Unit 2:
M2M & System Management

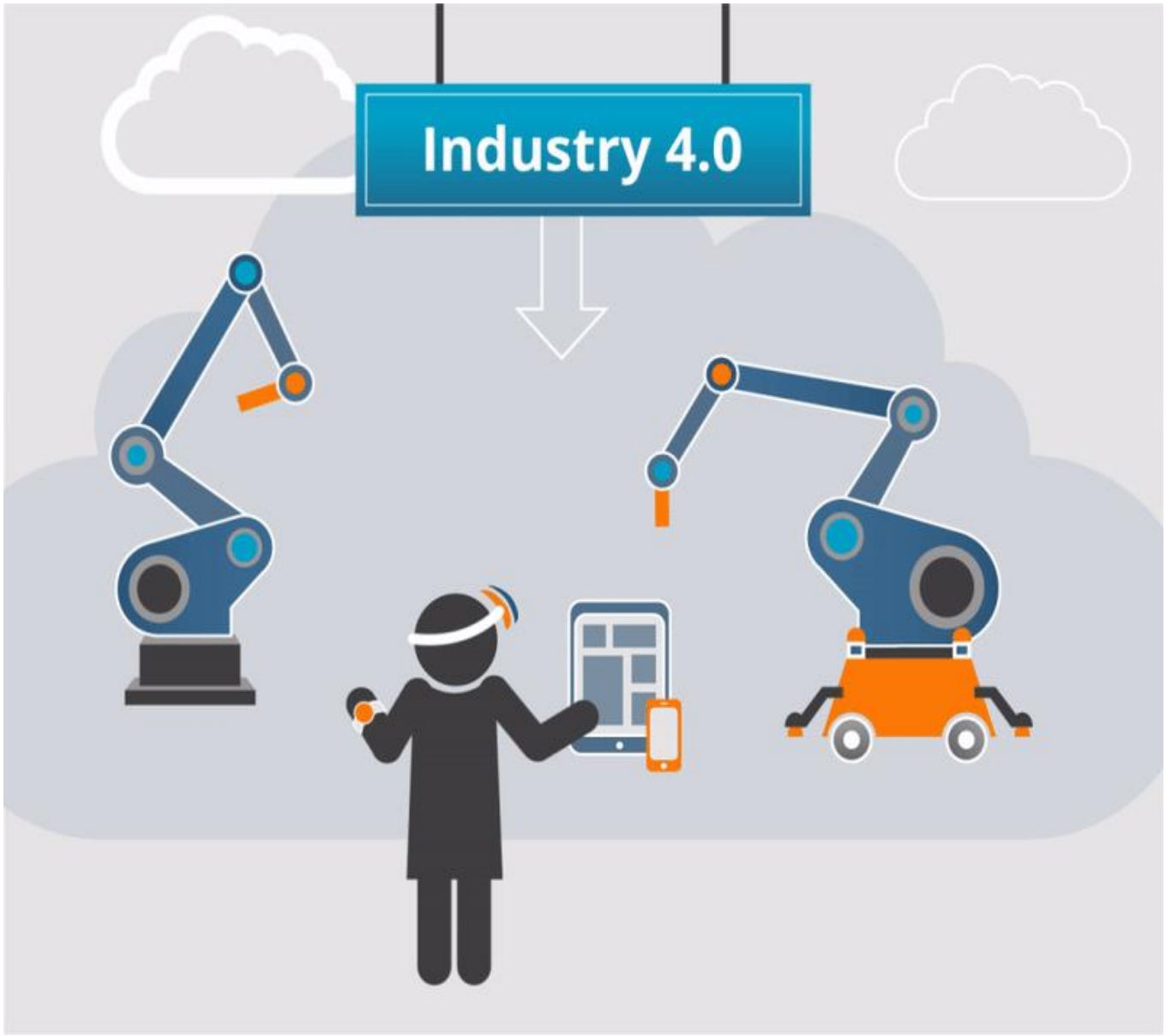
National Forensic Sciences University
Knowledge | Wisdom | Fulfilment
An Institution of National Importance
(Ministry of Home Affairs, Government of India)

Dr. Ujjaval Patel

Assistant Professor (IOT/SCADA)

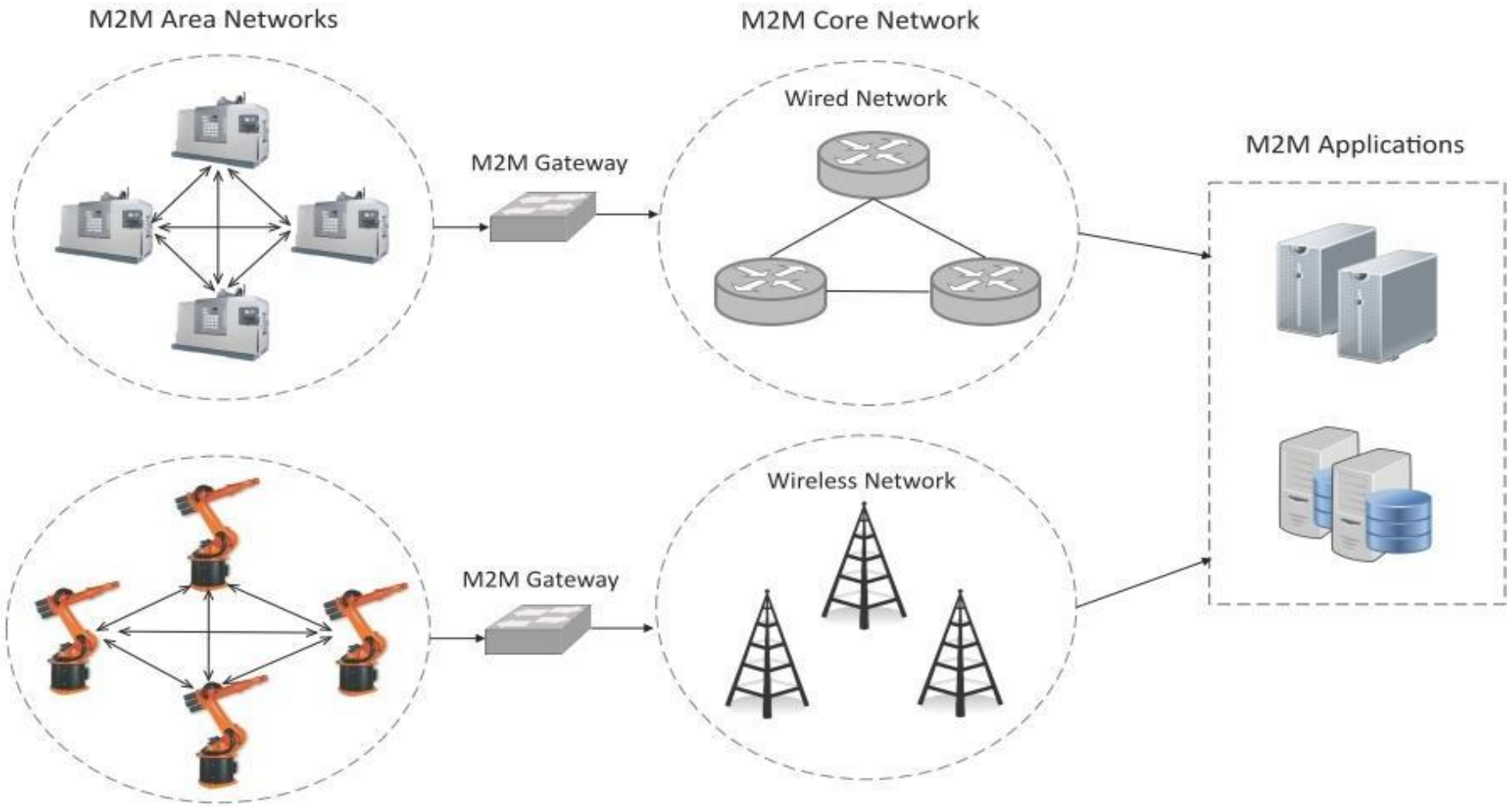
Unit Outlines:

- M2M
- Difference between IoT & M2M
- Software Defined Networking(SDN)
- Sensors used in IoT
- Network Function Virtualization (NFV)
- Simple Network Management Protocol (SNMP) & its limitations
- Network Operator Requirements
- IoT System Management with NETCONF & YANG



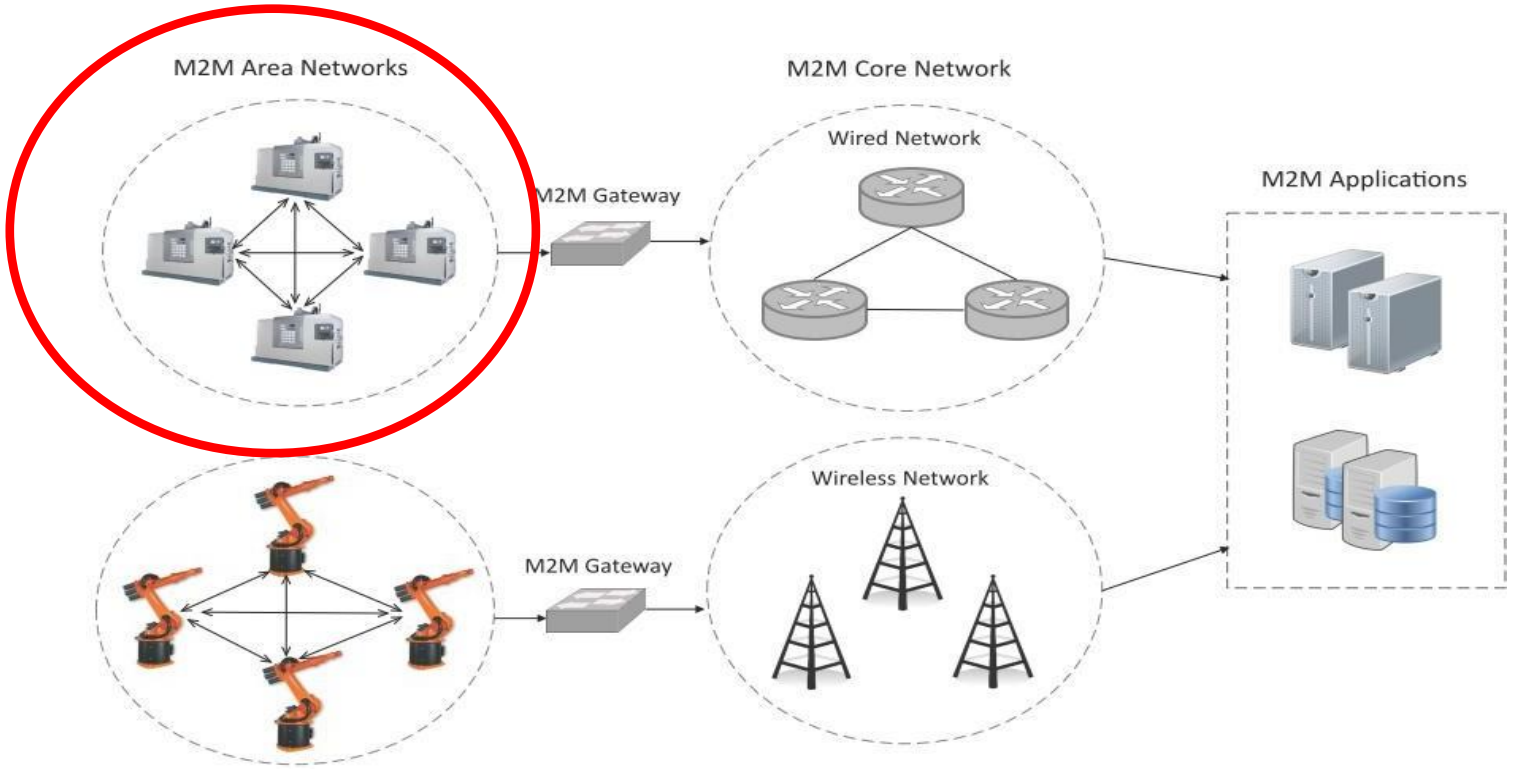
Introduction to Machine to Machine (M2M):

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



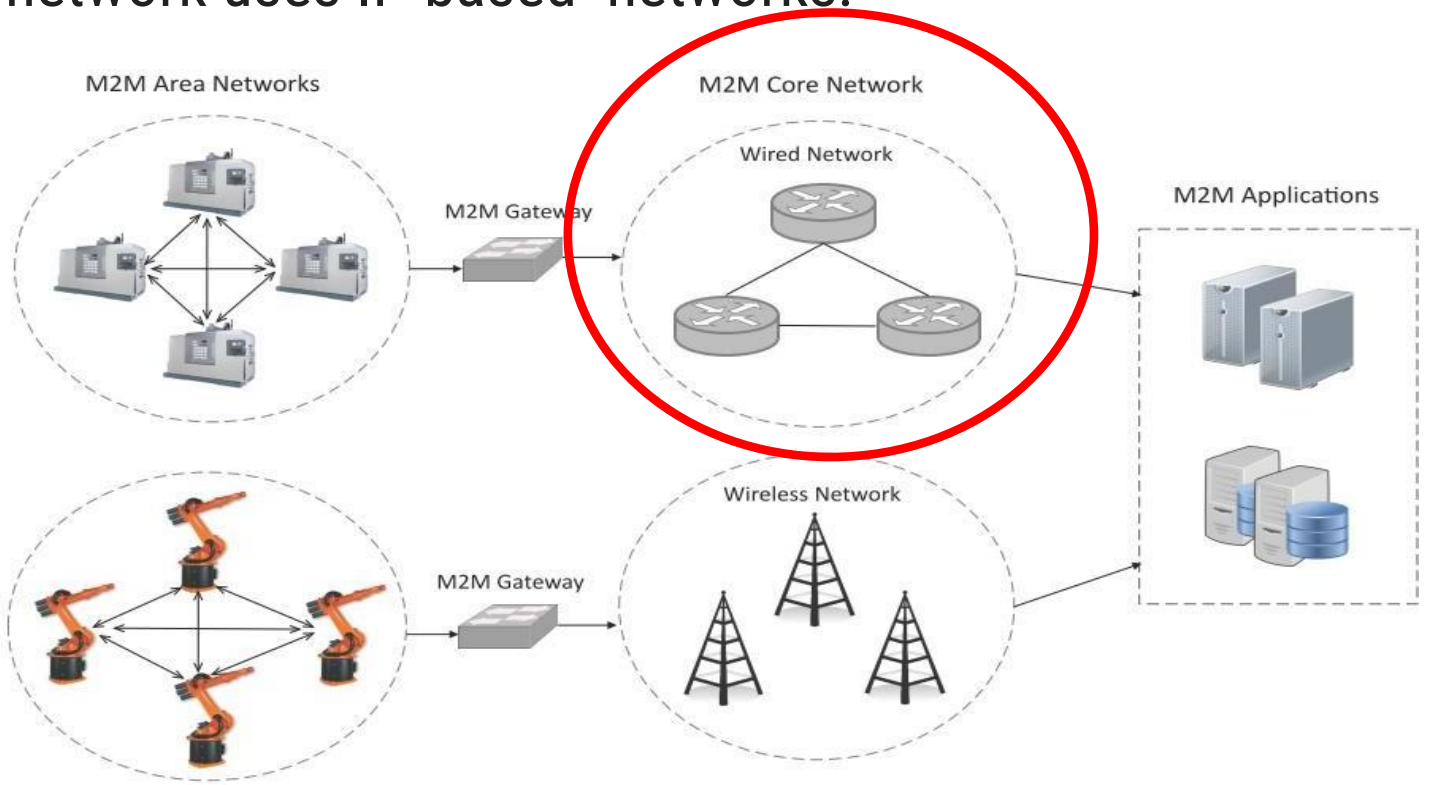
Introduction to Machine to Machine (M2M):

- 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, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.



Introduction to Machine to Machine (M2M):

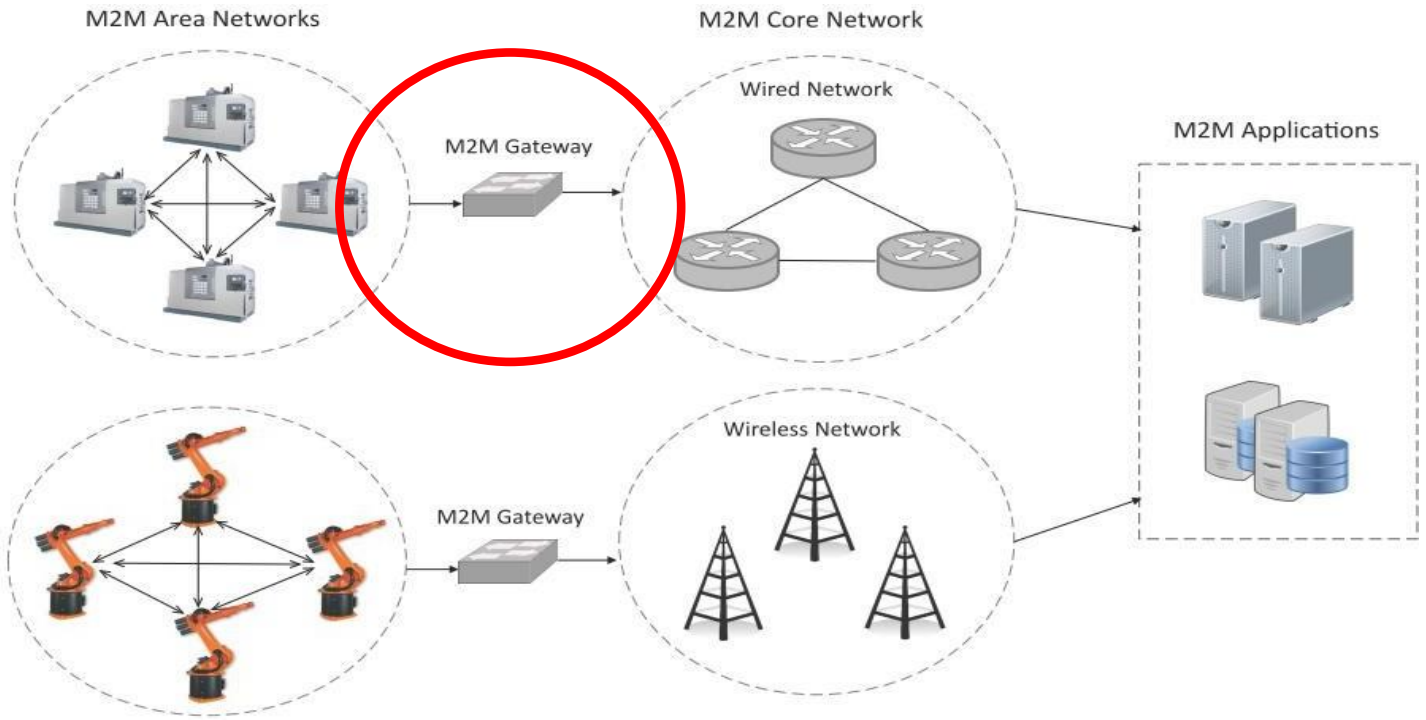
- The communication network provides connectivity to remote M2M area networks.
- The communication network can use either wired or wireless networks (IP- based).
- While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks.



Introduction to Machine to Machine (M2M):

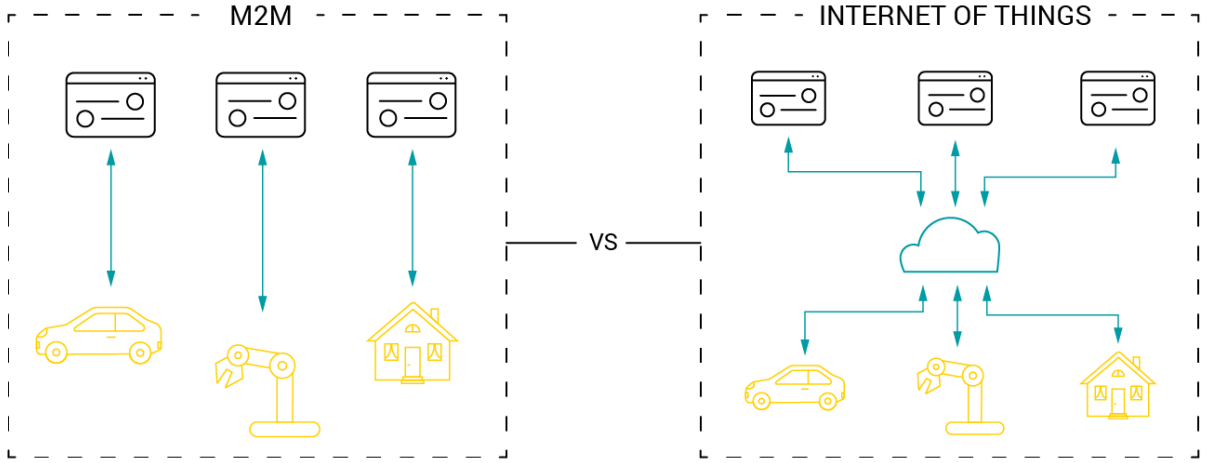
M2M Gateway:

- 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 & M2M:

M2M	IoT
Simple device-to-device communication usually within an embedded software at client site	Grand-scale projects and want-it-all approach
Isolated systems of devices using same standards	Integrates devices, data and applications across varying standards
Limited scalability options	Inherently more scalable
Wired or cellular network used for connectivity	Usually devices require active Internet connection
Extensive background of historical applications	State-of-the-art approach with roots in M2M

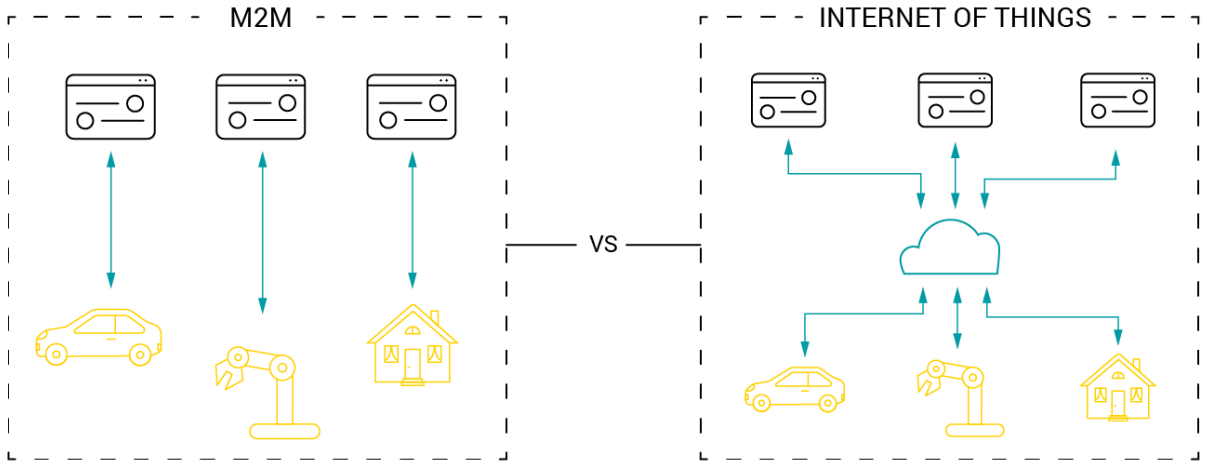


Communication Protocols

- ✓ 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.

