

Unit-1

Introduction to Internet of Things



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot

✉ bhushan.joshi@darshan.ac.in
📞 +91 8485979997



Outline

- ✓ Definition and Characteristics of IoT
- ✓ Physical Design of IoT
- ✓ IoT Components
- ✓ IoT Stack
- ✓ IoT Protocols & IoT communication models
- ✓ Enabling technologies
- ✓ Domain specific IoT Application areas

Disclaimer : "The images used in this presentation are sourced solely for educational and non-profit purposes. The author of this presentation asserts no ownership or copyright claims over them."



Introduction To IoT

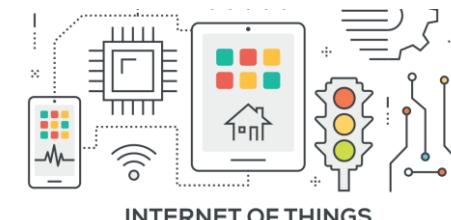
Section - 1

IoT : Internet of Things

- ▶ The Internet of Things (IoT) refers to a network of physical objects that are connected to the Internet.
- ▶ They can **exchange data and information** in order to improve productivity, efficiency, services, and more.
- ▶ A network of physical objects or ‘things’ that can interact with each other to share information and take action.
- ▶ The interconnection of uniquely identifiable embedded computing devices within the existing Internet infrastructure.
- ▶ The Internet of Things (IoT) refers to an integrated system of physical objects (**things**) embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data within the systems over the “**internet**”.

IoT : Everything Will be Connected

- ▶ The physical objects used in the IoT range from ordinary household objects to sophisticated industrial tools connected to the internet via embedded devices.
- ▶ IoT is one of the fastest emerging technologies, providing enormous beneficial opportunities for individuals, society, and governments.
- ▶ IoT is not a single technology, it's a **combination of technologies** and **domain knowledge**.
- ▶ As a result, engineers from different domains have to work together for building a complete IoT product.
- ▶ Life would be governed entirely by Internet and IoT in the near future.
- ▶ IoT is a sensor network of billions of smart devices that connect people, systems and other applications to collect and share data.



Introduction to Internet of Things (IoT): New Age

- ▶ Internet of Things (IoT) is an emerged as the **4th Industrial Revolution**.
- ▶ There are multiple ways to define IoT, but the basic of all the definitions remains the same.
- ▶ The IoT is not just limited to the connected or networked devices, but in a broad way IoT devices exchange meaningful information from one device to another to get desire result.



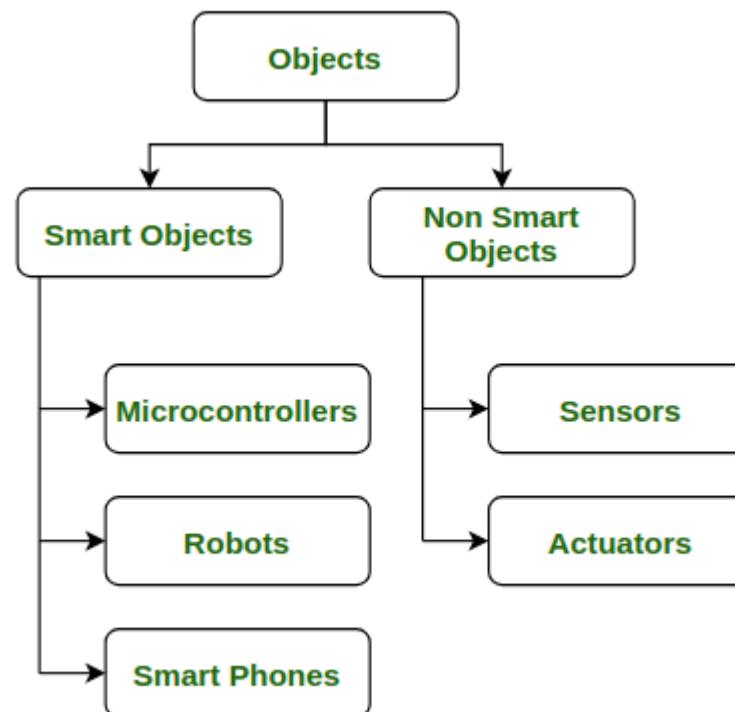
Things in IoT

- ▶ A **Thing** in the IoT can be any natural or man-made object that can be assigned an IP address and is able to transfer data over a network
 - A person with a heart monitor implant
 - A farm animal with a biochip transponder
 - An automobile that has built-in sensors to alert the driver when tire pressure is low
- ▶ IoT is a sensor network of billions of smart devices that connect people, systems and other applications to collect and share data.
- ▶ 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.



Things in IoT

1. Objects with intelligence or **Smart Objects**.
2. Objects without intelligence or **Non-Smart Objects**.



Things in IoT

- ▶ The “Thing” in a network can be monitored/measure.
 - For example, a temperature sensor could be a thing.
- ▶ Things are capable of exchanging data with other connected devices in the system.
- ▶ The data could be stored in a **centralized server (or cloud)**, processed there and a control action could be initiated.
- ▶ The data from these sensors are collected and sent it to the cloud or stored it in local server for data analysis. Based on the data analysis, the control action would be taken.



Things in IoT

► Not just sensors, the following can also be called as things:

- Industrial motors
- Wearable's (e.g., watch)
- Vehicles
- Shoes
- Heart monitoring implants
 - (e.g., pacemaker, ECG real-time tracking)
- Biochip transponders
 - (for animals in farms)
- Automobiles with built-in sensors (automobile feature real-time monitoring)
- Food quality



Things in IoT : A Smart Home

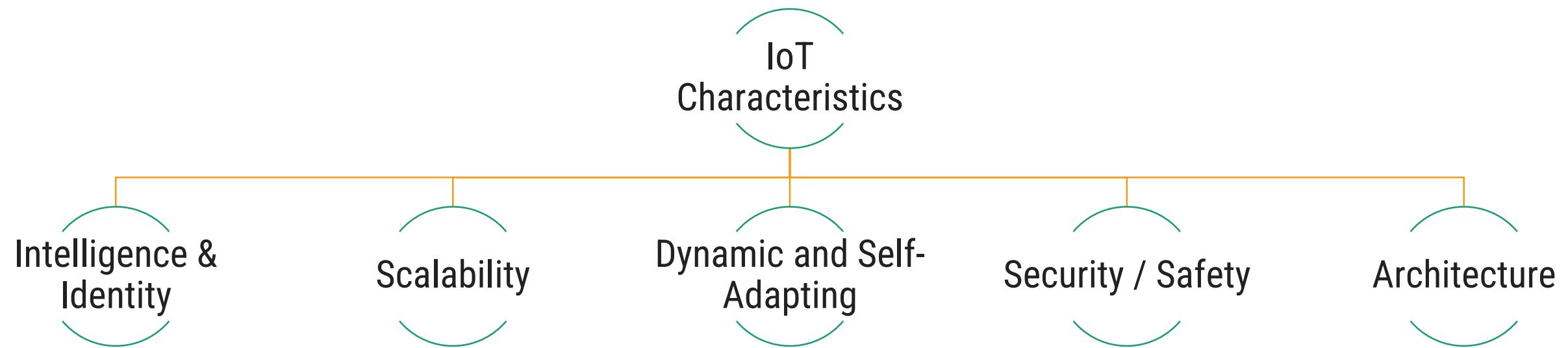
- ▶ In IoT-based Home Automation, the “**Things**” could be the following:
 - Lighting control and automation devices
 - Ventilation devices
 - Air conditioning [heating, ventilation and air conditioning (HVAC)] systems
 - Appliances such as washer/dryer
 - Air purifiers
 - Ovens or refrigerators/freezers that use Wi-Fi for remote monitoring
 - Security cameras



IoT Characteristics

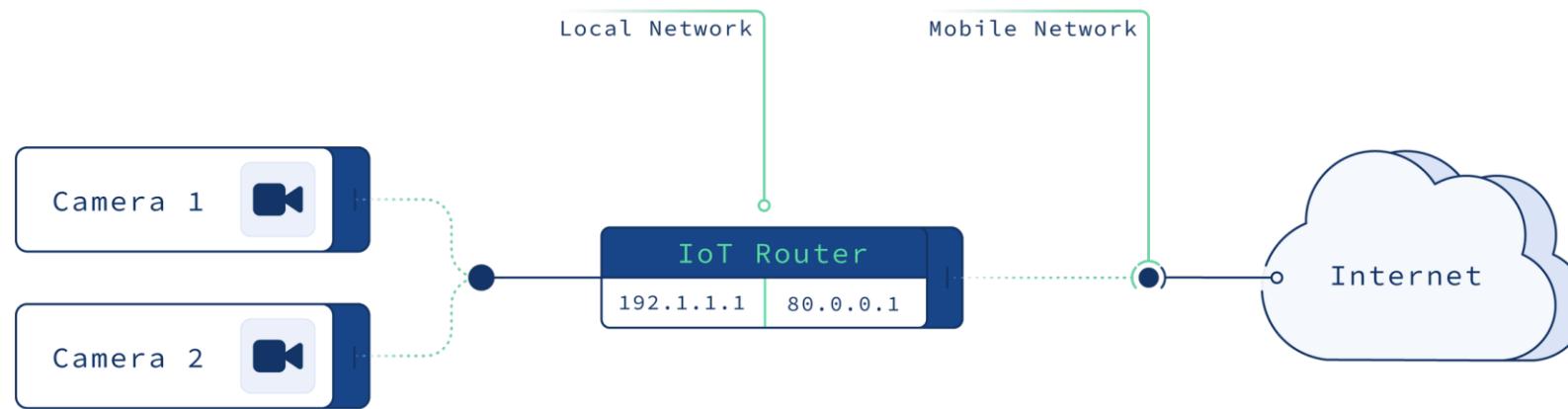
Section - 2

IoT Characteristics



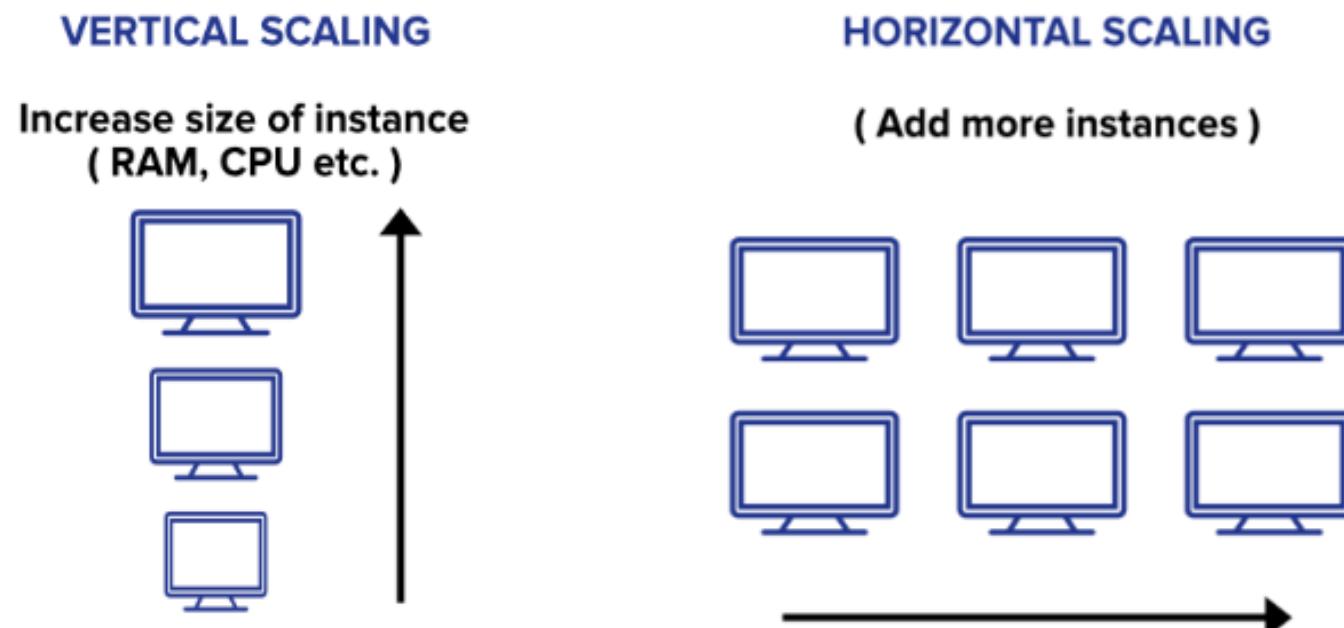
Characteristics of IoT : Intelligence and Identity

- ▶ The extraction of knowledge from the generated data is very important.
 - For example, a sensor generates data, but that data will only be useful if it is interpreted properly.
- ▶ Each IoT device has a **unique identity**.
- ▶ This identification is helpful in tracking the equipment and at times for querying its status.



Characteristics of IoT :**Scalability**

- ▶ The number of elements (devices) connected to IoT zone is increasing day by day.
- ▶ Therefore, an IoT setup should be capable of handling the **expansion**.
- ▶ **Vertical Scaling** : Expand capability in terms of processing power, Storage, etc.
- ▶ **Horizontal Scaling** : By multiplying with devices numbers or cloning system.



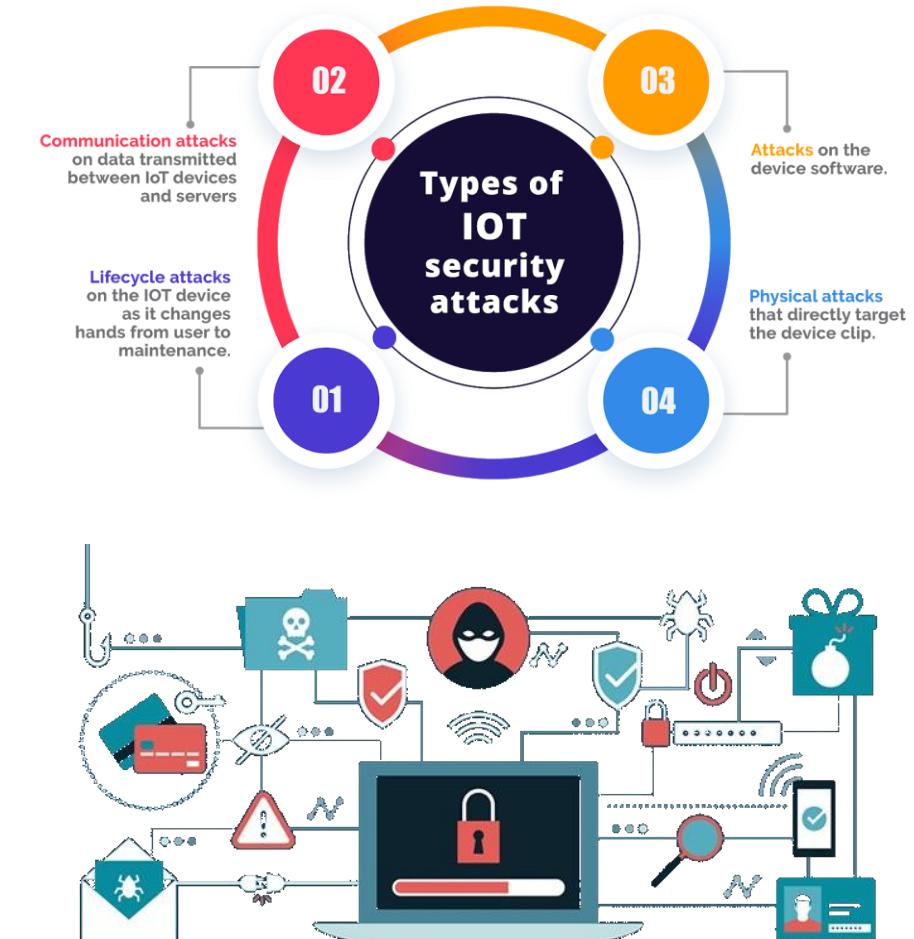
Characteristics of IoT : Dynamic & Self Adapting

- ▶ IoT devices should dynamically adapt themselves to the changing surroundings.
- ▶ For example surveillance camera. It should be flexible to work in different weather conditions and different light situations (Day/Night).
- ▶ Weather stations can automatically adjust climate conditions in greenhouses based on instructions.
- ▶ Temperature monitors can help farmers identify weather trends and detect abnormalities.
- ▶ They can also help farmers adjust watering schedules and ensure that crops receive the right amount of water at the right time.



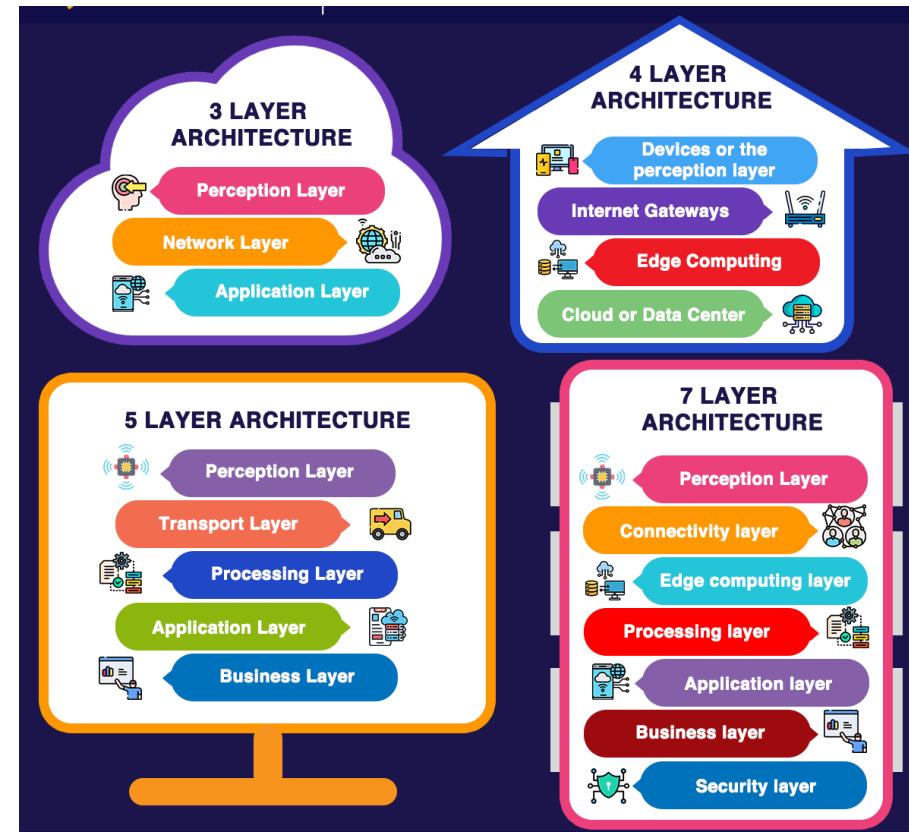
Characteristics of IoT : Security / Safety

- ▶ IoT security (internet of things security) is the technology segment focused on safeguarding connected devices and networks in IoT.
- ▶ Due to the unconventional manufacturing of IoT devices and the vast amount of data they handle, there's a constant threat of cyber attacks.
- ▶ Sensitive personal details of a user might be compromised when the devices are connected to the Internet. So **data security** is a major challenge.
- ▶ This could cause a loss to the user data.
- ▶ The Equipment in the huge IoT network may also be at risk



Characteristics of IoT : Architecture

- ▶ IoT architecture is yet not uniformed and standardized.
- ▶ It should be hybrid, supporting different manufacturer's products to function in the IoT network.

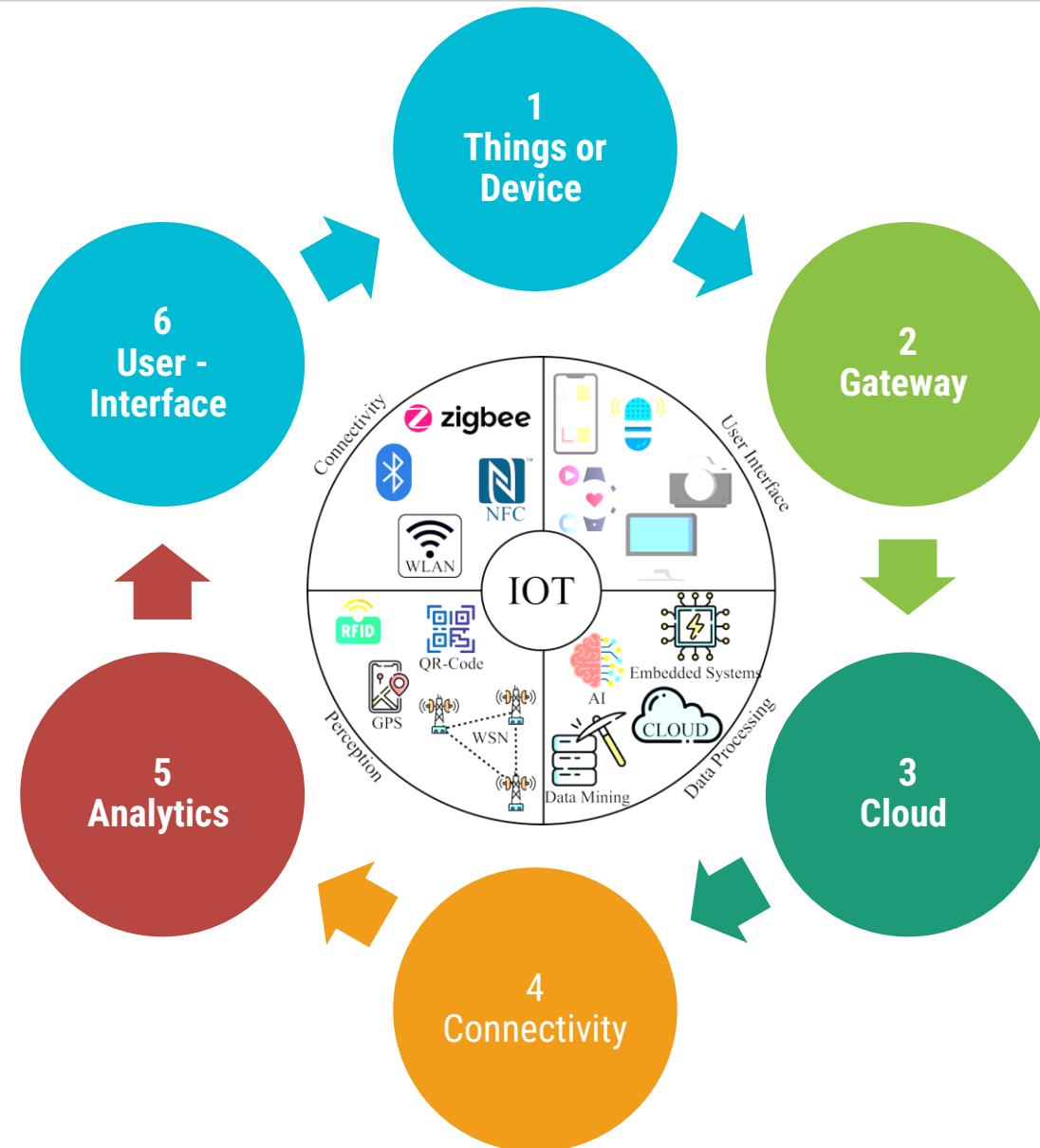




IoT Components

Section - 3

Components of IoT



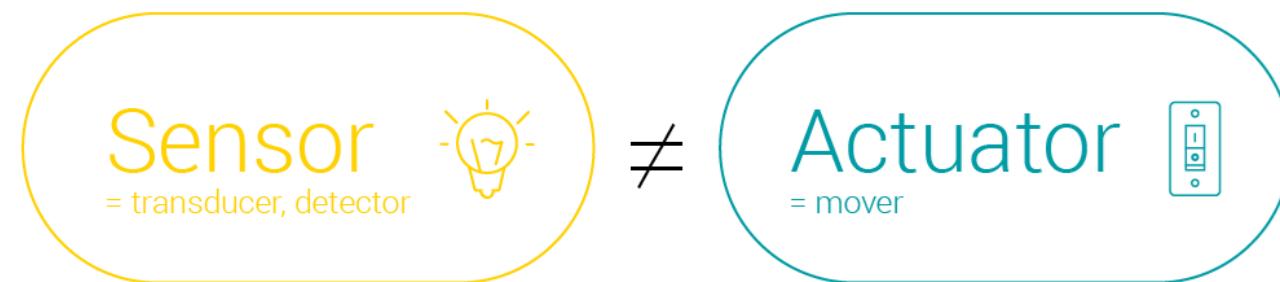
Components of IoT : Things or Devices

► Sensor

- The sensor is the layer that connects the IoT device to the outer environment or human being.
- As the name suggests, it senses the changes and sends data to the cloud for processing.
- These sensors continuously collect data from the environment and transmit the information to the next layer.
Example: pressure sensor, temperature sensor, and light intensity detectors.
- Sensors collect data from the environment.

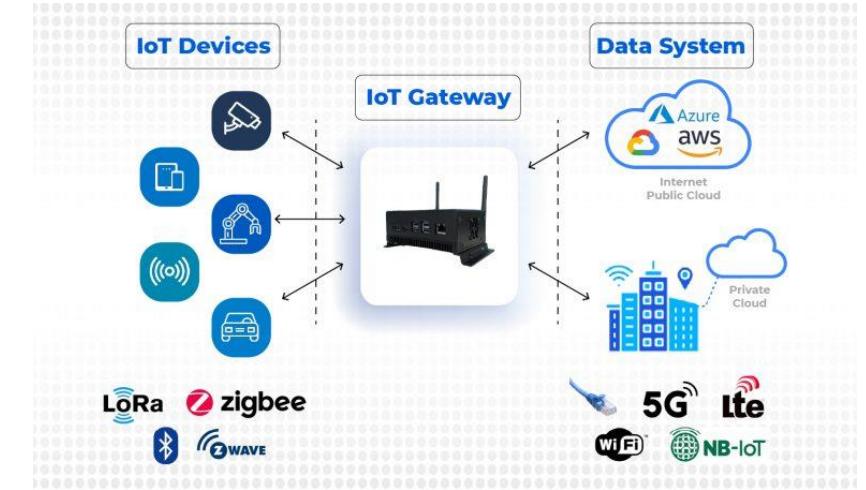
► Actuator

- An actuator is a device that produces a motion by converting energy and signals going into the system.
- Example : Motor, LED light
- Actuators performs the action



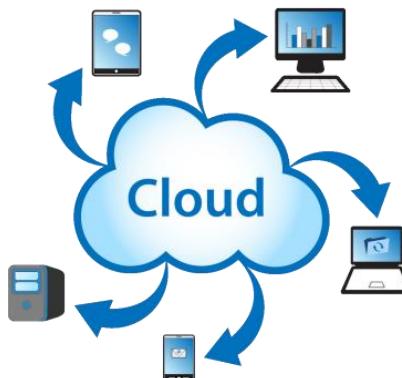
Components of IoT : **Gateway**

- ▶ IoT Gateway serves as a bridge between the devices and the cloud, allowing for communication, data collection, and data processing.
- ▶ Gateway facilitates **Data Flow Management And Protocol Layer** to transfer data from one device to another.
- ▶ It translates the network protocols for devices and provides encryption for the data flowing in the network.
- ▶ It's like a **layer between the cloud and devices** that filter away cyber attacks and illegal data access.



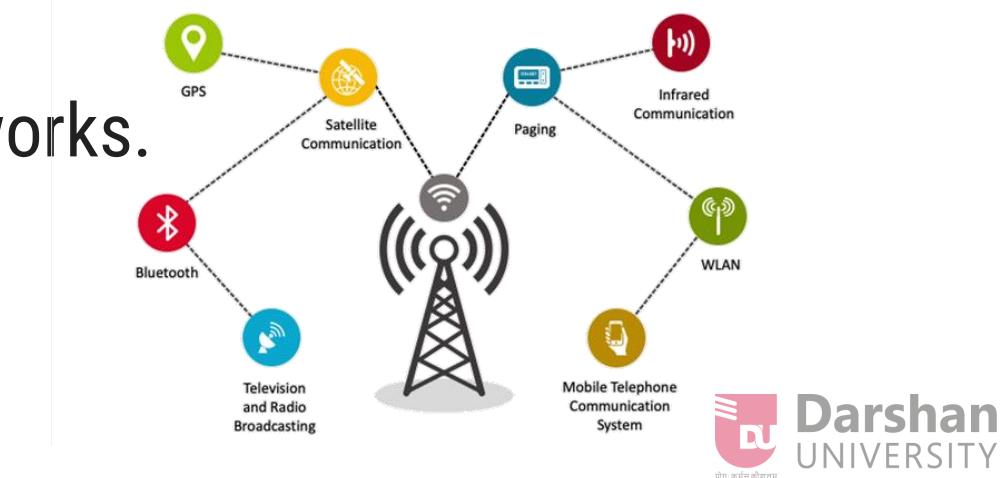
Components of IoT : Cloud

- ▶ IoT systems send massive data from devices, and this data needs to be managed efficiently to generate meaningful output. Cloud is one of the best option.
- ▶ To store this huge amount of data.
- ▶ It provides tools to collect, process, and store data.
- ▶ Data is readily available and remotely accessible through the internet.
- ▶ It also provides a platform for analytics.
- ▶ IoT cloud is a sophisticated, high-performance network of servers to perform high-speed processing of a huge amount of data.



Components of IoT : Connectivity

- ▶ The data collected by sensors need to be sent to the cloud for analytics.
- ▶ Data needs the internet as a medium to be transfer from one device to another.
- ▶ IoT devices **need to be connected to the internet all the time**, Connectivity is an important requirement of the IoT infrastructure.
- ▶ Things of IoT should be connected to the IoT infrastructure; **anyone, anywhere, anytime can connect**, this should be guaranteed at all times.
- ▶ Networks like Wi-Fi, Bluetooth, WAN, satellite networks make it easy to stay connected.
- ▶ Data can be sent to the cloud through these networks.



Components of IoT : Analytics

- ▶ Analytics is the process of **converting raw data into some meaningful form**.
- ▶ IoT analytics supports real-time analysis, which captures real-time changes and irregularities.
- ▶ The data is then converted into a format that is easy to understand by the end-user.
- ▶ As a result, users or businesses can analyse the trends shown in reports, predict the market and plan ahead for a successful implementation of their ideas.

| Date | Time | Device_ID | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 |
|------------|------------|-----------|---------------------|-----------------------|---------|------------------|----------------|----------------|----------------|
| 16/12/2006 | 5:24:00 PM | 3229987 | 4.216 | 0.418 | 234.84 | 18.4 | 0 | 1 | 17 |
| 16/12/2006 | 5:25:00 PM | 3229987 | 5.36 | 0.436 | 233.63 | 23 | 0 | 1 | 16 |
| 16/12/2006 | 5:26:00 PM | 3229987 | 5.374 | 0.498 | 233.29 | 23 | 0 | 2 | 17 |
| 16/12/2006 | 5:27:00 PM | 3229987 | 5.388 | 0.502 | 233.74 | 23 | 0 | 1 | 17 |
| 16/12/2006 | 5:28:00 PM | 3229987 | 3.666 | 0.528 | 235.68 | 15.8 | 0 | 1 | 17 |
| 16/12/2006 | 5:29:00 PM | 3229987 | 3.52 | 0.522 | 235.02 | 15 | 0 | 2 | 17 |
| 16/12/2006 | 5:30:00 PM | 3229987 | 3.702 | 0.52 | 235.09 | 15.8 | 0 | 1 | |
| 16/12/2006 | 5:31:00 PM | 3229987 | 3.7 | 0.52 | 235.22 | 15.8 | 0 | 1 | |
| 16/12/2006 | 5:32:00 PM | 3229987 | 3.668 | 0.51 | 233.99 | 15.8 | 0 | 1 | 17 |
| 16/12/2006 | 5:33:00 PM | 3229987 | 3.662 | 0.51 | 238.86 | 15.8 | 0 | 2 | 16 |
| 16/12/2006 | 5:34:00 PM | 3229987 | 4.448 | 0.498 | 232.86 | 19.6 | 0 | 1 | 17 |
| 16/12/2006 | 5:35:00 PM | 3229987 | 5.412 | 0.47 | 232.78 | 23.2 | 0 | 1 | 17 |
| 16/12/2006 | 5:36:00 PM | 3229987 | 5.224 | 0.478 | 232.99 | 22.4 | 0 | 1 | 16 |
| 16/12/2006 | 5:37:00 PM | 3229987 | 5.268 | 0.398 | 232.91 | 22.6 | 0 | 2 | 17 |
| 16/12/2006 | 5:38:00 PM | 3229987 | 4.054 | 0.422 | 235.24 | 17.6 | 0 | 1 | 17 |
| 16/12/2006 | 5:39:00 PM | 3229987 | 3.384 | 0.282 | 237.14 | 14.2 | 0 | 0 | 17 |
| 16/12/2006 | 5:40:00 PM | 3229987 | 3.27 | 0.152 | 236.73 | 13.8 | 0 | 0 | 17 |



Components of IoT : User interface

- ▶ The User interface is a visible, tangible part of IoT systems. It is that part of the system which **interacts with the end-user**.
- ▶ The information can be made available either in report format or in the form of some actions like trigger an alarm, a notification, etc.
- ▶ The user can also choose to perform some actions.
- ▶ Therefore, it is important to create a **user-friendly interface** that can be used without much effort and technical knowledge.
- ▶ The easier the user interface is, the more successful the product is.

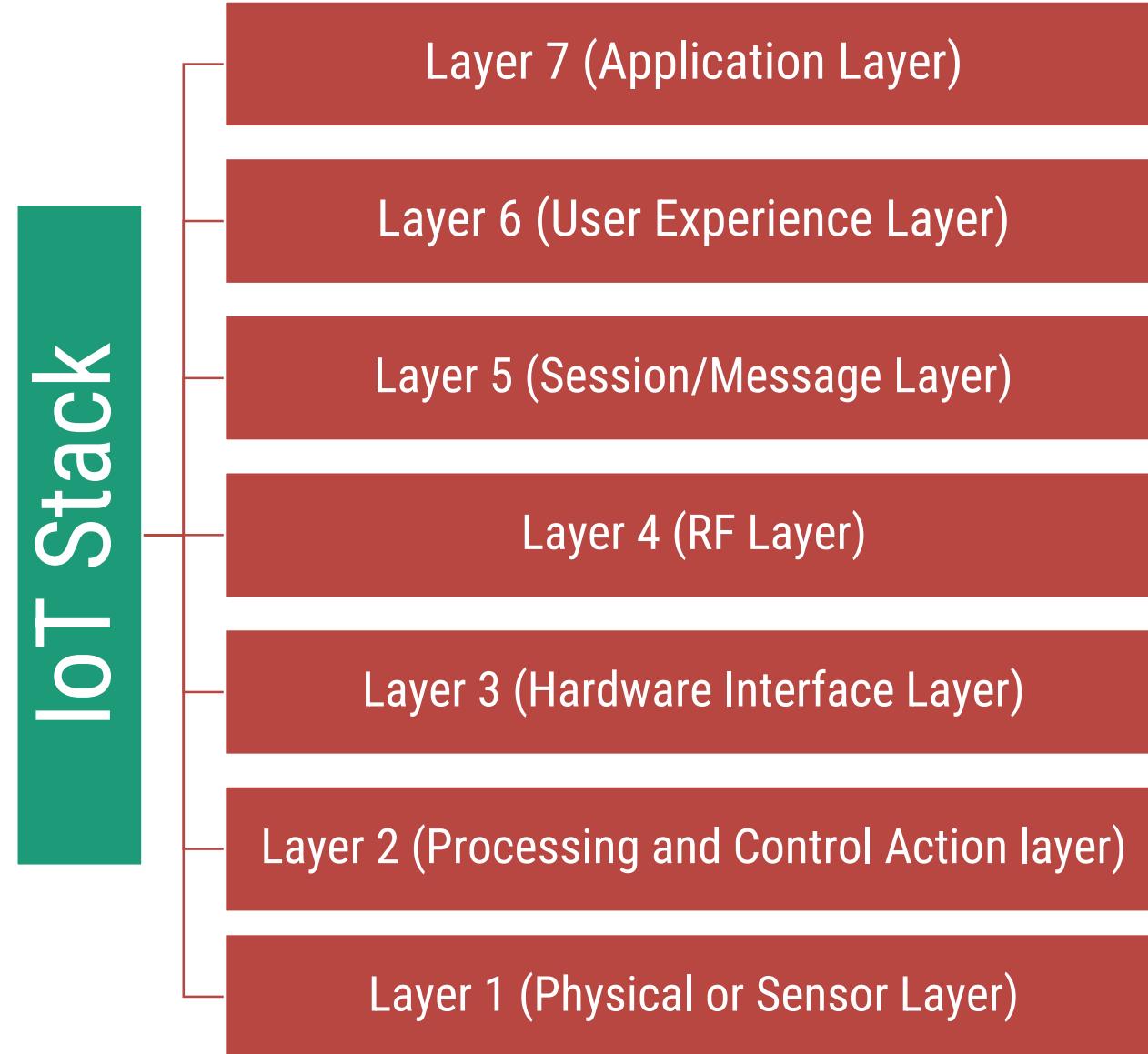




IoT Stack – 7 Layer Architecture

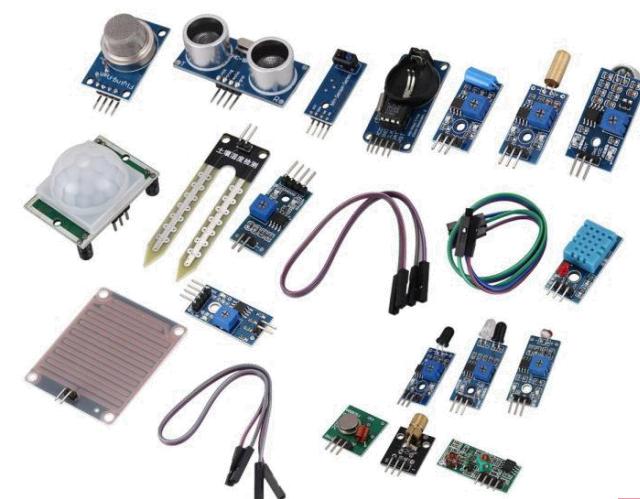
Section - 4

IoT Stack – 7 Layer Architecture



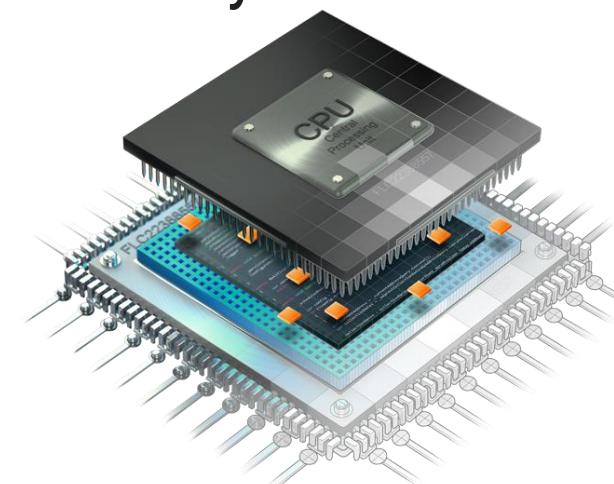
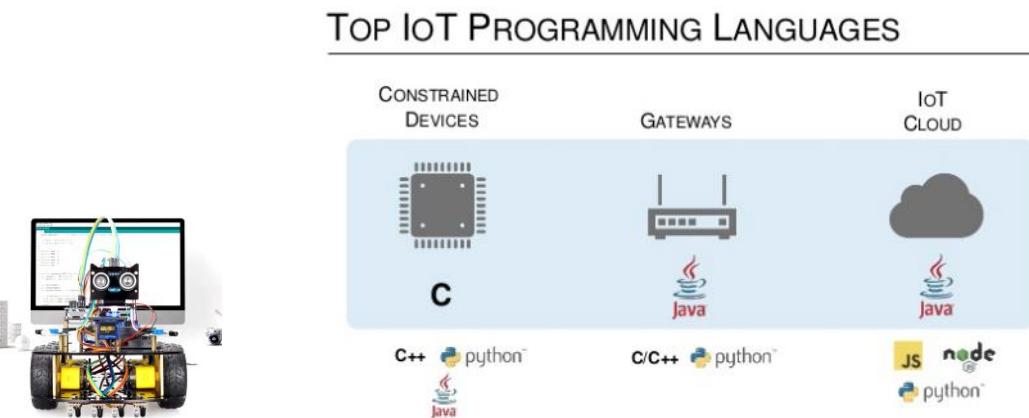
IoT Stack - Layer 1 (Physical or Sensor Layer)

- ▶ Physical components, which mainly includes **sensors and actuators**.
 - Example : Temperature sensor, pressure sensor, humidity sensor, etc.
- ▶ These all can be referred as physical layer components.
- ▶ This layer is responsible for **data collection and action execution**.
- ▶ Selection of sensors is important and choosing an appropriate sensor is the challenge in this layer.
- ▶ Function of layer :
 - A. Action execution (via Actuators)
 - B. Sensing data (via Sensors)



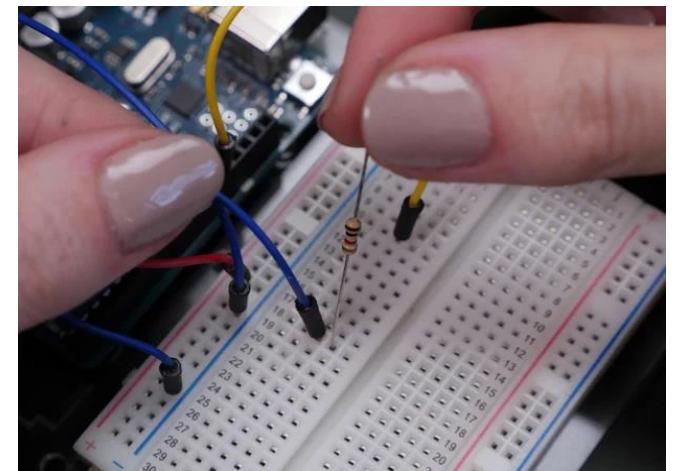
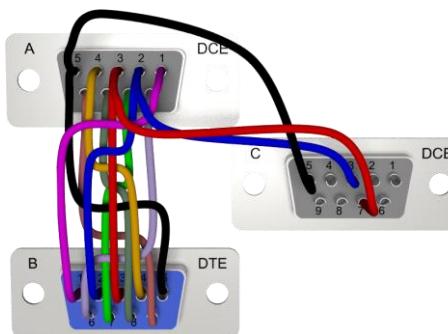
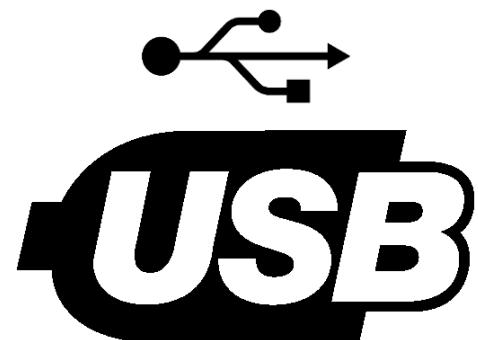
IoT Stack - Layer 2 (Processing and Control Action Layer)

- ▶ This important layer contains **core components** of IoT system.
- ▶ The **microcontrollers or processors** are found in this layer.
- ▶ The microcontrollers received data from the sensors.
- ▶ A variety of development kits are available in the market
 - Arduino, Raspberry Pi, Node MCU, PIC, ARM development boards, etc.
- ▶ Microcontroller/Processor and operating system play vital role at this layer.
- ▶ Data collected from the sensors is processed in this layer.



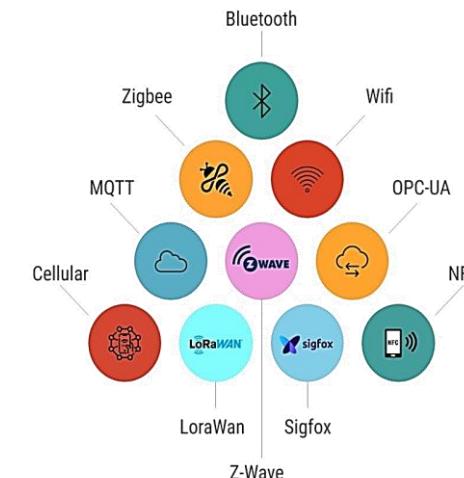
IoT Stack - Layer 3 (Hardware Interface Layer)

- ▶ Hardware Interface Layer connects the components.
- ▶ Hardware components and communication standards occupy this layer. such as...
 - RS232
 - CAN
 - SPI
 - SCI
 - I2C, etc.
- ▶ All these components ensure flawless communication

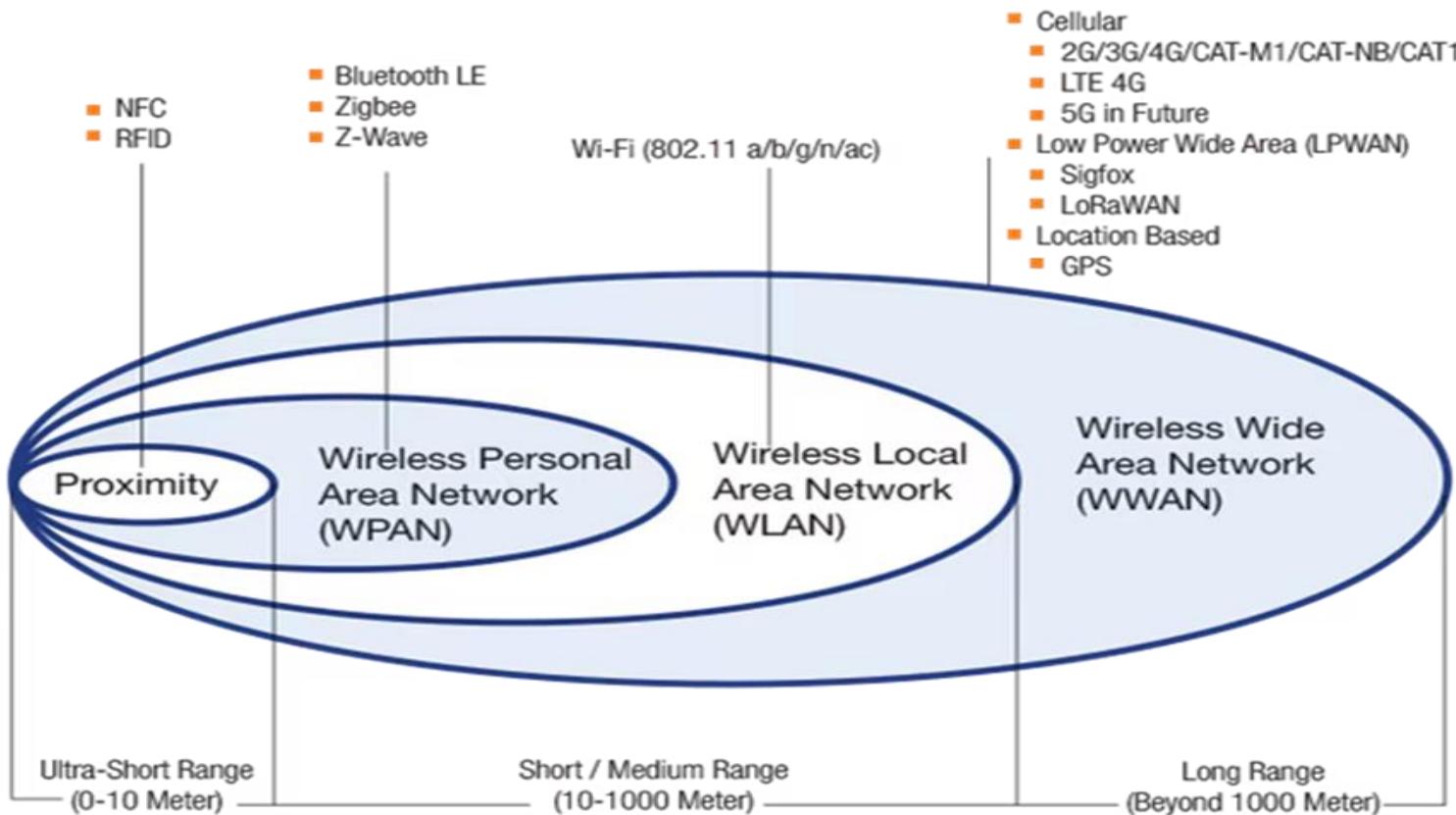


IoT Stack - Layer 4 (RF Layer)

- ▶ RF layer does communication of data using radio frequency based Electromagnetic (EM) waves. It is Wireless communication.
- ▶ RF plays a major role in the communication channel – whether it is short range or long range.
- ▶ Protocols used for communication and transport of data based on RF are listed in this layer.
- ▶ Some famous and common protocols are Wi-Fi, NFC, Bluetooth, Zig-bee, LoRa etc.



RF Protocols for an IoT Application

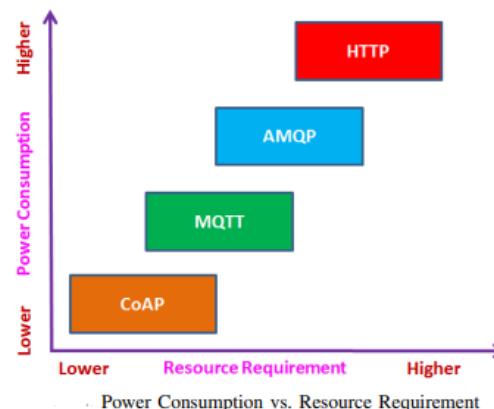


Choose As per
your Range and
Power
Requirement



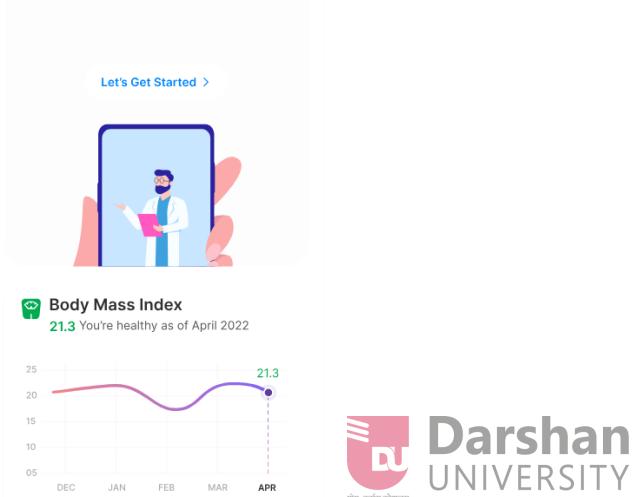
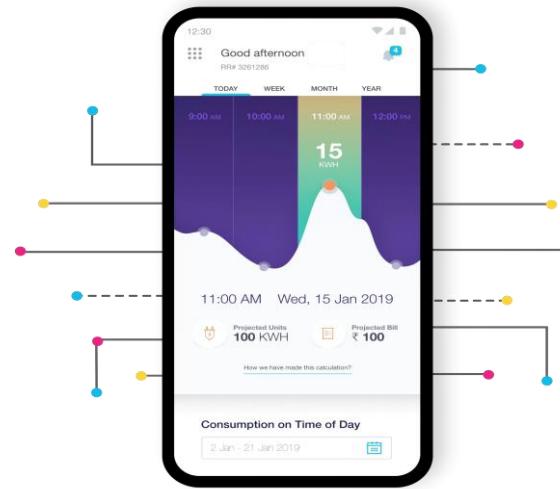
IoT Stack - Layer 5 (Session/Message Layer)

- ▶ A protocol in the Internet of Things (IoT) that manages and establishes communication sessions between users on different machines is a part of session /message layer.
- ▶ The Session Layer manages the connection between two endpoints of a network, by controlling data between sender and receiver.
- ▶ The session layer **protocols** are responsible for the actual transmission of data in the IoT ecosystem.
- ▶ Layer 5 (session layer) deals with the various messaging protocols as MQTT, CoAP.



IoT Stack - Layer 6 (User Experience Layer)

- ▶ This layer deals with **providing best experience to the end users** of IoT products.
- ▶ The 6th layer takes care of **rich UI designs** with lots of features, which provide a pleasing experience while using the service/system or product.
- ▶ Object-oriented programming languages, scripting languages, analytics tools, etc. all should be included in this layer.
- ▶ This is also known as User Experience and Visualization Layer.



IoT Stack - Layer 7 (Application Layer)

- ▶ Everything comes to perfection at this layer.
- ▶ This layer utilizes the rest six layers in order to develop desired application.
- ▶ It can range from a simple automation application to smart city application.





IoT Protocols And Communication

Section - 5

How Do IoT Devices Communicate with Each Other?

- ▶ IoT devices communicate through complex networks using different protocols designed for specific tasks and environments.

Communications can be classified

Device-to-device

Device-to-gateway

Device-to-cloud or
Data centre

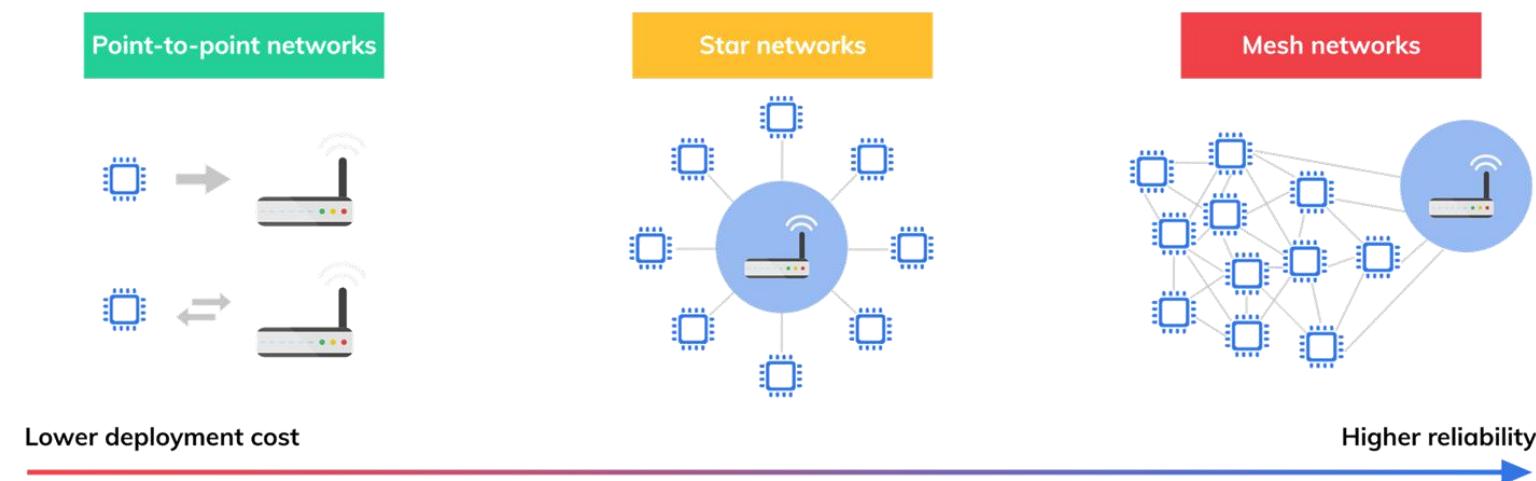
IoT Protocols

- ▶ Protocols are a set of rules for transmitting data between electronic devices according to a preset agreement regarding information structure and how each side will send and receive data.
- ▶ Each IoT protocol enables either **device-to-device**, **device-to-gateway** or **device-to-cloud/data center communication** -- or combinations of those communications.
- ▶ Factors such as **geographic and special location**, **power consumption needs**, **battery-operated options**, **the presence of physical barriers** and **cost** determine which protocol is optimal in an IoT deployment.
- ▶ IoT protocols and standards are broadly classified into two separate categories.
 1. **Network protocols for IoT** (Datalink layers (RF layer) / Physical layers)
 2. **IoT data protocols** (User Interface layer / Application layers)
- ▶ Data protocols mainly focus on information exchange, while network protocols provide methods of connecting IoT edge devices with other edge devices or the Internet.

IoT Communication Technologies: Network Protocols

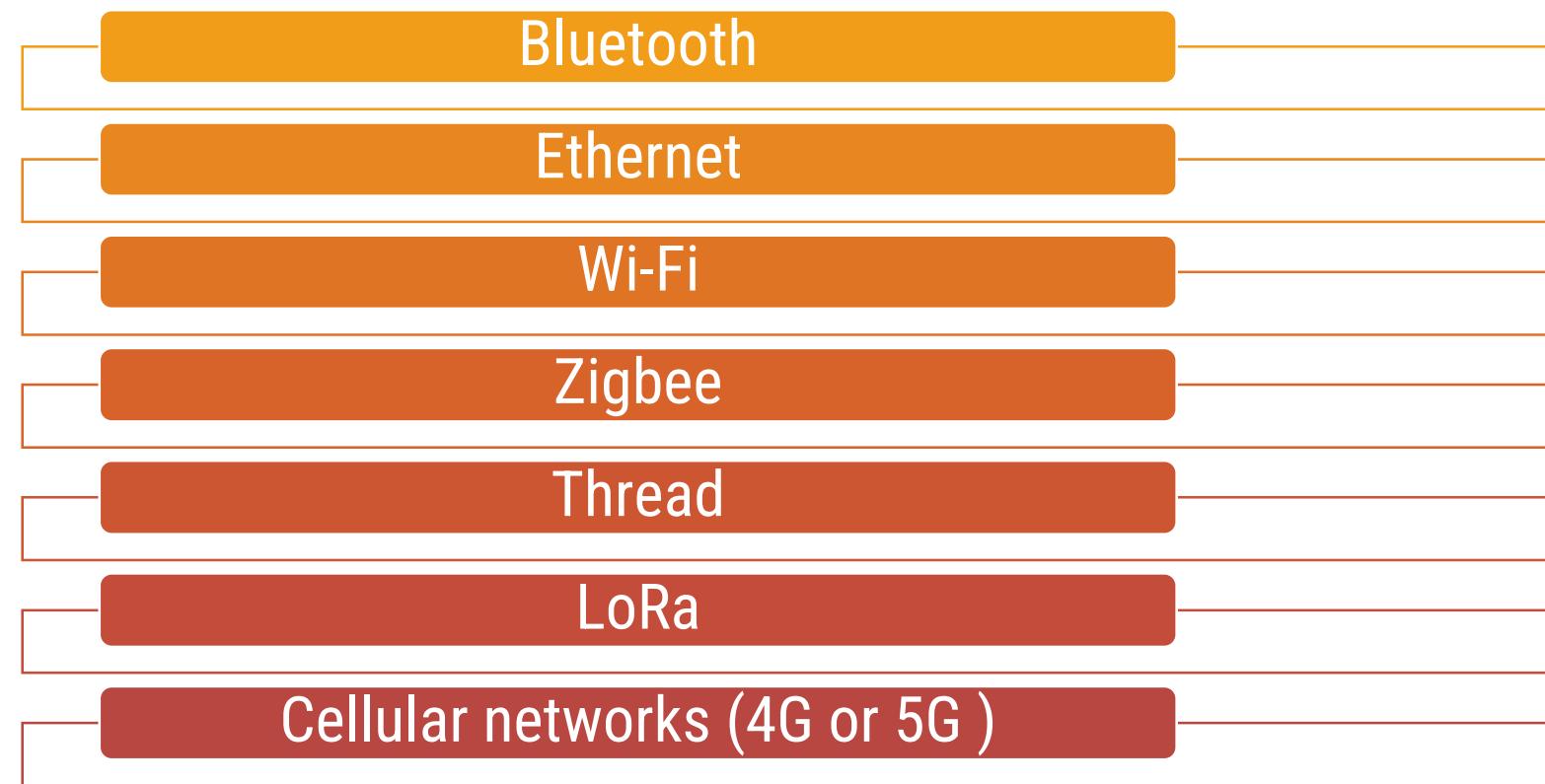
- ▶ IoT network communication protocols connect medium and high-power IoT devices over a network.
- ▶ This technology typically operates over the internet.
- ▶ There are three common ways to connect IoT devices: point-to-point, star networks, and mesh topology.

IoT connectivity: network topology



Network Protocols for IoT

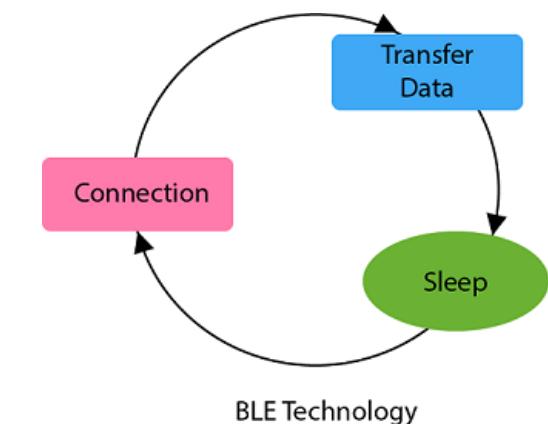
- ▶ IoT network protocols are used to connect devices over a network.
- ▶ These sets of protocols are typically used over the internet.
- ▶ Here are some examples of various IoT network protocols.



Bluetooth and Bluetooth Low Energy (BLE)

- ▶ Bluetooth is a **wireless technology** that allows data to be exchanged in **small amounts** over **short distances** at a **high speed**.
- ▶ Its ability to transfer data faster comes at the cost of relatively high power consumption.
- ▶ In 2010, Bluetooth was optimized for short-range IoT communications. Known as **BLE**, this version of the connectivity protocol boasts greater energy efficiency. However, their data transfer speed is also slower compared to Bluetooth.

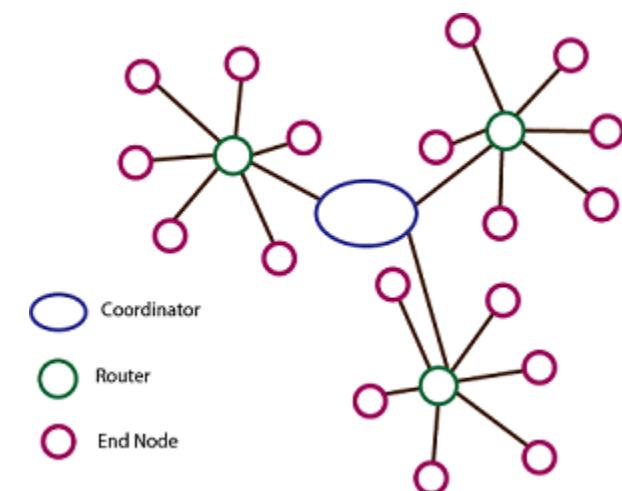
| <u>Advantages:</u> | <u>Disadvantages:</u> |
|---|---|
| Low latency | Uses the crowded 2.4 GHz frequency |
| Lower implementation costs due to hardware simplicity | Might not be suitable for large files |
| Easy internet access via a smartphone | A limited number of connected devices |
| Data encryption by default | |



ZigBee

- ▶ ZigBee is a robust and scalable IoT protocol that is used to gather sensor data in home automation and industrial applications.
- ▶ ZigBee transfers **small amounts of data** over moderate distances.
- ▶ ZigBee operates on a self-healing mesh topology, which makes it highly reliable.
- ▶ New devices can join the network after performing the “**handshake**” process, which takes merely 30 milliseconds

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|---|---|
| Can accommodate up to 65,000 devices | Uses the common 2.4 GHz frequency, which is prone to interferences |
| Low power consumption (small devices can operate on one battery for several years) | Requires a custom gateway, which is expensive |
| Relatively long range of communication | |



Z-Wave

- ▶ This is a low-power wireless IoT protocol, which is commonly used for smart home solutions and business applications.
- ▶ A single Z-Wave network can potentially handle a maximum of 232 devices, including the central controller.
- ▶ This technology operates at a different frequency in each country, meaning that users must purchase a new device when changing their location.
- ▶ **Z-Wave is a proprietary technology** managed by Z-Wave Alliance that oversees certifications.

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|--|------------------------|
| Low latency | Low data transfer rate |
| Avoids the crowded 2.4 GHz frequency used by Wi-Fi, Bluetooth, and Zigbee | Premium prices |
| Low power consumption | |
| Reasonable coverage | |



Wi-Fi (Wireless Fidelity)

- ▶ Wi-Fi uses Internet Protocol (IP) to connect devices on a Local Area Network (LAN).
- ▶ It ensures reliable and secure communication between closely located devices — for instance, gadgets within one IoT application deployment like smart home.
- ▶ This IoT protocol is relatively **cheap and easy to implement**.
- ▶ It works well with heavy files and can handle vast amounts of data.
- ▶ However, this IoT communication protocol is too power-consuming and has inherent coverage limitations.

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|--------------------------------|-------------------------------|
| Convenient and easy to install | High power consumption |
| High data transfer rate | Lack of scalability |
| | Short-distance communications |



Long-Range Radio Wide Area Network (LoRaWAN)

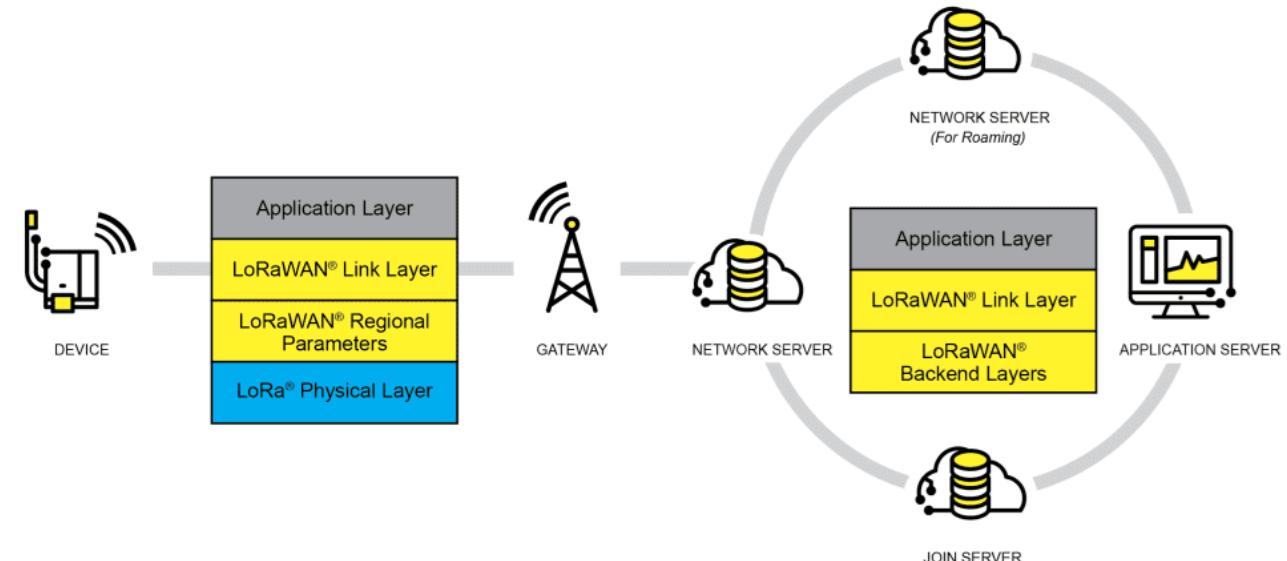
- ▶ This is a **non-cellular** wireless wide area network technology that connects devices over a long range, which makes it suitable for smart cities and industrial systems.
- ▶ LoRaWAN can potentially connect millions of IoT devices and is optimized for low power consumption.
- ▶ New devices can be either hard-coded or arranged into an **over-the-air connection**.
- ▶ A LoRa gateway gathers data from different sensors and transmits it to a server or the cloud over a standard IP protocol.
- ▶ LoRa Wide Area Network offers two security layers — one for the network layer and the other for the application.
- ▶ This IoT communication protocol is **not an option** for applications that **require low latency** or transfer a **large amount of data**.



Long-Range Radio Wide Area Network (LoRaWAN)

► LoRa Architecture:

LoRaWAN® Network Architecture



| <u>Advantages:</u> | <u>Disadvantages:</u> |
|----------------------------------|---|
| Scalability | Low data transfer rate |
| Covers large distances | Custom LoRa gateway |
| Low power consumption | Not suitable for real-time applications |
| Operates on unlicensed frequency | |

IoT Data protocols

- ▶ This application layer protocol allows systems to communicate with each other regardless of the platform being used. They provide communication with hardware on the user side—without the need for any internet connection.
- ▶ The connectivity in IoT data protocols and standards is through a wired or cellular network.

Advanced Message Queuing Protocol (AMQP)

Message Queuing Telemetry Transport (MQTT)

Constrained Application Protocol (CoAP)

Simple Object Access Protocol (SOAP)

Hypertext Transfer Protocol (HTTP)

Advanced Message Queuing Protocol (AMQP)

- ▶ AMQP guarantees secure and reliable communication, even in cases where the network is weak or one of the systems is temporarily inaccessible. It specifies "**forwarding addresses**" that messages can be routed to in the event of a connection failure.
- ▶ This protocol is popular in settings with server-based analytical environments, such as banking technology solutions. Otherwise, its application is rather limited due to its heaviness.

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|---|--|
| Reliability | Heavy memory requirements |
| Security | Slow data transmission due to large message size |
| Support for different messaging patterns (publish-subscribe, store-and-forward, and classic messaging queues) | |
| Extendibility with minimal effort | |

Message Queuing Telemetry Transport (MQTT)

- ▶ This lightweight IoT communication protocol ensures a reliable connection and operates on top of **TCP/IP** networks.
- ▶ MQTT is suitable for wireless systems with constrained bandwidth and high latency, such as mobile devices operating on an unreliable network. That's why Facebook used it for its online chats.
- ▶ Another application is remote monitoring, as MQTT shines at gathering small messages from devices with limited capacity at remote locations.

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|--|---|
| Low power consumption | Limited interoperability between devices from different vendors |
| Low bandwidth usage | Inherent security constraints (relies on short user names and passwords for authentication) |
| Ability to function well with unreliable connections | Poor extendibility |
| Data encryption by default | Limited choice of messaging patterns (only publish-subscribe) |

Constrained Application Protocol (CoAP)

- ▶ The Internet Engineering Task Force designed this IoT communication protocol to address the needs of **HTTP-based systems**.
- ▶ CoAP allows for seamless integration with HTTP, minimizing any additional overhead.
- ▶ It achieves this by supporting short wake-up and long sleep states. It enables HTTP clients to exchange information even with limited resources.
- ▶ The IoT protocol is widely used in building automation and smart energy applications.
- ▶ CoAP depends on User Datagram Protocol (UDP) packets for communication and message passing.
- ▶ This technology is utilized for **machine-to-machine** applications and allows devices with limited capacity, like low availability, to join the IoT environment.
- ▶ It can even work with microcontrollers with **only 10 KiB of RAM**.

Constrained Application Protocol (CoAP)

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|--|---|
| Highly secure as it uses DTLS parameters by default | Messages can reach a destination in the wrong order, which is a common issue with UDP |
| Easy to deploy | Difficulties communicating with devices behind Network Address Translation (NAT), which can generate dynamic IP addresses |
| Works efficiently with devices that have limited capabilities. | |

Data Distribution Service (DDS)

- ▶ Object Management Group (OMG) developed this IoT communication protocol for real-time systems.
- ▶ DDS offers a reliable and scalable data exchange using a **publish-subscribe pattern**.
- ▶ Its scalability is attributed to the fact that DDS supports the dynamic discovery of publishers and subscribers.
- ▶ It works well with the cloud and low-footprint devices and provides interoperable data sharing, which is software and hardware-independent.
- ▶ This protocol is believed to be the first open international middleware IoT standard.

| <u>Advantages:</u> | <u>Disadvantages:</u> |
|--|---|
| Scalable | Heavy on bandwidth (consumes twice as much traffic as MQTT) |
| Highly secure and powerful QoS mechanism | Only interfaces with web services via a gateway |
| Guarantees low-latency communication | |
| Connects devices from different vendors | |

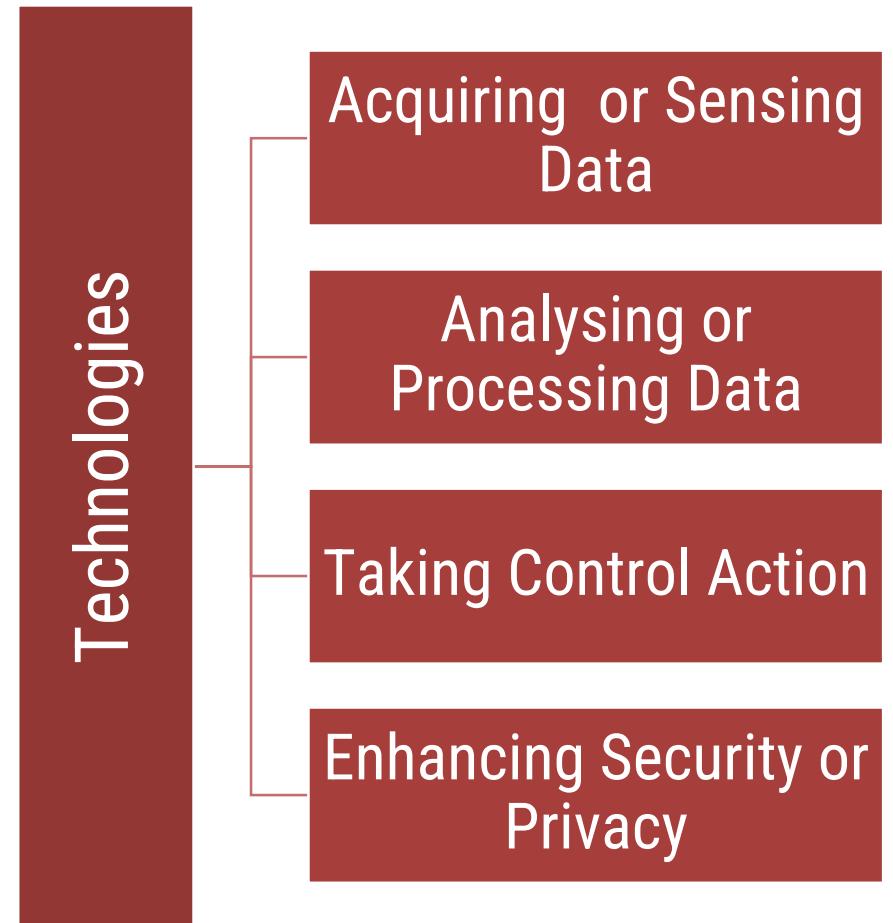


Enabling Technologies

Section - 6

Enabling Technologies

- ▶ IoT is a collection or group of many technologies and devices.
- ▶ The simplest of sensors, embedded systems, data analytics, communication protocols, security aspects and cloud computing with storage have all become enabling technologies.
- ▶ Enabling technologies/devices fall under one of the following categories:



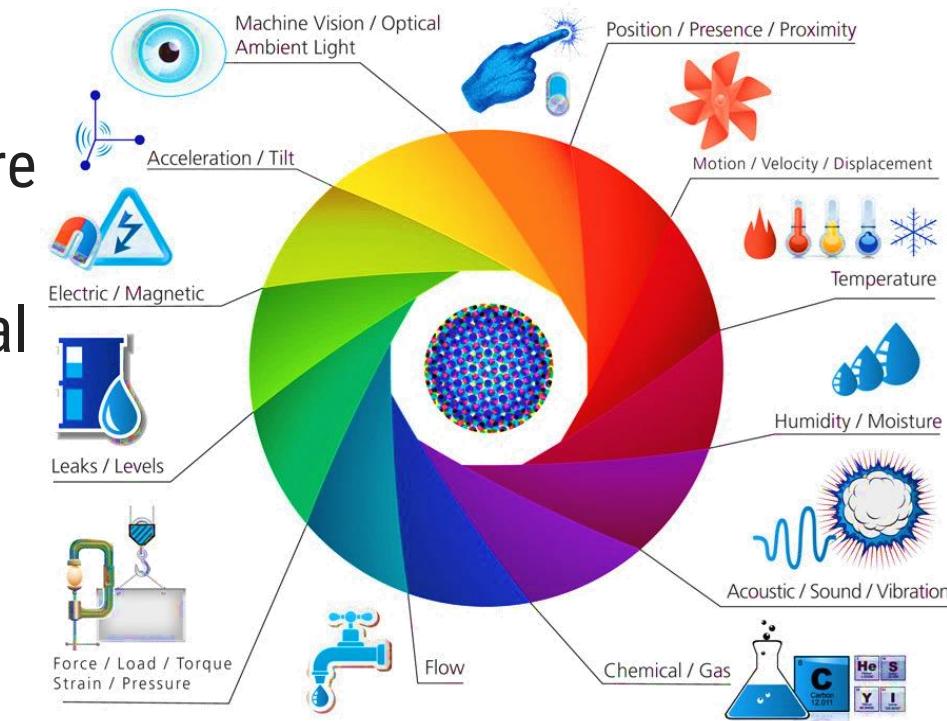
Enabling Technologies - **Sensors**

- ▶ Sensors are at the heart of any IoT application.
- ▶ As the name suggests, they sense the environment and retrieve data.
- ▶ Sensors are the starting point of any IoT application.
- ▶ It fetches data for us to operate on.
- ▶ Sensors could be **Analog or Digital**.
- ▶ Temperature sensor in a thermometer is an example of it.
 - It is used to build temperature monitoring



Enabling Technologies - Sensors

- ▶ Some examples of sensors that could be regarded as enabling technologies are as follows.
- ▶ **Weather tracking** system uses temperature/humidity/moisture sensors.
- ▶ Vehicle health monitoring sensors keep track of speed, tyre pressure, etc.
- ▶ On Board Diagnostics (OBDs) used for collecting all critical information from an automobile to detect error.
- ▶ **Vibration** sensors are used to track the quality of buildings/structures.
- ▶ **Water quality** is monitored through sensors that measure pH, chloride level, etc.



Enabling Technologies - Cloud Computing

- The next technology that is highly significant in IoT is **Cloud Computing**.
 - Cloud has grown much more popular because it serves as an affordable, effective and efficient medium for data storage.
 - Data storage plays a major role in IoT.

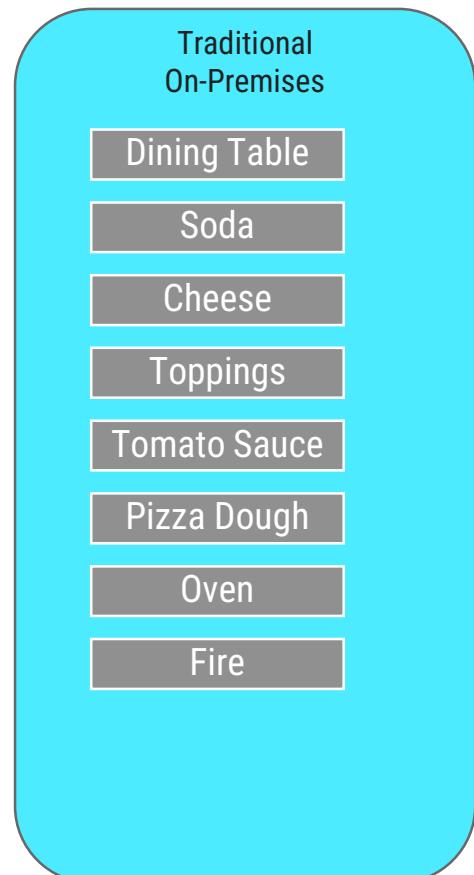


Service Models of Cloud Computing

- ▶ There are three main service models of cloud computing.
 1. Infrastructure as a Service (IaaS)
 2. Platform as a Service (PaaS)
 3. Software as a Service (SaaS).
- ▶ Each model represents a different part of the cloud computing stack.
- ▶ Each type of cloud service provides

Example of “as a Service”

Pizza as a Service



Made at Home



Take and Back



Pizza Delivered

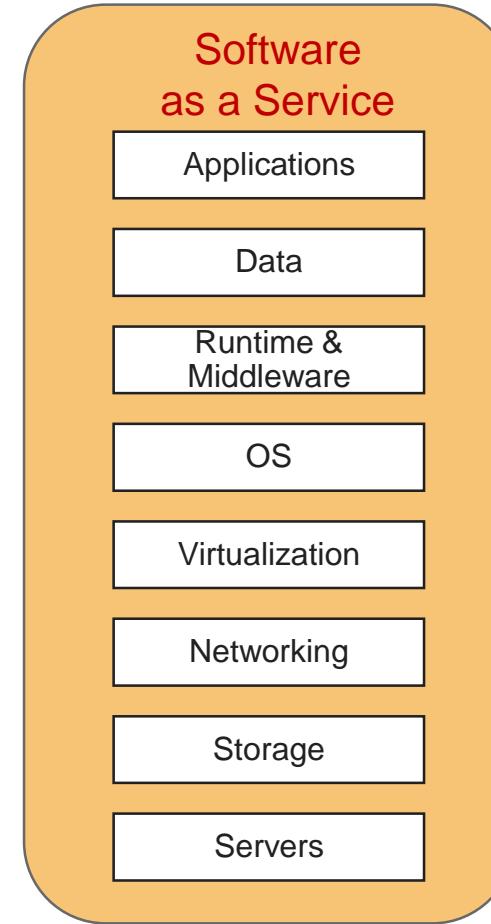
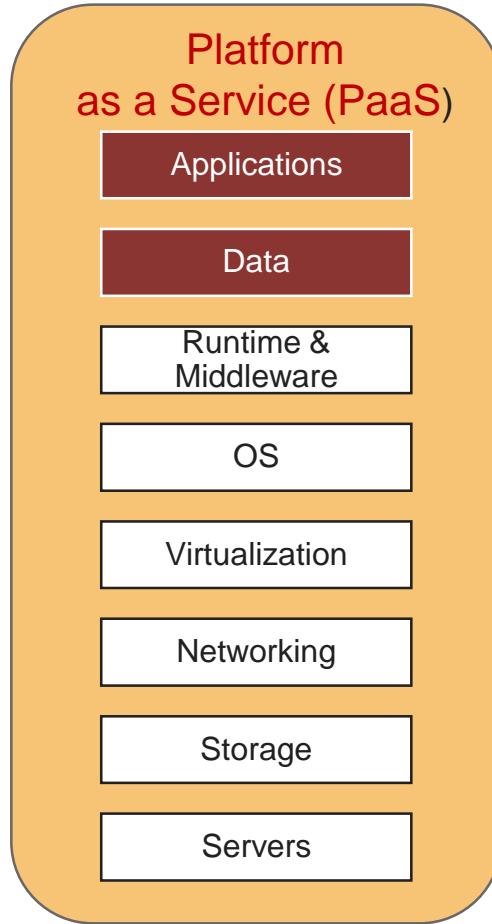
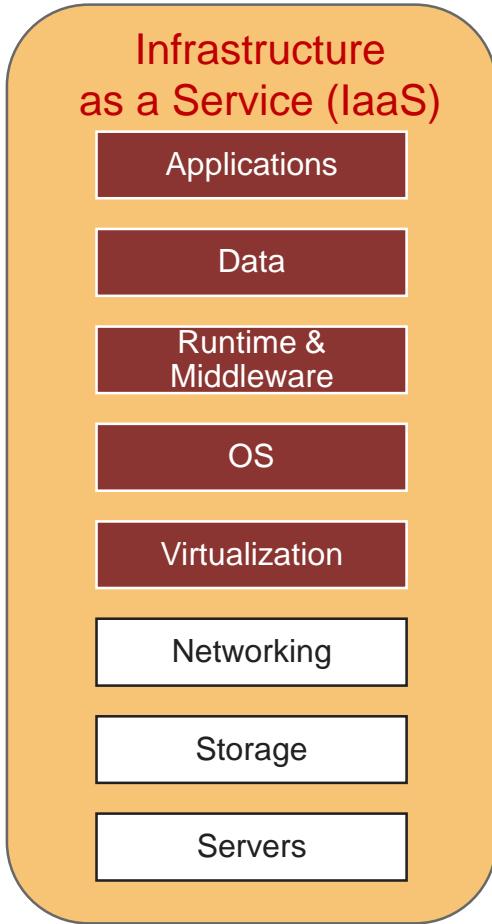
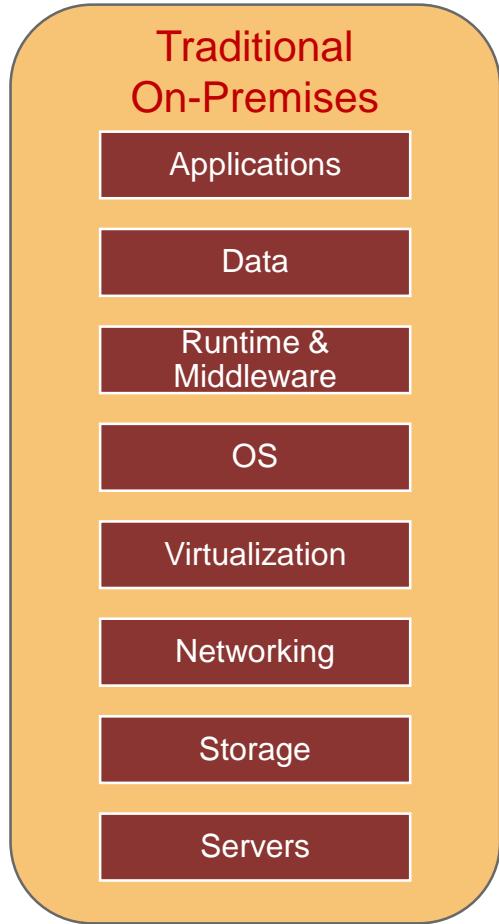


Dined at Restaurant

You Manage

Vendor Manage

Cloud Computing Services



You Manage

Vendor Manage

Clouds example

Examples of IaaS:



Google
Compute
Engine

Examples of PaaS:



App Engine

Examples of SaaS:



Google Apps

Enabling Technologies - Big Data Analytics

- ▶ Data is everywhere, and from every function or operation we get more data.
- ▶ IoT is all about collecting data from various sensory nodes.
- ▶ Handling the huge data is fundamental to make the application a success.
- ▶ The biggest challenge with big data is 4Vs. Volume, Variety, Speed (Velocity) at which it comes and its Veracity



Enabling Technologies - Big Data Analytics

Scale (Volume):

- Huge volume of data is generated every minute.
- Storage has become inexpensive and hence, cost-related challenges have reduced.
- Cloud Storage availability.



Data in doubt (Veracity):

- How accurate is all this data anyway? Because we are now rely on it
- The data's nature alters dynamically and uncertainty is often seen.
- So, it would be challenging to process this unstable data.

Complexity (Variety):

- Data no longer comes from one single source.
- It also comes in different formats (e.g., audio, video, text and image)
- Varieties of data becomes a huge challenge.

Speed (Velocity):

- The rate at which data is generated Very Fast.
- Also, data dynamics changes very frequently, new data is generated and moves around.

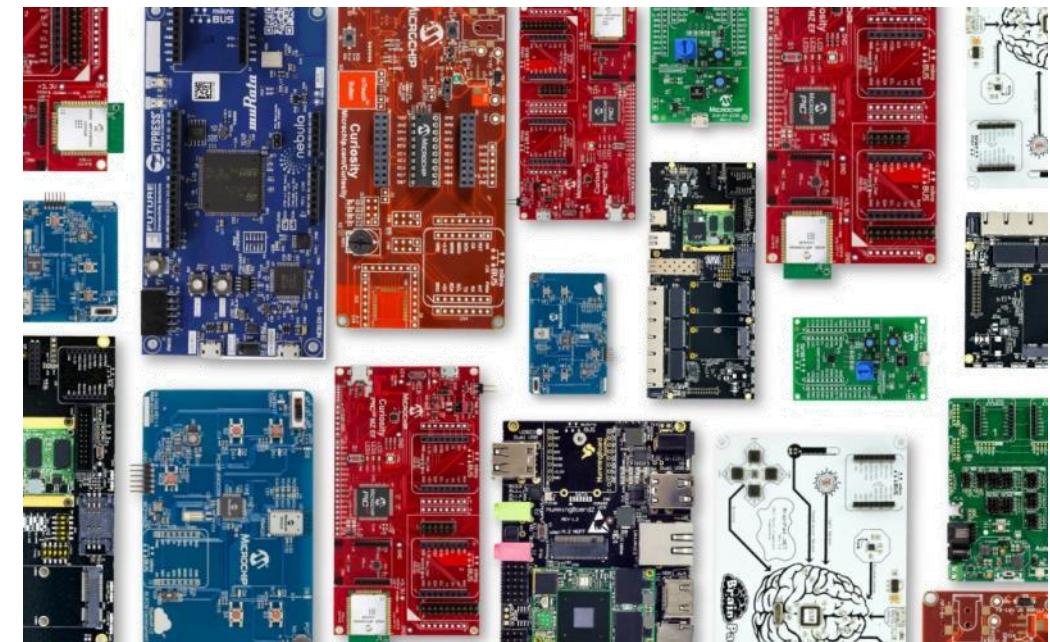
Enabling Technologies - **Big Data Analytics**

- ▶ In IoT data is everything.
- ▶ Data analytics is one of the enabling technologies for building a complete IoT application.
- ▶ So the question is: “Who is generating all this data?”
- ▶ A partial list to answer this is as follows:
 - Sensors from security systems.
 - Sensors from weather monitoring systems.
 - Sensors from car/navigation systems.
 - Sensors from water quality monitoring systems.
 - Data from wearable Devices (e.g., bands).
 - Data from industrial equipment (e.g., motor health).
 - Sensors from bridges/roads about traffic density and other factors.
 - Social media (e.g., tweets, photo uploads, etc.).



Enabling Technologies - **Embedded Computing Boards**

- ▶ An embedded computing board a very important component to bring IoT design to reality.
- ▶ For making the prototype the computing boards play vital role.
- ▶ The computing boards available in the market are driven by **microcontrollers or processors**.
- ▶ Some of the boards are as follows:
 - Raspberry Pi.
 - Arduino (many variants).
 - NodeMCU.
 - Intel Edison.
 - Dragon Board
- ▶ All these boards are small, yet smart.



IoT Level

Section - 7

IoT System Architectural Level

- ▶ Based on the architectural approach, IoT can be classified in five levels: **Level 1 to Level 5**.

□ **Level 1**

- ▶ It is of minimal complexity.
- ▶ The application has one sensor (temperature sensor, pressure sensor, etc).
- ▶ The data sensed is **stored locally and the data analysis is done locally**.
- ▶ Monitoring / control is done through an application.
- ▶ This is used for simple applications.
- ▶ Data generated in this level application is not huge.
- ▶ For example, a temperature sensor senses the room temperature and the data is stored and analysed locally.



Home Automation



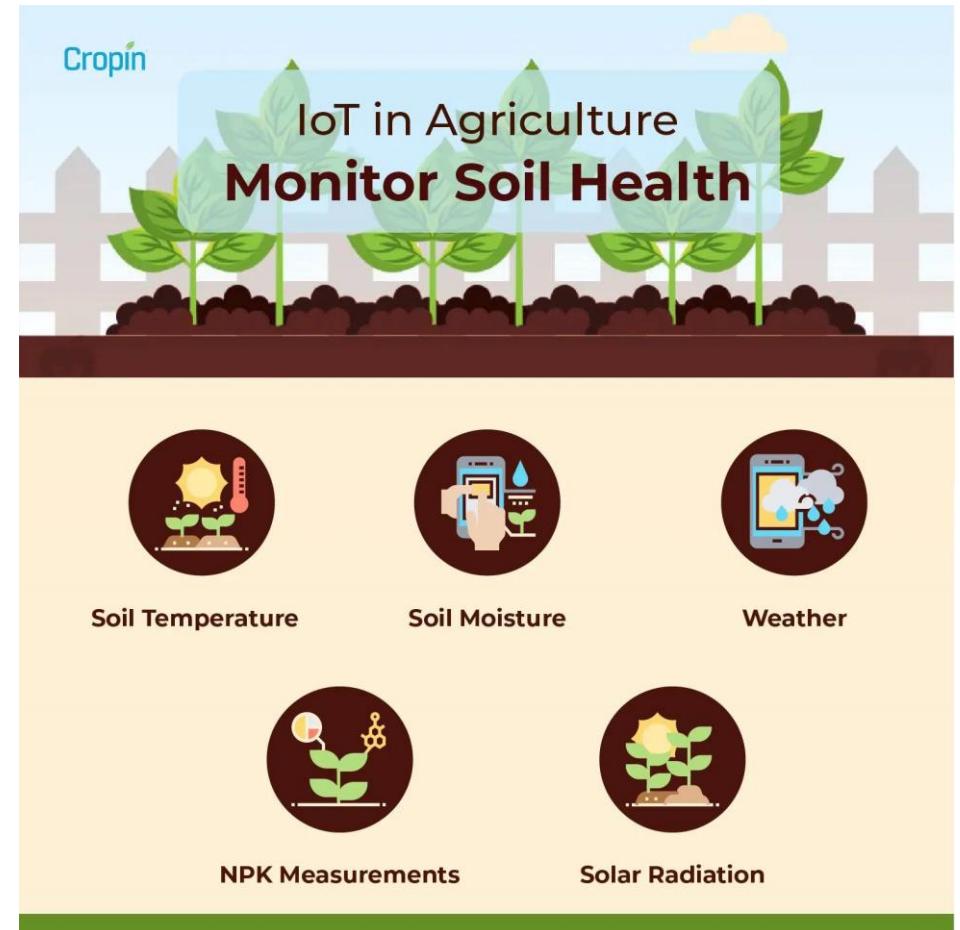
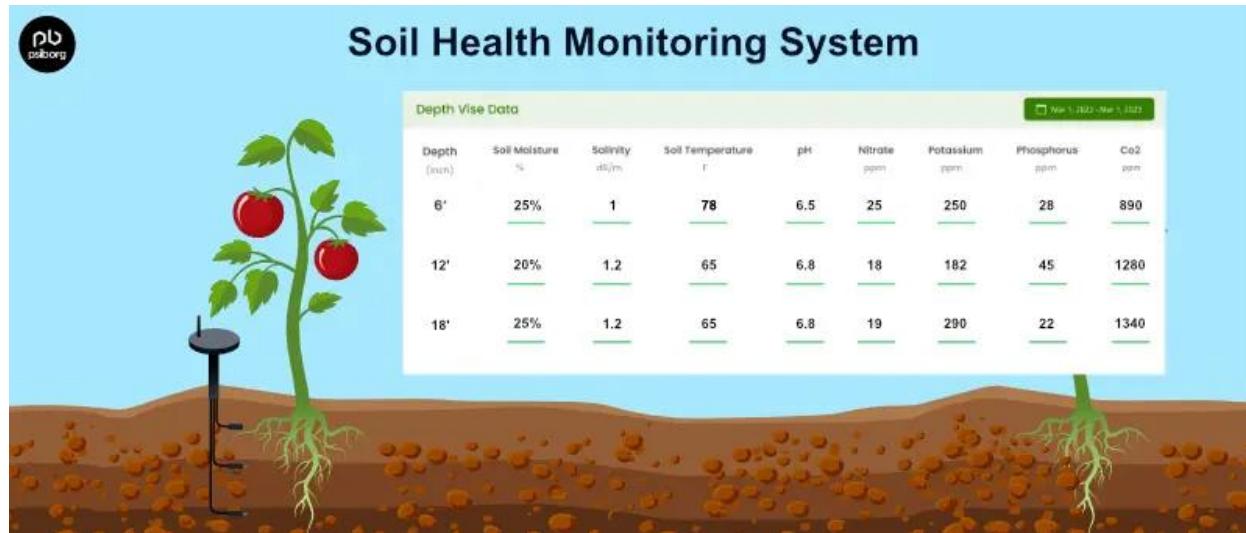
IoT System Architectural Level

□ Level 2

- ▶ The second level is slightly more complex than the previous level.
- ▶ The data is more voluminous and hence, **cloud storage** is preferred.
- ▶ The frequency of sensing done by the sensor is faster.
- ▶ The number of times sensing is done would be much more than Level 1.
- ▶ **The analysis is carried out locally**, while cloud is meant for storage only.
- ▶ Based on the data analysis, the control action can be triggered through the web application or mobile application.
- ▶ Some examples are agriculture applications, room freshening solutions based on odor, etc.



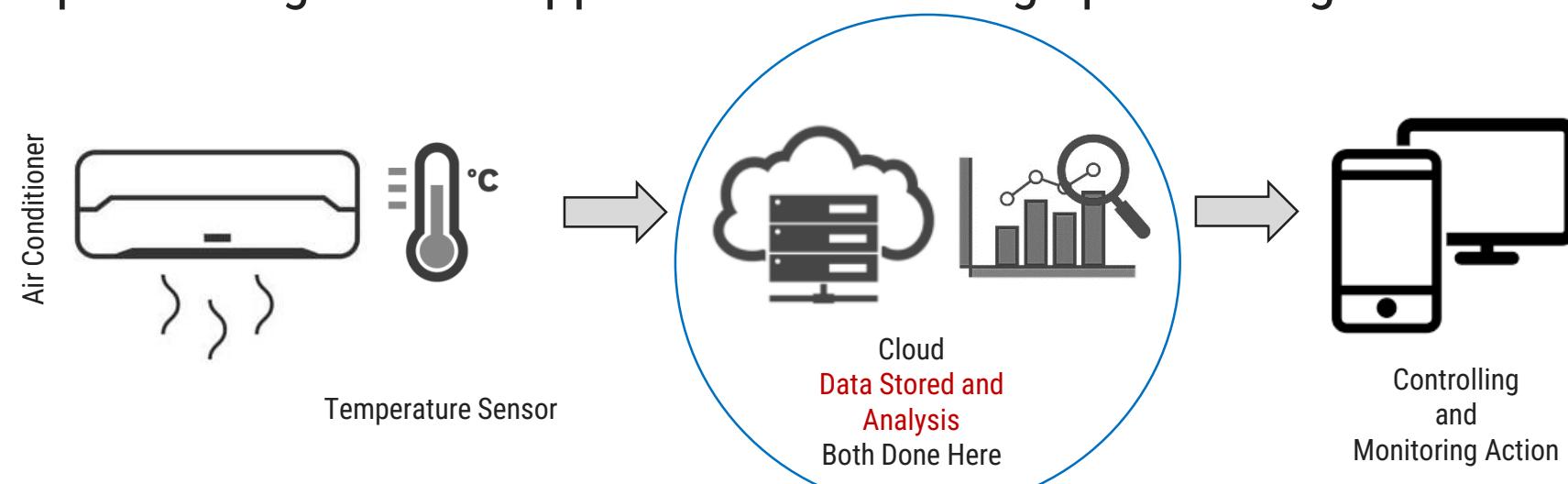
Smart Agriculture – Soil health monitor



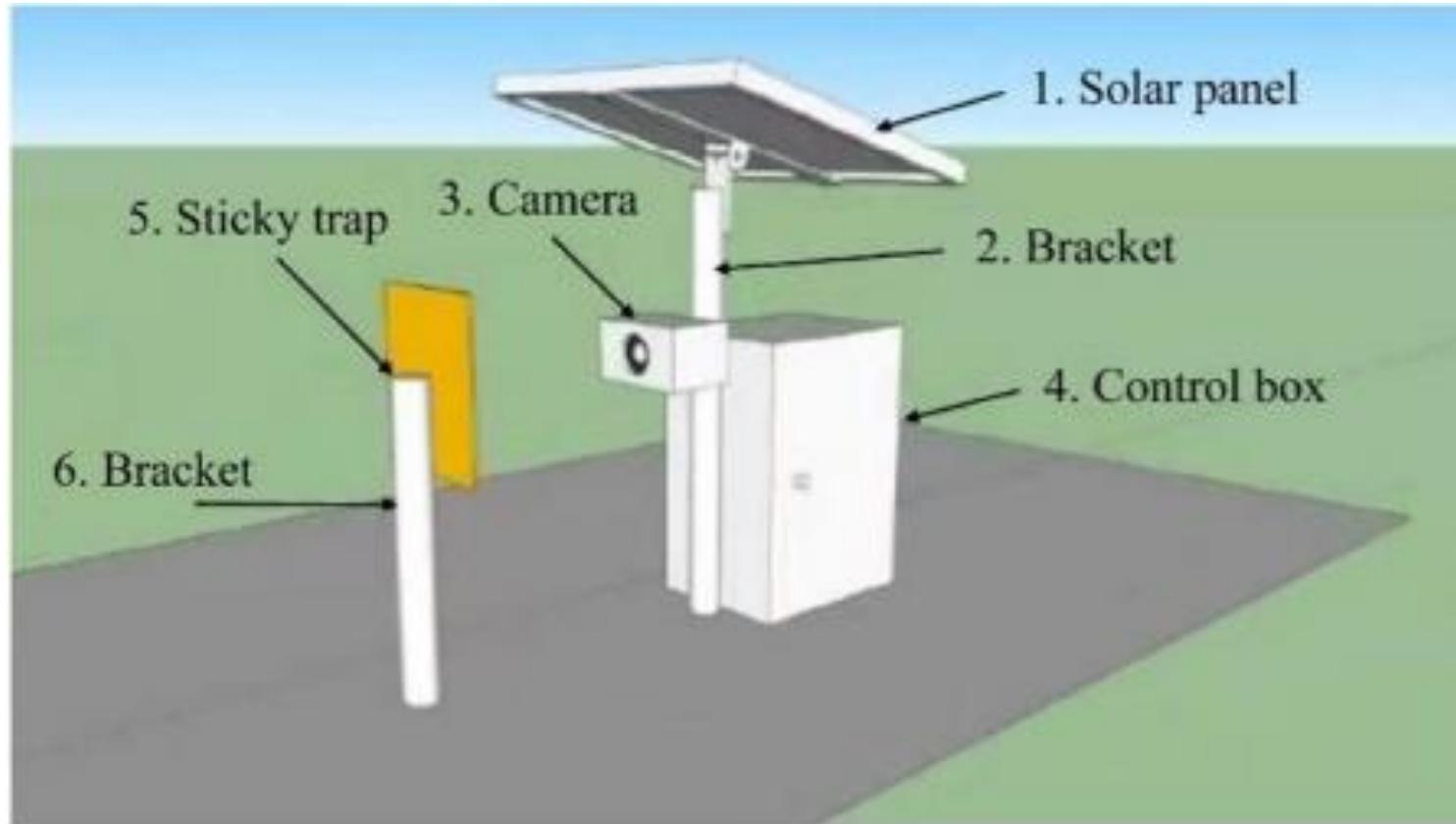
IoT System Architectural Level

□ Level 3

- ▶ The data is huge, frequency of sensing done by the sensor is faster and the data is stored on cloud.
- ▶ The difference is that the **analysis is also carried out on cloud**.
- ▶ Based on the data analysis, the control action can be triggered through the web application or mobile application.
- ▶ Some examples are agriculture applications with Image processing.



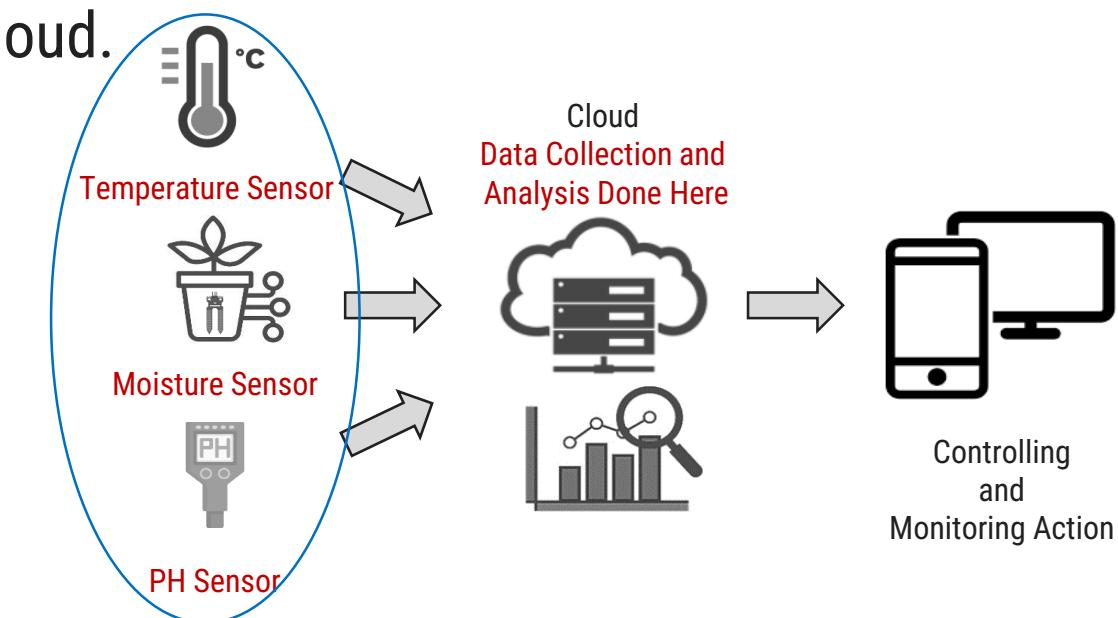
Smart Agriculture – Insect detection



IoT System Architectural Level

□ Level 4

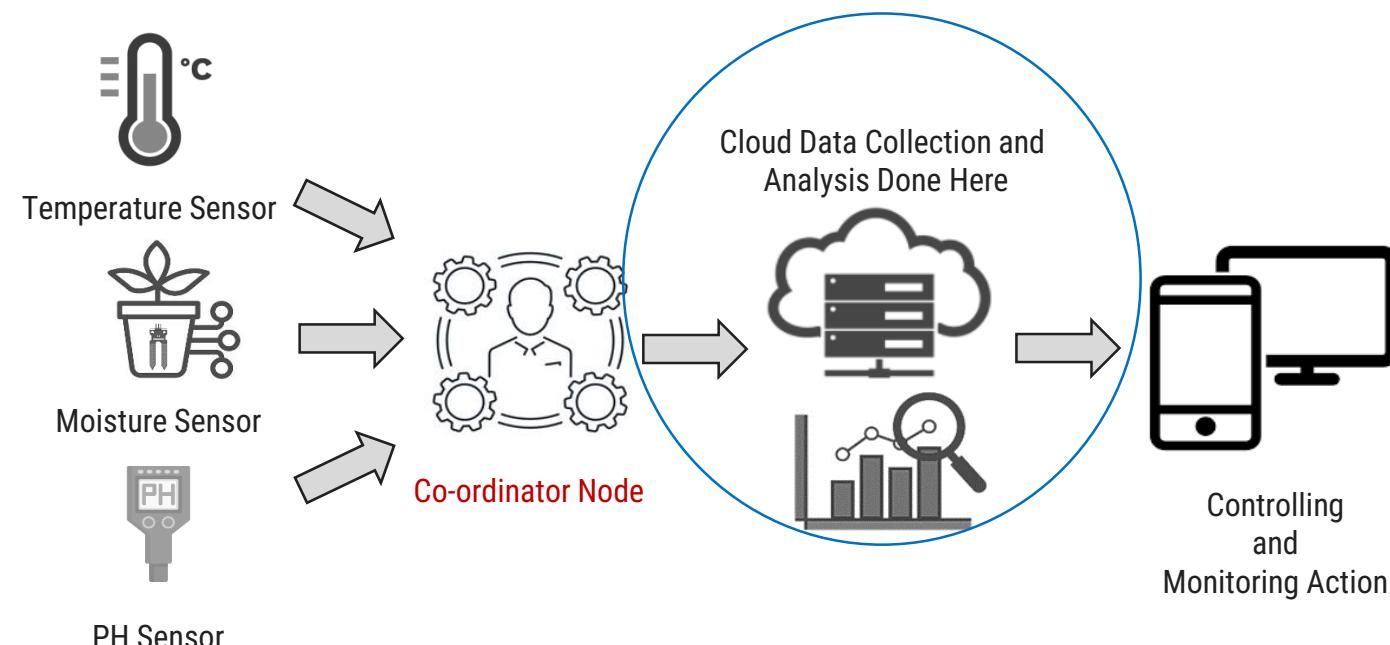
- ▶ With every passing level, the volume of data increases and hence the rate at which it is sensed also increases.
- ▶ At this level, **multiple nodes are present which are independent of each other.**
- ▶ These nodes upload data to the cloud.
- ▶ All the sensors upload the read sensory inputs on cloud storage.
- ▶ Analysis is also carried out on the cloud.



