```

You will notice that the editor does not have the `>>>` prompt. This is because what we write here will not be executed immediately; instead, it will just be stored in a file until we decide to run it. If you wanted, you could use nano or some other text editor to write the file, but the IDLE editor integrates nicely with Python. It also has some knowledge of the Python language and can thus serve as a memory aid when you are typing out programs.

We need a good place to keep all the Python programs we will be writing, so open the File Browser from the start menu (its under Accessories). Right-click over the main area and select New and then Folder from the pop-up menu (see [Figure 3-4](#)). Enter the name **Python** for the folder and press the RETURN key.

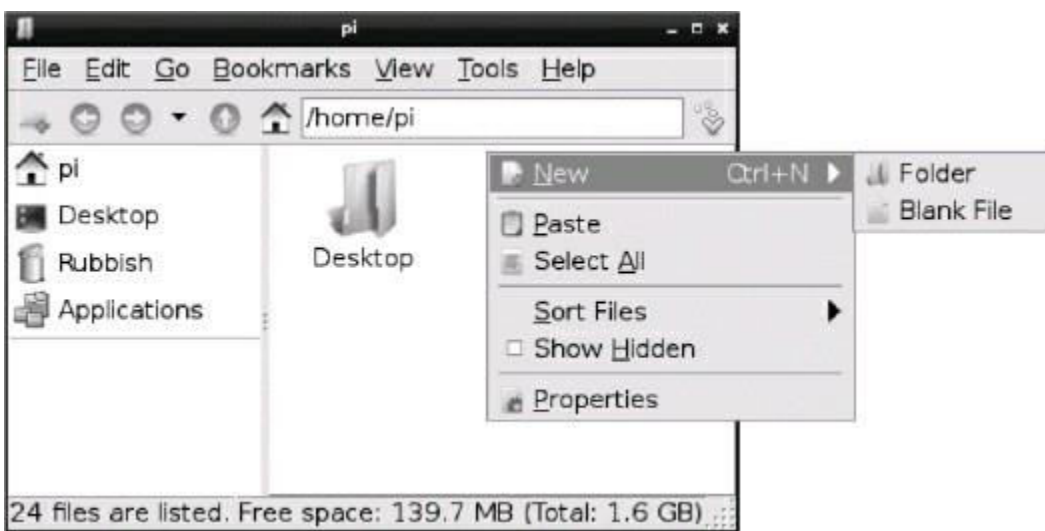


Figure 3-4 *Creating a Python folder*

Next, we need to switch back to our editor window and save the file using the File menu. Navigate to inside the new Python directory and give the file the name **hello.py**, as shown in [Figure 3-5](#).

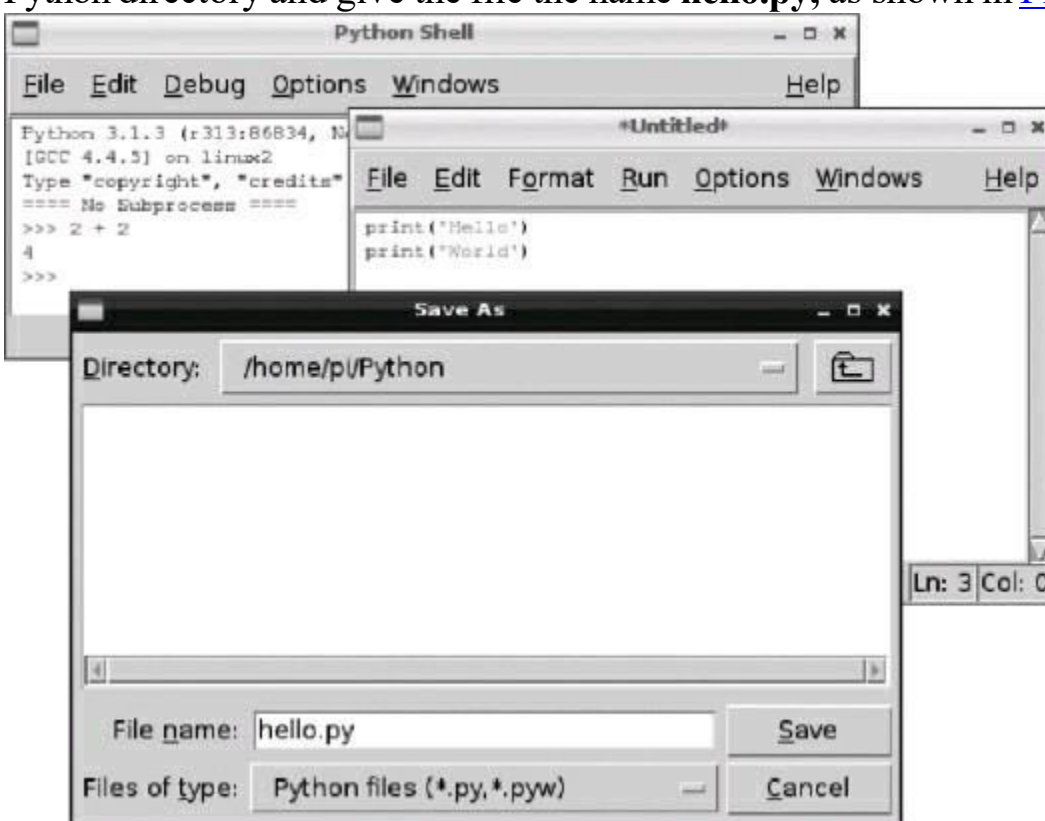


Figure 3-5 *Saving the program*

To actually run the program and see what it does, go to the Run menu and select Run Module. You should see the results of the program's execution in the Python Shell. It is no great surprise that the program prints the two words *Hello* and *World*, each on its own line.

What you type in the Python Shell does not get saved anywhere; therefore, if you exit IDLE and then start it up again, anything you typed in the Python Shell will be lost. However, because we saved our Editor file, we can load it at any time from the File menu.

NOTE *To save this book from becoming a series of screen dumps, from now on if I want you to type something in the Python Shell, I will proceed it with >>>. The results will then appear on the lines below it.*

Numbers

Numbers are fundamental to programming, and arithmetic is one of the things computers are very

good at. We will begin by experimenting with numbers, and the best place to experiment is the Python Shell.

Type the following into the Python Shell:

```
>>> 20 * 9 / 5 + 32
68.0
```

This isn't really advancing much beyond the 2 + 2 example we tried before. However, this example does tell us a few things:

- * means multiply.
- / means divide.
- Python does multiplication before division, and it does division before addition.

If you wanted to, you could add some parentheses to guarantee that everything happens in the right order, like this:

```
>>> (20 * 9 / 5) + 32
68.0
```

The numbers you have there are all whole numbers (or *integers* as they are called by programmers). We can also use a decimal point if we want to use such numbers. In programming, these kinds of numbers are called *floats*, which is short for *floating point*.

Variables

Sticking with the numbers theme for a moment, let's investigate variables. You can think of a variable as something that has a value. It is a bit like using letters as stand-ins for numbers in algebra. To begin, try entering the following:

```
>>> k = 9.0 / 5.0
```

The equals sign assigns a value to a variable. The variable must be on the left side and must be a single word (no spaces); however, it can be as long as you like and can contain numbers and the underscore character (`_`). Also, characters can be upper- and lowercase. Those are the rules for naming variables; however, there are also conventions. The difference is that if you break the rules, Python will complain, whereas if you break the conventions, other programmers may snort derisively and raise their eyebrows.

The conventions for variables are that they should start with a lowercase letter and should use an underscore between what in English would be words (for instance, `number_of_chickens`). The examples in [Table 3-1](#) give you some idea of what is legal and what is conventional.

Variable Name	Legal	Conventional
x	Yes	Yes
X	Yes	No
number_of_chickens	Yes	Yes
number of chickens	No	No
numberOfChickens	Yes	No
NumberOfChickens	Yes	No
2beOrNot2b	No	No
toBeOrNot2b	Yes	No

Table 3-1 Naming Variables

Many other languages use a different convention for variable names called bumpy-case or camel-case, where the words are separated by making the start of each word (except the first one) uppercase (for example, `numberOfChickens`). You will sometimes see this in Python example code. Ultimately, if the code is just for your own use, then how the variable is written does not really matter, but if your code is going to be read by others, it's a good idea to stick to the conventions.

By sticking to the naming conventions, it's easy for other Python programmers to understand your

program.

If you do something Python doesn't like or understand, you will get an error message. Try entering the following:

```
>>> 2beOrNot2b = 1
SyntaxError: invalid syntax
```

This is an error because you are trying to define a variable that starts with a digit, which is not allowed.

A little while ago, we assigned a value to the variable `k`. We can see what value it has by just entering `k`, like so:

```
>>> k
1.8
```

Python has remembered the value of `k`, so we can now use it in other expressions. Going back to our original expression, we could enter the following:

```
>>> 20 * k + 32
68.0
```

For Loops

Arithmetic is all very well, but it does not make for a very exciting program. Therefore, in this section you will learn about *looping*, which means telling Python to perform a task a number of times rather than just once. In the following example, you will need to enter more than one line of Python. When you press RETURN and go to the second line, you will notice that Python is waiting. It has not immediately run what you have typed because it knows that you have not finished yet. The `:` character at the end of the line means that there is more to do.

These extra tasks must each appear on an indented line. Therefore, in the following program, at the start of the second line you'll press TAB once and then type `print (x)`. To get this two-line program to actually run, press RETURN twice after the second line is entered.

```
>>> for x in range(1, 10):
    print(x)
```

```
1
2
3
4
5
6
7
8
9
>>>
```

This program has printed out the numbers between 1 and 9 rather than 1 and 10. The `range` command has an exclusive end point—that is, it doesn't include the last number in the range, but it does include the first.

You can check this out by just taking the range bit of the program and asking it to show its values as a list, like this:

```
>>> list(range(1, 10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Some of the punctuation here needs a little explaining. The parentheses are used to contain what are called *parameters*. In this case, `range` has two parameters: from (1) and to (10), separated by a comma.

The `for in` command has two parts. After the word `for` there must be a variable name. This variable will be assigned a new value each time around the loop. Therefore, the first time it will be 1, the next time 2, and so on. After the word `in`, Python expects to see something that works out to be a list of items. In this case, this is a list of the numbers between 1 and 9.

The `print` command also takes an argument that displays it in the Python Shell. Each time around the loop, the next value of `x` will be printed out.

Simulating Dice

We'll now build on what you just learned about loops to write a program that simulates throwing a die 10 times.

To do this, you will need to know how to generate a random number. So, first let's work out how to do that. If you didn't have this book, one way to find out how to generate a random number would be to type **random numbers python** into your search engine and look for fragments of code to type into the Python Shell. However, you do have this book, so here is what you need to write:

```
>>> import random
>>> random.randint(1,6)
2
```

Try entering the second line a few times, and you will see that you are getting different random numbers between 1 and 6.

The first line imports a library that tells Python how to generate numbers. You will learn much more about libraries later in this book, but for now you just need to know that we have to issue this command before we can start using the `randint` command that actually gives us a random number.

NOTE *I am being quite liberal with the use of the word command here. Strictly speaking, items such as `randint` are actually functions, not commands, but we will come to this later.*

Now that you can make a single random number, you need to combine this with your knowledge of loops to print off 10 random numbers at a time. This is getting beyond what can sensibly be typed into the Python Shell, so we will use the IDLE Editor.

You can either type in the examples from the text here or download all the Python examples used in the book from the book's website (www.raspberrypibook.com). Each programming example has a number. Thus, this program will be contained in the file `3_1_dice.py`, which can be loaded into the IDLE Editor.

At this stage, it is worth typing in the examples to help the concepts sink in. Open up a new IDLE Editor window, type the following into it, and then save your work:

```
#3_1_dice
import random
for x in range(1, 11):
    random_number = random.randint(1, 6)
    print(random_number)
```

The first line begins with a `#` character. This indicates that the entire line is not program code at all, but just a comment to anyone looking at the program. Comments like this provide a useful way of adding extra information about a program into the program file, without interfering with the operation of the program. In other words, Python will ignore any line that starts with `#`.

Now, from the Run menu, select Run Module. The result should look something like [Figure 3-6](#), where you can see the output in the Python Shell behind the Editor window.

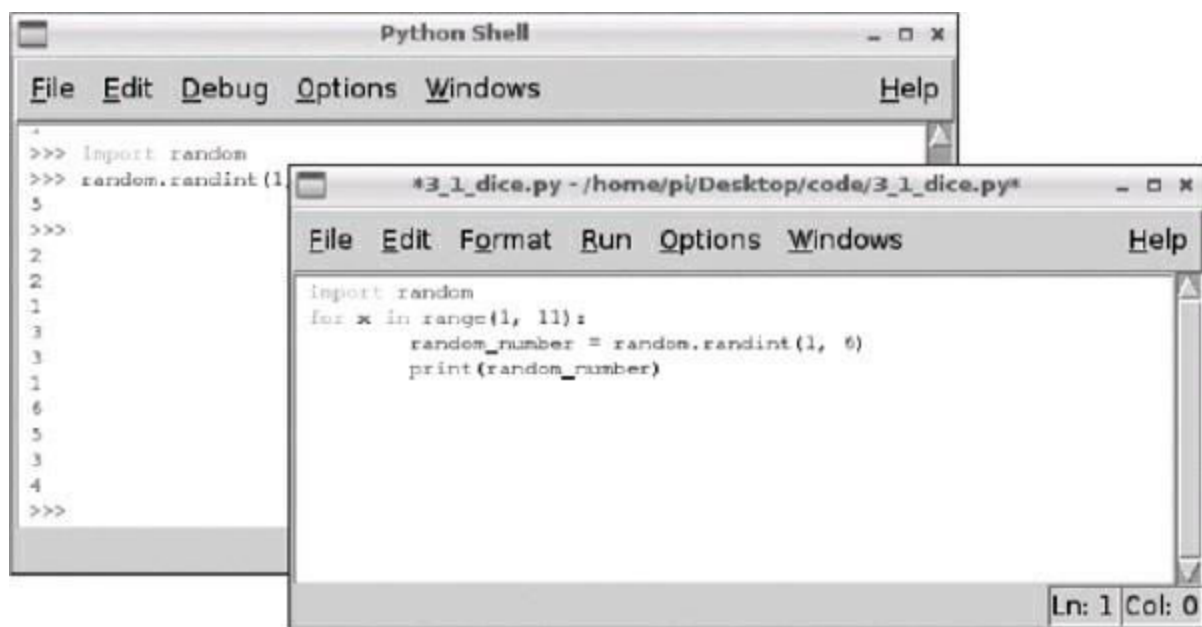


Figure 3-6 *The dice simulation*

If

Now it's time to spice up the dice program so that two dice are thrown, and if we get a total of 7 or 11, or any double, we will print a message after the throw. Type or load the following program into the IDLE Editor:

```
#3_2_double_dice
import random
for x in range(1, 11):
    throw_1 = random.randint(1, 6)
    throw_2 = random.randint(1, 6)
    total = throw_1 + throw_2
    print(total)
    if total == 7:
        print('Seven Thrown!')
    if total == 11:
        print('Eleven Thrown!')
    if throw_1 == throw_2:
        print('Double Thrown!')
```

When you run this program, you should see something like this:


```

6
7
Seven Thrown!
9
8
Double Thrown!
4
4
8
10
Double Thrown!
8
8
Double Thrown!

```

The first thing to notice about this program is that now two random numbers between 1 and 6 are generated. One for each of the dice. A new variable, `total`, is assigned to the sum of the two throws.

Next comes the interesting bit: the `if` command. The `if` command is immediately followed by a condition (in the first case, `total == 7`). There is then a colon (`:`), and the subsequent lines will only be executed by Python if the condition is true. At first sight, you might think there is a mistake in the condition because it uses `==` rather than `=`. The double equal sign is used when comparing items to see whether they are equal, whereas the single equal sign is used when assigning a value to a variable.

The second `if` is not tabbed in, so it will be executed regardless of whether the first `if` is true. This second `if` is just like the first, except that we are looking for a total of 11. The final `if` is a little different because it compares two variables (`throw_1` and `throw_2`) to see if they are the same, indicating that a double has been thrown.

Now, the next time you go to play *Monopoly* and find that the dice are missing, you know what to do: Just boot up your Raspberry Pi and write a little program.

Comparisons

To test to see whether two values are the same, we use `==`. This is called a *comparison operator*. The comparison operators we can use are shown in [Table 3-2](#).

Comparison	Description	Example
<code>==</code>	Equals	<code>total == 11</code>
<code>!=</code>	Not equals	<code>total != 11</code>
<code>></code>	Greater than	<code>total > 10</code>
<code><</code>	Less than	<code>total < 3</code>
<code>>=</code>	Greater than or equal to	<code>total >= 11</code>
<code><=</code>	Less than or equal to	<code>total <= 2</code>

Table 3-2 *Comparison Operators*

You can do some experimenting with these comparison operators in the Python Shell. Here's an example:

```

>>> 10 > 9
True

```

In this case, we have basically said to Python, "Is 10 greater than 9?" Python has replied, "True." Now let's ask Python whether 10 is less than 9:

```

>>> 10 < 9
False

```

Being Logical

You cannot fault the logic. When Python tells us “True” or “False,” it is not just displaying a message to us. `True` and `False` are special values called *logical values*. Any condition we use with an `if` statement will be turned into a logical value by Python when it is deciding whether or not to perform the next line.

These logical values can be combined rather like the way you perform arithmetic operations like plus and minus. It does not make sense to add `True` and `True`, but it does make sense sometimes to say `True AND True`.

As an example, if we wanted to display a message every time the total throw of our dice was between 5 and 9, we could write something like this:

```
if total >= 5 and total <= 9:
    print('not bad')
```

As well as `and`, we can use `or`. We can also use `not` to turn `True` into `False`, and vice versa, as shown here:

```
>>> not True
False
```

Thus, another way of saying the same thing would be to write the following:

```
if not (total < 5 or total > 9):
    print('not bad')
```

Exercise

Try incorporating the preceding test into the dice program. While you are at it, add two more `if` statements: one that prints “Good Throw!” if the throw is higher than 10 and one that prints “Unlucky!” if the throw is less than 4. Try your program out. If you get stuck, you can look at the solution in the file `3_3_double_dice_solution.py`.

Else

In the preceding example, you will see that some of the possible throws can be followed by more than one message. Any of the `if` lines could print an extra message if the condition is true. Sometimes you want a slightly different type of logic, so that if the condition is true, you do one thing and otherwise you do another. In Python, you use `else` to accomplish this:

```
>>> a = 7
>>> if a > 7:
    print('a is big')
else:
    print('a is small')
```

```
a is small
```

```
>>>
```

In this case, only one of the two messages will ever be printed.

Another variation on this is `elif`, which is short for *else if*. Thus, we could expand the previous example so that there are three mutually exclusive clauses, like this:

```
>>> a = 7
>>> if a > 9:
    print('a is very big')
elif a > 7:
    print('a is fairly big')
else:
    print('a is small')
```

a is small

```
>>>
```

While

Another command for looping is `while`, which works a little differently than `for`. The command `while` looks a bit like an `if` command in that it is immediately followed by a condition. In this case, the condition is for staying in the loop. In other words, the code inside the loop will be executed until the condition is no longer true. This means that you have to be careful to ensure that the condition will at some point be false; otherwise, the loop will continue forever and your program will appear to have hung.

To illustrate the use of `while`, the dice program has been modified so that it just keeps on rolling until a double 6 is rolled:

```
#3_4_double_dice_while
import random
throw_1 = random.randint(1, 6)
throw_2 = random.randint(1, 6)
while not (throw_1 == 6 and throw_2 == 6):
    total = throw_1 + throw_2
    print(total)
    throw_1 = random.randint(1, 6)
    throw_2 = random.randint(1, 6)
print('Double Six thrown!')
```

This program will work. Try it out. However, it is a little bigger than it should be. We are having to repeat the following lines twice—once before the loop starts and once inside the loop:

```
throw_1 = random.randint(1, 6)
throw_2 = random.randint(1, 6)
```

A well-known principle in programming is DRY (Don't Repeat Yourself). Although it's not a concern in a little program like this, as programs get more complex, you need to avoid the situation where the same code is used in more than one place, which makes the programs difficult to maintain.

We can use the command `break` to shorten the code and make it a bit “drier.” When Python encounters the command `break`, it breaks out of the loop. Here is the program again, this time using `break`:


```
#3_5_double_dice_while_break
import random
while True:
    throw_1 = random.randint(1, 6)
    throw_2 = random.randint(1, 6)
    total = throw_1 + throw_2
    print(total)
    if throw_1 == 6 and throw_2 == 6:
        break
print('Double Six thrown!')
```

The condition for staying in the loop is permanently set to `True`. The loop will continue until it gets to `break`, which will only happen after throwing a double 6.

Summary

You should now be happy to play with IDLE, trying things out in the Python Shell. I strongly recommend that you try altering some of the examples from this chapter, changing the code and seeing how that affects what the programs do.

In the next chapter, we will move on past numbers to look at some of the other types of data you can work with in Python.

4

Strings, Lists, and Dictionaries

This chapter could have had “and Functions” added to its title, but it was already long enough. In this chapter, you will first explore and play with the various ways of representing data and adding some structure to your programs in Python. You will then put everything you learned together into the simple game of Hangman, where you have to guess a word chosen at random by asking whether that word contains a particular letter.

The chapter ends with a reference section that tells you all you need to know about the most useful built-in functions for math, strings, lists, and dictionaries.

String Theory

No, this is not the Physics kind of String Theory. In programming, a *string* is a sequence of characters you use in your program. In Python, to make a variable that contains a string, you can just use the regular `=` operator to make the assignment, but rather than assigning the variable a number value, you assign it a string value by enclosing that value in single quotes, like this:

```
>>> book_name = 'Programming Raspberry Pi'
```

If you want to see the contents of a variable, you can do so either by entering just the variable name into the Python Shell or by using the `print` command, just as we did with variables that contain a number:

```
>>> book_name
'Programming Raspberry Pi'
>>> print(book_name)
Programming Raspberry Pi
```

There is a subtle difference between the results of each of these methods. If you just enter the variable name, Python puts single quotes around it so that you can tell it is a string. On the other hand, when you use `print`, Python just prints the value.

NOTE *You can also use double quotes to define a string, but the convention is to use single quotes unless you have a reason for using double quotes (for example, if the string you want to create has an apostrophe in it).*

You can find out how many characters a string has in it by doing this:

```
>>> len(book_name)
24
```

You can find the character at a particular place in the string like so:

```
>>> book_name[1]
'r'
```

Two things to notice here: first, the use of square brackets rather than the parentheses that are used for parameters and, second, that the positions start at 0 and not 1. To find the first letter of the string, you need to do the following:

```
>>> book_name[0]
'p'
```

If you put a number in that is too big for the length of the string, you will see this:

```
>>> book_name[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
```

This is an error, and it's Python's way of telling us that we have done something wrong. More

specifically, the “string index out of range” part of the message tells us that we have tried to access something that we can’t. In this case, that’s element 100 of a string that is only 24 characters long.

You can chop lumps out of a big string into a smaller string, like this:

```
>>> book_name[0:11]
'Programming'
```

The first number within the brackets is the starting position for the string we want to chop out, and the second number is not, as you might expect, the position of the last character you want, but rather the last character plus 1.

As an experiment, try and chop out the word *raspberry* from the title. If you do not specify the second number, it will default to the end of the string:

```
>>> book_name[12:]
'Raspberry Pi'
```

Similarly, if you do not specify the first number, it defaults to 0.

Finally, you can also join strings together by using + operator. Here’s an example:

```
>>> book_name + ' by Simon Monk'
'Programming Raspberry Pi by Simon Monk'
```

Lists

Earlier in the book when you were experimenting with numbers, a variable could only hold a single number. Sometimes, however, it is useful for a variable to hold a list of numbers or strings, or a mixture of both—or even a list of lists. [Figure 4-1](#) will help you to visualize what is going on when a variable is a list.

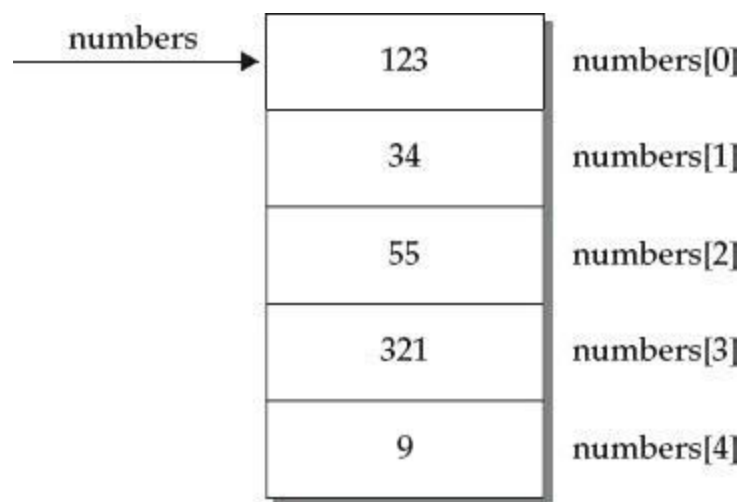


Figure 4-1 *An array*

Lists behave rather like strings. After all, a string is a list of characters. The following example shows you how to make a list. Notice how `len` works on lists as well as strings:

```
>>> numbers = [123, 34, 55, 321, 9]
>>> len(numbers)
5
```

Square brackets are used to indicate a list, and just like with strings we can use square brackets to find an individual element of a list or to make a shorter list from a bigger one:

```
>>> numbers[0]
123
>>> numbers[1:3]
[34, 55]
```

What’s more, you can use = to assign a new value to one of the items in the list, like this:

```
>>> numbers[0] = 1
>>> numbers
[1, 34, 55, 321, 9]
```

This changes the first element of the list (element 0) from 123 to just 1.

As with strings, you can join lists together using the `+` operator:

```
>>> more_numbers = [5, 66, 44]
>>> numbers + more_numbers
[1, 34, 55, 321, 9, 5, 66, 44]
```

If you want to sort the list, you can do this:

```
>>> numbers.sort()
>>> numbers
[1, 9, 34, 55, 321]
```

To remove an item from a list, you use the command `pop`, as shown next. If you do not specify an argument to `pop`, it will just remove the last element of the list and return it.

```
>>> numbers
[1, 9, 34, 55, 321]
>>> numbers.pop()
321
>>> numbers
[1, 9, 34, 55]
```

If you specify a number as the argument to `pop`, that is the position of the element to be removed.

Here's an example:

```
>>> numbers
[1, 9, 34, 55]
>>> numbers.pop(1)
9
>>> numbers
[1, 34, 55]
```

As well as removing items from a list, you can also insert an item into the list at a particular position. The function `insert` takes two arguments. The first is the position before which to insert, and the second argument is the item to insert.

```
>>> numbers
[1, 34, 55]
>>> numbers.insert(1, 66)
>>> numbers
[1, 66, 34, 55]
```

When you want to find out how long a list is, you use `len(numbers)`, but when you want to sort the list or “pop” an element off the list, you put a dot after the variable containing the list and then issue the command, like this:

```
numbers.sort()
```

These two different styles are a result of something called *object orientation*, which we will discuss in the next chapter.

Lists can be made into quite complex structures that contain other lists and a mixture of different types—numbers, strings, and logical values. [Figure 4-2](#) shows the list structure that results from the following line of code:

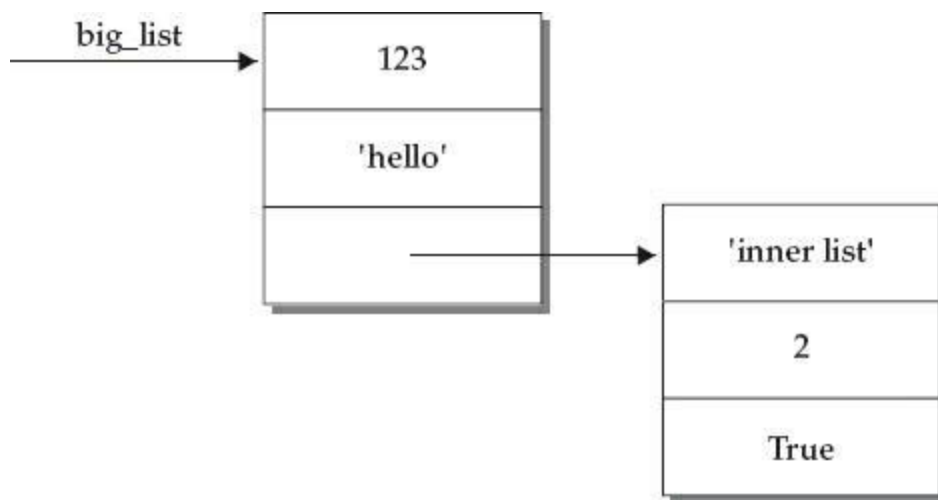


Figure 4-2 *A complex list*

```
>>> big_list = [123, 'hello', ['inner list', 2, True]]
>>> big_list
[123, 'hello', ['inner list', 2, True]]
```

You can combine what you know about lists with `for` loops and write a short program that creates a list and then prints out each element of the list on a separate line:

```
#4_1_list_and_for
list = [1, 'one', 2, True]
for item in list:
    print(item)
```

Here's the output of this program:

```
1
one
2
True
```

Functions

When you are writing small programs like the ones we have been writing so far, they only really perform one function, so there is little need to break them up. It is fairly easy to see what they are trying to achieve. As programs get larger, however, things get more complicated and it becomes necessary to break up your programs into units called *functions*. When we get even further into programming, we will look at better ways still of structuring our programs using classes and modules.

Many of the things I have been referring to as *commands* are actually functions that are built into Python. Examples of this are `range` and `print`.

