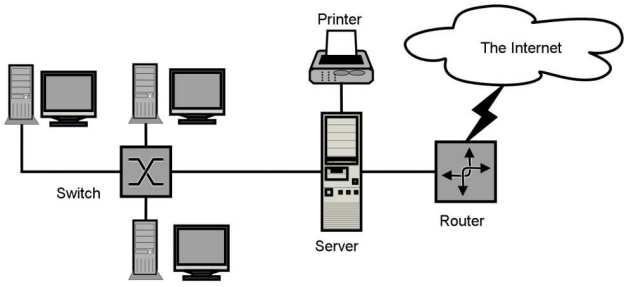
# WIRELESS SENSOR NETWORKS

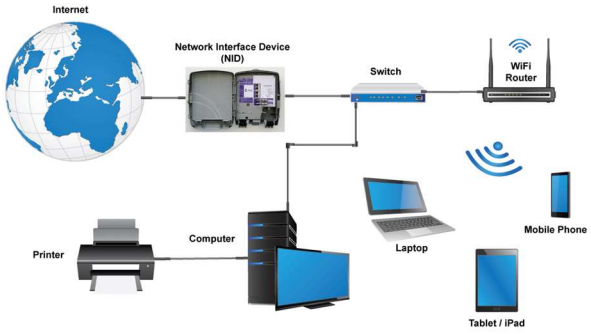
## COMPUTER NETWORK

- A **computer network** is any group of interconnected computing devices capable of sending or receiving data. A **computing device** isn't just a computer—it's any device that can run a program, such as a tablet, phone, or smart sensor.
- The computing devices are connected to each other through communication media like Co-axial Cable, CAT 6 Cable, Fiber Optic Cable, Radio Waves, Microwaves etc.
- Switch, Router, Hub and NICs (Network Interface Cards) are used to form computer network.



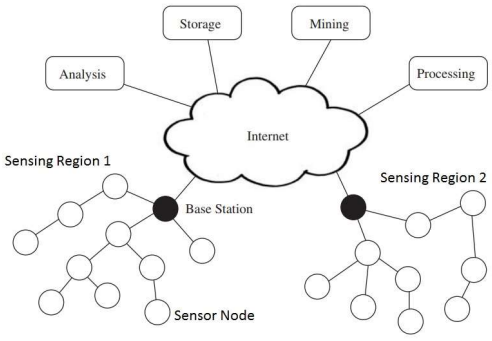
## WIRED Vs WIRELESS NETWORK

- A **wireless local-area network (LAN)** uses radio waves to connect devices such as laptops and mobile phones to the Internet and to your business network and its applications.
- A **wired network** uses cables to connect devices, such as laptop or desktop computers, to the Internet or another network. A wired network has some disadvantages when compared to a wireless network. The biggest disadvantage is that your device is tethered to a router. The most common wired networks use cables connected at one end to an Ethernet port on the network router and at the other end to a computer or other device.




## Wireless Sensor Networks

- A **wireless sensor network** is an infrastructure-less wireless network that is deployed in a large number of wireless sensors in an ad-hoc manner that is used to monitor the system, physical or environmental conditions.
- Sensor nodes are used in WSN with the onboard processor that manages and monitors the environment in a particular area. They are connected to the Base Station which acts as a processing unit in the WSN System.
- **Base Station** in a WSN System is connected through the Internet to share data.
- WSN can be used for processing, analysis, storage, and mining of the data.

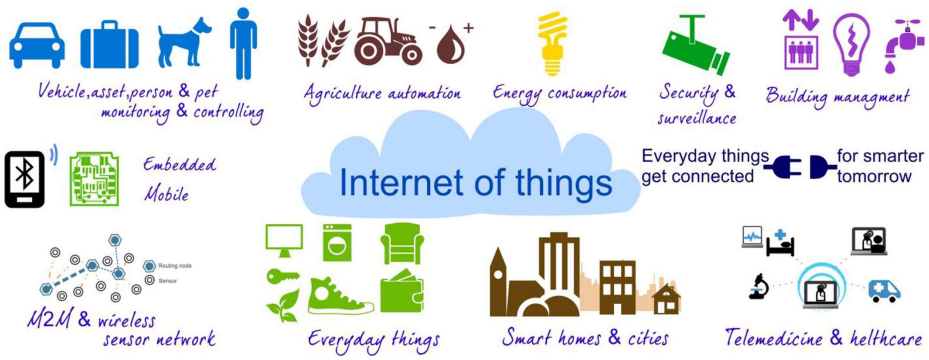


## Wireless Sensor Networks

- A **Wireless sensor network (WSN)** is a collection of sensor nodes that distributed in an arbitrary manner to solve a particular problem. The position of the node is predefined and based on random nature. Each node directly or indirectly connected with the base station (BS). BS is used to control and manages all sensor nodes.
- A Wireless Sensor Network consists of Sensor Nodes that are deployed in high density and often in large quantities and support sensing, data processing, embedded computing and connectivity.
- WSN is a wireless network that consists of base stations and numbers of nodes (wireless sensors). These networks are used to monitor physical or environmental conditions like sound, pressure, temperature, and co-operatively pass data through the network to the main location.

## IoT (Internet of Things)

- The **Internet of Things (IoT)** describes the network of physical objects—"things"—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet.
- The **Internet of Things** is the concept of connecting any device (so long as it has an on/off switch) to the Internet and to other connected devices. The IoT is a giant network of connected things and people – all of which collect and share data about the way they are used and about the environment around them.



## IoT (Internet of Things) Vs Wireless Sensor Networks (WSN)

| IoT | WSN |
|---|---|
| In an IoT system, all of the sensors directly send their information to the internet. For example, a sensor may be used to monitor the temperature of a body of water. In this case, the data will be immediately or periodically sent directly to the internet, where a server can process the data and it can be interpreted on a front-end interface. | Conversely, in a WSN, there is no direct connection to the internet. Instead, the various sensors connect to some kind of router or central node. A person may then route the data from the router or central node as they see fit. That being said, an IoT system can utilize a wireless sensor network by communicating with its router to gather data. |
| IoT exists at a higher level then WSN. In other words, WSN is often a technology used within an IoT system. | A large collection of sensors, as in a mesh network, can be used to individually gather data and send data through a router to the internet in an IoT system. |
| A fridge with the capability of sending temperature reading to the internet is unlikely to use a wireless sensor network but it is an IoT device. | A large collection of sensors used to monitor precipitation on an acre of land is likely to be considered a "wireless sensor network" if in fact all the sensors are wireless. This system may or may not be connected to an IoT system. |

## Wireless Sensor Network

**Why Sensors?**

- A sensor is a connection between the physical environment to the digital world for collecting and finding the real world information and convert them into a digital form or signal form to process, store and perform.
- Sensors can be integrated with the devices, machines and environment for various befits.
- Sensors are manufactured and developed by the advancement of the new technologies such as VLSI (Very Large Scale Integration), MEMS (Microelectromechanical System) and Wireless Communication.
- Sensors and Microcontrollers are tiny, low powered and inexpensive miniature devices.
- Sensors are used in applications of numerous areas such as
  - Structural Monitoring
  - Forecasting of Environmental Pollution and Flooding
  - Weather Monitoring
  - Precision Agriculture
  - Active Volcano Monitoring
  - Smart Transportation
  - Health Care Monitoring
  - Underground Mining
  - Tracking Chemical Plumes

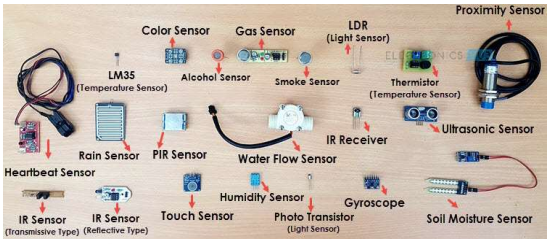
## Wireless Sensor Network

**What is Sensing?**

- Sensing is a technique to collect the information about the object, process and any event from the surrounding environment and convert them into readable form for the further processing.

**What is Sensor?**

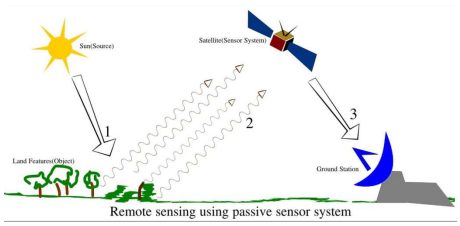
- An object or device which is used to perform the sensing task called Sensor.
- A sensor is a device that measures physical input from its environment and converts it into data that can be interpreted by either a human or a machine.
- Most sensors are electronic (the data is converted into electronic data), but some are more simple, such as a glass thermometer, which presents visual data. People use sensors to measure temperature, gauge distance, detect smoke, regulate pressure and a myriad of other uses.




## Wireless Sensor Network

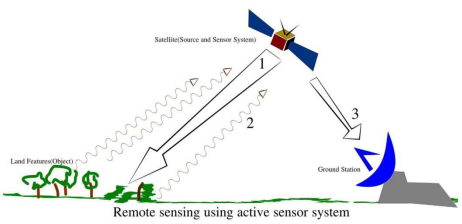
**Remote Sensor and Remote Sensing**

- A Remote Sensor is a device that don't touch the monitored object to gather the information and converts it into data that can be interpreted by either a human or a machine.
- The sensing units of human body like Eye, Ear and Nose are example of Remote Sensors as they don't need to touch the monitored object to collect the information.
- Remote sensing is the process of detecting and monitoring the physical characteristics of an area by measuring its reflected and emitted radiation at a distance (typically from satellite or aircraft). Special cameras collect remotely sensed images, which help researchers "sense" things about the Earth.




Remote sensing using passive sensor system

**Passive Remote Sensing**



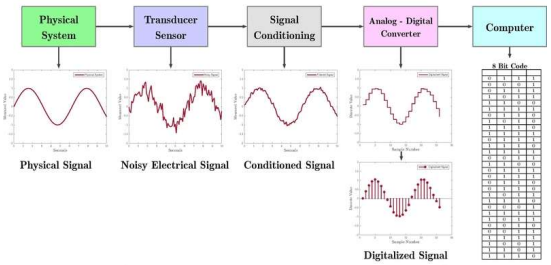

Remote sensing using active sensor system

**Active Remote Sensing**

## Wireless Sensor Network

### Data Acquisition

- A **transducer** is a device that converts energy from one form to another. Usually a **transducer** converts a signal in one form of energy to a signal in another.
- A sensor is a type of Transducer that converts different forms of energy into electrical energy and pass it to the controller and system for processing.
- The Sensing process is also known as Data Acquisition. **Data acquisition** is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer.
- The components of data acquisition systems include:
  - Sensors, to convert physical parameters to electrical signals.
  - Signal conditioning circuitry, to convert sensor signals into a form that can be converted to digital values.
  - Analog-to-digital converters, to convert conditioned sensor signals to digital values.



