DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An Autonomous Institution Affiliated to VTU, Belagavi)
DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

MODULE 1:

Meaning of system design, Importance of system design concepts, Design requirements, Computer Architecture, Application Architecture, Vertical scaling, Horizontal scaling, system design in networking: TCP and UDP protocols, Domain Name servers (DNS), HTTP Protocols.

WHAT IS SYSTEM DESIGN

System design is the **process** of designing the elements of a system such as the **architecture**, **modules**, and **components**, the different **interfaces** of those components, and the **data** that goes through that system.

OR

System design is the process of defining the architecture, components, modules, interfaces, and overall structure of a system to meet specified requirements and goals. It involves creating a blueprint that outlines how various elements interact and work together to achieve the desired functionality, performance, and reliability.

ELEMENTS OF A SYSTEM

- **Architecture** – This is the conceptual model that defines the structure, behaviour, and views of a system. We can use flowcharts to represent and illustrate the architecture.
- **Modules** – These are components that handle one specific task in a system. A combination of the modules makes up the system.
- **Components** – This provides a particular function or group of related functions. They are made up of modules.
- **Interfaces** – This is the shared boundary across which the components of the system exchange information and relate.
- **Data** – This is the management of the information and data flow.

Importance of System Design Concepts

1. System design is a critical skill for software engineers. A good system design is crucial for any company because it can have a significant impact on the success of the project and the overall performance of the company. A well-designed system can help a company have a competitive edge, increase efficiency and reduce costs, leading to better performance and profitability.
2. It allows them to **scale applications** and **create efficient architectures** that are resilient and performant.
3. Understanding system design helps engineers to think in an abstract way and come up with solutions to **complex problems**.
4. It also enables them to communicate effectively with other **engineers**, **architects**, and **stakeholders**.
5. System design involves understanding the requirements of a system and designing an architecture that meets those requirements. This includes understanding the trade-offs between **scalability**, **cost**, **performance**, **reliability**, and **maintainability**.
6. It also involves understanding the best practices and architectural patterns to use in order to make the system as robust as possible.
7. **When designing a system, it is important to consider the underlying infrastructure, such as databases, networks, and other services.**
8. It is also important to design for scalability and performance. This includes understanding **caching strategies**, **load balancing**, and **effective use of resources**.
9. It is also important to consider the security of the system. This includes understanding **authentication**, **authorization**, **encryption**, and other best practices.
10. It is also important to consider the impact of external systems such as **third-party APIs** and **services**.

DESIGN REQUIREMENTS

Availability:

Availability refers to the **readiness and accessibility** of a system or service to users at any given time. **It measures the percentage of time a system remains operational and usable.** High availability ensures that users can access the system without significant interruptions or downtime, typically achieved through redundancy, fault tolerance, and efficient recovery mechanisms.

How is availability measured?

Availability is usually measured as a percentage and is often expressed in terms of “uptime” versus “downtime” over a given period. For instance, a system with 99% availability means it is expected to be operational and accessible 99% of the time, while the remaining 1% represents the allowable downtime.

Why is Availability Important in System Design?

1. **User Experience:** Availability ensures that users can access the system and its services when needed, leading to a positive user experience. Systems that are frequently unavailable or experience downtime frustrate users and may lead to dissatisfaction, loss of trust, and even abandonment of the system in favor of alternatives.

1. **Business Continuity:** Availability is essential for maintaining business continuity and ensuring uninterrupted operations. For businesses that rely on their systems to deliver services or conduct transactions, even brief periods of downtime can result in significant financial losses, damage to reputation, and legal liabilities.

2. **Competitive Advantage:** High availability can be a competitive differentiator for businesses, particularly in industries where uptime and reliability are critical factors. Systems that offer better availability compared to competitors are more **likely to attract and retain customers**, leading to a competitive advantage in the market.
3. **Disaster Recovery:** Availability is closely related to disaster recovery and resilience. Designing systems with redundancy, failover mechanisms, and disaster recovery plans ensures that the system can withstand and recover from unexpected events such as hardware failures, network outages, natural disasters, or cyberattacks.
4. **Regulatory Compliance:** In many industries, there are regulatory requirements or standards that mandate a minimum level of system availability. Failure to comply with these regulations can result in legal consequences, fines, or sanctions. Therefore, designing systems with high availability is essential for ensuring regulatory compliance.

How do we achieve high availability?

High availability is essential for systems where continuous operation is vital, and any disruption could lead to financial losses, reputational damage, or even safety hazards. Commonly, systems with high availability requirements include banking applications, e-commerce platforms, healthcare systems, emergency response services, and cloud infrastructure.

System designers implement various strategies and technologies to achieve high availability, such as:

- **Redundancy:** Employ redundant components or servers to ensure that another can take over seamlessly if one fails. This can include redundancy at different levels, such as hardware, networking, and data centres.

- **Load balancing:** Distributing incoming requests across multiple servers or resources to prevent overload on any single component and improve overall system performance and fault tolerance.
- **Disaster Recovery (DR):** Having a comprehensive plan in place to recover the system in case of a catastrophic event that affects the primary infrastructure.
- **Monitoring and Alerting:** Implementing robust monitoring systems that can detect issues in real-time and notify administrators to take appropriate action promptly.
- **Performance optimization:** Ensuring that the system is designed and tuned to handle the expected load efficiently, reducing the risk of bottlenecks and failures.
- **Scalability:** Designing the system to scale easily by adding more resources when needed to accommodate increased demand.

SLO: Service Level Objective:

A service level objective (SLO) is an agreed-upon performance target for a particular service over a period of time. SLOs define the expected status of services and help stakeholders manage the health of specific services, as well as optimize decisions balancing innovation and reliability.

Service Level Agreements (SLAs):

Many organizations commit to meeting specific availability targets through SLAs with their customers or stakeholders. Failure to meet these SLAs can result in financial penalties or contractual obligations. Therefore, designing systems with high availability is crucial for meeting SLA requirements and maintaining customer satisfaction.

Reliability:

- It is defined as the probability of failure free operation of a system.
- The reliability of a device is considered high if it has repeatedly performed its function with success and low if it has tended to fail in repeated trials. The reliability of a system is defined as the probability of performing the intended function over a given period under specified operating conditions.

How to achieve high reliability?

Fault Tolerance:

Fault tolerance is the capability of a system to deliver uninterrupted service despite one or more of its components failing.

Fault-tolerant systems are designed to compensate for multiple failures. Such systems automatically detect a failure of the central processing unit, input/output (I/O) subsystem, memory cards, motherboard, power supply or network components. The failure point is identified, and a backup component or procedure immediately takes its place with no loss of service.

Fault tolerance can be provided with:

- software,
- embedded in hardware or
- enabled by some combination of the two.

Redundancy:

Redundancy and fault tolerance are both mechanisms that help ensure systems continue to function properly, even when there are faults:

Redundancy: The ability to configure backup components to take over if a component fails. For example, a system might have multiple power supply units (PSUs), so if the primary PSU fails, a redundant PSU can take over. Redundancy can be implemented at the system level, such as having a backup computer system.

Throughput:

Throughput is the maximum rate of transfer or capacity of the system. It is used as a metric to determine how much work the system can do in a given time frame. One promising way to increase the Throughput of the system is by splitting up the requests and distributing them to various resources.

Servers: It is a Measure of number of requests from the users served in a second(or a in a given amount of time)

Databases: : It is a Measure of number of queries executed per second.

Latency:

- Latency is a measure of the time duration to produce the result. It is the time spent or lags generating the desired output. In other words, latency helps us measure the speed of the system.
- Lower the Latency, the higher the speed of the system.
- Example: Network latency is the delay in time it takes for data to travel across a network. It's typically measured in milliseconds.

System design tradeoffs

Identifying and balancing these tradeoffs is an essential aspect of system design.

1. **Scalability vs. simplicity:** The more complex a system is, the harder it may be to scale it to handle a larger number of users or transactions.
2. **Flexibility vs. ease of use:** A system that is highly flexible may be more difficult to use, while one that is easy to use may be less flexible.
3. **Security vs. performance:** Increasing security measures may decrease the performance of a system, while increasing performance may decrease the level of security.
4. **Cost vs. functionality:** A system that offers more functionality may be more expensive to develop and maintain, while one that is simpler may be less costly.
5. **Availability vs. durability:** A system that is highly available may be less durable and vice versa.
6. **Maintainability vs. performance:** A system that is easy to maintain may have a lower performance compared to one that is difficult to maintain.
7. **Latency vs. throughput:** A system that focuses on low latency may have a lower throughput and vice versa.
8. **Customizability vs. compatibility:** A system that is highly customizable may be less compatible with other systems, while one that is more compatible may be less customizable.

