

Embedded Systems and IoT

Arpit Khandelwal

Index

- [Intro to Embedded Systems](#)
- Intro to IoT
- [Cloud Computing](#)
 - [Virtualization](#)
 - [Containerization](#)
 - [Docker](#)
 - [Kubernetes](#)
- Edge Computing

Embedded Systems

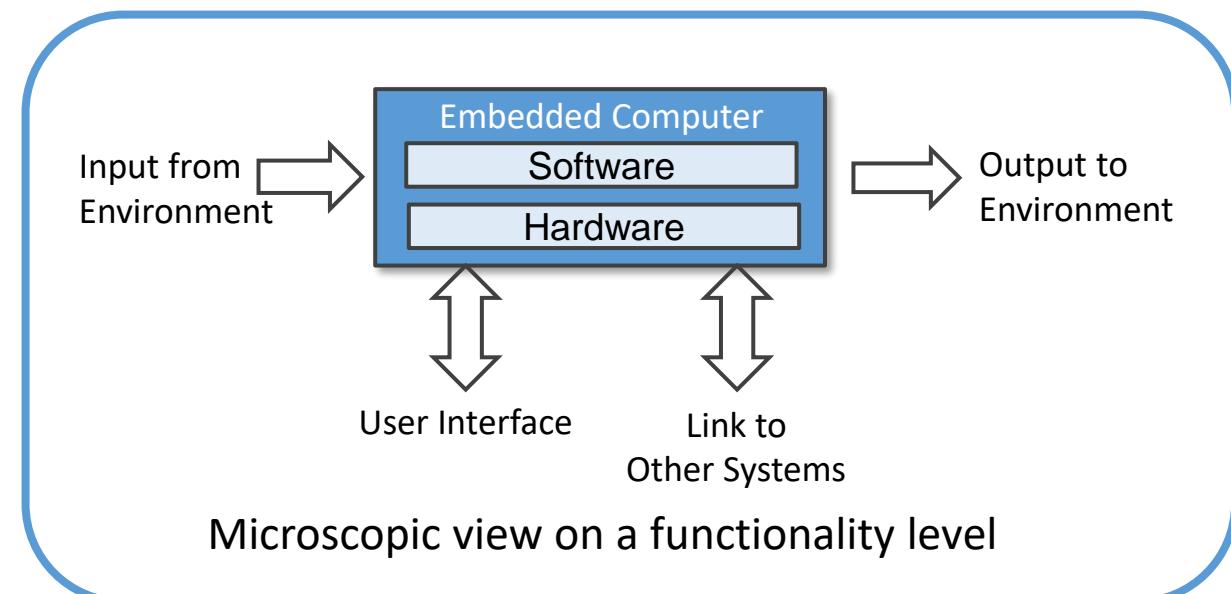
- Embedded Systems (ES) are actually computers embedded (hidden) into other systems (like washing machine, printer, AC etc. except desktop/laptop)
- They are microcontroller based systems designed to control a function or a range of functions and are *not meant to be programmed by the end user.*
- Embedded systems are mostly special-purpose or single functioned.
- They are constrained in cost, energy consumption, size and form factor and respond to inputs in real-time.

Embedded Systems

- Difference between a PC and ES is that PC is a general purpose computing device which can be programmed by the user to perform various tasks whereas ES is a special purpose or dedicated computing device which performs only the assigned task and cannot be programmed by the user.
- Hence, microprocessors are generally used in PCs while ES use microcontrollers.

Introduction to Embedded Systems

- What is an embedded system?
 - Application-specific computer system
 - Interacting with its environment
 - Build into a larger system
 - Often with real-time computing constraints
- What is the motivation for building an embedded system?
 - Better performance
 - More functions and features
 - Lower cost e.g. through automation
 - More dependable
 - Lower power



Embedded Systems

- Following components are essential to implement an Embedded System
 - 1. Hardware
 - 1. Processing Elements – microcontroller
 - 2. Peripherals
 - 1. I/O Devices
 - 2. Sensors and Actuators
 - 3. Display Units
 - 4. Power Management Units
 - 3. Memory
 - 4. Bus
 - 2. Software
 - 1. System Software – Compilers, Emulators, Debugging units
 - 2. Application Software – specific to application

Example Embedded System: Bike Computer

Functions:

- Speed measurement
- Distance measurement

Constraints:

- Size
- Cost
- Power and energy
- Weight

Inputs:

- Wheel rotation indicator
- Mode key

Output:

- Liquid crystal display

Use low-performance microcontroller:

- 9-bit, 10 MIPS

Input:
Wheel rotation
Mode key



Output:
Display speed and distance

Example: Gasoline Automobile Engine Control Unit

Functions:

- Fuel injection
- Air intake setting
- Spark timing
- Exhaust gas circulation
- Electronic throttle control
- Knock control

Constraints:

- Reliability in harsh environment
- Cost
- Size

Many inputs and outputs:

- Discrete sensors and actuators
- Network interface to rest of car

Use high-performance microcontroller:

- E.g. 32-bit, 3 MB flash memory, 50-300 MHz



Options for Building Embedded Systems

Dedicated Hardware
Software Running on Generic Hardware

Implementation	Design Cost	Unit Cost	Upgrades & Bug Fixes	Size	Weight	Power	System Speed
Discrete logic	low	mid	hard	large	high	?	very fast
ASIC	high (\$500K/mask set)	very low	hard	tiny – 1 die	very low	low	extremely fast
Programmable logic – FPGA, PLD	low to mid	mid	easy	small	low	medium to high	very fast
Micropocessor + memory + peripherals	low to mid	mid	easy	small to medium	low to moderate	medium	moderate
Microcontroller (int. memory & peripherals)	low	mid to low	easy	small	low	medium	slow to moderate
Embedded PC	low	high	easy	medium	moderate to high	medium to high	fast

Microcontroller based embedded system

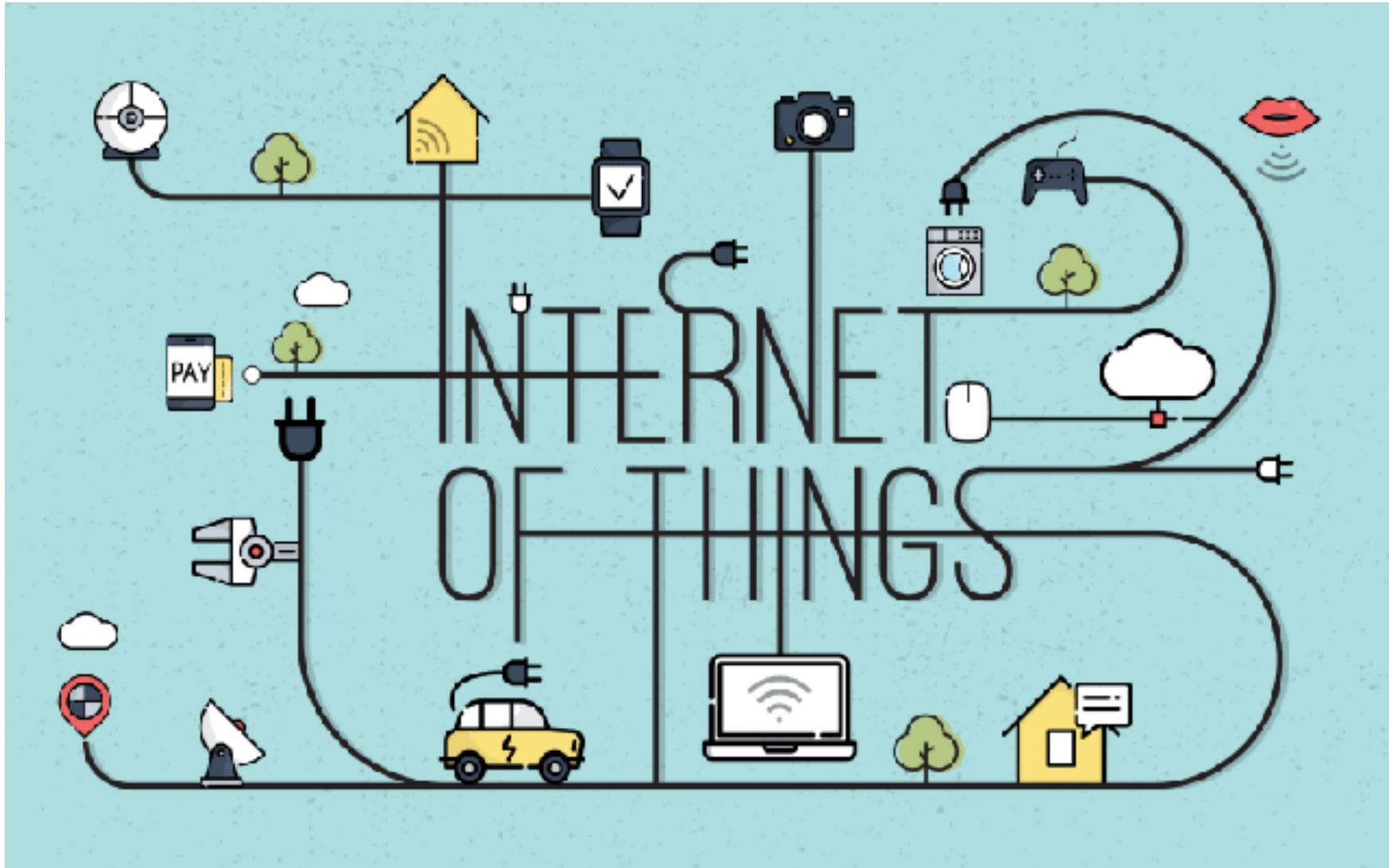
Benefits of Microcontroller-based Embedded Systems

- Greater performance and efficiency
 - Software makes it possible to provide sophisticated control
- Lower costs for mixed signal-processing systems
 - Less expensive components can be used
 - Manufacturing costs reduced
 - Operating costs reduced
 - Maintenance costs reduced
- More features
 - May not be possible or practical with other approaches
- Better dependability
 - Adaptive system which can compensate for failures
 - Better diagnostics to improve repair time

Impact of Constraints

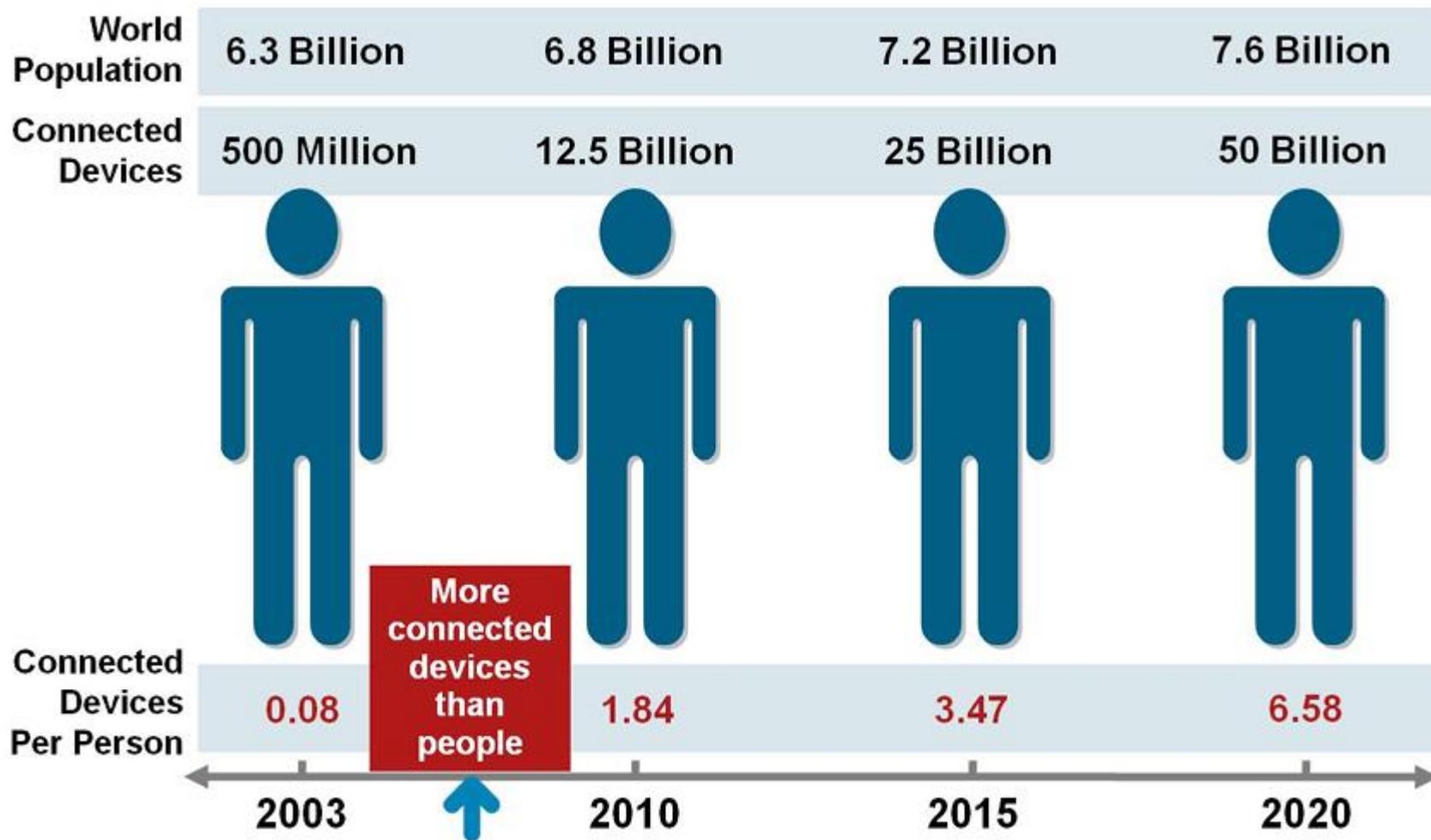
- Microcontrollers used (rather than microprocessors)
 - Include peripherals to interface with other devices, respond efficiently
 - On-chip RAM, ROM reduce circuit board complexity and cost
- Programming language
 - Programmed in the C language rather than Java (resulting in smaller and faster code – less expensive MCU)
 - Some performance-critical code may be in assembly language
- Operating system
 - Typically no OS, but instead a simple scheduler
 - If OS is used, it is likely to be a lean RTOS

Internet of Things (IoT)



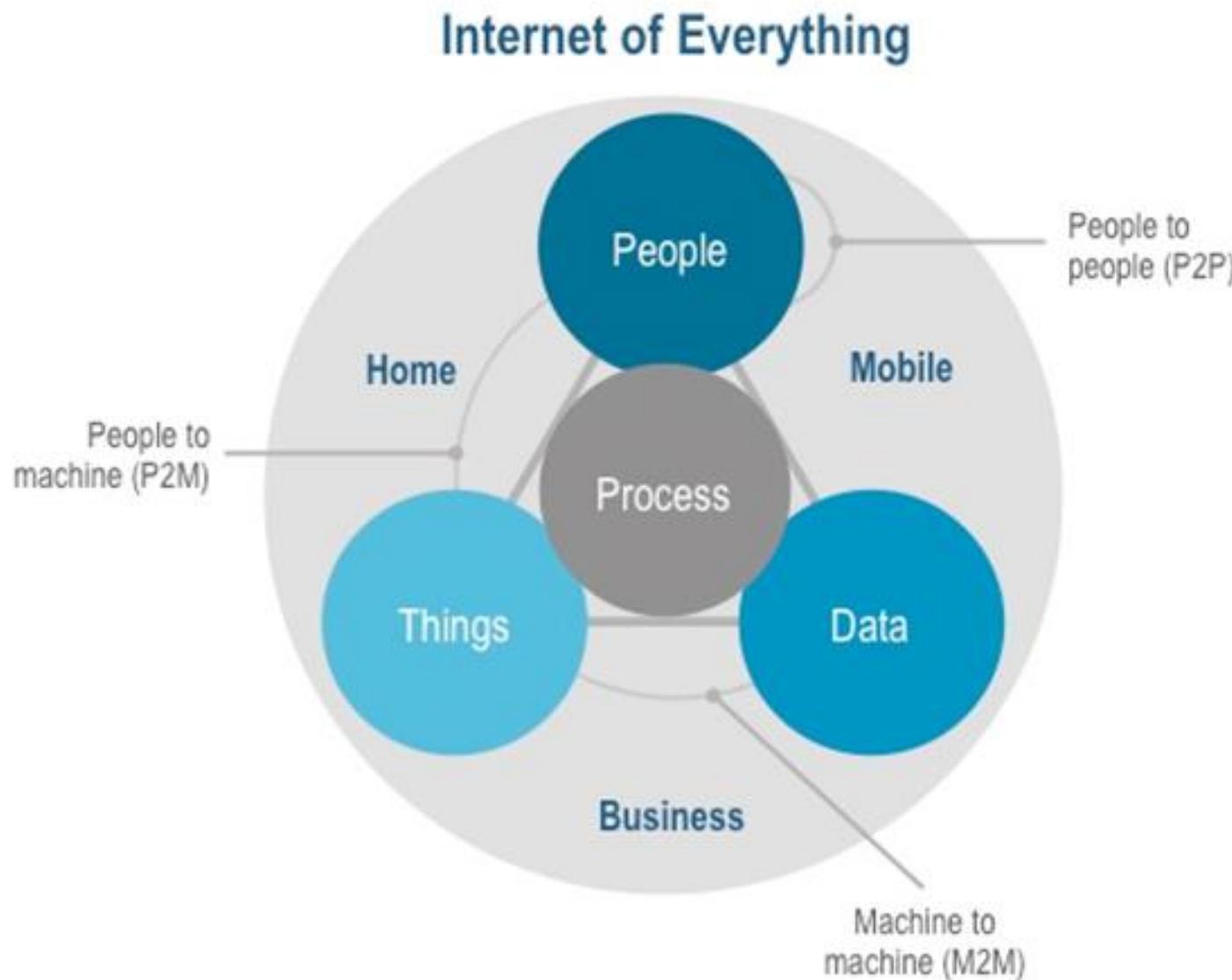
https://www.linkedin.com/pulse/hardware-design-challenges-embedded-internet-things-iot-komal-chauhan/?trk=related_artcle_Hardware%20Design%20Challenges%20of%20the%20Embedded%20Internet%20of%20Things%20%28IoT%29_article-card_title

Internet of Things (IoT)



Source: Cisco IBSG, April 2011

Internet of Everything (IoE)



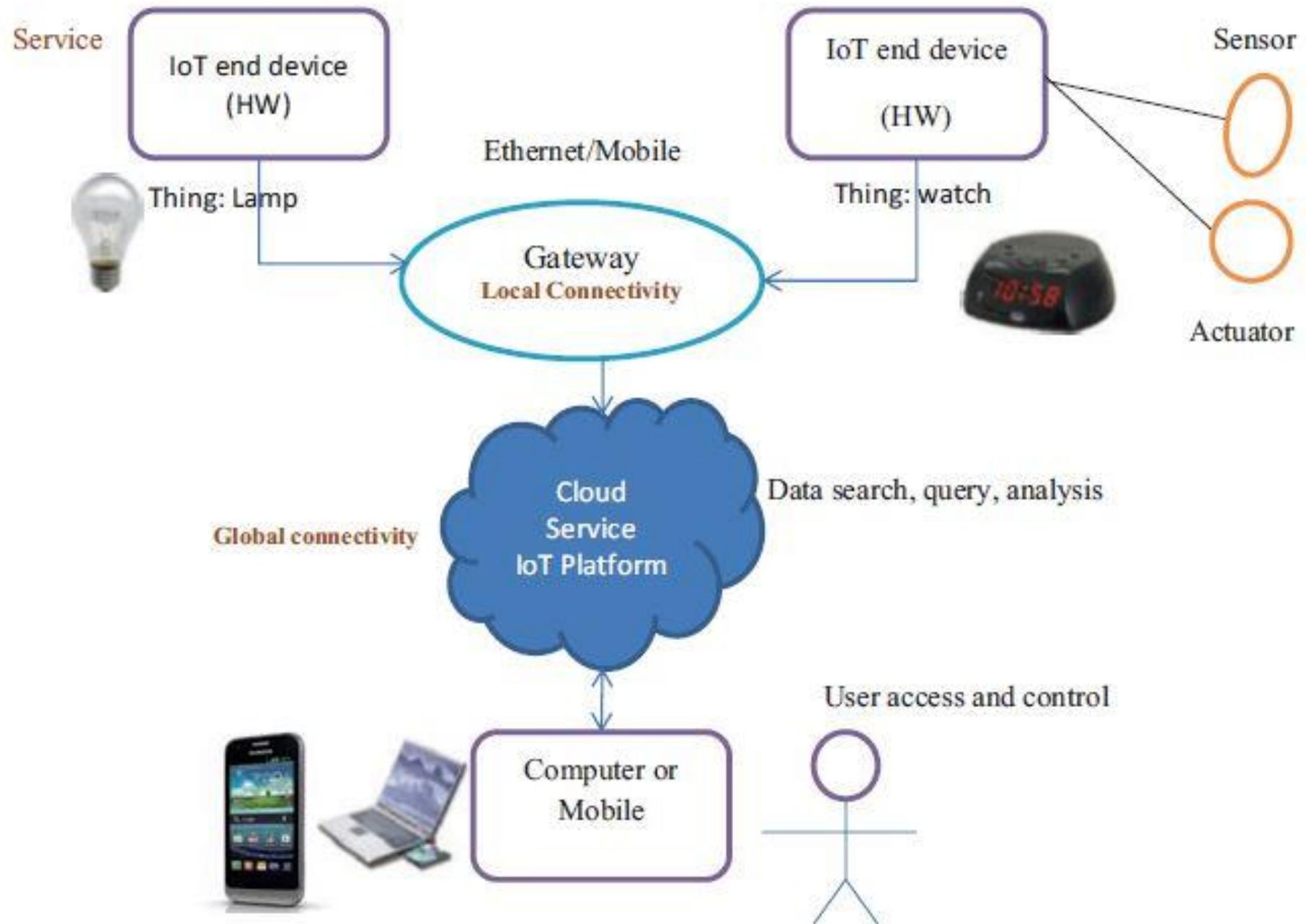
Internet of Things (IoT)

- IoT means anything can communicate with anything through any place and at any time using standard protocols with minimum or no human intervention.
- IoT is the inter networking of physical devices, vehicles, buildings and other items embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data.
- IoT incorporates following fields (majorly)
 - Sensors and Actuators
 - VLSI and Embedded Systems
 - Signal Processing
 - Communication
 - Machine Learning / AI / Deep Learning
 - Cybersecurity

IoT – Smart World

- IoT technology aims to convert this world into a smart world
- Word “Smart” here implies that all the “things” in this world can
 1. Sense their surroundings
 2. Understand the information
 3. Communicate the info to other “things”
 4. Receive info from other things
 5. Make their own decisions based on collected info
 6. Act on the decisions made
- All of these without involving a human being!!!!

IoT System Architecture



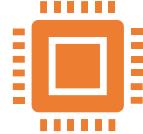
Important Requirements for IoT System

- Any IoT system must be
 1. Low power consuming
 2. Compact
 3. Self-sufficient in power and energy needs
 4. Low cost
 5. Long lifetime
 6. Reliable and secure
 7. Scalable and dynamic

Challenges in IoT systems

1. Large number of deployed systems
2. Heterogeneous nature of deployed devices
3. Deployment in hostile and uncharacterized environment
4. Resource constrained devices
5. Maintaining low cost
6. Mobility of nodes

Challenges in IoT systems



Better hardware and code that runs efficiently.



Ensuring energy efficient operation.



Aggregating large data sets and acting on it in real-time.



Modeling and predicting the behavior of complex systems.



Ensuring reliable connectivity and scalability.



Guaranteeing secure operation and data confidentiality

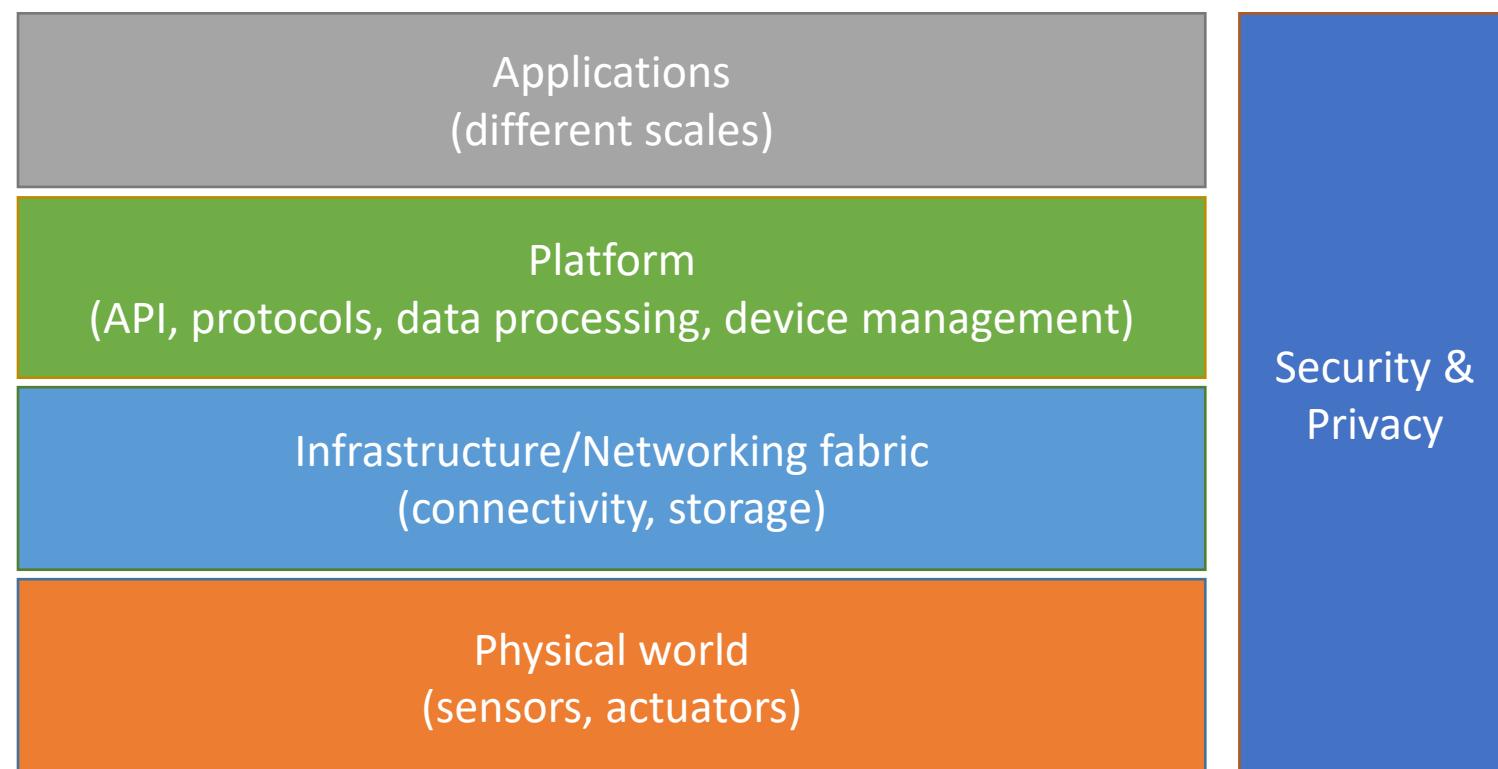
Developing easy to use products and services

Complying with regulations

IoT System Architecture

- IoT system architecture consists of following layers

1. Physical Layer
2. Network Layer
3. Service Layer/Middleware
4. Application Layer



Physical Layer

- Consists of following elements
 1. Sensors and actuators
 2. Basic signal processing circuits
 3. Basic computing circuits

Selecting a sensor

- How to select a suitable sensor for a particular application
 1. Type of measurement – Direct or Indirect
 2. Type of output – Analog or Digital

Sensor Characteristics

1. High Sensitivity
2. High Resolution
3. Compact Size
4. Low Cost
5. Low Power Operation
6. Quick Response Time
7. Linear Operation
8. Low Noise

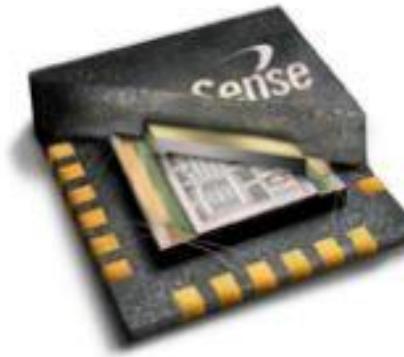
Sensors' Technologies

- Different sensor technologies available for IoT
 1. Mechanical sensors – resistive, capacitive, piezoelectric, magnetic etc
 2. Electronic sensors – semiconductor based
 3. Optical sensors
 4. MEMS/NEMS sensors - Micro/Nano Electro Mechanical Sensors
 5. RFID/Smart Sensors

MEMS

Invensense MPU-6050
6-axis gyroscope and accelerometer

- Miniaturized mechanical and electro-mechanical devices – all devices between 1 to 100 micron size.
- Adding movement to electronic devices.
- Also called **MicroSystems** or **Micromachines**.
- Used to fabricate both sensors and actuators.



4 x 4 x 1 mm

Signal Processing and Computing Hardware

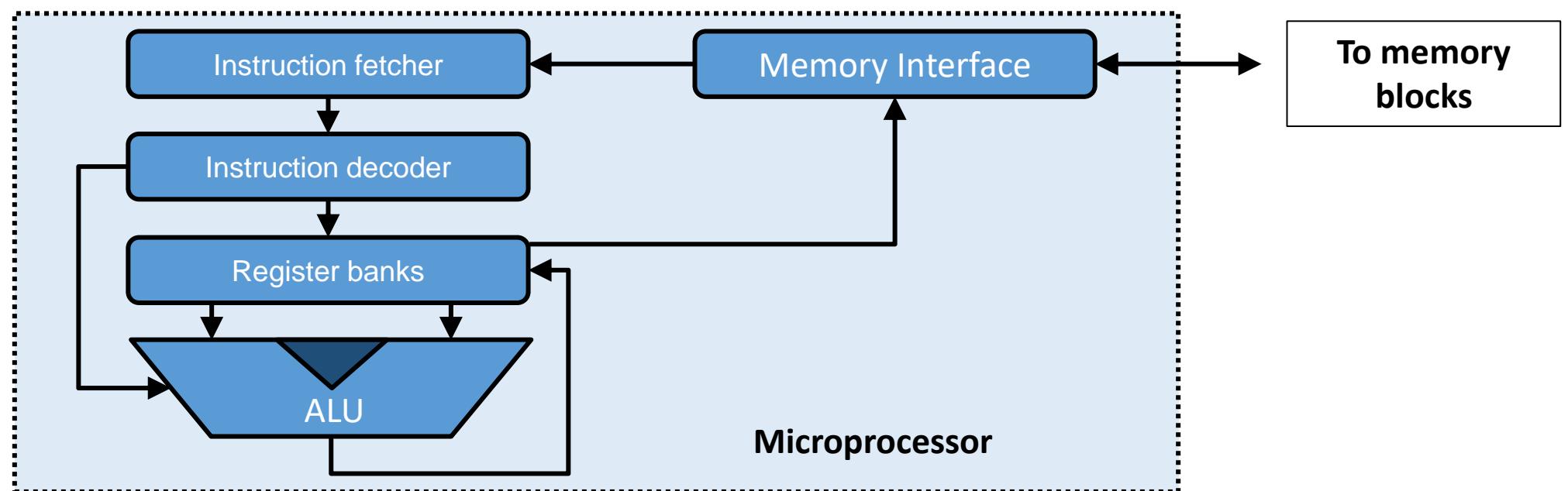
- Basic signal processing required on the sensor data *to extract useful information*
 1. Amplification
 2. A/D and D/A conversion
 3. Filtering
 4. Frequency domain conversion
- Following platforms are available for IoT signal processing and computing hardware
 1. ASIC – Application Specific Integrated Circuits
 2. FPGA – Field Programmable Gate Arrays
 3. Microprocessors/Microcontrollers

Microprocessors and Microcontrollers

- Microprocessor is just a CPU i.e. it consists of an arithmetic and logic unit (ALU), a control unit to synchronize and some registers to store temporary data.
- The memory and I/O peripherals are outside the microprocessor and are interfaced using external bus.
- A microcontroller contains CPU, memory (RAM and ROM) and I/O peripherals all integrated onto a single chip.
- Microcontroller uses internal bus to access the memory and I/O peripherals.

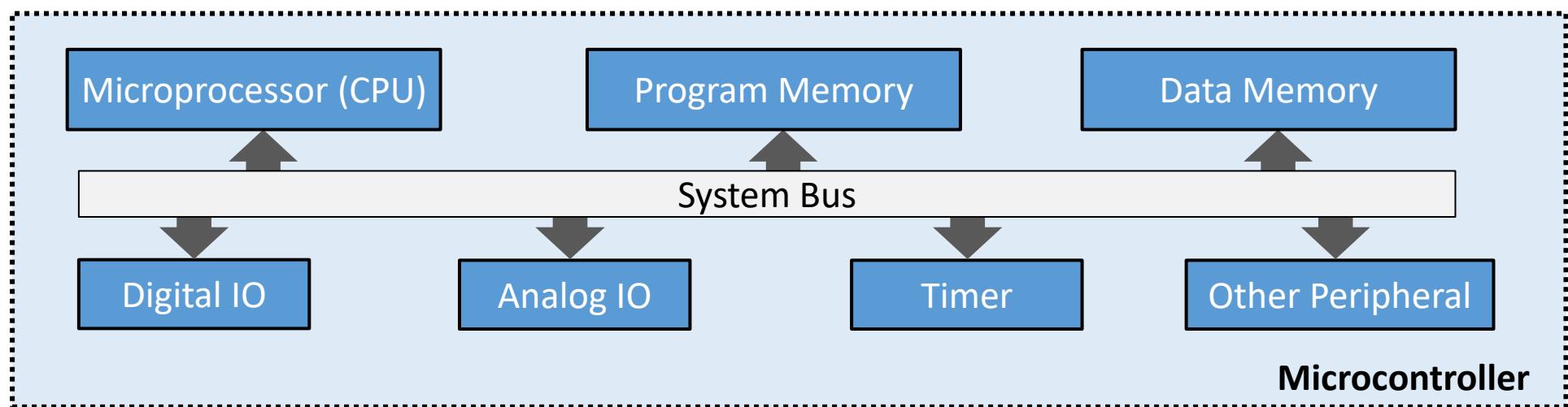
From a Processor to an Embedded System

- Microprocessor (central processing unit, CPU)
 - Typically defined as a single processor core that supports at least instruction fetching, decoding, and executing
 - Normally can be used for general-purpose computing, but needs to be supported with memory and Input/Outputs (IOs)



From a Processor to an Embedded System

- Microcontroller (microcontroller unit, MCU)
 - Typically has a single processor core
 - Has memory blocks, digital IOs, analog IOs, and other basic peripherals
 - Typically used for basic control purpose, such as embedded applications

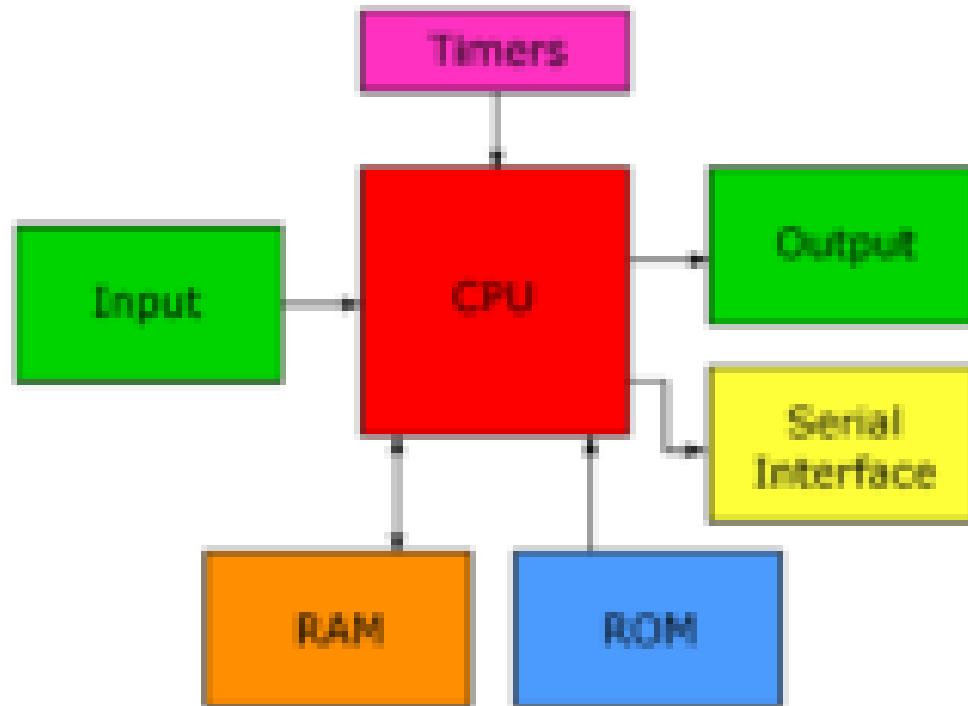


Microprocessors and Microcontrollers

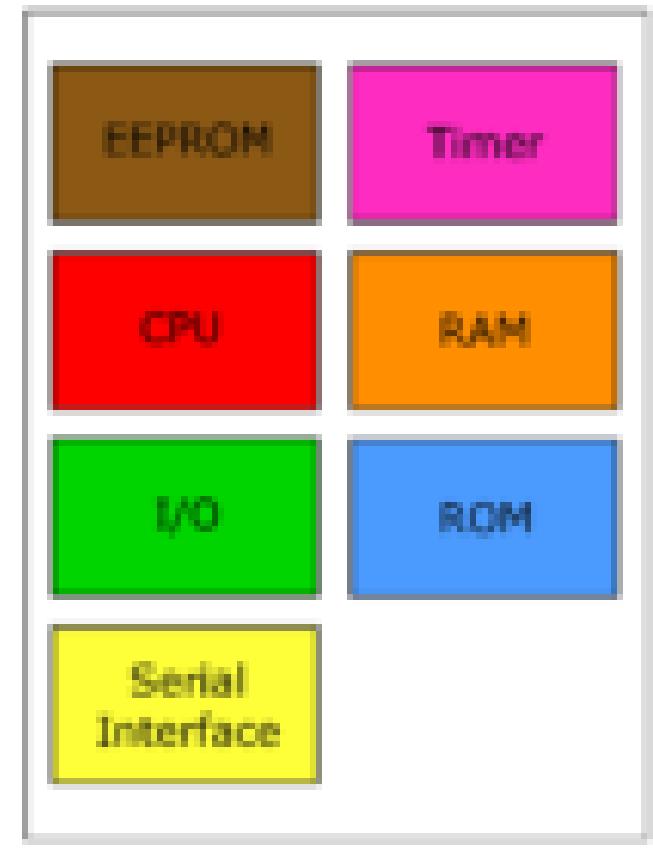


Microprocessors and Microcontrollers

Microprocessor: CPU
and several supporting chips.

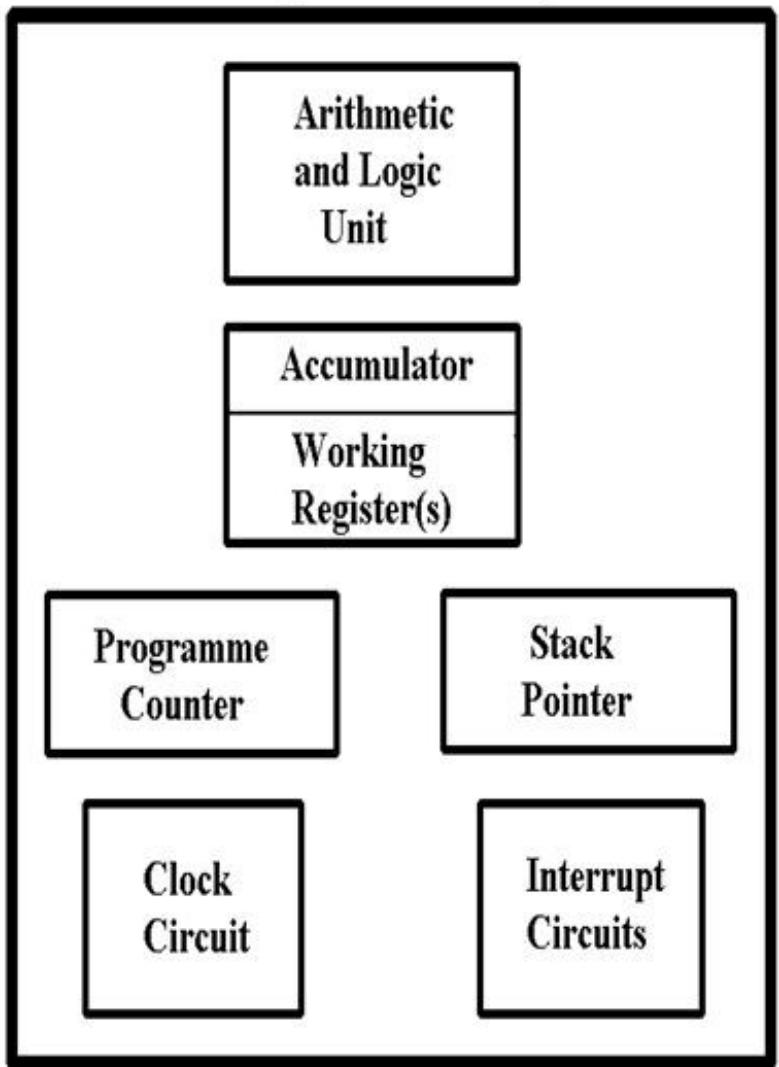


Microcontroller: CPU
on a single chip.

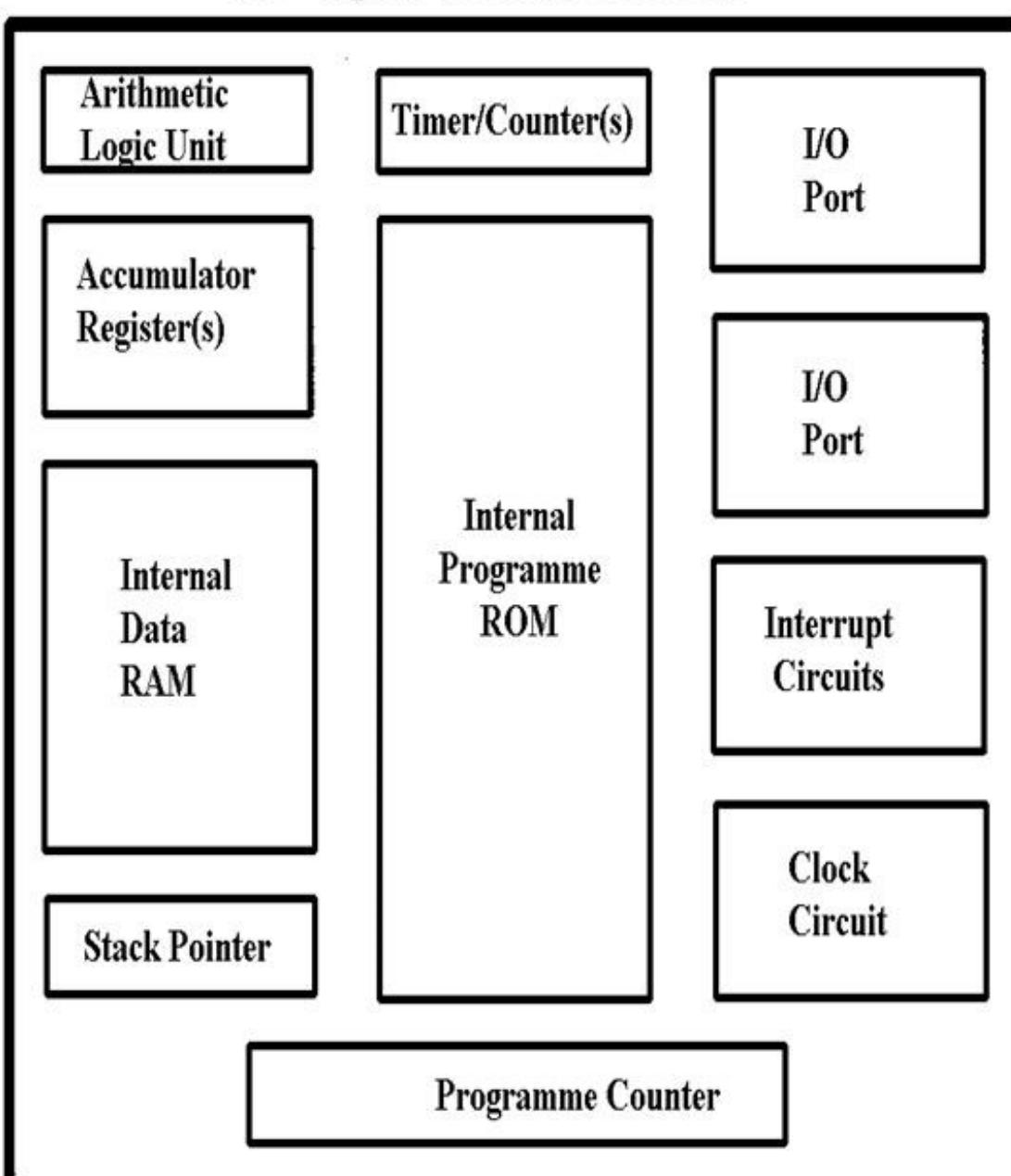


Comparison of microcontroller with microprocessor

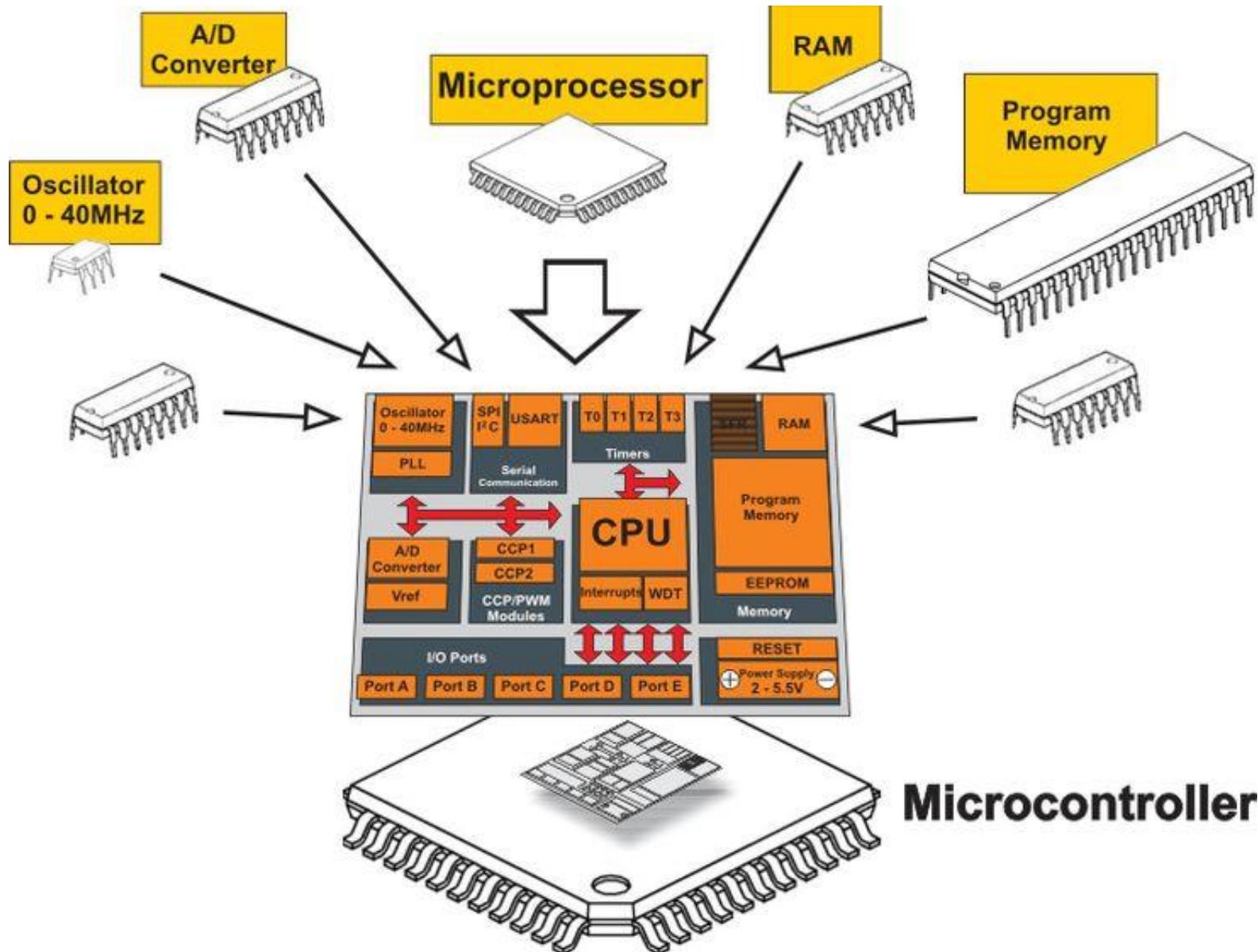
Block Diagram of a Microprocessor



Block Diagram of a Microcontroller



Microprocessors and Microcontrollers



Microcontrollers

- Microcontroller is a small computer on a single IC containing a RAM (Random Access Memory), ROM (Read Only Memory), timer and various I/O peripherals.
- Peripherals may include
 - Analog to Digital converters (ADC)
 - Digital to Analog converter (DAC)
 - Serial Communication Interface
- Microcontrollers are embedded inside a bigger system to perform a certain predefined function.
- The data received from external devices are stored in data memory and is processed according to the instructions stored in program memory of a microcontroller.

Available Microcontrollers



[Atmel AVR](#)



[AVR](#)



[ATX Mega](#)



[ATmega 328P](#)



[PIC 18F877A](#)



[8051](#)



[Arduino](#)

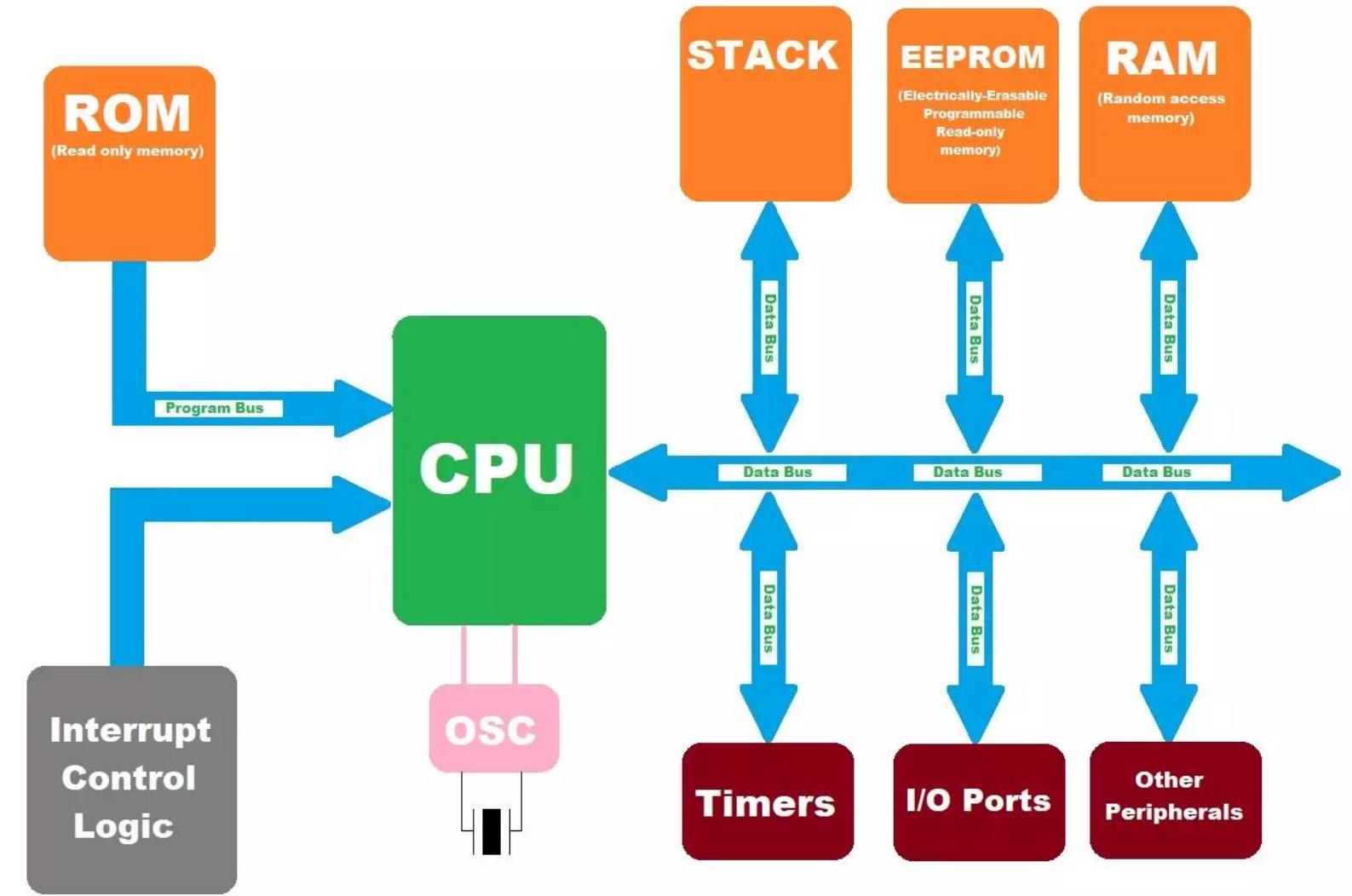


[ARM](#)

Microcontroller Architecture

Microcontrollers Architecture

(What is a Microcontroller ?)



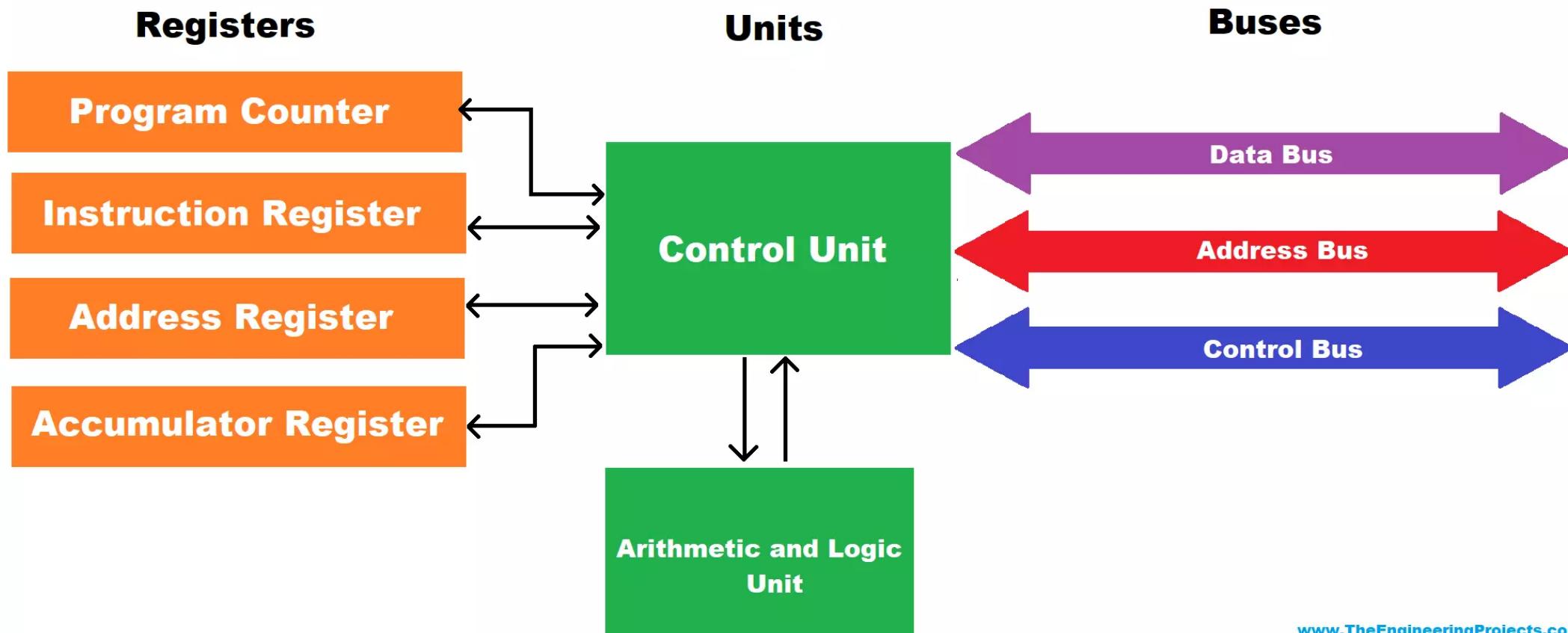
Microcontroller Architecture

A typical microcontroller consists of the following components

- CPU/Microprocessor – brain of the microcontroller
- ROM – non volatile memory
- RAM – volatile memory
- I/O Ports – to access data from outside world
- Oscillator – to generate clock pulses for synchronization
- Timers – pulse generation, modulation
- Interrupts – used to execute urgent tasks

Central Processing Unit (CPU)

- ➡ Central Processing Unit is regarded as the brain of the microcontroller.
- ➡ CPU takes instructions in the form of programming & executes them.



CPU (Central Processing Unit)

- CPU takes instructions in the form of programming and executes them on the data received via data bus.
- Bus is a collection of signal lines on which the data flows in the form of binary signals i.e. ‘0’ and ‘1’.
- CPU consists of several registers (storage elements)
 - Data Registers store the data
 - Address Registers store the address of the memory data access
 - Instruction Registers store the instruction or code which is executed on the data

CPU (Central Processing Unit)

- **Program Counter (Instruction Pointer)** – register that contains the address of the next instruction to be executed by the processor.
- As each instruction gets fetched, the value of program counter increases by 1.
- **Instruction Register** – holds the instruction being executed
- CPU fetches the instruction from the address provided by the program counter and stores it into the instruction register till its execution is completed.
- **Instruction Decoder** – This is connected to the instruction register and it decodes the instruction and establishes the sequence of events to follow.

CPU (Central Processing Unit)

- **Address Register** – stores the memory address from which data will be fetched or to which the data will be sent.
- **Accumulator Register** – where intermediate results of ALU operations are stored
- **Stack** – a reserve memory used to keep track of program's internal parameters such as functions, return addresses, passed parameters etc.
- **Stack pointer** – a register used to indicate the location of last item put on the stack
- **Interrupt** – request for the processor to temporarily stop its current operation and perform another time – sensitive task.
- Interrupts are classified as hardware or software interrupts.

CPU (Central Processing Unit)

- The sequence of events happening inside a CPU are as follows:
 1. At the beginning, the address of the first instruction is stored in the **program counter**.
 2. CPU then places this address on the **address bus** and simultaneously increments the program counter to point to the next instruction.
 3. Then, a **chip enable (CE)** signal and **memory read** signal is sent on the **control bus** to enable the ROM and fetch the code from the address placed on the address bus.
 4. Then, the instruction from ROM is placed on the **data bus** and this instruction gets stored in the instruction register of the CPU.

CPU (Central Processing Unit)

- The sequence of events happening inside a CPU are as follows:
 1. If the instruction is to copy content of A to C, then first the address of A is placed on the address bus.
 2. Then, a memory read signal is activated on the control bus.
 3. Then, data from A gets placed on the data bus and is stored in the accumulator register of the CPU.
 4. Then, address of C is placed on the address bus and a memory write signal is activated on the control bus.
 5. Then, finally data is placed on the data bus and gets stored at C.

CPU (Central Processing Unit)

- Thus, the CPU operation always starts with fetching the instruction, then decoding it and then executing it i.e. Fetch – Decode – Execute.
- Fetching and decoding of the instruction is done in one machine cycle while execution is done in another machine cycle.

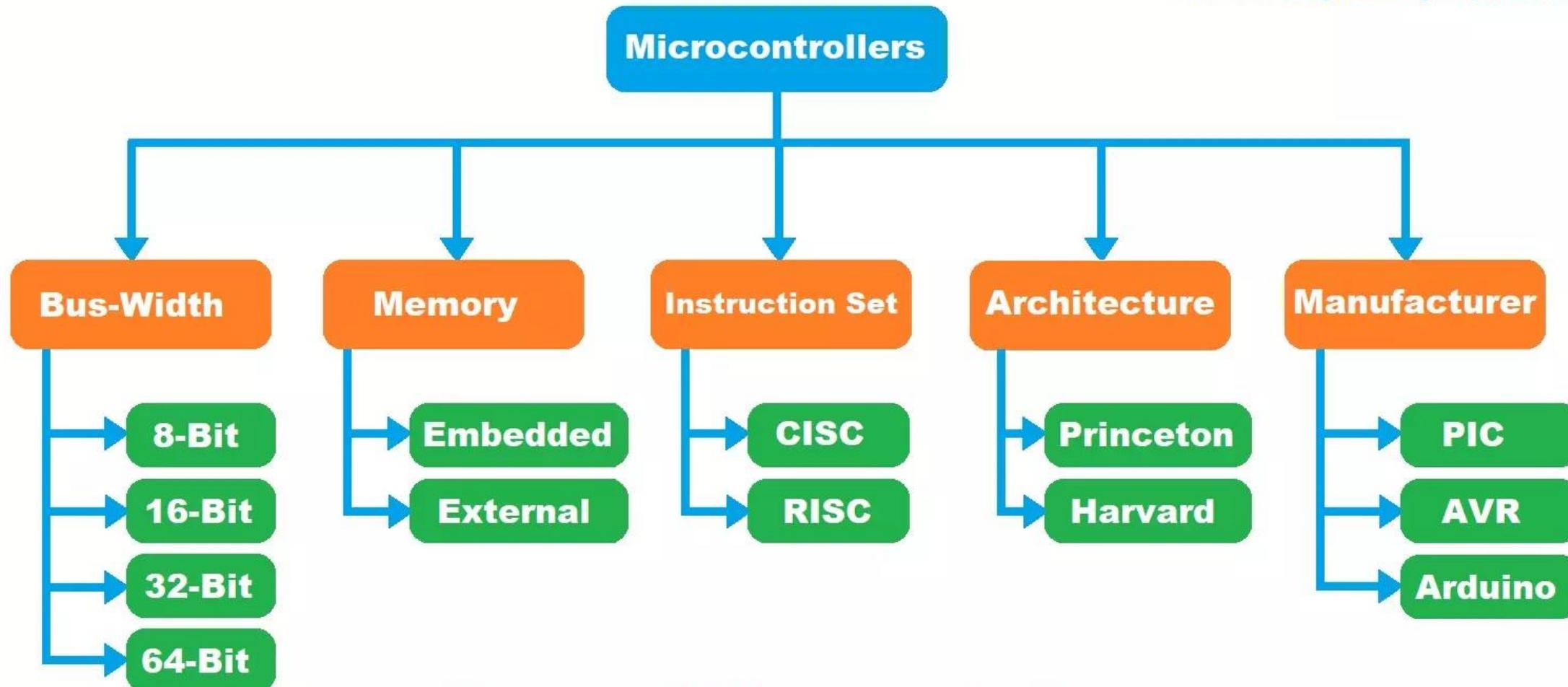
ROM – Read Only Memory

- ROM is used to store the code or program which we write for the microcontroller.
- ROM is a non-volatile type of memory i.e. it cannot be erased easily when the microcontroller is performing any function.
- Most microcontrollers use *EEPROM – Electrically Erasable Programmable Read Only Memory* to store the code.
- EEPROM cannot be erased by powering off the microcontroller, it can only be modified using a compiler.

RAM – Random Access Memory

- RAM is used to temporarily store the data which is being processed by the microcontroller.
- RAM is a type of volatile memory i.e. data can be easily erased and modified when microcontroller is in operation.
- Data can also be lost if microcontroller is powered off.
- Generally 2 types of RAM are present – Static RAM (SRAM) and Dynamic RAM (DRAM).
- RAM consists of general purpose registers and special purpose registers.

Classification of Microcontrollers



Types of Microcontrollers

Classification of Microcontrollers

- A 8-bit (64-bit) microcontroller can process 8 bits (64 bits) of data simultaneously.
- It can process integers from 0 to 255 and contains 8-bit registers and 8-bit data bus.
- Thus a higher bit microcontroller (32 or 64-bit) can
 - Process more amount of data simultaneously
 - Has higher operating speed
 - Has more memory to store the data
 - Has higher speed data bus
 - Consume more power
 - Require complex instruction set but also performs complex functions easily

RISC and CISC based microcontrollers

RISC – Reduced Instruction Set Computer

- Uses **simple instructions** that can be **executed in a single clock cycle**
- Each instruction is of **same length and executed in the same time** – pipelining is much easier here
- Complex commands are divided into multiple simpler commands
- More number of commands than CISC (longer code size) and hence more RAM is required
- Less transistors are used for storing instructions and more are available for general purpose registers

RISC and CISC based microcontrollers

RISC – Reduced Instruction Set Computer

- Load/Store Architecture – only Load and Store instructions access both registers and memory. Whereas, all data processing instructions work on registers only.

RISC and CISC based microcontrollers

CISC – Complex Instruction Set Computer

- Completes the task in as few lines of assembly as possible
- Instructions are complex
- Code is shorter as compared to RISC
- Very little RAM is required to store the instructions
- More transistors are used for storing complex instructions and less transistors are available for general purpose registers
- Different instructions of varying formats, lengths, addressing modes and requiring varying clock cycles for execution.

CPU Architectures

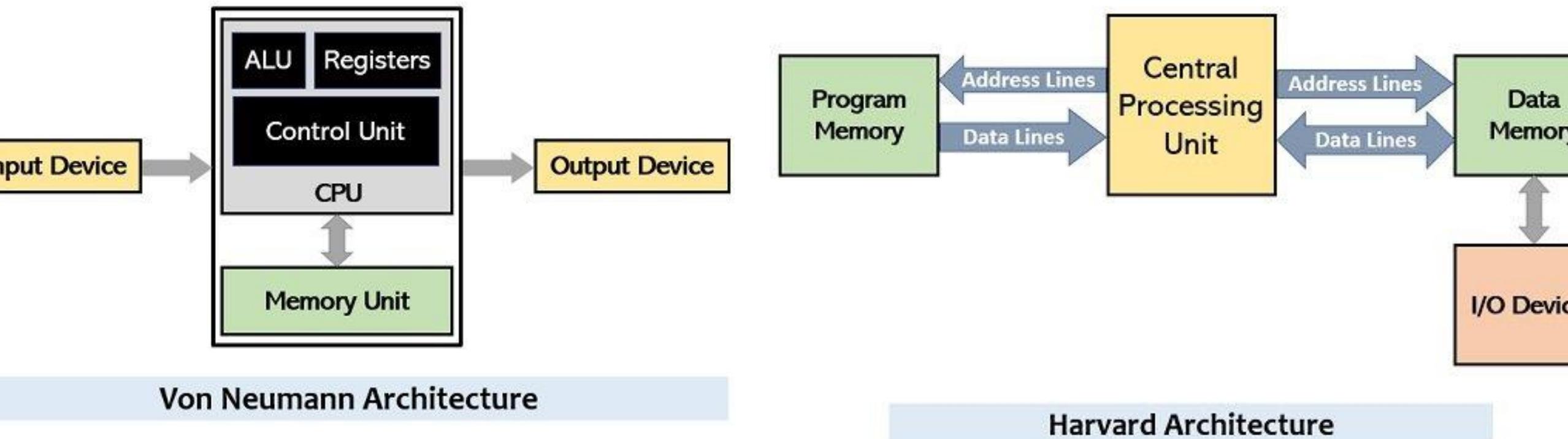
Based on CPU architectures, microcontrollers can be classified into 2 types:

1. Von Neumann (Princeton) Architecture
 2. Harvard Architecture
-
- In Von Neumann architecture, both instructions and data are stored in the same memory.
 - In Harvard architecture, instructions and data are stored in separate memories (ROM and RAM respectively)

CPU Architectures

- In Von Neumann architecture, a single bus is used to send address of the memory and fetch data from the memory.
- Here, getting instructions interferes with accessing RAM and also the instructions and data must have the same bit width (since they share the same bus).
- In Harvard architecture, program bus and data bus are separate and may be of different width.
- This helps in parallel processing and instruction pipelining i.e. next instruction can be fetched at the same time as the result of previous instruction is getting stored.

CPU Architectures

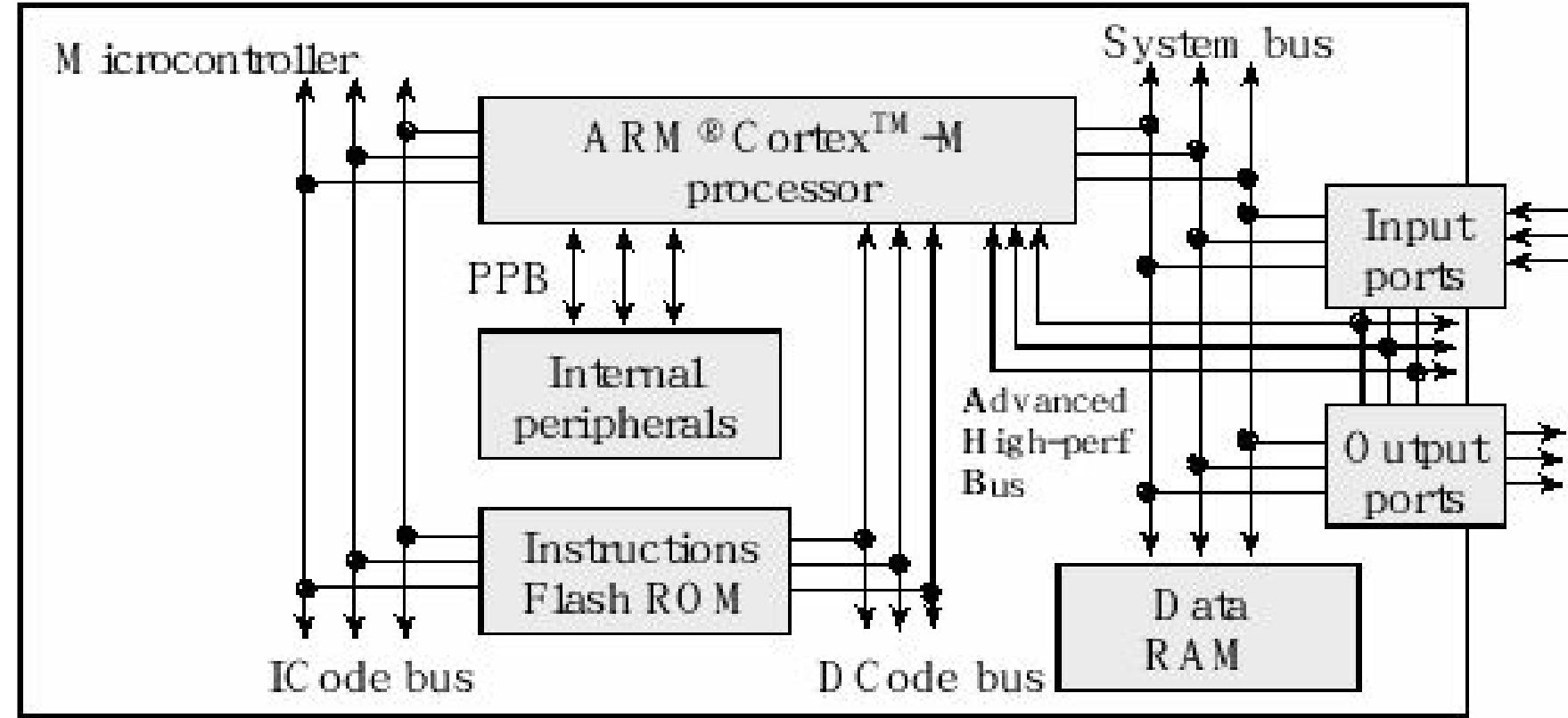


ARM Microcontrollers

- ARM – Advanced RISC Machine is an architecture based on RISC.
- ARM based microcontrollers are the most widely used ones in today's market.
- The first commercial ARM based microcontroller was developed by Acorn in 1985
- 3 series of ARM based microcontrollers
 - ARM Cortex A – used in smartphones, smart TVs and other gadgets
 - ARM Cortex R – used in space applications and missiles
 - ARM Cortex M – motor control and industrial automation

ARM Architecture

- ARM is based on Harvard Architecture – separate data and instruction memory/bus



- Instructions are fetched from Flash ROM using ICode bus
- Data are exchanged between memory and I/O ports using System bus

ARM Modes of Operation

ARM processors can operate in the following 7 modes:

Processor Mode	Code	Description
User	usr	Normal program execution mode
FIQ	fiq	Entered when a high priority (fast) interrupt is raised
IRQ	irq	Entered when a low-priority (normal) interrupt is raised
Supervisor	svc	A protected mode for the operating system (entered on reset and when software interrupt instruction is executed)
Abort	abt	Used to handle memory access violations
Undefined	und	Used to handle undefined instructions
System	sys	Runs privileged operating system tasks

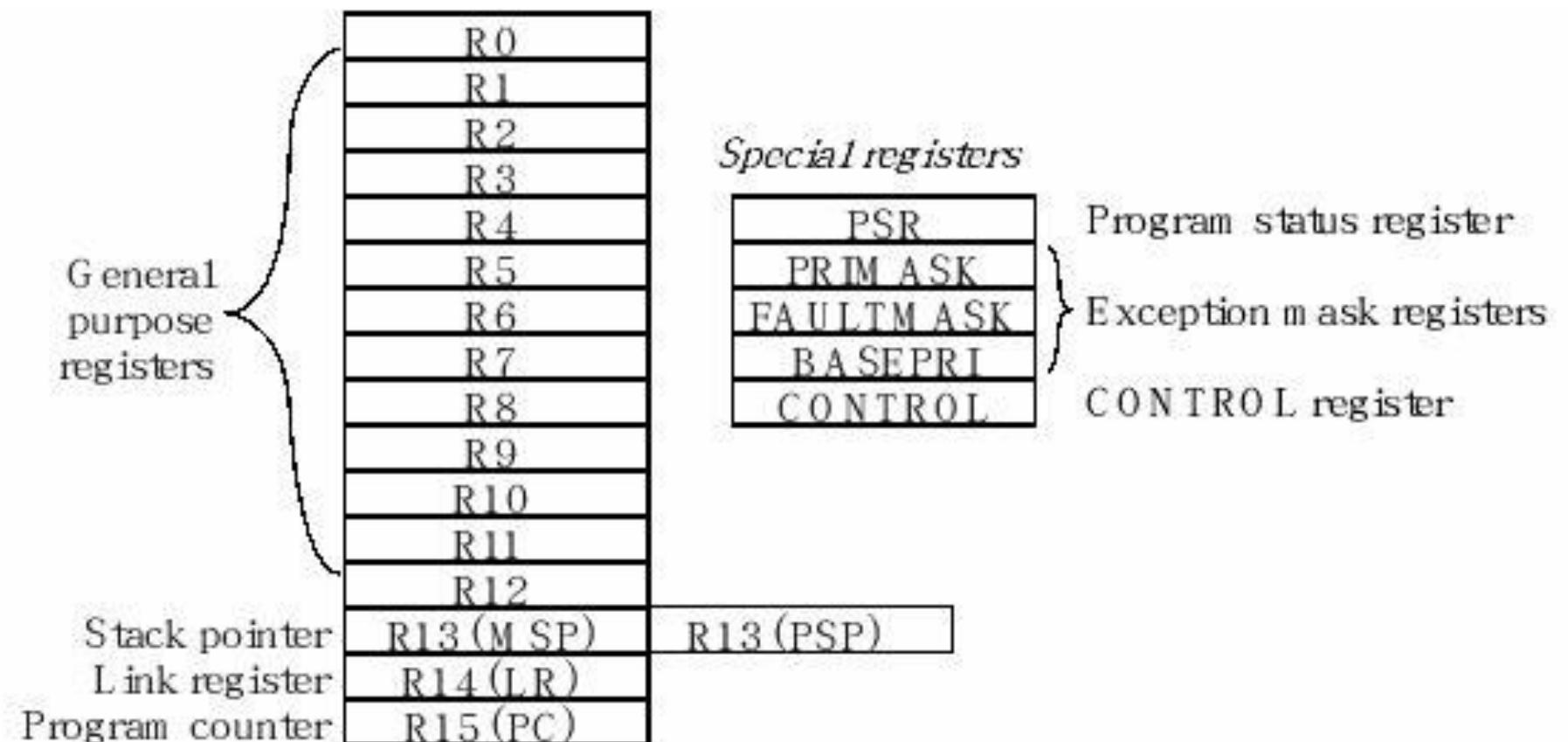
ARM Registers

- Registers are high speed storage inside the processors.

