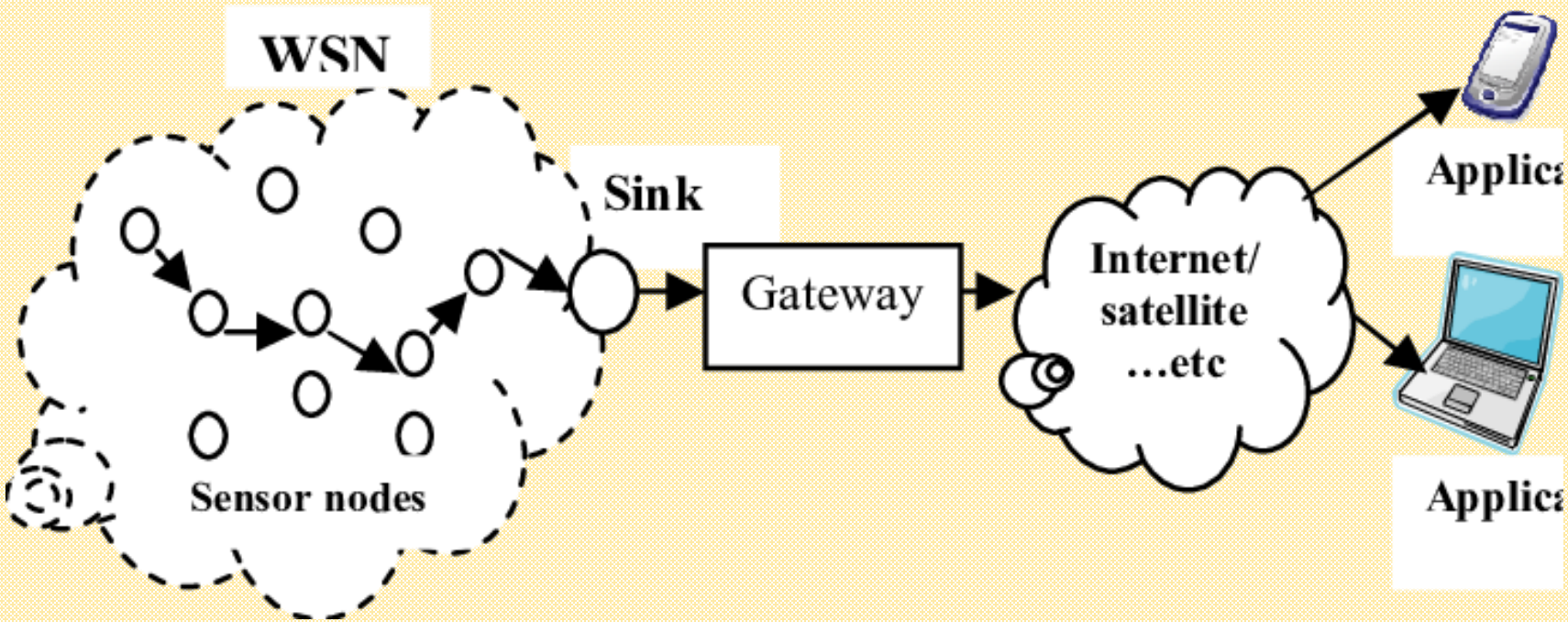
Wireless Sensor Network

- WSN: Things (sensor nodes) connected without a wire to gather some data.
- Wireless sensor network (WSN) refers to a group of dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location.
- Nodes are not directly connected to the Internet. Nodes route traffic to reach the sink node.
- WSN is sometime referred to as a subset of IoT.



The Internet of things (IoT)

IOT: WSN + Any physical object (Thing) + IP address + Internet + App + Cloud computing+ etc…

- Sensors send their data directly to the Internet because they have Internet Connection. The Internet of things (IoT) is the network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enable these objects to connect and exchange data.
- Each thing is uniquely identifiable through its embedded computing system but is able to inter-operate within the existing Internet infrastructure.
- All sensor data further processed and analyzed in the data analyzing area.

IoT Definitions

- The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

- The Internet of Things (IoT) describes the network of physical objects—“things”—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet.

- The Internet of things (IoT) describes physical objects (or groups of such objects) with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks.

- The Internet of Things (IoT) is a network of physical objects that are fitted with sensors, software and other technologies. Connected to the Internet, these ‘things’ are able to exchange real time data with other connected devices and systems over networks. These connected devices combine with automated systems to gather IoT data that can be analyzed to assist with tasks or learn how to improve a process.

WSN Vs IoT

Aspect	WSN (Wireless Sensor Network)	IoT (Internet of Things)
Definition	A network of spatially distributed sensor nodes to monitor physical or environmental conditions.	A broader ecosystem connecting various devices (including WSNs) to exchange data over the Internet.
Scope	Limited to sensing and short-range data transmission.	Encompasses sensing, processing, communication, and decision-making across devices globally.
Connectivity	Usually connected via local wireless protocols (e.g., Zigbee, Bluetooth, 6LoWPAN).	Uses both local (Wi-Fi, BLE) and global (Internet, cloud) connectivity.
Components	Mainly sensors, actuators, microcontrollers, and wireless transceivers.	Includes WSN components plus cloud platforms, AI, data analytics, and user interfaces.
Application Focus	Environmental monitoring, military, industrial sensing.	Smart homes, smart cities, healthcare, transportation, etc. (broader application range).
Internet Dependency	Not necessarily dependent on the Internet; often operates in isolated environments.	Strongly relies on Internet connectivity for data sharing and remote control.

IoT History

- While the Carnegie mellon University's vending machine was installed in 1982, this can't really be called the start of the IoT as a whole.

- The notion of the IoT was created in 1991 and further developed through the 1990s with Reza Raji describing the concept at the IEEE Spectrum in 1994.

- The actual term 'the Internet of Things' was created by Kevin Ashton in 1999, although the time when objects were connected directly to the Internet really began between 2008 and 2009.

- With more than 20 billion connected IoT devices today, experts are expecting this number to grow to 30 billion by 2025, 41 billion by 2027, and 50-75 billions by 2030. These figures include consumer IoT (smart home, wearables), industrial IoT (manufacturing, energy, logistics), smart cities, automotive, healthcare, and agriculture sectors.

IoT Characteristics

- **Connectivity** –

  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.

- **Intelligence and Identity**

  The extraction of knowledge from the generated data is very important.

- **Scalability**

  Scalability in IoT means the system can grow — adding more devices and handling more data — without losing speed or performance. It’s done by expanding hardware or using extra software layers to manage the huge data IoT generates.

- **Dynamic and Self-Adapting (Complexity)**

  IoT devices should dynamically adapt themselves to the changing contexts and scenarios.

IoT Characteristics

❖ IoT Architecture – Common Ecosystem

IoT architecture connects diverse devices from different manufacturers using various technologies and protocols. Its main role is to ensure **interoperability**, smooth **communication**, and that devices work together **without interfering** with each other — even as the number and types of devices grow.

❖ Self Configuring –

One key feature of IoT is that devices can **automatically set up**, **add themselves to a network**, and **update software** with minimal user effort. This makes IoT systems flexible and easy to expand.

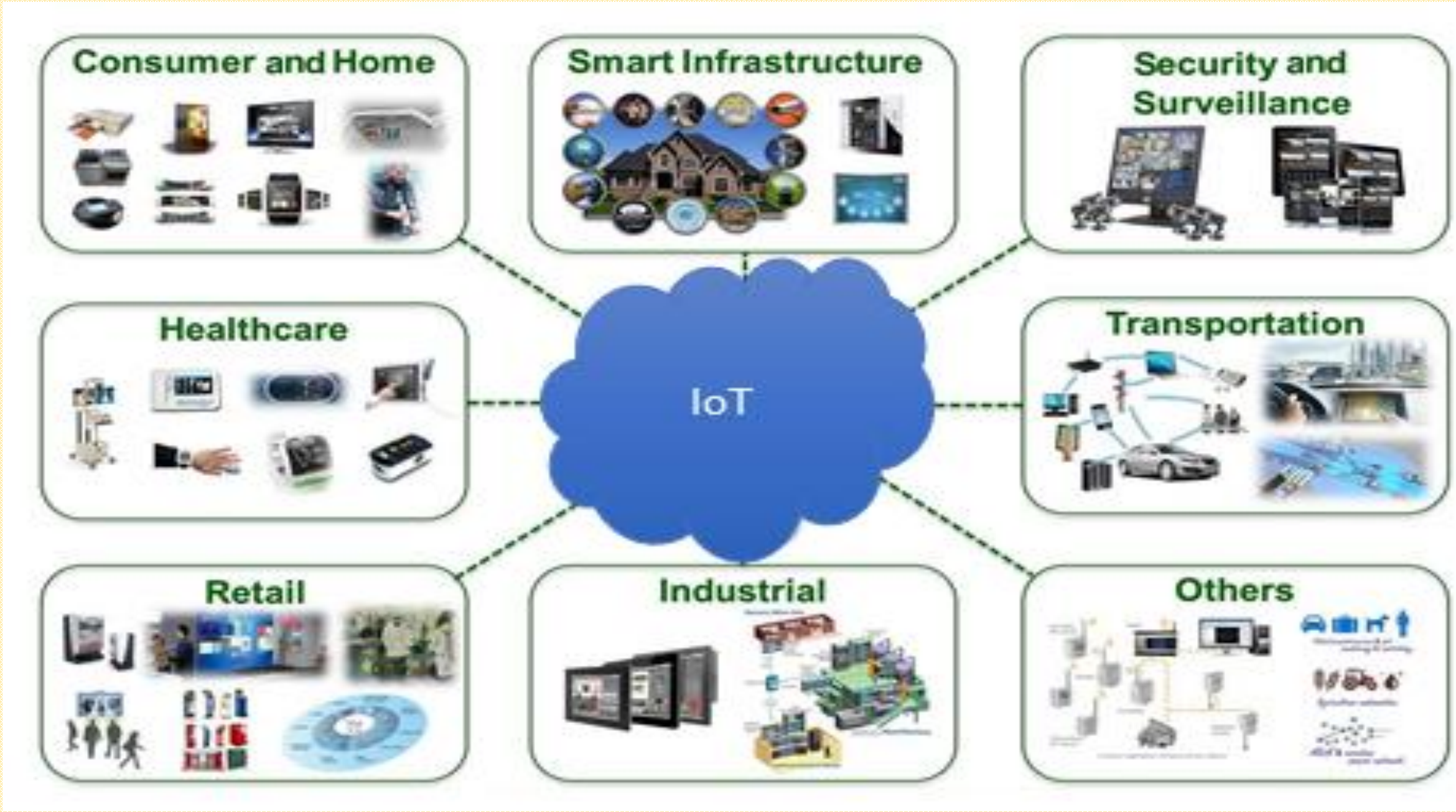
❖ Unique Identity of Things

Every IoT device has a **unique identity** (like a name or ID number). This unique ID helps distinguish each device, allows it to be managed, addressed, and controlled individually in a large network.

IoT Applications

Smart Home	Smart Lightening
	Smart Appliances
	Intrusion Detection
	Smoke/Gas Detection
Smart Cities	Smart Parking
	Smart Road & Traffic
	Structural Health Monitoring
	Emergency Response
Smart Environment	Weather Monitoring
	Air Pollution Monitoring
	Noise Pollution Monitoring
	Forest Fire Detection
Smart Energy	Smart Grids
	Renewable Energy Systems
	Prognostics

Smart Retail	Inventory Management
	Smart Payments
	Smart Vending Machine
Smart Logistics	Route Generation and Scheduling
	Fleet Tracking
	Ship Monitoring
	Remote Vehicle Diagnostics
Smart Agriculture	Smart Irrigation
	Green House Control
	Pest Spraying Control
	Plants Disease Detection
Smart Industry	Machine Diagnosis & Prognosis
	Indoor Air Quality Management
Smart Health & Lifestyle	Health & Fitness Monitoring
	Wearable Sensors


Consumer and Home
Smart Infrastructure
Security and Surveillance
Healthcare
IoT
Transportation
Retail
Industrial
Others

IoT Advantages

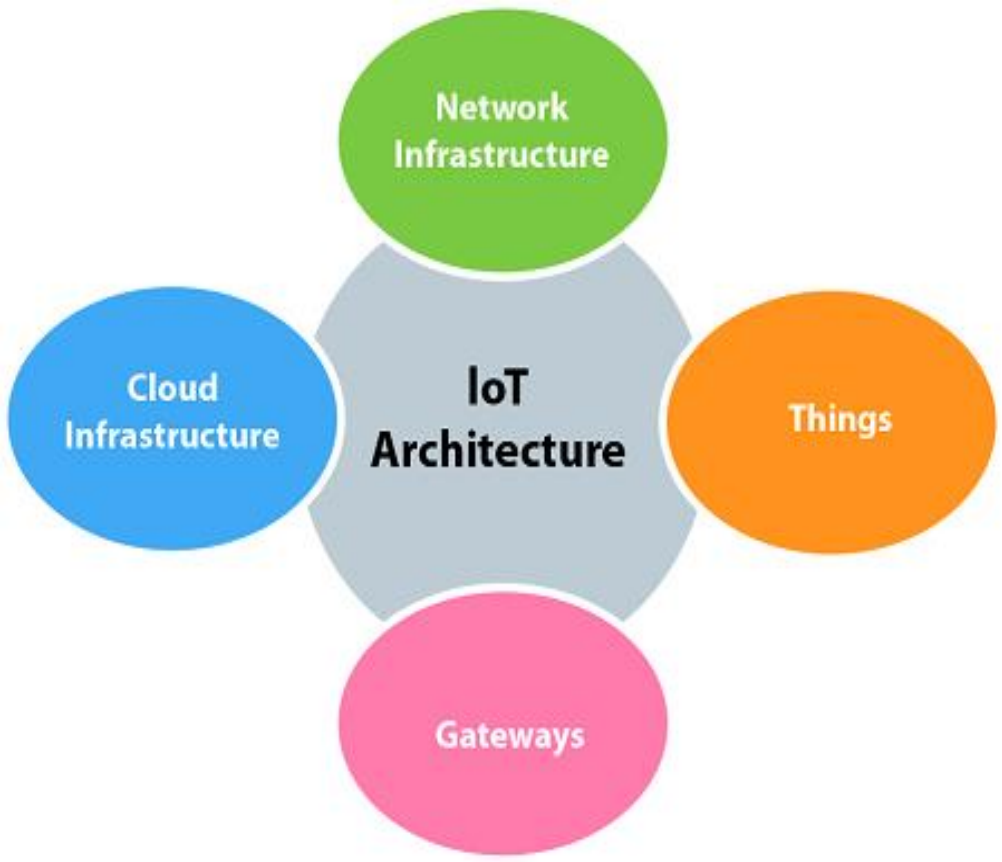
- **Automation and Control**
  - Enables automatic control of devices and processes.
  - Reduces human intervention and errors (e.g., smart thermostats, industrial robots).
- **Efficiency and Productivity**
  - Optimizes energy usage, time, and resources.
  - Boosts performance in sectors like agriculture (smart irrigation), manufacturing (predictive maintenance), and logistics (smart tracking).
- **Data Collection and Insights**
  - Continuously collects real-time data.
  - Helps in making data-driven decisions (e.g., health monitoring, traffic control).
- **Remote Monitoring and Management**
  - Monitor and control devices remotely (e.g., home automation, smart meters).
  - Increases convenience and safety.
- **Improved Quality of Life**
  - Smart homes, wearable health devices, and connected vehicles enhance comfort, safety, and health.
- **Cost Reduction**
  - Preventive maintenance and optimized operations can reduce operational costs in the long run.

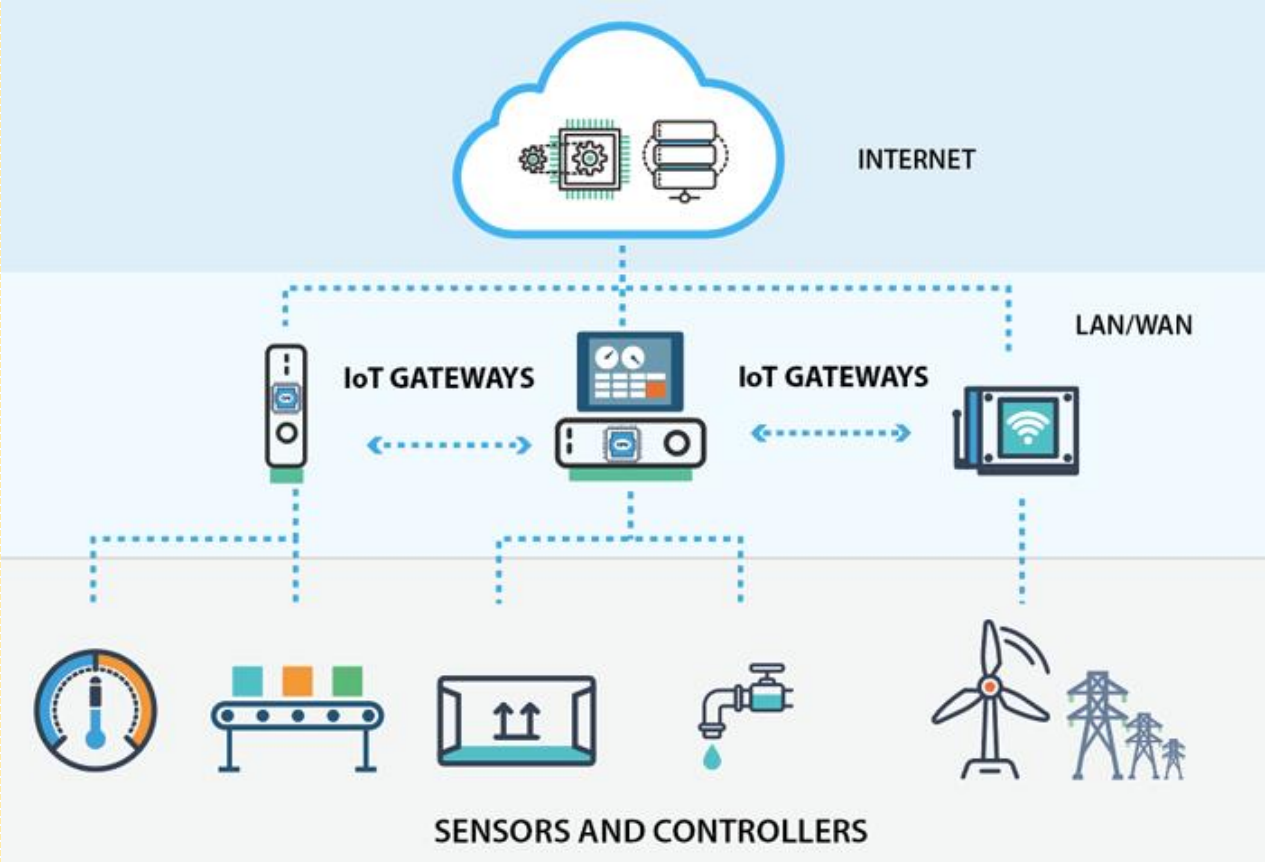
IoT Disadvantages

- **Security and Privacy Issues**
  - Vulnerable to cyberattacks, data breaches, and unauthorized access.
  - Lack of strong encryption in low-cost devices is a concern.
- **Complexity and Interoperability**
  - Integrating different IoT devices and platforms is complex.
  - No universal standard for communication protocols.
- **Dependence on Internet Connectivity**
  - Functionality is affected if the network is slow or down.
  - Offline operations are limited.
- **Data Overload and Management**
  - Massive volume of data requires robust processing, storage, and analytics systems.
- **High Initial Cost**
  - Setup of infrastructure, sensors, gateways, and analytics tools can be expensive.
- **Ethical and Social Concerns**
  - Constant surveillance may lead to ethical issues and reduced privacy.
  - Job displacement due to automation.

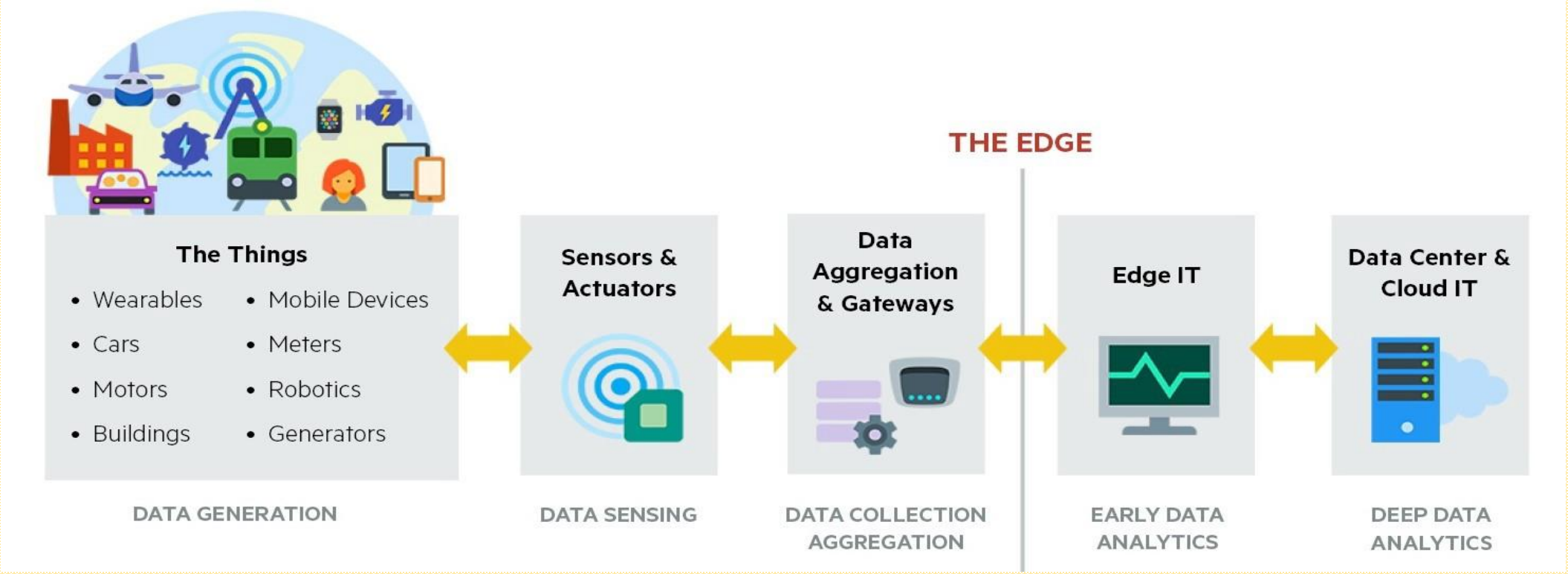
IoT Architecture Components

An IoT architecture is a mix of hardware and software components that interact together to make up a smart cyber-digital system. Interoperating with one another, these components make up a base for an IoT solution to be built upon. Before we dive into the details, let's get things straight: there's no one-size-fits-all approach to designing an IoT architecture. Still, the basic layout stays largely the same no matter the solution.





IoT Architecture



- **Devices:** This stage is about the actual devices in the IoT solutions. These devices could be sensors or actuators in the Perception layer. Those devices will generate data (in the case of sensors) or act on their environment (in the case of actuators). The data produced is converted in a digital form and transmitted to the internet gateway stage. Unless a critical decision must be made, the data is typically sent in a raw state to the next stage due to the limited resources of the devices themselves.

IoT Architecture

- **Internet gateways:** The **internet gateway** collects **raw data** from IoT devices and **pre-processes** it (like filtering or basic formatting) before sending it to the cloud. It can be built into the device or be a separate device that connects sensors to the Internet.

- **Edge or fog computing:** This layer processes **time-critical data** closer to where it's generated — at the **edge of the network** — for **faster response**. It handles **recent data** to detect urgent issues quickly, doing some **pre-processing** to reduce what needs to go to the cloud.

- **Cloud or data center:** In the final stage, all data is **stored, deeply analyzed, and managed** in the cloud or data center. Here, advanced tasks like **machine learning, big data analytics, and dashboards** run to extract valuable insights and support business decisions.

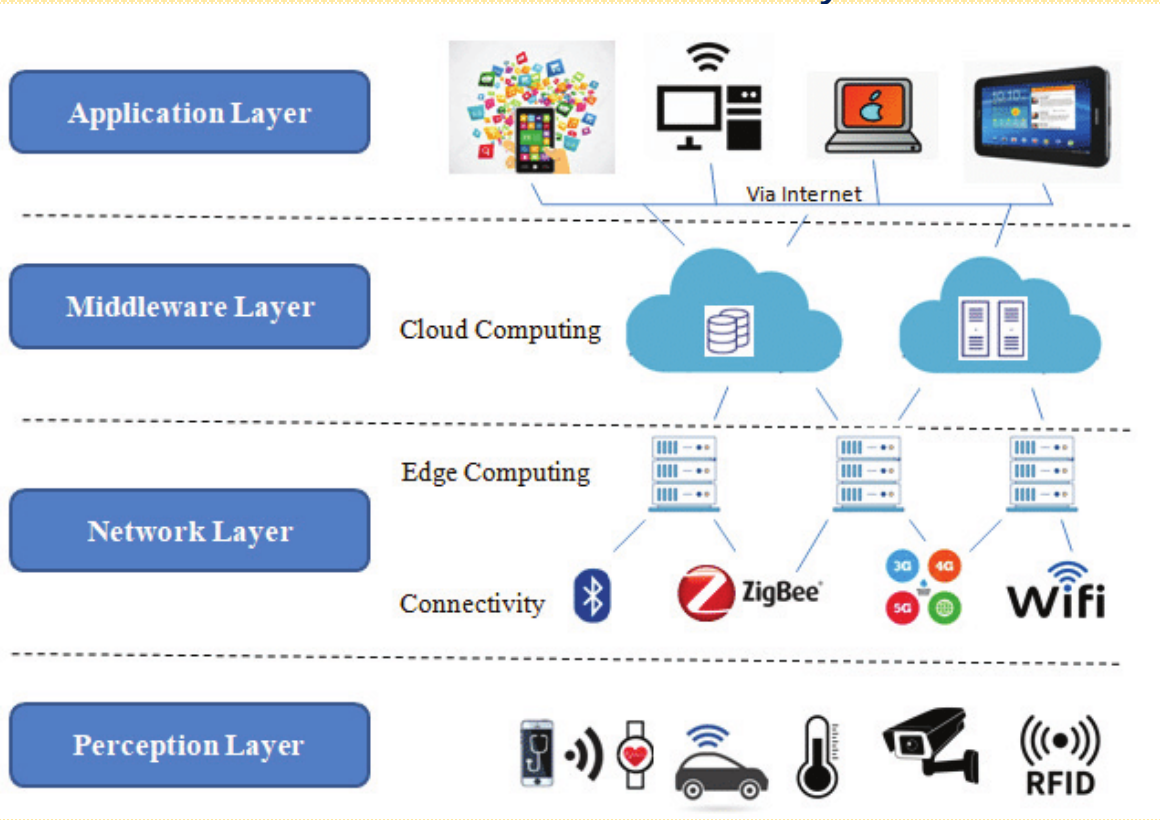
IoT Layered Architecture

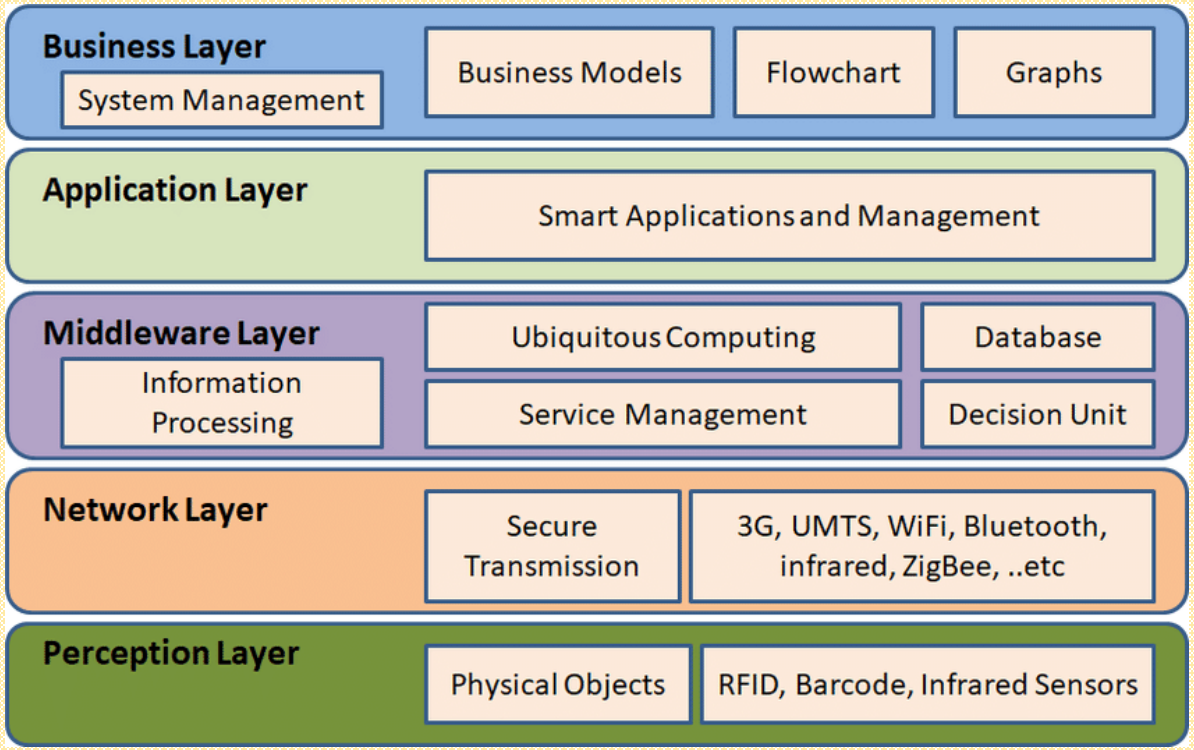
What is an IoT Architecture?

IoT Architecture refers to the structured framework that defines how IoT devices, networks, platforms, and applications interact with each other to collect, process, and use data effectively. It outlines the layers and components required for seamless communication, data management, and intelligent decision-making in an IoT ecosystem.

Typical Layers of IoT Architecture

A standard IoT architecture is commonly divided into **4 or 5 layers**. Here's a **4-layer model** for clarity:





IoT Layered Architecture

1. Connected Objects (Sensing Layer)

- This is the foundation of any IoT system. It includes physical devices such as:
  - **Sensors** (e.g., temperature, motion, humidity)
  - **Actuators** (devices that can act, like turning off a valve or light)
- **Function**:

  Sensors collect real-world data.

  Actuators respond to commands and interact with the physical environment.
- **Example**: A soil moisture sensor detects dryness and an actuator turns on the irrigation pump.

IoT Layered Architecture

2. Internet Gateway (Network Layer)

- Acts as a **bridge** between connected objects and processing systems.
- **Components**:
  - **Data Acquisition System (DAS)**: Collects and digitizes sensor data.
  - **Internet Gateway**: Routes data to the network for further processing.
- **Function**:
  - Aggregates data from multiple devices.
  - Converts analog signals into digital data.
  - Transmits data over local or wide-area networks.
- **Example**: A gateway collects data from sensors and transmits it over Wi-Fi or LTE to processing units.

IoT Layered Architecture

3. Edge IT Systems (Edge Computing Layer)

- **Description**: Performs **local processing** and **pre-analysis** of the data before it reaches the cloud.
- **Technologies**:
  - **Edge servers or local processors**
  - **Machine learning** for real-time decisions
  - **Visualization tools** for user-friendly output
- **Function**:
  - Reduces network load by filtering, summarizing, or acting on data locally.
  - Provides faster response times for critical applications.
- **Example**: A machine learning model on an edge device detects equipment failure and alerts the operator instantly.

IoT Layered Architecture

4. Data Centers and Cloud Storage (Application/Business Layer)

- **Description**: The **final stage** where data is stored, deeply analyzed, and integrated with other systems.
- **Technologies**:
  - **Cloud computing platforms** (e.g., AWS, Azure, Google Cloud)
  - **Big data analytics tools**
  - **Data warehousing**
- **Function**:
  - Long-term storage of sensor data.
  - In-depth analysis and integration with business intelligence systems.
  - Supports dashboards, reporting tools, and enterprise decision-making.
- **Example**: Cloud-based analytics combine sensor data with weather data to predict crop yield in smart agriculture.

IoT Designing

An IoT application structure have basically four design paradigms.

1. **Physical Design**
   - IoT Device
   - IoT Layer Architectures (Protocols)

2. **Logical Design**
   - IoT Functional Blocks
   - IoT Communication Models
   - IoT Communication APIs

3. **IoT Enabling Technologies**
   - Wireless Sensor Networks
   - Cloud Computing
   - Big Data Analytics
   - Communication Protocols
   - Embedded System
   - Artificial Intelligence & Machine Learning

IoT Designing

4. IoT Levels

- IoT Level 1
- IoT Level 2
- IoT Level 3
- IoT Level 4
- IoT Level 5
- IoT Level 6

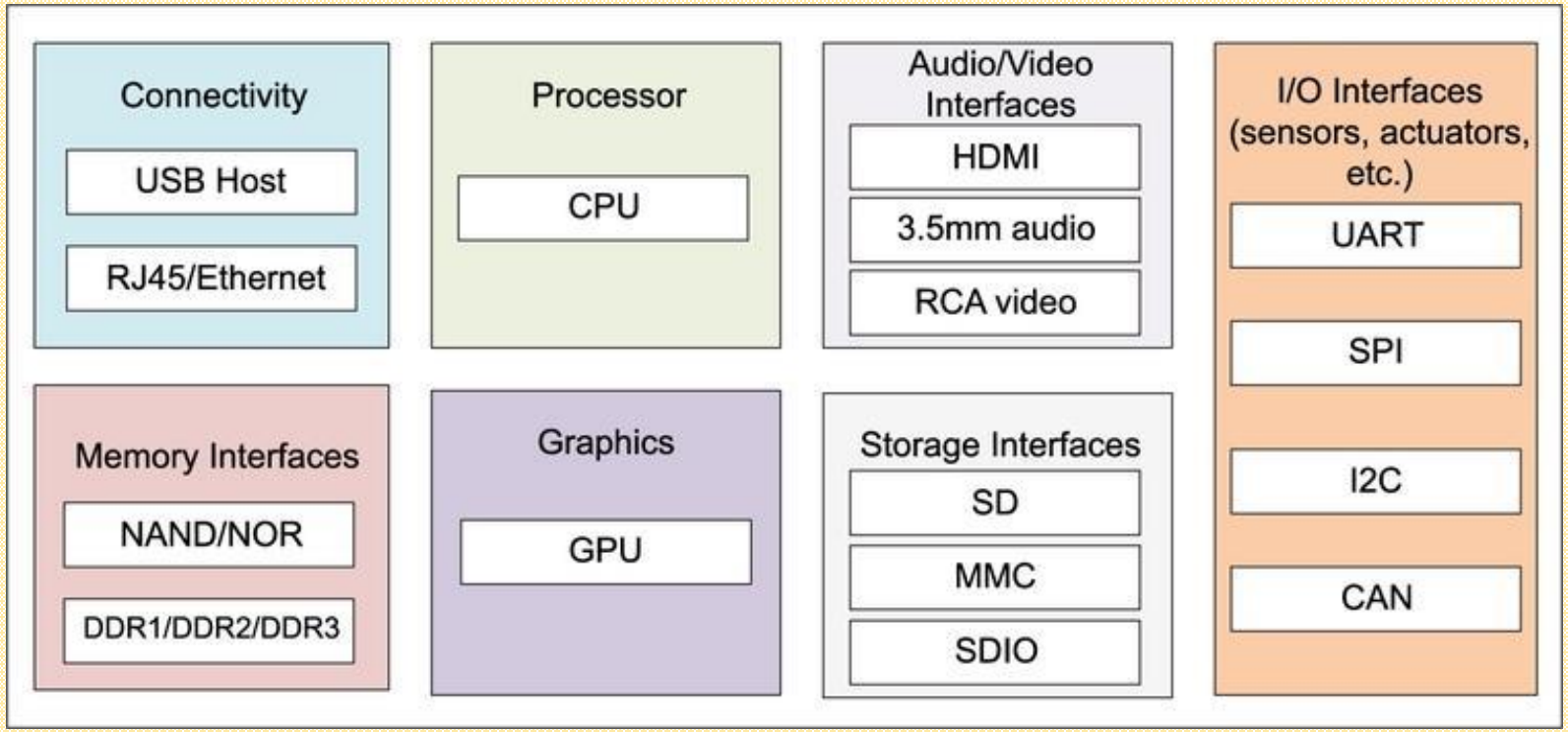
IoT Physical Design

IoT Things:

The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.

An IoT device may consist of several interfaces for communication to other devices both wired and wireless. These includes

(i) I/O interfaces for sensors,
(ii) Interfaces for internet connectivity
(iii) Memory and storage interfaces
(iv) Audio/Video interfaces.



IoT Physical Design

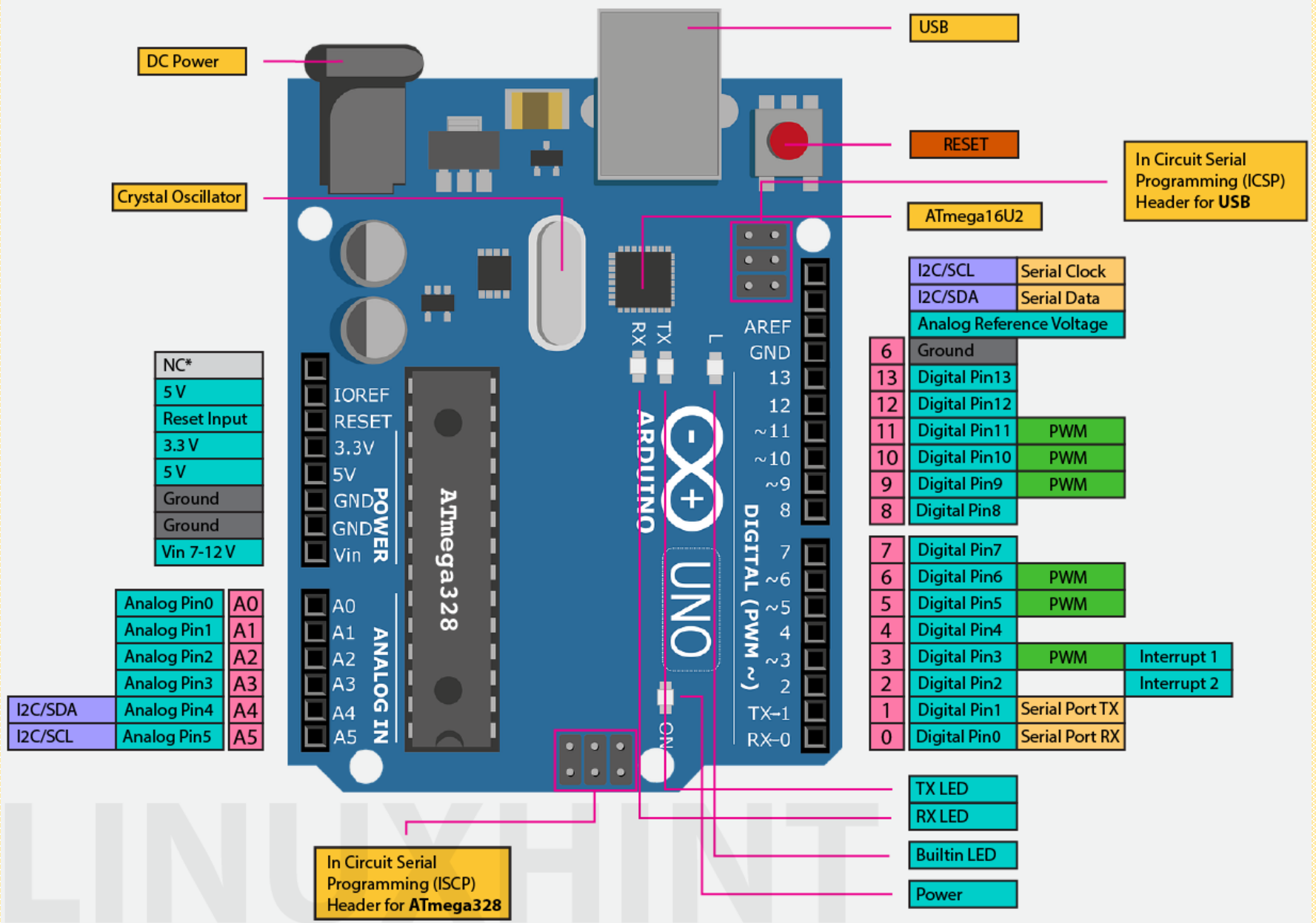
UART/RS232 - UART is usually an individual (or part of an) integrated circuit (IC) used for **serial communications over a computer or peripheral device serial port**. One or more UART peripherals are commonly integrated in microcontroller chips. Specialized UARTs are used for automobiles, smart cards and SIMs.

SPI - Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others.

I²C—Shorthand for inter-integrated circuit or I squared C, this is a low-speed serial bus that is used for sending data from one chip to another on the same PC board or over short cables between two pieces of equipment. I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

CAN Controller Area Network Interface Bus (CAN)—This is controller area network serial bus in an interface used to link micros together in small networks. It is used in automotive and industrial applications. The usual cable is twisted pair and a ground. Speeds range from 10 kbps to 1 Mbps with 20 kbps being typical. The data is sent in specifically formatted frames as determined by a standard protocol. It uses start and stop bits like the RS-232 and some similar control codes defined in ASCII. It also includes error detection and correction capability, so it is very reliable.

IoT Physical Design



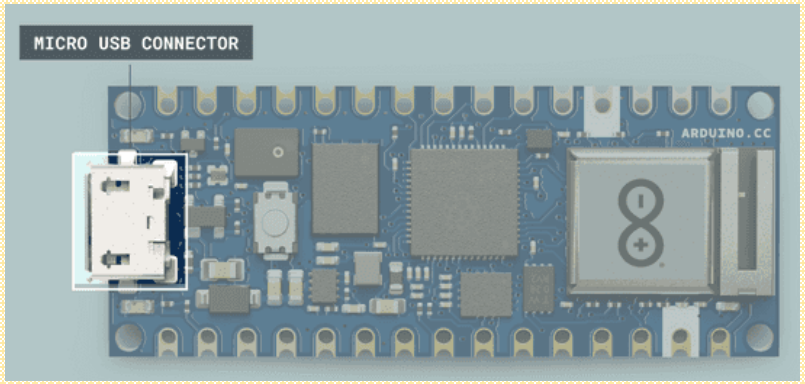
IoT Physical Design

IoT Physical Design

Powering Alternatives

Arduino boards have **five** options in which they can be powered:

1. Powering via USB connector
2. Powering via the onboard barrel jack connector (if available on the board)
3. Powering via the onboard battery connector (if available on the board)
4. Powering via the VIN (Voltage In) pin
5. Powering via the 3V3/5V pin*



*Powering your board via the 3V3/5V pins is not recommended, as it can damage your board's voltage regulator.

Powering via USB connector

- The USB connector provides a regulated 5V line to power the board's electronics. However, **5V from the USB connector can also power external components through the 5V pin** that can be found in Arduino boards.
- Arduino boards that run at 5V use the USB-regulated 5V line directly, boards that run at 3V3 regulate the 5V line from the USB connector to 3V3 using their onboard voltage regulator. Output current rating from the 5V pin will vary, depending on the 5V power source.
- Current from USB ports of computers is usually limited to 500mA.

IoT Physical Design

Powering via the onboard barrel jack connector (if available on the board)

The voltage line from the barrel jack connector is regulated in Arduino boards using their onboard voltage regulator; usually, it is first regulated to 5V and then regulated again to 3V3 in most Arduino boards. The **recommended voltage and current ratings for external regulated DC power supplies** connected to the barrel jack connector are summarized in the table below:

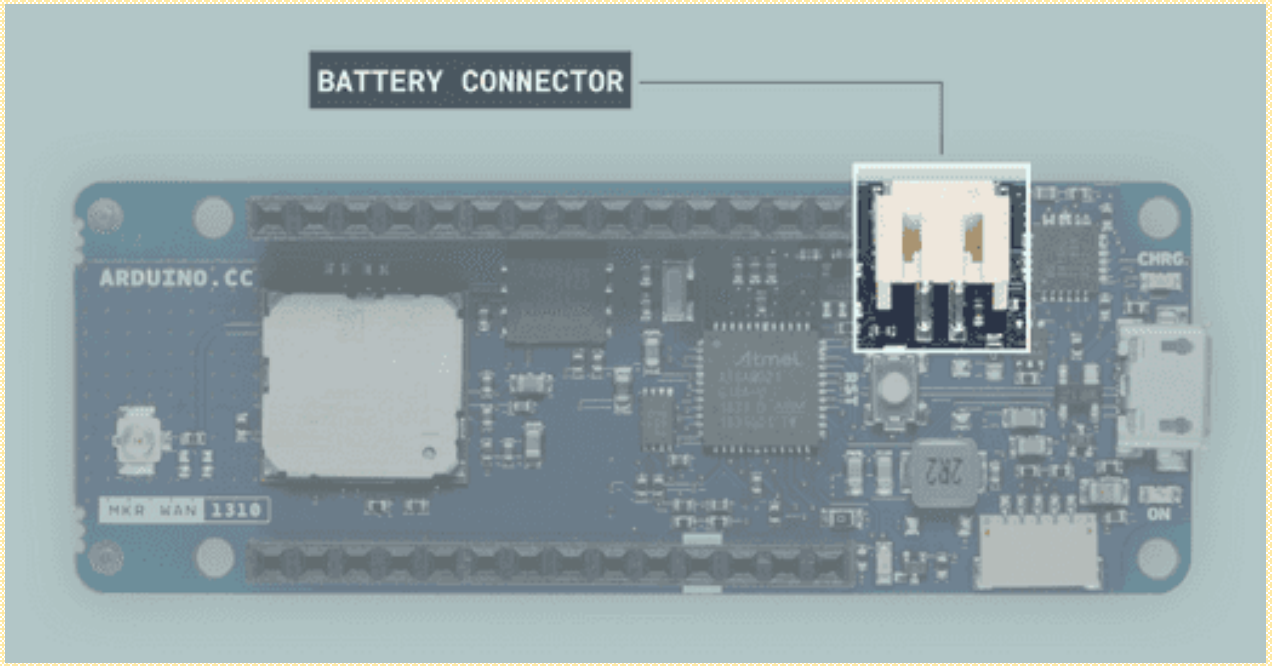
Board	External Power Supply Voltage (V)	External Power Supply Current (A)
Arduino UNO Rev3	7-12	1
Arduino UNO WiFi Rev2	7-12	1.5
Arduino Leonardo	7-12	1
Arduino Mega 2560 Rev3	7-12	1
Arduino Due	7-12	1.5
Arduino Zero	5-18	1

IoT Physical Design

Powering via the onboard battery connector (if available on the board)

Some Arduino boards have an **onboard battery connector** to connect a battery to the board and use it as its primary or secondary power supply. The Arduino boards with an onboard battery connector are the following:

- Arduino Portenta H7
- Arduino Nicla Sense ME
- Arduino Nicla Vision
- Arduino MKR NB 1500
- Arduino MKR Vidor 4000
- Arduino MKR WiFi 1010
- Arduino MKR ZERO
- Arduino MKR WAN 1310
- Arduino MKR GSM 1400



Battery connector of the Arduino MKR WAN 1310 board

Pro family boards use a 3-pin, 1.2mm SMD ACH battery connector; MKR family boards use a 2-pin, 2mm SMD PH battery connector.

IoT Physical Design

Powering via the VIN (Voltage In) pin

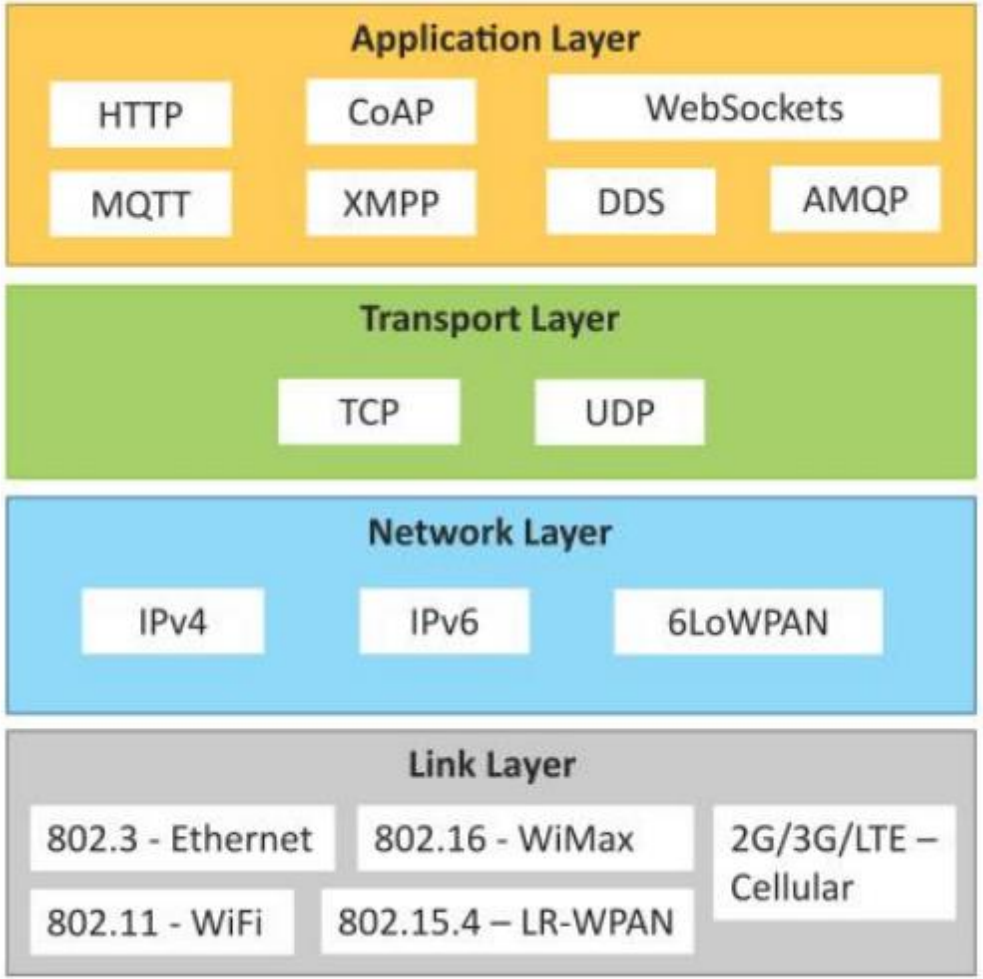
- The VIN pin in Arduino boards is a power pin with a dual function.
- This pin can work as a **voltage input for regulated external power supplies** that do not use a barrel jack connector.
- This pin can also work as a **voltage output when an external power supply is connected to the barrel jack connector** present in some Arduino boards.
- An important consideration is that the VIN pin is connected directly to the input pin of the onboard voltage regulator on Arduino boards.
- Since the VIN pin is directly connected to the voltage regulator, the **VIN pin does not have reverse polarity protection**.
- Use the VIN pin carefully to avoid damaging your Arduino board since it does not have reverse polarity protection.
- The **minimum and maximum voltages** that can be applied to the VIN pin are determined by the onboard voltage regulator on Arduino boards, varying from board to board. Those voltages are summarized in the table below:

Board	VIN Voltage (V)
UNO Mini	5-18
UNO Rev3	7-12
UNO WiFi Rev2	7-12
UNO Rev3 SMD	7-12
Leonardo	7-12
Mega 2560 Rev3	7-12
Due	7-12
Micro	7-12
Zero	5-18
Portenta H7	5
Nicla Sense ME	5
Nano RP2040 Connect	5-18
MKR NB 1500	5-7
MKR GSM 1400	5-7
MKR Vidor 4000	5-7
MKR WiFi 1010	5-7
MKR Zero	5-5.5
MKR1000 WIFI	5-5.5
MKR WAN 1300	5-5.5
MKR WAN 1310	5-7
Nano	7-12
Nano Every	7-18
Nano 33 IoT	5-18
Nano 33 BLE	5-18
Nano 33 BLE Sense	5-18

IoT Physical Design

The **IoT Protocol Layer Model** organizes various communication protocols used in IoT into **layered architecture**, similar to the OSI model. Each layer serves a specific role in **enabling communication, processing, and service delivery** across the IoT ecosystem.

Provides services to end-users and applications	MQTT, CoAP, HTTP, XMPP, AMQP, DDS
Reliable or lightweight data delivery	TCP, UDP, DTLS (for secure UDP)
Routing, addressing, and internetworking	IPv6, RPL, 6LoWPAN, ICMP
Node-to-node data transfer and error handling	IEEE 802.15.4, IEEE 802.11 (Wi-Fi), Ethernet, MAC, LPWAN
Data sensing, physical signal transmission	RFID, Zigbee, Bluetooth, NFC, Sensors, Actuators



IoT Physical Design

Protocols:

- **802.3-Ethernet:** IEEE802.3 is collection of wired Ethernet standards for the link layer. E.g.:
  - 802.3 uses co-axial cable;
  - 802.3i uses copper twisted pair connection;
  - 802.3j uses fiber optic connection;
  - 802.3ae uses Ethernet over fiber.

- **802.11-WiFi:** IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. E. g.
  - 802.11a operates in 5GHz band,
  - 802.11b and 802.11g operates in 2.4GHz band,
  - 802.11n operates in 2.4/5GHz band,
  - 802.11ac operates in 5GHz band,
  - 802.11ad operates in 60Ghz band.

