UNIT-1
INTRODUCTION AND CONCEPTS

Introduction to Internet of Things, Physical Design of IoT, Logical Design of IoT–IoT Enabling Technologies– IoT levels & Deployment Templates.

Domain Specific IoTs: Introduction–Home Automation–Cities, Environment–Energy–Retail, Logistics–Agriculture, Industry, Health & Lifestyle.

Introduction to Internet of Things

- IoT consist of things that have unique identities and are connected to internet.
- By 2020 there will be a total of 50 billion devices /things connected to internet.
- IoT is not limited to just connecting things to the internet but also allow things to communicate and exchange data
- Definition: A dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual “things” have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into information network, often communicate data associated with users and their environments.

Characteristics:

1) Dynamic & Self Adapting: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, users context or sensed environment.

Eg: the surveillance system is adapting itself based on context and changing conditions.

2) Self Configuring: allowing a large number of devices to work together to provide certain functionality.

3) Interoperable Communication Protocols: support a number of interoperable communication protocols and can communicate with other devices and also with infrastructure.

4) Unique Identity: Each IoT device has a unique identity and a unique identifier (IP address).

5) Integrated into Information Network: that allow them to communicate and exchange data with other devices and systems.

Applications of IoT:

1) Home : Smart lighting, Smart Appliances, Intrcasion detection, smoke/Gas detections.

2) Cities: Smart packing, Smart roads, Structural health monitoring, Emergency Response.

3) Environment: Weather monitoring, Air pollution monitoring, Noise pollution monitoring and forest fire detection.

4) Energy: Smart grids, Renewable Energy systems and prognostics.

5) Retail : Inventory management, Smart payment, small vending machines.

6) Logistics : Route generation and scheduling, Fgleet tracking, Shipment monitoring and Remote vehicle diagnostics.

7) Agriculture : Smart irrigation and green house control.

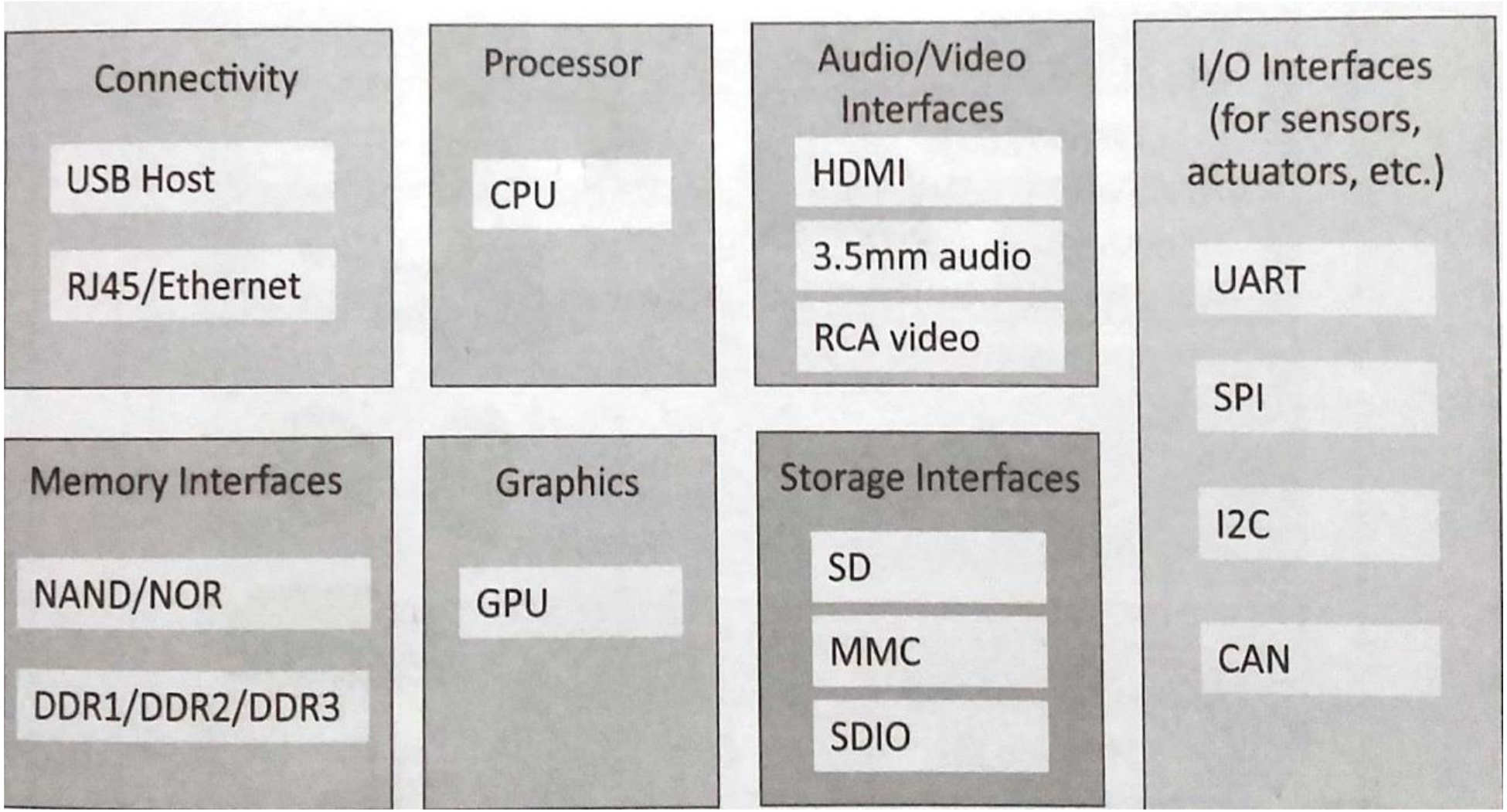
8) Industry : Machine diagnosis and prognosis, indoor air quality monitoring

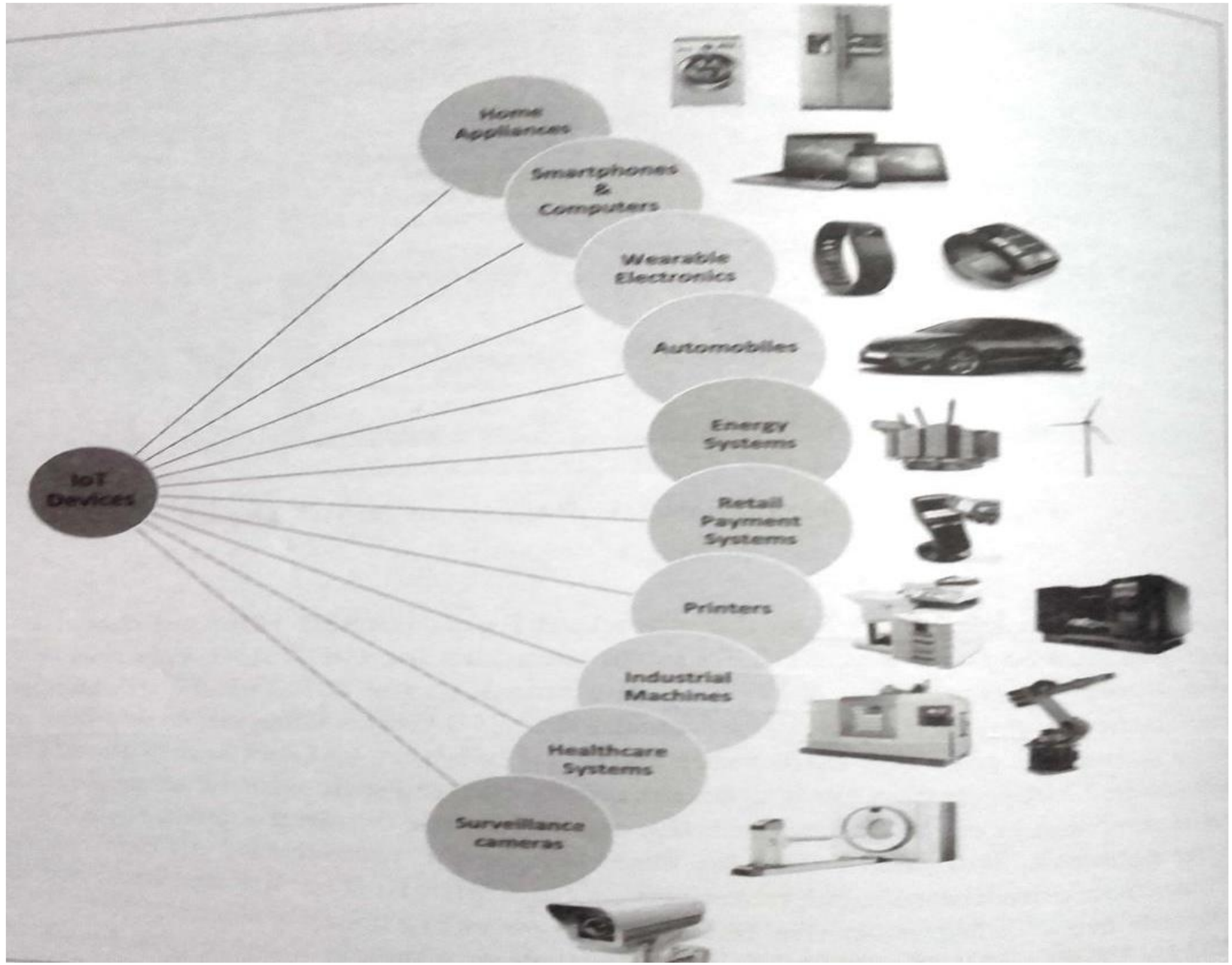
9) Health & Life Style: Health and fitness monitoring and wearable electronics.

Physical Design of IoT

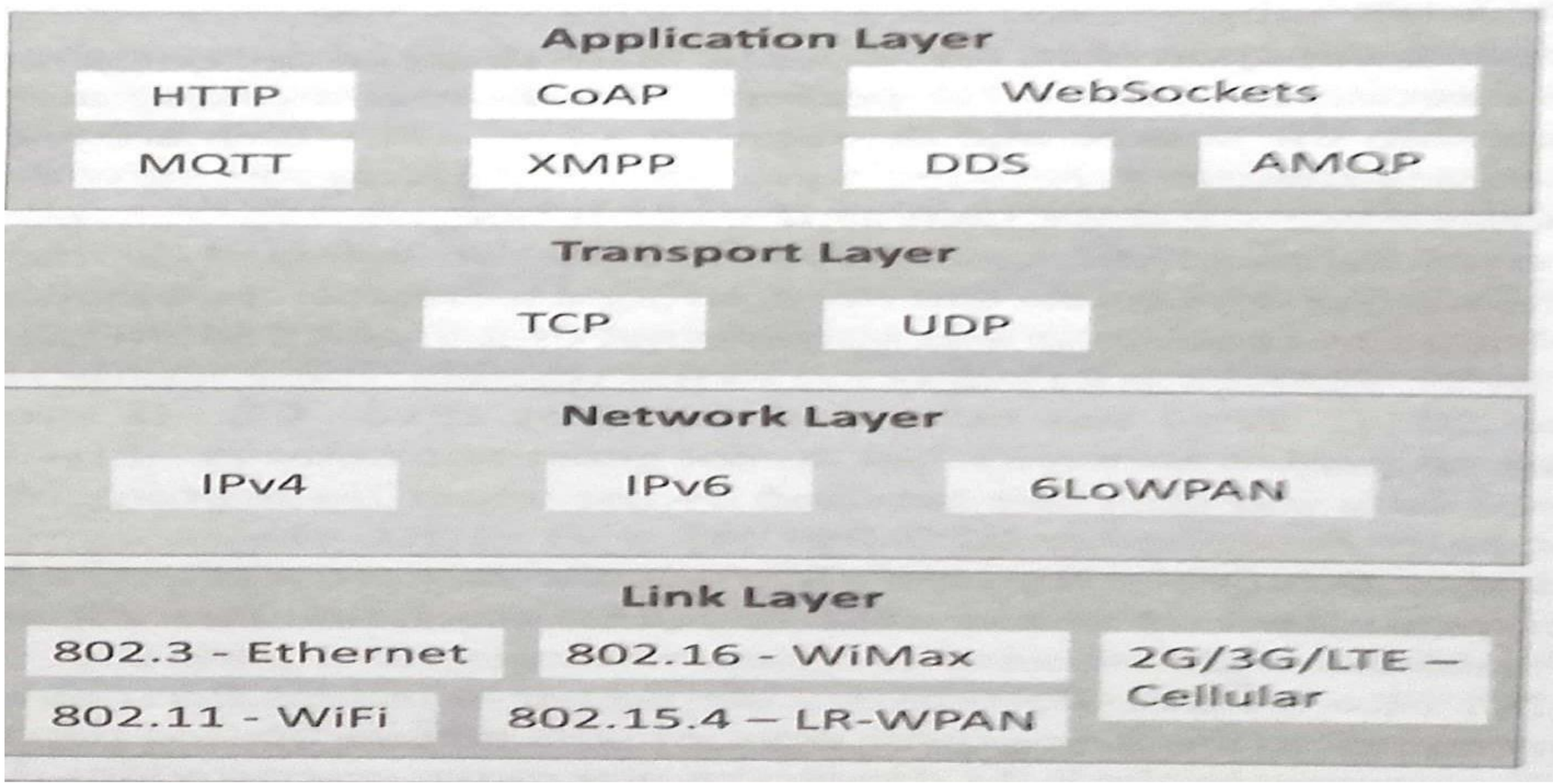
1. Thinks in IoT

- 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 and
  - (iv) audio/video interfaces.


Connectivity
USB Host
RJ45/Ethernet
Processor
CPU
Audio/Video Interfaces
HDMI
3.5mm audio
RCA video
I/O Interfaces (for sensors, actuators, etc.)
UART
SPI
I2C
CAN
Memory Interfaces
NAND/NOR
DDR1/DDR2/DDR3
Graphics
GPU
Storage Interfaces
SD
MMC
SDIO


IoT Devices
Home Appliances
Smartphones & Computers
Wearable Electronics
Automobiles
Energy Systems
Retail Payment Systems
Printers
Industrial Machines
Healthcare Systems
Surveillance cameras

2. IoT Protocols:



a) Link Layer :

- Protocols determine how data is physically sent over the networks physical layer or medium.
- Link layer determines how packets are coded and signaled by the hardware device over the medium to which the host is attached.

- **Protocols:**

- **802.3-Ethernet:** IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 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. Eg: 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 60Ghzband.
- **802.16 - WiMax:** IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.
- .**802.15.4-LR-WPAN:** IEEE802.15.4 is a collection of standards for low rate wireless personal area network(LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.
- **2G/3G/4G-Mobile Communication:** Data rates from 9.6kb/s(2G) to up to100Mb/s(4G).

B) **Network/Internet Layer:**

Responsible for sending IP datagrams from source network to destination network. Datagrams contains source and destination address.

Protocols:

- **IPv4:** Internet Protocol version4 is used to identify the devices on a network using a hierarchical addressing scheme. 32 bit address. Allows total of 2^32addresses.
- **IPv6:** Internet Protocol version6 uses 128 bit address scheme and allows 2^128 addresses.
- **6LOWPAN:**(IPv6overLowpowerWirelessPersonalAreaNetwork)operates in 2.4 GHz frequency range and data transfer 250 kb/s.

C) Transport Layer:

- Provides end-to-end message transfer capability independent of the underlying network.
- 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. Avoids network congestion and congestion collapse.
- **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 guaranted delivery.

D) **Application Layer:** Defines how the applications interface with lower layer protocols to send data over the network. Enables process-to-process communication using ports.

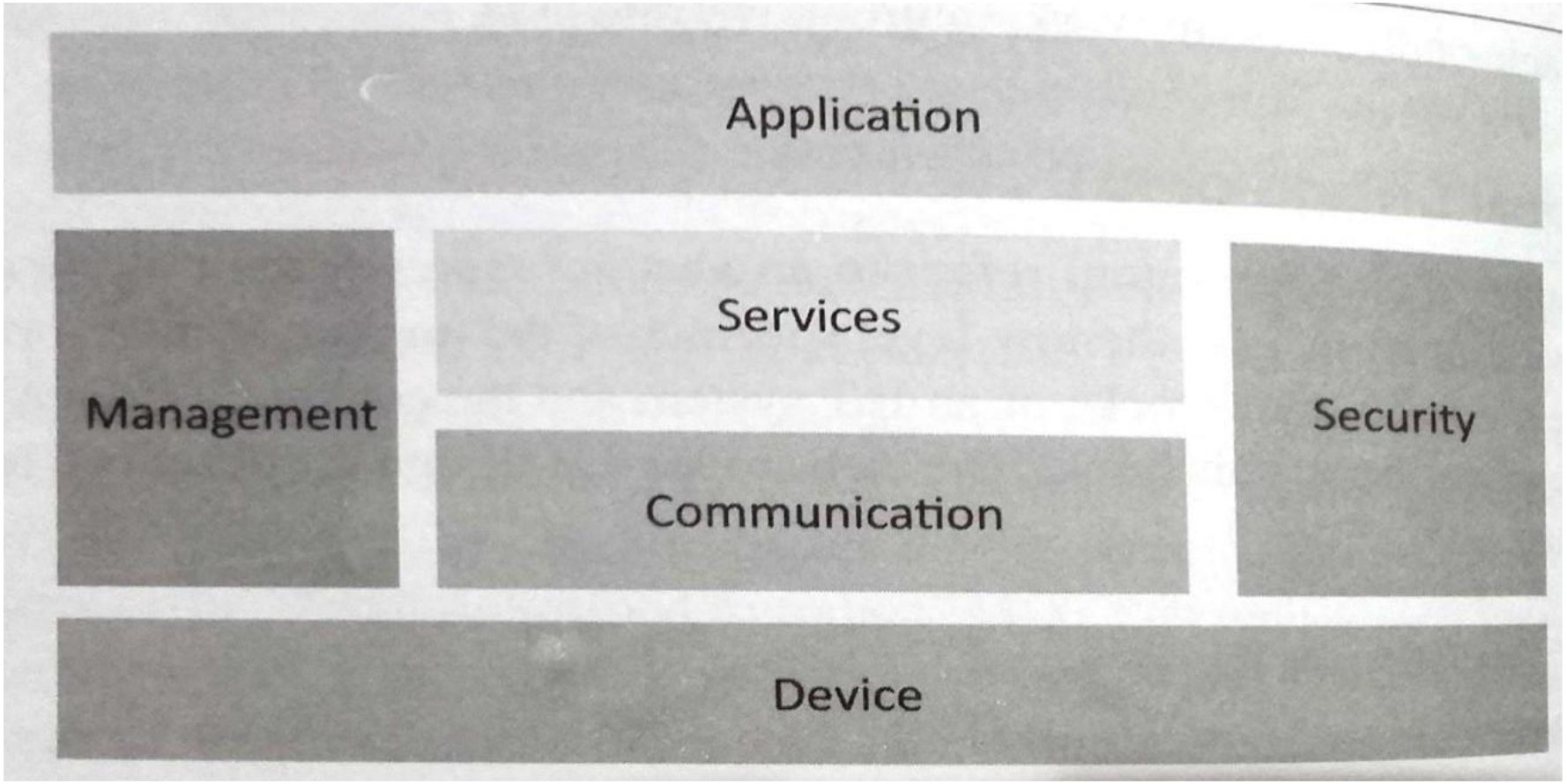
Protocols:

- **HTTP:** Hyper Text Transfer Protocol that forms foundation of WWW. Follow request- response model Stateless protocol.
- **CoAP:** Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained network. Uses client- server architecture.
- **WebSocket:** allows full duplex communication over a single socket connection.
- **MQTT:** Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.
- **XMPP:** Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- **DDS:** Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.
- **AMQP:** Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

Logical Design of IoT

1) IoT Functional Blocks 2) IoT Communication Models 3) IoT Comm. APIs

1) IoT Functional Blocks: Provide the system the capabilities for identification, sensing, actuation, communication and management.

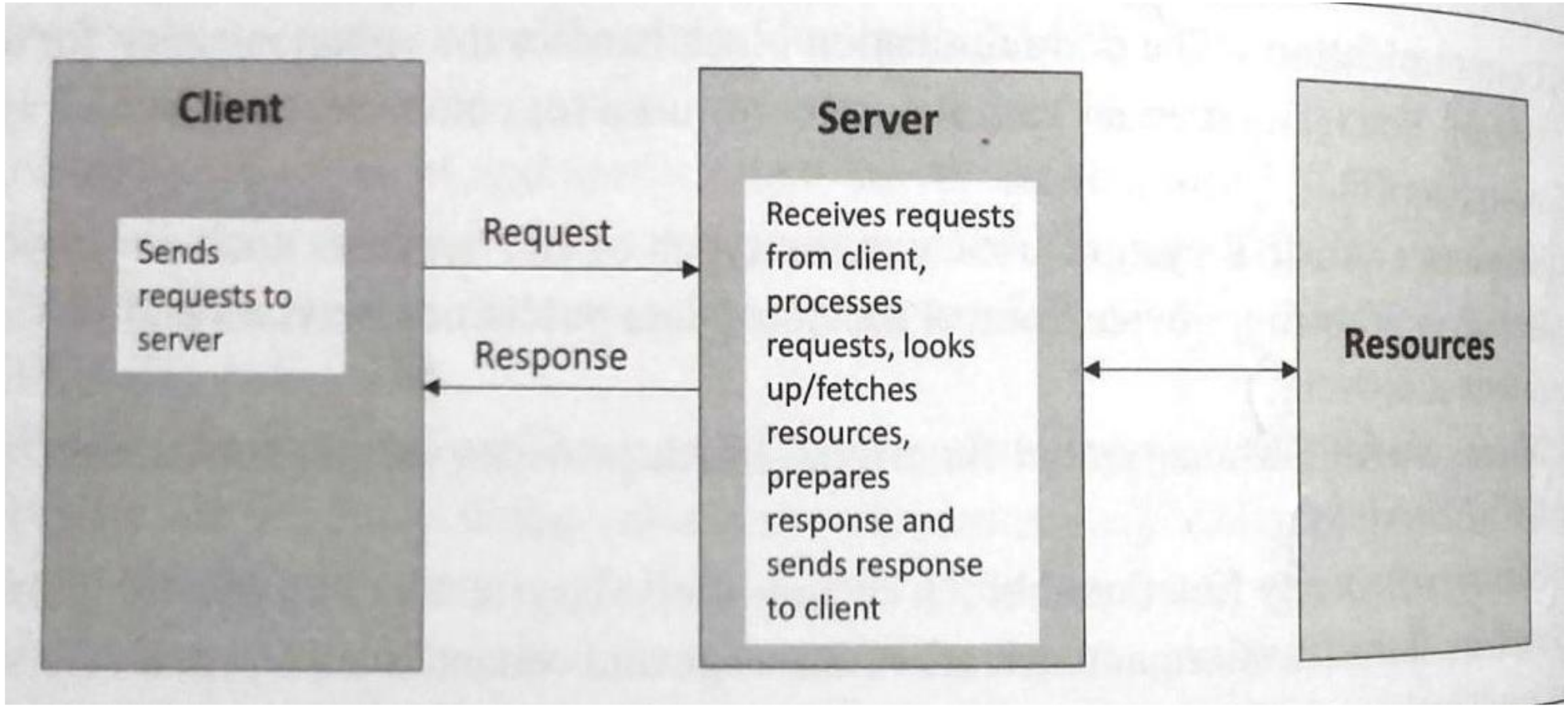


- **Device:** An IoT system includes of devices that provide sensing, actuation, monitoring and control functions.
- **Communication:** handles the communication for IoTsystem.
- **Services:** for device monitoring, device control services, data publishing services and services for device discovery.
- **Management:** Provides various functions to govern the IoT system.
- **Security:** Secures IoT system and priority functions such as authentication ,authorization, message and context integrity and data security.
- **Application:** IoT application provide an interface that the users can use to control and monitor various aspects of IoT system.

2) IoT Communication Models:

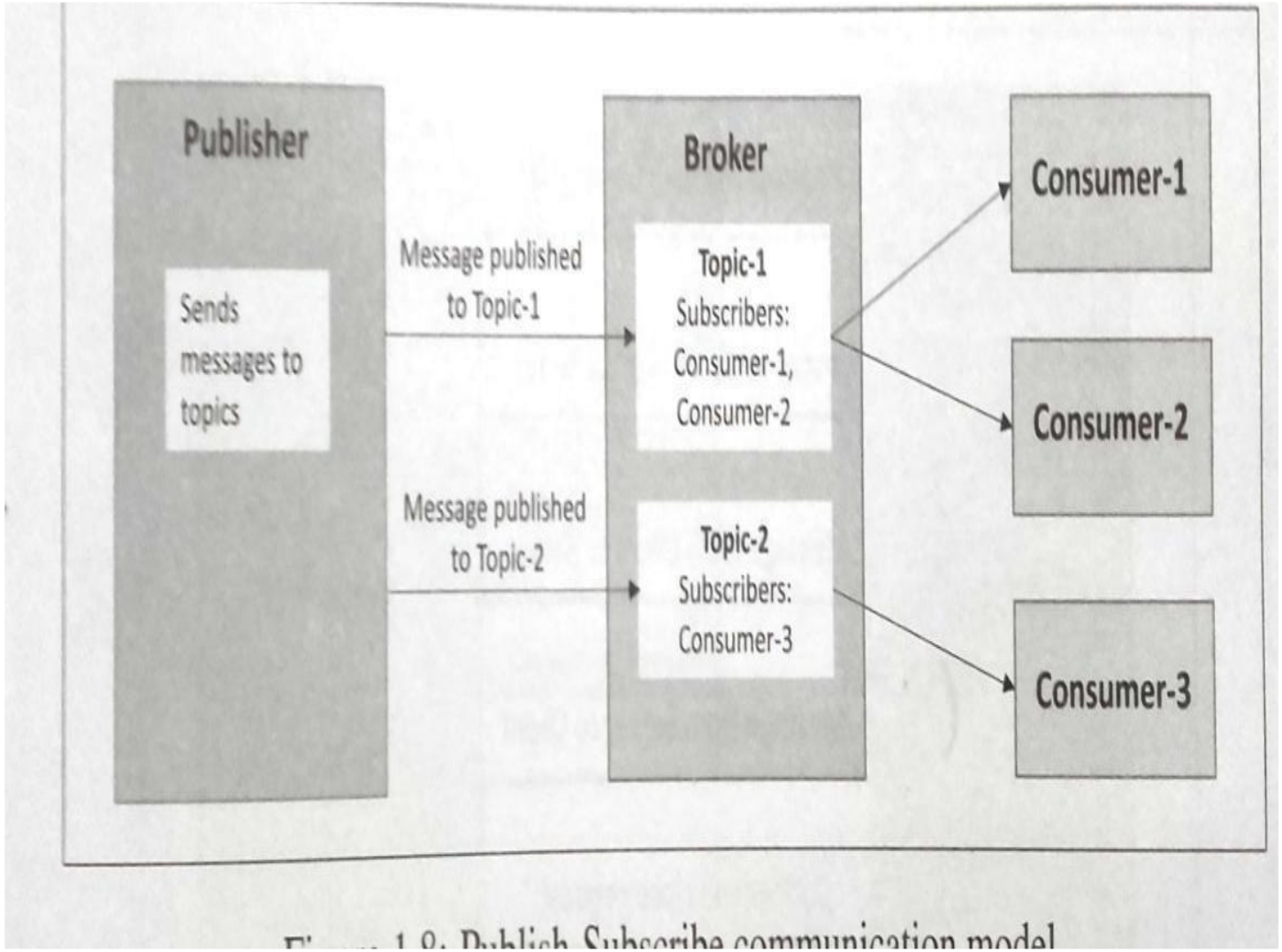
1) Request-Response 2) Publish-Subscibe 3)Push-Pull 4) ExclusivePair

1) Request-Response Model: In which the client sends request to the server and the server replies to requests. Is a stateless communication model and each request-response pair is independent of others.



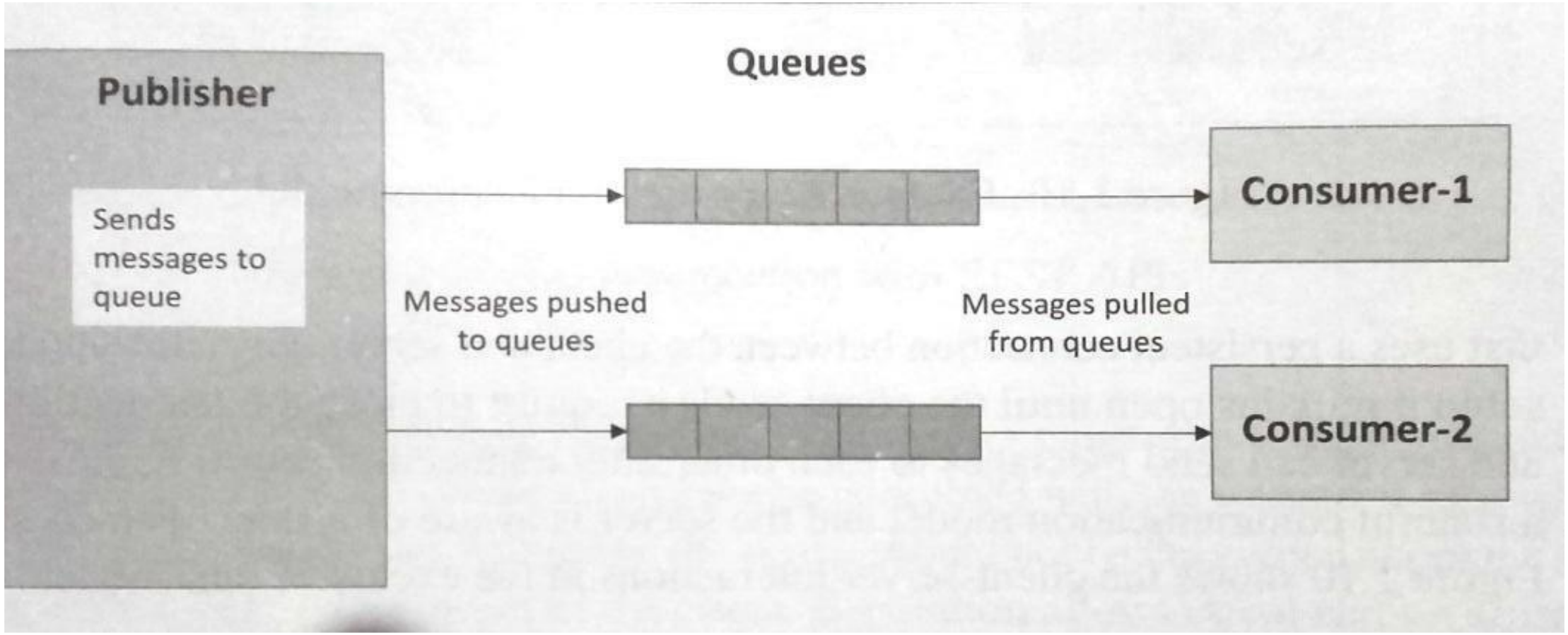
2) Publish-Subscibe Model:

- Involves publishers, brokers and consumers.
- Publishers are source of data. Publishers send 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.



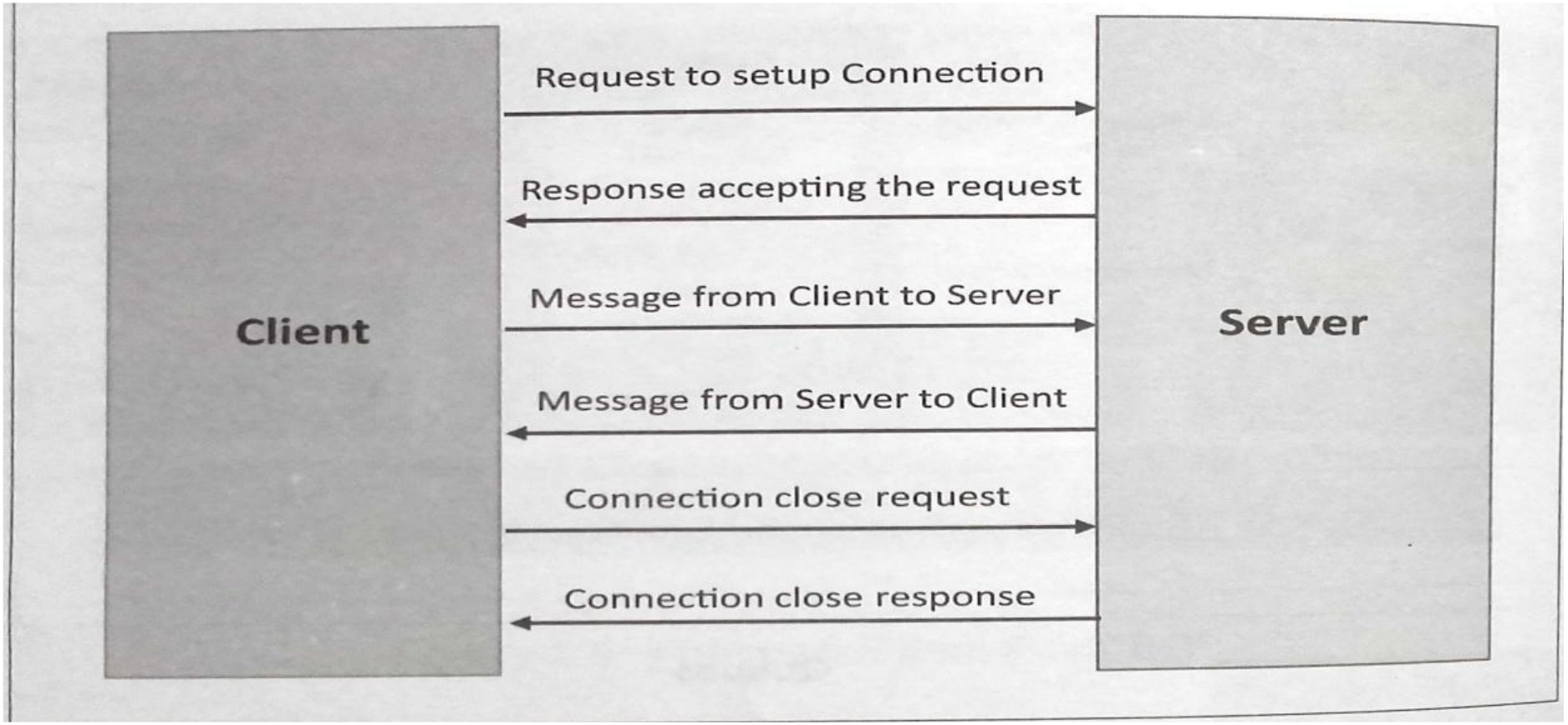
3) Push-Pull Model: in which data producers push data to queues and consumers pull data from the queues.

Producers do not need to aware of the consumers. Queues help in decoupling the message between the producers and consumers.



4) Exclusive Pair:

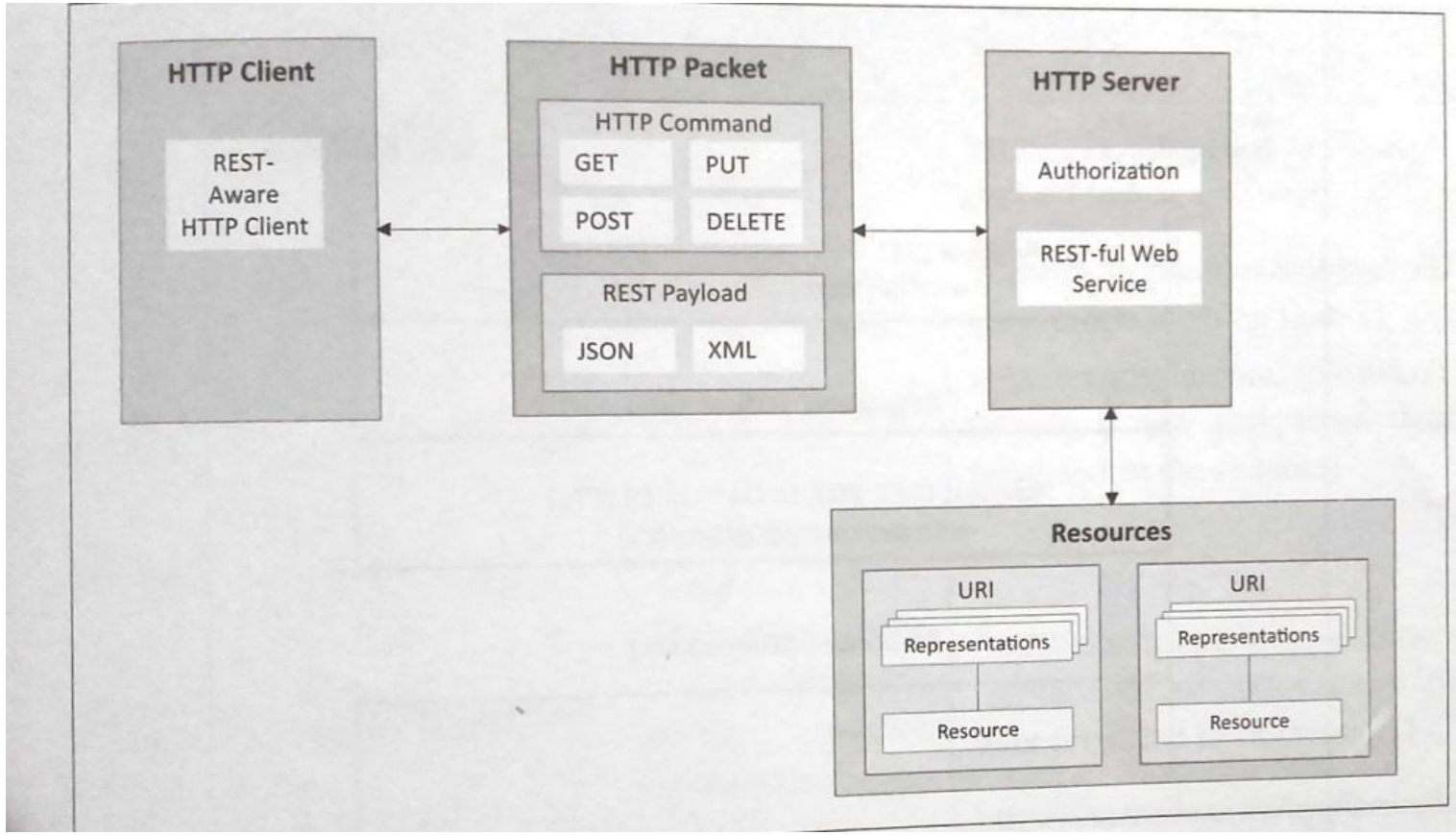
- is bi-directional, fully duplex communication model that uses a persistent connection between the client and server.
- Once connection is set up it remains open until the client send a request to close the connection.
- Is a stateful communication model and server is aware of all the open connections.



3) IoT Communication APIs:

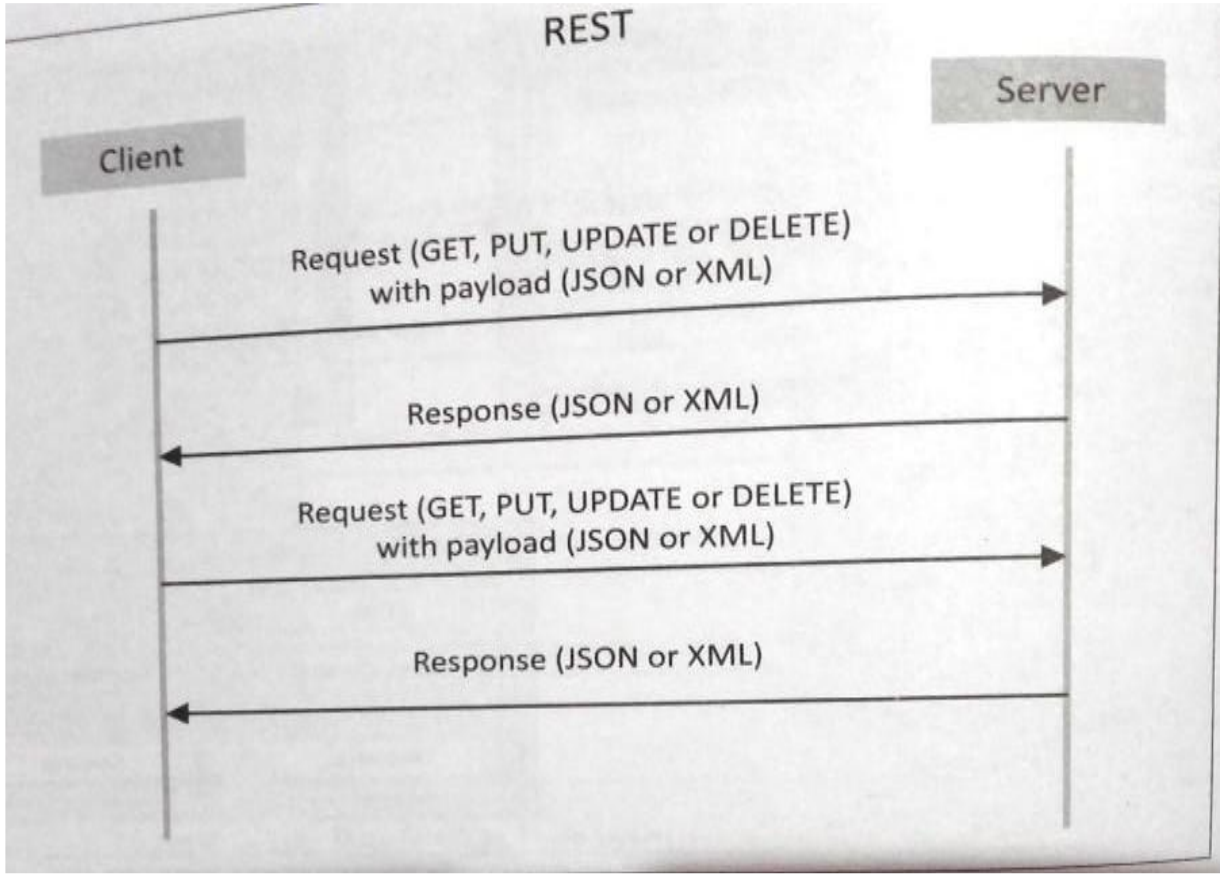
a) REST based communication APIs(Request-Response Based Model) :

- Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a systems resources and have resource states are addressed and transferred.



