Introduction to System Design

System Design is one of the most critical steps in software development. System Design affects all significant steps from software implementation to its deployment.

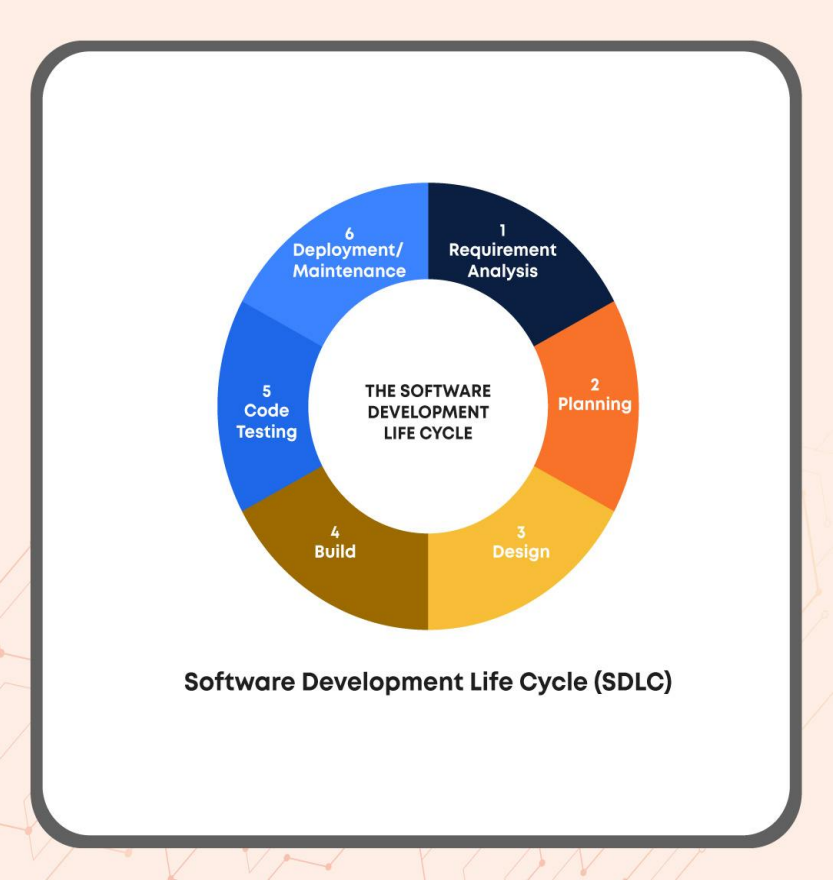
What is system design?

System Design is the process of creating the architecture for different components, interfaces and modules of the system and providing it with data helpful in implementing the system elements.

It is the way toward characterising, creating, and planning frameworks that fulfil the necessities and prerequisites of a particular business or association.

It is the stage where the SRS report is changed over into a component division architecture that can be executed and decides the resulting framework execution.

So, let's look at the various phases of the life cycle of software :



Software Development Life Cycle (SDLC)

The steps are as follows:

1. Requirement Analysis:

Functional and Non-Functional requirements are collected by interacting with the stakeholders and users using different requirement elicitation techniques.

2. Planning/Analysis:
   The requirements are defined and documented after further analysis.
3. System Design:
   Architecture is created for the system and its components (High-Level design and Low-Level design)
4. Coding/Implementation:
   The software is built using suitable programming languages and technologies.
5. Testing:
   The quality of the software is evaluated.
6. Deployment/Maintenance:
   Software is prepared for release.

System designing is the backbone of software development, and all the further steps directly depend on the design prepared. It is crucial for any person indulging in software development roles to understand and master them. System Design is also frequently asked in interviews for primary product-based companies like Facebook, Google, and Amazon. The system design involves identifying the sources of data, the nature and type of data available.
For example: In order to create a salary calculation app, there is a need to use inputs, such as attendance, leave details, additions or reductions etc. This helps to understand what kind of data is available and who provides it to the system so that the system can be designed taking into account all relevant factors.

Now since we have understood the significance of the subject, let's dive deep into it.

The objective of System Design

1. Practicality:
   System designing is crucial in making a system that suits the needs of target users in real-time scenarios.
2. Correctness:
   The system must satisfy all the functional and non-functional requirements of the stakeholders and users.
3. Completeness:
   The system should be complete in all terms, from components to functionality required and specified in the SRS (Software Requirement Specification).
   In addition, the system design leads to ensuring that the system is designed so that it meets the needs of the users and makes it user-oriented.
4. Efficiency:

The system should be resource-effective, and neither surpasses the cost nor fail to deliver any functionality. Another important goal of system design is concerned with creating a more efficient system to provide the required output and response time within the allotted time. In this way, the system design section ensures the safety and efficiency of the system.

5. Flexibility:
   The system should be able to adapt to the changing needs of the user. The other objective of this phase is to aim to build such a system that can be environmentally friendly and respond to changes if necessary.
6. Optimisation:
   It helps you understand how to optimize time (latency) and space (memory) for the individual components to run the entire system.
7. Reliability:
   System design helps to make software reliable. Reliability is defined as the failure-free system in a particular environment (or specific input cases) for a particular period of time.
   The system design also helps to achieve fault tolerance. **Fault tolerance** is the ability of the software to continue operating when one or two of its components stop working.

Advantages of System Design

1. Awareness of the full stack of the computer science i.e. API protocols, networking, databases, communication paradigms.
2. Increases efficiency and consistency.
3. The speedy software development process
4. Fewer mistakes, therefore, saves time and resources.
5. The actual procedure, therefore, saves resources.
6. Simpler and uncomplicated.
7. Reduces designing cost.

Monolithic Architecture

What is Monolithic Architecture?

The dictionary meaning of monolithic is “formed of a single large block”. Monolithic architecture defines the unified model in which the complete application is deployed as a single unit/system in system design.

In other words, if all the components and functionalities of a project are entangled and combined in a single codebase, then that is a monolithic application.

Monolithic Architecture has less complexity => Easier to understand => Higher productivity.

The web applications have different layers as follows:

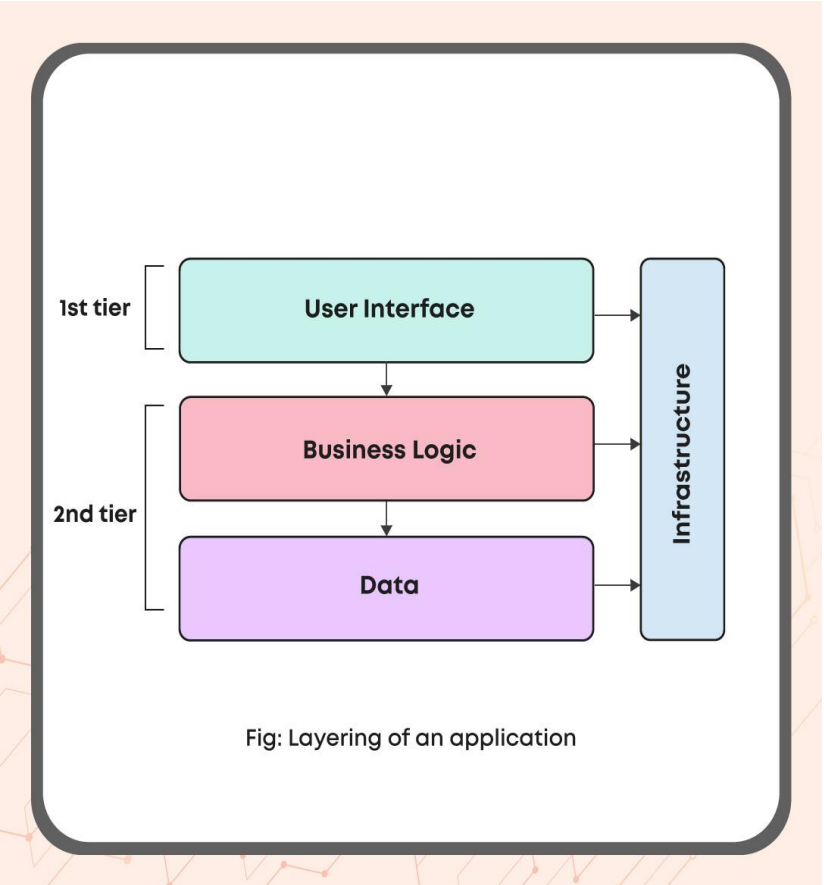


Fig: Layering of an application

1. User Interface (Front-End):
   The user interface can be a webpage or a desktop application. It is the closest layer to the target user using which the user can get things done. It takes the user’s request, interacts with the server, and displays the user’s result.

2. Business logic (Domain/Back-End):
   It comprises all the logic behind the application. The application server includes the business logic, receives requests from the native client, acts on them, and persists the data to storage.

3. Data Interface:
   It includes the data persistence mechanism (DB) or communication with other applications. It comprises the DataBase, message queues, etc. A database server that would be used by the application server for the persistence of data.

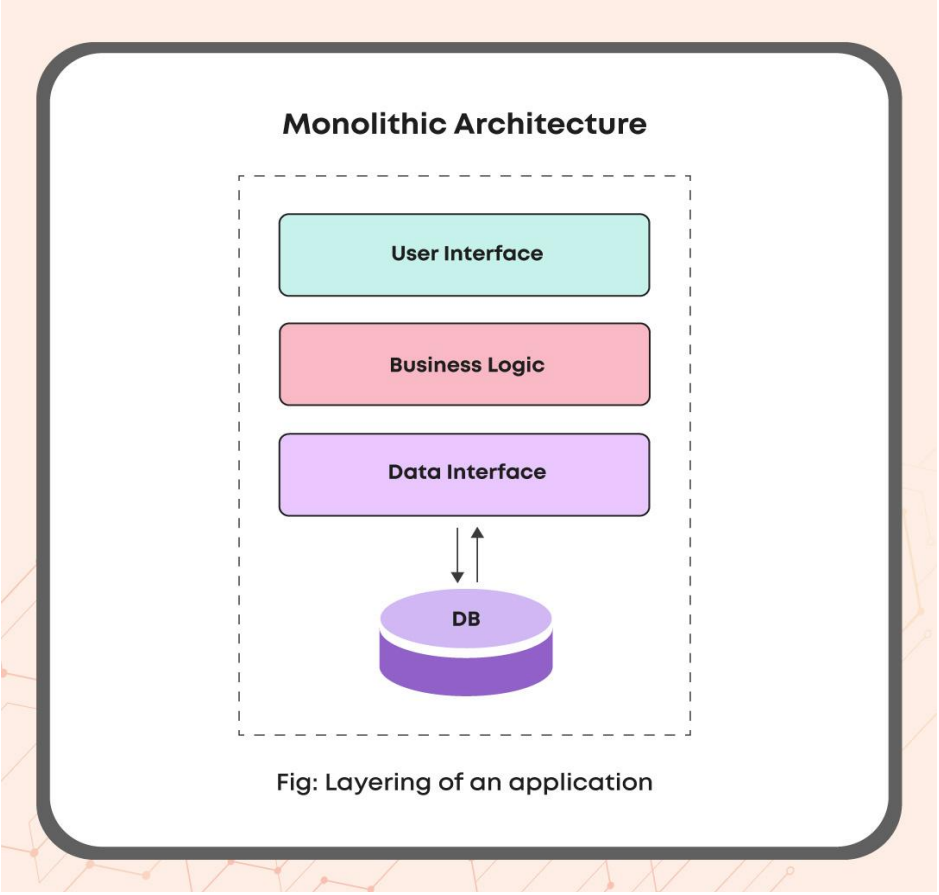


Fig: Layering of an application

Fig: All the layers of a system are combined in a single program.

Example:

Suppose we want to create a monolithic architecture for a medical system. The system's main components would be patients, doctors, wards, appointments, and the hospital unit. And the logical relationship would be that doctors give appointments to patients, and they visit the ward if required where the doctor treats the patient in the hospital.

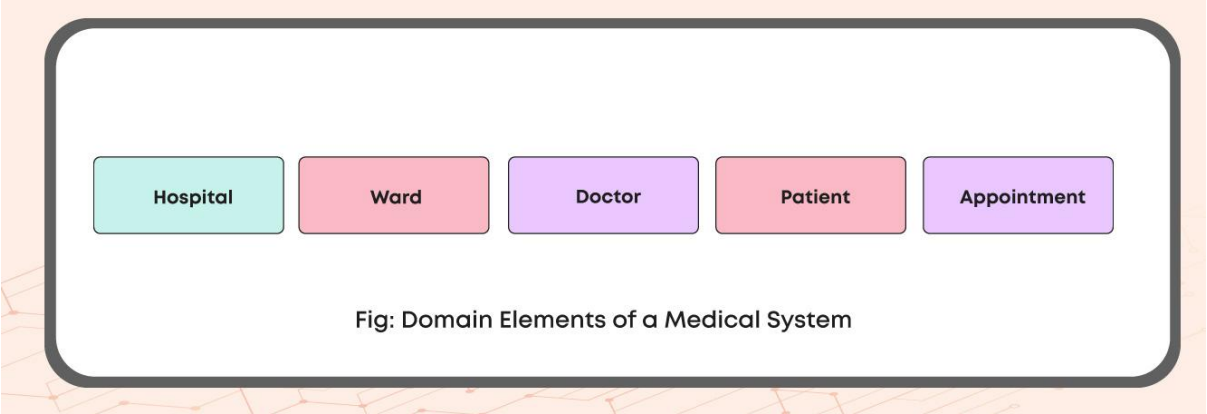


Fig: Domain Elements of a Medical System

The application would have a Frontend where the appointments would be booked or doctor's availability would be checked, and a backend where the user's requests would be entertained and dynamic results provided.

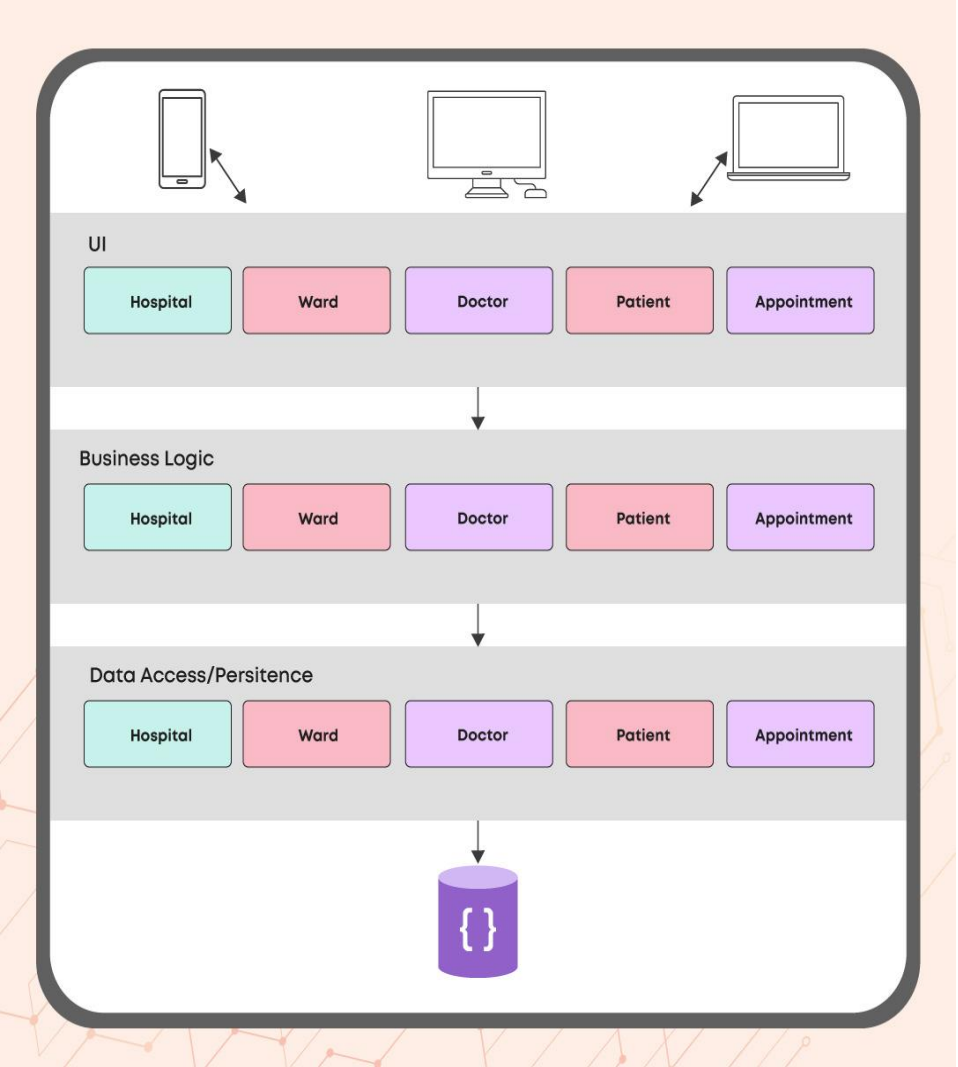


Figure — Monolithic Layered Architecture of a Medical System.

Benefits of monolithic architecture

1. Easy development:
   Monolithic architecture is the traditional model for design and therefore doesn't require many skilled developers due to less complexity.
2. Easy deployment:
   Since all the components are stored in a single repository, it is easy to handle, and only one directory has to be cared about.
3. Easy testing:
   The model is closely encapsulated. Therefore testing is end to end and faster.
4. Less cross-cutting problems:
   Cross-cutting issues are the ones that affect the whole of the system, like caching, logging and handling. Lesser issues because the single unit can be better handled.

Disadvantages of monolithic architecture

1. Difficult to understand because all the code is stored in a single directory.
2. It is challenging to introduce a change or replace a framework since every element is tightly encapsulated.
3. Making any changes would be possible only by redeployment of the complete software application.
4. Less scalable because each element has different scalability requirements.
5. Less reusability.

Microservices Architecture

What are microservices?

Microservices architecture is an evolved variant of service-oriented architecture that promotes software components to be loosely coupled. It is the most granulated type of architecture design and every service is completely independent of the other.

Features of microservices:

1. All the software components must be completely independent of each other.
2. Every service must deliver only one function.

When should microservices architecture be preferred?

A. When an application needs to be rebuilt due to a change in functionalities, the addition of new features
B. Big data applications require dedicated services for tasks such as data collection, processing, delivery etc
C. Applications requiring real-time data and executing different operations to deliver output immediately such as traffic control system

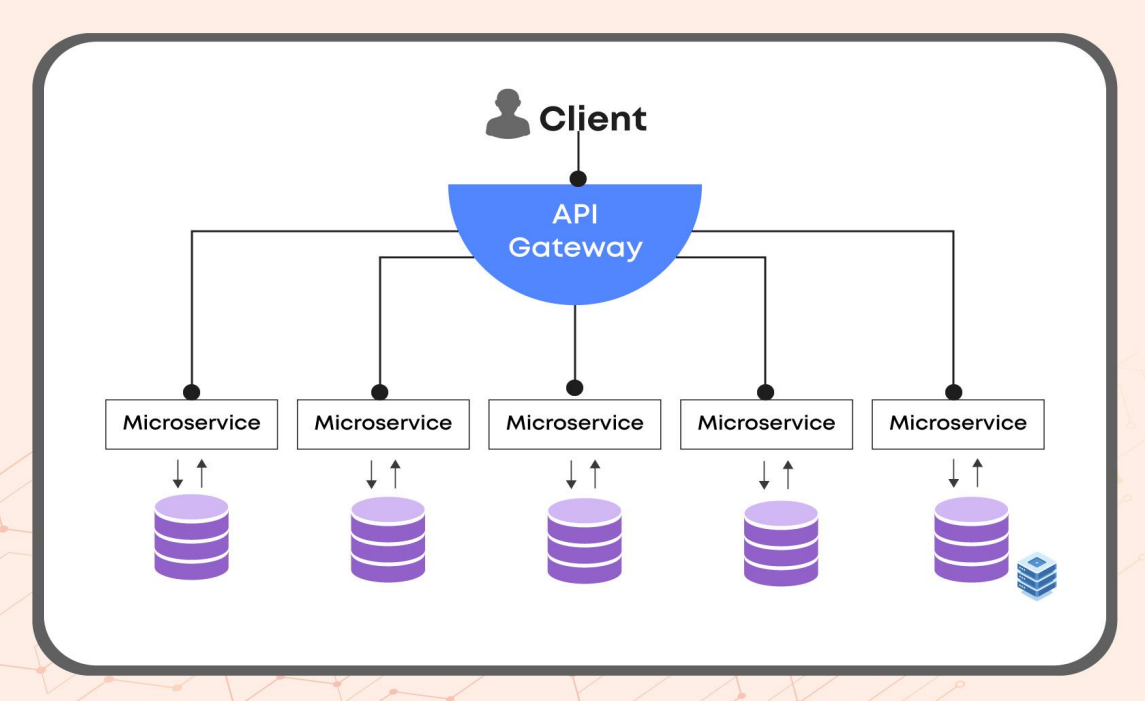


Fig: Microservices architecture

Differences between Service-Oriented Architecture and Microservices Architecture

Parameter	Service-Oriented Architecture	Microservices Architecture
Promotes	Software components reusability	Granulated and completely independent services
Data Storage	Can share the data storage	Each microservice has separate and independent data storage
Dependency	Different layers are dependent	Every data layer is independent
Size	Software size is large	Software size is comparatively smaller.
Communication	Use ESB(Enterprise Service Bus)	Use messaging system

Disadvantages of Microservices Architecture:

A. Complicated testing as distributed environment
B. Requires load balancing so that network latency can be decreased
C. Creating a lot of services for complex applications can lead to confusion between developers
D. Higher operational cost than monolithic architecture

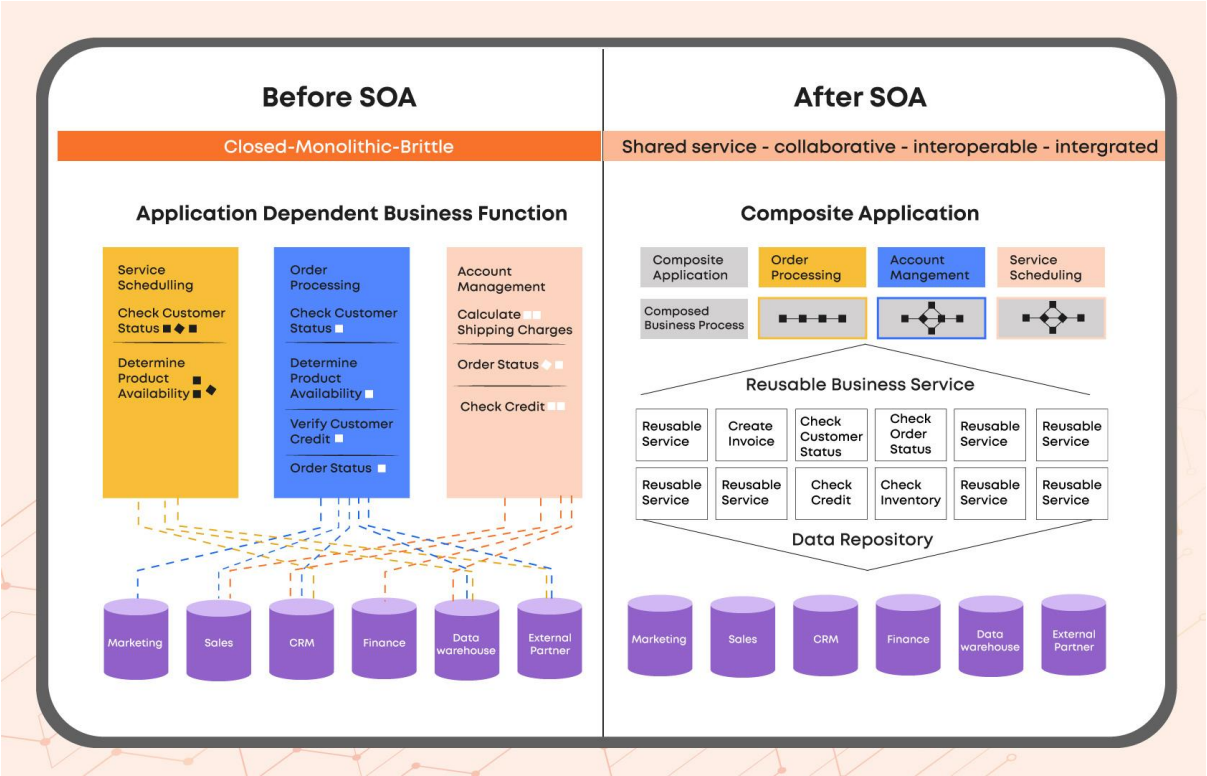
Advantages of Microservices Architecture:

A. As the services are independent of each other, different teams can independently develop, test and put the code into production
B. Flexibility to try new languages and technologies
C. No Single Point Of Failure(SPOF)

Service-Oriented Architecture

Definition:

Service-Oriented Architecture(SOA) is a style of architecture that promotes loose coupling and granular applications to make the components of the software reusable.



Example: In a ticket booking application, the main components would be

- the Booking service
- Authentication service
- Inventory management service
- Payment and confirmation notification service.

While booking a ticket, the booking service will make a request to the authentication service and therefore the booking service would act like a service requester and the authentication service as the service provider. Similarly, the booking service will make a request to the inventor service to check the availability of seats etc. All the services are separate components, separate code bases and are separately deployable. Therefore, this is an example of SOA architecture where different components are separate but communicate with each other to deliver common services.

Advantages of Service-Oriented Architecture:

1. Agile: All the components of Service-oriented architecture are separate so individual development on the components is feasible. Therefore, the time to go to production is less.
2. Selective scaling: Since the services are separated, the individual components scaling or selective scaling is possible.
3. Different tech stacks allowed: Different tech stacks and technologies can be used for the development of different components since all the services are separate components, separate code bases and are separately deployable.
4. Big size teams: Separate teams can be formed for different components of the same project without any conflict of management.
5. Loose Coupling: Leads to increased reusability of the software components.
6. Easier Debugging: Separation of concerns provides modularity and easier debuggability of individual applications
7. If properly designed, it can also lead to a contained blast radius in case a particular part of the system goes down

Disadvantages of Service-Oriented Architecture:

1. Higher Latency: Service-oriented architecture based applications require an additional network call between the service requester and the service provider leading to network delays/lags and therefore increasing latency.
2. Difficult testing: Testing is difficult because of moving boundaries since the software components are dynamic and changing.
3. Security: Security is complex because the software is highly granulated and loosely coupled so security must be ensured for each individual service.
4. Confused Developers: If the developers do not have knowledge of this architecture style this may lead to confusion leading to lower productivity.
5. Cascading Failures: Can lead to cascading failures in case of improperly planned interaction between multiple applications
6. Complex Understanding: This can lead to a difficult understanding of the whole system by a single person if all services developed in isolation

DISTRIBUTED SYSTEMS

What is a Distributed System?

A distributed system is a collection of multiple individual systems connected through a network that share resources, communicate and coordinate to achieve common goals.

A distributed system has multiple benefits over monolithic architecture, like higher scalability and solving the problem of SPOF(Single Point Of Failure).

Redundancy/Replication can help in avoiding the issue of data loss. In a distributed system environment since nodes can be geographically distributed, a node can crash due to any reason such as natural disasters, power failures or scalability issues. Since the data is replicated on multiple servers, these replicas provide the backup and can take over, avoiding the single point of failure for data.