- **802.16 - WiMAX:** IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. Fixed WiMAX provide data rates from 1.5 Mb/s to 1Gb/s. WiMax operates on frequency band 2-11 GHz and covers up to 50 km area. While mobile WiMax operates on 2.3 GHz, 2.5 GHz, 3.5 GHz and covers between 5-15 Kms area.

IoT Physical Design

- **802.15.4-LR-WPAN:**
  - **LR-WPAN** stands for **Low-Rate Wireless Personal Area Network**.
  - It is a wireless network designed for **short-range, low-data-rate, low-power, and low-cost communication** between devices, especially **sensors and embedded systems** in IoT applications.
  - LR-WPAN is standardized under **EEE 802.15.4.**
  - This standard defines the **physical (PHY)** and **MAC (Medium Access Control)** layers for low-rate wireless networks.
  - Provides data rate from 40kb/s to 250kb/s.
  - Typically ranges between 10-100 mtrs.

- **2G/3G/4G-Mobile Communication:** These are the different generations of mobile communications standards:
  - 2G including GSM and CDMA,
  - 3G including UMTS and CDMA3000,
  - 4G including LTE,
  - Data rates from 9.6kb/s(2G) up to100Mb/s(4G).

Aspects	ZigBee	WiFi	Bluetooth
standard	802.15.4	802.11a,b,g	802.15.1
application	automation, control	Web, e-mail, video	replacement of cableing
data rate	50 - 60 kbyte	> 1 Mbyte	> 250 kbyte
battery lifetime	> 1000	1 -5	1 -7
network size	65535	32	7
bandwith (kb/s)	20 - 250	11000	720
transmission distance	100+ m	100 m	10 m
advantage	reliability, performance, cost	data rate, flexibility	cost, comfortable

IoT Physical Design

LoRa

- LoRa, which is an abbreviation of “long range”, from RF technology for an LPWAN.
- LoRa is one of the most prominent wireless technologies in the low-power wide-area network (LPWAN) family.
- LoRa is a patented energy-efficient wireless communication protocol that achieves very low-power and very long-range transmissions, of over 10 km in line-of-sight, trading-off data rate, and time-on-air.
- LoRa’s duty cycle is limited by regional regulations because it operates using unlicensed sub-GHz radio bands, mostly on the 433, 868, and 915 MHz frequency bands.
- The data transmission rate supported by LoRa varies from 300 bps to 50 kbps, depending on spreading factor (SF) and channel bandwidth settings.
- Taking into account this and the low-transmission bandwidth, LoRa is naturally most suitable for applications where transmissions are sparse in time and payloads are relatively small.

IoT Physical Design

Network/Internet Layer:

Responsible for sending IP datagrams from source network to destination network. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- **IPv4:** Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32-bit address. Allows total of 2^{32}addresses. These addresses got exhausted in the year of 2011. IPv4 has been succeeded by IPv6. IP protocols establish connection over packet networks, but do not guarantee delivery of packets.

- **IPv6:**It is newest version of IPv4. Internet Protocol version6 uses 128-bit address scheme and allows 2^{128} addresses.

- **6LOWPAN**:(IPv6 over Low Power Wireless Personal Area Network) operates in 2.4 GHz frequency range and data transfer 250 kb/s. It brings IP protocol for low power devices having limited processing capability.

Transport Layer:

Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.

Protocols:

- **TCP:** Transmission Control Protocol used by web browsers (along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. TCP also provides the error detection capability so that duplicate packets can be discarded and loss packets are retransmitted. Avoids n/w congestion and congestion collapse. It also ensures flow control to adjust the mismatch of transmission and receiving speed of sender and receiver.

- **UDP:** User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery, ordering of messages and duplicate elimination.

Application Layer:

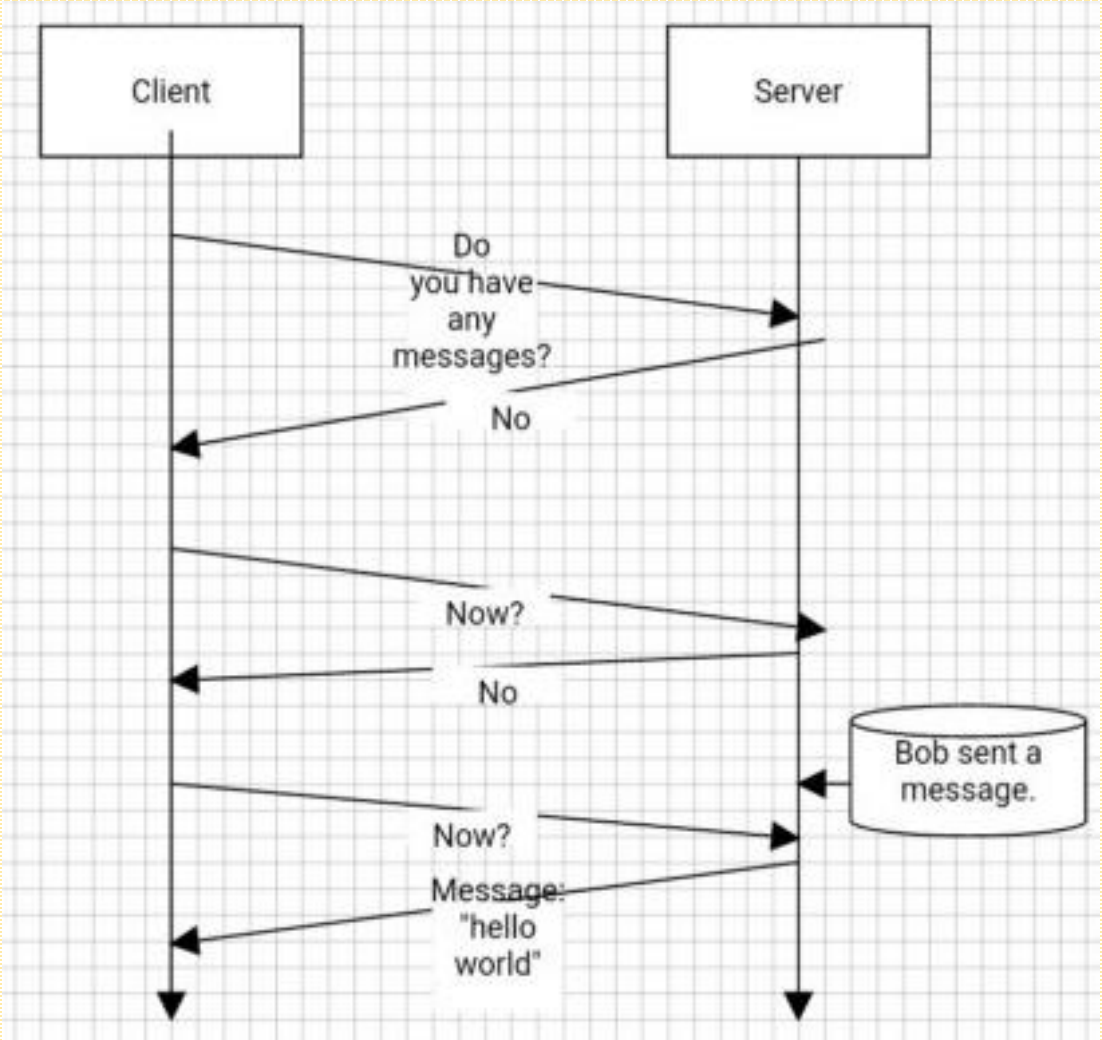
Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

HTTP

- Hyper Text Transfer Protocol that forms foundation of WWW.
- Follow request response model Stateless protocol.
- It was designed for communication between web browsers and web servers, but it can also be used for other purposes.
- HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response.
- HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests.
- Defaults to port **80** (ws://) and **443** (wss://) for secure.
- Applications of HTTP/HTTPs are Web Browsing, RESTful APIs, Web Services, IoT Communication (Limited)

If you are a developer, you probably know what HTTP **(or HTTPS)** is. It is seen every day, in your browser. A request is needed to respond; **you to constantly ask the server if there are new messages in order to receive them.**

You should also know that HTTP allows you to have different types of requests such as post, get or put, each with a different purpose.



Application Layer:

Web Socket:

It allows full duplex communication over a single socket connection. This protocol defines a full duplex communication from the ground up. Web sockets take a step forward in bringing desktop rich functionalities to the web browsers.

The main features of web sockets are as follows −

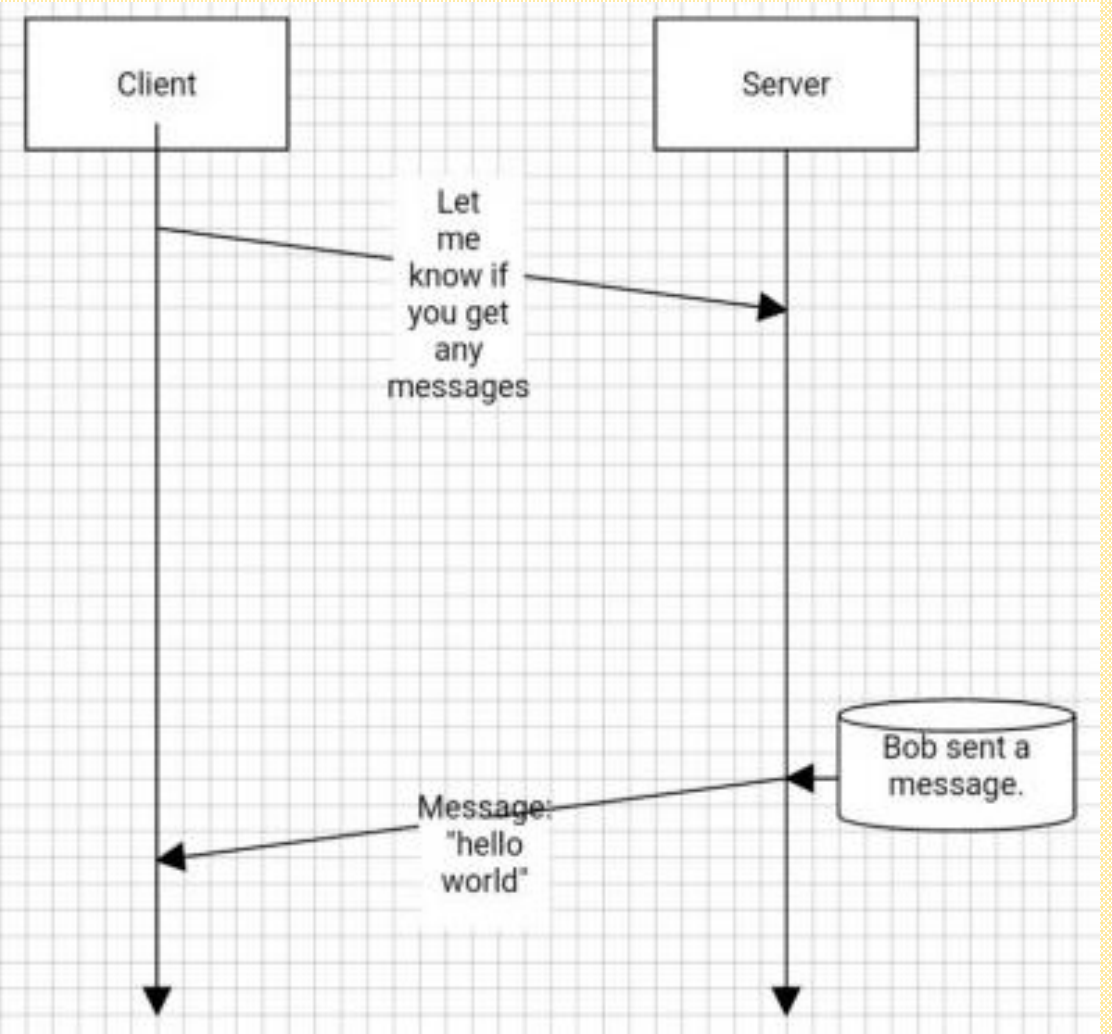
- Real time communication between web servers and clients is possible with the help of this protocol.
- Cross platform standard for real time communication between a client and the server.
- The biggest advantage of Web Socket is it provides a two-way communication (full duplex) over a single TCP connection.
- Both client and server can send/receive messages independently.
- Header Overhead is minimal compared to HTTP; Performance is better than HTTP.
- Defaults to port **80** (ws://) and **443** (wss://) for secure.
- Ideal for real-time applications like chat and gaming.

How WebSocket Works

1. Client sends an **HTTP request** with an Upgrade: websocket header.
2. Server responds with a **"101 Switching Protocols"** response.
3. The connection is upgraded from HTTP to WebSocket.
4. Now, both client and server can **push messages anytime** until the connection is closed.

Web Sockets on the other hand don't need you to send a request in order to respond. They allow bidirectional data flow so you just have to listen for any data.

You can just listen to the server and it will send you a message when it's available.



Web Socket Applications

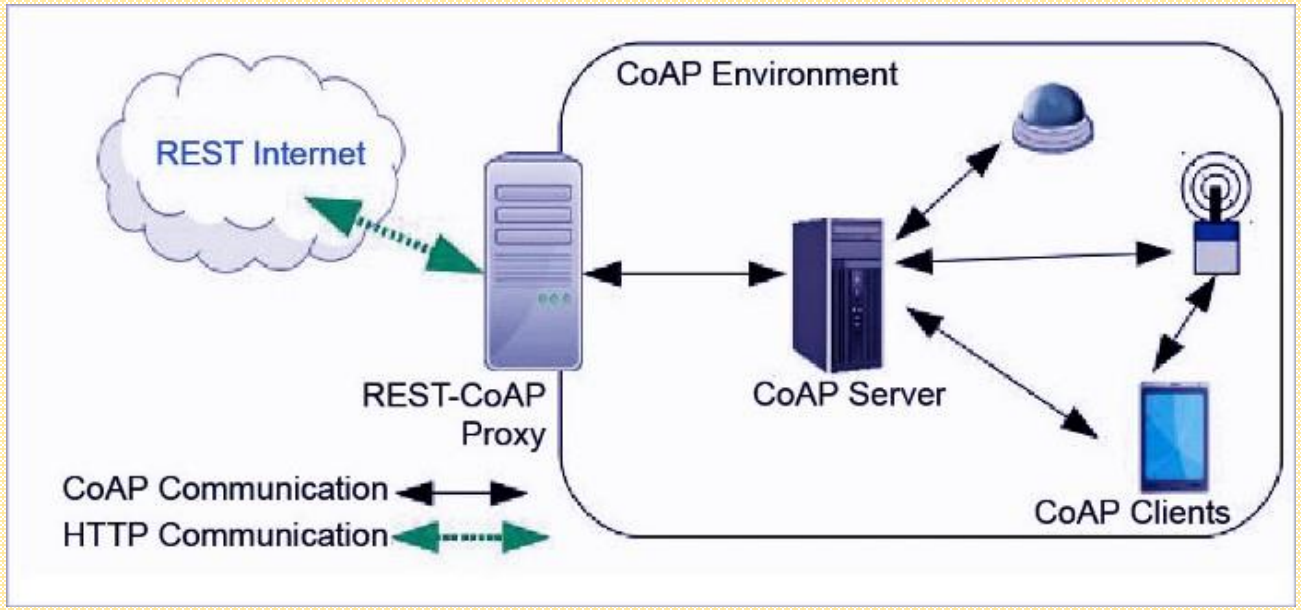
- ❖ Online multiplayer games
- ❖ Live stock market dashboards
- ❖ Real-time GPS tracking
- ❖ Collaborative document editing (e.g., Google Docs)
- ❖ IoT device communication

Application Layer:

CoAP

CoAP (Constrained Application Protocol) is a **lightweight web transfer protocol** designed for use in **constrained devices and networks**, such as **IoT systems**, where resources like power, bandwidth, memory, and processing are limited.

- It's similar to HTTP but optimized for **low-power** and **low-bandwidth** environments.
- Uses client server architecture. Like HTTP, CoAP is a document transfer protocol.
- Unlike HTTP, CoAP is designed for the needs of constrained devices.
- CoAP packets are much smaller than HTTP TCP flows.
- CoAP runs over UDP, not TCP.
- CoAP allows UDP broadcast and multicast to be used for addressing.
- CoAP follows a client/server model. Clients make requests to servers, servers send back responses.
- Clients may GET, PUT, POST and DELETE resources.



- CoAP is designed to interoperate with HTTP and the RESTful web at large through simple proxies. Because CoAP is datagram based, it may be used on top of SMS and other packet based communications protocols.

CoAP vs HTTP

Aspect	CoAP	HTTP
Protocol	UDP	TCP
Payload	Small, binary	Large, text-based
Use Case	IoT, constrained networks	General web applications
Header Size	Very small	Large
Reliability	Optional (confirmable msgs)	Reliable (TCP-based)

Example Application: Smart Home IoT System

Scenario: A smart temperature sensor reporting to a central home controller.

- The controller sends a GET request via CoAP to the temperature sensor.
- The sensor replies with the current temperature value.
- The controller can also **observe** the sensor, receiving updates whenever the temperature changes.

CoAP URI Example:

coap://192.168.0.10/sensors/temperature

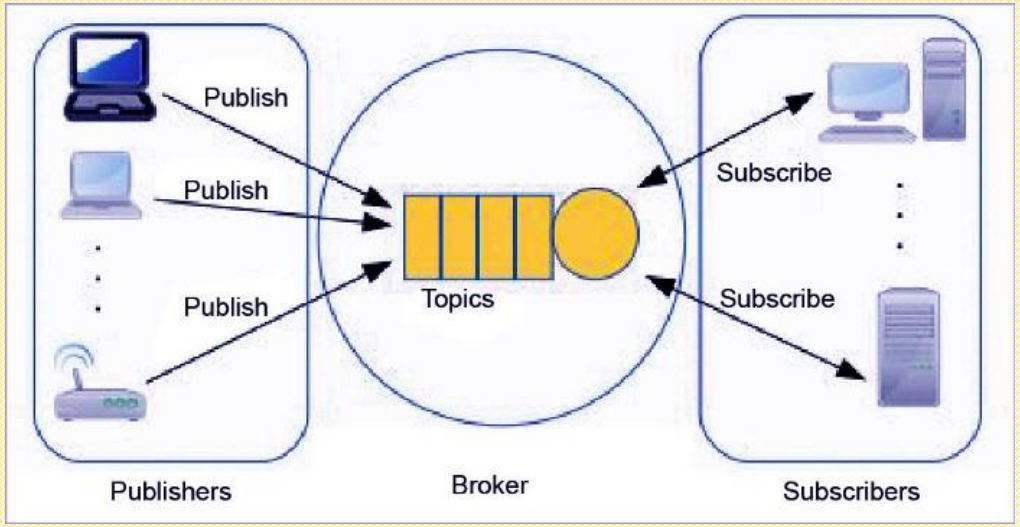
Why Use CoAP in IoT?

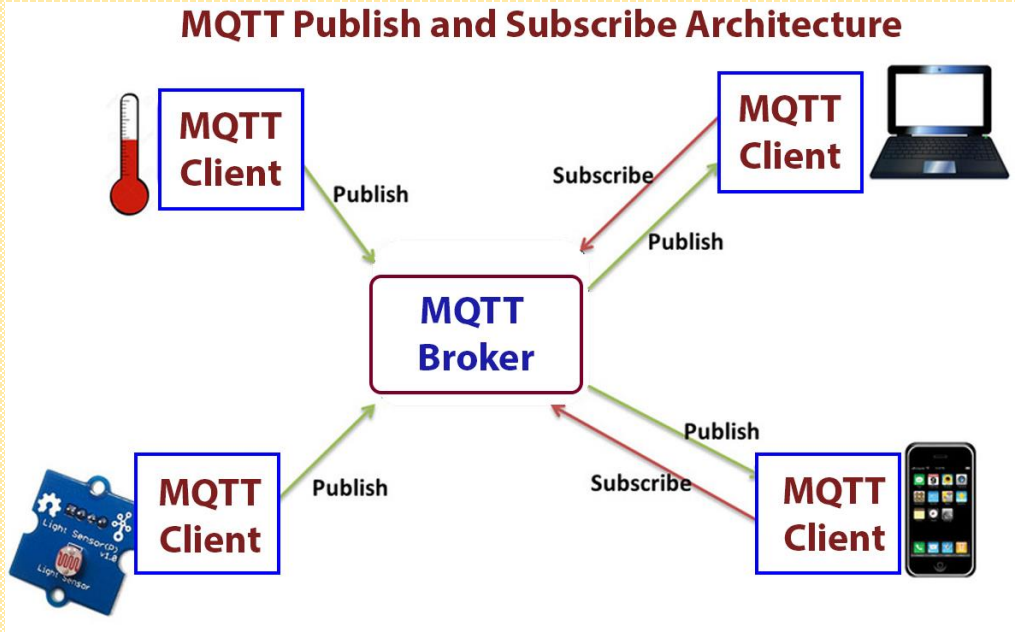
- Ideal for **low-power devices** (battery-operated).
- Suited for **lossy networks** (like LPWAN or 6LoWPAN).
- Enables **efficient communication** with microcontrollers and embedded systems.

Application Layer:

MQTT

- MQTT is a lightweight, publish/subscribe messaging protocol designed for low-bandwidth, high-latency, or unreliable networks — making it ideal for IoT (Internet of Things) applications.
- Uses client server architecture.
- Minimal overhead — ideal for constrained devices.
- Ensures reliable transmission over networks.
- Supports QoS Levels and ensure message delivery as needed (0- At most once, 1-At least once, 2-Exactly once).
- MQTT is a publish/subscribe messaging protocol designed for lightweight M2M communications.
- It was originally developed by IBM and is now an open standard.
- MQTT has a client/server model, where every sensor is a client and connects to a server, known as a broker, over TCP. MQTT is message oriented.





Application Layer:

MQTT Architecture

1. **Publisher**: Sends messages to a topic.
2. **Subscriber**: Receives messages from topics it subscribed to.
3. **Broker**: Central server that handles message routing between publishers and subscribers.

Example Use Case: Smart Agriculture

Scenario: A soil moisture sensor sends data to an MQTT broker. A farmer's smartphone app subscribes to this topic and gets real-time updates on soil conditions.

- **Publisher**: Soil moisture sensor
- **Broker**: Mosquitto MQTT broker (e.g., running on Raspberry Pi or cloud)
- **Subscriber**: Farmer's mobile app or dashboard

Common MQTT Brokers

Eclipse Mosquitto (Open source), **HiveMQ, EMQX, AWS IoT Core** (MQTT supported), **Google Cloud IoT**

Application Layer:

MQTT vs CoAP

Aspect	MQTT (Message Queuing Telemetry Transport)	CoAP (Constrained Application Protocol)
Protocol Type	Publish/Subscribe messaging protocol	RESTful request/response protocol (like HTTP)
Transport Layer	TCP/IP	UDP/IP
Message Model	Asynchronous, event-driven (via Broker)	Synchronous (client-server), Optional async via Observe
Architecture	Centralized (Broker-based)	Decentralized (peer-to-peer capable)
Security	TLS/SSL	DTLS (Datagram TLS)
Overhead	Medium (lightweight, but uses TCP)	Very Low (binary over UDP)
Reliability	Built-in with QoS (0, 1, 2)	Optional (Confirmable/Non-confirmable msgs)
Resource Usage	More suitable for slightly higher-resource devices	Ideal for very constrained devices
Best Use Cases	Telemetry, real-time messaging, remote monitoring	Resource-constrained IoT (e.g., sensors in agriculture)
Multicast Support	No	Yes (built-in support)

Application Layer:

XMPP

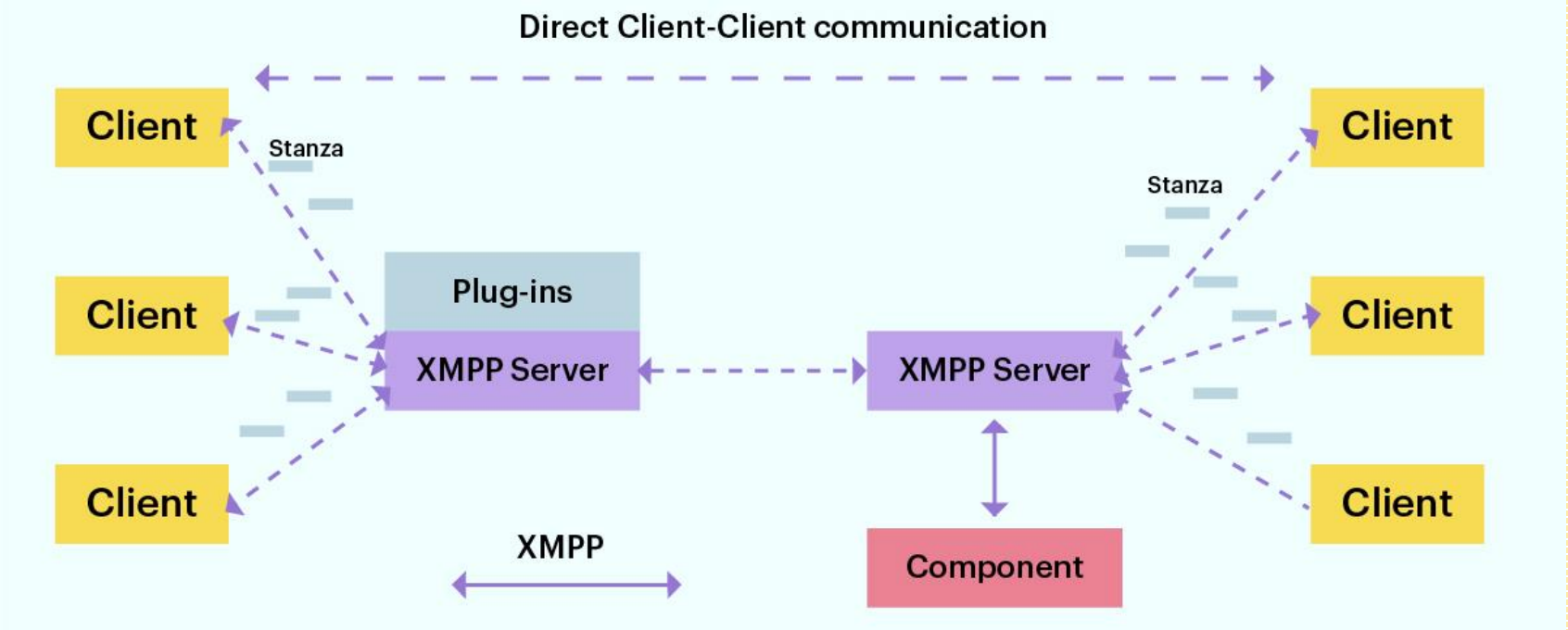
- Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- The XMPP protocols are free, open, public, and easily understandable (Defined by the **IETF** in RFCs like **RFC 6120, 6121, 6122)**
- **It is based on Client–Server** model with support for server-to-server communication.
- It runs over **TCP**, uses **port 5222** (for client) and **5269** (for server).
- New features can be added without breaking the core functionality.
- XMPP applications:
  - Instant Messaging (Google Talk (legacy), WhatsApp (initially), Jabber)
  - IoT Communication (Lightweight messaging for constrained devices)
  - Enterprise Messaging (Private chat servers (e.g., Openfire, ejabberd))
  - Gaming (Real-time player communication

  and remote systems monitoring, web services, lightweight middleware, cloud computing, and much more.

Application Layer:

How XMPP Works (Simplified):

1. Client connects to XMPP server (e.g., user@domain.com)
2. Server authenticates and maintains a session
3. Users exchange messages using XML stanzas (message, presence, iq)
4. Presence updates and subscriptions are handled in real time



Application Layer:

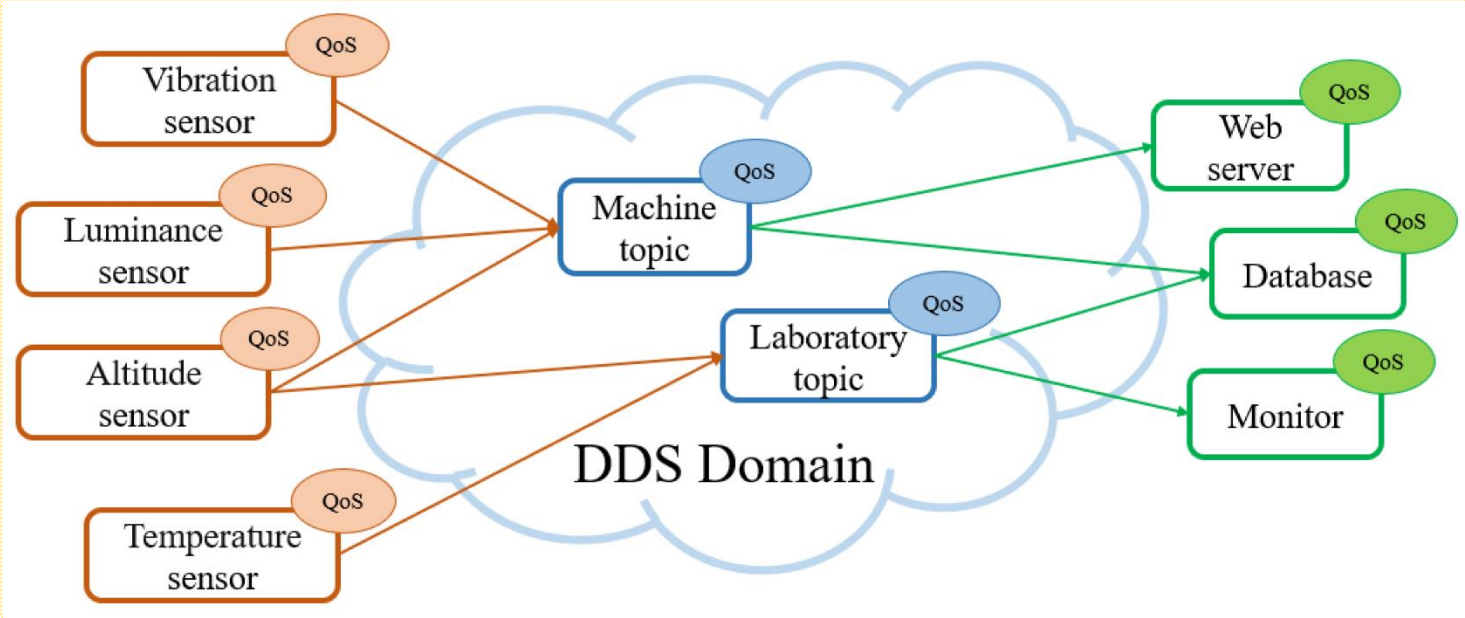
DDS

- **DDS (Data Distribution Service)** is a middleware **communication protocol** designed for **real-time**, **scalable**, and **high-performance** data exchange between devices and applications in distributed systems.
- The protocol uses broker-less architecture in the Internet of Things, unlike MQTT and CoAP protocols.
- It is an IoT protocol developed by Object Management Group for Machine to Machine Communication.
- It uses a publish-subscribe methodology for exchanging data.
- It uses multicasting to bring high-quality QoS to the applications.
- It follows a **publish-subscribe model** and is widely used in systems where **low latency**, **deterministic behavior**, and **quality of service (QoS)** are critical — such as autonomous vehicles, industrial IoT, robotics, aerospace, and defense.

Application Layer:

Key Features of DDS:

- **Real-time Communication:** Offers low-latency data delivery with predictable timing.
- **Publish-Subscribe Model:** Decouples data producers (publishers) and consumers (subscribers).
- **Quality of Service (QoS):** Users can define reliability, durability, latency, deadline, etc.
- **Decentralized Architecture:** No central broker needed; peers discover and communicate directly.
- **Automatic Discovery:** Nodes automatically detect and connect with compatible publishers/subscribers.
- **Scalability:** Supports thousands of nodes/devices and large data volumes.
- **Built-in Security:** Supports authentication, encryption, and access control.



Application Layer:

How DDS Works – Step by Step:

1. **Domain Participants:** Applications join a communication domain by creating a domain participant.

2. Publishers & Subscribers

- Publishers send data; Subscribers receive data.
- Both define the **topic** they deal with (e.g., TemperatureSensorData).

3. Data Writers & Data Readers

- Inside a publisher: **DataWriter** pushes data to a topic.
- Inside a subscriber: **DataReader** receives topic data.

4. Topic-Based Communication: Communication happens over **topics**, defined by name and data type.

5. Discovery Mechanism

- DDS uses **Real-Time Publish-Subscribe (RTPS)** protocol to automatically discover data producers and consumers.

6. Quality of Service (QoS) Policies

- Each communication entity defines its QoS needs (e.g., reliable delivery, minimum delay).
- DDS matches compatible QoS between publishers and subscribers.

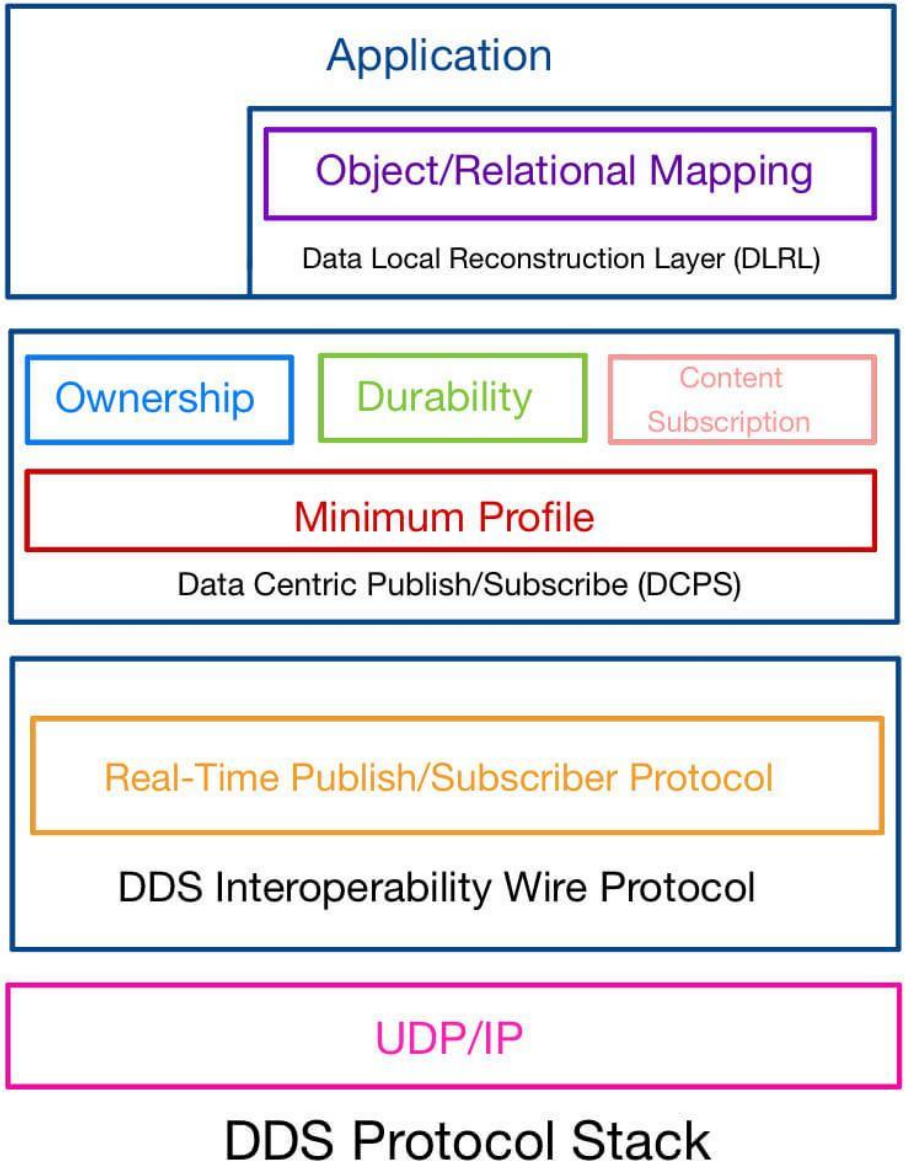
Application Layer:

DDS V1.2 API standard is language, operating system, and hardware architecture independent.

The DDS protocol stack. It consists of two primary layers:

- **DCPS (Data Centric Publish Subscribe) layer:** This layer handles the delivery of information to subscribers. The DCPS is a standard API for data-centric, topic-based, real-time publish/subscribe operations.
- **DLRL (Data Local Reconstruction Layer):** The DLRL provides an interface to DCPS functionalities. It allows for the sharing of distributed data among IoT-enabled devices. The DLRL is a standard API for creating object views from a collection of topics.

DDSI/RTPS V2.1 is a standard wire protocol, ensuring interoperability between different implementations of the DDS standard.



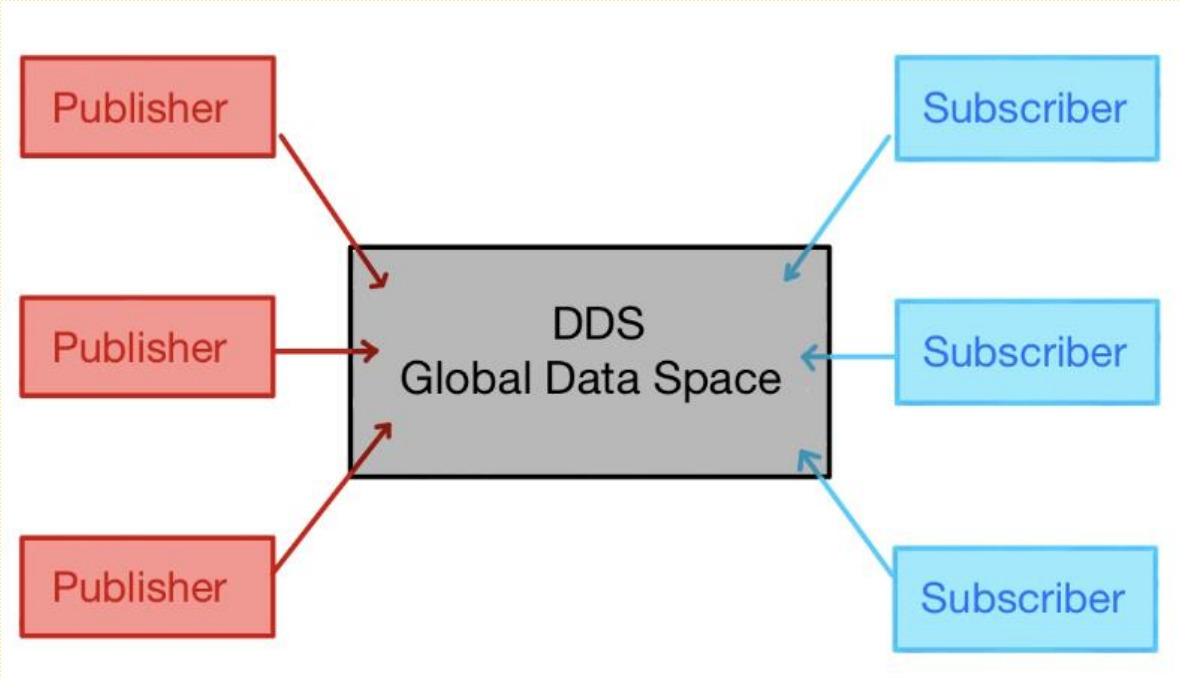
DDS Protocol Stack

Application Layer:

DDS Working

Data Distribution Service (or DDS) is a fully distributed Global Data Space, which makes it fully distributed to avoid introducing a single point of failure or bottleneck.

1. Applications can autonomously and asynchronously read as well as write data in the Global Data Space.
2. Since publishers and subscribers are dynamically discovered, they can join or leave the Global Data Space at any time.
3. Publishers and Subscribers show their intention to produce or consume a specific type of data, such as topics.
4. Subscriptions are matched by taking into account the topics with details like name, data type, etc.
5. All the subscriptions are dynamically matched, and the data flows from the publisher to the subscribers.



Application Layer:

DDS addresses the needs of applications like:

- Aerospace and defence,
- Air-traffic control,
- Autonomous vehicles,
- Medical devices,
- Robotics,
- Power generation,
- Simulation and testing,
- Smart grid management,
- Transportation systems, and other applications that require real-time data exchange.

Application Layer:

AMQP:

- AMQP stands for Advanced Messaging Queuing Protocol. It is a standard for cross platform messaging.
- Open-standard application layer protocol for message-oriented communication.
- Designed for reliable, secure, and interoperable messaging between distributed systems.
- Commonly used in enterprise and cloud applications.
- Supports both point-to-point and publish-subscribe model.
- Enterprise applications are being written in a number of dynamic languages such as Ruby, Perl and Python.

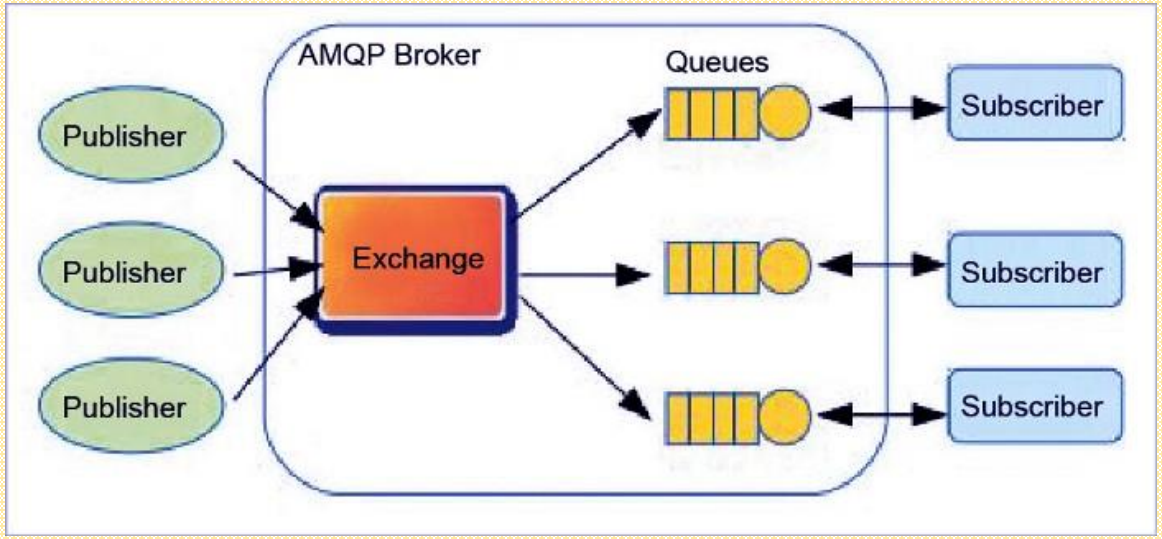
AMQP Key Features:

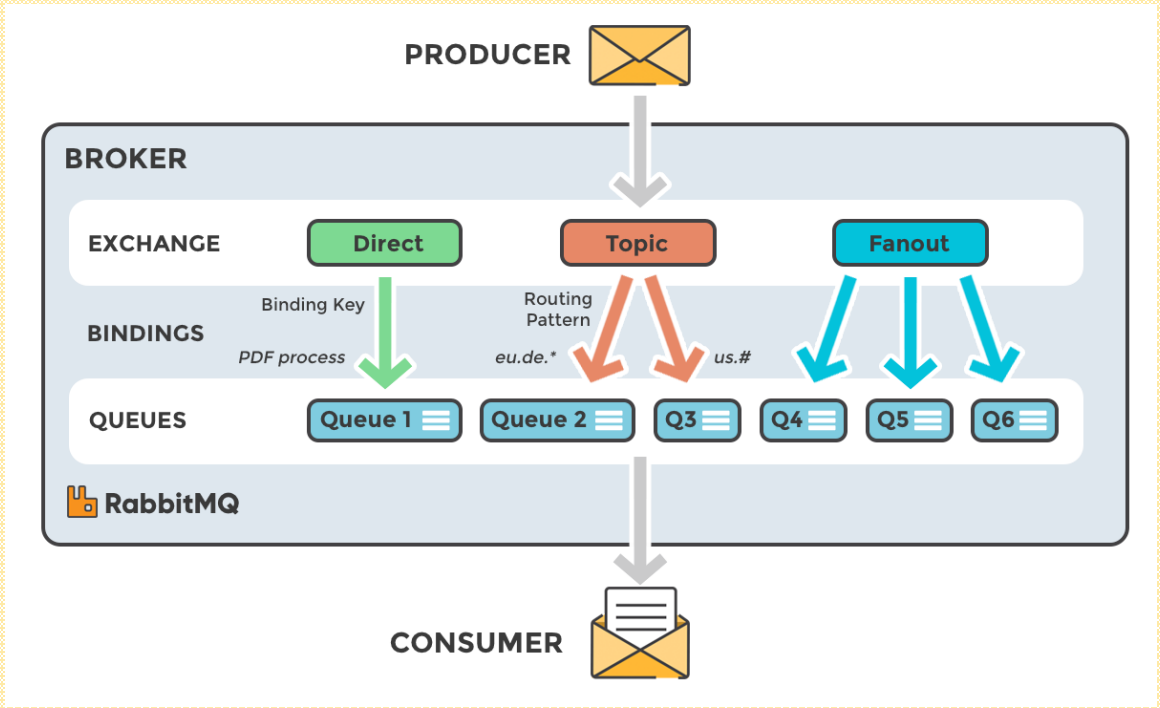
- **Reliable Message Delivery**: Supports message acknowledgment, persistence, and fault tolerance.
- **Interoperability**: Works across different platforms and languages.
- **Secure Communication**: Supports TLS, SASL, and encryption.
- **Routing Capabilities**: Uses exchanges (Direct, Topic, Fanout, Headers) to route messages.
- **Transactional Messaging**: Allows atomic message operations.
- **Queue Management**: Supports durable queues, priority queues, and delayed messaging.
- **Flow Control**: Prevents overload on message consumers.

Application Layer:

Core Components:

- **Producer:** Application that sends the messages.
- **Consumer:** Application that receives the messages.
- **Queue:** Buffer that stores messages.
- **Message:** Information that is sent from the producer to a consumer through RabbitMQ.
- **Connection:** A TCP connection between your application and the RabbitMQ broker.
- **Channel:** A virtual connection inside a connection. When publishing or consuming messages from a queue - it's all done over a channel.
- **Exchange:** Receives messages from producers and pushes them to queues depending on rules defined by the exchange type. To receive messages, a queue needs to be bound to at least one exchange.
- **Binding:** A binding is a link between a queue and an exchange.





Application Layer:

Types of Exchanges in AMQP

- **Direct:** The message is routed to the queues whose binding key exactly matches the routing key of the message. For example, if the queue is bound to the exchange with the binding key *pdfprocess, a message published to* the exchange with a routing key *pdfprocess is routed to that queue.*
- **Fanout:** A fanout exchange routes messages to all of the queues bound to it.
- **Topic:** The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
- **Headers:** Headers exchanges use the message header attributes for routing.

Use Cases:

- Banking and financial transaction systems.
- E-commerce platforms for order and inventory handling.
- Cloud-based backend systems.
- Enterprise micro services integration.
- Secure healthcare messaging systems.

Feature	CoAP	MQTT	AMQP	XMPP	DDS
Full Form	Constrained Application Protocol	Message Queuing Telemetry Transport	Advanced Message Queuing Protocol	Extensible Messaging and Presence Protocol	Data Distribution Service
Model	Client–Server (RESTful)	Publish–Subscribe (Brokered)	Publish–Subscribe (Brokered)	Publish–Subscribe & Presence-based	Publish–Subscribe (Brokerless)
Transport	UDP	TCP	TCP	TCP	TCP, UDP, Shared Memory
Message Format	Binary, compact	Lightweight binary	Binary (AMQP 1.0)	XML-based	Binary
QoS Support	Basic (Confirmable, Non-confirmable)	Yes (QoS 0, 1, 2)	High (Reliable delivery, transactions)	Basic (some via extensions)	Very High (20+ customizable policies)
Security	DTLS	SSL/TLS	SSL/TLS	TLS, SASL	TLS, RTPS security plugins
Best Use Case	Resource-constrained IoT devices	Sensor data transmission	Enterprise-level messaging	Chat, presence, pub-sub in real-time	Mission-critical real-time systems
Architecture Type	RESTful (Web-like)	Centralized broker	Centralized broker	Centralized or federated	Decentralized (peer-to-peer possible)
Scalability	Moderate	High	High	Moderate	Very High
Latency	Very Low	Low	Medium	Medium	Ultra Low
Interoperability	High (Web-compatible)	Medium	High	High	High

Based on Requirements

Requirement	Suggested Protocol
Battery-powered devices, low overhead	MQTT or CoAP
Secure, enterprise-level message queue	AMQP
Real-time performance with QoS tuning	DDS
Chat, messaging, presence-based systems	XMPP
RESTful interface with low power	CoAP

Example Matching Applications

Application	Recommended Protocol
Smart Street Lighting	MQTT / CoAP
Industrial Machine Monitoring	DDS / MQTT
Secure Online Order Processing System	AMQP
Smart Irrigation via Mobile App	CoAP / MQTT
Hospital Patient Alert System	XMPP / MQTT
Autonomous Drones Coordination	DDS

LOGICAL DESIGN OF IOT

The logical design of an IoT system refers to an abstract representation of entities and processes without going into the low-level specifies of implementation. it uses Functional Blocks, Communication Models, and Communication APIs to implement a system.

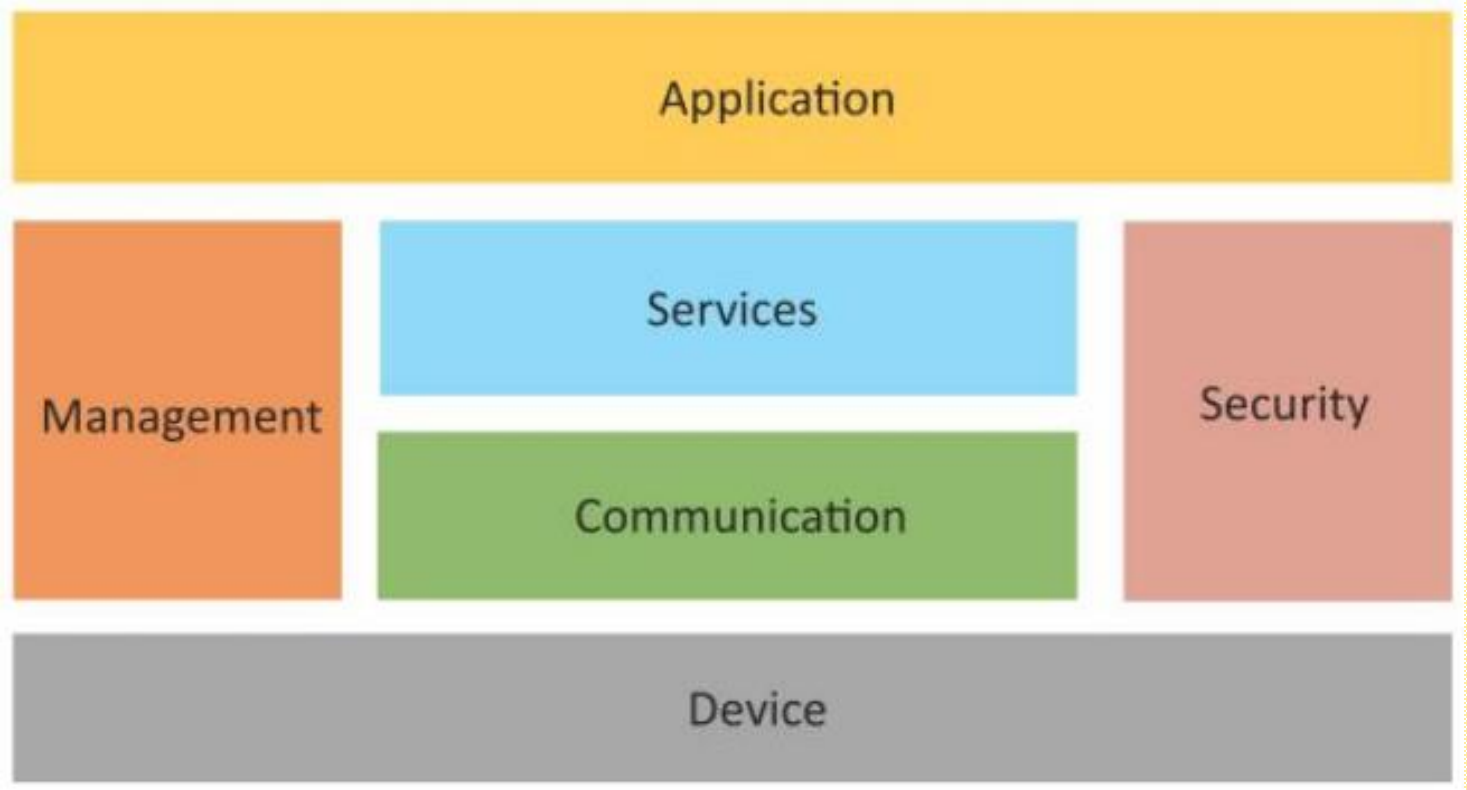
IoT Functional Blocks

IoT Communication Models

IoT Communication APIs

IoT Functional blocks

An IoT system consist number of functional blocks like Devices, services, communication, security, and application that provides the capability for sensing, actuation, identification, communication, and management.



LOGICAL DESIGN OF IOT

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

- **Device:** These devices are used to provide sensing and monitoring control functions that collect the data from the outer environment.
- **Communication:** This block handles the communication between the client and cloud-based server and sends/receives the data using protocols.
- **Services:** This functional block provides some services like monitoring and controlling a device and publishing and deleting the data and restore the system.
- **Management:** This functional block provides various functions that are used to manage an IoT system.
- **Security:** This block is used to secure an IoT system using some functions like authorization, data security, authentication, 2 step verification, etc.
- **Application:** It is an interface that provides a control system that use by users to view the status and analyse of system.

