



SINCE 1999

SYNERGY

A Modern Day Gurukul

SYNERGY INSTITUTE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
Academic Session 2023-24
LECTURE NOTE

Name of Faculty	: Mr. SMRUTI RANJAN DASH
Name of Subject	: INTERNET OF THINGS
Subject Code	: RIT7D001
Subject Credit	: 3
Semester	: VII
Year	: 4th
Course	: B.TECH
Branch	: COMPUTER SCIENCE & ENGINEERING
Admission Batch	: 2020-24

Chapter 1

Introduction to IoT

Outline

- IoT definition
- Characteristics of IoT
- Physical Design of IoT
- Logical Design of IoT
- IoT Protocols
- IoT Levels & Deployment Templates

Definition of IoT

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 the information network, often communicate data associated with users and their environments.

Characteristics of IoT

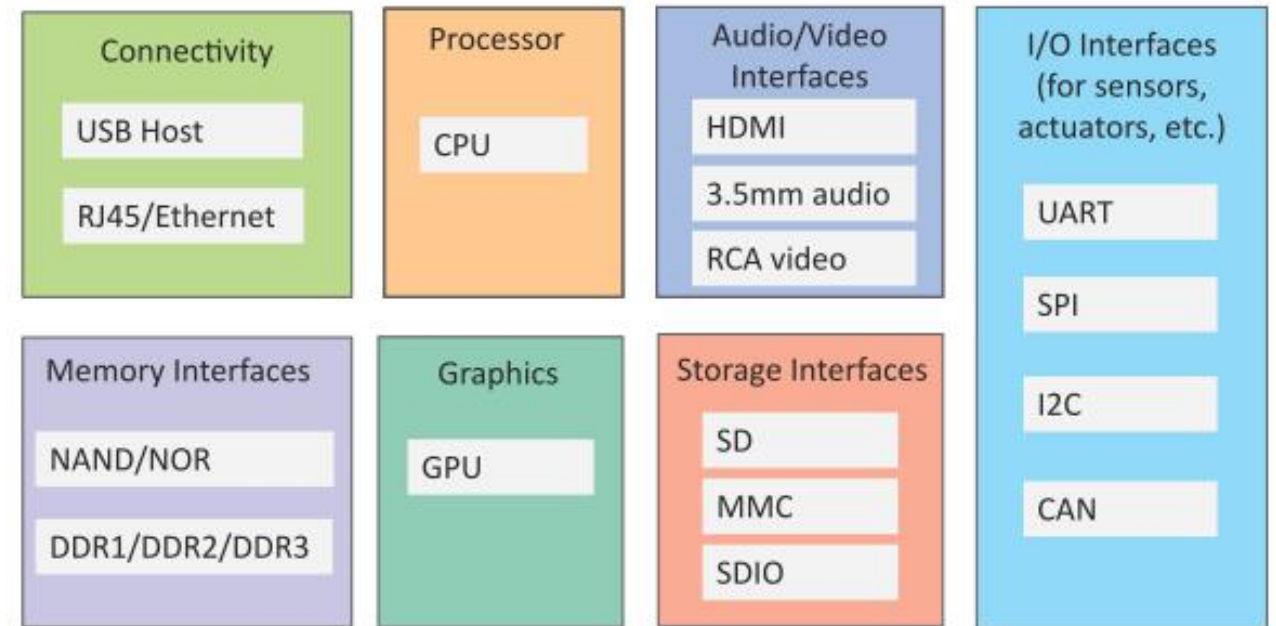
- Dynamic & Self-Adapting
- Self-Configuring
- Interoperable Communication Protocols
- Unique Identity
- Integrated into Information Network

Physical Design of IoT

- The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.
- IoT devices can:
 - Exchange data with other connected devices and applications (directly or indirectly), or
 - Collect data from other devices and process the data locally or
 - Send the data to centralized servers or cloud-based application back-ends for processing the data, or
 - Perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraints

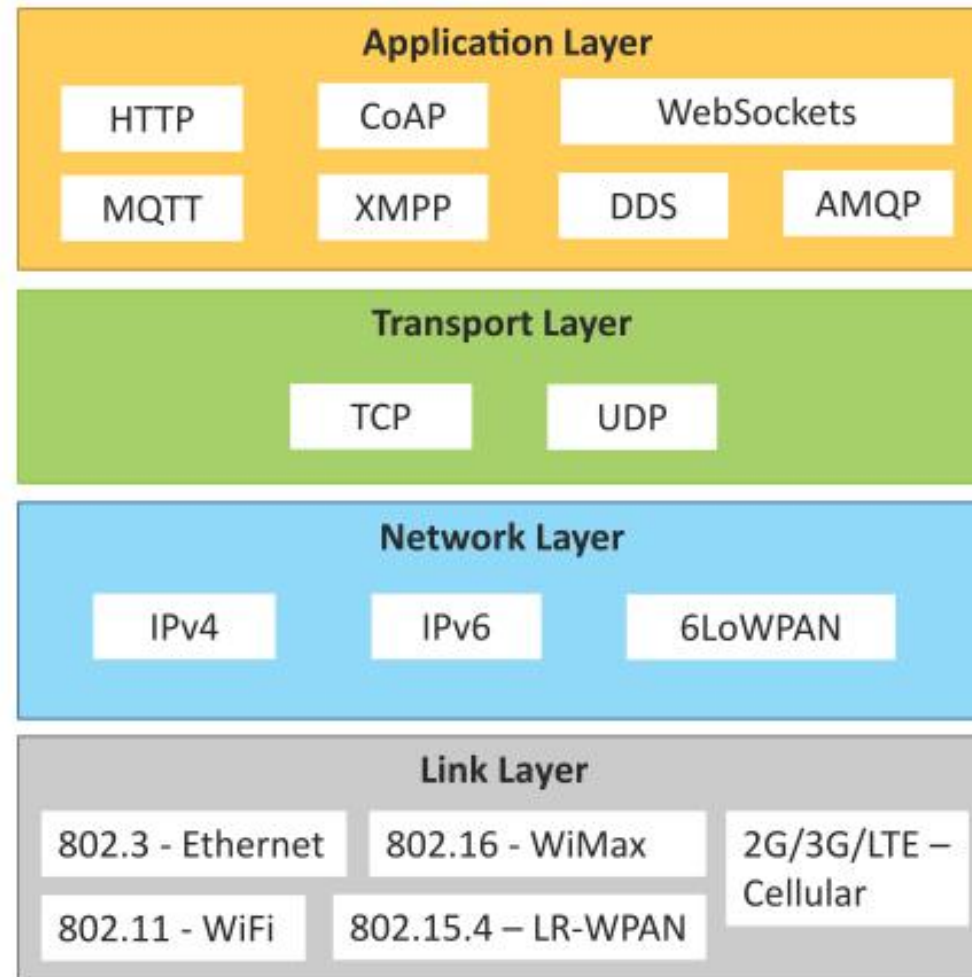
Generic block diagram of an IoT Device

- An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.
 - I/O interfaces for sensors
 - Interfaces for Internet connectivity
 - Memory and storage interfaces
 - Audio/video interfaces.



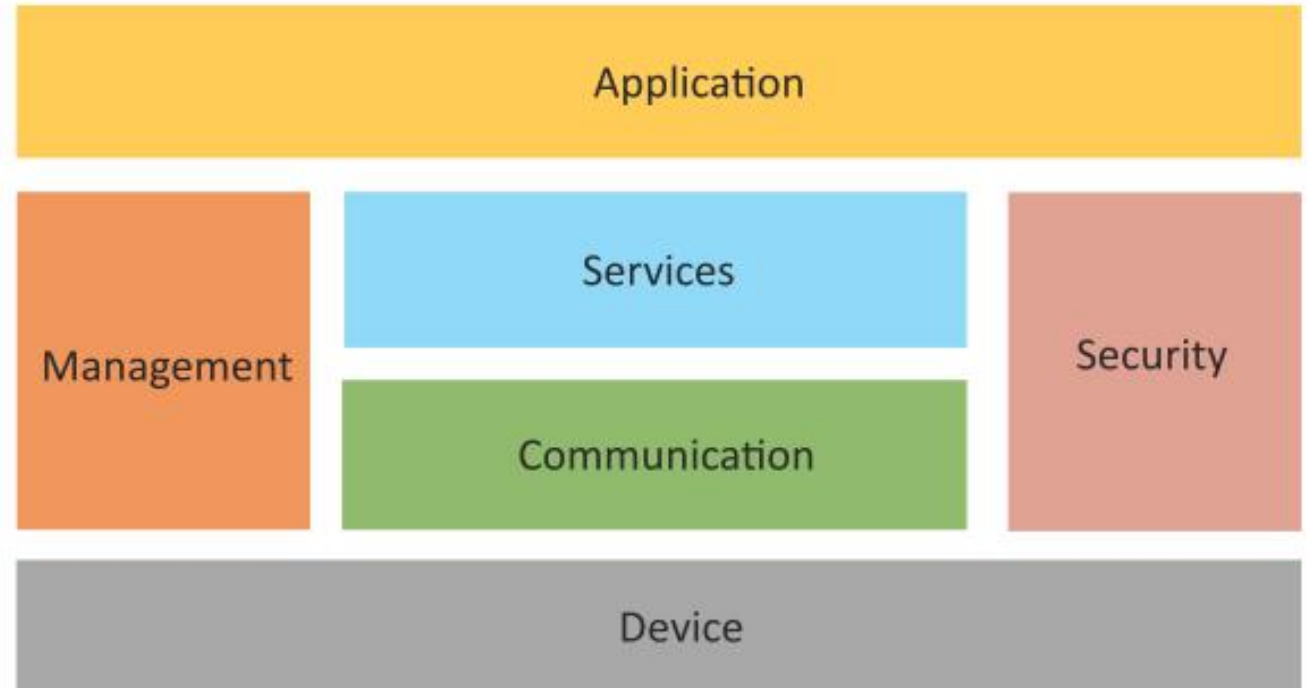
IoT Protocols

- Link Layer
 - 802.3 – Ethernet
 - 802.11 – WiFi
 - 802.16 – WiMax
 - 802.15.4 – LR-WPAN
 - 2G/3G/4G
- Network/Internet Layer
 - IPv4
 - IPv6
 - 6LoWPAN
- Transport Layer
 - TCP
 - UDP
- Application Layer
 - HTTP
 - CoAP
 - WebSocket
 - MQTT
 - XMPP
 - DDS
 - AMQP



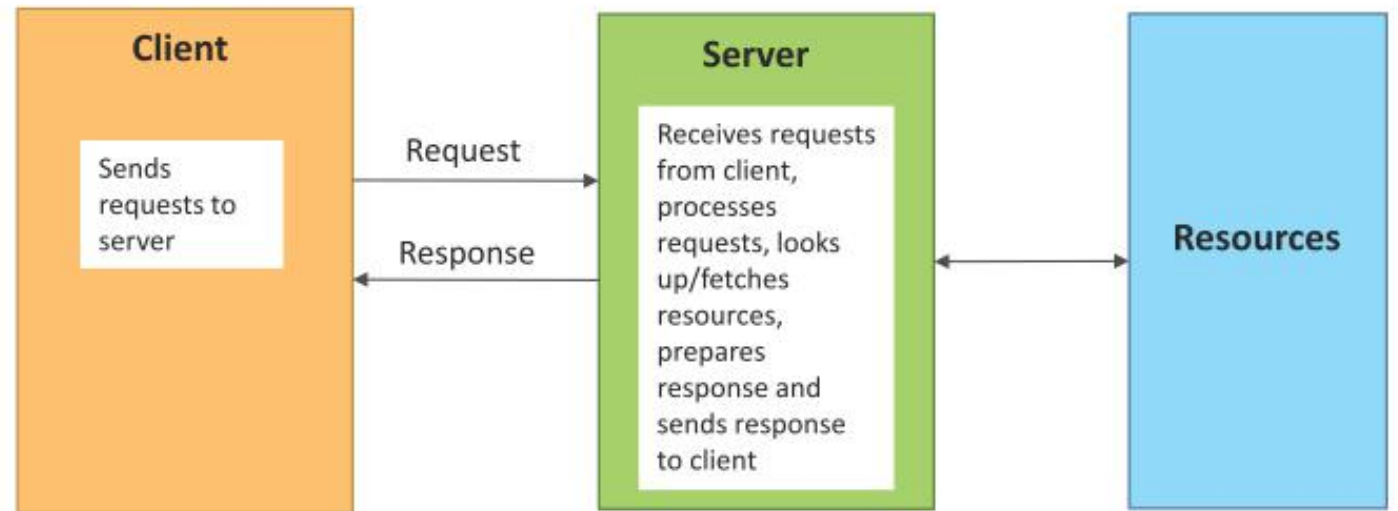
Logical Design of IoT

- Logical design of an IoT system refers to an abstract representation of the entities and processes without going into the low-level specifics of the implementation.
- An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication, and management.



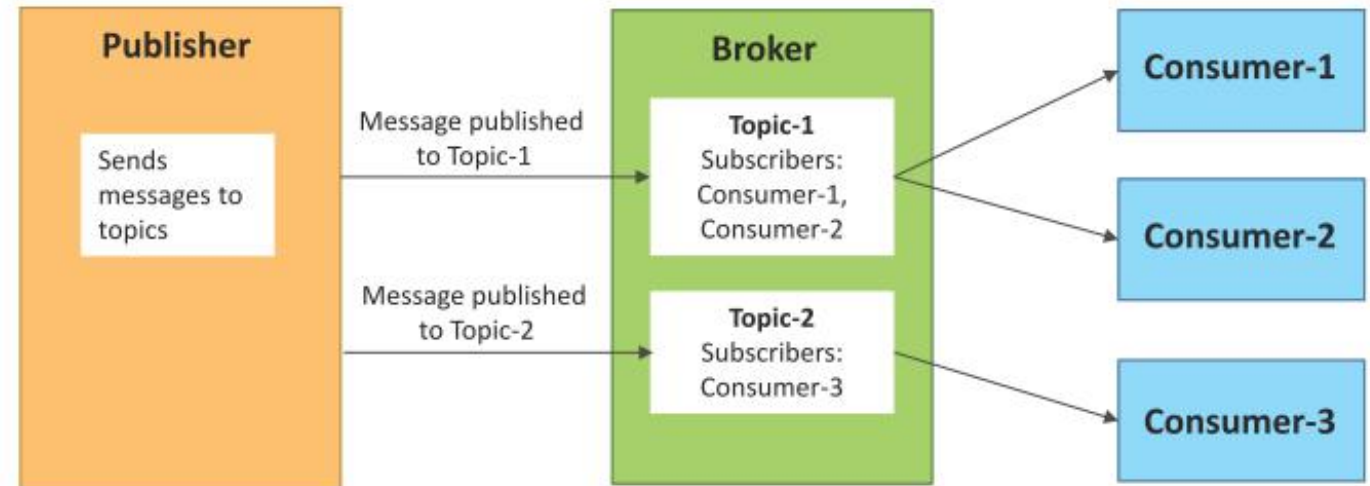
Request-Response communication model

- Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests.
- When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response, and then sends the response to the client.



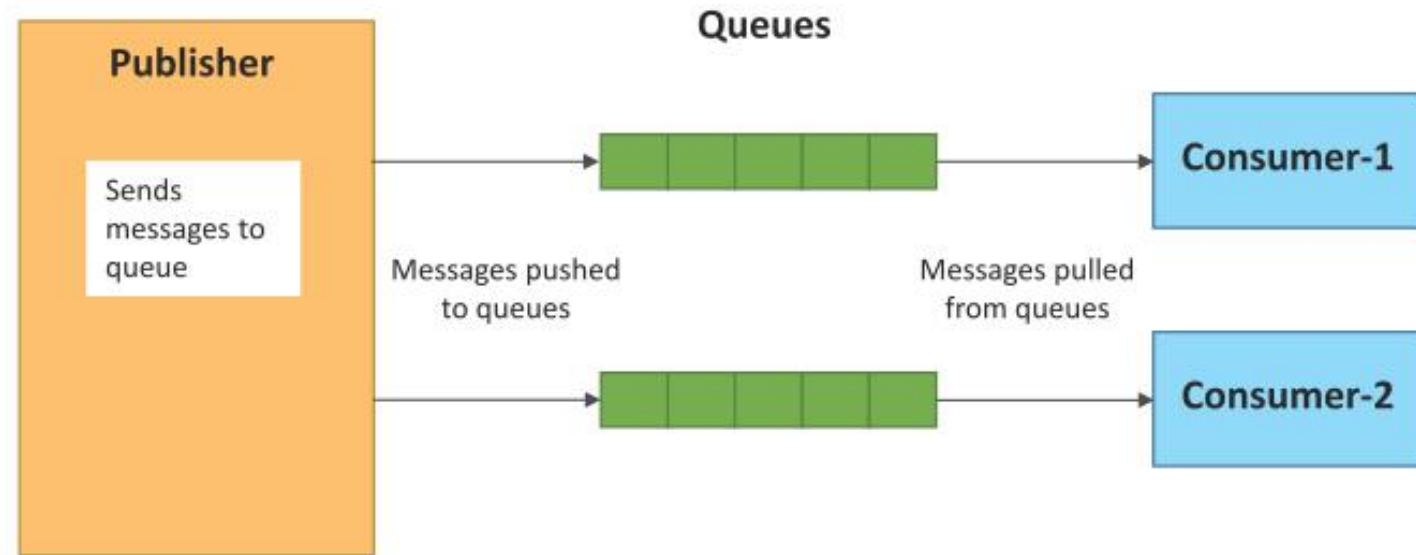
Publish-Subscribe communication model

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



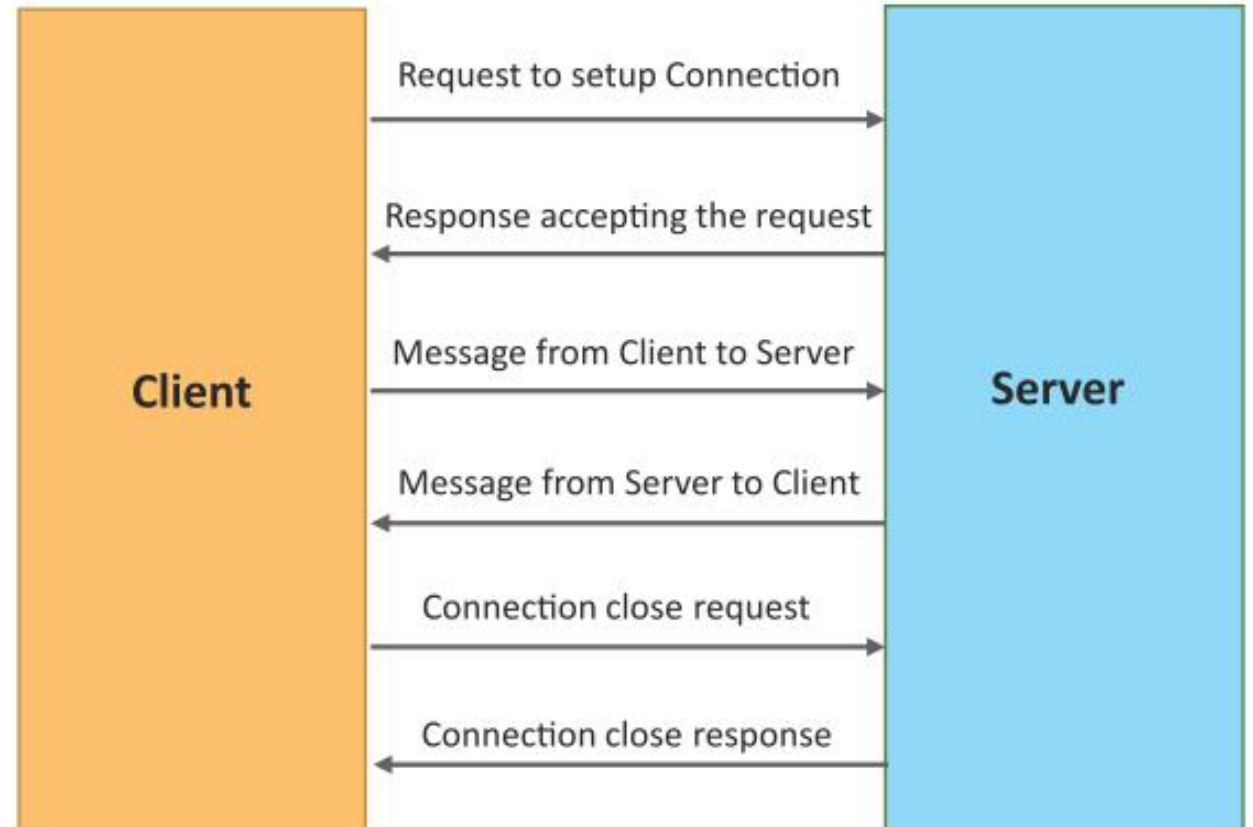
Push-Pull communication model

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



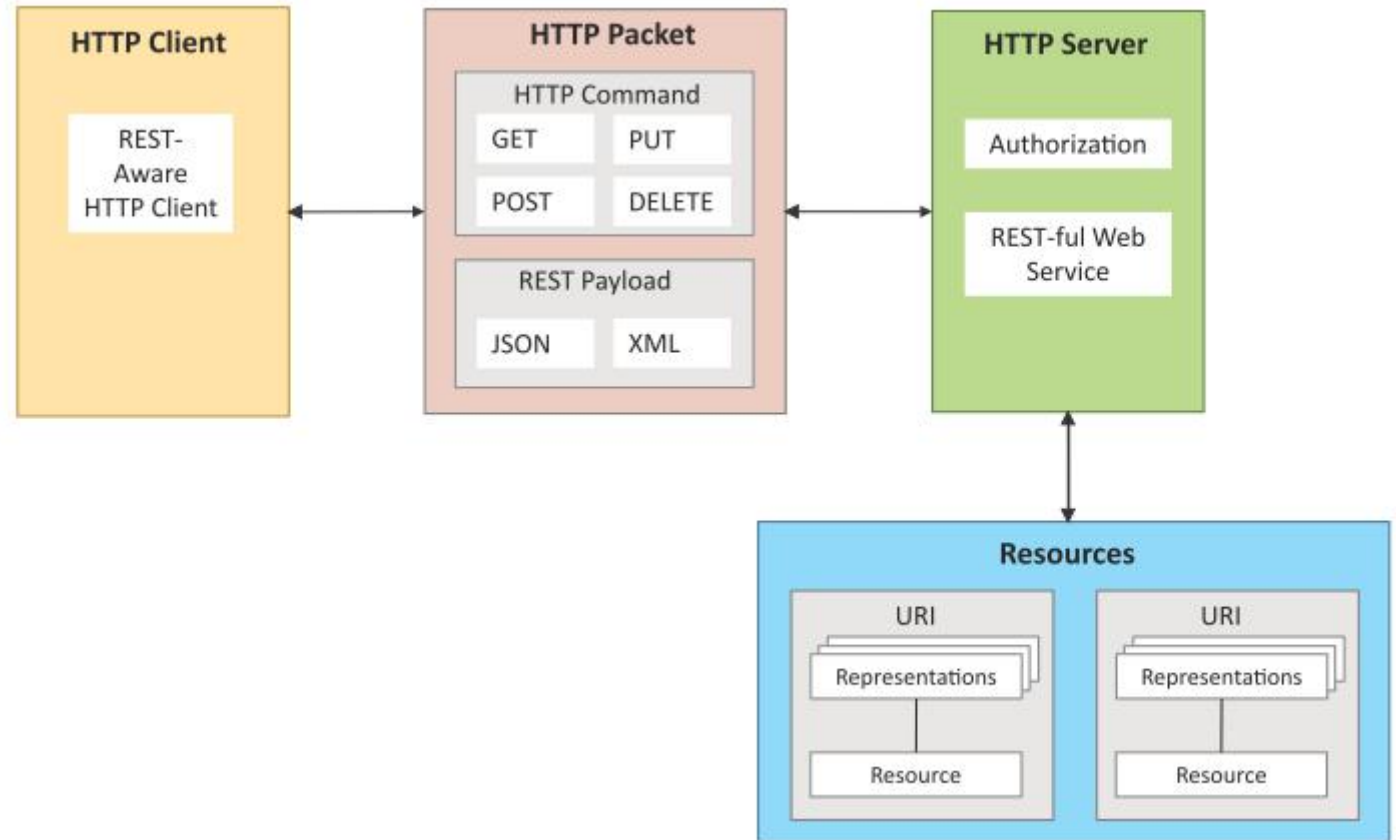
Exclusive Pair communication model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.
- Once the connection is setup it remains open until the client sends a request to close the connection.
- Client and server can send messages to each other after connection setup.



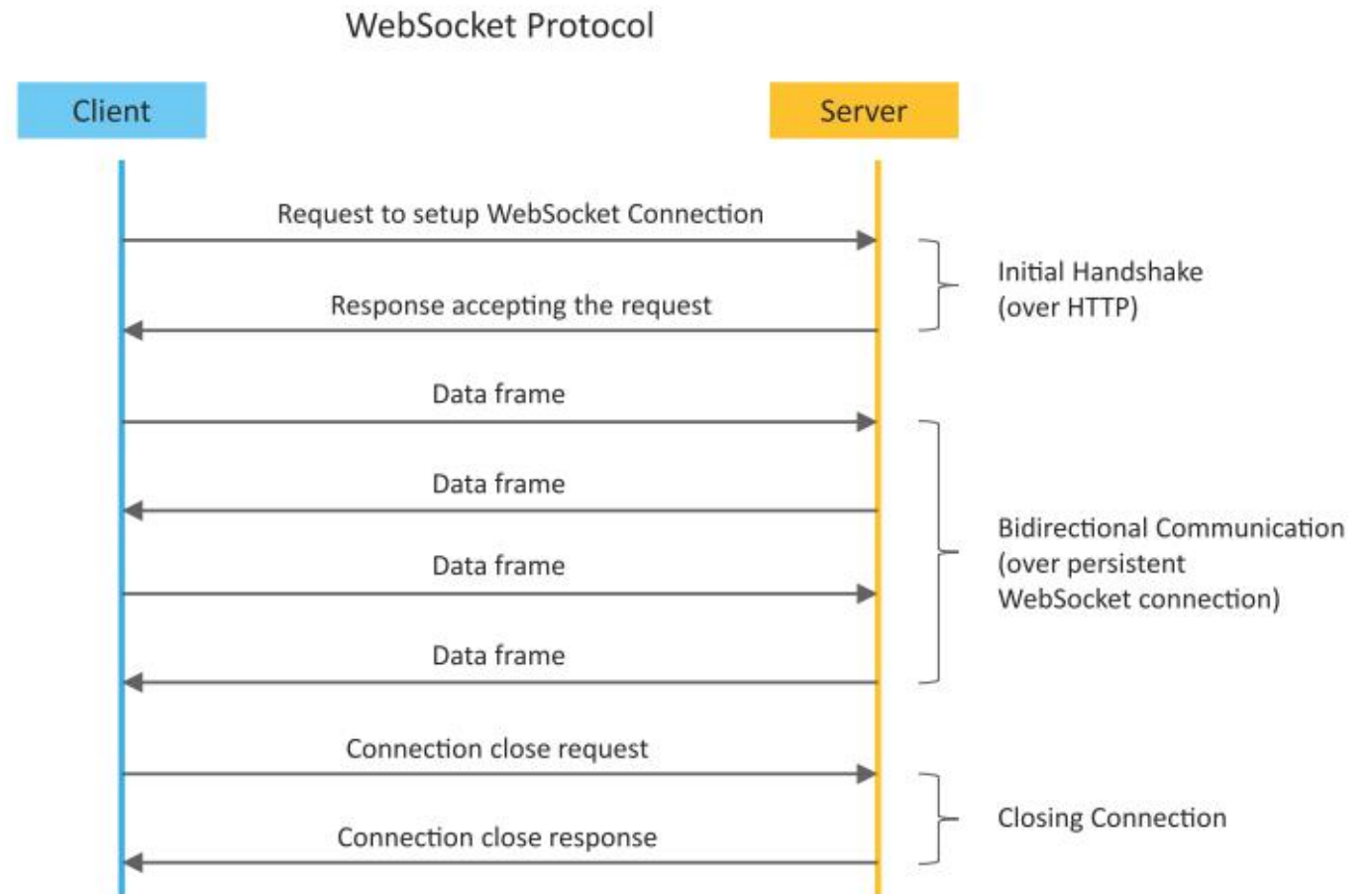
REST-based Communication APIs

- Representational State Transfer (REST) is a set of architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred.
- REST APIs follow the request-response communication model.
- The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia system.



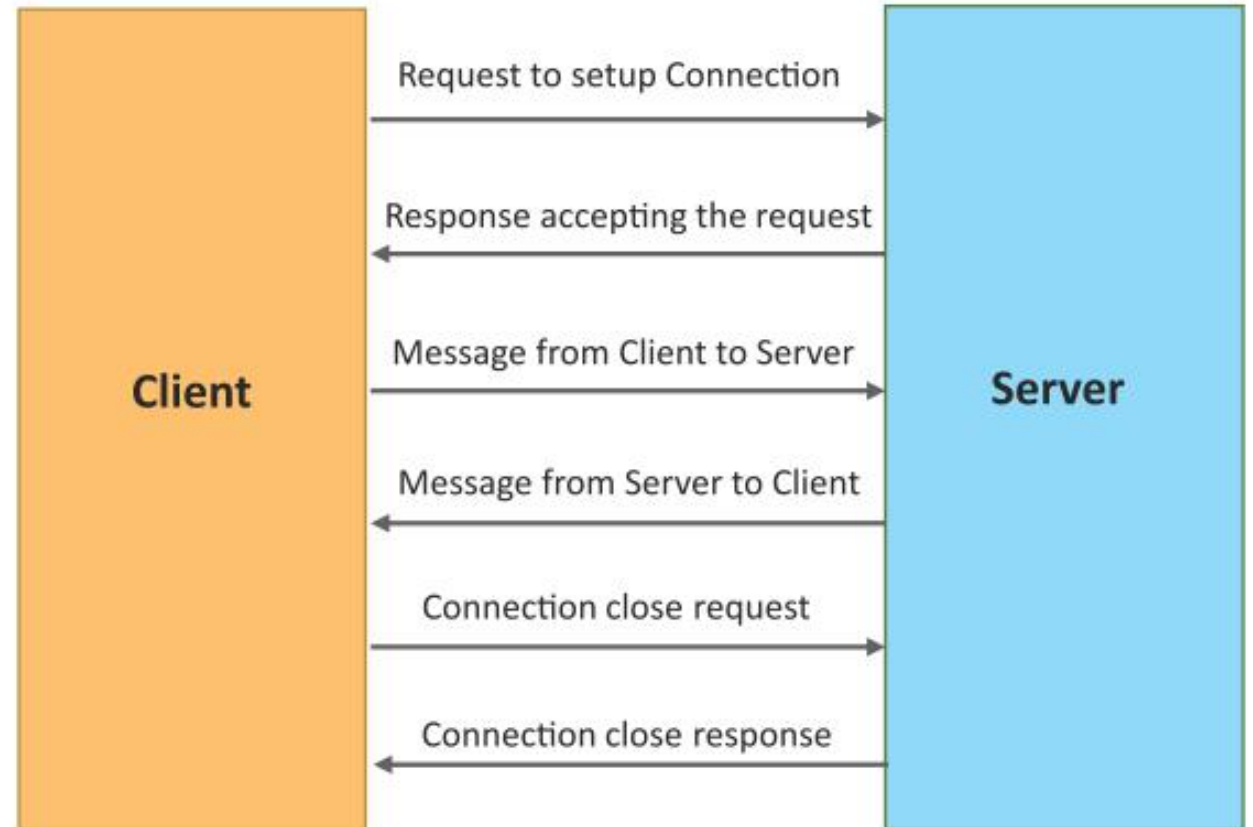
WebSocket-based Communication APIs

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



Exclusive Pair communication model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.
- Once the connection is setup it remains open until the client sends a request to close the connection.
- Client and server can send messages to each other after connection setup.