The biggest problem in software development of any sort is managing complexity. The best programmers write software that is easy to look at and understand and requires very little in the way of extra explanation. Functions are a key tool in creating easy-to-understand programs that can be changed without difficulty or risk of the whole thing falling into a crumpled mess.

A function is a little like a program within a program. We can use it to wrap up a sequence of commands we want to do. A function that we define can be called from anywhere in our program and will contain its own variables and its own list of commands. When the commands have been run, we are returned to just after wherever it was in the code we called the function in the first place.

As an example, let's create a function that simply takes a string as an argument and adds the word *please* to the end of it. Load the following file—or even better, type it in to a new Editor window—and then run it to see what happens:

```
#4_2_polite_function
def make_polite(sentence):
    polite_sentence = sentence + ' please'
    return polite_sentence

print(make_polite('Pass the salt'))
```

The function starts with the keyword `def`. This is followed by the name of the function, which follows the same naming conventions as variables. After that come the parameters inside parentheses and separated by commas if there are more than one. The first line must end with a colon.

Inside the function, we are using a new variable called `polite_sentence` that takes the parameter passed into the function and adds “ please” to it (including the leading space). This variable can only be used from inside the function.

The last line of the function is a `return` command. This specifies what value the function should give back to the code that called it. This is just like trigonometric functions such as `sin`, where you pass in an angle and get back a number. In this case, what is returned is the value in the variable `polite_sentence`.

To use the function, we just specify its name and supply it with the appropriate arguments. A return value is not mandatory, and some functions will just do something rather than calculate something. For example, we could write a rather pointless function that prints “Hello” a specified number of times:

```
#4_3_hello_n
def say_hello(n):
    for x in range(0, n):
        print('Hello')
```

```
say_hello(5)
```

This covers the basics of what we will need to do to write our game of Hangman. Although you’ll need to learn some other things, we can come back to these later.

Hangman

Hangman is a word-guessing game, usually played with pen and paper. One player chooses a word and draws a dash for each letter of the word, and the other player has to guess the word. They guess a letter at a time. If the letter guessed is not in the word, they lose a life and part of the hangman’s scaffold is drawn. If the letter is in the word, all occurrences of the letter are shown by replacing the dashes with the letters.

We are going to let Python think of a word and we will have to guess what it is. Rather than draw a scaffold, Python is just going to tell us how many lives we have left.

You are going to start with how to give Python a list of words to chose from. This sounds like a job for a list of strings:

```
words = ['chicken', 'dog', 'cat', 'mouse', 'frog']
```

The next thing the program needs to do is to pick one of those words at random. We can write a function that does that and test it on its own:


```
#4_4_hangman_words
import random

words = ['chicken', 'dog', 'cat', 'mouse', 'frog']

def pick_a_word():
    word_position = random.randint(0, len(words) - 1)
    return words[word_position]

print(pick_a_word())
```

Run this program a few times to check that it is picking different words from the list.

This is a good start, but it needs to fit into the structure of the game. The next thing to do is to define a new variable called `lives_remaining`. This will be an integer that we can start off at 14 and decrease by 1 every time a wrong guess is made. This type of variable is called a *global* variable, because unlike variables defined in functions, we can access it from anywhere in the program.

As well as a new variable, we are also going to write a function called `play` that controls the game. We know what `play` should do, we just don't have all the details yet. Therefore, we can write the function `play` and make up other functions that it will call, such as `get_guess` and `process_guess`, as well as use the function `pick_a_word` we've just written. Here it is:

```
def play():
    word = pick_a_word()
    while True:
        guess = get_guess(word)
        if process_guess(guess, word):
            print('You win! Well Done!')
            break
        if lives_remaining == 0:
            print('You are Hung!')
            print('The word was: ' + word)
            break
```

A game of Hangman first involves picking a word. There is then a loop that continues until either the word is guessed (`process_guess` returns `True`) or `lives_remaining` has been reduced to zero. Each time around the loop, we ask the user for another guess.

We cannot run this at the moment because the functions `get_guess` and `process_guess` don't exist yet. However, we can write what are called *stubs* for them that will at least let us try out our `play` function. Stubs are just versions of functions that don't do much; they are stand-ins for when the full versions of the functions are written.

```
def get_guess(word):
    return 'a'

def process_guess(guess, word):
    global lives_remaining
    lives_remaining = lives_remaining - 1
    return False
```

The stub for `get_guess` just simulates the player always guessing the letter *a*, and the stub for `process_guess` always assumes that the player guessed wrong and, thus, decreases `lives_remaining` by 1 and returns `False` to indicate that they didn't win.

The stub for `process_guess` is a bit more complicated. The first line tells Python that the `lives_remaining` variable is the global variable of that name. Without that line, Python assumes that it is a new variable local to the function. The stub then reduces the lives remaining by 1 and returns `False` to indicate that the user has not won yet. Eventually, we will put in checks to see if the player has guessed all the letters or the whole word.

Open the file `4_5_hangman_play.py` and run it. You will get a result similar to this:

```
You are Hung!
```

```
The word was: dog
```

What happened here is that we have whizzed through all 14 guesses very quickly, and Python has told us what the word was and that we have lost.

All we need to do to complete the program is to replace the stub functions with real functions, starting with `get_guess`, shown here:

```
def get_guess(word):
    print_word_with_blanks(word)
    print('Lives Remaining: ' + str(lives_remaining))
    guess = input(' Guess a letter or whole word?')
    return guess
```

The first thing `get_guess` does is to tell the player the current state of their efforts at guessing (something like “c--c--n”) using the function `print_word`. This is going to be another stub function for now. The player is then told how many lives they have left. Note that because we want to append a number (`lives_remaining`) after the string `Lives Remaining:`, the number variable must be converted into a string using the built-in `str` function.

The built-in function `input` prints the message in its parameter as a prompt and then returns anything that the user types. Note that in Python 2, the `input` function was called `raw_input`. Therefore, if you decide to use Python 2, change this function to `raw_input`.

Finally, the `get_guess` function returns whatever the user has typed.

The stub function `print_word` just reminds us that we have something else to write later:

```
def print_word_with_blanks(word):
    print('print_word_with_blanks: not done yet')
```

Open the file `4_6_hangman_get_guess.py` and run it. You will get a result similar to this:

```
not done yet
Lives Remaining: 14
 Guess a letter or whole word?x
not done yet
Lives Remaining: 13
 Guess a letter or whole word?y
not done yet
Lives Remaining: 12
 Guess a letter or whole word?
```

Enter guesses until all your lives are gone to verify that you get the “losing” message.

Next, we can create the proper version of `print_word`. This function needs to display something like “c--c--n,” so it needs to know which letters the player has guessed and which they haven’t. To do this, it uses a new global variable (this time a string) that contains all the guessed letters. Every time a letter is guessed, it gets added to this string:

```
guessed_letters = ''
```

Here is the function itself:


```
def print_word_with_blanks(word):
    display_word = ''
    for letter in word:
        if guessed_letters.find(letter) > -1:
            # letter found
            display_word = display_word + letter
        else:
            # letter not found
            display_word = display_word + '-'
    print display_word
```

This function starts with an empty string and then steps through each letter in the word. If the letter is one of the letters that the player has already guessed, it is added to `display_word`; otherwise, a hyphen (-) is added. The built-in function `find` is used to check whether the letter is in the `guessed_letters`. The `find` function returns -1 if the letter is not there; otherwise, it returns the position of the letter. All we really care about is whether or not it is there, so we just check that the result is greater than -1. Finally, the word is printed out.

Currently, every time `process_guess` is called, it doesn't do anything with the guess because it's still a stub. We can make it a bit less of a stub by having it add the guessed letter to `guessed_letters`, like so:

```
def process_guess(guess, word):
    global lives_remaining
    global guessed_letters
    lives_remaining = lives_remaining - 1
    guessed_letters = guessed_letters + guess
    return False
```

Open the file `4_7_hangman_print_word.py` and run it. You will get a result something like this:

```
-----
Lives Remaining: 14
Guess a letter or whole word?c
c--c---
Lives Remaining: 13
Guess a letter or whole word?h
ch-c---
Lives Remaining: 12
Guess a letter or whole word?
```

It's starting to look like the proper game now. However, there is still the stub for `process_guess` to fill out. We will do that next:

```
def process_guess(guess, word):
    if len(guess) > 1:
        return whole_word_guess(guess, word)
    else:
        return single_letter_guess(guess, word)
```

When the player enters a guess, they have two choices: They can either enter a single-letter guess or attempt to guess the whole word. In this method, we just decide which type of guess it is and call either `whole_word_guess` or `single_letter_guess`. Because these functions are both pretty straightforward, we will implement them directly rather than as stubs:

```

def single_letter_guess(guess, word):
    global guessed_letters
    global lives_remaining
    if word.find(guess) == -1:
        # word guess was incorrect
        lives_remaining = lives_remaining - 1
    guessed_letters = guessed_letters + guess
    if all_letters_guessed(word):
        return True

def all_letters_guessed(word):
    for letter in word:
        if guessed_letters.find(letter) == -1:
            return False
    return True

```

The function `whole_word_guess` is actually easier than the `single_letter_guess` function:

```

def whole_word_guess(guess, word):
    global lives_remaining
    if guess.lower() == word.lower():
        return True
    else:
        lives_remaining = lives_remaining - 1
        return False

```

All we have to do is compare the guess and the actual word to see if they are the same when they are both converted to lowercase. If they are not the same, a life is lost. The function returns `True` if the guess was correct; otherwise, it returns `False`.

That's the complete program. Open up `4_8_hangman_full.py` in the IDLE Editor and run it. The full listing is shown here for convenience:

```

#04_08_hangman_full
import random

words = ['chicken', 'dog', 'cat', 'mouse', 'frog']
lives_remaining = 14
guessed_letters = ''

def play():
    word = pick_a_word()
    while True:
        guess = get_guess(word)
        if process_guess(guess, word):
            print('You win! Well Done!')
            break
        if lives_remaining == 0:
            print('You are Hung!')
            print('The word was: ' + word)
            break

```

```

def pick_a_word():
    word_position = random.randint(0, len(words) - 1)
    return words[word_position]

def get_guess(word):
    print_word_with_blanks(word)
    print('Lives Remaining: ' + str(lives_remaining))
    guess = input(' Guess a letter or whole word?')
    return guess

def print_word_with_blanks(word):
    display_word = ''
    for letter in word:
        if guessed_letters.find(letter) > -1:
            # letter found
            display_word = display_word + letter
        else:
            # letter not found
            display_word = display_word + '-'
    print(display_word)

def process_guess(guess, word):
    if len(guess) > 1:
        return whole_word_guess(guess, word)
    else:
        return single_letter_guess(guess, word)

def whole_word_guess(guess, word):
    global lives_remaining
    if guess == word:
        return True
    else:
        lives_remaining = lives_remaining - 1
        return False

def single_letter_guess(guess, word):
    global guessed_letters
    global lives_remaining
    if word.find(guess) == -1:
        # letter guess was incorrect
        lives_remaining = lives_remaining - 1
        guessed_letters = guessed_letters + guess
    if all_letters_guessed(word):
        return True
    return False

def all_letters_guessed(word):
    for letter in word:
        if guessed_letters.find(letter) == -1:
            return False
    return True

play()

```

play()

The game as it stands has a few limitations. First, it is case sensitive, so you have to enter your guesses in lowercase, the same as the words in the `words` array. Second, if you accidentally type **aa** instead of **a** as a guess, it will treat this as a whole-word guess, even though it is too short to be the whole word. The game should probably spot this and only consider guesses the same length as the

secret word to be whole-word guesses.

As an exercise, you might like to try and correct these problems. Hint: For the case-sensitivity problem, experiment with the built-in function `lower`. You can look at a corrected version in the file `4_8_hangman_full_solution.py`.

Dictionaries

Lists are great when you want to access your data starting at the beginning and working your way through, but they can be slow and inefficient when they get large and you have a lot of data to trawl through (for example, looking for a particular entry). It's a bit like having a book with no index or table of contents. To find what you want, you have to read through the whole thing.

Dictionaries, as you might guess, provide a more efficient means of accessing a data structure when you want to go straight to an item of interest. When you use a dictionary, you associate a value with a key. Whenever you want that value, you ask for it using the key. It's a little bit like how a variable name has a value associated with it; however, the difference is that with a dictionary, the keys and values are created while the program is running.

Let's look at an example:

```
>>> eggs_per_week = {'Penny': 7, 'Amy': 6, 'Bernadette': 0}
>>> eggs_per_week['Penny']
7
>>> eggs_per_week['Penny'] = 5
>>> eggs_per_week
{'Amy': 6, 'Bernadette': 0, 'Penny': 5}
>>>
```

This example is concerned with recording the number of eggs each of my chickens is currently laying. Associated with each chicken's name is a number of eggs per week. When we want to retrieve the value for one of the hens (let's say Penny), we use that name in square brackets instead of the index number that we would use with a list. We can use the same syntax in assignments to change one of the values.

For example, if Bernadette were to lay an egg, we could update our records by doing this:

```
eggs_per_week['Bernadette'] = 1
```

You may have noticed that when the dictionary is printed, the items in it are not in the same order as we defined them. The dictionary does not keep track of the order in which items were defined. Also note that although we have used a string as the key and a number as the value, the key could be a string, a number, or a tuple (see the next section), but the value could be anything, including a list or another dictionary.

Tuples

On the face of it, tuples look just like lists, but without the square brackets. Therefore, we can define and access a tuple like this:

```
>>> tuple = 1, 2, 3
>>> tuple
(1, 2, 3)
>>> tuple[0]
1
```

However, if we try to change an element of a tuple, we get an error message, like this one:

```
>>> tuple[0] = 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

The reason for this error message is that tuples are *immutable*, meaning that you cannot change them. Strings and numbers are the same. Although you can change a variable to refer to a different string, number, or tuple, you cannot change the number itself. On the other hand, if the variable references a list, you could alter that list by adding, removing, or changing elements in it.

So, if a tuple is just a list that you cannot do much with, you might be wondering why you would want to use one. The answer is, tuples provide a useful way of creating a temporary collection of items. Python lets you do a couple of next tricks using tuples, as described in the next two subsections.

Multiple Assignment

To assign a value to a variable, you just use = operator, like this:

`a = 1`
Python also lets you do multiple assignments in a single line, like this:

```
>>> a, b, c = 1, 2, 3
```

```
>>> a
```

```
1
```

```
>>> b
```

```
2
```

```
>>> c
```

```
3
```

Multiple Return Values

Sometimes in a function, you want to return more than one value at a time. As an example, imagine a function that takes a list of numbers and returns the minimum and the maximum. Here is such an example:

```
#04_09_stats
def stats(numbers):
    numbers.sort()
    return (numbers[0], numbers[-1])
```

```
list = [5, 45, 12, 1, 78]
```

```
min, max = stats(list)
```

```
print(min)
```

```
print(max)
```

This method of finding the minimum and maximum is not terribly efficient, but it is a simple example. The list is sorted and then we take the first and last numbers. Note that `numbers[-1]` returns the last number because when you supply a negative index to an array or string, Python counts backward from the end of the list or string. Therefore, the position `-1` indicates the last element, `-2` the second to last, and so on.

Exceptions

Python uses exceptions to flag that something has gone wrong in your program. Errors can occur in any number of ways while your program is running. A common way we have already discussed is when you try to access an element of a list or string that is outside of the allowed range. Here's an example:


```
>>> list = [1, 2, 3, 4]
>>> list[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

If someone gets an error message like this while they are using your program, they will find it confusing to say the least. Therefore, Python provides a mechanism for intercepting such errors and allowing you to handle them in your own way:

```
try:
    list = [1, 2, 3, 4]
    list[4]
except IndexError:
    print('Oops')
```

We cover exceptions again in the next chapter, where you will learn about the hierarchy of the different types of error that can be caught.

Summary of Functions

This chapter was written to get you up to speed with the most important features of Python as quickly as possible. By necessity, we have glossed over a few things and left a few things out. Therefore, this section provides a reference of some of the key features and functions available for the main types we have discussed. Treat it as a resource you can refer back to as you progress through the book, and be sure to try out some of the functions to see how they work. There is no need to go through everything in this section—just know that it is here when you need it. Remember, the Python Shell is your friend.

For full details of everything in Python, refer to <http://docs.python.org/py3k>.

Numbers

Table 4-1 shows some of the functions you can use with numbers.

Function	Description	Example
<code>abs(x)</code>	Returns the absolute value (removes the - sign).	<pre>>>>abs(-12.3) 12.3</pre>
<code>bin(x)</code>	Used to convert to binary string.	<pre>>>> bin(23) '0b10111'</pre>
<code>complex(r,i)</code>	Creates a complex number with real and imaginary components. Used in science and engineering.	<pre>>>> complex(2,3) (2+3j)</pre>
<code>hex(x)</code>	Used to convert to hexadecimal string.	<pre>>>> hex(255) '0xff'</pre>
<code>oct(x)</code>	Used to convert to octal string.	<pre>>>> oct(9) '0o11'</pre>
<code>round(x, n)</code>	Round x to n decimal places.	<pre>>>> round(1.111111, 2) 1.11</pre>
<code>math.factorial(n)</code>	Factorial function (as in $4 \times 3 \times 2 \times 1$).	<pre>>>> math.factorial (4) 24</pre>
<code>math.log(x)</code>	Natural logarithm.	<pre>>>> math.log(10) 2.302585092994046</pre>
<code>math.pow(x, y)</code>	Raises x to the power of y (alternatively, use <code>x ** y</code>).	<pre>>>> math.pow(2, 8) 256.0</pre>
<code>math.sqrt(x)</code>	Square root.	<pre>>>> math.sqrt(16) 4.0</pre>
<code>math.sin, cos, tan, asin, acos, atan</code>	Trigonometry functions (radians).	<pre>>>> math.sin(math.pi / 2) 1.0</pre>

Table 4-1 *Number Functions*

Strings

String constants can be enclosed either with single quotes (most common) or with double quotes. Double quotes are useful if you want to include single quotes in the string, like this:

```
s = "It's 3 o'clock"
```

On some occasions you'll want to include special characters such as end-of-lines and tabs into a string. To do this, you use what are called *escape characters*, which begin with a backslash (\) character. Here are the only ones you are likely to need:

- `\t` Tab character
- `\n` Newline character

Table 4-2 shows some of the functions you can use with strings.

Function	Description	Example
<code>s.capitalize()</code>	Capitalizes the first letter and makes the rest lowercase.	<pre>>>> 'aBc'.capitalize() 'Abc'</pre>
<code>s.center(width)</code>	Pads the string with spaces, centering it. An optional extra parameter is used for the fill character.	<pre>>>> 'abc'.center(10, '-') '---abc---</pre>
<code>s.endswith(str)</code>	Returns True if the end of the string matches.	<pre>>>> 'abcdef' .endswith('def') True</pre>
<code>s.find(str)</code>	Returns the position of a substring. Optional extra arguments for the start and end positions can be used to limit the search.	<pre>>>> 'abcdef'.find('de') 3</pre>
<code>s.format(args)</code>	Formats a string using template markers using <code>{}</code> .	<pre>>>> "It's {0} pm".format('12') "It's 12 pm"</pre>
<code>s.isalnum()</code>	Returns True if all the characters in the string are letters or digits.	<pre>>>> '123abc'.isalnum() True</pre>
<code>s.isalpha()</code>	Returns True if all the characters are alphabetic.	<pre>>>> '123abc'.isalpha() False</pre>
<code>s.isspace()</code>	Returns True if the character is a space, tab, or other whitespace character.	<pre>>>> '\t'.isspace() True</pre>
<code>s.ljust(width)</code>	Like <code>center()</code> , but left-justified.	<pre>>>> 'abc'.ljust(10, '-') 'abc-----'</pre>
<code>s.lower()</code>	Converts a string into lowercase.	<pre>>>> 'AbCdE'.lower() 'abcde'</pre>
<code>s.replace(old, new)</code>	Replaces all occurrences of old with new.	<pre>>>> 'hello world' .replace('world', 'there') 'hello there'</pre>
<code>s.split()</code>	Returns a list of all the words in the string, separated by spaces. An optional parameter can be used to indicate a different splitting character. The end of line character (<code>\n</code>) is a popular choice.	<pre>>>> 'abc def'.split() ['abc', 'def']</pre>
<code>s.splitlines()</code>	Splits the string on the newline character.	
<code>s.strip()</code>	Removes whitespace from both ends of the string.	<pre>>>> ' a b '.strip() 'a b'</pre>
<code>s.upper()</code>	Refer to <code>lower()</code> , earlier in this table.	

Table 4-2 *String Functions*

Lists

We have already looked at most of the features of lists. [Table 4-3](#) summarizes these features.

Function	Description	Example
<code>del(a[i:j])</code>	Deletes elements from the array, from element <code>i</code> to element <code>j-1</code> .	<pre>>>> a = ['a', 'b', 'c'] >>> del(a[1:2]) >>> a ['a', 'c']</pre>
<code>a.append(x)</code>	Appends an element to the end of the list.	<pre>>>> a = ['a', 'b', 'c'] >>> a.append('d') >>> a ['a', 'b', 'c', 'd']</pre>
<code>a.count(x)</code>	Counts the occurrences of a particular element.	<pre>>>> a = ['a', 'b', 'a'] >>> a.count('a') 2</pre>
<code>a.index(x)</code>	Returns the index position of the first occurrence of <code>x</code> in <code>a</code> . Optional parameters can be used for the start and end index.	<pre>>>> a = ['a', 'b', 'c'] >>> a.index('b') 1</pre>
<code>a.insert(i, x)</code>	Inserts <code>x</code> at position <code>i</code> in the list.	<pre>>>> a = ['a', 'c'] >>> a.insert(1, 'b') >>> a ['a', 'b', 'c']</pre>
<code>a.pop()</code>	Returns the last element of the list and removes it. An optional parameter lets you specify another index position for the removal.	<pre>>>> ['a', 'b', 'c'] >>> a.pop(1) 'b' >>> a ['a', 'c']</pre>
<code>a.remove(x)</code>	Removes the element specified.	<pre>>>> a = ['a', 'b', 'c'] >>> a.remove('c') >>> a ['a', 'b']</pre>
<code>a.reverse()</code>	Reverses the list.	<pre>>>> a = ['a', 'b', 'c'] >>> a.reverse() >>> a ['c', 'b', 'a']</pre>
<code>a.sort()</code>	Sorts the list. Advanced options are available when sorting lists of objects. See the next chapter for details.	

Table 4-3 *List Functions*

Dictionaries

[Table 4-4](#) details a few things about dictionaries that you should know.

Function	Description	Example
<code>len(d)</code>	Returns the number of items in the dictionary.	<pre>>>> d = {'a':1, 'b':2} >>> len(d) 2</pre>
<code>del(d[key])</code>	Deletes an item from the dictionary.	<pre>>>> d = {'a':1, 'b':2} >>> del(d['a']) >>> d {'b': 2}</pre>
<code>key in d</code>	Returns True if the dictionary (d) contains the key.	<pre>>>> d = {'a':1, 'b':2} >>> 'a' in d True</pre>
<code>d.clear()</code>	Removes all items from the dictionary.	<pre>>>> d = {'a':1, 'b':2} >>> d.clear() >>> d {}</pre>
<code>get(key, default)</code>	Returns the value for the key, or default if the key is not there.	<pre>>>> d = {'a':1, 'b':2} >>> d.get('c', 'c') 'c'</pre>

Table 4-4 Dictionary Functions

Type Conversions

We have already discussed the situation where we want to convert a number into a string so that we can append it to another string. Python contains some built-in functions for converting items of one type to another, as detailed in [Table 4-5](#).

Function	Description	Example
<code>float(x)</code>	Converts x to a floating-point number.	<pre>>>> float('12.34') 12.34 >>> float(12) 12.0</pre>
<code>int(x)</code>	Optional argument used to specify the number base.	<pre>>>> int(12.34) 12 >>> int('FF', 16) 255</pre>
<code>list(x)</code>	Converts x to a list. This is also a handy way to get a list of dictionaries keys.	<pre>>>> list('abc') ['a', 'b', 'c'] >>> d = {'a':1, 'b':2} >>> list(d) ['a', 'b']</pre>

Table 4-5 Type Conversions

Summary

Many things in Python you will discover gradually. Therefore, do not despair at the thought of learning all these commands. Doing so is really not necessary because you can always search for Python commands or look them up.

In the next chapter, we take the next step and see how Python manages object orientation.

In this chapter, we discuss how to make and use our own modules, like the `random` module we used in [Chapter 3](#). We also discuss how Python implements object orientation, which allows programs to be structured into classes, each responsible for its own behavior. This helps to keep a check on the complexity of our programs and generally makes them easier to manage. The main mechanisms for doing this are classes and methods. You have already used built-in classes and methods in earlier chapters without necessarily knowing it.

Modules

Most computer languages have a concept like modules that allows you to create a group of functions that are in a convenient form for others to use—or even for yourself to use on different projects.

Python does this grouping of functions in a very simple and elegant way. Essentially, any file with Python code in it can be thought of as a module with the same name as the file. However, before we get into writing our own modules, let's look at how we use the modules already installed with Python. Using Modules

When we used the `random` module previously, we did something like this:

```
>>> import random
>>> random.randint(1, 6)
6
```

The first thing we do here is tell Python that we want to use the `random` module by using the `import` command. Somewhere in the Python installation is a file called `random.py` that contains a `randint` function as well as some other functions.

With so many modules available to us, there is a real danger that different modules might have functions with the same name. In such a case, how would Python know which one to use? Fortunately, we do not have to worry about this happening because we have imported the module, and none of the functions in the module are visible unless we prepend the module name and then a dot onto the front of the function name. Try omitting the module name, like this:

```
>>> import random
>>> randint(1, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'randint' is not defined
```

Having to put the module name in front of every call to a function that's used a lot can get tedious. Fortunately, we can make this a little easier by adding to the `import` command as follows:

```
>>> import random as r
>>> r.randint(1, 6)
2
```

This gives the module a local name within our program of just `r` rather than `random`, which saves us a bit of typing.

If you are certain a function you want to use from a library is not going to conflict with anything in your program, you can take things a stage further, as follows:

```
>>> from random import randint
>>> randint(1, 6)
5
```

To go even further, you can import everything from the module in one fell swoop. Unless you know

exactly what is in the module, however, this is not normally a good idea, but you can do it. Here's how:

```
>>> from random import *
>>> randint(1, 6)
2
```

In this case, the asterisk (*) means “everything.”

Useful Python Libraries

So far we have used the `random` module, but other modules are included in Python. These modules are often called Python's *standard library*. There are too many of these modules to list in full. However, you can always find a complete list of Python modules at <http://docs.python.org/release/3.1.5/library/index.html>. Here are some of the most useful modules you should take a look at:

- **string** String utilities
- **datetime** For manipulating dates and times
- **math** Math functions (sin, cos, and so on)
- **pickle** For saving and restoring data structures on file (see [Chapter 6](#))
- **urllib.request** For reading web pages (see [Chapter 6](#))
- **tkinter** For creating graphical user interfaces (see [Chapter 7](#))

Installing New Modules

In addition to the standard library modules, thousands of modules have been contributed by the Python community. One very popular module is `pygame`, which we will use in [Chapter 8](#). It's often available as a binary package, so you can install it by typing something like this:

```
sudo apt-get install python-pygame
```

For many modules, however, this is not the case, and you have to go through a bit more effort to install them.

Any module good enough to use will be packaged in the standard Python way. This means that to install it, you need to download a compressed file containing a directory for the module. Let's use the `RPi.GPIO` module we will use in [Chapter 11](#) as an example. To install this module, you first go to the module's website, find the Downloads section, and then fetch the archive file. This is shown in [Figure 5-1](#). The next step is to save the file to a convenient directory (for this example, use the Python directory we created in [Chapter 3](#)).

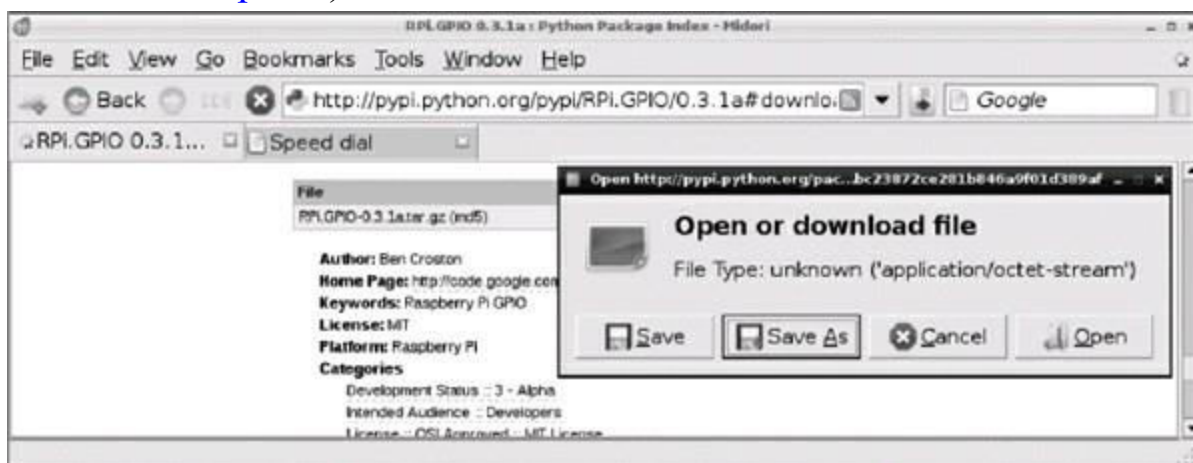


Figure 5-1 Downloading the `RPi.GPIO` module

Once the file has been saved, open LXTerminal and use `cd` to get to the Python directory, like so:

```
pi@raspberrypi:~/Python$ ls
RPi.GPIO-0.3.1a.tar.gz
```

Next, you need to extract the directory from the archive by entering the following command:

```
pi@raspberrypi:~/Python$ tar -xzf RPi.GPIO-0.3.1a.tar.gz
pi@raspberrypi:~/Python$ ls
RPi.GPIO-0.3.1a  RPi.GPIO-0.3.1a.tar.gz
```

Now that you have a new folder for the module, you need to “cd” into it and then run the `install` command. However, it is always worth checking the instructions first to see if there’s anything else you need to do. To see the instructions, type `more INSTALL.txt`.