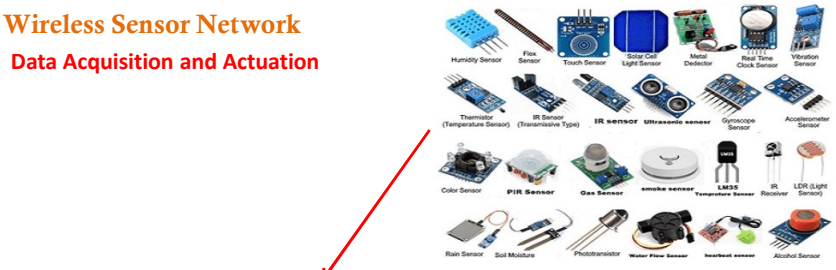

## Wireless Sensor Network

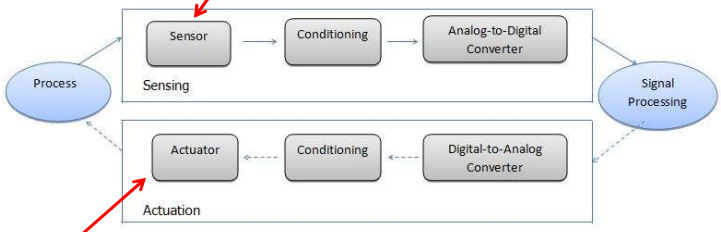
### Data Actuation

- An actuator is a device that activates process control equipment by using pneumatic, hydraulic, or electrical power. For example, a valve actuator opens and closes a valve to control fluid rate.
- An **actuator** is a component of a machine that is responsible for moving and controlling a mechanism or system, for example by opening a valve. In simple terms, it is a "mover".
- An actuator requires a control signal and a source of energy. The control signal is relatively low energy and may be electric voltage or current, pneumatic, or hydraulic fluid pressure, or even human power. Its main energy source may be an electric current, hydraulic pressure, or pneumatic pressure. When it receives a control signal, an actuator responds by converting the source's energy into mechanical motion. In the *electric*, *hydraulic*, and *pneumatic* sense, it is a form of automation or automatic control.
- An actuator is a mechanism by which a control system acts upon to perform an operation or task. The control system can be simple (a fixed mechanical or electronic system), software-based (e.g. a printer driver, robot control system), a human, or any other input.

## Wireless Sensor Network

### Data Acquisition and Actuation



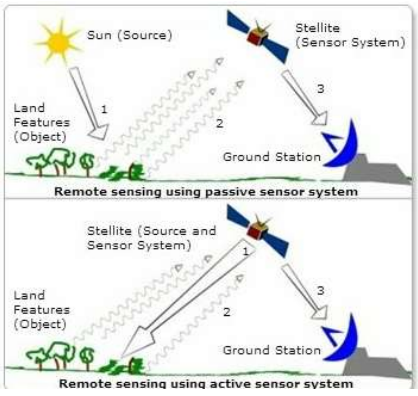





## Wireless Sensor Network

### Classification of Sensors

- **First Classification: Based on external power supply**
  1. Active Sensor: Active Sensors are those which require an external excitation signal or a power signal. e.g. Solar Cell, Piezoelectric Material and Thermocouple, etc.
  2. Passive Sensor: Passive Sensors, on the other hand, do not require any external power signal and directly generates output response. e. g. Photodiode, Strain Gauge, Thermistor etc.




- **Passive sensors** detect natural energy (radiation) that is emitted or reflected by the object or scene being observed. Reflected sunlight is the most common source of radiation measured by passive sensors.
- **Active sensors**, on the other hand, provide their own source of energy to illuminate the objects they observe. An active sensor emits radiation in the direction of the target to be investigated. The sensor then detects and measures the radiation that is reflected or backscattered from the target.

## Wireless Sensor Network

### Classification of Sensors

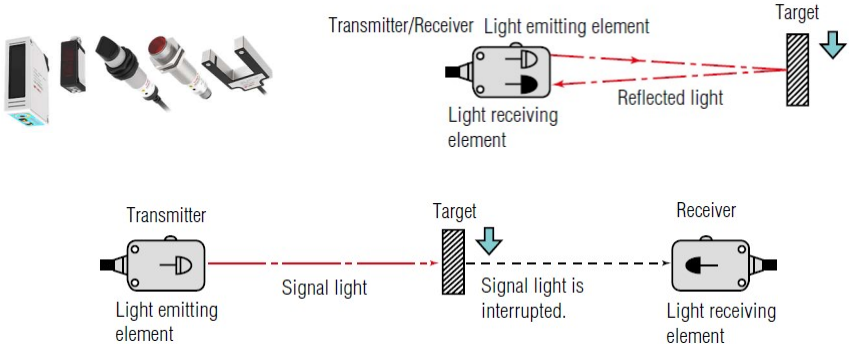
- **Second Classification:** Based on the physical property to be monitored such as Temperature, Flow, Pressure, Sound, Light etc.

| Type | Examples |
|---|---|
| Temperature | Thermistors, thermocouples |
| Pressure | Pressure gauges, barometers, ionization gauges |
| Optical | Photodiodes, phototransistors, infrared sensors, CCD sensors |
| Acoustic | Piezoelectric resonators, microphones |
| Mechanical | Strain gauges, tactile sensors, capacitive diaphragms, piezoresistive cells |
| Motion, vibration | Accelerometers, mass air flow sensors |
| Position | GPS, ultrasound-based sensors, infrared-based sensors, inclinometers |
| Electromagnetic | Hall-effect sensors, magnetometers |
| Chemical | pH sensors, electrochemical sensors, infrared gas sensors |
| Humidity | Capacitive and resistive sensors, hygrometers, MEMS-based humidity sensors |
| Radiation | Ionization detectors, Geiger-Mueller counters |

## Wireless Sensor Network

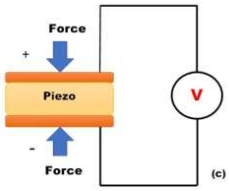
### Classification of Sensors

- **Third Classification:** The next classification is based on conversion phenomenon i.e. the input and the output. Some of the common conversion phenomena are Photoelectric, Thermoelectric, Electrochemical, Electromagnetic, Thermooptic, etc.
  - A **photoelectric sensor** emits a light beam (visible or infrared) from its light-emitting element. A reflective-type photoelectric sensor is used to detect the light beam reflected from the target. A thrubeam type sensor is used to measure the change in light quantity caused by the target crossing the optical axis.



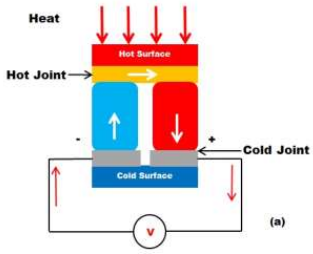
## Wireless Sensor Network

- **Piezoelectric effect:** Piezoelectric effect is a property of certain material to generate electrical energy by applying mechanical stress. It is an irreversible type of effect means inverse piezoelectric effect can deform the material when an electrical field is applied on it.
- The stress and strain may be applied from the different source like motion, noise, air flow and vibration. When mechanical stress is applied on a piezoelectric material, there is a shifting of positive charge and negative charge internally. It results an electrical field outside the material.
- There are two types of piezoelectric materials.
  1. Some Natural materials are Quartz, Topaz, Tourmaline, Cane Sugar, berlinite, Bone and Rochelle salt. Examples of artificial piezoelectric materials are lithium niobate, barium titanate and lead zirconate titanate etc.
  2. The electrical charge of piezoelectric material depends upon the various parameters and relationship.



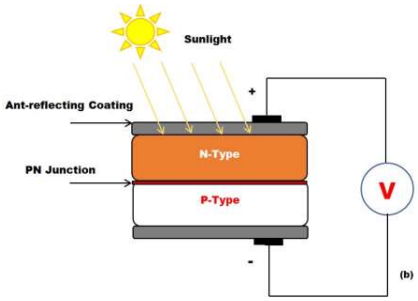

## Wireless Sensor Network

- **Thermoelectric Effect/Method:** It is an irreversible property of Thermoelectric Materials for converting Thermal Energy into Electrical Energy and vice-versa. These devices basically have three thermoelectric effects (Peltier, Seebeck, Thomson).
- Thermoelectric energy is produced by the Seebeck Effect due to the voltage generated by the difference of temperature at the joint of two different conductors.
- The thermoelectric energy harvester consists of TEG (Thermoelectric Generator) consists of a group of thermocouples connected in series to the main source of heat



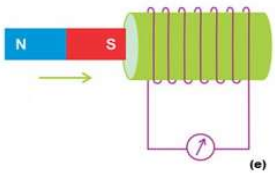
## Wireless Sensor Network

- **Photovoltaic Effect/Method:** The Solar/Light energy is converted into electric energy using the Photovoltaic Cell termed as Photovoltaic Effect. When a PV Cell is exposed by Sunlight/Light, it releases electrons and generate the electric current internally without any external device or source. The intensity of Energy depends upon the size of cell, efficiency and the light intensity .




## Wireless Sensor Network

- **Electromagnetic Effect/Method:** Electromagnetic energy is harvested by electromagnetic induction effect. When a wire/coil moves inside the magnetic field, electric current is induced in the wire.
- Electricity can be generated by rotating a magnet inside a coil of wire. This induces a current in the wire. As the magnetic field is changing direction repeatedly (as it turns), the direction of the induced current also changes. Therefore, alternating Current (AC) is produced. This type of generator is called an alternator. Energy can also be produced by electromagnetic effect due to vibration and fluid flow source of energy.



## Wireless Sensor Network

### Resistive Sensors

- The most commonly used type of transducer is variable resistance transducer. It is otherwise called as resistive sensors. It measures temperature, pressure, displacement, force, vibrations, etc. to understand the working principle, consider a conductor rod.
- Resistive sensors works on the principle that, the conductor length is directly proportional to resistance of the conductor and it is inversely related with area of the conductor. Therefore, L is denominated for conductor length, A for area of the conductor and R for resistance of conductor. ρ is the resistivity and it is constant for all the materials used for conductor construction.

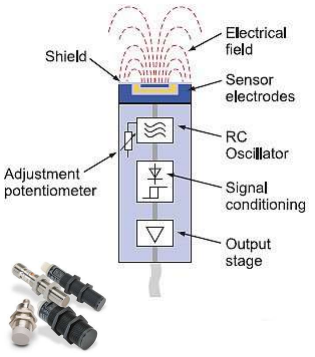
$$R = \frac{\rho L}{A}$$

- The resistance of the transducer varies due to external environmental factors and physical properties of the conductor. AC or DC devices are used to measure the resistance change. This transducer acts as both primary and secondary transducer. As a primary transducer, it converts physical quantity into mechanical signal. As a secondary transducer, the obtained mechanical signal is converted into electrical signal.
- There are different sensors available in the market which consider different factors in detecting resistance. Examples of some basic resistive sensors:
  Thermistors, Flex sensors, Force sensing resistors, Light dependent resistors (photoresistors)

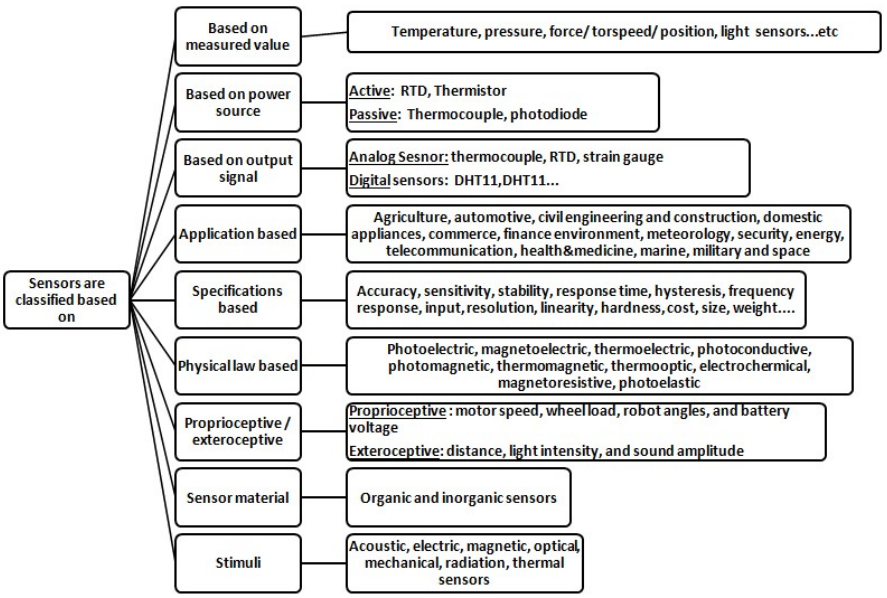
## Wireless Sensor Network

### Capacitive Sensors

- It is used to measure motion, proximity, acceleration, pressure, electric fields, chemical composition and liquid depth.
- A typical capacitor consists of two conductive elements (sometimes called plates) separated by some kind of insulating material that can be one of many different types including ceramic, plastic, paper, or other materials. The capacitance is determined as:
  C = εA/d, Where A is the Plate Area and d is the distance between two plates.
- Capacitive Proximity Sensors detect changes in the capacitance between the sensing object and the Sensor. As per the name, capacitive proximity sensors operate by noting a change in the capacitance read by the sensor.
- The amount of capacitance varies depending on the size and distance of the sensing object. An ordinary Capacitive Proximity Sensor is similar to a capacitor with two parallel plates, where the capacity of the two plates detected.
- One of the plates is the object being measured (with an imaginary ground), and the other is the Sensor's sensing surface. It detects the changes in the capacity generated between these two poles. The detection of the object depends on their dielectric constant, but they include resin and water in addition to metals.