Example: Telecommunication Networks, Real-Time Distributed Systems.

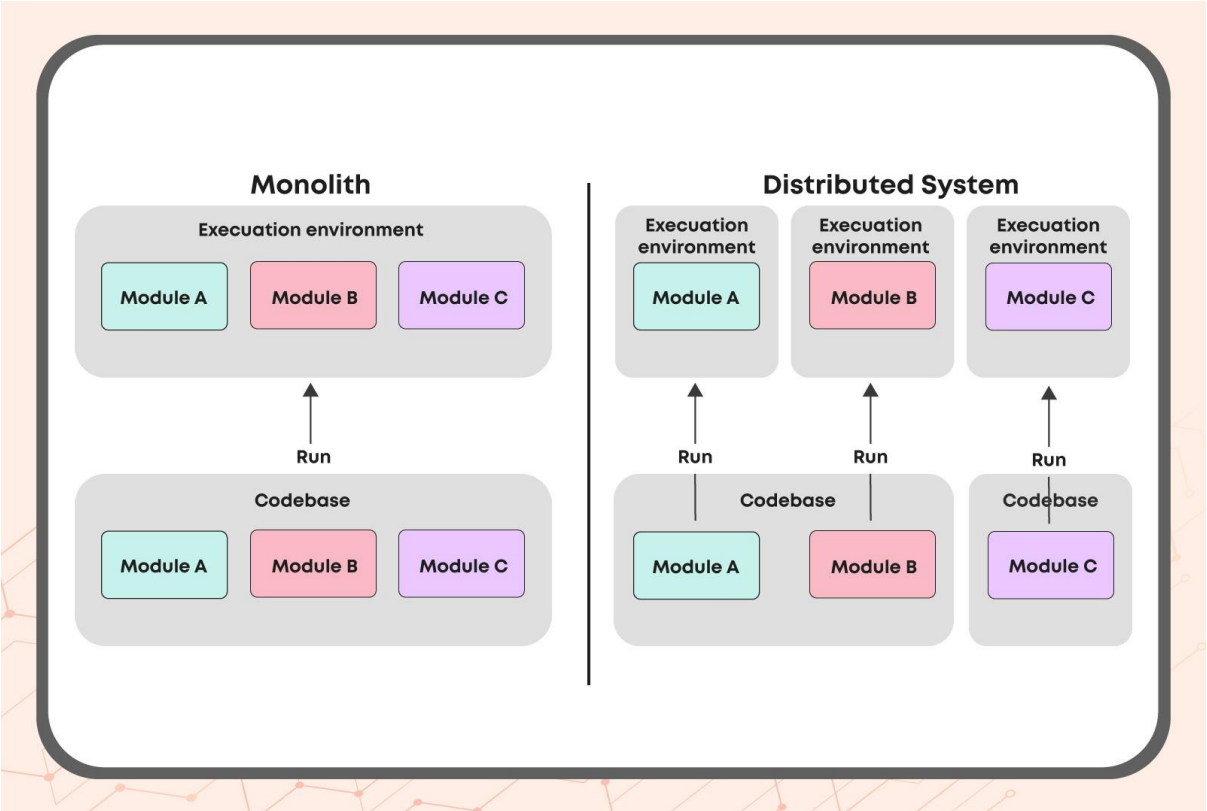


Fig: Monolithic architecture vs distributed system

Benefits of Distributed System

1. Low latency:

Distributed systems have higher speed because they can have multiple servers and the location of servers is close to the target users, thereby reducing the query time.

2. Scalability:
   Distributed systems allow higher scalability. Since it is a collection of independent machines, horizontal scaling of individual servers is possible.
3. Reliability:
   In contrast to monolithic architecture, distributed systems solve SPOF(Single Point Of Failure). Even if some individual units fail to work, the system would still be up and running. This increased reliability, and the system remains operational most of the time efficiently.

Disadvantages of Distributed System

1. Consistency:
   The higher number of devices and network complexities may make it difficult to synchronise the application states and manage data integrity.
2. Network Failure:
   Distributed systems communicate and coordinate using the network calls. During a network failure, it can lead to the transfer of conflicting information or even communication failure. This leads to poor overall system performance.
3. Complexity:
   Although distributed systems are highly scalable, the increase in the number of network points and hardware leads to a rise in the complexity of the complete system. It becomes challenging to manage the entire system.
4. Management:
   Additional functionality like load balancing(the process of distributing the load among various nodes of a distributed system), logging, and monitoring is required to manage the system and prevent failures.

Latency

We all have encountered multiple websites or web applications with high loading times like excessive buffering while streaming a video. These interruptions lead to a poor user experience. In short, this loading time is latency. In case of higher latency, it tends to slow the website load time to crawl and take up a lot of time before loading the complete web page. It is clear that a user would prefer a low latency website with faster responses.
It is very clear how latency affects the user experience and therefore largely impacts the system design. Let's understand the term better.

What is latency?

In short, latency means delay.

- Latency is the delay between a user's request and the web application's response to the corresponding action.
- It is the total time taken for a round trip of a data packet from one destined location to another.

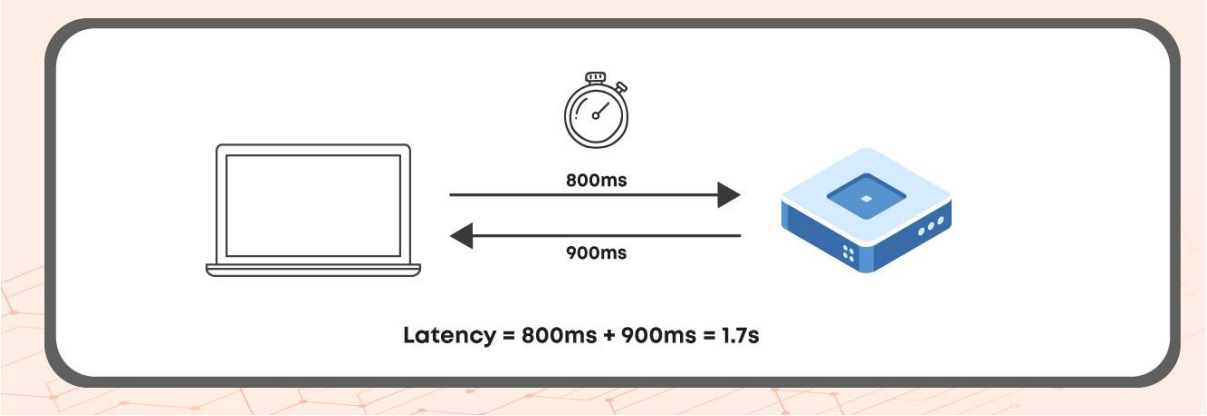


Fig: Example showing latency calculation (sum of time taken for round trip)

The unit of latency is milliseconds(ms) since it is a measure of time.

Main components that affect latency

Latency is majorly due to the Network Delays(for a distributed system) and the Computational Time.

These are the main components that affect latency:

1. Transmission media:
   Latency depends on the type of media that is being transmitted.
2. Packet size:
   Smaller the packet size, the faster the transmission. Packet size is directly proportional to latency.
3. Packet loss:
   Latency can be adversely affected by a huge loss of packets during transmission or different rates of transmission of each packet.
4. Signal strength:
   Poor signal increases latency. Latency is indirectly proportional to signal strength.
5. Propagation delays:
   Latency also depends on the distance between two communicating nodes. Higher the distance more the latency.
6. Other computer and storage delays:
   Retrieving and processing the stored information consumes time. More the time in accessing stored information greater the latency.

Latency and Architecture

Latency depends on two factors majorly:

1. Network Delays
2. Computational Delays

Monolithic architecture

Network latency is zero because there are no calls over the network.
All the calls in monolithic architecture are local.

Latency = Computational Delay ~~+ Network Delay~~ (zero)

Distributed Systems

During a call in a distributed system, signals are sent over different networks and transmitted back. This further adds network latency.

Latency = Computational Delay + Network Delay

Therefore we can conclude that,

Latency (Distributed System) > Latency (Monolithic System)

Importance of Latency

1. Latency has a significant impact on User Experience and User Satisfaction. User satisfaction increases with a decrease in response time.
2. Latency is also a very important factor for latency-sensitive applications.

Some examples of latency-sensitive applications are:

1. Capital Market
   Low latency is most important in the capital or stock market where each second affects the decisions and increases the profitability of trades. High latency can adversely affect the complete market and make the trading process slow leading to huge losses.
2. Vehicles
   Vehicles are largely dependent on edge computing and low latency. A feature like an airbag must be activated without delay. Time-sensitive features in vehicles must have low latency.

Reducing Latency

1. Use of CDN(Content Delivery Network):
   CDN helps to reduce latency. CDN servers are located at different points in order to reduce the distance between users and data doesn't need to travel long distances thereby saving time.
2. Upgrading computer hardware/software:
   Upgrading or tuning the computer hardware, software or mechanical systems can help reduce the computational delays which help in reducing the latency.
3. Caching
   In computers, a cache is a high-speed data storage layer that caches a chunk of data that is typically transient so that subsequent requests for that data can be

delivered up faster than if the data were accessed directly from its primary storage location. This also reduces latency.

Throughput

You have to organise a birthday party in a cafe. Suppose cafe A has 2 sittings and serves food in 15 minutes and cafe B has 5 sittings and serves food in 20 minutes. Which one will you prefer? Confused?
Let's calculate the rate of providing food for each cafe.
For cafe A, rate of serving food = number of sittings/serving time = 2/15 = 0.134 person/minute
For cafe B, rate of serving food = number of sittings/serving time = 5/20 = 0.25 person/minute
It is very clear now that cafe B has a higher rate of serving which means it serves more people per minute than cafe A. So, one should choose cafe B.
This rate helped us determine the better cafe's throughput.

You must have understood how important throughput is. Let's understand it better.

What is throughput?

Throughput is the amount of data transmitted per unit of time. It is the process flow rate.
Throughput is measured in bits per second i.e. bps.

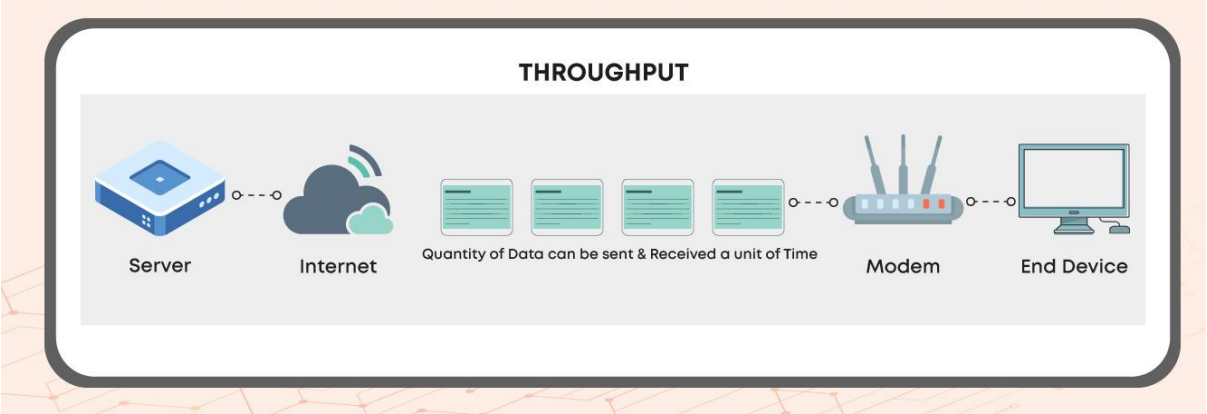


Fig: Throughput

Throughput Formula

Throughput = Inventory / Flow Time

Where,

Inventory: Number of units contained in a process
Flow Time: Total time a unit spends in the process
Throughput: Flow Rate. The number of units passing per unit time.

Example:
Suppose a car manufacturing company makes 100 cars. One car is manufactured in 10 days. Find the throughput.

Inventory (total number of cars): 100
Flow Time (Manufacturing time of one car): 10 days
Throughput : 100/10 = 10 cars/day
Therefore,

Flow Rate or Throughput = 10 cars/day

Causes of Low Throughput

1. Congestion:
   Similarly to the traffic jam, when multiple cars try to cross the road but at the end due to crossing paths, none is able to and the process is stuck. A higher number of process requests at the same time can cause congestion in the process.
2. Protocol overhead:
   If there is a requirement like handshakes or to-fro communication this leads to overloading of the protocol overheads instead of the content.
3. Latency:
   Higher latency (slow transmission of data) will also reduce the amount of data transmitted.

Throughput for different architectures

Monolithic Architecture:

- Limited resources and threads because of a single codebase handling all layers (single machine is limited in means of hardware and machine resources).
- The number of machines cannot be increased.

Limited Resources => Limited Throughput

Distributed System:

- Multiple machines are connected over a network.
- No limit on the availability of machines or resources available.
- Load balances help to distribute the load better. It avoids the situation where some nodes are completely ideal and some are highly overloaded.

Ample Resources => Higher Throughput

Improving Throughput

1. Improving the hardware/software or machine resources:
   Updating and improving the software/hardware or machine resources increase the throughput every time by improving the performance.
2. Improving the performance by using CDN(Content Delivery Network):
   CDN decreases the distance between the user and the server. This helps in shortening the response time since the distance between user and server is decreased. This improves the network performance.
3. Improving the performance by caching:
   Database Caching helps to reduce the retrieving latency which increases the throughput.
4. Monitoring and fixing all the performance bottlenecks
   Performance bottlenecks refer to network overloading which is a cause of low throughput as mentioned above. Monitoring and fixing these issues helps to improve the network performance.
5. Distributed computation using load balancers
   Load Balancing improves both the response time (latency) and resource utilisation. It avoids the situation where some nodes are completely ideal and some are highly overloaded.

Availability

You want to book an urgent train ticket to your hometown. You login into the IRCTC website to book the ticket. The server is already overloaded because of chhath puja next week. You are trying to access the site for the last two hours, but the site is down or unavailable.

How will you feel? Is this a good User Experience? Is this reliable?

This is where availability comes into play. It is an essential system aspect and performance measure of the application. Let's try to understand it better.

What is availability?

Availability is the probability of whether a system will work as needed when the user wants to make a request. If an application responds every time the user makes a request, then the application is an available application.

Example: Google has a very supportive system and is 100% available/very high availability.

Availability and Architecture

Monolithic Architecture:

- A single centralised system with all layers implemented in one codebase, Monolithic Architecture can be deployed on different machines but data reliability decreases.
- In case of a fault in one unit of the system, the complete machine goes down.
- Monolithic architecture is prone to a Single Point Of Failure(SPOF).
- During such cases, the machine would be unavailable.

Monolithic Architecture => Low availability (due to SPOF)

Distributed System:

- Multiple machines are connected over a network.
- Has a mechanism to avoid Single Point Of Failure(SPOF).
- Due to redundancy and replication, the availability increases.

Distributed System => High availability (due to Redundancy/Replication)

Therefore,

Availability of Distributed Systems > Availability of Monolithic Architecture.

Fault Tolerance/Partition Tolerance

Failures are frequent in distributed systems. During a failure, the system should not go down, and the system must handle the fault gracefully.

Distributed system handle failures using:

1. Redundancy
2. Replication

If the machine is fault-tolerant, the availability would be higher. Because in case of failure, there is a duplicate machine to operate in place.

Therefore, Fault tolerance is directly proportional to availability.

Fault Tolerance $\propto$ Availability

How to increase the availability?

We can increase the availability of an application using the following ways:

1. Eliminate single points of failure:
   Use of replication or redundancy for avoiding Single Points of Failure.
   Example: We can replicate very hardware devices like two routers, two switches, two servers, two power sources instead of one.
2. Ensure Automatic Failover:
   Automatic Failover is automatically moving a machine as standby during a failure to preserve its uptime. It is heavily used by Amazon Web Services, Google Cloud, Microsoft Azure etc.
3. Implement Geographic Redundancy:
   Hosting the server at different geographical locations preferably close to the user,always to have, increases network performance. Hosting on multiple servers provides high availability along with better performance.
4. Keep Improving and Updating:
   Scripted deployments can be used for automatically updating the server. This allows users to always have access to the latest and most efficient version with more reliable solutions to security vulnerabilities thereby increasing the availability.
5. Provide excellent support:

The Data Storage and Cloud Service Providers should be highly accessible and up 24*7. Extremely responsive support can help increase availability.

Consistency

Suppose, you deposit Rs.5000 in your Bank Account where before depositing your account balance was Rs.0. In the evening, you urgently need money for paying your house rent. You rush back to the ATM machine nearest to your home and try to withdraw money but the account balance is still 0. You connect with the technical staff and know that the database has not been updated yet and money can only be withdrawn once the server is updated. This is an example of poor consistency. Poor consistency causes irregular data flow through the system and is a cause of major system issues along with poor user experience.

Let’s deep dive into consistency.

What is consistency?

Consistency is uniformity in data. When a user requests data and the system reverts back with the same data irrespective of the geographical location, time etc.

For a consistent system, the server at which data is updated or changed should effectively replicate the new data to all the nodes before the user reads the data from any node.

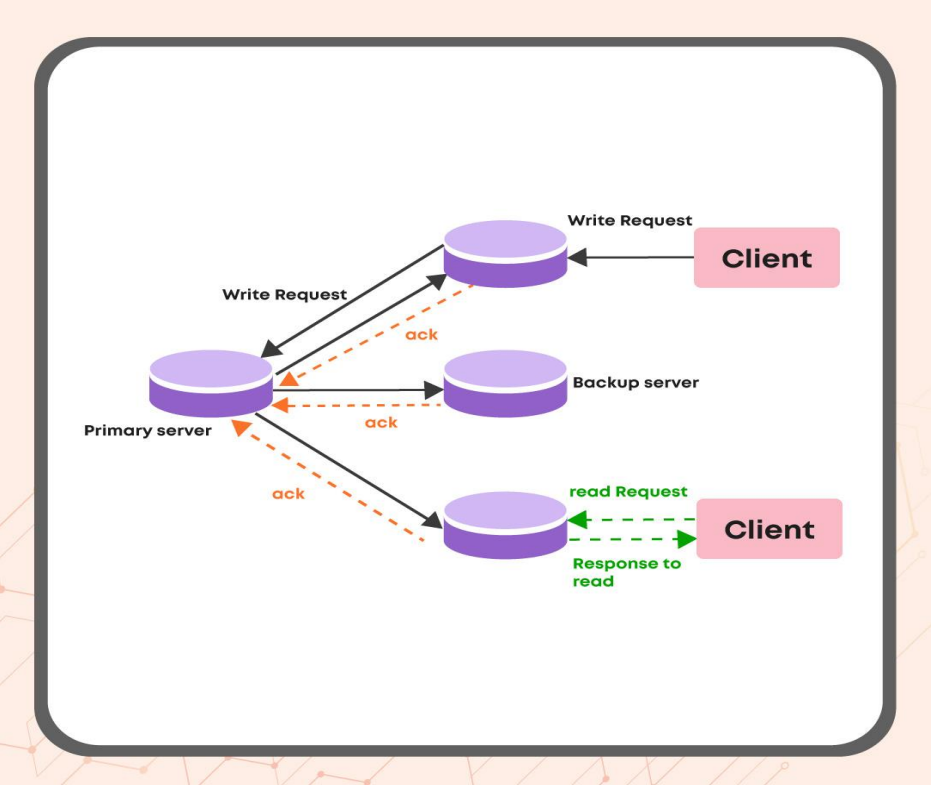


Fig: Consistency Model

Example: A system has three nodes A,B and C. The user has changed data at node A and the replication of new data from A to B takes t1 time and A to C takes t2 time. The system should not accept any read operations till t3 time which is the maximum of t1 and t2. So that no user gets inconsistent data or the old data.

Dirty read: Dirty Read is a phenomenon that occurs when a transaction reads data that has not yet been updated and returns the old data to the user.
For example: Suppose transaction A updates a row in node x. Before the same row is updated in node y, the user reads the information. This is when the user receives old data. This is a situation of dirty reading.

Consistency and Architecture

Monolithic Architecture:

- Centralised unit with all the code and layers implemented in a single codebase.
- Single node/server with no concept of replication.
- The data read is the data written last.
- Because the data is not replicated, the monolithic systems are naturally consistent.

Monolithic Architecture => No Replication => Natively Consistent

Distributed System:

- Use Data Replication for improving data accessibility and availability.
- Therefore, data consistency is not naturally present.
- It is the process of different nodes having the same value.
- In a distributed system, different nodes reach a single consistent state.
- The consistency depends on how fast the replicas of the data are updated.
- The faster the update, the better the consistency.
- Nodes must be efficiently synchronized and the read process must be stopped during the update process.

Distributed System => Replication => Needs efficient update

Factors Improving Consistency

1. Halt the read:
   For ensuring a consistent system, read must be stopped until data at all the nodes are updated.
2. Improving Network Bandwidth:
   Consistency depends on the speed of data update of replicas. By improving the network bandwidth, the speed can be increased. This will help synchronize the replicas.
3. Replication based on Distance aware strategies:
   If most of the replicas are stored in the same data center, the speed of update should increase and the process will be more optimised.

Types of Consistency

There are different types of consistency models.

1. Strong Consistency:
   It is the strongest model of memory coherence. Strict consistency is when the system doesn't allow read operation until all the nodes with replicate data are updated. For n number of nodes in the distributed system, the time of updating the replicas be t1,t2...tn. The maximum of t1,t2...tn is the time after which all the replicas are updated and overwritten with the latest data. The read process can be continued only after this.
   When a user requests data and the system reverts back with the latest data irrespective of the geographical location, time etc.

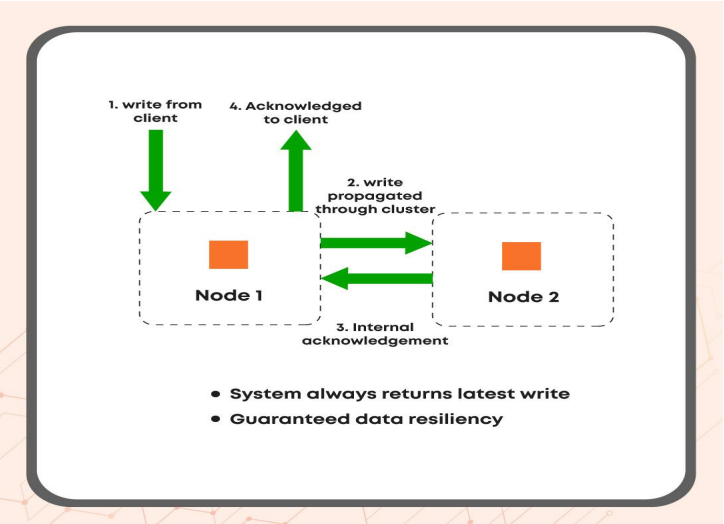


Fig: Strong Consistency

2. Eventual Consistency:

User Read requests are not halted till all the replicas are updated rather the update process is eventual. Some users might receive old data but eventually all the data is updated to the latest data. It weekends the aspect of consistency because it is an eventual process.

If the node where data was written breaks down due to any reason, the data would be lost. It also doesn't guarantee that the data responses back would be the latest replica.

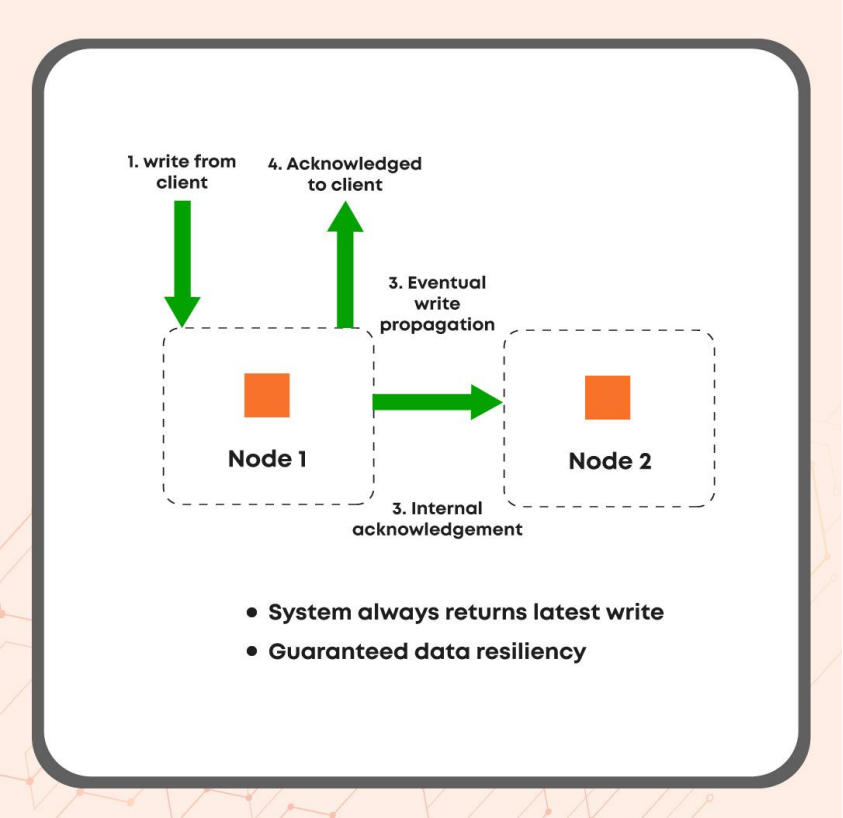


Fig: Eventual consistency

3. Weak Consistency:

It is the worst model of consistency which essentially doesn't ensure consistency. During a variable or data change by the user, it is not mandatory to inform or update the changed information in all the replicas. Each node would carry different states of data and therefore should be avoided.

There is no guarantee that all the nodes in a distributed system would carry the same information at a point in time.

CAP Theorem

What is CAP Theorem?

CAP Theorem is used to explain the Non-Functional requirements and their relationship for a distributed system. The CAP in CAP Theorem stands for:

C: Consistency
A: Availability
P: Partition Tolerance

Let's revise what these terms mean.

Consistency: Consistency is uniformity in data. When a user requests data and the system reverts back with the same data irrespective of the geographical location, time etc. For a consistent system, the server at which data is updated or changed should effectively replicate the new data to all the nodes before the user reads the data from any node.

Availability: Availability is the probability of whether a system will work as needed when the user wants to make a request. If an application responds every time the user makes a request, then the application is an available application.
Example: Google has a very supportive system and is 100% available/very high availability.

Partition Tolerance: Failures are frequent in distributed systems. During a failure, the system should not go down, and the system must handle the fault gracefully. This attribute of a system is known as partition tolerance.

The CAP Theorem states that it is possible to attain only two properties and the third would be always compromised. The system requirements should define which two properties should be chosen over the rest.
Example:

- The system designer can select Consistency and Partition Tolerance but the availability would be compromised then.
- The system designer can select Partition Tolerance and Availability but the consistency would be compromised then.

- The system designer can select Availability and Consistency but the Partition Tolerance would be compromised then.

Out of all these three desirable properties, Partition Tolerance should always be given priority else during a network breakdown, all the data would be lost. The designer can select or prioritize between availability and consistency.

