# System Design

Topics

**1. System Design**

- System design is the process of defining / designing the architecture, interfaces & data for a system that satisfies specific business requirements through coherent & efficient systems.
- A good system design requires us to think about everything, from infrastructure all the way down to the data & how it's stored.

**Why is system design so important?**

- System design helps us define a solution that meets the business requirements. It's one of the earliest decisions we can make when building a system.
- Often, it's essential to think from a high level as these decisions are very difficult to correct later. It also makes it easier to reason about & manage architectural changes as the system evolves.

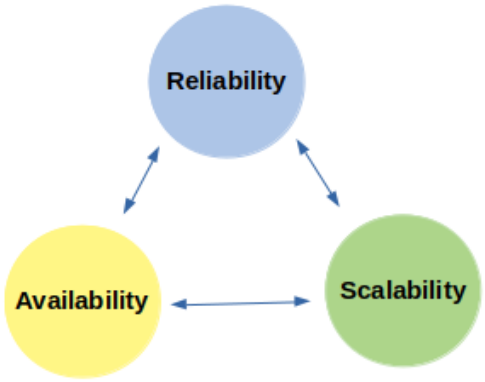
**Approaching a Design problem**

- When we're given a System design problem, we should approach it in a planned manner.
- Initially, the problem may look huge & one can easily get confused on how to start solving it & moreover, there is no fixed solution while you're designing a system.

**Step 1:** Breaking down the problem into small components (these components can be services or features)
**Step 2:** Communicating your ideas (design clearly on the whiteboard with flowcharts & diagrams on how to tackle the problem of scalability, DB design etc.)
**Step 3:** Assumptions that make sense (Make some reasonable assumptions while designing the system)
e.g.,

- Number of requests the system will be processing per day.
- Number of database calls made in a month.
- Efficiency rate of our caching system

In order to develop good system design, we need to ensure that our system is reliable, available, scalable & maintainable.




**a) Reliability in System design**

- A system is said to be reliable when it meets the end – user requirements without wearing out.
- A fault tolerant system can be the one that can continue to be functioning reliably even in the presence of faults.

**b) Availability in System design**

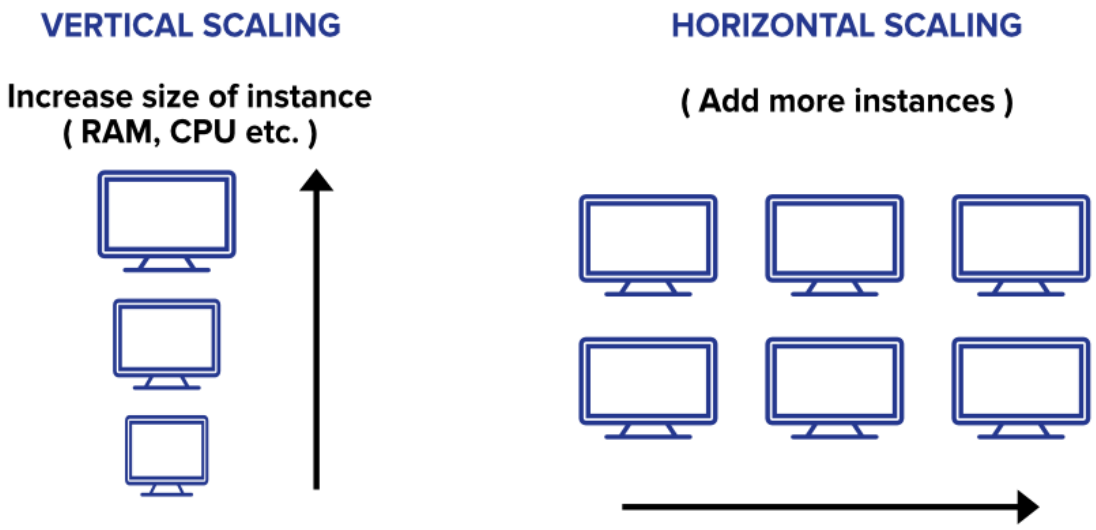
- Availability is a characteristic of a system which aims to ensure an agreed level of operational performance, also known as Uptime.
- It is essential for a system to ensure high availability in order to serve the user's request.
- The extent of Availability varies from System to System.