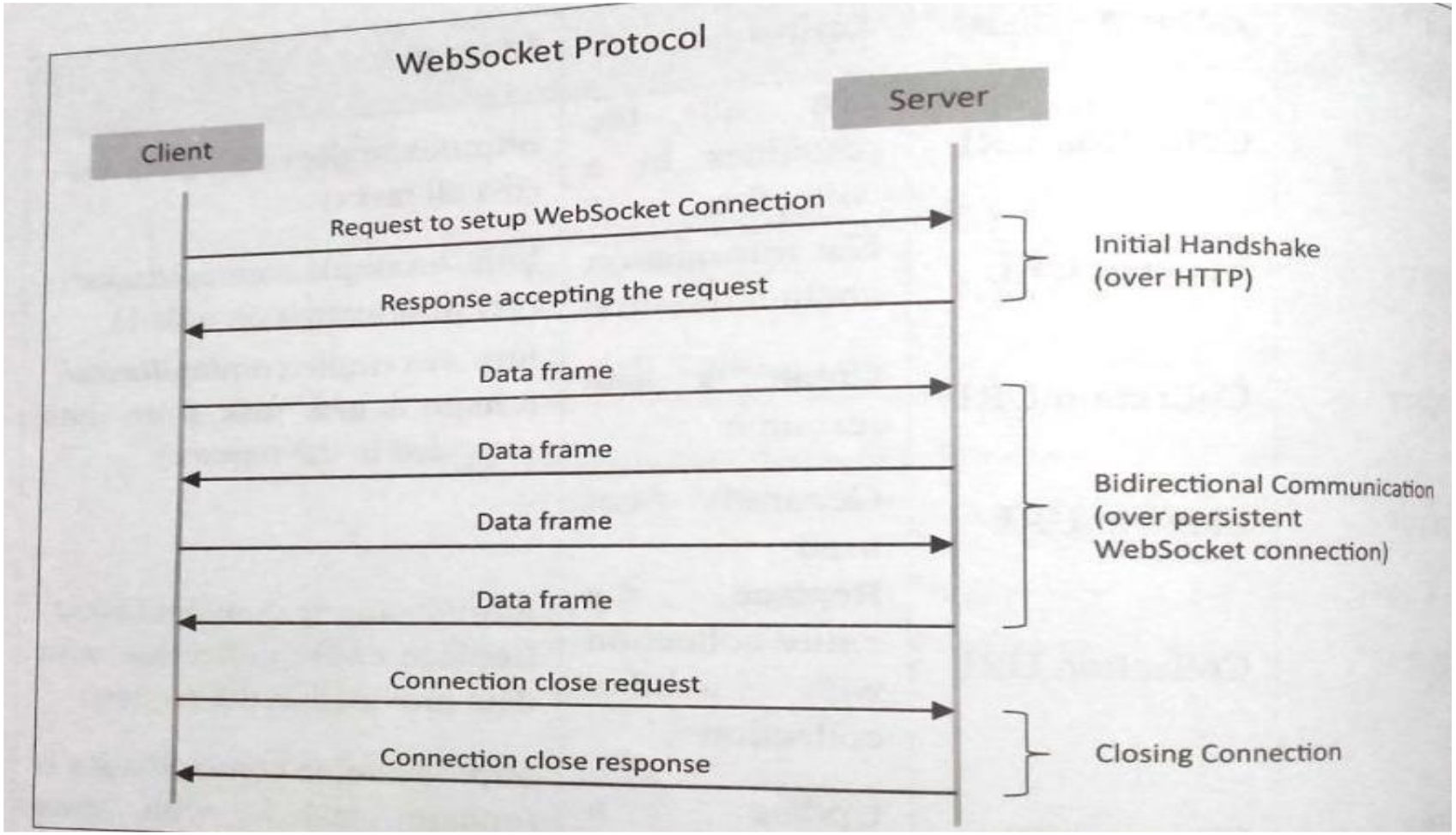
- **Client-Server:** The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.
- **Stateless:** Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- **Cache-able:** Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.
- **Layered System:** constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.
- **User Interface:** constraint requires that the method of communication between a client and a server must be uniform.
- **Code on Demand:** Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Request-Response model used by REST:



- RESTful web service is a collection of resources which are represented by URIs.
- RESTful web API has a base URI(e.g: http://example.com/api/tasks/).
- The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE).
- A RESTful web service can support various internet media types.

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.



IoT Enabling Technologies

- IoT is enabled by several technologies including Wireless Sensor Networks, Cloud Computing, Big Data Analytics, Embedded Systems, Security Protocols and architectures, Communication Protocols, Web Services, Mobile internet and semantic search engines.

1) **Wireless Sensor Network(WSN):** Comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. Zig Bee is one of the most popular wireless technologies used byWSNs.

WSNs used in IoT systems are described as follows:

- Weather Monitoring System: in which nodes collect temp, humidity and other data, which is aggregated and analyzed.
- Indoor air quality monitoring systems: to collect data on the indoor air quality and concentration of various gases.
- Soil Moisture Monitoring Systems: to monitor soil moisture at various locations.
- Surveillance Systems: use WSNs for collecting surveillance data (motion data detection).
- Smart Grids : use WSNs for monitoring grids at various points.
- Structural Health Monitoring Systems: Use WSNs to monitor the health of structures(building, bridges) by collecting vibrations from sensor nodes deployed at various points in the structure.

2) **Cloud Computing:** Services are offered to users in different forms.

- Infrastructure-as-a-service(IaaS):provides users the ability to provision computing and storage resources. These resources are provided to the users as a virtual machine instances and virtual storage.
- Platform-as-a-Service(PaaS): provides users the ability to develop and deploy application in cloud using the development tools, APIs, software libraries and services provided by the cloud service provider.
- Software-as-a-Service(SaaS): provides the user a complete software application or the user interface to the application itself.

3) **Big Data Analytics:** Some examples of big data generated by IoT are

- Sensor data generated by IoT systems.
- Machine sensor data collected from sensors established in industrial and energy systems.
- Health and fitness data generated IoT devices.
- Data generated by IoT systems for location and tracking vehicles.
- Data generated by retail inventory monitoring systems

4) **Communication Protocols:** form the back-bone of IoT systems and enable network connectivity and coupling to applications.

- Allow devices to exchange data over network.
- Define the exchange formats, data encoding addressing schemes for device and routing of packets from source to destination.
- It includes sequence control, flow control and retransmission of lost packets.

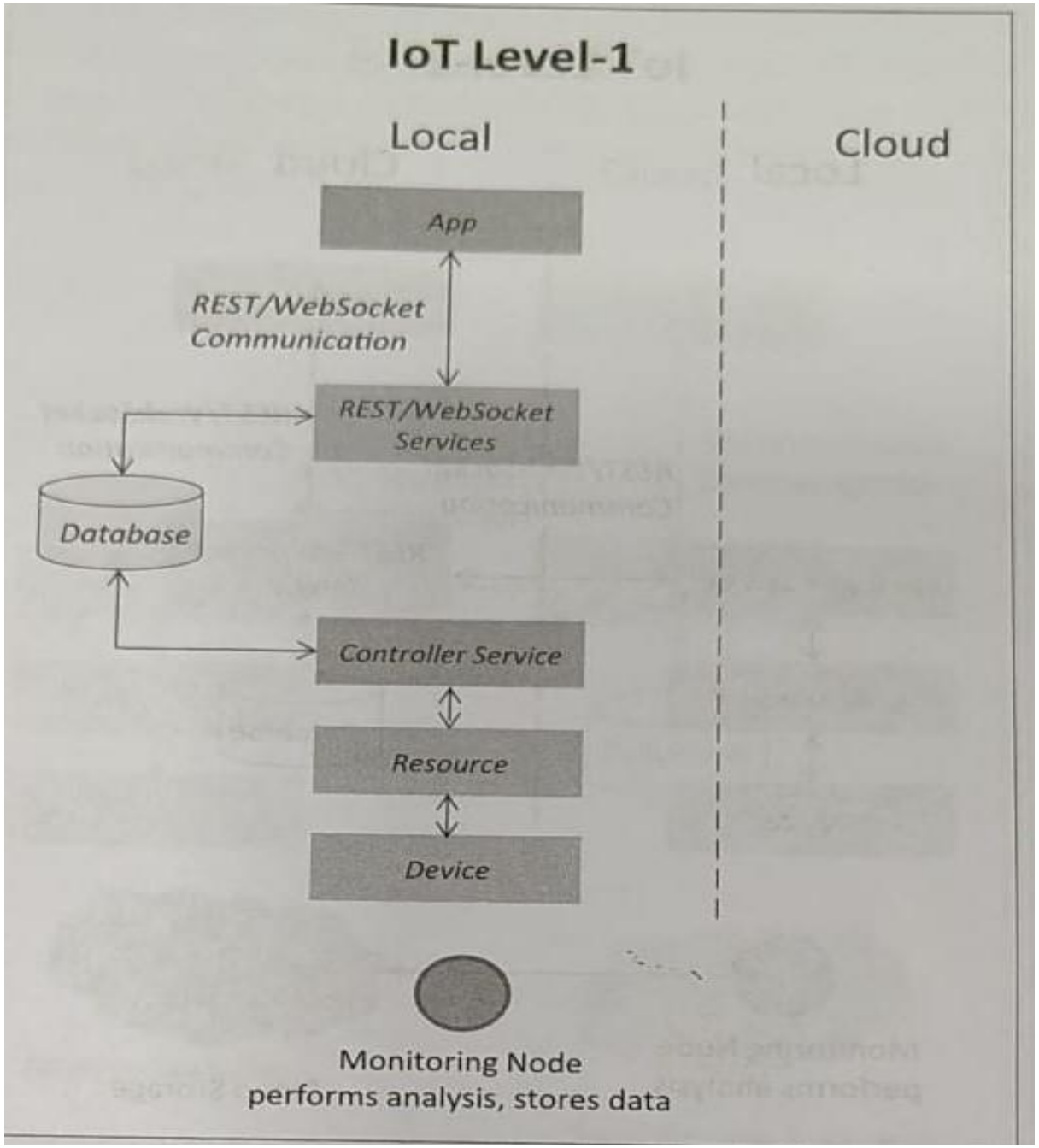
5) **Embedded Systems:** is a computer system that has computer hardware and software embedded to perform specific tasks.

-Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

IoT levels & Deployment Templates

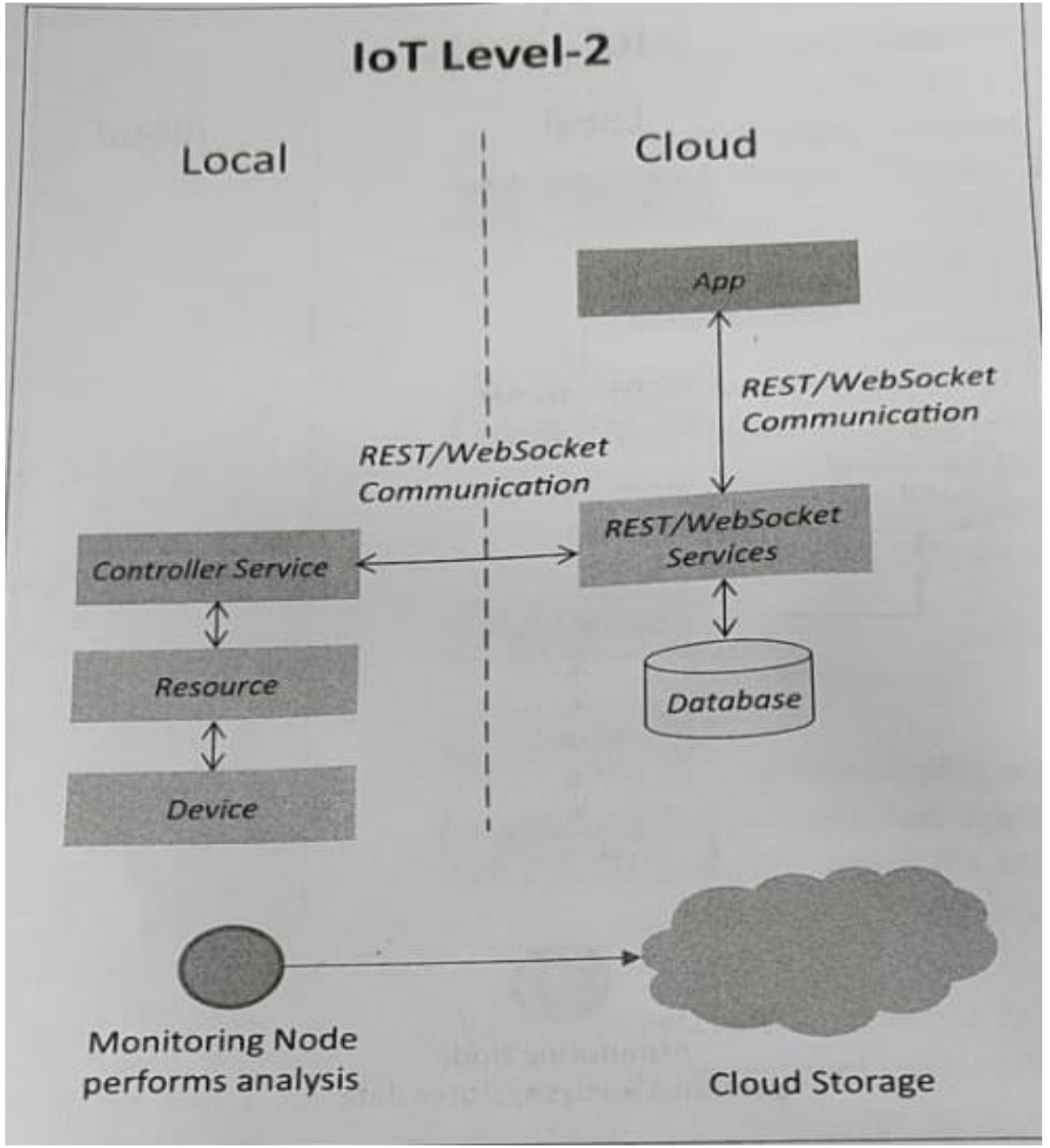
IoT Level-1:

- System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig.
- Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally intensive.
- An e.g., of IoT Level1 is Home automation.



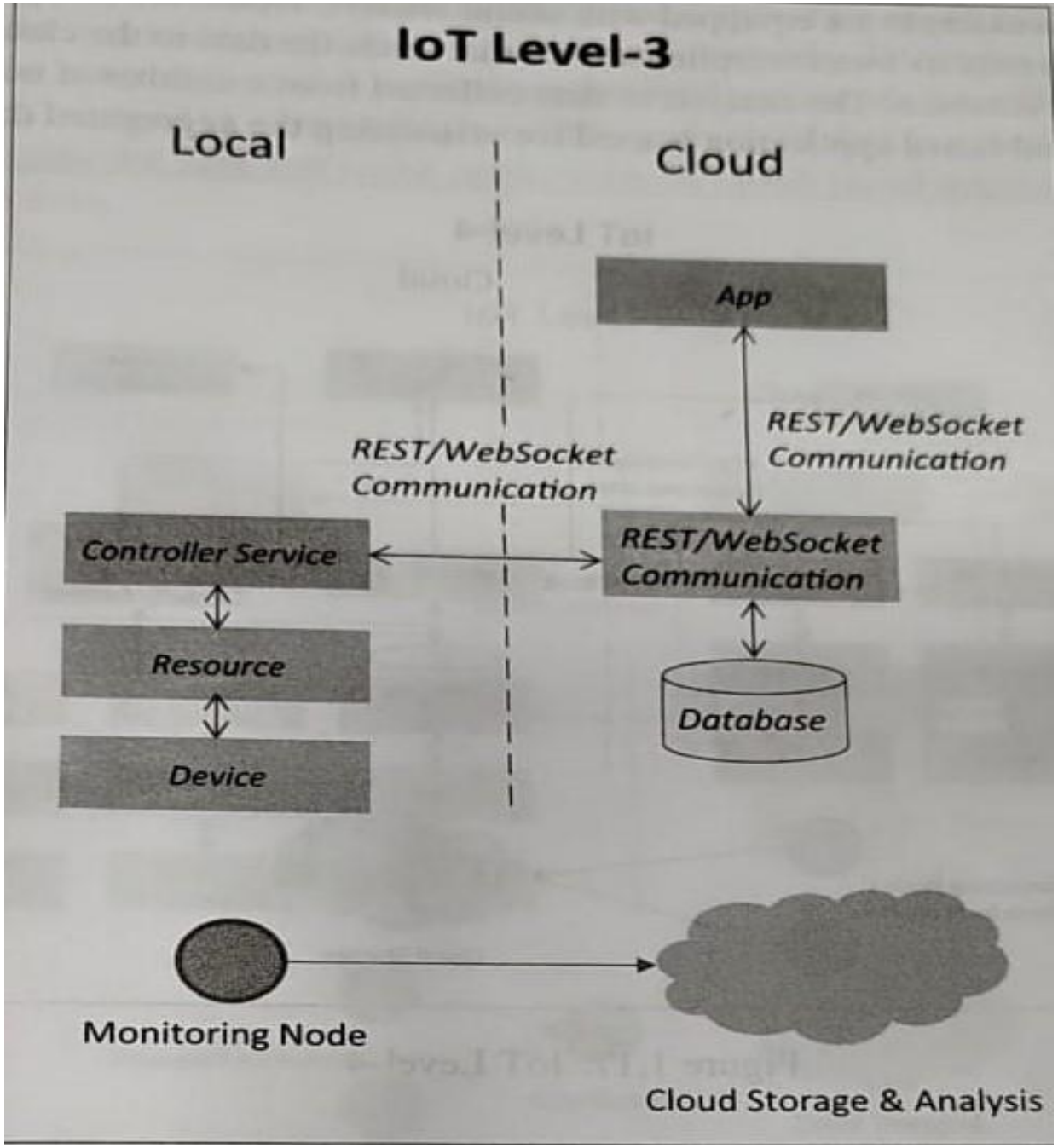
IoT Level2:

- has a single node that performs sensing and/or actuating and local analysis as shown in fig.
- Data is stored in cloud and application is usually cloud based.
- Level2 IoT systems are suitable for solutions where data are involved is big,
- however, the primary analysis requirement is not computationally intensive and can be done locally itself.
- An e,g., of Level2 IoT system for Smart Irrigation.



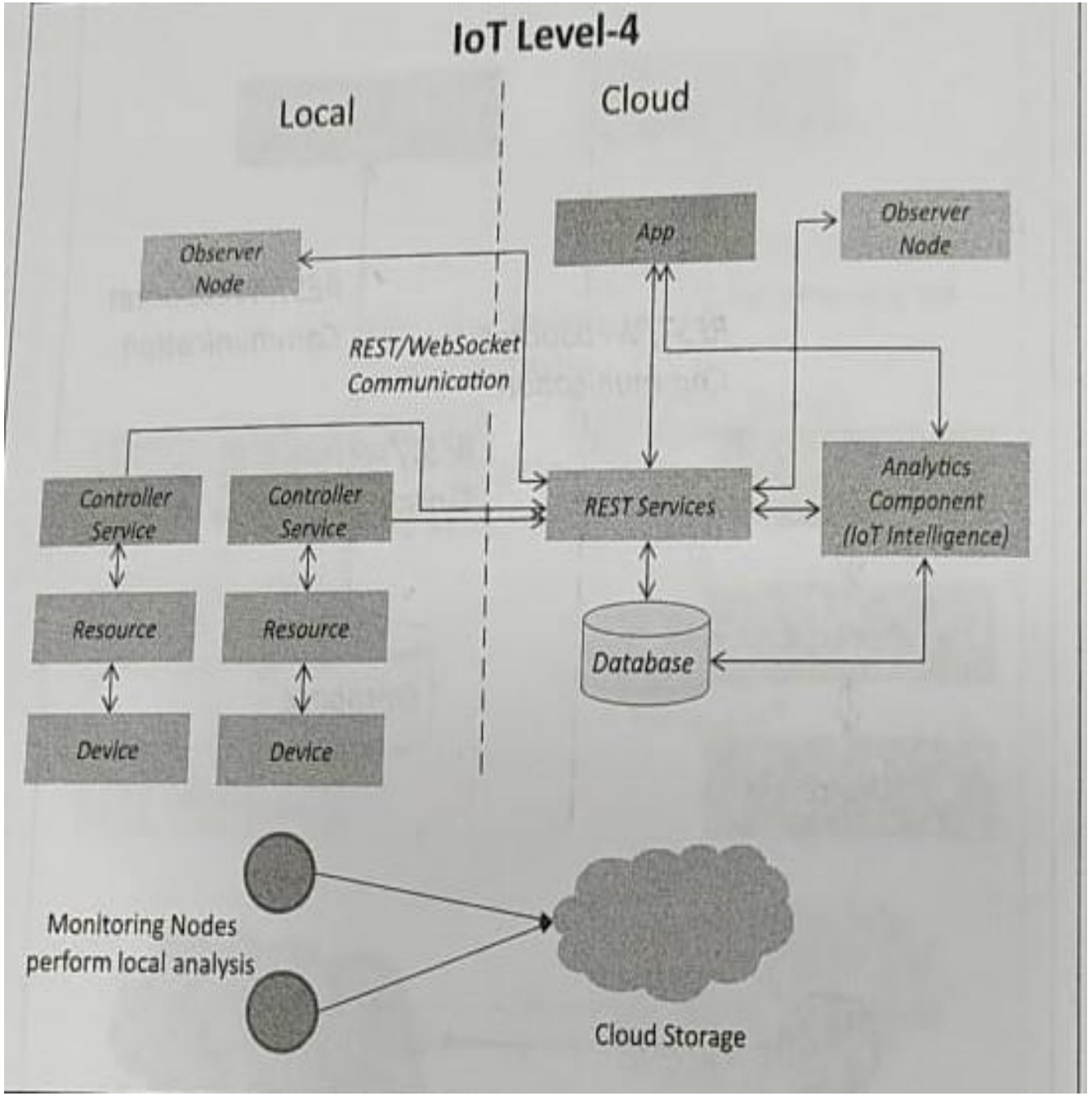
IoT Level3:

- system has a single node.
- Data is stored and analyzed in the cloud application is cloud based as shown in fig. Level3
- IoT systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive.
- An example of IoT level3 system for tracking package handling.



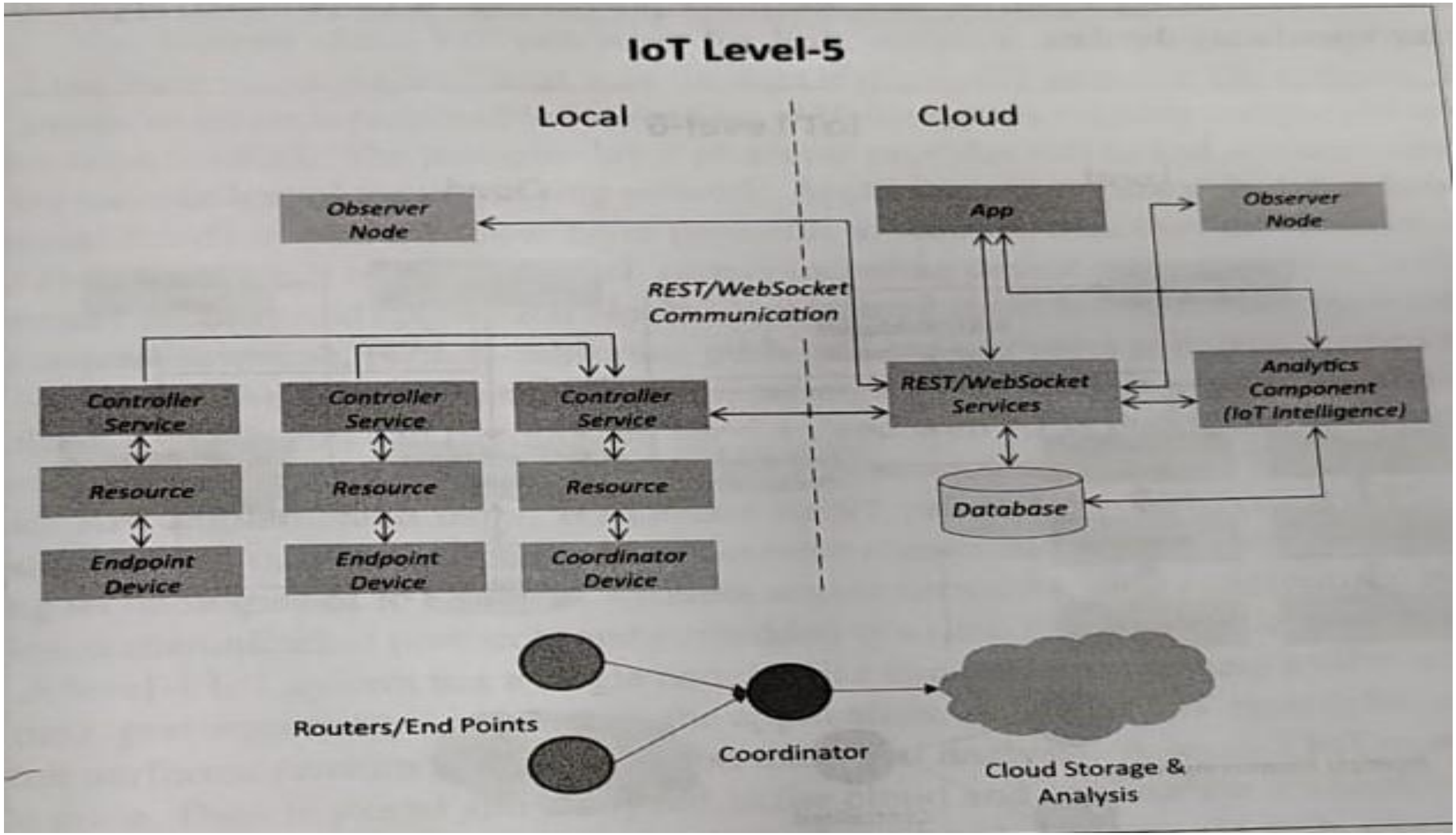
IoT Level4:

- System has multiple nodes that perform local analysis.
- Data is stored in the cloud and application is cloud based as shown in fig. Level4
- contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.
- An example of a Level4 IoT system for Noise Monitoring.



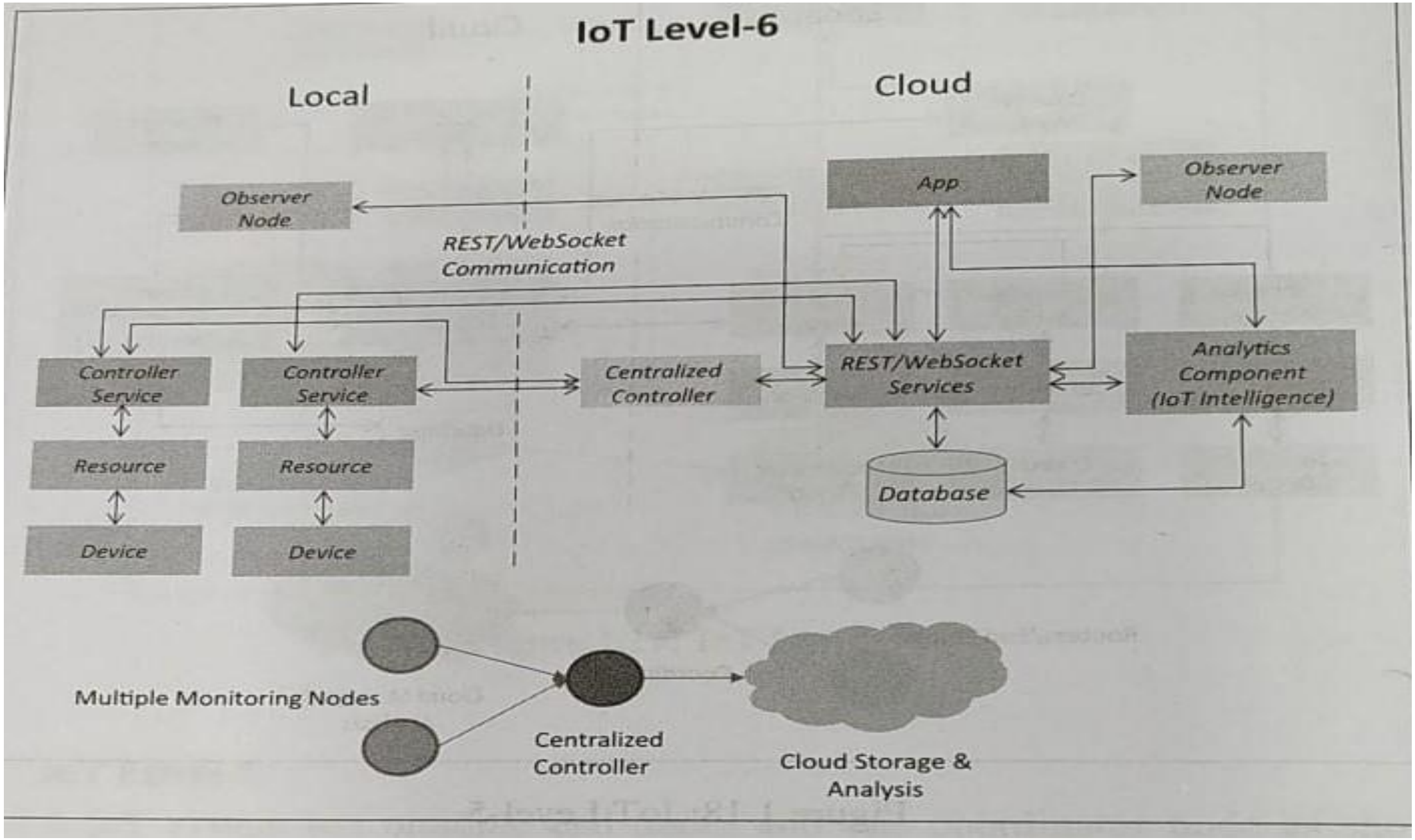
IoT Level5:

- System has multiple end nodes and one coordinator node as shown in fig.
- The end nodes that perform sensing and/or actuation.
- Coordinator node collects data from the end nodes and sends to the cloud.
- Data is stored and analyzed in the cloud and application is cloud based.
- Level5 IoT systems are suitable for solution based on wireless sensor network, in which data involved is big and analysis requirements are computationally intensive.
- An example of Level5 system for Forest Fire Detection.


IoT Level-5
Local
Cloud
Observer Node
App
Observer Node
REST/WebSocket Communication
Controller Service
Controller Service
Controller Service
REST/WebSocket Services
Analytics Component (IoT Intelligence)
Resource
Resource
Resource
Database
Endpoint Device
Endpoint Device
Coordinator Device
Routers/End Points
Coordinator
Cloud Storage & Analysis

IoT Level6:

- System has multiple independent end nodes that perform sensing and/or actuation and sensed data to the cloud. Data is stored in the cloud and application is cloud based as shown in fig.
- The analytics component analyses the data and stores the result in the cloud data base. The results are visualized with cloud based application.
- The centralized controller is aware of the status of all the end nodes and sends control commands to nodes.
- An example of a Level6 IoT system for Weather Monitoring System.


IoT Level-6
Local
Cloud
Observer Node
App
Observer Node
REST/WebSocket Communication
Controller Service
Controller Service
Centralized Controller
REST/WebSocket Services
Analytics Component (IoT Intelligence)
Resource
Resource
Database
Device
Device
Multiple Monitoring Nodes
Centralized Controller
Cloud Storage & Analysis

Domain Specific IoT's

1) Home Automation:

a) Smart Lighting: helps in saving energy by adapting the lighting to the ambient conditions and switching on/off or diming the light when needed.

b) Smart Appliances: make the management easier and also provide status information to the users remotely.

c) Intrusion Detection: use security cameras and sensors(PIR sensors and door sensors) to detect intrusion and raise alerts. Alerts can be in the form of SMS or email sent to the user.

d) Smoke/Gas Detectors: Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire. Alerts raised by smoke detectors can be in the form of signals to a fire alarm system. Gas detectors can detect the presence of harmful gases such as CO, LPGetc.,

2) Cities:

a) Smart Parking: make the search for parking space easier and convenient for drivers. Smart parking are powered by IoT systems that detect the no. of empty parking slots and send information over internet to smart application backends.

b) Smart Lighting: for roads, parks and buildings can help in saving energy.

c) Smart Roads: Equipped with sensors can provide information on driving condition, travel time estimating and alert in case of poor driving conditions, traffic condition and accidents.

d) Structural Health Monitoring: uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.

e) Surveillance: The video feeds from surveillance cameras can be aggregated in cloud based scalable storage solution.

f) Emergency Response: IoT systems for fire detection, gas and water leakage detection can help in generating alerts and minimizing their effects on the critical infrastructures

3. Environment:

a) Weather Monitoring: Systems collect data from a no. of sensors attached and send the data to cloud based applications and storage back ends. The data collected in cloud can then be analyzed and visualized by cloud based applications.

b) Air Pollution Monitoring: System can monitor emission of harmful gases(CO2, CO, NO, NO2 etc.,) by factories and automobiles using gaseous and meteorological sensors. The collected data can be analyzed to make informed decisions on pollutions control approaches.

c) Noise Pollution Monitoring: Due to growing urban development, noise levels in cities have increased and even become alarmingly high in some cities. IoT based noise pollution monitoring systems use a no. of noise monitoring systems that are deployed at different places in a city. The data on noise levels from the station is collected on servers or in the cloud. The collected data is then aggregated to generate noise maps.

d) Forest Fire Detection: Forest fire can cause damage to natural resources, property and human life. Early detection of forest fire can help in minimizing damage.

e) River Flood Detection: River floods can cause damage to natural and human resources and human life. Early warnings of floods can be given by monitoring the water level and flow rate. IoT based river flood monitoring system uses a no. of sensor nodes that monitor the water level and flow rate sensors.

4) Energy:

a) Smart Grids: is a data communication network integrated with the electrical grids that collects and analyze data captured in near-real-time about power transmission, distribution and consumption. Smart grid technology provides predictive information and recommendations to utilities, their suppliers, and their customers on how best to manage power. By using IoT based sensing and measurement technologies, the health of equipment and integrity of the grid can be evaluated.

b) Renewable Energy Systems: IoT based systems integrated with the transformers at the point of interconnection measure the electrical variables and how much power is fed into the grid. For wind energy systems, closed-loop controls can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides power support.

c) Prognostics: In systems such as power grids, real-time information is collected using specialized electrical sensors called Phasor Measurment Units(PMUs) at the substations. The information received from PMUs must be monitored in real-time for estimating the state of the system and for predicting failures.

5) Retail:

a) Inventory Management: IoT systems enable remote monitoring of inventory using data collected by RFID readers.

b) Smart Payments: Solutions such as contact-less payments powered by technologies such as Near Field Communication(NFC) and Bluetooth.

c) Smart Vending Machines: Sensors in a smart vending machines monitors its operations and send the data to cloud which can be used for predictive maintenance.

6) Logistics:

a) Route generation & scheduling: IoT based system backed by cloud can provide first response to the route generation queries and can be scaled upto serve a large transportation network.

b) Fleet Tracking: Use GPS to track locations of vehicles inreal-time.

c) Shipment Monitoring: IoT based shipment monitoring systems use sensors such as temp, humidity, to monitor the conditions and send data to cloud, where it can be analyzed to detect foods poilage.

d) Remote Vehicle Diagnostics: Systems use on-board IoT devices for collecting data on Vehicle operations(speed, RPMetc.,) and status of various vehicle subsystems.

7.Agriculture:

a) Smart Irrigation: to determine moisture amount in soil.
b) Green House Control: to improve productivity.