IoT System Architectural Level

□ Level 5

- ▶ At this level, the **amount of data is extensive and is sensed much faster.**
- ▶ Multiple nodes are involved in the applications **categorized** under Level 5 and these nodes are independent of each other.
- ▶ The sensing of data and its storage is the same as in all the previous levels.
- ▶ When an application is completely cloud oriented, it is computationally intensive in real time.





IoT Applications

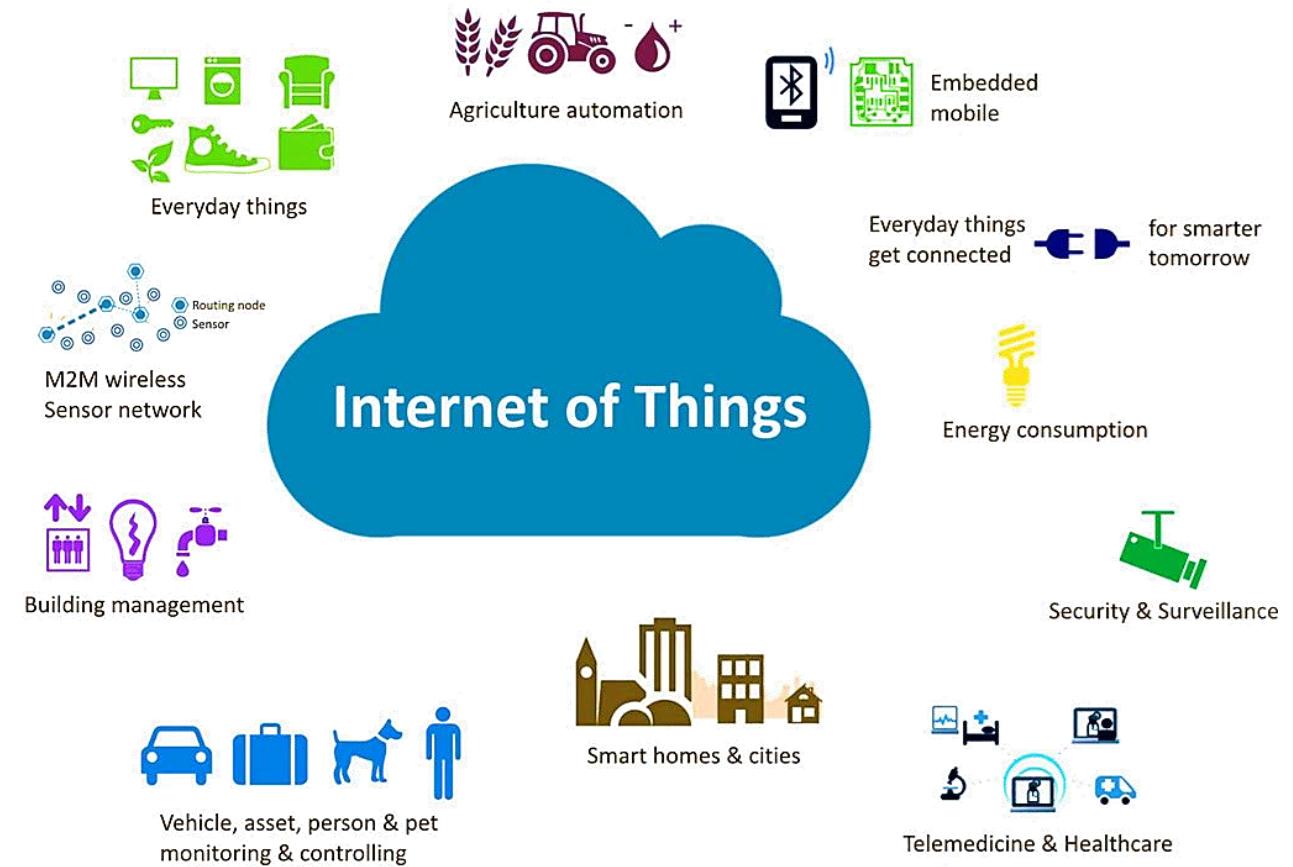
Section - 8

Application areas of IoT

► The scope and application areas of IoT is very huge.

► IoT can be used to build applications

- Agriculture
- Assets Tracking
- Energy Sector
- Defence
- Embedded Applications
- Education
- Waste Management
- Healthcare Products
- Telemedicine
- Safety And Security Sector
- Smart City Applications etc.



Smart Home Automation

- ▶ A smart home system can be something that makes our life quite easy.
- ▶ Starting from energy management where the power controls system in the AC appliances where we use the thermostat, all this is managed to cut down the power consumption that's taking place.
- ▶ A door management system, security management system, water management system are the part of this as well. Still, these are vital things that stand out in the smart home system.



Smart Home Automation

- ▶ The limitation of IoT in smart home application stops where our imagination stops. Anything that we wish to automate or want to make our life easier can be a part of smart home, a smartphone system as well.
- ▶ **Smart Lighting** – Smart lighting for home helps in saving energy by adapting the life to the ambient condition and switching on/off or dimming the light when needed. Smart lighting solutions for homes achieve energy saving by sensing the human movements and their environments and controlling the lights accordingly.
- ▶ **Smart Appliances** – Smart appliances with the management are here and also provide status information to the users remotely.
- ▶ Smart washer/dryer can be controlled remotely and notify when the washing and drying are complete. Smart refrigerators can keep track of the item store and send updates to the users when an item is low on stock.

- ▶ **Intrusion Detection** – Home intrusion detection systems use security cameras and sensors to detect intrusion and raise alerts.
- ▶ Alert can we inform of an SMS or an email sent to the user.
- ▶ Advanced systems can even send detailed alerts such as an image shoot or short video clips.
- ▶ **Smoke/gas detectors** – Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of Fire.
- ▶ It uses optical detection, ionization for Air sampling techniques to detect smoke.
- ▶ Gas detectors can detect the presence of harmful gases such as CO, LPG, etc.
- ▶ It can raise alerts in the human voice describing where the problem is.

Smart Agriculture

- ▶ Smart agriculture, on the other hand, is mostly used to denote the application of IoT solutions in agriculture.
- ▶ By using IoT sensors to collect environmental and machine metrics, farmers can make informed decisions, and improve just about every aspect of their work – from livestock to crop farming.
- ▶ For example, by using smart agriculture sensors to monitor the state of crops, farmers can define exactly how many pesticides and fertilizers they have to use to reach optimal efficiency. The same applies to the smart farming definition.



IoT in agriculture

- ▶ **1. Monitoring of climate conditions :** Probably the most popular smart agriculture gadgets are weather stations, combining various smart farming sensors. Located across the field, they collect various data from the environment and send it to the cloud.
- ▶ The provided measurements can be used to map the climate conditions, choose the appropriate crops, and take the required measures to improve their capacity (i.e. precision farming).

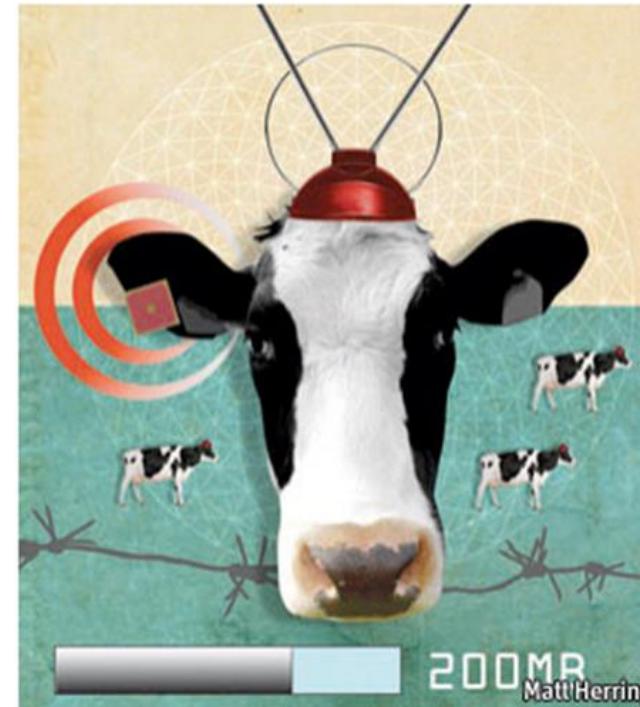


IoT in agriculture

- ▶ **2. Greenhouse automation :** Typically, farmers use manual intervention to control the greenhouse environment. The use of IoT sensors enables them to get accurate real-time information on greenhouse conditions such as lighting, temperature, soil condition, and humidity.
- ▶ In addition to sourcing environmental data, weather stations can automatically adjust the conditions to match the given parameters. Specifically, greenhouse automation systems use a similar principle.
- ▶ **3. Crop management :** One more type of IoT product in agriculture and another element of precision farming are crop management devices. Just like weather stations, they should be placed in the field to collect data specific to crop farming; from temperature and precipitation to leaf water potential and overall crop health.
- ▶ Thus, you can monitor your crop growth and any anomalies to effectively prevent any diseases or infestations that can harm your yield.

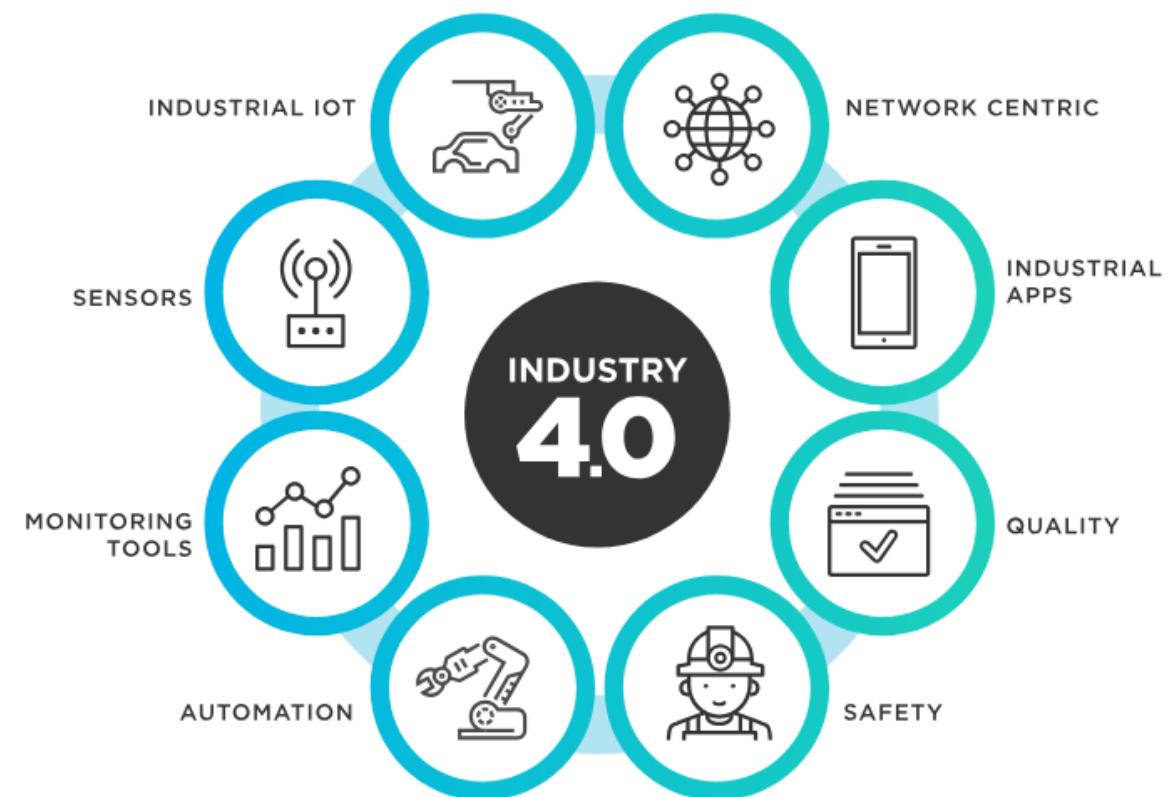
IoT in agriculture

- ▶ **4. Cattle monitoring and management :** There are IoT agriculture sensors that can be attached to the animals on a farm to monitor their health and log performance. Livestock tracking and monitoring help collect data on stock health, well-being, and physical location.
- ▶ For example, such sensors can identify sick animals so that farmers can separate them from the herd and avoid contamination. Using drones for real-time cattle tracking also helps farmers reduce staffing expenses. This works similarly to IoT devices for petcare.



IoT in Industry 4.0

- ▶ **Industrial IoT**, or the Industrial Internet of Things (IIoT), is a vital element of **Industry 4.0**.
- ▶ IIoT harnesses the power of smart machines and real-time analysis to make better use of the data that industrial machines have been churning out for years.
- The principal driver of IIoT is smart machines, for two reasons.
 1. The smart machines capture and analyze data in real-time, which humans cannot.
 2. The smart machines communicate their findings in a manner that is simple and fast, enabling faster and more accurate business decisions.



Applications of IIoT

- ▶ **Production :** Smart machines, enabled with IIoT, can self-monitor and anticipate possible production hurdles.
 - This results in lowered downtime and better efficiency.
- ▶ **Supply Chain : With IIoT, orders can automatically replenish stocks when needed.**
 - This reduces waste, maintains stock numbers, and makes sure the right amount of raw materials are always available.
 - With the automation of supply chains and ordering, employees can focus on more complex areas of functioning.
- ▶ **Building Management : Most building management issues can be addressed with IIoT technology.**
 - Sensor-driven climate control removes all the uncertainty related to managing a building's internal climate and takes all needed factors into consideration
 - such as the number of people, ventilation spots, machinery, and more.
 - IIoT enhances building security with smart devices that assess possible threats from any entry points of a building.

Applications of IIoT

► **Healthcare** : Healthcare has embraced smart devices for a long time now. Healthcare professionals can remotely monitor patients and are alerted by any status change. This makes healthcare more precise and personal. In the future, artificial intelligence may be able to assist with diagnoses, enabling doctors to treat patients more accurately and effectively.

Applications of IIoT

- ▶ **Retail** : IIoT technology in retail enables quick marketing decisions specific to each store. Companies can update storefronts based on region-specific consumer interests, and they can target audiences with smarter promotions. These data-driven insights make a store stand out from its competition.
- Sensors are not new technology as companies have used them to track goods or monitor machines for years.
- The difference in IIoT is the ability to adopt these changes on a larger scale due to the lowered costs of sensors, comprehensive wiring networks, and big-data analytics.

Summary

- ▶ IoT refers to the interconnection of computing devices embedded in everyday objects via the Internet, enabling them to send and receive data.
- ▶ IoT is not owned by any one engineering branch. It is a reality when multiple domains join forces and combine efforts.
- ▶ IoT is all about providing service to any device, anywhere, anybody, and any network.
- ▶ IoT has certain characteristics which are important:
 - ❖ Connectivity
 - ❖ Intelligence and identity
 - ❖ Scalability.
 - ❖ Dynamic and self-adapting (complexity)
 - ❖ Architecture
 - ❖ Safety

Summary

- ▶ “Things” refer to variety of devices. At times, even humans in the loop becomes a thing.
 - For anything to qualify as a “thing”, it requires identity. The “thing” can monitor, measure, etc.
 - Example, a temperature sensor could be a “thing”.
- ▶ One should understand that “**THINGS**” = **HARDWARE + SOFTWARE + DATA + SERVICE**
- ▶ IoT stack has seven layers, starting with sensor layer and ending with application layer just as OSI.
- ▶ Security/personnel safety, privacy, data extraction with consistency from complex environments, connectivity, power requirements, complexity involved and storage are the major challenges we face while building an IoT application.
- ▶ IoT application can be classified as Level 1, 2, 3, 4 or 5 based on the complexity and architecture involved.
- ▶ IoT is all about sense, connect, store, analyse data.

**Thank
You**



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot

✉ bhushan.joshi@darshan.ac.in
☎ +91 8485979997



Unit-2

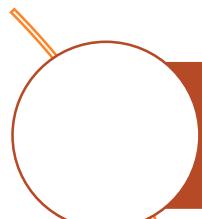
IoT Physical Devices and Endpoints



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot
 bhushan.joshi@darshan.ac.in
 +91 8485979997



Unit 2-A: IoT Physical Devices and Endpoints



IoT and M2M



Machine to Machine Communication



Software Define Network



IoT Data Acquisition

IoT : Internet of Things

- **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.
- IoT is an ecosystem of connected physical object that are accessible through internet.
- IoT means anything which can be connected to internet and can be controlled or monitored using internet from smart devices or PC.

M2M : Machine To Machine

- Machine-to-Machine, describes a communication method in which two or more machines interact via **wireless or wired** connections **without human intervention**.
- M2M refers to the **direct communication between devices**.
- M2M is another mode of creating connection between machines to transfer information and data from one machine to another machine.
- M2M uses Point-to-Point connection method. It has varied use nowadays.
- M2M is an technology that helps the devices to connect between devices without using internet.
- **M2M** is only subset of IoT .



IoT vs M2M

| | IoT | M2M |
|-----------------------------|---|--|
| Abbreviation | Internet of Things | Machine to Machine |
| Intelligence | Devices have objects that are responsible for decision making | Some degree of intelligence is observed in this. |
| Connection type used | The connection is via Network and using various communication types. | The connection is a point to point |
| Communication protocol used | Internet protocols are used such as HTTP, FTP, and Telnet. | Traditional protocols and communication technology techniques are used |
| Data Sharing | Data is shared between other applications that are used to improve the end-user experience. | Data is shared with only the communicating parties. |

IoT vs M2M

| | IoT | M2M |
|-----------------------|---|--|
| Internet | Internet connection is required for communication | Devices are not dependent on the Internet. |
| Type of Communication | It supports cloud communication | It supports point-to-point communication. |
| Computer System | Involves the usage of both Hardware and Software. | Mostly hardware-based technology |
| Scope | A large number of devices yet scope is large. | Limited Scope for devices. |
| Business Type used | Business 2 Business(B2B) and Business 2 Consumer(B2C) | Business 2 Business (B2B) |
| Open API support | Supports Open API integrations. | There is no support for Open APIs |

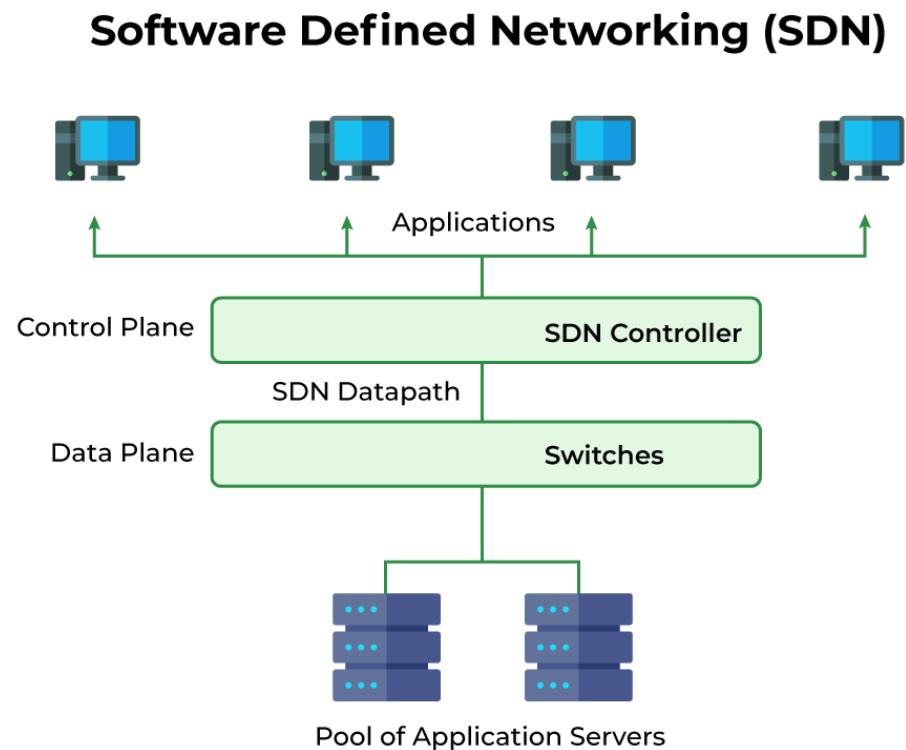


Software defined Network(SDN)

- SDN stands for **Software Defined Network**, which is a networking architecture approach.
- SDN enables the control and management of the network using software applications.
- Through Software Defined Network (SDN) networking behavior of the entire network and its devices are programmed in a centrally controlled manner through software applications using open APIs.
- To understand software-defined networks, we need to understand the various planes involved in networking.
 1. Data Plane
 2. Control Plane

- Data plane: All the activities involving as well as resulting from data packets sent by the end-user belong to this plane. This includes:
 - ❖ Forwarding of packets.
 - ❖ Segmentation and reassembly of data.
 - ❖ Replication of packets for multicasting.

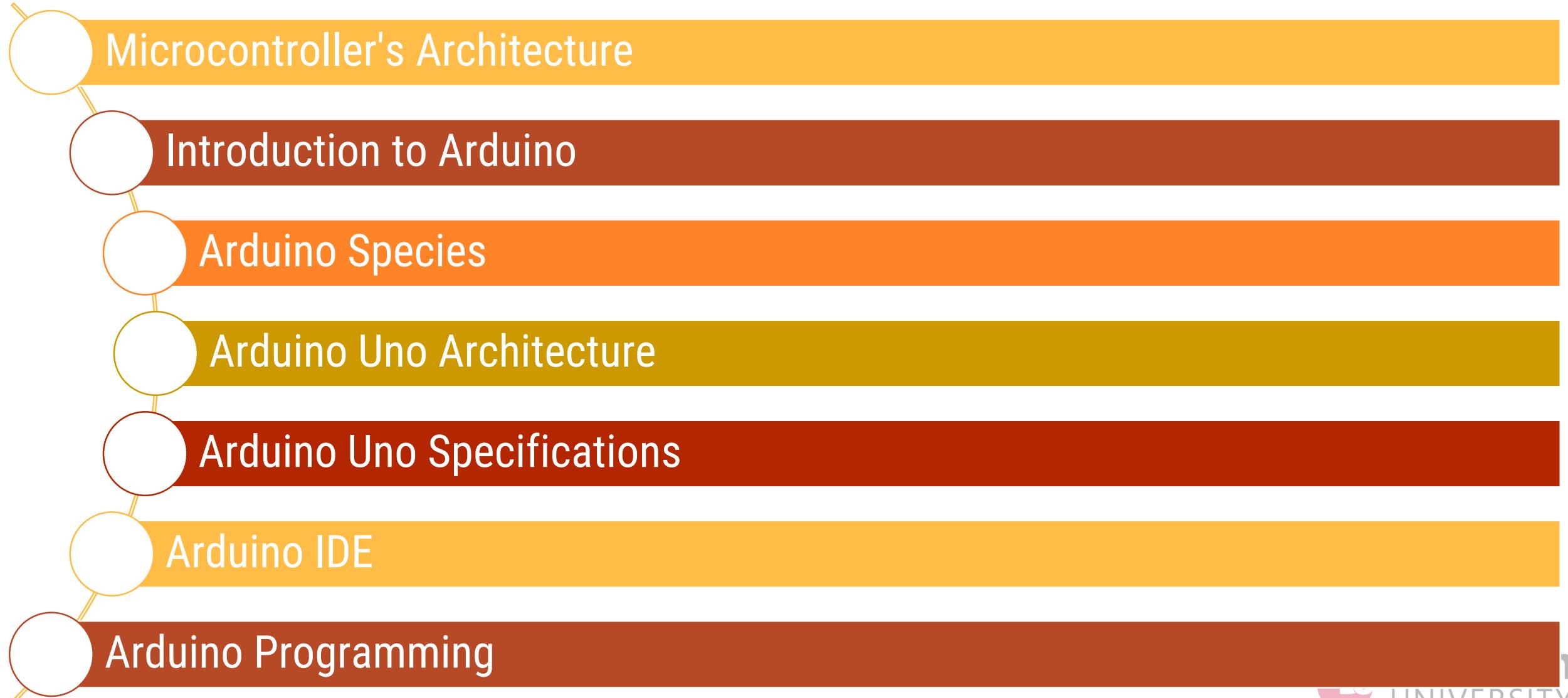
- Control plane: All activities necessary to perform data plane activities but do not involve end-user data packets belong to this plane.
- This is the brain of the network.
- The activities of the control plane include:
 - ❖ Making routing tables.
 - ❖ Setting packet handling policies.



IoT Data Acquisition System

- Data acquisition (DAQ) and protocols are pivotal building blocks of IoT technology.
- A data acquisition device helps users to make machines smarter by gathering and analyzing real-time data.
- After the acquisition of Data, Data should be converted in proper Protocol format.
- IoT protocols enable to exchange data in an organized and significant manner.
- Data Acquisition played two critical roles:
 1. **Identity** : Identify the data (Sensor and Data Type)
 2. **Authentication** : (Authenticity of Data)

Unit 2-B: Arduino



Examples of 8-bit µC

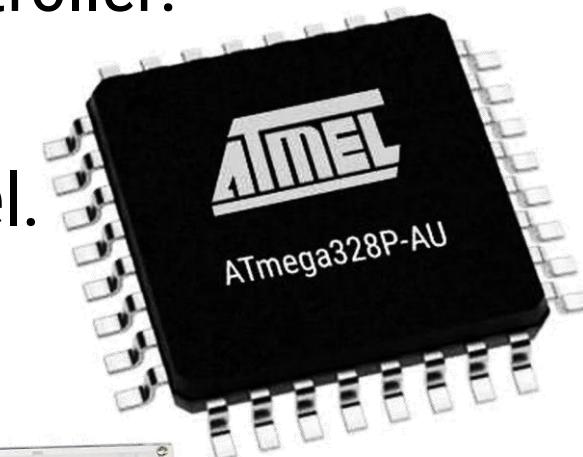
- Motorola's 6811
- Intel's 8051
- Zilog's Z8
- Microchip's PIC

- The 8051 family has the largest number of diversified (multiple source) suppliers:
 - Intel (original)
 - ATMEL
 - Philips/Signetics
 - AMD
 - Infineon (formerly Siemens)
 - Dallas Semiconductor/Maxim

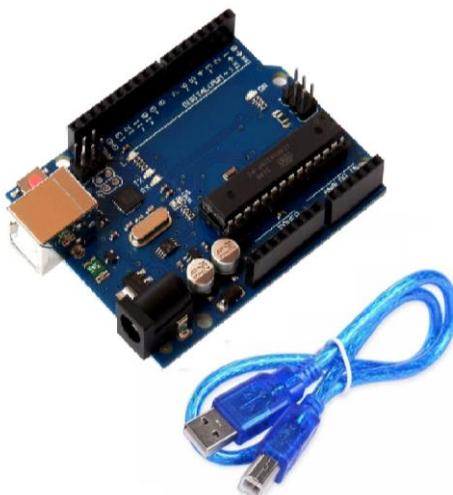


The Microcontroller inside Arduino UNO : Atmel ARV **ATmega328**

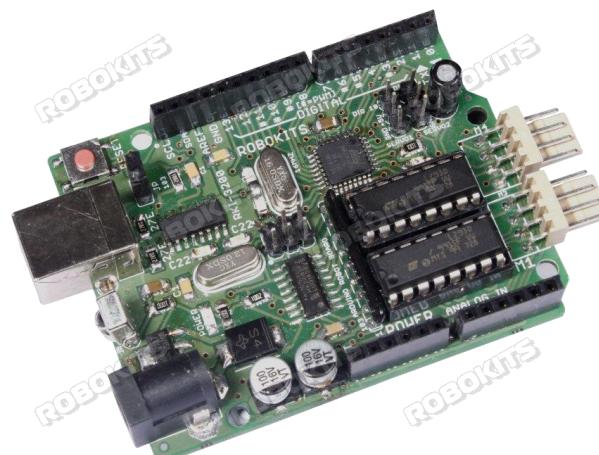
- The **Arduino Uno** is an open-source **microcontroller board**.
- This board is based on the Microchip **ATmega328P** microcontroller.
- The Board is developed by Arduino.cc.
- The Microcontroller inside the board is manufactured by Atmel.



Robu.in : ₹ 600.96

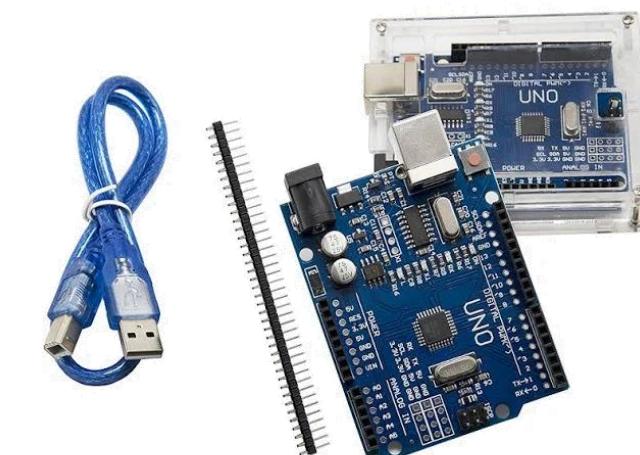


Robokits.co.in : ₹ 797



UNO R3 + MOTOR SHIELD

Amazon.in : ₹ 499



ATmega328P : Architecture



Memory And Instructions

**RISC
8 bit
Harvard**

Native data size 8 bits
(1 byte)

16-bit data addressing

Instruction



On-chip Memory

**3
Separate**

- **SRAM:** 2 KiB
8 bits
 - store data
- **Flash (ROM):** 32KiB
16 bits
 - program code
- **EEPROM :** 1KiB
16 bits
 - small data

On-Chip Memories



Communication Port

Port

- 1) Digital input/output
- 2) Analog input
- 3) Serial/Parallel
- 4) Pulse accumulator

Input- Output Port

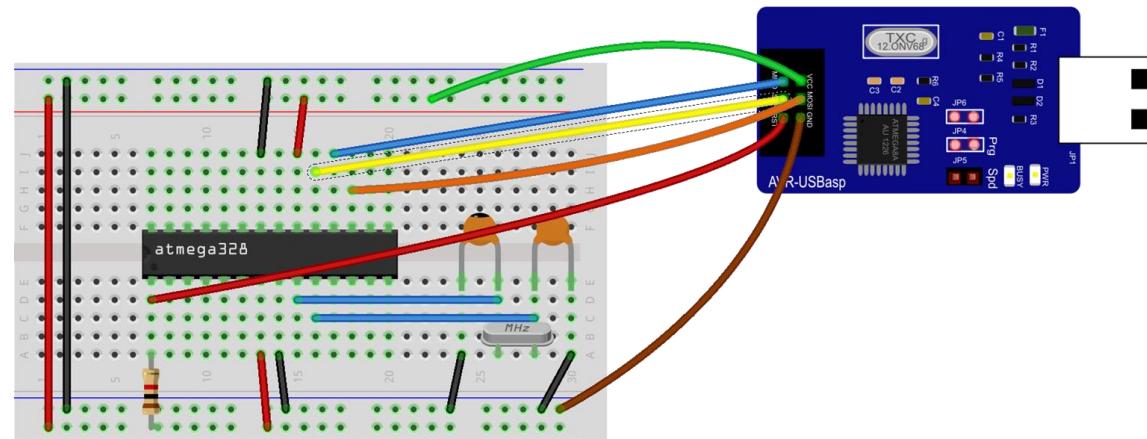


Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

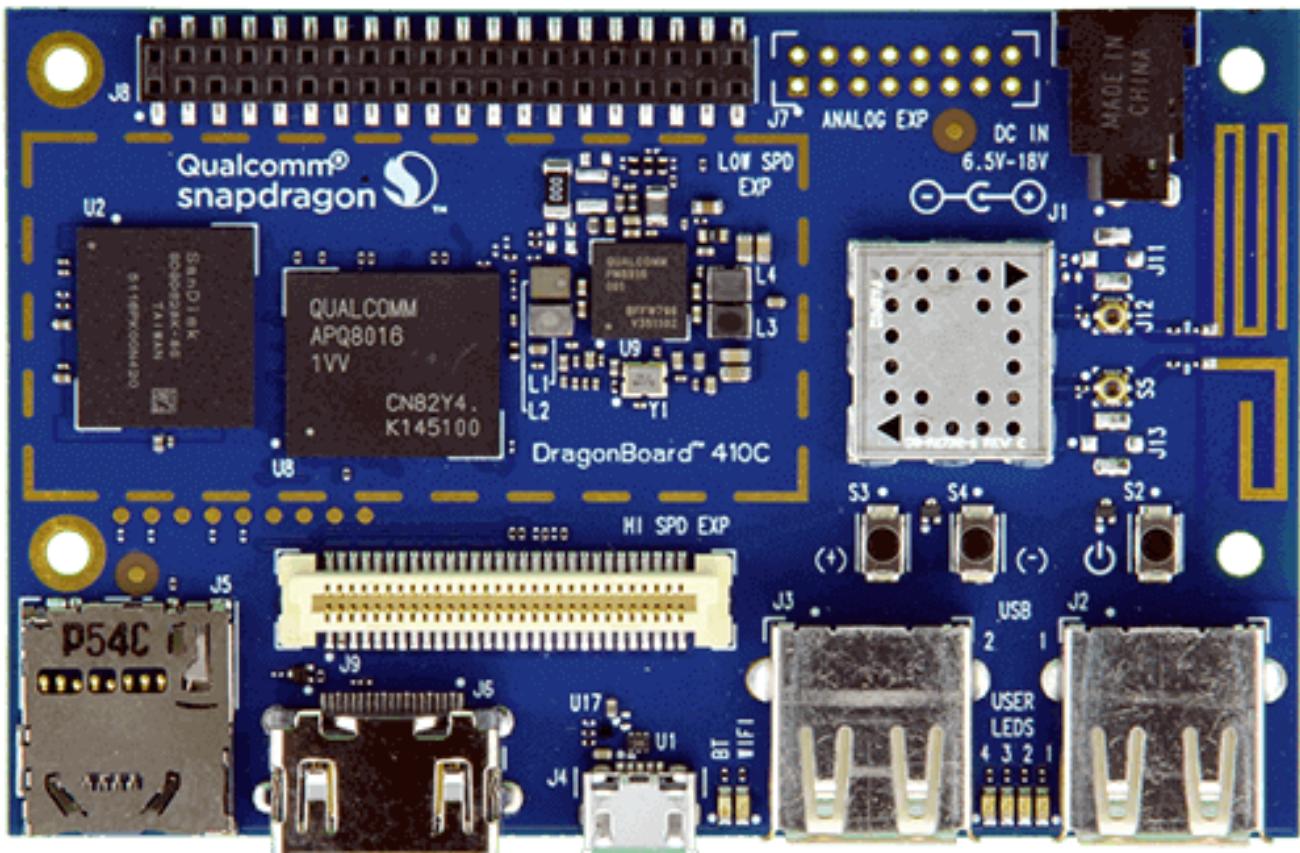
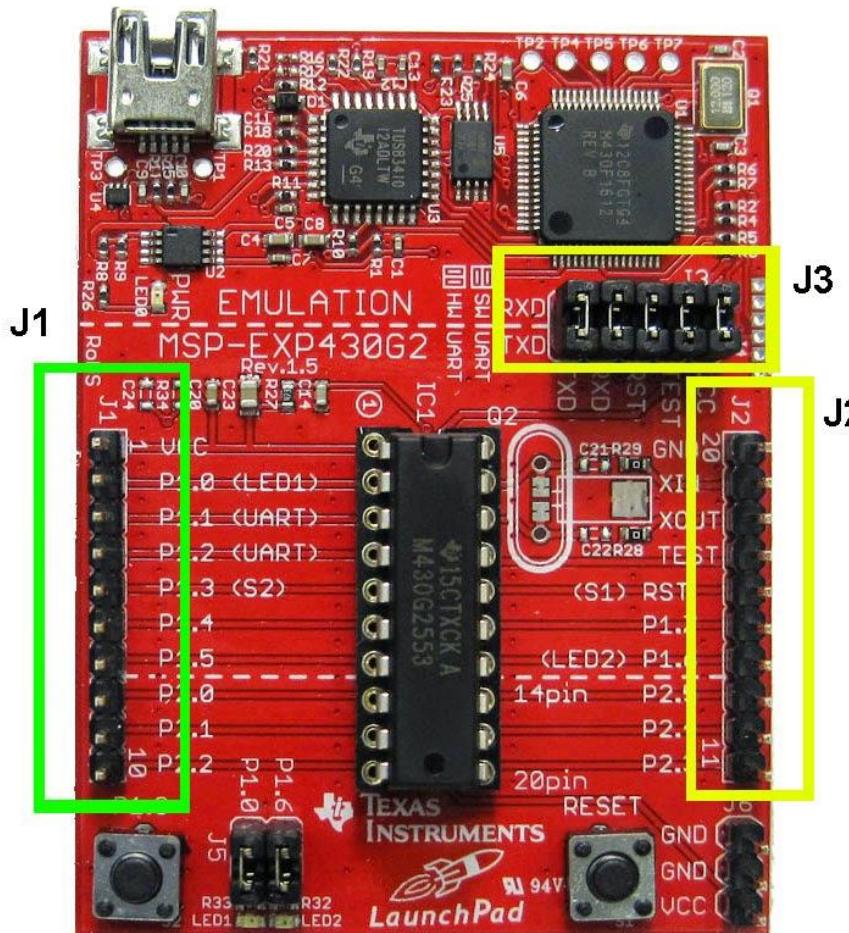
Why Development Board ?

- Microcontroller or Microprocessor has multiple Pins but doesn't have Port and other Peripheral devices.
- To Interface we need to design electronics circuit.
- Electronics Circuit design require specific skill set and knowledge.
- Development board provides this all ready made.
- Development board provides environment, where we can directly program controller or processor without worrying about circuit designing.



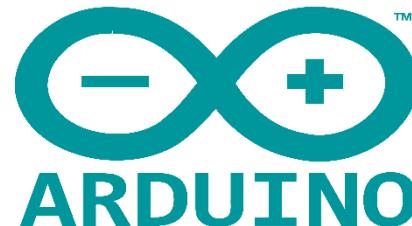
Development Board

- We are not going to Design Development Board.
- Various Development Board already available.



What is Arduino?

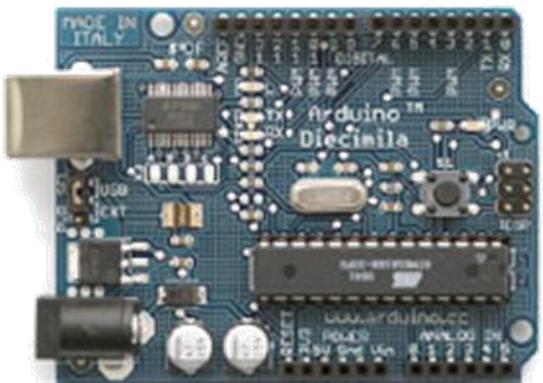
- Arduino is an open-source electronics platform based on easy-to-use **hardware** and **software**.
- **Software** : Arduino Software (IDE), based on Processing tool.
 - **Arduino programming language** (based on Wiring)
- **Hardware**: Arduino Board can be instructed by sending a set of instructions to the microcontroller on the board.
- Arduino boards are able to read inputs like light on a sensor, a finger on a button, or a twitter message and turn it into an output like activating a motor, turning on an LED, publishing something online.



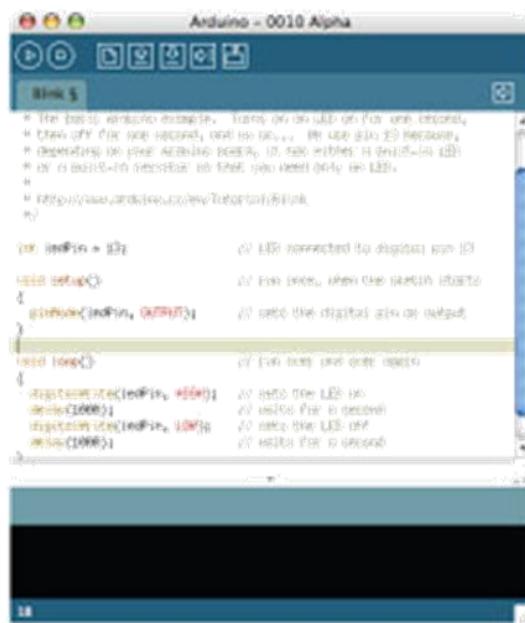


The word “Arduino” can mean 3 things

A physical piece
of hardware



A programming
environment

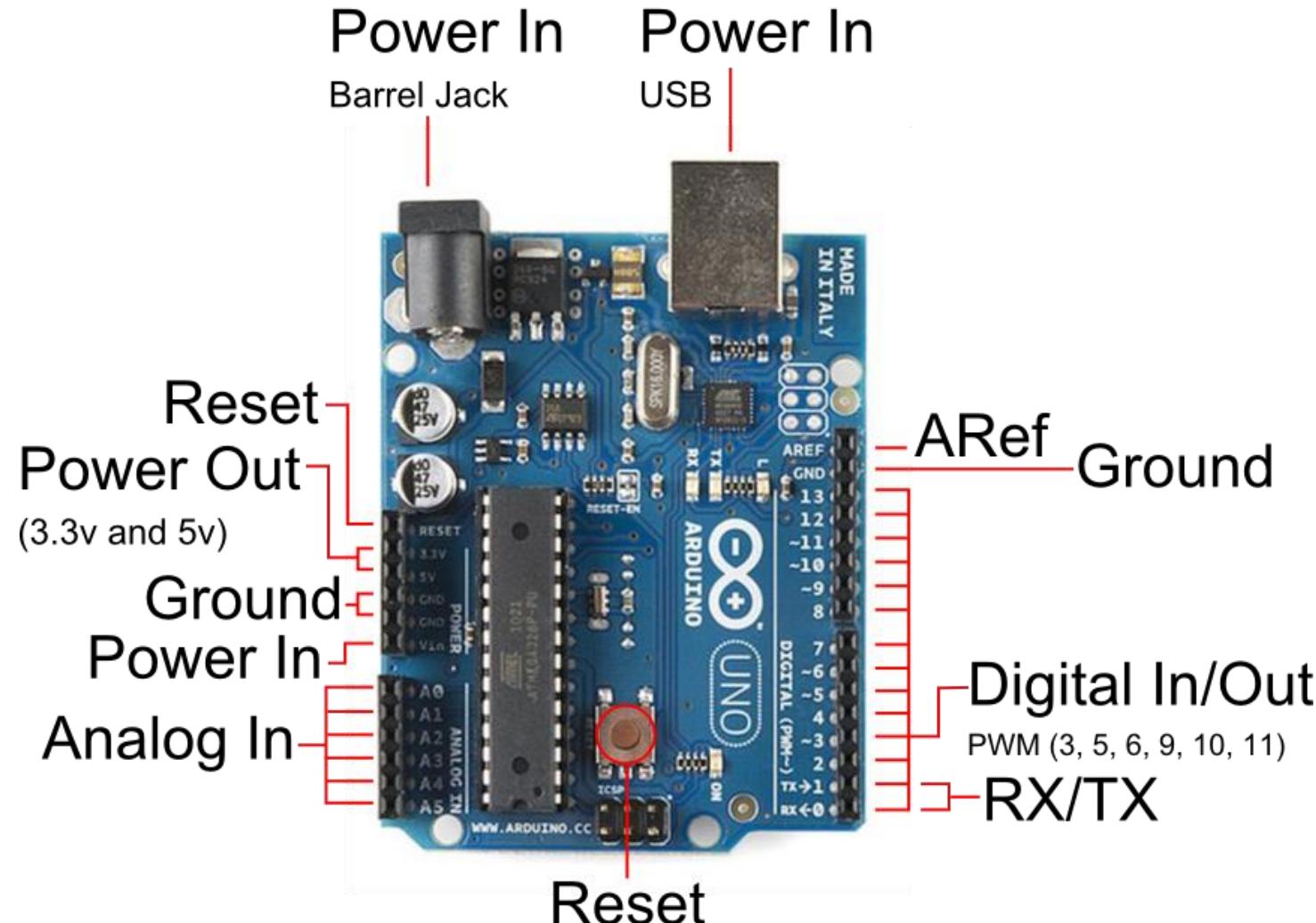


A community
& philosophy



The Arduino Development Board - UNO

Arduino Uno



Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

Arduino Shields

- Arduino shields are the boards, which are plugged over the Arduino board to expand its functionalities.

Why do we need Shields?

- It adds new functionalities to the Arduino projects.
- The shields can be attached and detached easily from the Arduino board. It does not require any complex wiring.
- It is easy to connect the shields by mounting them over the Arduino board.
- The hardware components on the shields can be easily implemented.

Arduino UNO Shield



Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

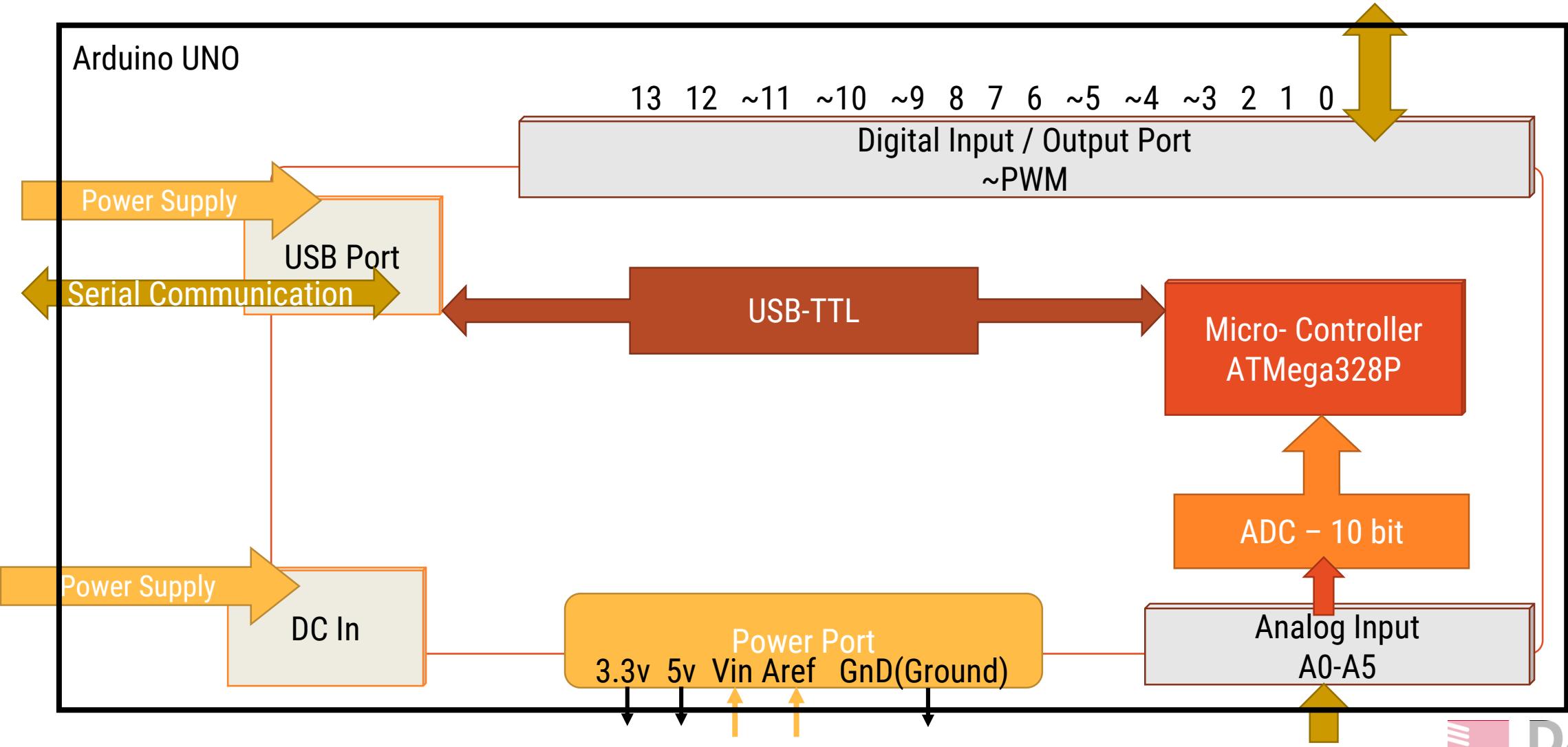
➤ Boot loader

- A boot loader, also called a boot manager, is a small program that places the operating system (OS) of a computer into memory.
 - When a computer is powered-up or restarted, the basic input/output system (BIOS) performs some initial tests, and then transfers control to the Master Boot Record (MBR) where the boot loader resides
- The AVR Boot-loader allows the programming or re-programming of the target AVR microcontroller using the PC serial port instead of a traditional programmer.
- Once the AVR Boot-loader is programmed into the microcontroller, it remains until the chip is erased

Firmware

- Firmware means the permanent software programmed into a read-only memory.
- Firmware is like a software of a product.
- So, Two categories of firmware are used in Arduino:
 - A. Boot-loader : (Default program for Chip startup)
 - USB-TTL
 - ATMEL Chip
 - B. The firmware:
 - (USB-TTL logic program) in the USB chip
 - Sketch in Arduino chip (User Program i.e. LED Blink)
- Just like we flash to install/update EEPROM firmware of a bios in the PC, we flash the Arduino boot-loader in a chip.

Arduino block diagram

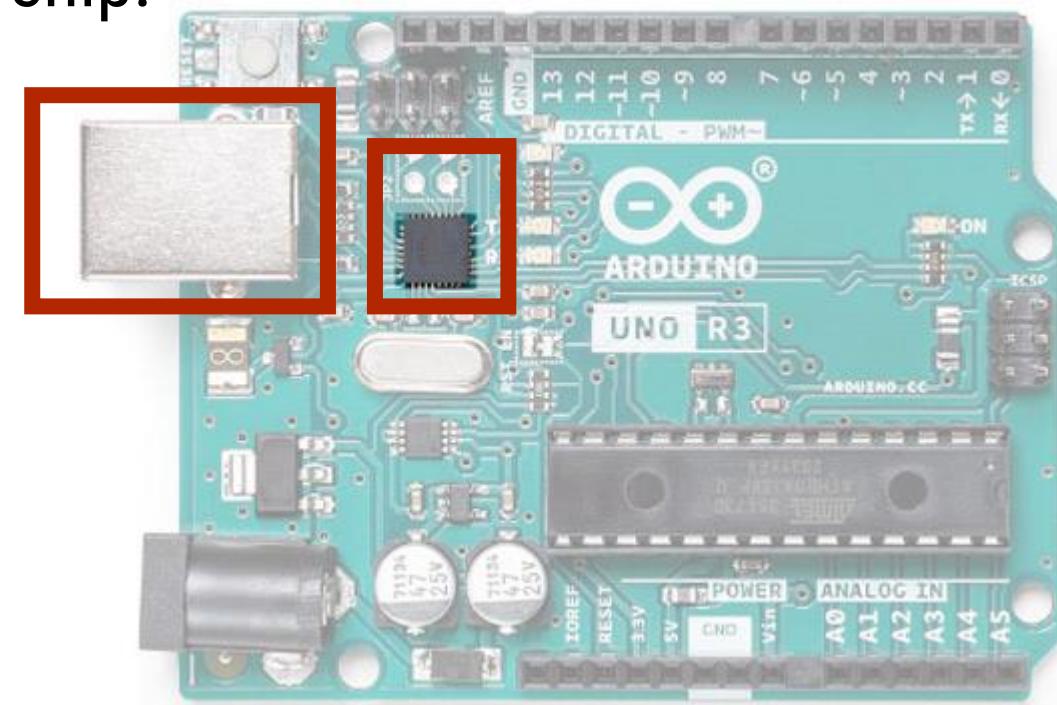


USB - B Socket

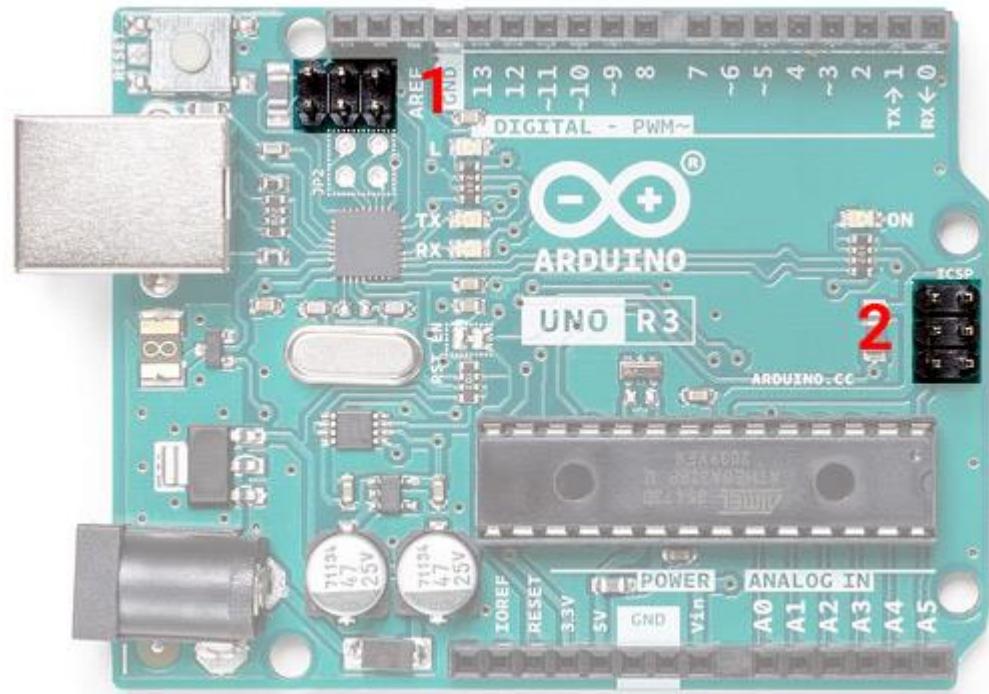
- The USB socket on the UNO has two functions.
- One is for communication.
 - ❖ Connect with the computer through a USB port.
 - ❖ To load the firmware into the Arduino with the help of the bootloader.
- Second is to power the Arduino.
 - ❖ The USB port can be used to power the Uno directly from any USB port.

USB-TTL Interface Chip

- To communicate with the computer, the Arduino relies on a USB-TTL interface.
- In UNO, ATMega16U with custom firmware act as a USB-TTL interface chip.



ICSP Pins- (In-circuit serial programming)



- Two 6 pin connectors.
- 1) One is near the USB – TTL Chip
- 2) At the end of the board.
- These pins are used to program those two microcontrollers.
- The USB – TTL chip on this board is an ATMega16U.
- The connector marked as 1 is used to program the **USB-TTL firmware** into this chip.
- The connector marked as 2 is used to burn the boot-loader into the **ATMega328 microcontroller**.

Four code on Arduino Board:

➤ **USB – TTL chip** : ATMega16U2

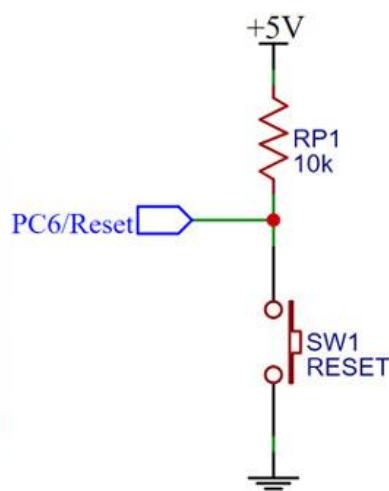
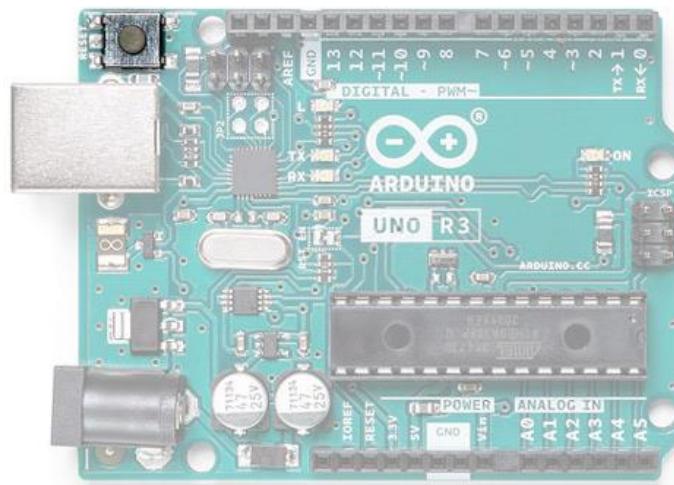
- a) Boot loader on the Atmega16U2 (USB to Serial) chip
- b) Main code on the Atmega16U2

➤ **ATMega328 microcontroller**

- c) Boot loader on the Atmega328P (main) chip
- d) Main code on the Atmega328P

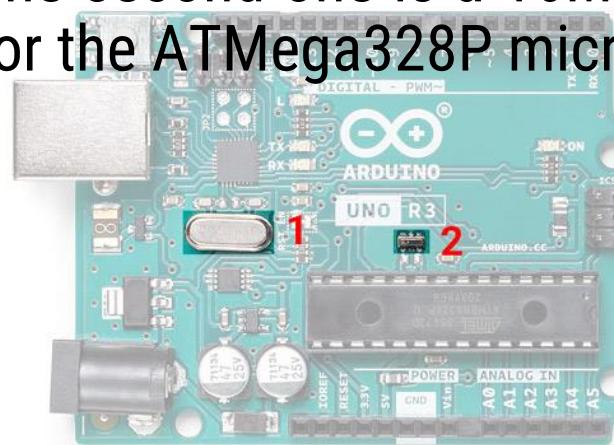
Reset Button

- To reset the ATMega328 microcontroller
- It's connected to the PC6/Reset pin, which is pulled up through a 10K.
- When the switch is pressed the pin is pulled to the ground and the chip(ATMega328 microcontroller) will reset.



Crystal Oscillator/ Ceramic resonator

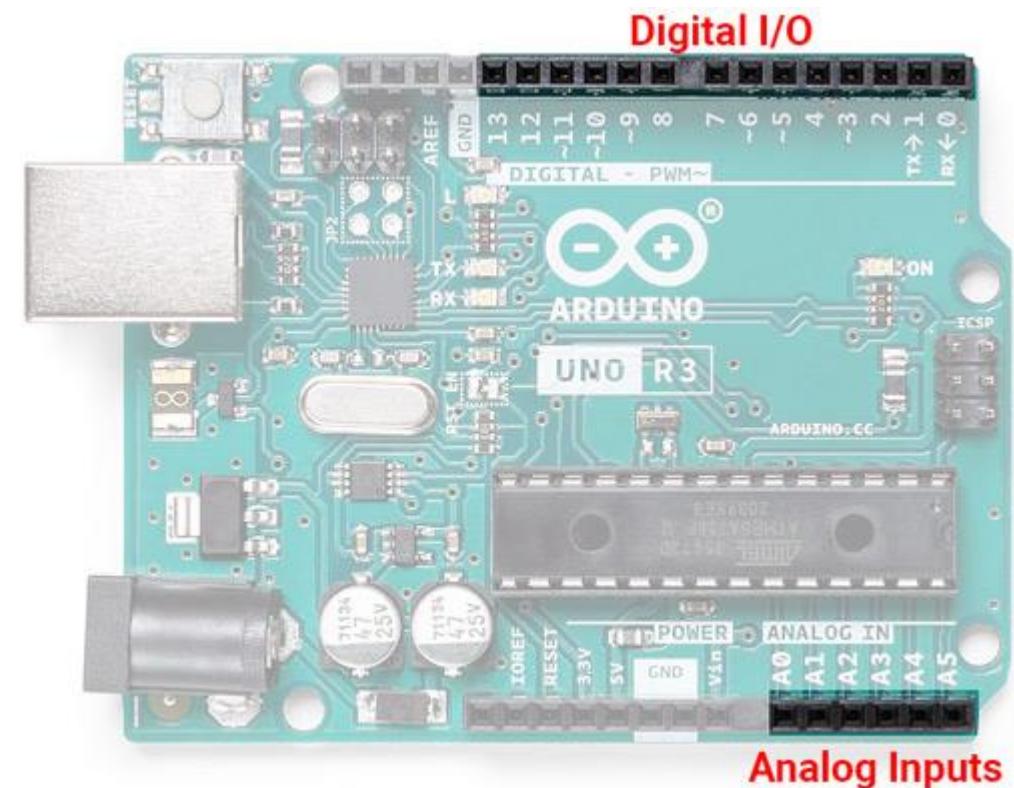
- For a microcontroller to work it needs a clock source.
- The clock circuit determines the speed with which the microcontroller operates.
- How many instructions per second it will execute is dependent on the clock frequency.
 1. The first one is a 16MHz crystal oscillator used for the ATMega16U2 chip
 2. The second one is a 16MHz resonator used for the ATMega328P microcontroller.



Darshan
UNIVERSITY

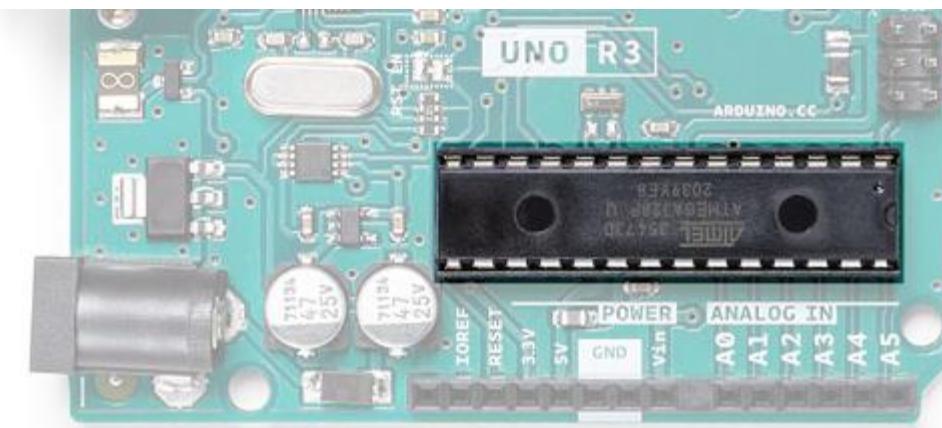
Digital and Analog I/O

- The Arduino UNO has 14 digital I/O pins and 6 Analog inputs.
- The digital I/O pins are 5V logic level and you can also use the Analog pins as digital I/O too.
- Arduino UNO supports 10 bit ADC inputs through A0-A5, which can be sampled and analyzed using UNO.(ADC Analog to digital Converter)
- Analog input Signal converted into Digital through ADC.



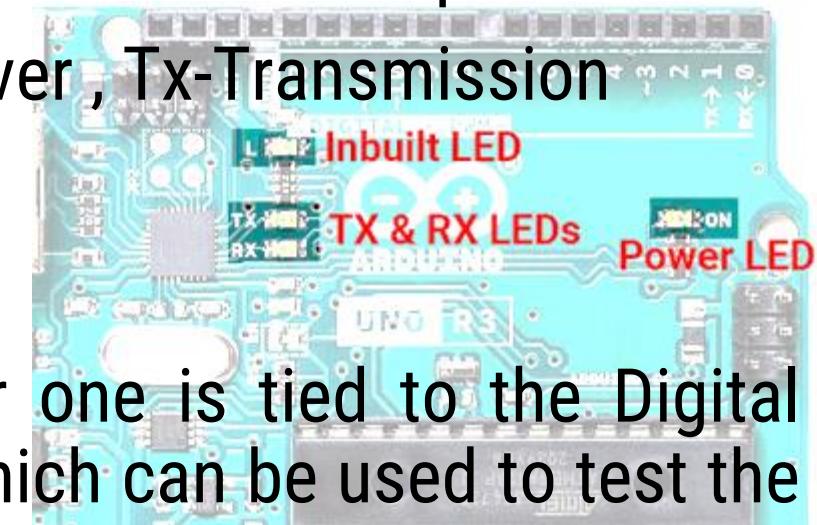
ATMega328P – The Brain

- The main component on the Arduino board – the ATMega328P Microcontroller.
- Atmega328P is pre-programmed with a boot-loader that allows you to directly upload the program to Arduino through USB without the need for an external programmer.



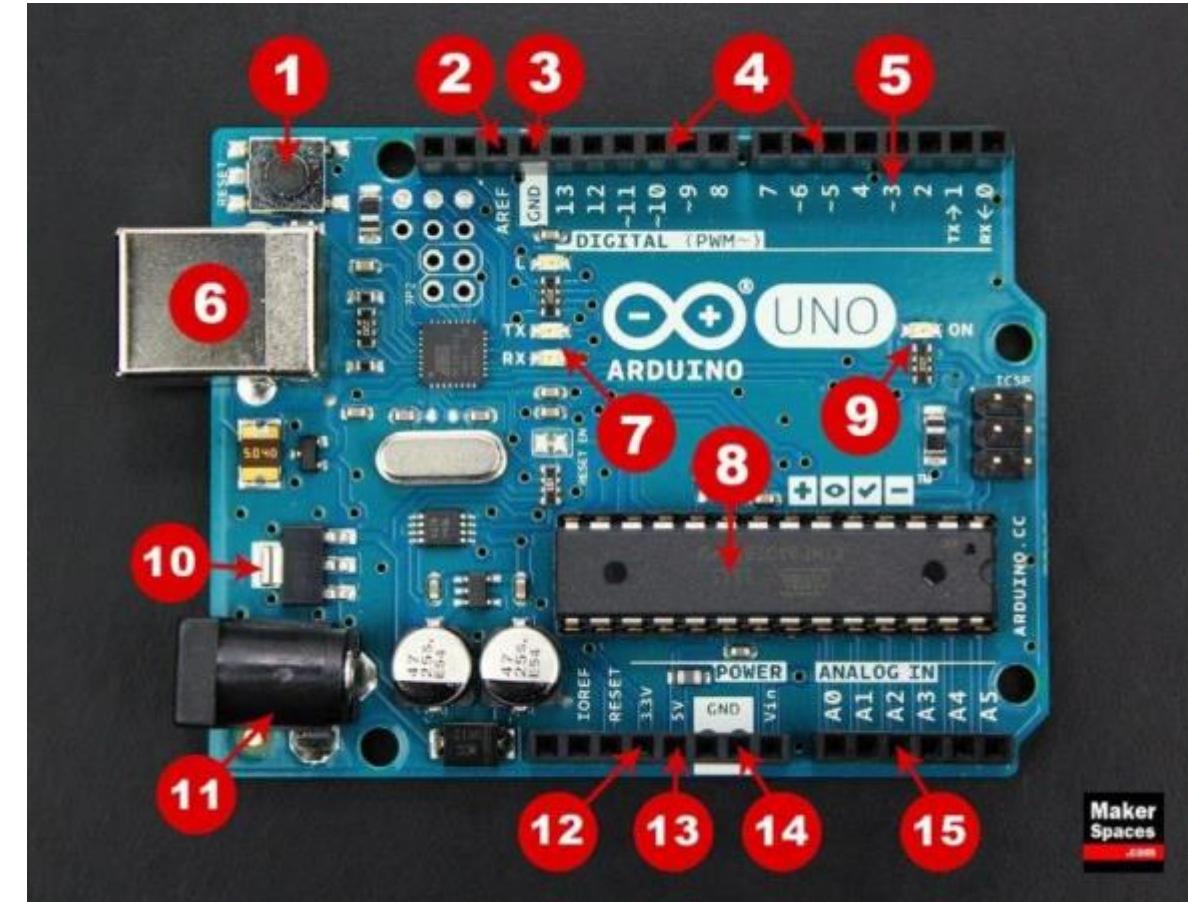
On-Board LED

- Uno has 4 LEDs onboard.
 - a) One is used as a power indicator.
 - Two LED are used to show the activity of the Rx and Tx pin.
 - Rx -Receiver , Tx-Transmission
 - c) Rx LED
 - d) Tx LED
 - The other one is tied to the Digital pin 13, which can be used to test the Arduino board or simply as an indicator.
 - d) LED - Inbuilt

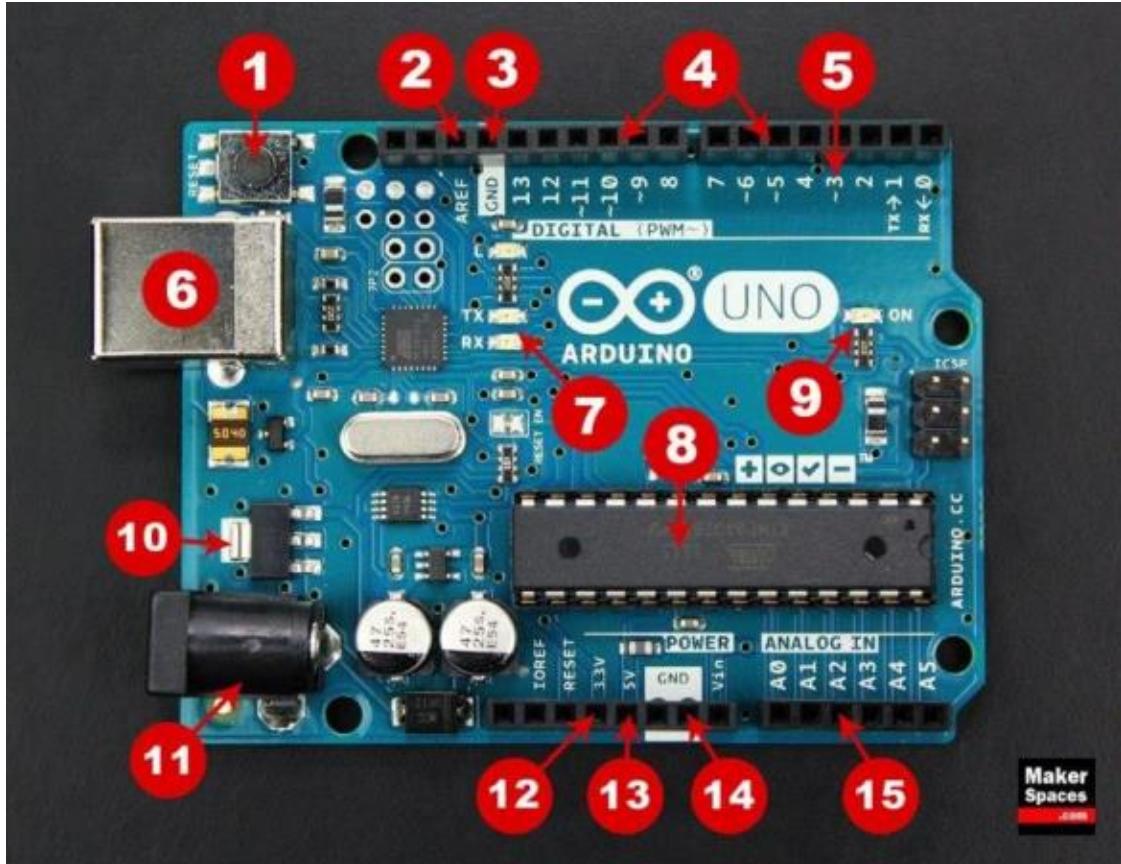


Arduino Uno - Components

- **1. Reset Button** – This will restart any code that is loaded to the Arduino board
- **2. AREF** – Stands for “Analog Reference” and is used to set an external reference voltage
- **3. Ground Pin** – There are a few ground pins on the Arduino and they all work the same
- **4. Digital Input/Output** – Pins 0-13 can be used for digital input or output.
- **5. PWM** – The pins marked with the (~) symbol can simulate analog output



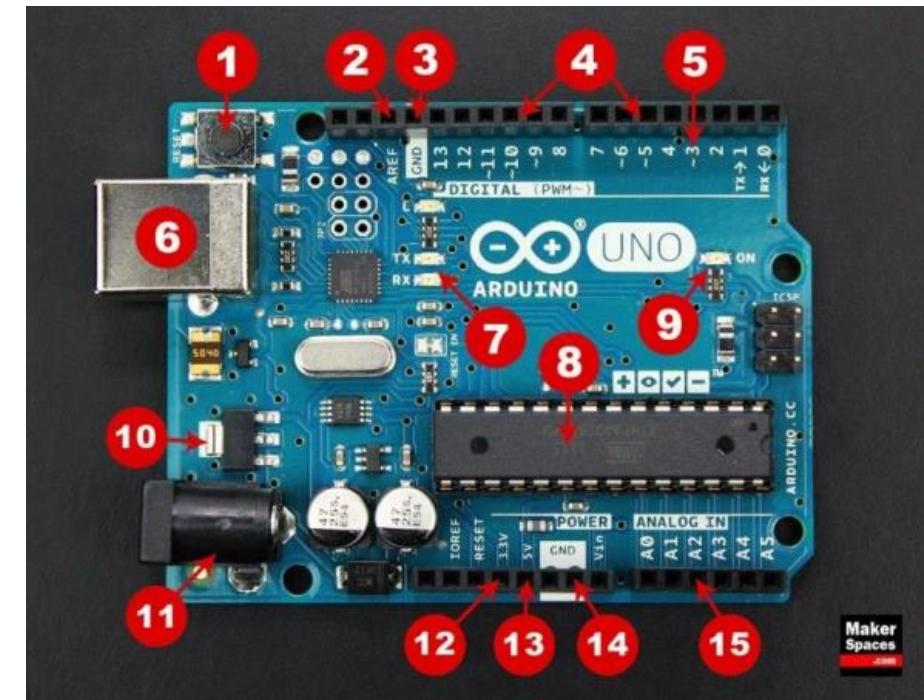
Arduino Uno - Components



- **6. USB Connection** – Used for powering up your Arduino and uploading sketches
- **7. TX/RX** – Transmit and receive data indication LEDs
- **8. ATmega Microcontroller** – This is the brains and is where the programs are stored.
- **9. Power LED Indicator** – This LED lights up anytime the board is plugged in a power source
- **10. Voltage Regulator** – This controls the amount of voltage going into the Arduino board

Arduino Uno - Components

- **11. DC Power Barrel Jack** – This is used for powering your Arduino with a power supply
- **12. 3.3V Pin** – This pin supplies 3.3 volts of power to your projects
- **13. 5V Pin** – This pin supplies 5 volts of power to your projects
- **14. Ground Pins** – There are a few ground pins on the Arduino and they all work the same
- **15. Analog Pins** – These pins can read the signal from an analog sensor and convert it to digital

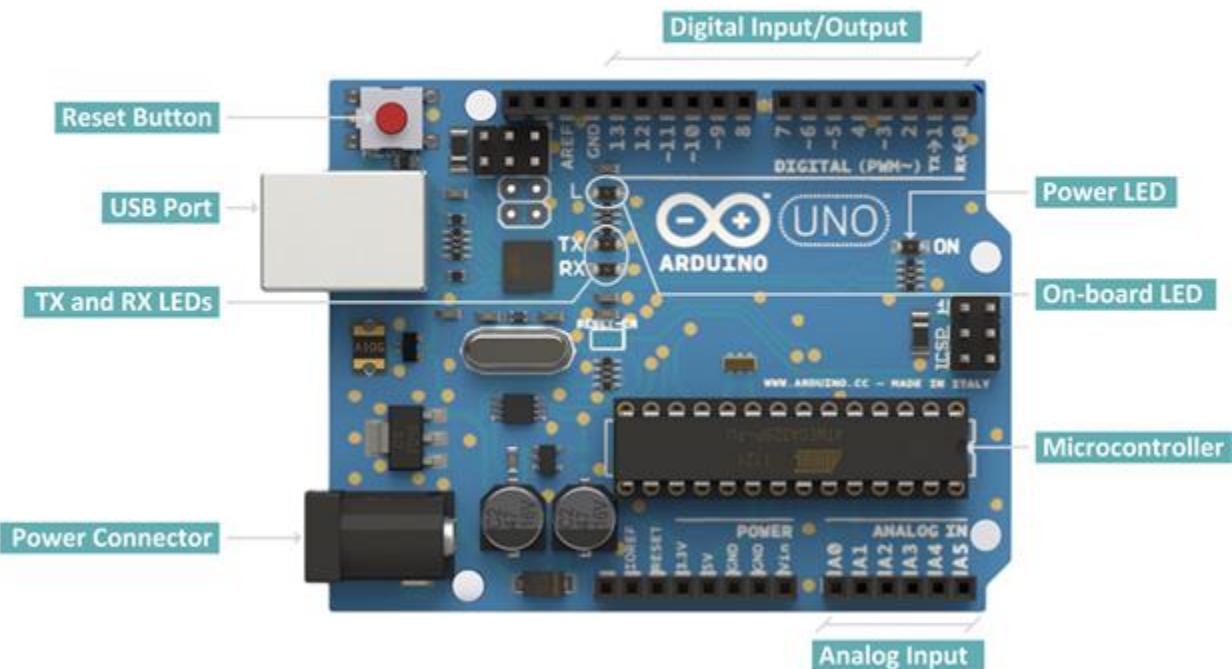


Arduino Power Supply

- The Arduino Uno needs a power source in order for it to operate and can be powered in a variety of ways.
- Connect the board directly to your computer via a USB cable.
- If you want your project to be mobile/ remote, consider using a 9V battery pack.
- The last method would be to use a 9V DC power supply.



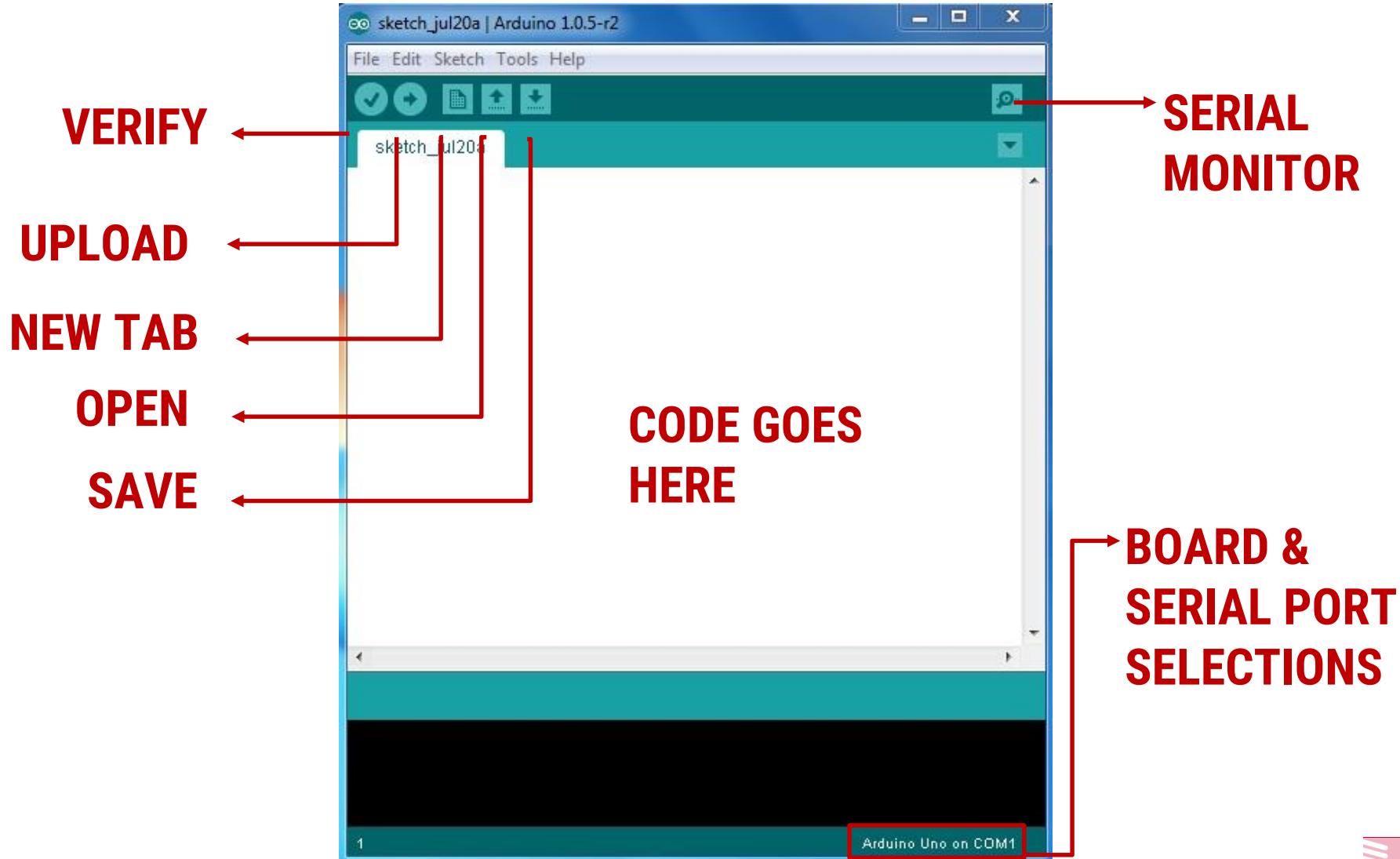
Arduino Uno – Digital And Analog Pins



- **Digital pins:**
- 14 digital IO pins
- 6 are PWM pins (3, 5, 6, 9, 10, and 11).

- **Analog pins:**
- 6 analog pins(A0, A1, A2, A3, A4, and A5)
- Takes analog values as an input

Arduino IDE



Arduino IDE

➤ IDE : Integrated Development Environment.

➤ 1.) Code Editor

- It is the space where we can write the code

➤ 2.) Text Console

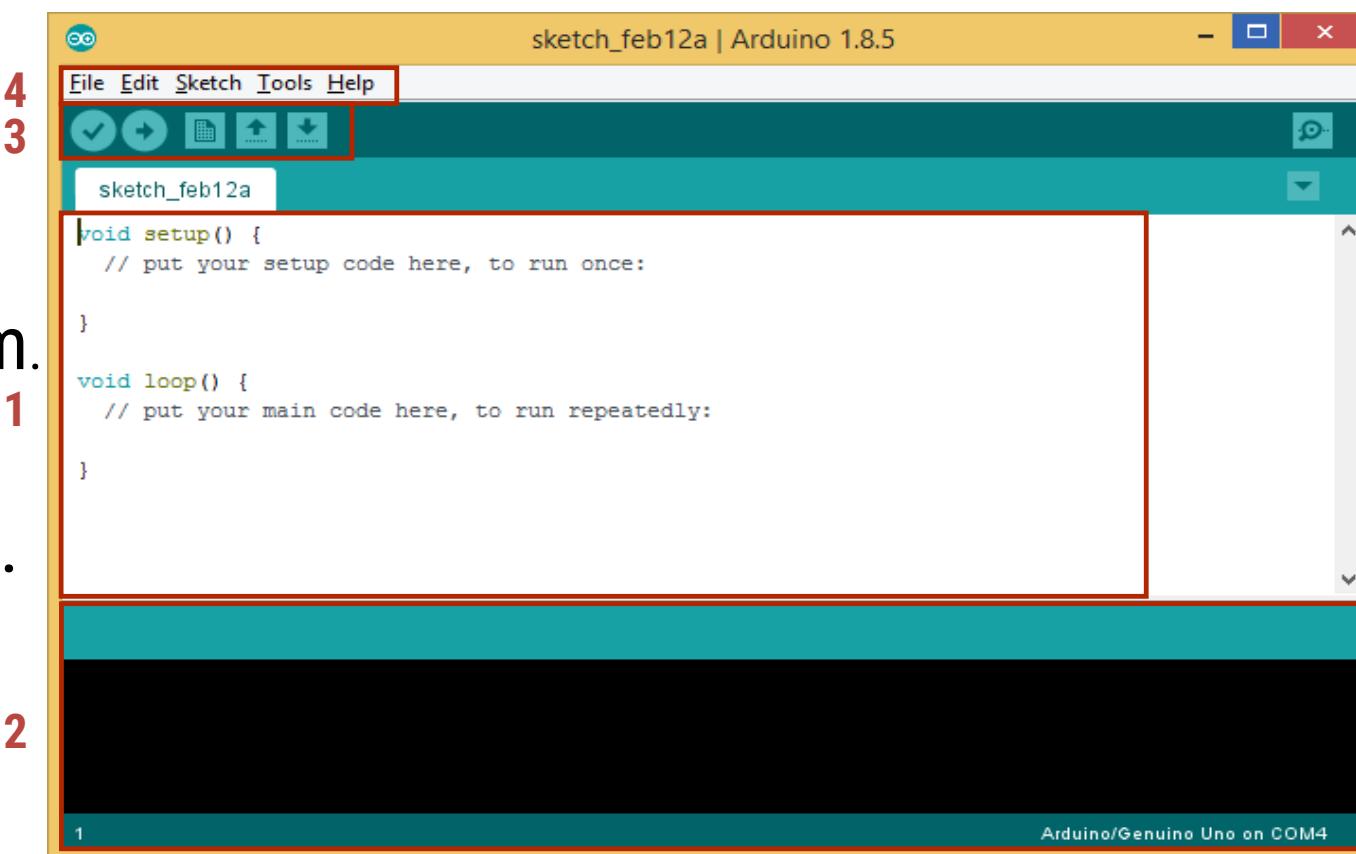
- For displaying the messages.

➤ 3.) Toolbar with buttons

- For common functions to perform.

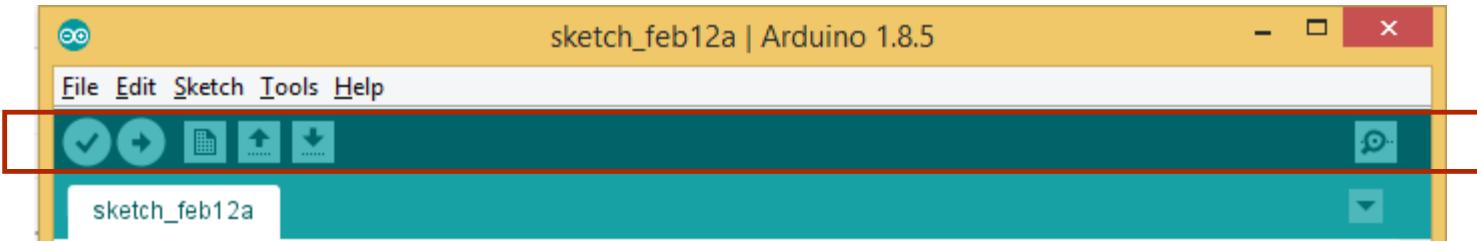
➤ 4.) Menu bar

- A menu bar with series of menus.



Arduino IDE - Toolbar

- The functions of buttons in toolbar can be explained as following:



Verify

Checks your code for errors compiling it.



Open

Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.



Upload

Compiles your code and uploads it to the configured board.



Save

Saves your sketch.



New

Creates a new sketch.



Serial Monitor

Opens the serial monitor.

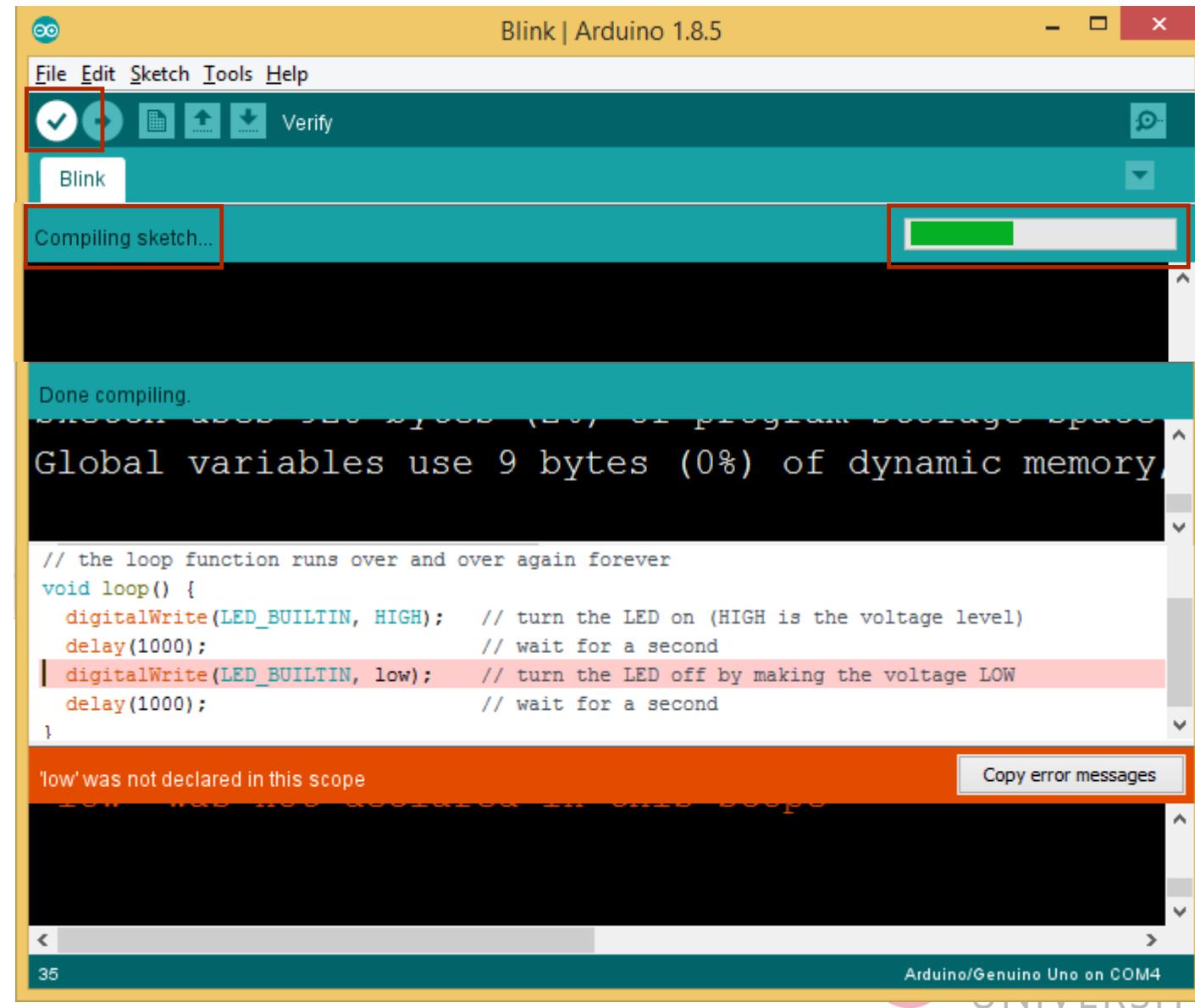


Darshan
UNIVERSITY

गोपा कर्मसु कौशलम्

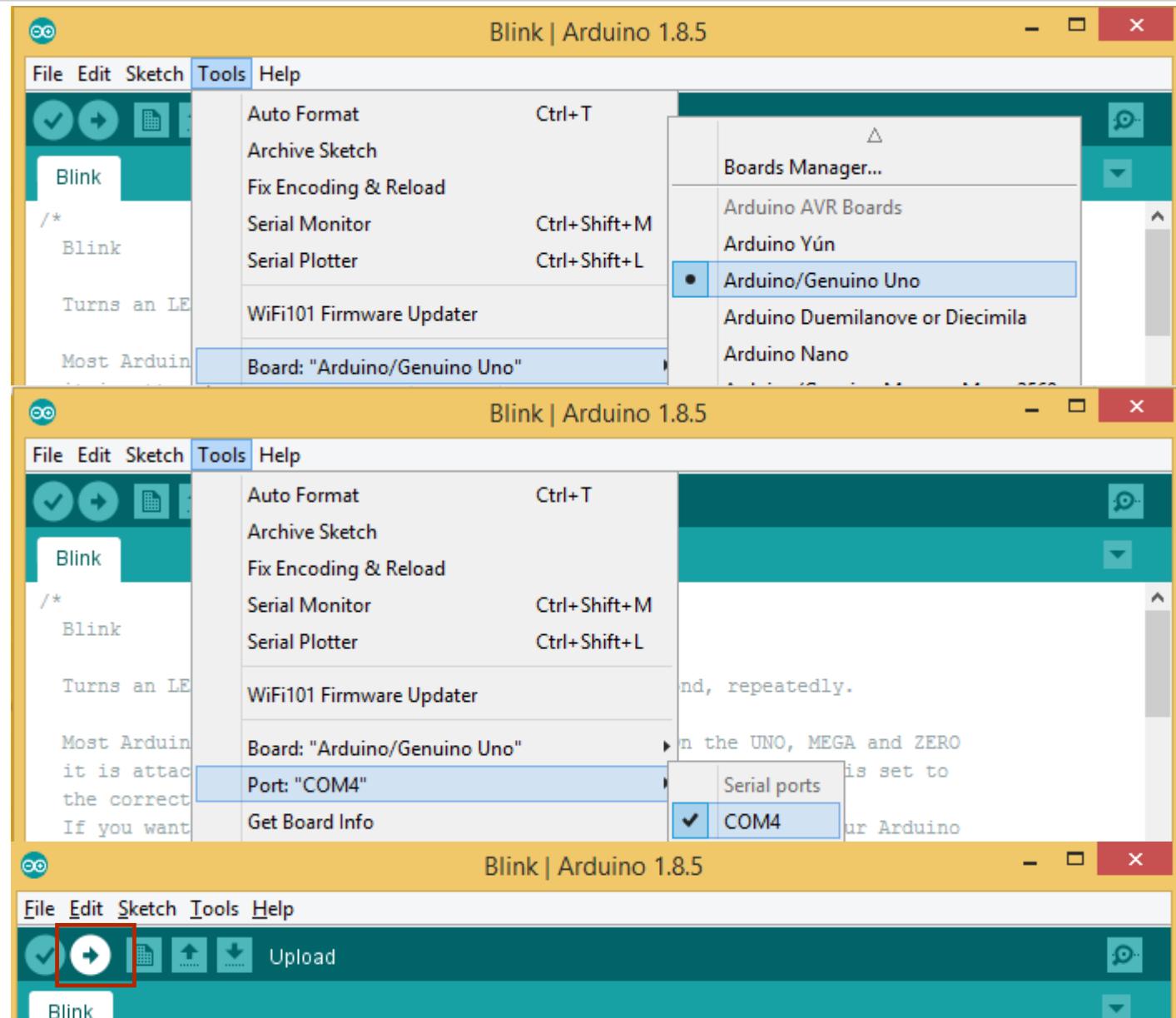
Compiling the Code

- To compile a code in Arduino click on the “Verify” button in menu bar.
- While compiling the code, a message “Compiling sketch” is shown with its progress in the status bar.
- If there is no error in the code, the message “Done compiling” is displayed in the status bar.



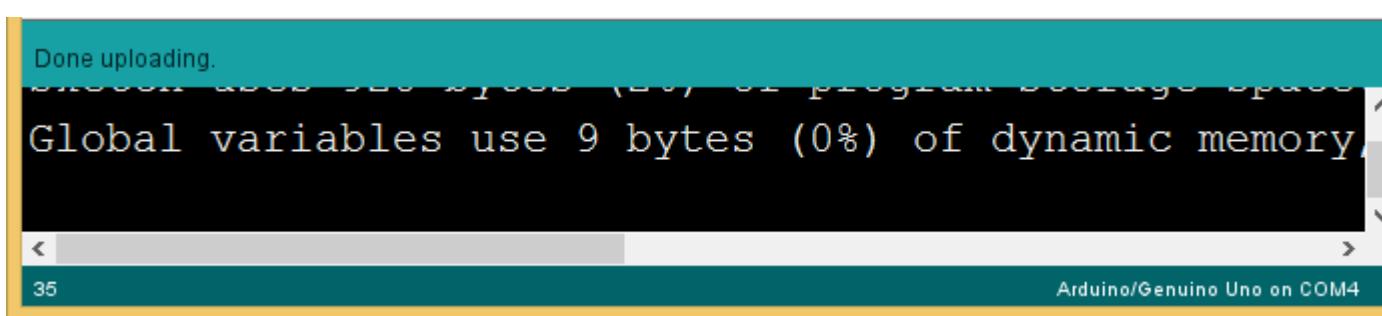
Upload the first code in Arduino Hardware

- Select the proper board type:
 - First, we need to select the appropriate board type in which we need to upload the program.
- Go to Tools > Board menu and select the board which is used as the hardware.
- Select port: Select the serial device of the board from the Tools > Port menu.
- This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports).



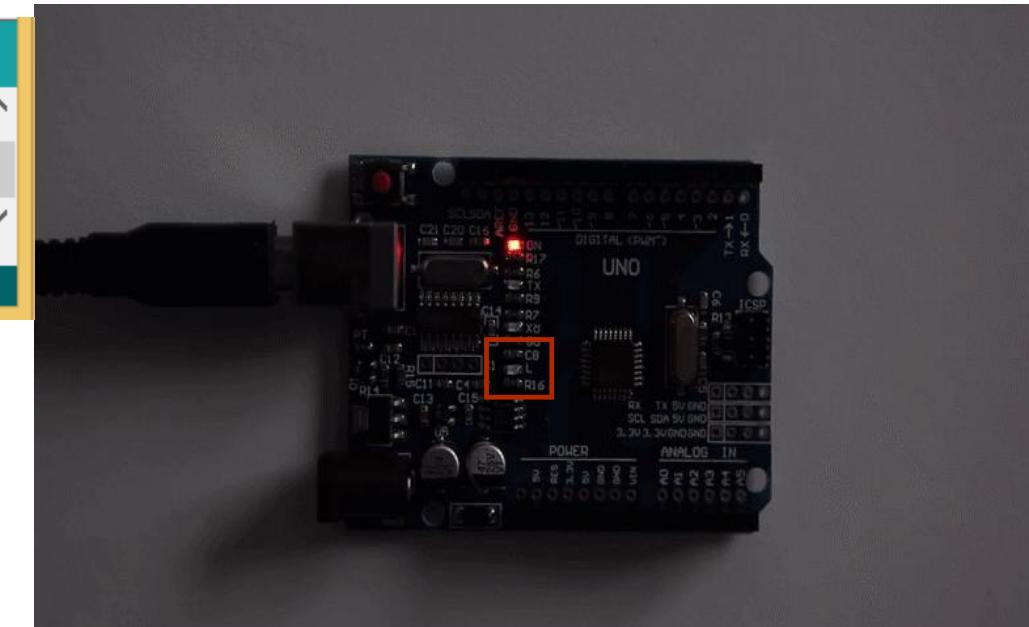
Upload the first code in Arduino Hardware

- While uploading process, TX and RX LED will be flashing.
- If uploading is done successfully, it will show a message “Done uploading” in the status bar.
- Once the program is uploaded, the built-in LED of the Arduino board will flash with one second delay.



The screenshot shows the Arduino IDE interface. The status bar at the bottom indicates "Done uploading." The main window displays the uploaded code, which includes a message about global variables and their memory usage. The serial monitor window shows the uploaded code and its output.

```
Done uploading.  
Global variables use 9 bytes (0%) of dynamic memory.  
35  
Arduino/Genuine Uno on COM4
```



Basic Coding Structure

```
void setup() {  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    // put your main code here, to run  
    repeatedly:  
  
}
```

Basic Coding Structure

➤ **setup() function**

- Called only once, when Board starts.
- To initialize variables, pin modes, start using libraries, etc.
- It is called only when the Arduino is powered on or reset.

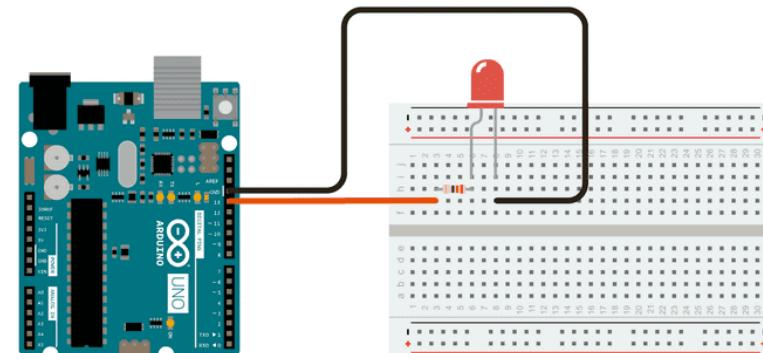
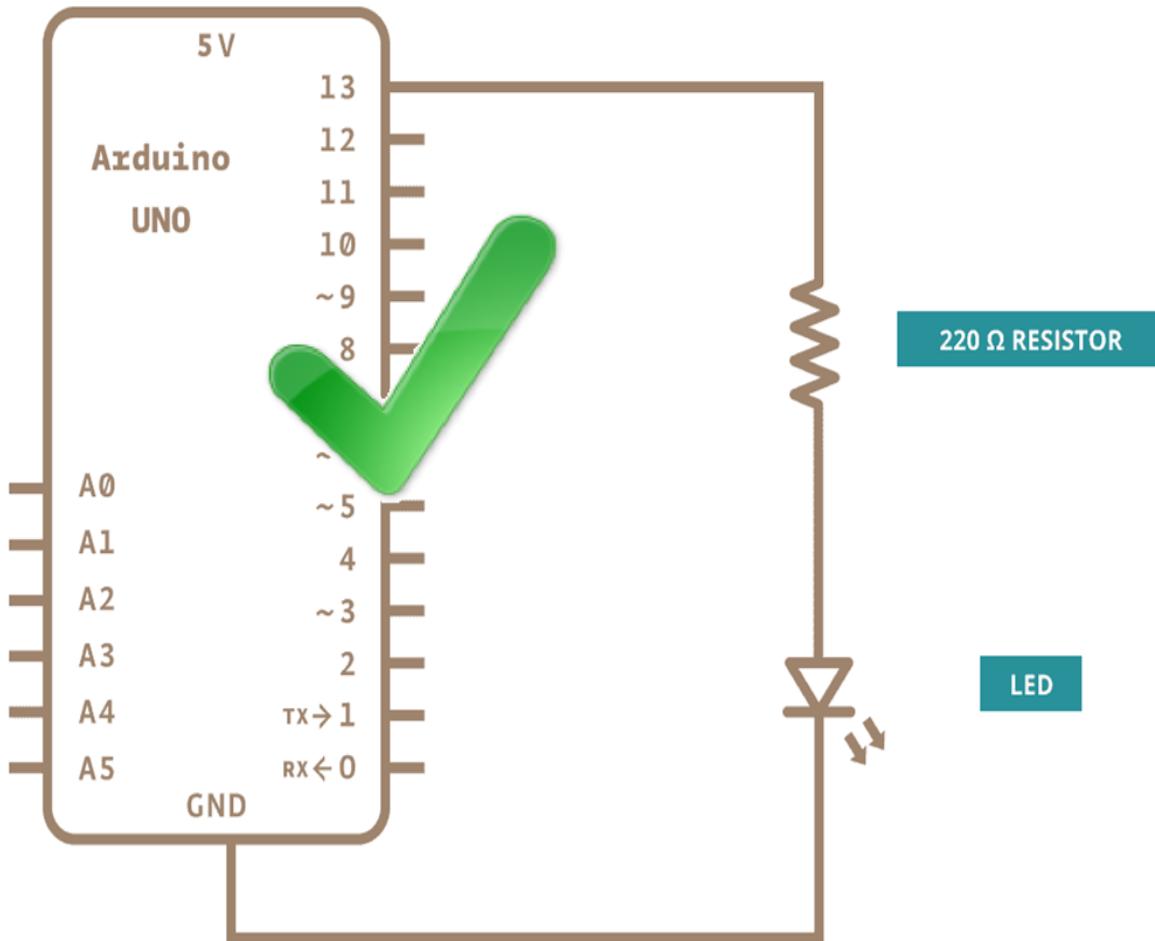
➤ **loop() function**

- The loop functions runs continuously till the device is powered off.
- The main logic of the code goes here.
 - Similar to while (1) for micro-controller programming.
- Code in the loop() section of the sketch is used to actively control the Arduino board.

➤ **Commenting**

- Any line that starts with two slashes (//) will not be read by the compiler.

Program 1 : Blink LED : Schematic



Sketch

- We should initialize the pin as input or output once only.
- Therefore the syntax for defining the pin as output is written in void setup () function.
- The code inside the loop() function runs continuously. void loop() behaves as while(1).