Difference between IoT & M2M:

M2M	IoT
Simple device-to-device communication usually within an embedded software at client site	Grand-scale projects and want-it-all approach
Isolated systems of devices using same standards	Integrates devices, data and applications across varying standards
Limited scalability options	Inherently more scalable
Wired or cellular network used for connectivity	Usually devices require active Internet connection
Extensive background of historical applications	State-of-the-art approach with roots in M2M

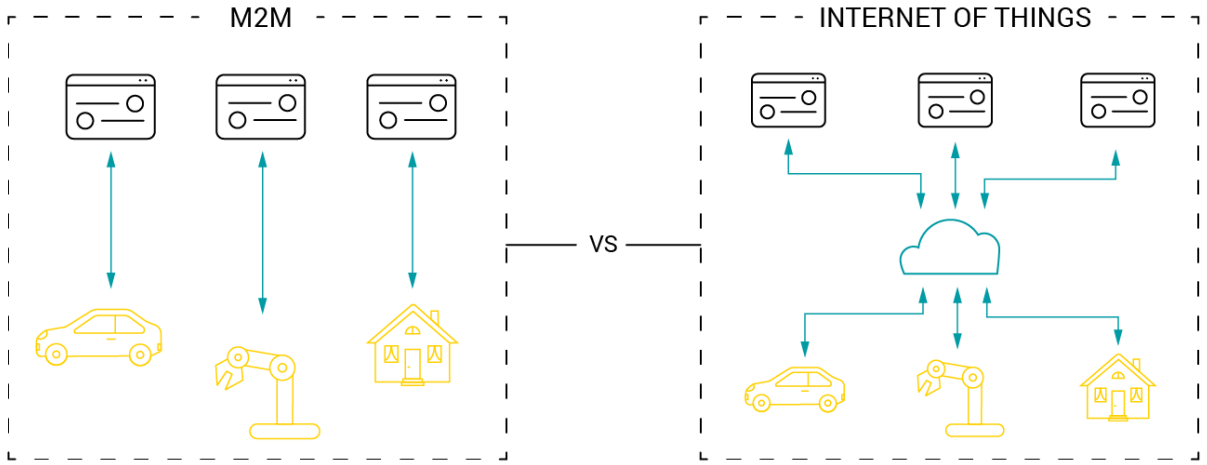


Machines in M2M vs Things in IoT

- ✓ 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 & M2M:

M2M	IoT
Simple device-to-device communication usually within an embedded software at client site	Grand-scale projects and want-it-all approach
Isolated systems of devices using same standards	Integrates devices, data and applications across varying standards
Limited scalability options	Inherently more scalable
Wired or cellular network used for connectivity	Usually devices require active Internet connection
Extensive background of historical applications	State-of-the-art approach with roots in M2M

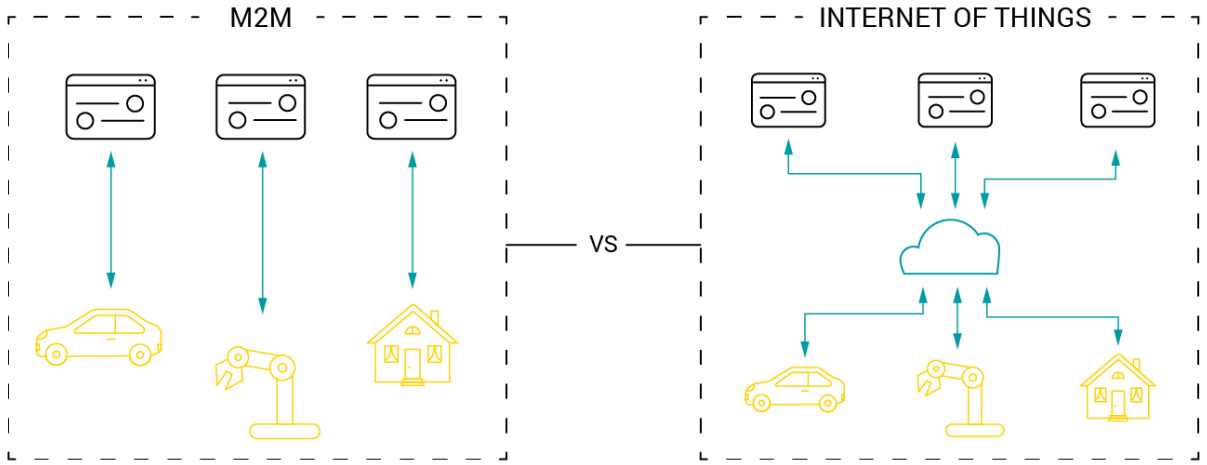


Hardware vs Software Emphasis

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

Difference between IoT & M2M:

M2M	IoT
Simple device-to-device communication usually within an embedded software at client site	Grand-scale projects and want-it-all approach
Isolated systems of devices using same standards	Integrates devices, data and applications across varying standards
Limited scalability options	Inherently more scalable
Wired or cellular network used for connectivity	Usually devices require active Internet connection
Extensive background of historical applications	State-of-the-art approach with roots in M2M

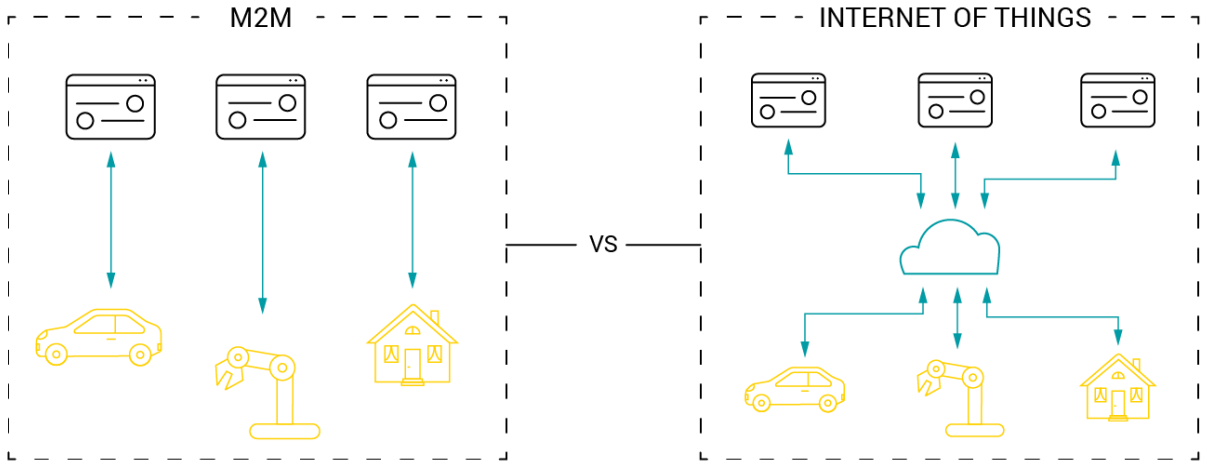


Data Collection & Analysis

- ✓ 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).

Difference between IoT & M2M:

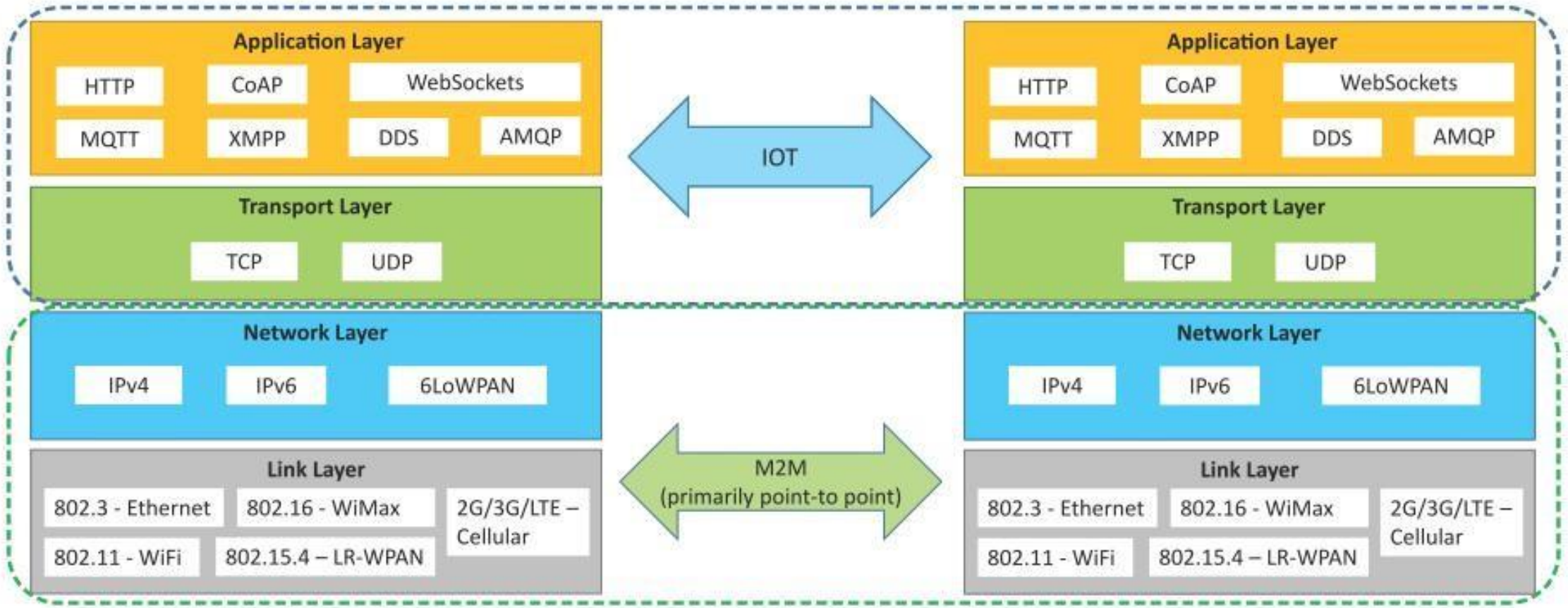
M2M	IoT
Simple device-to-device communication usually within an embedded software at client site	Grand-scale projects and want-it-all approach
Isolated systems of devices using same standards	Integrates devices, data and applications across varying standards
Limited scalability options	Inherently more scalable
Wired or cellular network used for connectivity	Usually devices require active Internet connection
Extensive background of historical applications	State-of-the-art approach with roots in M2M



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-premisis 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.

Difference between IoT & M2M:



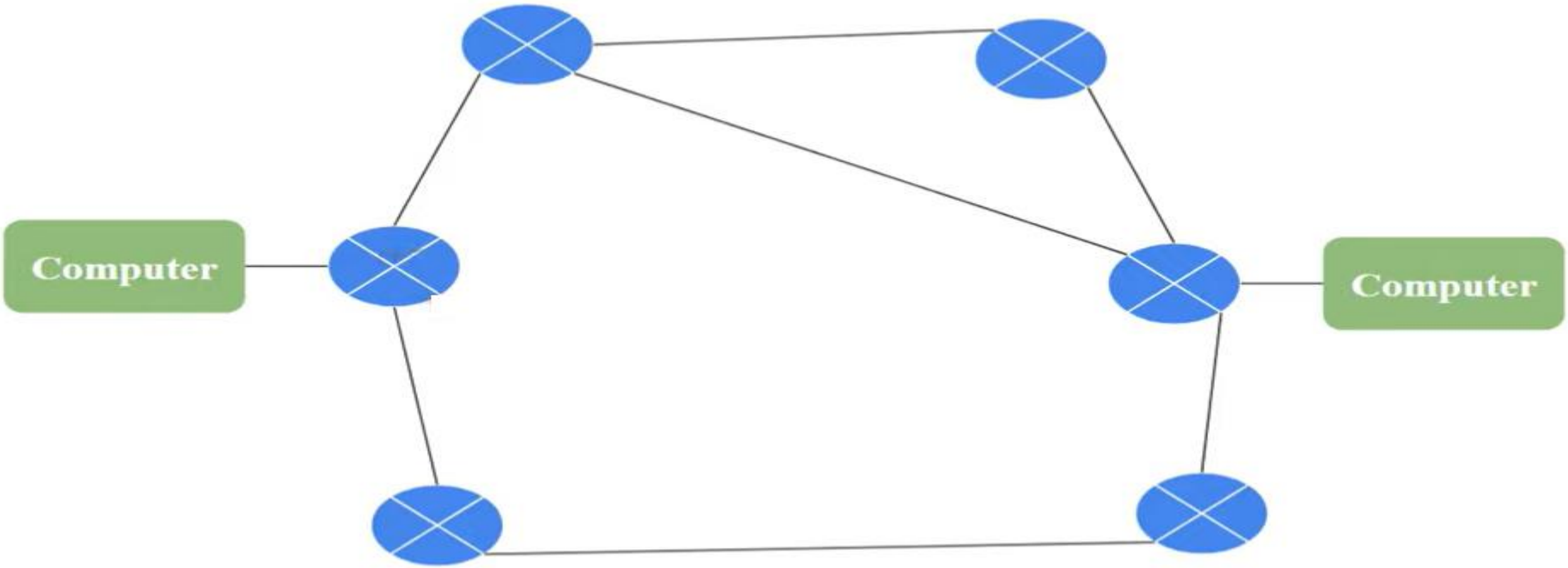

SDN

Software Defined Networking SDN

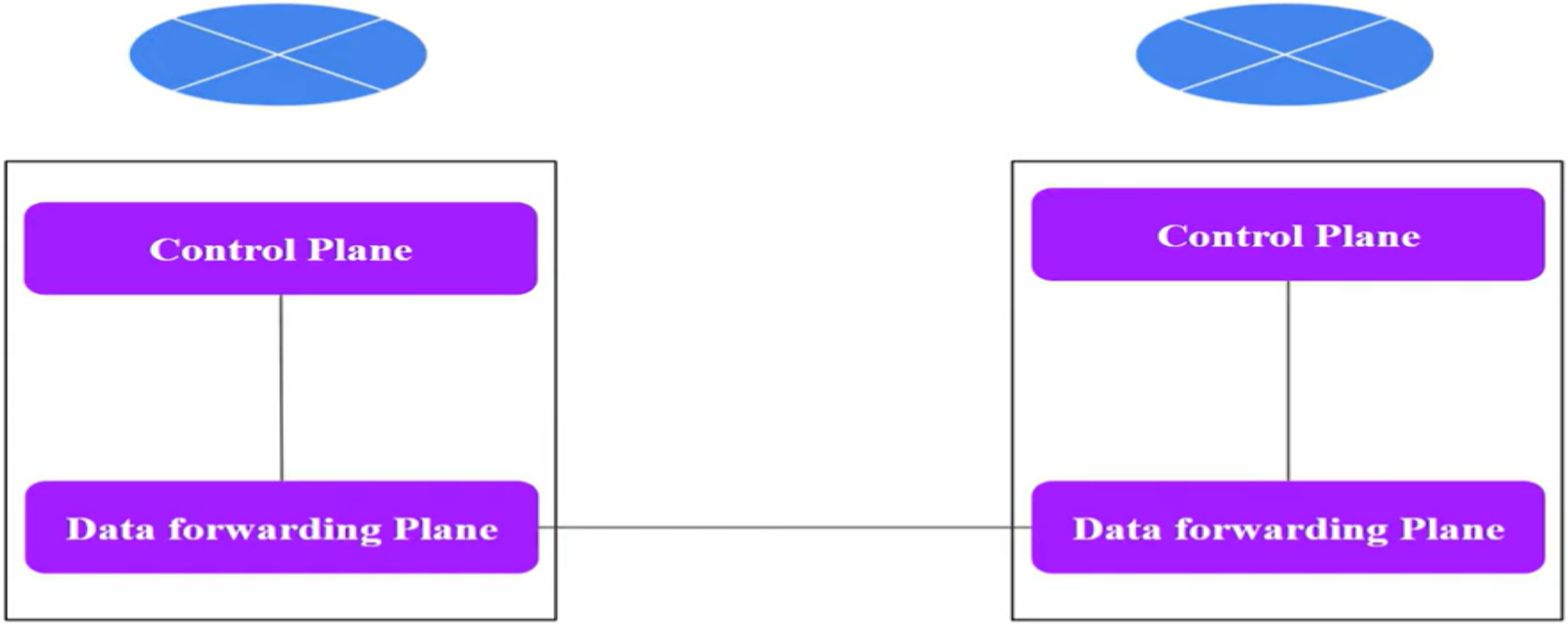
- Traditional Networking
- Software Defined Networking
- SDN Network architecture
- SDN controller
- Openflow protocol
- SDN benefits
- SDN challenges

Traditional Network

National Forensic Sciences University
Knowledge | Wisdom | Fulfilment
An Institution of National Importance
(Ministry of Home Affairs, Government of India)

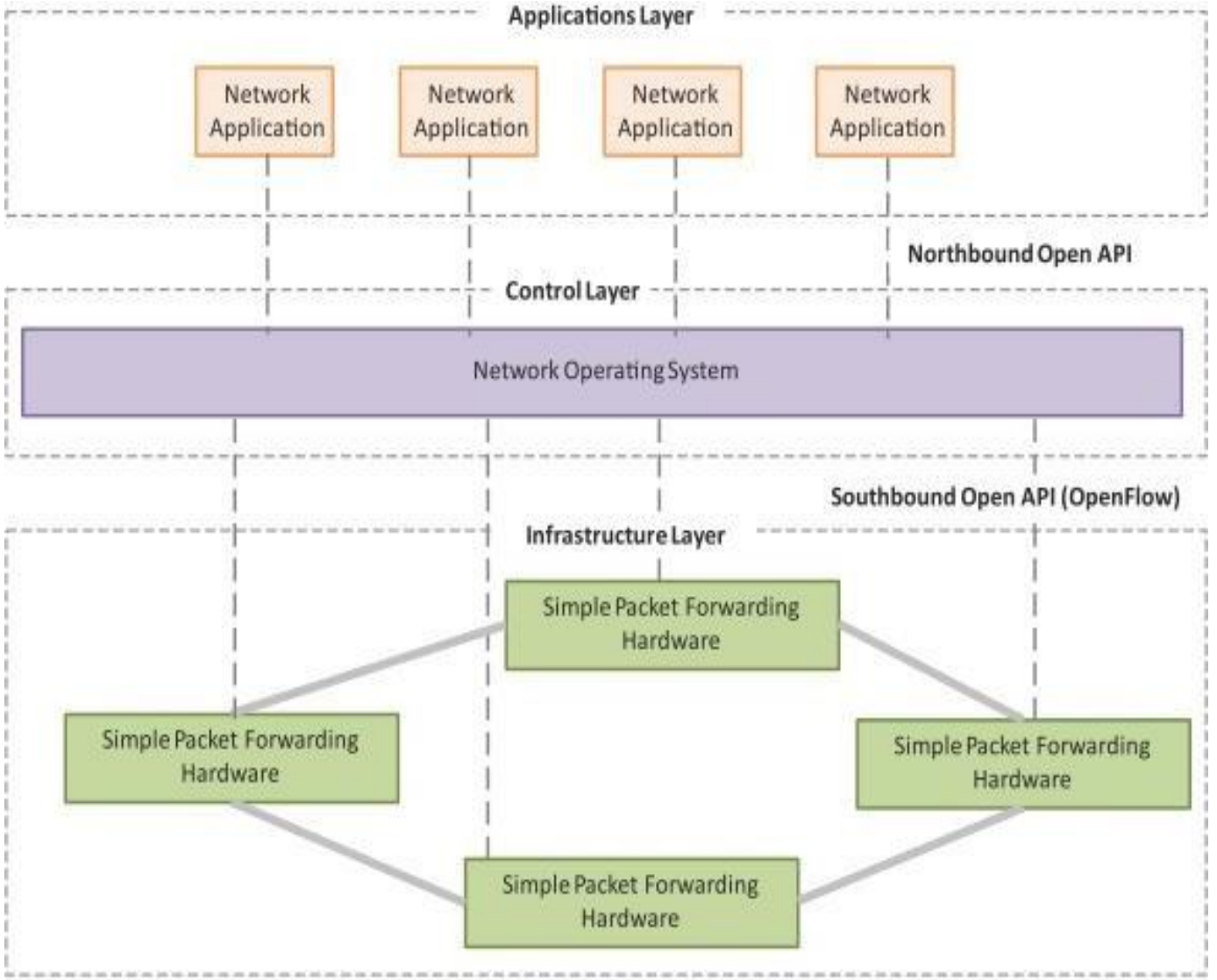


Traditional Network Architecture

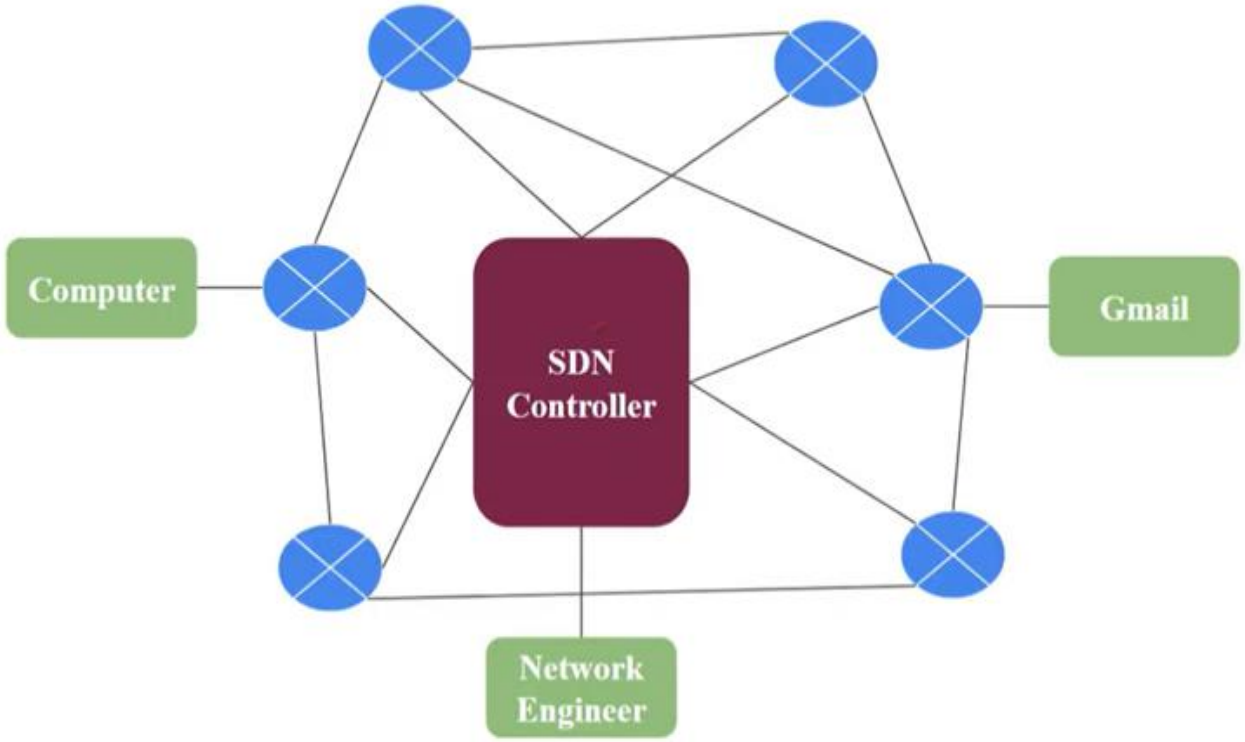


SDN:

- ✓ 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.



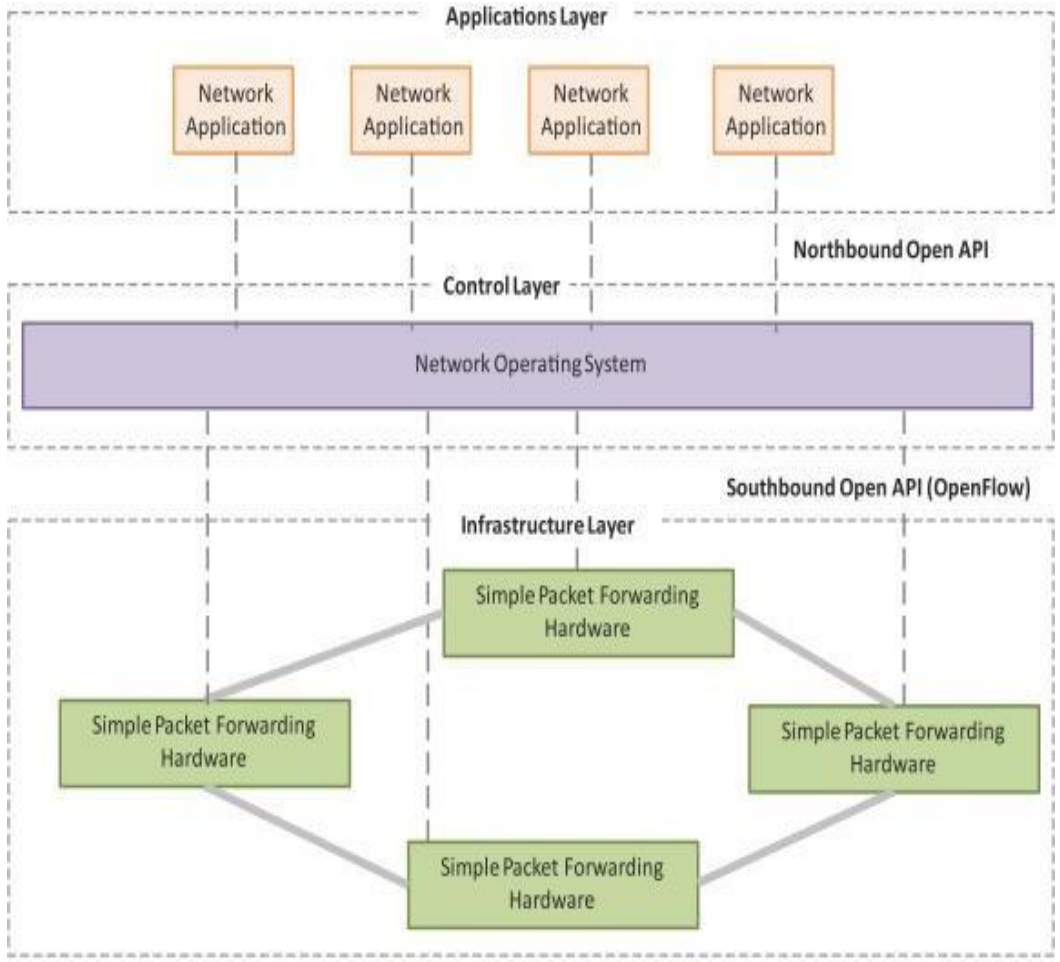
SDN:



- ✓ An SDN controller is an application in a software-defined networking (SDN) architecture that manages flow control for improved network management and application performance. The SDN controller platform typically runs on a server and uses protocols to tell switches where to send packets.

- ✓ SDN controllers direct traffic according to forwarding policies that a network operator puts in place, thereby minimizing manual configurations for individual network devices. By taking the control plane off of the network hardware and running it instead as software, the centralized controller facilitates automated network management and makes it easier to integrate and administer business applications. In effect, the SDN controller serves as a sort of operating system (OS) for the network.

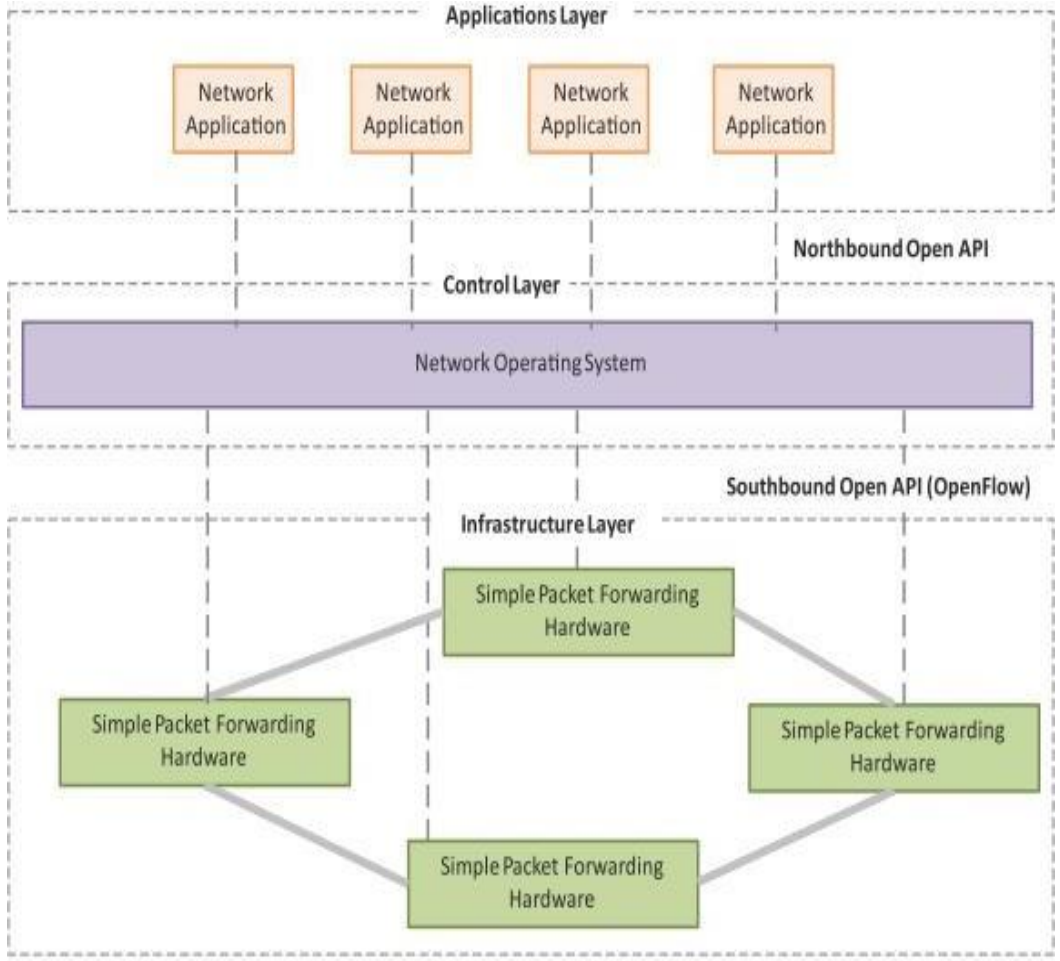
SDN:



- ✓ The controller is the core of a software-defined network. It resides between network devices at one end of the network and applications at the other end. Any communication between applications and network devices must go through the controller.