For e.g., Suppose we're designing a social media application then High availability is not much of a need i.e., a delay of few seconds can be tolerated. For example: Getting to view the post of your favorite post on Instagram with a delay of 5-10 seconds will not much of an issue.

- But if we're designing a system for hospitals, Data centers or Banking, then we should ensure that our system is highly available because a delay in the service can lead to huge loss.
- Points to keep in mind in order to ensure the availability of our system:
  - Our system should not have a single point of Failure i.e., basically our system should not be dependent on a single service in order to process al the requests because when that service fails then our entire system can be jeopardized & end up becoming unavailable.
  - Detecting the failure & resolving it at that point.

**c) Scalability in System design**

- Scalability refers to the ability of the system to cope up with the increasing load.
- While designing the system, we should keep in mind about the load experienced by the system. It is advised that if we have to design a system for load X then we should plan to design it for 10X & test it for 100X.
- For e.g., Suppose we're designing an E – commerce application then we should expect a spike in the load during a flash sale or when product is launched. In that case, our system should be smart enough to handle the increasing load efficiently & that makes it Scalable.
- In order to ensure Scalability, we should be able to compute the load our system will experience. Factors that describe the load on the system:
  - Number of requests coming to our system for getting processed per day.
  - Number of database calls made from our system
  - Amount of Cache Hit & Miss requests to our system
  - Users currently active on our system
- Solutions to scale the application for the enormous number of requests




1. **Vertical Scaling** (This approach is called **Scale – Up approach**)
   - Upgrading the capacity of a machine or moving to a new machine with more power.
   - We can add more power to our machine by adding better processors, increasing RAM or other power increasing adjustments.
   - Vertical scaling can be easily achieved by switching from small to bigger m/cs but remember that this involves downtime.
   - It doesn't require any partitioning of data & all the traffic resides on a single node with more capacity.
   - Less administrative effort as we need to manage just one system.
   - Application compatibility is maintained.
   - Easy implementation
   - Mostly used in small or mid – sized companies.
   - E.g., MySQL, Amazon RDS

**Drawbacks of Vertical Scaling**

- Replacing the server will require downtime in this approach.
- Limited potential for implementing network I/O or disk I/O i.e., limited scaling as there is finite scope of upgradability in the future.
- Greater risk of outrages & Hardware failures
- Implementation cost is expensive

2. **Horizontal Scaling** (This approach is called **Scale – Out approach**)
    - In horizontal scaling, we enhance the performance of the server by adding more machines to the network, sharing the processing & memory workload across multiple devices.
    - This approach is the best solution for projects which require high availability or failover.
    - Most organization choose this approach because it includes increasing I/O concurrency, reducing the load on existing nodes & increasing disk capacity.
    - **Horizontal scalability can be achieved with the help of a distributed file system, clustering & load balancing.**
    - Traffic can be managed effectively.
    - Google with its Gmail & YouTube, Facebook, Amazon etc. are heavily utilizing horizontal scaling.

**Drawbacks of Horizontal Scaling**

- Complicated Architectural Design
- High licensing fees
- High utility cost such as cooling & electricity, network equipment cost such as routers & switches

| S. No. | Vertical Scaling | Horizontal Scaling |
|---|---|---|
| 1. | Load balancing is not required.<br><br>In vertical scaling, it's just one m/c to handle the load so it doesn't require a load balancer | Load balancing is required.<br><br>Horizontal Scaling requires load balancing to distribute or spread the traffic among several m/cs |
| 2. | Single Point of failure<br><br>Vertical Scaling has only one m/c that's why it will have a single point of failure. | Resilient to system failure i.e., easier to run fault tolerance<br><br>Horizontal scaling is more resistant to system failure. If one of the m/c fails we can redirect the request to another m/c & the application won't face downtime. |
| 3. | Interprocess communication<br><br>Vertical Scaling works on Inter – process communication that is quite fast. | Utilizes Network calls (RPCs)<br><br>Horizontal scaling requires network communication or calls between m/cs. Network calls are slow & more to prone failure. |
| 4. | Data consistency<br><br>Vertical Scaling has only one m/c where all the requests will be redirected, so there is no issue of data inconsistency. | Data Inconsistency<br><br>In horizontal scaling, there is data inconsistency as different m/cs handle different requests which may lead to their data becoming out of sync |
| 5. | Hardware Limitation | Scales well i.e., easy to size or resize according to the needs |