Blink.ino

```
1 void setup() {  
2     pinMode(LED_BUILTIN, OUTPUT);  
3 }  
4 void loop() {  
5     digitalWrite(LED_BUILTIN, HIGH);  
6     delay(1000);  
7     digitalWrite(LED_BUILTIN, LOW);  
8     delay(1000);  
9 }
```

Initialize digital pin LED_BUILTIN which is at pin 13 as an output.

turn ON the LED by writing HIGH voltage level

A delay of 1000ms = 1 sec

turn OFF the LED by writing LOW voltage level

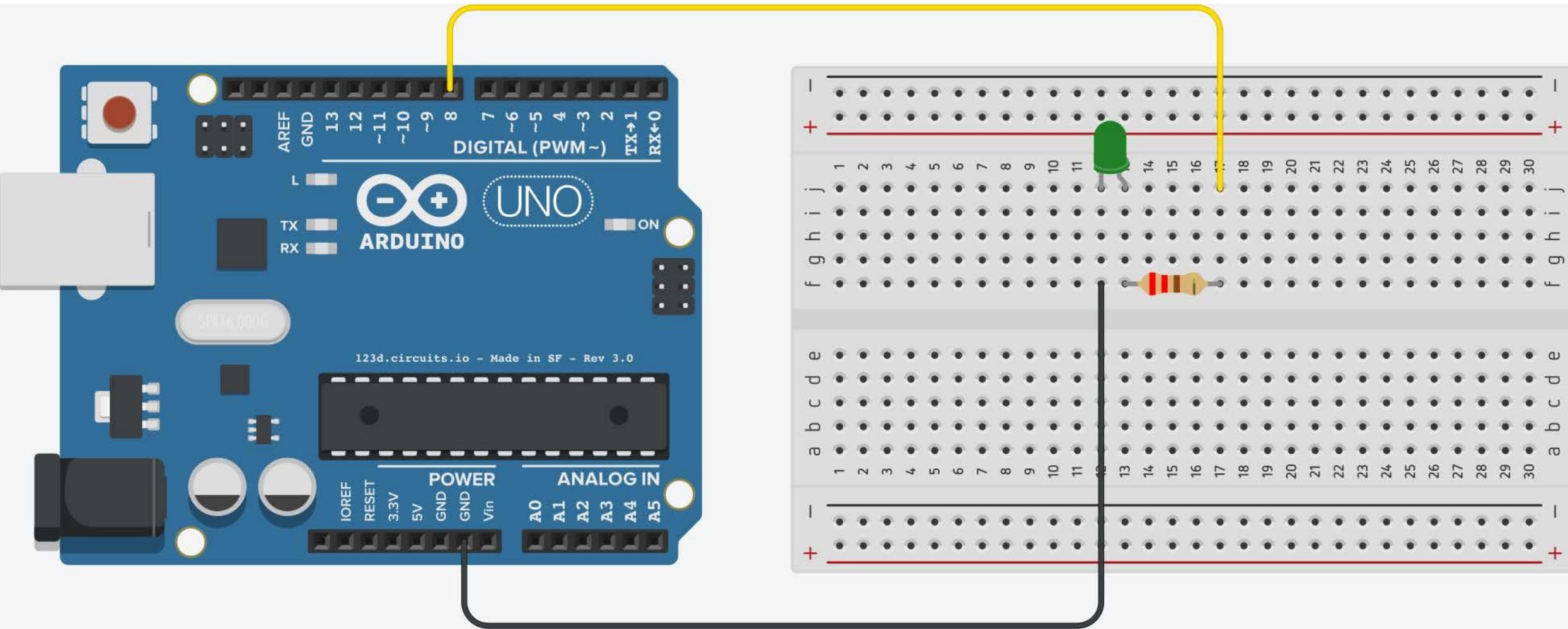
A delay of 1000ms = 1 sec



Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

Program 2 : LED on/off



sketch_01.ino

```
1 void setup() {
2     pinMode(8, OUTPUT);
3 }
4 void loop() {
5     digitalWrite(8, HIGH);      // Turn on the LED
6     delay(1000);              // Wait for one second
7     digitalWrite(8, LOW);      // Turn off the LED
8     delay(1000);              // Wait for one second
9 }
10
11
12
```

Data Types : void & Boolean

➤ Void

- The void keyword is used only in function declarations.
- The function is expected to return no information.

➤ Example :

```
void loop ()  
{ // rest of the code }
```

➤ Boolean

- A Boolean holds one of two values, true or false.
- Each Boolean variable occupies one byte of memory.

■ Example:

- ❖ **bool val = false ;**
- ❖ **bool state = true ;**

Data Types : byte and word

➤ byte

- A byte stores an **8-bit unsigned number** (from 0 to 255)
- Example: **byte m = 25 ;** //declaration of variable with type byte and initialize it with 25

➤ word

- On the Uno and other ATMEGA based boards, a word stores a **16-bit unsigned number**.
- Example: **word w = 1000 ;**
//declaration of variable with type word and initialize it with 1000

Data Types : int

➤ Int

- Integers are the primary data-type for number storage.
- int stores a **16-bit (2-byte) value**. (For UNO)
- This yields a range of -32,768 to 32,767 (minimum value of -2^{15} and a maximum value of $(2^{15}) - 1$)
- The **int size varies from board to board**, On the **Arduino Due**, int stores a **32-bit (4-byte) value**.
- Example: **int counter = 32 ; // declaration of variable with type int and initialize it with 32**

➤ unsigned int

- Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 ($2^{16} - 1$).
- Same as word data type.
- Example : **unsigned int counter = 60 ; // declaration of variable with**

Data Types : Char

➤ **char**

- one byte of memory that stores a character value
- Character literals are written in **single quotes** like this: 'A' and for multiple characters, strings use double quotes: "ABC".
- However, characters are stored as numbers (ASCII value), This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used.
- For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.
- Example:
 - **char cha = 'a';**
 - **char chc = 97 ;**

Data types : long and unsigned long

➤ long

- Long variables are extended size variables for number storage, and store **32 bits (4 bytes)**.
- From -2,147,483,648 to 2,147,483,647.
- Example: **long** velocity = 102346 ;

➤ unsigned long

- unsigned long variables are extended size variables for number storage and store **32 bits (4 bytes)**.
- Unlike standard long, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 ($2^{32} - 1$).
- Example : **unsigned long** velocity = 101006 ;

Data Types : Float and Double

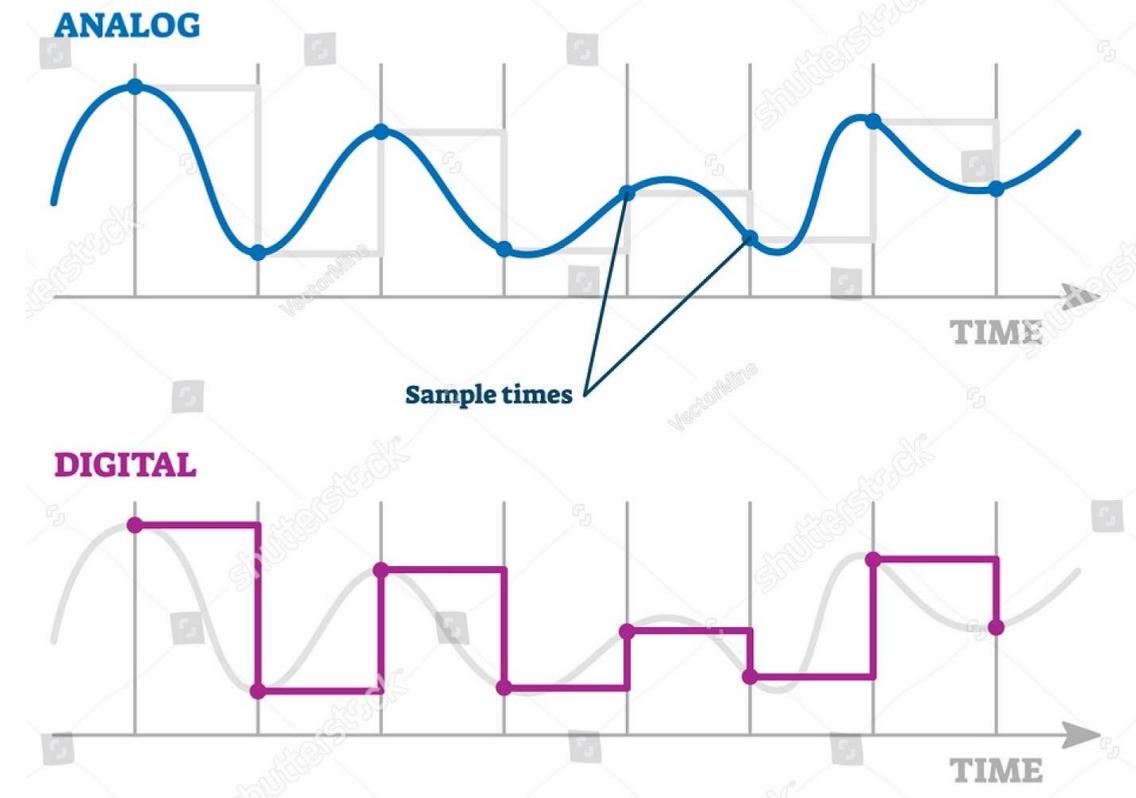
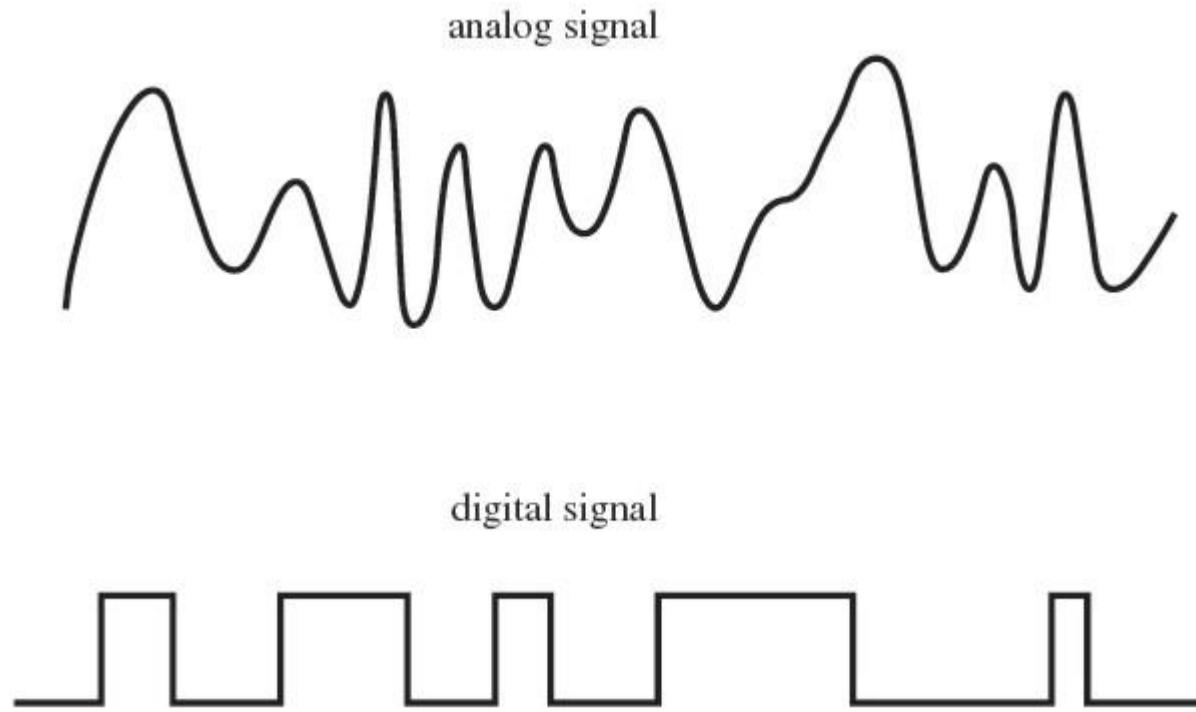
➤ float

- Data type for floating-point number is a number that has a decimal point.
- Floating-point numbers are often used to approximate the analog and continuous values because they have greater resolution than integers.
- Floating-point numbers can be as large as 3.4028235E+38 and as low as -3.4028235E+38.
- They are stored as 32 bits (4 bytes) of information.
- Example : **float** num = 1.352;

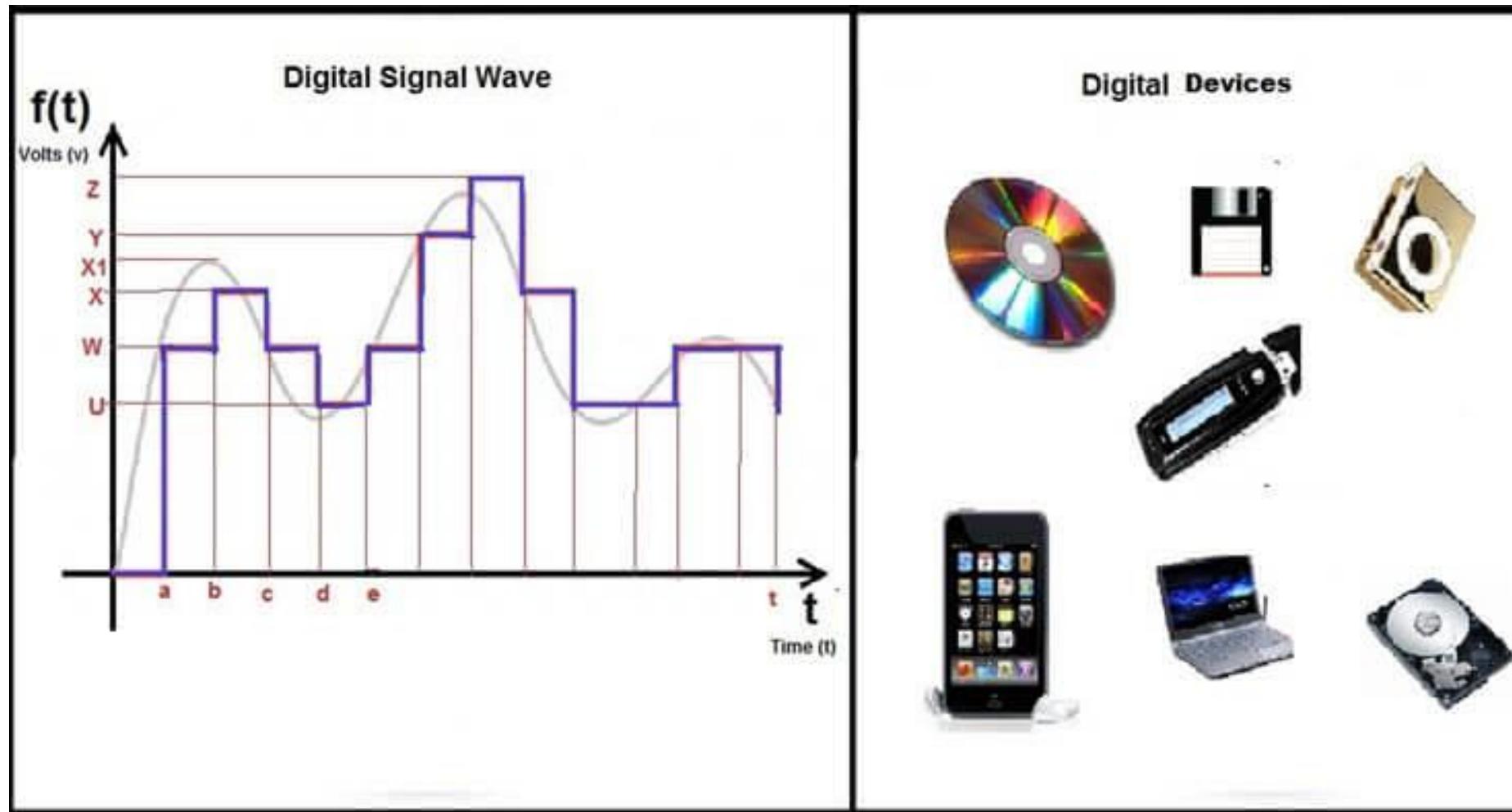
➤ double

- On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes.
- The double implementation is exactly the same as the float, with no gain in precision.
- Example: **double** num = 45.352 ;

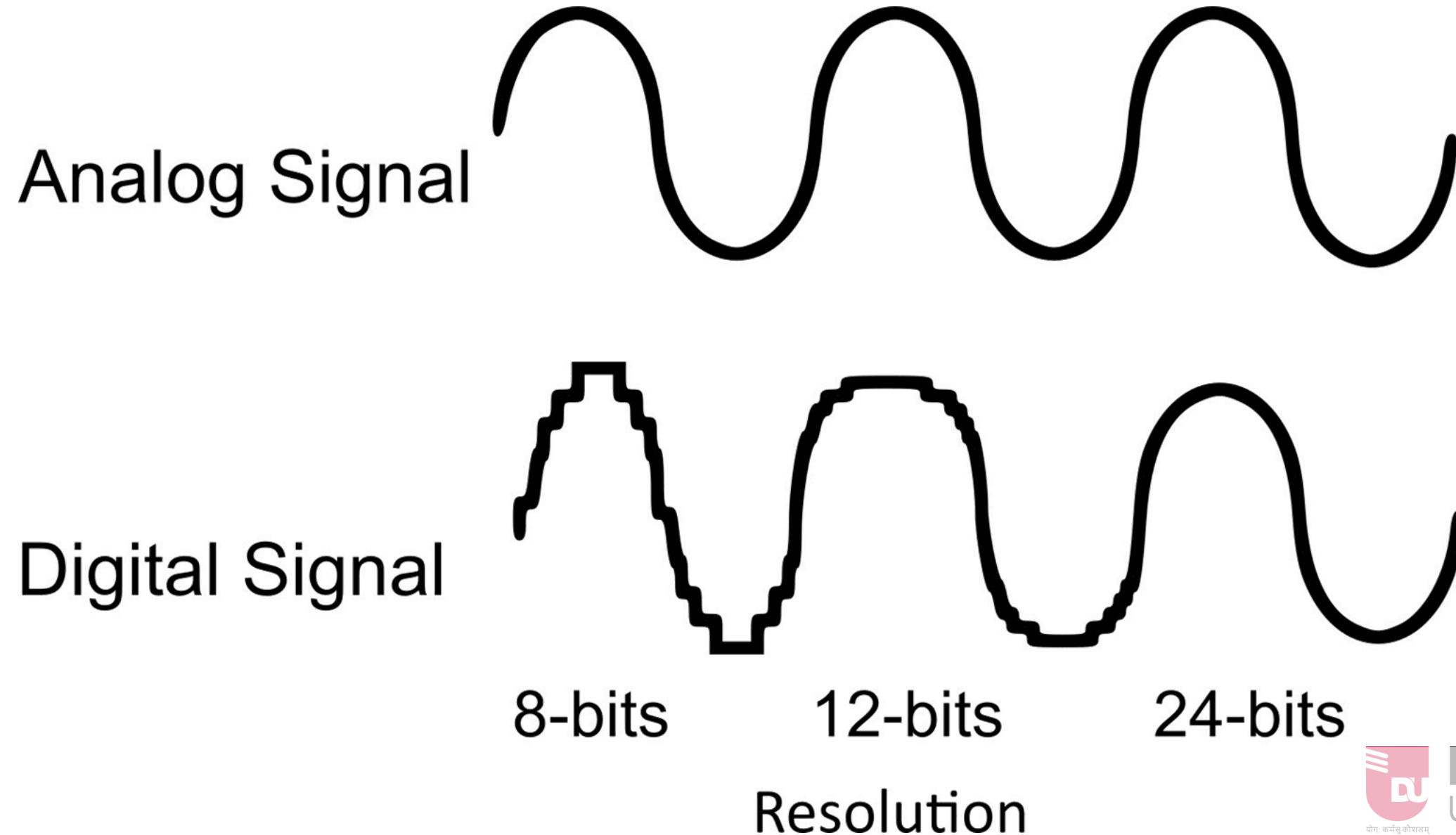
Digital And Analog



Analog to Digital



ADC Resolution



Functions in Arduino IDE-Digital Functions

➤ Digital Input - Output

pinMode()

digitalRead()

digitalWrite()



Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

Pin - Mode

1. pinMode():

➤ Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

➤ Syntax

➤ **pinMode(pin, mode)**

➤ **pin**: Arduino pin number to set the mode of.

➤ **mode**: INPUT, OUTPUT, or INPUT_PULLUP.

➤ Example:

```
void setup() {
    pinMode(13, OUTPUT);      // sets the digital pin 13 as output
}

void loop() {
    digitalWrite(13, HIGH);   // sets the digital pin 13 on
    delay(1000);             // waits for a second
    digitalWrite(13, LOW);    // sets the digital pin 13 off
    delay(1000);             // waits for a second
}
```

Digital Output

2. digitalWrite():

➤ Write a HIGH or a LOW value to a digital pin. HIGH means Logic 1. It will set 5V on the configured pin of Arduino

➤ Syntax

➤ **digitalWrite(pin, value)**

➤ **pin:** Arduino pin number.

➤ **value:** HIGH or LOW.

➤ Example:

```
void setup() {  
    pinMode(13, OUTPUT); // sets the digital pin 13 as output  
}  
  
void loop() {  
    digitalWrite(13, HIGH); // sets the digital pin 13 on  
    delay(1000); // waits for a second  
    digitalWrite(13, LOW); // sets the digital pin 13 off  
    delay(1000); // waits for a second  
}
```

Digital Input

3. **digitalRead()**:

➤ Reads the value from a specified digital pin, either HIGH or LOW.

➤ **Syntax**

➤ **digitalRead(pin)**

➤ **pin:** the Arduino pin number you want to read

➤ **Example:**

DigitalRead.ino

```
1 void setup() {  
2   pinMode(10, INPUT); //Pin 10 as Input pin  
3 }  
4  
5 void loop() {  
6   int data = digitalRead(10); //read the data on Pin  
7   10  
8 }
```



Darshan
UNIVERSITY

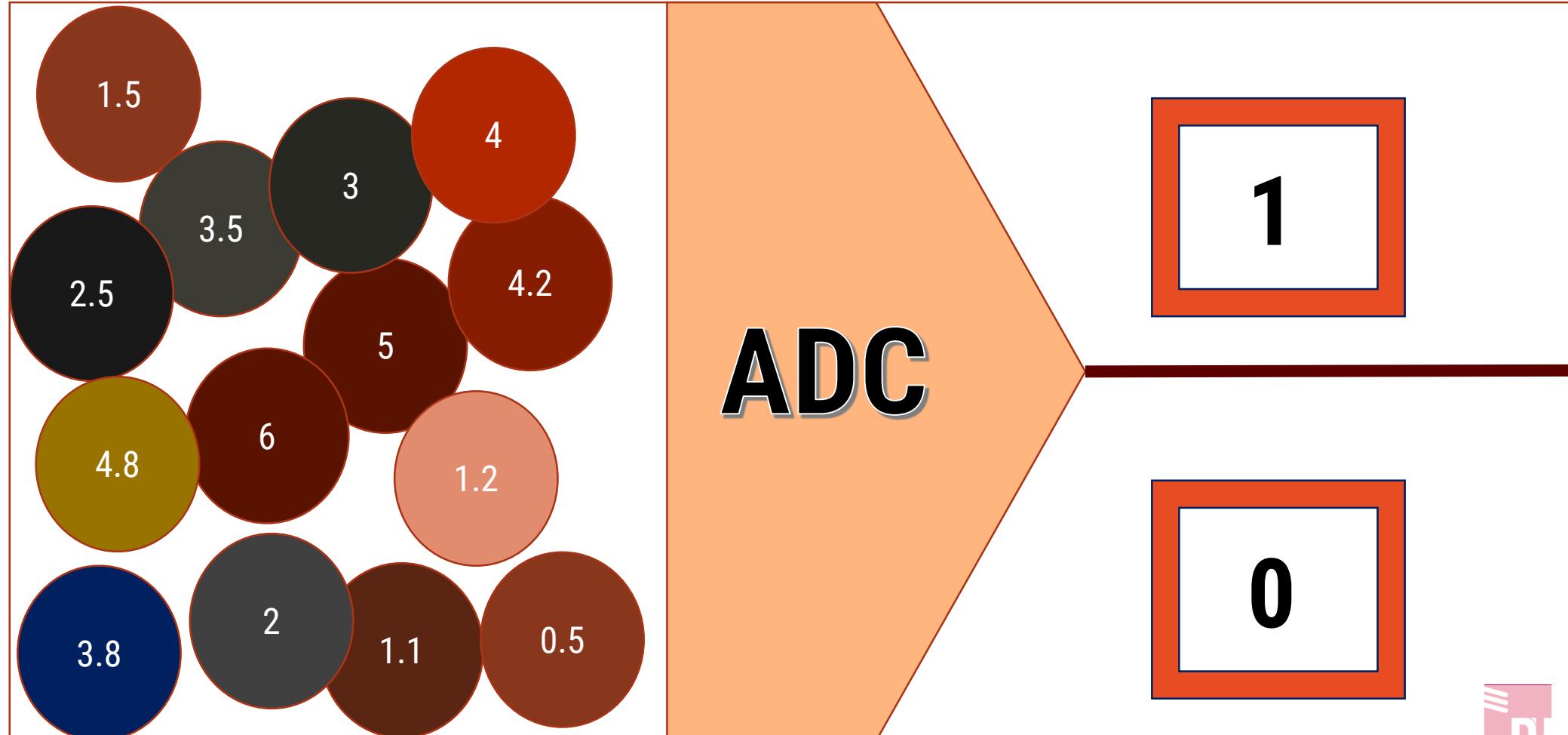
गोपा: कर्मसु कीशतम्

Program : Write code to read data from obstacle sensor

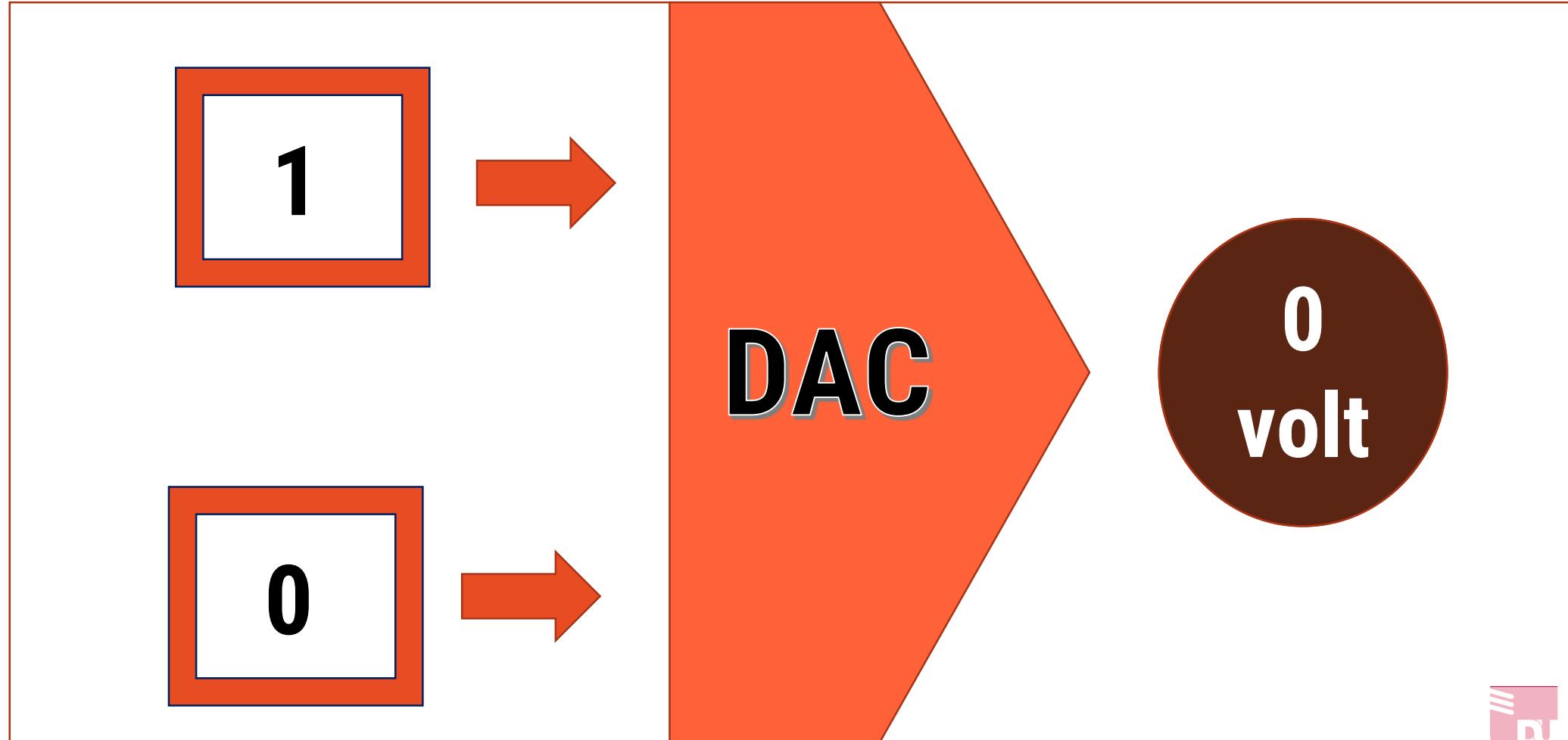
obstacle.ino

```
1 int obstacle = 0;
2 void setup() {
3     pinMode(2, INPUT); //pin 2 as Input pin
4     pinMode(13, OUTPUT); //In Built LED as Output
5     Serial.begin(9600); } //Start Serial Communication
6 void loop() {
7     obstacle = digitalRead(2); // Read from Pin:2
8     if (obstacle == LOW) {
9         Serial.print("Obstacle is present");
10        digitalWrite(13, HIGH);
11    }
12    else {
13        digitalWrite(13, LOW);
14    }
15    delay(1000);
16 }
```

1 bit ADC [For Digital Input] (Analog To Digital)



1 bit DAC [For Digital Output] (Digital To Analog)



Darshan
UNIVERSITY

गोपा: कर्मसु कौशलम्

Functions in Arduino IDE-Analog Functions

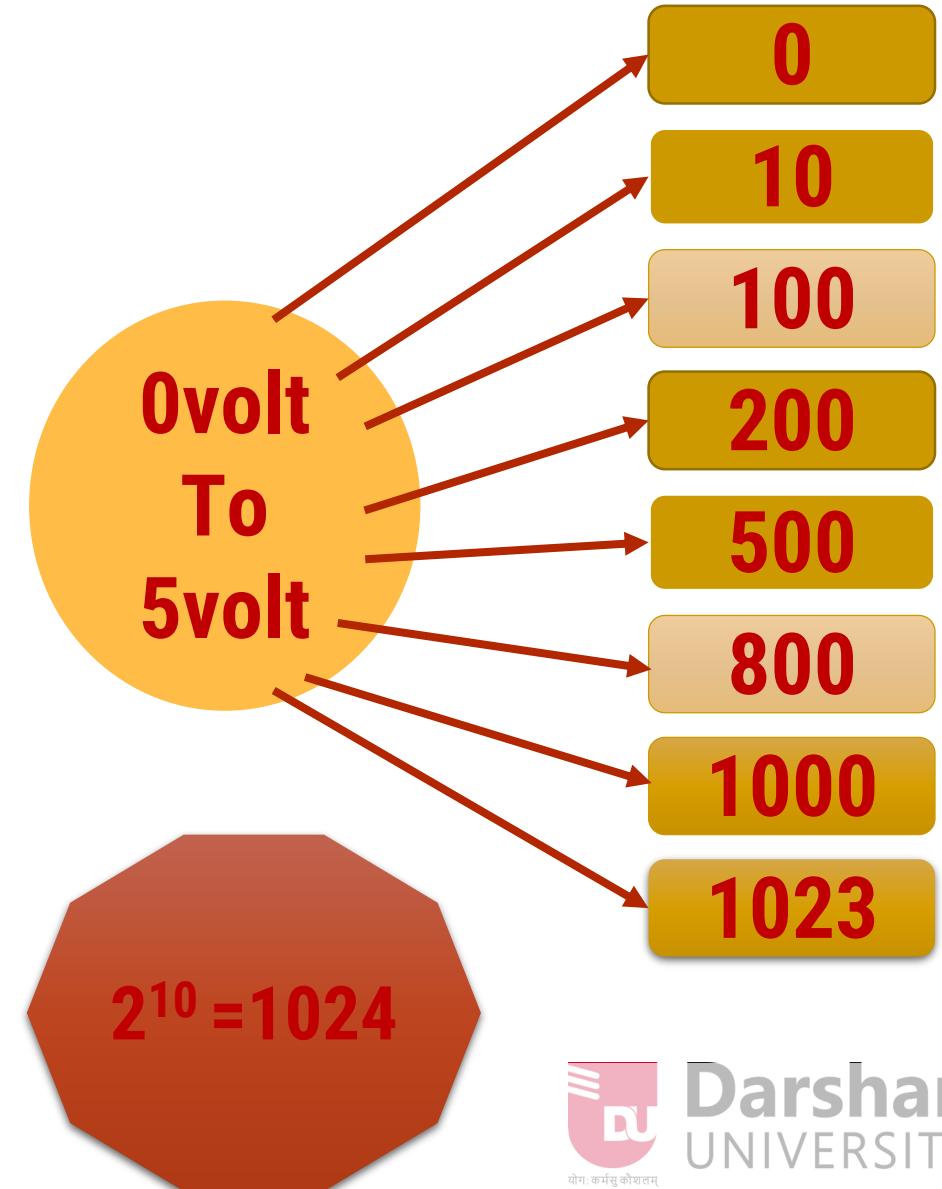
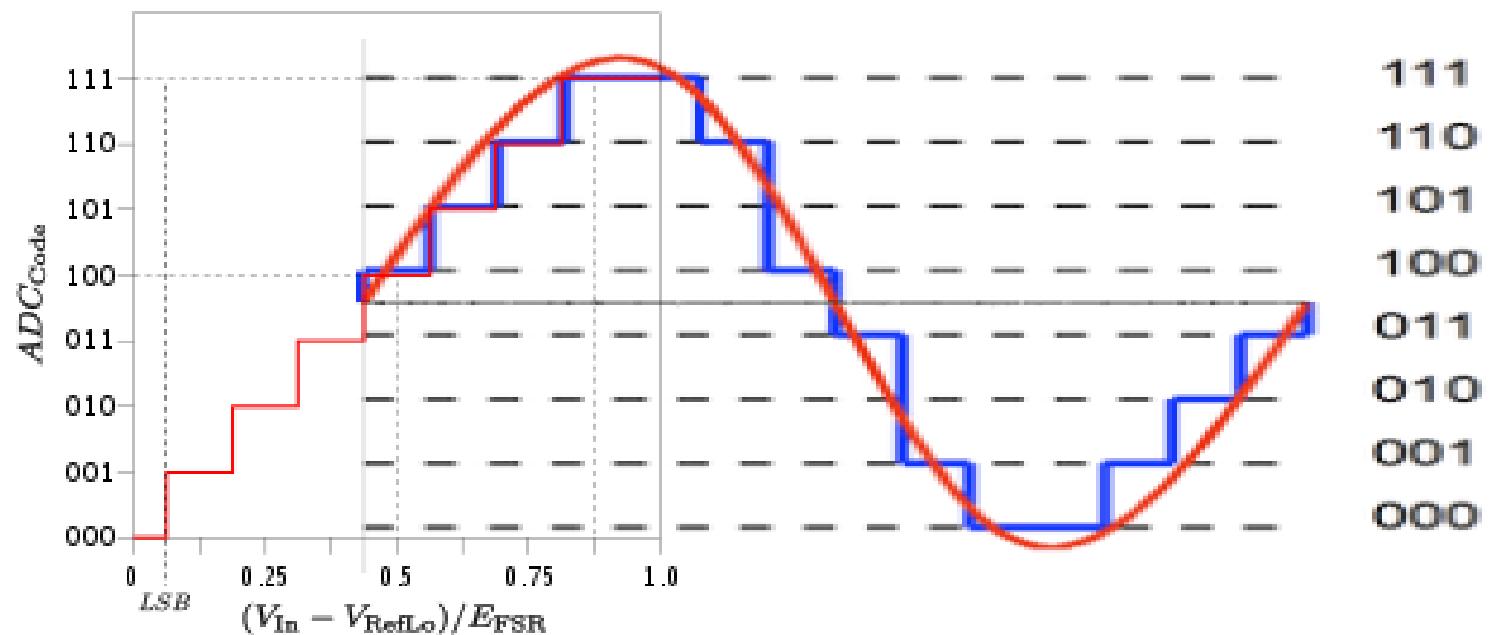
➤ Analog Input - Output

analogRead()

analogWrite()

analogReference()

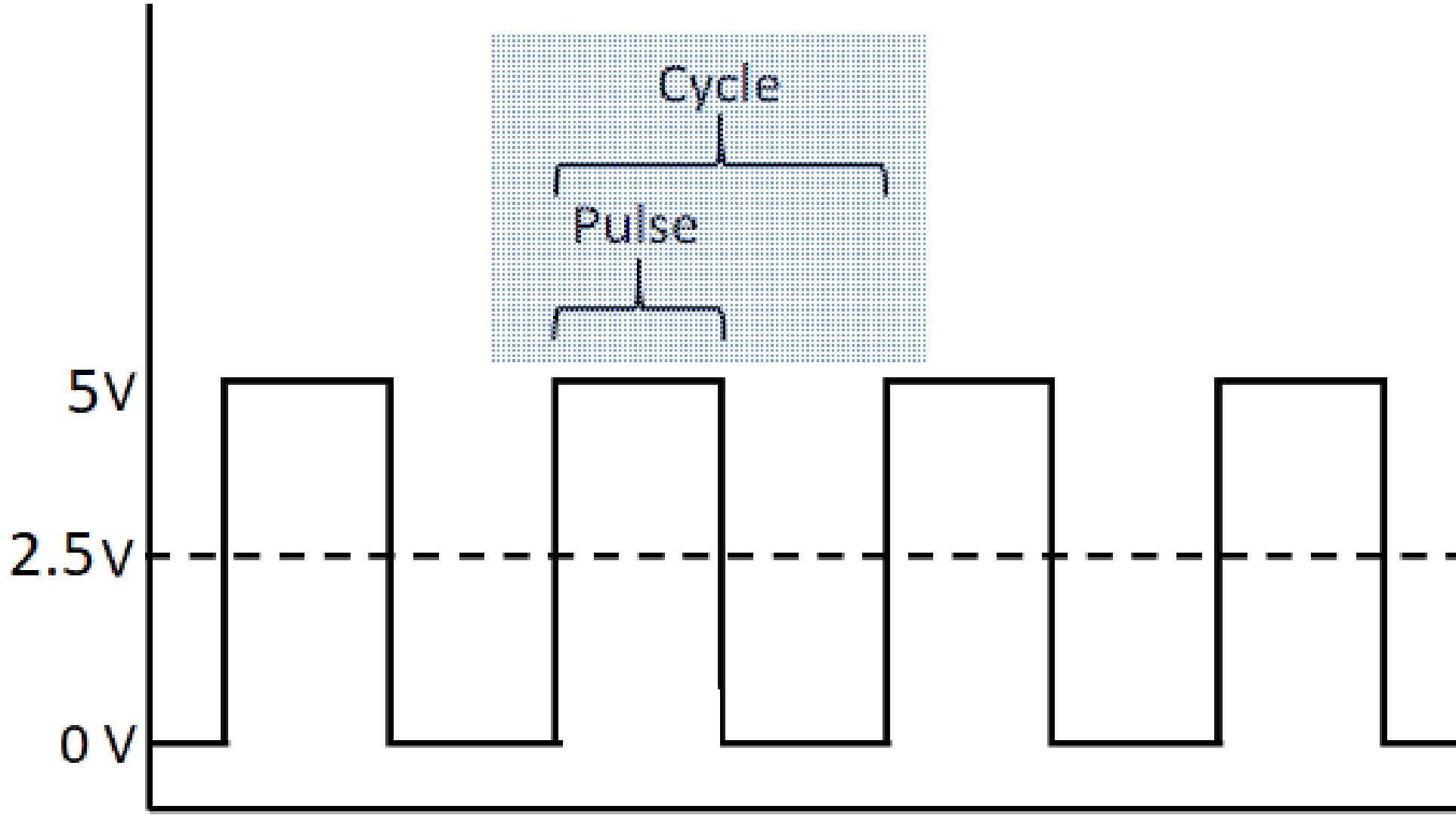
Analog Input with ADC : Analog to Digital



Digital Pin Output

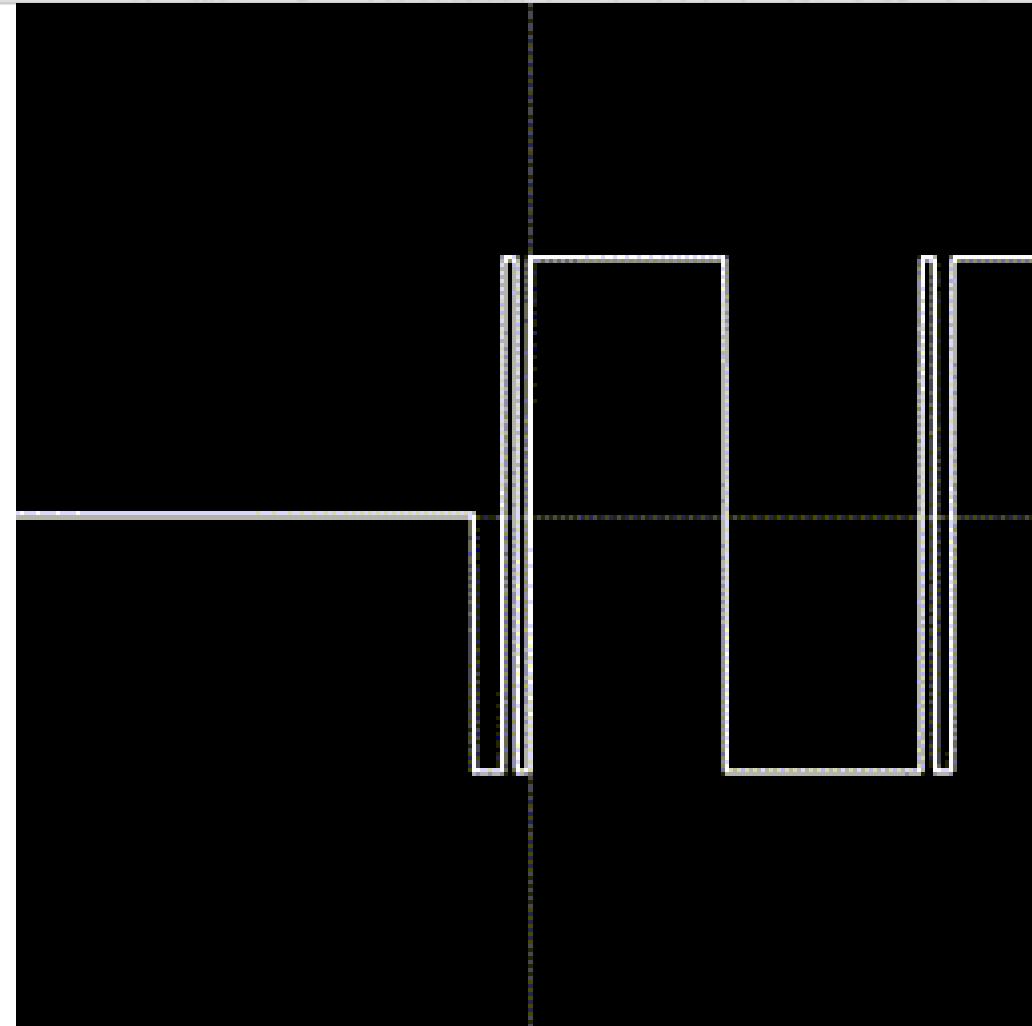


Digital Signal : Square wave



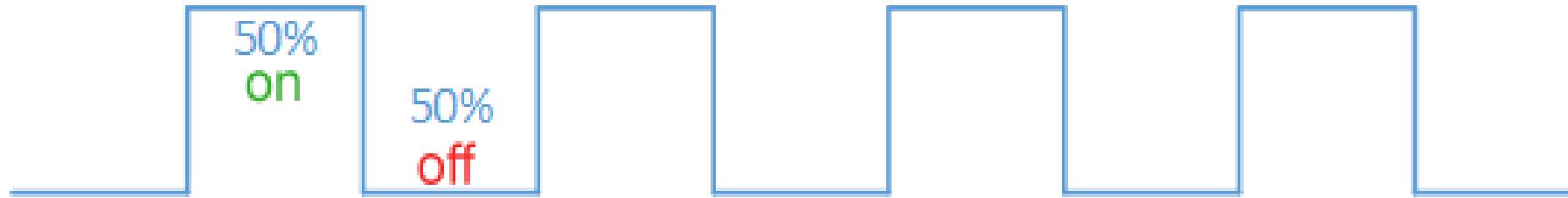
PWM : Pulse Width Modulation

- Pulse-width modulation (PWM), also known as pulse-duration modulation (PDM) or pulse-length modulation (PLM).
- It is a method of controlling the average power or amplitude delivered by an electrical signal.
- The average value of voltage (and current) fed to the load is controlled by switching the supply between 0 and 100%.

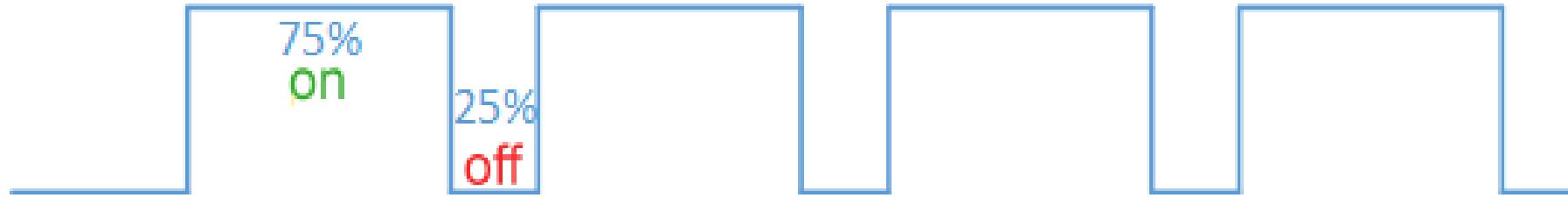


PWM : Pulse Width Modulation Duty Cycle variation

50% duty cycle



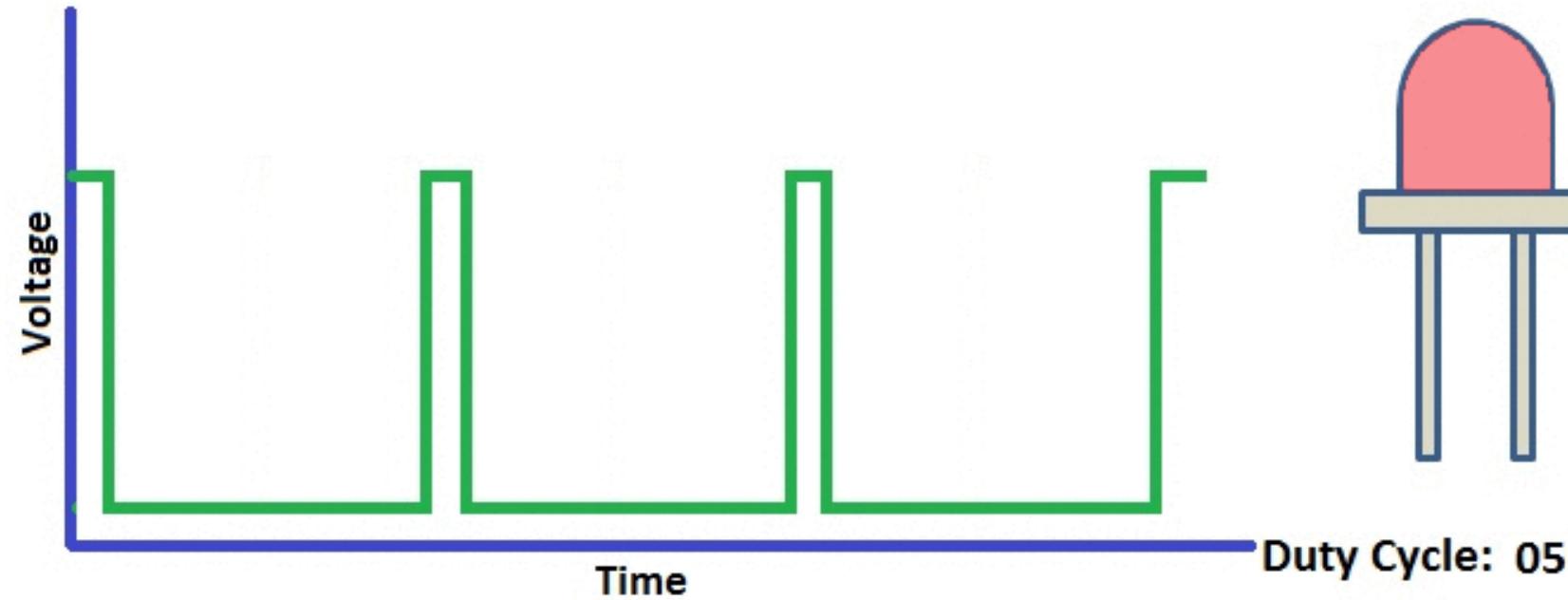
75% duty cycle



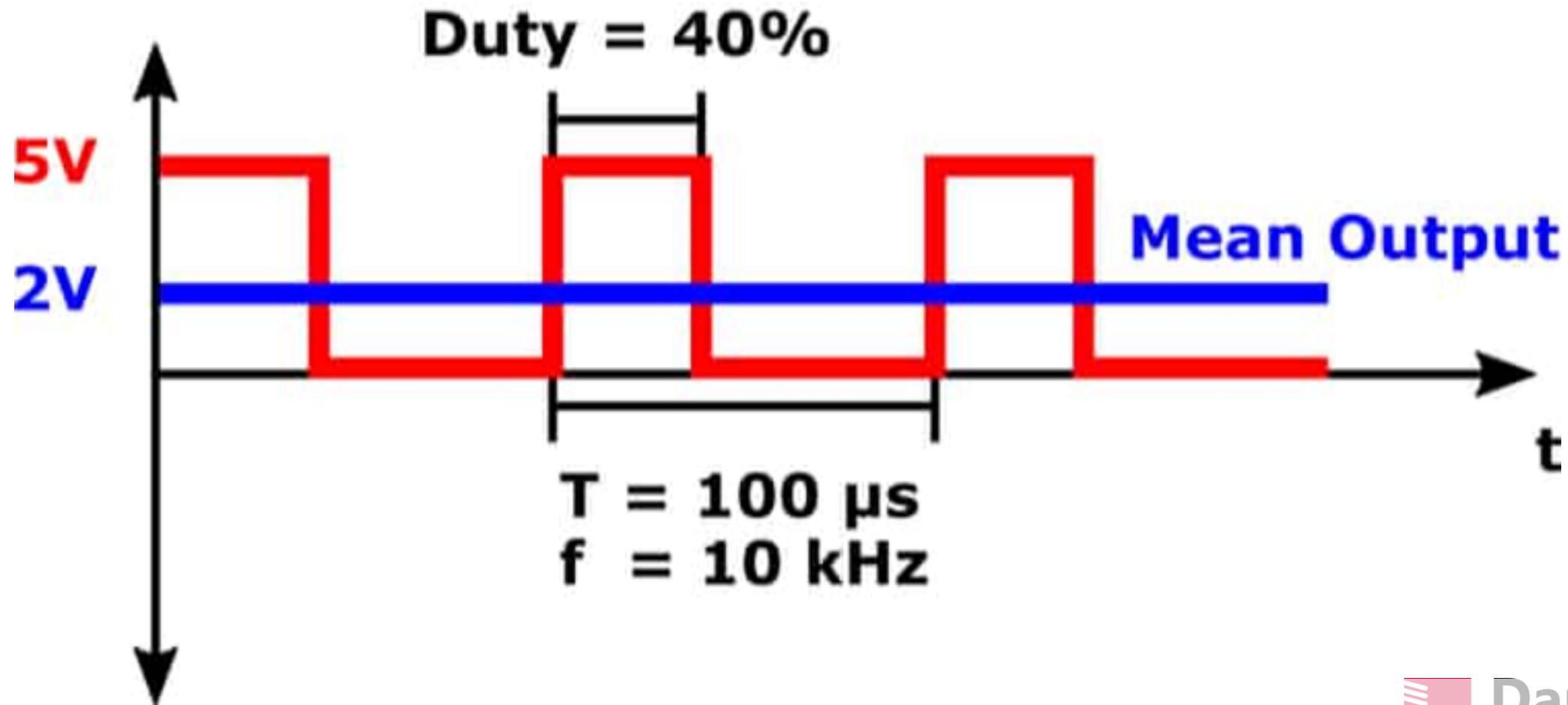
25% duty cycle



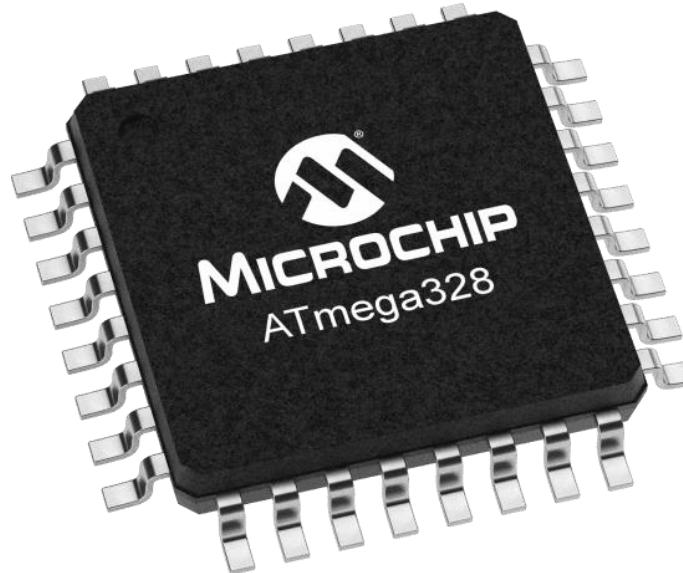
LED Fading with PWM



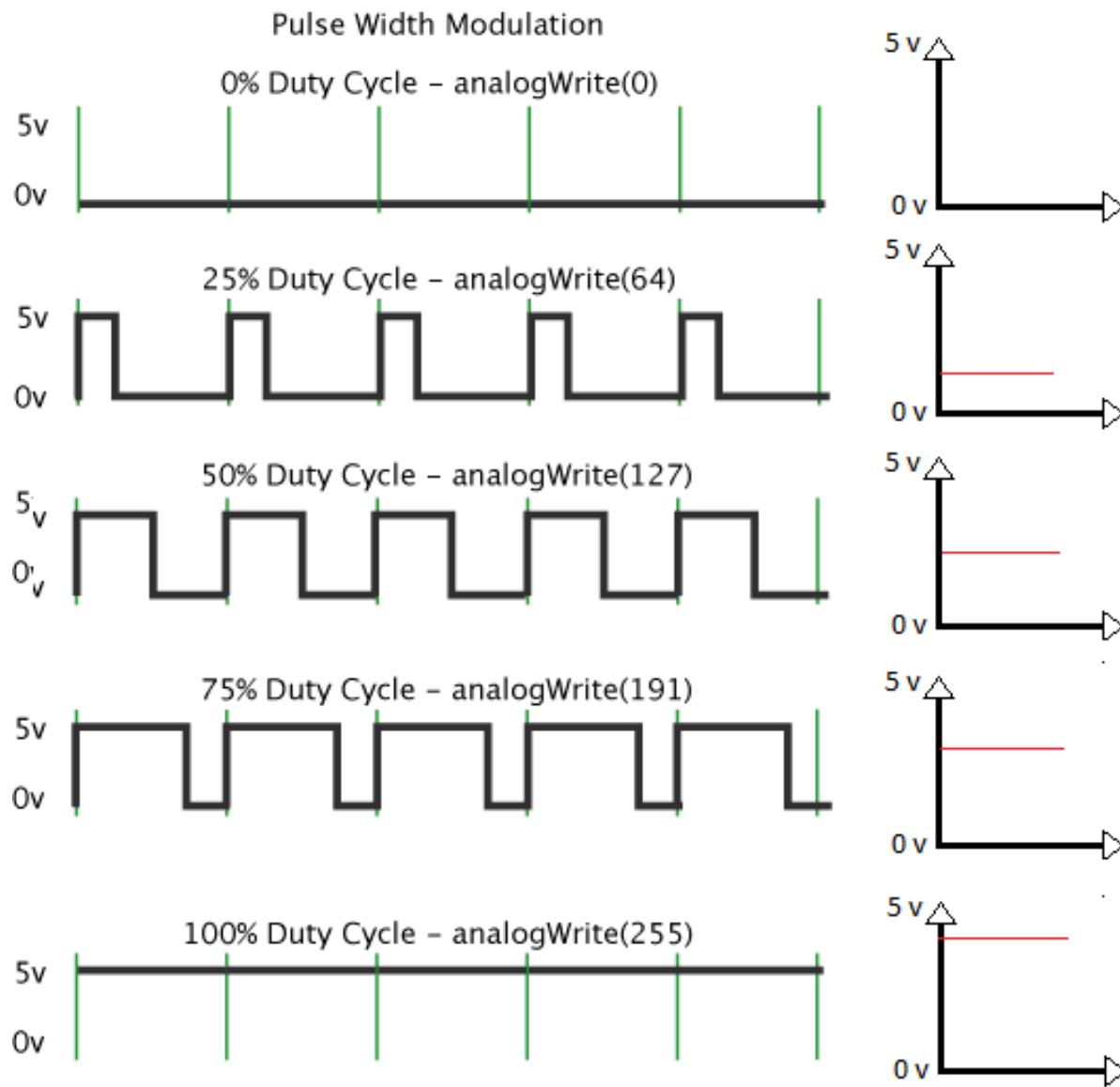
PWM Signal



Arduino UNO : Analog OUT

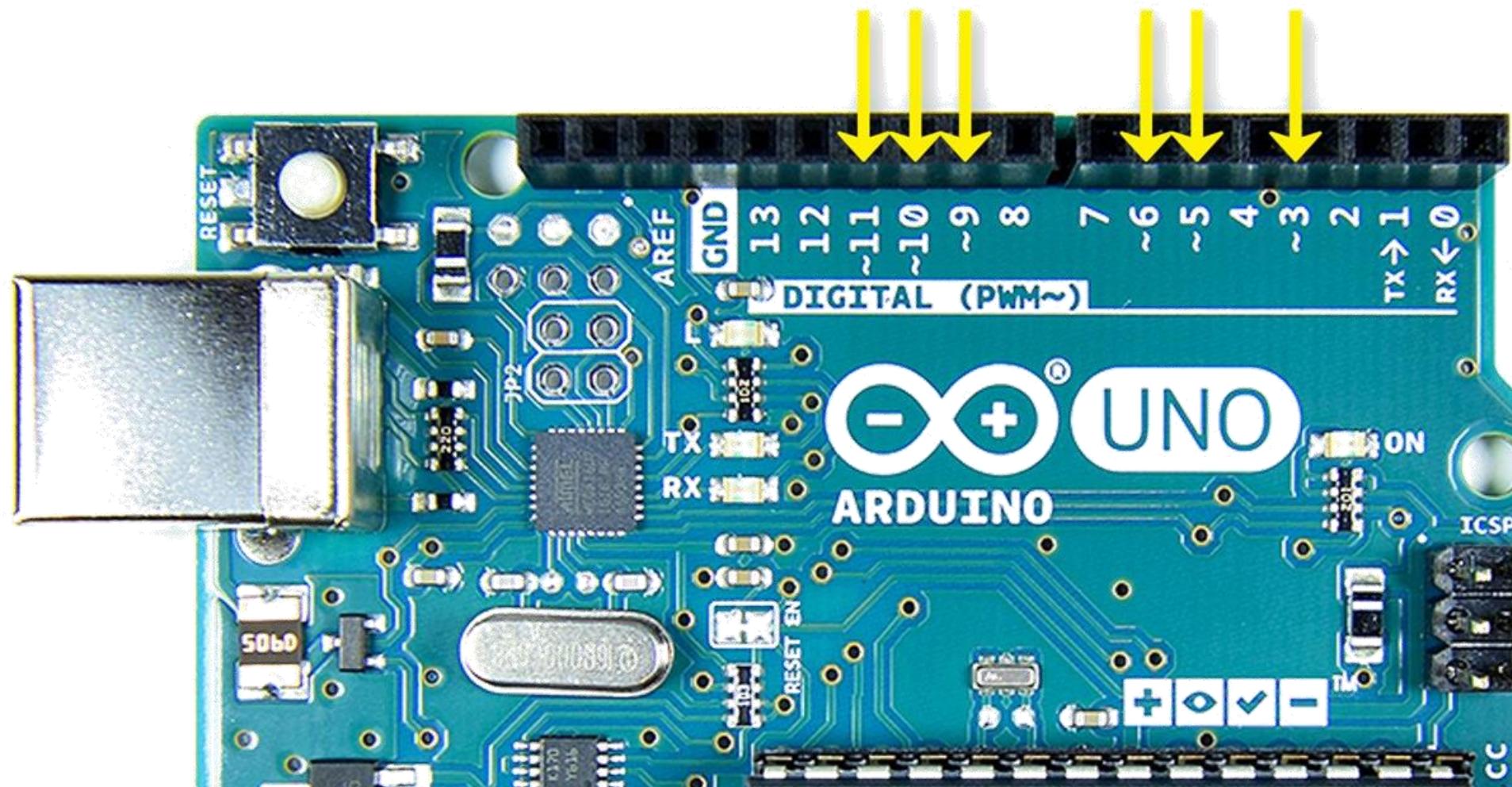


$$2^8 = 256$$



Arduino UNO :PWM

Pulse Width Modulation Pins



Analog Input

1. **analogRead()**:

- Reads the value from the specified analog pin.
- Arduino boards contain a multichannel, 10-bit **analog to digital converter**.
- This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values **between 0 and 1023**.

➤ **Syntax**

➤ **analogRead(pin)**

- **pin:** the name of the analog input pin to read from **(A0 to A5) (for Arduino UNO only)**

Program : Write a code to read data from gas sensor

gasSensor.ino

```
1 int gas_level;
2 void setup() {
3     pinMode(A0, INPUT);
4     Serial.begin(9600);
5 }
6 void loop() {
7     gas_level = analogRead(A0);
8     Serial.print("Gas Level : ");
9     Serial.println(gas_level);
10    delay(500);
11 }
12
13
14
15
16
```

Analog Output

2. analogWrite():

- Writes an analog value (PWM wave) to a pin.
- Can be used to light a LED at varying brightness or drive a motor at various speeds.
- After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() on the same pin.

➤ Syntax

➤ **analogWrite(pin, value)**

- **pin:** Arduino pin to write (Arduino UNO ~PWM : 3,5,6,9,10,11)
- **value:** 0-255

- The duty cycle: between 0 (always off) and 255 (always on).
- Allowed data types: int.

Program: Read from the potentiometer and set LED brightness accordingly.

potentioMeter.ino

```
1 ledPin = 9; // LED connected to digital pin 9
2 int analogPin = 3; // potentiometer connected to analog pin 3
3 int val = 0; // variable to store the read value
4
5 void setup() {
6     pinMode(ledPin, OUTPUT); // sets the pin as output
7 }
8 void loop() {
9     val = analogRead(analogPin); // read the input pin
10    analogWrite(ledPin, val / 4);
11
12 // analogRead values go from 0 to 1023
13 //analogWrite values from 0 to 255
14
15 }
16
```

Serial Communication Functions

➤ Serial:

- Used for communication between the Arduino board and a computer or other devices.

begin()

println()

print()

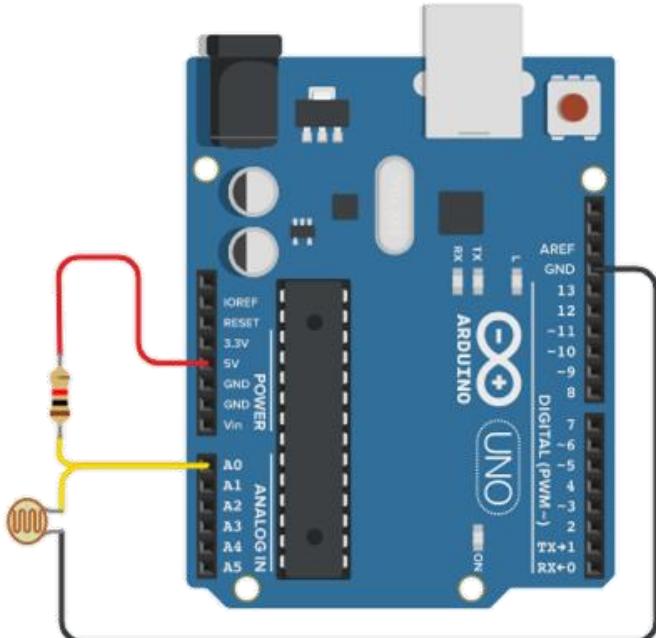
To start Serial communication

1. **Serial.begin()**

- It is used to start serial communication between Arduino board and other device.
- Normally, it is connected with computer for monitoring the data.
- Serial communication uses pin 0 and 1 also working as Rx and Tx respectively.
- **Syntax:**
 - **Serial.begin(baud_rate)**
 - **Serial.begin(speed, config)**
- **speed:** in bits per second (baud). Allowed data types: long.
- **config:** sets data, parity, and stop bits.

Functions in Arduino Communication Functions

➤ **Example:** LDR(Light Dependent Resistor) Sensor Interfacing with Arduino



LDR.ino

```
1 void setup(){
2     pinMode(A0, INPUT);
3     Serial.begin(9600); }
4 void loop(){
5     int light_intensity = analogRead(A0);
6     if (light_intensity > 800){
7         Serial.println("Darker");
8     }
9     else if (light_intensity < 500){
10        Serial.println("Too Bright");
11    }
12    delay(1000);
13 }
```

To print value on serial monitor.

2. **Serial.println()**

- Print the data with newline from Arduino to other device.
- Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').
- **Syntax:**
 - **Serial.println(val)**
 - **Serial.println(val, format)**
- **val:** the value to print. Allowed data types: any data type.
- **format:** specifies the number base (for integral data types) or number of decimal places (for floating point types).

To print value on serial monitor.

serialPrint.ino

```
1 int analogValue = 0; // variable to hold the analog value
2 void setup() {
3     Serial.begin(9600); // open the serial port at 9600 bps: }
4 void loop() {
5     analogValue = analogRead(A0);
6     // read the analog input on pin A0
7     // print it out in many formats:
8     Serial.println(analogValue); // print as an ASCII-encoded decimal
9     Serial.println(analogValue, DEC); // print as an ASCII-encoded decimal
10    Serial.println(analogValue, HEX); // print as an ASCII-encoded hexadecimal
11    Serial.println(analogValue, OCT); // print as an ASCII-encoded octal
12    Serial.println(analogValue, BIN); // print as an ASCII-encoded binary
13    // delay 10 milliseconds before the next reading:
14    delay(10); }
```

To print value on serial monitor

3. Serial.print()

➤ Print the data from Arduino to other device.

➤ Syntax

- Serial.print(val)
- Serial.print(val, format)

➤ val: the value to print.

➤ Allowed data types: any data type.

```
helloWorld.ino

1 void setup(){
2     Serial.begin(9600); }
3 void loop(){
4     Serial.print("Hello");
5     Serial.println(" Arduino");
6 }
7 }
```

Functions in Arduino IDE-Time Functions

➤ Time Functions:

delay()

delayMicroseconds()

millis()

micros()

Delay in milliseconds

1. **delay()**

➤ Pauses the program for the amount of time (in milliseconds) specified as parameter.

➤ **Syntax**

➤ **delay(ms)**

➤ **ms:** the number of milliseconds to pause.

➤ Allowed data types: unsigned long.

➤ **Example:**

```
void setup() {
    pinMode(13, OUTPUT);      // sets the digital pin 13 as output
}

void loop() {
    digitalWrite(13, HIGH);   // sets the digital pin 13 on
    delay(1000);             // waits for a second
    digitalWrite(13, LOW);    // sets the digital pin 13 off
    delay(1000);             // waits for a second
}
```

Delay in Microseconds

2. **delayMicroseconds()**

➤ It pauses the program for amount of time in microseconds specified as the parameter.

➤ **Syntax**

➤ **delayMicroseconds(μs)**

➤ **μs:** the number of microseconds to pause. Allowed data types: unsigned int.

➤ **Example:**

```
int outPin = 8;                      // digital pin 8

void setup() {
    pinMode(outPin, OUTPUT);        // sets the digital pin as output
}

void loop() {
    digitalWrite(outPin, HIGH);    // sets the pin on
    delayMicroseconds(50);         // pauses for 50 microseconds
    digitalWrite(outPin, LOW);     // sets the pin off
    delayMicroseconds(50);         // pauses for 50 microseconds
}
```

To keep the time since the Arduino program started.

3. millis()

- Returns the number of milliseconds passed since the Arduino board has began running the current program.
- This number will overflow (go back to zero), after approximately 50 days.
- **Syntax**

➤ **Int time = millis()**

4. micros()

- Returns the number of microseconds since the Arduino board began running the current program.
- This number will overflow (go back to zero), after approximately 70 minutes.
- **Syntax :**

▪ **int time = micros()**

Functions in Arduino IDE-Time Functions

Time.ino

```
1 unsigned long time1;
2 unsigned long time2;
3 void setup() {
4     Serial.begin(9600);
5 }
6 void loop() {
7     Serial.print("Time: ");
8     time1 = millis();
9     time2 = micros();
10    Serial.println(time1);
11 //prints time since program started in milliseconds
12    Serial.println(time2);
13 //prints time since program started in microseconds.
14    delay(1000);
15 //wait for 1 second so as not to send massive amounts of data
16 }
```

Functions in Arduino Math Functions

map()

max()

min()

pow()

sq()

sqrt()

1. map():

- Re-maps a number from one range to another.
- A value of from_Low would get mapped to to_Low, a value of from_High to to_High, values in-between to values in-between, etc.

➤ Syntax

➤ `map(value, from_Low , from_High , to_Low, to_High)`

- value: the number to map.
- from_Low: the lower bound of the value's current range.
- from_High: the upper bound of the value's current range.
- to_Low: the lower bound of the value's target range.
- to_High: the upper bound of the value's target range.

Mapping

```
/* Map an analog value from LDR to PWM output */  
  
void setup() {  
    pinMode(A0, INPUT);  
}  
  
void loop() {  
    int val = analogRead(A0);  
    val = map(val, 0, 1023, 0, 255);  
    analogWrite(9, val);  
}
```

LDR.ino

```
1 void setup(){  
2     pinMode(A0, INPUT);  
3     Serial.begin(9600); }  
4 void loop(){  
5     int light_intensity = analogRead(A0);  
6     if (light_intensity > 800){  
7         Serial.println("Darker");  
8     }  
9     else if (light_intensity < 500){  
10        Serial.println("Too Bright");  
11    }  
12    delay(1000);  
13 }  
14  
15  
16
```



Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

2. **max()**:

➤ Calculates the maximum of two numbers

➤ Syntax : **max(x, y)**

- x: the first number. Allowed data types: any data type.

- y: the second number. Allowed data types: any data type.

➤ It returns the larger of the two parameter values.

➤ Example:

- `sensVal = max(sensVal, 20); // assigns sensVal to the larger of sensVal or 20`

3. min():

- Calculates the minimum of two numbers.
- **Syntax** : `min(x, y)`
- `x`: the first number. Allowed data types: any data type.
- `y`: the second number. Allowed data types: any data type.
- It returns the smaller of the two parameter values.
- Example: `sensVal = min(sensVal, 100);` // assigns `sensVal` to the smaller of `sensVal` or `100`

4. **pow()**:

- Calculates the value of a number raised to a power. `pow()` can be used to raise a number to a fractional power.
- **Syntax** : `pow(base, exponent)`
- **base**: the number. Allowed data types: float.
- **exponent**: the power to which the base is raised. Allowed data types: float.
- The result of the exponentiation.
- Example: float z = `pow(x, y);`

Square of the Number

5. sq():

- Calculates the square of a number.
- **Syntax :** sq(x)
- x: the number. Allowed data types: any data type.
- The square of the number.
- **Example:**

```
square.ino
1 void setup() {
2     Serial.begin(9600);
3     int x = 4;
4     double y = sq(x);
5     Serial.print("The square of ");
6     Serial.print(x);
7     Serial.print(" is ");
8     Serial.println(y);
9 }
```

Square root of the Number

6. **sqrt()**:

- Calculates the square root of a number.
- **Syntax :** `sqrt(x)`
- `x`: the number. Allowed data types: any data type.
- The square root of the number.
- **Example:**

sqrt.ino

```
1 void setup() {  
2     Serial.begin(9600);  
3     int x = 4;  
4     double y = sqrt(x);  
5     Serial.print("The square root of ");  
6     Serial.print(x);  
7     Serial.print(" is ");  
8     Serial.println(y);  
9 }
```

Advanced Input Output Functions

pulseIn()

random()

randomSeed()

Measure duration of HIGH or LOW pulse

1. **pulseIn()**

- Reads a pulse HIGH or LOW from specified pin and wait for the time to go the pulse from HIGH to LOW or LOW to HIGH.
- Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.
- Works on pulses from 10 microseconds to 3 minutes in length.
- **Syntax**

- **pulseIn(pin, value)**
- **pulseIn(pin, value, timeout)**

- pin: the number of the Arduino pin on which you want to read the pulse.
- value: type of pulse to read: either HIGH or LOW.
- timeout (optional): the number of microseconds to wait for the pulse to start; default is one second.

Example : pulseIn()

pulseIn.ino

```
1 int pin = 7;
2 unsigned long duration;
3 void setup() {
4     Serial.begin(9600);
5     pinMode(pin, INPUT);
6 }
7 void loop() {
8     duration = pulseIn(pin, HIGH);
9     Serial.println(duration);
10 }
```

Random number generation

2. random()

➤ The random function generates pseudo-random numbers.

➤ Syntax

➤ random(max)

➤ random(min, max)

➤ min: lower bound of the random value, inclusive (optional).

➤ max: upper bound of the random value, exclusive.

3. randomSeed()

➤ it initializes the pseudo-random number generator.

Example: random number generation

RandomNumber.ino

```
1 long ranNumber;
2 void setup() {
3     Serial.begin(9600);
4     randomSeed(analogRead(A0));
5 }
6 void loop() {
7     ranNumber = random(300);
8     // generate a random number from 0 to 299
9     Serial.println(ranNumber);
10    ranNumber = random(10, 20);
11    // print a random number from 10 to 19
12    Serial.println(ranNumber);
13    delay(50);
14 }
15
16
```

Unit 2-C: Raspberry Pi

Introduction to Raspberry Pi

General Architecture of Raspberry Pi

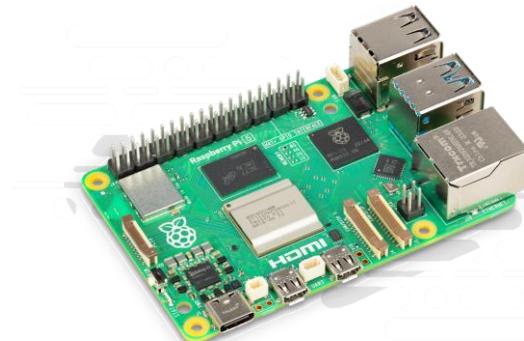
Installing OS in Raspberry Pi

Introduction to Python

Programming Raspberry Pi

Raspberry Pi

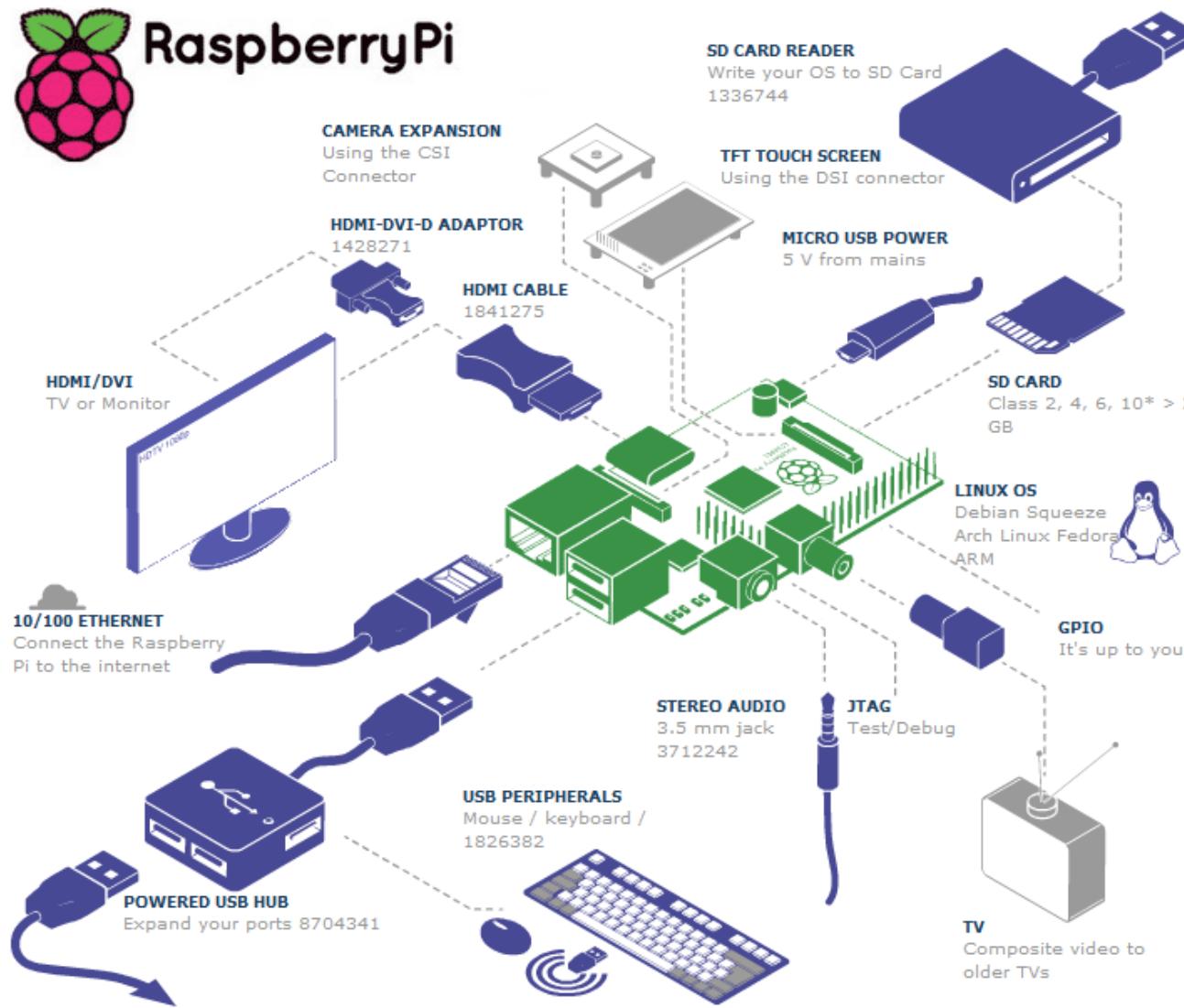
- Raspberry Pi is a single-board computers, developed by Raspberry Pi Foundation in association with **Broadcom**.
- Raspberry pi perhaps the most inspiring computer available today.
 - ❖ Because of its low cost and open design, the model became far more popular than anticipated.
 - ❖ It is widely used to make gaming devices, fitness gadgets, weather stations, and much more.
- In 2012, the company launched the Raspberry Pi and the current generations of regular Raspberry Pi boards are Zero, 1, 2, 3,4 and 5.



Raspberry Pi 5



Architecture, Layout and Interface of Raspberry Pi



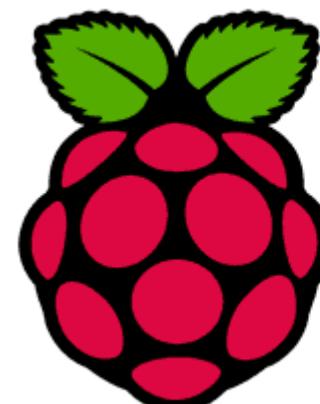
- Processor
- HDMI Port
- USB Port
- Ethernet Port
- SD Card Slot
- Camera Connector
- Composite Video Output
- Audio Output
- Micro USB Power
- GPIO Pins
- Status LEDs

Operating System for Raspberry Pi

- Raspberry Pi needs an operating system to work.
- Raspberry Pi OS (previously called Raspbian) is an official supported operating system.
- Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi.
- Download and install Raspberry Pi Imager to a computer with an SD card reader.
- Put the SD card into card reader and Run Raspberry Pi Imager.

NOOBS

- NOOBS (New Out Of the Box Software) is an exclusive install manager for OS installation process in the SD card of Raspberry Pi.
- NOOBS is extremely useful for beginners as it helps in easy OS installation without configuration process.
- Different Operating system that could be installed using NOOBS are:
 - Raspbian Raspberry Pi OS
 - OpenELEC
 - Pidora
 - RaspBMC
 - RISC OS
 - Arch Linux ARM

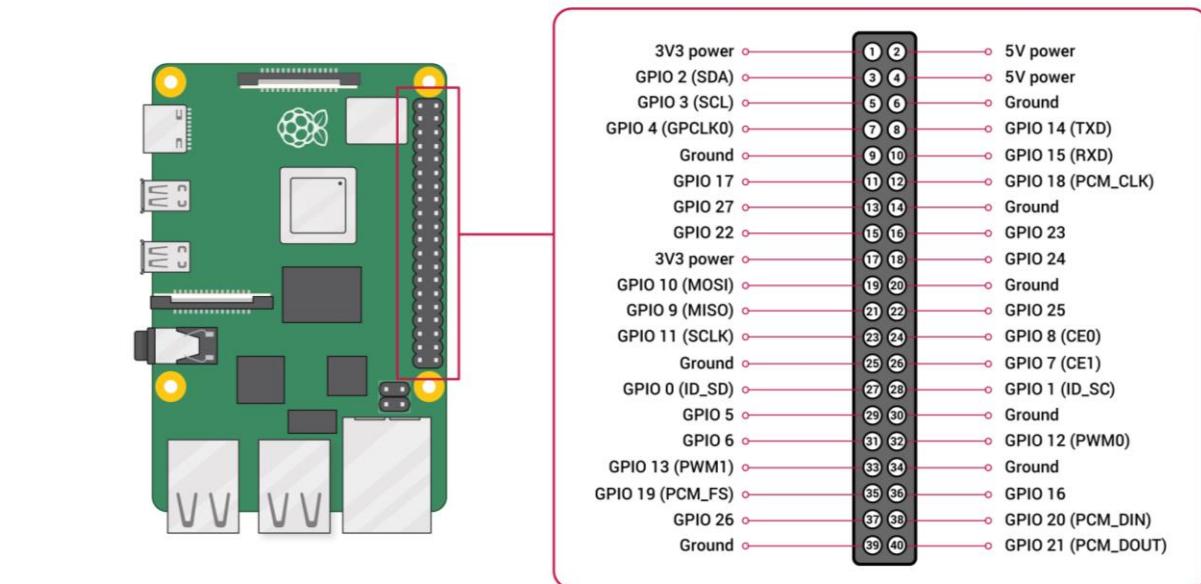


Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

GPIO and the 40-pin Header

- A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board.
- A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Raspberry Pi Zero, Raspberry Pi Zero W and Raspberry Pi Zero 2 W).
- Prior to the Raspberry Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header.
- One of the things that makes the Raspberry Pi better for learning electronics than most other computers is its ability to control the voltage on several of its easily accessible pins.
- GPIO header contains outputs for 3.3V, 5V, Ground, and lots of General Purpose Input/Output (GPIO) pins!



Programming

- Python is a powerful programming language that's easy to use, easy to read and write with Raspberry Pi.
- Python syntax is clean, with an emphasis on readability.
- Python uses standard English keywords.
- To control hardware connected to the Raspberry Pi we will use Python.
- One should be familiar with some of the basics of Python, including literals, variables, operators, control flow, scope, and data structures.
- We'll use the RPi.GPIO module to control all the GPIO of Raspberry Pi
- Pin Out in detail : <https://pinout.xyz/>

- What do these numbers mean?
- GPIO - General Purpose Input/Output, aka "BCM" or "Broadcom".
 - ❖ These are the big numbers, e.g. "GPIO 22".
 - ❖ You can use these with RPi.GPIO and GPIO Zero.
- Physical - or "Board" correspond to the pin's physical location on the header.
 - ❖ These are the small numbers next to the header, e.g. "Physical Pin 15".
- WiringPi - for Gordon Henderson's Wiring Pi library. (Like Arduino lang. digitalRead,digitalWrite)

40 Pin (Raspberry Pi 4)

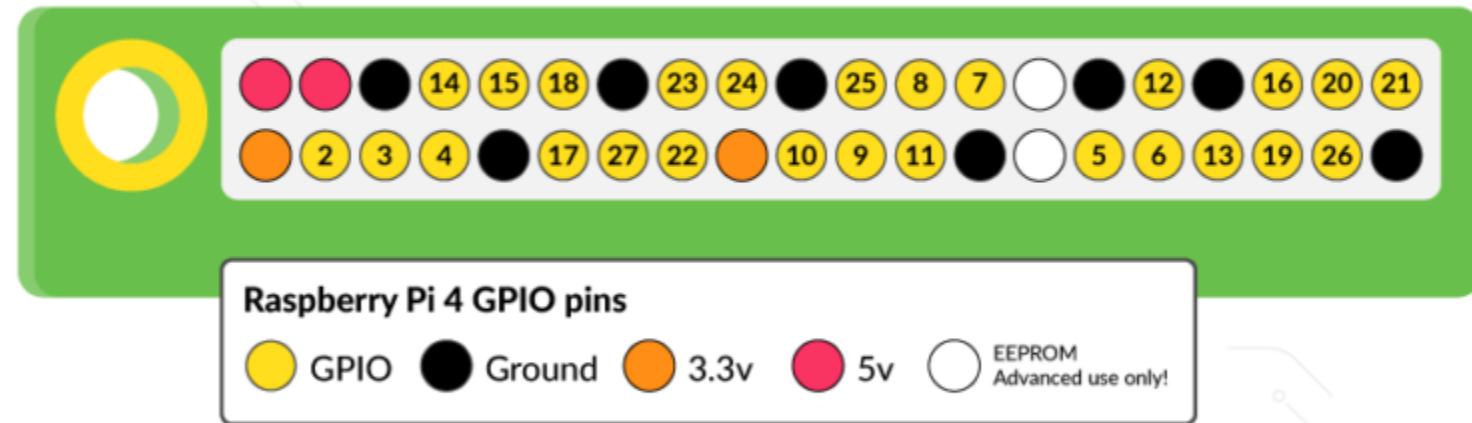
| PIN | NAME | | NAME | PIN |
|-----|--------------------------------|--|---------------------|-----|
| 01 | 3.3V DC Power | | 5V DC Power | 02 |
| 03 | GPIO02 (SDA1,I ² C) | | 5V DC Power | 04 |
| 05 | GPIO03 (SDL1,I ² C) | | Ground | 06 |
| 07 | GPIO04 (GPCLK0) | | GPIO14 (TXD0, UART) | 08 |
| 09 | Ground | | GPIO15 (RXD0, UART) | 10 |
| 11 | GPIO17 | | GPIO18(PWM0) | 12 |
| 13 | GPIO27 | | Ground | 14 |
| 15 | GPIO22 | | GPIO23 | 16 |
| 17 | 3.3V DC Power | | GPIO24 | 18 |
| 19 | GPIO10 (SP10_MOSI) | | Ground | 20 |

40 Pin (Raspberry Pi 4)

| | | | | |
|----|--------------------|--|---------------------|----|
| 21 | GPIO09 (SP10_MISO) | | GPIO25 | 22 |
| 23 | GPIO11 (SP10_CLK) | | GPIO08 (SPI0_CE0_N) | 24 |
| 25 | Ground | | GPIO07 (SPI0_CE1_N) | 26 |
| 27 | GPIO00 (SDA0, I²C) | | GPIO01 (SCL0, I²C) | 28 |
| 29 | GPIO05 | | Ground | 30 |
| 31 | GPIO06 | | GPIO12 (PWM0) | 32 |
| 33 | GPIO13 (PWM1) | | Ground | 34 |
| 35 | GPIO19 | | GPIO16 | 36 |
| 37 | GPIO26 | | GPIO20 | 38 |
| 39 | Ground | | GPIO21 | 40 |



40 Pin (Raspberry Pi 4)



Python Programming for GPIO

- Import the Rpi.GPIO module
 - **import RPi.GPIO as GPIO**
- Pin Numbering Declaration
 - After you've included the **RPi.GPIO** module, the next step is to determine which of the two pin-numbering schemes you want to use:
 - **GPIO.BOARD** -- Board numbering scheme. (Physical)
 - The pin numbers follow the pin numbers on header P1.
 - **GPIO.BCM** -- Broadcom chip-specific pin numbers. (Broadcom)
 - These pin numbers follow the lower-level numbering system defined by the Raspberry Pi's Broadcom-chip brain.
- To specify in your code which number-system is being used, use the **GPIO.setmode()** function.
 - **GPIO.setmode(GPIO.BCM)**
- Both the **import** and **setmode** lines of code are required, if you want to use Python.

➤ Setting a Pin Mode

- Similar to declare a "pin mode" in Arduino before you can use it as either an input or output.
- To set a pin mode, use the `setup([pin], [GPIO.IN, GPIO.OUT])` function.
- So, if you want to set pin 18 as an output,
 - `GPIO.setup(18, GPIO.OUT)`

➤ Digital Outputs

- To write a pin high or low, use the `GPIO.output([pin], [GPIO.LOW, GPIO.HIGH])` function.
- For example, if you want to set pin 18 high, write:
 - `GPIO.output(18, GPIO.HIGH)`
- Writing a pin to `GPIO.HIGH` will drive it to 3.3V, and `GPIO.LOW` will set it to 0V.
- Alternative to `GPIO.HIGH` and `GPIO.LOW`, you can use either `1`, `True`, `0` or `False` to set a pin value

➤ Analog Outputs

- PWM ("Analog") Output
- PWM on the Raspberry Pi is about as limited as can be – one
- Single pin is capable of it: 18 (i.e. board pin 12).
- To initialize PWM, use `GPIO.PWM([pin], [frequency])` function.
- To make the rest of your script-writing easier you can assign that instance to a variable.
- Then use `pwm.start([duty cycle])` function to set an initial value.
- For example...

➤ `pwm = GPIO.PWM(18, 1000)`
➤ `pwm.start(50)`

Python Programming for GPIO

➤ Inputs

- If a pin is configured as an input, you can use the `GPIO.input([pin])` function to read its value.
- The `input()` function will return either a True or False indicating whether the pin is HIGH or LOW.
- You can use an if statement to test this, will read pin 17 and print whether it's being read as HIGH or LOW

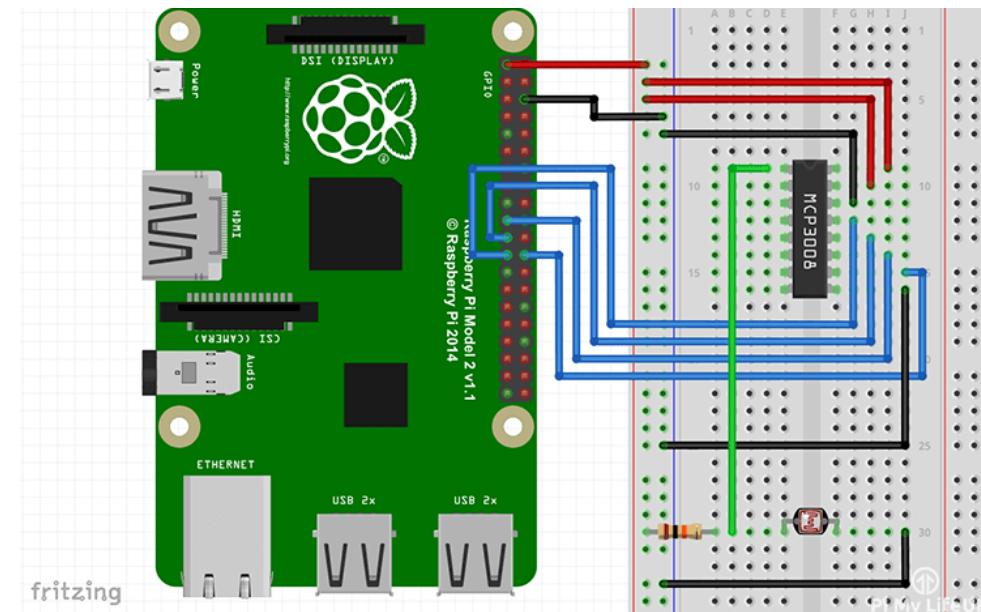
```
if GPIO.input(17):  
    print("Pin 11 is HIGH")  
else:  
    print("Pin 11 is LOW")
```

| Feature | Code Example |
|----------------|--|
| Set mode | <code>GPIO.setmode(GPIO.BCM)</code> |
| Set output pin | <code>GPIO.setup(18, GPIO.OUT)</code> |
| Set input pin | <code>GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)</code> |
| Read input | <code>GPIO.input(17)</code> |
| Write output | <code>GPIO.output(18, GPIO.HIGH)</code> |
| PWM | <code>pwm = GPIO.PWM(18, 100)</code> |
| Cleanup | <code>GPIO.cleanup()</code> |

| Feature | Code Example |
|--------------------|--|
| Set input pin | <code>GPIO.setup(17, GPIO.IN)</code> |
| Set with pull-up | <code>GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)</code> |
| Set with pull-down | <code>GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)</code> |
| Read input | <code>GPIO.input(17)</code> |
| Detect event | <code>GPIO.add_event_detect(17, GPIO.FALLING, callback=my_callback, bouncetime=300)</code> |

Analog Input

- The Raspberry Pi does not have built-in analog-to-digital converters (ADC) like an Arduino.
 - Its GPIO pins can only handle digital signals (HIGH or LOW).
 - To read analog sensors (e.g., potentiometers, temperature sensors, light sensors), you need an external ADC.
 - Recommendation : The MCP3008 chip is a 10-bit digital-to-analog converter that reads analog signals and sends them to a microcontroller via SPI communication protocol.



Python Code for Blinking LED

Blink.py

```
1 import time
2 import RPi.GPIO as GPIO
3 led_pin = 12 # Pin definitions
4 GPIO.setmode(GPIO.BCM) # Use "GPIO" pin numbering
5 GPIO.setup(led_pin, GPIO.OUT) # Set LED pin as output
6
7 while True: # Blink forever
8     GPIO.output(led_pin, GPIO.HIGH) # Turn LED on
9     time.sleep(1) # Delay for 1 second
10    GPIO.output(led_pin, GPIO.LOW) # Turn LED off
11    time.sleep(1) # Delay for 1 second
12
```

Python Code for Blinking LED

Blink.py

```
1 #import the GPIO and time package
2 import RPi.GPIO as GPIO
3 import time
4 GPIO.setmode(GPIO.BARD)
5 GPIO.setup(7, GPIO.OUT)
6
7 # loop through 50 times, on/off for 1 second
8
9 for i in range(50):
10     GPIO.output(7, True)
11     time.sleep(1)
12     GPIO.output(7, False)
13     time.sleep(1)
14     GPIO.cleanup() # Delay for 1 second
15
```

**Thank
You**



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot
 bhushan.joshi@darshan.ac.in
 +91 8485979997



Unit-3

Sensors, Microcontrollers and Interfacing



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot

 bhushan.joshi@darshan.ac.in
 +91 8485979997



Sensors, Microcontrollers and interfacing

IoT Data communication and Interfacing

Serial Communication - UART

SPI Protocol

I2C Protocol

MQTT Protocol

Sensors and Sensor Modules interfacing

IoT Data communication and Interfacing : Protocols

**IoT Data
Communication
Interfaces:**

- **UART Serial communication**
- **SPI**
- **I2C**

**IoT Messaging
Protocols:**

- **MQTT**
- **CoAP**
- **AMQP**

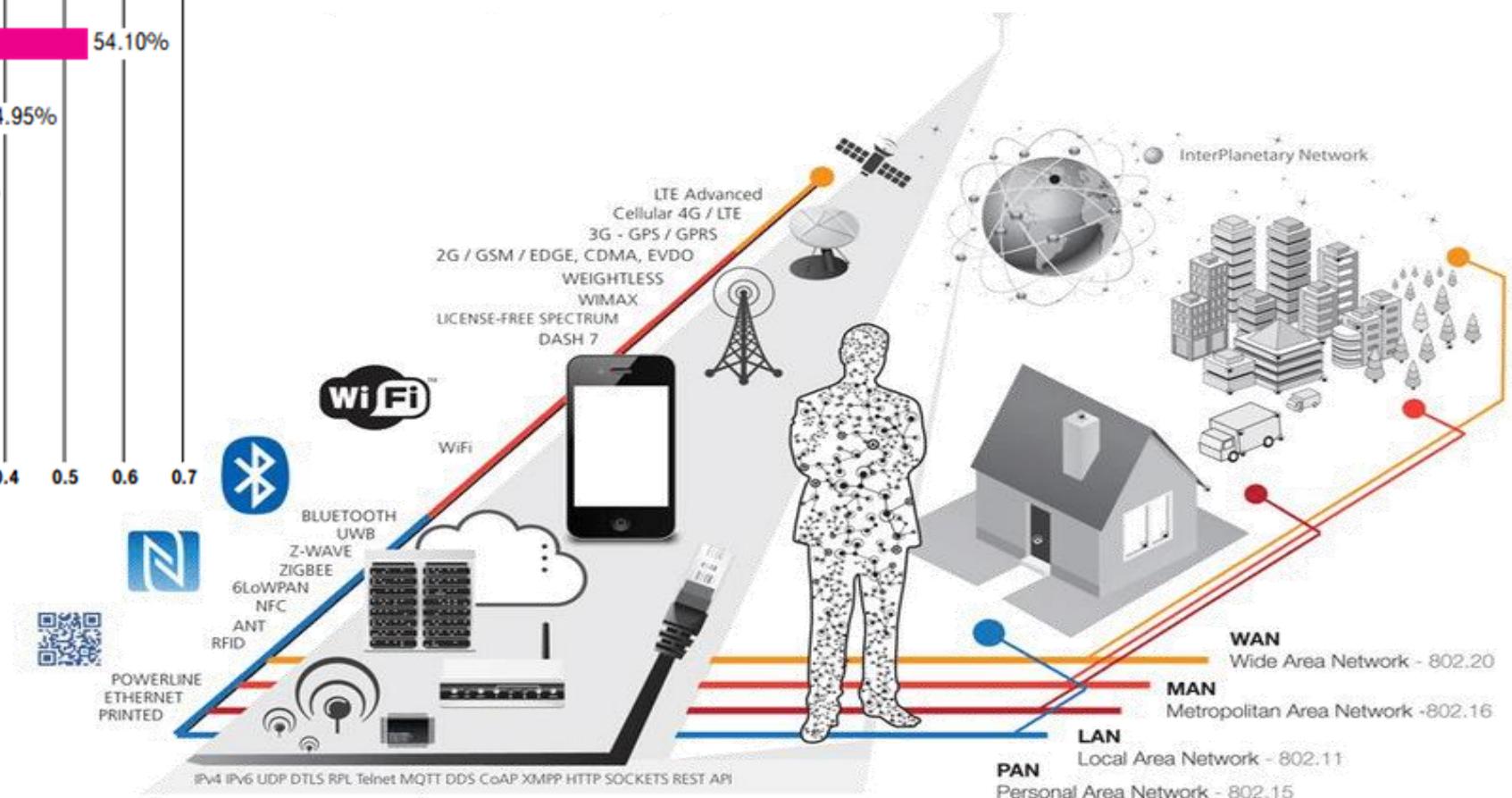
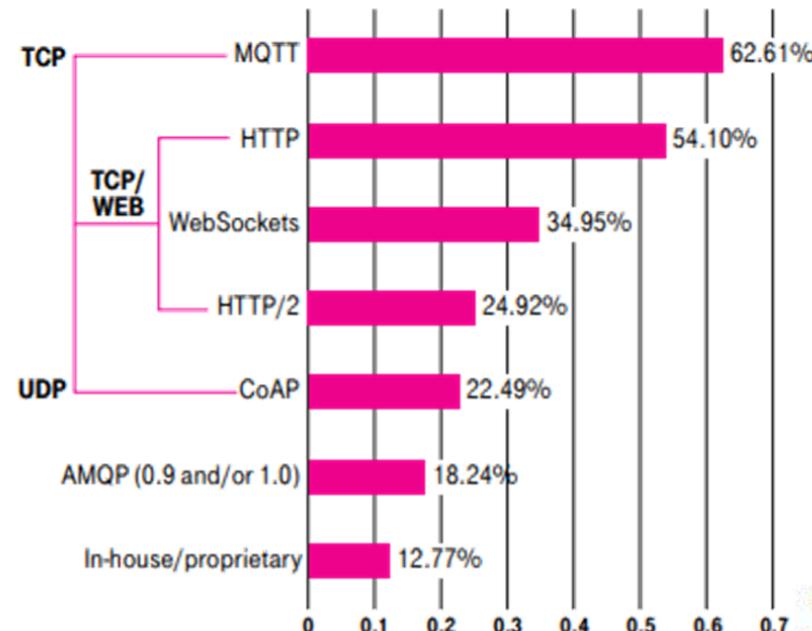
**IoT Network
Protocols:**

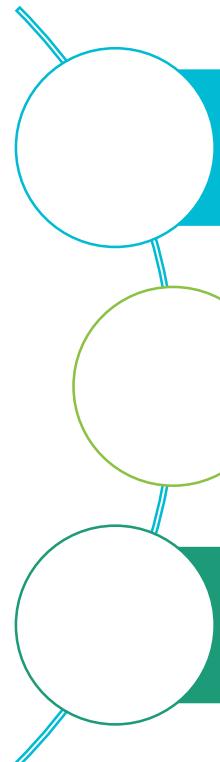
- **Bluetooth Low Energy**
- **Zigbee with Xbee Module.**

IoT Communication

- The prime focus of the internet of things is to offer communication **between various objects that are not traditional computers.**
- The connection of devices **over the internet.**
- These smart devices communicate with each other , exchange data , perform some tasks without any human involvement.
- These devices are embedded with electronics, software, network and sensors which help in communication.
- There are various protocols in IoT established to offer communication among them.
- Communication enables these devices to gather, exchange data which contribute in success of that IoT product/project.
- An "**IoT protocol**" is a set of rules and standards that govern how devices within an Internet of Things (IoT) network communicate and exchange data with each other.

IoT Communication





Asynchronous Serial Communication : UART

SPI (Serial Peripheral Interface) Protocol

I2C- Protocol

Serial Communication

- In Embedded Systems, Telecommunication, and Data Transmission applications, **Serial Communication** is known to be the process of **sending data one bit at a time (bit-by-bit) sequentially**, over the serial bus.
- It takes a **complete clock cycle** in order to transfer each bit from a one end to the other.
- Types:
 - ❖ Synchronous: Data is transmitted with a clock signal (e.g., SPI, I²C).
 - ❖ Asynchronous: No clock signal is required; devices agree on data rate (e.g., UART).

Serial Communication vs Parallel Communication

➤ Conversely, Parallel Communication is known to be the process of sending several bits, even bytes, as a whole only in a single clock cycle.

➤ Advantages of Serial Communication

- ❖ Simpler wiring and reduced cost.
- ❖ Suitable for long-distance communication (e.g., USB, RS-485).
- ❖ Less chance of signal interference.

➤ Advantages of Parallel Communication

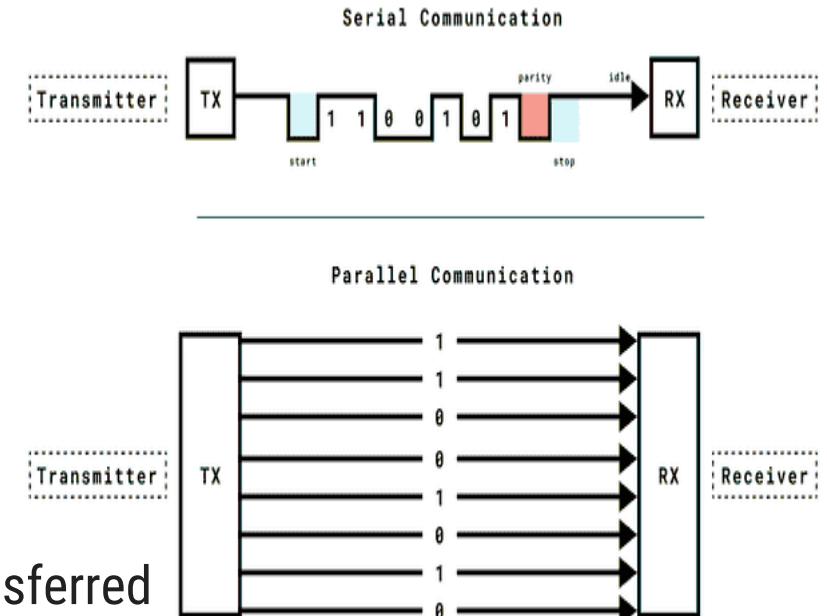
- ❖ Faster data transfer rates (used in high-speed buses like PCIe).
- ❖ Suitable for applications requiring large amounts of data to be transferred quickly over short distances (e.g., inside computers).

➤ Serial Communication: Arduino to sensors or PCs via UART, SPI, I²C.

- ❖ Long-distance communication (e.g., RS-232, Ethernet, USB).

➤ Parallel Communication: Internal hardware communication in computers (e.g., CPU to RAM via memory buses).

- ❖ Printers (parallel ports like Centronics in older models).



Serial Communication : UART

- **Universal Asynchronous Receiver-Transmitter (UART)**
- Universal Asynchronous Receiver-Transmitter (UART), a **serial communication protocol** that can be used to send data between an Arduino board and other devices.
- It allows an **asynchronous** serial communication in which the **data format** and **transmission speed** are configurable.
- UART is one of the most used device-to-device (serial) communication protocols.
- It is very simple, low-cost and easy to implement.