## Wireless Sensor Network




## Wireless Sensor Network: History

- DARPA:
  - Distributed Sensor Nets Workshop (1978)
  - Distributed Sensor Networks (DSN) program (early 1980s)
  - Sensor Information Technology (SensIT) program
- UCLA and Rockwell Science Center
  - Wireless Integrated Network Sensors (WINS)
  - Low Power Wireless Integrated Microsensor (LWIM) (1996)
- UC-Berkeley
  - Smart Dust project (1999)
  - concept of "motes": extremely small sensor nodes
- Berkeley Wireless Research Center (BWRC)
  - PicoRadio project (2000)
- MIT
  - µAMPS (micro-Adaptive Multidomain Power-aware Sensors) (2005)

## WSN Standards

The 802.11 standard is defined through several specifications of WLANs. It defines an over-the-air interface between a wireless client and a base station or between two wireless clients.

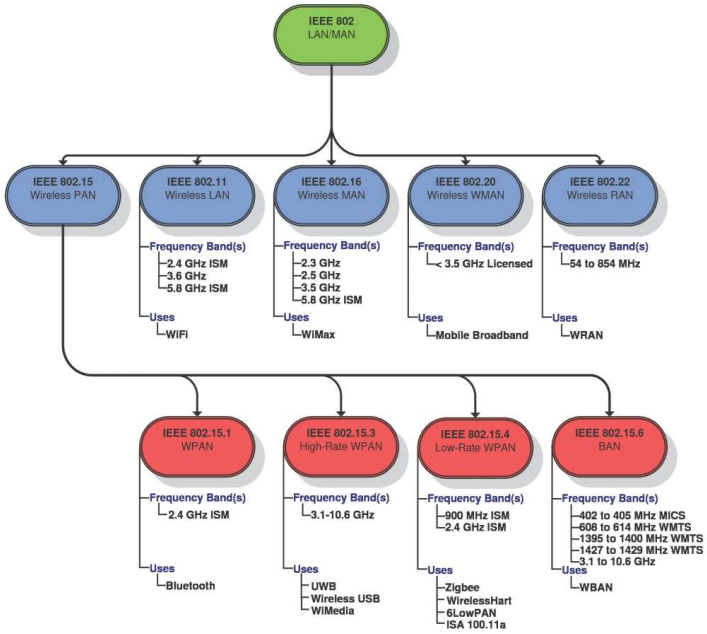
IEEE 802.11 Standards

| 802.11 protocol | Release | Freq. (GHz) | Band-width (MHz) | Data rate per stream (Mbit/s) | Allowable MIMO streams | Approximate indoor range (m) | Approximate indoor range (ft) | Approximate outdoor range (m) | Approximate outdoor range (ft) |
|---|---|---|---|---|---|---|---|---|---|
| - | Jun 1997 | 2.4 | 22 | 1, 2 | N/A | 20 | 66 | 100 | 330 |
| a | Sep 1999 | 5 | 20 | 6, 9, 12, 18, 24, 36, 48, 54 | N/A | 35 | 115 | 120 | 390 |
| | | 3.7 | | | | — | — | 5,000 | 16,000 |
| b | Sep 1999 | 2.4 | 22 | 1, 2, 5.5, 11 | N/A | 35 | 115 | 140 | 460 |
| g | Jun 2003 | 2.4 | 20 | 6, 9, 12, 18, 24, 36, 48, 54 | N/A | 38 | 125 | 140 | 460 |
| n | Oct 2009 | 2.4/5 | 20 | 7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2 | 4 | 70 | 230 | 250 | 820 |
| | | | 40 | 15, 30, 45, 60, 90, 120, 135, 150 | | 70 | 230 | 250 | 820 |
| ac | Dec 2013 | 5 | 20 | up to 87.6 | 8 | | | | |
| | | | 40 | up to 200 | | | | | |
| | | | 80 | up to 433.3 | | | | | |
| | | | 160 | up to 866.7 | | | | | |

IEEE802.11 was frequently used in early WSN and can still be used in current networks where the bandwidth demands are high (e.g. for Multimedia Sensors).

However, the high-energy overheads of IEEE802.11 is unsuitable for the low-power sensor networks. So, the variety of protocol have been developed for the need of low-power consumption and low data rates. e.g. 802.15.4 protocol has been designed specifically for short-range communication and low-power sensor networks.

## WSN Standards



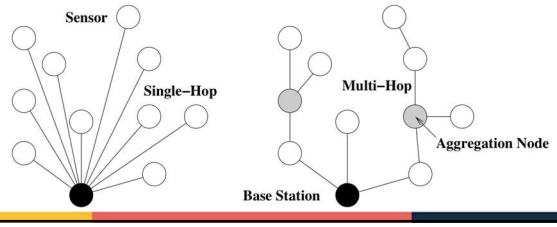
## Wireless Sensor Network Communication

**Star Topology (Single-hop Communication)**

- Every sensor directly communicate to the base station (single hop).
- May require large transmit power
- May be infeasible for large geographical area

**Mesh Topology (Multi-hop Communication)**

- Sensor serves as relay (forwarder) to the others sensor nodes.
- May reduce power consumption and allows for larger coverage area.
- Introduces the problems of routing (the task of finding a multi-hop path from a sensor node to the base station).
- Elimination of redundant data or aggregation of data that may be smaller than original data.




## Challenges in WSN: Energy

**Energy**

- Sensors typically powered through batteries
  - replace battery when depleted
  - recharge battery, e.g., using solar power
  - discard sensor node when battery depleted
- For batteries that cannot be recharged, sensor node should be able to operate during its entire **mission time** or until battery can be replaced
- Energy efficiency is affected by various aspects of sensor node/network design.
- Physical layer:
  - switching and leakage energy of CMOS-based processors.

$$E_{CPU} = E_{switch} + E_{leakage} = C_{total} * V_{dd}^{\ 2} + V_{dd} * I_{leak} * \Delta t$$

Where

| | |
|---|---|
| **Ctotal** | is the total capacitance switched by the computation |
| **Vdd** | is supply voltage |
| **Ileak** | is the leakage current |
| **Δt** | is the duration of computation |

## Challenges in WSN: Energy

**Energy**

- Medium access control layer:
  - contention-based strategies lead to energy-costly collisions
  - problem of idle listening
- Network layer:
  - responsible for finding energy-efficient routes
- Operating system:
  - small memory footprint and efficient task switching
- Security:
  - fast and simple algorithms for encryption, authentication, etc.
- Middleware:
  - in-network processing of sensor data can eliminate redundant data or aggregate sensor readings

## Challenges in WSN: Self Management

**Self Management**

- Ad-hoc deployment
  - ➢ many sensor networks are deployed "without design"
    - sensors dropped from airplanes (battlefield assessment)
    - sensors placed wherever currently needed (tracking patients in disaster zone)
    - moving sensors (robot teams exploring unknown terrain)
  - ➢ sensor node must have some or all of the following abilities
    - determine its location
    - determine identity of neighboring nodes
    - configure node parameters
    - discover route(s) to base station
    - initiate sensing responsibility
- Unattended operation
  - once deployed, WSN must operate without human intervention
  - device adapts to changes in topology, density, and traffic load
  - device adapts in response to failures

## Challenges in WSN: Self Management

- A Self-managing device
  - Monitor its surroundings,
  - Adapt to changes in the environment,
  - Cooperate with neighboring devices to form topologies or agree on sensing, processing, and communication strategies.

- Various Forms of Self-management
  - **self-organization** is the ability to adapt configuration parameters based on system and environmental state. (e.g. increase of transmission power in order to connect with the neighbor node).
  - **self-optimization** is the ability to monitor and optimize the use of the limited system resources.
  - **self-protection** is the ability recognize and protect from intrusions and attacks.
  - **self-healing** is the ability to discover, identify, and react to network disruptions.

## Challenges in WSN: Wireless Networking

Wireless communication faces a variety of challenges

- Attenuation:
  - limits radio range, i.e. a RF signal fades (i.e. decreases its power) while it propagates through a medium and it passes through obstacles. It can be represented by **Inverse-Square Law**

$$P_r \propto \frac{P_t}{d^2}$$

Where,

| | |
|---|---|
| Received Power | : Pr |
| Transmitted Power | : Pt |
| Distance from the source of signal | : d |

  - As a Consequences, an increase in distance require more transmission power. Therefore, distance can be divided into shorter distance (i.e. Multi-hop communication can be adopted).

## Challenges in WSN: Wireless Networking

- Multi-hop communication:
  - Nodes in a network cooperate with each other to identify efficient routes and serves as relay.
  - increased latency.
  - increased failure/error probability.
  - complicated by use of **duty cycles**,
    - ✓ Power Conservation Policies are used
    - ✓ Radios are switched off when they are not in use.
    - ✓ Neither Sensor Nodes receive signals from its neighbors nor serve as relay for other nodes.

    1. Wake-up and demand Duty Cycling.
       - ❑ Node can wakeup whenever needed
       - ❑ Devices use tow radios, a lower power radio is used to receive wakeup calls, a high power radio is activated in response of wakeup call.

    2. Adaptive Duty Cycling
       - ❑ Not all the nodes are allowed to sleep at the same time.
       - ❑ A subset of nodes remain active to form a network backbone.

## Challenges in WSN: Decentralization

- **Centralized** management (e.g., at the base station) of the network often not feasible to due large scale of network and energy constraints.

- Therefore, **decentralized** (or distributed) solutions often preferred, though they may perform worse than their centralized counterparts, but they may be more energy-efficient than centralized solutions.

- Example: routing
- Centralized:
  - BS collects information from all sensor nodes
  - BS establishes “optimal” routes (e.g., in terms of energy)
  - BS informs all sensor nodes of routes
  - can be expensive, especially when the topology changes frequently

- Decentralized:
  - each sensors makes routing decisions based on limited local information
  - routes may be nonoptimal, but route establishment/management can be much cheaper

## Challenges in WSN: Design Constraints

- Many hardware and software limitations affect the overall system design.
- Examples include:
  - Low processing speeds (to save energy)
  - Low storage capacities (to allow for small form factor and to save energy)
  - Lack of I/O components such as GPS receivers (reduce cost, size, energy)
  - Lack of software features such as multi-threading (reduce software complexity)

## Challenges in WSN: Security

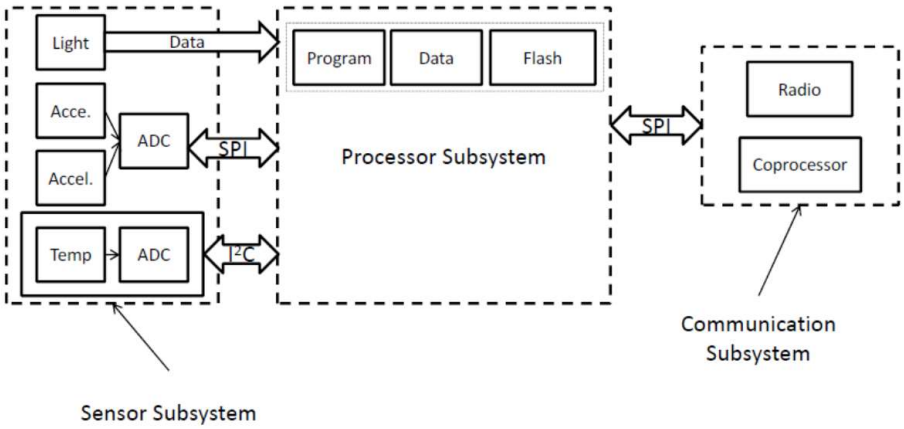
- Sensor networks often monitor critical infrastructure or carry sensitive information, making them desirable targets for attacks.
- Attacks may be facilitated by:
  - remote and unattended operation
  - wireless communication
  - lack of advanced security features due to cost, form factor, or energy
- Conventional security techniques often not feasible due to their computational, communication, and storage requirements.
- As a consequence, sensor networks require new solutions for intrusion detection, encryption, key establishment and distribution, node authentication, and secrecy.

