System Design Bootcamp – 20 System Design Concepts Every Engineer Must Know

We all know that **System Design** is the core concept behind the design of any distributed system. Therefore every person in the tech industry needs to have at least a basic understanding of what goes behind designing a System. With this intent, we have brought to you the ultimate System Design Interview Bootcamp, a one-stop solution for learning System Design.

The most important stage in any development process, be it Software or any other tech, is **Design**. Without the designing phase, you cannot jump to the implementation of the testing part. The same is the case with the System as well.

Important Topics For The System Design Interview Bootcamp

- 1. System Design Fundamentals
- 2. Procedure to Design Systems
- 3. What is High-Level Design?
- 4. Storage options in System Design
- 5. Message Queues
- 6. Types of File Systems
- 7. System Design Patterns
- 8. Databases in System Design
- 9. What is Low-Level Design?
- 10. What are distributed systems?
- 11. Distributed System Failures
- 12. Distributed System Fundamentals
- 13. UML Diagrams for System Design
- 14. Scalable web applications
- 15. Caching
- 16. Essential Security Measures in System Design
- 17. Machine Learning and System Design
- 18. Containerization and System Design
- 19. The cloud and System Design
- 20. Interview Guide for System Design

1. System Design Fundamentals

Let us first begin the System Design Interview Bootcamp with the basics and fundamental terms and concepts used in System Design.

1.1 Functional and Non-Functional Requirements

Functional Requirements

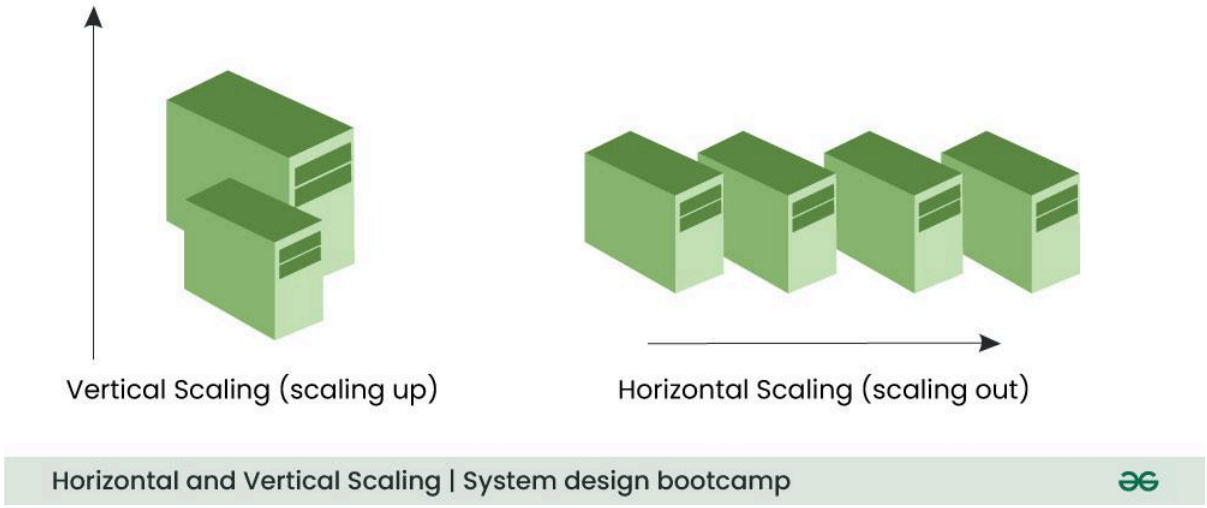
These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product

Non-functional requirements

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

Note: Functional vs non-Functional requirements

1.2 Horizontal and Vertical scaling

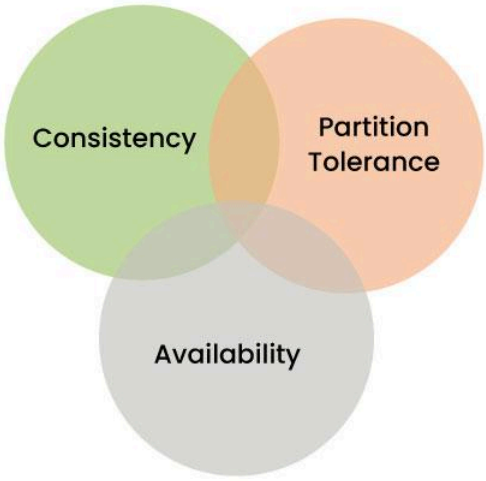


In horizontal scaling, we enhance the performance of the server by adding more machines to the network and sharing the processing and memory workload across multiple devices. We add more instances of the server to the existing pool of servers and distribute the load among these servers. In this approach, there is no need to change the server's capacity or replace the server. Also, like vertical scaling, there is no downtime while adding more servers to the network.