Good thing you checked! The instructions state that you need to do the following:

```
sudo apt-get install python3-dev
```

Finally, you are ready to run the module installer itself:

```
pi@raspberrypi:~/Python$ cd RPi.GPIO-0.3.1a
pi@raspberrypi:~/Python/RPi.GPIO-0.3.1a$ sudo python3
setup.py install
```

Once the module is installed, you will be able to import it from the Python Shell.

Object Orientation

Object orientation has much in common with modules. It shares the same goals of trying to group related items together so that they are easy to maintain and find. As the name suggests, object orientation is about objects. We have been unobtrusively using objects already. A string is an object, for example. Thus, when we type

```
>>> 'abc'.upper()
```

We are telling the string `'abc'` that we want a copy of it, but in uppercase. In object-oriented terms, `abc` is an *instance* of the built-in class `str` and `upper` is a *method* on the class `str`.

We can actually find out the class of an object, as shown here (note double underscores before and after the word `class`):

```
>>> 'abc'.__class__
<class 'str'>
>>> [1].__class__
<class 'list'>
>>> 12.34.__class__
<class 'float'>
```

Defining Classes

That’s enough of other people’s classes; let’s make some of our own. We are going to start by creating a class that does the job of converting measurements from one unit to another by multiplying a value by a scale factor.

We will give the class the catchy name `ScaleConverter`. Here is the listing for the whole class, plus a few lines of code to test it:

```
#05_01_converter
class ScaleConverter:
```



```

def __init__(self, units_from, units_to, factor):
    self.units_from = units_from
    self.units_to = units_to
    self.factor = factor

def description(self):
    return 'Convert ' + self.units_from + ' to ' + self.units_to

def convert(self, value):
    return value * self.factor

```

```

c1 = ScaleConverter('inches', 'mm', 25)
print(c1.description())
print('converting 2 inches')
print(str(c1.convert(2)) + c1.units_to)

```

This requires some explanation. The first line is fairly obvious: It states that we are beginning the definition of a class called `ScaleConverter`. The colon (:) on the end indicates that all that follows is part of the class definition until we get back to an indent level of the left margin again.

Inside the `ScaleConverter`, we can see what look like three function definitions. These functions belong to the class; they cannot be used except via an instance of the class. These kinds of functions that belong to a class are called *methods*.

The first method, `__init__`, looks a bit strange—its name has two underscore characters on either side. When Python is creating a new instance of a class, it automatically calls the method `__init__`. The number of parameters that `__init__` should have depends on how many parameters are supplied when an instance of the class is made. To unravel that, we need to look at this line at the end of the file:

```
c1 = ScaleConverter('inches', 'mm', 25)
```

This line creates a new instance of the `ScaleConverter`, specifying what the units being converted from and to are, as well as the scaling factor. The `__init__` method must have all these parameters, but it must also have a parameter called `self` as the first parameter:

```
def __init__(self, units_from, units_to, factor):
```

The parameter `self` refers to the object itself. Now, looking at the body of the `__init__` method, we see some assignments:

```

self.units_from = units_from
self.units_to = units_to
self.factor = factor

```

Each of these assignments creates a variable that belongs to the object and has its initial value set from the parameters passed in to `__init__`.

To recap, when we create a new `ScaleConverter` by typing something like

```
c1 = ScaleConverter('inches', 'mm', 25)
```

Python creates a new instance of `ScaleConverter` and assigns the values 'inches', 'mm', and 25 to its three variables: `self.units_from`, `self.units_to`, and `self.factor`.

The term *encapsulation* is often used in discussions of classes. It is the job of a class to encapsulate everything to do with the class. That means storing data (like the three variables) and things that you might want to do with the data in the form of the `description` and `convert` methods.

The first of these (`description`) takes the information that the `Converter` knows about its units and creates a string that describes it. As with `__init__`, all methods must have a first parameter of `self`. The method will probably need it to access the data of the class to which it belongs.

Try it yourself by running program 05_01_converter.py and then typing the following in the Python Shell:

```
>>> silly_converter = ScaleConverter('apples', 'grapes', 74)
>>> silly_converter.description()
'Convert apples to grapes'
```

The `convert` method has two parameters: the mandatory `self` parameter and a parameter called `value`. The method simply returns the result of multiplying the value passed in by `self.scale`:

```
>>> silly_converter.convert(3)
222
```

Inheritance

The `ScaleConverter` class is okay for units of length and things like that; however, it would not work for something like converting temperature from degrees Celsius (C) to degrees Fahrenheit (F). The formula for this is $F = C * 1.8 + 32$. There is both a scale factor (1.8) and an offset (32).

Let's create a class called `ScaleAndOffsetConverter` that is just like `ScaleConverter`, but with a factor as well as an offset. One way to do this would simply be to copy the whole of the code for `ScaleConverter` and change it a bit by adding the extra variable. It might, in fact, look something like this:

```
#05_02_converter_offset_bad
class ScaleAndOffsetConverter:

    def __init__(self, units_from, units_to, factor, offset):
        self.units_from = units_from
        self.units_to = units_to
        self.factor = factor
        self.offset = offset

    def description(self):
        return 'Convert ' + self.units_from + ' to ' + self.units_to

    def convert(self, value):
        return value * self.factor + self.offset

c2 = ScaleAndOffsetConverter('C', 'F', 1.8, 32)
print(c2.description())
print('converting 20C')
print(str(c2.convert(20)) + c2.units_to)
```

Assuming we want both types of converters in the program we are writing, then this is a bad way of doing it. It's bad because we are repeating code. The `description` method is actually identical, and `__init__` is almost the same. A much better way is to use something called *inheritance*.

The idea behind inheritance in classes is that when you want a specialized version of a class that already exists, you inherit all the parent class's variables and methods and just add new ones or override the ones that are different. [Figure 5-2](#) shows a class diagram for the two classes, indicating how `ScaleAndOffsetConverter` inherits from `ScaleConverter`, adds a new variable (`offset`), and overrides the method `convert` (because it will work a bit differently).

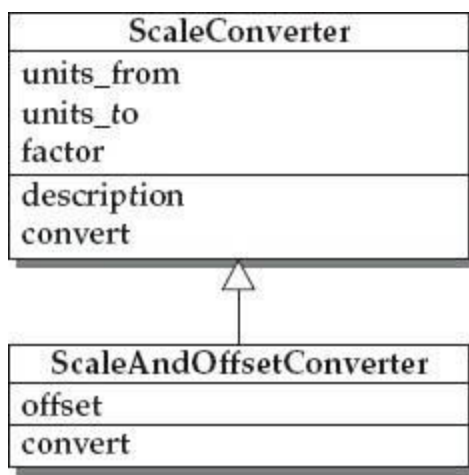


Figure 5-2 *An example of using inheritance*

Here is the class definition for `ScaleAndOffsetConverter` using inheritance:

```
class ScaleAndOffsetConverter(ScaleConverter):

    def __init__(self, units_from, units_to, factor, offset):
        ScaleConverter.__init__(self, units_from, units_to, factor)
        self.offset = offset

    def convert(self, value):
        return value * self.factor + self.offset
```

The first thing to notice is that the class definition for `ScaleAndOffsetConverter` has `ScaleConverter` in parentheses immediately after it. That is how you specify the parent class for a class.

The `__init__` method for the new “subclass” of `ScaleConverter` first invokes the `__init__` method of `ScaleConverter` before defining the new variable `offset`. The `convert` method will override the `convert` method in the parent class because we need to add on the offset for this kind of converter. You can run and experiment with the two classes together by running `05_03_converters_final.py`:

```
>>> c1 = ScaleConverter('inches', 'mm', 25)
>>> print(c1.description())
Convert inches to mm
>>> print('converting 2 inches')
converting 2 inches
>>> print(str(c1.convert(2)) + c1.units_to)
50mm
>>> c2 = ScaleAndOffsetConverter('C', 'F', 1.8, 32)
>>> print(c2.description())
Convert C to F
>>> print('converting 20C')
converting 20C
>>> print(str(c2.convert(20)) + c2.units_to)
68.0F
```

It’s a simple matter to convert these two classes into a module that we can use in other programs. In fact, we will use this module in [Chapter 7](#), where we attach a graphical user interface to it.

To turn this file into a module, we should first take the test code off the end of it and then give the file a more sensible name. Let’s call it `converters.py`. You will find this file in the downloads for this book. The module must be in the same directory as any program that wants to use it.

To use the module now, just do this:

```
>>> import converters
>>> c1 = converters.ScaleConverter('inches', 'mm', 25)
>>> print(c1.description())
Convert inches to mm
>>> print('converting 2 inches')
converting 2 inches
>>> print(str(c1.convert(2)) + c1.units_to)
50mm
```

Summary

Lots of modules are available for Python, and some are specifically for the Raspberry Pi, such as the `RPi.GPIO` library for controlling the GPIO pins. As you work through this book, you will encounter various modules. You will also find that as the programs you write get more complex, the benefits of an object-oriented approach to designing and coding your projects will keep everything more manageable.

In the next chapter, we look at using files and the Internet.

Python makes it easy for your programs to use files and connect to the Internet. You can read data from files, write data to files, and fetch content from the Internet. You can even check for new mail and tweet—all from your program.

Files

When you run a Python program, any values you have in variables will be lost. Files provide a means of making data more permanent.

Reading Files

Python makes reading the contents of a file extremely easy. As an example, we can convert the Hangman program from [Chapter 4](#) to read the list of words from a file rather than have them fixed in the program.

First of all, start a new file in IDLE and put some words in it, one per line. Then save the file with the name `hangman_words.txt` in the same directory as the Hangman program from [Chapter 4](#) (`04_08_hangman_full.py`). Note that in the Save dialog you will have to change the file type to `.txt` (see [Figure 6-1](#)).

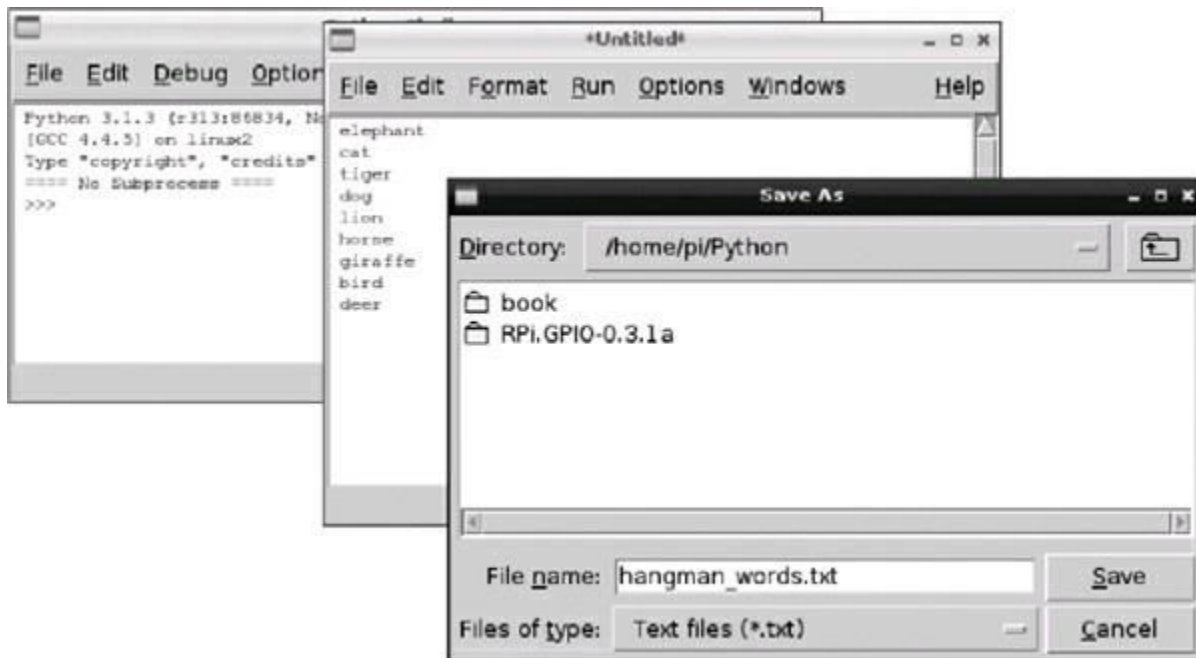


Figure 6-1 *Creating a text file in IDLE*

Before we modify the Hangman program itself, we can just experiment with reading the file in the Python console. Enter the following into the console:

```
>>> f = open('Python/hangman_words.txt')
```

Note that the Python console has a current directory of `/home/pi`, so the directory `Python` (or wherever you saved the file) must be included.

Next enter the following into the Python console:

```
>>> words = f.read()
>>> words
'elephant\ncat\ntiger\ndog\nlion\nhorse\ngiraffe\nbird\ndeer\n'
>>> words.splitlines()
['elephant', 'cat', 'tiger', 'dog', 'lion', 'horse', 'giraffe',
, 'bird', 'deer']
>>>
```

I told you it was easy! All we need to do to add this file to the Hangman program is replace the line

```
words = ['chicken', 'dog', 'cat', 'mouse', 'frog']
```

with the following lines:

```
f = open('hangman_words.txt')
words = f.read().splitlines()
f.close()
```

The line `f.close()` has been added. You should always call the `close` command when you are done with a file to free up operating system resources. Leaving a file open can lead to problems.

The full program is contained in the file `06_01_hangman_file.py`, and a suitable list of animal names can be found in the file `hangman_words.txt`. This program does nothing to check that the file exists before trying to read it. So, if the file isn't there, we get an error that looks something like this:

```
Traceback (most recent call last):
  File "06_01_hangman_file.py", line 4, in <module>
    f = open('hangman_words.txt')
IOError: [Errno 2] No such file or directory: 'hangman_words.txt'
```

To make this a bit more user friendly, the file-reading code needs to be inside a `try` command, like this:

```
try:
    f = open('hangman_words.txt')
    words = f.read().splitlines()
    f.close()
except IOError:
    print("Cannot find file 'hangman_words.txt'")
    exit()
```

Python will try to open the file, but because the file is missing it will not be able to. Therefore, the `except` part of the program will apply, and the more friendly message will be displayed. Because we cannot do anything without a list of words to guess, there is no point in continuing, so the `exit` command is used to quit.

In writing the error message, we have repeated the name of the file. Sticking strictly to the Don't Repeat Yourself (DRY) principle, the filename should be put in a variable, as shown next. That way, if we decide to use a different file, we only have to change the code in one place.

```
words_file = 'hangman_words.txt'
try:
    f = open(words_file)
    words = f.read().splitlines()
    f.close()
except IOError:
    print("Cannot find file: " + words_file)
    exit()
```

A modified version of Hangman with this code in it can be found in the file `06_02_hangman_file_try.py`.

Reading Big Files

The way we did things in the previous section is fine for a small file containing some words. However, if we were reading a really huge file (say, several megabytes), then two things would happen. First, it would take a significant amount of time for Python to read all the data. Second, because all the data is read at once, at least as much memory as the file size would be used, and for truly enormous files, that might result in Python running out of memory.

If you find yourself in the situation where you are reading a big file, you need to think about how

you are going to handle it. For example, if you were searching a file for a particular string, you could just read one line of the file at a time, like this:

```
#06_03_file_readline
words_file = 'hangman_words.txt'
try:
    f = open(words_file)
    line = f.readline()
    while line != '':
        if line == 'elephant\n':
            print('There is an elephant in the file')
            break
        line = f.readline()
    f.close()
except IOError:
    print("Cannot find file: " + words_file)
```

When the function `readline` gets to the last line of the file, it returns an empty string (`''`). Otherwise, it returns the contents of the line, including the end-of-line character (`\n`). If it reads a blank line that is actually just a gap between lines and not the end of the file, it will return just the end-of-line character (`\n`). By the program only reading one line at a time, the memory being used is only ever equivalent to one full line.

If the file is not broken into convenient lines, you can specify an argument in `read` that limits the number of characters read. For example, the following will just read the first 20 characters of a file:

```
>>> f = open('hangman_words.txt')
>>> f.read(20)
'elephant\ncat\ntiger\nd'
>>> f.close()
```

Writing Files

Writing files is almost as simple. When a file is opened, as well as specifying the name of the file to open, you can also specify the mode in which to open the file. The mode is represented by a character, and if no mode is specified it is assumed to be `r` for `read`. The modes are as follows:

- **r (read).**
- **w (write)** Replaces the contents of any existing file with that name.
- **a (append)** Appends anything to be written onto the end of an existing file.
- **r+** Opens the file for both reading and writing (not often used).

To write a file, you open it with a second parameter of `'w'`, `'a'`, or `'r+'`. Here's an example:

```
>>> f = open('test.txt', 'w')
>>> f.write('This file is not empty')
>>> f.close()
```

The File System

Occasionally, you will need to do some file-system-type operations on files (moving them, copying them, and so on). Python uses Linux to perform these actions, but provides a nice Python-style way of doing them. Many of these functions are in the `shutil` (shell utility) package. There's a number of subtle variations on the basic copy and move features that deal with file permissions and metadata. In this section, we just deal with the basic operations. You can refer to the official Python documentation for any other functions (<http://docs.python.org/release/3.1.5/library>).

Here's how to copy a file:

```
>>> import shutil
>>> shutil.copy('test.txt', 'test_copy.txt')
```

To move a file, either to change its name or move it to a different directory:

```
shutil.move('test_copy.txt', 'test_dup.txt')
```

This works on directories as well as files. If you want to copy an entire folder—including all its contents and its content's contents—you can use the function `copytree`. The rather dangerous function `rmtree`, on the other hand, will recursively remove a directory and all its contents—exercise extreme caution with this one!

The nicest way of finding out what is in a directory is via *globbing*. The package `glob` allows you to create a list of files in a directory by specifying a wildcard (*). Here's an example:

```
>>> import glob
glob.glob('*.txt')
['hangman_words.txt', 'test.txt', 'test_dup.txt']
```

If you just want all the files in the folder, you could use this:

```
glob.glob('*')
```

Pickling

Pickling involves saving the contents of a variable to a file in such a way that the file can be later loaded to get the original value back. The most common reason for wanting to do this is to save data between runs of a program. As an example, we can create a complex list containing another list and various other data objects and then pickle it into a file called `mylist.pickle`, like so:

```
>>> mylist = ['a', 123, [4, 5, True]]
>>> mylist
['a', 123, [4, 5, True]]
>>> import pickle
>>> f = open('mylist.pickle', 'w')
>>> pickle.dump(mylist, f)
>>> f.close()
```

If you find the file and open it in an editor to have a look, you will see something cryptic that looks like this:

```
(lp0
S'a'
p1
aI123
a(lp2
I4
aI5
aI01
aa.
```

That is to be expected; it is text, but it is not meant to be in human-readable form. To reconstruct a pickle file into an object, here is what you do:

```
>>> f = open('mylist.pickle')
>>> other_array = pickle.load(f)
>>> f.close()
>>> other_array
['a', 123, [4, 5, True]]
```

Internet

Most applications use the Internet in one way or another, even if it is just to check whether a new version of the application is available to remind the user about. You interact with a web server by sending HTTP (Hypertext Transfer Protocol) requests to it. The web server then sends a stream of

text back as a response. This text will be HTML (Hypertext Markup Language), the language used to create web pages.

Try entering the following code into the Python console.

```
>>> import urllib.request
>>> u = 'http://www.amazon.com/s/ref=nb_sb_noss?field-keywords=raspberrypi'
>>> f = urllib.request.urlopen(u)
>>> contents = f.read()
... lots of HTML
>>> f.close()
```

Note that you will need to execute the `read` line as soon as possible after opening the URL. What you have done here is to send a web request to www.amazon.com, asking it to search on “raspberrypi.” This has sent back the HTML for Amazon’s web page that would display (if you were using a browser) the list of search results.

If you look carefully at the structure of this web page, you can see that you can use it to provide a list of Raspberry Pi–related items found by Amazon. If you scroll around the text, you will find some lines like these:

```
<div class="productTitle"><a href="http://www.amazon
.com/Raspberry-User-Guide
-Gareth-Halfacree/dp/111846446X"> Raspberry Pi User Guide</a> <span
class="ptBrand">by <a href="/Gareth-Halfacree/e
/B0088CA5ZM">Gareth
Halfacree</a> and Eben Upton</span><span
class="binding"> (<span class
="format">Paperback</span> - Nov. 13, 2012)</span></div>
```

The key thing here is `<div class="productTitle">`. There is one instance of this before each of the search results. (It helps to have the same web page open in a browser for comparison.) What you want to do is copy out the actual title text. You could do this by finding the position of the text `productTitle`, counting two `>` characters, and then taking the text from that position until the next `<` character, like so:

```

#06_04_amazon_scraping
import urllib.request

u = 'http://www.amazon.com/s/ref=nb_sb_noss?field-
keywords=raspberrypi'
f = urllib.request.urlopen(u)
contents = str(f.read())
f.close()
i = 0
while True:
    i = contents.find('productTitle', i)
    if i == -1:
        break
    # Find the next two '>' after 'productTitle'
    i = contents.find('>', i+1)
    i = contents.find('>', i+1)
    # Find the first '<' after the two '>'
    j = contents.find('<', i+1)
    title = contents[i+2:j]
    print(title)

```

When you run this, you will mostly get a list of products. If you really get into this kind of thing, then search for “Regular Expressions in Python” on the Internet. Regular expressions are almost a language in their own right; they are used for doing complex searches and validations of text. They are not easy to learn or use, but they can simplify tasks like this one.

What we have done here is called *web scraping*, and it is not ideal for a number of reasons. First of all, organizations often do not like people “scraping” their web pages with automated programs. Therefore, you may get a warning or even banned from some sites.

Second, this action is very dependent on the structure of the web page. One tiny change on the website and everything could stop working. A much better approach is to look for an official web service interface to the site. Rather than returning the data as HTML, these services return much more easily processed data, often in XML or JSON format.

If you want to learn more about how to do this kind of thing, search the Internet for “web services in Python.”

Summary

This chapter has given you the basics of how to use files and access web pages from Python. There is actually a lot more to Python and the Internet, including accessing e-mail and other Internet protocols. For more information on this, have a look at the Python documentation at <http://docs.python.org/release/3.1.5/library/internet.html>.

Everything we have done so far has been text based. In fact, our Hangman game would not have looked out of place on a 1980s home computer. This chapter shows you how to create applications with a proper graphical user interface (GUI).

Tkinter

Tkinter is the Python interface to the Tk GUI system. Tk is not specific to Python; there are interfaces to it from many different languages, and it runs on pretty much any operating system, including Linux. Tkinter comes with Python, so there is no need to install anything. It is also the most commonly used tool for creating a GUI for Python.

Hello World

Tradition dictates that the first program you write with a new language or system should do something trivial, just to show it works! This usually means making the program display a message of “Hello World.” As you’ll recall, we already did this for Python back in [Chapter 3](#), so I’ll make no apologies for starting with this program:

```
#07_01_hello.py

from tkinter import *
root = Tk()
Label(root, text='Hello World').pack()
root.mainloop()
```

[Figure 7-1](#) shows the rather unimpressive application.

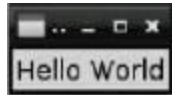


Figure 7-1 *Hello World in Tkinter*

You don’t need to worry about how all this works. You do, however, need to know that you must assign a variable to the object `Tk`. Here, we call this variable `root`, which is a common convention. We then create an instance of the class `Label`, whose first argument is `root`. This tells Tkinter that the label belongs to it. The second argument specifies the text to display in the label. Finally, the method `pack` is called on the label. This tells the label to pack itself into the space available. The method `pack` controls the layout of the items in the window. Shortly, we will use an alternative type of layout for the components in a grid.

Temperature Converter

To get started with Tkinter, you’ll gradually build up a simple application that provides a GUI for temperature conversion (see [Figure 7-2](#)). This application will use the `converter` module we created in [Chapter 5](#) to do the calculation.

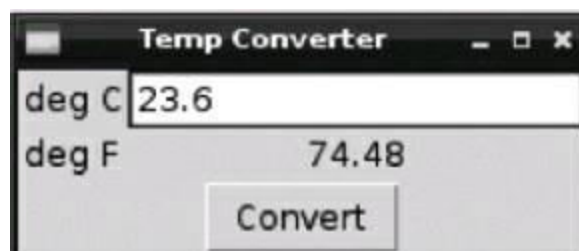


Figure 7-2 *A temperature conversion application*

Our Hello World application, despite being simple, is not well structured and would not lend itself

well to a more complex example. It is normal when building a GUI with Tkinter to use a class to represent each application window. Therefore, our first step is to make a framework in which to slot the application, starting with a window with the title “Temp Converter” and a single label:

```
#07_02_temp_framework.py

from tkinter import *

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        Label(frame, text='deg C').grid(row=0, column=0)
        button = Button(frame, text='Convert', command=self.convert)
        button.grid(row=1)

    def convert(self):
        print('Not implemented')

root = Tk()
root.wm_title('Temp Converter')
app = App(root)
root.mainloop()
```

We have added a class to the program called `App`. It has an `__init__` method that is used when a new instance of `App` is created in the following line:

```
app = App(root)
```

We pass in the `Tk` root object to `__init__` where the user interface is constructed.

As with the Hello World example, we are using a `Label`, but this time rather than adding the label to the root `Tk` object, we add the label to a `Frame` object that contains the label and other items that will eventually make up the window for our application. The structure of the user interface is shown in [Figure 7-3](#). Eventually, it will have all the elements shown.

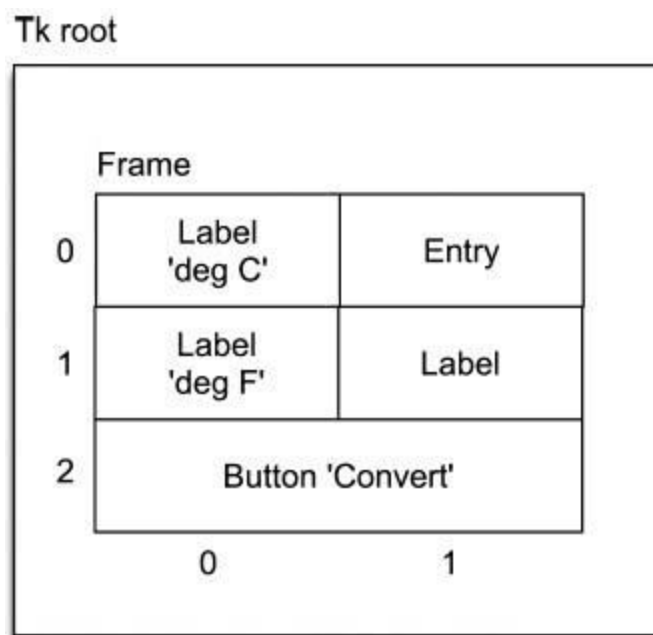


Figure 7-3 *Structure of the user interface*

The frame is “packed” into the root, but this time when we add the label, we use the method `grid` instead of `pack`. This allows us to specify a grid layout for the parts of our user interface. The field goes at position 0, 0 of the grid, and the button object that is created on the subsequent line is put on the second row of the grid (row 1). The button definition also specifies a “command” to be run when

the button is clicked. At the moment, this is just a stub that prints the message “Not implemented.”

The function `wm_title` sets the title of the window. [Figure 7-4](#) shows what the basic user interface looks like at this point.



Figure 7-4 *The basic user interface for the Temp Converter application*

The next step is to fill in the rest of the user interface. We need an “entry” into which a value for degrees C can be entered and two more labels. We need one permanent label that just reads “deg F” and a label to the right of it where the converted temperature will be displayed.

Tkinter has a special way of linking fields on the user interface with values. Therefore, when we need to get or set the value entered or displayed on a label or entry, we create an instance of a special variable object. This comes in various flavors, and the most common is `StringVar`. However, because we are entering and displaying numbers, we will use `DoubleVar`. *Double* means a double-precision floating-point number. This is just like a float, but more precise.

After we add in the rest of the user interface controls and the variables to interact with them, the program will look like this:

```
#07_03_temp_ui.py

from tkinter import *

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        Label(frame, text='deg C').grid(row=0, column=0)
        self.c_var = DoubleVar()
        Entry(frame, textvariable=self.c_var).grid(row=0, column=1)
        Label(frame, text='deg F').grid(row=1, column=0)
        self.result_var = DoubleVar()
        Label(frame, textvariable=self.result_var).grid(row=1, column=1)
        button = Button(frame, text='Convert', command=self.convert)
        button.grid(row=2, columnspan=2)

    def convert(self):
        print('Not implemented')

root = Tk()
root.wm_title('Temp Converter')
app = App(root)
root.mainloop()
```

The first `DoubleVar` (`c_var`) is assigned to the entry by specifying a `textvariable` property for it. This means that the entry will display what is in that `DoubleVar`, and if the value in the `DoubleVar` is changed, the field display will automatically update to show the new value. Also, when the user types something in the entry field, the value in the `DoubleVar` will change. Note that a new label of “deg F” has also been added.

The second `DoubleVar` is linked to another label that will eventually display the result of the calculation. We have added another attribute to the `grid` command that lays out the button. Because we specify `columnspan=2`, the button will stretch across both columns.

If you run the program, it will display the final user interface, but when you click the `Convert`

button, the message “Not Implemented” will be written to the Python console.