- ✓ The controller communicates with applications -- such as firewalls or load balancers - via northbound interfaces. The Open Networking Foundation (ONF) created a working group in 2013 focused specifically on northbound APIs and their development. The industry never settled on a standardized set, however, largely because application requirements vary so widely.

SDN:



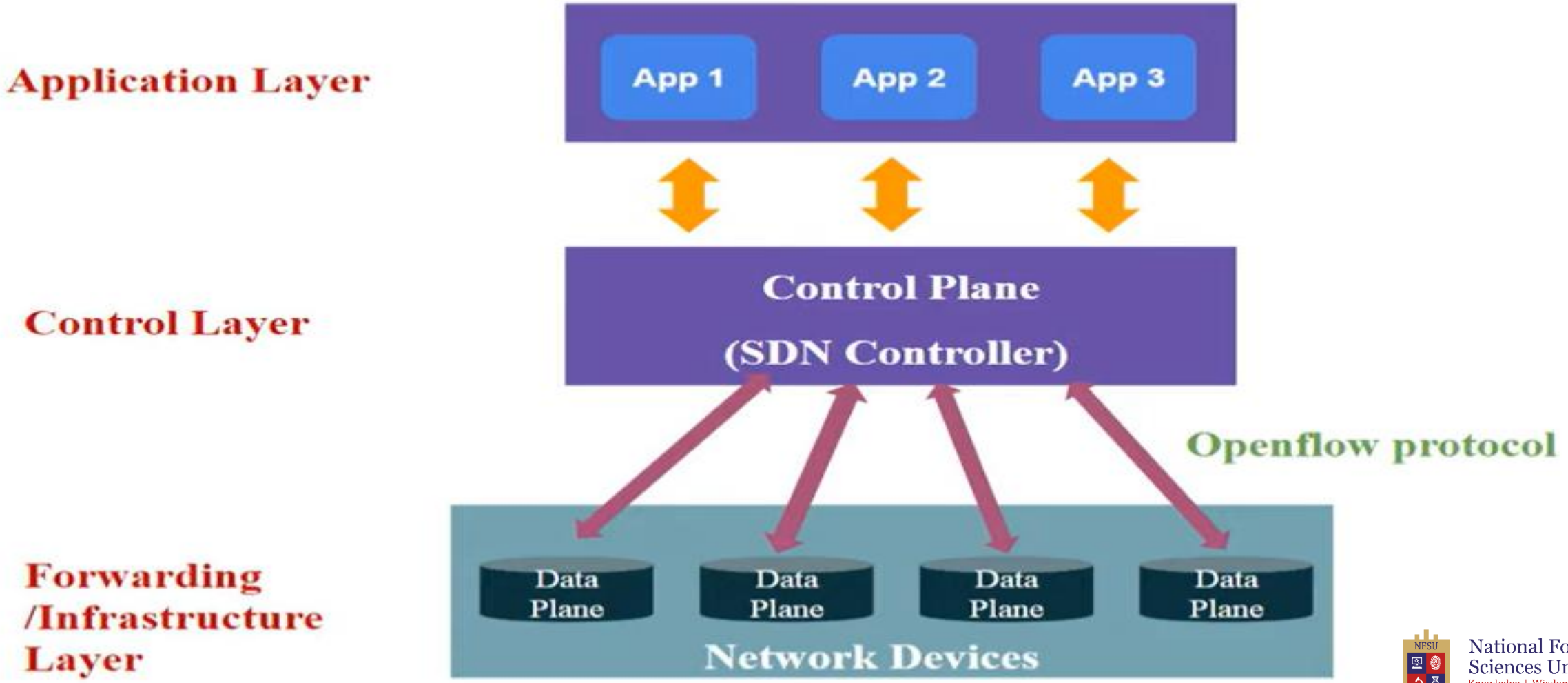
- ✓ The controller talks with individual network devices using a southbound interface, traditionally one like the OpenFlow protocol. These southbound protocols allow the controller to configure network devices and choose the optimal network path for application traffic. OpenFlow was created by ONF in 2011.

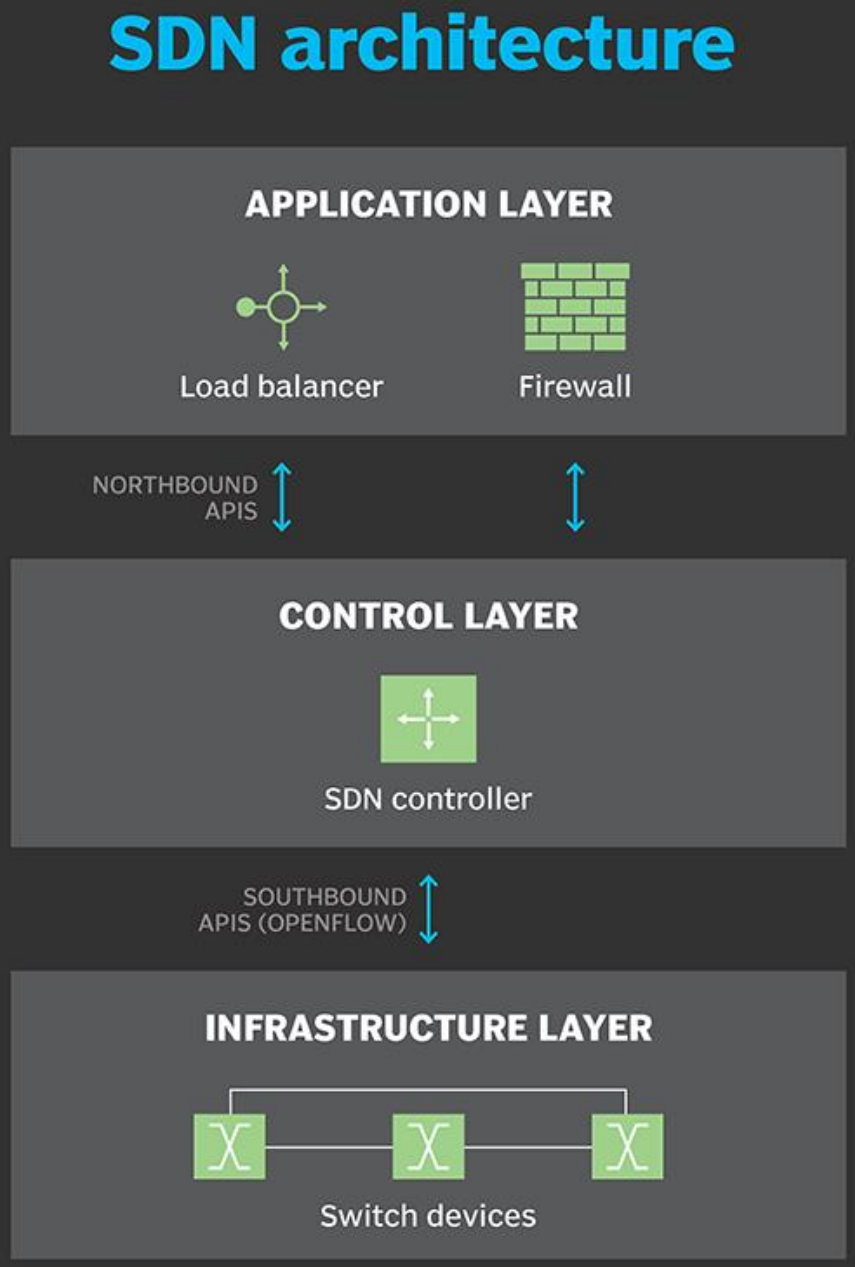
SDN controller vendors

Vendors that offer SDN controllers include the following:

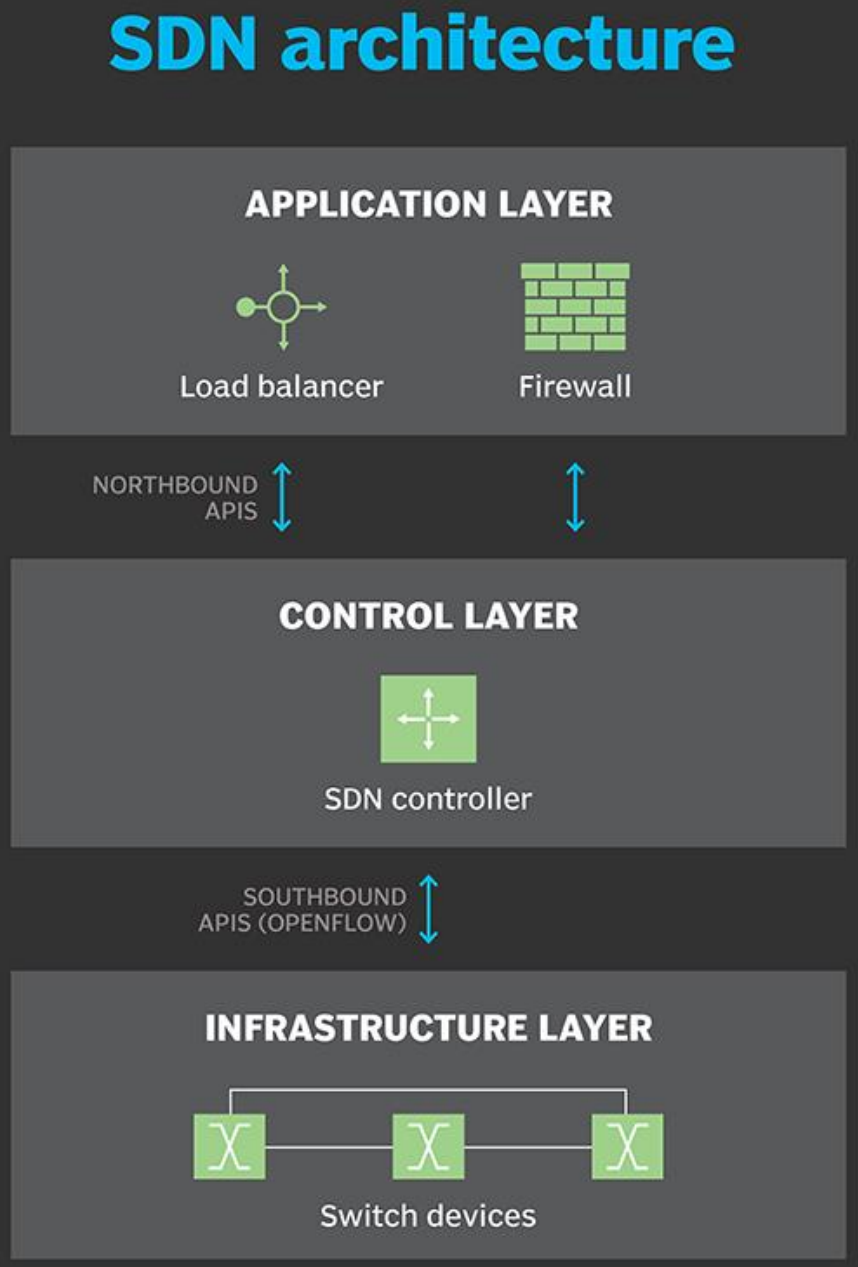
- ✓ Big Switch Networks
- ✓ Cisco
- ✓ Cumulus Networks
- ✓ Hewlett Packard Enterprise
- ✓ Juniper Networks
- ✓ Nuage Networks
- ✓ Pica8
- ✓ Pluribus Networks
- ✓ VMware

SDN Network Architecture





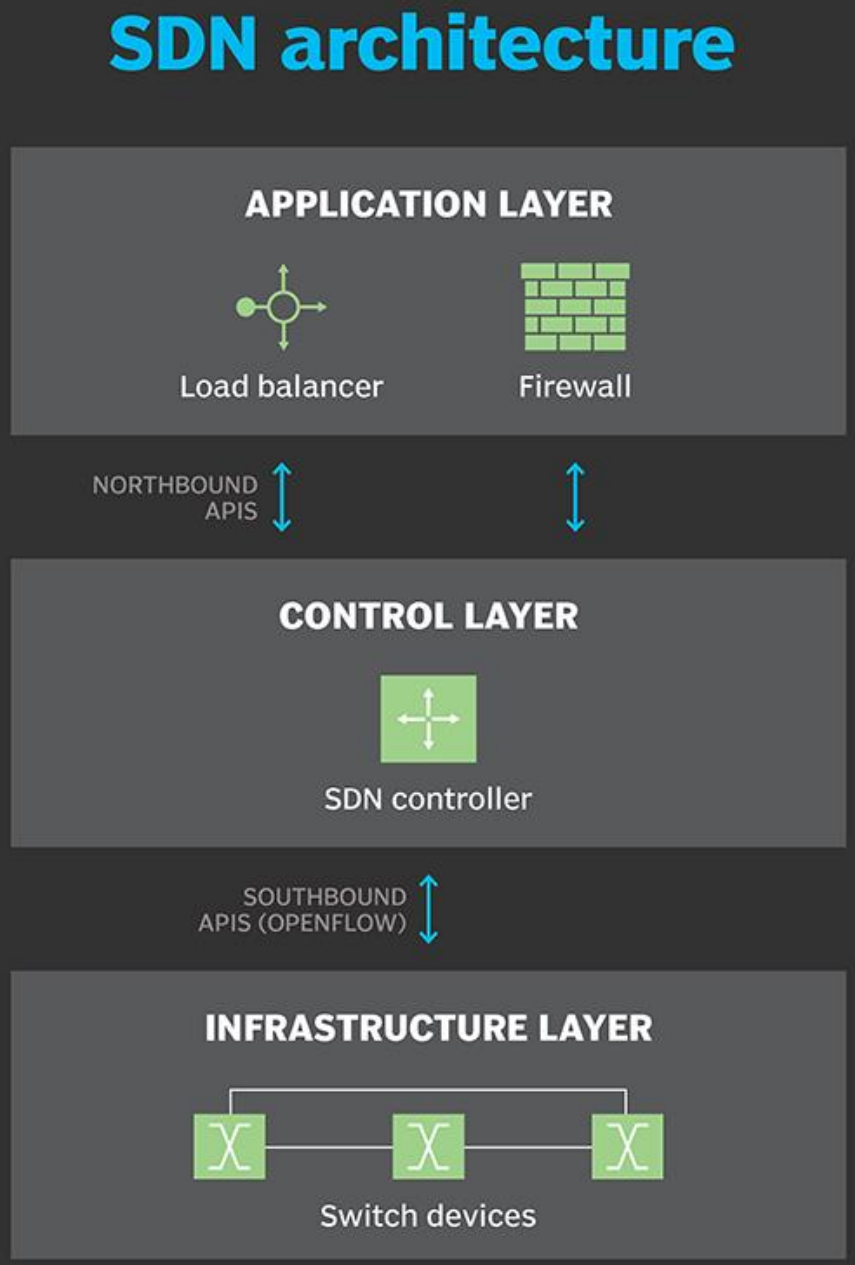
- ✓ Software-defined networking (SDN) is an architecture that abstracts different, distinguishable layers of a network to make networks agile and flexible. The goal of SDN is to improve network control by enabling enterprises and service providers to respond quickly to changing business requirements.
- ✓ In a software-defined network, a network engineer or administrator can shape traffic from a centralized control console without having to touch individual switches in the network. A centralized SDN controller directs the switches to deliver network services wherever they're needed, regardless of the specific connections between a server and devices.
- ✓ This process is a move away from traditional network architecture, in which individual network devices make traffic decisions based on their configured routing tables.



- ✓ A typical representation of SDN architecture comprises three layers:
  1. Application layer,
  2. Control layer and
  3. Infrastructure layer.
- ✓ These layers communicate using northbound and southbound application programming interfaces (APIs).

1. Application layer

- ✓ The application layer contains the typical network applications or functions organizations use. This can include intrusion detection systems, load balancing or firewalls. Where a traditional network would use a specialized appliance, such as a firewall or load balancer, a software-defined network replaces the appliance with an application that uses a controller to manage data plane behavior.



2. Control layer

- ✓ The control layer represents the centralized SDN controller software that acts as the brain of the software-defined network. This controller resides on a server and manages policies and traffic flows throughout the network.

3. Infrastructure layer

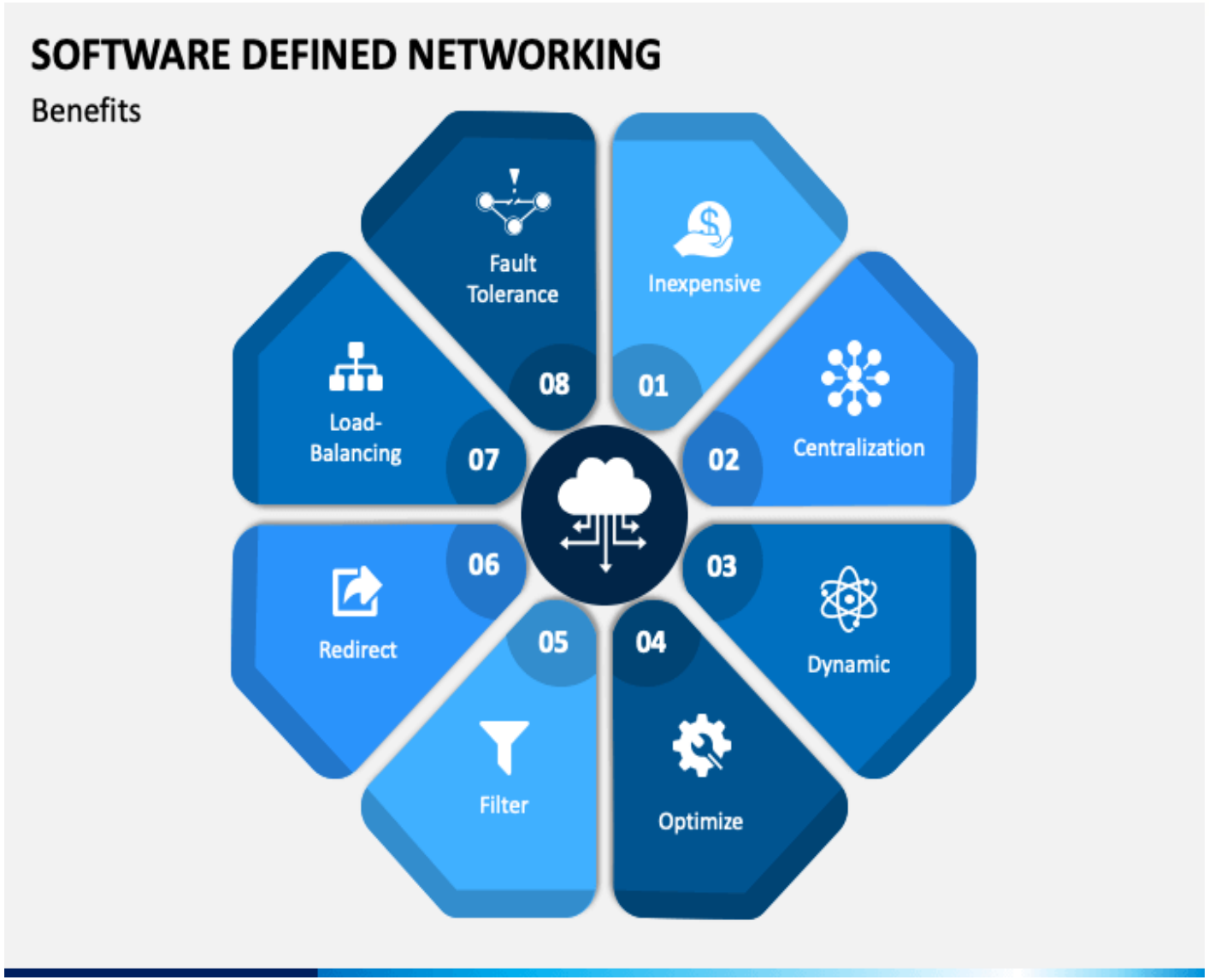
- ✓ The infrastructure layer is made up of the physical switches in the network. These switches forward the network traffic to their destinations.

APIs

- ✓ These three layers communicate using respective northbound and southbound APIs. Applications talk to the controller through its northbound interface. The controller and switches communicate using southbound interfaces, such as OpenFlow, although other protocols exist.

Benefits of SDN

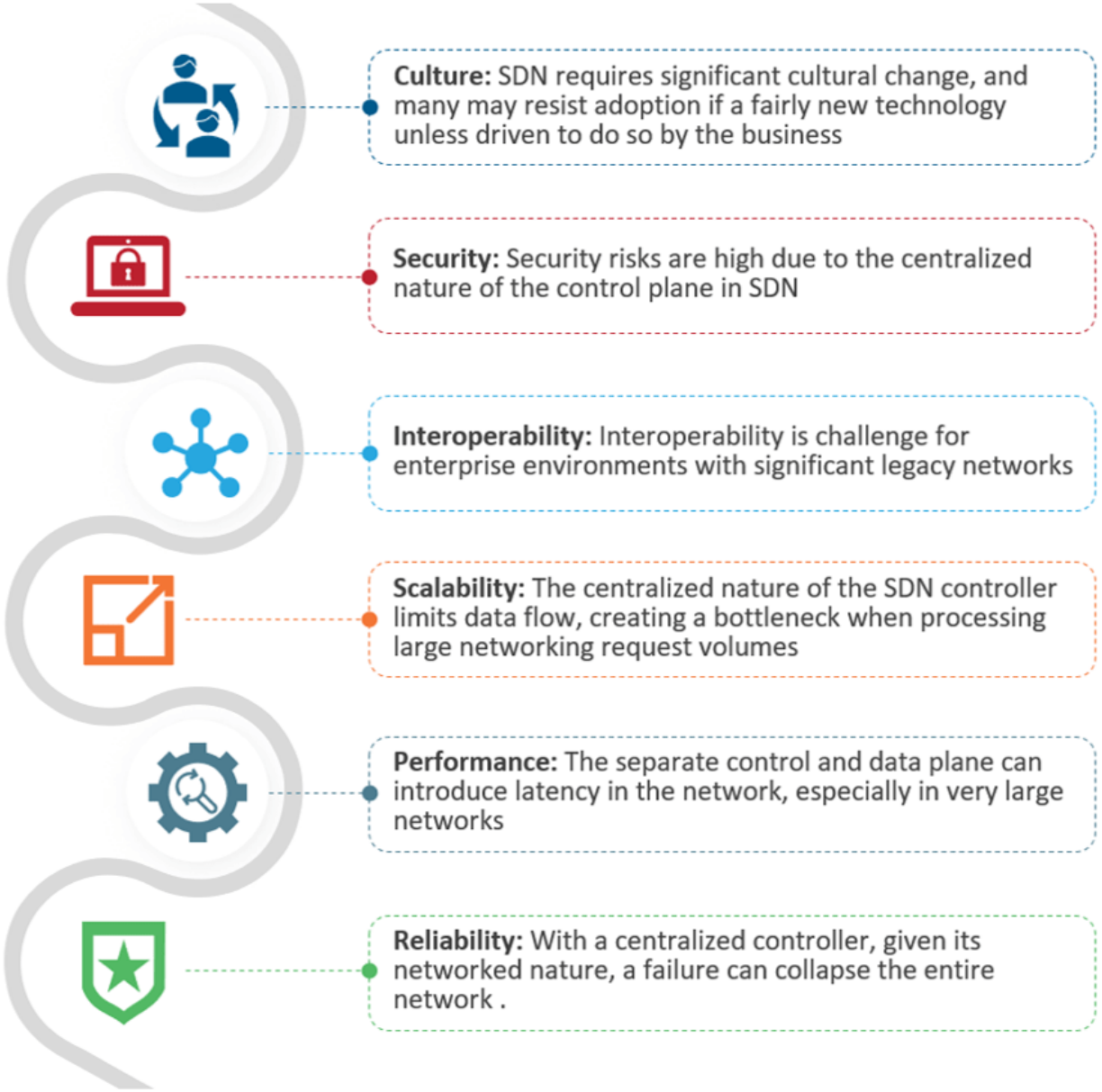
- Management :
- Programmable networks:
- Centralized management:
- Agility :
- Visibility :
- Cost Efficiency :
- Security:



SDN Insights:

- ✓ One major benefit of SDN controllers is that the centralized controller is aware of all the available network paths and can direct packets based on traffic requirements. Because of the controller's visibility into the network, it can automatically modify traffic flows and notify network operators about congested links.
- ✓ Companies can -- and should -- use more than one controller, adding a backup for redundancy. Three seems to be a common number among both commercial and open source SDN options. This redundancy will enable the network to continue running in the event of lost connectivity or controller susceptibility.
- ✓ The controller acts as a single point of failure, so securing it is pivotal to any software-defined network. Whoever owns the controller has access to the entire network.
- ✓ This means network operators should create security and authentication policies to ensure only the right people have access.

Challenges in SDN:



Culture: SDN requires significant cultural change, and many may resist adoption if a fairly new technology unless driven to do so by the business

Security: Security risks are high due to the centralized nature of the control plane in SDN

Interoperability: Interoperability is challenge for enterprise environments with significant legacy networks

Scalability: The centralized nature of the SDN controller limits data flow, creating a bottleneck when processing large networking request volumes

Performance: The separate control and data plane can introduce latency in the network, especially in very large networks

Reliability: With a centralized controller, given its networked nature, a failure can collapse the entire network .

Sensors and Their Interfacing

Dr. Ujjaval Patel

National Forensic Sciences University
Knowledge | Wisdom | Fulfilment
An Institution of National Importance
(Ministry of Home Affairs, Government of India)

Function of Sensor ?