Speed storage inside
The processors.

- R0 to R12 are general purpose registers which can

Contain either data or address.



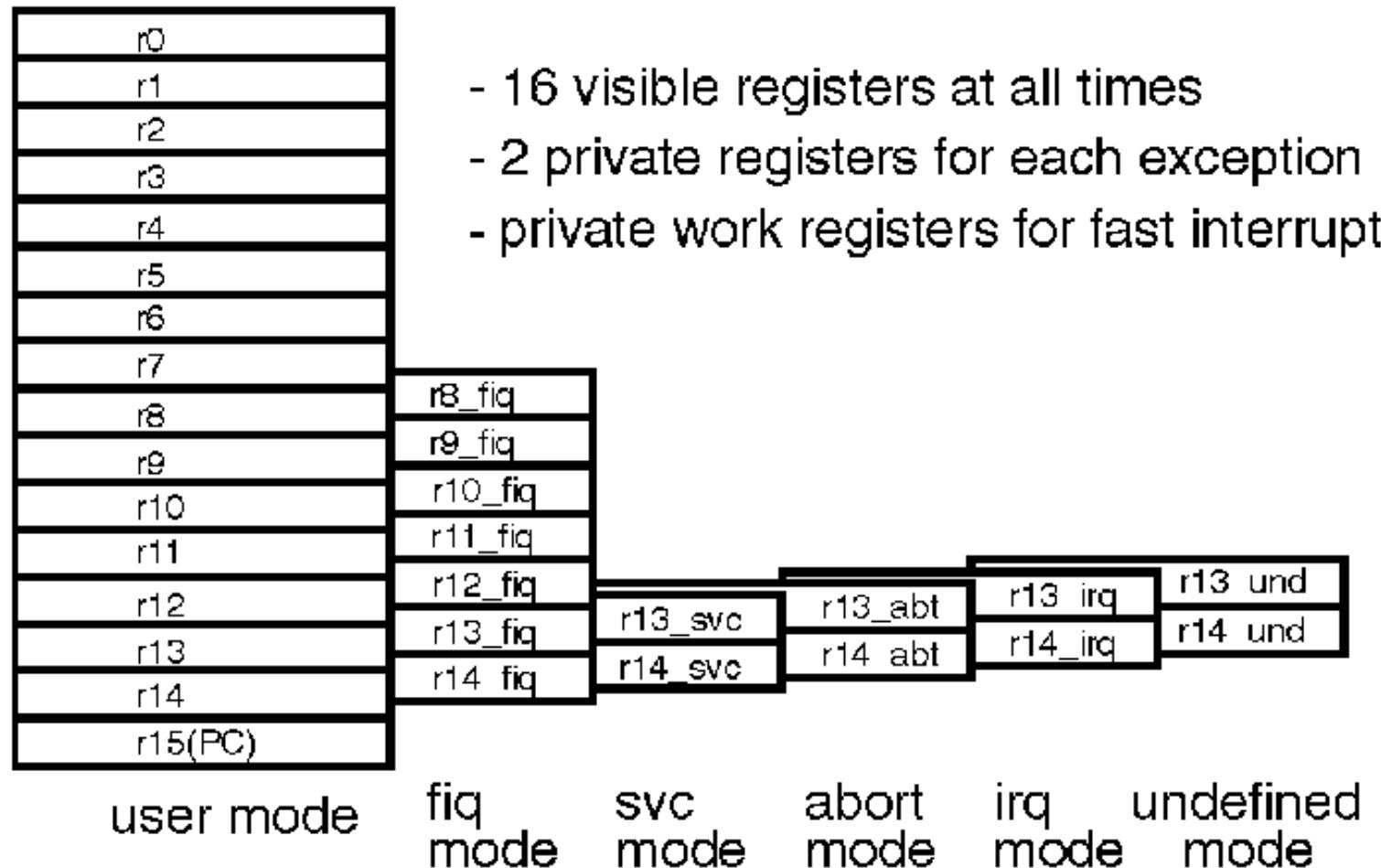
- Stack Pointer points to the top element of the stack and Program Counter points to the next instruction to be fetched from ROM.
- Link Register stores the return address when a subroutine is getting executed.

ARM Registers

- Recent ARM processors have 37 registers, each 32-bit long.
- Out of 37, there is 1 program counter, 1 current program status register (CPSR), 5 saved program status registers (SPSR) and 30 general purpose registers (GPR).
- CPSR is a register containing flags such as zero flag, carry flag, parity flag etc.
- SPSR works as stack register. Whenever an interrupt occurs, the value of CPSR is copied into SPSR and on the completion of interrupt handling, value of SPSR is copied back to CPSR.
- *The register organization discussed in coming slides is applicable only to ARM Cortex A and Cortex R processors and not Cortex M.*

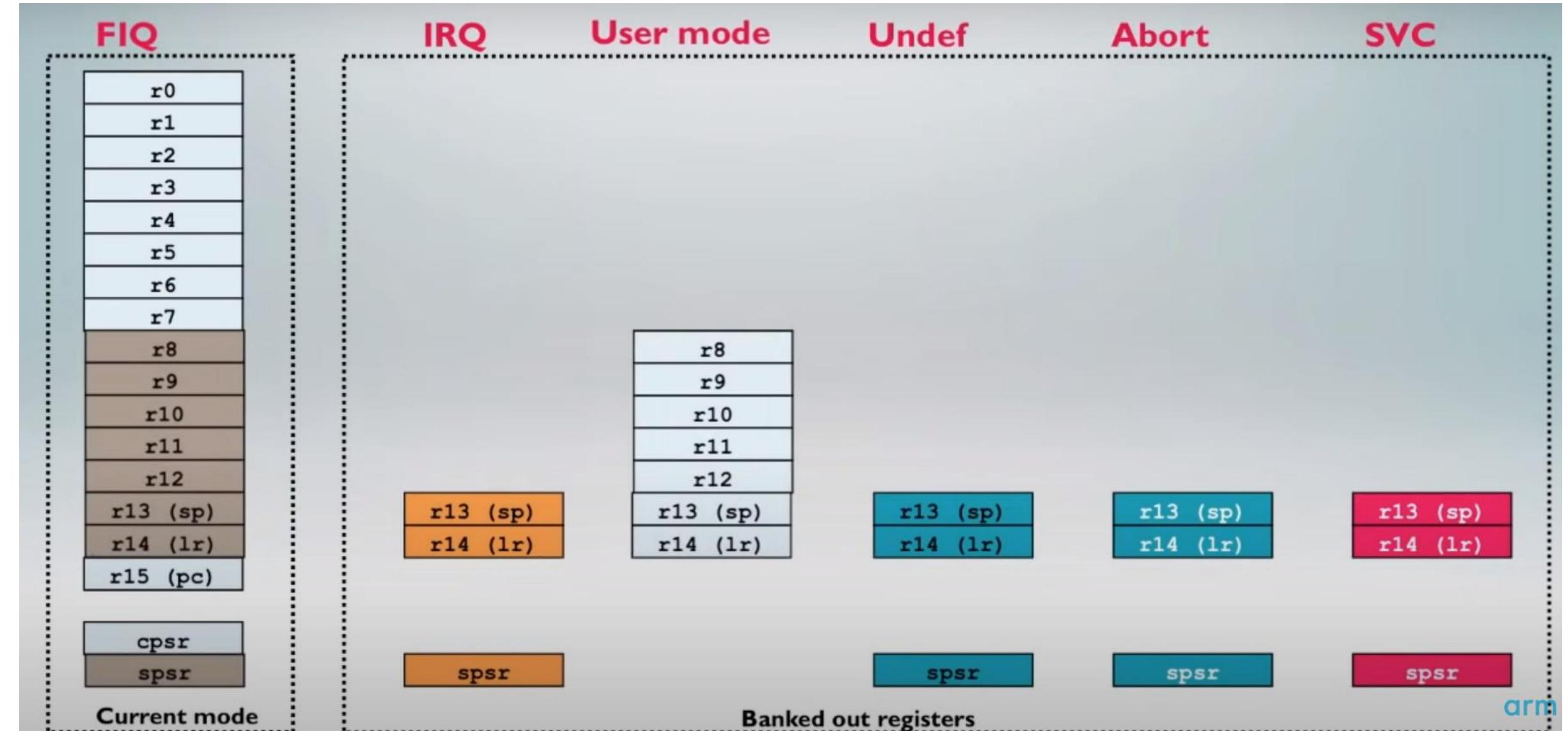
ARM Register Organization

- In user mode, 16 registers are visible (r0 to r15).
- Rest of the registers i.e. r8_fiq – r14_fiq, r13_svc etc. are banked out or invisible. These registers will become visible only in their respective modes.



- For ex. When processor enters IRQ mode, the registers r13 and r14 will be replaced by r13_irq and r14_irq.
- Thus, every mode in ARM has its own stack pointer (r13) and link register (r14).

ARM Register Organization



Register organization in FIQ mode will look like this

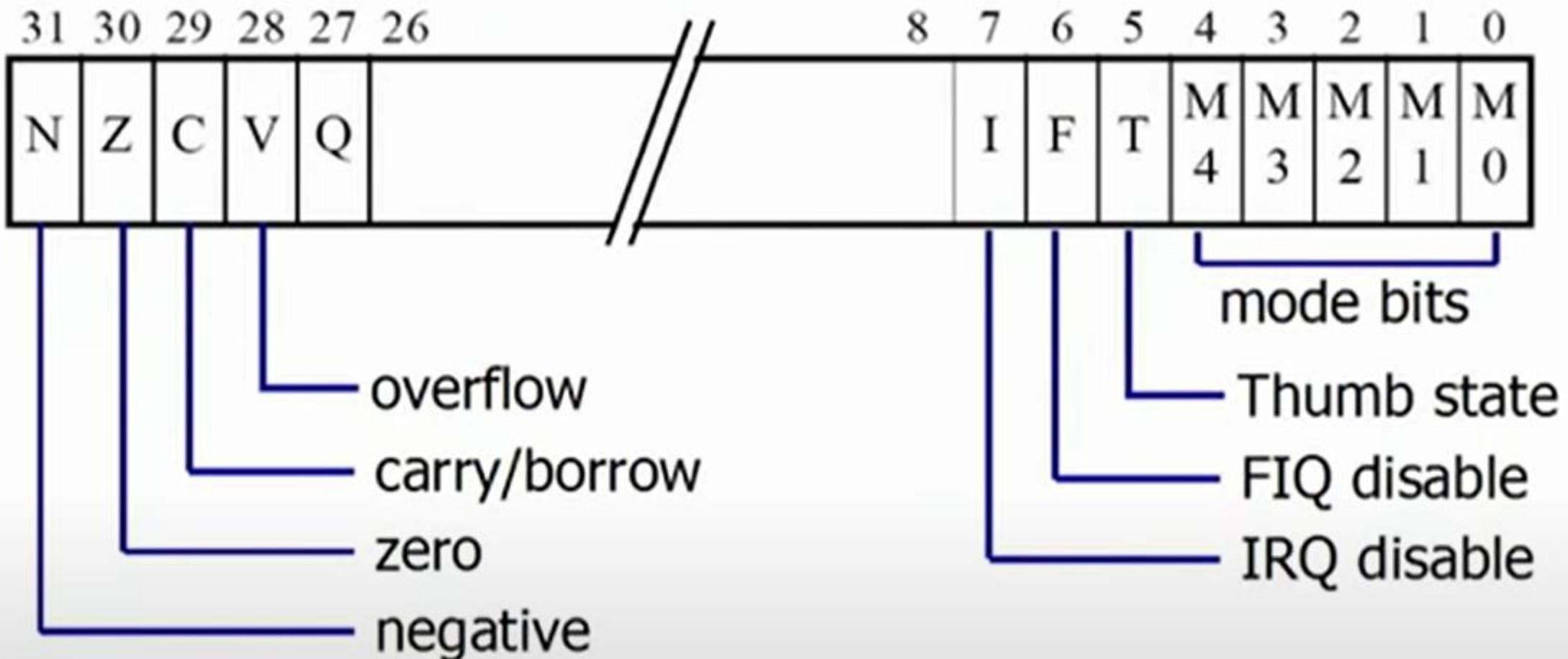


Note: System mode uses the User mode register set

ARM Registers - Program Counter

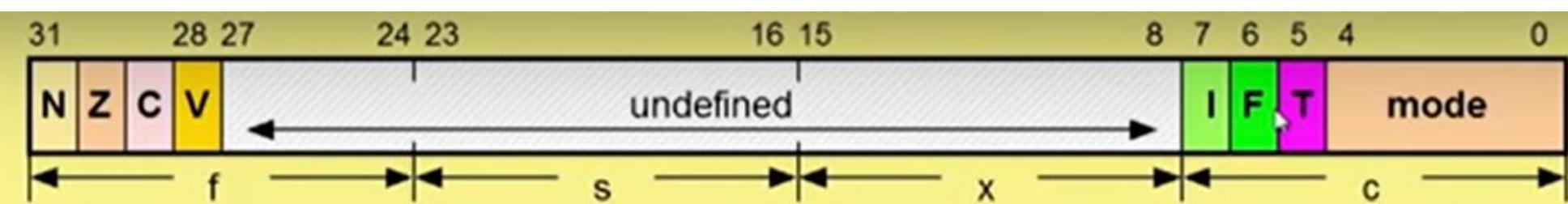
- When the processor is executing in *ARM mode*:
 - All the instructions are 32-bit wide and word aligned i.e. the address of each instruction is a multiple of 4 – last 2 bits of address are 0
 - Thus, last 2 bits of PC are 0
 - Due to pipelining, PC points 8 bytes (2 instructions) ahead of the current instruction
- When the processor is executing in *Thumb mode*:
 - All instructions are 16-bit wide and half-word aligned i.e. the address of each instruction is a multiple of 2 – last 1 bit of the address is ‘0’
 - Thus, last bit of the PC is ‘0’

ARM Registers – CPSR structure



- Mode bits are used to specify in which mode (out of 7) ARM is operating.

CPSR structure



Condition code flags

- N = Negative result from ALU
- Z = Zero result from ALU
- C = ALU operation Carried out
- V = ALU operation oVerflowed

Interrupt Disable bits.

- I = 1: Disables the IRQ.
- F = 1: Disables the FIQ.

T Bit (Arch. with Thumb mode only)

- T = 0: Processor in ARM state
- T = 1: Processor in Thumb state

Mode bits

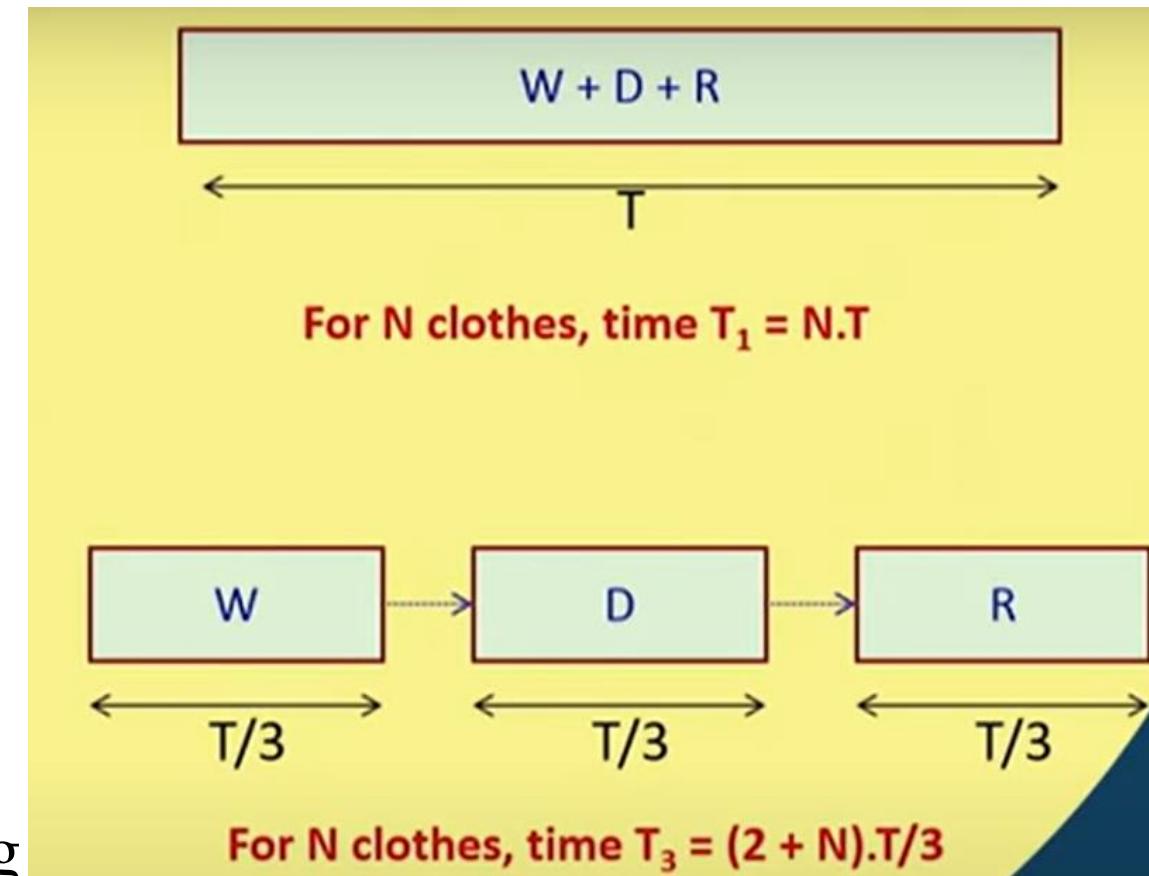
10000	User
10001	FIQ
10010	IRQ
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

Pipelining

- Pipelining is a mechanism of overlapped execution of several instructions by partitioning some execution into a set of k sub-computations (or stages).
- Pipelining can be carried out on instruction execution, memory access and arithmetic computations.
- Advantages of pipelining are
 - Very significant speedup in operation of processors (almost k times)
 - Very nominal increase in the cost of implementation

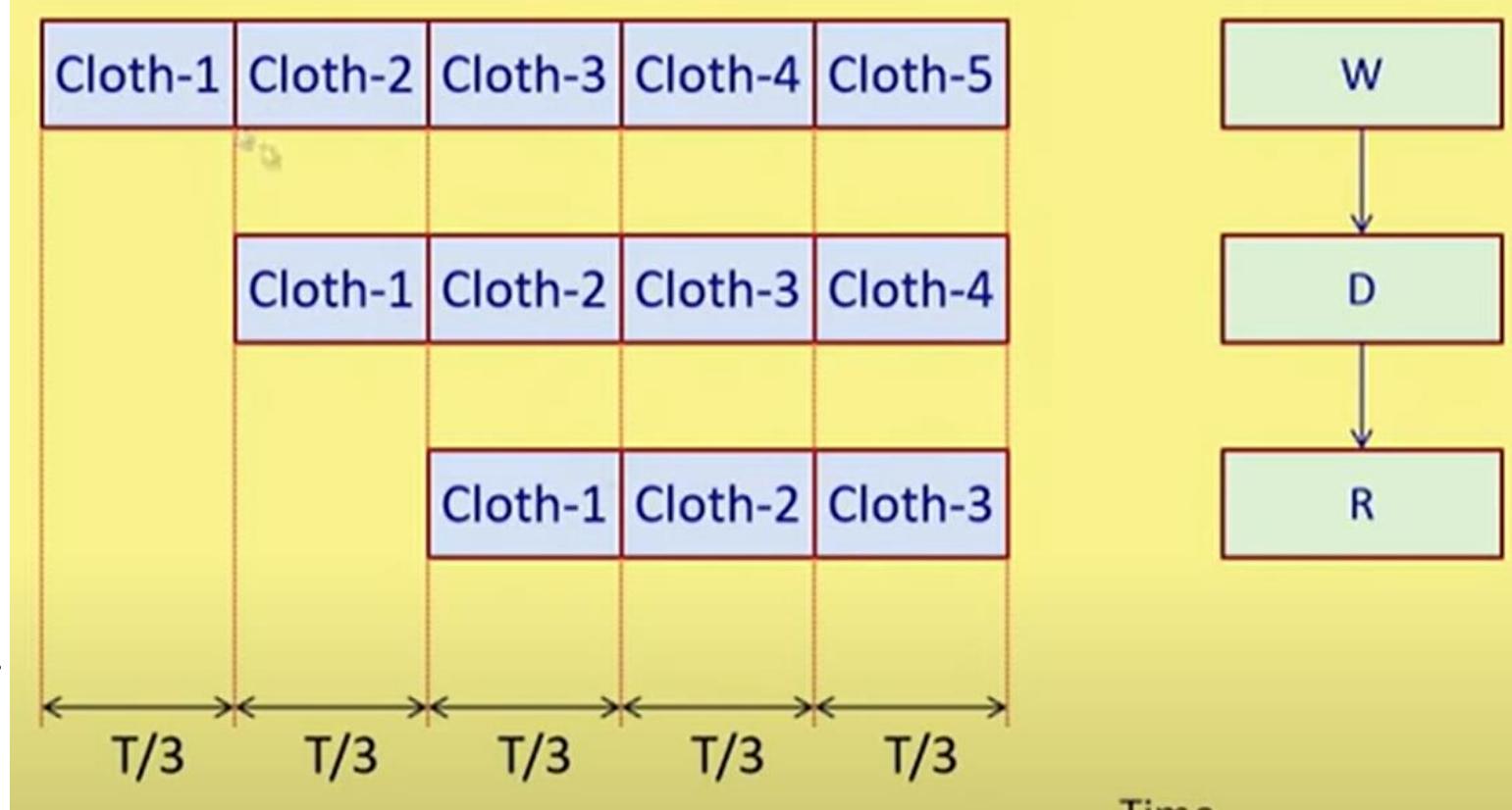
Pipelining

- Suppose there is a machine which can perform 3 operations on clothes – wash (W), dry (D) and iron (R).
- This can be implemented either by a single machine implementing all the tasks or by a set of 3 different machines – each implementing
- The single machine completes the task in time ‘T’ for 1 cloth and hence time required for N clothes is ‘N.T’



Pipelining

- If we divide the tasks into 3 machines, each machine requires ' $T/3$ ' time to complete its task and thus total time required for N clothes is reduced to $(2+N) \cdot T/3$

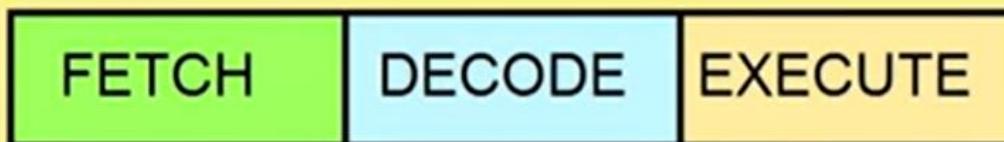


- This is because, each of the machines here is working parallelly as shown.
- In first $T/3$ duration, cloth 1 is being washed. In 2^{nd} $T/3$ duration, cloth 1 is being dried and at the same time cloth 2 is being washed. Similarly, in 3^{rd} $T/3$ duration, cloth 1 is being ironed, cloth 2 is being dried and cloth 3 is being washed.

Pipelining

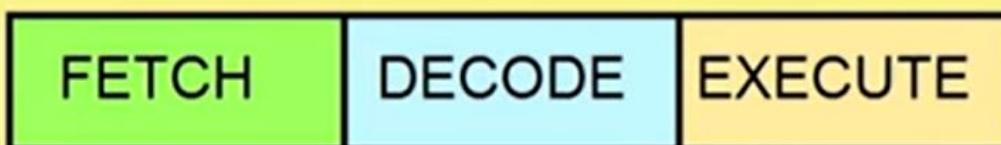
Pipelining in ARM7

1

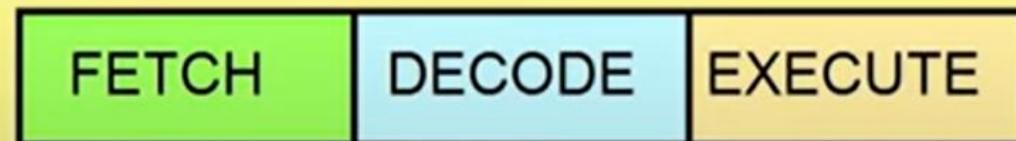


Simple instructions (like ADD, SUB) can complete at a rate of one instruction per cycle.

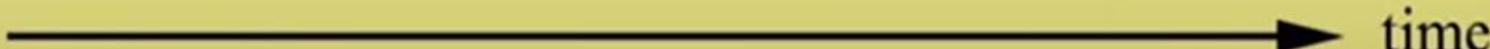
2



3



instruction

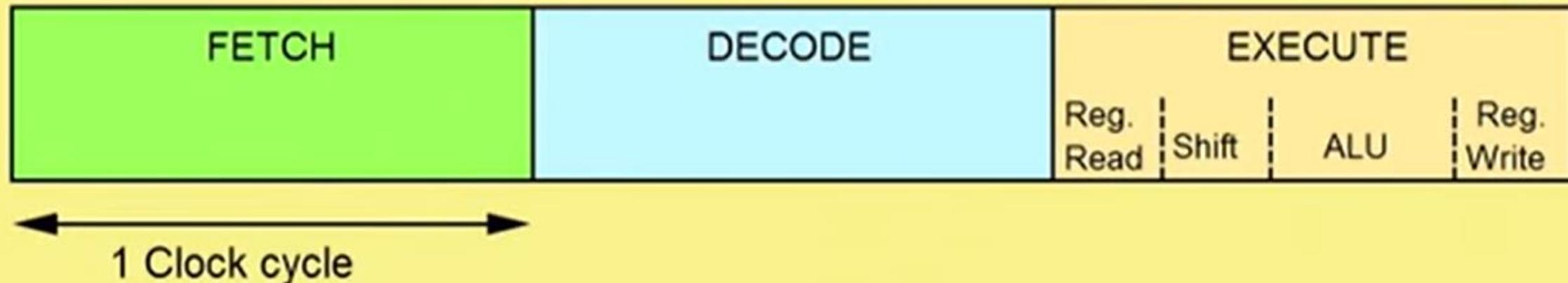


Pipelining in ARM

- Pipelining is performed in ARM processors on the fetching, decoding and execution of the instructions.
- In 1st clock cycle, 1st instruction is fetched. In 2nd clock cycle, 1st instruction is decoded and 2nd instruction is fetched. Similarly, in 3rd clock cycle, 1st instruction is executed, 2nd is decoded and 3rd instruction is fetched.
- Buffers – in order to execute pipelining efficiently, data from one stage needs to be stored in a temporary storage before being fed to the next stage. This is implemented using buffers (latches).
- Different generations of ARM processors have different stages of pipelining.

Pipelining in ARM

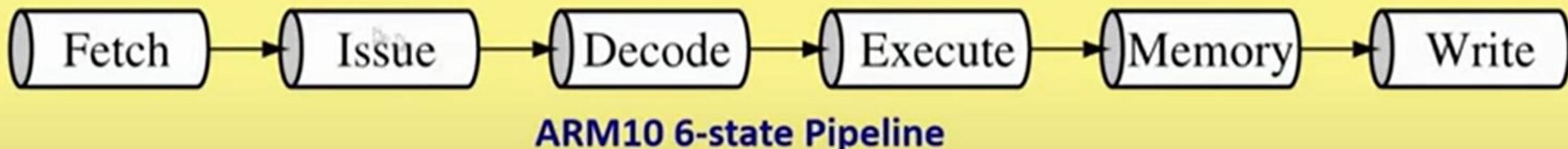
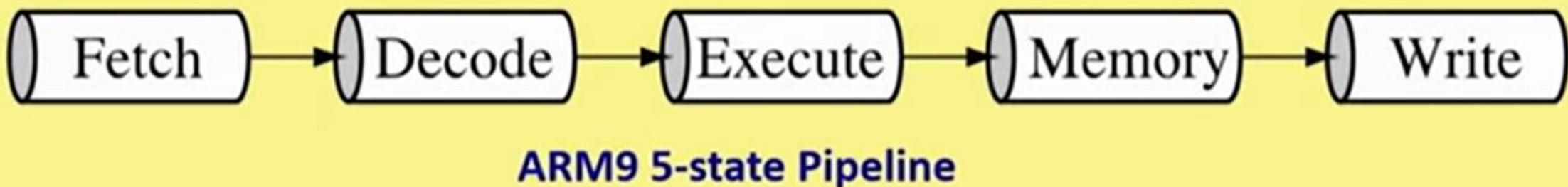
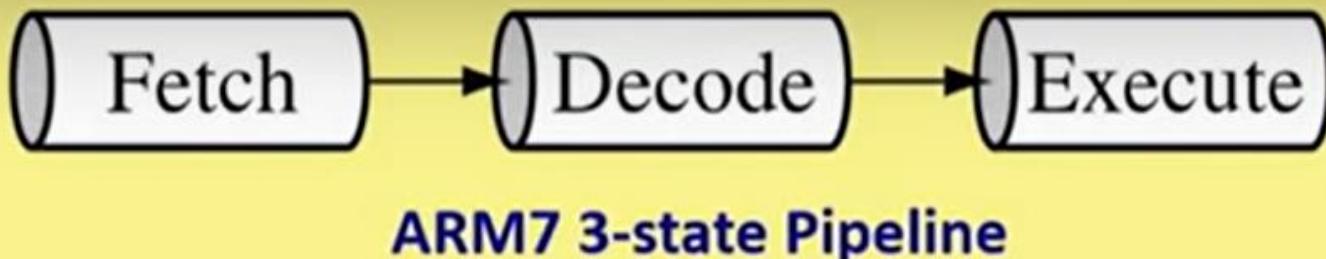
ARM7TDMI Pipeline



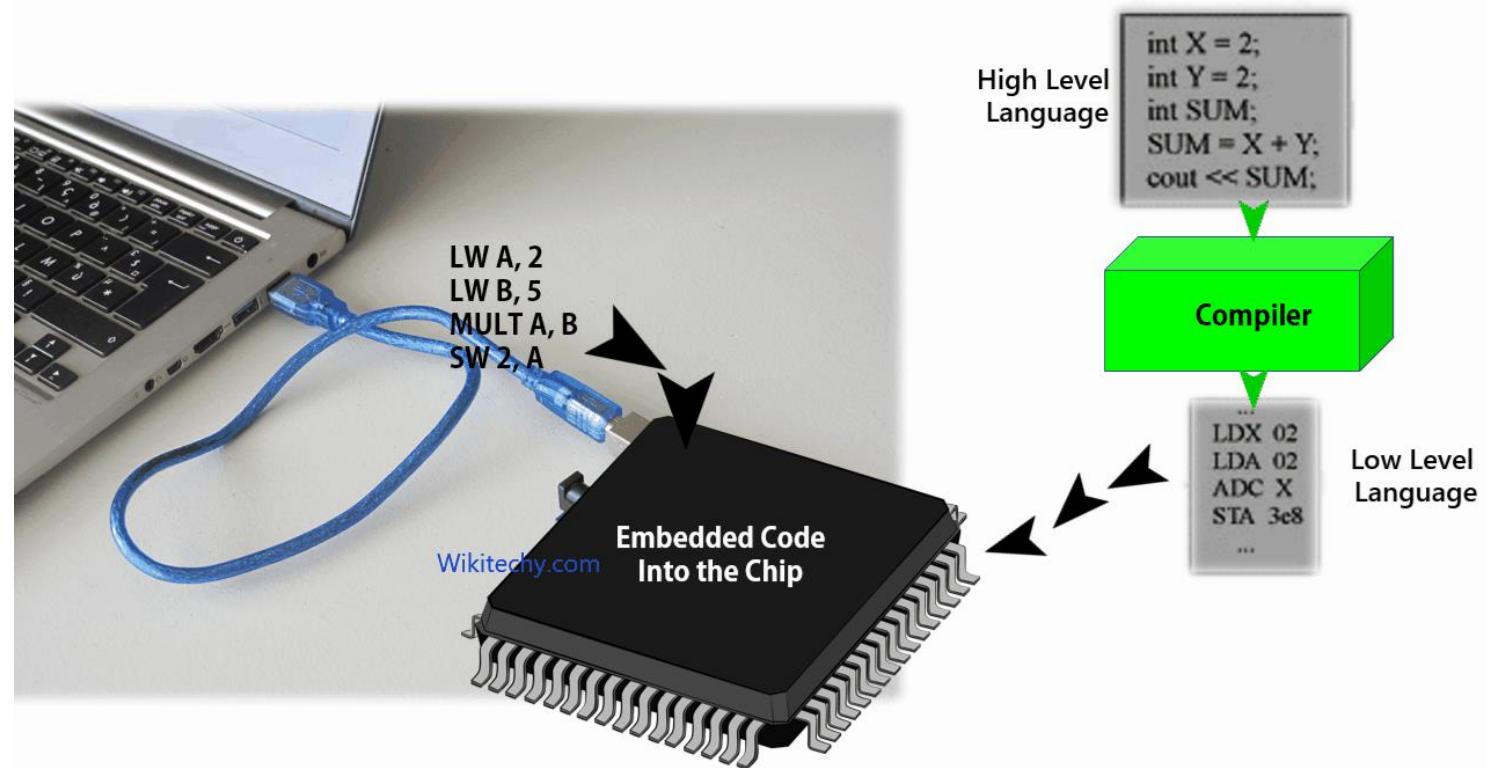
ARM9TDMI Pipeline



Pipelining in ARM



ARM Instruction Set



The instructions in ARM can be broadly classified into 3 types:

- Data Processing Instructions – operate only on the registers and not on memory
- Data Transfer Instructions – operate on both the registers and memory
- Control Flow Instructions – alter the value of PC, used to execute subroutine

(https://www.youtube.com/watch?v=CuuIBvHrvtA&ab_channel=IITKharagpurJuly2018)

ARM Instruction Set – Arithmetic Instructions

Data Processing Instructions – operate only on the registers and not on mem

ADD	r0, r1, r2	; r0 = r1 + r2
ADC	r0, r1, r2	; r0 = r1 + r2 + C (C is carry bit)
SUB	r0, r1, r2	; r0 = r1 - r2
SBC	r0, r1, r2	; r0 = r1 - r2 + C - 1
RSB	r0, r1, r2	; r0 = r2 - r1
RSC	r0, r1, r2	; r0 = r2 - r1 + C - 1

ARM Instruction Set – Multiply Instructions

- When two 32-bit numbers are multiplied using the ‘MUL’ instruction, the result is 64-bit, but only the least significant 32 bits [31:0] are stored.

```
MUL    r1,r2,r3 ; r1 = (r2 x r3) [31:0]
```

- Immediate operands are not supported in MUL instruction.

```
MLA    r1,r2,r3,r4 ; r1 = (r2 x r3 + r4) [31:0]
```

- MLA – Multiply and Accumulate – mostly used in DSP applications.

ARM Instruction Set – Logical Instructions

Bit-wise Logical Instructions – operate on each bit of the specified registers

BIC – Bit Clear instruction

AND	r0, r1, r2	; r0 = r1 and r2
ORR	r0, r1, r2	; r0 = r1 or r2
EOR	r0, r1, r2	; r0 = r1 xor r2
BIC	r0, r1, r2	; r0 = r1 and not r2

ARM Instruction Set – Immediate Operands

- Arithmetic and Logical operations can also be performed using numbers (called immediate operands) instead of registers, as shown

ADD	r1, r2, #2	; r1 = r2 + 2
SUB	r3, r3, #1	; r3 = r3 - 1
AND	r6, r4, #&0f	; r6 = r4[3:0]

Here, ‘#’ indicates immediate operand and ‘&’ denotes hexadecimal notation.

- Last instruction performs AND of the contents of register r4 and hexadecimal number 0f.

ARM Instruction Set – Comparison Instructions

- CPSR register has several flag bits – S (sign), C (carry), Z (zero), V (overflow).
- Comparison instructions affect the value of these bits
- Thus, result of comparison instructions are not stored in any register only the value of flag bits are modified accordingly.

CMP	r1, r2	; set cc on (r1 - r2)
CMN	r1, r2	; set cc on (r1 + r2)
TST	r1, r2	; set cc on (r1 and r2)
TEQ	r1, r2	; set cc on (r1 xor r2)

ARM Instruction Set – Move Instructions

- Move instructions are data transfer instructions.
- MVN – Move Negated.

```
MOV    r0,r2          ; r0 = r2
MVN    r0,r2          ; r0 = not r2
```

ARM Instruction Set – Shifted Register Operands

- The second operand in ARM can be shifted by specified bits as shown below.

```
ADD r1,r2,r3,LSL #3 ; r1 = r2 + (r3 << 3)
```

```
ADD r1,r2,r3,LSL r5 ; r1 = r2 + (r3 << r5)
```

- In first instruction, r3 is logically left shifted by 3 places while in second instruction, r3 is logically left shifted by the value of r5.

- **LSL**: logical shift left

- **ASL**: arithmetic shift left

- **LSR**: logical shift right

- **ASR**: arithmetic shift right

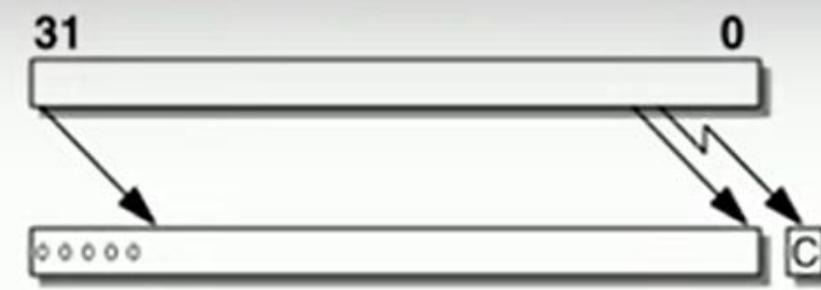
- **ROR**: rotate right

- **RRX**: rotate right extended by 1 bit

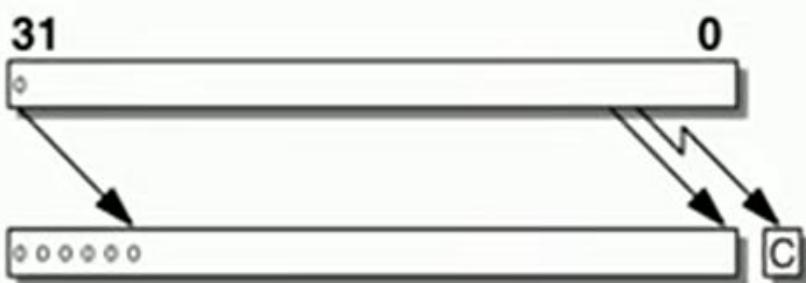
ARM



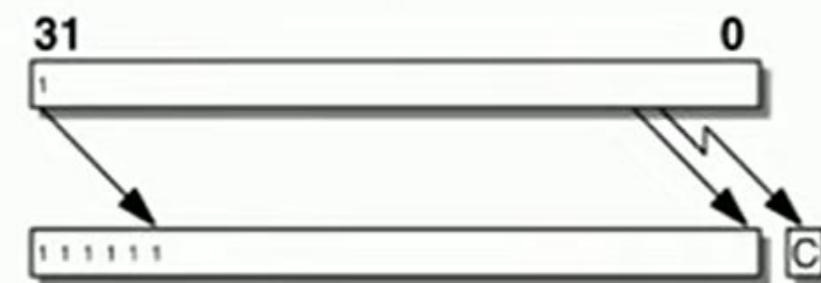
LSL # 5



LSR # 5



ASR # 5 – positive operand



ASR # 5 – negative operand



ROR # 5



RRX

ARM Instruction Set – Move Instructions

- In LSL and LSR, ‘0’ is added in the place of shifted value
- In ASR – if the MSB is ‘0’ (i.e. positive integer), then ‘0s’ are added in the shifted places and if the MSB is ‘1’ (i.e. negative integer), then ‘1’s are added in the shifted places.
- ROR performs rotation without using the carry bit (C) while RRX performs rotation using the carry bit.

ARM Exception Priority

- The exceptions in ARM have the priority as shown in the adjacent table.
- Reset has the highest priority – when Reset occurs, ARM branches to Reset handler and “I” and “F” bits in CPSR are both set to “1” i.e. both Fast Interrupt and Normal Interrupt are disabled.
- When Data Abort occurs, I bit is set to “1” i.e. only Normal Interrupt is disabled while Fast Interrupt is still enabled and it can be processed.

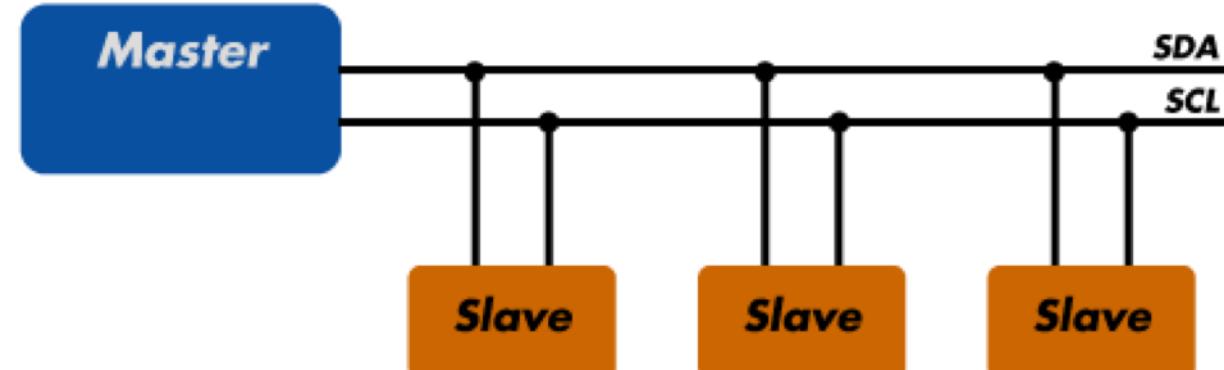
Exceptions	Priority	I bit	F bit
Reset	1	1	1
Data Abort	2	1	—
Fast Interrupt Request	3	1	1
Interrupt Request	4	1	—
Prefetch Abort	5	1	—
Software Interrupt	6	1	—
Undefined Instruction	6	1	—

Protocols – SPI, I2C, UART

- https://www.youtube.com/watch?v=U5CDf4TNARE&ab_channel=nptelhrd
- https://www.youtube.com/watch?v=tEvtb-mdJ4s&t=2094s&ab_channel=nptelhrd

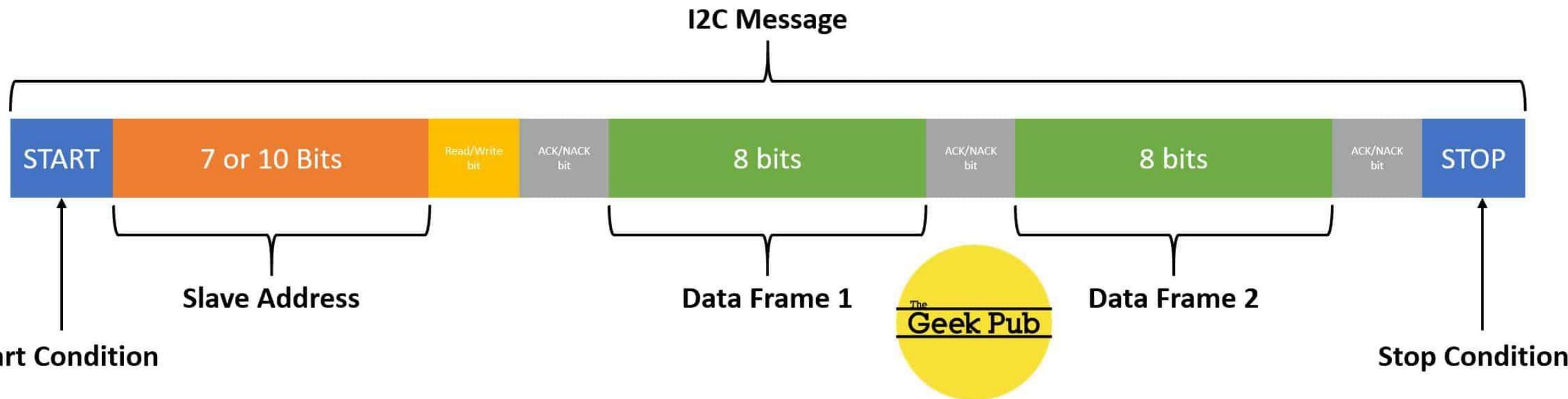
I2C (Inter Integrated Circuit)

- **Serial communication** protocol used to communicate between different ICs on a single board or between a microcontroller and different peripheral devices
- **Simple, low bandwidth** (500 kbps – few Mbps), **short distance, bidirectional but half-duplex, synchronous** protocol
- Used to link multiple devices together since it has a built-in addressing scheme
- There are 2 signal lines – SCL (Serial Clock) and SDA (Serial Data)
- Data is sent on the SDA line and SCL line is used for synchronization between the communicating devices.



I2C

- I2C message consists of the following fields
- The master generates the start and stop bits to denote the beginning and the end of message transfer.
- Data can be sent by the master (write) or by the slave (read).



I2C

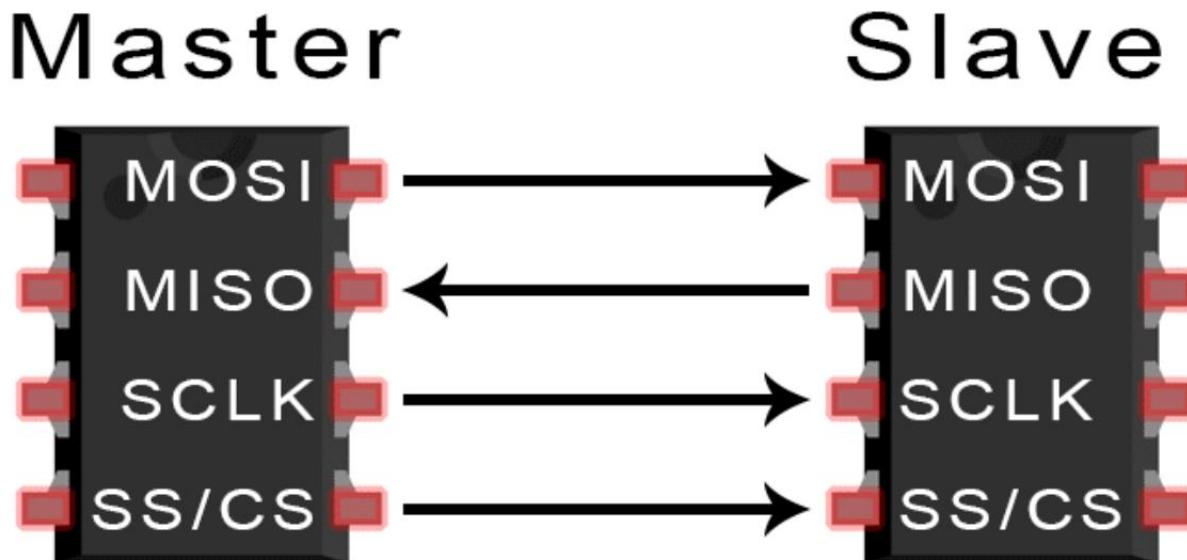
- After the Start bit, master sends the address of the slave with which it wants to communicate.
- All the slaves connected to a master then compare this address with their own address and the one with a match replies with an ACK message.
- Data is sent in packets of 9 bits – 8 bit of message and 1 ACK bit which is send by the receiver to the transmitter denoting successful reception of message.
- **Clock Stretching** – if the slave is not ready to accept more data from the master, it can **hold the SCL line low**, hence disabling the master to raise the SCL line and sending more data

I2C

- I2C protocol supports multiple masters and multiple slaves.
- Every microcontroller has an I2C interface which enables it use I2C protocol.
- Data frame of I2C protocol is fixed – 8 bits hence bigger message need to be broken down into smaller messages.
- Messages are always sent in MSB-first format.

SPI – Serial Peripheral Interface

- SPI allows **half/full duplex, synchronous, fast, serial** communication with external devices.
- Operates in Master – Slave configuration
- Provides 8 or 16-bit transfer frame format selection
- Programmable data order with MSB first or LSB first shifting.
- SPI can transfer data continuously and without interruption i.e. there are no Start and Stop bits present in the frame.
- There is no slave addressing system.



SPI – Serial Peripheral Interface

- SPI needs 5 pins
 1. MISO (Master In Slave Out) – this pin can be used to receive data in master mode and transmit data in slave mode.
 2. MOSI (Master Out Slave In) – this pin can be used to transmit data in master mode and receive data in slave mode
 3. SCK (Serial Clock) – a device configured as master will generate and control the clock for synchronization
 4. SS (Slave Select) – used to select the slave for communication
 5. GND

SPI – Serial Peripheral Interface

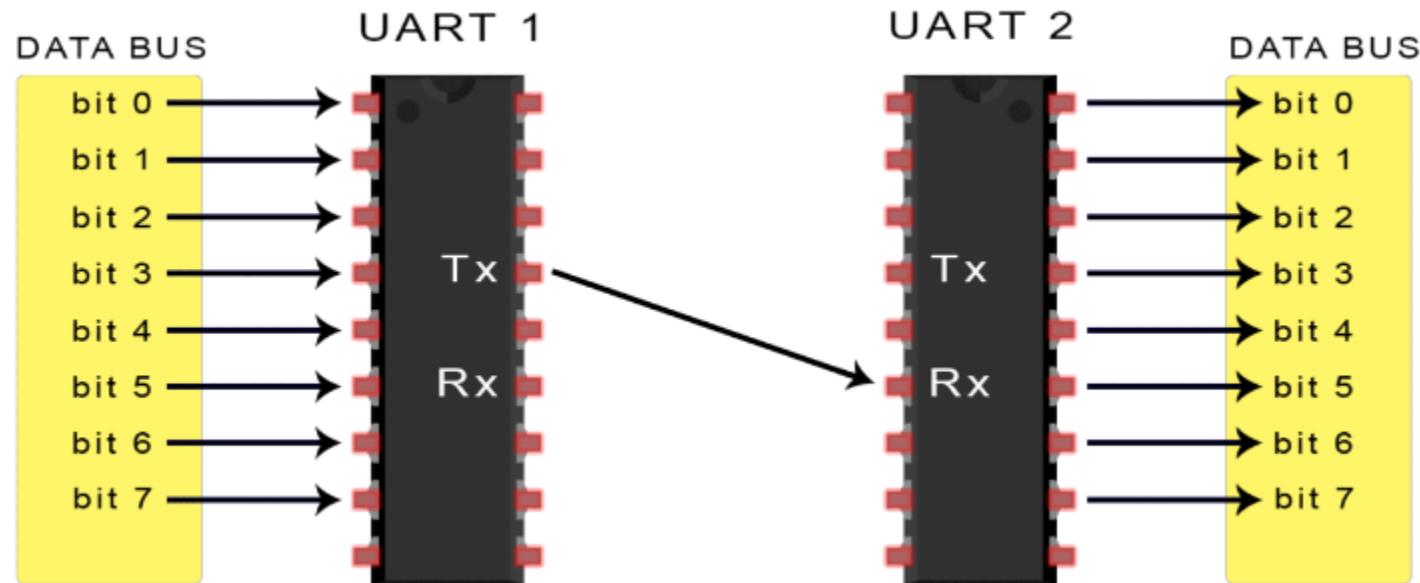
- A device configured as Master will transmit data on the MOSI pin and receive data on the MISO pin.
- A device configured as Slave will receive data on the MOSI pin and transmit data on the MISO pin.
- Thus MOSI pins and MISO pins of master and slave devices will be connected together. This makes the full-duplex communication possible.
- Most important drawback of SPI protocol is the number of pins required for data transfer. This limits the number of peripheral devices that can be connected to a single microcontroller.

UART – Universal Asynchronous Receiver / Transmitter

- UART transmits the data asynchronously – there is no clock signal present.

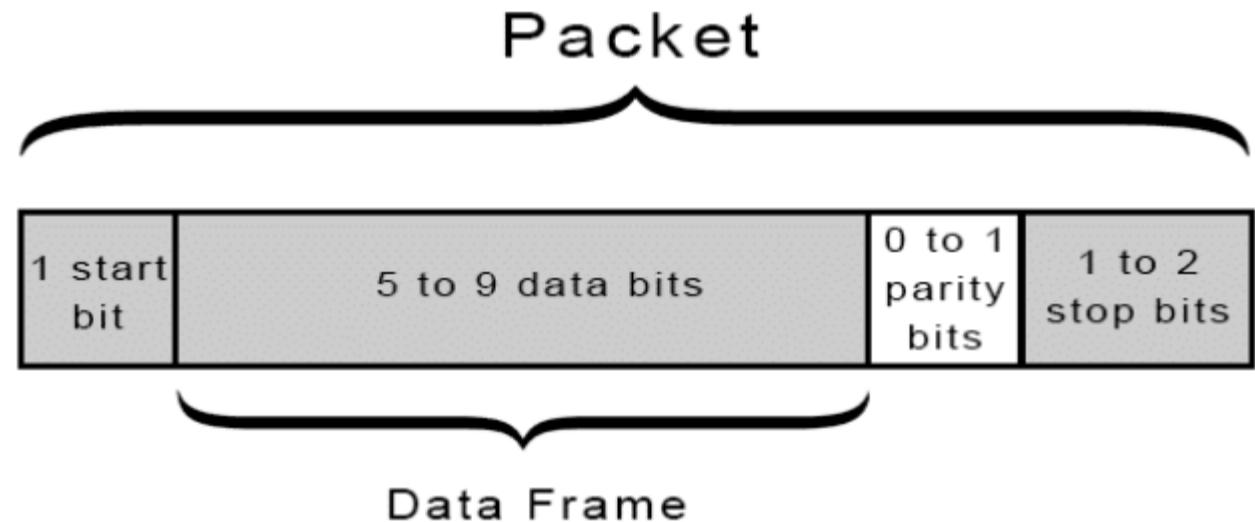
- UART has only 2 signal lines Transmit (Tx) and Receive (Rx).

- The transmitting UART adds **start and stop bits** to the packet to denote the beginning and end of data transmission.
- Both transmitting and receiving UARTs work on the same frequency known as **baud rate** which is the speed of data transfer.



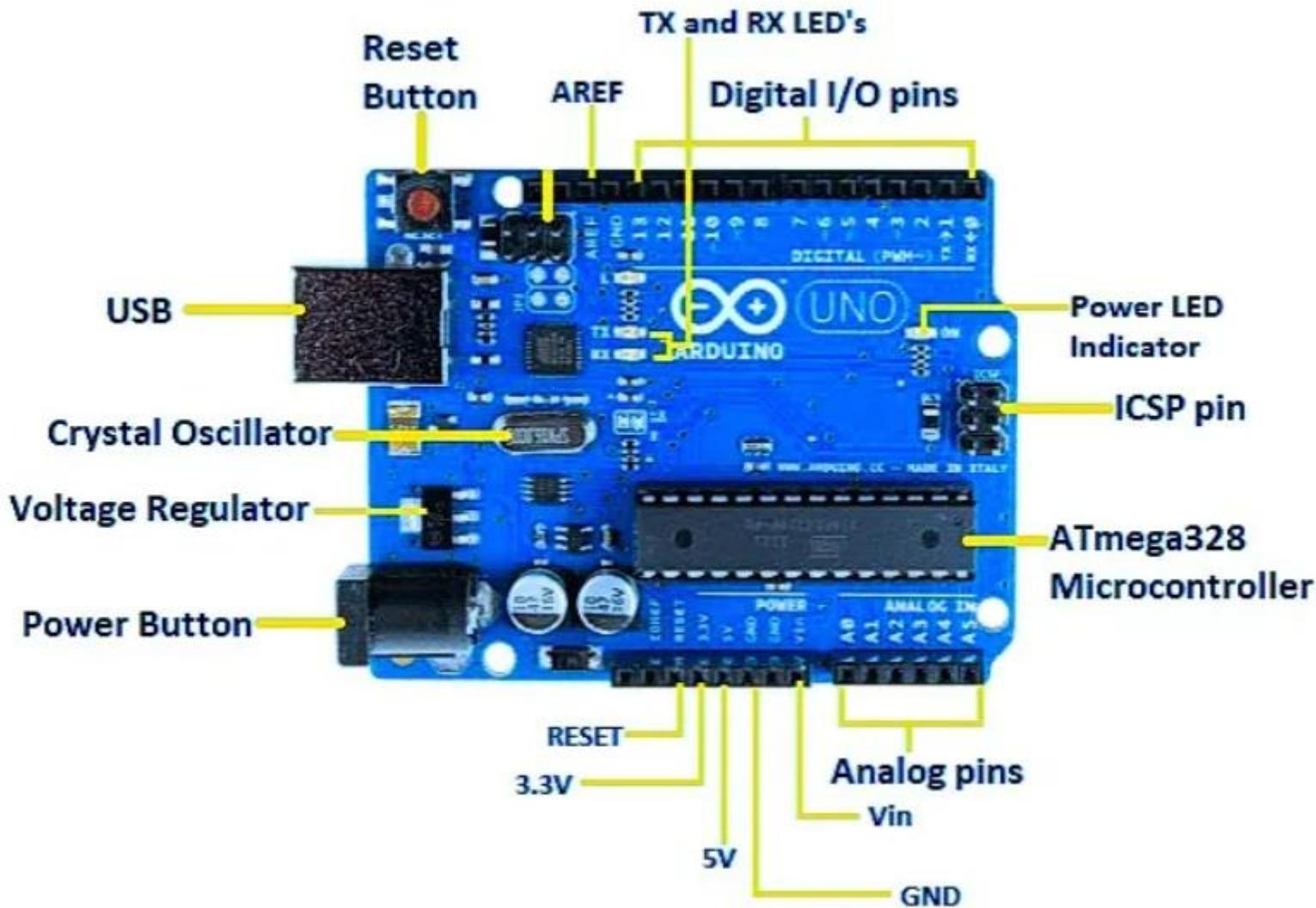
UART – Universal Asynchronous Receiver / Transmitter

- UART packet looks as shown
- Parity bit is added for error detection at the Rx end.
- Parity bit indicates whether the number of 1's in the data is ODD (0 parity) or EVEN (1 parity).
- The transmitter sets the parity bit to ‘1’ or ‘0’ depending upon the number of 1's. the receiver then counts the number of 1's and verifies it with the parity bit. If there's mismatch, an error is detected and retransmission occurs.



Arduino Boards

UNO
Board



UNO SPECS

Microcontroller	ATmega328P – 8-bit AVR family microcontroller
Operating Voltage	5V
Recommended Input Voltage	7-12V
Input Voltage Limits	6-20V
Analog Input Pins	6 (A0 – A5)
Digital I/O Pins	14 (Out of which 6 provide PWM output)
DC Current on I/O Pins	40 mA
DC Current on 3.3V Pin	50 mA
Flash Memory	32 KB (0.5 KB is used for Boot loader)
SRAM	2 KB
EEPROM	1 KB
Frequency (Clock Speed)	16 MHz

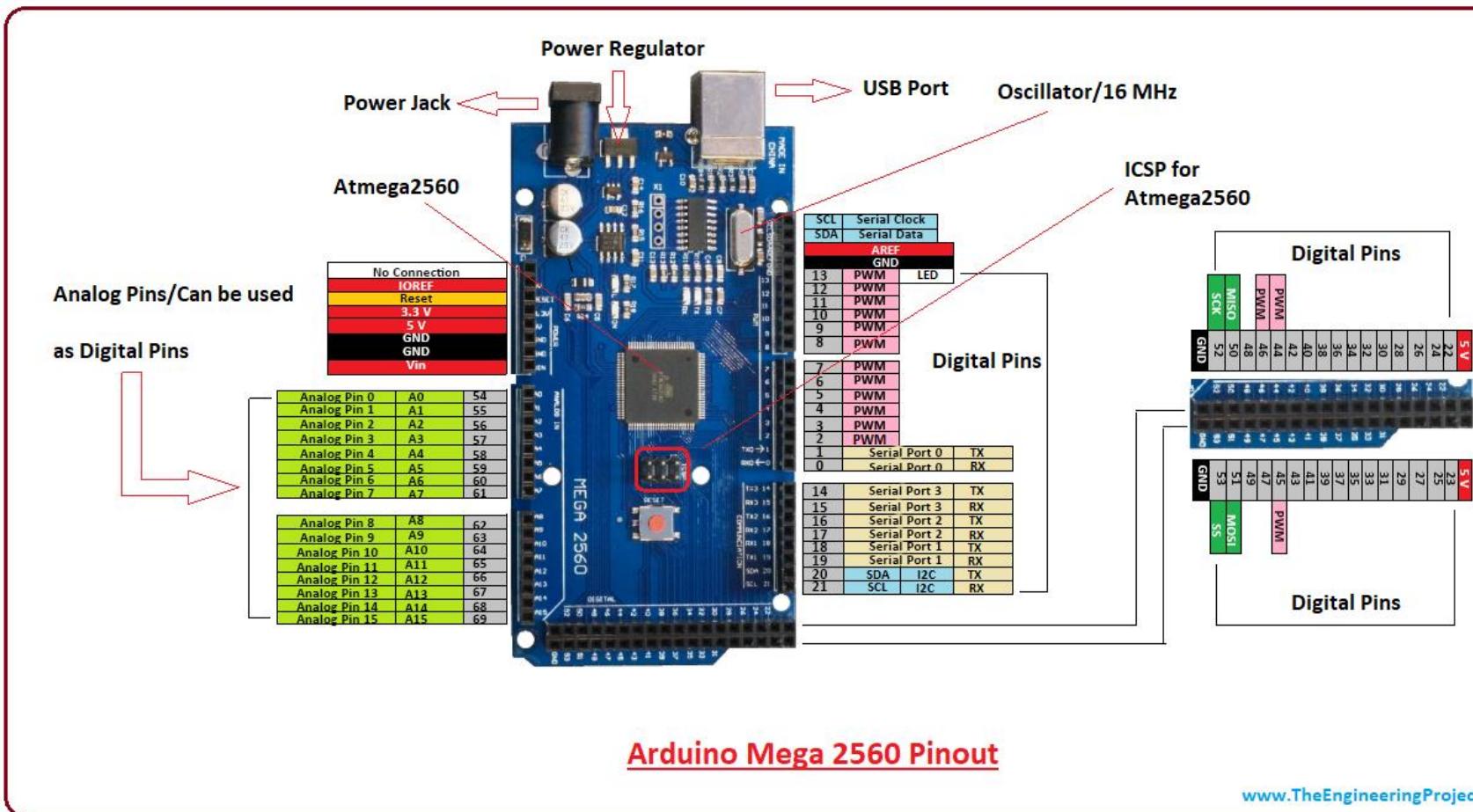
Nano SPECS



Microcontroller	ATmega328P – 8 bit AVR family microcontroller
Operating Voltage	5V
Recommended Input Voltage for Vin pin	7-12V
Analog Input Pins	6 (A0 – A5)
Digital I/O Pins	14 (Out of which 6 provide PWM output)
DC Current on I/O Pins	40 mA
DC Current on 3.3V Pin	50 mA
Flash Memory	32 KB (2 KB is used for Boot loader)
SRAM	2 KB
EEPROM	1 KB
Frequency (Clock Speed)	16 MHz
Communication	IIC, SPI, USART

Arduino Mega

- Atmega Processor
- 54 Digital I/O Pins
 - 16 Analog Input pins
 - 14 PWM pins
 - 6 UARTs
- 16 MHz Crystal Oscillator
- Flash Memory – 256 kB
- SRAM – 8 kB
- EEPROM – 4 kB

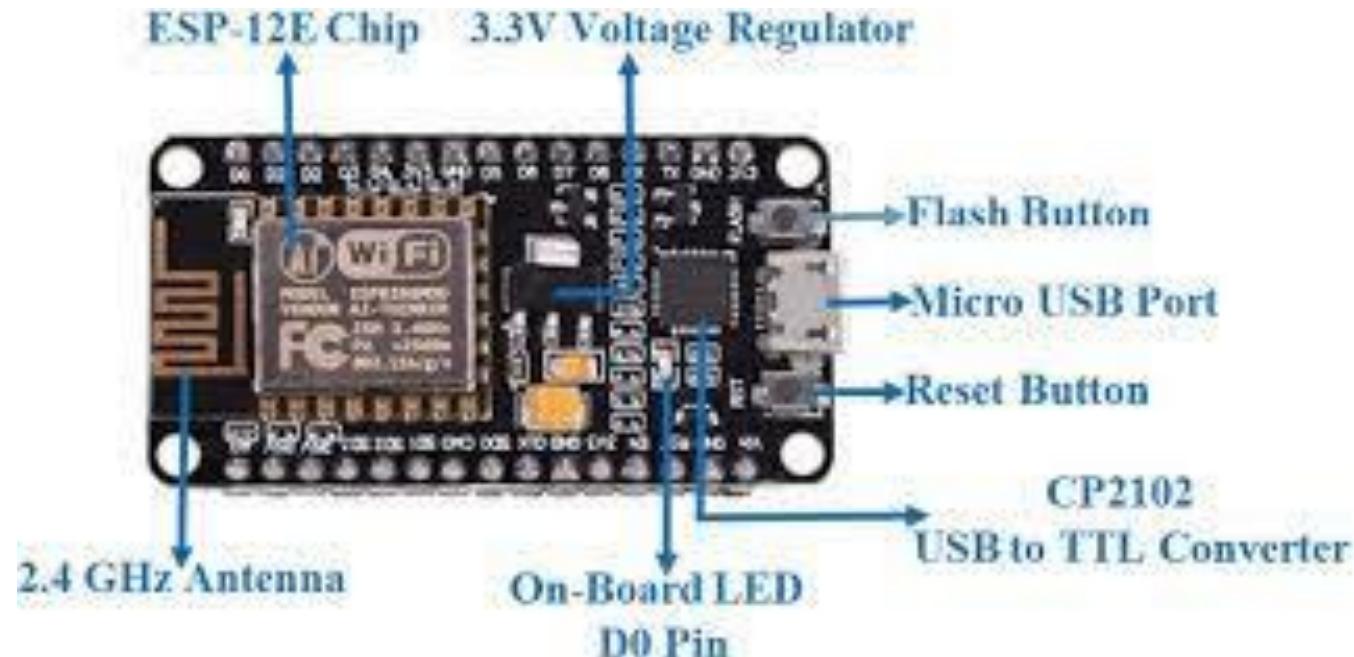


Arduino Boards

Name	Processor	Operating/Input Voltage	CPU speed	Analog In/Out	Digital IO/PWM	EEPROM / SRAM[kB]	Flash	USB	USART
Mega	ATmega2560	5V / 7-12V	16 MHz	16 / 0	54 / 15	4 / 8	256	Regular	4
Uno	ATmega328P	5V / 7-12V	16 MHz	8 / 0	14 / 6	1 / 2	32	Mini	1

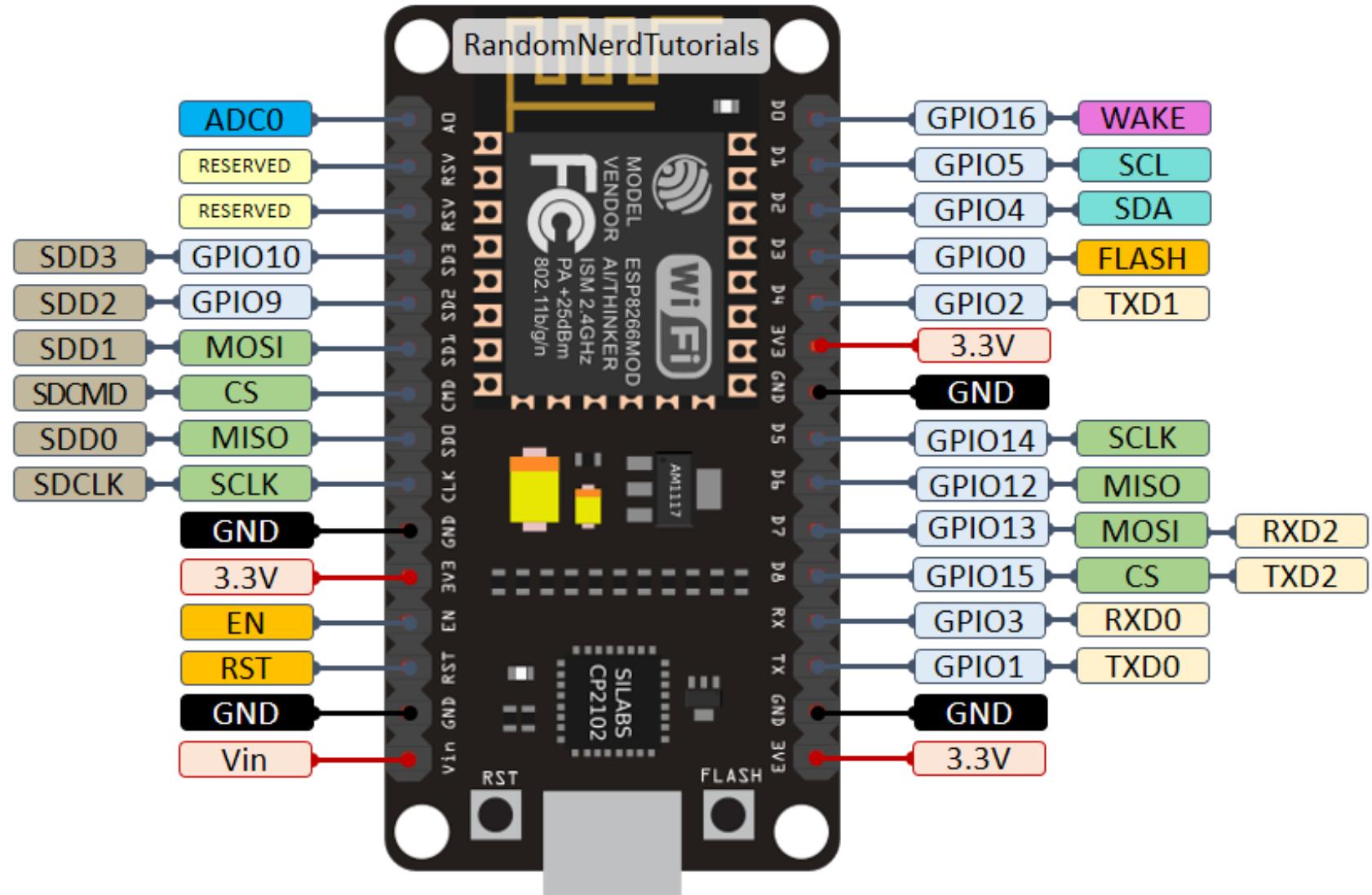
ESP8266

- 32-bit L106 processor from Tensilica
- Clock speed – 80 or 160 MHz
- 32 kB instruction RAM
- 32 kB cache RAM
- 80 kB data RAM
- WiFi module
- 16 GPIO pins
- 4 SPI pins
- 4 UART pins (TXD0, RXD0, TXD1, RXD1)
- 2 I2C pins



ESP8266

- Atmega



Working of each pins in Arduino board

- I2C – Inter IC Protocol
 - Serial communication protocol
 - Uses 2 lines SCL (Serial Clock) and SDA (Serial Data)
 - SCL is used for synchronization of data transfer
 - SDA line is used for actual data transfer
- PWM pins
 - used to get analog output with digital means.
 - Used for controlling brightness of LEDs, speed of DC motor etc.
 - Arduino digital pins give either 0 or 5 V as output which provides only 2 levels of control for LED brightness or motor speed. If we want more finer control, PWM pins can be used and duty cycle of the output can be varied.
 - If the square wave on-off voltage is varied fast enough, it gives the impression of analog signal.