IoT Levels & Deployment Templates

An IoT system comprises of the following components:

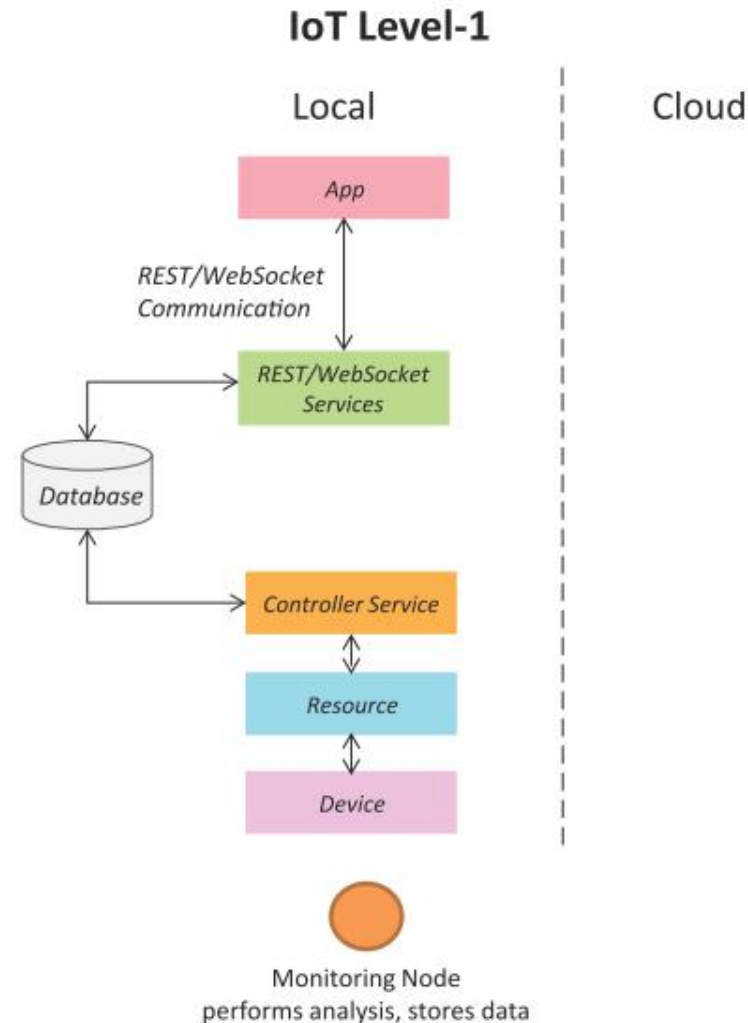
- **Device:** An IoT device allows identification, remote sensing, actuating and remote monitoring capabilities. You learned about various examples of IoT devices in section
- **Resource:** Resources are software components on the IoT device for accessing, processing, and storing sensor information, or controlling actuators connected to the device. Resources also include the software components that enable network access for the device.
- **Controller Service:** Controller service is a native service that runs on the device and interacts with the web services. Controller service sends data from the device to the web service and receives commands from the application (via web services) for controlling the device.

IoT Levels & Deployment Templates

- **Database:** Database can be either local or in the cloud and stores the data generated by the IoT device.
- **Web Service:** Web services serve as a link between the IoT device, application, database and analysis components. Web service can be either implemented using HTTP and REST principles (REST service) or using WebSocket protocol (WebSocket service).
- **Analysis Component:** The Analysis Component is responsible for analyzing the IoT data and generate results in a form which are easy for the user to understand.
- **Application:** IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and view the processed data.

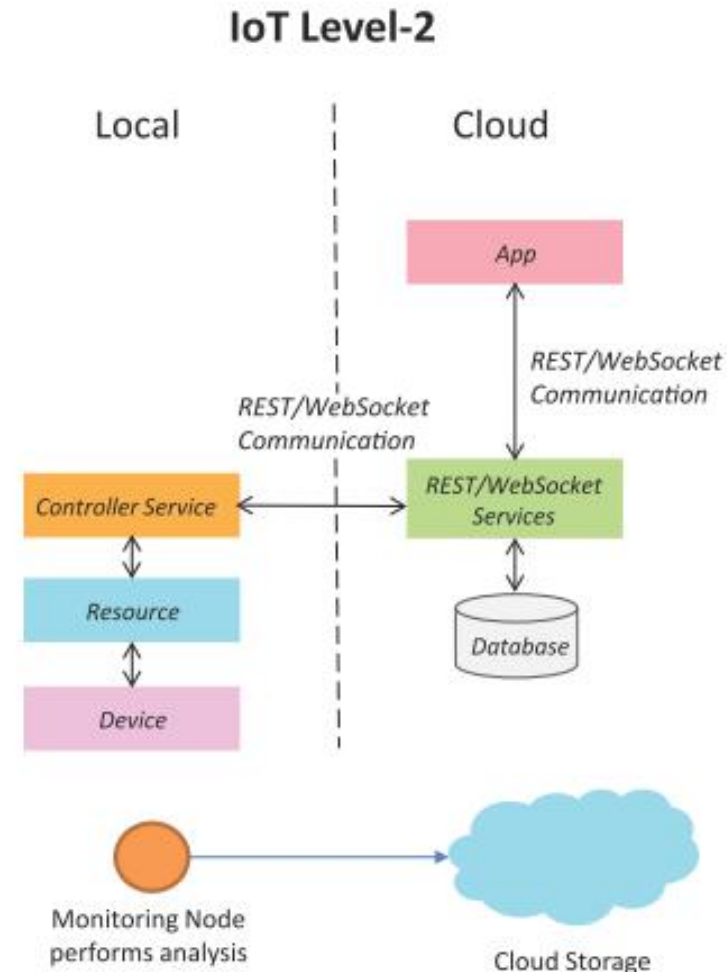
IoT Level-1

- A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application
- Level-1 IoT systems are suitable for modeling low-cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.



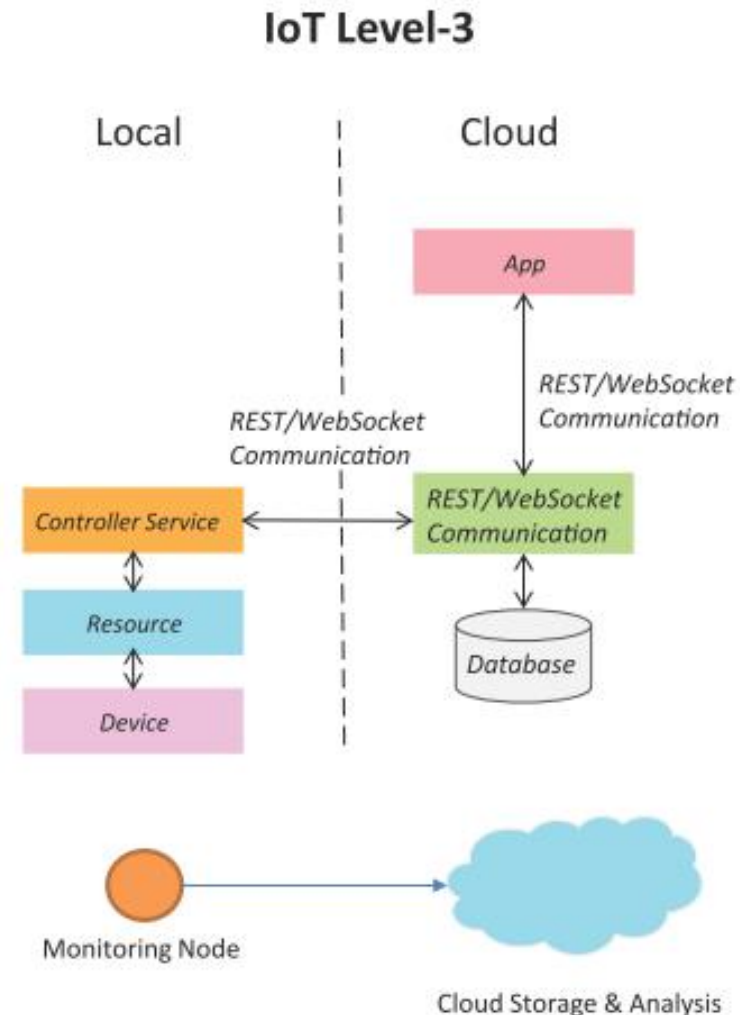
IoT Level-2

- A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis.
- Data is stored in the cloud and application is usually cloud-based.
- Level-2 IoT systems are suitable for solutions where the data involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself.



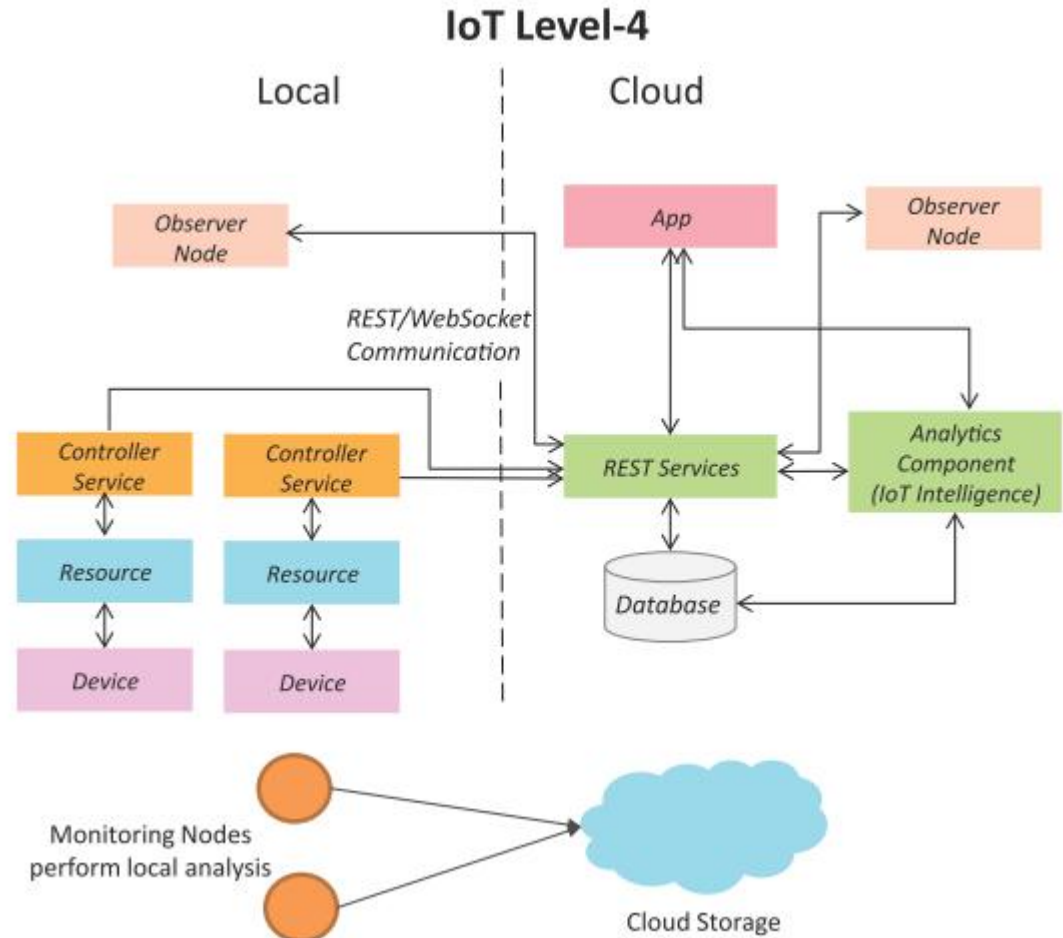
IoT Level-3

- A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and application is cloud-based.
- Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.



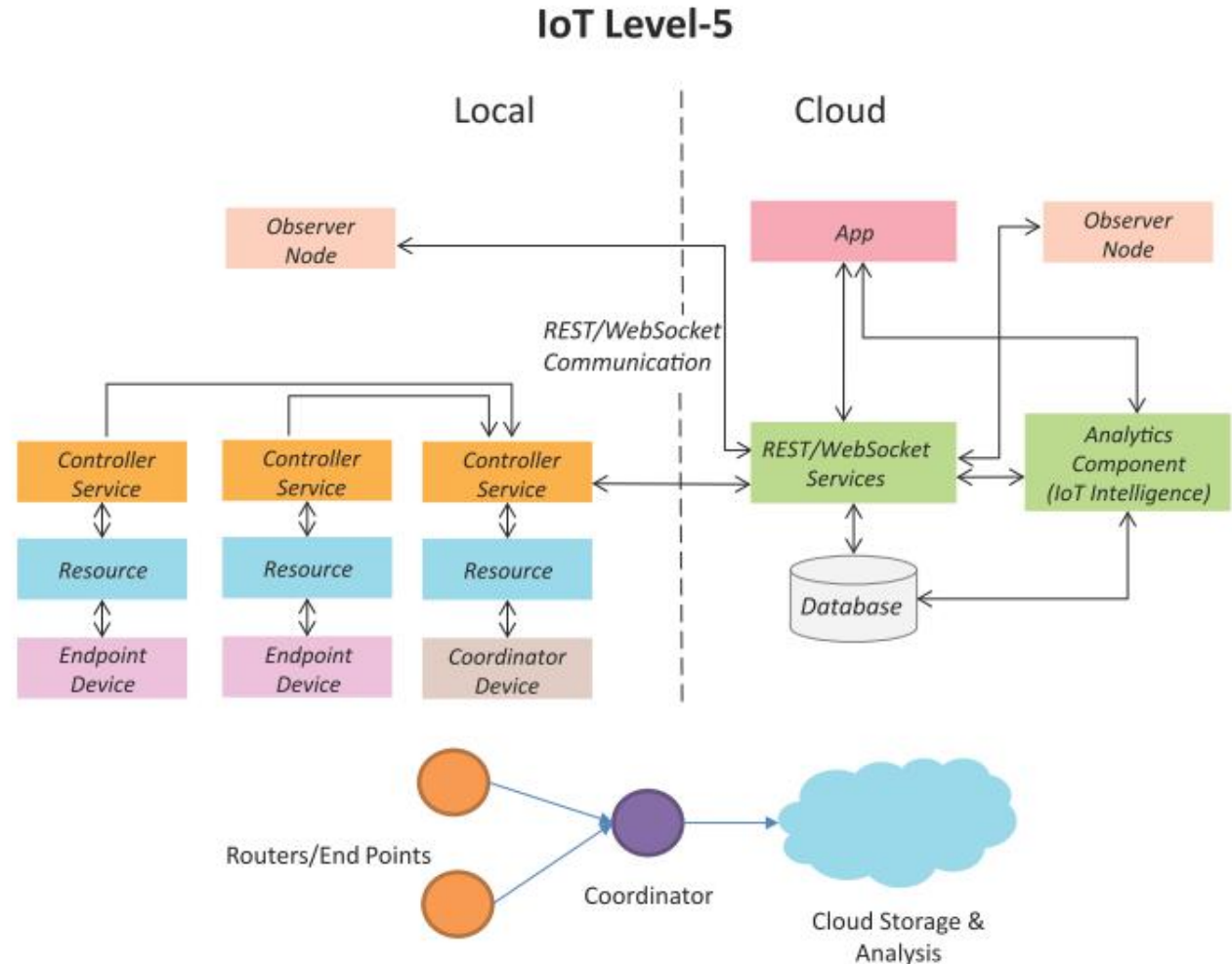
IoT Level-4

- A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based.
- Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.
- Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.



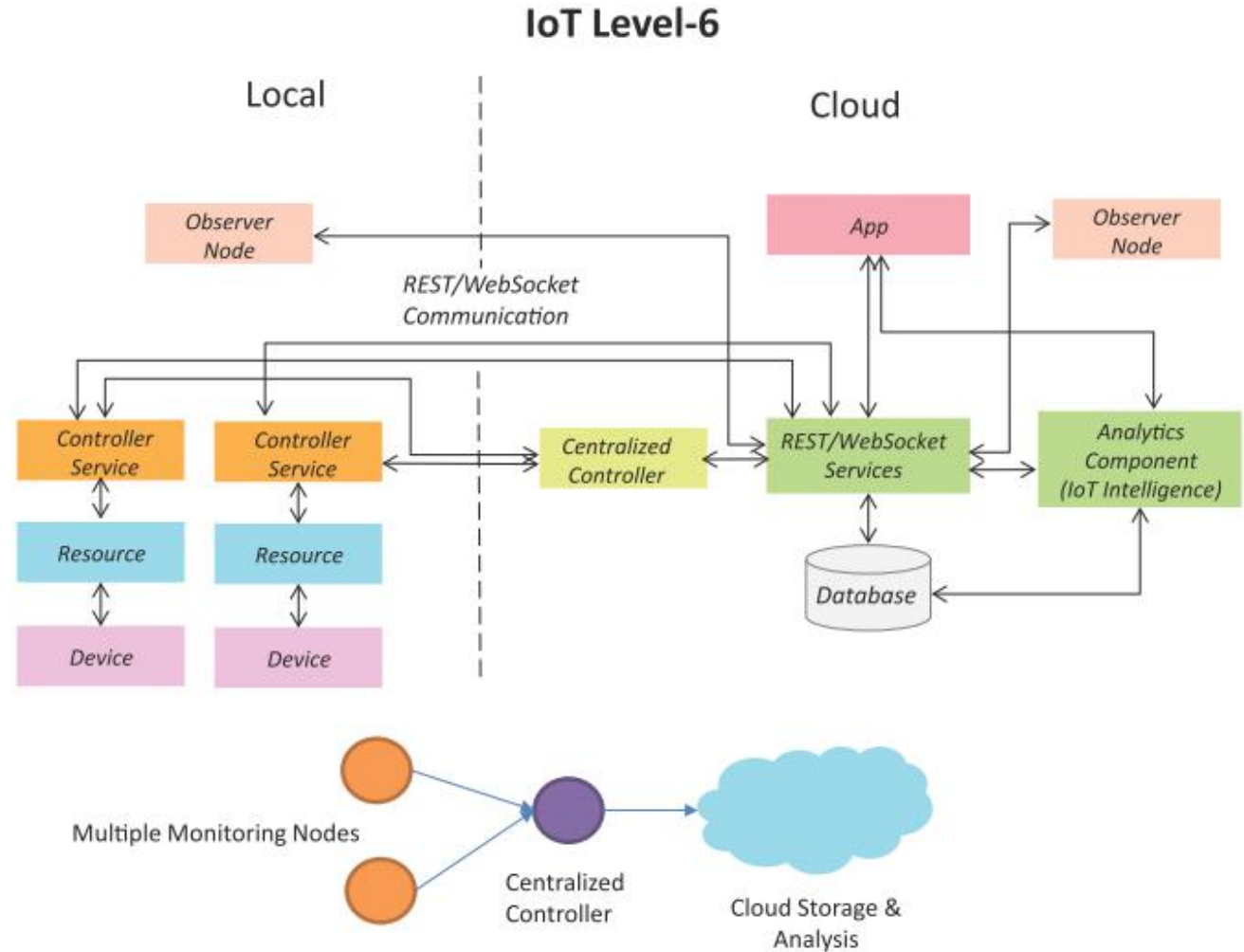
IoT Level-5

- A level-5 IoT system has multiple end nodes and one coordinator node.
- 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.
- Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.

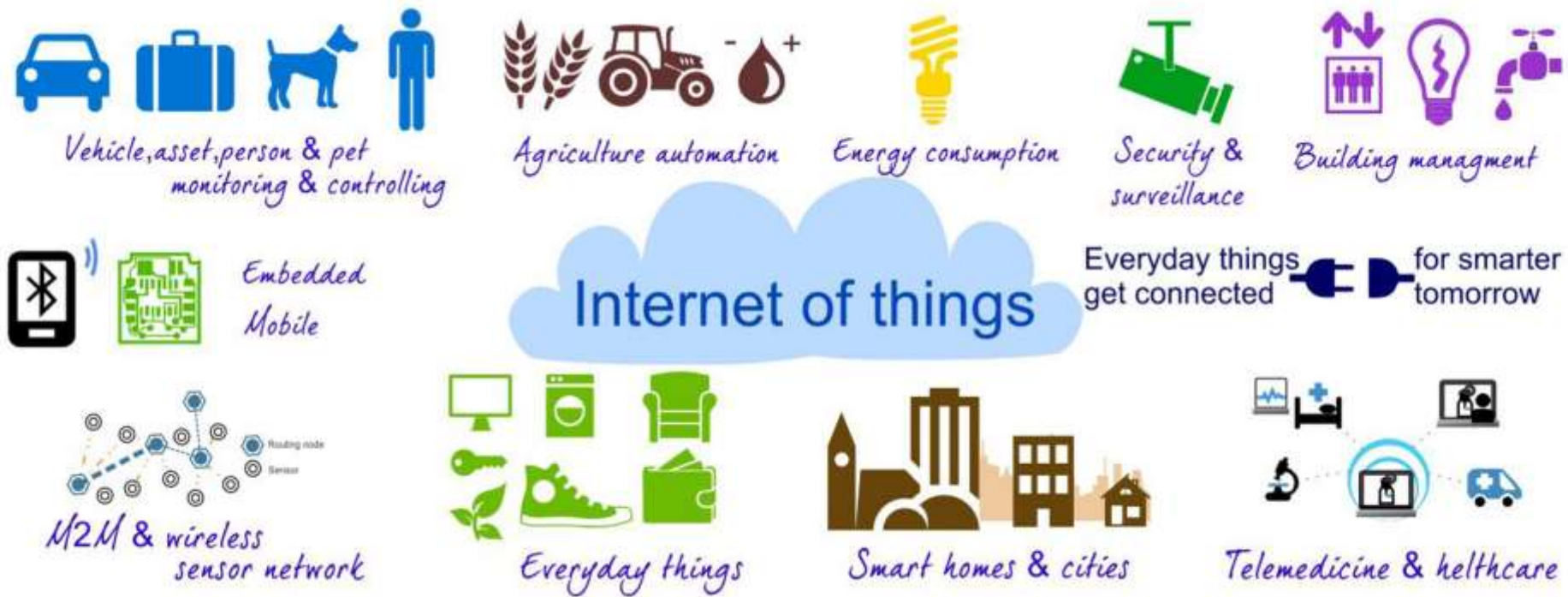


IoT Level-6

- A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.
- Data is stored in the cloud and application is cloud-based.
- The analytics component analyzes the data and stores the results in the cloud database.
- The results are visualized with the cloud-based application.
- The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.



Domain Specific IoTs



Outline

IoT Applications for :

- Home
- Cities
- Environment
- Energy Systems
- Retail
- Logistics
- Industry
- Agriculture
- Health & Lifestyle



Healthcare



Energy



Building



Retail



Security



Home



Education



Transportation



IoT



Agriculture



Factory



Cloud Computing



Home Automation

IoT applications for smart homes:

- *Smart Lighting*
- *Smart Appliances*
- *Intrusion Detection*
- *Smoke / Gas Detectors*



Home Automation

Smart Lighting

- Smart lighting achieve **energy savings** by sensing the human movements and their environments and controlling the lights accordingly.
- Key enabling technologies for smart lighting include :
 - ***Solid state lighting (such as LED lights)***
 - ***IP-enabled lights***
- Wireless-enabled and Internet connected lights can be controlled remotely from IoT applications such as a mobile or web application.
- Paper:
 - *Energy-aware wireless sensor network with ambient intelligence for smart LED lighting system control* [IECON, 2011] → presented controllable LED lighting system that is embedded with ambient intelligence gathered from a distributed smart WSN to optimize and control the lighting system to be more efficient and user-oriented.



Home Automation

Smart Appliances

- Smart appliances make the management easier and provide status information of appliances to the users remotely. **E.g:** smart washer/dryer that can be controlled remotely and notify when the washing/drying cycle is complete.
- **OpenRemote** is an open source automation platform for smart home and building that can control various appliances using mobile and web applications.
- It comprises of three components:
 - a Controller → manages scheduling and runtime integration between devices.
 - a Designer → allows to create both configuration for the controller and user interface designs.
 - Control Panel → allows to interact with devices and control them.
- Paper:
 - *An IoT-based Appliance Control System for Smart Home* [ICICIP, 2013] → implemented an IoT based appliance control system for smart homes that uses a smart-central controller to set up a wireless sensor and actuator network and control modules for appliances.



Home Automation

Intrusion Detection

- Home intrusion detection systems use *security cameras* and *sensors* to detect intrusions and raise alerts.
- The form of the alerts can be in form:
 - *SMS*
 - *Email*
 - *Image grab or a short video clip as an email attachment*
- Papers :
 - *Could controlled intrusion detection and burglary prevention stratagems in home automation systems* [BCFIC, 2012] → present a controlled intrusion detection system that uses location-aware services, where the geo-location of each node of a home automation system is independently detected and stored in the cloud
 - *An Intelligent Intrusion Detection System Based on UPnP Technology for Smart Living* [ISDA, 2008] → implement an intrusion detection system that uses image processing to recognize the intrusion and extract the intrusion subject and generate Universal-Plug-and-Play (UPnP-based) instant messaging for alerts.



Home Automation

Smoke / Gas Detectors

- Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire.
- It uses optical detection, ionization or air sampling techniques to detect smoke
- The form of the alert can be in form :
 - *Signals that send to a fire alarm system*
- Gas detector can detect the presence of harmful gases such as carbon monoxide (CO), liquid petroleum gas (LPG), etc.
- Paper :
 - *Development of Multipurpose Gas Leakage and Fire Detector with Alarm System [TIIEC, 2013] → designed a system that can detects gas leakage and smoke and gives visual level indication.*



Cities

IoT applications for smart cities:

1. *Smart Parking*
2. *Smart Lighting for Road*
3. *Smart Road*
4. *Structural Health Monitoring*
5. *Surveillance*
6. *Emergency Response*



Smart Parking

- Finding the parking space in the crowded city can be time consuming and frustrating
- Smart parking makes the search for parking space easier and convenient for driver.
- It can detect the number of empty parking slots and send the information over the Internet to the smart parking applications which can be accessed by the drivers using their smartphones, tablets, and in car navigation systems.
- Sensors are used for each parking slot to detect whether the slot is empty or not, and this information is aggregated by local controller and then sent over the Internet to database.
- Paper :
 - *Design and implementation of a prototype Smart PARKing (SPARK) system using WSN* [International Conference on Advanced Information Networking and Applications Workshop, 2009] → designed and implemented a prototype smart parking system based on wireless sensor network technology with features like remote parking monitoring, automate guidance, and parking reservation mechanism.



Smart Lighting for Roads

- It can help in saving energy
- Smart lighting for roads allows lighting to be dynamically controlled and also adaptive to ambient conditions.
- Smart light connected to the Internet can be controlled remotely to configure lighting schedules and lighting intensity.
- Custom lighting configurations can be set for different situations such as a foggy day, a festival, etc.
- Paper :
 - *Smart Lighting solutions for Smart Cities* [International Conference on Advance Information Networking and Applications Workshop, 2013] → described the need for smart lighting system in smart cities, smart lighting features and how to develop interoperable smart lighting solutions.



Smart Roads

- Smart Roads provides information on driving conditions, travel time estimates and alerts in case of poor driving conditions, traffic congestions and accidents.
- Such information can help in making the roads safer and help in reducing traffic jams
- Information sensed from the roads can be communicated via internet to cloud-based applications and social media and disseminated to the drivers who subscribe to such applications.
- Paper:
 - *Sensor networks for smart roads* [PerCom Workshop, 2006] → proposed a distributed and autonomous system of sensor network nodes for improving driving safety on public roads, the system can provide the driver and passengers with a consistent view of the road situation a few hundred metres ahead of them or a few dozen miles away, so that they can react to potential dangers early enough.



Structural Health Monitoring

- It uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.
- The data collected from these sensors is analyzed to assess the health of the structures.
- By analyzing the data it is possible to detect cracks and mechanical breakdowns, locate the damages to a structure and also calculate the remaining life of the structure.
- Using such systems, advance warnings can be given in the case of imminent failure of the structure.
- Paper:
 - *Environmental Effect Removal Based Structural Health Monitoring in the Internet of Things* [International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2013] → proposed an environmental effect removal based structural health monitoring scheme in an IoT environment.
 - *Energy harvesting technologies for structural health monitoring applications* [IEEE Conference on Technologies for Sustainability, 2013] → Explored energy harvesting technologies of harvesting ambient energy, such as mechanical vibrations, sunlight, and wind.



Surveillance

- Surveillance of infrastructure, public transport and events in cities is required to ensure safety and security.
- City wide surveillance infrastructure comprising of large number of distributed and Internet connected video surveillance cameras can be created.
- The video feeds from surveillance cameras can be aggregated in cloud-based scalable storage solutions.
- Cloud-based video analytics applications can be developed to search for patterns of specific events from the video feeds.