- ✓ They perform some input functions by sensing or feeling the physical changes in characteristics of a system in response to a stimuli.
- ✓ For example heat is converted to electrical signals in a temperature sensor, or atmospheric pressure is converted to electrical signals in a barometer.

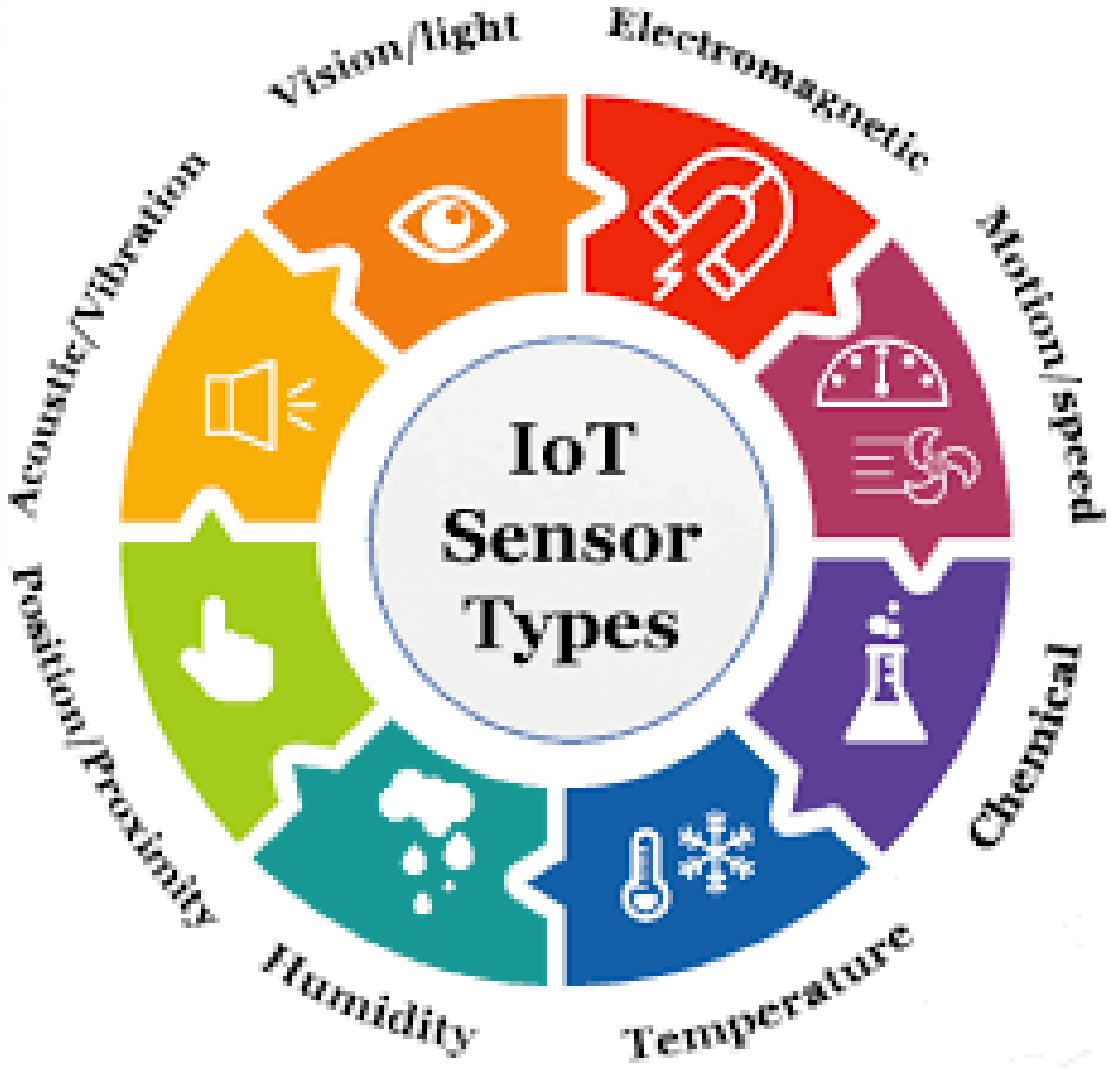
Features of Sensors

- ✓ It is only sensitive to the measured property (e.g., A temperature sensor senses the ambient temperature of a room.)
- ✓ It is insensitive to any other property likely to be encountered in its application (e.g., A temperature sensor does not bother about light or pressure while sensing the temperature.)
- ✓ It does not influence the measured property (e.g., measuring the temperature does not reduce or increase the temperature).

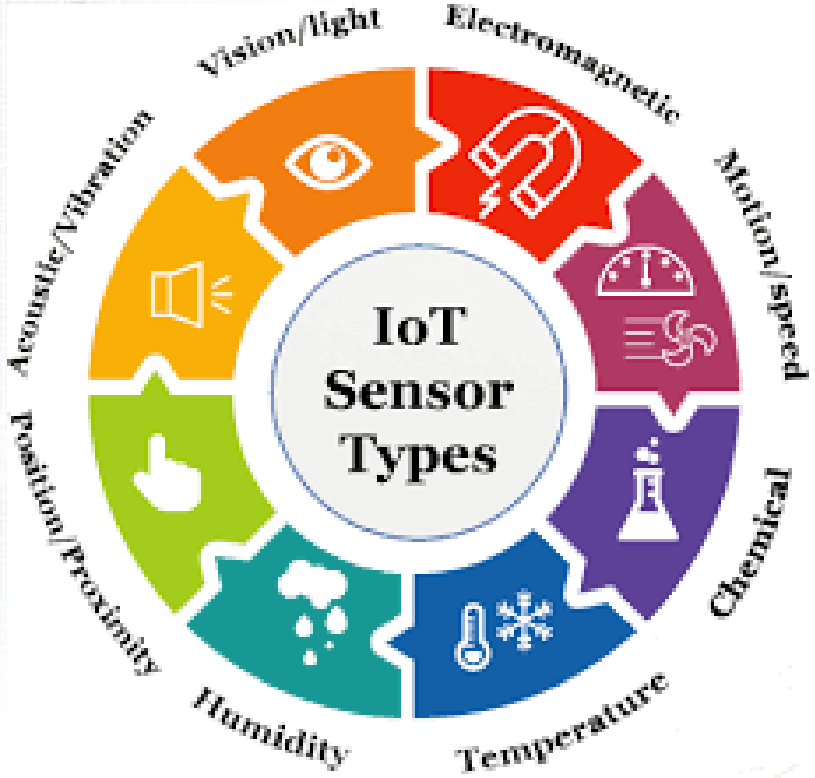
Sensor: Specifications & Resolutions

- ✓ **Specification:** Range of stimuli (Which is to be measured)
- ✓ The resolution of a sensor is the smallest change it can detect in the quantity that it is measuring.
- ✓ Resolution of a sensor indicates smallest output it can process at any time.
- ✓ The more is the resolution of a sensor, the more accurate is its precision.

Types of Sensors:



Types of Sensors:



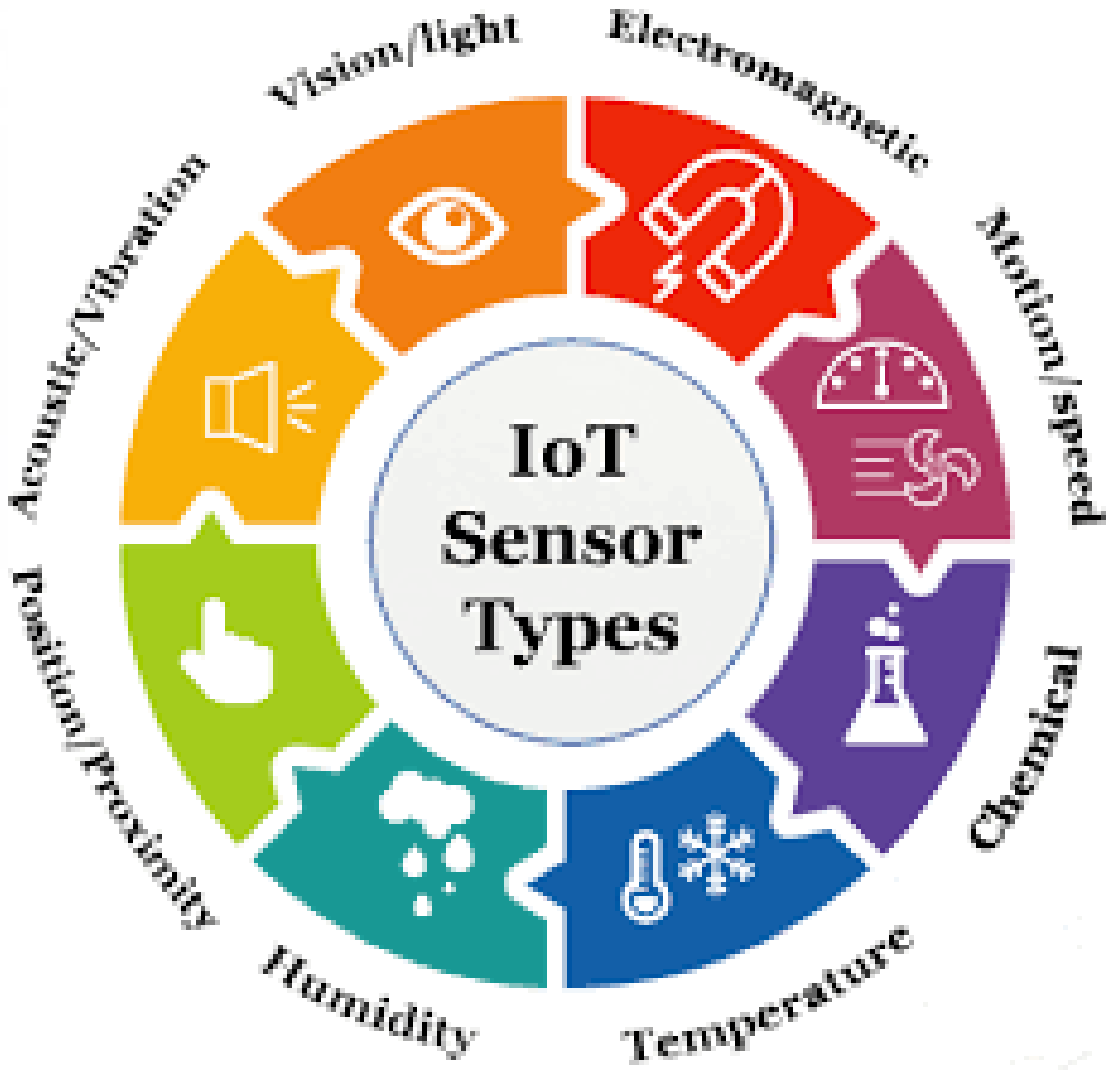
Based on Output

- Analog
- Digital

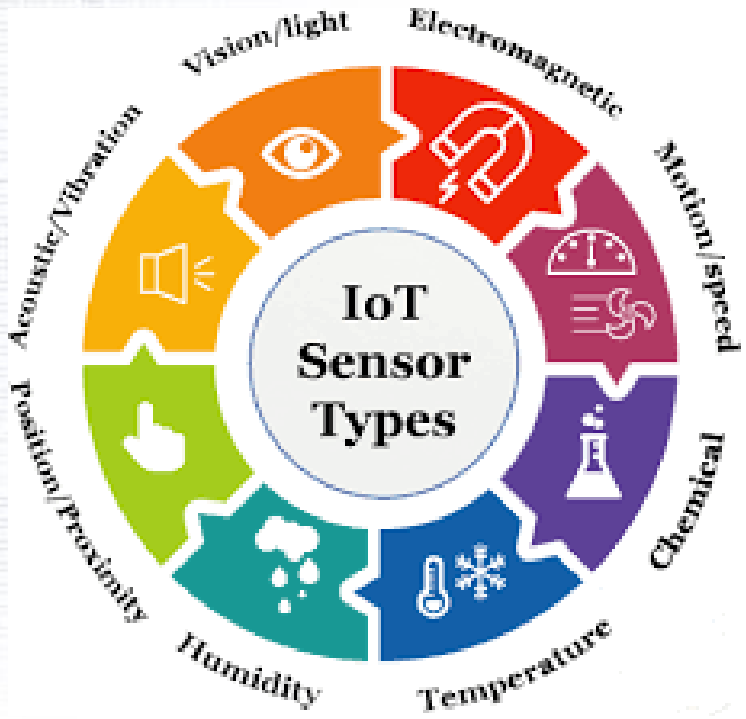
Based on Data type

- Scalar
- Vector/ Multimedia

Types of Sensors:
Based on quantity to be measured

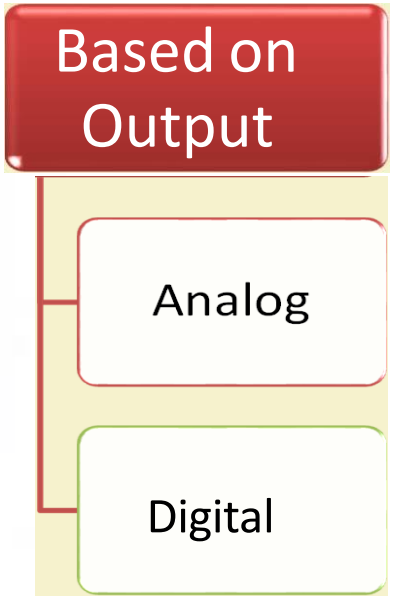


Types of Sensors:
Based on quantity to be measured

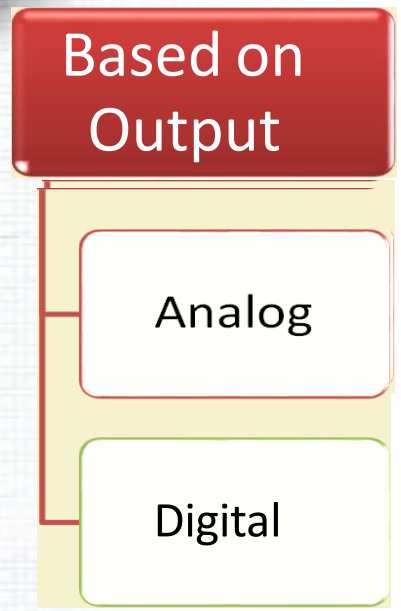


Light	• Light Dependent resistor • Photo-diode
Temperature	• Thermocouple • Thermistor
Force	• Strain gauge • Pressure switch
Position	• Potentiometer, Encoders • Opto-coupler
Speed	• Reflective/ Opto-coupler • Doppler effect sensor
Sound	• Carbon Microphone • Piezoelectric Crystal
Chemical	• Liquid Chemical sensor • Gaseous chemical sensor

Types of Sensors: Based on output

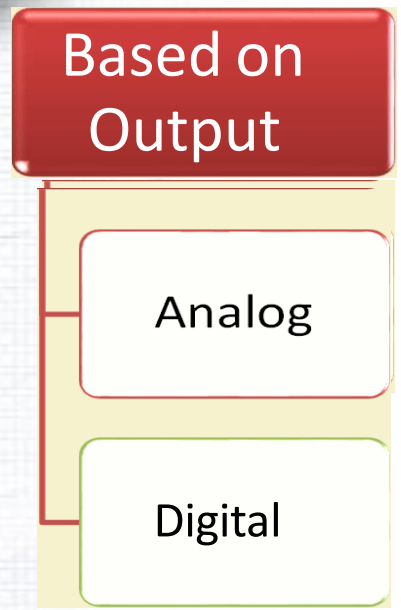


Types of Sensors: Analog Sensors



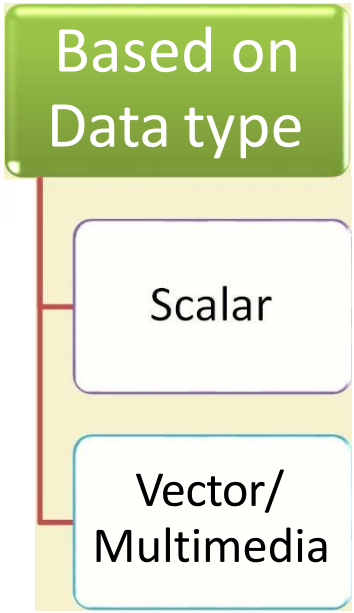
- Analog Sensors produce a continuous output signal or voltage which is generally proportional to the quantity being measured.
- Physical quantities such as Temperature, Speed, Pressure, Displacement, Strain etc. are all analog quantities as they tend to be continuous in nature.
- For example, the temperature of a liquid can be measured using a thermometer or thermocouple (e.g. in geysers) which continuously responds to temperature changes as the liquid is heated up or cooled down.

Types of Sensors:
Digital Sensors

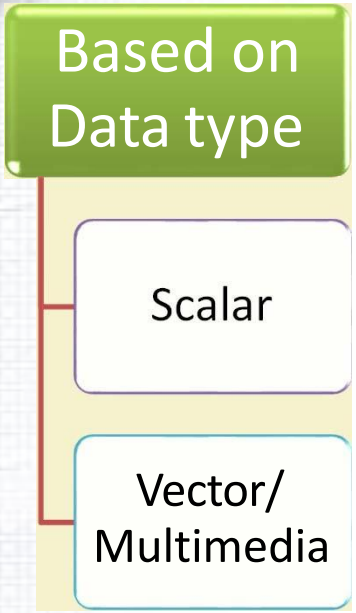


- Digital Sensors produce discrete digital output signals or voltages that are a digital representation of the quantity being measured.
- Digital sensors produce a binary output signal in the form of a logic "1" or a logic "0", ("ON" or "OFF").
- Digital signal only produces discrete (non-continuous) values, which may be output as a single "bit" (serial transmission), or by combining the bits to produce a single "byte" output (parallel transmission).

Types of Sensors: Based on Data Type

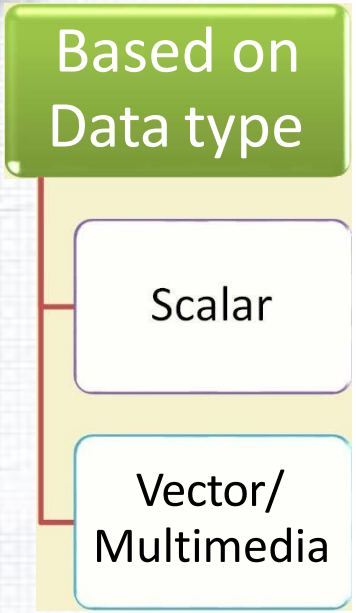


Types of Sensors: Scalar Sensors



- ✓ Scalar Sensors produce output signal or voltage which is generally proportional to the magnitude of the quantity being measured.
- ✓ Physical quantities such as temperature, color, pressure, strain, etc. are all scalar quantities as only their magnitude is sufficient to convey an information.
- ✓ For example, the temperature of a room can be measured using a thermometer or thermocouple, which responds to temperature changes irrespective of the orientation of the sensor or its direction.

Types of Sensors: Vector Sensors



- ✓ Vector Sensors produce output signal or voltage which is generally proportional to the magnitude, direction, as well as the orientation of the quantity being measured.
- ✓ Physical quantities such as sound, image, velocity, acceleration, orientation, etc. are all vector quantities, as only their magnitude is not sufficient to convey the complete information.
- ✓ For example, the acceleration of a body can be measured using an accelerometer, which gives the components of acceleration of the body with respect to the x,y,z coordinate axes.

Errors in Sensors

- **Sensitivity Error:** The output of a sensor under real conditions may differ from the actual value. This is called a sensitivity error.
- **Linear Error:** If the output signal differs from the correct value by a constant, the sensor has an offset error or bias.
- **Nonlinear Error:** Nonlinearity is deviation of a sensor's transfer function (TF) from a straight line transfer function.
- Most sensors have linear behavior.
- **Drift:** If the output signal slowly changes independent of the measured property, this is defined as drift. Long term drift over months or years is caused by physical changes in the sensor.
- **Noise** is a random deviation of the signal that varies in time.
- **Hysteresis error** causes the sensor output value to vary depending on the sensor's previous input values.
- The present reading depends on the past input values.

Definition of Sensor

- **Sensor** is a device that detects or measures a physical property and records, indicates, or otherwise responds to it.
- A **sensor** is a device that measures **physical input** from its environment and converts it into data that can be **interpreted by either a human or a machine**.
- Normally, the sensors are input devices and there are two types of sensors.
  - **Analog Sensors**
  - **Digital Sensors**

List of Sensors

- The list of sensors which is to be covered in the chapter is as follows:

MQ-02/05 Gas Sensor

Obstacle Sensor

Ultrasonic Distance Sensor

LDR Sensor

Heartbeat Sensor

Gyro Sensor

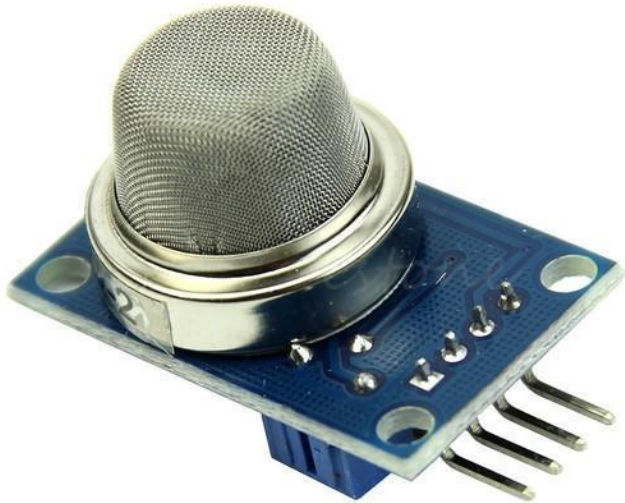
GPS Sensor

Color Sensor

pH Sensor

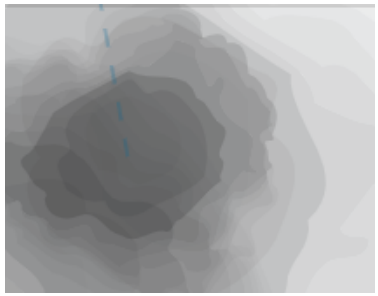
MQ-02/05 Gas Sensor

- MQ -02 sensor is used to **detect smoke.**
- H2, LPG, CH4 alcohol and smoke can be detected by MQ-02.
- MQ-05 **is not sensitive to smoke**, hence less used in the application.
- Pinout of MQ-02 sensor module are
  - Vcc – Connected to 5V
  - Gnd – Connected to ground
  - D0 – Digital output pin
  - A0 – Analog output pin



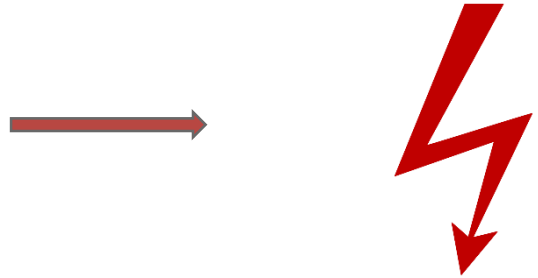
MQ-02/05 Gas Sensor – Working principle

- When any flammable gas passes through the coil of the sensor, the coil burns and **internal resistance decreases.**
- This results in an **increased voltage** across it. Hence we get variable voltage at A0 pin of MQ-05 gas sensor.
- If gas present -> voltage is high.
- If gas is not present -> voltage is low.



Smoke/Gas

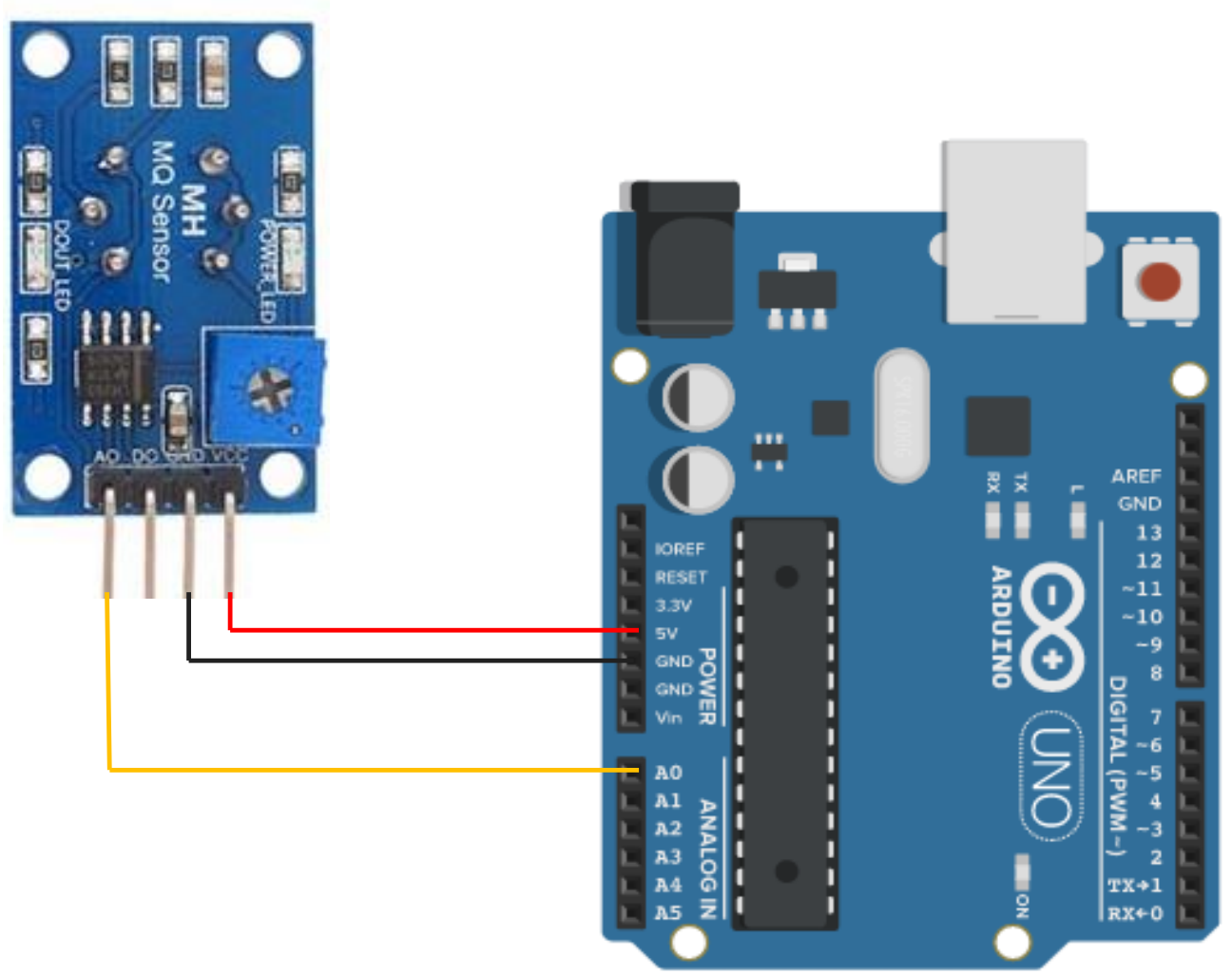
MQ-05 Gas Sensor



Variable Voltage

MQ-02/05 Gas Sensor – Interfacing with Arduino

- The interfacing of MQ-02 with Arduino Uno is as shown in the figure.
- Here, Vcc and Gnd pins of MQ-02 are connected to 5V and GND of Arduino board.
- We want to take the input from a Gas sensor in **an analog** format so pin A0 is connected to one of the analog pins of Arduino i.e. A0.