LOGICAL DESIGN OF IOT

IoT Communication Models

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

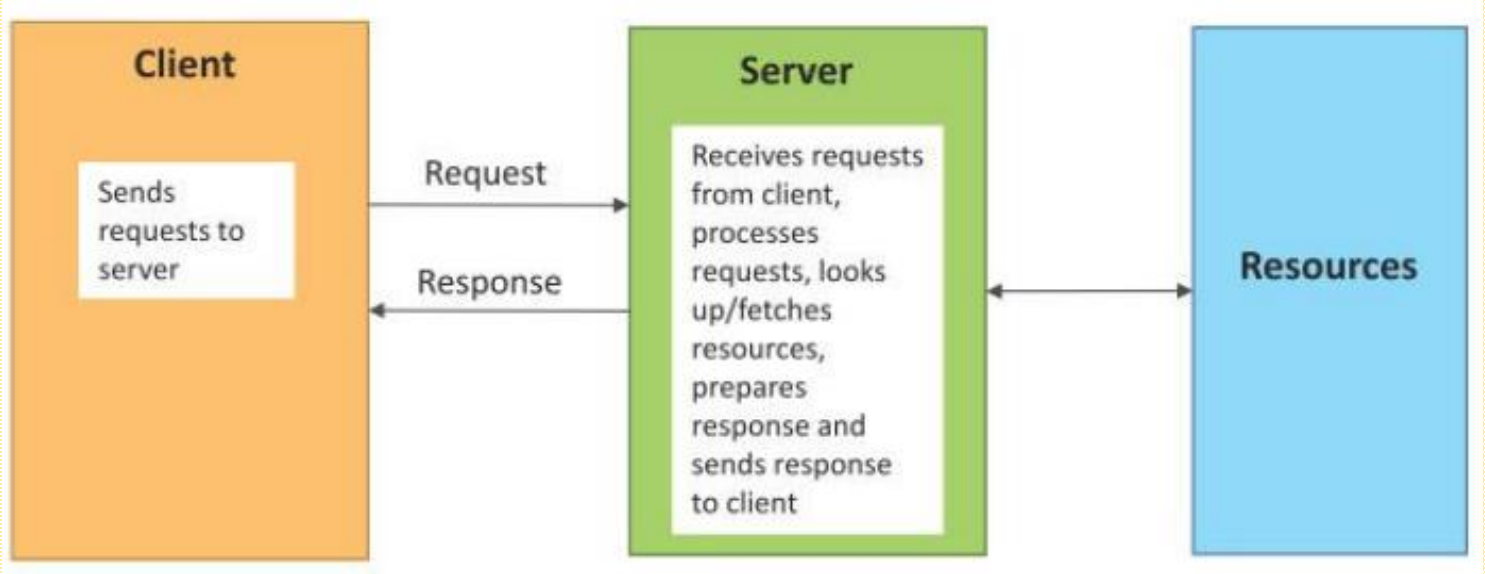
Request-Response Communication Model

This model is a communication model in which a client sends the request for data to the server and the server responds according to the request. when a server receives a request it fetches the data, retrieves the resources and prepares the response, and then sends the data back to the client.

In simple terms, we can say that in the request-response model server send the response of equivalent on the request of the client. in this model, HTTP works as a request-response protocol between a client and server.

Example

When we search a query on a browser then the browser submits an HTTP request to the server and then the server returns a response to the browser(client).



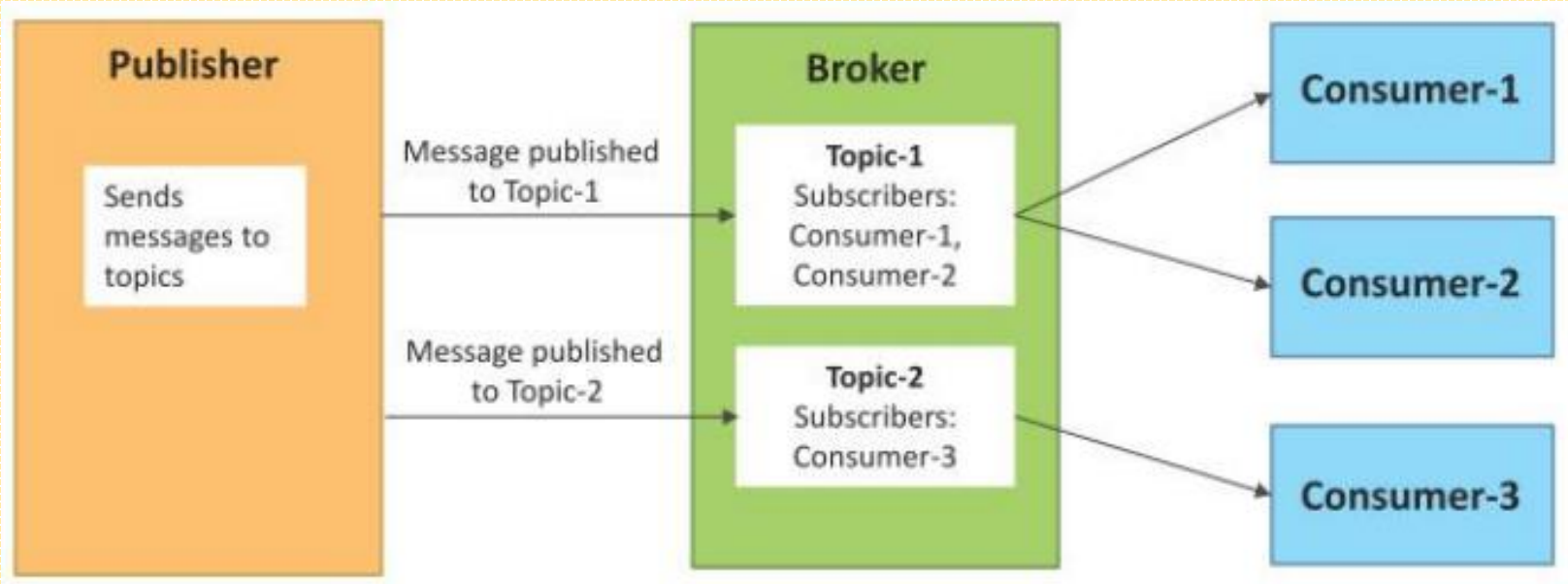
LOGICAL DESIGN OF IOT

Publish-Subscribe Communication Model

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

Example

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



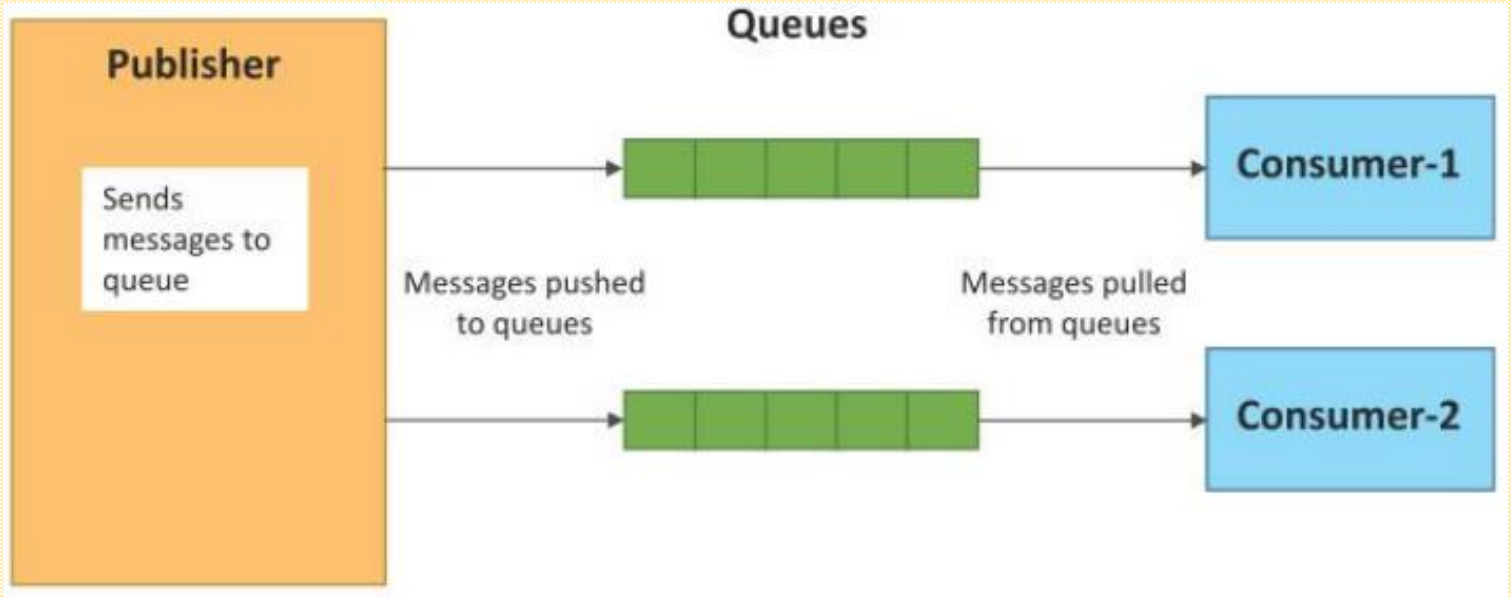
LOGICAL DESIGN OF IOT

Push-Pull Communication Model

- Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.
- Queues help in decoupling the messaging between the producers and consumers.
- Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate at which the consumers pull data.

Example

When we visit a website we saw a number of posts that published in a queue and according to our requirements, we click on a post and start reading it.



LOGICAL DESIGN OF IOT

Push-Pull Communication Model Example

Smart Agriculture System

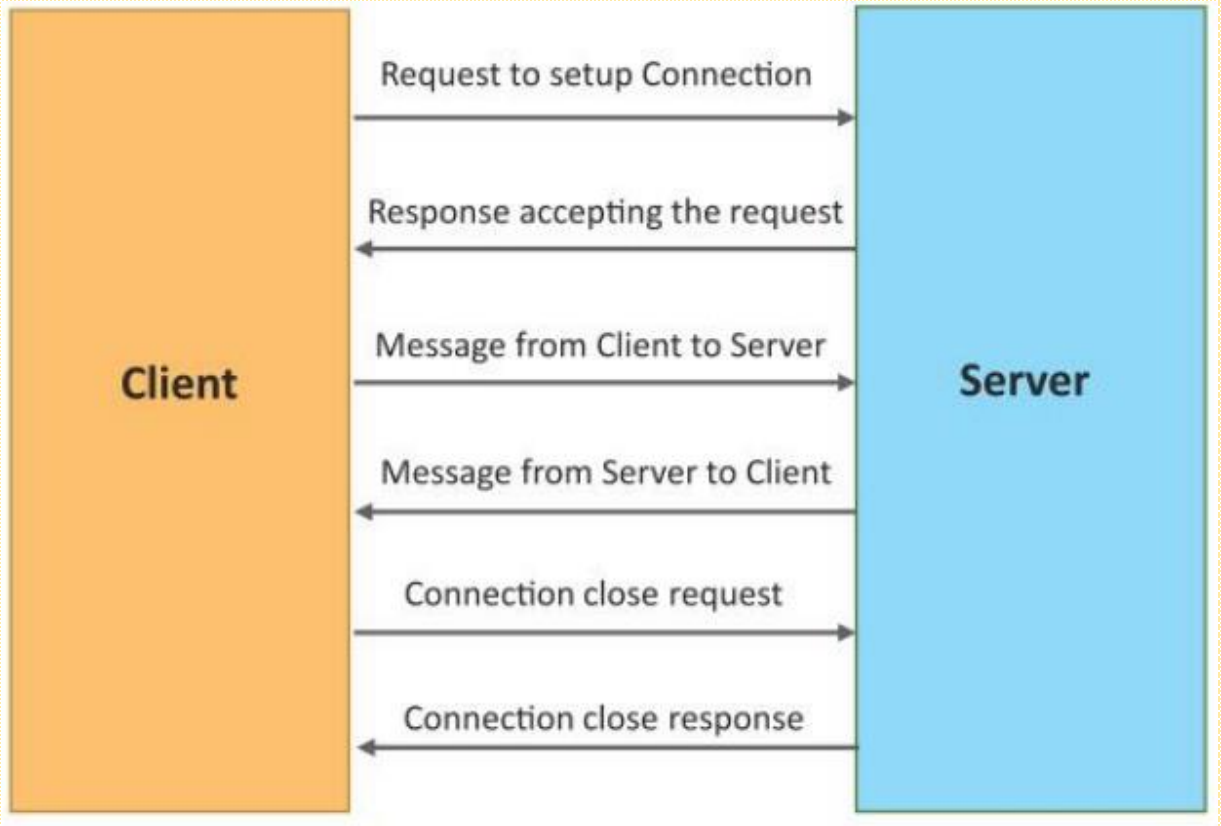
- **Sensors** in a field (e.g., soil moisture, temperature) collect data every 10 minutes.
- These sensors **push** their readings to a **central broker/cloud database** (like AWS IoT or a local edge server).
- A **farmer's dashboard app** periodically **pulls** the data to display graphs or send irrigation commands.
- **Push:** IoT devices **send data automatically** to a broker/cloud (producer role).
- Applications **request the latest data** when needed (consumer role).
- **Example:** Soil sensors pushing data; farmer's app pulling updates on demand.

Function	Technology
Data Push	MQTT, HTTP POST, CoAP
Data Pull	HTTP GET, REST API, GraphQL
Broker Layer	Kafka, RabbitMQ, AWS IoT

LOGICAL DESIGN OF IOT

Exclusive Pair Communication Model

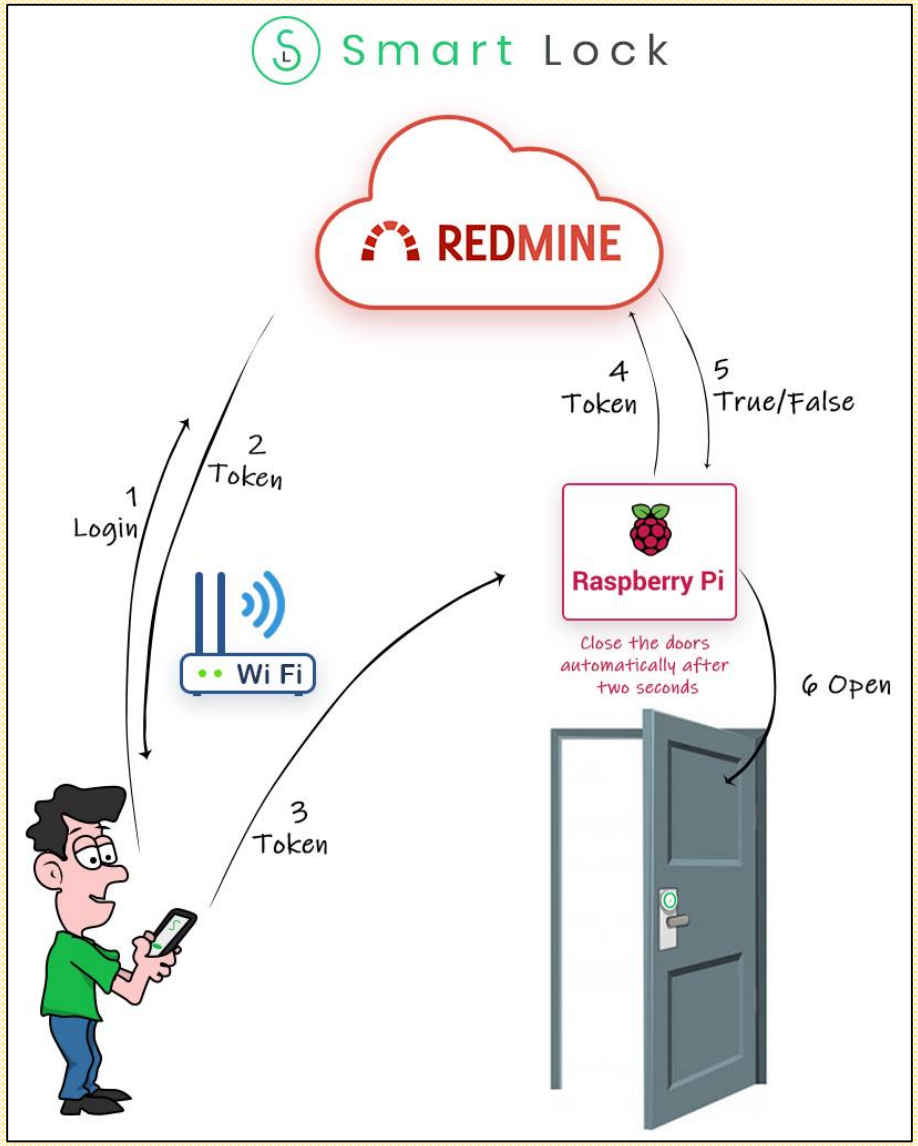
- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.
  - Once the connection is setup it remains open until the client sends a request to close the connection.
  - Client and server can send messages to each other after connection setup.
- Often used in **device-to-device (D2D)** communication
- Ensures **privacy, security, and directness**
- Typically implemented using **persistent socket connections**



LOGICAL DESIGN OF IOT

Example: Smart Lock and Smartphone App

- A **smart lock** at your home is paired exclusively with your **personal smartphone**.
- The communication between the lock and the phone is **encrypted** and **persistent**, meaning:
- Only that smartphone can lock/unlock the door.
- No other device can control the smart lock unless explicitly paired.
- This is often managed using **TLS-secured sockets** over protocols like MQTT/WebSockets or Bluetooth Low Energy (BLE).



LOGICAL DESIGN OF IOT

IoT communication APIs

Generally, we used Two APIs for IoT Communication. These IoT Communication APIs are:

- REST-based Communication APIs
- Web Socket-based Communication API

REST-based Communication APIs

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

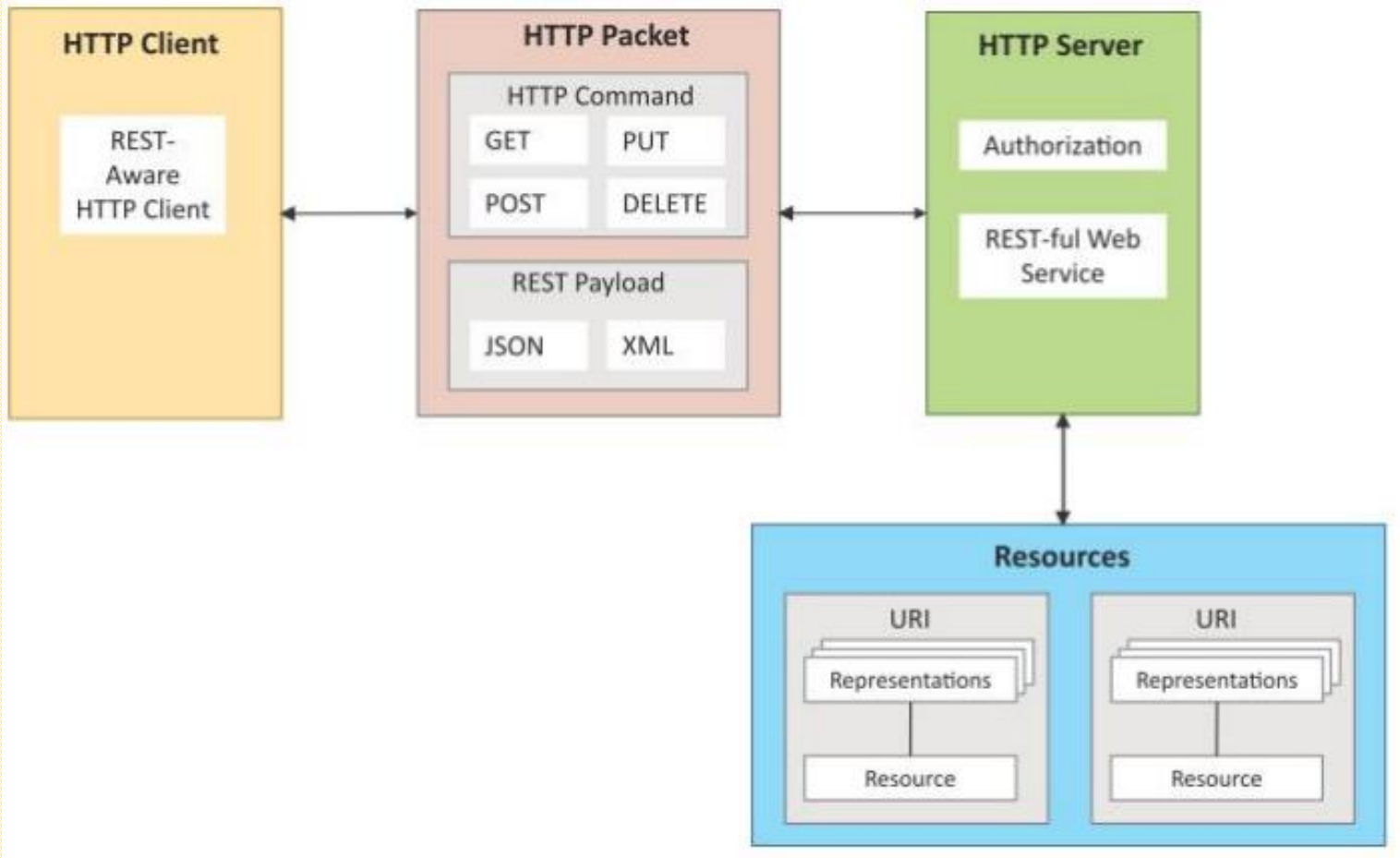
RESTful APIs are designed to be lightweight, stateless, and scalable, using standard web protocols and conventions to make interactions between distributed systems efficient and maintainable.

LOGICAL DESIGN OF IOT

The rest architectural constraint are as follows:

Client-server – A key feature is the client-server architecture, where the client and server are independent. This separation of concerns allows each component to evolve independently, provided the interface between them is maintained.

Stateless – Each client request to the server must contain all the information needed to understand and process the request. The server does not store any client context between requests, which makes the interactions simple and scalable.



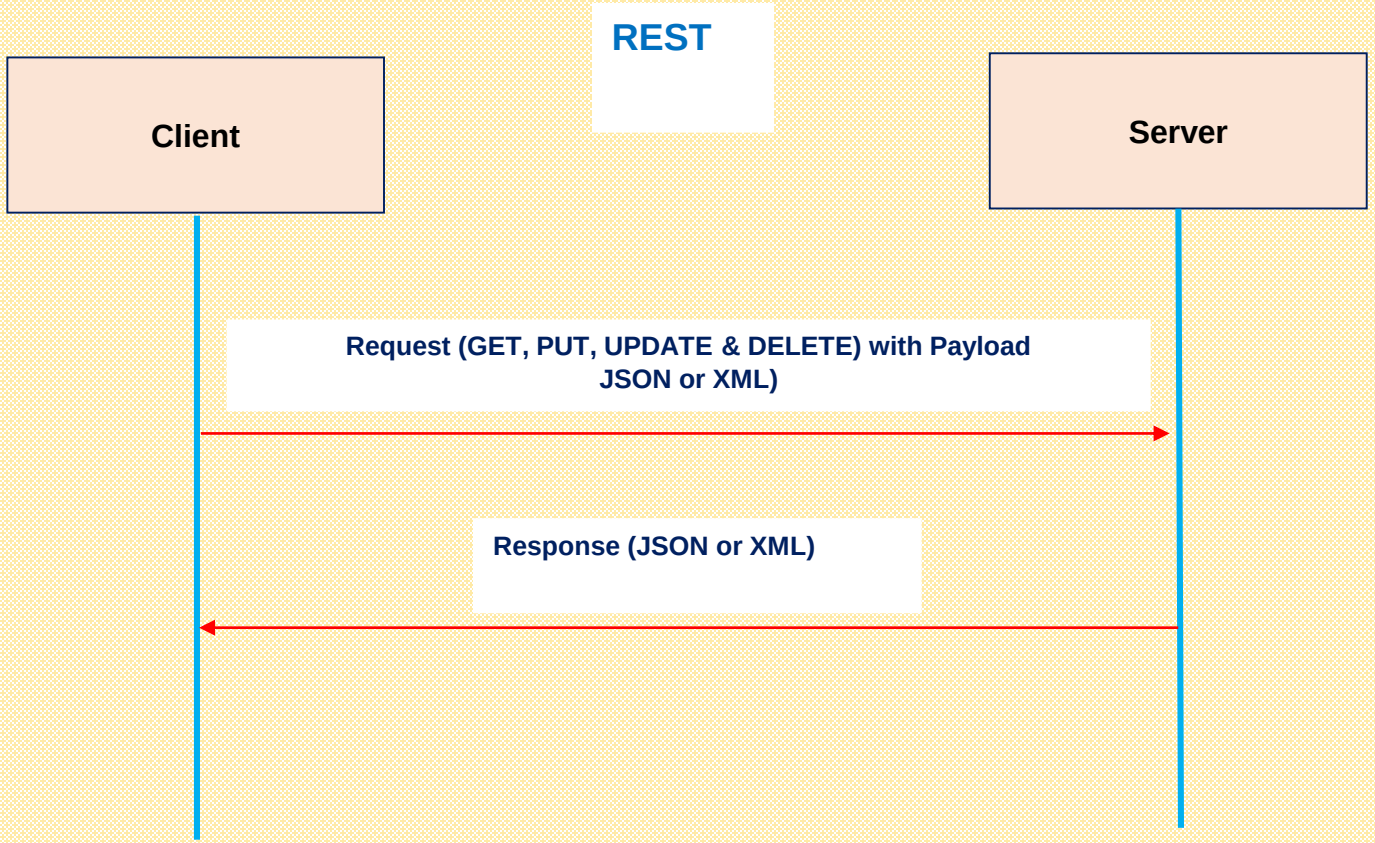
LOGICAL DESIGN OF IOT

- **Cache-able** – It refers to the ability to store responses to client requests in such a way that future, identical requests can be served faster without needing to contact the server again. This helps reduce latency, improve performance, and decrease the load on servers.
- **Layered system** – The use of layered system architecture allows REST APIs to be composed of hierarchical layers, enabling features like load balancing, caching, and proxy servers without the client knowing the underlying implementation.
- **Uniform interface** – It ensures that all interactions between client and server are standardized and follow a consistent structure. This includes the use of standard HTTP methods like GET, POST, PUT, DELETE, and PATCH, and resource identification through URIs.
- **Representation of Resources:** Resources are abstract concepts, and they are represented in a format such as JSON or XML when transferred over the network. For example, a user resource may be represented as a JSON object with fields like name, email, and ID.
- **Code on demand** – Servers can extend client functionality by sending executable code like JavaScript. Although not commonly used, this feature can add flexibility in some use cases.

LOGICAL DESIGN OF IOT

Request Response Model used by REST API:

A RESTful web service is a “WebAPI” implemented using HTTP and REST principles. REST is most popular IoT Communication APIs.



LOGICAL DESIGN OF IOT

HTTP Request Methods and Actions

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

LOGICAL DESIGN OF IOT

HTTP Request Feature, Methods and Actions

Feature	Description
Stateless	Each request from client to server must contain all the information needed.
Client-Server	Separation between client (user interface) and server (data processing).
Uniform Interface	Uses standard HTTP methods like GET, POST, PUT, DELETE.
Resource-Based	Resources are identified using URIs (e.g., /temperature, /device/1).
Cacheable	Responses can be cached to improve performance.

Method	Description	Example URI	Action Performed
GET	Retrieve resource	/sensor/temperature	Get temperature data
POST	Create new resource	/device	Add a new IoT device
PUT	Update existing resource	/device/123	Update device settings
DELETE	Delete a resource	/device/123	Remove a device

LOGICAL DESIGN OF IOT

REST API in Smart Agriculture System

Feature	Request	Response

GET Request to Read Sensor Data	GET /sensor/moisture HTTP/1.1
	Host: api.smartfarm.com
		{
		"sensor_id": "moisture01",
		"value": "23%",
		"timestamp": "2025-08-05T10:45:00Z"
		}

POST Request to Add a New Device	POST /device HTTP/1.1
	Content-Type: application/json
	{
	"device_id": "pump1",
	"type": "water_pump",
	"location": "Field A"
	}
		{
		"message": "Device added successfully", "device_id": "pump1"
		}

PUT Request to Update Device Configuration	PUT /device/pump1 HTTP/1.1
	Content-Type: application/json
	{
	"status": "ON", "mode": "auto"
	}
		{
		"message": "Pump configuration updated"
		}

DELETE Request to Remove a Device	DELETE /device/pump1 HTTP/1.1	{
		"message": "Device pump1 removed"
		}

LOGICAL DESIGN OF IOT

WebSocket based communication API

- WebSocket is a full-duplex, bidirectional communication protocol that operates over a single TCP connection.
- This API uses full-duplex communication so it does not require a new connection setup every time when it requests new data.
- Unlike HTTP (which is request-response-based and stateless), WebSocket allows real-time, continuous communication between a client (like a browser or IoT device) and a server.
- WebSocket communication begins with a connection setup request sent by the client to the server. The request (called WebSocket handshake) is sent over HTTP and the server interprets it is an upgrade request. If the server supports WebSocket protocol, the server responds to the WebSocket handshake response.
- This type of API reduces the traffic and latency of data and makes sure that each time when we request new data it cannot terminate the request.

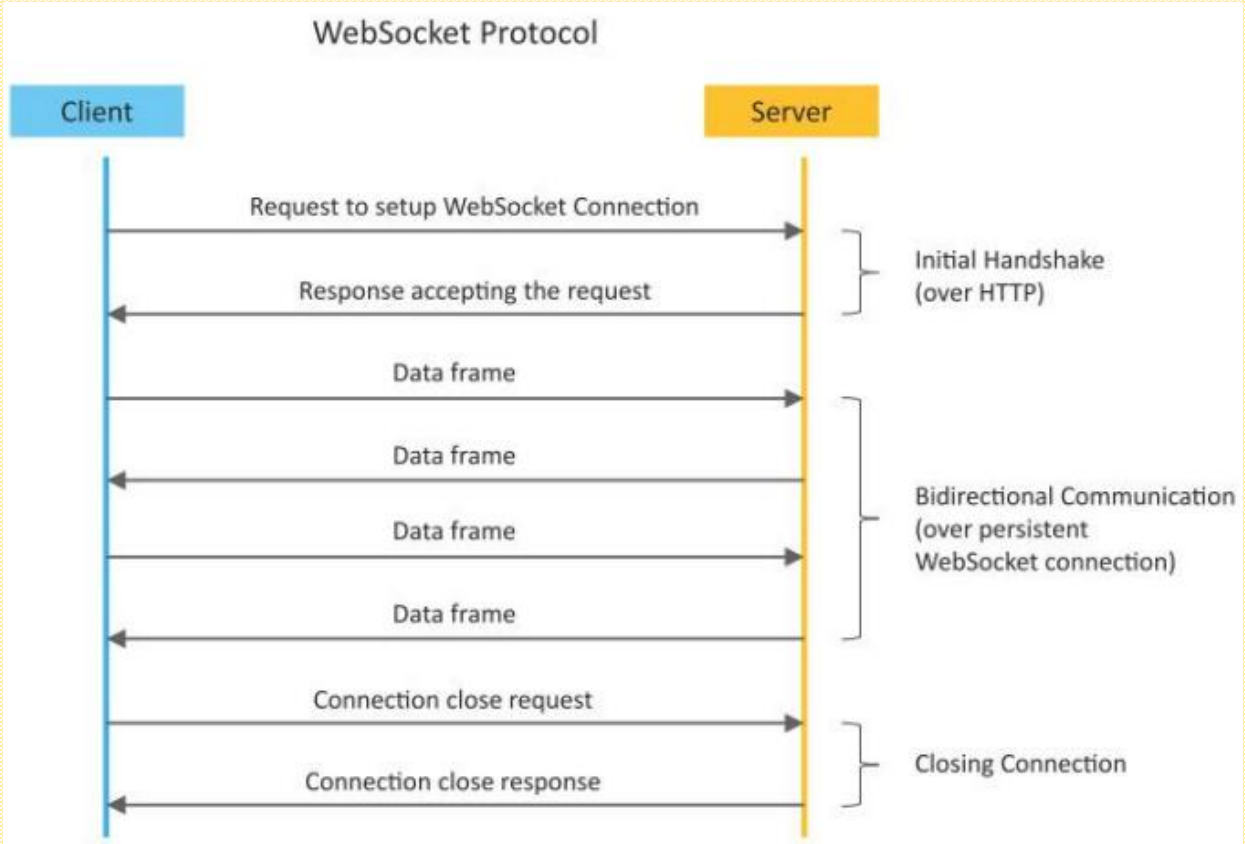
LOGICAL DESIGN OF IOT

Key Features of WebSocket based communication API

- **Full Duplex Connection:** Both client and server can send messages independently at any time.
- **Low Latency (Realtime):** Ideal for real-time communication (chat apps, IoT monitoring, gaming).
- **Persistent Connection:** Maintains a single connection until one side closes it.
- **Efficient:** Less overhead compared to opening new HTTP connections for each message.

How it works:

- **Client initiates a handshake** using HTTP with an Upgrade header.
- **Server accepts** and switches the protocol from HTTP to WebSocket.
- **Bi-directional communication** begins.
- **Either side can close** the connection at any time.



IOT ENABLING TECHNOLOGIES

There are several technologies which play important role in IoT Applications. The technologies are:

- Wireless Sensor Networks
- Cloud Computing
- Big Data Analytics
- Embedded Systems
- Communication Protocols
- Security Protocols
- Web Services
- Mobile Internet
- Semantic Search Engines

IOT ENABLING TECHNOLOGIES

Wireless Sensor Networks

- **A Wireless Sensor Network is a network** of distributed devices with sensors that monitor physical or environmental conditions like temperature, humidity, motion, vibration, etc.
- A typical WSN consists of:
  - **End Nodes**: Devices with sensors attached.
  - **Routers**: Forward data packets from end nodes to the coordinator.
  - **Coordinator**: Collects data from all nodes and acts as a gateway to the Internet.

How WSNs Work:

- WSNs rely on wireless protocols like IEEE 802.15.4.
- ZigBee is a popular wireless technology based on this standard.
  - Operates at **2.4 GHz**.
  - Supports data rates up to **250 KB/s**.
  - Typical range: **10 to 100 meters** (depends on power output & environment).

IOT ENABLING TECHNOLOGIES

Applications of Wireless Sensor Networks in IoT

- **Weather Monitoring Systems**
  - Collect temperature, humidity, and other environmental data.
  - Data is aggregated and analyzed.
- **Indoor Air Quality Monitoring**
  - Measure air quality and gas concentrations indoors.
- **Soil Moisture Monitoring**
  - Measure soil moisture at different locations (useful in agriculture).
- **Surveillance Systems**
  - Detect motion and collect security data.
- **Smart Grids**
  - Monitor electrical grid status at multiple points.
- **Structural Health Monitoring**
  - Monitor vibration in buildings or bridges to check structural integrity.

IOT ENABLING TECHNOLOGIES

Key Advantages of WSNs:

- **Large scale deployment** of low-cost, low-power nodes.
- **Continuous monitoring** of environmental & physical conditions.
- **Self-organizing:** No manual reconfiguration needed when nodes fail or new nodes are added.
- **Robust:** Can reconfigure itself automatically for reliable operation.

IOT ENABLING TECHNOLOGIES

Cloud Computing

- Cloud computing is a transformative technology that delivers **applications, computing power, storage, and networking services** over the Internet. It works on a **“pay-as-you-go”** model — users pay only for the resources they use.
- The “cloud” is a term that simply means “the internet.” Computing involves the infrastructures and systems that allow a computer to run and build, deploy, or interact with information.
- In cloud computing, this means that instead of hosting infrastructure, systems, or applications on your hard drive or an on-site server, you’re hosting it on virtual/online servers that connect to your computer through secure networks.

Key Features:

- **On-demand resources:** Users can provision computing, storage, and networking resources as needed.
- **Self-service:** Users don’t need direct interaction with a human provider to provision resources.
- **Multi-tenancy:** Resources are pooled to serve multiple users simultaneously.
- **Platform independence:** Accessible from different devices (PCs, laptops, tablets, smartphones).
- **Virtualization:** Users access virtual machines and storage, not the underlying physical hardware.

IOT ENABLING TECHNOLOGIES

Cloud Computing Service Models:

- **SaaS or Software as a Service.**
  - SaaS means instead of installing software on your computer, you access the platform online.
  - Provides **ready-to-use software applications** over the Internet.
  - User just uses the app — no need to manage servers, OS, or software updates.
  - Platform independent — accessible via browser on any device.
  - Example: A web-based IoT dashboard that shows real-time data and analytics. Deploying a custom server for storing IoT sensor data.
    - Square, which processes payments online
    - Google Apps such as Google Drive or Calendar
    - Slack, which allows collaboration and chat between other users

IOT ENABLING TECHNOLOGIES

Cloud Computing Service Models:

- **IaaS or Infrastructure as a Service.**
  - IaaS provides infrastructure components such as servers, storage, networking, security, and moreover the cloud.
  - Provides virtual machines, storage, and networking.
  - Users manage: virtual machines, OS, applications.
  - **Billing:** Pay for compute/storage used.
  - Example: Deploying a custom server for storing IoT sensor data. Develop an IoT analytics application using cloud-hosted tools.
    - Dropbox, a file storage and sharing system
    - Microsoft Azure, which offers backup and disaster recovery services, hosting, and more
    - Rackspace, which offers data, security, and infrastructure services.

IOT ENABLING TECHNOLOGIES

Cloud Computing Service Models:

❖ **PaaS or Platform as a Service.**

- PaaS provides computing platforms such as operating systems, programming language execution environments, databases, and web servers.
- Provides a platform to develop, run, and manage applications.
- Includes development tools, APIs, and services.
- Users only manage their applications — not the underlying infrastructure.
- Example: Develop an IoT analytics application using cloud-hosted tools:
  - Google App Engine and Heroku, which allow developers to develop and serve apps

- **Serverless Computing.** Serverless computing (also called simply “Serverless”) is simply using a server on the cloud. This offers more elasticity, easier maintenance, and is often more price effective than hosting servers on-site.

IOT ENABLING TECHNOLOGIES

Big Data Analytics

- Big Data is a massive amount of data sets that cannot be stored, processed, or analyzed using traditional tools.
- **So, what makes data "big"?**
  - Big data is characterized by the five V's: volume, velocity, variety, variability, and value. It's complex, so making sense of all of the data in the business requires both innovative technologies and analytical skills.
- Big data analytics describes the process of uncovering trends, patterns, and correlations in large amounts of raw data to help make data-informed decisions. These processes use familiar statistical analysis techniques—like clustering and regression—and apply them to more extensive datasets with the help of newer tools.

Why Big Data Analytics Matters for IoT

- Big Data Analytics allows IoT systems to:
  - Detect patterns (e.g., predicting machine failures)
  - Make smart decisions (e.g., adjusting temperature automatically)
  - Optimize processes (e.g., reduce energy usage)
  - Provide insights for innovation and improved services.

IOT ENABLING TECHNOLOGIES

Big Data Analytics in IoT

- Big data analytics refers to collecting, processing, cleaning, and analyzing large datasets to help organizations operationalize their big data.
- IoT systems generate massive amounts of data every second from sensors, devices, and machines. Managing this data requires Big Data Analytics, which uses advanced techniques and tools to:
  - **Clean** and organize raw data (data cleansing and wrangling)
  - **Store** data efficiently
  - **Process & analyze** it for patterns, trends, or insights
  - **Visualize** results to support decision-making

IOT ENABLING TECHNOLOGIES

Big Data Analytics

- Analyze Data: Getting big data into a usable state takes time. Once it's ready, advanced analytics processes can turn big data into big insights. Some of these big data analysis methods include:
  - **Data mining** sorts through large datasets to identify patterns and relationships by identifying anomalies and creating data clusters.
  - **Predictive analytics** uses an organization's historical data to make predictions about the future, identifying upcoming risks and opportunities.
  - **Deep learning** imitates human learning patterns by using artificial intelligence and machine learning to layer algorithms and find patterns in the most complex and abstract data.

IOT ENABLING TECHNOLOGIES

Tool/Technology	Purpose/Description
Hadoop	Open-source framework to store and process large datasets on clusters of commodity hardware. Handles structured and unstructured data.
MapReduce	Core component of Hadoop. Performs distributed processing in two steps: Map (distribute/filter data) and Reduce (aggregate results).
YARN (Yet Another Resource Negotiator)	Resource management and job scheduling technology for Hadoop clusters.
Spark	Open-source cluster computing framework. Supports fast batch and stream processing with in-memory computing.
NoSQL Databases	Non-relational databases for unstructured/variable data. Examples: MongoDB (frequent data changes), Cassandra (distributed storage for big chunks of data).
Tableau	End-to-end analytics and visualization platform for prepping, analyzing, and sharing big data insights.
Talend	Tool for data integration and management (ETL – Extract, Transform, Load).
STORM	Open-source real-time computational system for processing data streams instantly.
Kafka	Distributed streaming platform for fault-tolerant, high-throughput data pipelines and real-time messaging.

IOT ENABLING TECHNOLOGIES

Artificial Intelligence & Machine Learning in IoT

- Artificial Intelligence (AI) in IoT makes devices “smart” — they can learn, reason, and make decisions without constant human instructions.
- Machine Learning (ML) is a branch of AI that trains IoT systems to learn from data, recognize patterns, and improve automatically with experience.

How AI & ML Work in IoT

- **Collect Data:**

  IoT devices generate huge amounts of real-time data through sensors.
- **Analyze & Learn:**

  ML algorithms analyze this data to find patterns, detect anomalies, or make predictions.
- **Act Smartly:**

  Based on analysis, IoT devices can automatically adjust, send alerts, or take actions without human intervention.

IOT ENABLING TECHNOLOGIES

Key Benefits:

- Enables automation and intelligent decision-making.
- Reduces human effort and errors.
- Improves efficiency and safety.
- Provides valuable insights from IoT data.

Examples:

- **Smart Home:**

  AI enables voice assistants (like Alexa) to learn user preferences and automate home devices.

- **Predictive Maintenance:**

  ML models predict equipment failures in factories, so maintenance can be done **before** breakdowns occur.

- **Healthcare:**

  Wearable IoT devices use AI to track health data and alert users or doctors about unusual patterns.

- **Smart Vehicles:**

  Self-driving cars use AI and ML to detect objects, read signs, and make driving decisions in real-time.

IOT ENABLING TECHNOLOGIES

Communication Protocols in IoT

- Communication protocols form the backbone of IoT systems.
- They enable connectivity between IoT devices and allow interaction with applications.
- Protocols ensure seamless data exchange over the network.

Purpose of Communication Protocols:

- Allow devices to exchange data reliably and efficiently.
- Enable coupling between sensors, actuators, and cloud/server applications.

Types of Protocols:

- **Link Layer Protocols** – e.g., Ethernet, Wi-Fi, Bluetooth, Zigbee
- **Network Layer Protocols** – e.g., IPv4, IPv6
- **Transport Layer Protocols** – e.g., TCP, UDP
- **Application Layer Protocols** – e.g., HTTP, MQTT, CoAP

IOT ENABLING TECHNOLOGIES

Key Functions:

- **Data Exchange Formats**: Define how data is structured.
- **Data Encoding**: Representing data in machine-readable formats.
- **Addressing Scheme**: Ensures data reaches the correct destination device.
- **Routing**: Guides data from source to destination through the network.

Additional Functionalities:

- **Sequence Control**:
  - Maintains the correct order of packets.
  - Helps in detecting missing or duplicate packets.
- **Flow Control**:
  - Regulates the rate of data transmission.
  - Ensures sender doesn’t overwhelm receiver or the network.
- **Retransmission of Lost Packets**:
  - Enhances reliability by re-sending lost or corrupted data packets.

IOT ENABLING TECHNOLOGIES

What is an Embedded System in IoT?

An Embedded System is a microcontroller or microprocessor-based hardware system with software programmed for a specific function. In the context of IoT (Internet of Things), embedded systems form the core of “smart” devices — they sense, process, communicate, and often actuate in response to environmental data.

Key Components of an IoT Embedded System

Component	Function	Examples
Microcontroller / Microprocessor	Controls the system, executes code	Arduino, ESP32, Raspberry Pi Pico
Sensors	Collect data from the environment	DHT11 (Temp/Humidity), MQ-135 (Gas)
Actuators	Perform physical actions	Motors, Relays, LED, Buzzer
Communication Module	Enables data transfer between device and cloud/network	Wi-Fi (ESP8266), Bluetooth, LoRa
Power Supply	Powers the device	Batteries, Solar cells, Power banks
Memory	Stores code (flash) and data (RAM)	EEPROM, SRAM
Software/Firmware	Embedded code for control logic and communication	C/C++, MicroPython, Arduino IDE

IOT ENABLING TECHNOLOGIES

How Embedded Systems Work in IoT

- **Sense**: Data is collected through sensors (e.g., temperature, light).
- **Process**: Microcontroller processes the data (e.g., apply threshold or logic).
- **Communicate**: Sends data to a server/cloud via communication protocols.
- **Actuate**: Performs an action if required (e.g., turn on fan if too hot).
- **Monitor/Control**: Remote dashboard or app monitors or controls the device.

Example: Smart Street Light System

Component	Details
Microcontroller	ESP32 (Wi-Fi enabled)
Sensor	LDR (Light sensor)
Actuator	Relay module connected to street light
Communication	Wi-Fi + HTTP API to send data to cloud
Power	Solar panel with battery
Functionality	Turns light ON at dusk, OFF at dawn

IOT ENABLING TECHNOLOGIES

Applications of Embedded Systems in IoT

- Smart Home Devices (thermostats, security systems)
- Industrial Automation (monitoring and control)
- Agriculture (soil moisture, irrigation control)
- Health Monitoring (wearable ECG, pulse oximeter)
- Smart Cities (waste bins, traffic lights, pollution control)
- Environmental Monitoring (weather stations, air quality)

Popular Embedded Boards for IoT

Board	Features
Arduino Uno	Easy to use, digital/analog I/O, no Wi-Fi by default
ESP8266/ESP32	Built-in Wi-Fi/Bluetooth, cheap, powerful
Raspberry Pi	Linux-based, powerful for edge computing
STM32	Industrial-grade, real-time control
BeagleBone Black	High-performance embedded Linux

IOT LEVELS: IoT Level 1

What is a Level-1 IoT System?

- A Level-1 IoT System is the simplest type of IoT system. It consists of:
- A **single node/device** that handles:
  - Sensing/Actuation
  - Data storage
  - Local processing/analysis
  - Application hosting
- These systems are:
  - Low-cost, low-complexity
  - Suitable for applications where:
    - Data is not very large
    - Processing requirements are not computationally intensive

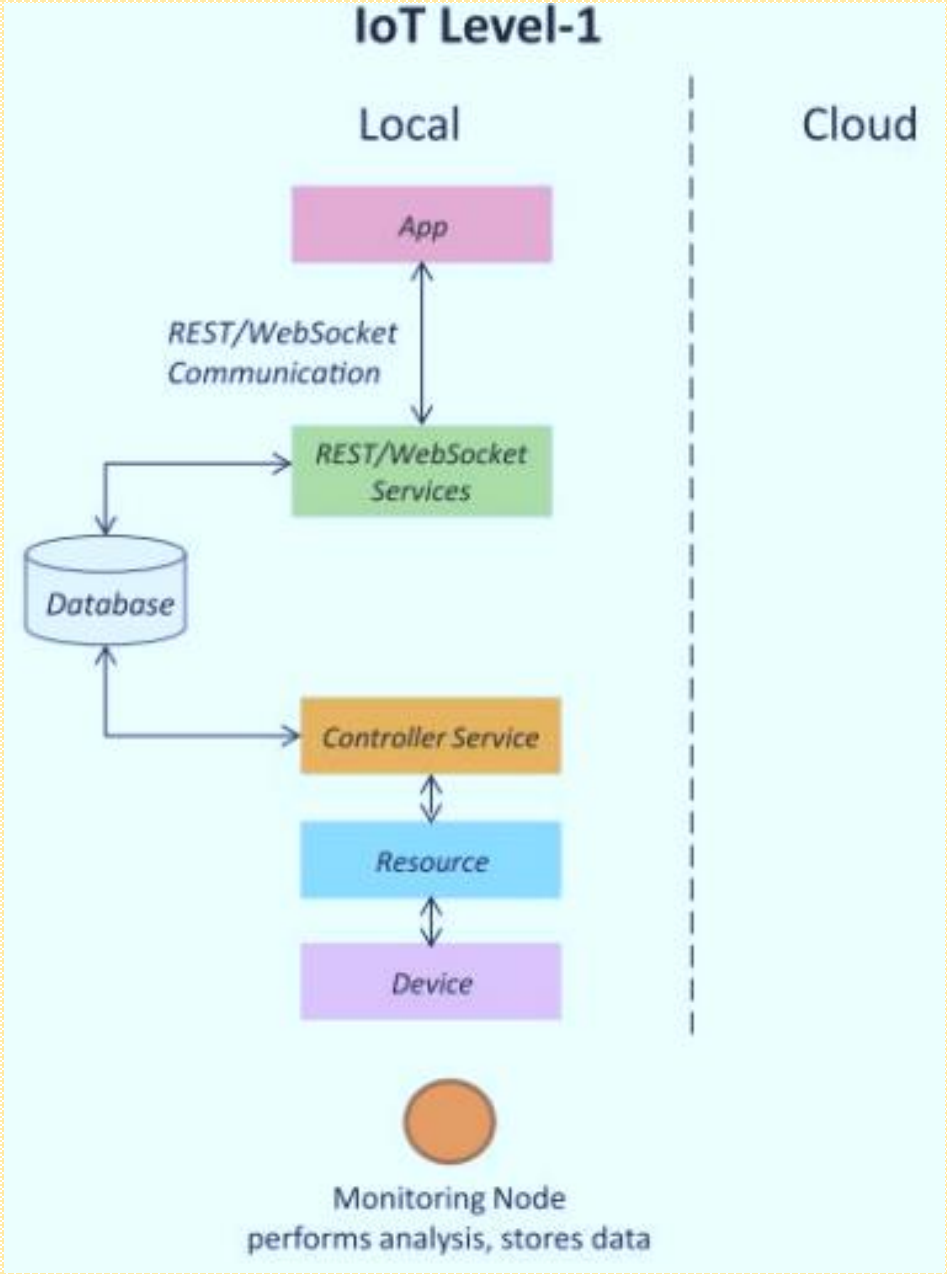
Key Features of Level-1 IoT System:

- **Single Node Function:** All functions (sensing, control, DB, UI) on one device.
- **Local Processing:** No need for cloud or external analytics
- **REST Services:** Allow remote control and automation logic
- **Lightweight & Efficient:** Good for small home setups and educational projects

IOT LEVELS: IoT Level 1

Example Home Automation: Components

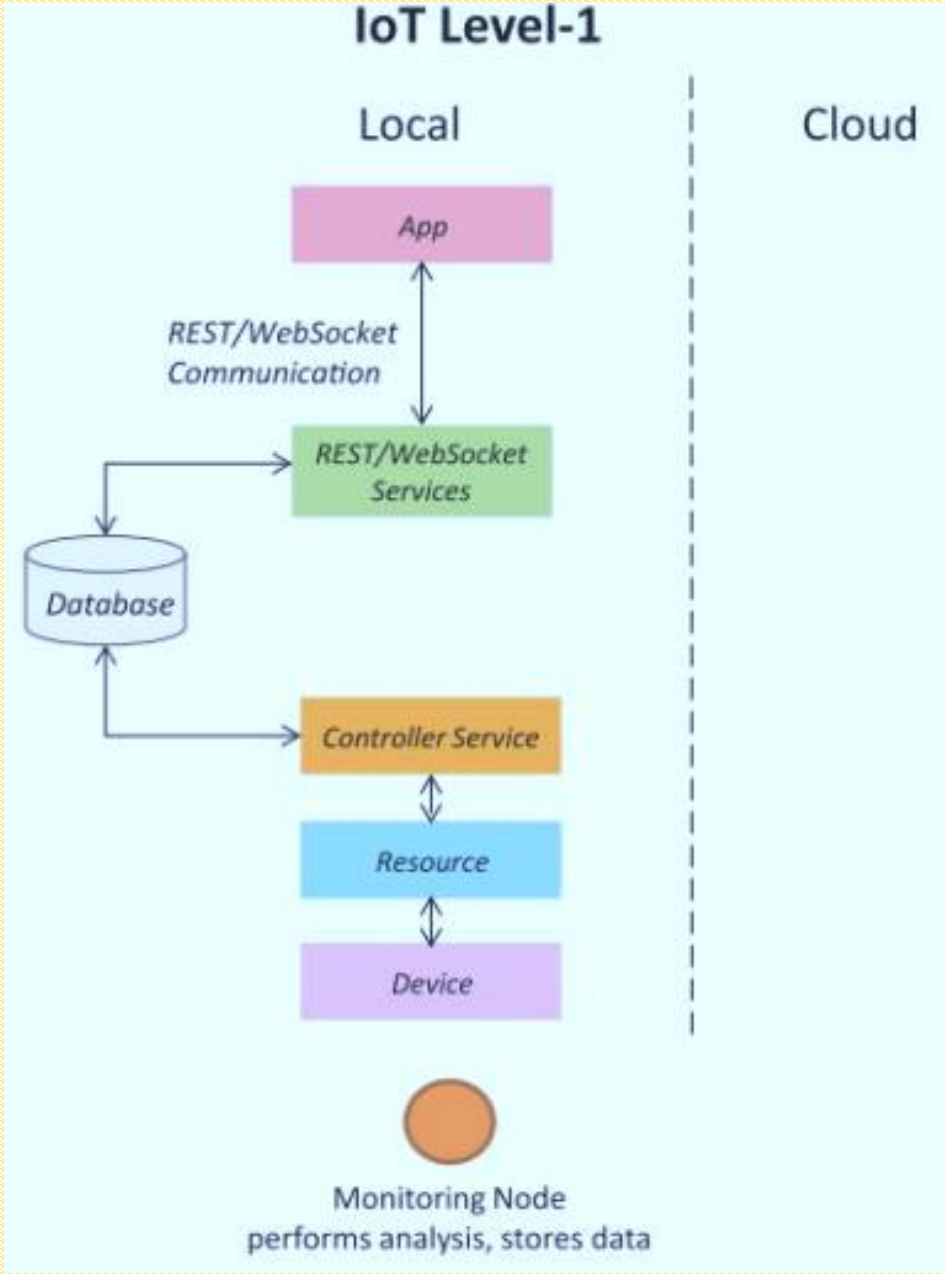
Component	Role
Single Node (e.g., Raspberry Pi)	Performs sensing, control, database storage, and hosting the UI
Sensors (e.g., LDR)	Detect environmental condition (e.g., light level)
Actuators (e.g., relay switches)	Switch lights/appliances ON or OFF
Database (e.g., MySQL)	Stores current state of devices
REST API (e.g., Django REST)	Allows services to retrieve/update device states
Web Application (UI)	User interacts with this to control appliances



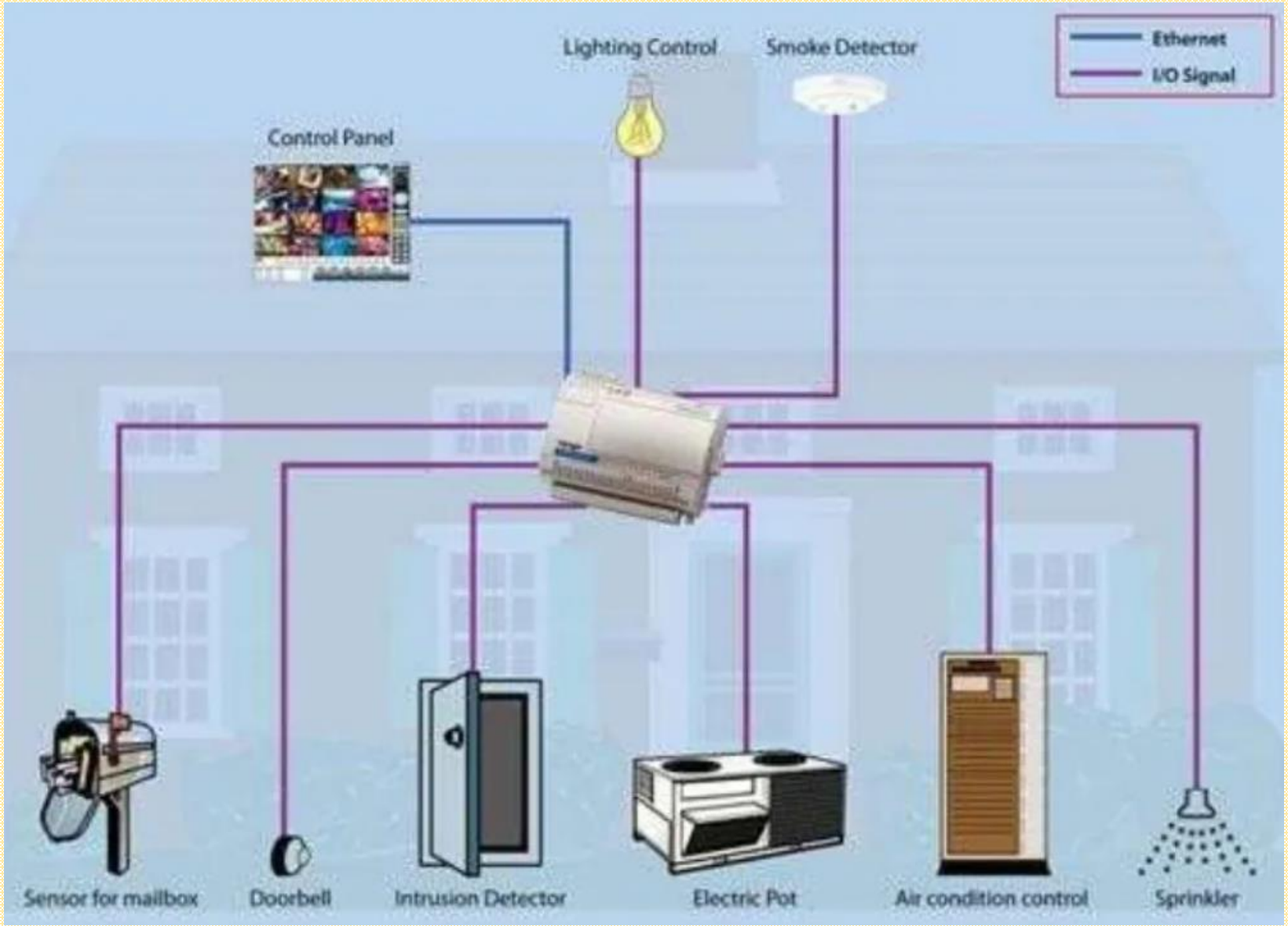
IOT LEVELS: IoT Level 1

How It Works

1. Sensors collect data (e.g., light level in a room).
2. Data is stored in a local database on the device.
3. Controller Service continuously checks the light level:
   - If in auto mode, it turns lights ON/OFF based on light level.
4. Web Application allows:
   - User to see current light state
   - User to control light manually (if in manual mode)
5. RESTful APIs provide services to update/fetch status.
6. Since the device is connected to the Internet, users can:
   - Access the application remotely (e.g., via browser or mobile).