## Comparison: Traditional Networks Vs Wireless Sensor Networks

| Traditional Networks | Wireless Sensor Networks |
|---|---|
| General-purpose design; serving many applications | Single-purpose design; serving one specific application |
| Typical primary design concerns are network performance and latencies; energy is not a primary concern | Energy is the main constraint in the design of all node and network components |
| Networks are designed and engineered according to plans | Deployment, network structure, and resource use are often ad-hoc (without planning) |
| Devices and networks operate in controlled and mild environments | Sensor networks often operate in environments with harsh conditions |
| Maintenance and repair are common and networks are typically easy to access | Physical access to sensor nodes is often difficult or even impossible |
| Component failure is addressed through maintenance and repair | Component failure is expected and addressed in the design of the network |
| Obtaining global network knowledge is typically feasible and centralized management is possible | Most decisions are made localized without the support of a central manager |

## Node Architecture

- Wireless sensor nodes are the essential building blocks in a wireless sensor network
  - Sensing, Processing, and Communication
  - Stores and executes the communication protocols as well as data processing algorithms
- The node consists of sensing, processing, communication, and power subsystems.

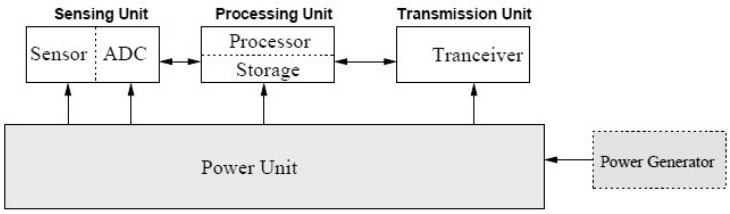


## Node Architecture

- Sensing units are usually composed of two subunits:
  - Sensors and
  - Analogue to Digital Converters (ADCs).
    The analogue signals produced by the sensors are converted to digital signals by the ADC, and then fed into the processing unit.
- The processing unit is generally associated with a small storage unit and it can manage the procedures that make the sensor node collaborate with the other nodes to carry out the assigned sensing tasks. The processor subsystem is the central element of the node and the choice of a processor determines the tradeoff between flexibility and efficiency – in terms of both energy and performance. There are several processors as options:
  - Microcontroller
  - Digital Signal Processor (DSP)
  - Application-specific Integrated Circuit (ASIC)
  - Field Programmable Gate Array (FPGA)
- A transceiver unit connects the node to the network.
  - Serial Peripheral Interface (SPI)
  - Inter-Integrated Circuit ($I^2C$)

## Node Architecture

- The power subsystem provides DC power to all the other subsystems to bias their active components such as crystal oscillators, amplifiers, registers, and counters. Moreover, it provides DC–DC converters so that each subsystem can obtain the right amount of bias voltage.



## Node Architecture: The Sensing Subsystem

- The Sensing Subsystem integrates the sensors as per the requirement of the application. The following table shows the details of sensors used in applications.

| Sensor | Application Area | Sensed Event | Explanation |
|---|---|---|---|
| Accelerometer | AVM | 2D and 3D acceleration of movements of people and objects | Volcano activities |
| | SHM | | Stiffness of a structure |
| | Health care | | Stiffness of bones, limbs, joints; Motor fluctuation in Parkinson's disease |
| | Transportation | | Irregularities in rail, axle box or wheels of a train system |
| | SCM | | Defect of fragile objects during transportation |
| Acoustic emission sensor | SHM | Elastic waves generated by the energy released during crack propagation | Measures micro-structural changes or displacements |
| Acoustic sensor | Transportation & Pipelines | Acoustic pressure vibration | Vehicle detection; Measure structural irregularities; Gas contamination |
| Capacitance sensor | PA | Solute concentration | Measure the water content of a soil |

## Node Architecture: The Sensing Subsystem

| Sensor | Application Area | Sensed Event | Explanation |
|---|---|---|---|
| ECG | Health care | Heart rate | |
| EEG | | Brain electrical activity | |
| EMG | | Muscle activity | |
| Electrical sensors | PA | Electrical capacitance or inductance affected by the composition of tested soil | Measure of nutrient contents and distribution |
| Gyroscope | Health care | Angular velocity | Detection of gait phases |
| Humidity sensor | PA & HM | Relative and absolute humidity | |
| Infrasonic sensor | AVM | Concussive acoustic waves – earth quake or volcanic eruption | |
| Magnetic sensor | Transportation | Presence, intensity, direction, rotation and variation of a magnetic field | Presence, speed and density of a vehicle on a street; congestion |
| Oximeter | Health care | Blood oxygenation of patient's hemoglobin | Cardiovascular exertion and trending of exertion relative to activity |
| pH sensor | Pipeline (water) | Concentration of hydrogen ions | Indicates the acid and alkaline content of a water measure of cleanliness |

## Node Architecture: The Sensing Subsystem

| Sensor | Application Area | Sensed Event | Explanation |
|---|---|---|---|
| Photo acoustic spectroscopy | Pipeline | Gas sensing | Detects gas leak in a pipeline |
| Piezoelectric cylinder | Pipeline | Gas velocity | A leak produces a high frequency noise that produces a high frequency noise that produces vibration |
| Soil moisture sensor | PA | Soil moisture | Fertilizer and water management |
| Temperature sensor | PA & HM | Pressure exerted on a fluid | |
| Passive infrared sensor | Health care & HM | Infrared radiation from objects | Motion detection |
| Seismic sensor | AVM | Measure primary and secondary seismic waves (Body wave, ambient vibration) | Detection of earth quake |
| Oxygen sensor | Health care | Amount and proportion of oxygen in the blood | |
| Blood flow sensor | Health care | The Doppler shift of a reflected ultrasonic wave in the blood | |

## Node Architecture: The Sensing Subsystem

ADC converts the output of a sensor - which is a continuous, analog signal - into a digital signal. It requires two steps:

1. The analog signal has to be quantized
   - allowable discrete values is influenced :
     a) by the frequency and magnitude of the signal
     b) by the available processing and storage resources

2. The sampling frequency
   - Nyquist rate does not suffice because of noise and transmission error
   - Resolution of ADC - an expression of the number of bits that can be used to encode the digital output

$$Q = \frac{E_{pp}}{2^M}$$

   - where Q is the resolution in volts per step (volts per output code); *Epp* is the peak-to-peak analog voltage; *M* is the ADC's resolution in bits

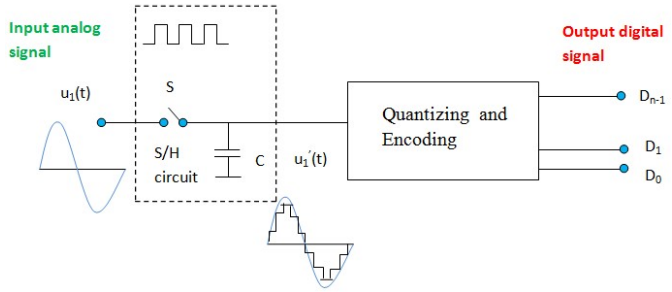
## Node Architecture: The Sensing Subsystem

There are mainly two steps involves in the process of conversion. They are

- Sampling and Holding
- Quantizing and Encoding

**Sampling and Holding**

In the process of Sample and hold (S/H), the continuous signal will gets sampled and freeze (hold) the value at a steady level for a particular least period of time. It is done to remove variations in input signal which can alter the conversion process and thereby increases the accuracy. The minimum sampling rate has to be two times the maximum data frequency of the input signal.




## Node Architecture: The Sensing Subsystem

**Quantizing and Encoding**

For understanding quantizing, we can first go through the term Resolution used in ADC. It is the smallest variation in analog signal that will result in a variation in the digital output. This actually represents the quantization error.

$$Resolution, \triangle V = \frac{V_r}{2^N}$$

V → Reference voltage range
$2^N$ → Number of states
N → Number of bits in digital output

**Quantizing:** It is the process in which the reference signal is partitioned into several discrete quanta and then the input signal is matched with the correct quantum.

**Encoding:** Here; for each quantum, a unique digital code will be assigned and after that the input signal is allocated with this digital code. The process of quantizing and encoding is demonstrated in the table below.

| Analog signal | | | Digital o/p |
|---|---|---|---|
| 7.5 | 7 | 7Δ=7V | 111 |
| 6.5 | 6 | 6Δ=6V | 110 |
| 5.5 | 5 | 5Δ=5V | 101 |
| 4.5 | 4 | 4Δ=4V | 100 |
| 3.5 | 3 | 3Δ=3V | 011 |
| 2.5 | 2 | 2Δ=2V | 010 |
| 1.5 | 1 | 1Δ=1V | 001 |
| 0.5 | 0 | 0Δ=0V | 000 |

$$\pm \frac{1}{2}\triangle V = \pm 0.5V$$

From the above table we can observe that only one digital value is used to represent the whole range of voltage in an interval. Thus, an error will occur and it is called quantization error. This is the noise introduced by the process of quantization. Here the maximum quantization error is

## Node Architecture: The Processor Subsystem

- The processor subsystem brings together
  - all the other subsystems and some additional peripherals.
  - Its main purpose is to process (execute) instructions pertaining to sensing, communication, and self-organization.
- It consists of
  - a processor chip,
  - a nonvolatile memory (usually an internal flash memory) for storing program instructions, an active memory for temporarily storing the sensed data, and
  - an internal clock, among other things.
- A wide range of off-the-shelf processors are available for building a wireless sensor node, one has to make a careful choice, as it affects the cost, flexibility, performance, and energy consumption of the node.
  - Microcontroller
  - Digital Signal Processor (DSP)
  - Application-specific Integrated Circuit (ASIC)
  - Field Programmable Gate Array (FPGA)

## Node Architecture: The Processor Subsystem

- If the sensing task is well defined from the outset and does not change over time, a designer may choose either a **field programmable gate array or a digital signal processor**. These processors are very efficient in terms of their energy consumption; and for most simple sensing tasks, they are quite adequate.
- The sensing goal changes or a modification may be required. Moreover, the software that runs on the wireless sensor node may require occasional updates or remote debugging. Such tasks require a considerable amount of computation and processing space at runtime. In which case, **special-purpose, energy-efficient processors** are not suitable.
- WSNs are emerging technologies; and the research community is still active with research for developing energy-efficient communication protocols and signal-processing algorithms. As this requires dynamic code installation and update, the **microcontroller** is the best option.