8. Industry:

a) Machine diagnosis and prognosis
b) Indoor Air Quality Monitoring

9. Health and LifeStyle:

a) Health & Fitness Monitoring
b) Wearable Electronics

The End

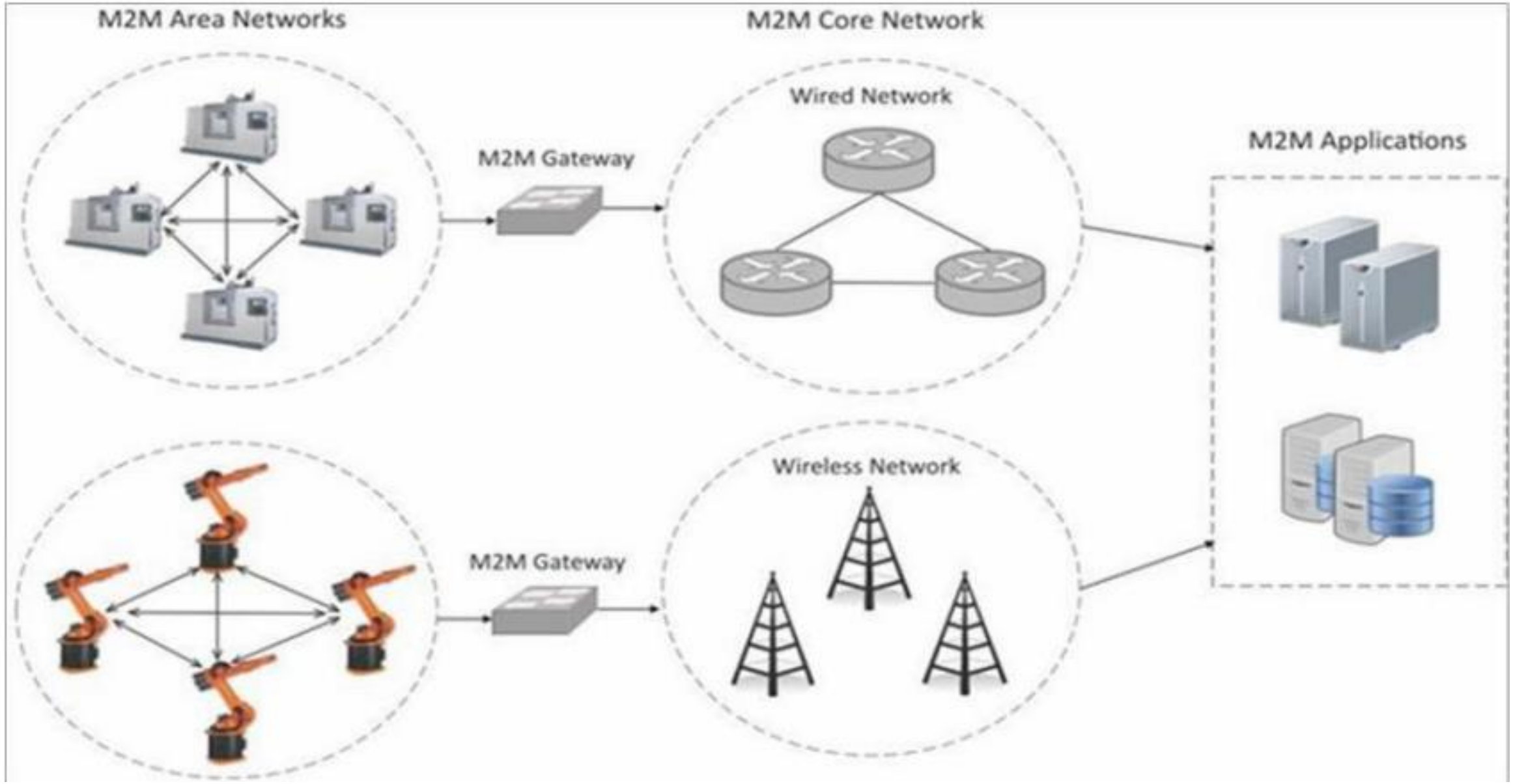
Unit-2
IOT AND M2M

Introduction –M2M, Difference between IoT and M2M, SDN and NFV for IoT

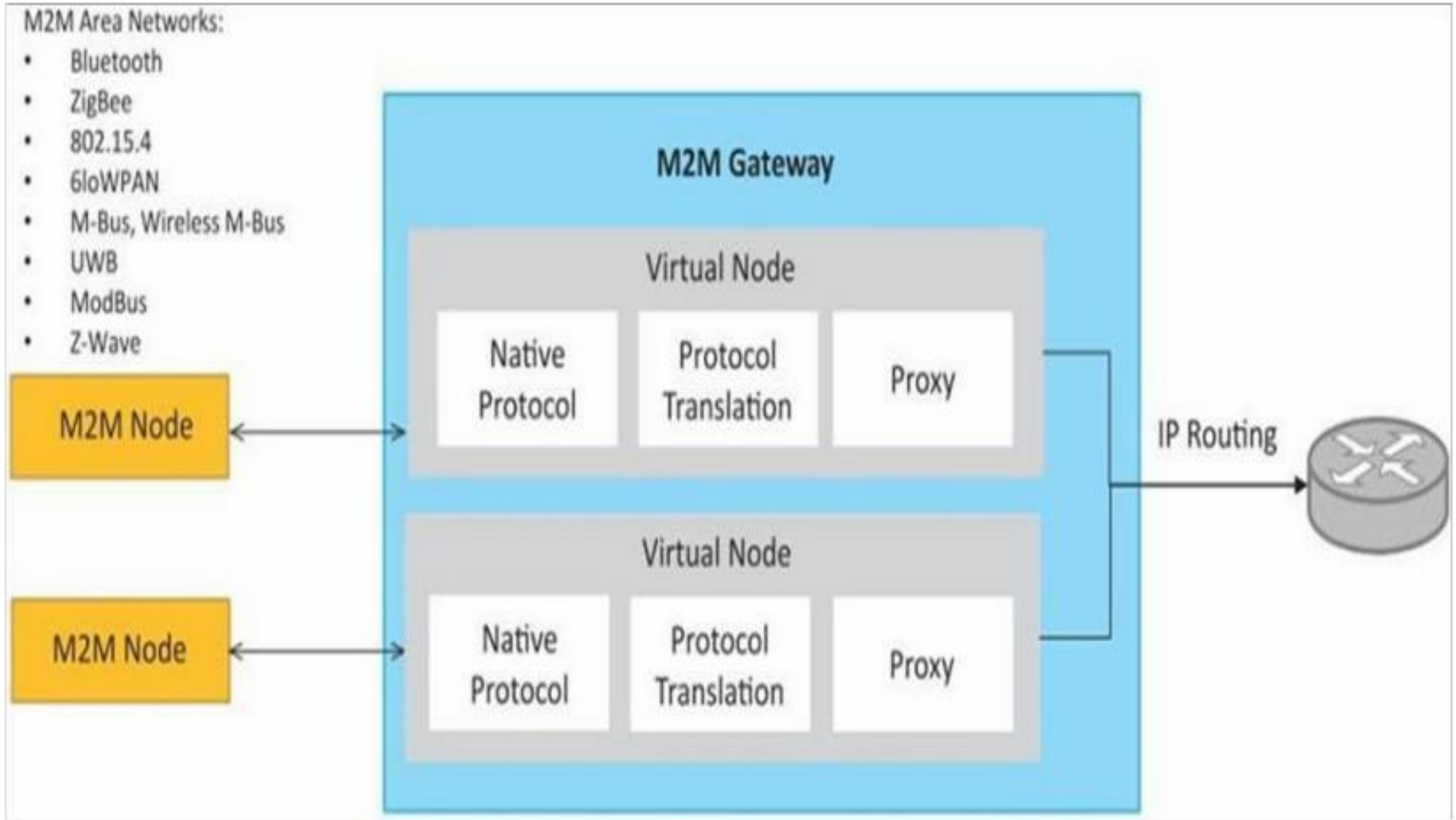
IoT System management with NETCONF-YANG: Need for IoT Systems Management –Simple network Management protocol (SNMP)–Network operator requirements, NETCONF, YANG, IOT systems management with NETCONF, YANG–NETOPEER.

Introduction: M2M

- Machine-to-Machine (M2M) refers to networking of machines(or devices) for the purpose of remote monitoring and control and data exchange.
- Term which is often synonymous with IoT is Machine-to-Machine (M2M).
- IoT and M2M are often used interchangeably.
- Fig. Shows the end-to-end architecture of M2M systems comprises of M2M area networks, communication networks and application domain.


M2M Area Networks
M2M Core Network
Wired Network
M2M Gateway
M2M Applications
Wireless Network
M2M Gateway

- An M2M area network comprises of machines(or M2M nodes) which have embedded network modules for sensing, actuation and communicating various communication protocols can be used for M2M LAN such as ZigBee, Bluetooth, M-bus, Wireless M-Bus etc., These protocols provide connectivity between M2M nodes within an M2M area network.
- The communication network provides connectivity to remote M2M area networks.
- The communication network can use either wired or wireless network(IP based).
- While the M2M are networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based network.
- Since non-IP based protocols are used within M2M area network, the M2M nodes within one network cannot communicate with nodes in an external network.
- To enable the communication between remote M2M are network, M2M gateways are used.


M2M Area Networks:
Bluetooth
ZigBee
802.15.4
6loWPAN
M-Bus, Wireless M-Bus
UWB
ModBus
Z-Wave
M2M Node
M2M Node
M2M Gateway
Virtual Node
Native Protocol
Protocol Translation
Proxy
Virtual Node
Native Protocol
Protocol Translation
Proxy
IP Routing

- Fig. Shows a block diagram of an M2M gateway.
- The communication between M2M nodes and the M2M gateway is based on the communication protocols which are simple to the M2M are network.
- M2M gateway performs protocol translations to enable IP-connectivity for M2M are networks.
- M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol(IP).
- With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

Difference Between M2M and IoT

1) Communication Protocols:

- Commonly uses M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus tec.,
- In IoT uses HTTP, CoAP, WebSocket , MQTT ,XMPP ,DDS ,AMQP etc.,

2) Machines in M2M Vs Things in IoT:

- Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.

3) Hardware Vs Software Emphasis:

- the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.

4) Data Collection &Analysis

- M2M data is collected in point solutions and often in on-premises storage infrastructure.
- The data in IoT is collected in the cloud (can be public, private or hybrid cloud).

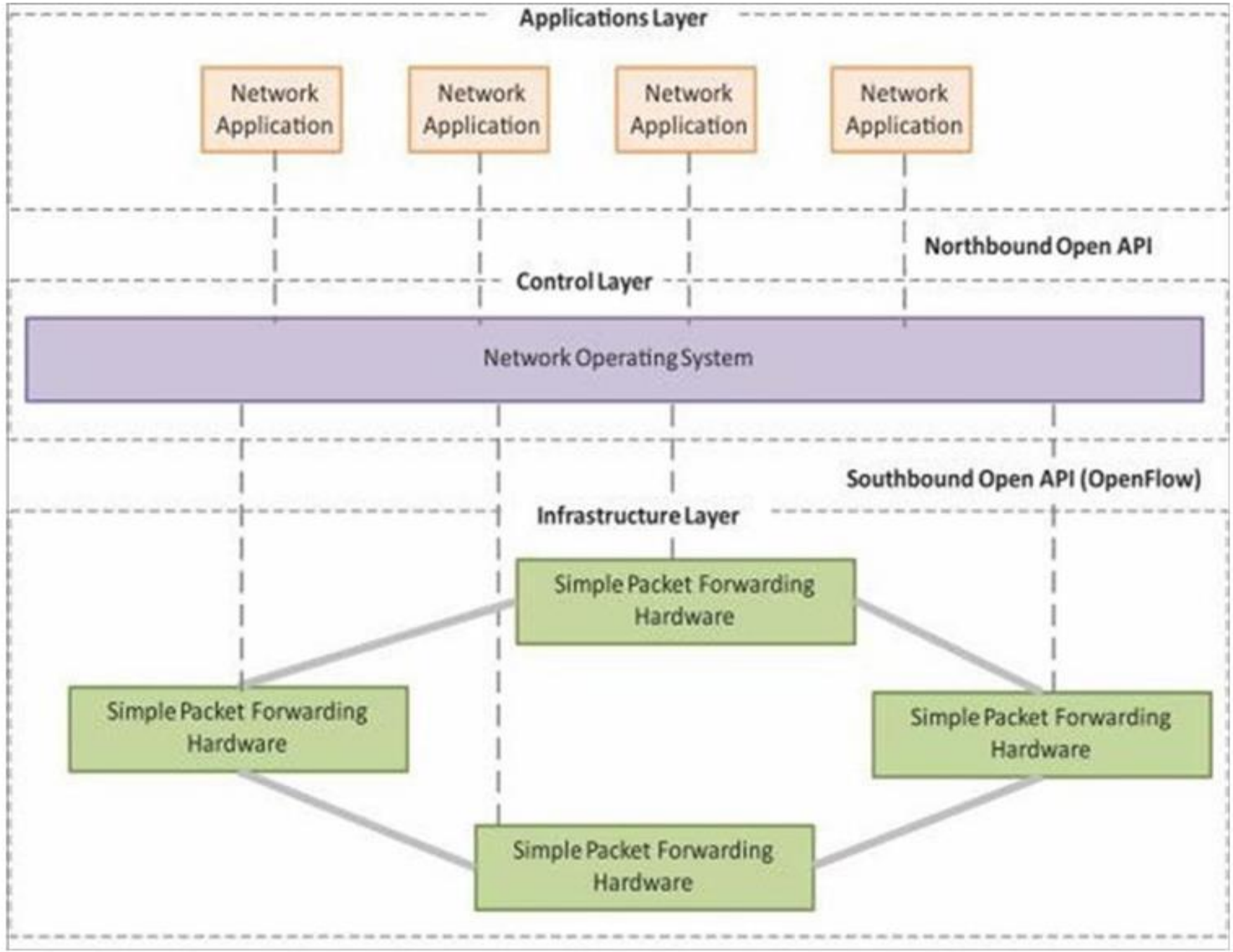
5) Application:

- M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premises enterprise applications.
- IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

SDN and NFV for IoT

Software Defined Networking(SDN):

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

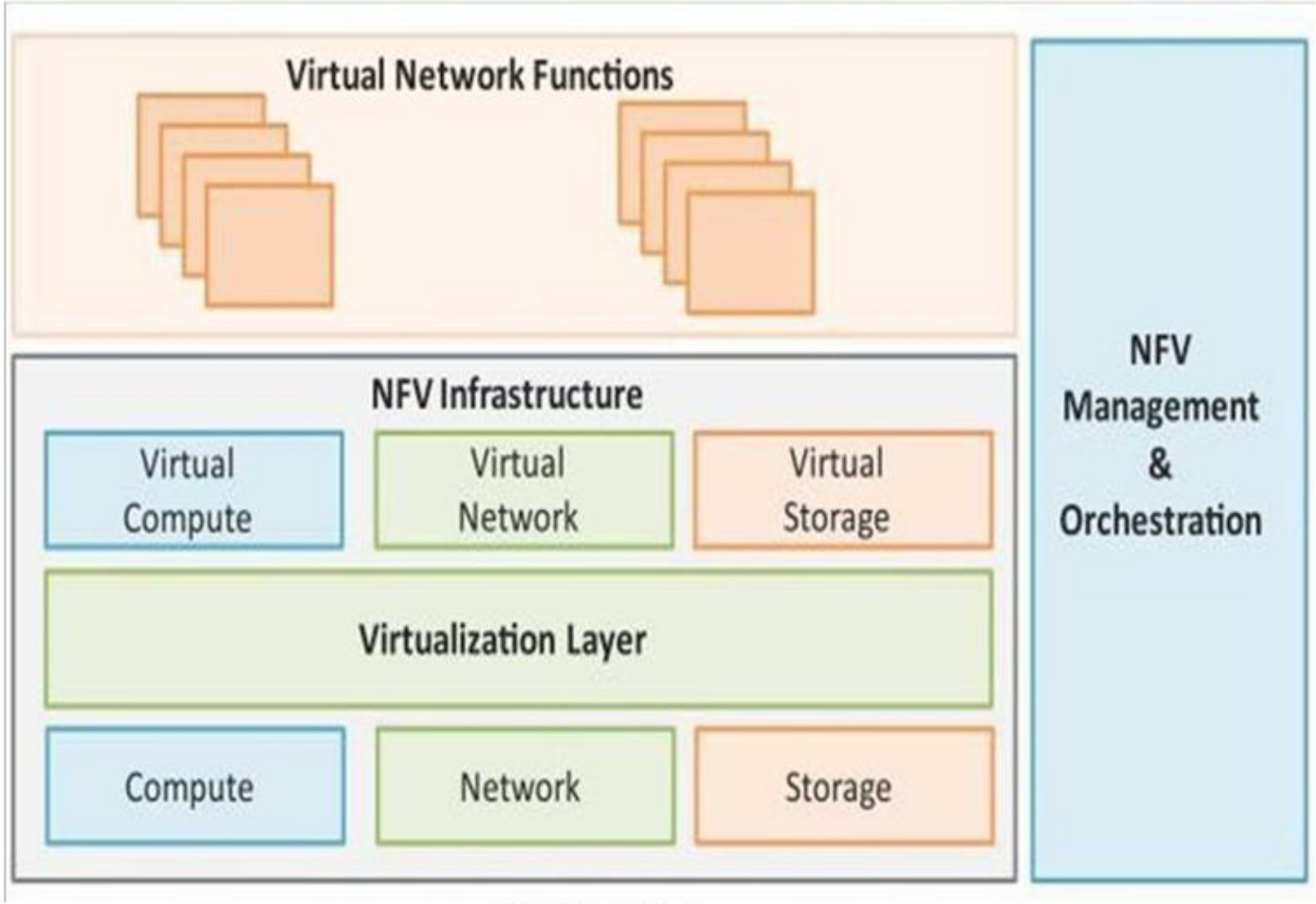

Applications Layer
Network Application
Network Application
Network Application
Network Application
Northbound Open API
Control Layer
Network Operating System
Southbound Open API (OpenFlow)
Infrastructure Layer
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware

- Key elements of SDN:

1) Centralized Network: Controller With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.
2) Programmable Open APIs: SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).
3) Standard Communication Interface(OpenFlow): SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.

Network Function Virtualization(NFV):

- Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.
- NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.



NFV Architecture

Key elements of NFV:

1) Virtualized Network Function(VNF): VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
2) NFV Infrastructure(NFVI): NFVI includes compute, network and storage resources that are virtualized.
3) NFV Management and Orchestration: NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

Need for IoT Systems Management

Need for IoT Systems Management Managing multiple devices within a single system requires advanced management capabilities.

1. Automating Configuration : IoT system management capabilities can help in automating the system configuration.
2. Monitoring Operational & Statistical Data : Management systems can help in monitoring operational and statistical data of a system. This data can be used for fault diagnosis or prognosis.
3. Improved Reliability: A management system that allows validating the system configurations before they are put into effect can help in improving the system reliability.

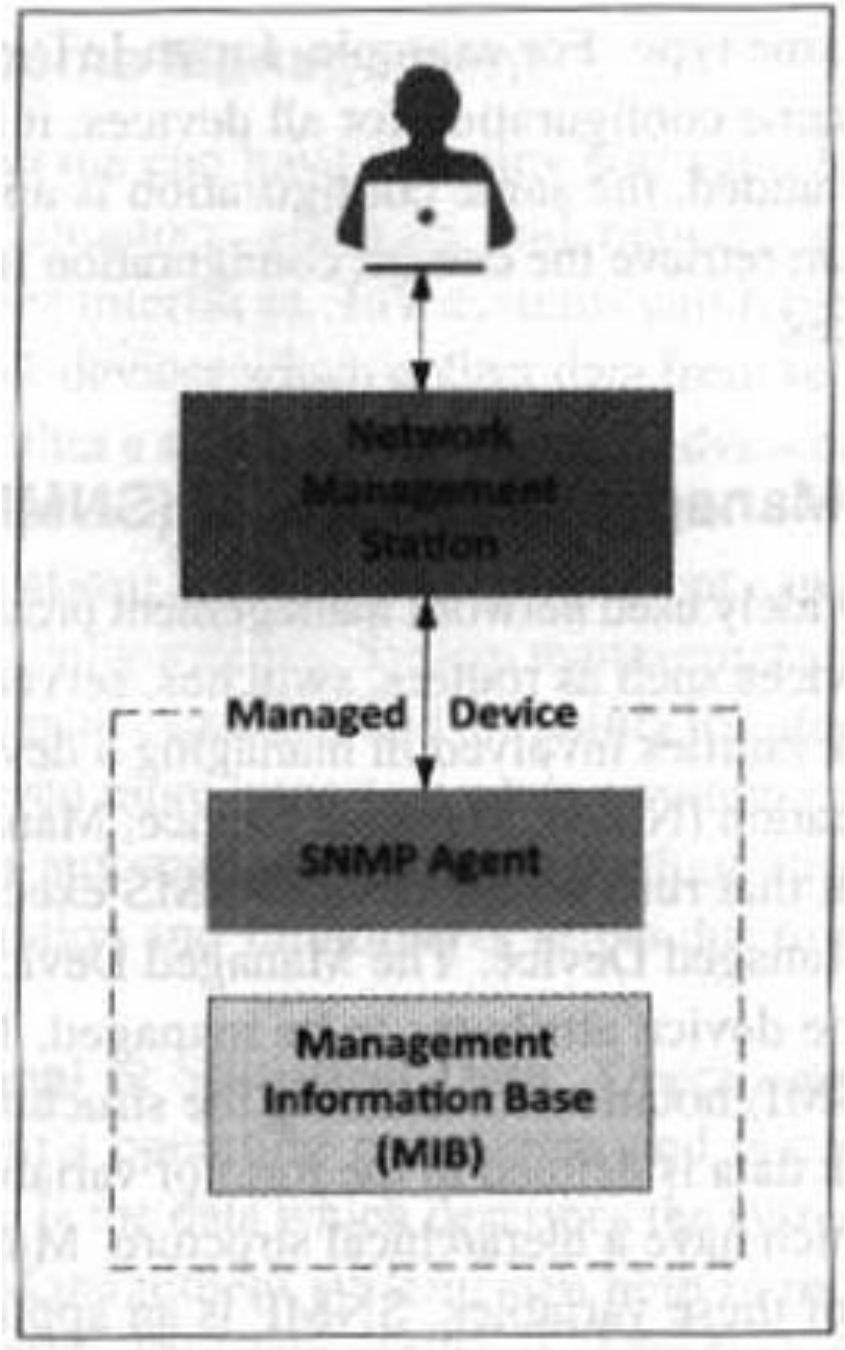
4. System Wide Configurations : For IoT systems that consists of multiple devices or nodes, ensuring system wide configuration can be critical for the correct functioning of the system.

5. Multiple System Configurations : For some systems it may be desirable to have multiple valid configurations which are applied at different times or in certain conditions.

6. Retrieving & Reusing Configurations : Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for other devices of the same type.

Simple network Management protocol (SNMP)

- SNMP is a well-known and widely used network management protocol.
- Using to monitor and configuring network such as routers, switches, servers, printer, etc.,
- NMS (Network Management Station): execute SNMP commands to monitor and configure the Management device.
- MIB (Management Information Base): which has all the information of the device attributes to be managed.



Limitation of SNMP:

- SNMP was designed to provide a simple management interface between the management applications and the managed devices.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.

- MIB often lack writable objects without which device configuration is not possible using SNMP. With the absence of writable object, SNMP can be used only for device monitoring and status polling.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP, SNMP does not support easy retrival and playback of configurations.
- Earlier version of SNMP did not have strong security features making the management information vulnerable to network intruders. Through security features were added in the later version of SNMP, it increased the complexity a lot.

Network Operator Requirements

- Ease of use: From the operators point of view, ease of use is the key requirement for any network management technology.
- Distinction between configuration and state data: Configuration data is the set of writable data that is required to transform the system from its initial state to its current state. State data is the data which is not configurable. State data includes operational data which is collected by the system at runtime and statistical data which describes the system performance. For an effective management solution, it is important to make a clear distinction between configuration and state data.
- Fetch configuration and state data separately: it should be possible to fetch the configuration and state data separately from the management device. This is useful when the configuration and state data from different devices needs to be compared.

- Configuration of the network as a whole: It should be possible for operators to configure the network as a whole rather than individual devices. This is important for systems which have multiple devices and configuring them within one network wide transaction is required to ensure the correct operation of the system.
- Configuration transactions across devices: Configuration transactions across multiple device should be supported.
- Configuration deltas: It should be possible to generate the operations necessary for going from one configuration state to another. The devices should support configuration deltas with minimal state changes.
- Dump and restore configurations: It should be possible to dump configurations from devices and restore configurations to devices.
- Configuration validation: It should be possible to validate configuration.
- Configuration database schemas: There is a need for standardized configuration data base schemas or data models across operators.

- Role-based access control: Devices should support role-based access control model. So that a user is given the minimum access necessary to perform a required task.
- Consistency configurations: Devices should not arbitrarily reorder data, so that it is possible to use text processing tools such as *diff* to compare configurations.
- Consistency of access control lists: It should be possible to do consistency checks of access control lists across devices.
- Multiple configuration sets: There should be support for multiple configurations sets on devices. This way a distinction can be provided between candidate and active configurations.
- Support for both data oriented and task oriented access control: While SNMP access control is data-oridented. CLI access control is usually task oriented. There should be support for both types of access control.

NETCONF

- NETCONF: 'Network Configuration Protocol' is a session-based network management protocol.
- NETCONF allows state or configuration data and manipulating configuration data on network devices.
- For network management architecture based on NETCONF, the terms client and management system and the terms server and device are often used interchangeably (shown in Figure).
- NETCONF works on SSH transport protocol. In addition to Secure Shell Transport Layer Protocol (SSH), NETCONF implementations can support other transport mappings such as Blocks Extensible Exchange Protocol (BEEP).
- Transport layer provides end-to-end connectivity and ensure reliable delivery of messages.
- NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.
- The RPC layer provides mechanism for encoding of RPC calls and notifications.

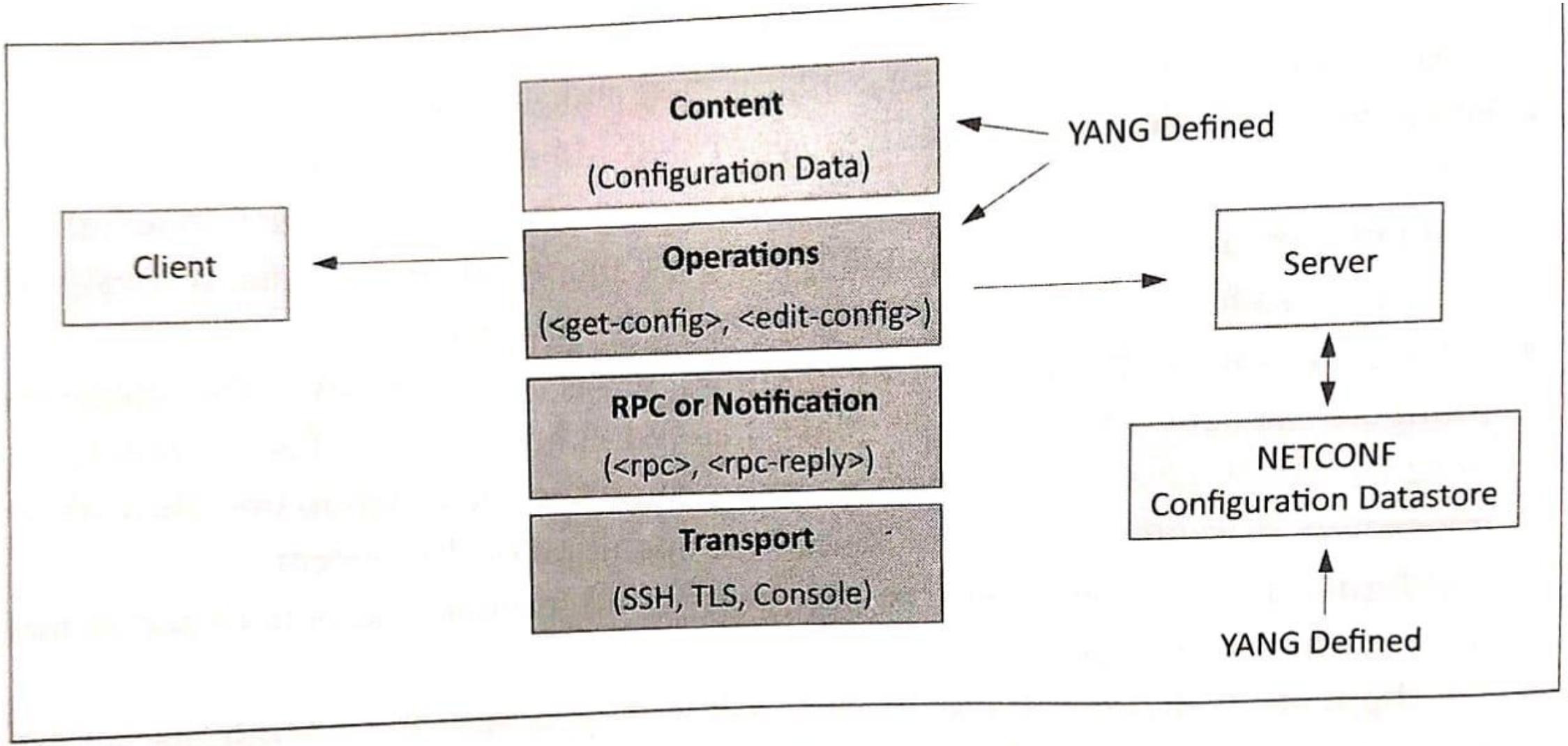


Figure 4.2: NETCONF protocol layers

Operation	Description
connect	Connect to a NETCONF server
get	Retrieve the running configuration and state information
get-config	Retrieve all or a portion of a configuration datastore
edit-config	Loads all or part of a specified configuration to the specified target configuration
copy-config	Create or replace an entire target configuration datastore with a complete source configuration
delete-config	Delete the contents of a configuration datastore
lock	Lock a configuration datastore for exclusive edits by a client
unlock	Release the lock on a configuration datastore
get-schema	This operation is used to retrieve a schema from the NETCONF server
commit	Commit the candidate configuration as the device's new current configuration
close-session	Gracefully terminate a NETCONF session
kill-session	Forcefully terminate a NETCONF session

Table 4.1: List of commonly used NETCONF RPC methods

- NETCONF provides various operations to retrieve and edit configuration data from network devices.
- The Content Layer consists of configuration and state data which is XML-encoded.
- The schema of the configuration and state data is defined in a data modeling language called YANG.
- NETCONF provides a clear separation of the configuration and state data.
- The configuration data resides within a NETCONF configuration datastore on the server. The NETCONF server resides on the network device.

- When a session is established the client and server exchange 'hello" messages which contain information on their capabilities.
- Client can then send multiple requests to the server for retrieving or editing the configuration data.
- NETCONF allows the management client to discover the capabilities of the server (on the device).
- NETCONF gives access to the native capabilities of the device. NETCONF defines one or more configuration datastores.
- A configuration store contains all the configuration information to bring the device from its initial state to the operational state.
- NETCONF is a connection oriented protocol and NETCONF connection persists between protocol operations.
- NETCONF overcomes the limitations of SNMP and is suitable not only for monitoring state information, but also for configuration management.

YANG

- YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol.
- YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.
- YANG modules defines the data exchanged between the NETCONF client and server.
- YANG modules having various node types, they are, (shown in table)

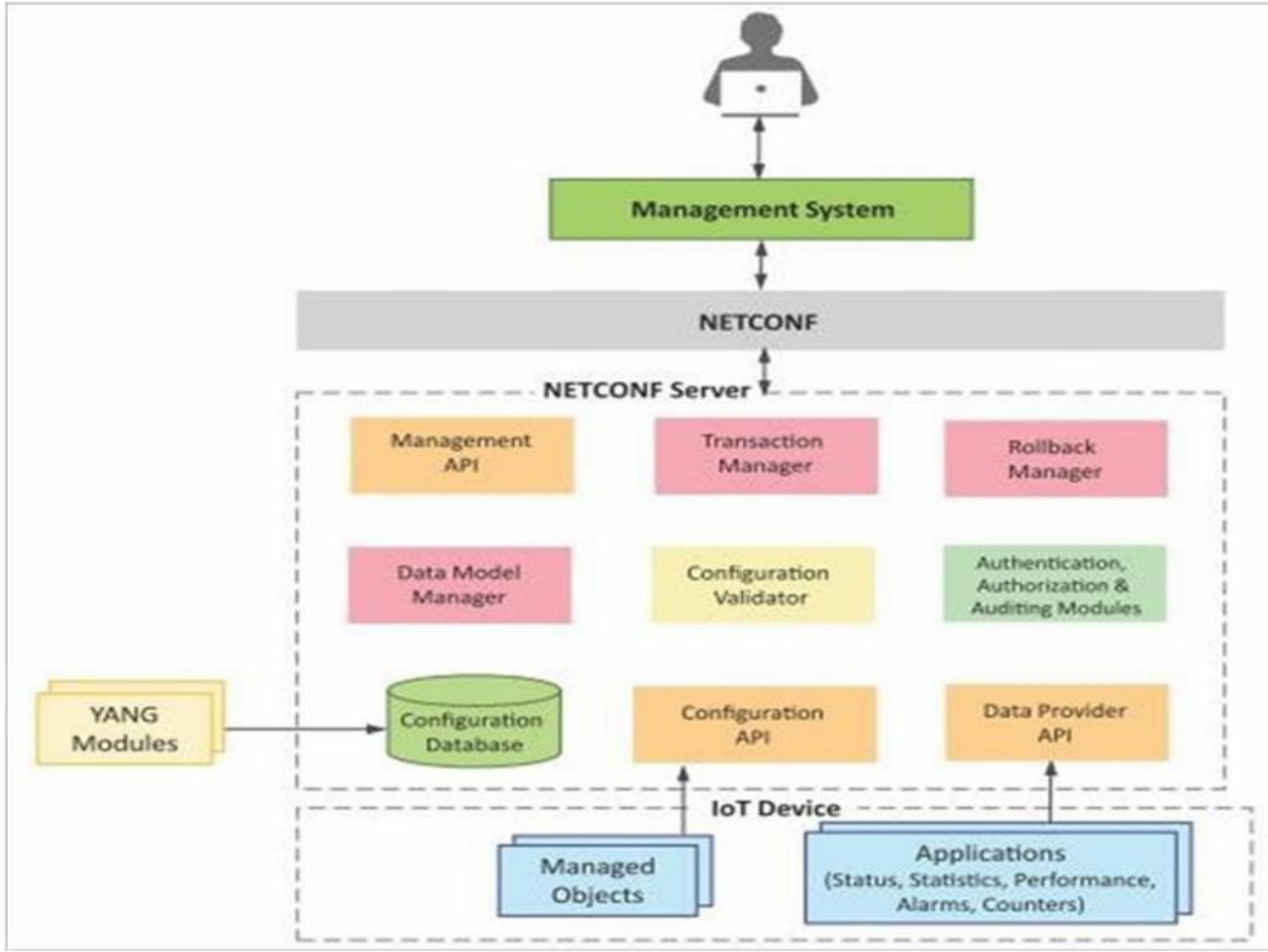
Node Type	Description
Leaf Nodes	Contains simple data structures such as an integer or a string. Leaf has exactly one value of a particular type and no child nodes.
Leaf-List Nodes	Is a sequence of leaf nodes with exactly one value of a particular type per leaf.
Container Nodes	Used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).
List Nodes	Defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely identified by the values of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type.

Table 4.2: YANG Node Types

IoT System management with NETCONF, YANG

- YANG is a data modelling language used to model configuration and state data manipulated by the NETCONF protocol.
- The generic approach of IoT device management with NETCONF-YANG. Roles of various components are:

1) Management System
2) Management API
3) Transaction Manager
4) Rollback Manager
5) Data Model Manager
6) Configuration Validator
7) Configuration Database
8) Configuration API
9) Data Provider API


Management System
NETCONF
NETCONF Server
Management API
Transaction Manager
Rollback Manager
Data Model Manager
Configuration Validator
Authentication, Authorization & Auditing Modules
YANG Modules
Configuration Database
Configuration API
Data Provider API
IoT Device
Managed Objects
Applications
(Status, Statistics, Performance, Alarms, Counters)

1) Management System : The operator uses a management system to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
2) Management API : allows management application to start NETCONF sessions.
3) Transaction Manager: executes all the NETCONF transactions and ensures that ACID properties hold true for the transactions.
4) Rollback Manager : is responsible for generating all the transactions necessary to rollback a current configuration to its original state.

5) Data Model Manager : Keeps track of all the YANG data models and the corresponding managed objects. Also keeps track of the applications which provide data for each part of a data model.
6) Configuration Validator : checks if the resulting configuration after applying a transaction would be a valid configuration.
7) Configuration Database : contains both configuration and operational data.
8) Configuration API : Using the configuration API the application on the IoT device can be read configuration data from the configuration datastore and write operational data to the operational datastore.
9) Data Provider API: Applications on the IoT device can register for callbacks for various events using the Data Provider API. Through the Data Provider API, the applications can report statistics and operational ldata.

NETOPEER

- Netopeer is set of open source NETCONF tools built on the Libnetconf library.
- Netopeer-server: Netopeer-server is a NETCONF protocol server that runs on the managed device. Netopeer-server provide an environment for configuring the device using NETCONF RPC operations and also retrieving the state data from the device.
- Netopeer-agent: Netopeer-agent is the NETCONF protocol agent running as a SSH/TLS subsystems. NETopeer-agent accepts incoming NETCONF connection and passes the NETCONF RPC operations received from the NETCONF client to the Netopeer-server.

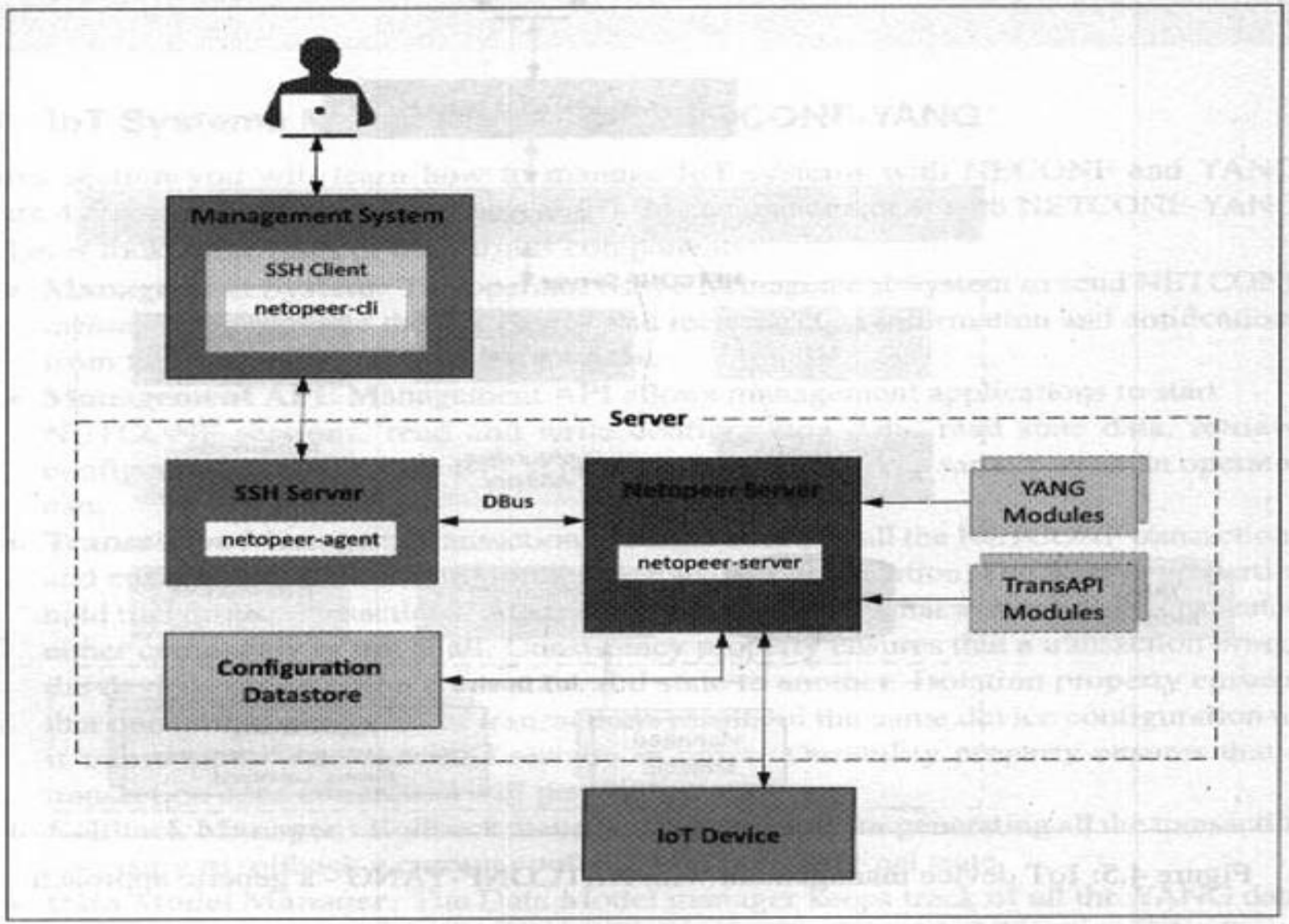


Figure 4.6: IoT device management with NETCONF - a specific approach based on Netopeer tools

- Netopper-cli: Netopper-cli is a NETCONF client that provide a command line interface(cli) for interacting with the Netopeer server. The operator can use the Netopeer-cli from the management system to send NETCONF RPC operations for configuring the device and retrieving the state information.
- Netopeer manager: Netopeer-manager allows managing the YANG and Libnetconf Transaction API(TransAPI) modules on the Netopeer-server. With Netopeer-manager module can be loaded or removed from the server.
- Netopeer-configurator: Netopeer-configurator is a tool that can be used to configure the Netopeer-server.

Steps for IoT device Management with NETCONF-YANG:

1) Create a YANG model of the system that defines the configuration and state data of the system.
2) Complete the YANG model with the “Inctool” which comes with Libnetconf.
3) Fill in the IoT device management code in the TransAPI module.
4) Build the callbacks C file to generate the libraryfile.
5) Load the YANG module and the TransAPI module into the Netopeer server using Netopeer manager tool.
6) The operator can now connect from the management system to the Netopeer server using the Netopeer CLI.
7) Operator can issue NETCONF commands from the Netopeer CLI. Command can be issued to change the configuration data, get operational data or execute an RPC on the IoTdevice.

Unit – III

DESIGN METHOLOGY INTERNET OF THINGS

IoT Platforms Design Methodology, Introduction, IoT Design Methodology, Case Study on IoT System for Weather Monitoring

Motivation for Using Python – IoT Systems, logical Design using Python, installing Python, Python Data Types& Data Structures, Control flow, functions, Modules, Packages, File Handling, Data/Time Operations, Classes, Python Packages of Interest for IoT.

IoT Platforms Design Methodology

Introduction:

- IoT systems comprise of multiple components and deployment tiers.
- IoT defined six different levels. Each level is suited for different applications and has different component and deployment configurations.
- The IoT systems involve interactions between various components such as IoT devices and network resources, web services, analytics components, application and database servers.
- IoT system designers may find it difficult to evaluate the available alternatives.
- IoT system designers often tend to design IoT systems keeping specific product/services in mind.
- Any problem in systems, the designer updating the system design to add new features or replacing a particular product/service choice.