| | | |
|---|---|---|
| 6. | Implementation cost is expensive | Implementation cost is less expensive than Vertical scaling |
| 7. | E.g., MySQL, Amazon RDS | E.g., Cassendra, MongoDB |
| 8. | Can experience downtime while upgrading | High availability |

## Which one is right for an application? Vertical scaling or Horizontal Scaling

- Both of the scaling has some pros & cons. There will be always some tradeoff.
- Certain factors are important to consider for better understanding of the approach we should take
    - Performance requirements or performance characteristics of an application.
    - System throughput, System response time, System availability requirement
    - Is the system fault – tolerant? If so, what is the degree of it?
    - Is the design reliable?
    - What level of consistency do we care about?
    - What's the scalability goal of the application (you might have some short – term or immediate one's goal, but what's going to happen in the long run?)

- Most of the time in big organizations, Engineers take some good qualities of vertical scaling and some good qualities of horizontal scaling. They follow the hybrid approach of combining the speed & consistency of vertical scaling, with the resilience & infinite scalability of horizontal scaling.

**2. System Design Terminologies**

- **System:** A system is an orderly grouping of independent components linked together according to a plan to achieve a specific goal.
- **Machine:** Machine denotes a single computer where we can host our application, database, etc.
    - It could denote a logical entity as well. E.g., AWS & other cloud providers provide a slice of a bigger m/c. The abstracted slice for the machine is also said to be a machine.
- **Server/Application:** Server/Application denotes the application that is running the backend code & is serving requests on a particular port on the m/c.
- **Client:** Clients are the devices/applications through which request is sent to the server.
- **Distributed System:** A system with multiple components on different machines which communicate & coordinate with one another.
- **Node:** A node denotes a single m/c in the distributed system.
- **Memory:** RAM
- **Storage:** File – system / Disk space.
- **In – Memory:** Data stored in the memory (RAM). It's faster to write to memory than to disk.
- **Resources:** Memory, Storage, CPU etc. Resources are what costs money. Storage is very cheap compared to memory.
- **Faults:** Faults are the errors that arise in a particular component of the system. An occurrence of fault doesn't guarantee the failure of the system.
- **Failure:** Failure is the state when the system is not able to perform as expected. It's no longer able to provide certain services to the end – users.

## ❖ HLD Vs LLD

| S.No. | Comparison basic | HLD | LLD |
|---|---|---|---|
| 1. | Stands for | High – Level Design | Low – Level Design |
| 2. | Alternate name | Macro level design of a system | Micro level design of a system |
| 3. | | Designing the architecture of the system without going into details of the code design or database schema design | Designing the code/class structure & database tables considering the entities, their relationships, & interactions, properties, methods etc. |

| | | | |
|---|---|---|---|
| 4. | Definition | HLD is the general system design means it refers to the overall system design | LLD is like detailing HLD means it refers to component level design process |
| 5. | Order of design phase | HLD is created before LLD | LLD is created after HLD |
| 6. | Conversions | HLD changes the client or business requirement (SRS) into a high – level solution. | LLD changes the high-level solution to a comprehensive solution. |
| 7. | Created by | Solution Architect | Designers & Developers |
| 8. | Input Criteria | The input measure in HLD is SRS. (S/W Requirement Specification) | The input measure in LLD is Reviewed HLD. |
| 9. | Output Criteria | The o/p measures in HLD are functional design, database design, & review record. | The o/p bases in the LLD are the Unit test plan & Program specification. |

- **Performance:** The amount of work that the system does. Increasing performance means the system should be able to do more amount of work.
- **Scalability:** A service is said to be scalable if when we increase the resource in a system, it results in increased performance in a manner proportional to the resource added.
  Note: Generally, the aim is to have both good performance (fast for a single user) as well as a scalable system (fast for multiple users).