The last step is to replace the stubbed-out “convert” method with a real method that uses the `converters` module from [Chapter 5](#). To do this, we need to import the module. In order to reduce how much we need to type, we will import everything, as follows:

```
from converters import *
```

For the sake of efficiency, it is better if we create a single “converter” during `__init__` and just use the same one every time the button is clicked. Therefore, we create a variable called `self.t_conv` to reference the convertor. The `convert` method then just becomes this:

```
def convert(self):
    c = self.c_var.get()
    self.result_var.set(self.t_conv.convert(c))
```

Here is the full listing of the program:

#07_04_temp_final.py

```
from tkinter import *
from converters import *
```

```
class App:
```

```
    def __init__(self, master):
        self.t_conv = ScaleAndOffsetConverter('C', 'F', 1.8, 32)
        frame = Frame(master)
        frame.pack()
        Label(frame, text='deg C').grid(row=0, column=0)
        self.c_var = DoubleVar()
        Entry(frame, textvariable=self.c_var).grid(row=0, column=1)
        Label(frame, text='deg F').grid(row=1, column=0)
        self.result_var = DoubleVar()
        Label(frame, textvariable=self.result_var).grid(row=1, column=1)
        button = Button(frame, text='Convert', command=self.convert)
        button.grid(row=2, columnspan=2)

    def convert(self):
        c = self.c_var.get()
        self.result_var.set(self.t_conv.convert(c))
```

```
root = Tk()
root.wm_title('Temp Converter')
app = App(root)
root.mainloop()
```

Other GUI Widgets

In the temperature converter, we just used text fields (class `Entry`) and labels (class `Label`). As you would expect, you can build lots of other user interface controls into your application. [Figure 7-5](#) shows the main screen of a “kitchen sink” application that illustrates most of the controls you can use in Tkinter. This program is available as `07_05_kitchen_sink.py`.



Figure 7-5 A “kitchen sink” application

Checkbutton

The Checkbox widget (first column, second row of [Figure 7-5](#)) is created like this:

```
Checkbutton(frame, text='Checkbutton')
```

This line of code just creates a Checkbutton with a label next to it. If we have gone to the effort of placing a check box on the window, we’ll also want a way of finding out whether or not it is checked.

The way to do this is to use a special “variable” like we did in the temperature converter example. In the following example, we use a StringVar, but if the values of `onvalue` and `offvalue` were numbers, we could use an IntVar instead.

```
check_var = StringVar()
check = Checkbutton(frame, text='Checkbutton',
                    variable=check_var, onvalue='Y', offvalue='N')
check.grid(row=1, column=0)
```

Listbox

To display a list of items from which one or multiple items can be selected, a Listbox is used (refer to the center of [Figure 7-5](#)). Here’s an example:

```
listbox = Listbox(frame, height=3, selectmode=BROWSE)
for item in ['red', 'green', 'blue', 'yellow', 'pink']:
    listbox.insert(END, item)
listbox.grid(row=1, column=1)
```

In this case, it just displays a list of colors. Each string has to be added to the list individually. The word `END` indicates that the item should go at the end of the list.

You can control the way selections are made on the Listbox using the `selectmode` property, which can be set to one of the following:

- **SINGLE** Only one selection at a time.
- **BROWSE** Similar to **SINGLE**, but allows selection using the mouse. This appears to be indistinguishable from **SINGLE** in Tkinter on the Pi.
- **MULTIPLE** SHIFT-click to select more than one row.
- **EXTENDED** Like **MULTIPLE**, but also allows the CTRL-SHIFT-click selection of ranges.

Unlike with other widgets that use StringVar or some other type of special variable to get values in and out, to find out which items of the Listbox are selected, you have to ask it using the method `curselection`. This returns a collection of selection indexes. Thus, if the first, second, and fourth items in the list are selected, you will get a list like this:

```
[0, 1, 3]
```

When `selectmode` is **SINGLE**, you still get a list back, but with just one value in it.

Spinbox

Spinboxes provide an alternative way of making a single selection from a list:

```
Spinbox(frame, values=('a', 'b', 'c')).grid(row=3)
```

The `get` method returns the currently displayed item in the Spinbox, not its selection index.

Layouts

Laying out the different parts of your application so that everything looks good, even when you resize the window, is one of the most tricky parts of building a GUI.

You will often find yourself putting one kind of layout inside another. For example, the overall shape of the “kitchen sink” application is a 3×3 grid, but within that grid is another frame for the two radio buttons:

```
radio_frame = Frame(frame)
radio_selection = StringVar()
b1 = Radiobutton(radio_frame, text='portrait',
    variable=radio_selection, value='P')
b1.pack(side=LEFT)
b2 = Radiobutton(radio_frame, text='landscape',
    variable=radio_selection, value='L')
b2.pack(side=LEFT)
radio_frame.grid(row=1, column=2)
```

This approach is quite common, and it is a good idea to sketch out the layout of your controllers on paper before you start writing the code.

One particular problem you will encounter when creating a GUI is controlling what happens when the window is resized. You will normally want to keep some widgets in the same place and at the same size, while allowing other widgets to expand.

As an example of this, we can build a simple window like the one shown in [Figure 7-6](#), which has a Listbox (on the left) that stays the same size and an expandable message area (on the right) that expands as the window is resized.

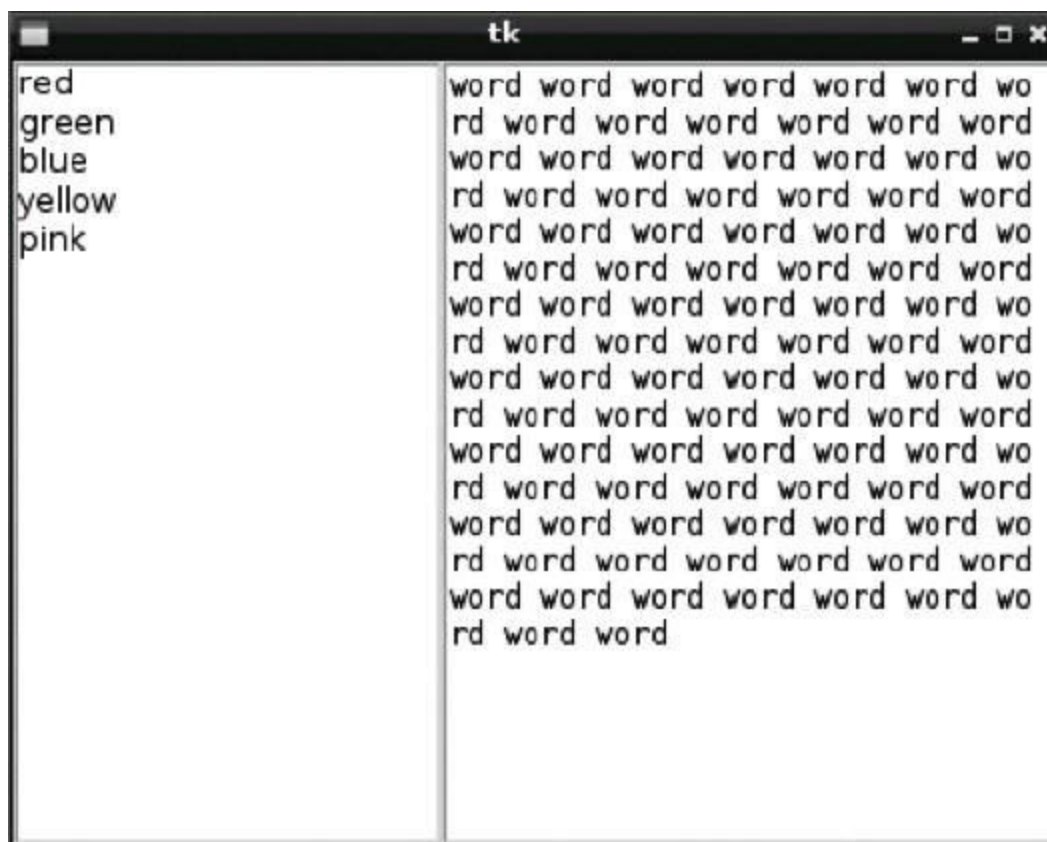


Figure 7-6 *An example of resizing a window*

The code for this is shown here:


```
#07_06_resizing.py
```

```
from tkinter import *
```

```
class App:
```

```
    def __init__(self, master):
        frame = Frame(master)
        frame.pack(fill=BOTH, expand=1)
        #Listbox
        listbox = Listbox(frame)
        for item in ['red', 'green', 'blue', 'yellow', 'pink']:
            listbox.insert(END, item)
        listbox.grid(row=0, column=0, sticky=W+E+N+S)

        #Message
        text = Text(frame, relief=SUNKEN)
        text.grid(row=0, column=1, sticky=W+E+N+S)
        text.insert(END, 'word ' * 100)
        frame.columnconfigure(1, weight=1)
        frame.rowconfigure(0, weight=1)
```

```
root = Tk()
app = App(root)
root.geometry("400x300+0+0")
root.mainloop()
```

The key to understanding such layouts is the use of the `sticky` attributes of the components to decide which walls of their grid cell they should stick to. To control which of the columns and rows expand when the window is resized, you use the `columnconfigure` and `rowconfigure` commands. [Figure 7-7](#) shows the arrangement of GUI components that make up this window. The lines indicate where the edge of a user interface item is required to “stick” to its containing wall.

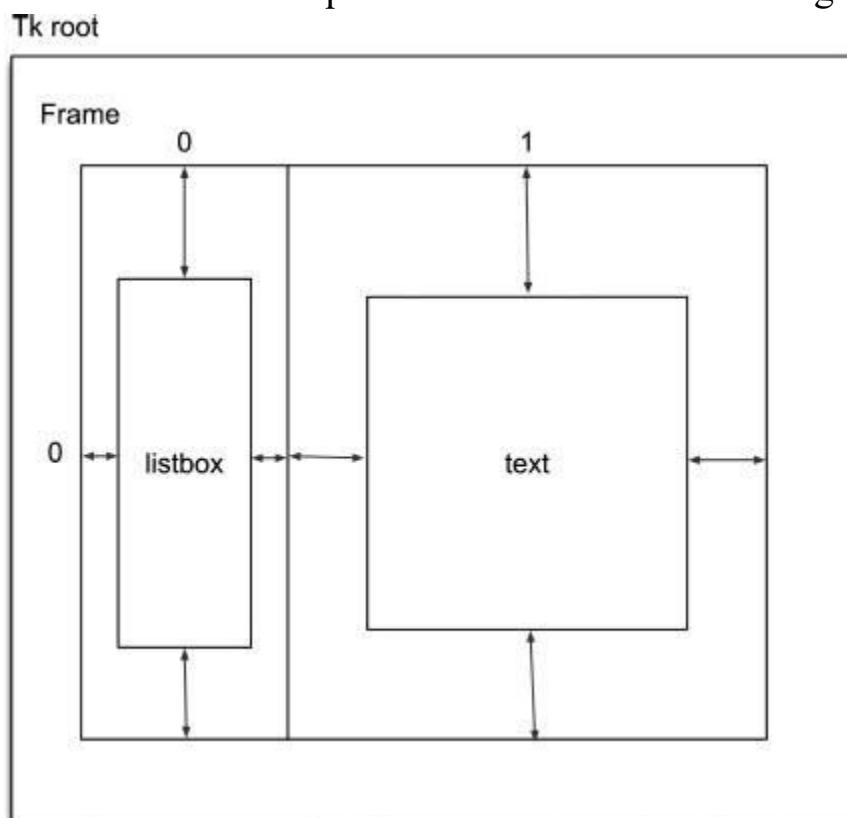


Figure 7-7 Layout for the resizing window example

Let's go through the code for this example so that things start to make sense. First, the line

```
frame.pack(fill=BOTH, expand=1)
```

ensures that the frame will fill the enclosing root window so that if the root window changes in size, so will the frame.

Having created the Listbox, we add it to the frame's grid layout using the following line:

```
listbox.grid(row=0, column=0, sticky=W+E+N+S)
```

This specifies that the Listbox should go in position row 0, column 0, but then the `sticky` attribute says that the west, east, north, and south sides of the Listbox should stay connected to the enclosing grid. The constants `W`, `E`, `N`, and `S` are numeric constants that can be added together in any order. The Text widget is added to the frame's grid in just the same way, and its content is initialized to the word *word* repeated 100 times.

The final part to the puzzle is getting the resizing behavior we want for a text area that expands to the right and a list area that doesn't. To do this, we use the `columnconfigure` and `rowconfigure` methods:

```
frame.columnconfigure(1, weight=1)
frame.rowconfigure(0, weight=1)
```

By default, rows and columns do not expand at all when their enclosing user interface element expands. We do not want column 0 to expand, so we can leave that alone. However, we do want column 1 to expand to the right, and we want row 0 (the only row) to be able to expand downward. We do this by giving them a "weight" using the `columnconfigure` and `rowconfigure` methods. If, for example, we had multiple columns that we want to expand evenly, we would give them the same weight (typically 1). If, however, we want one of the columns to expand at twice the rate of the other, we would give it twice the weight. In this case, we only have one column and one row that we need expanding, so they can both be given a weight of 1.

Scrollbar

If you shrink down the window for the program `07_06_resizing.py`, you will notice that there's no scrollbar to access text that's hidden. You can still get to the text, but clearly a scrollbar would help.

Scrollbars are widgets in their own right, and the trick for making them work with something like a Text, Message, or Listbox widget is to lay them out next to each other and then link them together.

Figure 7-8 shows a Text widget with a scrollbar.



Figure 7-8 Scrolling a Text widget

The code for this is as follows:

```
#07_07_scrolling.py
```

```
from tkinter import *

class App:

    def __init__(self, master):
        scrollbar = Scrollbar(master)
        scrollbar.pack(side=RIGHT, fill=Y)
        text = Text(master, yscrollcommand=scrollbar.set)
        text.pack(side=LEFT, fill=BOTH)
        text.insert(END, 'word ' * 1000)
        scrollbar.config(command=text.yview)

root = Tk()
root.wm_title('Scrolling')
app = App(root)
root.mainloop()
```

In this example, we use the `pack` layout, positioning the scrollbar on the right and the text area on the left. The `fill` attribute specifies that the Text widget is allowed to use all free space on *both* the X and Y dimensions.

To link the scrollbar to the Text widget, we set the `yscrollcommand` property of the Text widget to the `set` method of the scrollbar. Similarly, the `command` attribute of the scrollbar is set to `text.yview`.

Dialogs

It is sometimes useful to pop up a little window with a message and make the user click OK before they can do anything else (see [Figure 7-9](#)). These windows are called *modal dialogs*, and Tkinter has a whole range of them in the package `tkinter.messagebox`.



Figure 7-9 An alert dialog

The following example shows how to display such an alert. As well as `showinfo`, `tkinter.messagebox` also has the functions `showwarning` and `showerror` that work just the same, but display a different symbol in the window.

```
#07_08_gen_dialogs.py
```

```
from tkinter import *
import tkinter.messagebox as mb

class App:

    def __init__(self, master):
        b=Button(master, text='Press Me', command=self.info).pack()
```

```
def info(self):
    mb.showinfo('Information', "Please don't press that button again!")
```

```
root = Tk()
app = App(root)
root.mainloop()
```

Other kinds of dialogs can be found in the packages `tkinter.colorchooser` and `tkinter.filedialog`.

Color Chooser

The Color Chooser returns a color as separate RGB components as well as a standard hex color string (see [Figure 7-10](#)).



Figure 7-10 *The Color Chooser*

#07_09_color_chooser.py

```
from tkinter import *
import tkinter.colorchooser as cc

class App:

    def __init__(self, master):
        b=Button(master, text='Color..', command=self.ask_color).pack()

    def ask_color(self):
        (rgb, hx) = cc.askcolor()
        print("rgb=" + str(rgb) + " hx=" + hx)

root = Tk()
app = App(root)
root.mainloop()
```

This code returns something like this:

```
rgb=(255.99609375, 92.359375, 116.453125) hx=#ff5c74
```

File Chooser

File Choosers can be found in the package `tkinter.filedialog`. These follow exactly the same pattern as the other dialogs we have looked at.

Menus

You can give your applications menus. As an example, we can create a very simple application with an entry field and a couple of menu options (see [Figure 7-11](#)).

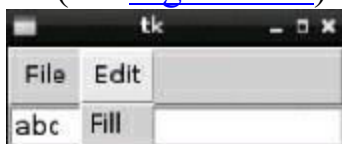


Figure 7-11 *Menus*


```
#07_10_menus.py
```

```
from tkinter import *

class App:

    def __init__(self, master):
        self.entry_text = StringVar()
        Entry(master, textvariable=self.entry_text).pack()

        menubar = Menu(root)

        filemenu = Menu(menubar, tearoff=0)
        filemenu.add_command(label='Quit', command=exit)
        menubar.add_cascade(label='File', menu=filemenu)

        editmenu = Menu(menubar, tearoff=0)
        editmenu.add_command(label='Fill', command=self.fill)
        menubar.add_cascade(label='Edit', menu=editmenu)

        master.config(menu=menubar)

    def fill(self):
        self.entry_text.set('abc')

root = Tk()
app = App(root)

root.mainloop()
```

The first step is to create a root `Menu`. This is the single object that will contain all the menus (File and Edit, in this case, along with all the menu options).

```
menubar = Menu(root)
```

To create the File menu, with its single option, Quit, we first create another instance of `Menu` and then add a command for Quit and finally add the File menu to the root `Menu`:

```
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label='Quit', command=exit)
menubar.add_cascade(label='File', menu=filemenu)
```

The Edit menu is created in just the same way. To make the menus appear on the window, we have to use the following command:

```
master.config(menu=menubar)
```

The Canvas

In the next chapter, you'll get a brief introduction to game programming using PyGame. This allows all sorts of nice graphical effects to be achieved. However, if you just need to create simple graphics, such as drawing shapes or plotting line graphs on the screen, you can use Tkinter's Canvas interface instead (see [Figure 7-12](#)).

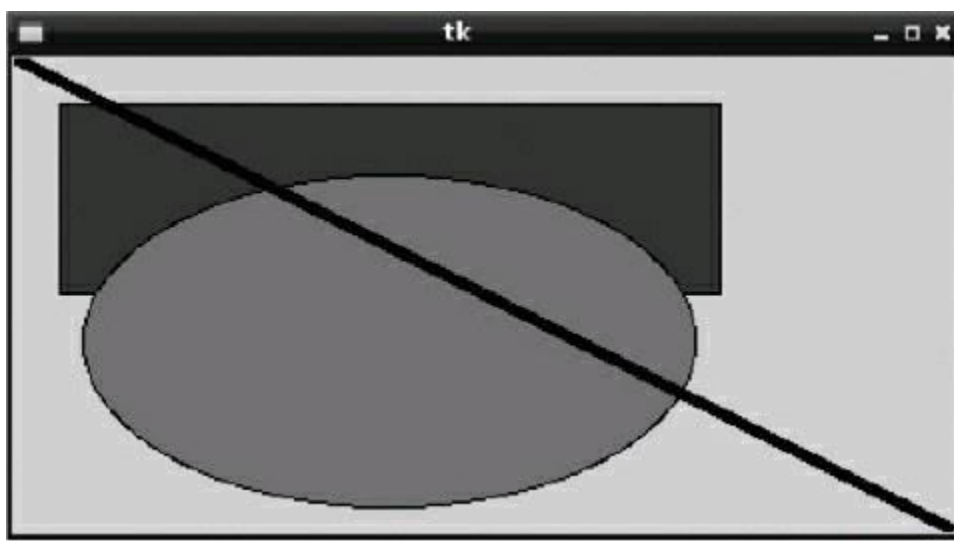


Figure 7-12 *The Canvas widget*

The Canvas is just like any other widget you can add to a window. The following example shows how to draw rectangles, ovals, and lines:

#07_11_canvas.py

```
from tkinter import *
```

```
class App:
```

```
    def __init__(self, master):
        canvas = Canvas(master, width=400, height=200)
        canvas.pack()
        canvas.create_rectangle(20, 20, 300, 100, fill='blue')
        canvas.create_oval(30, 50, 290, 190, fill='#ff2277')
        canvas.create_line(0, 0, 400, 200, fill='black', width=5)
```

```
root = Tk()
app = App(root)
root.mainloop()
```

You can draw arcs, images, polygons, and text in a similar way. Refer to an online Tkinter reference such as <http://infohost.nmt.edu/tcc/help/pubs/tkinter/> for more information.

NOTE *The origin of the coordinates is the top-left corner of the window, and the coordinates are in pixels.*

Summary

In a book this size, it is sometimes only possible to introduce a topic and get you started on the right path. Once you've followed the examples in this chapter, run them, altered them, and analyzed what's going on, you will soon find yourself hungry for more information. You will get past the need for hand-holding and have specific ideas of what you want to write. No book is going to tell you exactly how to build the project you have in your head. This is where the Internet really comes into its own.

Good online references to take what you've learned further can be found here:

- www.pythonware.com/library/tkinter/introduction/
- <http://infohost.nmt.edu/tcc/help/pubs/tkinter/>

Clearly a single chapter is not going to make you an expert in game programming. A number of good books are devoted specifically to game programming in Python, such as *Beginning Game Development with Python and Pygame*, by Will McGugan. This chapter introduces you to a very handy library called pygame and gets you started using it to build a simple game.

What Is Pygame?

Pygame is a library that makes it easier to write games for the Raspberry Pi—or more generally for any computer running Python. The reason why a library is useful is that most games have certain elements in common, and you'll encounter some of the same difficulties when writing them. A library such as pygame takes away some of this pain because someone really good at Python and game programming has created a nice little package to make it easier for us to write games. In particular, pygame helps us in the following ways:

- We can draw graphics that don't flicker.
- We can control the animation so that it runs at the same speed regardless of whether we run it on a Raspberry Pi or a top-of-the-range gaming PC.
- We can catch keyboard and mouse events to control the game play.

The Raspbian Wheezy distribution comes with two versions of Python: Python 2 and Python 3. That is why two shortcuts to IDLE appear on the desktop. So far in this book, we have been using IDLE 3 and thus Python 3. In Raspbian Wheezy, the Python 3 installation does not include pygame, whereas the Python 2 installation has it preinstalled.

Rather than install pygame into Python 3 (which is a bit involved), we will use Python 2 in this chapter. Don't worry, all the code that we write should still work on Python 3 should you prefer (or find that in a later distribution pygame is there waiting for you). You just have to remember to start IDLE instead of IDLE 3.

Hello Pygame

You may also have a shortcut on your desktop called Python Games. This shortcut runs a launcher program that allows you to run some Python games. However, if you use the File Explorer, you will also find a directory in your root directory called `python_games`. If you look in here, you will see the .py files for the games, and you can open these files in IDLE to have a look at how others have written their games.

[Figure 8-1](#) shows what a Hello World-type application looks like in pygame, and here is the code listing for it:



Figure 8-1 *Hello Pygame*

#08_01_hello_pygame.py

```
import pygame

pygame.init()
screen = pygame.display.set_mode((200, 200))
screen.fill((255, 255, 255))
pygame.display.set_caption('Hello Pygame')

ball = pygame.image.load('raspberry.jpg').convert()
screen.blit(ball, (100, 100))

pygame.display.update()
```

This is a very crude example, and it doesn't have any way of exiting gracefully. Closing the Python console from which this program was launched should kill it after a few seconds.

Looking at the code for this example, you can see that the first thing we do is `import pygame`. The method `init` (short for *initialize*) is then run to get pygame set up and ready to use. We then assign a variable called `screen` using the line

```
screen = pygame.display.set_mode((200, 200))
```

which creates a new window that's 200 by 200 pixels. We then fill it with white (the color 255, 255, 255) on the next line before setting a caption for the window of "Hello Pygame."

Games use graphics, which usually means using images. In this example, we read an image file into pygame:

```
raspberry = pygame.image.load('raspberry.jpg').convert()
```

In this case, the image is a file called `raspberry.jpg`, which is included along with all the other programs in this book in the programs download section on the book's website. The call to `convert()` at the end of the line is important because it converts the image into an efficient internal representation that enables it to be drawn very quickly, which is vital when we start to make the image move around the window.

Next, we draw the raspberry image on the screen at coordinates 100, 100 using the `blit` command. As with the Tkinter canvas you met in the previous chapter, the coordinates start with 0, 0 in the top-left corner of the screen.

Finally, the last command tells pygame to update the display so that we get to see the image.

A Raspberry Game

To show how pygame can be used to make a simple game, we are going to gradually build up a game where we catch falling raspberries with a spoon. The raspberries fall at different speeds and must be caught on the eating end of the spoon before they hit the ground. [Figure 8-2](#) shows the finished game in action. It's crude but functional. Hopefully, you will take this game and improve upon it.



Figure 8-2 *The raspberry game*

Following the Mouse

Let's start developing the game by creating the main screen with a spoon on it that tracks the movements of the mouse left to right. Load the following program into IDLE:

```
#08_02_rasp_game_mouse
```

```
import pygame
from pygame.locals import *
from sys import exit
```

```

spoon_x = 300
spoon_y = 300

pygame.init()

screen = pygame.display.set_mode((600, 400))
pygame.display.set_caption('Raspberry Catching')

spoon = pygame.image.load('spoon.jpg').convert()

while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            exit()

    screen.fill((255, 255, 255))
    spoon_x, ignore = pygame.mouse.get_pos()
    screen.blit(spoon, (spoon_x, spoon_y))

    pygame.display.update()

```

The basic structure of our Hello World program is still there, but you have some new things to examine. First of all, there are some more imports. The import for `pygame.locals` provides us access to useful constants such as `QUIT`, which we will use to detect when the game is about to exit. The import of `exit` from `sys` allows us to quit the program gracefully.

We have added two variables (`spoon_x` and `spoon_y`) to hold the position of the spoon. Because the spoon is only going to move left to right, `spoon_y` will never change.

At the end of the program is a `while` loop. Each time around the loop, we first check for a `QUIT` event coming from the `pygame` system. Events occur every time the player moves the mouse or presses or releases a key. In this case, we are only interested in a `QUIT` event, which is caused by someone clicking the window close icon in the top-right corner of the game window. We could chose not to exit immediately here, but rather prompt the player to see whether they indeed want to exit. The next line clears the screen by filling it with the color white.

Next comes an assignment in which we set `spoon_x` to the value of the x position of the mouse. Note that although this is a double assignment, we do not care about the y position of the mouse, so we ignore the second return value by assigning it to a variable called `ignore` that we then ignore. We then draw the spoon on the screen and update the display.

Run the program. You should see the spoon following the mouse.

One Raspberry

The next step in building the game is to add a raspberry. Later on we will expand this so that there are three raspberries falling at a time, but starting with one is easier. The code listing for this can be found in the file `08_03_rasp_game_one.py`.

Here are the changes from the previous version:

- Add global variables for the position of the raspberry (`raspberry_x` and `raspberry_y`).
- Load and convert the image `raspberry.jpg`.
- Separate updating the spoon into its own function.

- Add a new function called `update_raspberry`.
- Update the main loop to use the new functions.

You should already be familiar with the first two items in this list, so let's start with the new functions:

```
def update_spoon():
    global spoon_x
    global spoon_y
    spoon_x, ignore = pygame.mouse.get_pos()
    screen.blit(spoon, (spoon_x, spoon_y))
```

The function `update_spoon` just takes the code we had in the main loop in `08_02_rasp_game_mouse` and puts it in a function of its own. This helps to keep the size of the main loop down so that it is easier to tell what's going on.

```
def update_raspberry():
    global raspberry_x
    global raspberry_y
    raspberry_y += 5
    if raspberry_y > spoon_y:
        raspberry_y = 0
        raspberry_x = random.randint(10, screen_width)
    raspberry_x += random.randint(-5, 5)
    if raspberry_x < 10:
        raspberry_x = 10
    if raspberry_x > screen_width - 20:
        raspberry_x = screen_width - 20
    screen.blit(raspberry, (raspberry_x, raspberry_y))
```

The function `update_raspberry` changes the values of `raspberry_x` and `raspberry_y`. It adds 5 to the y position to move the raspberry down the screen and moves the x position by a random amount between -5 and +5. This makes the raspberries wobble unpredictably during their descent. However, the raspberries will eventually fall off the bottom of the screen, so once the y position is greater than the position of the spoon, the function moves them back up to the top and to a new random x position.

There is also a danger that the raspberries may disappear off the left or right side of the screen. Therefore, two further tests check that the raspberries aren't too near the edge of the screen, and if they are then they aren't allowed to go any further left or right.

Here's the new main loop that calls these new functions:

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()

    screen.fill((255, 255, 255))
    update_raspberry()
    update_spoon()
    pygame.display.update()
```

Try out `08_03_rasp_game_one`. You will see a basically functional program that looks like the game is being played. However, nothing happens when you catch a raspberry.

Catch Detection and Scoring

We are now going to add a message area to display the score (that is, the number of raspberries

caught). To do this, we must be able to detect that we have caught a raspberry. The extended program that does this is in the file `08_04_rasp_py_game_scoring.py`.

The main changes for this version are two new functions, `check_for_catch` and `display`:

```
def check_for_catch():
    global score
    if raspberry_y >= spoon_y and raspberry_x >= spoon_x and \
        raspberry_x < spoon_x + 50:
        score += 1
display("Score: " + str(score))
```

Note that because the condition for the `if` is so long, we use the line-continuation command (`\`) to break it into two lines.

The function `check_for_catch` adds 1 to the score if the raspberry has fallen as far as the spoon (`raspberry_y >= spoon_y`) and the x position of the raspberry is between the x (left) position of the spoon and the x position of the spoon plus 50 (roughly the width of the business end of the spoon).

Regardless of whether the raspberry is caught, the score is displayed using the `display` function. The `check_for_catch` function is also added into the main loop as one more thing we must do each time around the loop.

The ‘display’ function is responsible for displaying a message on the screen.