IoT Design Methodology

IoT designers often tend to design the system keeping specific products in mind

Step 1: Purpose & Requirements Specification

Step 2: Process Specification

Step 3 : Domain Model Specification

Step 4: Information Model Specification

Step 5: Service Specifications

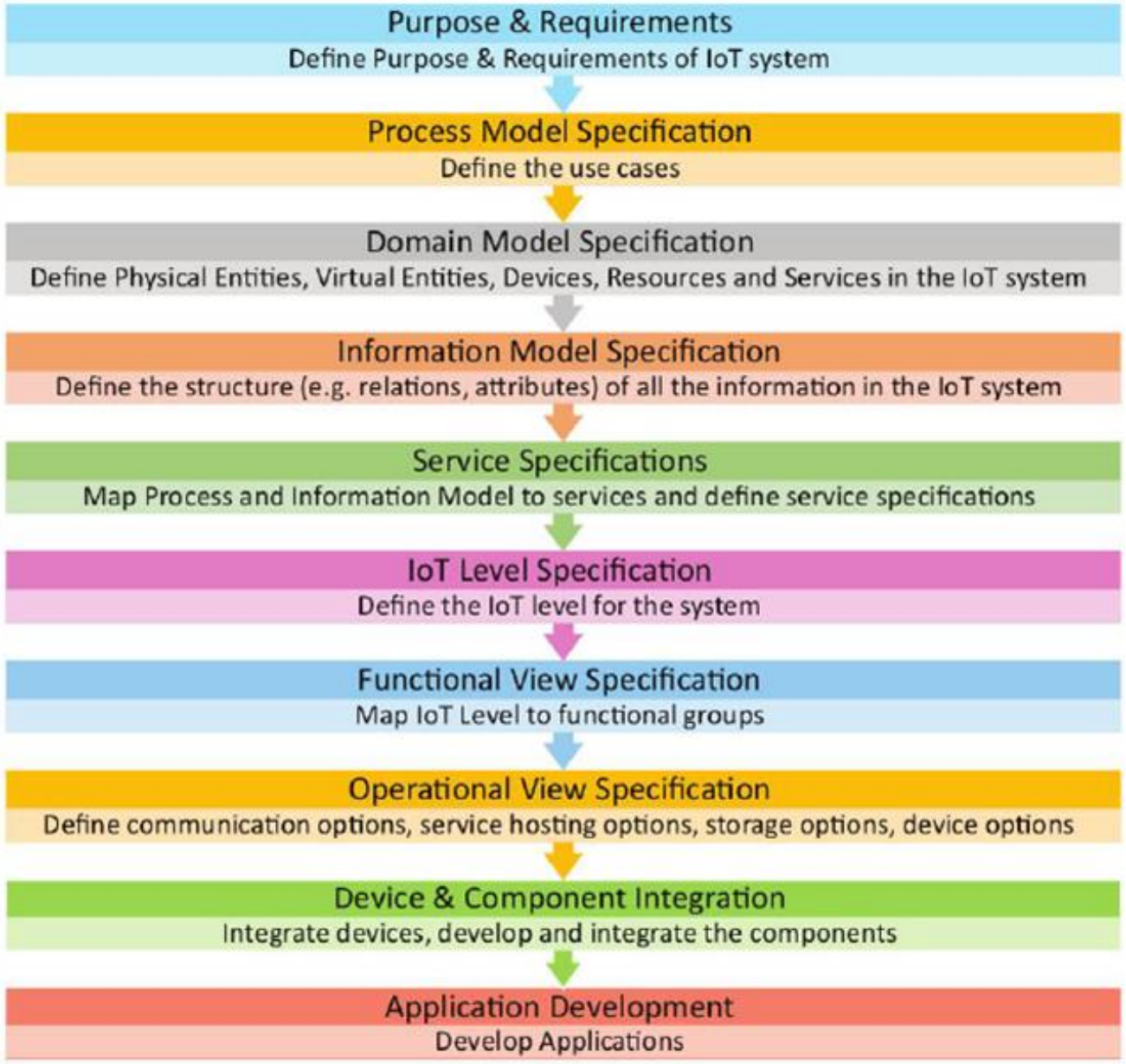
Step 6: IoT Level Specification

Step 7:Functional View Specification

Step 8 : Operational View Specification

Step 9: Device & Component Integration

Step 10: Application Development



We will look at a generic design methodology which is independent of specific product, service or programming language. IoT systems designed with this methodology will have reduced design time, testing time, maintenance time, complexity and better interoperability.

Step 1: Purpose & Requirements Specification

First step is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements are captured.

Requirements can be:

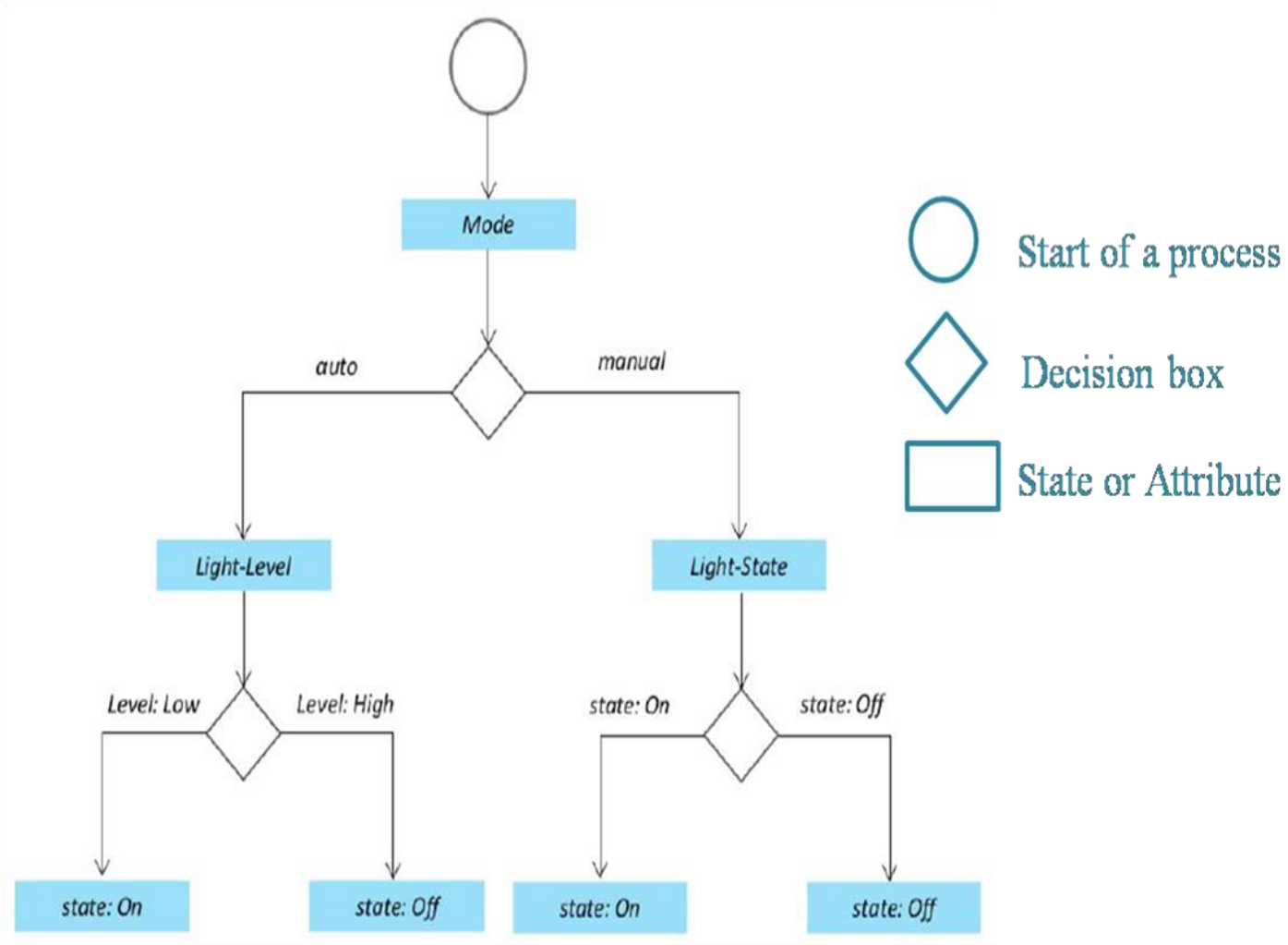
- Data collection requirements
- Data analysis requirements
- System management requirements
- Security requirements
- User interface requirements

For home automation system the purpose and requirements specification is as follows:

Purpose	A home automation system that allows controlling the lights remotely using a web application
Behaviour	Home automation system should support two modes: auto and manual **Auto:** System measures the light level in the room and switches on the light when it is dark **Manual:** Allows remotely switching lights on and off
System Management	System should provide remote monitoring and control functions
Data Analysis	System should perform local analysis of the data
Application Deployment	Application should be deployed locally, but should be accessible remotely
Security	Should provide basic security like user authentication

Step 2: Process Specification

- The Second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.

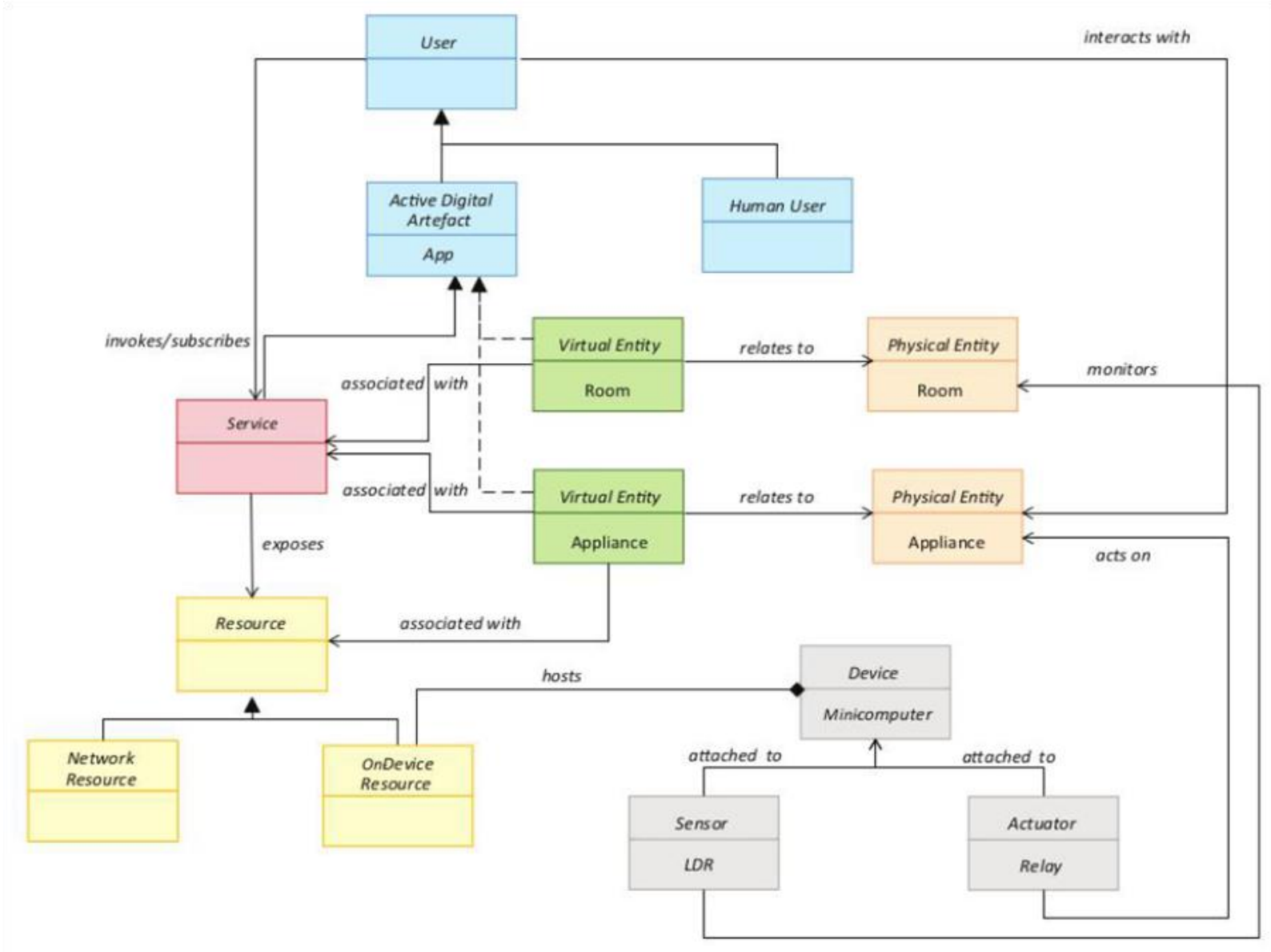


Step 3: Domain Model Specifications

- The domain model describes the main concepts, entities and objects in the domain of the IoT system to be designed.
- Domain model defines the attributes of the objects and relationships between objects.
- The domain model is independent of any specific technology or platform.
- Using domain model, system designers can get an understanding of the IoT domain for which the system is to be designed.
- The entities, objects and concepts defined in the domain model of home automation system include the following:

Physical Entity	• The physical identifiable objects in the environment • IoT system provides information about the physical entity (using sensors) or performs actuation upon the physical entity
Virtual Entity	• Virtual entity is a representation of the physical entity in the digital world • For every physical entity there is a virtual entity
Device	• Devices provide a medium for interaction between physical and virtual entities • Devices are used to gather information from or perform actuation on physical entities
Resource	• Resources are software components which can be either on-device or network-resources • On-device resources are hosted on the device and provide sensing or actuation (eg: operating system) • Network-resources include software components that are available on the network (eg: database)
Service	• Services provide an interface for interacting with the physical entity • Services access resources to perform operations on physical entities

The domain model specification diagram for home automation system is as shown in the below figure.



One-way Association

Generalization/Specialization

Aggregation Relationship

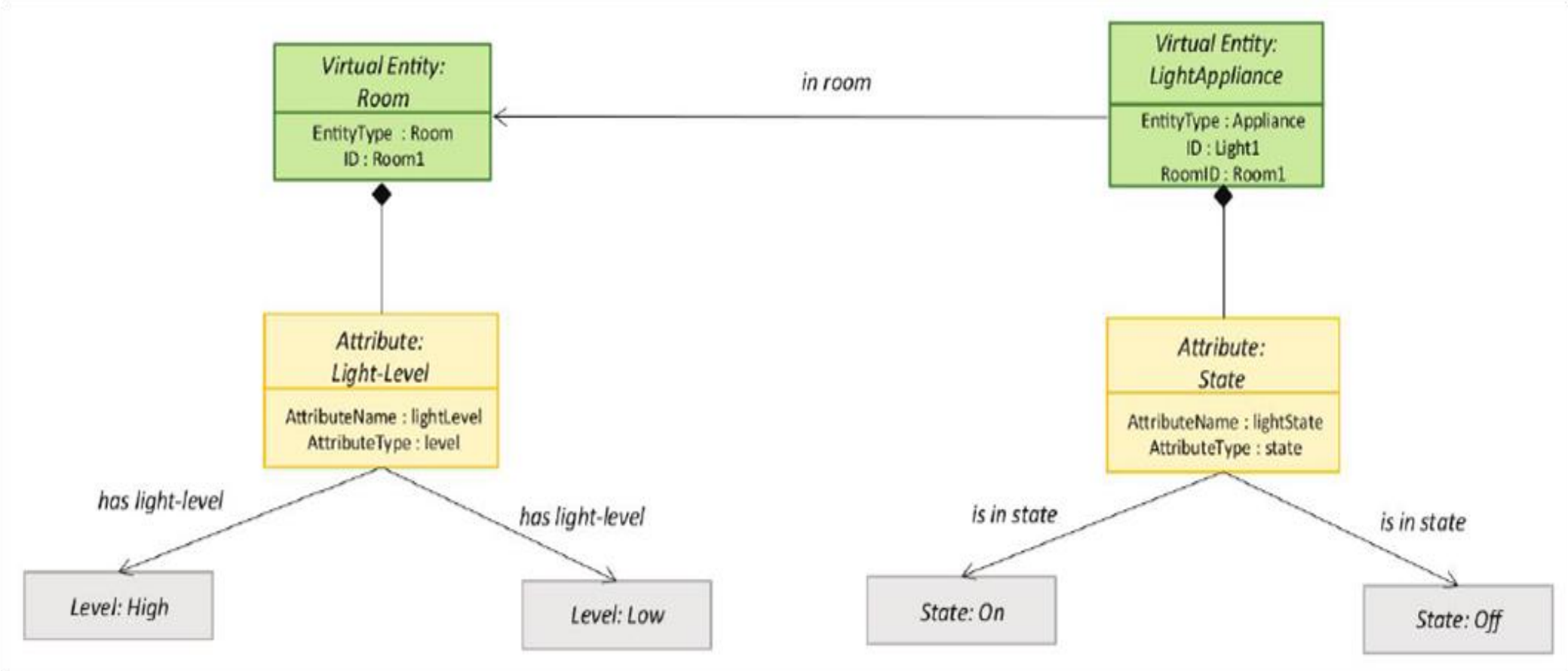
Type

Type: Entity, service, resource, device, attribute

Step 4: Information Model Specification

- Information model defines the structure of all the information in the IoT system.
- To define the information model, we first list the virtual entities. Later more details like attributes and relationships are added.

The information model specification for home automation system is as shown below:

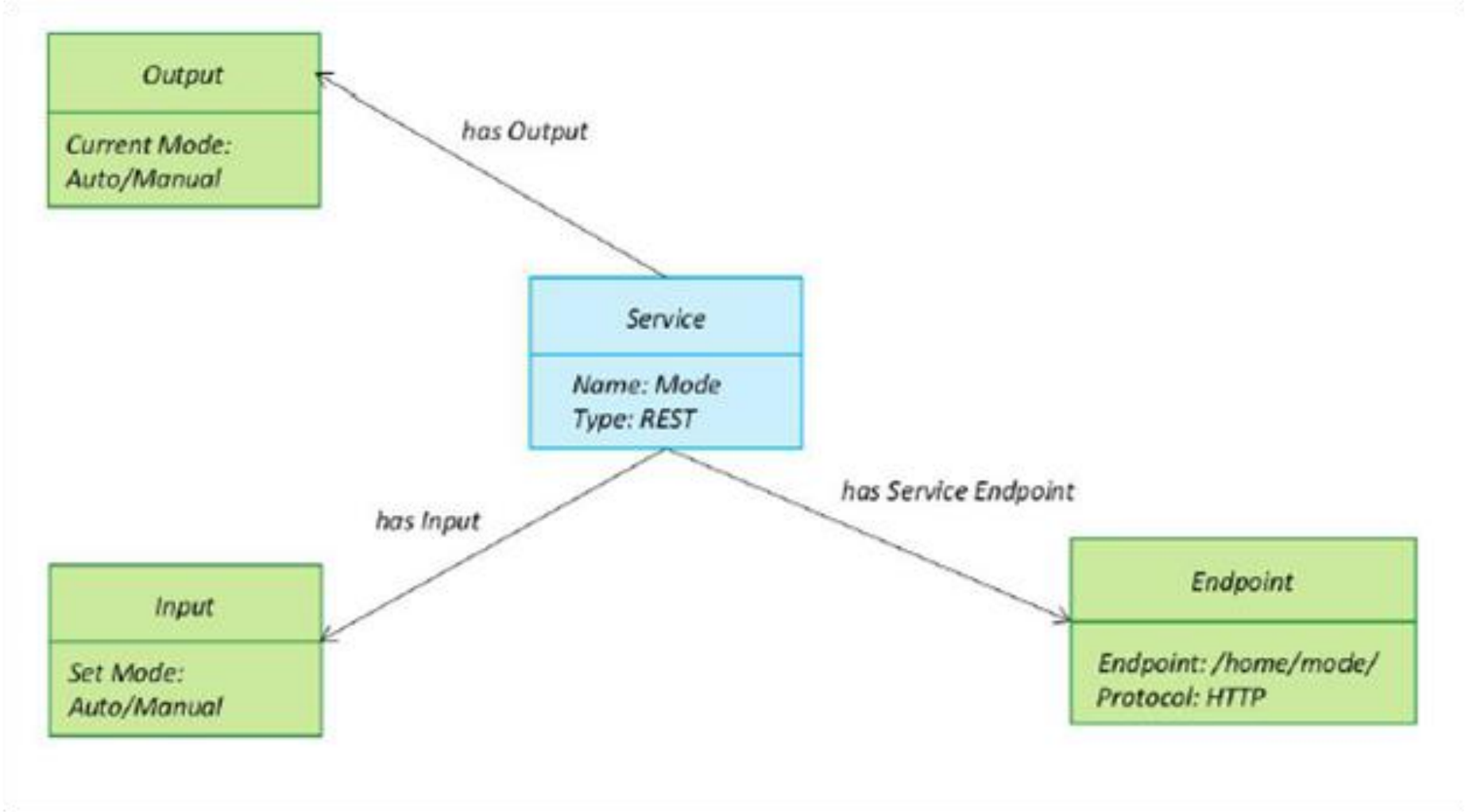


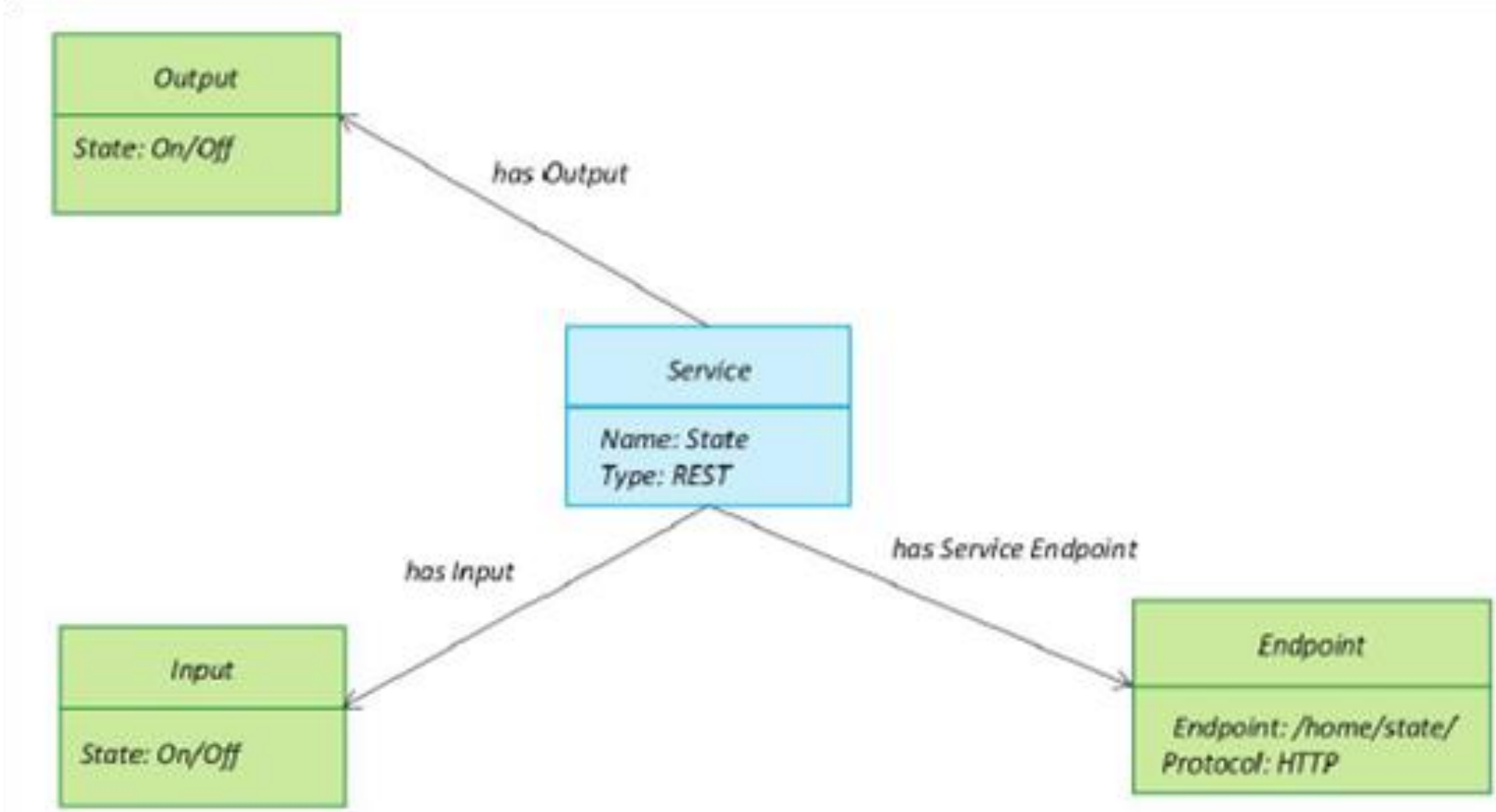
Step 5: Service Specifications

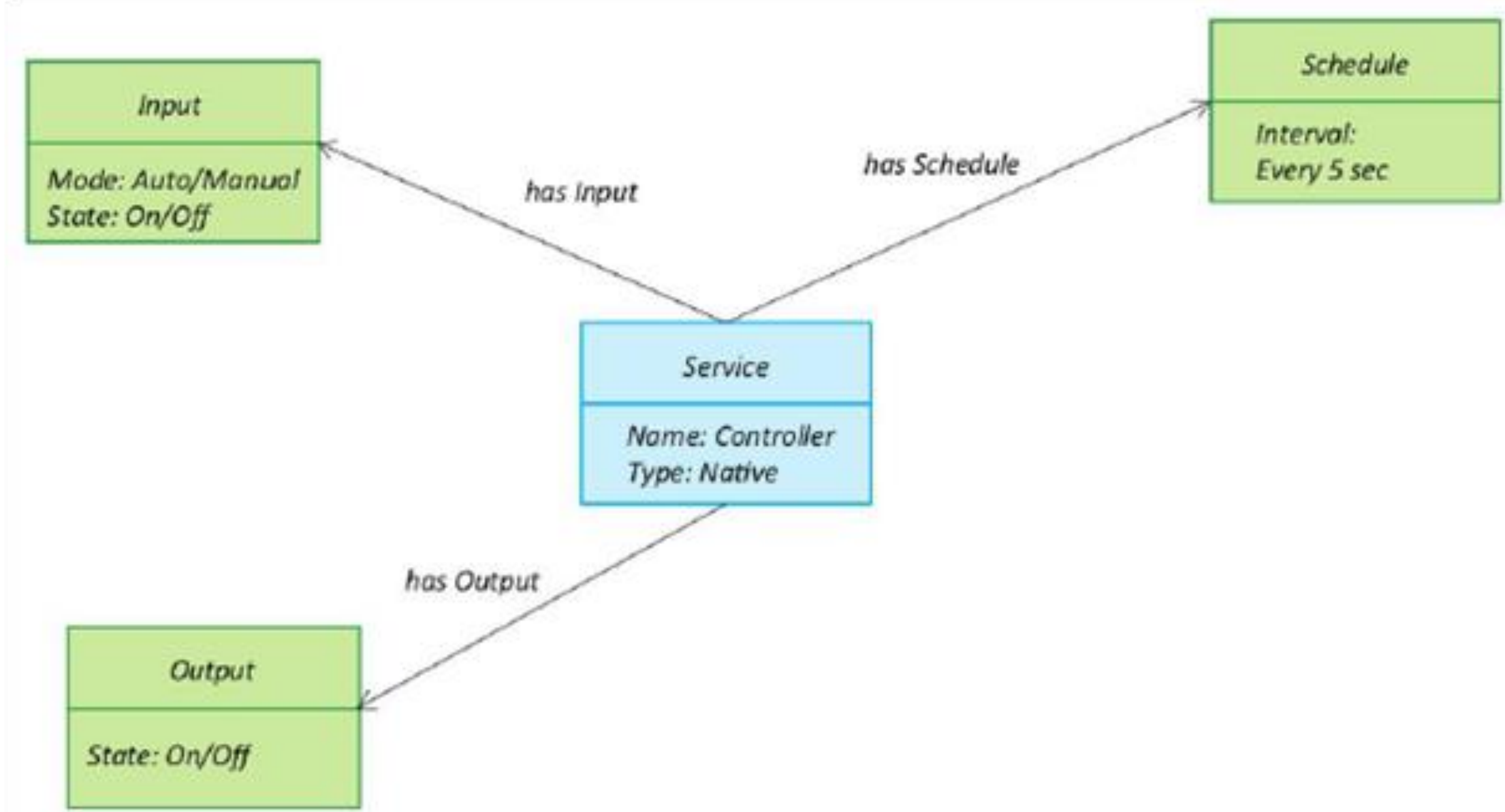
- The service specification defines the following:
    - Services in the system
    - Service types
    - Service inputs/output
    - Service endpoints
    - Service schedules
    - Service preconditions
    - Service effects

- 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.

The service specification for each state in home automation systems are as shown below:

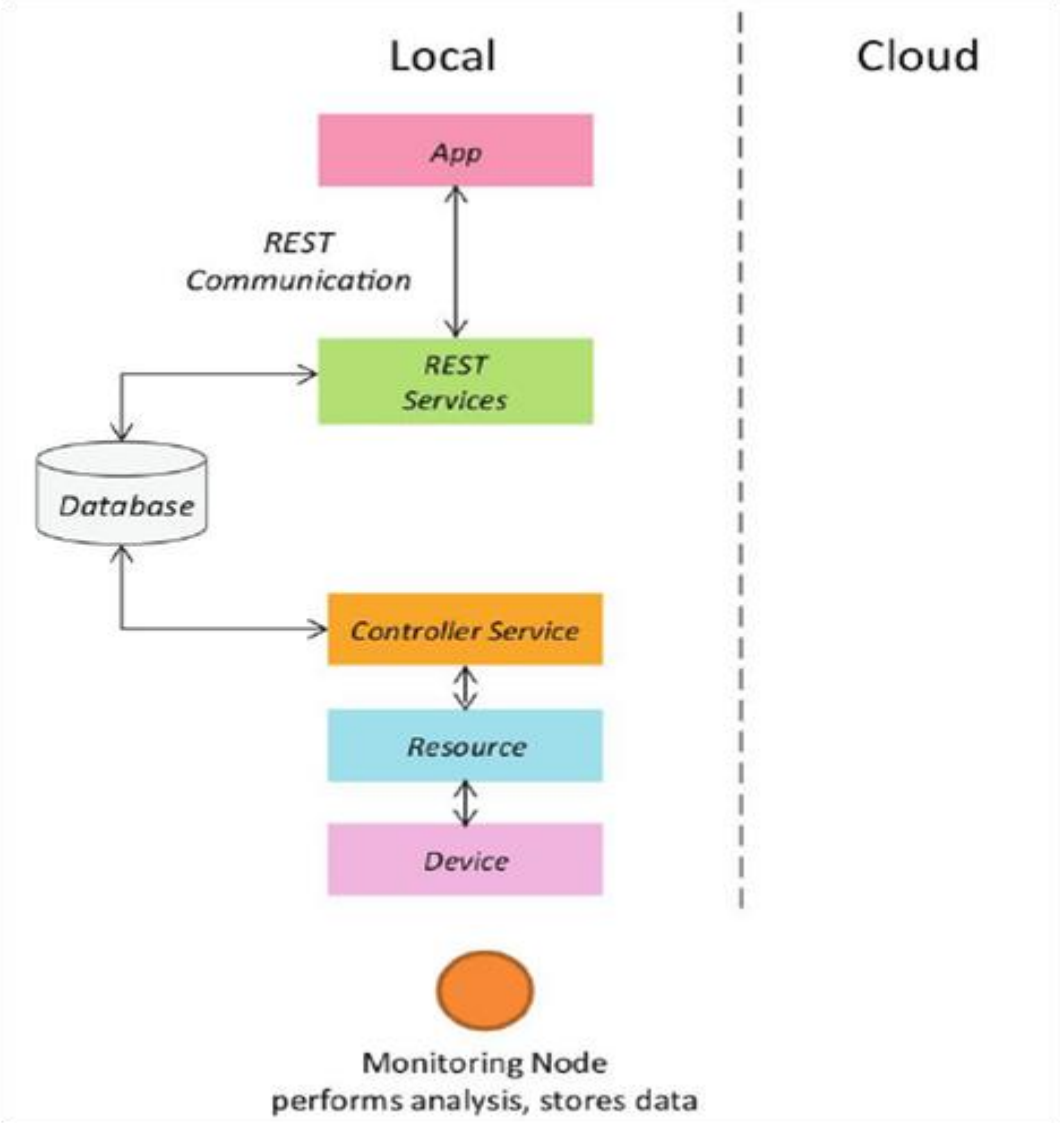



Output
State: On/Off
has Output
Service
Name: State
Type: REST
has Input
has Service Endpoint
Input
State: On/Off
Endpoint
Endpoint: /home/state/
Protocol: HTTP


Input
Mode: Auto/Manual
State: On/Off
has Input
has Schedule
Schedule
Interval:
Every 5 sec
Service
Name: Controller
Type: Native
has Output
Output
State: On/Off

Step 6: IoT Level Specification

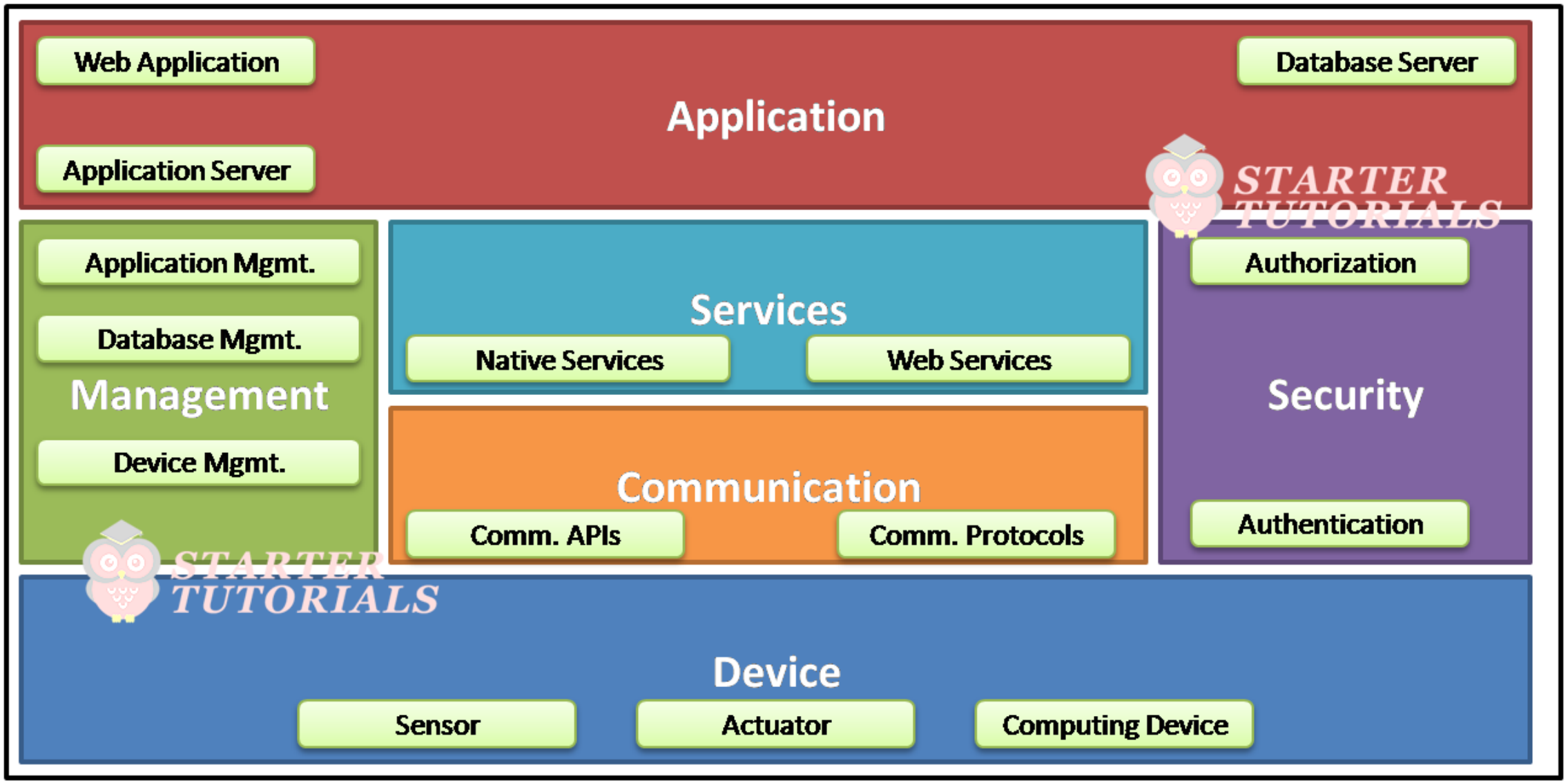
- The sixth step in the IoT design methodology is to define the IoT level for the system (unit-1, we defined five IoT deployment levels)
- The deployment level for home automation system is shown in the below figure.



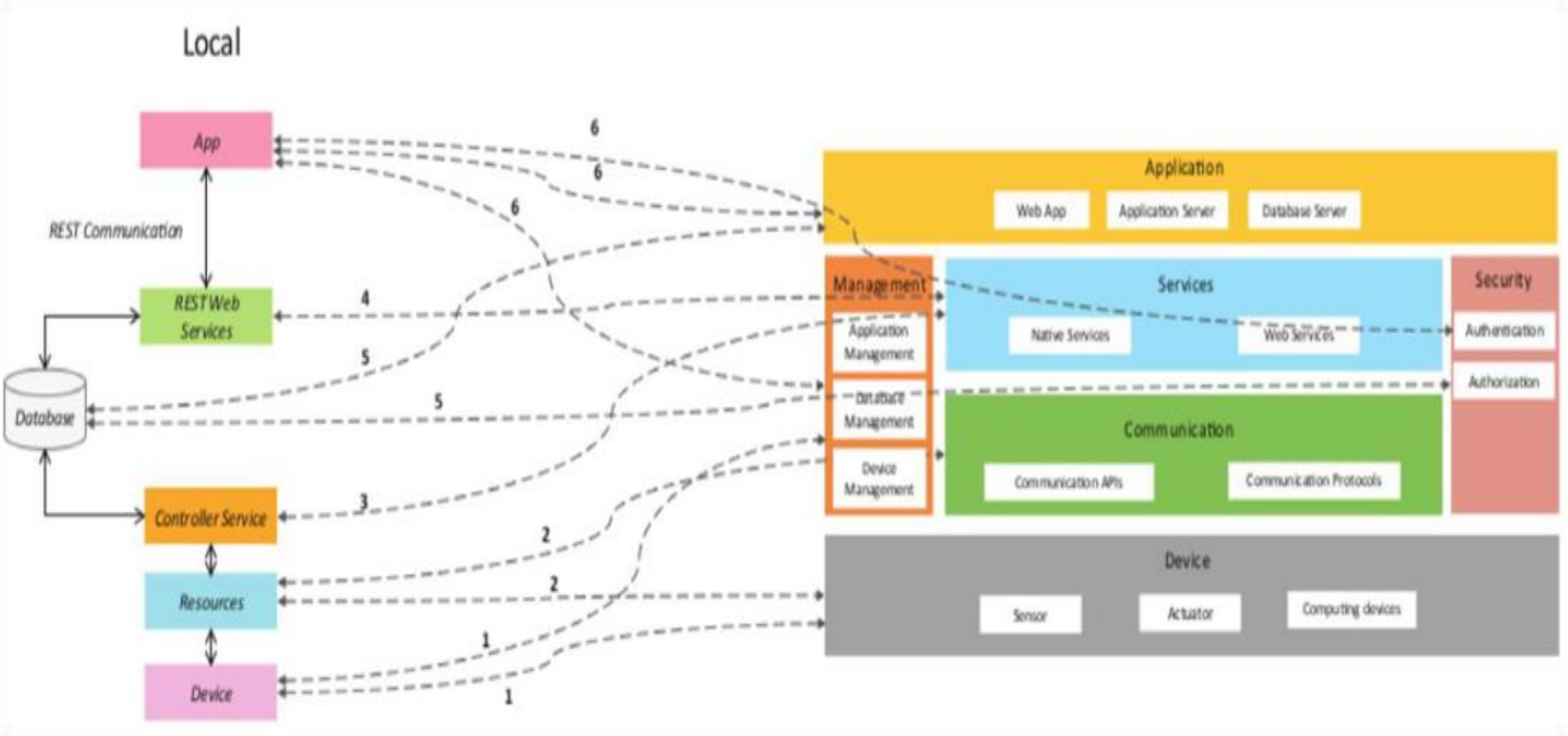
Step 7: Functional View Specification

- The functional view defines the functions of the IoT systems grouped into various functional groups.
- Each functional group provides functionalities for interacting with concepts in the domain model and information related to the concepts.
- The functional groups in a functional view include: Device, Communication, Services, Management, Security, and Application.