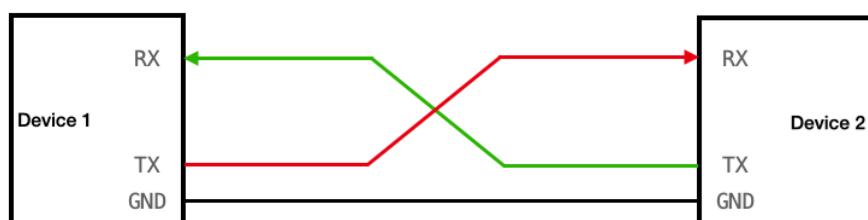
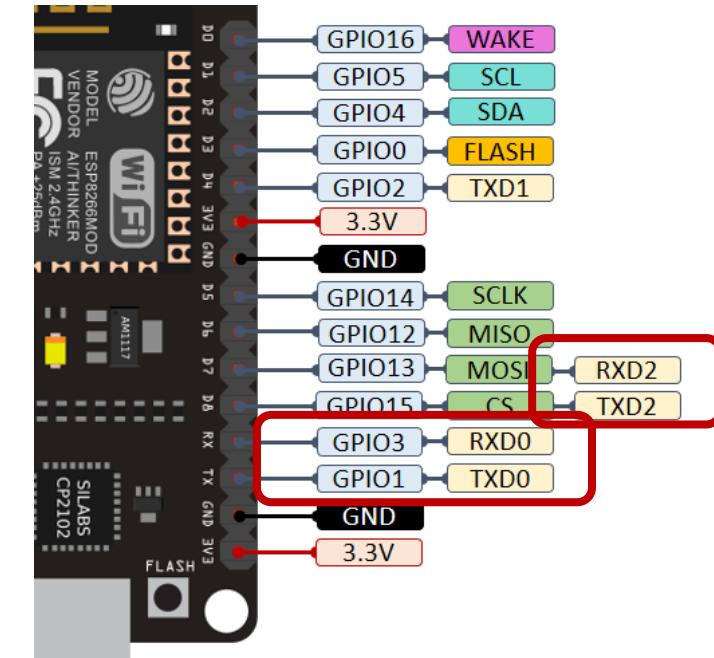
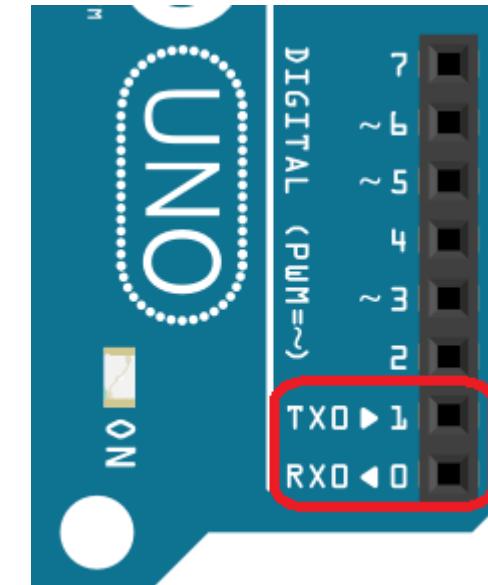
Arduino : Serial Class

- Communication via UART is enabled by the Serial class, which has a number of methods available, including reading & writing data.
- The Serial class have several methods with some of the essentials being:
 - ❖ `begin()` - begins serial communication, with a specified baud rate
(many examples use either 9600 or 115200).
 - ❖ `print()` - prints the content to the Serial Monitor.
 - ❖ `println()` - prints the content to the Serial Monitor, and adds a new line.
 - ❖ `available()` - checks if serial data is available or not.
(if you send a command from the Serial Monitor).
 - ❖ `read()` - reads data from the serial port.
 - ❖ `write()` - writes data to the serial port.

Arduino UART Pins

- The default TX/RX pins on an Arduino board are the D0(RX) and D1(TX) pins.
- Some boards have additional serial ports, see table below:

| Form Factor | RX | TX | RX1 | TX1 | RX2 | TX2 | RX3 | TX3 |
|-------------|----|----|-----|-----|-----|-----|-----|-----|
| MKR | D0 | D1 | | | | | | |
| UNO | D0 | D1 | | | | | | |
| Nano | D0 | D1 | | | | | | |
| Mega | D0 | D1 | D19 | D18 | D17 | D16 | D15 | D14 |



How UART Works ?

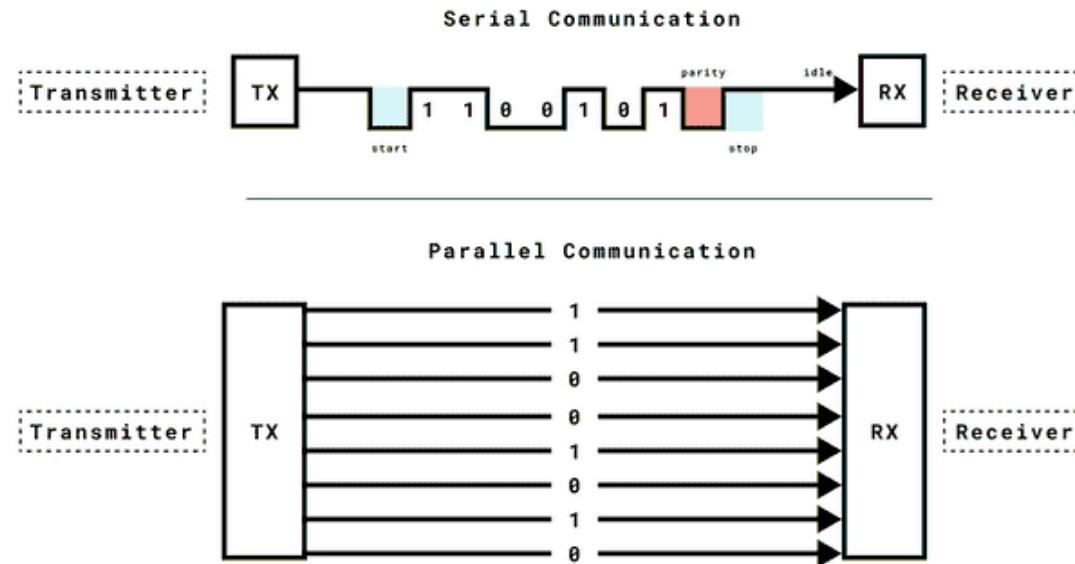
➤ UART operates by transmitting data as a series of bits, including

- ❖ Start bit
- ❖ data bits
- ❖ an optional parity bit,
- ❖ stop bit(s).

➤ Unlike parallel communication, where multiple bits are transmitted simultaneously, UART sends data serially, one bit at a time.

➤ As the name reveals the protocol operates asynchronous which means that it doesn't rely on a shared clock signal.

➤ Instead, it uses predefined baud rates to determine the timing of data bits.



UART Communication

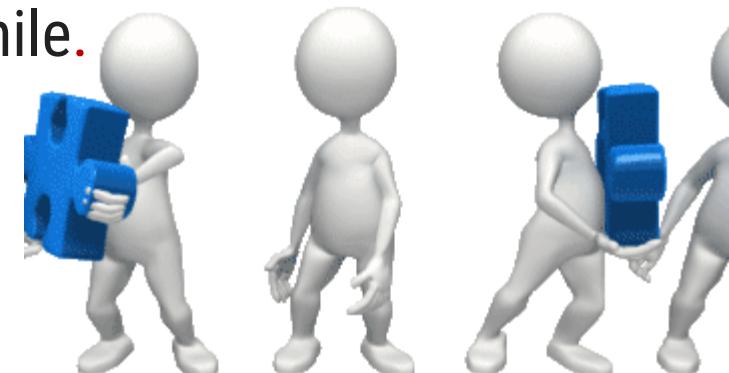
➤ Key Components:

- ❖ 1)The Transmitter 2)The Receiver 3)The Baud Rate

- The transmitter collects data from a source, formats it into serial bits, and sends it via a TX (Transmit) pin.
- The receiver receives it via a RX (Receive) pin, processes incoming serial data and converts it into parallel data for the host system.
- **Baud Rate :**The baud rate is specified in bits per second (bps) and represents the number of bits transmitted in one second.
- The baud rate determines the speed of data transmission.
- **In UART, both the transmitting and receiving devices must agree on the same baud rate to ensure successful communication.**

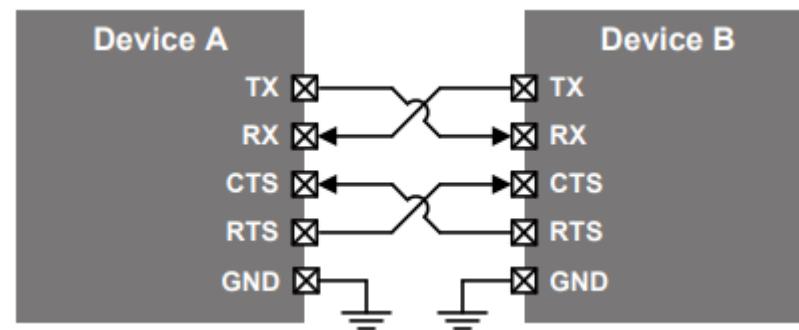
UART : Flow Control

- **UART Flow Control** is a method for **slow and fast devices** to communicate with each other over UART without the risk of losing data.
- Flow control provides extra signalling to inform the transmitter that it should stop (pause) or start (resume) the transmission.
- Consider the case where two units are communicating over UART.
 - ❖ a **transmitter T** is sending a long stream of bytes to a **receiver R**.
 - ❖ R is a slower device than T, and R cannot keep up.
 - ❖ R needs to either do some processing on the data or empty some buffers before it can keep receiving data.
 - ❖ R needs to tell T to **stop** transmitting for a while.



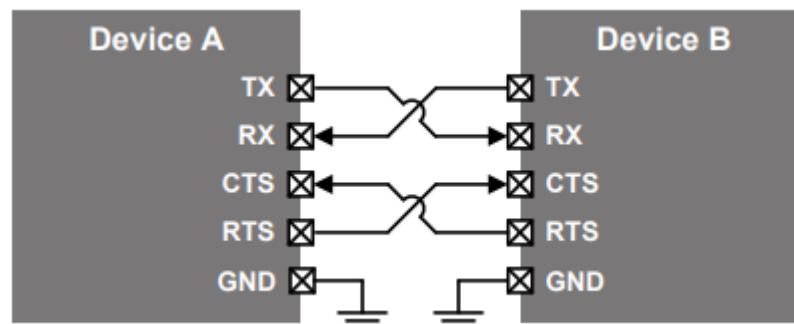
UART : Flow Control

- Several forms of flow control exist.
 - ❖ **Hardware flow control :**
 - It uses **extra wires**, where the logic level on these wires define whether the transmitter should **keep sending data or stop**.
 - ❖ **Software flow control :**
 - **Special characters** are sent over the normal data lines to **start or stop** the transmission.
- UART hardware flow control is fully supported by UART driver, and UART software flow control is partially supported by the driver.



UART : Hardware flow control (also called RTS/CTS flow control)

- Two extra wires are needed in addition to the data lines.
- They are called **RTS** (Request to Send) and **CTS** (Clear to Send).
- These wires are cross-coupled between the two devices, so RTS on one device is connected to CTS on the remote device and vice versa.



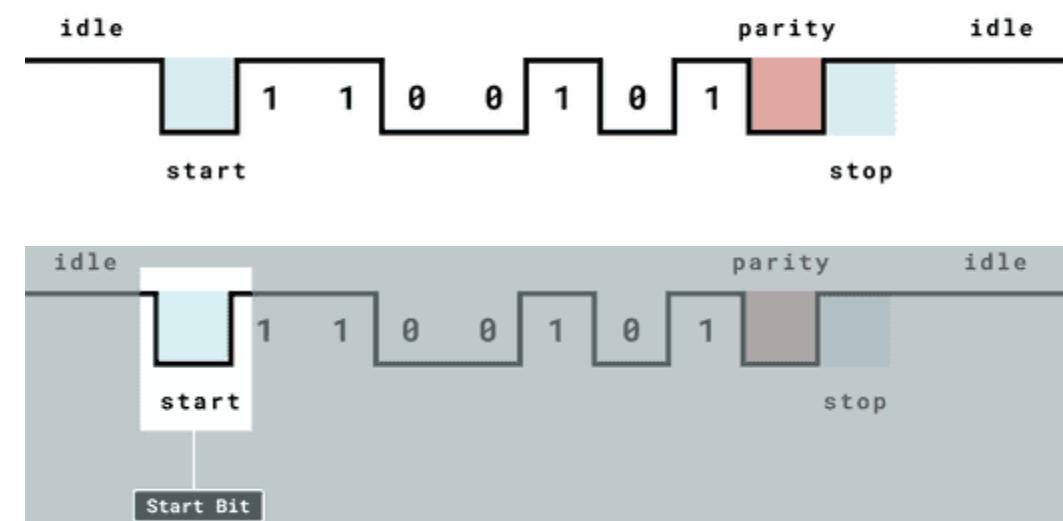
UART Messages

- Each data frame is encapsulated by start and stop bits.
- These bits serve a vital role in establishing the boundaries of data transmission and ensuring synchronization between the sender and receiver.

- **Start Bit :**

- A single start bit is transmitted at the beginning of each UART frame.
- The **start bit is always logic low (0)** for UART communication.
- When the **receiver detects a start bit, it knows that a new data frame is beginning**, and it prepares to receive the incoming bits.

UART Frame Format

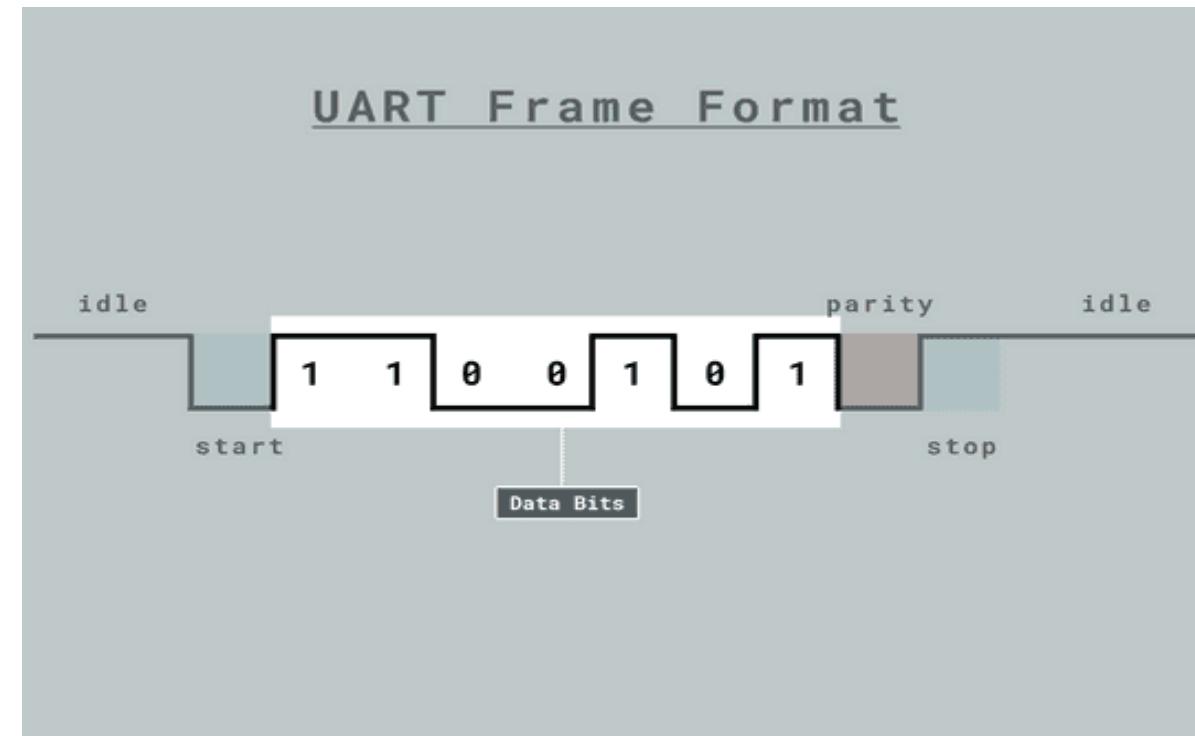


UART Messages

- **Data Bits :**
- Data bits are a fundamental component of UART communication as they carry the actual information to be transmitted.
- The number of data bits in a UART frame can vary, but a common and widely used configuration is **8 bits or 7-bit or 6-bit**.
- **Character Size :**
- The character size in UART communication is defined by the number of data bits within a frame.
- **8-Bit:** This is the most prevalent character size in UART communication.
- **7-Bit:** In cases where data size needs to be

smaller, 7-bit character size is utilized.

- **6-Bit:** For even more compact data representation.

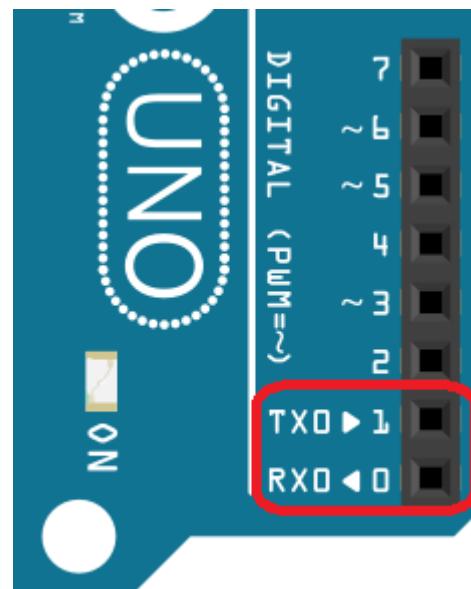


UART Communication : Data Encoding And Parity

- **Data Encoding :**
- Data bits represent the characters or data using **binary** encoding, where each bit corresponds to a power of 2.
- This encoding allows for the transmission of a wide range of information, making UART versatile for various data types, from simple text characters to complex binary data.
- **Parity Bit :**
- The parity bit is a bit added to the transmitted data and tells the receiver if the number of 1's in the data transmitted is odd or even.
- The possible setting for Parity Bit is Odd or Even.
 - ❖ ODD – the parity bit is '**1**' if there is an **odd** number of 1's in the data frame
 - ❖ EVEN – the parity bit is '**0**' if there is an **even** number of 1's in the data frame

Arduino UNO : Serial Communication

- Arduino has one or more UART pins depending on the board.
- Arduino **Uno** has only one UART interface found on **pin 0 (RX0)** and **pin 1 (TX0)**.
- Figure shows the location of the UART TX and RX pins.
- The Arduino pins 0 and 1 are also used for communicating with the Arduino IDE via the USB.
- **So if you will upload sketches to your UNO, be sure to first disconnect any wires on pins 0 and 1**



In Arduino serial communication, the default number of data bits is 8

Serial Functions

Check the Port

- `if(Serial)`
- `available()`
- `availableForWrite()`

Start / Stop

- `begin()`
- `end()`

Print on Serial Monitor

- `print()`
- `println()`

Read

- `read()`
- `peek()`
- `readBytes()`
- `readBytesUntil()`
- `readString()`
- `readStringUntil()`
- `flush()`

Write Data

- `write()`

Communication Functions

- **Serial.begin()** : Initializes serial communication
- **Serial.end()** : Ends serial communication
- **Serial.available()** : Gets the number of bytes available for reading
- **Serial.read()** : Reads **1Byte** serial data
- **Serial.peek()** : Reads **1Byte** serial data **without removing** it from the buffer
- **Serial.flush()** : Waits for the transmission of the serial data to complete
- **Serial.write()** : Data transmission
- **Serial.print()** : Send string (no line feed)
- **Serial.println()** : Send string (with line feed)

Serial Port availability

Read Data

sketch_01.ino

```
1 void setup() {  
2 //Initialize serial  
3 //and wait for port to  
4 //open:  
5 Serial.begin(9600);  
6 while (!Serial) { ;  
7 // wait for serial  
//port to connect.  
9 //Needed for native  
//USB  
11 }  
12 }
```

sketch_01.ino

```
1 int incomingByte = 0;  
2 // for incoming serial data  
3 void setup() {  
4 Serial.begin(9600);  
5 // sets data rate to 9600 bps  
6 }  
7 void loop() {  
8 // reply only when you receive data:  
9 if (Serial.available() > 0) {  
10 // read the incoming byte:  
11 incomingByte = Serial.read();  
12 // say what you got:  
13 Serial.print("I received: ");  
14 Serial.println(incomingByte, DEC);  
15 }
```



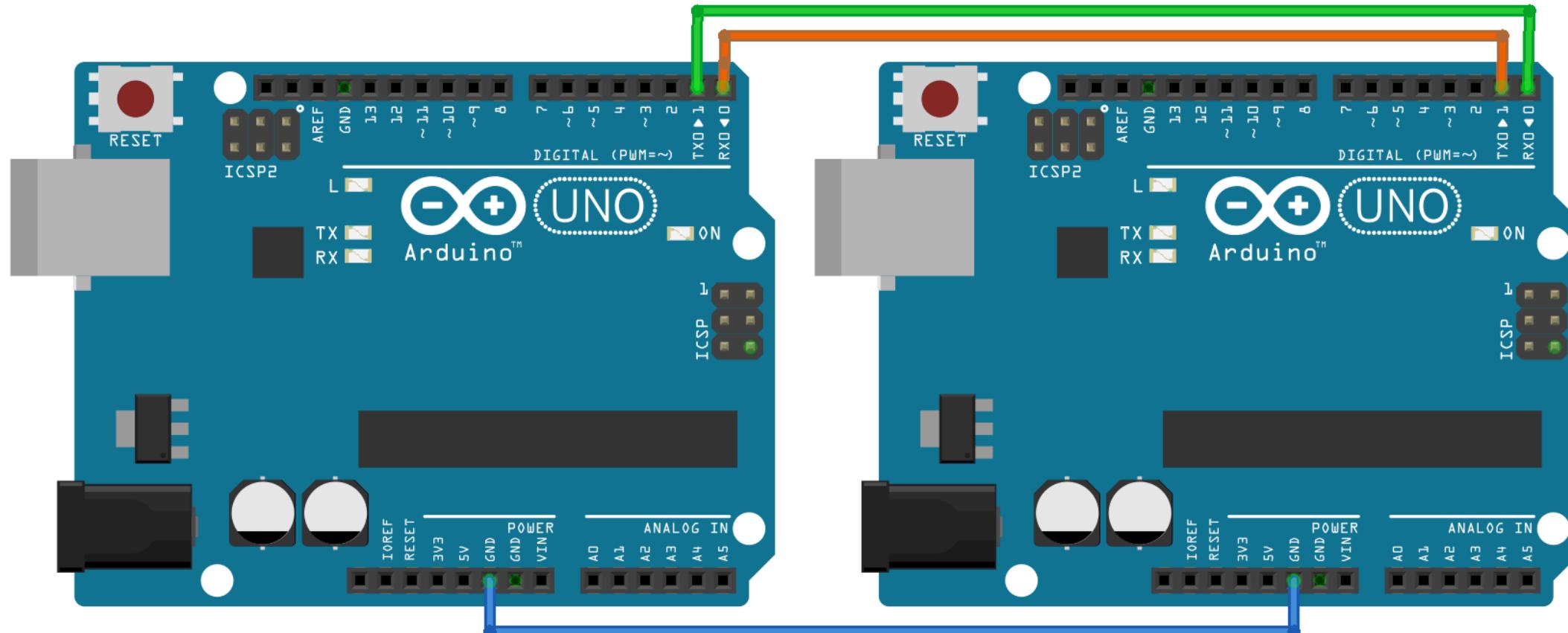
Darshan
UNIVERSITY

To Write data on Serial port : Serial.write()

- Serial.write()
- For the Data transmission
- Syntax:- Serial.write(data)
 - ❖ Parameter: data ⇒ The value to send
- Serial.write(buf, len)
 - ❖ Parameter:
 - buf ⇒ The array to send
 - len ⇒ The number of bytes
- Return: Bytes of data to be sent

```
Write.ino
1 void setup() {
2     Serial.begin(9600);
3     //To Start Serial Communication
4 }
5 void loop() {
6     Serial.write('A');
7     //Write one byte data
8     Serial.write(0x04);
9     delay(1000);
10    // Wait for one second
11 }
12 }
```

Serial Communication between Two Arduino Board



Code for Sender -Receiver Arduino

Sender.ino

```
1 char mystr[5] = "Hello";
2 //String data
3 void setup() {
4 // Begin the Serial at
//9600 Baud
5 Serial.begin(9600); }
6 void loop() {
7 Serial.write(mystr,5);
//Write the serial data
8 delay(1000);
9 }
10
11 }
```

receiver.ino

```
1 char mystr[10];
2 //Initialized variable to
3 store received data
4 void setup() {
5 // Begin the Serial at 9600
6 Serial.begin(9600); }
7 void loop() {
8 Serial.readBytes(mystr,5);
//Read the serial data and
//store in var
9
10 Serial.println(mystr);
11 //Print data on Serial
12 //Monitor
13 delay(1000);
14 }
15 }
```

Software Serial Port

- Arduino boards have built in support for serial communication on pins 0 and 1, but what if you need more serial ports?
- The **SoftwareSerial** Library has been developed to allow serial communication to take place on the other digital pins of your boards, using software to replicate the functionality of the hardwired RX and TX lines
- The SoftwareSerial library in Arduino allows you to create additional serial communication ports on digital pins, which is useful if the hardware UART (pins 0 and 1 on most Arduinos) is already in use.
- This enables you to communicate with multiple devices (e.g., sensors, modules) that use serial communication.
- **#include <SoftwareSerial.h>**

Software Serial

- The SoftwareSerial library implements serial communication in software, using any pair of digital pins for RX (receive) and TX (transmit).
- It works at a range of baud rates, but since it's software-based, it's less efficient and slower than hardware UART.
- SoftwareSerial library has the following known limitations:
 - ❖ It cannot transmit and receive data at the same time.
 - ❖ If using multiple software serial ports, only one can receive data at a time.

```
// Create a SoftwareSerial object  
#include <SoftwareSerial.h>  
SoftwareSerial mySerial (RXPIN, TXPIN);
```

Software Serial Example Code

sketch_01.ino

```
1 #include <SoftwareSerial.h>
2 // Define RX and TX pins for SoftwareSerial
3 #define RX_PIN 10
4 #define TX_PIN 11
5 // Create a SoftwareSerial object
6 SoftwareSerial mySerial(RX_PIN, TX_PIN);
7 void setup() {
8 // Begin communication on the software serial port
9 mySerial.begin(9600);
10 // Begin communication on the hardware serial port
11 (USB)
12 Serial.begin(9600);
13 Serial.println("SoftwareSerial Ready");
14 }
15
```

Software Serial Example Code

sketch_01.ino

```
16 void loop() {  
17 // Send data from hardware serial to software serial  
18     if (Serial.available()) {  
19         char data = Serial.read(); //Serial.read(), reads one byte of data  
20         mySerial.write(data);  
21         // Send data to the software serial port  
22         Serial.print("Sent to SoftwareSerial: ");  
23         Serial.println(data);  
24     }  
25 // Receive data from software serial and print to hardware serial  
26     if (mySerial.available()) {  
27         char data = mySerial.read();  
28         Serial.print("Received from SoftwareSerial: ");  
29         Serial.println(data); }  
30 }
```

SPI: Serial Peripheral Interface

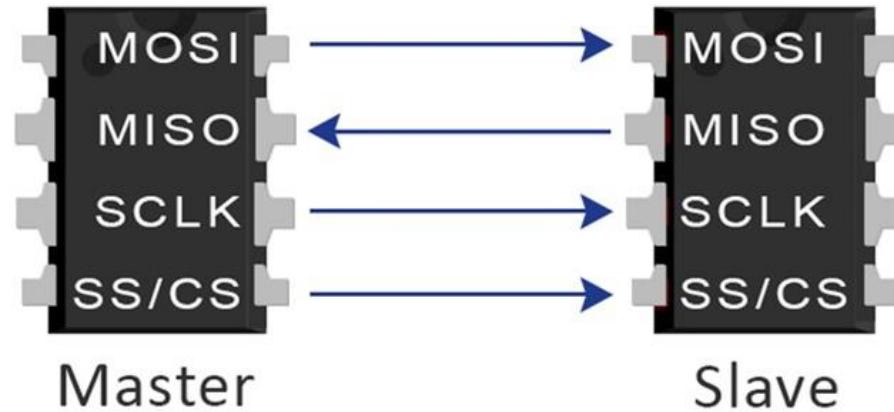
- SPI stands for **Serial Peripheral Interface**
 - ❖ It is a **Serial Bus Communication Protocol**
 - ❖ One of the most popular serial synchronous bus protocols (for short-distance communication)
 - ❖ its origin goes back to late 1980s developed by **Motorola**
 - ❖ Supports **high-speed synchronous data transfer.**

SPI: Serial Peripheral Interface

- Devices communicate in **master/slave mode**.
 - ❖ Master device **initiates** the data communication
 - ❖ Master device generates a **Serial Clock** for synchronous.
- SPI can be multi-slave but **it can't be multi-master**.
 - ❖ Each slave is connected to the master device via Slave Select line.
- During the data transfer process
 - ❖ The **master** always send **8 to 16 bits data** to the slave
 - ❖ The **slave** always sends **8 bits data** to the master
- SPI is a **full-duplex** master-slave communication protocol.
 - ❖ This means only a single master and a single slave can communicate on the bus at the same time.

SPI Buses

- SPI Interface uses **four wires** for communication.
- Hence it is also known as a four-wire serial communication protocol.



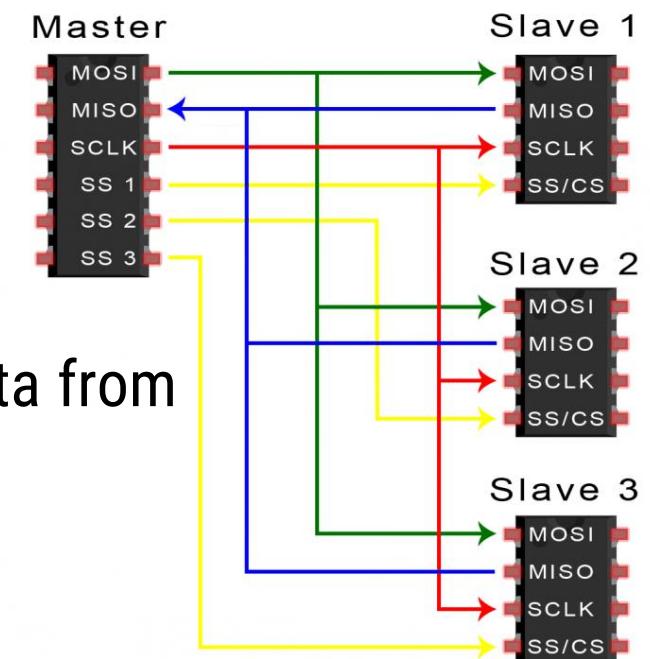
1. **MOSI** (Master Out-Slave In)
2. **MISO** (Master In-Slave Out)
3. **SCLK** (Serial Clock, which produces by the master)
4. **SS** (Slave select or Chip select line which is used to select specific slave during the communication)

SPI Pins

- ❖ Pin Name: **MOSI**
- The MOSI signal is a **unidirectional** signal used for transfer serial data from the Master to the Slave.
- When a device is a **Master**, **serial data is output** on this signal.
- When a device is a **Slave**, **serial data is input** on this signal.

❖ Pin Name: **MISO**

- The MISO signal is a unidirectional signal used to transfer serial data from the slave to the master.
- When a device is a **Slave**, **serial data is output** on this signal.
- When a device is a **Master**, **serial data is input** on this signal.
- When a slave device is **not selected**, the slave drives the signal with **high impedance**



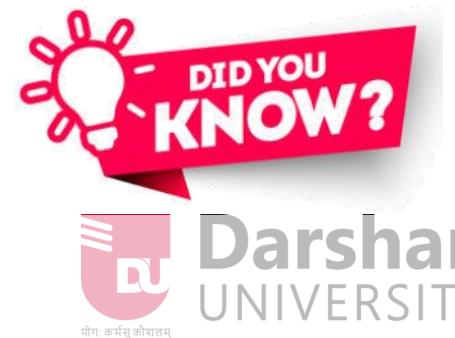
SPI Pins

- ❖ Pin Name: **SCLK**
- Clock signals were utilized by the SPI to **synchronize data flow** across the SPI Interface.
- The **frequency of the clock signal** controls how quickly information is transferred as one bit of data is sent during each clock cycle.
- Clock is **only active** during the data transfer.
- Master device **configures and generates** the clock signal
- ❖ Pin Name: **SS / CS**
- The Slave Select or Chip Select line is used to **select specific slave** during the communication.
- Master Device can **handle** multiple slave devices on the bus by selecting them one by one.
 - ❖ Because the SS line connects each slave with the master, **there is no unique address for each slave** like for the I2C communication.

SPCR –SPI Control Register



- SPIE - Enables the SPI interrupt when 1
- SPE - Enables the SPI when 1
- DORD - Sends data least Significant Bit First when 1, most Significant Bit first when 0
- MSTR - Sets the Arduino in controller mode when 1, peripheral mode when 0
- CPOL - Sets the data clock to be idle when high if set to 1, idle when low if set to 0
- CPHA - Samples data on the falling edge of the data clock when 1, rising edge when 0
- SPR1 and SPR0 - Sets the SPI speed
 - ❖ 00 is fastest (4MHz)
 - ❖ 11 is slowest (250KHz)



Datasheet – ATMEGA328P

18.5.1 SPCR – SPI Control Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | SPCR |
|---------------|------|-----|------|------|------|------|------|------|------|
| 0x2C (0x4C) | SPIE | SPE | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR register is set and the if the global interrupt enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects master SPI mode when written to one, and slave SPI mode when written logic zero. If SS is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI master mode.

- **Bit 3 – CPOL: Clock Polarity**



BV Macro :

_BV(SPE) is a macro that shifts the bit position of the SPE constant to create a bitmask.

It stands for **Bit Value** and is defined as:

```
#define _BV(bit) (1 << (bit))
```

For SPE, which corresponds to bit 6, _BV(SPE) results in 0b01000000.

Table 18-5. Relationship Between SCK and the Oscillator Frequency

| SPI2X | SPR1 | SPR0 | SCK Frequency |
|-------|------|------|---------------|
| 0 | 0 | 0 | $f_{osc}/4$ |
| 0 | 0 | 1 | $f_{osc}/16$ |
| 0 | 1 | 0 | $f_{osc}/64$ |
| 0 | 1 | 1 | $f_{osc}/128$ |
| 1 | 0 | 0 | $f_{osc}/2$ |
| 1 | 0 | 1 | $f_{osc}/8$ |
| 1 | 1 | 0 | $f_{osc}/32$ |
| 1 | 1 | 1 | $f_{osc}/64$ |

Datasheet – ATMEGA328P

➤ ATMEGA328P Registers :



18.5.3 SPDR – SPI Data Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|-----|-----|-----|-----|-----|-----|-----|-----|-----------|
| 0x2E (0x4E) | MSB | | | | | | | LSB | SPDR |
| Read/Write | R/W | |
| Initial Value | X | X | X | X | X | X | X | X | Undefined |

The SPI data register is a read/write register used for data transfer between the register file and the SPI shift register. Writing to the register initiates data transmission. Reading the register causes the shift register Receive buffer to be read.

18.5.2 SPSR – SPI Status Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|------|------|---|---|---|---|---|-------|------|
| 0x2D (0x4D) | SPIF | WCOL | - | - | - | - | - | SPI2X | SPSR |
| Read/Write | R | R | R | R | R | R | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

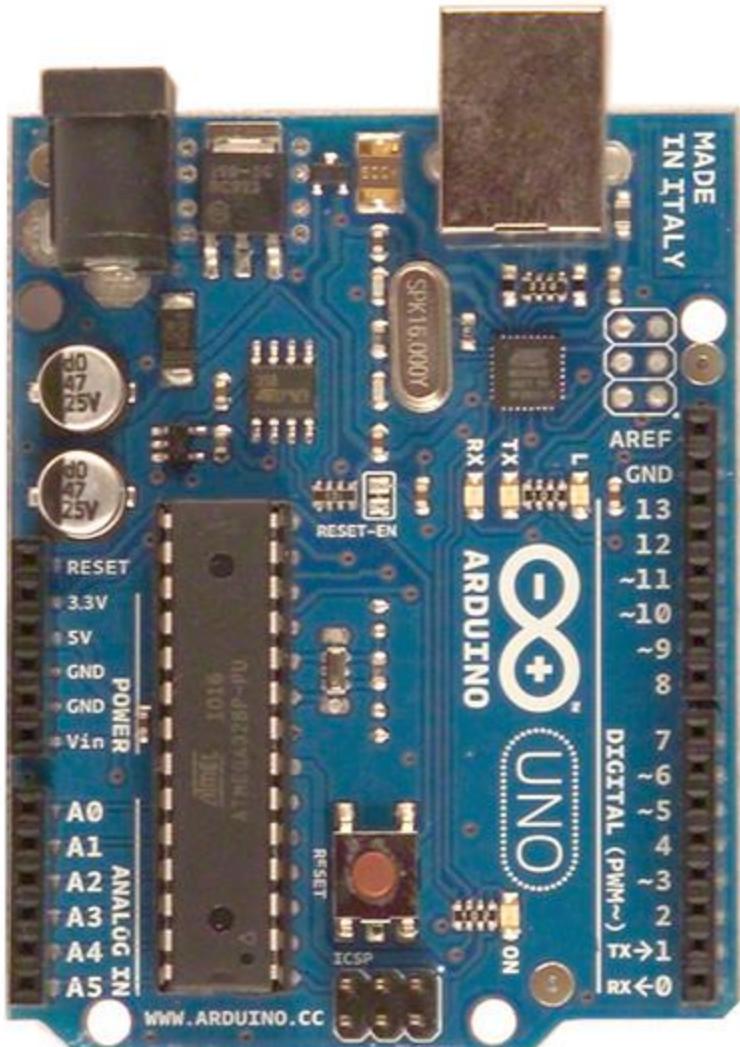
- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set. If SPIF is set and SPI interrupt are enabled. If SS is an input and is driven low when the SPI is in master mode, this will also set SPIF. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF flag can be cleared by first reading the SPI status register with SPIF set, then accessing the SPI data register (SPDR).

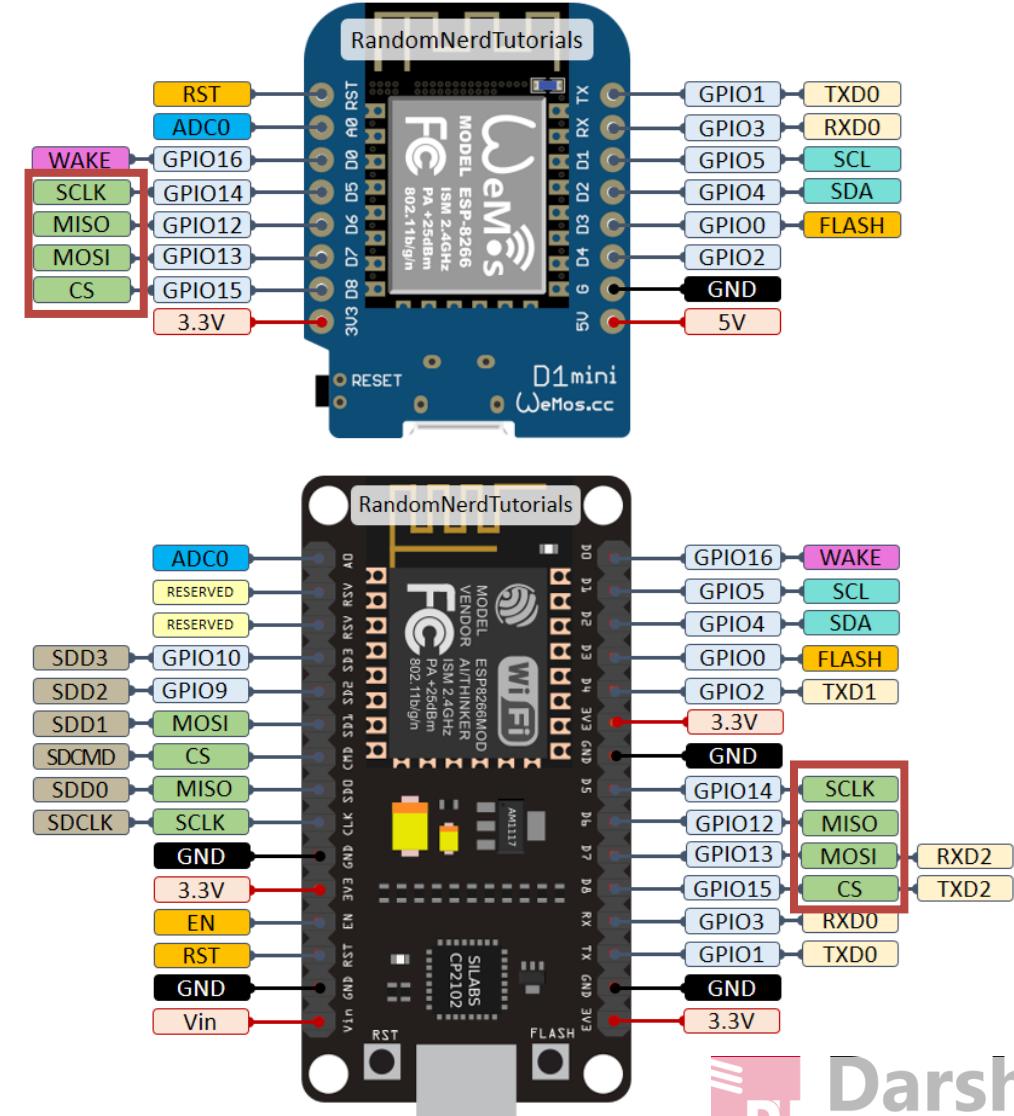
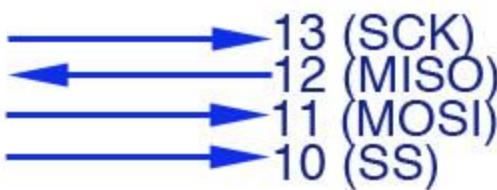
- **Bit 6 – WCOL: Write Collision Flag**



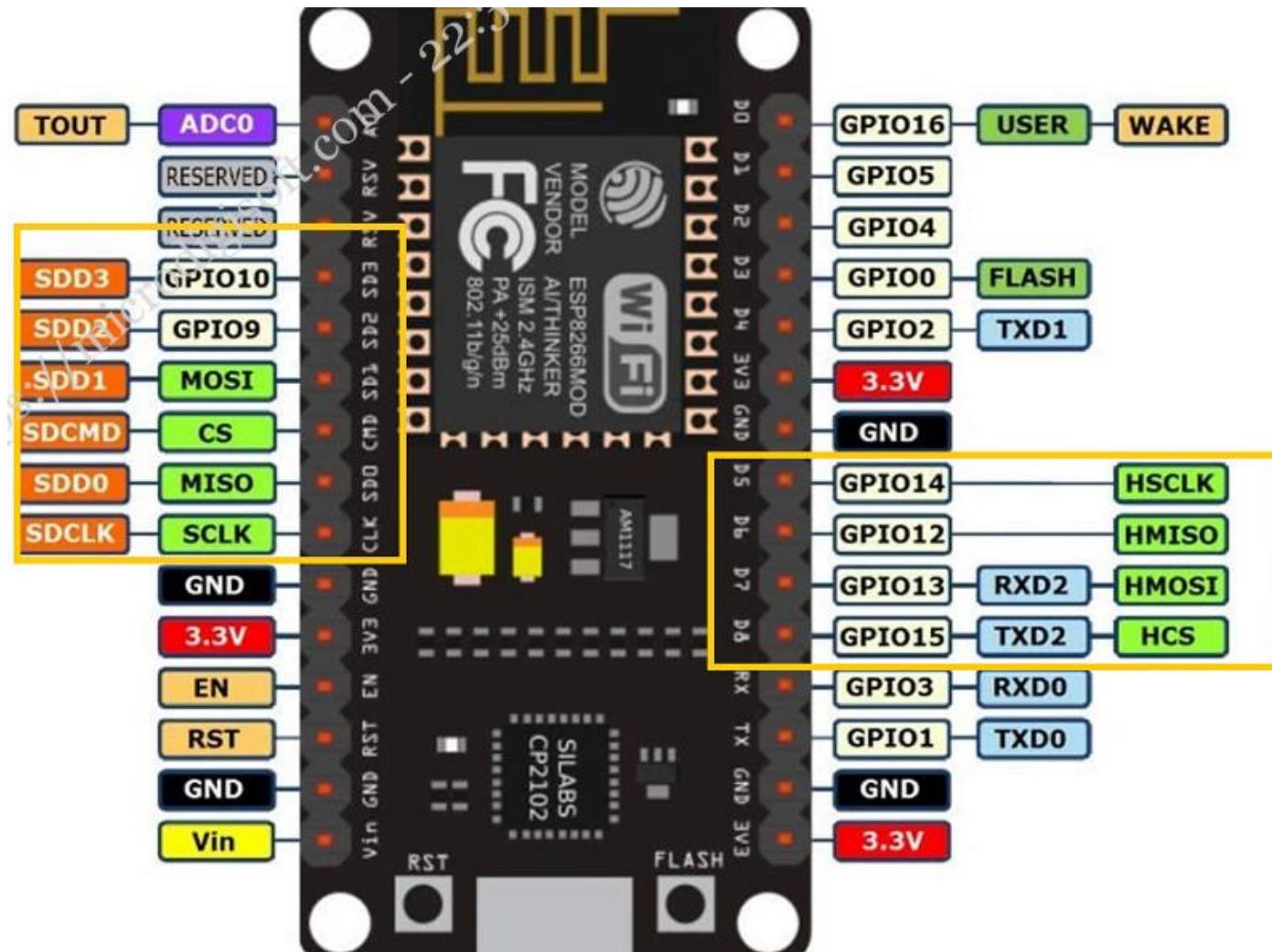
Arduino SPI



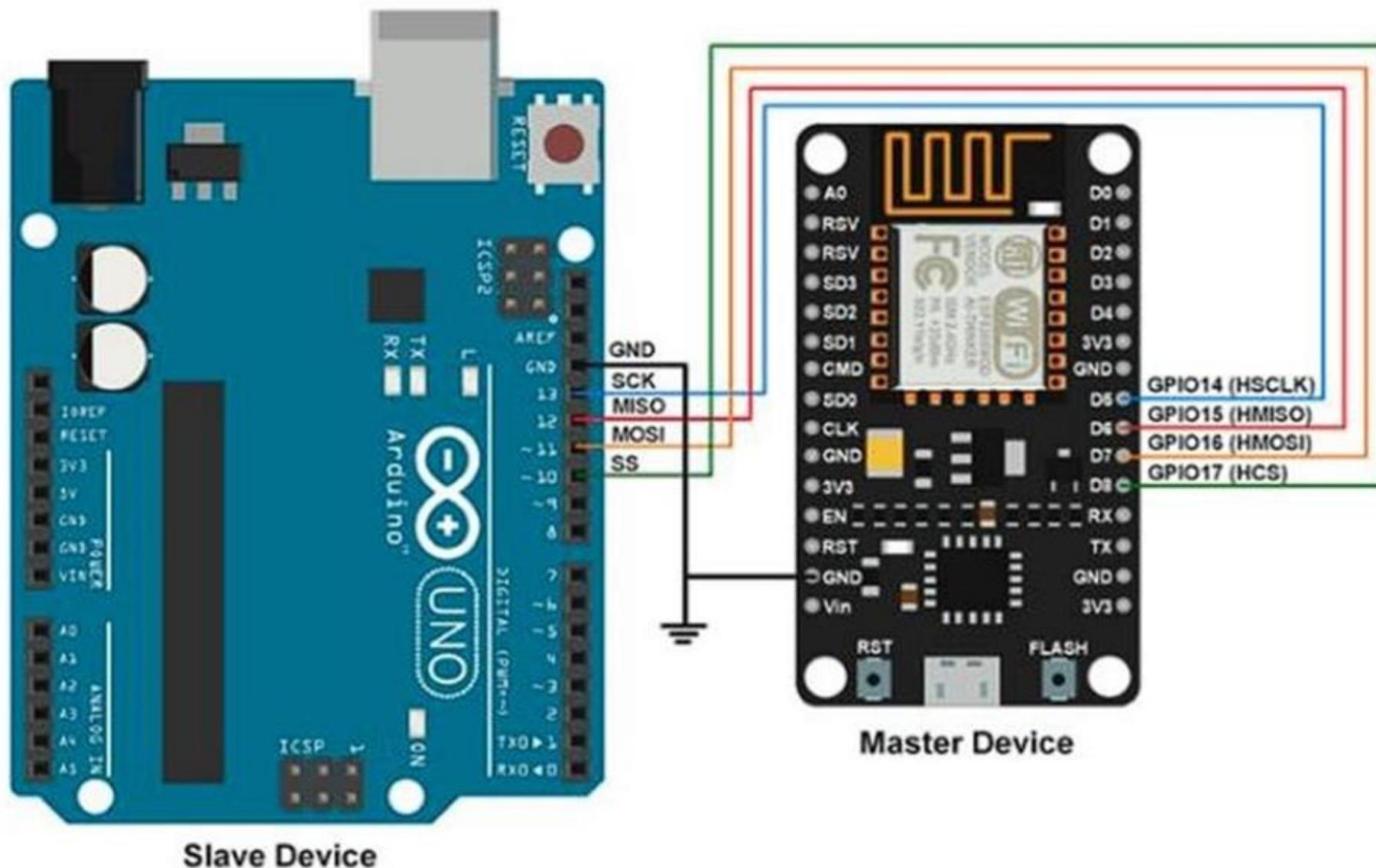
SPI pins



NodeMCU SPI



Node MCU – Arduino UNO



SPI Header File

- #include<SPI.h>
- SPI.begin()
 - Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.
 - ❖ SPI.begin();
- SPI.setClockDivider(divider)
 - To set the SPI clock divider relative to the system clock.
 - On AVR based boards, the dividers available are 2, 4, 8, 16, 32, 64 or 128.
 - The default setting is SPI_CLOCK_DIV4, which sets the SPI clock to one-quarter of the frequency of the system clock (5 Mhz for the boards at 20 MHz).
 - Divider - It could be :
 - ❖ SPI_CLOCK_DIV2, SPI_CLOCK_DIV4, SPI_CLOCK_DIV8, SPI_CLOCK_DIV16, SPI_CLOCK_DIV32, SPI_CLOCK_DIV64, SPI_CLOCK_DIV128.

❑ SPI.transfer(val)

- Purpose: Sends and receives a single byte of data over the SPI bus.
- The SPI transfer is based on a simultaneous send and receive: the received data is returned in receivedVal.
- SPI.transfer(byte data) function sends the byte (data) out via the MOSI pin.
- Simultaneously, it reads a byte from the MISO pin.
- SPI is a full-duplex protocol, so data is sent and received simultaneously.

```
byte receivedData;
```

```
receivedData = SPI.transfer(0x42); // Send 0x42 and store the received byte
```

SPI Header File

❑ SPI.attachInterrupt(handler)

- Function to be called when a slave device receives data from the master.

❑ SPI.beginTransaction(SPISettings(speedMaximum, dataOrder, dataMode))

- The SPISettings object defines the configuration:
- Clock speed: Frequency of the SPI clock (e.g., 4000000 for 4 MHz).
- dataOrder(MSBFIRST or LSBFIRST) (Master Sends MSB First, Slave sends LSB first)
- data mode: Clock polarity and phase (one of SPI_MODE0, SPI_MODE1, SPI_MODE2, SPI_MODE3).
- Used for advanced control when dealing with multiple SPI devices that require different configurations.

SPI Mode

- Mode 0 (the default) - Clock is normally low (CPOL = 0), and the data is sampled on the transition from low to high (leading edge) (CPHA = 0).
- Mode 1 - Clock is normally low (CPOL = 0), and the data is sampled on the transition from high to low (trailing edge) (CPHA = 1).
- Mode 2 - Clock is normally high (CPOL = 1), and the data is sampled on the transition from high to low (leading edge) (CPHA = 0).
- Mode 3 - Clock is normally high (CPOL = 1), and the data is sampled on the transition from low to high (trailing edge) (CPHA = 1).

❑ SPI.endTransaction()

- Purpose: Ends the SPI transaction started by SPI.beginTransaction().

❑ SPI.end()

- Purpose: Disables the SPI bus.

NodeMCU (Master Device Code)

Master.ino

```
1 #include <SPI.h>
2
3 void setup (void)
4 {
5     Serial.begin(115200);
6     //set baud rate to 115200 for usart
7
8     digitalWrite(SS, HIGH);
9     // disable Slave Select
10
11    SPI.begin ();
12    SPI.setClockDivider(SPI_CLOCK_DIV8);
13 //divide the clock by 8 }
```

NodeMCU (Master Device Code)

Master.ino

```
14 void loop (void) {  
15     char c;  
16     digitalWrite(SS, LOW);  
17     // enable Slave Select  
18     // send test string  
19     for (const char * p = "Hello, world!\r" ; c = *p; p++)  
20     {  
21         SPI.transfer (c);  
22         Serial.print(c);  
23     }  
24     digitalWrite(SS, HIGH); // disable Slave Select  
25     delay(2000);  
26 }
```

Arduino UNO (Slave Device Code)

Slave.ino

```
1 #include <SPI.h>
2 char buff [50];
3 volatile byte indx;
4 volatile boolean process;
5
6 void setup (void) {
7     Serial.begin (115200);
8     pinMode(MISO, OUTPUT);
9     // have to send on master in so it set as output
10    SPCR |= _BV(SPE);
11    // turn on SPI in slave mode
12    indx = 0; // buffer empty
13    process = false;
14    SPI.attachInterrupt(); // turn on interrupt }
```

Arduino UNO (Slave Device Code)

Slave.ino

```
15 ISR (SPI_STC_vect) // SPI interrupt routine
16 {
17     byte c = SPDR;
18     // read byte from SPI Data Register
19     if (indx < sizeof (buff))
20     {
21         buff [indx++] = c;
22         // save data in the next index in the array buff
23
24         if (c == '\r')
25             //check for the end of the word
26
27         process = true; }
28 }
```

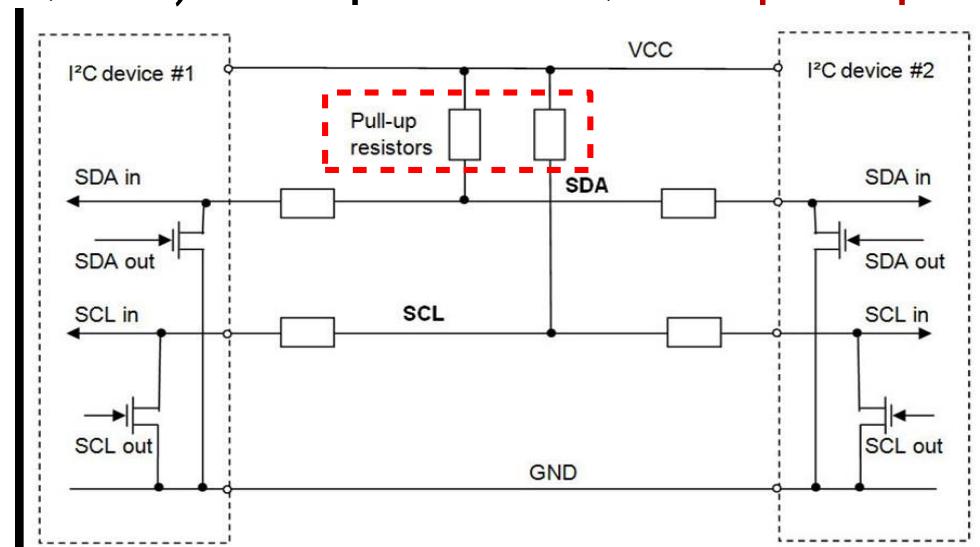
Arduino UNO (Slave Device Code)

Slave.ino

```
29 void loop (void) {  
30     if (process) {  
31         process = false;  
32         //reset the process  
33  
34         Serial.println (buff);  
35         //print the array on serial monitor  
36  
37         idx= 0; //reset buffer index to zero }  
38 }  
39  
40  
41  
42 }
```

I²C Protocol :Inter Integrated Circuit

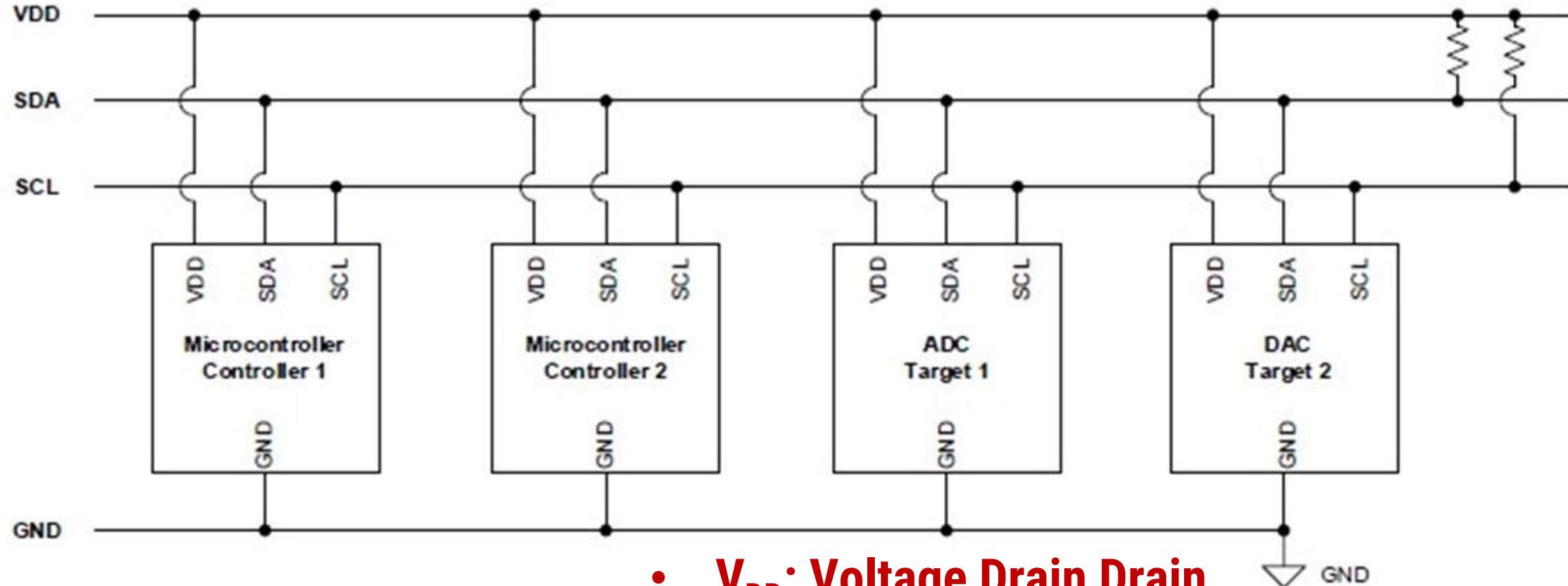
- One of the widely used Serial Communication protocols, created by **Philips Semiconductor** in 1982 (Now it is **NXP Semiconductor**)
- Only two communication lines for all devices on the bus (SDA, SCL).
- Bi-directional communication
- I²C link is **half-duplex**, means only one device transmits at any given time.
- Allows for **multiple controllers** and **multiple targets**
- Both the signal lines (SDA, SCL) are 'open drain', thus **pull-up resistors** are needed.



I2C Communication Modes

| I2C Mode | Speed | |
|------------------|----------|---|
| Standard Mode | 100 kbps | Similar in implementation, with different timing requirements |
| Fast Mode | 400 kbps | |
| Fast Mode Plus | 1 Mbps | |
| High Speed Mode | 3.4 Mbps | Requires specific controller code for high speed transfer |
| Ultra -Fast Mode | 5 Mbps | |

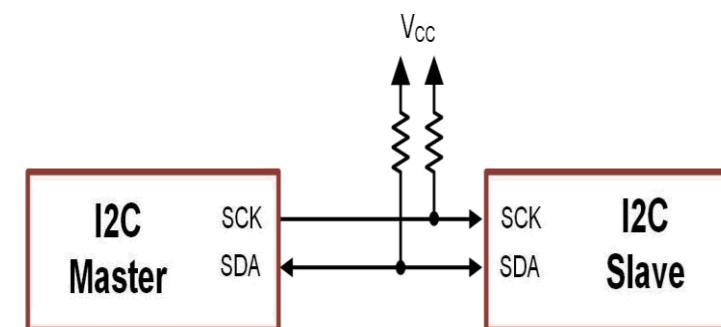
I2C Physical Layer



- **V_{DD}: Voltage Drain Drain**
- **SDA: Serial Data**
- **SCL: Serial Clock**

I2C Protocol Operation

- **Master device** – the device that initiates communication and controls the clock.
- **Slave device** – a device on the bus that is read or written to, but does not initiate transmission or provide a clock.
- **Slave address** – a unique and predetermined address for each slave on the bus.
- This address is used by the master to indicate which slave it wants to communicate with.
- **Idle** – when both SDA and SCL are held high by the pull-up resistors and no I2C device is attempting to communicate.
- **Busy** – when devices are driving the bus.
- **Messages** – how I2C information is transferred.



Operation Steps in I2C - 1 to 6

No Communication
•SCL and SDA both are in Idle

Master Starts

- A master initiates by pulling SDA LOW while SCL is still HIGH.

Clock Pulse Numbering

Both master and slave count clock pulse

Slave Address

Master Sends 7bit or 10bit

Read (HIGH) Write(LOW)

For the Slave

Slave Acknowledgement

Each Slave check .
If Addr. Found
Give Ack.

01



02



03



04



05



06



SCL

1

SDA

2

Start

3

Slave Address

4

5

R/W ACK

6

Operation Steps in I2C - 7 to 09

Master Proceeds with Data

- Master checks for Ack. And Proceeds Data

Data Ack.

- Receiver sends ACK after getting each byte.

STOP Condition occurs

SDA: LOW to HIGH
SCL : Remains HIGH

07



08



09



7

Data

8

ACK

9

Stop



Darshan
UNIVERSITY

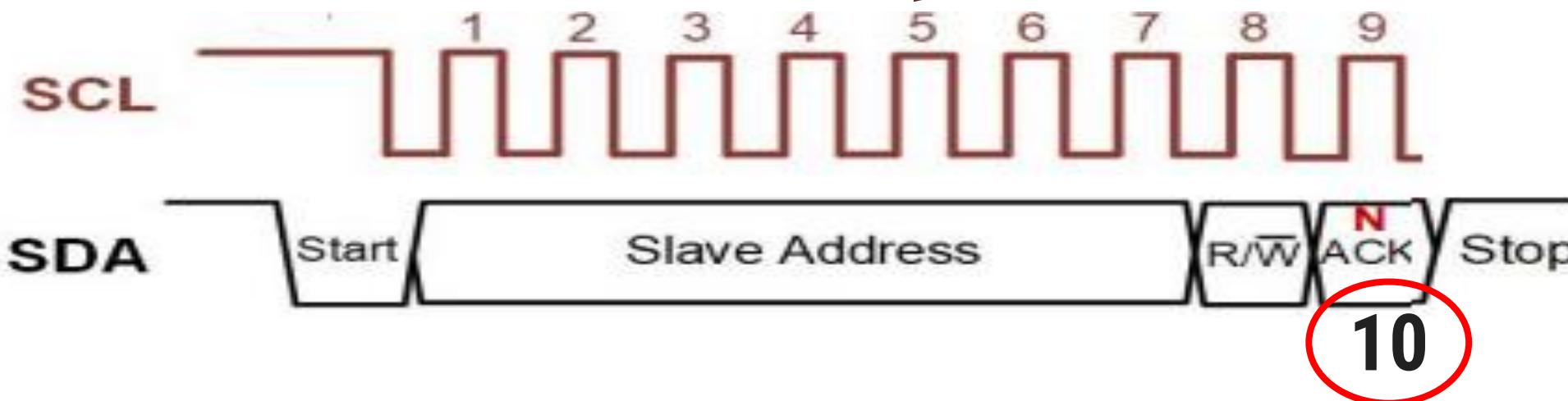
गोपा: कर्मसु कीशतम्

Operation Steps in I2C - 10

NACK

When No slave
found with
Addr.

10



Data Transfer Structure

General structure of a 2-byte transfer

| START | Slave address | Rd/nWr | ACK | Data | ACK | Data | ACK | STOP |
|-------|---------------|--------|-------|--------|-------|--------|-------|-------|
| 1 bit | 7 bits | 1 bit | 1 bit | 8 bits | 1 bit | 8 bits | 1 bit | 1 bit |

Writing 2-bytes (shaded bits are put on the bus by the master)

| START | Slave address | 0 | 0 | Data | 0 | Data | 0 | STOP |
|-------|---------------|-------|-------|--------|-------|--------|-------|-------|
| 1 bit | 7 bits | 1 bit | 1 bit | 8 bits | 1 bit | 8 bits | 1 bit | 1 bit |

Reading 2-bytes (shaded bits are put on the bus by the master)

| START | Slave address | 1 | 0 | Data | 0 | Data | 1 | STOP |
|-------|---------------|-------|-------|--------|-------|--------|-------|-------|
| 1 bit | 7 bits | 1 bit | 1 bit | 8 bits | 1 bit | 8 bits | 1 bit | 1 bit |

Wire Library

- This Wire library allows you to communicate with I2C/TWI devices
- There are both 7 or 8-bit versions of I2C addresses. 7 bits identify the device, and the 8 th bit determines if it's being written to or read from.
- The Wire library uses 7 bit addresses throughout. However, the addresses from 0 to 7 are not used because are reserved so the first address that can be used is 8.
- Functions in Wire.h :

begin()

requestFrom()

endTransmission()

available()

setClock()

onRequest()

clearWireTimeoutFlag()

end()

beginTransmission()

write()

read()

onReceive()

setWireTimeout()

getWireTimeoutFlag()

NodeMCU (Master Device Code)

Master-I2C.ino

```
1 #include <Wire.h>
2 void setup() {
3     Serial.begin(9600);
4     /* begin serial for debug */
5     Wire.begin(D1, D2);
6     /* join i2c bus with SDA=D1 and SCL=D2 of NodeMCU */
7 }
8
9 void loop() {
10    Wire.beginTransmission(8);
11    /* begin with device address 8 */
12
13
14
15 }
```

NodeMCU (Master Device Code)

Master-I2C.ino

```
16      Wire.write("Hello Arduino");
17      /* sends hello string */
18
19      Wire.endTransmission();
20      /* stop transmitting */
21
22      Wire.requestFrom(8, 13);
23      /*request&read 13 byte data from slave device #8*/
24
25      while(Wire.available()){
26          char c = Wire.read();
27          Serial.print(c);
28          Serial.println();
29          delay(1000);
30      }
```

Arduino Uno (Slave Device Code)

Slave-I2C.ino

```
1 #include <Wire.h>
2 void setup() {
3     Wire.begin(8);
4     /* join i2c bus with address 8 */
5
6     Wire.onReceive(receiveEvent);
7     /* register receive event */
8
9     Wire.onRequest(requestEvent);
10    /* register request event */
11
12    Serial.begin(9600); /* start serial for debug */ }
```

Arduino Uno (Slave Device Code)

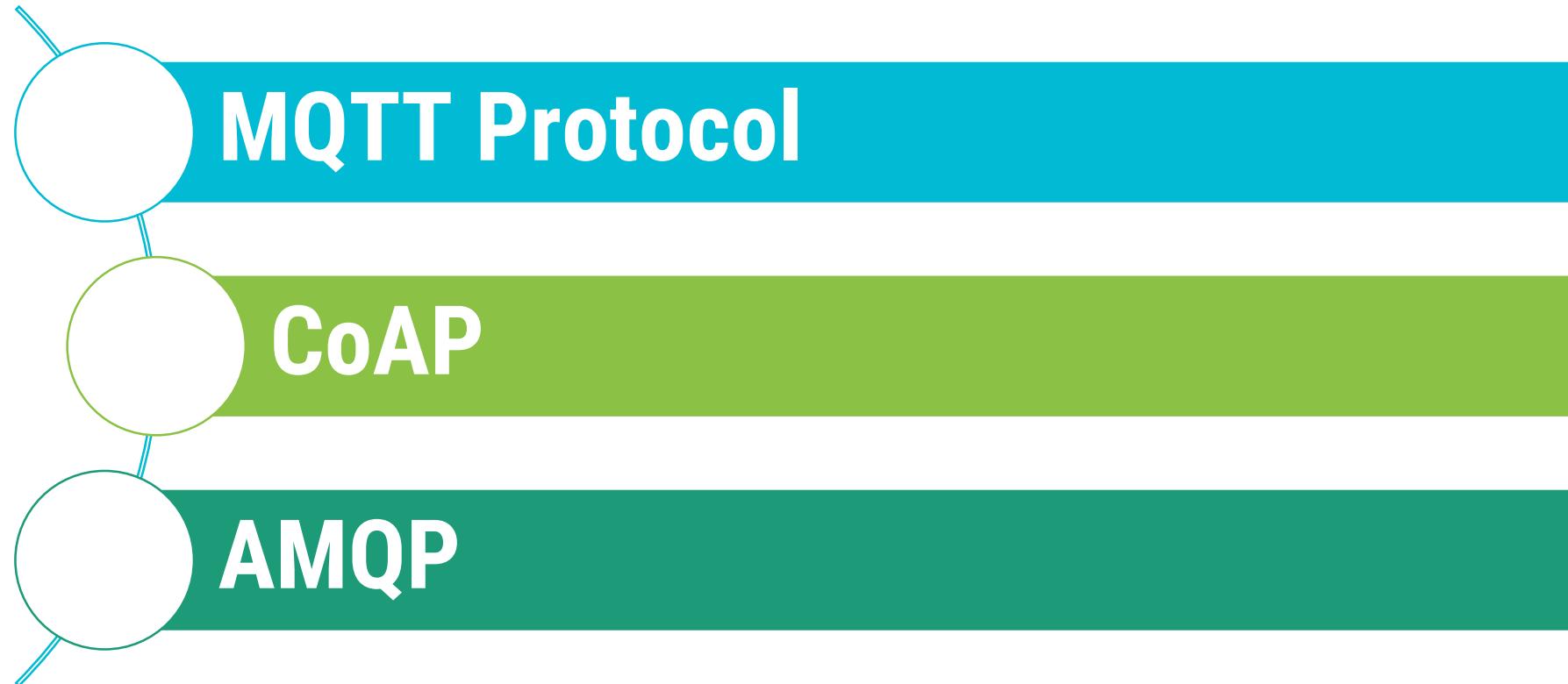
Slave-I2C.ino

```
13 void loop() {  
14     delay(100); }  
15 /* function that executes whenever data is received from master */  
16 void receiveEvent(int howMany)  
17 {  
18     while (Wire.available()>0)  
19     {  
20         char c = Wire.read();  
21         /* receive byte as a character */  
22         Serial.print(c);  
23         /* print the character */ }  
24         Serial.println();  
25         /* to newline */  
26 }
```

Arduino Uno (Slave Device Code)

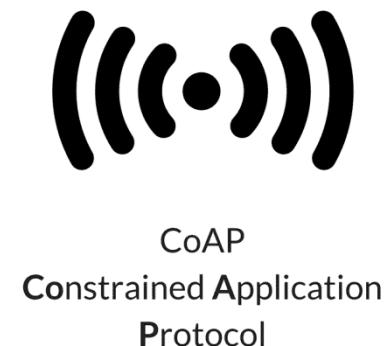
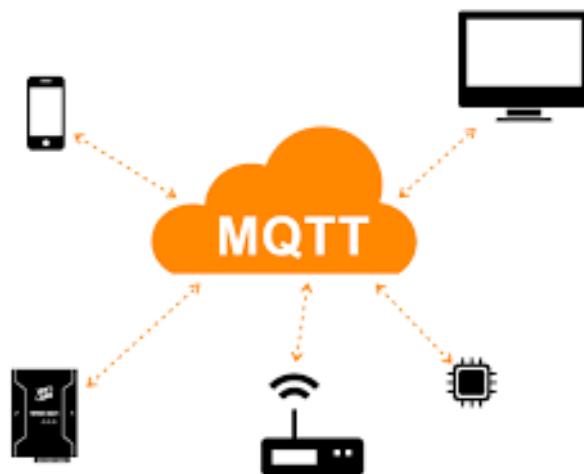
Slave-I2C.ino

```
27 /* function that executes whenever data is
28 requested from master */
29
30 void requestEvent() {
31     Wire.write("Hello NodeMCU");
32     /*send string on request */
33 }
34
```



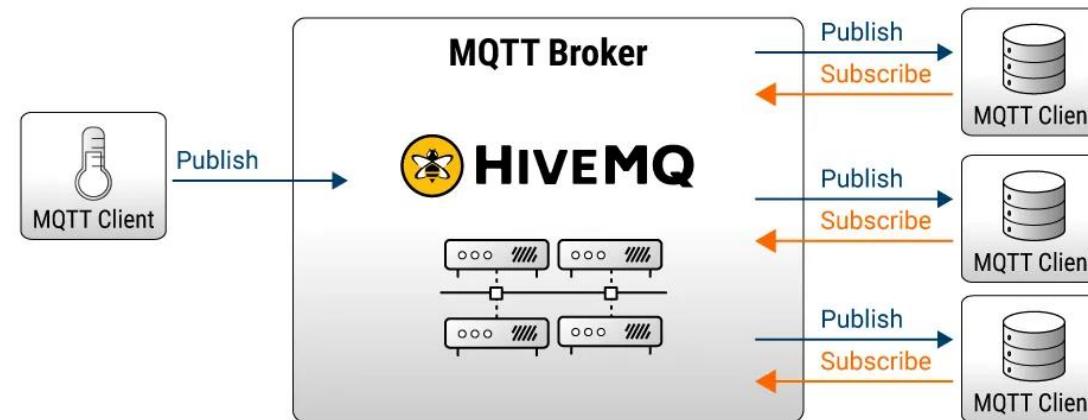
Messaging Protocols in IoT

- Messaging Protocols are very important for the transfer of data in terms of messages.
- They are useful for send/receive of a message to/from the cloud in IoT applications.
- In the section, two messaging protocols are discussed.
 1. Message Queuing Telemetry Transport (**MQTT**)
 2. Constrained Application Protocol (**CoAP**)



Message Queuing Telemetry Transport (MQTT)

- The MQTT full form is Message Queuing Telemetry Transport.
- It is a protocol that is mainly used for connecting devices with each other on the internet with backend services.
- IoT has one of the biggest challenges of resource constraints, the lightweight protocol is more preferred.
- MQTT is widely used in IoT applications as it is a lightweight protocol.
- **Lightweight** means, it can work with **minimal resources** and does not require any specific hardware architecture or additional resources.

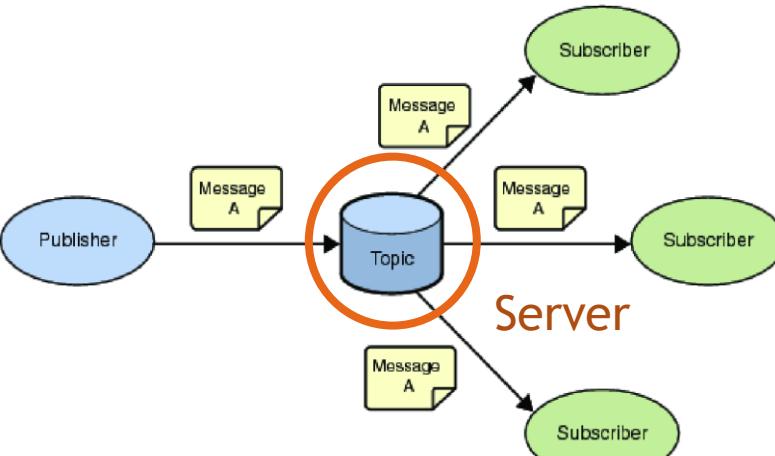


MQTT : Message Queueing Telemetry Transport protocol

- It's **Reliable, Lightweight, Cost-effective protocol** that transports messages between devices.
- Suited for the transport of **telemetry data** (sensor and actuator data)
- MQTT is a **machine-to-machine** connectivity protocol that runs over **TCP/IP**.
- MQTT is based on a **publish- subscribe** structure.
- **Publish** : A Publisher **sends messages according to Topics**, to specified Brokers.
- **Broker** : A Broker acts as a **switchboard**, accepting messages from publishers on specified topics, and sending them to subscribers to those Topics.
- **Subscribe** : A Subscriber receives messages from connected Brokers and specified Topics.
- MQTT is an Event based IoT middleware (one to many).
- It can be supported by some of the smallest measuring and monitoring devices (ex. Arduino).

Publish / Subscribe Messaging (One to Many)

- A producer **publishes** a message (publication) on a **topic** (subject).
- A consumer **subscribes** (makes a subscription) for messages on a topic (subject).
- A message server (called **Broker**) **matches** publications to subscriptions

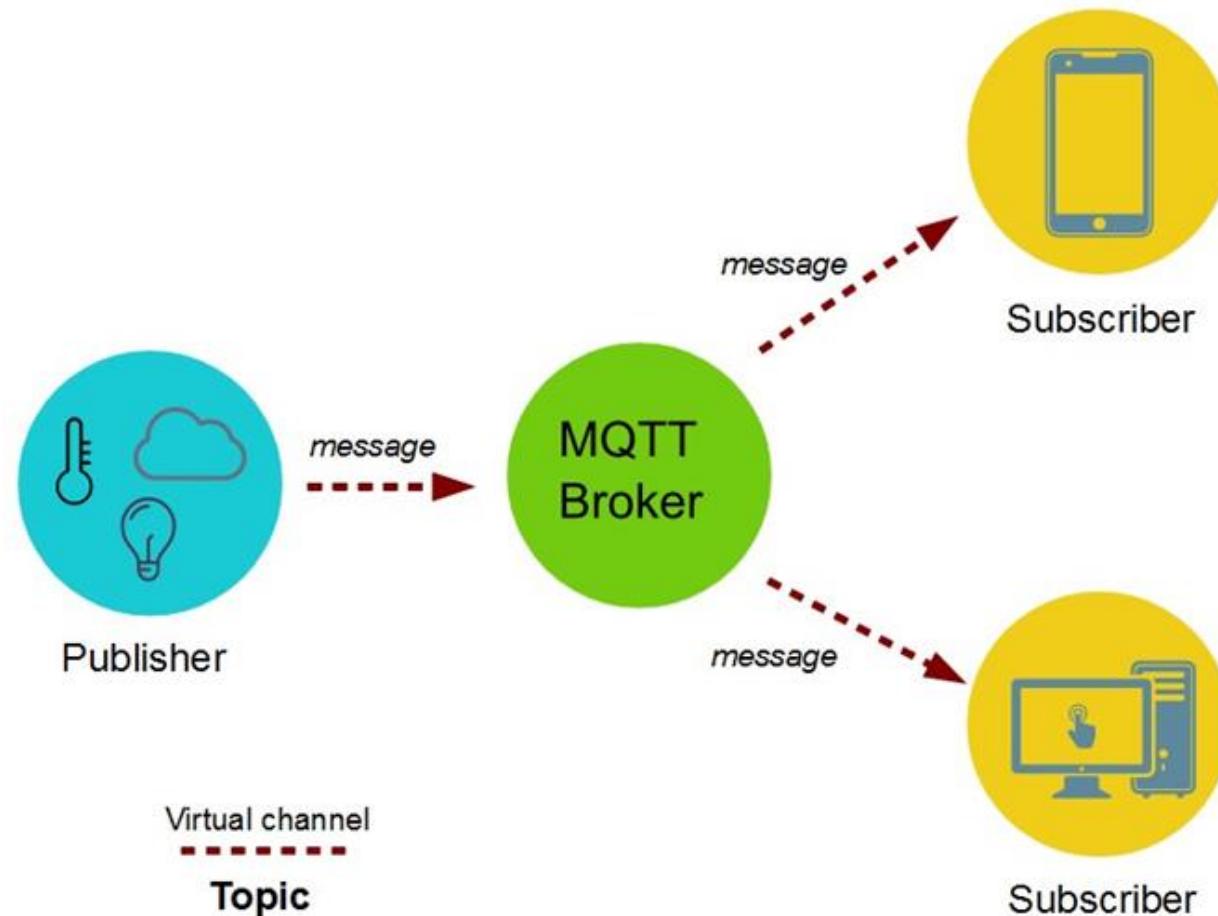


- Publish / Subscribe has three important characteristics:
 1. It **decouples** senders and receivers, allowing for more flexible applications.
 2. It can take a single message and distribute it to many consumers.
 3. This **collection of consumers can change over time**, and vary based on the nature of the message.

MQTT Architecture

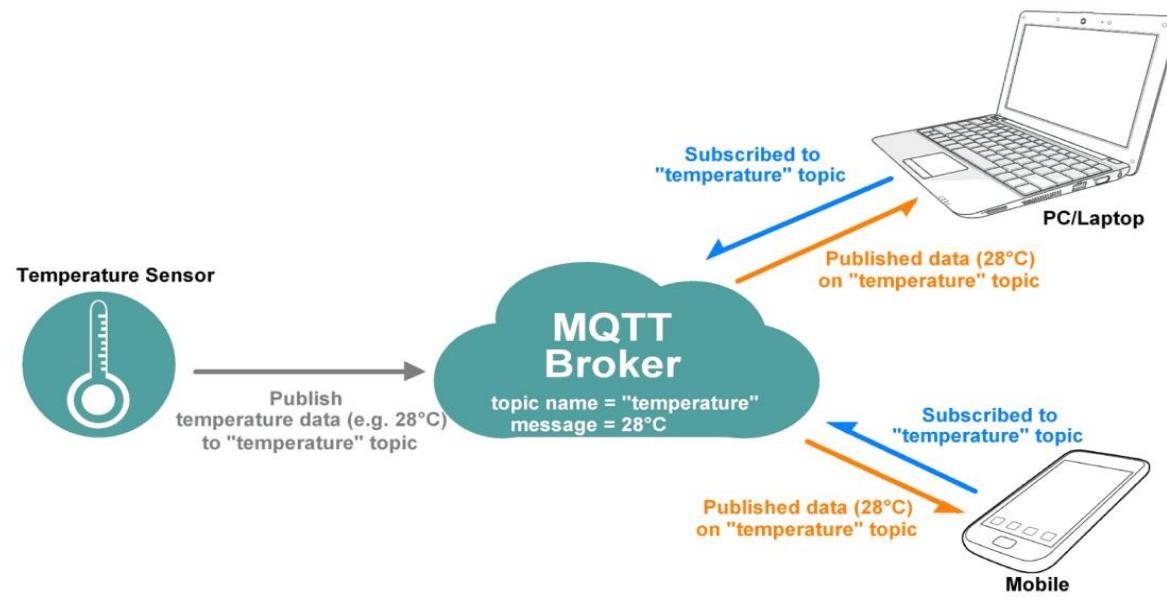
➤ Architecture of MQTT:

1. Publisher
2. Broker
3. Subscribers



MQTT Communication

- Who will get the message ?
 - ❖ If no matches, the message is discarded
 - ❖ If one / more matches, the message is delivered to each matching subscriber/consumer
- **Asynchronous communication model** with messages (events).
- Publisher and Subscriber need not be online at the same time.



MQTT Topic

- A topic forms the namespace
 - ❖ Is hierarchical with each “sub topic” separated by a: Forwardslash /
- An example:
- A house **publishes** information about itself on:
 - ❖ <country>/<state>/<town>/<postcode>/<house>/energyConsumption
 - ❖ <country>/<state>/<town>/<postcode>/<house>/solarEnerg
 - ❖ <country>/<state>/<town>/<postcode>/<house>/alarmState
- And **subscribes** for control commands:
 - ❖ <country>/<state>/<town>/<postcode>/<house>/thermostat/setTemp

MQTT Subscriber

- A subscriber can subscribe to an absolute topic or can use **wildcards**:

- ❖ **Single-level** wildcards “+” can appear anywhere in the topic string
- ❖ **Multi-level** wildcards “#” must appear at the end of the string
- ❖ Wildcards must be next to a separator
- ❖ **Cannot be used wildcards when publishing**

MQTT Topics & Wildcards

- Topics are hierarchical (like filesystem path):

- /wsn/sensor/R1/temperature
- /wsn/sensor/R1/pressure
- /wsn/sensor/R2/temperature
- /wsn/sensor/R2/pressure

- A Subscriber can use wildcards in topics:

- /wsn/sensor/+/temperature
- /wsn/sensor/R1/+
- /wsn/sensor/#



Darshan
UNIVERSITY

गोपा: कर्मसु कीशताम्

Subscription types

- A subscription can be **durable** or **non-durable**

Durable:

- Once a subscription is in place, A broker will forward matching messages to the subscriber immediately if the subscriber is connected.
- If the subscriber is not connected, messages are **stored** on the server/broker until the next time the subscriber connects.

Non-durable / Transient (i.e. subscription ends with client session):

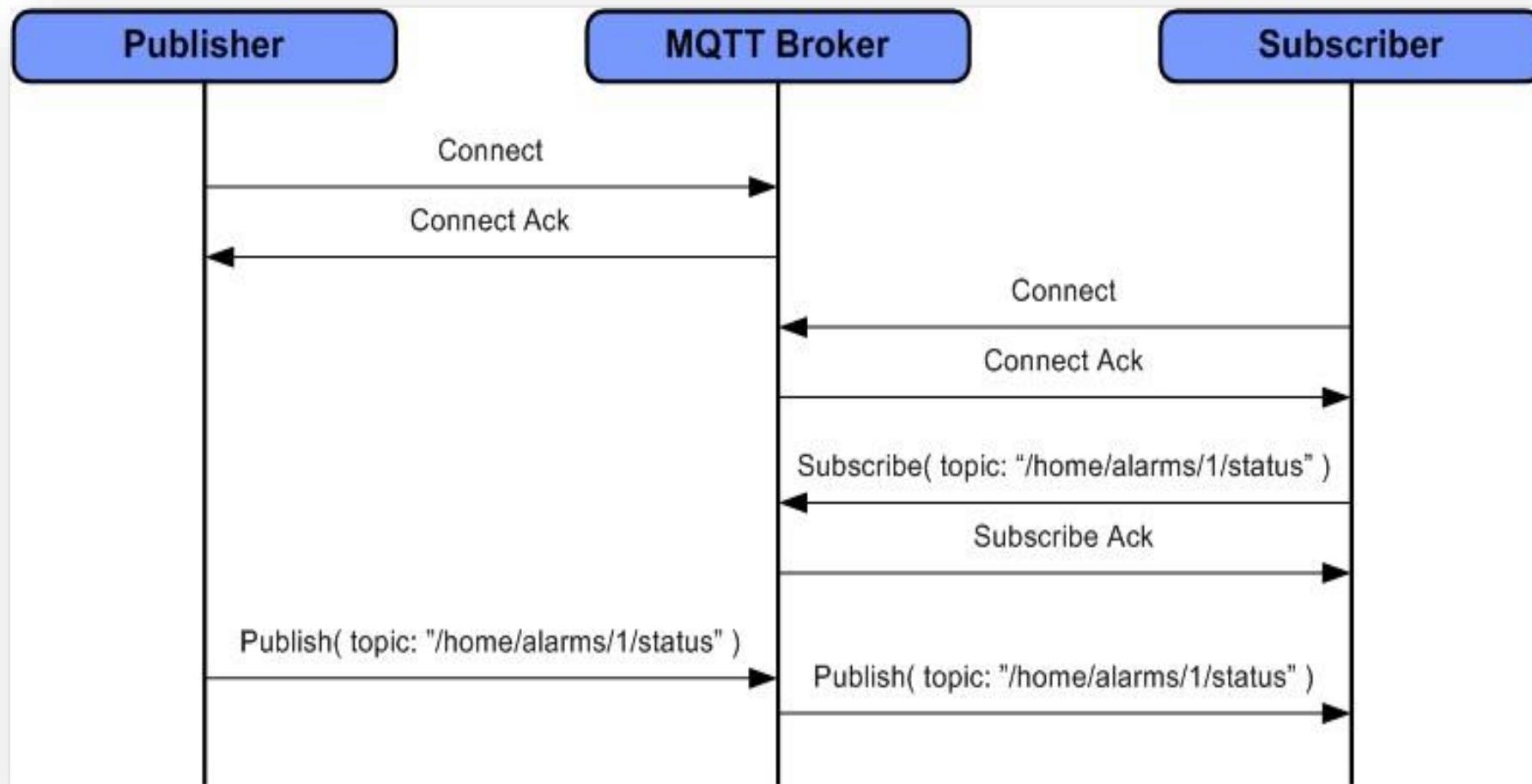
- The subscription lifetime is the same as the time the subscriber is connected to the server / broker.
- Messages are **not stored** on the server/broker.

MQTT publish-subscribe architecture :Control packet

- The MQTT messages are delivered asynchronously (“push”) through publish subscribe architecture.
- The MQTT protocol works by exchanging a series of MQTT control packets in a defined way.
- Each control packet has a specific purpose and every bit in the packet is carefully crafted to reduce the data transmitted over the network.
- A MQTT topology has a **MQTT server** and a **MQTT client**
- MQTT client and server communicate through different control packets.

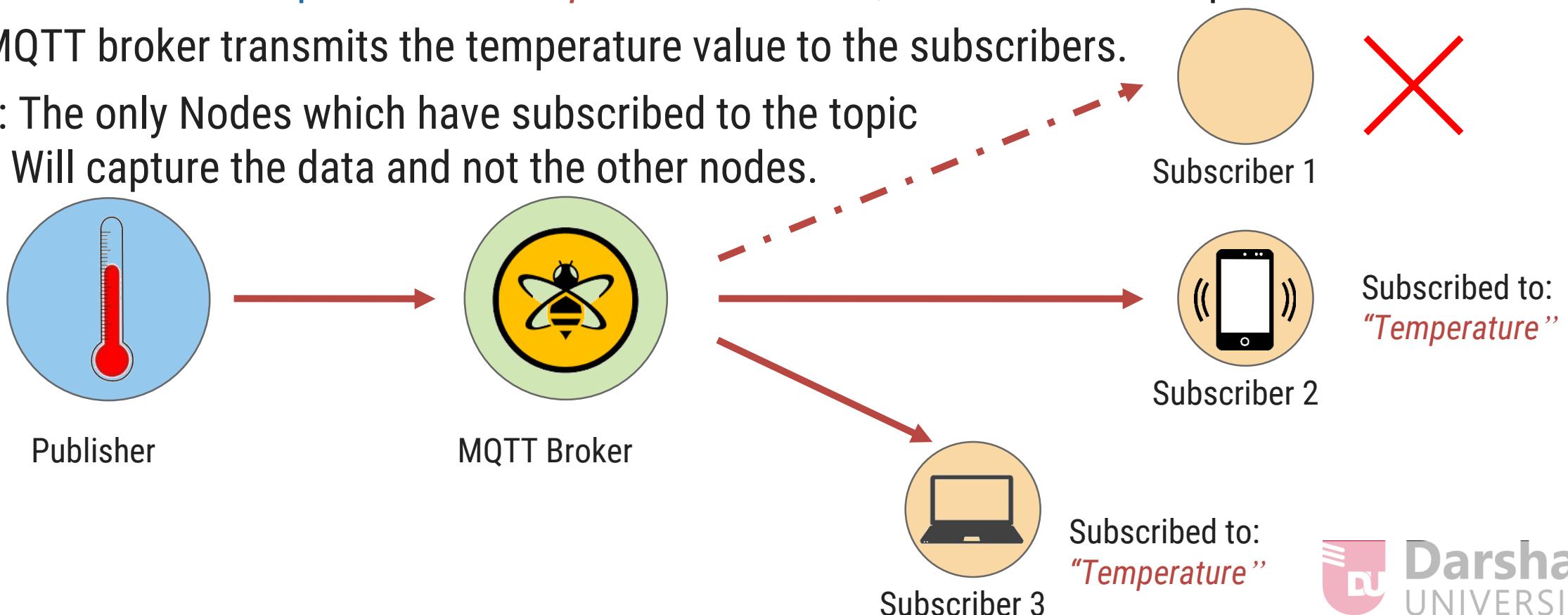
| Control packet | Direction of flow | Description |
|----------------|--------------------------------------|--|
| CONNECT | Client to Server | Client request to connect to Server |
| CONNACK | Server to Client | Connect acknowledgment |
| PUBLISH | Client to Server or Server to Client | Publish message |
| PUBACK | Client to Server or Server to Client | Publish acknowledgment |
| PUBREC | Client to Server or Server to Client | Publish received (assured delivery part 1) |
| PUBREL | Client to Server or Server to Client | Publish release (assured delivery part 2) |
| PUBCOMP | Client to Server or Server to Client | Publish complete (assured delivery part 3) |
| SUBSCRIBE | Client to Server | Client subscribe request |
| SUBACK | Server to Client | Subscribe acknowledgment |
| UNSUBSCRIBE | Client to Server | Unsubscribe request |
| UNSUBACK | Server to Client | Unsubscribe acknowledgment |
| PINGREQ | Client to Server | PING request |
| PINGRESP | Server to Client | PING response |
| DISCONNECT | Client to Server | Client is disconnecting |

MQTT Communication Sample:



MQTT – Example

- Suppose a temperature sensor is connected to any system and we want to monitor that temperature.
- The controller connected to the temperature sensor publishes the temperature value to the MQTT broker with a **topic** name: "*Temperature*". Hence, it is considered a publisher.
- The MQTT broker transmits the temperature value to the subscribers.
- Note : The only Nodes which have subscribed to the topic Will capture the data and not the other nodes.

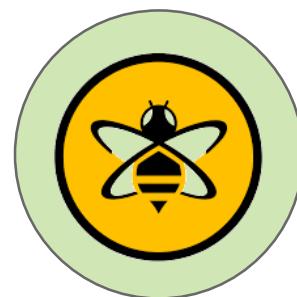


MQTT - Example

- Can publisher and subscriber interchange their role?
- Let us take an example of a **Driverless car**.
- A GPS sensor is connected to the car which gives the current location of the car.
- The system in car, publishes the location to MQTT broker with topic name – **CurrentLocation**.
- The server/computer subscribes to the topic **CurrentLocation** and receives the current location of the car.



Publisher: **CurrentLocation**



MQTT Broker



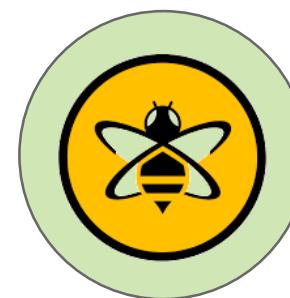
Subscriber: **CurrentLocation**

MQTT - Example

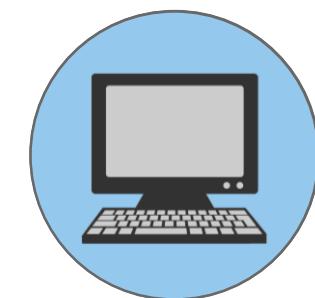
- Based on the currently selected destination, the server/computer computes the direction of the car.
- This data is published to an MQTT broker with a topic named – **Direction**.
- To get the command from the server, the system in the car has to subscribe to the topic named **Direction**.
- According to the command received from the server, the car will run in particular direction.



Publisher: **CurrentLocation**
Subscriber: **Direction**



MQTT Broker

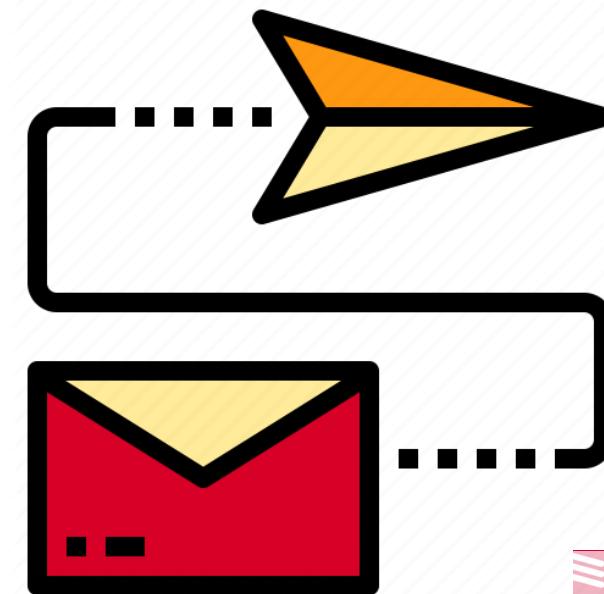
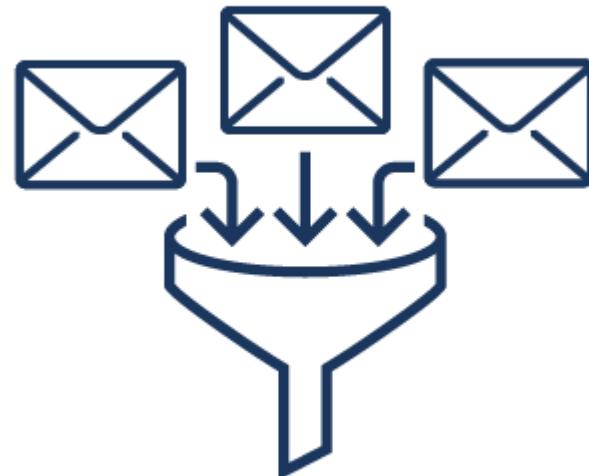


Subscriber: **CurrentLocation**
Publisher: **Direction**

MQTT - Working

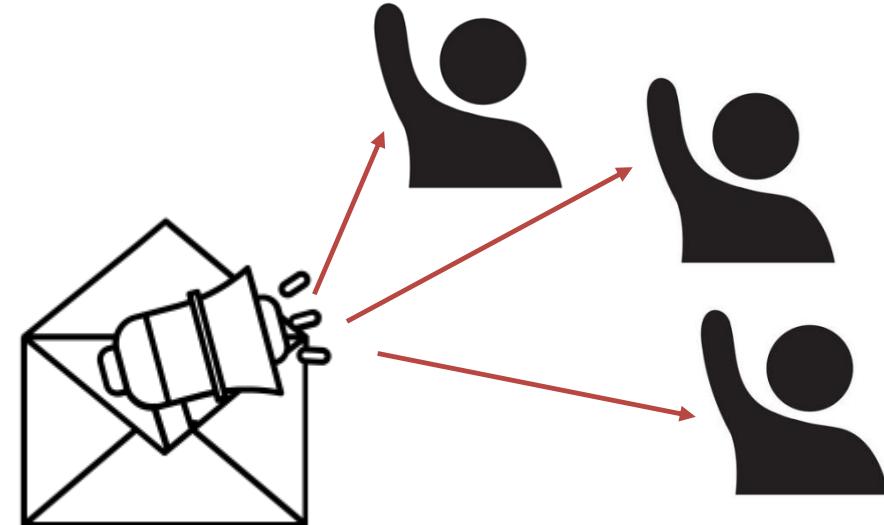
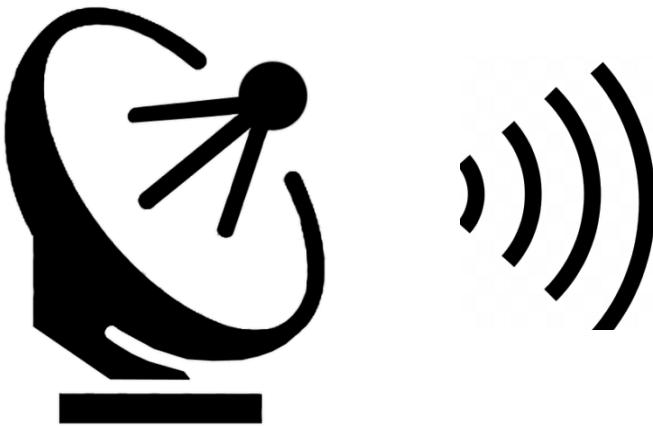
- In MQTT, clients do not have any address like a typical networking scheme.
- The broker filters the messages based on the subscribed topics.
- The messages will then be circulated to respective subscribers.
- The topic name can be anything in string format.
- The MQTT working can be explained with an example of the television broadcast.

~~Client_Address~~



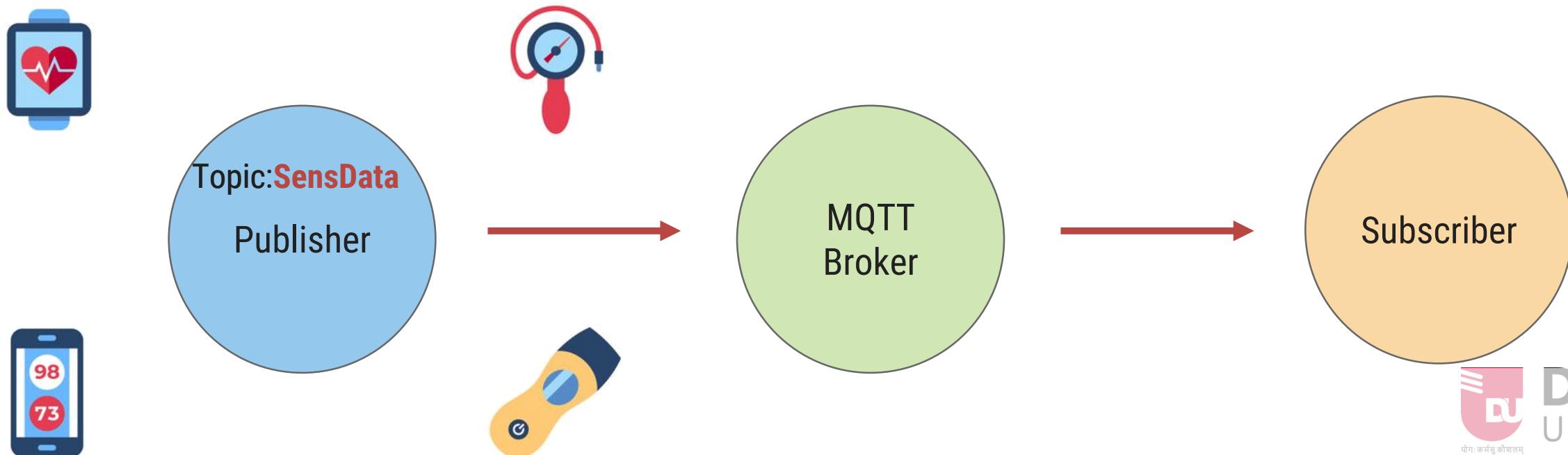
MQTT - Working

- Television broadcaster station broadcasts all the channels.
- The viewers subscribe to their favorite channels only.
- Similarly, in MQTT, the publisher node publishes data with the topic name and interested subscribers subscribe to the topic.
- Note that there can be multiple subscribers of a same topic as well as a single subscriber can be subscribe to multiple topics.



MQTT – Node(s)

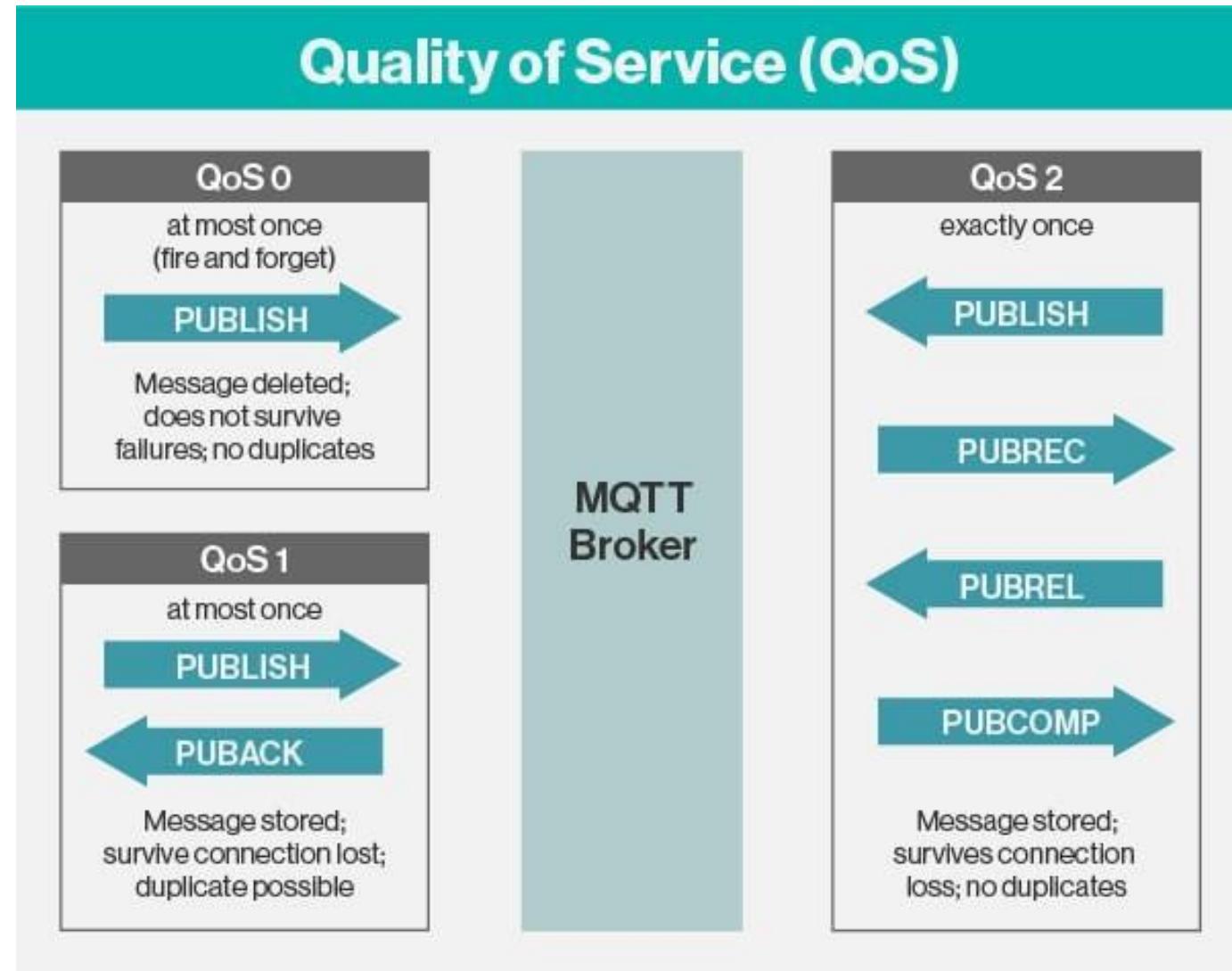
- Node(s) in MQTT collects the information from sensors or other input devices in case of the publisher.
- Connects it to the messaging server known as the MQTT broker.
- A specific string – Topic is used to publish the message.
- Let other nodes understand the topic. These nodes are considered subscribers.
- Any Node can be publisher or subscriber and node is also referred to as client.