- **Latency:** Latency is the time required to perform some action or to produce some result. Latency is measured in units of time – hours, minutes, seconds, nanoseconds, or clock periods.
- **Throughput:** Throughput is the number of actions executed or results produced per unit of time.
  Note: Generally, the aim is to have low latency & high throughput.

## Important system design concepts

- IP (Internet Protocol)
  - An IP address is a unique address that identifies a device on the internet or a local network.
  - IP (Internet Protocol) is the set of rules governing the format of data sent via the internet or local network.
  - IP addresses provide a way to differentiate b/w different computers, routers, & websites. IP addresses contain location information & make device accessible for communication.

- Versions (IPv4, IPv6)
  - The original internet protocol is IPv4 which uses a 32-bit numeric dot-decimal notation that only allows for around 4 billion IP addresses. As internet adoption grew, IPv4 was not enough
    e.g., ***102.22.192.181***

  - IPv6 is a new protocol (introduced in 1998/mid-2000s) which uses 128-bit alphanumeric hexadecimal notation that allows for about ~340e+36 IP addresses. That's more than enough to meet the growing demand for years to come
    e.g., ***2001:odb8:85a3:0000:8a2e:0370:7334***

- Types of IP addresses: Public, Private, Static, Dynamic
  - **Public:** A public IP address is an address where one primary address is associated with our whole network. In this type of IP address, each of the connected devices has the same IP address.
    e.g., IP address provided to your router by the ISP.

  - **Private:** A private IP address is a unique IP number assigned to every device that connects to our internet network, which includes devices like computers, tablets, & smartphones which are used in our household.
    e.g., IP addresses generated by the home router for the devices

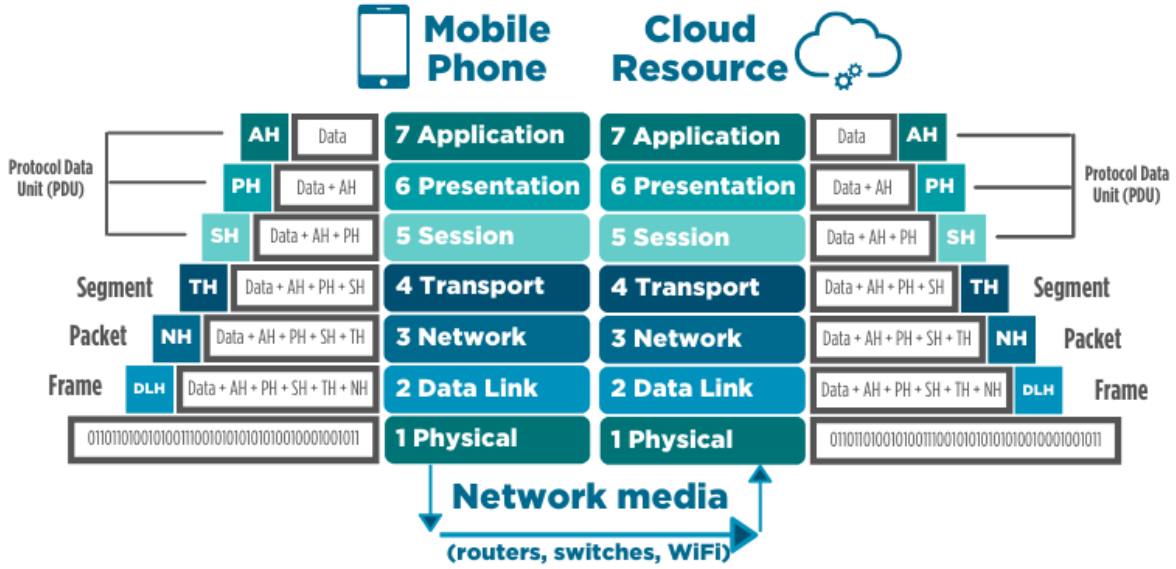
- **Static:** A static IP address doesn't change & is one that was manually created, as opposed to having been assigned. These addresses are usually more expensive but are more reliable
  e.g., They are usually used for important things like reliable geo-location services, remote access, server hosting etc.