COMPUTER ARCHITECTURE

CPU DESIGN:

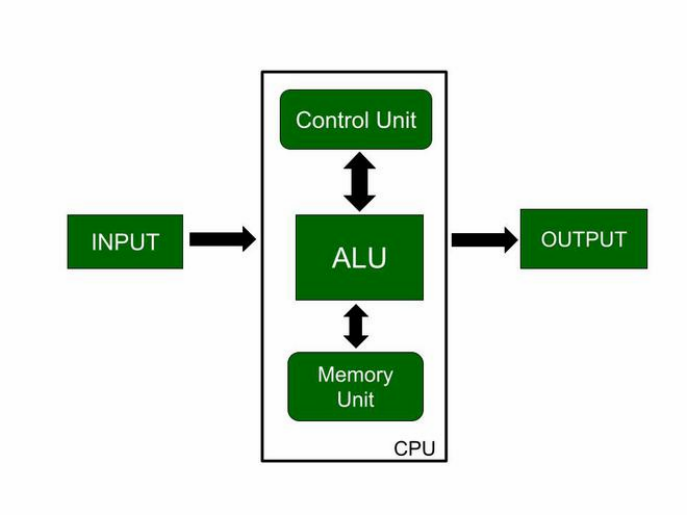
A CPU is hardware that performs data input/output, processing, and storage functions for a computer system.

Different Parts of CPU

Now, the CPU consists of 3 major units, which are:

- Memory or Storage Unit
- Control Unit
- ALU(Arithmetic Logic Unit)

Let us now look at the block diagram of the computer:



Here, in this diagram, the three major components are also shown. So, let us discuss these major components in detail.

Memory or Storage Unit

As the name suggests this unit can store instructions, data, and intermediate results. The memory unit is responsible for transferring information to other units of the computer when needed. It is also known as an internal storage unit or the main memory or the primary storage or Random Access Memory (RAM) as all these are storage devices.

Its size affects speed, power, and performance. There are two types of memory in the computer, which are primary memory and secondary memory. Some main functions of memory units are listed below:

- Data and instructions are stored in memory units which are required for processing.
- It also stores the intermediate results of any calculation or task when they are in process.
- The final results of processing are stored in the memory units before these results are released to an output device for giving the output to the user.
- All sorts of inputs and outputs are transmitted through the memory unit.

Control Unit

As the name suggests, a control unit controls the operations of all parts of the computer but it does not carry out any data processing operations. Executing already stored instructions, It instructs the computer by using the electrical signals to instruct the computer system. It takes instructions from the memory unit and then decodes the instructions after that it executes those instructions. So, it controls the functioning of the computer. Its main task is to maintain the flow of information across the processor. Some main functions of the control unit are listed below:

- Controlling of data and transfer of data and instructions is done by the control unit among other parts of the computer.

- The control unit is responsible for managing all the units of the computer.
- The main task of the control unit is to obtain the instructions or data that is input from the memory unit, interpret them, and then direct the operation of the computer according to that.
- The control unit is responsible for communication with Input and output devices for the transfer of data or results from memory.
- The control unit is not responsible for the processing of data or storing data.

ALU (Arithmetic Logic Unit)

ALU (Arithmetic Logic Unit) is responsible for performing arithmetic and logical functions or operations. It consists of two subsections, which are:

- **Arithmetic Section:** By arithmetic operations, we mean operations like addition, subtraction, multiplication, and division, and all these operations and functions are performed by ALU. Also, all the complex operations are done by making repetitive use of the mentioned operations by ALU.
- **Logic Section:** By Logical operations, we mean operations or functions like selecting, comparing, matching, and merging the data, and all these are performed by ALU.

Note: The CPU may contain more than one ALU and it can be used for maintaining timers that help run the computer system.

MEMORY HIRARCHY:

2. Memory Hierarchy

The memory hierarchy in low-power embedded systems is designed to balance cost, speed, and power efficiency:

- **Random Access Memory (RAM)**: RAM is the primary memory where data and instructions are stored while the CPU is running. For embedded systems, the RAM size is typically small (a few kilobytes to a few megabytes) to reduce power consumption and cost.
- **Cache Memory**: Embedded systems may have a small cache to store frequently accessed data close to the CPU. Typically, they might only use L1 cache due to power and cost constraints. This reduces the need for the CPU to access slower RAM and helps conserve power.
- **Read-Only Memory (ROM)**: ROM stores permanent instructions or data needed to boot the system and perform essential functions. In embedded systems, ROM often contains firmware, which provides a simple, low-power way to ensure reliable operation.

3. Input/Output (I/O) Components

Input/output components are essential in embedded systems, allowing them to interact with the external environment:

- **Role of I/O Components**: I/O components, such as sensors, buttons, displays, and communication interfaces, allow the system to receive input from and provide output to the user or other systems. These components can include serial interfaces (like UART, SPI, or I2C) for simple communication.

MAIN PARTS OF PROCESSOR

- The **processor** contains ALU, control-circuitry and many registers.
- The processor contains „n" general-purpose registers $\mathbf{R_0}$ through $\mathbf{R_{n-1}}$.
- The **IR** holds the instruction that is currently being executed.
- The **control-unit** generates the timing-signals that determine when a given action is to take place.
- The **PC** contains the memory-address of the next-instruction to be fetched & executed.
- During the execution of an instruction, the contents of PC are updated to point to next instruction.
- The **MAR** holds the address of the memory-location to be accessed.
- The **MDR** contains the data to be written into or read out of the addressed location.
- MAR and MDR facilitates the communication with memory.

  (IR → Instruction-Register, PC → Program Counter)
  (MAR → Memory Address Register, MDR→ Memory Data Register)

STEPS TO EXECUTE AN INSTRUCTION

1) The address of first instruction (to be executed) gets loaded into PC.
2) The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal to memory.
3) After certain amount of elapsed time, the first instruction is read out of memory and placed into MDR.
4) Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded & executed.
5) To fetch an operand, it's address is placed into MAR & control-unit issues Read signal. As a result, the operand is transferred from memory into MDR, and then it is transferred from MDR to ALU.
6) Likewise required number of operands is fetched into processor.
7) Finally, ALU performs the desired operation.
8) If the result of this operation is to be stored in the memory, then the result is sent to the MDR.
9) The address of the location where the result is to be stored is sent to the MAR and a Write cycle is initiated.
10) At some point during execution, contents of PC are incremented to point to next instruction in the program.

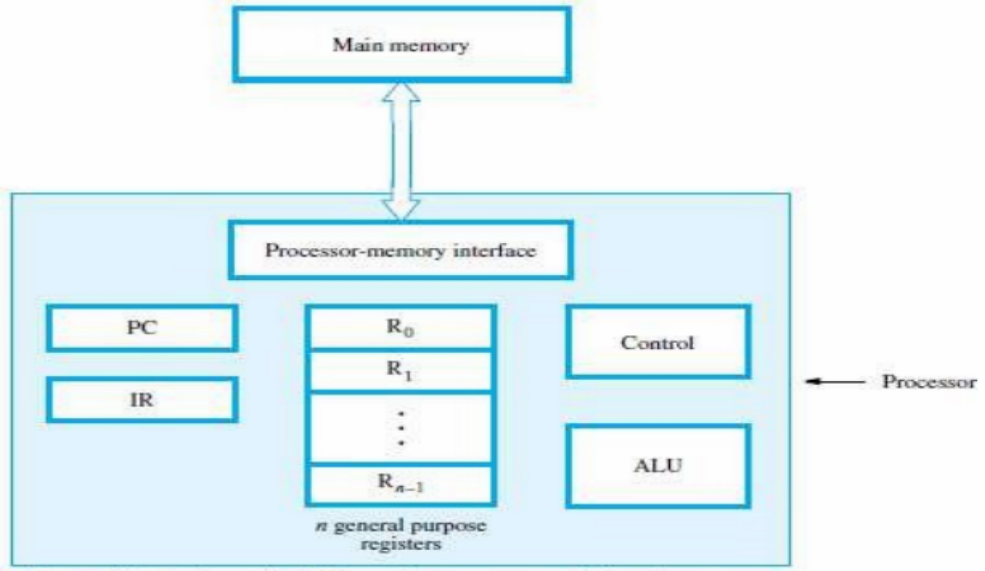


Figure 1.2 Connection between the processor and the main memory.