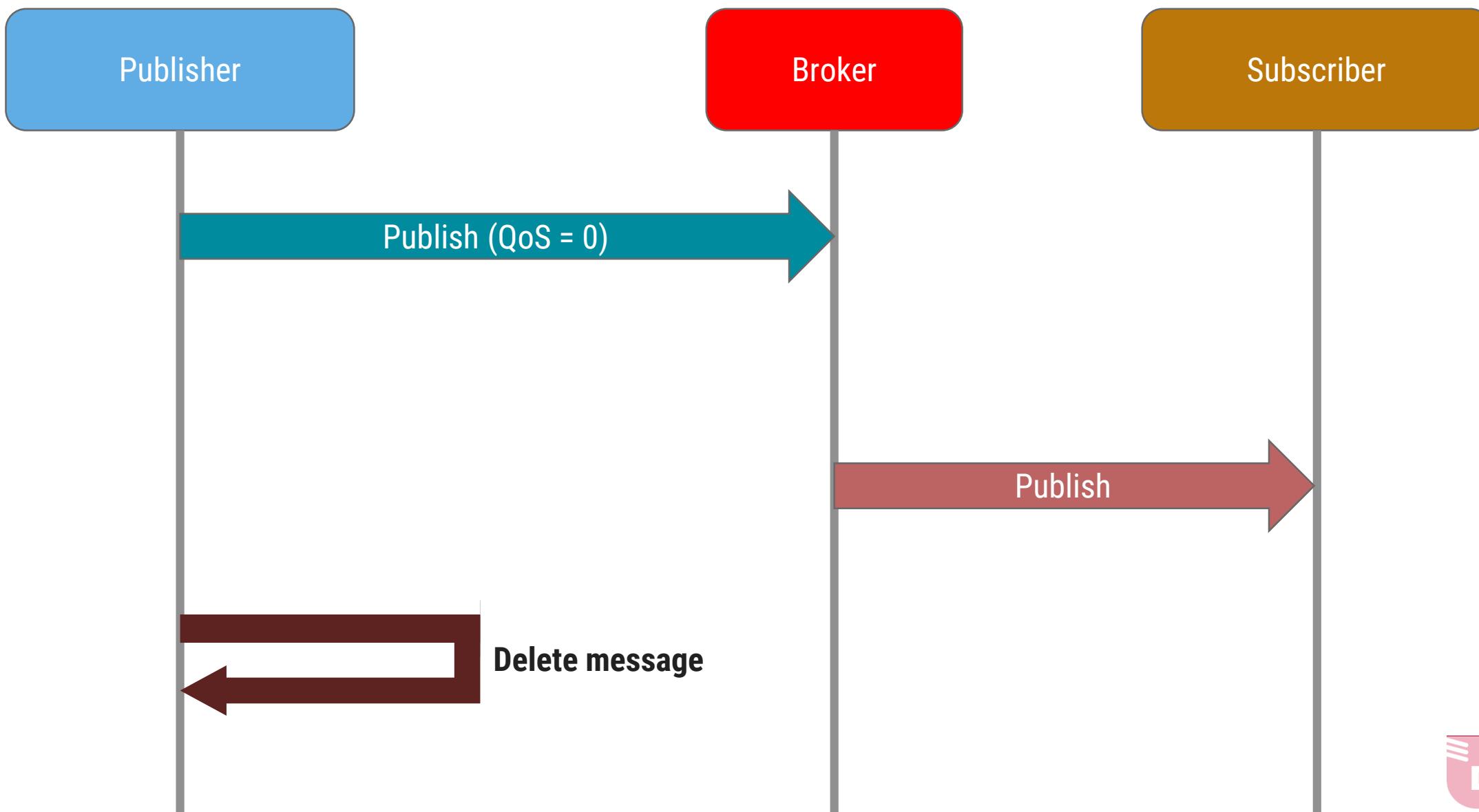
Quality of Service (QoS) for MQTT

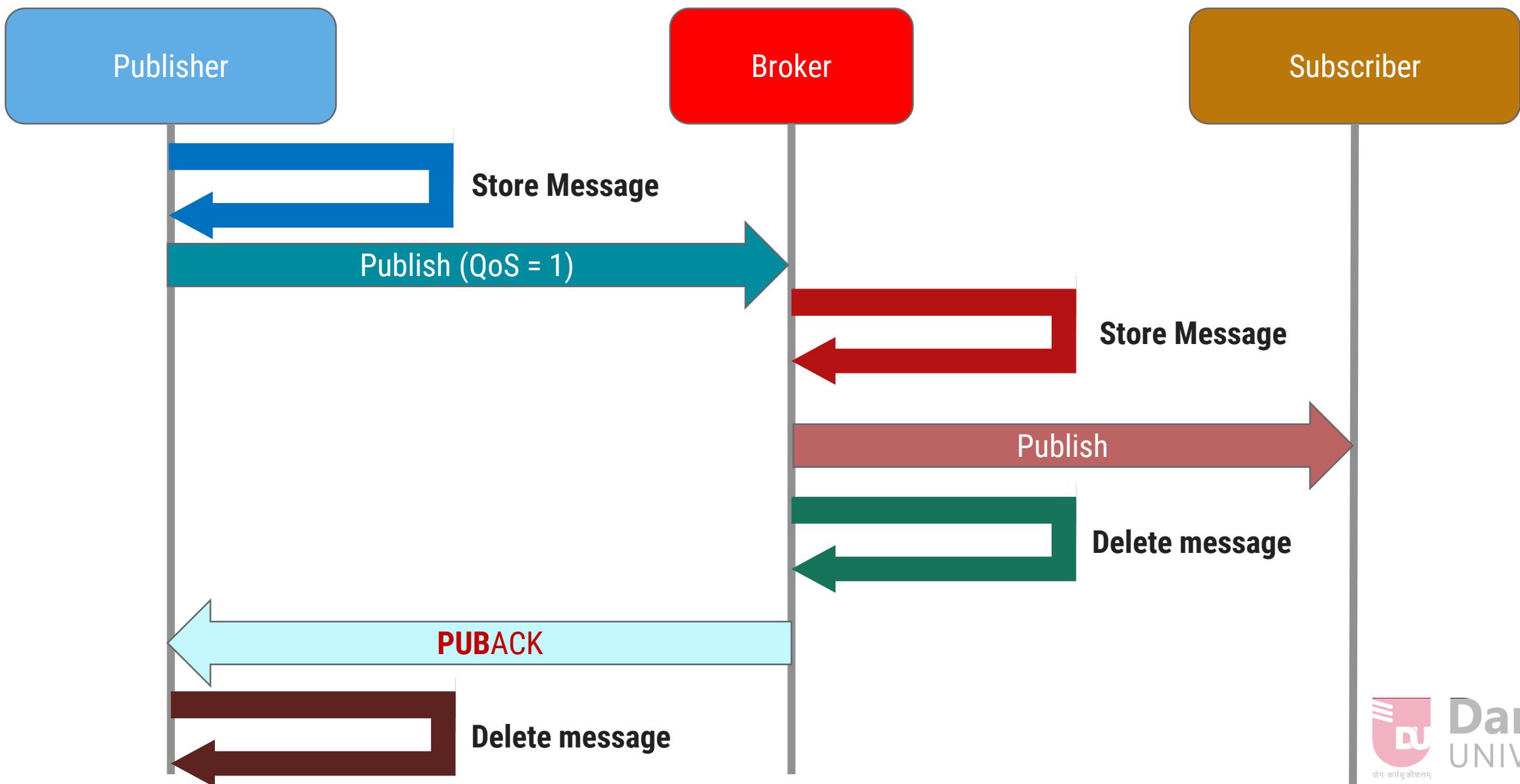
- Quality of service (QoS) levels determine how each MQTT message is delivered and must be specified for every message sent through MQTT.
- Three QoS for message delivery could be achieved using MQTT:
- **QoS 0 =** (At most once) - where messages are delivered according to the best efforts of the operating environment. Message loss can occur.
- **QoS 1 =** (At least once) - where messages are assured to arrive but duplicates can occur.
- **QoS 2 =** (Exactly once) - where message are assured to arrive exactly once.
- There is a simple rule when considering performance impact of QoS :
“The higher the QoS, the lower the performance”.

QoS : Quality of Service

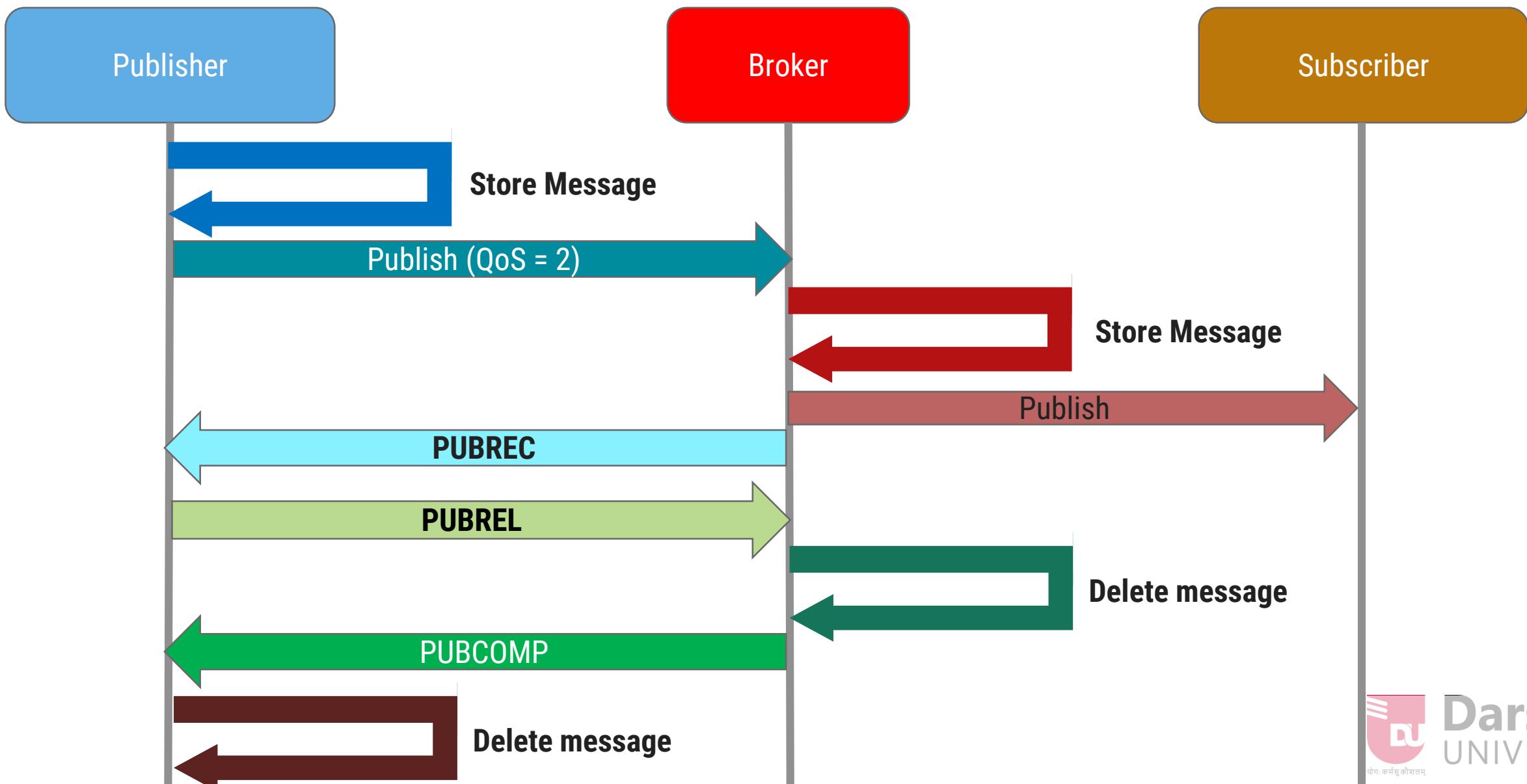


QoS - 0





QoS - 2



MQTT Clients and APIs

- You can develop an MQTT client application by programming directly to the MQTT protocol specification however it is more convenient to use a prebuilt client
- Open Source clients available in **Eclipse Paho project** : Languages : **C, C++, Java, JavaScript, Lua, Python and Go.**
- Clients for other languages are available, see mqtt.org/software. E.g. Delphi, Erlang, .Net, Objective-C, PERL, PHP, Ruby.
- Not all of the client libraries listed on **mqtt.org** are current. Some are at an early or experimental stage of development, whilst others are stable and mature.

Arduino IDE : ESP8266 + MQTT

➤ Library : PubSubClient by Nick O'Leary

The screenshot shows the Arduino Library Manager window. The search bar at the top right contains the text "PubSubClient". Below the search bar, there are two dropdown menus: "Type" set to "All" and "Topic" set to "All". A list of libraries is displayed:

- PubSubClient**
by Oguz Kagan YAGLIOGLU
[BETA] Perfect Lights Custom Devices official library This library depends on PubSubClient and ArduinoJson.
[More info](#)
- PubSubClient**
by Nick O'Leary Version 2.8.0 INSTALLED
A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000.
[More info](#)
Select version [▼](#) [Install](#)
- PubSubClientTools**
by Simon Christmann
Tools for easier usage of PubSubClient Provides useful tools for PubSubClient, however they may consume more power and storage. Therefore it's recommended for powerful microcontrollers like ESP8266.
[More info](#)

In the bottom right corner of the window, there is a "Close" button.



MQTT Publisher – Code Implementation

- The code for MQTT publisher is as shown in below.

MQTT_publisher.ino

```
1 #include <ESP8266WiFi.h>
2 #include <PubSubClient.h>
3
4 const char* ssid = "IoT_Demo";
5 const char* pwd = "diet@iot";
6 const char* mqtt_server =
    "broker.mqtt-dashboard.com";
7
8 WiFiClient espClient;
9 PubSubClient client(espClient);
10 int value = 0;
11 unsigned long lastMsg = 0;
12 #define MSG_BUFFER_SIZE (50)
13 char msg[MSG_BUFFER_SIZE];
```

Including the “**ESP8266WiFi**” library for enabling Wi-Fi connection with protocols.

Including the “**PubSubClient**” library for use of MQTT protocol and their functions.

Defining parameters for Wi-Fi connection.

Defining MQTT server/broker for publishing the message on particular topic.

MQTT Publisher – Code Implementation

MQTT_publisher.ino

```
14 void setup_wifi() {  
15     delay(10);  
16     Serial.println();  
17     Serial.print("Connecting to ");  
18     Serial.println(ssid);  
19     WiFi.mode(WIFI_STA);  
20     WiFi.begin(ssid, pwd);  
21  
22     while(WiFi.status()!=WL_CONNECTED)  
23     {  
24         delay(500);  
25         Serial.print(".");  
26     }  
27     Serial.println("WiFi connected");  
28     Serial.println("IP address: ");  
29     Serial.println(WiFi.localIP());  
30 }
```

Initializing Wi-Fi module as station mode.

Connecting Wi-Fi module with given SSID and Password

Try until Wi-Fi is not connected to given SSID with delay of half second.

MQTT Publisher – Code Implementation

MQTT_publisher.ino

```
31 void setup() {  
32     Serial.begin(115200);  
33     setup_wifi();  
34     client.setServer(mqtt_server, 1883);  
35     while (!client.connected()) {  
36         Serial.print("Attempting MQTT");  
37         String clientId="ESPClient1234";  
38         if (client.connect(  
39             clientId.c_str())) {  
40             Serial.println("connected");  
41             client.publish("darshan/6thsem/ce", "You are in Darshan");  
42         }  
43     }
```

Connects the client to the MQTT Server stored in *mqtt_server* with port number **1883**.

Execute the syntax in loop until ESP is not connected to MQTT broker.

Defining a *clientId* for MQTT server/broker.

Returns the pointer of string *clientId* to the array.

Publishing announcement message after connecting to the MQTT broker with Topic name : **darshan/6thsem/ce**

MQTT Publisher – Code Implementation

MQTT_publisher.ino

```
44 void loop() {  
45  
46     client.loop();  
47     unsigned long now = millis();  
48     if (now - lastMsg > 2000)  
49     {  
50         lastMsg = now;  
51         ++value;  
52         sprintf (msg, MSG_BUFFER_SIZE,"hello world #%ld", value);  
53         Serial.print("Publish message:");  
54         Serial.println(msg);  
55         client.publish("darshan/6thsem/ce", msg);  
56     }  
57 }  
58 }
```

Publishes the *msg* in the given topic name:
darshan/6thsem/ce

This function allows client to process incoming messages, publish data and refresh the connection.

Creating a string in *msg* with given buffer size and assigning the value “hello world” with concatenation of integer named *value*.

MQTT Subscriber – Code Implementation

- The code for MQTT subscriber is as shown in below.

MQTT_subscriber.ino

```
1 #include <ESP8266WiFi.h>
2 #include <PubSubClient.h>
3
4 const char* ssid = "IoT_Demo";
5 const char* pwd = "diet@iot";
6 const char* mqtt_server =
    "broker.mqtt-dashboard.com";
7
8 WiFiClient espClient;
9 PubSubClient client(espClient);
```

Including the “**ESP8266WiFi**” library for enabling Wi-Fi connection with protocols.

Including the “**PubSubClient**” library for use of MQTT protocol and their functions.

Defining parameters for Wi-Fi connection.

Defining MQTT server/broker for publishing the message on particular topic.

MQTT Subscriber – Code Implementation

MQTT_subscriber.ino

```
10 void setup_wifi() {  
11     delay(10);  
12     Serial.println();  
13     Serial.print("Connecting to ");  
14     Serial.println(ssid);  
15     WiFi.mode(WIFI_STA);  
16     WiFi.begin(ssid, pwd);  
17  
18     while(WiFi.status()!=WL_CONNECTED) {  
19         delay(500);  
20         Serial.print(".");  
21     }  
22     Serial.println("WiFi connected");  
23     Serial.println("IP address: ");  
24     Serial.println(WiFi.localIP());  
25 }  
26
```

Initializing Wi-Fi module as station mode.

Connecting Wi-Fi module with given SSID and Password

Try until Wi-Fi is not connected to given SSID with delay of half second.

MQTT Subscriber – Code Implementation

MQTT_subscriber.ino

```
27 void callback(char* topic,  
byte* payload, unsigned int length)  
{  
    Serial.print("Message arrived [");  
    Serial.print(topic);  
    Serial.print("] ");  
    for (int i = 0; i < length; i++)  
    {  
        Serial.print((char)payload[i]);  
    }  
    Serial.println();  
}
```

A *callback* function for receiving messages every time when *client.loop()* executes.

This function accepts the arguments *char* of topic, char* of payload* and the *length of payload*.

Executing the for loop until all the characters of the payload are printed.

Typecasting of *payload[i]* into equivalent character and then printing on serial monitor.

MQTT Subscriber – Code Implementation

MQTT_subscriber.ino

```
38 void setup() {  
39   Serial.begin(115200);  
40   setup_wifi();  
41   client.setServer(mqtt_server,1883);  
42   client.setCallback(callback);  
43   while (!client.connected()) {  
44     Serial.print("Attempting MQTT");  
45     String clientId="ESPClient1234";  
46     if (client.connect(  
47           clientId.c_str())) {  
48       Serial.println("connected");  
49       client.subscribe("darshan/6thsem/ce");  
50     }  
51   }  
52 }  
53 void loop() {  
54   client.loop(); }  
55 }
```

This is built-in function `client.setCallback()` in `PubSubClient` library and it calls a function `callback()` every time when `client.loop()` function is executed.

Subscribing to the Topic named “darshan/6thsem/ce”

Sensors And Actuators

- Sensors and actuators are critical components of Embedded systems.
- These are utilized in various real-world applications, including flight control systems in aircraft, process control systems in nuclear reactors, and power plants that require automated control.
- Sensors and actuators differ primarily in their purpose; the sensor is utilized to **track environmental changes** by measurement, whereas the actuator is utilized when monitoring is **combined with control**, such as controlling physical changes.

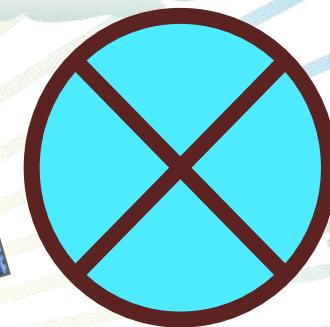
Sensors

- **Sensors:** Sensors are the ways of getting information into your device, finding out things about your surroundings.
- A sensor is a **device** that detects changes and events in a physical environment.
- It may convert **physical parameters** like humidity, pressure, temperature, heat, motion, etc., into **electrical signals**.
- Electrical signal can be converted into a human-readable display and sent across a network for additional processing.
- Types of Sensor (Based on power requirement) :
 - **Active sensors**
 - **Passive sensors.**
- Active sensors necessitate a power supply, whereas passive sensors don't require a power supply.

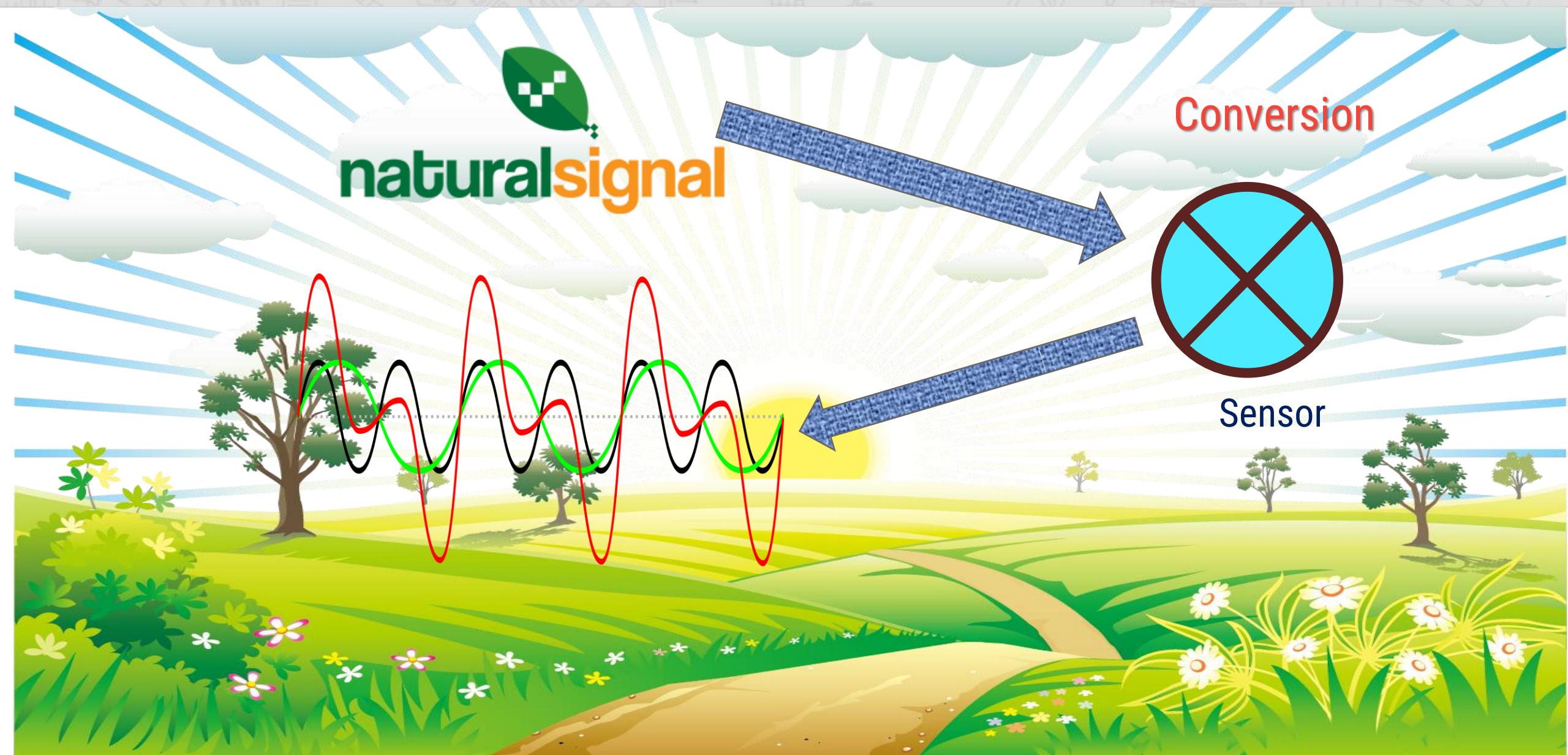
Sensors Functionality



Conversion



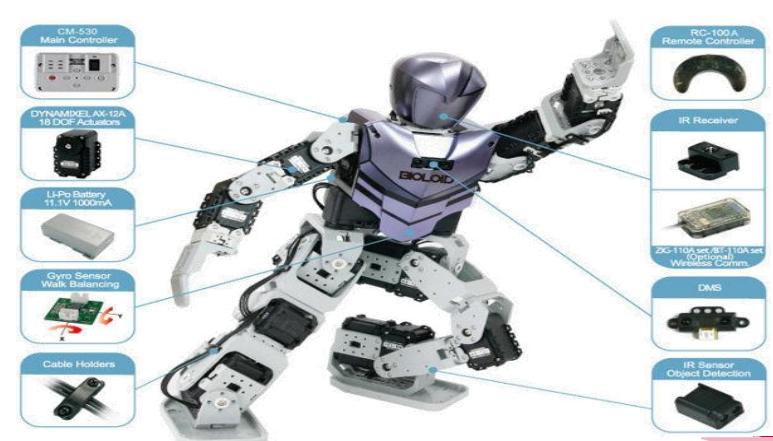
Sensor



Sensors



| Human Sensor | Equivalent Robot Sensor |
|--------------|--------------------------------------|
| eyes | light sensor, ultrasonic sensor |
| ears | sound sensor |
| skin | touch, temperature, pressure sensors |
| nose | none yet... |
| tongue | none yet... |

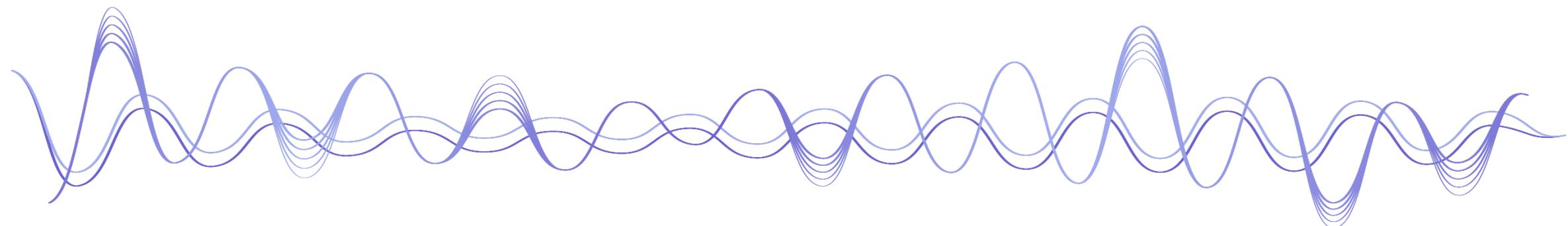


Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

Sensors Types (Based on Sensor's Output Signal)

- The sensors are **input devices** and there are two types of sensors.
- **Analog Sensors** : Generates Analog output Voltage.



- **Digital Sensors** : Generates Digital output Voltage



Sensors

Obstacle Sensor

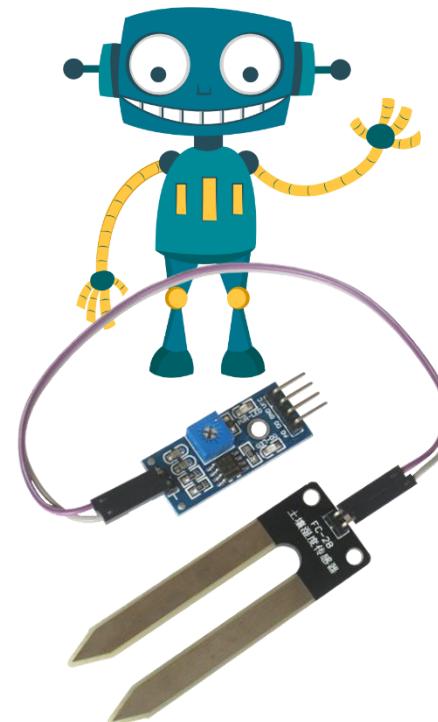


MQ-02/05 Gas Sensor

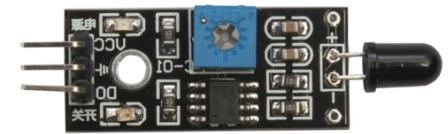
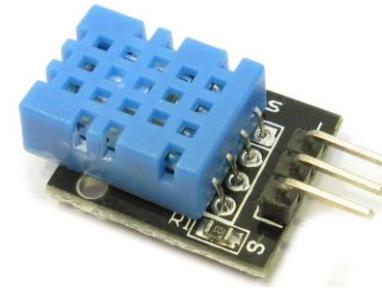


LDR Sensor

Ultrasonic Distance Sensor



Temperature and Humidity Sensor

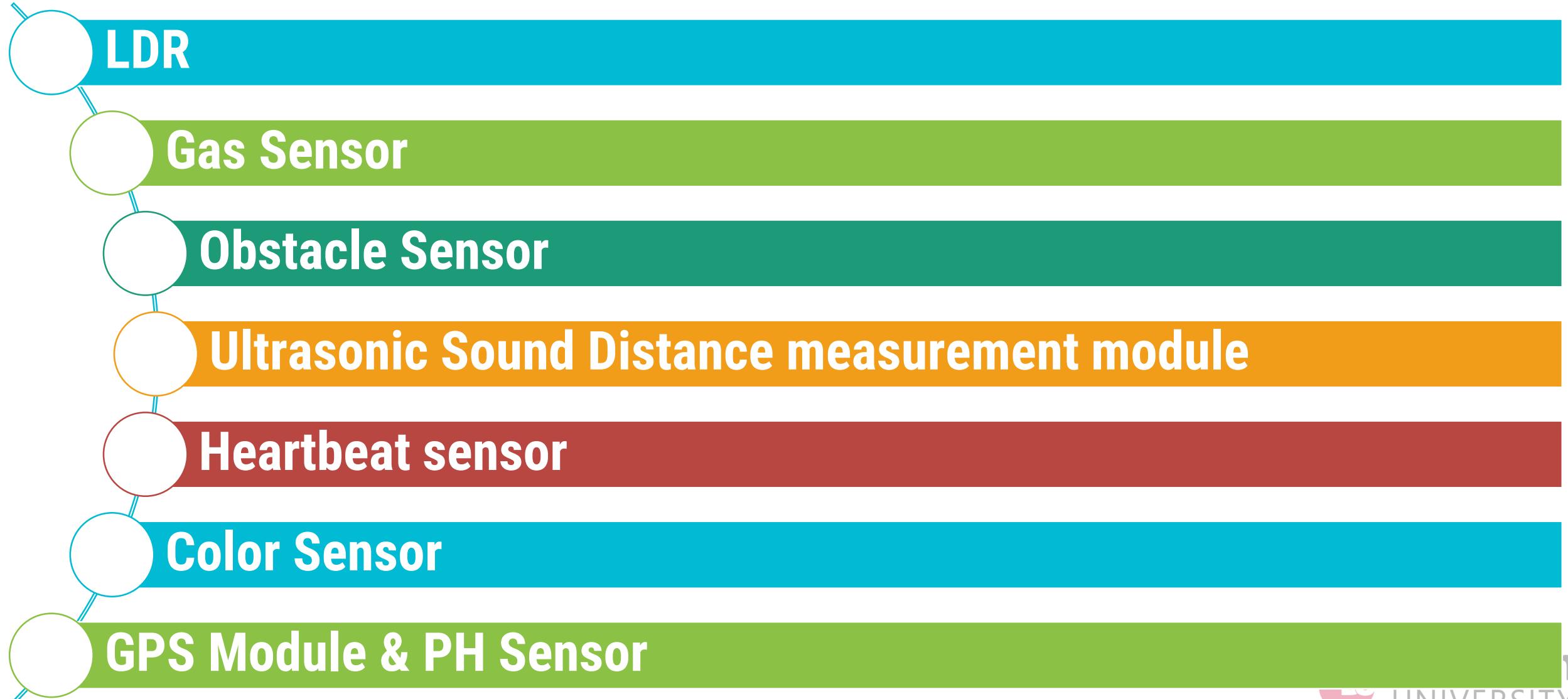


Flame Sensor



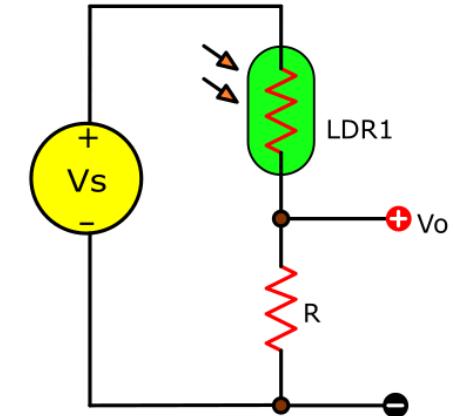
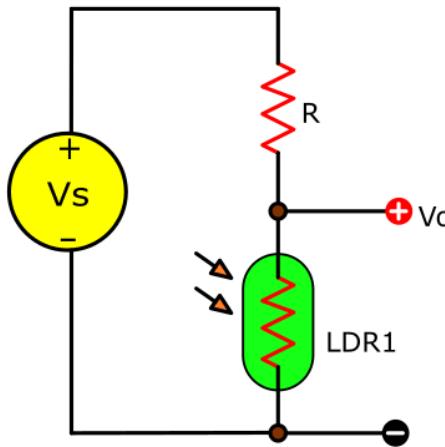
Heartbeat Sensor

Sensor and Sensor Module



LDR Sensor

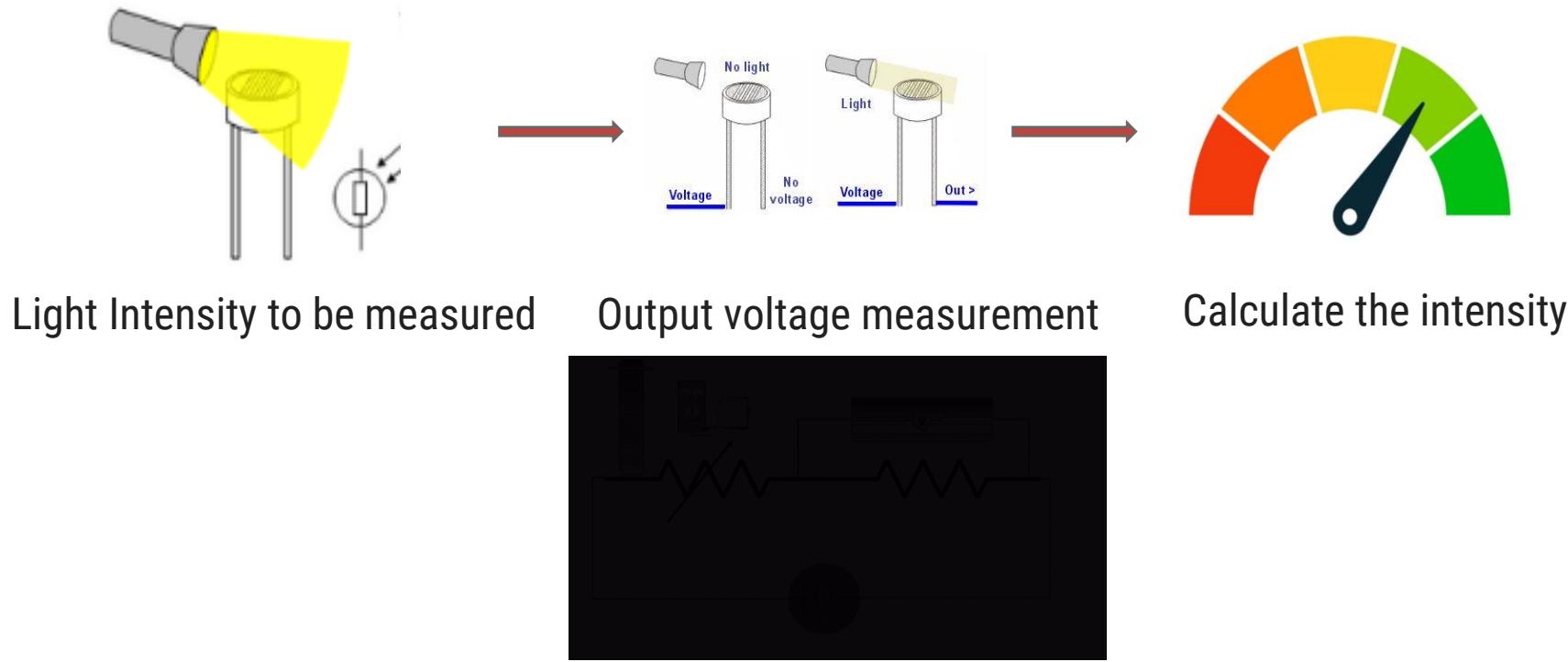
- LDR is known as **Light Dependent Resistor**.



- The internal resistance of LDR changes with respect to light intensity.
- Normally LDR works with **voltage divider network**.
- The LDR is used in the application where the task is performed with light intensity as input.

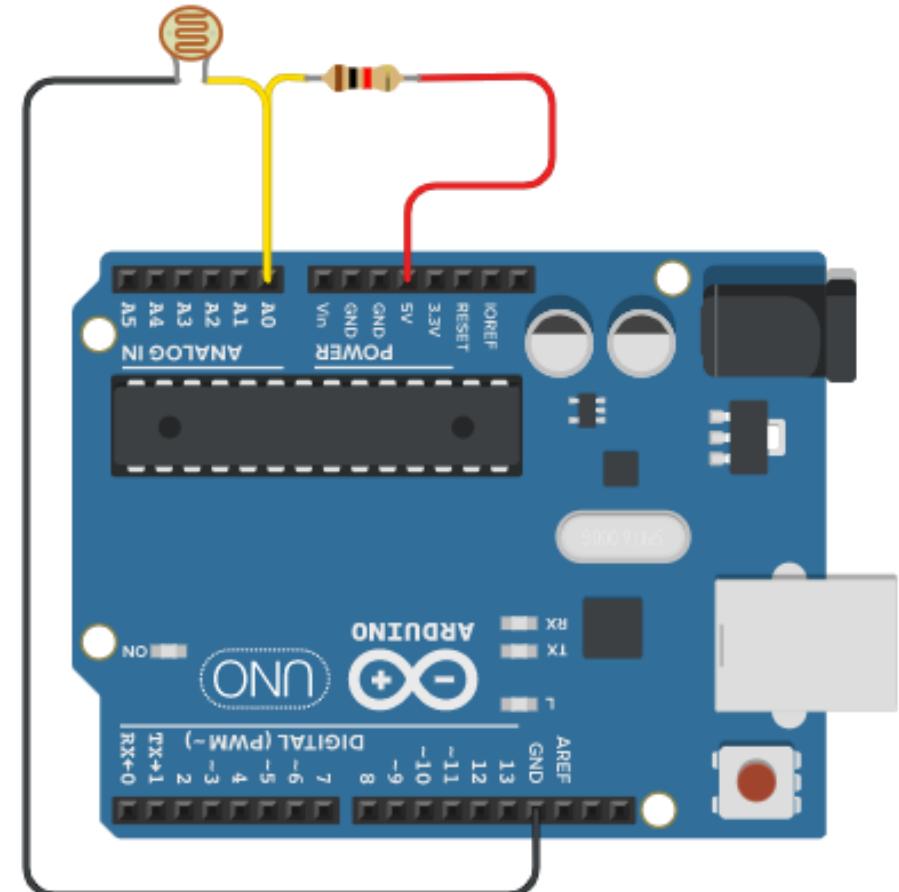
LDR Sensor – Working principle

- As the **light intensity** on LDR is **higher**, the **resistance** of LDR **decreases**.
- The decreased resistance will drop more voltage across LDR according to Ohm's Law.
- The presence of light can be identified by the value of voltage across LDR.



LDR Sensor- Interfacing with Arduino

- The interfacing of LDR sensor with Arduino Uno is as shown in figure.
- Here, one terminal of **10kΩ resistor** is connected to **5V of Arduino** and **second terminal** is connected to **A0 pin**.
- One terminal of **LDR** is connected to **A0** pin of Arduino and other is connected to **Gnd**.
- This creates a voltage divider network between fixed resistor and LDR.
- The voltage at A0 pin remains near to 5V when LDR is in dark (high resistance).
- The voltage at A0 pin remains near to 0V when LDR is in Bright light (low resistance).

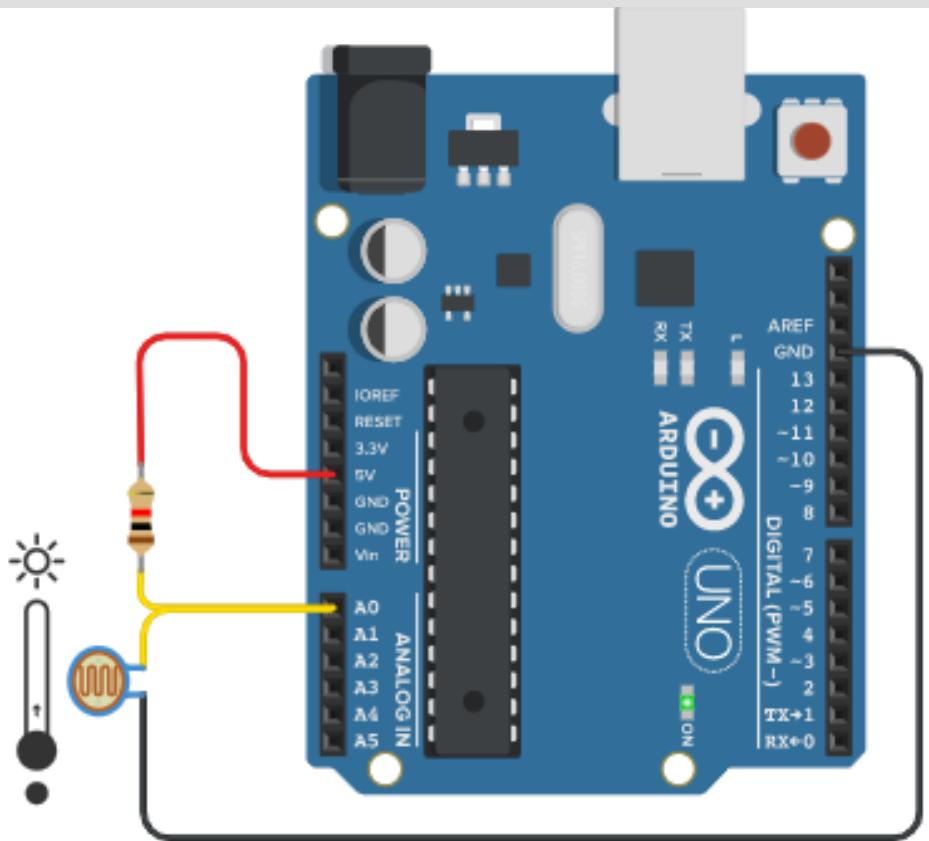


LDR Sensor – Code explanation

Light-intensity-measure.ino

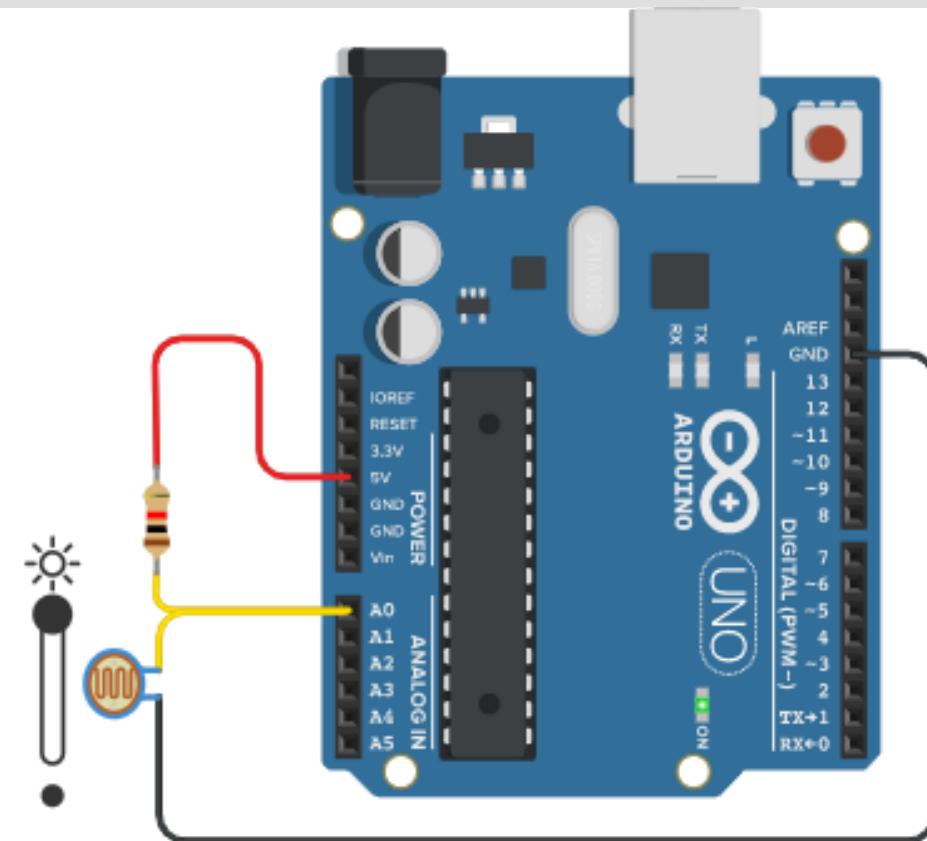
```
1 void setup()
2 {
3     pinMode(A0, INPUT);
4     Serial.begin(9600);
5 }
6
7 void loop()
8 {
9     int light_intensity = analogRead(A0);
10    if (light_intensity > 750)
11    {
12        Serial.println("Darker");
13    }
14    else if (light_intensity < 400)
15    {
16        Serial.println("Too Bright");
17    }
18    delay(1000);
19 }
```

LDR Sensor – Output



Serial Monitor

Darker
Darker
Darker
Darker



Serial Monitor

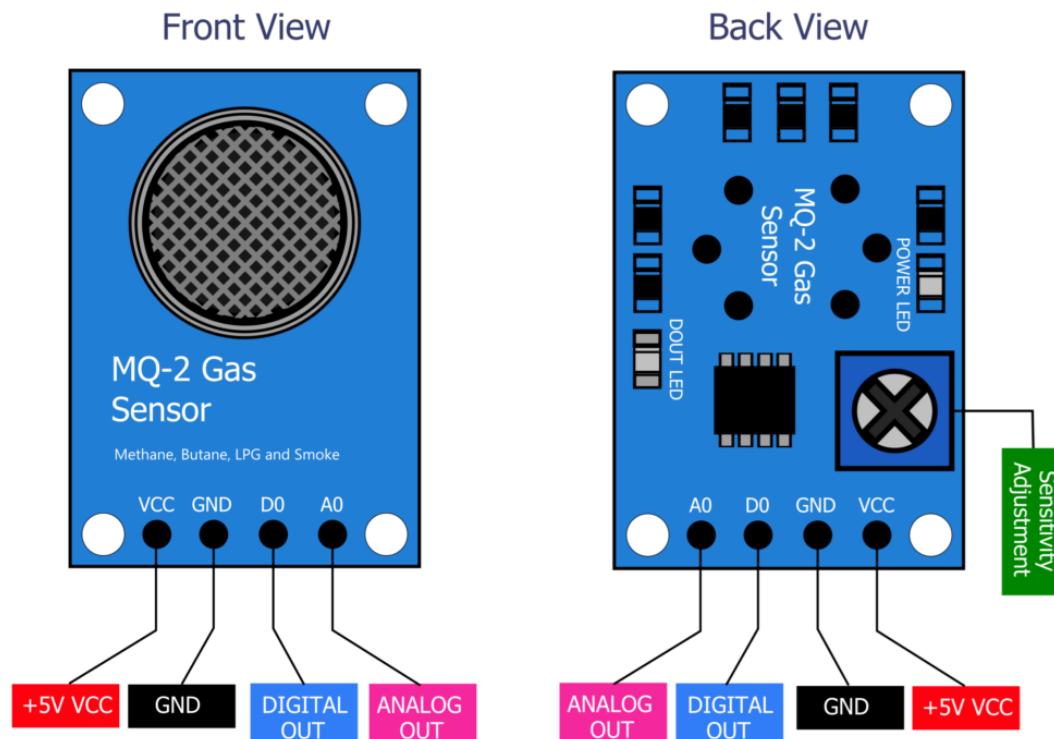
Too Bright
Too Bright
Too Bright
Too Bright



Gas Sensor : MQ-02/05

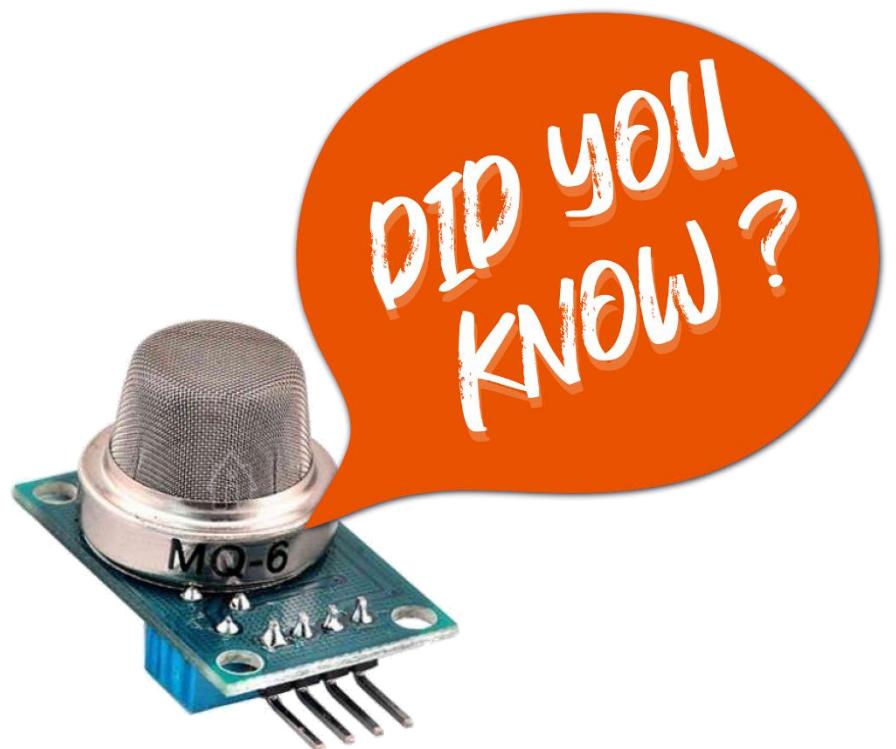
- MQ -02 sensor is used to detect smoke.
- H₂, LPG, CH₄ alcohol and smoke can be detected by MQ-02.
- MQ-05 is not sensitive to smoke, hence less used in the application.
- Pin-out of MQ-02

- ❖ Vcc – Connected to 5V
- ❖ Gnd – Connected to ground
- ❖ D0 – Digital output pin
- ❖ A0 - Analog Outout pin



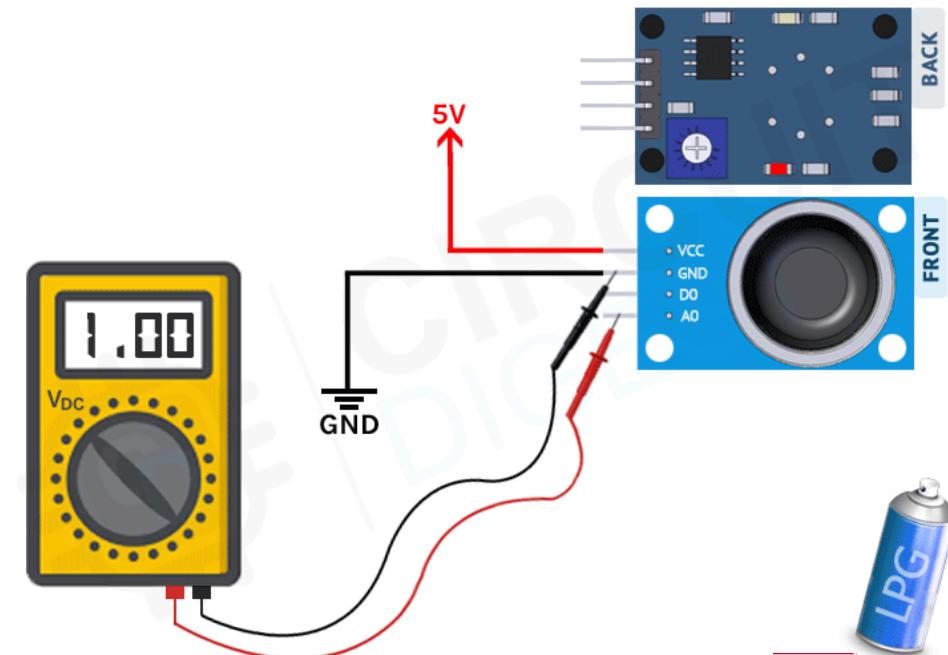
MQ Series Sensor

- MQ-2 - Methane, Butane, LPG, smoke
- MQ-3 - Alcohol, Ethanol, smoke
- MQ-4 - Methane, CNG Gas
- MQ-5 - Natural gas, LPG
- **MQ-6 - LPG, butane gas**
- MQ-7 - Carbon Monoxide
- MQ-8 - Hydrogen Gas
- MQ-9 - Carbon Monoxide, flammable gasses
- MQ131 - Ozone
- MQ135 - Air Quality (CO, Ammonia, Benzene, Alcohol, smoke)
- MQ136 - Hydrogen Sulphide gas
- MQ137 - Ammonia
- MQ138 - Benzene, Toluene, Alcohol, Acetone, Propane, Formaldehyde gas, Hydrogen
- MQ214 - Methane, Natural gas



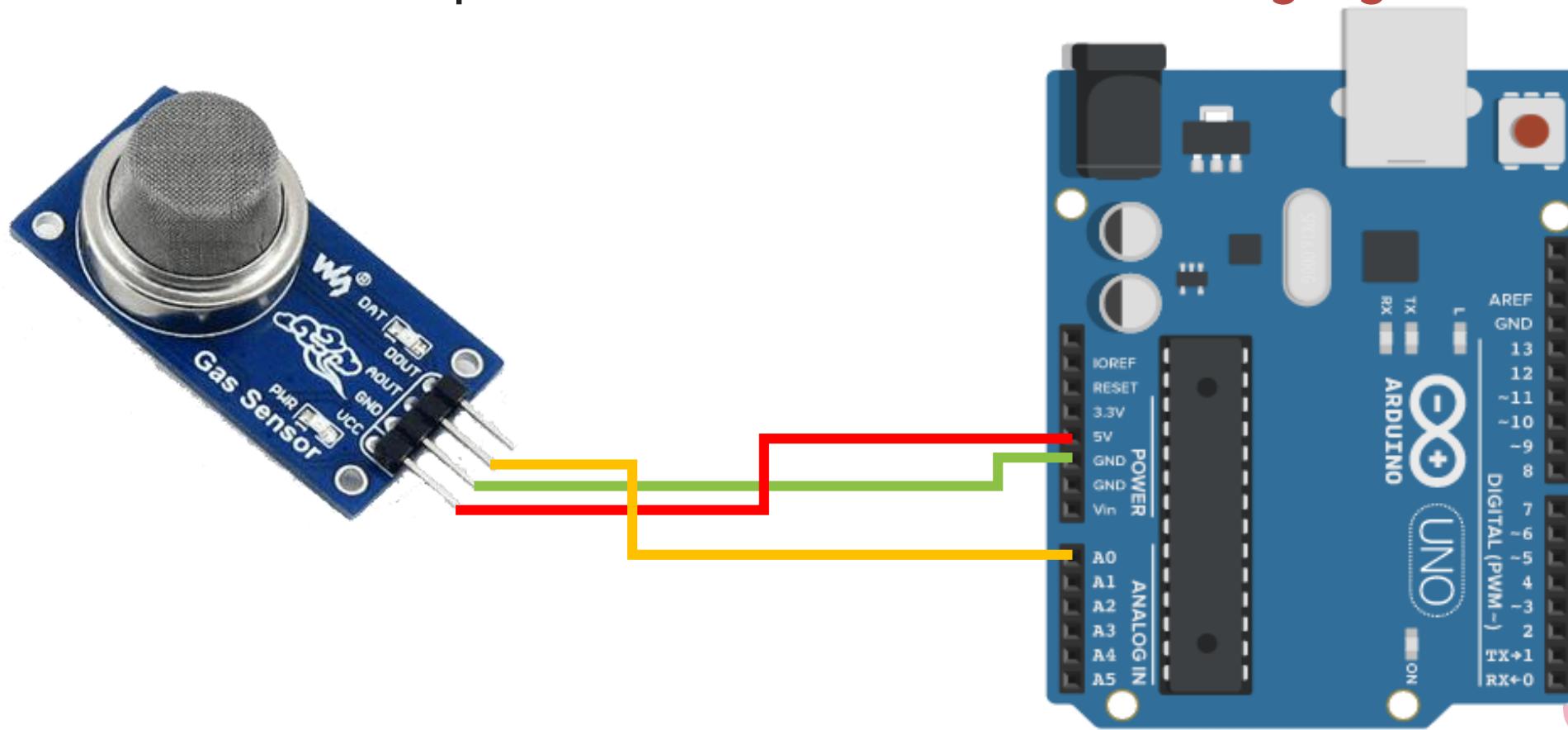
MQ-02/05 Gas Sensor – Working principle

- When any flammable gas passes through the coil of the sensor, the coil burns and Internal Resistance Decreases.
- This results in an Increased Voltage across it. Hence we get variable voltage at Analog Out (A0) pin of MQ-05 gas sensor.
- If gas present -> voltage is high.



MQ-02/05 Gas Sensor – Interfacing with Arduino

- The interfacing of MQ-02 with Arduino Uno is as shown in the figure.
- Here, Vcc and Gnd pins of MQ-02 are connected to 5V and GND of Arduino board.
- We want to take the input from a Gas sensor in **an analog signal**.



MQ-02/05 Gas Sensor – Code With Explanation

gas_sensor.ino

```
1 int gas_level;
2 void setup() {
3     pinMode(A0, INPUT);
4     Serial.begin(9600);
5 }
6
7 void loop() {
8     gas_level = analogRead(A0);
9     Serial.print("Gas Level : ");
10    Serial.println(gas_level);
11    delay(500);
12 }
```

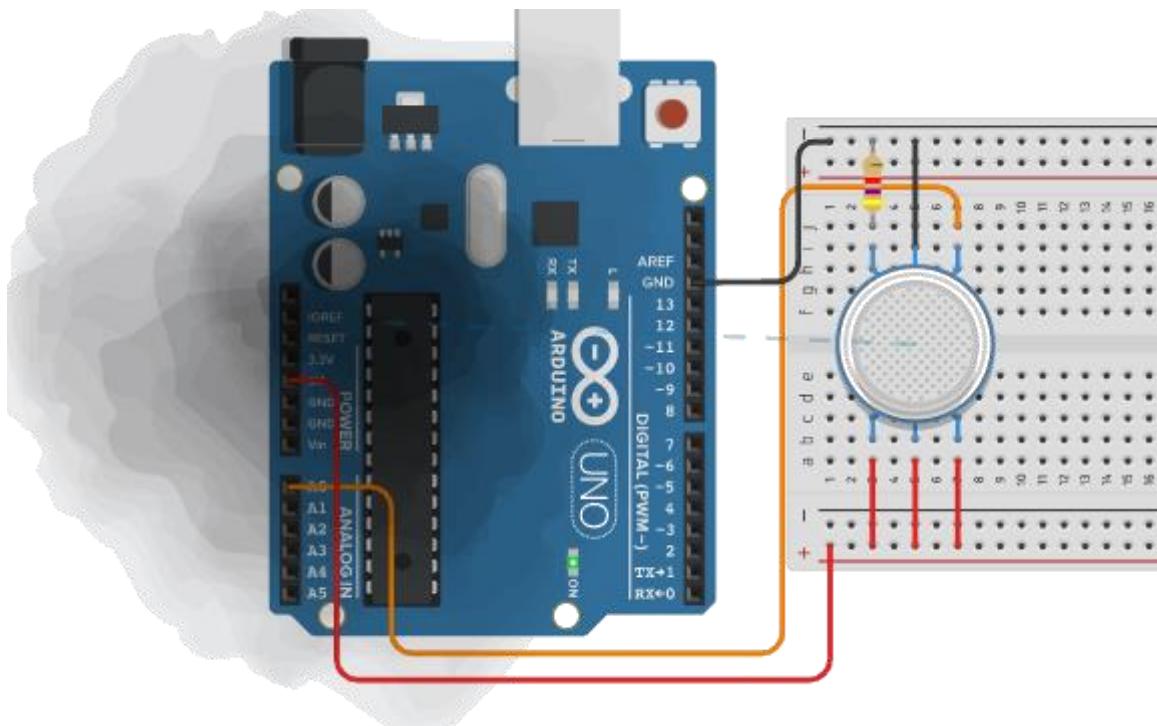
Initialize the serial communication between Arduino and computer with baud rate 9600.

Reads the analog value from specified pin and stores it in the mentioned variable.

Prints data on the serial port in ASCII text given in double quotes.

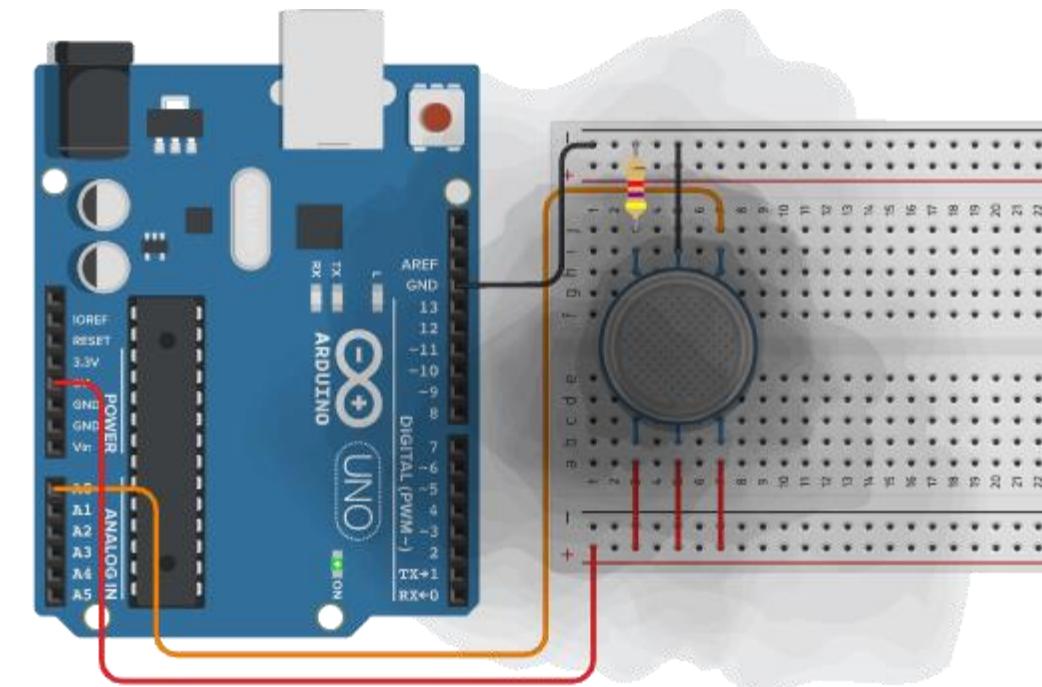
Prints the data of specified variable on the serial port and also prints new line at the end.

MQ-02/05 Gas Sensor – Output



Serial Monitor

```
Gas Level : 306
```

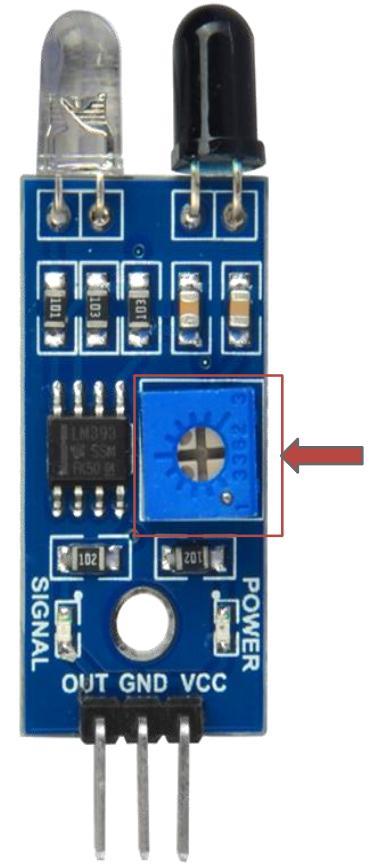
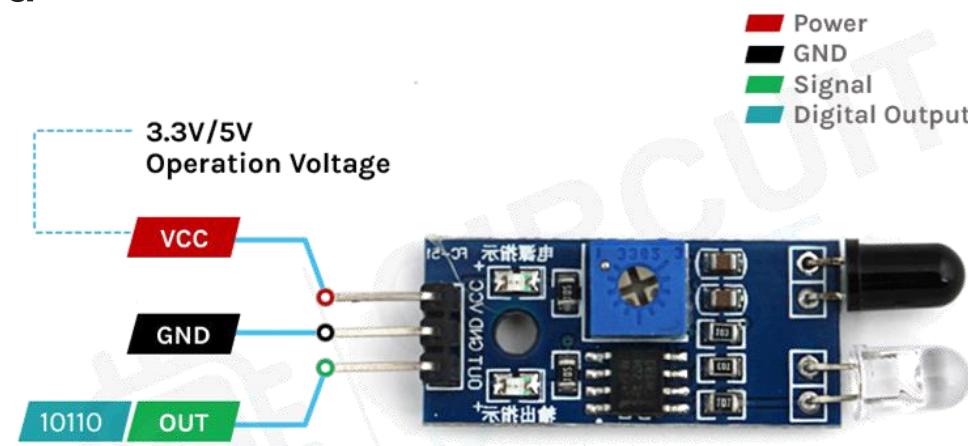


Serial Monitor

```
Gas Level : 745
```

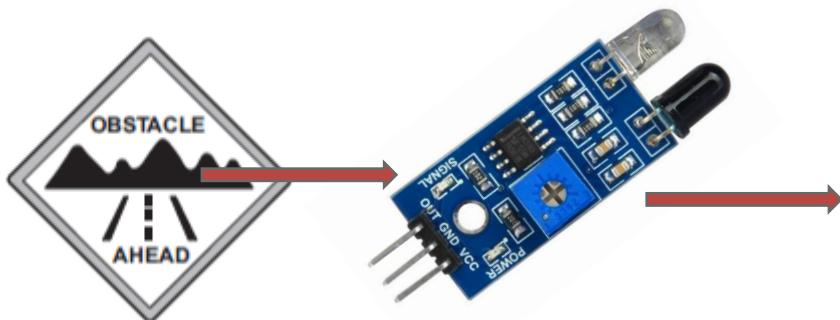
Obstacle Sensor

- Obstacle sensor is used for **detection of obstacle**.
- It is a digital sensor, hence gives binary output '1' or '0'.
- The **range** for detection of obstacle can be changed by **potentiometer** given.
- Pin-out :
 - ❖ Vcc – Connected to 5V
 - ❖ Gnd – Connected to ground
 - ❖ Out - Output pin (Digital)



Obstacle Sensor – Working principle

- IR emitter transmits IR signals and IR receiver receives those signals from reflection.
- If the obstacle is present then the transmitted signals are **reflected** back by obstacle and if they have amplitude greater than threshold the output signal will be '**0**'.
- If the obstacle is not present then the transmitted signals are **not reflected** and the output signal will be '**1**'.

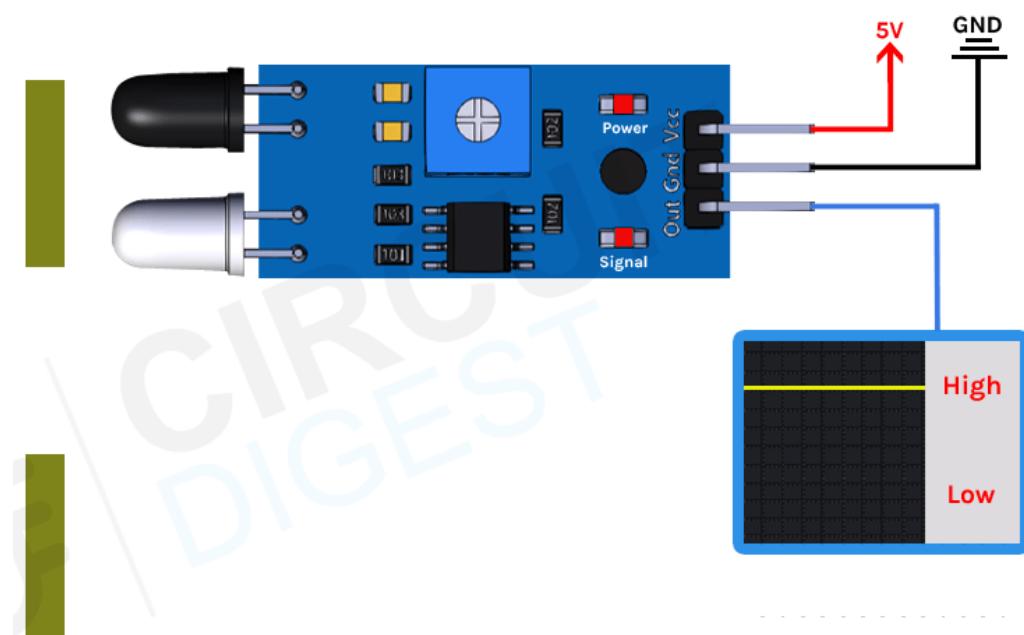


Obstacle in the way

Obstacle IR Sensor

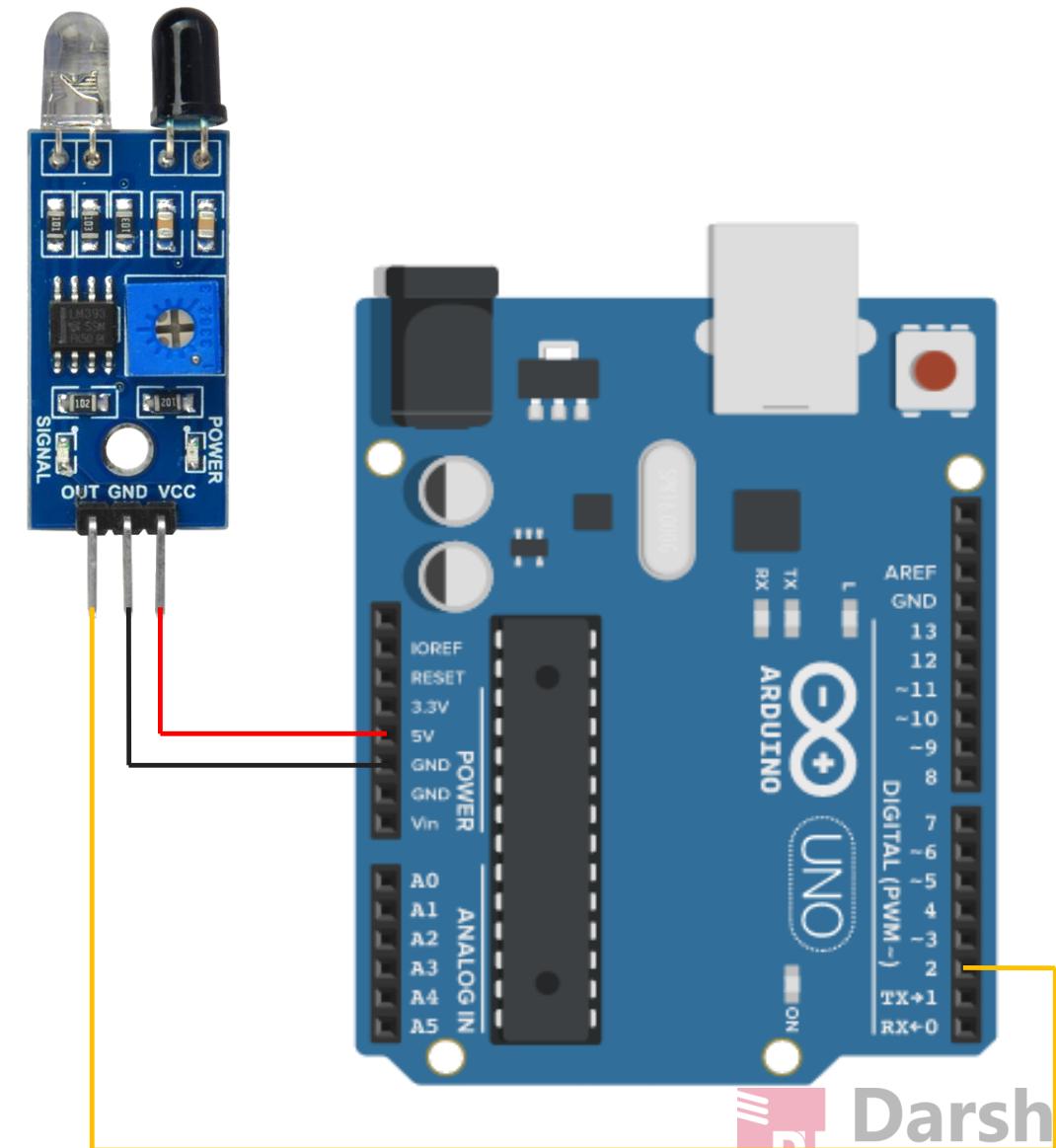


Digital Output



Obstacle Sensor – Interfacing with Arduino

- The interfacing of obstacle sensor with Arduino Uno is as shown in figure.
- Here, Vcc and Gnd pins of obstacle sensor is connected to 5V and Gnd of Arduino board.
- The output of obstacle sensor is in **digital signal**.



Obstacle Sensor – Code explanation

obstacle-detect.ino

```
1 int obstacle_pres=0;
2 void setup() {
3     pinMode(2, INPUT);
4     pinMode(13, OUTPUT);
5     Serial.begin(9600);
6 }
7
8 void loop() {
9     obstacle_pres = digitalRead(2);
10    if (obstacle_pres == LOW)
11    {
12        Serial.print("Obstacle is present");
13        digitalWrite(13,HIGH);
14    } else{
15        digitalWrite(13,LOW);
16        delay(1000);
17 }
```

Reads the digital data from specified pin and stores it into given variable.

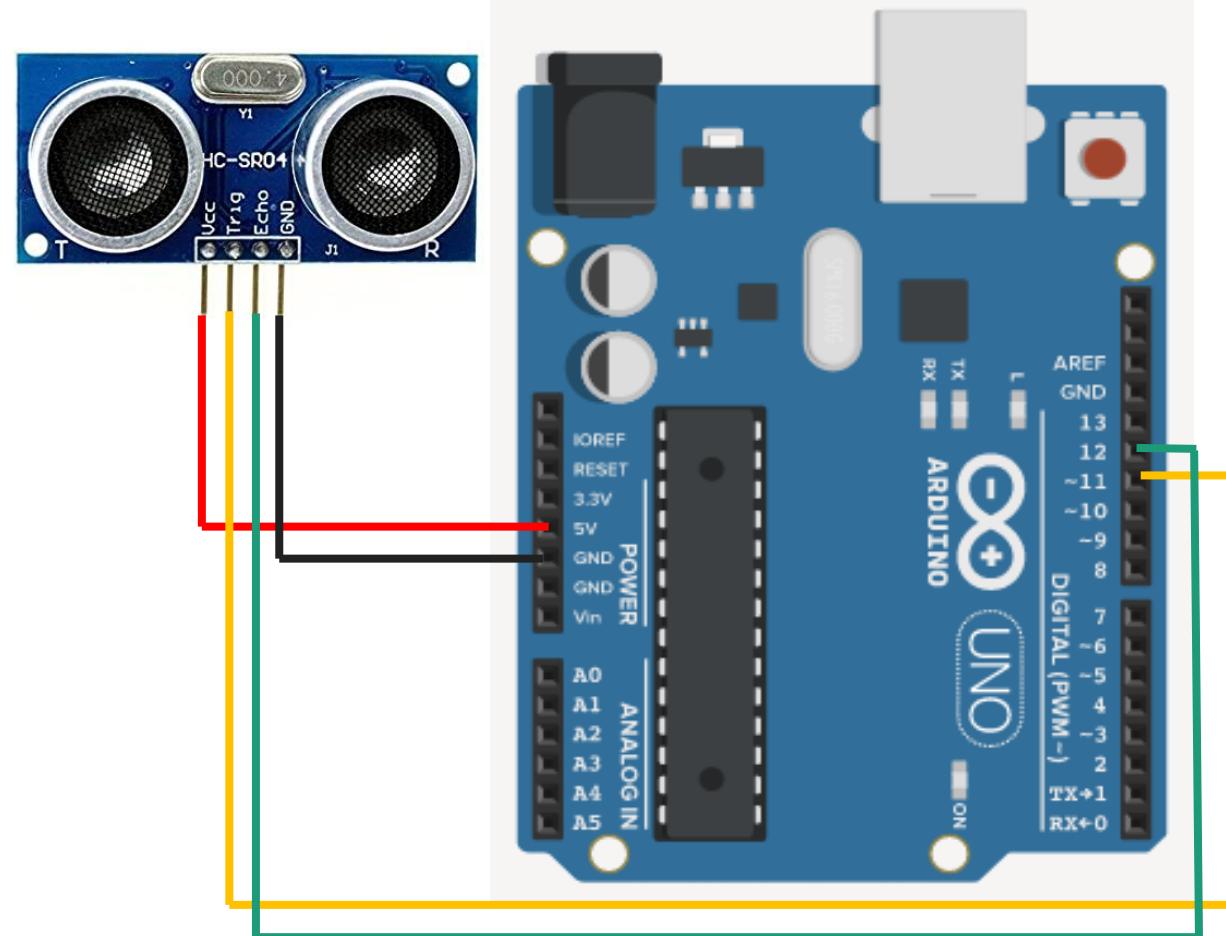
HC-SR04 Ultrasonic Sound Sensor

- Ultrasonic Sound Sensor is used to measure the **distance of an object**.
- The sensor can only measure the distance of object. It can not identify the type of object.
- It is also known as Ultrasonic distance sensor.
- Pin-out of HC-SR04 module are
 - ❖ Vcc – Connected to 5V
 - ❖ Gnd – Connected to ground
 - ❖ Trigger – Generates the transmit pulse
 - ❖ Echo – Receives echo from obstacle



HC-SR04 Ultrasonic Sound Sensor- Interfacing with Arduino

- The interfacing of **HC-SR04 sensor** with Arduino Uno is as shown in figure.
- Vcc** and **Gnd** pins of HC-SR04 sensor are connected to **5V** and **Gnd** of Arduino board.
- The **Trigger** and **Echo** pin is connected to (any) **digital pins** of Arduino board.
- In this case, trigger is connected to 11 and echo is connected to 12.



- The sensor is composed of two ultrasonic transducers.
- One is transmitter which outputs ultrasonic sound pulses and the other is receiver which listens for reflected waves.

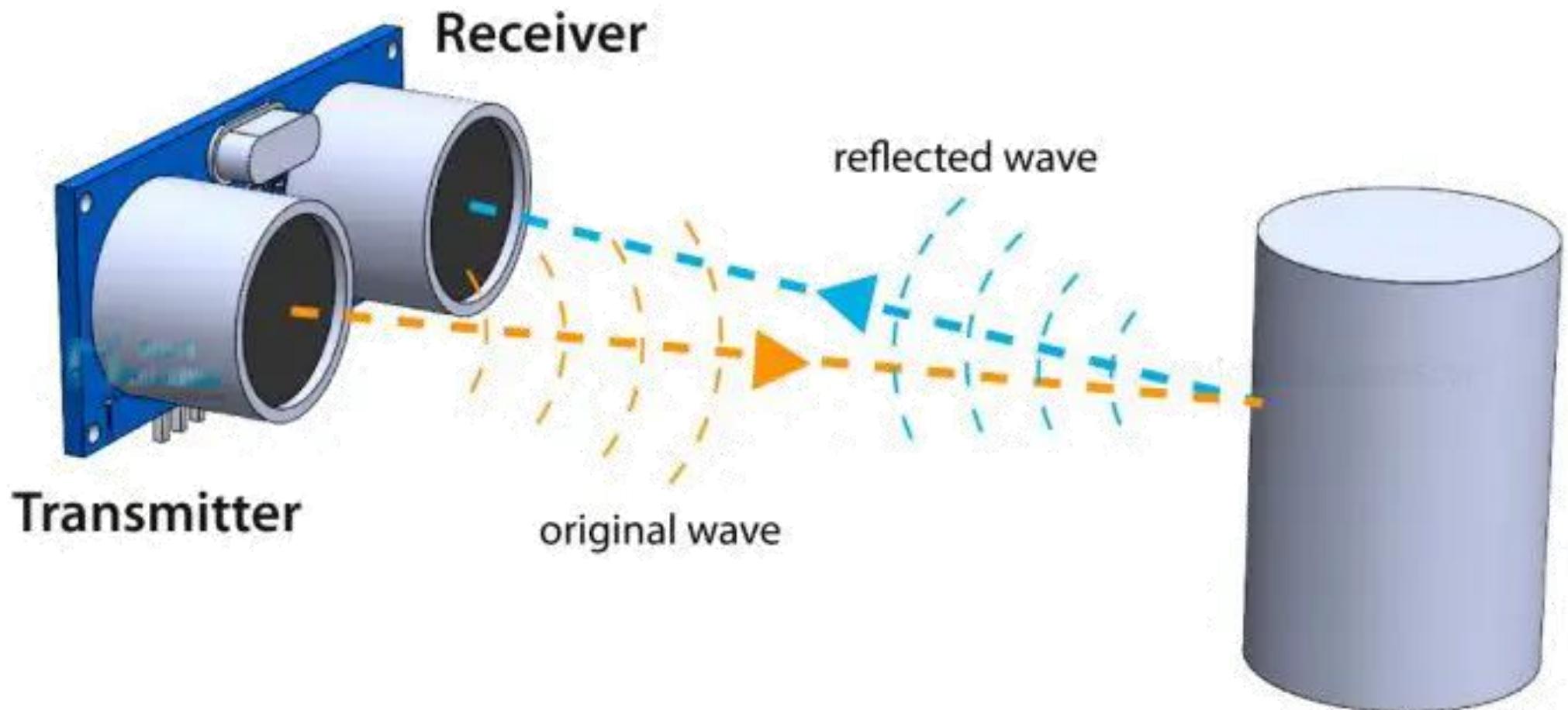
➤ The HC-SR04 Ultrasonic Range Sensor Features:

- ❖ Input Voltage: 5V
- ❖ Current Draw: 20mA (Max)
- ❖ Digital Output: 5V
- ❖ Digital Output: 0V (Low)
- ❖ Working Temperature: -15°C to 70°C
- ❖ Sensing Angle: 30° Cone
- ❖ Angle of Effect: 15° Cone
- ❖ Ultrasonic Frequency: **40kHz**
- ❖ **Speed of sound : 343 mtr/sec**
- ❖ Range: **2cm - 400cm** (about an inch to 13 feet)

$$\frac{343 * 100 \text{ cm}}{10^6 \mu\text{sec}} = 0.0343$$

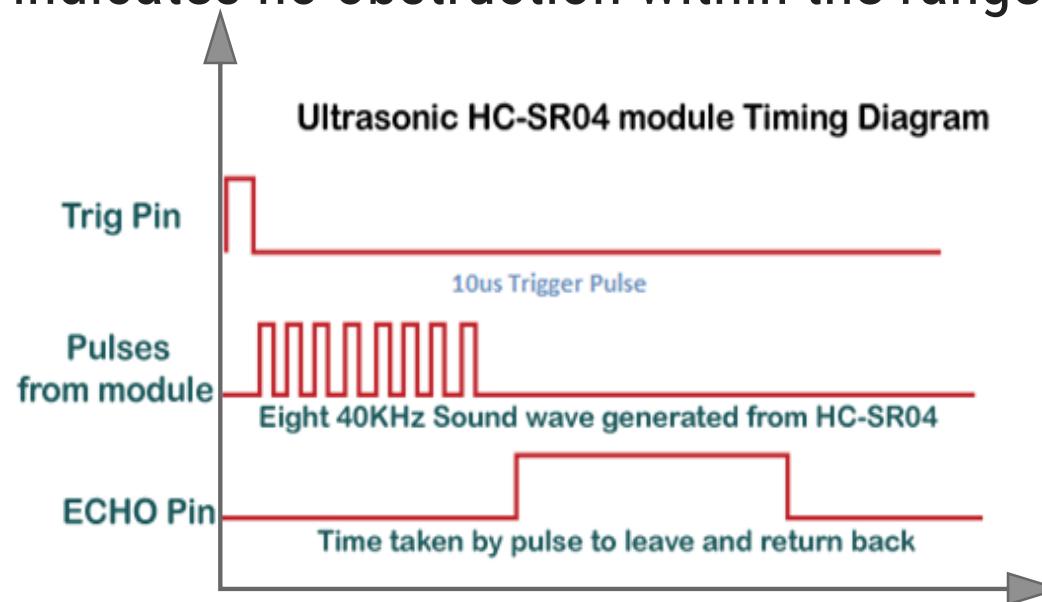
HC-SR04 Ultrasonic Sound Sensor – Transmitter and Receiver

- Ultrasonic Sound Sensor Transmit and Receive Sound wave



HC-SR04 Ultrasonic Sound Sensor – Working principle

- When the **Trigger Pin** is set **HIGH** for $10\mu\text{s}$.
 - ❖ The sensor transmits an ultrasonic burst of **eight pulses** at **40 kHz**.
- These 8 ultrasonic pulses travel through the air away from the transmitter.
- Meanwhile the echo pin goes **HIGH** to initiate the echo-back signal.
- If those pulses are not reflected back, the echo signal times out and goes **low** after **38ms** (38 milliseconds).
- Thus a pulse of 38ms indicates no obstruction within the range of the sensor.



HC-SR04 Ultrasonic Sound Sensor – Working principle

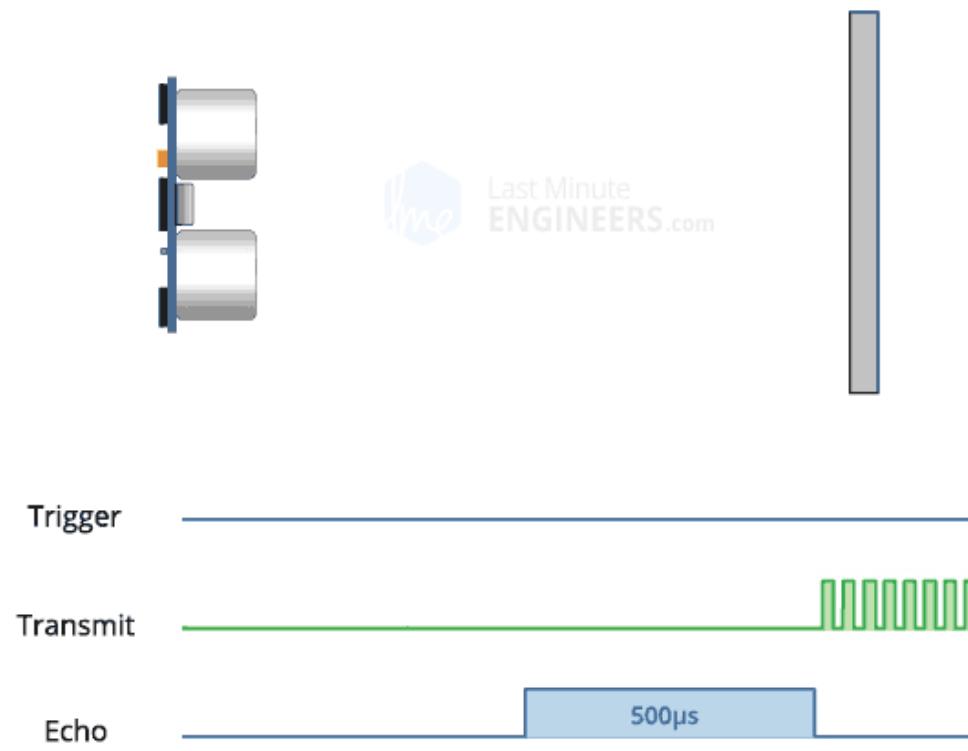


Last Minute
ENGINEERS.com



HC-SR04 Ultrasonic Sound Sensor – Working principle

- If those pulses are reflected back, the echo pin goes **LOW** as soon as the signal is received.
- This generates a pulse on the echo pin whose width varies from **150 µs to 25 ms** depending on the time taken to receive the signal.
- The width of the received pulse is used to calculate the distance from the reflected object.



HC-SR04 Ultrasonic Sound Sensor –Timing Diagram

- Step 1: Make Trigger Pin High for the 10 μs .
 - ❖ (Module Will Generates the Sound Wave)
 - ❖ Module makes Echo Pin **HIGH**.
- Echo Pin goes **LOW**, once it received pulse.
- Step 2: Get the Value of **Time duration**, from which **ECHO** pin **HIGH** duration.
 - ❖ **delayMicroseconds()**: It pauses the program for amount of time in microseconds specified as the parameter.
 - Syntax : `delayMicroseconds(μs)`
 - Parameters : The number of microseconds to pause.
 - ❖ **pulseIn()** : Reads a pulse HIGH or LOW from specified pin and wait for the time to go the pulse from HIGH to LOW or LOW to HIGH. It returns the time required to transit the pulse in variable.
 - Syntax : `pulseIn(pin, value)`
 - Parameters - Pin: The number of the Arduino pin on which you want to read the pulse.
 - Value: The type of pulse that you want to read.

HC-SR04 Ultrasonic Sound Sensor – Code explanation

- The code in Arduino for HC-SR04 Ultrasonic Sound Sensor can be written as following

Distance-measure.ino

```
1 int trigPin = 11;      // Trigger
2 int echoPin = 12;      // Echo
3 long duration, cm, inches;
4
5 void setup() {
6     Serial.begin (9600);
7     pinMode(trigPin, OUTPUT);
8     pinMode(echoPin, INPUT);
9 }
10 void loop() {
11     digitalWrite(trigPin, LOW);
12     delayMicroseconds(5);
13     digitalWrite(trigPin, HIGH);
14     delayMicroseconds(10);
15     digitalWrite(trigPin, LOW);
```

Generates the delay of specified microseconds.

High for 10 μ Sec.

HC-SR04 Ultrasonic Sound Sensor – Code explanation (Cont.)

Distance-measure.ino

```
16  
17     duration = pulseIn(echoPin, HIGH);  
18  
19 // Convert the time into a distance  
20 cm = (duration/2) / 29.1;  
21 inches = (duration/2) / 74;  
22 Serial.print(inches);  
23 Serial.print("in, ");  
24 Serial.print(cm);  
25 Serial.print("cm");  
26 Serial.println();  
27  
28 delay(250);  
29 }
```

Reads a HIGH pulse on specified pin and returns the value of time in microsecond for transition from LOW to HIGH.

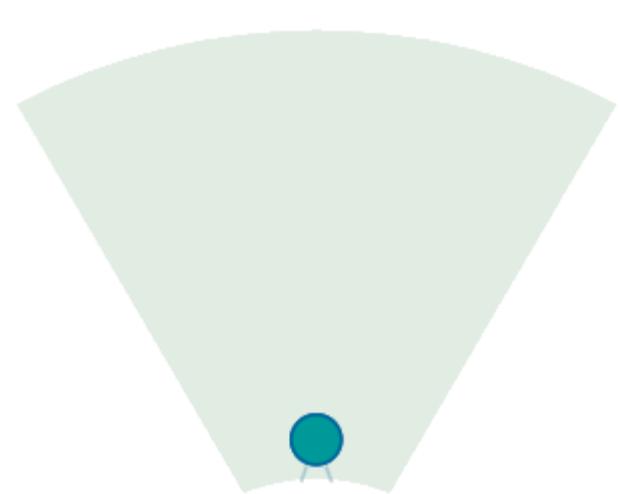
$$\text{Distance(m)} = \text{Speed (m/s)} \times \text{Time (s)}$$

But, Time of pulse we receive is in μs and distance we want to find is in cm.

$$\text{Distance(cm)} = \text{Speed (cm}/\mu\text{s}) \times \text{Time (\mu s)}$$

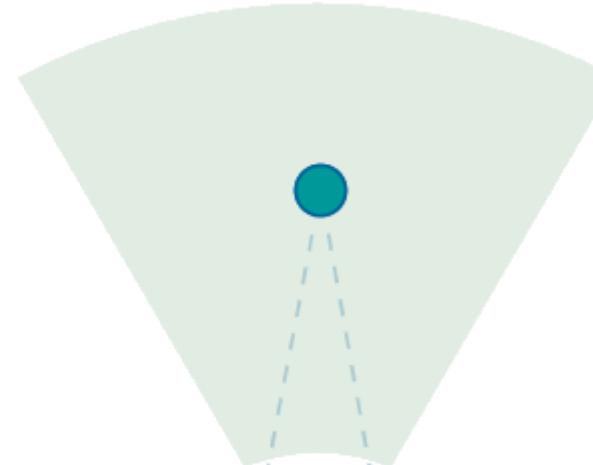
Speed of sound is **343 mtr/sec** which is converted into $0.0343 \text{ cm}/\mu\text{s}$.
So either we have to multiply 0.0343 or divide 29.1

HC-SR04 Ultrasonic Sound Sensor – Output



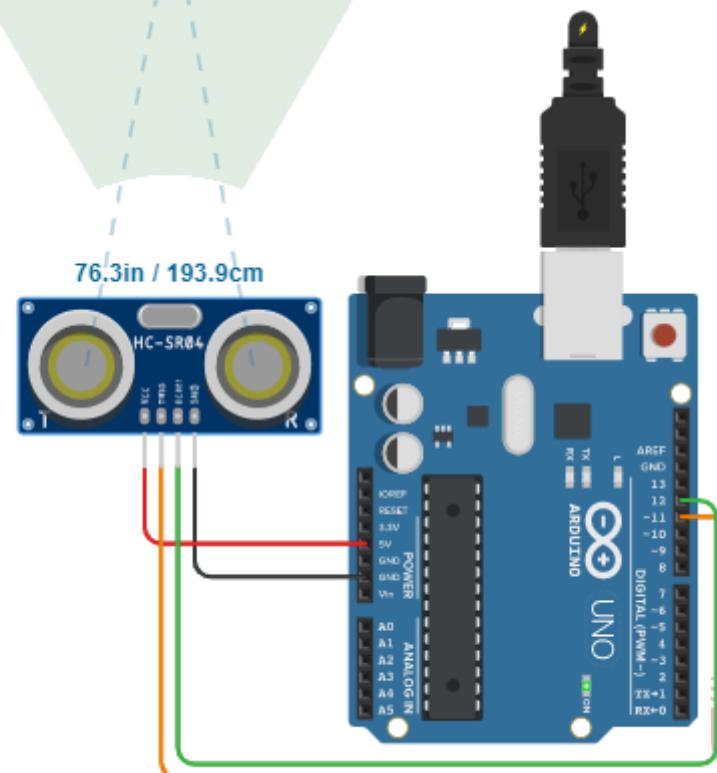
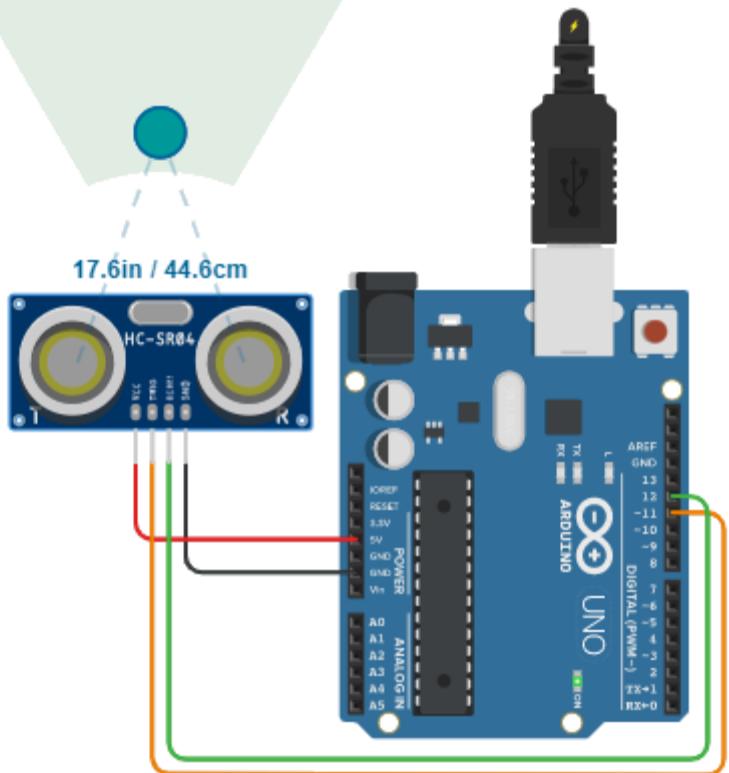
Serial Monitor

17in, 44cm
17in, 44cm
17in, 44cm
17in, 44cm
17in, 44cm



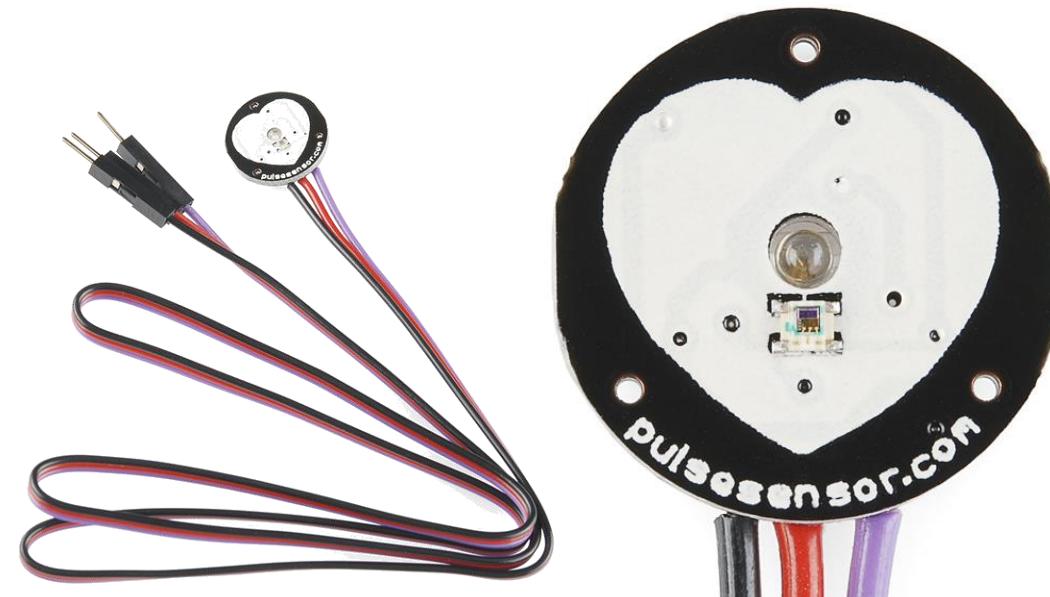
Serial Monitor

76in, 193cm
76in, 193cm
76in, 193cm
76in, 193cm
76in, 193cm



Heartbeat Sensor

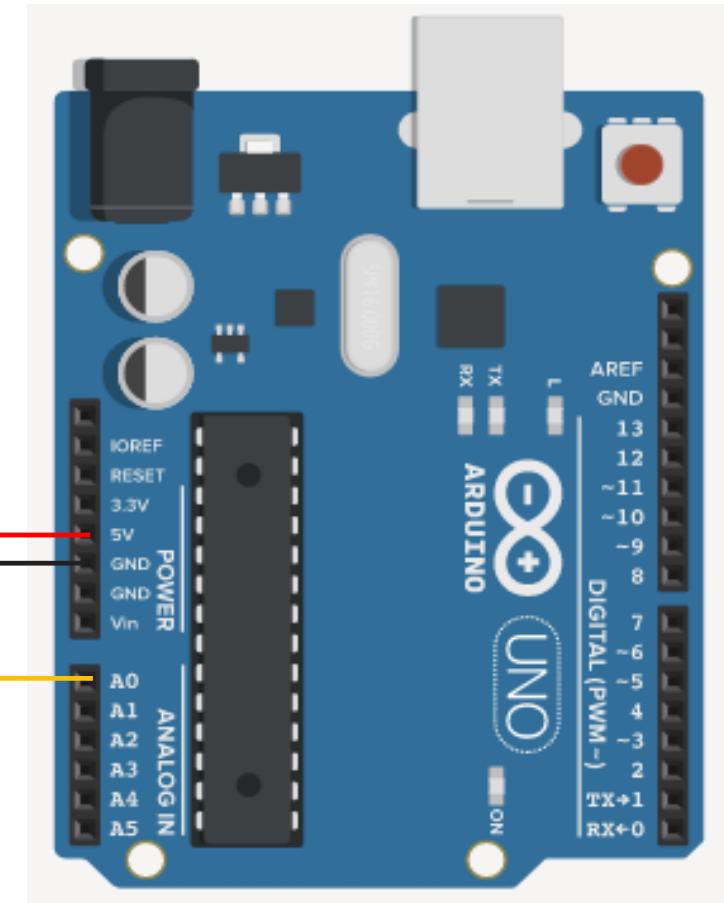
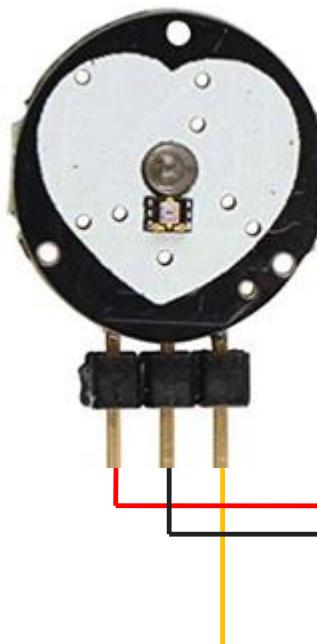
- Heartbeat sensor is having typical application in health care domain.
- This is mainly used in wearable devices to monitor the heart beat or heart rate of the person.
- The sensor is inexpensive and generates square waves for each pulse.
- The averaging of pulses gives analog voltage according to the heartbeat.



- Pinout of heartbeat sensor module are
 - ❖ Vcc – Connected to 5V
 - ❖ Gnd – Connected to ground
 - ❖ A0 – Analog output pin

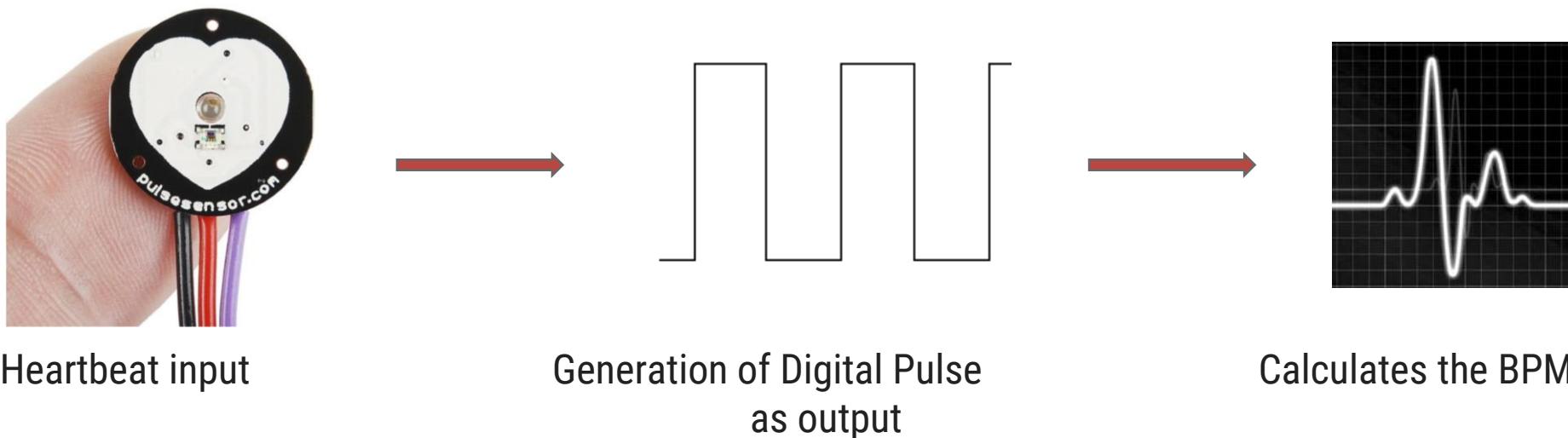
Heartbeat Sensor – Interfacing with Arduino

- The interfacing of Heartbeat sensor with Arduino Uno is as shown in the figure.
- Here, Vcc and Gnd pins of heartbeat sensor are connected to 5V and GND of Arduino board.
- We want to take the input from the sensor in an analog format so the **signal pin** of the sensor is connected to one of the **analog pins of Arduino** i.e. A0.