Emergency Response

- IoT systems can be used for monitoring the critical infrastructure cities such as buildings, gas, and water pipelines, public transport and power substations.
- IoT systems for critical infrastructure monitoring enable aggregation and sharing of information collected from larger number of sensors.
- Using cloud-based architectures, multi-modal information such as sensor data, audio, video feeds can be analyzed in near real-time to detect adverse events.
- The alert can be in the form :
 - Alerts sent to the public
 - Re-rerouting of traffic
 - Evacuations of the affected areas



Environment

IoT applications for smart environments:

- 1. Weather Monitoring*
- 2. Air Pollution Monitoring*
- 3. Noise Pollution Monitoring*
- 4. Forest Fire Detection*
- 5. River Flood Detection*



Environment

Weather Monitoring

- It collects data from a number of sensor attached such as temperature, humidity, pressure, etc and send the data to cloud-based applications and store back-ends.
- The data collected in the cloud can then be analyzed and visualized by cloud-based applications.
- Weather alert can be sent to the subscribed users from such applications.
- AirPi is a weather and air quality monitoring kit capable of recording and uploading information about temperature, humidity, air pressure, light levels, UV levels, carbon monoxide, nitrogen dioxide and smoke level to the Internet.
- Paper:
 - PeWeMoS – Pervasive Weather Monitoring System [ICPCA, 2008] → Presented a pervasive weather monitoring system that is integrated with buses to measure weather variables like humidity, temperature, and air quality during the bus path



Environment

Air Pollution Monitoring

- IoT based air pollution monitoring system can monitor emission of harmful gases by factories and automobiles using gaseous and meteorological sensors.
- The collected data can be analyzed to make informed decisions on pollutions control approaches.
- Paper:
 - Wireless sensor network for real-time air pollution monitorings [ICCSPA, 2013] → Presented a real time air quality monitoring system that comprises of several distributed monitoring stations that communicate via wireless with a back-end server using machine-to machine communication.



Environment

Noise Pollution Monitoring

- Noise pollution monitoring can help in generating noise maps for cities.
- It can help the policy maker in making policies to control noise levels near residential areas, school and parks.
- It uses a number of noise monitoring stations that are deployed at different places in a city.
- The data on noise levels from the stations is collected on servers or in the cloud and then the collected data is aggregate to generate noise maps.
- Papers :
 - Noise mapping in urban environments : Applications at Suez city center [ICCIE, 2009]Presented a noise mapping study for a city which revealed that the city suffered from serious noise pollution.
 - SoundOfCity – Continuous noise monitoring for a health city [PerComW,2013] → Designed a smartphone application that allows the users to continuously measure noise levels and send to a central server here all generated information is aggregated and mapped to a meaningful noise visualization map.



Environment

Forest Fire Detection

- IoT based forest fire detection system use a number of monitoring nodes deployed at different location in a forest.
- Each monitoring node collects measurements on ambient condition including temperature, humidity, light levels, etc.
- Early detection of forest fires can help in minimizing the damage.
- Papers:
 - *A novel accurate forest fire detection system using wireless sensor networks* [International Conference on Mobile Ad-hoc and Sensor Networks, 2011] → Presented a forest fire detection system based on wireless sensor network. The system uses multi-criteria detection which is implemented by the artificial neural network. The ANN fuses sensing data corresponding to ,multiple attributes of a forest fire such as temperature, humidity, infrared and visible light to detect forest fires.



Environment

River Flood Detection

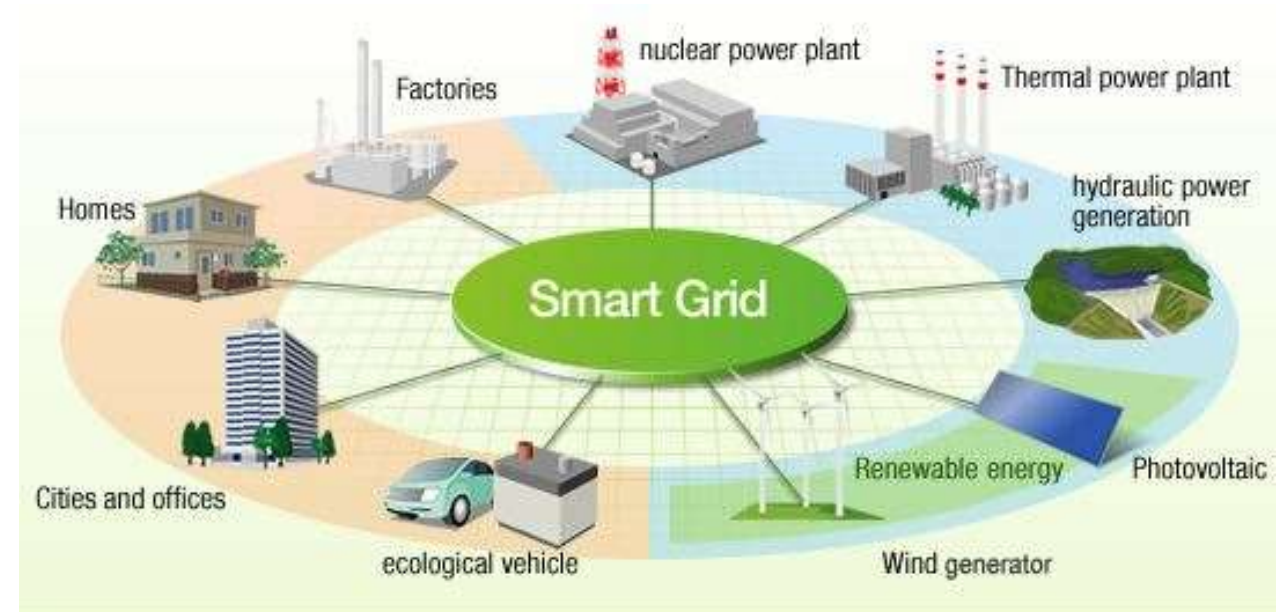
- IoT based river flood monitoring system uses a number of sensor nodes that monitor the water level using ultrasonic sensors and flow rate using velocity sensors.
- Data from these sensors is aggregated in a server or in the cloud, monitoring applications raise alerts when rapid increase in water level and flow rate is detected.
- Papers:
 - RFMS : Real time flood monitoring system with wireless sensor networks [MASS, 2008] → Described a river flood monitoring system that measures river and weather conditions through wireless sensor nodes equipped with different sensors
 - Urban Flash Flood Monitoring, Mapping and Forecasting via a Tailored Sensor Network System [ICNSC, 2006] → Described a motes-based sensor network for river flood monitoring that includes a water level monitoring module, network video recorder module, and data processing module that provides floods information in the form of raw data, predict data, and video feed.



Energy

IoT applications for smart energy systems:

1. *Smart Grid*
2. *Renewable Energy Systems*
3. *Prognostics*



Energy

Smart Grids

- Smart grid technology provides predictive information and recommendations to utilize, their suppliers, and their customers on how best to manage power.
- Smart grid collect the data regarding :
 - Electricity generation
 - Electricity consumption
 - Storage
 - Distribution and equipment health data
- By analyzing the data on power generation, transmission and consumption of smart grids can improve efficiency throughout the electric system.
- Storage collection and analysis of smart grids data in the cloud can help in dynamic optimization of system operations, maintenance, and planning.
- Cloud-based monitoring of smart grids data can improve energy usage levels via energy feedback to users coupled with real-time pricing information.
- Condition monitoring data collected from power generation and transmission systems can help in detecting faults and predicting outages.



Renewable Energy System

- Due to the variability in the output from renewable energy sources (such as solar and wind), integrating them into the grid can cause grid stability and reliability problems.
- IoT based systems integrated with the transformer at the point of interconnection measure the electrical variables and how much power is fed into the grid
- To ensure the grid stability, one solution is to simply cut off the overproductions.
- Paper:
 - *Communication systems for grid integration of renewable energy resources* [IEEE Network, 2011] → Provided the closed-loop controls for wind energy system that can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides reactive power support.



Prognostics

- IoT based prognostic real-time health management systems can predict performance of machines of energy systems by analyzing the extent of deviation of a system from its normal operating profiles.
- In the system such as power grids, real time information is collected using specialized electrical sensors called Phasor Measurement Units (PMU)
- Analyzing massive amounts of maintenance data collected from sensors in energy systems and equipment can provide predictions for impending failures.
- OpenPDC is a set of applications for processing of streaming time-series data collected from Phasor Measurements Units (PMUs) in real-time.



Retail

IoT applications in smart retail systems:

1. *Inventory Management*
2. *Smart Payments*
3. *Smart Vending Machines*



Inventory Management

- IoT system using Radio Frequency Identification (RFID) tags can help inventory management and maintaining the right inventory levels.
- RFID tags attached to the products allow them to be tracked in the real-time so that the inventory levels can be determined accurately and products which are low on stock can be replenished.
- Tracking can be done using RFID readers attached to the retail store shelves or in the warehouse.
- Paper:
 - *RFID data-based inventory management of time-sensitive materials* [IECON, 2005] → described an RFID data-based inventory management system for time-sensitive materials



Smart Payments

- Smart payments solutions such as contact-less payments powered technologies such as Near field communication (NFC) and Bluetooth.
- NFC is a set of standards for smart-phones and other devices to communicate with each other by bringing them into proximity or by touching them
- Customer can store the credit card information in their NFC-enabled smart-phones and make payments by bringing the smart-phone near the point of sale terminals.
- NFC maybe used in combination with Bluetooth, where NFC initiates initial pairing of devices to establish a Bluetooth connection while the actual data transfer takes place over Bluetooth.



Smart Vending Machines

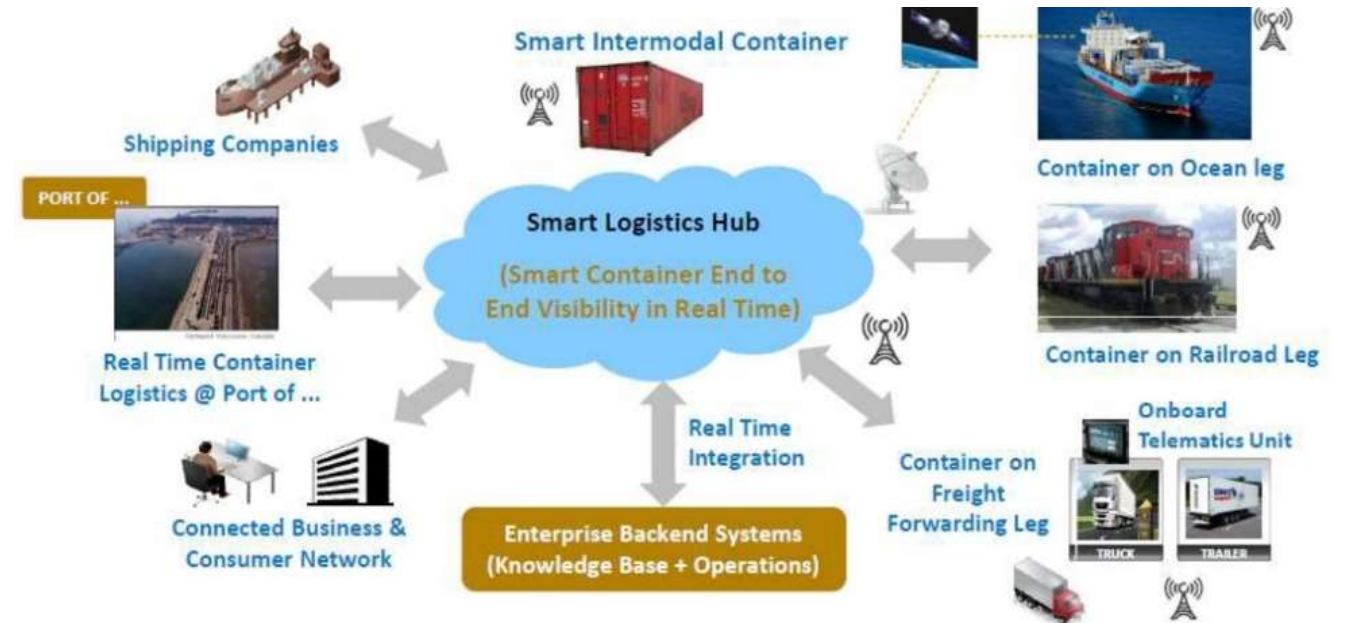
- Smart vending machines connected to the Internet allow remote monitoring of inventory levels, elastic pricing of products, promotions, and contact-less payments using NFC.
- Smart-phone applications that communicate with smart vending machines allow user preferences to be remembered and learned with time. E.g: when a user moves from one vending machine to the other and pair the smart-phone, the user preference and favorite product will be saved and then that data is used for predictive maintenance.
- Smart vending machines can communicate each others, so if a product out of stock in a machine, the user can be routed to nearest machine
- For perishable items, the smart vending machines can reduce the price as the expiry date nears.



Logistic

IoT applications for smart logistic systems:

1. *Fleet Tracking*
2. *Shipment Monitoring*
3. *Remote Vehicle Diagnostics*



Fleet Tracking

- Vehicle fleet tracking systems use GPS technology to track the locations of the vehicles in the real-time.
- Cloud-based fleet tracking systems can be scaled up on demand to handle large number of vehicles,
- The vehicle locations and routers data can be aggregated and analyzed for detecting bottlenecks I the supply chain such as traffic congestions on routes, assignments and generation of alternative routes, and supply chain optimization
- Paper:
 - *A Fleet Monitoring System for Advanced Tracking of commercial Vehicles* [IEEE International Conference in Systems, Man and Cybernetics, 2006] → provided a system that can analyze messages sent from the vehicles to identify unexpected incidents and discrepancies between actual and planned data, so that remedial actions can be taken.



Shipment Monitoring

- Shipment monitoring solutions for transportation systems allow monitoring the conditions inside containers.
- E.g : Containers carrying fresh food produce can be monitored to prevent spoilage of food. IoT based shipment monitoring systems use sensors such as temperature, pressure, humidity, for instance, to monitor the conditions inside the containers and send the data to the cloud, where it can be analyzed to detect food spoilage.
- Paper:
 - *On a Cloud-Based Information Technology Framework for Data Driven Intelligent Transportation System* [Journal of Transportation Technologies, 2013] → proposed a cloud based framework for real time fresh food supply tracking and monitoring
 - *Container Integrity and Condition Monitoring using RF Vibration Sensor Tags* [IEEE International Conference on Automation Science and Engineering, 2007] → Proposed a system that can monitor the vibrations patterns of a container and its contents to reveal information related to its operating environment and integrity during transport, handling, and storage.



Remote Vehicle Diagnostics

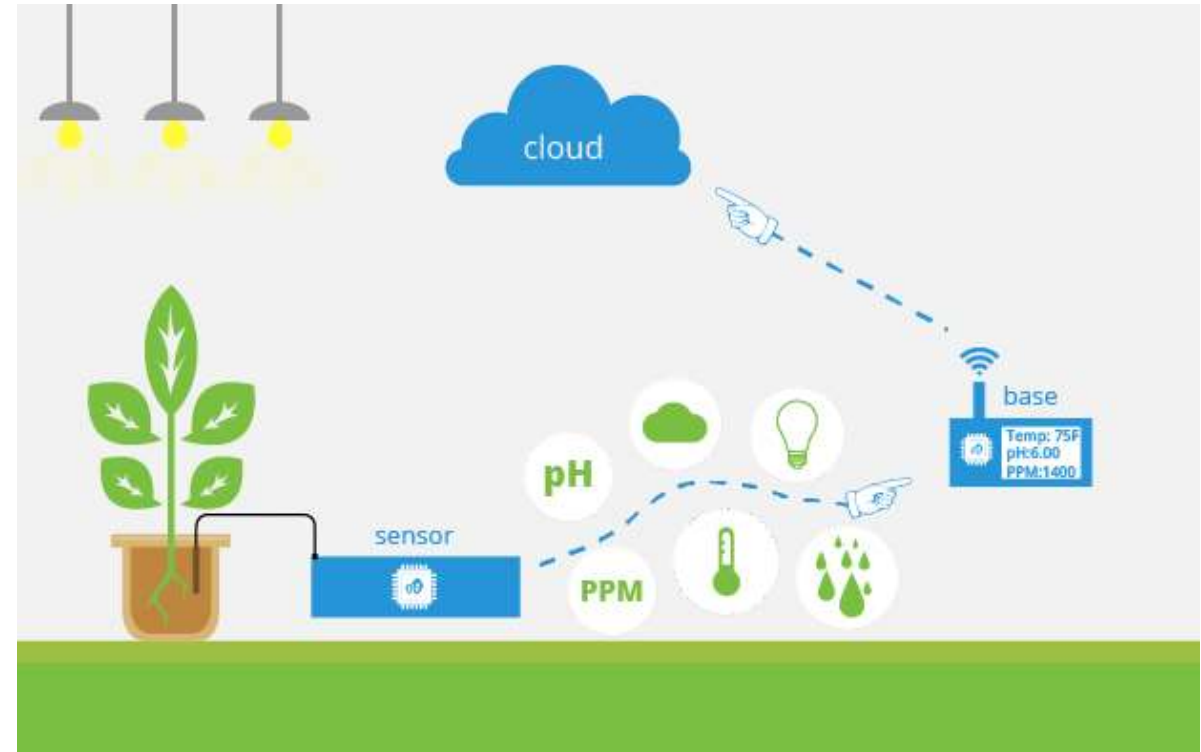
- It can detect faults in the vehicles or warn of impending faults.
- These diagnostic systems use on-board IoT devices for collecting data on vehicle operation such as speed, engine RPM, coolant temperature, fault code number and status of various vehicle sub-system.
- Modern commercial vehicles support on-board diagnostic (OBD) standard such as OBD-II
- OBD systems provide real-time data on the status of vehicle sub-systems and diagnostic trouble codes which allow rapidly identifying the faults in the vehicle.
- IoT based vehicle diagnostic systems can send the vehicle data to centralized servers or the cloud where it can be analyzed to generate alerts and suggest remedial actions.



Agriculture

IoT applications for smart agriculture:

1. *Smart Irrigation*
2. *Green House Control*



Agriculture

Smart Irrigation

- Smart irrigation system can improve crop yields while saving water.
- Smart irrigation systems use IoT devices with soil moisture sensors to determine the amount of moisture on the soil and release the flow of the water through the irrigation pipes only when the moisture levels go below a predefined threshold.
- It also collects moisture level measurements on the server or in the cloud where the collected data can be analyzed to plan watering schedules.
- *Cultivar's RainCloud* is a device for smart irrigation that uses water valves, soil sensors, and a WiFi enabled programmable computer. [<http://ecultivar.com/rain-cloud-product-project/>]



Agriculture

Green House Control

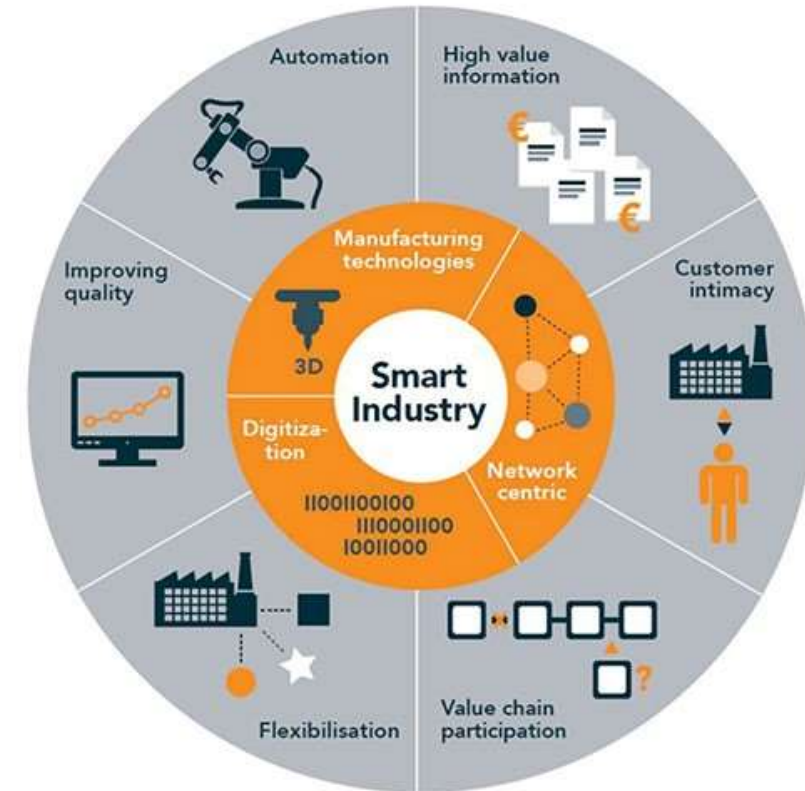
- It controls temperature, humidity, soil, moisture, light, and carbon dioxide level that are monitored by sensors and climatological conditions that are controlled automatically using actuation devices.
- IoT systems play an importance role in green house control and help in improving productivity.
- The data collected from various sensors is stored on centralized servers or in the cloud where analysis is performed to optimize the control strategies and also correlate the productivity with different control strategies.
- Paper:
 - *Wireless sensing and control for precision Green house management* [ICST, 2012] → Provided a system that uses wireless sensor network to monitor and control the agricultural parameters like temperature and humidity in the real time for better management and maintenance of agricultural production.



Industry

IoT applications in smart industry:

1. *Machine Diagnosis & Prognosis*
2. *Indoor Air Quality Monitoring*



Machine Diagnosis & Prognosis

- Machine prognosis refers to predicting the performance of machine by analyzing the data on the current operating conditions and how much deviations exist from the normal operating condition.
- Machine diagnosis refers to determining the cause of a machine fault.
- Sensors in machine can monitor the operating conditions such as temperature and vibration levels, sensor data measurements are done on timescales of few milliseconds to few seconds which leads to generation of massive amount of data.
- Case-based reasoning (CBR) is a commonly used method that finds solutions to new problems based on past experience.
- CBR is an effective technique for problem solving in the fields in which it is hard to establish a quantitative mathematical model, such as machine diagnosis and prognosis.



Indoor Air Quality Monitoring

- Harmful and toxic gases such as carbon monoxide (CO), nitrogen monoxide (NO), Nitrogen Dioxide, etc can cause serious health problem of the workers.
- IoT based gas monitoring systems can help in monitoring the indoor air quality using various gas sensors.
- The indoor air quality can be placed for different locations
- Wireless sensor networks based IoT devices can identify the hazardous zones, so that corrective measures can be taken to ensure proper ventilation.
- Papers:
 - *A hybrid sensor system for indoor air quality monitoring* [IEEE International Conference on Distributed Computing in Sensor System, 2013] → presented a hybrid sensor system for indoor air quality monitoring which contains both stationary sensor and mobile sensors.
 - *Indoor air quality monitoring using wireless sensor network* [International Conference on Sensing Technology, 2012] → provided a wireless solution for indoor air quality monitoring that measures the environmental parameters like temperature, humidity, gaseous pollutants , aerosol and particulate matter to determine the indoor air quality.



Health & Lifestyle

IoT applications in smart health & lifestyle:

1. *Health & Fitness Monitoring*
2. *Wearable Electronics*



Health & Lifestyle

Health & Fitness Monitoring

- Wearable IoT devices allow to continuous monitoring of physiological parameters such as blood pressure, heart rate, body temperature, etc than can help in continuous health and fitness monitoring.
- It can analyze the collected health-care data to determine any health conditions or anomalies.
- The wearable devices may can be in various form such as:
 - Belts
 - Wrist-bands
- Papers:
 - *Toward ubiquitous mobility solutions for body sensor network health care* [IEEE Communications Magazine, 2012] → Proposed an ubiquitous mobility approach for body sensor network in health-care
 - *A wireless sensor network compatible wearable u-healthcare monitoring system using integrated ECG, accelerometer and SpO2* [International Conference of the IEEE Engineering in Medicine and Biology Society, 2008] → Designed a wearable ubiquitous health-care monitoring system that uses integrated electrocardiogram (ECG), accelerometer and oxygen saturation (SpO2) sensors.



Health & Lifestyle

Wearable Electronics

- Wearable electronics such as wearable gadgets (smart watch, smart glasses, wristbands, etc) provide various functions and features to assist us in our daily activities and making us lead healthy lifestyles.
- Using the smart watch, the users can search the internet, play audio/video files, make calls, play games, etc.
- Smart glasses allows users to take photos and record videos, get map directions, check flight status or search internet using voice commands
- Smart shoes can monitor the walking or running speeds and jumps with the help of embedded sensors and be paired with smart-phone to visualize the data.
- Smart wristbands can track the daily exercise and calories burnt.



Chapter 3

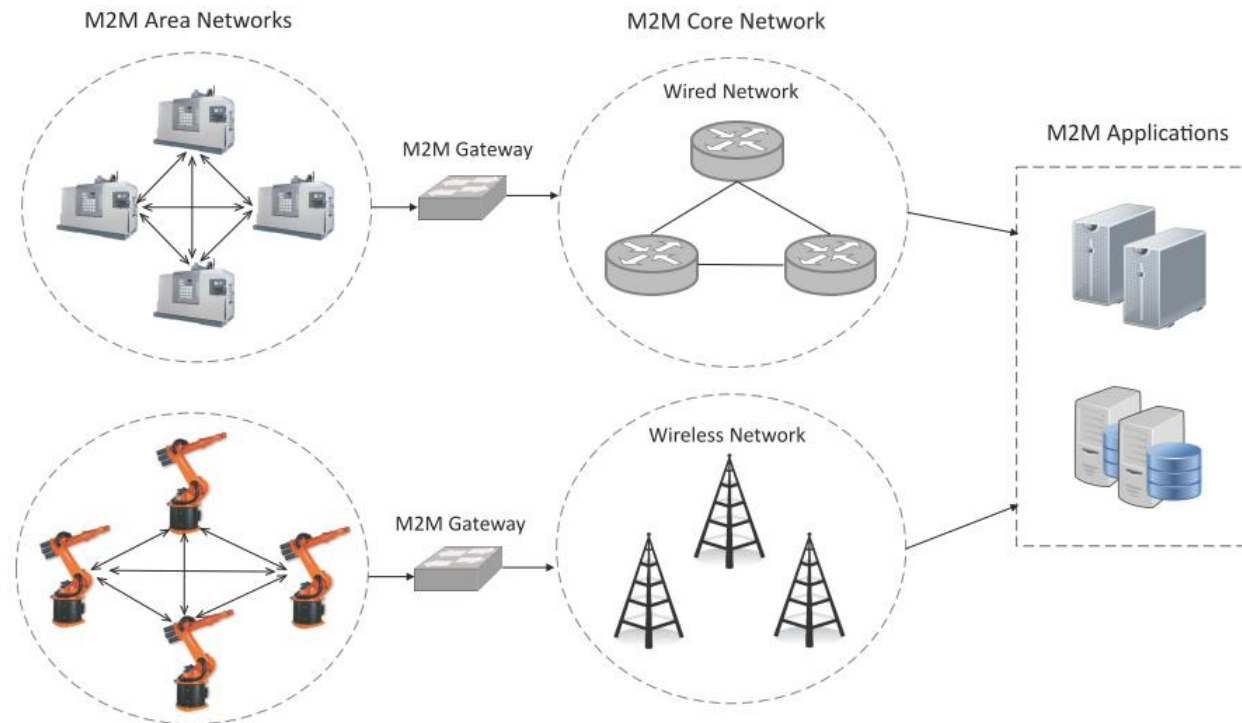
IoT & M2M

Outline

- M2M
- Differences and Similarities between M2M and IoT
- SDN and NFV for IoT

Machine-to-Machine (M2M)

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

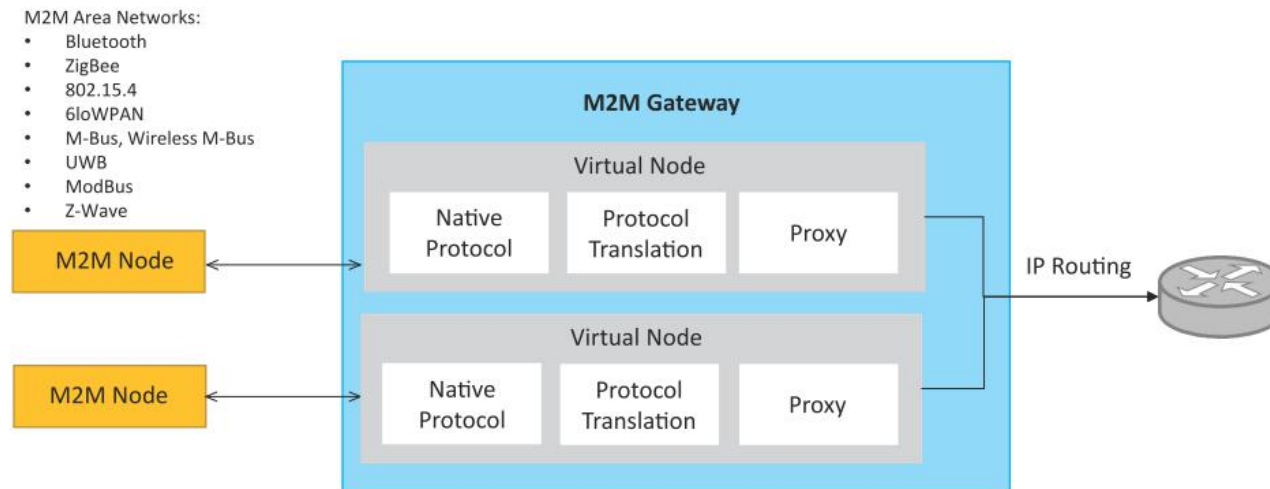


Machine-to-Machine (M2M)