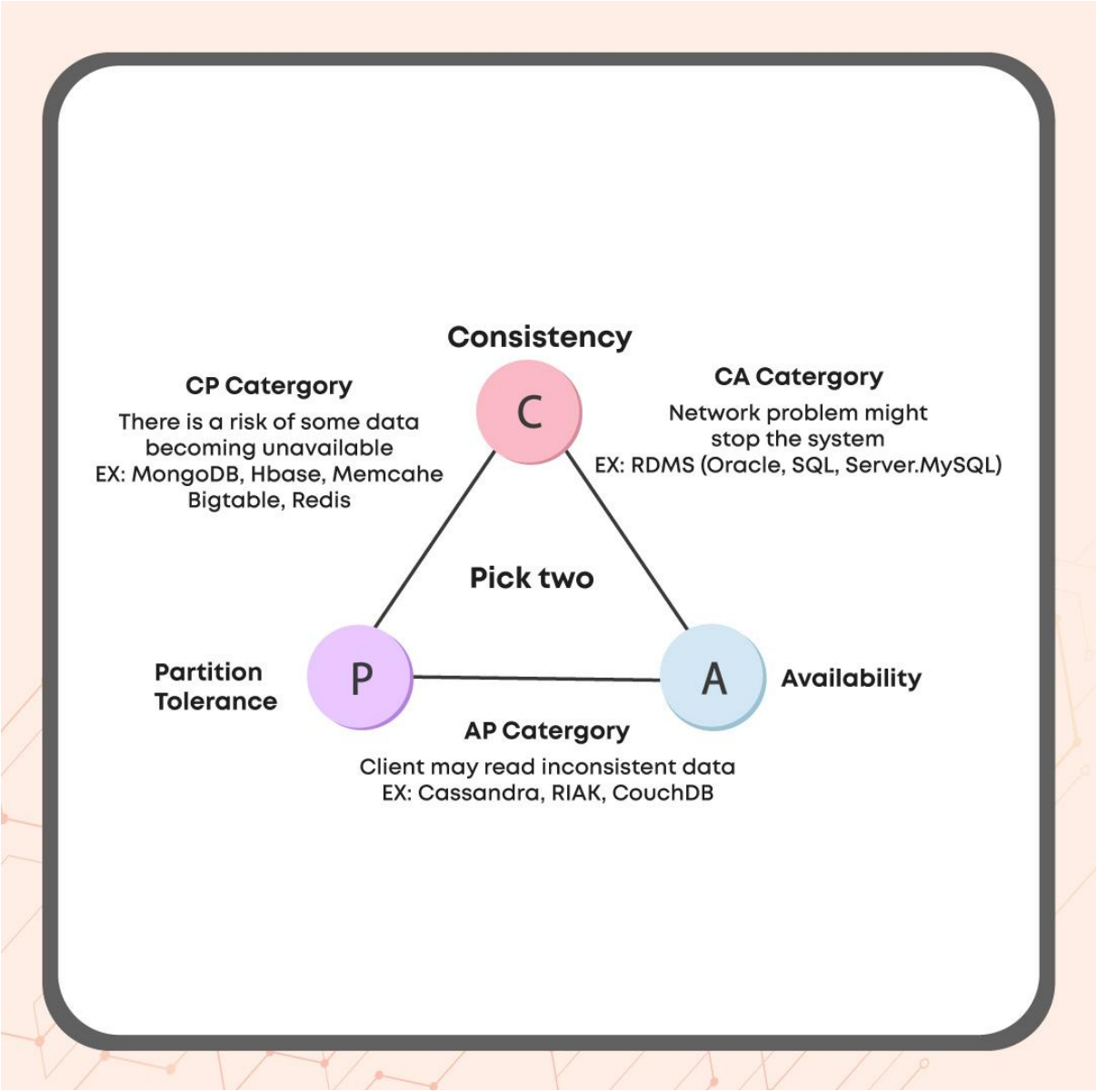


Fig: CAP Theorem

Redundancy and Replication

Why Redundancy and Replication?

Redundancy and replication are essential for maintaining availability and data integrity. Let's see how.

Faults and data loss is prevalent in distributed systems because of:

1. Commodity Hardware
2. Network Fault

Due to the above two factors, chances of failure of distributed systems are very high because the commodity hardware and network are highly unreliable.

Therefore, distributed systems are fault-prone systems.

We need to handle the faults and failures gracefully, known as Partition Tolerance/Failure Tolerance.

Partition/Failure Tolerance can be achieved by:

1. Replication
2. Redundancy

Now, let’s understand each term.

What is Redundancy?

Redundancy is simply the duplication of nodes or components so that when a node or component fails, the duplicate node is available to service customers. This is how redundancy helps in maintaining availability, failure recovery or failure management. The idea behind redundancy is to make backup paths speed, efficient and available.

Example: Global Positioning System (GPS)

Types of Redundancy

There are two types of redundancy:

1. Active Redundancy:

   Active Redundancy is considered when each unit is operating/active and responding to the action. Multiple nodes are connected to a load balancer, and each unit receives an equal load.

   Example: A building requires five generators to power the complete building. Now, for failure tolerance, we can use additional generators according to the principle of redundancy. We can have n+1 (6 generators) for supporting single

node failure as even if one generator of all the working six generators fail, the power won't cut. This is how active redundancy works; the system keeps operating even in the case of failure of any node.

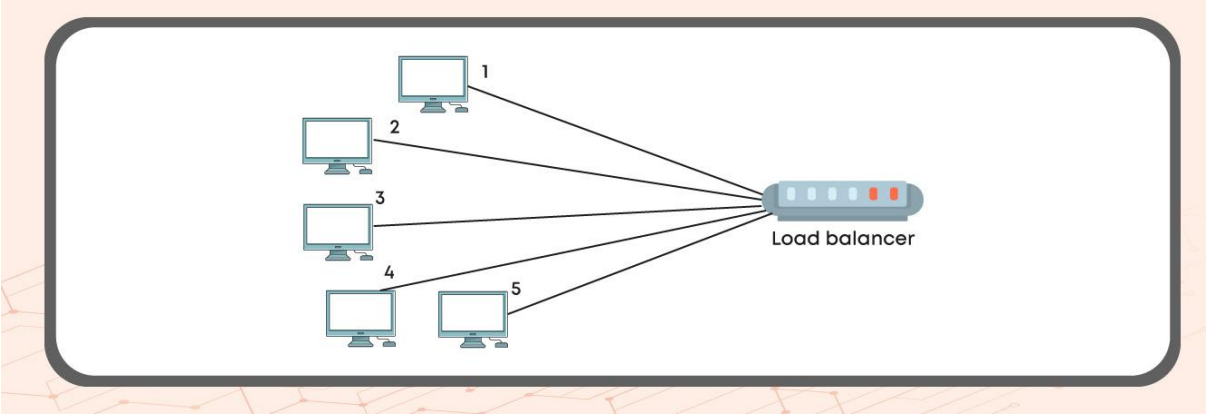


Fig: Active Redundancy

2. Passive Redundancy:
   Passive Redundancy is considered when one node is active or operational and the other is not operating. During the breakdown of the active node, the passive node maintains availability by becoming the active node.
   Example: A car has four tires and one spare tire. Four of the tires are operational and keep the vehicle moving. In the case of one flat tire, the spare tire can be used, which was earlier acting as a passive component.

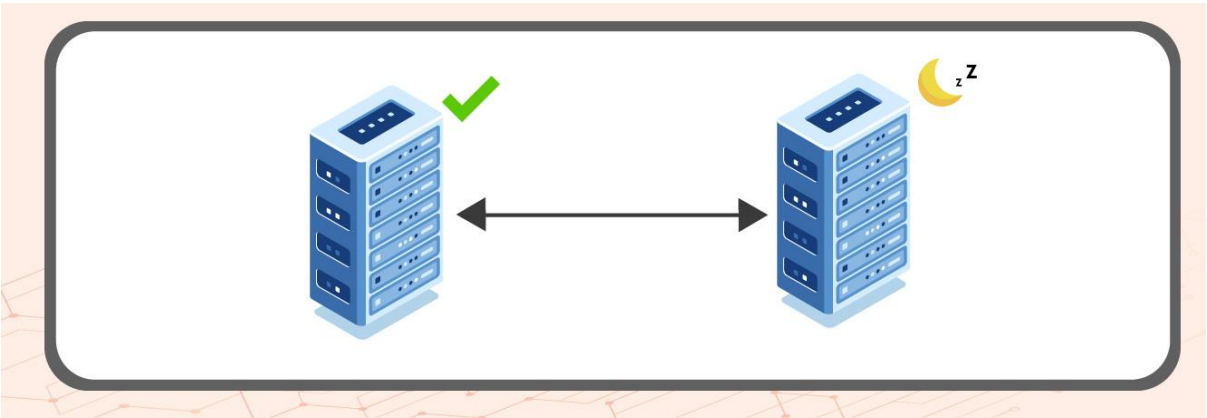


Fig: Passive Redundancy

What is Replication?

Replication is the management process of different data storage where each element is stored in multiple copies hosted on other servers. Simply, it is the copying of data on various machines.

It is the synchronisation of different machines.

Replication helps to ensure consistency between redundant resources to improve fault tolerance and reliability.

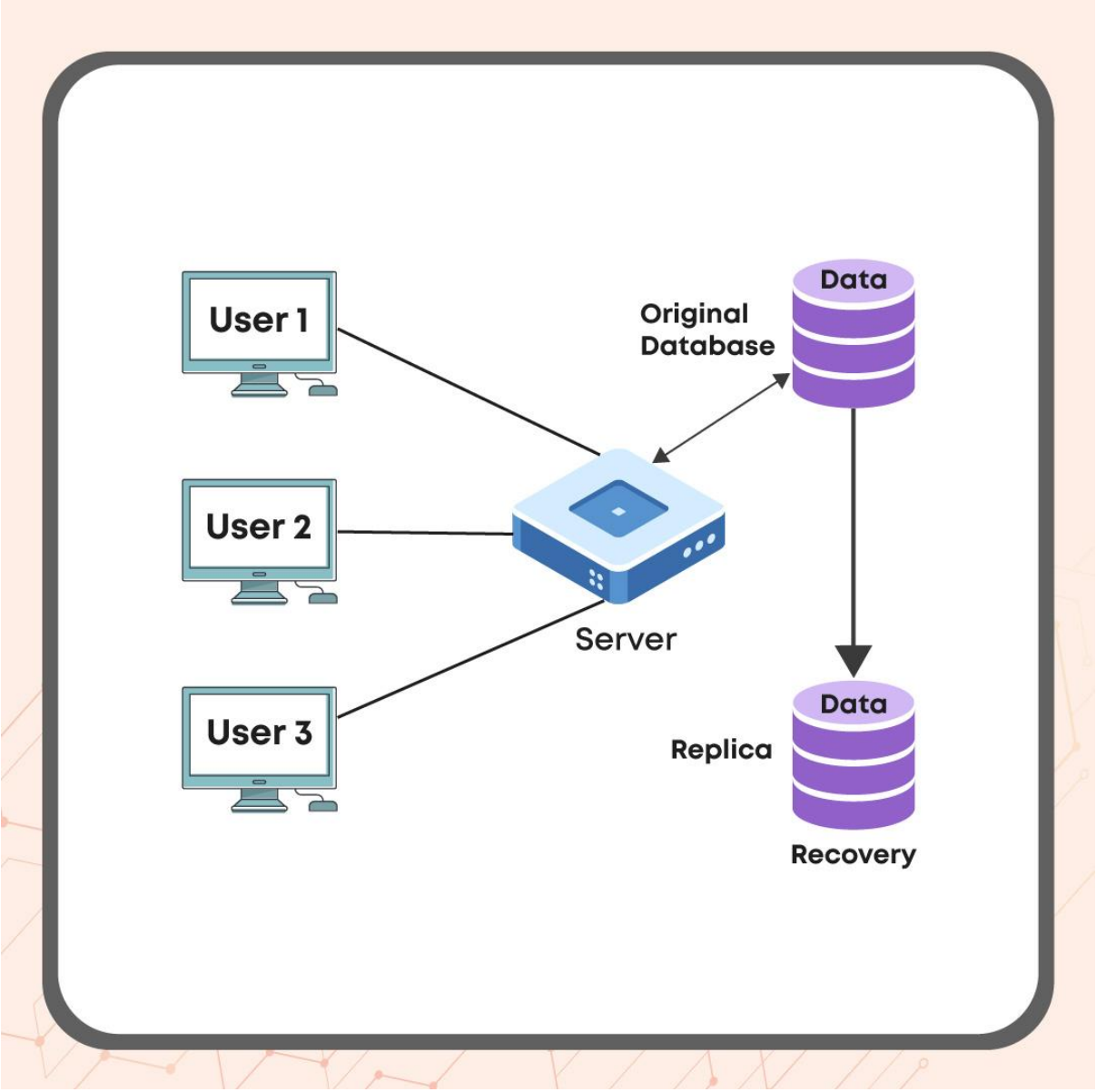


Fig: Replication

Types of Replication

There are two types of Replication.

1. Active Replication:

Active replication is when all the nodes of a cluster are connected, and the data is replicated on every cluster node. It is performed by processing the same request on all the replica machines.

Active replication is used mostly when a single master cannot tolerate all writes, so we need all nodes to accept writes. Then they sync data among themselves. Reads should preferably be below in such cases but could be the same as writes. An example of such a system is "how many times x item was viewed on amazon"

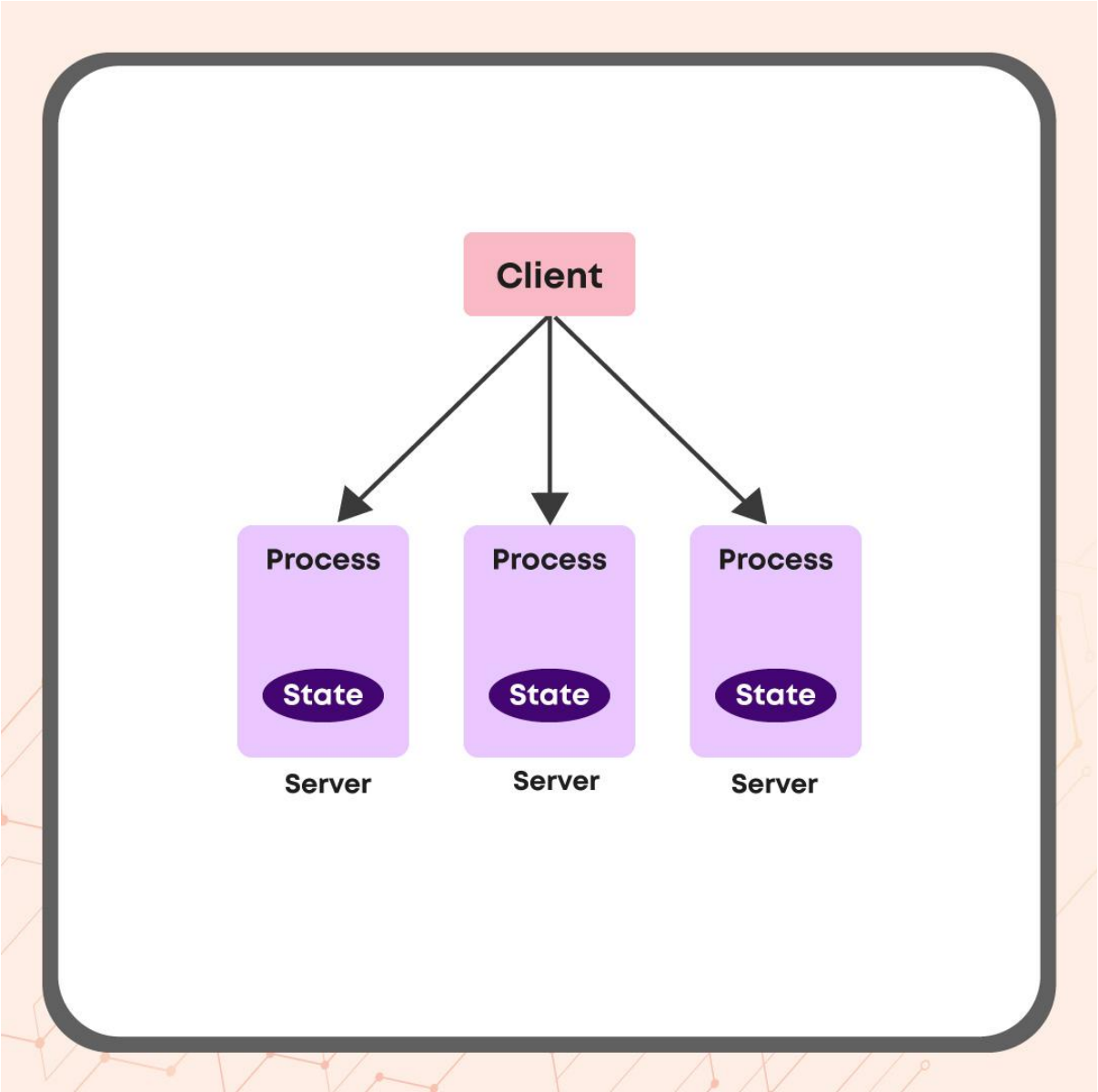


Fig: Active Replication

2. Passive Replication:

Passive replication is when a cluster of nodes has one machine as the master machine and other machines as the slave machine. The read and write operations are completed on the master machine, but the data is replicated on all the slave machines in offline sessions. If the master machine goes down, any of the slave machines can be promoted as the master machine.

Passive replication is used in systems that have low write rates by high read rates. So a single master is enough to accept all writes and can replicate it to all the slaves (sync/async).

Examples of such systems could be Wikipedia.

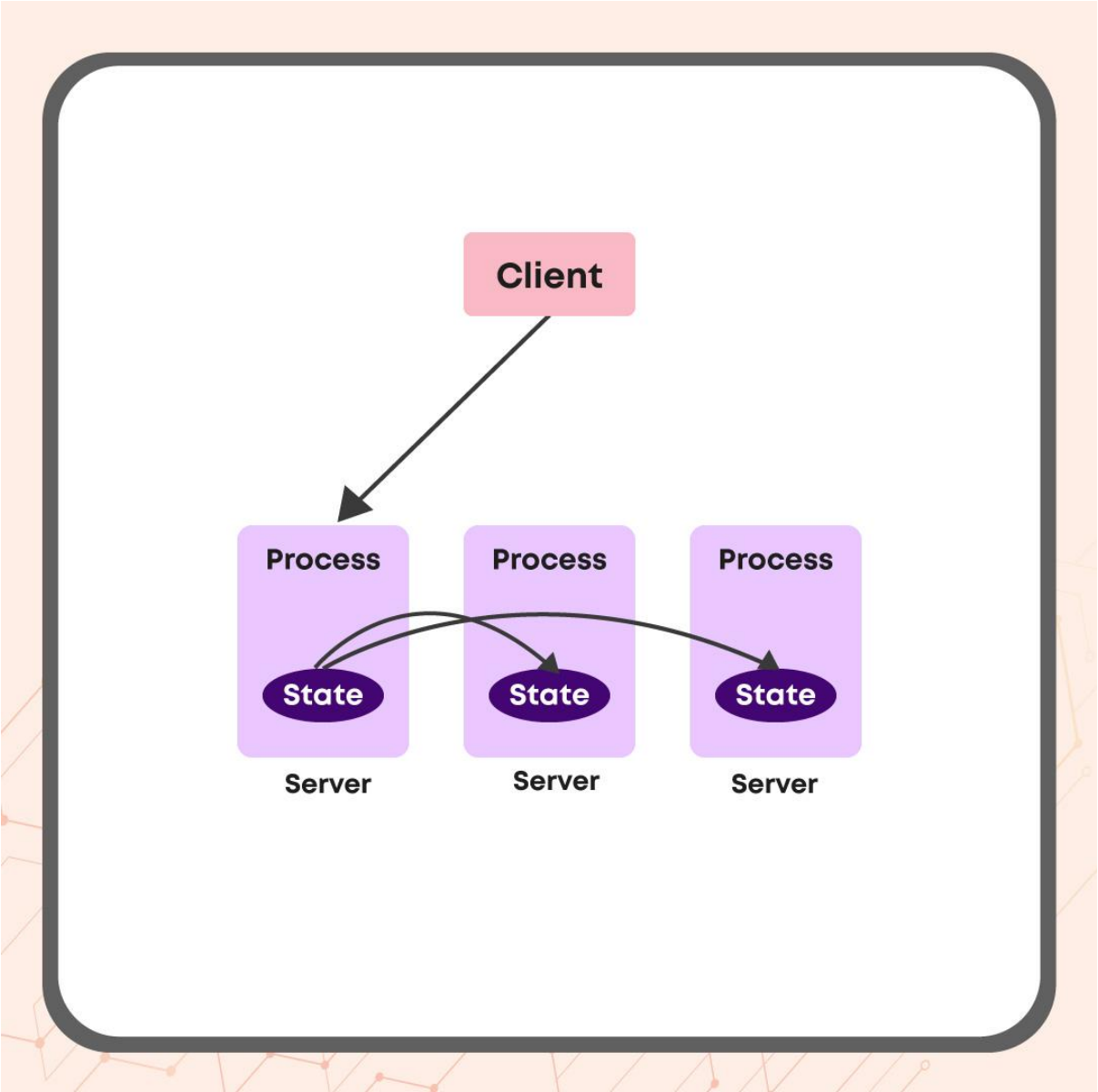


Fig: Passive Replication

High-Level Design(HLD) and Low-Level Design(LLD)

System design architecture can be depicted as High-Level Design(HLD) or Low-Level Design(LLD). Let's look at what they signify.

1. **High-Level Design(HLD):**
   This kind of architecture takes into consideration the main components that would be developed for the resulting product. The designer only focuses on a high-level overview, including
   - principal components,
   - database
   - services
   - relationships between each module/component

   with a brief description of the system.

2. **Low-Level Design(LLD):**
   On the contrary, this kind of architecture considers the in-depth and detailed design of each component mentioned in the High-Level Design of the system. It exposes the logical relationship between different elements and a detailed description of all the modules. Low-Level Design includes more technical information as compared to High-Level Design. Low-Level Design includes the description of the following components:
   - IP Address
   - Class Diagrams, Sequence Diagram, Activity Diagrams
   - VLAN and Port Numbering
   - Information about all the platforms, services, and processes of the product would depend on
   - Algorithm and pseudocode
   - Different Classes, interfaces and the relationship between them
   - Relationships between the modules and system features
   - External Interface Requirements, Hardware and Software Interfaces
   - Design and Implementation Constraints

Example 1:
Suppose Ram wants to create a social media app. The high-level design will consist of an overview of the system like a mobile app for ios/Android, an app for OS/X & Windows, Web GUI, SQL/No-SQL database. In contrast, the low-level design would consist of how each user application is logically linked to other modules and all the extra details of each component mentioned in the HLD that’s useful and necessary before developers can start writing the code directly.

Example 2:

Let’s try to design a Cab Booking System like Ola, Uber etc. An application like this has extensive design and functionalities involved. Well, let's try designing one.

First, we must understand the Functional and Non-Functional Requirements of such applications.

Functional Requirements:

- Checking nearest cab availability
- Booking a cab
- Fare calculation and payment
- GPS tracking for the cab

Non-Functional Requirements:

- Security (secure transactions)
- Reliability
- Scalability
- Less Response Time

High Level Design:

The High Level Design would consist of the major modules for Desktop and Mobile Client like:

- Manage Car Details
- Manage Customer Details
- Manage Booking Details
- Manage Price Details
- Manage Payment Details
- Manage Car Insurance Details
- Map Service

The admin application would mainly consist of a cab request service and cab finder service. Also, remember that the live location of a driver is always shared with the system, this can be further managed by a tracking module i.e. the location service.
The trip completion event would be managed by the payment service which would calculate the price on the basis of distance covered and the rate per unit distance.
High-Level Design would include a brief description of the basic functionalities like user login to the system, checking credentials, Check car availability in vicinity etc and the databases like profile database, authentication database.

Low-Level Design:

The Low Level Design would be the more detailed description of the High-Level Design. It would carry the details like:

- Payment Gateways linked to the payment service.

- Database along with the driver service, Payment Service, User Service, Trip Archives etc
- Detailed map service for location tracking.
- Private classes, private methods, private attributes
- Data structures and algorithms to be used

High-Level Design Vs Low-Level Design

The below table summarises the difference between HLD and LLD.

Parameter	High-Level Design (Macro-Level/System Design)	Low-Level Design (Micro-Level/ Detailed Design)
Abbreviations	HLD	LLD
Input	SRS (Software Requirement Specification)	Reviewed HLD (High- Level Design).
Definition	Describes the main components that would be developed for the resulting product.	Describes the design of each element mentioned in the High-Level Design of the system.
Content	The system architecture details, database design, services and processes, the relationship between various modules, and features.	Classes, interfaces, relationships between different classes, and actual logic of the various components.
Chronological order in the design phase	Created first	Created after High-Level Design is completed.
Technicality involved	Less technical	More Technical
Target Audience	Management and program team	Used by implementers and the coding team.

Database

File-Based Storage System/File Based Database Management System:

A file-based storage system is a database management system where data is stored in the form of files. This system allows access to a single file at a time. It is a collection of flat files(single tables stored in one file) that do not carry any correlation with other files in the system.

Example: Microsoft NTFS(New Technology File System), Hierarchical File System of Apple.

Challenges of File-Based Storage System

This storage system has multiple disadvantages like:

1. Data Redundancy:
   In a traditional file management system, redundancy occurs because the same data is stored at different places. If the data is updated or replaced at someplace and not updated at other places, the same data would have multiple copies with differences and it would be difficult to find out the latest data. This causes a threat to data security and integrity. This can cause multiple data anomalies like delete anomaly, insert anomaly and read anomaly. This also causes data inconsistency.
2. Poor data security:
   Because of data redundancy, multiple copies of the same data are present in the system which causes a possibility of data breaches by unauthorised users. Therefore, it is a serious threat to data security.
3. Slow:
   The traditional file management system is not very quick because it needs a lot of ad-hoc queries and more extensive programming comparatively for report generation. Therefore, only programmers can deal with this storage system easily.
4. Not so efficient data retrieval:
   While incorporating this file-based management system in today's applications, there would be multiple performance issues and the speed is very slow. Due to these challenges, data retrieval is not very efficient.

To solve these problems, a new database management system was introduced known as RDBMS(Relational Database Management System) to solve these problems.

Relational Database Management System(RDBMS):

It is one of the most popular database management systems. RDBMS is a software that performs all data-related operations on a relational database like a store, manage, query, and retrieve data. It is based on the relational model. The significant components of RDBMS are tables. Data is represented in the form of tables. The relationship between the two tables is represented by foreign keys.

Example: MYSQL, Oracle etc.

Advantages of Relational Database Management System over File System

1. Data redundancy and inconsistency

   Redundancy refers to the concept of data repetition, which means that any data item may have multiple copies. Because each user sets and maintains the files required for a specific application to execute, the file system is unable to control data redundancy. It's possible that two users are sharing the same files and data for different programmes. As a result, changes performed by one user do not appear in files utilised by other users, resulting in data inconsistency. RDBMS, on the other hand, manages redundancy by keeping a single data repository that is defined once and accessed by multiple users. Data remains constant because there is no or little redundancy.