IOT LEVELS: IoT Level 1



IOT LEVELS: IoT Level 2

What is a Level-2 IoT System?

- A Level-2 IoT System is characterized by:
  - A single node/device that handles:
    - Sensing/Actuation
    - Local analysis
  - Cloud storage for data
  - Cloud-hosted application (UI)
- These systems are suitable when:
  - Data volume is large
  - Analysis is light, and can still be done locally

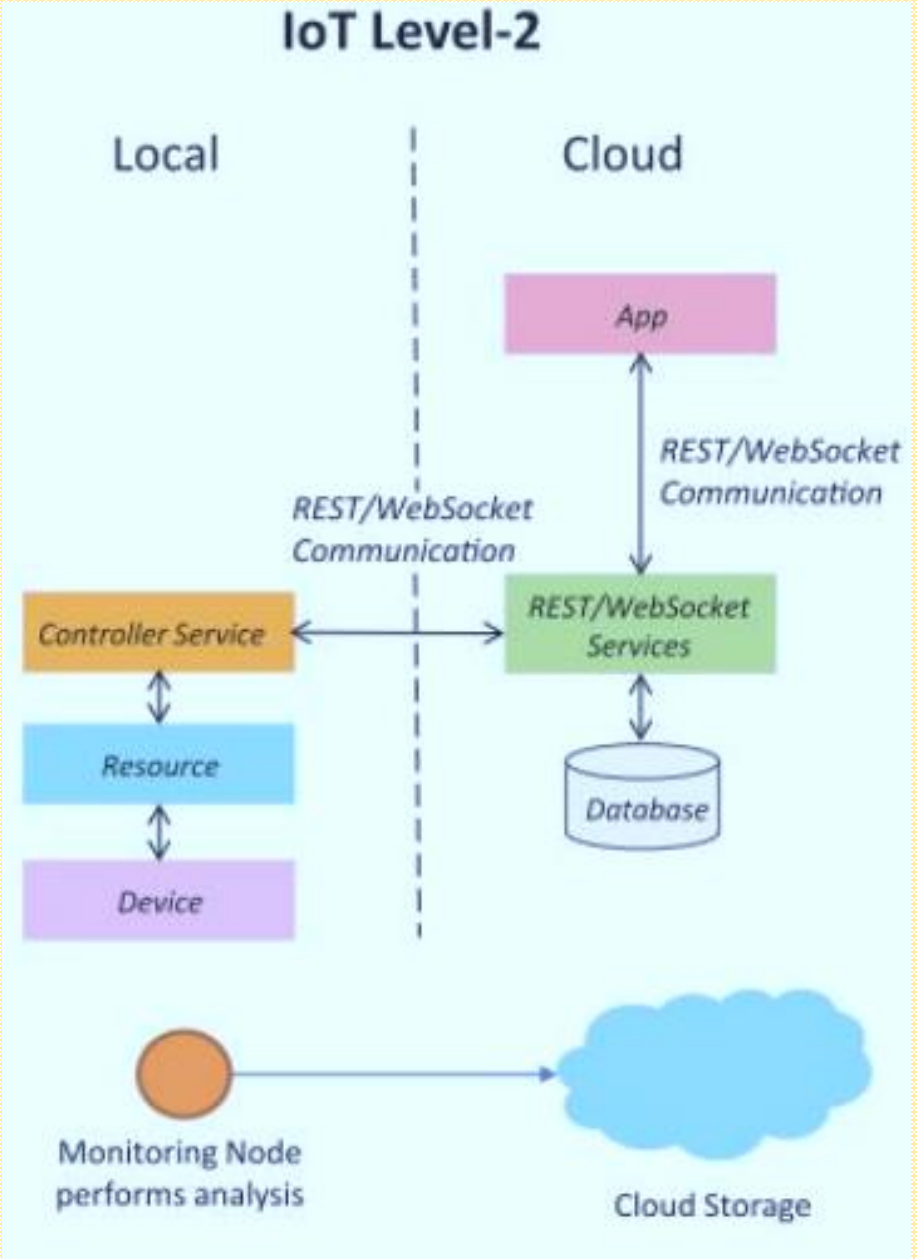
Key Features of Level-2 IoT System:

- Single-node sensing/actuation: Performs local control and sends data to cloud.
- Cloud data storage: Suited for large, historical datasets.
- Cloud-based application: Offers visualization and user-friendly dashboards.
- REST APIs: Enable remote communication with the cloud system.
- Semi-autonomous: Basic intelligence at the edge, enhanced insights from cloud.

IOT LEVELS: IoT Level 2

Example Smart Irrigation System

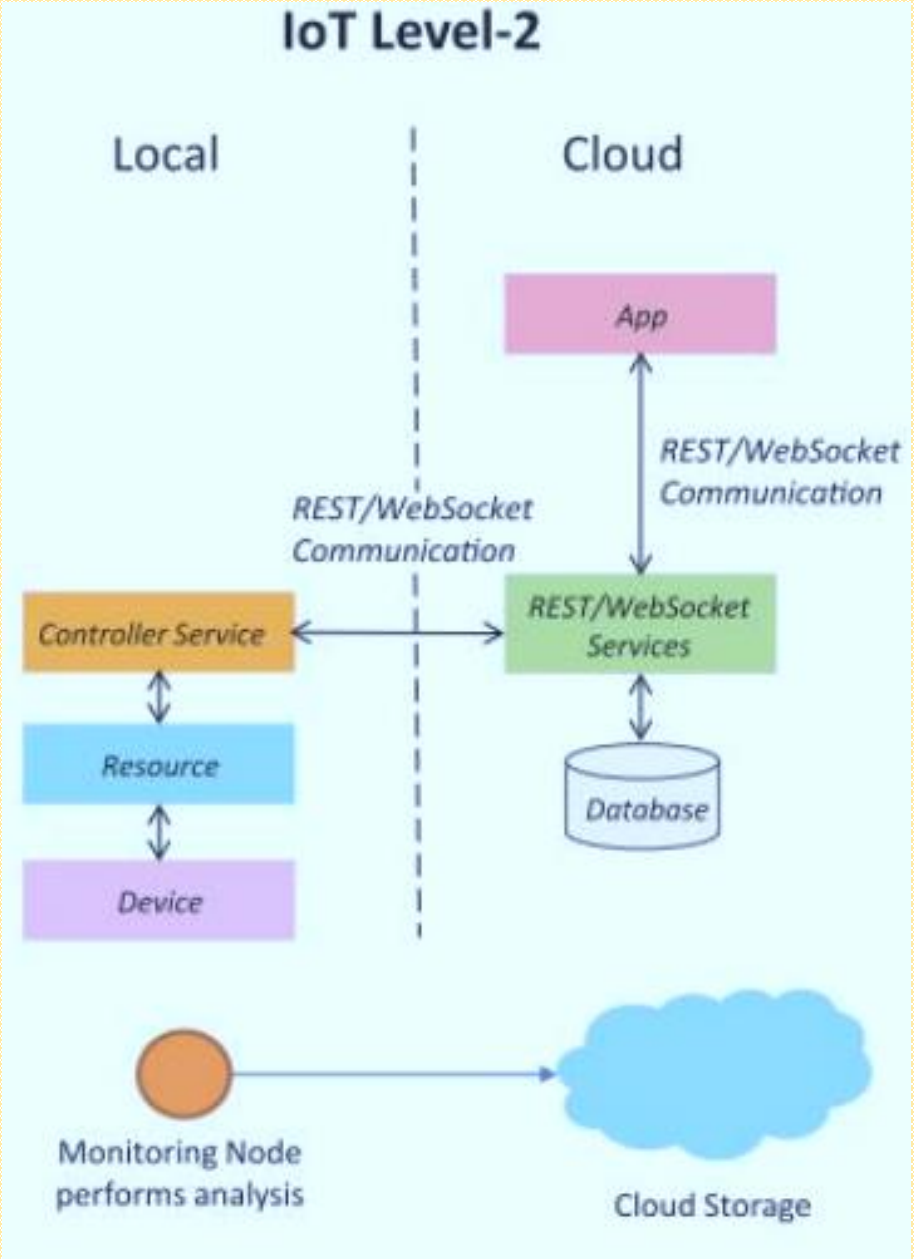
Component	Role
Single Node (e.g., Arduino or Raspberry Pi)	Collects soil moisture data and controls irrigation
Sensors (e.g. Moisture Sensors)	Measure the moisture content in soil
Actuator (e.g., solenoid valve)	Turns irrigation system ON/OFF based on moisture level
Cloud Database	Stores historical sensor data
Cloud REST API	Used for uploading/downloading moisture readings
Cloud-based Application	Provides visualization and analytics (moisture trends, decisions)



IOT LEVELS: IoT Level 2

How It Works

1. Sensors read soil moisture levels.
2. The controller service monitors the readings:
   - If moisture < threshold → sends command to actuator to irrigate.
3. Sensor data is also sent to the cloud.
4. Cloud REST API stores data in cloud database.
5. A web/mobile application (hosted in the cloud) provides:
   - Helps in making data Visualization of moisture trends
   - Dashboard to track irrigation history
6. Helps in making data-driven irrigation decisions.



IOT LEVELS: IoT Level-3

What is a Level-3 IoT System?

- A Level-3 IoT system is designed with a single node (device) that performs sensing and/or actuation, but data storage, analysis, and application hosting are entirely cloud-based.
- This level is suited for applications where:
  - Large volumes of data are generated
  - Computational analysis is intensive
  - Centralized processing is preferred

Why Use Level-3 IoT?

- When real-time monitoring and complex analysis are needed.
- For applications where centralized cloud access is critical.
- For scalable and globally accessible systems.

Key Features of Level-3 IoT System:

- **Single-node sensing/actuation:** Performs local control and sends data to cloud.
- **Cloud data storage:** Suited for large, historical datasets.
- **Cloud data analysis**: enables scalable, real-time processing, storage, and intelligent insights from sensor data using cloud platforms and services.
- **Cloud-based application:** Offers visualization and user-friendly dashboards.
- **REST APIs**: Enable remote communication with the cloud system.

IOT LEVELS: IoT Level-3

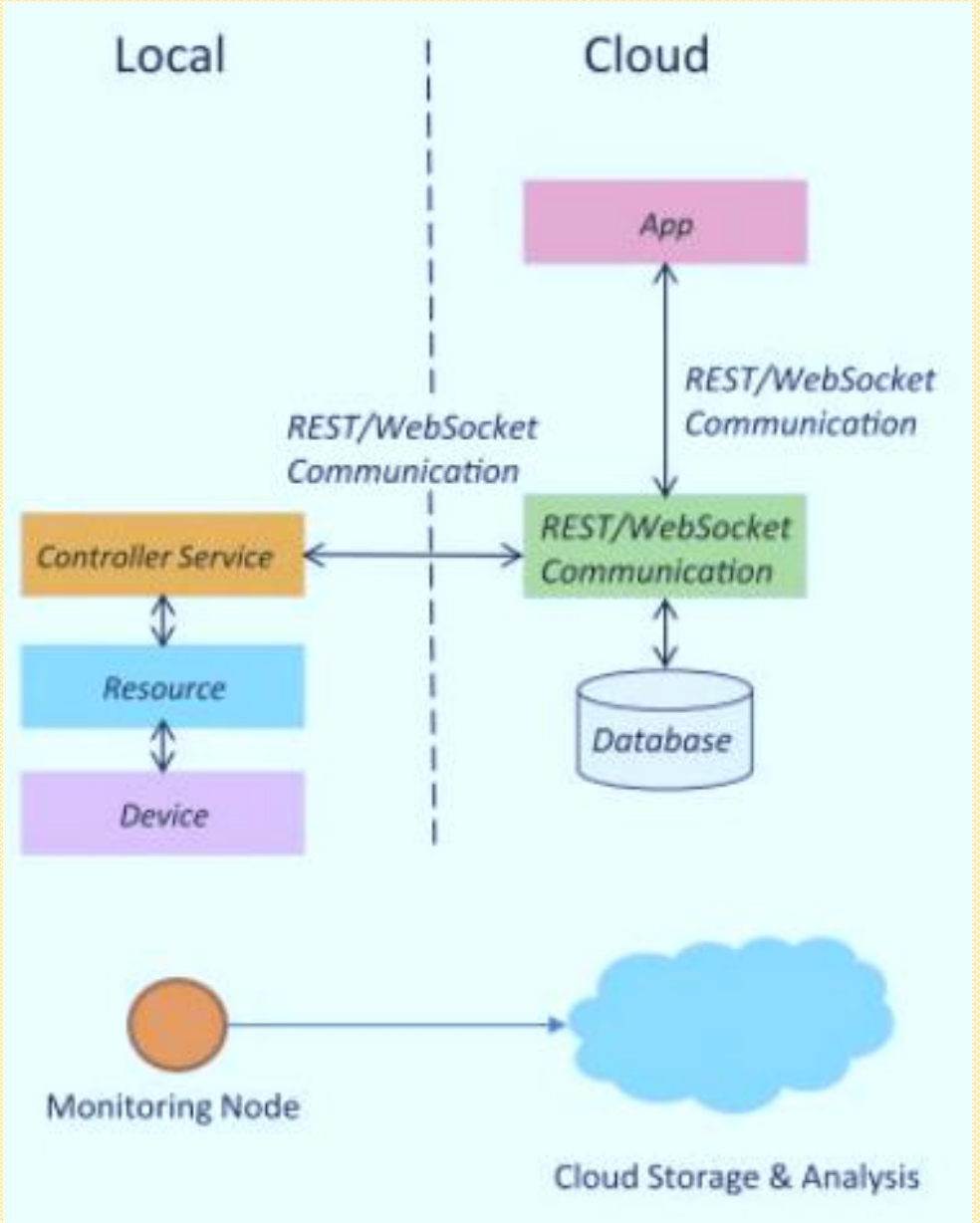
Example Package Handling System

Category	Component	Function/Role
IoT Device	Microcontroller/SBC (e.g., ESP32, Raspberry Pi)	▪ Reads sensor data and handles network communication
Sensors	Accelerometer (e.g., MPU6050)	▪ Measures vibration and impact forces during transit
	Gyroscope (e.g., MPU6050)	▪ Detects angular motion and orientation changes
Communication	WebSocket Client	▪ Sends data from device to cloud in real-time
	Wi-Fi Module (built-in or external)	▪ Enables Internet connectivity for the device
Cloud Platform	WebSocket Server (e.g., Node.js, AWS IoT Core)	▪ Receives real-time data from the device
	Cloud Database (e.g., Firebase, AWS DynamoDB)	▪ Stores vibration data and timestamps
Application	Web Dashboard (ReactJS / Angular)	▪ Displays vibration events, graphs, and package history
	Alert System (via Email/SMS/API)	▪ Notifies stakeholders if vibration exceeds safe threshold
Analytics	Rule-based or ML Engine	▪ Analyzes patterns in vibration data to detect potential damage or mishandling
Power Supply	Battery Pack / USB Power	▪ Powers the IoT device during shipment

IOT LEVELS: IoT Level-3

How It Works: Package Handling System

- **Sensors Used**: Accelerometer & Gyroscope on the package
- **Monitoring**: Real-time tracking of vibration levels during shipment
- **Data Flow**:
  - Sensor data → Sent in real-time to the cloud via WebSocket
  - Stored in cloud storage
  - Visualized in a cloud-based application
- **Cloud Services**:
  - WebSocket: Real-time communication (faster than REST)
  - Cloud Database: Store vibration levels
  - Cloud App: Visualize and analyze data, trigger alerts

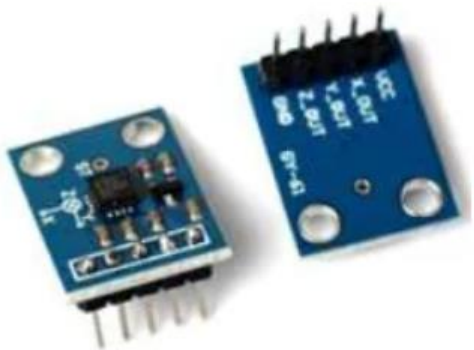


IOT LEVELS: IoT Level-3

Sensors used

Accelrometer

sense movement or vibrations



Gyroscope

Gives orientation info

IOT LEVELS: IoT Level-4

What is a Level-4 IoT System?

- The Level-4 IoT system is designed for complex, large-scale, and computation-intensive applications that require multiple data-collecting nodes and both local and cloud-based processing. This level supports distributed data collection and centralized cloud analytics, enabling robust and scalable IoT solutions.

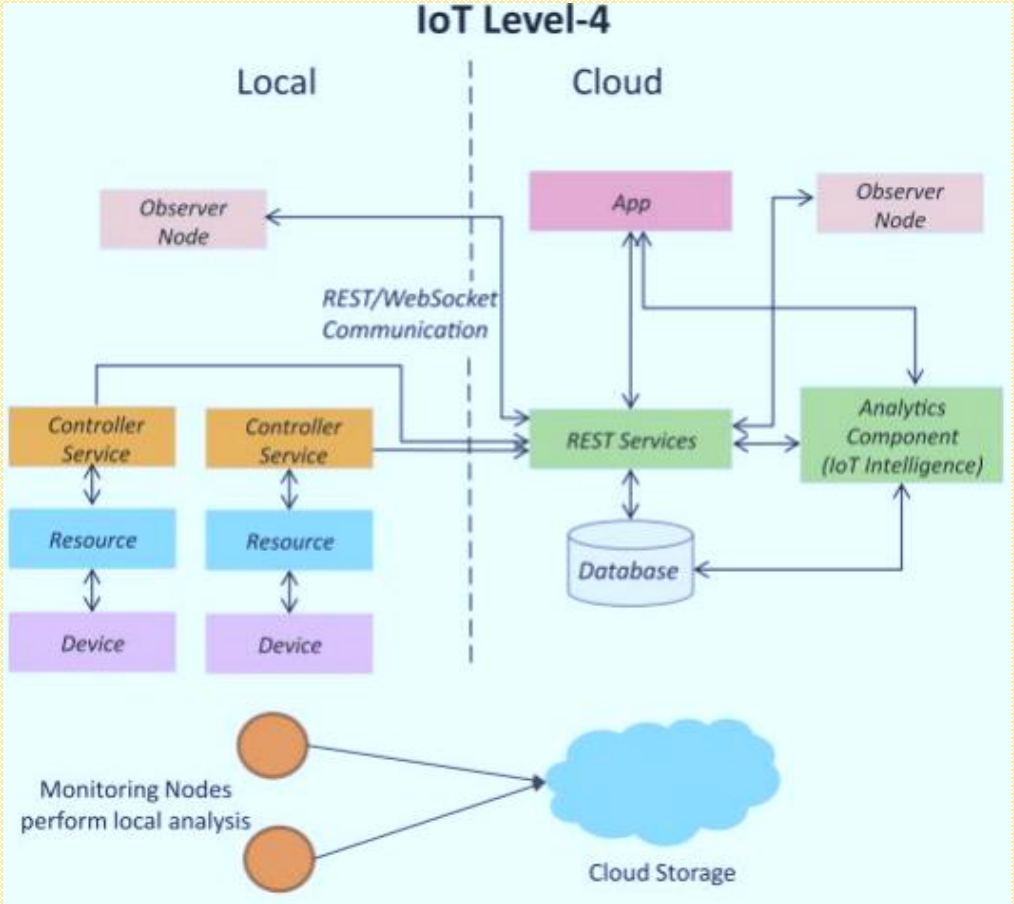
Use Cases for Level-4 IoT Systems

- **Smart City Applications** (e.g., traffic noise, pollution tracking)
- **Environmental Monitoring** (air quality, water levels, forest fire detection)
- **Distributed Industrial Systems**
- **Precision Agriculture with Distributed Sensor Clusters**
- **Large-Scale Surveillance Systems**

IOT LEVELS: IoT Level-4

Characteristics of Level-4 System

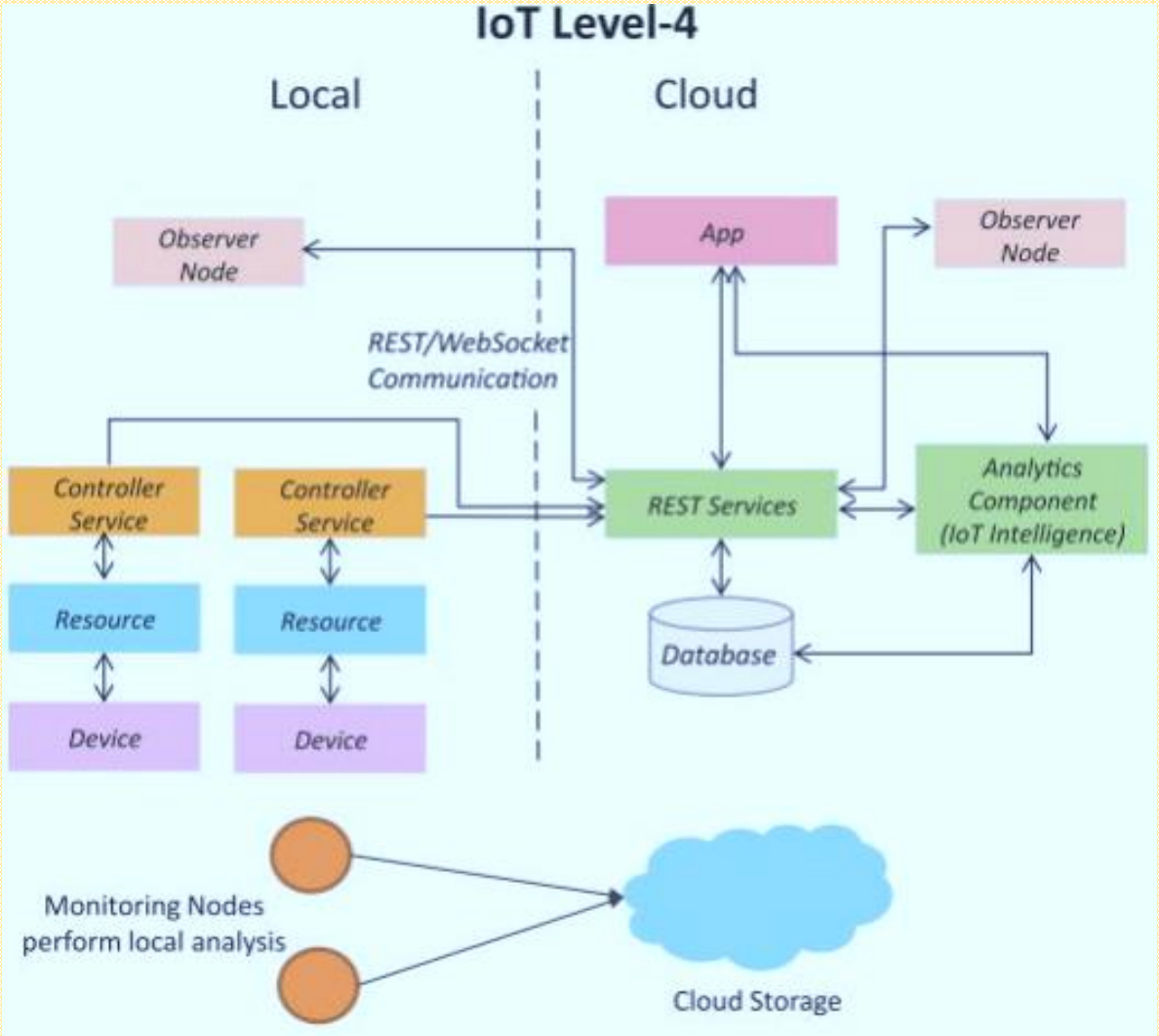
Feature	Description
Nodes	Multiple IoT nodes deployed across different locations
Local Processing	Each node can process data locally using built-in logic or controller services
Cloud Integration	All nodes send their data to the cloud for further analysis and storage
Application Hosting	The main application (UI and logic) resides in the cloud
Scalability	Highly scalable due to the distributed nature of data collection and centralized cloud handling
Data Volume	Suitable for applications involving large amounts of data
Computational Load	The system supports computationally intensive analysis and aggregation in the cloud
Control Scope	Each node can control devices locally; observer nodes only observe and analyze data



IOT LEVELS: IoT Level-4

Architecture Overview

- **Multiple Nodes**: The system uses several independent IoT nodes, each equipped with sensors and controllers. These nodes operate autonomously and perform local analysis on the collected data.
- **Cloud Storage**: The processed or raw data from each node is sent to the cloud, where it is stored in a centralized database.
- **Cloud-Based Application**: A web or cloud-based application aggregates, analyzes, and visualizes data collected from all the nodes.
- **Observer Nodes**: These are cloud-based nodes that subscribe to data feeds from IoT devices. While they process information, they do not control any physical devices.



IOT LEVELS: IoT Level-4

Example: Level-4 IoT system for environmental noise monitoring:

- **System Setup:**
  - Nodes are placed in multiple locations (e.g., street corners, construction sites).
  - Each node consists of:
    - A microcontroller or single-board computer (e.g., Raspberry Pi)
    - A sound sensor (microphone module or noise-level sensor)
    - A controller service to collect and pre-process noise data
- **Local Functionality:**
  - Each node monitors sound levels in real time.
  - It processes the data locally (e.g., to determine decibel levels or detect patterns).
- **Cloud Functionality:**
  - Each node sends data to the cloud.
  - Data is stored in a cloud database and analyzed for trends or thresholds.
  - A cloud-based web application is used to visualize aggregated noise data, view heatmaps, or generate alerts.
- **Observer Nodes:**
  - These are analytics or decision-making systems that subscribe to cloud data for further processing (e.g., to optimize urban planning), but do not control any field device.

IOT LEVELS: IoT Level-5

What is a Level-5 IoT System?

- A Level-5 IoT system consists of multiple end nodes and a coordinator node.
- The end nodes perform sensing and/or actuation, and send data to the coordinator node, which aggregates the data and transmits it to the cloud for storage, analysis, and visualization.
- The application is cloud-based, and the analysis is typically computationally intensive, suited for wireless sensor networks (WSNs) with large-scale, distributed data.

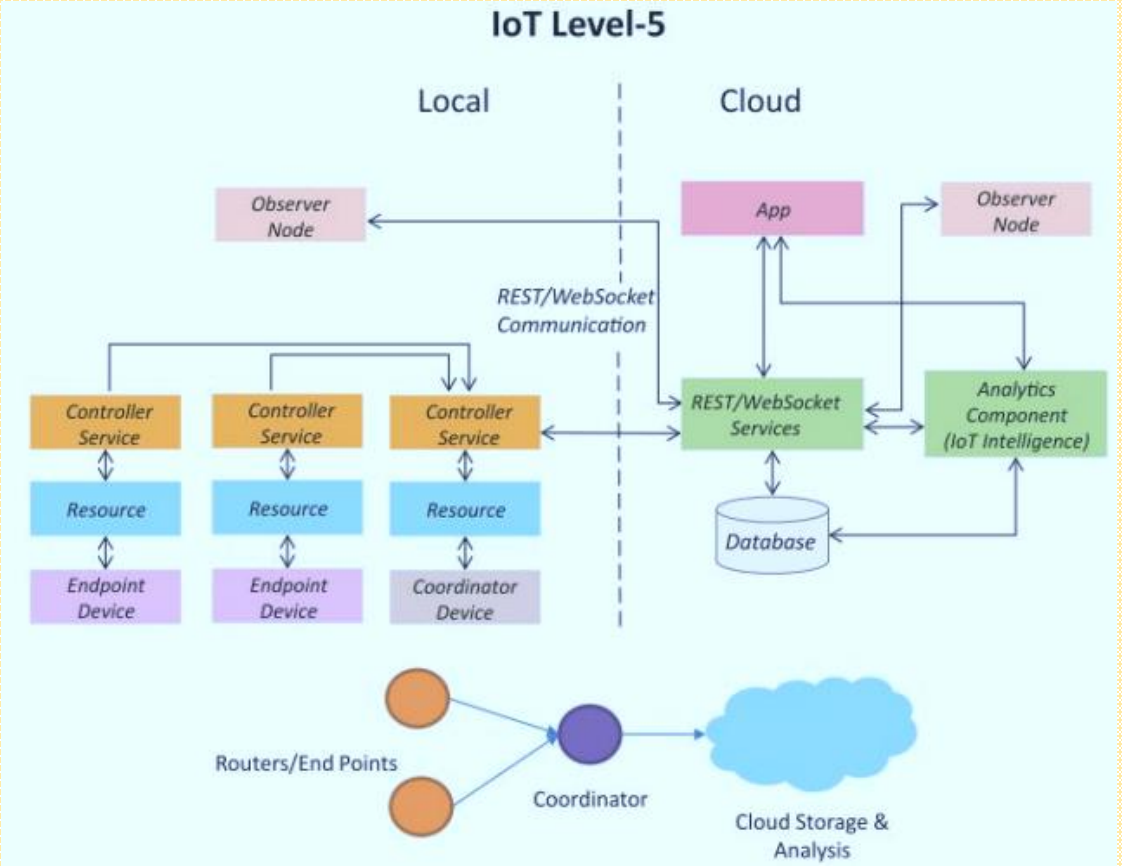
Key Features

- **High scalability** (handles many sensors).
- **Distributed sensing, centralized analysis**.
- **Real-time alerts** for fire risk based on sensor thresholds.
- Suitable for **large-scale environmental monitoring** applications.

IOT LEVELS: IoT Level-5

Example: Forest Fire Detection System (Components Used)

Component Type	Component Name/Description
End Nodes	Microcontrollers with sensor interface (e.g., Arduino, ESP32)
Sensors	Temperature sensor (e.g., DHT11), Humidity sensor, CO_2 sensor (e.g., MG-811)
Coordinator Node	Raspberry Pi / Edge Gateway Device
Communication Protocols	LoRa, Zigbee (for node-to-coordinator), Wi-Fi/Ethernet (coordinator to cloud)
Cloud Platform	AWS IoT, Google Cloud IoT, or Azure IoT Hub
Cloud Database	MongoDB Atlas, Firebase Realtime DB, or Amazon DynamoDB
Cloud App (Dashboard)	Web-based application using tools like Node-RED, Django, or Grafana



IOT LEVELS: IoT Level-5

Example: Forest Fire Detection System (Components Used)

How it Works

1. End Nodes:

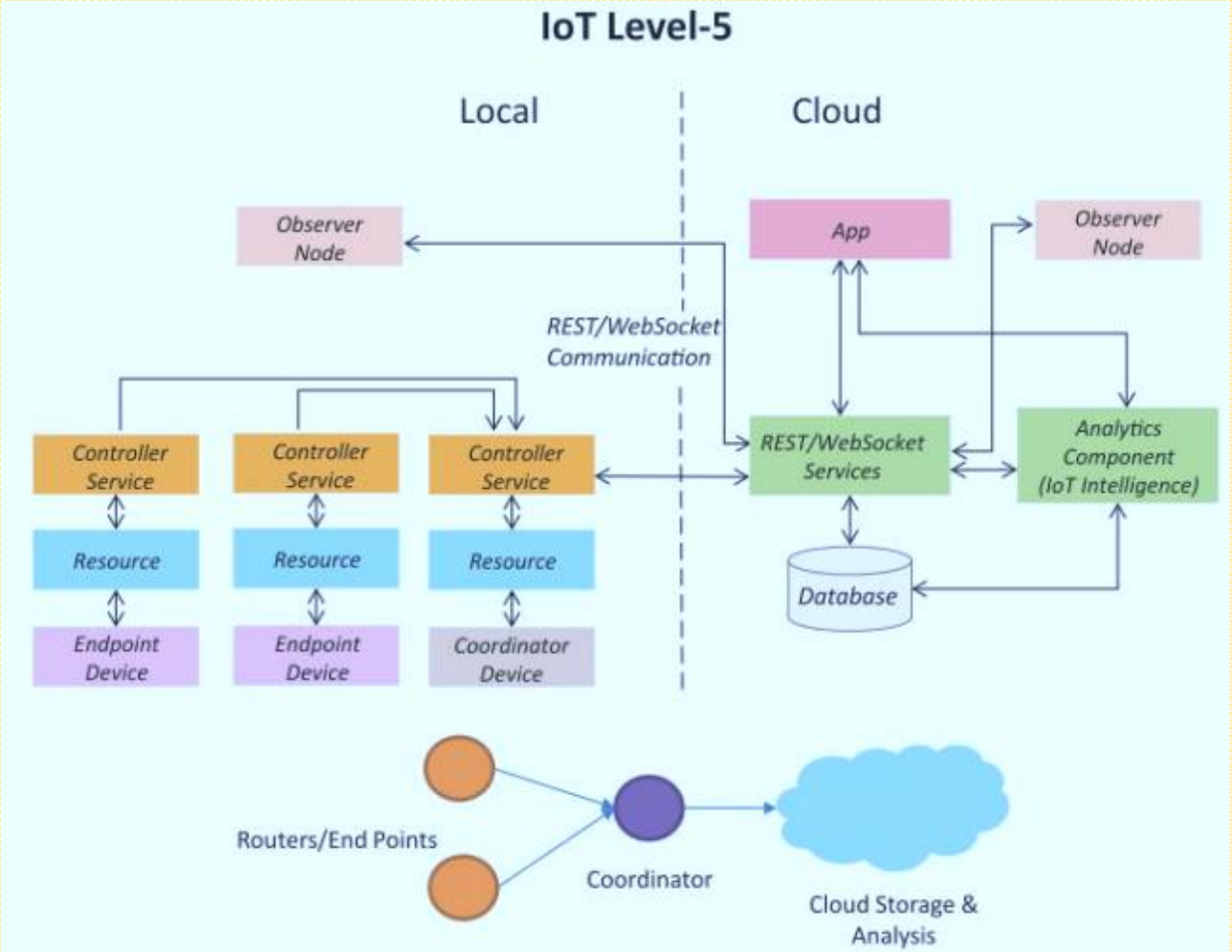
- Deployed in various forest areas.
- Continuously collect data on temperature, humidity, and CO_2 levels.

2. Coordinator Node:

- Gathers sensor data from all end nodes.
- Provides Internet gateway connectivity.

3. Cloud System:

- Stores all data in a cloud database.
- Performs data aggregation and predictive analysis (e.g., fire risk prediction using AI/ML models).
- Displays visualizations through a web dashboard.



IOT LEVELS: IoT Level-6

What is a Level-6 IoT System?

- Multiple independent end nodes (devices) are deployed.
- These end nodes sense (measure) or actuate (control) something.
- They send the data to a cloud platform.
- The cloud stores the data and runs analytics on it.
- Results are shown to users via a cloud-based application (like a dashboard).
- A centralized controller in the cloud keeps track of the status of all end nodes and can send control commands back to them, enabling remote monitoring and control in real-time.

Key Features

- **Real-time monitoring**.
- **Cloud-centric storage and processing**.
- **Remote control** from a central place.
- **Scalable** — you can add more nodes anywhere.

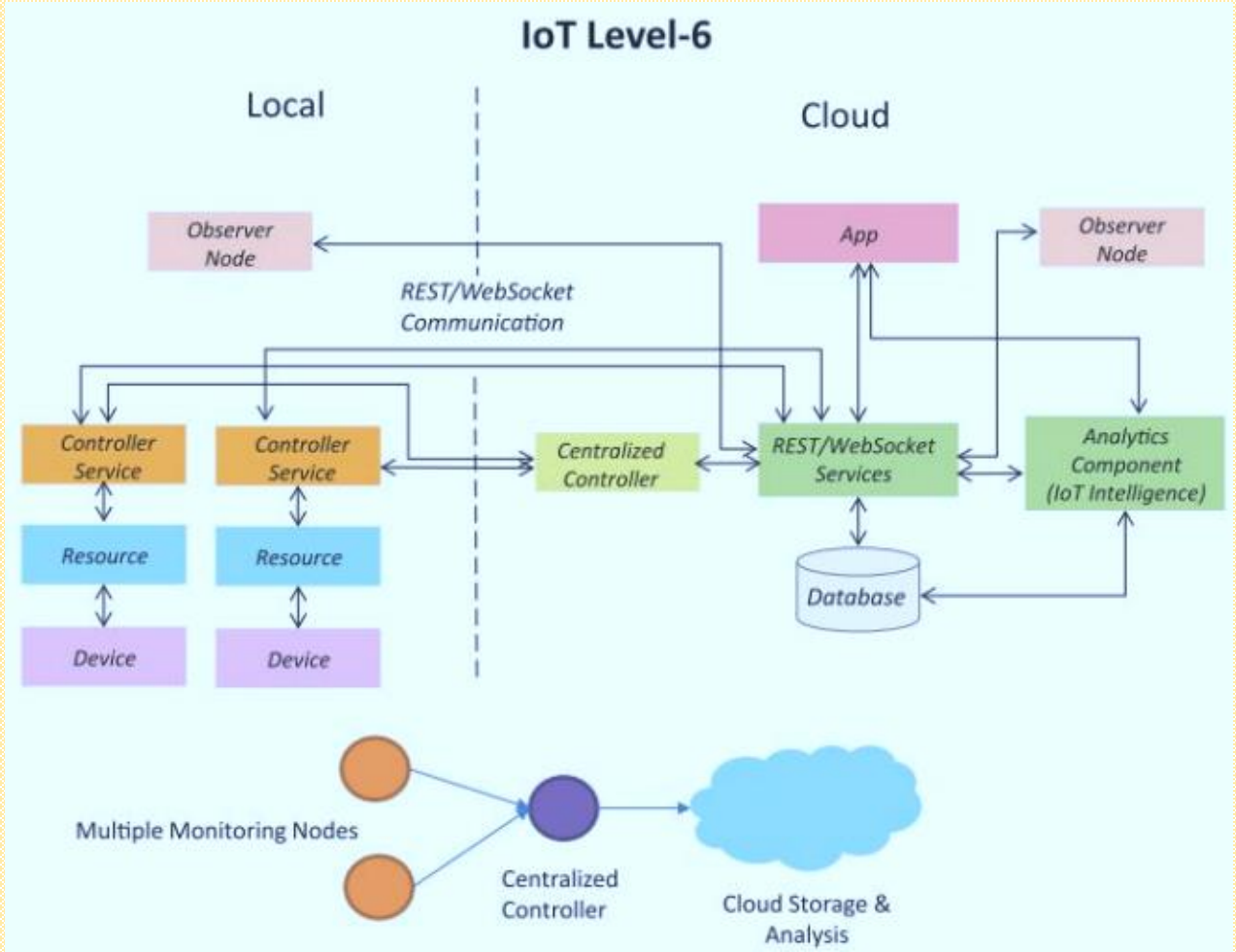
IOT LEVELS: IoT Level-6

Example: Weather Monitoring System

A weather monitoring network with many sensor nodes in different locations, sending real-time temperature, humidity, and pressure data to the cloud, which analyzes and visualizes it for users, while also allowing remote control if needed.

How it Works

- **Sensor Nodes**: Measure data (temperature, humidity, pressure).
- **Cloud**: Stores and analyzes data.
- **Dashboard/App**: Visualizes results for the user.
- **Central Controller**: Monitors all nodes and sends back commands.
- **Actuator Nodes**: Perform actions based on commands (e.g., open a vent).



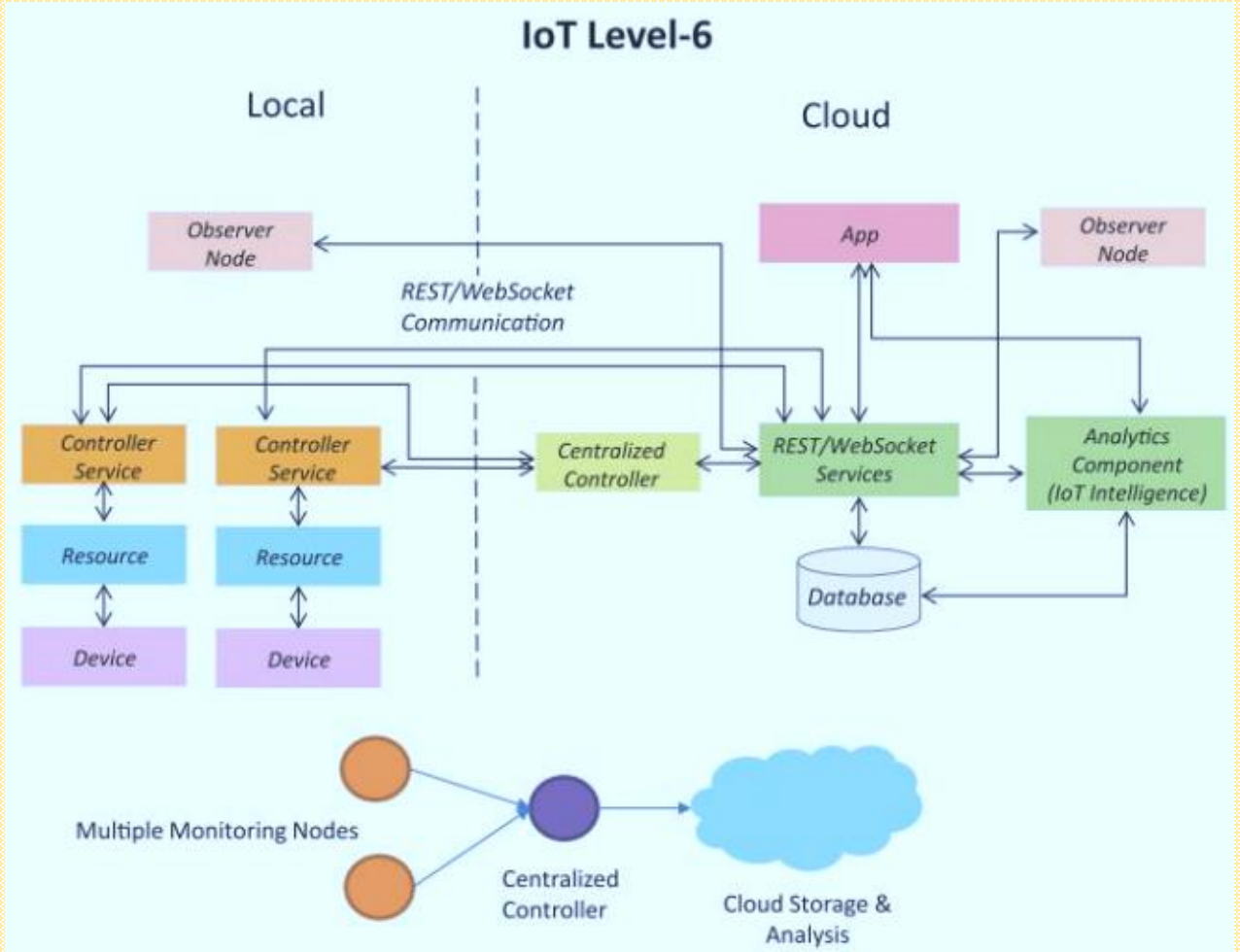
IOT LEVELS: IoT Level-6

Example: Weather Monitoring System

How it Works

Imagine a **city-wide weather monitoring system**:

- **End nodes**: Weather stations in different parts of the city.
- Each station has sensors for temperature, humidity, and pressure.
- These stations send live data to the cloud using WebSocket (real-time).
- In the cloud:
  - Data is stored in a database.
  - Analytics software aggregates data, finds trends, and makes predictions (like tomorrow's weather).
- Users see the results in a web/mobile app.
- If needed, the central controller can send commands (like activating a fan or heater at a remote node).



COMPARISON OF IOT LEVELS (IoT Level 1 – IoT Level 6)

Aspect	IoT Level 1	IoT Level 2	IoT Level 3	IoT Level 4	IoT Level 5	IoT Level 6
System Nodes	Single node	Single node	Single node	Multiple independent nodes	Multiple end nodes + 1 coordinator	Multiple end nodes + 1 Central Controller
Data Storage	Local (on-device)	Cloud	Cloud	Cloud	Cloud	Cloud
Data Analysis	Local	Local	Cloud	Cloud (from multiple nodes)	Cloud (from coordinator)	Cloud (central analytics + control)
Application	Local (on the same device)	Cloud-based	Cloud-based	Cloud-based	Cloud-based	Cloud-based
Communication	REST APIs	REST APIs	WebSocket / REST	REST / WebSocket	REST/WebSocket	WebSocket (real-time)
Control	Local automatic or manual control	Local control + cloud monitoring	Cloud-based control + real-time alerts	No control — observer nodes only	Centralized control via coordinator node	Centralized control in cloud, commands sent to actuators
Example	Home automation (light control)	Smart irrigation system	Package handling system	Noise monitoring (city-wide)	Forest fire detection system	City-wide weather monitoring
Components	Raspberry Pi, LDR, Relay switch	Soil sensors, controller, cloud REST API	Accelerometer, gyroscope, WebSocket API	Sound sensors, multiple nodes	Temp/Humidity/CO_2 sensors, coordinator node	Temp/Humidity/Pressure sensors, actuator nodes, central controller
Analysis Type	Simple status monitoring	Threshold-based actions	Event-triggered, threshold-based	Aggregation, visualization	Aggregation, pattern recognition	Real-time, predictive + control
Scalability	Low	Low–Moderate	Moderate	High	High	Very High
Use Case Type	Low-cost, low-complexity	Low-cost + Cloud storage	Cloud-driven alert system	City-scale sensing + visualization	Environmental monitoring + decision-making	Scalable, real-time monitoring + remote control

Machine-to-Machine (M2M)

- **M2M**: It is a network of machines (or devices) for the purpose of monitoring, control and data exchange.
- M2M happens when machines automatically exchange information between them, without relying on human intervention or middleware devices.
- According to the strictest definitions, there should be absolutely zero human intervention or other devices in between for communication between two devices. However, there might be some human intervention or servers in the middle of data transfer between two devices in practical use cases.
- **Examples:**
  - A vending machine relaying its stock levels to the supply chain software,
  - An ATM asking for authorization to dispense cash from the bank servers,
  - A credit card reader authorizing a transaction, etc.

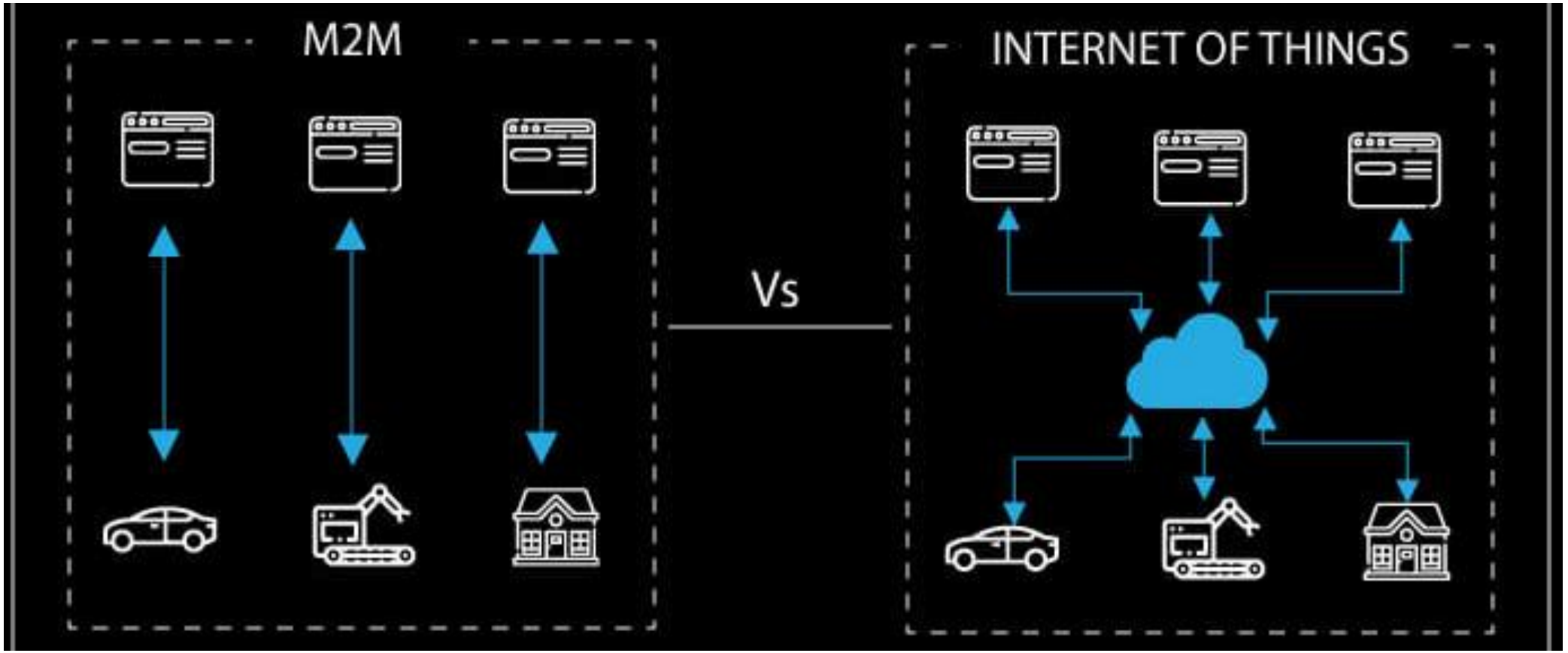
Machine-to-Machine (M2M)

- **Examples:**
  - A washing machine/dryer connected with M2M communication capabilities can be used to send a signal at the end of the washing/drying cycle to the users. The machine will have smart capabilities to enable communication to send the signal to the user.
  - HVAC systems with M2M communication capabilities can be turned on in advance using a smartphone. The house/room will have the perfect temperature by the time you arrive home.
  - Smart refrigerators can analyze what is left in them and automatically create a shopping list according to routine needs. It can even have the capability to order the products in time to arrive fresh in time for consumption.