BUS STRUCTURE

- A bus is a group of lines that serves as a connecting path for several devices.
- A bus may be lines or wires.
- The lines carry data or address or control signal.
- There are 2 types of Bus structures: 1) Single Bus Structure and 2) Multiple Bus Structure.

1) Single Bus Structure

- Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time.
- Bus control lines are used to arbitrate multiple requests for use of the bus.
- **Advantages:**
  1) Low cost &
  2) Flexibility for attaching peripheral devices.

2) Multiple Bus Structure

- Systems that contain multiple buses achieve more concurrency in operations.
- Two or more transfers can be carried out at the same time.
- **Advantage:** Better performance.
- **Disadvantage:** Increased cost.

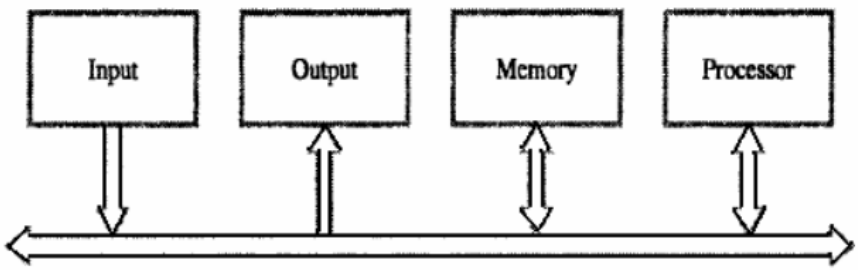


Figure 1.3 Single-bus structure.

- The devices connected to a bus vary widely in their speed of operation.
- To synchronize their operational-speed, buffer-registers can be used.
- **Buffer Registers**
  → are included with the devices to hold the information during transfers.
  → prevent a high-speed processor from being locked to a slow I/O device during data transfers.

PERFORMANCE

- The most important measure of performance of a computer is how quickly it can execute programs.
- The speed of a computer is affected by the design of
  1) Instruction-set.
  2) Hardware & the technology in which the hardware is implemented.
  3) Software including the operating system.
- Because programs are usually written in a HLL, performance is also affected by the compiler that translates programs into machine language. (HLL→ High Level Language).
- For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.

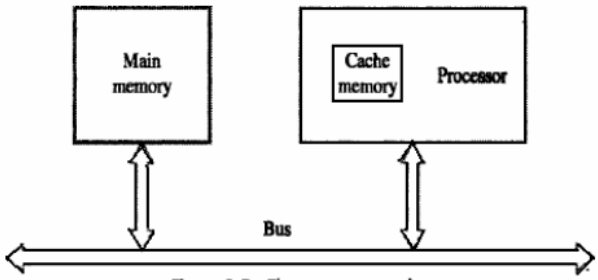


Figure 1.5 The processor cache.

examine the flow of program instructions and data between the memory & the processor.

- At the start of execution, all program instructions are stored in the main-memory.
- As execution proceeds, instructions are fetched into the processor, and a copy is placed in the cache.
- Later, if the same instruction is needed a second time, it is read directly from the cache.
- A program will be executed faster
  if movement of instruction/data between the main-memory and the processor is minimized
  which is achieved by using the cache.

PROCESSOR CLOCK

- Processor circuits are controlled by a timing signal called a **Clock**.
- The clock defines regular time intervals called **Clock Cycles**.
- To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps such that each step can be completed in one clock cycle.
- Let P = Length of one clock cycle
  R = Clock rate.
- Relation between P and R is given by

$$R = \frac{1}{P}$$

- R is measured in cycles per second.
- Cycles per second is also called Hertz (Hz)

BASIC PERFORMANCE EQUATION

- Let T = Processor time required to executed a program.
  N = Actual number of instruction executions.
  S = Average number of basic steps needed to execute one machine instruction.
  R = Clock rate in cycles per second.
- The program execution time is given by

$$T = \frac{N \times S}{R} \qquad \text{------(1)}$$

- Equ1 is referred to as the basic performance equation.
- To achieve high performance, the computer designer must reduce the value of T, which means reducing N and S, and increasing R.
  - The value of N is reduced if source program is compiled into fewer machine instructions.
  - The value of S is reduced if instructions have a smaller number of basic steps to perform.
  - The value of R can be increased by using a higher frequency clock.
- Care has to be taken while modifying values since changes in one parameter may affect the other.

CLOCK RATE

- There are 2 possibilities for increasing the clock rate R:
  1) Improving the IC technology makes logic-circuits faster.
     This reduces the time needed to compute a basic step. (IC → integrated circuits).
     This allows the clock period P to be reduced and the clock rate R to be increased.
  2) Reducing the amount of processing done in one basic step also reduces the clock period P.
- In presence of a cache, the percentage of accesses to the main-memory is small.
  Hence, much of performance-gain expected from the use of faster technology can be realized.
  The value of T will be reduced by same factor as R is increased „." S & N are not affected.

PERFORMANCE MEASUREMENT

- Benchmark refers to standard task used to measure how well a processor operates.
- The Performance Measure is the time taken by a computer to execute a given benchmark.
- SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC → System Performance Evaluation Corporation).
- SPEC Rating is given by

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

- SPEC rating = 50 → The computer under test is 50 times as fast as reference-computer.
- The test is repeated for all the programs in the SPEC suite.
  Then, the geometric mean of the results is computed.
- Let $SPEC_i$ = Rating for program „i' in the suite.
  Overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{\frac{1}{n}}$$

where n = no. of programs in the suite.

INSTRUCTION SET: CISC AND RISC

RISC	CISC
Simple instructions taking one cycle.	Complex instructions taking multiple cycle.
Instructions are executed by hardwired control unit.	Instructions are executed by microprogrammed control unit.
Few instructions.	Many instructions.
Fixed format instructions.	Variable format instructions.
Few addressing modes, and most instructions have register to register addressing mode.	Many addressing modes.
Multiple register set.	Single register set.
Highly pipelined.	No pipelined or less pipelined.

What is Pipelining?

Pipelining is an arrangement of the CPU's hardware components to raise the CPU's general performance. In a pipelined processor, procedures called 'stages' are accomplished in parallel, and the execution of more than one line of instruction occurs.

Now let us look at a real-life example that should operate based on the pipelined operation concept. Consider a water bottle packaging plant. For this case, let there be 3 processes that a bottle should go through, inserting the bottle(I), Filling water in the bottle(F), Sealing the bottle(S).

It will be helpful for us to label these stages as stage 1, stage 2, and stage 3. Let each stage take 1 minute to complete its operation. Now, in a non-pipelined operation, a bottle is first inserted in the plant, and after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Likewise, when the bottle is in stage 3 both stage 1 and stage 2 are inactive. But in pipelined operation, when the bottle is in stage 2, the bottle in stage 1 can be reloaded. In the same way,

during the bottle 3 there could be one bottle in the 1st and 2nd stage accordingly. Therefore at the end of stage 3, we receive a new bottle for every minute. Hence, the average time taken to manufacture 1 bottle is:

Therefore, the average time intervals of manufacturing each bottle is:

Without pipelining = 9/3 minutes = 3m

```
I F S | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)
```

With pipelining = 5/3 minutes = 1.67m

```
I F S | |
| I F S |
| | I F S (5 minutes)
```

Thus, pipelined operation increases the efficiency of a system.

- **Stage 1 (Instruction Fetch):** In this stage the CPU fetches the instructions from the address present in the memory location whose value is stored in the program counter.
- **Stage 2 (Instruction Decode):** In this stage, the instruction is decoded and register file is accessed to obtain the values of registers used in the instruction.
- **Stage 3 (Instruction Execute):** In this stage some of activities are done such as ALU operations.
- **Stage 4 (Memory Access):** In this stage, memory operands are read and written from/to the memory that is present in the instruction.
- **Stage 5 (Write Back):** In this stage, computed/fetched value is written back to the register present in the instructions.