2. Data sharing

   The file system does not allow data sharing or it is too complicated. Due to the centralised structure in RDBMS, data may be simply exchanged.

3. Data concurrency

   When more than one user accesses the same data at the same time, this is referred to as data concurrency. Anomalies occur when one user's edits are overwritten by changes made by another user. There is no method in the file system to prevent abnormalities. A locking system is provided by RDBMS to prevent abnormalities from occurring.

4. Data searching

Each file system search activity necessitates the creation of a separate application programme. RDBMS, on the other hand, has built-in searching capabilities. To access data from the database, the user merely needs to submit a short query.

5. Data integrity
   Before putting data into a database, some constraints may need to be applied to the data. There is no process in the file system to check these constraints automatically. RDBMS, on the other hand, ensures data integrity by enforcing user-defined restrictions on data.

Challenges of Relational Database Management System

This storage system had multiple disadvantages like:

1. Rigid Schema:
   In case the data schema is dynamic and changes, RDBMS would not be compatible.
2. Limited Scaling Patterns:
   The limitation here is that scaling patterns are very limited, unlike NoSQL DBs.
3. Cost:
   RDBMS is comparatively expensive software.
4. Skills:
   The data administrator must be skilled in order to use this storage system.

We'll go through the two basic types of databases:

1. Relational Databases (SQL based).
2. NoSQL databases

SQL Database

SQL is a computer language that was created for managing data stored in a relational database management system. The data in relational DBMSs appear as tables of rows and columns with a well-defined structure and dependencies.

SQL databases require little engineering effort to secure because of their integrated structure and data storage method. They're a fantastic fit for creating and maintaining complicated software solutions where every interaction has a variety of outcomes. ACID compliance is one of the SQL foundations (Atomicity, Consistency,

Isolation, Durability). If we are creating eCommerce or financial apps, for example, where database integrity is crucial, ACID compliance is the way to go.

Some examples of SQL databases

- MySQL
- Oracle
- PostgreSQL

Some Applications of SQL are

- Developers and DBAs (Database Administrators) use SQL to create Data Integration Scripts.
- It is used to handle analytical queries in order to evaluate data and derive insights from it.
- Information Retreival
- Insertion, Deletion, and Updation are examples of data and database table manipulation.

Advantages for using SQL Database:

1. It has a straightforward structure that corresponds to the majority of data types seen in most programmes.
2. It makes use of SQL, which is widely used and enables JOIN operations by default.
3. Allows for quick data updates. Because the entire database is saved on a single machine and relationships between entries are used as pointers, we can update a record once and have all of its associated records updates at the same time.
4. Atomic transactions are also supported by relational databases.

NoSQL Database

It stands for “non-SQL” database, or we can say that it is a non-relational database. It is complementary to RDBMS. It is built on the principle of a distributed system. Therefore, they are natively scalable. NoSQL is the umbrella term comprising of four different types of databases.

Example: MongoDB, Redis etc.

Advantages of NoSQL Database:

1. Semi-structured data

2. Dynamic or flexible schema
3. Non-relational data
4. No need for complex joins
5. Store many TB (or PB) of data
6. Very data-intensive workload
7. Very high throughput for IOPS

Example 1:

You wish to develop an e-commerce website that would contain multiple databases and tables storing user's login information and staff details including ID, date of joining, phone number, address, email id.

What would be the suitable database choice here out of SQL or NoSQL databases?

Answer 1: The appropriate database here is an SQL database since the data is structured primarily. Transaction-oriented systems, such as customer relationship management tools, accounting software, and e-commerce platforms, benefit greatly from SQL databases.

Example 2:

Let's pretend you're working on the next Google Analytics. You've decided to monitor IP addresses, browsers, and device types. However, you later realise that you might want to keep track of Browser Size as well. It's not straightforward to add an extra column to your table because analytics databases can include millions or billions of records. It would simply be too time-consuming.

What will you choose from SQL and NoSQL databases for this situation?

Answer 2: We should select the NoSQL database here since it has a dynamic and flexible schema that would allow us to add and append rows in the database without changing and re-ordering each row of the tracking table.

The four different types of NoSQL Databases are:

1. Key-value store
2. Document store
3. Column-oriented database
4. Graph database

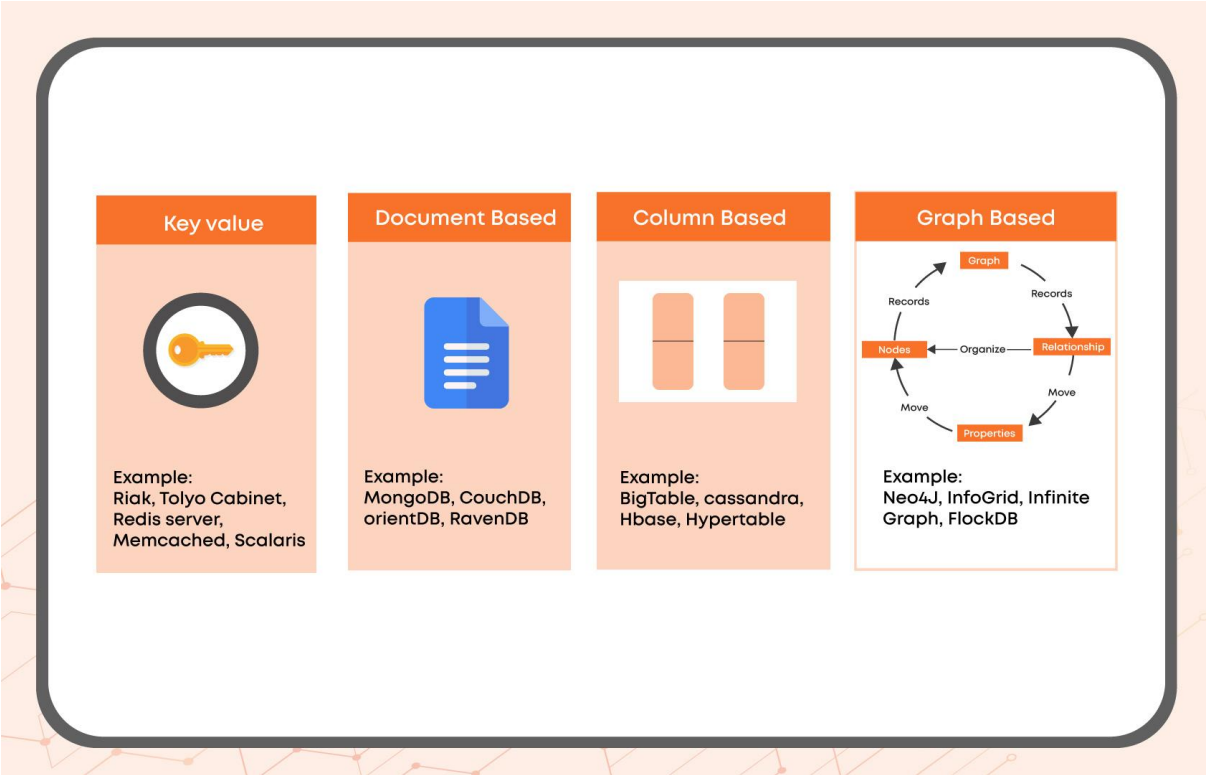


Fig: Different types of NoSQL Databases

1. **Key-Value Database:**

   It stores data in pairs of keys and values. Therefore, it is a non-relational database because the data is not stored in a table but in the form of the key-value method. It is widely used as a caching solution. Suppose we need to store the names of mentors working at Coding Ninjas. The “name” would be the key, and the value would be different names of mentors.

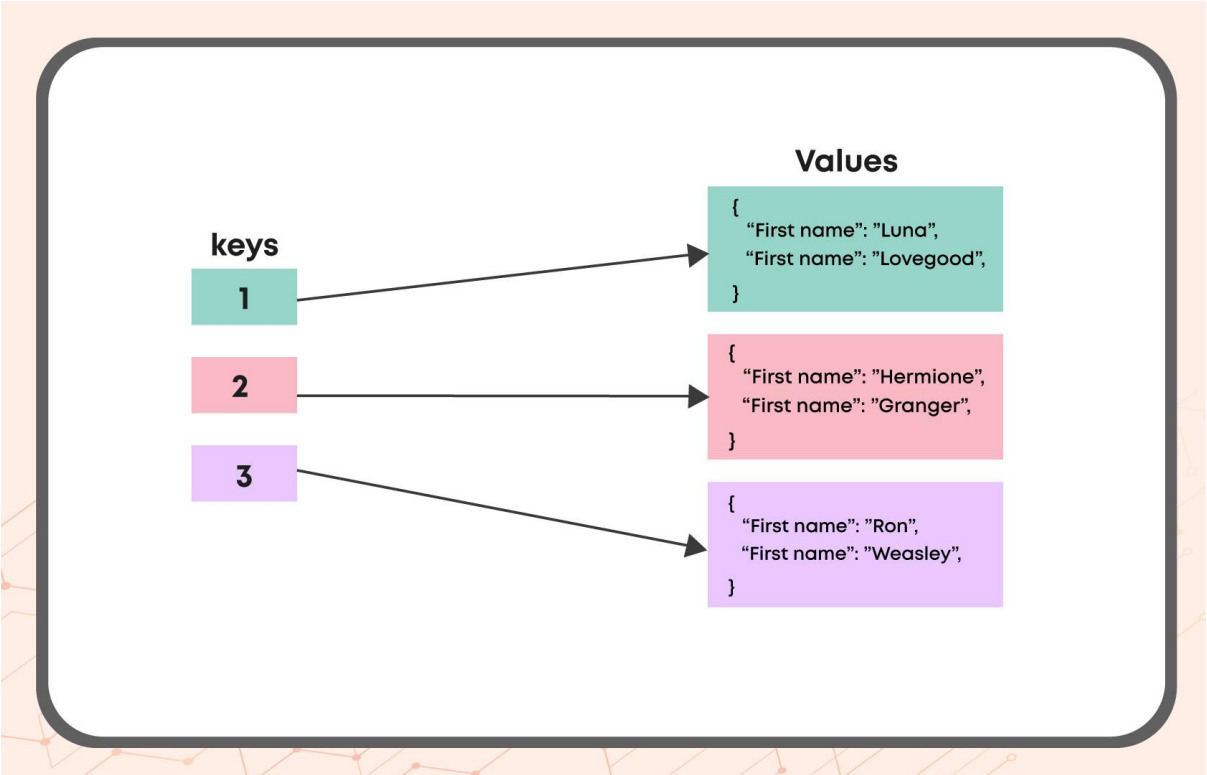
   The TTL column (time to live) is a helpful feature in this database; it may be set differently for each entry and indicates when it should be destroyed from the database.

   Example: Redis

   When should we use Key-Value DB?

   Because it is fast and does not require extensive queries, it is primarily used for caching. The TTL function is also highly beneficial for caching.

   It can also be used for any other type of data with a key-value format that requires quick querying.



2. **Document Database:**

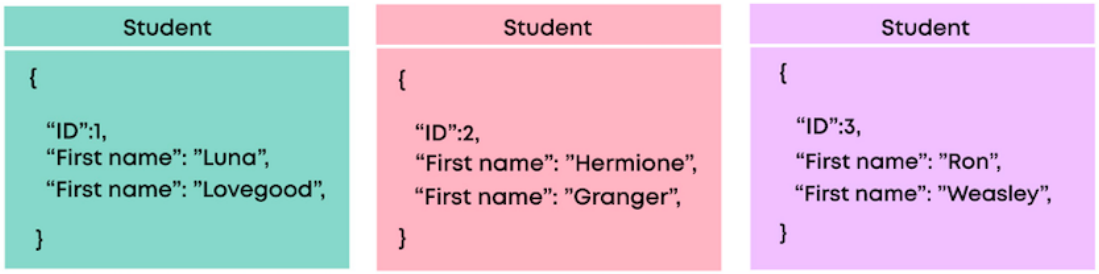
   Data is stored in the form of JSON like documents. It combines the concepts of RDBMS and NoSQL databases. It combines the relationship concept from RDBMS and dynamic schema and horizontal scaling from NoSQL databases.

   Example: MongoDB

   When should we use Document DB?

   Data analysis: This database facilitates parallel computations because separate records are not logically or structurally dependent on one another.

   This makes it simple to run big data analytics on our data.



3. **Columnar Database:**

A columnar database is one in which the columns are stored together instead of rows. Most of the operations are column-oriented in an application, and because of that, the aggregation in such databases is rapid. It is widely used for running analytical queries or Time Series Data.
Example: Cassandra

When should we use Columnar DB?
When we query on a subset of your data's columns.
Because it just needs to read these specific columns, columnar DB conducts such queries quickly (while row-based DB would have to read the entire data).

- Each column represents a feature, which is frequent in data science. Data scientist frequently trains models with subsets of the features and frequently examine feature-score relationships (correlation, variance, significance).
- They often save a lot more attributes in our logs database but only use a few in each query, which is also common with logs.

Row oriented (Relational)

Students		
ID	**First name**	**Last name**
1	Luna	Lovegood
2	Hermione	Granger
3	Ron	Weasley

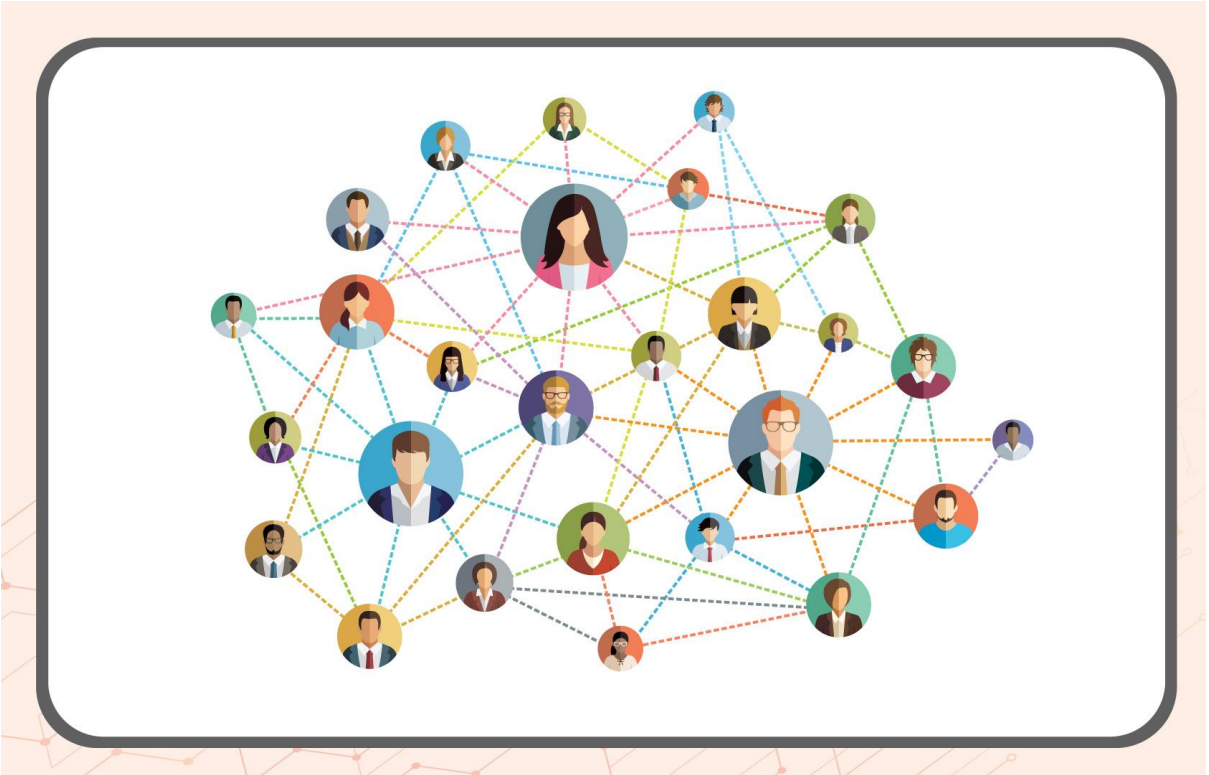
Column oriented

Students		
ID	**First name**	**Last name**
1	Luna	Lovegood
2	Hermione	Granger
3	Ron	Weasley

4. **Graph Database:**

Graph database represents and stores entities and relationships in the form of graph data structure. It is majorly used for social networks and applications incorporating relationships with entities like google maps

.



Example: Neo4j

When should we use Graph DB?

When your data is in the form of a graph, such as knowledge graphs or social networks.

Summary on usage

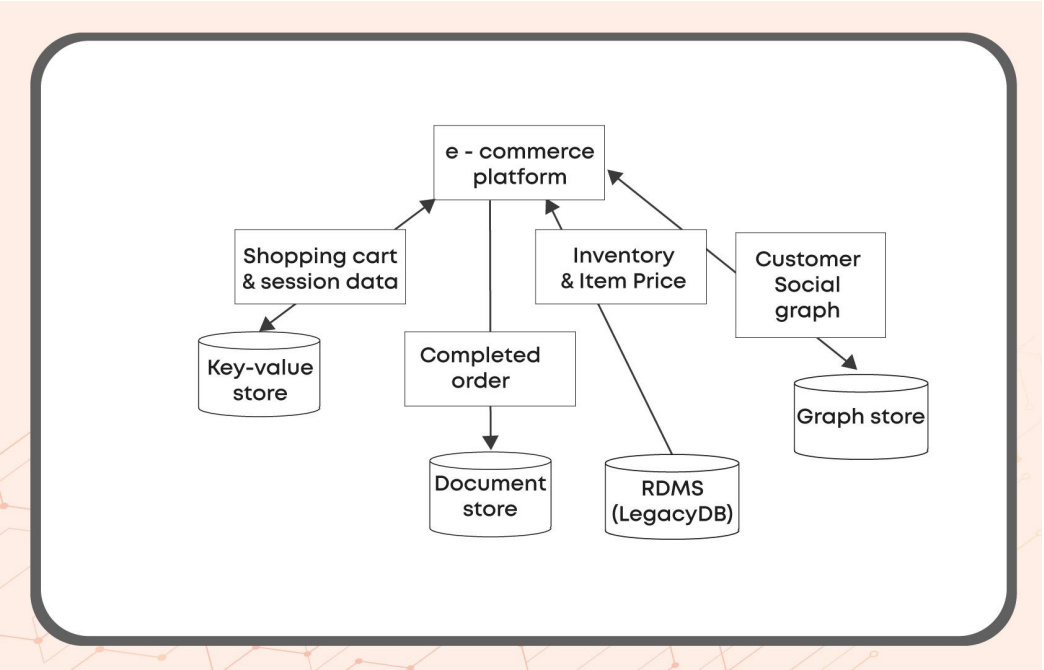
Key-Value Database	Caching
Graph Database	A graph like data for example social networks.
Columnar Database	If querying on columns is required
Document Database	Flexible Schema or we might need to change the schema in future

Polyglot Persistence

Polyglot persistence refers to the use of different data storage technologies across an application or within smaller components of an application to meet varying data storage demands.
Polyglot persistence occurs when one particular application or project uses multiple databases depending on varied data requirements. Different databases can be used for various components of the single application to meet the individual requirements best. Because of varying needs and increasing competition, most of the applications use polyglot persistence these days.

Example: Let's take the example of an e-commerce website like Flipkart. It has multiple functions like inventory management, shopping cart management, payments etc. For better performance, it uses different databases instead of storing all the information in a single database.
It uses different databases like:

1. Oracle for payments: We will need ACID(Atomicity, Consistency, Isolation, Durability) properties here provided by SQL database. It will provide lightning-fast transactions, easy horizontal scaling, and support for JSON documents within records as well as JSON queries.
2. MongoDB for storing data of the product: Flexibility of MongoDB documents can be used here. Each MongoDB document can store data in the form of sophisticated JSON constructs. MongoDB is therefore perfect for storing almost anything, including very big catalogues with thousands of versions per item.
3. Redis(key-value database) for shopping e-cart content: The key-value database can be used for storing cart content. If we wanted to store user session data, shopping cart information, and user preferences, we could just store all of them in the same bucket with a single key and single value for all of these objects.
4. ES(Document DB) for text-based search functions: Structured search is available in NoSQL databases that use document stores, and it preserves the best capabilities of Boolean and full-text keyword search.
5. Cassandra (Document Database) for the data warehouse: A columnar database stores data by columns rather than by rows, which makes it suitable for analytical query processing, and thus for data warehouses.



Normalisation:

Normalisation is the process of organising the data in different tables. Data is stored in multiple tables to avoid redundancy and data anomalies.

We need to perform Normalization on our database to reduce Data Redundancy. It minimizes redundancy using certain rules or sets of rules.

There are many types of normal forms, although we are going to focus on 1Nf, 2NF, 3NF and BCNF (also known as 3.5 NF).

The most commonly used normal forms are:

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)
4. Boyce & Codd normal form (BCNF)

Types of Normal Forms:

1. **First Normal Form (1NF):** This is Step 1 of the Normalisation Process.
   For a Relation/table to justify 1NF it needs to satisfy 4 basic conditions:
   - Each attribute should contain atomic values. (i.e. No multivalued attributes)
   - Each Value stored in an attribute should be of the same type.
   - All the attributes in a table should have unique names.

- The order of the data stored in the table doesn't matter.

2. **Second Normal Form:** For a Relation/table to justify 2NF it needs to satisfy 2 rules:
   - It should be in First Normal Form.
   - It should not have any partial dependencies i.e. when a nonprime attribute is derivable from only a part of a candidate key.

3. **Second Normal Form:** For a Relation/table to justify 2NF it needs to satisfy 2 rules:
   - It should be in First Normal Form.
   - It should not have any partial dependencies i.e. when a nonprime attribute is derivable from only a part of a candidate key.

4. **Boyce-Codd Normal Form (BCNF):** It is an extension of 3NF and is also known as the 3.5 Normal Form.
   For a table to be in the Boyce-Codd Normal Form, it should satisfy 2 rules:
   - It should be in the Third Normal Form.
   - A prime attribute shouldn't be dependent on a non-prime attribute. (i.e. if M→ N, then M is a superkey)

Denormalization:

It is the opposite of Normalisation. Normalisation is breaking the data and organising it in different tables whereas denormalisation is combining the data. Denormalisation combines the data and organises it in a single table. Denormalization is the process of adding redundant data in the normalised relational database to optimise the performance.

Benefits of denormalisation:

1. Faster data read operations
2. Simpler and more accessible to query
3. High data availability
4. Requires less computation
5. Reduced network calls
6. High data accessibility

Challenges of decentralisation:

1. Slow write operations
2. Increases complexity
3. Wastage of memory
4. Data inconsistency

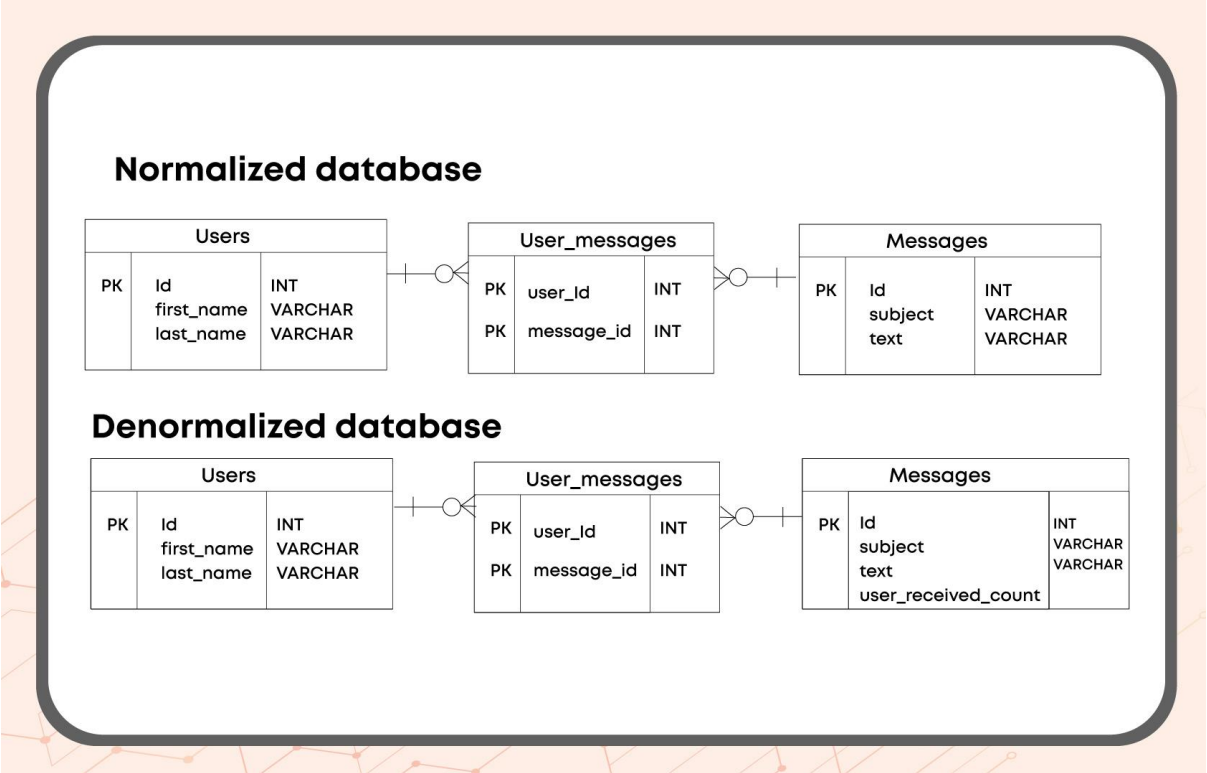


Fig: Normalization vs Denormalization

Indexing

Suppose you visit a pharmacy shop and want to buy Paracetamol. The medicines are arranged in a particular order and therefore the pharmacist is able to search and provide you with the medicine very quickly. This ordering of medicines for optimising search algorithms is known as indexing. Let's try to understand it better.

What is indexing?

Indexing is a process by which queries and other database operations can be optimised by minimising the time required for processing the query. B-trees data structure is used to store the indexes. Indexing should only be used if the data is huge and the application is read-intensive. If an application is write intensive indexing might lower the performance of write operation.