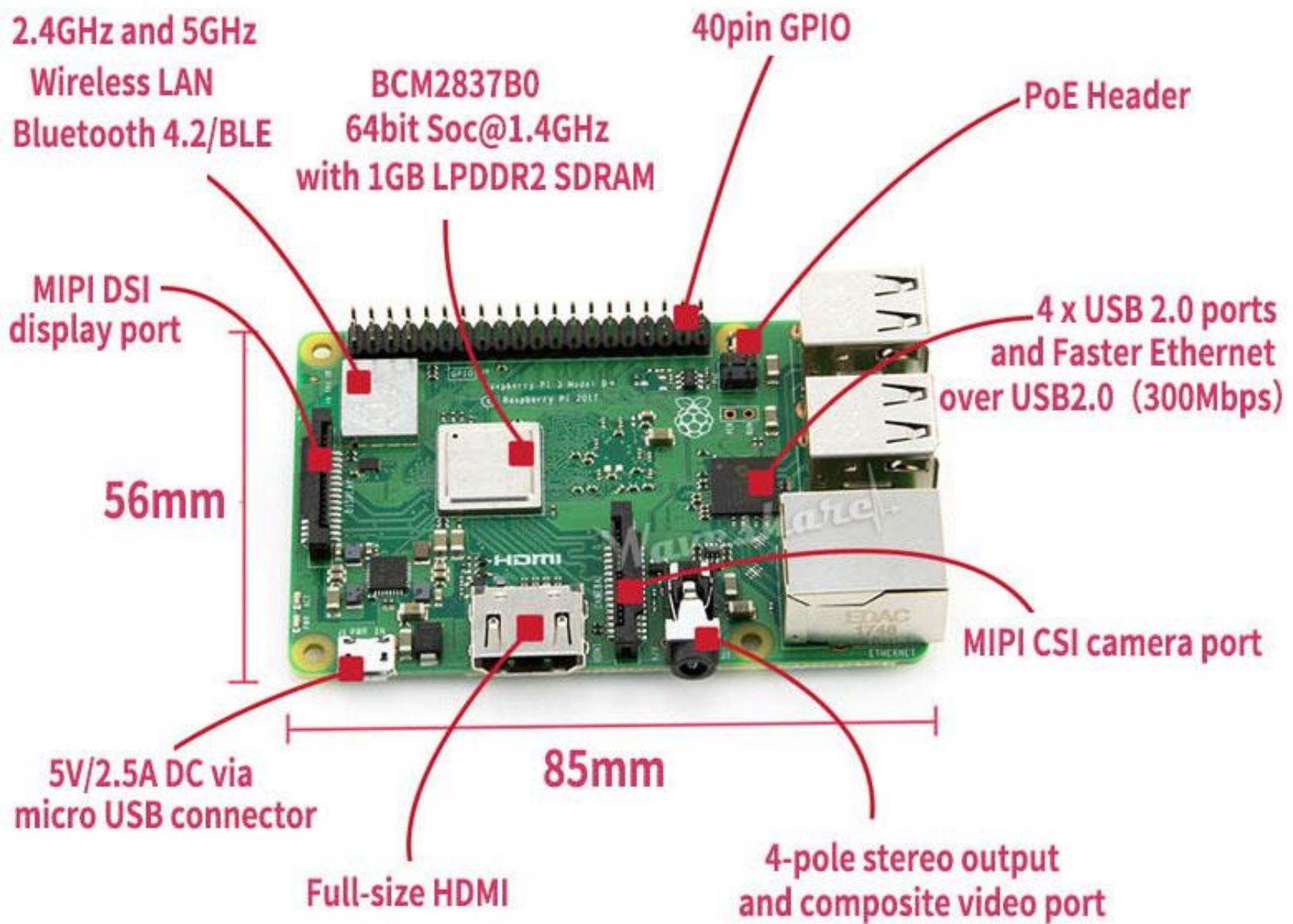
Working of each pins in Arduino board

- I2C – Inter IC Protocol
 - Serial communication protocol
 - Uses 2 lines SCL (Serial Clock) and SDA (Serial Data)
 - SCL is used for synchronization of data transfer
 - SDA line is used for actual data transfer
- PWM pins
 - used to get analog output with digital means.
 - Used for controlling brightness of LEDs, speed of DC motor etc.
 -
- UART pins
 - Used for communication between 2 MCU or computers
 - Denoted by TX and RX

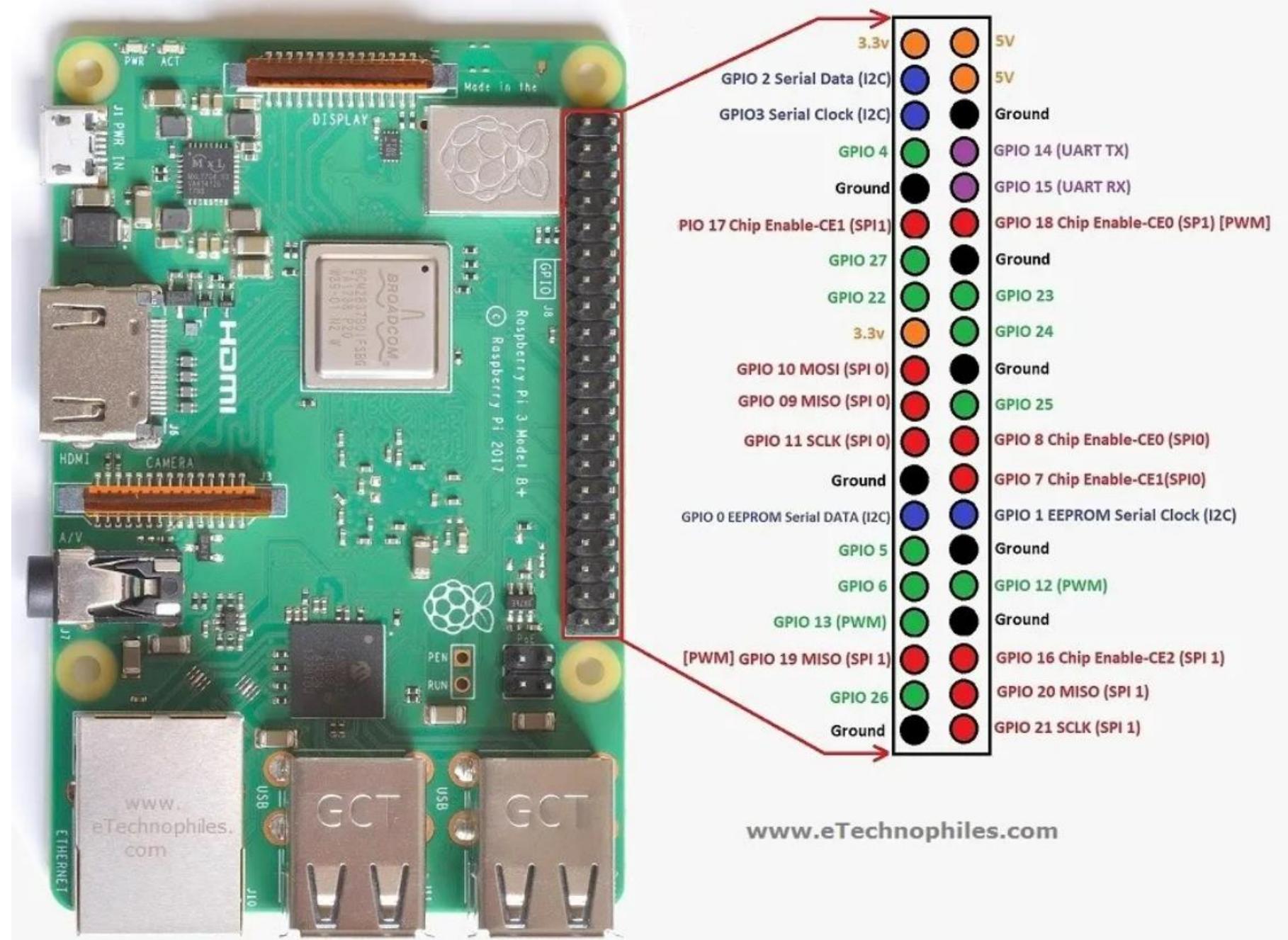
Raspberry Pi

- 1.4 GHz, 64-bit Quad-core ARM Cortex A53 Processor
- 1 GB DDR2 RAM
- WiFi – 2.4 GHz and 5 GHz
- Bluetooth 4.2
- Ethernet
- USB Ports
- 40 GPIO Pins
- Supports Linux based OS
- *Doesn't support analog inputs (doesn't have an ADC)*

Raspberry Pi 3 B+ Model



Raspberry Pi 3 B+ Model



Platform	Connectivity	Microcontroller	Cost
Arduino Yun	WiFi & Ethernet	ATmega32u4 & AtherosAR9331	\$75
Raspberry Pi	WiFi, BLE & Ethernet	64 bit ARM Cortex-A53 Quad Core	\$40
ESP8266	WiFi	Tensilica L106 32-bit	\$3
Beaglebone Black	WiFi & BLE	OSD 3358ARM 1 GHz Cortex-A8	\$70
Particle Photon	WiFi	STM32F205 120 MHz ARM Cortex M3	\$20
Arduino Nano	Nil	ATmega328	\$3

TI CC1310 – LPWAN Module

- 32 bit Cortex M3 Processor, 48 MHz

Main attraction is RF Core

- Communicates on sub-1 GHz frequencies
- Dedicated radio controller Cortex M0
- Supports multiple physical layers and RF Standards

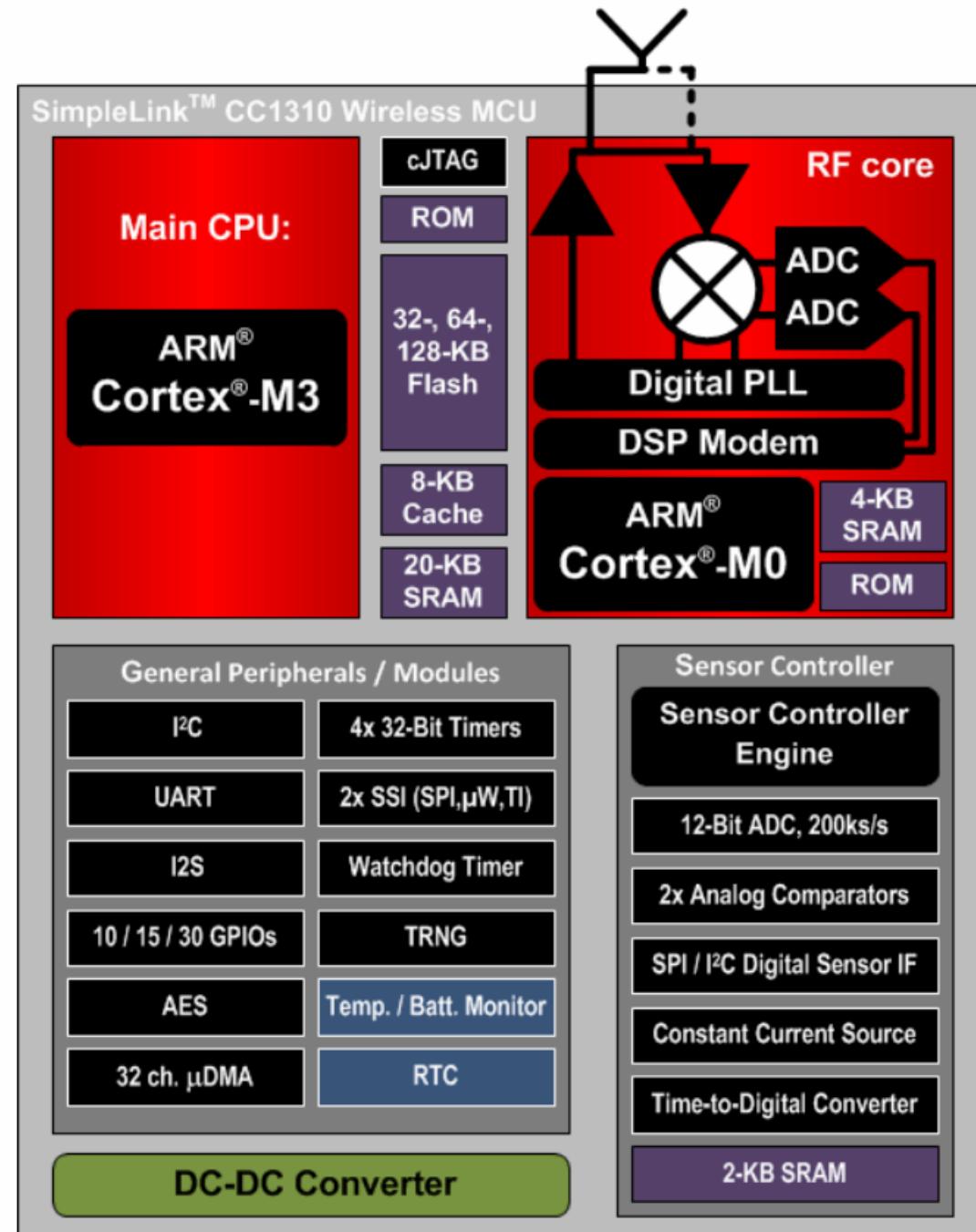


TI CC1310 – LPWAN Module

- 32 bit Cortex M3 Processor, 48 MHz

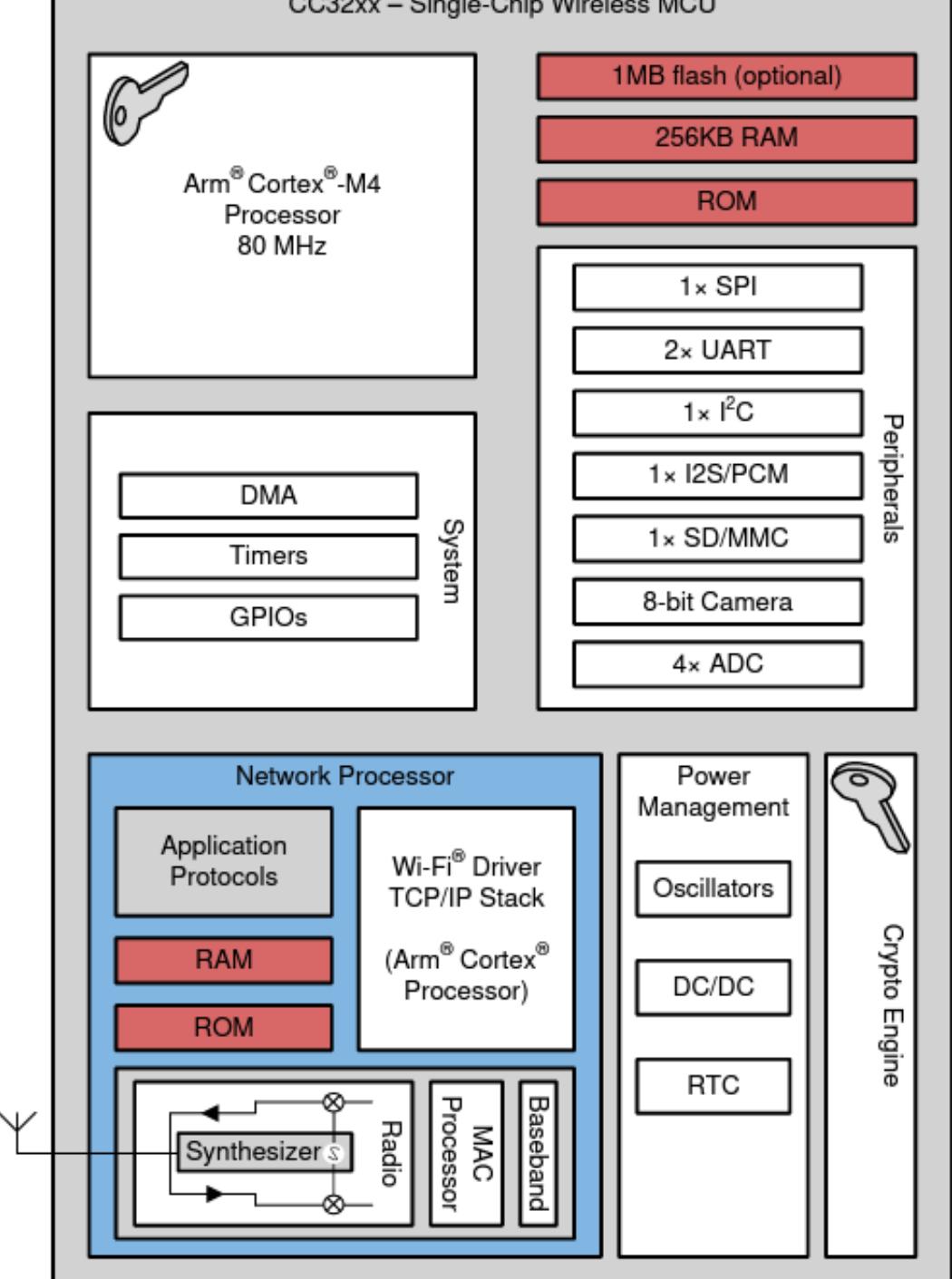
Main attraction is RF Core

- Communicates on sub-1 GHz frequencies
- Dedicated radio controller Cortex M0
- Supports multiple physical layers and RF Standards



TI CC3220SF – WiFi Module

- 32 bit Cortex M4 Processor, 80 MHz
- WiFi Direct
- Access Point support
- IPv6 TCP/IP Stack
- RESTful API Support
- Secure Sockets (SSLv3, TLS 1.0, TLS 1.1, TLS 1.2)



Selecting a MCU

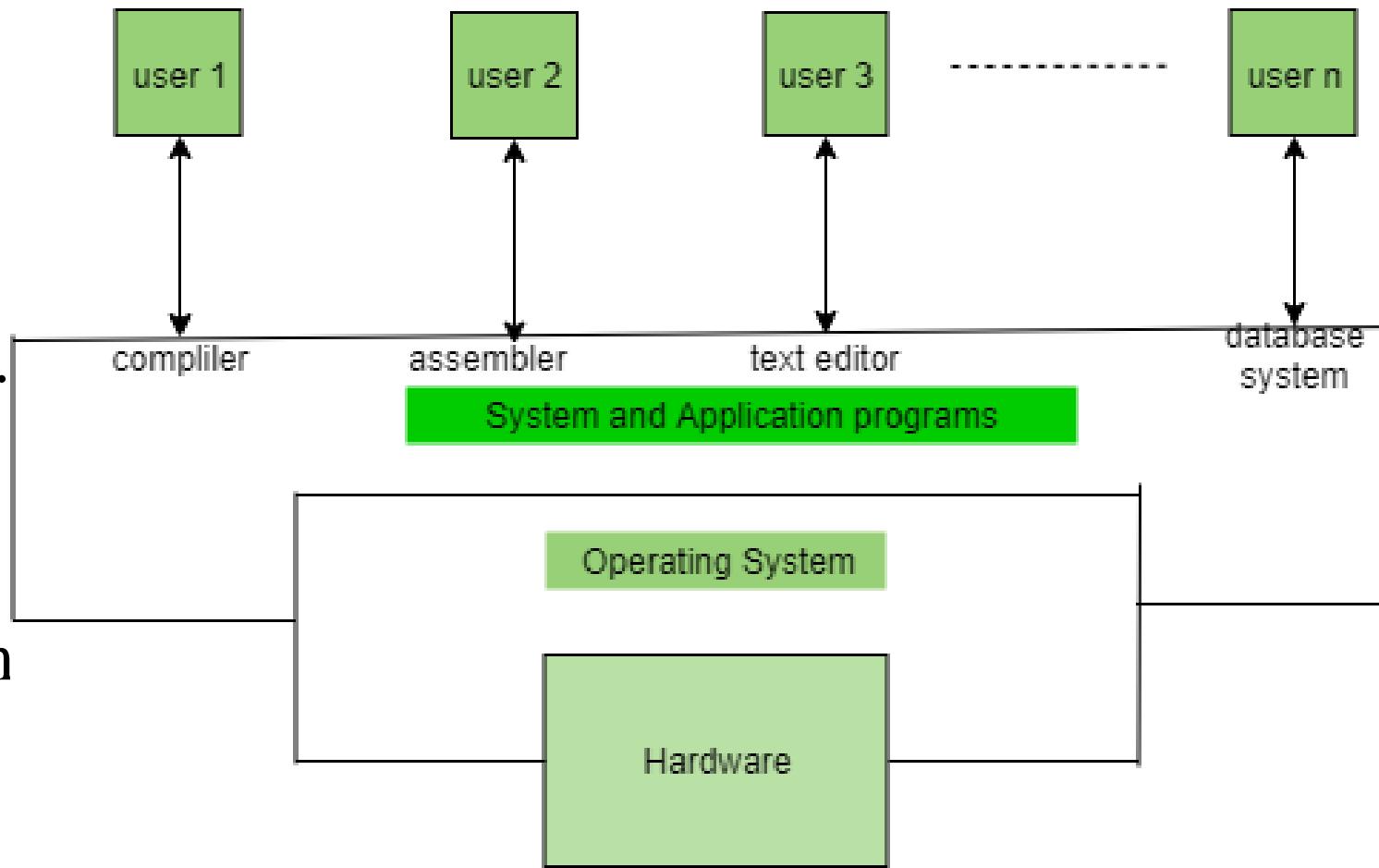
How to select a suitable MCU for a particular application

- No. of data lines (8,16,32,64 - bit)
- Memory Available
- Speed of operation (specified in MHz)
- Energy consumption (specified in uA/MHz) – *energy consumption increases with increase in speed of operation*
- Architecture (RISC or CISC)

- How to build a MCU from scratch

Operating System

- An operating system (OS) is a software that manages or controls the resources of a microcontroller.

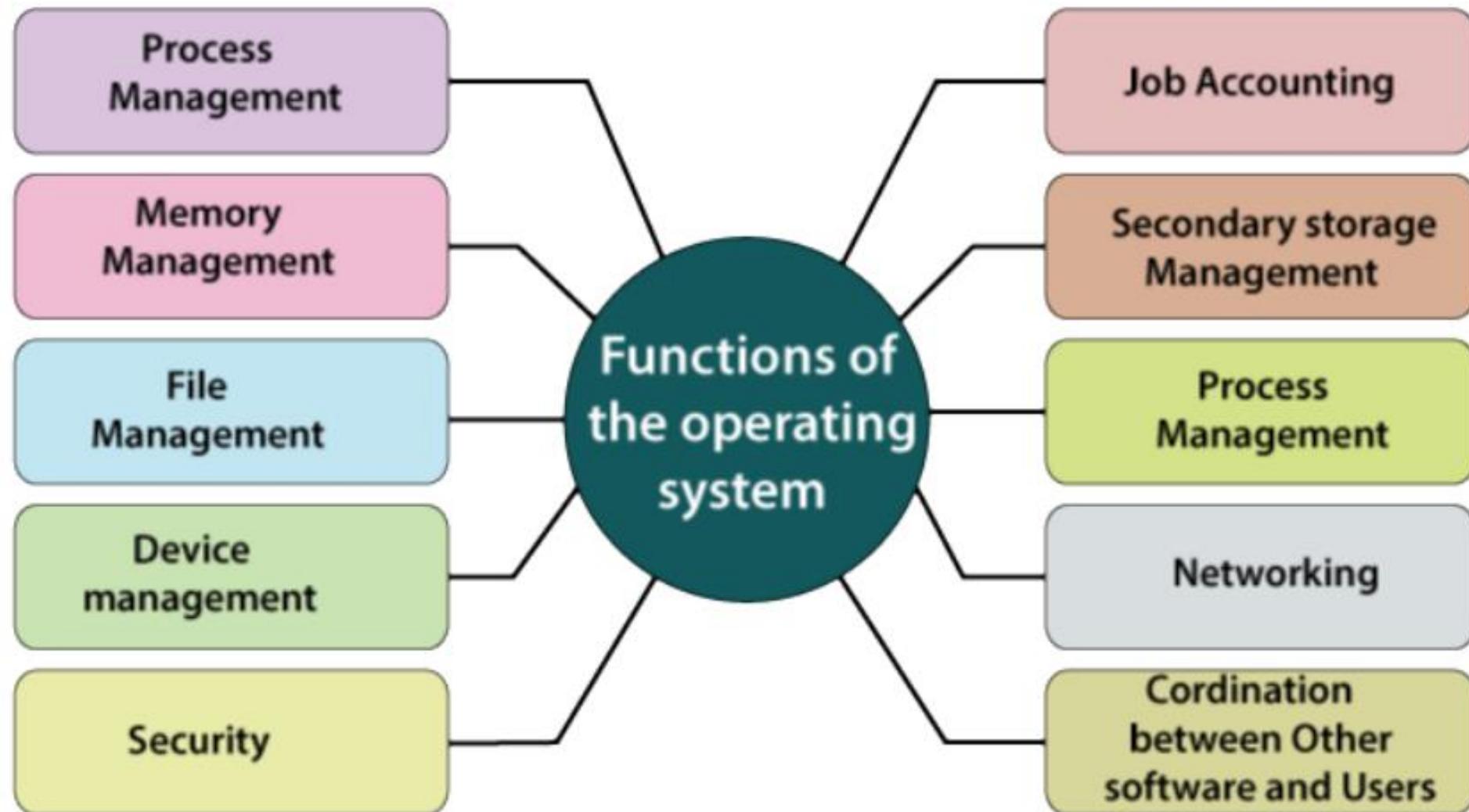


- It acts as an interface between hardware and application program and provides a platform on which the applications may run.

- It performs the scheduling of tasks, manages the files (texts, images) and also provides the drivers for various external devices.

Operating System

- An OS performs the following functions:



GPOS vs RTOS

• GPOS (General Purpose OS)	RTOS (Real-Time OS)
Designed for generalized applications	Designed for dedicated applications
Designed for multi-user environment	Designed for single-user environment
Task scheduling is NOT priority based	Task scheduling is priority-based
Does NOT have task deadline	Has task deadline
Used in PC, Laptop. Ex – Windows, Linux, Android, iOS	Used in embedded systems. Ex – Contiki, FreeRTOS, ARM mbed
Have large memory	Does NOT have large memory

Signal Processing and Computing Software

- Following OS platforms are available for IoT
 1. Raspbian OS
 2. Contiki
 3. Free RTOS
 4. RIOT
 5. TinyOS
 6. OpenWSN and many more....

Signal Processing and Computing Software

- Programming of IoT devices can be done in following languages

1. C/C++
2. Python
3. R
4. Java
5. Erlang
6. Swift
7. PHP
8. GO
9. Rust

IoT Network and Application Layer

- IoT network architecture consists of following network
 - 1. Personal Area Network – devices connected to same network within small geographical area
 - 2. Local Area Network – devices connected to different network within small geographical area
 - 3. Wide Area Network – devices connected to different networks in a large geographical area

PAN Technologies

1. Bluetooth – IEEE 802.15.1
2. ZigBee – IEEE 802.15.3
3. Xbee
4. RFID
5. Z Wave
6. NFC

LAN and WAN Technologies

- Following technologies are available for LAN
 1. WiFi – IEEE 802.11
 2. WiMax

Following technologies are available for WAN

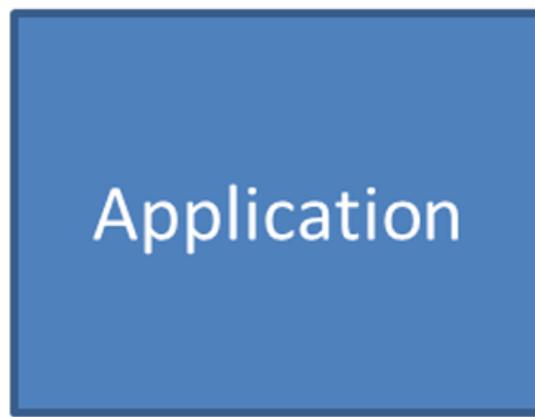
1. 4G/5G
2. LoRaWAN
3. SigFox
4. NB-IoT
5. Neul

	Frequency	Range	Data rate	Power	Encryption	Modulation
Bluetooth	2.4GHz	50–150 m	1Mbps	30 mA	AES	GPSK
ZigBee	2.4GHz	10–100 m	250 kbps	30 mA	AES	BPSK
Z-Wave	900 MHz	30 m	100 kbps	2.5 mA	AES	BFSK
WIFI	2.4GHz and 5GHz	50 m	600 mbps	High	AES	–
Cellular (4G)	2.1 GHz	200 km	3–10 mbps	High	RC4	GFSK
RFID	Many	200 m	4 mbps	Low power	RC4	FSK, PSK
NFC	13.56 MHz	10 cm	100–420 kbps	50 mA	RSA	ASK
Sigfox	900 MHz	30–50 km	1 kbps	100 mW	AES	GFSK
Neul	900 MHz	10 km	100 kbps	100 mW	AES	GFSK
LoRaWAN	Many	2–5 km	50 kbps	100 mW	AES	GFSK
6LoWPAN	2.4GHz	100 m	250 kbps	1–2 years lifetime on batteries	AES	BPSK

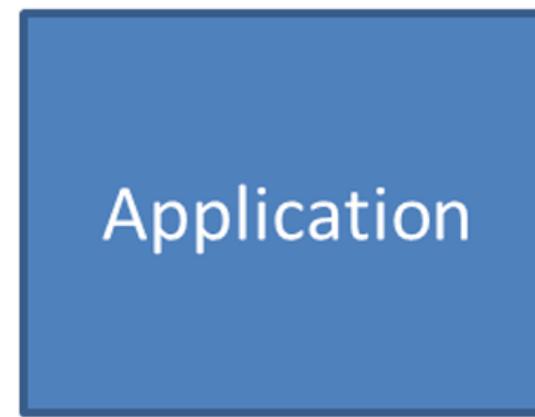
OSI Model



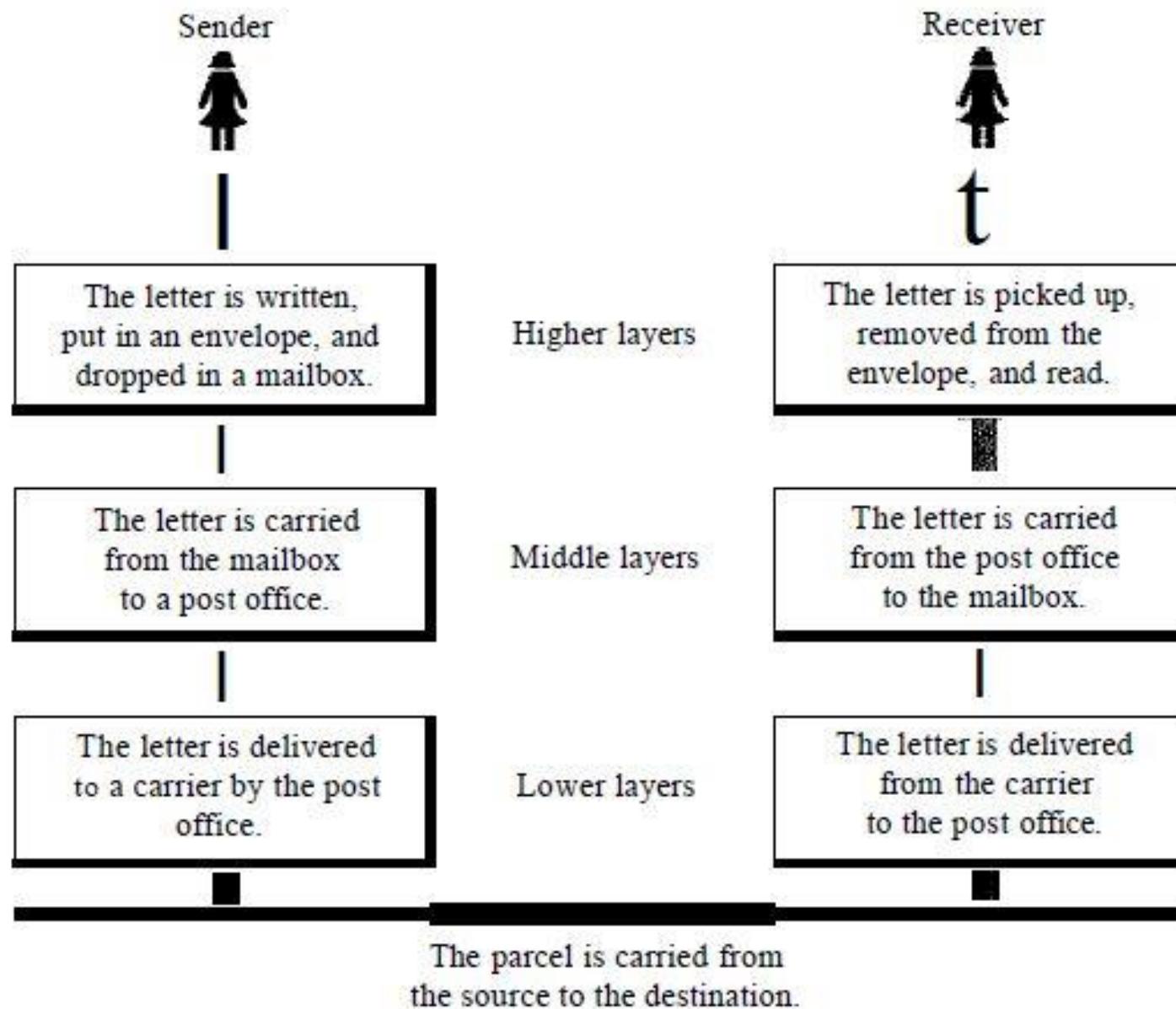
TCP/IP
Original



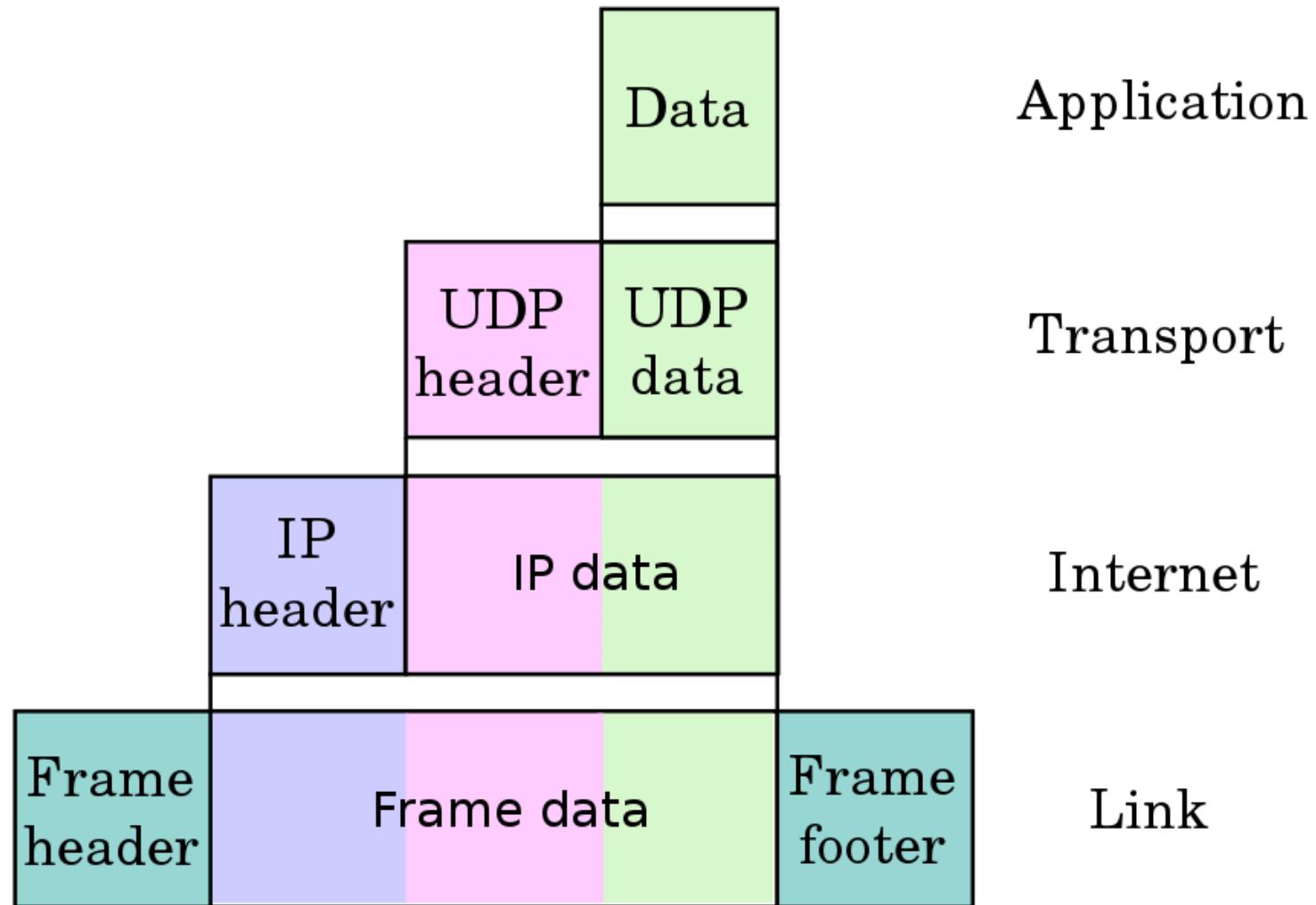
TCP/IP
Updated



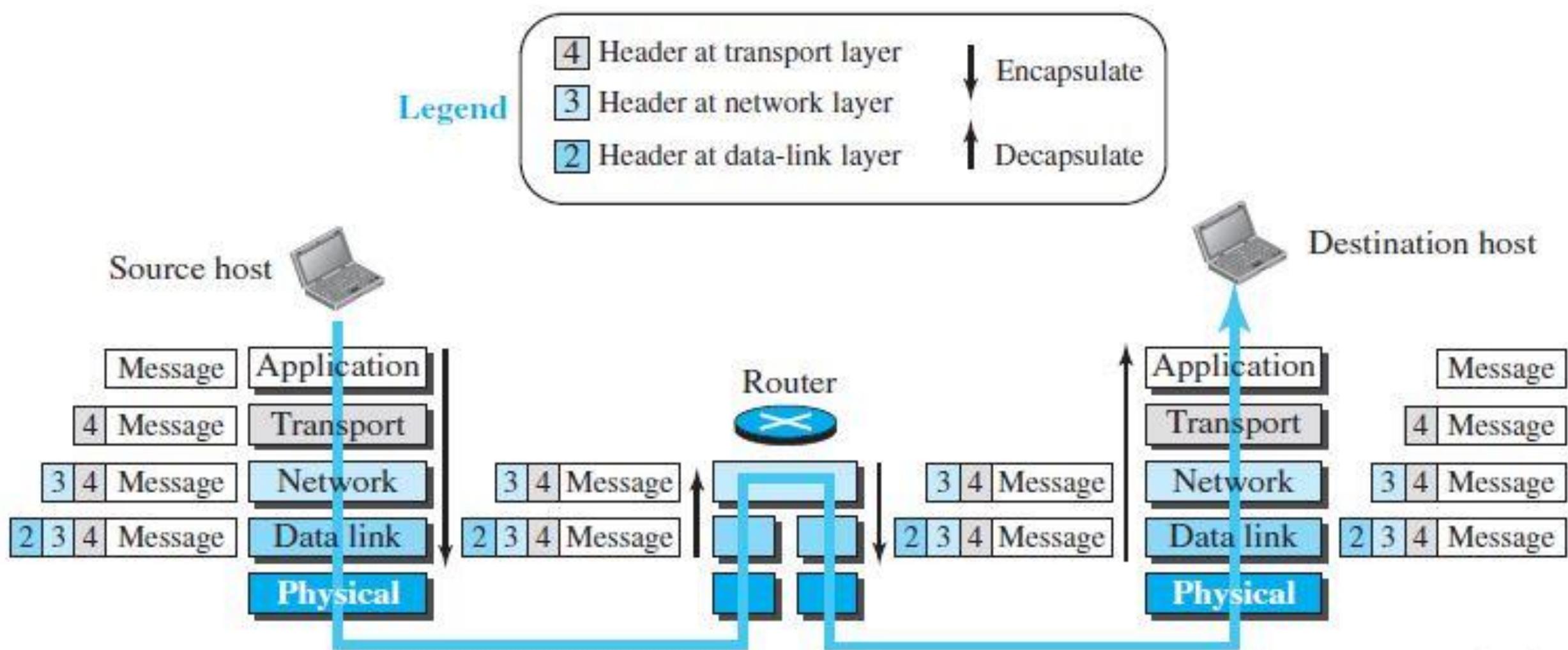
Layered Communication Analogy



Data Flow in TCP/IP Stack



Encapsulation and Decapsulation



Application Layer

- Generates the data and specifies the protocol for client-to-server communication over the connection established by the lower layers.
- Key protocols used are http, ssh, ftp.
- The protocols define how messages are formatted and transmitted and what actions web servers and browsers should take in response to various commands.
- Unit of data is message.

Transport Layer

- Provides end-to-end message transfer services *between 2 or more network* that are *independent of format of user data*
- Attaches *port numbers of source and destination* to the data packet
- Key protocols are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Unit of data is *segment*.
- TCP provides error control, flow control, congestion control .UDP does not provide any of the above.
- Transport layer devices – *Gateways, Firewalls*.

Network Layer

- Manages the data flow between different networks. Unit of data is packet.
- Performs routing of data – selection of best possible route from one network to another
- Attaches IP addresses of sender and receiver in the frame
- IP address to a device is provided by DHCP server only when the device is connected to a network.
- Network Layer Devices – Routers
- IPv4, IPv6 and 6LoWPAN (for IoT devices)

Data Link Layer

- Decides how the communication medium is accessed
- Divided into 2 sublayers – Logical Link Layer and MAC (Medium Access Control) Layer
- Forms data frames – arranges bits into frames. Unit of data is frame.
- Attaches receiver and sender MAC addresses to the frames using Network Interface Cards (NIC) controller.
- Provides medium access control in case of multiplexing
- Data link layer devices – switches, bridges, network interface cards.

Physical Layer

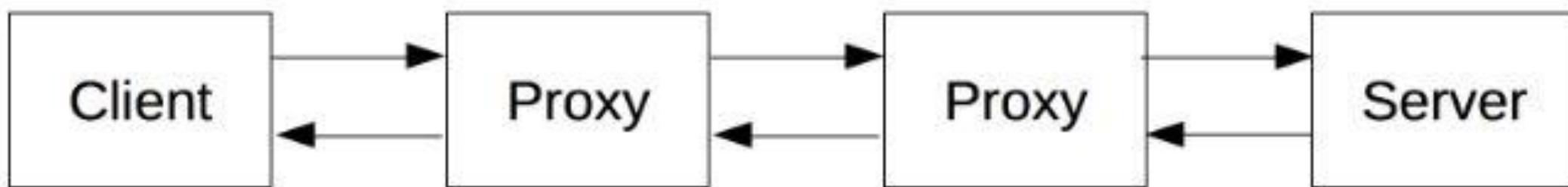
- Converts data (bits) to signals.
- Responsible for actual physical connection between the nodes – wired or wireless.
- Controls bit rate, bit synchronization etc.
- Defines the topology of the network – star, mesh or hybrid.
- Defines the data flow – simplex, half-duplex, full-duplex etc.
- Physical layer devices – cables, modems

	IoT Stack	Web Stack
TCP/IP Model		
<i>Data Format</i>	Binary, JSON, CBOR	HTML, XML, JSON
<i>Application Layer</i>	CoAP, MQTT, XMPP, AMQP	HTTP, DHCP, DNS, TLS/SSL
<i>Transport Layer</i>	UDP, DTLS	TCP, UDP
<i>Internet Layer</i>	IPv6/IP Routing 6LoWPAN	IPv6, IPv4, IPSec
<i>Network/Link Layer</i>	IEEE 802.15.4 MAC IEEE 802.15.4 PHY / Physical Radio	Ethernet (IEEE 802.3), DSL, ISDN, Wireless LAN (IEEE 802.11), Wi-Fi

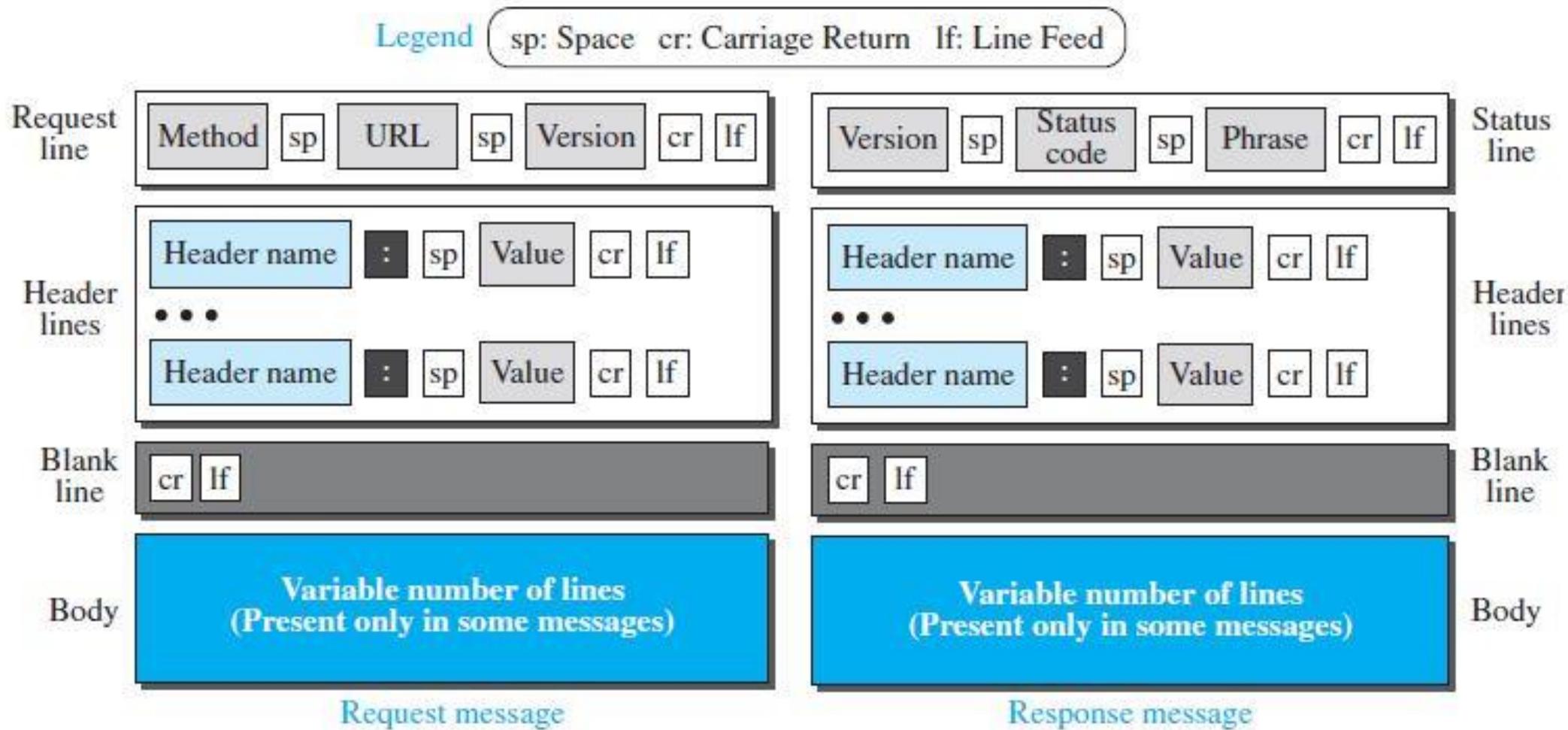
Application	Transport	Network	Data Link	Physical
Generates message and specifies data format	Provides end to end message delivery, independent of format	Data flow between different networks	How communication medium is accessed	Controls bit rate, bit synchronization
PCs, Laptops, Apps	Gateways, Firewalls	Routers	Switches, Network Interface Cards	LAN Wire
	Attaches port number	Attaches IP address	Attaches MAC address	
Message	Segment	Packet	Frame	Signal
	Flow control, error control, congestion control	Finds best suitable path	Collision avoidance	
HTTP, FTP, SSH (CoAP)	TCP, UDP	IPv6, 6LoWPAN		

HTTP (Hypertext Transfer Protocol)

- Defines how the client-server programs can be written to retrieve info from webpages.
- Connection-oriented protocol based on request-response.
- Between client and server there are other computers, modems, routers etc. which act as proxies.



HTTP Message Format



HTTP Request

- Requests are the messages send by the client to the server.
- A request message consists of 3 sub-parts
 - Request Line
 - Header
 - Main body
- Request line contains method, identifier of the resource and protocol version.
- Header contains the additional information about the client – like authorization, permissions etc.
- Main body contains the actual data (whether requested info is an image, text etc.)

HTTP Request Line

- It contains method, identifier of the resource and protocol version.
- Method can be anyone of the following
 1. GET – used to retrieve the information from a given server
 2. POST – used to send data to the server (like file upload, customer info etc.)
 3. PUT – used to replace all current representations of the target resource with the uploaded content
 4. DELETE – remove all the current representations of the target resource
 5. CONNECT – establishes a tunnel to the server identified by the resource
 6. TRACE – performs a message loop-back test along the path to target resource
 7. PATCH – applies partial modifications to a resource

HTTP Header Types

<i>Header</i>	<i>Description</i>
User-agent	Identifies the client program
Accept	Shows the media format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
Host	Shows the host and port number of the client
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Cookie	Returns the cookie to the server (explained later)
If-Modified-Since	If the file is modified since a specific date

HTTP Response

- Message sent from server to the client. It is used to provide the client with the resource it has requested or convey an error in the process.
- An HTTP response consists of the following fields:
 - Status line
 - Header
 - Message body
- Status line contains 3 fields
 - HTTP Version number
 - Status code
 - Reason phrase

HTTP Response

- Status code – 3 digit number that indicates the result of the request
 - 1xx – shows the request was received and continuing the process
 - 2xx – Success – request was received, understood and accepted
 - 3xx – Redirection – further action must be taken to complete the request
 - 4xx – Client Error – request contains incorrect syntax
 - 5xx – Server Error – server failed to fulfill a valid request
- Reason phrase – text summarizing the meaning of status code (e.g. OK)

Ex: HTTP/1.1 200 OK – version is 1.1, status code is 200 and phrase is OK

HTTP Request example

- Suppose a client request academic calendar.html from a server running on academics.iitj.ac.in, the request and response messages will look like this

GET /academiccalendar.html HTTP/1.1

HTTP/1.1 200 OK

User-Agent: Chrome/13.0

Date: Friday, August 18, 2023 4:00 PM

Host: www.academics.iitj.ac.in

Server: Apache/2.2.14

Accept-language: en-us

Last Modified: Wed, August 16

Accept-encoding: gzip, deflate

Content-Length: 88

Connection: Keep Alive

Content-Type: text/html

HTTP Request example

- Suppose a client request something that does not exist on the server, it will return a response with error code ‘404’.

Server response

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>

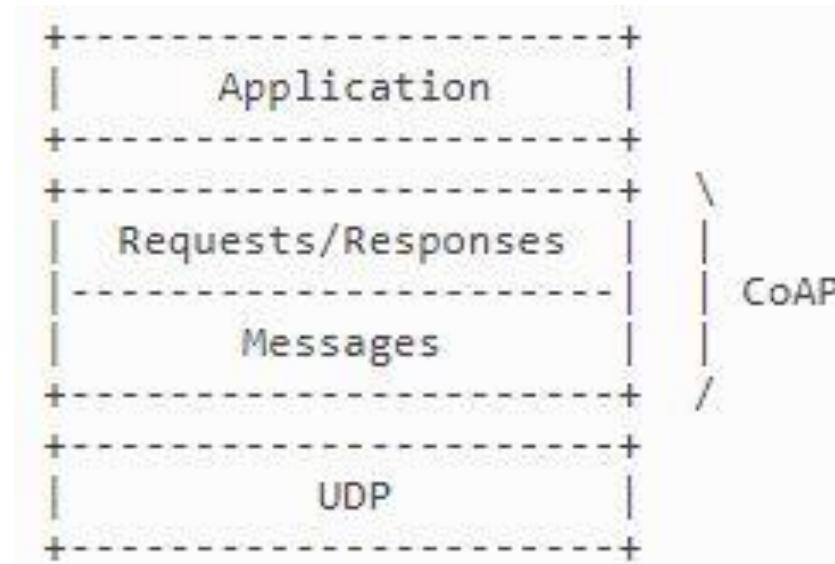
<head>
  <title>404 Not Found</title>
</head>

<body>
  <h1>Not Found</h1>
  <p>The requested URL /t.html was not found on this server.</p>
</body>