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

M2M gateway

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



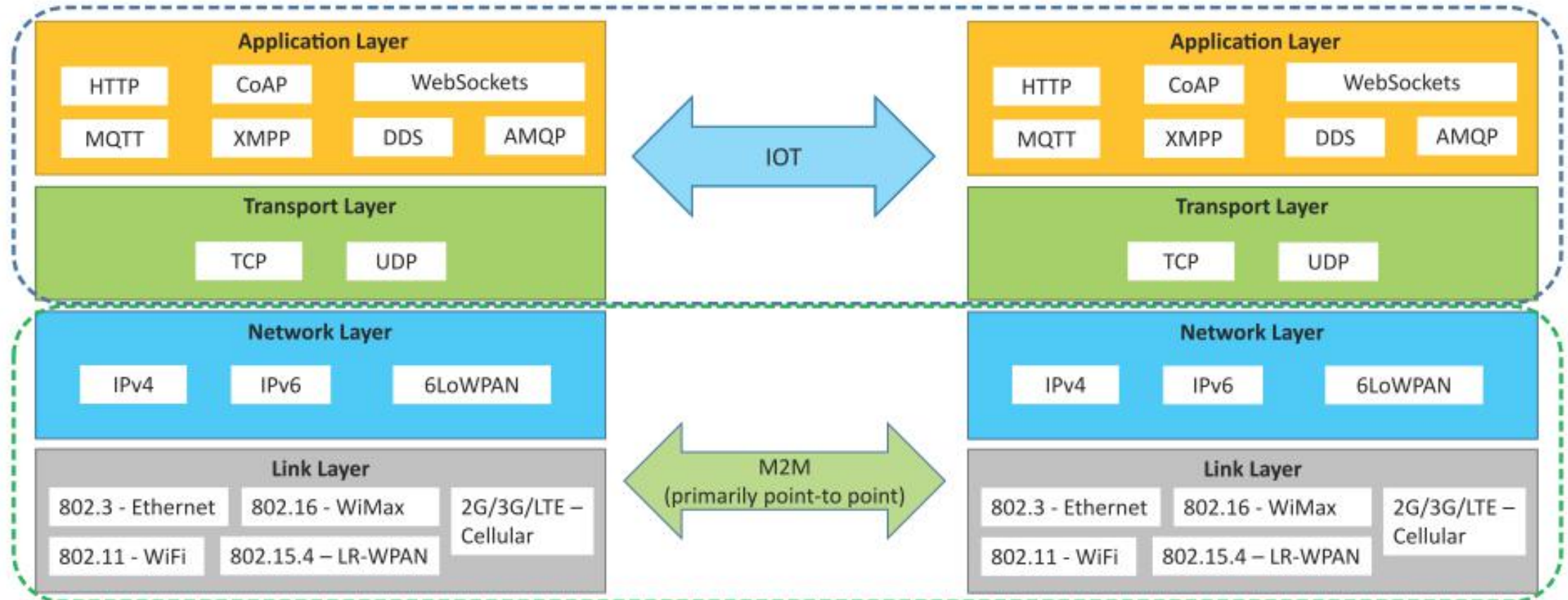
Difference between IoT and M2M

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

Difference between IoT and M2M

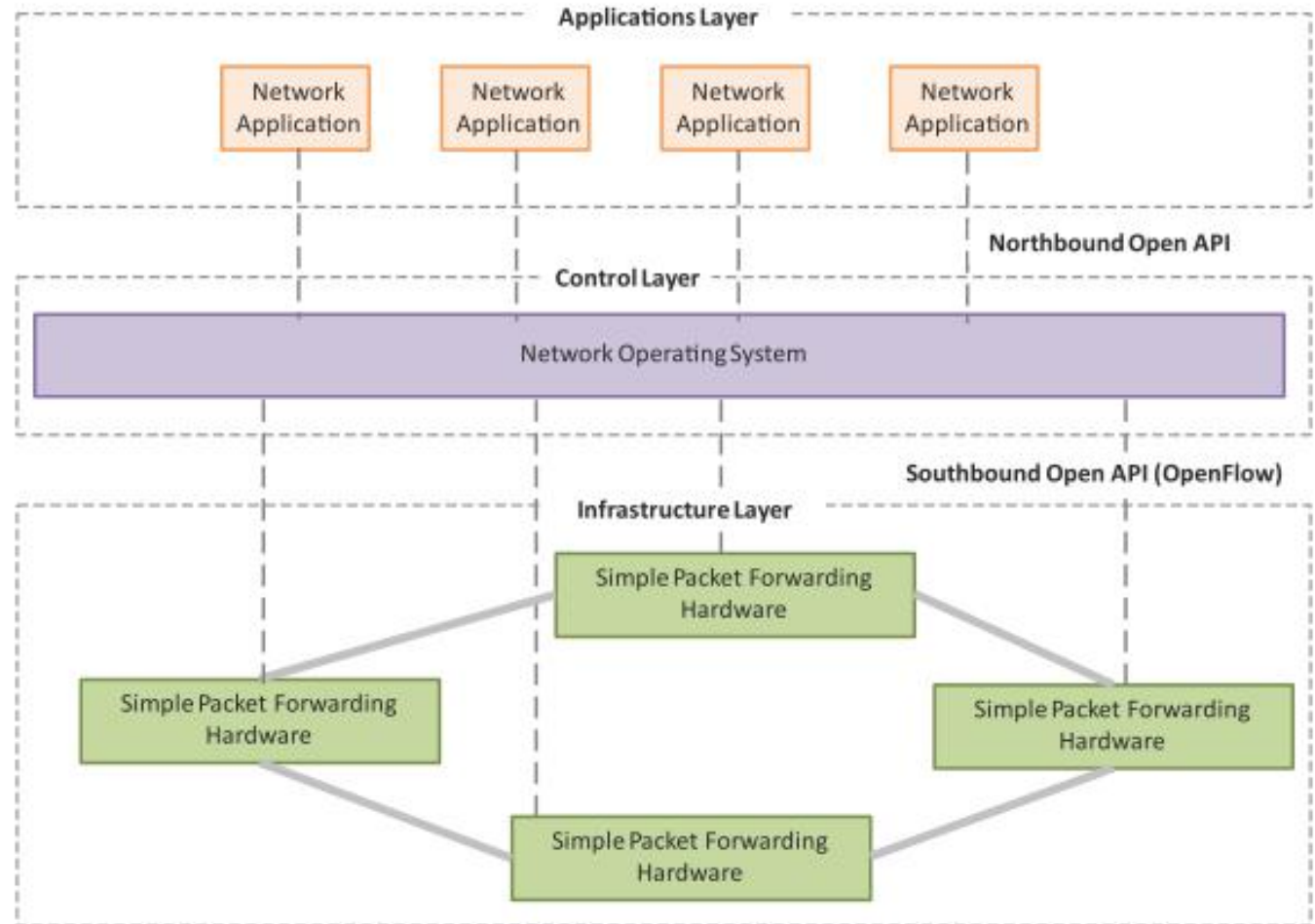
- Hardware vs Software Emphasis
 - While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
- Data Collection & Analysis
 - M2M data is collected in point solutions and often in on-premises storage infrastructure.
 - In contrast to M2M, the data in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

Communication in IoT vs M2M



SDN

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

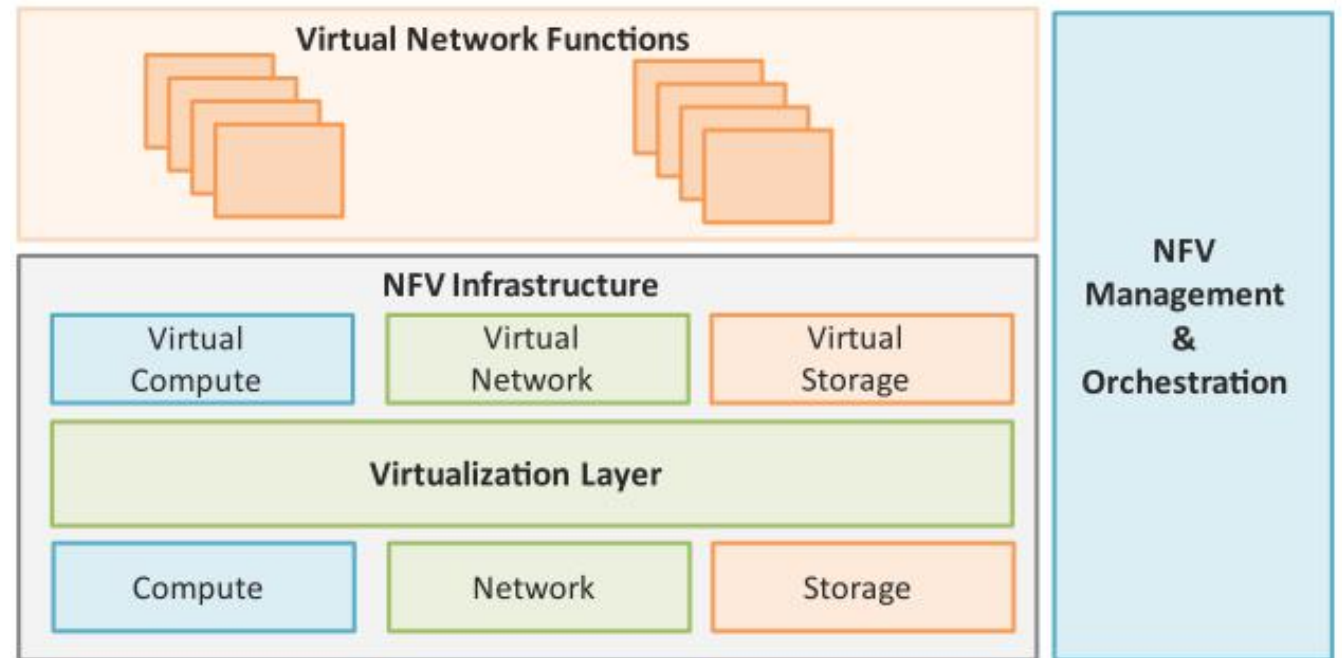


Key elements of SDN

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

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.

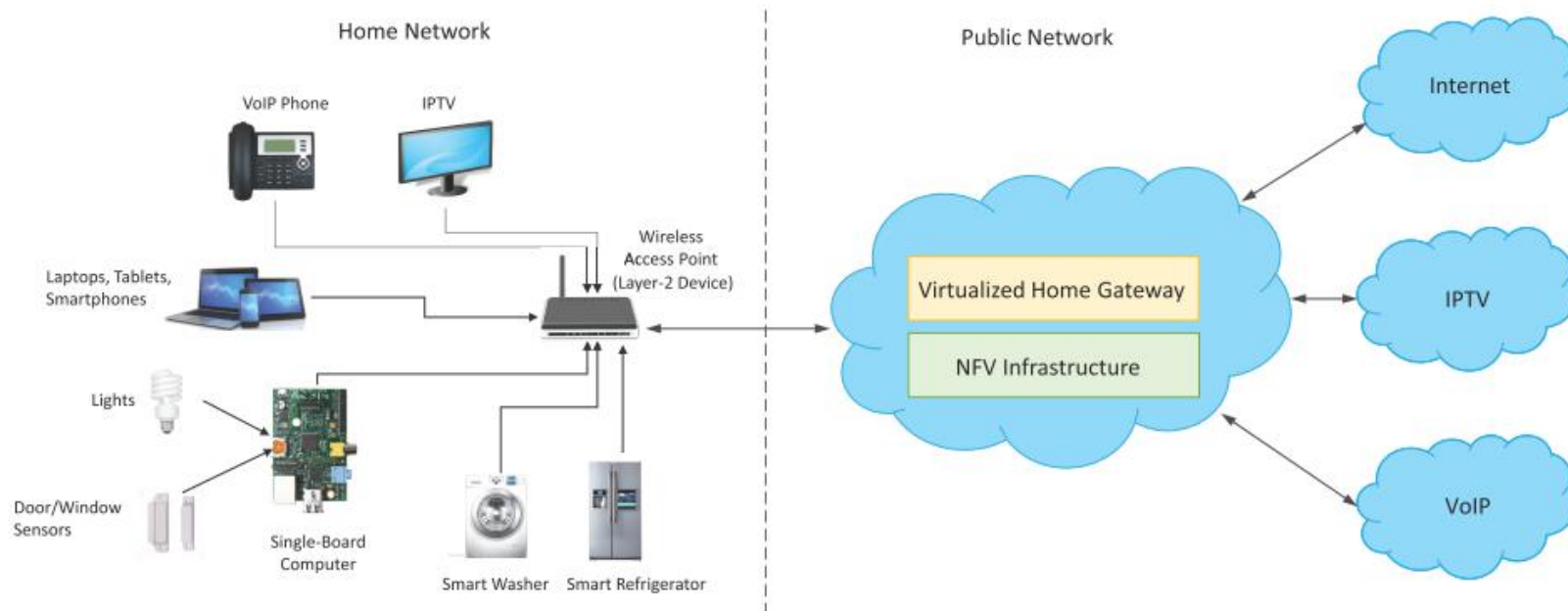


Key elements of NFV

- Virtualized Network Function (VNF):
 - VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
- NFV Infrastructure (NFVI):
 - NFVI includes compute, network and storage resources that are virtualized.
- 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.

NFV Use Case

- NFV can be used to virtualize the Home Gateway. The NFV infrastructure in the cloud hosts a virtualized Home Gateway. The virtualized gateway provides private IP addresses to the devices in the home. The virtualized gateway also connects to network services such as VoIP and IPTV.



Chapter 4

IoT System Management with NETCONF-YANG

Outline

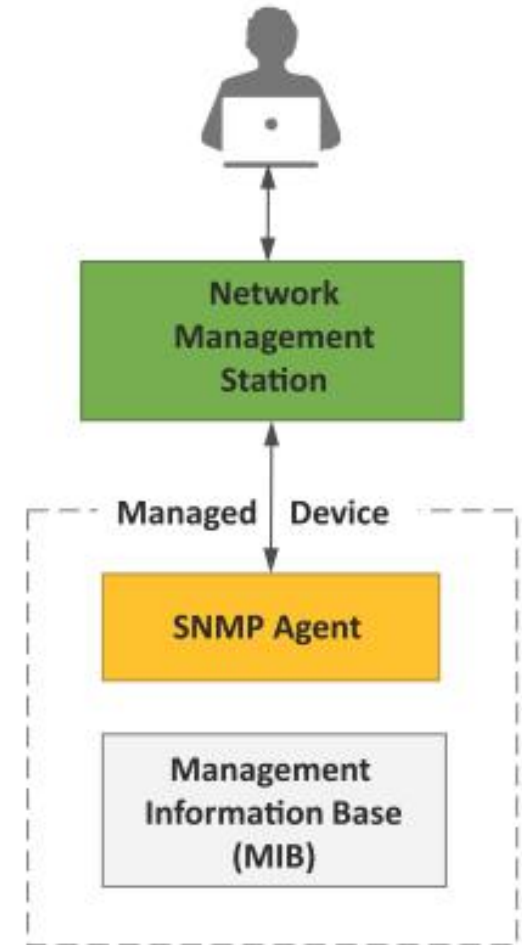
- Need for IoT Systems Management
- SNMP
- Network Operator Requirements
- NETCONF
- YANG
- IoT Systems Management with NETCONF-YANG

Need for IoT Systems Management

- Automating Configuration
- Monitoring Operational & Statistical Data
- Improved Reliability
- System Wide Configurations
- Multiple System Configurations
- Retrieving & Reusing Configurations

Simple Network Management Protocol (SNMP)

- SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc.
- SNMP components include
 - Network Management Station (NMS)
 - Managed Device
 - Management Information Base (MIB)
 - SNMP Agent that runs on the device



Limitations of SNMP

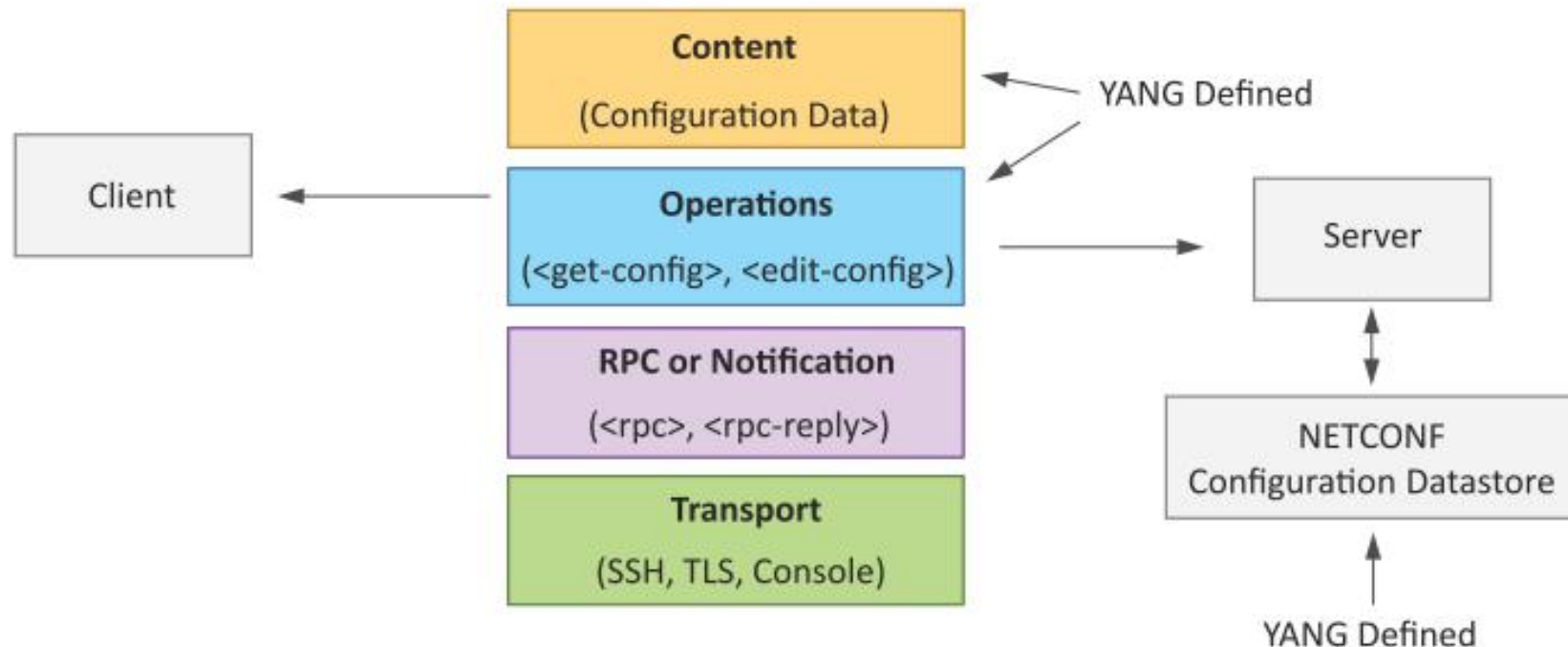
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features.

Network Operator Requirements

- Ease of use
- Distinction between configuration and state data
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration validation
- Configuration database schemas
- Comparing configurations
- Role-based access control
- Consistency of access control lists:
- Multiple configuration sets
- Support for both data-oriented and task-oriented access control

NETCONF

- Network Configuration Protocol (NETCONF) is a session-based network management protocol. NETCONF allows retrieving state or configuration data and manipulating configuration data on network devices



NETCONF

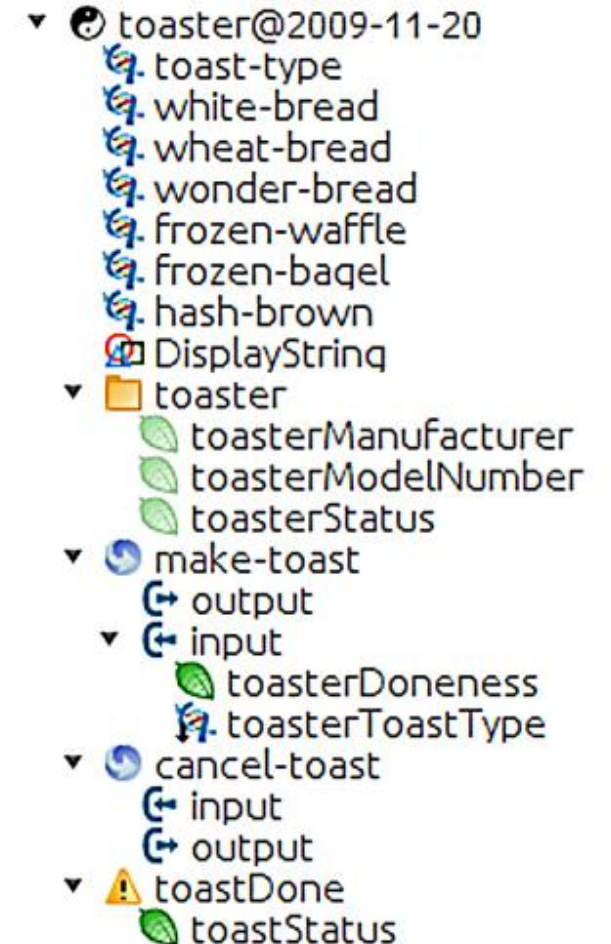
- NETCONF works on SSH transport protocol.
- Transport layer provides end-to-end connectivity and ensure reliable delivery of messages.
- NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.
- The RPC layer provides mechanism for encoding of RPC calls and notifications.
- 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.

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.
- A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure.
- The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs.
- Leaf nodes are organized using 'container' or 'list' constructs.
- A YANG module can import definitions from other modules.
- Constraints can be defined on the data nodes, e.g. allowed values.
- YANG can model both configuration data and state data using the 'config' statement.

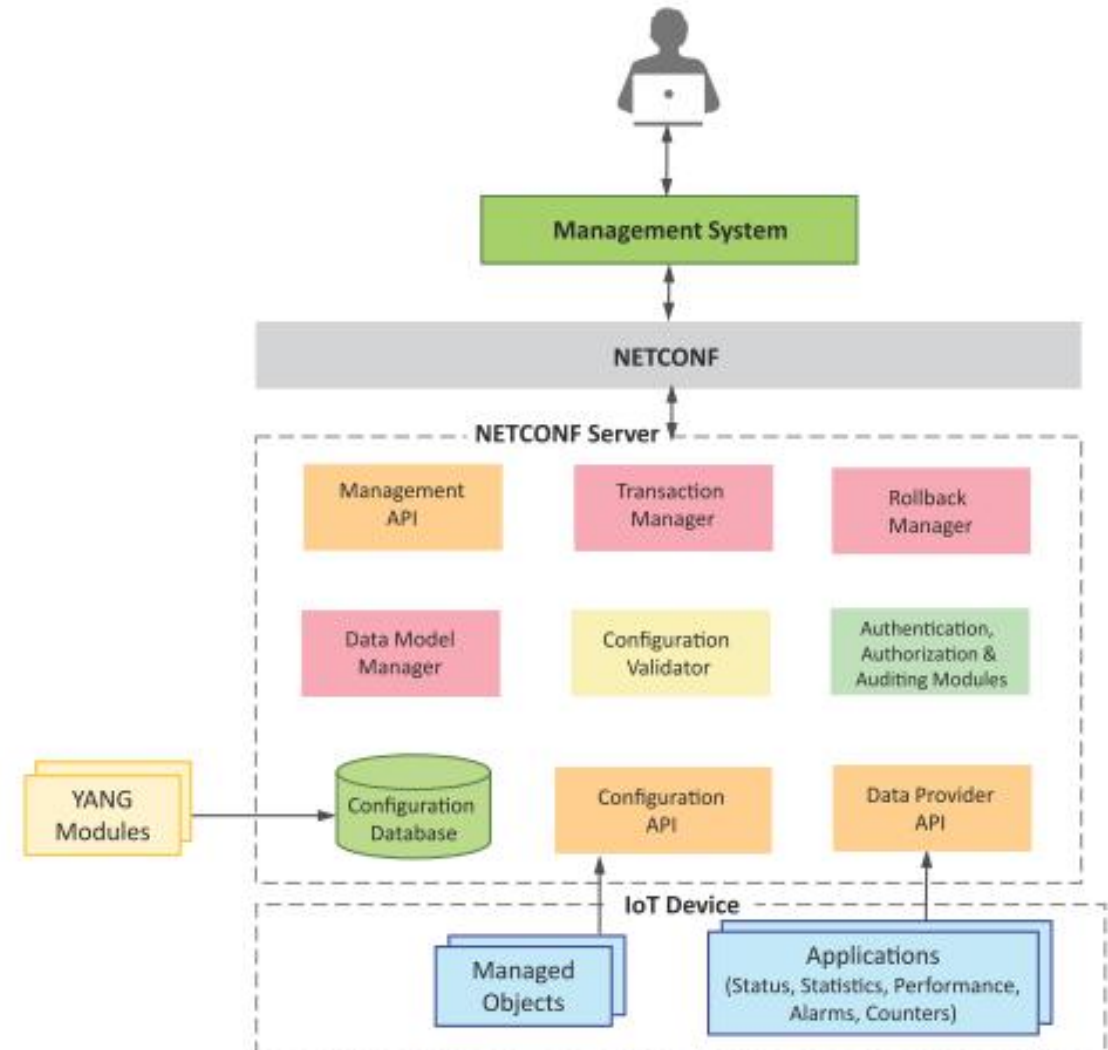
YANG Module Example

- This YANG module is a YANG version of the toaster MIB
- The toaster YANG module begins with the header information followed by identity declarations which define various bread types.
- The leaf nodes ('toasterManufacturer', 'toasterModelNumber' and toasterStatus') are defined in the 'toaster' container.
- Each leaf node definition has a type and optionally a description and default value.
- The module has two RPC definitions ('make-toast' and 'cancel-toast').



IoT Systems Management with NETCONF-YANG

- Management System
- Management API
- Transaction Manager
- Rollback Manager
- Data Model Manager
- Configuration Validator
- Configuration Database
- Configuration API
- Data Provider API



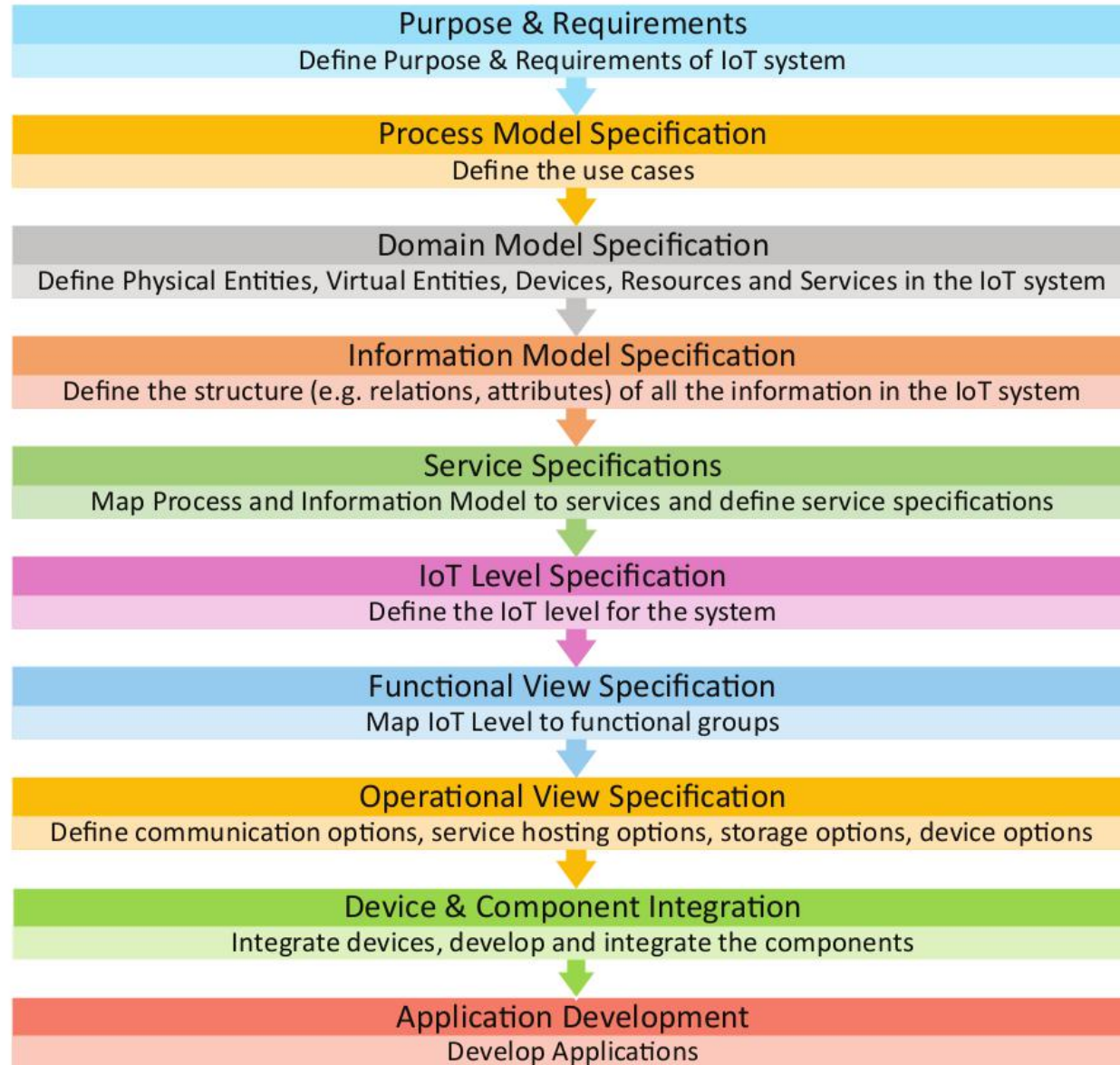
Chapter 5

IoT Design Methodology

Outline

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

IoT Design Methodology - Steps



Step 1: Purpose & Requirements Specification

- The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured.

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 Specification

- The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

Step 4: Information Model Specification

- The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.

Step 5: Service Specifications

- The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

Step 6: IoT Level Specification

- The sixth step in the IoT design methodology is to define the IoT level for the system. In Chapter-1, we defined five IoT deployment levels.

Step 7: Functional View Specification

- The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.

Step 8: Operational View Specification

- The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc

Step 9: Device & Component Integration

- The ninth step in the IoT design methodology is the integration of the devices and components.

Step 10: Application Development

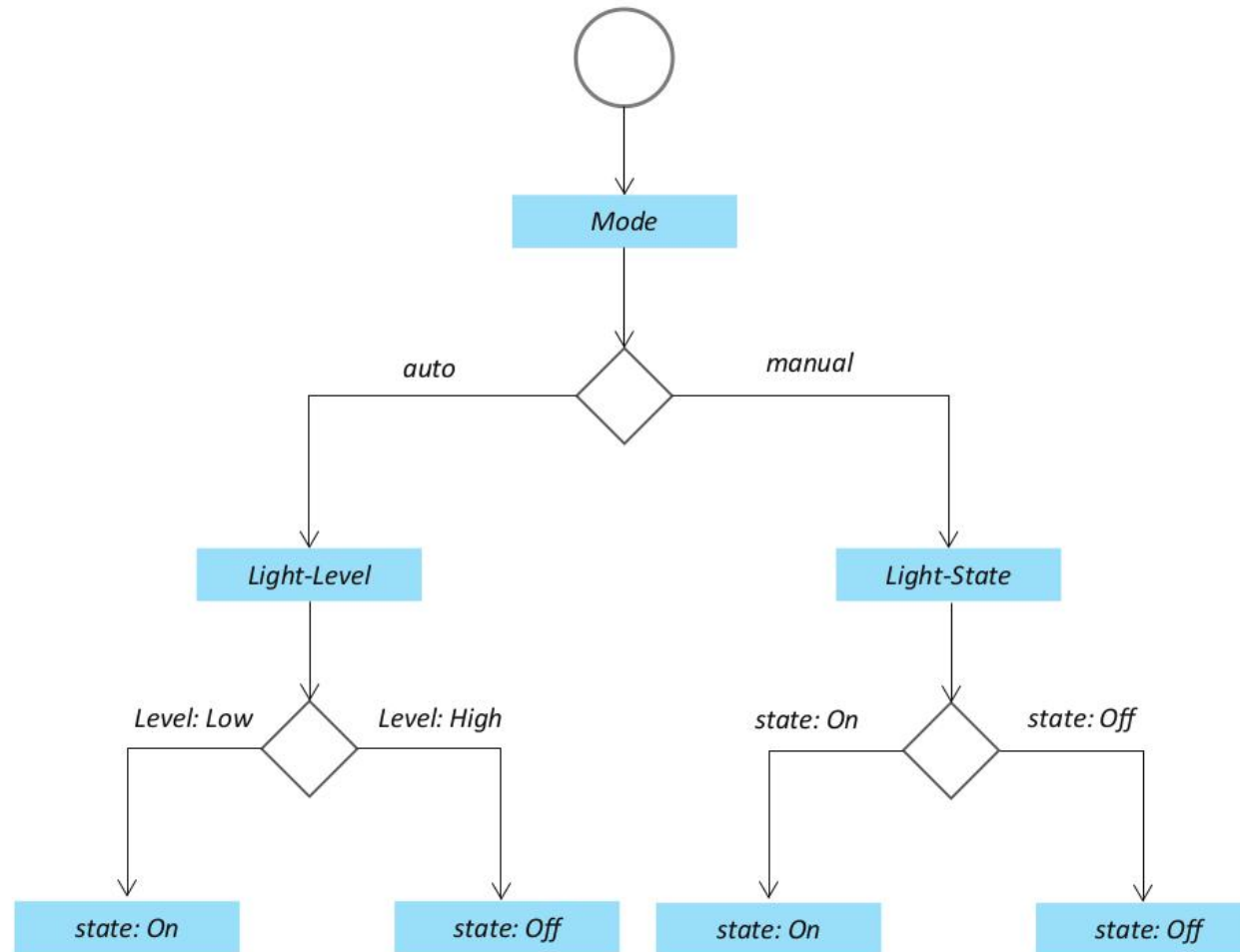
- The final step in the IoT design methodology is to develop the IoT application.