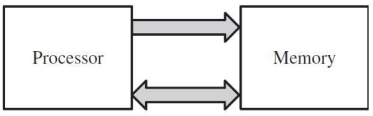
## Node Architecture: The Processor Subsystem

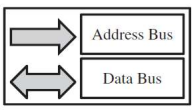
- A major concern in resource-constrained processors is the efficient execution of algorithms, since this requires the transferring of information from and to memory.
- This includes program instructions and the data to be processed or manipulated. For example, in WSNs, the data stem from the physical sensors and the program instructions relate to communication, self-organization, data compression, and aggregation algorithms.
- The processor subsystem can be designed by employing one of the three basic computer architectures!
  - *Von Neumann architecture*
  - *Harvard architecture*
  - *Super-Harvard (SHARC) architecture*

## Node Architecture: The Processor Subsystem

**Von Neumann architecture**

- A single memory space that is used by program instructions and data.
- A single bus to transfer data between the processor and the memory.
- A relatively slow processing speed because each data transfer requires a separate clock.

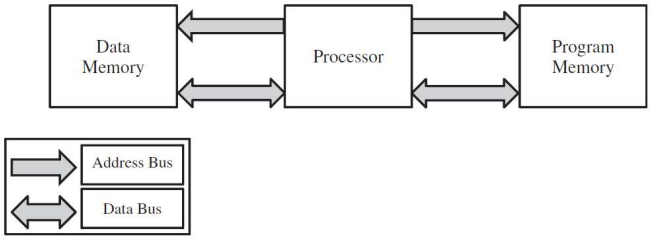


## Node Architecture: The Processor Subsystem

**Harvard architecture**

- provides *separate memory spaces* - storing program instructions and data
- each memory space is interfaced with the processor with a separate data bus
- program instructions and data can be accessed *at the same time*
- a special *single instruction, multiple data (SIMD)* operation, a special arithmetic operation and a bit reverse
- supports multi-tasking operating systems; but does not provide virtual memory protection

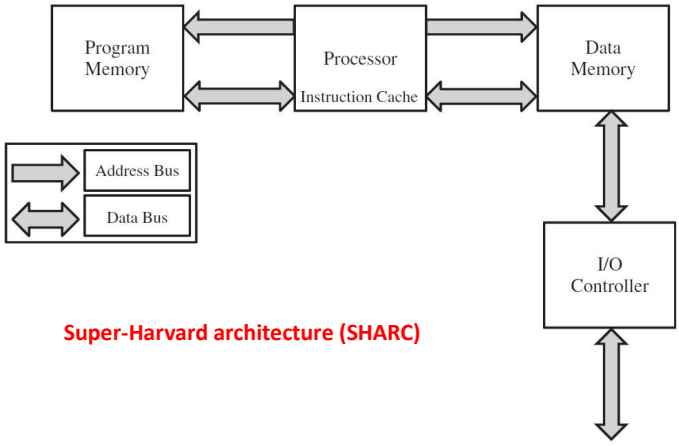



## Node Architecture: The Processor Subsystem

Super-Harvard architecture (SHARC)

- An extension of the Harvard architecture
- *Adds* two components to the Harvard architecture
  - Internal instruction cache - temporarily store frequently used instructions - enhances performance
  - An underutilized program memory can be used as a temporary relocation place for data
  - Why Direct Memory Access (DMA) ?
    - costly CPU cycles can be invested in a different task
    - program memory bus and data memory bus accessible from outside the chip

## Node Architecture: The Processor Subsystem




**Super-Harvard architecture (SHARC)**

## Node Architecture: The Processor Subsystem

**Microcontroller**

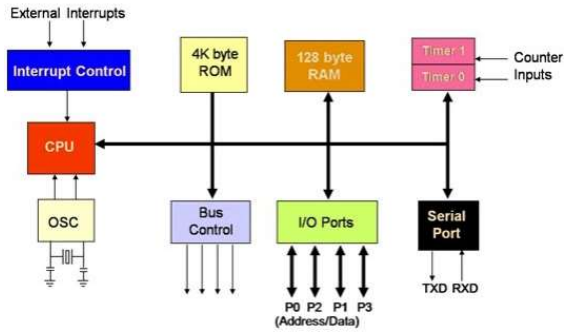
- It is a computer on a single integrated circuit, consisting of a comparatively simple central processing unit and additional components such as high-speed uses, a memory unit, a watchdog timer, and an external clock.
- Microcontrollers are integrated in many products and embedded devices. Today such simple systems as elevators, ventilators, office machines, household appliances, power tools, and toys ubiquitously employ microcontrollers.

***Structure of microcontroller***

integrates the following components:

- CPU core
- Volatile memory (RAM) for data storage
- ROM, EPROM, EEPROM, or Flash memory
- Parallel I/O interfaces
- Discrete input and output bits
- Clock generator
- One or more internal analog-to-digital converters
- Serial communications interfaces

## Node Architecture: The Processor Subsystem




***Advantages:***

- suitable for building computationally less intensive, standalone applications, because of its *compact construction*, *small size*, *low-power consumption*, and *low cost*
- *high speed* of the programming and eases debugging, because of the use of higher-level programming languages

***Disadvantages:***

- *not* as *powerful* and as *efficient* as some custom-made processors (such as DSPs and FPGAs)
- some applications (simple sensing tasks but large scale deployments) may prefer to use architecturally simple but energy- and cost-efficient processors

## Node Architecture: The Processor Subsystem

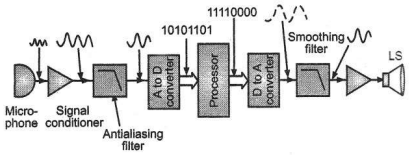
***Digital Signal Processor***

The main function:

- process discrete signals with digital filters
- filters minimize the effect of noise on a signal or enhance or modify the spectral characteristics of a signal
- while analog signal processing requires complex hardware components, digital signal processors (DSP) requires simple adders, multipliers, and delay circuits
- DSPs are highly efficient
- most DSPs are designed with the Harvard Architecture

***Advantages:***

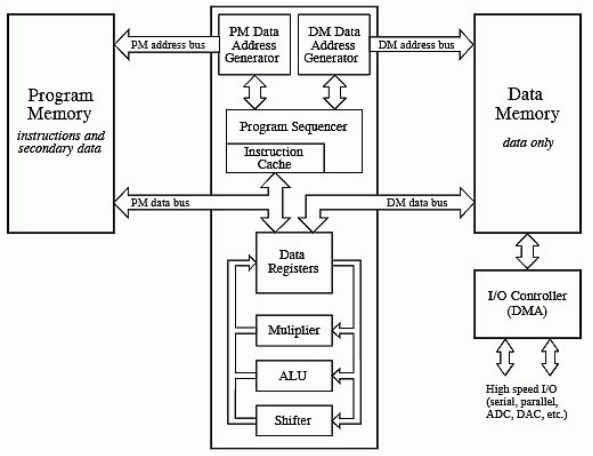
- *powerful* and *complex* digital filters can be realized with *commonplace* DSPs
- *useful for applications* that require the deployment of nodes in harsh physical settings (where the signal transmission suffers corruption due to noise and interference and, hence, requires aggressive signal processing)



## Node Architecture: The Processor Subsystem

***Disadvantage:***

- some tasks require *protocols* (and not numerical operations) that *require* periodical *upgrades* or *modifications* (i.e., the networks should support flexibility in network reprogramming)




## Node Architecture: The Processor Subsystem

***Application-specific Integrated Circuit***

- ASIC is an *IC* that can be customized for a specific application
- Two types of design approaches: *full-customized* and *half-customized*
- Full-customized IC:
  - Some logic cells, circuits, or layout are custom made in order to optimize cell performance
  - Includes features which are not defined by the standard cell library
  - Expensive and long design time
- Half-customized ASICs are built with logic cells that are available in the standard library
- In both cases, the final logic structure is configured by the end user - an ASIC is a *cost efficient solution, flexible,* and *reusable*

## Node Architecture: The Processor Subsystem

***Advantages:***

- Relatively *simple design*; can be optimized to *meet a specific customer demand*
- *Multiple* microprocessor *cores* and *embedded software* can be designed in a *single* cell

***Disadvantage:***

- *High* development *costs* and *lack of re-configurability*

***Application:***

- ASICs are not meant to replace microcontrollers or DSPs but to complement them
- Handle rudimentary and low-level tasks
- To decouple these tasks from the main processing subsystem

## Node Architecture: The Processor Subsystem

***Field Programmable Gate Array (FPGA)***

The distinction between ASICs and FPGAs is not always clear

- FPGAs are *more complex* in design and *more flexible* to program
- FPGAs are programmed electrically, by modifying a packaged part
- Programming is done with the support of circuit diagrams and hardware description languages, such as VHDL and Verilog

Typical features of a FPGA are summarized as follows:

- In a FPGA, none of the mask layers is customized;
- A FPGA includes some programmable logic components, or *logic blocks* – these are: a 4-input lookup table (LUT), a flip-flop, and an output block;
- There is a well-defined and formal method for programming the basic logic cells and the interconnect;
- There is a matrix of programmable interconnects surrounding the basic logic cells producing a configuration instance; and
- There are programmable I/O cells that surround the core.

## Node Architecture: The Processor Subsystem

***Advantages***

- *Higher bandwidth* compared to DSPs
- *Flexible* in their application
- Support *parallel processing*
- Work with *floating point representation*
- Greater *flexibility of control*

***Disadvantages***

- *Complex*
- The design and realization process is *costly*

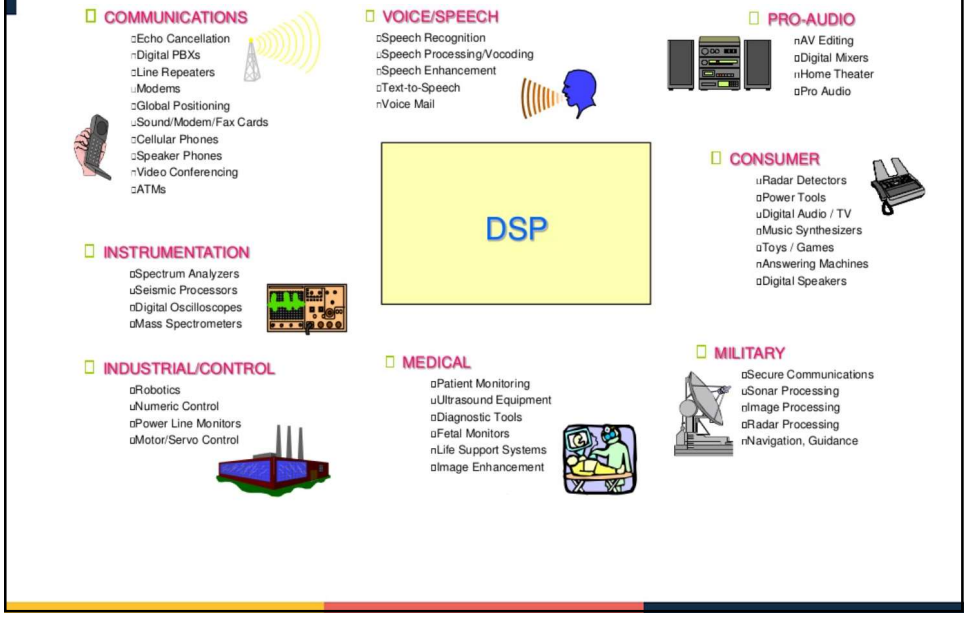
## Node Architecture: The Processor Subsystem

***Comparison ASIC, FGPA, Microcontroller & DSP***

| | ASIC | FGPA | UP/UC | DSP |
|---|---|---|---|---|
| **Flexibility** | None | Limited | High | High |
| **Design Time** | Long | Medium | Short | Short |
| **Power Consumption** | Low | Low-Medium | Medium-High | Low-Medium |
| **Performance** | High | High | Low-Medium | Medium-High |
| **Development Cost** | High | Medium | Low | Low |
| **Production Cost** | Low | Low-Medium | Medium-High | Low-Medium |

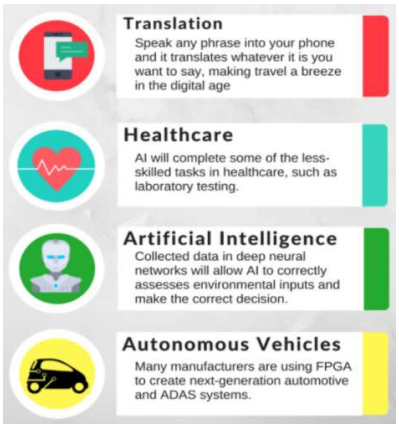
- Working with a ***micro-controller*** is *preferred* if the design goal is to achieve *flexibility*
- Working with the other mentioned options is preferred if power consumption and computational efficiency is desired.
- ***DSPs*** are expensive, large in size and less flexible; they are *best for signal processing*, with specific algorithms.
- ***FPGAs*** *are faster* than both microcontrollers and digital signal processors and support *parallel computing*; but their production cost and the programming difficulty make them *less suitable.*
- ***ASICs*** have *higher bandwidths*; they are *the smallest* in size, *perform much better*, and *consume less power* than any of the other processing types; but have a *high cost* of production owing to the complex design process.