In simple terms upgrading the capacity of a single machine or moving to a new machine with more power is called vertical scaling. You can add more power to your machine by adding better processors, increasing RAM, or other power-increasing adjustments.

Vertical scaling can be easily achieved by switching from small to bigger machines but remember that this involves downtime.

1.3 CAP Theorem

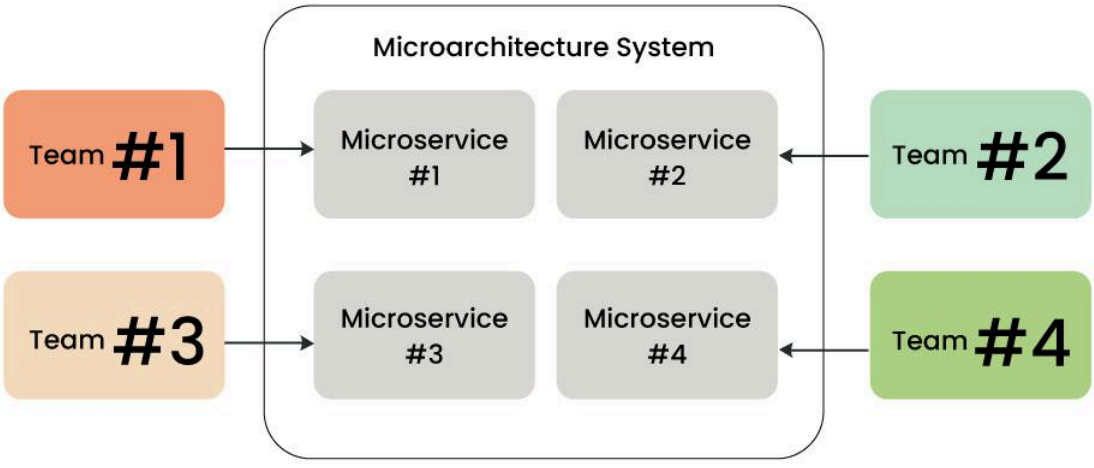


CAP theorem | System design bootcamp

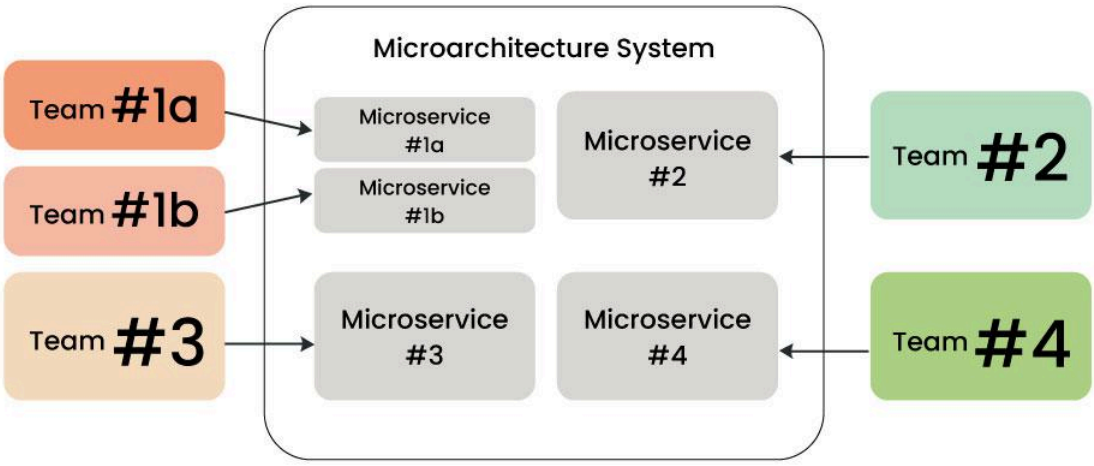
The three letters in CAP refer to three desirable properties of distributed systems with replicated data: **consistency** (among replicated copies), **availability** (of the system for read and write operations) and **partition tolerance** (in the face of the nodes in the system being partitioned by a network fault).

The CAP theorem states that it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication.