Home Automation Case Study

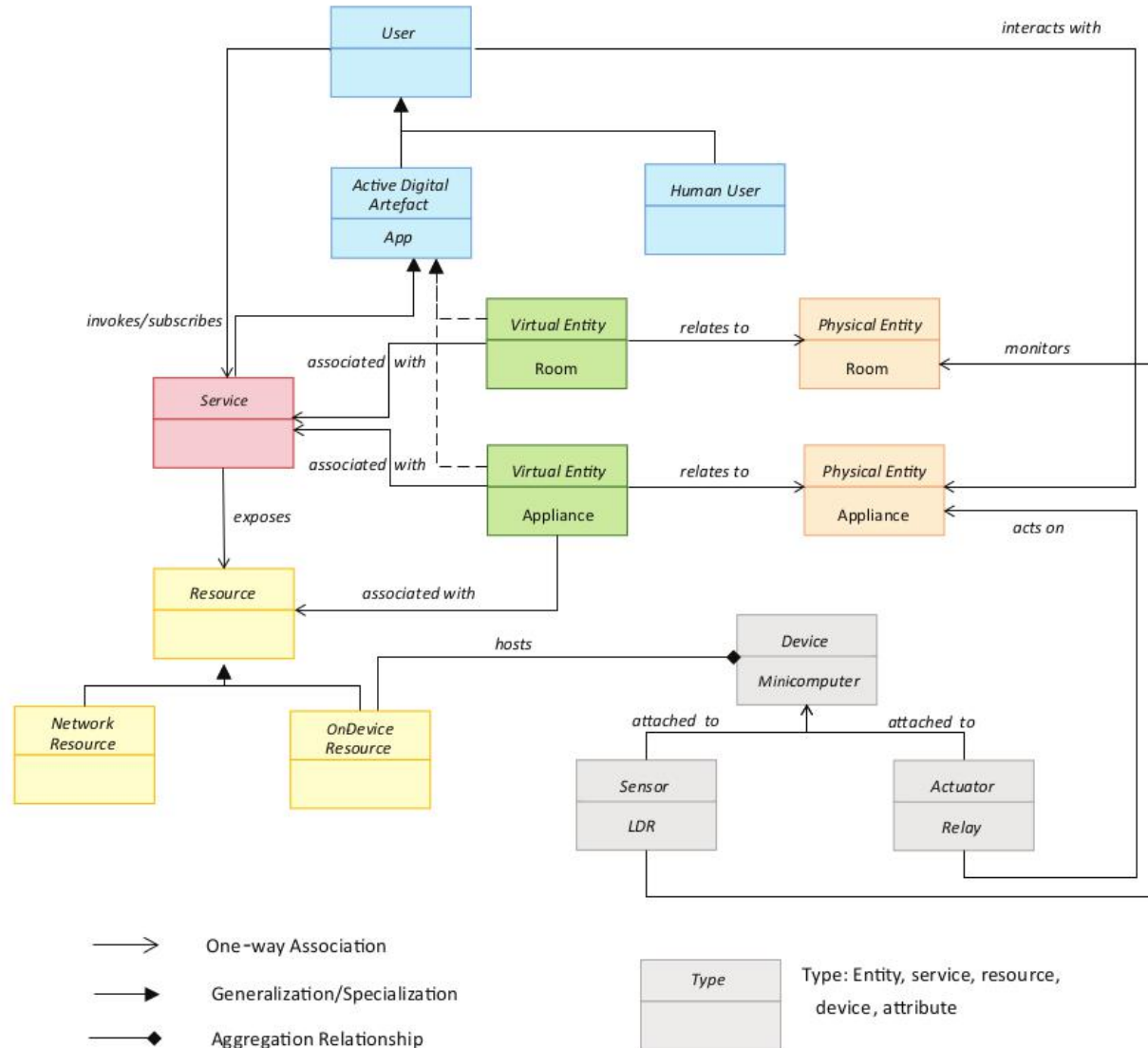
Step:1 - Purpose & Requirements

- Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:
 - Purpose : A home automation system that allows controlling of the lights in a home remotely using a web application.
 - Behavior : The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.
 - System Management Requirement : The system should provide remote monitoring and control functions.
 - Data Analysis Requirement : The system should perform local analysis of the data.
 - Application Deployment Requirement : The application should be deployed locally on the device, but should be accessible remotely.
 - Security Requirement : The system should have basic user authentication capability.

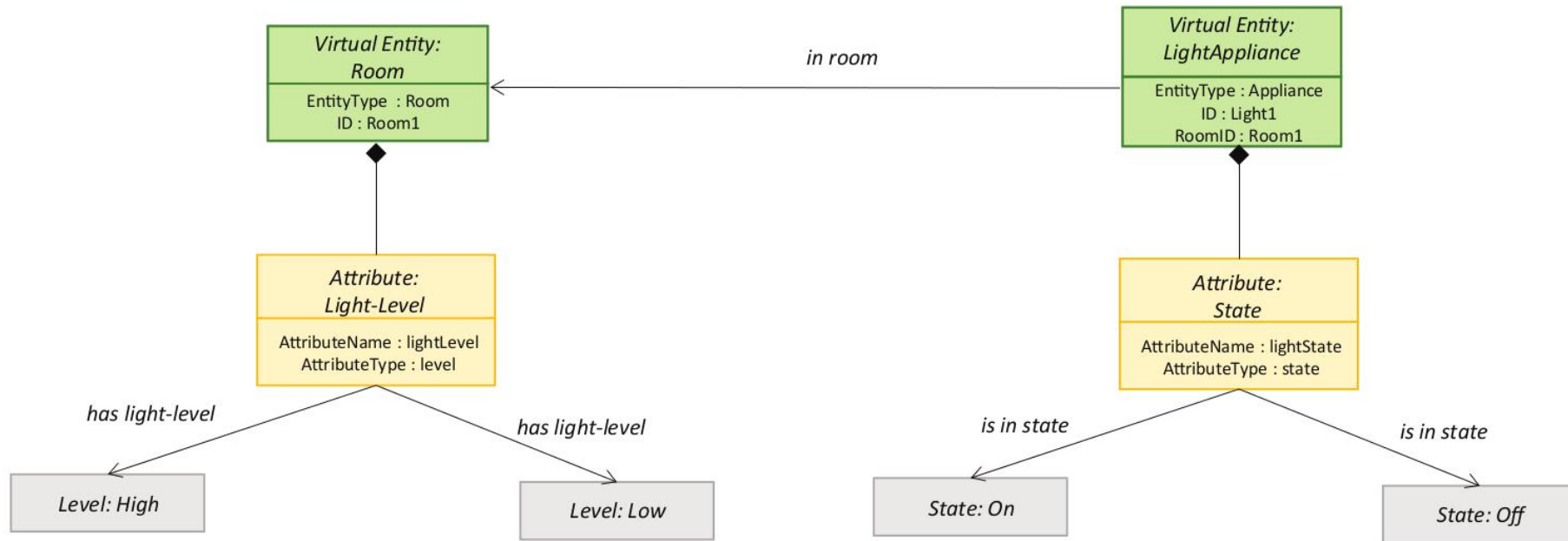
Step:2 - Process Specification



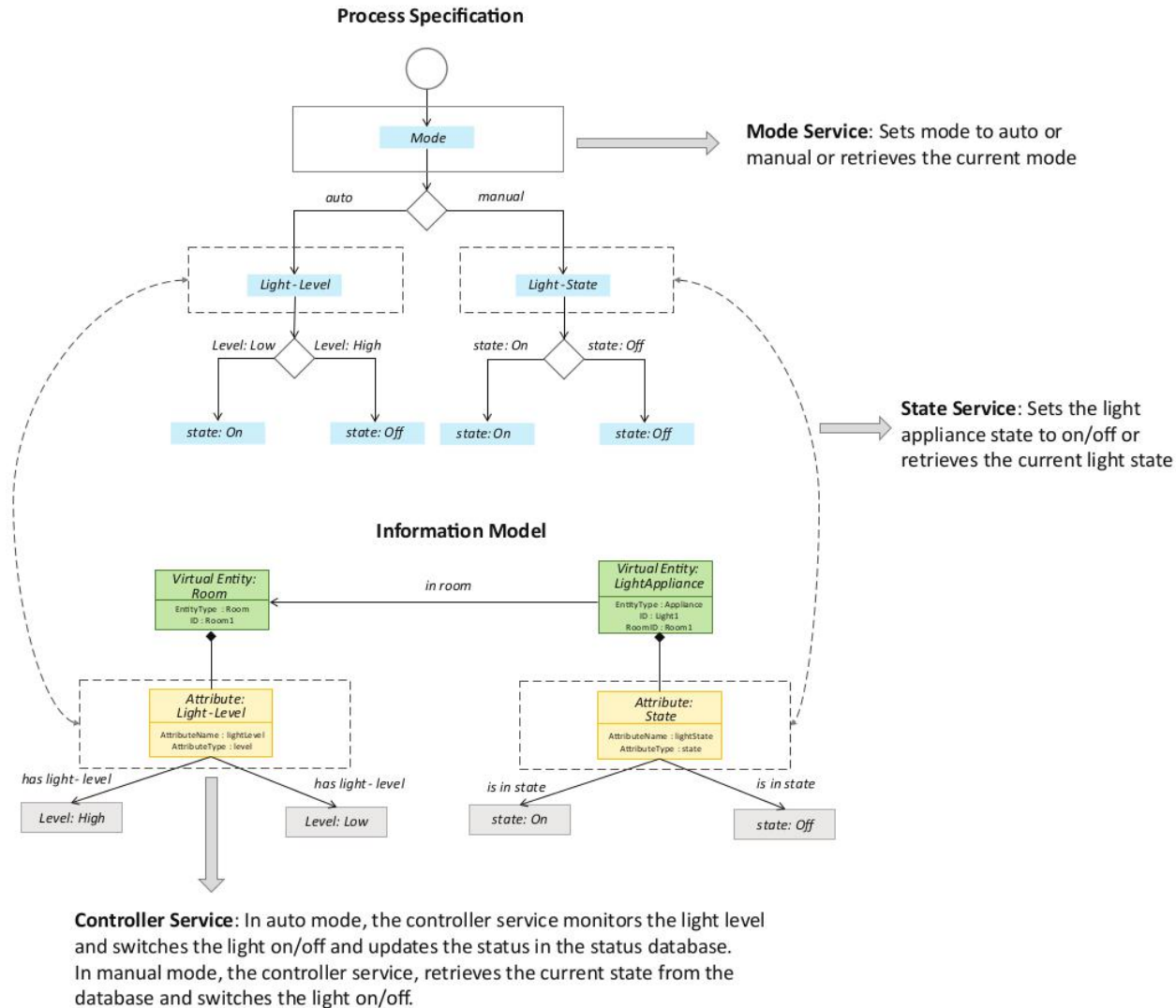
Step 3: Domain Model Specification



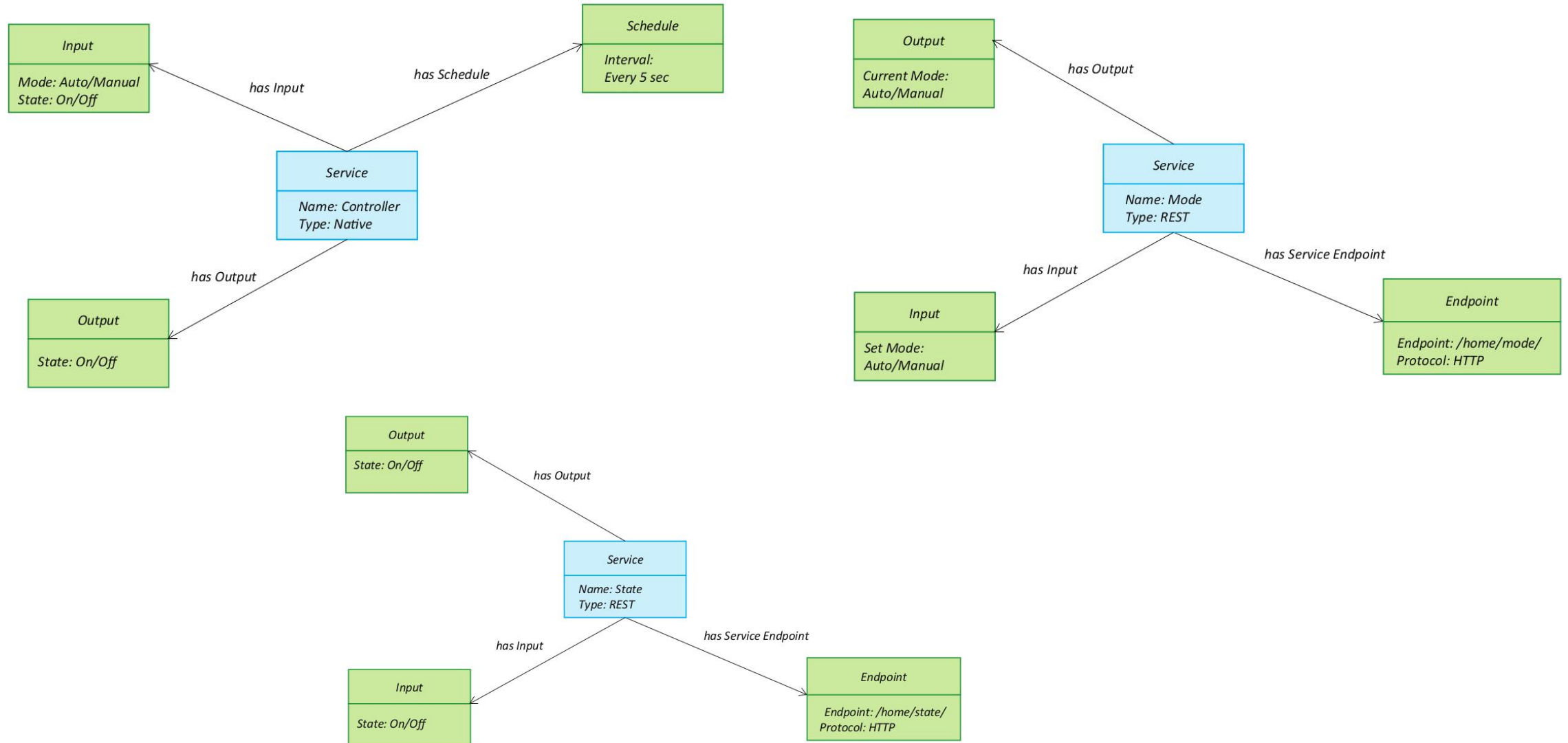
Step 4: Information Model Specification



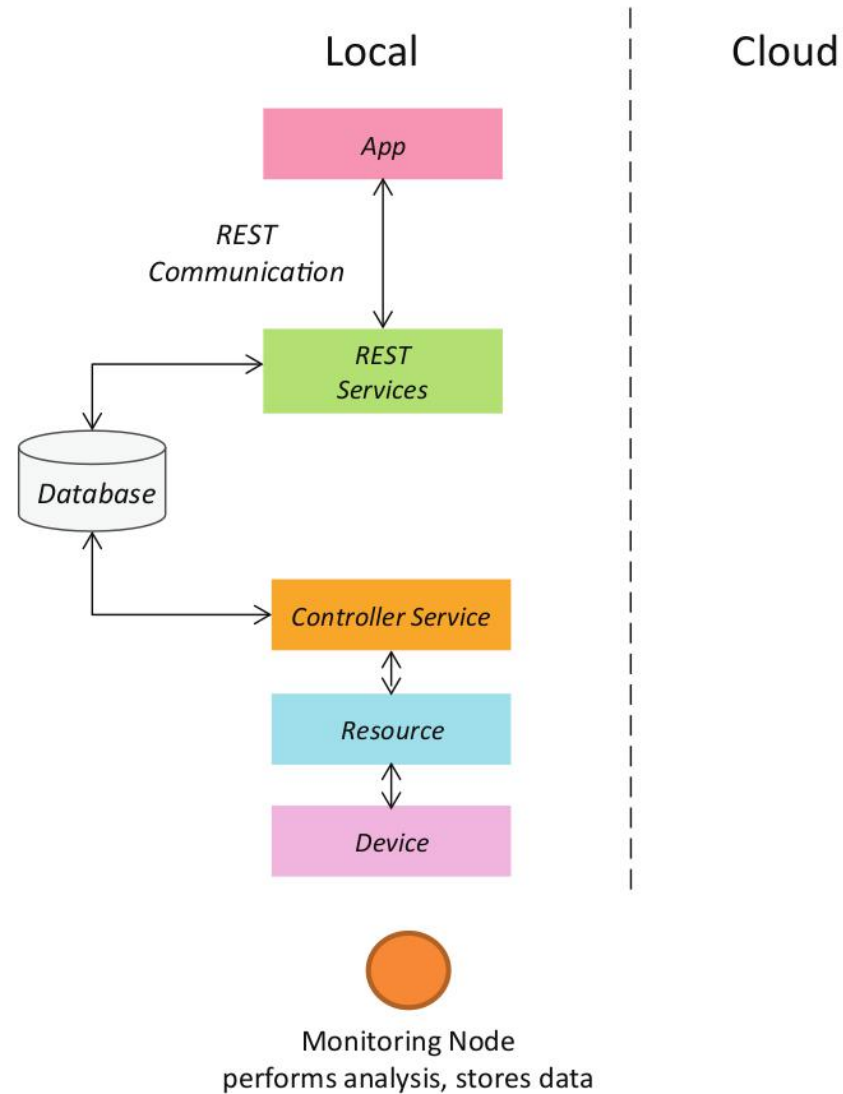
Step 5: Service Specifications



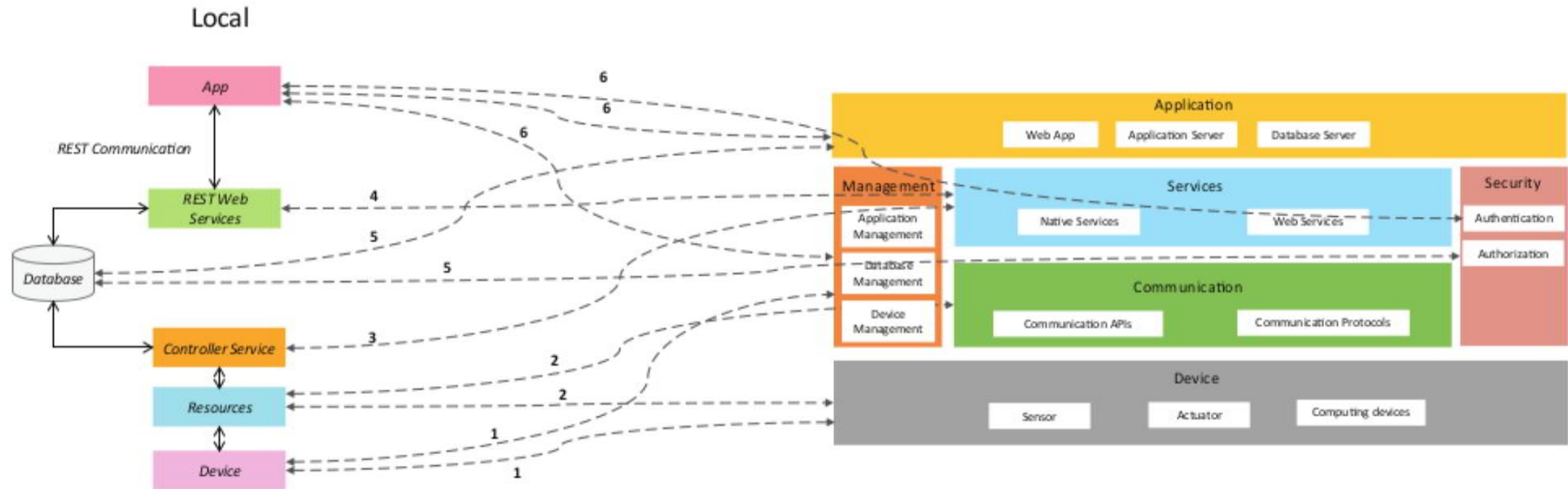
Step 5: Service Specifications



Step 6: IoT Level Specification



Step 7: Functional View Specification



1. IoT device maps to the Device FG (sensors, actuators devices, computing devices) and the Management FG (device management)

4. Web Services map to Services FG (web services)

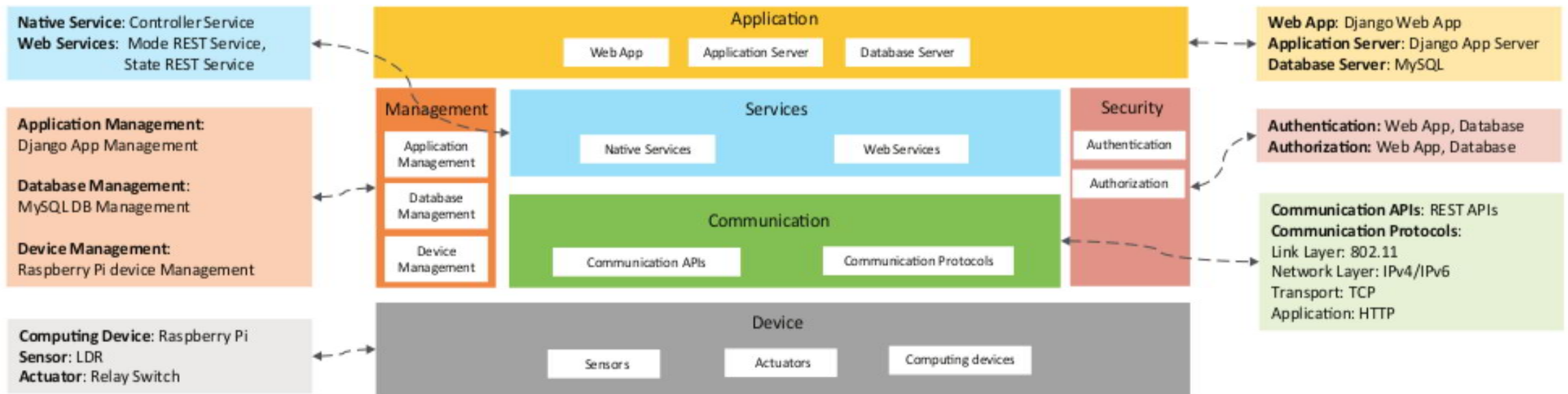
2. Resources map to the Device FG (on-device resource) and Communication FG (communication APIs and protocols)

5. Database maps to the Management FG (database management) and Security FG (database security)

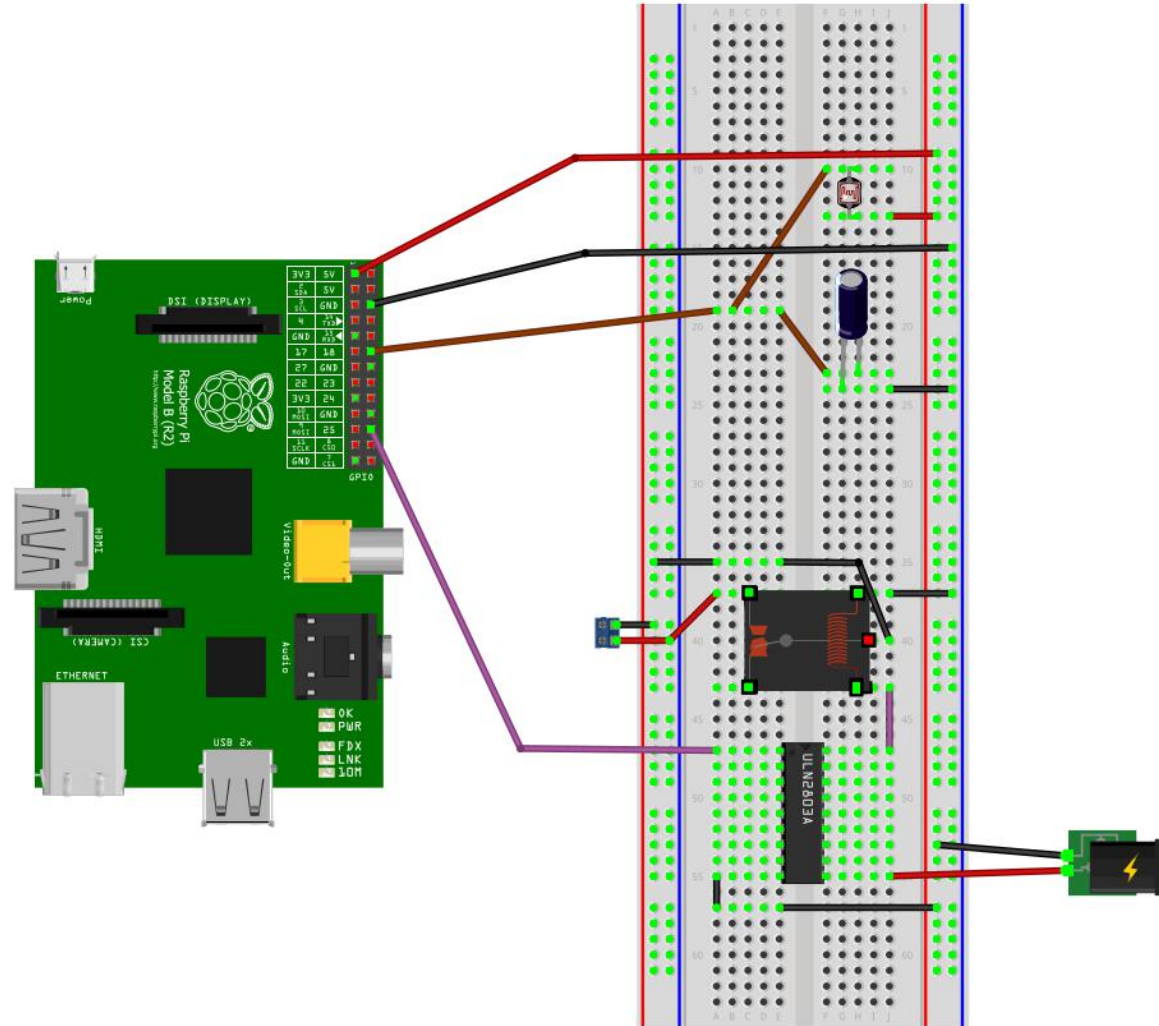
3. Controller service maps to the Services FG (native service). Web Services map to Services FG (web services)

6. Application maps to the Application FG (web application, application and database servers), Management FG (app management) and Security FG (app security)

Step 8: Operational View Specification

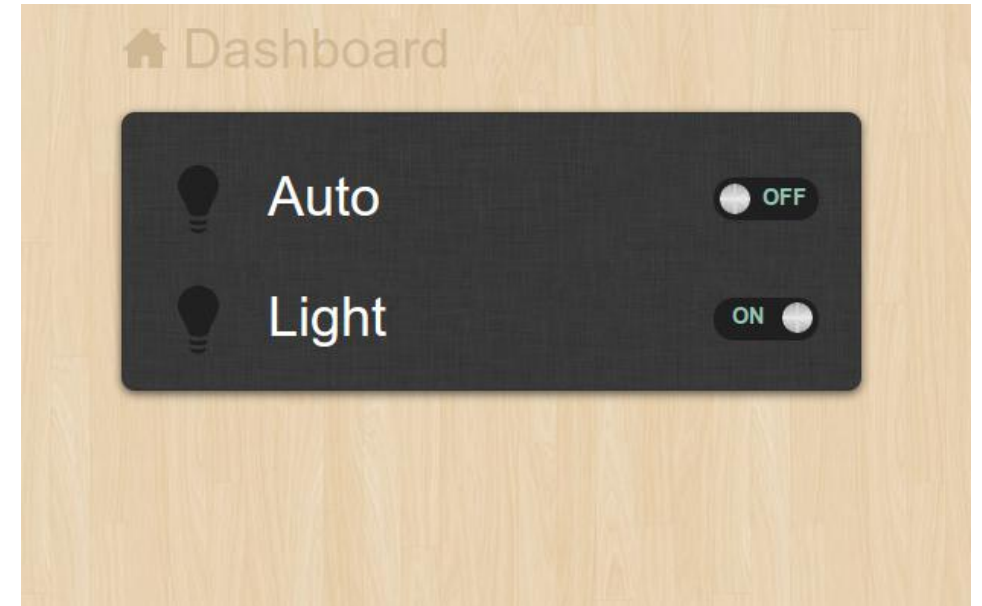


Step 9: Device & Component Integration



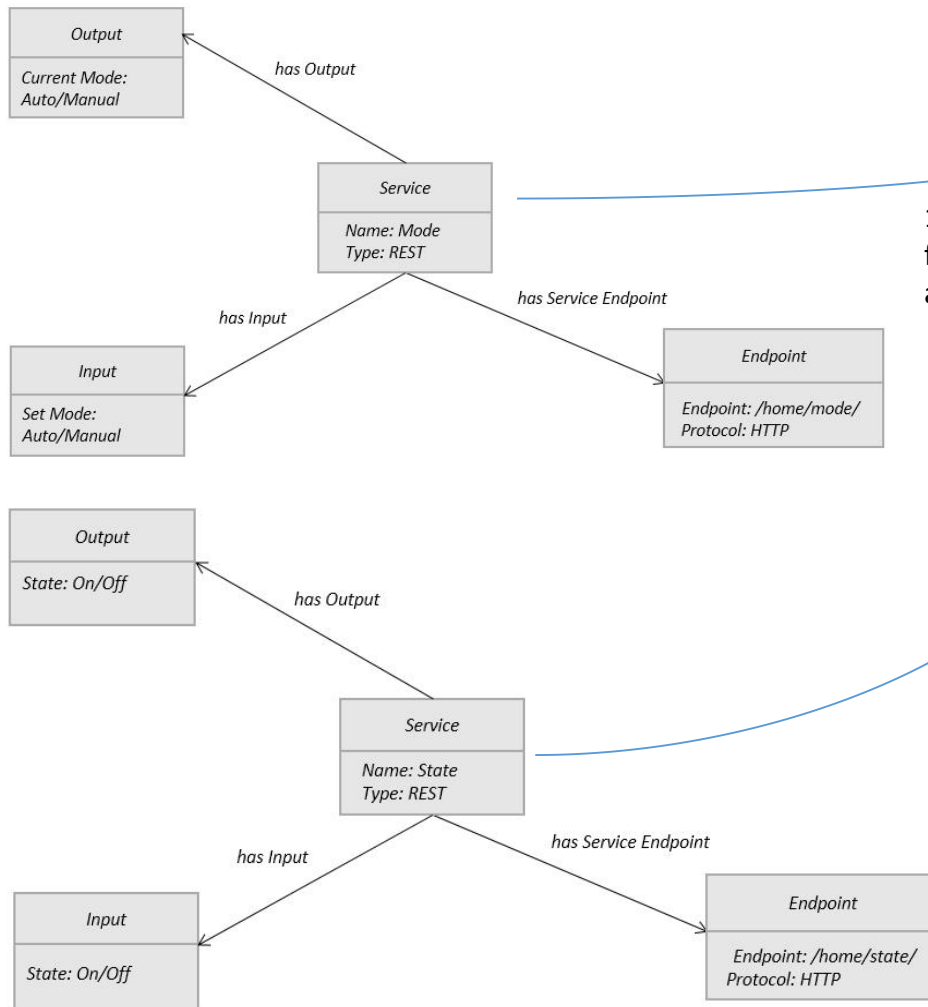
Step 10: Application Development

- Auto
 - Controls the light appliance automatically based on the lighting conditions in the room
- Light
 - When Auto mode is off, it is used for manually controlling the light appliance.
 - When Auto mode is on, it reflects the current state of the light appliance.