## Node Architecture: Communication Interfaces




## Node Architecture: Communication Interfaces

*4 Popular FGPA Application Areas*



## Node Architecture: Communication Interfaces

*Microcontroller Application Areas*

***Point of Sale***

- P.O.S.
- Vending Machine

***Telecom***

- PABX
- Bridges

***Appliances***

- High-end appliance
- User interface

***Measurement***

- Testing equipment
- Medical equipment

***USB***

- USB key
- Smart card reader
- High end peripherals

***Automation***

- PLC
- User interface

## Node Architecture: Communication Interfaces

*ASIC Application Areas*


***Apple A6 Processor***

***Aerospace Engineering***

## Node Architecture: Communication Interfaces

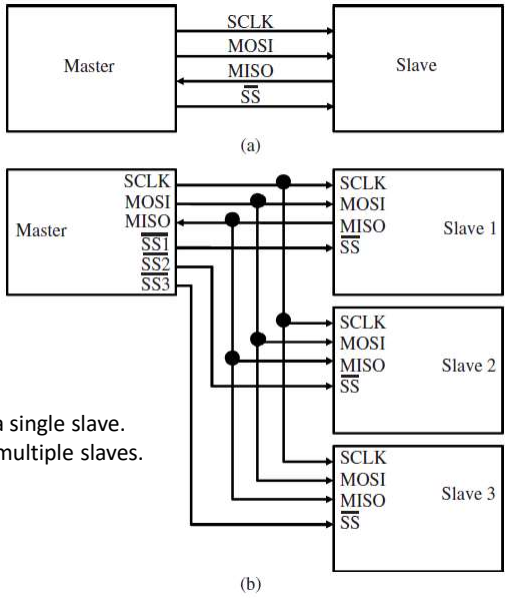
- *Fast* and *energy efficient data transfer* between the subsystems of a wireless sensor node *is vital.*
  - However, the practical size of the node puts restriction on system buses
  - Communication via a parallel bus is *faster* than a serial transmission
  - A parallel bus needs *more space*
- Therefore, considering *the size of the node*, *parallel buses* are never supported.
- The choice is often between *serial interfaces* :
  - Serial Peripheral Interface (SPI)
  - General Purpose Input/Output (GPIO)
  - Secure Data Input/Output (SDIO)
  - Inter-Integrated Circuit (I2C)
- Among these, the most commonly used buses are *SPI* and *I2C.*

## Node Architecture: Communication Interfaces

***Serial Peripheral Interface (SPI)***

- SPI (Motorola, in the mid-80s)!
  - *high-speed*, *full-duplex synchronous* serial bus
  - does *not* have an official standard, but use of the SPI interface should conform to the implementation specification of others - correct communication
- The SPI bus defines *four pins*:
  - MOSI (MasterOut/SlaveIn)
    - Used to transmit data *from the master to the slave* when a device is configured as a master
  - MISO (MasterIn/SlaveOut)
  - SCLK (Serial Clock)
    - Used by the *master* to send the clock signal that is needed to *synchronize transmission*
    - Used by the *slave* to read this signal synchronize transmission
  - CS (Chip Select) - communicate via the CS port

## Node Architecture: Communication Interfaces




Connecting devices with the SPI bus.
(a) A single master connecting with a single slave.
(b) A single master connecting with multiple slaves.

## Node Architecture: Communication Interfaces

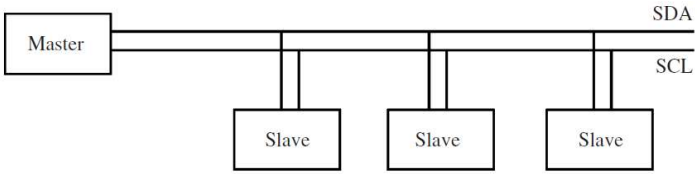
- Both master and slave devices hold a shift register
- Every device in every transmission must read and send data
- SPI supports a *synchronous communication protocol*
  - The master and the slave must agree on the timing
  - Master and slave should agree on two additional parameters
    - clock polarity (*CPOL*) - defines whether a clock is used as high- or low-active
    - clock phase (*CPHA*) - determines the times when the data in the registers is allowed to change and when the written data can be read

| SPI mode | CPOL | CPHA | Description |
|---|---|---|---|
| 0 | 0 | 0 | SCLK is low-active.<br>Sampling is allowed on odd clock edges.<br>Data changes on even clock edges. |
| 1 | 0 | 1 | SCLK is low-active.<br>Sampling is allowed on even clock edges.<br>Data changes on odd clock edges. |
| 2 | 1 | 0 | SCLK is high-active.<br>Sampling is allowed on odd clock edges.<br>Data changes on even clock edges. |
| 3 | 1 | 1 | SCLK is high-active.<br>Sampling is allowed on even clock edges.<br>Data changes on odd clock edges. |

## Node Architecture: Communication Interfaces

### Inter-Integrated Circuit

- Every device type that uses I2C must have a unique address that will be used to communicate with a device
- In earlier versions, a *7 bit address* was used, allowing 112 devices to be uniquely addressed - due to an increasing number of devices, it *is insufficient*
- Currently I2C uses *10 bit addressing*
- I2C is a *multi-master half-duplex synchronous serial* bus
  - only two bidirectional lines: (unlike SPI, which uses four)
    1. Serial Clock (SCL)
    2. Serial Data (SDA)




Connecting devices with the I2C serial bus

## Node Architecture: Communication Interfaces

### Inter-Integrated Circuit

- Since each master generates its own clock signal, communicating devices must *synchronize their clock speeds*
  - a slower slave device could wrongly detect its address on the SDA line while a faster master device is sending data to a third device!
- I2C requires arbitration between master devices *wanting* to send or receive data at the same time
  - *no* fair arbitration *algorithm*
  - rather the master that holds the SDA line low for *the longest time wins* the medium

## Node Architecture: Communication Interfaces

**Inter-Integrated Circuit**

- enables a device to read data *at a byte level* for a fast communication
  - the device can hold the SCL low until it completes reading or sending the next byte - called *handshaking*
- The *aim* of I2C is *to minimize costs* for connecting devices
  - accommodating lower transmission speeds
- I2C defines two speed modes:
  - *a fast-mode* - a bit rate of up to 400Kbps
  - *high-speed mode* - a transmission rate of up to 3.4 Mbps
  - they are downwards compatible to ensure communication with older components

## Node Architecture: Communication Interfaces

| SPI | $I^2C$ |
|---|---|
| • Four lines enable full-duplex transmission. | • Two lines reduce space and simplify circuit layout. Lowers costs. |
| • No addressing is required due to CS. This reduces overhead and increases throughput. However, one needs additional hardware configurations to connect more than one slave. | • Addressing enables multimaster mode, which in turn enables more than one device to initiate communication. |
| • Allowing only one master avoids conflicts. | • Multimaster mode is prone to conflicts when two or more master devices communicate simultaneously. Arbitration is required. |
| • Hardware requirement support increases with an increasing number of connected devices and therefore raises costs. | • Hardware requirement is independent of the number of devices using the bus. |
| • The master's clock is configured according to the slave's speed. This frees the slaves from requiring clocking a device. However, speed adaptation slows down the master. | • Slower devices may stretch the clock thereby increasing latency and keeping other devices waiting for accessing the bus. |
| • Speed depends on the maximum speed of the slowest device. | • Speed is limited to 3.4 MHz and all devices need to support the highest speed that is used in the system, otherwise a slower device may wrongly detect its device address. |
| • Heterogeneous register size allows flexibility in the devices that are supported. | • Homogeneous register size reduces overhead, since no additional control bits are required to be transmitted. |
| • Combined registers imply every transmission should be read. | • Devices that do not read or provide data are not forced to provide potentially useless bytes. |
| • The absence of an official standard leads to application specific implementations. | • Official standard eases integration of devices since developers can rely on a certain implementation. |

## Operating System

- An operating system (OS) in a WSN is
  - A thin software layer that
  - Logically resides between the node's hardware and the application.
  - Provides basic programming abstractions to application developers.
- Its main task is to
  - Enable applications to interact with hardware resources, to
  - Schedule and prioritize tasks, and
  - Arbitrate between contending applications and services that try to seize resources.

## Operating System

- Additional features include:
  - Memory Management;
  - Power Management;
  - File Management;
  - Networking;
  - A set of programming environments and tools – commands, command interpreters, command editors, compiler, debuggers, etc. – to enable users to develop, debug, and execute their own programs; and
  - Legal entry points into the operating system for accessing sensitive resources such as writing to input components.

## Operating System

Operating systems are classified as:

- Single-task/Multitasking and

| Single-task OS | Multitasking OS |
|---|---|
| A single-task operating system processes one task at a time. | a multitasking operating system can execute multiple tasks simultaneously. |
| It requires a large amount of memory to manage the states of multiple tasks but they enable tasks with different complexity to execute in parallel. | In a single-task OS, one task is executed at a time, therefore, as a rule tasks should have a short duration. |

- Single-user/Multiuser operating systems
  - In a single-user OS, one user (the owner of the resources) is active at a time, whereas a multi-user operating system allows multiple users to share the resources of a single system at the same time

## Operating System

The choice of a particular OS depends on several factors;

- ***Functional Aspects***
  - *Data Types*
  - *Scheduling*
  - *Stacks*
  - *System Calls*
  - *Handling Interrupts*
  - *Multithreading*
  - *Thread-based vs. Event-based Programming*
  - *Memory Allocation*
- ***Non-Functional Aspects***
  - Separation of Concern
  - System Overhead
  - Portability
  - Dynamic Reprogramming
- ***Prototypes***
  - TinyOS
  - SOS
  - Contiki
  - LiteOS

## Operating System- Functional Aspects

***Data Types***

- Different subsystems of WSN communicate with each other for various reasons such as to exchange data, delegate functionalities, and signaling.
- Interactions between the different subsystems take place through:
  - Well-formulated protocols
  - Data types
- Complex data types have strong expression power but consume resources - struct and enum.
- Simple data types are resource efficient but have limited expression capability - C programming language.

## Operating System- Functional Aspects

***Scheduling***

- Task scheduling is process to organize, prioritize and execute the tasks to determines the efficiency of the OS.
- Two scheduling mechanisms:

***1. Queuing-based Scheduling***

- In a queuing-based scheduling, tasks originating from the various subsystems are temporarily stored in a queue and executed serially according to a predefined rule.
- Some operating systems enable tasks to specify priority levels so that they can be given precedence.
- Types of Queuing-based Scheduling
  - ***FIFO (first-in-first-out) Scheme***
    - Tasks are processed according to their arrival time.
    - A task that arrives first will be executed first as soon as the processor is free.
    - It's non-preemptive scheduling, OS will execute the task to the end before another task is admitted for execution.
    - FIFO - the simplest and has minimum system overhead, but treats tasks unfairly; since tasks of long duration may block short-duration tasks for a long time.
    - Poor in performance as average wait time is high.
    - Easy to understand and implement.
    - Its implementation is based on FIFO queue.