Execution in a pipelined processor:

Execution sequence of instructions in a pipelined processor can be visualized using a space-time diagram. For example, consider a processor having 4 stages and let there be 2 instructions to be executed. We can visualize the execution sequence through the following space-time diagrams:

Non-Overlapped Execution

Stage / Cycle	1	2	3	4	5	6	7	8
S1	I_1				I_2			
S2		I_1				I_2		
S3			I_1				I_2	
S4				I_1				I_2

Total time = 8 Cycle

Overlapped Execution

Stage / Cycle	1	2	3	4	5
S1	I_1	I_2			
S2		I_1	I_2		
S3			I_1	I_2	
S4				I_1	I_2

Total time = 5 Cycle **Pipeline Stages**

APPLICATION ARCHITECTURE

In this high-level overview, we'll explore the architecture of a production-grade application. This will serve as a foundation for the rest of the course, allowing us to delve into each component in more detail later on.

Within a production application architecture, various components work together to create a robust system, we'll provide a brief introduction to these components here.

A developer's perspective

We can start viewing this application architecture from the perspective of a developer, which will be familiar to most of you. Developers write code that is **deployed** to a **server**. For now, let's define a **server** as a computer that handles requests from another computer. This server also requires **persistent storage** to store the application's data. A server may have built-in storage, but that has its limitations in terms of size. As such, a server may talk to an external storage system (database, cloud etc). This storage may not be part of the same server, and is instead connected through a **network**.

A user's perspective

A user is someone who makes a request from the server, usually through a web browser. In this case, the web browser is the **client** to whom the server responds to.

If a user wanted to use a front-end feature, the server will respond with the necessary JavaScript/HTML/CSS code, compiled to display what the user requested. But, what if we have a lot of users and the single server cannot handle all of the requests on its own? There is bound to be a bottleneck, either through our RAM or our CPU. To maintain performance while dealing with multiple users, we will need to scale our server.

Scaling our server

To handle multiple requests, it might be a good idea to add more RAM or upgrade to a CPU with more cores and higher clocking speed. However, every computer has a limitation in terms of upgrades. Upgrading components *within* the same computer is referred to as **vertical scaling**.

We can also have multiple servers running our code, and we can distribute the user requests among these servers. This way, not all users are talking to one server, which ensures that the speed of each server remains intact. This also ensures that if one server were to go down, we can direct our traffic to one of our other servers. This is known as **horizontal scaling**.

Generally, in large systems, we prefer horizontal scaling, as it is much more powerful, and can be achieved with commodity hardware (i.e., relatively inexpensive, standard hardware). However, it also requires much more engineering effort, as we need to ensure that the servers are communicating with each other, and that the user requests are being distributed evenly.

For simple applications however, vertical scaling may be sufficient and the easier solution to implement. Even some services within Amazon Prime Video were recently migrated from a microservice architecture to a monolithic architecture.

But with multiple servers, what determines which requests go to which server? This is achieved through a **load balancer**. A load balancer will evenly distribute the incoming requests across a group of servers.

It's also important to remember that servers don't exist in isolation. It is highly likely that servers are interacting with external servers, through APIs. For example, the neetcode.io website interacts with other services like Stripe, through an API.

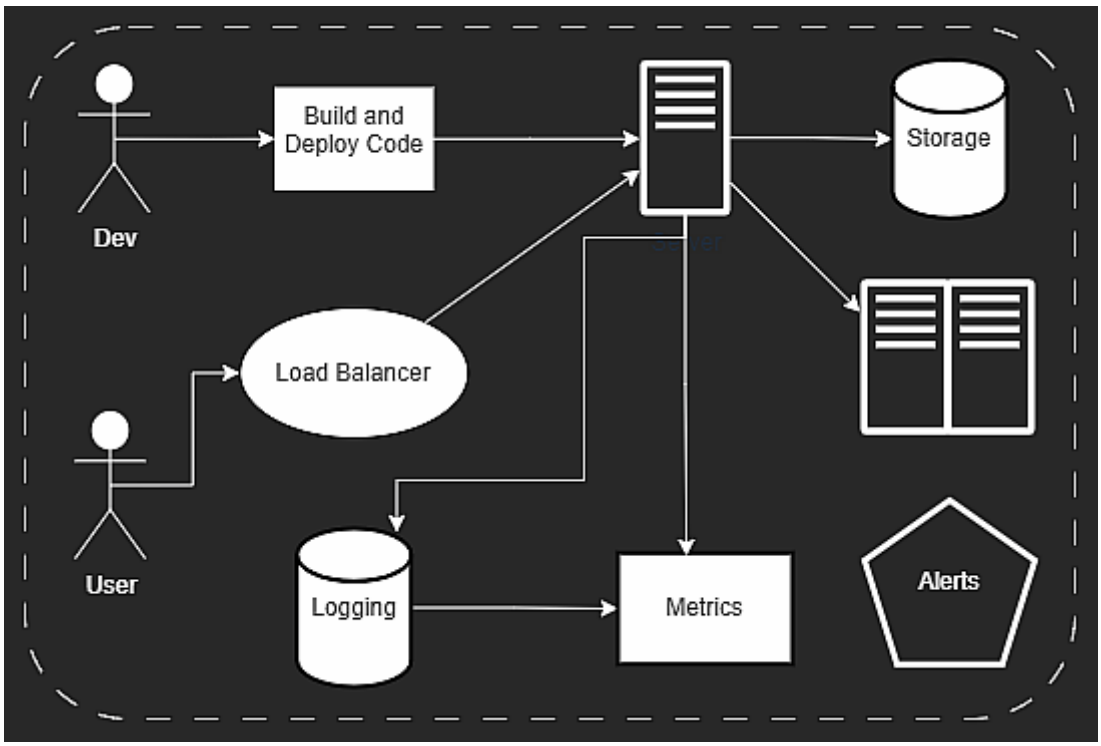
Logging and Metrics

Servers also have **logging** services, which gives the developer a log of all the activity that happened. Logs can be written to the same server, but for better reliability they are commonly written to *another* external server.

This gives developers insight into how the requests went, if any errors occured, or what happened before a server crashed. However, logs don't provide the complete picture. If our RAM has become the bottleneck of our server, or our CPU resources are restricting the requests being handled efficiently, we require a **metrics** service. A metric service will collect data from different sources within our server environment, such as CPU usage, network traffic etc. This allows developers to gain insights into server's behavior and identify potential bottlenecks.

Alerts

As developers, we wouldn't want to keep checking metrics to see if any unexpected behavior exhibits itself. This would be like checking your phone every 55 minutes for a notification. It is more ideal to receive a push notification. We can program alerts so that whenever a certain metric fails to meet the target, the developers receive a push notification. For example, if 100%100% of the user requests receive successful responses, we could set an alert to be notified if this metric dips under 95%95%.



The visual above demonstrates (on a very high level) how the components interact with each other, and what components the users interacts with and what components the developer interacts with.

2. What is Scalability?

Scalability refers to the ability of a system to handle increasing amounts of workload or requests without sacrificing performance or incurring excessive costs.

There are two main types of scalability:

- **Vertical scaling or Scale-up**
- **Horizontal scaling or Scale-out**

What is Vertical Scaling?

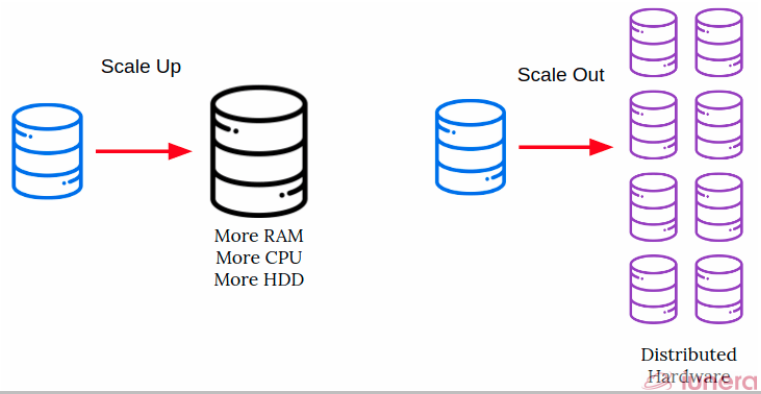
Vertical scaling, also known as **scaling up**, refers to the process of increasing the capacity or capabilities of an individual hardware or software component within a system. Vertical scaling aims to improve the performance and capacity of the system to handle higher loads or more complex tasks without changing the fundamental architecture or adding additional servers.