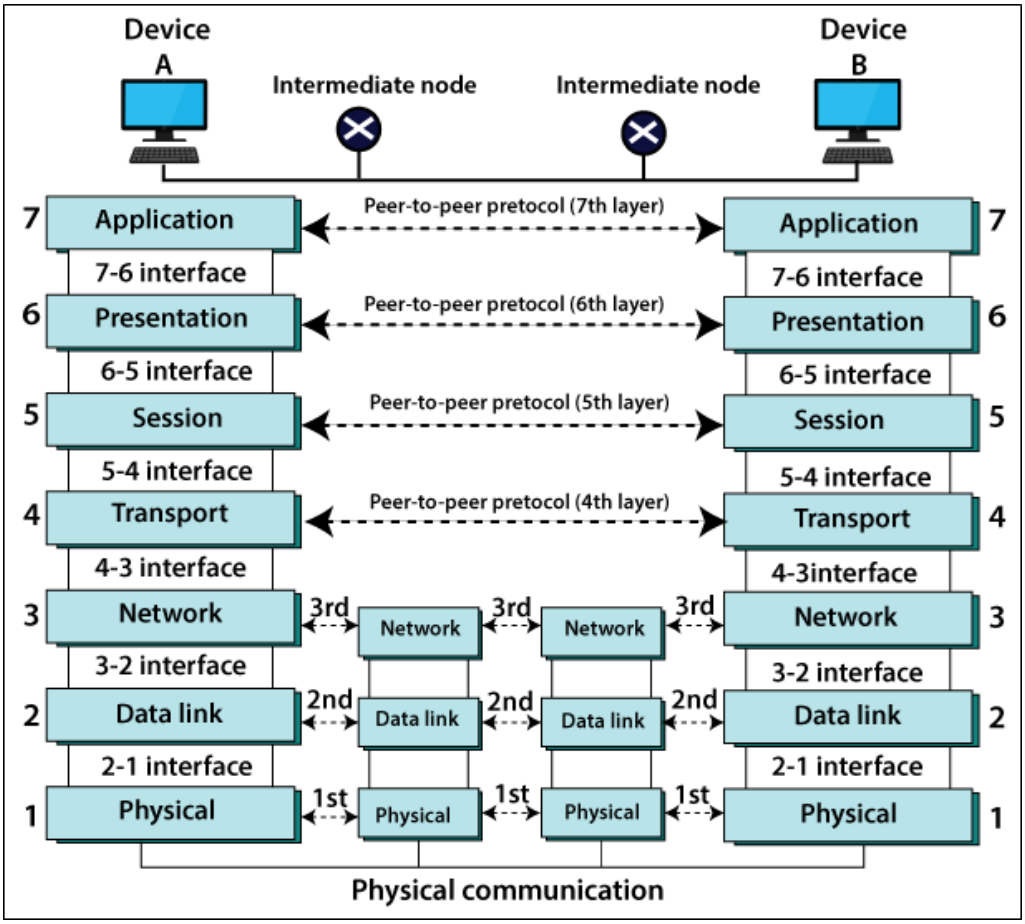
- **Dynamic:** A dynamic IP address changes from time to time & is not always the same. It has been assigned by a Dynamic Host Configuration Protocol (DHCP) server.
  e.g., They are more commonly used for consumer equipment & personal use.