Implementation: RESTful Web Services

REST services implemented with Django REST Framework



1. Map services to models. Model fields store the states (on/off, auto/manual)

```
# Models – models.py
from django.db import models

class Mode(models.Model):
    name = models.CharField(max_length=50)

class State(models.Model):
    name = models.CharField(max_length=50)
```

2. Write Model serializers. Serializers allow complex data (such as model instances) to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

```
# Serializers – serializers.py
from myapp.models import Mode, State
from rest_framework import serializers

class ModeSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Mode
        fields = ('url', 'name')

class StateSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = State
        fields = ('url', 'name')
```

Implementation: RESTful Web Services


Models – models.py

```
from django.db import models

class Mode(models.Model):
    name = models.CharField(max_length=50)

class State(models.Model):
    name = models.CharField(max_length=50)
```

3. Write ViewSets for the Models which combine the logic for a set of related views in a single class.



Views – views.py

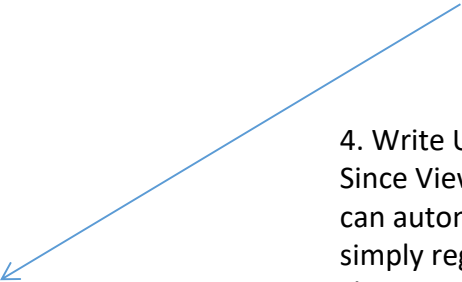
```
from myapp.models import Mode, State
from rest_framework import viewsets
from myapp.serializers import ModeSerializer, StateSerializer

class ModeViewSet(viewsets.ModelViewSet):
    queryset = Mode.objects.all()
    serializer_class = ModeSerializer

class StateViewSet(viewsets.ModelViewSet):
    queryset = State.objects.all()
    serializer_class = StateSerializer
```

URL Patterns – urls.py

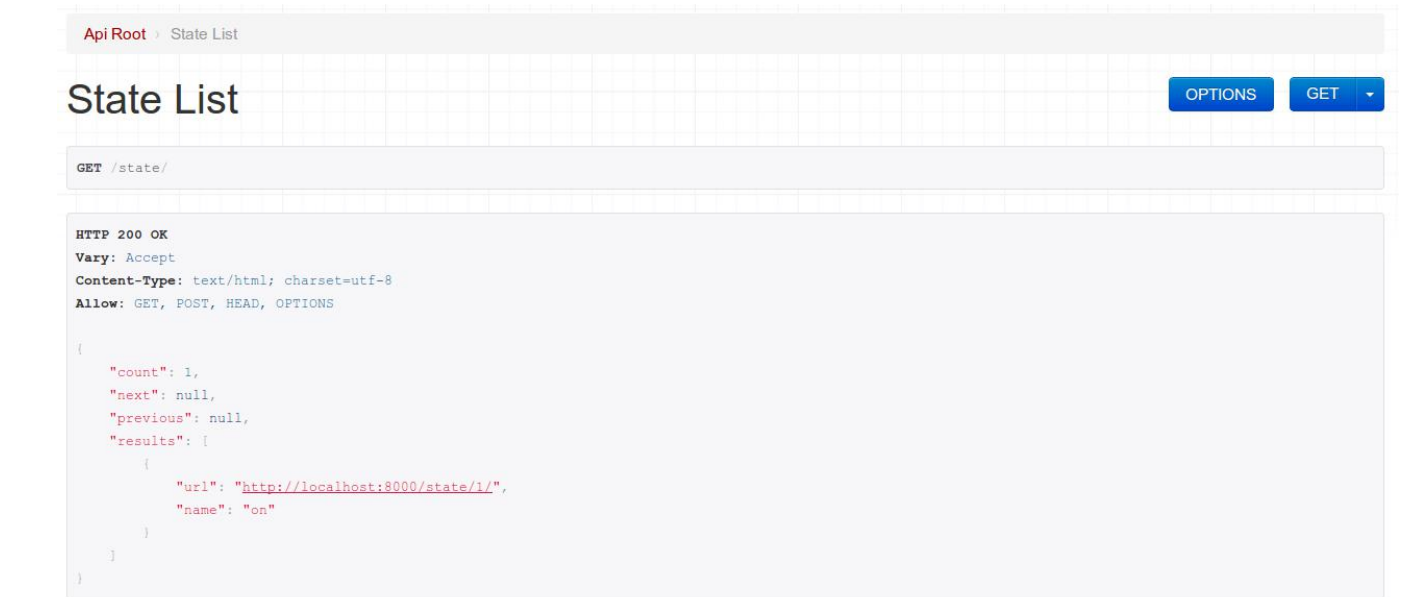
```
from django.conf.urls import patterns, include, url
from django.contrib import admin
from rest_framework import routers
from myapp import views
admin.autodiscover()
router = routers.DefaultRouter()
router.register(r'mode', views.ModeViewSet)
router.register(r'state', views.StateViewSet)
urlpatterns = patterns("",
    url(r'^$', include(router.urls)),
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^home/', 'myapp.views.home'),
)
```



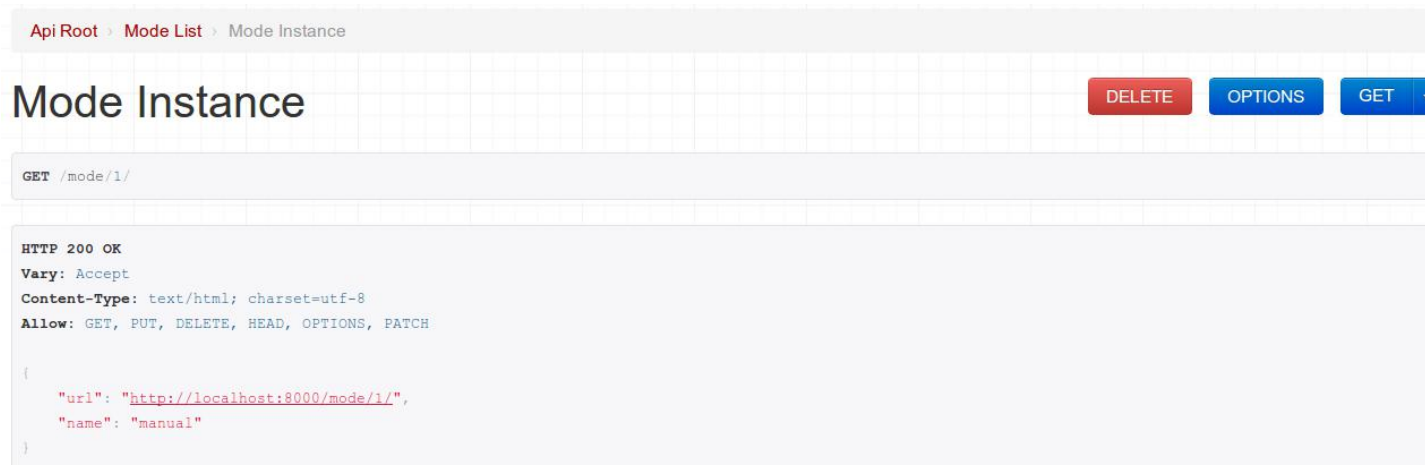
4. Write URL patterns for the services. Since ViewSets are used instead of views, we can automatically generate the URL conf by simply registering the viewsets with a router class. Routers automatically determining how the URLs for an application should be mapped to the logic that deals with handling incoming requests.

Implementation: RESTful Web Services

Screenshot of browsable
State REST API

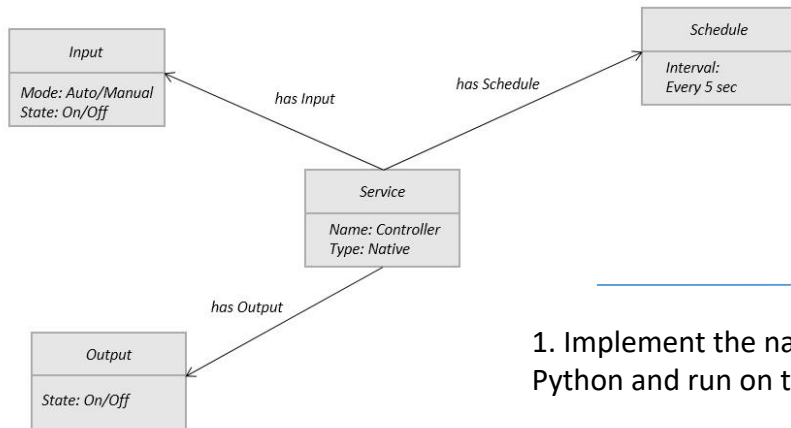


Screenshot of browsable
Mode REST API



Implementation: Controller Native Service

Native service deployed locally



1. Implement the native service in Python and run on the device

#Controller service

```
import RPi.GPIO as GPIO
import time
import sqlite3 as lite
import sys
```

```
con = lite.connect('database.sqlite')
cur = con.cursor()
```

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

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

```
def switchOnLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.HIGH)
```

```
def switchOffLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.LOW)
```

```
def runAutoMode():
    ldr_reading = readLdr(LDR_PIN)
    if ldr_reading < threshold:
        switchOnLight(LIGHT_PIN)
        setCurrentState('on')
    else:
        switchOffLight(LIGHT_PIN)
        setCurrentState('off')

def runManualMode():
    state = getCurrentState()
    if state=='on':
        switchOnLight(LIGHT_PIN)
        setCurrentState('on')
    elif state=='off':
        switchOffLight(LIGHT_PIN)
        setCurrentState('off')

def getCurrentMode():
    cur.execute('SELECT * FROM myapp_mode')
    data = cur.fetchone()      #(1, u'auto')
    return data[1]

def getCurrentState():
    cur.execute('SELECT * FROM myapp_state')
    data = cur.fetchone()      #(1, u'on')
    return data[1]

def setCurrentState(val):
    query='UPDATE myapp_state set name="'+val+'"'
    cur.execute(query)

while True:
    currentMode=getCurrentMode()
    if currentMode=='auto':
        runAutoMode()
    elif currentMode=='manual':
        runManualMode()
    time.sleep(5)
```

Implementation: Application

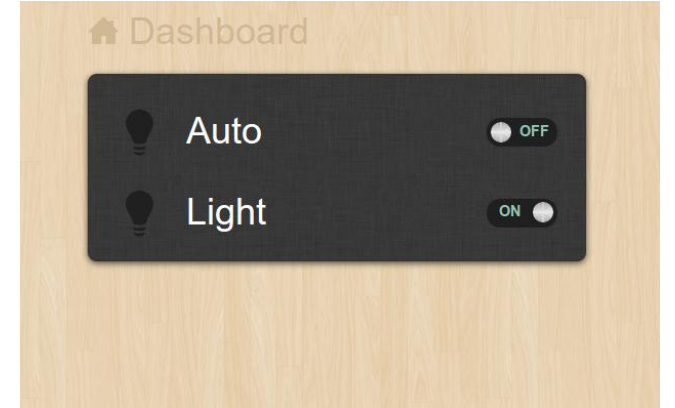
1. Implement Django Application View

```
# Views – views.py
def home(request):
    out=""
    if 'on' in request.POST:
        values = {"name": "on"}
        r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']
    if 'off' in request.POST:
        values = {"name": "off"}
        r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']
    if 'auto' in request.POST:
        values = {"name": "auto"}
        r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']
    if 'manual' in request.POST:
        values = {"name": "manual"}
        r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']

    r=requests.get('http://127.0.0.1:8000/mode/1/', auth=('username', 'password'))
    result=r.text
    output = json.loads(result)
    currentmode=output['name']
    r=requests.get('http://127.0.0.1:8000/state/1/', auth=('username', 'password'))
    result=r.text
    output = json.loads(result)
    currentstate=output['name']
    return render_to_response('lights.html',{r':out, 'currentmode':currentmode, 'currentstate':currentstate},
    context_instance=RequestContext(request))
```

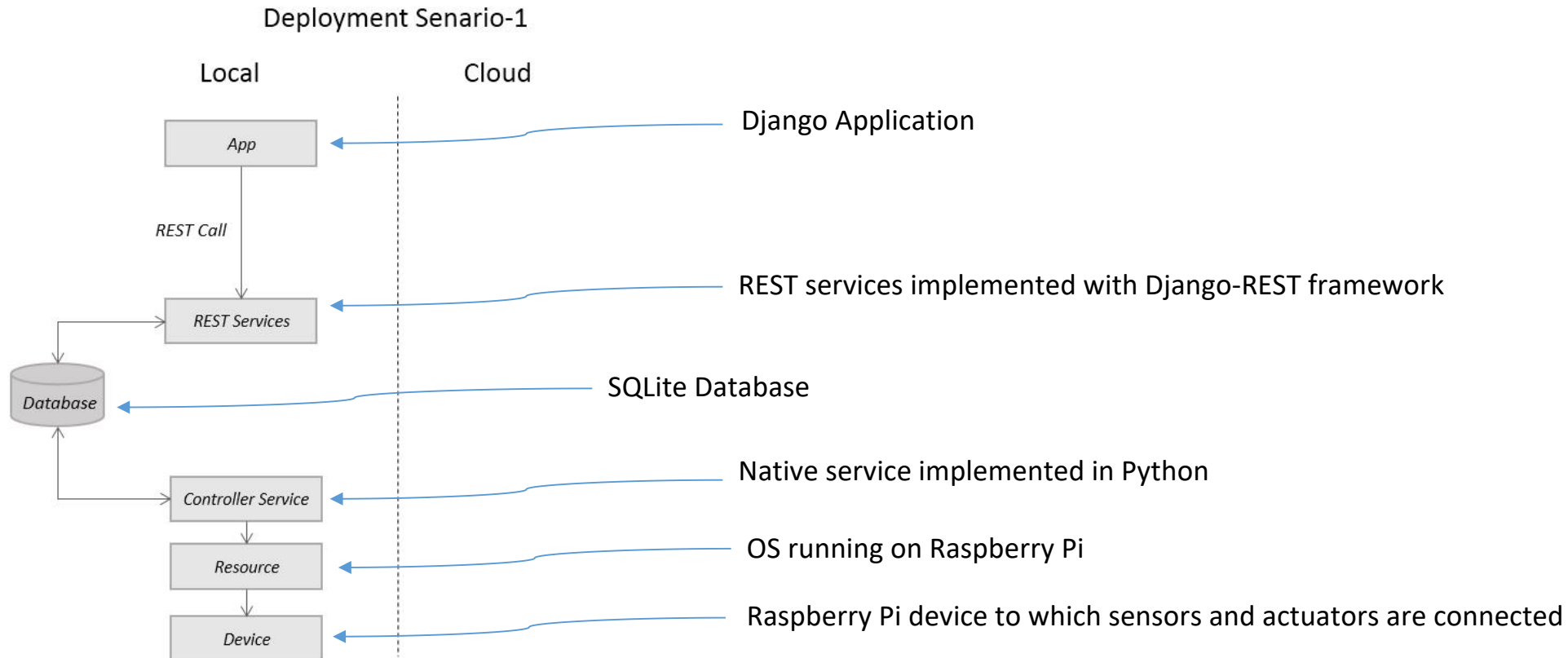
Implementation: Application

2. Implement Django Application Template

[illegible]

Finally - Integrate the System

- Setup the device
- Deploy and run the REST and Native services
- Deploy and run the Application
- Setup the database



Chapter 6

IoT Systems – Logical Design using Python

Outline

- Introduction to Python
- Installing Python
- Python Data Types & Data Structures
- Control Flow
- Functions
- Modules
- Packages
- File Input/Output
- Date/Time Operations
- Classes

Python

- Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.
- The main characteristics of Python are:
 - Multi-paradigm programming language
 - Python supports more than one programming paradigms including object-oriented programming and structured programming
 - Interpreted Language
 - Python is an interpreted language and does not require an explicit compilation step. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.
 - Interactive Language
 - Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

Python - Benefits

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

Python - Setup

- Windows

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

- Linux

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

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

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

Numbers

- Numbers
 - Number data type is used to store numeric values. Numbers are immutable data types, therefore changing the value of a number data type results in a newly allocated object.

#Integer

```
>>>a=5
>>>type(a)
<type 'int'>
```

#Floating Point

```
>>>b=2.5
>>>type(b)
<type 'float'>
```

#Long

```
>>>x=9898878787676L
>>>type(x)
<type 'long'>
```

#Complex

```
>>>y=2+5j
>>>y
(2+5j)
>>>type(y)
<type 'complex'>
>>>y.real
2
>>>y.imag
5
```

#Addition

```
>>>c=a+b
>>>c
7.5
>>>type(c)
<type 'float'>
```

#Subtraction

```
>>>d=a-b
>>>d
2.5
>>>type(d)
<type 'float'>
```

#Multiplication

```
>>>e=a*b
>>>e
12.5
>>>type(e)
<type 'float'>
```

#Division

```
>>>f=b/a
>>>f
0.5
>>>type(f)
<type float'>
```

#Power

```
>>>g=a**2
>>>g
25
```

Strings

- Strings
 - A string is simply a list of characters in order. There are no limits to the number of characters you can have in a string.

#Create string

```
>>>s="Hello World!"
>>>type(s)
<type 'str'>
```

#String concatenation

```
>>>t="This is sample program."
>>>r = s+t
>>>r
'Hello World!This is sample program.'
```

#Get length of string

```
>>>len(s)
12
```

#Convert string to integer

```
>>>x="100"
>>>type(s)
<type 'str'>
>>>y=int(x)
>>>y
100
```

#Print string

```
>>>print s
Hello World!
```

#Formatting output

```
>>>print "The string (The string (Hello World!)
has 12 characters"
```

#Convert to upper/lower case

```
>>>s.upper()
'HELLO WORLD!'
>>>s.lower()
'hello world!'
```

#Accessing sub-strings

```
>>>s[0]
'H'
>>>s[6:]
'World!'
>>>s[6:-1]
'World'
```

#strip: Returns a copy of the string with the
#leading and trailing characters removed.

```
>>>s.strip("!")
'Hello World'
```


Lists

- Lists

- List a compound data type used to group together other values. List items need not all have the same type. A list contains items separated by commas and enclosed within square brackets.

#Create List

```
>>>fruits=['apple','orange','banana','mango']
>>>type(fruits)
<type 'list'>
```

#Get Length of List

```
>>>len(fruits)
4
```

#Access List Elements

```
>>>fruits[1]
'orange'
>>>fruits[1:3]
['orange', 'banana']
>>>fruits[1:]
['orange', 'banana', 'mango']
```

#Appending an item to a list

```
>>>fruits.append('pear')
>>>fruits
['apple', 'orange', 'banana', 'mango', 'pear']
```

#Removing an item from a list

```
>>>fruits.remove('mango')
>>>fruits
['apple', 'orange', 'banana', 'pear']
```

#Inserting an item to a list

```
>>>fruits.insert(1,'mango')
>>>fruits
['apple', 'mango', 'orange', 'banana', 'pear']
```

#Combining lists

```
>>>vegetables=['potato','carrot','onion','beans','radish']
>>>vegetables
['potato', 'carrot', 'onion', 'beans', 'radish']

>>>eatables=fruits+vegetables
>>>eatables
['apple', 'mango', 'orange', 'banana', 'pear', 'potato', 'carrot', 'onion', 'beans', 'radish']
```

#Mixed data types in a list

```
>>>mixed=['data',5,100.1,8287398L]
>>>type(mixed)
<type 'list'>
>>>type(mixed[0])
<type 'str'>
>>>type(mixed[1])
<type 'int'>
>>>type(mixed[2])
<type 'float'>
>>>type(mixed[3])
<type 'long'>
```

#Change individual elements of a list

```
>>>mixed[0]=mixed[0]+" items"
>>>mixed[1]=mixed[1]+1
>>>mixed[2]=mixed[2]+0.05
>>>mixed
['data items', 6, 100.14999999999999, 8287398L]
```

#Lists can be nested

```
>>>nested=[fruits,vegetables]
>>>nested
[['apple', 'mango', 'orange', 'banana', 'pear'],
 ['potato', 'carrot', 'onion', 'beans', 'radish']]
```

Tuples

- Tuples

- A tuple is a sequence data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed within parentheses. Unlike lists, the elements of tuples cannot be changed, so tuples can be thought of as read-only lists.

#Create a Tuple

```
>>>fruits=("apple","mango","banana","pineapple")
>>>fruits
('apple', 'mango', 'banana', 'pineapple')
```

```
>>>type(fruits)
<type 'tuple'>
```

#Get length of tuple

```
>>>len(fruits)
4
```

#Get an element from a tuple

```
>>>fruits[0]
'apple'
>>>fruits[:2]
('apple', 'mango')
```

#Combining tuples

```
>>>vegetables=('potato','carrot','onion','radish')
>>>eatables=fruits+vegetables
>>>eatables
('apple', 'mango', 'banana', 'pineapple', 'potato', 'carrot', 'onion', 'radish')
```

Dictionaries

- Dictionaries

- Dictionary is a mapping data type or a kind of hash table that maps keys to values. Keys in a dictionary can be of any data type, though numbers and strings are commonly used for keys. Values in a dictionary can be any data type or object.

#Create a dictionary

```
>>>student={'name':'Mary','id':'8776','major':'CS'}
>>>student
{'major': 'CS', 'name': 'Mary', 'id': '8776'}
>>>type(student)
<type 'dict'>
```

#Get length of a dictionary

```
>>>len(student)
3
```

#Get the value of a key in dictionary

```
>>>student['name']
'Mary'
```

#Get all items in a dictionary

```
>>>student.items()
[('gender', 'female'), ('major', 'CS'), ('name', 'Mary'), ('id', '8776')]
```

#Get all keys in a dictionary

```
>>>student.keys()
['gender', 'major', 'name', 'id']
```

#Get all values in a dictionary

```
>>>student.values()
['female', 'CS', 'Mary', '8776']
```

#Add new key-value pair

```
>>>student['gender']='female'
>>>student
{'gende
r': 'female', 'major': 'CS', 'name': 'Mary', 'id': '8776'}
```

#A value in a dictionary can be another dictionary

```
>>>student1={'name':'David','id':'9876','major':'ECE'}
>>>students={'1': student,'2':student1}
>>>students
{'1':
{'gende
r': 'female', 'major': 'CS', 'name': 'Mary', 'id': '8776'}, '2':
{
major': 'ECE', 'name': 'David', 'id': '9876'}}
```

#Check if dictionary has a key

```
>>>student.has_key('name')
True
>>>student.has_key('grade')
False
```

Type Conversions

- Type conversion examples

#Convert to string

```
>>>a=10000
```

```
>>>str(a)
```

```
'10000'
```

#Convert to int

```
>>>b="2013"
```

```
>>>int(b)
```

```
2013
```

#Convert to float

```
>>>float(b)
```

```
2013.0
```

#Convert to long

```
>>>long(b)
```

```
2013L
```

#Convert to list

```
>>>s="aeiou"
```

```
>>>list(s)
```

```
['a', 'e', 'i', 'o', 'u']
```

#Convert to set

```
>>>x=['mango','apple','banana','mango','banana']
```

```
>>>set(x)
```

```
set(['mango', 'apple', 'banana'])
```

Control Flow – if statement

- The *if* statement in Python is similar to the *if* statement in other languages.

```
>>>a = 25**5
>>>if a>10000:
        print "More"
    else:
        print "Less"

More
```

```
>>>if a>10000:
    if a<1000000:
        print "Between 10k and 100k"
    else:
        print "More than 100k"
    elif a==10000:
        print "Equal to 10k"
    else:
        print "Less than 10k"

More than 100k
```

```
>>>s="Hello World"
>>>if "World" in s:
        s=s+"!"
    print s

Hello World!
```

```
>>>student={'name':'Mary','id':'8776'}
>>>if not student.has_key('major'):
        student['major']='CS'

>>>student
{'major': 'CS', 'name': 'Mary', 'id': '8776'}
```

Control Flow – for statement

- The *for* statement in Python iterates over items of any sequence (list, string, etc.) in the order in which they appear in the sequence.
- This behavior is different from the *for* statement in other languages such as C in which an initialization, incrementing and stopping criteria are provided.

#Looping over characters in a string

```
helloString = "Hello World"

for c in helloString:
    print c
```

#Looping over items in a list

```
fruits=['apple','orange','banana','mango']

i=0
for item in fruits:
    print "Fruit-%d: %s" % (i,item)
    i=i+1
```

#Looping over keys in a dictionary

```
student
=
'name': 'Mary', 'id': '8776', 'gender': 'female', 'major': 'CS'

for key in student:
    print "%s: %s" % (key,student[key])
```

Control Flow – while statement

- The while statement in Python executes the statements within the while loop as long as the while condition is true.

```
#Prints even numbers upto 100
```

```
>>> i = 0
```

```
>>> while i<=100:
```

```
    if i%2 == 0:
```

```
        print i
```

```
    i = i+1
```

Control Flow – range statement

- The range statement in Python generates a list of numbers in arithmetic progression.

#Generate a list of numbers from 0 – 9

```
>>>range (10)  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

#Generate a list of numbers from 10 - 100 with increments of 10

```
>>>range(10,110,10)  
[10, 20, 30, 40, 50, 60, 70, 80, 90,100]
```


Control Flow – break/continue statements

- The *break* and *continue* statements in Python are similar to the statements in C.

- Break

- Break statement breaks out of the for/while loop

#Break statement example

```
>>>y=1
>>>for x in range(4,256,4):
    y = y * x
    if y > 512:
        break
    print y

4
32
384
```

- Continue

- Continue statement continues with the next iteration.

#Continue statement example

```
>>>fruits=['apple','orange','banana','mango']
>>>for item in fruits:
    if item == "banana":
        continue
    else:
        print item

apple
orange
mango
```

Control Flow – pass statement

- The *pass* statement in Python is a null operation.
- The *pass* statement is used when a statement is required syntactically but you do not want any command or code to execute.

```
>fruits=['apple','orange','banana','mango']
>for item in fruits:
    if item == "banana":
        pass
    else:
        print item

apple
orange
mango
```

Functions

- A function is a block of code that takes information in (in the form of parameters), does some computation, and returns a new piece of information based on the parameter information.
- A function in Python is a block of code that begins with the keyword *def* followed by the function name and parentheses. The function parameters are enclosed within the parenthesis.
- The code block within a function begins after a colon that comes after the parenthesis enclosing the parameters.
- The first statement of the function body can optionally be a documentation string or docstring.

```
students = { '1': {'name': 'Bob', 'grade': 2.5},  
            '2': {'name': 'Mary', 'grade': 3.5},  
            '3': {'name': 'David', 'grade': 4.2},  
            '4': {'name': 'John', 'grade': 4.1},  
            '5': {'name': 'Alex', 'grade': 3.8}}
```

```
def averageGrade(students):  
    "This function computes the average grade"  
    sum = 0.0  
    for key in students:  
        sum = sum + students[key]['grade']  
    average = sum/len(students)  
    return average
```

```
avg = averageGrade(students)  
print "The average grade is: %0.2f" % (avg)
```

Functions - Default Arguments

- Functions can have default values of the parameters.
- If a function with default values is called with fewer parameters or without any parameter, the default values of the parameters are used

```
>>>def displayFruits(fruits=['apple','orange']):  
    print "There are %d fruits in the list" % (len(fruits))  
    for item in fruits:  
        print item  
  
#Using default arguments  
>>>displayFruits()  
apple  
orange  
  
>>>fruits = ['banana', 'pear', 'mango']  
>>>displayFruits(fruits)  
banana  
pear  
mango
```

Functions - Passing by Reference

- All parameters in the Python functions are passed by reference.
- If a parameter is changed within a function the change also reflected back in the calling function.

```
>>>def displayFruits(fruits):  
    print "There are %d fruits in the list" % (len(fruits))  
    for item in fruits:  
        print item  
    print "Adding one more fruit"  
    fruits.append('mango')  
  
>>>fruits = ['banana', 'pear', 'apple']  
>>>displayFruits(fruits)  
There are 3 fruits in the list  
banana  
pear  
apple  
  
#Adding one more fruit  
>>>print "There are %d fruits in the list" % (len(fruits))  
There are 4 fruits in the list
```

Functions - Keyword Arguments

- Functions can also be called using keyword arguments that identifies the arguments by the parameter name when the function is called.