Vertical Scaling is achieved by:

- Adding faster processors, increasing RAM, or a larger hard drive and other power-increasing adjustments.
- By doing so, you're vertically scaling your system, increasing its capacity to handle more demanding tasks.

Characteristics of vertical scaling:

- It doesn't require any partitioning of data and all the traffic resides on a **single node with more resources**.
- Its implementation is easy.
- Less administrative effort as you need to manage just one system.
- Application compatibility is maintained.
- Mostly used in small and mid-sized companies.
- MySQL and Amazon RDS is a good examples of vertical scaling.

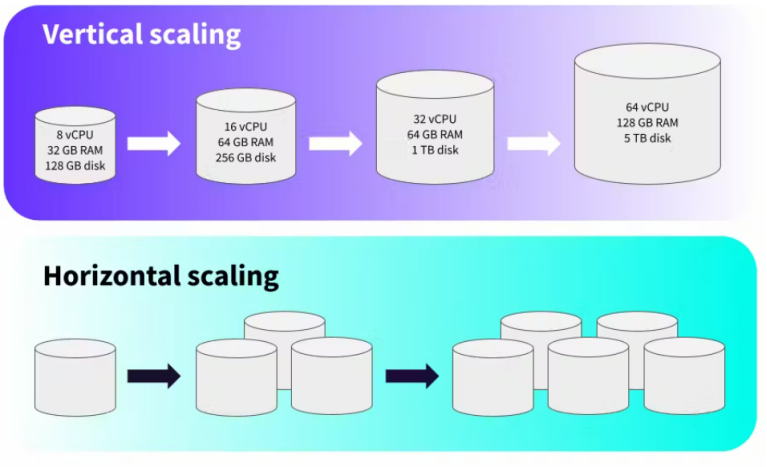


What is Horizontal Scaling?

Horizontal scaling, also known as scaling out, refers to the process of increasing the capacity or performance of a system by adding more machines or servers to distribute the workload across a larger number of individual units. In this approach, there is no need to change the capacity of the server or replace the server. a

Characteristics of the horizontal scaling

- This approach is also known as the '**scale-out**' approach.
- Horizontal scalability can be achieved with the help of a distributed file system, clustering, and load–balancing.
- Traffic can be managed effectively.
- Easier to run fault tolerance.
- Easy to upgrade.
- Instant and continuous availability.
- Easy to size and resize according to your needs.
- Implementation cost is less expensive compared to scaling up.
- Google with its Gmail and YouTube, Yahoo, Facebook, eBay, Amazon, etc. are heavily utilizing horizontal scaling.
- Cassandra and MongoDB are good examples of horizontal scaling.



Aspect	Horizontal Scaling	Vertical Scaling
Resource Addition	Adds more machines or servers to distribute workload	Enhances resources of individual components
Cost Effectiveness	Generally more cost-effective for large-scale systems	Initially simpler, but can become costlier long-term
Flexibility	Offers greater flexibility as it's easier to add units	Limited flexibility, especially with hardware
Fault Tolerance	Enhances fault tolerance by distributing workload	Limited fault tolerance as it relies on a single unit
Performance	Performance can improve as workload is distributed	Performance may improve, but can hit hardware limits
Single Point of Failure	Less prone to single points of failure	Potential single points of failure due to one unit
Complexity	Can introduce complexity in managing distributed system	Simpler to manage as it involves fewer components
Machine Communication	Horizontal scaling relies heavily on network communication to coordinate tasks and share data between distributed machines	Vertical scaling primarily involves interprocess communication within a single machine or between closely coupled processes, minimizing the need for network communication

Describe the applications in which vertical scaling concept is used

Vertical scaling, also known as "scaling up," involves increasing the capacity of a single server or machine by adding more powerful resources, such as CPU, RAM, or storage. This approach enhances the server's performance, allowing it to handle a larger workload without distributing it across multiple machines. Here are some common applications where vertical scaling is used:

1. Database Systems

- **Relational Databases**: Systems like MySQL, PostgreSQL, and Oracle often use vertical scaling to improve performance for transactional and analytical

workloads. By adding more resources to a single database server, organizations can handle more complex queries, larger data volumes, and higher transaction rates.

- **In-Memory Databases**: Systems such as Redis and Memcached benefit from vertical scaling as they require large amounts of RAM to store data in memory for rapid access. Scaling up allows these databases to maintain more data in memory, supporting faster processing and reduced latency.

2. Enterprise Resource Planning (ERP) Systems

- ERP systems, like SAP or Oracle ERP, often rely on vertical scaling due to their resource-intensive nature. Since these systems integrate various business processes into a single system, they require substantial computing power, especially in organizations with large-scale operations.
- Scaling up allows the ERP system to manage increased data volumes, more simultaneous users, and more complex processing needs without distributing the load across multiple servers, which can simplify management and reduce latency.

3. Web Servers for Single-Instance Applications

- Some web applications are designed to run on a single server, often due to simplicity or cost-effectiveness. Vertical scaling is useful for such applications, as adding resources to the server can accommodate higher traffic or increased load without changing the application architecture.
- Examples include small to medium-sized e-commerce sites, content management systems (like WordPress), or web applications hosted on virtual private servers (VPS) that need more power to handle growing traffic.

4. Virtualization Hosts

- Virtualization platforms, like VMware ESXi or Microsoft Hyper-V, use vertical scaling to support more virtual machines (VMs) or containers on a single physical host. By adding resources to the host machine, organizations can run more VMs or containers simultaneously, maximizing the use of the hardware.
- This approach is particularly useful in environments where consolidation of workloads onto fewer servers is desired, such as in data centers aiming to reduce their physical footprint and energy consumption.

5. Data Processing and Analytics

- In some cases, data processing applications, like data mining or batch processing, benefit from vertical scaling. For example, a server running Apache Spark for big data processing can be scaled up with more CPU and memory to handle larger datasets or perform more complex calculations faster.

- Vertical scaling is advantageous when the processing framework is optimized to use a single powerful machine, which can reduce latency in data processing tasks.

6. Applications with Licensing Constraints

- Some software applications, particularly those that require expensive licensing per server, may use vertical scaling to maximize performance on a single machine. This allows the organization to avoid additional licensing costs that would come with deploying the application on multiple servers.

7. Legacy Systems and Monolithic Applications

- Older applications or monolithic systems that were not designed for distributed environments often rely on vertical scaling. Adding resources to the existing server allows these legacy applications to meet growing demand without requiring a full-scale rearchitecture to enable horizontal scaling.

Vertical scaling is particularly suited to applications that are single-instance, where centralized processing is beneficial, or where the application architecture does not easily allow for distribution across multiple servers. It can simplify system management, but it does have limitations, as a single machine's capacity is finite. Eventually, horizontal scaling (adding more machines) may be required to handle continued growth.

Describe the applications in which horizontal scaling concept is used

Horizontal scaling, also known as "scaling out," involves adding more servers or machines to a system to distribute the workload across multiple devices. This approach allows a system to handle increasing loads by adding more units rather than increasing the capacity of individual servers. Here are some common applications where horizontal scaling is particularly beneficial:

1. Web Applications with High Traffic

- **Social Media Platforms**: Applications like Facebook, Twitter, and Instagram serve millions of users concurrently, requiring large-scale distributed systems. Horizontal scaling allows these platforms to handle high traffic volumes by distributing user requests across multiple servers, ensuring consistent performance and availability.
- **E-Commerce Platforms**: Sites like Amazon or eBay experience fluctuating traffic, especially during peak times (e.g., holidays, sales events). Horizontal scaling enables these platforms to add more servers dynamically to manage surges in traffic, ensuring fast response times and a seamless shopping experience.

2. Content Delivery Networks (CDNs)

- CDNs, such as Cloudflare or Akamai, rely on horizontal scaling to deliver content (e.g., images, videos, web pages) to users from multiple geographically distributed servers. By distributing the load across various

locations, CDNs can deliver content quickly and efficiently to users worldwide, reducing latency and improving the user experience.