MQ-02/05 Gas Sensor – Code explanation

- The code in Arduino for MQ-02 gas sensor can be written as following

gas_sensor.ino

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

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

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

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

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

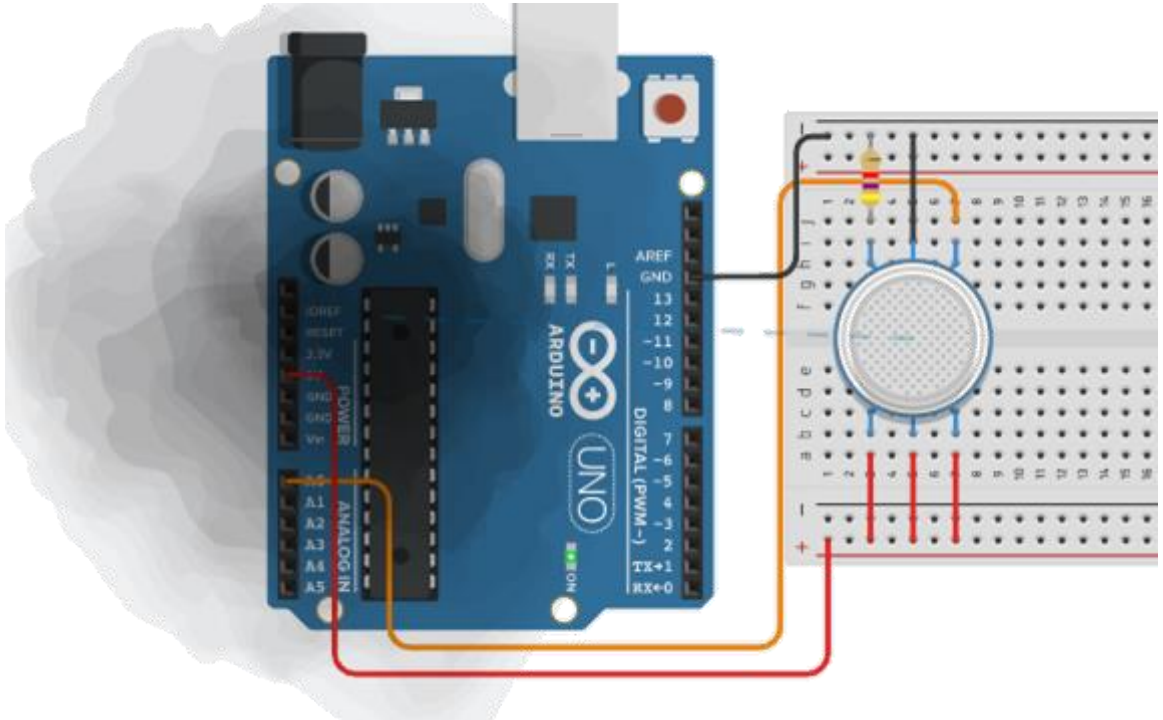
Functions in Arduino IDE

- Serial.begin( ): It is used to start serial communication and set the speed of serial communication between Arduino board and other device. Normally, it is connected with computer for monitoring the data. Serial communication uses pin 0 and 1 also working as Rx and Tx respectively.
  - Syntax : Serial.begin(baud_rate)
  - Parameters :
    - baud_rate: It defines the speed of data transfer rate between Arduino and other device. It is compulsory to set this baud rate same at both the side for proper transfer of data.
- Serial.print( ) : Print the data from Arduino to other device.
  - Syntax : Serial.print("text")
  - Parameters :
    - The text which is to be printed by Arduino board is written in double quote so as to convert it in its ASCII values.
- Serial.println( ) : Print the data with newline from Arduino to other device.
  - Syntax : Serial.println("text")
  - Parameters :
    - The text which is to be printed by Arduino board is written in double quote so as to convert it in its ASCII values.

Functions in Arduino IDE (Cont.)

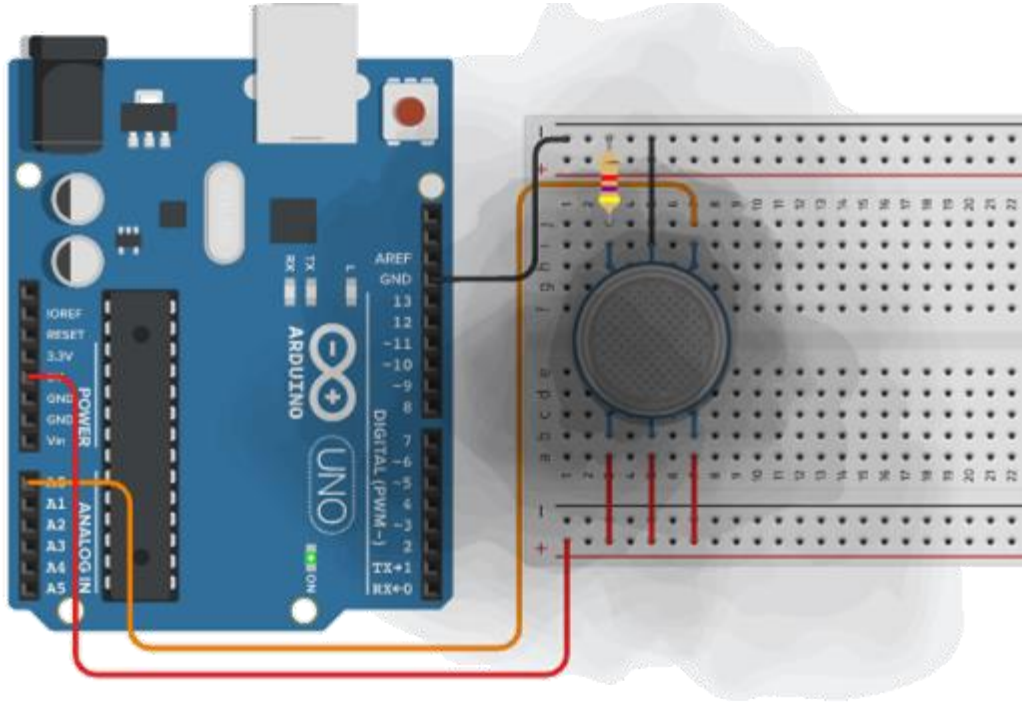
- analogRead( ): The syntax reads analog value from specified pin and converts it into 1024 levels or in 0 to 1023 range. The values is stored into specified variable in the code statement. The value is converted into 1024 levels because Arduino has 10 bit built in A-D converter unit.
  - Syntax : analogRead(pin)
  - Parameters :
    - pin: The Arduino analog pin number.

MQ-02/05 Gas Sensor – Output



Serial Monitor

```
Gas Level : 306
Gas Level : 306
Gas Level : 306
Gas Level : 306
Gas Level : 306
Gas Level : 306
```

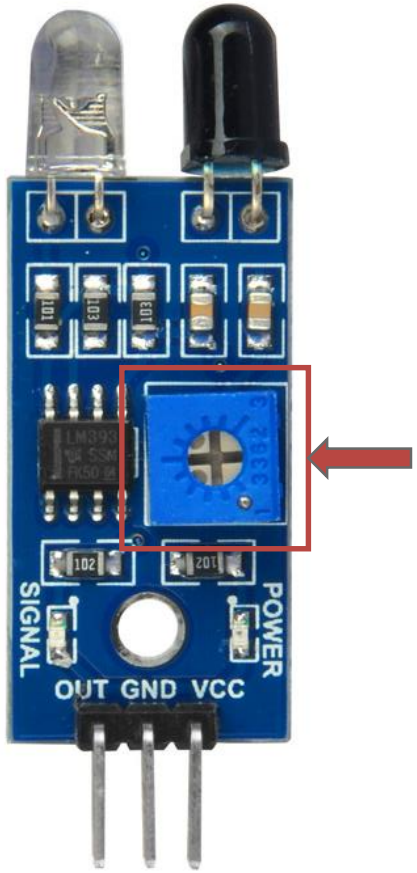


Serial Monitor

```
Gas Level : 745
Gas Level : 745
Gas Level : 745
Gas Level : 745
Gas Level : 745
Gas Level : 745
```

Obstacle Sensor

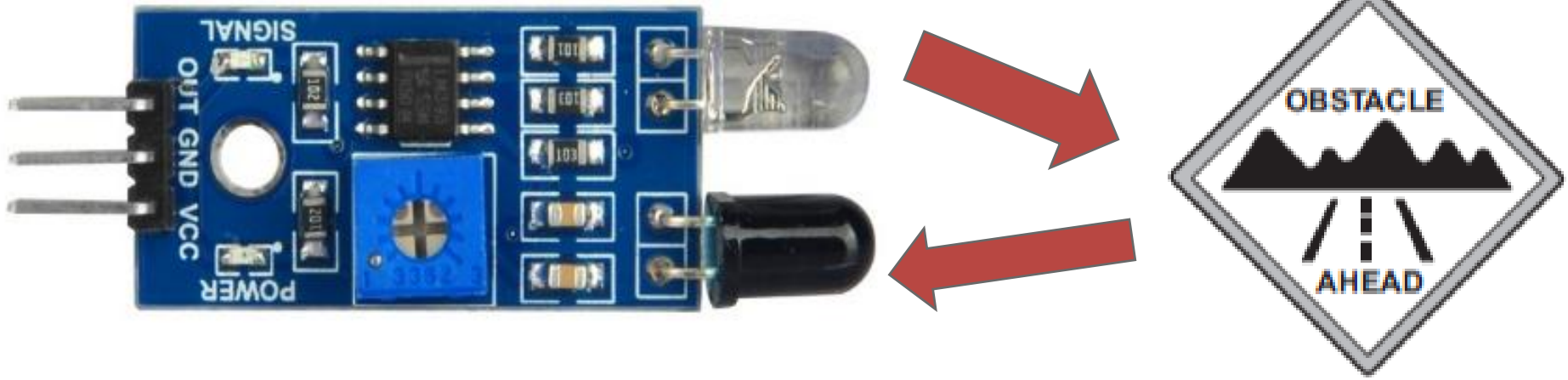
- Obstacle sensor is used for **detection of obstacle**.
- It is a digital sensor, hence gives binary output ‘1’ or ‘0’.
- The **range** for detection of obstacle can be changed by **potentiometer** given.
- Sensitivity up to **30cm adjustable**.
- Pinout of obstacle sensor module are
  - Vcc – Connected to 5V
  - Gnd – Connected to ground
  - Out/D0 – Digital output pin



Obstacle Sensor – Working principle

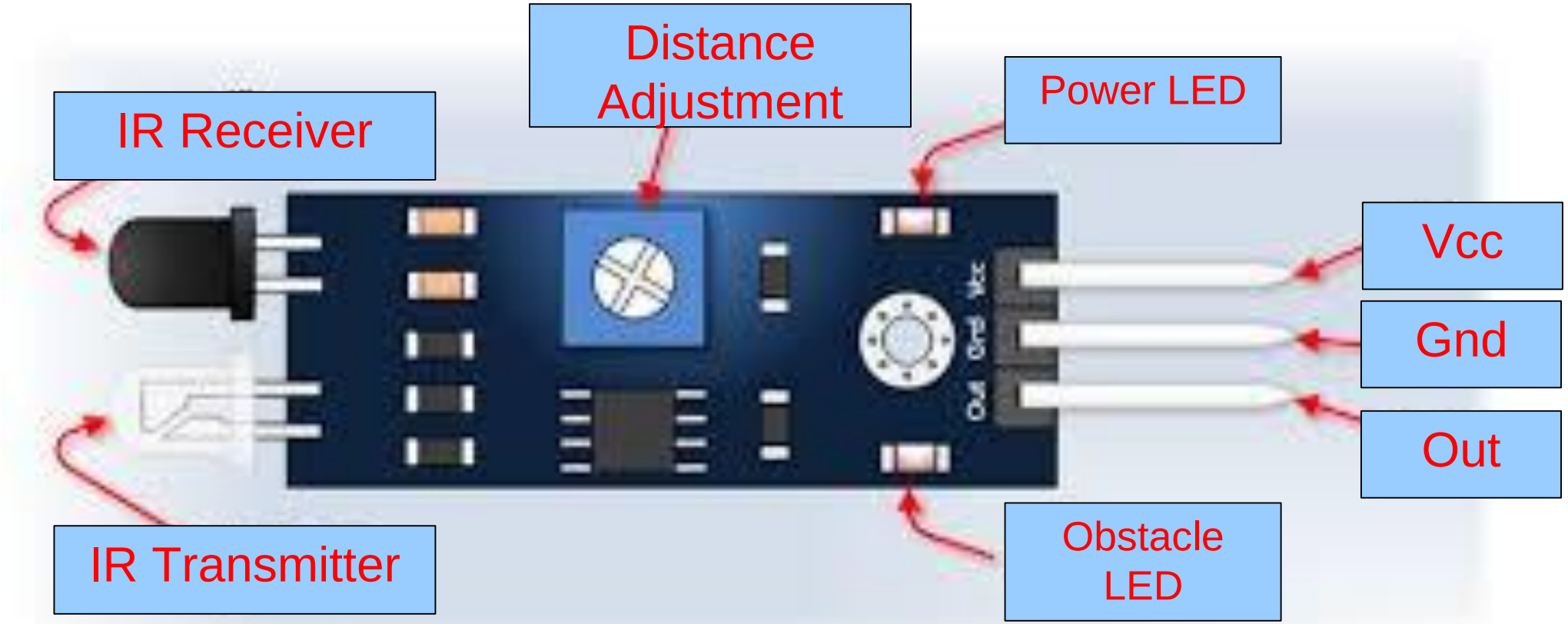
- IR emitter transmits IR signals and IR receiver receives those signals from reflection.
- If the obstacle is present then the transmitted signals are **reflected** back by obstacle and if they have amplitude greater then threshold the output signal will be **'0'**.
- If the obstacle is not present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

Digital Output



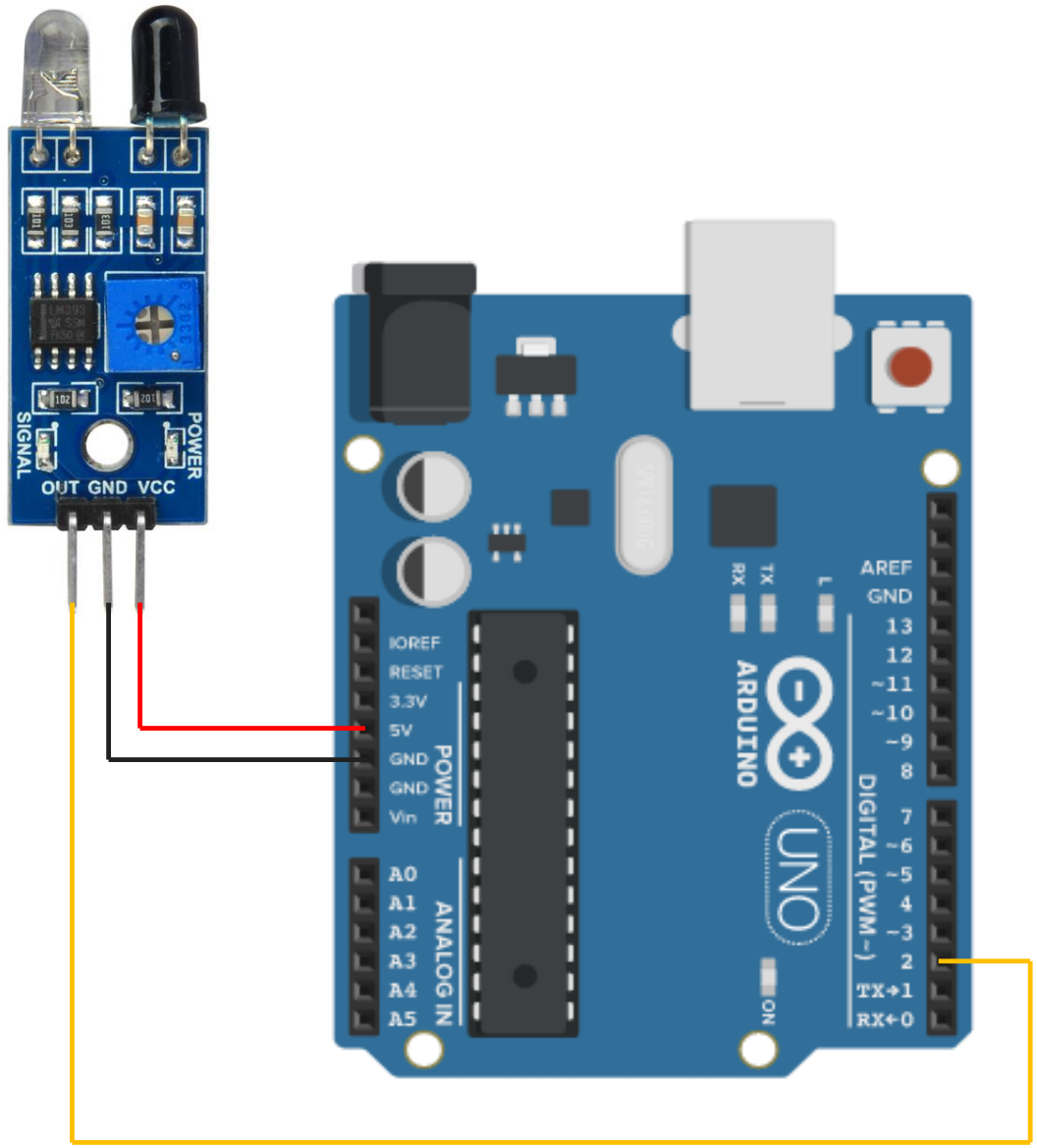
Obstacle IR Sensor

Obstacle Sensor



Obstacle Sensor – Interfacing with Arduino

- The interfacing of obstacle sensor with Arduino Uno is as shown in figure.
- Here, Vcc and Gnd pins of obstacle sensor is connected to 5V and Gnd of Arduino board.
- The output of obstacle sensor is in **digital** form. So it is connected to digital pin 2 of Arduino.



Obstacle Sensor – Code explanation

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

obstacle-detect.ino

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

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

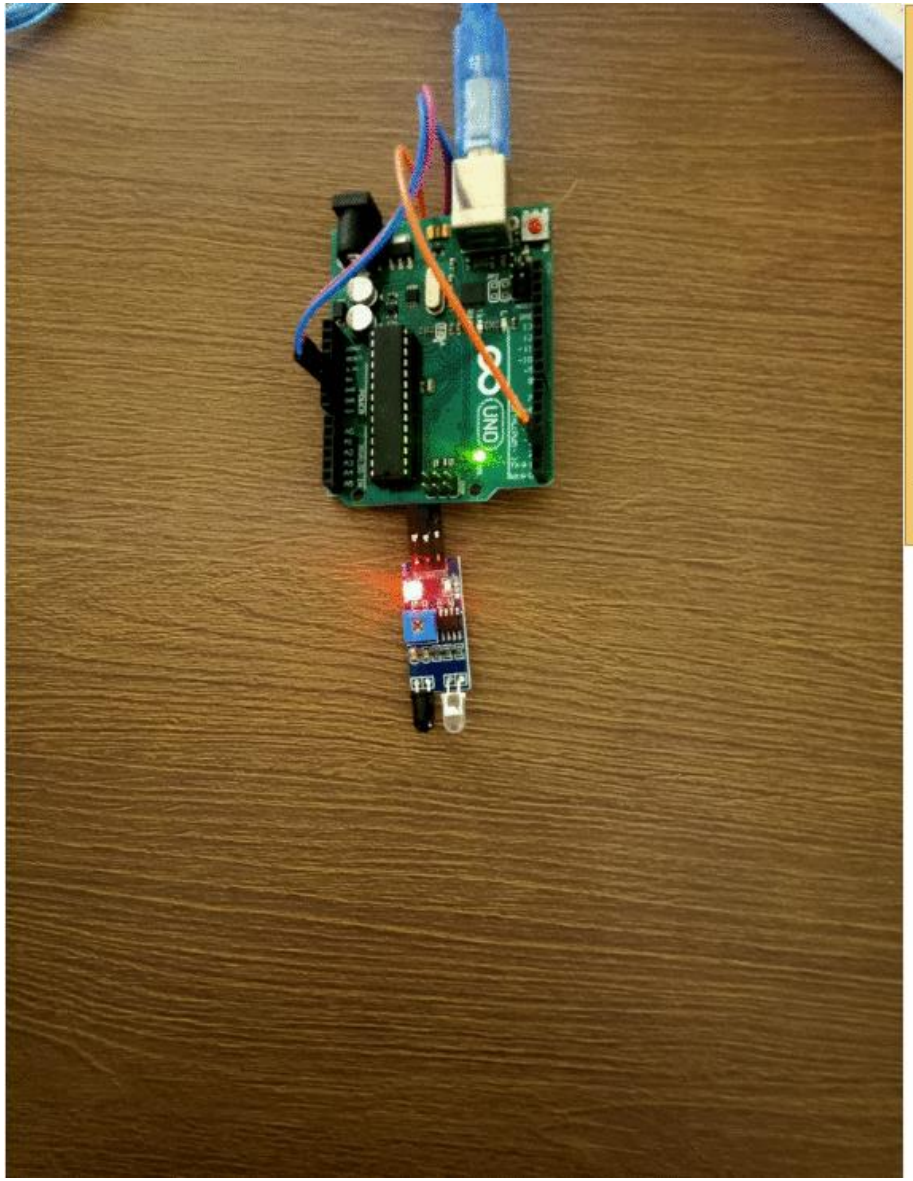
Obstacle Sensor – Code explanation (Cont.)

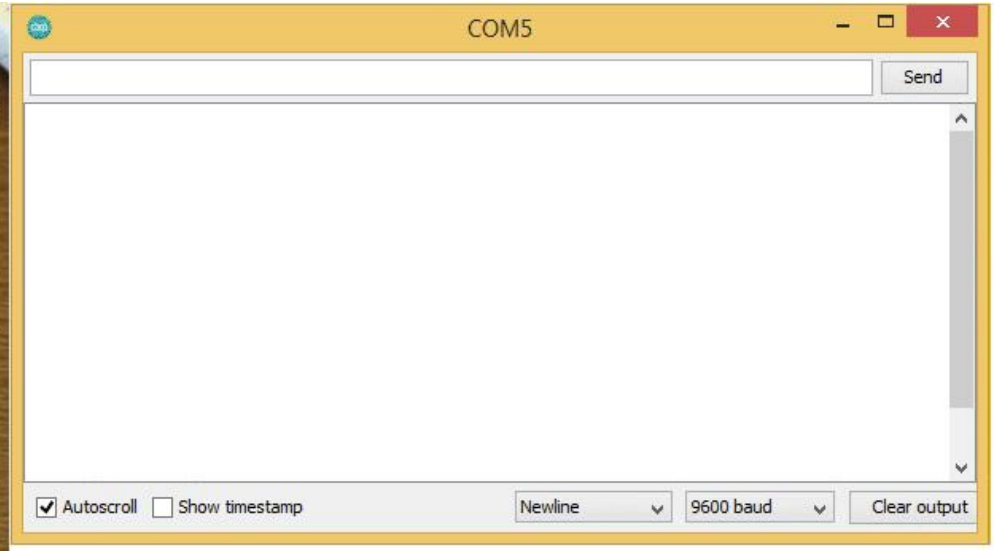
obstacle-detect.ino

```
13    else
14    {
15       digitalWrite(13,LOW);
16    }
17    delay(1000);
18 }
```

- digitalRead( ) : Reads digital data from specified pin and stores it into given variable.
  - Syntax : digitalRead(pin)
  - Parameters :
    - Pin : The Arduino digital pin number.

Obstacle Sensor – Output



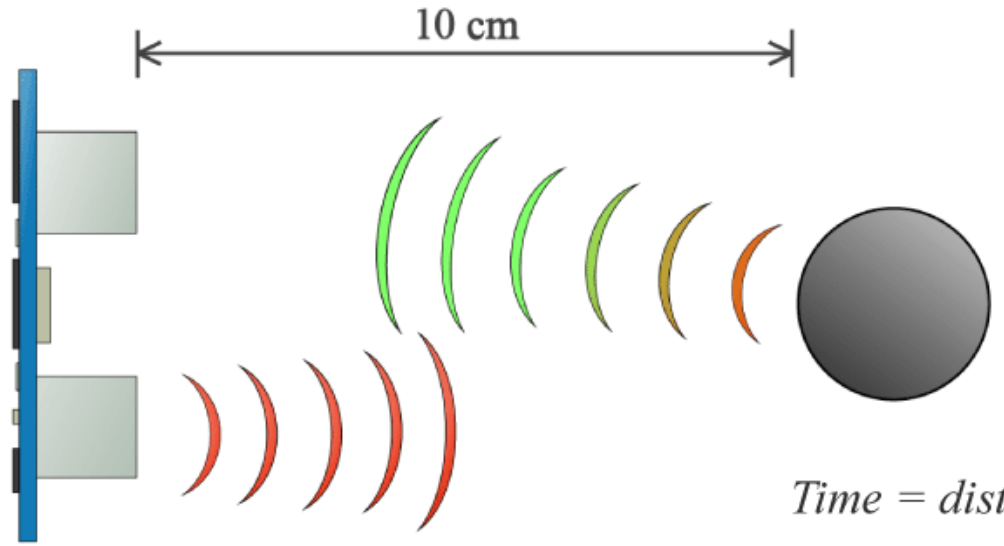


HC-SR04 Ultrasonic Sound Sensor

- Ultrasonic Sound Sensor is used to measure the **distance of an object (2 cm – 500 cm).**
- The sensor can only measure the distance of object. It can not identify the type of object.
- It is also known as Ultrasonic distance sensor.
- Ultrasonic Frequency : 40 kHz
- Pinout of HC-SR04 module are
  - Vcc – Connected to 5V
  - Gnd – Connected to ground
  - Trigger – Generates the transmit pulse
  - Echo – Receives echo from obstacle

HC-SR04 Ultrasonic Sound Sensor – Working principle

- The emitter transmits ultrasonic sound waves which are reflected by near by objects.
- The reflected pulse is received by sensor.
- Some mathematical operations are performed to obtain the distance value.
- The distance value is converted into desired unit.