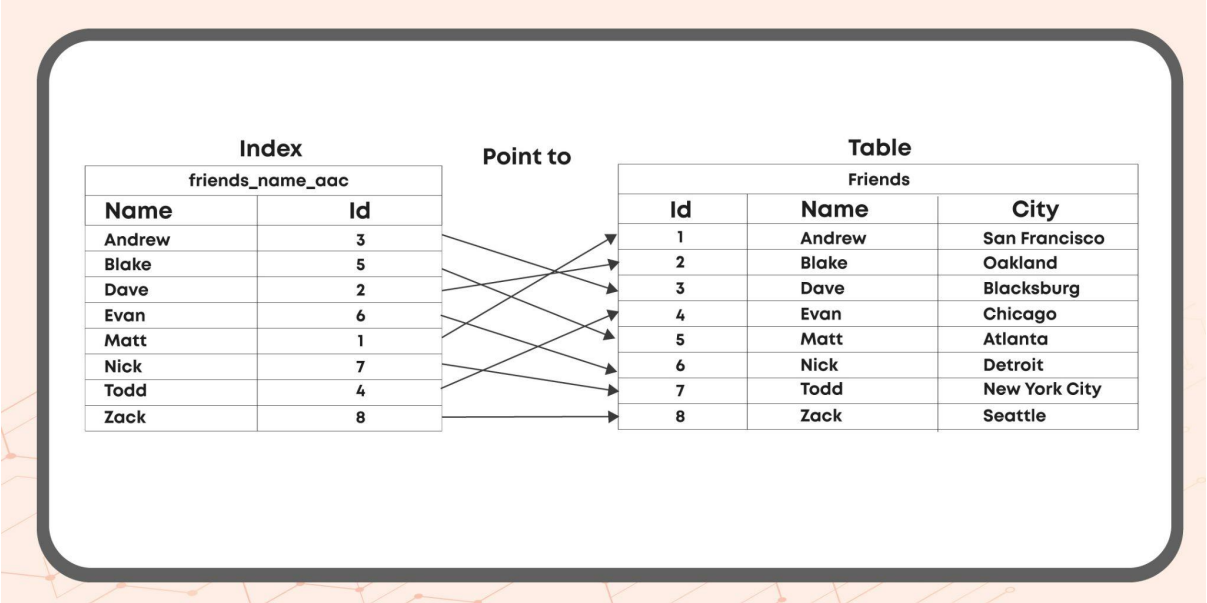


Fig: Example of Indexing in a table

How does indexing help?

Indexing has multiple advantages like:

1. Faster SELECT query
2. Helps to remove duplicates from a row or make the row unique.
3. For fill-text index, we can search against large string values like finding the string for a substring.

Disadvantages of indexing

1. Requires additional space for indexing elements
2. It slows down the Insert, update and deletes query because during updating the index should also be updated but speeds up the update if the where conditions take an indexed field.

Partitioning

What is partitioning?

Partitioning is the database process of dividing a very large table into several smaller parts. By breaking large tables into smaller individual tables, queries that access only a small part of the data can run faster because there is fewer data to scan. Partitioning is done when the data is huge and a single machine cannot handle all the data.

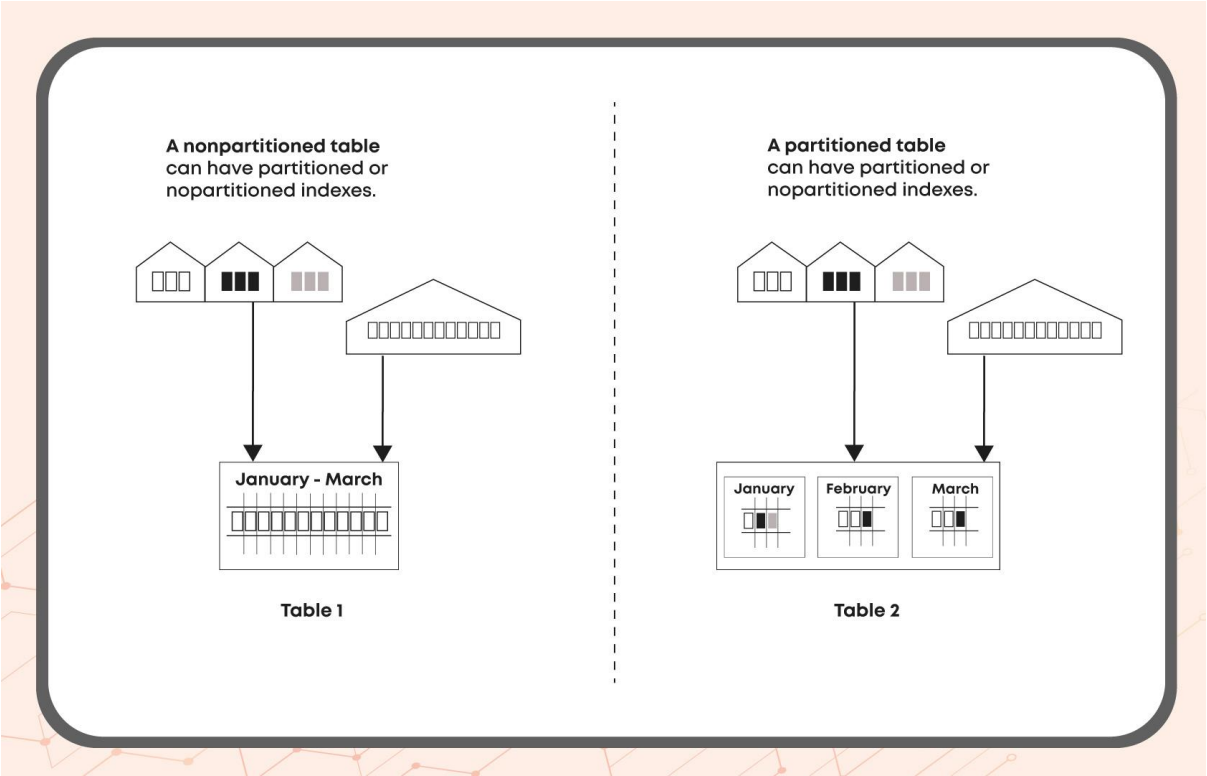


Fig: Non-Partitioned vs Partitioned Table

Advantages of partitioning

1. Scalability:
   For an application with a large amount of data, all the data cannot fit in a single machine. When the data increases the data must be split and put in different machines. Therefore, partitioning helps in achieving the high scalability of the application.
2. Manageability:

Partitioning of data makes it more manageable. Processing queries on smaller chunks of data is much more efficient.

3. Performance Optimization:
   Since the data is split into different machines. All the queries should be targeted on the same machine and can be concurrently run on all the smaller machines increasing performance and speed.
4. Availability:
   All the data is not contained in a single machine so even if a node fails or breaks down, the other nodes would be up and running and delivering data.
5. Load Balancing:
   Load is distributed over multiple machines and no single machine is overloaded now.

Disadvantages of partitioning

1. More hardware management: increased DevOps load
2. Software complexity: need to maintain the logic of splitting data, routing queries, aggregating compute results
3. More overhead when combined with redundancy

Partitioning Methods

There are multiple partitioning methods or strategies like:

Horizontal Partitioning/Shading/Range-based partitioning:

Horizontal partitioning splits large tables into smaller manageable parts without the need to create separate tables for each part. The data in a partitioned table is physically stored in row groups called partitions. Each partition can be accessed and saved separately. Each shard has the same schema as the original database in horizontal partitioning.

Example: Perhaps customers with ZIP codes less than 50,000 are stored in East Customers and customers with ZIP codes greater than or equal to 50,000 are stored in West Customers. The two partitioned tables are CustomersEast and CustomersWest, and a join view can be created on both to provide a complete view of all customers.

Challenges of horizontal partitioning:

1. Range-based:
   During the range-based partitioning if the data is not equally distributed then that may lead to an unbalanced state.
2. Increased complexity:
   Shading may increase the complexity of the system since they are located on different machines.
3. Expensive: In cases of range read queries, or queries spanning across multiple machines; we need to fetch results from all machines and aggregate results, hence can make it expensive.

Vertical partitioning:

Vertical partitioning involves creating tables with fewer columns and using additional tables to store the remaining columns. Data is broken down in the vertical style. Vertical table partitioning is mainly used to improve SQL Server performance, especially when the query retrieves all columns from a table that contains a large amount of text or BLOB columns.

Example: Suppose you have a large table containing employee reports, containing basic information such as report name, ID, number of reports, and a large column with report descriptions. Suppose that about 95% of the users are searching for the name, number, etc. of the report, and only about 5% of requests open the report description field and see the description. Assuming that all of these searches will result in a clustered index scan, and since the index scan reads all the rows in the table, the cost of the query is proportional to the total number of rows in the table. Our goal is to minimize the number of I / O operations and reduce the search cost. To reduce query costs, we will change the SQL Server database schema and split the EmployeeReports table vertically.

Challenges of vertical partitioning:

1. Slow JOIN queries:
   To join queries on different tables, the system has to make network calls to each of the tables and then join the data. The table which is partitioned on multiple machines makes it very slow. JOIN operations are required to get data from different shards which may lead to slower join queries.

2. Further partitioning required:

Sometimes further table partition is required if one particular table has grown large and it needs additional partitioning.

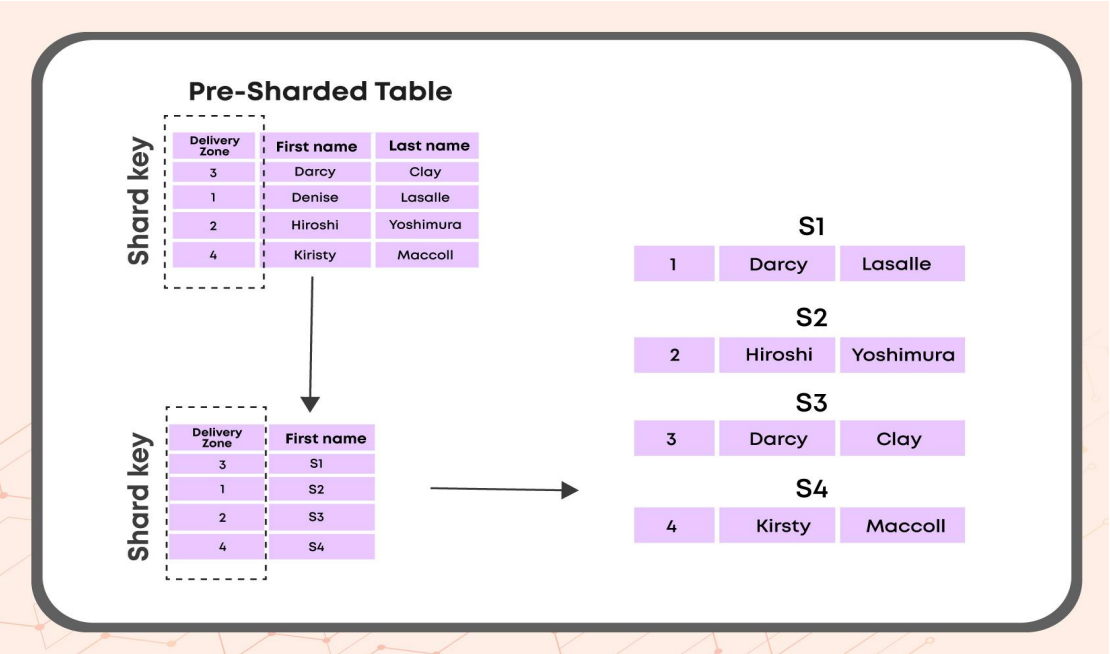
Directory-Based Partitioning:

A search service that knows the partition scheme and abstracts it from the database access code. It allows adding database servers or changing the partition scheme without affecting the application. It leads to a loosely coupled application and a horizontally scalable application. Directory-based is more flexible than the key-based or range-based partitioning as in

- key-based partitioning we need to use a hash function that cannot be modified frequently.
- In range-based partitioning, the values of ranges are decided which cannot be modified.

But, in directory-based partitioning we can use any algorithm to assign data in the shards, it is a more dynamic approach and hence more flexible.

Example: To use directory-based sharding, you'll need to design and maintain a lookup table that uses a shard key to track which shard contains which data. A lookup table, in a nutshell, is a table that contains a static collection of information about where specific data may be found. A simple example of directory-based sharding is shown in the diagram below:



Challenges of Directory-Based partitioning:

1. Complex:
   This kind of partitioning is complex to implement because the directory server has a mapping of each record with their corresponding servers.
2. Single Point of Failure:
   If the directory server breaks down or fails, it would be a single point of failure scenario. Therefore, redundancy is essential in such partitioning.

Partition Criteria

1. Key or Hash-Based Partitioning:
   We apply the hash function to the key attribute of the entry to find out the partition number.

   Issues with hash-based partitioning:

   1. Dynamic adding or removing servers is difficult.
   2. During the migration of data, the hash function should be changed.
   3. Changing the hash function requires re-distribution of data.
   4. Redistribution causes server downtime.
   5. Re-distribution is also very expensive.

   Example: Partition based on the year in which a student took admission in the particular school.

```
CREATE TABLE students (
   id INT NOT NULL,
   fname VARCHAR(30),
   lname VARCHAR(30),
   admitted DATE NOT NULL DEFAULT '1970-01-01',
   separated DATE NOT NULL DEFAULT '9999-12-31',
   student_code INT,
   store_id INT
)
PARTITION BY HASH( YEAR(admitted) )
PARTITIONS 4;
```

2. List Partitioning:

The partition is selected on the basis of the column that matches one of the sets of discrete values. The particular partition is assigned a list of suitable values.

Example:

```
PARTITION BY LIST(store_id) (
   PARTITION pNorth_Delhi VALUES IN (3,5,6,9,17),
   PARTITION pEast_Delhi VALUES IN (1,2,10,11,19,20),
   PARTITION pWest_Delhi VALUES IN (4,12,13,14,18),
   PARTITION pCentral_Delhi VALUES IN (7,8,15,16)
);
```

3. Round Robin Partitioning:

   Suppose we have n partitions, then the ith tuple is assigned to partition number i%n. That means that the ith data would go to (i%n) nodes. Data is assigned in sequential order. This partitioning criterion ensures the even distribution of data.

4. Consistent Hashing:

   This is a new kind of partitioning.

   There was an issue with the hash-based partitioning that on adding the new servers, the hash function had to be changed. Changing the hash function would mean redistribution of data and server being non-available. This would be very expensive.

   In the calculation, the consistent hashing is a special kind of hashing, so when the hash table is resized, only the keys whose number of keys are and need to be redistributed on average. It is a technique that is used to build scalability into the system's storage architecture from scratch.

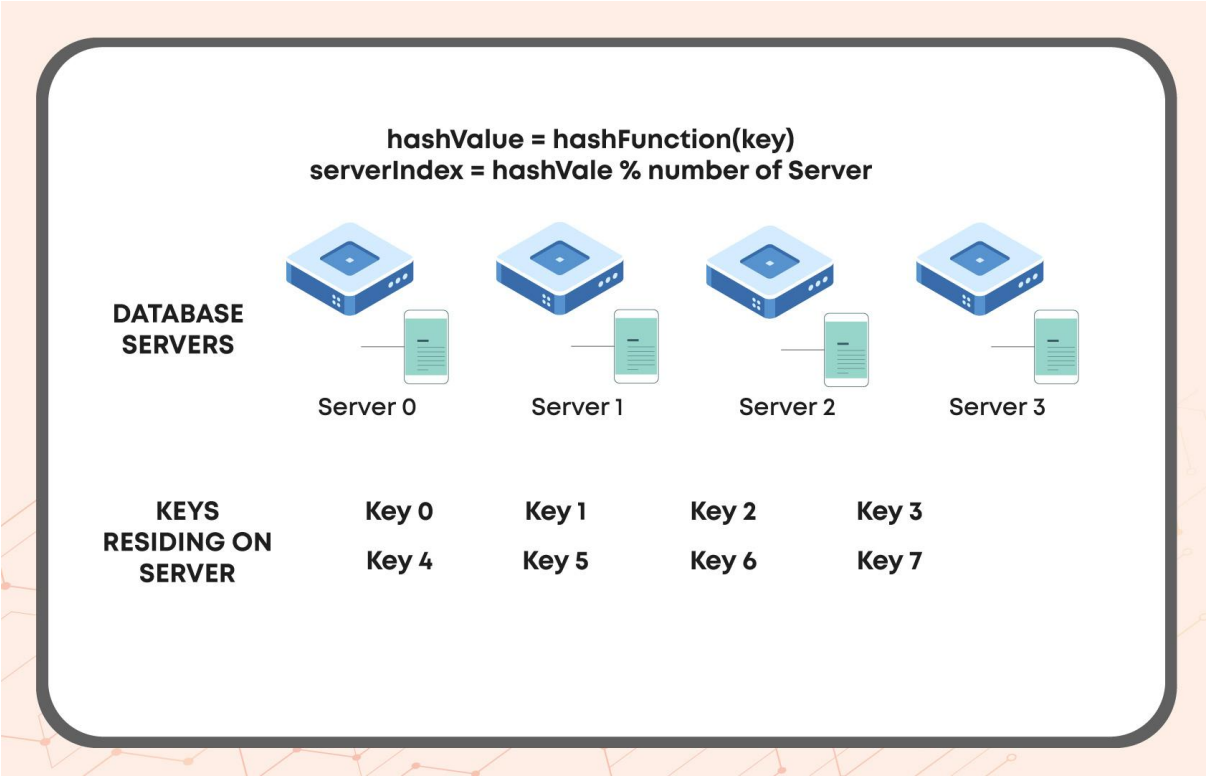
   The approach of consistent hashing is as follows:

1. hash_Value = Hash_Function(Key)
2. server_Index = hash_Value % n

Example:

- Suppose we have 4 database servers
- Suppose our hash function returns a value from 0 to 7
- We suppose that "key0" generates a hash value or 0 when passing our hash function, "key1" returns 1, etc. Etc.

- The server_Index of "key0" is 0, "key1" is 1, and so on.



Caching

Caching is an important topic of system design and is a widely used technique with multiple benefits.

You must have noticed that many websites are super fast and load the elements very fast, while some new websites take a lot of loading time. Slow loading gives a poor user experience. In order to prevent slow loading like excessive buffering during streaming a video, websites use caching, which helps speed up the process.

What is Caching?

In computers, a cache is a high-speed data storage layer that caches a chunk of data that is typically transient so that subsequent requests for that data can be delivered up faster than if the data were accessed directly from its primary storage location. Caching allows you to reuse data that has been previously retrieved or computed quickly. Caching helps in the reduction of the number of read calls, API calls and network I/O calls by creating a local instance of the static information.

Some commonly cached items include:

1. HTML pages (partial or complete)
2. API responses
3. Database queries

How does Caching work?

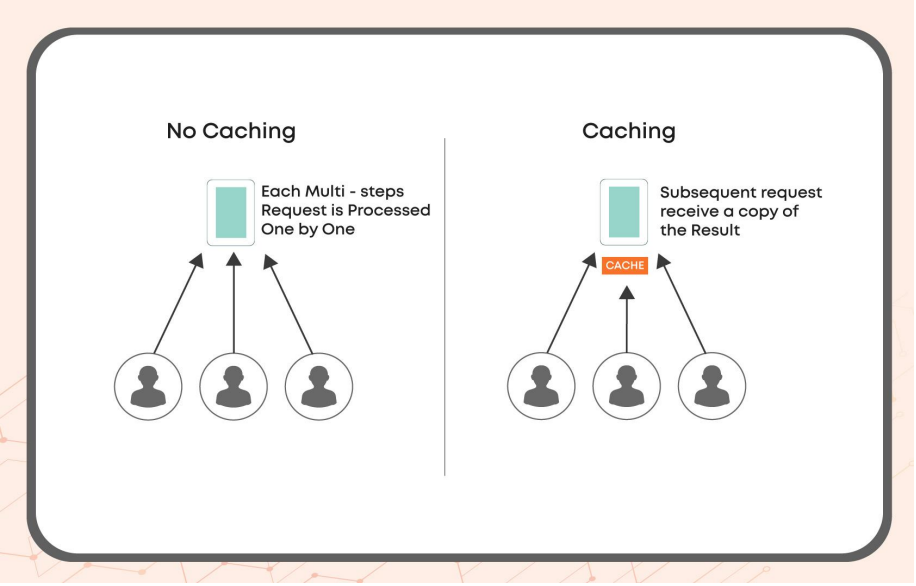


Fig: This is how Caching functions

The data in a cache is most of the times kept in rapid access hardware like RAM (Random-access
memory), but it can also be utilised with a software component. The fundamental goal
is to improve data retrieval performance by eliminating the need to contact the slower
storage layer behind it.
In contrast to persistent storage, a cache often stores a subset of data transiently.

What places can caching be used?

A web application would have a data storage layer, frontend and backend layer if the data in the database is not frequently changed or updated, instead of again and again accessing the required data from the database, which adds on the response time. The Backend layer can have a temporary instance of the database, which is known as a cache. Caching is inbuilt in the Operating System for this above algorithm. Caching can be used for JS or CSS components if they do not change frequently. Caching can also be used in the frontend if some element is not changing frequently; we can cache it. Caching can be used at various places.

Significant applications of caching are:

1. CDN(Content Delivery Network)
2. Application Server Cache

Application Server Cache:

We can locally store this information in memory for a web application consisting of front-end, heavy computation or a database, and the elements are not frequently changed or updated. This local storage is known as a cache.

CDN(Content Delivery Network):

When the frontend has multiple static elements like CSS, JavaScript, etc., which are not changed frequently, instead of serving them from the backend, they can be delivered from a third party layer known as Content Delivery Network. They cache static content and provide the data at a faster rate to the frontend or application.

Types of Caching Solutions

There are two types of cache:

1. In Memory/ Local Cache:
   It is used for a single system when the cache must be stored in the local memory.
   Example: Google Guava Cache, Memcache

2. Distributed Cache/External Cache:
   It is used when the cache needs to be shared between multiple systems. Therefore, we store the cache as a distributed system and can be shared by all the servers.
   Example: Redis

Deciding when to use caching:

1. Static Data:
   Caching would be useful if the data is not changing very frequently, we can store it and use it directly. In case the data is changing quickly, caching would not help much.

2. Type of Application:
   The application can be either read-intensive(number of read operations are more than the number of write operations like Wikipedia) or write-intensive(number of write operations are more than the number of read operations like twitter). Caching would be more useful for the read-intensive application. For a write-intensive application, data would be changing very fast and therefore caching should not be used.

Caching Strategies

Caching patterns are the design patterns for integrating a cache into a system are known as. Cache-aside or gradual loading (a reactive strategy) and write-through are two standard methods (a proactive approach).

a). Write-Around Cache

In the write around the cache, data is directly written to the permanent storage and not the cache so that the write operations on cache are reduced. Therefore, the cache line(the memory transferred to cache) is invalidated without writing back because the permanent storage is already up to date. When the data is not found in the cache, the situation is called a cache miss.

Cache Miss: A cache miss happens when a data entry is requested from the cache, but the entry does not exist for the supplied key (or simply, a miss).

Advantages of write-around cache:

1. Faster write operations
2. Slow read operations due to a cache miss

b). Write-Through Cache

The order in which the cache is populated is reversed in a write-through cache.

1. The primary database is updated by the application, batch, or backend process.
2. The data in the cache is also updated immediately afterwards.

Advantages of write-through cache:

1. No Stale Data
2. High Consistency

Disadvantages of write-through cache:

1. Slow write operations

c). Write-Back Cache

Write back is when data is written in cache only when the data changes and writing is done only to the cache.

Advantages of write-back cache:

1. Write operation is very fast

Disadvantages of write-back cache:

1. Loss of data in case the cache fails

Cache Eviction Strategies

The sequence in which entries are removed from a full cache is determined by the eviction policy of the cache (used for removing the old data or replacing it with the new data). It is used to make space so that new entries can be added.

1. LRU (Least Recently Used):
   LRU (Least Recently Used) is a popular eviction policy that prioritises removing entries that haven't been used in a long time. To keep track of the relative utilisation of entries, LRU requires additional RAM.

2. MRU(Most Recently Used):
   The entry which is most recently used is evicted. We keep a track of the data most used and remove it and leave the one which is less used.

3. LFU (Least Frequently Used):
   LFU (Least Frequently Used) prioritises the removal of the least frequently used entries. LFU, once again, necessitates additional memory to keep track of the utilisation of entries.

4. FIFO(First In First Out):
   FIFO removes items from the cache in the order in which they were added. It evicts the first data that was added.

5. LIFO(Last In First Out):
   LIFO evicts the last added data.

6. RR(Random Replacement):
   This makes the cache random. It is used when all the elements have an equal probability of getting evicted.

Benefits of Caching:

1. Improve Application Performance
2. Reduce Database Cost
3. Reduce the Load on the Backend
4. Predictable Performance
5. Eliminate Database Hotspots
6. Increase Read Throughput (IOPS)

Disadvantages of Caching

1. Cache memory is expensive and has a limited capacity.
2. As information is stored in the cache, it makes the page heavy.
3. Sometimes the updated information does not show as the cache is not updated.

Applications of Caching:

Caches can be used in various technology levels, including operating systems, networking layers such as Content Delivery Networks (CDN) and DNS, web applications, and databases.

1. Many read-intensive application workloads, such as Q&A portals, gaming, media sharing, and social networking could benefit from caching to improve latency and IOPS.
2. Database queries, computationally complex calculations, API requests/responses, and web artefacts such as HTML, JavaScript, and picture files are all examples of cached data.
3. An In-Memory data layer operating as a cache benefits compute-intensive applications that manipulate data sets, like recommendation engines and high-performance computing simulations.
4. Massive data sets must be accessed in real-time across clusters of servers that can span hundreds of nodes in these applications. Manipulation of this data in a disk-based store is a significant constraint for many applications due to the speed of the underlying hardware.

Load Balancers

Let us suppose you are working for an IT Company and experience that the company doesn't have any management system. Some employees do not have any tasks, and some are overburdened with responsibilities and functions. You discuss the issue with the company's CEO, and he appoints a manager to your team. The manager now considers all aspects like capabilities, availability, additional responsibilities, etc., and assigns work. The team is now better managed, and no employee faces either shortage of work or feels overburdened. The team is now a better place to work mentally and physically. We can say that the manager worked towards load balancing and can be described as a load balancer.

Let's understand more about load balancing.

What is load balancing?

Load Balancing is the process of efficient distribution of application or network traffic across all nodes in a distributed system.

The devices used for ensuring load balancing are load balancers.

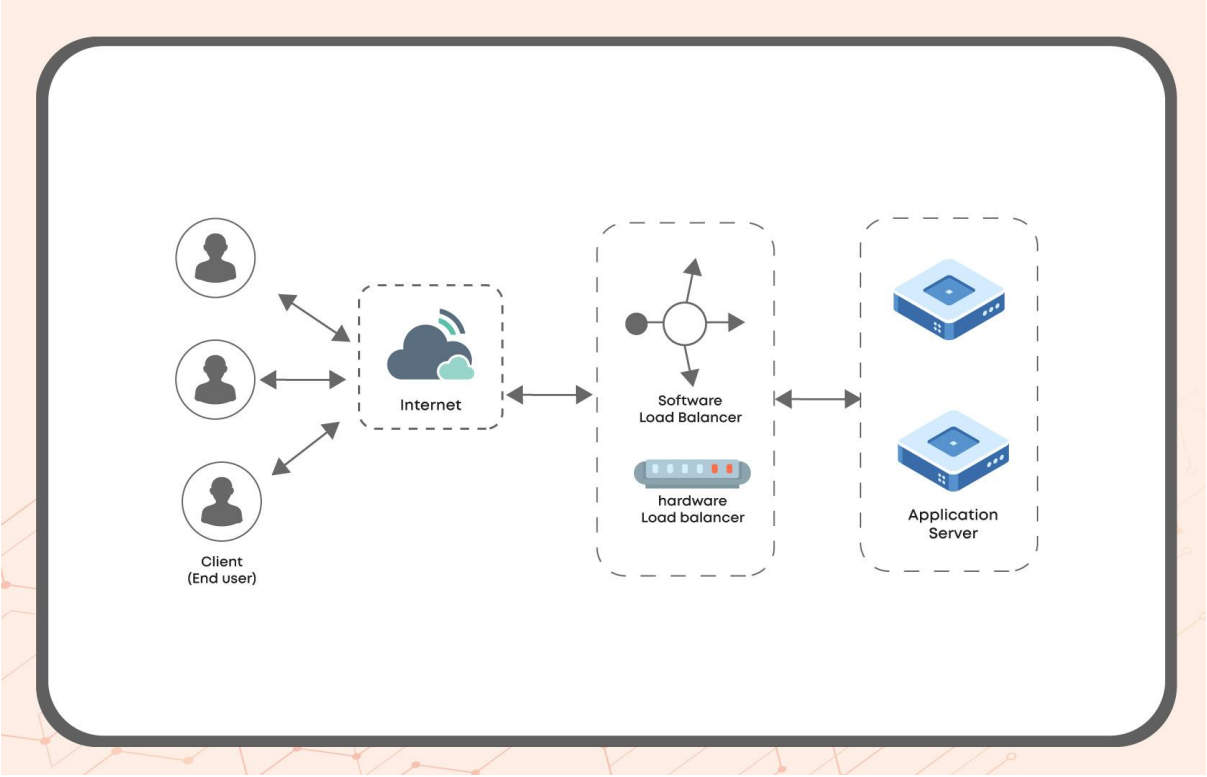


Fig: Load Balancing

Roles of Load Balancer

1. The work or load distribution is equal over every node.
2. Should keep a check on which nodes are not operational or available.
3. Efficiently distribute/manage work so that it is completed before time.
4. Distribution should be such that the total capacity is utilised and speed is maximised.
5. Load balancers should ensure high scalability, high throughput and high availability(if the node is not operational, the request is passed to another node that is up and running).