Working of Machine-to-Machine (M2M)

- Machine to machine (M2M) is a comprehensive concept that can be used to describe any technology that allows networked devices to exchange information and inform actions automatically without human interference.
- M2M communication devices system is standalone network equipment that uses point-to-point communication between machines, sensors and hardware.
- Today, M2M is among the fastest-growing technologies because it can connect millions of devices within a single network.
- M2M technology’s purpose is to collect the data from sensors and transmit them to the cloud. It uses a mobile network for transmission purposes, which makes it a cost-effective technology. The working of M2M is that the data is collected from the device sensors, sent to the cloud via mobile towers.

Machine-to-Machine (M2M)



Wired M2M Communication

Wired and Wireless M2M Communication

The two main types of M2M communication are wired communication and wireless communication. Each of these functions a bit differently.

Wired M2M Communication

- In wired M2M communication, the data transfer between the devices occurs over a wired transfer medium. It can be fiber optic cables, EtherCAT, or even coaxial cables. The communication between devices in the same LAN network connected with only wired connections is also part of wired M2M communication.
- Wired communication networks are becoming rarer now. Most use cases are moving to wireless networks.
- Wired networks are only used in cases where wireless signals are susceptible to interference.
- It is also used in legacy systems that can be difficult to upgrade communication networks to a wireless system.

Wireless M2M Communication

When machines use wireless communication methodology to communicate between devices, it is wireless M2M communication. M2M communication that majorly uses the wireless network for communication is termed as Internet of Things or IoT. The wireless communication methodologies used have a wide range from radio waves to the latest 5G technology.

- **RFID** – RFID or Radio Frequency Identification is quite an old technology that has stood the test of time. RFID technology communicates over radio wave frequency to enable wireless transfer of data. Now it is widely used in the logistics industry to track a package from start to finish. It is also used for provenance technology block chains.
- **NFC** – Near-Field Communication or NFC is also similar to RFID but can only be used for short-range transfer of data. It is widely used for access control and payment systems. Payment applications like Apple pay and contactless credit cards make use of NFC technology to transfer data. It can also be used as an efficient mode of communication to transfer information between smart phones within a short range.

Wireless M2M Communication

- **WiFi** – WiFi or wireless fidelity is widely used in homes and offices to access the internet wirelessly. The same network can also transfer data between devices connected to the same WiFi network. There have been many newer iterations of WiFi technology that increased the bandwidth and reduced communication latency. The latest WiFi technology that FCC has approved is WiFi 6e and can bring massive improvement for IIoT communication.
- **Cellular Network** – Cellular networks can also be made use of for wireless M2M communication. The very first wireless credit card readers used GSM technology or other 2G technologies for communication. Later communication was possible over HSPA (3G) and LTE (4G) networks. Now wireless communication is at the precipice of disruption with 5G communication technologies. 5G dramatically improves over 4G technology in terms of bandwidth and latency. Thousands of devices would be able to ‘talk to each other’ very fast and with very minimal lag over a 5G network. Implementation of 5G in the industrial setting is said to be akin to a new industrial revolution in the works.

Applications of Machine-to-Machine (M2M)

1. **Security :** Surveillances, Alarm systems, Access control, Car/driver security
2. **Tracking & Tracing :** Fleet Management, Order Management, Pay as you drive, Asset Tracking, Navigation, Traffic information, Road tolling, Traffic optimization/steering
3. **Payment :** Point of sales, Vending machines, Gaming machines
4. **Health :** Monitoring vital signs, Supporting the aged or handicapped, Web Access Telemedicine points, Remote diagnostics
5. **Remote Maintenance/Control :** Sensors, Lighting, Pumps, Valves, Elevator control, Vending machine control, Vehicle diagnostics
6. **Metering :** Power, Gas, Water, Heating, Grid control, Industrial metering
7. **Manufacturing :** Production chain monitoring and automation
8. **Facility Management :** Home / building / campus automation

Key Features of Machine-to-Machine (M2M)

Some of the key features of M2M communication system are given below:

1. **Low Mobility :** M2M Devices do not move, move infrequently, or move only within a certain region
2. **Time Controlled :** Send or receive data only at certain pre-defined periods
3. **Time Tolerant :** Data transfer can be delayed
4. **Packet Switched :** Network operator to provide packet switched service with or without an MSISDN
5. **Online small Data Transmissions:** MTC Devices frequently send or receive small amounts of data.
6. **Monitoring:** Not intend to prevent theft or vandalism but provide functionality to detect the events
7. **Low Power Consumption :** To improve the ability of the system to efficiently service M2M applications
8. **Location Specific Trigger :** Intending to trigger M2M device in a particular area e.g. wake up the device

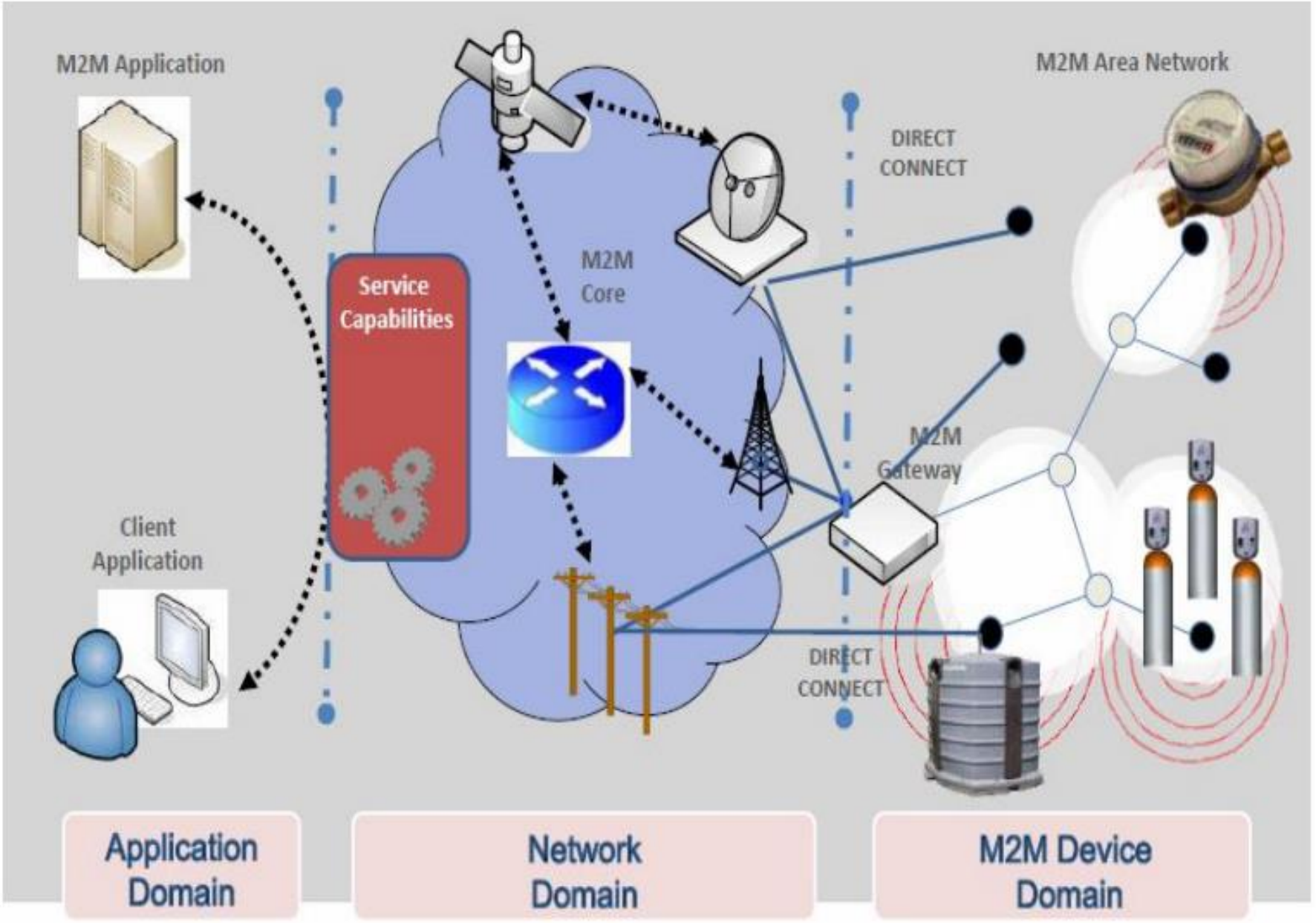
Architecture & Components of Machine-to-Machine (M2M)

1. **M2M Device:**

- Device capable of replying to request for data contained within those devices or capable of transmitting data autonomously.
- Sensors and communication devices are the endpoints of M2M applications. Generally, devices can connect directly to an operator's network, or they will probably interconnect using WPAN technologies such as ZigBee or Bluetooth.
- Backhaul to an operator's network is than achieved via gateways that encapsulate and manage all devices. Consequently, addressing and identifying, e.g., routing, of the devices relies heavily on the gateways. Devices that connect via gateways are normally outside the operator's responsibility but belong to M2M applications that are provided by service or application providers.

2. **M2M Area Network (Device Domain):** Provide connectivity between M2M Devices and M2M Gateways, e.g. personal area network.

Architecture & Components of Machine-to-Machine (M2M)



Architecture & Components of Machine-to-Machine (M2M)

2. M2M Gateway:

- Equipment that uses M2M capabilities to ensure M2M Devices inter-working and interconnection to the communication network.
- Gateways and routers are the endpoints of the operator’s network in scenarios where sensors and M2M devices do not connect directly to the network.
- An **M2M Gateway (Machine-to-Machine Gateway)** is an important component in **IoT and MTC (Machine-Type Communication)** systems.
- It acts as a **bridge between devices (sensors, machines, controllers) and external networks (cloud, internet, or enterprise systems).**

Architecture & Components of Machine-to-Machine (M2M)

Functions of an M2M Gateway

1. **Protocol Translation**

   Devices use different communication protocols (Zigbee, Bluetooth, Modbus, CAN, etc.).

   The gateway **translates these into standard IP-based protocols** (like MQTT, CoAP, HTTP).

   Example: A Zigbee-based smart bulb communicating with a cloud server over MQTT via the gateway.

2. **Data Aggregation & Filtering**

   Collects data from multiple devices.

   Filters or aggregates before sending to reduce bandwidth usage.

   Example: Instead of sending every single temperature reading, it may send average, min, and max values every 5 minutes.

Architecture & Components of Machine-to-Machine (M2M)

Functions of an M2M Gateway

3. **Security & Authentication**

   Provides device authentication, data encryption, and secure communication with cloud servers.

   Example: Encrypts sensor data before uploading to AWS IoT or Azure IoT Hub.

4. **Local Processing (Edge Computing)**

   Performs basic analytics and decision-making locally before sending data to the cloud.

   Reduces latency and dependency on internet connectivity.

   Example: In smart homes, the gateway can turn on a fan immediately when temperature > 30°C, without waiting for cloud commands.

5. **Network Bridging**

   Connects local non-IP devices to IP networks (like 4G/5G, Wi-Fi, Ethernet).

   Example: Connecting LoRaWAN sensor nodes to the internet via cellular.

Architecture & Components of Machine-to-Machine (M2M)

6. Device & Resource Management

Manages device configuration, firmware updates (OTA), and monitoring.

Example: Updating all smart meters in a city with new software via the gateway.

Example of M2M Gateway in Real Life

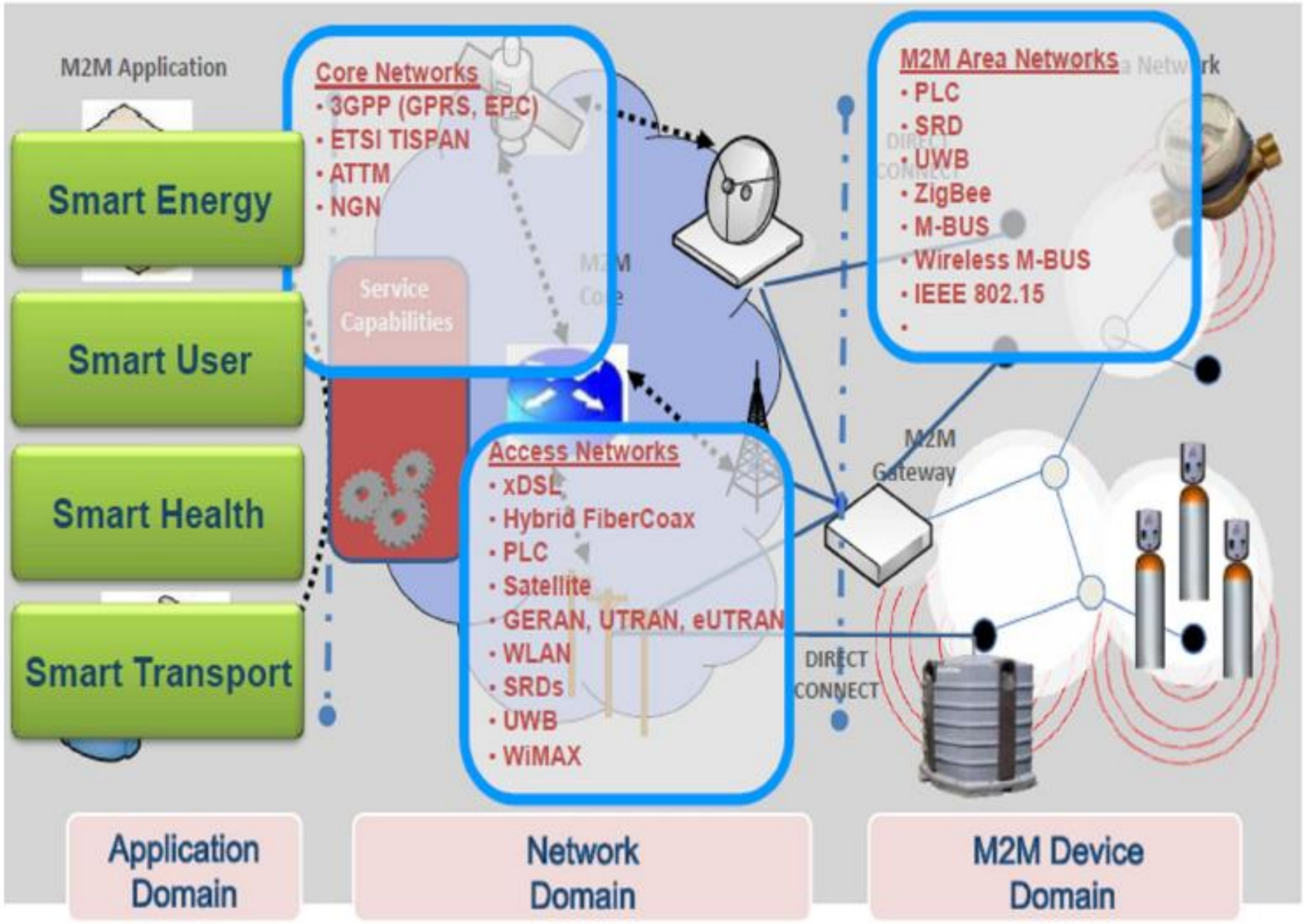
- **Smart City**: Streetlights with Zigbee controllers → Gateway translates to MQTT → Sends data to cloud for monitoring.
- **Industrial IoT**: Machines use Modbus → Gateway converts to OPC-UA/MQTT → Cloud for predictive maintenance.
- **Smart Home Hub (like Amazon Echo, Google Nest Hub)** → Works as an M2M Gateway between IoT devices and the internet.

Architecture & Components of Machine-to-Machine (M2M)

3. **M2M Communication Networks (Network Domain):** It covers the communications between the M2M Gateway(s) and M2M application(s), e.g. xDSL, LTE, WiMAX, and WLAN.

4. **M2M Applications:** It contains the middleware layer where data goes through various application services and is used by the specific business-processing engines. M2M applications will be based on the infrastructural assets (e.g., access enablers) that are provided by the operator. Applications may either target at end users, such as user of a specific M2M solution, or at other application providers to offer more refined building blocks by which they can build more sophisticated M2M solutions and services. e.g. customer care functionality, elaborate billing functions, etc. Those services, or service enablers, may be designed and offered by an application provider, but they might be offered by the operator via the operator platform itself.

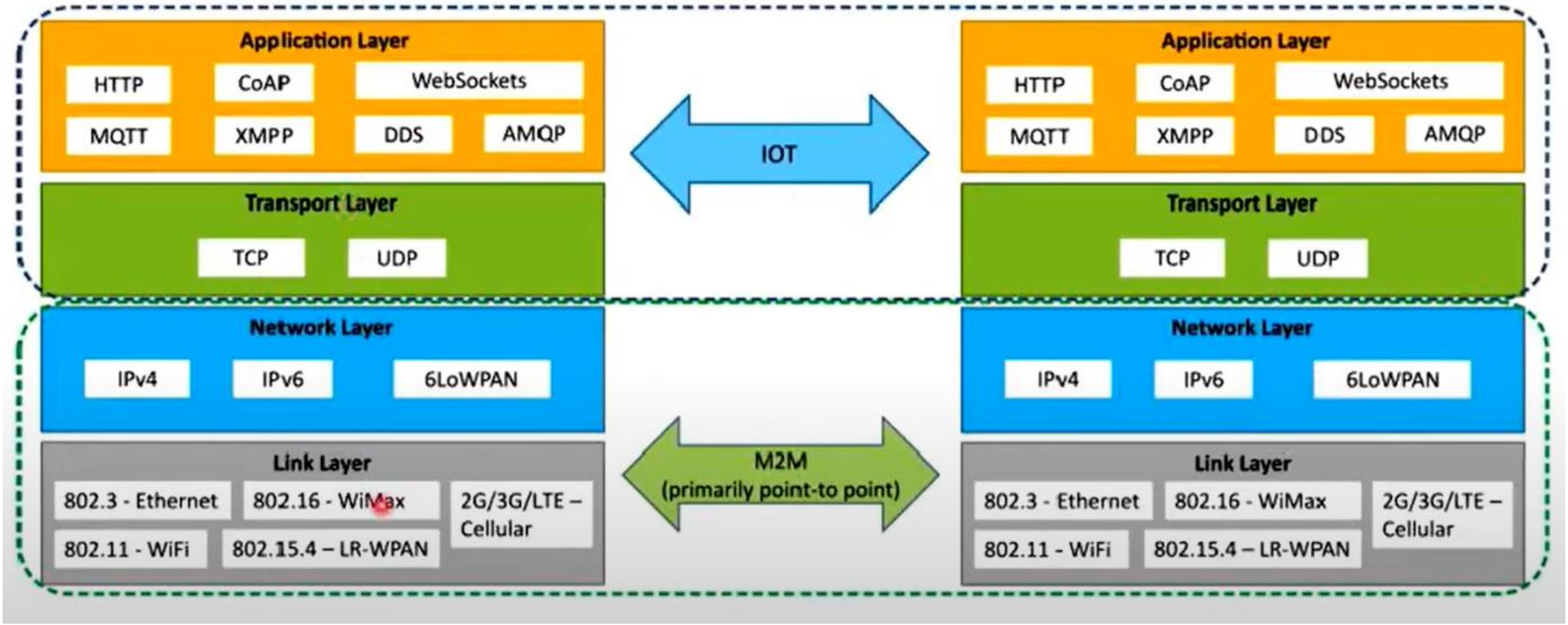
Architecture & Components of Machine-to-Machine (M2M)



IoT Vs M2M

Topic	M2M	IoT
Communication Protocols	Proprietary or Non-IP Based	IP Based
Machines in M2M vs Things in IoT	Homogeneous Machine	Physical Objects that have Unique Identifiers
Hardware vs Software Emphasis	More On Hardware	More On Software
Data Collection & Analysis	Collected In Point Solutions And Often In On-premises Storage	Collected In The Cloud (Can Be Public, Private Or Hybrid Cloud)
Applications	Diagnosis Applications, Service Management Applications, And On-premises Enterprise Applications	Analytics Applications, Enterprise Applications, Remote Diagnosis And Management Applications, etc.

IoT Vs M2M


Application Layer
HTTP
CoAP
WebSockets
MQTT
XMPP
DDS
AMQP
Transport Layer
TCP
UDP
Network Layer
IPv4
IPv6
6LoWPAN
Link Layer
802.3 - Ethernet
802.16 - WiMax
2G/3G/LTE – Cellular
802.11 - WiFi
802.15.4 – LR-WPAN
IOT
M2M
(primarily point-to point)
Application Layer
HTTP
CoAP
WebSockets
MQTT
XMPP
DDS
AMQP
Transport Layer
TCP
UDP
Network Layer
IPv4
IPv6
6LoWPAN
Link Layer
802.3 - Ethernet
802.16 - WiMax
2G/3G/LTE – Cellular
802.11 - WiFi
802.15.4 – LR-WPAN

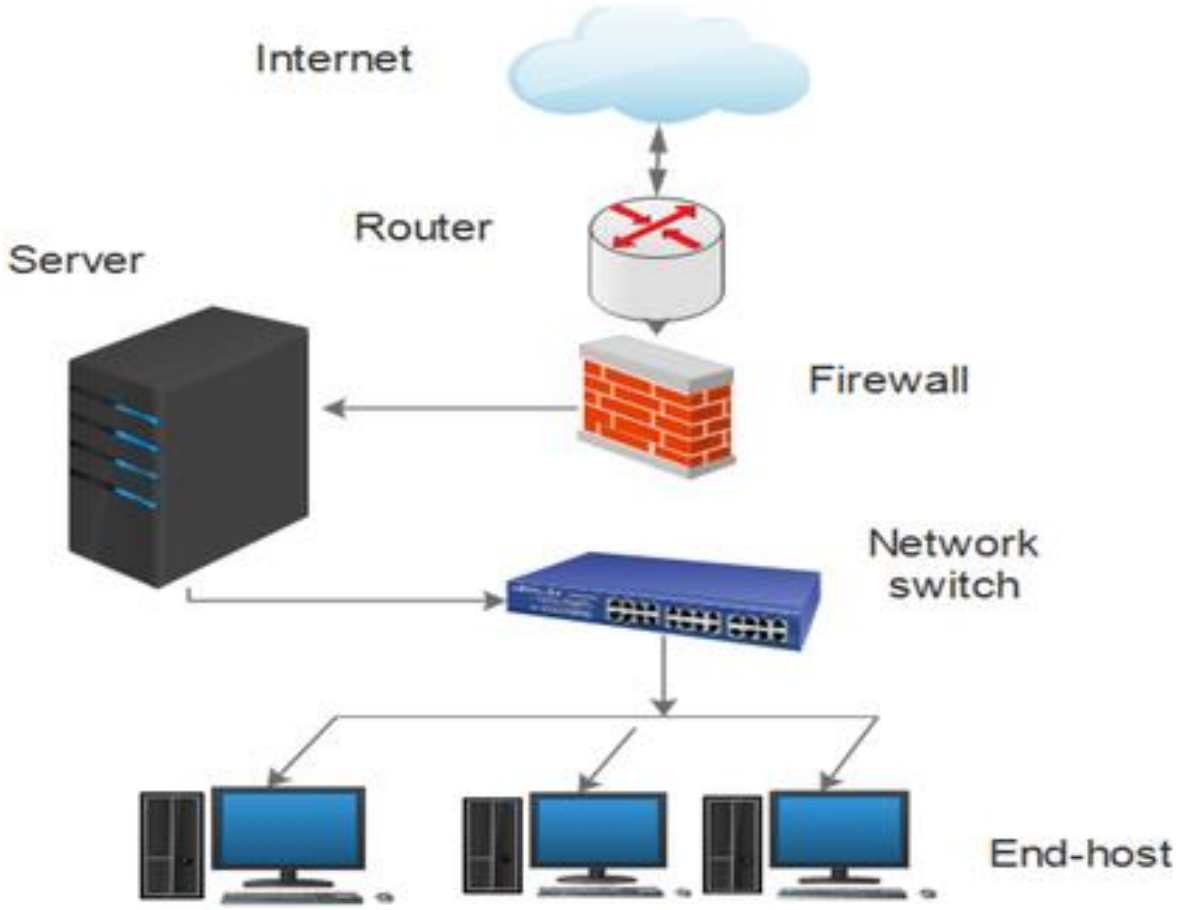
SDN (Software Defined Network)

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently and centrally controlled, or ‘programmed,’ using software applications. This helps operators manage the entire network consistently and holistically, regardless of the underlying network technology.

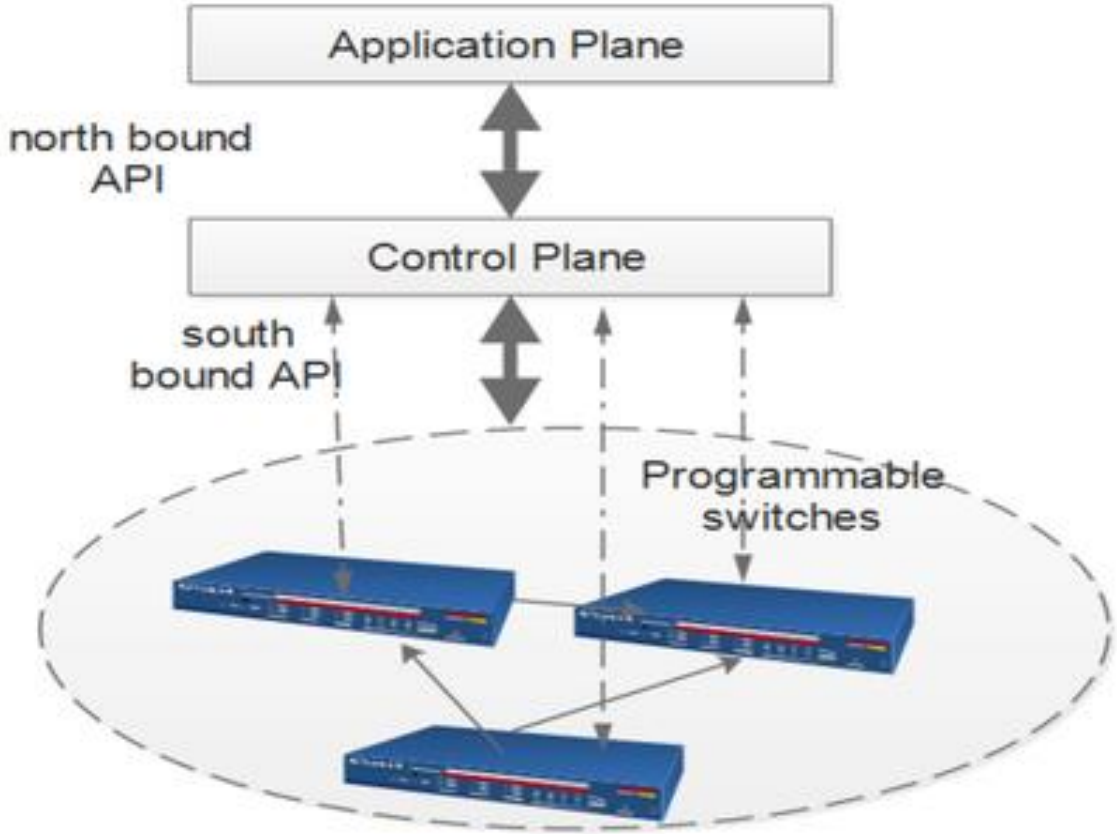
This model differs from that of traditional networks, which use dedicated hardware devices (i.e., routers and switches) to control network traffic. SDN can create and control a virtual network – or control a traditional hardware – via software.

While network virtualization allows organizations to segment different virtual networks within a single physical network, or to connect devices on different physical networks to create a single virtual network, software-defined networking enables a new way of controlling the routing of data packets through a centralized server.

SDN (Software Defined Network)

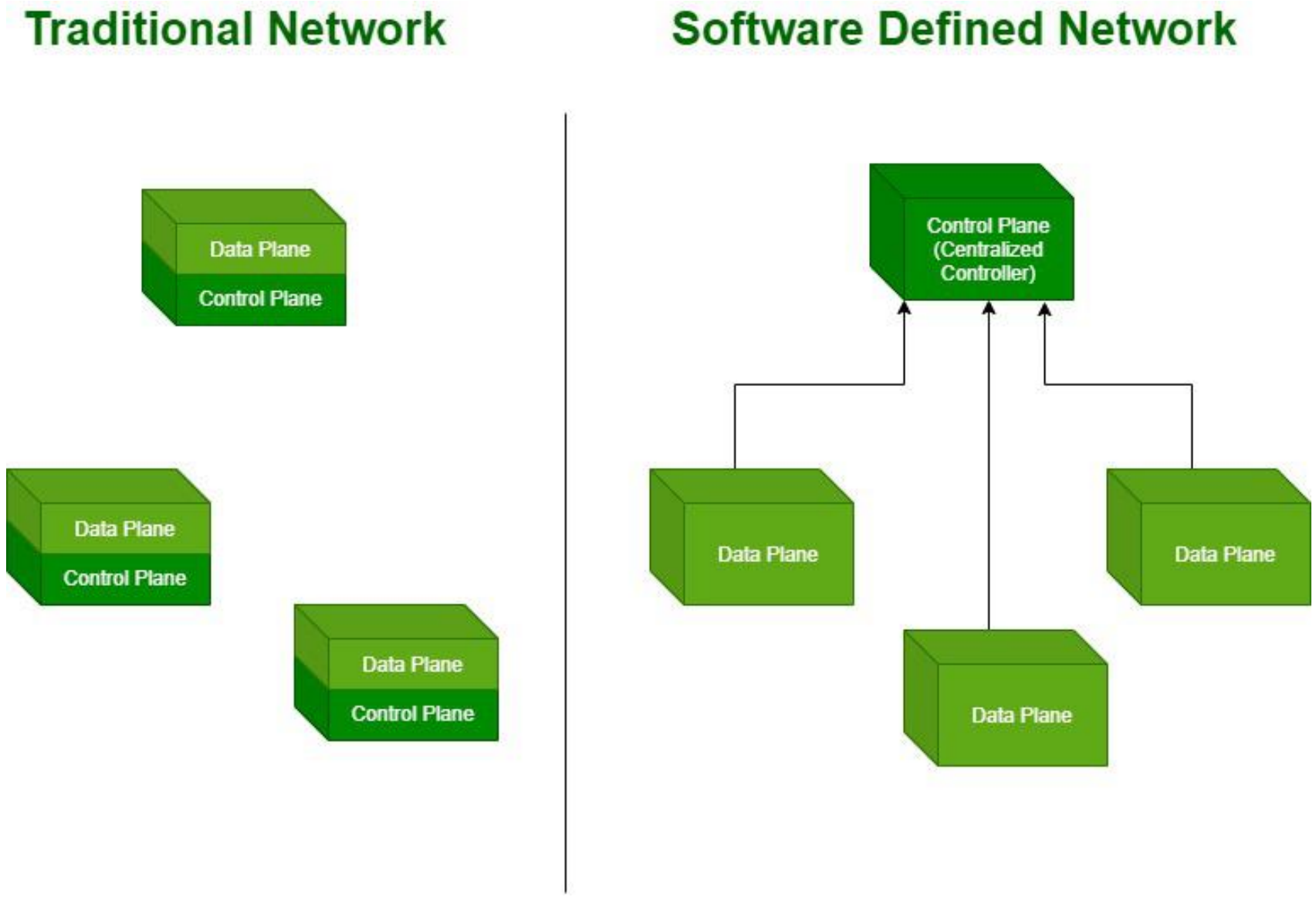


a Traditional Network architecture



b SDN architecture

SDN (Software Defined Network)



SDN (Software Defined Network)

Software-Defined Networking	Traditional Networking
It uses a virtualized approach to manage networks.	It depends on dedicated hardware devices to control network traffic.
It has centralized control using a software-based controller.	It has distributed control, with each device managing its own operations.
It is programmable. So it is highly flexible.	It is non-programmable. So it is less adaptable to changes.
It supports open interfaces for interoperability.	It depends on proprietary systems from single vendors.
Data and control planes are separated.	Data and control planes are combined in each device.
You can automate configuration. So it can save time.	It requires manual configuration. It takes more time.
It can prioritize specific network packets based on needs.	It handles all network traffic in the same way, without prioritization.

SDN (Software Defined Network)

Software-Defined Networking	Traditional Networking
It is easier to program and reprogram as needs evolve.	It is tough to modify and reprogram the network once its in place.
It is cost-effective due to simplified hardware needs.	It has higher hardware costs because of specialized devices.
It has lower structural complexity to manage.	It has higher structural complexity. So it is tough to manage.
It is easier to troubleshoot and report issues due to centralized control.	Its troubleshooting is tough because of distributed control.
Its maintenance costs are lower than traditional networks.	Its maintenance costs have higher than SDN.

SDN Architecture

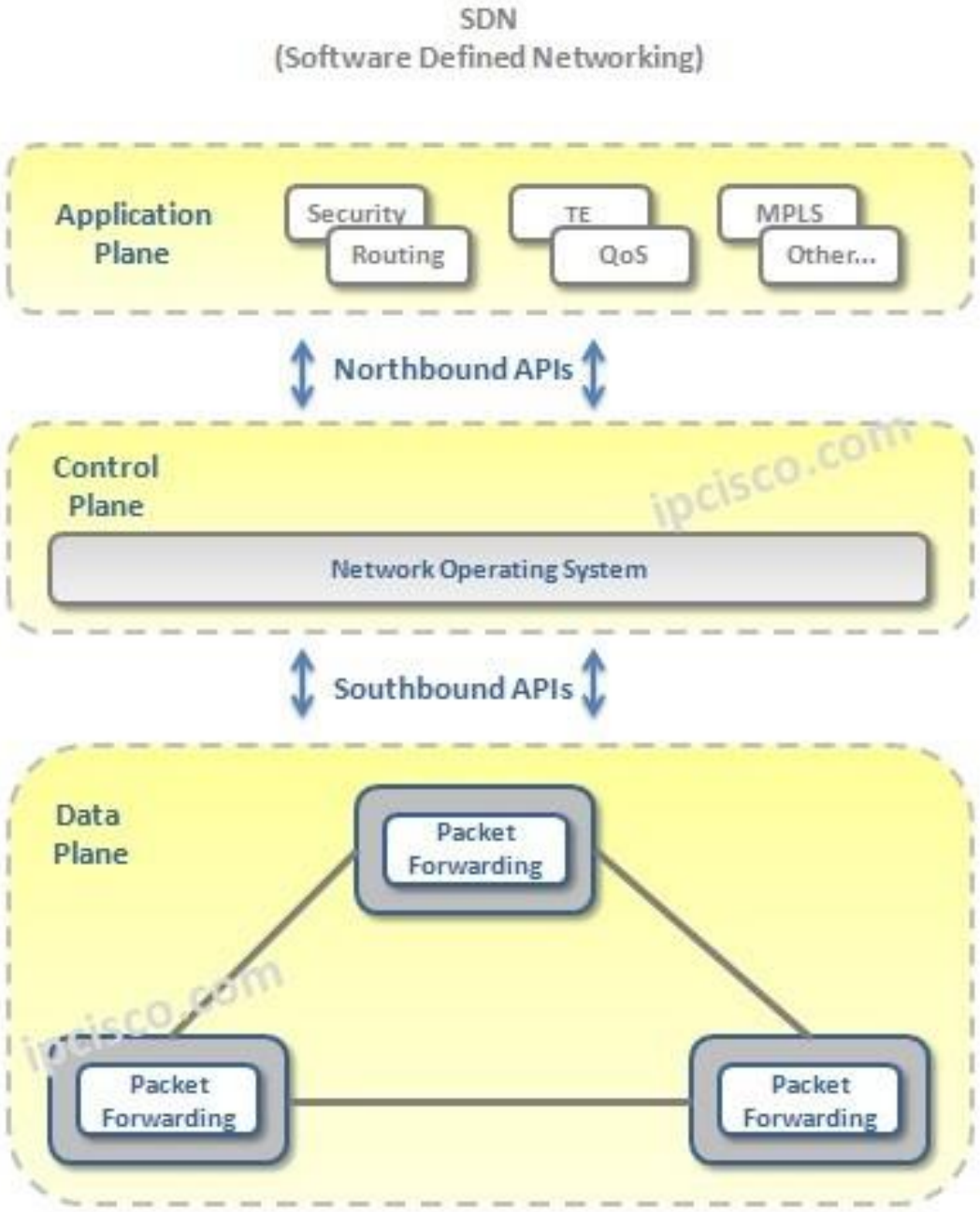
The SDN Architecture is:

- **DIRECTLY PROGRAMMABLE:** Network control is directly programmable because it is decoupled from forwarding functions.
- **AGILE:** Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
- **CENTRALLY MANAGED:** Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- **PROGRAMMATICALLY CONFIGURED:** SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
- **OPEN STANDARDS-BASED AND VENDOR-NEUTRAL:** When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

SDN (Software Defined Network)

SDN Architecture consist of different components. Here, we will see all these SDN Architecture Components and their duty one by one. What are these SDN Components? The basic SDN Terms, SDN Components are:

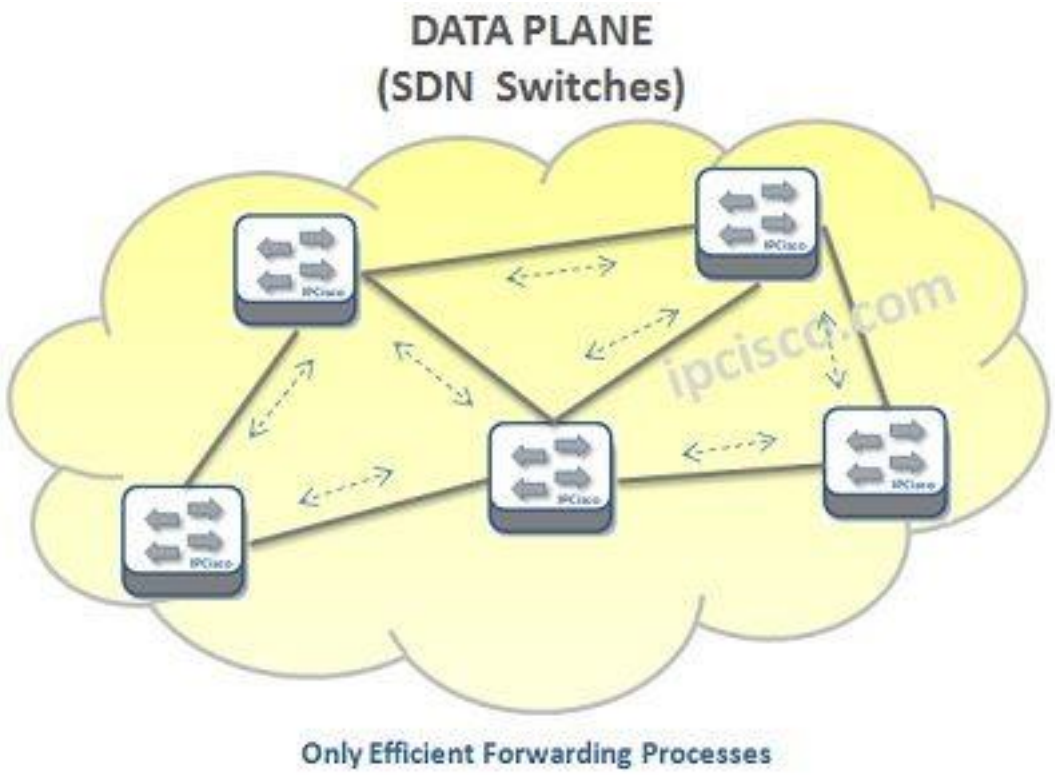
- Network Devices (Data Plane)
- SDN Controller (Control Plane)
- Southbound Interface
- Northbound Interface
- Network Operating System (NOS)
- Application and Services (Application Plane)



SDN (Software Defined Network)

SDN Architecture : Network Devices (Data Plane)

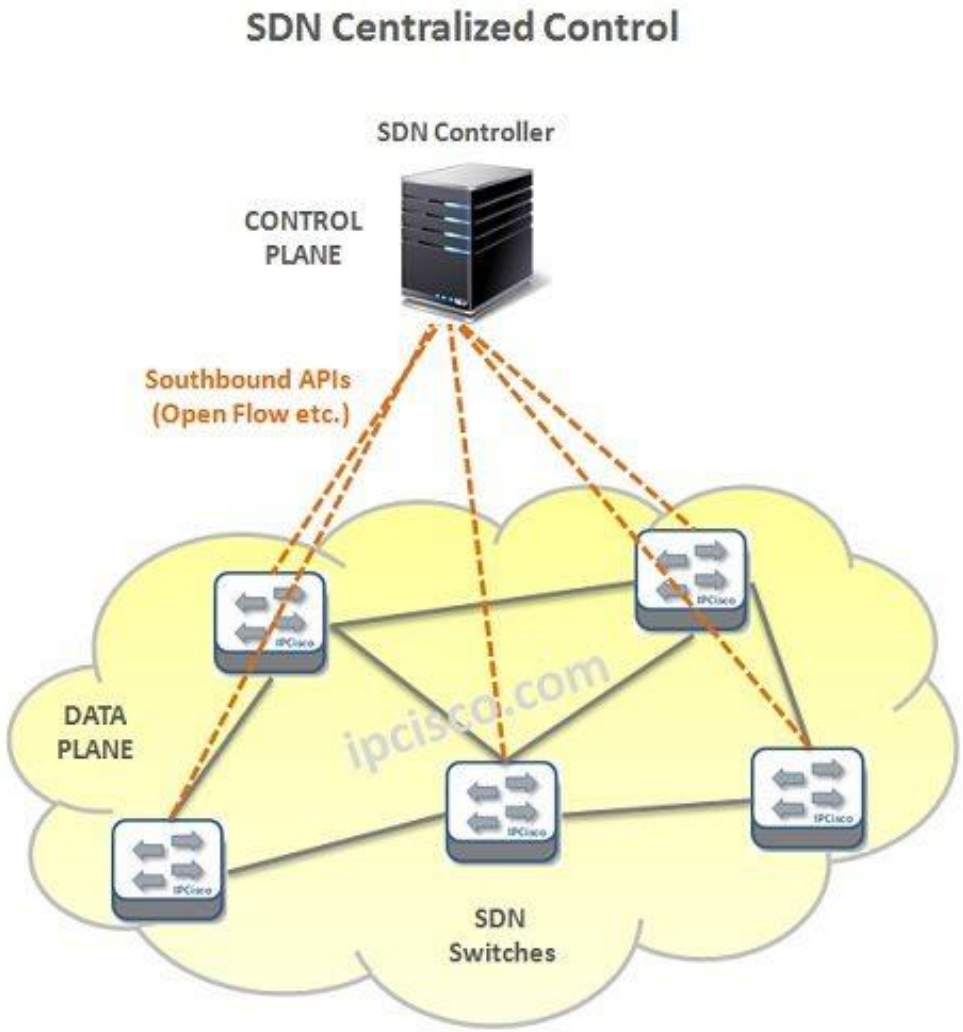
- The **data plane** (also called the forwarding plane) is made up of network devices like switches, routers, and access points that forward packets based on flow rules.
- **Functions:**
  - It **forwards data packets** between network endpoints according to the flow tables programmed by the SDN controller.
  - It does **not make routing decisions** — it just executes what the controller instructs.
- **Example:**
  - OpenFlow-enabled switches that take instructions from the SDN controller about how to handle specific traffic flows.



SDN (Software Defined Network)

SDN Architecture : SDN Controller (Control Plane)

- SDN Controller is the Center of the SDN Architecture and the most important one of SDN Architecture Components.
- The **brain of the SDN network**. It is a logically centralized software component that manages and controls all data plane devices.
- SDN Controller communicate and control these upper and lower layer with APIs through Interfaces.



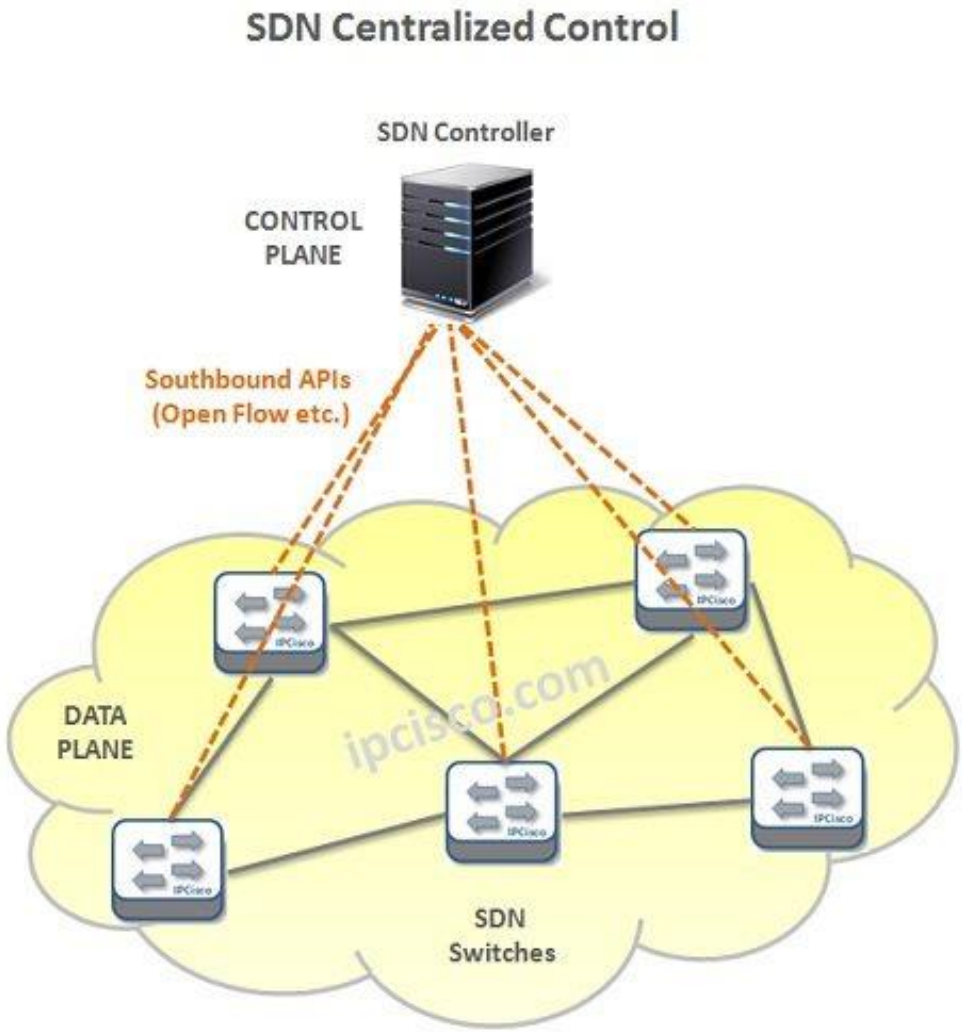
- **Functions:**
  - Maintains a **global view** of the entire network.
  - Installs forwarding rules on switches (flow entries).
  - Optimizes traffic flow, enforces security policies, and performs load balancing.
- **Examples of Controllers:**

  OpenDaylight, ONOS, Ryu, Floodlight.

SDN (Software Defined Network)

SDN Architecture : Southbound APIs

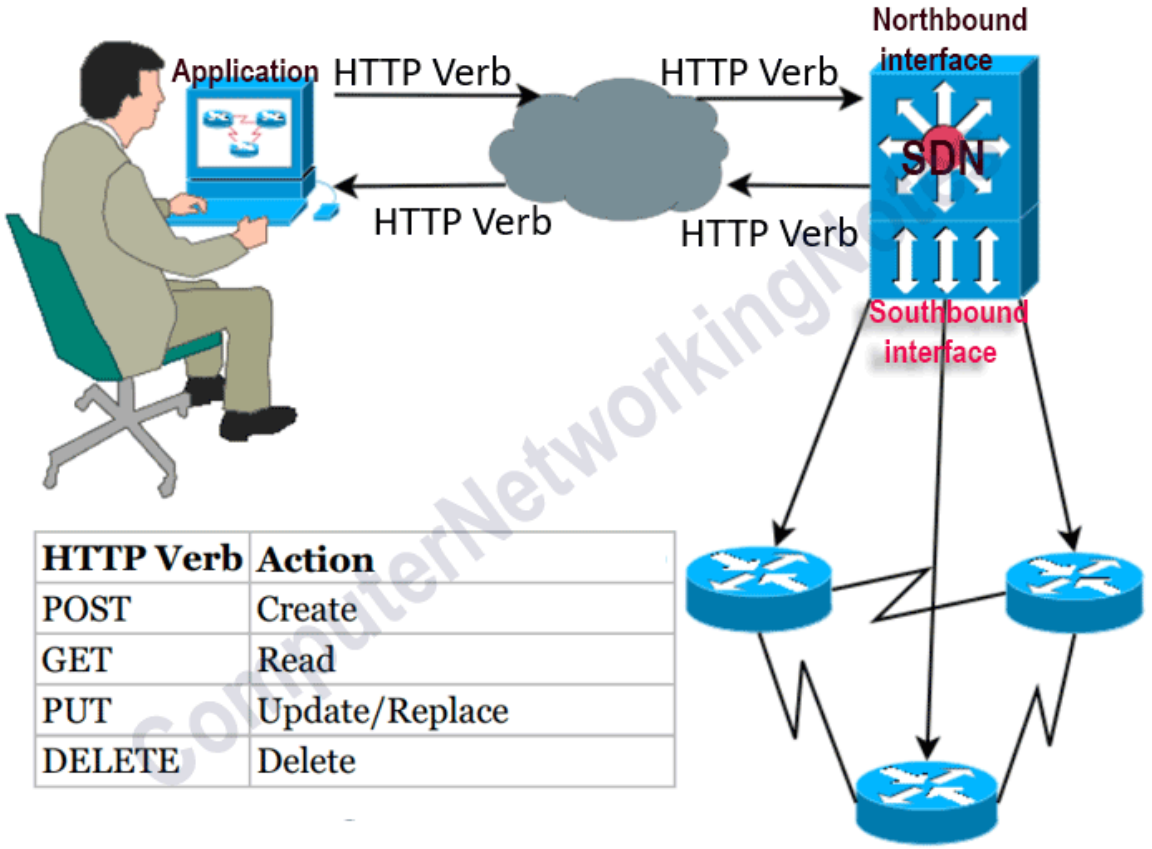
- The communication channel between the SDN controller (control plane) and network devices (data plane).
- **Functions:**
  - Allows the controller to **send instructions** (like flow rules) to switches and receive status updates back.
  - Ensures compatibility between hardware and software layers.
- **Common Protocols:**
  - **OpenFlow** (most popular)
  - NETCONF, BGP-LS, P4Runtime, gNMI



SDN (Software Defined Network)

SDN Architecture : Northbound APIs

- The communication channel between the SDN controller and applications/services (application plane).
- **Functions:**
  - Provides **APIs** (usually REST APIs) that allow applications to request network information or instruct the controller to configure the network.
  - Enables **programmability** — network admins can write custom apps for security, QoS, load balancing, etc.

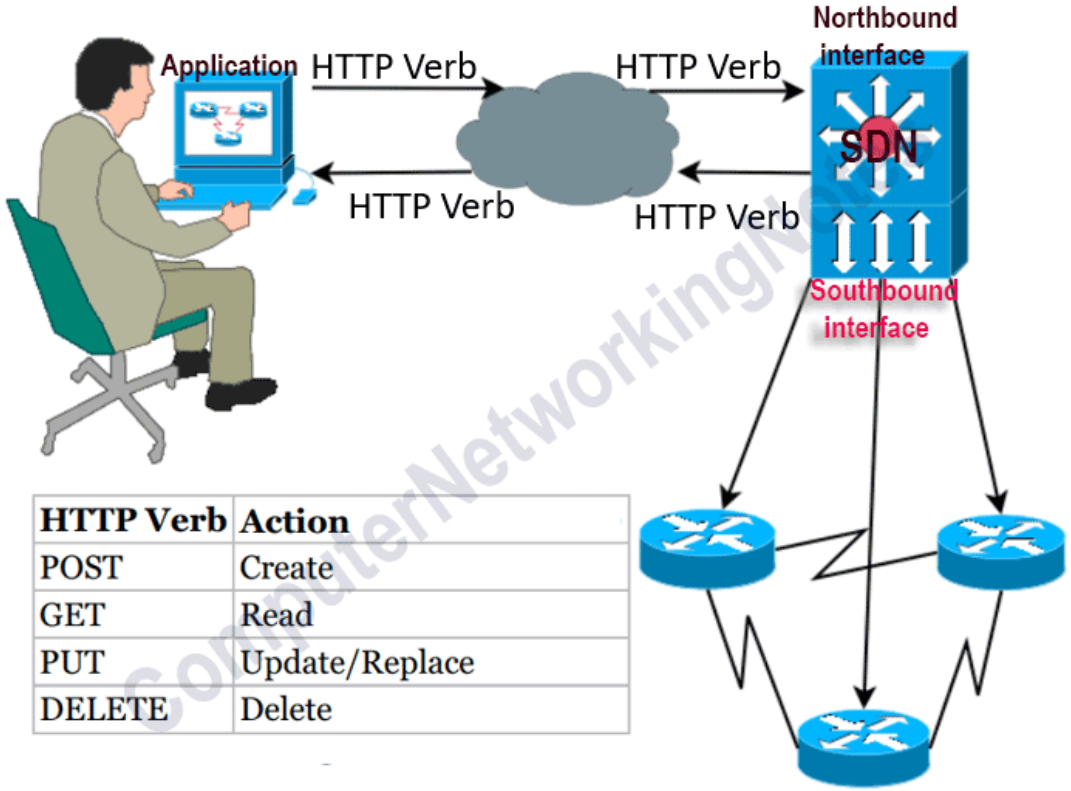


HTTP Verb	Action
POST	Create
GET	Read
PUT	Update/Replace
DELETE	Delete

SDN (Software Defined Network)

SDN Architecture : Northbound APIs

- The administrator sends HTTP GET requests to the SDN controller.
- In SDN implementation, the application uses HTTP GET requests to identify an object on the SDN controller, typically a data structure that the application needs to learn and then process.
- For example, it might identify an object that lists physical interfaces on a specific network device along with the status of each.