The functional view specification for home automation system is shown in the below figure:



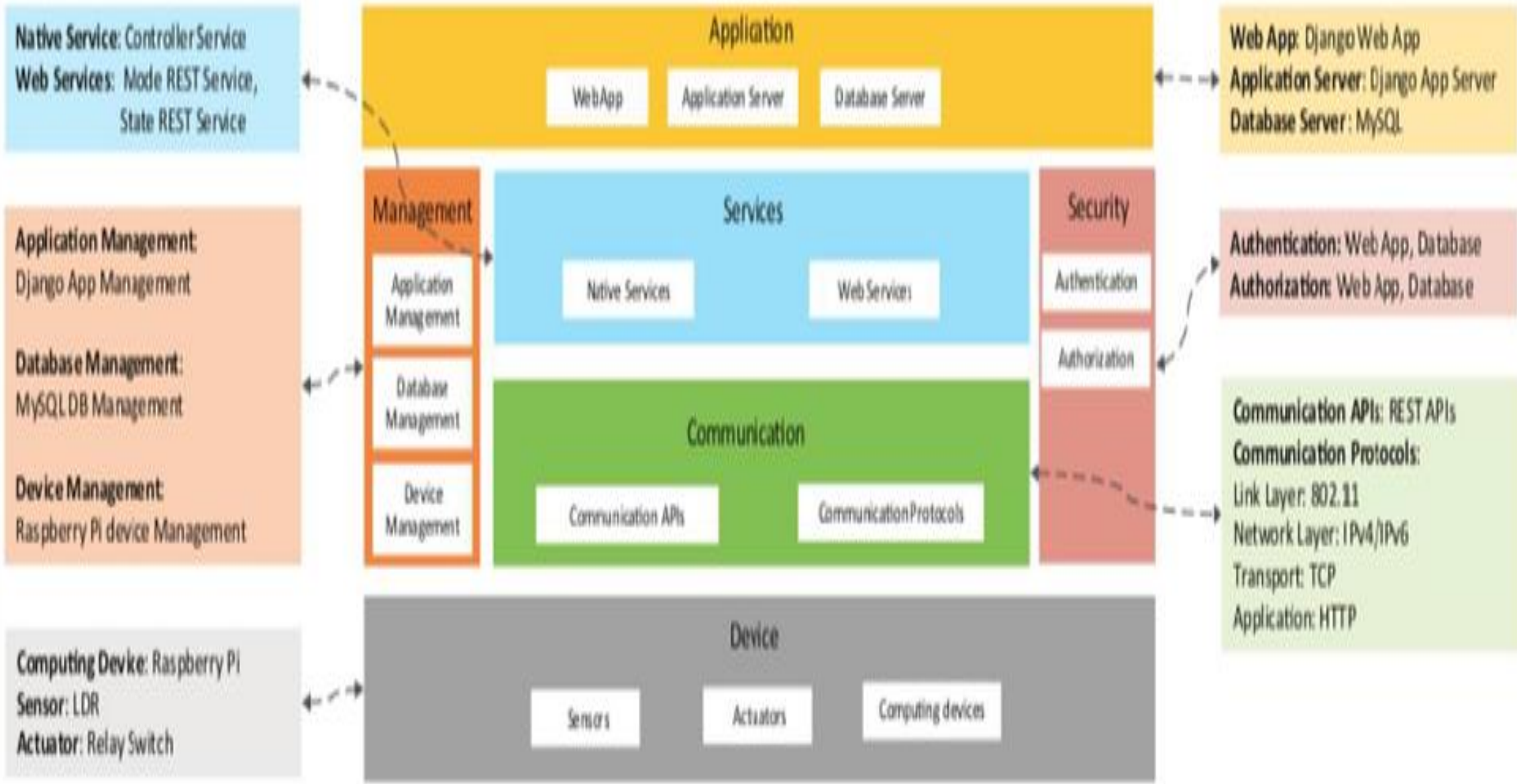
The mapping between the IoT level and the functional groups is as shown in the below figure.



Step 8: Operational view specification

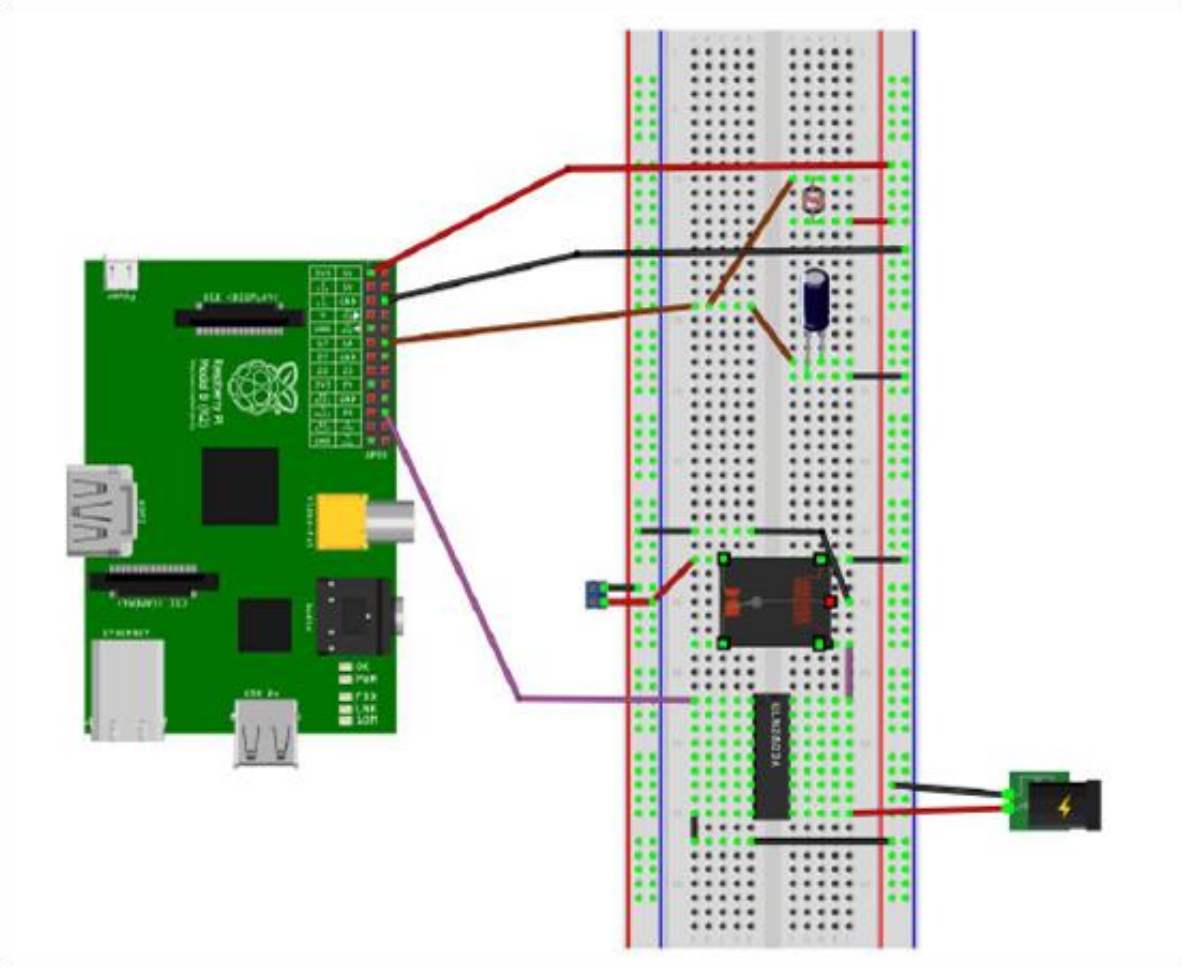
- In this step, various options related to the IoT system deployment and operation are defined, such as:
  - Service hosting options
  - Storage options
  - Device options
  - Application hosting options

- The options chosen for home automation system are as shown in the below figure.


Native Service: Controller Service
Web Services: Node REST Service, State REST Service
Appliation
WebApp
Application Server
Database Server
Web App: Django Web App
Application Server: Django App Server
Database Server: MySQL
Application Management: Django App Management
Database Management: MySQL DB Management
Device Management: Raspberry Pi device Management
Management
Application Management
Database Management
Device Management
Services
Native Services
Web Services
Communication
Communication APIs
Communication Protocols
Security
Authentication
Authorization
Authentication: Web App, Database
Authorization: Web App, Database
Communication APIs: REST APIs
Communication Protocols:
Link Layer: 802.11
Network Layer: IPv4/IPv6
Transport: TCP
Application: HTTP
Computing Devke: Raspberry Pi
Sensor: LDR
Actuator: Relay Switch
Device
Sensors
Actuators
Computing devices

Step 9: Device & Component Integration

- In this step the devices like sensors, computing devices and other components are integrated together.
- The interconnection of different components in our home automation system are as shown in the figure given below.



Step 10: Application Development

- Using all the information from previous steps, we will develop the application (code) for the IoT system.
- The application interface for home automation system is shown below.

Case Study on IoT System for Weather Monitoring

Refer Enclosed PDF and Submit case study with your own idea and Experience

Motivation for Using Python

- Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.
- The main characteristics of Python are:

1) Multi-paradigm programming language.

2) Python supports more than one programming paradigms including object- oriented programming and structured programming.

3) Python is an interpreted language and does not require an explicit compilation step.

4) The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.

5) Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

Benefits:

- **Easy-to-learn, read and maintain**
  - Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages. Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.
- **Object and Procedure Oriented**
  - Python supports both procedure-oriented programming and object-oriented programming. Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.
- **Extendable**
  - Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.
- **Scalable**
  - Due to the minimalistic nature of Python, it provides a manageable structure for large programs.
- **Portable**
  - Since Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source
- **Broad Library Support**
  - Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.

Python Setup

- Windows
  - Python binaries for Windows can be downloaded from http://www.python.org/getit .
  - For the examples and exercise in this book, you would require Python 2.7 which can be directly downloaded from http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi
  - Once the python binary is installed you can run the python shell at the command prompt using

```
> python
```

- Linux

```
#Install Dependencies
sudo apt-get install build-essential
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev  libc6-dev libbz2-dev

#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5

#Install Python
./configure
make
sudo make install
```

Datatypes

- Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
- There are various data types in Python. Some of the important types are listed below.

Python Numbers

- Integers, floating point numbers and complex numbers falls under Python numbers category.
- They are defined as int, float and complex class in Python.
- We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

Script.py

```
1. a = 5
2. print(a, "is of type", type(a))
 3. a = 2.0
4. print(a, "is of type", type(a))
 5. a = 1+2j
6. print(a, "is complex number?", isinstance(1+2j,complex))
```

```
>>> a = 12345678901234567 89
>>> a
    12345678901234567 89
>>> b = 0.12345678901234567 89
>>> b
    0.1234567890123456 8
>>> c = 1+2j
>>> c
    (1+2j)
```

Python List :

- List is an ordered sequence of items.
- It is one of the most used datatype in Python and is very flexible.
- All the items in a list do not need to be of the same type.
- Declaring a list is pretty straight forward.
- Items separated by commas are enclosed within brackets [].

```
>>> a = [1, 2.2, 'python']
```

- We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Script.py

1. a = [5,10,15,20,25,30,35,40]
2. a[2] = 15
3. print("a[2] = ", a[2])
4. a[0:3] = [5, 10, 15]
5. print("a[0:3] = ", a[0:3])
6. a[5:] = [30, 35, 40]
7. print("a[5:] = ", a[5:])

- Lists are mutable, meaning; value of elements of a list can be altered.
- >>> a = [1,2,3]
- >>> a[2]=4
- >>> a [1, 2, 4]

Python Tuple :

- Tuple is an ordered sequences of items same as list.
- The only difference is that tuples are immutable.Tuples once created cannot be modified.
- Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.
- It is defined within parentheses () where items are separated by commas.

```
>>> t = (5,'program', 1+3j)
```

Script.py

```
t = (5,'program', 1+3j)
# t[1] = 'program'
print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
# Generates error
# Tuples are immutable
t[0] = 10
```

Python Strings :

- String is sequence of Unicode characters.
- We can use single quotes or double quotes to represent strings.
- Multi-line strings can be denoted using triple quotes, ''' or """.

```
>>> s = "This is a string"
>>> s = '''a multiline
```

- Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

Script.py

- a ={5,2,3,1,4}
- # printing set variable
- print("a = ", a)
- # data type of variable a
- print(type(a))

- We can perform set operations like union, intersection on two sets.
- Set have unique values. They eliminate duplicates. Since, set are unordered collection, indexing has no meaning.
- Hence the slicing operator [] does not work. It is generally used when we have a huge amount of data.
- Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
- In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

```
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

- We use key to retrieve the respective value. But not the other way around.

Script.py

```
d ={1:'value','key':2}
 print(type(d))
print("d[1] = ",d[1]);
print("d['key'] = ", d['key']);
# Generates error
print("d[2] = ",d[2]);
```

Python if...else Statement

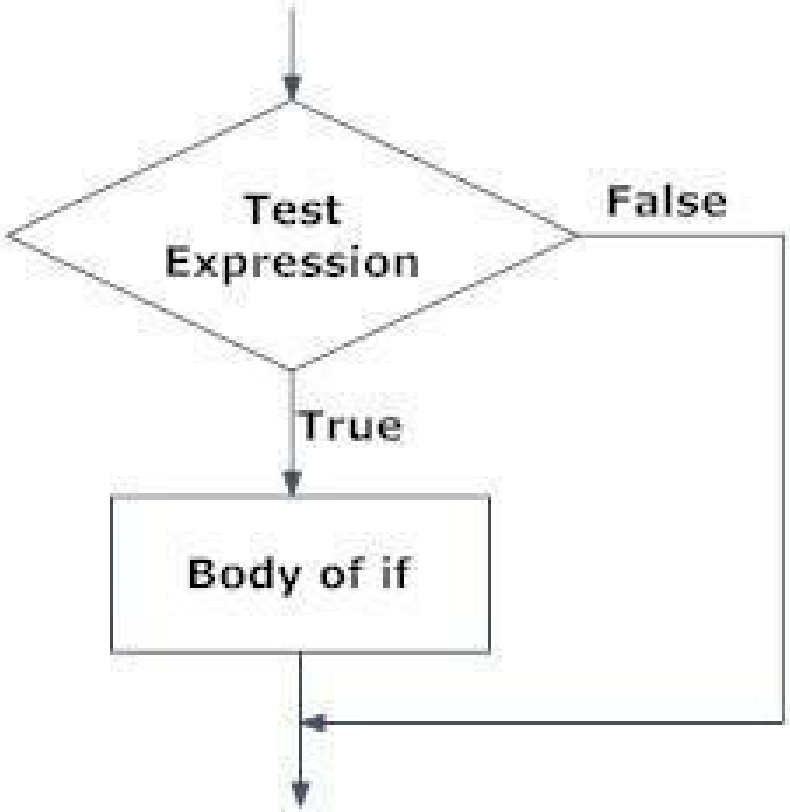


Fig: Operation of if statement

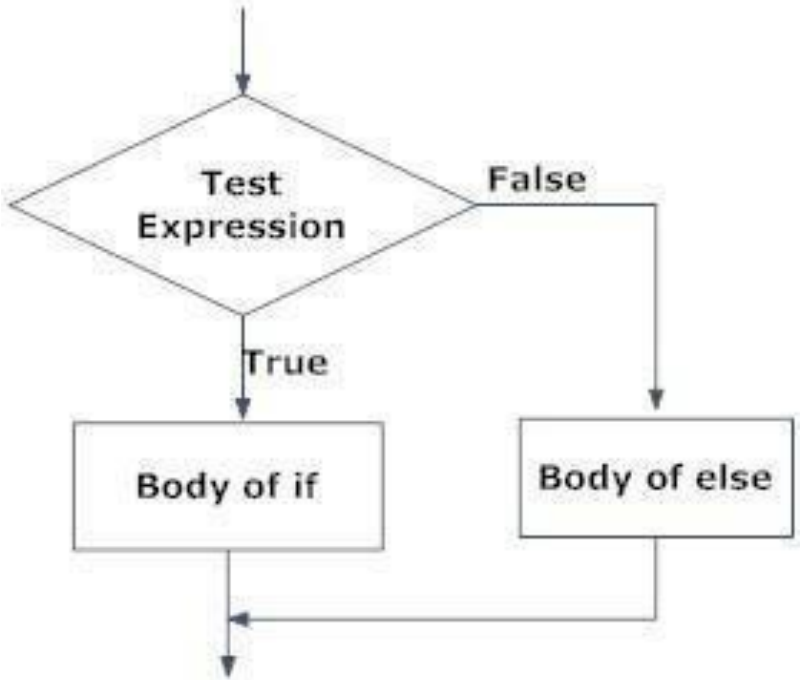


Fig: Operation of if...else statement

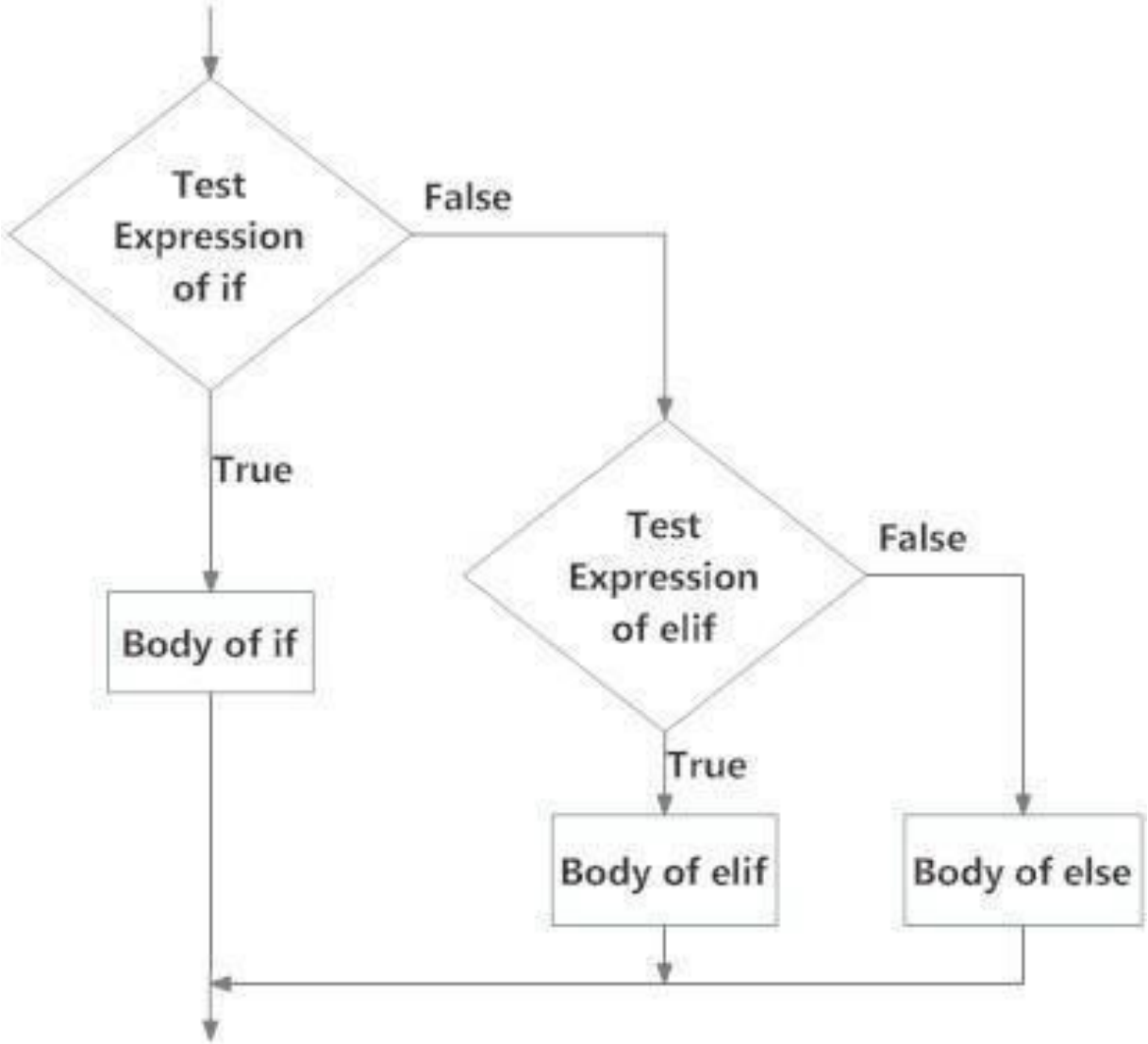


Fig: Operation of if...elif...else statement

Python Nested if Example:

```
# In this program, we input a number check if the number is
  positive #or negative or zero and display an appropriate
  message
# This time we use nested if
num = float(input("Enter a number: ")) if num >= 0:
if num == 0: print("Zero")
else:
print("Positive number")
else:
print("Negative number")
```

Python for Loop

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.
- Syntax of for Loop

```
for val in sequence:
    Body of for
```

- Here, val is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.
- The body of for loop is separated from the rest of the code using indentation.

Syntax

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
        sum = sum+val
# Output: The sum is 48
print("The sum is", sum)
```

when you run the program, the output will be:

```
The sum is 48
```

The range() function

- We can generate a sequence of numbers using range() function.
- range(10) will generate numbers from 0 to 9 (10 numbers).
- We can also define the start, stop and step size as range(start,stop,step size).
- step size defaults to 1 if not provided.
- # Output: range(0, 10) print(range(10))
- # Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- print(list(range(10)))

- # Output: [2, 3, 4, 5, 6, 7]
- print(list(range(2, 8)))
- # Output: [2, 5, 8, 11, 14, 17]
- print(list(range(2, 20, 3)))

- We can use the range() function in for loops to iterate through a sequence of numbers.
- It can be combined with the len() function to iterate though a sequence using indexing.
- Here is an example.

```
# Program to iterate through a list using indexing
genre = ['pop', 'rock', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
        print("I like", genre[i])
```

- When you run the program, the output will be:

```
I like pop
I like rock
I like jazz
```

while loop in Python

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- We generally use this loop when we don't know beforehand, the number of times to iterate.
- **Syntax of while Loop in Python**

```
while test_expression:
    Body of while
```

- In while loop, test expression is checked first.
- The body of the loop is entered only if the test_expression evaluates to True.
- After one iteration, the test expression is checked again.
- This process continues until the test_expression evaluates to False.
- In Python, the body of the while loop is determined through indentation.
- Body starts with indentation and the first unindented line marks the end.
- Python interprets any non-zero value as True.
- None and 0 are interpreted asFalse.

```
# Program to add natural  numbers upto  sum = 1+2+3+...+n
# To take input from the user, n = int(input("Enter n: "))
n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
        sum = sum + i
        i=i+1
# updatecounter print the sum
print("The sum is", sum)
```

When you run the program, the output will be:

Enter n: 10 The sum is 55

Python Modules

- A file containing a set of functions you want to include in the application is called Module.

Create a Module

- To create a module just save the code you want in a file with the file extension .py:

Example

- Save this code in a file named mymodule.py

```
def greeting(name):
    print("Hello, " + name)
```

Use a Module

- Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule
mymodule.greeting("Jonathan")
```

Note: When using a function from a module,

use the syntax: **module_name.function_name.**

Variables in Module

- The module can contain functions, as already described, but also variables of all types(arrays, dictionaries, objects etc):

Example

- Save this code in the file mymodule.py

```
person1 = {"name": "John","age": 36,"country": "Norway"}
```

Example

- Import the module named mymodule, and access the person1 dictionary: import mymodule

```
a = mymodule.person1["age"]
print(a)
```

Naming a Module

- You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module

- You can create an alias when you import a module, by using the as keyword:

Example

- Create an alias for mymodule called mx:

```
import mymodule as mx

a = mx.person1["age"]

print(a)
```

Built-in Modules

- There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the platform module:

```
import platform
x = platform.system()
print(x)
```

Using the dir() Function

- There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example

List all the defined names belonging to the platform module:

```
import platform
x = dir(platform)
print(x)
```

Note: The dir() function can be used on all modules, also the ones you create yourself.

Import from Module

- You can choose to import only parts from a module, by using the from keyword.

Example

The module named mymodule has one function and one dictionary:

```
def greeting(name):
print("Hello, " + name)
person1 = {"name": "John", "age": 36, "country": "Norway"}
```

Example

Import only the person1 dictionary from the module:

```
from mymodule import person1
print (person1["age"])
```

- **Note: When importing using the from keyword, do not use the module name when referring to elements in the module.**
- **Example: person1["age"], not mymodule.person1["age"].**

Packages :

- We don't usually store all of our files in our computer in the same location.
- We use a well- organized hierarchy of directories for easier access. Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory.
- Similarly, Python has packages for directories and modules for files.
- As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.
- Similar, as a directory can contain sub-directories and files, a Python package can have sub- packages and modules.
- A directory must contain a file namedinit.py in order for Python to consider it as a package.
- This file can be left empty but we generally place the initialization code for that package in this file.

Package Module Structure in Python Programming Importing module from a package

- We can import modules from packages using the dot (.) operator. For example, if want to import the start module in the above example, it is done as follows.

  import Game.Level.start

- Now if this module contains a function named select_difficulty(), we must use the full name to reference it.

  Game.Level.start.select_difficulty(2)

- If this construct seems lengthy, we can import the module without the package prefix as follows.

  from Game.Level import start

- We can now call the function simply as follows.

  start.select_difficulty(2)

- Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

  from Game.Level.start import select_difficulty

- Now we can directly call this function.

  select_difficulty(2)

- Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding. While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

Files

- File is a named location on disk to store related information.
- It is used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order.
    1. Open a file
    2. Read or write (perform operation)
    3. Close the file

How to open a file?

```
>>> f=open("test.txt") # open file in current directory
>>> f = open("C:/Python33/README.txt") # specifying full path
```

Mode	Description
'r‘	Open a file for reading. (default)
'w’	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

```
f=open("test.txt") # equivalent to 'r' or 'rt'
f = open("test.txt",'w') # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

How to close a file Using Python?

- A safer way is to use a try...finally block. try:

  f = open("test.txt",encoding = 'utf-8') # perform file operations
- finally:

  f.close()

How to write to File Using Python?

- with open("test.txt",'w',encoding = 'utf-8') as f:
  f.write("my first file\n")
  f.write("This file\n\n")
  f.write("contains three lines\n")

How to read files in Python?

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4) # read the first 4 data
'This'
>>>f.read(4) # read the next 4 data
' is'
>>>f.read() # read in the rest till end of file
'my first file\nThis file\n contains three lines\n'
>>> f.read() # further reading returns empty sting ''
>>>f.tell() # get the current file position 56
>>> f.seek(0) # bring file cursor to initial position 0
>>> print(f.read()) # read the entire file This is my first file
```

Method	Description
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read at most n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

Date & time Operation

```
import datetime

x = datetime.datetime.now()
print(x)
```

Date Output:
When we execute the code from the example above the result will be:
2024-09-30 18:29:30.670698

Classes:

- A Class is like an object constructor.
- Create a class named MyClass, with a property named x:

```
class MyClass:
    x = 5
```

- Now we can use the class named MyClass to create objects:

```
p1 = MyClass()
print(p1.x)
```

Python Packages of Interest for IoT

- JSON – JavaScript Object Notation
- XML – Extensible Markup Language
- HTTPLib & URLLib (Hyper Text Transfer Protocol Lib & Uniform Resource Locator Lib)
- SMTPLib – Simple mail transfer Protocol

JSON:

- Easy to read and write data-interchange format.
- Built into two structures: a collection of name-value pairs (python dictionary) and ordered lists of values (python list)
- Is used for serializing and transmitting structed data over a network connection.
- Ex.: transmitting data between a server and web application.

XML:

- Is a data format for structured document interchange.
- The Phython minidom Library provides a minimal implementation of the document object model interface and has an API similar to that in other language.

HTTPLib & URLLib:

- Used in network/internet programming.
- HTTPLib is an HTTP Client library and URLLib is library for fetching URLs

SMTP Lib:

- Which handles sending email and routing email between mail servers.
- The python smtplib modeule provides an SMTP client session object that can be used to send email.
- To send an email, first connection is established with the SMTP server by calling smtplib.SMTP with the SMTP server name and port.
- The user name and password provided arethen used to login into the server.
- The email is then sent by calling server.sendmail function with the form address, to address list and message as input parameters.

Django-REST Framework.

3. State service - REST-ful web service, hosted on device, implemented with Django-REST Framework.

- Application:
  Web Application - Django Web Application,
  Application Server - Django App Server,
  Database Server - MySQL.
- Security:
  Authentication: Web App, Database
  Authorization: Web App, Database

- Management:
  Application Management - Django App Management
  Database Management - MySQL DB Management,
  Device Management - Raspberry Pi device Management.

5.2.9 Step 9: Device & Component Integration

The ninth step in the IoT design methodology is the integration of the devices and components. Figure 5.12 shows a schematic diagram of the home automation IoT system. The devices and components used in this example are Raspberry Pi mini computer, LDR sensor and relay switch actuator. A detailed description of Raspberry Pi board and how to interface sensors and actuators with the board is provided in later chapters.

5.2.10 Step 10: Application Development

The final step in the IoT design methodology is to develop the IoT application. Figure 5.13 shows a screenshot of the home automation web application. The application has controls for the mode (auto on or auto off) and the light (on or off). In the auto mode, the IoT system controls the light appliance automatically based on the lighting conditions in the room. When auto mode is enabled the light control in the application is disabled and it reflects the current state of the light. When the auto mode is disabled, the light control is enabled and it is used for manually controlling the light.

5.3 Case Study on IoT System for Weather Monitoring

In this section we present a case study on design of an IoT system for weather monitoring using the IoT design methodology. The purpose of the weather monitoring system is to collect data on environmental conditions such as temperature, pressure, humidity and light

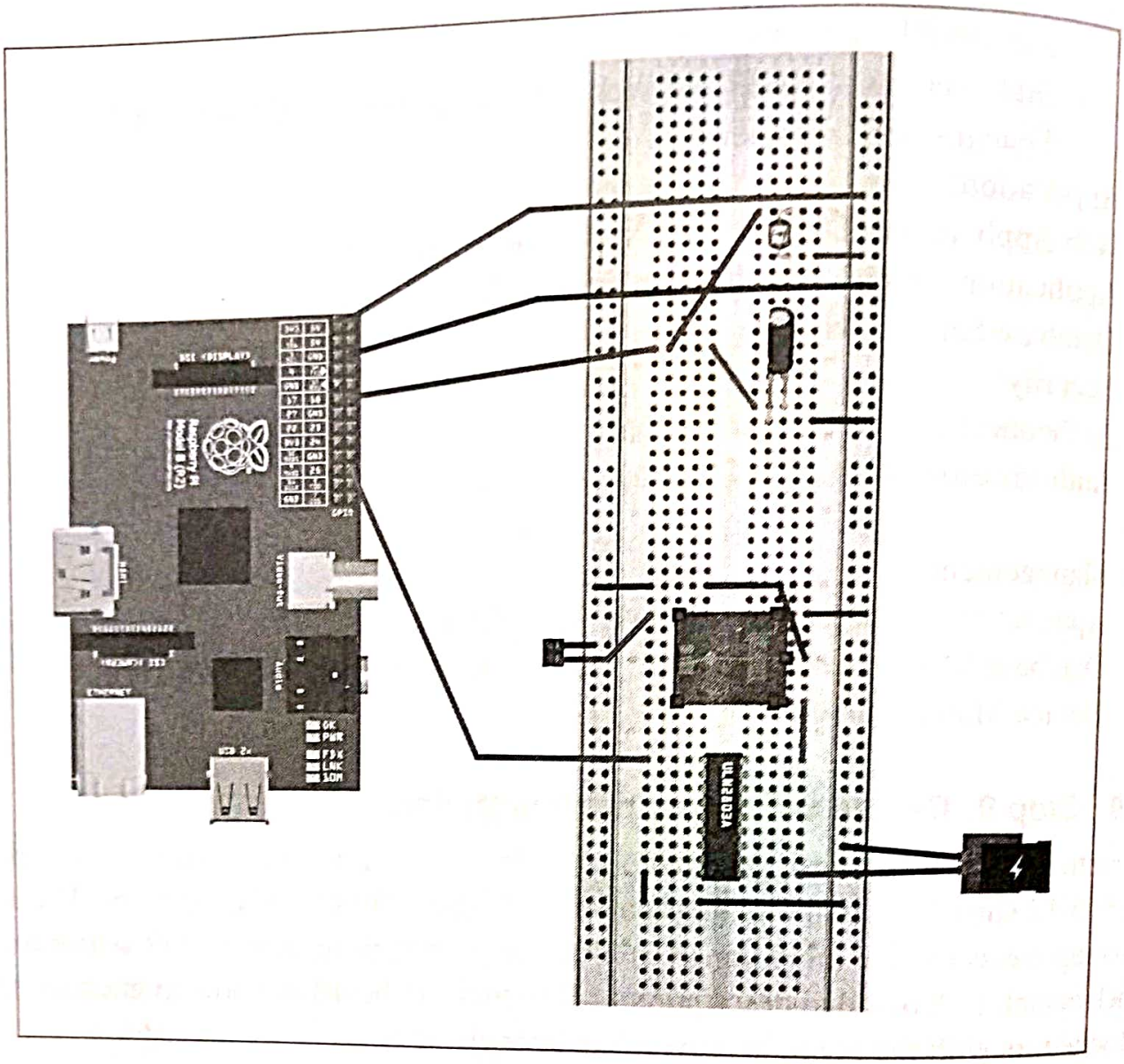


Figure 5.12: Schematic diagram of the home automation IoT system showing the device, sensor and actuator integrated

in an area using multiple end nodes. The end nodes send the data to the cloud where the data is aggregated and analyzed.

Figure 5.14 shows the process specification for the weather monitoring system. The process specification shows that the sensors are read after fixed intervals and the sensor measurements are stored.

Figure 5.15 shows the domain model for the weather monitoring system. In this domain model the physical entity is the environment which is being monitored. There is a virtual entity for the environment. Devices include temperature sensor, pressure sensor, humidity sensor, light sensor and single-board mini computer. Resources are software components which can be either on-device or network-resources. Services include the controller service that monitors the temperature, pressure, humidity and light and sends the readings to the

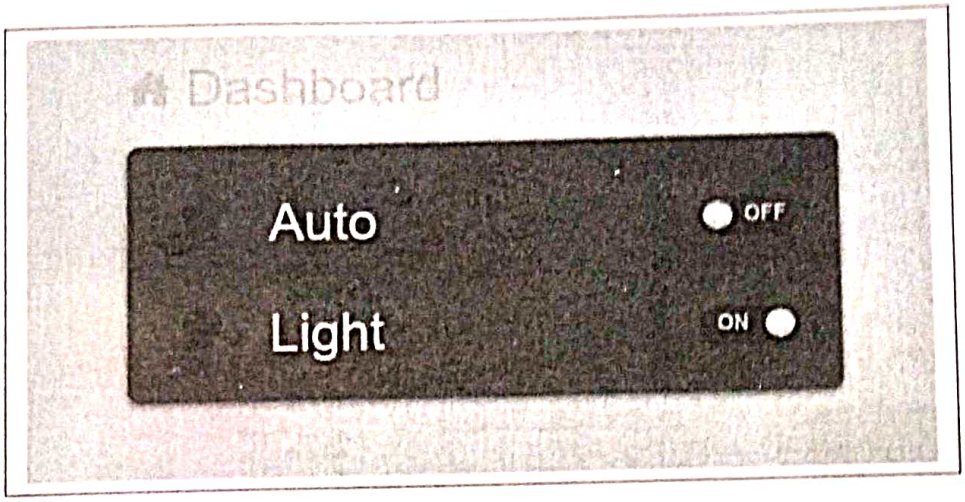


Figure 5.13: Home automation web application screenshot

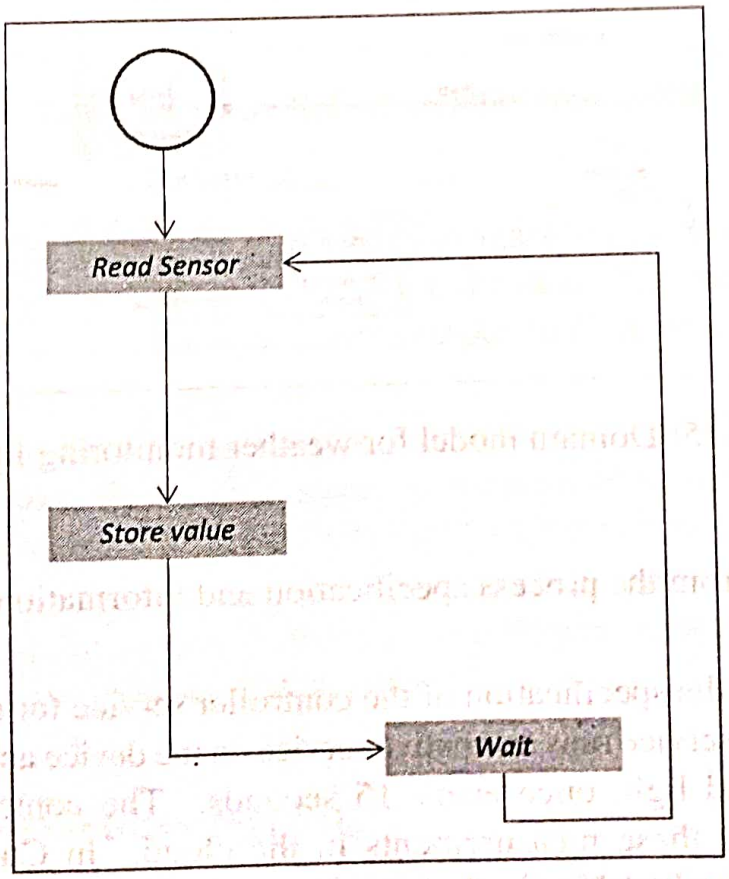


Figure 5.14: Process specification for weather monitoring IoT system

cloud.

Figure 5.16 shows the information model for the weather monitoring system. In this example, there is one virtual entity for the environment being sensed. The virtual entity has attributes - temperature, pressure, humidity and light. Figure 5.17 shows an example of

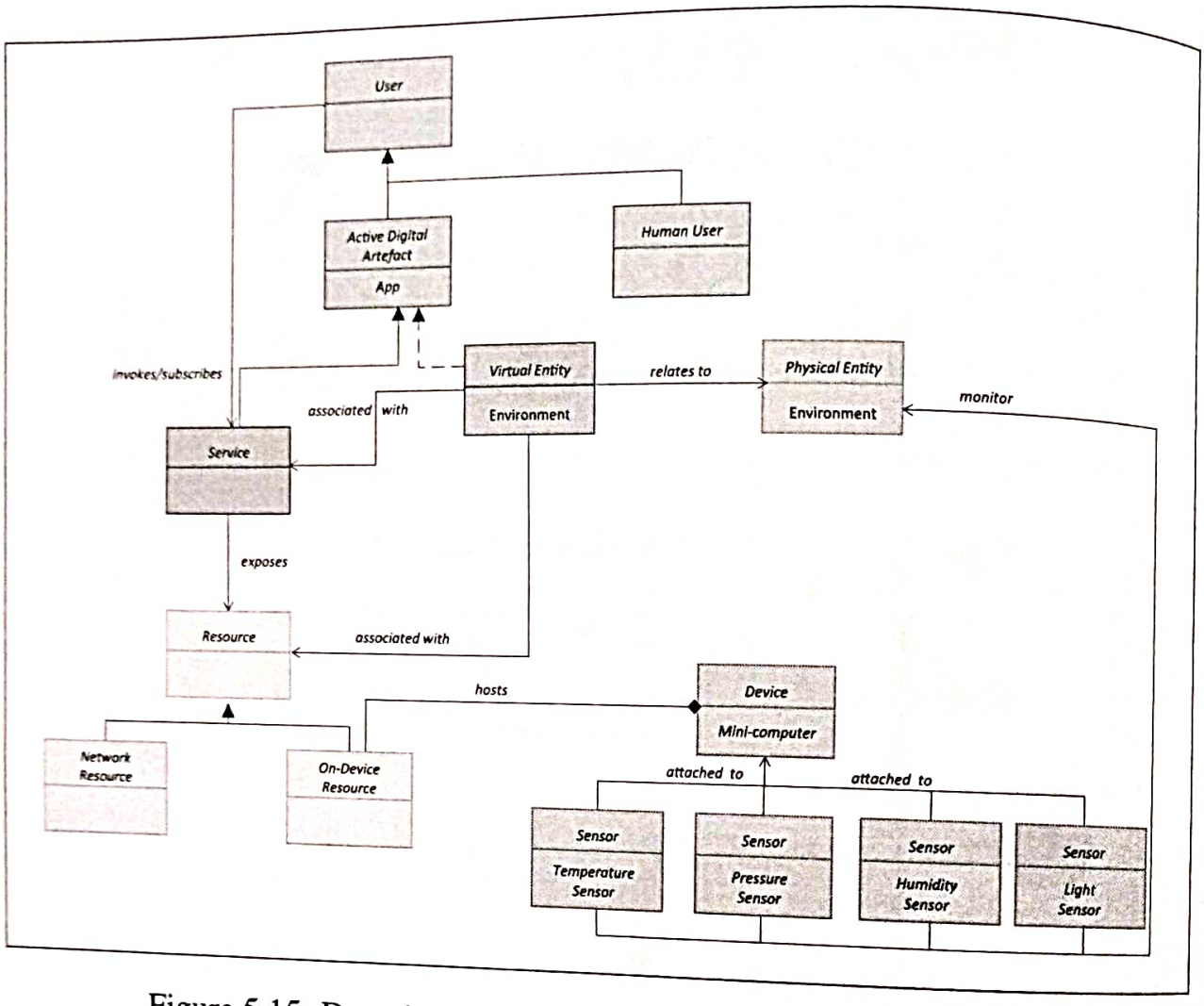


Figure 5.15: Domain model for weather monitoring IoT system

deriving the services from the process specification and information model for the weather monitoring system.

Figure 5.18 shows the specification of the controller service for the weather monitoring system. The controller service runs as a native service on the device and monitors temperature, pressure, humidity and light once every 15 seconds. The controller service calls the REST service to store these measurements in the cloud. In Chapter-8 we describe a Platform-as-a-Service called Xively that can be used for creating solutions for Internet of Things. An implementation of a controller service that calls the Xively REST API to store data in Xively cloud is described in Chapter-9.

Figure 5.19 shows the deployment design for the system. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and pressure in an area. The end nodes are equipped with various sensors (such as temperature, pressure, humidity and light). The end nodes send the data to the cloud and the data is stored in a cloud

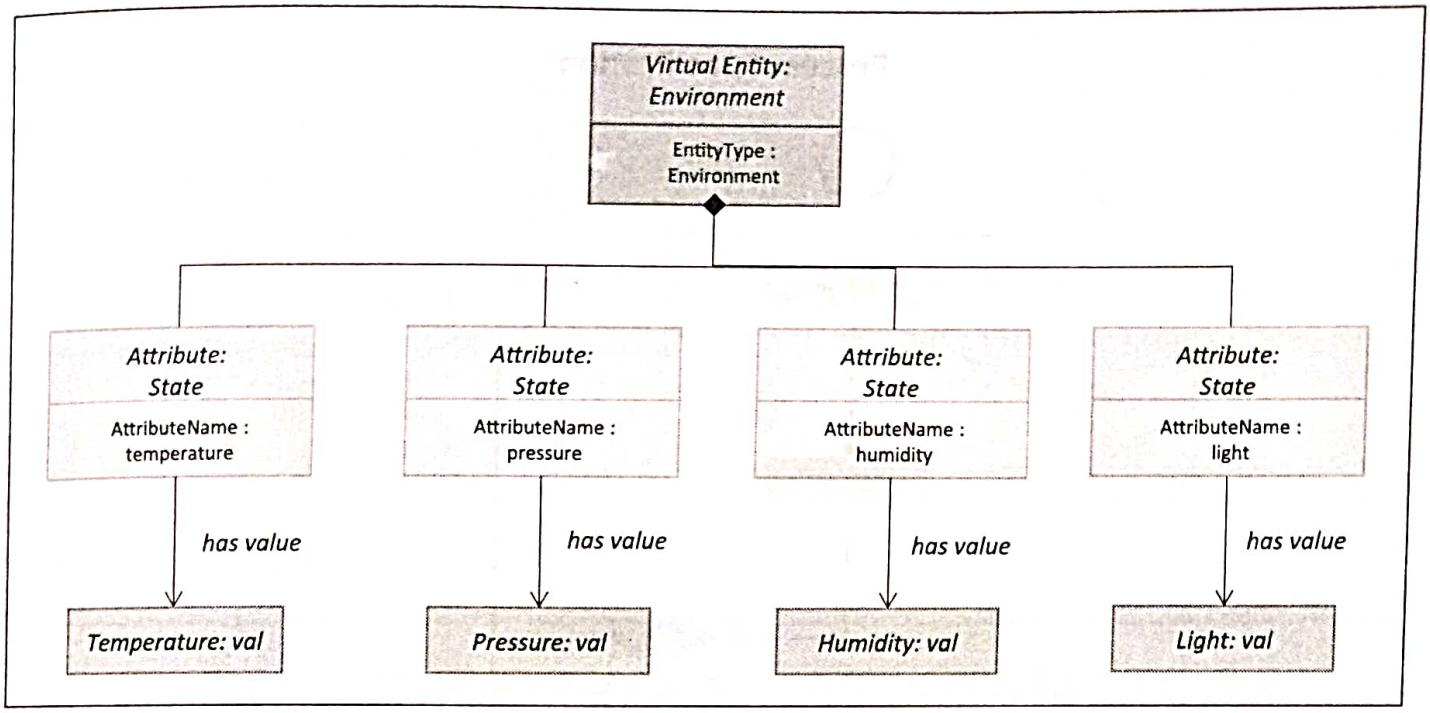


Figure 5.16: Information model for weather monitoring IoT system

database. The analysis of data is done in the cloud to aggregate the data and make predictions. A cloud-based application is used for visualizing the data. The centralized controller can send control commands to the end nodes, for example, to configure the monitoring interval on the end nodes.

Figure 5.20 shows an example of mapping deployment level to functional groups for the weather monitoring system. Figure 5.21 shows an example of mapping functional groups to operational view specifications for the weather monitoring system.

Figure 5.22 shows a schematic diagram of the weather monitoring system. The devices and components used in this example are Raspberry Pi mini computer, temperature sensor, humidity sensor, pressure sensor and LDR sensor.

5.4 Motivation for Using Python

This book uses the Python language for all the examples, though the basic principles apply to other high level languages. In this section we explain the motivation for using Python for developing IoT systems. Python is a minimalistic language with English-like keywords and fewer syntactical constructions as compared to other languages. This makes Python easier to learn and understand. Moreover, Python code is compact as compared to other languages. Python is an interpreted language and does not require an explicit compilation step. The Python interpreter converts the Python code to the intermediate byte code, specific

Unit- IV
IOT PHYSICAL DEVICES & ENDPOINT

What is an IOT devices, Exemplary Devices: Raspberry Pi, About the Board, Linux on Raspberry Pi, Raspberry Pi Interfaces, Programming Raspberry Pi with Python–Other IoT Devices.

What is an IOT devices ?

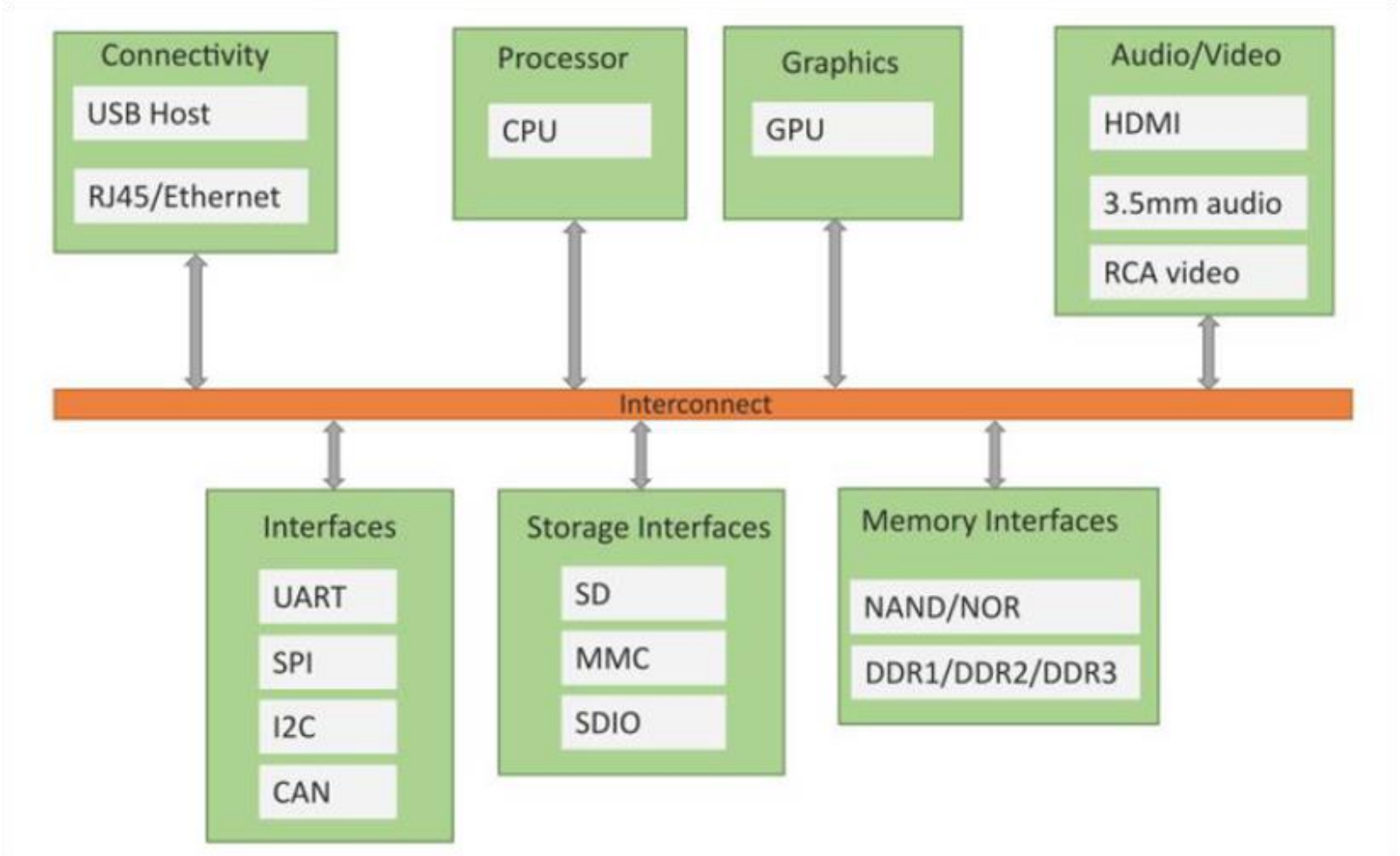
- A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smartTV, computer, refrigerator, car, etc.).
- IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely

IoT Device Examples

- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information abouts its operation and health monitoring data to a server.
- A car which sends information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

Basic building blocks of an IoT Device

- Sensing: Sensors can be either on-board the IoT device or attached to the device.
- Actuation: IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device.
- Communication: Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications.
- Analysis & Processing: Analysis and processing modules are responsible for making sense of the collected data.


Connectivity
USB Host
RJ45/Ethernet
Processor
CPU
Graphics
GPU
Audio/Video
HDMI
3.5mm audio
RCA video
Interconnect
Interfaces
UART
SPI
I2C
CAN
Storage Interfaces
SD
MMC
SDIO
Memory Interfaces
NAND/NOR
DDR1/DDR2/DDR3

Exemplary Devices: Raspberry Pi

- Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.
- Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.
- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins. Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".

About the Board:

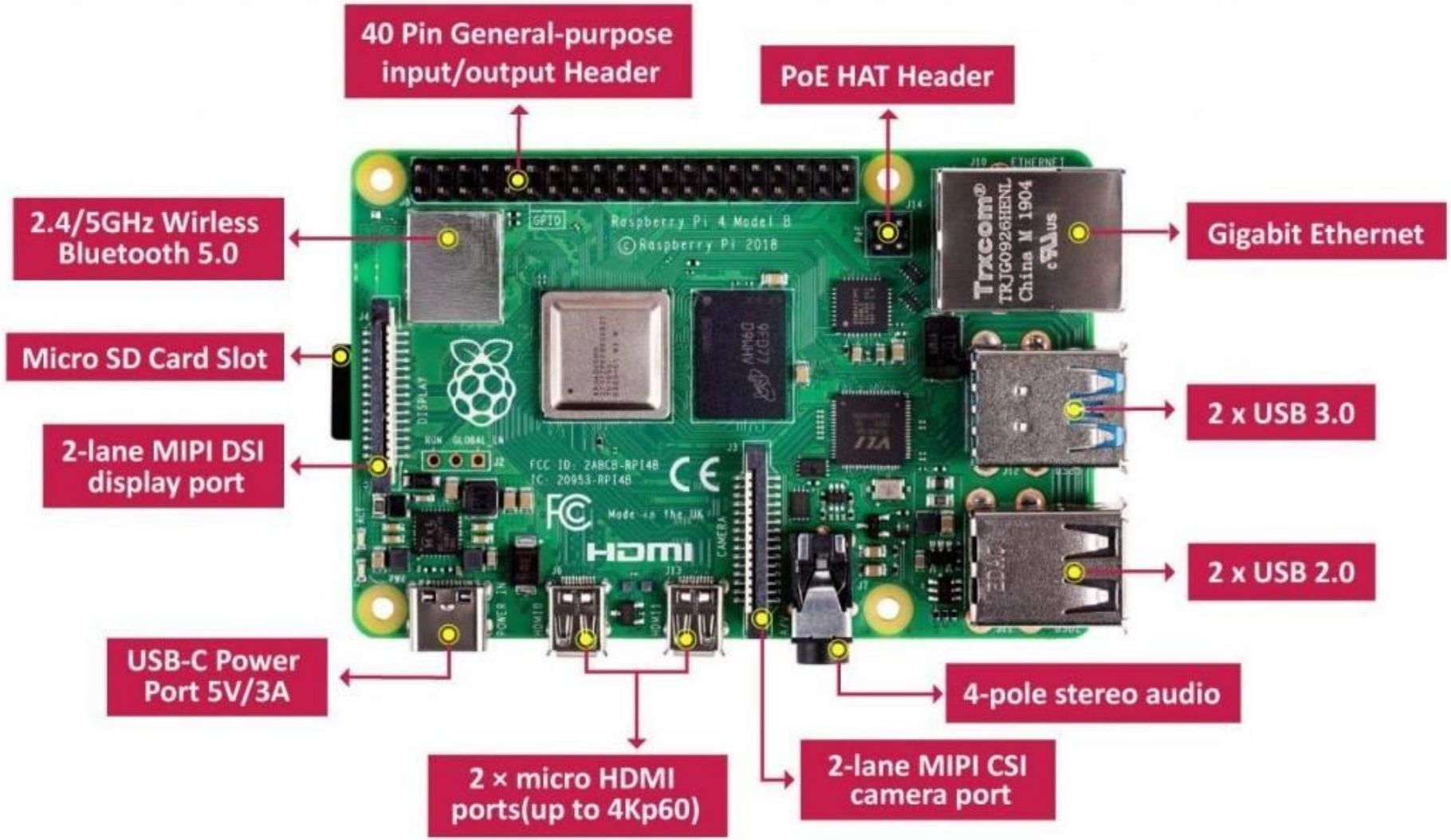
- Processor & RAM: Raspberry Pi(RPI) is based on an ARM processor. The latest version of Raspberry Pi comes with 700 MHz Low power ARM11761z-F processor and 512 MB SDRAM.
- USB Ports: RPI comes with two USB 2.0 ports. The USB ports on RPI can provide a current upto 100mA. For connecting devices that draw current more than 100mA,

- Ethernet Ports: RPI comes with standards RJ45 Ethernet Port. You can connect an Ethernet cable or a USB with adapter to provide Internet connectivity.
- HDMI Output: The HDMI port on RPI provides both video and audio output. You can connect the RPI to a monitor using an HDMI cable. For monitors that have a DVI port but no HDMI port, you can use an HDMI to DVI adapter/cable.
- Composite Video output: RPI comes with a composite video output with an RCA jack that supports both PAL and NTSC video output. The RCA jack can be used to connect old televisions that have an RCA input only.
- Audio output: Raspberry Pi has a 3.5mm audio output jack. This audio jack is used for providing audio output to old televisions along with the RCA jack for video. The audio quality from this jack is inferior to the HDMI output.

- GPIO Pins: RPI comes with a number of general-purpose input/output pins whose in RPI GPIO headers. There are four types of pins on RPI true GPIO pins. I2C interface pins, SPI interface pins and serial Rx and Tx pins.
- Display Serial Interface(DSI): The DSI interface can be used to connect an LCD panel to RPI.
- Camera Serial Interface (CSI): The CSI Interface can be used to connect a camera module to RPI.
- SD Card slot: RPI does no have a built in operating system and storage, you can plug-in an SD card loaded with a Linux image to the SD card slot. You will require at least an 8 GB SD card for setting up NOOBS (New out-of-the-Box Software).
- Power Input: RPI has a micro-USB connector for power input

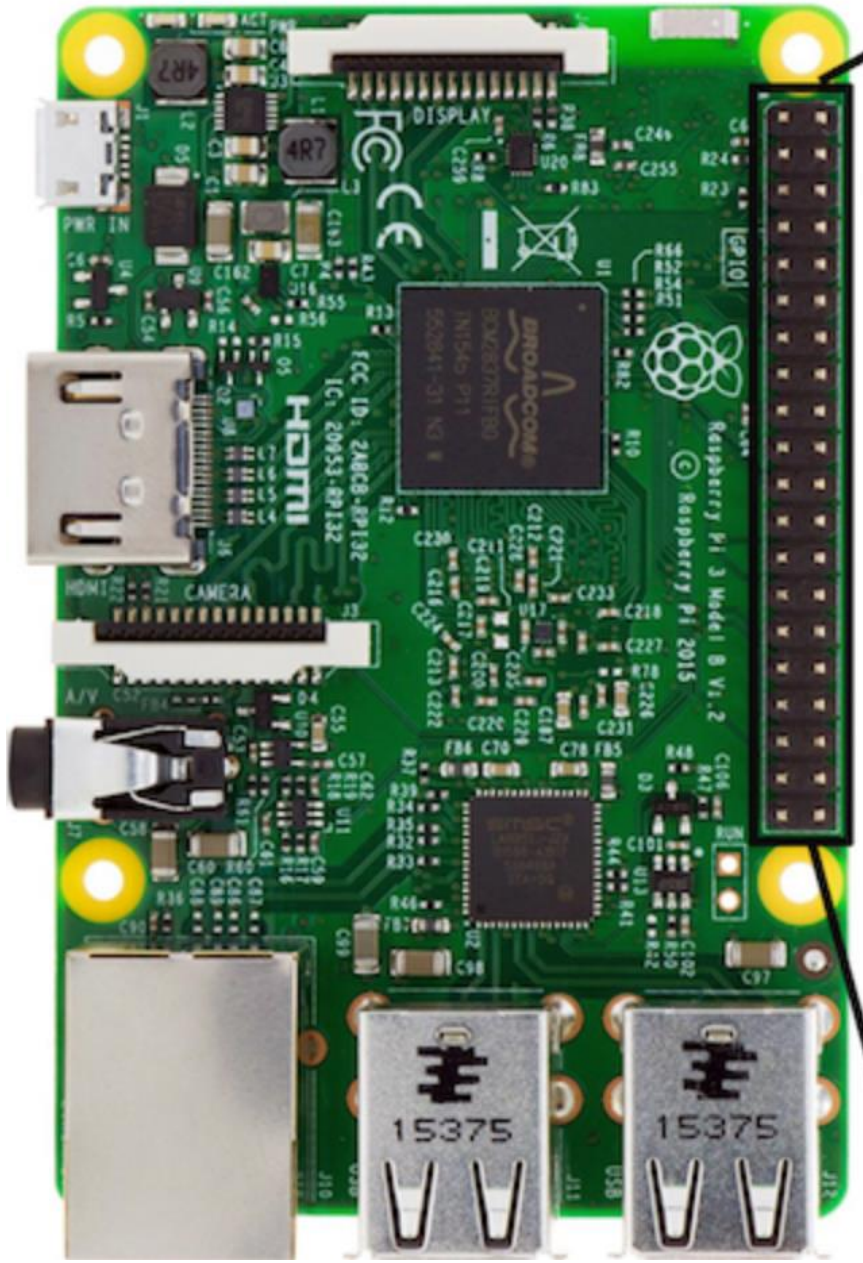
- Status LEDs: RPI has five status LEDs. Show in table.

Status LED	Function
ACT	SD card access
PWR	3.3v power is present
FDX	Full duplex LAN connected
LNK	Link / Network activity
100	100 Mbit LAN connected


40 Pin General-purpose input/output Header
PoE HAT Header
2.4/5GHz Wirless Bluetooth 5.0
Gigabit Ethernet
Micro SD Card Slot
2 x USB 3.0
2-lane MIPI DSI display port
2 x USB 2.0
USB-C Power Port 5V/3A
4-pole stereo audio