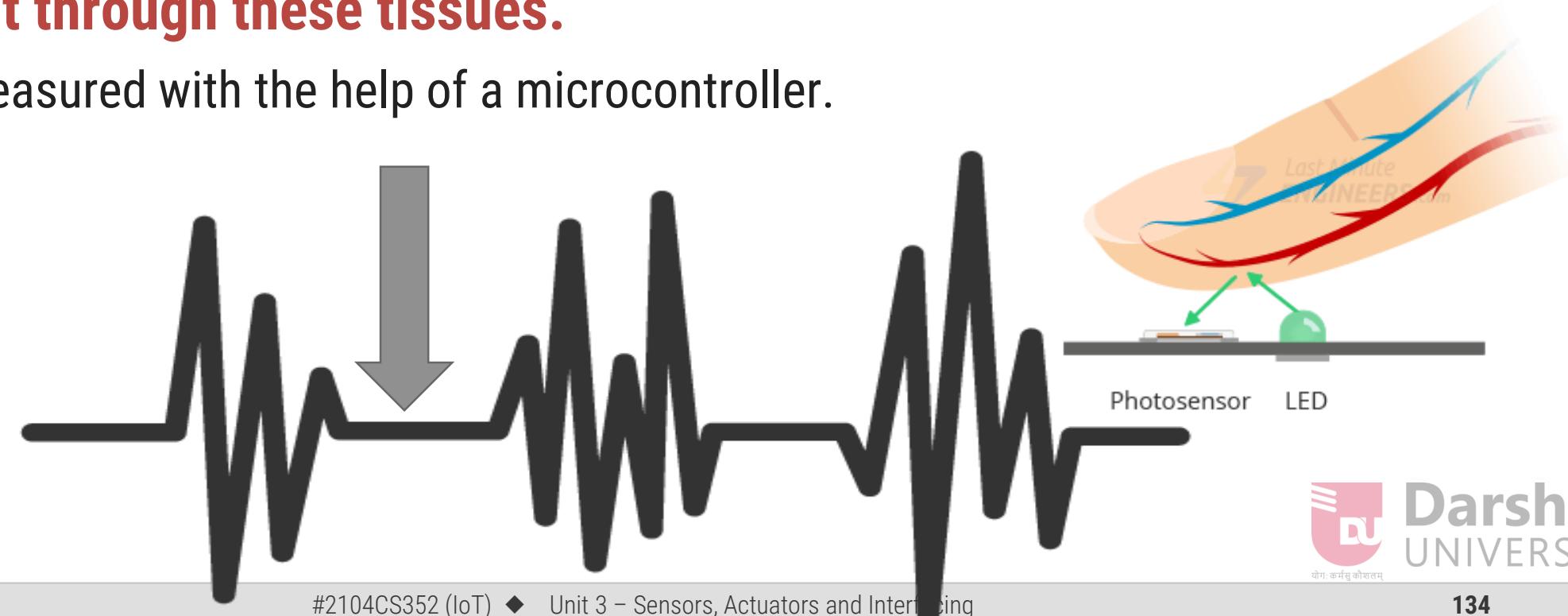
Heartbeat Sensor – Working principle

- The sensor senses the heartbeat as per blood circulation in the body.
- It generates the output pulse signal according to the rate at which heart beats.
- The output pulses are considered as PWM waves. Hence they are converted into analog signals.
- This **analog signal** is converted **to digital** which gives the output beat rate.



Heartbeat Sensor – Working principle

- When a heartbeat occurs, blood is pumped through the human body and gets squeezed into the capillary tissues.
- Consequently, the **volume** of these capillary tissues **increases**.
- But in between the two consecutive heartbeats, this volume inside capillary tissues decreases.
- **This change in volume between the heartbeats affects the amount of light that will transmit through these tissues.**
- This can be measured with the help of a microcontroller.



Heartbeat Sensor – Working principle

- The pulse sensor module has a light that helps in measuring the pulse rate.
- **When we place the finger on the pulse sensor, the light reflected will change based on the volume of blood inside the capillary blood vessels.**
- This variation in light transmission and reflection can be obtained as a pulse from the output of the pulse sensor.
- This pulse can be then conditioned to measure heartbeat and then programmed accordingly to read as heartbeat count using Arduino.

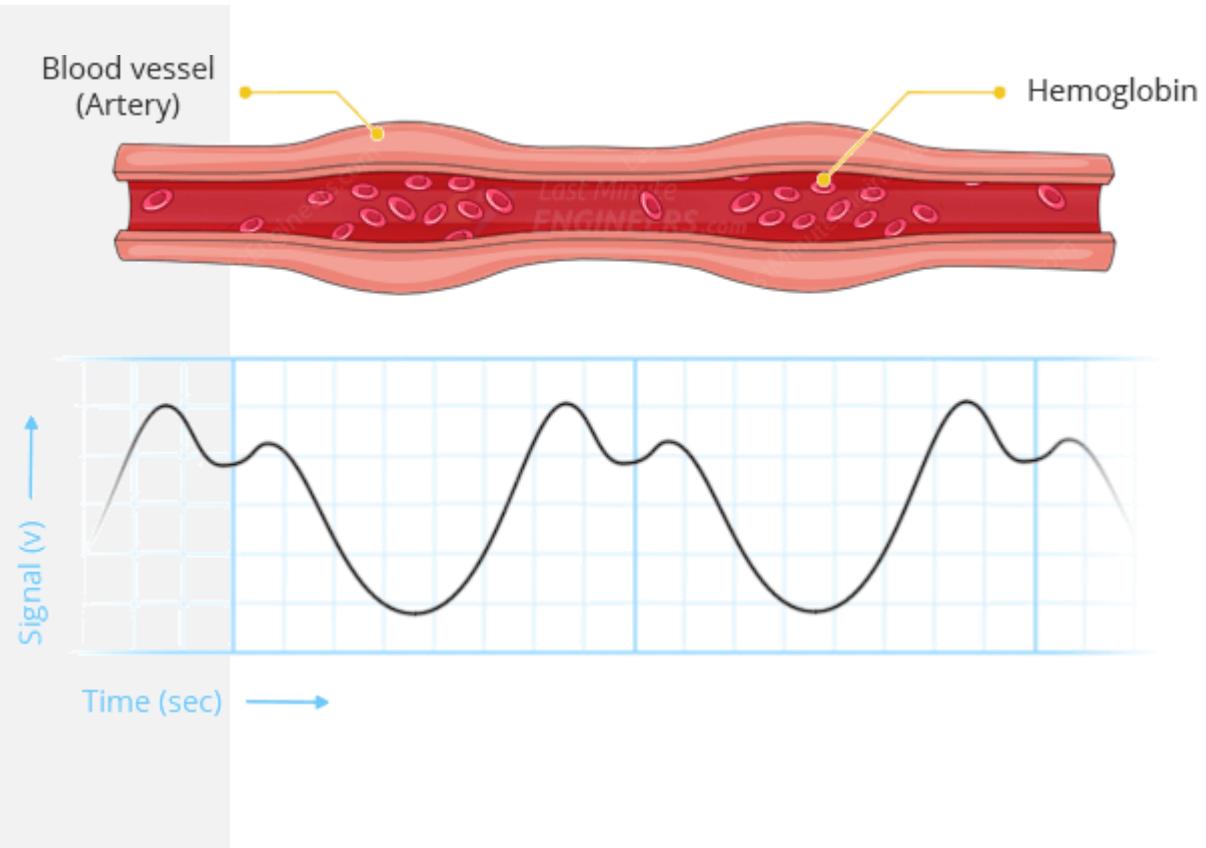
Heartbeat Sensor – Code explanation

- ▶ The code in Arduino for Heartbeat sensor can be written as following:

Heartbeat_sensor.ino

```
1 int UpperThreshold = 518;
2 int LowerThreshold = 490;
3 int reading = 0;
4 float BPM = 0.0;
5 bool IgnoreReading = false;
6 bool FirstPulseDetected = false;
7 unsigned long FirstPulseTime = 0;
8 unsigned long SecondPulseTime = 0;
9 unsigned long PulseInterval = 0;

10
11 void setup(){
12     Serial.begin(9600);
13 }
```



Heartbeat Sensor – Code explanation (Cont.)

Heartbeat_sensor.ino

```
14 void loop(){
15     reading = analogRead(A0);
16     if(reading > UpperThreshold && IgnoreReading == false)
17     {
18         if(FirstPulseDetected == false)
19         {
20             FirstPulseTime = millis();
21             FirstPulseDetected = true;
22         }
23     else
24     {
25         SecondPulseTime = millis();
26         PulseInterval = SecondPulseTime -FirstPulseTime;
27         FirstPulseTime = SecondPulseTime;
28         FirstPulseDetected = false;
29     }
30     IgnoreReading = true;
31 } //Heart beat leading edge detection
```

millis() function gives the time in millisecond from where the Arduino board powered or reset.

Heartbeat Sensor – Code explanation (Cont.)

Heartbeat_sensor.ino

```
31 if(reading < LowerThreshold)
32 {
33     IgnoreReading = false;
34     //Hear beat trailing edge detection
35     BPM = (1.0/PulseInterval) * 60.0 * 1000;
36     Serial.print(BPM);
37     Serial.println(" BPM");
38     Serial.flush();
39 }
40 }
```

Pulse-Interval is in milliseconds.
∴ Rate = 1/Time

$$\therefore \text{Rate} = \frac{1}{\text{PulseInterval(ms)}}$$

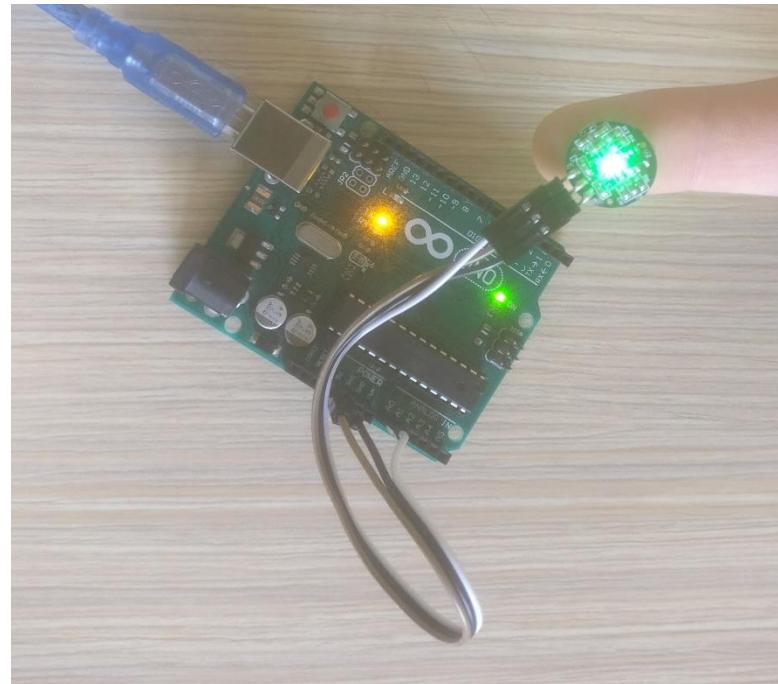
$$\therefore \text{Rate} = \frac{1}{\text{PulseInterval (s)} / 1000}$$

$$\therefore \text{Rate} = \frac{1}{\text{PulseInterval(s)}} * 1000$$

$$\therefore \text{Rate in min} = \frac{1}{\text{PulseInterval}} * 1000 * 60$$

- millis() : Returns the number of milliseconds passed since the Arduino started running the current program.
 - ❖ Syntax : millis ()

Heartbeat Sensor – Output



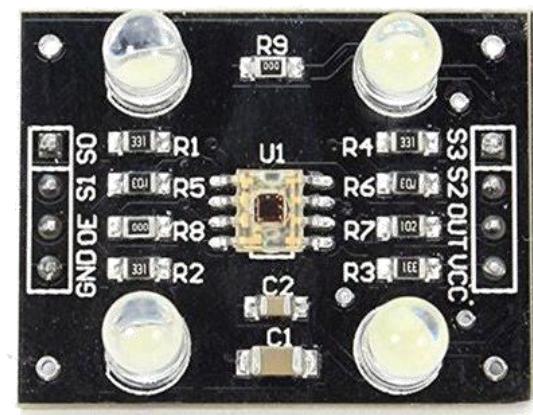
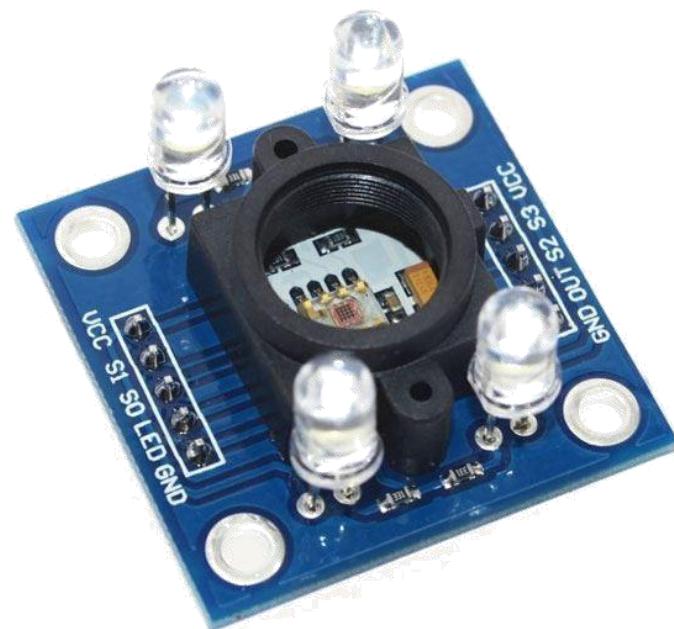
```
83.57 BPM
82.42 BPM
82.42
```

Autoscroll Show timestamp

Newline 9600 baud Clear output

Colour Sensor

- ▶ Colour sensor is used to detect the RGB colour coordinates of a particular colour.
- ▶ The colour sensor module has TSC3200 IC that converts colour to frequency by enabling a particular colour diode turn by turn.
- ▶ The colour sensor is mainly used in the application where the objects are identified by their colour.
- ▶ Pin-out of colour sensor module are
 - Vcc – Connected to 5V
 - Gnd – Connected to ground
 - S0, S1 – Used for output frequency scaling
 - S2, S3 – Type of photodiode selected
 - (OE)' – Enable for output
 - OUT – Output frequency (f_o)



Colour Sensor (Cont.)

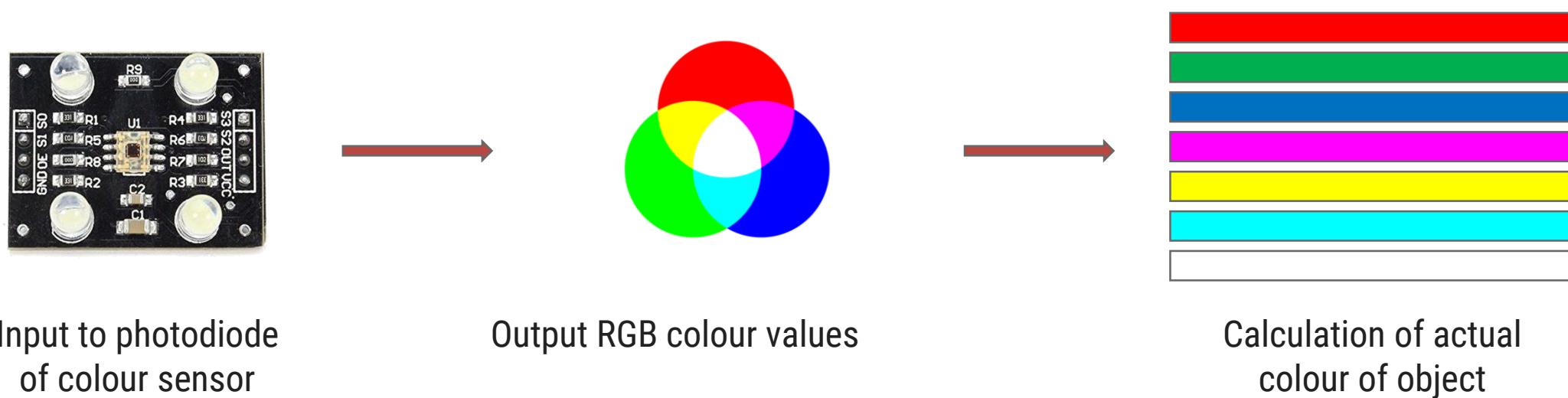
- ▶ The combination of S0 and S1 are used for output frequency scaling.
- ▶ The different microcontrollers have different counter functionality and limitations. Hence, we need to scale the frequency according to microcontroller used.
- ▶ The percentage of output scaling for different combinations of S0 and S1 are shown in the table.
- ▶ The S3 and S4 pins are used for photodiode selection. The photodiodes are associated with different colour filters.
- ▶ The table shows combinations of S3 and S4 for different photodiodes.

| S0 | S1 | Output Frequency Scaling (f_0) | Typical full-scale Frequency |
|----|----|------------------------------------|------------------------------|
| L | L | Power Down | ----- |
| L | H | 2% | 10 – 12 KHz |
| H | L | 20% | 100 – 120 KHz |
| H | H | 100% | 500 – 600 KHz |

| S2 | S3 | Photo Diode Type |
|----|----|------------------|
| L | L | Red |
| L | H | Blue |
| H | L | Clear(No Filter) |
| H | H | Green |

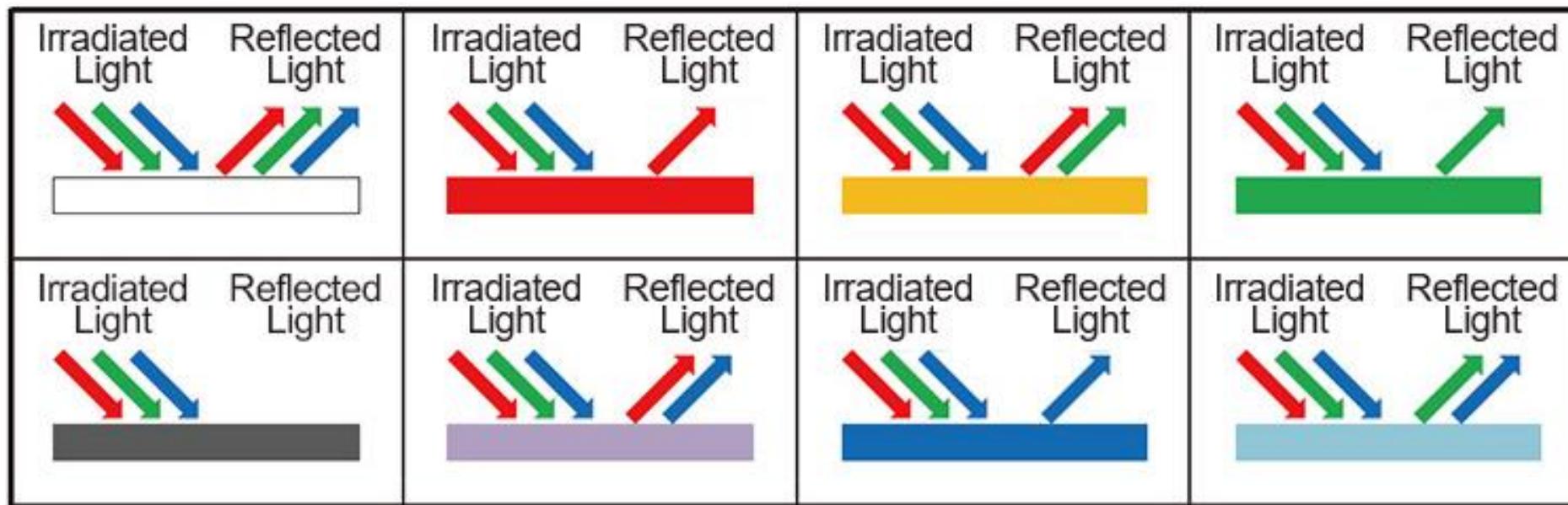
Colour Sensor – Working principle

- ▶ The sensor works by imparting a bright light on an object and recording the reflected colour by the object.
- ▶ The selected photodiode for colour of red, green and blue converts the amount of light to current.
- ▶ These RGB values are further processed to identify the exact colour combination



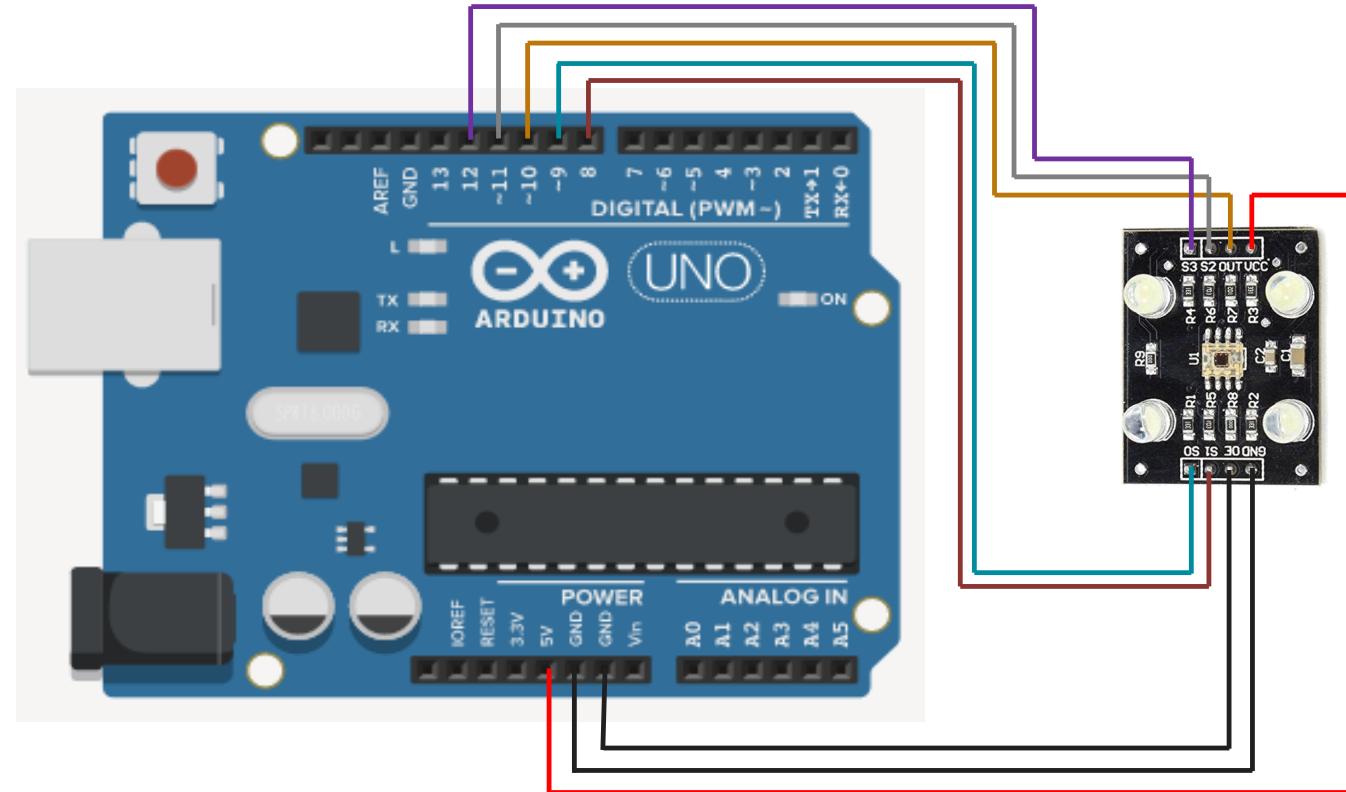
Colour Sensor – Working Principle

- When an object is irradiated with light containing RGB components, the colour of the reflected light will change depending on the colour of the object.
- For example, if the object is red, the reflected light component will be red. For a yellow object, the reflected light will be red and green, and if the object is white all three components will be reflected.



Colour Sensor – Interfacing with Arduino

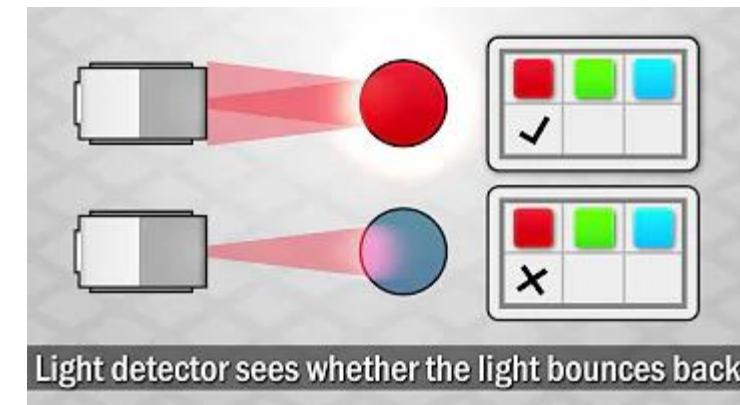
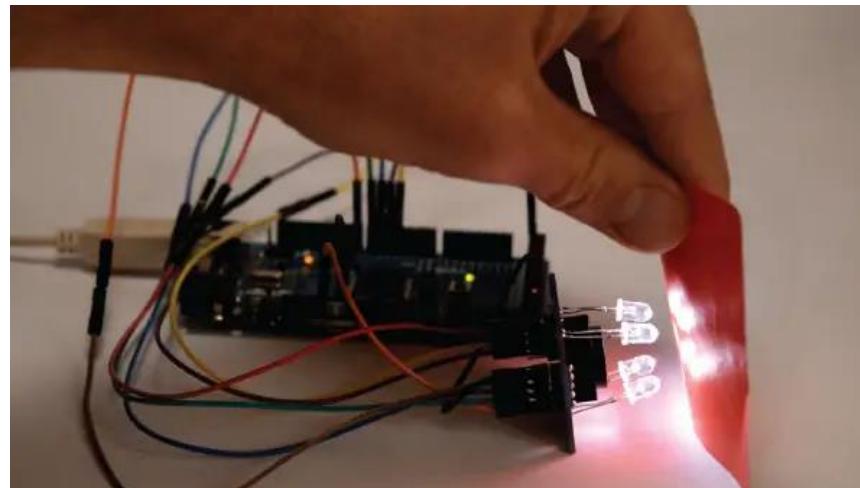
- ▶ The interfacing of Colour Sensor with Arduino Uno is as shown in figure.
- ▶ Here, Vcc and Gnd pins of Colour Sensor are connected to 5V and Gnd of Arduino board.
- ▶ As we need to enable the sensor OE' is connected to GND.
- ▶ All selection pins S0, S1, S2 and S3 are connected to digital Pins of Arduino that is 9,8,11, and 12 respectively.
- ▶ The output of colour sensor are taken as a pulse count. So OUT pin is also connected to digital pin. In our case it is connected to pin 10.
- ▶ **The lower the value given by the sensor the higher is that colour.**



Colour Sensor – Colour Range

By Experiment only we can get exact calibrated range

| | |
|-------|-------|
| RED | 10-56 |
| GREEN | 30-90 |
| BLUE | 25-70 |



Colour Sensor – Code explanation

- The code in Arduino for Colour sensor can be written as following:

Colour_sensor.ino

```
1 #define s0 9
2 #define s1 8
3 #define s2 11
4 #define s3 12
5 #define out 10
6
7 int frequency=0;
8
9 void setup()
10 {
11     pinMode(s0,OUTPUT);
12     pinMode(s1,OUTPUT);
13     pinMode(s2,OUTPUT);
14     pinMode(s3,OUTPUT);
15     pinMode(out,INPUT);
```

Colour Sensor – Code explanation (Cont.)

Colour_sensor.ino

```
16     Serial.begin(9600);
17     digitalWrite(s0,HIGH);
18     digitalWrite(s1,HIGH);
19 }
20 void loop()
21 {
22     digitalWrite(s2,HIGH);
23     digitalWrite(s3,HIGH);
24     frequency = pulseIn(out,LOW);
25     frequency = map(frequency, 30,90,255,0);→
26     Serial.print("G = ");
27     Serial.print(frequency);
28     Serial.print("\t");
29     delay(100);
30 }
```

This syntax maps the value of given number or variable from its defined range to desired range.

Colour Sensor – Code explanation (Cont.)

Colour_sensor.ino

```
31  digitalWrite(s2,LOW);
32  digitalWrite(s3,HIGH);
33  frequency = pulseIn(out,LOW);
34  frequency = map(frequency,25,70,255,0);
35  Serial.print("B = ");
36  Serial.print(frequency);
37  Serial.print("\t");
38  delay(100);
39  digitalWrite(s2,LOW);
40  digitalWrite(s3,HIGH);
41  frequency = pulseIn(out,LOW);
42  frequency = map(frequency,10,56,255,0);
43  Serial.print("R = ");
44  Serial.print(frequency);
45  Serial.print("\t");
46  delay(100);
47 }
```

GPS Sensor

- ▶ GPS sensor is used to get the location data of the place where sensor is situated.
- ▶ The data from sensor is acquired in National Marine Electronics Association (NEMA) format. The format is used by marine department for communication.
- ▶ There are multiple variants of GPS sensors available in the market. We can select based on our requirements.
- ▶ Here, the given sensor is GY-GPS6MV2 as shown in the image.
- ▶ Pin-out of GPS sensor module are
 - Vcc – Connected to 5V
 - Gnd – Connected to ground
 - Tx – Serial Data Transmit
 - Rx – Serial Data Receive



GPS Sensor – Working principle

- ▶ The sensor is placed in the system whose location is to be tracked.
- ▶ The GPS device communicates with GPS satellite and the satellite returns its current location in form of latitude and longitude.
- ▶ The sensor transmits the data to the microcontroller in a predefined format from which we can extract the tracked location.



GPS device with antenna
for communication

Latitude
Longitude

Returns latitude and longitude
as output



Tracked Location

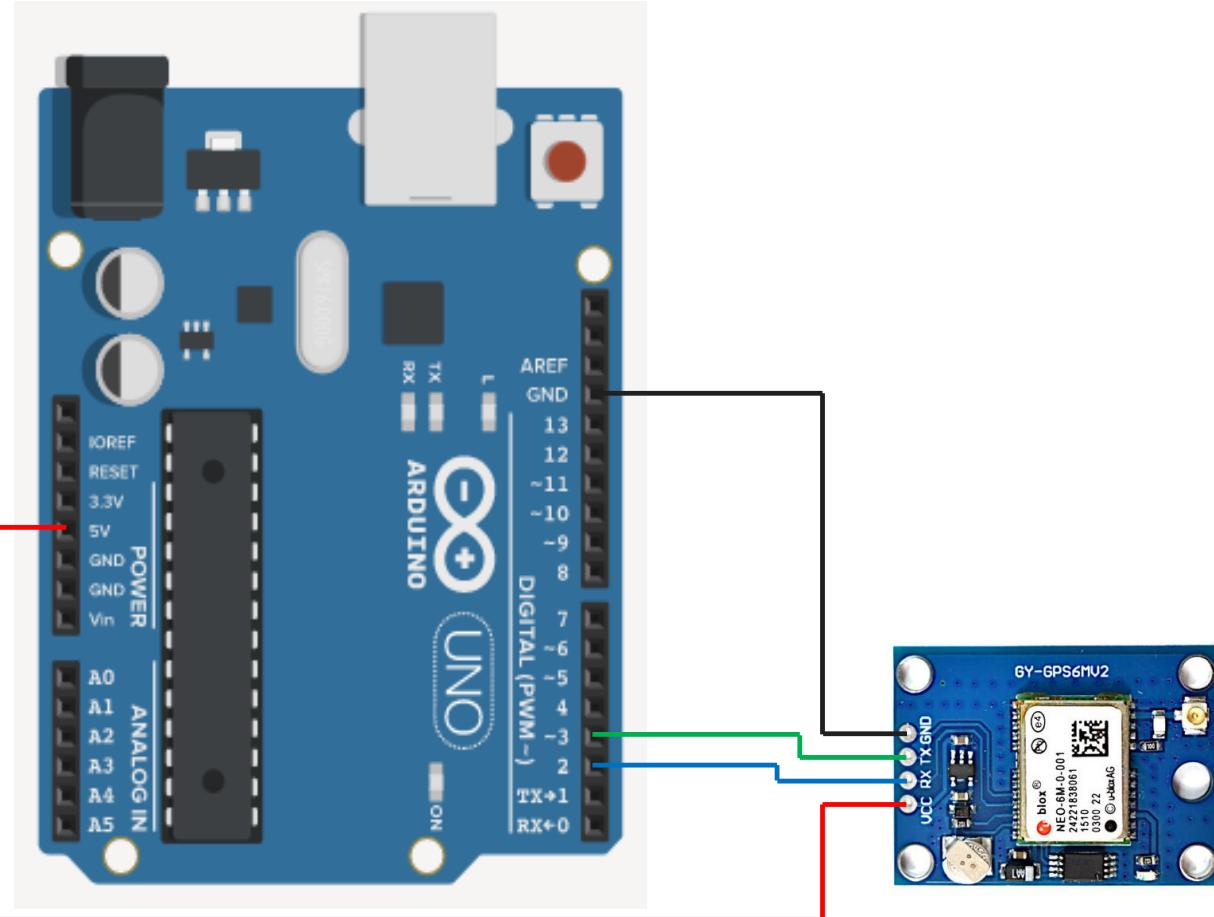


Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

GPS Sensor – Interfacing with Arduino

- ▶ The interfacing of GPS Sensor with Arduino Uno is as shown in figure.
- ▶ Here, Vcc and Gnd pins of GPS Sensor are connected to 5V and Gnd of Arduino board.
- ▶ As the GPS communicates with Arduino serially, we need to connect Tx and Rx pin of GPS with serial pins of Arduino.
- ▶ But, we want the data received from the GPS sensor in serial monitor also to read the current location.
- ▶ Therefore, we need to use any digital pins of Arduino board as Serial transmit and receive pins which is done by software serial.
- ▶ Here, the Tx and Rx pins of GPS are connected with pins 3 and 2 respectively.



GPS Sensor – Code explanation

- The code in Arduino for GPS sensor can be written as following:

GPS_sensor.ino

```
1 #include <TinyGPS++.h>
2 #include <SoftwareSerial.h>
3 static const int RXPin = 3, TXPin = 2;
4 static const uint32_t GPSBaud = 9600;
5 TinyGPSPlus gps;
6
7 SoftwareSerial ss(RXPin, TXPin);
8
9 void setup()
10 {
11   Serial.begin(115200);
12   ss.begin(GPSBaud);
13 }
```

Includes the TinyGPS++ library to use built in functions for reading GPS sensor.

Includes the **Software-Serial** library to use software serial communication

Defining *gps* as object of TinyGPSPlus

Defining *ss* as object of SoftwareSerial



Darshan
UNIVERSITY

गोपा कर्मसु कौशलम्

GPS Sensor – Code explanation (Cont.)

GPS_sensor.ino

```
14 void loop()
15 {
16     while (ss.available() > 0)
17         gps.encode(ss.read());           →
18     displayInfo();
19
20     if (millis()>5000 &&
21         gps.charsProcessed()<10)
22     {
23         Serial.println("No GPS detected: Check wiring.");
24         while(true);
25     }
26 }
27 }
```

The syntax reads the data serially from GPS and *gps.encode()* function encodes that data as per the format.

GPS Sensor – Code explanation (Cont.)

GPS_sensor.ino

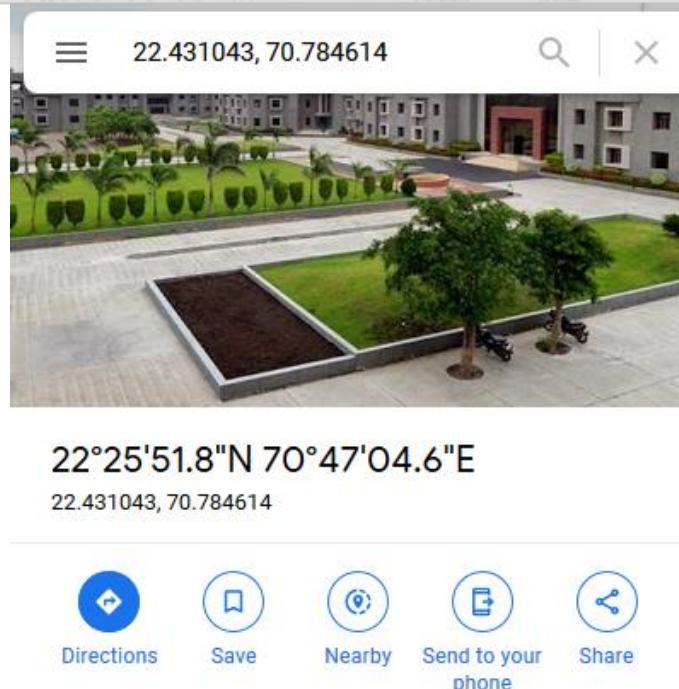
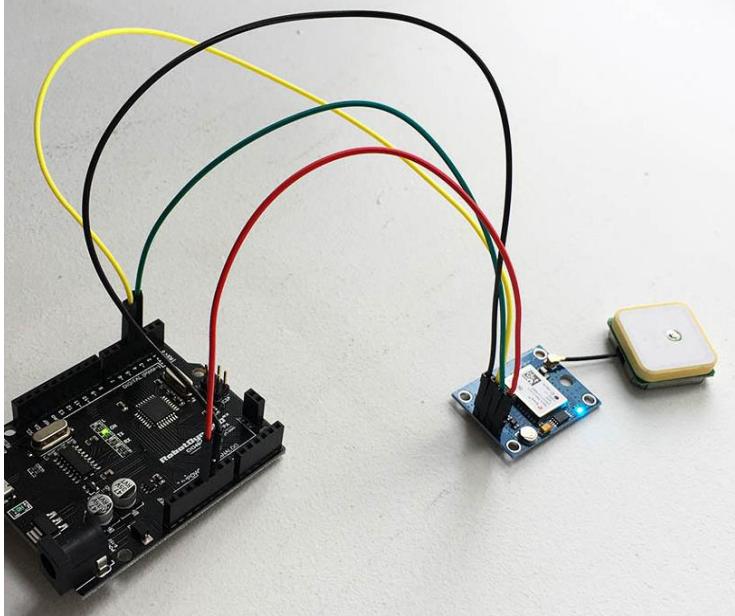
```
28 void displayInfo() {  
29     Serial.print("Location: ");  
30     if (gps.location.isValid())  
31     {  
32         Serial.print(gps.location.lat(), 6);  
33         Serial.print(",");  
34         Serial.print(gps.location.lng(), 6);  
35     }  
36     Serial.println();  
37 }
```

gps.location.isValid() function returns “TRUE” if the location is in valid form

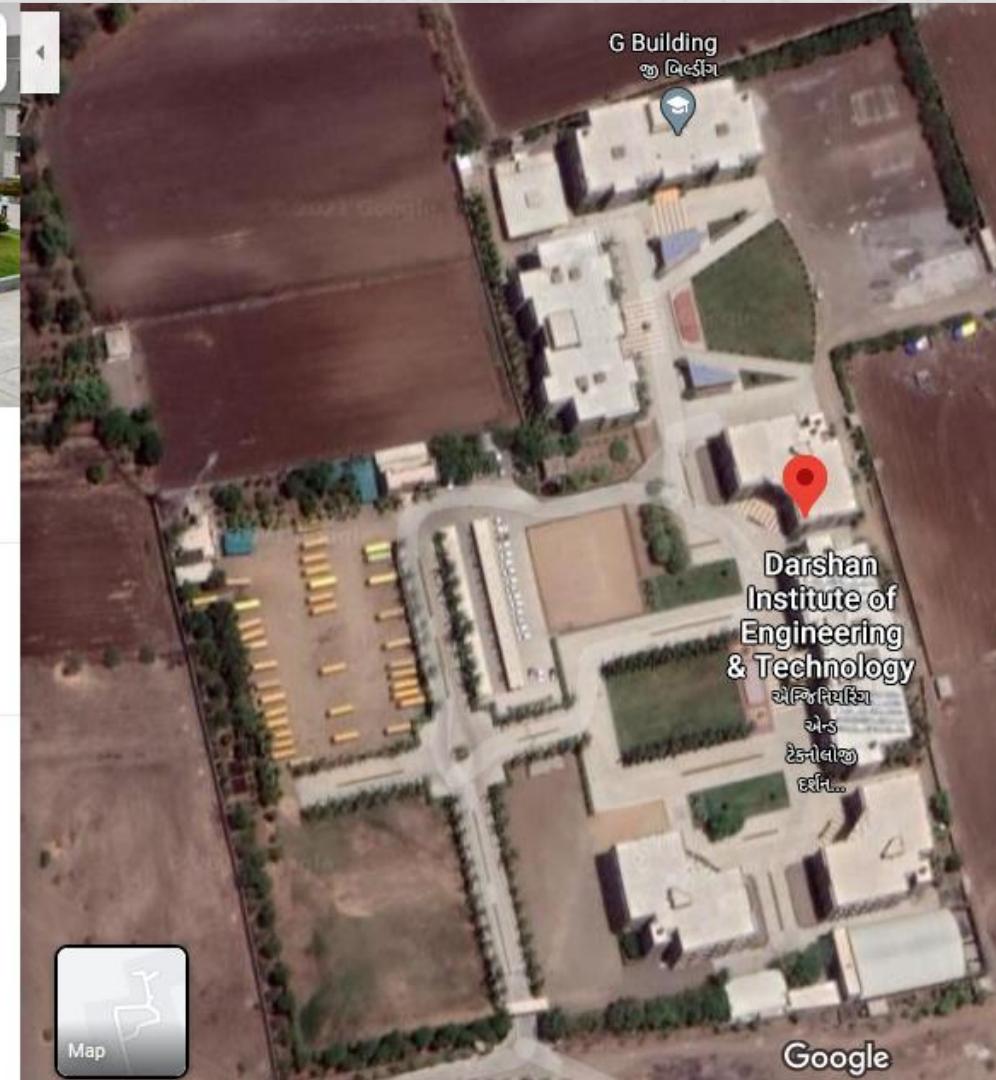
gps.location.lat() function returns the current latitude value and it is printed with 6 decimal point precision.

gps.location.lng() function returns the current longitude value and it is printed with 6 decimal point precision.

GPS Sensor – Output



```
COM4  
|  
Location: 22.431043,70.784614  
Location: 22.431043,70.784614  
Location: 22.431043,70.784614  
Location: 22.431043,70.784614  
Location: 22.431043,70.784614  
Location: 22.431043,70.784614  
Location: 22.431043,70.784614
```



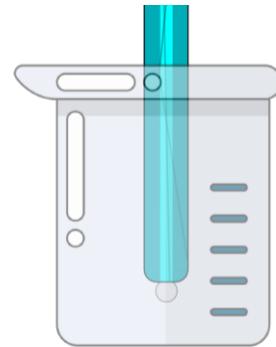
pH Sensor

- ▶ The pH sensor is used to detect hydrogen ions concentration of a liquid.
- ▶ pH sensor are mostly used in laboratories to test the acidity of solution.
- ▶ By the use of pH sensor we can identify the pH of a solution and whether it is **acid or base**.



pH Sensor – Working principle

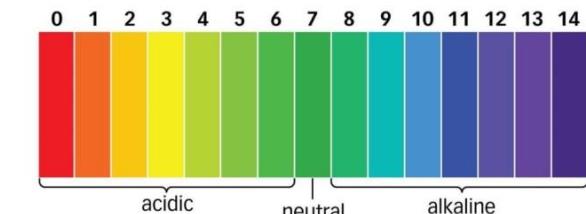
- When this sensor is placed in a solution, the smaller ions penetrate the boundary area of glass and the larger ions remain in the solution. This creates potential difference.
- The pH meter measures the difference in electrical potential between the pH electrodes.
- The potential difference generates different analog values for different liquids.
- By knowing the analog value of standard water, the pH value of other liquid can be determined.



pH Sensor with
Electrodes



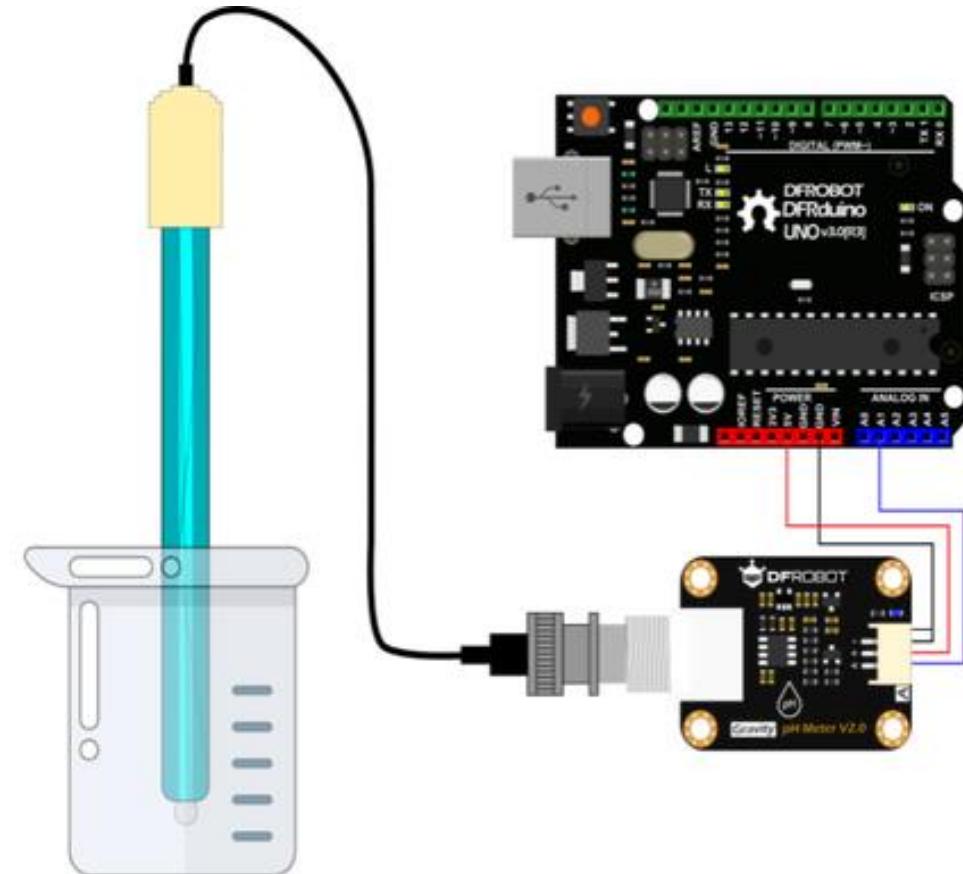
Output Voltage



Determination of alkalinity
of the liquid

pH Sensor – Interfacing with Arduino

- ▶ The interfacing of pH sensor with Arduino is as shown in figure.
- ▶ Here, Vcc and Gnd pins of pH Sensor are connected to 5V and Gnd of Arduino board.
- ▶ The output pin of pH sensor is connected to analog pin of Arduino.
- ▶ The connection of pH sensor is done with voltage converter via BNC pin.



pH Sensor – Code explanation

- The code in Arduino for pH sensor can be written as following:

PH_sensor.ino

```
1 #define SensorPin A0
2 #define Offset 0
3 unsigned long int avgValue;
4 float b;
5 int buf[10],temp;
6
7 void setup()
8 {
9     pinMode(SensorPin,INPUT);
10    Serial.begin(9600);
11 }
```

Defining *offset* for correction of pH value. The offset value can be obtained by measuring the pH of standard water.

Defining *buf* as an integer array.

pH Sensor – Code explanation (Cont.)

PH_sensor.ino

```
12 void loop()
13 {
14     for(int i=0;i<10;i++) {
15         buf[i]=analogRead(SensorPin);
16         delay(10);
17     }
18     for(int i=0;i<9;i++) {
19         for(int j=i+1;j<10;j++) {
20             if(buf[i]>buf[j])
21             {
22                 temp=buf[i];
23                 buf[i]=buf[j];
24                 buf[j]=temp;
25             }
26         }
27     }
}
```

Taking 10 different values of analog voltage from the pin where sensor is connected

Sorting the elements of array *buf* in ascending order.



Darshan
UNIVERSITY

गोपा: कर्मसु कीशतम्

pH Sensor – Code explanation (Cont.)

PH_sensor.ino

```
28 avgValue=0;  
29 for(int i=2;i<8;i++)  
30     avgValue+=buf[i];  
31 avgValue = avgValue/6;  
32 float phValue=(float)((avgValue)*  
33                                5.0/1024);  
34 phValue=3.5*phValue+Offset;  
35 Serial.print("pH: ");  
36 Serial.print(phValue,2);  
37 Serial.println(" ");  
38 delay(800);  
39 }
```

Avoiding lower and upper two (total four) values of array and taking the average of elements of rest of the array.

Converting the average value into equivalent millivolts.

Converting millivolts to its equivalent pH values as given in [datasheet](#) and adjusting with offset.

Sensor summary

- **1. Temperature Sensor:** This tool helps us figure out how hot or cold something is.
- **2. Proximity Sensor:** It's like a detective's radar, detecting if something is nearby without touching it.
- **3. Accelerometer:** This tool feels movements, like a detective sensing if someone's jumping, running, or sitting still.
- **4. IR Sensor (Infrared Sensor):** This one sees things that are too dark or far away for our eyes to spot.
- **5. Pressure Sensor:** It's like a detective's hand, feeling how hard something is pushing or squeezing.
- **6. Light Sensor:** This tool notices how bright or dark a place is, almost like a detective's eyes.

Sensor summary

- **7. Ultrasonic Sensor:** It sends out sound waves to measure distances like a detective shouting to see how far away something is.
- **8. Smoke, Gas, and Alcohol Sensor:** These are like detectives for smells—they detect if something's in the air that shouldn't be.
- **9. Touch Sensor:** This one feels when something is touched or pressed, just like a detective feeling for clues.
- **10. Color Sensor:** It notices the colors of things, just like a detective studying the color of clothes or objects.
- **11. Humidity Sensor:** It's like a detective for moisture, figuring out how damp or dry the air is.
- **12. Position Sensor:** This tool knows if something has moved or changed position, like a detective marking where things are.

Sensor summary

- **13. Magnetic Sensor (Hall Effect Sensor):** It detects invisible magnetic fields, like a detective using a special tool to find hidden clues.
- **14. Microphone (Sound Sensor):** This one hears noises and sounds, like a detective listening for clues or conversations.
- **15. Tilt Sensor:** It feels when something tilts or moves in a particular direction, just like a detective sensing if something is leaning.
- **16. Flow and Level Sensor:** These tools measure how much of something is moving or how much there is of it like a detective gauging the flow of water in a river.
- **17. PIR Sensor:** It detects heat or motion in a room, like a detective sensing if someone's inside without even seeing them.
- **18. Strain and Weight Sensor:** These sense if something is being stretched, bent, or how heavy something is, just like a detective weighing evidence.

**Thank
You**



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot

✉ bhushan.joshi@darshan.ac.in
☎ +91 8485979997



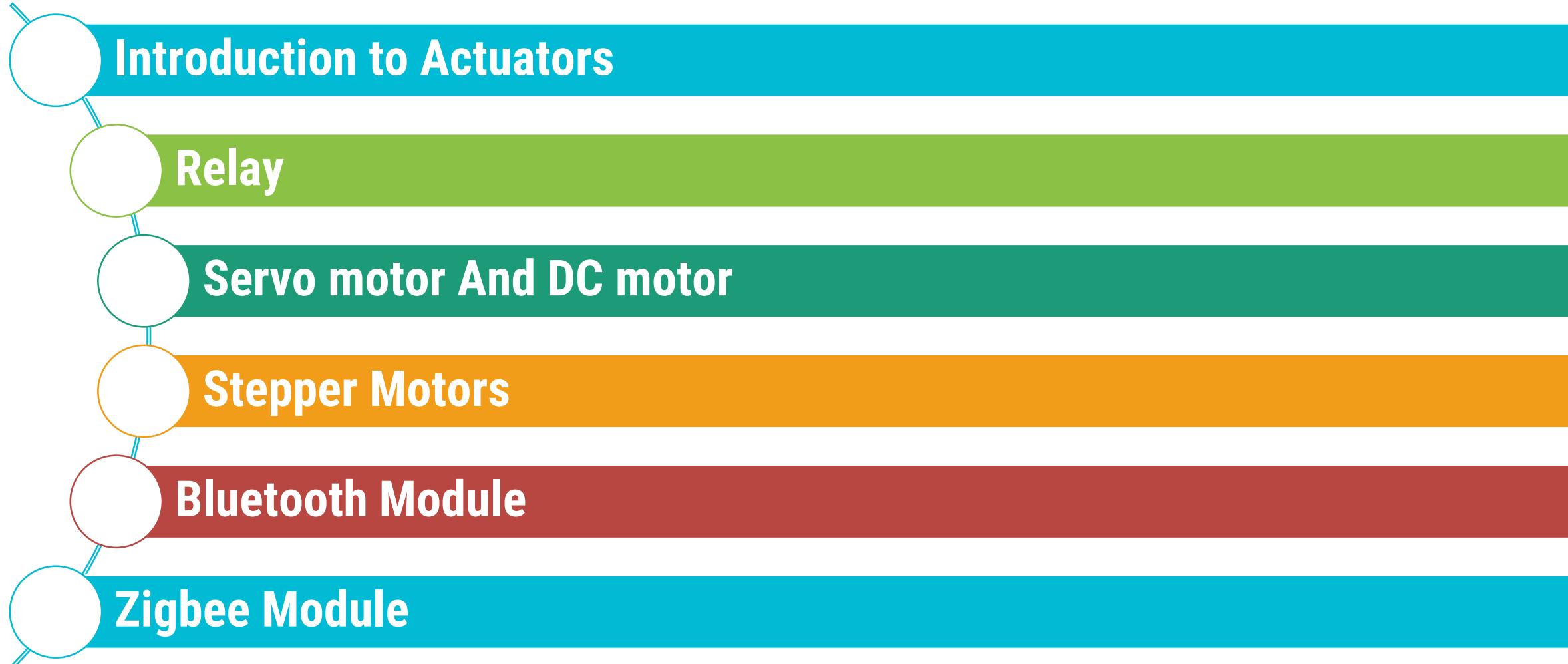
Unit-4

Controlling Hardware



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot
✉ bhushan.joshi@darshan.ac.in
📞 +91 8485979997

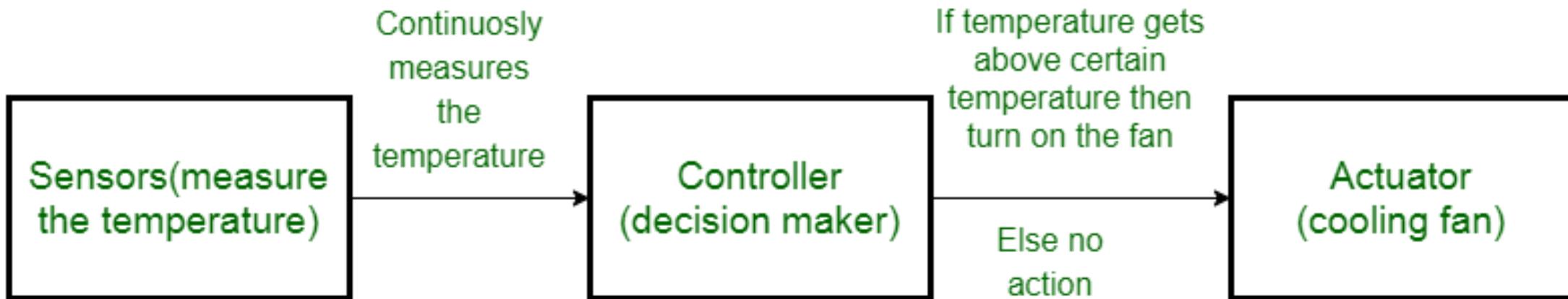




- An IoT device is made up of a Physical object (“thing”) + Controller (“brain”) + Sensors + Actuators + Networks (Internet).
 - An actuator is a machine component or system that moves or controls the mechanism or the system.
 - Sensors in the device sense the environment, then control signals are generated for the **actuators according to the actions needed to perform**.
 - Actuators are the output devices like LEDs, motors, lights, and so on, which let your device do something to the outside world.
 - A servo motor is an example of an actuator.

Working of Actuators

- The following diagram shows what actuators do, the controller directs the actuator based on the sensor data to do the work.

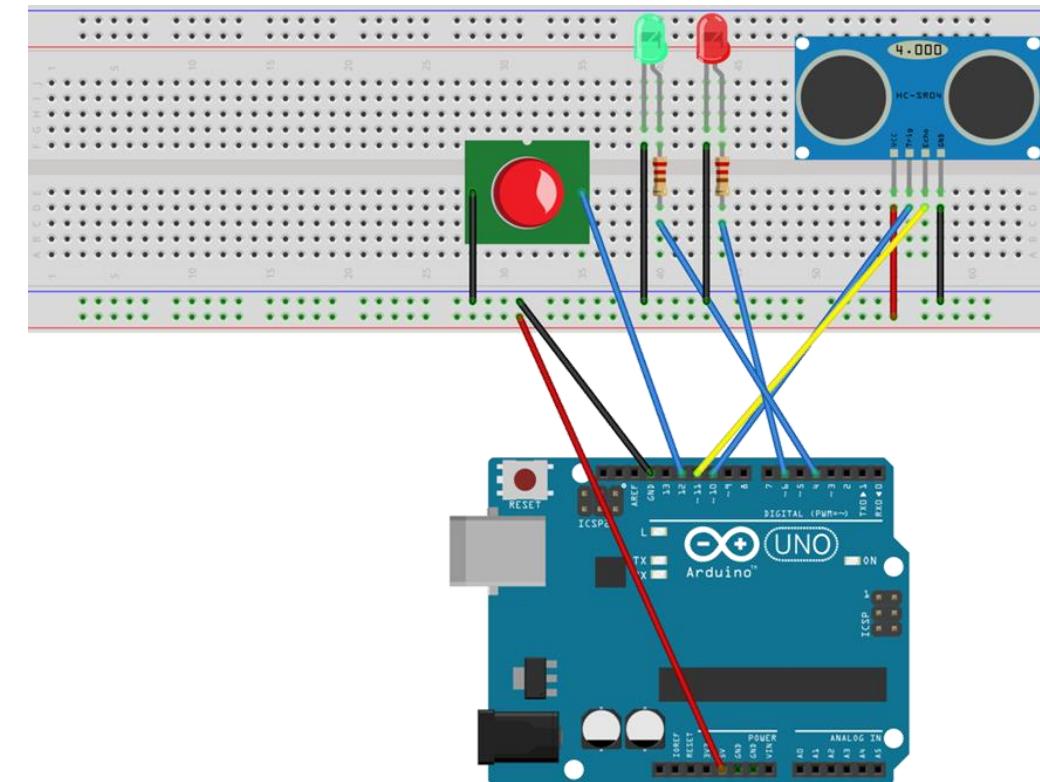


Types of Actuators :

- **1. Hydraulic Actuators** : A hydraulic actuator uses hydraulic power to perform a mechanical operation.
- **2. Pneumatic Actuators** : A pneumatic actuator uses energy formed by vacuum or compressed air at high pressure to convert into either linear or rotary motion
- **3. Electrical Actuators** : An electric actuator uses electrical energy.

Connect LED and Buzzer with Arduino

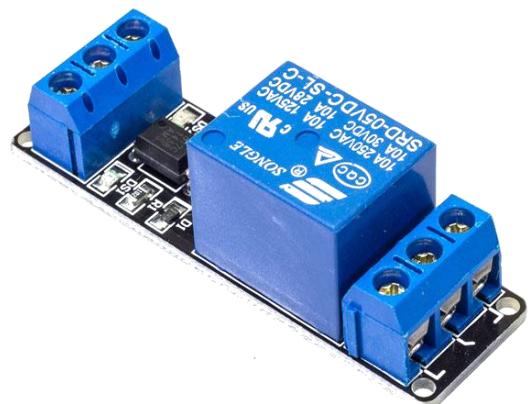
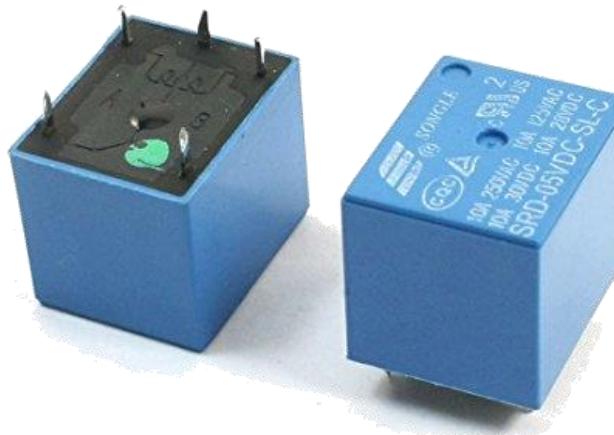
- LED and Buzzer Both are generate some output signal based on the input voltage.
- LED converts Electrical Energy into the Light Energy spectrum.
- Buzzer converts Electrical energy into Sound signal.
- We can connect both as output device.
- pinMode for both : OUTPUT.



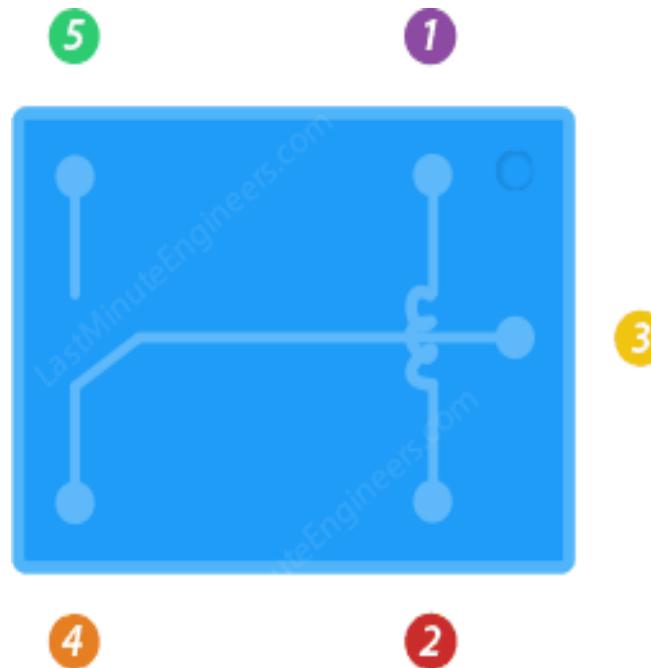
Relay

A relay is an electrically activated switch.

- 5V Relay Module 10A current capacity.
- This small Relay Board works from a 5V signal. It uses a transistor to switch the relay on so can be connected directly to a microcontroller pin.
- When the **relay is not activated**, the common pin is in contact with the **NC pin** and when it is **activated**, the common pin will break away from contact with the NC pin and subsequently makes contact with the **NO pin**.



Relay Terminals

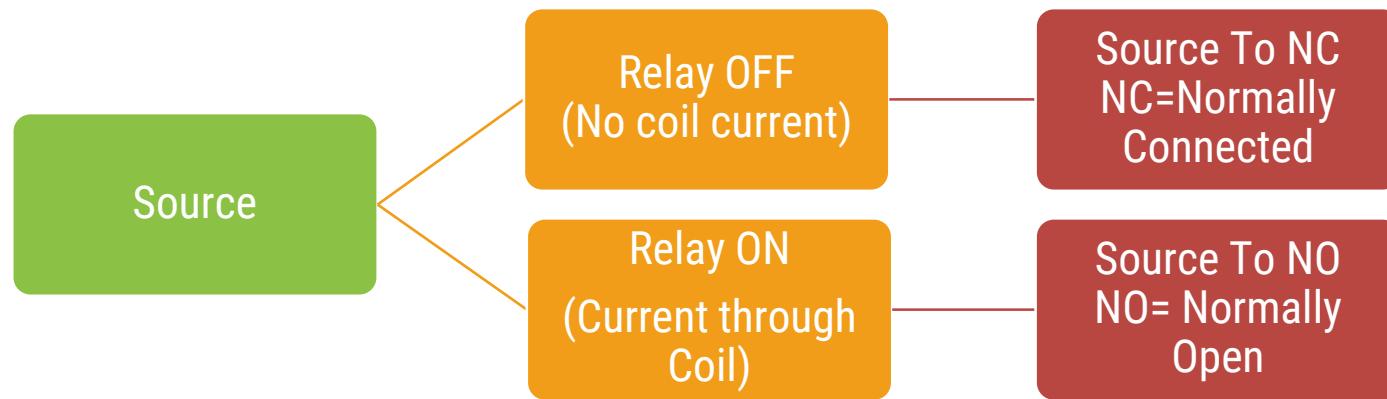


- 1 Coil 1
- 2 Coil 2
- 3 COM - Common
- 4 NC - Normally Closed
- 5 NO - Normally Open

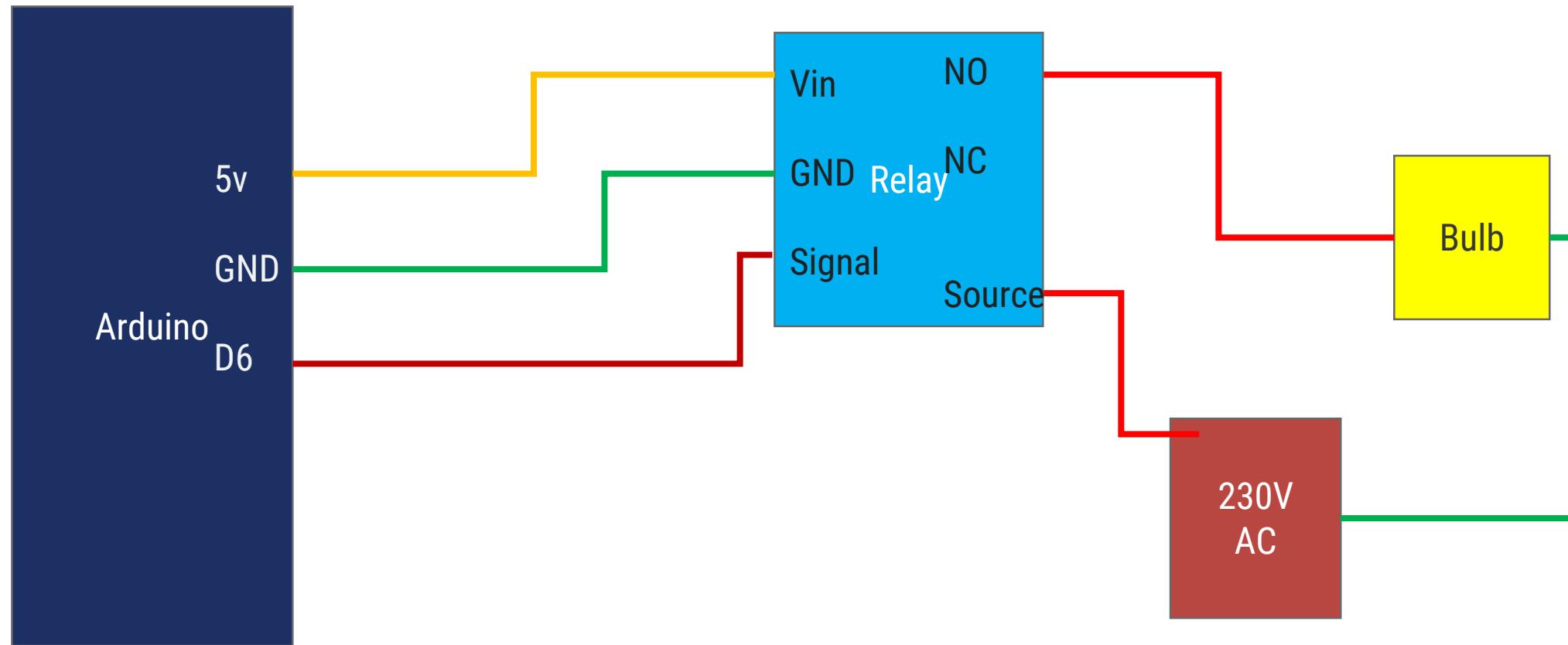
Relay Module



How Relay Works?



Relay Connection

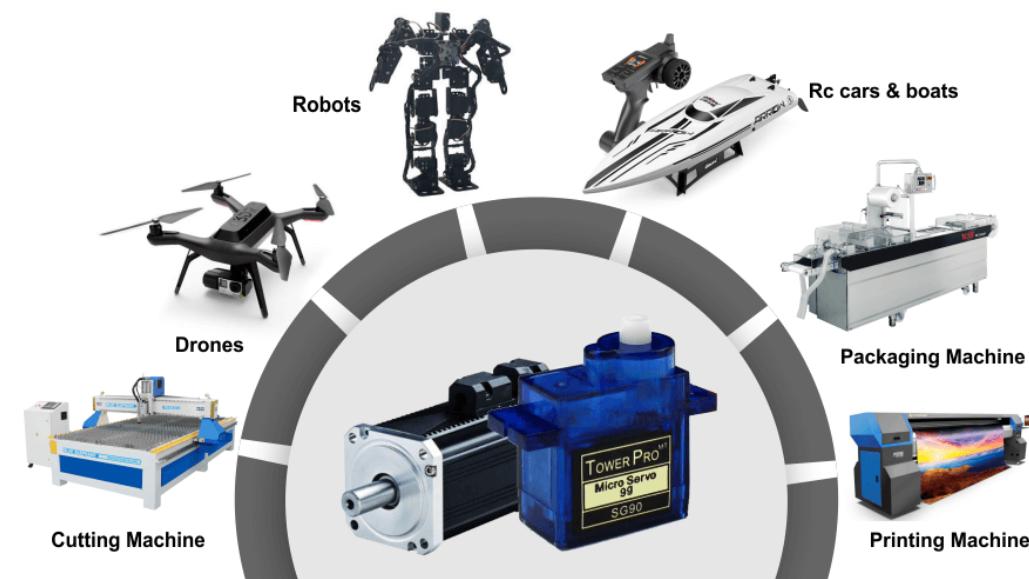


Relay.ino

```
1 int RelayPin = 6;  
2  
3 void setup() {  
4     // Set RelayPin as an output pin  
5     pinMode(RelayPin, OUTPUT);  
6 }  
7  
8 void loop() {  
9     // Let's turn on the relay...  
10    digitalWrite(RelayPin, HIGH);  
11    delay(3000);  
12  
13    // Let's turn off the relay...  
14    digitalWrite(RelayPin, LOW);  
15    delay(3000);  
16 }
```

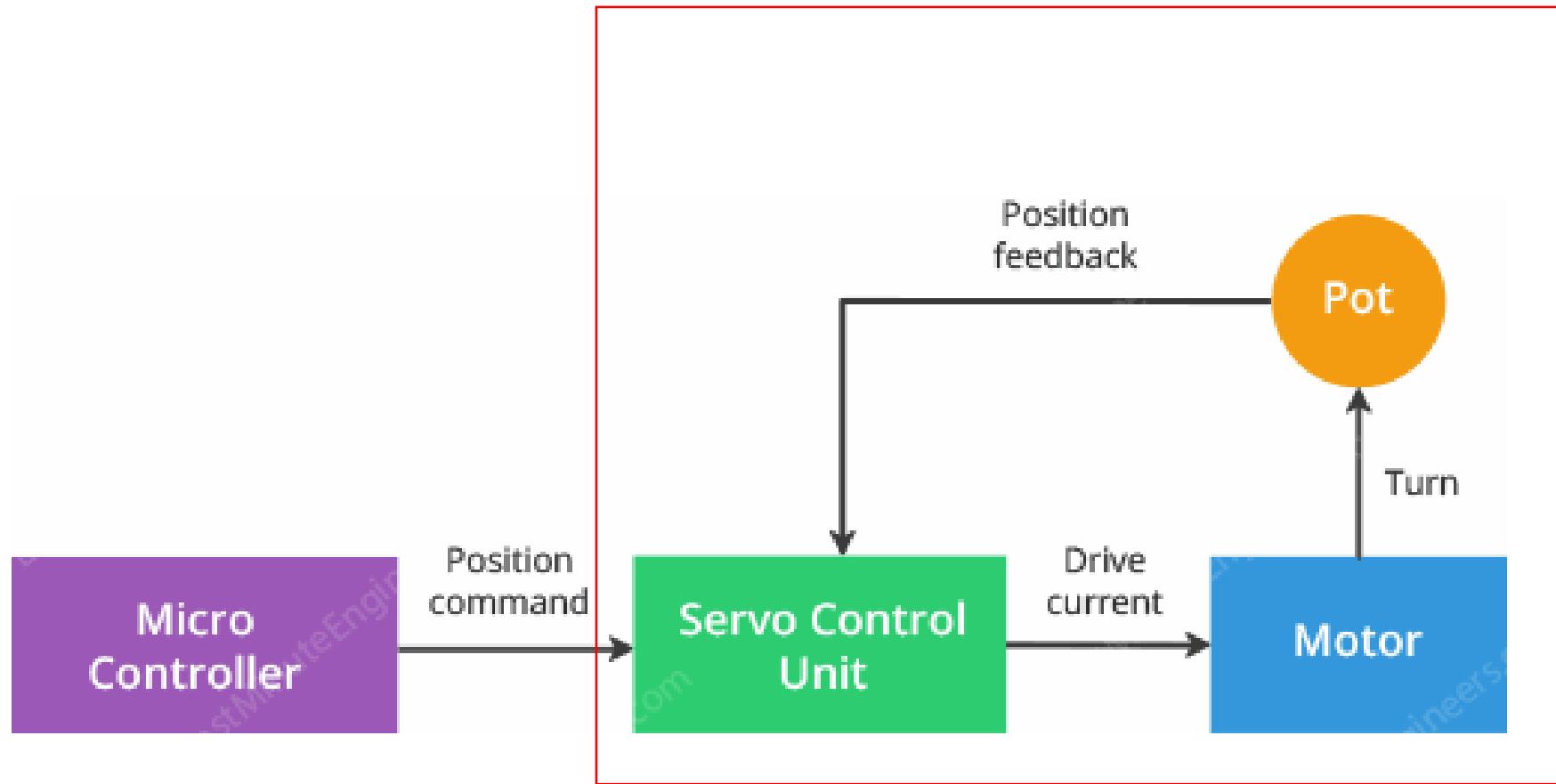
Applications of Relay

- Commonly used in switching circuits.
- For Home Automation projects to switch AC loads
- To Control (On/Off) Heavy loads at a pre-determined time/condition
- Used in safety circuits to disconnect the load from supply in event of failure
- Used in Automobiles electronics for controlling indicators glass motors etc.



What is Servo?

- **Servo** is a general term for a **closed loop control system**.
- A closed loop system uses the feedback signal to adjust the speed and direction of the motor to achieve the desired result.



Servo Motor

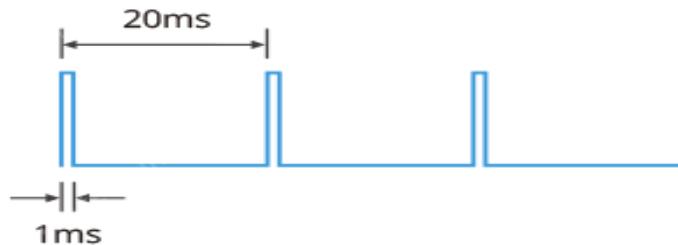
A servo motor is an example of an actuator.

- They are linear or rotatory actuators, can move to a given specified angular or linear position.
- We can use servo motors for IoT applications and make the motor rotate to 90 degrees, 180 degrees, etc., as per our need.
- A servomotor is **a closed-loop servomechanism** that uses **position feedback** to control its motion and final position.
- The input to its control is a signal (either analogue or digital) representing the position commanded for the output shaft.



Servo Motor Working

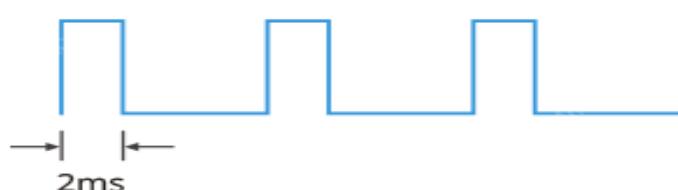
- This signal generation will be taken care of by Arduino's servo library.
- All we have to do is use the servo library.



0 Degrees

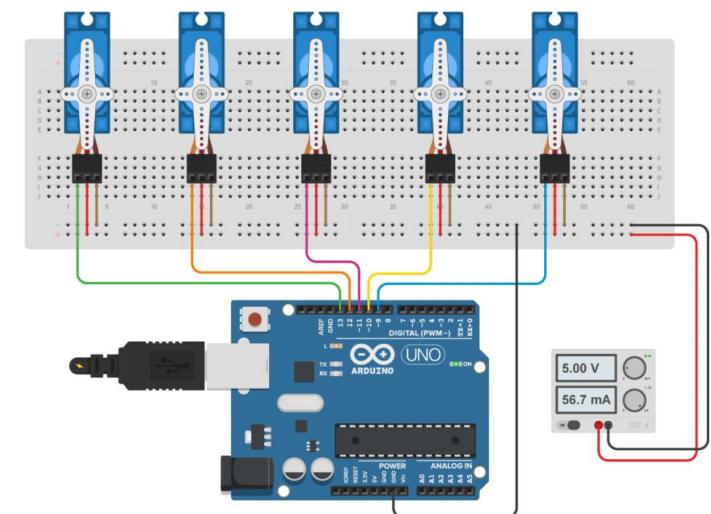
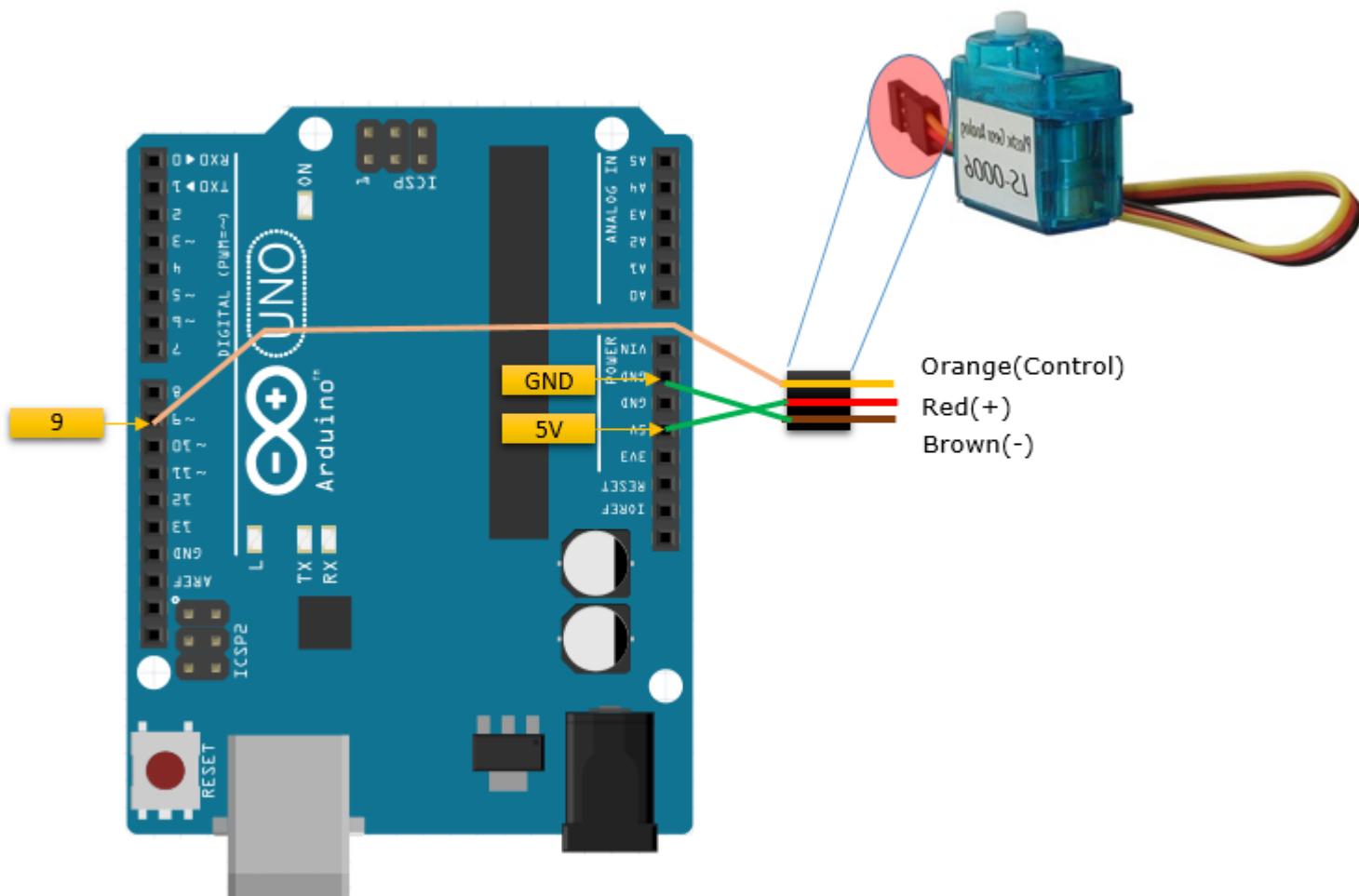


90 Degrees



180 Degrees

Servo Motor with Arduino



Servo Motor with Arduino



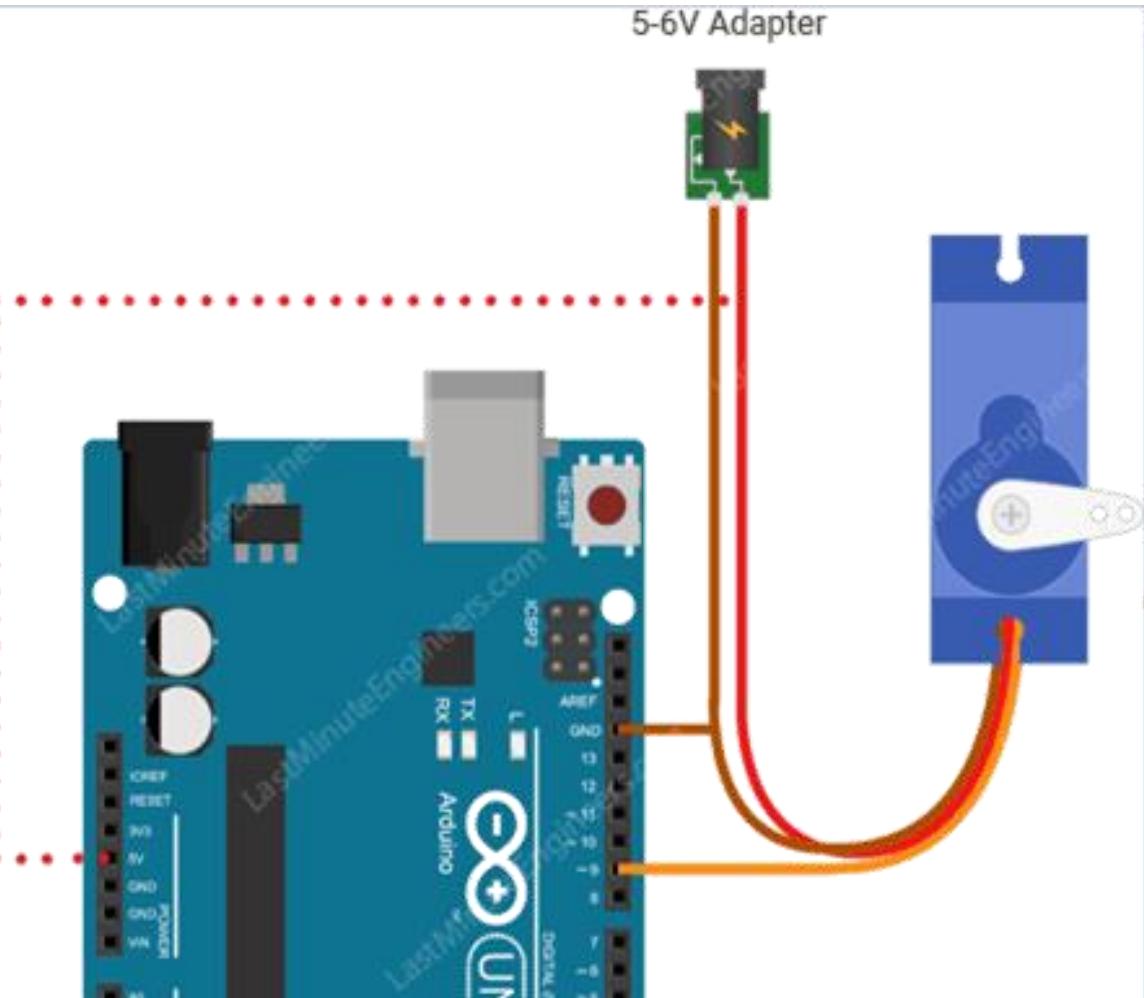
Servo Pinout

GND is a common ground for both the motor and logic.

5V is a positive voltage that powers the servo.

Control is input for the control system.

If you are using a micro servo then you can power it directly from the Arduino



Servo with Arduino

- There is a **Servo library**, which allows Arduino boards to control a variety of servo motors.
- This library can control a great number of servos.
- It makes careful use of timers: the library can control 12 servos using only 1 timer.
 - ❖ On the Arduino Due you can control up to 60 servos.

■ Methods

1. attach()// attach pin to servo variable
2. write()//write value to the servo motor variable for angle
3. read()//read the last write(); means current angle
4. attached()//attach servo variable to the particular pin

(Note: Arduino 0016 and earlier, the Servo library supports servos on only two pins: 9 and 10.)

5. detach() //If all Servo variables are detached, then pins 9 and 10 can be used for PWM output with analogWrite().

Servo Methods

| ■ attach() | ■ write() | ■ read() | ■ attached() | ■ detach() |
|---|---|---|--|---|
| <ul style="list-style-type: none">• Attach the Servo variable to a pin. | <ul style="list-style-type: none">• Writes a value to the servo, controlling the shaft accordingly.• On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. | <ul style="list-style-type: none">• Read the current angle of the servo (the value passed to the last call to write()). | <ul style="list-style-type: none">• Check whether the Servo variable is attached to a pin. | <ul style="list-style-type: none">• Detach the Servo variable from its pin. |
| • Syntax | • Syntax | • Syntax | • Syntax | • Syntax |
| ➤ <code>servo.attach(pin)</code> | ➤ <code>servo.write(angle)</code> | ➤ <code>servo.read()</code> | ➤ <code>servo.attached()</code> | ➤ <code>servo.detach()</code> |

Servo Methods

| attach() | write() | read() | attached() | detach() |
|--|--|---|---|--|
| <ul style="list-style-type: none">• Syntax➤ servo.attach(pin) | <ul style="list-style-type: none">• Syntax➤ servo.write(angle) | <ul style="list-style-type: none">• Syntax➤ servo.read() | <ul style="list-style-type: none">• Syntax➤ servo.attached() | <ul style="list-style-type: none">• Syntax➤ servo.detach() |
| <pre>#include <Servo.h> Servo myservo; void setup() { myservo.attach(9); //attach servo to pin 9 } void loop() { }</pre> | <pre>#include <Servo.h> Servo myservo; void setup() { myservo.attach (9); myservo.write (90); // set servo to mid-point } void loop() { myservo.write (0); // set servo to start-point }</pre> | <pre>#include <Servo.h> Servo myservo; void setup() { myservo.attach (9); myservo.write (90); } void loop() { int agl; agl = myservo.read(); // return angle value //here agl = 90; }</pre> | <pre>#include <Servo.h> Servo myservo; void setup() { myservo.attach (9); myservo.write (90); // set servo to mid-point } void loop() { bool joined; joined = myservo.attached(); // return true if attached //return false if not attached }</pre> | <pre>#include <Servo.h> Servo myservo; void setup() { myservo.attach (9); } void loop() { //code myservo.detach (9); //detach Servo from //pin 9 }</pre> |

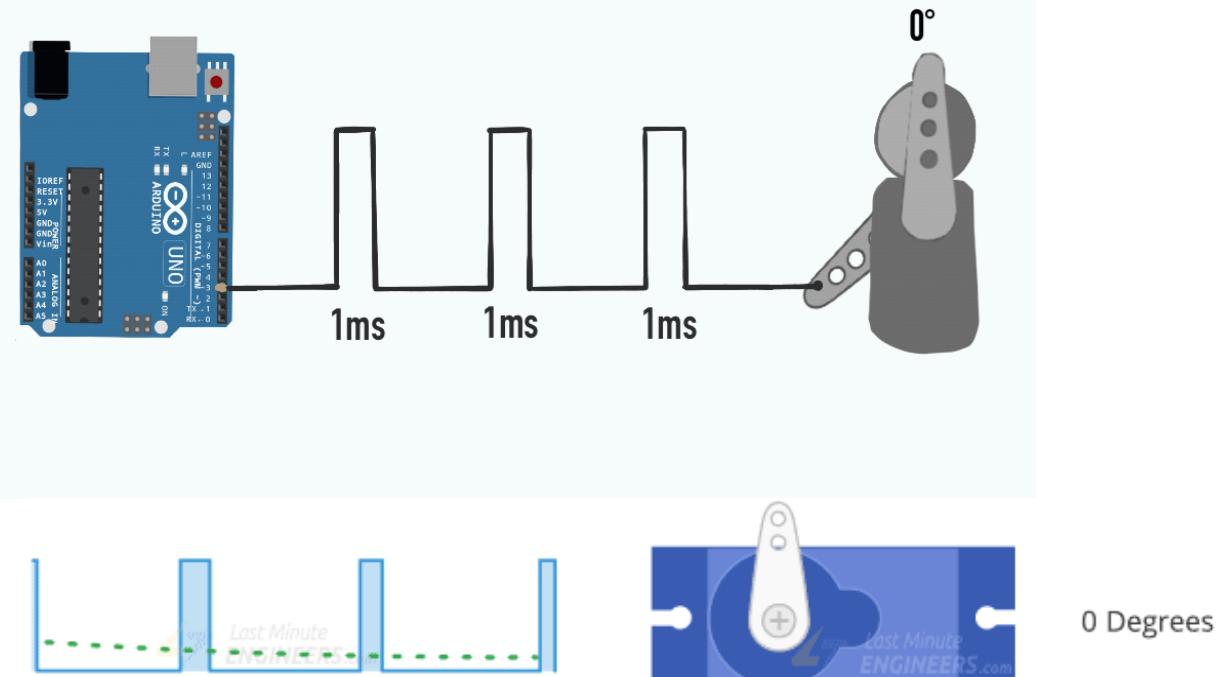
Servo motor – Code explanation

- The code in Arduino for Servo motor can be written as following:

servo.ino

```
1 #include <Servo.h>
2
3 Servo darshanservo,btechservo;
4 void setup() {
5   Serial.begin(9600);
6   btechservo.attach(9);//attach servo to pin 9
7   darshanservo.attach(11);//attach servo to pin 11
8 }
9 void loop()
{
10   darshanservo.write(10);
11   btechservo.write(45);
12   delay(15);
13 }
14
15
16 }
```

SERVO MOTOR CONTROL



Arduino - DC Motor

- There are three different type of motors -
 - ❖ DC motor
 - ❖ Servo motor
 - ❖ Stepper motor
- A DC motor (Direct Current motor) is the most common type of motor.
- DC motors normally have just two leads, one positive and one negative.
- If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.



Motor Speed Control

dcmotor.ino

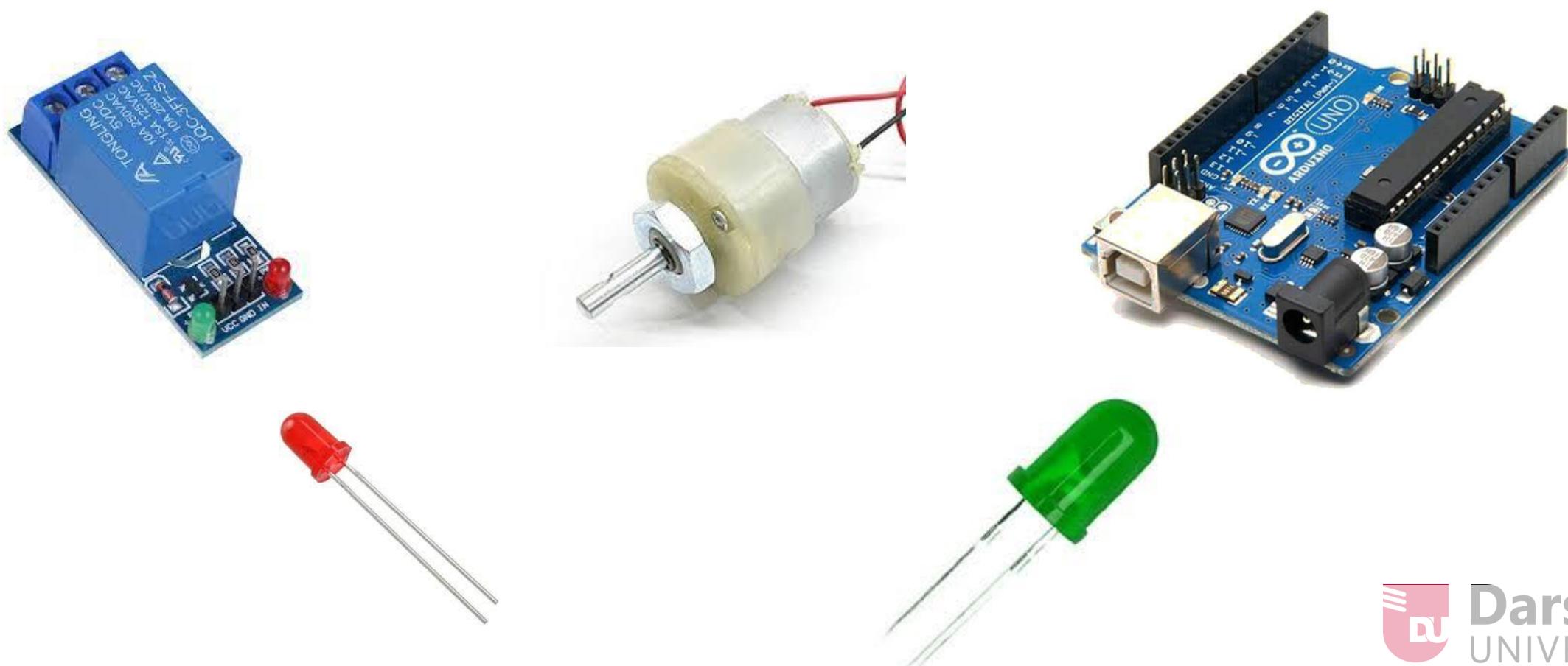
```
1 int motorPin = 9;
2 void setup() {
3     pinMode(motorPin, OUTPUT);
4     Serial.begin(9600);
5     while (! Serial);
6     Serial.println("Speed 0 to 255");
7 }
8 void loop() {
9     if (Serial.available()) {
10         int speed = Serial.parseInt();
11         if (speed >= 0 && speed <= 255) {
12             analogWrite(motorPin, speed);
13         }
14     }
15 }
16 }
```



Darshan
UNIVERSITY

गोपा: कर्मसु कौशलम्

Clock Wise and Anti Clockwise Rotation



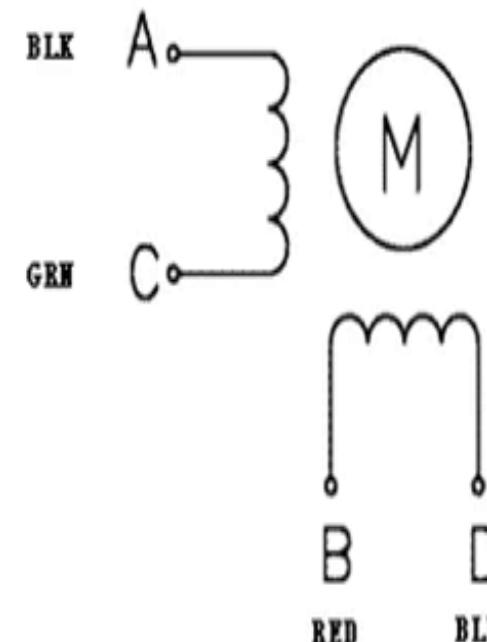
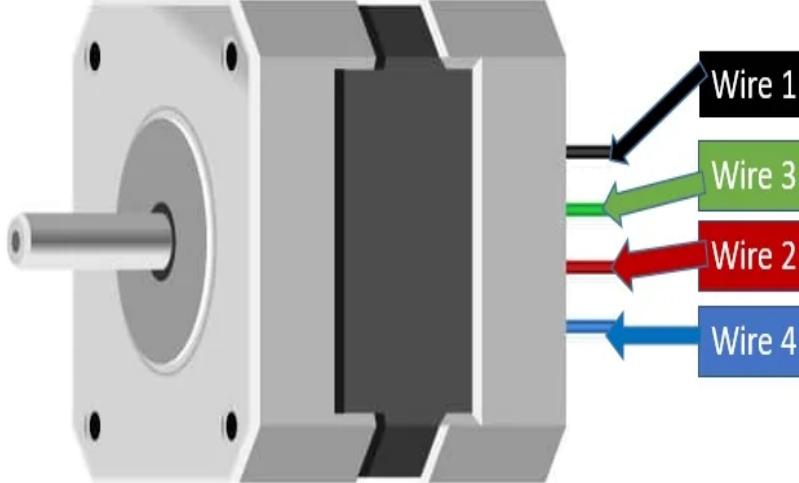
Stepper Motor

- Stepper motors are often an extremely important component in a motion control system.
- A Stepper Motor or a step moto divides a full rotation into a number of steps.
- A brushed DC motor rotates continuously when a fixed DC voltage is applied to it, while a step motor rotates in discrete step angles.
- The Stepper Motors therefore are manufactured with steps per revolution of 12, 24, 72, 144, 180, and 200, resulting in stepping angles of 30, 15, 5, 2.5, 2, and 1.8 degrees per step.
- The stepper motor can be controlled with or without feedback.
- The motor spins very fast in one direction or another.
- We can vary the speed with the amount of power given to the motor, but we cannot tell the propeller to stop at a specific position like servo motor.



Bipolar Stepper Motor

- Bipolar Stepper Motor :
 - ❖ Motors always have 2 coils and 4 wires.
- By driving the current in separate directions through each of the coils, we can have a total of 4 different states:
 1. Coil A current flowing 'left to right'.
 2. Coil A current flowing 'right to left'.
 3. Coil B current flowing 'left to right'.
 4. Coil B current flowing 'right to left'.



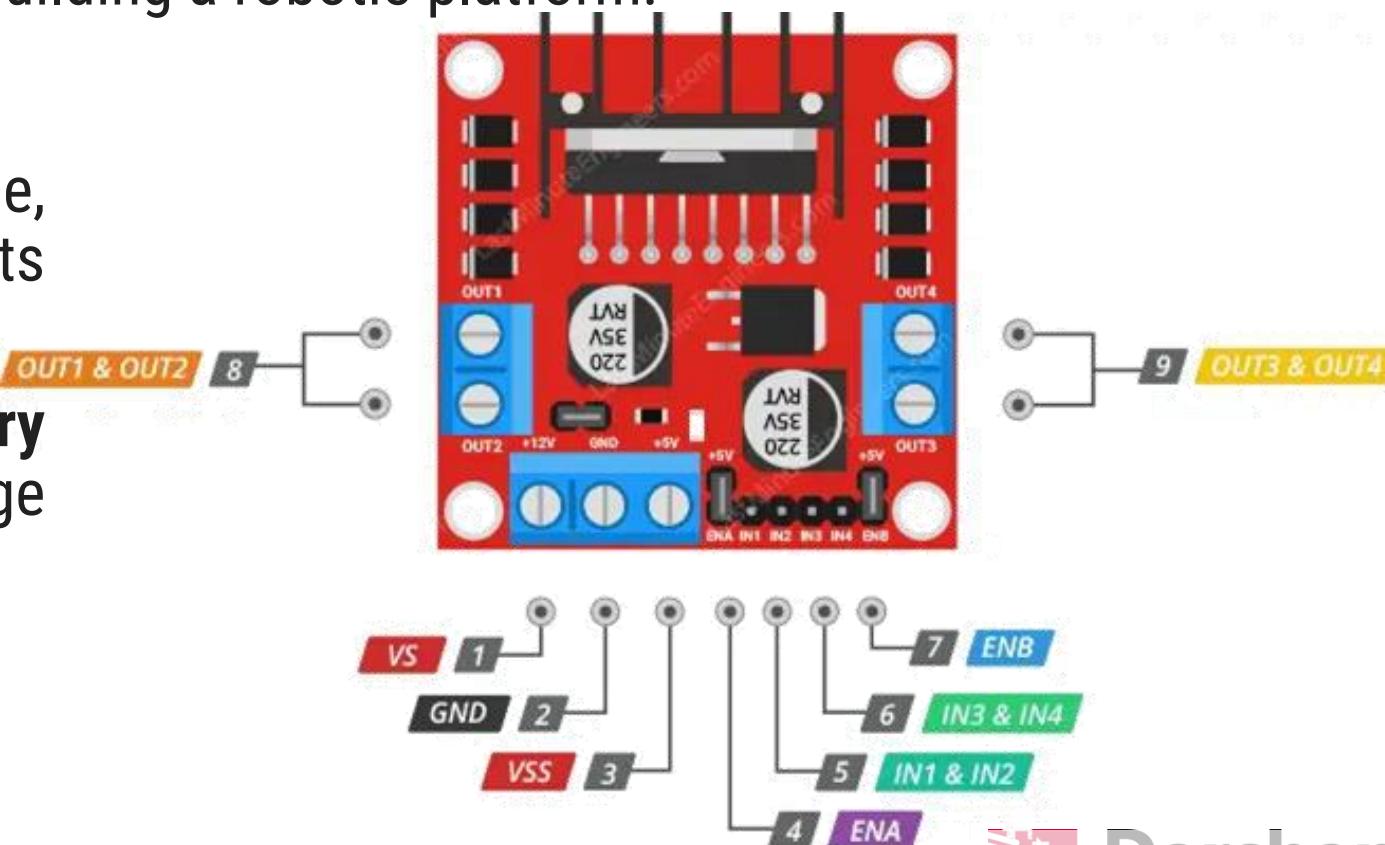
| Pin No | Name | Name | Colour |
|--------|------|------|--------|
| 1 | A | A+ | Black |
| 2 | C | A- | Green |
| 3 | B | B+ | Red |
| 4 | D | B- | Blue |

Terminals

Coil A and Coil B

L298N : Driving IC

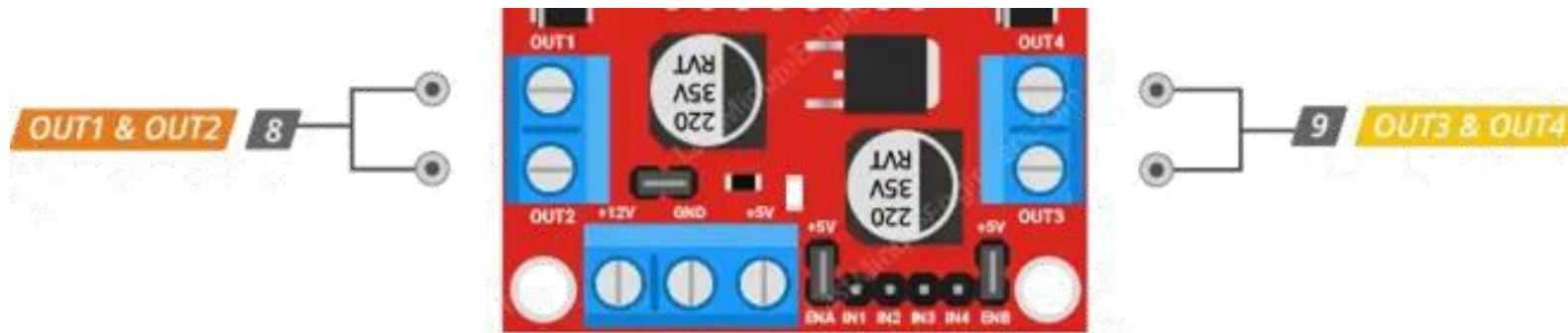
- **L298N** is a Motor Driver Chip.
- The L298N chip contains two standard H-bridges capable of driving **a pair of DC motors or single Stepper motor** making it ideal for building a robotic platform.
- **Power Pins:**
- **VS** pin powers the IC's internal H-Bridge, which **drives the motors**. This pin accepts input voltages ranging from **5 to 12V**.
- **VSS** is used to power the logic circuitry within the L298N IC, and can range between **5V and 7V**.
- **GND** is the common ground pin.



Darshan
UNIVERSITY
गोपा कर्मसु कौशलम्

L298N : DC Motor

- **Output Pins :**
- **Two DC Motor:**
- The output channels of the L298N motor driver:
- **OUT1** and **OUT2** for **motor A**
- **OUT3** and **OUT4** for **motor B.**
- We can connect two 5-12V DC motors to these terminals.



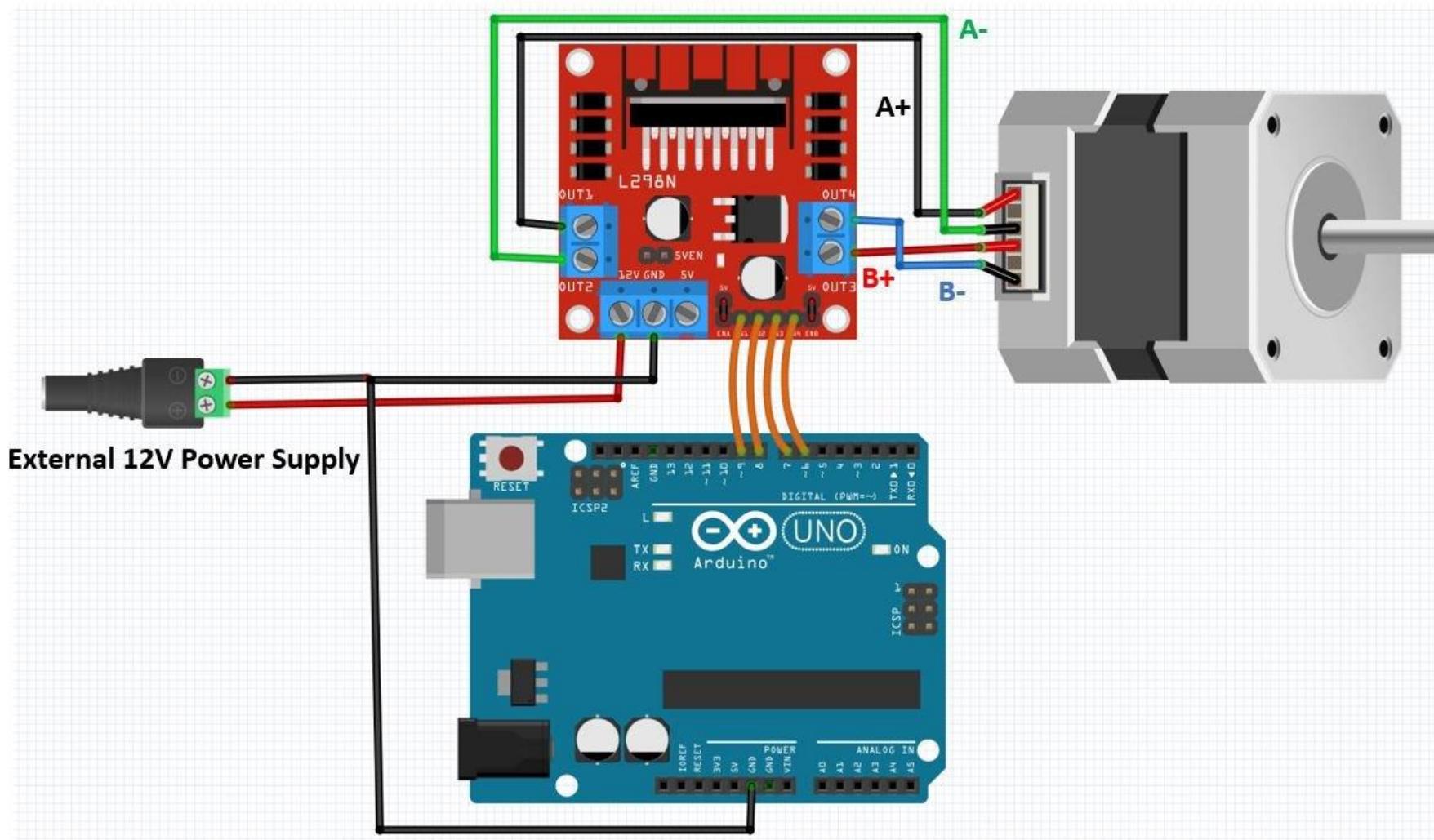
L298N : Stepper Motor

- **Stepper Motor:**
- For which ever stepper motor you are using refer to its datasheet to find the correct colored wires to differentiate the A+, A-, B+ and B- wires. This will enable us to properly connect the motor with the OUT1, OUT2, OUT3 and OUT4 pins of the driver module.
- **Black wire = A+ → OUT1**
- **Green wire = A- → OUT2**
- **Red wire = B+ → OUT3**
- **Blue wire = B- → OUT4**
- Connect positive terminal of 12V power supply with 12V terminal and negative with common ground.