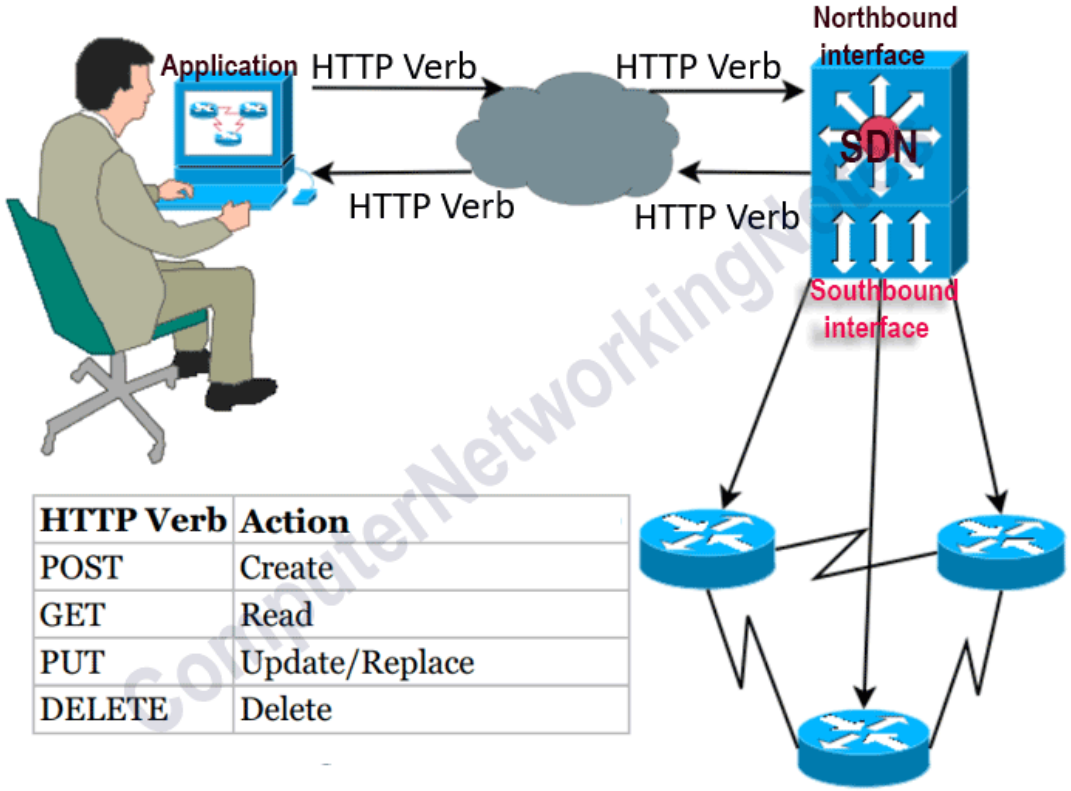
HTTP Verb	Action
POST	Create
GET	Read
PUT	Update/Replace
DELETE	Delete

- The controller sends back an HTTP GET response message with the object. The response message includes the information the application needs.
- The main difference between a browser and an SDN application's HTTP requests is the format of the data they retrieve. Browsers receive web pages, while SDN applications receive structured data.

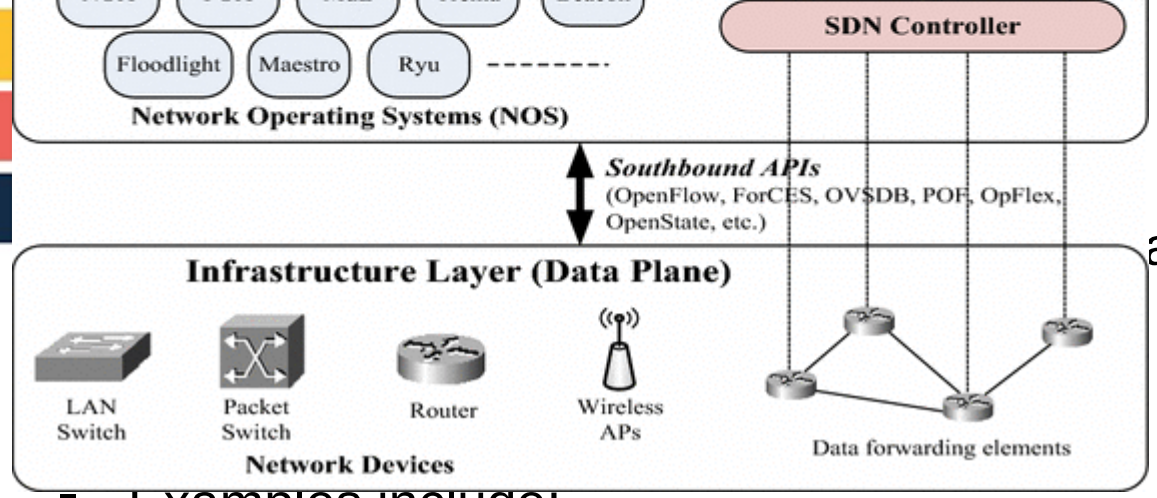
SDN (Software Defined Network)

SDN Architecture : Network OS

- The software platform that runs on the SDN controller.
- **Role:**
  - Provides the **abstraction layer** of the underlying network devices.
  - Offers services like topology discovery, statistics collection, and device management.
  - Acts as the **middleware** between hardware (data plane) and applications (application plane).
- **Examples:**
  - OpenDaylight's Karaf container, ONOS (Open Network Operating System).



HTTP Verb	Action
POST	Create
GET	Read
PUT	Update/Replace
DELETE	Delete

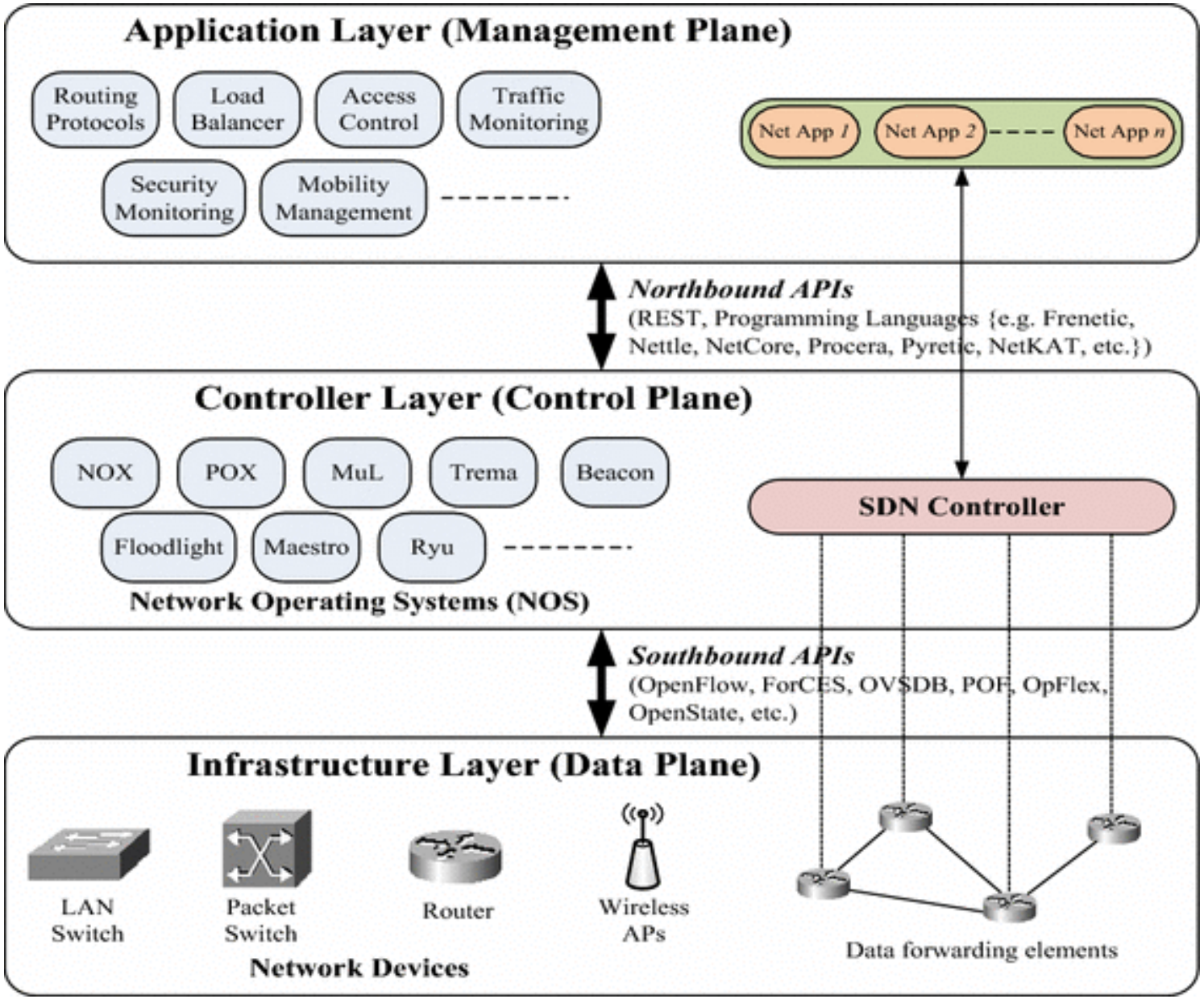


applications run.

program the network behavior.

- Examples include:
  - **Load Balancing Apps** (distribute traffic efficiently)
  - **Firewall Apps** (security enforcement)
  - **Traffic Engineering Apps** (optimize network performance)
  - **Monitoring Apps** (collect stats and visualize network health)

SDN (Software Defined Network)



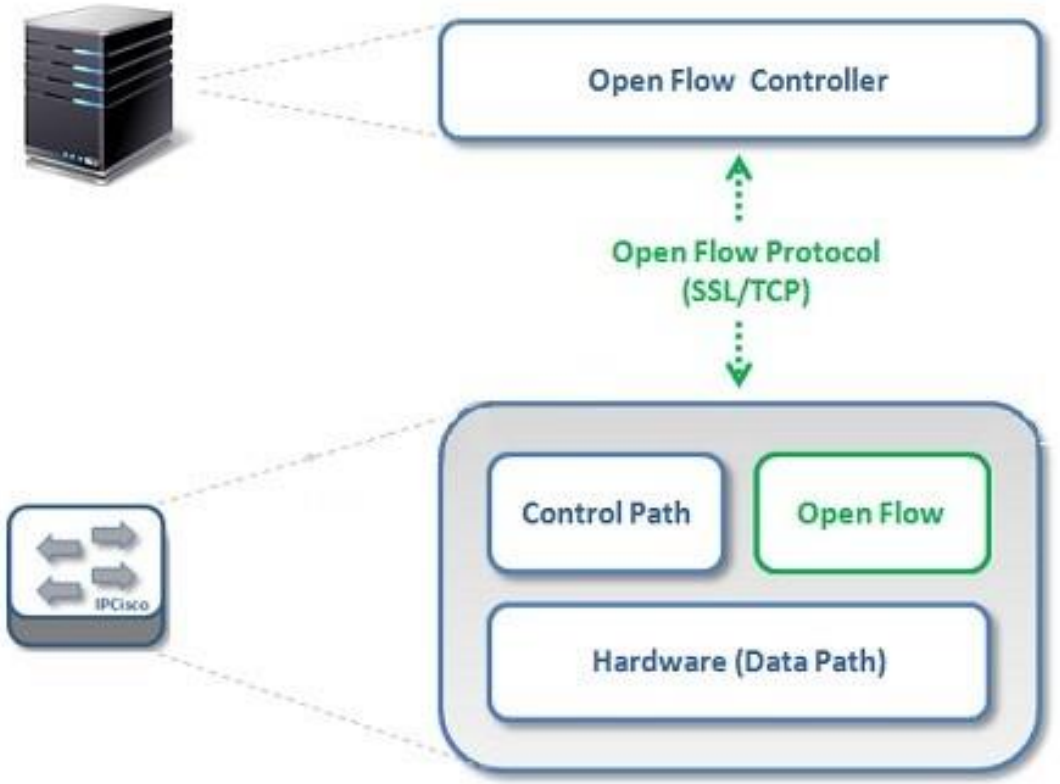
SDN Terminology

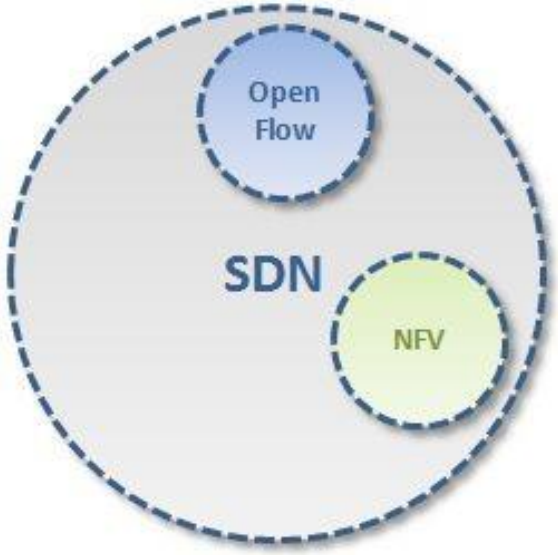
SDN Protocol Terms

- **NFV : Network Functions Virtualization.** Virtualization of the different skilled physical network devices with their virtual counterparts.
- **Flow :** Sequence of packet between the source and the destination.
- **Flow Rules :** Actions set for the flow.
- **The Flow Table :** Tables on the switch that handles the traffic flow.
- **Open Flow :** Southbound Interface **API**, that provide the communication of SDN Controller and SDN Data Plane devices.

Open Flow Overview

- Open Flow is a Standard based Layer 2 Communication protocol used between Controller and Switch in SDN.
- It allows to access to the Forwarding Plane of the network devices.
- Open Flow was standardized by a non-profit consortium, Open Networking Foundation (ONF) that has many network vendors as a member.
- This protocol had firstly developed for testing the new protocols in campus networks. But later, it has evolved to used in SDN.
- Generally, Open Flow is mixed with SDN. But this is not true, SDN is not Open Flow. Open Flow is a subset of SDN. Let's see the difference of these two terms.





Open Flow Overview

Key Components of OpenFlow

- **Flow Table** (inside the switch)
  - A set of **flow entries** that define what to do with packets.
  - Each entry has:
    - **Match fields** – like source IP, destination IP, TCP port, VLAN ID, etc.
    - **Counters** – how many packets/bytes matched.
    - **Actions** – forward to port, drop, modify header, etc.
- **SDN Controller**
  - A centralized software that decides network policies.
  - Installs, updates, or removes flow entries on switches.
- **OpenFlow Channel**
  - A secure communication path (often over TCP/TLS) between controller and switches.

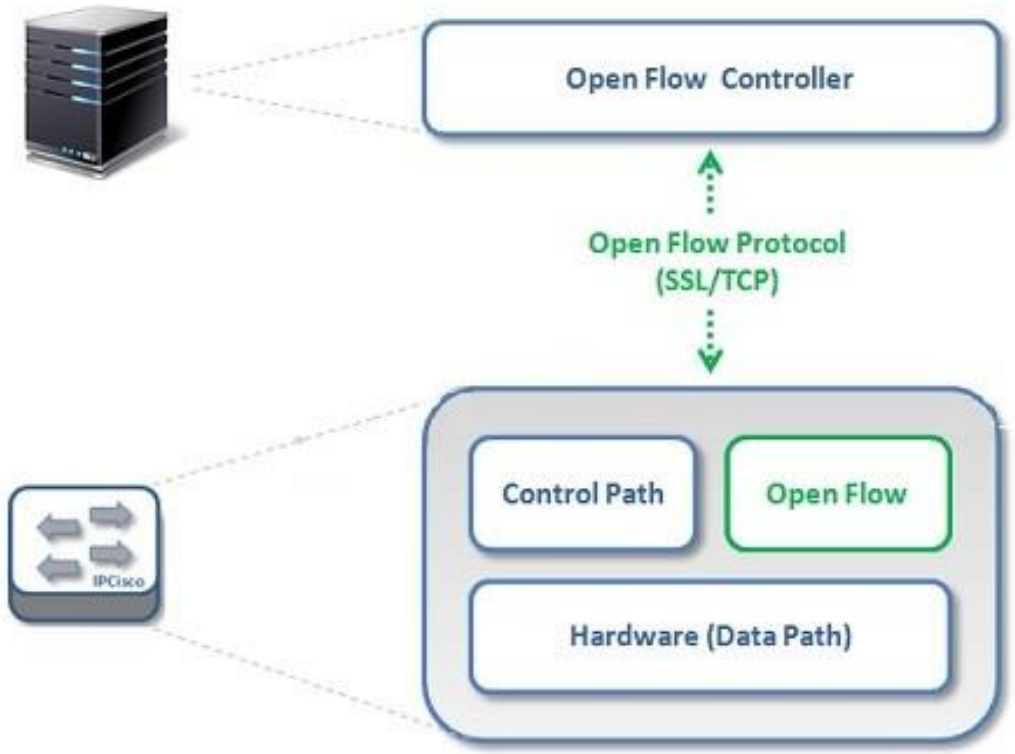
Open Flow Overview

How OpenFlow Works (Step-by-Step)

- **A packet arrives** at a switch.
- The switch **checks its flow table**:
  - If a matching rule is found → take the specified action (forward/drop/modify).
  - If **no match** → the switch sends a **Packet-In message** to the controller.
- **Controller decides** what to do and sends back a **Flow-Mod message**:
  - Installs a new rule in the switch's flow table.
  - Tells the switch what to do with that packet (and future packets in the same flow).
- **Switch applies the action** (forwards packet, drops it, etc.).

Open Flow Overview

- Open Flow is the protocol used in SDN, that is used to communicate forwarding plane and control plane of the network. In other words, the communication between Controller and the Network Devices are done with Open Flow.
- It allows operations, manipulations on Network Devices over Open Flow Interface.
- **Open Flow** has some common roles. A nice summary of these roles are given below:
  - Open Flow allows separation of control and data planes in the network.
  - It provides centralization of the control.
  - Open Flow uses Flow based control mechanism.
  - Takes advantage routing tables in Ethernet switches and routers.



NFV (Network Function Virtualization)

BACKGROUND

Traditional physical network hardware has always been difficult to change and upgrade. Introducing a software patch or rolling out a new service on a physical network can take months to complete. This is both time consuming and costly.

As a result, Over-The-Top, OTT operators have been able to gain a significant market foothold as they only utilize the network and do not inherit its problems. The OTT operators can launch a software-based internet platform that can be deployed almost immediately and run over the top of existing hardware.

NFV, network functions virtualization started in with mobile telecommunications networks with the promise of making networks more flexible and far easier to upgrade and change.

The concept of network functions virtualization was first introduced this concept in November 2012 as part of the ETSI ISG to provide hardware-related CAPEX and OPEX reductions.

As more developments have been made with NFV, service agility has become one of the main drivers for the development of network functions virtualization

NFV (Network Function Virtualization)

Network Functions Virtualization, NFV is an approach to telecommunications networking where the network entities that traditionally used dedicated hardware items are now replaced with computers on which software runs to provide the same functionality.

By running a network based around NFV, Network Functions Virtualization techniques, it is easier to expand and modify the network, and it is able to provide considerably more flexibility as well as being able to standardize on much of the hardware as it consists of additional computing power. In this way costs can be considerably reduced.

NFV (Network Function Virtualization)

NFV, network functions virtualization is a concept that virtualizes major elements of a network. In this way, rather than having a dedicated item of hardware to provide a given function, software running on a computer/server is used.

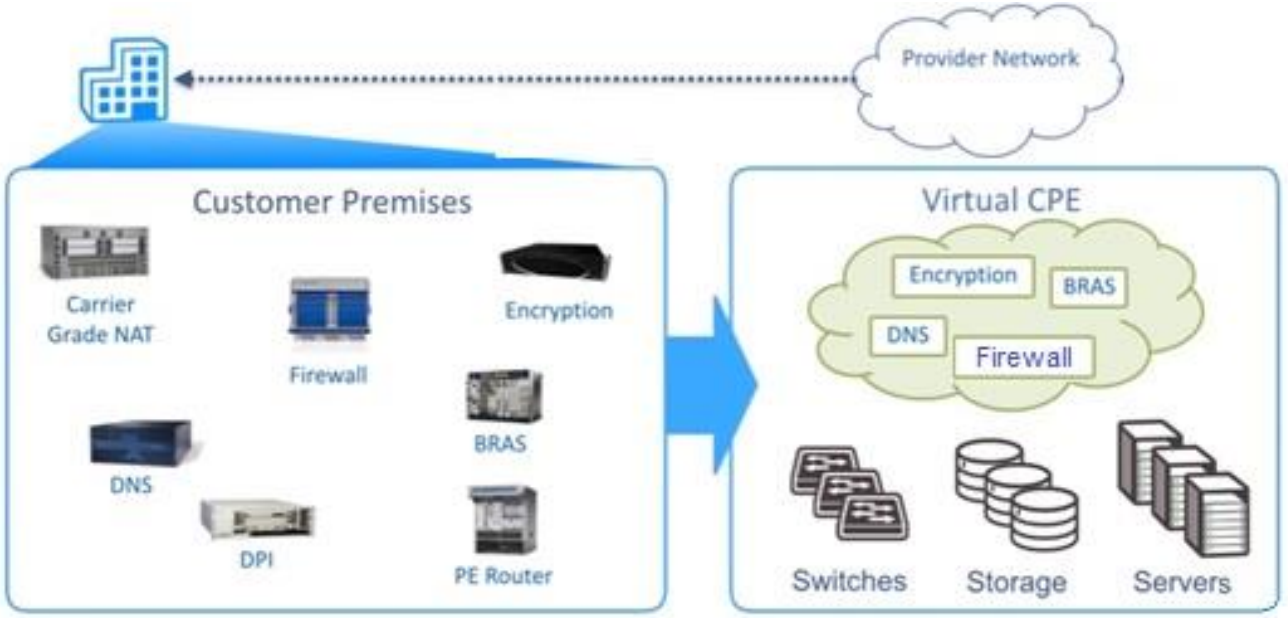
In this way entire classes of network node functions can be set up as building blocks that can be connected to create overall telecommunications networks.

NFV utilizes traditional server virtualization, but extends the concept significantly. In this way one or more virtual machines running different software and providing different processes, on top of industry standard high volume servers, are able to provide the functions of switches and storage, or even cloud computing infrastructure, instead of having custom hardware appliances for each network function.

Examples of the virtualized functions that can be provided include: virtualized load balancers, firewalls, intrusion detection devices, WAN accelerators, routers, access control and billing.

NFV (Network Function Virtualization)

The role of NFV is to relocate network functions from dedicated appliances to generic servers. NFV allows network operators to implement network policy without worrying about where to place the functions in the network and how to route traffic through these functions.



NFV moves all the functions as listed below in a common x86 architecture using virtualization.
• Router • Firewall • Web server • Load Balancer • Switch • Media Server

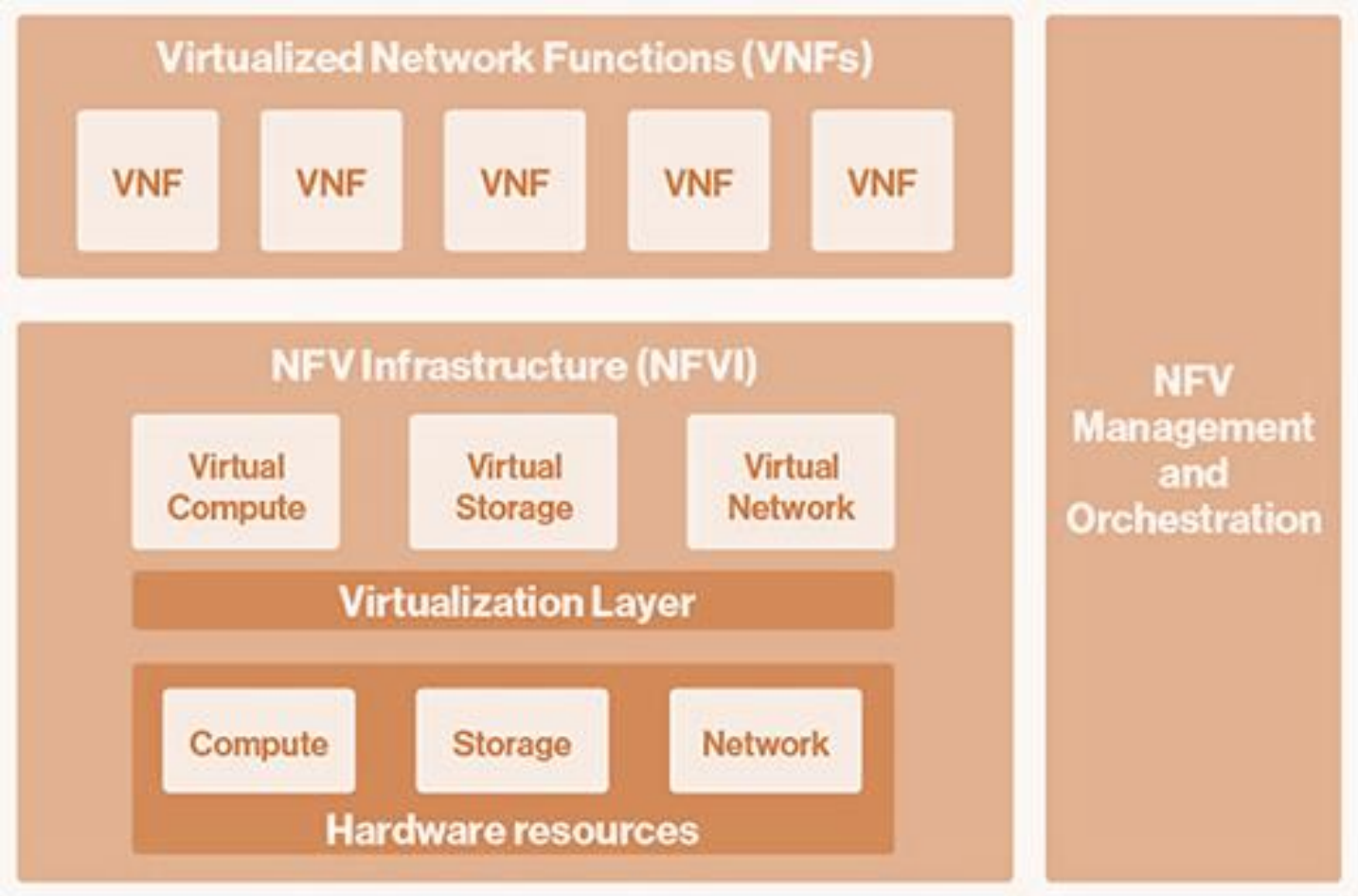
NFV helps in achieving following benefits for the operators.
• Standard hardware • Less complex • very flexible • reduced power • lower CapEx • lower OpEx • Test new applications • Low risk • Reduced TTM • Open market to software suppliers

NFV Framework

NFV FRAMEWORK

As with any system, a network using NFV techniques can be broken down into a number of elements. Those for Network Functions Virtualization are:

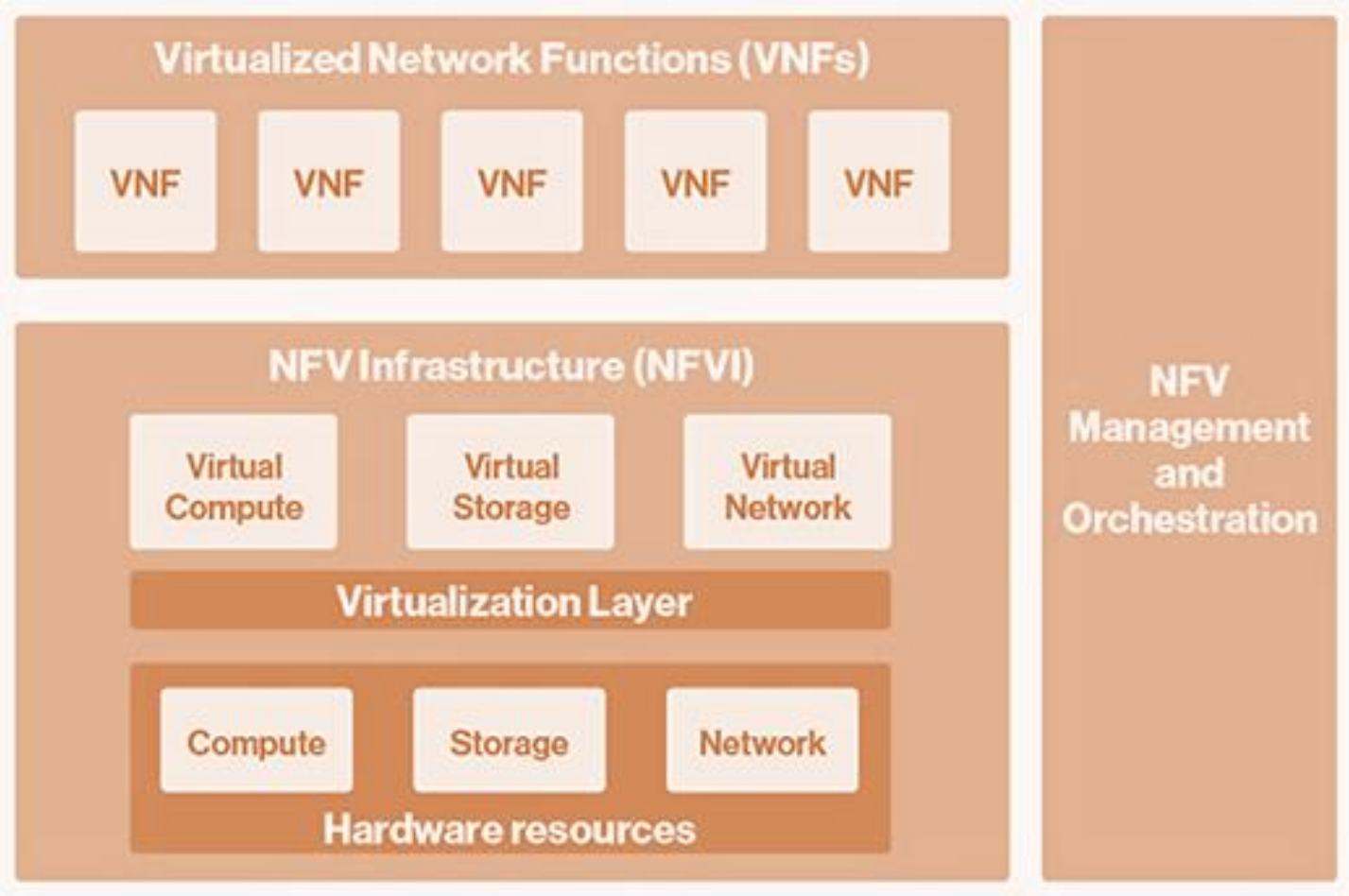
Virtualized Network Functions, VNF: The virtualized network functions comprise the software used to create the various network functions in their virtualized format. These are then deployed onto the hardware, i.e. the Network Function Virtualization Infrastructure.



NFV Framework

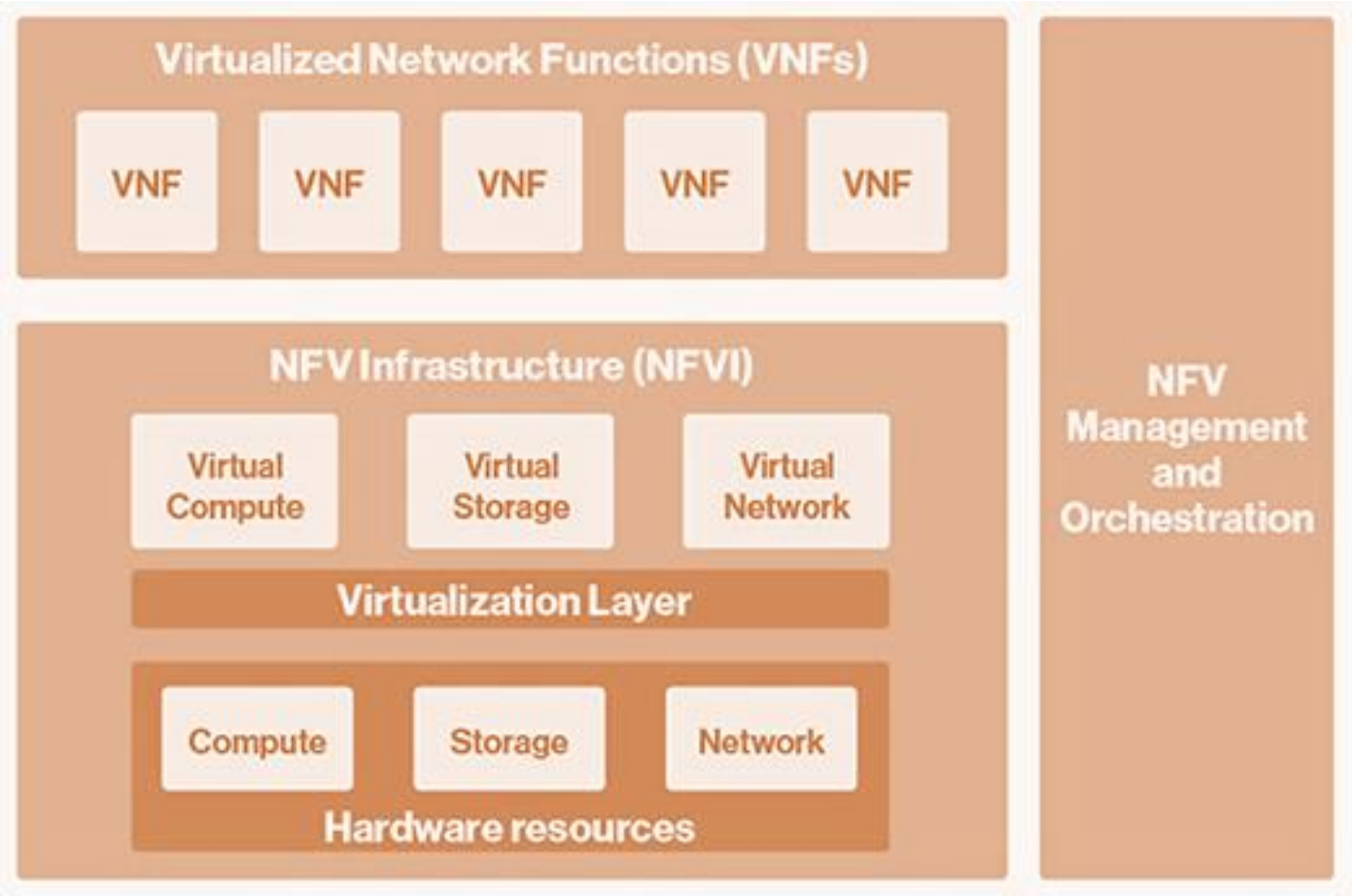
Network function virtualization infrastructure, NFVI: The NFVI consists of all the hardware and software components which are contained within the environment in which VNFs are deployed.

One of the advantages of NFV is that the NFV-Infrastructure, NFVI can be located across several physical locations, allowing operators to typically place their centers at the most convenient locations. The network providing connectivity between these locations is part of the NFV-Infrastructure.



NFV Framework

Network functions virtualization management and orchestration architectural framework, NFV-MANO Architectural Framework: NFV-MANO consists of the various functional blocks in whatever form that enables the exchange information, manipulation and storage needed manage and run the NFVI and VNFs, the network to operate correctly and provide significant improvements in efficiency and performance over other forms of network.



The NFVI and the NFV-MANO areas of the network are built within the overall NFV platform. This platform implements carrier-grade features used to manage and monitor the various components, recover from failures and provide effective security. These functions are all needed to run a public carrier network.

VNF (Virtualized Network Functions)

- Virtualized network functions or VNFs are the software realizations of the various network functions that can be deployed on a Network Functions Virtualization Infrastructure and needed to enable the network to operate.

- In this way, a virtual network function or VNF handles a specific network function that run on one or more virtual machines on top of the hardware networking infrastructure.

- The individual virtual network functions, VNFs, can be considered to be building blocks and they can be connected or combined together, providing all the capabilities required to provide a complete networking communication service.

VNF (Virtualized Network Functions)

Examples of various virtual network functions can be found within all areas of a telecommunications network and they can include:

- Switching: BNG, CG-NAT, routers.
- Tunneling gateway elements: IPSec/SSL VPN gateways.
- Traffic analysis: DPI, QoE measurement.
- Signaling: SBCs, IMS.
- Application-level optimization: CDNs, load Balancers.
- Home routers and set top boxes.
- Mobile network nodes: HLR/HSS, MME, SGSN, GGSN/PDN-GW, RNC.
- Network-wide functions: AAA servers policy control, charging platforms.
- Security functions: firewalls, intrusion detection systems, virus scanners, spam protection.

In this way, it can be seen that a huge number of VNFs can be run on a network using Network Functions Virtualization.

SDN Vs VNF

Features	SDN	NFV
Focus or major role	SDN focuses on data center.	NFV focuses on service providers or operators.
Strategy	It splits the control and data forwarding planes.	It replaces hardware network devices with software.
protocol	Uses OpenFlow	Not finalized yet, does support OpenFlow
Where the applications will run?	Applications run on industry standard servers or switches	Applications run on industry standard servers
Prime initiative supporters	Vendors of enterprise networking software and hardware.	Telecom service providers or operators.
Business initiator	Corporate IT	Service provider
Customer benefit or end user benefit	Drives down complexity and cost and increases agility.	Drives down complexity and cost and increases agility.
Initial applications	Cloud orchestration and networking	Routers, Firewalls, Gateways, CDN, WAN Accelerators, SLA Assurance
Formalization body	Open Networking Foundation (ONF)	ETSI NFV Working Group

MODULE 3
IOT DESIGN METHODOLOGY

- What is an IoT Device-Basic?
- Building blocks of an IoT Device
- Exemplary Device:
  - Raspberry Pi, About the Board, Linux on Raspberry Pi, Raspberry Pi Interfaces – Serial, SPI, I2C, Programming, Raspberry Pi with Python-Controlling LED with Raspberry Pi ,
  - Interfacing an LED and Switch with Raspberry Pi,
  - Interfacing a Light Sensor (LDR) with Raspberry Pi ,
- Other IoT Devices- pcDuino, Beagle Bone Black, Cubieboard

What is an IoT Device?

- An **IoT device** is any object or "Thing" that:
  - Has a unique identifier.
  - Can send and/or receive data (including user data) over a network.
  - Is connected to the Internet and communicates with other systems.
- **Examples of IoT Devices:**
  - Smartphones, smart TVs, refrigerators, cars, wearables, computers.
- These devices share information about themselves or their environment using sensors or actuators.

Functions of IoT Devices:

- **Data Sensing**: Gather environmental or system data.
- **Communication**: Transmit and receive data via the internet/network.
- **Remote Actuation**: Trigger actions remotely (e.g., turning on lights).
- **Monitoring and Control**: Used for automation and user control.

Design Methodology in the Context of IoT Systems

Category	Example IoT Device	Functionality
Smart Home	Smart Thermostat (e.g., Nest)	Adjusts room temperature based on user habits and environmental data
	Smart Lock	Remote door access control and monitoring
	Smart Light Bulbs (e.g., Philips Hue)	Remote-controlled lighting with scheduling and color customization
Healthcare	Smart Glucose Monitor	Tracks and uploads blood sugar data to cloud for medical review
	Smart ECG Monitor	Sends real-time ECG data to hospitals
	Smart Inhalers	Tracks usage and connects to patient mobile apps
Industrial IoT (IIoT)	Vibration Sensors in Machinery	Detects abnormal conditions for predictive maintenance
	Smart Power Meters	Monitors industrial power usage and provides analytics
Agriculture	Soil Moisture Sensor	Measures soil moisture levels and sends data to central control
	Automated Irrigation System	Controls water valves based on weather and moisture data
Transportation	GPS Trackers on Vehicles	Real-time tracking and route optimization
	Smart Parking Sensors	Detects empty parking spaces and guides drivers
Retail	Smart Shelves	Monitor product quantity and alert when restocking is needed
	Connected Point-of-Sale Devices	Enable inventory tracking and analytics
Environment Monitoring	Air Quality Monitoring Devices (e.g., CO_2, PM2.5 sensors)	Track pollution levels and upload data to cloud for analysis
	Smart Water Quality Sensors	Detect contaminants in water pipelines

Basic Building Blocks of an IoT Device

An IoT device is composed of several **functional modules**, each responsible for a specific task. These modules work together to enable sensing, communication, decision-making, and actuation.

❖ **Sensing**

- Sensors collect information from the environment.
- Can be on-board (integrated into the device) or attached externally.
- Examples: temperature, humidity, light intensity, etc.
- Data is sent to other devices or to cloud-based servers/storage.

❖ **Actuation**

- Responsible for performing actions based on received commands.
- Actuators can turn physical devices on/off, move parts, trigger alerts, etc.
- Example: A relay switch connected to an IoT device can turn on/off a fan or a light.

Basic Building Blocks of an IoT Device

❖ Communication

- Enables data transmission between the IoT device and:
  - Other devices
  - Cloud storage/servers
- Supports receiving control commands from remote applications.
- May include Wi-Fi, Bluetooth, Zigbee, LTE, etc.

❖ Analysis & Processing

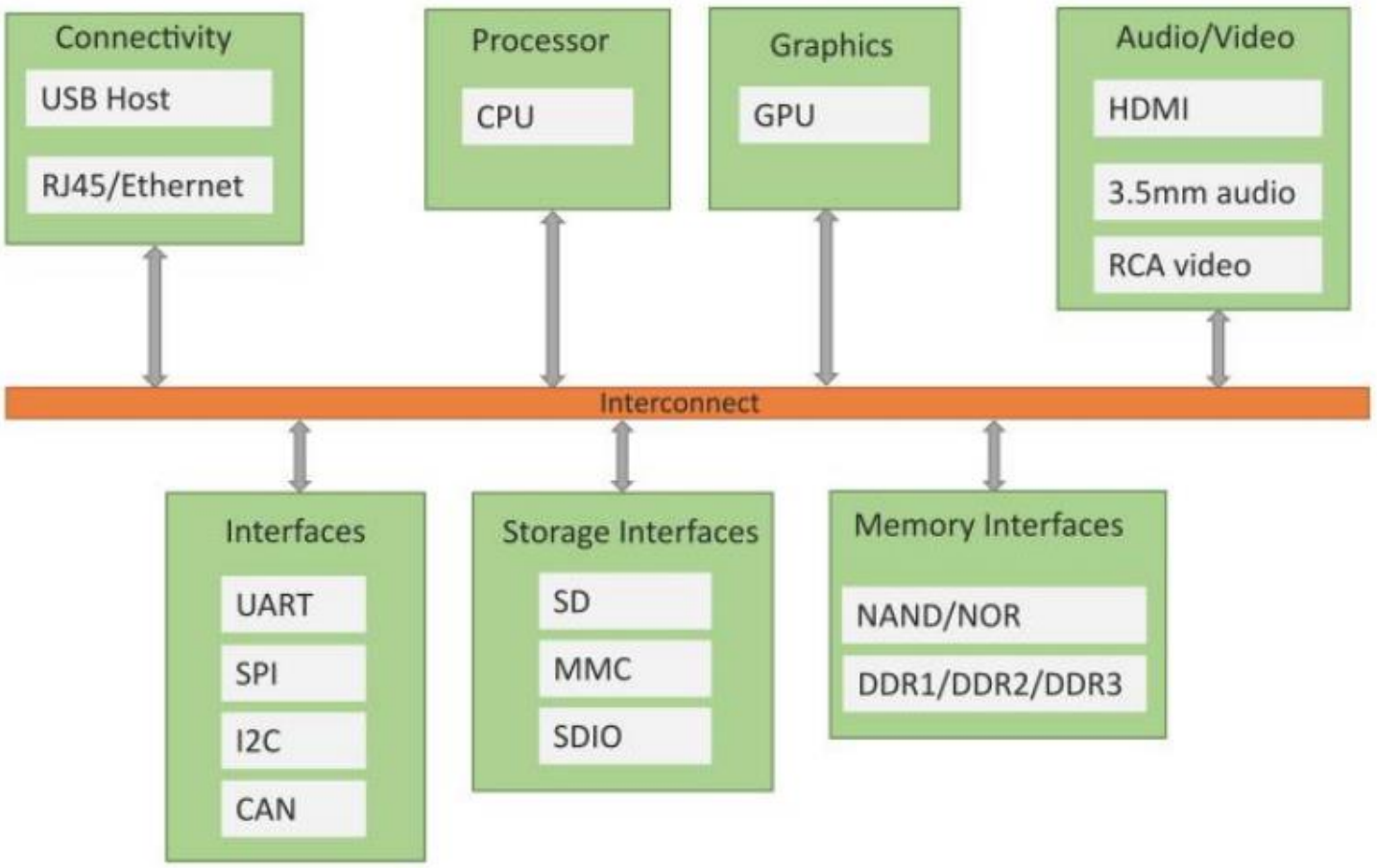
- Processes the collected data to extract useful insights.
- Involves filtering, aggregation, decision-making, etc.
- May be done locally (on the device) or remotely (in the cloud).

Basic Building Blocks of an IoT Device

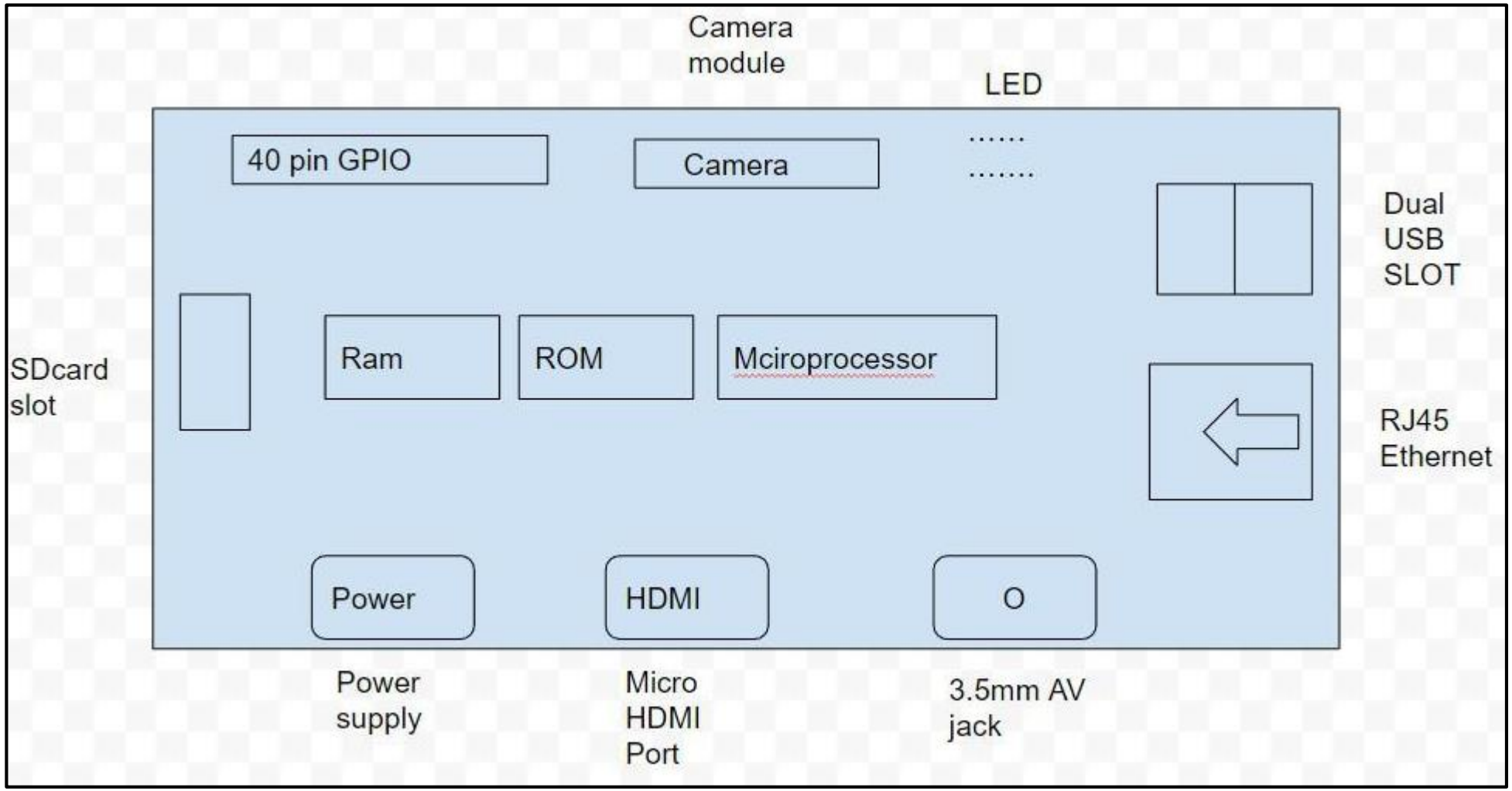
Example IoT Device Platform:

Raspberry Pi

- A widely used single-board computer (SBC).
- Features: CPU, GPU, RAM, storage, multiple interfaces.
- Chosen for its:
  - Low cost
  - Accessibility
  - Strong online community & documentation



The Raspberry Pi Board Block Diagram



The Raspberry Pi Board

- **Processor**
  - Raspberry Pi is built on an ARM processor.
  - Model B (Revision 2) includes a 700 MHz ARM1176JZF-S CPU and 512 MB SDRAM.
- **USB Ports**
  - Two USB 2.0 ports available.
  - Each port provides 100 mA.
  - Devices needing more power require an external powered USB hub.
- **Ethernet Port**
  - Standard RJ45 Ethernet connection.
  - Supports wired internet.
  - USB Wi-Fi adapter can also be used for wireless connectivity.

The Raspberry Pi Board

- **HDMI Output**
  - HDMI port provides audio + video output.
  - Can connect to TVs/monitors with HDMI.
  - For DVI monitors, use HDMI-to-DVI adapter.
- **Composite Video Output**
  - RCA jack provides PAL/NTSC composite video.
  - Suitable for older TVs without HDMI input.
- **Audio Output**
  - 3.5 mm audio jack for analog stereo audio.
  - Works alongside the RCA video output.
- **GPIO Pins**
  - Multiple General Purpose I/O pins.
  - Supports I2C, SPI, UART communication.
  - Used to interface sensors, actuators, and external devices.

The Raspberry Pi Board

- **Display Interface (DSI)**
  - DSI port connects LCD display panels.
- **Camera Interface (CSI)**
  - CSI port connects the Raspberry Pi Camera Module.
- **Status LEDs**
  - Raspberry Pi includes five status LEDs showing power, activity, and network status.
- **SD Card Slot**
  - Used for OS installation and storage.
  - Raspberry Pi requires SD card with Linux OS (NOOBS/Raspbian).
- **Power Input**
  - Powered via micro-USB connector.

Raspberry Pi Supported Operating Systems

Linux on Raspberry Pi

Raspberry Pi supports multiple Linux-based operating systems, each designed for different uses and performance requirements.

1. Raspbian

- A Debian Wheezy–based Linux distribution optimized specifically for Raspberry Pi.
- Official and recommended OS for Raspberry Pi.
- Lightweight and stable, suitable for beginners and general applications.

2. Arch

- A lightweight Arch Linux port for ARM devices.
- Designed for users who prefer a minimal, customizable Linux environment.

3. Pidora

- A Fedora Linux distribution customized for Raspberry Pi.
- Provides a user-friendly Fedora experience on ARM-based systems.

Raspberry Pi Supported Operating Systems

4. RaspBMC

- Linux-based XBMC media center distribution for Raspberry Pi.
- Designed for multimedia applications such as video playback and home media servers.

5. OpenELEC

- A fast and user-friendly XBMC/Kodi media center Linux distribution.
- Extremely lightweight and optimized for media playback.

6. RISC OS

- A very fast, compact, and efficient operating system.
- Unlike Linux-based OSes, RISC OS is designed with simplicity and performance in mind.

Raspberry Pi Interfaces

Raspberry Pi includes multiple hardware interfaces—Serial, SPI, and I^2C—which enable communication with sensors, modules, and peripheral devices.

1. Serial Interface (UART)

The Raspberry Pi has a serial communication interface commonly referred to as UART (Universal Asynchronous Receiver/Transmitter).