</html>
```

CoAP (Constrained Application Protocol)

- Protocol similar to HTTP build for resource constrained devices.
- Works on request/response model but handles the interchange of data asynchronously.
- This asynchronous handling is made possible by using another layer of messages that support optional reliability



(From COAP RFC - <https://www.rfc-editor.org/rfc/rfc7252>)

CoAP (Constrained Application Protocol)

- Confirmable Message (CON)
 - message which requires acknowledgement and must be retransmitted in case packets are lost.
 - Extremely important requests which should not be dropped are sent by encapsulating in a confirmable message
- Acknowledgement Message (ACK)
 - Reply to a confirmable message
- Non-confirmable message (NON)
 - Messages which do not require an acknowledgement
 - Ex – repeated readings from a sensor

CoAP (Constrained Application Protocol)

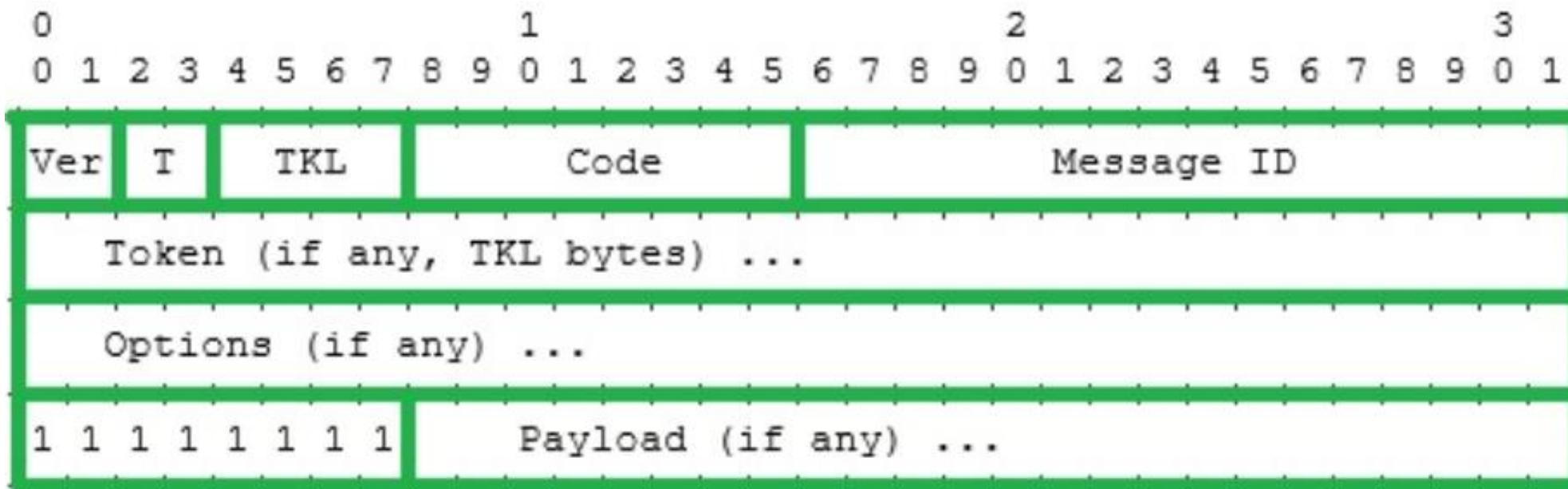
- Reset Message (RST)
 - Indicates that a specific message was received but was not processed properly by the receiver or some context is missing from the original message
 - Can also be used to check the liveness of the receiver i.e. Ping request
- Piggybacked response
 - The response to a particular request is included right in the acknowledgement message
 - No separate message needs to be sent for the response of a request.
- Separate response
 - Confirmable message is first acknowledged with an empty message and actual response is sent later.
 - Used when the client does not have the response at the time when request is made

CoAP (Constrained Application Protocol)

- Token (Request ID)
 - Each request has a unique ID called token which is used to correlate the request and response in case of “separate response”.
 - In separate response when a client responds to a particular request at a later time, it uses its token to identify the request and sends back the response with the token.

CoAP Message Format

- 4 byte header followed by 0-8 bytes of Token
- Token value is used to correlate requests and responses of a particular client.



CoAP Message Format

- Ver – 2 bit unsigned integer – indicates the CoAP version number
- T – 2 bit unsigned integer – Type of message – Confirmable (0), Non-confirmable (1), Acknowledgement (2) and Reset (3).
- TKL – 4 bit unsigned integer – Token Length
- Code – 8 bit unsigned integer – indicates response code in ‘c.dd’ format
- Message ID – 16 bit unsigned integer

CoAP Message Format

- The 8 bit code field is written in the format: “class.code”.
- 3 MSB denotes the class while remaining bits denote the code as shown

0	1	2	3	4	5	6	7
Class	Code						

- If class field is ‘0’, it implies the ‘method’ (same as in HTTP)
- If class field is ‘2’, it implies ‘success’ (same as in HTTP)
- If class field is ‘4’, it implies ‘client error’ (same as in HTTP)
- If class field is ‘5’, it implies ‘server error’ (same as in HTTP)

CoAP Message Format

- Thus 0.01 code will imply GET command, 2.01 will imply successfully created....

• Method: 0.XX	• Success: 2.XX	• Client Error: 4.XX	• Server error: 5.XX
0. EMPTY	1. Created	0. Bad Request	0. Internal server error
1. GET	2. Deleted	1. Unauthorized	1. Not implemented
2. POST	3. Valid	2. Bad Option	2. Bad gateway
3. PUT	4. Changed	3. Forbidden	3. Service unavailable
4. DELETE	5. Content	4. Not Found	4. Gateway timeout
5. FETCH	31. Continue	5. Method Not Allowed	5. Proxying not supported
6. PATCH		6. Not Acceptable	
7. iPATCH		8. Request Entity Incomplete	
		9. Conflict	
		12. Precondition Failed	
		13. Request Entity Too Large	
		15. Unsupported Content-Format	

CoAP Messaging Model

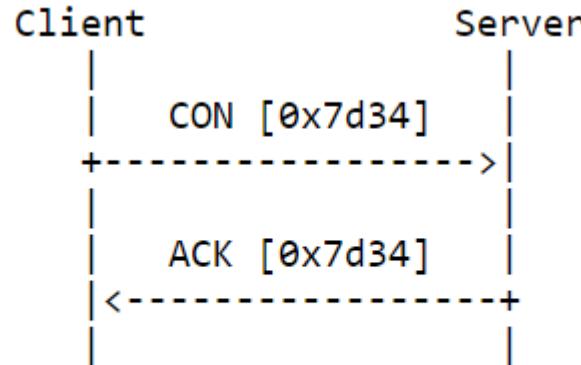


Figure 2: Reliable Message Transmission

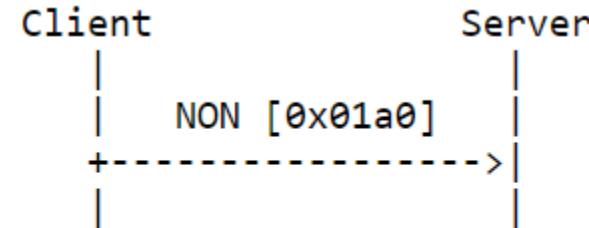


Figure 3: Unreliable Message Transmission

- CON stands for confirmable message
- Each message has its own unique message ID (0x7d34) which is returned back as it is by the server in the ACK message.

CoAP Messaging Model

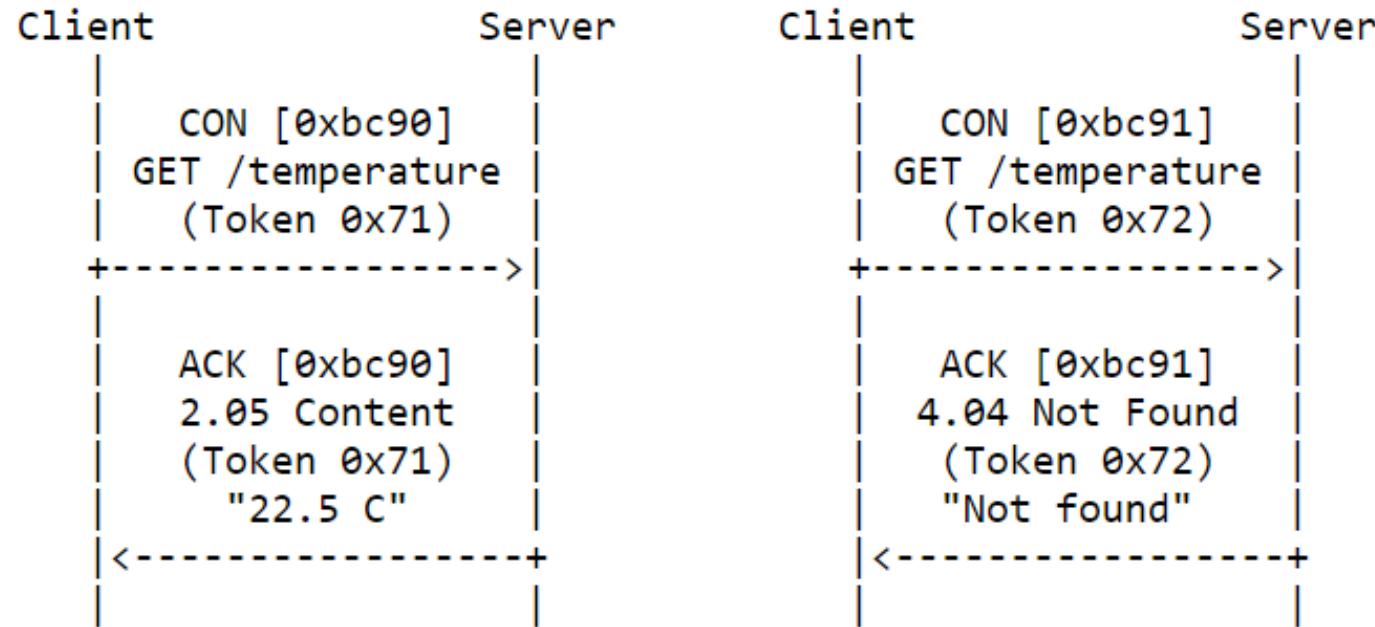


Figure 4: Two GET Requests with Piggybacked Responses

- Client requests temperature value from the server using GET command
- Server replies with the value 22.5 C (if it has the data) or with a message ‘Not found’ (if it does not have the data)
- 2.05 is the response code

CoAP Messaging Model

- A

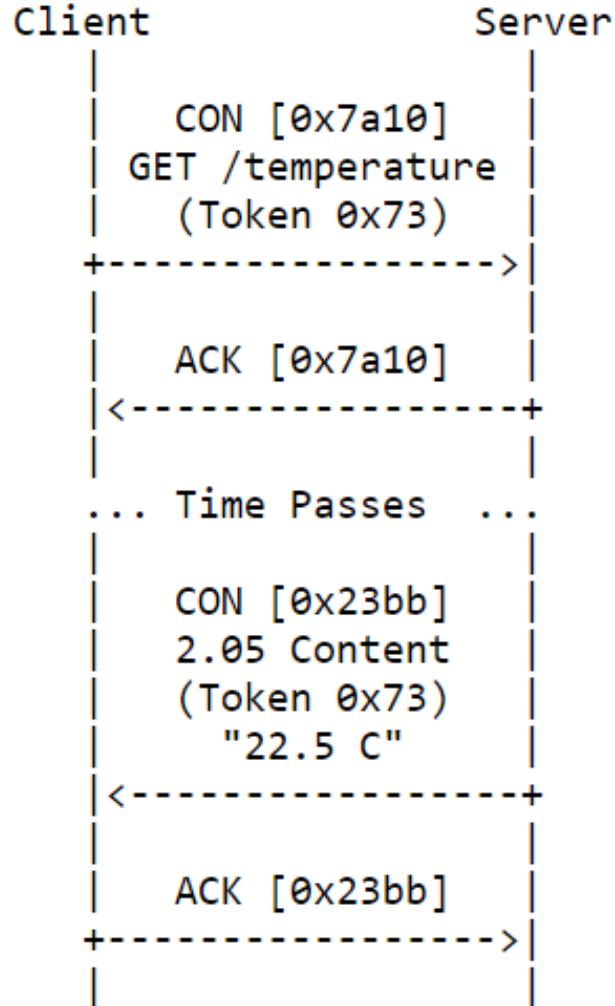


Figure 5: A GET Request with a Separate Response

CoAP Messaging Model

- A

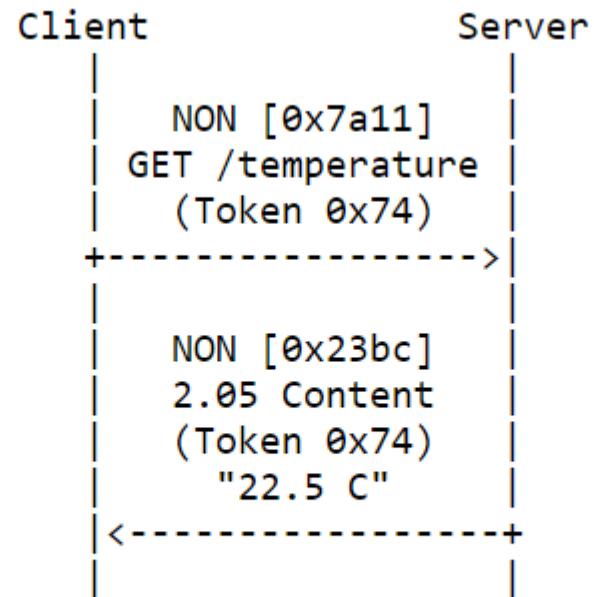


Figure 6: A Request and a Response Carried in Non-confirmable Messages

CoAP

- Both CoAP and HTTP use REST (Representational State Transfer) architecture and are compatible with RESTful API.
- Thus, devices operating on CoAP can easily communicate with devices operating on HTTP and vice versa.

HTTP vs CoAP

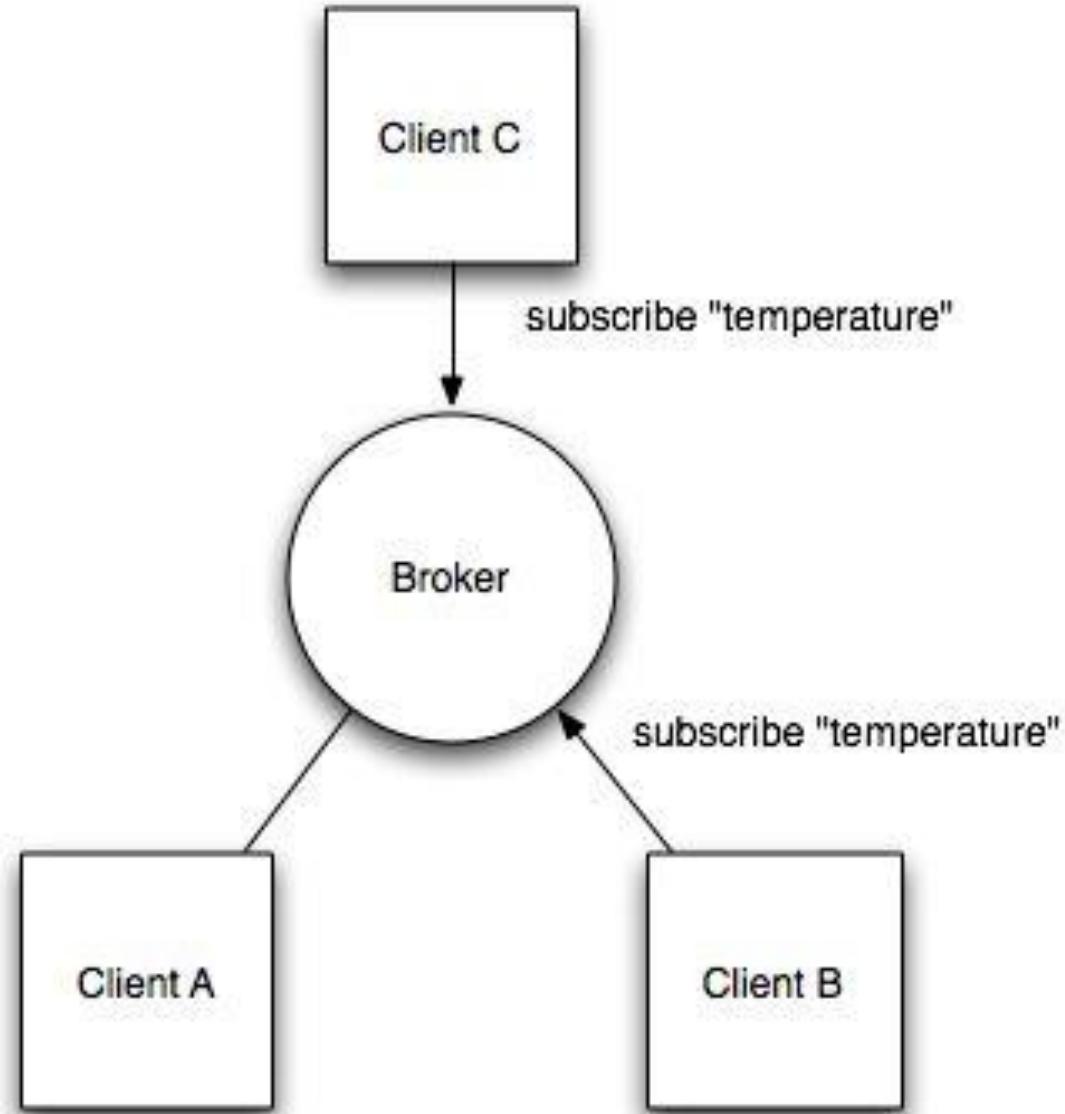
HTTP (HyperText Transfer Protocol)	CoAP (Constrained Application Protocol)
Works on TCP and IP	Works on UDP and 6LoWPAN
Doesn't support broadcasting	Supports broadcast and multicast
Complex with more overhead	Simple and less overhead
Designed for high resource devices	Designed for resource constrained devices
Header size of around 1 kB	Header size of 4 bytes

MQTT (Message Queue Telemetry Transport)

- Application layer protocol based on *publish/subscribe model* working on TCP
- Publishers are sensors which generate data
- Subscribers are applications interested in the data generated by publishers
- Brokers connect publishers to subscribers
- Sensor data is grouped into topics and subscribers subscribe to a topic.
- Subscribers receive all the data published in a particular topic

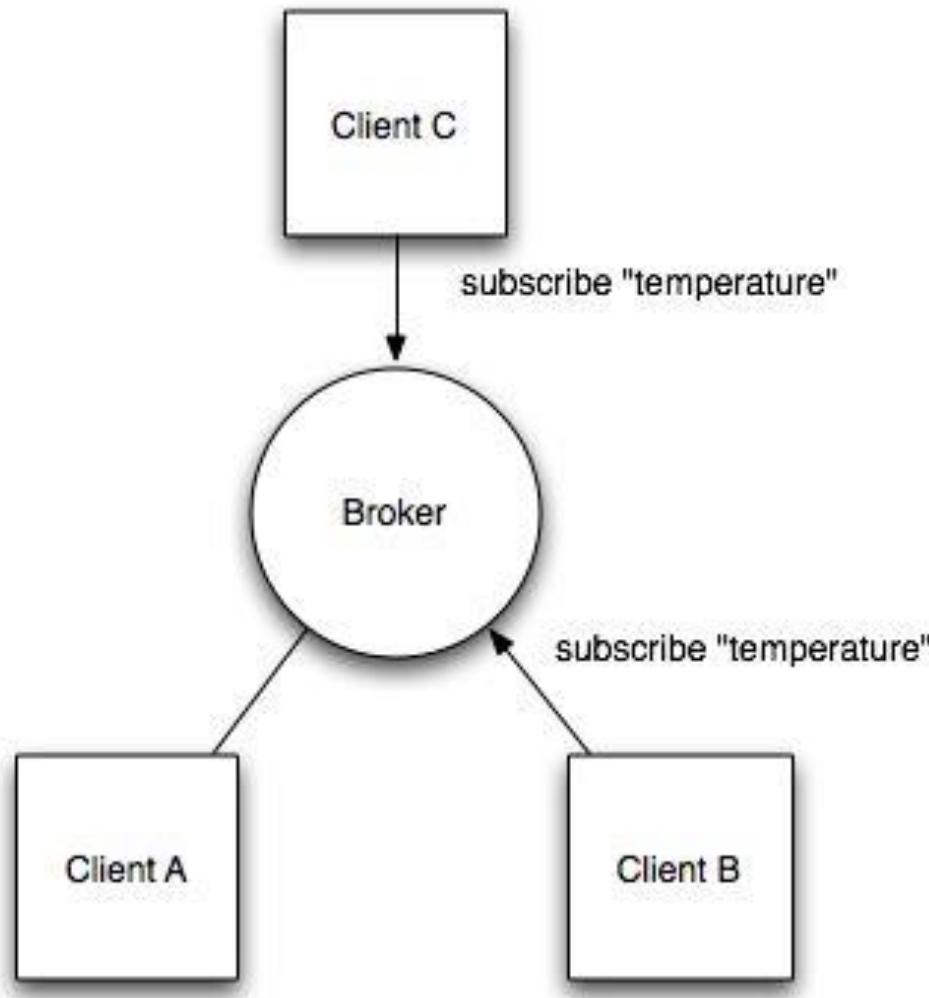
MQTT Example

- 3 clients connected through a broker and subscribed to a topic ‘temperature’
- Every client can be a publisher as well as subscriber i.e. everyone can generate data and consume data
- A connection is always present between broker and client since MQTT works on TCP
- Data transmission is event-driven and not Request – Response type as in CoAP



MQTT Example

- Here client A publishes data on the topic called “temperature” while client B and C subscribe to it.
- Client A can publish data on other topics like pressure, humidity etc.
- If client B subscribes to topic pressure, then it will get data of pressure only and not other topics.
- Examples of open source broker apps – HiveMQ, Mosquitto.



MQTT Salient Features

- In MQTT, there is no direct connection between a publisher and a subscriber i.e. publisher doesn't know who subscriber is and vice versa.
- The broker stores or purge messages based on whether there is a subscriber for a particular topic or not.
- Thus, MQTT is *highly scalable protocol as there is virtually no connection between the end devices* and the broker need to store messages indefinitely.
- As messages can be divided into topics and sub-topics, *filtering of data is possible in MQTT*. This helps in reducing the memory requirement of the subscriber as it gets only the data it has asked for.

MQTT Quality of Service

- The publisher need not announce the topic it is going to publish beforehand. For ex., a sensor node in MQTT can start publishing data on any topic and the broker has to accept it irrespective of whether there are any subscribers for it.
- In case there are no subscribers for a particular topic, the broker can store the data infinitely or can purge the data. This can be decided by the user based on the application i.e. broker can be configured as required.
-

MQTT Messages

- 3 main type of messages in MQTT are
 1. CONNECT – used by clients to send connection request to broker
 2. PUBLISH – used by clients to send messages to broker
 3. SUBSCRIBE – used by clients to receive messages from the broker
- Other type of messages used are
 1. CONNACK – acknowledgement sent by the broker in response to CONNECT
 2. PUBACK, SUBACK
 3. UNSUBSCRIBE
 4. PUBREC
 5. PUBREL

MQTT Messages

- The CONNECT message contains the following fields:
 1. Client ID – addresses of the client
 2. Username and Password – for authentication
 3. Clean Session – used to inform broker whether to retain the message or purge it if there are no subscribers. If this field is set as TRUE messages are purged and if it is set as FALSE, messages are retained.

MQTT Messages

- The CONNECT message contains the following fields:
 1. Last Will Topic – in case of an unexpected shutdown of the client, the topic in the last will topic field remains as the last topic published by the client
 2. Last Will Message – last message of a client in case of unexpected shutdown
 3. Last Will Retain – used to inform the broker whether to retain the last will message or not
 4. Keep Alive – used to check whether the session is alive or not

MQTT Messages

- The CONNECT message contains the following fields:

MQTT-Packet:

CONNECT

contains:

clientId

cleanSession

username (optional)

password (optional)

lastWillTopic (optional)

lastWillQos (optional)

lastWillMessage (optional)

lastWillRetain (optional)

keepAlive



Example

"client-1"

true

"hans"

"letmein"

"/hans/will"

2

"unexpected exit"

false

60

MQTT-Packet:

CONNACK

contains:

sessionPresent

returnCode



Example

true

0

MQTT Messages

- The PUBLISH message contains the following fields:

MQTT-Packet:	PUBLISH	
contains:		Example
packetId	(always 0 for qos 0)	4314
topicName		"topic/1"
qos		1
retainFlag		false
payload		"temperature:32.5"
dupFlag		false

MQTT Messages

- Retain flag is set ‘TRUE’ when the client wants the broker to retain the value of a particular packet.
- This is helpful in the case when the client knows it is about to go offline.
- The retained value is treated as the “last known good value” from the client.
- Future clients get the retained value when they subscribe to the topic after the client has gone offline.

MQTT Messages

- The SUBSCRIBE message contains the following fields:

MQTT-Packet:	
SUBSCRIBE	
contains:	Example
packetId	4312
qos1 }	1
topic1	"topic/1"
qos2 }	0
topic2	"topic/2"
...	...

MQTT Quality of Service

- MQTT protocol offers 3 QoS options that cater to different application scenarios.
The 3 QoS options are:

0. At most once (QoS 0)

- The sender sends the message only once and no longer cares whether it is received or not (fire and forget).
- There is no resending mechanism
- Requires lowest amount of effort and network
- No guarantee of successful message transmission
- can be used to send ambient sensor data where it does not matter if an individual reading is lost and next one would be sent soon after

MQTT Quality of Service

More details from (<https://www.emqx.com/en/blog/introduction-to-mqtt-qos>)

MQTT Quality of Service

1. At Least Once (QoS 1)

- The sender sends the message once and wait for sometime for acknowledgment from the broker and then retransmits the message.
- Guaranteed reception of message but multiple copies of message may be received by the receiver

2. Exactly Once (QoS 2)

- Guarantee that each message is received only once.
- Safest and slowest service.
- Useful in critical scenarios like billing systems where duplicate and lost messages could lead to incorrect charges being applied.

MQTT Quality of Service

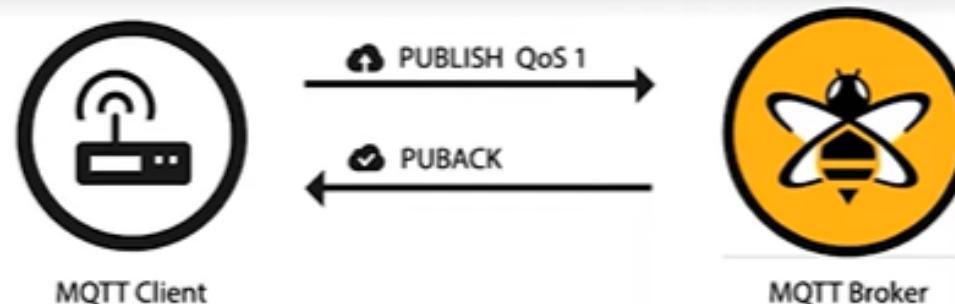
- In QoS 1,
 1. Sender sends a packet with unique packet ID using PUBLISH command and stores it locally until acknowledgement is received
 2. Receiver send PUBACK packet indicating that packet is received
 3. If no PUBACK is received, sender retransmits the previous packet
- Packet ID is used to match the PUBLISH packet with the corresponding PUBACK packet

MQTT Quality of Service

- In QoS 2
 1. Sender sends a packet with a unique packet ID using PUBLISH command and locally stores a copy of it
 2. Receiver sends a PUBREC message indicating packet is received
 3. Sender then sends a PUBREL packet indicating that the above used packet ID will be released and it deletes its locally stored copy.
 4. Finally, receiver sends a PUBCOMP packet to signal the completion of message transfer

MQTT Quality of Service 1

Quality of Services 1



MQTT-Packet:

PUBLISH



contains:

packetId (always 0 for qos 0)
topicName
qos
retainFlag
payload
dupFlag

Example

4314
"topic/1"
1
false
"temperature:32.5"
false

MQTT-Packet:

PUBACK



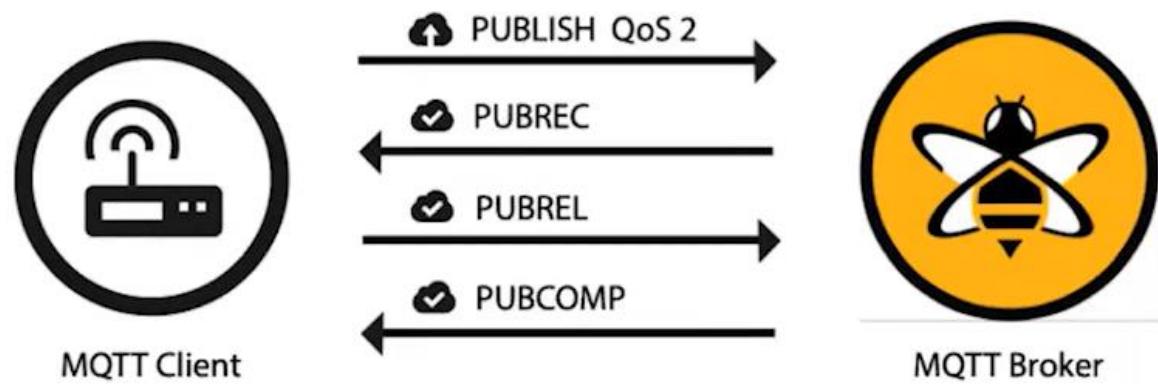
contains:

packetId

Example

4314

MQTT Quality of Service 2



MQTT vs CoAP

MQTT	CoAP
Works on TCP	Works on UDP
Connection oriented protocol	Connectionless protocol
Publish – Subscribe Model	Request – Response Model
Best suitable for one-to-many and many-to-many transmissions	Best suitable for one-to-one transmission
Only subscribed data is available to the client	All the requested data is available to the client
Header size – 2 Bytes	Header size – 4 Bytes

MQTT Persistent Message

- <https://www.emqx.com/en/blog/mqtt-session>
- <https://www.ibm.com/docs/en/ibm-mq/7.5?topic=concepts-message-persistence>

MQTT Example

- Mosquitto / HiveMQ – open source MQTT broker software for Ubuntu, can also run on Rpi or other MCU which have TCP/IP stack.

(https://www.youtube.com/watch?v=AsDHEDbyLfg&ab_channel=BRUHAutomation – Video on how to install Mosquitto on Rpi)

- Paho – open source MQTT client software for Ubuntu

Criteria	HTTP	CoAP	MQTT
Architecture	Client/Server	Client/Server or Client/Broker	Client/Broker
Abstraction	Request/Response	Request/Response or Publish/Subscribe	Publish/Subscribe
Header Size	Undefined	4 Byte	2 Byte
Message size	Large and Undefined (depends on the web server or the programming technology)	Small and Undefined (normally small to fit in single IP datagram)	Small and Undefined 256 MB maximum size
Operations/Methods	Get, Post, Head, Put, Patch, Options, Connect, Delete	Get, Post, Put, Delete	Connect, Disconnect, Publish, Subscribe, Unsubscribe
Quality of Service (QoS) Reliability	Limited (via Transport Protocol - TCP)	Confirmable Message or Non-confirmable Message	QoS 0 - At most once delivery QoS 1 - At least once delivery QoS 2 - Exactly once delivery
Transport Protocol	TCP	UDP, TCP	TCP (MQTT-SN can use UDP)
Security	TLS/SSL	DTLS/IPSEC	TLS/SSL
Default Port	80/443 (TLS/SSL)	5683 (UDP)/5684 (DTLS)	1883/8883 (TLS/SSL)

TCP (Transmission Control Protocol)

- TCP is a transport layer protocol – it handles *process to process communication*.
- A network layer protocol like *IPv6 handles machine to machine (host to host) communication*.
- There may be several processes running on a machine which may need to send data to processes running on another machine. This data transfer between processes running on different machines (hosts) is handled by transport layer protocols.

TCP (Transmission Control Protocol)

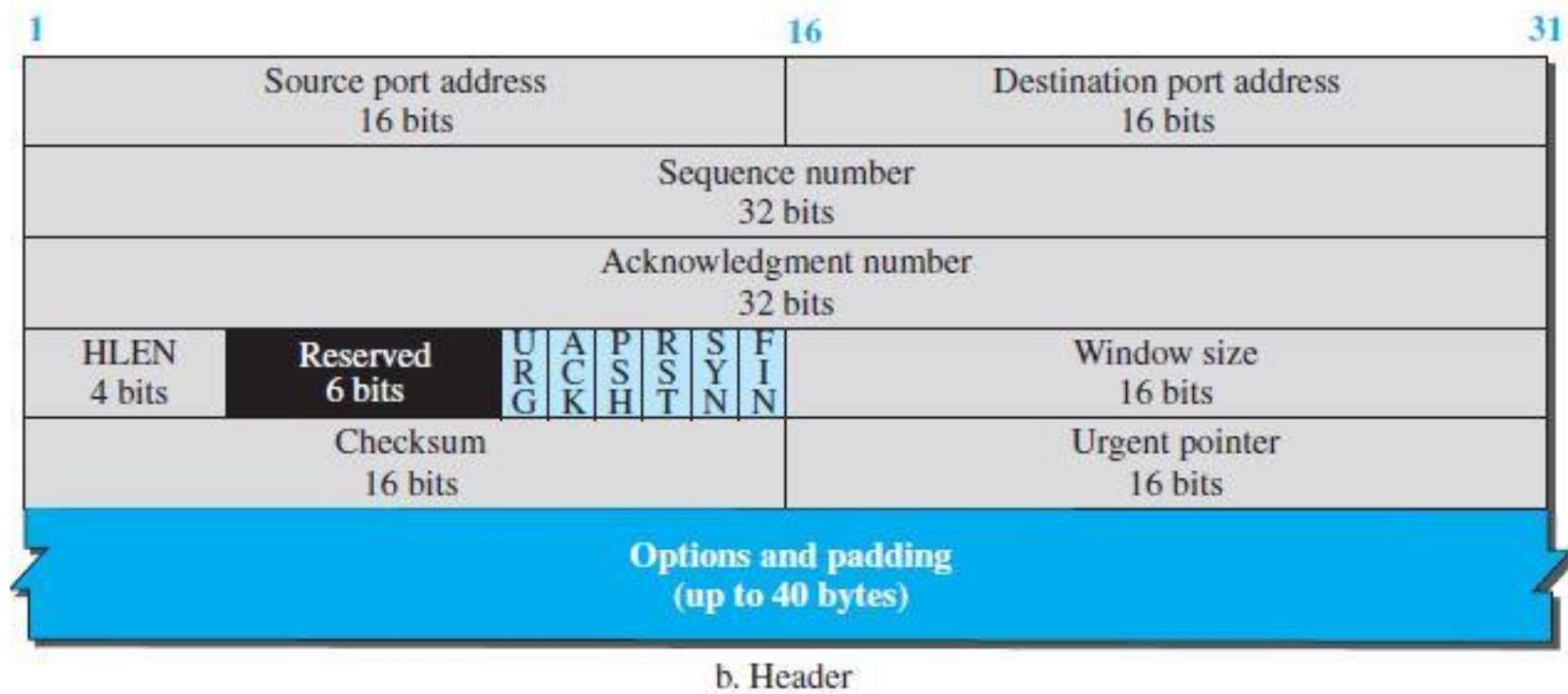
- Connection – oriented and reliable transport layer protocol which involves 3 phases
 - Connection establishment
 - Data transfer
 - Connection termination
- Application layer provides a stream of messages which is then broken down into different segments, each having a time-stamp for correlation.
- The segments can be of varying size.
- Utilizes buffers to match the data rate at the source and destination.

TCP (Transmission Control Protocol)

- *Reliable protocol* – the receiver sends ACK for the received segments
- If the ACK is not received, the segments are *retransmitted*. This may also lead to multiple duplicate segments being sent.

TCP Segment Format

- Header – 20 to 60 bytes with no limit on segment size
- Different fields in header are used for various controls/services provided by TCP



TCP Header Fields

- Source / Destination Port Address – 16 bit addresses of client and server
- Sequence Number – unique number for each segment used for flow control and reassembling of segments at the receiver
- Acknowledgment Number – unique number given to every acknowledgment segment
- HLEN – length of the header
- Reserved bits – reserved for future use

TCP Header Fields

- Control Fields – 6 flags used for different purpose
- URG – Urgent pointer is valid (1) or not (0)
- ACK – Acknowledgment is valid (1) or not (0)
- PSH – Request for push
- RST – Reset the connection
- SYN – Synchronize sequence number
- FIN – Terminate the connection

TCP Header Fields

- When URG bit is 1, it indicates to the receiver that
 - Some data in the current segment is urgent
 - The urgent data is identified using ‘*urgent pointer*’ present in the header
 - Urgent data has to be prioritized and sent to the application using a separate channel
- When PSH bit is 1, it means
 - All the segments in the buffer are immediately pushed to the receiving application
 - The buffer must be freed up immediately
 - Generally PSH bit is set to ‘0’ because it disrupts the normal working of the receiver
- When SYN bit is 1, it means
 - The current sequence number is the initial sequence number
 - It is the start of synchronization between the sender and the receiver

TCP Header Fields

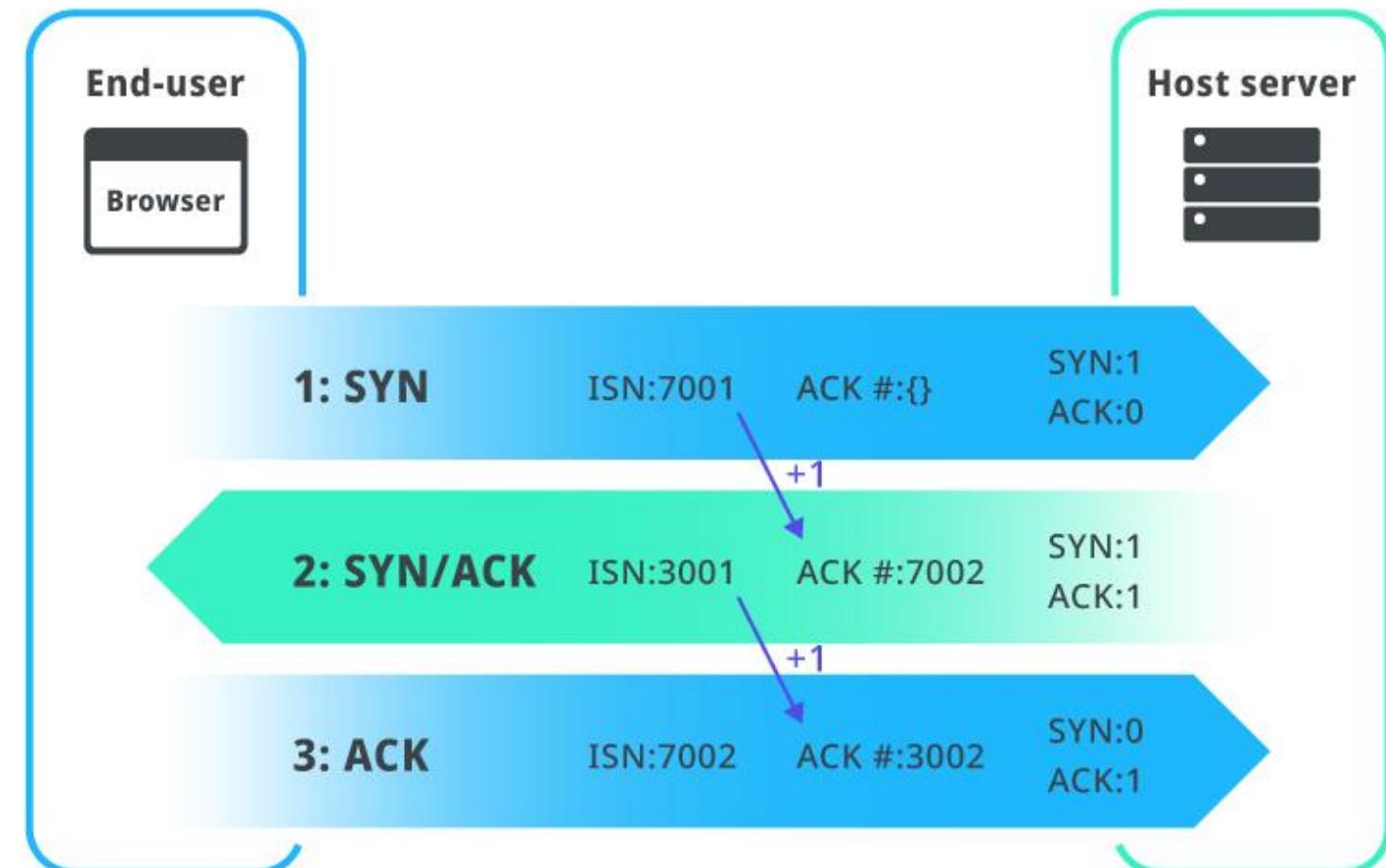
- Window Size – Size of the window that must be maintained as decided by the receiver. This is a necessity of sliding window protocol.
- Checksum – used for error detection
- Urgent pointer – defines the number to be added to the sequence number to determine the last urgent byte in the data section of the segment

TCP 3 way Handshake

- Before the communication, the connection is established in TCP using a 3 way handshaking process as shown

1. Browser (client) sends a connection establishment request using SYN and a sequence number e.g. 7001

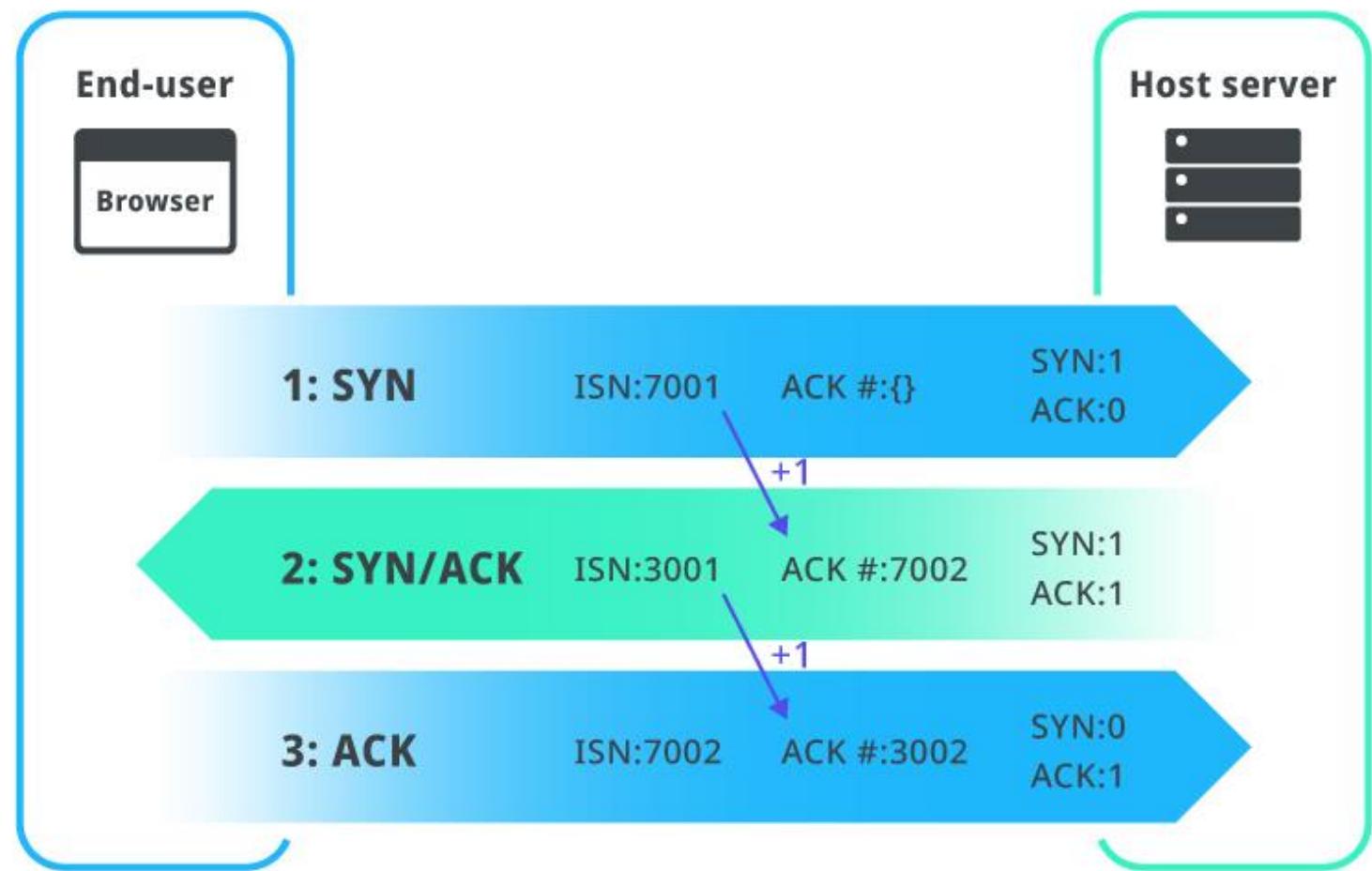
2. Server acknowledges this request by sending an ACK corresponding to the sequence number i.e. 7002 and initializes the connection using its own sequence number e.g. 3001.



TCP

3. The client acknowledges the connection initiation by sending an ACK corresponding to the sequence number of the previous segment i.e. 3002

- The ACK field in segment 2 specifies the expected sequence number of the next segment i.e. in this example ACK field of segment 2 is 7002, which means server is expecting that the next segment should have sequence number 7002 which is fulfilled by the client in segment 3 (ISN: 7002)



Advantages of TCP

- Flow control – balances the rate at which data is created and consumed
- Error control – provides mechanism for
 - Detecting and resending corrupted segments
 - Resending lost segments
 - Storing out-of-order segments until missing segments arrive
 - Detecting and discarding duplicated segments
- Congestion control – detecting congestion in network and real-time decision on how many segments to be transmitted.

Sequence of commands in TCP

- The following sequence of commands are executed from the client side in every TCP request
 1. `sock_init ()` – create a socket
 2. `Connect ()` – initiate a connection
 3. `read/write` – read/write data
 4. `Close ()` – close a connection

Sequence of commands in TCP

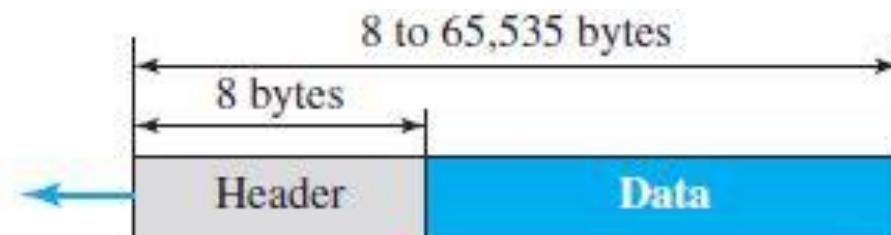
- The following sequence of commands are executed from the server side in every TCP request
 1. `sock_init ()` – create a socket
 2. `bind ()` – register port with the system
 3. `Listen ()` – establish client connection
 4. `Accept ()` – accept client connection
 5. `read/write` – read/write data
 6. `Close ()` – close connection

UDP (User Datagram Protocol)

- Simple transport layer protocol which *does not provide error control, flow control and congestion control.*
- Connectionless protocol – each datagram is an independent entity.
- UDP packets are called User Datagram and can have max size of 65 kB.
- Very simple protocol with minimum overhead.
- Used for sending small packets in *unreliable* manner – means UDP does not do anything extra to provide reliability. It relies on the reliability of IP.

UDP Packet (Datagram)

- 64 kB packet with 8 byte header
- Header contains source and destination port number, total length and checksum.
- Since UDP is connectionless, the process cannot send stream of data to UDP and expect it to chop into different related datagrams.
- Message generated by application layer has to be less than 65 kB so that it can fit into a single datagram in UDP.



a. UDP user datagram



b. Header format

Advantages of UDP

- Since its connectionless, the overhead required for establishing the connection is reduced and hence the *size and delay of transmission decreases* - extremely useful for multicasting or broadcasting.
- Very useful when simple and small packets are to be sent – TCP requires 9 packets to be exchanged even for simple requests, while UDP needs 2 packets.
- UDP does not resend packets in case they are lost, thus it does not create more traffic in the network.
- Use of UDP depends on the requirement of the application.

Sequence of commands in UDP

- Since UDP is a connectionless protocol, the client-server just create a socket and start the data transfer.
- Thus only 3 commands are executed from the client side:
 1. `sock_init()` – create a socket
 2. Read/write – read/write data
 3. `Close()` – shutdown
- From server side also, only 4 commands are executed – `sock_init()`, `bind()`, read/write, and `close()`.

IoT Transport Layer – TCP vs UDP

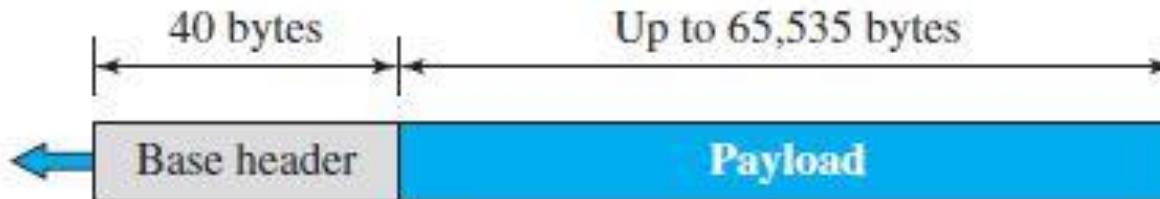
TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Connection oriented protocol i.e. establishing of connection before transmission of packet is necessary.	Datagram oriented protocol i.e. connection establishing is not necessary for transmission.
Large overhead for connection establishment.	Very low overhead.
Reliable as connection is established.	Non-reliable.
Provides extensive error checking mechanisms.	Basic error checking only.
Sequencing of data i.e. arriving of packets in particular order is achieved.	No sequencing of data.
20 – 60 Bytes header.	8 byte header.
Retransmission of packets is possible.	Retransmission not possible

IPv6 (Internet Protocol version 6)

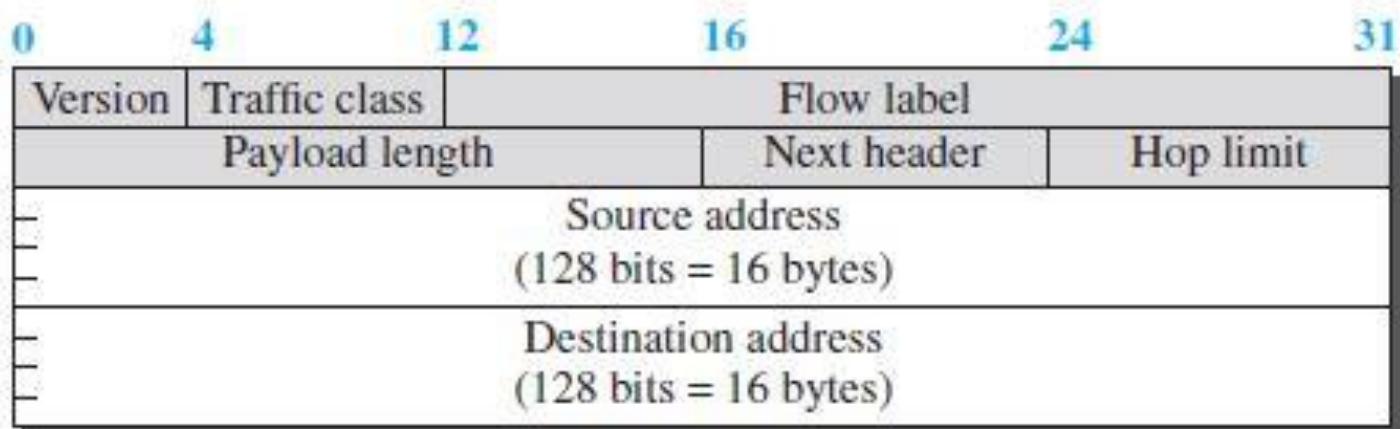
- Networking protocol that allows identification of computers and routing of data packets based on IP addresses of source, destination and intermediate routers/gateways.
- It's a modification of IPv4 with increased number of addresses possible.
- All packets are routed individually towards the destination and may reach the destination at different times and must be reassembled there.
- Allows for multicasting of messages.

IPv6 Packet Format

- Uses 128 bits (16 bytes) address to identify a computer/node.
- Header occupies 320 bits (40 bytes)
- Total packet size – 64 kB
- The header shown in fig is just a ‘base header’ which contains only the basic functionalities of IPv6



a. IPv6 packet



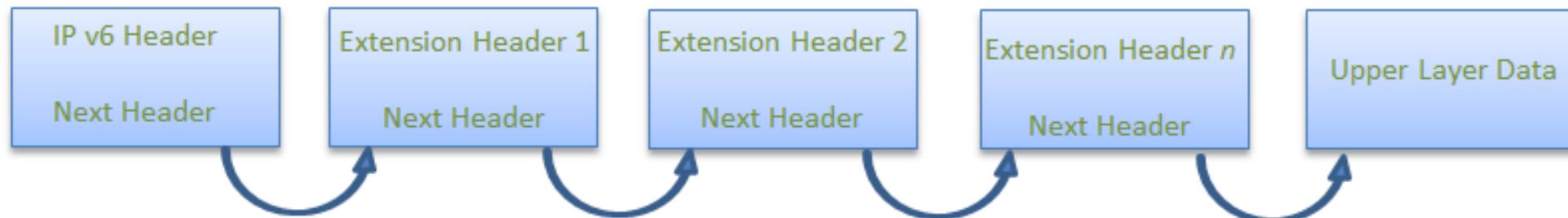
b. Base header

IPv6 Packet Format

- Traffic class – indicates the class or priority of the packets
 - helps routers to handle the traffic based on priority of packets
 - in case of congestion, packets with least priority are dropped
 - priority can be set by the source node and routers can change it
- Flow label – used by the source to label the packets belonging to the same flow
 - allows the routers to identify the packets of the same flow
 - along with flow label, the source also specifies the lifetime of the flow
- Hop limit – maximum number of intermediate nodes a packet is allowed to travel
 - its value decreases by one each time the packet passes through a router
 - if its value reaches 0, the packet is discarded
 - used to discard the packets stuck in infinite loop due to routing error

IPv6 Packet Format

- The ‘next header’ field points to another header known as ‘extension header’ which has advanced functionalities related to IPv6.
- Next header field in the base header points to the first extension header, and this first extension header points to second extension header and so on.
- Thus, there can be 0, 1 or multiple extension headers in a message.

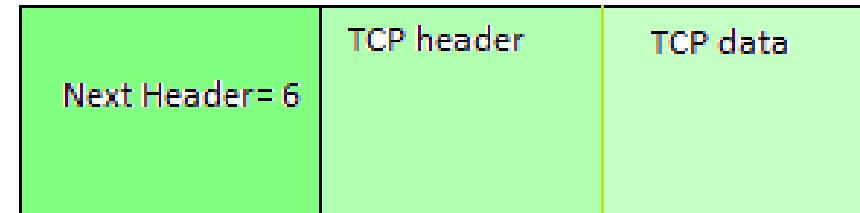


IPv6 Packet Format

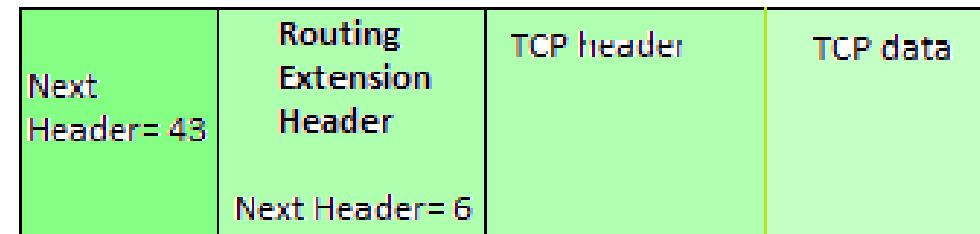
- The value in the ‘next header’ field and the corresponding extension header types are shown in the table below.

Order	Header Type	Next Header Code
1	Basic IPv6 Header	-
2	Hop-by-Hop Options	0
3	Destination Options (with Routing Options)	60
4	Routing Header	43
5	Fragment Header	44
6	Authentication Header	51
7	Encapsulation Security Payload Header	50
8	Destination Options	60
9	Mobility Header	135
	No next header	59
Upper Layer	TCP	6
Upper Layer	UDP	17
Upper Layer	ICMPv6	58

Example: TCP is used in IPv6 packet



Example2:



IPv6 Packet Format – Extension Header

- If next header = 59, then there is no extension header present
- If next header = 6, then the next header is of the upper layer i.e. TCP
- If next header = 17, then the next header is of the upper layer i.e. UDP
- The functionalities of some of the extension headers is as shown in the table.

Ext. Header	Description
Hop-by-Hop Options	Examined by all devices on the path
Destination Options (with routing options)	Examined by destination of the packet
Routing Header	Methods to take routing decision
Fragment Header	Contains parameters of fragmented datagram done by source
Authentication Header	verify authenticity
Encapsulating Security Payload	Carries Encrypted data

IPv6 Packet Format

- Hop-by-hop options header is examined and altered by every node in the packet's path. Options may include information about authentication.
 - Destination options header is examined only by the destination node and cannot be altered by the intermediate nodes.

Hop-by-Hop Options and Destination Options extension header format

IPv6 Packet Format

- Routing extension header is used to direct a packet to one or more intermediate nodes in its path. It also specifies the methods to route the packet.
 - The format of routing extension header is as shown below:

Routing extension header format

6LoWPAN

- *IPv6 over Low Power Wireless Personal Area Network* - allows transmission of IPv6 packets in a network of resource constrained devices.
- Based on IEEE 802.15.4 standard.
- Performs extensive compression (encoding) of IPv6 headers by removing unnecessary fields and keeps other services intact.
- Other network layer protocols available for network discovery and management (such as SNMP, SLP) can be used by resource constrained sensor networks.

6LoWPAN Headers

- 6LoWPAN packets do not have a fixed header size and they use the concept of stacked header i.e. header size varies with the requirements of the packet.
- Types of headers defined in 6LoWPAN
 - Dispatch header – base header
 - Mesh header – type of extension header
 - Fragmentation header – type of extension header
 - HC1 header
- In simplest case – only Dispatch and HC1 header are required.
- When large packets are sent – Fragmentation header is required
- When mesh networking is done – Mesh header is required

Header Compression in 6LoWPAN

IPv6 header

Ver	Traffic class	Flow label	Payload length	Next header	Hop limit	Source address 64-bit prefix, 64-bit HD	Destination address 64-bit prefix, 64-bit HD	40 bytes
-----	---------------	------------	----------------	-------------	-----------	--	---	----------

1. Compressed header, FE80::CAFE:00FF:FE00:0100 → FE80::CAFE:00FF:FE00:0200

Dispatch	Compr. header	2 bytes
----------	---------------	---------

2. Compressed header, 2001::DEC4:E3A1:FE24:9600 → 2001::4455:84C6:39BB:A2DD

Dispatch	Compr. header	CID	Hop limit	Destination address 64-bit HD	12 bytes
----------	---------------	-----	-----------	----------------------------------	----------

3. Compressed header, 2001::DEC4:E3A1:FE24:9600 → 2001::4455:84C6:39BB:A2DD