Bipolar Stepper Motor Connection With Arduino UNO & LM298

■ Connection with Arduino:



Code

Stepper.ino

```
1 /* Stepper Motor Control - one revolution
2 This program drives a unipolar or bipolar stepper motor.
3 The motor is attached to digital pins 8 - 11 of the Arduino.
4 The motor should revolve one revolution in one direction, then
5 one revolution in the other direction. */
6 #include <Stepper.h>
7 const int stepsPerRevolution = 200;
8 // change this to fit the number of steps per revolution
9 #define IN1 9
10 #define IN2 8
11 #define IN3 7
12 #define IN4 6
13 // initialize the stepper library on pins 6 through 9:
14 Stepper myStepper(stepsPerRevolution, IN1, IN2, IN3, IN3);
15 //200 steps
```

Stepper.ino

```
15 void setup() {  
16   myStepper.setSpeed(60); // set the speed at 60 rpm:  
17   Serial.begin(9600); // initialize the serial port: 9600 baud rate  
18 }  
19 void loop() {  
20   // step one revolution in one direction:  
21   Serial.println("clockwise");  
22   myStepper.step(stepsPerRevolution); //+ve Clockwise  
23   delay(500);  
24   // step one revolution in the other direction:  
25   Serial.println("counterclockwise");  
26   myStepper.step(-stepsPerRevolution); // -ve Counter clockwise  
27   delay(500);  
28 }
```

- Bluetooth is everywhere nowadays. It's a word we hear all the time and is in millions of products we use everyday, including headsets, cellphones, laptops, game controllers, activity trackers, and so on.
- One of the most affordable and widely used Bluetooth modules is the HC-05.

□HC-05 Hardware Overview

- The HC-05 is a Bluetooth-to-Serial-Bridge module that allows wireless communications between two microcontrollers or between a microcontroller and a smartphone, laptop, or desktop PC with Bluetooth capability.
- It's perfect for **directly replacing a wired asynchronous serial interface**.
- Each of these modules contains a Bluetooth transceiver, meaning they're capable of both sending and receiving data.

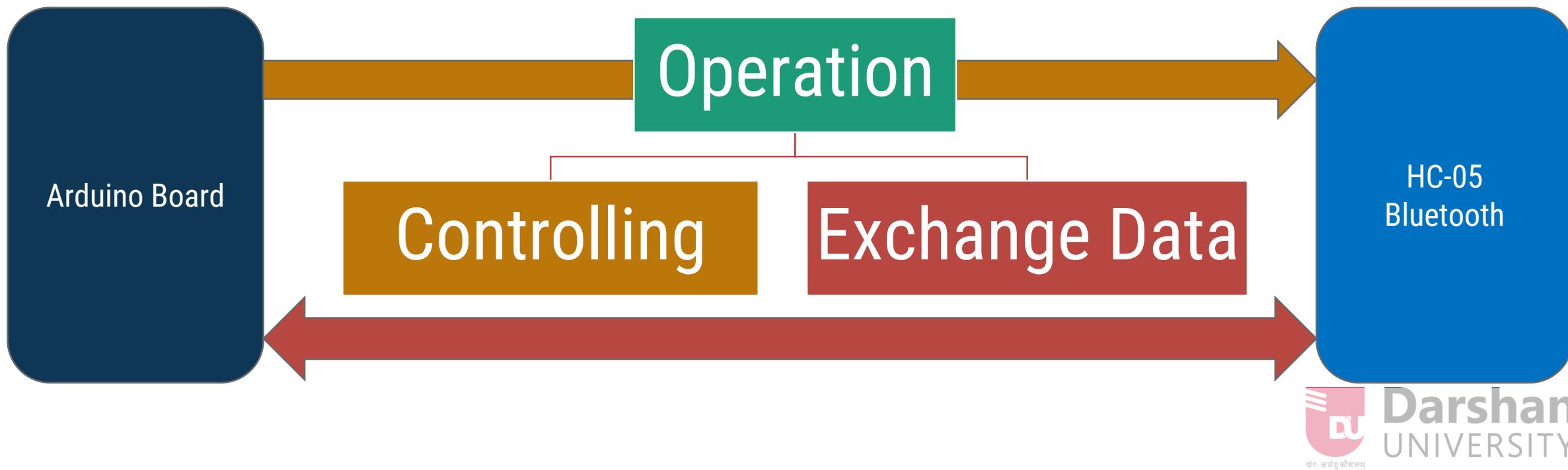
Bluetooth Module : HC-05

- HC-05 is Class 2 Bluetooth device.
- HC-05 has a nominal range of 10 m.
- HC-05 module has a **3.3V logic level**.
- There's no need to mess with Bluetooth protocols or the stack. Just send data over a serial interface, and it's piped through to whatever Bluetooth device it's connected to.



HC-05: Modes of Operation

- Basic Two Operation:
 1. Controlling the HC-05 module
 2. Sending user data through module
- Both operation accomplished through the serial interface.



HC-05 Pinout (6 Pin)

1. **STATE pin** can be used to determine the current status of the HC-05 module.
 - ❖ The State pin is LOW when the module is not paired and HIGH when it is.
2. **RXD pin** receives serial data from the microcontroller. It should be connected to the **TX** of the microcontroller. Please note that this pin is not 5V-tolerant. Therefore, **before connecting the module to a 5V microcontroller, the microcontroller's Tx signal must be stepped down to 3.3V.**
3. **TXD pin** sends serial data to the microcontroller. It should be connected to the **RX** of the microcontroller.
4. **GND** is the ground pin, common to any other device connected to the module.
5. **VCC** is where you connect the positive supply voltage. This voltage supply signal is routed to the HC-05 chip via a **3.3V regulator**. It should range from 3.6V to 6V.
6. **EN** is connected to the on-board regulator enable pin and **is pulled high by a 220k resistor**. Pulling this pin low disables the regulator, which consequently turns off the HC-05.

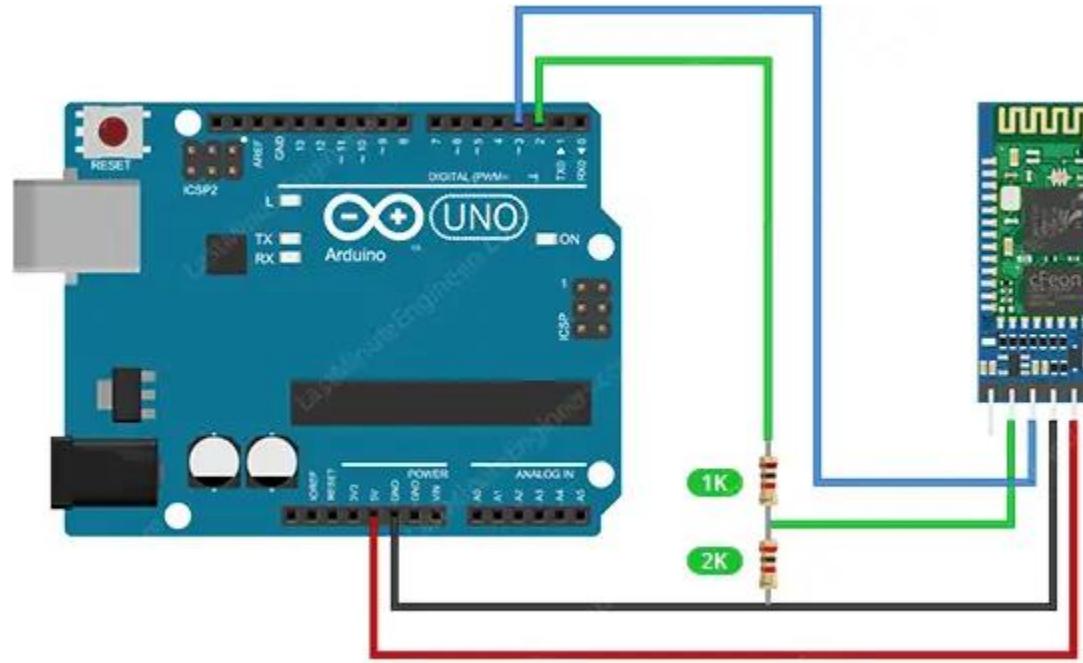
HC-05 vs HC-06

- The HC-05 and HC-06 are Bluetooth modules for Arduinos that differ in their programming and mode.
- The HC-05 can act as both a **master** and a **slave** device.
- The HC-06 can **only** act as a **slave** device.
- The HC-05 is also easier to program than the HC-06.

Wiring a HC-05 Module to an Arduino

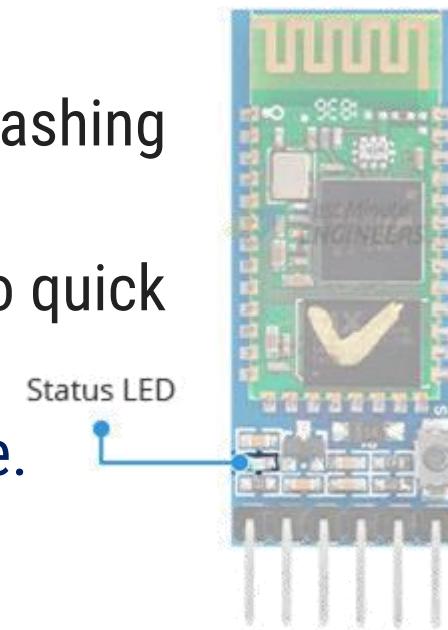
| HC-05 Module | Arduino | Notes |
|--------------|---------|-----------------------------------|
| VCC | 5V | - |
| GND | GND | - |
| TXD | D3 | - |
| RXD | D2 | Use level shifter if using 5V MCU |

Software Serial Port
to avoid bus contention
and ensure that the HC-05
doesn't receive any
unintended data during a
sketch upload.



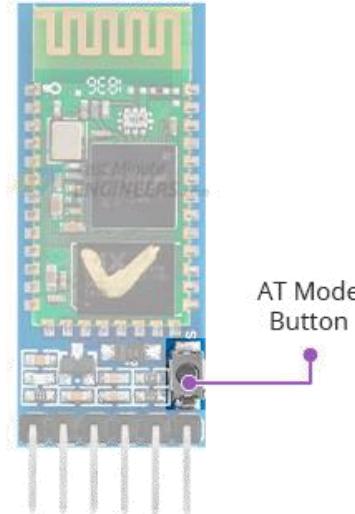
Status LED

- Many HC-05 modules come with an onboard LED.
- LED blinks at various rates to indicate the status:
 - ❖ When **powered up**, the module enters Bluetooth **pairing mode**, with the LED flashing rapidly at about 2 Hz.
 - ❖ When the **module is paired** with a device, the LED flash pattern changes to two quick flashes, followed by a pause, and then repeats.
 - ❖ When the module is put **into AT mode**, the LED blinks at a slow and steady rate.
 - ❖ Default Baud rate for AT mode is 38400 bps, but it can be changed if necessary.



AT - Mode

- AT Mode is the configuration mode.
- With sending **Hayes AT-style** commands to the HC-05 module we can change its settings like name, baud rate, password, etc.
- **Normally, the HC-05 module is in data mode.**
- To put it into AT mode, we need to press-and-hold the onboard button while powering up the module.
- The LED will then start blinking at a slow and steady rate, indicating that the module is in AT mode.
- Once in AT mode, we can send AT commands to the module over the **UART** port.
- **The module will respond** to the commands, either by acknowledging the command, providing the requested data, or signaling an error.



AT-Mode

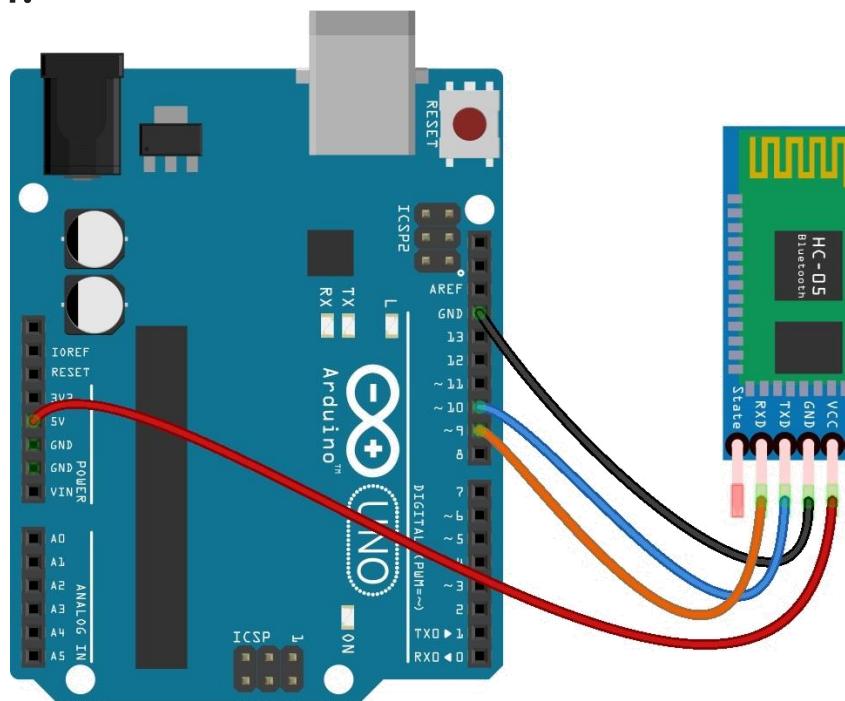
- AT commands are essentially modem instructions.
- Originally developed by the modem maker **Hayes** as means to operate their dial-up landline products.
- AT commands — the 'AT' stands for 'come to ATtention' — are now used by all modems, of all types.
- The commands usually start with “AT+” followed by the specific command.
- “AT+NAME?” queries the name of the module.
- “AT+NAME=MyHC05” changes the name to “MyHC05”

- AT : Check the connection.
- AT+NAME : See default name or Set Name
- AT+ADDR : see default address
- AT+VERSION : See version
- AT+UART : See baudrate or Set Baud rate
- AT+ROLE: See role of bt module(1=master/0=slave)
- AT+RESET : Reset and exit AT mode
- AT+ORGL : Restore factory settings
- AT+PSWD: see default password

- AT+NAME: Change name. No space between name and command.
- AT+NAMEPROTOTYPE will set the name to PROTOTYPE.
- AT+BAUD: change baud rate, x is baud rate code, no space between command and code.
- To change baud rate, type AT+BAUDX, where X=1 to 9.
- 1 set to 1200bps
- 2 set to 2400bps
- 3 set to 4800bps
- 4 set to 9600bps (Default)
- 5 set to 19200bps
- 6 set to 38400bps
- 7 set to 57600bps
- 8 set to 115200bps
- so sending AT+BAUD4 will set the baud rate to 9600.

Data Mode

- In Data Mode, the HC-05 module acts as a transparent data gateway.
- When the HC-05 receives data, it removes the Bluetooth headers and trailers and sends it to the UART port.
- When data is written to the UART port, the HC-05 constructs a Bluetooth packet and sends it over the Bluetooth wireless connection.



Serial Communication with Bluetooth Module

- The code in Arduino for Serial communication for Bluetooth Module can be written as following:

bluetooth.ino

```
1 #include <SoftwareSerial.h>
2 //Create software serial object to communicate with HC-05
3
4 SoftwareSerial mySerial(3, 2);
//HC-05 Tx & Rx is connected to Arduino #3 & #2
5
6
7 void setup() {
8 //Begin serial communication with Arduino and Arduino IDE Serial Monitor
9     Serial.begin(9600);
10
11 //Begin serial communication with Arduino and HC-05
12     mySerial.begin(9600);
13 Serial.println("Initializing..."); 
14 Serial.println("The device started, now you can pair it with bluetooth!"); 
15 }
```

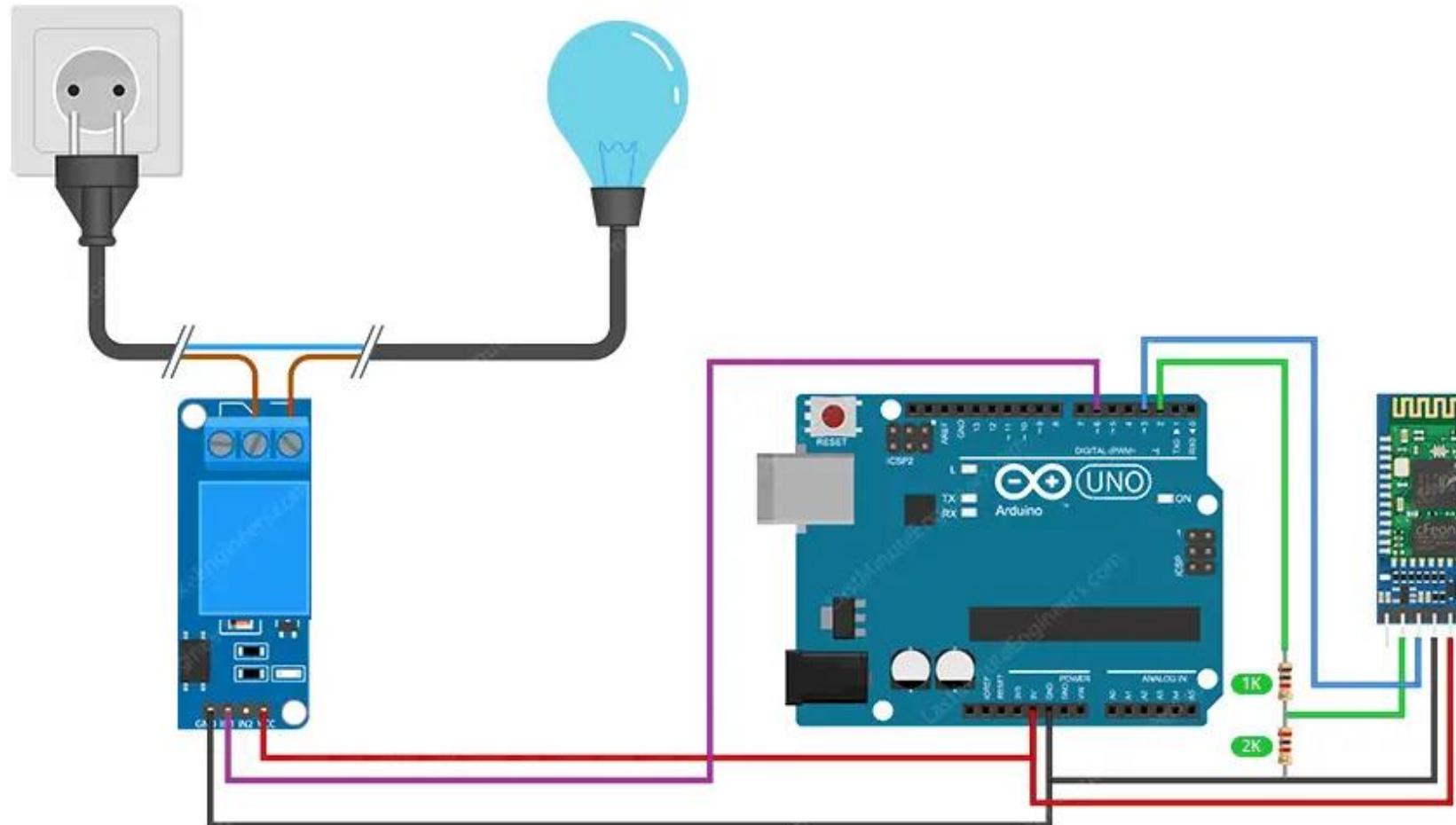
Serial Communication with Bluetooth Module

servo.ino

```
16 void loop() {  
17  
18     if(Serial.available())  
19     {  
20         mySerial.write(Serial.read());  
21         //Forward what Serial received to Software Serial Port  
22     }  
23  
24     if(mySerial.available())  
25     {  
26         Serial.write(mySerial.read());  
27         //Forward what Software Serial received to Serial Port  
28     }  
29     delay(20);  
30 }
```

Bluetooth-Controlled Relay Example

- Control relays wirelessly using Bluetooth can be useful for home automation, smart lighting, security systems, and other similar applications.



Serial Communication with Bluetooth Module

- The code in Arduino for Serial communication for Bluetooth Module can be written as following:

bluetooth.ino

```
1 #include <SoftwareSerial.h>
2
3 // GPIO where relay is connected to
4 const int relayPin = 6;
5
6 // Handle received messages
7 String message = "";
8
9 // Create software serial object to communicate with HC-05
10 SoftwareSerial mySerial(3, 2);
11 //HC-05 Tx & Rx is connected to Arduino #3 & #2
12
13 void setup() {
14 // Begin serial communication with Arduino and Arduino IDE (Serial Monitor)
15 Serial.begin(9600);
```

Serial Communication with Bluetooth Module

- The code in Arduino for Serial communication for Bluetooth Module can be written as following:

bluetooth.ino

```
16 // Initialize relayPin as an output
17 pinMode(relayPin, OUTPUT);
18 digitalWrite(relayPin, LOW);
19 //Begin serial communication with Arduino and HC-05
20 mySerial.begin(9600);
21 Serial.println("Initializing...");
22 Serial.println("The device started, now you can pair it with bluetooth!");
23 }
24     void loop()
25     {
26         if (mySerial.available()){
27             char incomingChar = mySerial.read();
28             if (incomingChar != '\n'){
29                 message += String(incomingChar);
30             }
31     }
```

Serial Communication with Bluetooth Module

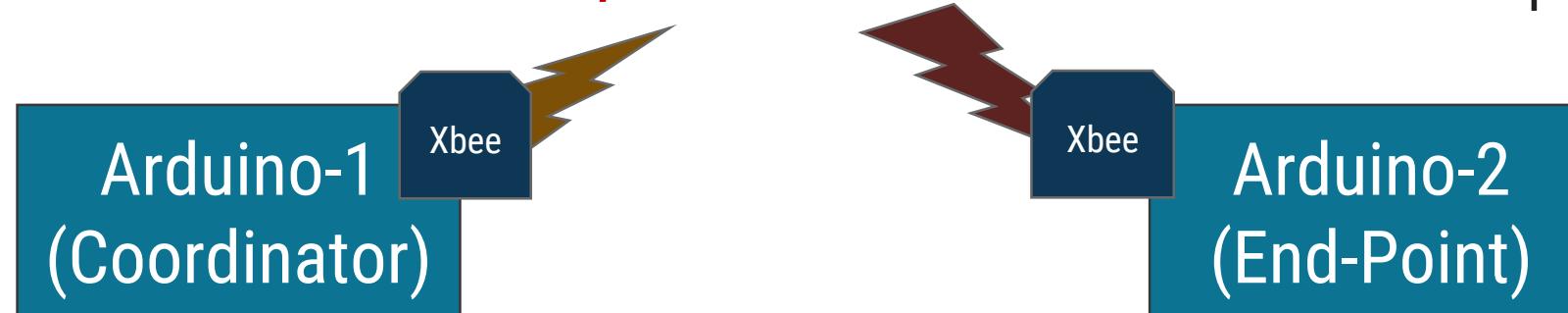
- The code in Arduino for Serial communication for Bluetooth Module can be written as following:

bluetooth.ino

```
31     else  {
32             message = "";
33         }
34         Serial.write(incomingChar);
35     }
36
37 // Check received message and control output accordingly
38     if (message == "on"){
39         digitalWrite(relayPin, HIGH);
40     }
41     else if (message == "off"){
42         digitalWrite(relayPin, LOW);
43     }
44 delay(20);
45 }
```

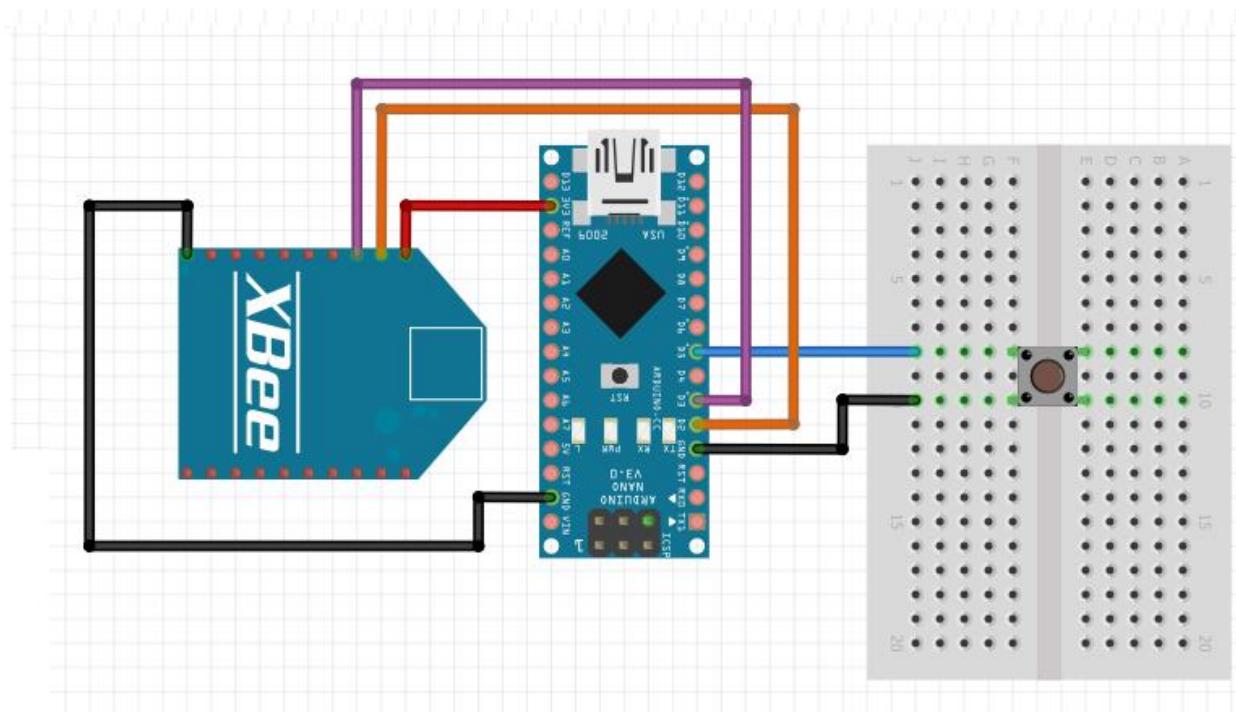
Zigbee/XBee Module with Arduino and NodeMCU

- Zigbee is a popular wireless communication protocol used to transfer a small amount of data with very low power.
- It is widely used in applications where data has to be shared among many nodes within personal space and with the advent of the Internet of things (IoT).
- **XBee is a module**, a product, which supports many wireless communication protocols like ZigBee, 802.15.4, 868 MHz modules, etc.
- **Zigbee is a standard protocol** used to establish wireless networking (ZigBee communication).
- To establish **Zigbee communication**:
- A receiver (**endpoint**) (receive data) and transmitter (**coordinator**) (Send Data).
- **XBee module connected with Arduino/NodeMCU** which works either as endpoint or coordinator



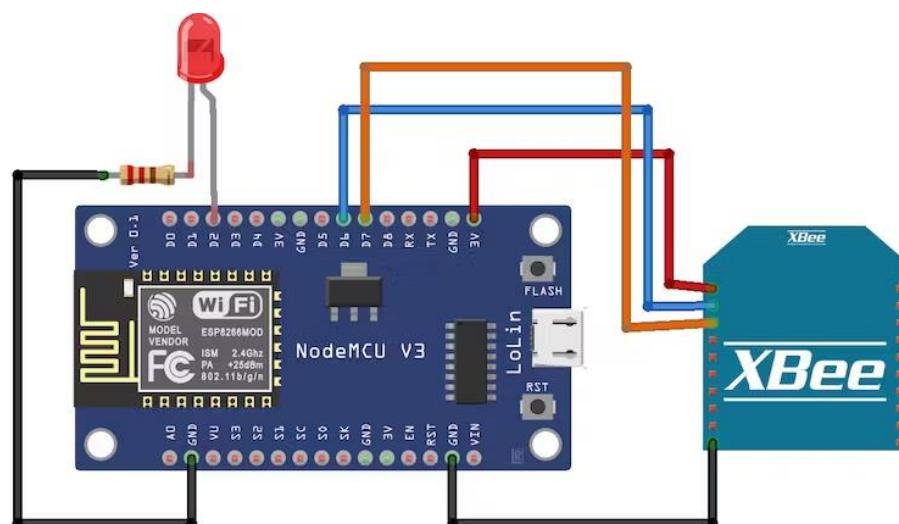
Interfacing XBee With Arduino (Transmitting Side)

- Connect **VCC (pin 1)** of XBee module to **3.3V** of Arduino Nano .
- **GND (pin 10)** of XBee to **GND** of Arduino Nano.
- These two connections make up for powering the transmitting side XBee module.
- Connect **Dout (pin 2)** to **D2** of Arduino Nano.
- **Din (pin 3)** to **D3** of Arduino Nano.
- **Pushbutton** Connected to **D5**.
 - ❖ Active Low.(**INPUT_PULLUP**)



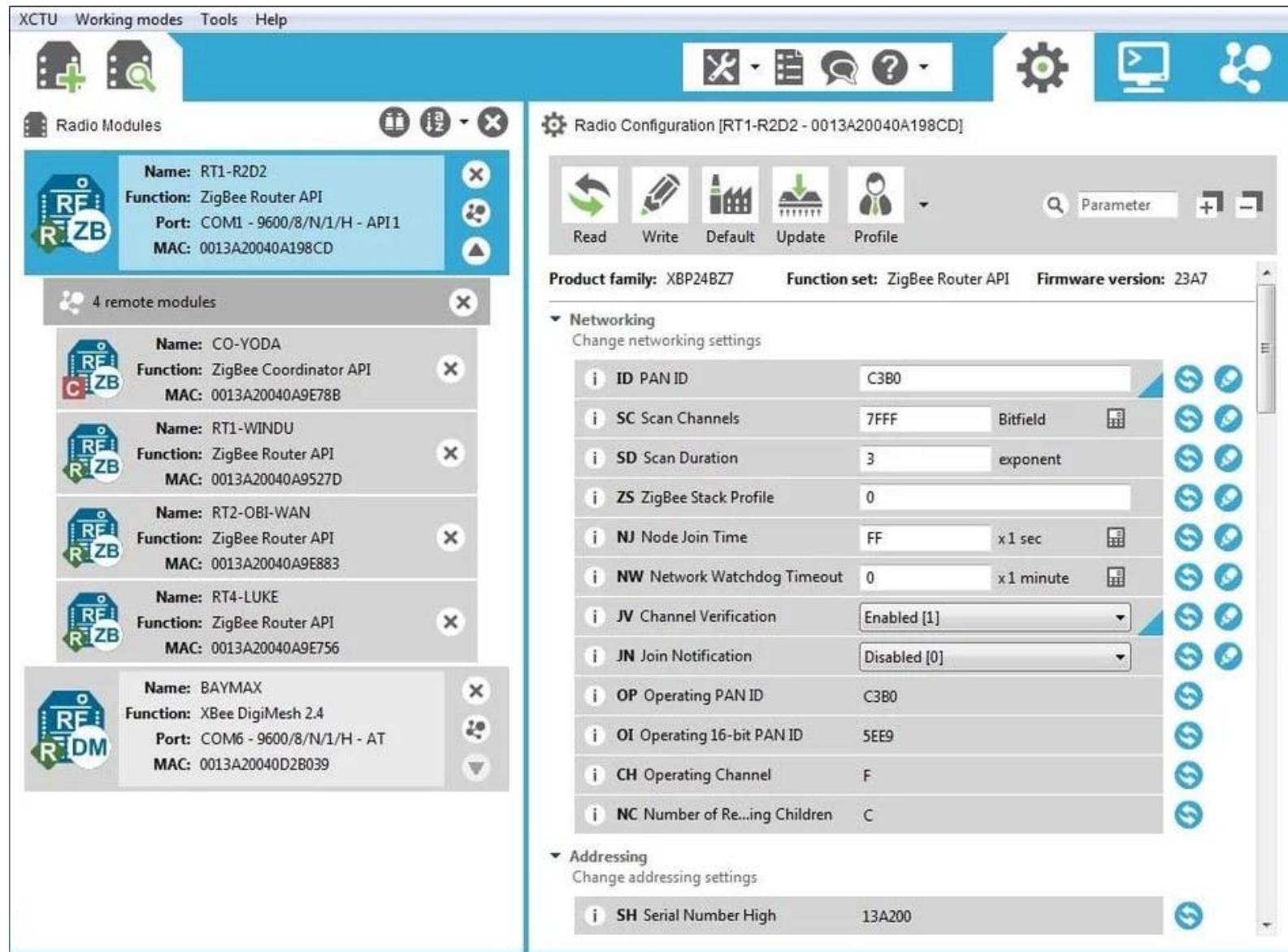
Interfacing XBee With NodeMCU (Receiving Side)

- Connect **Gnd(pin10)** of XBee to **GND of NodeMCU**.
- Connect **Vcc (Pin1)** of XBee to **3.3V of NodeMCU**.
- The above-mentioned two connections supply the power to the XBee module as well.
- Then connect **Dout (pin2) of XBee** to **D6 pin of NodeMCU** and **Din (pin3) of XBee** to **D7 pin of NodeMCU** for receiving data.
- As an indication of whether the data is received or not, we've used LED.
 - ❖ For this, connect LED anode to **D2 of NodeMCU** and LED cathode to GND through a 220ohm resistor of NodeMCU.



Configure and testing of XBee devices

- Need XCTU software.
- It's an easy-to-use, free, multi-platform application for RF XBee modules.
- Download the XCTU software and install it as well.
- After that, open the application and connect XBee using a USB to serial converter or an explorer board.
- Check the COM port of the Xbee in the device manager.
- Configure And Update Both Xbee Board.



Serial Communication with Xbee Module (Tx)

- The code in Arduino for Serial communication for Xbee Module can be written as following:

Xbee.ino

```
1 #include "SoftwareSerial.h"
2 SoftwareSerial XBee(2,3);
3 int BUTTON = 5;
4 boolean toggle = false;
5 //this variable keeps track of alternative click of the button
6
7 void setup() {
8
9     Serial.begin(9600);
10    pinMode(BUTTON, INPUT_PULLUP);
11    XBee.begin(9600);
12 }
13
```

Serial Communication with Xbee Module (Tx)

Xbee.ino

```
16 void loop() {  
17     //When button is pressed (GPIO pulled low) send 1  
18     if (digitalRead(BUTTON) == LOW && !toggle) {  
19         Serial.println("Turn on LED");  
20         toggle = true;  
21         XBee.write('1');  
22         delay(1000); }  
23  
24     //When button is pressed second time (GPIO pulled low) send 0  
25     else if (digitalRead(BUTTON) == LOW && toggle) {  
26         Serial.println("Turn off LED");  
27         toggle = false;  
28         XBee.write('0');  
29         delay(1000); }  
30 }
```

Serial Communication with Xbee Module (Rx)

- The code in Arduino for Serial communication for Xbee Module can be written as following:

Xbee.ino

```
1 #include<SoftwareSerial.h>
2 int led = 2;
3 int received = 0;
4 int i;
5
6 //For communicating with zigbee
7 SoftwareSerial zigbee(13,12);
8 void setup() {
9
10 Serial.begin(9600);
11 zigbee.begin(9600);
12 pinMode(led, OUTPUT);
13 }
```

Serial Communication with Xbee Module(Rx)

- The code in Arduino for Serial communication for Xbee Module can be written as following:

Xbee.ino

```
14 void loop() {  
15     //check if the data is received  
16     if (zigbee.available() > 0) {  
17         received = zigbee.read();  
18         //if the data is 0, turn off the LED  
19         if (received == '0') {  
20             Serial.println("Turning off LED");  
21             digitalWrite(led, LOW); }  
22         //if the data is 1, turn on the LED  
23         else if (received == '1') {  
24             Serial.println("Turning on LED");  
25             digitalWrite(led, HIGH); }  
26     }  
27 }
```

**Thank
You**



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot

✉ bhushan.joshi@darshan.ac.in
☎ +91 8485979997



Unit-5

Domain Specific Applications of IoT



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot

✉ bhushan.joshi@darshan.ac.in
📞 +91 8485979997

Domain Specific applications of IoT

IoT Application Architecture

Home automation, Industry applications, Surveillance applications & Other IoT applications

Introduction to Python And Different IoT tools

Developing IoT application through IoT tools

Developing sensor-based application through embedded system platform

IoT and Cloud computing



IoT Applications

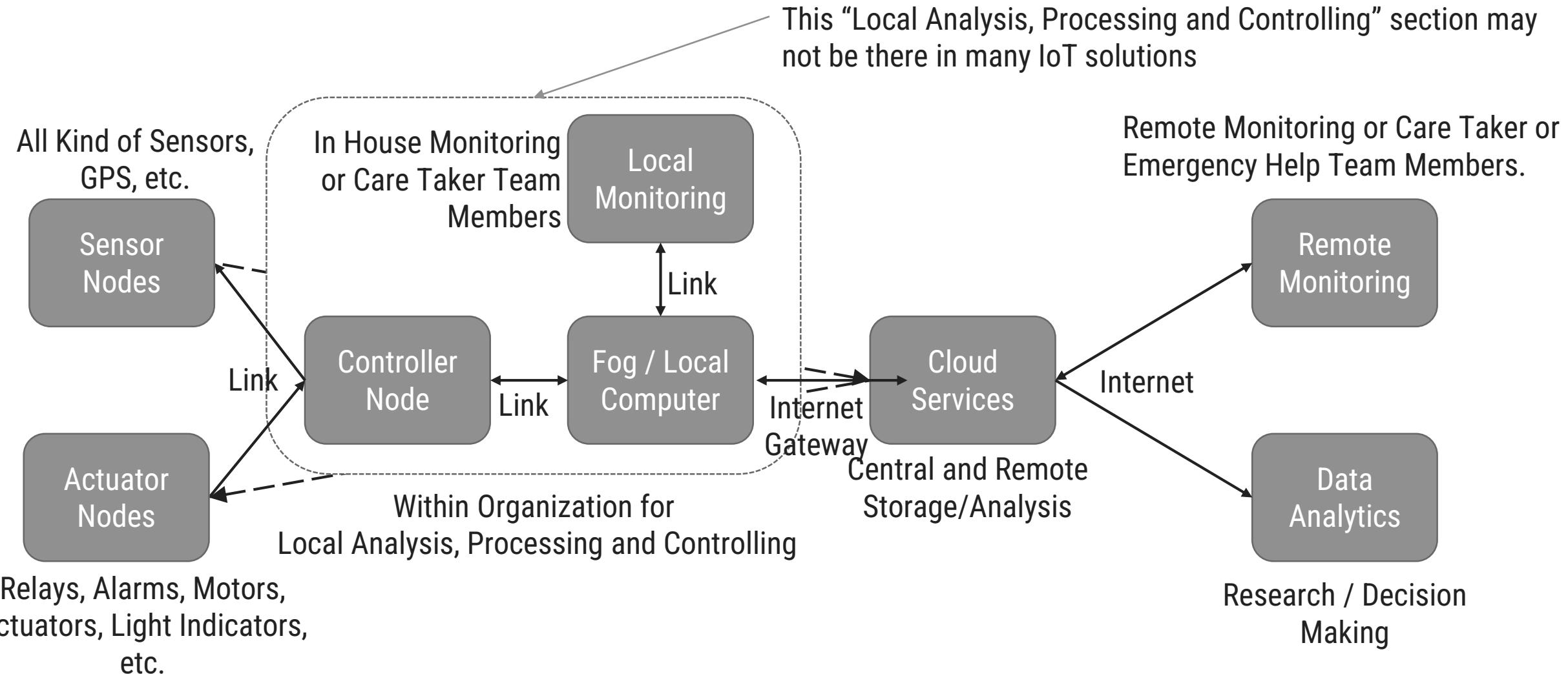
Section - 1

Applications with IoT

- ▶ Using IoT we can build many applications but here, few IoT Applications will be discussed.
- ▶ We will cover following topics of the IoT applications.
 - **Overview** – General Practice
 - **Importance** – Why we need?
 - **Requirements** – Materials and Facility
 - **Architecture** – Functional Block Diagram
 - **How it works** – General Working of IoT Applications
- ▶ Before we explore them one by one, let's understand the common architecture of the IoT applications.

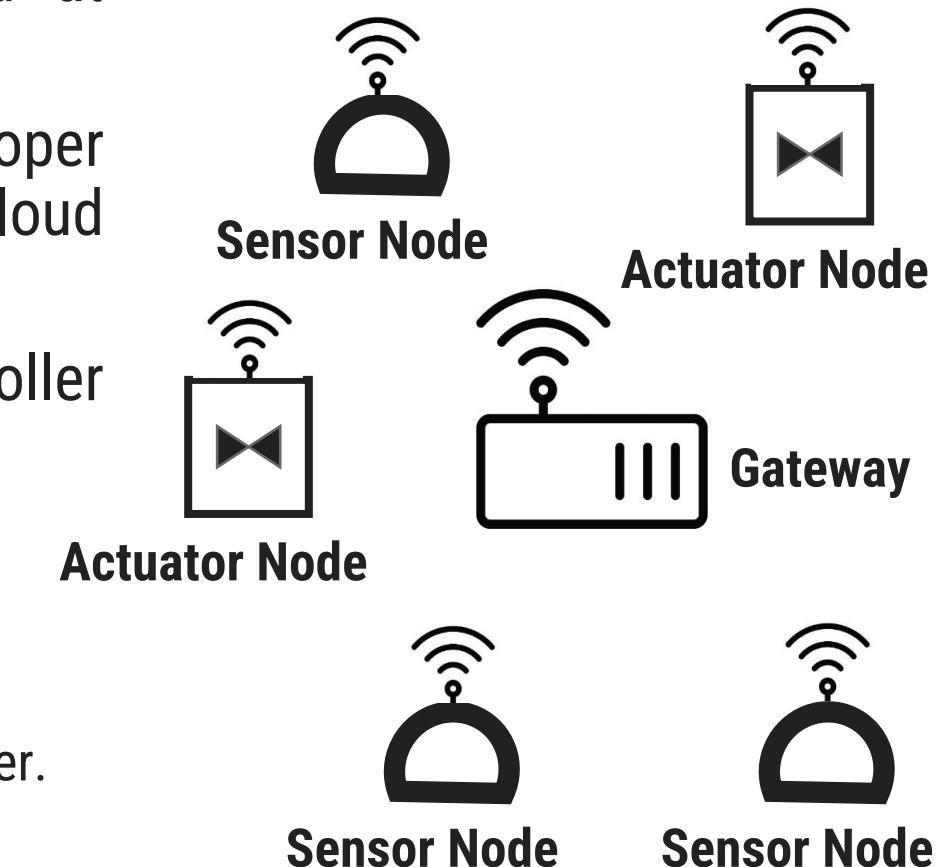


Common Architecture for IoT Application



How it works?

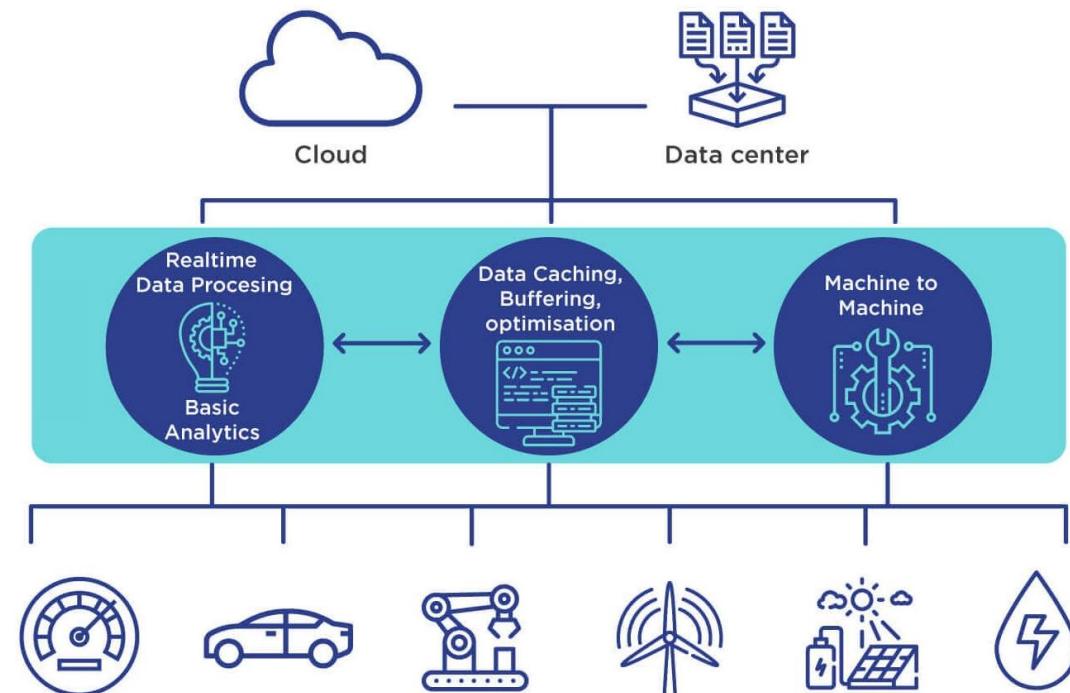
- ▶ All the sensor and actuator nodes should be placed at appropriate place.
- ▶ The sensor and actuator nodes should have proper connectivity with controller nodes or direct to the cloud server.
- ▶ The sensor nodes sends the sensed data to the controller nodes or directly to the cloud server.
- ▶ The controller node:
 - Collects the data from all the sensors.
 - Apply some business logic on it as per the requirements
 - Send it to the Fog or local computer, or directly to the cloud server.



How it works?

Fog Computing

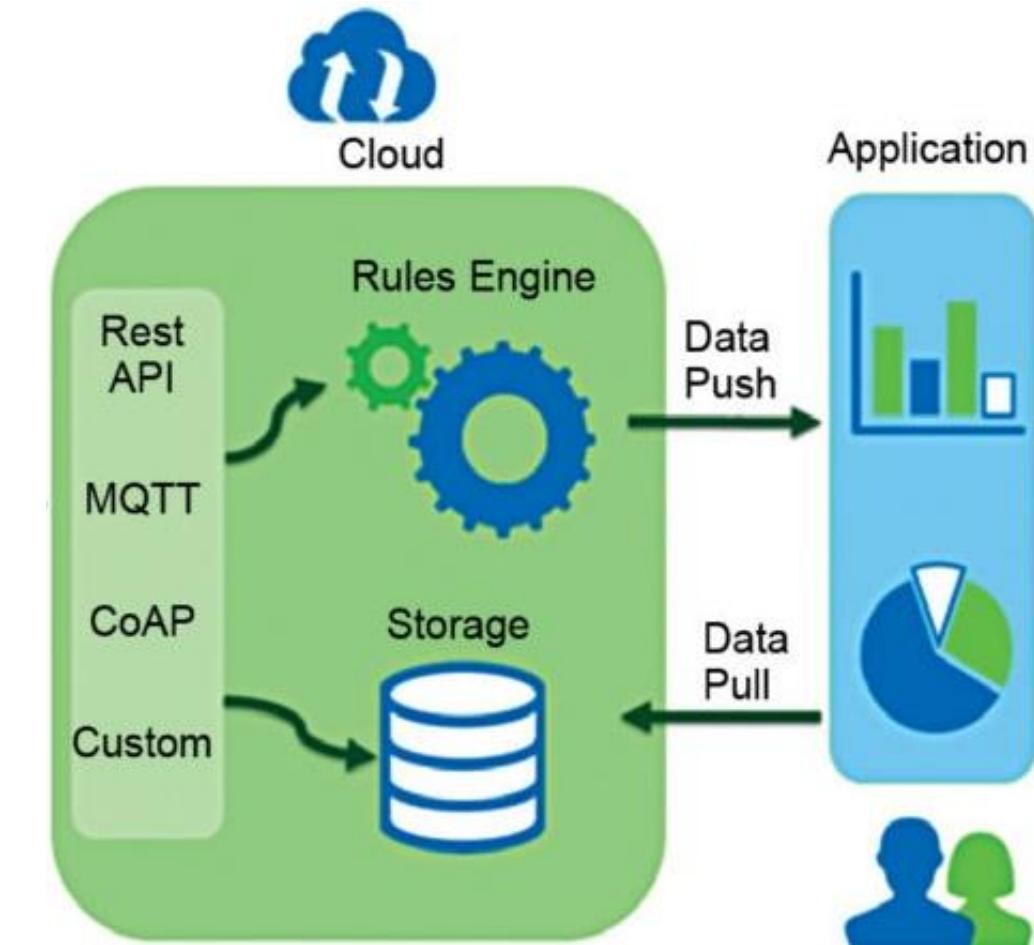
- ▶ All the **controller** nodes should have **connected with** the **Fog or local computer network**, or **directly** to the **cloud server** via Internet.
- ▶ **Fog or local computer**
 - Receives all the **data from** all the **controller nodes**,
 - Execute some **business logic** and
 - Do **analysis** on the collected data, and
 - Send the filtered limited **data to** the **cloud computer** as well as
 - Send the **alert messages** to the **local team**.
- ▶ The **Fog computer** should have **Internet connectivity** for the **cloud connectivity**.

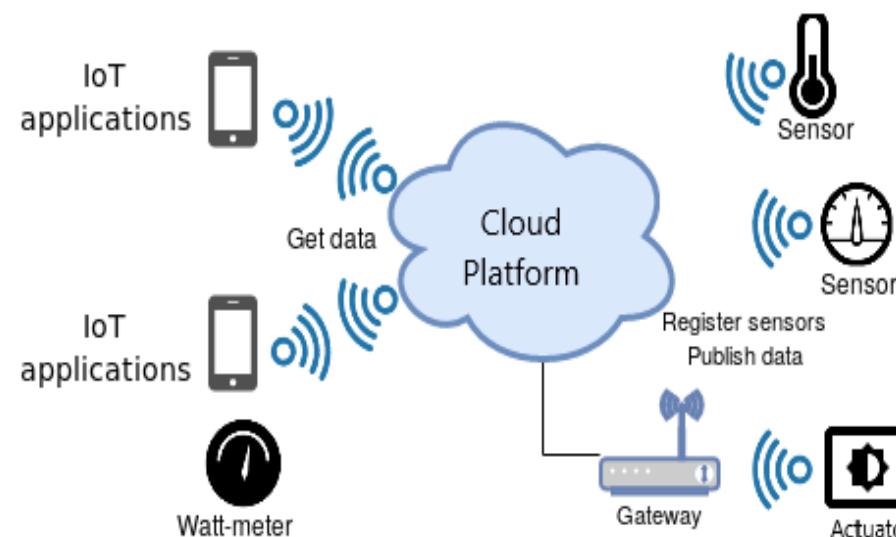


How it works?

Cloud Computing

- ▶ All the in House Monitoring team members can access status of all the sensors from the Fog/Local computer.
- ▶ The cloud
 - Stores all the data received from the Fog/Local computer or directly from the sensors,
 - Execute some business logic and
 - Do analysis on the stored data.
- ▶ All those who have rights to access the cloud data can get desire information from it, like
 - Remote monitoring team members,
 - Data analytics





- ▶ The care taker and data analytics teams may execute corrective action based on the analyzed data.
- ▶ They may
 - Send command to the local computer or to the actuators directly and
 - Broadcast alert messages to the concern persons.
- ▶ All the actuator nodes receives command from the cloud directly or from the controller or local computer.
- ▶ The “Local Analysis, Processing and Controlling” section may not be there in many IoT solutions.



Health Care Application

Section - 2

- ▶ The **healthcare** sector consists of **medical** and **related goods** and services.
- ▶ Healthcare sector provides medical services for maintaining or improving health via
 - Prevention
 - Diagnosis
 - Treatment
 - Recovery or cure of disease, illness, injury, and other physical and mental harms in people.
- ▶ Healthcare sector is much diversified and is full of opportunities in every segment.



- ▶ Healthcare is **delivered** by health professionals.
 - Medicine
 - Dentistry
 - Pharmacy
 - Nursing
 - Psychology
 - And other health professions are all part of health care.
- ▶ Healthcare has become one of India's largest sector in terms of revenue and employment.



- ▶ Before Internet of Things, **patients' interactions with doctors** were limited to visits.
- ▶ There was no way **doctors** or **hospitals** could monitor patients' health continuously.
- ▶ **IoT enabled devices** have made **remote monitoring** in the healthcare sector **possible**.
- ▶ **IoT helps**
 - To keep patients **safe** and **healthy**,
 - Empowering **physicians** to deliver superlative care.



- ▶ Remote monitoring of patient's health helps in
 - Reducing the length of hospital stay and
 - Prevents re-admissions.
- ▶ IoT has applications in healthcare that benefit to
 - Patients
 - Families
 - Physicians
 - Hospitals and
 - Insurance



Healthcare Application with IoT

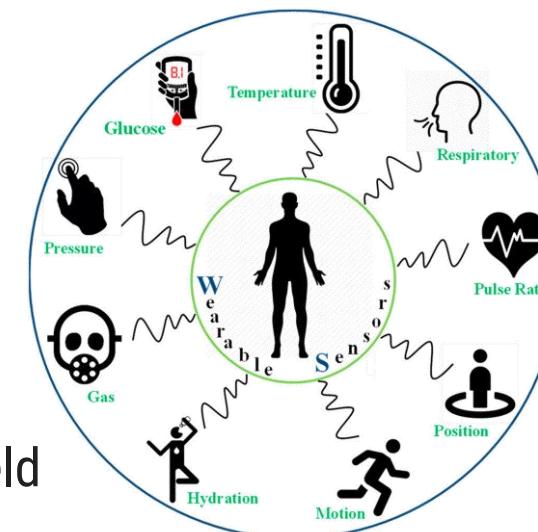
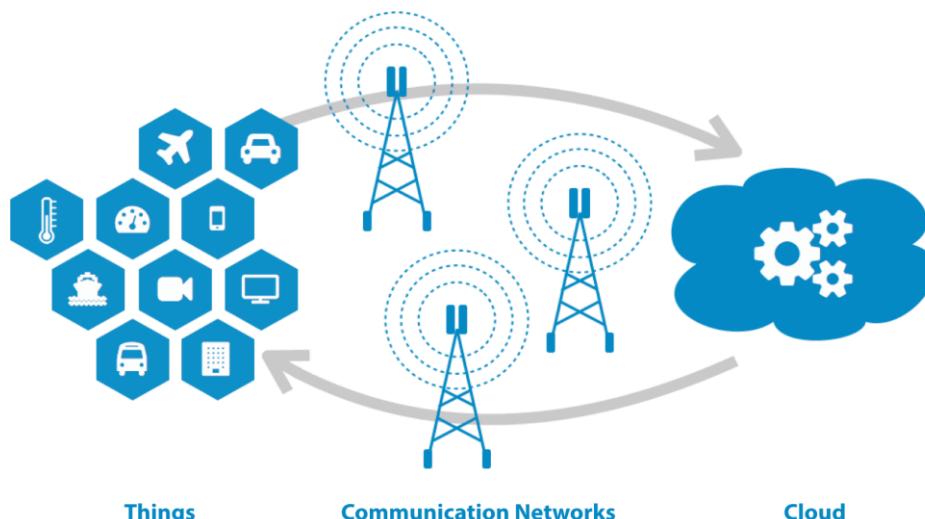
Requirements for IoT Setup

► Sensors

- Heartbeat Sensor
- Oxygen Level Sensor
- Glucose Level Sensor

► Controller

- ESP8266 Node MCU or
- Arduino Uno with Wi-Fi or Network Shield



► For Monitoring and Analysis

- PC
- Mobile
- Tablet



► For Fog Computing

- Raspberry Pi or
- PC

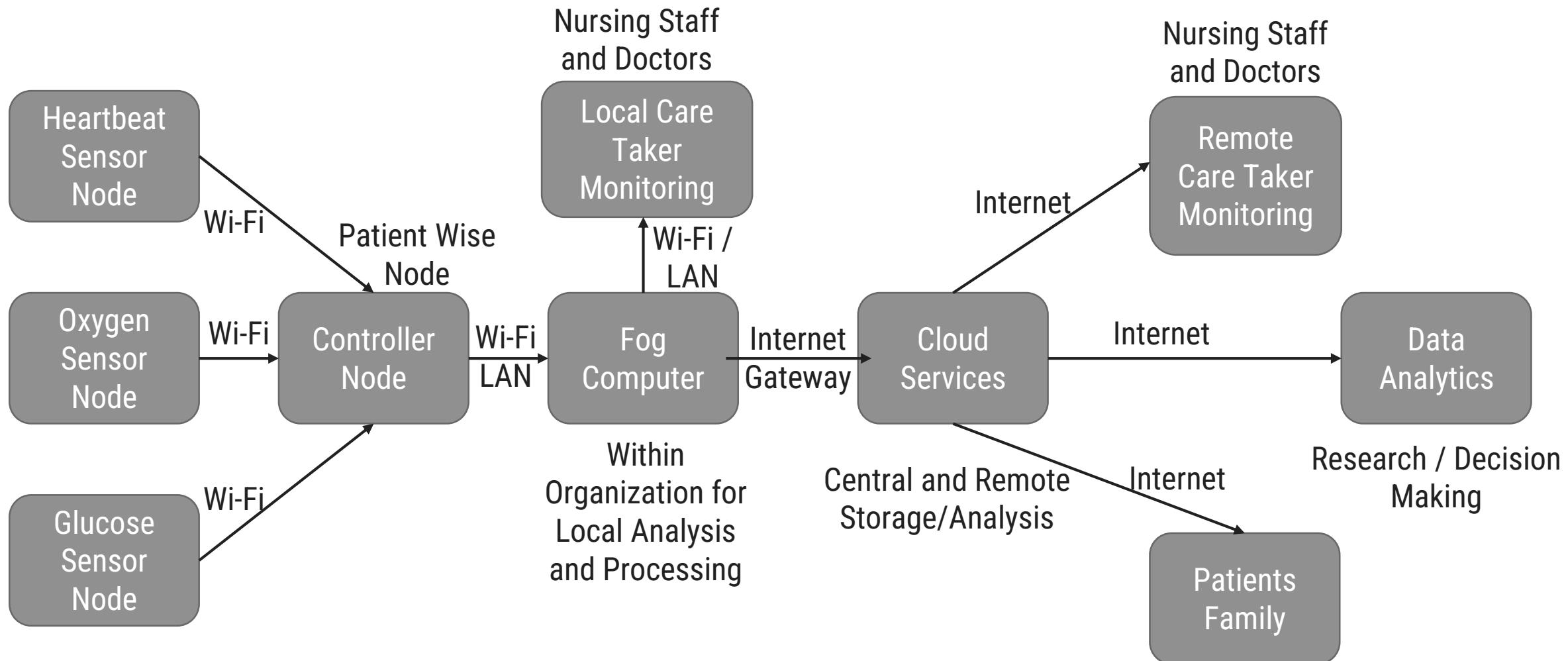
► LAN and WAN Connectivity

- Router and Switches
- Wireless Access Point

► Cloud Services

Healthcare Application with IoT

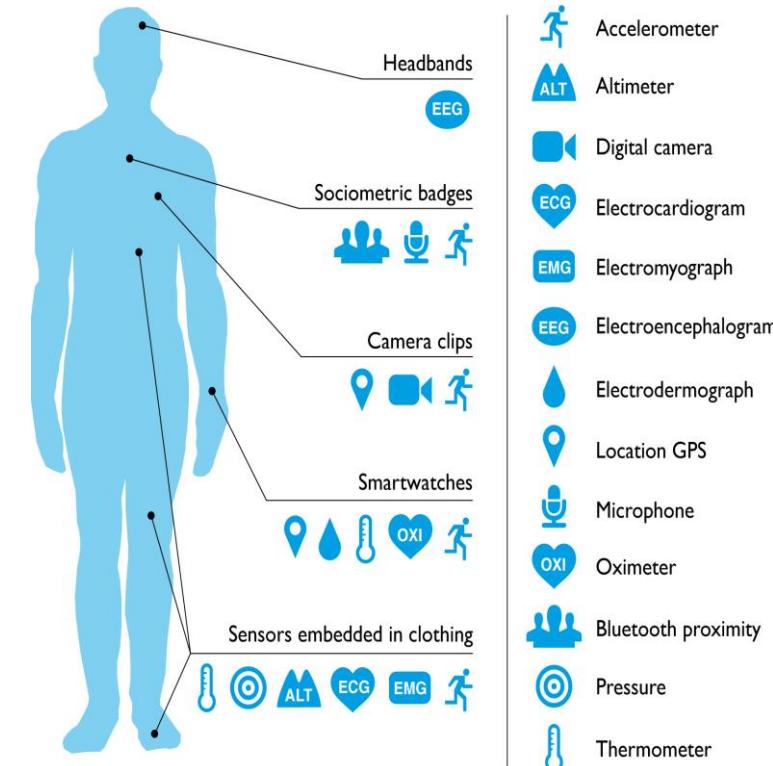
Architecture



Healthcare Application with IoT

How it works?

- ▶ All the **sensor** nodes should be **placed on patient body**, either direct or **using wearables**.
- ▶ The sensor nodes should have Wi-Fi connectivity with a controller node of the patient.
- ▶ The sensor nodes sends the sensed data to the controller node.
- ▶ The controller node collects
 - The data from all the sensors,
 - Apply some business logic on it as per the requirements and
 - Send it to Fog computer.
- ▶ All the controller nodes of all the patients should have connected with the Fog computer network.



Healthcare Application with IoT

How it works?

► Fog computer

- Receives all the data from all the controller nodes
- Execute some business logic and
- Do analysis on the collected data, and
- Send the filtered limited data to the cloud computer as well as
- Send the alert messages to the care taker team (nursing staff and doctors).

► The Fog computer should have Internet connectivity for the cloud communication.

► All the local care taker team members can be accessed health status of all the patients from the Fog computer.



- ▶ The cloud
 - Stores all the data received from the Fog
 - Execute some business logic
 - Do analysis on the stored data
- ▶ All **those who have rights** to access the **cloud** data can **get health records of the patients**, like
 - Remote care taker team members
 - Data analytics team
 - **Patient family** members
- ▶ The **care taker** and data analytics teams may **execute corrective action** based on the data and alert messages.



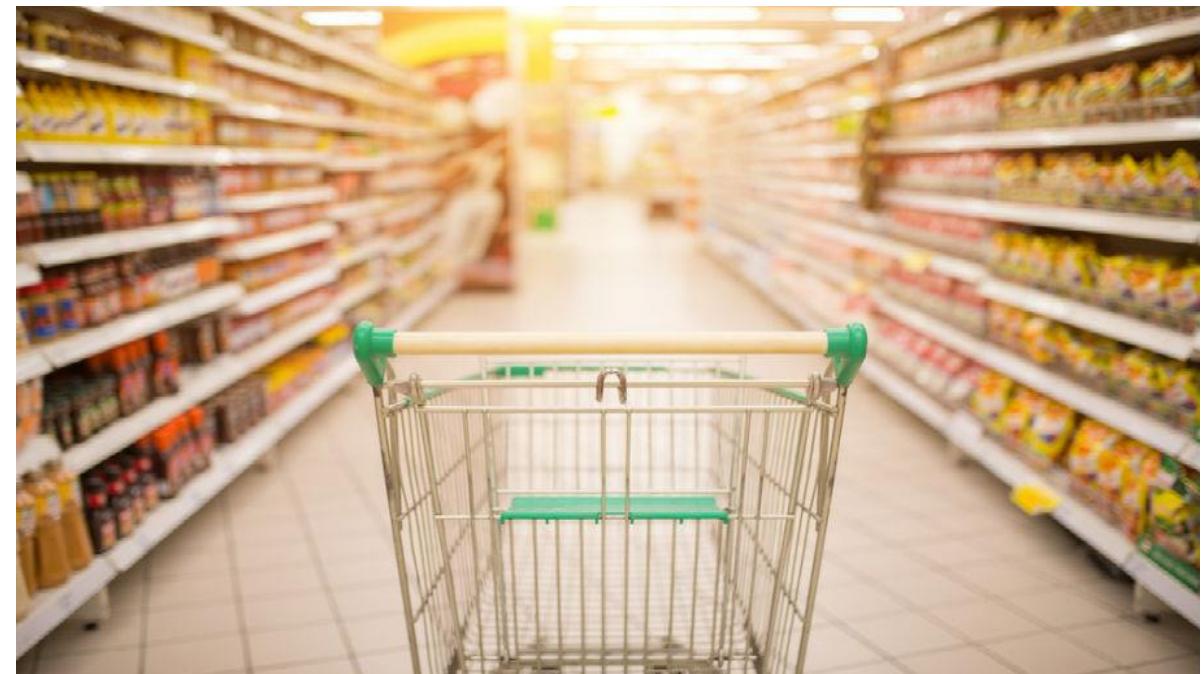
- ▶ IoT has changed people's lives by enabling constant tracking of health conditions.
- ▶ Wearables connected devices like blood pressure and heart rate monitoring cuffs, glucometer etc. give patients attention.
- ▶ On any changes in the routine activities of a person, alert message sends to family members and health service providers.
- ▶ IoT enables healthcare professionals to be more watchful.
- ▶ Data collected from IoT devices can help physicians identify the best treatment process.
- ▶ IoT devices tagged with sensors are used for tracking real time location of medical equipment like wheelchairs, nebulizers, oxygen pumps and other monitoring equipment.
- ▶ IoT-enabled hygiene monitoring devices help in preventing patients from getting infected.
- ▶ It also help in asset management like pharmacy inventory control, and environmental monitoring, for instance, checking refrigerator temperature.



Retail Applications

Section - 3

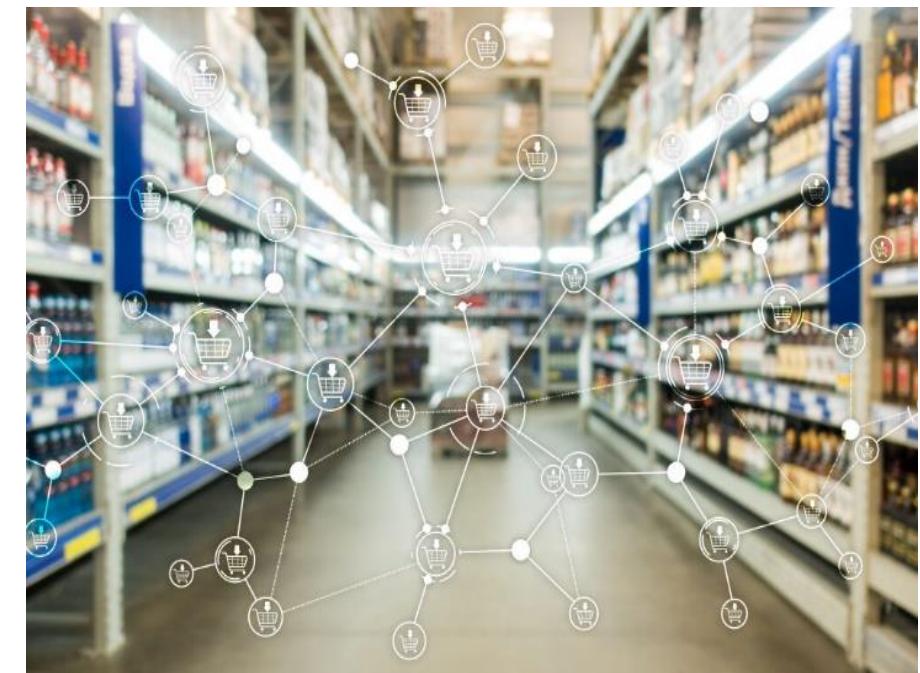
- ▶ Retail is the process of selling consumer goods or services to end-users for earning a profit.
- ▶ The retail sectors can be divided into
 - Food
 - Clothing & Textiles
 - Consumer Durables
 - Footwear
 - Jewelry
 - FMCG



IoT Application in Retail

Importance

- ▶ In the retail industry customer plays key roll
- ▶ The retail industry is rapidly transforming with IoT solutions.
- ▶ IoT is taking the center stage in the sector.
- ▶ Inventory management has been an **expensive** and **tedious process**.
- ▶ IoT system automatically monitors inventory.
- ▶ Send alert messages to managers if a certain item is **running low** or **going to expire soon**.
- ▶ IoT devices are also helpful for avoiding
 - **Oversupply**
 - **Shortage of goods** and
 - **Thefts in stores**



- ▶ Customers are notified about **discounts** and **offers in real-time**.
- ▶ It give data about **customer behavior** and **routes**.
- ▶ IoT helps to increase
 - Customer loyalty
 - Boost sales
 - Offer a personalized experience
 - Helps to maintain Stock



► Sensors

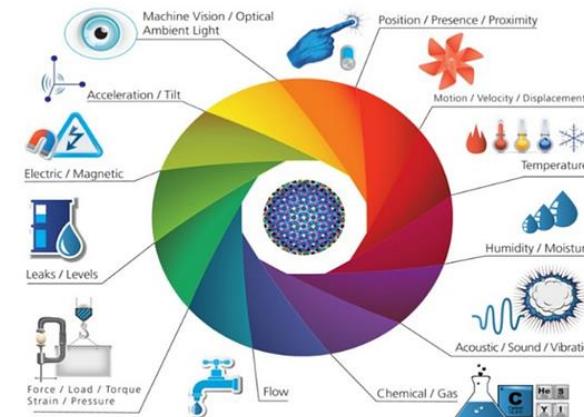
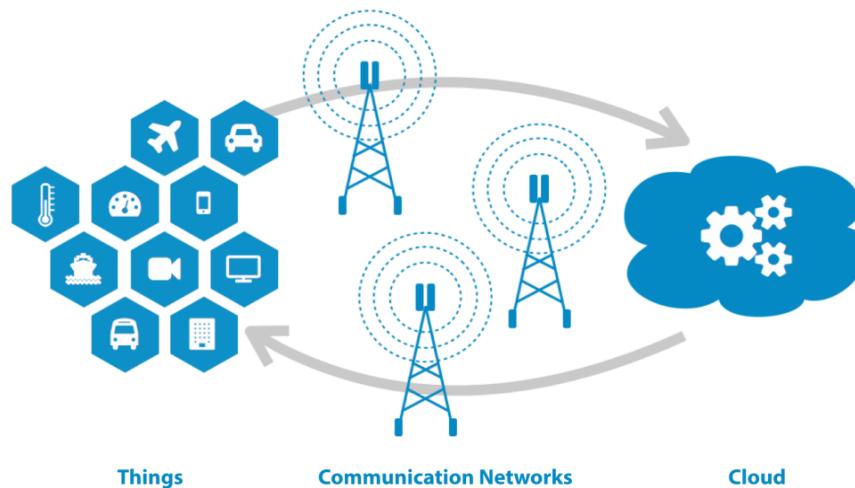
- Touch Sensors
- Gas Sensors
- Fire Sensors
- IR Sensors

► Actuators and Tags

- Alarm
- Water Sprinkler
- Beacons
- RFID Tag and RFID Reader

► Controller

- ESP32 MCU or
- Arduino Uno with Wi-Fi or Network Shield



► For Monitoring and Analysis

- PC
- Mobile
- Tablet



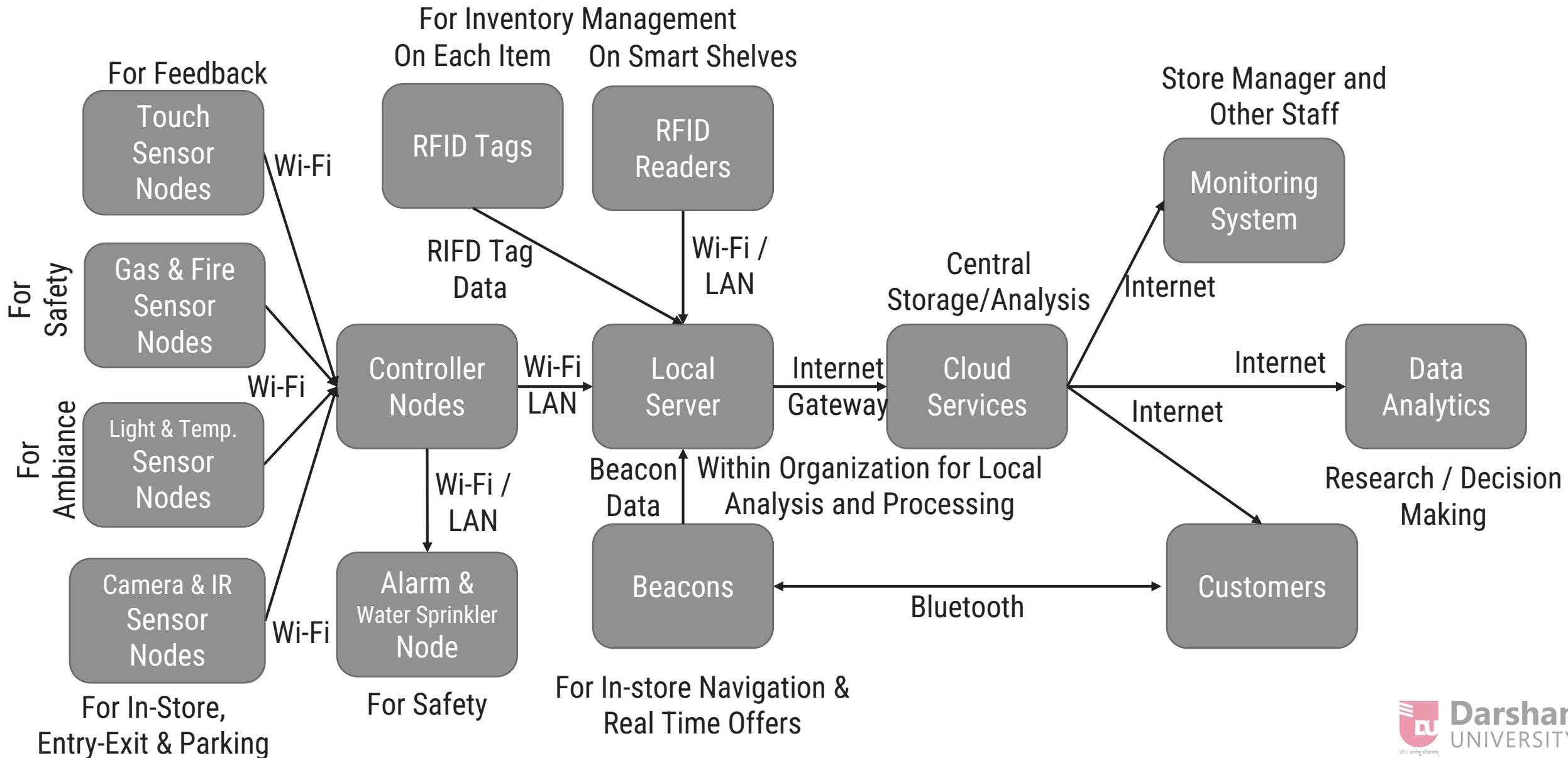
► LAN and WAN Connectivity

- Router and Switches
- Wireless Access Point

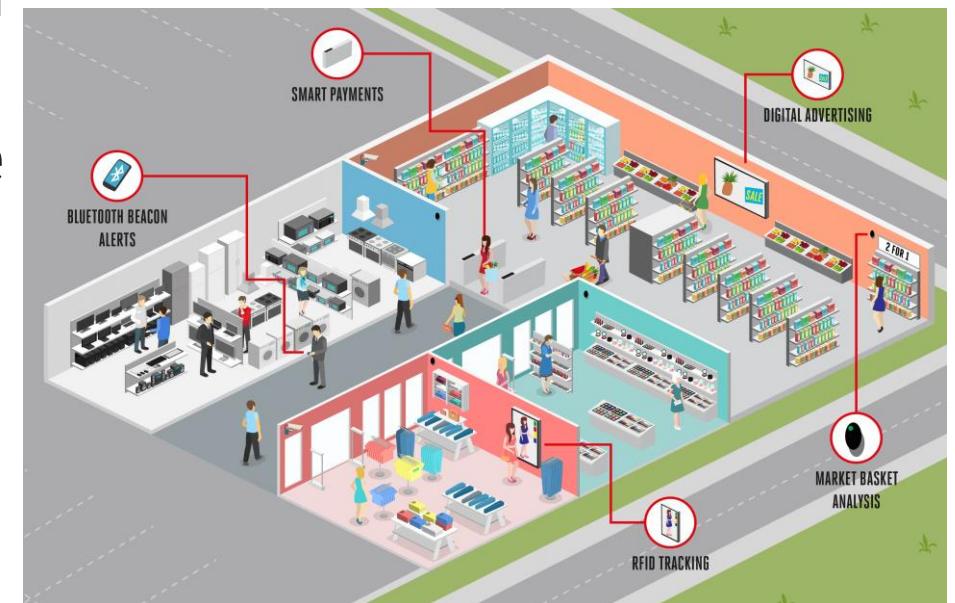
► Cloud Services

IoT Application in Retail

Architecture



- ▶ All the **sensor nodes** and **Beacons** should be placed at various places in the store, washroom and parking area.
- ▶ The sensor nodes should have Wi-Fi connectivity with controller nodes.
- ▶ The sensor nodes sends the sensed data to the controller nodes.
 - Collects the data from all the sensors,
 - Apply some business logic on it as per the requirements
- ▶ The controller nodes



- ▶ RFID readers should be placed at items display shelves to detect item presence in the shelves for the stock management.
- ▶ All the controller nodes and RFID readers should have connected with the Local Area Network.
- ▶ RFID tags should be attached with each and every item which are likely to be monitored.
- ▶ Using the RFID tags one can monitor location and stock of the item.
- ▶ RFID tags is also useful for auto billing and queue less exit.



- ▶ Touch sensors are used for customers' feedback about ambiance and cleanliness at various places like washroom, trial room etc.
- ▶ Light and Temperature sensors are used for ambiance monitoring which leads to energy saving and customer comfort.
- ▶ Camera and IR sensors are placed for monitoring people density at various places, person in-out ratio for store and the parking area.
- ▶ Gas sensors are used for monitoring various gases levels in the store.
- ▶ Fire sensor are used for detecting fire in the store and send the alert signal to the Alarm & Water Sprinkler



- ▶ An **alarm** system and the **water sprinkling** system are used **for fire safety**, should have connected with a controller node
- ▶ Local Server
 - Receives all the data from all the controller nodes,
 - Execute some business logic and
 - Do analysis on the collected data, and
 - Send the essential data to the cloud computer.
- ▶ The server computer should have Internet connectivity for the cloud communication.
- ▶ The cloud
 - Stores all the data received from the local server,
 - Execute some business logic and
 - Do analysis on the stored data



- ▶ All those who have rights to **access the cloud data** can **get desire information** and **alerts**, like
 - Store manager
 - Staff members
 - Data analytics team
 - customers
- ▶ The **store manager** and data **analytics teams** may **execute corrective action based on the data**.



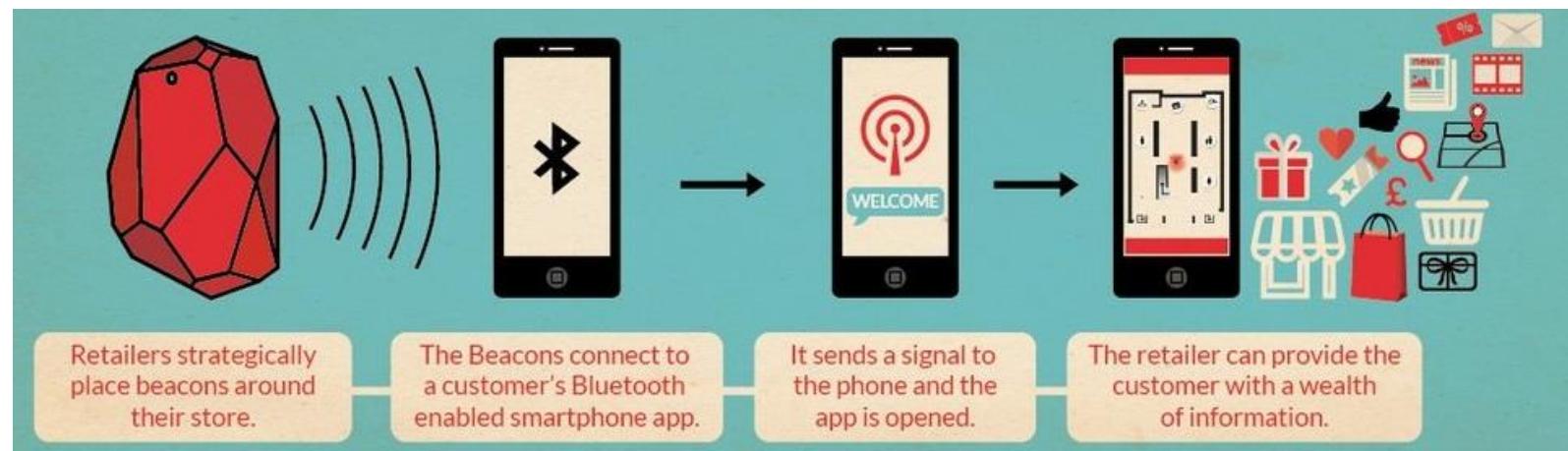
► Beacons, a Bluetooth Low Energy (BLE) device

- Automatically connect with customers' smartphones when they entered in the coverage area.
- Send push notifications directly to the customers.

► Beacons can be installed at various places

- For customer in-store navigation,
- To know foot fall in particular area of the store, and
- Collecting data about customer behavior and routes.

► Customers are notified about discounts and offers in real-time using the beacons system.



- ▶ IoT will dramatically transform and innovate the retail industry in the coming years.
- ▶ The integration of IoT solutions will enable retail companies,
 - To create successful marketing campaigns based on customer behavior,
 - Deliver high-quality services,
 - Improve inventory management, and
 - Reduce operational costs.



Driver Assistance

Section - 4

Driver Assistance Application with IoT

Overview

- ▶ Vehicle manufacturing companies are trying to make vehicles more comfortable and safe for their passengers.
- ▶ They are experimenting to design a system that could monitor and alert the user suffering from highway hypnosis or white line fever.
- ▶ As well as they need an IoT based system to prevent accidents caused by drowsy driver.



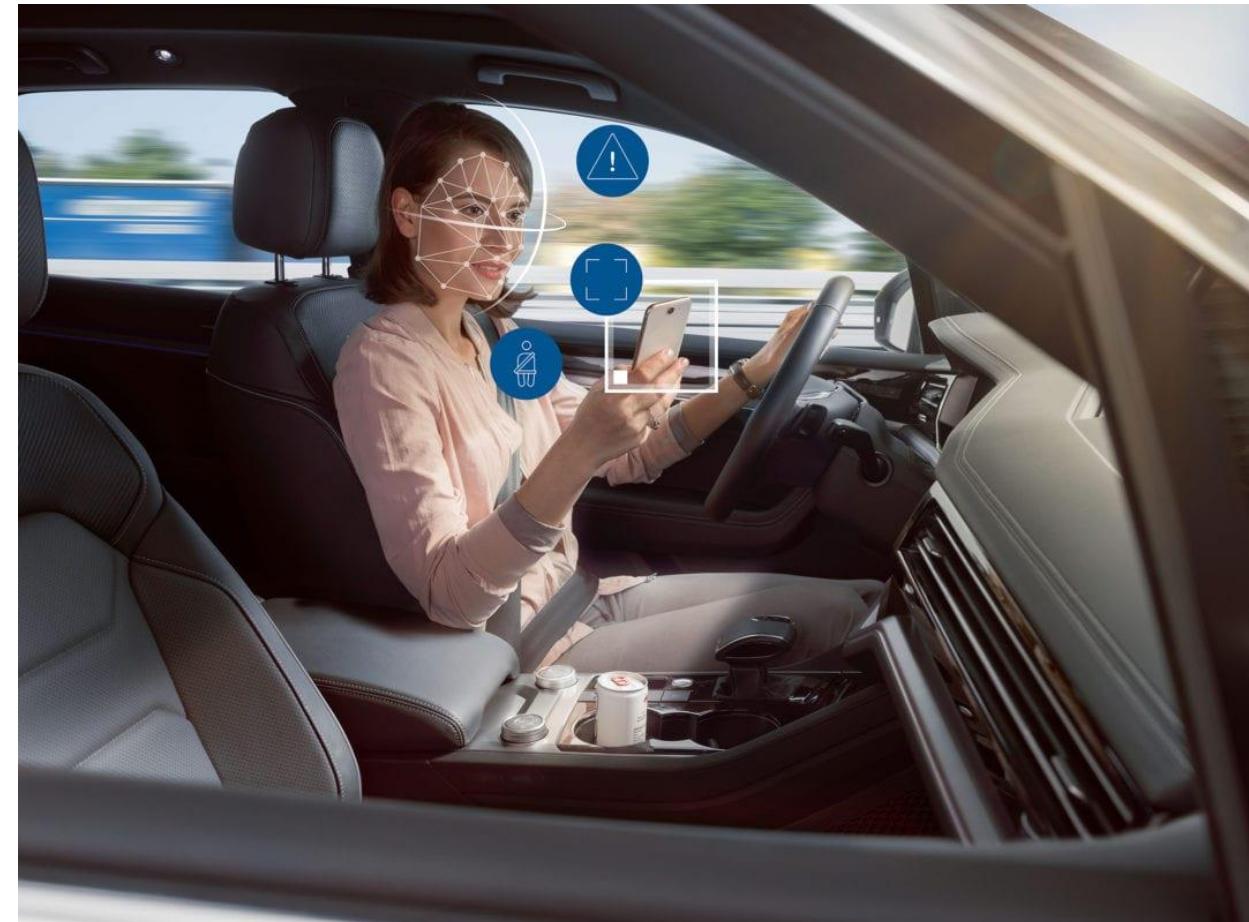
Driver Assistance Application with IoT

- ▶ Driver's **drowsiness** is one of the **major factor** is reported in **road accidents**.
- ▶ Drowsiness is a kind of **sleeping sensation** which can lead for closing eyes.
- ▶ Drowsiness **make** one **physically inactive** and **insensitive** to surroundings.
- ▶ So we **need to monitor** driver's **consciousness** along with his **acceleration pattern** and **steering angle**.
- ▶ If any **deviation** is **detected**, a system should be there to **alert** the **driver**.
- ▶ A process of detecting the deviation and drowsiness of the driver should be developed



Driver Assistance Application with IoT

- ▶ Using Image Processing and Machine Learning (ML) technology a system can be developed.
- ▶ The system detect whether the driver has deviated or not, by monitoring the driver's
 - Facial movements
 - Eye Aspects Ration (EAR) for drowsiness
 - Acceleration pattern
 - Change in the steering angle



Driver Assistance Application with IoT

Requirements for IoT Setup

► Sensors

- Ultrasonic Sensor

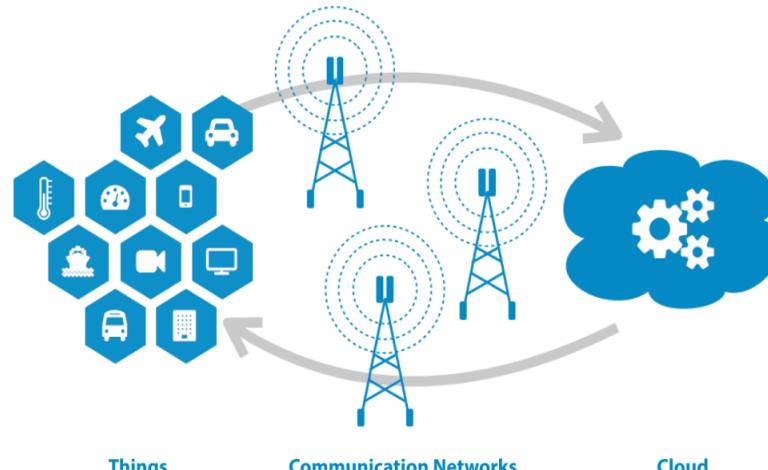
► Cameras

► On Board Diagnostic (OBD)

Access

► Controlling

- Raspberry Pi



- WAN Connectivity
- Cloud Services

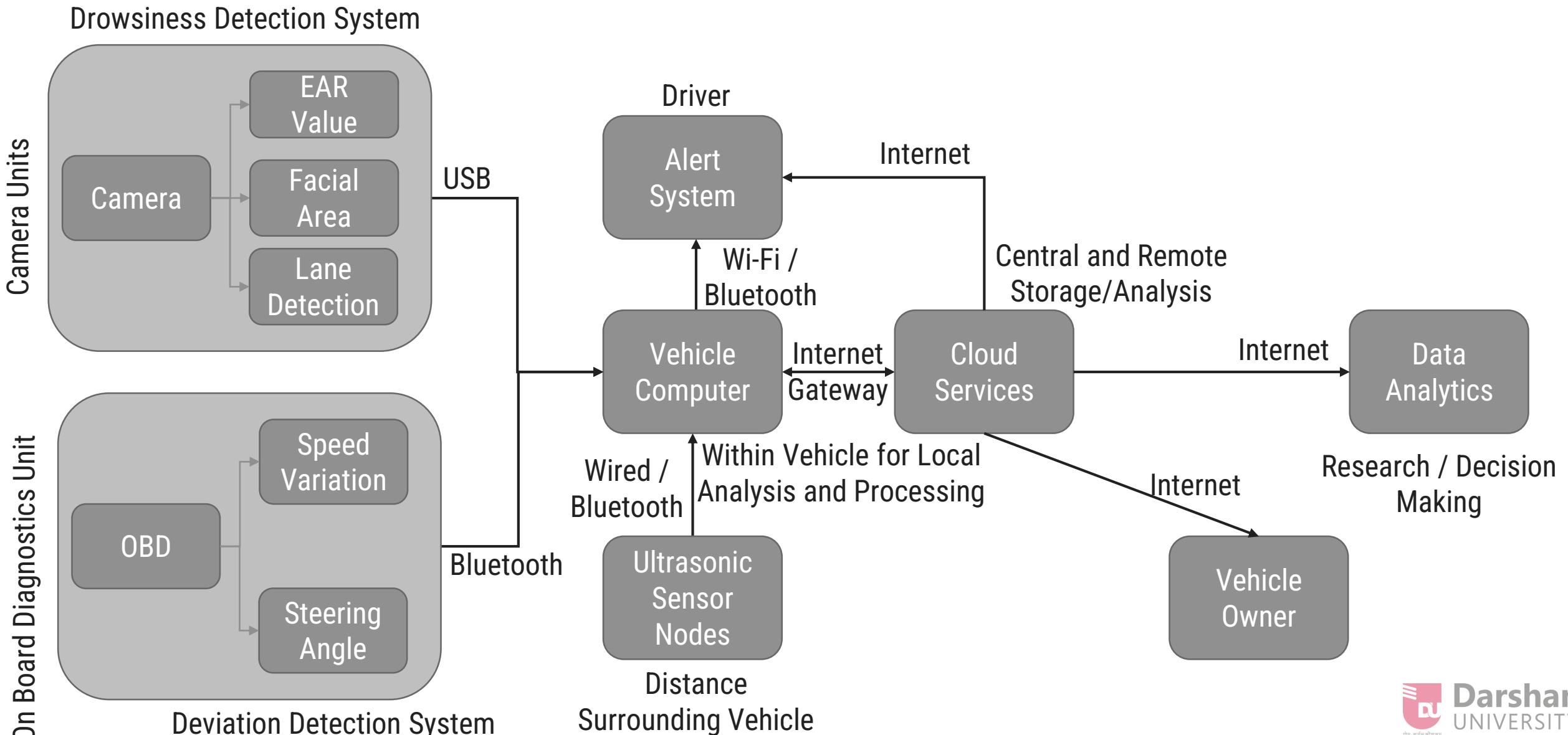
► For Monitoring and Analysis

- PC
- Mobile
- Tablet



Driver Assistance Application with IoT

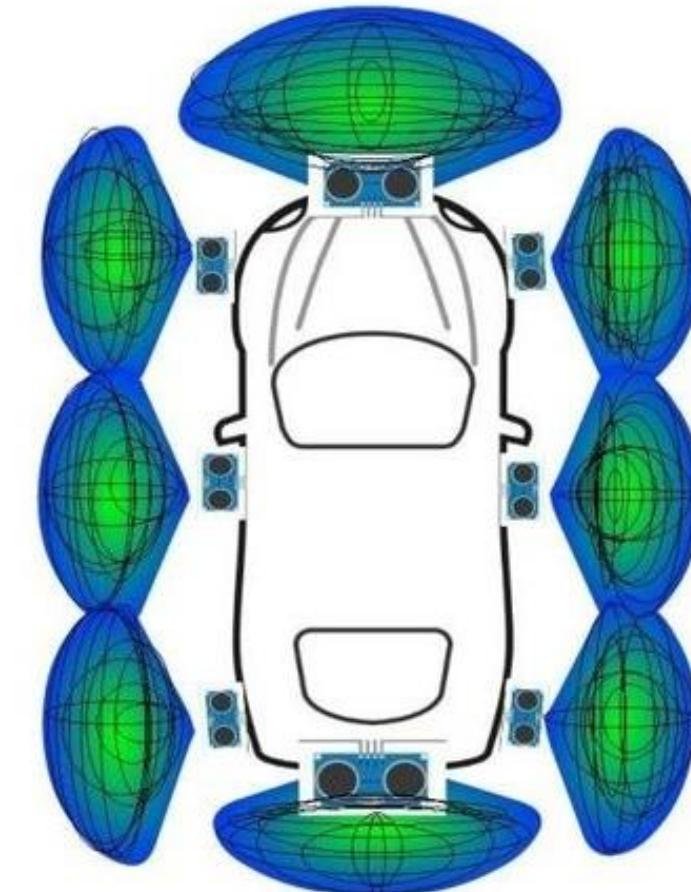
Architecture



Driver Assistance Application with IoT

How it works?

- ▶ Camera units should be placed on board for capture the face of driver and road view.
- ▶ The camera units should have USB connectivity with vehicle computer.
- ▶ The camera units sends the sensed data to the vehicle computer.
- ▶ The ultrasonic sensor nodes should be placed at various place on outside the vehicle for detecting surrounding distance.
- ▶ All the sensor nodes should have Wired / Bluetooth connectivity with the Vehicle Computer.



Driver Assistance Application with IoT

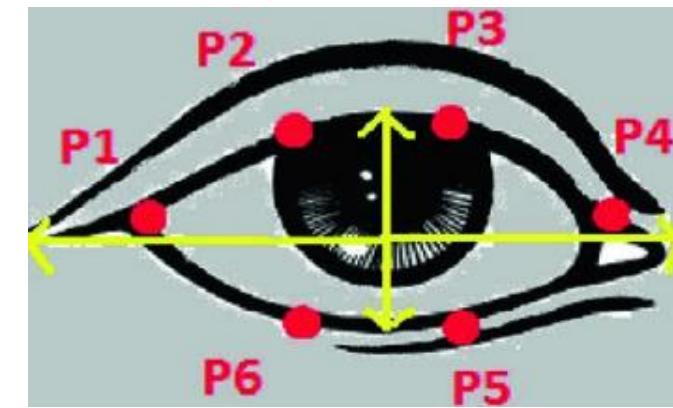
- ▶ The OBD unit of the vehicle should have Bluetooth connectivity with the vehicle computer
- ▶ The vehicle computer
 - Collects the data from the camera units, OBD unit and the sensor nodes, and
 - Apply some business logic,
 - Do analysis on the collected data on it as per the requirements and
 - Send the filtered limited data to the cloud computer as well as
 - Send the alert messages to the driver.
- ▶ The vehicle computer should have Internet connectivity for the cloud communication.



Driver Assistance Application with IoT

How it works?

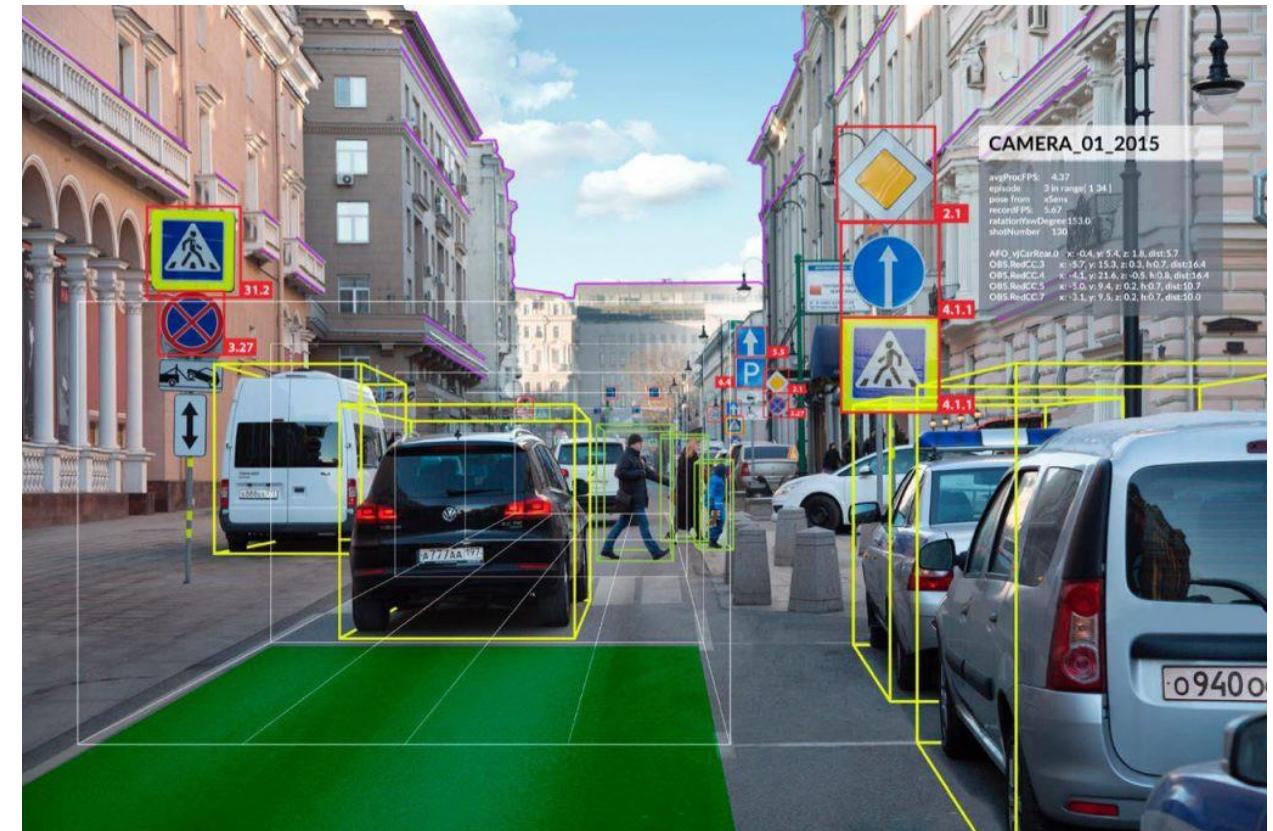
- ▶ The driver side **camera capture** the **drive facial** and **eye movement**.
- ▶ The camera **send** the captured **images** to vehicle computer for **EAR value** and other **calculation**.
- ▶ **EAR (Eye Aspect Ratio)** is measured using following equation.
- ▶
$$\text{EAR} = \frac{(P2 - P6) + (P3 - P5)}{2(P2 - P4)}$$
 - Where
 - $(P2 - P6)$ is distance between points P_2 and P_6
 - $(P3 - P5)$ is distance between points P_3 and P_5
 - $(P1 - P4)$ is distance between points P_1 and P_4
- ▶ Using **image processing** the system **finds** the **eye positions** and **calculate** the above mentioned **distance of the six points**.
- ▶ When the driver **closes** his **eye**, the **EAR value** eventually **drops to zero** and the system **detects drowsiness** of the driver.



Driver Assistance Application with IoT

How it works?

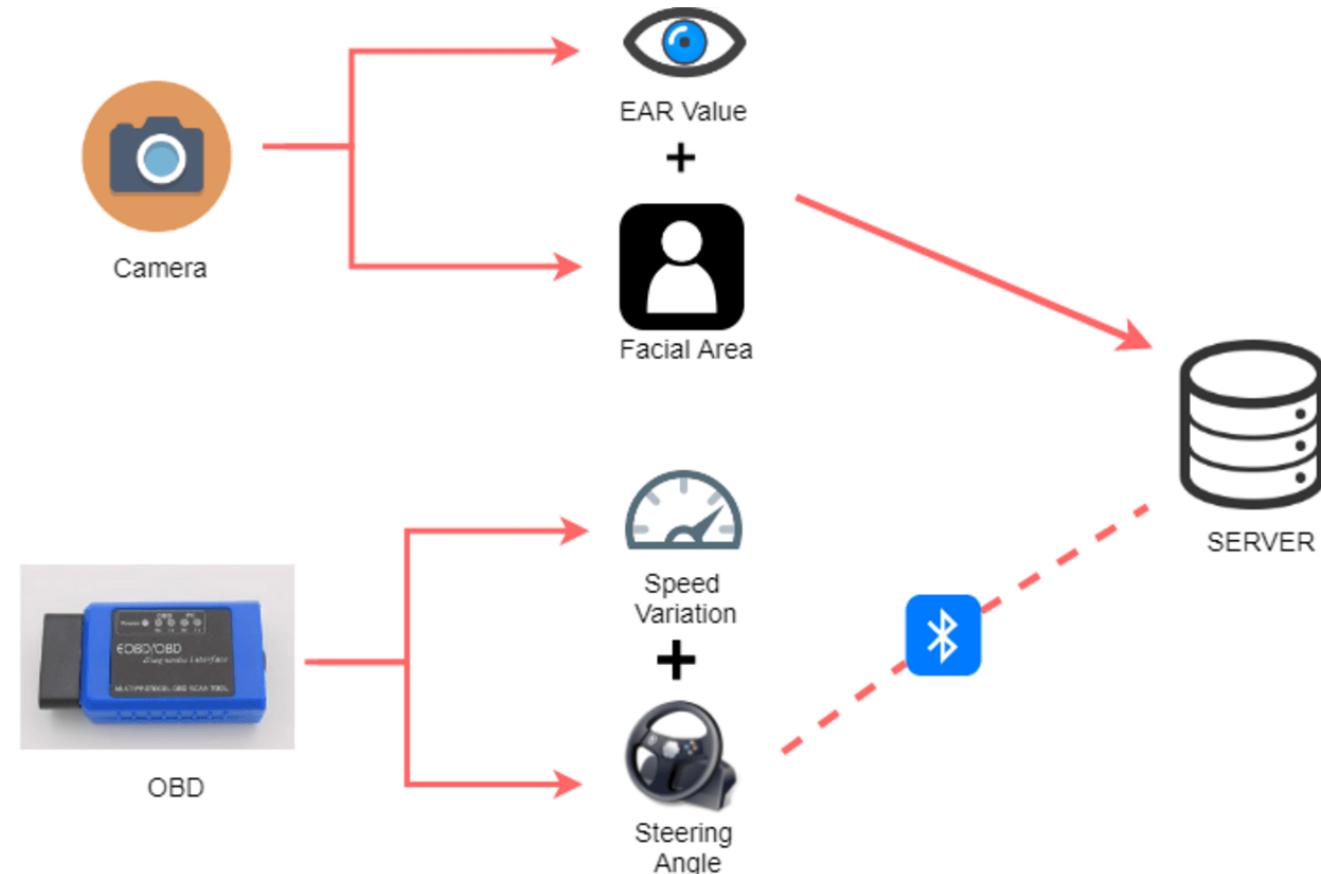
- ▶ The road side camera capture the **front view** images.
- ▶ The capture **images** sends to the vehicle computer for
 - Detecting lane
 - Objects
 - Sign boards
- ▶ Deviation is monitored using OBD (On-Board Diagnostics) device.



Driver Assistance Application with IoT

How it works?

- ▶ OBD can record the pattern of acceleration and steering angles during the drowsiness of the driver.
- ▶ Compare the recorded values with the threshold value and the historical data for the actual deviation.
- ▶ All the calculated data is sent to the data analytics and obtained result, is used to alert the driver.
- ▶ The cloud data also useful for the vehicle owner.



Collision Impact Detection Using IoT

Overview

- ▶ Modern vehicles are equipped with the collision detection mechanism.
- ▶ It blow up airbags when an accident occurs and save lives in major cases.
- ▶ But if the collision is massive then the airbags may not sufficient.
- ▶ In these cases emergency help needed to save the lives.
- ▶ So we need a system that measure the severity impact of the collision and send alert message.



Collision Impact Detection Using IoT

Overview

► The messages contents

- Location of the accident
- Severity impact,
- Fire status,
- Speed of the vehicle,
- Number of passengers,
- Time of the incident, etc.

► Broadcast the messages to all the authorities like

- Ambulance service,
- Hospitals,
- Highway patrolling team,
- Fire brigade,
- Police,
- Family member of the vehicle owner, etc.