- It uses two main pins:
  - **Rx (Receive)** – receives serial data from external devices.
  - **Tx (Transmit)** – sends serial data to external devices.
- UART Used for:
  - Debugging and terminal communication.
  - Interfacing GPS modules, GSM modules, Bluetooth modules, etc.

Communication is asynchronous, meaning no clock wire is used.

Raspberry Pi Interfaces

2. SPI Interface (Serial Peripheral Interface)

SPI is a synchronous serial communication protocol used for high-speed data transfer between Raspberry Pi and one or more peripheral devices.

❖ **Key Features of SPI:**

- Works on master–slave architecture.
- Raspberry Pi functions as the Master.
- Peripherals (e.g., ADCs, sensors, displays) act as Slaves.
- Supports high-speed, full-duplex communication.

❖ **SPI Pins on Raspberry Pi:**

There are five important pins for SPI communication:

- **MISO (Master In Slave Out):** Used to send data from the slave to the master.
- **MOSI (Master Out Slave In):** Used to send data from the master to the slave.
- **SCK (Serial Clock):** Clock signal generated by the master. Synchronizes data transfer between master and slave.

Raspberry Pi Interfaces

- **CE0 (Chip Enable 0):** Also called CS0 (Chip Select 0). Active low signal used to enable/activate the first slave device.
- **CE1 (Chip Enable 1):** Also called CS1 (Chip Select 1). Used to enable/activate the second slave device.

❖ **How SPI Works:**

- Master selects the slave using CE0/CE1.
- Clock (SCK) is sent from master to slave.
- Data is exchanged:
  - Master → Slave using MOSI.
  - Slave → Master using MISO.

❖ SPI supports fast data transfer, making it suitable for displays (OLED, LCD), high-speed ADCs, temperature sensors, SD card modules, etc.

Raspberry Pi Interfaces

3. I^2C Interface (Inter-Integrated Circuit)

I^2C is a **two-wire**, synchronous serial communication protocol designed to connect multiple low-speed peripherals using minimal pin connections.

❖ **I^2C Pins on Raspberry Pi:**

- **SDA (Serial Data Line)** – used for transferring data.
- **SCL (Serial Clock Line)** – used to synchronize data transfer between devices.

❖ **Key Characteristics:**

- Supports multiple devices on the same bus (multi-slave communication).
- Only two wires are used regardless of number of connected devices.
- Devices have unique 7-bit or 10-bit addresses for identification.
- Suitable for:
  - Real-time clocks (RTC modules)
  - Digital sensors (temperature, pressure, environment sensors)

Raspberry Pi Interfaces

- I/O expanders
- Small EEPROMs
- Display modules

❖ **How I^2C Works:**

- Raspberry Pi acts as the Master.
- Communication is synchronized using SCL.
- Data is transferred through SDA, one bit at a time.
- Very useful for low-speed, short-distance communication.

Programming Raspberry Pi with Python

- Raspberry Pi runs on Linux and comes with Python installed by default, making it ideal for beginners and IoT developers. Using Python, you can control the Raspberry Pi's GPIO (General Purpose Input/Output) pins, which allow interaction with sensors, switches, and electronic components.

- Python programs on Raspberry Pi function just like on any regular computer, but the ability to use GPIO pins makes it uniquely suitable for embedded system applications and Internet of Things (IoT) projects.

- Using GPIO along with other interfaces such as SPI, I^2C, and Serial, Raspberry Pi can read sensor data, trigger devices, control motors, send information to servers, or perform automated actions like sending emails or activating a relay.

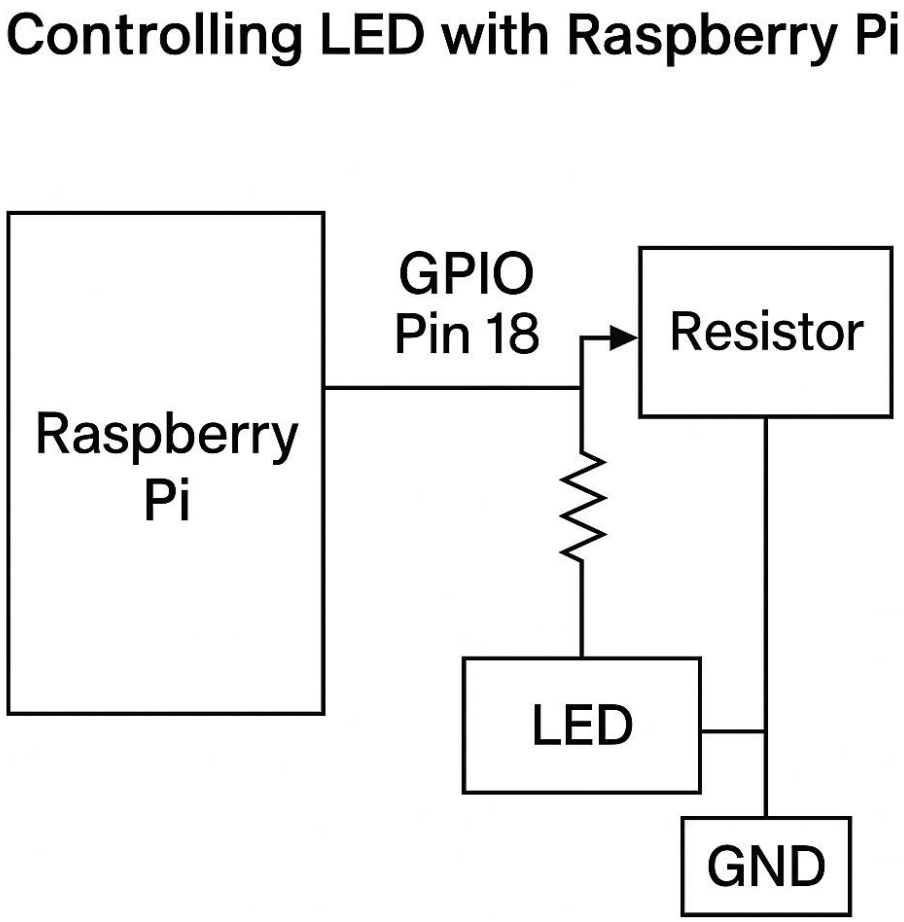
Programming Raspberry Pi with Python

1. Controlling an LED with Raspberry Pi

To understand GPIO usage, we start with a simple example—turning an LED ON and OFF using Python on Raspberry Pi.

Hardware Setup

- An LED is connected to **GPIO pin 18** on the Raspberry Pi.
- A resistor (typically 220Ω–330Ω) is used in series to protect the LED.
- The ground (GND) pin of Raspberry Pi is connected to the negative terminal of the LED.



Programming Raspberry Pi with Python

Controlling LED Using Raspberry Pi Console (Terminal)

You can switch the LED ON/OFF directly from the terminal using Linux file operations on GPIO.

- **Step 1: Export the GPIO pin**

  echo 18 > /sys/class/gpio/export

- **Step 2: Set the pin direction**

  echo out > /sys/class/gpio/gpio18/direction

- **Step 3: Turn LED ON**

  echo 1 > /sys/class/gpio/gpio18/value

- **Step 4: Turn LED OFF**

  echo 0 > /sys/class/gpio/gpio18/value

This method demonstrates how hardware pins can be controlled using simple Linux commands.

Programming Raspberry Pi with Python

Python Program for Blinking an LED

Below is a Python script that blinks the LED connected to **GPIO18** every 1 second:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)          # Use BCM pin numbering
GPIO.setup(18, GPIO.OUT)        # Set pin 18 as output

while True:
    GPIO.output(18, True)       # LED ON
    time.sleep(1)               # Wait 1 second
    GPIO.output(18, False)      # LED OFF
    time.sleep(1)               # Wait 1 second
```

The LED continues to blink until the program is stopped manually.

Programming Raspberry Pi with Python

2. Interfacing an LED and Switch with Raspberry Pi

In this section, we explore a practical example of using Raspberry Pi's GPIO pins to interface both an LED (output device) and a switch (input device). The objective is to toggle the LED ON/OFF when the switch is pressed.

This example demonstrates:

- How to read digital input from a GPIO pin.
- How to control an output device (LED) based on the input.
- How to write simple IoT-style Python programs.

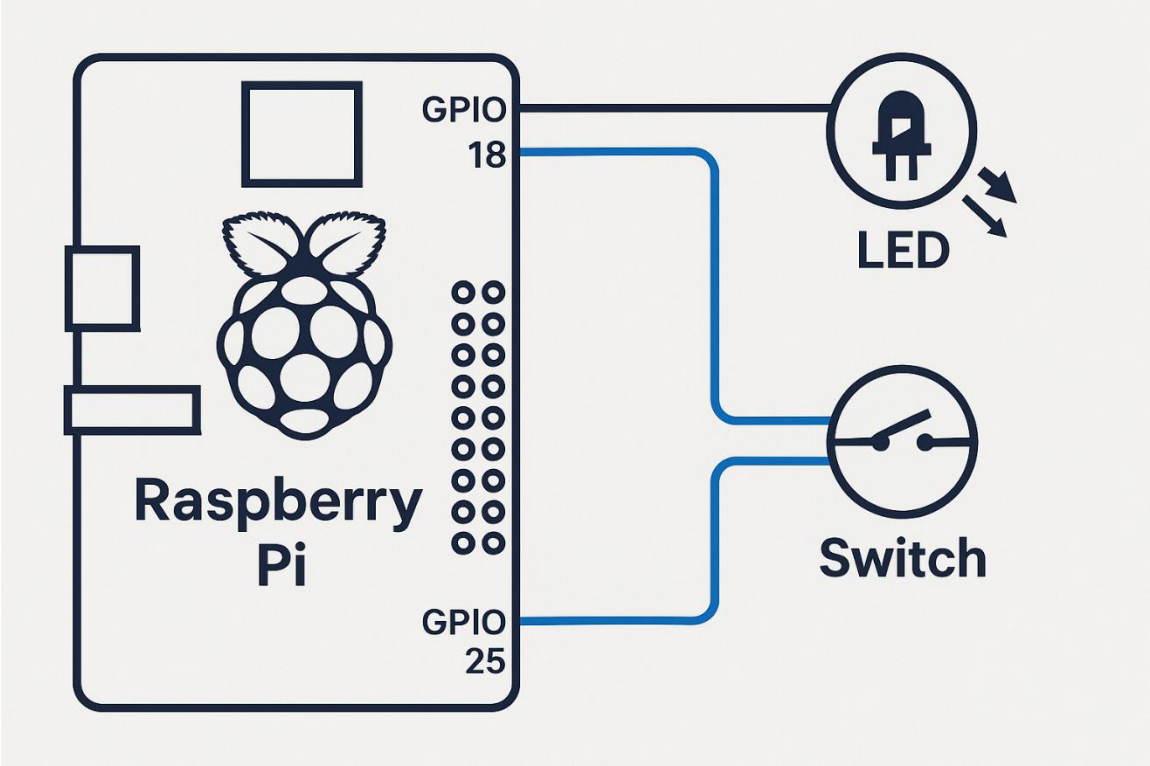
Programming Raspberry Pi with Python

Hardware Explanation

Connections

- LED is connected to GPIO pin 18 (configured as OUTPUT).
- Switch is connected to GPIO pin 25 (configured as INPUT).
- A resistor is used with the LED to prevent excessive current.

When the switch is pressed, Raspberry Pi detects a HIGH signal on GPIO pin 25. This triggers a function to toggle the LED state. Figure 7.10 (shown in your image) displays the wiring diagram on a breadboard and Raspberry Pi.



Programming Raspberry Pi with Python

Python Program: Controlling LED with a Switch

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)     # Use BCM numbering

# Configure switch pin as Input
GPIO.setup(25, GPIO.IN)

# Configure LED pin as Output
GPIO.setup(18, GPIO.OUT)

state = False              # Initial LED state (OFF)
```

Step 1: Importing Libraries

- RPi.GPIO is used to control GPIO pins.
- sleep() adds small delays to improve stability.

Step 2: GPIO Mode Selection

- BCM mode uses the processor pin numbers (not physical board pin numbers).

Step 3: Setting Up Pins

- GPIO 25 is for reading input from the switch.
- GPIO 18 controls the LED.

Step 4: LED State Variable

- This variable keeps track of whether LED is ON or OFF.

Programming Raspberry Pi with Python

Python Program: Controlling LED with a Switch

```
# Function to toggle LED state
def toggleLED(pin):
    global state
    state = not state      # Reverse the current state (ON → OFF or OFF → ON)
    GPIO.output(pin, state)

# Infinite loop to monitor switch input
try:
    while True:
        if GPIO.input(25) == True:    # If switch is pressed
            toggleLED(18)             # Change LED state
            sleep(0.1)                # Small delay to avoid bouncing
except KeyboardInterrupt:
    GPIO.cleanup()                  # Clean exit on Ctrl+C
```

Step 5: Toggle Function

- Each time the function is called, the LED switches to the opposite state.

Step 6: Infinite Loop

- The program checks the input pin continuously.
- When the switch is pressed, LED toggles.
- Delay avoids rapid toggling due to noise or mechanical switch bounce.

Step 7: Graceful Exit

Ensures the program exits safely if the user presses **Ctrl + C**.

Programming Raspberry Pi with Python

3. Interfacing a Light Sensor (LDR) with Raspberry Pi

A Light Dependent Resistor (LDR) is a sensor whose resistance varies based on the amount of light falling on it.

- **More light → Low resistance**
- **Less light → High resistance**

Using an LDR with Raspberry Pi allows us to detect ambient light levels and automatically switch ON/OFF an LED or light.

Hardware Setup Explanation

❖ **Connections**

- One side of the LDR is connected to 3.3V of Raspberry Pi.
- The other side of the LDR is connected to:
  - One leg of a 1 µF capacitor, and
  - A GPIO input pin (GPIO 18).
- The other leg of the capacitor is connected to GND.
- An LED is connected to GPIO pin 25 and controlled based on the LDR reading.

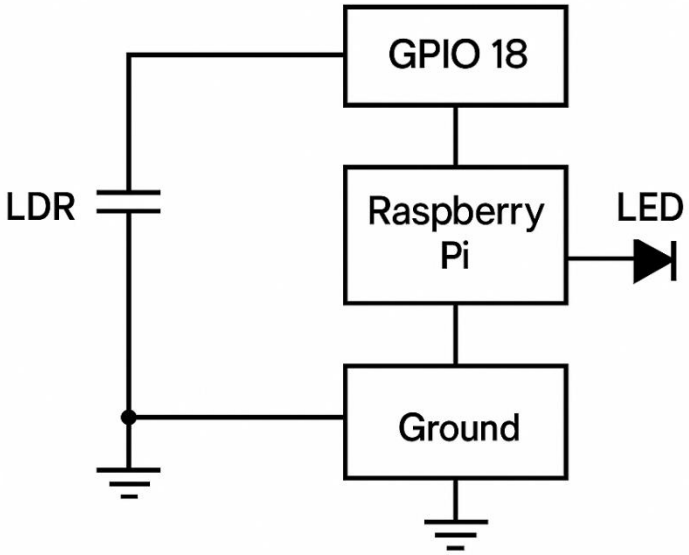
Programming Raspberry Pi with Python

- **Why capacitor is used?**
  - The Raspberry Pi cannot measure analog values directly.

    So, a resistor-capacitor (RC) timing method is used:
    - LDR + capacitor forms an RC circuit.
    - Time taken for the capacitor to charge is inversely proportional to light.
    - Raspberry Pi measures this charging time using digital input polling.

    Thus:
    - **Bright light → LDR resistance low → capacitor charges faster → small count**
    - **Low light → LDR resistance high → capacitor charges slowly → large count**

Interfacing a Light Sensor (LDR) with Raspberry Pi



Programming Raspberry Pi with Python

❖ How the Python Program Works

The Python program repeatedly:

- Discharges the capacitor.
- Measures time taken for the capacitor to charge to HIGH.
- Compares this time with a threshold.
- Turns the LED ON/OFF based on the reading.

```
import RPi.GPIO as GPIO
import time


GPIO.setmode(GPIO.BCM)
ldr_threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25
```

Step 1: Initialize Parameters

- ldr_threshold is the value above/below which light is considered low or high.
- LDR_PIN is used to read light value.
- LIGHT_PIN controls the LED.

Programming Raspberry Pi with Python

```
def readLDR(PIN):
    reading=0
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
```

```
    while(GPIO.input(PIN)==False):
        reading=reading+1
    return reading
```

Step 2: Discharge the capacitor
This forces the capacitor to 0V (discharged).

Step 3: Switch to input mode
Now Raspberry Pi waits for capacitor to charge.

Step 4: Count the time until capacitor charges
- The loop continues until the capacitor voltage reaches a HIGH level.
- More light → fewer counts
- Less light → more counts

Programming Raspberry Pi with Python

```
def switchOnlight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, True)

def switchOfflight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)

while True:
    ldr_reading = readLDR(LDR_PIN)
    if ldr_reading < ldr_threshold:
        switchOnlight(LIGHT_PIN)
    else:
        switchOfflight(LIGHT_PIN)
    time.sleep(1)
```

Step 5: Functions to Switch Light ON/OFF

Step 6: Logic

- If **LDR reading < threshold** → environment is **bright** → LED ON.
- If **LDR reading > threshold** → environment is **dark** → LED OFF.

This makes the LED work like an **automatic night lamp**.

Other IoT Devices

Raspberry Pi is one of the most popular single-board mini-computers, but there are several alternatives used for IoT applications. These boards also provide GPIO, networking, multimedia interfaces, and support for various operating systems.

Examples of such devices:

- **pcDuino**,
- **BeagleBone Black**, and
- **Cubieboard**.

Other IoT Devices

1. pcDuino

- pcDuino is an Arduino-pin-compatible single-board mini-computer designed for high performance and cost-effectiveness.
- It is powered by a 1 GHz ARM Cortex-A8 processor, enabling it to run full PC-class operating systems such as:
  - **Ubuntu Linux**
  - **Android ICS**
- **Key Features**

- Has HDMI interface for video output, similar to Raspberry Pi.
- Supports multiple programming languages:
  - **C / C++** (using GNU toolchain)
  - **Java** (using Android SDK environment)
  - **Python**
- Provides Arduino-style headers, making it easy to interface with sensors and actuators.
- Suitable for applications where both Arduino compatibility and Linux-level processing are required.

Other IoT Devices

2. BeagleBone Black

BeagleBone Black is another popular single-board computer and is considered a more powerful alternative to Raspberry Pi.

- **Processor & Performance**
  - Powered by a 1 GHz ARM Cortex-A8 processor (similar to pcDuino)
  - Offers higher processing capabilities and more real-time IO features.
- **Operating System Support**
  - Supports both Linux and Android operating systems.
  - Provides easier real-time processing because of its PRU (Programmable Real-Time Units).
- **Key Features**
  - HDMI video/audio output
  - USB and Ethernet ports
  - Extensive GPIO capability
  - Ideal for robotics, automation, and industrial IoT applications

BeagleBone Black is chosen when low-level hardware control and high-speed, deterministic I/O are required.

Other IoT Devices

3. Cubieboard

Cubieboard is a more powerful and versatile IoT-oriented mini-computer compared to traditional boards.

- **Processor**
  - Powered by a dual-core ARM Cortex-A7 processor, providing greater performance.
- **Connectivity and I/O**
  - Cubieboard includes a wide variety of interfaces: USB ports, HDMI, IR (Infrared), Serial port, Ethernet, SATA (for external hard drives), 96-pin extended interface
- **Storage**
  - Provides SATA support, enabling high-speed storage (useful for servers and multimedia systems).
- **Operating Systems**
  - Capable of running: Linux distributions, Android
- **Use Cases**
  - Home media centers
  - Network storage systems
  - IoT gateways
  - Embedded computing projects requiring more storage and processing power

MODULE 3
IOT DESIGN METHODOLOGY

- Purpose & Requirements Specification,
- Process Specification,
- Domain Model Specification,
- Information Model Specification,
- Service Specifications,
- IoT Level Specification,
- Functional View Specification,
- Operational View Specification,
- Device & Component Integration,
- Application Development,
- Case Study on IoT System for Weather Monitoring,
- Motivation for Using Python

What is IoT Design Methodology?

- Design Methodology is a structured, systematic approach used to plan, create, implement, test, and evaluate a system, product, or process.

- It consists of a series of well-defined steps or phases that guide designers in making informed decisions to meet specific goals and requirements.

- "A design methodology provides a structured way to develop complex systems like IoT, ensuring clarity, reusability, flexibility, and reduced development and maintenance effort.“

- Key Characteristics of a Design Methodology
  - Follows a step-by-step process
  - Ensures consistency and repeatability
  - Facilitates modularity, interoperability, and scalability
  - Encourages evaluation and comparison of alternative designs
  - Promotes best practices and standardization

Design Methodology in the Context of IoT Systems

In IoT system design, a design methodology helps to:

- Manage the complexity of integrating hardware, software, cloud, and network components.
- Select and configure sensors, actuators, protocols, and platforms effectively.
- Avoid vendor or product lock-in by promoting technology-agnostic designs.
- Ensure the system is interoperable, maintainable, and scalable.
- Enable systematic evaluation and testing of alternatives.

- **Why Use a Design Methodology in IoT?**
  - Without a formal methodology:
    - Designs may become rigid, hard to upgrade or replace.
    - Integration of new components becomes tedious.
    - Testing and troubleshooting become time-consuming.

Introduction to IoT System Design

- IoT (Internet of Things) systems involve multiple layers and complex interactions between:
  - Smart devices
  - Networks
  - Web services
  - Application servers
  - Database systems
  - Analytics components
- Each IoT system level is:
  - Tailored for specific applications
  - Has unique components
  - Requires different deployment configurations

These levels dictate how IoT systems are built, deployed, and managed.

Challenges in IoT System Design

- **Complexity of Interactions**
  - IoT systems involve coordination between hardware and software, often across heterogeneous platforms.
- **Overwhelming Design Choices**
  - Designers face numerous alternatives in:
    - Devices
    - Networks
    - Protocols
    - Platforms
    - Services
- **Vendor/Product Lock-In**
  - Designs are often tied to specific products or vendors.
  - Leads to:
    - Lack of interoperability
    - Difficulty in upgrades
    - Complete re-design when replacing components

Need for a Generic Design Methodology

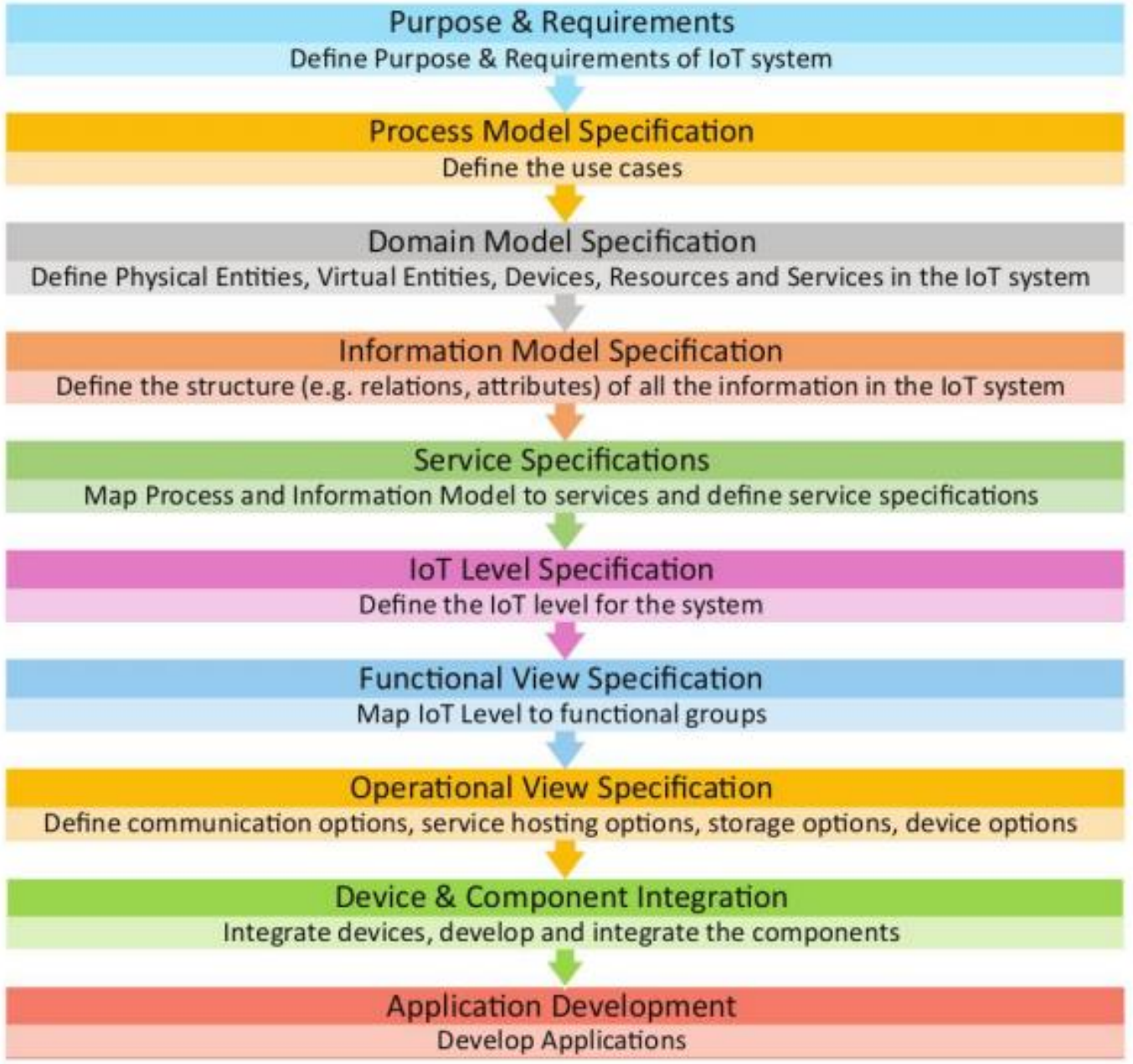
- A generic approach helps to:
  - Avoid lock-in with specific technologies or vendors
  - Maintain flexibility and modularity
  - Facilitate future upgrades and feature addition
- Proposed IoT System Design Methodology
  - Independent of:
    - Product
    - Vendor
    - Service
    - Programming language
  - Based on the IoT-A Reference Model
    - But broad enough to include other industry frameworks

Advantages of the Proposed Methodology

- Reduced Design and Maintenance Time
- Improved Interoperability
- Ease of Comparison between various component choices
- Simplified Testing and Validation
- Reusable and Extensible Designs

Generic IoT System Design Methodology

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification
- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development



Purpose & Requirements Specification

- The first and foundational step in the IoT system design process is to **define the purpose and requirements** of the system.
- This step focuses on understanding and capturing:
  - **System Purpose** – What the IoT system aims to achieve.
  - **System Behavior** – How the system is expected to operate under different conditions.
  - **Functional and Non-Functional Requirements**, including:
    - Data collection requirements
    - Data analysis requirements
    - System management needs
    - Data privacy and security concerns
    - User interface (UI) and usability expectations

Purpose & Requirements Specification

- Applying this methodology to a smart home example:
  - **Purpose:**
    - To automate household functions (e.g., lighting, security, temperature) for convenience, safety, and energy efficiency.
  - **Requirements May Include:**
    - Collect data from sensors (motion, temperature, gas, etc.)
    - Analyze patterns of usage and suggest optimizations
    - Enable remote control via mobile or web interfaces
    - Ensure secure access to device controls and data
    - Provide user-friendly dashboards for configuration

Process Model Specification

- The Process Model Specification defines the logical flow and operational behavior of an IoT system based on use cases. It distinguishes between system-driven and user-driven actions, enabling automated and manual control workflows.
- What This Step Involves
  - **Identify modes of operation** (e.g., automatic vs. manual).
  - **Design process diagrams** that describe:
    - States
    - Decisions
    - System behavior based on conditions
  - Process diagrams typically use:
    - **Circles** to denote the start of a process
    - **Diamonds** for decision-making points
    - **Rectangles** for system states or attributes

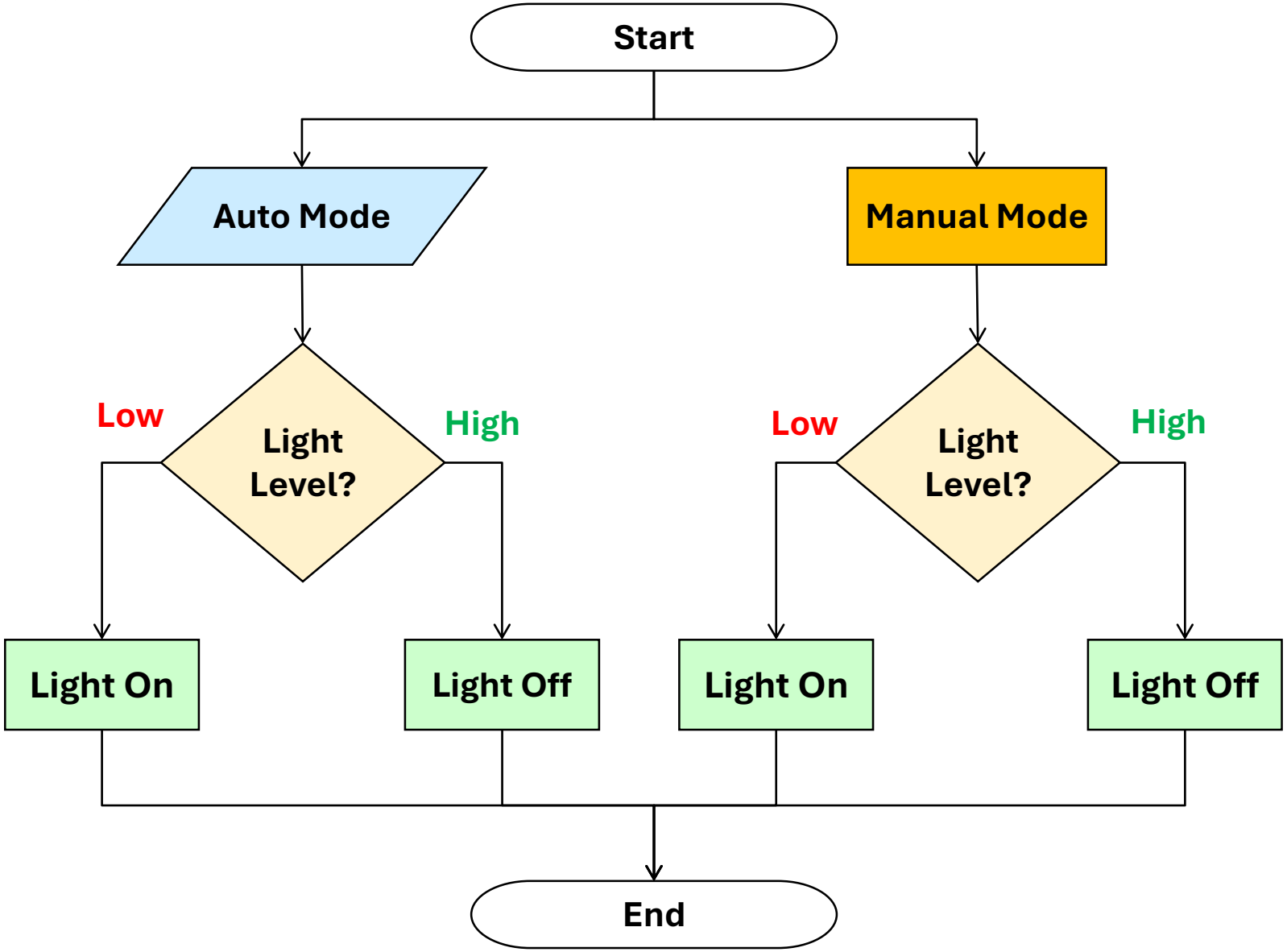
Process Model Specification

Smart Home Automation (Light Control System)

- **System Modes:**
- **Auto Mode**
  - The system **automatically monitors the light level** using sensors.
  - If **light level is low**, the system turns the **light ON**.
  - If **light level is high**, the system turns the **light OFF**.
- **Manual Mode**
  - The system **checks the light setting by the user**.
  - If the **user sets light to ON**, the system turns it ON.
  - If the **user sets light to OFF**, the system turns it OFF.

Process Flow Diagram

Smart Light Control



Domain Model Specification

- The **Domain Model** defines the **core entities, objects, and relationships** within the IoT system being designed. It provides an **abstract representation** of:
  - Concepts
  - Physical/Virtual objects
  - Their interactions
- It identifies and describes:
  - **Entities** (both physical and virtual)
  - **Devices** used for sensing and control
  - **Resources** (software modules on devices or networks)
  - **Services** that expose system functionality to users or applications
- It helps designers **visualize the structure** of the system **without depending on specific technology or vendors**.
- This model is **platform-independent.**

Domain Model Specification

Smart Home Automation

Component	Example in Smart Home
Physical Entity	A room (monitored for brightness), and a light bulb (to be controlled)
Virtual Entity	Digital representation of the room and the bulb (used in apps/dashboards)
Device	A microcontroller (e.g., ESP32 or Raspberry Pi) with a light sensor and relay module
Resource	On-device software that reads light levels and controls bulb via logic
Service	A mobile app service or web interface that turns the light ON/OFF based on user request

- The **domain model** explains **what components** exist in the IoT system.
- It separates the **real-world entities** from their **digital control and representation**.
- It's essential for designing systems that are **scalable**, **modular**, and **interoperable**.

Domain Model Diagram

Smart Home Automation

Association Type

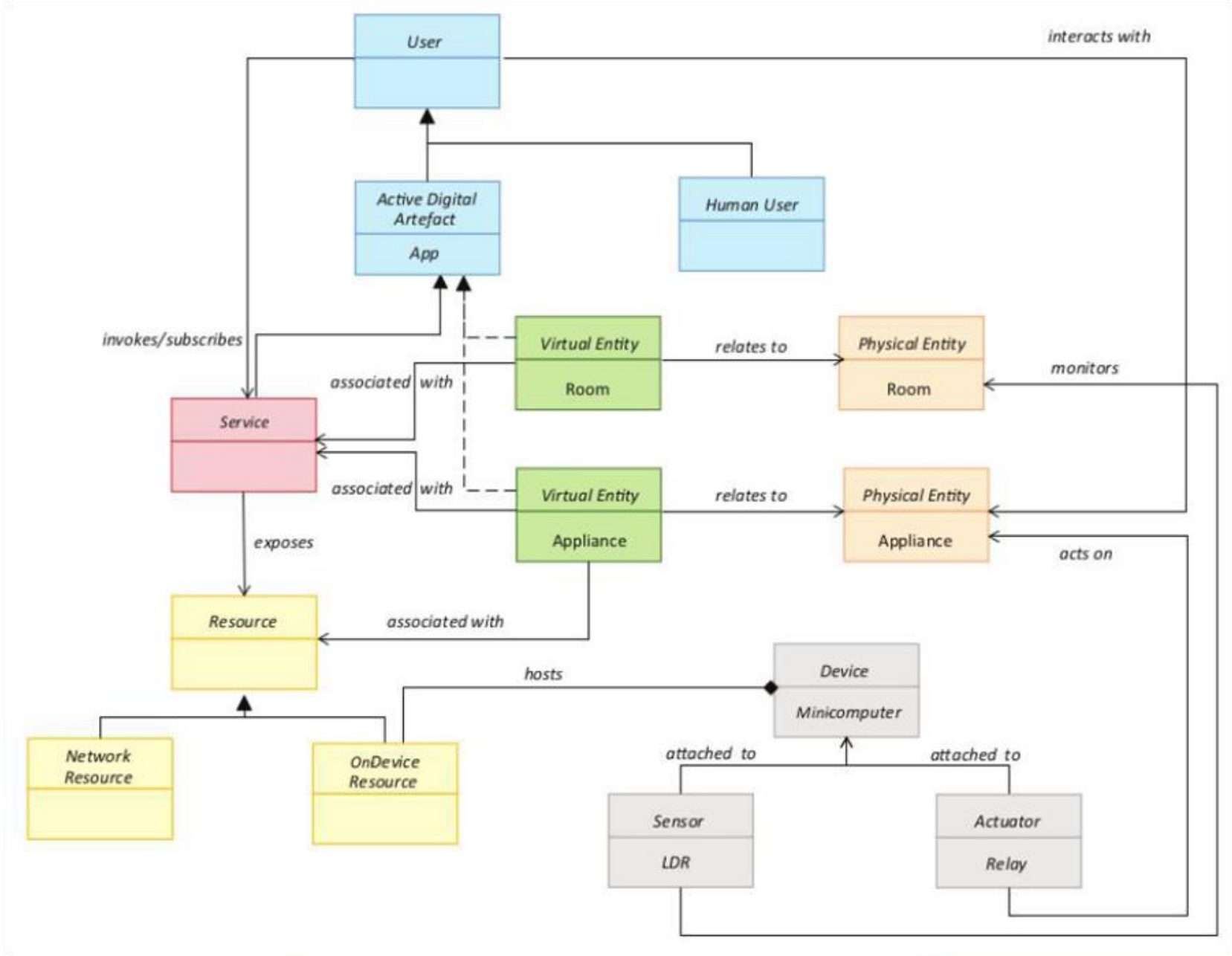
One-way

Two-way

Aggregation

Type: Entity, Device, Service, Resource, Attribute

Type



Information Model Specification

- The Information Model specifies what kind of data is involved in an IoT system by defining the attributes and relationships of virtual entities. This step bridges the gap between conceptual models and implementable data structures.
- The **Information Model** defines the **structure of all the information** in an IoT system. It focuses on:
  - **Attributes** of Virtual Entities
  - **Relationships** among entities
  - Data semantics — **what information** is present and **how it is logically organized**
- **Purpose of Information Modeling**
  - Adds **detail** to the abstract virtual entities defined in the **Domain Model**
  - Makes system design **clear**, **structured**, and **ready for implementation**
  - Enables **data mapping** in services and applications

Information Model Specification

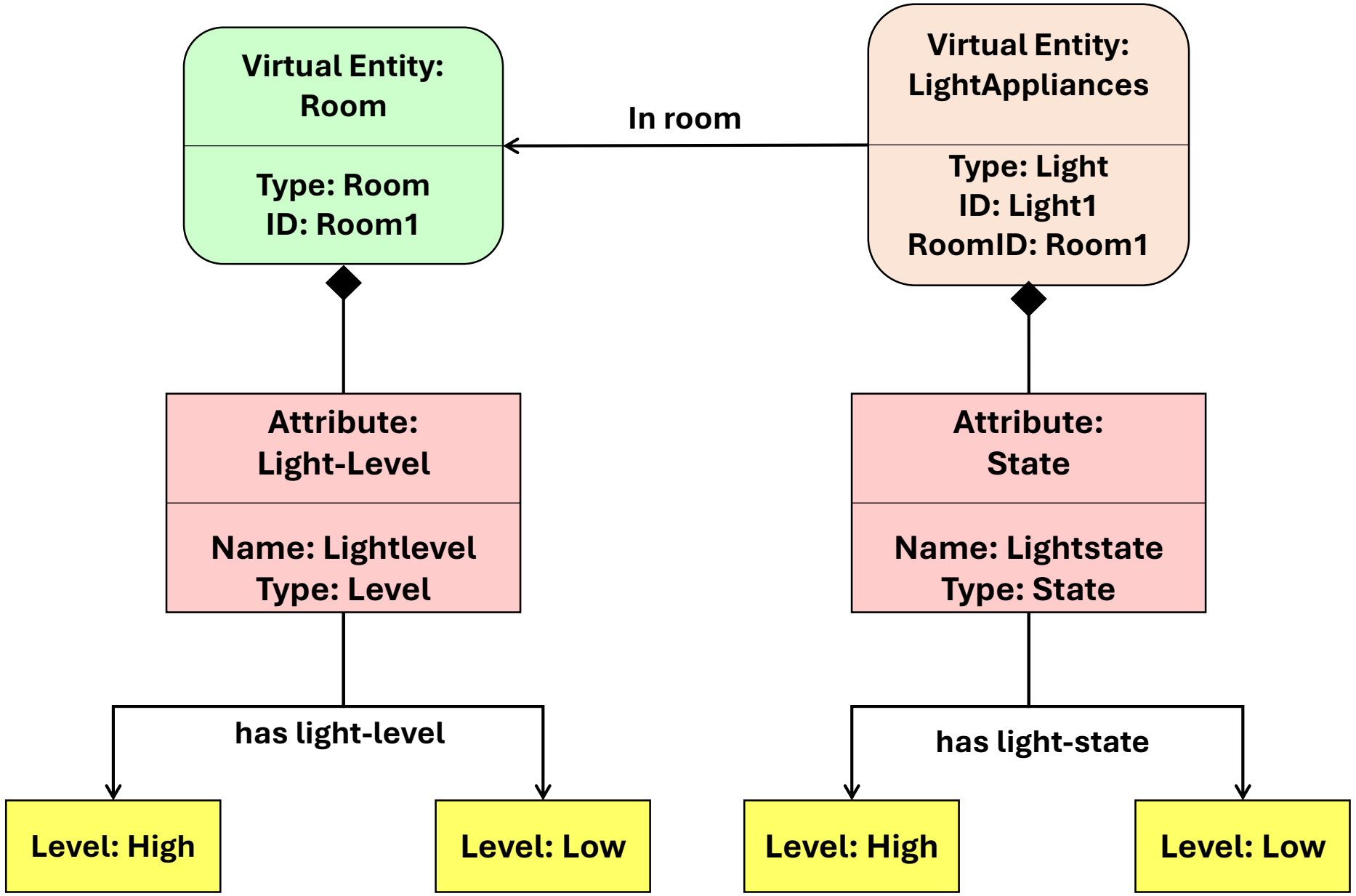
Example: Smart Home Automation System

In the home automation use case:

- There are two **Virtual Entities**:
  - **Virtual Light Entity**
    - Attribute: light_state (e.g., ON/OFF)
  - **Virtual Room Entity**
    - Attribute: light_level (e.g., LOW/MEDIUM/HIGH)

These attributes define what data the system collects and uses for logic, decisions, and services.

Information Model Specification



Service Specification

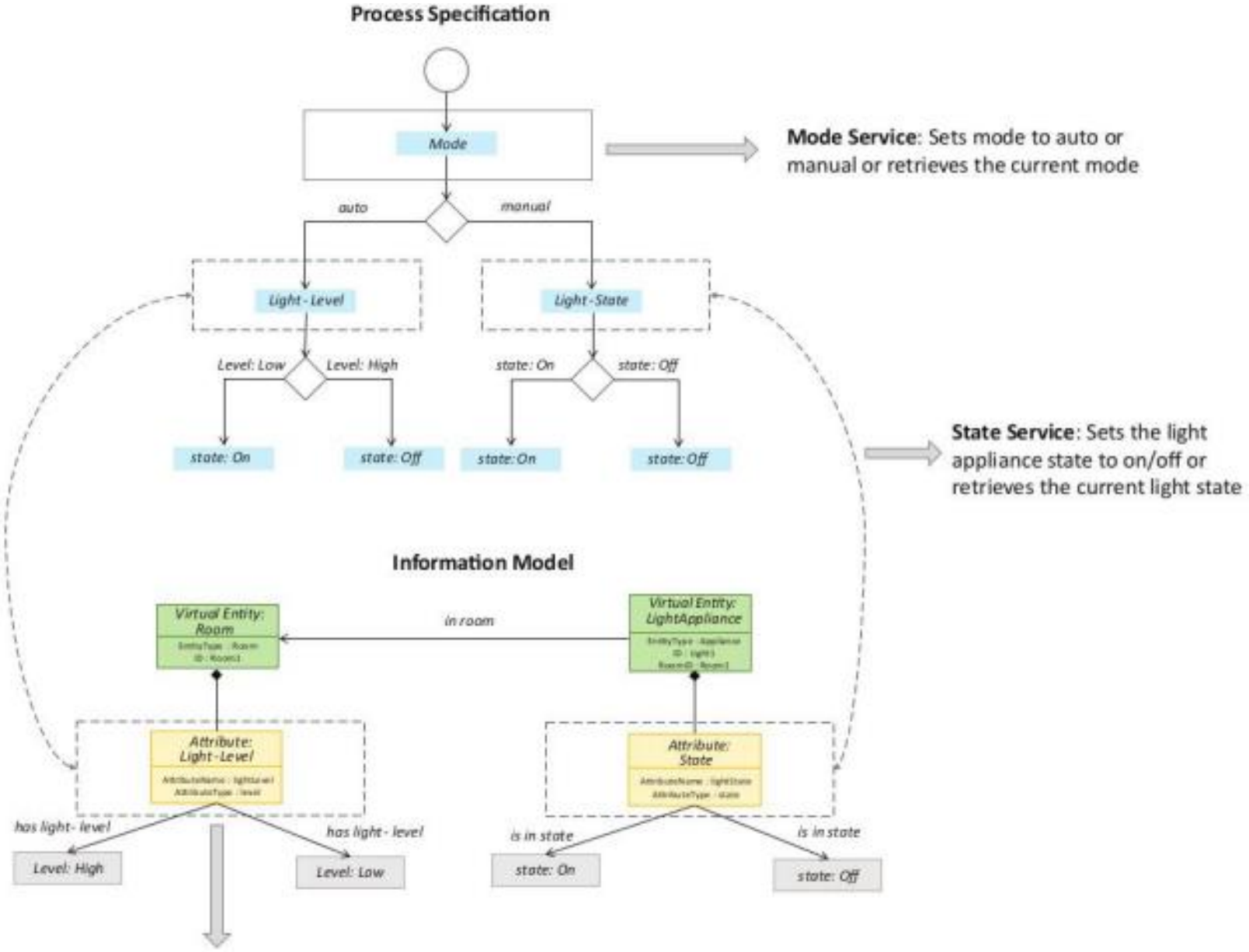
- Service Specification defines how the system will interact with users and devices by exposing state-changing and monitoring operations in the form of services.
- Service specification defines the **services** provided by an IoT system, including:
  - **Service types**: e.g., mode setting, state change, monitoring
  - **Service inputs/outputs**: data received and sent
  - **Service endpoints**: where services are invoked
  - **Service schedules**: timing/frequency of execution
  - **Service preconditions**: required conditions for service activation
  - **Service effects**: outcomes after service execution
- For each state and attribute in the process specification and information model, we define a service. Services either change the state of attributes or retrieve their current values.

Service Specification

Example: Smart Home Automation System

- From the **Process Model** and **Information Model**, we identify **states** (e.g., ON/OFF) and **attributes** (e.g., light-level).
- Based on those, the following services are defined:
- Types of Services
  - **Mode Service**
    - Sets the mode to **Auto** or **Manual**
    - Can also **retrieve** the current mode
    - Example: User chooses “Manual” via app
  - **State Service**
    - Sets the **state** of the light appliance to ON or OFF
    - Or retrieves the current state
    - Example: App turns the light ON, or asks the system what the light's current state is
  - **Controller Service**
    - Works in **Auto mode**
    - Monitors **light level**
    - Automatically switches the **light ON/OFF** based on light-level

Service Specification

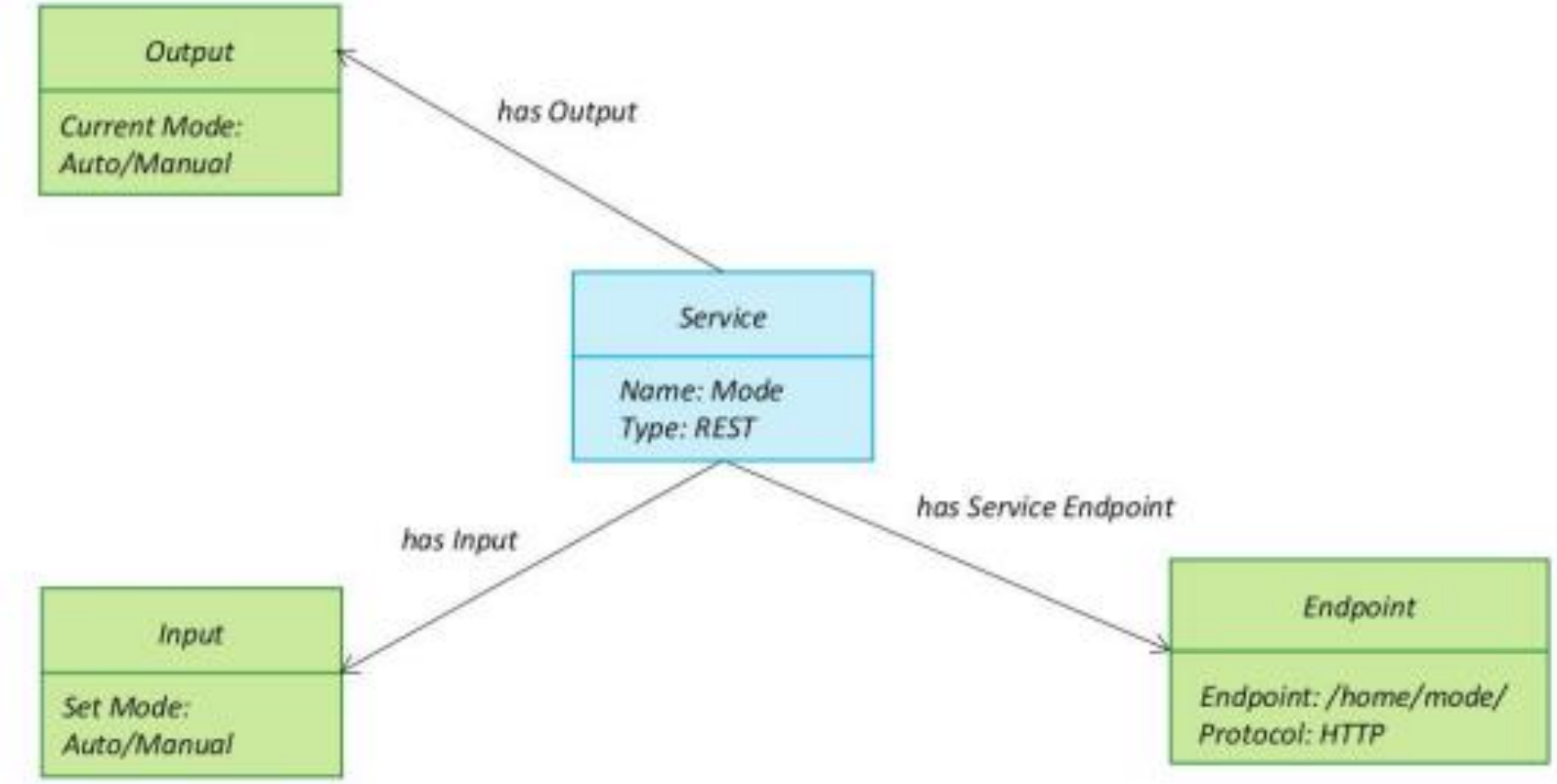


Mode Service: Sets mode to auto or manual or retrieves the current mode

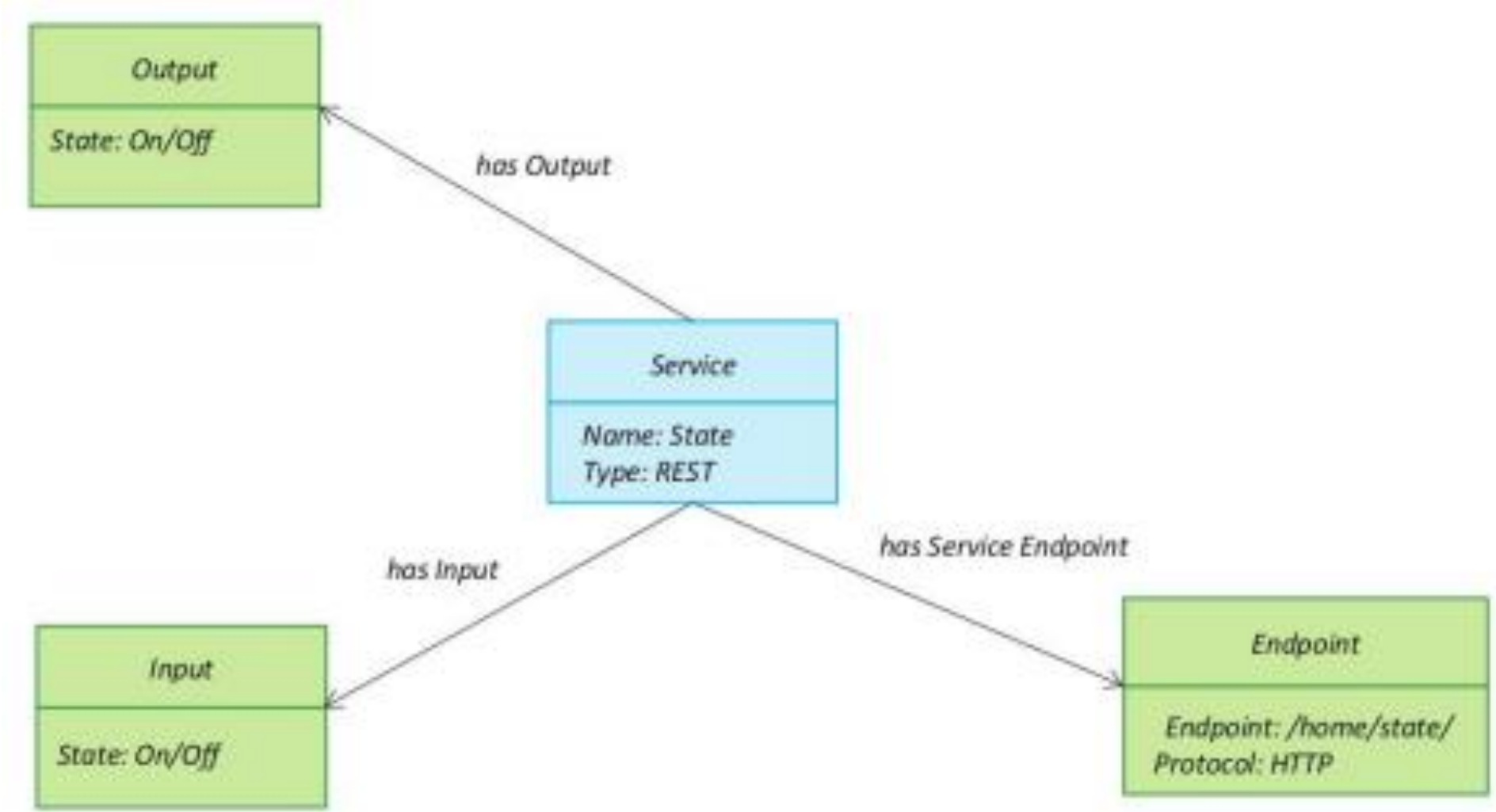
State Service: Sets the light appliance state to on/off or retrieves the current light state

Controller Service: In auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database. In manual mode, the controller service, retrieves the current state from the database and switches the light on/off.