2 × micro HDMI ports(up to 4Kp60)
2-lane MIPI CSI camera port

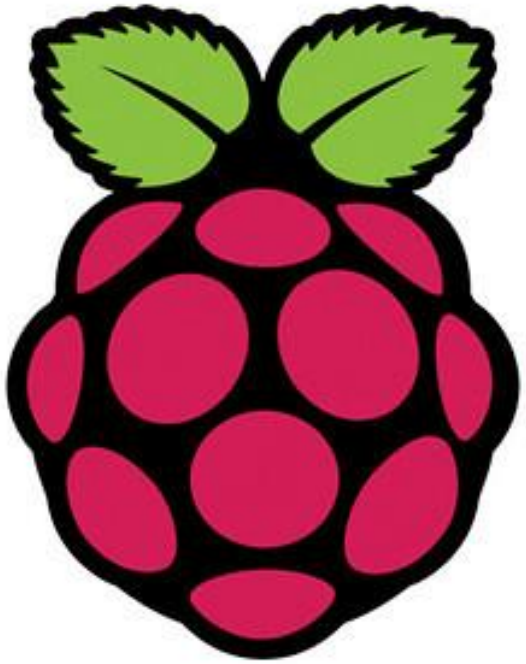
Linux on Raspberry Pi

1. Raspbian: Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.
2. Arch: Arch is an Arch Linux port for AMD devices.
3. Pidora: Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
4. RaspBMC: RaspBMC is an XBMC media-center distribution for Raspberry Pi.
5. OpenELEC: OpenELEC is a fast and user-friendly XBMC media-center distribution.
6. RISC OS: RISC OS is a very fast and compact operating system.

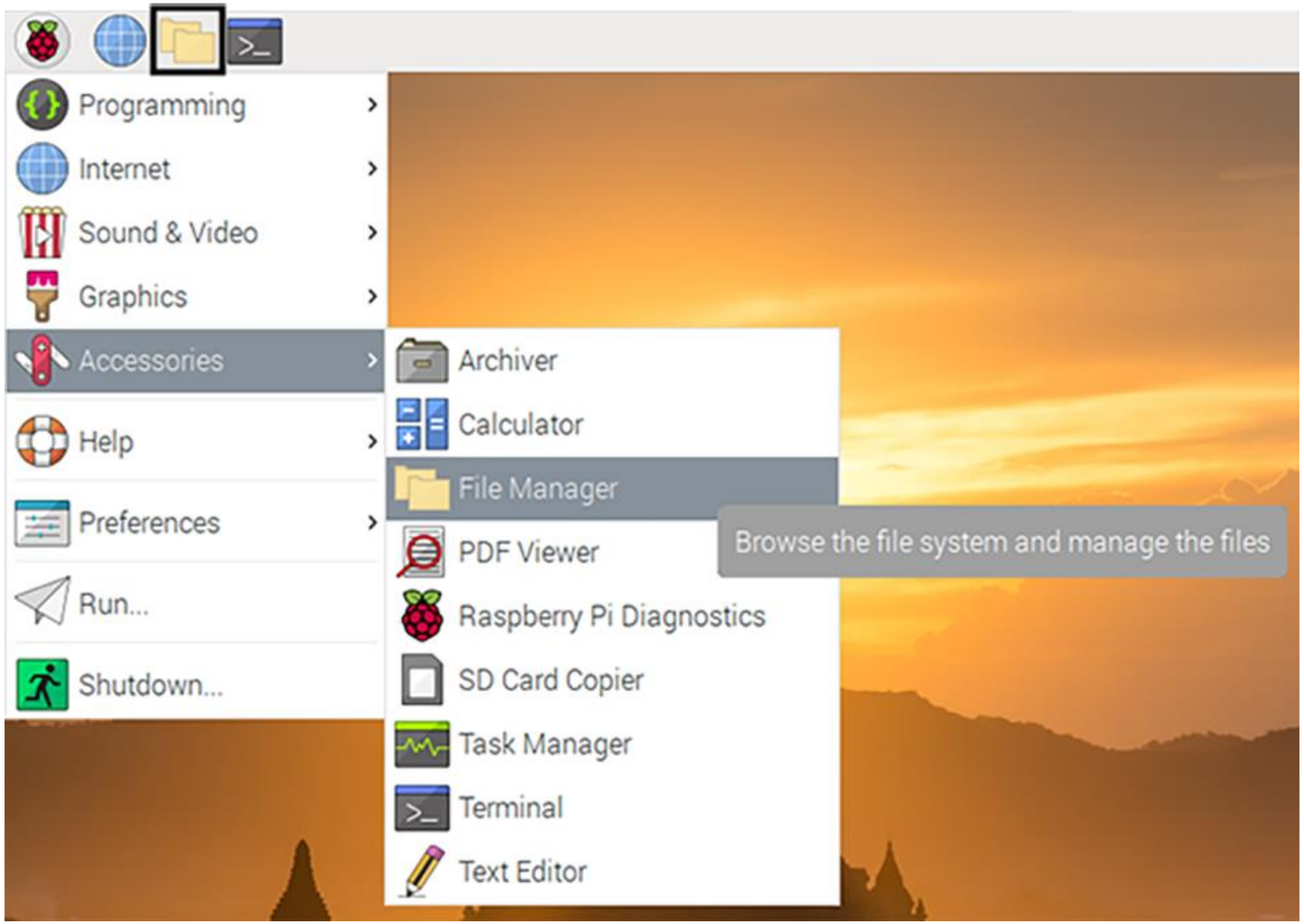


	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

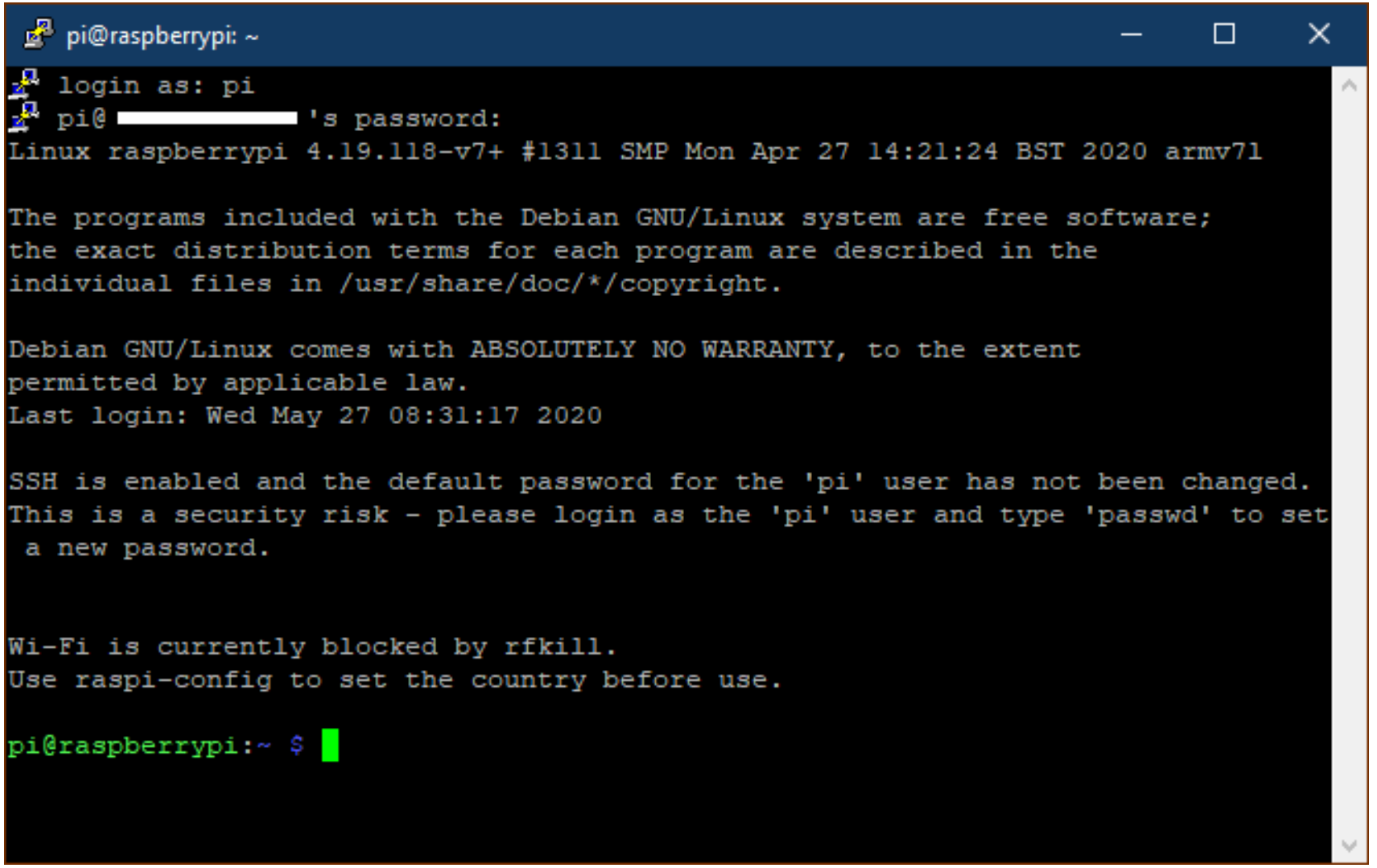
Rasbian Linux Desktop



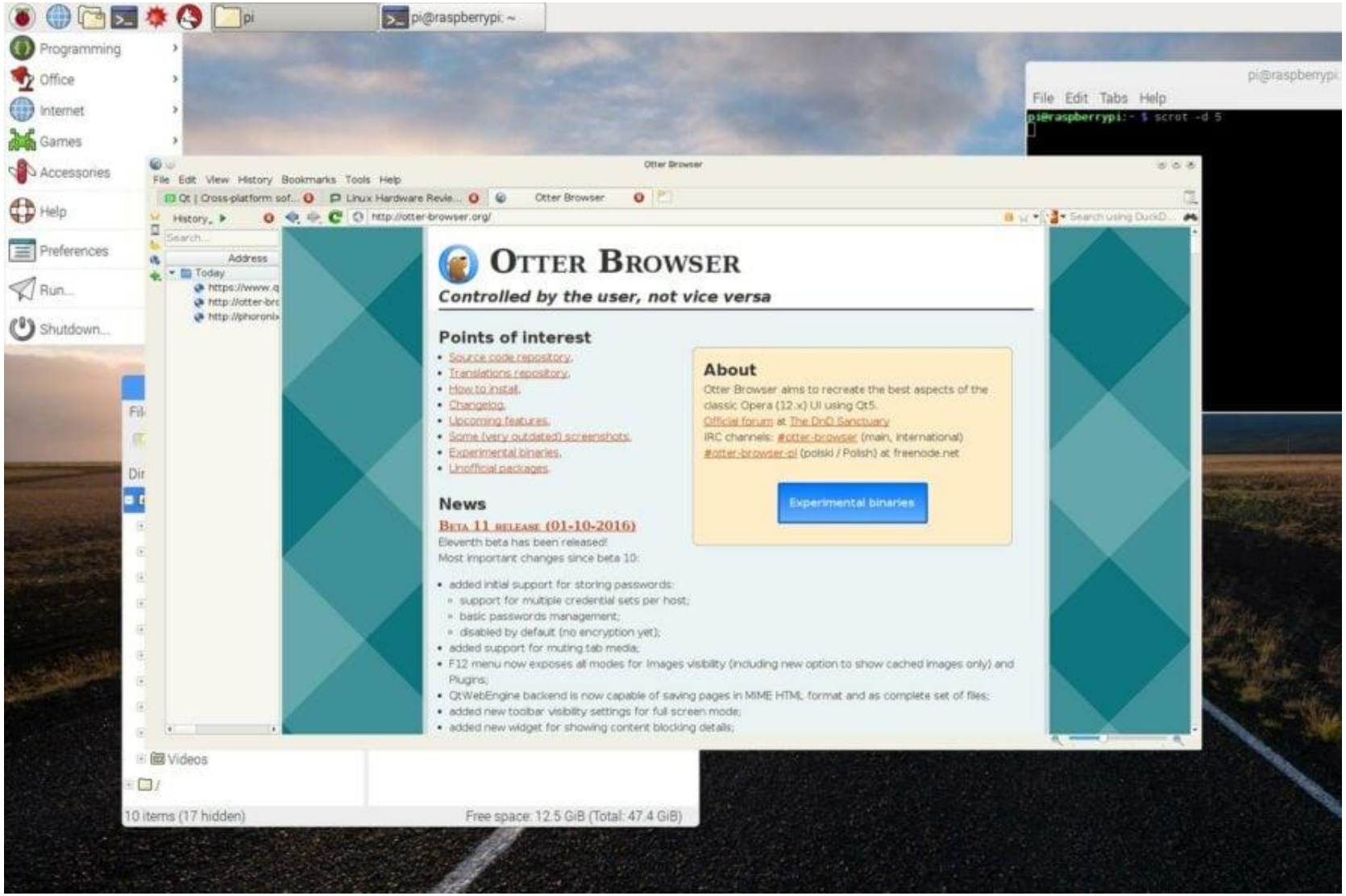
File explorer on Raspberry Pi



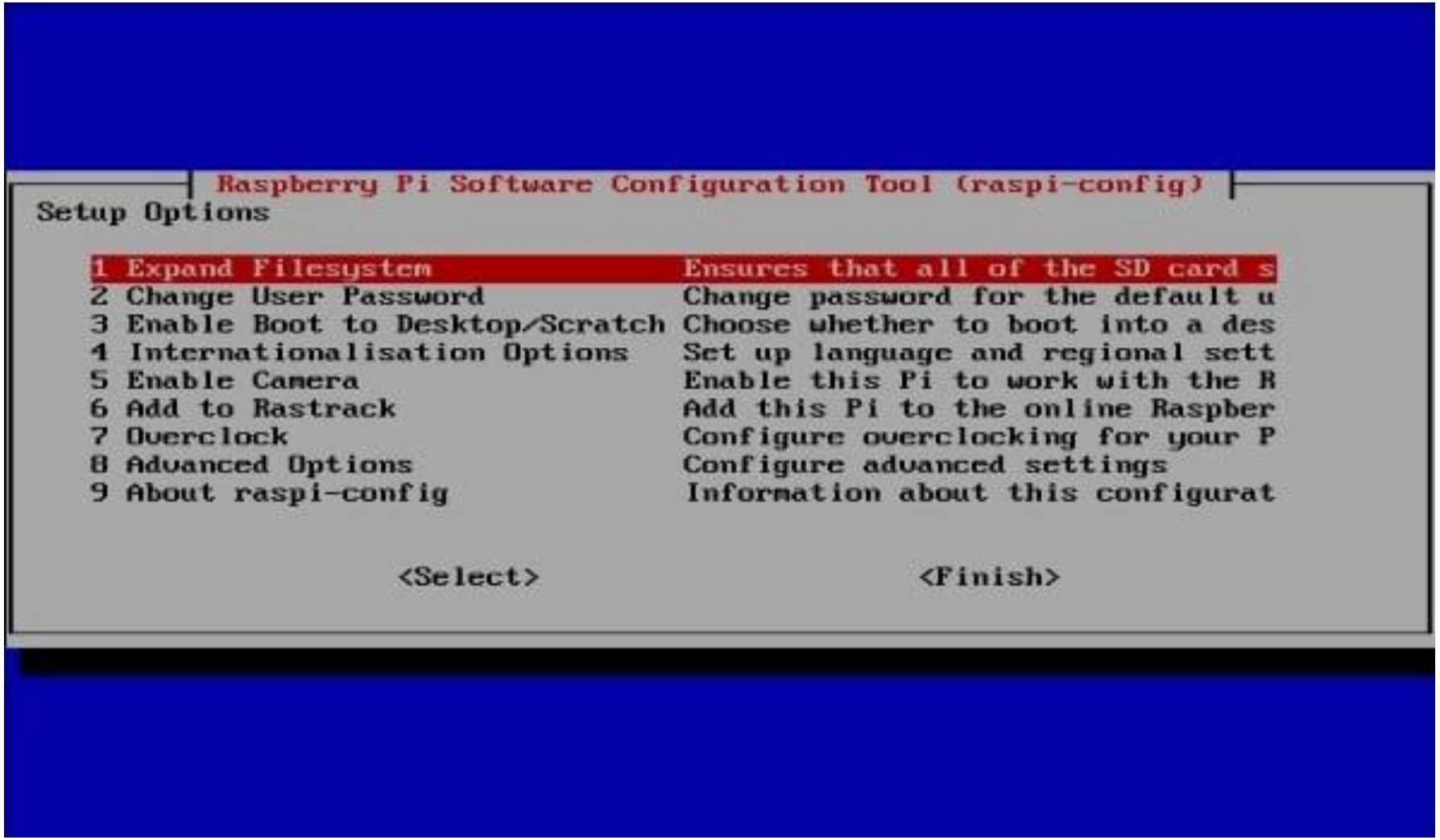
Console on Raspberry Pi



browser on Raspberry pi



Raspberry Pi Configuration tools:



Raspberry Pi frequently used commands

Command	Function	Example
cd	change directory	cd/home/pi
cat	show file contents	cat file.txt
ls	list files and folders	ls/home/pi
locate	search for a file	locate file.txt
lsusb	list usb devices	lsusb
pwd	print name for present working directory	pwd
mkdir	make directory	mkdir/home/pi/new
mv	move(rename) file	mv sourcefile.txt destfile.txt
rm	remove file	rm file.txt
reboot	reboot device	sudo reboot
shutdown	shutdown device	sudo shutdown –h now

Raspberry Pi frequently used commands

Command	Function	Example
grep	Print lines matching a pattern	grep –r "pi"/home/
df	Report file system disk space usage	df -Th
ipconfig	Configure a network interface	ipconfig
netstat	Print network connections, routing tables, interface statistics	Netstat -lntp
tar	Extract /create archive	Tar –xzf foo.tar.gz
wget	Non-interactive network downloader	Wget http://example.com/filr.tar.gz

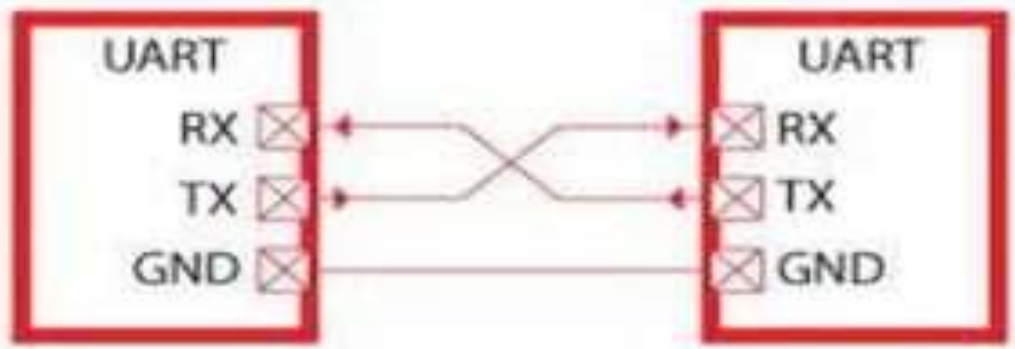
Raspberry Pi Interfaces

- Serial Interface / UART Interface
  - Universal Asynchronous Receiver and Transmitter(UART)
- SPI
  - Serial Peripheral Interface (SPI)
- I2C
  - Inter-Integrated Circuits (I2C)

Raspberry Pi Interfaces

- Serial
  - The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.
- SPI
  - Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.
- I2C
  - The I2C interface pins on Raspberry Pi allow you to connect hardware modules.
  - I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

Serial / UART



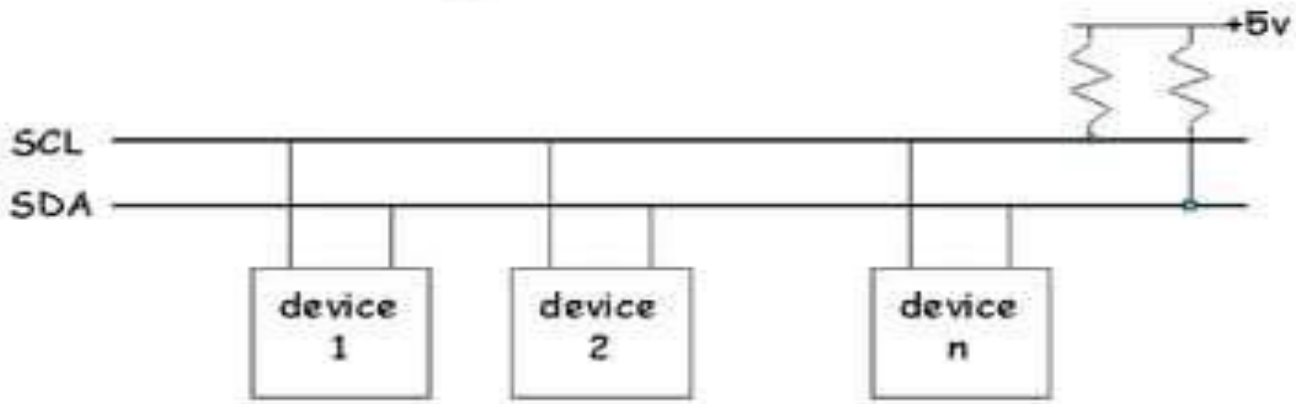
SPI

- In SPI Connection, there is one master device and one or more peripheral devices.
- There are five pins on Raspberry Pi for SPI interface.
  - MISO (Master In Slave Out) – Master Line for Sending Data to the peripherals.
  - MOSI (Master Out Slave In) – Slave Line for sending data to the master.
  - SCK (Serial Clock) – Clock Generated by Master to Synchronize data transmission.
  - CE0 (Chip Enable 0) – To Enable or Disable devices.
  - CE1 (Chip Enable 1) – To Enable or Disable devices.

I2C

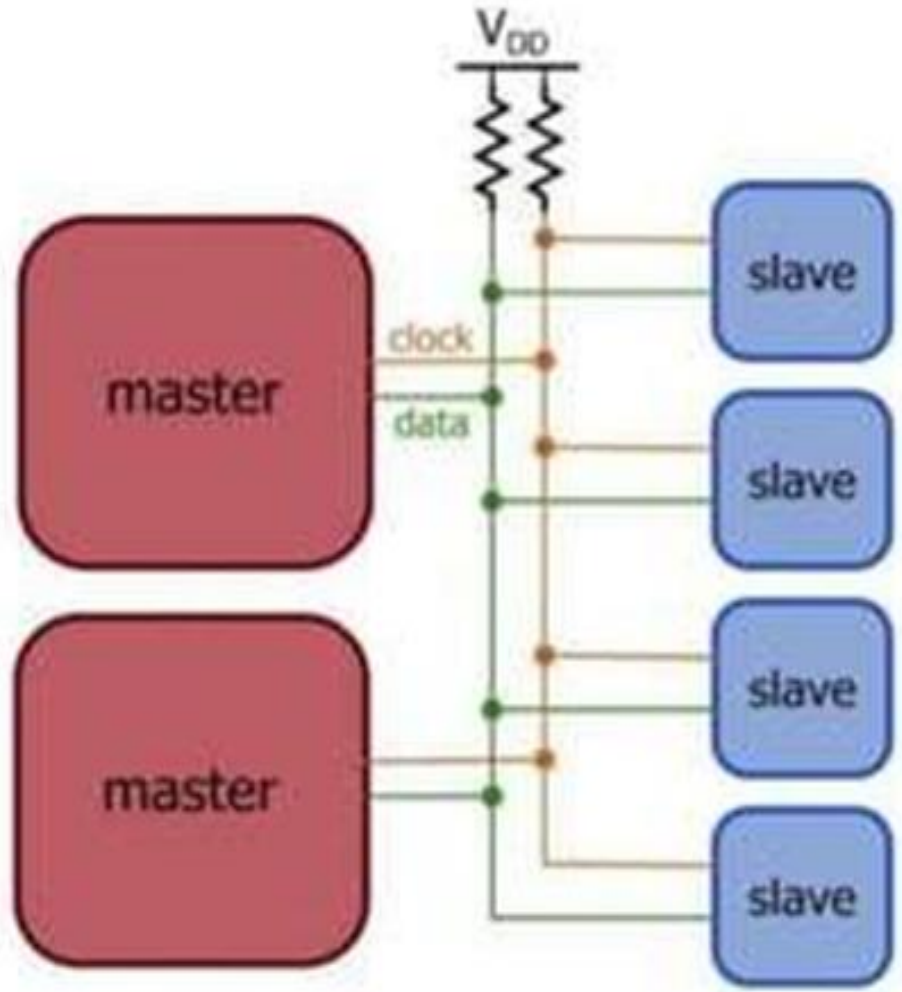
Inter-Integrated Circuit Bus (I2C)

- Modular connections on a printed circuit board
- Multi-point connections (needs addressing)
- Synchronous transfer (but adapts to slowest device)
- Similar to Controller Area Network (CAN) protocol used in automotive applications



INTER-INTEGRATED CIRCUIT (I²C)

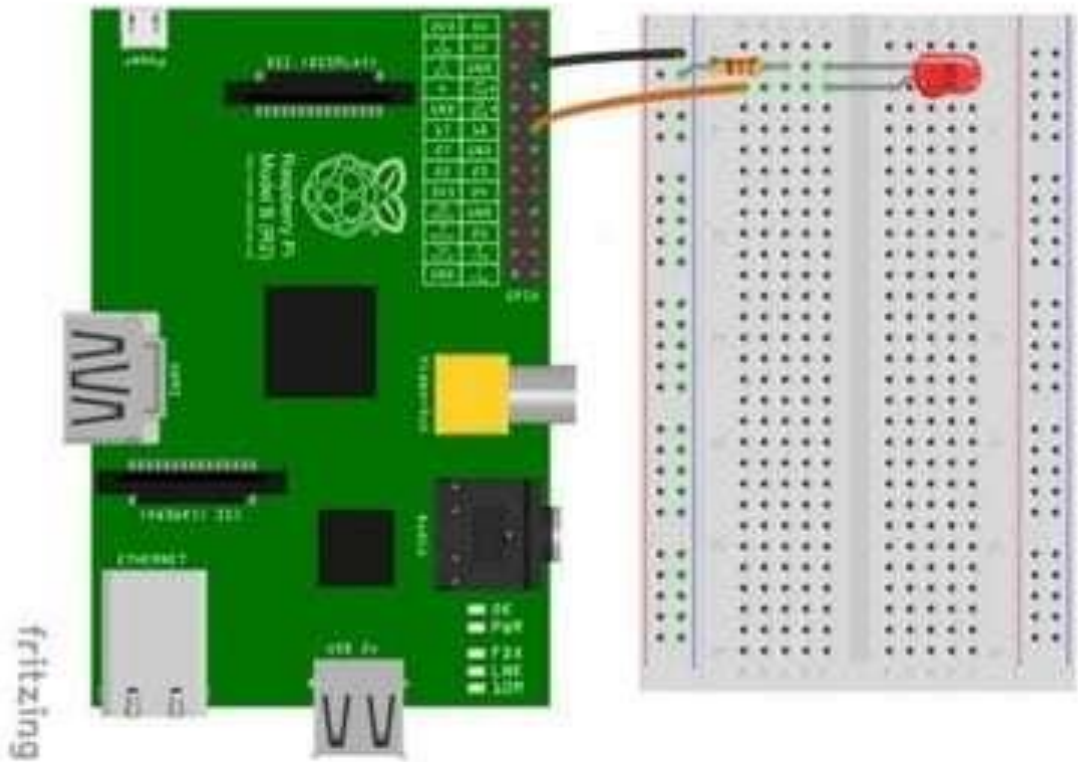
- Serial, 8-bit oriented, Bidirectional Half Duplex, 2-wire bus used for communications between integrated circuits on the same PCB.
- Developed by Philips Semiconductors in 1982 (now NXP Semiconductors).
- Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL).
- Master device generates the clock signal and terminates the transfer.
- Each device is software addressable by a unique address.
- Master/Slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers.
- True multi-master bus including collision detection, arbitration (using wired-AND) and clock synchronization to prevent data corruption.



Python Programming with Raspberry Pi

- Python programming with Raspberry PI with focus of
  - Interfacing external gadgets
  - Controlling output
  - Reading input from pins
- GPIO pins on Raspberry Pi that makes it useful device for IoT.
- We can interface a wide variety of sensors and actuators with Raspberry Pi using the GPIO pins and SPI, I2C and Serial Interfaces.
- Input from the sensors connected to Raspberry Pi can be processed and various actions can be taken, for
  - Instance,
  - Sending data to a server
  - Sending an email
  - Triggering a relay switch

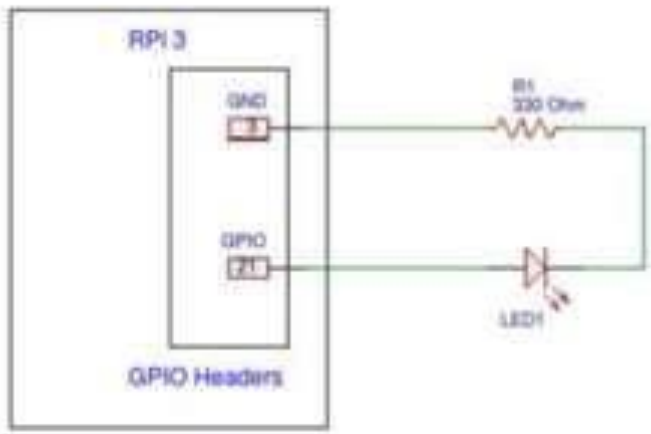
Controlling LED with Raspberry Pi



Components Reuired

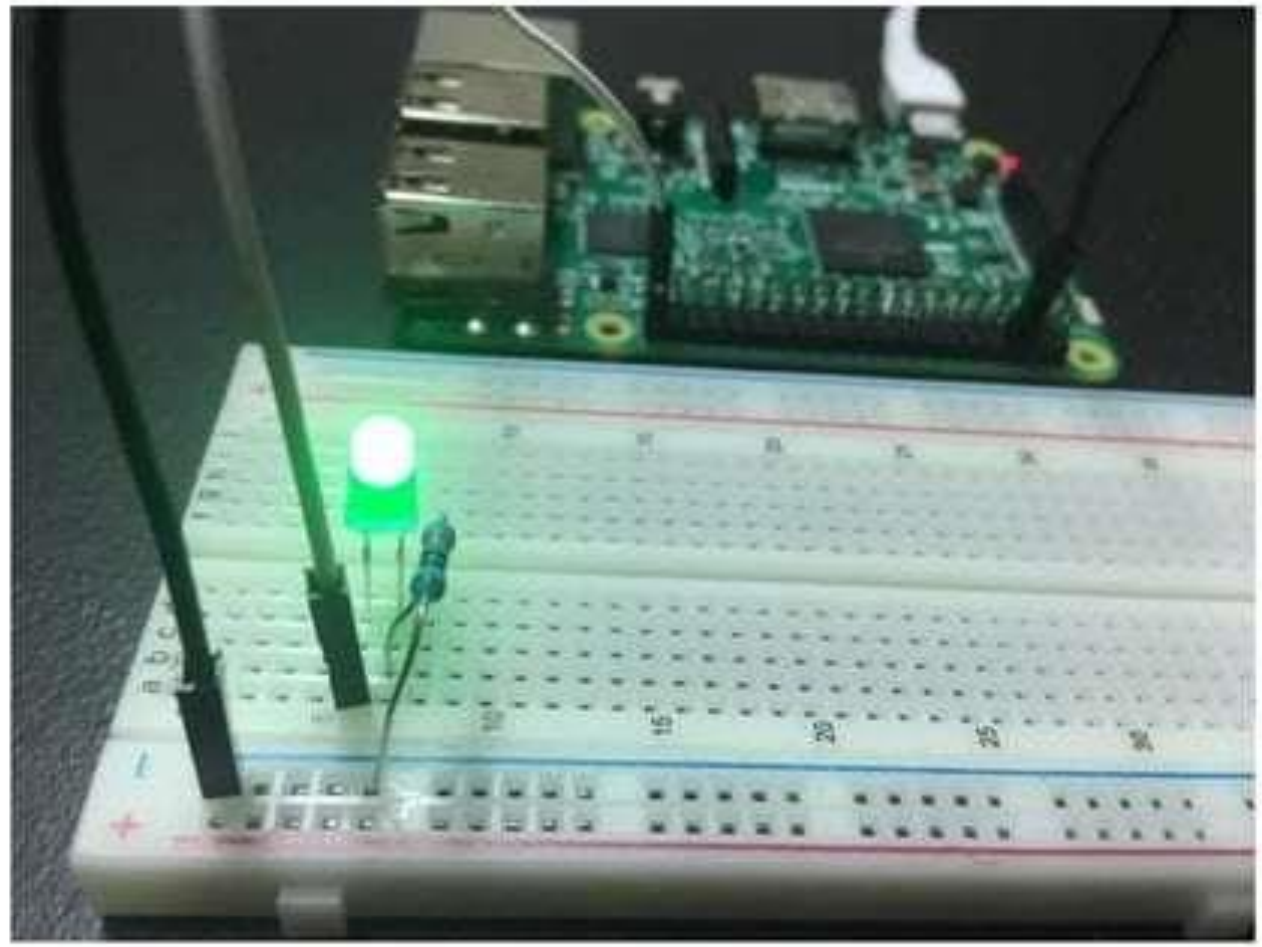
- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. LED
  3. Resistor - 330 ohm
  4. Breadboard
  5. 2 Male-Female Jumper Wires

Step 2: Connecting the Circuit



Python Code

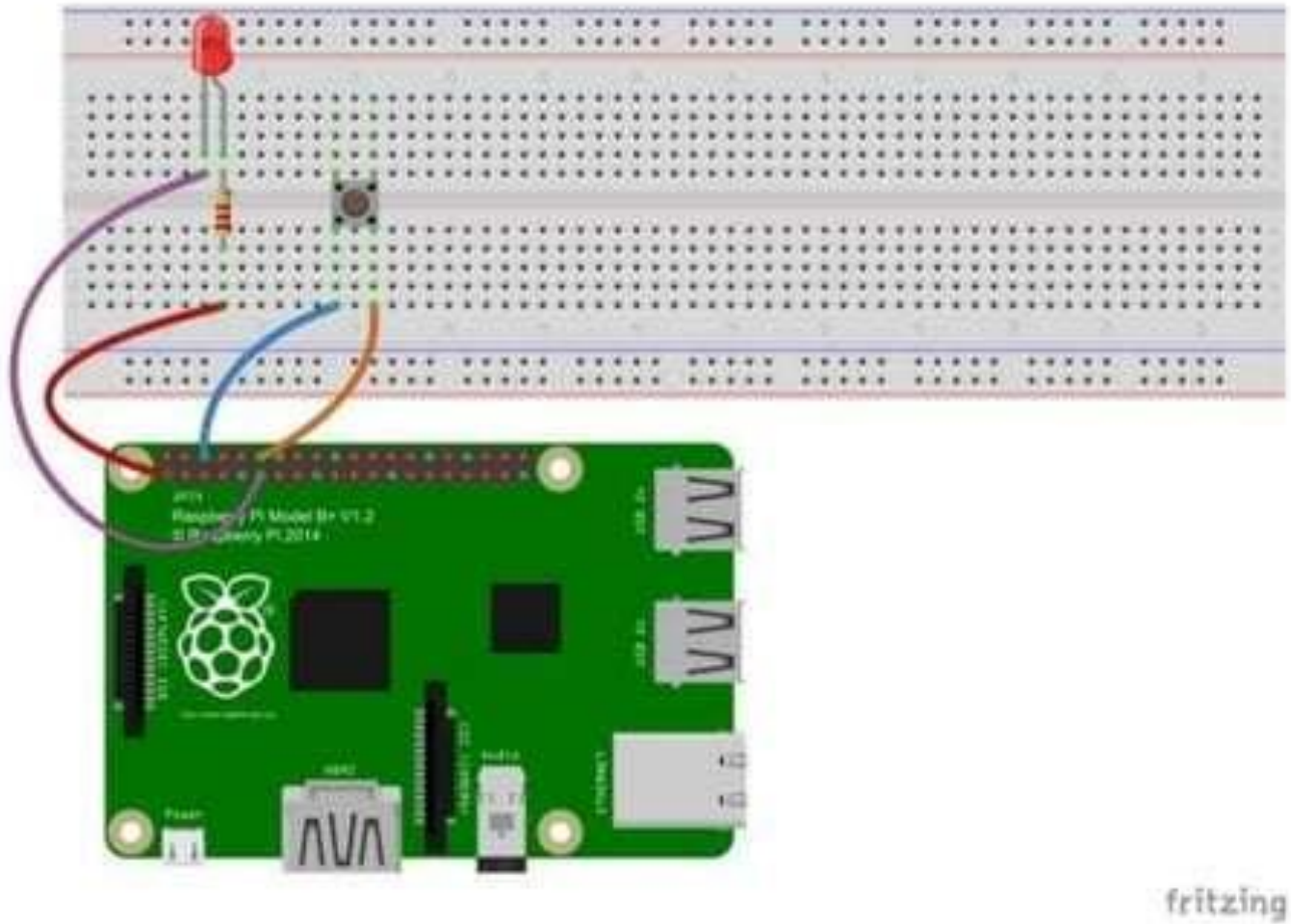
- import RPi.GPIO as GPIO
  import time
  GPIO.setmode(GPIO.BCM)
  GPIO.setwarnings(False)
  GPIO.setup(21,GPIO.OUT)
  print "LED on"
  GPIO.output(21,GPIO.HIGH)
  time.sleep(10)
  print "LED off"
  GPIO.output(21,GPIO.LOW)



Python programming with Raspberry PI with focus of

- Interfacing external gadgets
- Controlling output
- Reading input from pins

Controlling LED with Switch on Raspberry Pi



Components Reuired

- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. LED
  3. Resistor - 330 ohm
  4. Breadboard
  5. Button / Switch
  6. 2 Male-Female Jumper Wires

Components

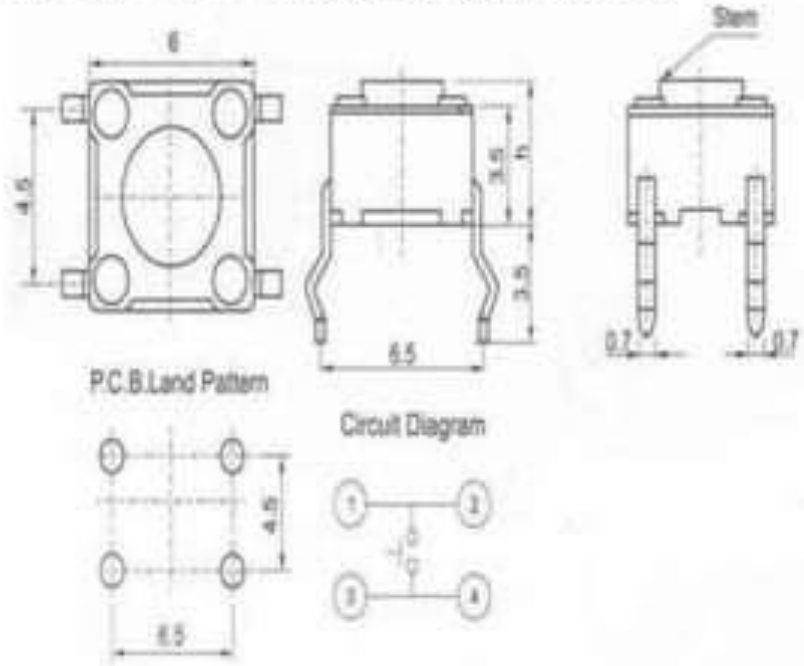
Switch / Button

- Buttons are a common component used to control electronic devices.
- They are usually used as switches to connect or disconnect circuits.
- Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures.
- Pins pointed out by the arrows of same color are meant to be connected.

Switch / Button

Switch / Button

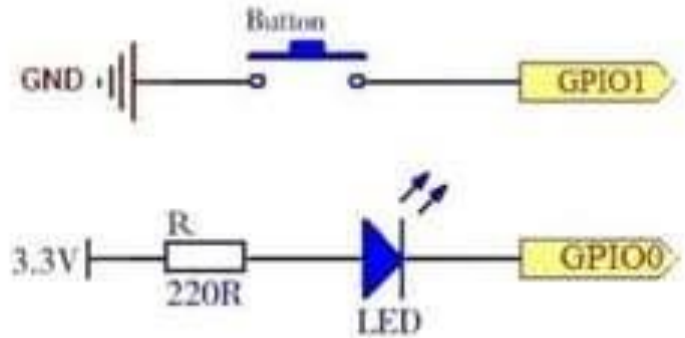
- When the button is pressed, the pins pointed by the blue arrow will connect to the pins pointed by the red arrow (see the above figure), thus closing the circuit, as shown in the following diagrams.



Switch / Button

- Generally, the button can be connected directly to the LED in a circuit to turn on or off the LED, which is comparatively simple.
- However, sometimes the LED will brighten automatically without any button pressed, which is caused by various kinds of external interference.
- In order to avoid this interference, a pull-down resistor is used – usually connect a 1K–10KΩ resistor between the button and GND. It can be connected to GND to consume the interference when the button is off.
- Use a normally open button as the input of Raspberry Pi.
- When the button is pressed, the GPIO connected to the button will turn into low level (0V).
- We can detect the state of the GPIO connected to the button through programming. That is, if the GPIO turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up.

CIRCUIT Diagram



Experimental Procedures

- **Step 1**: Build the circuit
- **Step 2:** Create a Python / C code
- **Step 3:** Run

  sudo python 02_btnAndLed.py

- Now, press the button, and the LED will light up; press the button again, and the LED will go out. At the same time, the state of the LED will be printed on the screen.

Python Code

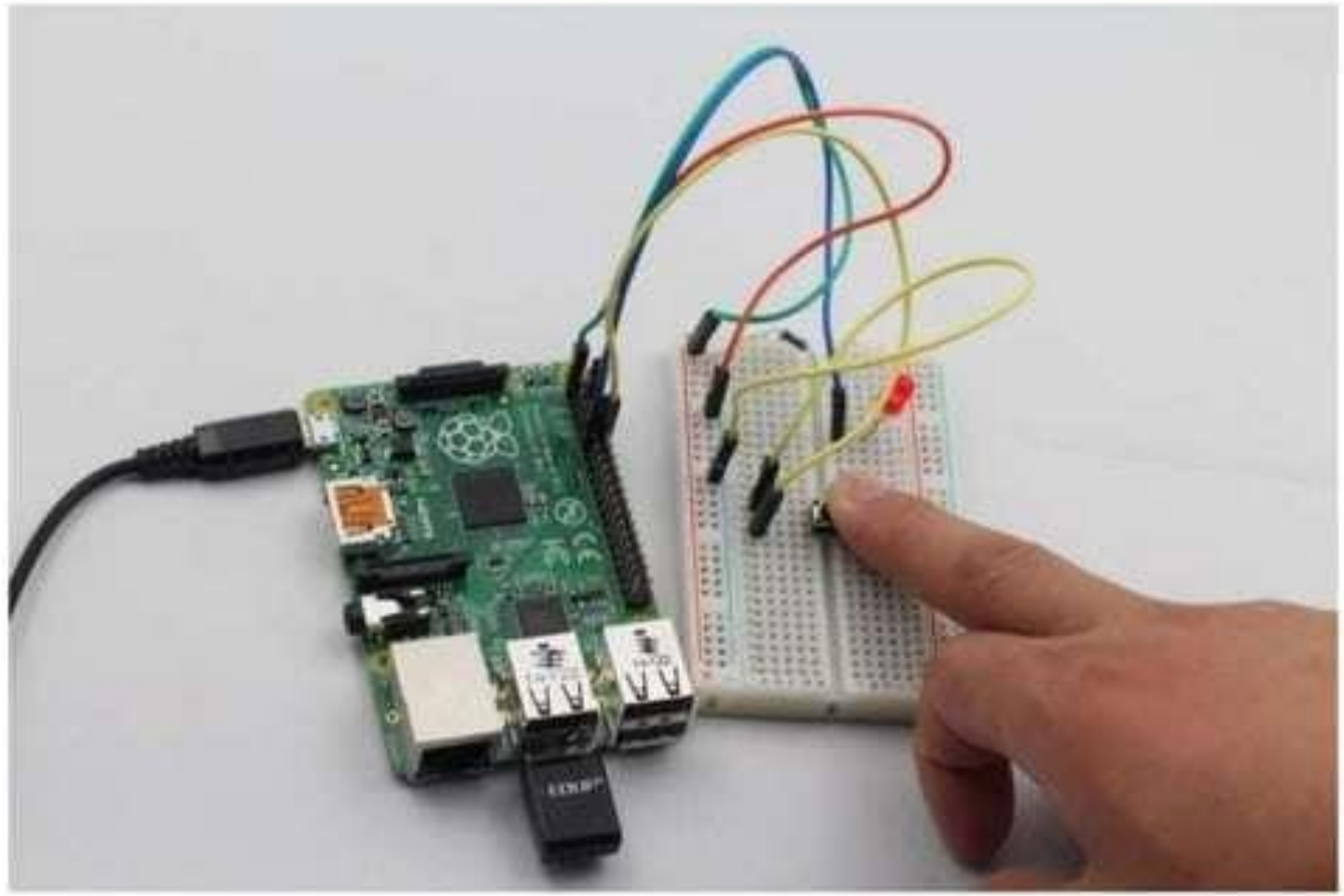
- #!/usr/bin/env python
- import RPi.GPIO as GPIO
- import time
- LedPin = 11 # pin11 --- led
- BtnPin = 12 # pin12 --- button
- Led_status = 1
- def setup():
  - GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
  - GPIO.setup(LedPin, GPIO.OUT) # Set LedPin's mode is output
  - GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Set BtnPin's mode is input, and pull up to high level(3.3V)
  - GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

- def swLed(ev=None):
  - global Led_status
  - Led_status = not Led_status
  - GPIO.output(LedPin, Led_status) # switch led status(on-->off; off-->on)
  - if Led_status == 1:
    - print 'led off...'
  - else:
    - print '...led on'

Python Code (Contd.,)

- def loop():
  - GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed, bouncetime=200) # wait for falling and set bouncetime to prevent the callback function from being called multiple times when the button is pressed
  - while True:
    - time.sleep(1) # Don't do anything
- def destroy():
  - GPIO.output(LedPin, GPIO.HIGH) # led off
  - GPIO.cleanup() # Release resource
- if __name__ == '__main__': # Program start from here
  - setup()
  - try:
    - loop()
  - except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy() will be executed.
    - destroy()

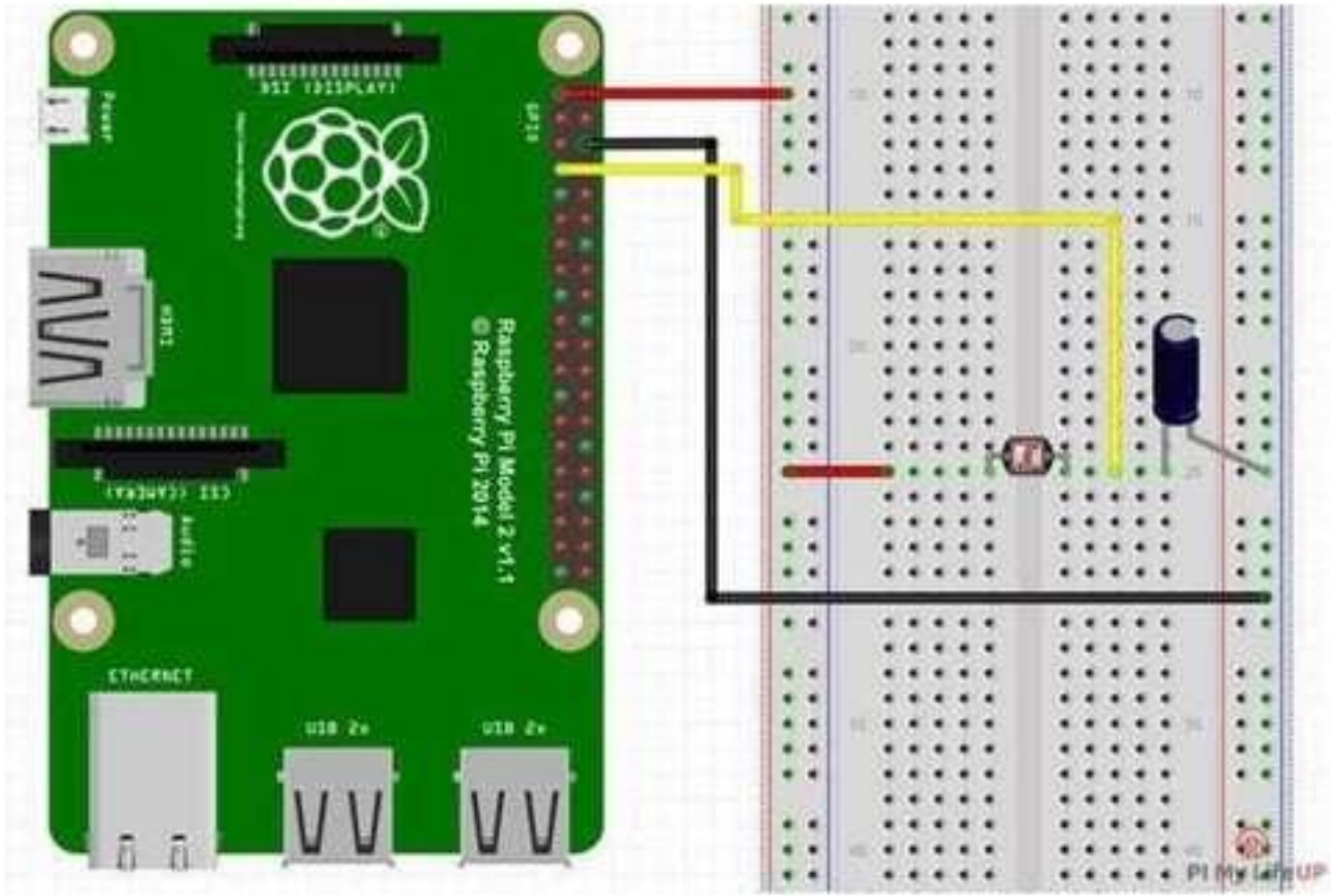
Interfacing



Applications

- Interfacing a Push Button with Raspberry Pi might not seem as a big project but it definitely helps us in understanding the concept of reading from Input pins.
- A similar concept can be applied to other input devices like different types of Sensors
  - PIR Sensor,
  - Ultrasonic Sensor,
  - Touch Sensor, and so forth.

Interfacing a Light Sensor (LDR) with Raspberry Pi



Components Reuired

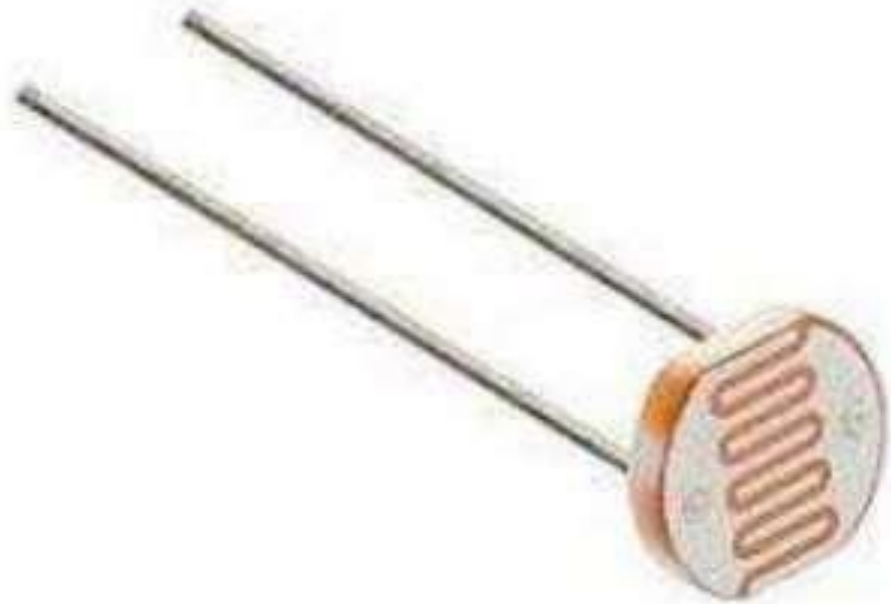
- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. Light Sensor (LDR Sensor)
  3. Capacitor - 1µF
  4. Breadboard
  5. 2 Male-Female Jumper Wires

Components

LDR Sensor

- The **light-dependent resistor** or also known as the LDR sensor is the most important piece of equipment in our circuit. Without it, we wouldn't be able to detect whether it is dark or light.
- In the **light**, this sensor will have a resistance of only a **few hundred ohms.**
- In the **dark**, it can have a resistance of several **mega ohms**.

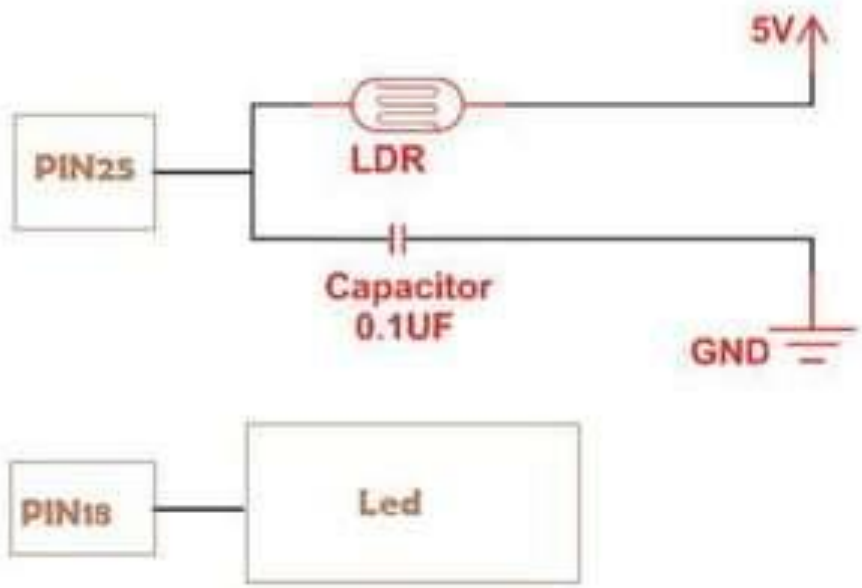
LDR Sensor



LDR Sensor

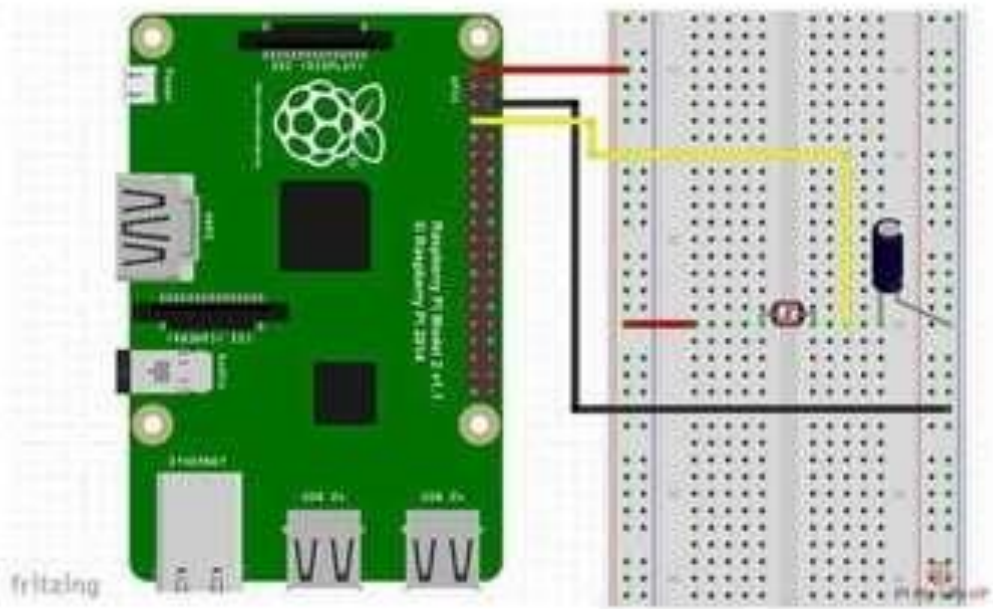
- A **photoresistor**, or light-dependent resistor (LDR), or photocell is a resistor whose **resistance will decrease when incident light intensity increase**; in other words, it exhibits photoconductivity.
- A photoresistor is made of a **high resistance semiconductor.**
- If light falling on the device is of high enough frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electron (and its hole partner) conduct electricity, thereby lowering resistance.

CIRCUIT Diagram



LDR Sensor

- In this implementation we learn how to Interface a light Sensor (LDR) with Raspberry Pi and turning an LED on/off based on the ligh-level sensed.
- Look given image, Connect one side of LDR to 3.3V and other side to a1µF capacitor and also to a GPIO pin (pin 18 in this example).
- An LED is connected to pin 18 which is controlled based on the light-level sensed.
- The readLDR() function returns a count which is proportional to the light level.
- In this function the LDR pin is set to output and low and then to input.
- At this point the capacitor starts charging through the resistor (and a counter is started) until the input pin reads high (this happens when capacitor voltage becomes greater than 1.4V).
- The counter is stopped when the input reads high. The final count is proportional to the light level as greater the amount of light, smaller is the LDR resistance and greater is the time taken to charge the capacitor.


fritzing

Experimental Procedures

- **Step 1**: Build the circuit
- **Step 2:** Create a Python / C code
- **Step 3:** Run

  sudo python 03_ldrAndLed.py

Importing Packages

- To begin, we import the **GPIO** package that we will need so that we can communicate with the GPIO pins.
- We also import the **time** package, so we're able to put the script to sleep for when we need to.