3. Distributed Databases and Data Stores

- **NoSQL Databases**: Systems like Cassandra, MongoDB, and Couchbase are designed for horizontal scaling, allowing them to handle large volumes of unstructured or semi-structured data. These databases can distribute data across multiple nodes, ensuring high availability, fault tolerance, and efficient data processing.
- **Distributed File Systems**: Solutions like Hadoop Distributed File System (HDFS) and Amazon S3 use horizontal scaling to store and manage large datasets across multiple servers. This enables processing of massive amounts of data and supports data redundancy and high availability.

4. Microservices Architectures

- Modern applications built with a microservices architecture are inherently suited to horizontal scaling. Each microservice can be deployed independently on different servers or containers, allowing the system to scale out specific services based on demand. For example, an e-commerce platform might scale the “search” or “checkout” services horizontally during peak times without affecting other services.

5. Cloud Computing and Virtualized Environments

- **Cloud Platforms**: Services like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) enable horizontal scaling by allowing users to spin up multiple instances as needed. Applications in the cloud often leverage auto-scaling features to add or remove instances dynamically based on demand, optimizing resource usage and cost.
- **Container Orchestration Systems**: Platforms like Kubernetes and Docker Swarm use horizontal scaling to manage containerized applications. By adding or removing containers across multiple nodes, these systems can maintain application performance and availability as the load changes.

6. Big Data Processing and Analytics

- **Distributed Data Processing Frameworks**: Systems like Apache Spark, Apache Flink, and Apache Storm are designed for horizontal scaling, enabling the processing of large datasets across multiple nodes. This approach is essential for tasks like real-time analytics, data mining, and machine learning, where processing power and speed are critical.
- **MapReduce**: Horizontal scaling is fundamental to the MapReduce framework, allowing data processing tasks to be distributed across many machines. This is particularly useful for large-scale data analytics, where massive datasets are processed in parallel to extract insights efficiently.

7. Online Gaming and Real-Time Collaboration Platforms

- **Massively Multiplayer Online Games (MMOs)**: Games like World of Warcraft and Fortnite handle thousands of concurrent players by horizontally scaling their game servers. This ensures that players experience minimal lag and that the system can manage varying levels of traffic.
- **Real-Time Collaboration Tools**: Applications like Slack, Zoom, and Google Docs require real-time data synchronization and collaboration among users. Horizontal scaling allows these platforms to add more servers as the number of concurrent users grows, ensuring that all users experience smooth interactions without latency.

8. IoT (Internet of Things) Platforms

- IoT platforms often need to process data from millions of devices, which generate vast amounts of data in real time. Horizontal scaling allows these platforms to add more servers as the number of devices grows, ensuring that they can handle the influx of data and process it efficiently for tasks like monitoring, analytics, and real-time decision-making.

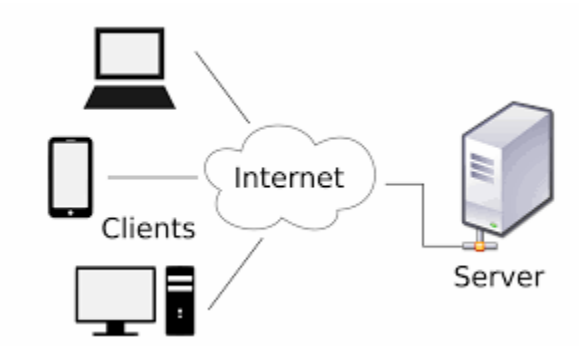
9. Load Balancing and High Availability Systems

- Horizontal scaling is essential for systems requiring high availability and fault tolerance. By distributing traffic across multiple servers, load balancers ensure that the system can handle failures without impacting user experience. This approach is common in mission-critical applications like banking, healthcare, and telecommunications, where downtime can have severe consequences.

Horizontal scaling is ideal for applications that require high availability, can benefit from distributed processing, or must support large and dynamic workloads. By adding more servers, horizontal scaling provides resilience, redundancy, and the flexibility to scale resources according to demand. This approach enables organizations to handle growth effectively and maintain performance and reliability.

SYSTEM DESIGNING IN NETWORKING

A computer network is a collection of interconnected devices that share resources and information. These devices can include computers, servers, printers, and other hardware. Networks allow for the efficient exchange of data, enabling various applications such as email, file sharing, and internet browsing.



IP ADDRESS

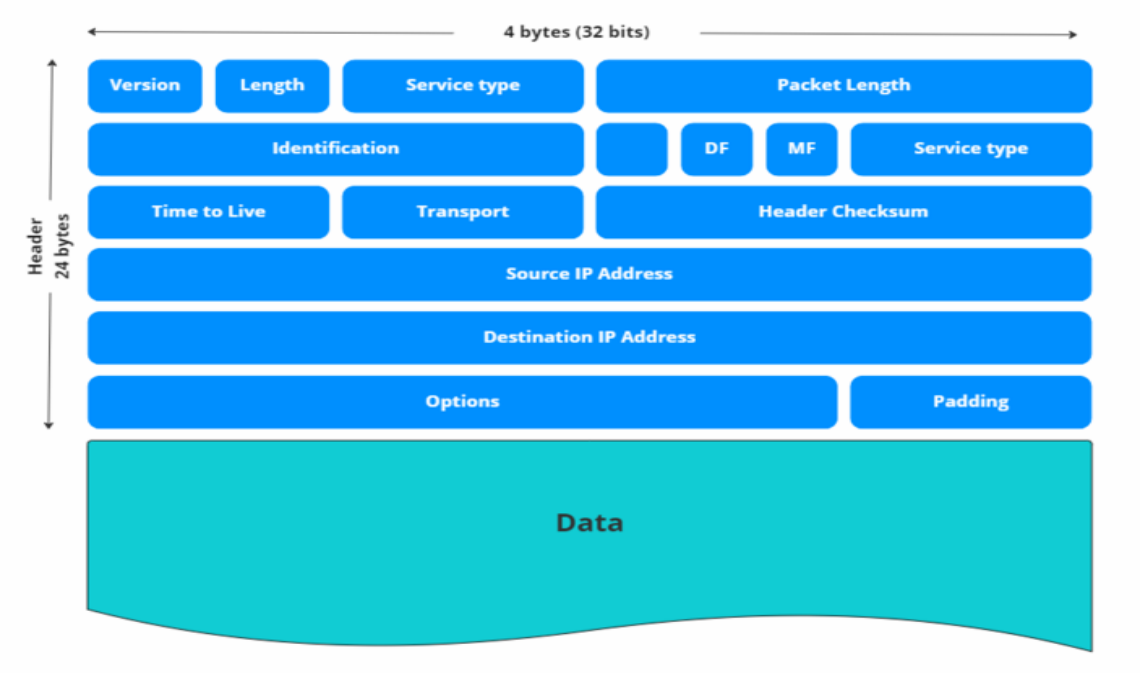
An IP address is a unique address that is used to identify computers or nodes on the internet. This address is just a string of numbers written in a certain format. It is generally expressed in a set of numbers for example 192.155.12.1. Here each number in the set is from 0 to 255 range. Or we can say that a full IP address ranges from 0.0.0.0 to 255.255.255.255. And these IP addresses are assigned by IANA(known as Internet Corporation For Internet Assigned Numbers Authority). With 32 bit, 4 billion distinct IP address can be stored.

But what is Internet protocol? This is just a set of rules that makes the internet work. You are able to read this article because your computer or phone has a unique address where the page that you requested.

IPV4 – IP Address: a whole IPv4 binary address can be represented by **32-bits** of binary digits. In IPv4, a unique sequence of bits is assigned to a computer.

IPv6: But, there is a problem with the IPv4 address. With IPv4, we can connect only the above number of 4 billion devices uniquely, and apparently, there are much more devices in the world to be connected to the internet. So, gradually we are making our way to **IPv6 Address** which is a **128-bit IP address**. In human-friendly form, IPv6 is written as a group of 8 hexadecimal numbers separated with colons(:). But in the computer-friendly form, it can be written as 128 bits of 0s and 1s.

The 2 systems in the internet can communicate using IP- Internet protocol and the data exchanged between the different computers are called packets.