```
>>>def
printStudentRecords(name,age=20,major='CS'):
    print "Name: " + name
    print "Age: " + str(age)
    print "Major: " + major
```

#This will give error as name is required argument

```
>>>printStudentRecords()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: printStudentRecords() takes at least 1
argument (0 given)
```

#Correct use

```
>>>printStudentRecords(name='Alex')
Name: Alex
Age: 20
Major: CS
```

```
>>>printStudentRecords(name='Bob',age=22,major='ECE')
Name: Bob
Age: 22
Major: ECE
```

```
>>>printStudentRecords(name='Alan',major='ECE')
Name: Alan
Age: 20
Major: ECE
```

#name is a formal argument.

****kwargs is a keyword argument that receives all arguments except the formal argument as a dictionary.**

```
>>>def student(name, **kwargs):
    print "Student Name: " + name
    for key in kwargs:
        print key + ':' + kwargs[key]
```

```
>>>student(name='Bob', age='20', major = 'CS')
Student Name: Bob
age: 20
major: CS
```

Functions - Variable Length Arguments

- Python functions can have variable length arguments. The variable length arguments are passed to as a tuple to the function with an argument prefixed with asterix (*)

```
>>>def student(name, *varargs):  
    print "Student Name: " + name  
    for item in varargs:  
        print item  
  
>>>student('Nav')  
Student Name: Nav  
  
>>>student('Amy', 'Age: 24')  
Student Name: Amy  
Age: 24  
  
>>>student('Bob', 'Age: 20', 'Major: CS')  
Student Name: Bob  
Age: 20  
Major: CS
```

Modules

- Python allows organizing the program code into different modules which improves the code readability and management.
- A module is a Python file that defines some functionality in the form of functions or classes.
- Modules can be imported using the import keyword.
- Modules to be imported must be present in the search path.

```
#student module - saved as student.py
def averageGrade(students):
    sum = 0.0
    for key in students:
        sum = sum + students[key]['grade']
    average = sum/len(students)
    return average

def printRecords(students):
    print "There are %d students" %(len(students))
    i=1
    for key in students:
        print "Student-%d: " % (i)
        print "Name: " + students[key]['name']
        print "Grade: " + str(students[key]['grade'])
        i = i+1
```

```
# Importing a specific function from a module
>>>from student import averageGrade
```

```
# Listing all names defines in a module
>>>dir(student)
```

#Using student module

```
>>>import student
>>>students = {'1': 'name': 'Bob', 'grade': 2.5,
'2': 'name': 'Mary', 'grade': 3.5,
'3': 'name': 'David', 'grade': 4.2,
'4': 'name': 'John', 'grade': 4.1,
'5': 'name': 'Alex', 'grade': 3.8}
```

```
>>>student.printRecords(students)
```

There are 5 students

Student-1:

Name: Bob

Grade: 2.5

Student-2:

Name: David

Grade: 4.2

Student-3:

Name: Mary

Grade: 3.5

Student-4:

Name: Alex

Grade: 3.8

Student-5:

Name: John

Grade: 4.1

```
>>>avg = student.averageGrade(students)
>>>print "The average grade is: %0.2f" % (avg)
3.62
```


Packages

- Python package is hierarchical file structure that consists of modules and subpackages.
- Packages allow better organization of modules related to a single application environment.

```
# skimage package listing

skimage/      Top level package
  __init__.py Treat directory as a package

  color/ color color subpackage
    __init__.py
    colorconv.py
    colorlabel.py
    rgb_colors.py

  draw/ draw draw subpackage
    __init__.py
    draw.py
    setup.py

  exposure/ exposure subpackage
    __init__.py
    _adapthist.py
    exposure.py

  feature/ feature subpackage
    __init__.py
    _brief.py
    _daisy.py

...
```

File Handling

- Python allows reading and writing to files using the file object.
- The `open(filename, mode)` function is used to get a file object.
- The mode can be read (r), write (w), append (a), read and write (r+ or w+), read-binary (rb), write-binary (wb), etc.
- After the file contents have been read the `close` function is called which closes the file object.

Example of reading an entire file

```
>>>fp = open('file.txt','r')
>>>content = fp.read()
>>>print content
This is a test file.
>>>fp.close()
```

Example of reading line by line

```
>>>fp = open('file1.txt','r')
>>>print "Line-1: " + fp.readline()
Line-1: Python supports more than one programming paradigms.
>>>print "Line-2: " + fp.readline()
Line-2: Python is an interpreted language.
>>>fp.close()
```

Example of reading lines in a loop

```
>>>fp = open('file1.txt','r')
>>>lines = fp.readlines()
>>>for line in lines:
    print line
```

Python supports more than one programming paradigms.
Python is an interpreted language.

File Handling

Example of reading a certain number of bytes

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>fp.close()
```

Example of seeking to a certain position

```
>>>fp = open('file.txt','r')
>>>fp.seek(10,0)
>>>content = fp.read(10)
>>>print content
ports more
>>>fp.close()
```

Example of getting the current position of read

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>currentpos = fp.tell()
>>>print currentpos
<built-in method tell of file object at 0x0000000002391390>
>>>fp.close()
```

Example of writing to a file

```
>>>fo = open('file1.txt','w')
>>>content='This is an example of writing to a file in
Python.'
>>>fo.write(content)
>>>fo.close()
```

Date/Time Operations

- Python provides several functions for date and time access and conversions.
- The datetime module allows manipulating date and time in several ways.
- The time module in Python provides various time-related functions.

Examples of manipulating with date

```
>>>from datetime import date

>>>now = date.today()
>>>print "Date: " + now.strftime("%m-%d-%y")
Date: 07-24-13

>>>print "Day of Week: " + now.strftime("%A")
Day of Week: Wednesday

>>>print "Month: " + now.strftime("%B")
Month: July

>>>then = date(2013, 6, 7)
>>>timediff = now - then
>>>timediff.days
47
```

Examples of manipulating with time

```
>>>import time
>>>nowtime = time.time()
>>>time.localtime(nowtime)
time.struct_time(tm_year=2013, tm_mon=7, tm_mday=24, tm_ec=51, tm_wday=2, tm_yday=205,
tm_isdst=0)

>>>time.asctime(time.localtime(nowtime))
'Wed Jul 24 16:14:51 2013'

>>>time.strftime("The date is %d-%m-%y. Today is a %A. It is %H hours, %M minutes and %S seconds now.")
'The date is 24-07-13. Today is a Wednesday. It is 16 hours, 15 minutes and 14 seconds now.'
```

Classes

- Python is an Object-Oriented Programming (OOP) language. Python provides all the standard features of Object Oriented Programming such as classes, class variables, class methods, inheritance, function overloading, and operator overloading.
- Class
 - A class is simply a representation of a type of object and user-defined prototype for an object that is composed of three things: a name, attributes, and operations/methods.
- Instance/Object
 - Object is an instance of the data structure defined by a class.
- Inheritance
 - Inheritance is the process of forming a new class from an existing class or base class.
- Function overloading
 - Function overloading is a form of polymorphism that allows a function to have different meanings, depending on its context.
- Operator overloading
 - Operator overloading is a form of polymorphism that allows assignment of more than one function to a particular operator.
- Function overriding
 - Function overriding allows a child class to provide a specific implementation of a function that is already provided by the base class. Child class implementation of the overridden function has the same name, parameters and return type as the function in the base class.

Class Example

- The variable *studentCount* is a class variable that is shared by all instances of the class *Student* and is accessed by *Student.studentCount*.
- The variables *name*, *id* and *grades* are instance variables which are specific to each instance of the class.
- There is a special method by the name `__init__()` which is the class constructor.
- The class constructor initializes a new instance when it is created. The function `__del__()` is the class destructor

```
# Examples of a class
class Student:
    studentCount = 0

    def __init__(self, name, id):
        print "Constructor called"
        self.name = name
        self.id = id
        Student.studentCount = Student.studentCount + 1
        self.grades={}

    def __del__(self):
        print "Destructor called"

    def getStudentCount(self):
        return Student.studentCount

    def addGrade(self, key, value):
        self.grades[key]=value
    def getGrade(self, key):
        return self.grades[key]

    def printGrades(self):
        for key in self.grades:
            print key + ": " + self.grades[key]
```

```
>>>s = Student('Steve','98928')
Constructor called

>>>s.addGrade('Math','90')
>>>s.addGrade('Physics','85')
>>>s.printGrades()
Physics: 85
Math: 90

>>>mathgrade = s.getGrade('Math')
>>>print mathgrade
90

>>>count = s.getStudentCount()
>>>print count
1

>>>del s
Destructor called
```

Class Inheritance

- In this example Shape is the base class and Circle is the derived class. The class Circle inherits the attributes of the Shape class.
- The child class Circle overrides the methods and attributes of the base class (eg. draw() function defined in the base class Shape is overridden in child class Circle).

Examples of class inheritance

class Shape:

```
def __init__(self):
    print "Base class constructor"
    self.color = 'Green'
    self.lineWeight = 10.0

def draw(self):
    print "Draw - to be implemented"
    def setColor(self, c):
        self.color = c
    def getColor(self):
        return self.color

def setLineWeight(self,lwt):
    self.lineWeight = lwt

def getLineWeight(self):
    return self.lineWeight
```

class Circle(Shape):

```
def __init__(self, c,r):
    print "Child class constructor"
    self.center = c
    self.radius = r
    self.color = 'Green'
    self.lineWeight = 10.0
    self.__label = 'Hidden circle label'

def setCenter(self,c):
    self.center = c
def getCenter(self):
    return self.center

def setRadius(self,r):
    self.radius = r

def getRadius(self):
    return self.radius

def draw(self):
    print "Draw Circle (overridden function)"
```

class Point:

```
def __init__(self, x, y):
    self.xCoordinate = x
    self.yCoordinate = y

def setXCoordinate(self,x):
    self.xCoordinate = x

def getXCoordinate(self):
    return self.xCoordinate

def setYCoordinate(self,y):
    self.yCoordinate = y

def getYCoordinate(self):
    return self.yCoordinate
```

```
>>>p = Point(2,4)
>>>circ = Circle(p,7)
Child class constructor
>>>circ.getColor()
'Green'
>>>circ.setColor('Red')
>>>circ.getColor()
'Red'
>>>circ.getLineWeight()
10.0
>>>circ.getCenter().getXCoordinate()
2
>>>circ.getCenter().getYCoordinate()
4
>>>circ.draw()
Draw Circle (overridden function)
>>>circ.radius
7
```

Further Reading

- Code Academy Python Tutorial, <http://www.codecademy.com/tracks/python>
- Google's Python Class, <https://developers.google.com/edu/python/>
- Python Quick Reference Cheat Sheet, <http://www.addedbytes.com/cheat-sheets/python-cheat-sheet/>
- PyCharm Python IDE, <http://www.jetbrains.com/pycharm/>

Chapter 7

IoT Physical Devices & Endpoints

Outline

- Basic building blocks of an IoT Device
- Exemplary Device: Raspberry Pi
- Raspberry Pi interfaces
- Programming Raspberry Pi with Python
- Other IoT devices

What is an IoT Device

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

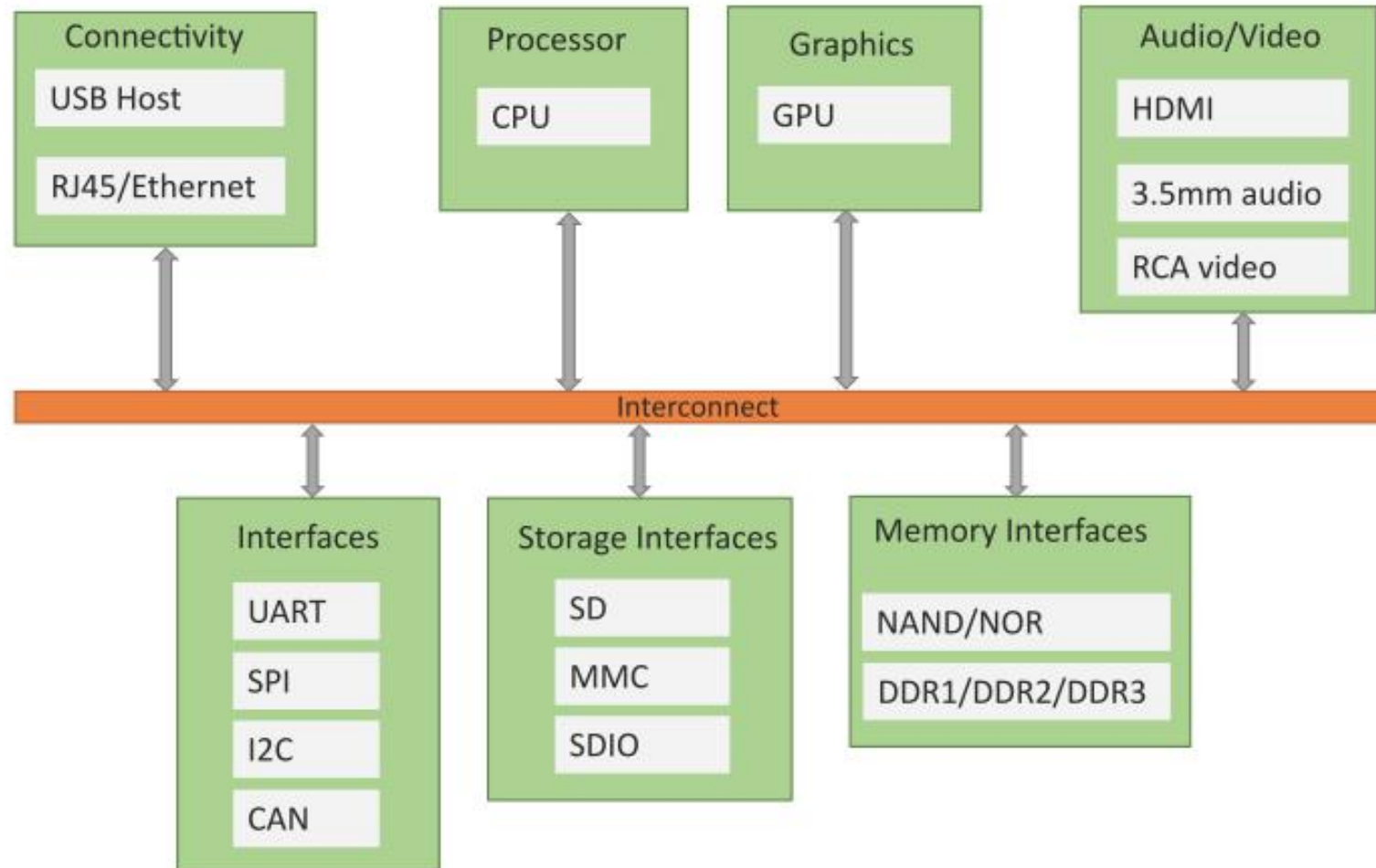
IoT Device Examples

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

Basic building blocks of an IoT Device

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

Block diagram of an IoT Device



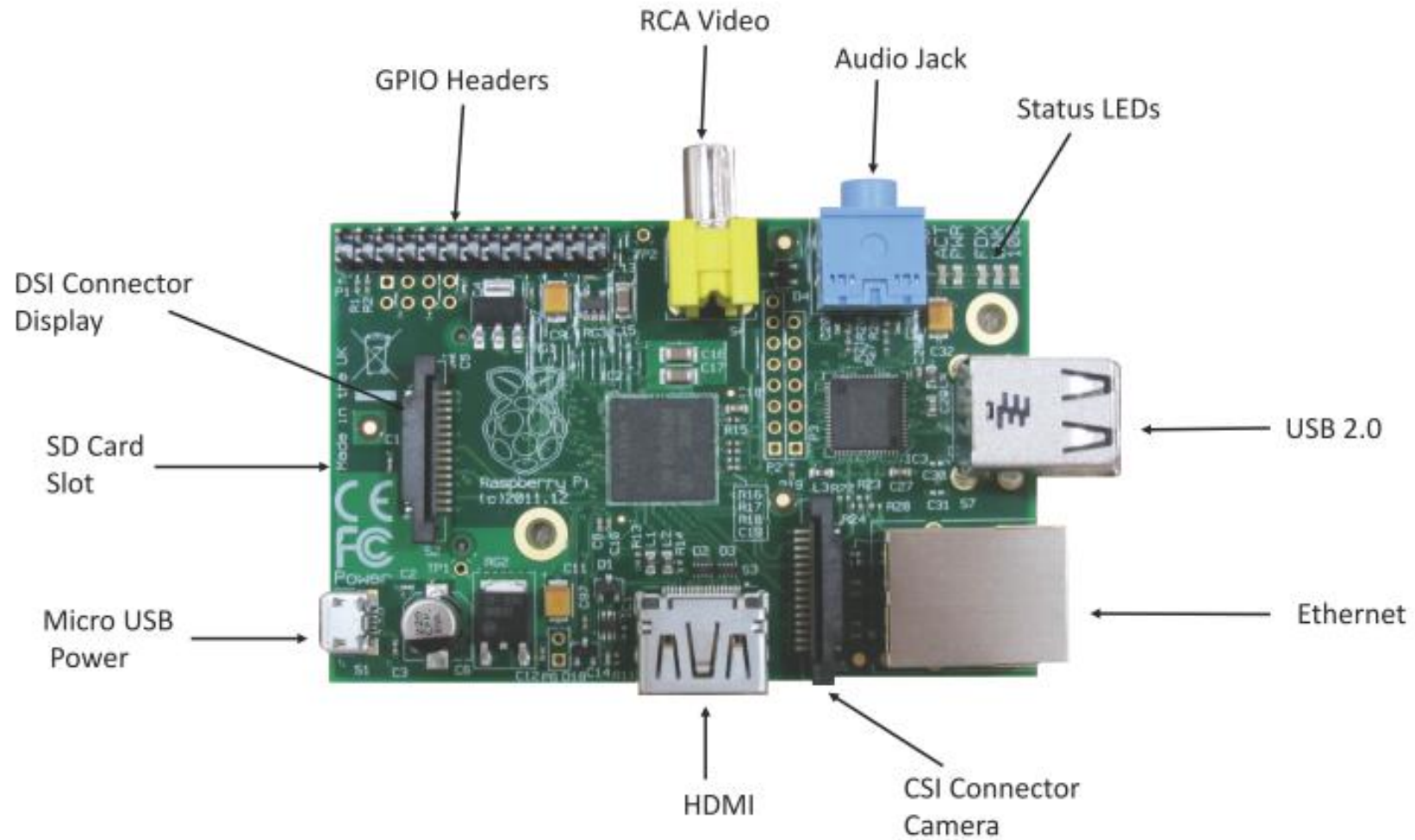
Exemplary Device: Raspberry Pi

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

Exemplary Device: Raspberry Pi

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

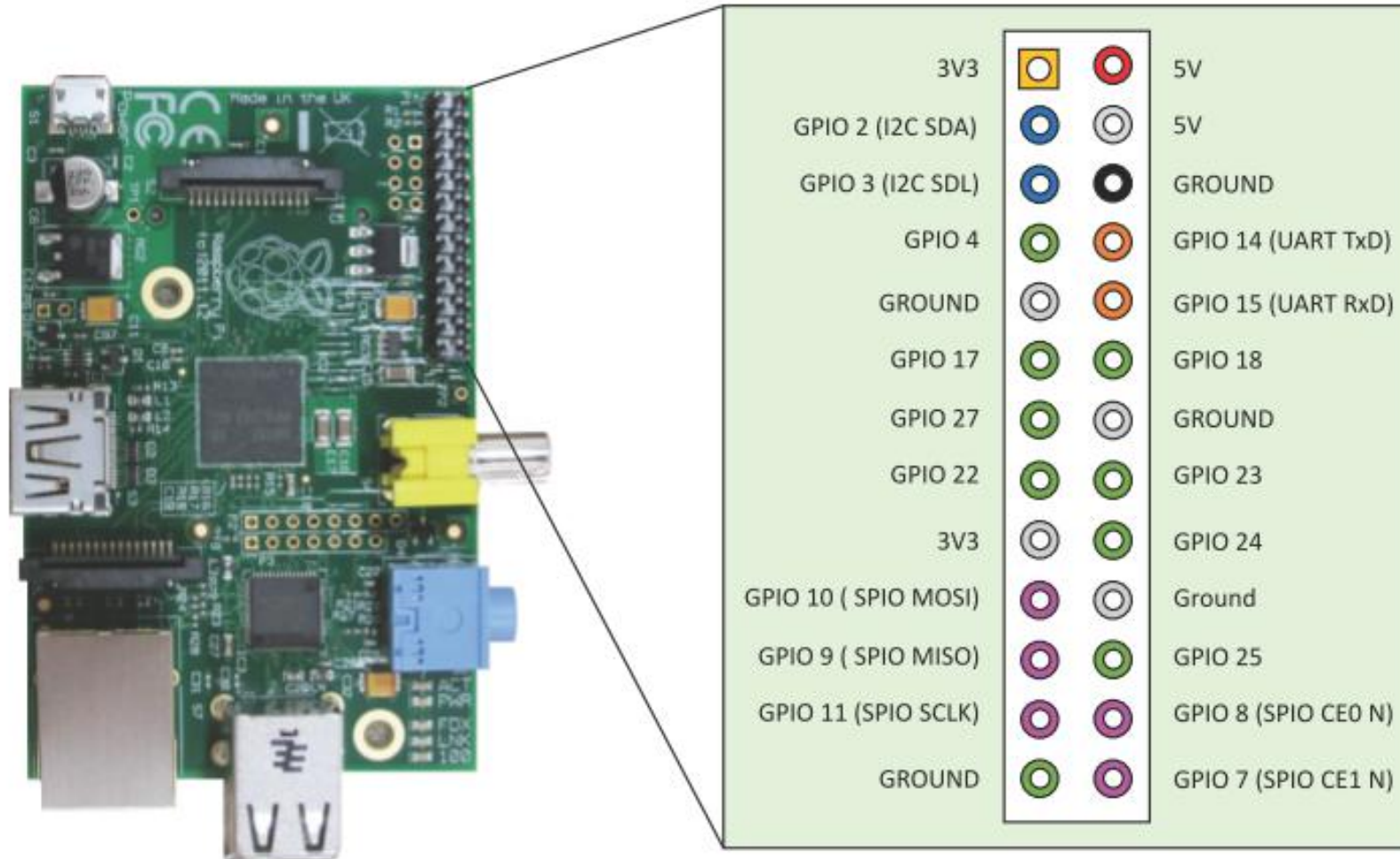
Raspberry Pi



Linux on Raspberry Pi

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

Raspberry Pi GPIO



Raspberry Pi Interfaces

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

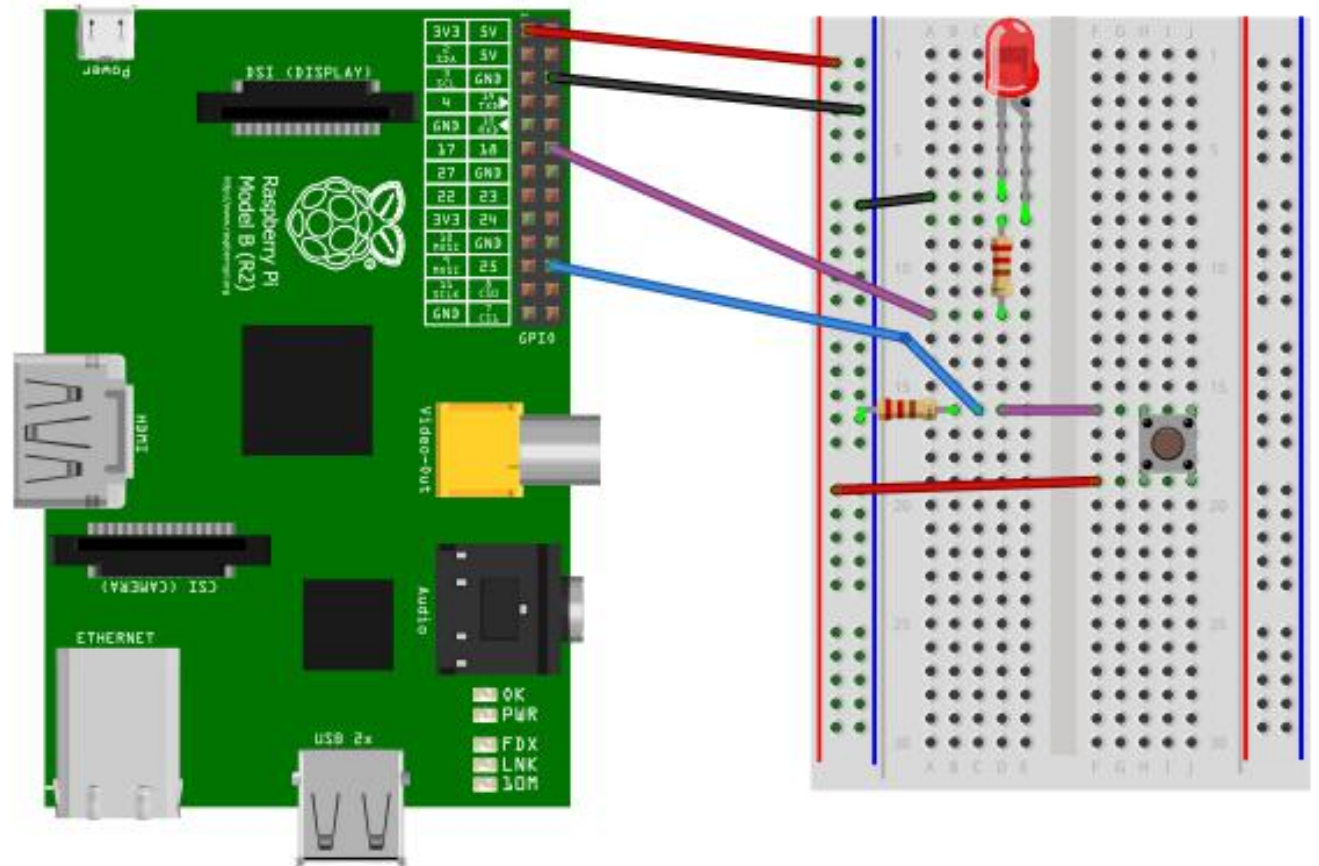
Raspberry Pi Example: Interfacing LED and switch with Raspberry Pi

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
```

```
#Switch Pin
GPIO.setup(25, GPIO.IN)
#LED Pin
GPIO.setup(18, GPIO.OUT)
state=False
```

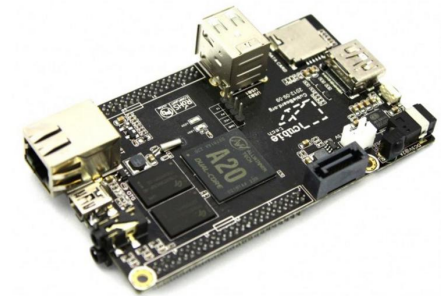
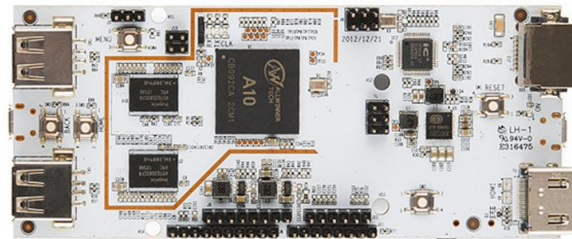
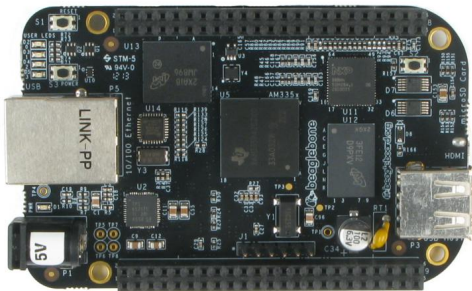
```
def toggleLED(pin):
    state = not state
    GPIO.output(pin, state)
```

```
while True:
    try:
        if (GPIO.input(25) == True):
            toggleLED(pin)
            sleep(.01)
    except KeyboardInterrupt:
        exit()
```



Other Devices

- pcDuino
- BeagleBone Black
- Cubieboard



Chapter 8

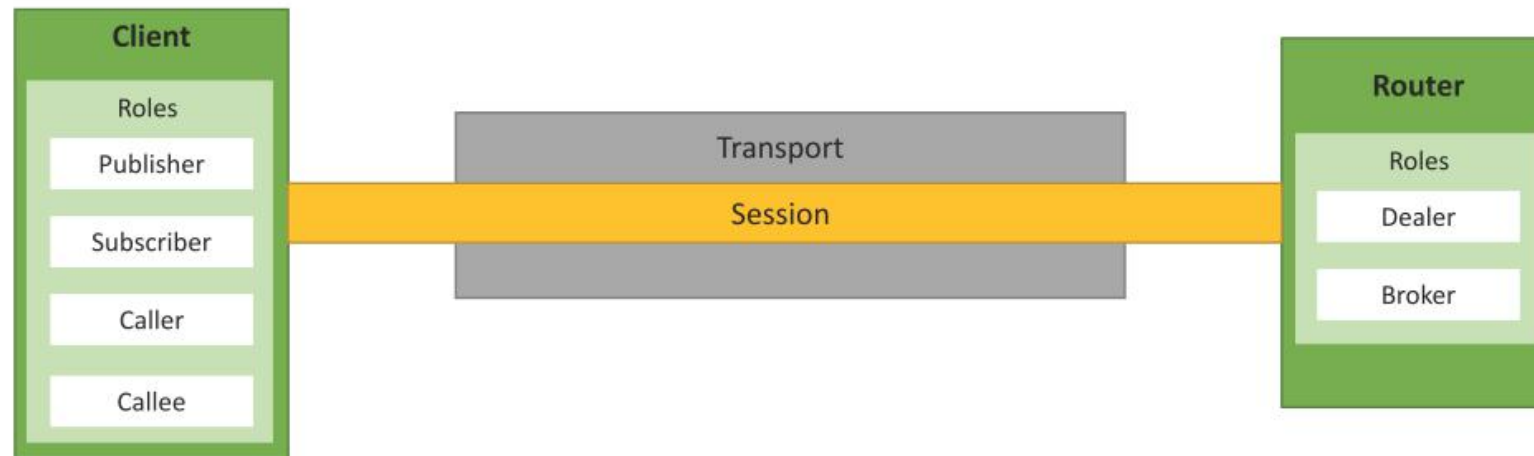
IoT Physical Servers & Cloud Offerings

Outline

- WAMP - AutoBahn for IoT
- Python for Amazon Web Services
- Python for MapReduce
- Python Packages of Interest
- Python Web Application Framework - Django
- Development with Django

WAMP for IoT

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



WAMP - Concepts

- 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 the topic.

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).

Amazon EC2 – Python Example

- Boto is a Python package that provides interfaces to Amazon Web Services (AWS)
 - 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.