Ultrasonic waves

speed of sound:

$v = 340\ m/s$

$v = 0{,}034\ cm/\mu s$

Time = distance / speed:

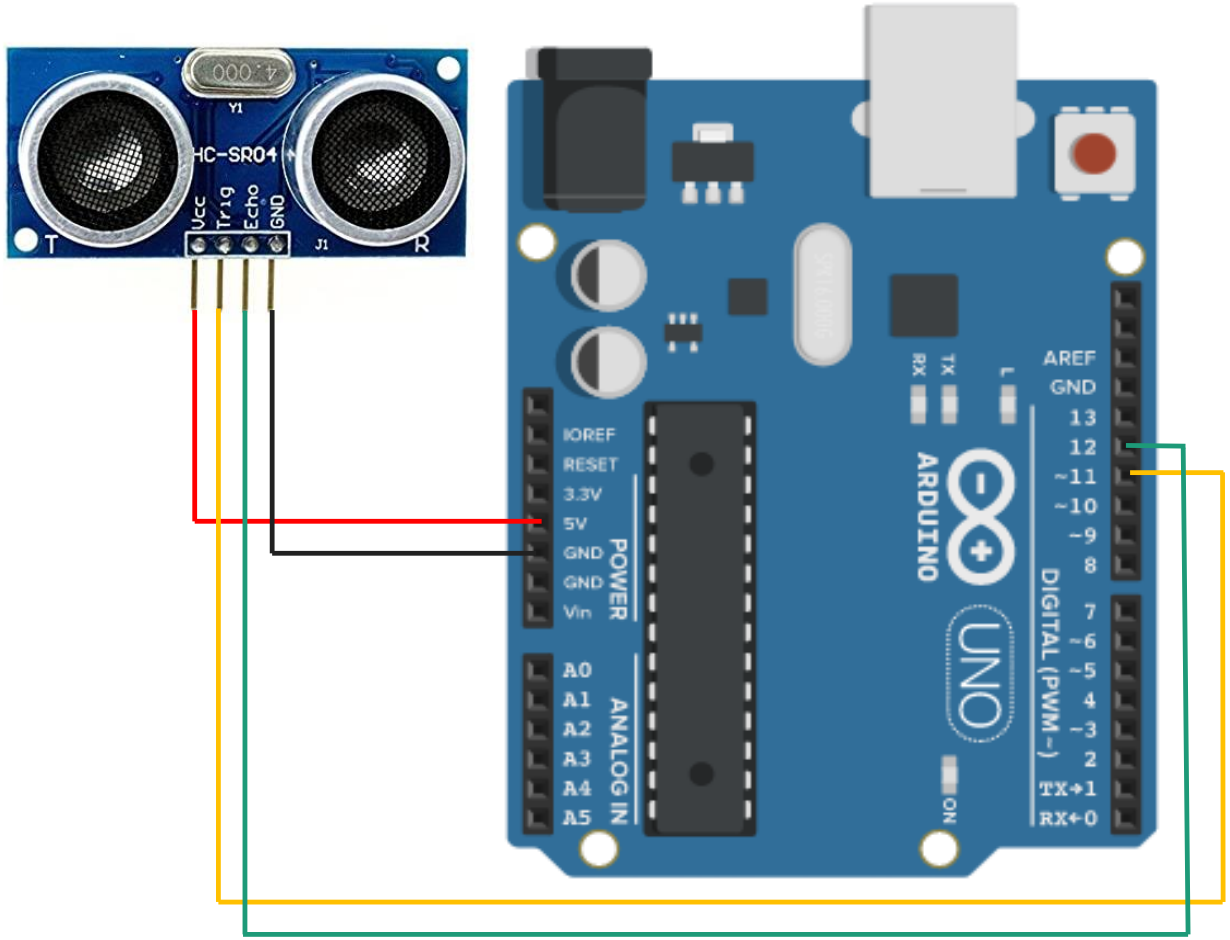
$t = s / v = 10 / 0{,}034 = 294\ \mu s$

Distance:

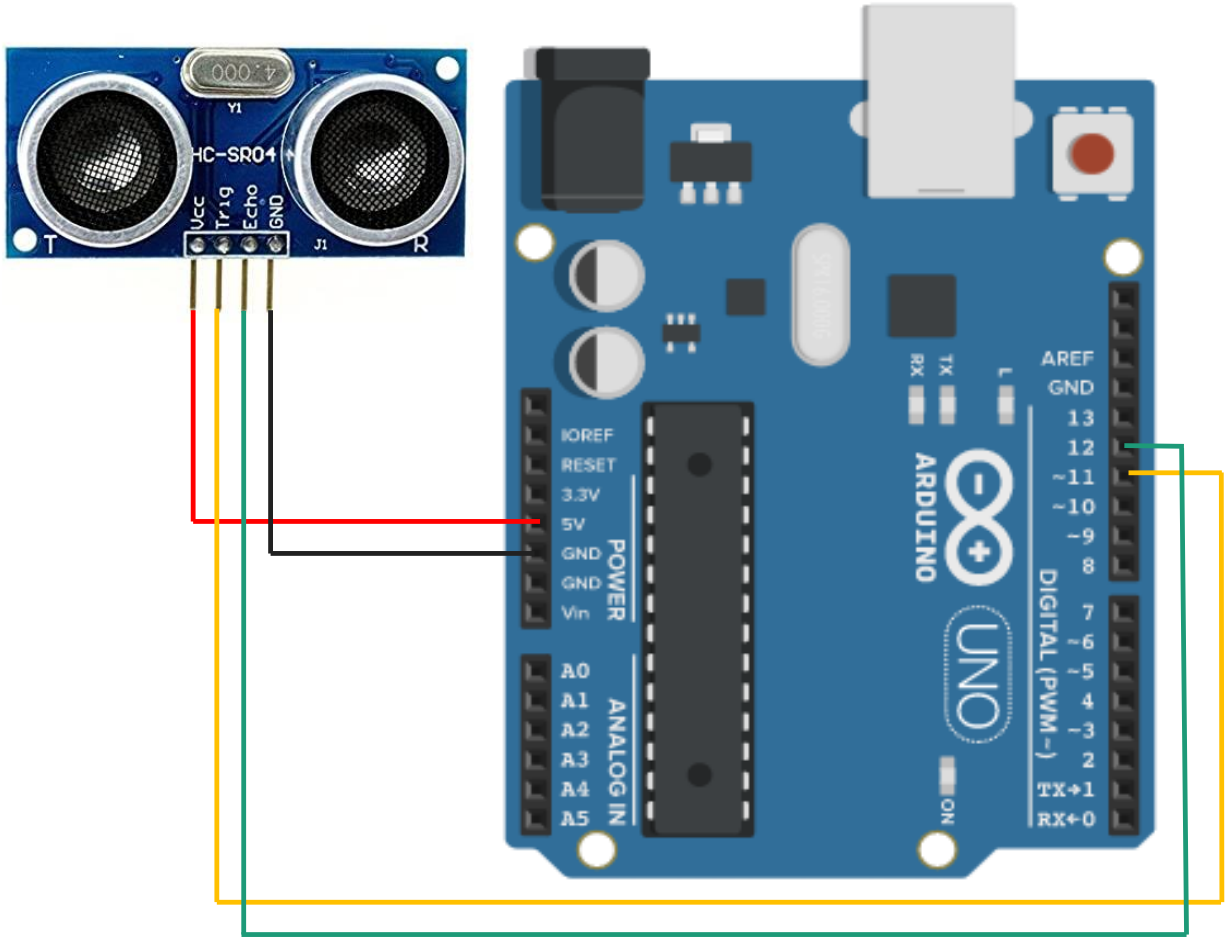
$s = t \cdot 0{,}034 / 2$

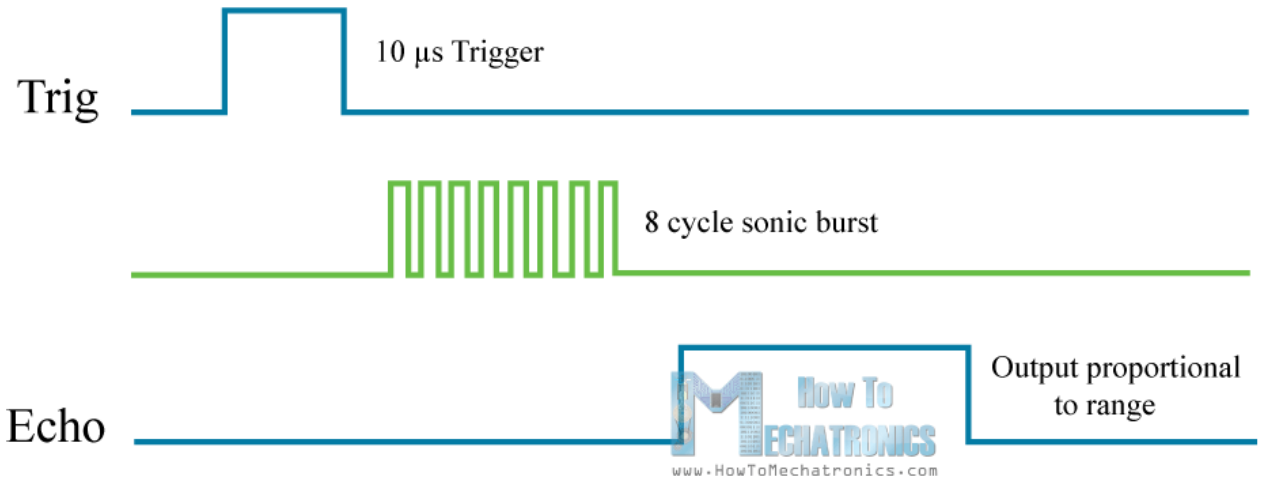
HC-SR04 Ultrasonic Sound Sensor– Interfacing with Arduino

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



HC-SR04 Ultrasonic Sound Sensor– Interfacing with Arduino





HC-SR04 Ultrasonic Sound Sensor – Code explanation

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

Distance-measure.ino

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

Generates the delay of specified microseconds.

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

Distance-measure.ino

```
16    pinMode(echoPin, INPUT);
17    duration = pulseIn(echoPin, HIGH);
18
19    // Convert the time into a distance
20    cm = (duration/2) * 0.0343;
21    inches = (duration/2) / 74;
22    Serial.print(inches);
23    Serial.print("in, ");
24    Serial.print(cm);
25    Serial.print("cm");
26    Serial.println();
27
28    delay(250);
29  }
```

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

Distance(m) = Speed (m/s)x Time (s)

But, Time of pulse we receive is in µs and distance we want to find is in cm.
Distance(cm) = Speed (cm/µs) x Time (µs)

Speed of sound is 343m/s which is converted into 0.0343 cm/µs.
So either we have to multiply 0.0343 or divide 29.1

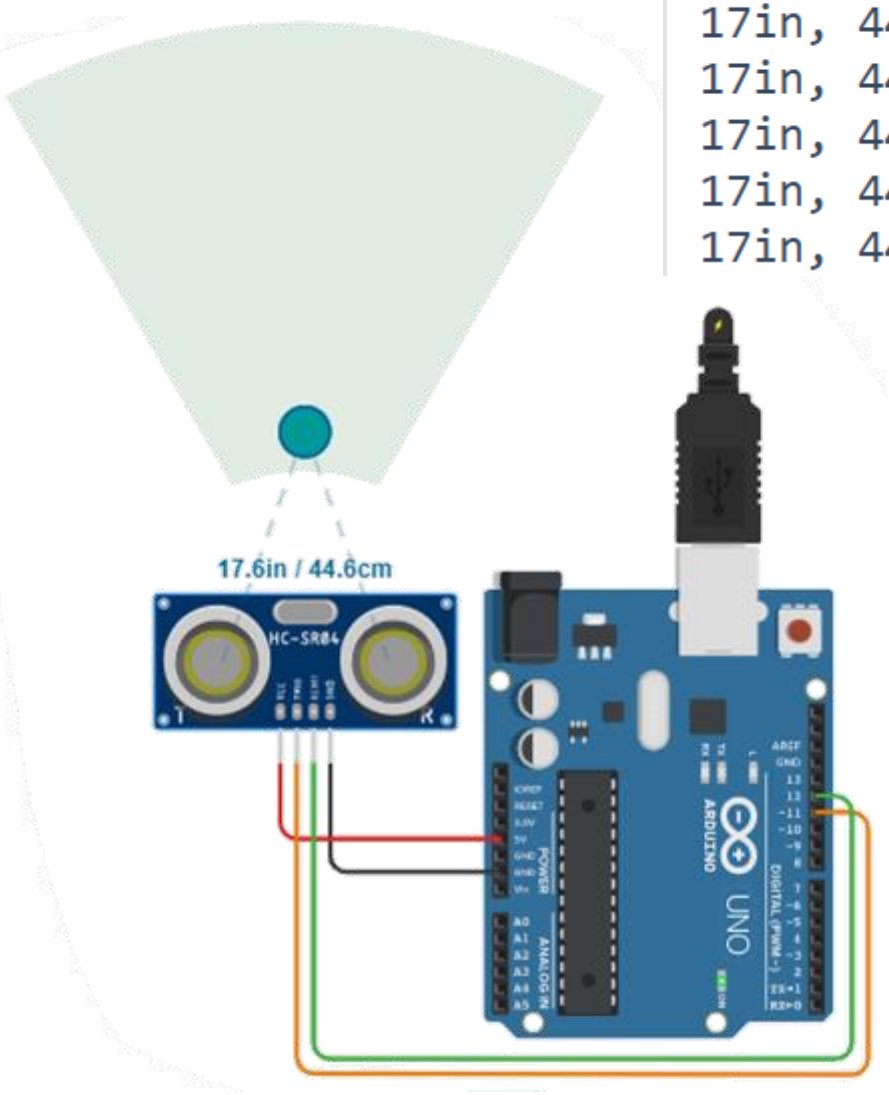
Functions in Arduino IDE

- delayMicroseconds( ): It pauses the program for amount of time in microseconds specified as the parameter.
  - Syntax : delayMicroseconds(μs)
  - Parameters :
    - μs : The number of microseconds to pause.
- pulseIn( ) : Reads a pulse HIGH or LOW from specified pin and wait for the time to go the pulse from HIGH to LOW or LOW to HIGH. It returns the time required to transit the pulse in variable.
  - Syntax : pulseIn(pin, value)
  - Parameters :
    - Pin: The number of the Arduino pin on which you want to read the pulse.
    - Value: The type of pulse that you want to read.

HC-SR04 Ultrasonic Sound Sensor – Output

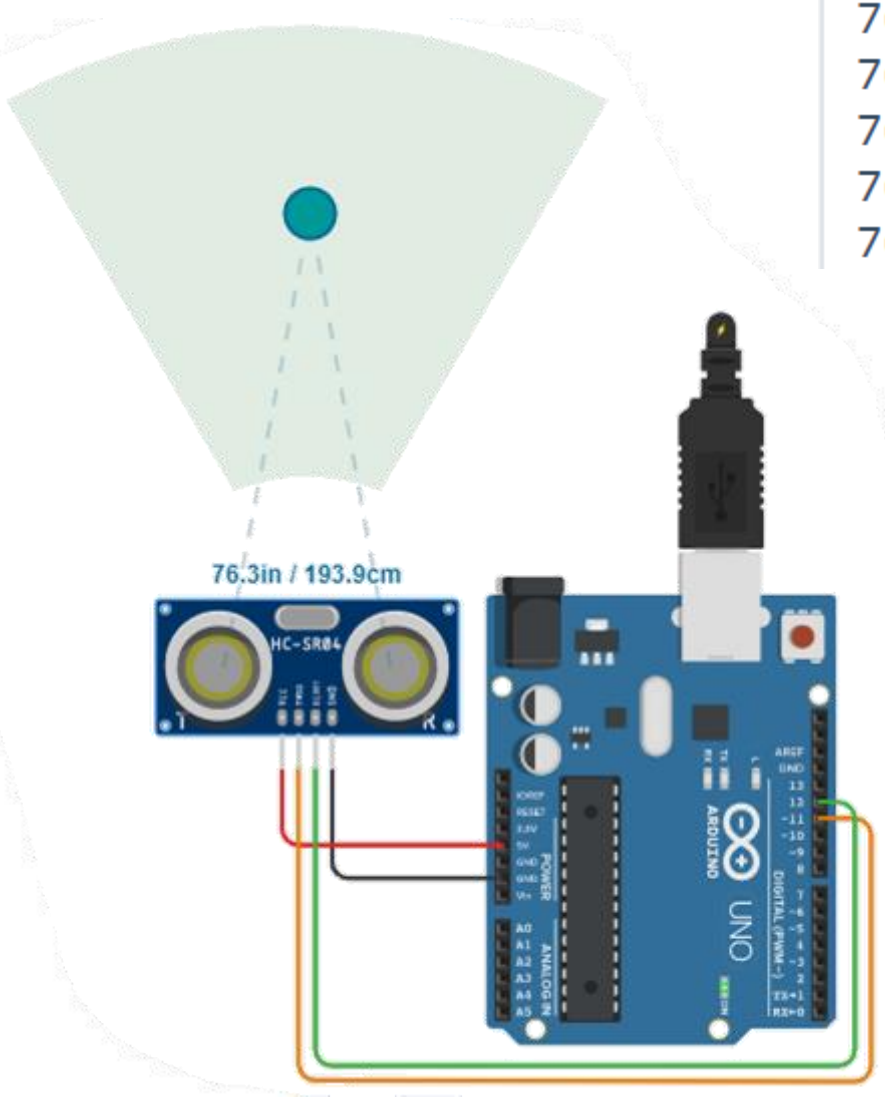
Serial Monitor

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

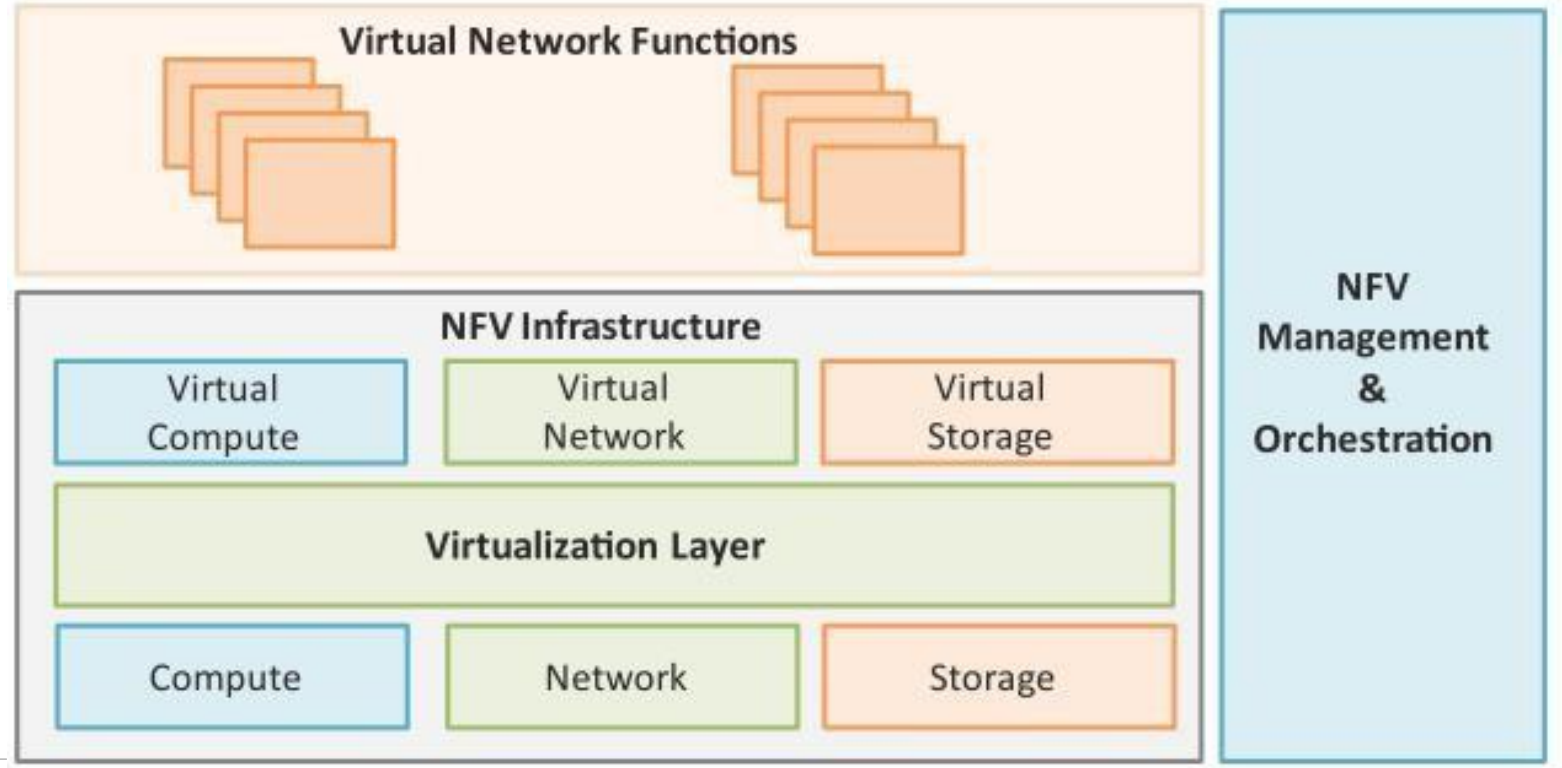


Serial Monitor

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



Unit 2:
Network Function Virtualization



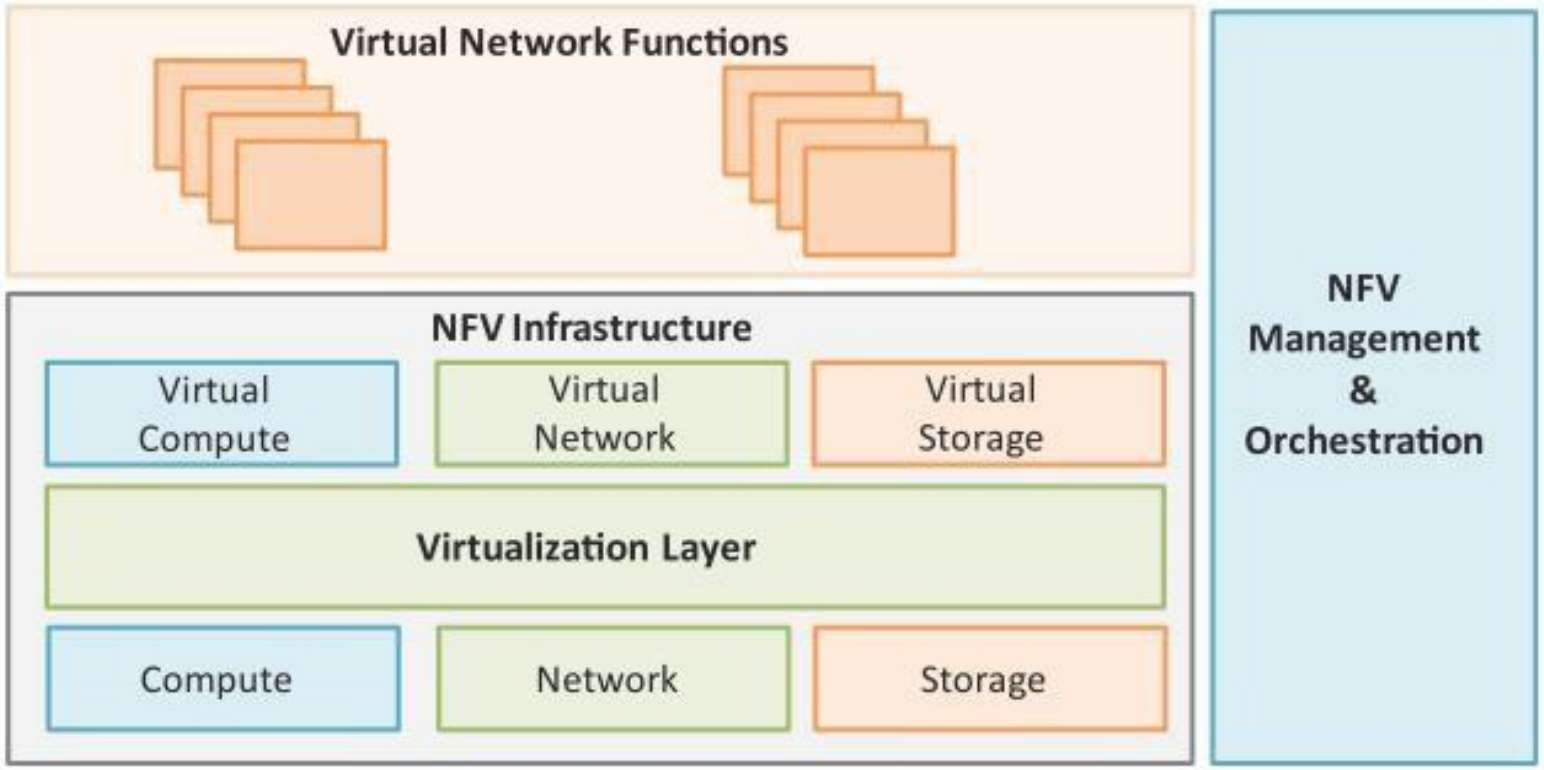
National Forensic Sciences University
Knowledge | Wisdom | Fulfilment
An Institution of National Importance
(Ministry of Home Affairs, Government of India)

Dr. Ujjaval Patel

Assistant Professor (IOT/SCADA)

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.