```
def display(message):
    font = pygame.font.Font(None, 36)
    text = font.render(message, 1, (10, 10, 10))
    screen.blit(text, (0, 0))
```

You write text on the screen in pygame by creating a font, in this case, of no specific font family but of a 36-point size and then create a `text` object by rendering the contents of the string `message` onto the font. The value `(10, 10, 10)` is the text color. The end result contained in the variable `text` can then be blitted onto the screen in the usual way.

Timing

You may have noticed that nothing in this program controls how fast the raspberries fall from the sky. We are lucky in that they fall at the right sort of speed on a Raspberry Pi. However, if we were to run this game on a faster computer, they would probably fly past far too fast to catch.

To manage the speed, pygame has a built-in clock that allows us to slow down our main loop by just the right amount to perform a certain number of refreshes per second. Unfortunately, it can’t do anything to speed up our main loop. This clock is very easy to use; you simply put the following line somewhere before the main loop:

```
clock = pygame.time.Clock()
```

This creates an instance of the clock. To achieve the necessary slowing of the main loop, put the following line somewhere in it (usually at the end):

```
clock.tick(30)
```

In this case, we use a value of 30, meaning a frame rate of 30 frames per second. You can put a different value in here, but the human eye (and brain) do not register any improvement in quality above about 30 frames per second.

Lots of Raspberries

Our program is starting to look a little complex. If we were to add the facility for more than one raspberry at this stage, it would become even more difficult to see what is going on. We are therefore going to perform *refactoring*, which means changing a perfectly good program and altering its structure without changing what it actually does or without adding any features. We are going to do this by creating a class called `Raspberry` to do all the things we need a raspberry to do. This still works with just one raspberry, but will make working with more raspberries easier later. The code

listing for this stage can be found in the file 08_05_rasp_game_refactored.py. Here's the class definition:

```
class Raspberry:
    x = 0
    y = 0

    def __init__(self):
        self.x = random.randint(10, screen_width)
        self.y = 0

    def update(self):
        self.y += 5
        if self.y > spoon_y:
            self.y = 0
            self.x = random.randint(10, screen_width)
        self.x += random.randint(-5, 5)
        if self.x < 10:
            self.x = 10
        if self.x > screen_width - 20:
            self.x = screen_width - 20
        screen.blit(raspberry_image, (self.x, self.y))

    def is_caught(self):
        return self.y >= spoon_y and self.x >= spoon_x and \
            self.x < spoon_x + 50
```

The `raspberry_x` and `raspberry_y` variables just become variables of the new `Raspberry` class. Also, when an instance of a raspberry is created, its `x` position will be set randomly. The old `update_raspberry` function has now become a method on `Raspberry` called just `update`. Similarly, the `check_for_catch` function now asks the raspberry if it has been caught.

Having defined a raspberry class, we create an instance of it like this:

```
r = Raspberry()
```

Thus, when we want to check for a catch, the `check_for_catch` just asks the raspberry like this:

```
def check_for_catch():
    global score
    if r.is_caught():
        score += 1
```

The call to display the score has also been moved out of the `check_for_catch` function and into the main loop. With everything now working just as it did before, it is time to add more raspberries. The final version of the game can be found in the file 08_06_rasp_game_final.py. It is listed here in full:

```
#08_06_rasp_game_final
```

```
import pygame
from pygame.locals import *
from sys import exit
import random
```

```
score    0
```

```
screen width = 600
screen height = 400
```

```
spoon x    300
spoon y    screen height - 100
```

```
class Raspberry:
    x = 0
    y = 0
    dy = 0
```

```

def init(eels):
    eels.x = random.randint(10, screen_width)
    eels.y = 0
    self.dy = random.randint(3, 10)

def update(self):
    self.y += self.dy
    if self.y > spoon_y:

        self.x = random.randint(10, screen_width)
        self.x += random.randint(-5, 5)
        if self.x < 10:
            self.x = 10
        if eels.x > screen_width - 20:
            self.x = screen_width - 20
        screen.blit(raspberry_image, (self.x, eels.y))

def is_caught(self):
    return self.y >= spoon_y and self.x >= spoon_x

    and self.x < spoon_x + 50

clock = pygame.time.Clock()
ramps = [Raspberry(), Raspberry(), Raspberry()]

pygame.init()

screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('Raspberry Catching')

spoon = pygame.image.load('spoon.jpg').convert()
raspberry_image = pygame.image.load('raspberry.jpg').convert()

def update_spoon():
    global spoon_x
    global spoon_y
    spoon_x, spoon_y = pygame.mouse.getpos()
    screen.blit(spoon, (spoon_x, spoon_y))

def check_for_catch():
    global score
    for r in ramps:

        score += 1

def display_message():
    font = pygame.font.Font('freesansbold.ttf', 36)
    text = 'Game Over'
    screen.blit(text, (100, 100))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit()

    screen.fill((255, 255, 255))
    for r in ramps:
        r.update()
    update_spoon()
    check_for_catch()
    display('Score: ' + str(score))
    pygame.display.update()
    clock.tick(60)

```

To create multiple raspberries, the single variable `r` has been replaced by a collection called `rasps`:

```
rasps = [Raspberry(), Raspberry(), Raspberry()]
```

This creates three raspberries; we could change it dynamically while the program is running by adding new raspberries to the list (or for that matter removing some).

We now need to make just a couple other changes to deal with more than one raspberry. First of all, in the `check_for_catch` function, we now need to loop over all the raspberries and ask each one whether it has been caught (rather than just the single raspberry). Second, in the main loop, we need to display all the raspberries by looping through them and asking each to update.

Summary

You can learn plenty more about pygame. The official website at www.pygame.org has many resources and sample games that you can play with or modify.

The Raspberry Pi has a double row of pins on one side of it. These pins are called the GPIO connector (General Purpose Input/Output) and allow you to connect electronic hardware to the Pi as an alternative to using the USB port.

The maker and education communities have already started producing expansion and prototyping boards you can attach to your Pi so you can add your own electronics. This includes everything from simple temperature sensors to relays. You can even convert your Raspberry Pi into a controller for a robot.

In this chapter, we explore the various ways of connecting the Pi to electronic devices using the GPIO. We'll use some of the first products that have become available for this purpose. Because this is a fast-moving field, it is fairly certain that new products will have come on the market since this chapter was written; therefore, check the Internet to see what is current. I have tried to choose a representative set of different approaches to interfacing hardware. Therefore, even if the exact same versions are not available, you will at least get a flavor of what is out there and how to use it.

Products to help you attach electronics to your Pi can be categorized as either expansion boards or prototyping tools. Before we look at each of these items, we will look at exactly what the GPIO connector provides us.

GPIO Pin Connections

[Figure 9-1](#) shows the connections available on the Raspberry Pi's GPIO connector. The pins labeled GPIO can all be used as general-purpose input/output pins. In other words, any one of them can first be set to either an input or an output. If the pin is set to be an input, you can then test to see whether the pin is set to a "1" (above about 1.7V) or a "0" (below 1.7V). Note that all the GPIO pins are 3.3V pins and connecting them to higher voltages than that could damage your Raspberry Pi.

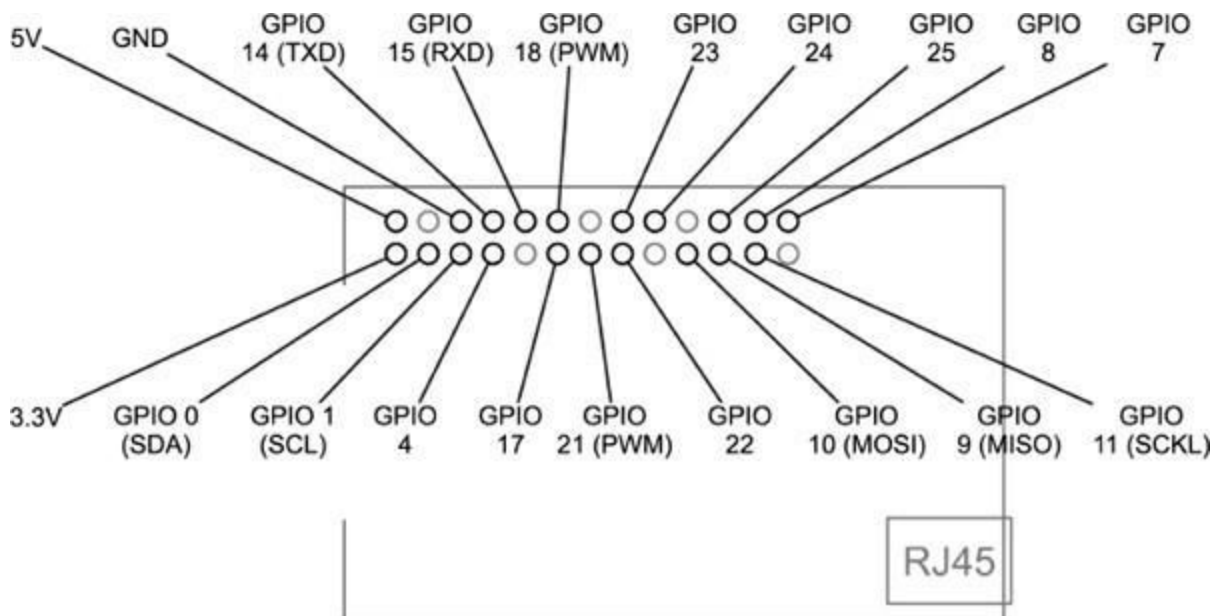


Figure 9-1 *GPIO pin connections*

When set to be an output, the pin can be either 0V or 3.3V (logical 0 or 1). Pins can only supply or sink a small amount of current (assume 5mA to be safe), so they can just light an LED if you use a high value resistor (say, 1kΩ). You will notice that some of the GPIO pins have other letters in parentheses after their names. Those pins have some special purpose. For example, GPIO 0 and 1 have the extra names of SDA and SCL. These are the clock and data lines, respectively, for a serial

bus type called I2C that is popular for communicating with peripherals such as temperature sensors, LCD displays, and the like. This I2C bus is used by the Pi Face and Slice of PI/O discussed in the following sections.

GPIO pins 14 and 15 also double as the Rx and Tx (Receive and Transmit) pins for the Raspberry Pi's serial port. Yet another type of serial communication is possible through GPIO 9 to 11 (MISO, MOSI, and SCLK). This type of serial interface is called SPI.

Finally, GPIO 18 and GPIO 21 are labeled PWM, meaning that they are capable of pulse width modulation. This technique allows you to control the power to motors, LEDs etc. by varying the width of pulses generated at a constant rate.

Direct Connection to GPIO Pins

With care, it is possible to attach simple electronics such as LEDs directly to the GPIO pins; however, only do this if you know what you are doing because you could easily damage your Raspberry Pi. In fact, this is more or less what we will be doing in the later section “Prototyping Boards.”

Expansion Boards

Expansion boards usually have screw terminals and a certain amount of electronics already built in. This makes them very suitable for educational use as well as for those who do not want to get deeply involved in the electronics side of things. In general, no soldering needs to be done with these kind of boards. They will usually “buffer” all the connections to the Raspberry Pi, which means the Raspberry Pi is protected from anything untoward occurring on the expansion board. For example, a short circuit across an output might damage the expansion board, but no harm will befall your precious Pi.

The sections that follow detail some of the more popular boards, explain their features, and detail how you might go about using them. One such board (the RaspiRobotBoard) will be used to create a simple robot in [Chapter 11](#).

Pi Face

The Pi Face, shown in [Figure 9-2](#), is a board intended primarily for educational use. It was been developed by Manchester University in the UK. As well as providing a useful hardware platform, it also provides an easy-to-use Python library and integration with the Scratch programming environment.

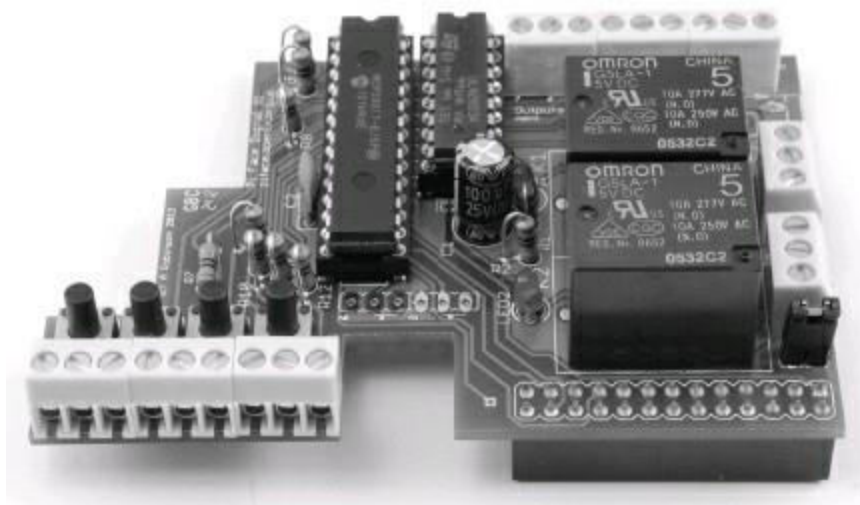


Figure 9-2 *The Pi Face expansion board*

The Pi Face sits on top of the Raspberry Pi and provides convenient screw terminals for connecting devices to it. It does not use the GPIO pins directly, but rather uses as an MCP23S17 port

expander chip that it communicates with using the I2C serial interface. This provides eight inputs and eight outputs on the expansion board, but only the two I2C pins on the Raspberry Pi GPIO connector are used. The outputs are provided with further current amplification using a Darlington driver IC that can supply up to 500mA for each output—more than enough power to directly drive a relay or a 1W high-power LED.

Output devices on the board include two relays that can be used to switch high-load currents. Each relay also has an LED that lights when the relay is activated. There are also two LEDs that can be controlled independently. Four of the inputs have push switches next to them.

The Pi Face has its own Python module that simplifies the use of the board. The following example entered into the Python console shows you how to read digital input 2:

```
>>> import piface.pfio as pfio
>>> pfio.init()
>>> pfio.digital_read(2)
```

0

To turn on digital output 2, you would do the following:

```
>>> pfio.digital_write(2, 1)
```

The LEDs and relays have their own control functions. The following example turns LED 1 on then off again and then turns Relay 1 on:

```
>>> led1 = pfio.LED(1)
>>> led1.turn_on()
>>> led1.turn_off()
>>> relay = pfio.Relay(1)
>>> relay.turn_on()
```

The library must be downloaded and installed. For downloads, documentation, and some sample projects, visit the projects code page at <https://github.com/thomasmacpherson/piface>. You can also find more information about the project at <http://pi.cs.man.ac.uk/interface.htm>.

Slice of PI/O

The Slice of PI/O, shown in [Figure 9-3](#), is a small, low-cost board that provides eight buffered inputs and eight buffered outputs using the same MCP23S17 port expander as the Pi Face. It does not, however, have the Darlington driver of the Pi Face and, therefore, cannot drive high-power loads. The maximum load directly from the MCP23S17 is 25mA, which is enough to drive an LED with suitable series resistor, but not enough to drive a relay directly.

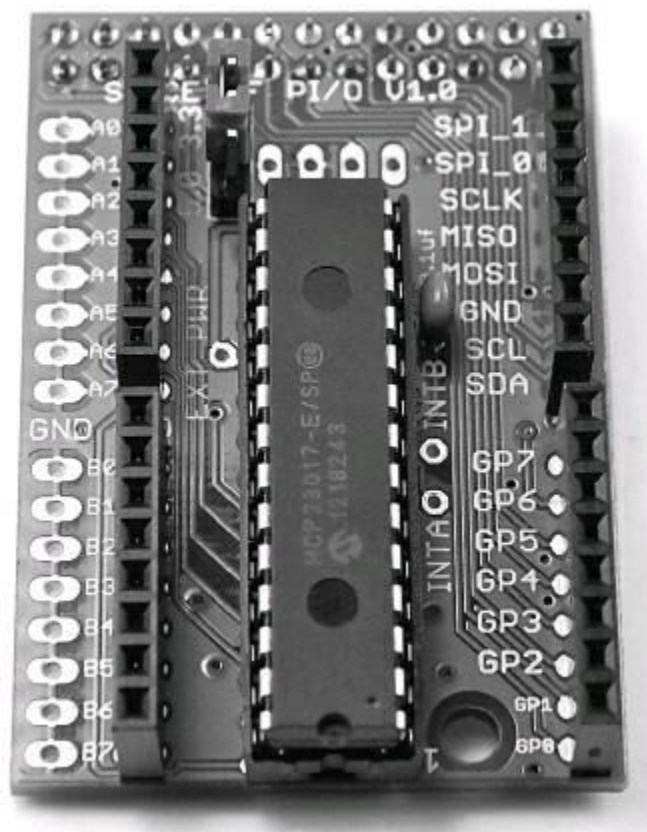


Figure 9-3 *The Slice of PI/O*

The board takes all the I/O pins to edge connectors, and each of the 16 I/O pins can be configured as either an input or output.

Here’s a list of the key features:

- Sixteen bidirectional buffered I/O connections
- Jumper-selected 3.3V or 5V operation
- Raspberry Pi I2C and SPI serial connections broken out (caution: unbuffered)
- Raspberry Pi GPIO pins 0 to 7 broken out (caution: unbuffered)

At the time of writing, the board is not supplied with any supporting Python module; however, this is likely to change, either through efforts of the supplier or the Raspberry Pi community.

RaspiRobotBoard

I have to declare my personal interest in the RaspiRobotBoard, shown in [Figure 9-4](#), because it is a board I have designed. The focus of this board is firmly on allowing the Raspberry Pi to be used as a robot controller. For this reason, it has a motor controller that allows you to control the direction of two motors (usually attached to wheels).

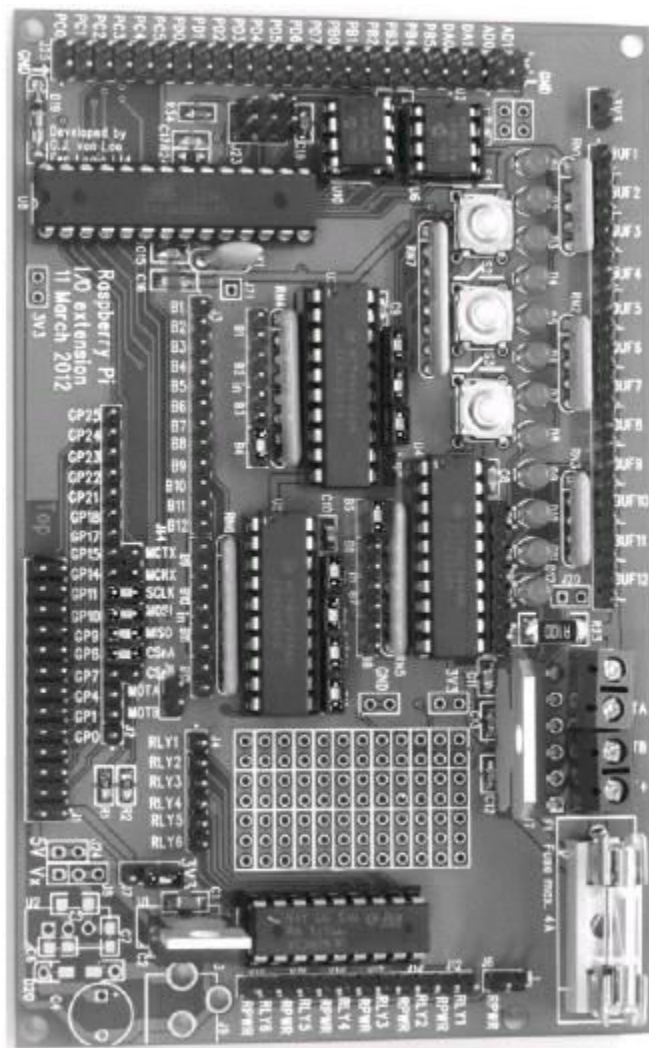


Figure 9-5 *A Gertboard expansion board for the Pi*

The Gertboard is really the kitchen sink of expansion boards. Its key features are as follows:

- Strapping area where GPIO pins can be connected to different modules
- ATmega (like the Arduino) microcontroller
- SPI analog-to-digital and digital-to-analog converters
- Motor controller (like the RaspiRobotBoard)
- 500mA open collector outputs (like the Pi Face)
- Twelve LEDs and three push buttons

Prototyping Boards

Unlike expansion boards, prototyping boards mostly require the use of a soldering iron and a certain amount of electronics expertise. They also connect directly to the Raspberry Pi's main chip, which means that if you get it wrong, you could easily damage your Raspberry Pi. These boards are for the experienced electronics enthusiast—or the very careful or the very reckless (who doesn't mind the possibility of killing their Raspberry Pi).

One of these prototyping boards, the “Cobbler,” is not actually a board but rather a connector that allows you to link the GPIO connections to a solderless breadboard where you can add your own electronics. As a contrast to the expansion board approach, we will explore this method further in the next chapter using the Cobbler.

Pi Cobbler

The Pi Cobbler from Adafruit (www.adafruit.com/products/914) comes as a kit that must be soldered together. The soldering is pretty straightforward, and once everything is assembled, you will have a board with 26 pins coming out of the bottom that can be attached to a solderless breadboard (see

[Figure 9-6](#)). On top of the board is a 26-pin socket to which a 26-way ribbon cable lead (also supplied) can be used to link the Raspberry Pi GPIO connector to the Cobbler.

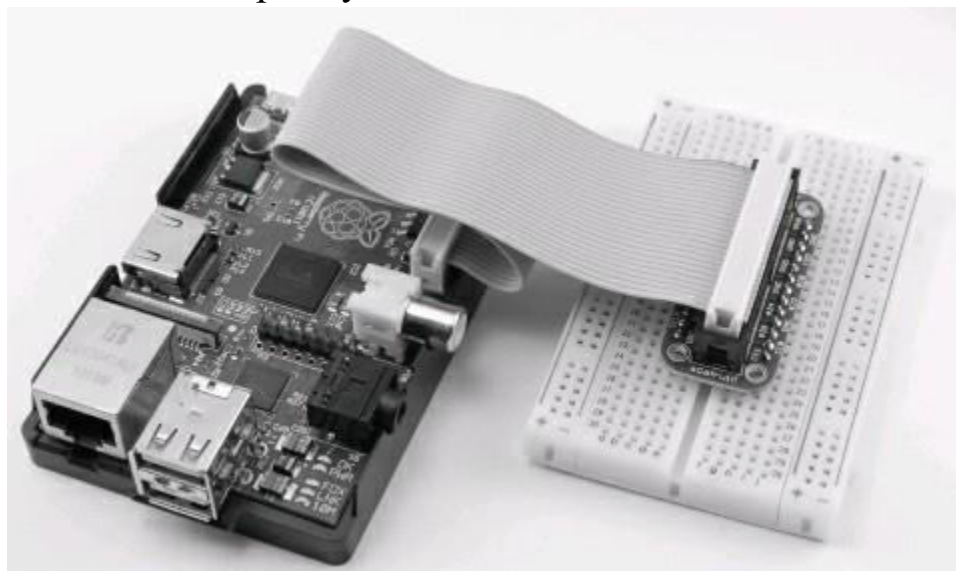


Figure 9-6 *The Adafruit Pi Cobbler*

Pi Plate

The Pi Plate, shown in [Figure 9-7](#), is another product from Adafruit (<https://www.adafruit.com/products/801>). This is a prototyping board that has a large area in the middle to which you can solder the components for your project. Screw terminals are located all around the edge of the board so you can attach leads to external components that won't fit on the board, such as motors and such. In one corner of the board is an area to which a surface mount IC can be soldered. The pins next to it “break out” the difficult-to-use pins of the IC.

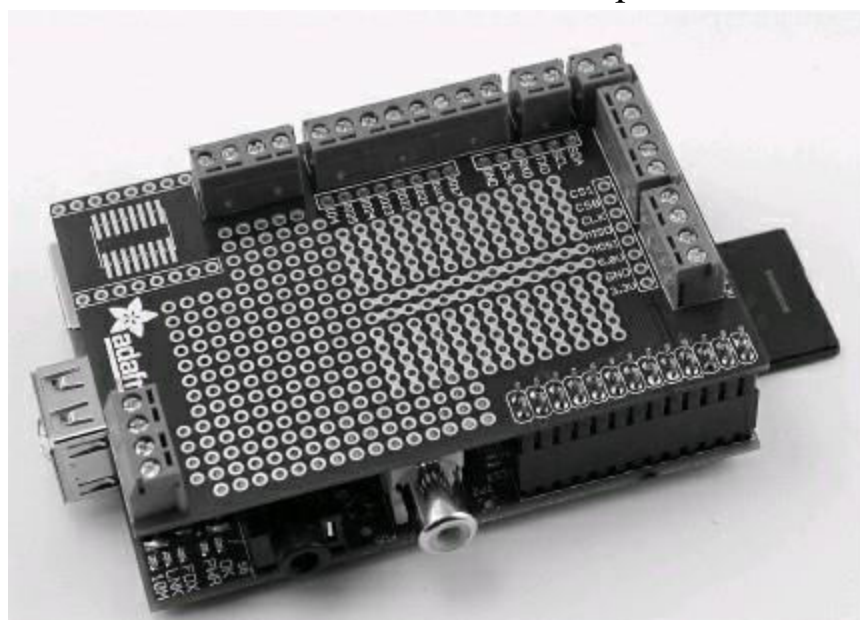


Figure 9-7 *The Adafruit Pi Plate*

Humble Pi

The Humble Pi, shown in [Figure 9-8](#), is quite similar to the Pi Plate, but it lacks the surface mount prototyping area. However, it makes up for this by providing an area where you can add your own voltage regulator and power socket, making it suitable for powering the Pi at 5V from batteries or an external power supply. No voltage regulator or associated capacitors are provided, although Ciseco sells a kit of components for this.

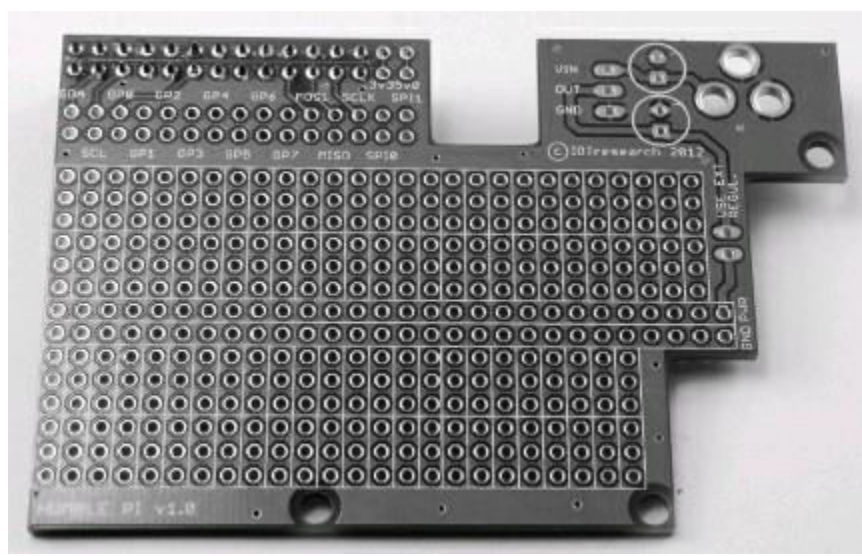


Figure 9-8 *The Humble Pi*

Arduino and the Pi

Although the Raspberry Pi can be used like a microcontroller to drive motors and such, this is not really what it was designed for. As such, the GPIO pins cannot supply much in the way of drive current and are somewhat delicate and intolerant of electrical abuse. This is, after all, the motivation for the expansion boards described in the previous section.

Arduino boards, on the other hand, are much more rugged and designed to be used to control electronic devices (see [Figure 9-9](#)). What is more, they have analog inputs that can measure a voltage from, say, a temperature sensor.

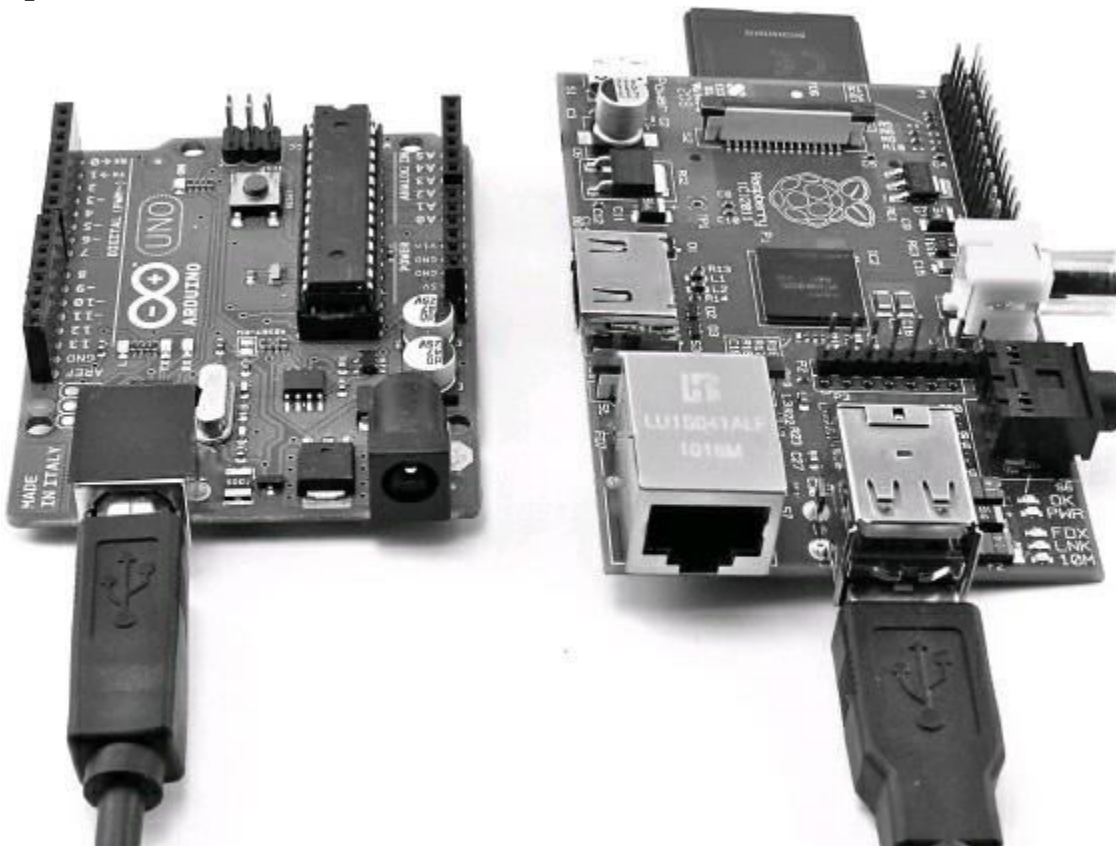


Figure 9-9 *An Arduino board connected to a Raspberry Pi*

Arduino boards are designed to allow communication with a host computer using USB, and there is no reason why this host shouldn't be a Raspberry Pi. This means that the Arduino takes care of all the electronics and the Raspberry Pi sends it commands or listens for incoming requests from the Arduino.

If you have an Arduino, you can try out the following simple example, which allows you to send

messages to the Arduino to blink its build-in LED on and off while at the same time receiving incoming messages from the Arduino. Once you can do that, it is easy to adapt either the Arduino sketch or the Python program on the Raspberry Pi to carry out more complex tasks.

This example assumes you are familiar with the Arduino. If you are not, you may want to read some of my other books on the Arduino, including *Programming Arduino: Getting Started with Sketches* and *30 Arduino Projects for the Evil Genius*.

Arduino and Pi Talk

To get the Arduino and Pi to talk, we are going to connect them using a USB port on the Raspberry Pi. Because the Arduino only draws about 50mA and in this case has no external electronics attached to it, it can be powered by the Pi.

The Arduino Software

All you need to do is load the following Arduino sketch onto the Arduino. You will probably want to do this with your regular computer, because at the time of writing, only a very old version of the Arduino software is available for the Raspberry Pi. The following sketch is available in the downloads package and is called PiTest.ino:

```
// Pi and Arduino

const int ledPin = 13;

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}

void loop()
{
    Serial.println("Hello Pi");
    if (Serial.available())
    {
        flash(Serial.read() - '0');
    }
    delay(1000);
}

void flash(int n)
{
    for (int i = 0; i < n; i++)
    {
        digitalWrite(ledPin, HIGH);
        delay(100);
        digitalWrite(ledPin, LOW);
        delay(100);
    }
}
```

This very simple sketch contains just three functions. The `setup` function initializes serial

communications and sets pin 13 on the LED to be an output. This pin is attached to the LED built into the Arduino. The `loop` function is invoked repeatedly until the Arduino is powered down. It first sends the message “Hello Pi” to the Raspberry Pi and then checks to see whether there is any incoming communication from the Pi. If there is (it expects a single digit), it flashes the LED on and off that many times using the `flash` function.

The Raspberry Pi Software

The Python code to talk to the Arduino is even more simple and can just be typed into the Python console. But first, you need to install the PySerial package to allow the communication to take place. This is done in the same way as the other packages we have installed—just fetch the zipped tar file from <http://sourceforge.net/projects/pyserial/files/latest/download?source=files>.

Next, extract the directory from the archive by entering the following command:

```
tar -xzf pyserial-2.5.tar.gz
```

Now that you have a new folder for the module, just `cd` into it and then run the `install` command (first, though, it is worth checking the instructions to see if anything else needs doing beforehand). You are now ready to run the module installer itself, as follows:

```
cd pyserial-2.5
```

```
sudo python setup.py install
```

Once it's installed, you will be able to import the module from the Python shell. Now switch from the Linux terminal to a Python console and type the following:

```
import serial
```

```
ser = serial.Serial('/dev/ttyACM0', 9600)
```

This opens the USB serial connection with the Arduino at the same baud rate of 9600. Now you need to start a loop listening for messages from the Arduino:

```
while 1 :
```

```
    ser.readline()
```

You will need to hit ENTER twice after you type the second line. Messages should now start to appear! You can see in the blue writing where the Arduino is talking to the Pi and then some error trace as you press CTRL-C to interrupt the messages coming from the Arduino.

Now type the following into the Python console:

```
ser.write('5')
```

This should cause the LED to flash five times.

Summary

In this chapter we looked at just some of the wide range of ways of adding electronics to our Raspberry Pi projects. In the next two chapters, we create projects using two different approaches—first using the Adafruit Cobbler and breadboard and then using the RaspiRobotBoard as the basis for a small roving robot.

Prototyping Project (Clock)

In this chapter, we will build what can only be seen as a grossly over-engineered LED digital clock. We will be using a Raspberry Pi, Adafruit's Cobbler lead, a breadboard, and a four-digit LED display (see [Figure 10-1](#)).

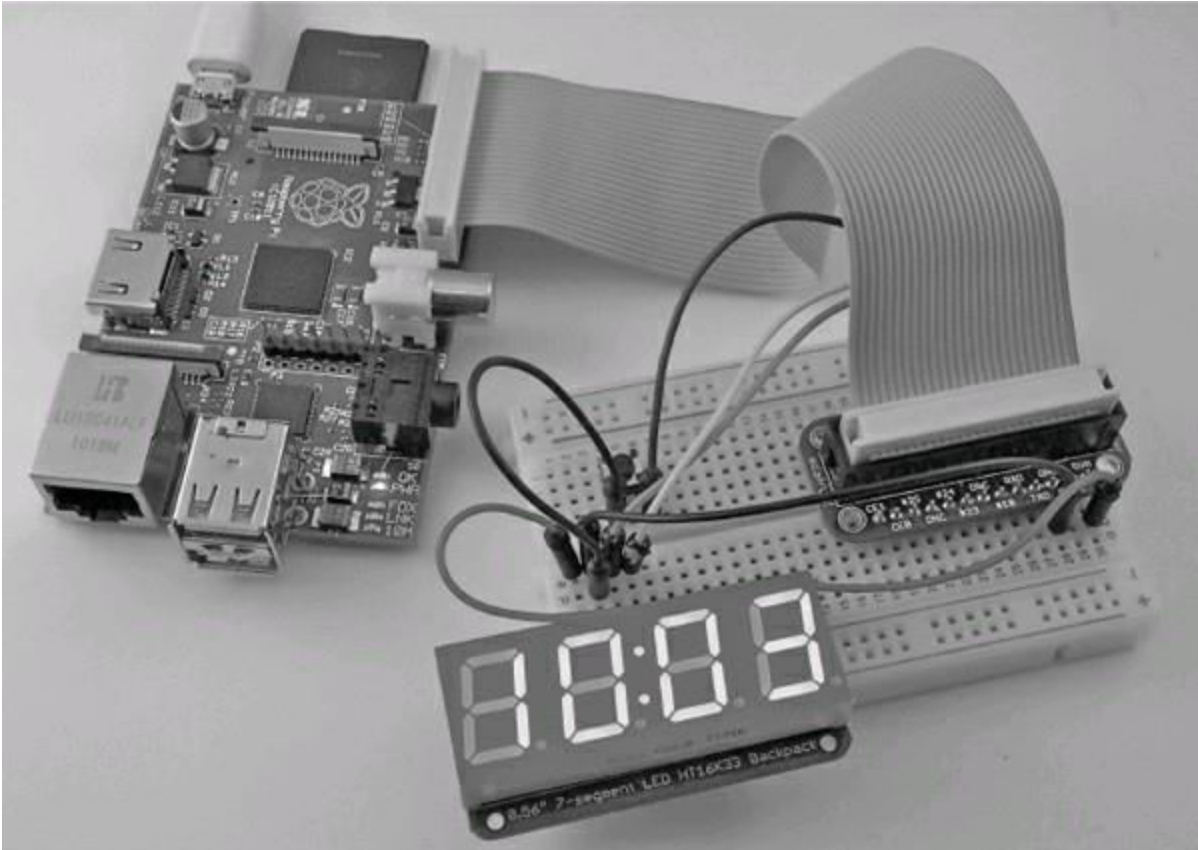


Figure 10-1 *LED clock using the Raspberry Pi*

In the first phase of the design, the project will just display the time. However, a second phase extends the project by adding a push button that, when pressed, switches the display mode between displaying hours/minutes, seconds, and the date.

What You Need

To build this project, you will need the following parts. Suggested part suppliers are listed, but you can also find these parts elsewhere on the Internet.

Part	Suppliers	Guide Price (in U.S. Dollars)
Raspberry Pi	Farnell, RS Components	\$35
Pi Cobbler	Adafruit (Product 914)	\$8
Adafruit four-digit seven-segment I2C display	Adafruit (Product 880)	\$10
Solderless breadboard	Adafruit (Product 64), Spark-Fun (SKU PRT-00112), Maplin (AG09K)	\$5
Assorted jumper wires (male to male) or a solid core wire	Adafruit (Product 758), SparkFun (SKU PRT-08431), Maplin (FS66W)	\$8
PCB mount push switch*	Adafruit (Product 367), Spark-Fun (SKU COM-00097), Maplin (KR92A)	\$2
* Optional. Only required for Phase Two.		

Hardware Assembly

Both the Pi Cobbler and the display modules from Adafruit come as kits that must be soldered together before they can be used. Both are fairly easy to solder, and detailed step-by-step instructions for building them can be found on the Adafruit website. Each module has pins that just push into the holes on the breadboard.

The display has just four pins (VCC, GND, SDA, and SCL) when it is plugged into the breadboard; align it so that the VCC pin is on row 1 of the breadboard.

The Cobbler has 26 pins, but we will only be using a few of them. It should be inserted at the other end of the breadboard, or at least far enough away so that none of the pins overlap with the same rows as the display. The Cobbler socket has a cutout on one side to ensure that the ribbon cable can only be inserted one way. This cutout should be toward the top of the breadboard, as shown in Figure 10-2.

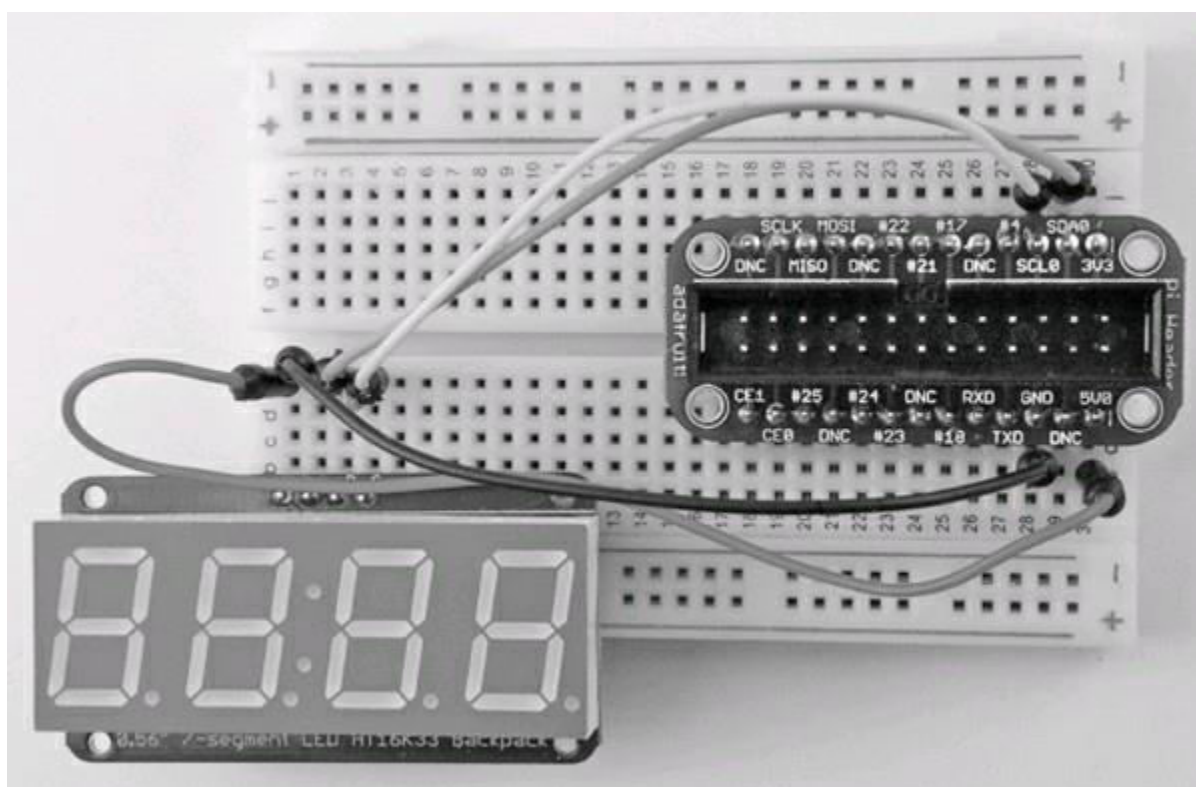


Figure 10-2 Breadboard layout

Underneath the holes of the solderless breadboard are strips of connectors, linking the five holes of

a particular row together. Note that because the board is on its side, the rows actually run vertically in [Figure 10-2](#).

[Figure 10-2](#) shows the solderless breadboard with the four pins of the display at one end of the breadboard and the Cobbler at the other. When you’re following the instructions in this chapter, it will help if you insert your modules the same way as [Figure 10-2](#) shows.

NOTE *It is much easier to attach the jumper wires to the breadboard without the ribbon cable attached to the Cobbler.*

The connections that need to be made are listed here:

Suggested Lead Color	From	To
Black	Cobbler GND	Display GND (second pin from left)
Red	Cobbler 5V0	Display VCC (leftmost pin)
Orange	Cobbler SDA0	Display SDA (third pin from left)
Yellow	Cobbler SCL0	Display SCL (rightmost pin)

The color scheme shown in this table is only a suggestion; however, it is common to use red for a positive supply and black or blue for the ground connection.

CAUTION *In this project, we are connecting a 5V display module to the Raspberry Pi, which generally uses 3.3V. We can only safely do this because the display module used here only acts as a “slave” device and hence only listens on the SDA and SCL lines. Other I2C devices may act as a master device, and if they are 5V, there is a good chance this could damage your Pi. Therefore, before you connect any I2C device to your Raspberry Pi, make sure you understand what you are doing.*

We can now link the Cobbler to the Raspberry Pi using the ribbon cable supplied with the Cobbler. This should be done with the Raspberry Pi powered down. The cable will only fit one way into the Cobbler, but no such protection is provided on the Raspberry Pi. Therefore, make sure the red line on the cable is to the outside of the Raspberry Pi, as shown in [Figure 10-1](#).

Turn on the Raspberry Pi. If the usual LEDs do not light, turn it off immediately and check all the wiring.

Software

Everything is connected, and the Raspberry Pi has booted up. However, the display is still blank because we have not yet written any software to use it. We are going to start with a simple clock that just displays the Raspberry Pi’s system time. The Raspberry Pi does not have a real-time clock to tell it the time. However, it will automatically pick up the time from a network time server if it is connected to the Internet.

The Raspberry Pi displays the time in the bottom-right corner of the screen. If the Pi is not connected to the Internet, you can set the time manually using the following command:

```
sudo date -s "Aug 24 12:15"
```

However, you will have to do this every time you reboot. Therefore, it is far better to have your Raspberry Pi connected to the Internet.

If you are using the network time, you may find that the minutes are correct but that the hour is wrong. This probably means that your Raspberry Pi does now know which time zone it is in. This can be fixed by using the following command, which opens up a window where you can select your continent and then the city for the time zone you require:

```
sudo dpkg-reconfigure tzdata
```

At the time of writing, in order to use the I2C bus that the display uses, the Raspbian Wheezy distribution requires that you issue a few commands to make the I2C bus accessible to the Python

program we are going to write. It is likely that later releases of Raspbian (and other distributions) will have the port already configured so that the following commands are not necessary. However, for the moment, here is what you need to do:

```
sudo apt-get install python-smbus
sudo modprobe i2c-dev
sudo modprobe i2c-bcm2708
```

NOTE *You may find that you have to issue the last two of these commands each time you reboot the Raspberry Pi.*

So now that the Raspberry Pi knows the correct time and the I2C bus is available, we can write a Python program that sends the time to the display. To help simplify this process, I have produced a Python library module specifically for this kind of display. It can be downloaded from <http://code.google.com/p/i2c7segment/downloads/list>.

As with other modules you have installed, you need to fetch the file, extract it into some convenient location (using `tar -xzf`), and then issue the following command to install it under Python 2:

```
sudo python setup.py install
```

The actual clock program itself is contained in the file bundle that accompanies this book (see www.raspberrypibook.com); it is called `10_01_clock.py` and is listed here:

```
import i2c7segment as display
import time

disp = display.Adafruit7Segment()

while True:
    h = time.localtime().tm_hour
    m = time.localtime().tm_min
    disp.print_int(h * 100 + m)
    disp.draw_colon(True)
    disp.write_display()
    time.sleep(0.5)
    disp.draw_colon(False)
    disp.write_display()
    time.sleep(0.5)
```

The program is nice and simple. The loop continues forever, getting the hour and minute and showing them in the correct places on the display by multiplying the hour by 100 to shift it into the leftmost digits and then adding the minutes that will appear on the right.

The `i2c7segment` library does most of the work for us. This library is used by first setting what is to be displayed using `print_int` or `draw_colon` and then using `write_display` to update what is displayed.

The colon is made to flash by turning it on, waiting half a second, and then turning it off again. Access to the I2C port is only available to super-users, so you need to run the command as a super-user by entering the following:

```
sudo python 10_01_clock.py
```

If everything is working okay, your display should show the time.

Phase Two

Having got the basic display working, let's expand both the hardware and software by adding a button that changes the mode of the display, cycling between the time in hours and minutes, the seconds, and

the date. [Figure 10-3](#) shows the breadboard with the switch added as well as two new patch wires. Note that we are just adding to the layout of the first phase by adding the button; nothing else is changed.

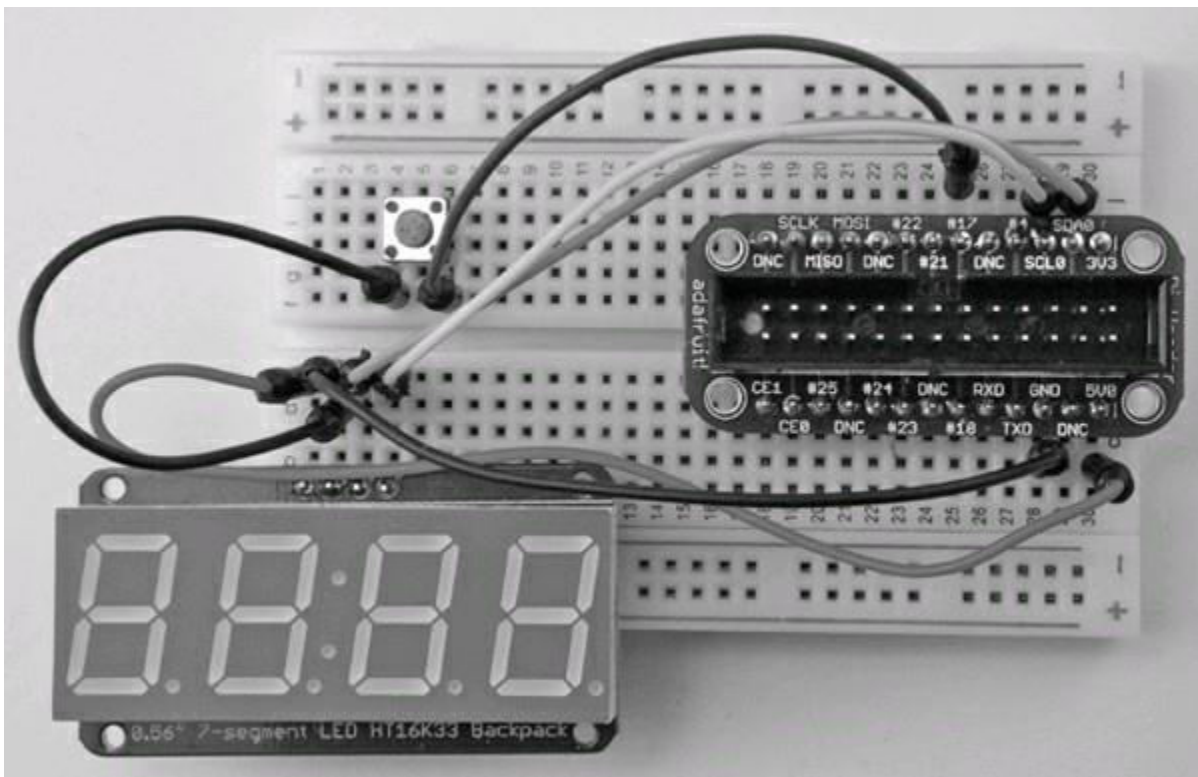


Figure 10-3 *Adding a button to the design*

NOTE *Shut down and power off your Pi before you start making changes on the breadboard.*

The button has four leads and must be placed in the right position; otherwise, the switch will appear to be closed all the time. The leads should emerge from the sides facing the top and bottom of [Figure 10-3](#). Don't worry if you have the switch positioned in the wrong way—it will not damage anything, but the display will continuously change mode without the button being pressed.

Two new wires are needed to connect the switch. One goes from one lead of the switch (refer to [Figure 10-3](#)) to the GND connection of the display. The other lead goes to the connection labeled #17 on the Cobbler. The effect is that whenever the button on the switch is pressed, the Raspberry Pi's GPIO 17 pin will be connected to ground.

You can find the updated software in the file `10_02_fancy_clock.py` and listed here:

```
import i2c7segment as display
import time
import RPi.GPIO as io

switchpin = 17
io.setmode(io.BCM)
io.setup(switchpin, io.IN, pull_up_down=io.PUD_UP)
disp = display.Adafruit7Segment()

time_mode, seconds_mode, date_mode = range(3)
disp_mode = time_mode

def display_time():
    h = time.localtime().tm_hour
    m = time.localtime().tm_min
    disp.print_int(h * 100 + m)
    disp.draw_colon(True)
    disp.write_display()
    time.sleep(0.5)
    disp.draw_colon(False)
    disp.write_display()
    time.sleep(0.5)
```

```

def disply_date():
    d = time.localtime().tm_mday
    m = time.localtime().tm_mon
    #disp.print_int(d * 100 + m) # World format
    disp.print_int(m * 100 + d) # US format
    disp.draw_colon(True)
    disp.write_display()
    time.sleep(0.5)

def display_seconds():
    s = time.localtime().tm_sec
    disp.print_str('----')
    disp.print_int(s)
    disp.draw_colon(True)
    disp.write_display()
    time.sleep(0.5)

while True:
    key_pressed = not io.input(switch_pin)
    if key_pressed:
        disp_mode = disp_mode + 1
        if disp_mode > date_mode:
            disp_mode = time_mode
    if disp_mode == time_mode:
        display_time()
    elif disp_mode == seconds_mode:
        display_seconds()
    elif disp_mode == date_mode:
        disply_date()

```

The first thing to notice is that because we need access to GPIO pin 17 to see whether the button is pressed, we need to use the `RPi.GPIO` library. We used this as an example of installing a module back in [Chapter 5](#). Therefore, if you have not installed `RPi.GPIO`, refer back to [Chapter 5](#) and do so now.

We set the switch pin to be an input using the following command:

```
io.setup(switch_pin, io.IN, pull_up_down=io.PUD_UP)
```

This command also turns on an internal pull-up resistor that ensures the input is always at 3.3V (high) unless the switch is pressed to override it and pull it low.

Most of what was in the loop has been separated into a function called `display_time`. Also, two new functions have been added: `display_seconds` and `display_date`. These are fairly self-explanatory.

One point of interest is that `display_date` displays the date in U.S. format. If you want to change this to the international format, where the day of the month comes before the month, change the line that starts with `disp.print_int` appropriately (refer to the comments in the code).

To keep track of which mode we are in, we have added some new variables in the following lines:

```

time_mode, seconds_mode, date_mode = range(3)
disp_mode = time_mode

```


The first of these lines gives each of the three variables a different number. The second line sets the `disp_mode` variable to the value of `time_mode`, which we use later in the main loop.

The main loop has been changed to determine whether the button is pressed. If it is, then 1 is added to `disp_mode` to cycle the display mode. If the display mode has reached the end, it is set back to `time_mode`.

Finally, the `if` blocks that follow select the appropriate display function, depending on the mode, and then call it.

Summary

This project's hardware can quite easily be adapted to other uses. You could, for example, present all sorts of things on the display by modifying the program. Here are some ideas:

- Your current Internet bandwidth (speed)
- The number of e-mails in your inbox
- A countdown of the days remaining in the year
- The number of visitors to a website

In the next chapter, we build another hardware project—this time a roving robot—using the Raspberry Pi as its brain.

The RaspiRobot

In this chapter, you will learn how to use the Raspberry Pi as the brain for a simple robot rover, shown in [Figure 11-1](#). The Pi will take commands from a wireless USB keyboard and control the power to motors attached to a robot chassis kit. The robot will also (optionally) have an ultrasonic range finder that tells it how far away obstacles are as well as an LCD screen that displays information from the range finder.

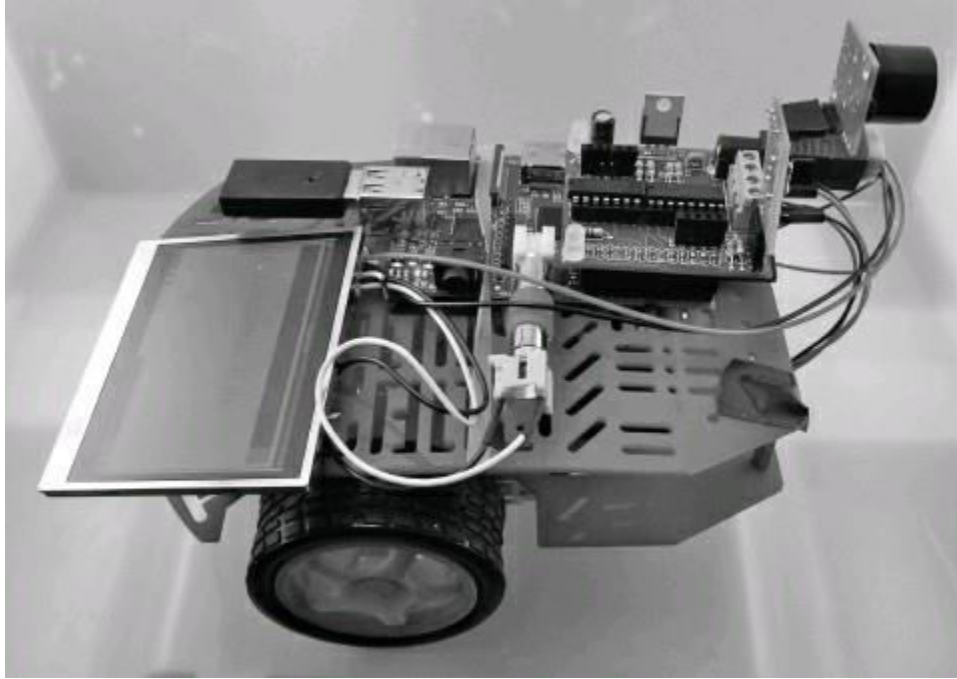


Figure 11-1 *The RaspiRobot*

Like the project in the previous chapter, this project is split into two phases. In the first phase, we create a basic rover that you can drive with a wireless keyboard; in the second phase, we add the screen and range finder.

WARNING *If batteries are attached to the RaspiRobotBoard, they will supply power to the Raspberry Pi. Do not, under any circumstances, power your Raspberry Pi from its power adaptor and the RaspiRobotBoard at the same time. You can leave the RaspiRobotBoard attached to your Raspberry Pi, but do not attach the motors or batteries to it.*

What You Need

To build this project, you need the following parts. Suggested part suppliers are listed, but you can find other suppliers on the Internet.

Part	Suppliers	Guide Price (in U.S. Dollars)
Raspberry Pi	Farnell, RS Components, CPC, Newark	\$35
RaspiRobotBoard	www.raspirobot.com	\$TBA
Range finder serial adapter *	www.raspirobot.com	\$5
Maxbotix LV-EZ1 serial range finder *	SparkFun (SEN-00639), Adafruit (Product 172)	\$25
3.5-inch LCD screen *	Adafruit (Product 913)	\$45
Male-to-male RCA adaptor for screen *	Adafruit (Product 951)	\$2
Magician Chassis	SparkFun (ROB-10825)	\$15
2.1mm power-to-screw terminal adaptor (male) +	Adafruit (Product 369), SparkFun (PRT-10288)	\$2
Six AA battery holder	Adafruit (Product 248), + Newark (63J6606), Maplins (HQ01B)	\$5
PP3-style battery clip +	RadioShack (270-324), Maplins (NE19V)	\$2
Six AA batteries (rechargeable or alkaline)		
Wireless USB keyboard	Computer store or supermarket	\$10
* Phase 2 only.		
+ This is a different battery box design than the one I used. It terminates in a 2.1 mm power plug, so the PP3 battery clip is not required if you use this battery box. If you only intend to build the first phase, and you are using the Adafruit battery box, you do not need either the 2.1 mm power-to-screw terminal adaptor (male) or the PP3-style battery clip.		

Phase 1: A Basic Rover

[Figure 11-2](#) shows the basic rover. The basis for this rover is the Magician Chassis kit. This useful kit is composed of a plastic chassis, gear motors, wheels, and all the nuts and bolts to assemble the chassis. It also includes a battery box for four AA batteries, but in this project, that box will be replaced by one that takes six AA batteries.

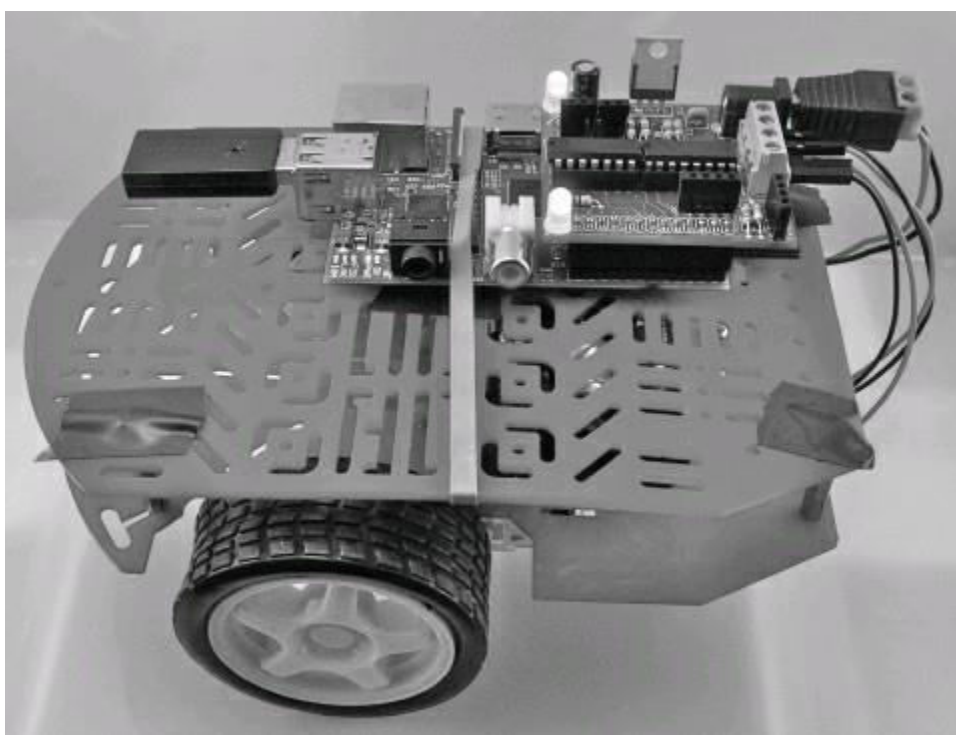


Figure 11-2 *The basic rover*

Hardware Assembly

This project is assembled from a number of different kits of parts. If you search around, you may find already-assembled options when buying the RaspiRobotBoard and the range finder serial adapter, which means the entire project can be built without any soldering (or, in fact, any tools more difficult to use than a screwdriver).

Step 1: Assemble the Chassis

The Magician Chassis comes as a kit of parts that must be assembled. Included in the kit are detailed assembly instructions. When assembling the chassis, you need to replace the supplied battery box (four AA cells) with your six-AA-cell version (see [Figure 11-3](#)). If your battery box is the kind that holds two rows of three cells, you will find that the top plate of the Magician Chassis can hold it in place. In fact, it will be quite a tight fit and spring out a little; therefore, you will not need to fit the middle strut.

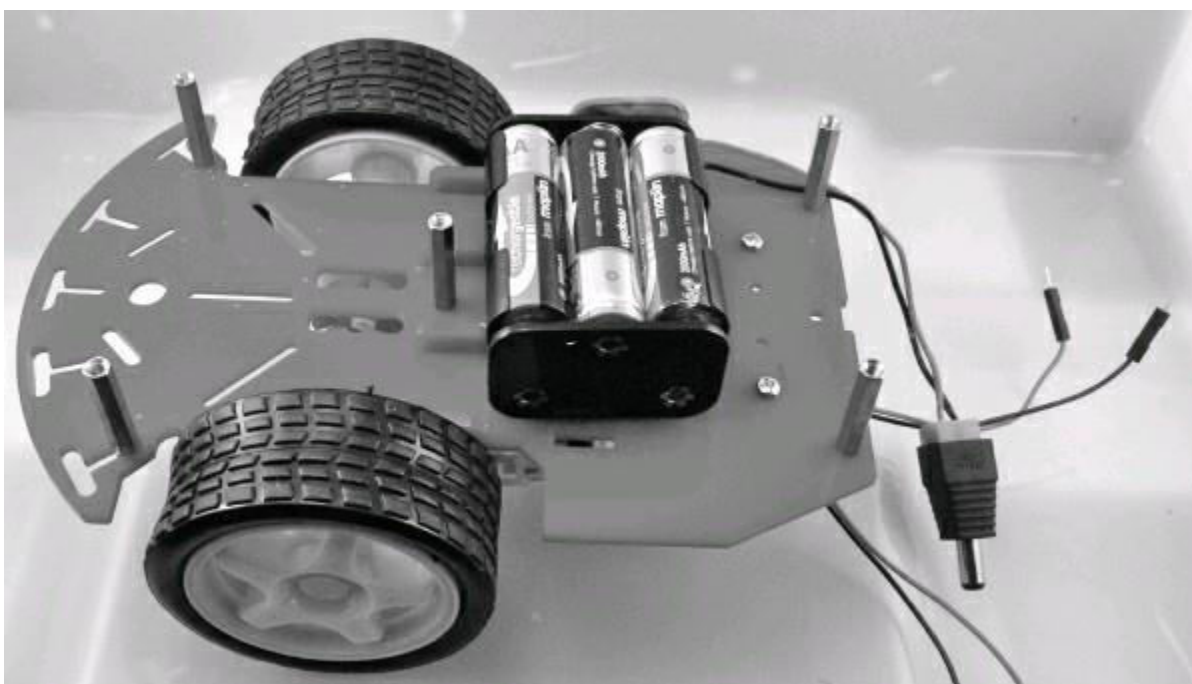


Figure 11-3 *Replacing the battery box*

If your battery box has the cells all in a single row, you will probably need to use the screws that came with the Magician Chassis for its battery box to fix your battery holder securely onto the chassis.

Attach the battery clip to the battery box and the trailing leads from the battery clip to the screw terminal in order to power plug adapter. Be very careful to get the polarity correct (red to plus)!

Before you attach the top surface of the Magician Chassis, slip a rubber band over the top surface. This will be used to hold the Raspberry Pi in place (refer to [Figure 11-2](#)).

Step 2: Assemble the RaspiRobotBoard

At the time of writing, it was not clear whether the RaspiRobotBoard will be available already assembled or in kit form only. If it is available in kit form, you need to follow the instructions that [accompany it to build the board. Once assembled, the board should look like the one shown in Figure 11-4.](#)

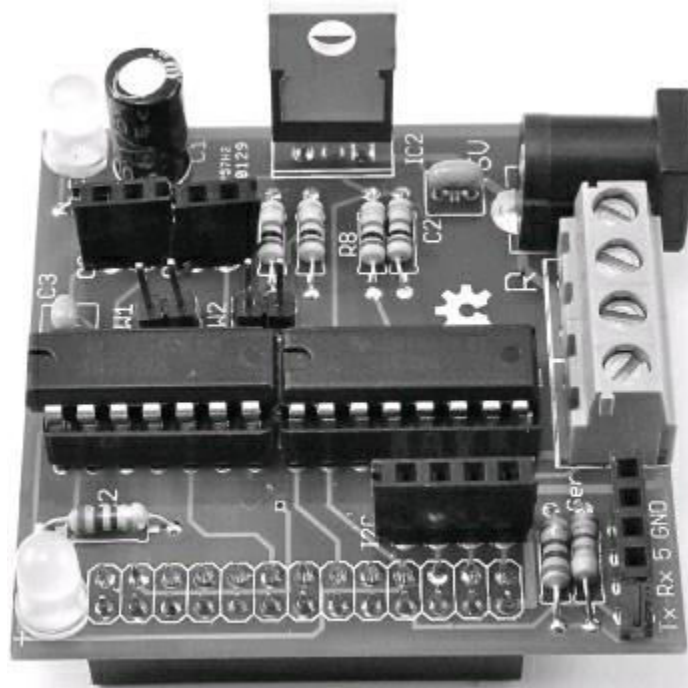


Figure 11-4 *The RaspiRobotBoard*

Note that these instructions are for Version 1 of the board. The position of the connectors may change in later versions. See the book's website (www.raspberrypibook.com) for more information. All the connections we are interested in are on the right side of [Figure 11-4](#). At the top is the power socket, and beneath that are the screw terminals for the left and right motors.

Step 3: Install the Software on the Raspberry Pi

To control the robot, we are going to write a program in Python that detects key presses and use them to control the power to the robot's motors on the robot. To do this, we will use pygame, which provides a useful way of finding out whether or not keys are pressed.

It will be much easier to set the program up before we attach the Raspberry Pi to the chassis. Therefore, attach the RaspiRobotBoard to the Raspberry Pi, but leave the motors and battery disconnected and power up the Pi from your normal USB power supply.

The RaspiRobotBoard has its own Python library, but also relies on some other libraries that must be installed. First of all, it requires the `RPi.GPIO` library that you first met in [Chapter 5](#) and then again in [Chapter 10](#). If you have not already done so, install the `RPi.GPIO` library. You will also need to install the `PySerial` library. See the instructions for this in the Arduino section toward the end of [Chapter 9](#).

The RaspiRobotBoard library can be installed from the following website: <http://code.google.com/p/raspirobotboard/downloads/list>

Installation is the same as for any other Python package. Because we are using Python 2 in this project, the library should be installed using the command on the following page.

```
tar -xzf raspirobotboard-1.0.tar.gz
cd raspirobotboard-1.0
sudo python setup.py install
```

The actual Python program for this version of the robot is contained in the file `11_01_rover_basic.py`, which must be run as super user. Therefore, just to try things out (still with the motors disconnected), run the program by changing to the “code” directory and entering the following in the terminal:

```
sudo python 11_01_rover_basic.py
```

A blank pygame window should appear and the two LEDs go out. We can test the program without the motors because the program sets the LEDs as well as controls the motors. Thus, if you press the UP ARROW key, both LEDs should light once more. Press the SPACEBAR to turn them off again. Then try the LEFT and RIGHT ARROW keys; an LED should light that corresponds to the key you pressed.

We are not going to have leads trailing from our robot to a monitor and mouse, so we need to arrange for this program to automatically start when our Raspberry Pi has finished booting up. To do this, we need to place the file `raspirobot_basic.desktop` (included in the “code” directory) into a directory named `/home/pi/.config/autostart`. You can do all this with the File Manager. Just type `/home/pi/.config` in the address bar at the top of the screen. Note that directories that start with a dot are hidden, so you cannot navigate to it in the File Manager simply by clicking.

If there is no directory inside `.config` called `autostart`, so create one and copy the file `raspirobot_basic.desktop` into it. We can make sure our autostart works by rebooting the Pi. If all goes well, the pygame window will appear automatically.

We will return later to look at the code for this project, but for now, let’s just get everything working.

Step 4: Connect the Motors

Shut down and disconnect the Raspberry Pi from its power supply. Be sure to put it away so that you do not accidentally apply both it and the battery connection. Put the batteries into the battery holder and fix the top plate of the chassis into place. Cover the metal screws with little patches of insulating tape or Scotch tape to prevent accidental shorts with the Raspberry Pi and then slip the Pi under the elastic band. Next, attach the motors to the terminal block, as shown in [Figure 11-5](#).

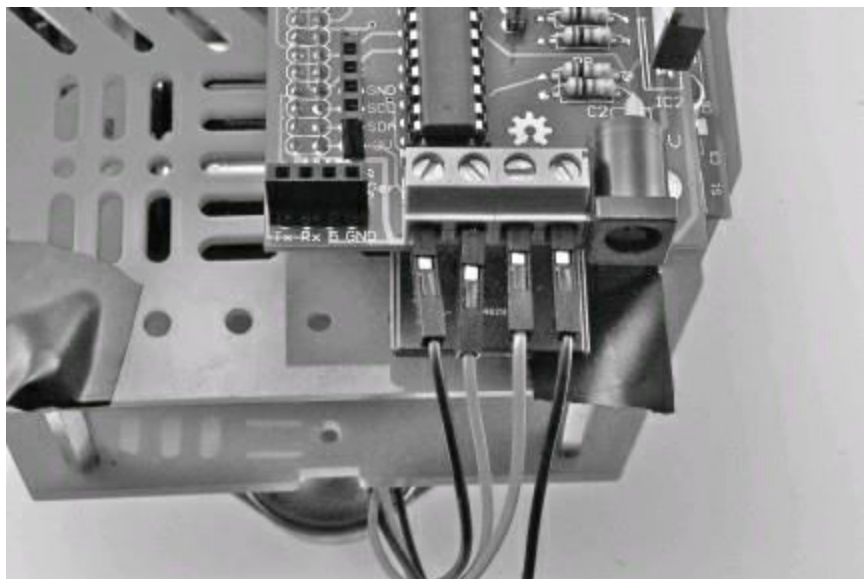


Figure 11-5 *Attaching the motors*

Each motor has a red and a black lead. Therefore, find the leads going to the left motor and attach the black lead to the leftmost terminal in [Figure 11-5](#). Attach the red lead from the same motor to the second-from-left terminal block. For the other motor, put the red lead in the third-from-left terminal and the black lead in the remaining screw terminal.

Step 5: Try It Out

That's it. We are ready to go! Attach the USB dongle from the wireless keyboard to the Pi and then attach the plug from the battery lead into the power socket on the RaspiRobotBoard. The LEDs on the Raspberry Pi should flicker as it starts to boot. If this does not happen, immediately disconnect the battery and check your work.

Initially the LEDs on the RaspiRobotBoard should both be lit, but when the Python program starts to run, they should both turn off. Wait another second or two to allow the program to start up properly and then try pressing the `ARROW` and `SPACEBAR` keys on your keyboard. Your RaspiRobot should start to move!

About the Software

The software for the first phase is listed here:

```
from raspirobotboard import *  
import pygame  
import sys  
from pygame.locals import *
```



```

rr = RaspiRobot()

pygame.init()
screen = pygame.display.set_mode((640, 480))
pygame.display.set_caption('RaspiRobot')
pygame.mouse.set_visible(0)

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            sys.exit()
        if event.type == KEYDOWN:
            if event.key == K_UP:
                rr.forward()
                rr.set_led1(True)
                rr.set_led2(True)
            elif event.key == K_DOWN:
                rr.set_led1(True)
                rr.set_led2(True)
                rr.reverse()
            elif event.key == K_RIGHT:
                rr.set_led1(False)
                rr.set_led2(True)
                rr.right()
            elif event.key == K_LEFT:
                rr.set_led1(True)
                rr.set_led2(False)
                rr.left()
            elif event.key == K_SPACE:
                rr.stop()
                rr.set_led1(False)
                rr.set_led2(False)

```

NOTE If you skipped [Chapter 8](#) on pygame, now might be a good time to read through it.

The program starts by importing the library modules it needs. It then creates an instance of the class `RaspiRobot` and assigns it to the variable `rr`. The main loop first checks for a `QUIT` event, and if it find one it exists the program. The rest of the loop is concerned with checking all of the keys and issuing the appropriate commands if a key is pressed. For example, if the `UP ARROW` key (`K_UP`) is pressed, the `RaspiRobot` is sent the command `forward`, which sets the motors to both go forward as well as sets both LEDs on.

Phase 2: Adding a Range Finder and Screen

When you complete Phase 2, your `RaspiRobot` will look like the one shown earlier in [Figure 11-1](#). Disconnect the battery from the `RaspiRobotBoard` so that we can start making the necessary changes to complete this phase.

Step 1: Assemble the Range Finder Serial Adapter

The serial range finder module, shown in [Figure 11-6](#), outputs an inverted signal; therefore, a tiny

board with a single transistor and resistor on it must be used to invert the signal back to normal. Full instructions for assembling this little adaptor board can be found on the book's website (www.raspberrypibook.com).

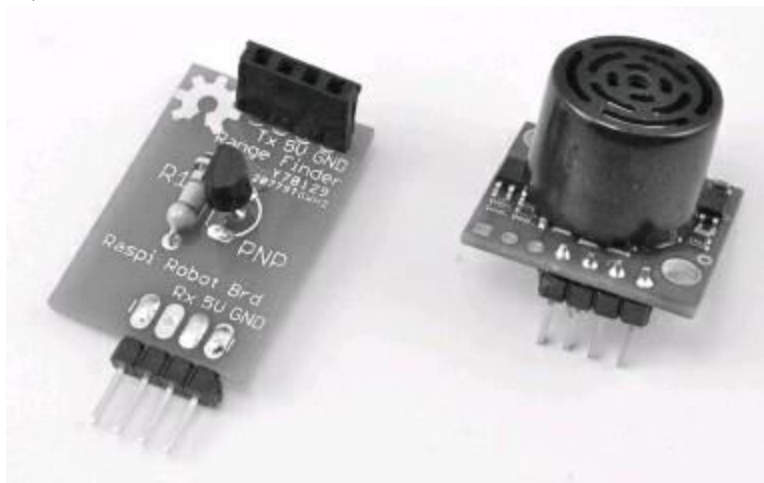


Figure 11-6 *The range finder serial adapter and range finder module*

The range finder module plugs into the top of the adapter, and the bottom of the adapter fits into the serial socket, as shown in [Figure 11-7](#).

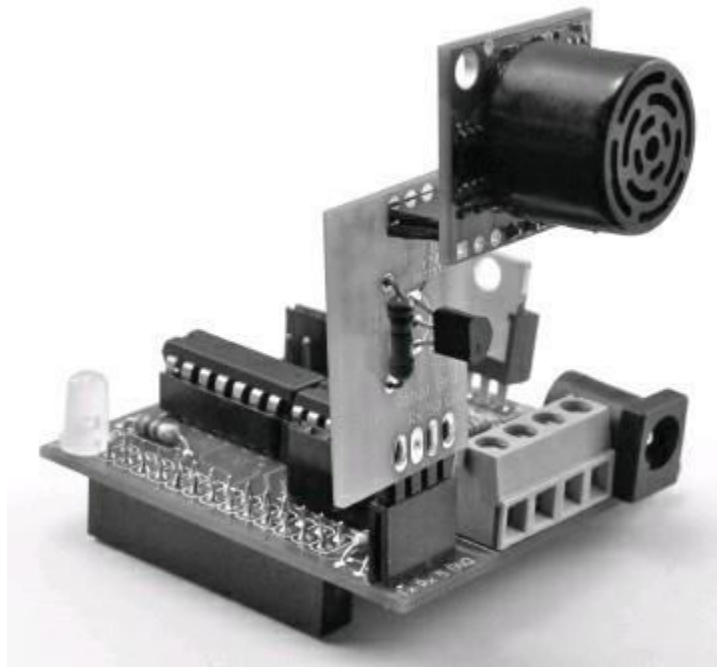


Figure 11-7 *Assembling the range finder and adapter*

Step 2: Attach the Screen

The LCD screen comes in two parts: the screen itself and the driver board. These are connected together with a rather delicate ribbon cable. I attached the two together with a cushioned self-adhesive pad. Be careful where you attach the screen, and treat the display delicately.

The screen comes with power leads (red and black) as well as two RCA plugs. To neaten things up, I cut off one of the RCA plugs (the one connected to the middle cables in the white connector plug). Refer to [Figure 11-8](#). If this seems too drastic, an alternative is to fasten it somewhere with a cable tie so that it's out of the way.

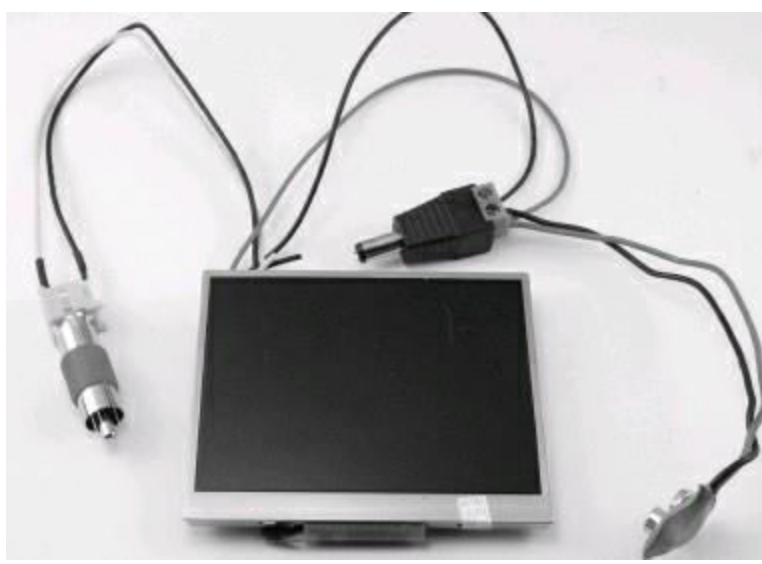


Figure 11-8 *Wiring the display*

To the remaining RCA plug I attached the male-to-male RCA adapter. The power leads are then twisted onto the power leads of the same color from the battery clip and inserted into the screw terminals of the plug adapter. If your battery box is terminated in a plug, you can snip off the plug and strip the insulation of the wires. These wires can then be used as if they were the leads from the battery clip. Either way, the project wiring is summarized in [Figure 11-9](#).

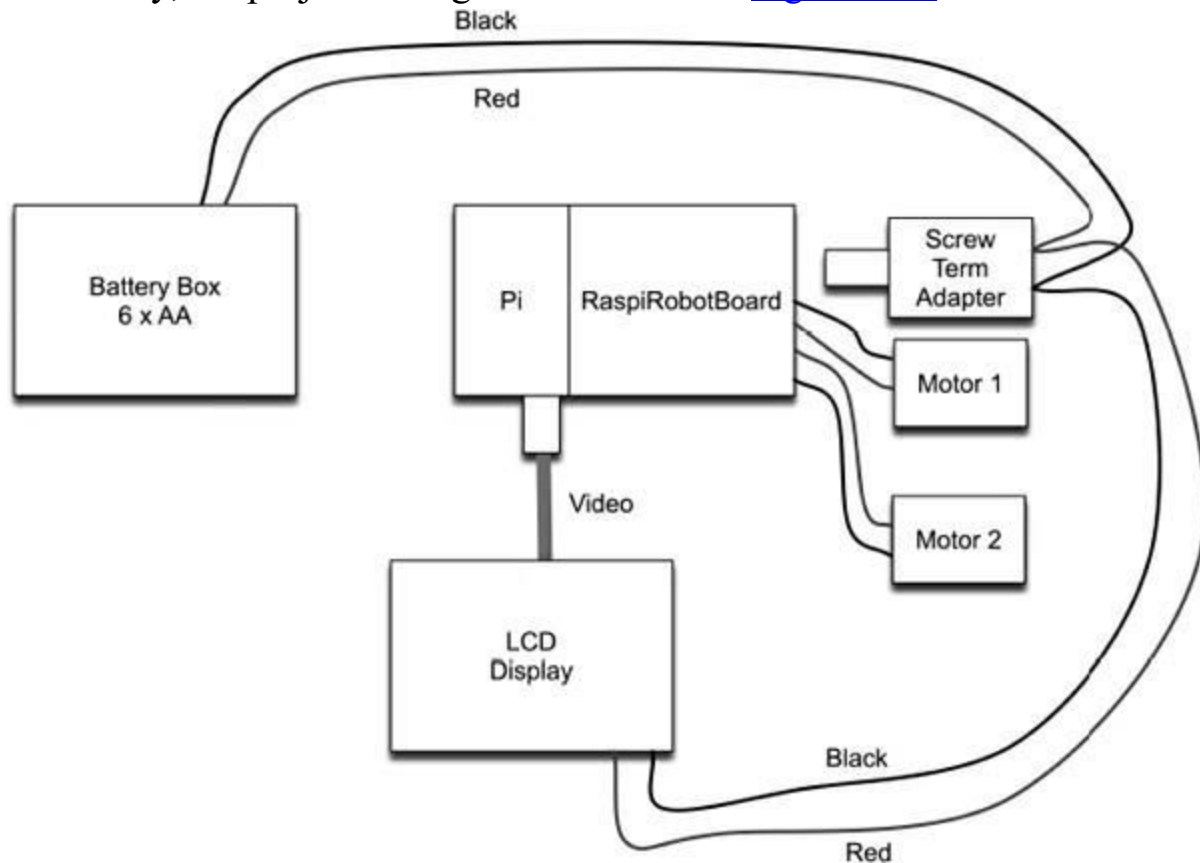


Figure 11-9 *Wiring diagram*

One consequence of this wiring arrangement is that the display will still be connected to the battery, even if you unplug the power on the RaspiRobot- Board. For this reason, use the battery clip on the battery box to turn the robot on and off.

NOTE *More adventurous readers might like to add the luxury of an on/off switch.*

The screen is attached to the chassis by means of adhesive putty. This is not a good permanent solution; some kind of plastic bracket might be better.

Step 3: Update the Software

The updated hardware needs an update to the software to accompany it. The program can be found in the file `11_02_rover_plus.py`. You also need to arrange for this program to start rather than the simpler old program. To do this, copy the file `raspirobot_plus.desktop` into the directory `/home/pi/.config/autostart` and remove the `raspirobot_basic.desktop` file from that folder; otherwise, both programs would start.

Note that because in this phase of the project, the Raspberry Pi has a screen and a keyboard (albeit a very small one), it is possible to make the changes described here, but you'll be using a tiny screen. If this proves too difficult, then as before, disconnect the battery, detach the motors, and power the Raspberry Pi from its USB power supply with its regular monitor, keyboard, and mouse.

Step 4: Run It

That's it! The project is ready to run. As always, if the LEDs on the Raspberry Pi don't come on straight away, disconnect the batteries and look for the problem. The Raspberry Pi is pretty power hungry for a battery-powered device. The screen also uses quite a lot of power. Therefore, to avoid too much recharging of batteries, you should disconnect them when not in use.

Revised Software

The new program is bigger than the old one, so it is not listed here in full. You can open it up in IDLE to take a look. The main differences are, as you would expect, the distance sensing and the display. The function `get_range` is shown here:

```
def get_range():
    try:
        dist = rr.get_range_inch()
    except:
        dist = 0
    return dist
```

This function is a very thin wrapper around a call to `get_range_inch` in the `RaspiRobot` module. The exception handling is added because if the range finder does not work for any reason (say, it isn't plugged in), exceptions will be raised. This function just intercepts any such exceptions and returns a distance of 0 if that happens.

The `update_display` function first gets the distance and then displays it along with a graphical indication of the closeness of any obstacles, as shown in [Figure 11-10](#).

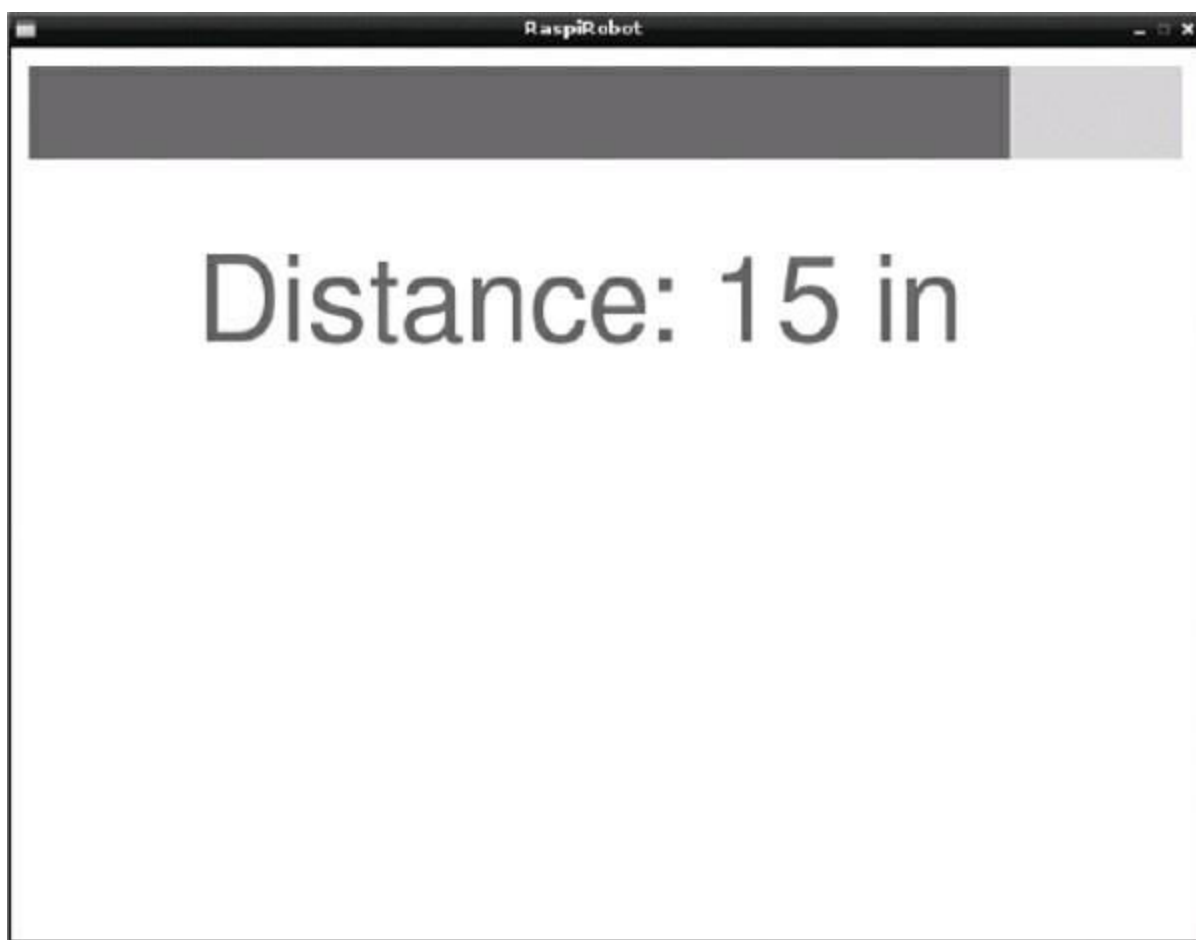


Figure 11-10 *The RaspiRobot display*

The code for this is shown here:

```
def update_distance():
    dist = get_range()
    if dist == 0:
        return
    message = 'Distance: ' + str(dist) + ' in'
    text_surface = font.render(message, True, (127, 127, 127))
    screen.fill((255, 255, 255))
    screen.blit(text_surface, (100, 100))

    w = screen.get_width() - 20
    proximity = ((100 - dist) / 100.0) * w
    if proximity < 0:
        proximity = 0
    pygame.draw.rect(screen, (0, 255, 0), Rect((10, 10), (w, 50)))
    pygame.draw.rect(screen, (255, 0, 0), Rect((10, 10), (proximity, 50)))
    pygame.display.update()
```

The distance is measured and a message is constructed into a surface that is then blitted onto the display. The graphical representation is created by drawing a fixed-size green rectangle and then drawing a red rectangle on top of it whose width depends on the distance sensed by the range finder.

Summary

This project can be treated as the basis for your own robot projects. The RaspiRobotBoard has two extra outputs that could be used to drive a buzzer or control other electronics. Another interesting way of extending the project would be to write software that allows the robot to spin on the spot, and use the range finder to create a sonar-style chart of the room containing the robot. With a Raspberry Pi camera module and a Wi-Fi dongle, all sorts of other possibilities for tele-presence devices arise!

The final chapter of this book looks at where to go next with your Raspberry Pi and provides some

useful pointers to other Raspberry Pi resources.

The Raspberry Pi is a phenomenally flexible device that you can use in all sorts of situations—as a desktop computer replacement, a media center, or an embedded computer to be used as a control system.

This chapter provides some pointers for different ways of using your Raspberry Pi and details some resources available to you for programming the Raspberry Pi and making use of it in interesting ways around the home.

Linux Resources

The Raspberry Pi is, of course, one of many computers that runs Linux. You will find useful information in most books on Linux; in particular, look for books that relate to the distribution you are using, which for Raspbian will be Debian.

Aside from the File Manager and applications that require further explanation, you'll want to know more about using the Terminal and configuring Linux. A useful book in this area is *The Linux Command Line: A Complete Introduction*, by William E. Shotts, Jr. Many good resources for learning more about Linux can be found on the Internet, so let your search engine be your friend.

Python Resources

Python is not specific to the Raspberry Pi, and you can find many books and Internet resources devoted to it. For a gentle introduction to Python, you might want to pick up *Python: Visual QuickStart Guide*, by Toby Donaldson. It's similar to this book in style, but provides a different perspective. Also, it's written in a friendly, reassuring manner. If you want something a bit more meaty, but still essentially a beginner's text, consider *Python Programming: An Introduction to Computer Science*, by John Zelle.

When it comes to learning more about Pygame, you'll find *Beginning Game Development with Python and Pygame*, by Will McGugan, to be quite helpful.

Finally, here are some good web resources for Python you'll probably want to add to your browser's favorites list:

- <http://docs.python.org/py3k/> The official Python site, complete with useful tutorials and reference material.
- www.pythonware.com/library/tkinter/introduction/ A useful reference for Tkinter.
- <http://zetcode.com/gui/tkinter/layout/> This tutorial sheds some much needed light on laying out widgets in Tkinter.
- www.pygame.org The official Pygame site. It contains news, tutorials, reference material, and sample code.

Raspberry Pi Resources

The official website of the Raspberry Pi Foundation is www.raspberrypi.org. This website contains a wealth of useful information, and it's the place to find announcements relating to happenings in the world of Raspberry Pi.

The forums are particularly useful when you are looking for the answer to some knotty problem. You can search the forum for information from others who have already tried to do what you are trying to do, you can post questions, or you can just show off what you've done to the community. When you're looking to update your Raspberry Pi distribution image, this is probably the best place to turn. The downloads page lists the distributions currently in vogue.

The Raspberry Pi even has its own online magazine, wittily named *The MagPi*. This is a free PDF download (www.themagpi.com) and contains a good mixture of features and “how-to” articles that will inspire you to do great things with your Pi.

For more information about the hardware side of using the Raspberry Pi, the following links are useful:

- http://elinux.org/RPi_VerifiedPeripherals A list of peripherals verified as working with the Raspberry Pi.
- http://elinux.org/RPi_Low-level_peripherals A list of peripherals for interfacing with the GPIO connector.
- www.element14.com/community/docs/DOC-43016/ A datasheet for the Broadcom chip at the heart of the Raspberry Pi. (This is not for the faint of heart!)

If you are interested in buying hardware add-ons and components for your Raspberry Pi, Adafruit has a whole section devoted to the Raspberry Pi. SparkFun also sells Raspberry Pi add-on boards and modules.

Other Programming Languages

In this book, we have looked exclusively at programming the Raspberry Pi in Python, and with some justification: Python is a popular language that provides a good compromise between ease of use and power. However, Python is by no means the only choice when it comes to programming the Raspberry Pi. The Raspbian Wheezy distribution includes several other languages.

Scratch

Scratch is a visual programming language developed by MIT. It has become popular in education circles as a way of encouraging youngsters to learn programming. Scratch includes its own development environment, like IDLE for Python, but programming is carried out by dragging and dropping programming structures rather than simply typing text.

Figure 12-1 shows a section of one of the sample programs provided with Scratch for the game *Pong*, where a ball is bounced on a paddle.

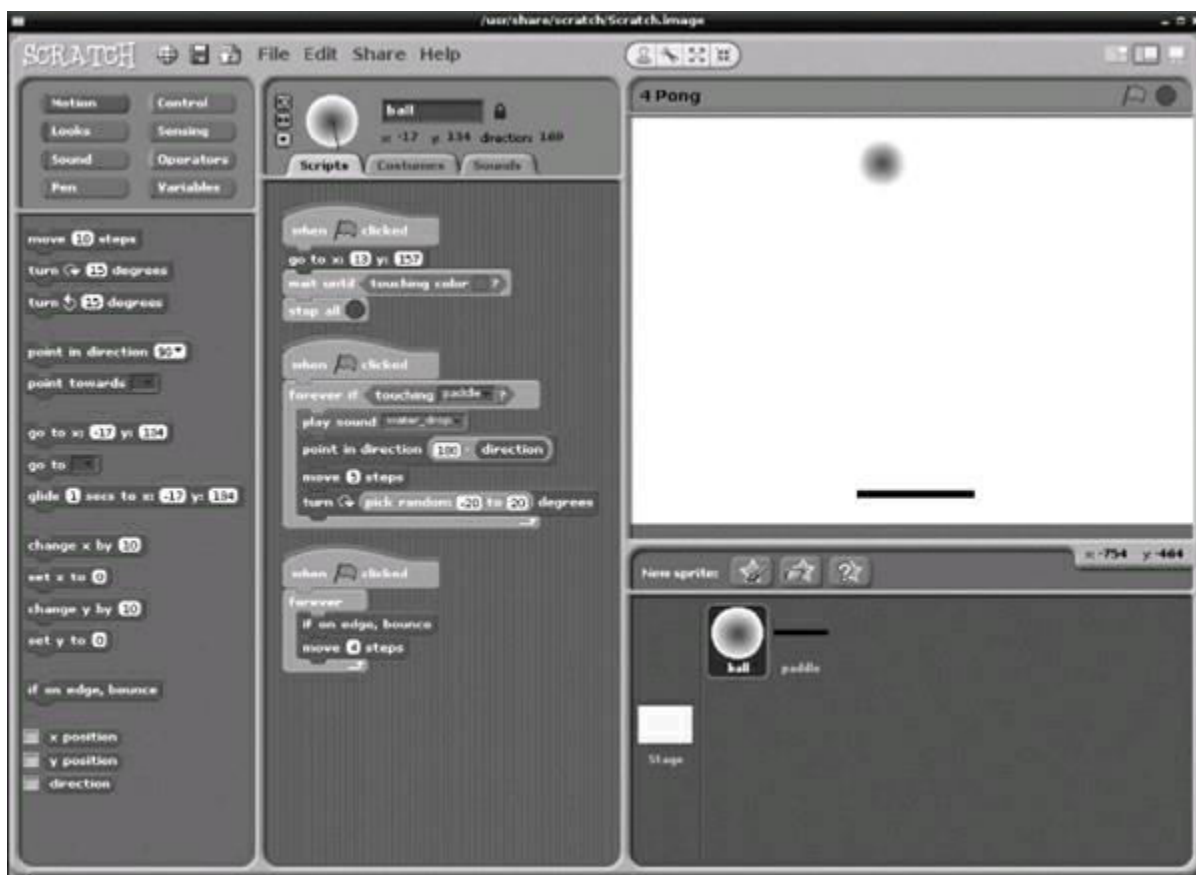


Figure 12-1 *Editing a program in Scratch*

C

The C programming language is the language used to implement Linux, and the GNU C compiler is included as part of the Raspbian Wheezy distribution. To try out a little “Hello World” type of program in C, use IDLE to create a file with the following contents:

```
#include<stdio.h>
main()
{
    printf("\n\nHello World\n\n");
}
```

Save the file, giving it the name `hello.c`. Then, from the same directory as that file, type the following command in the terminal:

```
gcc hello.c -o hello
```

This will run the C compiler (`gcc`), converting `hello.c` into an executable program called just `hello`. You can run it from the command line by typing the following:

```
./hello
```

The IDLE editor window and command line are shown in [Figure 12-2](#), where you can also see the output produced. Notice that the `\n` characters create blank lines around the message.

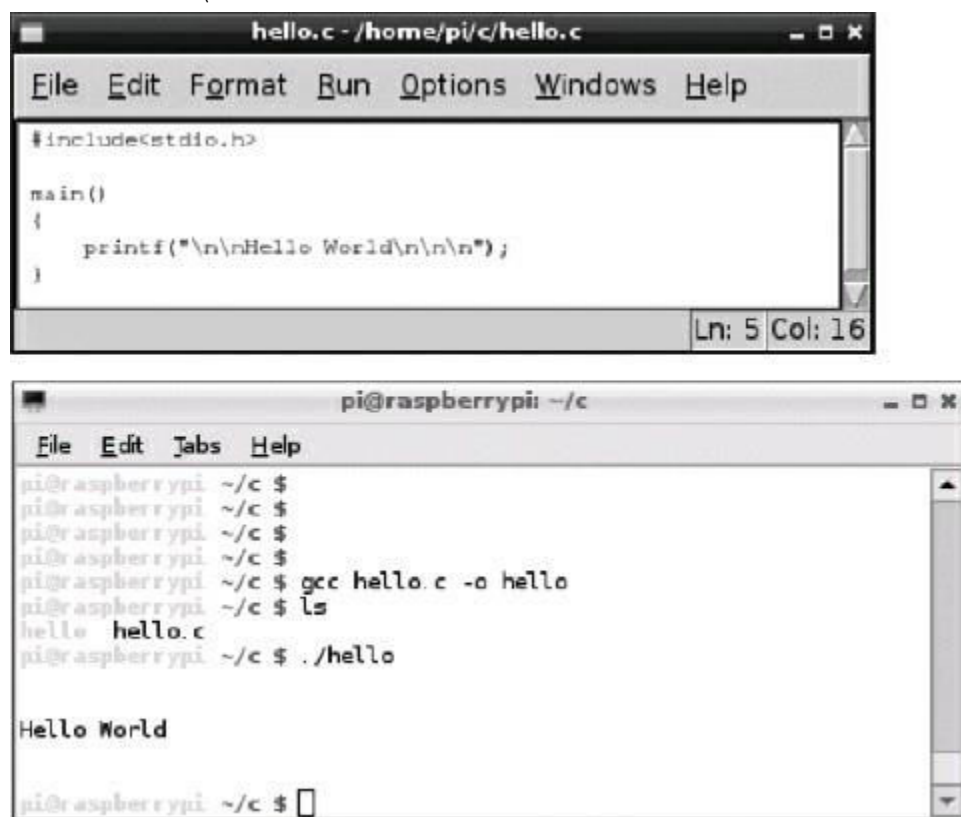


Figure 12-2 *Compiling a C program*

Applications and Projects

Any new piece of technology such as the Raspberry Pi is bound to attract a community of innovative enthusiasts determined to find interesting uses for the Raspberry Pi. At the time of writing, a few interesting projects were in progress, as detailed next.

Media Center (Raspbmc)

Raspbmc is a distribution for the Raspberry Pi that turns it into a media center you can use to play movies and audio stored on USB media attached to the Pi, or you can stream audio and video from other devices such as iPads that are connected to your home network. Raspbmc is based on the successful XBMC project, which started life as a project to use Microsoft Xboxes as media centers.

However, it's available on a wide range of platforms.

With the low price of the Raspberry Pi, it seems likely that a lot of them will find their way into little boxes next to the TV—especially now that many TVs have a USB port that can supply the Pi with power.

You can find out more about Raspbmc at www.raspbmc.com/about/, you can learn about the XBMC project at www.xbmc.org. All the software is, of course, open source.

Home Automation

Many small-scale projects are in progress that use the Raspberry Pi for home automation, or *domotics* as it is also known. The ease with which sensors and actuators can be attached to it, either directly or via an Arduino, make the Pi eminently suitable as a control center.

Most approaches have the Raspberry Pi hosting a web server on the local network so that a browser anywhere on the network can be used to control various functions in the home, such as turning lights on and off or controlling the thermostat.

Summary

The Raspberry Pi is a very flexible and low-cost device that will assuredly find many ways of being useful to us. Even as just a simple home computer for web browsing on the TV, it is perfectly adequate (and much cheaper than most other methods). You'll probably find yourself buying more Raspberry Pi units as you start to embed them in projects around your home.

Finally, don't forget to make use of this book's website (www.raspberrypibook.com), where you can find software downloads, ways of contacting the author, as well as errata for the book.

Symbols

.""" (quotes), defining strings in Python
 ". """" (double quotes), defining strings in Python
 / (divide), working with numbers in Python Shell
 / (slash), in directory syntax
 \ (backslash), as escape character
 \ (line-continuation command), breaking line into two lines
 + (addition) operator, using with lists
 = (assignment). *See* assignment (=)
 == (comparison operator), in Python
 * (multiply), working with numbers in Python Shell
 [] (square brackets), in Python syntax for lists

Numbers

12C bus, making accessible to Python
 30 *Arduino Projects for the Evil Genius* (Monk)

A

a (append) file mode
 Abiword word processor
 actuators, attaching to Raspberry Pi
 Adafruit

- parts used in LED digital clock project
- Pi Cobbler from
- Pi Plate from

 addition (+) operator, using with lists
 alert dialog
 animation, benefits of Pygame in creating
 append (a) file mode
 applications, installing/uninstalling with `apt-get` command. *See also* software
`apt-get` command, installing/ uninstalling applications
 Arduino boards

- connecting to Raspberry Pi via USB
- overview of
- Raspberry Pi software and
- software for

 arithmetic, typing commands in Python shell
 arrays, variable
 assignment (=)

- of string value
- using with lists
- of value to variable

 audio socket, in tour of Raspberry Pi

B

batteries

- adding battery box to Magician chassis

caution regarding attaching to Raspberry Pi

Beginning Game Development with Python and Pygame (McGugan)

blit command, for drawing images at coordinates

boot up, Raspberry Pi for first time

breadboard

from Adafruit

attaching jumper wires to

`break` command, for breaking out of loops

Broadcom chip

datasheet for

System on a Chip from

bumpy case, variable convention

button, adding to LED digital clock

C

C language, included with Raspbian Wheezy distribution

cable connectors, in tour of Raspberry Pi

camel case, variable convention

camera, cable connector for

Canvas interface, Tkinter

case, gathering components for Raspberry Pi setup

`cd` (change directory) command, for changing current directory

chassis, assembling for robot rover project

Checkbutton widget

classes

creating a `Raspberry` class

defining

inheritance and

of objects

clock

GPIO pin for

managing speed element of game

software for LED digital clock project

Color Chooser, in GUI

`columnconfigure` command, in layouts

command line

`cd` (change directory) command

`ls` (list) command

overview of

`pwd` (print working directory) command

`sudo` (super-user do)

comments, use in Python

comparison operator (`==`), in Python

connections, GPIO pins

`convert` method

`converters` module

- coordinates, drawing images at
- `copytree` function, for moving or changing names of files or directories

D

- data lines, GPIO pin for
- `datetime` module
- `def` keyword, starting function with
- `description` method, class methods
- desktop
 - booting to
 - LXDE windowing environment
 - overview of
- dialogs, in GUI
- dice simulation, generating random numbers in
- dictionaries
 - functions
 - overview of
- directories
 - globbing
 - moving or changing name of
 - navigating
- display
 - attaching LCD screen to robot rover
 - gathering components for Raspberry Pi setup
 - Raspberry Pi video adapters

- distros (Linux distributions)

- domotics, uses of Raspberry Pi for home automation

- Don't Repeat Yourself (DRY) principle, in programming

- double quotes ("."), defining strings in Python

- `DoubleVar`

- drawing shapes, with Canvas interface

- DRY (Don't Repeat Yourself) principle, in programming

- DVI connectors

 - gathering components for Raspberry Pi setup

 - Raspberry Pi video adapters

E

- educational use, Pi Face expansion board intended for

- `elif` command

- `else` logic, in Python

- encapsulation, classes and

- equals (=), assigning values to variables

- errors, flagging

- escape characters, including special characters in strings

- Ethernet cable, getting Raspberry Pi online

- exceptions

- expansion boards

- Gertboard expansion board
- overview of
- Pi Face expansion board
- RaspiRobotBoard expansion board
- Slice of PI/O expansion board

F

- false, logical values in Python
- File Browser, creating new files and folders
- File Chooser, in GUI
- File Explorer, locating files with
- File Manager
 - on Raspberry Pi desktop
 - viewing available free space
- File menu
- files
 - creating
 - file-system operations on
 - Internet and
 - moving or changing name of
 - overview of
 - pickling
 - reading
 - writing
- file-system
- floating point numbers
 - double-precision
 - working with numbers in Python Shell
- folders
 - creating
 - moving or changing name of
- for in command
- for loops, in Python
- functions
 - dictionaries
 - grouping. *See* modules lists
 - numbers
 - overview of
 - strings
 - stubs as stand-in for
 - type conversion

G

- games programming
 - Hangman game. *See* Hangman game
 - Hello World in Pygame
 - overview of

Pygame library

Raspberry game example. *See* Raspberry game example

General Purpose Input/Output pins. *See* GPIO (General Purpose Input/Output) pins

Gertboard expansion board

global variables

use in Hangman game

use in Raspberry game

globbing, for finding out what is in a directory

GNU C compiler, in Raspbian Wheezy distribution

Gnumeric spreadsheet

GPIO (General Purpose Input/ Output) pins

adding button to LED digital clock

connections

expansion boards

Gertboard expansion board

Humble Pi prototyping board

overview of

peripherals that interface with

Pi Cobbler prototyping board

Pi Face expansion board

Pi Plate prototyping board

prototyping boards

RaspiRobotBoard expansion board

Slice of PI/O expansion board

in tour of Raspberry Pi

graphical user interface. *See* GUI (graphical user interface)

graphics

adding images to games

benefits of Pygame in creating

GUI (graphical user interface)

Checkbutton widget

Color Chooser

dialogs

File Chooser

Hello World in Tkinter

layouts

Listbox widget

menus

overview of

scrollbars

Spinbox widget

temperature converter example

Tkinter

Tkinter Canvas interface

widgets

H

Hangman game

- converting to work with files

- overview of

hardware

- adding range finder to robot rover

- Arduino boards

- Arduino software and

- assembling for LED digital clock project

- assembling RaspiRobotBoard kit

- assembling robot chassis

- attaching LCD screen to robot rover

- connecting Arduino boards to Pi via USB

- connecting motors to robot rover

- expanding LED digital clock project

- expansion boards

- Gertboard expansion board

- GPIO connections

- Humble Pi prototyping board

- overview of

- Pi Cobbler prototyping board

- Pi Face expansion board

- Pi Plate prototyping board

- Pi software and

- prototyping boards

- RaspiRobotBoard expansion board

- Slice of PI/O expansion board

- updating software to accommodate hardware added to robot

HDMI (High-Definition Multimedia Interface)

- gathering components for Raspberry Pi setup

- Raspberry Pi video adapters

- in tour of Raspberry Pi

Hello World

- in C

- in Tkinter

High-Definition Multimedia Interface. *See* HDMI (High-Definition Multimedia Interface)

home automation, uses of Raspberry Pi

HTML (Hypertext Markup Language)

HTTP (Hypertext Transfer Protocol)

Humble Pi prototyping board

I

IDLE program

- accessing

- creating new file

- editing in Python

- running hangman game in

- running new programs

- `if` command

- images

- adding to games

- converting and using in Raspberry game

- immutability, of tuples, strings, and numbers

- importing, creating pygame and

- inheritance

- `init` method

- `input` function

- input/output (I/O). *See* GPIO (General Purpose Input/Output) pins

- `insert` command

- instances, class

- integers, working with numbers in Python Shell

- Internet

- connecting Raspberry Pi to

- resources for Raspberry Pi

- web services in Python

- I/O (input/output). *See* GPIO (General Purpose Input/Output) pins

J

- jumper wires, attaching to breadboard

K

- keyboards

- controlling game play with

- gathering components for Raspberry Pi setup

- keys, to dictionary values

L

- layouts, in GUI

- LCD screen

- attaching to robot rover

- cable connector for

- LED digital clock. *See* prototyping project (LED digital clock)

- LEDs

- on Gertboard expansion board

- on Pi Face expansion board

- on RaspiRobotBoard expansion board

- on Slice of Pi expansion board

- `len`

- using with lists

- using with strings

- libraries

- installing for RaspiRobot expansion board

- Pygame

- standard library of modules

line-continuation command (\), breaking line into two lines
The Linux Command Line: A Complete Introduction (Shotts)

Linux OS

- distributions (distros)

- Raspberry Pi running

- resources for

Listbox widget

lists

- functions

- overview of

logical values, in Python

looping

- for loops

- while loops

ls (list) command

LXDE windowing environment

LXTerminal

M

Magician Chassis kit, as basis for robot rover

The MagPi magazine

math module

media center, turning Raspberry Pi into

menus, in GUI

methods, class

micro USB socket, in tour of Raspberry Pi

microcontrollers

- Arduino boards as

- on Gertboard expansion board

Midori web browser, connecting to Internet via

Mobel B, comparing Raspberry Pi models

modal dialogs

Model A, comparing Raspberry Pi models

modules

- converting file into

- installing new

- object orientation compared with

- overview of

- standard library of

- using

monitors. *See* display

motor controller

- Arduino boards as

- in Gertboard expansion board

- in RaspiRobotBoard

motors

- caution regarding attaching to Raspberry Pi
- connecting to robot rover

mouse

- controlling game play with
- gathering components for Raspberry Pi setup
- tracking movement in Raspberry game

multiple assignment

N

networking, connecting Raspberry Pi to Internet

numbers

- functions
- in Python

O

object orientation

open source, Linux OS as

output. *See* GPIO (General Purpose Input/Output) pins

P

packages, installing/uninstalling. *See also* software

parameters, parentheses in syntax of

passwords, `sudo` (super-user do) and

peripherals

- GPIO pin for
- verified for use with Raspberry Pi

Pi Cobbler prototyping board

- assembling hardware for LED digital clock
- overview of

- use in LED digital clock project

Pi Face expansion board

Pi Plate prototyping board

`pickle` module

pickling files

`pop` command, removing items from lists

power supplies

- gathering components for Raspberry Pi setup
- in tour of Raspberry Pi

`print` command

Programming Arduino: Getting Started with Sketches (Monk)

programming languages, included with Raspbian Wheezy distribution

programs. *See* software

prototyping boards

- Humble Pi prototyping board
- overview of

- Pi Cobbler prototyping board

- Pi Plate prototyping board

prototyping project (LED digital clock)

- assembling hardware for
- creating software for
- expanding hardware and software capabilities
- overview of
- parts needed for

pulse with modulation, GPIO pin for

`pwd` (print working directory) command, for showing current directory

Pygame

- Hello World application in
- installing pygame module
- library

- Raspberry game example. *See* Raspberry game example
- resources for

PySerial package, Python code talking to Arduino and

Python basics

- comparison operators
- editor for
- `else` logic and
- generating random numbers in dice simulation
- logical values
- for loops
- overview of
- Python Shell
- regular expressions in
- resources for
- variables
- versions of Python
- while loops
- working with numbers

Python Programming: An Introduction to Computer Science (Zelle)

Python Shell

- typing commands in
- working with numbers

Python: Visual QuickStart Guide (Donaldson)

Q

quotes (`""`), defining strings in Python

R

`r` (read) file mode

`r+` (read and write) file mode

radio buttons, in layouts

`randint` function

- generating random numbers in
- in random module

random module

random numbers, generating in dice simulation

- range command
- range finder, adding to robot rover
- Rasbmc, for media center
- Raspberry game example
 - adding raspberries to
 - catching raspberries and displaying game score
 - creating a `Raspberry` class and refactoring
 - managing speed element of
 - overview of
 - tracking movement of mouse in
- Raspberry Pi foundation
 - overview of
 - website
- Raspberry Pi, introduction
 - booting up for first time
 - components needed for setting up
 - resources for
 - setting up
 - tour of
 - what it is
 - what you can do with it
- Raspbian Wheezy
 - browsing packages available for
 - Linux distributions with
 - making I2C bus accessible to Python
 - programming languages included with
 - Python versions in
 - recommended Linux distribution for Raspberry Pi
 - sparseness of
- RaspiRobotBoard expansion board
 - assembling kit for robot rover project
 - installing library for
 - overview of
- `raw_input` function, in Python
- RCA video connector
 - on LCD screen
 - in tour of Raspberry Pi
- `read (r)` file mode
- `read and write (r+)` file mode
- receive (Rx), GPIO pin for
- refactoring, Raspberry game example
- regular expressions, in Python
- resources
 - for Linux OS
 - for Python

- for Raspberry Pi
- return values
 - functions and
 - multiple
- RGB color
- ribbon cable, attaching Pi Cobbler to Raspberry Pi
- RJ-45 connectors, connecting Raspberry Pi to Internet
- `rmtree` function, for removing directories
- robot rover project
 - adding range finder
 - assembling chassis for
 - assembling RaspiRobotBoard kit
 - attaching LCD screen to
 - connecting motors to
 - installing software for
 - overview of
 - parts needed for
 - testing
 - updating software to accommodate added hardware
- robots, using RaspiRobotBoard as controller
- root directories, navigating
- root Menu
- `rowconfigure` command, in layouts
- Run Module command, running new programs
- S**
- scoring functionality, adding to games
- Scratch programming language
 - languages included with Raspbian Wheezy distribution
 - Pi Face expansion board integrated with
- scrollbars
 - in GUI
 - Text widget with
- SD card slot, in tour of Raspberry Pi
- SD cards
 - configuring on boot up
 - gathering components for Raspberry Pi setup
 - replacing hard drive in Raspberry Pi
- `selectmode` property, Listbox widget
- sensors
 - Arduino boards and
 - communicating with
 - ease of attaching to Raspberry Pi
 - expansion boards and prototyping boards and
- serial adapter, for range finder
- serial communication, GPIO pin for

- serial interface (MCP23S17)
 - Pi Face expansion board connected via
 - Slice of Pi expansion board connected via
- `shutil` (shell utility) package, file system functions in
- slash (/), in directory syntax
- Slice of PI/O expansion board
- software
 - adding for LED digital clock project
 - Arduino software
 - expanding LED digital clock project
 - installing for robot rover project
 - Raspberry Pi software
 - updating robot software to accommodate added hardware
- soldering
 - Adafruit modules
 - prototyping boards requiring
- `sort`, using with lists
- speed (or timing), managing speed element of game
- Spinbox widget
- spreadsheet, Gnumeric
- square brackets ([]), in Python syntax for lists
- standard library, of modules
- `sticky` attributes, use in layouts
- `string` module
- strings
 - functions
 - overview of
- StringVar
- stubs, stand-ins for functions
- `sudo` (super-user do)
- super-user do (`sudo`)
- System on a Chip, from Broadcom
- T**
- temperature converter example
- Text widget, with scrollbars
- timing (speed), managing speed element of game
- Tk GUI system
- Tkinter
 - Canvas interface
 - Checkbutton widget
 - Color Chooser
 - creating GUI with
 - dialogs
 - File Chooser
 - Hello World in

- layouts in
- Listbox widget
- menus
- resources for
- scrollbars in
- Spinbox widget
- temperature converter example
- `tkinter` module

- transmit (Tx), GPIO pin for
- true, logical values in Python
- `try` command, file-reading code in
- tuples

- TV, using as monitor
- type conversion functions

- U**
- `urllib.request` module

- USB**
 - connecting Arduino boards to Pi via
 - for keyboard and mouse
 - in tour of Raspberry Pi
 - for Wi-Fi

- USB hub

- V**
- values
 - assigning to strings
 - assigning to variables
 - keys to dictionary values
 - logical values in Python

- van Loo, Gert

- variables
 - assignment of value to
 - global
 - lists or arrays
 - in Python
 - `Raspberry` class
 - saving contents to a file (pickling)
 - viewing variable content
- voltage, GPIO pins rated at 3.3V
- voltage regulators
 - on Humble Pi prototyping board
 - on RaspiRobotBoard expansion board

- W**
- `w` (`write`) file mode
- web resources, for Python
- web scraping

web services, in Python

while loops

in Python

use in Raspberry game

widgets

Checkbutton widget

Listbox widget

overview of

Spinbox widget

Wi-Fi, USB wireless adapter for

windows, resizing

word processors

write (w) file mode

X

XBMC project