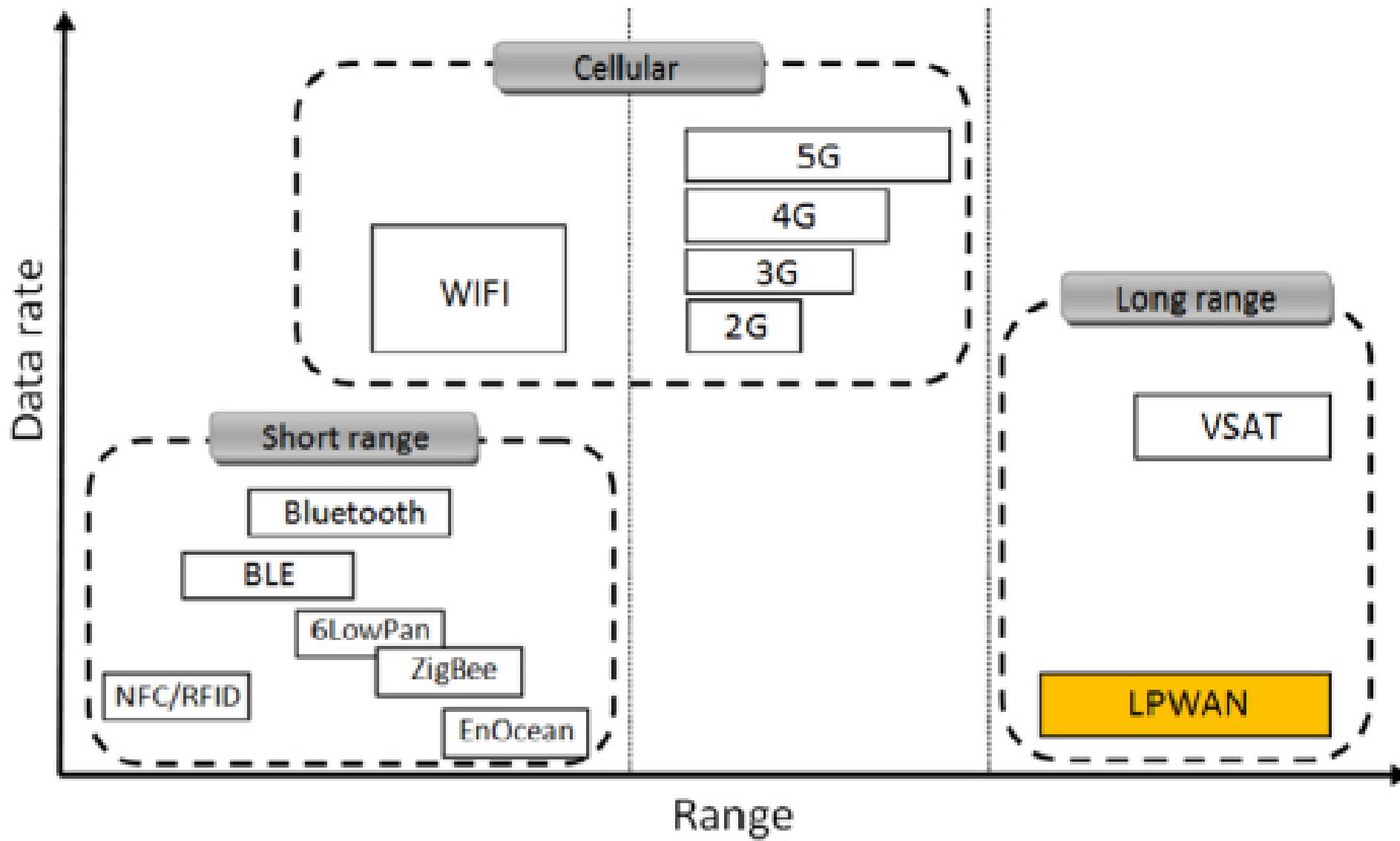
Dispatch	Compr. header	CID	Hop limit	Source address 64-bit prefix	Destination address 64-bit prefix, 64-bit HD	20 bytes
----------	---------------	-----	-----------	---------------------------------	---	----------

6LoWPAN Headers

- A 2 byte header is used when data is to be transmitted between the devices in the same 6LoWPAN network.
- Here, devices are identified using the MAC address and no IP address is needed.
- 12 byte header is used when the destination node is outside the 6LoWPAN network
- 20 byte header is used when both source and destination nodes are outside the 6LoWPAN network.

IPv6	6LoWPAN
Most commonly used networking protocol in web applications	Transmit IPv6 packets over network of resource constrained devices
Header – 40 bytes	Header – 2, 12 or 20 bytes
Packet size – 64 kB	Packet size – 127 bytes
IP address of every node – 128 bit	Address of every node – 16 bit
Uses OFDM	Uses QPSK
Less delay and more throughput	More delay and less throughput
Better packet re-ordering rate	Poor packet re-ordering rate

Communication Technologies



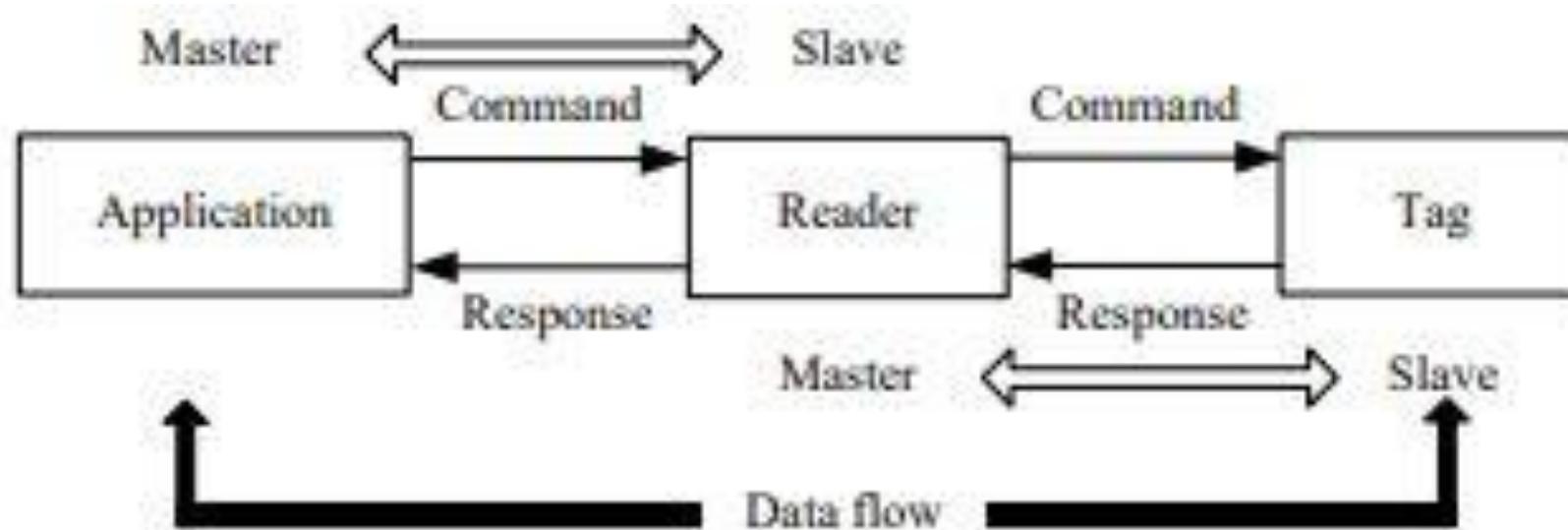
RFID (Radio Frequency Identification)

- Radio Frequency Identification – a short distance wireless communication system consisting of a reader (Interrogator) and a tag (Transponder).
- RFID works in different frequency ranges
 1. Low Frequency (LF) RFID – 125 to 134 kHz – Animal Tracking
 2. High Frequency (HF) RFID – 13.56 MHz – NFC
 3. Ultra High Frequency (UHF) RFID – 433 and 860 MHz (865 – 868 MHz)
 4. Microwave RFID – 2.4 GHz

(https://www.youtube.com/watch?v=um8HZIPED-k&ab_channel=AveryDennison)

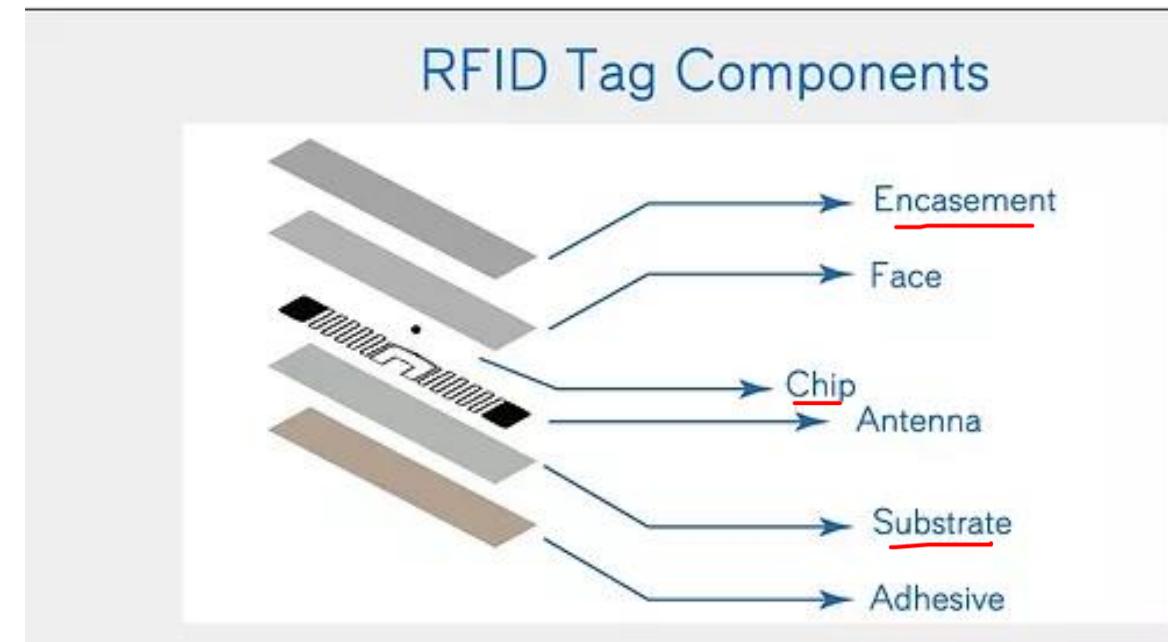
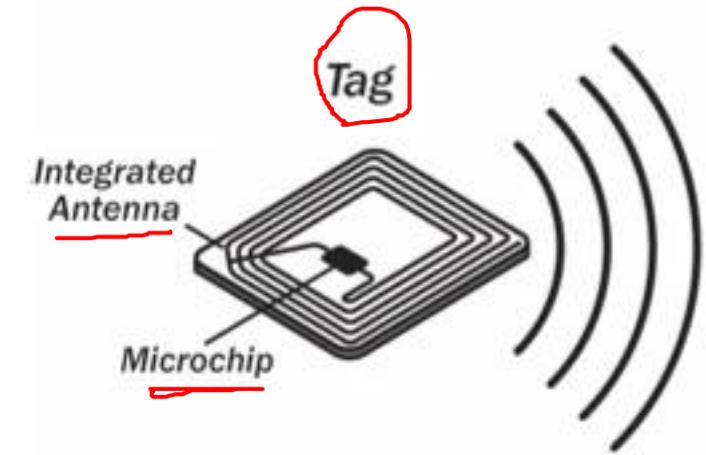
RFID (Radio Frequency Identification)

- 3 types of RFID technology
 - Active – tag is internally powered
 - Passive – tag derives energy from EM wave
 - Chipless – energy harvesting from environment



RFID (Radio Frequency Identification)

- A typical RFID tag is as shown in the figure
- It contains an antenna and a chip adhered to a substrate (maybe flexible or rigid).

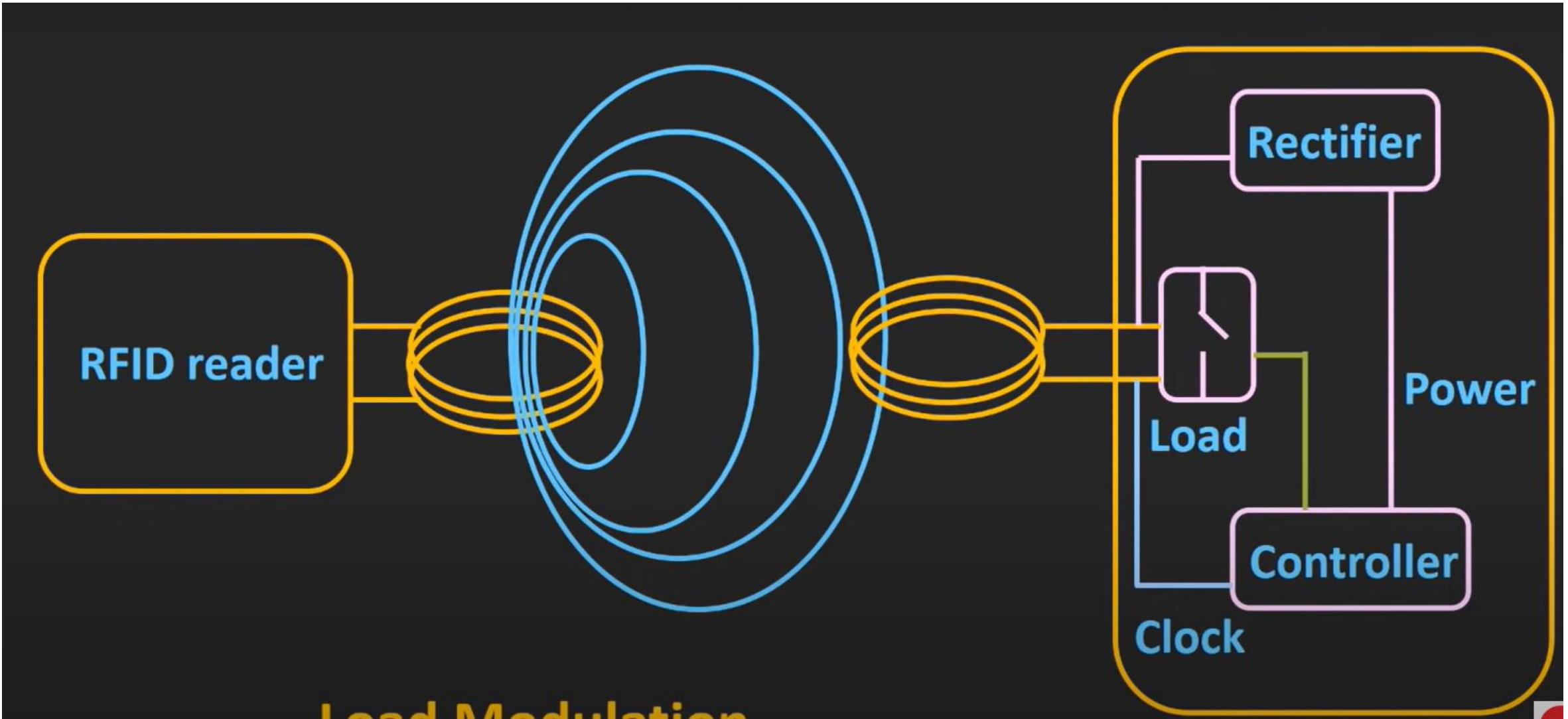


RFID (Radio Frequency Identification)

- Choosing the correct RFID tag for your application depends on the following
 - Surface to tag (metal, plastic etc)
 - Type of attachment (adhesive, epoxy, metal screws etc)
 - Type of environment (hot, cold, humid)
 - Size limitations
 - Reading range
- The antenna of reader and tag must be of the same polarization to achieve maximum power transfer and longer range.
- The tag and reader must be complaint with EPC – Gen 2 protocol which is an air interface protocol defining the physical and logical requirements of an RFID system involving interrogators and passive tags.

RFID Working Principle

- Inductive Coupling – for small distances



RFID Working Principle

- In this method the reader and tag contain a coil which is used to transfer energy from the reader to tag and data from tag to reader.
- The working principle is similar to that of a transformer.
- The varying magnetic field of the reader induces a voltage in the tag which powers it up and activates it.
- The data present in the tag then modulates its magnetic field which in turn modulates the voltage at the reader. The controller present in the reader decodes this change and extracts the data sent by the tag.

RFID Working Principle

- For larger distances between reader and tag, Electromagnetic Coupling is used.
- Here, the reader sends an EM signal through its antenna towards the tag. The tag gets powered up and send back a weak signal through its antenna.
- This signal is known as backscattered signal whose intensity (amplitude) varies according to the data stored in the tag.
- This variation in amplitude of backscattered signal induces an equivalent impedance change in the reader which demodulates it to extract the data.

RFID (Radio Frequency Identification)

- An RFID tag contains different types of memory banks
 - 1. Reserve Memory
 - 1. Used to store the passwords
 - 2. EPC Memory
 - 1. Stores the Electronic Product Code (EPC) information
 - 2. 96 bits of memory
 - 3. TID (Tag ID) Memory
 - 1. Used to store the unique ID of the tag (like MAC ID of a device)
 - 4. User memory – stores the data or information

RFID (Radio Frequency Identification)

- External sensors may be connected to RFID tags using I2C or SPI connections.
- When the reader interrogates the tag, these sensors get powered up and their data is stored in the user memory of the tag.
- Tag then backscatters this data towards the reader where it is decoded.

RFID (Radio Frequency Identification)

- Key challenges in a RFID system are:
 1. Collision due to simultaneous tag responses – reduction in throughput, wastage of bandwidth, energy.

RFID (Radio Frequency Identification)

- RFID uses Framed Slotted ALOHA protocol along with Binary Tree Splitting to avoid collisions in the MAC layer and increase the throughput.
- This is useful when multiple tags are read in the same time slot and there are high chances of collision of packets arising from each tag.

LPWAN Technologies

- LPWAN - Low Power Wide Area Networks
- Interconnects low-bandwidth, battery powered devices with low bit rates over long distances.
- Provides low power and low cost line of sight communication characteristics up to 10-40 km in rural areas and 1-10 km in urban areas
- LPWAN is not a single technology, but a group of various similar technologies that have different applications

Need for LPWAN Technologies

- ZigBee, Xbee and WiFi have very short range and need dense deployment to provide better connectivity.
- Many routers and gateways make their deployment expensive.
- Cellular technologies (3G, 4G) provide wide coverage but consume large amount of power.
- A technology which can provide range more than ZigBee and WiFi while being less complex and less power consuming than 3G and 4G is needed for IoT applications

Characteristics of LPWAN Technologies

- Long Range Communication
 - Use of sub-1 GHz band
 - Narrowband and spread spectrum modulation techniques leading to less noise and interference (data rate is sacrificed)
- Ultra-low Power Operation
 - Use of star topology where each node is connected to a central node where maximum data processing is performed, thus reducing power requirements of sensor nodes.
 - Duty cycling – devices are turned off when not in use and wake up only when an event occurs
 - No carrier sensing protocol – ALOHA is used (if you have data, transmit it)

Characteristics of LPWAN Technologies

- Low cost operation
 - Minimum infrastructure – a single base station can handle thousands of nodes as data rate and bandwidth of each is less
 - Unlicensed spectrum is used
- Scalability
 - Diversity in time, space and frequency is efficiently utilized

Classification of LPWAN Technologies

- LPWAN technologies can be classified into 2 categories – *proprietary and standard-based.*
- Proprietary LPWAN Technologies
 - SigFox
 - Ingenu
 - Telensa
 - Qowisio
 - Nwave
- Standards-based LPWAN
 - LoRa and LoRaWAN
 - Weightless
 - NB-IoT
 - LTE-M

LoRa Technology

Traditional Cellular

Long Range
High Data Rates
Low Battery Life
High Cost



LPWAN (3-5B in 2022)

Long Range
Low Data Rates
Long Battery Life
Low Cost
High Capacity Potential

Cat-M1

Long Range
High Data Rates
Low Battery Life
Medium Cost

Local Area Network (Wi-Fi)

Short Range
High Data Rates
Low Battery Life
Medium Cost

Narrow-Band IoT (NB-IoT)

Stationary Devices
Short Range (indoor coverage)
Low Data Rates
Good Battery Life
Low Cost

Personal Area Network (Bluetooth®)

Very Short Range
Low data rates
Good Battery Life
Low Cost

LoRa

- LoRa is a *physical layer modulation technology* developed by Semtech, operates in 868, 915 and 433 MHz range.
- Modulates the signals using CSS (chirp spread spectrum) technique. Bandwidth (BW) – frequency range over which the modulated signal is spread
- 6 different spreading factors (SF) are used. Higher SF allows longer range and lower SF allows higher data rate.
- Combines forward error correction (FEC) to improve receiver sensitivity.
- Data rate ranges from 300 bps to 50 kbps depending upon spreading factor and channel bandwidth.

LoRa

- The sensor nodes can transmit on any channel available at any time using any data rate, as long as the following rules are observed
 1. The end device changes the channel in pseudo-random fashion for every transmission – this frequency diversity ensures less interference.
 2. The sensor node changes its transmit periodicity (duty cycle) in a pseudo-random manner – this prevents synchronization between sensor nodes.
 3. The sensor node complied with all the regional regulations governing its behavior in the band it is currently operating including limitations on duty cycle, transmit time etc.

LoRa

- Increasing the SF increases the packet's time on air (ToA), thus increasing its range and the probability of being accurately received. But this also increases the likelihood of collisions with other packets, thus saturating the receiver's demodulation channels and exceeding the permissible duty cycle limitations.
- Sensor nodes close to the gateway should transmit data at lower SF as very little link budget (range) is needed. While a sensor node far away from the gateway must transmit at higher SF to increase its packet's ToA.
- Transmitting at lower SF also reduces the battery consumption of the nodes.

LoRa

- For the regularly utilized frequency bands (863 – 868 MHz), the duty cycle limit is set at 1% i.e. the device can send the data for only 1% of the time.
- Each LoRa symbol consists of 2^{SF} chirps i.e. 2^{SF} bits per symbol.

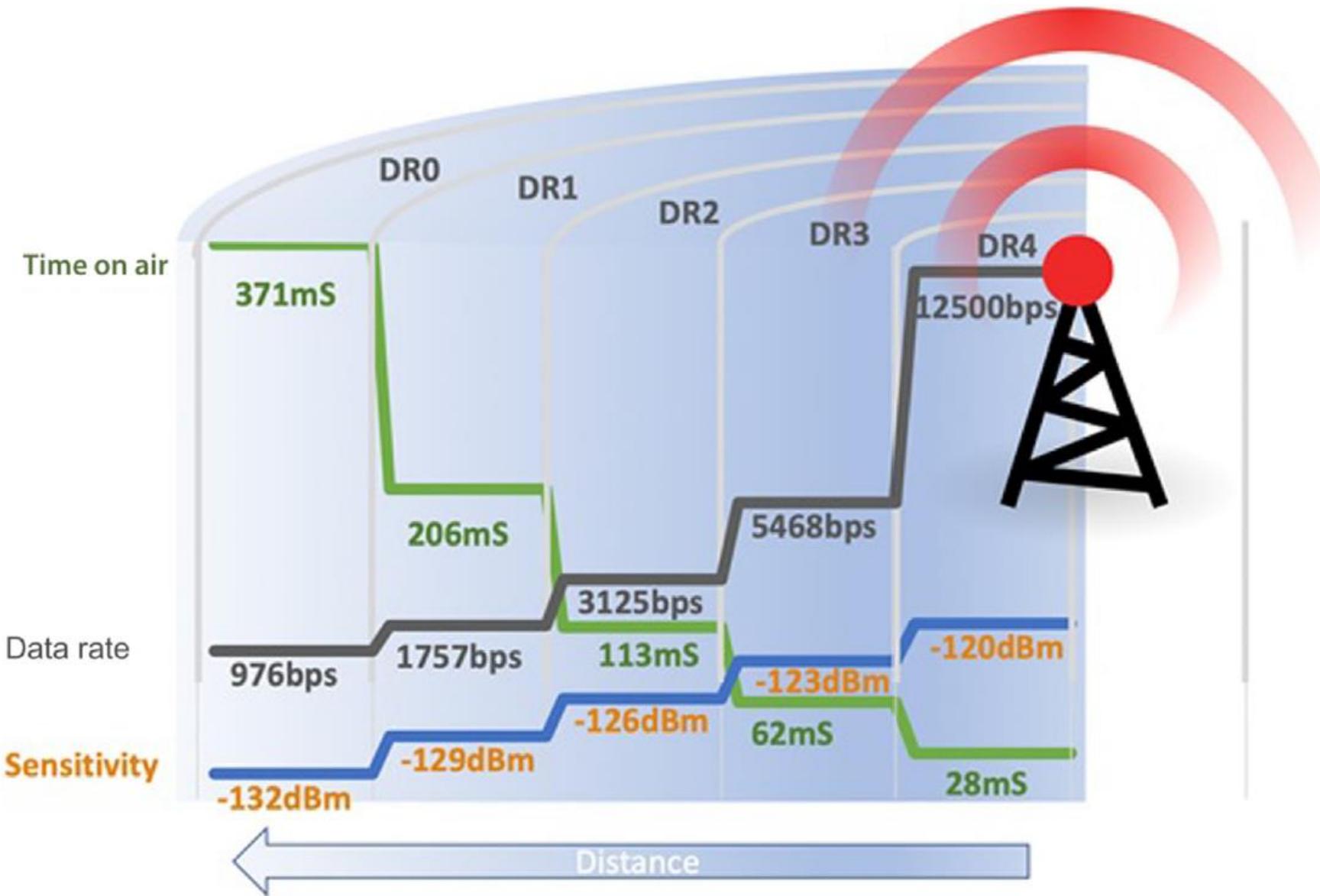
LoRa

- Details of different SFs is given in the table below

Spreading Factor (For UL at 125 KHz)	Bit Rate	Range (Depends on Terrain)	Time on Air for an 11-byte payload
SF10	980 bps	8 km	371 ms
SF9	1760 bps	6 km	185 ms
SF8	3125 bps	4 km	103 ms
SF7	5470 bps	2 km	61 ms

- Signals modulated at different SFs and transmitted on the same frequency do not interfere with each other i.e. SFs are orthogonal.

LoRa



LoRa

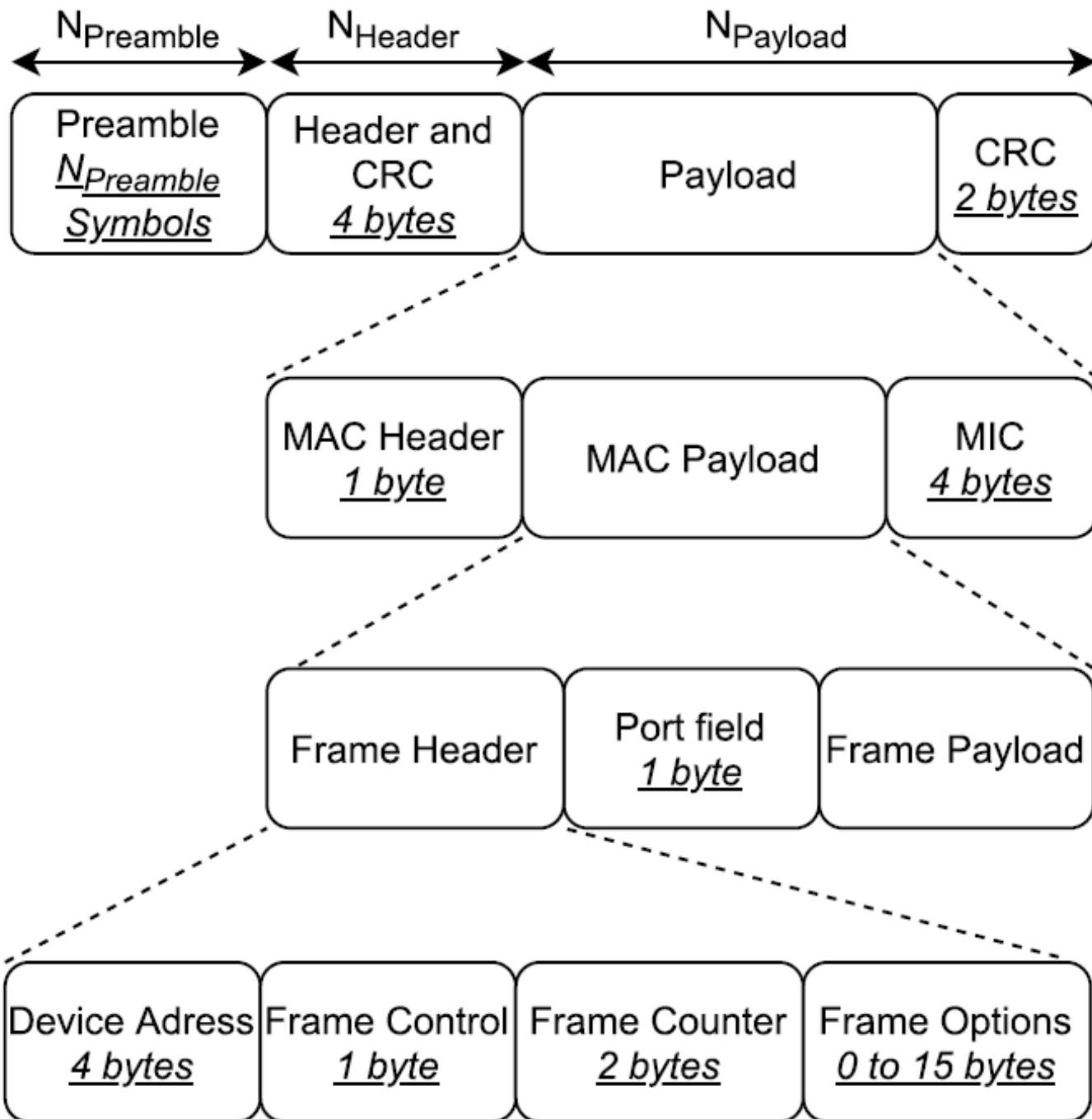
- Coding rate is used for error correction and is given as

$$CR = \frac{4}{4+n}$$

n can be between 1 and 4. Smaller the code, longer the transmission time.

- CR is the degree of redundancy implemented by the FEC.
- LoRa has a variable data rate given by $R_b = SF \times \frac{BW}{2^{SF}} \times CR$

- The physical layer message format of LoRa is as shown
- CRC field allows receiver to ignore packets with invalid header.
- Message Integrity Check (MIC) is the payload's digital signature.

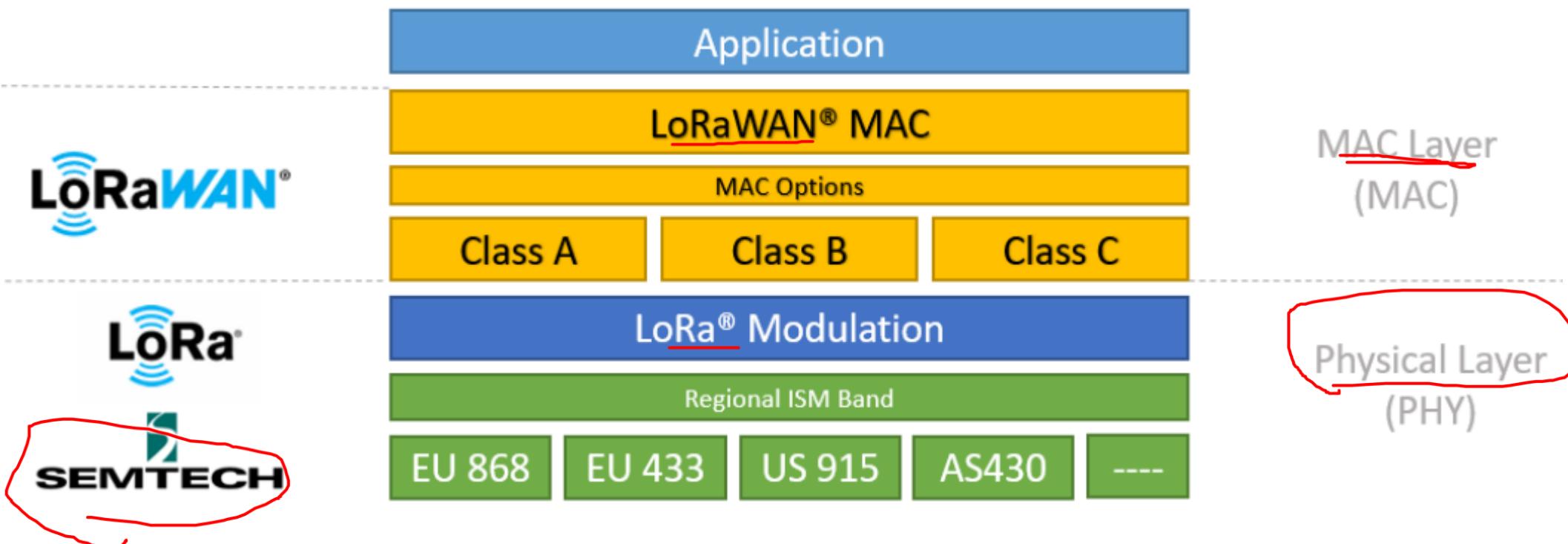


LoRaWAN

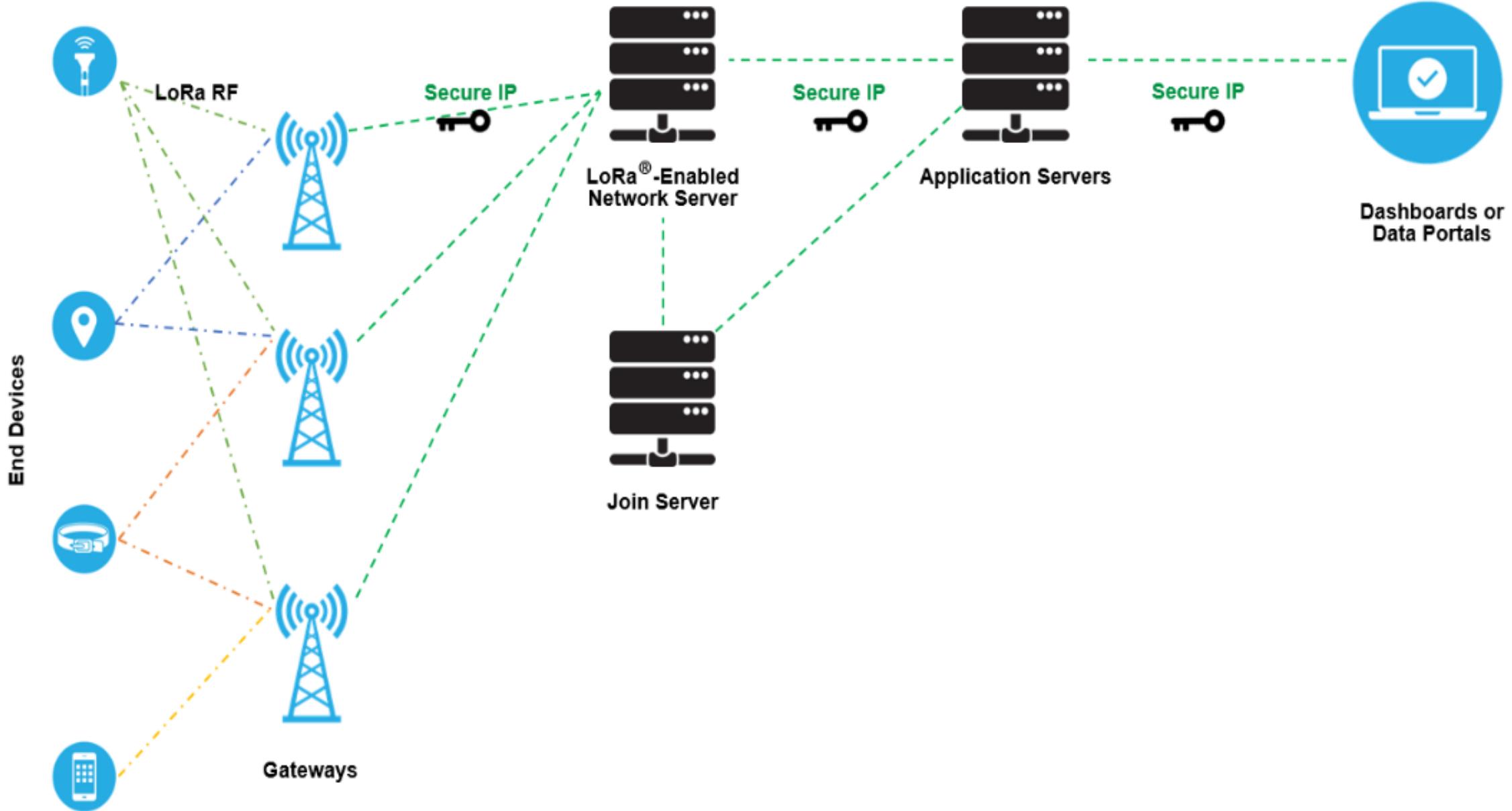
- LoRaWAN is the networking (MAC Layer) protocol of LoRa which is open-source unlike that of SigFox and NB-IoT. This has led to widespread deployment of LoRa.
- LoRaWAN is an open protocol that delivers secure bidirectional communication, mobility and localization services standardized and maintained by the LoRa alliance.
- LoRaWAN has less equipment and installation cost. It is also highly flexible as it allows fine tuning of several parameters.

LoRaWAN

- As can be seen, LoRa is a physical layer technology while LoRaWAN is a MAC layer protocol



LoRaWAN Network Architecture



LoRaWAN

- Each LoRaWAN end device is connected to multiple gateways in the area and each data packet send by the end device is received by all the gateways.
- This greatly reduces the packet error rate and allows for low cost geo-location of mobile nodes i.e. mobile nodes need to spend much battery power in geo-location services.
- The gateway simply check the integrity of the messages received from the end nodes and forward them to the LoRa network servers by attaching RSSI values to the message.
- As network server may receive multiple copies of the same message from different gateways, it perform de-duplication and deletes the copies.
- While sending a downlink message, the network server selects the gateway with highest RSSI value as it is the one closest to the end device.

LoRaWAN

- Gateway is connected to the network server via the Internet.
- Network server manages the entire network by dynamically controlling the parameters.
- Establishes secure 128-bit AES connections for end-to-end data transfer.
- Ensures the authenticity of every end node by performing address checking.
- Performs queueing of downlink payload coming from any application server to any device connected to the network.
- Forwarding join-request and join-accept messages between the devices and the join server.

LoRaWAN

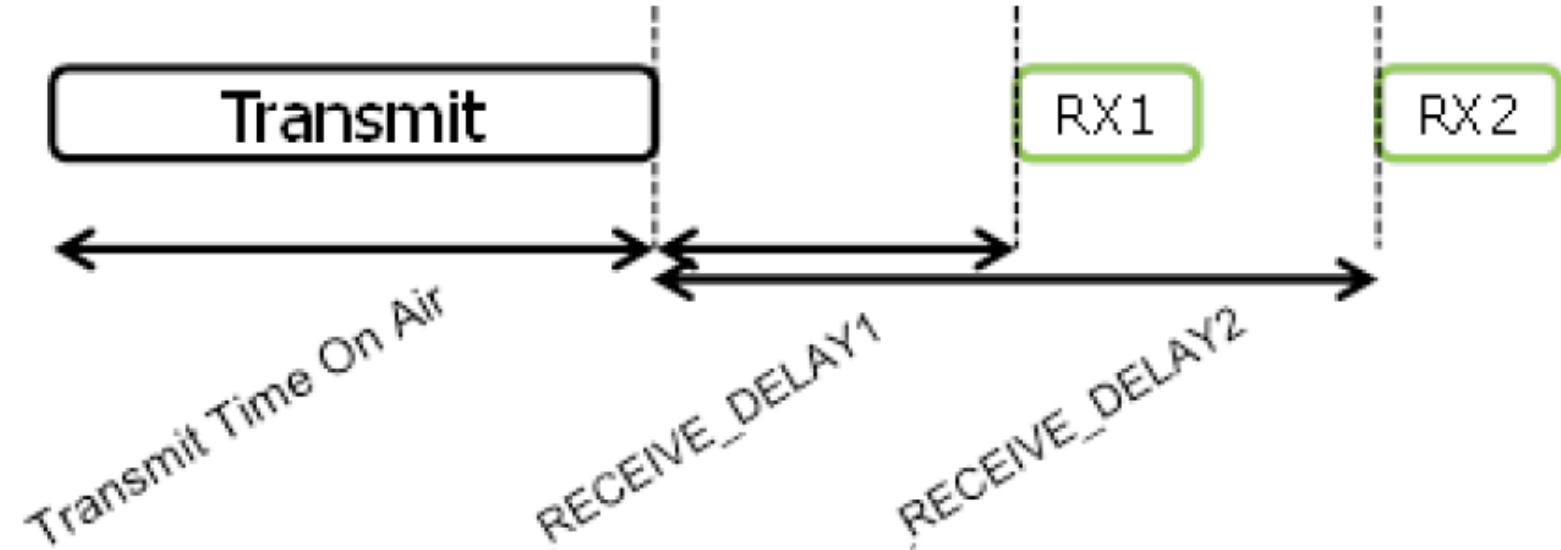
- Join server manages *over-the-air activation* of the end devices to be added to the network using symmetric key cryptography.
- It signals to the network server which application server should be connected to a end device and performs encryption key derivations.
- It communicates Network Session Key of the device to the network server and Application Session Key to the application server.
- The join server must contain the following information for each end-device
 - DevEUI (End-device serial unique identifier)
 - Appkey (Application Encryption key)
 - NwkKey (Network Encryption Key)
 - Application Server Identifier
 - End-device service profile

LoRaWAN

- Class A devices
 - communication is scheduled by the end node whenever it is needed
 - each uplink transmission is followed by 2 short downlink messages called receive windows
 - has lowest power consumption but highest latency
 - downlink communication starts only when end node has sent an uplink message. For all other time, there is no downlink message which saves power.
 - the first RX window uses the same frequency channel as the uplink frame while second RX window uses fixed configurable frequency and data rate.

LoRaWAN – Class A Devices

- Class A devices
 - end node SHALL open one or two receive windows after each uplink transmission
 - If no packet destined for the sensor node is received during RX1, it SHALL open RX2.
 - If a preamble is received in any receive windows then sensor node should remain active till the downlink frame is demodulated
 - If a frame is detected and successfully demodulated during RX1, then the sensor node SHALL NOT open RX2.



LoRaWAN – Class A Devices

- First receive window (RX1) uses frequency and data rate that are functions of uplink frequency and data rate and this relation is region-specific.
- Second receive window (RX2) uses fixed, configurable frequency and data rate which can be modified using MAC commands. The default frequency and data rate are region specific.

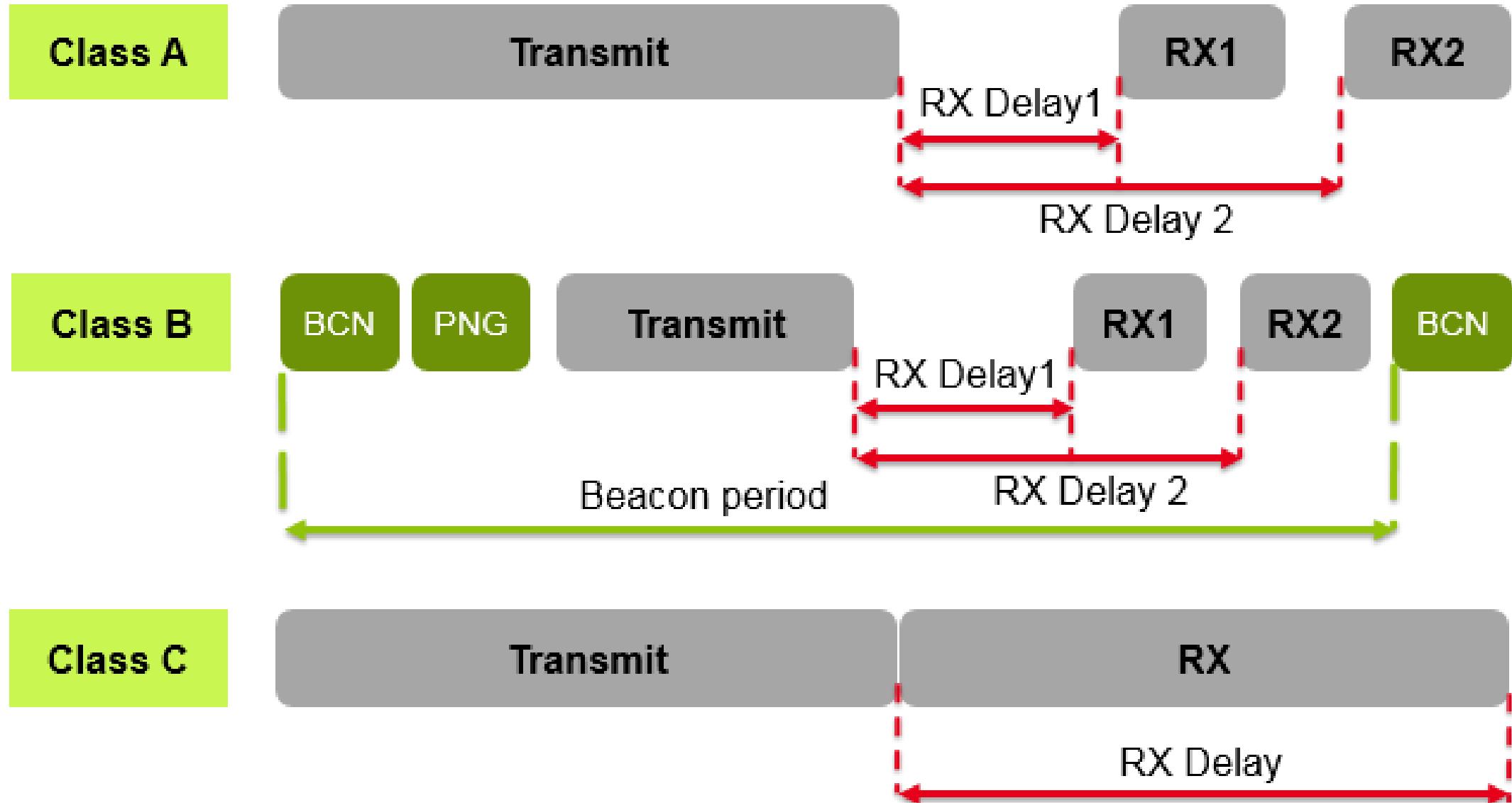
LoRaWAN

- Class B devices – bidirectional devices with scheduled time slot
 - the end devices open receive windows periodically
 - gateway initiates a downlink transmission by sending a beacon every 128 secs
 - to receive the beacon the end device must keep its receiver on for at least one beacon period
 - on receiving the beacon, the end device opens its receive window
 - consumes more power but has less latency as compared to class A operation

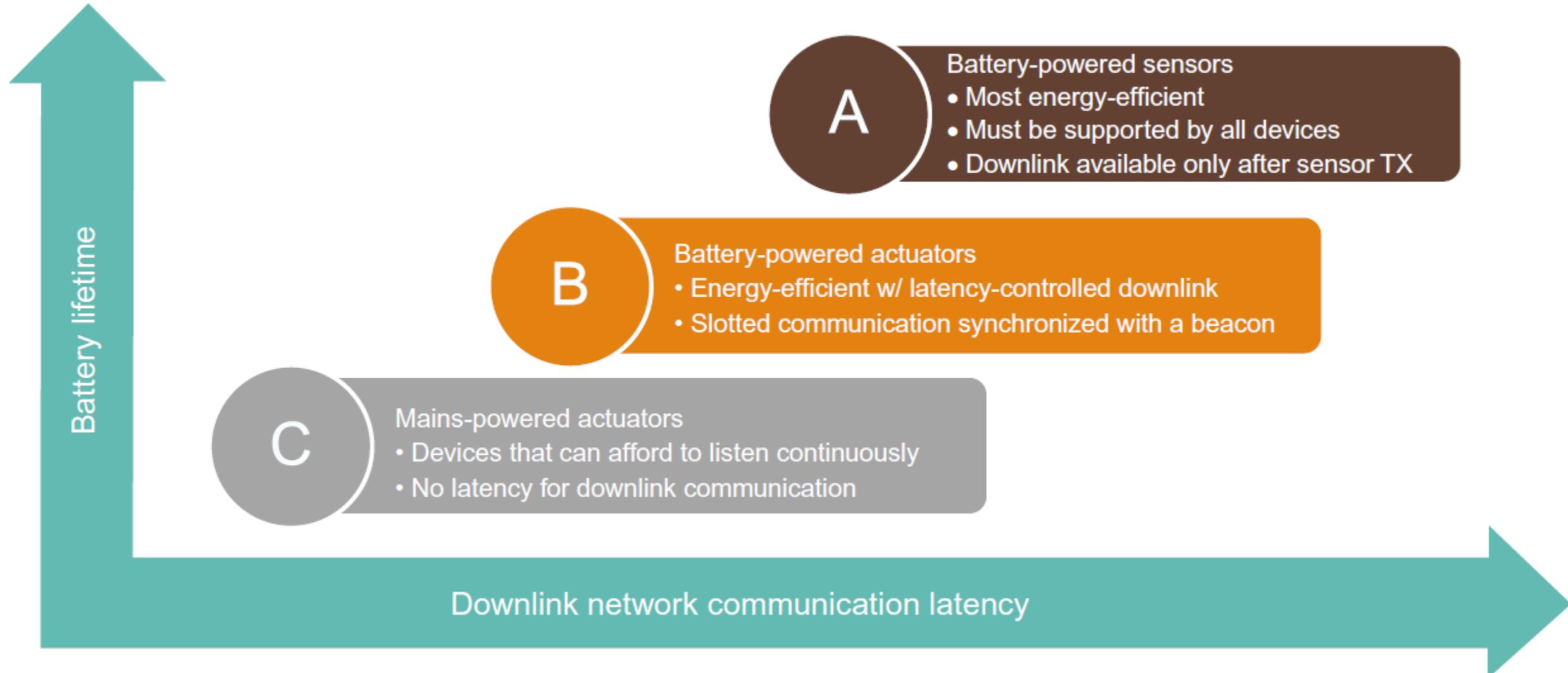
LoRaWAN

- Class C devices
 - receive windows of end nodes are always open (except when transmitting)
 - consumes highest power but has lowest latency
- LoRaWAN devices are typically connected in star-of-stars technology where each end device transmits to all gateways in its range rather than having direct link to a single gateway.
- Standard LoRaWAN doesn't allow node-to-node communication, though this can be implemented by using different MAC layers built on LoRa modulation.
- Key metrics used to quantify the QoS of LoRaWAN are packet delivery ratio (PDR), RSSI, SNR and throughput.

LoRaWAN



LoRaWAN



LoRaWAN Packet Format

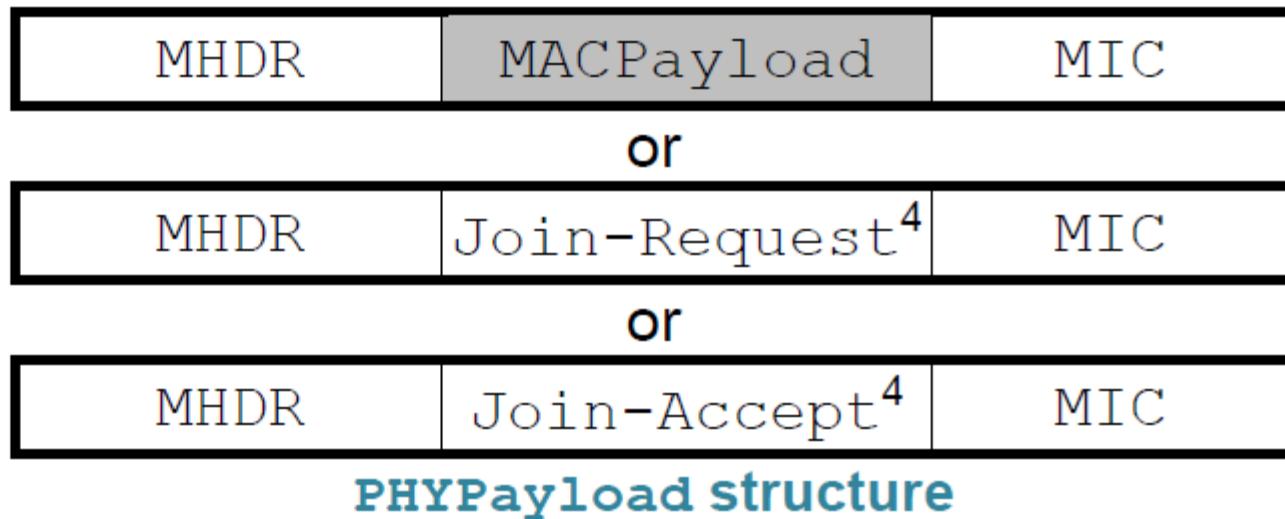
- Each uplink and downlink packet carries a PHY payload – starting with a single octet MAC header (MHDR) followed by MAC Payload and ending with 4 octet Message Integrity Code (MIC).

Size (octets)	1	$7..M$	4
PHY Payload	MHDR	MAC Payload	MIC

- The maximum length of MACPayload field (M) is region-specific and data rate specific.

(From the document “LoRaWAN Link Layer Specification v1.0.4”)

PHYPayload:



MACPayload:



FHDR:



LoRaWAN Packet Format

- The 8-bit long MAC header contains the following fields

Bits	[7:5]	[4:2]	[1:0]
MHDR	FType	RFU	Major

- FType – Frame Type – LoRaWAN distinguishes among following frame types

FType	Description
000	Join-Request
001	Join-Accept
010	unconfirmed data uplink
011	unconfirmed data downlink
100	confirmed data uplink
101	confirmed data downlink
110	RFU
111	Proprietary

(From the document “LoRaWAN Specification”)

LoRaWAN

1. <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>
2. <https://lora-alliance.org/lorawan-for-developers/>
3. <https://docs.arduino.cc/learn/communication/lorawan-101>
4. <https://turborobo.in/arduino-mkr-wan-1300-lora-connectivity>

SigFox

- Operates in 868, 915 and 433 MHz with *ultra-narrowband 100 Hz channels.*
- Very low noise levels, high receiver sensitivity, inexpensive antenna design.
- Maximum data rate of 100 bps – slowest among all LPWAN technologies
- Max number of messages limited to 140/day over uplink and 4/day over downlink
- For reliability, SigFox transmits the message multiple times which results in increased power consumption.
- SigFox is deployed by network operators and the users need to pay subscription charges, while LoRa can be deployed as an own independent network with no need of subscription charges.

NB-IoT

- Based on LTE standard and uses licensed frequency bands.
- Three modes of operation possible
 - Standalone operation – utilizing the existing GSM frequencies
 - Guard band operation – utilizing LTE carrier's guard bands
 - In-band operation – utilizing resource blocks within a LTE carrier
- Removes many features of LTE such as handover, carrier aggregation, dual connectivity etc.
- Has much better QoS than SigFox or LoRa but at higher cost.

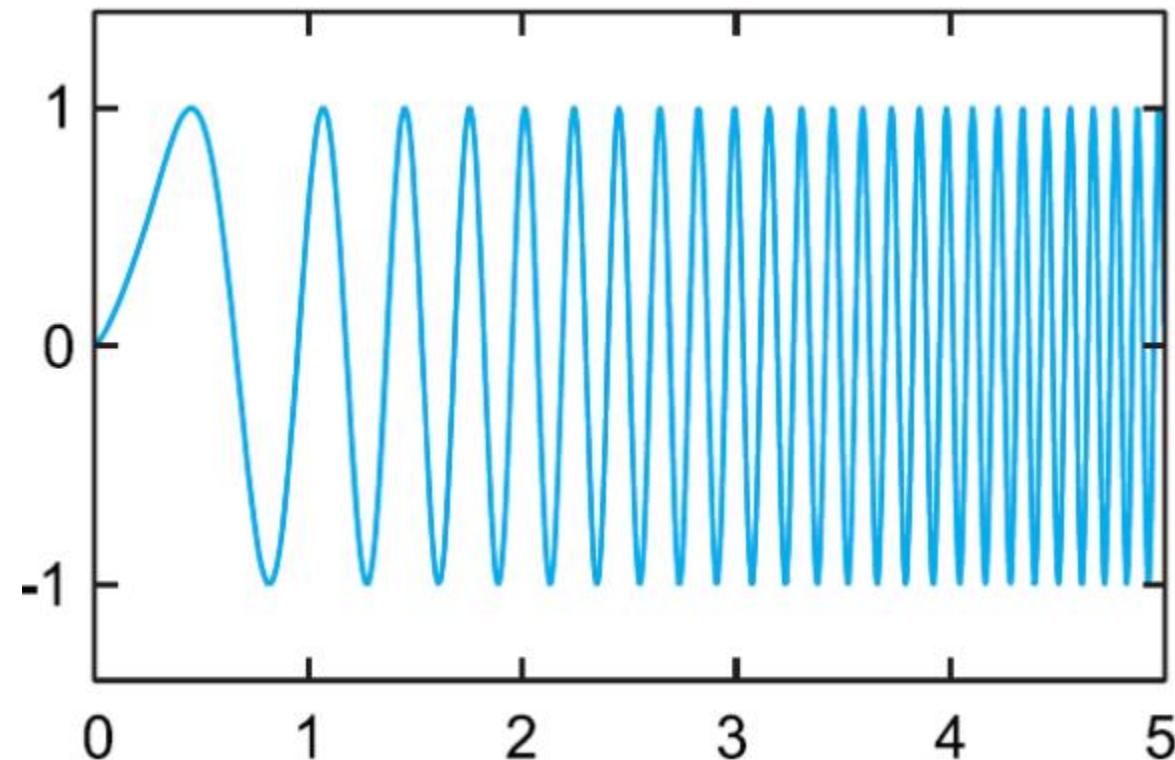
Comparison of different LPWAN technologies

LPWAN Technology	Reported Range (Max)	Max Data Rate	Frequency	Modulation	Max Payload Length	Security	Bandwidth
LoRaWAN	15 km (Rural), 2 km (Urban)	27 kbps	868, 433 MHz	CSS, FSK	243 bytes	AES	250 kHz
SigFox	40 km (Rural), 10 km (Urban)	100 bps	868, 433 MHz	BPSK	12 bytes	No	100 Hz
Ingenu	6 km	624 kbps	2.4 GHz	DSSS			
CC1310	25 km	1.2 kbps	868, 433 MHz	GFSK			
NB IoT	10 km (Rural), 1 km (Urban)	200 kbps	Licensed LTE band	QPSK	1600 bytes	LTE	200 kHz

Type of modulation	Spectral efficiency (bits/s/Hz)
FSK	<1 (depends on modulation index)
GMSK	1.35
BPSK	1
QPSK	2
8PSK	3
16QAM	4
64QAM	6
OFDM	>10 (depends on the type of modulation and the number of subcarriers)

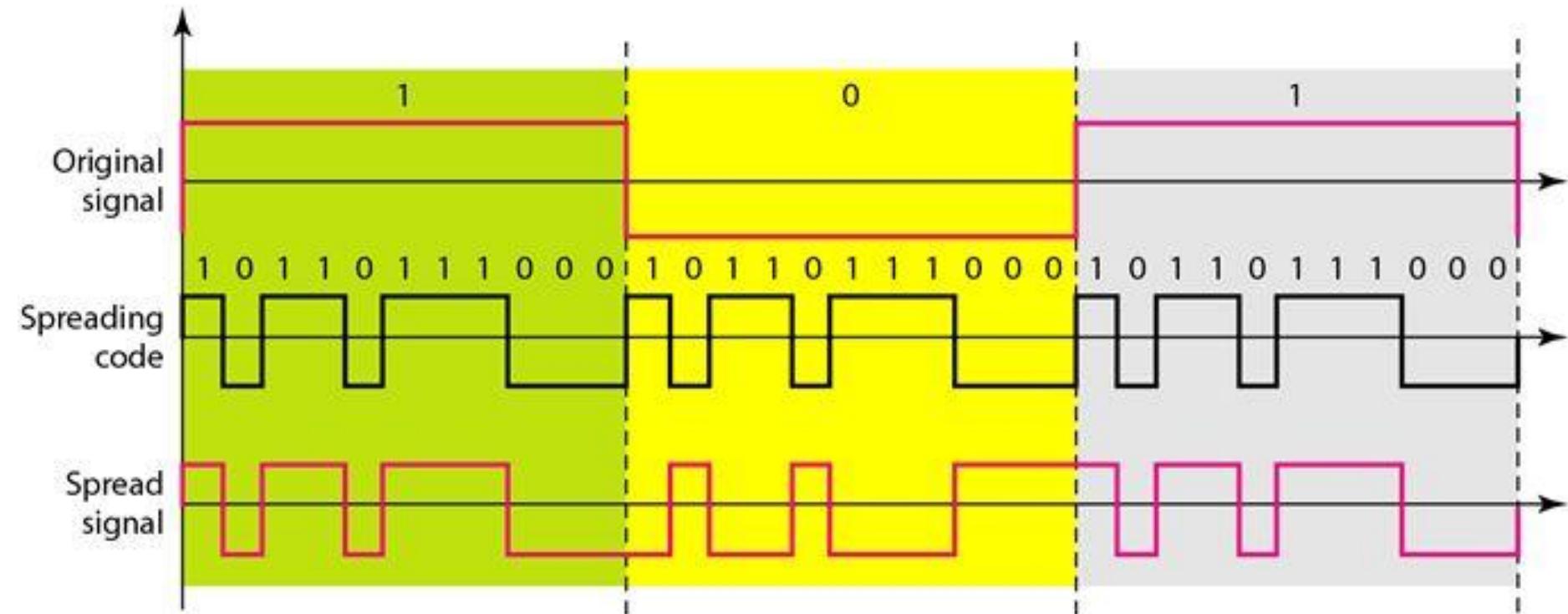
Chirp Spread Spectrum (CSS)

- Chirp is a sinusoidal signal whose frequency either increases or decreases with time i.e. it is a linear frequency modulated signal.
- Spread spectrum is a technique in which a signal generated at a particular bandwidth is deliberately spread in the frequency domain, resulting in a signal with wider BW.
- CSS uses these chirp pulses to encode the information. It also uses its entire allocated BW to broadcast a signal, making it robust to channel noise.



Direct Sequence Spread Spectrum (DSSS)

- The frequency of final signal i.e. spread signal is much greater than the original signal.



- At the receiver, the signal is again multiplied with the same spreading code to get back the original signal.

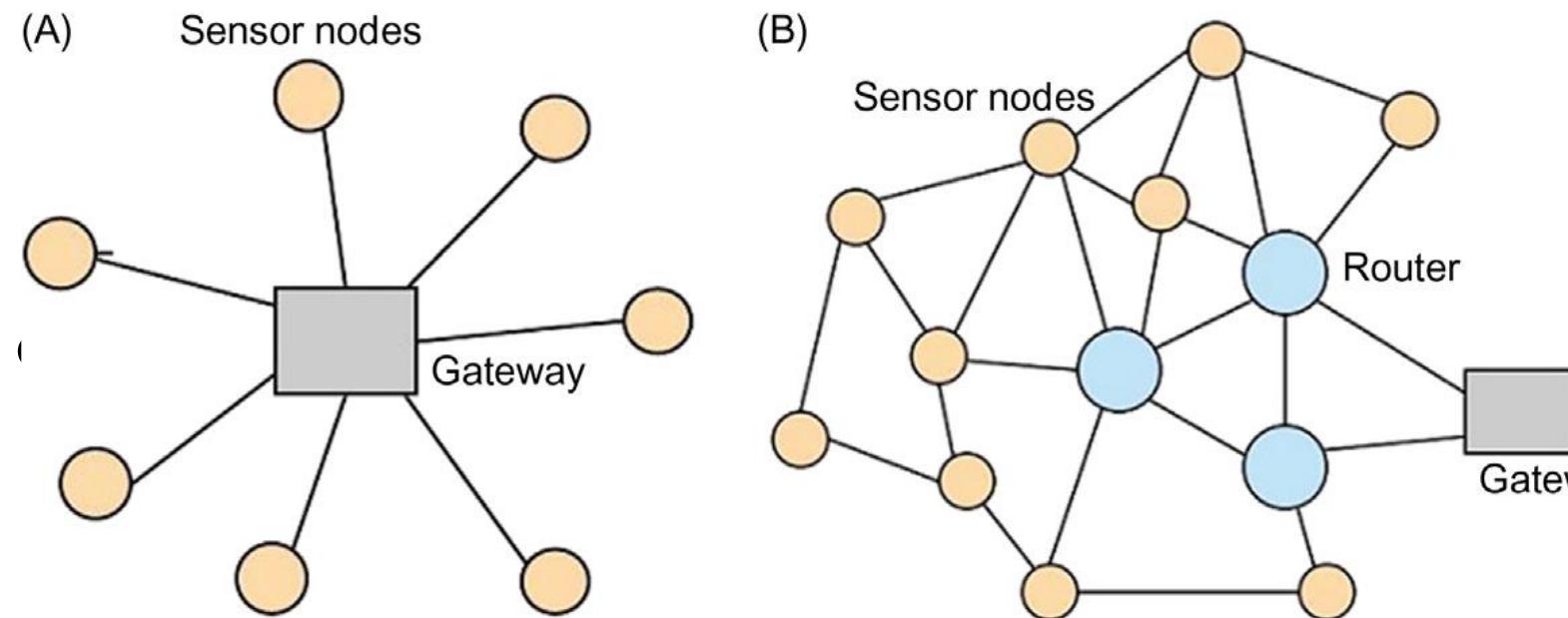
- Spreading in frequency domain increases the link budget and also the transmission range.

Direct Sequence Spread Spectrum (DSSS)

- The ratio of the spreading code's chip rate and the original signal's bit rate is called the processing gain.
- Higher processing gain allows the receiver to recover the original signal even if the channel SNR is very low. Thus, higher processing gain allows the transmitter to transmit at lower power and is beneficial for resource constrained IoT devices.

Network Topologies

- In IoT architecture, the sensor nodes can be connected in (A) Star topology or (B) Mesh topology as shown.



- In Star topology, all sensor nodes are connected to a gateway which is then connected to a cloud server. The individual sensor nodes are not connected to each other.
- In Mesh topology, the sensor nodes are connected to one another and also to the gateway. Few sensor nodes act as routers which collect the data from other sensor nodes and forward it to the gateway nodes.

Network Topologies

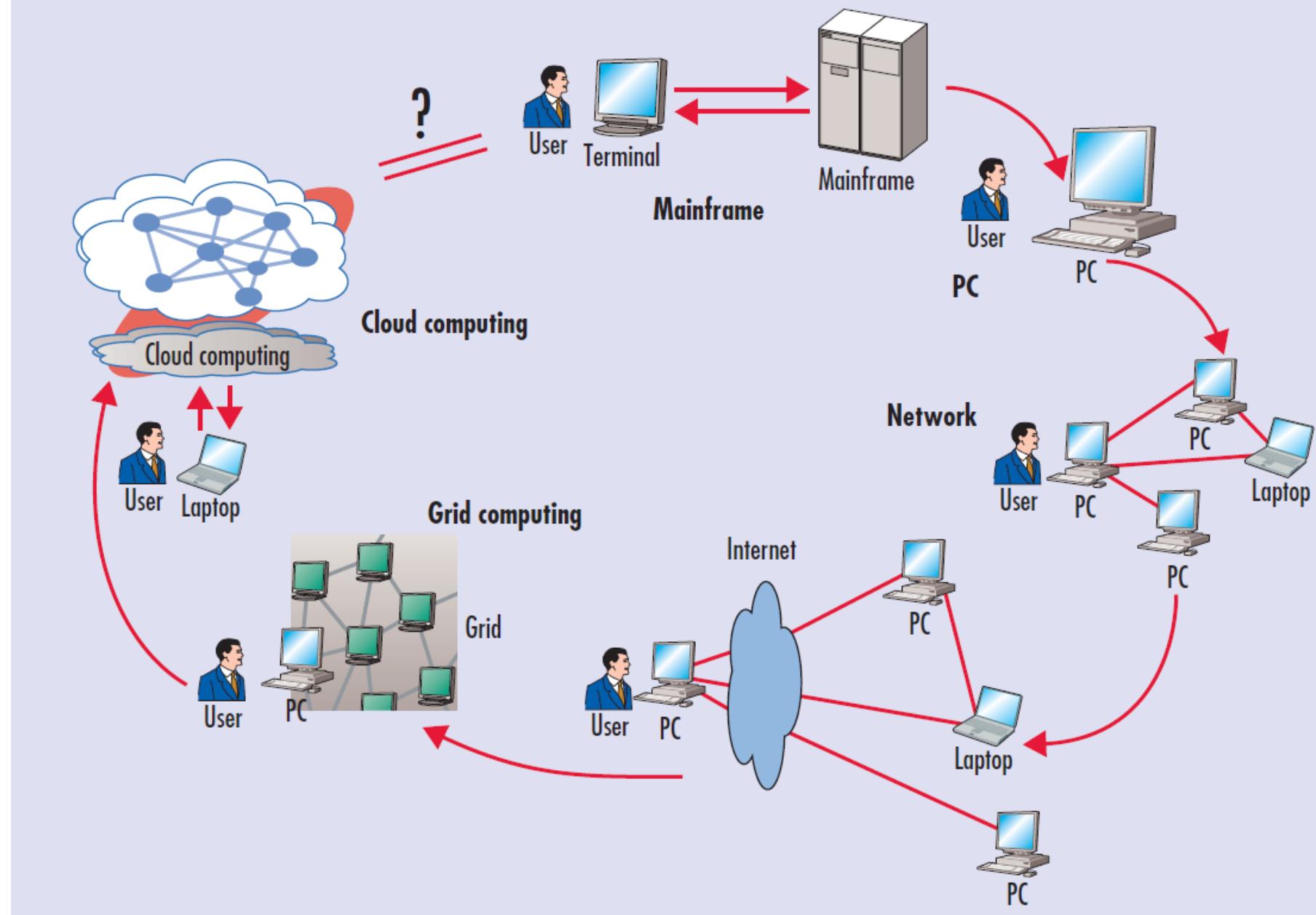
Star	Mesh
No connection between individual nodes	Individual nodes connected to each other
Single data path from sensor node to server, through gateway	Multiple data paths from sensor node to server
Simple topology	Complex topology
Routing algorithms, routing tables etc. are not needed	Routing algorithms, routing tables etc. are needed
Gateway has to be a resource heavy device as sensors are directly connected to it	Gateway need not be resource heavy as routers can share the load
Transmission is single-hop and faster	Transmission can be multi-hop and hence slower
A faulty or hacked sensor node can be easily identified and isolated	If a sensor node is hacked, other nodes connected to it are also at risk of being hacked
Readily scalable	Not easily scalable

Cloud Computing in IoT

Evolution of Computing

- Fig shows the evolution of computing paradigms over the years.

(From “cloud computing new wine or just a new bottle” by Jeffery Voas and Jia Zhang)



Evolution of Computing

- Mainframe computing – computer was a dummy piece (terminals) and computing was performed on a resource extensive mainframe device
- PC computing – personal computers became powerful enough to perform computing operations
- Network computing – Local Area Network where multiple PCs were connected to each other and resources were shared
- Internet computing – many LANs connected together forming a global network

Evolution of Computing

- Grid computing – similar to Distributed computing where computing tasks are distributed over different devices which are shared among the users
- Cloud computing – a centralized resource performing all the computing tasks is shared by many users. The resource (also known as datacenter) is extremely powerful and can perform complex computing tasks.

Sharing of Resources

- Generally when we talk about resources in a web or IoT network, we can categorize these as storage resources, computing resources and networking resources.
- These resources are typically hardware devices which perform the functions of data storage (database), computing (servers) and networking/routing (routers).
- In order to utilize these resources efficiently, they must be shared among different users so that scaling or dynamic allocation is possible.
- In order to enable sharing of resources, virtualization is necessary because if the resources are rigid, they cannot be shared among different users.

Sharing of Resources

- Thus, for sharing and virtualization, the hardware devices need to be controlled by some software applications.
- Based on which resources are being virtualized (and shared), we come across different technologies
 1. **Cloud Computing** – computing and storage resources are virtualized and shared
 2. **Software Defined Networking (SDN)** – networking resources are virtualized

Cloud Computing

- Cloud computing is shared pool of storage/computing resources that can be accessed on demand and offered dynamically to users
- Storage, computing, applications – run in cloud servers
- PC – low cost, low power, low resource device.
- Cloud service providers setup and maintain their own data centers. They create different virtual environments of this which can be used and accessed using APIs.

Virtual Machines and Containers

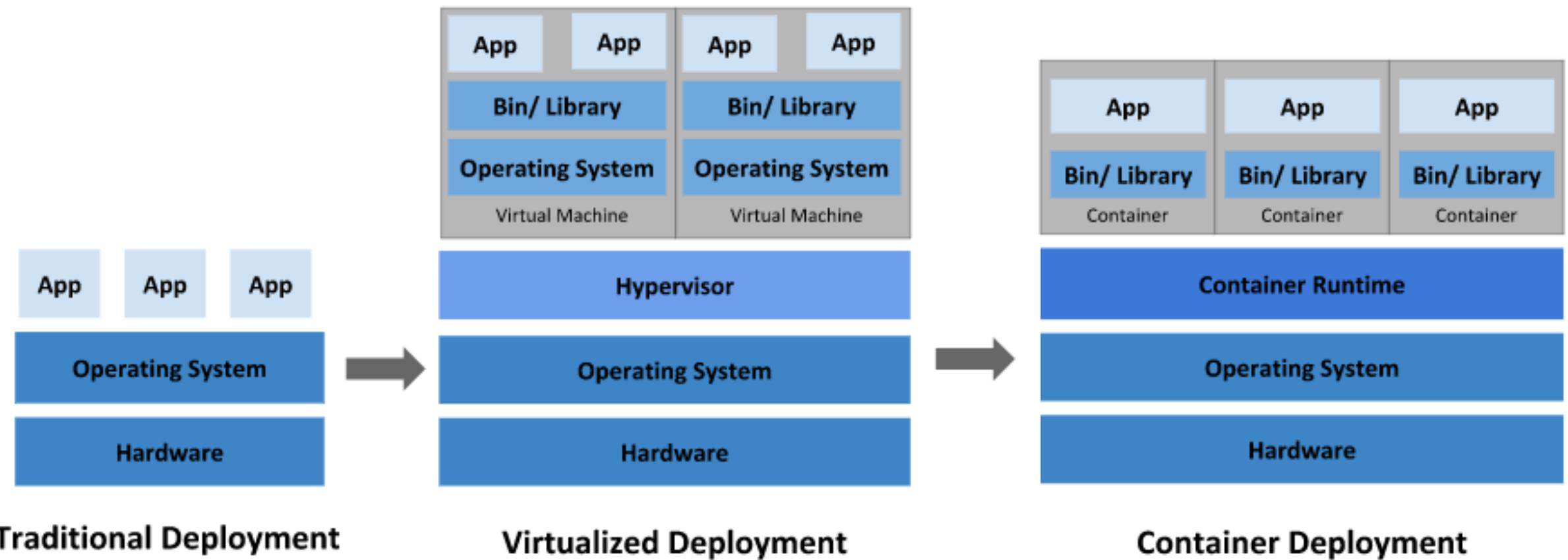
- Traditional cloud computing relies on the technology of virtualization or virtual machines.
- Cloud service providers setup and maintain their own data centers and create different virtual environments that use the underlying hardware resources.
- Users can programmatically launch a virtual machine in a remote server, can SSH into it and use it to install any software or test a program using a API.
- Now, VMs are getting replaced by containers.

Virtualization

- Creating a virtual version of a physical resource (server, storage, network, processing power) using a software.
- Multiple OS and apps can run on the same machine using its underlying hardware.
- Virtualization allows sharing of a single physical resource with multiple clients at the same time, thus making effective utilization of it.
- Assigns a logical name to physical resource and provides a pointer to that on demand.

Virtualization

- Example of virtualization



Virtualization

- Traditionally, one PC had a single OS which can run multiple apps. Suppose memory of this PC is 128 GB – out of which 10 GB would be taken by OS and rest 118 GB would be available for the apps.
- Now, if we make this PC dual boot – i.e. install 2 OS. We can run diverse apps on these OS but the resources available for each app will reduce. 2 OS will take up 20 GB and only 108 GB will be available for apps.
- Thus, installing more OS on the PC does not improve the performance but simply allows the user to install diverse apps on it.
- Also, it may happen that 1 app is taking up most of the memory and because of which rest of the apps are underperforming.

Virtualization

- Now suppose we install a Virtual Machine app (VMWare or Virtual Box) on the PC.
- Now, inside this VM app we can install separate OS, inside which we can install other apps.
- In this way, we have created another machine (PC) inside our host machine (PC).
- In this arrangement, suppose installing VM on the host PC requires 6 GB of space. If we install another OS inside the VM, it will NOT consume 10 GB of space as in the traditional case.
- This is because, *original VM is just an app for the host PC*. Whatever is installed inside the VM does not directly affect the host PC.

Virtualization

- We can install multiple VMs on a single host PC and run multiple OS inside these VMs.
- The VM app is actually a hypervisor – a shielding layer which shields the host PC from all the complexities inside the VM.
- Because of hypervisor, the host PC treats the VM as just another app even though in reality VM is another PC in itself.
- *Hypervisor is actually fooling the host PC into believing that VM is just an app.*
- Virtualization – fooling the host PC!!!

Virtualization

- One example of virtualization is the movie Rocket Singh – Salesman of the Year.
- In that movie, an employee of a company starts his own company using the resources of the parent company. So, it's like a company running inside another company.
- The parent company treats him as a normal employee but he has his own employees working for him.
- So, Rocket Singh in a legal way is virtualization.

Virtualization

- Another example – suppose you have INR 10 lakh with you and you want to invest
- Case 1
 - You buy shares of 10 different companies (1 lakh each)
 - You need to monitor these companies by your own, keep track of their progress, calculate the profit/loss etc.
 - This will consume lot of your mental resources
- Case 2
 - You hire 10 brokers and give them 1 lakh each