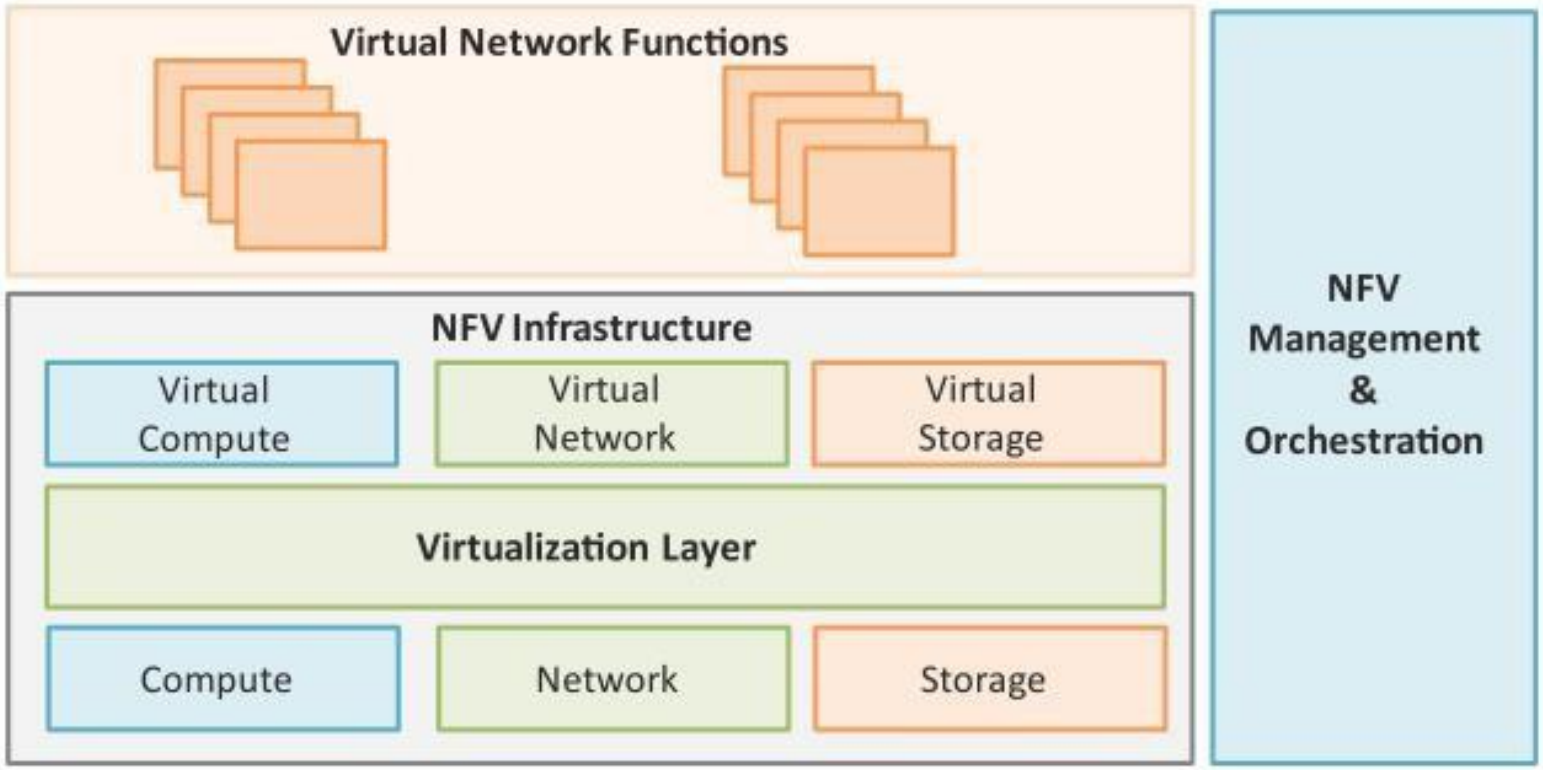
Key elements of NFV:

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.

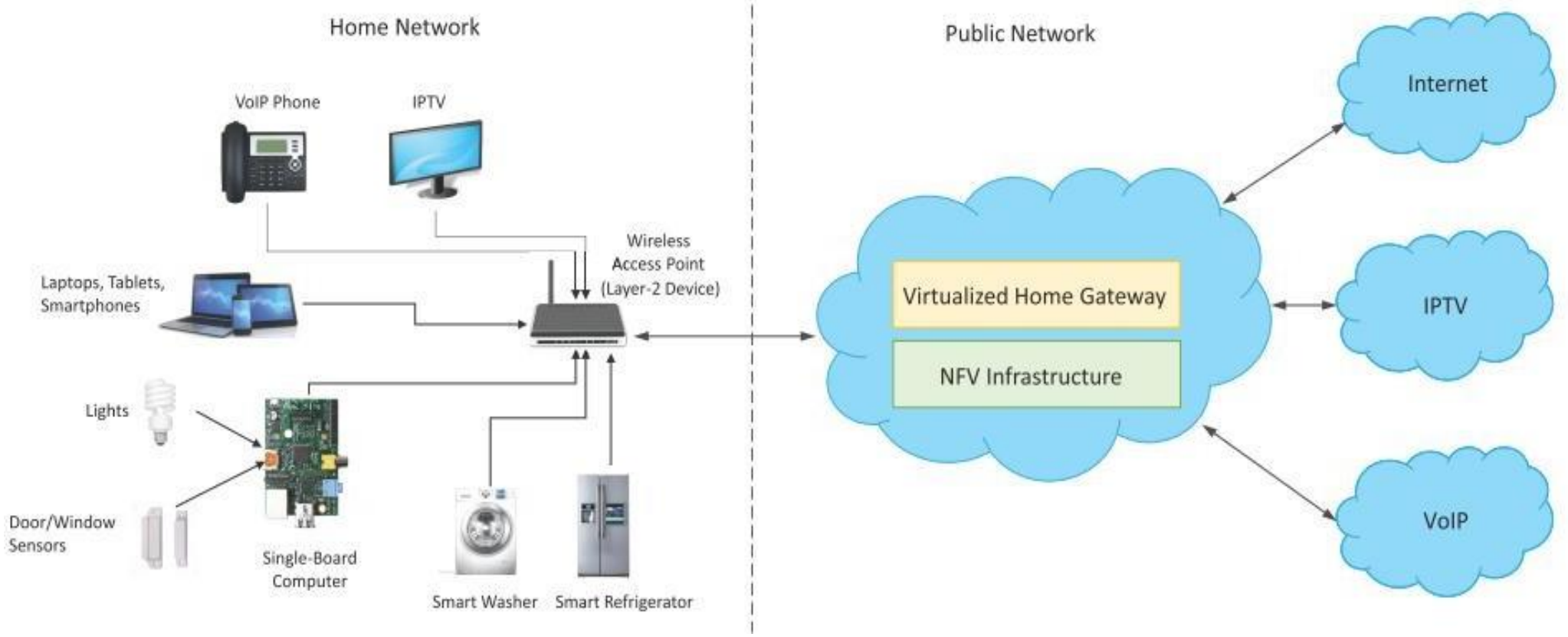


NFV Management and Orchestration:

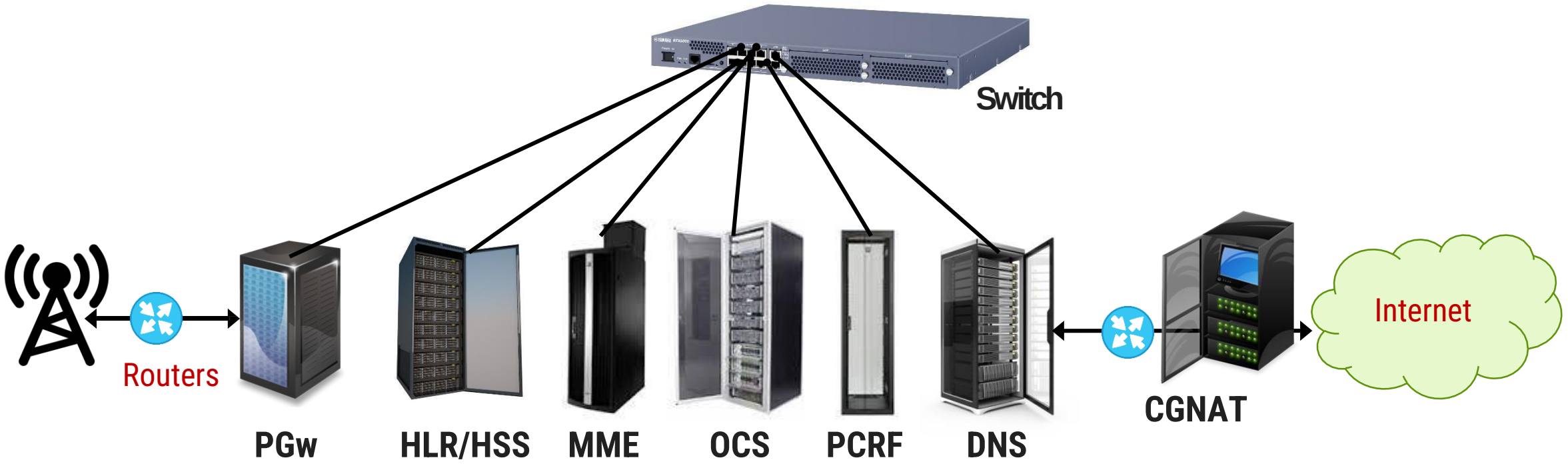
✓ NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

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.



NFV : Network function Virtualization

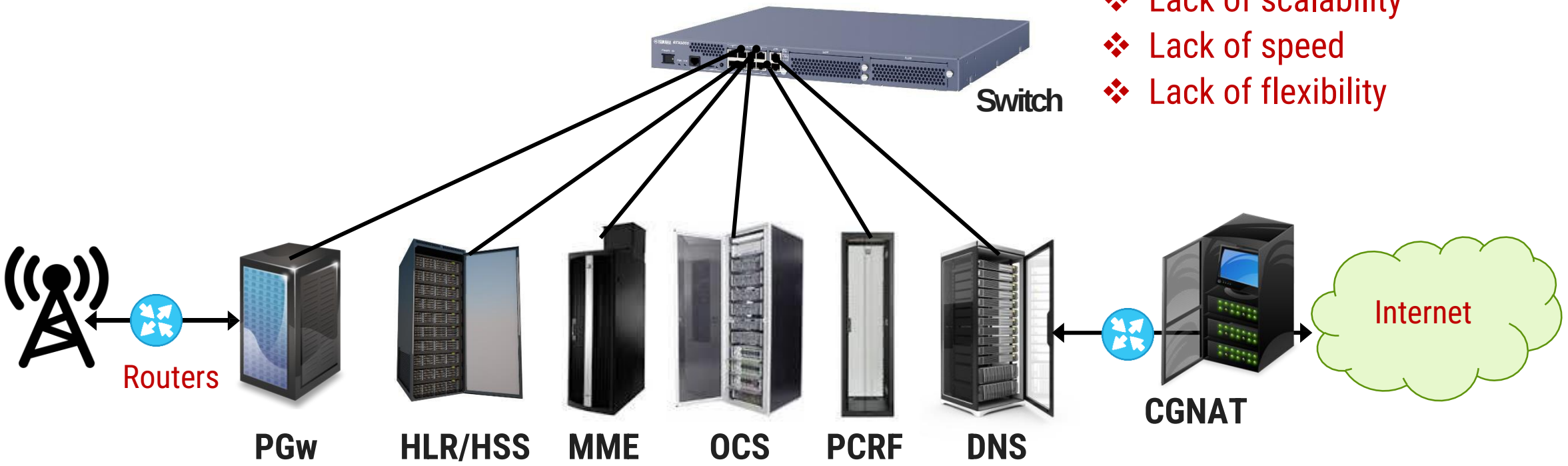


Specialized Appliances / Purpose Build
Combination of Capacity & Functionality

Traditional LTE Network

NFV : Network function Virtualization

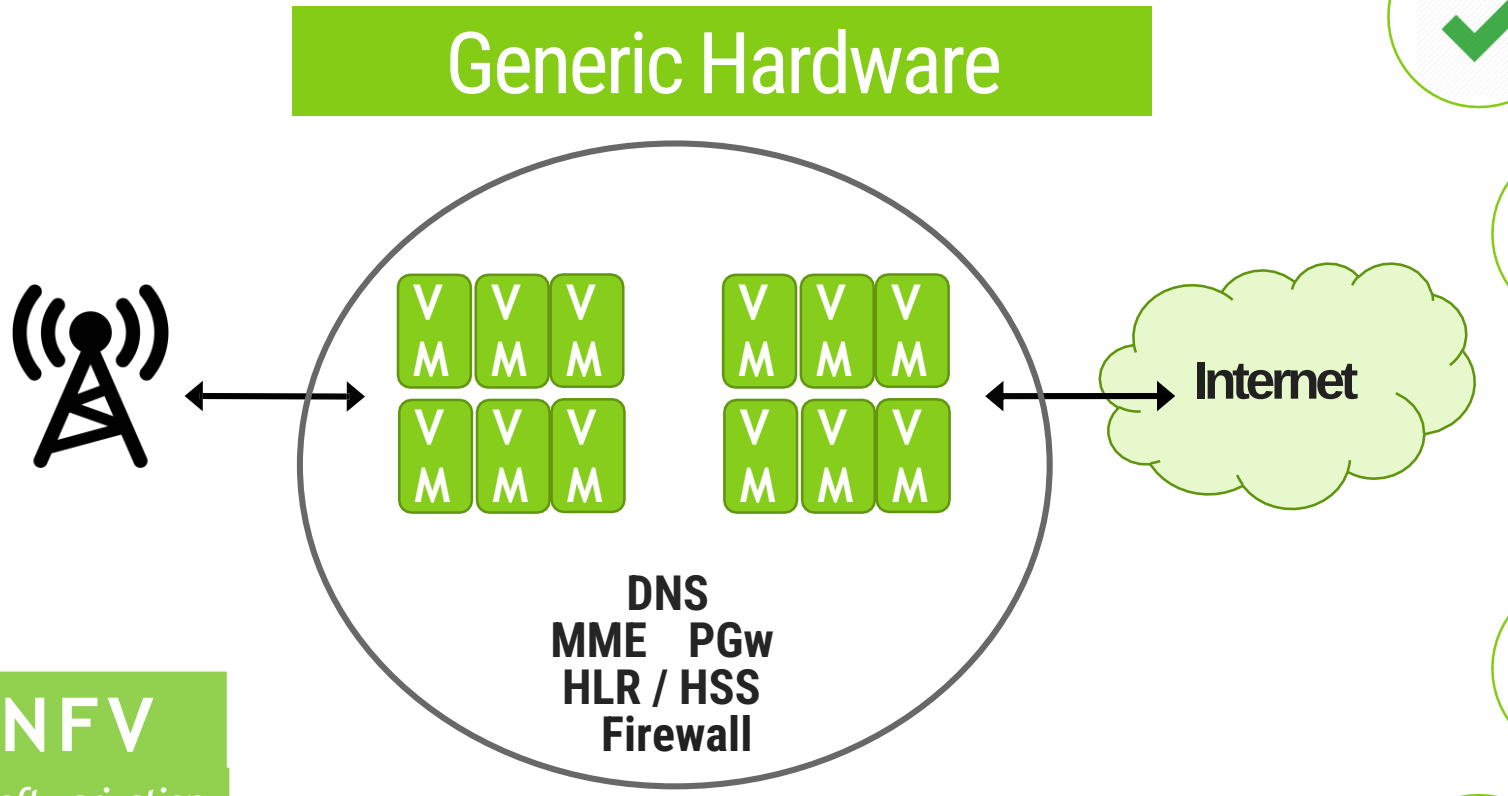
- One node, one set of function and one typical application
- Lack of scalability
- Lack of speed
- Lack of flexibility



Specialized Appliances / Purpose Build
Combination of Capacity & Functionality

NFV : Network function Virtualization

Virtual Network



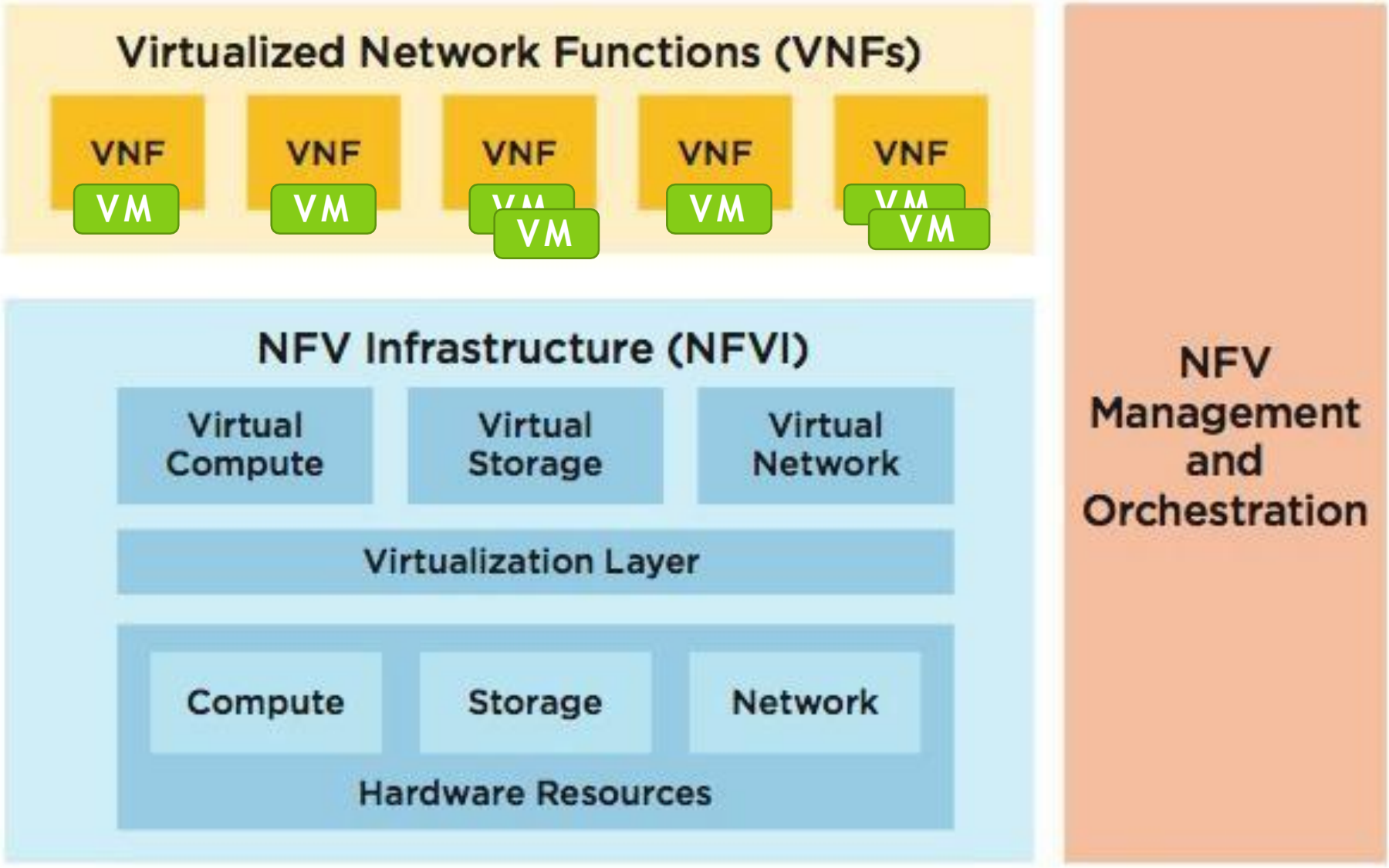
NFV

- *Softwarization*
- *Virtualization*
- *Orchestration*

Features of NFV

- Purpose Build hardware to Generic Hardware
- App running on Software
- Separation of Network Function & Capacity
- Easy Capacity Scale Up / Down
- VM are Building Blocks

NFV Architecture



NFV Layers

Governing Specs

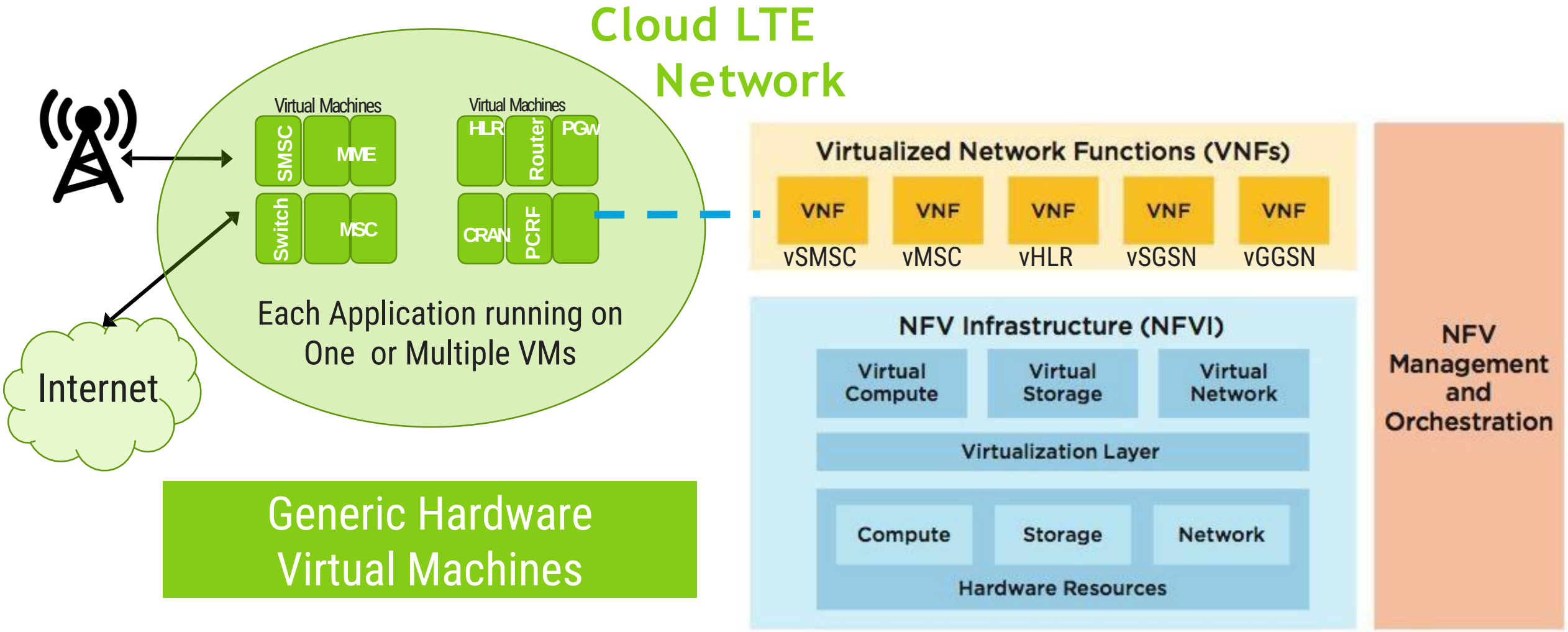
ETSI World Class Standards

October 2012 White Paper

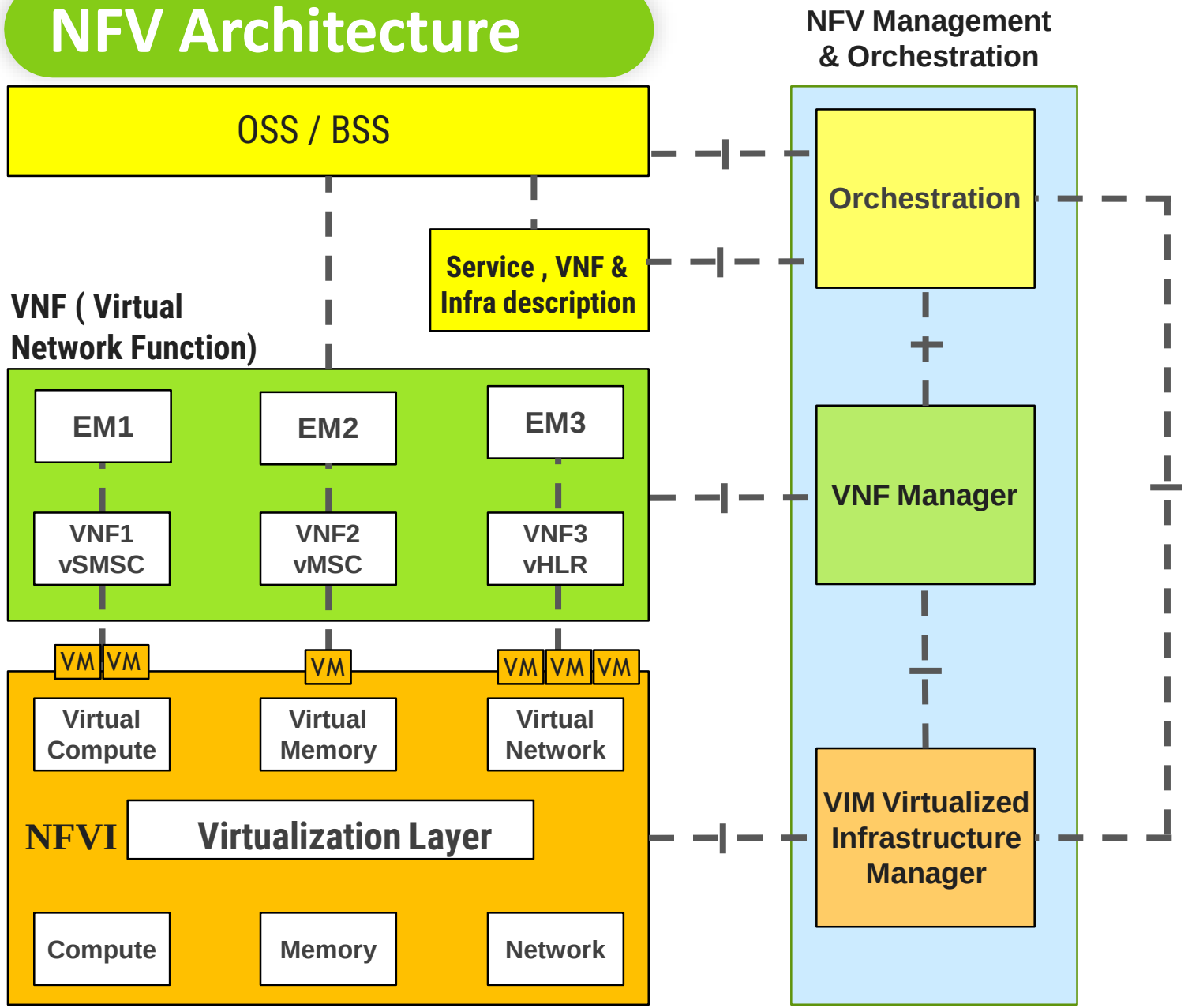
- AT&T USA
- BT
- CenturyLink
- China Mobile
- Colt
- Deutsche Telekom
- KDDI
- Orange
- Telecom Italia
- Telefonica
- Telstra
- Verizon