1.4 Microservices



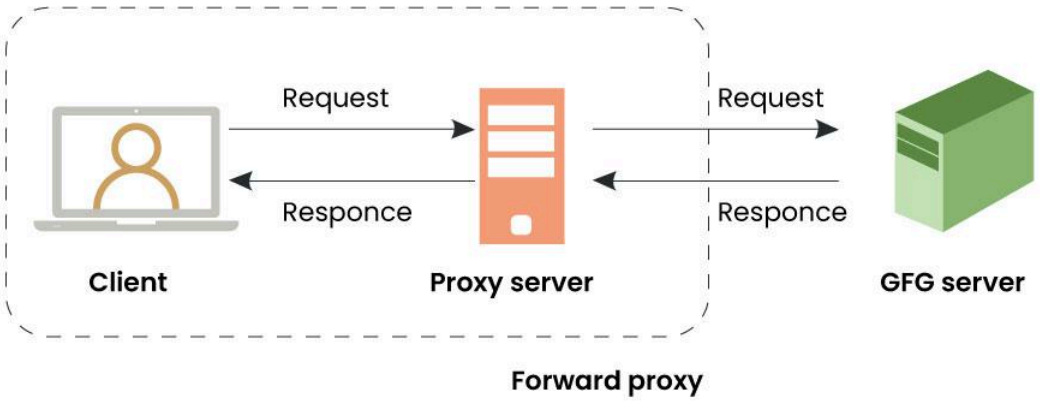
Microarchitecture System | System design bootcamp



Microarchitecture System | System design bootcamp

Microservice is a small, loosely coupled distributed service. It has evolved as a solution to the scalability, independently deployable, and innovation challenges with Monolithic architecture. It allows you to take a large application and decompose or break it into easily manageable small components with narrowly defined responsibilities. It is considered the building block of modern applications. Microservices can be written in a variety of **programming languages**, and **frameworks**, and each service acts as a mini-application on its own.

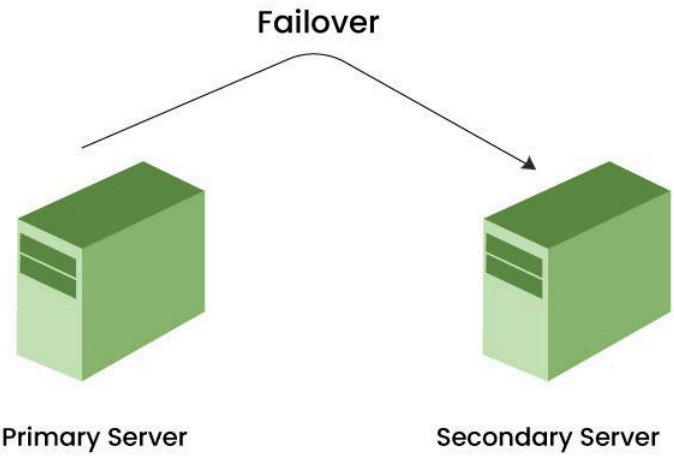
1.5 Proxy Servers



Proxy Server | System design bootcamp

Proxy servers act as intermediaries between client devices and servers. They improve performance by caching frequently requested content, provide security by filtering incoming traffic, and enable load balancing for efficient distribution of requests.

1.6 Redundancy and Replication



Redundancy and Replication | System design bootcamp

Redundancy in system design is the intentional inclusion of extra components, systems, or resources to ensure continued functionality in the event of a failure. Redundancy aims to eliminate single points of failure, enhancing system reliability and fault tolerance. This can be achieved through techniques such as duplicating critical hardware components or having backup systems ready to take over seamlessly.

Data Replication | System design bootcamp

Replication, on the other hand, involves creating and maintaining copies of data or entire systems across multiple locations. The primary goal of replication is to improve data availability, distribute the load, and enhance fault tolerance. In distributed databases, data replication ensures that if one server or node fails, another can take over with the same dataset. This redundancy of data contributes to resilience, reducing the risk of data loss and improving overall system performance.

2. Procedure to Design Systems

System design is the process of designing the architecture and components of a software system to meet specific business requirements. The process involves defining the system's architecture, components, modules, and interfaces, and identifying the technologies and tools that will be used to implement the system. Here are some steps to get started with system design:

- **Understand the requirements:** Before you begin designing the system, you need to understand the requirements. This involves talking to stakeholders and users, reviewing existing documentation, and analyzing the business processes that the system will support.
- **Define the system architecture:** Once you have a clear understanding of the requirements, you can begin defining the system architecture. This involves identifying the major components of the system and the interfaces between them.
- **Choose the technology stack:** Based on the requirements and the system architecture, you can select the technology stack. This includes choosing the programming language, database, frameworks, and libraries that will be used to implement the system.
- **Design the modules:** Next, you need to design the modules that will make up the system. This involves defining the functions that each module will perform and the data that it will manipulate.
- **Plan for scalability:** As you design the system, you need to consider how it will scale. This involves identifying potential bottlenecks and designing the system to handle increased loads.
- **Consider security and privacy:** Security and privacy should be a key consideration in system design, as mentioned later in this System Design Interview Bootcamp. This involves identifying potential security threats and designing the system to mitigate them.
- **Test and validate:** Once the system design is complete, you need to test and validate it. This involves creating test cases and scenarios that simulate real-world usage and verifying that the system meets the requirements.