 - These 10 brokers will make their own portfolio – each will have its own set of companies
 - They will manage their portfolio independently of one another
 - You are only concerned with the net profit
 - Your mental resources are efficiently utilized by large number of companies now

Virtualization

- Virtualization involves use of the following terms

1. Virtual Machine

1. A **software-defined computer** that runs on a physical computer with a separate operating system and computing resources.
2. Multiple VMs can run on a single physical machine.
3. Each VM can be used by a different client

2. Host Machine

1. The physical machine on which multiple VMs are running is called host machine

3. Hypervisor

1. Software component that manages multiple VMs on a single host machine.
2. It ensures that each VM gets the allocated resources and does not interfere with other VMs.

Virtualization Types

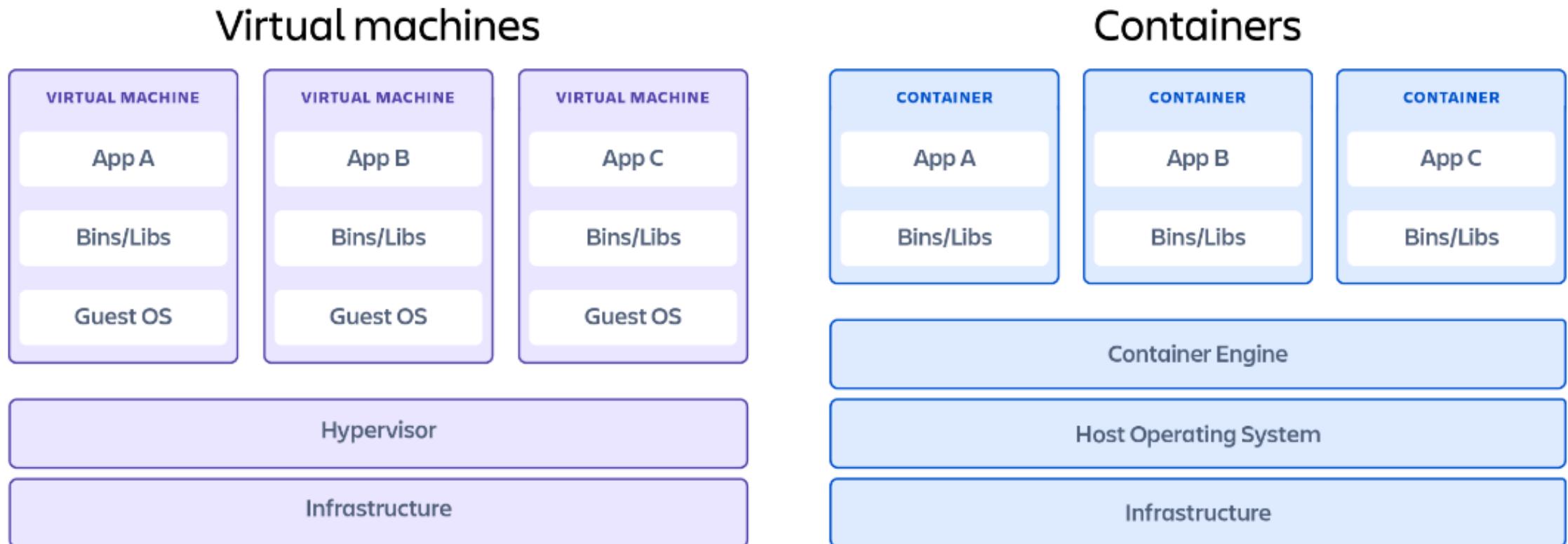
- Server Virtualization
 - 1. Partitions a physical server into multiple virtual servers
- Storage Virtualization
 - 1. Combining all the physical data storage and creating large unit of virtual data storage which can be managed by using a software.
 - 2. Storage activities such as archiving, backup and recovery can be streamlined.
- Network Virtualization
 - 1. Different components of a computer network like switches, routers etc. can be virtualized and controlled from a central location.

Virtualization Types

- Desktop Virtualization
- Application Virtualization
- Data Virtualization

Virtual Machines vs Containers

- Each VM has its own OS (known as Guest OS) which runs on the host infra and is managed by the hypervisor. This makes VM very heavyweight.
- Containers do not have separate OS – making them lightweight.



Virtual Machines and Containers

- Containerization is a way to deploy application code to run on any physical or virtual environment without changes.
- Developers bundle the application code with related libraries, binary files, configuration files and other dependencies in a single software package called container.
- Container can run independently on any platform.
- Containerization is equivalent to application virtualization.

Containers and Containerization

- Containerization



Containerization

- Containerization is a software deployment process where the apps' code, its binaries and libraries are grouped together in a single package.
- This package can now run on any infrastructure and any OS i.e. the same container of an application can work on a Windows or Linux machine.
- Software developers use containerization to deploy applications on multiple environments without rewriting the application code.
- Containers are highly scalable and lightweight. A virtual machine can launch a containerized application faster as it does not need to boot the OS.

(<https://aws.amazon.com/what-is/containerization/#:~:text=Containerization%20is%20a%20software%20deployment,matched%20your%20machine's%20operating%20system.>)

Containerization

- Containers are highly fault tolerant. A single faulty container doesn't affect the other containers, thus increasing the resiliency and availability of the application.
- Containers run in isolated environment. The software developer can change or modify the application code without interfering with the operating system, hardware and other application services.

(<https://aws.amazon.com/what-is/containerization/#:~:text=Containerization%20is%20a%20software%20deployment,matched%20your%20machine's%20operating%20system.>)

Containerization

- Best example of containerized apps – ready to eat food packets.
- The food packet contains all the ingredients required to make that dish. Packets of different dish contains different ingredients i.e. ready to eat pack of poha will have different contents than that of a upma.
- The user just needs to empty the packet in a can of boiling water i.e. all the containers share the same OS (hot water in this case).
- Different containers (food packets) can be stored together in a single refrigerator.

Containerization

- The technology of containerization consists of the following sub-technologies:
 1. Builder – a tool or a series of tools used to build a container
 2. Engine – an application used to run a container
 3. Orchestration – a technology used to manage the containers running together (e.g. Kubernetes)

Docker

- A platform which allows to build, test and deploy applications quickly.
- Docker packages software into containers.
- It's an example of PaaS which uses OS level virtualization to deliver software in packages called containers.
- Docker performs OS virtualization – creates multiple instances of the host OS and lets containers run on them.

Docker Terms

1. Docker Image
 1. Contains executable application source code as well as tools libraries and dependencies
 2. Multiple docker images can be created from the same base image and they will share the commonalities of the stack
2. Docker Containers
 1. Live, running instances of docker images
 2. Images are read-only files while containers are life, ephemeral, executable content
 3. Users can interact with containers using docker commands
3. Docker File

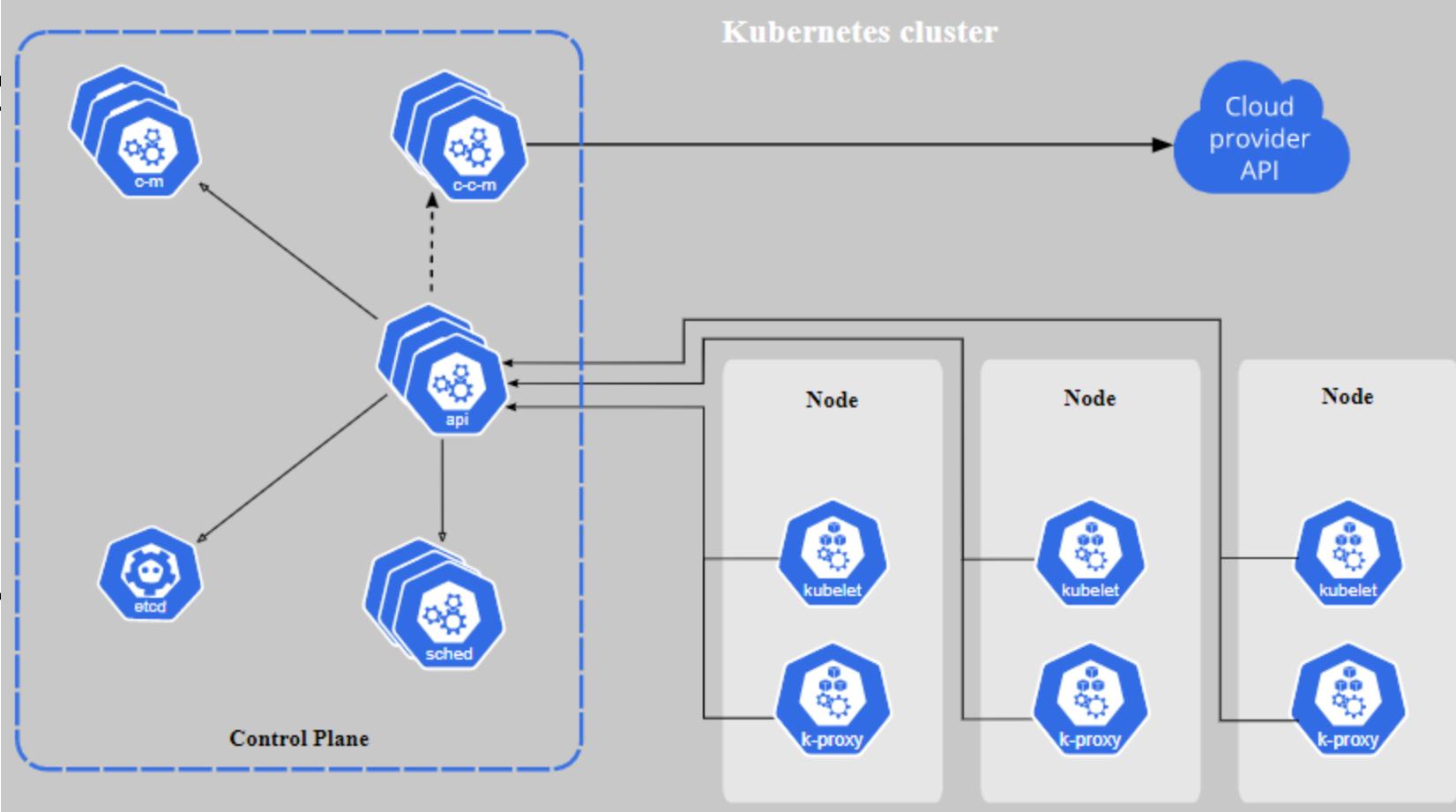
Kubernetes (K8S)

- Container management or orchestration engine. Performs the following tasks:
- *Load Balancing* – manages the traffic of data handled by each container. If any container is overloaded, then the traffic is diverted to other containers.
- *Automated containerization* – user can provide K8S with a cluster of nodes that it can use to run containerized tasks. User can tell how much CPU and memory each container needs and K8S will fit the containers in the overall memory efficiently.
- *Self-healing* – K8S can restart the containers that fail, kill or replace the containers that do not pass the user-defined health check and doesn't advertise these containers to the client.

<https://kubernetes.io/docs/concepts/overview/>

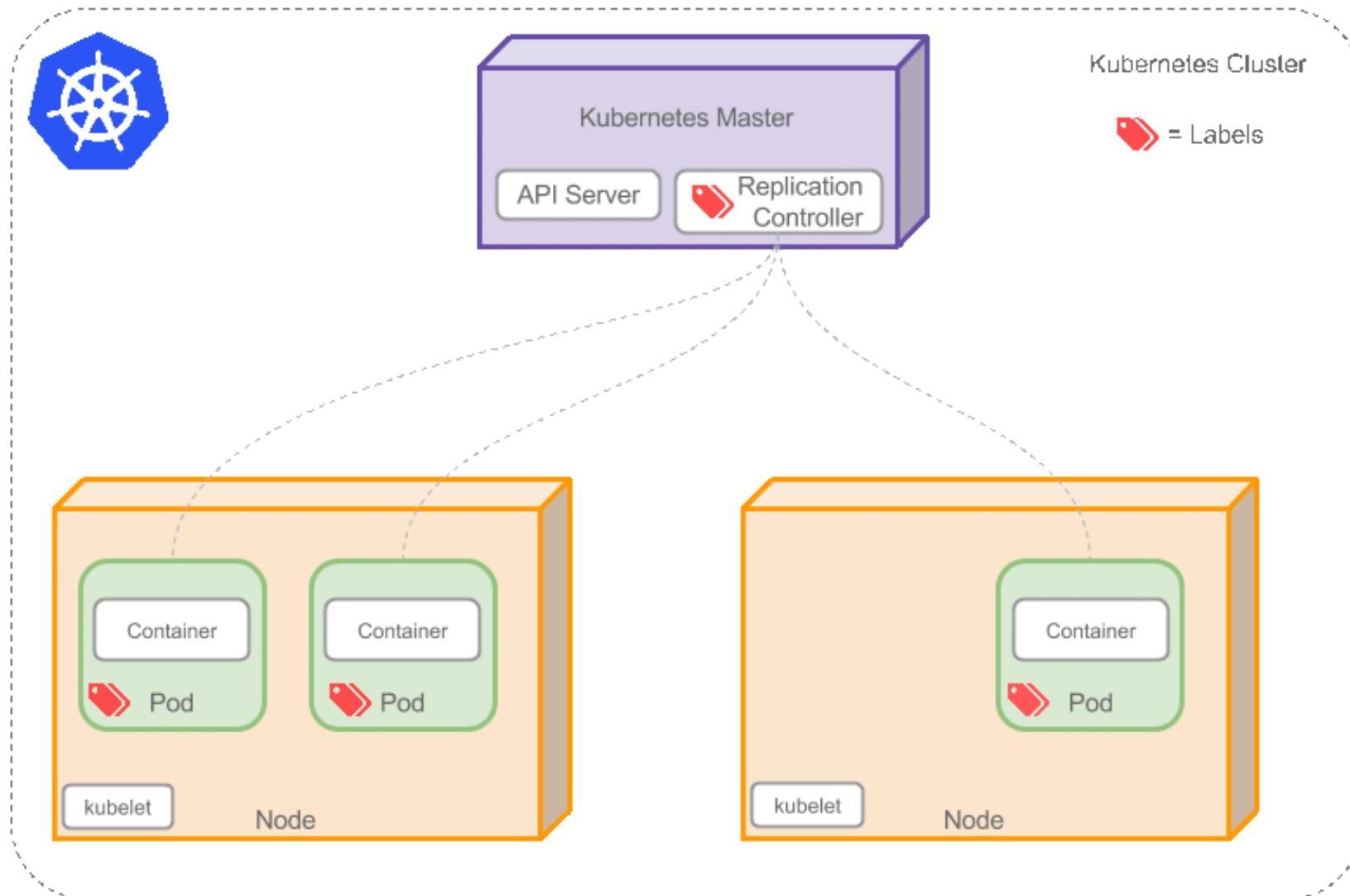
Kubernetes Architecture

- When we deploy k8s, we Get a cluster.
- A cluster consists of several K8s nodes or worker machine.
- A node hosts a set of pods.
- A pod represents a set of running containers.
- Thus a node consists of several containers running together and cluster consists of several nodes.



Kubernetes Architecture

- W



Kubernetes Architecture

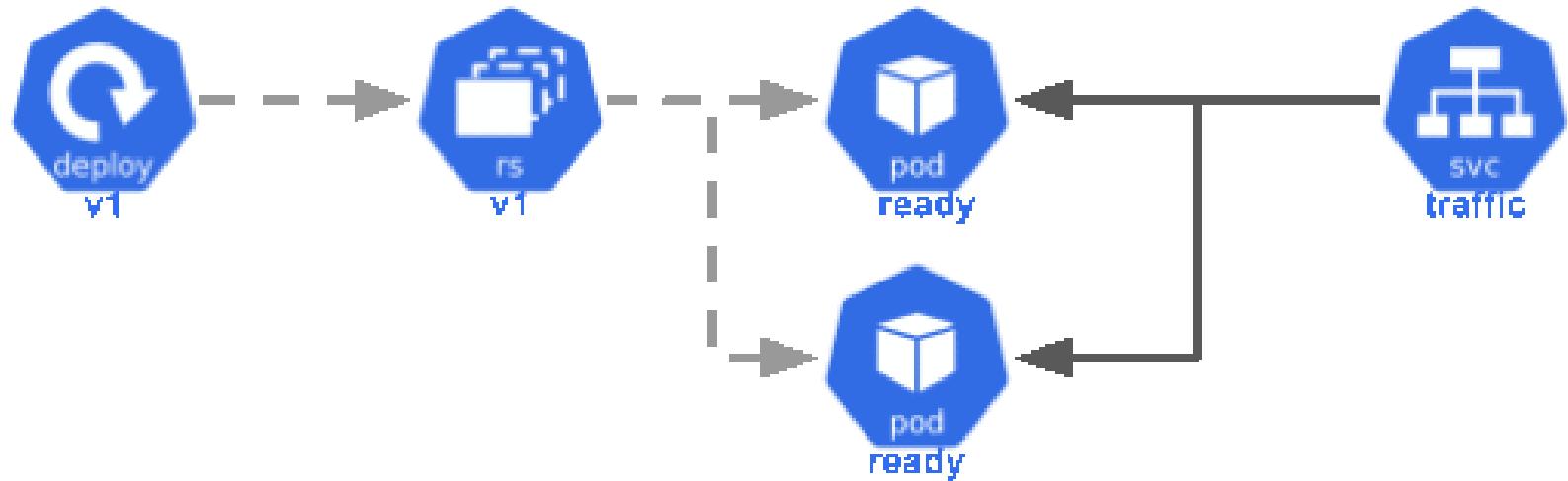
- The node consists of the following components:
 1. Container Runtime Engine – manages the life-cycle of a container. Ex: Docker
 2. Kubelet – an agent that communicates with the control plane to ensure that the containers are running. Whenever control plane wants a specific action to be taken on a container, the kubelet receives the pod specs from the API server.
 3. Kube Proxy – a network proxy that facilitates networking services for UDP and TCP packets.

Kubernetes Architecture

- Control plane – manages the whole cluster.
- Any machine can be setup to work as the control plane. Control plane consists of the following components:
 1. *API Server* – acts as a gateway between the cluster and client. Clients use the API server as a tunnel to the pods and containers and authenticate via API server.
 2. *Scheduler* – stores the resource usage data of each node and performs dynamic resource allocation and management.
 3. *Controller Manager* (c-m) or *Cloud Controller Manager* (c-c-m) –
 4. *etcd* – database which stores configuration data and info about the cluster state

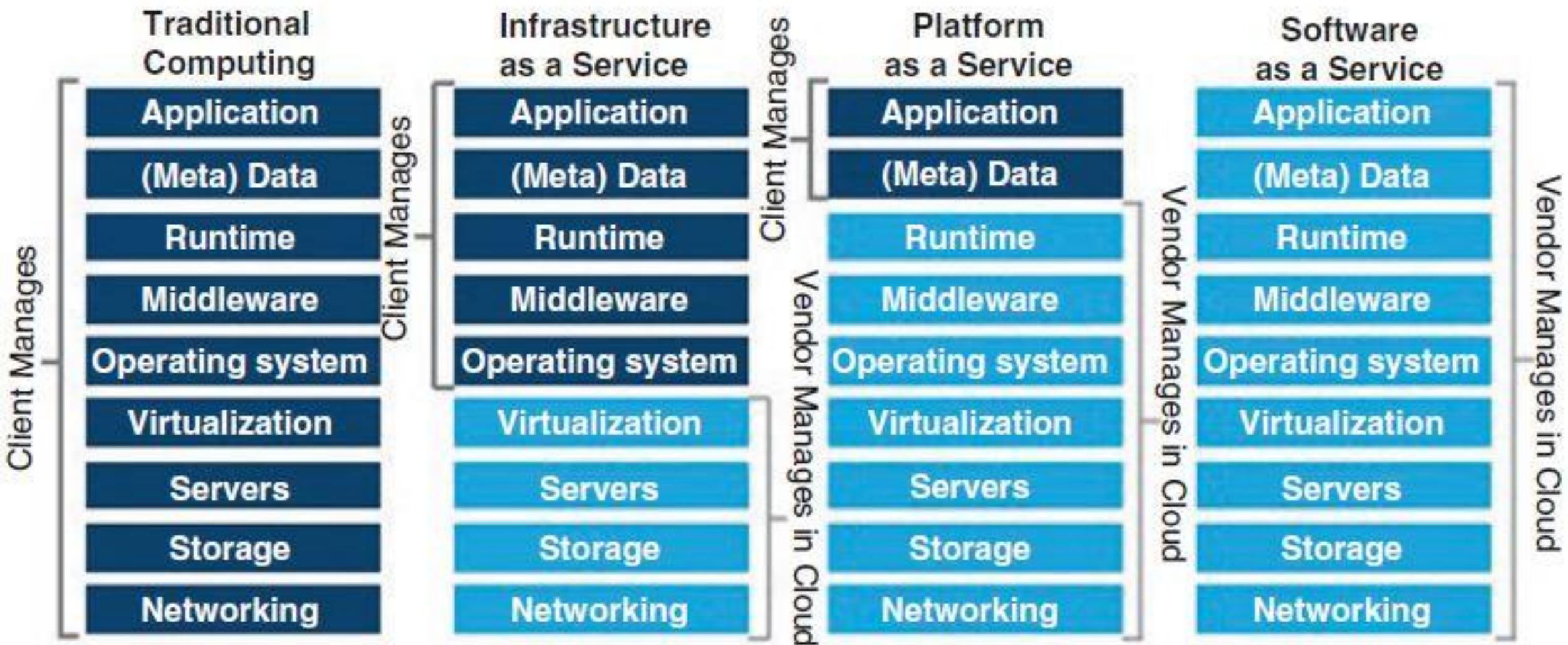
Kubernetes Architecture

- Control



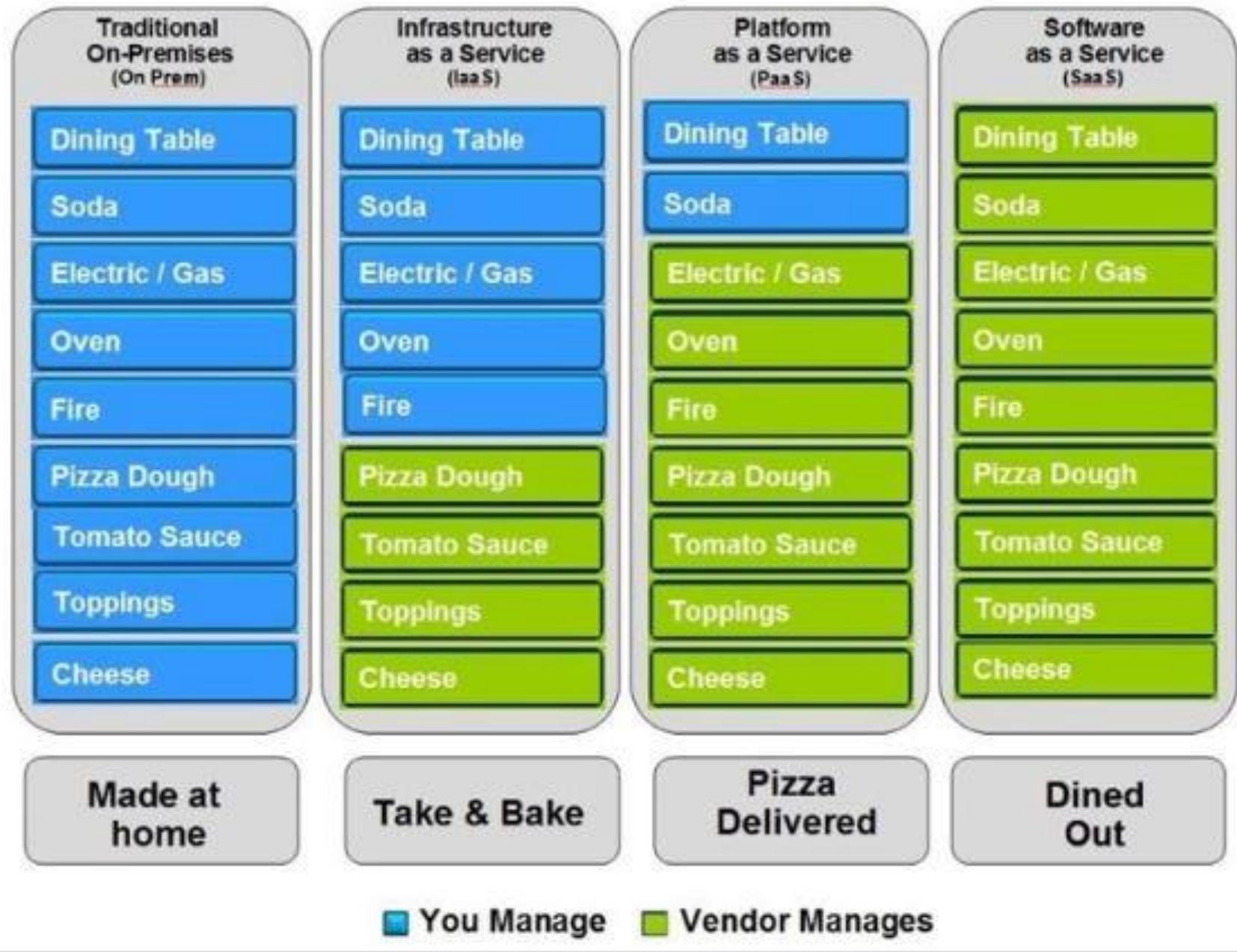
Cloud Computing Models

- Cloud computing services are divided into 3 categories
 1. Infrastructure as a Service (IaaS) – hardware, network and servers in cloud
 2. Platform as a Service (PaaS) – IaaS + OS and middleware in cloud
 3. Software as a Service (SaaS) – everything in cloud



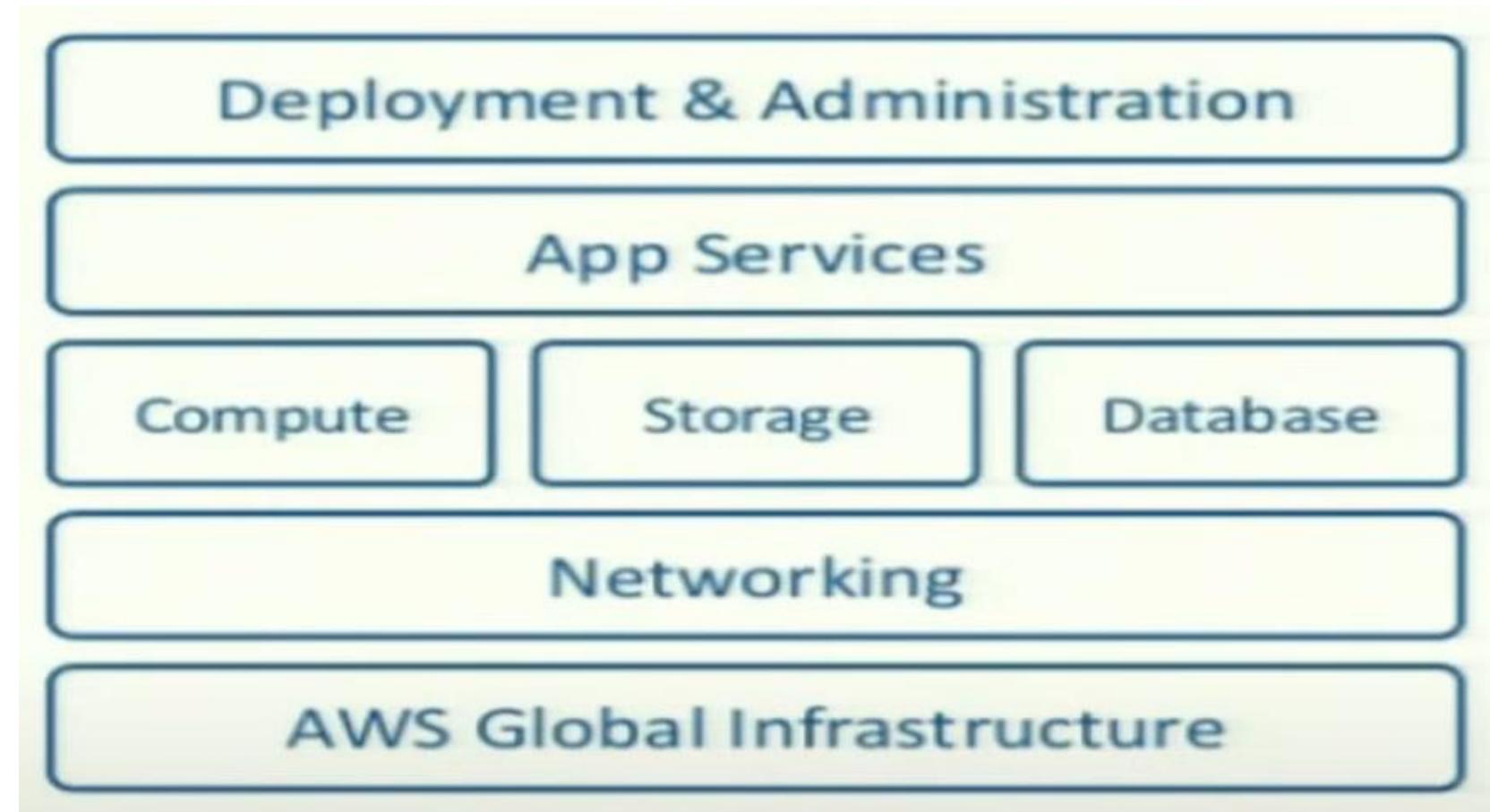
Infrastructure as a Service	Platform as a Service	Software as a Service
Vendor provides resources like virtual machines and virtual storage.	Vendor provides access to run time environment and development tools	Vendor provides access to softwares hosted at its facility
Used by network architects	Used by web application developers	Used by general customers
Requires considerable technical knowledge	Requires knowledge of web development	No requirement of technical knowledge

Pizza as a Service



Typical Public Cloud Architecture

- Any public cloud service can be divided into the 5 layered architecture, as shown in the fig.
- These layers describe the components involved in rendering of cloud based services.



AWS Regions

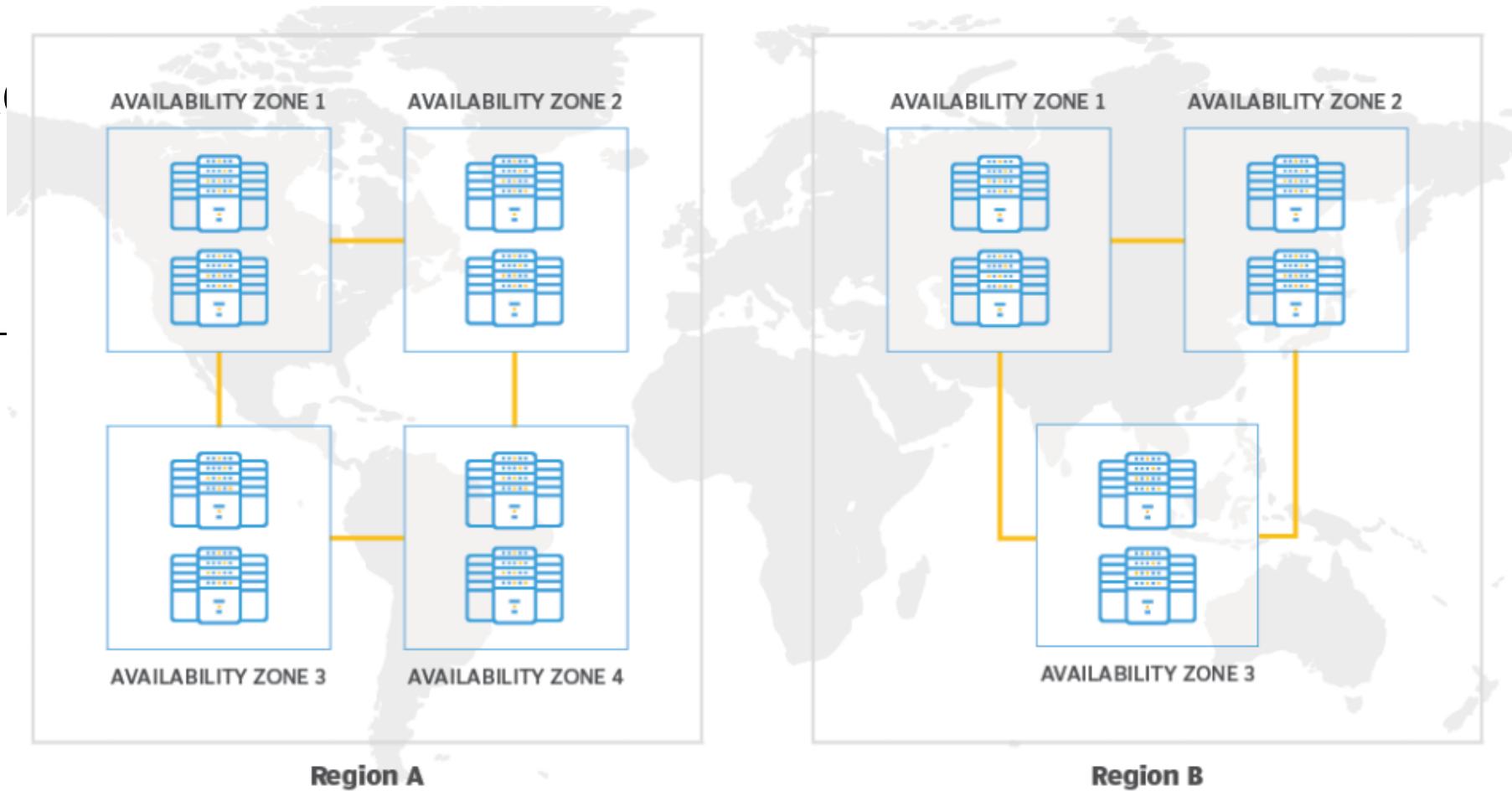
Global Infrastructure

- Global infrastructure refers to the network of huge data-centers of the vendors, located around the world.
- An example of data centers of AWS is shown.
- Similarly, other cloud providers like Google, Microsoft etc have their network of data centers spread across the globe.



Global Infrastructure

- Global infrastructure of AWS is divided into regions and availability zones.
- A region is an independent collection of AWS resources in a particular geographic area. While using the AWS resources, the first step user needs to perform is to select this region (e.g. Asia, Western Europe, North America etc.)
- Each region has multiple isolated locations known as Availability Zones (AZ).

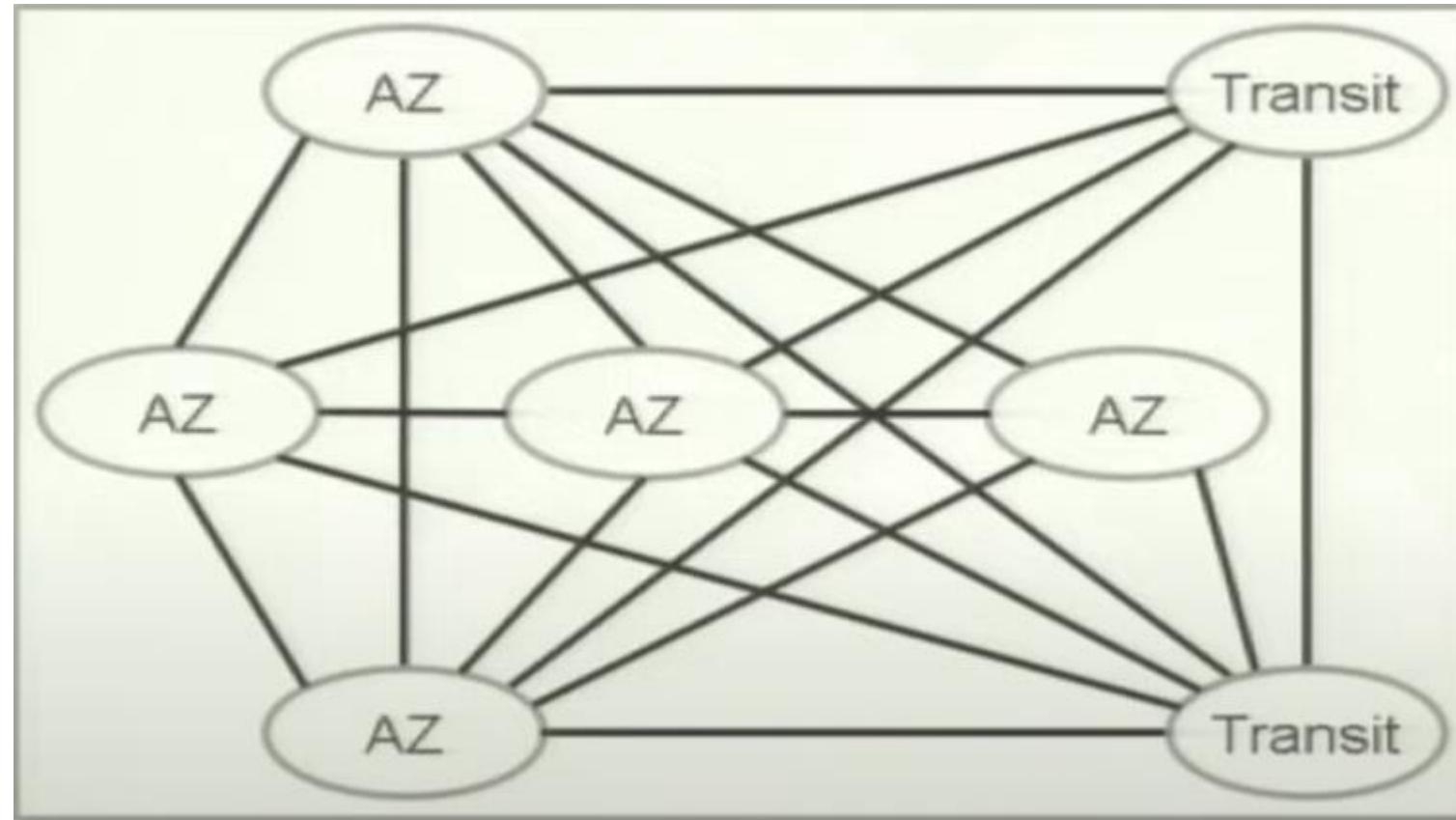


Global Infrastructure

Region name	Region code	Number of AZs
US East (Northern Virginia)	us-east-1	6
US East (Ohio)	us-east-2	3
US West (Oregon)	us-west-2	4
US West (Northern California)	us-west-1	3
AWS GovCloud (US-East)	us-gov-east-1	3
AWS GovCloud (US-West)	us-gov-west-1	3
Canada (Central)	ca-central-1	3
South America (São Paulo)	sa-east-1	3

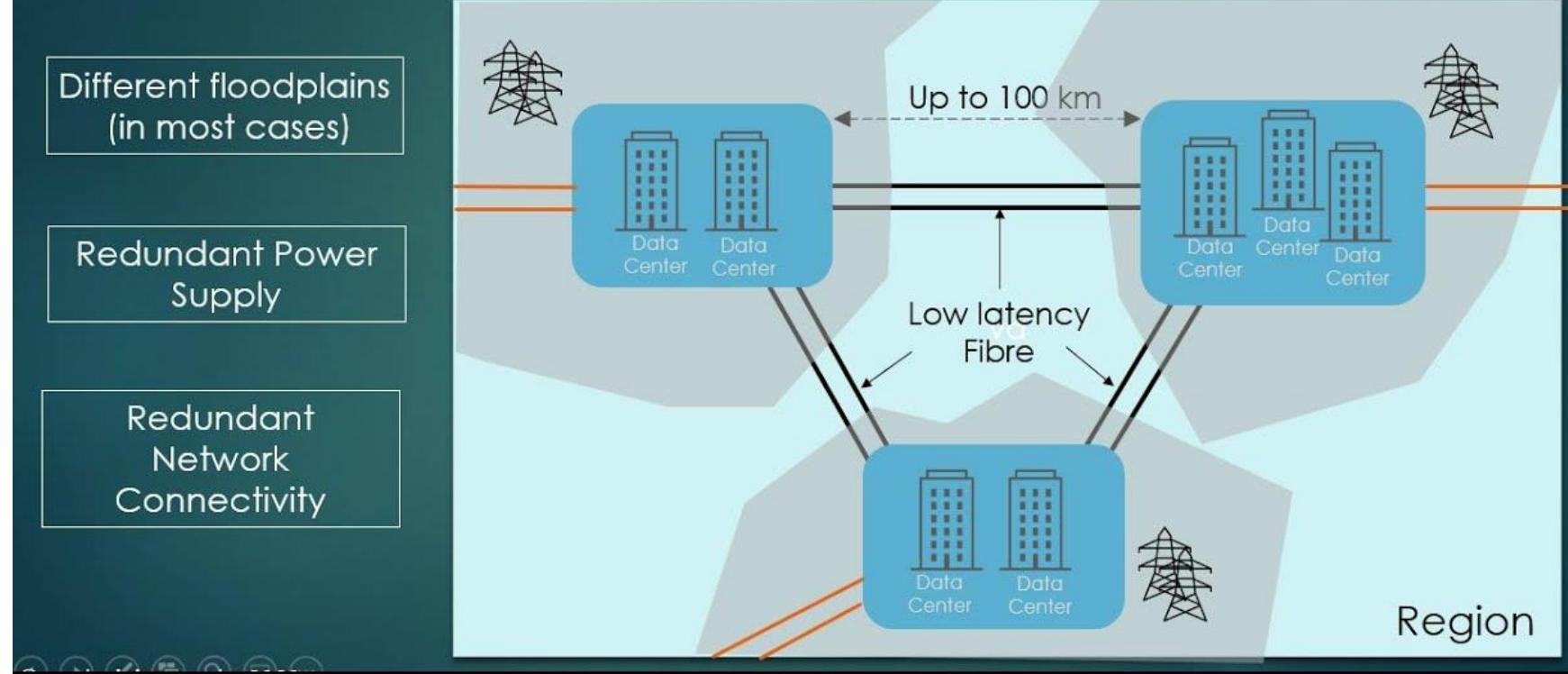
Global Infrastructure

- Availability zones also work independently and provide failure resistance.
- As shown, different AZs in a region are connected in a mesh topology through high-bandwidth links (Fiber Optic DWDM).
- Services running on one AZ is copied to 1 or more AZs so that if one AZ fails then services will keep running on another AZ.
- The AZ which acts as a gateway to another region is called transit node.



Global Infrastructure

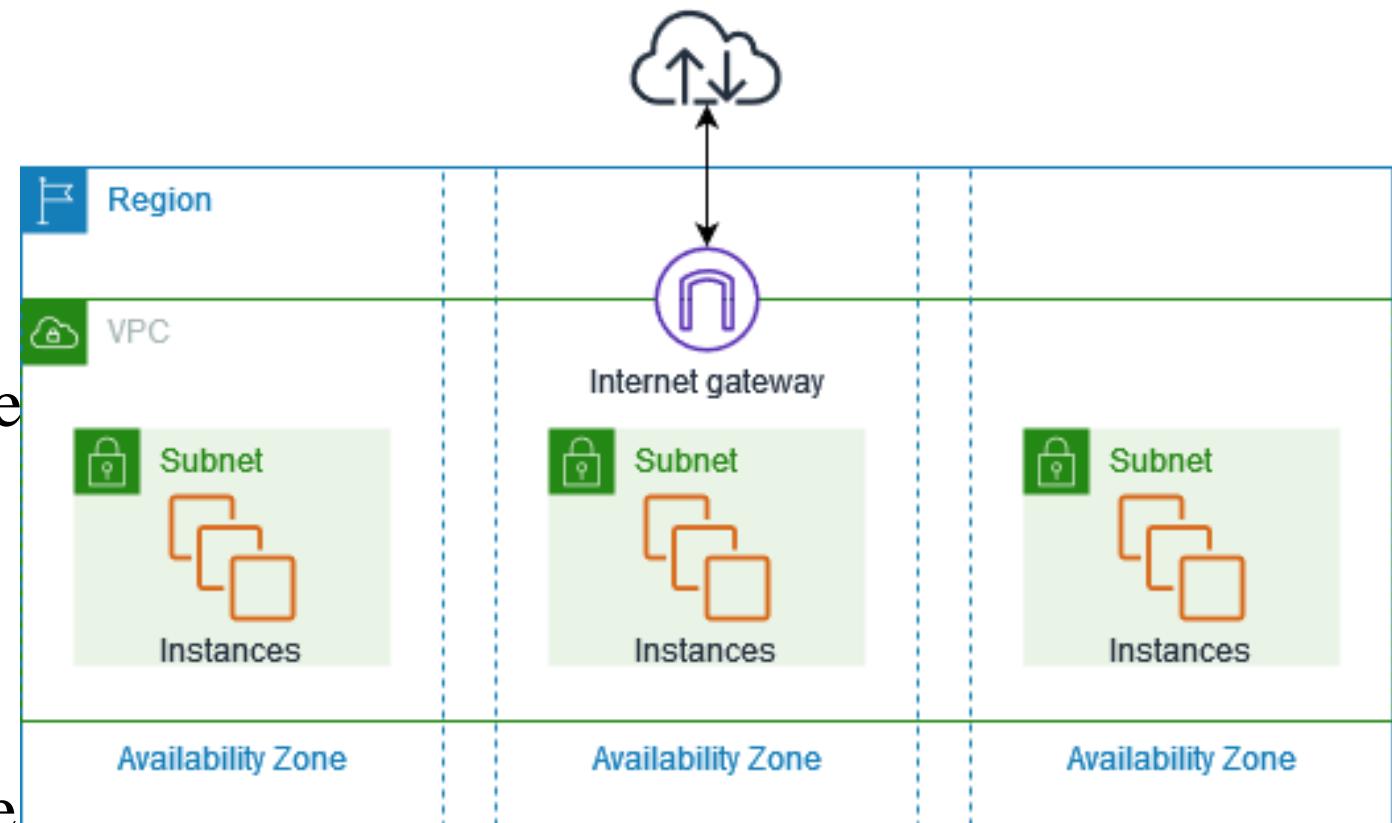
1 Region = Multiple AZs (Min 3)
1 AZ = Cluster of Data centres



- Each AZ contains multiple data centers – typically 4 to 6.
- Thus, region is a collection of AZ and AZ is a collection of data centers.

Global Infrastructure

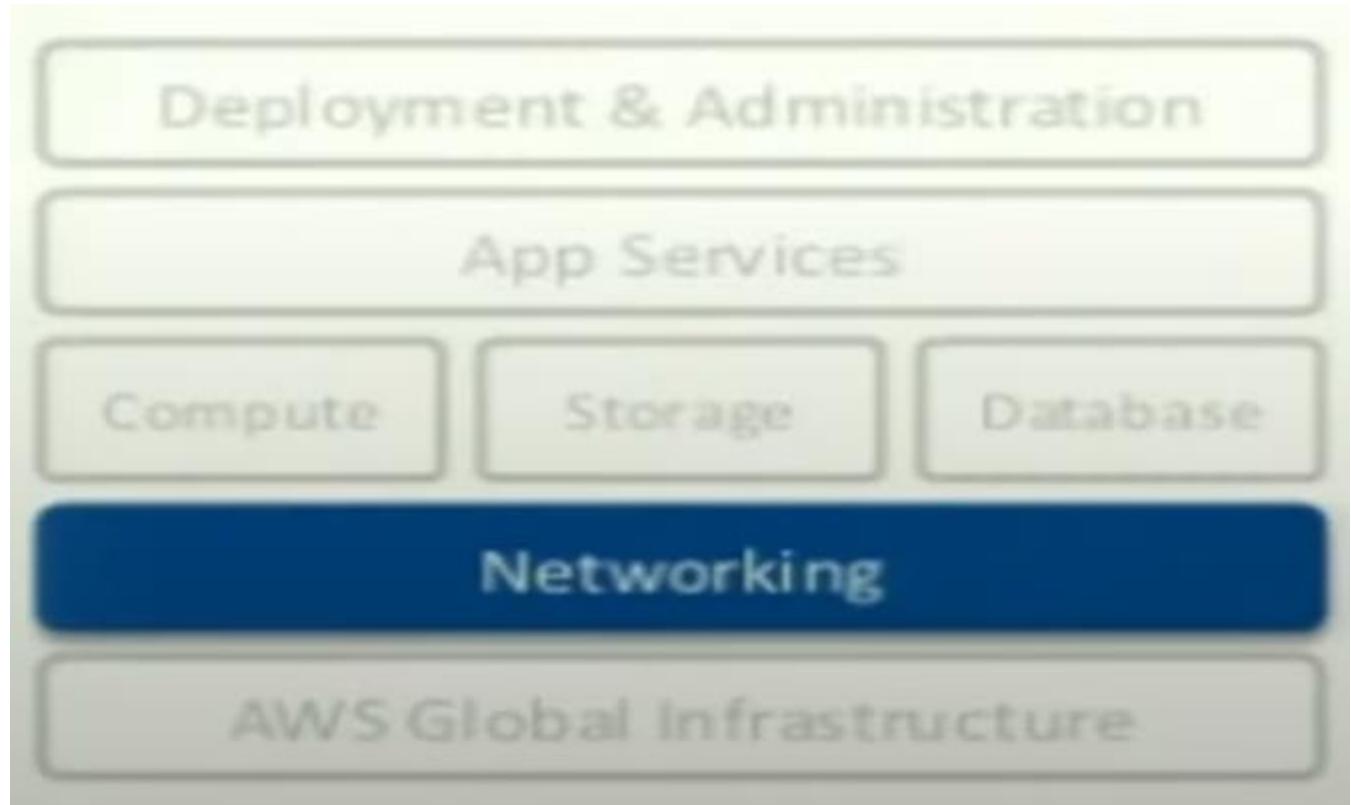
- Whenever a user creates an account In AWS and requests a service (maybe IaaS, PaaS or SaaS) – an instance is Created in the nearest AZ.



- The user is assigned a virtual private Cloud (VPC) for this instance.
- VPC is just like a virtual machine running on the data-center inside the AZ, which will cater to all the services requested by a particular user.

Networking

- Networking services between client – Datacenter and between different Datacenters is provided using:
 1. Direct Connection
 2. VPN
 3. Virtual Private Cloud
 4. Domain Name System (Route 53)



Compute

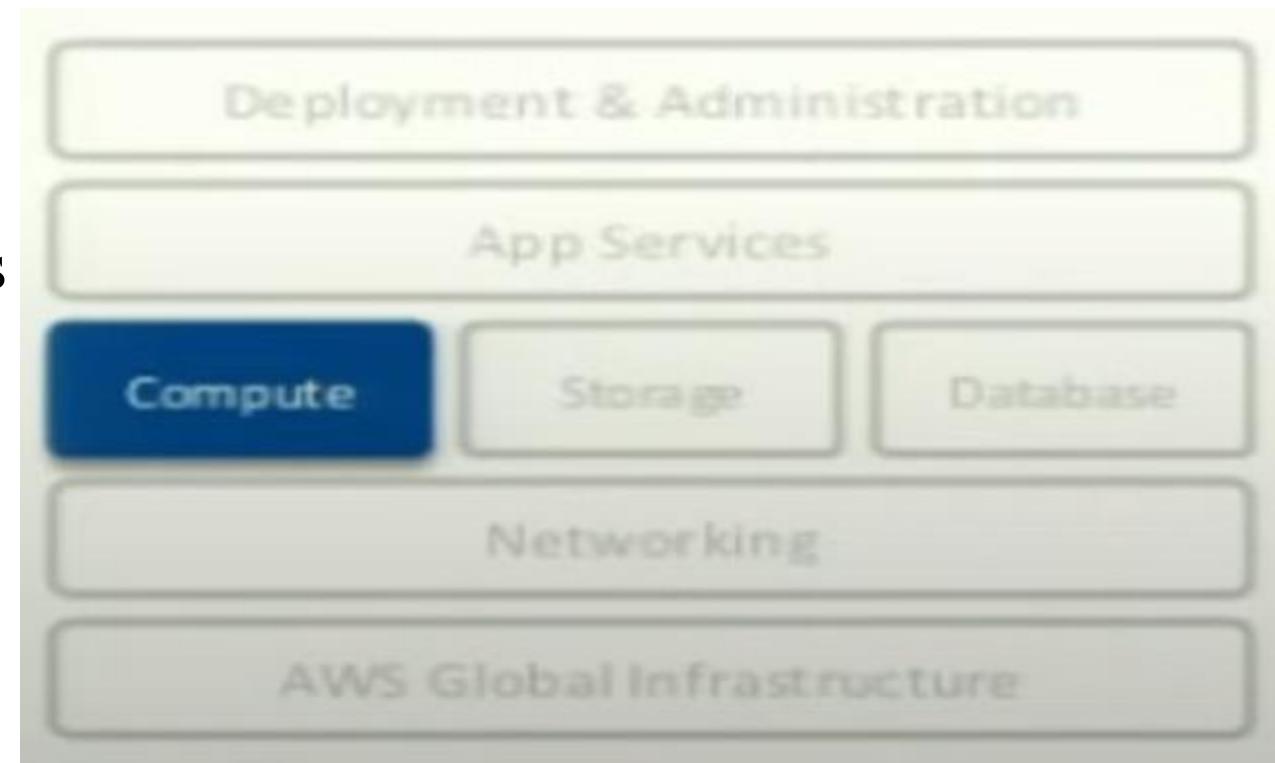
- AWS provides range of compute services

1. EC2 (Elastic Compute Cloud)

1. IaaS facility
2. Range of CPU, memory and local disk Options.

2. Auto Scaling

1. Automatic resizing of compute clusters based on demand
2. Helps in maintaining application availability – used by app developers



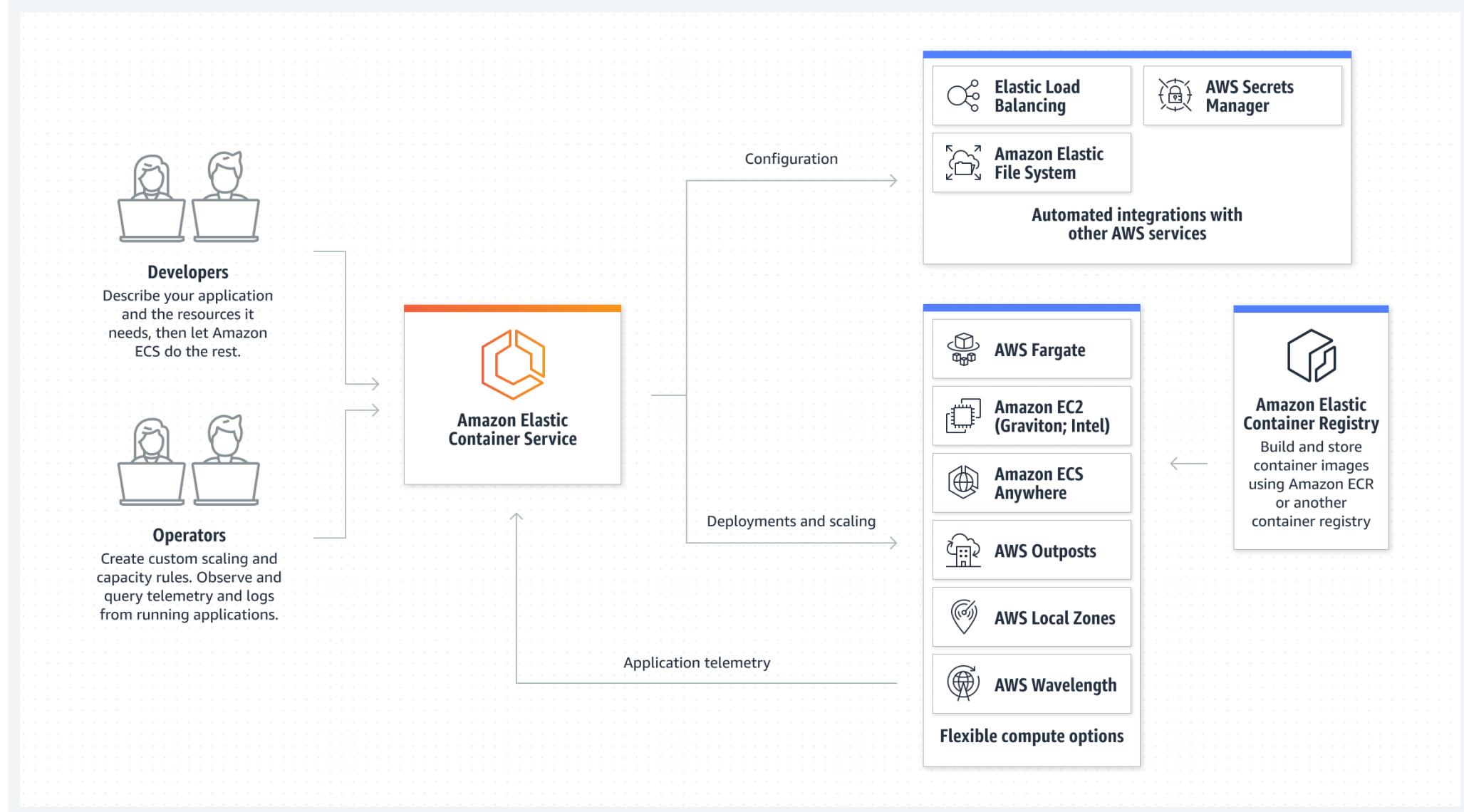
[Link](#)

Compute – Auto Scaling Service



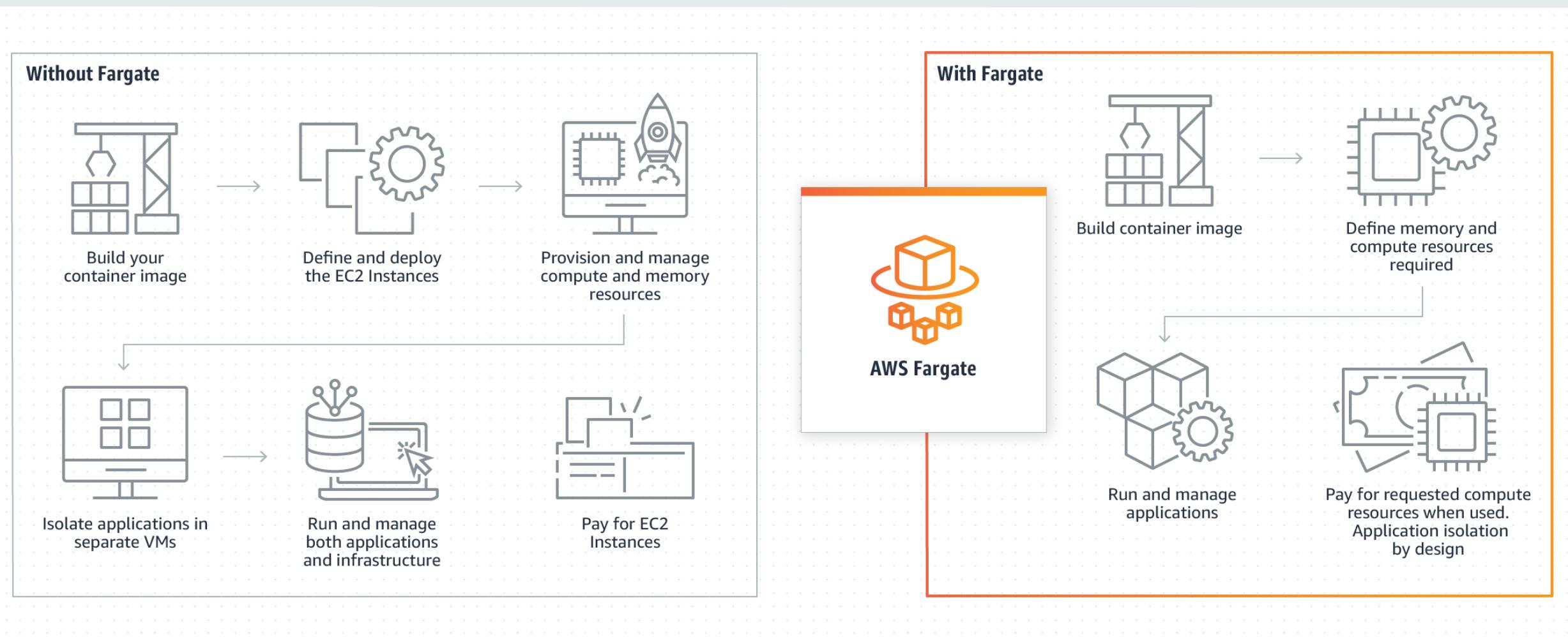
Compute – Elastic Container Service (ECS)

- ECS provides deployment, management, scaling of containerized applications.



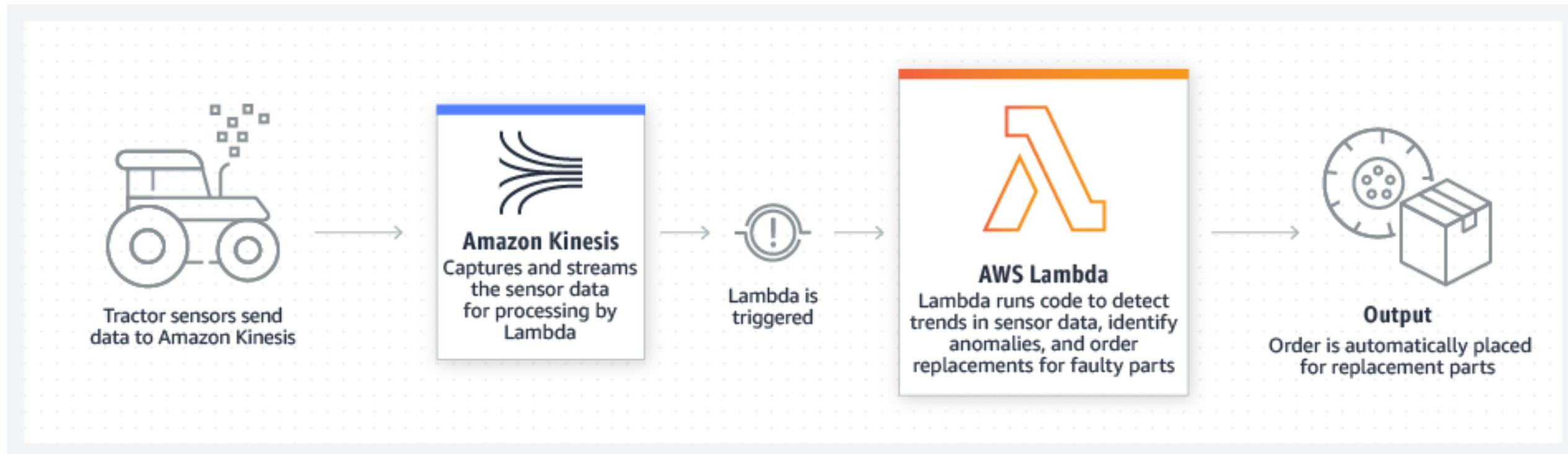
Compute – AWS Fargate

- AWS Fargate is a serverless, pay-as-you-go compute engine.



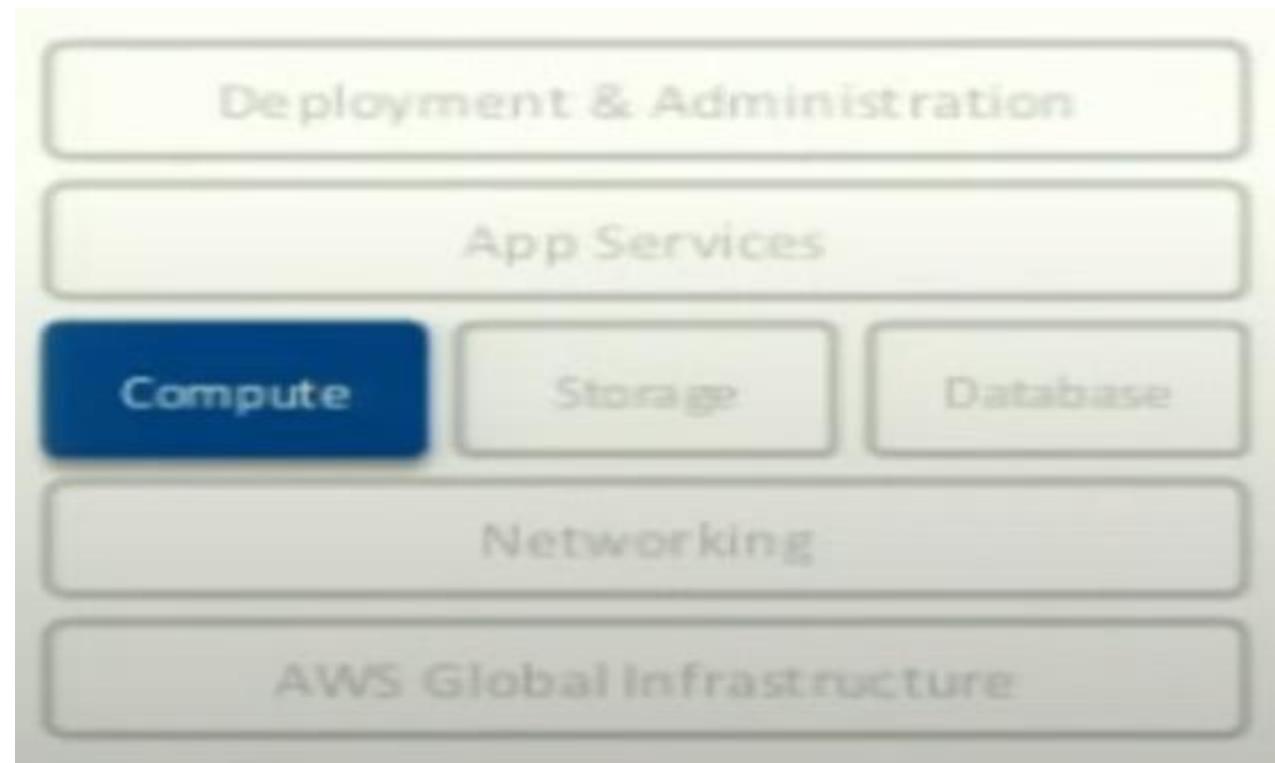
Compute – AWS Lambda

- AWS Lambda is a serverless, *event-driven compute service* that allows running microcodes for any application without provisioning or managing servers.



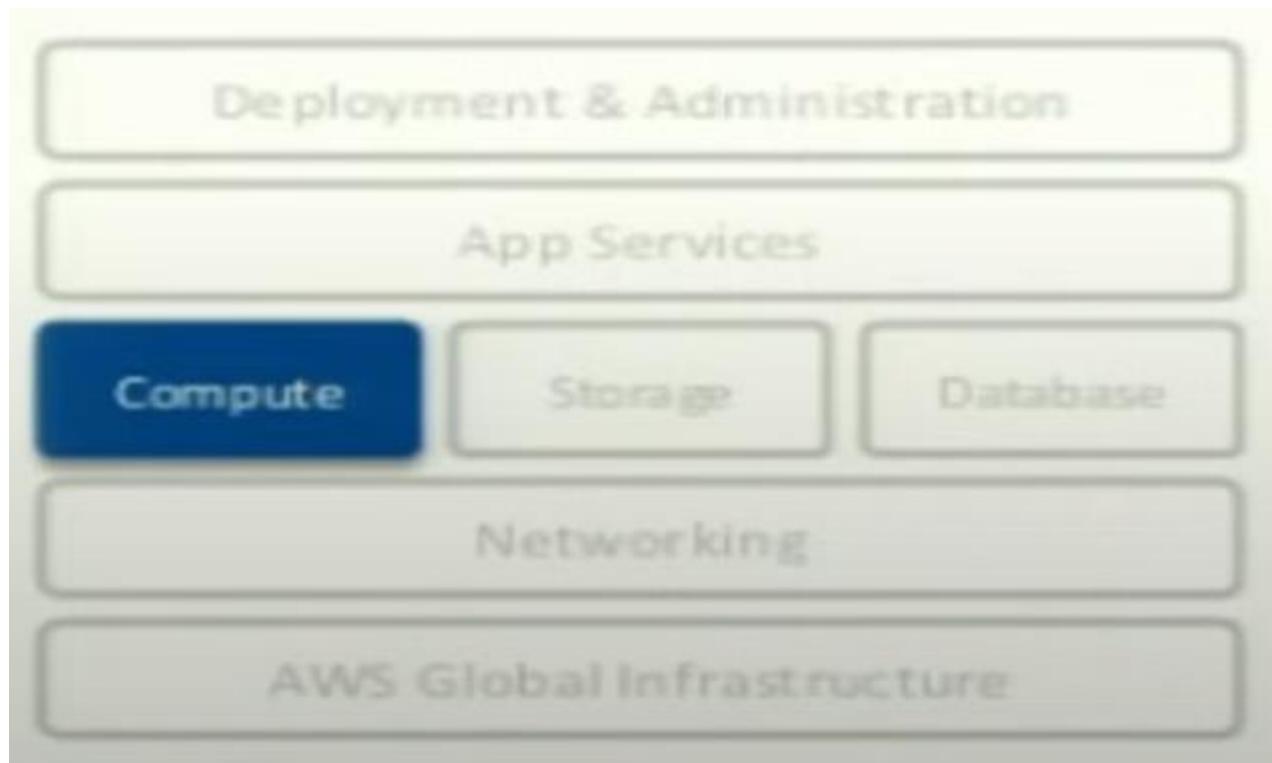
Storage

- Storage services in AWS involve
 1. S3 Durable Storage
 2. Elastic Block Store



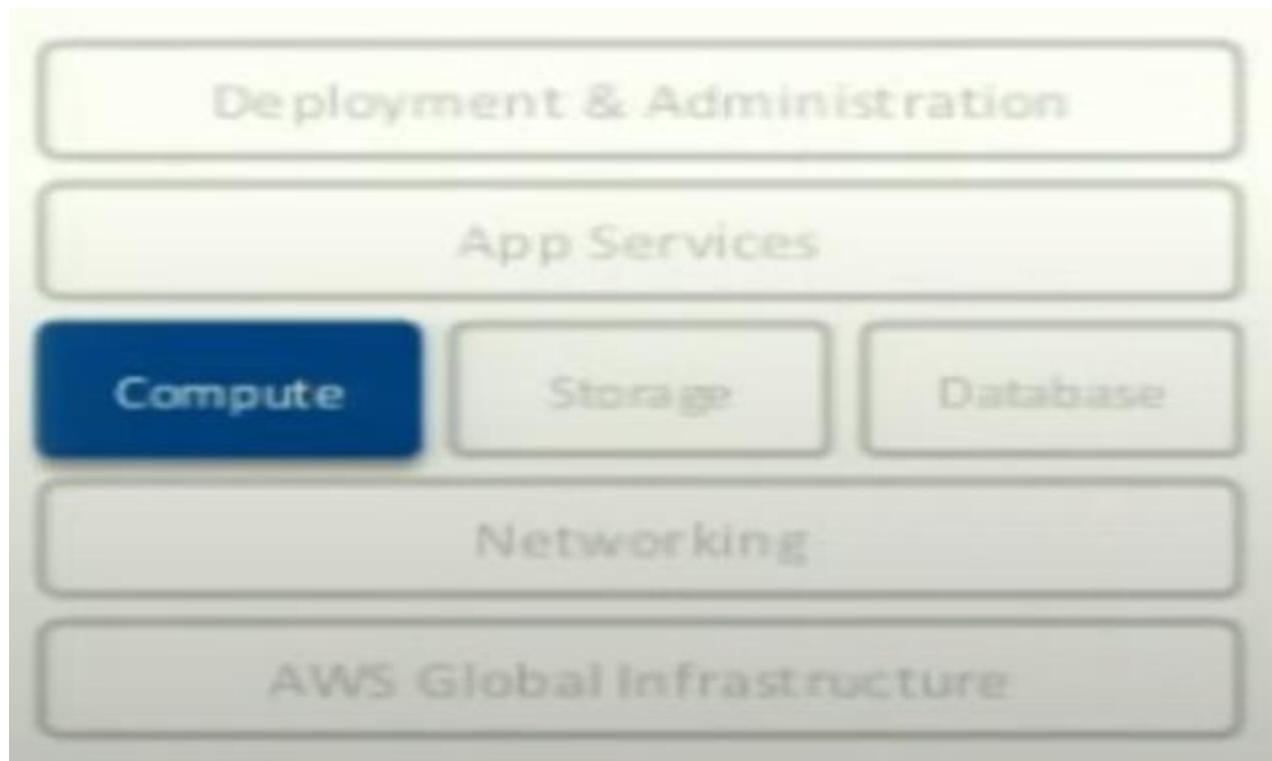
Database

- Database services in AWS involve
 1. Relational Database Service
 2. DynamoDB



App Services

- App services in AWS involve
 1. Amazon SQS
 2. Amazon SES
 3. CloudFront



Content Delivery Network

- Cloud storage today is complemented using content delivery network technology.
- Generally, whenever any file is stored in the cloud it can be replicated and cached at multiple locations using the CDN technology.
- This makes the sharing of the file much more efficient.
- Suppose, a file is not cached and is available only at a single location. If this file is being simultaneously accessed by many people around the globe then the response time will increase.
- Creating multiple copies of such files and caching them at different locations will reduce the response time and make the file easily accessible.

Content Delivery Network

- CDN is a geographically distributed and interconnected group of servers that caches content close to the end user.
- Allows quick transfer of assets needed to load internet content such as images, videos.
- CDN reduces latency, bandwidth consumption and prevents the original servers from attacks such as DDoS, man in the middle etc.
- CDN enhances the user experience by faster loading of images, videos, html files etc.

Content Delivery Network

Example of a CDN

- A server present in Mumbai hosts the static web content of a website whose content is requested by a user in Kerala.
- Upon receiving the request, the server sends the data to the user and also sends a copy of the data to the CDN point of presence that's geographically closest to the user.
- Next time, the same user or any other user in that geographical area requests the same content, it will be delivered by the CDN server and not the original server.

Advantages of Cloud Computing

- Centralized computing where all computing resources are present in large datacenters managed by few service providers.
- Many organizations can avoid the large cost of creating and managing their own datacenters by using computing resources from large service providers.
- Easy scalability
- Ease of access – anywhere, anytime, anyhow.

Serverless Computing

- Next gen of cloud computing which provides backend services on an as-use basis.
- In today's cloud computing scenario, software development companies rent servers to run, test and deploy their applications. Generally, these companies rent more space than desired (maybe as a safe side for future expansion).
- Thus, the company is paying rent for the services which are not being used currently.
- In serverless computing, the pricing is done based on the computation and there is no need to reserve and pay for fixed amount of server space. If less apps are being developed, less servers can be rented and they can be scaled up when needed.

<https://www.cloudflare.com/learning/serverless/what-is-serverless/>

Serverless Computing

- Traditional cloud computing is like mobile data plan in which you pay for 1.5 GB of data daily even if you are not going to use it everyday.
- Serverless computing is like paying for only the amount of bytes you have consumed. Like, if yesterday you consumed 1 GB pay for that and if today you consumed 500 MB then pay less.
- Serverless computing does not mean that servers are not involved. It simply implies that server management is taken care of by the vendor and not the developer.

Function as a Service (FaaS)

- A serverless backend service allowing developers to execute *modular pieces of code* on the edge. These small modular pieces of codes are called functions.
- The developer can write or update a small piece of code on the fly which can then be executed in response to an event, such as user clicking a web element.
- Extremely helpful in implementing *microservices*.
- Large applications are divided into microservices which are further divided into several functions which are then offered as a service.

<https://www.cloudflare.com/learning/serverless/glossary/function-as-a-service-faas/>

Function as a Service (FaaS) -

- Each function performs only one action – efficient, lightweight codes which load immediately in response to an event
- Each function is independent – a function doesn't call other functions (no nesting)
- Less libraries in the function – more libraries slow down the function and make it harder to scale.

<https://www.ibm.com/topics/faas>

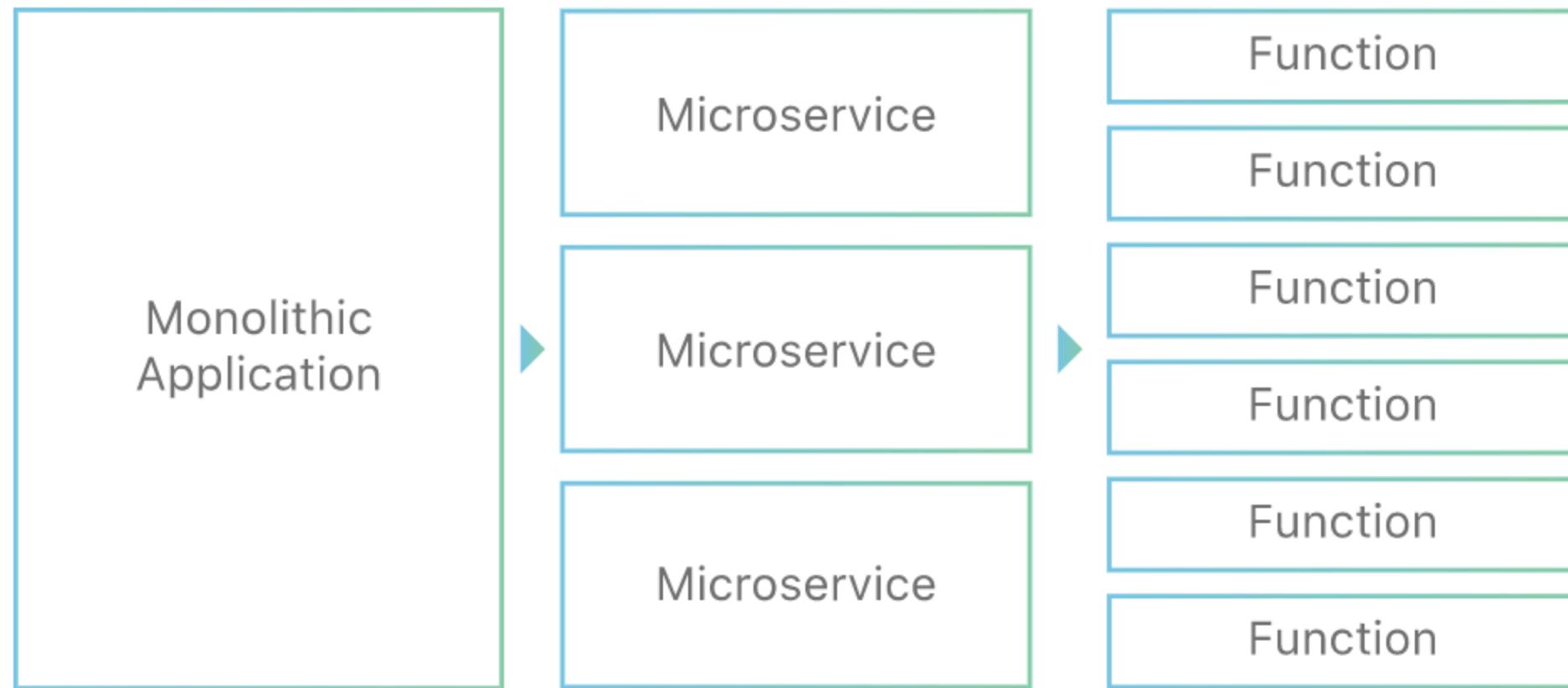
Function as a Service (FaaS) - Advantages

- Focus more on code, not infrastructure
- Pay only for the resources you use and when you use them
 - In FaaS, you pay only when an action occurs
 - When action is done everything stops – no code runs, no server is allocated, no costs are incurred
 - Costing is dynamic and applicable for scheduled tasks
- Faster deployment and provisioning of microservices (add-ons).

<https://www.ibm.com/topics/faas>

Microservice

- Microservice involves building an application using a set of small modular components or functions.



- This is in contrast with the monolithic app development approach where all the code is interwoven into a large system.
- In a monolithic architecture, even a minor change in the app requires hectic deployment process.

Microservice

- In microservice architecture, developers can create and modify small pieces of code using serverless computing and FaaS.
- Each microservice performs its own function, runs independently of other parts of the application, operates in its own environment and stores its own data.
- If the human body is considered as an app, different systems such as nervous system, respiratory system, digestive system can be considered as microservice.
-

IoT Cloud Platforms

1. Amazon Web Services (AWS)
2. Microsoft Azure
3. IBM Watson
4. Cisco IoT Cloud Connect
5. Salesforce IoT Cloud
6. Firebase (Google)
7. Parse (Facebook)

IoT Cloud Platform Comparison

	Azure	AWS	IBM Watson	ThingSpeak
Protocols	HTTP, MQTT	HTTP, MQTT	HTTP, MQTT	HTTP, MQTT
Certified hardware	Raspberry Pi2, Intel	Intel, TI	Arduino UNO, Raspberry Pi, ARM	Arduino UNO, Raspberry Pi, Particle
Languages	.Net	JAVA, C	Python, JAVA, C	JAVA
Pricing	F (messages per day, number of devices)	F (messages traffic)	F (data traffic, number of devices, data storage)	Open source/free

Need of Edge Computing

- The computing power of cloud is tremendous but network bandwidth forms the bottleneck.
- Bandwidth has not increased as fast as the data generation capacity.
- The speed of data transportation to the cloud has become the ultimate limitation.
- Around 1 GB of data is generated by a self-driven car every second and it requires real-time processing.
- Sending this data to cloud and waiting for processing takes too long.

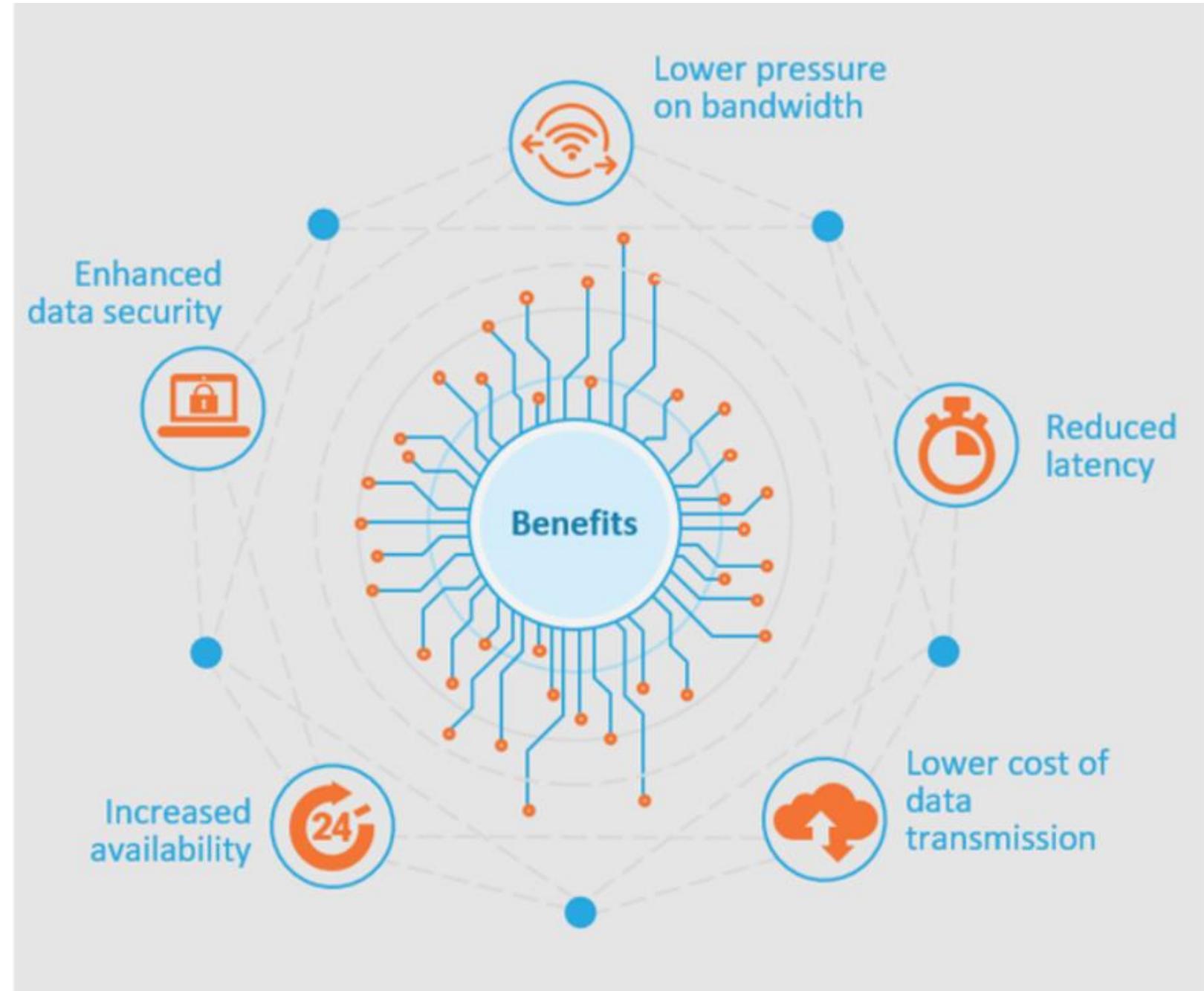
Fog/Edge Computing

- Processing of time sensitive data on the edge of the network i.e. closest to where data is generated.
- The definition of edge is arbitrary
 1. Mobile phone is the edge between body things and cloud
 2. Gateway is the edge between home devices and cloud

Edge Computing Advantages

1. Minimize latency
2. Reduce bandwidth requirement
3. Resolve data privacy and security
4. Improve reliability
5. Cloud Offloading

Edge Computing Advantages



Latency Minimization

- With the computing capability present close to the sensors, the time required for decision making is greatly reduced.
- Latency-sensitive actions have to be separated from latency-tolerable actions.
- Applications such as autonomous vehicles, AR and VR require real-time processing of sensor data and hence cannot afford high latency.

(From “Edge Computing: Vision and Challenges by Weishong Shi”)

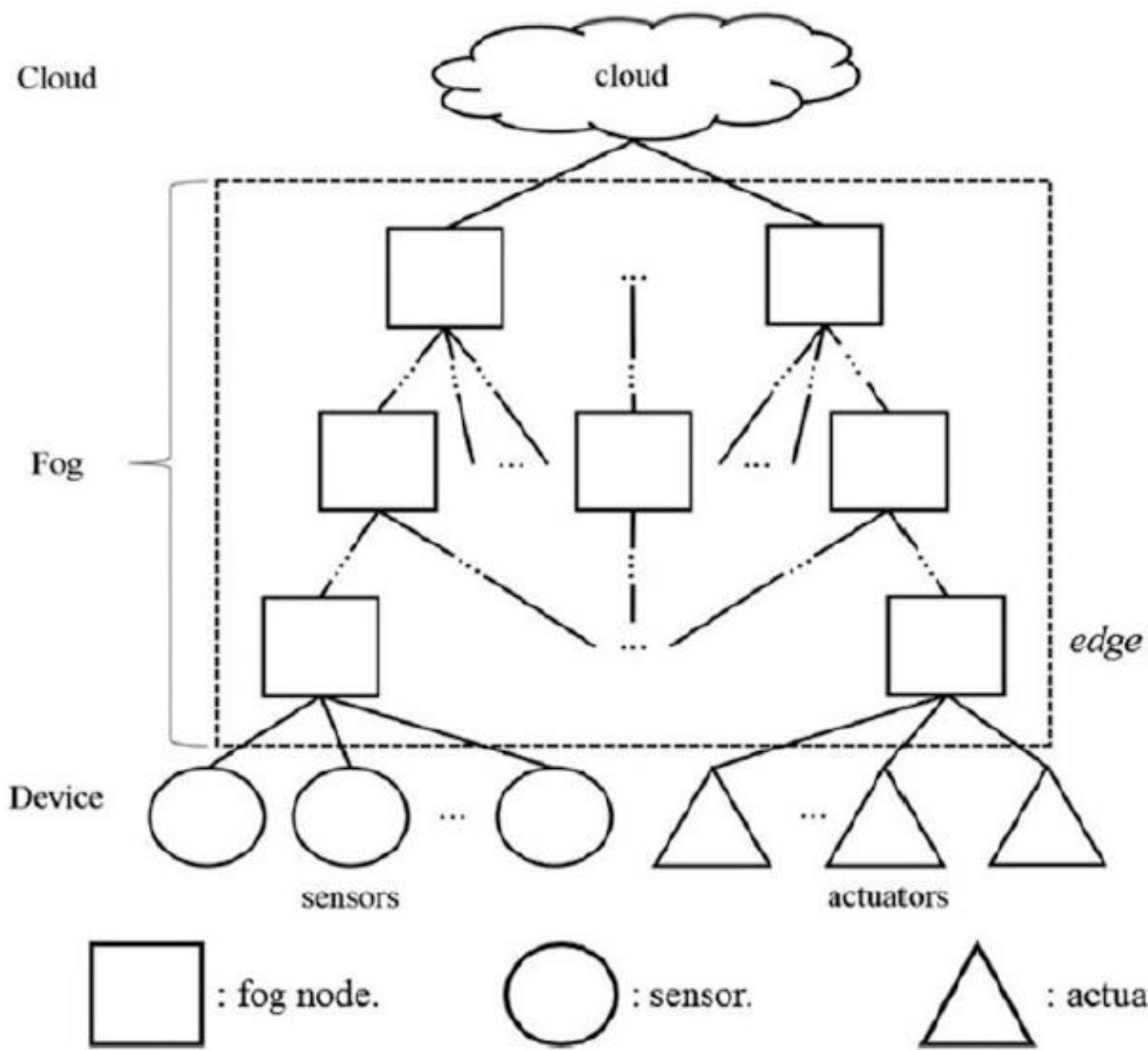
Bandwidth Minimization

- With some of the data processed at the edge, the data to be transmitted to the cloud decreases, thus reducing the bandwidth.
- In video processing applications, only annotated video data is sent to the cloud and not the whole video.
- Thus, a video annotation software is run on the edge computing device which reduces the data to be sent to cloud.

Data Privacy

- Edge computing device can filter out sensitive data so that it is not shared with everyone over the cloud.
- Software modules called privacy mediators are run on the edge devices that separate the sensitive data and stores it.
- This further reduces the bandwidth required for cloud sharing.

	Cloud	Fog
Location	<ul style="list-style-type: none"> Centralized in a small number of big data centers 	<ul style="list-style-type: none"> Often distributed in many locations, potentially over large geographical areas, closer to users. Distributed fog nodes and systems can be controlled in centralized or distributed manners.
Size	<ul style="list-style-type: none"> Cloud data centers are very large in size, each typically contain tens of thousands of servers. 	<ul style="list-style-type: none"> A fog in each location can be small (e.g., one single fog node in a manufacturing plant or onboard a vehicle) or as large as required to meet customer demands. A large number of small fog nodes may be used to form a large fog system.
Applications	<ul style="list-style-type: none"> Typically support applications that can tolerate round-trip delays in the order of a few seconds or longer. 	<ul style="list-style-type: none"> Support significantly more time-critical applications that require latencies below tens of milliseconds or even lower.



Edge Server Functions

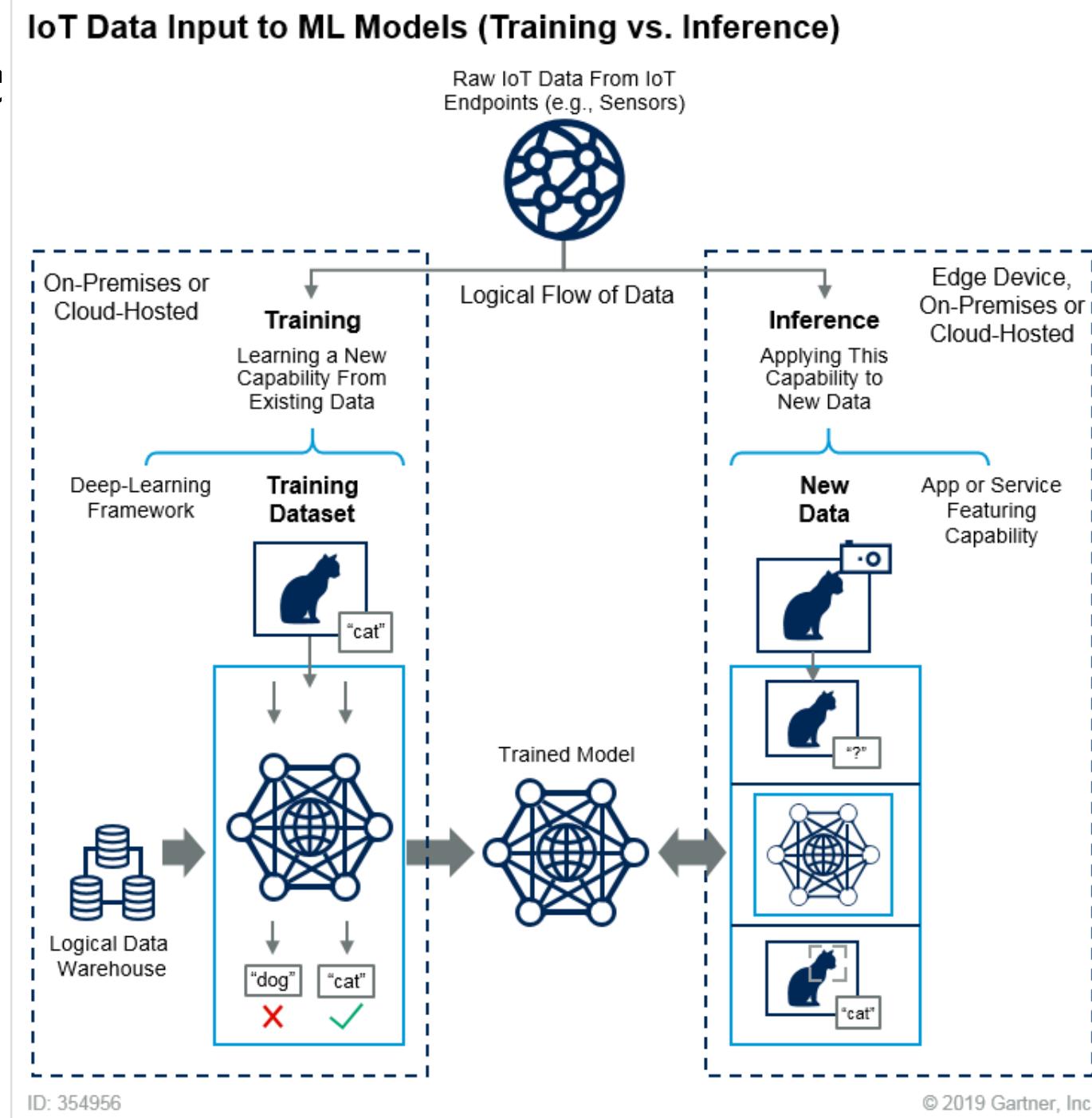
- Let's consider an example of a Smart Home System. The typical functions that an edge server needs to perform in this case are:
 1. Pre-processing of data received from different sensors
 2. Managing all the devices (electrical appliances) connected together in a smart home
 3. Real time processing of time-sensitive data and decision making using AI/ML
 4. Routing of the data to the cloud server
 5. Storing the local copy of the important data
 6. Providing data security, privacy and authentication
- Each of these functions require a different service/engine to be run on the edge device.
- For managing the devices, a service like KubeEdge or Bosch IoT Suite needs to be run on the edge server which will help in device management.

Edge Server Functions

- For real-time processing of data, stream processing engines such as Apache Spark needs to be installed on the edge server.
- Edge device must also be compatible with all the protocols such as MQTT, HTTP because different sensor nodes can use different protocols.
- Edge device must support data communication using varied technologies such as BLE, Zigbee, WiFi, LoRa etc.
-

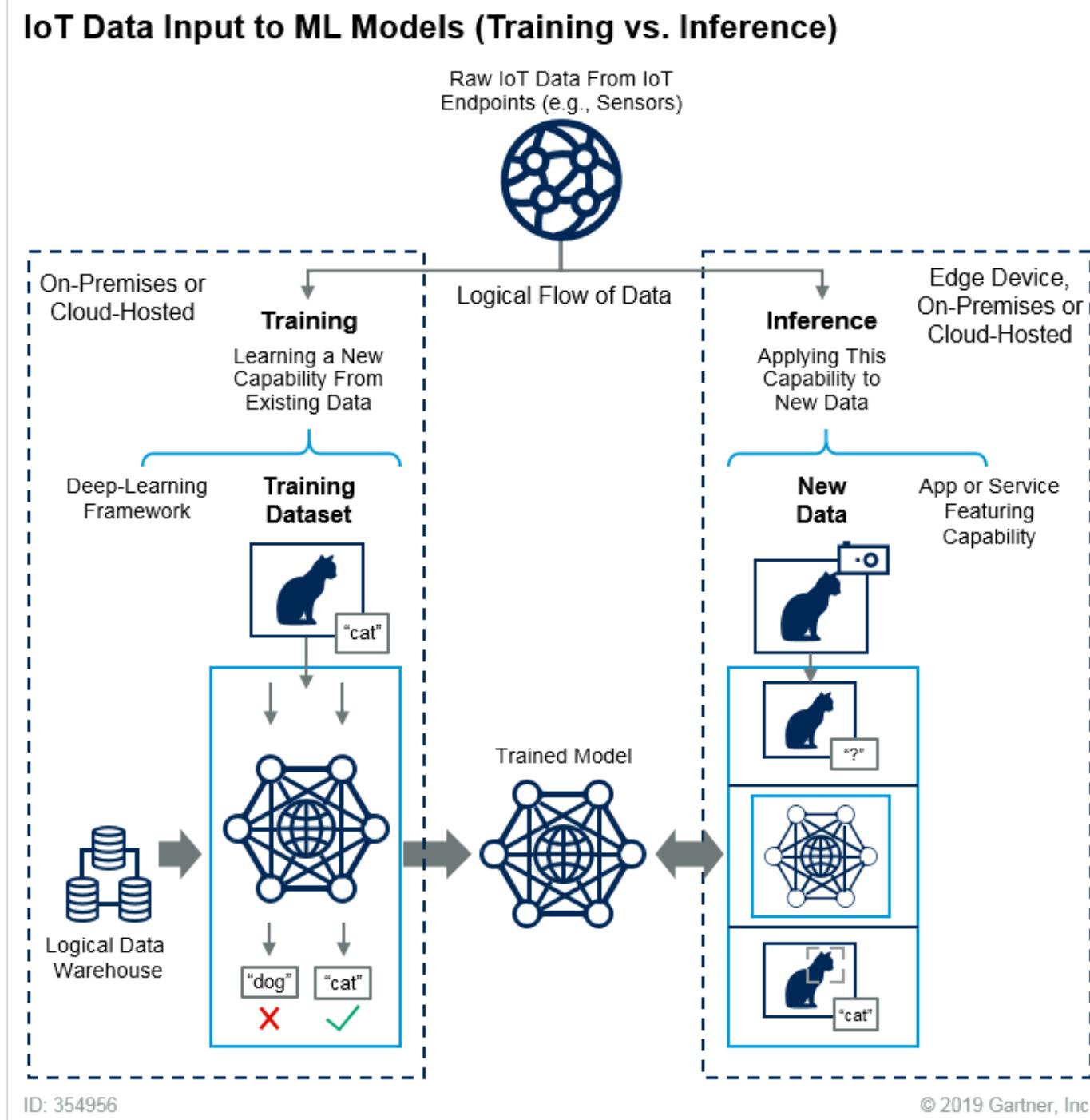
Machine Learning on Edge

- Generally, ML involves 2 steps – Training and Inference.
- In Training, the deep learning framework or model is trained using the data-set. Data set can be existing or can be generated by the IoT devices.
- Training an ML model requires huge amount of data and powerful processors hence it is generally performed on cloud servers.



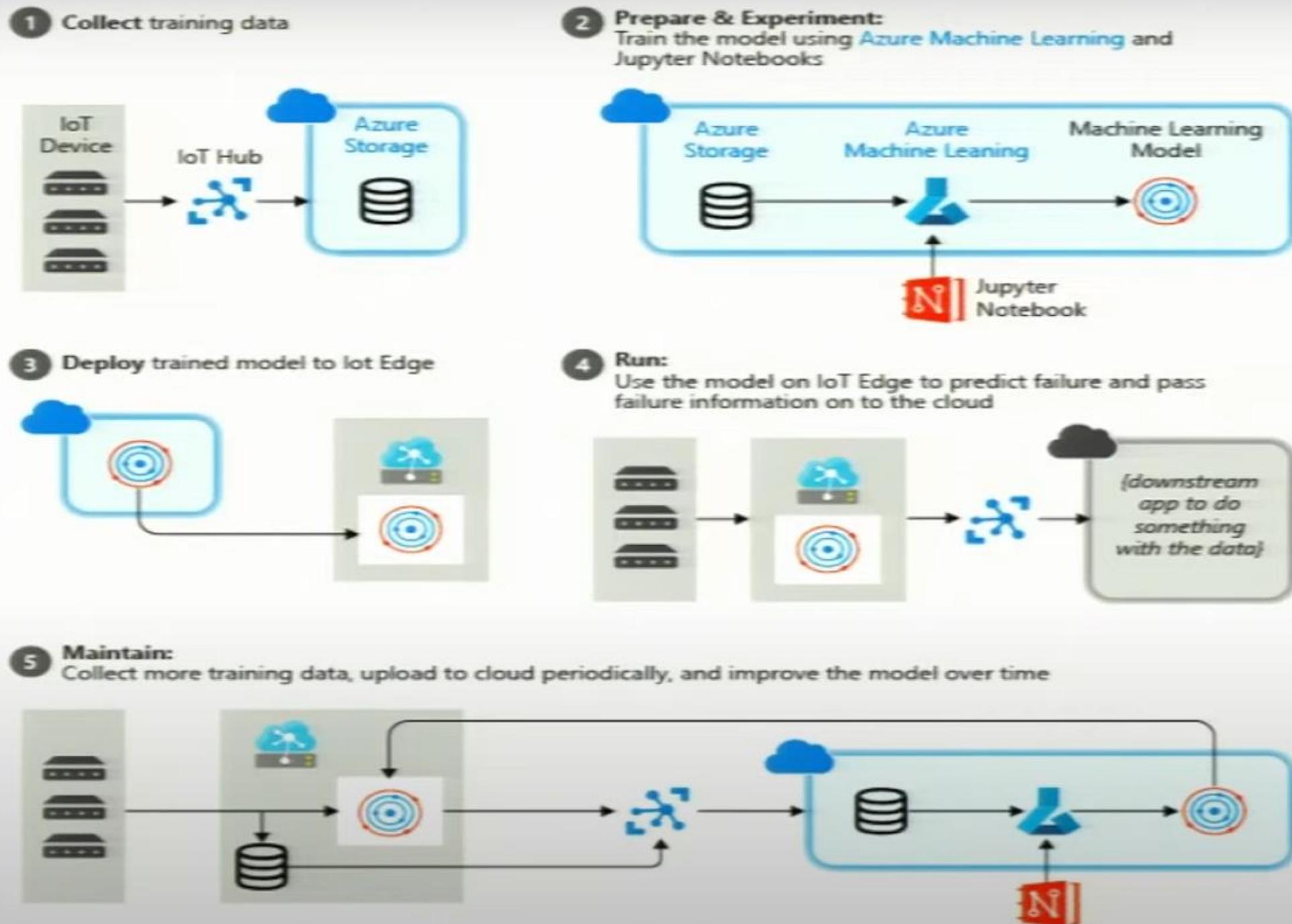
Machine Learning on Edge

- Once the ML model is trained on the Cloud server, it is deployed on the edge devices as shown.
- New data generated by the IoT devices is then fed to this model for inferencing or making a prediction/decision.
- Thus, data generated by the IoT devices is fed simultaneously to the cloud (for training) and edge (for inferencing).



Machine Learning

- Workflow of ML on the edge

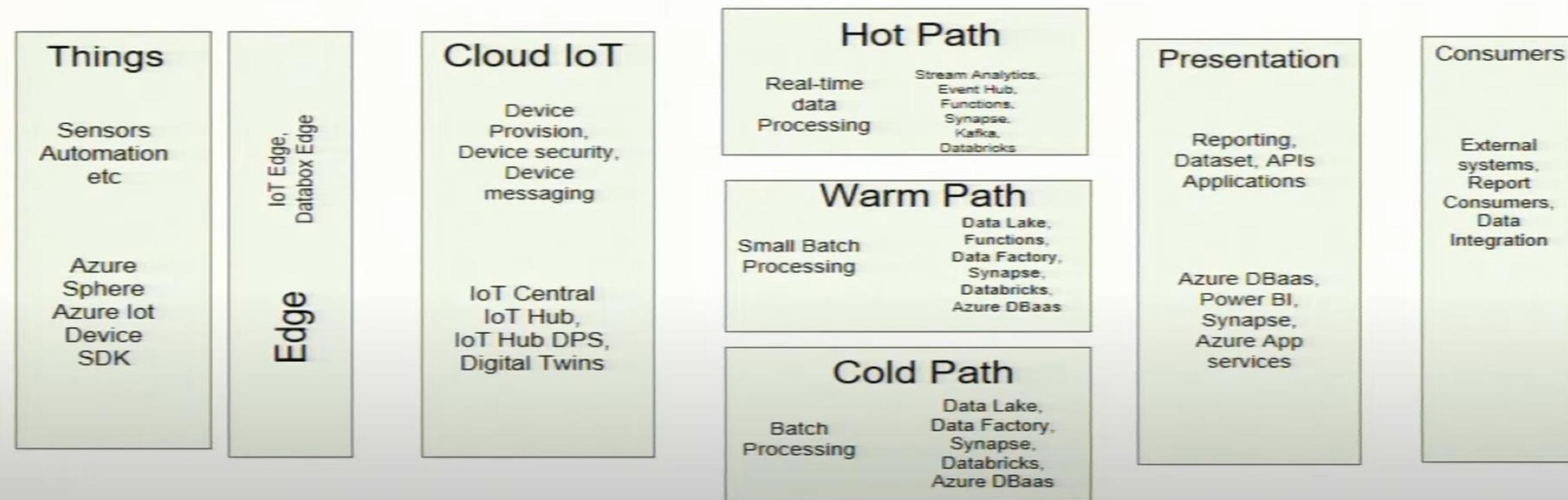


Machine Learning on Edge

- Raw data collected from the sensors cannot be directly used to train the ML model.
- Data preparation needs to be done before feeding it to the ML model.
- Data preparation involves data clean-up, data reformatting or preprocessing to inject additional information.
- It also involves calculating an explicit label for every data point in the sample based on actual observations of the data.
- This step allows the ML model to find correlations between the sensor data patterns.

Data flow in IoT

- Flow of data in a typical IoT network



https://www.youtube.com/watch?v=lWVlSfKtAK4&t=1361s&ab_channel=IITKanpurJuly2018

Data Flow in IoT

- Hot path signifies real-time processing of time sensitive data using stream analysis tools such as EventHub, Synapse, Kafka etc.
- Cold path refers to the processing of time-tolerant data which is aggregated over a period of them and then processed together i.e. batch processing. The tools used here are Data Lake, Data Factory and Data Bricks.

Functionalities of Edge Computing Platform

- Any edge computing platform shall perform the following functions:
 1. IoT Device Management
 2. Data Ingestion
 3. M2M Brokers
 4. Data Storage
 5. Function as a Service
 6. Running AI/ML Algorithms

[\(https://www.youtube.com/watch?v=hmBe8WrG0LA&t=1s&ab_channel=IITKanpurJuly2018\)](https://www.youtube.com/watch?v=hmBe8WrG0LA&t=1s&ab_channel=IITKanpurJuly2018)

Data Ingestion

- Process of importing large, assorted data files from multiple sources into a single data base, where it can be accessed and analyzed.
- The data can be in multiple formats and it is sanitized and transformed into a uniform format using an extract/transform/load (ETL) process.
- Large volume of data coming at high speed and throughput.
- Involves connecting to data sources, collecting the data, pre-processing it, extracting the data and detecting the changed data.
- Popular data ingestion tools – Apache Kafka, Amazon Kinesis

Types of Data Ingestion

- Batch Processing
 - Data is collected in discrete manner and sent to applications or servers
 - Grouping of data is performed at the edge server according to predefined criteria
 - Not suitable for real-time analysis, Less expensive
 - Popular framework – Hadoop, Apache Spark
- Stream Processing
 - Real-time collection and processing of data – hot path
 - Computationally more expensive
 - Popular engines – Apache Spark, Apache Storm, Apache Samza.
- Micro Batching
 - Middle way between stream processing and batch processing
 - Data is grouped into smaller chunks and processed more frequently

Data Pre-Processing

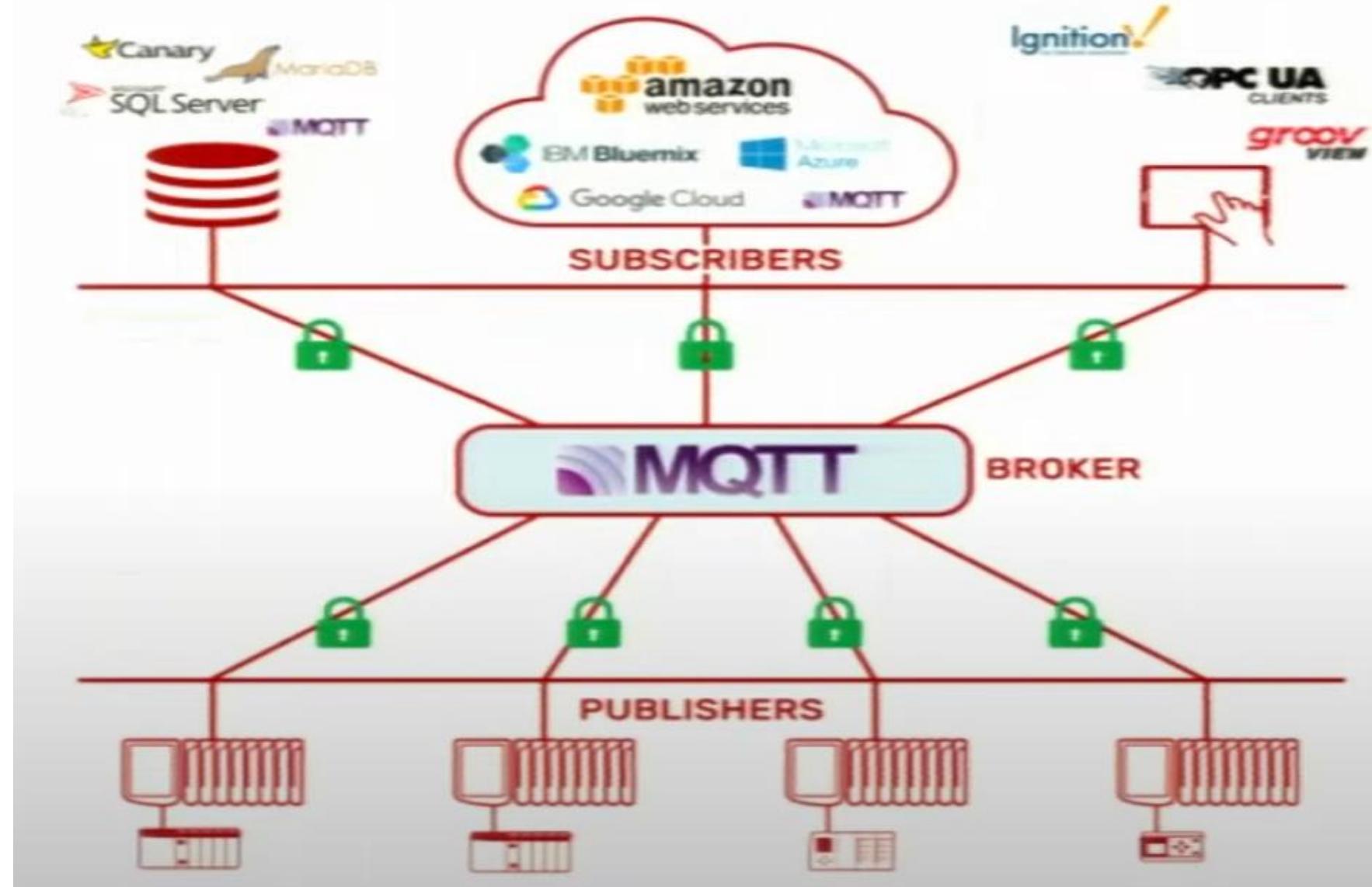
- After data ingestion, the edge server has to perform the pre-processing of raw data in order to make it suitable for analysis.
- Pre-processing of data involves – data cleaning, transformation and integration.
- Data cleaning – identifying and correcting errors such as missing values, data duplication and outliers etc.
- Data integration – combining data from multiple sources to create a unique data-set. This data can be in different format, structure and semantics.

Data Pre-Processing

- Data Transformation – converting data into suitable format for analysis. Mostly done using normalization, standardization and discretization.
- Data Reduction – Reducing the size of dataset while preserving the important information. Steps performed for data reduction are feature selection, feature extraction, sampling, clustering and compression.

M2M Broker

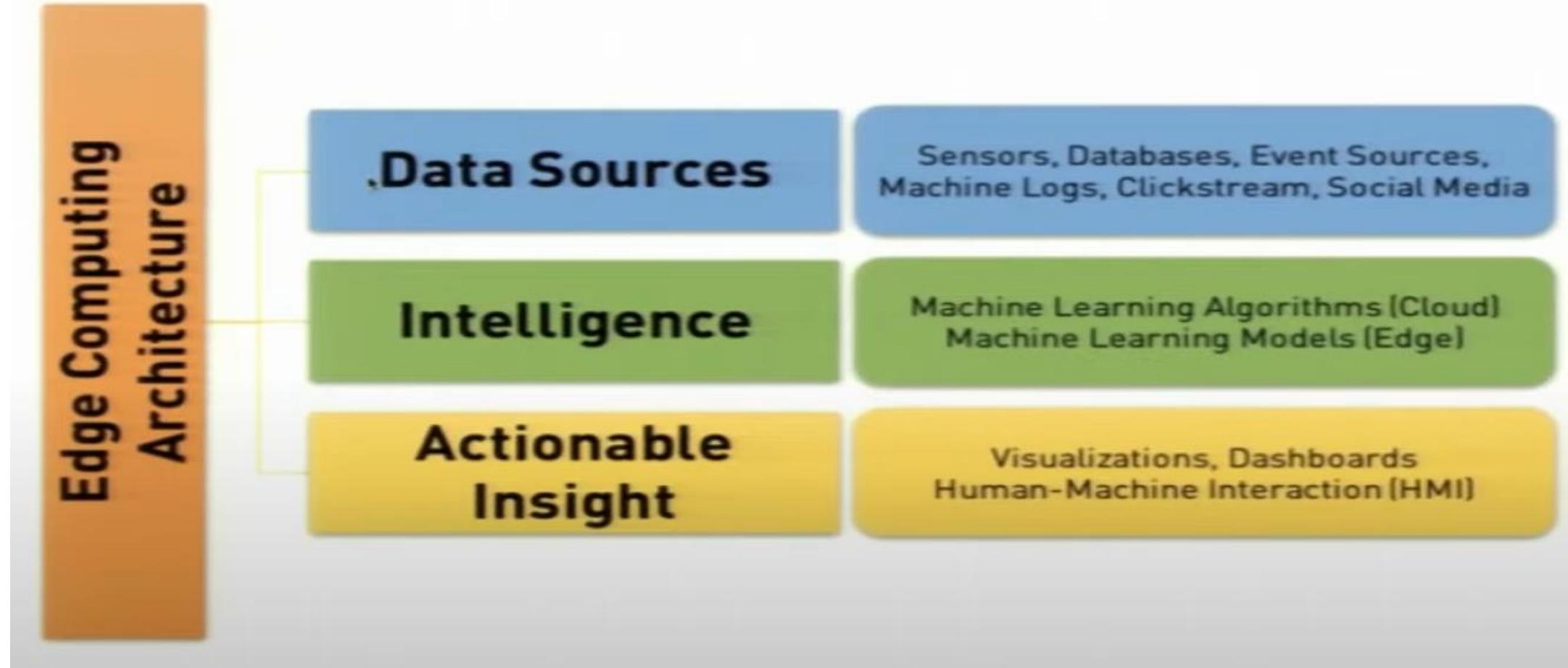
- The edge device has to act as a broker for machine to machine communication
- Sensor nodes will act as publishers and applications will act as subscribers.



Data Storage