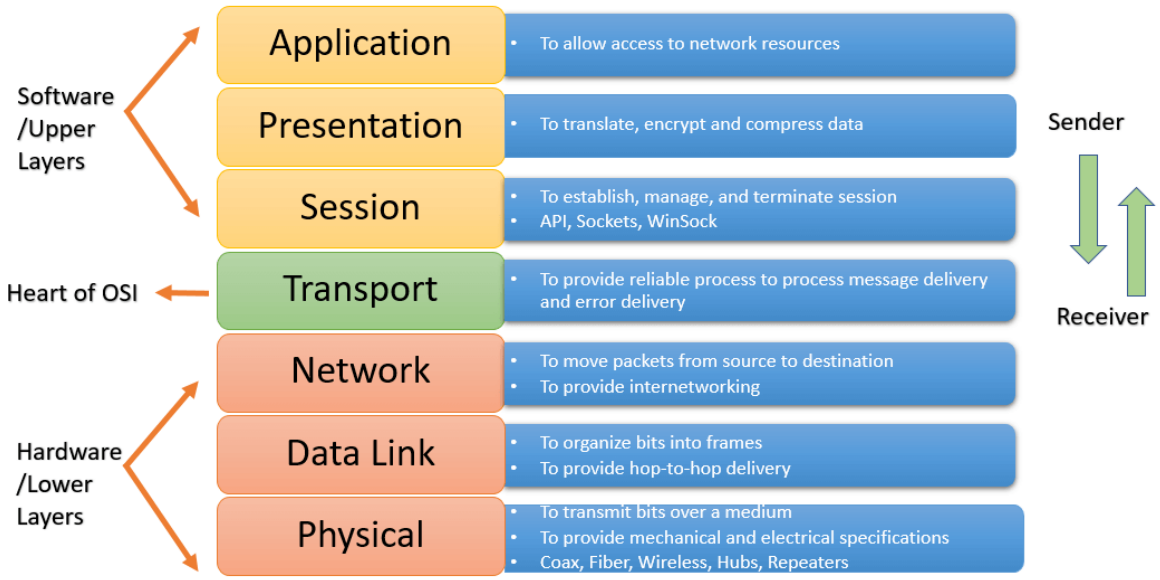
**Note:** Dynamic IP addresses are the most common type of IP address. They are cheaper to deploy & allow us to reuse IP addresses within a network as needed.

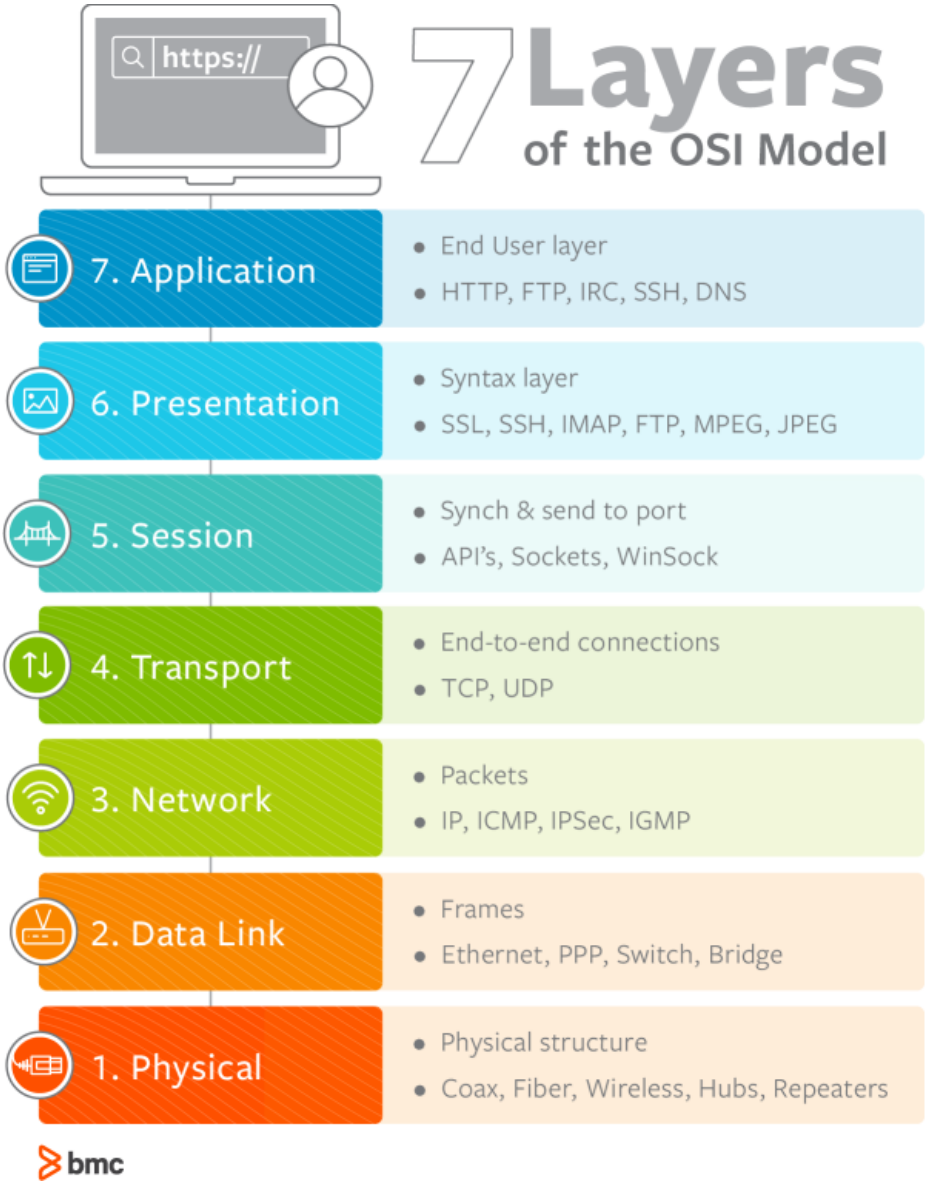
❖ OSI Model (Open System Interconnection Model)