## Operating System- Functional Aspects

### FCFS (Example)

| Process | Duration | Order | Arrival Time |
|---|---|---|---|
| P1 | 24 | 1 | 0 |
| P2 | 3 | 2 | 0 |
| P3 | 4 | 3 | 0 |

**Gantt Chart :**


P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

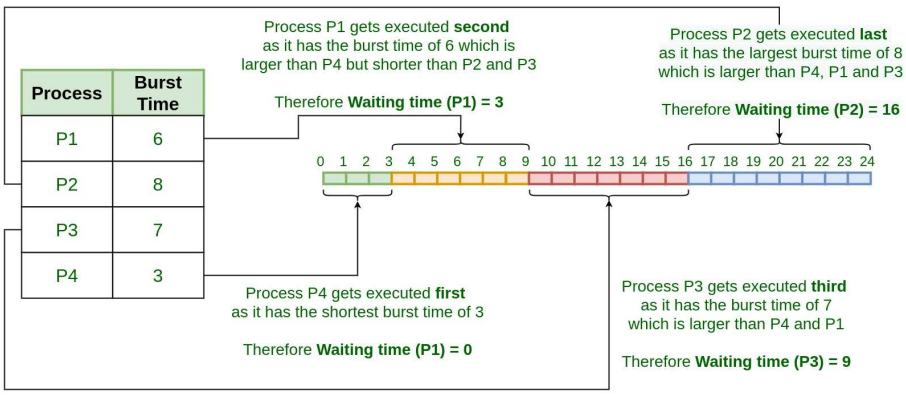
## Operating System- Functional Aspects

❑ ***Shortest Job First (SJF)***

- In a sorted queue scheme, tasks in a queue are sorted according to some criteria.
- One way is to sort tasks according to their estimated execution duration. This approach prevents long-duration tasks from blocking short-duration tasks. The approach is also known as the shortest job first (SJF) rule.
- Sorted queue - e.g., shortest job first (SJF), Sorting incurs system overhead, since each task in the queue must be evaluated to estimate execution duration and to sort tasks accordingly.
- This is the best approach to minimize waiting time.
- It is of two types:
  - Non Pre-emptive
  - Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)
- Easy to implement in Batch systems where required CPU time is known in advance.

## Operating System- Functional Aspects

***Shortest Job First (SJF) Example***




## Operating System- Functional Aspects

- ***Non-preemptive SJF***
  - ✓ Sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.
  - ✓ This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed.
  - ✓ This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.
- ***Pre-emptive Shortest Job First***
  - ✓ In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

## Operating System- Functional Aspects

❑ ***Round Robin Scheduling***

- It is mainly designed for time-sharing systems.
- This algorithm is similar to FCFS scheduling, but in Round Robin(RR) scheduling, preemption is added which enables the system to switch between processes.
- A fixed time is allotted to each process, called a **quantum**, for execution.
- Once a process is executed for the given time period that process is preempted and another process executes for the given time period.
- **Context switching** is used to save states of preempted processes.
- This algorithm is simple and easy to implement.
- This algorithm is **starvation-free** as all processes get a fair share of CPU.
- The length of time quantum is generally from 10 to 100 milliseconds in length.

## Operating System- Functional Aspects

❑ ***Round Robin Scheduling***

- It's Preemptive Algorithms.
- One of the oldest, easiest, and fairest algorithm.
- It is a real-time algorithm because it responds to the event within a specific time limit.
- In this algorithm, the time slice should be the minimum that is assigned to a specific task that needs to be processed. Though it may vary for different operating systems.
- It is a **hybrid model** and is clock-driven in nature.
- It is a widely used scheduling method in the traditional operating system.

## Operating System- Functional Aspects

***Round Robin Scheduling Example***

TIME SLICE = 4

| PROCESS | ARRIVAL TIME | BURST TIME | |
|---|---|---|---|
| | | TOTAL | REMAINING |
| P1 | 0 | 8 | 8 |
| P2 | 1 | 6 | 6 |
| P3 | 3 | 3 | 3 |
| P4 | 5 | 2 | 2 |
| P5 | 6 | 4 | 4 |

READY QUEUE:

| | | | | | | |
|---|---|---|---|---|---|---|
| P1 | P2 | P3 | P1 | P4 | P5 | P2 |

GANTT CHART:


| Process | Duration | Order | Arrival Time |
|---|---|---|---|
| P1 | 3 | 1 | 0 |
| P2 | 4 | 2 | 0 |
| P3 | 3 | 3 | 0 |

**Suppose time quantum is 1 unit.**

| P1 | P2 | P3 | P1 | P2 | P3 | P1 | P2 | P3 | P2 |
|---|---|---|---|---|---|---|---|---|---|

0 10

**P1 waiting time : 4**

**P2 waiting time: 6**

**P3 waiting time: 6**

**The average waiting time(AWT) : (4+6+6)/3=5.33**

## Operating System- Functional Aspects

Regardless of how tasks are executed, a scheduler can be either!

- A non-preemptive scheduler - a task is executed to the end, may not be interrupted by another task.
- Preemptive scheduler - a task of higher priority may interrupt a task of low priority.

| Parameter | Preemptive Scheduling | Non-preemptive Scheduling |
|---|---|---|
| **Basic** | In this resources(CPU Cycle) are allocated to a process for a limited time. | Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state. |
| **Interrupt** | Process can be interrupted in between. | Process can not be interrupted until it terminates itself or its time is up. |
| **Starvation** | If a process having high priority frequently arrives in the ready queue, low priority process may starve. | If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve. |
| **Overhead** | It has overheads of scheduling the processes. | It does not have overheads. |
| **CPU Utilization** | In preemptive scheduling, CPU utilization is high. | It is low in non preemptive scheduling. |
| **Examples** | Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First. | Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First. |

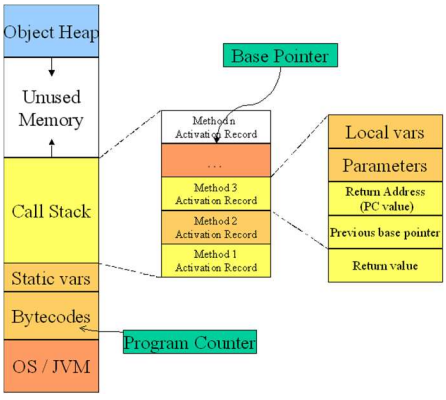
## Operating System- Functional Aspects

***Stack***

- A stack is a data structure that is used to temporarily store data objects in memory by piling one upon another.
- The objects are accessed on a last-in first-out (LIFO) basis.
- The processor core uses stacks to store system state information when it begins executing subroutines. This way it "remembers" where to return after the subroutine is completed.
- Subroutines can also call other subroutines by storing the state of the current subroutine on top of the previous state information in the stack. When the subroutine is completed, the processor pulls the first address it finds on the top of the stack and jumps to that location.
- In a multithreaded OS, each thread requires its own stack to manage state information.
- This is one of the reasons why multithreaded operating systems are expensive in WSNs.

## Operating System- Functional Aspects

***Stack***




**Call and return process**

- When a method/function is called

1. An **Activation Record** is created; its size depends on the number and size of the local variables and parameters.
2. The **Base Pointer** "Register" value is saved in the special location reserved for it
3. The **Program Counter** "Register" value is saved in the **Return Address** location
4. The Base Pointer is now reset to the new base (top of the call stack prior to the creation of the AR)
5. The Program Counter is set to the location of the first bytecode of the method being called
6. Copies the calling parameters into the Parameter region
7. Initializes local variables in the local variable region

- While the method executes, the local variables and parameters are simply found by adding a constant associated with each variable/parameter to the Base Pointer.

- When a method returns

1. Get the program counter from the activation record and replace what's in the PC register
2. Get the base pointer value from the AR and replace what's in the BP register
3. Pop the Activation Record entirely from the stack.

There is also a **Stack Pointer** that allow other temporary use of the stack above the top AR.

## Operating System- Functional Aspects

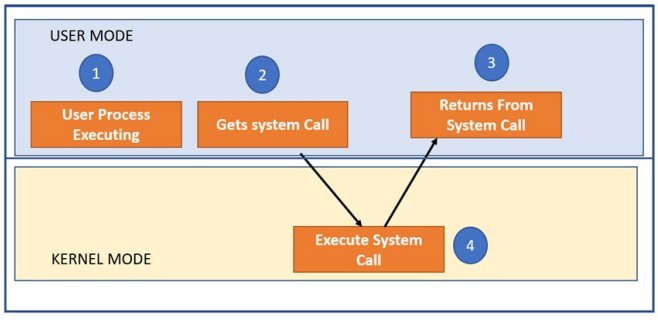
***System Calls***

- A **system call** is a mechanism that provides the interface between a process and the operating system.
- It is a programmatic method in which a computer program requests a service from the kernel of the OS.
- System call offers the services of the operating system to the user programs via API (Application Programming Interface).
- System calls are the only entry points for the kernel system.
- When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch**.
- Generally, system calls are made by the user level programs in the following situations:
  - Creating, opening, closing and deleting files in the file system.
  - Creating and managing new processes.
  - Creating a connection in the network, sending and receiving packets.
  - Requesting access to a hardware device, like a mouse or a printer.

## Operating System- Functional Aspects

***System Calls***

- Users invoke these operations whenever they wish to access a hardware resource such as a sensor, watchdog timer, or the radio without the need to concern themselves how the hardware is accessed




**Step 1)** The processes executed in the user mode till the time a system call interrupts it.

**Step 2)** After that, the system call is executed in the kernel-mode on a priority basis.

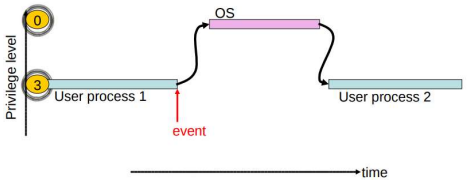
**Step 3)** Once system call execution is over, control returns to the user mode.,

**Step 4)** The execution of user processes resumed in Kernel mode.

## Operating System- Functional Aspects

***Handling Interrupts***

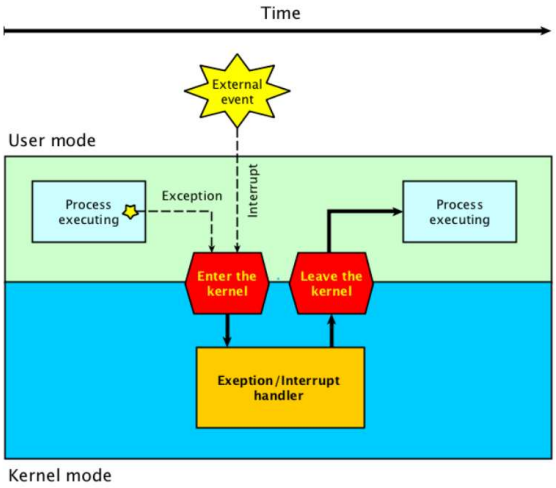
- An interrupt is an asynchronous signal generated by
  - a hardware device (a sensor, a watchdog timer, a radio)
  - several system events
  - OS itself




- An interrupt causes:
  - the processor to interrupt executing the present instruction
  - to call for an appropriate interrupt handler
- Interrupt signals can have different priority levels, a high priority interrupt can interrupt a low level interrupt
- Interrupt mask: let programs choose whether or not they wish to be interrupted.

## Operating System- Functional Aspects

- Interrupts are important because they give the user better control over the computer.
- Without interrupts, a user may have to wait for a given application to have a higher priority over the CPU to be ran. This ensures that the CPU will deal with the process immediately.