3. What is High-Level Design?

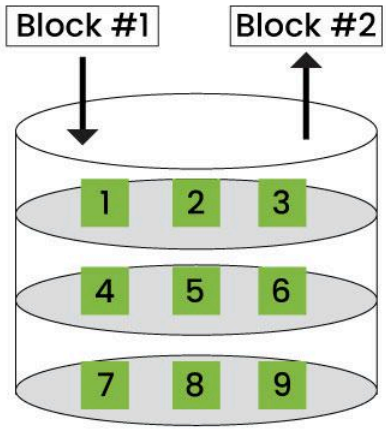
High-level design or **HLD** refers to the overall system, a design that consists description of the system architecture and design and is a generic system design that includes:

- System architecture
- Database design
- Brief description of systems, services, platforms, and relationships among modules.

> **High-level design** or **HLD** is also known as **macro level designing**.

4. Storage options in System Design

4.1 Block Storage



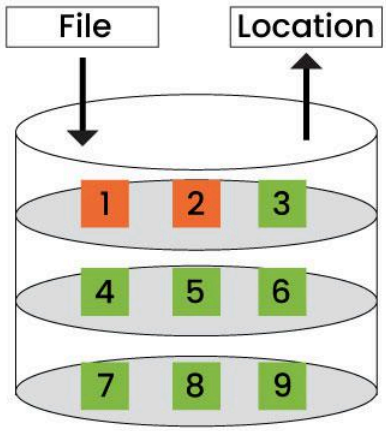
Fixed-size blocks in specific locations

Block Storage | System design bootcamp

Block storage involves dividing data into fixed-sized blocks and storing them on block devices such as hard drives or solid-state drives (SSDs). These blocks are accessed using low-level block-level protocols, typically through storage area networks (SANs) or direct-attached storage (DAS).

4.2 File Storage

File Storage is very important in every System Design. So it is very important in this System Design Interview Bootcamp to learn about it in detail.

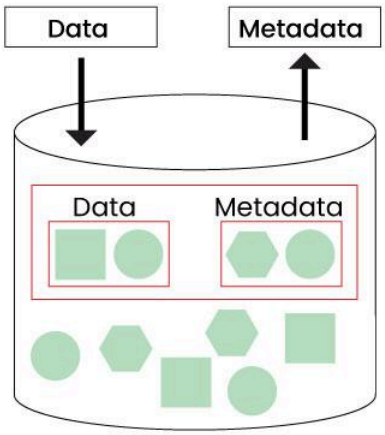


Specific folders in a fixed, logical order

File Storage | System design bootcamp

File storage stores data as files and presents it to its final users as a hierarchical directories structure. It is typically accessed using file-level protocols like Network File System (NFS) or Server Message Block (SMB). File storage can be implemented using network-attached storage (NAS) devices or distributed file systems.

4.3 Object Storage

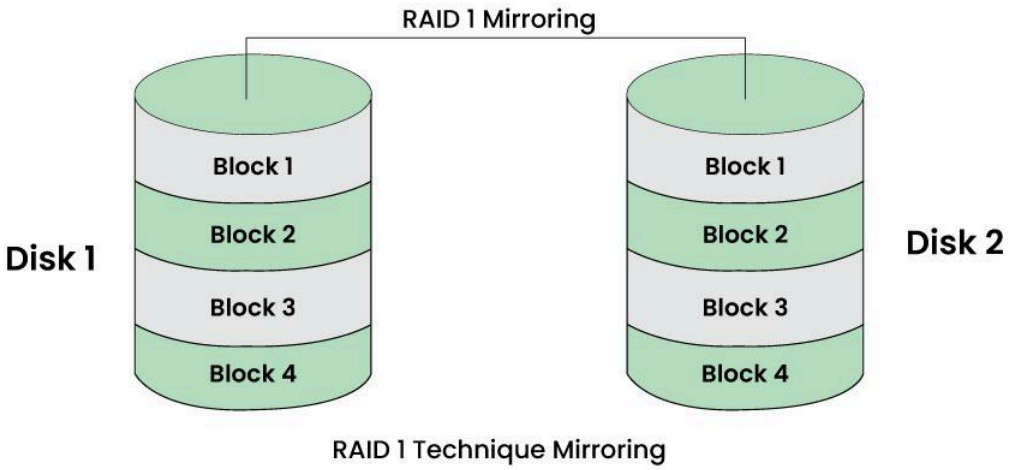


Scalable storage driven by metadata

Object Storage | System design bootcamp

Files are divided into little parts and dispersed over hardware in a flat structure. Instead of being maintained as files in directories or as blocks on servers, the data is divided up into discrete parts called objects and kept in a single repository with object storage. Object storage systems typically use a RESTful API for accessing and managing data.