- The edge device has to store structured (quantitative) as well as unstructured (qualitative) data.
- Object storage – unstructured data like video feed goes into object storage
- No SQL/Time-Series Database – more structured data goes into time series database.

Edge Computing Architecture



- Edge computing architecture can be divided into 3 layers as shown

Challenges in Edge Computing

1. Programmability
2. Naming
3. Data Abstraction
4. Application Distribution
5. Service Management

(From “Edge Computing: Vision and Challenges by Weishong Shi”)

Programmability

- In cloud computing, all the computing and storage services run on the same platform. For ex, in Microsoft AWS all the services run on Windows platform.
- So its easy for an application developer to write a code for that particular platform.
- But in edge computing, the edge nodes are heterogeneous since they have to deal with heterogeneous sensor nodes.
- Hence the platform on which edge node operate is also heterogeneous. Thus writing application codes for such heterogeneous platforms is a challenge.

Naming

- Naming – giving unique identity to each and every device connected on a network.
- DNS (Domain Name System) – responsible for unique name of device – like roll numbers of students.
- Things in IoT – tremendously large in number, heterogeneous, mobile.
- Naming schemes to incorporate these properties is a challenge.

Data Abstraction

- Hiding all but relevant data about an object to reduce complexity and increase efficiency.
- Edge nodes abstracts data into following categories
 1. Data for real time processing – highest priority
 2. Data stored in edge device for further processing – moderate priority
 3. Data send to cloud platform – least priority
 4. Data to be discarded after storing

Data Abstraction

- Data received at edge node – huge volume, different format (text, speech, video etc.), intermittent.
- Abstraction also involves – noise removal, event detection and privacy check etc.
- Performing these tasks on **huge amount** of data in **real time** by **resource constrained** edge node is a challenge.

Application Distribution

- Due to limited resources at the edge devices, the applications running must also be limited.
- Application offloading has to be performed where some applications can be run on other edge devices or cloud platform.

Service Management

- With multiple applications running at the edge node providing multiple services, some basic tasks must be performed in terms of service management.
 1. Differentiation / Prioritizing
 2. Extensibility
 3. Isolation

Differentiation / Prioritizing

- Certain applications or processes must be given priority over others in terms of execution.
- For ex., in a Smart Home system the safety and security of the home must be given priority. Any incident of fire hazard or robbery must be resolved at the earliest.
- The personal healthcare devices connected to the smart home must be given priority. For ex., if someone suffers cardiac arrest then it must be reported at the earliest.

Extensibility

- The applications running on the edge node must have the capability to incorporate new nodes and removal of dysfunctional nodes.
- For ex., new things or appliances purchased in a smart home must easily be integrated with the existing edge node.

Isolation

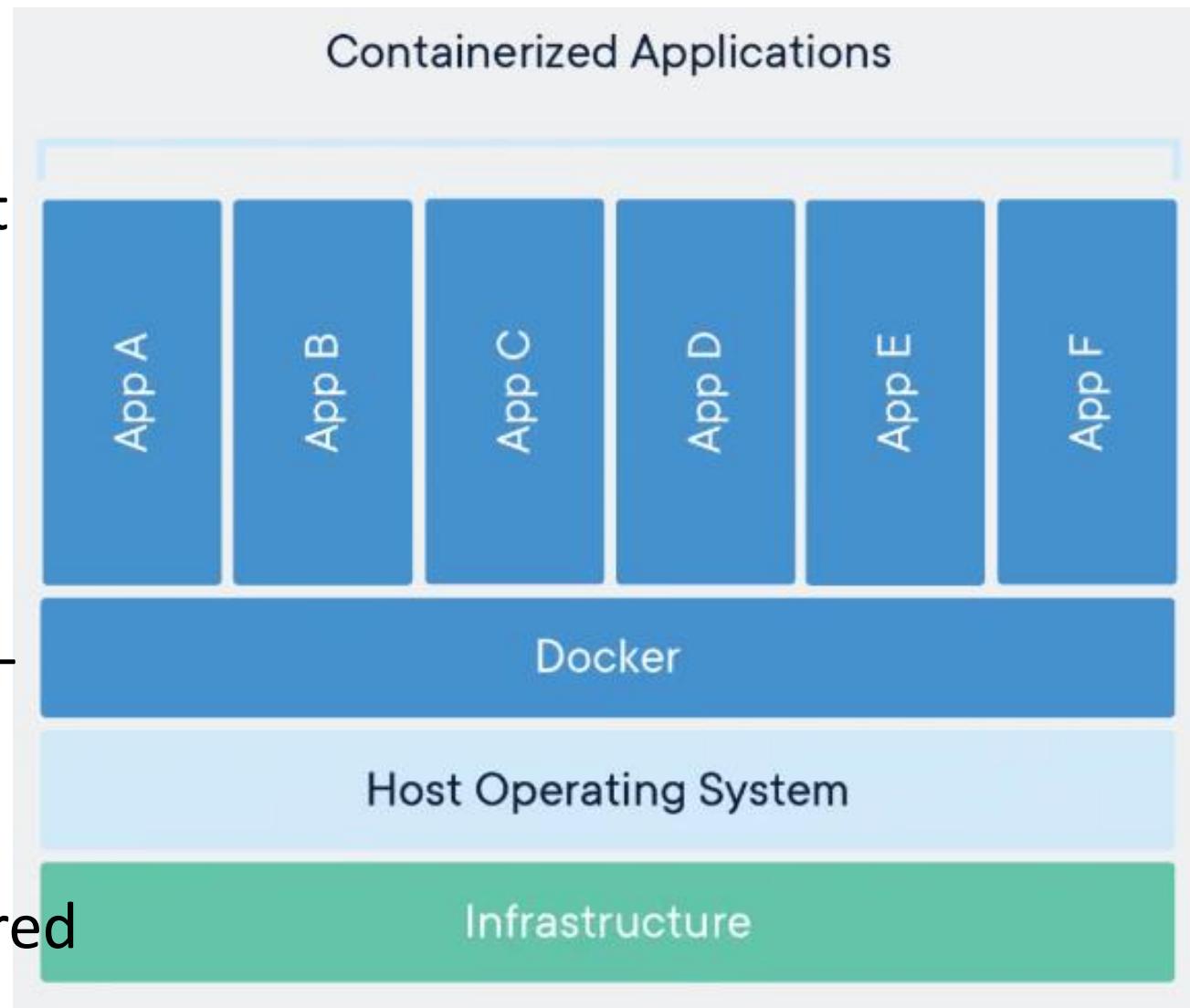
- Different applications using the same resource should be isolated from each other.
- Failure of one application should not affect the resource of another one.
- For ex., the lights in a room can be controlled by light sensor as well as motion sensor. The failure of one of them should not affect the control of lights by another.

KubeEdge architecture

- Kubernetes is an open source container orchestration system/platform for automating software development, scaling and management.
- A container packages up code and all the dependencies of an application so that the app can run quickly and reliably from one computing environment to another.
- Containerization isolate application from each other on a shared OS.
- Containerized applications run on top of a host which in turn runs on a OS.

KubeEdge architecture

- Docker is an open-source project that automates the deployment of apps as portable, self sufficient containers that can run on cloud / edge.
- Docker containers can run anywhere - on customer premises or cloud.
- Pod is a group of containers with shared storage/network resources and a specification of how to run the containers.

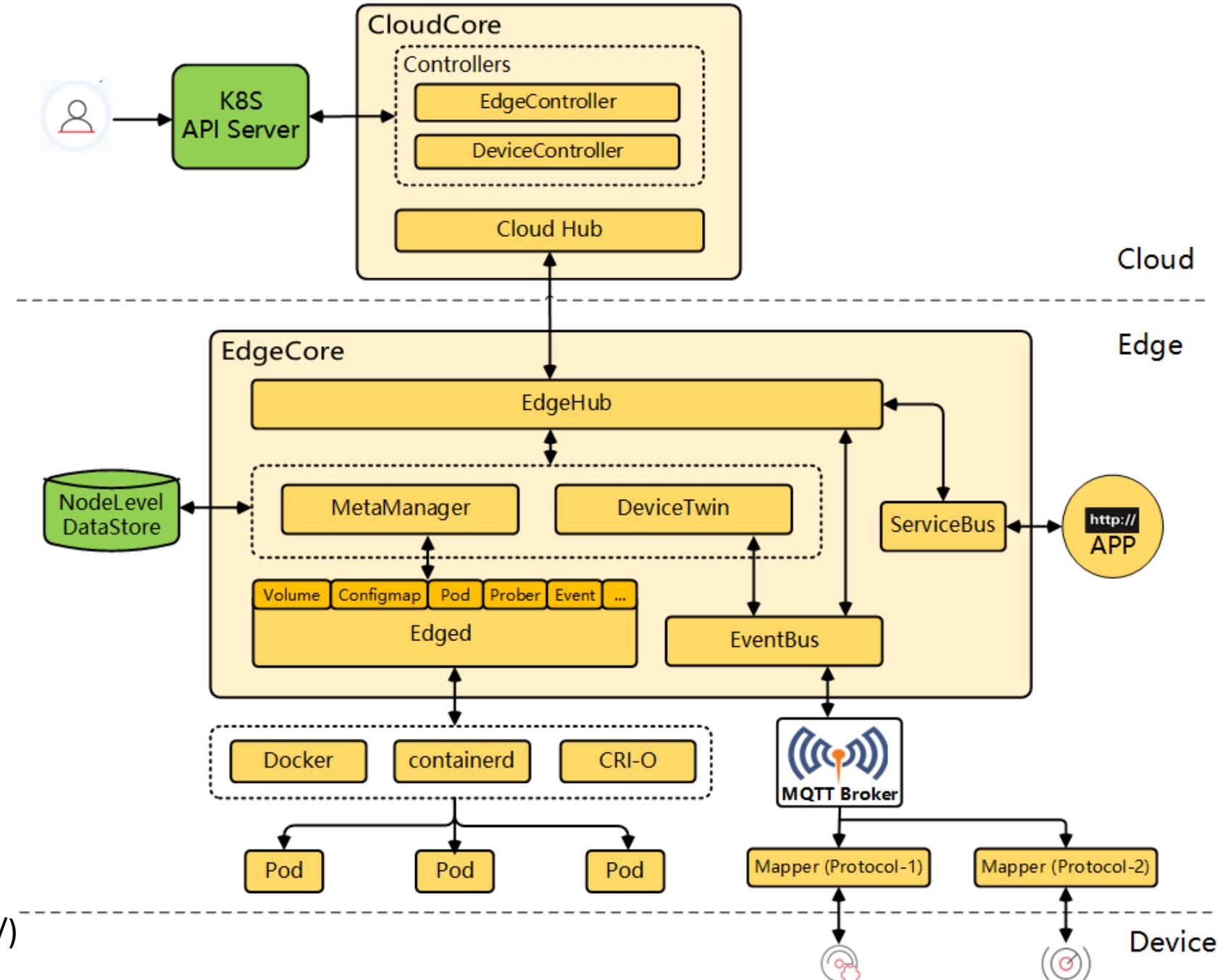


(<https://www.docker.com/resources/what-container/>) (<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined>)

KubeEdge architecture

- Containerized application orchestration and device management can be done at the edge node using open source platform such as KubeEdge.
- A typical example of cloud and edge based IoT system is shown. Here developers can write regular HTTP/MQTT applications, containerize them and can run them anywhere – at the edge or cloud.

Edge Computing Architecture



KubeEdge architecture

- Edged – an agent that runs on the edge node and manages containerized apps
- EdgeHub – a web-socket client responsible for interacting with cloud service. It involves function such as synching cloud-side updates with the edge and reporting status of edge devices to cloud.
- CloudHub – a similar interface to EdgeHub on the cloud side.
- EdgeController – manages edge nodes and pod metadata so that the data can be targeted to a specific edge node
- EventBus – MQTT client to interact with MQTT services
- ServiceBus – HTTP client to interact with HTTP services

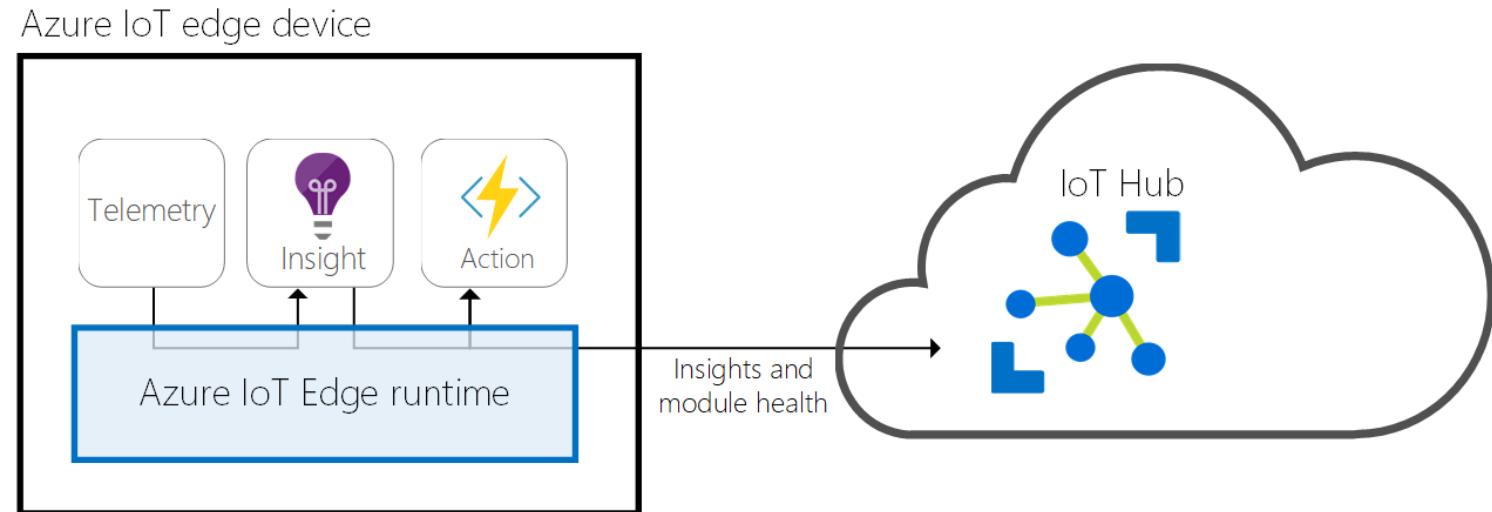
Azure IoT Edge

- Azure IoT Edge – a device focused runtime that enables the user to deploy, run and monitor containerized Linux workloads.
- Azure IoT Edge is made up of 3 components:
 1. IoT Edge Modules – containers that run Azure services, third-party services or user's own code. Modules are deployed on edge devices and execute locally.
 1. IoT Edge Runtime – runs on each edge device and manages the modules deployed.
 1. Cloud Based Interface – to remotely monitor and manage IoT devices

https://www.youtube.com/watch?v=1ZvtIdy_eU&t=1s&ab_channel=IITKanpurJuly2018

Azure IoT Edge

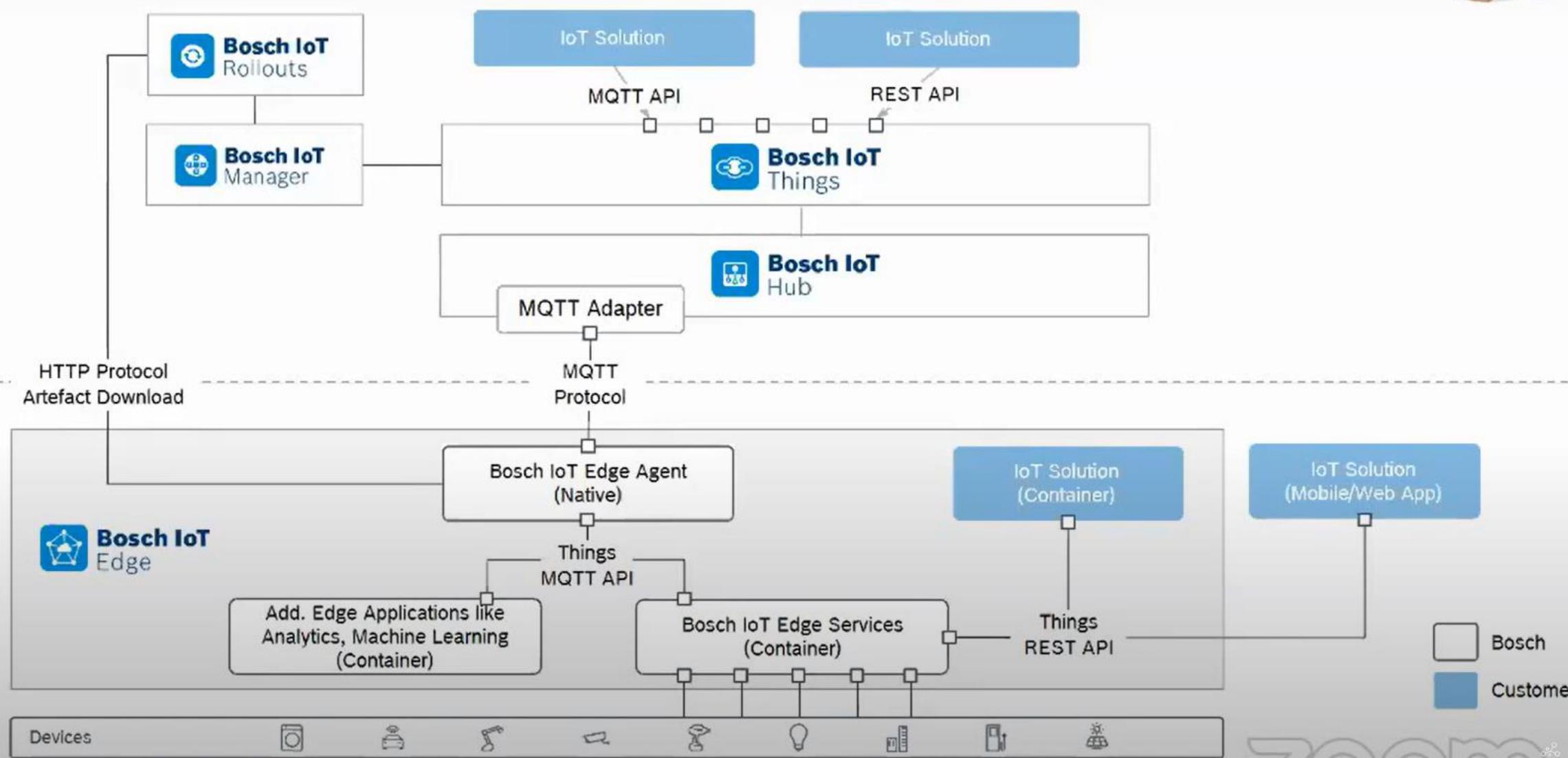
- Azure IoT Edge device shown
In the figure contains 3 modules
Performing telemetry,
Insight and action.



- All these modules are managed by IoT Edge runtime. Runtime is a collection of programs that turn a device into IoT Edge device.
- These modules are run as containers on the edge device.

Bosch IoT Edge Suite

Cloud



Bosch IoT Edge

- Similar to Azure IoT Edge, in Bosch IoT Edge suite, the functionalities like analytics, ML and other services are bundled into containers and are deployed on the edge device.
- These IoT edge devices interact with the cloud service via IoT Hub present there.

Security and Privacy in IoT

IoT Security vs Cybersecurity

- Cybersecurity generally deals only with the security of data – like data privacy, confidentiality etc.
- IoT security must also deal with the security of associated hardware (sensor nodes) in addition to data security.
- Thus, cybersecurity can be considered as a subset of IoT security.
- And as the sensor nodes in any IoT system are heterogeneous, IoT security is not merely the application of a single, static set of meta-security rules applied over networked devices.
- Each IoT system requires its unique security solution due to its unique application.

General Security Requirements of IoT System

1. Authenticity – identification of any party involved in storage, transmission and processing of data
2. Integrity – confirmation of the correctness of data
3. Confidentiality – protection of data stored, transmitted from getting disclosed
4. Access Control – service provision only to authorized users
5. Dependability – uninterrupted service in case of errors and failures

Challenges in securing IoT systems

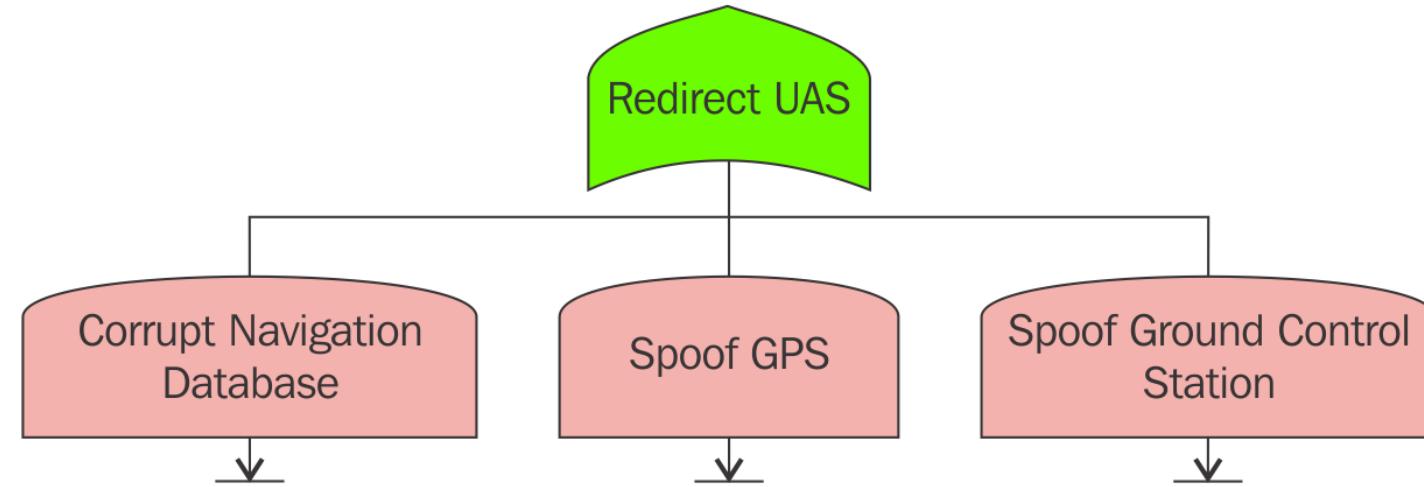
1. Large number of deployed systems
2. Heterogeneous nature of deployed devices
3. Deployment in hostile and uncharacterized environment
4. Resource constrained devices
5. Maintaining low cost
6. Mobility of nodes

Building Attack Tree

- Attack trees are conceptual diagrams showing how a target can be potentially attacked.
- They are important in understanding the critical vulnerabilities of an IoT system which can then be used to design security solutions.
- Attack tree must include all the possible ways in which a attack can be implemented by the attacker.

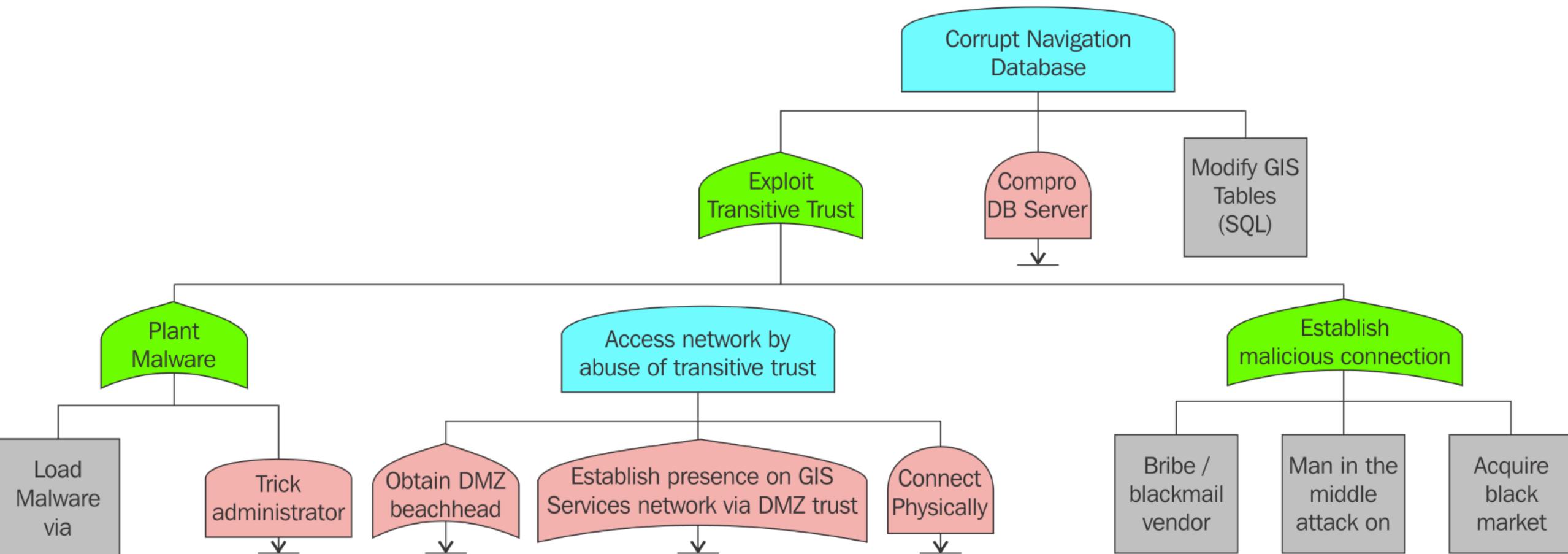
Building Attack Tree

- Consider an example attack where the attacker wants to redirect a Unmanned Aerial System (UAS) or a drone to a false location.



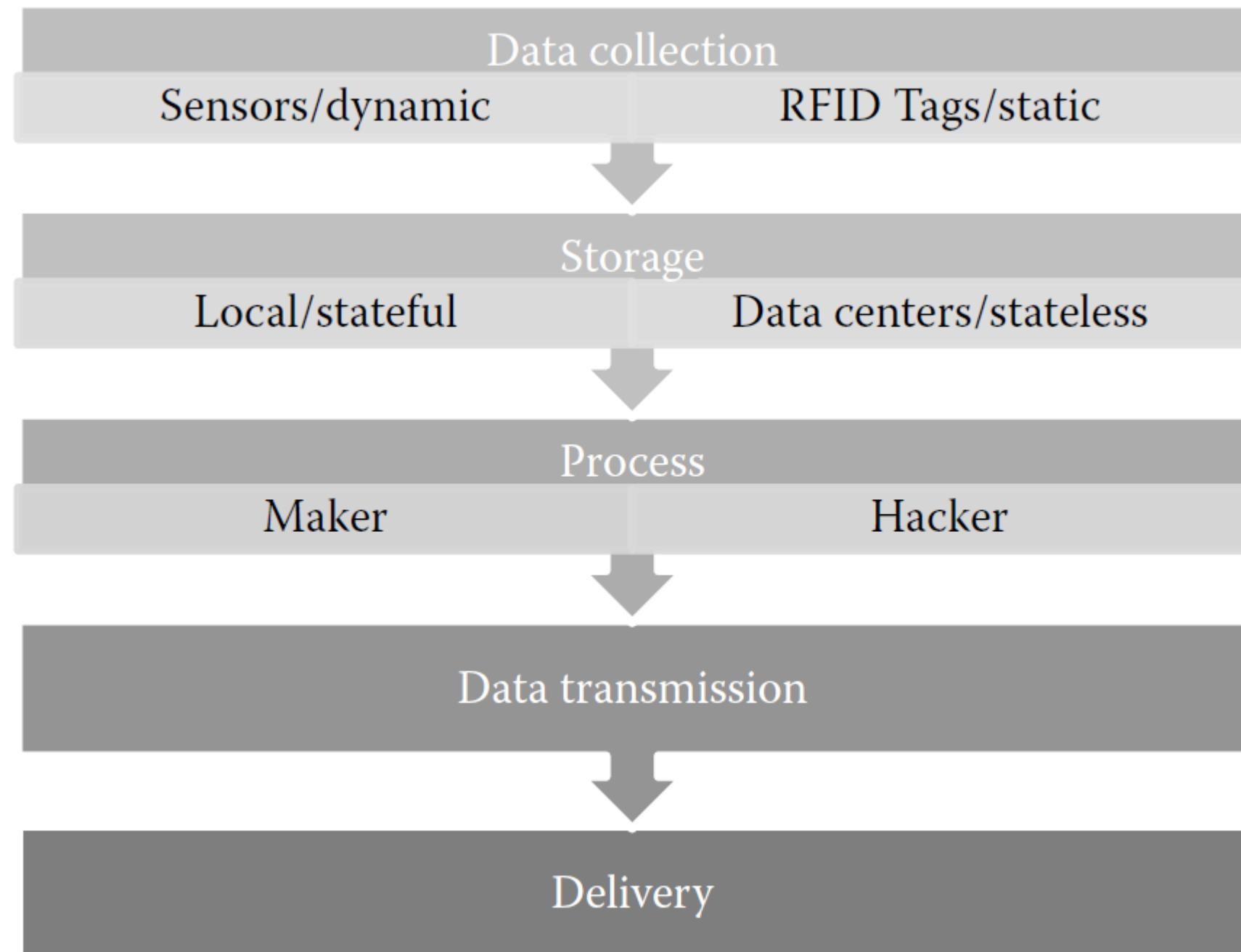
- The attacker can do it in the ways given in the tree diagram.
- Each of these way can single-handedly achieve the attacker's goal. Hence the attacking mechanisms are modeled as AND gate and the goal is modeled as OR gate (since it can be achieved by ANY of the mechanism).
- Now each of these mechanisms can be expanded through proper understanding as shown in the next slide.

Building Attack Tree



IoT System Phases

- Any IoT system can be classified into different phases as shown.



(From Book – Security and Privacy in
IoTs: Models, Algorithms and
Implementations)

Attacks on Phases

- Attacks can occur on all the Phases of an IoT system.

Data Perception

- Data leakage, data sovereignty,
- Data breach, data authentication

Storage

- Attack on availability, access control, integrity
- Denial of service, impersonation, modification of sensitive data

Processing

- Attack on authentication

Transmission

- Channel security , session hijack
- Routing protocols, flooding

Delivery end-to-end

- Man or machine
- Maker or hacker

Attacks on Phases

- Data leakage – export of unauthorized data to an unintended destination.
- Data loss – losing the data accidentally due to hardware/software failure.
- Data authentication – receiving data from unauthorized users.

Attack on Data Availability – data centers becoming unavailable to their intended clients. This can be caused by the following:

- Flooding by attackers – DDoS attack
- Flooding by legitimates – Flash Crowd
- Flooding by spoofing
- Flooding by aggressive legitimates

Attacks on Phases

- Distributed Denial of Service (DDoS) attack – overburdening of data centers / edge node by several malicious node.
- Flash Crowd – overload condition caused by huge number of legitimate users requesting the resources of data center simultaneously e.g. too many users accessing the same website at the same time (IRCTC Tatkal Booking).
- Flooding by spoofing – identity (IP address) of a legitimate user is faked by a malicious user which consumes the resources.
- Flooding by aggressive legitimates – restless users who repeatedly initiate similar requests within a short span of time.

Attacks on Phases

Data Modification

- Data can be modified by malicious users either at the origin (sensor node), in transit (channel) or at the gateways.
- Data modification may change the
 - Content of the information
 - Sequence of the packet delivery
 - Time of the packet delivery

Attacks

Attacks
can also
be
classified
based on
the layers
architecture

Application layer

- Revealing sensitive data
- Data destruction

User authentication
Intellectual property

Transport layer

- Denial of service
- Masquerade
- Cross heterogenous

Distributed denial of service
Man-in-the-middle

Network layer

- Routing protocol
- Address compromise

Sensing/perception layer

- External attack
- Witch attack
- Worm hole and sewage pool
- Broadcast authentication and flooding

Link layer attack
HELLO flooding
Selective forwarding
Access control

Attacks on IoT System

- Wormhole attack – attacker records data from a node, tunnels it to another external node and then retransmits it back to the original network.
- Selective forwarding – malicious nodes capture certain packets and drop them completely from the network.
- Sinkhole attack – sensor node is compromised and attracts all the information from the surrounding nodes, thus acting as a sink.
- Computational Attack – malicious action in computation system that affects the correct execution of program

Attacks on IoT system

- Side Channel Attack – attacks on the power and timing peripherals instead of the actual algorithm
- Eavesdropping, traffic analysis and passive monitoring.
- Man-in-the-middle attack – attacker establishes independent connection between 2 nodes such that they believe they are talking to each other. But actual communication occurs through attacker.
- Routing Threat – data is routed to different path or routed in closed loop

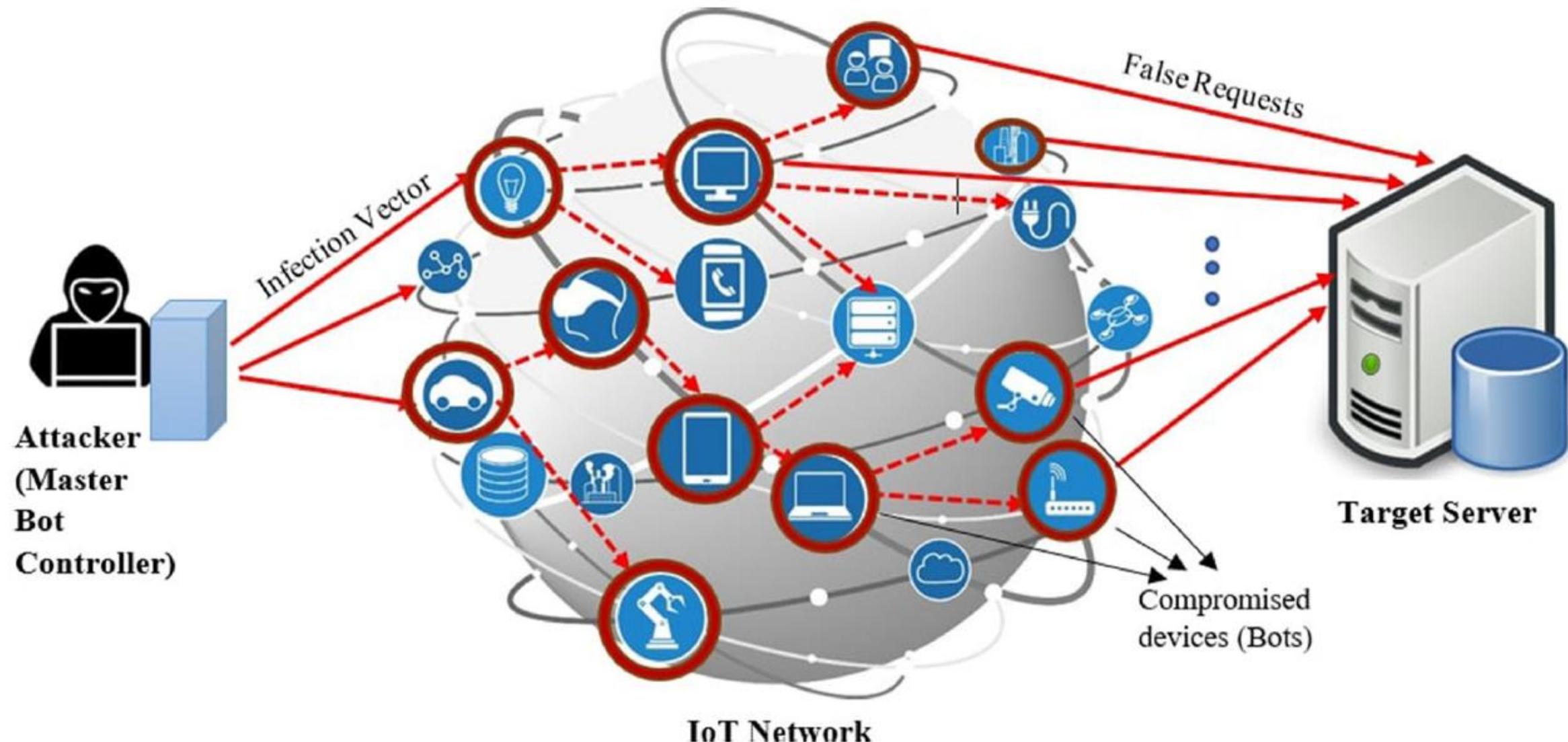
DDoS Attack

- In a typical DDoS attack, one or more IoT devices are compromised when an attacker installs malware on them.
- Such compromised IoT devices are called bots and the attacker is called Master Bot Controller.
- The bots act as slave to the controller and flood the server with fake requests, thus utilizing the resources completely.
- The network of bots and the controller is called botnet.

DDoS Attack

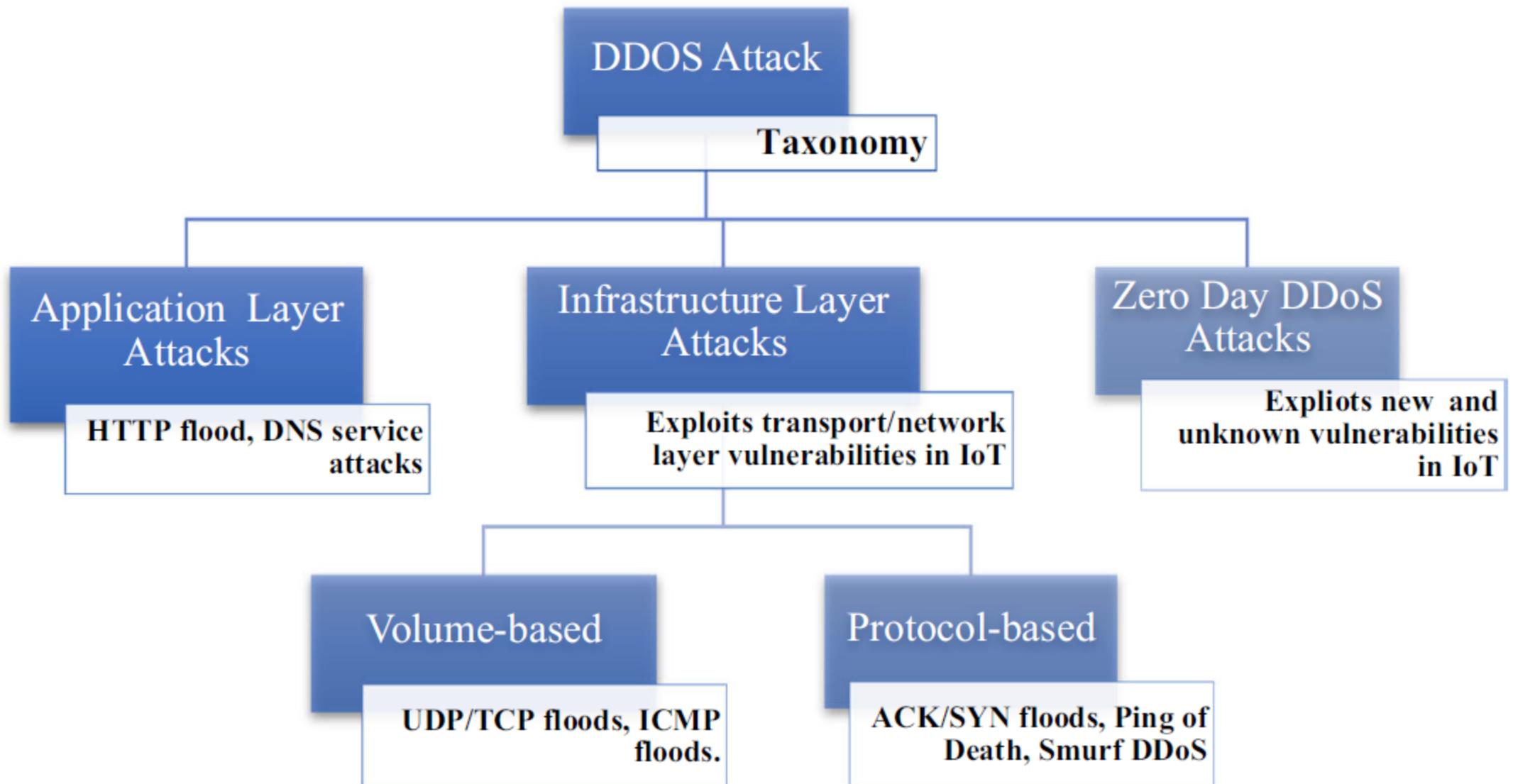
- The attacker generally targets a sensor node which has very limited resources and hence very basic security.
- Most IoT service providers provide enhanced security to gateways/routers as they are computationally powerful but provide very basic security to sensor nodes (default username and password).
- Attackers use a database of default username and password to brute force into these sensor nodes and compromise them.
- Thus getting entry into an IoT network is done through the weakest link.

DDoS Attack



From (A survey of DDoS attacking techniques and defence mechanisms in IoT network by Ruchi Vishwakarma and Ankit Jain)

DDoS Attack



From (A survey of DDoS attacking techniques and defence mechanisms in IoT network by Ruchi Vishwakarma and Ankit Jain)

DDoS Attack Types

- DDoS attack can occur on every layer of the IoT network as shown in previous figure
- Application Layer attacks occur on web applications while Infrastructure Layer attacks occur on the transport and network layer devices and services.
- Application layer attacks are harder to detect as they tend to generate traffic at a lower rate and the request seems to be legitimate.
- Infrastructure Layer attacks are volume based or protocol based and usually employ reflection or amplification techniques.

DDoS Attack Types

1. ACK and SYN Flood Attack

- Occurs during the 3-way handshake process of establishing connection in TCP
- The attacker sends ACK bit enabled packet having forged source address to the target device.
- The target device drops the packet as it does not have any established connection with the attacker.
- This requires processing of each incoming packet which leads to resource depletion of the target device.

DDoS Attack Types

1. ACK and SYN Flood Attack

- In SYN flood, attacker send SYN packet to the target node at very high rate for initiating the 3 way handshake process.
- The target node replies by sending ACK+SYN packet to the attacker.
- The attacker does not reply with ACK packet, thus making the connection half open and keeping the target device waiting for ACK packet.

DDoS Attack Types

2. Amplification Attack

- The attacker may establish a connection with the server/DNS using a forged IP address.
- Attacker then sends request to the server demanding very high volume of data.
- The server fulfills the demand, but the data gets sent to the target device instead of the attacker.
- The target receives this huge amount of data and processes it, leading to resource wastage.

DDoS Attack Types

3. UDP Fragment Attack

- The attacker continuously sends false UDP fragments to the target device which needs to be reassembled.
- The target device becomes overloaded in reassembling these packets which ultimately results in other services being slowed down.
- The packets after reassembling exceed the maximum allowable length.
- This type of attack is also called *Ping of Death* attack.

DDoS Attack Types

4. UDP/HTTP Flood Attack

- The attacker continuously sends packets to random ports of the target device anonymously which are not intended for any application.
- The target device is forced to check each port for such packets and then cannot find any corresponding application.
- It then has to send destination unreachable packet, which results in wastage of its resources.

DDoS Defence Mechanisms

1. Learning automata based techniques

- A threshold is set which defines the maximum value of serving capability of a server based on its computational resources
- If this threshold is crossed, then a DALERT (DDoS Alert) signal is issued throughout the network.
- Attacking device is identified by identifying the IP sending most number of service requests and its details are circulated throughout the network.
- All packets from this device are discarded if it fails to be legitimate during the sampling.

DDoS Defense Mechanisms

2. Honeypot based DDoS Defense

- Honeypots are used as trap for the intruders.
- Anomalies are detected in the incoming requests using the intruder detection system.
- Requests showing anomaly are directed towards the honeypot instead of the main server.
- Information of the suspect is stored and a verification request is made on the behalf of main server to check for the client's authenticity.
- If the client is an intruder, it is blocked from the network.

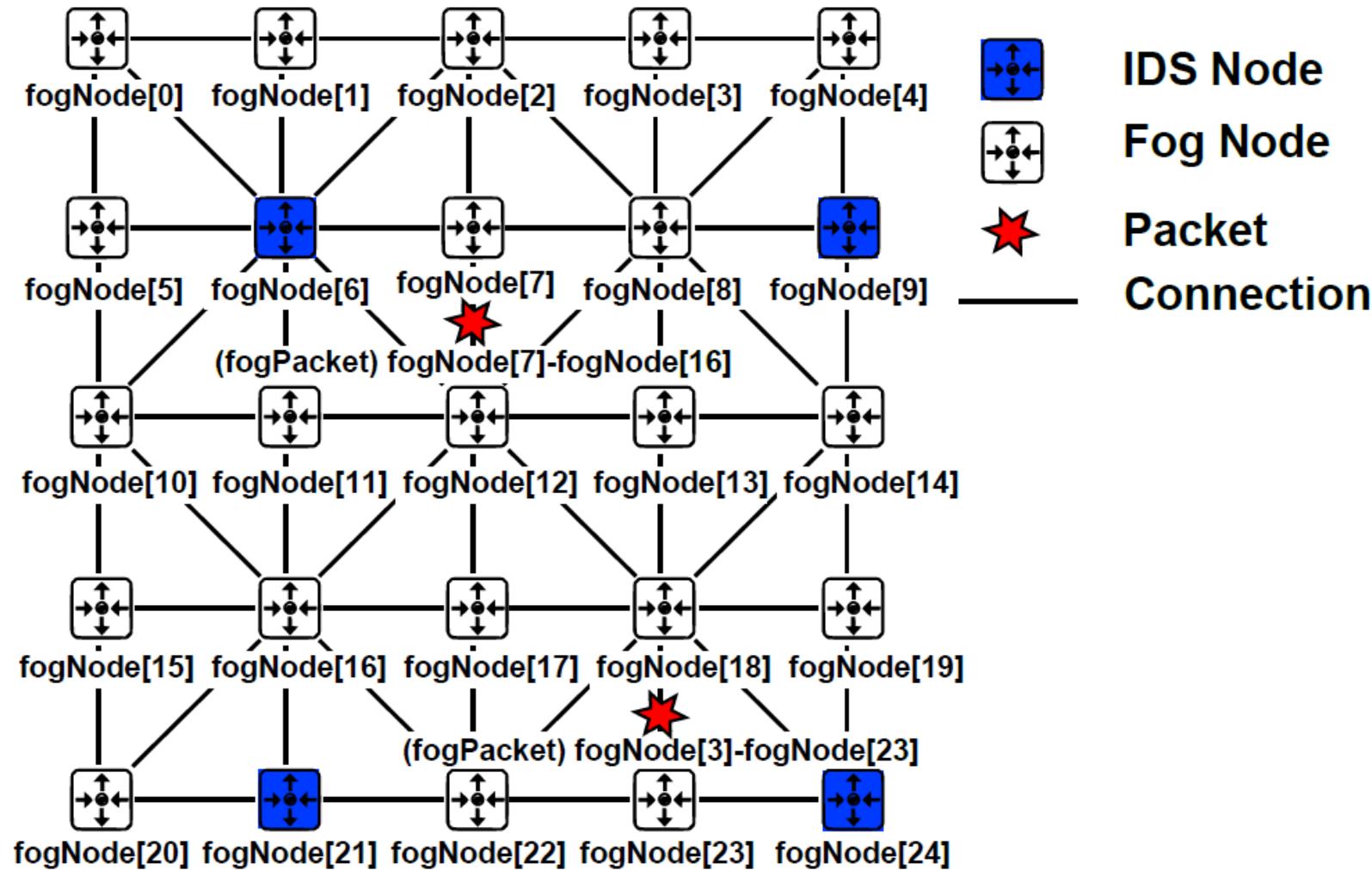
DDoS Defence Mechanisms

3. SDN based techniques

- In SDN, the data plane and control plane are decoupled from each other.
- Network control is handled by a centralized SDN controller which is capable of understanding the complete topology of the network.
- SDN controller can dynamically manipulate the flow of data packets in the network based on the condition of the overall network.
- SDN controller is a resource rich device and perform traffic analysis using SVMs, ANNs and other ML based algorithms.
- Thus, identification and mitigation of DDoS attacks become much easier in SDN than in the traditional network.

Man in the Middle Attack Detection

- A system to detect Man in The Middle attack is shown.
- Intrusion Detection Systems (IDS) nodes are installed in the network such that all fog/edge computing nodes are at one hop distance from them.
- This system assumes that MiM attack happens on fog nodes.



(From “A Detection and Prevention Technique for Man in the Middle Attack in Fog Computing” by Faruq Aliyu et al)

Man in the Middle Attack Detection

- IDS nodes perform periodic interrogation of the fog nodes. It acquires encryption key from the cloud and distributes it all fog nodes.
- The interrogation packets are encrypted and they consist of an integer value. It is expected that fog nodes will decrypt the packet, multiply the integer value by 2, encrypt the result and retransmit it to IDS.
- When the returned packet fails this criteria, it is assumed that the node is compromised.
- Additionally, IDS records the round trip time of the packets and if it exceeds certain threshold, then the node is termed as compromised.

Wormhole Attack

- The attacker gathers packets or bits from packets, tunnels them to another point in a network and then replays them into the network from that point.
- Wormhole attack is possible even if the attacker has not compromised any node and the communication protocols provide authenticity and confidentiality.
- The attacker can gather the information of the packets that are not even directed towards it, just by overhearing the information.
- The wormhole attack puts the attacker in very powerful position as compared to other nodes.

(From “Wormhole Attacks in Wireless Networks” by Yih Chun Hu)

Wormhole Attack

- The attacker, being a resource extensive device can now improve the metric of the packet and then transmit it into the network.
- Thus the wormhole route will now be treated as the best route and all the data is now transferred only through the attacker and all other routes are discarded by the network.
- The attacker now has control over the packet flow and can either drop the packets selectively, modify them or can completely discard all the packets, leading to a DoS attack like situation.

Detecting Wormhole Attack

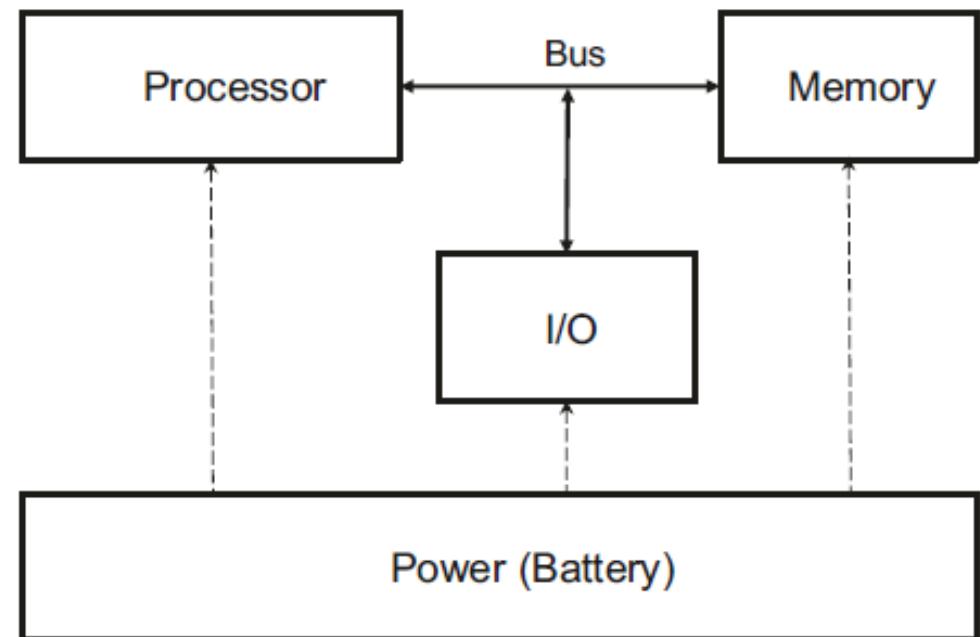
- In order to detect wormhole attack, leashes are introduced in the packets.
- Leash is an information added to the packet to restrict its maximum allowable transmission distance.
- There can be 2 types of leashes – geographical and temporal.
- Geographical leash ensures that the recipient of a packet is within a certain distance from the transmitter.
- Temporal leash ensures that the packet has an upper bound on its lifetime.

Detecting Wormhole Attack

- When packets are sent over multiple hops, each transmission requires use of a new leash.
- This allows the receiver of the packet to detect whether the packet has travelled longer than what was set in the leash.

Physical Layer Security in IoT

- Includes hardware as well as software Security
- IoT nodes are present in external areas which are easily accessible by attackers.



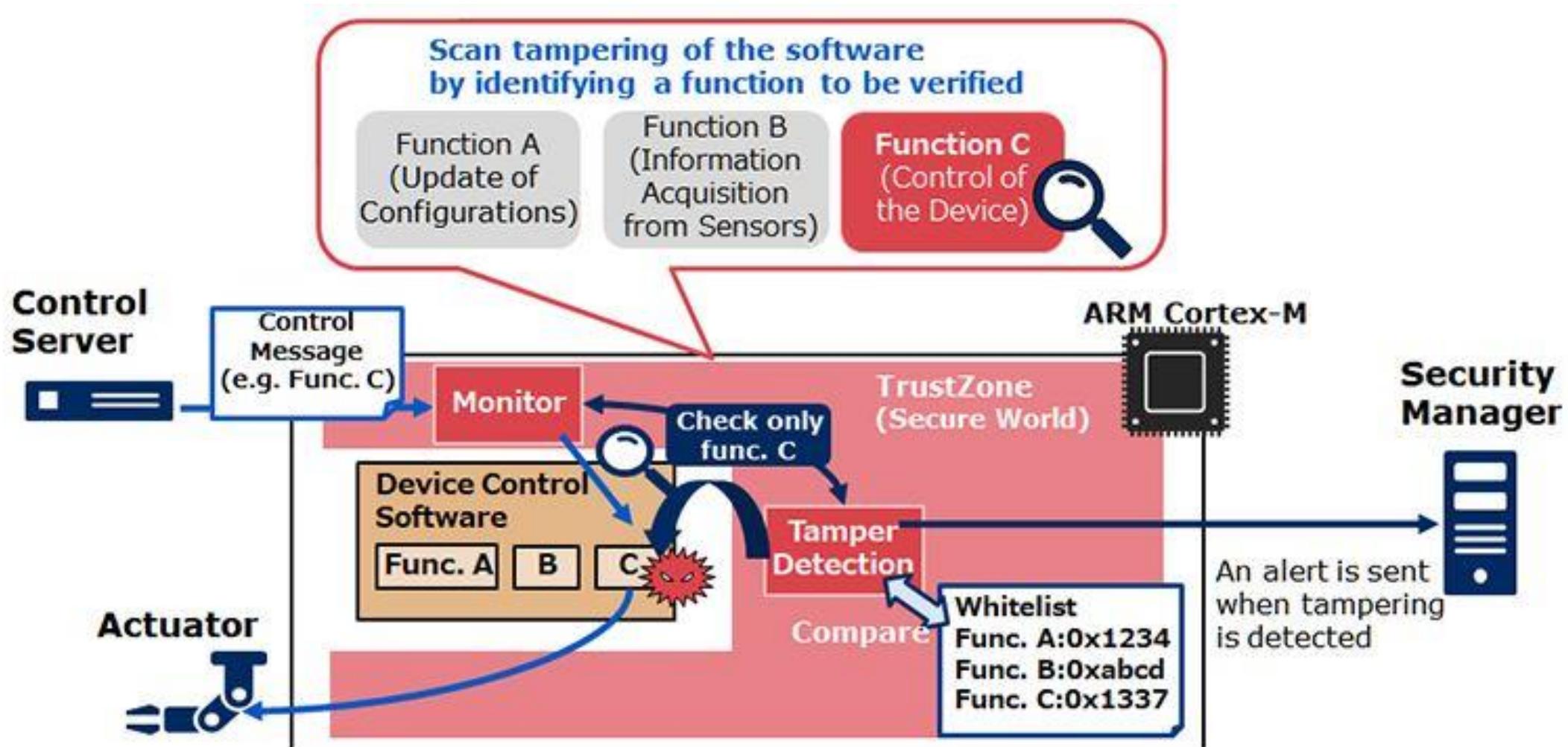
Anti-tampering Techniques

- First line of defence against the attacks on any IoT system.
- Detection, response and prevention of tampering attacks that may include physical as well as software attacks.

Tamper Detection

1. Ability of the device to sense that an active attempt to compromise the device integrity or the data associated with it is in progress.
2. Tamper detection is performed using a collection of sensors that detect change in the device temperature, input voltage variations, input frequency variations, X rays, Gamma rays etc.
3. Common tamper detection schemes scan the complete software for any irregularities, but for real-time operations, a segmented scanning scheme is implemented by NEC Technologies.

NEC's Tamper Detection Scheme



Tamper Response

1. In addition to indicating, does some actions after tampering such as
 - Total power shut down
 - Destroying of sensitive data
 - Activate a physical indicator

Tamper Prevention

- Using security fuses to prevent unauthorized access to the memory. ID authentication is performed when an access is attempted and if the authentication fails, then access is denied.
- Layout and data bus scrambling can be used to confuse an attacker by cleverly routing the data buses through a maze or puzzle.

Transport Layer Security

- TLS aims to provide security including privacy, integrity and authenticity through the use of cryptography between 2 or more computer applications.
- It runs in the application layer.
- Port 80 is used for unencrypted HTTP traffic while port 443 is used for TLS enabled HTTP traffic.
- First step is a TLS handshake between a client and server where both the parties agree on various parameters used to establish connection's security.

(From Wikipedia page of TLS)

Transport Layer Security

- The handshake begins when a client connects to a TLS enabled server requesting a secure connection and presents a list of supported ciphers and hash functions.
- Server picks the cipher and hash functions it also supports and notifies client.
- The server then provides identification in the form of a digital certificate. The certificate contains server name, trusted certificate authority that vouches for the authenticity of the certificate and the server's public encryption key.
- The client confirms the validity of the certificate before proceeding.

Transport Layer Security

- The client encrypts a random number with server's public key and send the result to the server (which only the server should be able to decrypt with its private key).
- Both parties then use the random number to generate a unique session key for subsequent encryption and decryption of messages.
-

Cryptography

- Cryptography methods can be divided into 3 categories
 1. Encryption
 2. Hashing
 3. Signing
- Symmetric key cryptography is an encryption method in which both the sender and receiver share the same key. Symmetric key ciphers are implemented either as block ciphers or stream ciphers. Ex. DES (Data Encryption Standard) and AES (Advanced Encryption Standard).
- Asymmetric or Public Key cryptography refers to a method in which 2 inter-related keys are generated – public key and private key.

Cryptography

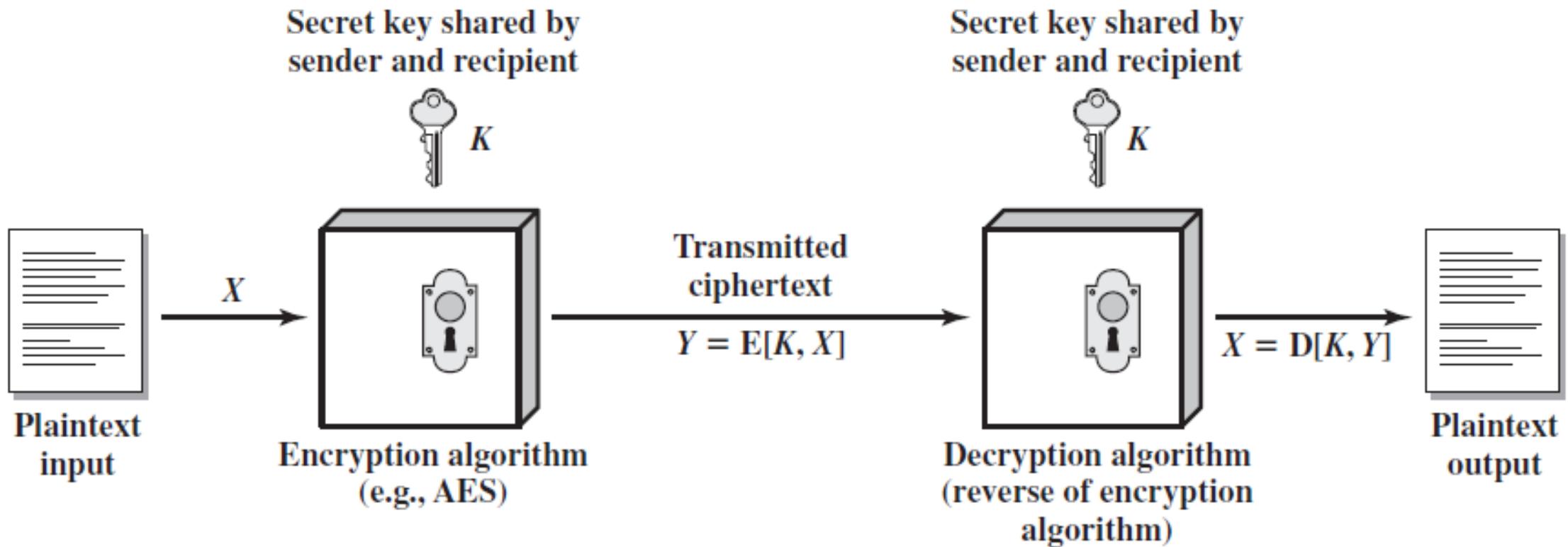
- The public key may be freely distributed while its paired private key remains secret. The public key is used for encryption while the private key is used for decryption.
- Examples are RSA and ECC algorithms.
- In signing or Digital Signature, private key is used for encryption while public key is used for decryption.

Block and Stream Ciphers

- Stream cipher – each bit of information stream is encrypted one at a time by XORing with one bit of cipher stream.
- Cipher stream is generated using Linear Feedback Shift Registers using random seed value which serves as the key.
- Block cipher – the information signal is divided into groups of bits called block and the whole block is encrypted at a time.
- An unvarying transformation is used in block cipher as opposed to random cipher stream in stream cipher.

Symmetric Key Encryption

- Typical flow of symmetric encryption is as shown



(From “Network Security Essentials – Applications and Standards” book by William Stallings)

Symmetric Key Encryption

- In symmetric key encryption, the algorithm is public and known to everyone, while the key is secret and known only to the authorized sender and receiver.
- The algorithm used must be very strong such that even if the attacker gets access to the ciphertext or number of ciphertexts together, he should not be able to decipher the plaintext or discover the key.
- Since encryption algorithms are not secret, several manufacturers can develop low cost chip implementations of these algorithms and hence symmetric key encryption is very widely used.

Symmetric Key Encryption

- Generally, all the encryption algorithms work on 2 general principles: substitution and transposition.
- In substitution, each element of plaintext (bit or letter) is mapped to another element while in transposition the elements are rearranged.
- Most encryption systems resort to multiple stages of substitutions and transpositions in such a way that all the operations are reversible (i.e. no information is lost).

Data Encryption Standard (DES)

- DES is the most widely used block cipher based symmetric encryption scheme.

Asymmetric Encryption

- 2 keys are used – public and private key
- Public key is known to everyone and private keys are different for sender as well as receiver
- Secret keys are generated at the sender and receiver end by some mathematical operations and both are compared. When both of the secret keys match, the secure link is established.
- In RSA algorithm, the sender (browser) sends the connection request along with its public key. The receiver (server) then encrypts the data with this key and send it back to the sender. The sender then decrypts the data with its private key.

Asymmetric Encryption – RSA Algorithm

>> Generating Public Key :

- Select two prime no's. Suppose $P = 53$ and $Q = 59$.
Now First part of the Public key : $n = P \times Q = 3127$.
- We also need a small exponent say e :