## Operating System- Functional Aspects

***Handling Interrupts***

There are three types of interrupts:

- **Hardware Interrupts** are generated by hardware devices to signal that they need some attention from the OS.
  - They may have just received some data (e.g., keystrokes on the keyboard or an data on the Ethernet card); or
  - They have just completed a task which the operating system previous requested, such as transferring data between the hard drive and memory.
- **Software Interrupts** are generated by programs when they want to request a system call to be performed by the operating system.
- **Traps** are generated by the CPU itself to indicate that some error or condition occurred for which assistance from the operating system is needed.

## Operating System- Functional Aspects

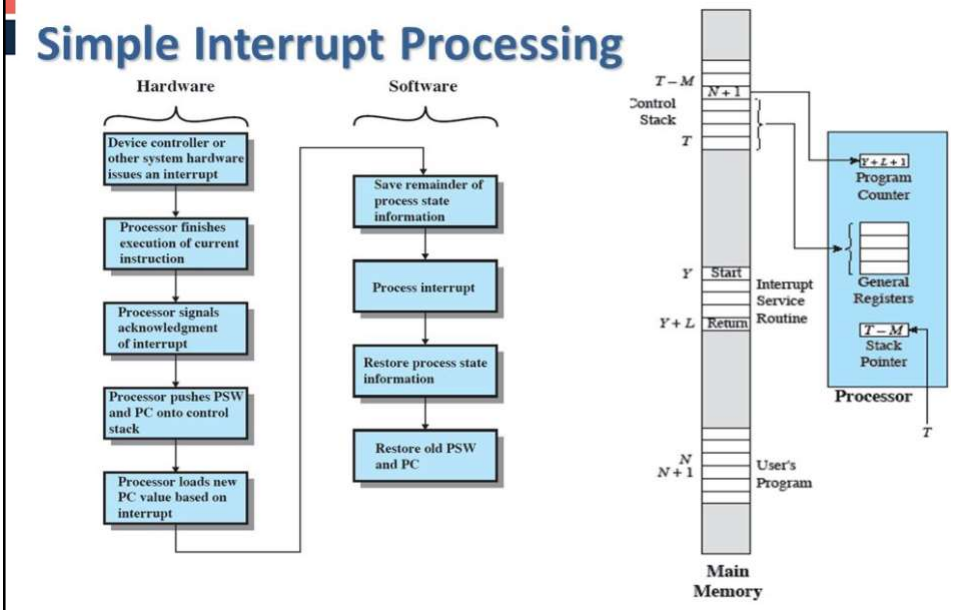
***Handling Interrupts***

When an interrupt gets active, the microcontroller goes through the following steps –

- The microcontroller closes the currently executing instruction and saves the address of the next instruction (PC) on the stack.
- It also saves the current status of all the interrupts internally (i.e., not on the stack).
- It jumps to the memory location of the interrupt vector table that holds the address of the interrupts service routine.
- The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine, which is RETI (return from interrupt).
- Upon executing the RETI instruction, the microcontroller returns to the location where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then, it start to execute from that address.

## Operating System- Functional Aspects

### Simple Interrupt Processing




## Operating System- Functional Aspects

***Multithreading***

- A *thread* is the path taken by a processor or a program during its execution
  - *Multi-threading* - a task is divided into several logical pieces
  - scheduled independent from each other
  - executed concurrently
  - Threads of the same task share a common data and address space and can communicate with each other if necessary.
- Two advantages of a multi-threaded OS:
  1. tasks do not block other tasks
  2. short-duration tasks can be executed along with long-duration tasks
- Threads cannot be created endlessly
  - the creation of threads *slows down* the processor
  - no sufficient resources to divide
- The OS can keep the number of threads to a *manageable size* using a thread pool

## Operating System- Functional Aspects

***Thread-based Vs. Event-based Programming***

- Decision whether to use threads or events programming:
  - need for separate stacks
  - need to estimate maximum size for saving context information
- ***Thread-based programs*** use multiple threads of control within:
  - a single program
  - a single address space
- ***Advantage:***
  - a thread blocked can be suspended while other tasks are executed in different threads
- ***Disadvantages:***
  - must carefully protect shared data structures with locks
  - use condition variables to coordinate the execution of threads
  - In general, program code written for multithreading environments is complex, bug-prone, and may lead to deadlocks and race conditions.

## Operating System- Functional Aspects

- In ***event-based programming*****:** use events and event handlers event-handlers register with the OS scheduler to be notified when a named event occurs.
- A loop function:
  - polls for events
  - calls the appropriate event-handlers when events occur
- An event is processed to completion
  - unless its handler reaches at a blocking operation (callback and returns control to the scheduler).

## Operating System- Functional Aspects

***Memory Allocation***

- The memory unit is a precious resource
- Reading and writing to memory is costly
- How and for how long a memory is allocated for a piece of program determines the speed of task execution

Memory can be allocated to a program:

- *Statically* - a frugal approach, but the requirement of memory *must be known* in advance
  - memory is used efficiently
  - runtime adaptation is not allowed

- *Dynamically* - the requirement of memory is *not known* in advance (on a transient basis)
  - enables flexibility in programming
  - but produces a considerable management overhead

## Operating System- Functional Aspects

***Memory Allocation***

- As a strategy to increase the memory capacity of a node, most architectures use EEPROM or flash memory to store program code.

- Consequently, it is possible to deploy relatively complex applications and communication protocols.

- However, reading and writing to flash memory is costly with respect to energy consumption.

## Operating System- Non-functional Aspects

***Separation of Concern***

- In general-purpose operating systems,
  - separation between the operating system and the applications layer.
  - interact with each other through well-defined interfaces and system calls.
  - operating system itself has several distinct services that can be upgraded, debugged, or altogether removed independently.
- The operation systems can provide:
  - a number of lightweight modules - "wired" together, or
  - an indivisible system kernel + a set of library components for building an application, or
  - a kernel + a set of reconfigurable low-level services
- Separation of concern enables:
  - flexible and efficient reprogramming and reconfiguration
  - An update or an upgrade can be made as a whole or in part as required. It enables the efficient use of communication bandwidth as well as memory space during a software update.

## Operating System- Non-functional Aspects

***System Overhead***

- An operating system executes program code – requires its own share of resources
- The resources consumed by the OS are the system's overhead, it depends on
  - the size of the operating system
  - the type of services that the OS provides to the higher-level services and applications
- Presently WSN, resources are measured in terms of kilobytes and a few megahertz.
- The resources have to be shared by programs that carry out:
  - sensing
  - data aggregation
  - self-organization
  - network management
  - network communication

## Operating System- Non-functional Aspects

***Portability***

- Ideally, operating systems should be able to co-exist and collaborate with each other
- However, existing operating systems do *not* provide this type of support
- In order to accommodate unforeseen requirements, operating systems should be portable and extensible

## Operating System- Non-functional Aspects

***Dynamic Reprogramming***

- Once a wireless sensor network is deployed, it may be necessary to reprogram some part of the application or the operating system for the following reasons:
  - the network may not perform optimally
  - both the application requirements and the network's operating environment can change over time
  - may be necessary to detect and fix bugs
- Manual replacement may not be feasible - develop an operating system to provide dynamic reprogramming support, which depends on
  - clear separation between the application and the OS
  - the OS can receive software updates and assemble and store it in memory
  - OS should make sure that this is indeed an updated version
  - OS can remove the piece of software that should be updated and install and configure the new version
  - all these consume resources and may cause their own bugs

## Operating System- Non-functional Aspects

***Dynamic Reprogramming***

- Software reprogramming (update) requires robust *code dissemination protocols*:
  - splitting and compressing the code
  - ensuring code consistency and version controlling
  - providing a robust dissemination strategy to deliver the code over a wireless link

## Operating System- Prototypes

***Tiny OS***

- TinyOS is *the most widely used*, *richly documented*, and *tool-assisted* runtime environment in WSN
  - static memory allocation
  - event-based system

- TinyOS's architecture consists of
  - a scheduler
  - a set of components, which are classified into!
    1. configuration components - "wiring" (how models are connected with each other)
    2. modules - the basic building blocks of a TinyOS program

## Operating System- Prototypes

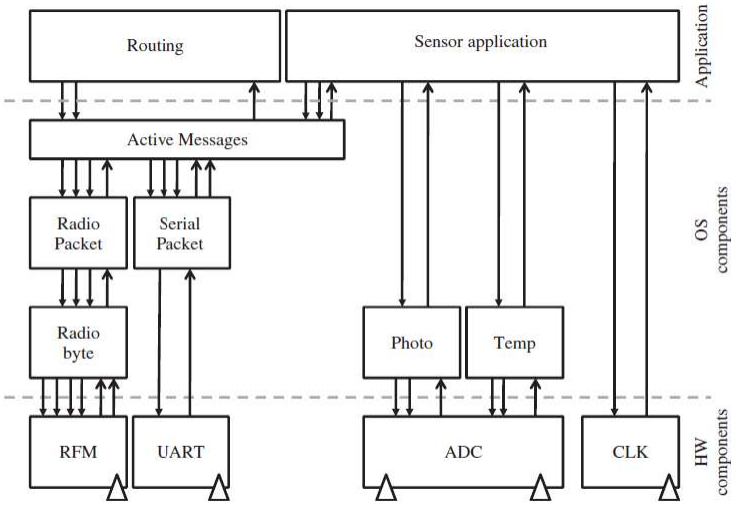
***Tiny OS***

- A component is made up of
  - a frame
  - command handlers
  - event handlers
  - a set of non-preemptive tasks

- A component is similar to an object in object-based programming languages:
  - it encapsulates state and interacts through well-defined interfaces
  - an interface that can define commands, event handlers, and tasks

## Operating System- Prototypes

***Tiny OS***




Logical distinction between the low-level and high-level components

## Operating System- Prototypes

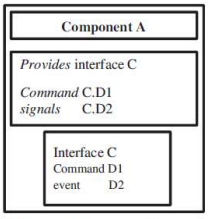
***Tiny OS***

- Components are structured hierarchically and communicate with each other through commands and events:
  - higher-level components issue commands to lower-level components
  - lower-level components signal events to higher-level components
- In Figure, two components at the highest level communicate asynchronously through active messages
  - routing component - establishing and maintaining the network
  - sensor application - responsible for sensing and processing

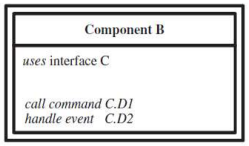
## Operating System- Prototypes

***Tiny OS***

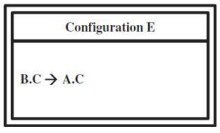
The logical structure of components and component configurations




A TinyOS component providing an interface.




A TinyOS component that uses an interface.




A TinyOS configuration that wires an interface provider and an interface user.

*Component A* declares its service by providing *interface C*, which in turn provides *command D1* and signals *event D2*.

*Component B* expresses interest in *interface C* by declaring a call to *command D1* and by providing an event handler to process *event D2*.

a binding between *Component A* and *Component B* is established through the *Configuration E*.

## Operating System- Prototypes

***Tiny OS***

- The fundamental building blocks of a TinyOS runtime environment: *tasks, commands,* and *events*
  - enabling effective communication between the components of a single frame
- *Tasks :*
  - *monolithic processes* - should execute to completion – they cannot be preempted by other tasks, though they can be interrupted by events
  - possible to allocate a single stack to store context information
  - call lower level commands; signal higher level events; and post (schedule) other tasks
  - scheduled based on FIFO principle (in TinyOS)

## Operating System- Prototypes

***Tiny OS***

- ***Commands:***
  - non-blocking requests made by higher-level components to
  - lower-level components
  - split-phase operation:
    - a function call returns immediately
    - the called function notifies the caller when the task is completed

- ***Events:***
  - events are processed by the event handler
  - event handlers are called when hardware events occur
  - an event handler may react to the occurrence of an event in different ways
    - deposit information into its frame, post tasks, signal higher level events, or call lower level commands