Where should the load balancer be placed?

Load balancers are placed where it has the capacity to terminate connections to the public IPs. Load balancers can be placed to manage multiple ICs(Industrial Control Systems). For example, the three layers of a system: FrontEnd, BackEnd and DataBase consist of various servers. The load balancer can be placed at multiple places like: in front of the Front-end , Back-end and data storage layers. Load Balancers helps to manage multiple instances and supports horizontal scaling.

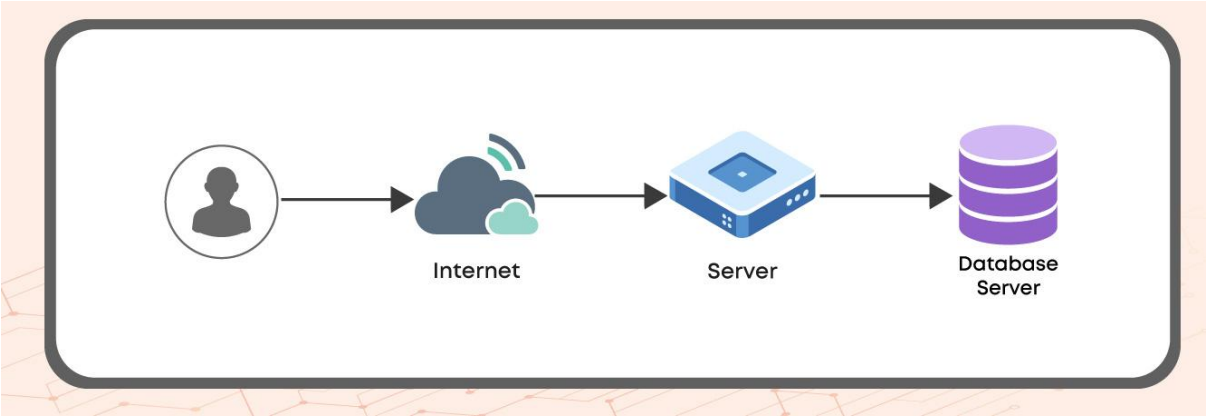


Fig: Load Balancers

Benefits of Load Balancing

1. <u>Optimisation:</u>
   Load Balancers helps in resource utilisation and lower response time, thereby optimising the system in a high traffic environment.

2. Better User Experience:
   Load balancers help in reducing the latency and increasing availability, making the user's request go smoothly and error-free.

3. Prevents Downtime:
   Load Balancers maintain a record of servers that are non-operational and distribute the traffic accordingly, therefore ensuring security and preventing downtime, which also helps increase profits and productivity.

4. Flexibility:
   Load Balancers have the flexibility to re-route the traffic in case of any breakdown and work on server maintenance to ensure efficiency.

5. Scalability:
   When the traffic of a web application increases suddenly, load balancers can use physical or virtual servers to deliver the responses without any disruption.

6. Redundancy:
   Load Balancing also provides in-build redundancy by re-routing the traffic in case of any failure.

When to use Load Balancers?

1. When the application has multiple instances or multiple servers, load balancers can be used for load management in such cases.
2. Application traffic is distributed over multiple nodes or servers.
3. In a high traffic environment, load balancers are essential for ensuring scalability, availability and latency.

Challenges of Load Balancing

1. Single Point Of Failure:
   During a load balancer failure or breakdown, the complete system is interrupted and will be unavailable for the time periods leading to a bad user experience. During a load balancer malfunctioning, the communication between clients and servers would be broken. To solve this issue, we can use redundancy. The system can have an active load balancer and one passive load

balancer. In case the active load balancer goes down, the passive can become the active load balancer then.

Load Balancing Algorithms

Multiple algorithms can be used for the efficient distribution of load across different nodes or servers. The algorithm should depend on the type of application the load balancer has to be used for. Here are some load balancing algorithms:

1. Round Robin :
   Round robin is one of the most used and simplest load balancing algorithms. The user's requests are distributed to the server in a rotation fashion.
   Example: Suppose there are five servers(A, B, C, D, E) in a distributed system. There are ten requests from the client. The first request would go to A; the second request would go to B; the third request would go to C; the fourth request would go to D; the fifth request would go to E and then back to the first. The sixth would go back to A, seventh to B and so on.

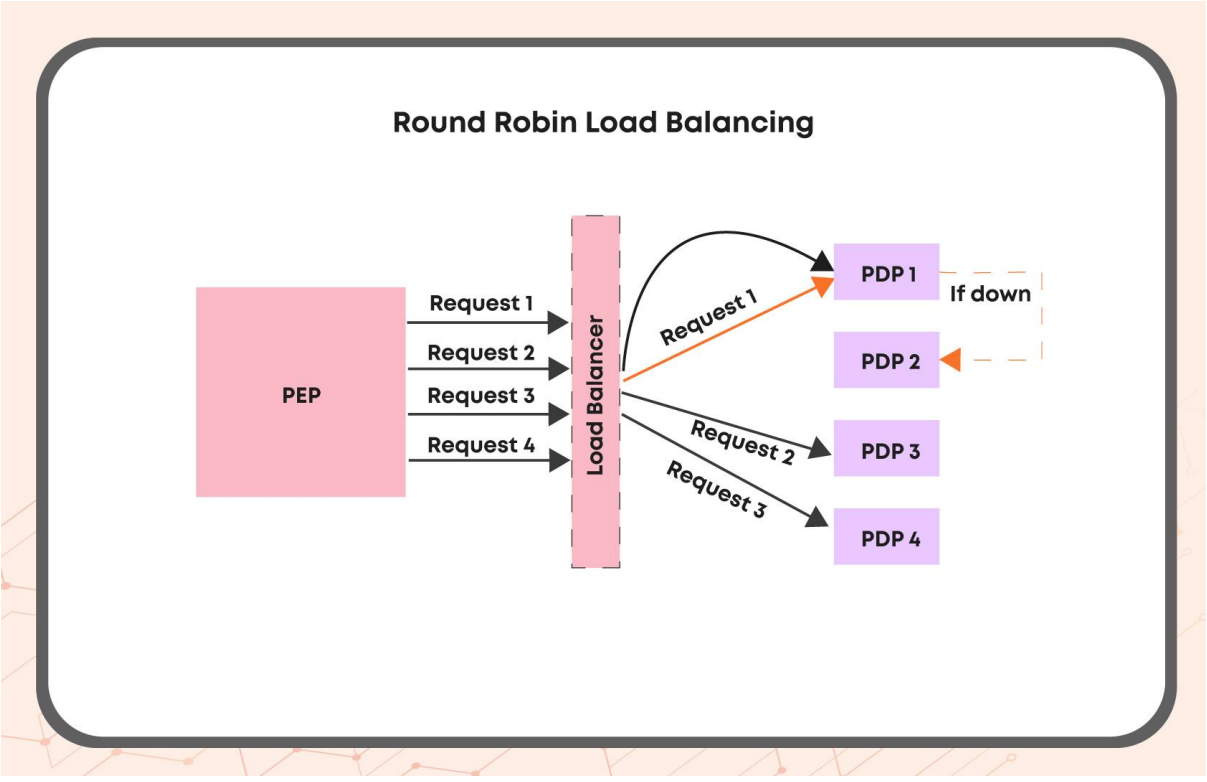


Fig: Round Robin Load Balancing Algorithm

2. Weighted Round Robin:
   It is similar to the Round Robin when the servers are of different capacities.

Example: Suppose there are two servers(A. B) in a distributed system. There are four client requests. The capacity of Server B is twice the capacity of Server A. The first request is distributed to server A, the second request is given to server B, but since the capacity of server B is twice server A, the third request should be given to B. The fourth request is passed to server A then.

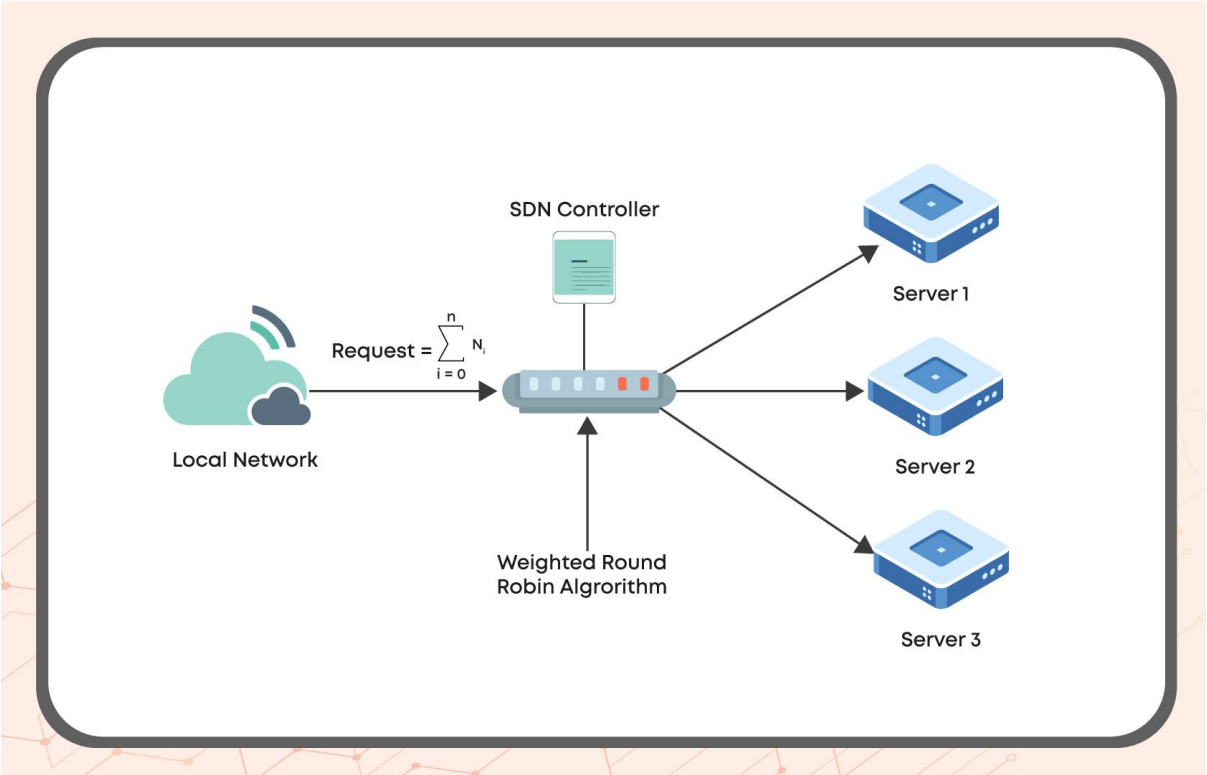


Fig: Weighted Round Robin Algorithm

3. Source IP Hash:
   Source IP Hash combines the server and client's source and destination IP addresses to produce a hash key. The key can be used to determine the request distribution.

4. IP Hash Algorithm:
   The servers have almost equal capacity, and the hash function is used for random or unbiased distribution of requests to the nodes.

5. Least Connection Algorithm:
   This algorithm is used when deciding based on the availability of open connections on a server.

6. Least Response Time:
   In this algorithm, the request is distributed based on the server which has the least response time.

Synchronous Communication

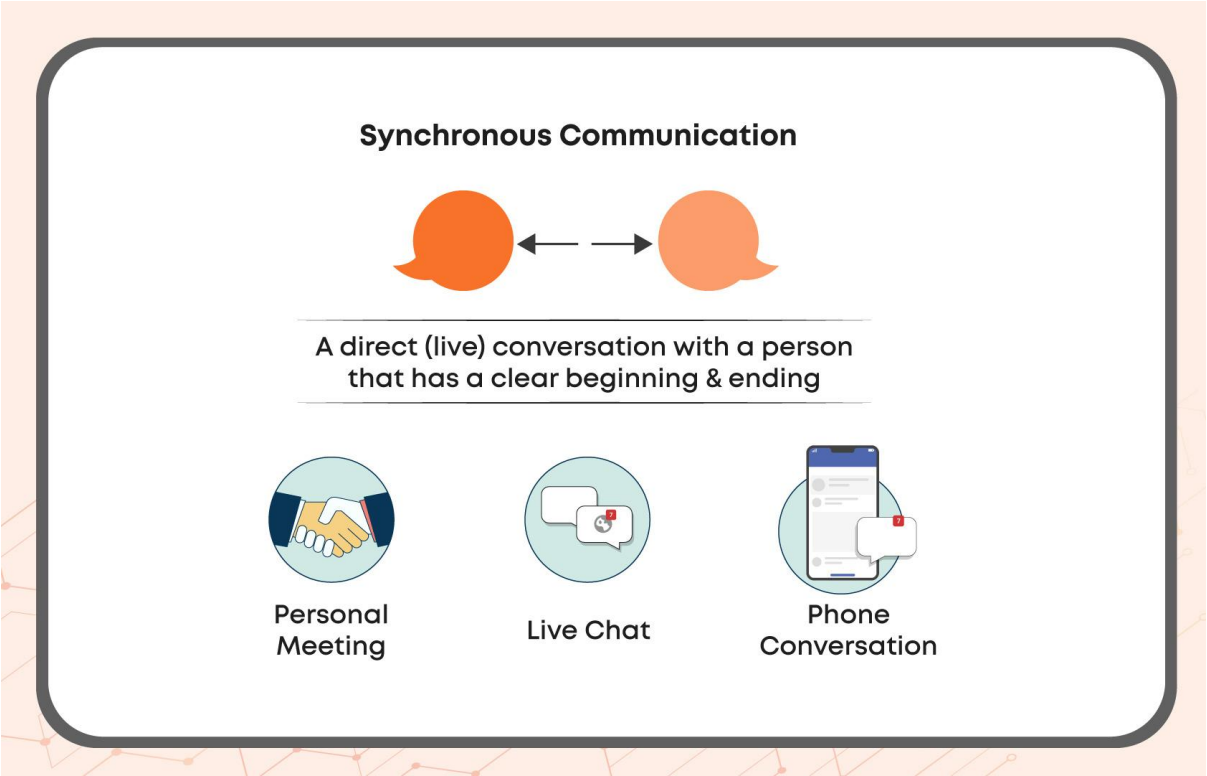
Suppose you go to an Icecream Parlour and buy a Mango Popsicle and pay the money to the ice cream vendor. You have to wait for some time till the vendor prepares your order and delivers it to you. Till the time order is being prepared, you cannot leave the place and this is an example of synchronous communication. Since you are "blocked" until you receive your order, it can also be called a blocking call.

Synchronous Communication is the "Communication in sync".

What is synchronous communication?

Synchronous communication is a type of communication between two or more parties where they exchange information from start to finish without any interruption. If the client has made a request and keeps waiting until it is satisfied then that is an example of synchronous communication/blocking call.

Example: An application A requests some data from application B without which it cannot proceed on the next processing step. Application A is blocked or the process is halted until application B responds with the required information.



Synchronous Communication

Why is synchronous communication required?

1. Real-Time Communication:
   Synchronous communication is essential when the information exchanges can happen in real-time.
   Example: Two applications are dependent on each other for sharing information. The application has to wait until its request is fulfilled by the second application.

2. Transaction:
   In synchronous communication, the exchange has to take place from start to finish, that is the thread/system/client must wait till the other point of communication has responded. The system has to wait till the successful completion of the process.
   Example: During a money transfer, either the system must NOT transfer money and show some error or transfer it successfully. In such situations, synchronous communication is essential.

3. Consistency:
   High Consistency is ensured when all the replicas of the data are updated with the most recent data after a write operation takes place. The read operations are blocked until all the replicas are updated and their synchronous communication would be required to achieve high consistency.

How is the synchronization call achieved?

The synchronous call is based on the concept of blocking. Till the request is not completed the node making the request has to halt and wait until the response is received.

Example: Thread A1 of an application requests information from thread A10 of that application. Until the processing to deliver the request is not completed by thread A10, thread A1 has to wait and block the process. The process would continue once the request is completed by thread A10. A Real-Life Example of this would be talking with your friend over a video call.

Asynchronous Communication

Now, when we are already aware of synchronous communication, the process is blocked until the request is complete. Asynchronous is opposite to it. You mail your colleague regarding some office work. You are not supposed to halt your daily activities, wait for the email response, and then continue your routine. When you do not need to wait and can do anything, communication is an example of asynchronous communication.

Asynchronous communication is “Communication not in sync”.

What is asynchronous communication?

Asynchronous communication is a type of communication when the client has the freedom to start or pause other processes instead of waiting for the response. The client can do anything on the application without waiting for the response, as it can take some time for the response to arrive.

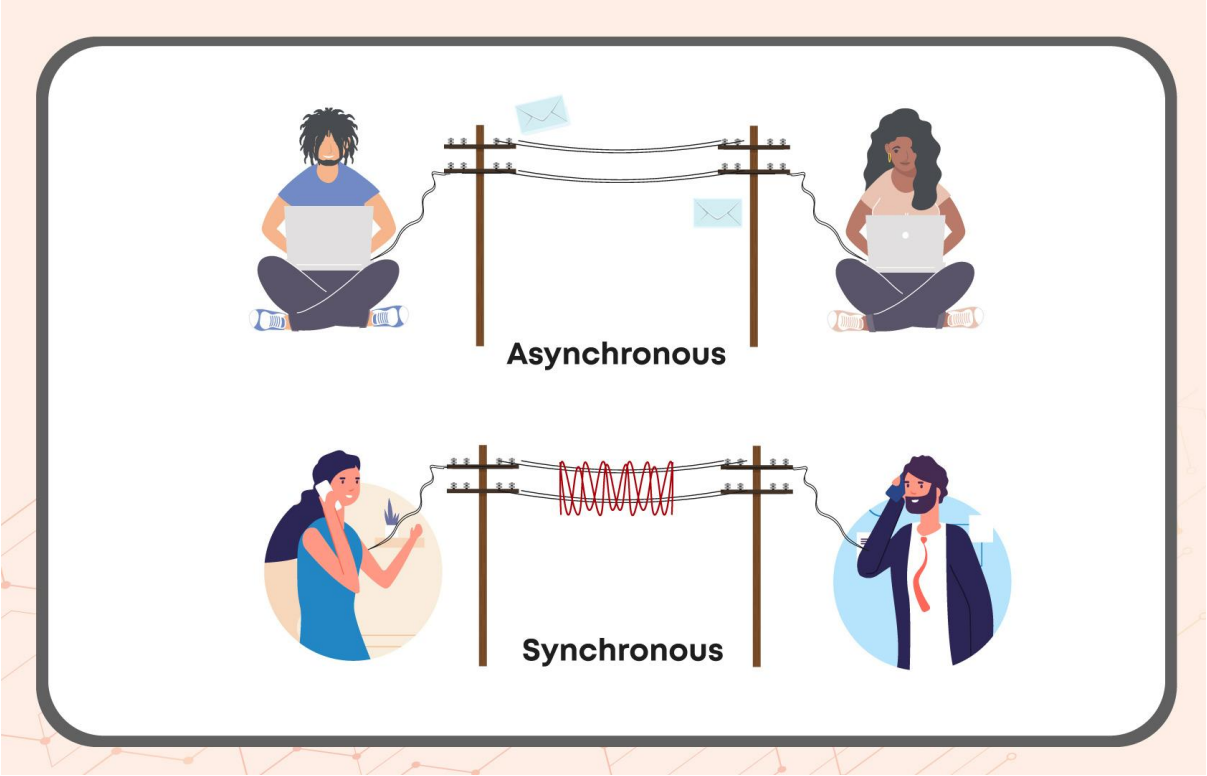


Fig: Asynchronous communication vs Synchronous Communication

Examples of asynchronous communication:

- Messaging
- Email
- Intranet
- Social Media
- NewsGroups
- World Wide Web

Why is asynchronous communication needed?

1. Computation takes a lot of time:
   If the computation and pre-processing for a response takes a lot of time, it would be more efficient for the client to start some other process and revisit when possible or available.
   Example: The notification service in a company can be slow because of network congestion or numerous requests. In such a situation instead of creating an open connection, it is better to use asynchronous communication that would save time and resources.

2. Scalability/Performance of the application:
   Since the client or the process is not blocked or doesn't have to wait for the response, time and resources are saved, leading to better performance of the application.
   Example: You have bought some product from amazon, and the last step of ordering is the confirmation email. It might take a lot of time to receive the notification email depending on many factors like traffic etc.; the client can resume other activities instead of sitting idle and waiting for the email.

3. Huge Rate Differences:
   If the request rate of the client is much higher than the response rate, then the other part will lead to cascading failure(Failure due to overloading systems) because of increased load.
   Example: Application A makes requests with a rate of 1000 requests/second. At the same time, application B can only deliver 200 responses/second. After some time, application B would be overloaded and breakdown leading to cascading failure because of the considerable differences in the rates.

How do Asynchronous Communication works?

1. Request-Response Pattern/Tracking ID Based:
   A tracking id is issued by the application to the request. The requesting application/client can check the job's progress using that tracking id, and a response is delivered once the computation is completed.
   Example: Application A makes a request to application B. Suppose the request would take 1 hour to complete. Application B gives application A a tracking id, say S1. Application A keeps checking the status using id S1. After one hour, when the computation or pre-processing is completed, the response is delivered by application B.

2. Message Queue Based:
   There is no direct communication between both ends. The request is pushed in the queue for application 2. Once the request is completed, Application B notifies application A that the job is complete.

Applications of Asynchronous Communication:

1. Ecommerce/Booking Involving:
   Any application involving Booking confirmation emails/messages/notifications use asynchronous communication because there is no fixed time that the user needs to wait to receive the confirmation message.

2. Bank:
   A bank has multiple sections like transactional service and reporting service. Once a transaction is made, the reporting service has to generate a report for the entire working day. Since it is not an immediate required service with no fixed time, it is also completed using asynchronous communication.

3. Sentiment-prediction system:
   The sentiment prediction system can be developed for predicting the sentiments based on tweets on Twitter. Twitter has a dynamic load depending on various factors. Synchronous communication cannot happen here because sometimes the prediction system would have numerous requests, leading to overloading and cascading failure and very few requests. Therefore, we can use queue-based asynchronous communication here.

Scalability

Websites Vs Web Applications

A website is a group of interlinked web pages that provides textual or visual data that can just be read. Web sites are platforms that help users navigate and extract the required information.

In contrast, a Web Application provides dynamic data that can be read and written/manipulated on the web page. Web Application runs on a web server.

In short,

Web Applications = Website + Additional Functionalities + Interactive Elements

Example:

In a blog website like medium.com, the user can read blogs. Whereas in a web application like youtube, the user can stream videos, upload videos, and download content.

Website: codingninjas.com, google.com

Web Applications: YouTube, Twitter, Google apps

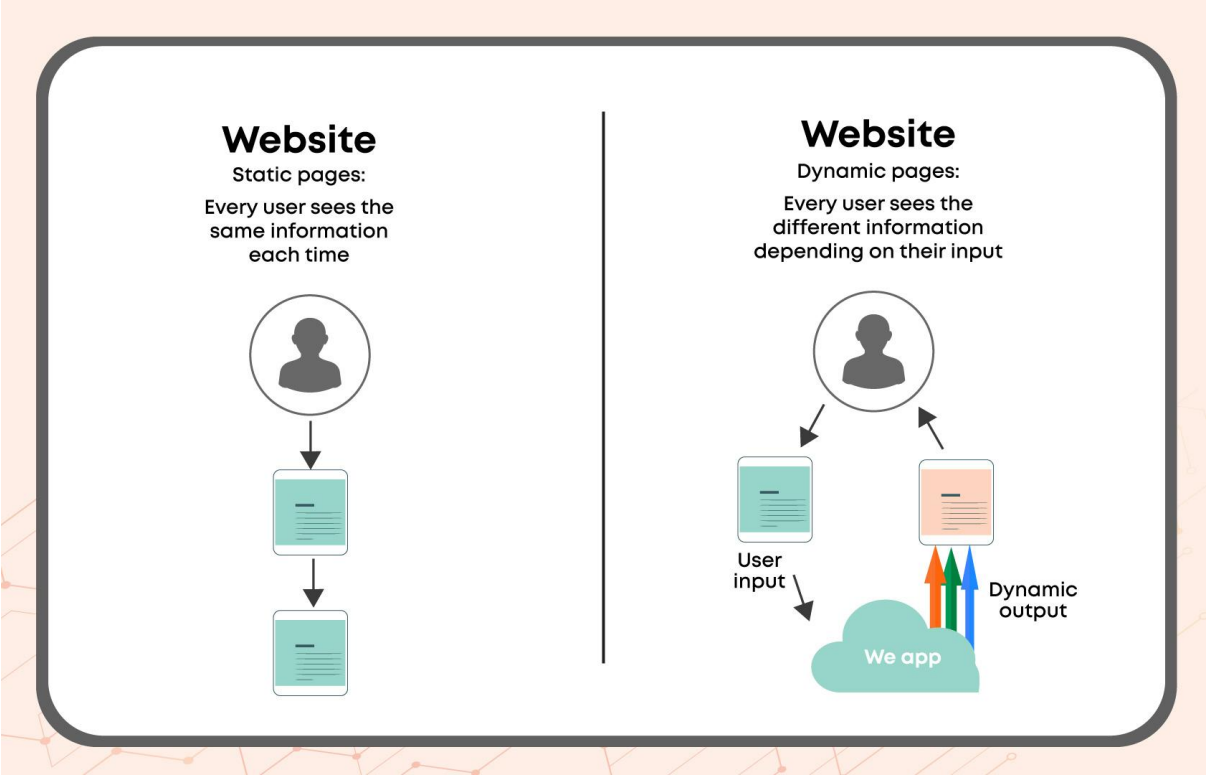


Fig: Website Vs Web Application

What is scalability in system design?

Scalability measures a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.

Scalability can be achieved in two ways: Vertical Scaling and Horizontal Scaling.