- The OSI Model is a logical & conceptual model that defines network communication used by systems open to interconnection & communication with other systems.
- The OSI model also defines a logical network & effectively describes computer packet transfer by using various layers of protocols.
- The OSI Model can be seen as a universal language for computer networking based on the concept of splitting up a communication system into 7 abstract layers, each one stacked upon the last.




Device A
Device B
Intermediate node
Intermediate node
7 Application
Peer-to-peer pretocol (7th layer)
Application 7
7-6 interface
6 Presentation
Peer-to-peer pretocol (6th layer)
Presentation 6
6-5 interface
5 Session
Peer-to-peer pretocol (5th layer)
Session 5
5-4 interface
4 Transport
Peer-to-peer pretocol (4th layer)
Transport 4
4-3 interface
4-3interface
3 Network
3rd
Network
Network
Network 3
3-2 interface
2 Data link
2nd
Data link
Data link
Data link 2
2-1 interface
1 Physical
1st
Physical
Physical
Physical 1
Physical communication


Application
To allow access to network resources
Presentation
To translate, encrypt and compress data
Session
To establish, manage, and terminate session
API, Sockets, WinSock
Transport
To provide reliable process to process message delivery and error delivery
Network
To move packets from source to destination
To provide internetworking
Data Link
To organize bits into frames
To provide hop-to-hop delivery
Physical
To transmit bits over a medium
To provide mechanical and electrical specifications
Coax, Fiber, Wireless, Hubs, Repeaters
Software /Upper Layers
Heart of OSI
Hardware /Lower Layers
Sender
Receiver


https://
7 Layers
of the OSI Model
7. Application
End User layer
HTTP, FTP, IRC, SSH, DNS
6. Presentation
Syntax layer
SSL, SSH, IMAP, FTP, MPEG, JPEG
5. Session
Synch & send to port
API's, Sockets, WinSock
4. Transport
End-to-end connections
TCP, UDP
3. Network
Packets
IP, ICMP, IPSec, IGMP
2. Data Link
Frames
Ethernet, PPP, Switch, Bridge
1. Physical
Physical structure
Coax, Fiber, Wireless, Hubs, Repeaters
bmc