But e Must be
 - An integer.
 - Not be a factor of n .
 - $1 < e < \Phi(n)$ [$\Phi(n)$ is discussed below],
Let us now consider it to be equal to 3.
- Our Public Key is made of n and e

Asymmetric Encryption

>> Generating Private Key :

- We need to calculate $\Phi(n)$:

Such that $\Phi(n) = (P-1)(Q-1)$

so, $\Phi(n) = 3016$

- Now calculate Private Key, d :

$d = (k * \Phi(n) + 1) / e$ for some integer k

For $k = 2$, value of d is 2011.

Now we are ready with our – Public Key ($n = 3127$ and $e = 3$) and Private Key($d = 2011$)

Asymmetric Encryption

Now we will encrypt "HI" :

- Convert letters to numbers : H = 8 and I = 9
- Thus **Encrypted Data** $c = 89^e \text{ mod } n$.

Thus our Encrypted Data comes out to be 1394

Now we will decrypt **1394** :

- **Decrypted Data** = $c^d \text{ mod } n$.

Thus our Encrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

Diffie Hellman Algorithm

Alice

Bob

Public Keys available = P, G Public Keys available = P, G

Private Key Selected = a

Private Key Selected = b

Key generated =

$$x = G^a \text{mod} P$$

Key generated =

$$y = G^b \text{mod} P$$

Exchange of generated keys takes place

Key received = y

key received = x

Generated Secret Key =

$$k_a = y^a \text{mod} P$$

Generated Secret Key =

$$k_b = x^b \text{mod} P$$

(<https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/>) Algebraically, it can be shown that

$$k_a = k_b$$

Message Authentication

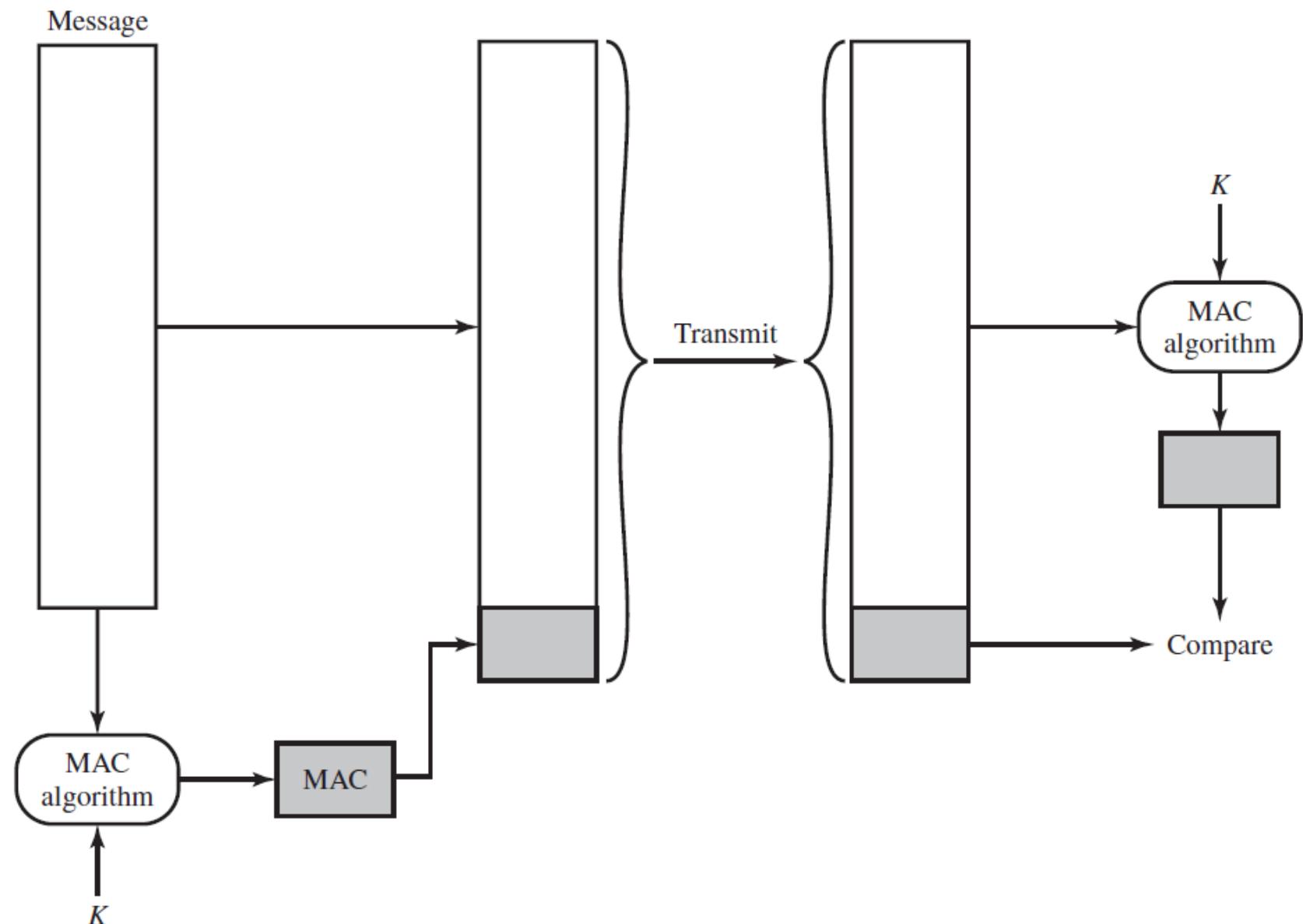
- Data encryption protects against passive attacks such as eavesdropping. Message authentication must be used to protect against active attacks such as falsification or modification of data.
- A message is said to be authentic when it is genuine and comes from a trusted source.
- In message authentication, the communicating parties verify that the received messages are authentic.

Message Authentication Code (MAC)

- One way of authentication is using a message authentication code (MAC).
- In this method, the sender and receiver share a secret key (K). At the start of transmission, the sender calculates a MAC as a mathematical function of the message (M) and the key (K) i.e. $\text{MAC} = F(M, K)$.
- This MAC is appended to the original message and transmitted. The receiver performs the same mathematical operation on the message using the same secret key and generates a new MAC.
- The comparison between the MACs at the sender and receiver verifies the authenticity of the message.

Message Authentication Code (MAC)

- Process of message authentication using MAC is shown in the figure.



Message Authentication Code (MAC)

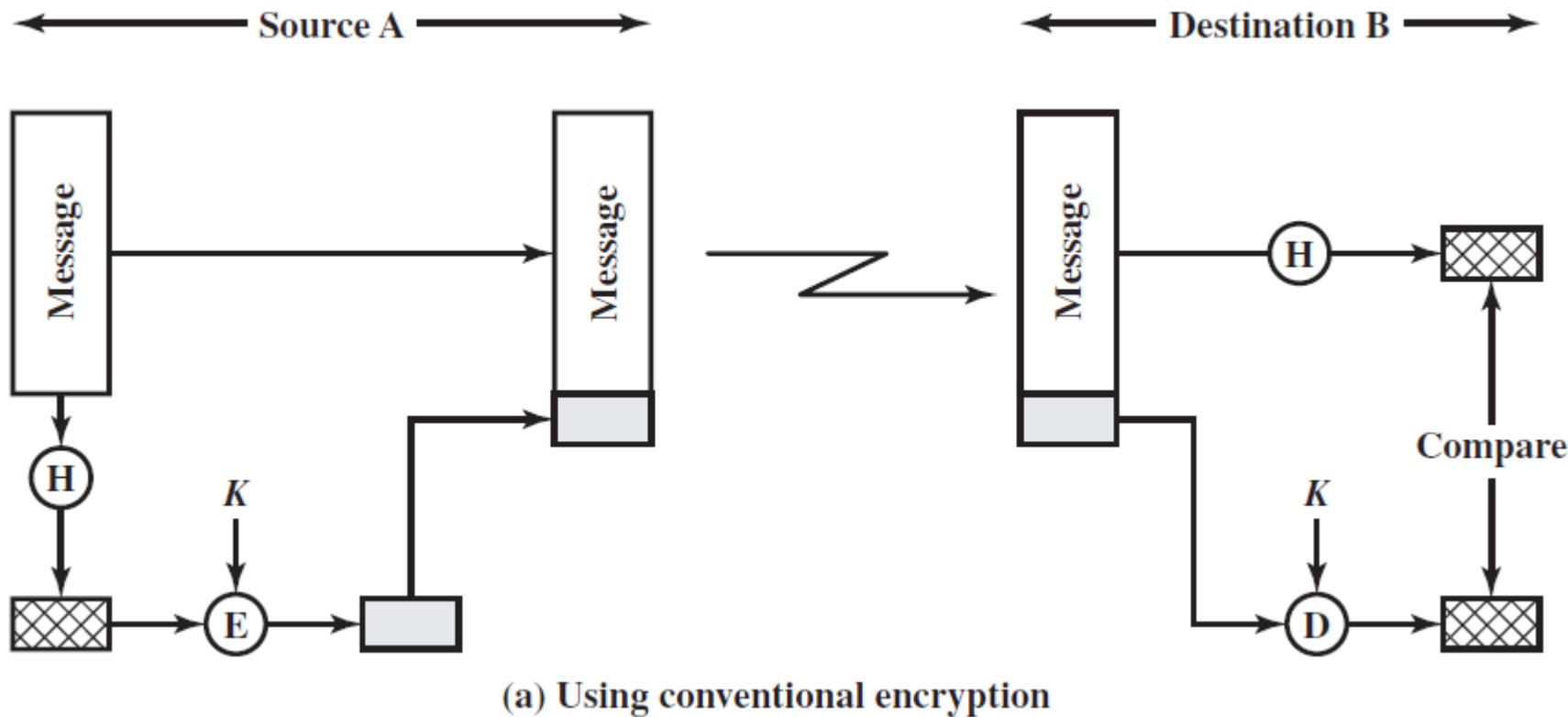
- In this method, the message may or may not be encrypted but generally message encryption and authentication are performed together.
- As the MAC is generated by a secret key, an attacker cannot alter the MAC (even if he alters the message).
- Thus, if the message is altered by an attacker, the MAC generated at the receiver will be different and receiver will come to know that the message is not received from a legit sender.
- The process is same as encryption but the authentication algorithm need not be reversible.

Hashing

- Hashing converts any form of data into a unique string of texts by performing a mathematical operation.
- Difference between hashing and encryption is that an encrypted data can be decrypted using the same key, but hashed data cannot be recovered easily.
- Generating the data back from its hash is very difficult.
- Salting is the method of adding a random string to the data before hashing it and then storing the ‘salt value’ with the hash.

Hashing

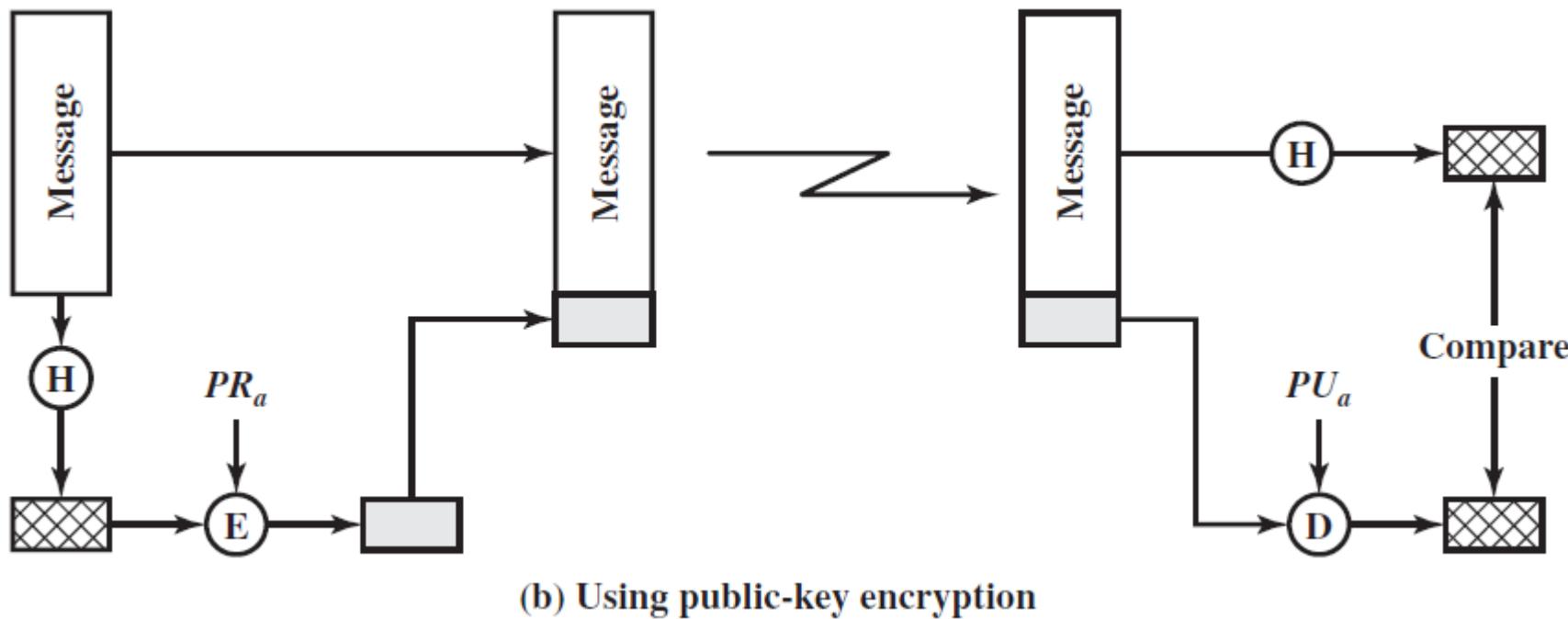
- Hashing using conventional encryption



(From “Network Security Essentials – Applications and Standards” book by William Stallings)

Hashing

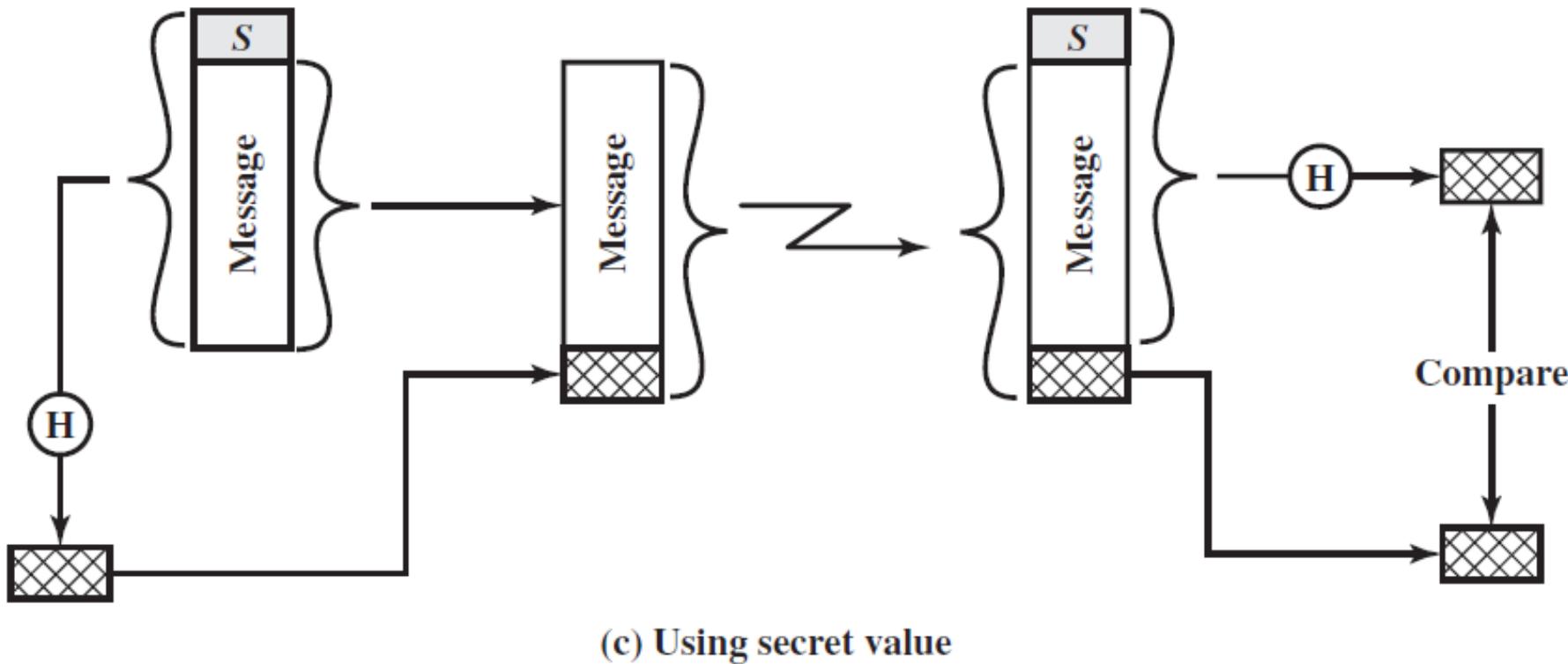
- Hashing using public key encryption



(From “Network Security Essentials – Applications and Standards” book by William Stallings)

Hashing

- Hashing using secret value (S) which is known beforehand to both the parties



(From “Network Security Essentials – Applications and Standards” book by William Stallings)

Authentication Methods in IoT

1. One-time One Cipher

- Dynamic variable cipher based on request-reply mechanism
- Pre-shared matrix is communicated between the parties.
- Parties generate random key co-ordinates which are transmitted over the network.
- Key is generated from these co-ordinates at the nodes.
- All messages are encrypted with device ID, time-stamp and key co-ordinates.

Authentication Methods in IoT

2. Mutual Authentication Method

- Sender and receiver authenticate each other by comparing their digital certificates.
- Certificate exchange is done through Transport Layer Security (TLS) protocol.

Other Authentication Schemes

1. Algebraic Eraser (AE)
 2. NTRU – public key generated by solving some linear equations
-
- Traditional encryption schemes
 1. AES – Advanced Encryption Standard
 2. RSA – Rivest Shamir Adleman
 3. DH – Diffie Hellman
 - IoT encryption scheme
 1. Elliptic Curve Cryptography (ECC)
 2. SHA-3 and SHA - 256 (Secure Hash Algorithm)
 3. Lightweight Cryptography

Lightweight Cryptography

- Cryptography techniques occupying less memory and consuming less power suitable for resource constrained devices.
- Small block size, small key size, small code.
- Hash functions like Quark, Marvin.
- Block ciphers like PRESENT, SPONGENT.

Comparison of some lightweight ciphers

Table 1. FELICS results for lightweight ciphers [6].

General info			AVR (8-bit)			MSP (16-bit)			ARM (32-bit)		
Name	Block	Key	Code	RAM	Time	Code	RAM	Time	Code	RAM	Time
Chaskey	128	128	770	84	1597	490	86	1351	178	80	614
SPECK	64	96	448	53	2829	328	48	1959	256	56	1003
SPECK	64	128	452	53	2917	332	48	2013	276	60	972
Chaskey-LTS	128	128	770	84	413	492	86	2064	178	80	790
SIMON	64	96	600	57	4269	460	56	2905	416	64	1335
SIMON	64	128	608	57	4445	468	56	3015	388	64	1453
LEA	128	128	906	80	4023	722	78	2814	520	112	1171
RECTANGLE	64	128	602	56	4381	480	54	2651	452	76	2432
RECTANGLE	64	80	606	56	4433	480	54	2651	452	76	2338
SPARX	64	128	662	51	4397	580	52	2261	654	72	2338
SPARX	128	128	1184	74	5478	1036	72	3057	1468	104	2935
RC5-20	64	128	1068	63	8812	532	60	15,925	372	64	1919
AES	128	128	1246	81	3408	1170	80	4497	1348	124	4044
HIGHT	64	128	636	56	6231	636	52	7117	670	100	5532
Fantomas	128	128	1712	76	9689	1920	78	3602	2184	184	4550
Robin	128	128	2530	108	7813	1942	80	4913	2188	184	6250

("Lightweight cryptography methods" by William Buchanan)

Preventing DoS attacks

1. Packet Logging

1. The transmitter keeps a log of transmitted packets i.e. time stamp, amount of data etc.
2. The receiver compares the received packet with the log-book of transmitter to detect whether it has been attacked or not.

2. IP Traceback – Retracing the IP address of malicious node and eliminating it.

1. Probabilistic Packet Marking – router marks the packet with its address.
2. Deterministic Packet Marking

3. A list of all legitimate IP addresses is maintained at edge node and all incoming packets are compared with the list.

4. Entropy based traffic classification using machine learning and deep learning.

(Wikipedia page of IP Traceback)

Classification of Security Issues in IoT

- Security issues in IoT can be classified as
 1. Low level security issues – affect physical and data-link layer
 2. Intermediate level issues – affect network and transport layer
 3. High level issues – affect application layer

(From 'IoT Security: Review, blockchain solutions and open challenges' by Minhaj Ahmad Khan)

Low level security issues

- Jamming
 - Deterioration of networks by emitting RF signals without following a specific protocol
 - Can severely affect the sending and receiving of data packets
- Low level Sybil attack
 - Fake nodes using forged MAC address depleting the network resources
 - Legitimate nodes may be denied access to resources
- Insecure physical interface
 - Attack on hardware and software of physical devices
- Sleep deprivation attack
 - Attacks causing sensor nodes to stay awake, leading to depletion of battery

Intermediate Level Security Issues

- Insecure neighbor discovery
 - Authorizing a malicious node to receive or transmit data
 - May lead to DoS attack
- Replay or duplication of packets
 - Malicious nodes send multiple duplicate packets causing difficulty in reassembly at receiver
 - May lead to depletion of receiver resources, buffer overflow and rebooting
- Buffer reservation attack
 - Malicious nodes send incomplete packets causing buffers at Rx to be permanently busy
 - Other legitimate packets may be discarded – type of DoS attack
- Routing attack
 - Data routed through malicious nodes, leading to leakage of information

Intermediate Level Security Issues

- Sybil attack
 - Fake nodes resulting in spamming, disseminating malware and phishing attacks.
- Session hijacking
 - Fake node may impersonate a victim node and establish a session with another node
 - Results in denial of service and whole session needs to be re-established

High Level Security Issues

- Insecure interfaces
 - The APIs used to access IoT devices and services may be attacked by malware.
- Insecure software/firmware
 - Codes written for IoT services in JSON, XML etc. can be accessed and tampered.
- CoAP security
 - CoAP has small header as compared to HTTP and hence less number of bits are used for encryption, making it easier to attack
 - Leads to DoS attack

Basics of Networking

- Computer networks are divided into 3 planes
 1. Data plane – devices which are responsible to forwarding the data
 2. Control plane – the protocols which populate the routing tables of devices in data plane
 3. Management plane – software services (like SNMP) which remotely monitors and configure the control functionality.
- Network policy is defined in the management plane, enforced by the control plane and executed by the data plane.
- In traditional IP networks, the control and data planes are coupled together and embedded into the same networking device.

Software Defined Networking

- Current IP networks are complex and hard to manage. Its difficult to configure a network, then reconfigure it according to the needs and faults.
- Each network device needs to be configured separately by network operator based on vendor specific commands.
- Devices are rigid – routers will behave only as routers, switches as switches, firewalls as firewalls etc.
- Due to this rigidness, it takes almost 10 years for a new routing protocol to be designed, tested and deployed.

(From the paper “Software defined networking – a comprehensive survey” by Diego Kreutz et. al)

Software Defined Networking

- Managing such rigid network is also difficult and several additional middleboxes such as firewalls, intrusion detection system, load balancers, deep packet inspection engines must be installed for network management.
- This increases the overall complexity of the network.

Software Defined Networking

- SDN – a network architecture where forwarding state in the data plane device is controlled by a *remote* control plane which is *decoupled* from the data plane
- SDN separates the control plane from the data plane, thus increasing the flexibility of the network.
- SDN – virtualizing networking devices like routers, switches etc. and making them controllable using a software.
-

SDN Characteristics

- SDN separates control plane and data plane by use of well-defined application programming interfaces (APIs) and removing control functionalities from network switches.
- Network switches become simple forwarding devices (data plane) controlled by centralized network controllers (control plane) running APIs.
- Control planes are *logically centralized but physically distributed.*
- Controllers install a set of rules in switches which can make the switch behave as a router, switch, firewall etc. dynamically.
- Thus, SDN devices are programmable and flexible.

SDN Characteristics

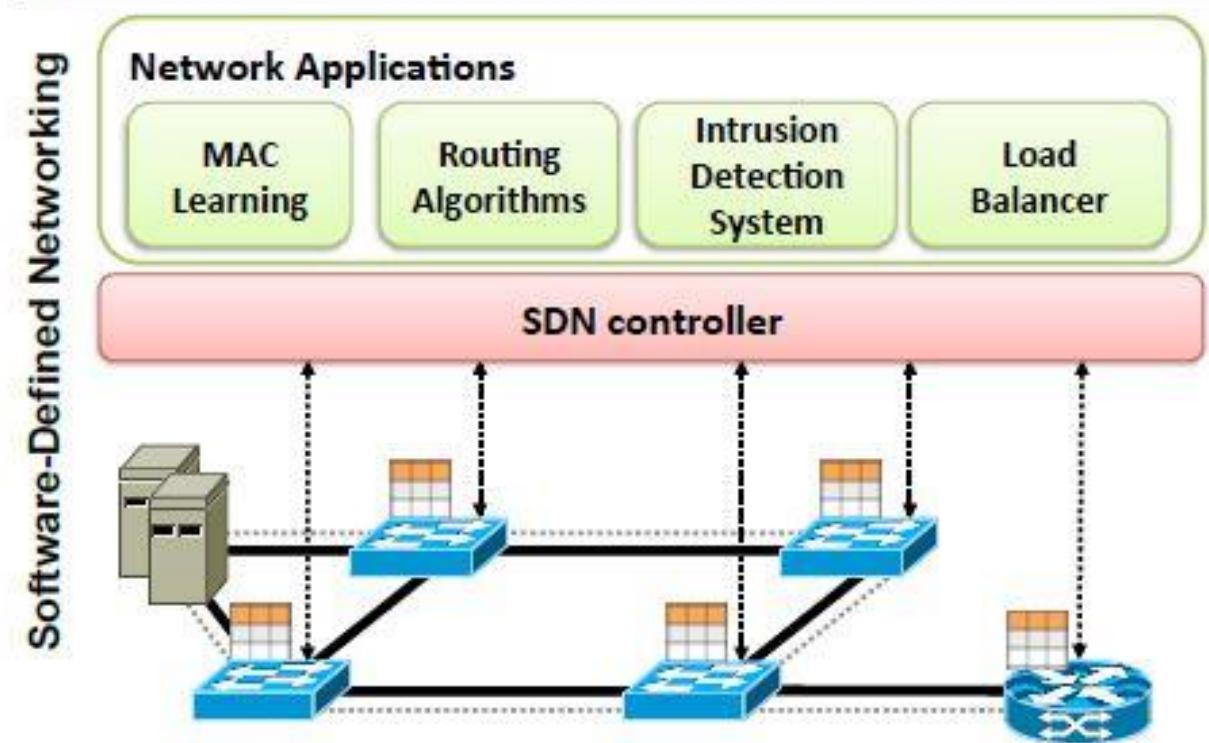
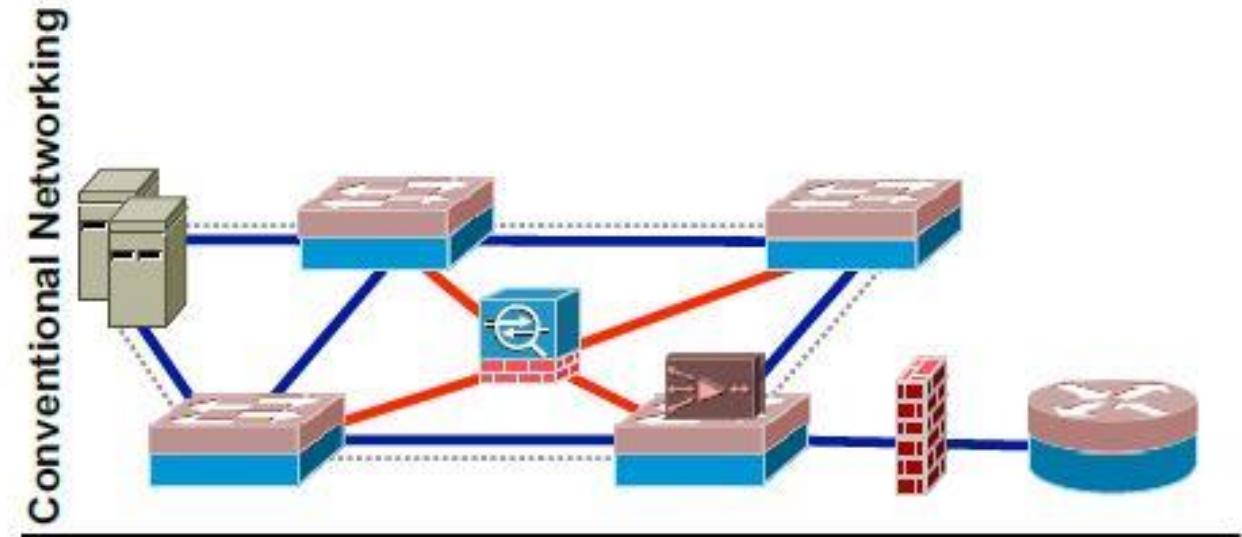
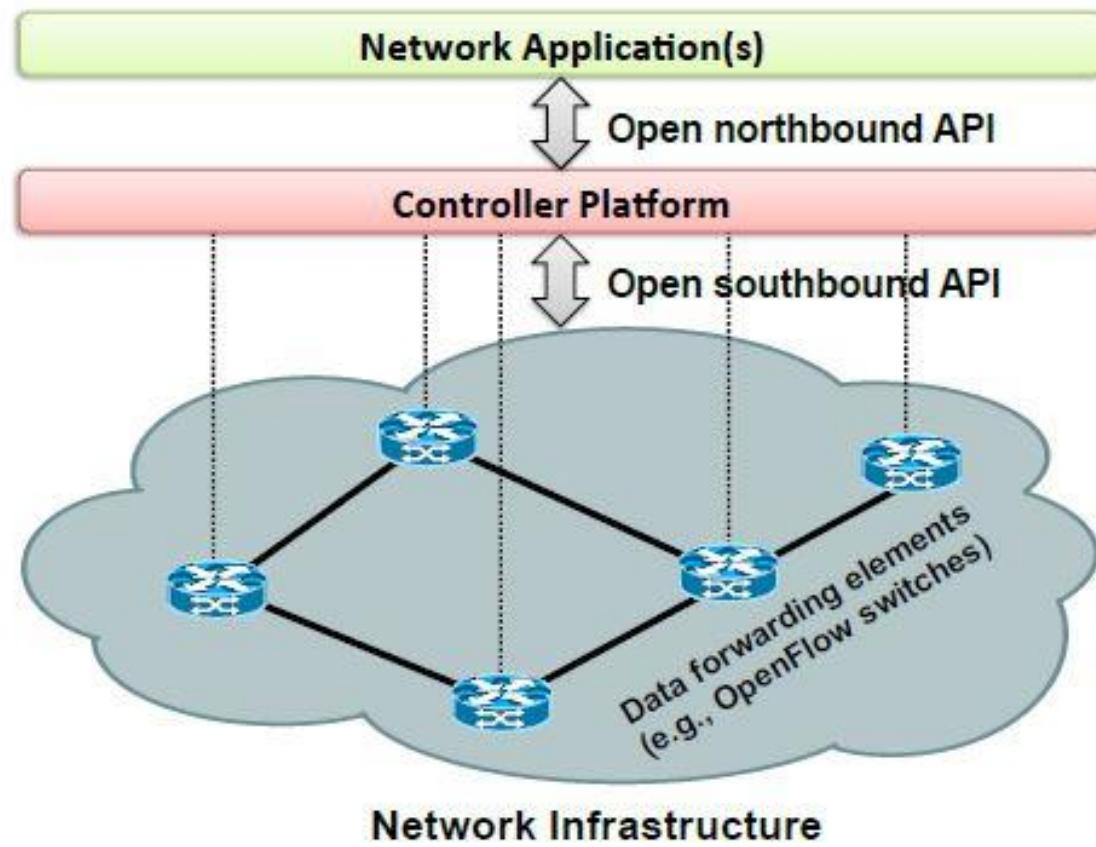
- SDN controller directly controls the state in the data plane using APIs – the most widely used API is *OpenFlow*.
- Data plane consists of simple OpenFlow switches. Each switch has one or more tables of packet handling rules (known as flow table).
- These rules decide the action performed on the data packets – forwarding, dropping or modifying.
- The rules are installed by the SDN controller and by changing the rules the controller can program the OpenFlow switch to behave like a router, firewall, gateway etc.
- Thus, *the function of the switch is defined by the software installed from the controller – hence the term Software Defined Networking.*

SDN Characteristics

- SDN controller is also known as Network Operating System (NOS) – its functionality is same as traditional operating system.
- NOS is a software platform which runs on a commodity server technology and handles the control logic.
- The network is programmable through software applications running on top of NOS that interacts with underlying data plane devices.
- The SDN controller has the global knowledge of the network.

SDN Characteristics

- Whether the forwarding device will behave like a router or firewall or gateway is decided by the applications running on the SDN controller.
- Integrating and managing various applications becomes very easy due to logical centralization.
- All the applications can take the advantage of global network information present at the controller. This helps in better decision making.
- Applications can take action from any part of the network.

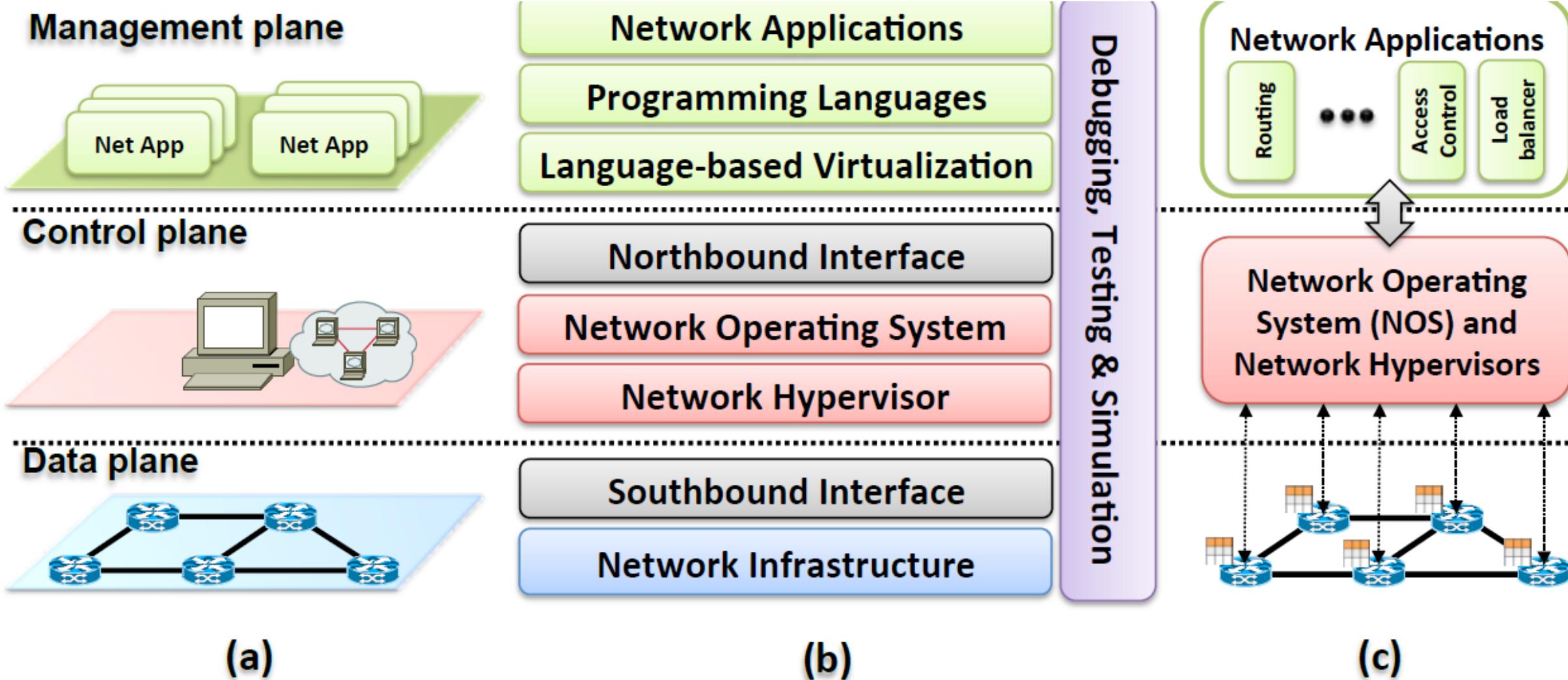


SDN Components

A typical SDN architecture has the following components (as seen in previous slide):

1. Forwarding devices
 1. Simple devices in data plane performing data forwarding functions.
 2. Have well-defined instruction sets and *built on open-source platforms*.
 3. Programmed by the control plane elements
2. Southbound API
 1. Defines and installs the instruction set of forwarding devices
 2. Defines the communication protocols between forwarding devices and control plane elements
3. Northbound API
 1. An interface for application developers to design and deploy apps running on SDN controller

SDN Layered Architecture



Software-Defined Networks in (a) planes, (b) layers, and (c) system design architecture

SDN Layered Architecture

At

Network Function Virtualization

- Moving network functions such as encryption, routing, firewall etc. from hardware to software (virtual servers).
- Instead of installing hardware at the nodes, virtual machines are installed which consist of switches, storage and servers. Thus instead of using proprietary hardware for each network function, generic hardware i.e. virtual machine is used for every network function.
- The hardware of VMs is programmable.
- These network functions are packaged as virtual machines. If a service provider wants to add a new network function, he can simply add a new virtual machine instead of another hardware.

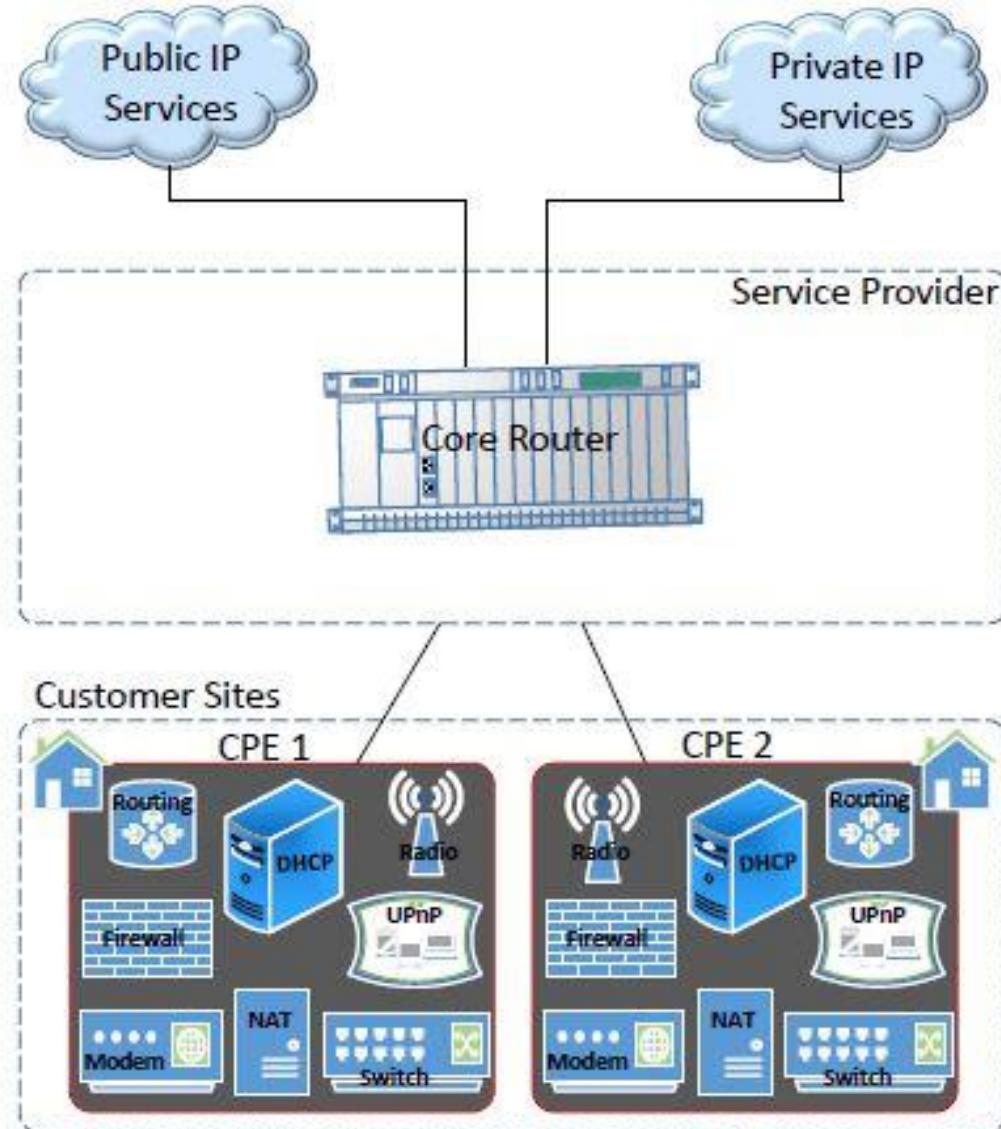


Fig. 1. Traditional CPE Implementations

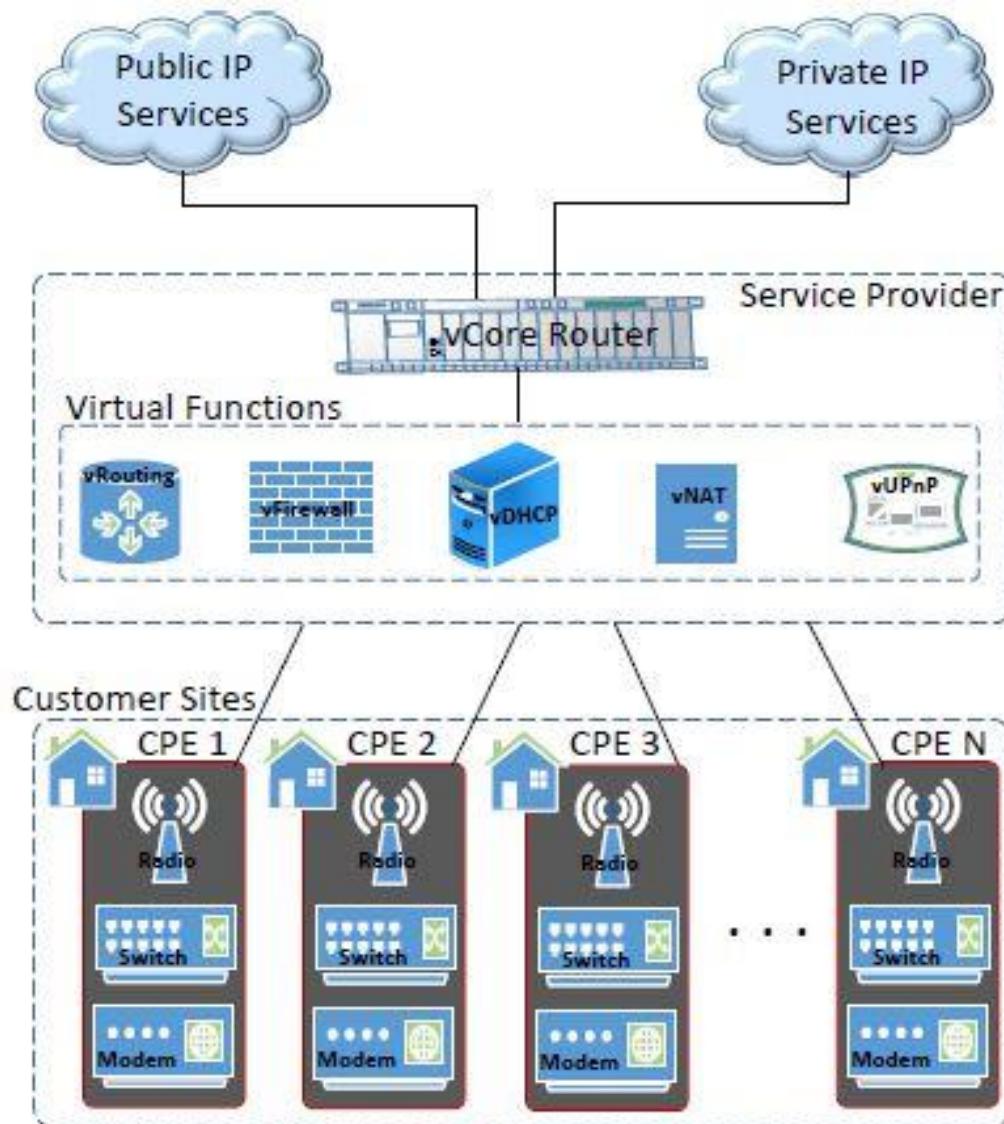


Fig. 2. Possible CPE Implementation with NFV

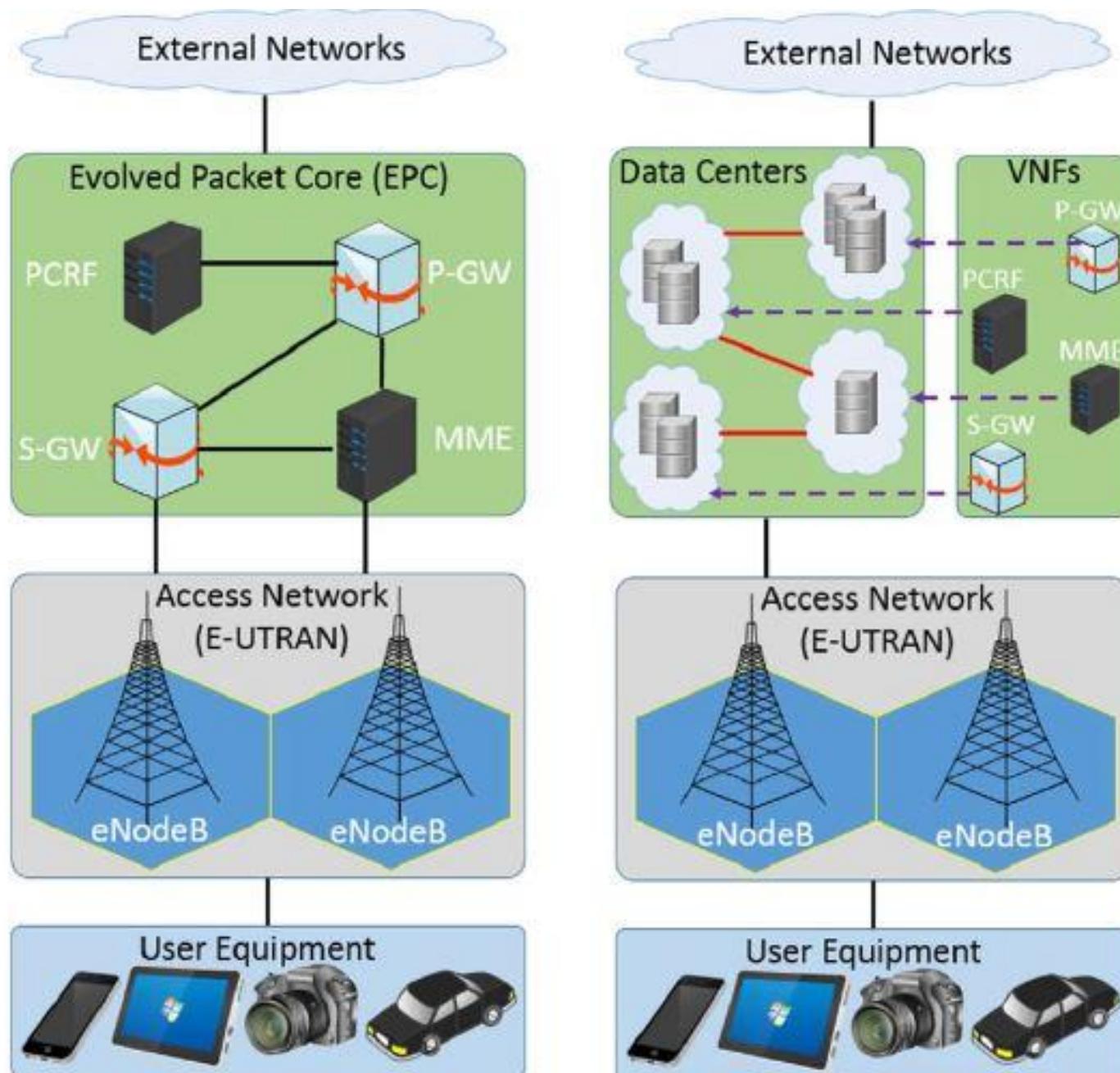
(NFV : State of the art and research challenges by Rashid Mijumbi)

Virtualizing CPE

- In traditional networks, all network functions such as firewall, DHCP, switching, routing etc. needs to be provided by separate physical devices installed at every customer premises equipment (CPE).
- Thus, for any modification of functions, a technician from ISP must come to CPE and implement the change required. This increases OPEX.
- In case of NFV, some functions are transferred to the data centers of the ISP which can be modified easily, thus simplifying the CPE as well.

Virtualizing EPC

- Evolved Packet Core (EPC) is the core network of LTE.
- EPC performs network functions such as subscriber tracking, mobility management and session management. Thus EPC is the backbone of LTE system.
- Traditional networks require separate Physical device for each function.

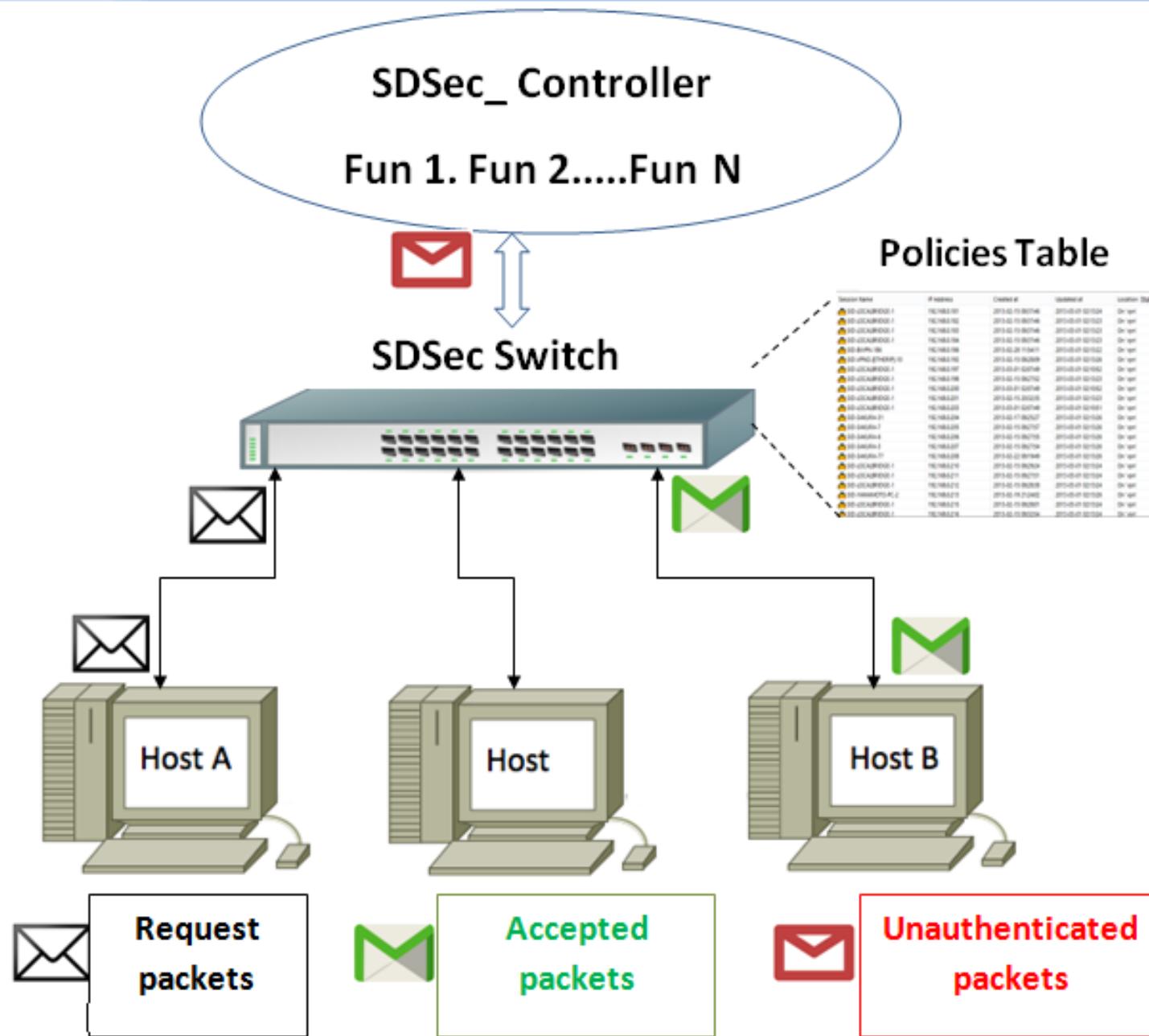


Virtualizing EPC

- Any new service demanded by customer would involve lot of OPEX for service provider in case of traditional networks.
- In NFV, some or all functions of EPC are transferred to cloud servers and can be managed remotely through software. Thus, OPEX for the service provider is reduced and it can provide better services.
- The resources allocated to each network function can also be modified easily and also functions can be upgraded through software to provide new services.

Software Defined Security

- Centralized security using SDN and NFV. Security plane is decoupled from data and control plane.
- Security functions are virtualized and all the network traffic information is available at the single node.
- There is no specific security device in any of the edge node. The policies exist within the SDN controller and are propagated across the nodes as and when required.



(“SDSecurity – A software defined security experimental framework” by Ala Darabseh)

Energy Harvesting in IoT

- Powering up of IoT devices is a challenge because
 1. Tremendous number of devices
 2. Locations are remote
 3. Expected lifetime is very high
- Techniques available for powering IoT devices
 1. Tethered power supply – normal power supply taken from main line
 2. Energy storage – rechargeable batteries attached to IoT devices
 3. Wireless Power Transfer
 4. Energy Harvesting

Wireless Power Transfer

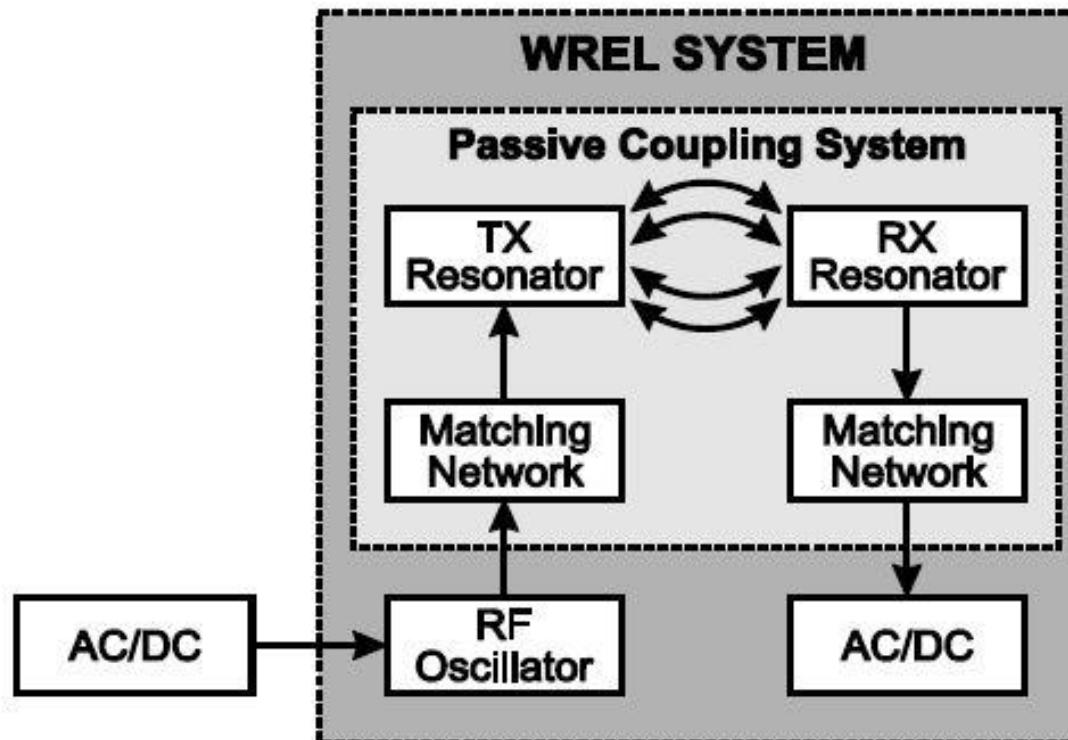
- Different techniques of wireless power transfer are
 1. Inductive Coupling
 2. Power transfer through EM waves – SWIPT
 3. Magnetic Resonance Coupling
- Inductive Coupling – works on same principle as a transformer
- Implemented in wireless charging mobiles by Samsung
- Range limited to few centimeters
- Only used as a replacement of power cords.

Power Transfer Through EM Waves

- Passive RFID tags extract power from the EM waves transmitted by the reader
- SWIPT – Simultaneous Wireless Information and Power Transfer
- Both information and power are transmitted simultaneously over the same EM wave.
- Omnidirectional and Unidirectional antennas are used for power transfer.
- Omnidirectional antenna has lower range but can power up many devices simultaneously.

Magnetic Resonance Coupling

- Uses a pair of coupled coils with additional capacitance which makes transmitter and receiver have same resonant frequency.
- Due to resonance effect, efficient power transfer over a considerable distance is achieved.



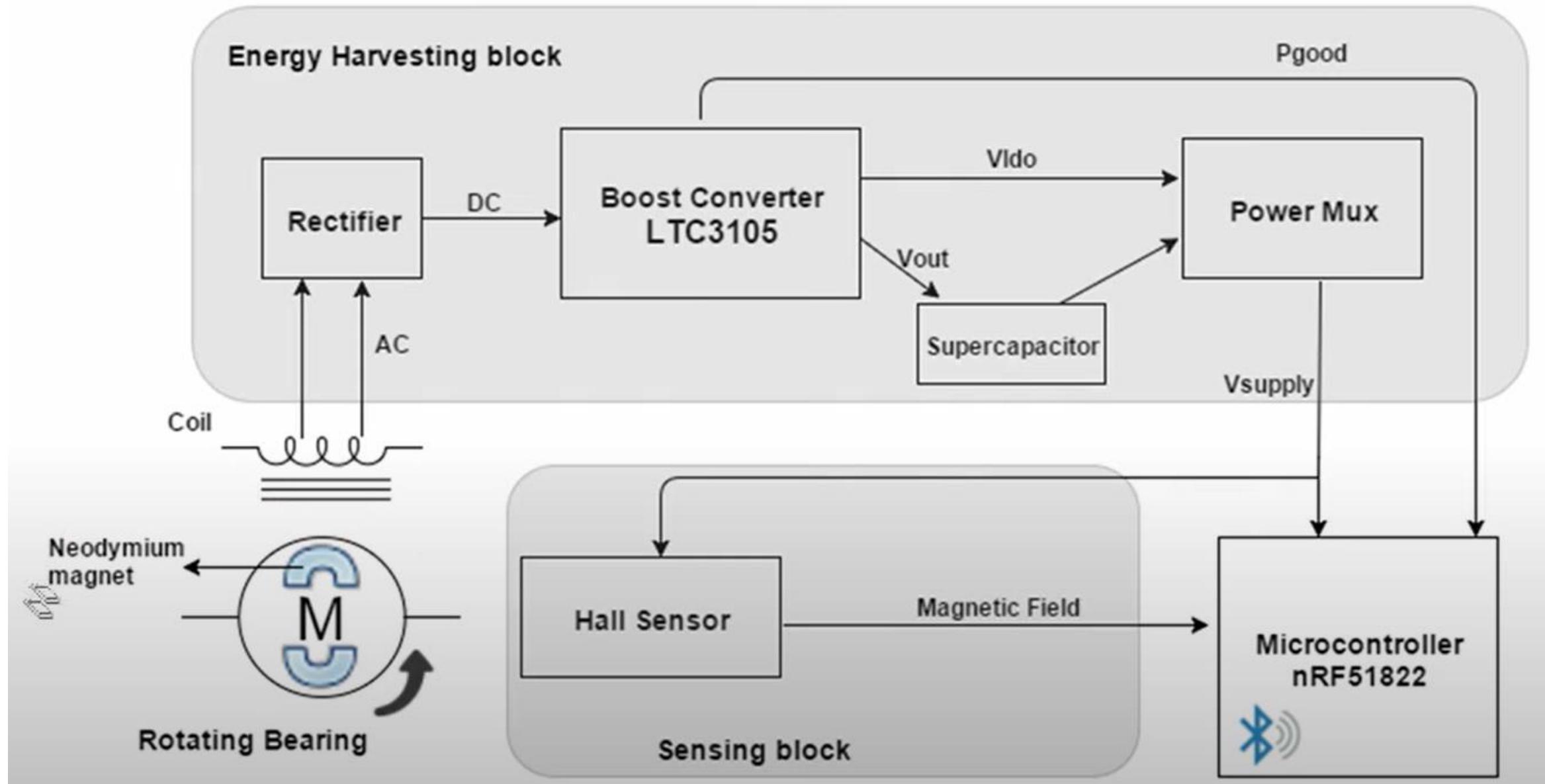
(Magnetic Resonance Wireless Power Transfer by Marco Dionigi)

Energy Harvesting

- Refers to extraction of energy from natural and artificial environment and converting it to electricity.
- Generally transducers are used to achieve this effect.
- Following 4 types of ambient energy sources are used for EH
 - 1. Radiant (EM)
 - 2. Mechanical
 - 3. Thermal
 - 4. Magnetic

Energy	Transducer	Possible application scenario
Radiant	Photovoltaic	Solar-powered wireless sensor node [74, 81]
		Indoor light-powered wireless sensor node [19, 120]
	Radio-frequency	Wi-Fi signal-powered digital temperature meter [108]
		Radio signal-powered wireless sensor node [75]
Mechanical	Piezoelectric	Vibration-powered wireless structural health monitor [70]
		Vibration-powered wireless sensor node [99]
	Electromagnetic	Wind-powered wireless sensor node [74]
		Vibration-powered battery charging [100]
Thermal	Thermoelectric	Human body heat-powered wearable sensor [55]
		Thermal gradient-powered wireless sensor node [28, 94, 111]
Magnetic	Coil	Current transformer-powered wireless sensor node [86]

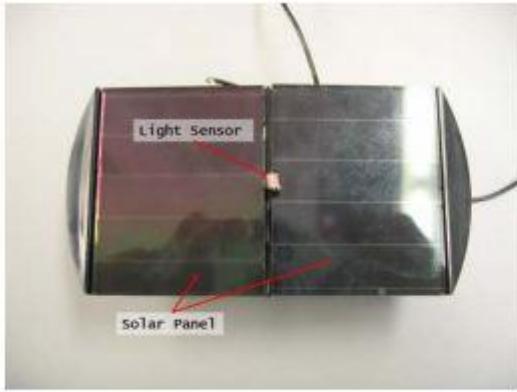
Energy Harvesting Examples



- Energy harvesting example for a magnetic field monitoring system.
- The rotating magnet induces an EMF in the coil which is then converted into a DC voltage using a bridge rectifier.
- This DC voltage which ranges in mV is then applied to boost converter module which boosts it to 3.3 V.
- The output voltage of boost converter (V_{out}) is stored in a super-capacitor.
- The low dropout regulator (V_{LDO}) output of boost converter is given to power mux which acts as an initial power supply to the microcontroller.

- V_{ido} is low power output and thus the initial supply to the microcontroller (V_{supply}) is low and it operates only the sensing module.
- High power operations such as data processing and communication are not carried out till now i.e. microcontroller is in *sleep mode*.
- After some time as more and more energy is harvested, the output value of boost converter rises and reaches to 3.3 V. At this point, the third output of boost converter i.e. P_{good} signal is turned on.
- This signal tells the microcontroller that the power is now good enough to turn on the high power operations.
- Now the computation and communication of data starts.

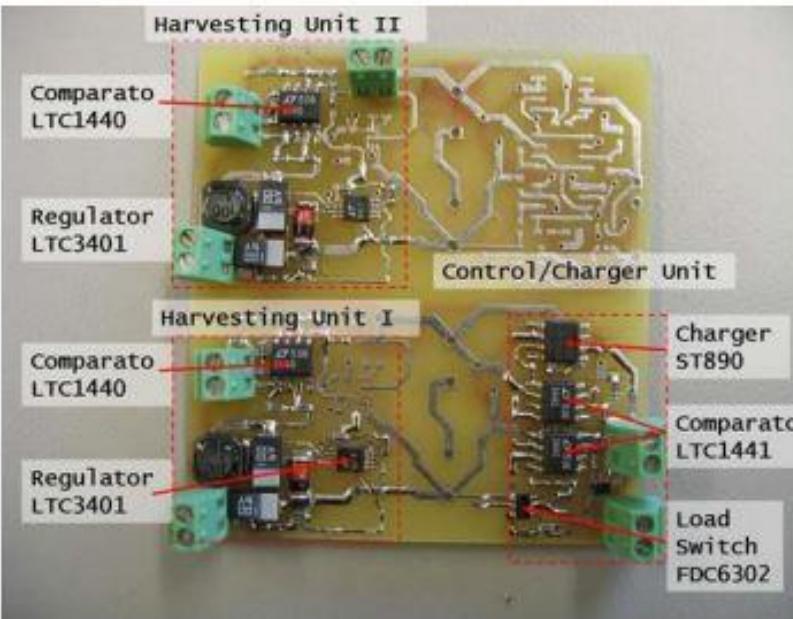
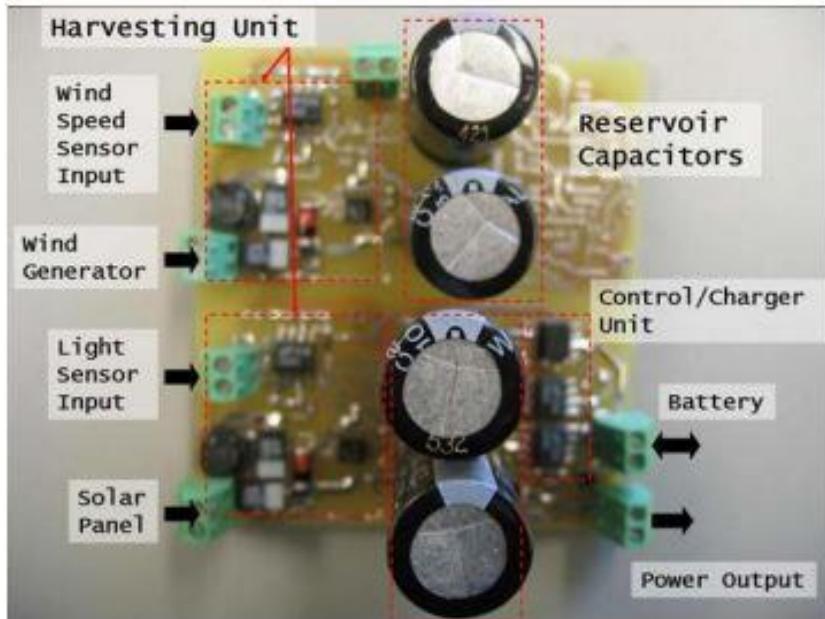
Solar and Wind Energy Harvesting Implementation



(a) Solar Panel with Light Sensor

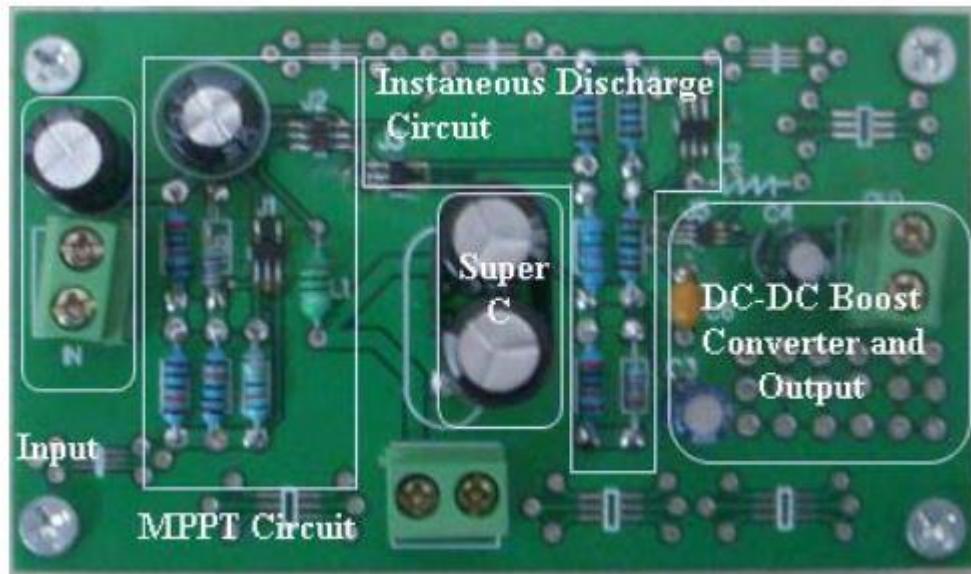
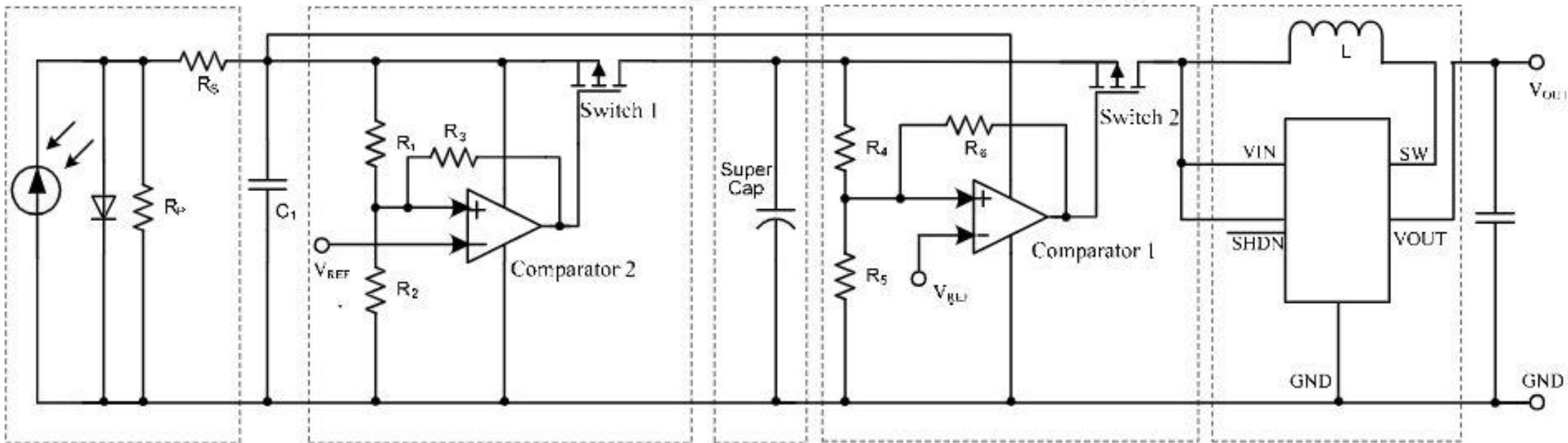


(b) Wind Generator with Rotor Speed Sensor



(AmbiMax: Autonomous energy harvesting platform for multisupply wireless sensor nodes by Park. C)

Light Energy Transducer MPPT Circuit Storage Element Instantaneous Discharging Circuit DC-DC Boost Converter



(Indoor Light Energy Harvesting System for Energy-aware Wireless Sensor Node)

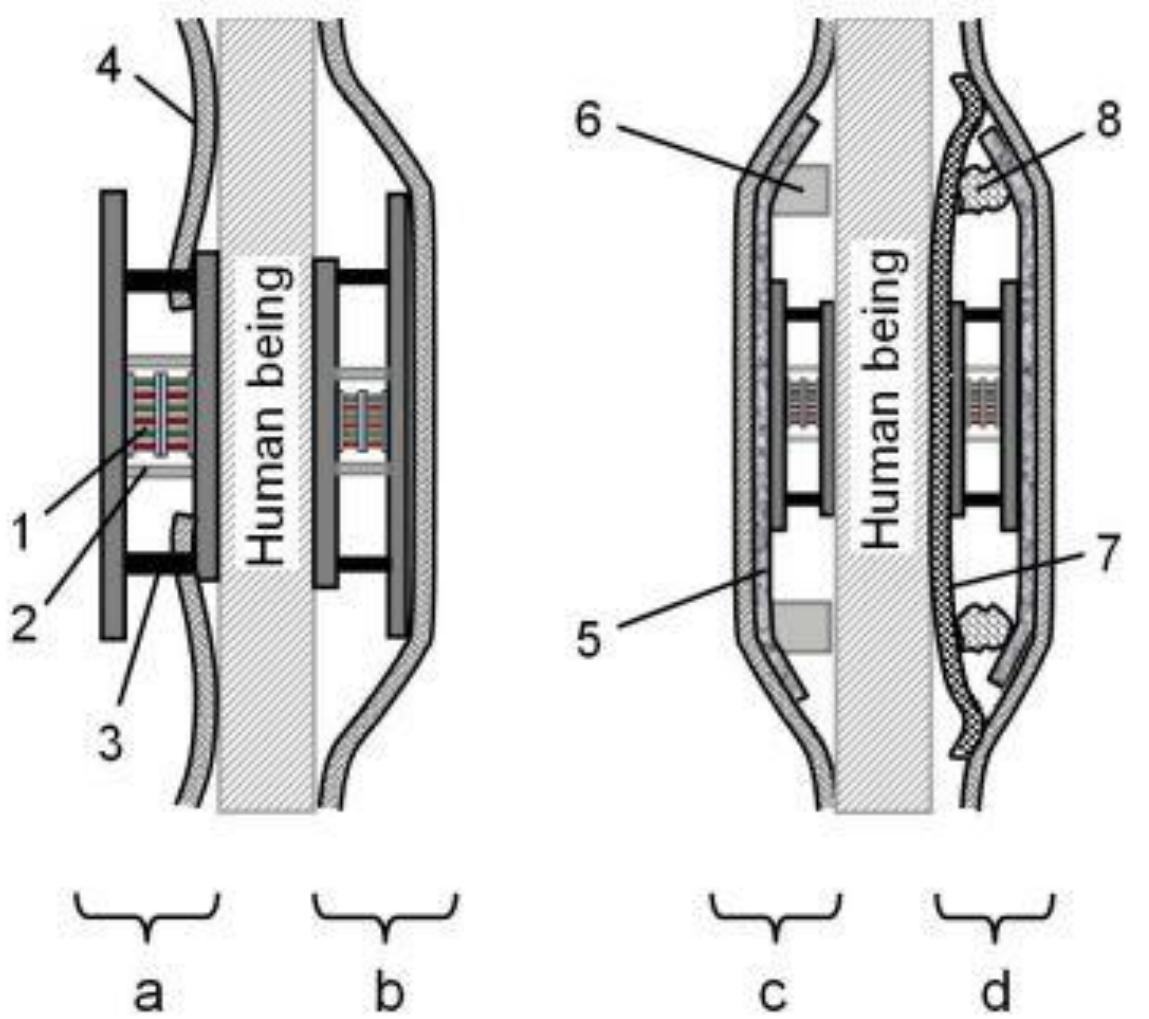


Fig. 8. (a)-(d) Four ways to integrate TEH #2 in garments (not in scale). Numbers denote: (1) thermopile, (2) polyethylene encapsulation, (3) thermally insulating spacers, (4) textile, (5) a carbon fabric heat-spreading layer, (6) neoprene foam spacers, (7) an additional textile layer, and (8) cotton-based spacers. (a) TEH passes through the hole made in textile. (b) TEH is placed under the textile layer. (c) Heat-spreading layer is glued to the cold plate and to the cotton, while foam spacers enable keeping some distance between the heat-spreading layer and skin. (d) Additional layer of cotton is glued to the TEH and sewed inside of clothing.

Thermocouple



(Thermoelectric Energy Harvesting of Human Body Heat for Wearable Sensors by Vladimir Leonov)

Challenges in Energy Harvesting

- The available energy sources are
 1. Uncontrollable
 2. Unpredictable
 3. Low power densities

Source	Harvestable power density ($\mu\text{W}/\text{cm}^2$)
Light	
Solar	100,000–15,000
Artificial	10–100
Vibration and motion	
Human body	4
Industrial	100–200
Thermal gradient	
Human body	30
Industrial	1,000–10,000
Radio-frequency	
Cell tower	0.1

Recent IoT startups

Start up	Area of work
FogHorn Systems	AI and ML on edge devices
Myriota	Low-cost, low-power connectivity for remote IoT devices
Ordr	AI based IoT security
Phizzle	Software stack to improve the power efficiency of edge devices
Roambee	Real-time shipment tracking and asset monitoring
Seeq	Predictive Maintenance
Skylo	Large scale IoT connectivity using NB-IoT via satellites
Augury	Predictive maintenance using wireless sensors and AI
Circonus	IoT data analytics using ML
Claroty	IoT security – threat detection and network monitoring

Application Programming Interface (API)

- API is a set of definitions and protocols used to build and integrate an app.
- API acts as a mediator between the client and web service provider and helps the user to get the desired resources (HTML, MXL or JSON document).
- API lets your product or service to communicate with another product or service without having to know how they are implemented. It helps simplify how developer integrates a new app into an existing architecture.

Time and Clock Synchronization in IoT

- IoT system consists of sensor nodes which may be distributed geographically but are connected to the same server.
- Each of these sensor nodes run on their own clock (called system clock) and timestamp their data accordingly.
- But, it may happen that the clocks of different sensor nodes are not synchronized with one another and thus the timestamps are not uniform.
- This happens when sensor nodes fall in different time zones or are geographically far apart from one another.

Time and Clock Synchronization in IoT

- It may also happen that sensor nodes placed in proximity to one another uses clocks of separate rate i.e. one sensor node working on 2 GHz processor while another working on 3 GHz processor.
- Relative difference in the clock values of 2 sensor nodes is called clock skew.
- Relative difference in the clock frequencies of 2 sensor nodes is called clock drift.
- Both clock skew and drift causes asynchronization between different sensor nodes which may lead to different timestamps of data, causing confusion and incorrect decision making at the server.

Time and Clock Synchronization in IoT

- Suppose 2 clients (X and Y) are trying to book a airline ticket from the same server.
- The ticket remaining is the last ticket and X, Y and airline server are located in different time zones.
- X sends a request first with local time stamp 8:30 PM to the server and get the ticket booked. Y sends a request after some time but the local time-stamp is 8:20 PM and as there is no ticket available, gets a denial message.
- Now if someone checks the log of airline server, it will appear that a request at 8:30 got the confirmed ticket while the one at 8:20 did not. This will create confusion.

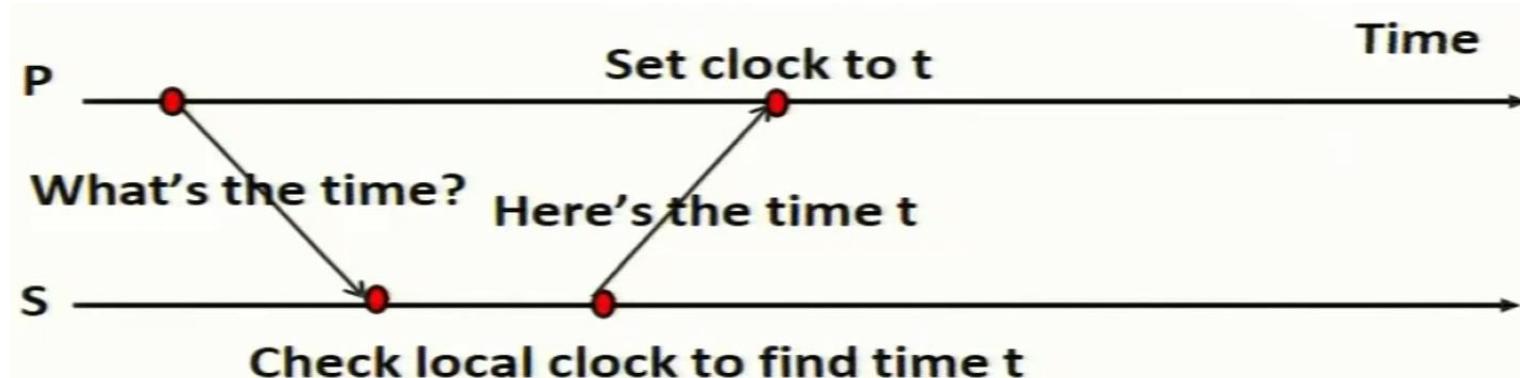
External and Internal Time Synchronization in IoT

- In external synchronization, each process's clock inside a sensor node ($C(i)$) is kept within a tolerable or acceptable limit (D) w.r.t to an external clock (S). This external clock can be connected to UTC (Universal Clock Time) or an atomic clock.
- For external synchronization $|C(i) - S| < D$, for all times.
- In internal synchronization, the clocks of 2 processes within the same sensor node are kept within a tolerable or acceptable limit i.e.
$$|C(i) - C(j)| < D$$
- There's no need of an external clock for internal synchronization.

https://www.youtube.com/watch?v=kDro_razhmw&t=285s&ab_channel=IITKanpurJuly2018

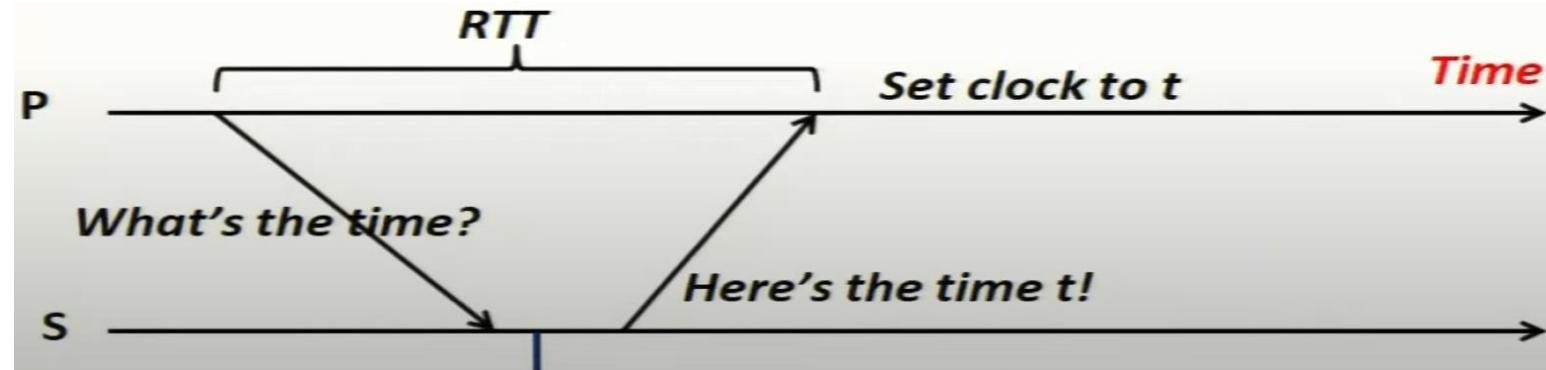
External and Internal Time Synchronization in IoT

- External synchronization happens as shown
- The process (P) in the client asks the external standard server (S) about the time and then sets its clock according to the response received from the server.
- But, in this method few other timings need to be included to calculate the actual time ‘t’:
 1. Latency of messages between the client and server (L1)
 2. Latency of messages between the server and client (L2)
 3. Round trip time elapsed between the request sent and response received at P (RTT)



Christian's Algorithm

- All these timings must be included to exactly synchronize the time between client and server.



- The latency L_1 and L_2 depends upon the protocols being used for communication like TCP/IP which have their own overheads.
- The actual time at P when it receives the response is between $[t+L_2, t+RTT-L_1]$
- Now, P sets its time halfway through this interval to: $t+(RTT+L_2 - L_1)/2$

Appendix

- Prof Rajiv Misra's course on NPTEL
- Lecture 5 – ML on Edge
- Lecture 6 – ML based Image Classifier on IoT Edge ()
- Lecture 7 – Intro to docker containers and Kubernetes