1. Vertical Scaling

Vertical scaling increases the scale of a system by adding more configuration or hardware for better computation power or storage. In practice, this would mean adding better processors, increasing RAM, or other power increasing adjustments. Scaling here is done through multi-core by spreading the load between the CPU and RAM resources.

Example: MySQL

Pros of Scaling-Up

- It consumes less power as compared to maintaining multiple servers
- Administrative efforts are reduced as a single machine is to be managed
- Cooling costs are lesser
- Reduced software costs
- Implementation becomes easy
- The application compatibility is retained

Cons of Scaling up

- There is a high risk of hardware failure which can cause bigger problems.
- There is less scope for upgrading the system.
- It can become a Single point of failure.
- There is a limit to increase resources like memory storage, RAM as one single machine might not be able to handle it.

2. Horizontal Scaling

Horizontal scaling is a process of increasing the scale of a system by adding more machines. This entails collecting and connecting multiple devices to take on more system requests.

Example: Cassandra, MongoDB

Pros of Scaling-out

- Cost-effective compared to scaling-up

- Takes advantage of smaller systems
- Easily upgradable
- Resilience is improved due to the presence of discrete, multiple systems
- Fault-tolerance can be handled easily
- Supporting linear increases in capacity

Cons of Scaling-out

- The licensing fees are more
- Utility costs such as cooling and electricity are high
- It has a bigger footprint in the Data Center
- More networking equipment is needed

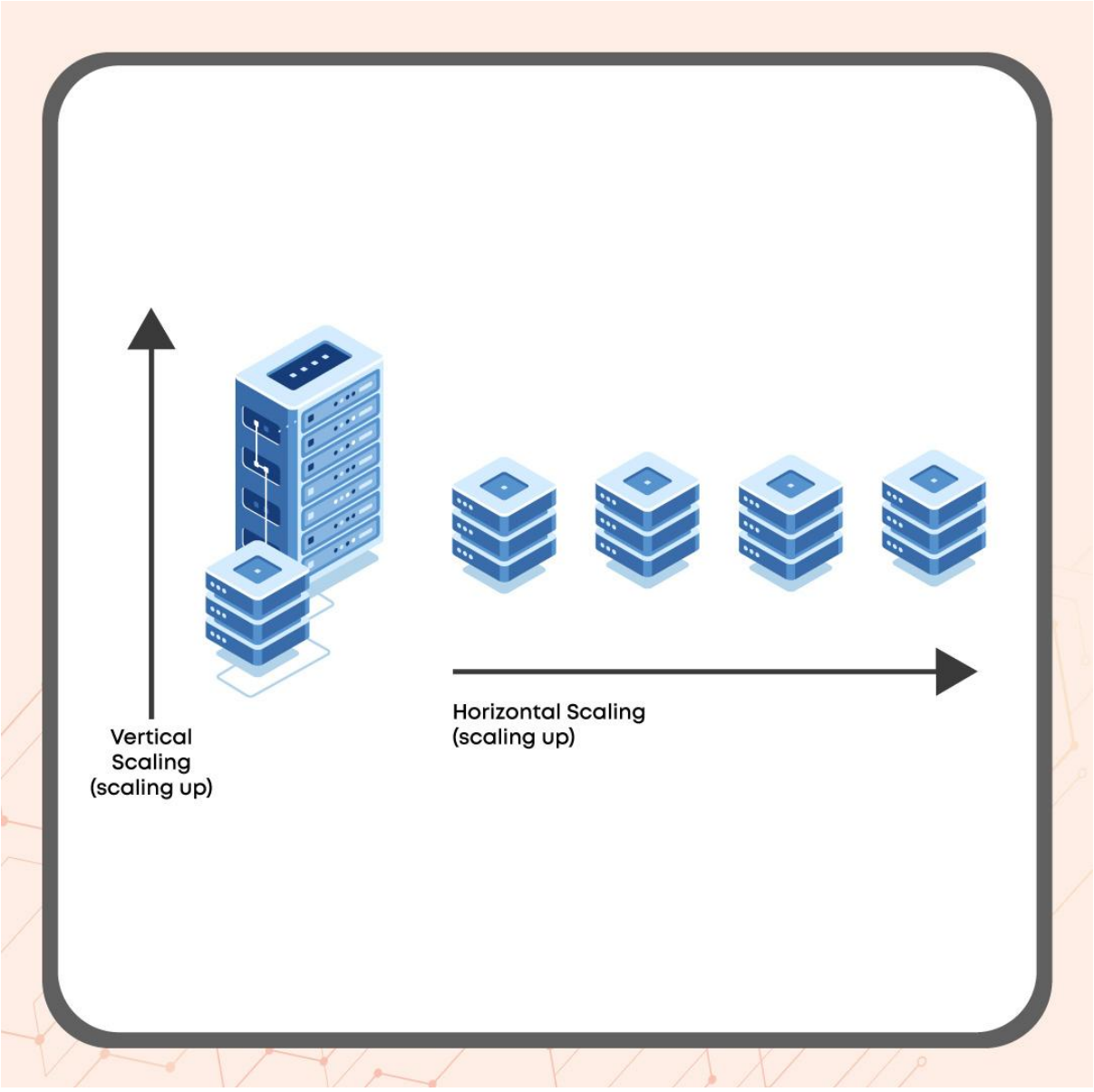


Fig: Horizontal and Vertical Scaling

Limitations —

There is no limit to horizontal scaling, but it would not be cost or space efficient compared to vertical scaling. Also, the performance of a single machine using horizontal scaling cannot continuously be increased in proportion to the hardware added because the performance curve comes to a standstill at the maximum limit.

Another limitation of vertical scaling is that it might be impossible to scale vertically after a limit due to limitations in hardware technology.

Scaling	Pros	Cons
Vertical Scaling	• Comparatively more areas of application • Lower power consumption as compared to running multiple servers • Easy to install and manage the hardware in a single machine	• Expensive, so more financial investments required. • Bigger outage due to hardware failure • Low availability • Limited upgradability in future
Horizontal Scaling	• Cheaper than vertical scaling • High availability • Easier to run integration testing and fault tolerance	• Limited applications • Higher cost on utility like electricity • Extra load since software has to handle data distribution and parallel processing

Vertical Scaling Vs Horizontal Scaling

Parameter	**Horizontal Scaling**	**Vertical Scaling**

Databases	It is based on the partitioning of data.	In this, the data lives on a single machine and scaling is done through multi-core. That is the load is divided between the CPU and RAM of the machine.
Downtime	Adding machines to the existing pool means making it possible to scale with less downtime.	Having a capacity of a single machine means scaling beyond its limit can lead to high downtime.
Data Sharing	Data sharing is complex in horizontal scaling as it consists of many machines.	Data sharing is easy in vertical scaling as single machine message passing can be done by just passing the reference.
Examples	MongoDB,Cassandra	MySQL, Amazon RDS

Communication Protocols

What is the communication protocol?

A communication protocol is a rule system that allows two or more entities in the communication system to transmit information through any type of change in physical quantities. The protocol defines the rules, syntax, semantics and timing of communication and possible error recovery methods. These protocols can be implemented using hardware, software, or a combination of both.

Example: HyperText Transfer Protocol(HTTP), WiFi

Push and Pull Model

1. Push: In the push protocol, the client opens the connection with the server and keeps it always active. The server will use this single connection to send (push) all-new events to the client each time. In other words, the server pushes new events to the client. This method reduces the server load.

   Example: Suppose your mobile phone is always connected to the mobile network. We can judge the connectivity by the signal bar on the phone screen. When the caller makes a call, the network will send the call to your mobile phone through the active connection that your mobile phone already has. This is a push.

2. Pull/Polling: In the pull protocol, the client periodically connects to the server, checks and gets (pulls) the latest events, and then closes the connection and disconnects from the server. The client repeats the entire process to receive updates on new events. In this mode, the client will periodically pull recent events from the server. It is the default method of HTTP communication.

   Example: When we are waiting for a specific show on the TV, we will switch ON the TV repeatedly and check if our particular show has started, and then close it again if it hasn't. This is pulling.

The pull Model is more commonly used than the push Model.

The flow of data from Client -> Server: Pull Model
The flow of data from Server -> Client: Push Model

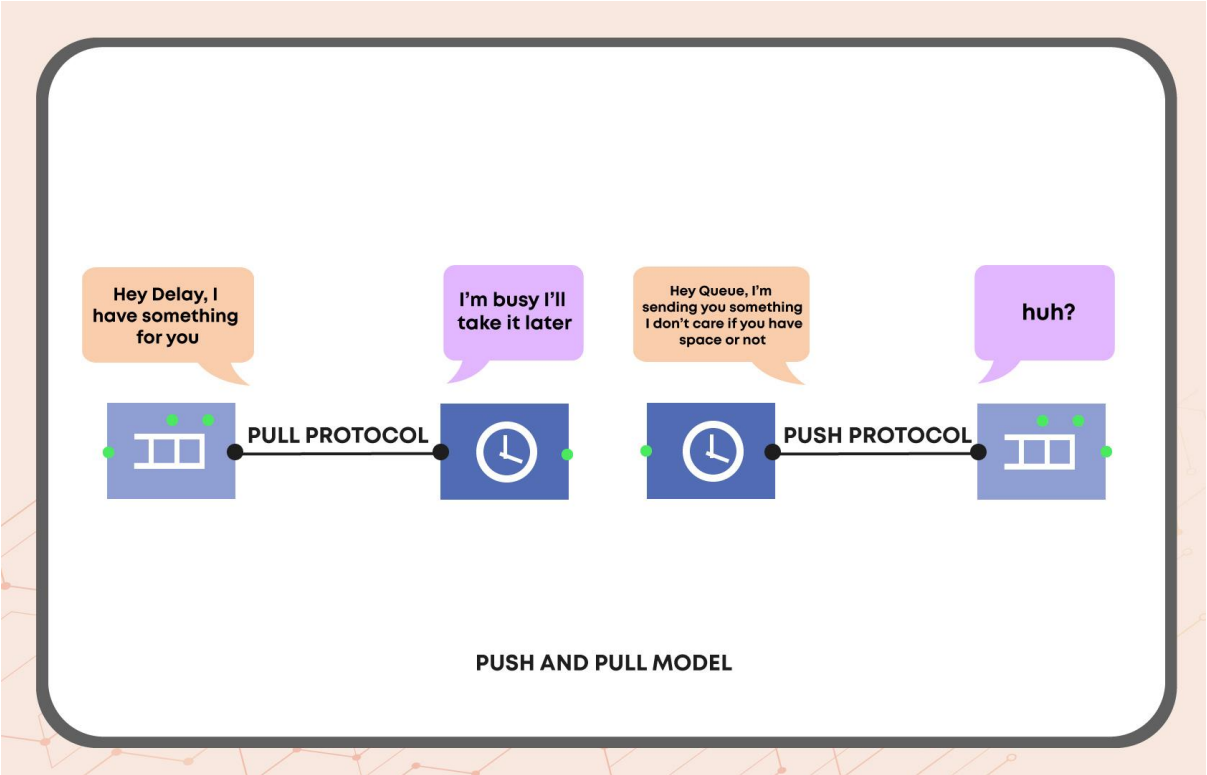


Fig: Push and Pull Model

Steps of Client-Server Communication

1. The client initiates the communication and waits for the response.
2. Computation and processing at the server end.
3. The server sends back the response.

When to use Push and Pull Protocols:

- The push approach is better in a low write rate and a large number of client use cases. (a celebrity makes a status update on Instagram and has millions of followers; so we use a push strategy for the user's news feed to update)
- The poll approach is better in a high write rate and a low number of client use cases. (a web crawler can poll for updates on wiki pages to update its index for search engines)

3. Long Polling: To overcome the deficiency of the Polling protocol, web application developers can implement a technique called HTTP long polling, in which the client polls the server for new information. The server keeps the request open until new data is available. Once available, the server will respond and send further

information. When the client receives new information, it immediately sends another request and repeats the operation. This effectively simulates the insert function of the server.

Challenges of long polling:

1. Ordering of the message cannot be guaranteed when multiple connections are established.
2. Message delivery cannot be guaranteed, which may lead to message loss.

The steps of long polling are:

1. The client sends the request.
2. If the data is not available, the server will wait for the computation. This causes a delayed response.
3. The client gets free once the response is obtained.

4. Socket Connection: A socket is an endpoint of a two-way connection link between two servers/nodes over the network. The socket is identified by a port number so that the TCP layer can identify the application to which the data will be sent. Each TCP connection can be uniquely identified by its two endpoints. Sockets are usually used for interaction between client and server. The plug has a typical flow of events. In the connection-oriented client-server model, the socket in the server process waits for a request from the client. To this end, the server first sets (binds) an address, which the client can use to find the server. It is used for applications that need a dedicated connection that remains open so that the data can flow in proper order securely and immediately from both sides.

5. Server-Sent Events(SSE): In the Server-Sent Events, the client subscribes to the server "stream", and the server will send a message ("stream of events") to the client until the server or client closes the stream. The server decides when and what to send to the client, for example, once the data changes. SSE is usually used to send continuous data streams or message updates to the browser client. In short, the server sending event is the time when the update is sent from the server (not pulled or requested) to the browser.

Polling	Socket Connection	Server-Sent Events
Client requests the Server	The socket on the server process waits for requests from a client.	The server sends the message to the Client
One way connection	Two-way connection	One way connectionS
Short-Lived Connection	Long-Lived Connection	Long-Lived Connection

Message-Based Communication

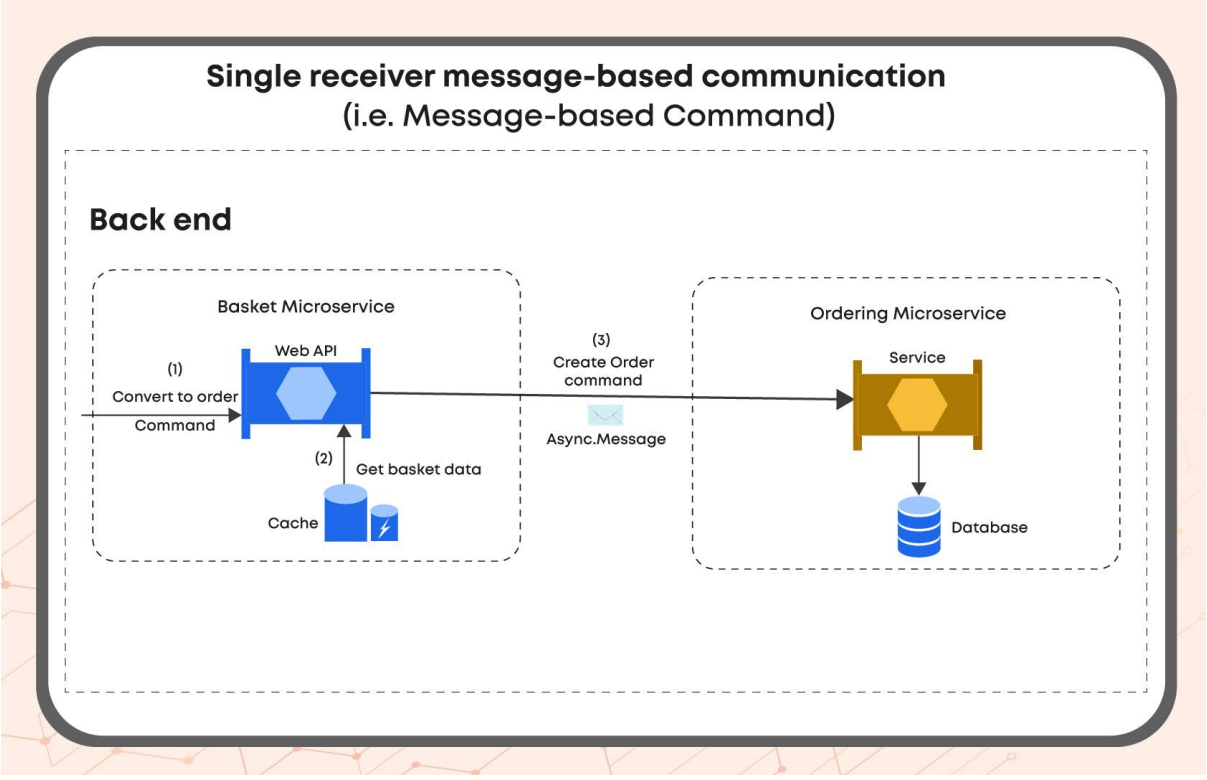
Message-Based Communication is an example of Asynchronous Communication.

Definition

Message-Based Communication is the exchange of information based on messages. The client makes a request in the form of a message to a service. The service replies back with the response in the form of a message. Since it is asynchronous communication, the client is not required to halt the process and wait for the process but has the freedom to start or stop any other process.

Terms used:

- Producer: The person sending the message.
- Consumer: The person receiving the message.
- Agent: The medium between producer and consumer responsible for successfully delivering the producer's message to the consumer is called the agent. The agent is generally a queue. A queue functions in the FIFO(First In First Out) Sequence or LILO(Last In Last Out) Sequence. A queue acts like a temporary data storage where the messages can be stored.



Models of Message-Based Communication System

1. P2P(Peer to Peer Messaging)
   The exchange of information between two peer computers takes place using a queue as an agent, and the request message is only targeted to a single consumer.
   Example: You send an email to your teacher to recheck your last test. The email is only targeted to your teacher and no one else. This is an example of P2P communication.

2. Publish-Subscribe Model
   The exchange of information between two peer computers takes place using a queue as an agent, and the request message is target to multiple consumers.
   Example: Tanuja has subscribed to a newsletter group. Every week, a newsletter is sent to all the members of the newsletter group and not only Tanuja, but Tanuja receives a copy of it. Similarly to the youtube channel subscriptions, a user can subscribe to a channel, and the single video would be broadcasted to multiple channel subscribers. The agent used for the exchange of information must be a queue in this model of messaging.

Industrial Tools for Asynchronous Communication:

- Kafka(Messaging Solution)
- RabbitMQ/ActiveMQ(Message-broker Software)
- Amazon Kinesis
- Akka
- Apache Spark

REST API

Framework Requirements of Communication

The framework to be used for effective communication must:

A. Enable communication over the network
B. Language neutral
C. Light
D. Fast

REST Framework is used to enable communication between two applications. REST Frameworks provides all the above-mentioned services.

REST stands for “representational state transfer”.

What is REST API?

REST API is an application programming interface(set of protocols used for building a web application) and it is a set of functions using which requests can be initiated and responses received and transfer data over the network. It is the way to communicate with other applications. The client-server communication model is based on HTTP protocol.

Example Use Case: Suppose you are doing a survey on the COVID health facilities and want to analyse the statistics of a government hospital in your area. You have built an application for analysis using machine learning which would input the statistics of Government hospitals. To transfer data between the web hospital management system and your application, a communication framework/interface is required, which would be the REST API. The hospital management system has to expose data, and the analysis application would fetch the data over the network. To fetch the data, the analysis application would need a data address(IP Address of the machine on which target data is present). This address of data is known as URI(Unique Resources Identifier). The application must know the operation to be used from the CRUD operations.
CRUD stands for Create, Read, Update and Delete. The operations are performed using HTTP calls.

Types of Requests:

A. GET -> Read data
B. POST -> Create/insert data
C. PUT -> Update data
D. DELETE-> Delete data

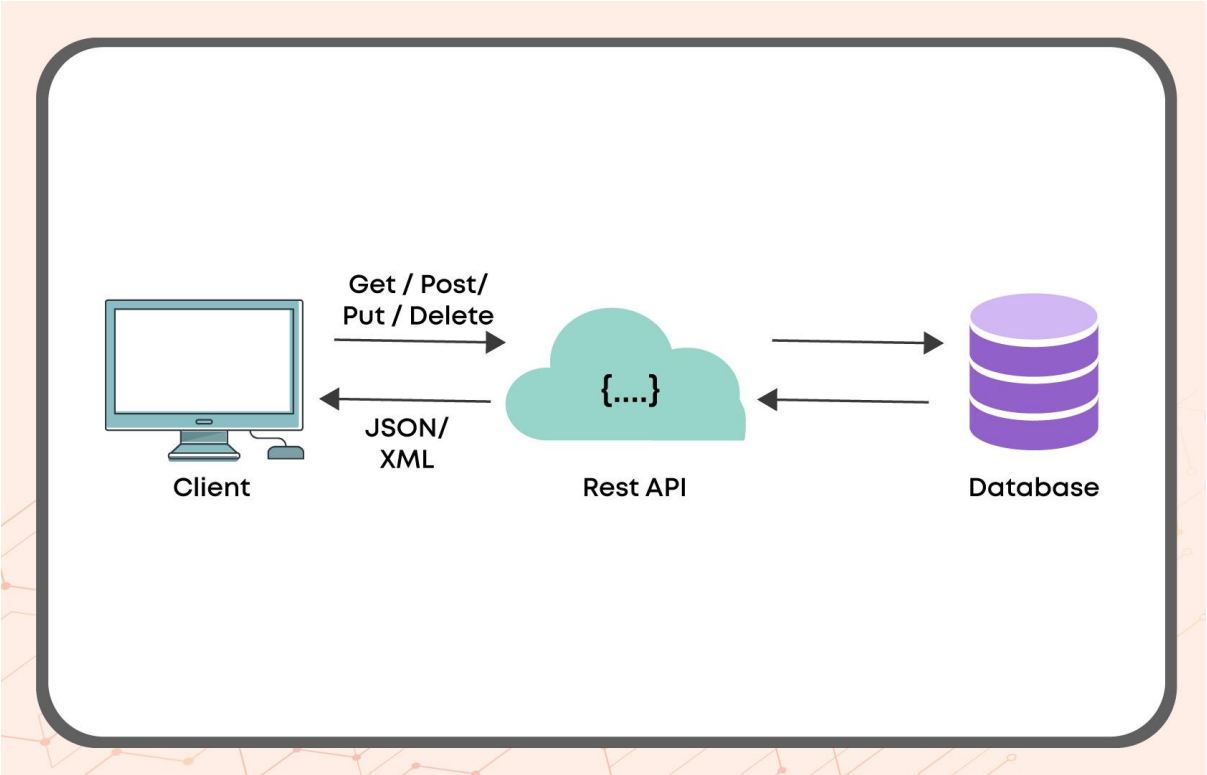


Fig: REST API

Advantages of using REST API

1. Simple: REST API is easier to adapt to the new syntax and build.
2. Resources: It uses fewer resources.
3. Reliability and Scalability: The client and server are separated, increasing scalability because the developers can scale the product better and less complex.

Drawbacks of REST API

1. REST typically relies on a few verbs (GET, POST, PUT, DELETE, and PATCH) which sometimes don't fit your use case.

For example, moving expired documents to the archive folder might not cleanly fit within these verbs.

2. Fetching complicated resources with nested hierarchies requires multiple round trips between the client and server to render single views, e.g. fetching content of a blog entry and the comments on that entry. For mobile applications operating in variable network conditions, these multiple roundtrips are highly undesirable.
3. Over time, more fields might be added to API response, and older clients will receive all new data fields, even those they do not need. As a result, it bloats the payload size and leads to larger latencies.

Time

What is time?

- Time is a measurable period in terms of events moving from past, present and future.
- We derive the idea of time from the order of occurrence of the event/process.
- It is used to describe the observation of the event.
- Time is one of the seven Fundamental Units.
- The SI unit of time is second.

How to measure time?

Time can be measured using a clock based on the atomic physics mechanism, gravity, spring mechanism etc.

Time and Architecture

Monolithic System:

- They are centralised units existing in a single location/machine.
- For mapping the time of occurrence of different events we can therefore use physical clocks.
- The use of logical time is not mandatory.

Distributed System:

- Since a distributed system is a spatial distribution of nodes and they can be present in different parts of the world.
- Multiple issues are encountered by the distributed systems while using physical clocks.
- Logical time should be used.

Issues encountered while using the physical clock to measure time in the case of Distributed Systems:

1. No concept of Global Physical Time.
2. Problem in finding the precise ordering of the events.
3. Network delays can change the original sequence of events.
4. Nodes in different geographical locations would follow different time zones.

Let’s learn more about logical time.

What is the logical time?

For a logical clock, two data structures are used to solve the problem of ordering and different time zones. It has two major concepts: Process marks its own event and the protocol updates after each local event.
The data structures used are separate for a logical global time and logical local time.

1. Logical Local time: Logical Local time carries information about the events of the system. Local time uses a Process that marks its own event and a protocol to update after each regional event.
2. Logical Global time: Logical Global time carries the local information about the global time. It stores the local knowledge about the global time and uses a protocol to update when processes exchange data.

Lamport Logical Clock

- It was developed by Leslie Lamport in 1978.
- It is a method of finding out the order of events in a distributed system.
- Because of the absence of a global time concept in a distributed system, Lamport logical clock is essentially required.
- It is used to provide the partial order of events and the numerical measure of what happened before the relationship.

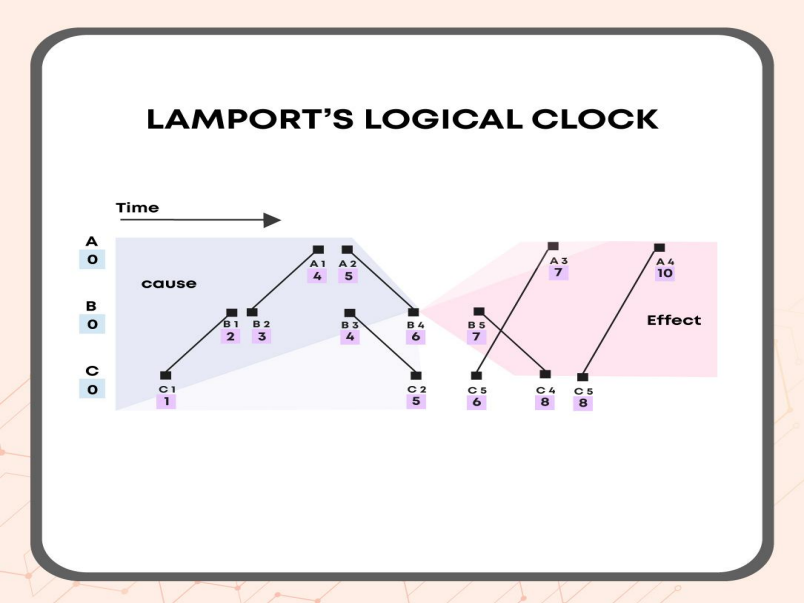


Fig: Lamport Logical Clock

Algorithm of Lamport Logical Clock:

1. Each process uses a local counter of the integer data type, which is initialised to zero.
2. Whenever an event(send or receive event) occurs, the process increments the counter.
3. The counter is assigned as the time stamp for each event.
4. Send event carries its own timestamp/counter.
5. For a receive event, the timestamp would equal the mathematical max of the local clock and message timestamp plus one.

Proxies

Ram is ill and decides not to go to college today. But it is a very important class and the attendance will be affecting the final grade. Ram thinks of an idea and asks his friend Shyam to mark Ram's attendance by signing against Ram's name in the attendance sheet. Shyam does the same. When the teacher asks the students to sign against their names to mark the attendance, Shyam signs against his and Ram's name according to the plan. This is an example of proxy and Shyam is acting as a proxy here. Because Shyam acted like Ram and signed on behalf of him.