- **Version:** This field indicates the IP protocol version, such as IPv4 or IPv6, which helps determine how the packet should be routed.
- **Total length:** Specifies the packet's total length of the data payload (the message) and the header, which are essential for the recipient to understand the packet's contents.
- **Protocol:** Determines the high-level protocol, like TCP, UDP, etc., used in the data payload.
- **Time to Live (TTL):** Used to prevent packets from circulating indefinitely, which could cause network congestion. Every time a packet gets forwarded, the router decreases the TTL by one, and when the TTL becomes zero, the packet gets discarded, helping to avoid congestion.
- **Source IP address:** Specifies the IP address of the device that sent the packet. It's used to reply to the sender, and if a router cannot send the packet to the following location, it sends a message to the source using the source IP address to indicate the problem.
- **Destination IP address:** Identifies the IP address of the intended recipient of the packet. Routers use it to determine where to forward the packet to reach its final destination.

The IP protocol can send piece 1 information using HTTP requests to the server, but when several piece of data in terms of multiple packets are to be sent and received from the server, then it will be difficult to tracks all the packets from client computer to destination computer and reassembling the packets at the destination computer is a challenge. To overcome this problem, TCP protocol was introduced.

The TCP header section can be added in the IP packets which holds the serial number of the packet, helps in reassembling the packets at the destination system (server).

Classification of IP Address

An IP Address is basically classified into two types:

- Private IP Address
- Public IP Address

What is a Private IP Address?

The Private IP Address of a system is the IP address that is used to communicate within the same network.

What is a Public IP Address?

The Public IP Address of a system is the IP address that is used to communicate outside the network.

What is Static IP Address?

A Static IP address is an IP address that does not changed frequently or constantly it is reserved for a specific computer or device.

Assigning Static IP address is common in servers, network devices or any device that has to have a fixed address that can be accessed from a distance.

What is Dynamic IP Address?

A Dynamic IP address is an IP address which is changed from time to time.

PORT NUMBER:

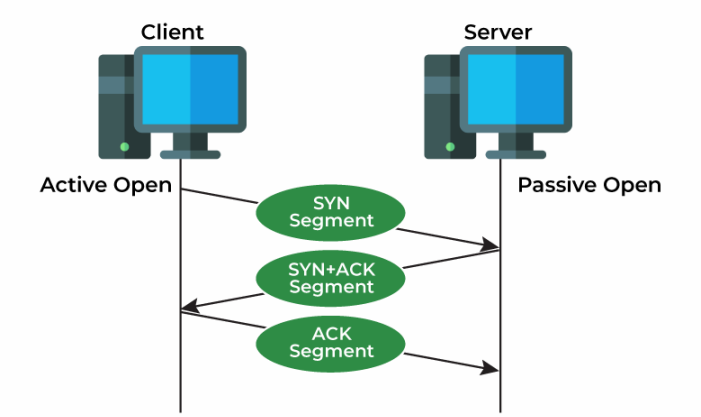
A port is a virtual point where network connections start and end. Ports are software-based and managed by a computer's operating system. Each port is associated with a specific process or service.

Ports are standardized across all network-connected devices, with each port assigned a number. Most ports are reserved for certain protocols — for example, all Hypertext Transfer Protocol (HTTP) messages go to port 80. While IP addresses enable messages to go to and from specific devices, port numbers allow targeting of specific services or applications within those devices.

Transmission Control Protocol (TCP)

TCP (Transmission Control Protocol) is one of the main protocols of the Internet

protocol suite. It lies between the Application and Network Layers which are used in providing reliable delivery services. It is a connection-oriented protocol for communications that helps in the exchange of messages between different devices over a network. The Internet Protocol (IP), which establishes the technique for sending data packets between computers, works with TCP.



Transmission Control Protocol

Features of TCP

- TCP keeps track of the segments being transmitted or received by assigning numbers to every single one of them.
- Flow control limits the rate at which a sender transfers data. This is done to ensure reliable delivery.

- TCP implements an error control mechanism for reliable data transfer.
- TCP takes into account the level of congestion in the network.

Applications of TCP

- **World Wide Web (WWW)** : When you browse websites, TCP ensures reliable data transfer between your browser and web servers.
- **Email** : TCP is used for sending and receiving emails. Protocols like **SMTP** (Simple Mail Transfer Protocol) handle email delivery across servers.
- **File Transfer Protocol (FTP)** : FTP relies on TCP to transfer large files securely. Whether you're uploading or downloading files, TCP ensures data integrity.
- **Secure Shell (SSH)** : SSH sessions, commonly used for remote administration, rely on TCP for encrypted communication between client and server.
- **Streaming Media** : Services like Netflix, YouTube, and Spotify use TCP to stream videos and music. It ensures smooth playback by managing data segments and retransmissions.

Advantages of TCP

- It is reliable for maintaining a connection between Sender and Receiver.
- It is responsible for sending data in a particular sequence.
- If the data packets are lost then TCP can resend the lost packets.
- Its operations are not dependent on Operating System .
- It allows and supports many routing protocols.
- It can reduce the speed of data based on the speed of the receiver.

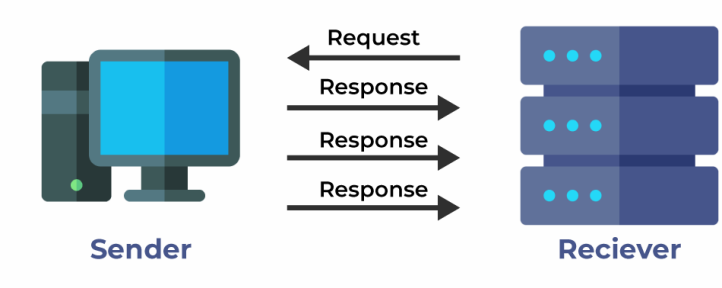
Disadvantages of TCP

- It is slower than UDP and it takes more bandwidth.
- Slower upon starting of transfer of a file.
- Not suitable for LAN and PAN Networks.
- It does not have a multicast or broadcast category.
- It does not load the whole page if a single data of the page is missing.

User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as the UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection before

data transfer. The UDP helps to establish low-latency and loss-tolerating connections establish over the network. The UDP enables process-to-process communication.



User Datagram Protocol

Features of UDP

- Used for simple request-response communication when the size of data is less and hence there is lesser concern about flow and error control.
- It is a suitable protocol for multicasting as UDP supports packet switching .
- UDP is used for some routing update protocols like RIP(Routing Information Protocol) .
- Normally used for real-time applications which can not tolerate uneven delays between sections of a received message.

Application of UDP

- **Real-Time Multimedia Streaming** : UDP is ideal for streaming audio and video content. Its low-latency nature ensures smooth playback, even if occasional data loss occurs.
- **Online Gaming** : Many online games rely on UDP for fast communication between players.
- **DNS (Domain Name System) Queries** : When your device looks up domain names (like converting “www.example.com” to an IP address), UDP handles these requests efficiently .
- **Network Monitoring** : Tools that monitor network performance often use UDP for lightweight, rapid data exchange.
- **Multicasting** : UDP supports packet switching, making it suitable for multicasting scenarios where data needs to be sent to multiple recipients simultaneously.
- **Routing Update Protocols** : Some routing protocols, like RIP (Routing Information Protocol), utilize UDP for exchanging routing information among routers.

Advantages of UDP

- It does not require any connection for sending or receiving data.

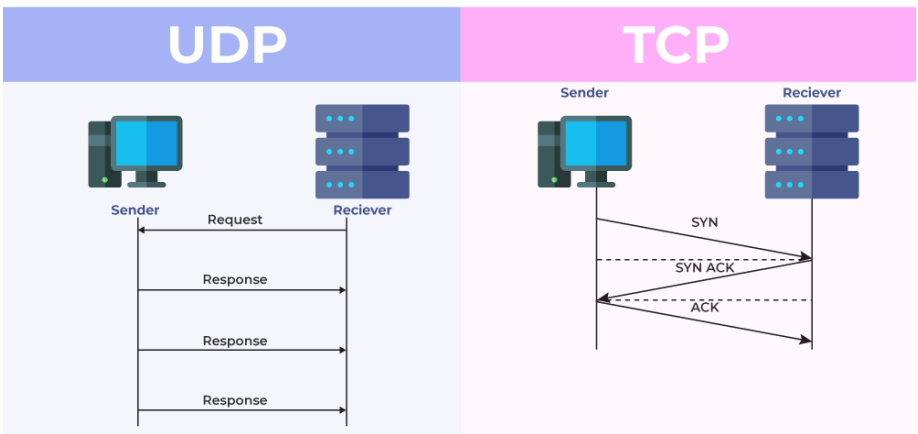
- Broadcast and Multicast are available in UDP.
- UDP can operate on a large range of networks.
- UDP has live and real-time data.
- UDP can deliver data if all the components of the data are not complete.