Collision Impact Detection Using IoT

Importance

- ▶ If a **single accident** occur in along highway then it can be **managed without the data of severity** of the collision.
- ▶ When **multiple accident** occurs on the highway then the **severity data** is **very important** to save lives of the **very serious victims**.
- ▶ Based on the data and **severity impact**, **all** the **emergency help** will be **reached timely** at the point.
- ▶ Thus, we **need** an **IoT based system** that sense the necessary parameters of the collision.



Collision Impact Detection Using IoT

Requirements for IoT Setup

Sensors

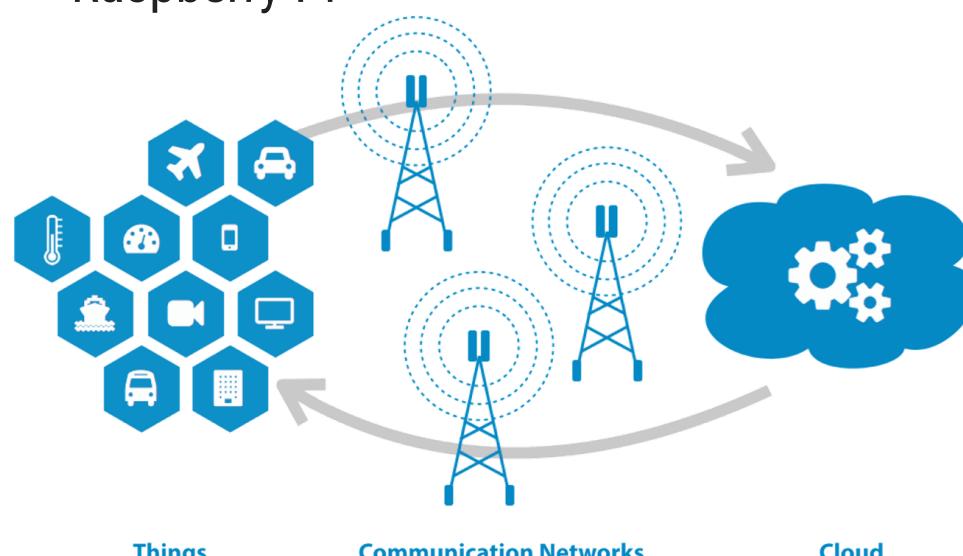
- FSR Sensor
- Fire Sensor
- GPS Sensor

Cameras

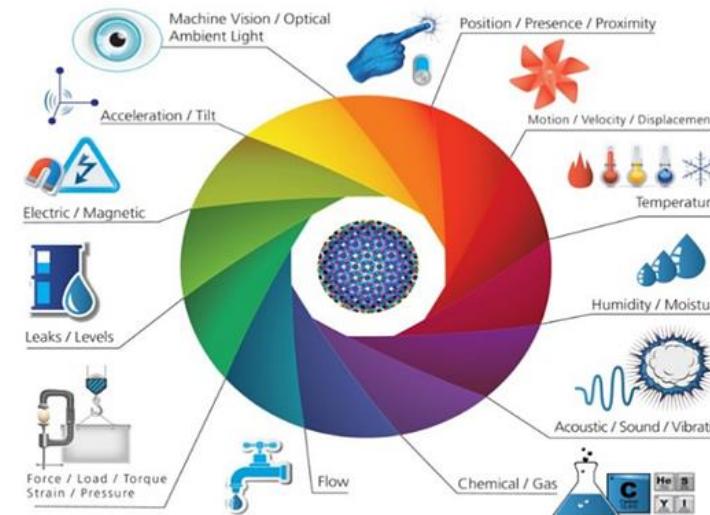
OBD Access

Controlling

- Raspberry Pi



- ▶ WAN Connectivity
- ▶ Cloud Services



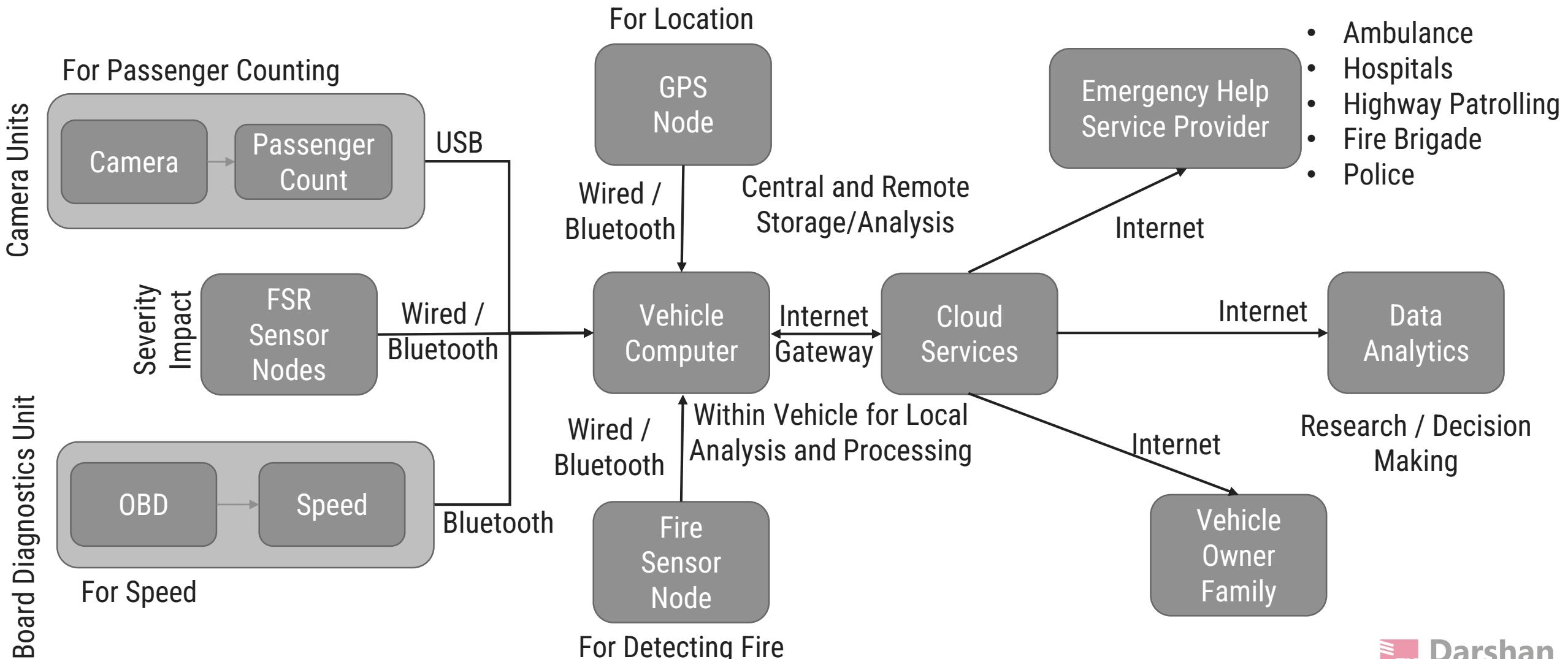
For Monitoring and Analysis

- PC
- Mobile
- Tablet



Collision Impact Detection Using IoT

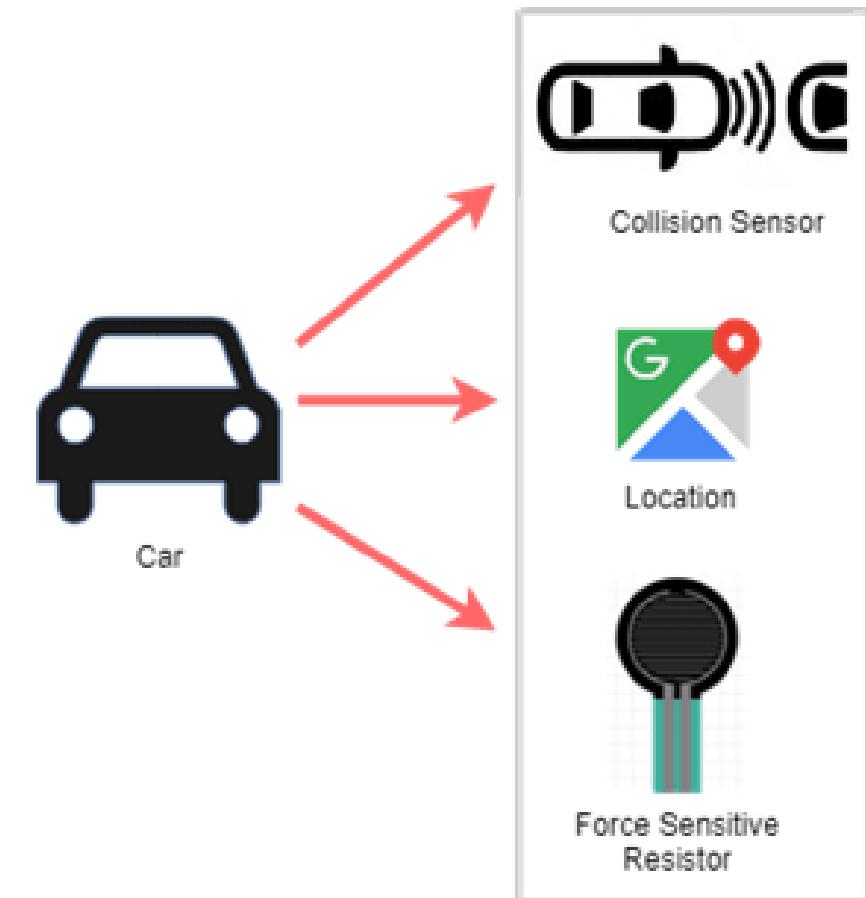
Architecture



Collision Impact Detection Using IoT

How it works?

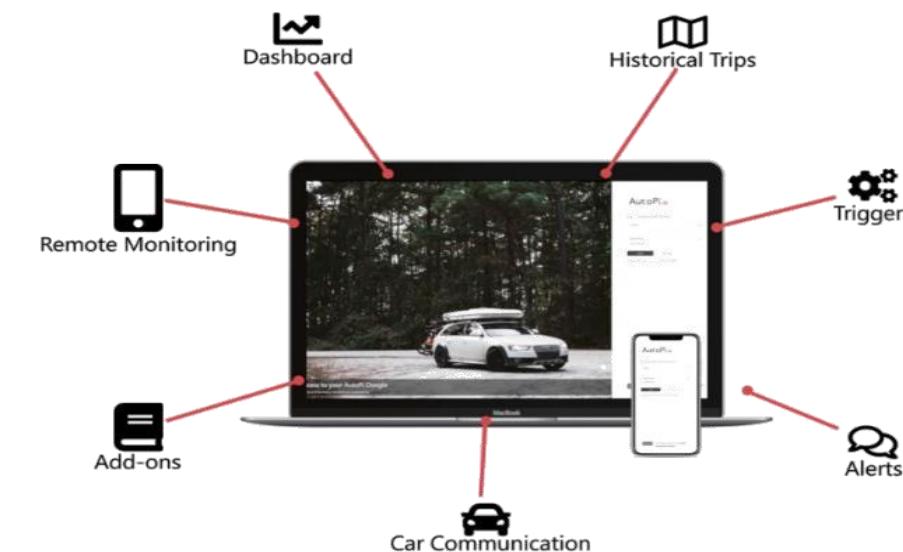
- ▶ Force sensitive resistor (FSR) sensors are useful to measure pressure applied on the vehicle during the collision.
- ▶ The FSR sensor are mounted on the vehicle at the places where impact probability is higher than the other part of vehicle during an accident.
- ▶ Based on the FSR data we can estimate the severity of the collision.
- ▶ Fire sensor and GPS should be placed at appropriate place of the vehicle.
- ▶ All the sensor nodes should have Wired / Bluetooth connectivity with the vehicle computer.
- ▶ Camera units should be placed on board for capturing all the passengers.



Collision Impact Detection Using IoT

How it works?

- ▶ The camera units should have **USB connectivity** with vehicle computer.
- ▶ The **camera** units **sends** the **captured images** to the **vehicle computer**.
- ▶ The **OBD** unit of the vehicle should have **Bluetooth connectivity** with the vehicle computer
- ▶ The vehicle computer
 - Collects the **data** from the **camera** units, **OBD** unit and the **sensor nodes**.
 - Apply some business **logic**,
 - Do **analysis** on the collected data on it as per the requirements.
 - Send the analyzed **data** to the **cloud** computer.
 - Should have Internet connectivity



Collision Impact Detection Using IoT

How it works?

- ▶ Using **OBD** (On-Board Diagnostics) device the system **records speeds** of the vehicle.
- ▶ The fire sensor and GPS also sends the sensed data to the vehicle computer.
- ▶ Based on the analyzed data and **severity** of the collision the **cloud system sent alert messages to all the emergency help service provider and family member** of the vehicle owner.
- ▶ The **cloud data** is used for **data analytics** and to **provide the priority based emergency services**.
- ▶ The system is **automatic activated when the collision is detected**.





Water Quality Monitor

Section - 5

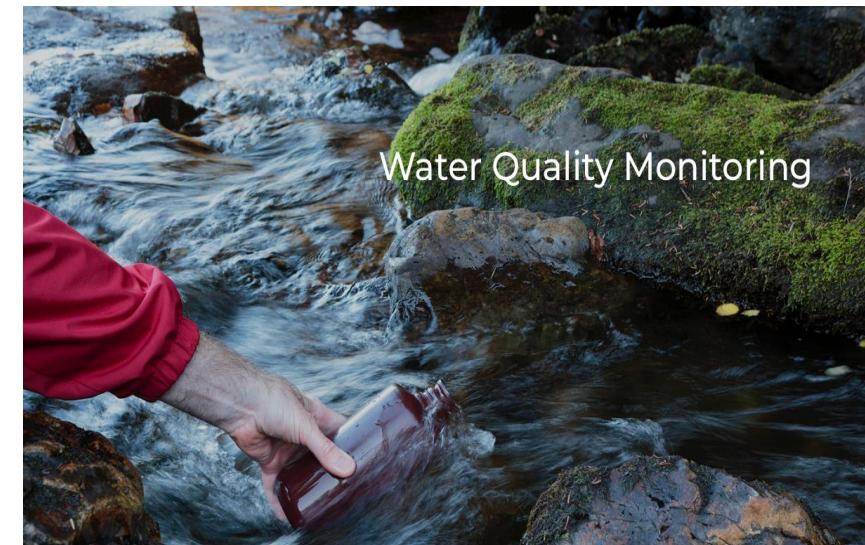
- ▶ At present water quality is monitored very expensive and inefficient by manual or nodal network methods
- ▶ Manual Method
 - Water samples are taken manually from the different points of a water area.
 - The samples sent to laboratories
 - Various tests are conducted to find the values of
 - PH, Conductivity and Temperature
 - TDS, Dissolved Oxygen
 - Turbidity
 - Chloride Content
 - The reports of the test sends to the concern authorities with in a week.



Water Quality Monitoring - IoT Based Application

Overview

- ▶ Water is essential for life and the polluted water is a one of the major challenge in the world.
- ▶ So water quality monitoring is necessary for us.
- ▶ Water Quality Monitoring (WQM) helps in preventing and controlling water pollutions.
- ▶ Water quality should be checked regularly to avoid serious health issues.
- ▶ IoT can help to develop a perfect WQM system.



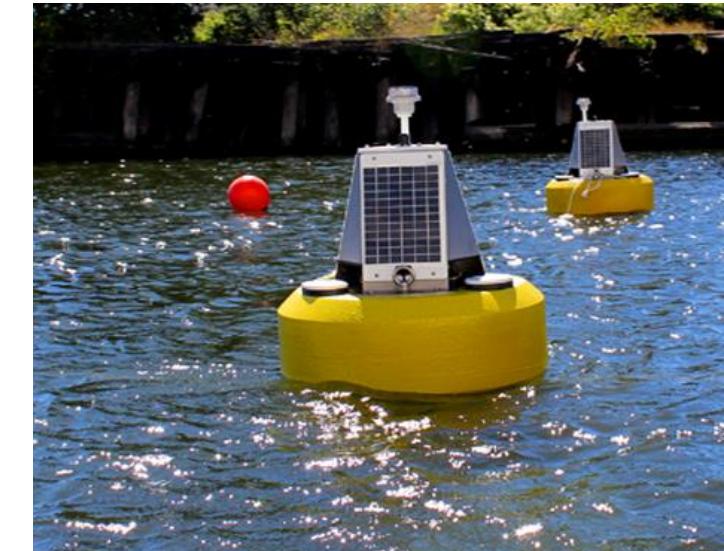
Manual Method

- ▶ The methods have some **limitations** like
 - Chance of **human error** in the sampling
 - Sampling **cost** is **very high** because of to cover very large area of waterbody
 - Sampling and Testing both are **time consuming**
 - **Not real-time** monitoring which leads to may serious issues.



► Nodal Network Method (Electronics Sensor Monitoring)

- Water quality is **measured by** electronic **sensors** installed in the different points of a water area.
- All the sensors are **wirelessly connected** with the Internet.
- The water quality **data sends to** the **cloud** server at **regular interval**.
- This method is giving **real-time data** contra to the manual method.
- System architecture of the **monitoring system** is **divided into** following four areas
 - Sensing
 - Data Collection
 - Processing
 - Communication



- ▶ **Nodal Network Method
(Electronics Sensor Monitoring)**

- ▶ The methods have following **limitations**

- Very expansive
- Maintenance of the sensors along with power source are challenging
- Natural disaster and aquatic animal movement in the waterbody may damage the sensors
- The sensors are not moveable



Water Quality Monitoring - IoT Based Application

Importance

- ▶ A **low cost solution** is possible for WQM using latest technology, Data Analytics, USVs, Machine Learning, AI and IoT.
- ▶ No need to **install sensors permanently** into the waterbody in the solution
- ▶ An **autonomous boat** (USV), mounted **with** all the necessary **sensors** can
 - Collect the **water quality data** and
 - Send the **data to the cloud** for data analytics
- ▶ All the **necessary parameters** of water can be **tracked and measured**.
- ▶ Detailed report of the **containment status** of the **entire waterbody** can be **generated**.



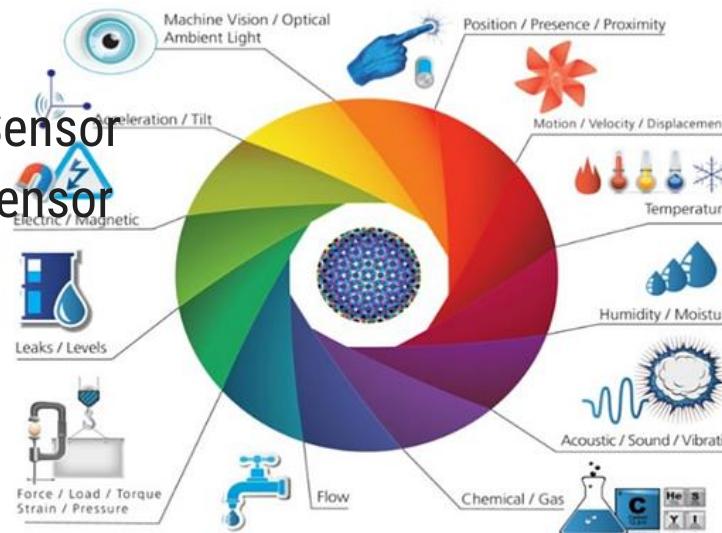
Water Quality Monitoring - IoT Based Application

Requirements for IoT Setup

► Water Sensors Kit

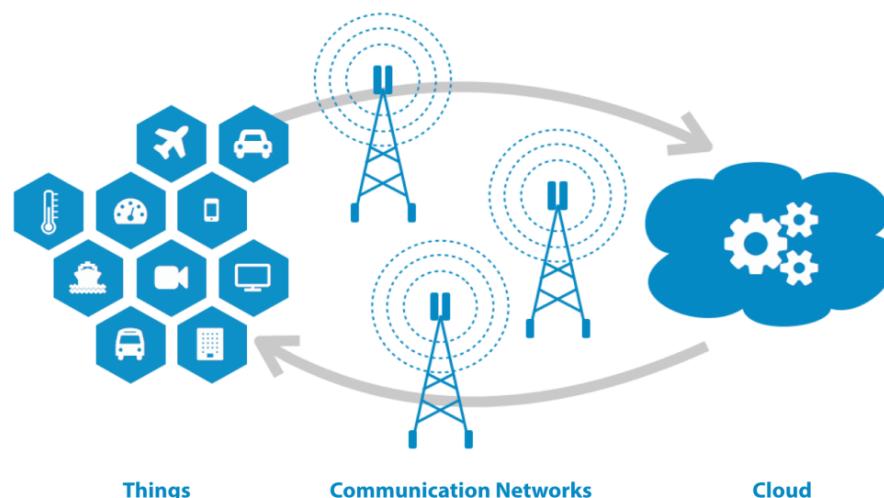
- PH Sensor
- GPS Sensor
- Turbidity Sensor
- TDS and Dissolved Oxygen Sensor

- Temperature Sensor
- Conductivity Sensor



► Controller

- ESP32 MCU or
- Arduino Uno with Wi-Fi or Network Shield



► For Monitoring and Analysis

- PC
- Mobile
- Tablet



► WAN Connectivity

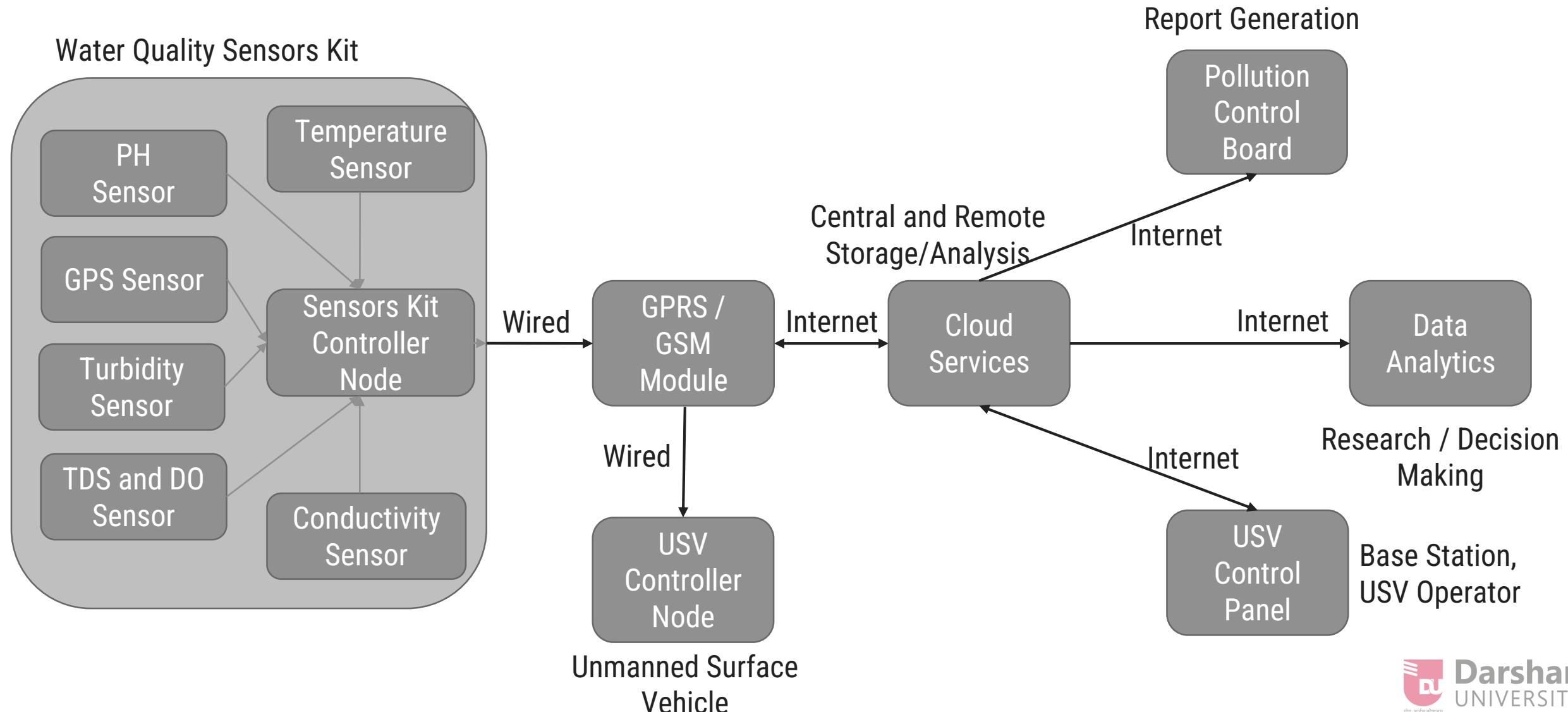
- GPRS or GSM Module

► Cloud Services

► Unmanned Surface Vehicle (USV)

Water Quality Monitoring - IoT Based Application

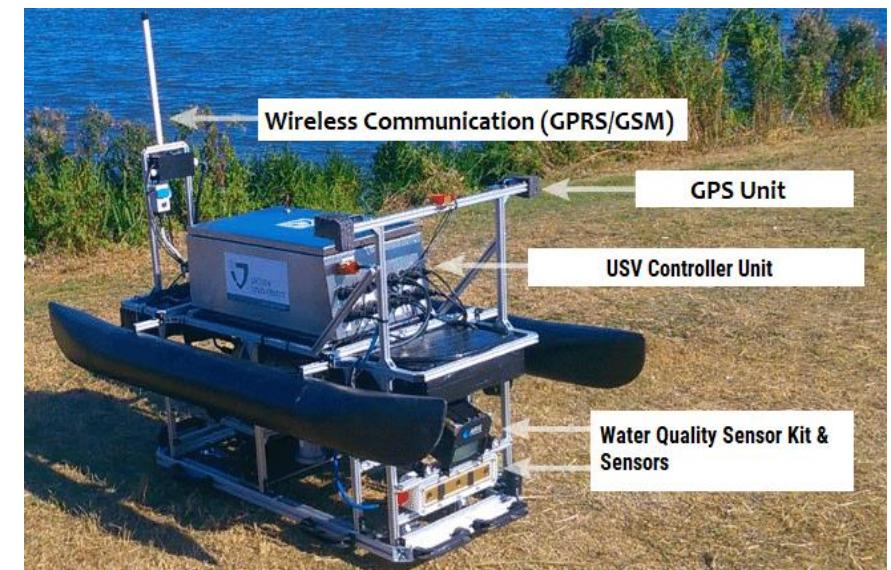
Architecture



Water Quality Monitoring - IoT Based Application

How it works?

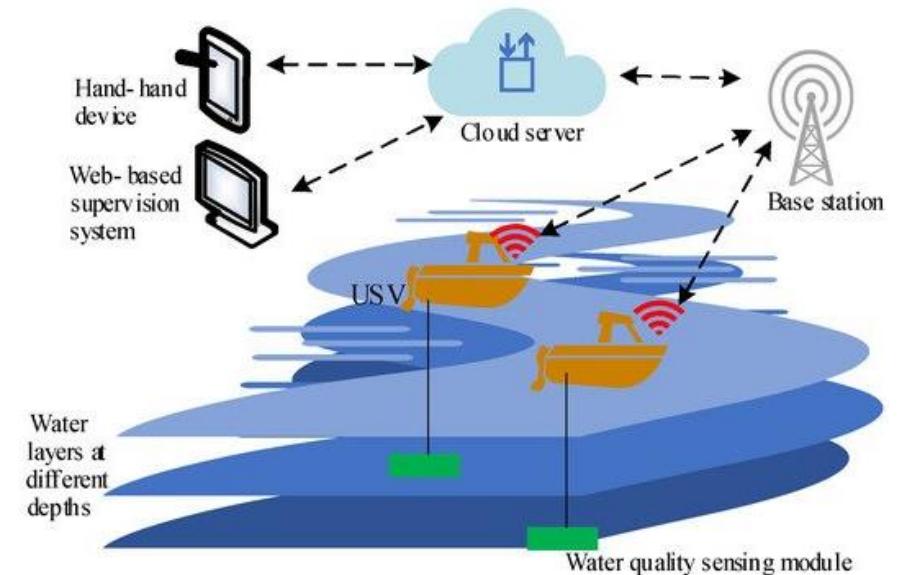
- ▶ A bundle of all the sensors or a **water quality sensor kit** should be mounted **on** an **autonomous** or unmanned surface **vehicle** (USV).
- ▶ The **USV** can be **operated by** an embedded **controller node**.
- ▶ The USV should have **GSM** or **GPRS internet connectivity** with the cloud server.
- ▶ The **sensor kit sense** the following **values**
 - **PH Value** of Water - PH Sensor
 - **Temperature** of Water - Temperature Sensor
 - **Conductivity** of Water - Conductivity Sensor
 - **Clarity** of Water - Turbidity Sensor
 - **Dissolved Solid and Oxygen** in Water - TDS and DO Sensor
 - **Location** of the reading - GPS Sensor



Water Quality Monitoring - IoT Based Application

How it works?

- ▶ The **kit controller node** sends all the sensed **data to the cloud** server via GPRS/GSM module
- ▶ The **GPRS/GSM** module is used to **provide the Internet connectivity** with the cloud server.
- ▶ The **cloud**
 - stores all the **data** received from the **USVs**,
 - **Execute** some business **logic** and
 - **Do analysis** on the stored data.
- ▶ All those who have **rights to access** the **cloud data** can **get** water quality records of the waterbody.
- ▶ Like,
 - Members of Pollution Control Board
 - Data analytics team
 - USV operator



Water Quality Monitoring - IoT Based Application

How it works?

- ▶ Authorities of the board and data analytics teams may execute corrective action based on the data.
- ▶ An USV control panel is installed in the base station of the WQM system.
- ▶ The control panel should have internet connectivity for controlling the USV.
- ▶ Using the control panel, an operator can set paths and control the USV.





IoT with Python

Section - 6

Introduction To Python : Python in IoT

- ▶ Python's popularity in IoT is attributed to its ease of use, an abundance of libraries and frameworks, and its versatility.
- ▶ Python offers a rich set of tools and libraries that simplify IoT development.
- ▶ Some of the best solutions for IoT in the Python programming language are as follows:
 - ✓ Python on Raspberry Pi
 - ✓ Python on PyBoard
 - ✓ ESP8266, ESP32 with Micropython



IoT Tools

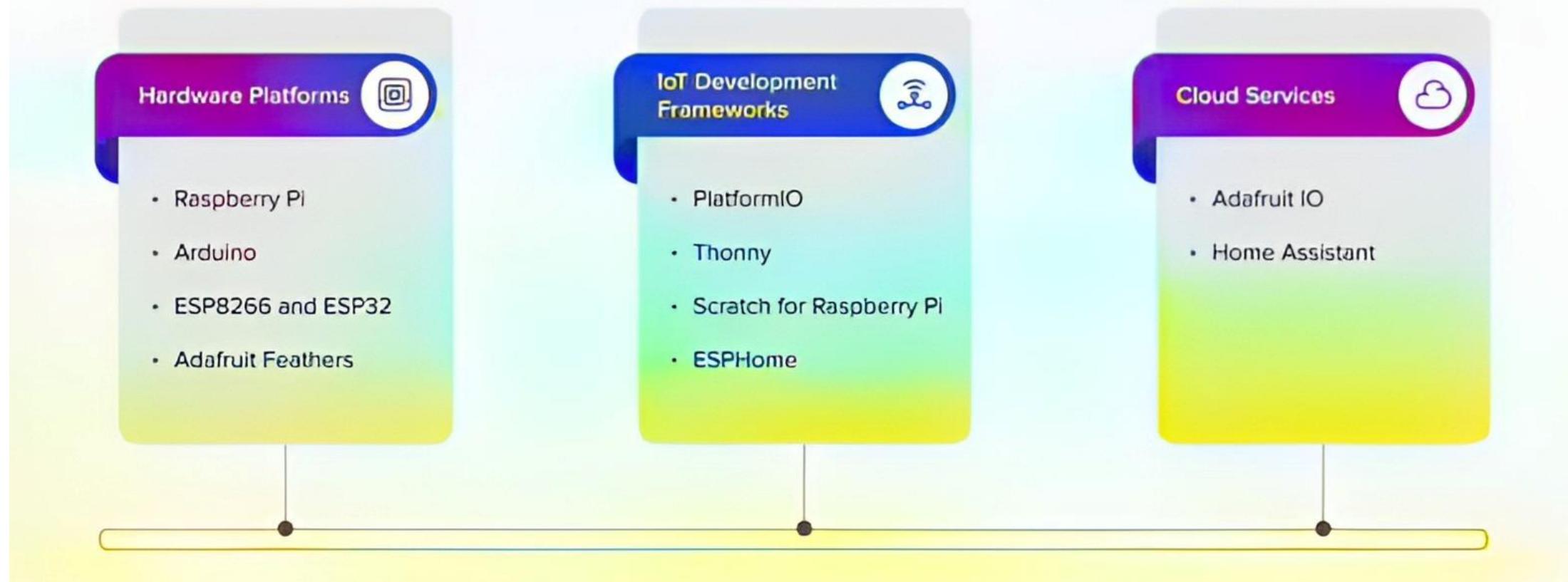
Section - 7

IoT Tools

- ▶ The variety of tools and platforms can be overwhelming when you're just getting into IoT.
- ▶ There are many Tools available for the development of the IoT System from prototyping to production, simple to sophisticated, open-source to commercial.
- ▶ We can Classify tools as per the level of complexity like:
 - IoT Tools for **Hobbyists and Small Projects**
 - IoT Tools for Middle-Ranged Business Automation
 - Advanced IoT Tools for Enterprise and Smart Factories

IoT Tools for Small Projects

IoT Tools for Hobbyists and Small Projects



Hardware Platforms

- ▶ **Raspberry Pi** is a must-have for makers and coders. This tiny yet powerful computer can connect to sensors, actuators, and other hardware. Program it with Python, Scratch, or JavaScript for limitless possibilities.
- ▶ **Arduino** is a go-to choice for prototyping with sensors and actuators. Its simple IDE and Wiring language make it perfect for beginners and pros alike. You can connect your Arduino to the cloud with ease.
- ▶ **ESP8266 and ESP32** are super affordable WiFi-enabled microcontrollers.
 - Program them in C/C++, MicroPython, or Arduino to build wireless projects quickly.
- ▶ **Adafruit Feathers** are ready-to-use microcontroller boards with built-in WiFi and Bluetooth.
 - Simply plug in the sensors, run the code, and you're ready to go. Their user-friendly design gets you prototyping in no time.

IoT Development Frameworks

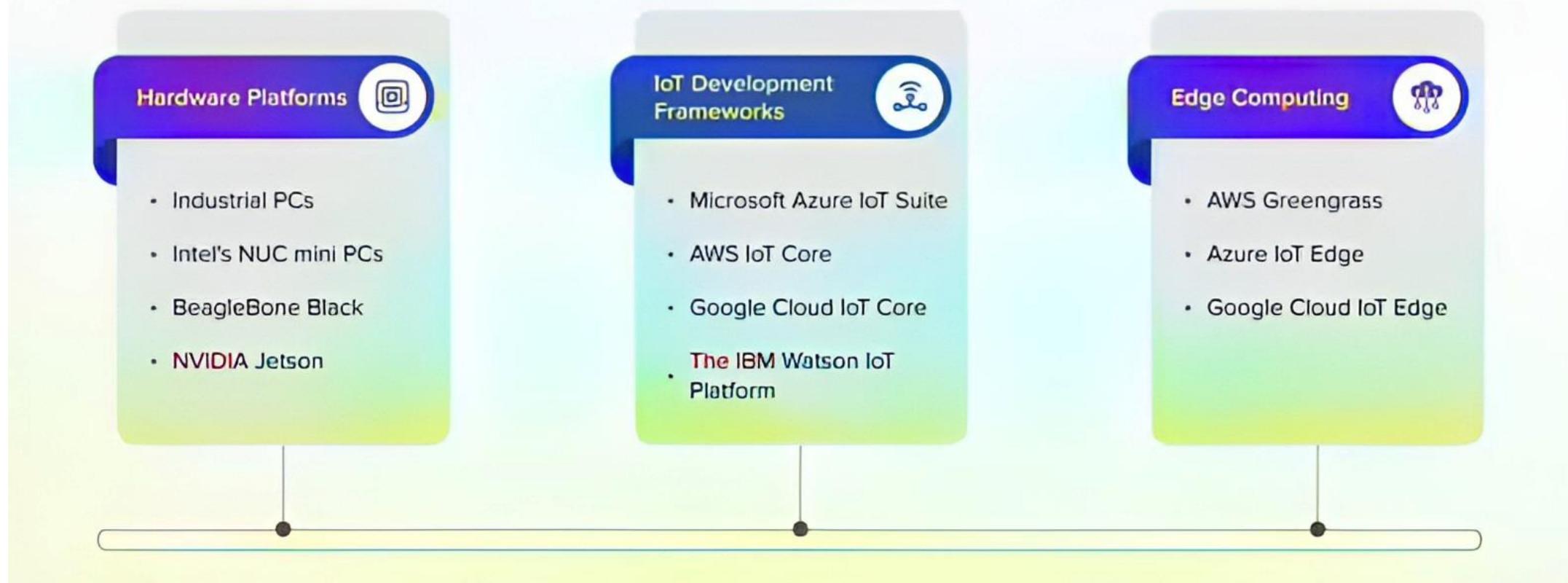
- ▶ **Arduino IDE** is the most popular IoT development framework, it is an open-source platform that makes creating hardware projects easy.
 - The intuitive interface and simple coding language C/C++ help quickly prototype your ideas.
- ▶ **PlatformIO** is an open-source ecosystem for IoT development, supporting a vast range of development boards and microcontrollers. It also has a robust library manager, advanced debugging tools, and continuous integration.
- ▶ **Thonny** is a beginner-friendly **IDE for Python programming** with a simple interface designed for learning.
 - Thonny is perfect for IoT projects using microcontrollers like the mentioned Raspberry Pi with the support of Python.
- ▶ **ESPHome** is an open-source framework for home automation that lets you control your ESP8266/ESP32-based devices from any phone or tablet. ESPHome has a simple setup and integrates with popular smart home software.

Cloud Services

- ▶ **Adafruit IO** is an easy-to-use cloud platform to store and retrieve data from sensors, actuators, and more.
- ▶ **Home Assistant** is an open-source home automation platform that prioritizes local control and privacy. It is powered by an active developer community producing integrations for hundreds of devices. The platform enables you to automate your lights, sensors, appliances, and much more. Home Assistant can run on your hardware or in the cloud.

IoT Tools for Middle- Ranged Business

IoT Tools for Middle-Ranged Business Automation



Hardware Platforms

- ▶ **Industrial PCs** offer the stability, reliability, and longevity required for IoT edge devices. With features like extended temperature range, vibration resistance, and fast processors, industrial PCs can serve as the backbone of your IoT infrastructure.
- ▶ **Intel's NUC (The Next Unit of Computing) mini PCs** are perfect for IoT applications that require a small footprint but plenty of processing power. The wide range of models offers different connectivity options, CPU choices, and expandability to suit various IoT needs.
- ▶ Being an open-source electronics platform powered by an embedded Linux board, the tiny **BeagleBone Black** is packed with useful features. It offers programmable GPIOs, a USB host, an Ethernet port, and more, making it ideal for building IoT prototypes and proof of concepts.
- ▶ **NVIDIA Jetson** modules bring the power of AI and GPU acceleration to edge devices with features like HDMI output, USB 3.0, Gigabit Ethernet, and more

IoT Development Frameworks

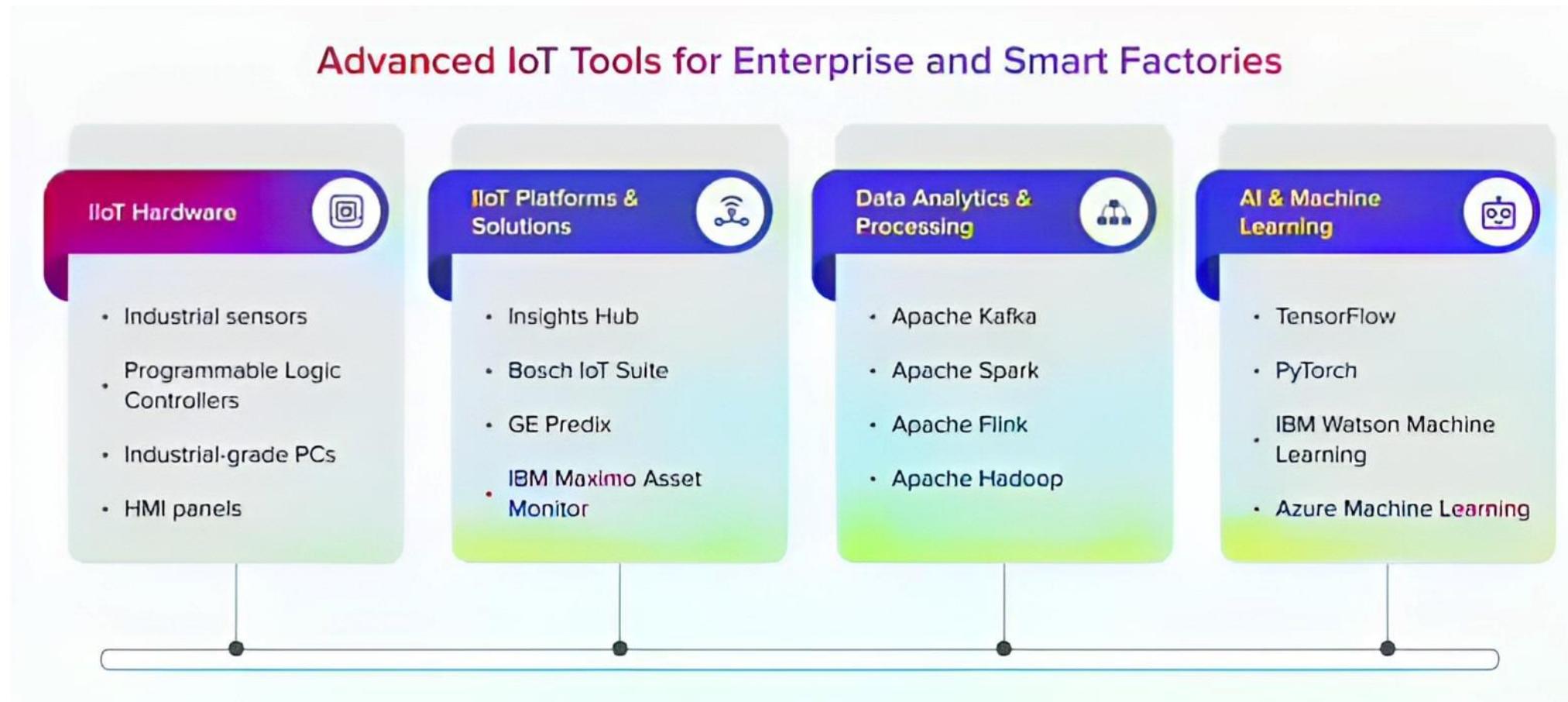
- ▶ **Microsoft Azure IoT Suite** is a comprehensive set of cloud services and APIs that helps you connect, monitor, and control billions of IoT assets.
 - You can collect data from IoT devices at any scale, analyze the data using [Azure Machine Learning](#), and take action on the insights using business logic.
- ▶ **AWS IoT Core** is used to connect IoT devices to the [AWS Cloud](#).
 - It provides a managed cloud service that lets you easily supply and register millions of IoT devices and interact with them seamlessly and securely.
- ▶ **Google Cloud IoT Core** is a fully managed service that will help you connect, manage, and ingest data from globally dispersed devices.
 - It provides a scalable, efficient, and economical IoT data pipeline to [Google Cloud](#).
- ▶ **IBM Watson IoT Platform** helps businesses collect and organize data from billions of connected devices.
 - It offers an open and scalable platform for quickly developing IoT applications and insights.

Edge Computing

- ▶ **AWS Greengrass** is perfect for running local computing, messaging, data caching, and ML inference at the edge.
 - Deploy your cloud logic to connected devices so they can act locally on the data they generate while still using the **AWS Cloud** for management, analytics, and durable storage.
- ▶ Run AI and custom business logic on edge devices with the same Azure services you use in the cloud. **Azure IoT Edge** extends Azure IoT to the edge of the network, enabling IoT solutions that deliver low latency to operate autonomously and access data where it's created.
- ▶ **Google Cloud IoT Edge** is a tool to deploy and run containerized services directly on edge devices to process data and act on it immediately. It brings AI and advanced analytics to the edge with on-device machine learning.
- ▶ The future is smart, so get out there and start connecting!

IoT Tools for Enterprise and Smart Factories

Advanced IoT Tools for Enterprise and Smart Factories



Industrial IoT (IIoT) Hardware

- ▶ Get real-time data to power automation and predictive maintenance with **industrial sensors** designed to withstand harsh industrial environments.
- ▶ Industrial Sensors will monitor critical parameters like temperature, pressure, vibration, and flow rate.
- ▶ **Programmable Logic Controllers (PLCs)** act as the brains of your automation system, controlling machinery and processes.
- ▶ The latest PLCs offer increased processing power, connectivity options, and advanced features to support IIoT applications.
- ▶ **Industrial-grade PCs** serve as data collection and communications hubs, connecting machines to the industrial Internet.
- ▶ **Human-machine interface (HMI) panels** provide operators with visual feedback and control of automation systems.
- ▶ Modern HMIs feature intuitive touchscreens, advanced graphics, and IoT connectivity for remote monitoring.

IIoT Platforms & Solutions

- ▶ **Siemens MindSphere (now Insights Hub)** is an advanced IIoT system that collects data from connected products and assets.
- ▶ It then applies artificial intelligence and analytics to gain insights, optimizing processes and operations.
- ▶ **Bosch IoT Suite** is a cloud-based platform connecting products and machines to the Internet of Things.
- ▶ **GE Predix** is an industrial-grade operating system for the Industrial Internet that allows data from machines to be analyzed and used for optimization.
- ▶ **Predix** collects data from assets and applies analytics and AI to generate insights that drive improvements in operations.
- ▶ **IBM Maximo Asset Monitor** is a comprehensive solution that uses IoT sensors and AI to gain insights into the health and performance of physical assets.



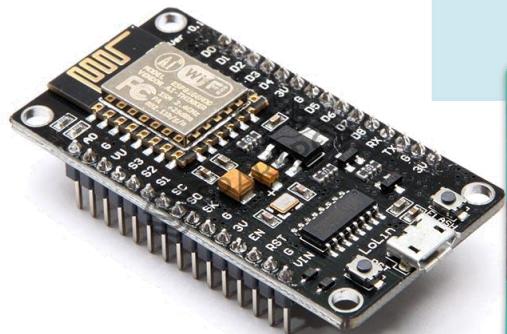
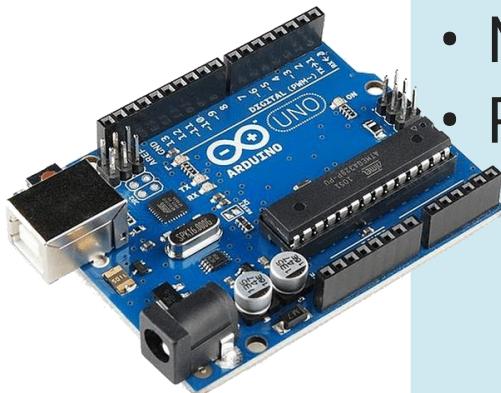
IoT Application Development

Section - 8

Developing IoT Application

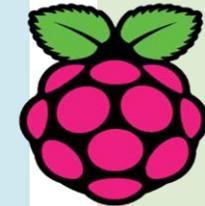
Hardware

- Arduino UNO
- Node MCU
- Raspberry Pi



IoT Development Framework

- Arduino IDE
- Thonny



Raspberry Pi



MQTT Server

- Hive MQ



HIVEMQ

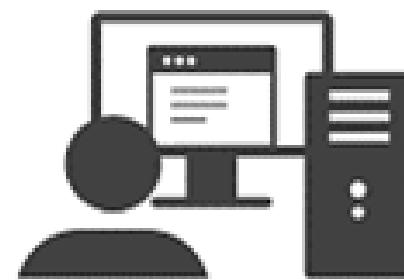


Cloud Computing and IoT

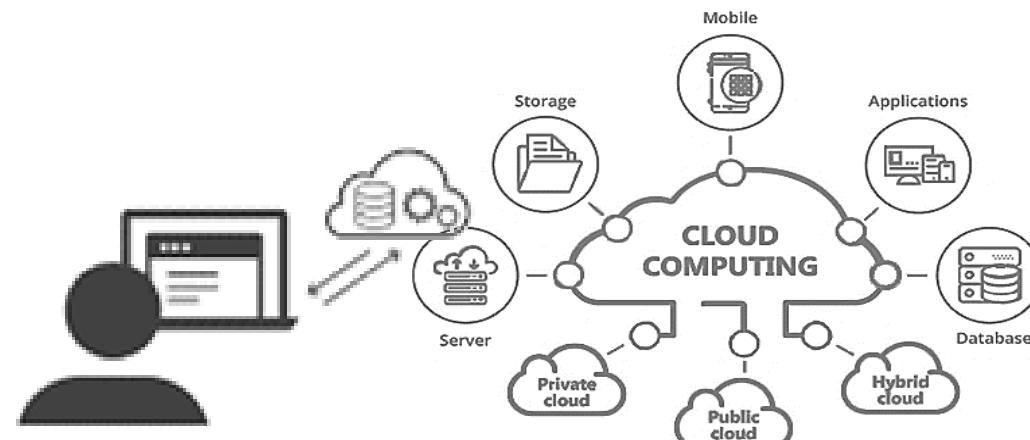
Section - 9

Introduction to Cloud Computing

- ▶ Cloud computing : “Cloud computing is a model for enabling **universal, convenient, on-demand** network access to a shared pool of **configurable computing resources** (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”.
- ▶ Cloud computing is the **delivery of computing services over the internet**, by providing **flexible, affordable, effective and efficient** resources for development.



Application logic runs
on user's computer



Application logic runs
in the cloud

Benefits of Cloud Computing

- ▶ The availability of virtually **unlimited storage and processing capabilities at low cost** enabled the realization of a new computing model.
- ▶ Large companies (like Amazon, Google, Facebook, etc.) widely adopted this paradigm for **delivering services over the Internet**, gaining both economical and technical benefits.
- ▶ Cloud computing model is attractive since it frees the business owner from the need to invest in the infrastructure, renting resources according to needs and **only paying for the usage**.
- ▶ Cloud Computing allows to **decrease operating costs**, that includes servers, storage, databases, networking, software, analytics, and intelligence.

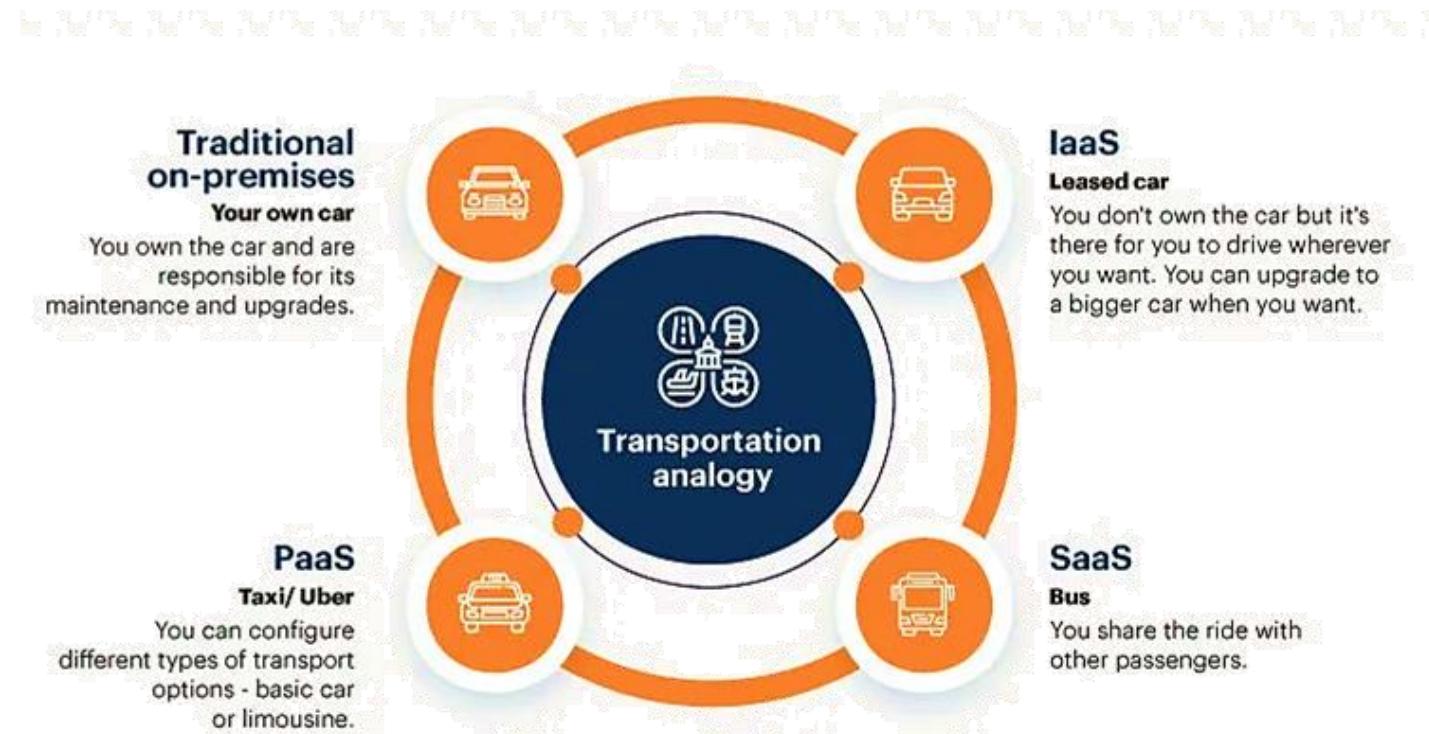
Service Models of Cloud Computing

- ▶ There are three main service models of cloud computing.
 1. Infrastructure as a Service (IaaS)
 2. Platform as a Service (PaaS)
 3. Software as a Service (SaaS).
- ▶ Each model represents a different part of the cloud computing stack.

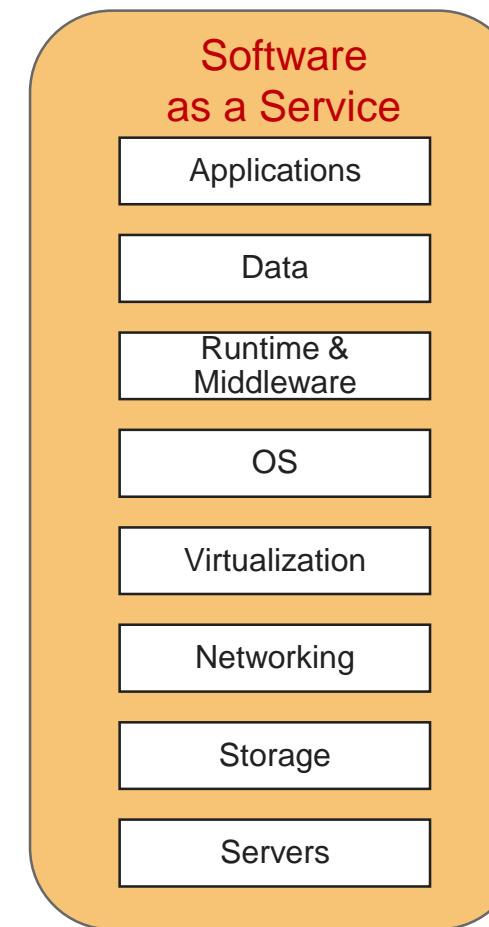
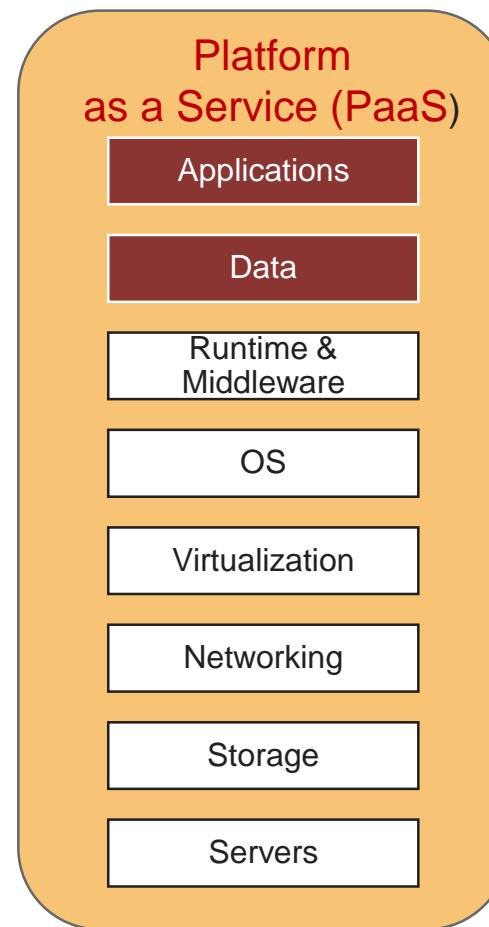
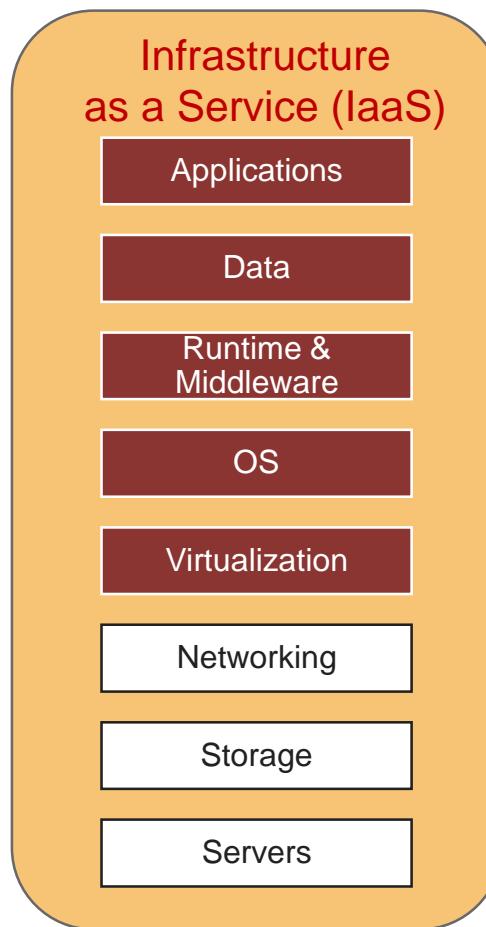
Cloud Computing Services

- ▶ **Software as a Service (SaaS).**
- ▶ SaaS provides **application-level** services.
- ▶ It is tailored to a wide variety of business needs.
- ▶ Such as
 - Customer relationship management (CRM)
 - Marketing automation
 - Business analytics.
- **Platform as a Service (PaaS).**
 - Geared toward software development teams.
 - PaaS provides **computing and storage infrastructure** and also a **development platform layer**, with components such as
 - ❖ Web servers
 - ❖ Database management systems
 - ❖ Software development kits (SDKs) for various programming languages.
- **Infrastructure as a Service (IaaS)**
 - IaaS provides users access to **raw computing resources**.
 - Such as
 - ❖ Processing power
 - ❖ Data storage capacity
 - ❖ Networking, in the context of a secure virtual data centre.

Cloud Computing Services - Transportation analogy



Cloud Computing Services



You Manage

Vendor Manage

Software-as-a-Service (SaaS)

- ▶ Complete software application as a service is provided to the user that is **run and managed by the service provider**.
- ▶ It can also be called application as a service as a pay monthly, yearly etc. **subscription**.
- ▶ In SaaS, user **don't need to worry about software up-gradation and management**.
- ▶ A common example of a SaaS application is web-based email where you can send and receive email without having to manage the server that the email program is written.



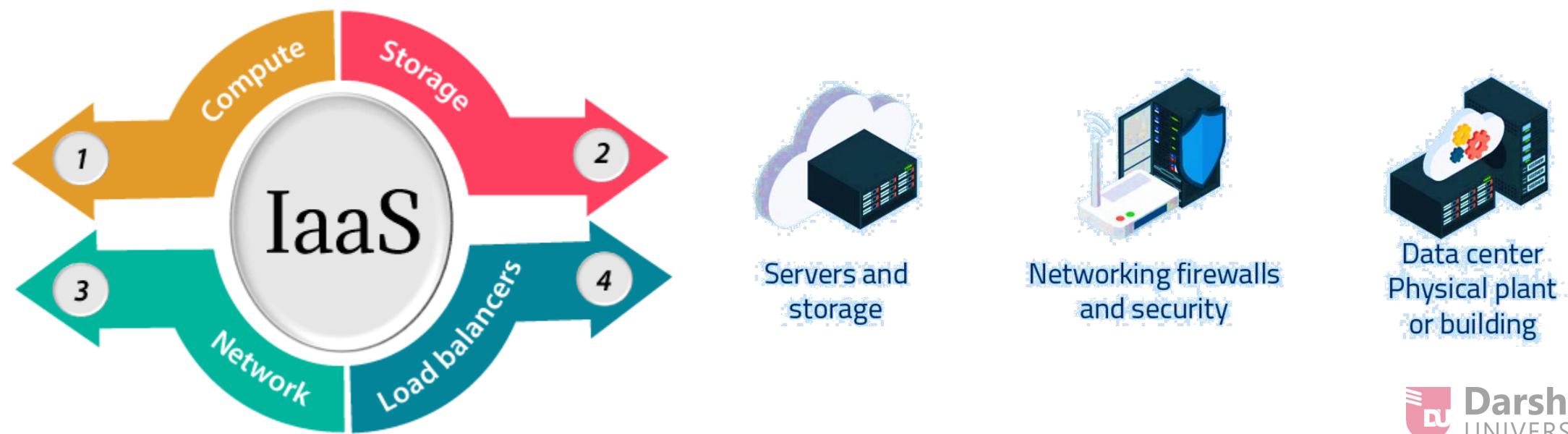
Platform-as-a-Service (PaaS)

- ▶ It provides a **development environment** to application developers.
- ▶ Operating system, programming-language execution environment, database, web server, development tools, APIs, libraries, etc., will be provided by the cloud service provider.
- ▶ Users have to **build, manage and maintain the applications**.
- ▶ **Application developers develop and run their software on a cloud platform instead of directly buying and managing the essential hardware and software layers.**
- ▶ AWS Lambda, Google App Engine, IBM Cloud Foundry, Oracle Cloud Platform, Red Hat-OpenShift, Zoho etc.

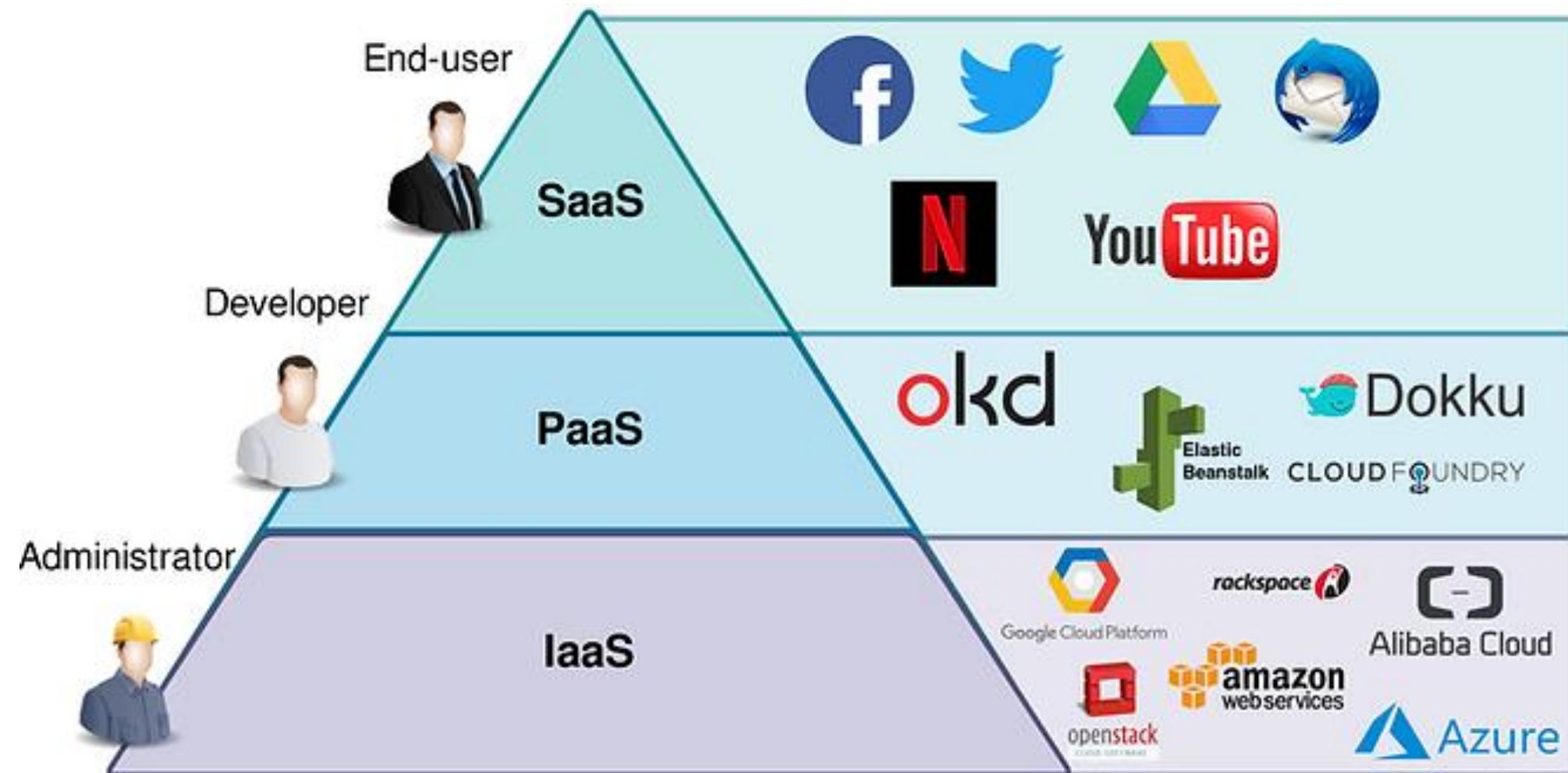


Infrastructure-as-a-Service (IaaS)

- ▶ IaaS provides **access to computing hardware, networking features, and data storage space**.
- ▶ Where the consumer is able to deploy and run software, which can include operating systems and applications.
- ▶ The consumer does not manage or control the underlying cloud infrastructure but has **control over operating systems, storage, and deployed applications**.
- ▶ This service provides the highest level access.



Cloud Computing Services Examples



Types of Cloud Based on Services

► Software-as-a-Service (SaaS):

- Complete software application as a service is provided to the user.
- It is application as a service as well, pay monthly, yearly etc. as subscription

► Platform-as-a-Service (PaaS):

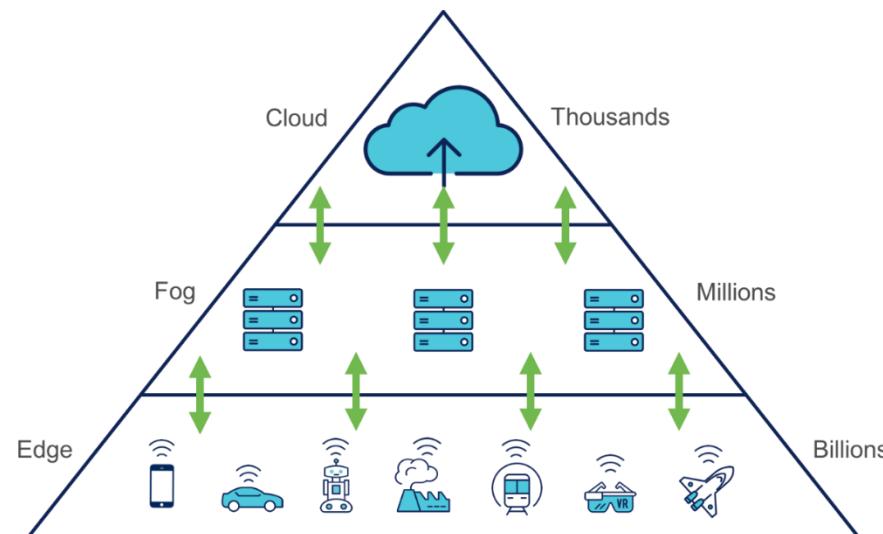
- Development tools, APIs, libraries etc. will be provided by the cloud service provider.
- Users have to build, manage and maintain the applications (provides platform to develop)

► Infrastructure-as-a-Service (IaaS):

- Virtual Machines kind of support.
- Users shall manage the machines.
- Users shall select the OS and underlying applications.
- PAY as YOU use! (with IaaS approach, you choose virtual machines over physical machines)

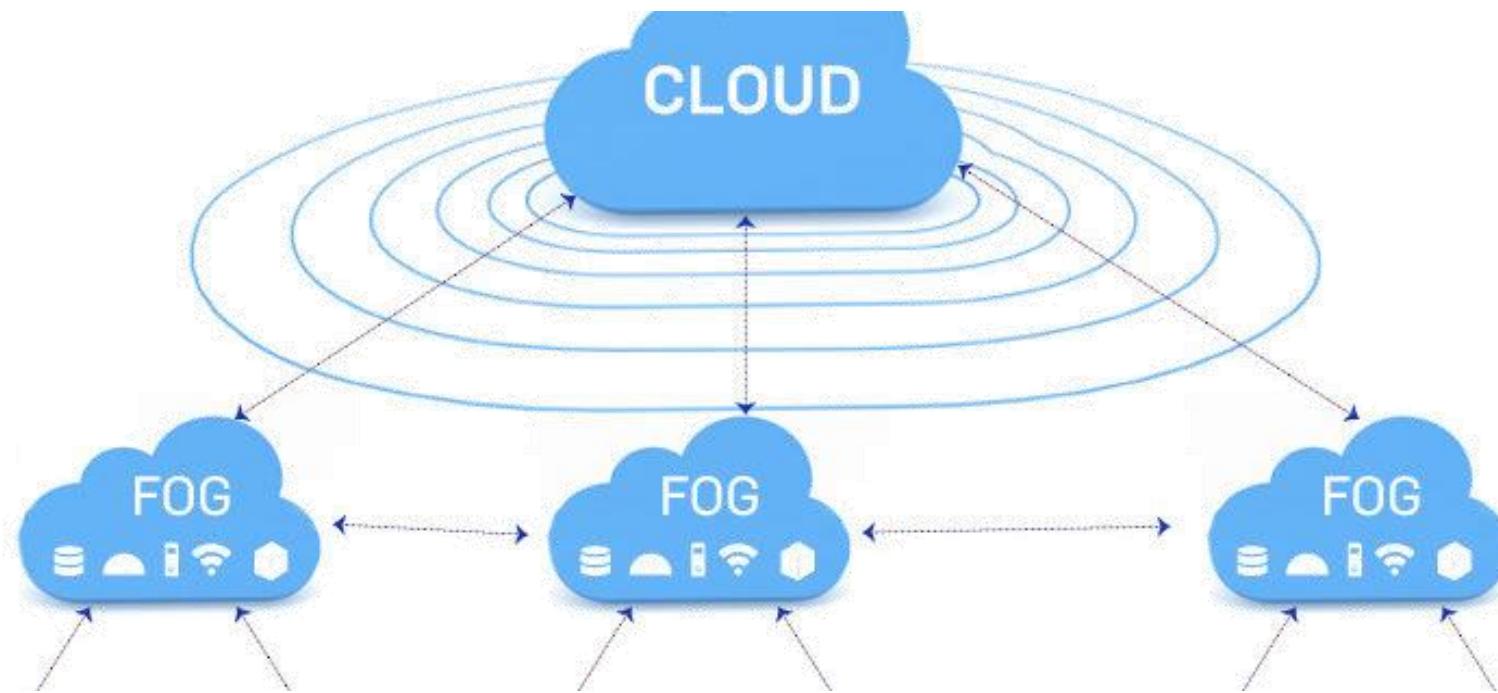
Fog Computing

- ▶ Cisco coined the term fog computing (or fogging) in 2014, so it is new to the general public.
- ▶ Fog and cloud computing are inter-twined.
- ▶ In nature
 - Fog is closer to Earth than clouds
- ▶ In the tech world, it's the same;
 - Fog is closer to end-users, bringing cloud capabilities to the ground.

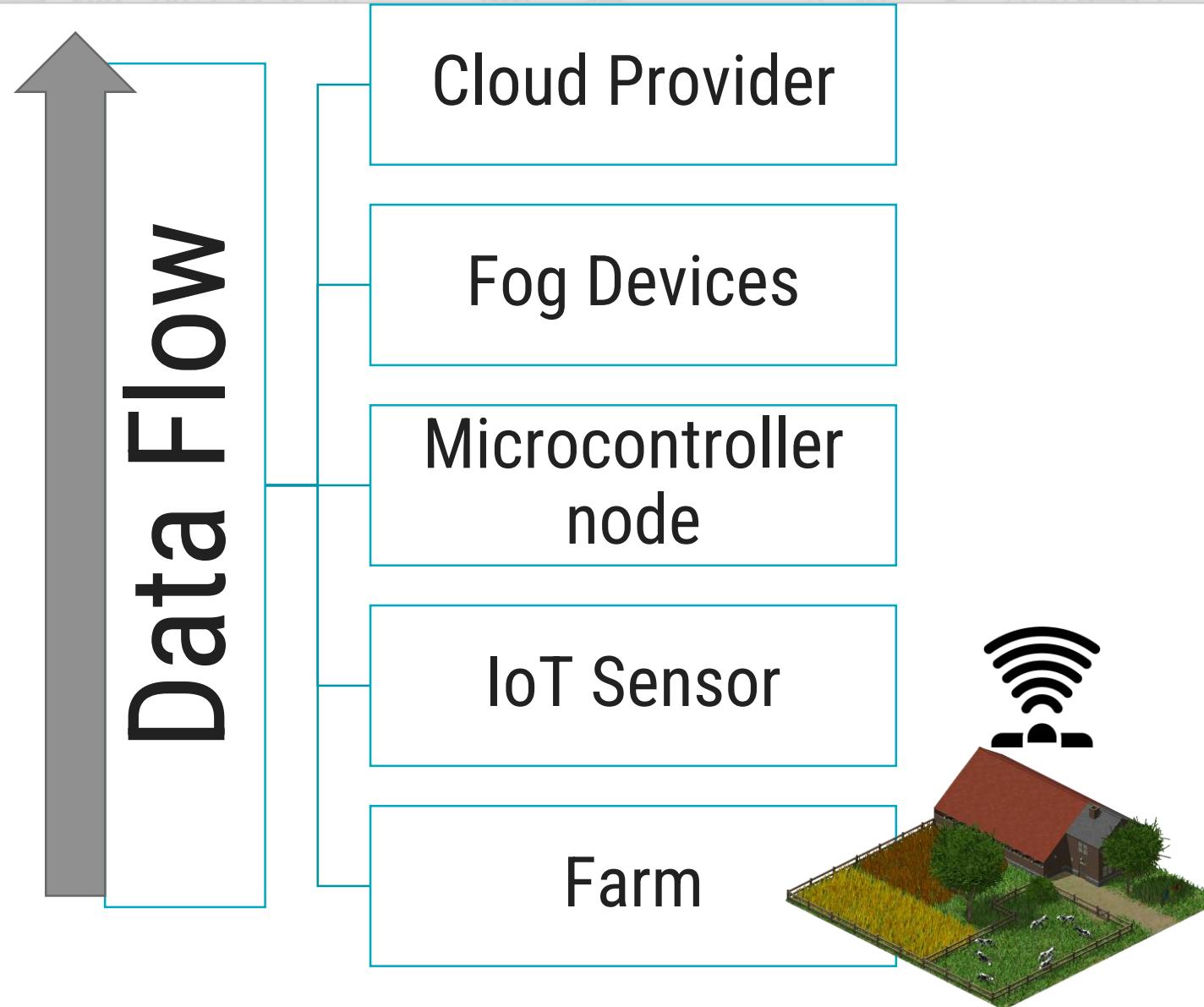


Fog Computing

- ▶ Fog is an extension of cloud computing that consists of multiple edge nodes directly connected to physical devices.
- ▶ The considerable processing power of edge nodes allows them to compute large amounts of data without sending them to distant servers.

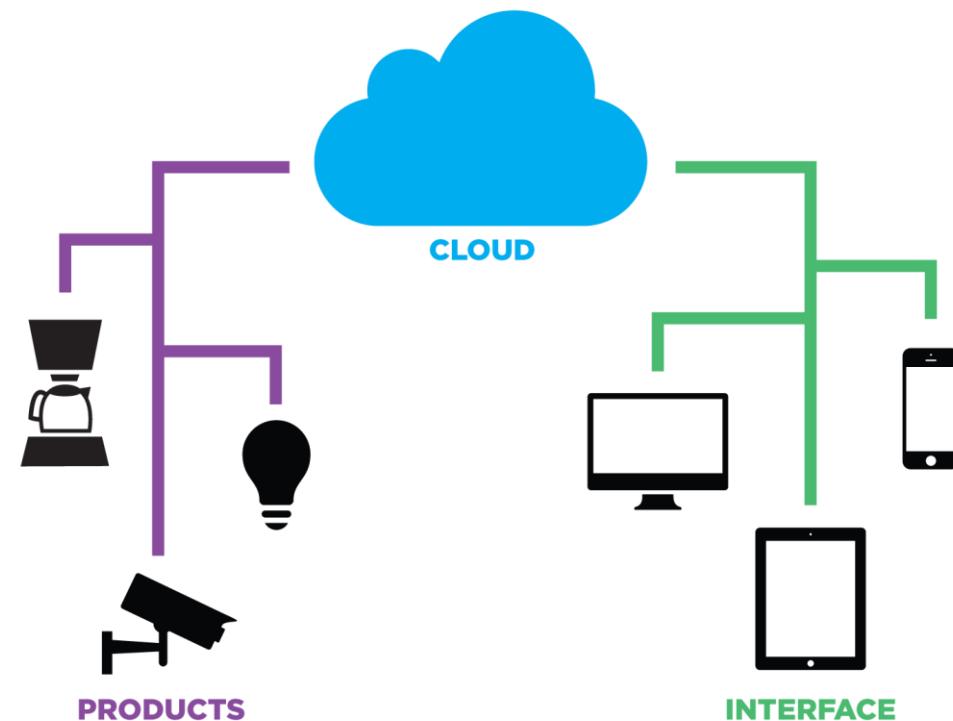
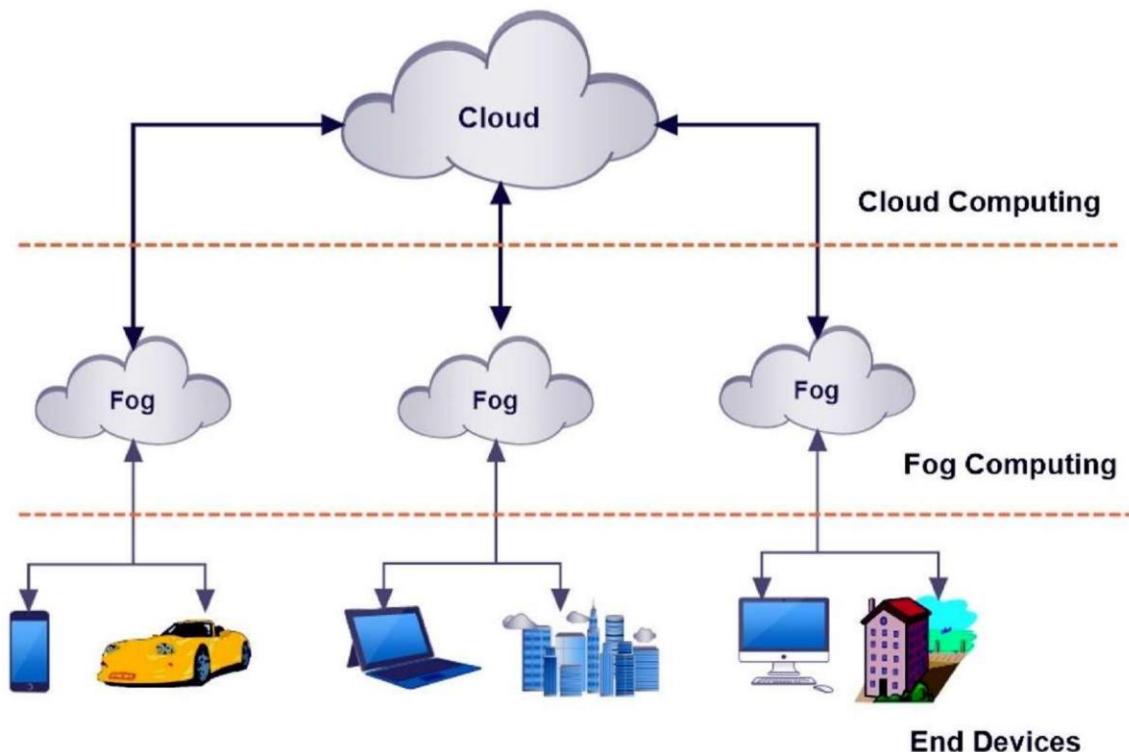


Data Flow – Smart Farm IoT System



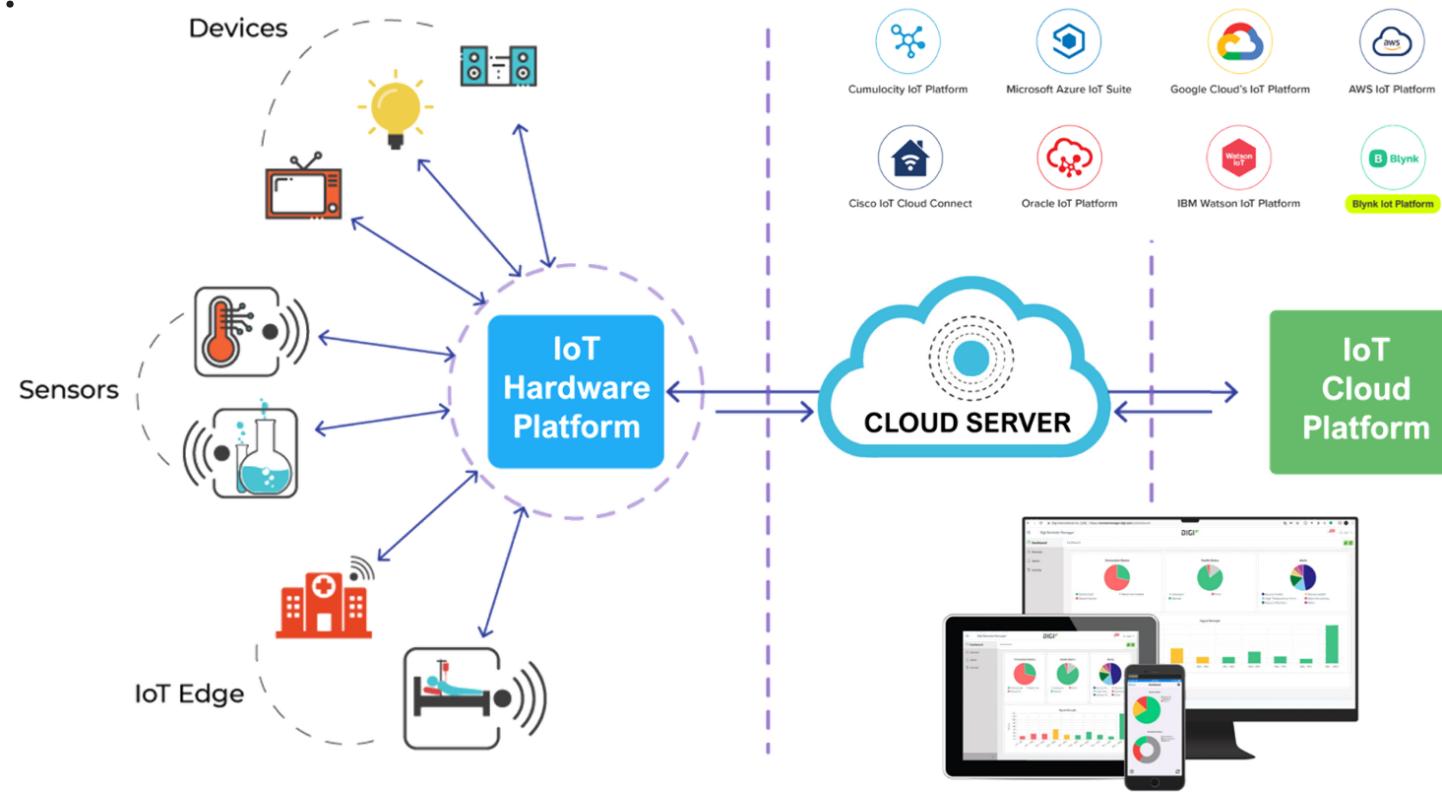
Fog Computing vs. Cloud Computing

- Fog Computing
- Multiple Nodes and Nodes Connected to cloud
- Cloud Computing
- Directly the End-Devices are connected to Cloud



How it Works?

- ▶ 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.



IoT Clouds



Cumulocity IoT Platform



Microsoft Azure IoT Suite



Google Cloud's IoT Platform



AWS IoT Platform



Cisco IoT Cloud Connect



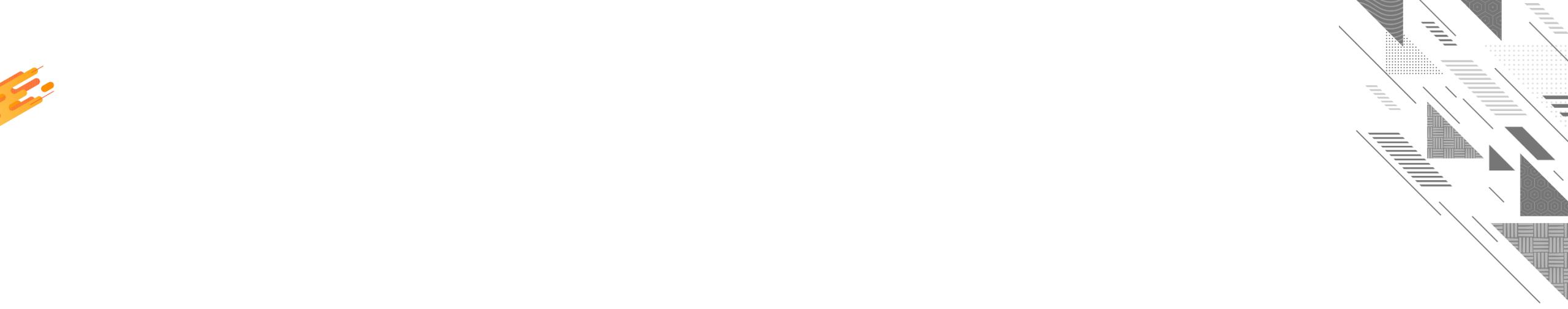
Oracle IoT Platform



IBM Watson IoT Platform



Blynk IoT Platform



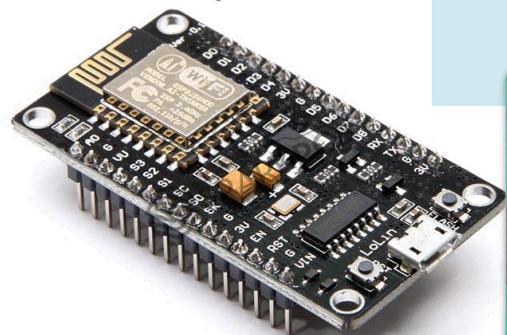
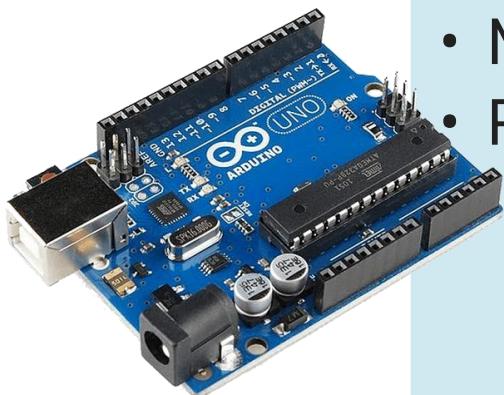
IoT Application Development with IoT Cloud (Blynk)

Section - 10

Developing IoT Application with IoT Cloud

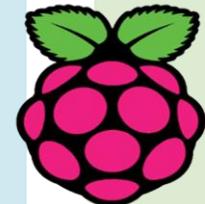
Hardware

- Arduino UNO
- Node MCU
- Raspberry Pi



IoT Development Framework

- Arduino IDE
- Thonny



Raspberry Pi



IoT Cloud

- Blynk
- Arduino IoT Cloud

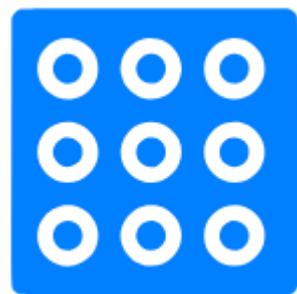


IoT Cloud - Blynk

- ▶ Blynk is a full suite of software, required to prototype, deploy, and remotely manage connected electronic devices at any scale: from personal IoT projects to millions of commercial connected products.
- ▶ Blynk is a Cloud service with **PaaS** type.



IoT CLOUD



Fogwing



Google Cloud IoT Core

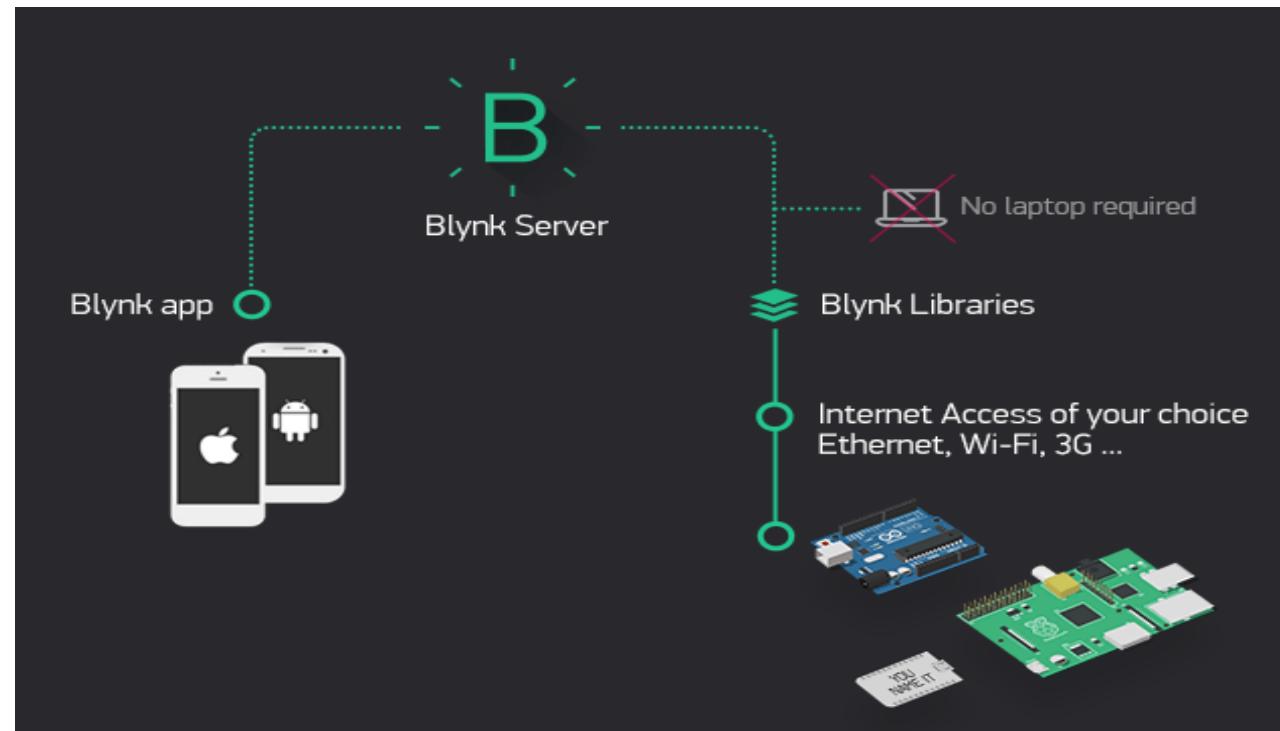


Blynk Cloud

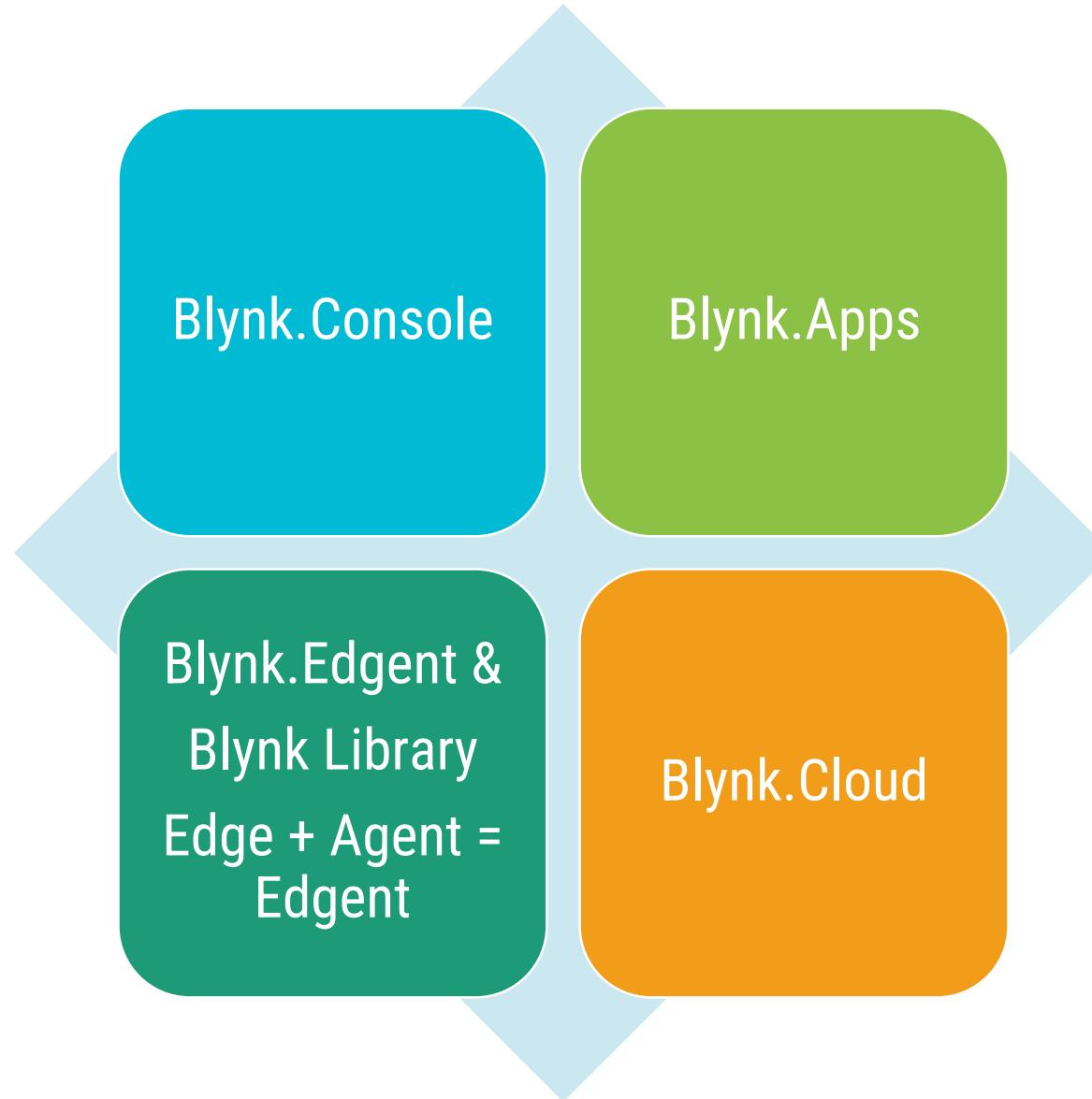
- ▶ Blynk is considered a Platform as a Service (PaaS).
- ▶ It provides a cloud-based platform for developers and businesses to build, deploy, and manage connected electronic devices.
- ▶ Blynk provides the infrastructure to develop users' own IoT applications on top of Blynk's core functionalities rather than just providing a pre-built software service like a SaaS would.

Blynk Cloud Service for IoT

- ▶ Blynk is a comprehensive software suite that enables the prototyping, deployment, and remote management of connected electronic devices at any scale.
- ▶ Blynk empowers users to connect their hardware to the cloud and create iOS, Android, and web applications, analyze real-time and historical data from devices, remotely control them from anywhere, receive important notifications, and much more.



Components of the Blynk IoT Platform

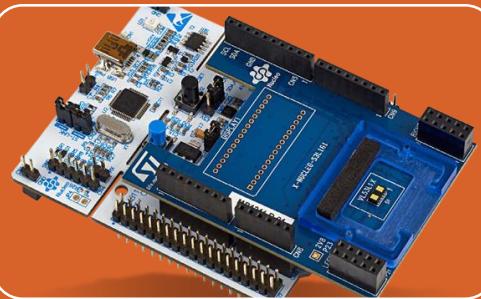


What You Need to Start Blynk



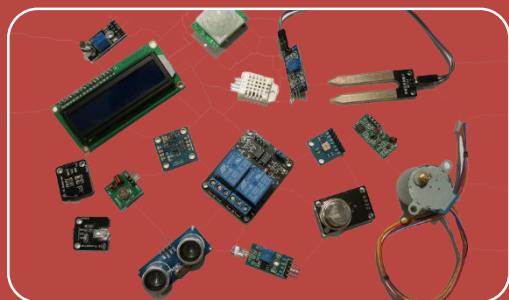
A Smart Phone with

- Android OS v4.2 +
- iOS v9+



Arduino like Board (For connecting with Cloud)

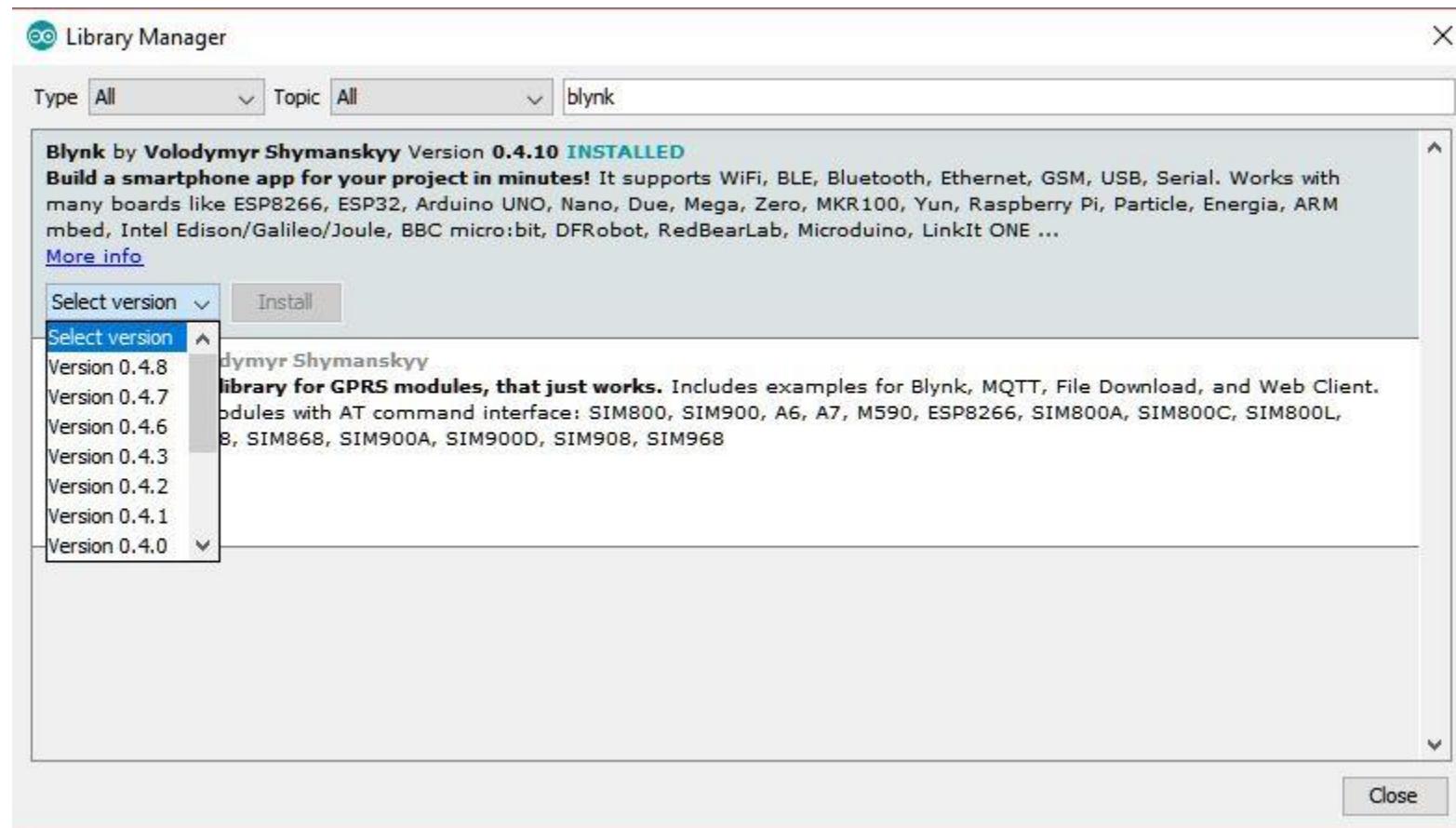
- Blynk Board
- ESP8266 Board (NodeMCU, WeMos D1, Witty Cloud)
- ESP32
- Nordic nRF51/nRF52



IoT Hardware (For Sensor and Actuators)

- Hardware Modules (any model) with Sensor and Actuators
- Modules like:
 - Arduino (any Model)
 - Raspberry Pi (any model)

Include Header file/ Library for Blynk



Blynk Authentication & initialization

- We need to give the Proper Template Id and Authentication token for the Device identity and connectivity with Blynk Cloud :

```
#define BLYNK_TEMPLATE_ID          "TMPLSU_PLFEC"  
#define BLYNK_DEVICE_NAME         "Quickstart Device"  
#define BLYNK_AUTH_TOKEN          "Zk6h1VfK1Ohi5Y8vWwy47gWnJCZf-c5d"
```

- To Start the communication with Blynk Cloud we have to call **Blynk.begin** method:

```
void setup()  
{    // Debug console  
    Serial.begin(115200);  
    // Begins the Serial Communication with 115200 baudrate  
    Blynk.begin(auth, ssid, pass);  
    // Begins the communication with Blynk Cloud  
}  
  
void loop()  
{  
    Blynk.run();  
    // Write Your Loop Code Here  
}
```

Read the value from Virtual Pin

- ▶ `BLYNK_WRITE()`
- ▶ This function used to read the changes or current value from dashboard.
- ▶ Once the value is received from the Dashboard, we can process it for further changes on Hardware board.
- ▶ `BLYNK_WRITE()` function is called every time the Virtual Pin 1 state changes.
- ▶ `BLYNK_WRITE()` function reads the value of virtual variable from the Dashboard.

```
BLYNK_WRITE(v0) //v0 Virtual pin v0
{
    // Set incoming value from pin v0 to a variable
    int value = param.asInt();

}
```

Write the Value on Virtual Pin

- ▶ **Blynk.virtualWrite(Vpin, value)**
- ▶ This function used to send the data to the Dashboard.
- ▶ This function will send value or variable from Arduino board to Dashboard.
- ▶ **Syntax:**

Blynk.virtualWrite(Virtual pin, value)

- ▶ **Virtual Pin :** Virtual Pin number assigned to particular widgets.
- ▶ **Value :** variable or any static value

// You can send any value at any time.

Blynk.virtualWrite (V1, 10);

// Here we Write Value 10 to pin V1

Demo Code : Blynk Cloud with NodeMCU -1

BlynkNodeMCU.demo

```
1 // Template ID, Device Name and Auth Token are provided by the Blynk.Cloud  
2 // See the Device Info tab, or Template settings  
3  
4 #define BLYNK_TEMPLATE_ID          "TMPLSU_PLFEC"  
5 #define BLYNK_DEVICE_NAME         "Quickstart Device"  
6 #define BLYNK_AUTH_TOKEN          "Zk6h1VfK1Ohi5Y8vWwy47gWnJCZf-c5d"  
7  
8 // Comment this out to disable prints and save space  
9 #define BLYNK_PRINT Serial  
10 #include <ESP8266WiFi.h> //Header File for the ESP8266 Wi-Fi Module support  
11 #include <BlynkSimpleEsp8266.h> // Library file for the Blynk cloud support for  
12 //ESP8266  
13 char auth[] = BLYNK_AUTH_TOKEN;  
14 // Your WiFi credentials.  
15 // Set password to "" for open networks.  
16 char ssid[] = "IoT_Demo";  
17 char pass[] = "diet@iot";
```

Demo Code : Blynk Cloud with NodeMCU -2

BlynkNodeMCU.ino

```
18
19 // This function is called every time the Virtual Pin 0 state changes
20 BLYNK_WRITE(V0)
21 {
22     // Set incoming value from pin V0 to a variable
23     int value = param.asInt();
24
25     // Update state to the V1, virtual pin
26     Blynk.virtualWrite(V1, value);
27 }
28
29 // This function is called every time the device is connected to the Blynk.Cloud
30 BLYNK_CONNECTED()
31 {
32     //
33 }
34
```

Demo Code : Blynk Cloud with NodeMCU -3

BlynkNodeMCU.ino

```
30 void setup()
31 {
32     // Debug console
33     Serial.begin(115200); // Begins the Serial Communication with 115200 baudrate
34
35     Blynk.begin(auth, ssid, pass); // Begins the communication with Blynk Cloud
36 }
37
38 void loop()
39 {
40     Blynk.run();
41
42     // You can inject your own code or combine it with other sketches.
43     // Check other examples on how to communicate with Blynk. Remember
44     // to avoid delay() function!
45 }
```

**Thank
You**



Prof. Bhushan D. Joshi
Computer Engineering Department
Darshan University, Rajkot

✉ bhushan.joshi@darshan.ac.in
☎ +91 8485979997