NFV : Network function Virtualization

Components of NFV



NFV Architecture



NFV Layers

NFVI

(Network Function Virtualization Infrastructure)

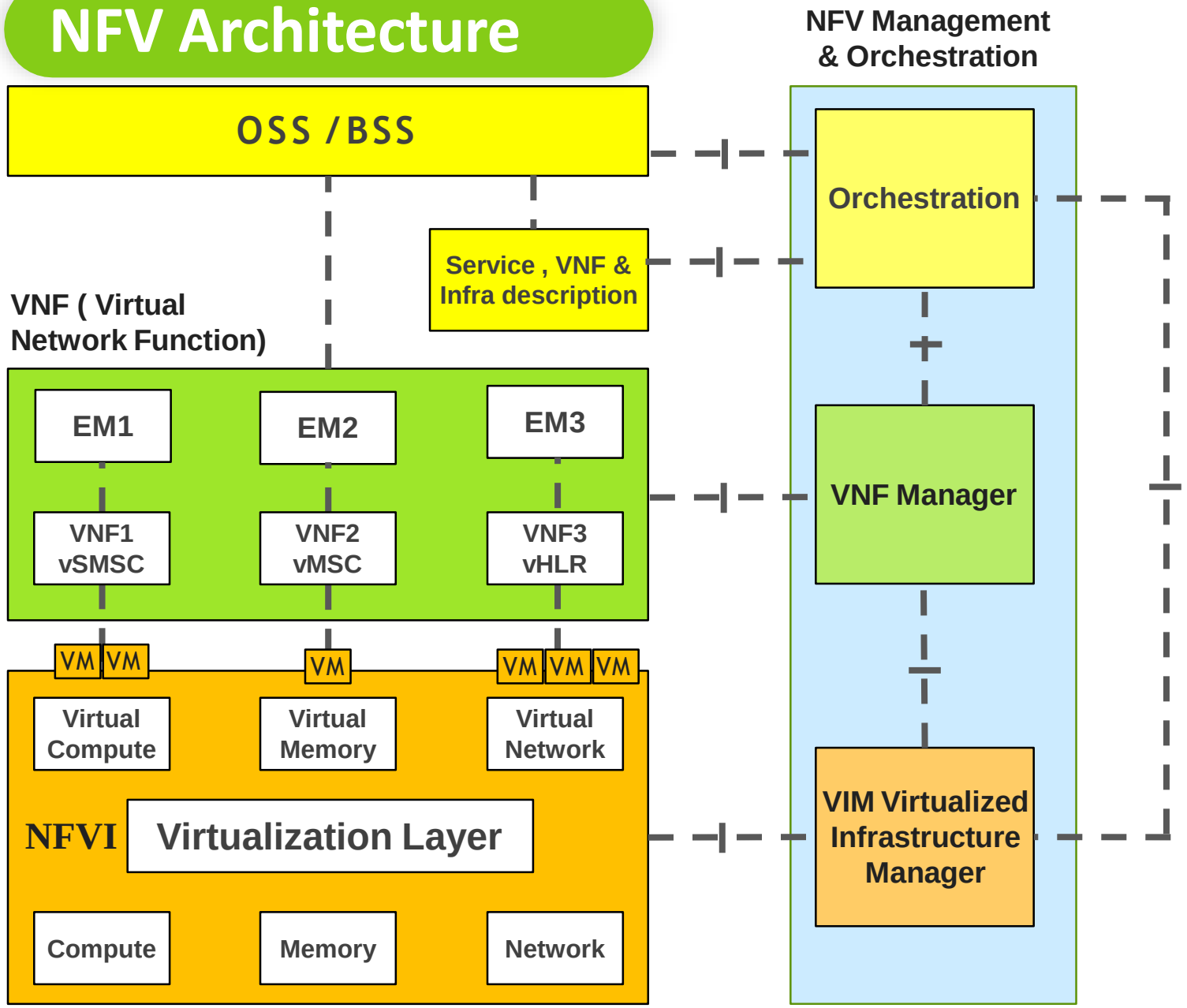
- Manage Physical part in NFVI
- Hypervisor : Responsible for abstracting physical resources into virtual resources

VIM

(Virtualized Infrastructure Manager)

- Management / Control for NFVI
- Also manages Reports & events

NFV Architecture



VNF

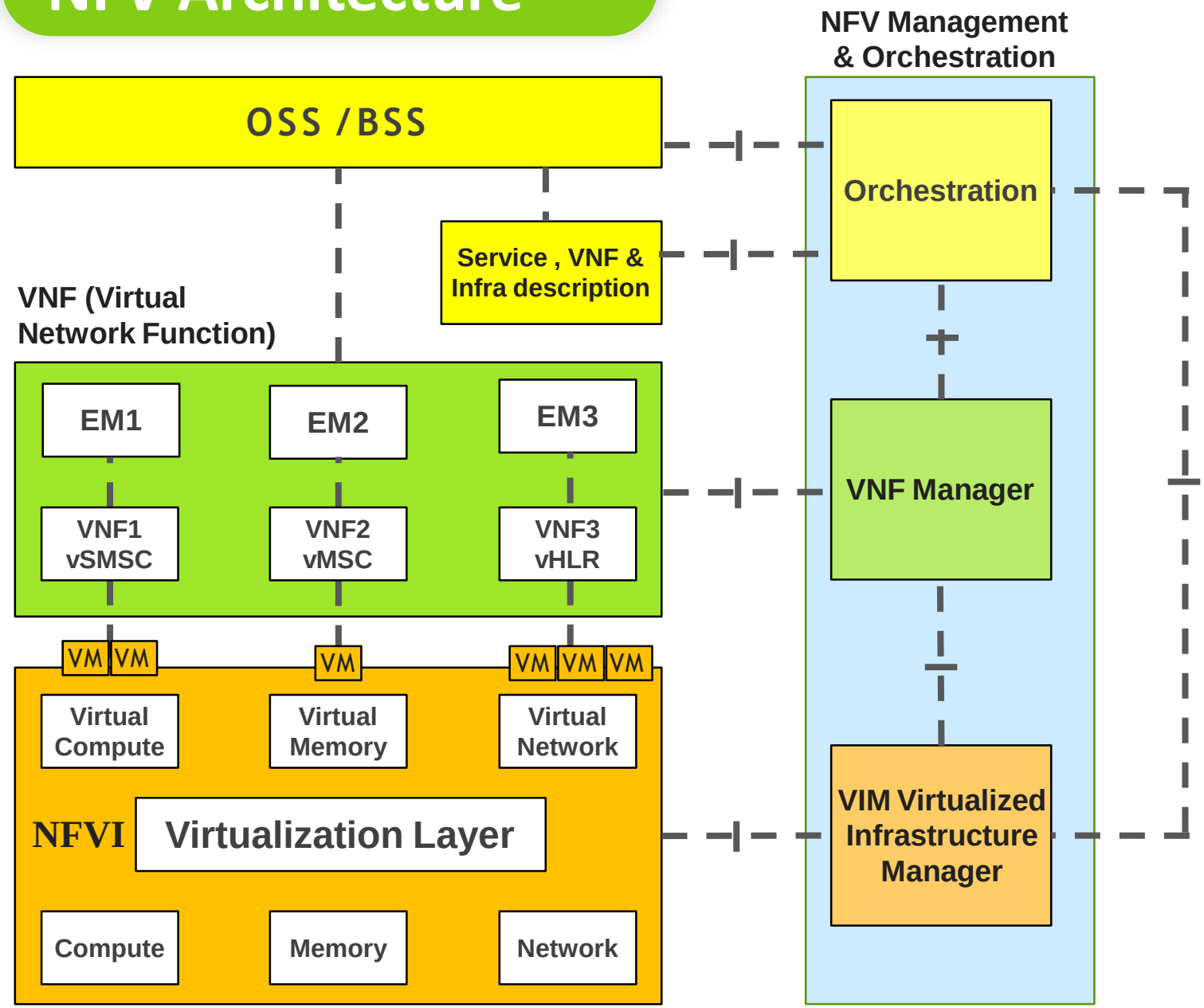
- A VNF is the basic block in NFV Architecture , Example :
  - ✓ vIMS
  - ✓ vMME
  - ✓ vMSC
  - ✓ vSwitch
  - ✓ vRouter

- **EM (Element Management):** FCAPS of Application such as Link down , KPI Degradation etc.

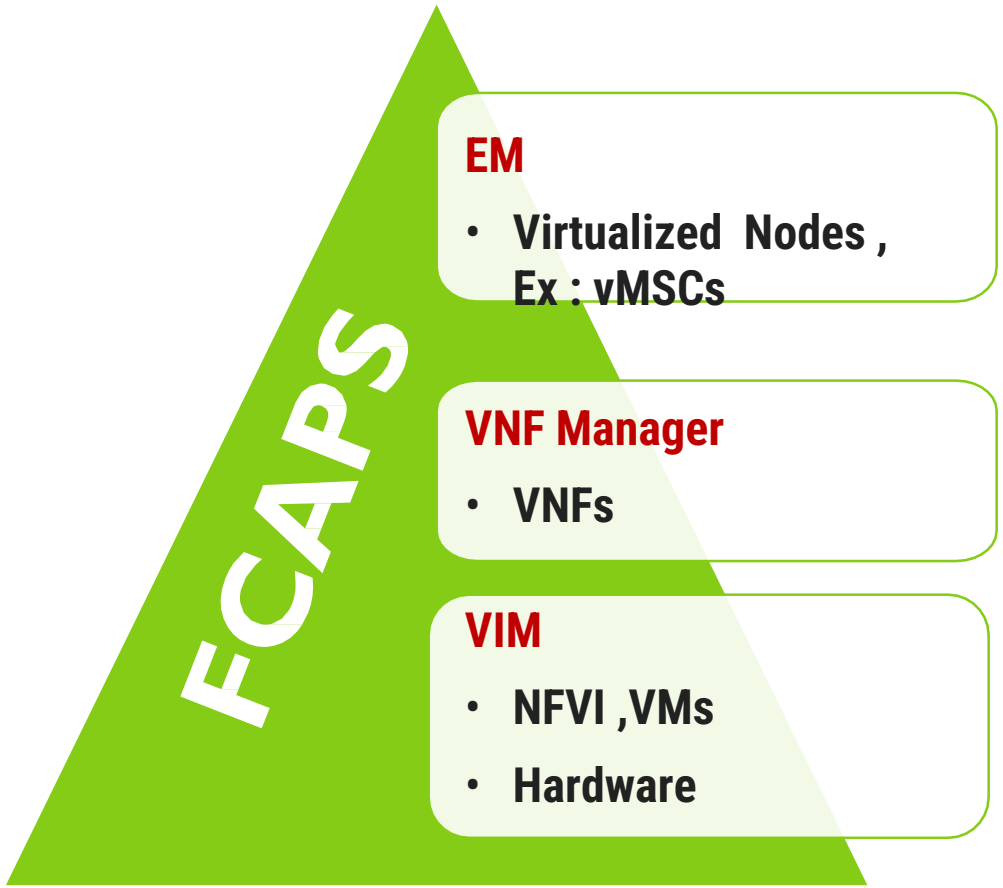
VNF Manager

- Life cycle management of VNFs :
  - ✓ setting up/
  - ✓ Maintaining
  - ✓ Tearing down
- FCAPS of Virtualization and VNF

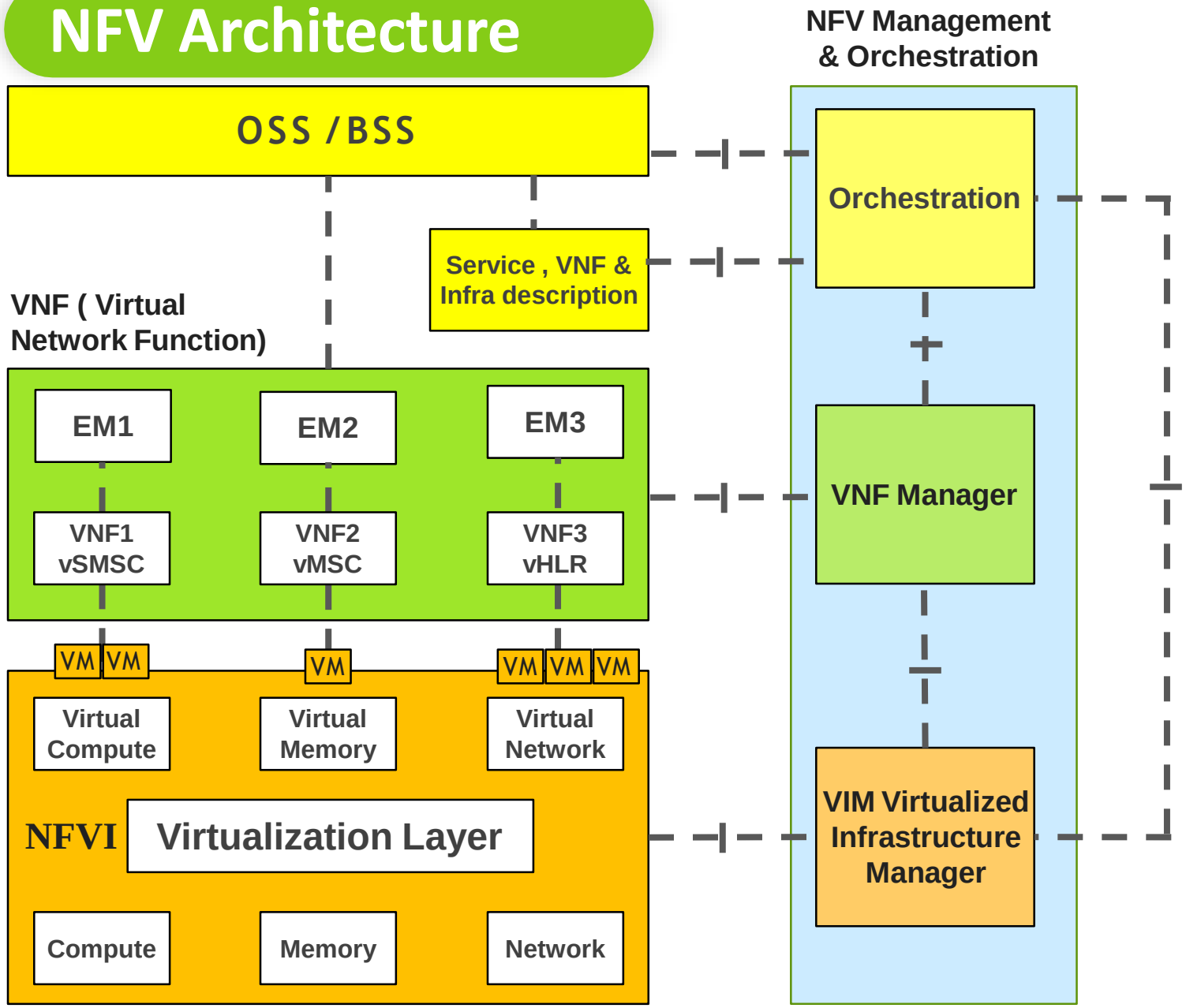
NFV Architecture



FCAPS Layers



NFV Architecture



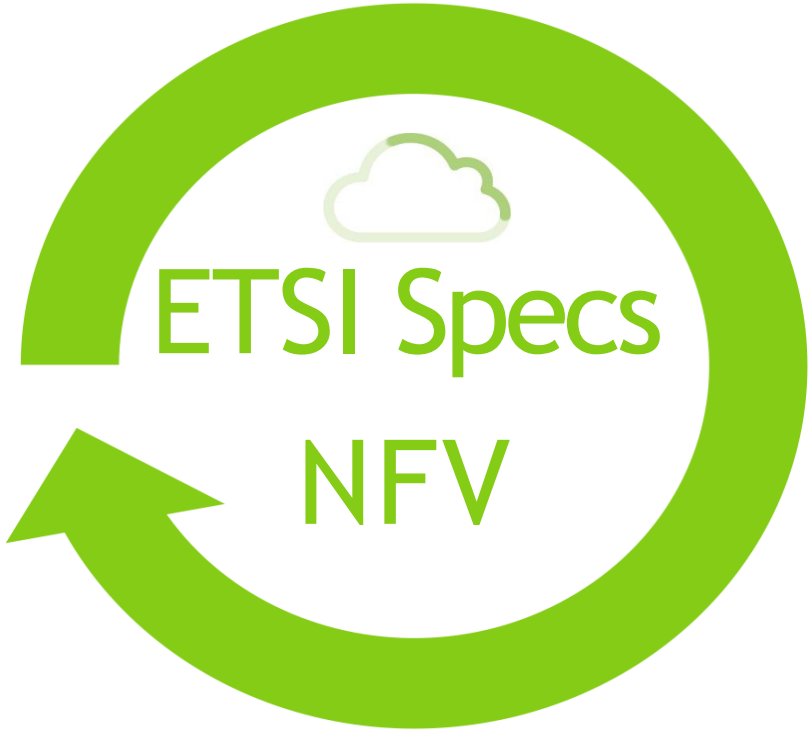
NFV Layers

NFV Orchestrator (NFVO)

- Key to Automation of SDN & NFV
- Performs :-
  - Resource orchestration
  - Network service orchestration
- Global resource management of NFVI resources via VNFM & VIM
- Allocating and scaling resources
- Single Orchestrator for NFV service

NFV Architecture

https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports



NFV

IoT System Management with NETCONF-YANG

Outline

- Need for IoT Systems Management
- SNMP
- SNMP Limitations
- System Management with NETCONF-YANG

Need for IoT Systems Management

1. Automating Configuration
2. Monitoring Operational & Statistical Data
3. Improved Reliability
4. System Wide Configurations
5. Multiple System Configurations
6. Retrieving & Reusing Configurations

Need for IoT Systems Management

1. Automating Configuration

- System management interfaces provide predictive and easy-to-use management capability to automation system configuration when a system consists of multiple devices or nodes.
- Ensures all devices have the same configuration and variations or errors due to manual configurations are avoided.

Need for IoT Systems Management

2. Monitoring Operational & Statistical Data

- **Operational data:** the system's operating parameters that are collected by the system at runtime.

- **Statistical data:** system performance (e.g. CPU and memory usage) data for fault diagnosis or prognosis (forecasting).

- Management system can help in monitoring operational and statistical data of a system. This data can be used for fault diagnosis.

Need for IoT Systems Management

3. Improved Reliability

- A Management system that allows validating system configuration before they are put in to effect can help in improving system reliability.

Need for IoT Systems Management

4 . System Wide Configurations

- IoT systems consist of multiple devices or nodes, which have wide system configurations for the correct functioning.
- Each device is configured separately (either manual or automated).
- Used in system faults or undesirable outcomes.
- Ensures that the configuration changes are either applied to all devices or to none.
- In the failure, the configuration changes are rolled back.

Need for IoT Systems Management

5 . Multiple System Configurations

- Some systems have multiple valid configurations that are used at different times or in certain conditions.
- It can be provided by SNMP.

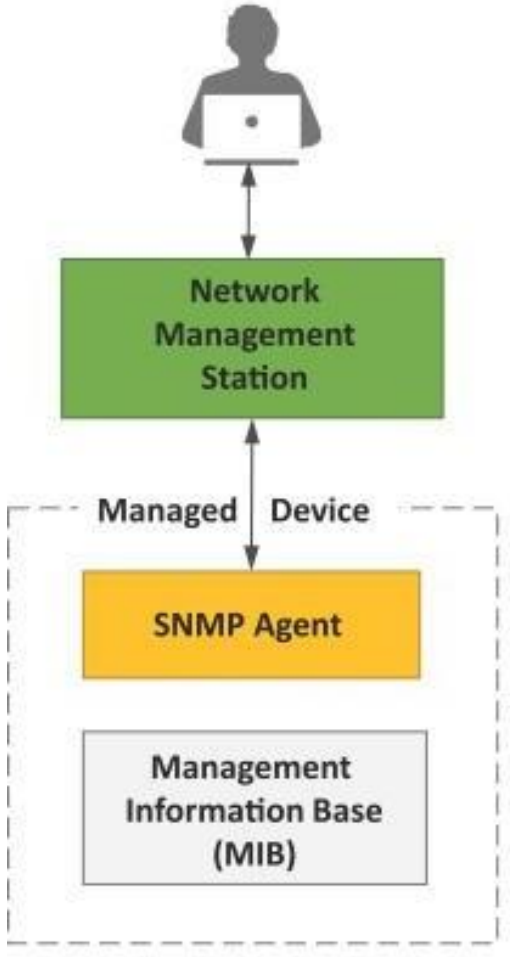
Need for IoT Systems Management

6. Retrieving & Reusing Configurations

- Help in reusing the configurations for other devices of the same type.
- Ensure that when a new device is added, the same configuration is applied.
- The management system can retrieve the current configuration from a device and apply the same to the new devices.

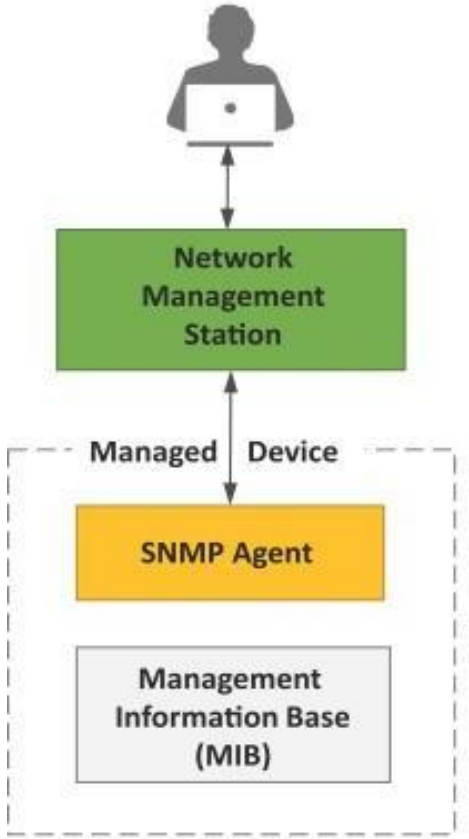
Simple Network Management Protocol (SNMP)

- 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 component include
  - Network Management Station (NMS)
  - Managed Device
  - Management Information Base (MIB)
  - SNMP Agent that runs on the device



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
- The components of the entities involved in managing a device with SNMP, including the Network Management Station (NMS), Managed Device, Management Information Base (MIB) and the SNMP Agent that runs on the device.
- NMS executes SNMP commands to monitor and configure the Managed Device.
- The Managed Device contains the MIB which has all the information of the device attributes to be managed. MIBs use the Structure of Management Information (SMI) notation for defining the structure of the management data.



Limitations of SNMP

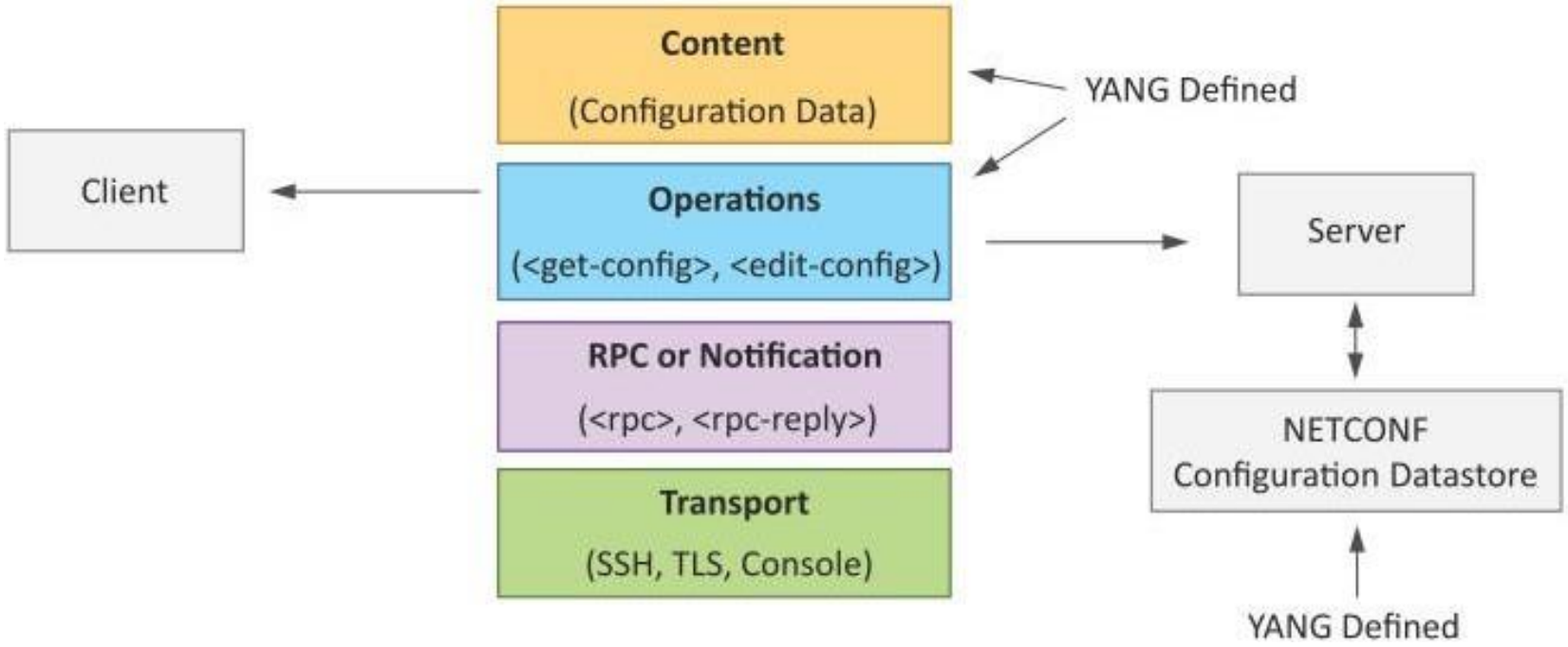
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement 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.

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