Disadvantages of UDP

- We can not have any way to acknowledge the successful transfer of data.
- UDP cannot have the mechanism to track the sequence of data.
- UDP is connectionless, and due to this, it is unreliable to transfer data.
- In case of a Collision, UDP packets are dropped by Routers in comparison to TCP.
- UDP can drop packets in case of detection of errors.

Which Protocol is Better: TCP or UDP?

The answer to this question is difficult because it totally depends on what work we are doing and what type of data is being delivered. UDP is better in the case of online gaming as it allows us to work lag-free. TCP is better if we are transferring data like photos, videos, etc. because it ensures that data must be correct has to be sent. In general, both TCP and UDP are useful in the context of the work assigned by us. Both have advantages upon the works we are performing, that's why it is difficult to say, which one is better.



Difference Between TCP and UDP

Where TCP is Used?

- Sending Emails
- Transferring Files
- Web Browsing

Where UDP is Used?

- Gaming
- Video Streaming
- Online Video Chats

The Domain Name System (DNS)

is a hierarchical, distributed database system that translates human-readable domain names (like www.example.com) into IP addresses (like 192.0.2.1) that computers use to identify each other on the network. DNS is a core component of the internet, enabling users to access websites and other resources using easy-to-remember names instead of numerical addresses. The system design aspects of DNS cover its hierarchical structure, components, types of DNS servers, resolution process, and fault tolerance mechanisms.

1. Hierarchical Structure

- **Domain Name Space**: DNS uses a hierarchical namespace divided into multiple levels, where each level represents a part of the domain name (e.g., com, example, and www).
- **Root Level**: At the top of the hierarchy, the root level is represented by a dot (.), though it's usually implicit and not typed out.
- **Top-Level Domains (TLDs)**: Below the root are TLDs like .com, .org, .net, and country code TLDs (ccTLDs) like .uk, .jp.
- **Second-Level Domains**: These are directly below TLDs (e.g., example in example.com).
- **Subdomains**: Domains under the second-level domain (e.g., www in www.example.com).

2. Key Components

- **DNS Zones**: A zone is a part of the DNS namespace managed by an organization. Each zone can contain multiple domains and subdomains.
- **Resource Records (RRs)**: DNS stores different types of resource records, such as:
    - **A (Address) Record**: Maps a domain name to an IPv4 address.
    - **AAAA Record**: Maps a domain name to an IPv6 address.
    - **CNAME (Canonical Name) Record**: Maps a domain name to another domain (used for aliases).
    - **MX (Mail Exchange) Record**: Specifies the mail servers for a domain.

- **NS (Name Server) Record**: Specifies the authoritative name servers for a domain.
- **TXT Record**: Holds arbitrary text data, often used for domain verification.

4. DNS Resolution Process

- **Recursive Query**: A client (e.g., a web browser) sends a DNS query to a recursive resolver. If the resolver doesn't have the answer cached, it will query the root server, then TLD server, and finally the authoritative server.
- **Iterative Query**: In iterative queries, each DNS server provides the best answer it knows. The client may need to query multiple servers sequentially, following the chain down the hierarchy.
- **Caching**: DNS resolvers cache responses to reduce latency and decrease the number of queries to upper-level servers. The Time-to-Live (TTL) value specifies how long the information is stored.

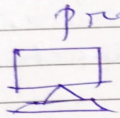
5. Fault Tolerance and Load Balancing

- **Redundancy**: DNS is highly redundant. Multiple DNS servers (e.g., root servers, TLD servers) are available worldwide to handle queries, ensuring no single point of failure.
- **Caching**: Caching increases the resilience of DNS by reducing the load on authoritative servers and providing quick responses to repeated queries.
- **Round-Robin Load Balancing**: DNS can be configured to provide multiple IP addresses for a single domain. Each request may get a different IP, distributing the load across multiple servers.
- **Anycast**: Used by DNS servers to route queries to the nearest or best-performing server, improving response times and reliability.

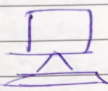
DNS is designed to be scalable, resilient, and fast, with multiple layers of caching, redundancy, and a hierarchical structure that allows for easy management of the vast number of domain names on the internet.

HTTP protocols

→ It is an application protocol

→ 

Client —



Server

→ Client can be another Server, it is not always an end user

→ many times servers communicate with each other

→ one of the server can take the role of the client and another server could be the actual server which will be responding to the requests

RPC – Remote procedure call – helps in moving data between the machines (Client & Server)

Here Remote means that 2 machines communicating with each other are actually Seperated by the network Just like function call.

→ So Here the Client machine runs the code that is actually in server, which is done by RPC.

→ The Code looks like it is running from in the Client machine, but will be actually a network call

Ex 

Client

server

Ex listVideo() – is a function called by the client, but the code for this function is actually written on different machine, usually servers.

HTTP → A Protocol of Internet

Ip – Internet protocol (helps to move the data between the machines, It is made more reliable using Tcp protocol).

→ HTTP is a an application layer protocol that is built on Ip and Tcp.

→ HTTP is a Request & Response protocol.

→ Client makes a request and server response, using HTTP.
→ As HTTP is built on top of TCP/IP, every thing is handled by the HTTP requests and responses the client and server may not know each other and may be in different Countries.
→ HTTP3 is the latest version.
→ HTTP requests contains.
ⓐ Header
ⓑ body of request
Header – 3 Groups.
ⓐ General Header.
→ This Contains the URL path
→ Also contains Request Method which is "GET", other most commonly used methods are POST, DELETE & PUT
→ Status Code – Green Dot shows that it was successfull request
→ Header also Contains Request header – & the Response header.

Request header – is what the client has actually requested to server. It has accept field, which says in which form the data from the server is accepted usually it will be in HTTP format.

Response header – Contains what Content Type was actually returned.

→ Every Request has to Go to some URL. Ex – youtube.com
→ this in turn is served by multiple HTTP Methods Ex for HTTP Methods are
ⓐ GET
ⓑ POST
ⓒ PUT
ⓓ DELETE
ⓐ Youtube.com will be an End point for GET Method
→ With GET Method, it returns the HTML page of youtube.com. GET Method retrieves Info.
ⓑ POST → Creates user Info. like creating user account or uploading a vedio in youtube, or creating a comment in youtube, so POST Method is meant for creating. Some info Creating on the server, is done by POST Method.
→ GET Requests dont have the body, they only has headers, but other methods like POST, PUT, DELETE has Got body which helps in passing an Information.

PUT Requests/Methods
→ Helps in updations.
PUT, POST DELETE performs CRUD operations.

Corresponds.
C - Create → POST
R - Read - GET
U - Update - PUT
D - Delete - DRlete.

Status Codes:
If the client requested the server, but due to some reason, the server was failed to respond / fulfill the response.

Informational Responses
— 100-199
Successfull Responses
— 200-299
Redirection message
— 300-399
Client Error Responses
— 400-499
Server Error Responses
500 - 599
400 - Bad request
401 - unauthorized Access
404 - Resource not found.
500 - Internal service Error
502 - Gateway error
503 - service unavailable

SSL/TLS → HTTS protocol work with this Layer.
HTTPS - Hyper text transfer protocol (Secure)
SSL - Secure socket Layer.
SSL is outdated, TLS is widely used
TSL - Transport Layer Security

HTTPS - is prone to Man in the middle attacks.
→ In the process of requests and responses between the client and server, using HTTPS, with the help of SSL/TLS, that data or requests and Responses are actually Encrypted.

Ex Client sends App pass-word as 1234 (actual password). It is encrypted and sent to server as Xyz, server decrypts and understands that the actually password is 1234

this Encryption is achieved using SSL/TLS and HTTPS achieves the secure exchange of data between client & server by implementing SSL/TLS.

→ using all these Concepts we can build very powerful applications.