```
import RPi.GPIO as GPIO
import time
```

Pin Assignment

- We then set the GPIO mode to GPIO.BCM, and this means all the numbering we use in this script will refer to the physical numbering of the pins.
- Since we only have one **input/output** pin, we only need to set one variable. Set this variable to the number of the pin you have acting as the input/output pin.

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

Reading the Light Value

- Next, we have a function called readLDR() that requires one parameter, which is the pin number to the circuit. In this function, we initialize a variable called **reading**, and we will return this value once the pin goes to high.
- We then set our pin to act as an output and then set it to low. Next, we have the script sleep for 10ms.
- After this, we then set the pin to become an input, and then we enter a **while loop**. We stay in this loop until the pin goes to high, this is when the capacitor charges to about 3/4.
- Once the pin goes high, we return the count value to the main function. You can use this value to turn on and off an LED, activate something else, or log the data and keep statistics on any variance in light.

```
def readLDR(PIN):
    reading = 0
    GPIO.setup(LIGHT_PIN, GPIO.OUT)
    GPIO.output(PIN, false)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
    while (GPIO.input (PIN) ==Flase):
        reading=reading+1
    return reading
```

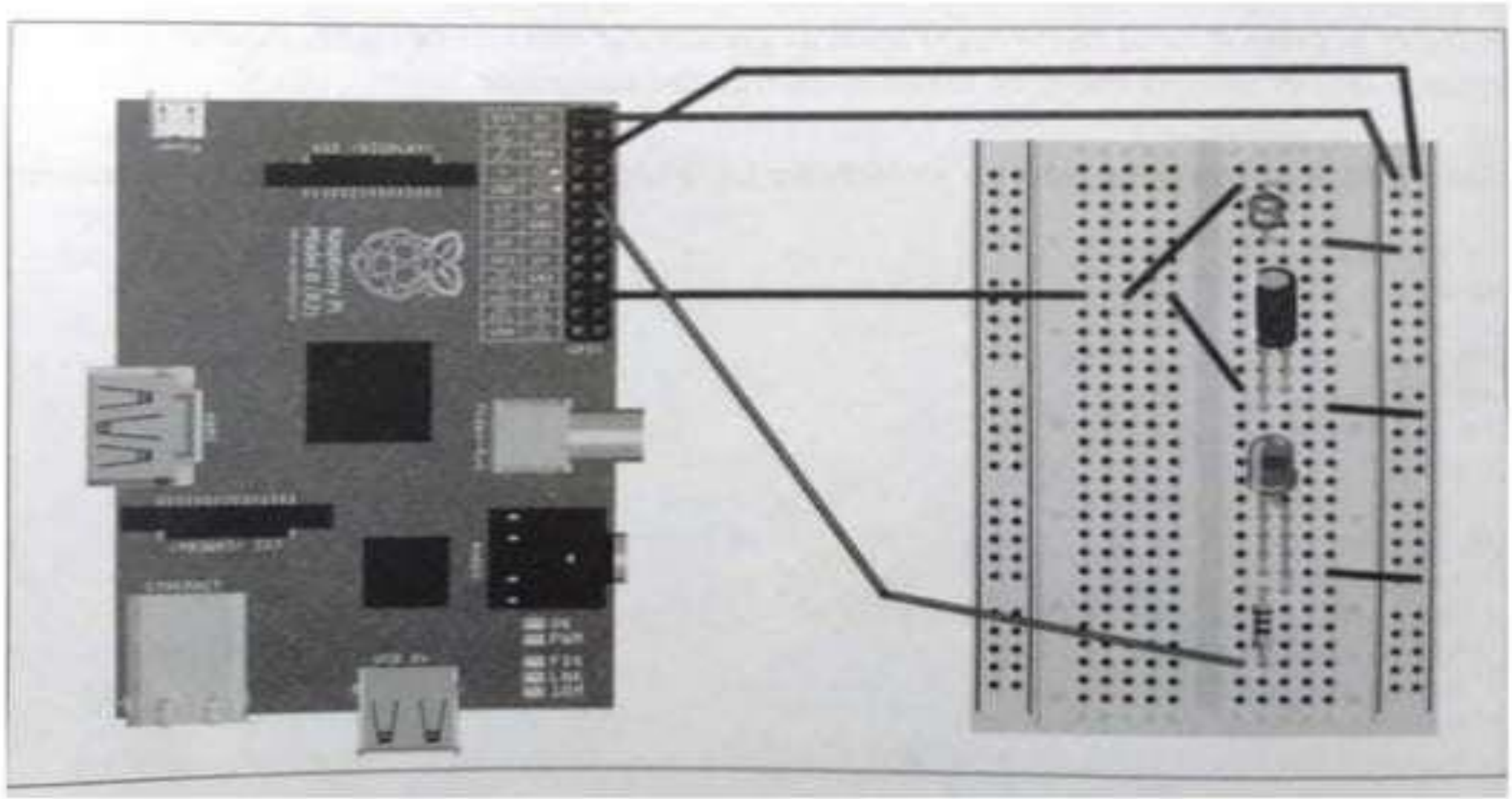
Python Code

```
# Example code Interfacing a light Sensor (LDR) with Raspberry Pi
import RPi.GPIO as GPIO
import time

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

def readLDR(PIN):
        reading = 0
        GPIO.setup(LIGHT_PIN, GPIO.OUT)
        GPIO.output(PIN, false)
        time.sleep(0.1)
        GPIO.setup(PIN, GPIO.IN)
        while (GPIO.input (PIN) ==Flase):
                reading=reading+1
        return reading
```

Python Code to Toggle the LED

```
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)
```

Interfacing



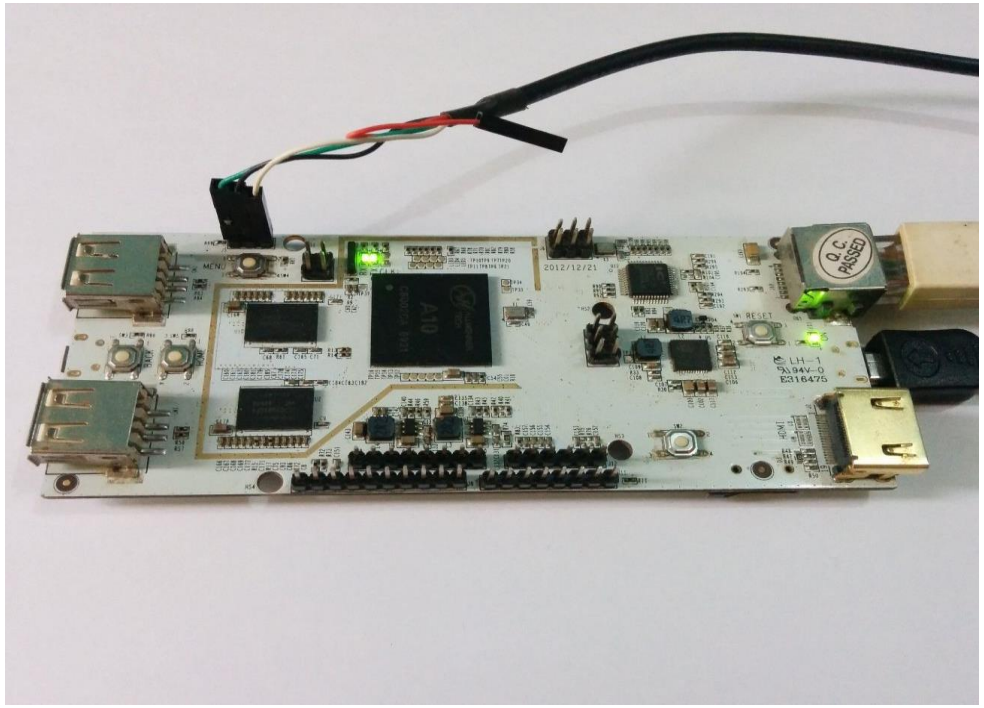
Applications

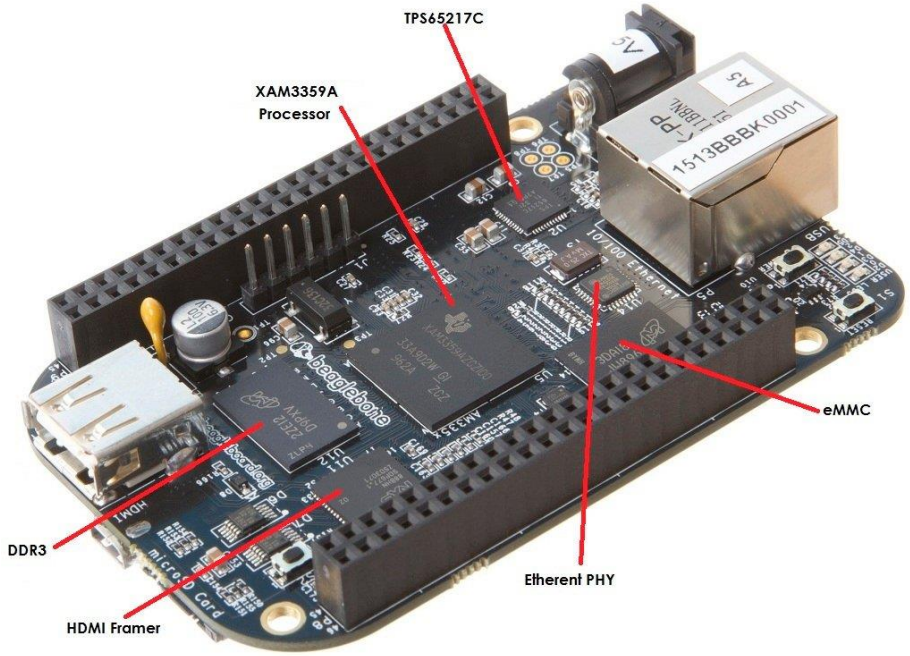
- There are countless uses for a light sensor in a circuit. I will just name a few that I thought of while I was writing up this PPT.
- **Light Activated Alarm** – I mentioned this one earlier, but you can use the LDR to detect when it starts to get light so you can sound an alarm to wake you up. If the program and sensor are accurate, then you can have the alarm slowly get louder as it gets lighter.
- **Garden monitor** – A light sensor could be used in a garden to check how much sun a certain area of the garden is getting. This could be useful information if you're planting something that needs lots of sun or vice versa.
- **Room Monitor** – Want to make sure lights are always turned off in a certain room? You could use this to alert you whenever some light is detected where it shouldn't be.

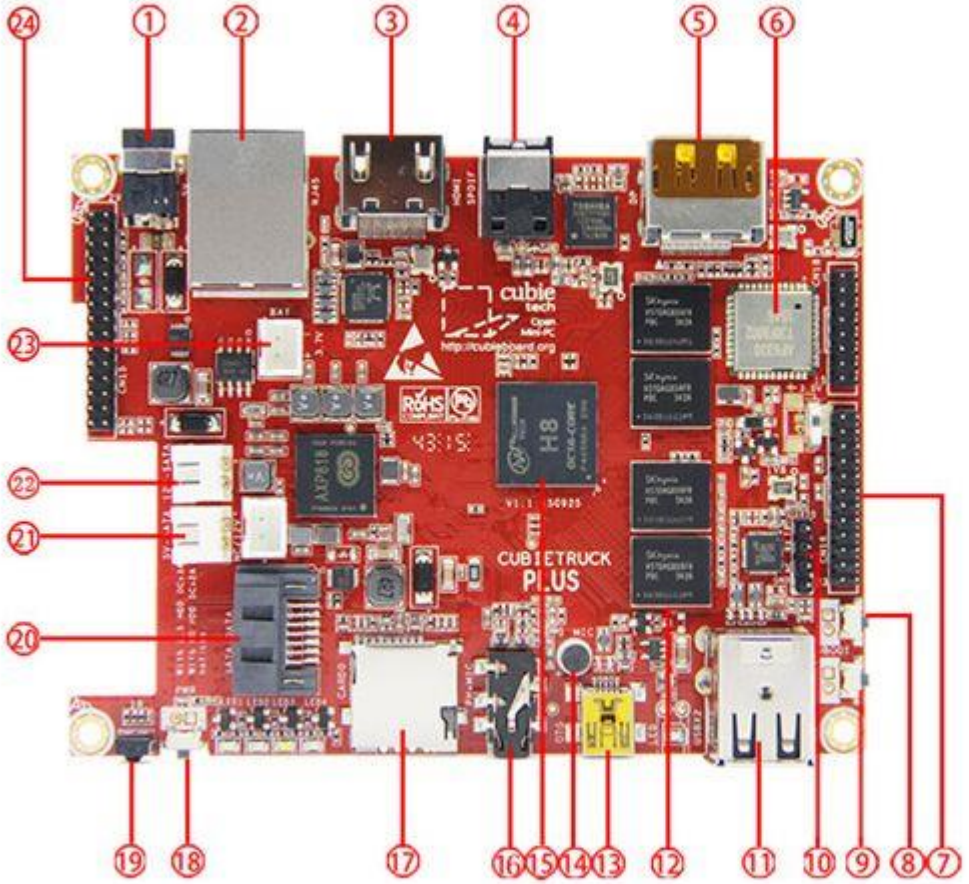
Other IoT Devices:

	Raspberry Pi	pcDuino	BeagleBone Black	Cubieboard
CPU	700 MHz ARM1176JZ-F Processor	1GHz ARM Cortex A8	AM335x 1GHz ARM Cortex-A8	Dual core 1GHz ARM Cortex-A7
GPU	Dual Core VideoCore IV Multimedia Co-Processor	Mali 400	PowerVR SGX530	Dual core ARM Mali 400 MP2
Memory	512MB	1GB	512MB	1GB
Storage	-	2GB Flash (ATmega328)	2GB on-board flash storage	4GB NAND Flash
Networking	10/100M Ethernet	10/100M Ethernet	10/100M Ethernet	10/100M Ethernet
Input/Output	2 USB, SD, MMC, SDIO card slot	-	4+1 USB, MicroSD slot	2 USB, MicroSD slot, SATA, IR sensor

Interfaces	GPIO, SPI, I2C, serial	Serial, ADC, PWM, GPIO, I2C, SPI	69 pin GPIO, SPI, I2C, 4 serial, CAN, GPMC, AIN, MMC, XDMA	96 extend pin interface, including I2C, SPI, RGB/LVDS, CSI/TS, FM-IN, ADC, CVBS, VGA, SPDIF-OUT, R-TP
OS	Rasbian, Pidora, RISC OS, Arch Linux	Ubuntu, Android	Angstrom Linux, Android, Ubuntu	Android, Official Linux distribution
Video	HDMI, Composite RCA (PAL and NTSC)	HDMI	HDMI	HDMI
Audio	3.5mm jack, HDMI	HDMI	HDMI	HDMI
Power	5VDC/700mA	5V/2A	5VDC/460mA	5VDC/2A







1. Power Jack
2. Ethernet Port
3. HDMI Port
4. SPDIF FIBER
5. Display Port
6. WIFI+BT
7. Extended Pin
8. UBOOT Key
9. RESET Key
10. UART Head Pin
11. USB 2.0*2
12. DRAM
13. Mini USB OTG
14. MIC
15. CPU
16. Earphone+MIC
17. TF-CARD
18. PWER ON/OFF
19. IR Sensor
20. SATA-DATA
21. 5V-SATA
22. 12V-SATA
23. Li-Battery
24. Extended Pin

The End

UNIT-5

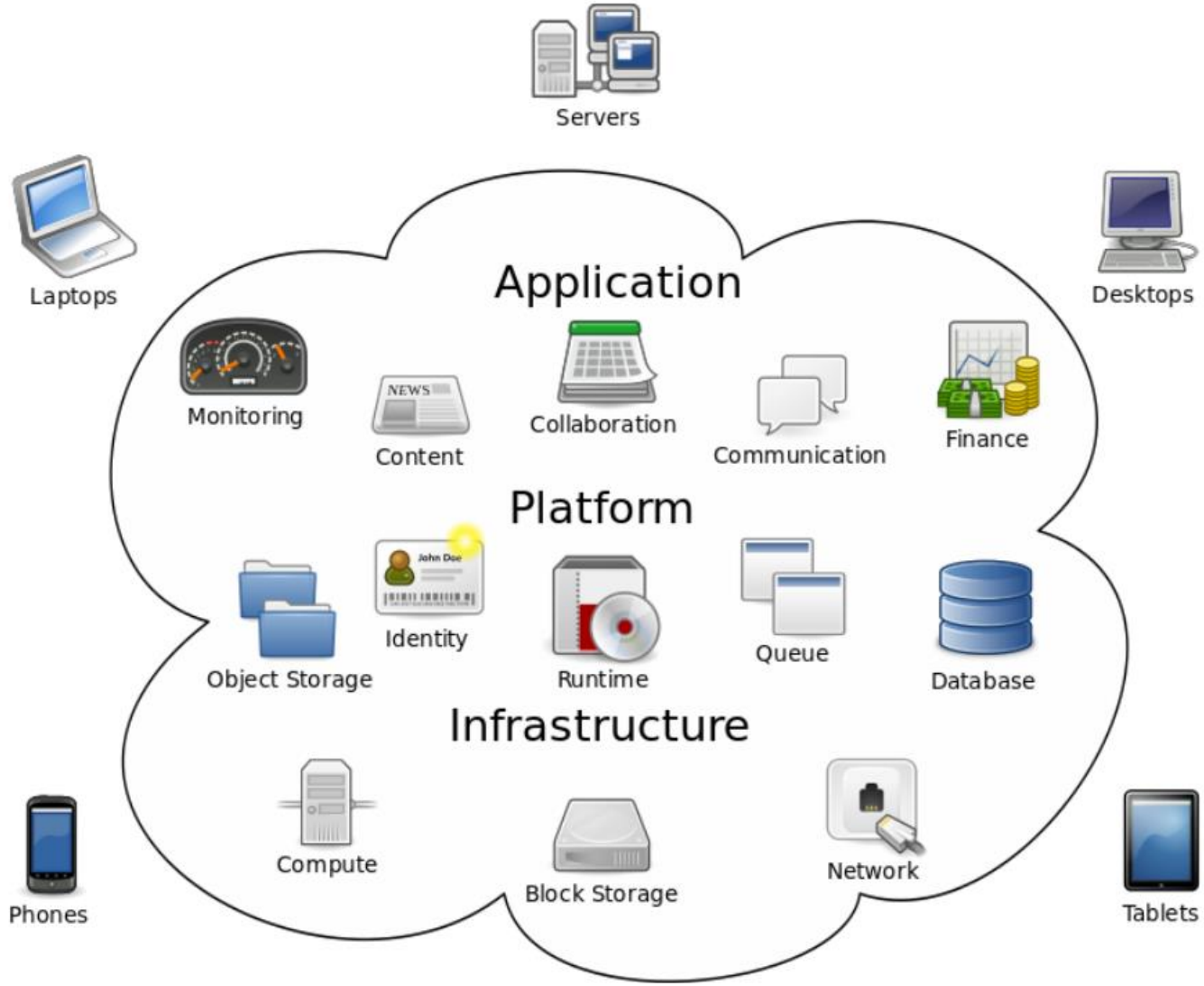
IOT PHYSICAL SERVERS &CLOUD OFFERINGS

Introduction to Cloud Storage Models & Communication APIs, WAMP-Auto Bahn for IoT, Xively Cloud for IoT, Python Web Application Framework, Django, Designing a REST ful Web API, Amazon Web services for IoT, SkyNet IoT Messaging Platform.

Introduction to Cloud Storage Models & Communication APIs

- The Internet of Things (IoT) involves the internet-connected devices we use to perform the processes and services that support our way of life.
- Another component set to help IoT succeed is cloud computing, which acts as a sort of front end.
- Cloud computing is an increasingly popular service that offers several advantages to IOT, and is based on the concept of allowing users to perform normal computing tasks using services delivered entirely over the internet.
- Another example: you have a problem with your mobile device and you need to reformat it or reinstall the operating system. You can use Google Photos to upload your photos to internet-based storage. After the reformat or reinstall, you can then either move the photos back to you device or you can view the photos on your device from the internet when you want.

- In truth, cloud computing and IoT are tightly coupled.
- Cloud computing as a paradigm for big data storage and analytics.
- While IoT is exciting on its own, the real innovation will come from combining it with cloud computing.
- The combination of cloud computing and IoT will enable new monitoring services and powerful processing of sensory data streams.
- For example, sensory data can be uploaded and stored with cloud computing, later to be used intelligently for smart monitoring and actuation with other smart devices.
- Ultimately, the goal is to be able to transform data to insight and drive productive, cost-effective action from those insights.
- The cloud effectively serves as the brain to improved decision-making and optimized internet-based interactions.
- Cloud computing offers a practical utility-based model that will enable businesses and users to access applications on demand anytime and from any where.



Cloud computing

Characteristics of Cloud Service

- First, the cloud computing of IoT is an on-demand self service, meaning it's there when you need it. Cloud computing is a web-based service that can be accessed without any special assistance or permission from other people; however, you need at minimum some sort of internet access.
- Second, the cloud computing of IoT involves broad network access, meaning it offers several connectivity options. Cloud computing resources can be accessed through a wide variety of internet-connected devices such as tablets, mobile devices and laptops. This level of convenience means users can access those resources in a wide variety of manners, even from older devices. Again, though, this emphasizes the need for network access points.

- Third, cloud computing allows for resource pooling, meaning information can be shared with those who know where and how (have permission) to access the resource, anytime and anywhere. This lends to broader collaboration or closer connections with other users. From an IoT perspective, just as we can easily assign an IP address to every "thing" on the planet, we can share the "address" of the cloud-based protected and stored information with others and pool resources
- Fourth, cloud computing features rapid elasticity, meaning users can readily scale the service to their needs. You can easily and quickly edit your software setup, add or remove users, increase storage space, etc. This characteristic will further empower IoT by providing elastic computing power, storage and networking.
- Finally, the cloud computing of IoT is a measured service, meaning you get what you pay for. Providers can easily measure usage statistics such as storage, processing, bandwidth and active user accounts inside your cloud instance.

- Deployment models Deployment in cloud computing comprises four deployment models: private cloud, public cloud, community cloud and hybrid cloud.
- A cloud storage API is an application program interface that connects a locally-based application to a cloud-based storage system, so that a user can send data to it and access and work with data stored in it.
- An application program interface (API) is code that allows two software programs to communicate with each other. The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application.

3-types of API

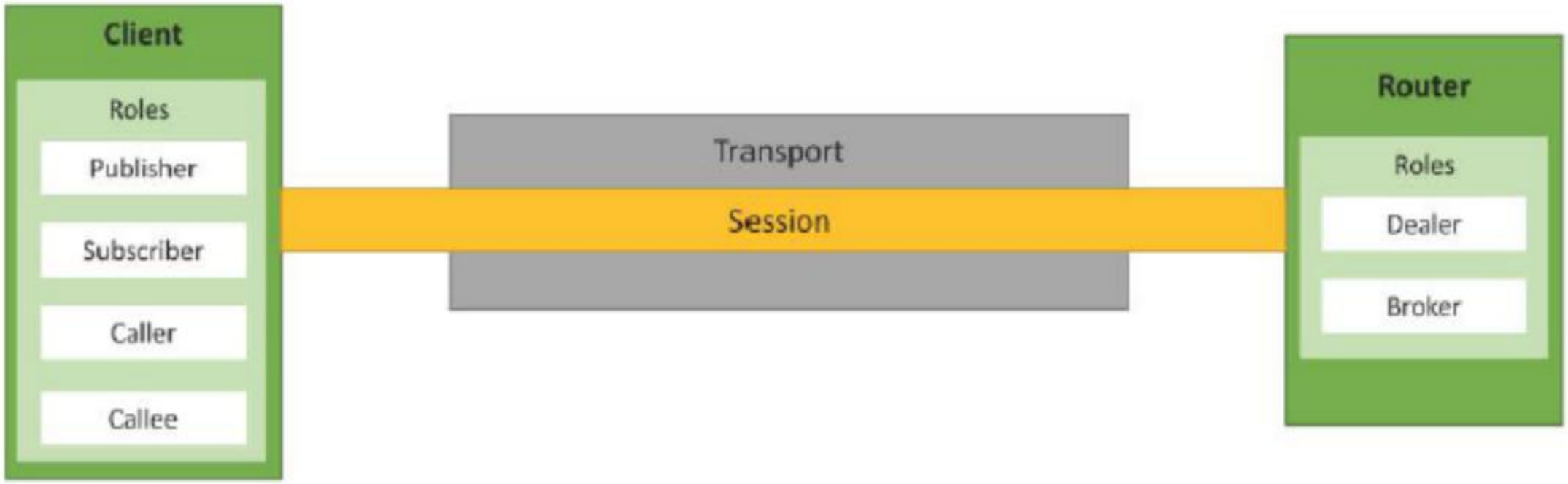
1. Local APIs are the original form, from which the name came. They offer OS or middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local API form.

2. Web APIs are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API. Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.

3. Program APIs are based on remote procedure call (RPC) technology that makes a remote program component appear to be local to the rest of the software. Service oriented architecture (SOA) APIs, such as Microsoft's WS-series of APIs, are program APIs.

WAMP – AutoBahn for IoT

- Web Application Messaging Protocol (WAMP) is a sub-protocol of Websocket which provides publish-subscribe and remote procedure call (RPC) messaging patterns.



- Transport: Transport is channel that connects two peers.
- Session: Session is a conversation between two peers that runs over a transport.
- Client: Clients are peers that can have one or more roles. In publish-subscribe model client can have following roles:
  – Publisher: Publisher publishes events (including payload) to the topic maintained by the Broker.
  – Subscriber: Subscriber subscribes to the topics and receives the events including the payload.
  In RPC model client can have following roles:
  – Caller: Caller issues calls to the remote procedures along with call arguments.
  – Callee: Callee executes the procedures to which the calls are issued by the caller and returns the results back to the caller.

Router: Routers are peers that perform generic call and event routing.

In publish-subscribe model Router has the role of a Broker:

– Broker: Broker acts as a router and routes messages published to a topic to all subscribers subscribed to thetopic. In RPC model Router has the role of a Broker:

– Dealer: Dealer acts a router and routes RPC calls from the Caller to the Callee and routes results from Callee to Caller.

- Application Code: Application code runs on the Clients (Publisher, Subscriber, Callee or Caller).

Xively Cloud for IoT

- Use of Cloud IoT cloud-based service
- The service provides for the data collection, data points, messages and calculation objects.
- The service also provisions for the generation and communication of alerts, triggers and feeds to the user.
- A user is an application or service. The user obtains responses or feeds from the cloud service.

Pachube platform: for data capture in real-time over the Internet

• Cosm: a changed domain name, where using a concept of console, one can monitor the feeds

• Xively is the latest domain name.

A commercial PaaS for the IoT/M2M

• A data aggregator and data mining website often integrated into the Web of Things

• An IoT PaaS for services and business services.

Xively PaaS services:

- Data visualisation for data of connected sensors to IoT devices.
- Graphical plots of collected data.
- Generates alerts.
- Access to historical data
- Generates feeds which can be real-world objects of own or others

Xively HTTP based APIs :

- Easy to implement on device hardware acting as clients to Xively web services
- APIs connect to the web service and send data.
- APIs provides services for logging, sharing and displaying sensor data of all

Xively Support:

- The platform supports the REST, WebSockets and MQTT protocols and connects the devices to Xively Cloud Services
- Native SDKs for Android, Arduino, ARM mbed, Java, PHP, Ruby, and Python languages
- Developers can use the workflow of prototyping, deployment and management through the tools provided at Xively

Xively APIs :

- Enable interface with Python, HTML5, HTML5 server, tornado
- Interface with WebSocket Server and WebSockets
- Interface with an RPC (Remote Procedure Call).

Xively PaaS services: (PaaS is a cloud computing model that provides a complete on-demand platform for developing, running and managing applications)

- Enables services
- Business services platform which connects the products, including collaboration products
- Rescue, Boldchat, join.me, and operations to Internet
- Data collection in real-time over Internet

Xively Methods for IoT Devices Data:

- Concept of users, feeds, data streams, data points and triggers
- Data feed typically a single location (e.g. a device or devices network),
- Data streams are of individual sensors associated with that location (for example, ambient lights, temperatures, power consumption).
- Pull or Push (Automatic or Manual Feed) Xively Data formats and Structures
- Number of data formats and structures enable the interaction, data collection and services
- Support exists for JSON , XML and CSV
- Structures: Tabular, spreadsheet, Excel, Data numbers and Text with a comma-separated values in file

Xively Uses in IoT/M2M :

- Private and Public Data Access
- Data streams, Data points and Triggers
- Creating and Managing Feeds
- Visualizing Data

Python Web Application Framework - Django

Django is an open source web application framework for developing web applications in Python.

- A web application framework in general is a collection of solutions, packages and best practices that allows development of web applications and dynamic websites.
- Django is based on the Model-Template-View architecture and provides a separation of the data model from the business rules and the user interface.
- Django provides a unified API to a database backend.
- Thus web applications built with Django can work with different databases without requiring any code changes.
- With this flexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem, Django is best suited for cloud applications.
- Django consists of an object-relational mapper, a web templating system and a regular-expression based URL dispatcher.
- Django is Model-Template-View (MTV) framework.

Model:

The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc. A Django model is a Python class that outlines the variables and methods for a particular type of data.

Template :

In a typical Django web application, the template is simply an HTML page with a few extra placeholders. Django's template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV, etc.)

View :

The view ties the model to the template. The view is where you write the code that actually generates the web pages. View determines what data is to be displayed, retrieves the data from the database and passes the data to the template.

Designing a RESTful Web API,

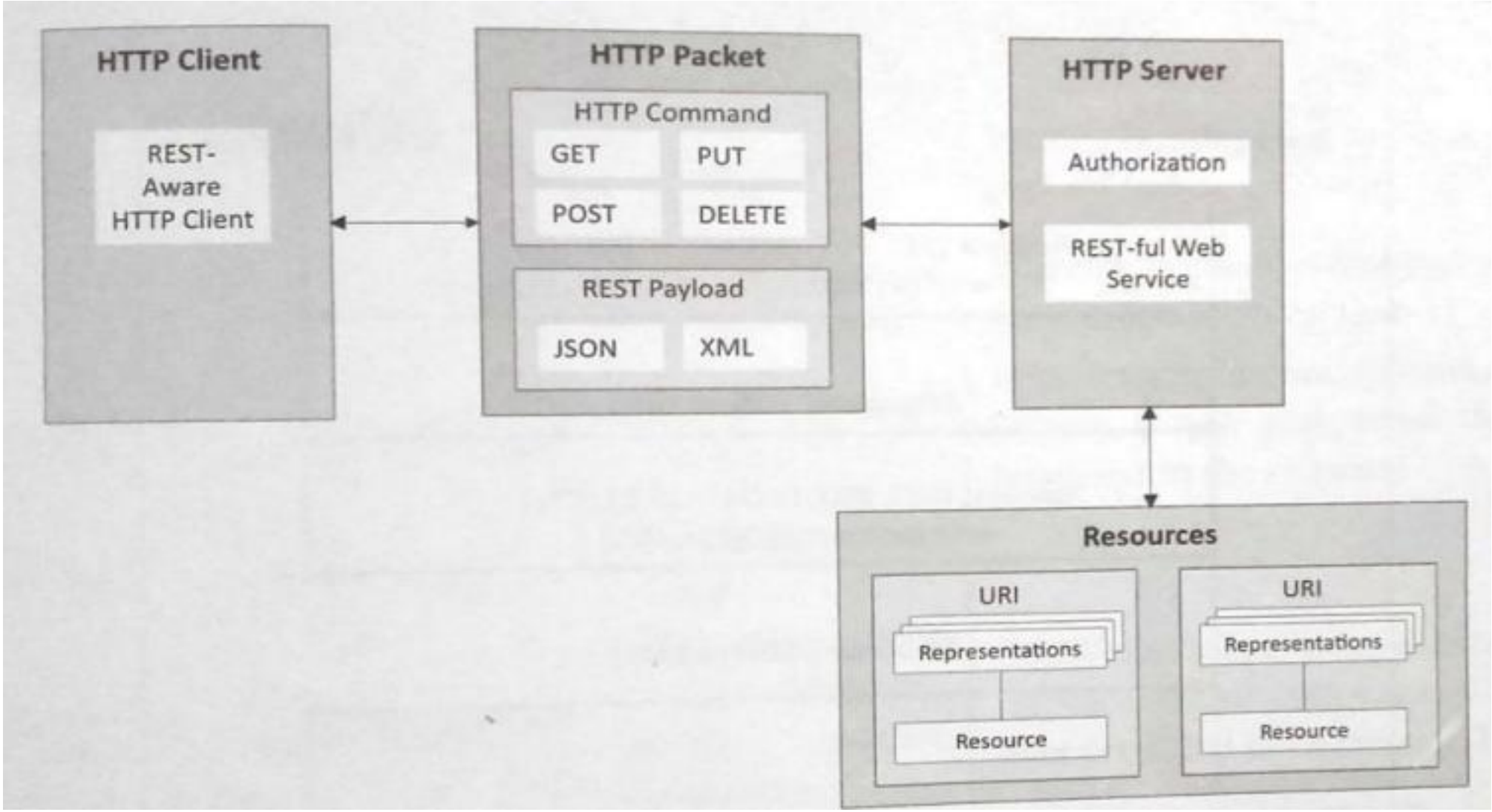
i) REST based communication APIs(Request-Response Based Model)

ii) WebSocket based Communication APIs(Exclusive Pair Based Model)

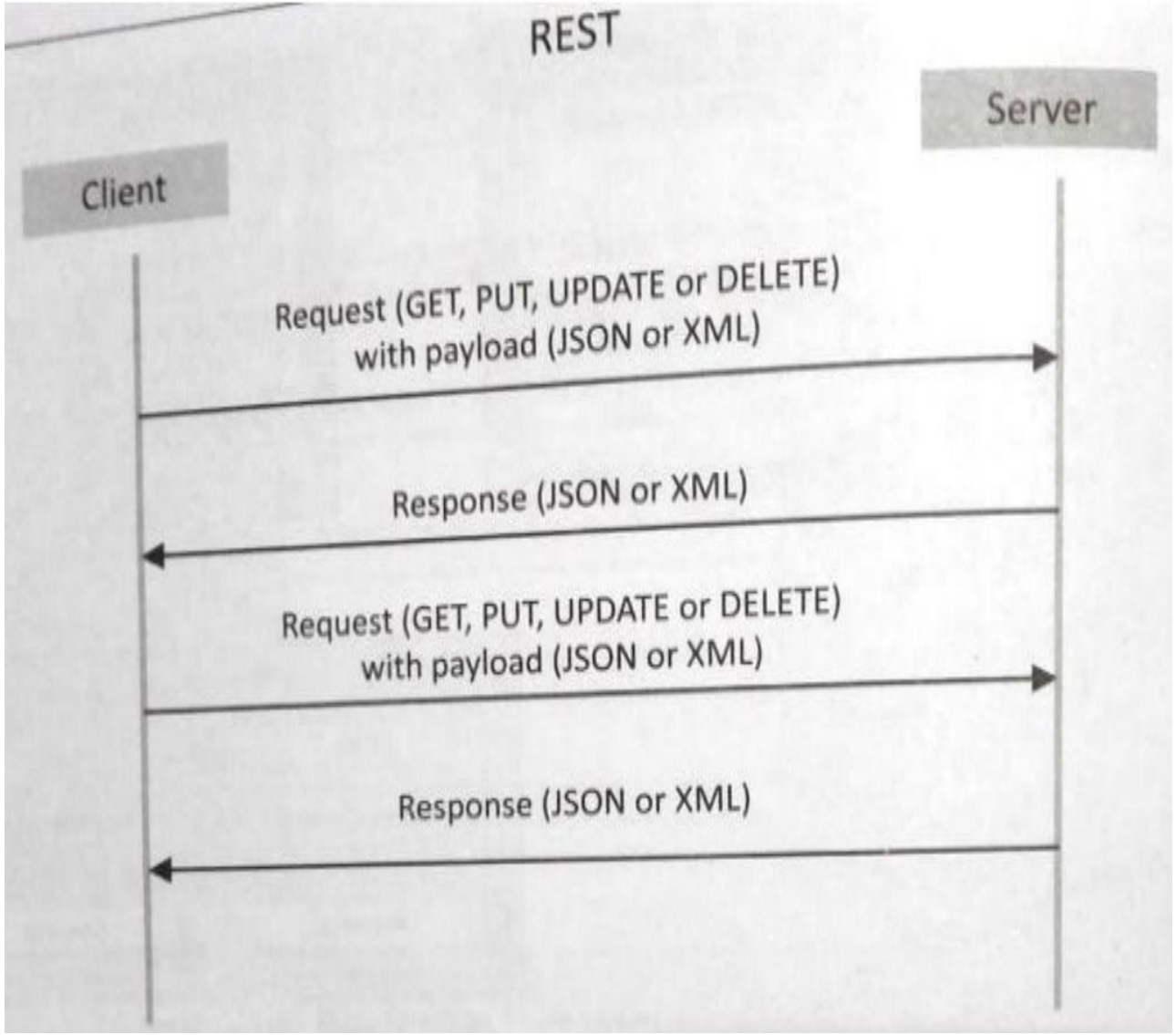
i) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system‘s resources and have resource states are addressed and transferred.

- Client-Server: The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.

- The REST architectural constraints are as follows: The below figure shows the communication between client server with REST APIs.



- Stateless: Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage of any stored context on the server.
- Cache-able: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- Layered System: constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.
- User Interface: constraint requires that the method of communication between a client and a server must be uniform.
- Code on Demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

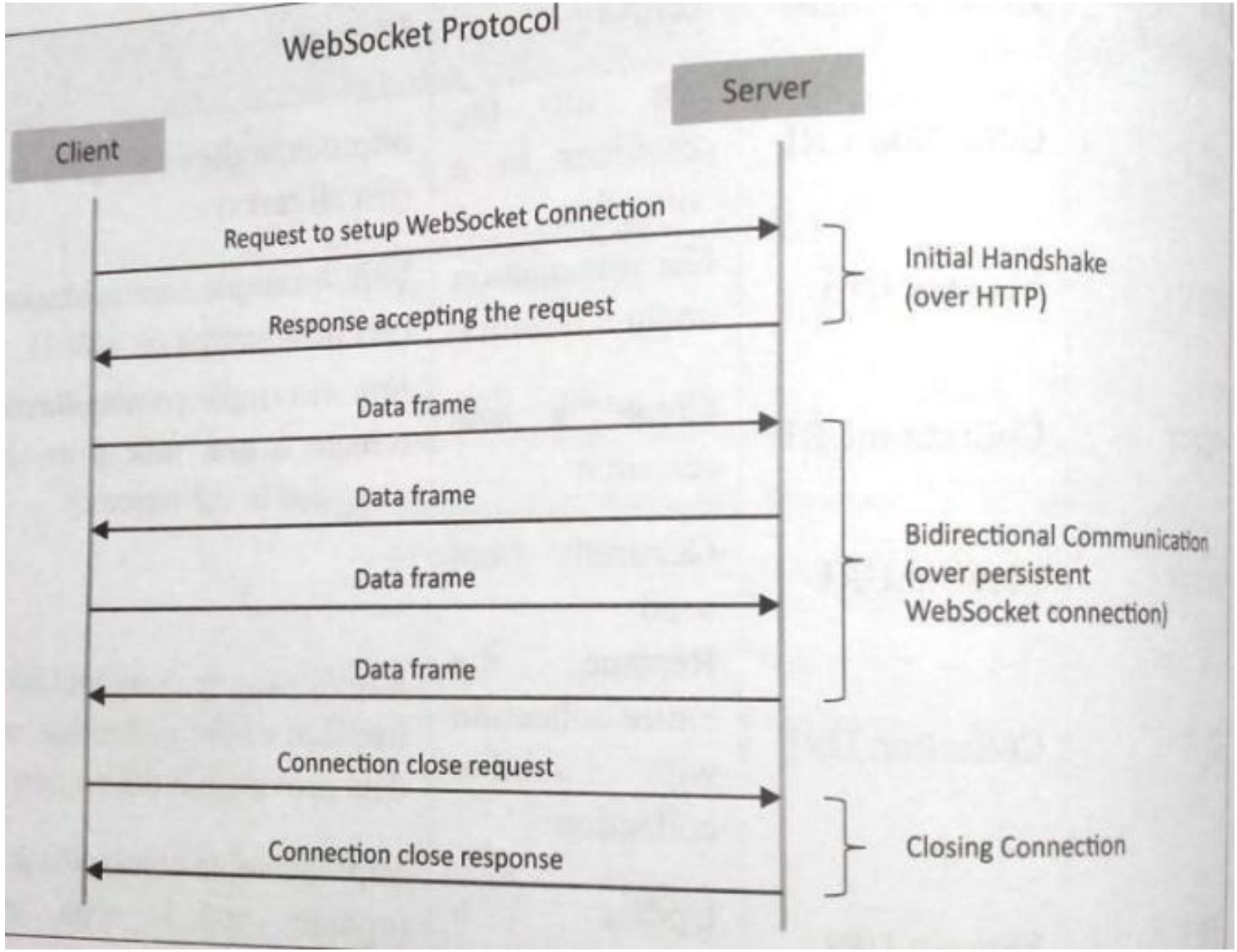


- The Request-Response model used by REST:
- RESTful web service is a collection of resources which are represented by URIs.
- RESTful web API has a base URI(e.g: http://example.com/api/tasks/).

- The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE).
- A RESTful web service can support various internet media types.

ii) WebSocket Based Communication APIs

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.



Amazon Web services for IoT

i) Amazon EC2 (Elastic Compute Cloud):

In this example, a connection to EC2 service is first established by calling boto.ec2.connect_to_region.

- The EC2 region, AWS access key and AWS secret key are passed to this function. After connecting to EC2 , a new instance is launched using the conn.run_instances function.
- The AMI-ID, instance type, EC2 key handle and security group are passed to this function.

ii) Amazon AutoScaling:

- A connection to AutoScaling service is first established by calling boto.ec2.autoscale.connect_to_region function.
- Launch Configuration :After connecting to AutoScaling service, a new launch configuration is created by calling conn.create_launch_configuration. Launch configuration contains instructions on how to launch new instances including the AMI-ID, instance type, security groups, etc.
- AutoScaling Group:After creating a launch configuration, it is then associated with a new AutoScaling group. AutoScaling group is created by calling conn.create_auto_scaling_group. The settings for AutoScaling group such as the maximum and minimum number of instances in the group, the launch configuration, availability zones, optional load balancer to use with the group, etc.

- AutoScaling Policies :

➢After creating an AutoScaling group, the policies for scaling up and scaling down are defined.

➢In this example, a scale up policy with adjustment type Change In Capacity and scaling_ad justment = 1 is defined.

➢Similarly a scale down policy with adjustment type ChangeInCapacity and scaling_ad justment = -1 is defined.

- CloudWatch Alarms:

➢With the scaling policies defined, the next step is to create Amazon CloudWatch alarms that trigger these policies.

➢ The scale up alarm is defined using the CPUUtilization metric with the Average statistic and threshold greater 70% for a period of 60 sec. The scale up policy created previously is associated with this alarm. This alarm is triggered when the average CPU utilization of the instances in the group becomes greater than 70% for more than 60 seconds.

➢The scale down alarm is defined in a similar manner with a threshold less than 50%.

iii)Amazon S3 (Simple Storage Service):

- In this example, a connection to S3 service is first established by calling boto.connect_s3 function.
- The upload_to_s3_bucket_path function uploads the file to the S3 bucket specified at the specified path.

iv)Amazon RDS (Relational Database Services):

In this example, a connection to RDS service is first established by calling boto.rds.connect_to_region function.

- The RDS region, AWS access key and AWS secret key are passed to this function.
- After connecting to RDS service, the conn.create_dbinstance function is called to launch a new RDS instance.
- The input parameters to this function include the instance ID, database size, instance type, database username, database password, database port, database engine (e.g. MySQL5.1), database name, security groups, etc.

v) Amazon DynamoDB:

In this example, a connection to DynamoDB service is first established by calling boto.dynamodb.connect_to_region.

- After connecting to DynamoDB service, a schema for the new table is created by calling conn.create_schema.
- The schema includes the hash key and range key names and types.
- A DynamoDB table is then created by calling conn.create_table function with the table schema, read units and write units as input parameters.

SkyNet IoT Messaging Platform

- SkyNet is running on a dozen Amazon EC2 servers and has nearly 50,000 registered smart devices including: Arduinos, Sparks, Raspberry Pis, Intel Galileos, and BeagleBoards, Matthieu said.
- SkyNet runs as an IoT platform-as-a-service (PaaS) as well as a private cloud through Docker, the new lightweight container technology.
- The platform is written in Node.js and released under an MIT open source license on GitHub.
- The single SkyNet API supports the following IoT protocols: HTTP, REST, WebSockets, MQTT (Message Queue Telemetry Transport), and CoAP (Constrained Application Protocol) for guaranteed message delivery and low-bandwidth satellite communications, Matthieu said.
- Every connected device is assigned a 36 character UUID and secret token that act as the device's strong credentials.
- Security permissions can be assigned to allow device discoverability, configuration, and messaging.

The End