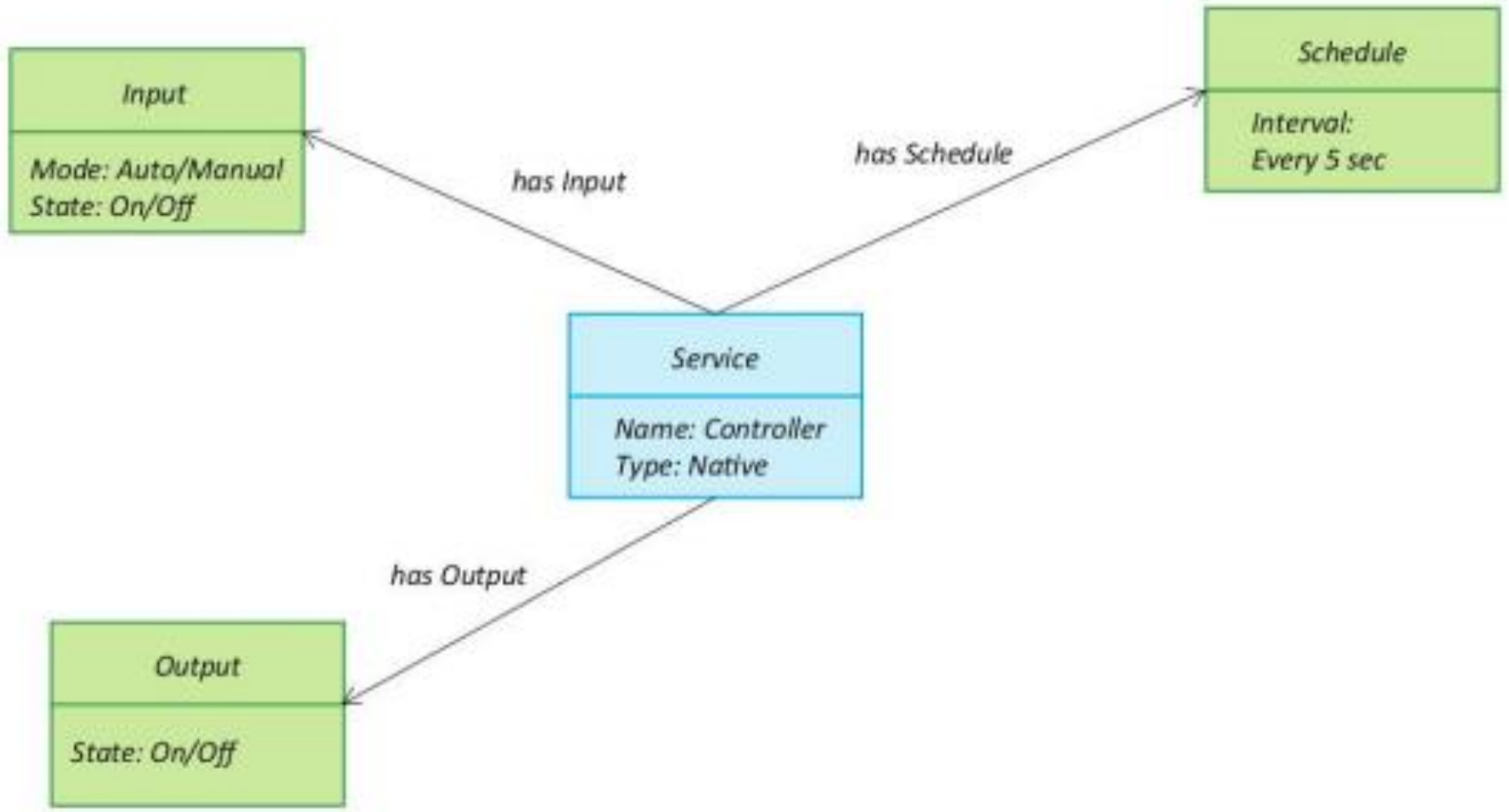
Service Specification



Service Specification



Service Specification



IoT Level Specification

- IoT Level Specification classifies the system based on its deployment complexity and scope. It helps determine appropriate architectural, functional, and communication choices.

- This step involves **defining the IoT level** at which the system operates, based on a previously established **IoT deployment level framework**.

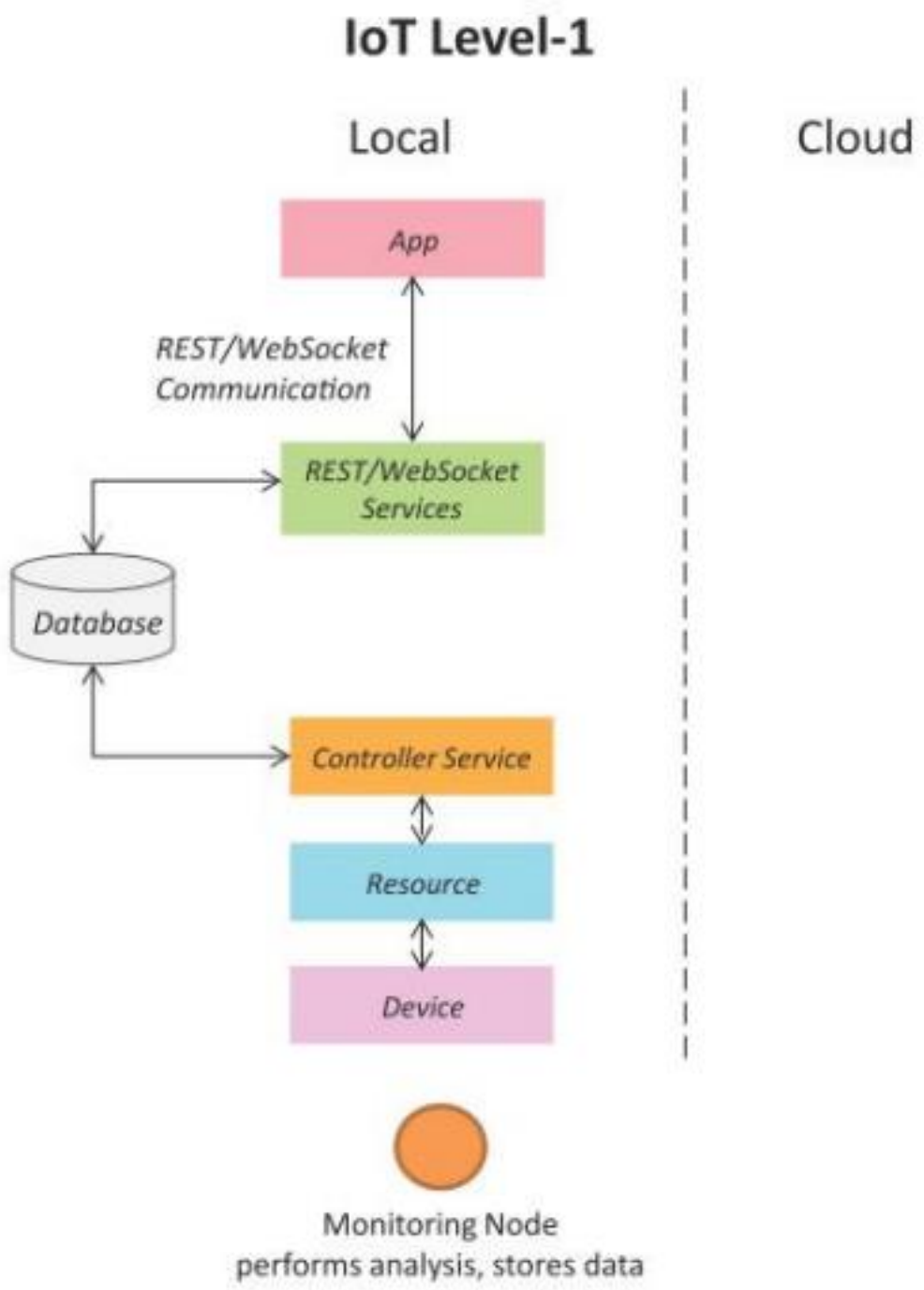
- In earlier chapters, **six IoT deployment levels** were defined, representing increasing complexity and scalability.

- Implications of Choosing IoT Level
  - Determines **architecture complexity**
  - Affects **communication mechanisms**
  - Influences **security, scalability, and cost**

IoT Level Specification

Example: Home Automation System

- The home automation system described here is classified as a **Level-1 IoT system**.
- This means:
  - The system operates **locally**, possibly within a **single premise or room**.
  - It does **not involve cloud integration** or multi-device orchestration across distributed environments.
  - Simple controllers and sensors are used.



Functional View Specification

- The Functional View Specification defines how the system's functionalities are grouped and mapped to subsystems like devices, communication, services, security, and user interfaces. It forms the functional architecture of the IoT solution."

- The **Functional View (FV)** organizes the **functions** of the IoT system into **Functional Groups (FGs)**.

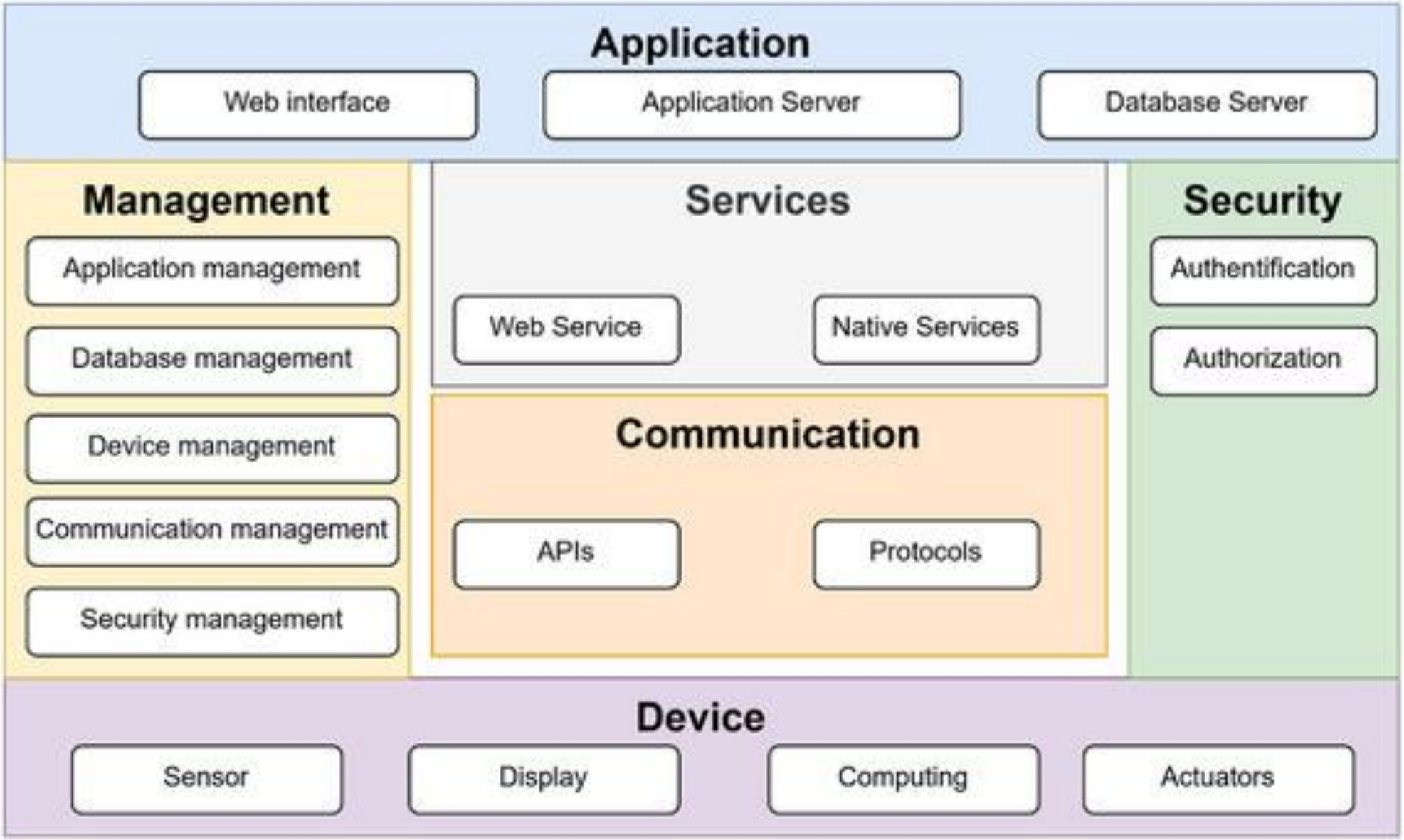
  Each Functional Group:

  - Interacts with the **entities defined in the Domain Model**
  - Provides specific functionalities like sensing, communication, management, etc.

- This classification of FGs allows:

  - **Modular development** of IoT systems
  - **Clear separation of concerns**
  - Easier **testing**, **deployment**, and **maintenance**

Functional View Specification

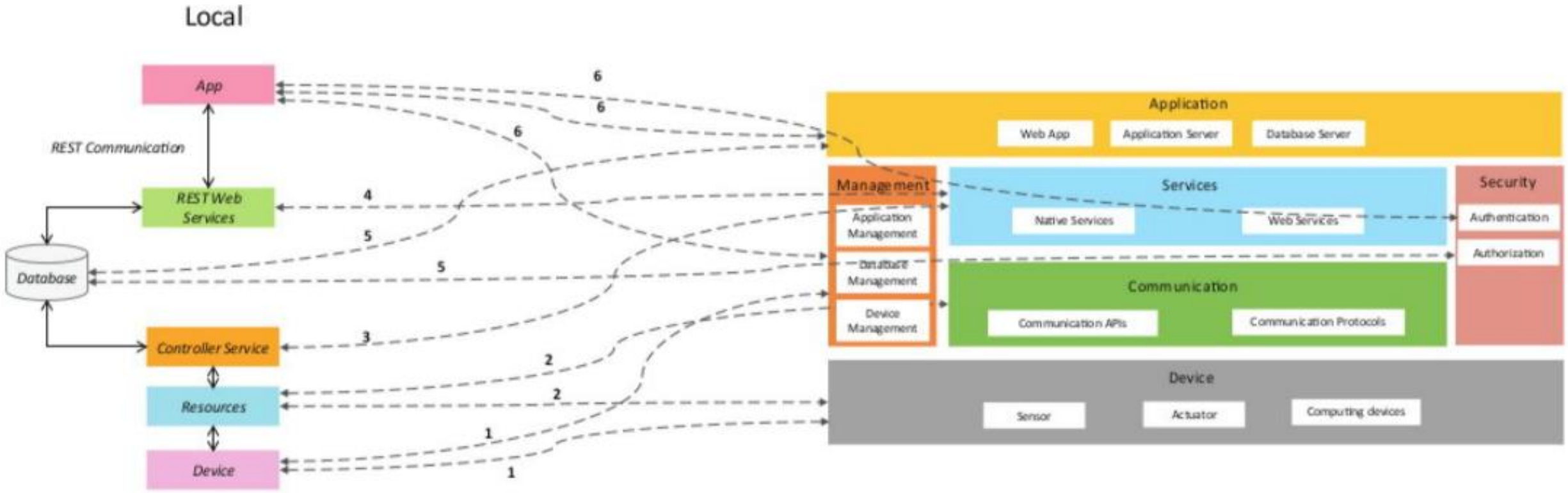


Functional View Specification

Functional Groups in IoT Design Methodology

Functional Group (FG)	Purpose / Description	Example (Smart Home Automation)
Device FG	Contains hardware devices responsible for sensing and actuation.	- Light Sensor (detects brightness) - Relay Module (controls light) - Microcontroller (e.g., Raspberry Pi)
Communication FG	Manages communication between IoT components using protocols and APIs.	- Protocols: Wi-Fi (802.11), IPv4, TCP, HTTP - API: RESTful API for controlling devices remotely
Services FG	Provides core services such as device control, monitoring, and data access.	- Mode Service (Auto/Manual) - State Service (ON/OFF) - Controller Service (automated light control)
Management FG	Handles configuration, provisioning, software updates, and system management.	- Device configuration interface - Over-the-air (OTA) firmware update tool
Security FG	Implements security policies like authentication, authorization, and data protection.	- User login (app authentication) - Access control to prevent unauthorized control of devices
Application FG	Provides user interface and application logic for monitoring and control.	- Mobile App/Web App showing light status and allowing user to control the light manually

Functional View Specification



IoT System Component	Mapped Functional Group(s)	Explanation
IoT Devices	Device FG (sensors, actuators, computing devices) Management FG (device management)	Devices are part of sensing/control, and also require configuration and monitoring
Resources	Device FG (on-device resource) Communication FG (APIs and protocols)	Resources include embedded logic and also APIs for interaction
Controller Services	Services FG (native services) Services FG (web services)	Native logic like automated control; web services expose this logic via APIs
Web Services	Services FG	Web-accessible APIs for interacting with the system
Database	Management FG (database management) Security FG (database security)	Used to store and manage system data securely
Application (UI, app logic, dashboard)	Application FG (web app, application & DB servers) Management FG (app management) Security FG (app security)	Front-end interfaces and their backend servers require both management and security enforcement

Functional View Specification

Mapping of IoT Components to Functional Groups (FGs)

IoT Component	Mapped Functional Group (FG)	Explanation
Sensors & Actuators	Device FG	Used for monitoring (e.g., light sensor) and control (e.g., relay to switch light)
Mini-computer / Microcontroller	Device FG, Management FG	Performs control logic and can be managed (e.g., update firmware, configure settings)
On-device Software (e.g., controller app)	Device FG	Executes sensing/control logic on the device
REST API for light control	Communication FG	Provides service endpoints for communication between devices/apps
Wi-Fi (802.11), IP, HTTP protocols	Communication FG	Enables connectivity and data transfer over the network
Mode and State Services	Services FG	Provide APIs to get/set device states and control logic
Controller Logic (e.g., Auto Light)	Services FG	Native service handling light-level sensing and automatic control
Web Application / Mobile App	Application FG	User-facing app to control/view light state and automation
Database for storing light logs	Management FG, Security FG	Data management and protection via access controls and encryption
User Login / Role Management	Security FG	Secures the system with authentication and role-based access

Operational View Specification

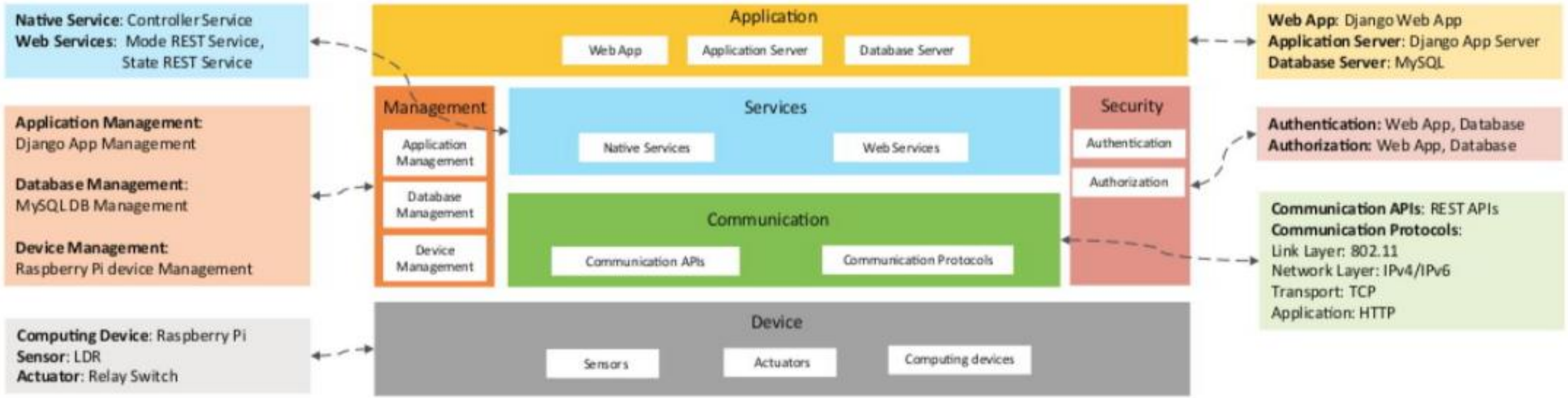
- The **Operational View** is the **eighth step** in the IoT Design Methodology. It focuses on how the different parts of the IoT system are **deployed**, **executed**, and **interact** in the real world. While the functional view tells **what** functions the system performs, the operational view specifies **how** and **where** these functions are carried out.
- **Why the Operational View Matters**
  - **Helps in Technology Selection**
    - Choose Suitable APIs, Protocols, and Platforms
  - **Supports modular deployment**
    - Clear view of where and how components run
  - **Aids scalability and maintenance**
    - Better management of apps, devices, and services
  - Defines security layers and standard communication mechanisms
    - Defines security layers and standard communication mechanisms

Operational View Specification

❖ Key Aspects of the Operational View

Aspect	Description
Device Options	Types of hardware used (e.g., Raspberry Pi, sensors, actuators) and where services run
Service Hosting	Whether services are hosted on-device or remotely (cloud or edge server)
Storage Options	Choice of database and where it's hosted (local or remote)
Communication APIs	Defines how components interact (e.g., RESTful APIs, WebSocket APIs)
Communication Protocols	Specifies network protocols used for communication (e.g., HTTP, TCP/IP, 802.11 Wi-Fi)
Application Stack	Technologies used for web/app development, server, and database setup
Security Controls	Mechanisms for authentication, authorization, and data protection
Management Systems	Interfaces and services to manage applications, databases, and devices

Operational View Specification



Operational View Specification

Example: Operational View for Smart Home Automation

Let's apply the operational view to a **home automation system controlling lights**.

- **Devices:**
  - **Raspberry Pi** – acts as the central computing hub.
  - **LDR (Light Dependent Resistor)** – used to sense light levels.
  - **Relay Module** – to turn lights ON or OFF.
- **Communication:**
  - **Protocols**:
    - Link Layer: 802.11 (Wi-Fi)
    - Network Layer: IPv4/IPv6
    - Transport Layer: TCP
    - Application Layer: HTTP
  - **APIs**: RESTful APIs are used for communication between components.

Operational View Specification

- **Services:**
  - **Controller Service** – implemented in Python, runs on the Raspberry Pi to control lighting based on sensor data.
  - **Mode Service & State Service** – RESTful services built using Django-REST Framework, also hosted on the Pi.
- **Application:**
  - **Django Web App** – user interface to monitor and control lights.
  - **Django App Server** – runs the logic.
  - **MySQL Database** – stores sensor data and logs.
- **Security:**
  - **Authentication & Authorization** – handled at both web app and database layers to prevent unauthorized access.
- **Management:**
  - **Application Management** – managed via Django admin tools.
  - **Database Management** – using MySQL tools or remote DB admin panels.
  - **Device Management** – manage Raspberry Pi via SSH or VNC tools.

Device and Components Integration

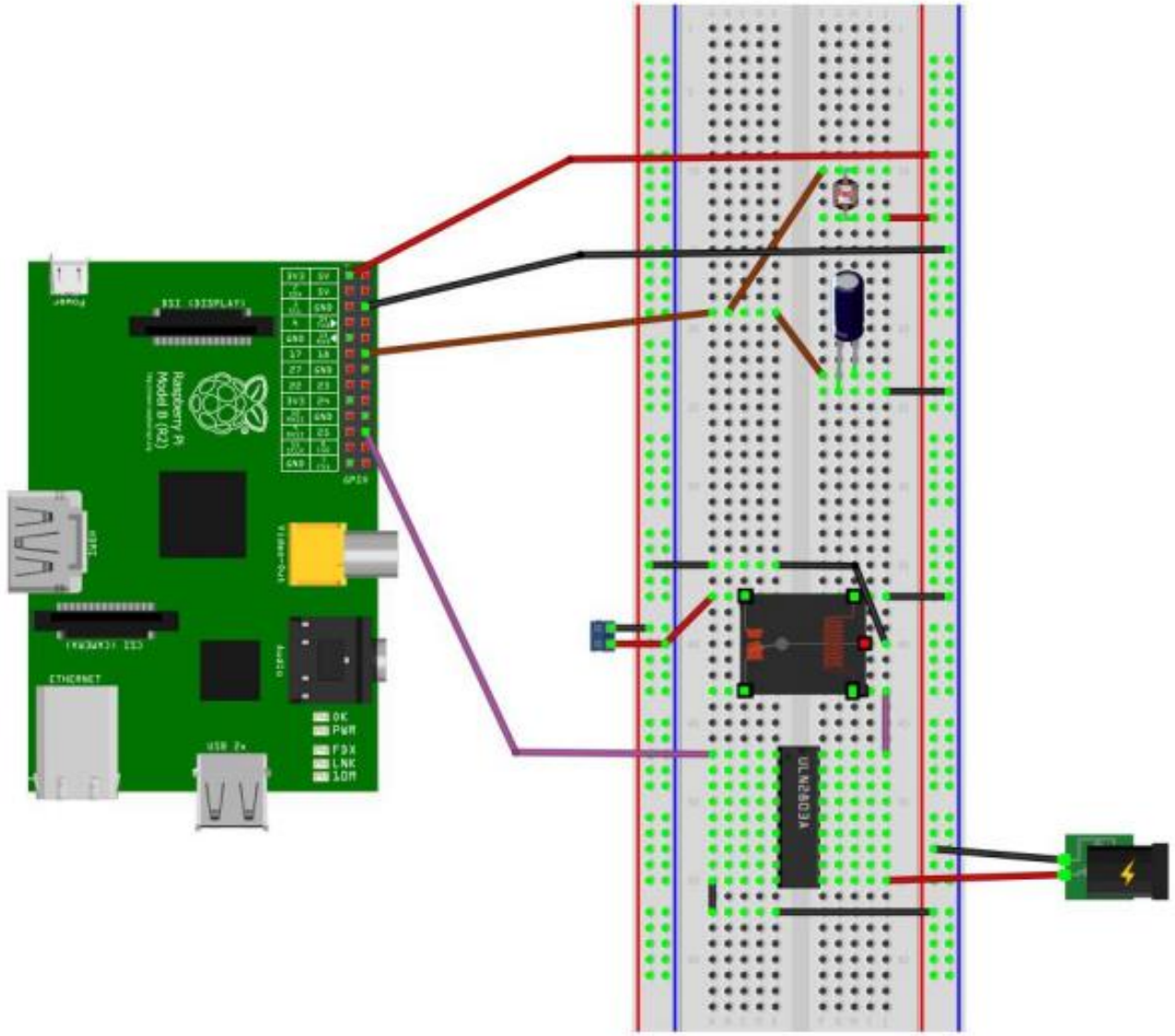
- This step focuses on **assembling** and **interfacing** the physical components (hardware) of the IoT system so that they can work together smoothly.
- **What is Device and Component Integration?**
  - It is the process of:
    - Selecting the **correct hardware components**
    - Ensuring **interoperability** between them
    - **Interfacing** sensors and actuators with the **computing device**
    - Making sure that the hardware setup supports the **application and operational view**
- **Importance of This Step**
  - Ensures correct data acquisition (Sensors must be properly wired and read accurately)
  - Enables real-world actuation (Relay must receive correct signals from Pi to control appliances)
  - Supports software-hardware linkage (Allows the software logic (controller, services) to interact with devices)

Device and Components Integration

❖ Components in the Home Automation Example

Component	Function
Raspberry Pi	Acts as the central controller or computing device
LDR Sensor	Detects light intensity in the room
Relay Switch (Actuator)	Used to switch the light ON/OFF based on decisions made by the controller

Step	Description
Hardware Wiring	Connect LDR to analog input and relay to GPIO of Raspberry Pi
Driver Installation	Install required Python libraries (e.g., gpiozero, RPi.GPIO)
Sensor Calibration	Test LDR input values under different lighting conditions
Control Logic	Write code to control relay based on LDR readings or user input
API/Service Setup	Develop REST APIs or Python scripts to control these hardware interactions



Application Development

- The final step in the IoT design methodology is to **develop the actual IoT application** that:
  - Provides a **user interface**
  - Enables control and monitoring of the IoT system
  - Integrates all the system logic and services defined in previous steps
- Why This Step is Important
  - Provides **usability** to the end-user
  - Connects all the layers: sensors, controller, communication, services
  - Enables **real-time control and monitoring**
  - Helps test and validate the complete IoT system functionality

Application Development

❖ **Example: Home Automation Web Application**

- The application is a **web interface** that allows users to:
- **Switch between Auto and Manual Mode**
- **Turn the light ON or OFF manually** (only when in Manual Mode)

❖ **How the System Works:**

Mode	Control Mechanism
Auto Mode ON	The system uses the LDR sensor to monitor light level and automatically controls the light (via relay switch). The UI disables manual control and simply shows the current light state.
Auto Mode OFF	The system allows the user to manually turn ON/OFF the light from the application interface. The automatic LDR-based control is bypassed.

Application Development

❖ Backend Logic Implementation

Component	Tool/Framework Used
Web Framework	Django (Python)
REST API	Django-REST Framework
Hardware Control	Python scripts interfacing with GPIO pins
Database	MySQL (for storing light/mode state)
Frontend Controls	HTML + CSS + JavaScript

Use of Big Data in IoT (Short Notes)

The Internet of Things (IoT) consists of billions of connected devices and sensors that continuously generate huge amounts of data. This massive, fast, and diverse data is known as Big Data. Since traditional tools cannot handle such data, Big Data technologies are essential for storing, processing, and analyzing IoT information. Together, IoT and Big Data create intelligent systems that enable automation, prediction, and smarter decision-making.

IoT data is explained using the **5 Vs of Big Data**:

- **Volume:** Large amounts of sensor data generated continuously.
- **Velocity:** High-speed, real-time data transmission.
- **Variety:** Different types of data—structured, unstructured, multimedia, JSON, etc.
- **Veracity:** Sensor noise and inaccuracies requiring cleaning.
- **Value:** Useful insights gained through analysis.

A typical IoT–Big Data architecture includes:

1. **Data collection** using devices and protocols like MQTT, CoAP, LoRaWAN, ZigBee, or 5G.
2. **Edge/gateway processing** for filtering, aggregation, or pre-processing.
3. **Data storage** in systems such as HDFS, NoSQL databases (MongoDB, Cassandra), cloud storage (AWS S3, Azure), or time-series databases.
4. **Data processing** using batch tools (Hadoop MapReduce) or real-time stream tools (Kafka, Spark Streaming, Flink).

Big Data analytics helps extract meaningful insights from IoT data. Machine learning and statistical techniques support anomaly detection, predictive maintenance, energy optimization, and real-time monitoring. Examples include predicting machine failures in industries, detecting health abnormalities using wearables, and optimizing traffic in smart cities.

Major advantages include:

- **Real-time analytics** for instant monitoring and quick response.
- **High scalability** to handle huge data volumes.
- **Predictive analytics** for forecasting trends and improving decisions.

Challenges include large computational requirements, noisy or incomplete data, privacy concerns, and difficulty integrating diverse devices and data formats. However, advancements in cloud computing, edge computing, and AI are reducing these limitations.

In conclusion, Big Data is crucial for turning IoT-generated raw data into valuable insights. It enables smarter operations, automation, and innovation across healthcare, agriculture, industry, transportation, and smart homes. The integration of IoT and Big Data continues to drive the development of intelligent, data-driven systems for the future.

IoT Visualization

IoT Visualization is the process of converting large volumes of IoT-generated data into visual formats such as charts, graphs, dashboards, and maps. It helps users understand complex sensor data, monitor system performance, detect anomalies, and make quick data-driven decisions.

Need for IoT Visualization (Short Points)

- **Understand complex data:** IoT devices generate huge, diverse data. Visualization makes it easier to interpret patterns and insights.
- **Real-time monitoring:** Dashboards show live data, helping identify anomalies instantly.
- **Better decision-making:** Visual data supports quick operational and strategic decisions.
- **Identify trends:** Helps find patterns, correlations, and long-term changes.
- **Improves communication:** Presents data in a simple, intuitive way for both technical and non-technical users.
- **User engagement:** Interactive visuals encourage exploration and deeper analysis.
- **Resource optimization:** Helps identify inefficiencies in energy, water, or equipment usage.

Data Sources for IoT Visualization (Short Points)

- **Sensors & Devices:** Temperature, humidity, pressure, speed, GPS, etc.
- **Network Data:** Signal strength, connectivity, bandwidth, and network performance.
- **Cloud Platforms:** Cloud-based data storage and analytics systems used for visual dashboards.
- **Data Streams:** Real-time sensor streams showing live metrics and alerts.

IoT Visualization Techniques (Short Points)

- **Time-Series Visualization:** Line charts, heatmaps, and area charts for tracking changes over time.
- **Geospatial Maps:** Plotting IoT data on maps to analyze location-based insights.
- **Dashboards:** A unified screen showing key metrics and KPIs.
- **Interactive Visuals:** Filters, drill-downs, and hover insights to explore trends.

Popular Tools for IoT Visualization

- **Tableau** – Advanced interactive dashboards
- **Power BI** – Microsoft's powerful visualization platform
- **ThingSpeak** – Real-time visualization for IoT sensors
- **InfluxDB** – Time-series database with built-in visual tools
- **Grafana** – Customizable dashboards for IoT and server metrics

Applications of IoT Visualization

- **Smart Cities:** Traffic, waste management, air quality monitoring
- **Industrial IoT:** Process monitoring, predictive maintenance
- **Healthcare:** Wearables, patient monitoring, anomaly alerts
- **Agriculture:** Soil moisture, crop health, weather insights
- **Retail:** Customer behavior, inventory tracking
- **Smart Homes:** Energy usage, security alerts, automation

Challenges in IoT Visualization

- Very large and fast data streams
- Privacy and security concerns
- Noisy or low-quality data
- Real-time processing requirements
- Need for proper context to avoid misinterpretation

IoT visualization transforms complex IoT data into clear, actionable insights. With effective tools and techniques, organizations can improve decision-making, optimize operations, and fully utilize the potential of IoT systems.

Low-Power Design: Bluetooth Low Energy (BLE)

Introduction

Bluetooth Low Energy (BLE) is a wireless communication technology designed specifically for **low-power, short-range** communication between devices. It is part of the Bluetooth 4.0+ specification and is widely used in IoT applications like wearables, health trackers, smart homes, and beacons.

BLE focuses on **energy efficiency**, allowing small battery-powered devices to operate for **months or years** without frequent charging.

Key Features of BLE

- **Ultra-low power consumption:** Uses very little energy during data transmission and remains in sleep mode most of the time.
- **Short-range communication:** Typically, up to **10–30 meters** (can extend up to 100 m with BLE 5.0).
- **Low-cost hardware:** BLE chips are inexpensive and widely available.
- **Fast connection setup:** Devices can connect and transfer small amounts of data quickly.
- **Compatibility:** Supported by smartphones, embedded devices, and most IoT platforms.
- **Flexible topology:** Supports star topology (central–peripheral) and broadcasting.

BLE Architecture and Components

BLE uses a layered architecture similar to classic Bluetooth but optimized for low-power communication:

1. Physical Layer (PHY)

- 2.4 GHz ISM band
- Uses Gaussian Frequency-Shift Keying (GFSK)
- BLE 5.0 supports 1 Mbps, 2 Mbps, and long-range PHY

2. Link Layer

- Controls advertising, scanning, connection, and data transfer
- Manages device roles:
    - Central (master)
    - Peripheral (slave)

- Broadcaster
- Observer

3. Host Layer Includes:

- L2CAP **(Logical Link Control and Adaptation Protocol)**
- Attribute Protocol (ATT)
- Generic Attribute Profile (GATT) **– organizes data as services and characteristics**
- Security Manager **for pairing and encryption**

Power-Saving Mechanisms in BLE

BLE achieves low power consumption using:

1. Advertising and Scanning Intervals

- Devices broadcast data at intervals
- Longer intervals = less power consumption

2. Connection Interval

- Time between active data exchanges
- Larger interval reduces wake-ups → saves power

3. Sleep Modes

- Device sleeps when no data is exchanged
- Wake-on-radio architecture

4. Low Duty Cycle Operation

- Minimal active transmission time
- Device remains mostly idle

5. Small Data Packets

- Less power required for transmission
- Effective for sensor-type applications

BLE Communication Process

Step 1: Advertising

Peripheral device broadcasts short packets with its identity.

Step 2: Scanning

Central device listens for advertising packets.

Step 3: Connection Establishment

Central initiates a connection request.

Step 4: Data Exchange

Uses GATT profile to exchange services and characteristics.

Step 5: Low-power Sleep

Devices return to sleep when inactive.

Applications of BLE

- **Wearable devices:** Smartwatches, fitness bands, heart-rate monitors
- **Healthcare:** Glucose meters, pulse oximeters
- **Smart home:** Locks, lights, air-quality sensors
- **Asset tracking:** BLE beacons in malls, airports
- **Industrial IoT:** Machine monitoring, condition sensors
- **Retail:** iBeacon-based advertising
- **Automotive:** Keyless entry, sensors

Advantages of BLE

- Highly energy-efficient
- Low cost and widely supported
- Quick connection setup
- Secure communication (AES-128 encryption)
- Suitable for battery-powered devices
- Good for small, infrequent data transfers

Limitations of BLE

- Not suitable for high-bandwidth applications
- Shorter range compared to Wi-Fi
- Limited payload size
- Lower throughput

Overview of Range Extension Techniques (Data Mining and Mesh Networking)

In IoT systems, **range extension techniques** are essential to ensure that devices spread across large or remote areas can communicate effectively. Two important approaches used to extend communication range and enhance network performance are **Data Mining** and **Mesh Networking**.

Data Mining for Range Extension

Data mining helps extend the effective communication range of IoT systems not by increasing radio distance, but by improving **data efficiency, prediction, and routing decisions**. IoT devices often generate massive amounts of sensor data, and mining this data enables the system to operate intelligently even when devices are far apart or intermittently connected.

How Data Mining Supports Range Extension

- **Predictive Analytics:** Helps predict sensor behavior, device failure, or connectivity patterns, allowing the system to work with fewer transmissions and reducing the need for continuous long-range communication.
- **Data Aggregation & Compression:** Reduces the amount of data sent over long distances, making communication more efficient.
- **Pattern Recognition:** Identifies optimal times for transmission based on signal quality, network load, or distance.
- **Improved Routing Decisions:** Data mining algorithms analyze historical routing paths to select the most reliable nodes for forwarding data in long-range networks.

The effective communication capability increases even when physical range is limited, because the system becomes smarter and more efficient in handling long-distance data flow.

Mesh Networking for Range Extension

Mesh networking is one of the most powerful techniques for extending IoT communication range. In a mesh network, each node acts both as a device and a router. Data can hop from one node to another until it reaches its destination, greatly extending the total network coverage.

Key Features of Mesh Networking

- **Multi-Hop Communication:** Devices forward messages to neighboring nodes, enabling long-distance transmission even with low-power radios like BLE or Zigbee.
- **Self-Healing:** If one node fails, the network automatically finds an alternate route.
- **Scalability:** Adding more nodes increases both coverage and reliability.
- **Energy Efficiency:** Each node only needs short-range communication, saving power while extending the total range.

How Mesh Networking Extends Range

Device A --> Node 1 --> Node 2 --> Node 3 --> Gateway

Even if Device A is far from the gateway, intermediate nodes relay the signal across the network.

A large geographic area can be covered using many low-power devices without increasing individual transmission power.

Internet of Everything (IoE)

The **Internet of Everything (IoE)** is an advanced concept that extends the idea of the Internet of Things (IoT). While IoT focuses mainly on connecting **devices and sensors**, IoE expands this scope by connecting **people, processes, data, and things** into a unified intelligent system. The goal of IoE is to create smarter interactions, improve decision-making, and deliver more meaningful and personalized experiences.

IoE views the digital world not just as a network of objects but as a system where **all elements interact** to generate greater value. For example, IoE uses data analytics to interpret sensor data, processes to automate responses, and human inputs to guide system behavior. Through this, IoE creates a highly integrated environment where information flows efficiently across different components.

Four Key Components of IoE

1. People

People connect through devices, apps, and online platforms. Their behaviours, preferences, and feedback help customize digital services and improve system performance.

2. Data

Massive amounts of data are collected from sensors, devices, and users. IoE uses analytics and AI to convert raw data into actionable insights that support automation and intelligent decision-making.

3. Processes

Processes ensure that the right information reaches the right person or device at the right time. This includes automated workflows, security protocols, and optimized communication methods.

4. Things

These include everyday physical objects—sensors, machines, vehicles, appliances—that communicate and interact over networks.

How IoE Works

IoE creates an interconnected ecosystem where:

- **Things sense and collect data**
- **Data is processed and analyzed**
- **Processes automate responses**
- **People interact and make decisions**

This continuous feedback loop creates a highly efficient and intelligent system.

Benefits of IoE

- Improved automation and efficiency

- Better decision-making through real-time analytics
- Enhanced customer experiences
- Reduced energy and operational waste
- Smarter cities, industries, and homes

Examples of IoE

- Smart cities using sensors, data analytics, and automated control
- Healthcare systems integrating wearables, patient data, and medical processes
- Smart transportation with connected vehicles and traffic analytics
- Smart retail using customer data and interactive systems

Overview of Android App Development Tools

Android app development relies on a set of tools and environments that help developers design, build, test, and deploy applications efficiently. The primary tool is **Android Studio**, Google's official IDE, which provides a complete development environment including code editing, debugging, UI designing, and performance monitoring. Along with this, developers use the **Android SDK**, which contains essential APIs, libraries, and command-line tools required for app creation. Build management, virtual testing, and version control tools also play a crucial role in ensuring smooth development and high-quality output. These tools together enable developers to create responsive, secure, and feature-rich Android applications.

Key Android App Development Tools

1. Android Studio

- Official IDE by Google
- Provides code editor, layout designer, and debugging tools
- Supports Gradle-based project management

2. Android SDK (Software Development Kit)

- Contains libraries, APIs, and development tools
- Includes important components like ADB, emulator, and build tools

3. ADB (Android Debug Bridge)

- Command-line tool for communicating with physical Android devices
- Used for debugging, installing apps, and accessing device logs

4. Android Emulator

- Allows testing apps on virtual devices
- Supports different screen sizes, resolutions, and Android versions

5. Layout Editor

- Visual drag-and-drop tool for UI design
- Helps create responsive layouts using XML

6. Gradle Build System

- Automates the build process
- Manages dependencies and project configurations

7. Firebase

- Cloud-based platform for backend services
- Provides authentication, cloud storage, push notifications, and analytics

8. Version Control Systems (Git/GitHub)

- Used to track code changes and collaborate with teams
- Essential for managing large projects and backups

9. Android Jetpack Libraries

- Set of components for UI, navigation, lifecycle management, and background tasks
- Helps build modern, stable, and maintainable apps

Overview of iOS App Development Tools

iOS app development requires a specialized set of tools provided primarily by Apple to build, test, and deploy applications for iPhones, iPads, and other iOS devices. The main development environment is **Xcode**, Apple's official IDE, which offers a complete suite of features such as code editing, UI design, debugging, and performance testing. Developers use **Swift** and **Objective-C** as the primary programming languages, along with frameworks like **SwiftUI**, **UIKit**, and **CoreData** to build powerful and user-friendly applications. Apple also provides additional tools such as the **iOS Simulator** for virtual testing, **Instruments** for performance analysis, and **TestFlight** for beta testing. Together, these tools help developers create efficient, secure, and high-quality iOS applications optimized for Apple's ecosystem.

Key iOS App Development Tools

1. Xcode (Official IDE)

- Main software used for developing iOS apps
- Includes code editor, interface builder, compiler, and debugger
- Supports Swift, Objective-C, and SwiftUI
- Provides tools for building, running, and testing iOS apps

2. Swift Programming Language

- Modern, fast, and safe language designed by Apple

- Easier syntax and better performance compared to Objective-C
- Widely used for new iOS app development

3. Objective-C
    - Traditional programming language for iOS/macOS
    - Still used in legacy apps and some Apple frameworks
4. Interface Builder (within Xcode)
    - Visual drag-and-drop tool for designing app interfaces
    - Supports Auto Layout for responsive UI design
5. iOS Simulator
    - Allows testing apps on virtual iPhones/iPads
    - Helps verify app behavior on different screen sizes and iOS versions
6. Instruments
    - Powerful performance analysis tool
    - Helps detect memory leaks, CPU usage, battery drain, and UI lag
7. TestFlight
    - Official platform for beta testing
    - Allows developers to distribute pre-release versions to testers
    - Collects feedback and crash reports
8. Cocoa Touch Framework
    - Provides essential libraries for app features
    - Supports gestures, animations, notifications, and other iOS functionalities
9. SwiftUI Framework
    - Modern UI toolkit for building app interfaces with declarative coding
    - Enables faster development and live previews
10. Git/GitHub (Version Control)
    - Tracks code changes and supports teamwork
    - Essential for large, collaborative iOS projects

Use of Big Data in IoT

IoT devices continuously produce data in various formats, including structured, semi-structured, and unstructured forms. Big Data technologies provide the infrastructure and tools needed to process, store, and analyze this data efficiently.

Key Applications of Big Data in IoT:

1. **Data Storage and Management:**
    - Platforms like Hadoop, Apache Spark, and cloud storage solutions (AWS, Azure, Google Cloud) handle the massive data volume, velocity, and variety from IoT devices.
2. **Real-Time Analytics:**
    - Enables processing streaming data for real-time monitoring, anomaly detection, and alerts, e.g., detecting faults in industrial equipment or unusual patterns in energy usage.
3. **Predictive Maintenance:**
    - Analyzing historical IoT data to predict failures and schedule maintenance before breakdowns, reducing downtime and costs.
4. **Personalization:**
    - In smart homes or healthcare, Big Data algorithms analyze user behavior to provide tailored recommendations or automated controls.
5. **Enhanced Decision-Making:**
    - Provides stakeholders with insights into patterns and trends, enabling data-driven decisions across industries like agriculture, logistics, or urban planning.

Use of Visualization in IoT

Visualization helps to interpret the complex Big Data generated by IoT in an easily digestible and interactive manner. By converting raw data into visual formats, such as charts, graphs, or maps, it becomes easier to identify trends, outliers, and actionable information.

Key Applications of Visualization in IoT:

1. **Real-Time Monitoring Dashboards:**
    - Displays data streams from IoT devices in real time, such as smart factory equipment status, environmental conditions, or fleet locations.
2. **Trend Analysis and Forecasting:**
    - Historical data visualizations like time-series charts help identify patterns and predict future scenarios, e.g., electricity consumption trends.
3. **Geospatial Data Visualization:**
    - Mapping data onto geographic locations, such as vehicle tracking, crop health in smart agriculture, or pollution hotspots in smart cities.

4. **Anomaly Detection:**
    - Using visual tools to highlight deviations from expected performance, enabling rapid responses to issues.
5. **User Experience Improvement:**
    - Intuitive, user-friendly visualizations (e.g., gauges, heatmaps) make IoT solutions more accessible to non-technical users.

Industry 4.0

Industry 4.0, also known as the **Fourth Industrial Revolution**, refers to the integration of advanced digital technologies into manufacturing and industrial processes. It represents a shift towards smart factories and intelligent production systems that leverage connectivity, automation, and real-time data. This transformation enhances efficiency, flexibility, and innovation across industries.

Key Features of Industry 4.0

1. **Interconnectivity:**
    - Machines, devices, sensors, and people are interconnected via the Internet of Things (IoT), enabling seamless communication and collaboration.
2. **Data-Driven Decision Making:**
    - Real-time data collection and analysis through Big Data and analytics tools empower informed and adaptive decision-making.
3. **Automation and Robotics:**
    - Advanced robotics and autonomous systems automate repetitive and hazardous tasks, improving precision and safety.
4. **Cyber-Physical Systems (CPS):**
    - Integration of physical machinery with digital systems allows monitoring, control, and simulation of processes in real time.
5. **Cloud Computing:**
    - Centralized data storage and computing resources enable scalable and on-demand access to computational power and services.
6. **Artificial Intelligence (AI) and Machine Learning:**
    - AI-powered systems optimize production processes, predictive maintenance, and supply chain management through adaptive learning.
7. **Additive Manufacturing (3D Printing):**
    - Enables rapid prototyping, customization, and efficient production of complex components.
8. **Digital Twins:**
    - Virtual replicas of physical assets or systems allow real-time simulation, monitoring, and optimization.
9. **Enhanced Human-Machine Interaction:**
    - Advanced interfaces like augmented reality (AR) and virtual reality (VR) facilitate intuitive interaction between humans and machines.
10. **Sustainability:**
    - Focuses on reducing waste, energy consumption, and environmental impact through resource-efficient production processes.

Key Technologies in Industry 4.0

- **Internet of Things (IoT):** Facilitates real-time monitoring and control.
- **Big Data and Analytics:** Processes massive datasets to identify trends and make predictions.
- **AI and Machine Learning:** Automates decision-making and optimizes operations.
- **Robotics and Automation:** Enhances productivity and precision.
- **Cloud and Edge Computing:** Supports flexible and scalable IT infrastructure.
- **Blockchain:** Ensures data security and transparency.
- **5G Connectivity:** Provides high-speed communication for seamless operations.

Benefits of Industry 4.0

1. **Increased Efficiency:**
    - Automation and real-time insights streamline processes, reduce downtime, and improve productivity.
2. **Cost Reduction:**
    - Predictive maintenance and optimized resource use lower operational expenses.
3. **Customization:**
    - Enables mass customization of products to meet individual customer needs.
4. **Improved Quality:**
    - Continuous monitoring and advanced analytics enhance product quality and reduce defects.
5. **Supply Chain Optimization:**
    - Provides end-to-end visibility and agility in supply chain management.
6. **Workplace Safety:**
    - Robotics and automation reduce the risks associated with hazardous tasks.
7. **Sustainability:**
    - Promotes energy efficiency and waste reduction.

Applications of Industry 4.0

- **Smart Factories:** Fully automated and connected production lines.
- **Predictive Maintenance:** Using IoT and AI to anticipate equipment failures.
- **Supply Chain Management:** Real-time tracking and optimization of logistics.
- **Healthcare:** AI-assisted diagnostics and precision manufacturing of medical devices.
- **Agriculture:** Smart farming using IoT sensors and drones.

Overview of RFID (Radio Frequency Identification)

RFID (Radio Frequency Identification) is a technology that uses electromagnetic fields to automatically identify and track tags attached to objects. These tags contain electronically stored information, which can be read by RFID readers without the need for direct line-of-sight, making it a highly versatile and efficient method for identification and data capture.

Components of an RFID System

1. **RFID Tag (Transponder):**
    - **Passive Tags:** Powered by the electromagnetic field from the reader; no internal battery.
    - **Active Tags:** Contain a battery and can transmit signals over longer distances.
    - **Semi-Passive Tags:** Use a battery for internal operations but rely on the reader's signal for communication.
    - Tags consist of:
        - **Chip:** Stores information (e.g., serial number).
        - **Antenna:** Transmits and receives radio signals.
2. **RFID Reader (Interrogator):**
    - Sends out radio waves to power passive tags and communicate with them.
    - Receives and processes the data transmitted by the tags.
3. **Middleware/Software:**
    - Connects the RFID reader to enterprise systems, managing the collected data and integrating it with applications.

How RFID Works

1. The RFID reader emits a radio signal.
2. The tag's antenna picks up the signal and powers the chip (for passive tags).
3. The chip processes the signal and sends back the stored data.
4. The reader captures the tag's response and transmits the data to a computer or cloud system for processing

Overview of iOS

iOS is Apple Inc.'s proprietary operating system designed for its mobile devices, including the iPhone, iPad, and iPod Touch. Known for its user-friendly interface, security features, and seamless integration with Apple's ecosystem, iOS has become one of the most popular mobile operating systems globally.

Key Features of iOS

1. **User Interface (UI):**
    - A clean, intuitive, and touch-optimized interface with gestures like swipe, tap, and pinch for navigation.
2. **App Store:**
    - A digital marketplace offering millions of applications for productivity, entertainment, education, and more.
3. **Security and Privacy:**
    - Features like Face ID/Touch ID, data encryption, and app sandboxing ensure user safety.
    - Strong privacy policies, such as App Tracking Transparency (ATT), give users control over their data.
4. **Seamless Ecosystem Integration:**
    - Works effortlessly with other Apple devices (e.g., Macs, Apple Watch, AirPods) using features like Handoff, Continuity, and iCloud.
5. **Regular Updates:**
    - Apple provides consistent software updates, including new features, security patches, and performance enhancements, even for older devices.
6. **Built-in Apps:**
    - Essential apps like Safari, Mail, iMessage, FaceTime, and Photos come pre-installed for daily tasks.
7. **Voice Assistant (Siri):**
    - A smart assistant capable of performing tasks, answering queries, and integrating with third-party apps.
8. **Focus and Productivity Features:**
    - Tools like Focus Mode, Screen Time, and Shortcuts improve productivity and allow personalized device usage.