#Python program for launching an EC2 instance

```
import boto.ec2
from time import sleep
ACCESS_KEY=<enter access key>
SECRET_KEY=<enter secret key>

REGION="us-east-1"
AMI_ID = "ami-d0f89fb9"
EC2_KEY_HANDLE = <enter key handle>
INSTANCE_TYPE="t1.micro"
SECGROUP_HANDLE="default"

conn = boto.ec2.connect_to_region(REGION, aws_access_key_id=ACCESS_KEY,
                                  aws_secret_access_key=SECRET_KEY)

reservation = conn.run_instances(image_id=AMI_ID, key_name=EC2_KEY_HANDLE,
                                  instance_type=INSTANCE_TYPE,
                                  security_groups = [ SECGROUP_HANDLE, ] )
```

Amazon AutoScaling – Python Example

- AutoScaling Service
 - 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.

```
#Python program for creating an AutoScaling group (code excerpt)
import boto.ec2.autoscale
:
print "Connecting to Autoscaling Service"
conn = boto.ec2.autoscale.connect_to_region(REGION,
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY)

print "Creating launch configuration"

lc = LaunchConfiguration(name='My-Launch-Config-2',
    image_id=AMI_ID,
    key_name=EC2_KEY_HANDLE,
    instance_type=INSTANCE_TYPE,
    security_groups = [ SECGROUP_HANDLE, ])
conn.create_launch_configuration(lc)

print "Creating auto-scaling group"

ag = AutoScalingGroup(group_name='My-Group',
    availability_zones=['us-east-1b'],
    launch_config=lc, min_size=1, max_size=2,
    connection=conn)
conn.create_auto_scaling_group(ag)
```

Amazon AutoScaling – Python Example

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

#Creating auto-scaling policies

```
scale_up_policy = ScalingPolicy(name='scale_up',
                                adjustment_type='ChangeInCapacity',
                                as_name='My-Group',
                                scaling_adjustment=1,
                                cooldown=180)

scale_down_policy = ScalingPolicy(name='scale_down',
                                   adjustment_type='ChangeInCapacity',
                                   as_name='My-Group',
                                   scaling_adjustment=-1,
                                   cooldown=180)

conn.create_scaling_policy(scale_up_policy)
conn.create_scaling_policy(scale_down_policy)
```

Amazon AutoScaling – Python Example

- 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%.

#Connecting to CloudWatch

```
cloudwatch = boto.ec2.cloudwatch.connect_to_region(REGION,
                                                    aws_access_key_id=ACCESS_KEY,
                                                    aws_secret_access_key=SECRET_KEY)
alarm_dimensions = {"AutoScalingGroupName": 'My-Group'}
```

#Creating scale-up alarm

```
scale_up_alarm = MetricAlarm(
    name='scale_up_on_cpu', namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',
    comparison='>', threshold='70',
    period='60', evaluation_periods=2,
    alarm_actions=[scale_up_policy.policy_arn],
    dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_up_alarm)
```

#Creating scale-down alarm

```
scale_down_alarm = MetricAlarm(
    name='scale_down_on_cpu', namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',
    comparison='<', threshold='40',
    period='60', evaluation_periods=2,
    alarm_actions=[scale_down_policy.policy_arn],
    dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_down_alarm)
```

Amazon S3 – Python Example

- 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.

```
# Python program for uploading a file to an S3 bucket
import boto.s3

conn = boto.connect_s3(aws_access_key_id='<enter>',
    aws_secret_access_key='<enter>')

def percent_cb(complete, total):
    print('.')

def upload_to_s3_bucket_path(bucketname, path, filename):
    mybucket = conn.get_bucket(bucketname)
    fullkeyname=os.path.join(path,filename)
    key = mybucket.new_key(fullkeyname)
    key.set_contents_from_filename(filename, cb=percent_cb, num_cb=10)
```

Amazon RDS – Python Example

- 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.

#Python program for launching an RDS instance (excerpt)

```
import boto.rds
```

```
ACCESS_KEY="<enter>"
SECRET_KEY="<enter>"
REGION="us-east-1"
INSTANCE_TYPE="db.t1.micro"
ID = "MySQL-db-instance-3"
USERNAME = 'root'
PASSWORD = 'password'
DB_PORT = 3306
DB_SIZE = 5
DB_ENGINE = 'MySQL5.1'
DB_NAME = 'mytestdb'
SECGROUP_HANDLE="default"
```

#Connecting to RDS

```
conn = boto.rds.connect_to_region(REGION,
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY)
```

#Creating an RDS instance

```
db = conn.create_dbinstance(ID, DB_SIZE, INSTANCE_TYPE,
    USERNAME, PASSWORD, port=DB_PORT, engine=DB_ENGINE,
    db_name=DB_NAME, security_groups = [ SECGROUP_HANDLE, ] )
```


Amazon DynamoDB – Python Example

- 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.

Python program for creating a DynamoDB table (excerpt)

```
import boto.dynamodb
```

```
ACCESS_KEY="<enter>"
```

```
SECRET_KEY="<enter>"
```

```
REGION="us-east-1"
```

#Connecting to DynamoDB

```
conn = boto.dynamodb.connect_to_region(REGION,  
    aws_access_key_id=ACCESS_KEY,  
    aws_secret_access_key=SECRET_KEY)
```

```
table_schema = conn.create_schema(  
    hash_key_name='msgid',  
    hash_key_proto_value=str,  
    range_key_name='date',  
    range_key_proto_value=str  
)
```

#Creating table with schema

```
table = conn.create_table(  
    name='my-test-table',  
    schema=table_schema,  
    read_units=1,  
    write_units=1
```

Python for MapReduce

- The example shows inverted index mapper program.
- The map function reads the data from the standard input (stdin) and splits the tab-limited data into document-ID and contents of the document.
- The map function emits key-value pairs where key is each word in the document and value is the document-ID.

#Inverted Index Mapper in Python

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    doc_id, content = line.split("\t")
    words = content.split()
    for word in words:
        print '%s%s' % (word, doc_id)
```

Python for MapReduce

- The example shows inverted index reducer program.
- The key-value pairs emitted by the map phase are shuffled to the reducers and grouped by the key.
- The reducer reads the key-value pairs grouped by the same key from the standard input (stdin) and creates a list of document-IDs in which the word occurs.
- The output of reducer contains key value pairs where key is a unique word and value is the list of document-IDs in which the word occurs.

#Inverted Index Reducer in Python

```
#!/usr/bin/env python
import sys
current_word = None
current_docids = []
word = None

for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, doc_id = line.split(' ')
    if current_word == word:
        current_docids.append(doc_id)
    else:
        if current_word:
            print '%s%s' % (current_word, current_docids)
            current_docids = []
        current_docids.append(doc_id)
        current_word = word
```

Python Packages of Interest

- JSON
 - JavaScript Object Notation (JSON) is an easy to read and write data-interchange format. JSON is used as an alternative to XML and is easy for machines to parse and generate. JSON is built on two structures - a collection of name-value pairs (e.g. a Python dictionary) and ordered lists of values (e.g.. a Python list).
- XML
 - XML (Extensible Markup Language) is a data format for structured document interchange. The Python minidom library provides a minimal implementation of the Document Object Model interface and has an API similar to that in other languages.
- HTTPLib & URLLib
 - HTTPLib2 and URLLib2 are Python libraries used in network/internet programming
- SMTPLib
 - Simple Mail Transfer Protocol (SMTP) is a protocol which handles sending email and routing e-mail between mail servers. The Python smtpplib module provides an SMTP client session object that can be used to send email.
- NumPy
 - NumPy is a package for scientific computing in Python. NumPy provides support for large multi-dimensional arrays and matrices
- Scikit-learn
 - Scikit-learn is an open source machine learning library for Python that provides implementations of various machine learning algorithms for classification, clustering, regression and dimension reduction problems.

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 Architecture

- 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.

Further Reading

- boto, <http://boto.readthedocs.org/en/latest/>
- Python JSON package, <http://docs.python.org/library/json.html>
- Python socket package, <http://docs.python.org/2/library/socket.html>
- Python email package, <http://docs.python.org/2/library/email>
- Python HTTPLib, <http://code.google.com/p/httplib2/>
- Python URLLib, <http://docs.python.org/2/howto/urllib2.html>
- Python SMTPLib, <http://docs.python.org/2/library/smtplib.html>
- NumPy, <http://www.numpy.org/>
- Scikit-learn, <http://scikit-learn.org/stable/>
- Django, <https://docs.djangoproject.com/en/1.5/>
- Google App Engine, <https://developers.google.com/appengine/>
- Google Cloud Storage, <https://developers.google.com/storage/>
- Google BigQuery, <https://developers.google.com/bigquery/>
- Google Cloud Datastore, <http://developers.google.com/datastore/>
- Google Cloud SQL, <https://developers.google.com/cloud-sql/>
- AFINN, http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010
- Tweepy Package, <https://github.com/tweepy/tweepy>

Chapter 10

Data Analytics for IoT

Outline

- Overview of Hadoop ecosystem
- MapReduce architecture
- MapReduce job execution flow
- MapReduce schedulers

Hadoop Ecosystem

- Apache Hadoop is an open source framework for distributed batch processing of big data.

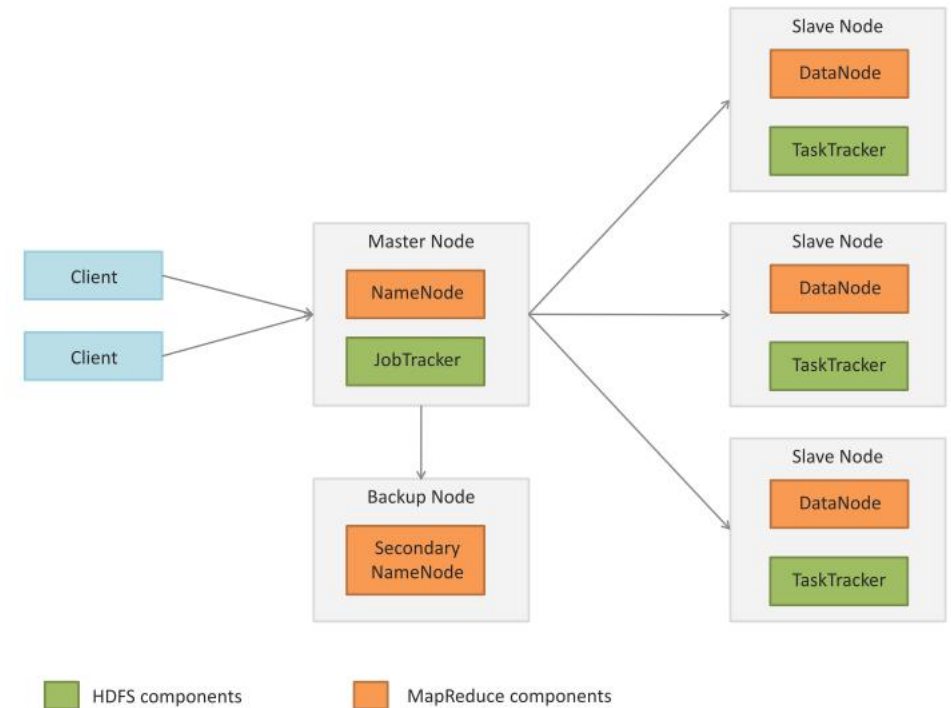
- Hadoop Ecosystem includes:

- Hadoop MapReduce
- HDFS
- YARN
- HBase
- Zookeeper
- Pig
- Hive
- Mahout
- Chukwa
- Cassandra
- Avro
- Oozie
- Flume
- Sqoop



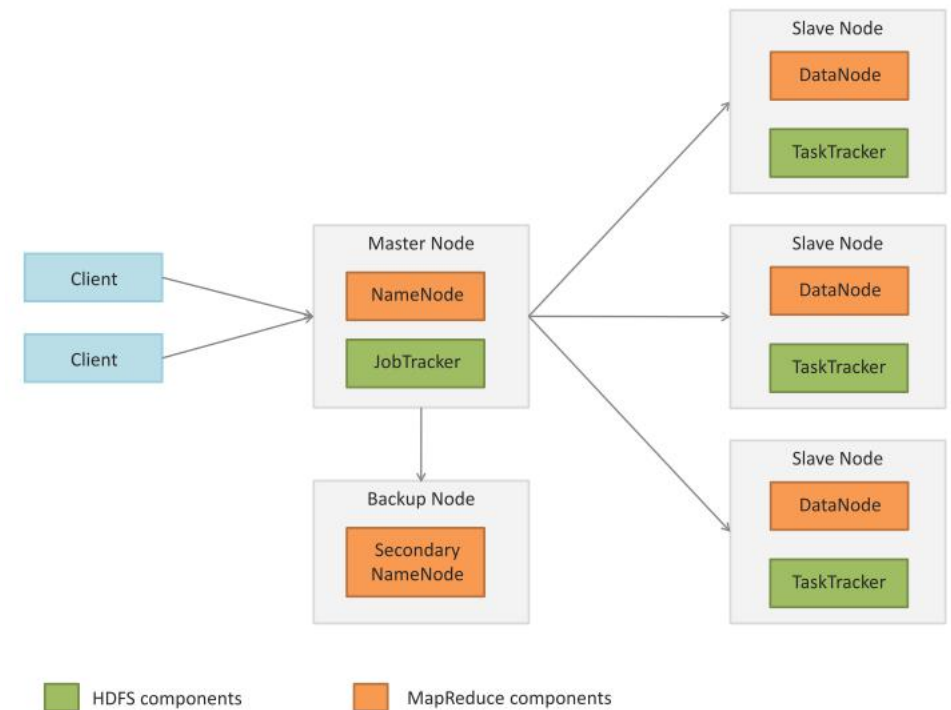
Apache Hadoop

- A Hadoop cluster comprises of a Master node, backup node and a number of slave nodes.
- The master node runs the NameNode and JobTracker processes and the slave nodes run the DataNode and TaskTracker components of Hadoop.
- The backup node runs the Secondary NameNode process.
- NameNode
 - NameNode keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself. Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file.
- Secondary NameNode
 - NameNode is a Single Point of Failure for the HDFS Cluster. An optional Secondary NameNode which is hosted on a separate machine creates checkpoints of the namespace.
- JobTracker
 - The JobTracker is the service within Hadoop that distributes MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.



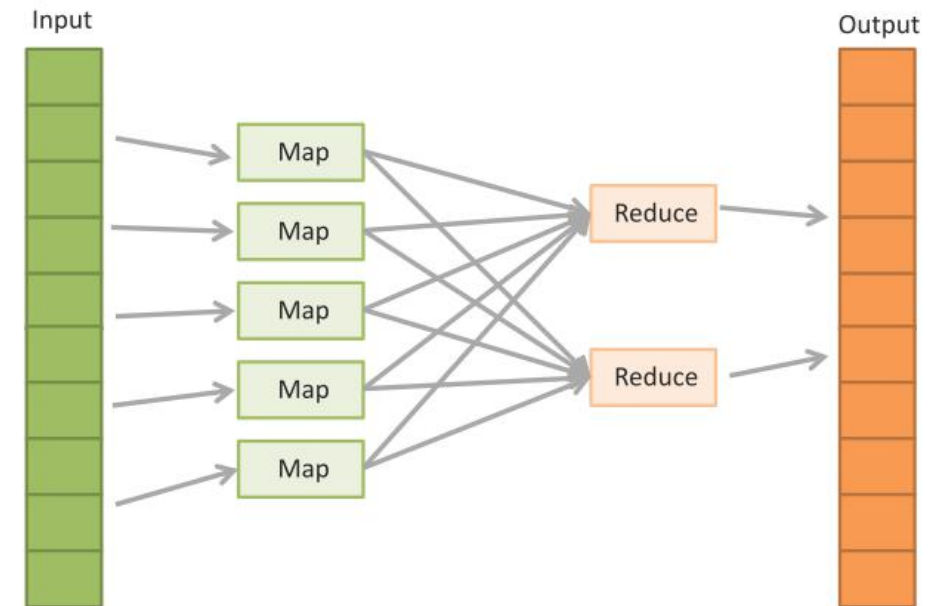
Apache Hadoop

- TaskTracker
 - TaskTracker is a node in a Hadoop cluster that accepts Map, Reduce and Shuffle tasks from the JobTracker.
 - Each TaskTracker has a defined number of slots which indicate the number of tasks that it can accept.
- DataNode
 - A DataNode stores data in an HDFS file system.
 - A functional HDFS filesystem has more than one DataNode, with data replicated across them.
 - DataNodes respond to requests from the NameNode for filesystem operations.
 - Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data.
 - Similarly, MapReduce operations assigned to TaskTracker instances near a DataNode, talk directly to the DataNode to access the files.
 - TaskTracker instances can be deployed on the same servers that host DataNode instances, so that MapReduce operations are performed close to the data.



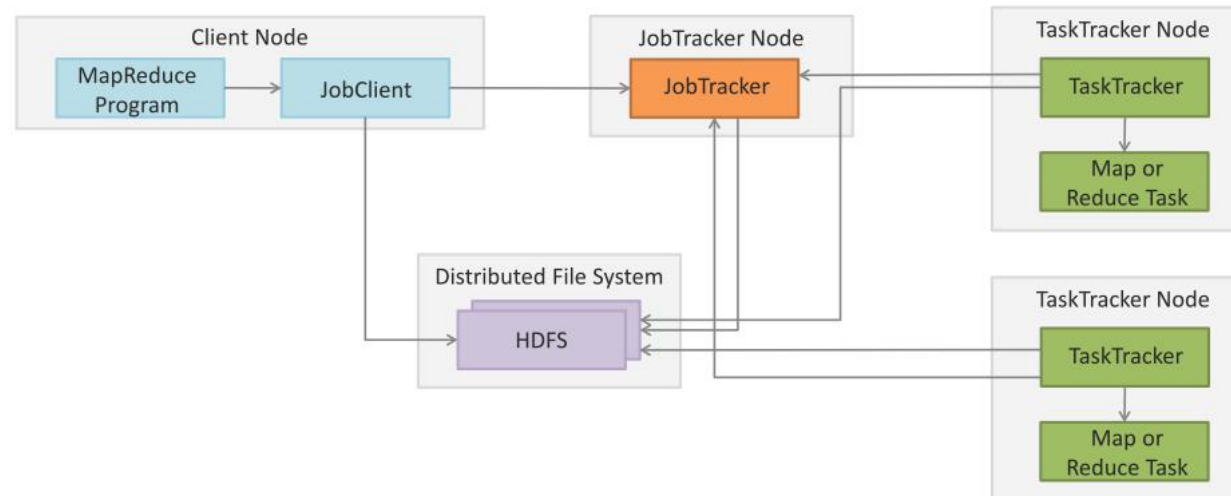
MapReduce

- MapReduce job consists of two phases:
 - Map: In the Map phase, data is read from a distributed file system and partitioned among a set of computing nodes in the cluster. The data is sent to the nodes as a set of key-value pairs. The Map tasks process the input records independently of each other and produce intermediate results as key-value pairs. The intermediate results are stored on the local disk of the node running the Map task.
 - Reduce: When all the Map tasks are completed, the Reduce phase begins in which the intermediate data with the same key is aggregated.
- Optional Combine Task
 - An optional Combine task can be used to perform data aggregation on the intermediate data of the same key for the output of the mapper before transferring the output to the Reduce task.



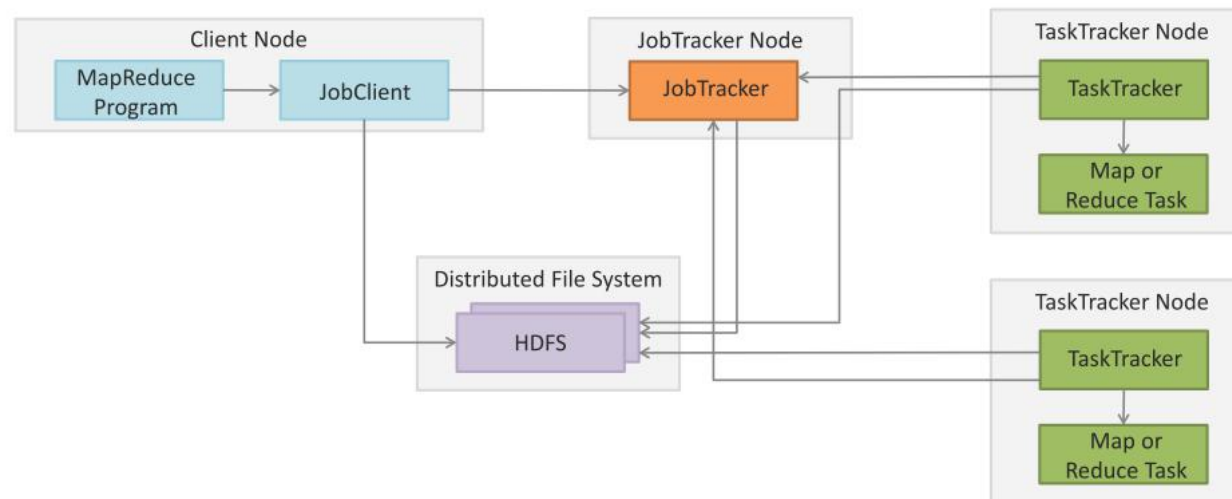
MapReduce Job Execution Workflow

- MapReduce job execution starts when the client applications submit jobs to the Job tracker.
- The JobTracker returns a JobID to the client application. The JobTracker talks to the NameNode to determine the location of the data.
- The JobTracker locates TaskTracker nodes with available slots at/or near the data.
- The TaskTrackers send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that they are still alive. These messages also inform the JobTracker of the number of available slots, so the JobTracker can stay up to date with where in the cluster, new work can be delegated.



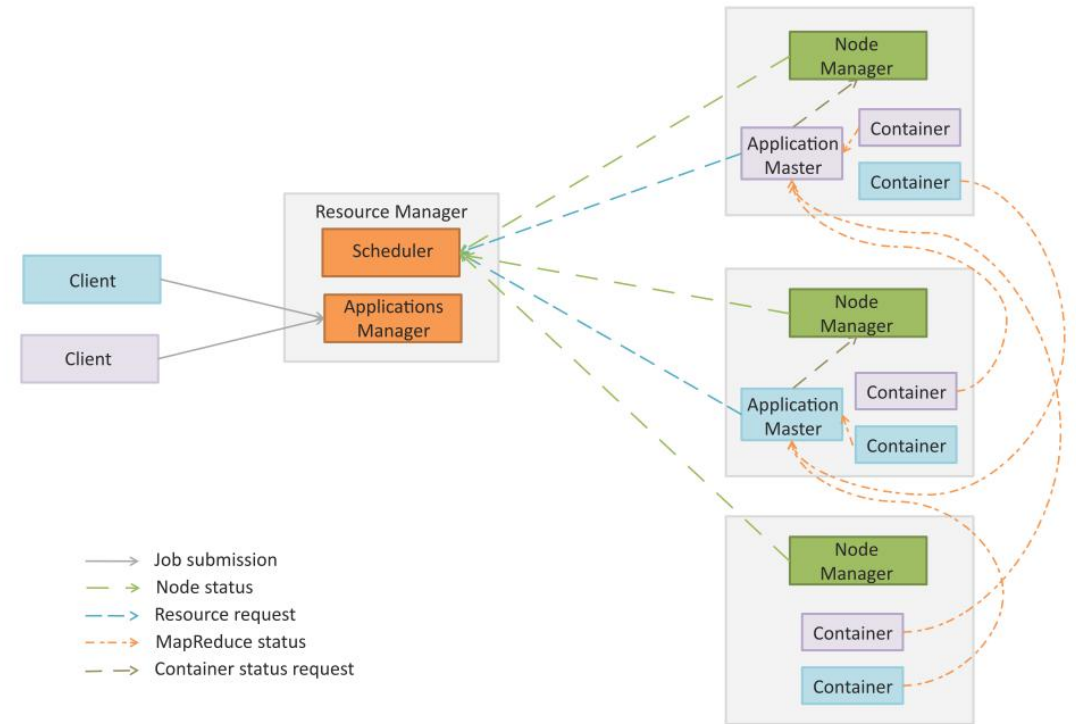
MapReduce Job Execution Workflow

- The JobTracker submits the work to the TaskTracker nodes when they poll for tasks. To choose a task for a TaskTracker, the JobTracker uses various scheduling algorithms (default is FIFO).
- The TaskTracker nodes are monitored using the heartbeat signals that are sent by the TaskTrackers to JobTracker.
- The TaskTracker spawns a separate JVM process for each task so that any task failure does not bring down the TaskTracker.
- The TaskTracker monitors these spawned processes while capturing the output and exit codes. When the process finishes, successfully or not, the TaskTracker notifies the JobTracker. When the job is completed, the JobTracker updates its status.



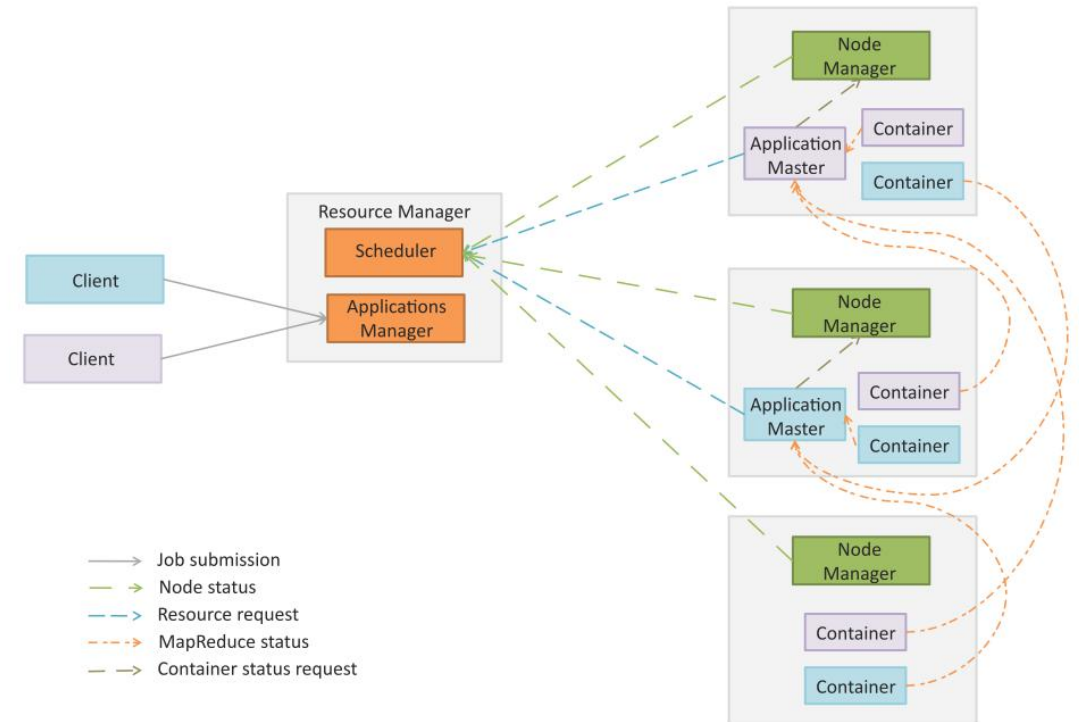
MapReduce 2.0 - YARN

- In Hadoop 2.0 the original processing engine of Hadoop (MapReduce) has been separated from the resource management (which is now part of YARN).
- This makes YARN effectively an operating system for Hadoop that supports different processing engines on a Hadoop cluster such as MapReduce for batch processing, Apache Tez for interactive queries, Apache Storm for stream processing, etc.
- YARN architecture divides the two major functions of the JobTracker - resource management and job life-cycle management - into separate components:
 - ResourceManager
 - ApplicationMaster.



YARN Components

- **Resource Manager (RM):** RM manages the global assignment of compute resources to applications. RM consists of two main services:
 - **Scheduler:** Scheduler is a pluggable service that manages and enforces the resource scheduling policy in the cluster.
 - **Applications Manager (AsM):** AsM manages the running Application Masters in the cluster. AsM is responsible for starting application masters and for monitoring and restarting them on different nodes in case of failures.
- **Application Master (AM):** A per-application AM manages the application's life cycle. AM is responsible for negotiating resources from the RM and working with the NMs to execute and monitor the tasks.
- **Node Manager (NM):** A per-machine NM manages the user processes on that machine.
- **Containers:** Container is a bundle of resources allocated by RM (memory, CPU, network, etc.). A container is a conceptual entity that grants an application the privilege to use a certain amount of resources on a given machine to run a component task.



Hadoop Schedulers

- Hadoop scheduler is a pluggable component that makes it open to support different scheduling algorithms.
- The default scheduler in Hadoop is FIFO.
- Two advanced schedulers are also available - the Fair Scheduler, developed at Facebook, and the Capacity Scheduler, developed at Yahoo.
- The pluggable scheduler framework provides the flexibility to support a variety of workloads with varying priority and performance constraints.
- Efficient job scheduling makes Hadoop a multi-tasking system that can process multiple data sets for multiple jobs for multiple users simultaneously.

FIFO Scheduler

- FIFO is the default scheduler in Hadoop that maintains a work queue in which the jobs are queued.
- The scheduler pulls jobs in first in first out manner (oldest job first) for scheduling.
- There is no concept of priority or size of job in FIFO scheduler.

Fair Scheduler

- The Fair Scheduler allocates resources evenly between multiple jobs and also provides capacity guarantees.
- Fair Scheduler assigns resources to jobs such that each job gets an equal share of the available resources on average over time.
- Tasks slots that are free are assigned to the new jobs, so that each job gets roughly the same amount of CPU time.
- Job Pools
 - The Fair Scheduler maintains a set of pools into which jobs are placed. Each pool has a guaranteed capacity.
 - When there is a single job running, all the resources are assigned to that job. When there are multiple jobs in the pools, each pool gets at least as many task slots as guaranteed.
 - Each pool receives at least the minimum share.
 - When a pool does not require the guaranteed share the excess capacity is split between other jobs.
- Fairness
 - The scheduler computes periodically the difference between the computing time received by each job and the time it should have received in ideal scheduling.
 - The job which has the highest deficit of the compute time received is scheduled next.

Capacity Scheduler

- The Capacity Scheduler has similar functionality as the Fair Scheduler but adopts a different scheduling philosophy.
- Queues
 - In Capacity Scheduler, you define a number of named queues each with a configurable number of map and reduce slots.
 - Each queue is also assigned a guaranteed capacity.
 - The Capacity Scheduler gives each queue its capacity when it contains jobs, and shares any unused capacity between the queues. Within each queue FIFO scheduling with priority is used.
- Fairness
 - For fairness, it is possible to place a limit on the percentage of running tasks per user, so that users share a cluster equally.
 - A wait time for each queue can be configured. When a queue is not scheduled for more than the wait time, it can preempt tasks of other queues to get its fair share.

Further Reading

- Apache Hadoop, <http://hadoop.apache.org>
- Apache Hive, <http://hive.apache.org>
- Apache HBase, <http://hbase.apache.org>
- Apache Chukwa, <http://chukwa.apache.org>
- Apache Flume, <http://flume.apache.org>
- Apache Zookeeper, <http://zookeeper.apache.org>
- Apache Avro, <http://avro.apache.org>
- Apache Oozie, <http://oozie.apache.org>
- Apache Storm, <http://storm-project.net>
- Apache Tez, <http://tez.incubator.apache.org>
- Apache Cassandra, <http://cassandra.apache.org>
- Apache Mahout, <http://mahout.apache.org>
- Apache Pig, <http://pig.apache.org>
- Apache Sqoop, <http://sqoop.apache.org>