Definition:

The proxy server provides a gateway between the user and the Internet. The server is called a "proxy" because it runs between the end-user and the websites you visit online. We use an IP address when your computer connects to the Internet.

A proxy server, also known as a "proxy" or "application-level gateway", acts as a local network (for example, all computers in a business or building) and a large network (for example, as the Internet.

Types of Proxy:

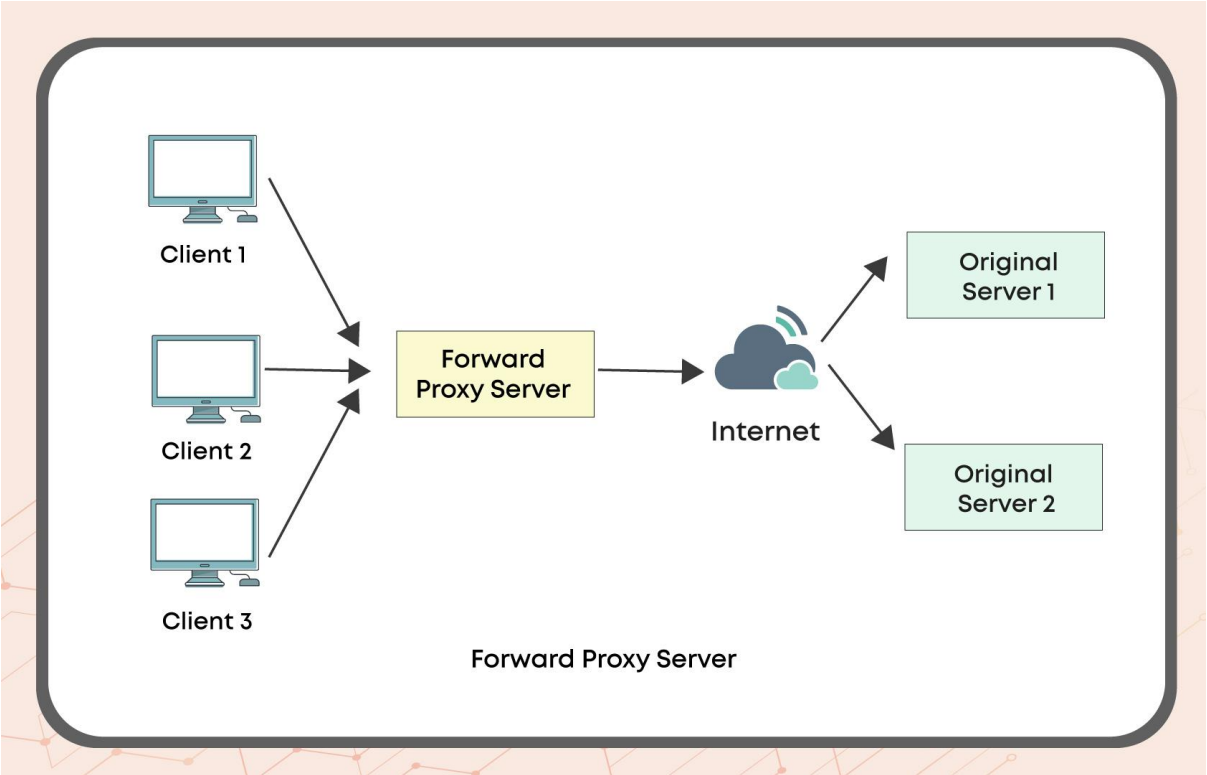
1. **Forward Proxy:** Forward proxy is the most common form of proxy server, which is generally used to pass requests from an isolated private network to the Internet through a firewall. Using a forward proxy, you can deny or allow requests from isolated networks or intranets to pass through the firewall.
   When one of these clients tries to connect to a file transfer server on the Internet, its request must first go through a forwarding proxy.

   According to the forwarding proxy settings, the request can be allowed or denied. If allowed, the request will be forwarded to the firewall and then to the file transfer server. From the perspective of the file transfer server, it is the proxy server that makes the request, not the client. Therefore, when the server responds, it directs its response to the proxy. But when the forwarding agent receives a response, it recognizes it as a response to a previous request. It then sends the response to the requesting client. Because the proxy server can track requests, responses, their origin and destination, different clients can send multiple requests to different servers through a forwarding proxy, and

the proxy will act as an intermediary for all these requests. Likewise, some requests will be allowed, while other requests will be denied.

Functions of forwarding proxy:

1. To provide protection to the client by filtering the outgoing requests and incoming responses
2. Enforcing “terms of use” of a network
3. Caching external site contents for better user experience



2. **Reverse Proxy:** Reverse proxies are servers located in front of web servers and forward client requests (such as web browsers) to these web servers. Reverse proxies are often used to help improve security, performance, and reliability.

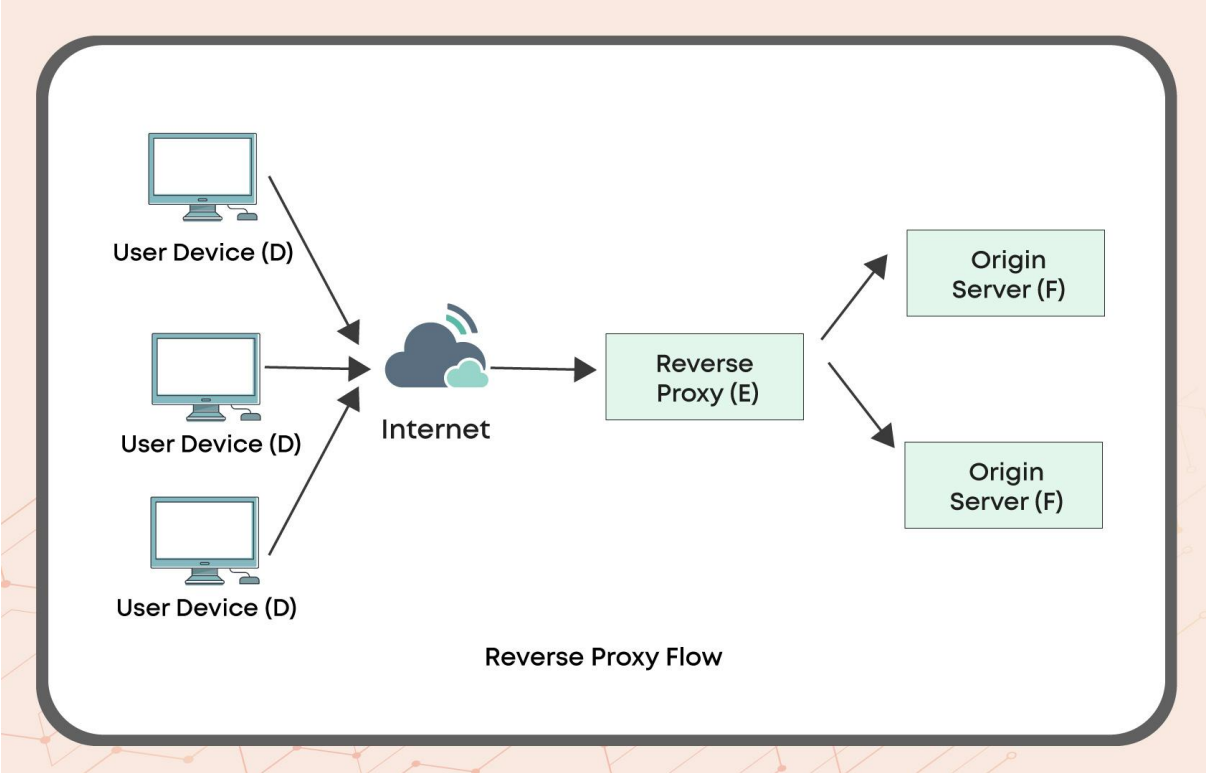
   The reverse proxy server is present before one or more web servers, intercepting requests from clients. This is different from the forward proxy, which is located in front of the client. With a reverse proxy, when a client sends a request to the origin server of the website, these requests will be intercepted by the reverse proxy server at the edge of the network. Then the

reverse proxy server will send the request and receive the response from the original server.

The difference between a direct proxy and a reverse proxy is subtle but important. A simplified way of summarizing is to say that the forwarding proxy is in front of the client and to ensure that no origin server communicates directly with that particular client. On the other hand, the reverse proxy is located in front of the actual server to make sure that there exists no communication of the client with the actual server.

Functions of Reverse Proxy:

1. Provide load balancing for the server
2. Prevention from malicious attacks such as Denial of Services
3. Allows administrators to swap the backend servers without disturbing the traffic



Applications of Reverse Proxy:

API Gateway:

API Gateway is an API management tool located between the client and the collection of back-end services. The API gateway is implemented using the reverse proxy to accept all application programming interface (API) calls, add the various

services required to implement them and return the appropriate results. For the clients, an API gateway can hide the details of how the application is divided into microservices.

Functions of API Gateway:

1. It can authenticate an incoming API call
2. API Gateway can function as a Load Balancer(equal distribution of load) and keep track of requests sent to different nodes of the network
3. API Gateway can also behave as a single point of entry for the external APIs

Web Application

What is a web application?

A Web Application provides dynamic data that can be read and written/manipulated on the web page. Web Application runs on a web server. They are simply the applications that run on the web.

In short,

Web Applications = Website + Additional Functionalities + Interactive Elements

Example:

In a web application like youtube, the user can stream videos, upload videos, and download content.

Web Applications: YouTube, Twitter, Google apps

Components of Web Applications:

1. Frontend/Visualisation Layer
2. Business/Backend Layer
3. Data Storage Layer

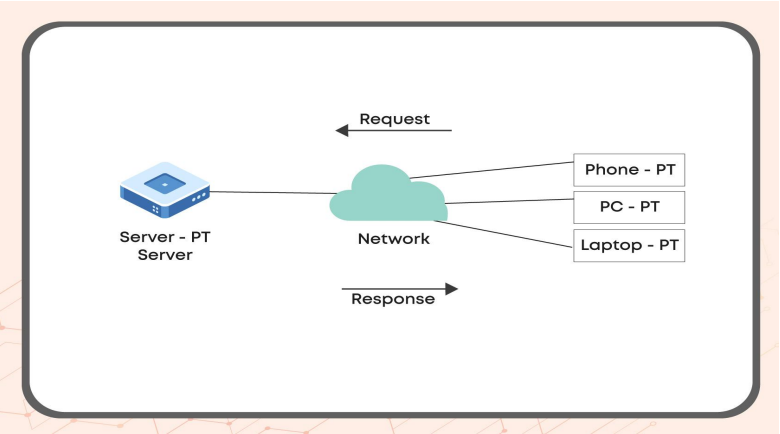
Client-Server Application:

Client-Server Applications are types of applications in which the process runs on the client system and requests information from the server system over the Network. Client-server architecture is the basic building block of the web and it is based on the Request-Response model.

Example: World Wide Web, Network Printing etc. In an e-commerce web application, the shopping cart system makes a request to the inventory to check on the availability of a certain product and then add it to the cart. The shopping cart system is the client and the inventory is the server here.

Client: System that initiates the communication over the Network

Server: System that receives the request over the Network



Advantages of Client-Server Model

1. Centralized System: The system is centralized since all the data is stored on the server.
2. Cost-Efficient: Cost efficient because the model doesn't require much maintenance and provides data-recovery methods.
3. Increased System Capacity: The model is dynamic since the roles are not fixed. Server can also behave as client in other situations and client can behave as server when required.
4. Secure: Since all the data is centrally stored, the architecture is comparatively secure than peer-to-peer architecture.

Disadvantages of Client-Server Model

1. Denial Of Service(DOS) Attacks: The client server model is prone to denial of service attacks. Denial of service attacks are cyber attacks when the system is made unavailable by disrupting service to the target users.
2. Overloading: In case if multiple clients simultaneously make a request to the same server, the server can be overloaded leading to failure and network congestion.
3. Network Failure: The client server model is based on request-response model over the network. During any network disruption or failure, the complete model can be affected.

Web Servers

What is a web server?

Web servers are tools or programs that help keep the web application always up and running and available to the end-user. The function of the web server is to display the website content. A web server is a machine that enables the delivery of websites. It's a computer application that manages web pages and distributes them when required. The web server's primary goal is to deliver content by storing, processing, and distributing web pages and delivering to the end-users. The Hypertext Transfer Protocol(HTTP) is used for this intercommunication.

It comprises three parts: a physical server, an operating system, and software that work together to speed up the HTTP (Hypertext Transfer Protocol) communication process.

Example: A web application has a frontend, backend and data storage layer. Each layer has a web server which simply is a process running on a machine. The web server displayed the content to the end-user by processing and delivering the web pages. All the layers are running under a web server which enables the different components of an application to be always available and operational.

Some standard and most used servers are Apache and Nginx.

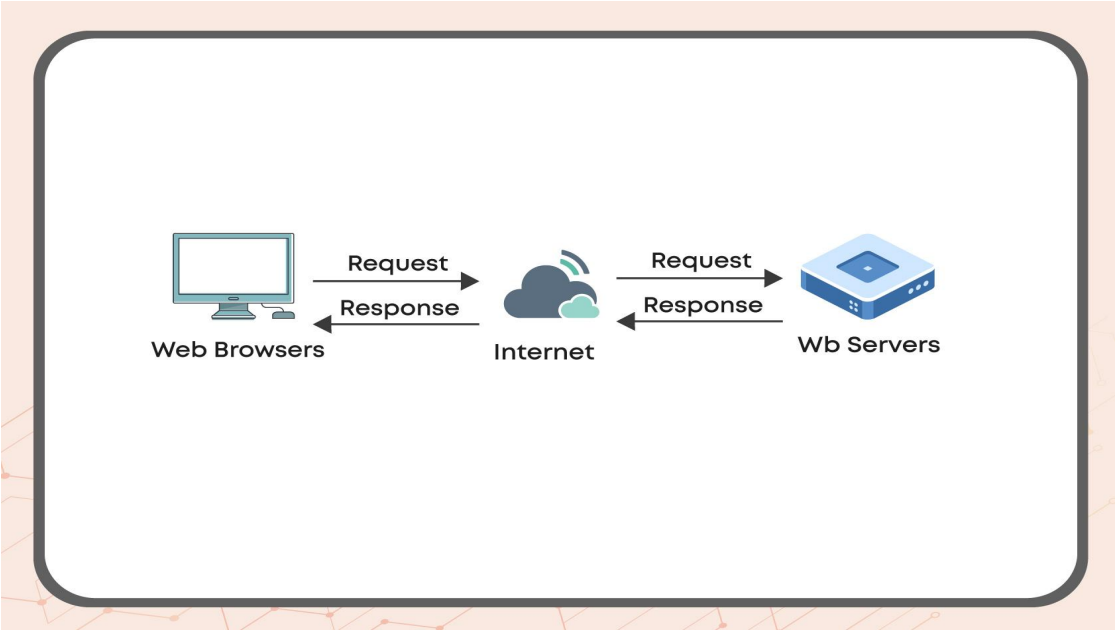


Fig: Web Server

Commonly Used Servers:

1. Apache Servers: Apache's job as a Web server is to accept directory (HTTP) requests from Internet users and provide the requested information in the form of files and Web pages. Much of the software and code on the Internet is built to work with Apache's features.

2. Tomcat Servers: Tomcat is used for Java web applications that do not require the entire Java EE requirements but still require a reliable tool. Because Tomcat serves as a Web server and a Servlet container, it is not considered a complete application server.

3. Node.js servers: HTTP is a built-in module in Node. Js that allows it to send data through the HyperText Transfer Protocol (HTTP). The HTTP module can create an HTTP server that listens for server ports and responds to requests.

Application Security

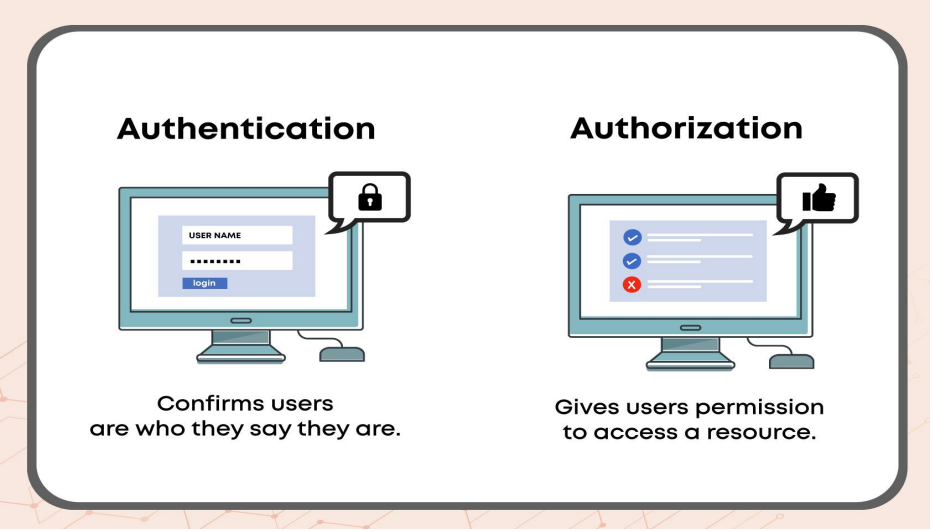
What is Application Security?

Application security is an application-level security measure that prevents the theft or hacking of application code and data. Application security can include hardware, software, and processes that identify or mitigate security vulnerabilities.

The main aspects of Application Security:

1. Authentication: Security Authentication is the process of confirming the user's credentials and identity. Operating systems typically identify/authenticate users using three methods: password, physical identification, and biometrics.
   Example: When we go to the airport, you are supposed to show some identity proof like an Adhar Card or Passport to enter inside the airport. This is an example of authentication.

2. Authorization: System security authentication is the process of giving a user access to a particular resource or feature. Authorization also signifies the system access control and user privileges in a secure environment, authorization should always follow certification.
   Example: Suppose you have written an essay in a google doc. You are the owner of the google doc but can grant access for editing, viewing or commenting as required. Simply, you can authorize other people to access the google doc. This is an authorisation.



Basic Authentication:

Basic authentication works by asking website visitors to enter their username and password. This method is widely used because it is supported by most browsers and web servers.

The advantages of basic authentication are:

1. Works through a proxy server.
2. Compatible with most internet browsers.
3. Allows users to access resources that are not on the IIS server.

Challenges of basic authentication are:

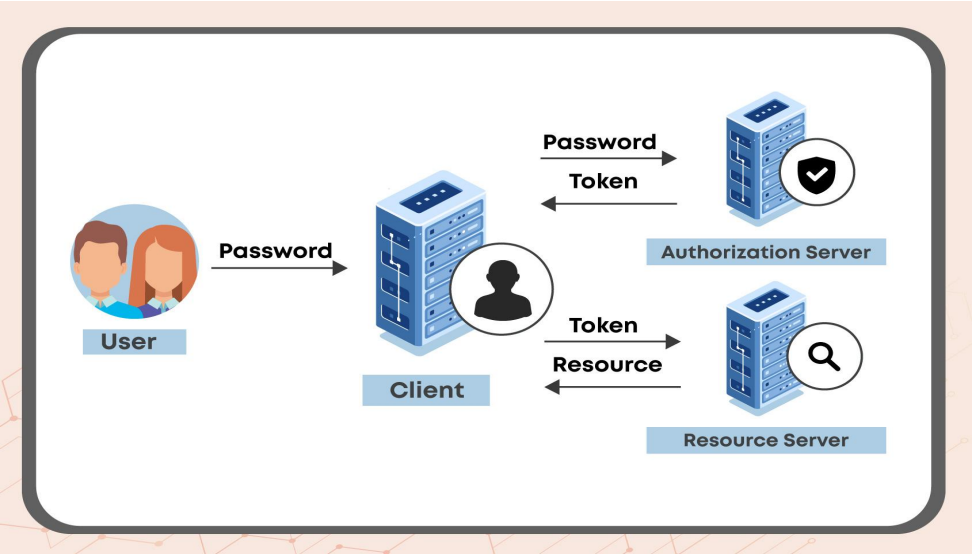
1. Information is sent over the network in cleartext. The information is encoded in base64 encoding but is sent in an unencoded format. Passwords sent with Basic authentication can be cracked easily.
2. By default, users must be able to log on locally to use Basic authentication.
3. Basic authentication is vulnerable to replay attacks.

Basic authentication does not encrypt user credentials, so traffic must always be sent over an encrypted SSL session. Users who authenticate with Basic Authentication must provide a valid username and password.

Token-Based Authentication/Access Token Based Authentication

Token-Based Authentication also known as Access token authentication is the process of validation and identity verification of the user. Other web authentication methods include biometrics and password authentication. Each authentication method is unique, but all methods fall into one of three categories: knowledge (what you know), inheritance (what you have), and possession (what you have).

Example: Bank Websites like ICICI Bank. Application expires if left idle for more time.

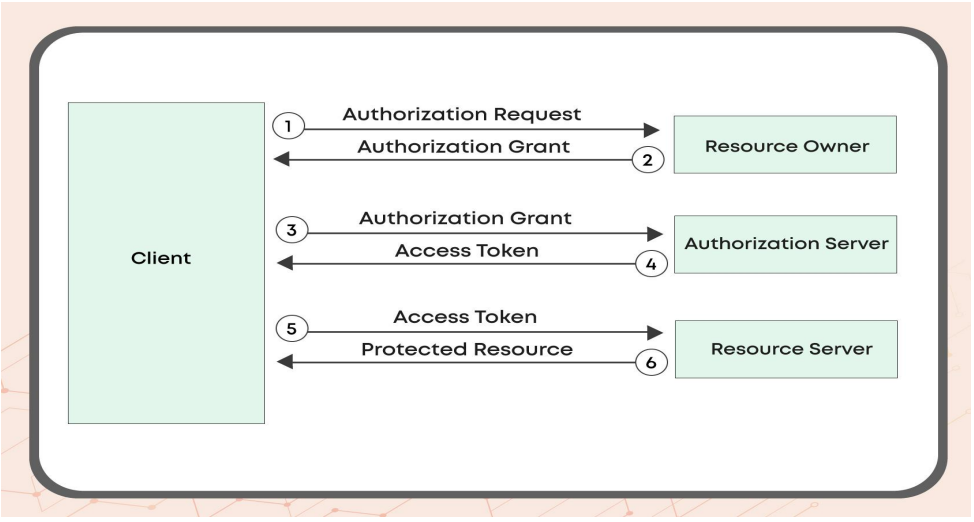


Password authentication falls under the category of Knowledge because users use previously created words and phrases to verify their identity. Biometrics is an example of "what you are" because it uses biological characteristics such as fingerprints. Finally, token-based authentication belongs in the owner category.

OAuth Authentication:

OAuth is an authentication protocol that allows the user to trust the application and interact with another without revealing the user's password. OAuth is an open standard authorization framework or protocol that provides "specified and secure access" functionality to your application.

For example, you can tell Facebook that ESPN.com can connect to your profile or post updates to your Timeline without having to specify your Facebook password on ESPN. This greatly reduces the risk. Your Facebook password will remain secure even if ESPN is compromised.



Tier Architecture

A web application would be designed according to the n-tier architecture where tiers are different layers of architecture. A tier is a logical separation between different components of the application. Tier architecture helps make modifications and updation of different components easy. It helps in assigning dedicated tasks and roles to each component.

For n=1, It is known as single-tier architecture/one-tier architecture
For n=2, It is known as a two-tier architecture
For n=2, It is known as a three-tier architecture
And so on.

Layers of a web application:

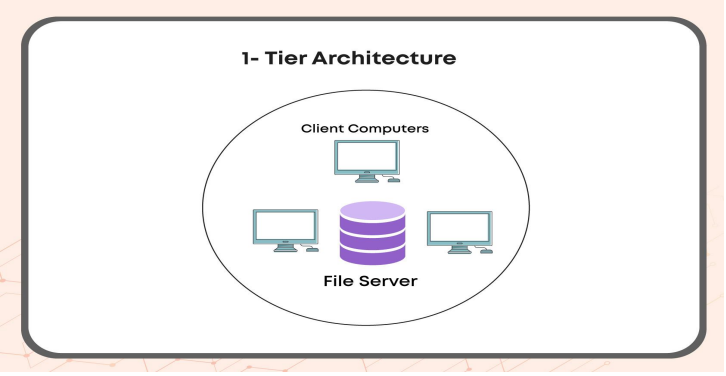
A. Visualisation Layer
B. Business Logic Layer
C. Data Storage Layer

The most common type of architecture is three-tier architecture.

One-Tier architecture:

One-tier architecture included storing all the software components and data on a single platform or server. One-tier architecture is when all the three layers of application including the visualisation layer, business logic layer and data storage layer are present in a single machine. It is similar to monolithic architecture. Because there are no network calls for the one-tier architecture, the network latency is minimum.

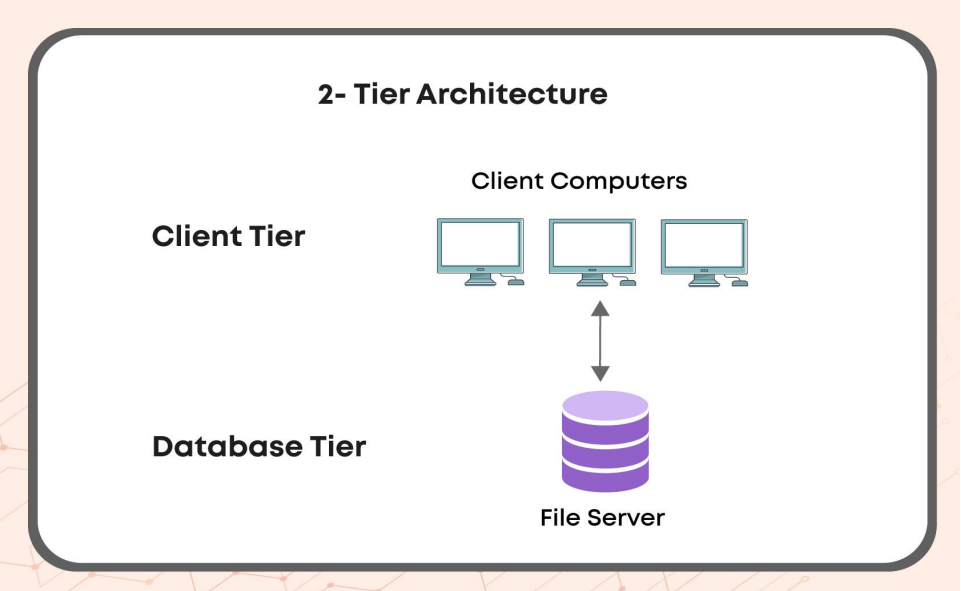
Example: Database in your local machine and accessing it using SQL queries.



Two-tier architecture:

The complete application is divided into two tiers. The first tier consists of the visualisation layer or the client tier. The second tier consists of the business logic layer and the data storage layer or the application/business logic tier.

Example: Client-Server architecture



Three-tier architecture:

It is the most common kind of architecture. The complete application is divided into three different tiers. The first tier is the frontend or client tier and consists of the visualisation layer. The second tier consists of the business logic layer involving processing and computation. The third tier is the data storage tier which includes reading and writing operations on the databases. Three-tier architecture takes less space at the client-side (Mobile apps, ATMs, website) as the logic and data are stored at separate servers. Hence, light apps will lead to more users willing to use the app.