4.4 Redundant Disk Arrays (RAID)



RAID 1 Technique Mirroring

RAID | System design bootcamp

RAID combines multiple physical disk drives into a single logical unit to improve performance, reliability, or a combination of both. RAID is very transparent to the underlying system. This means, that to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes to the existing code.

5. Message Queues

Message queues facilitate communication between distributed systems by allowing asynchronous communication. This decouples the components, enabling them to operate independently and improving system reliability. It is a form of communication and data transfer mechanism used in computer science and system design. It functions as a temporary storage and routing system for messages exchanged between different components, applications, or systems within a larger software architecture.

5. 1 Kafka Message Queue

Kafka is a distributed streaming platform that excels in handling real-time data streams. It is used for building real-time data pipelines and streaming applications.

6. Types of File Systems

6.1 Google File System(GFS)

Google Inc. developed the Google File System (GFS), a scalable distributed file system (DFS), to meet the company’s growing data processing needs. GFS offers fault tolerance, dependability, scalability, availability, and performance to big networks and connected nodes. GFS is made up of a number of storage systems constructed from inexpensive commodity hardware parts. The search engine, which creates enormous volumes of data that must be kept, is only one example of how it is customized to meet Google’s various data use and storage requirements.

6.2 Hadoop Distributed File System(HDFS)

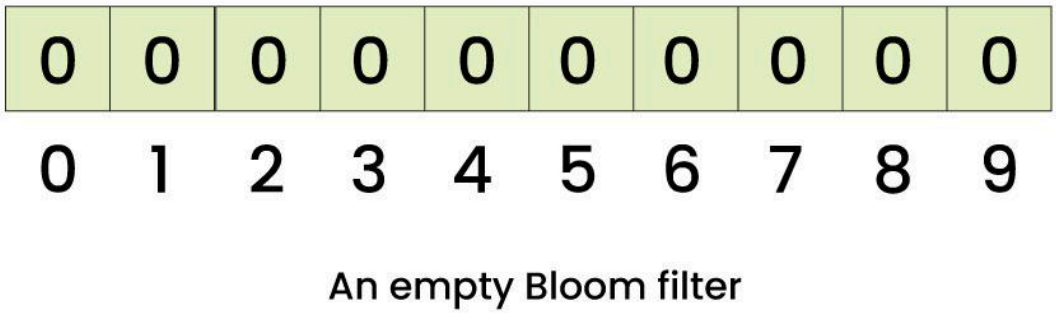
With growing data velocity the data size easily outgrows the storage limit of a machine. A solution would be to store the data across a network of machines. Such filesystems are called distributed filesystems. Since data is stored across a network all the complications of a network come in.

This is where Hadoop comes in. It provides one of the most reliable filesystems. HDFS (Hadoop Distributed File System) is a unique design that provides storage for extremely large files with streaming data access pattern and it runs on commodity hardware.

7. Design Patterns in System Design Interview Bootcamp

Design patterns are used to represent some of the best practices adapted by experienced object-oriented software developers. A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences.

7.1 Bloom Filters

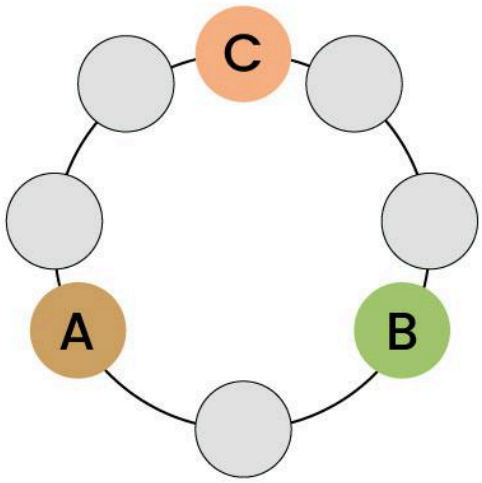


An empty Bloom filter

Bloom Filter | System design bootcamp

It is a space-efficient probabilistic data structure used to test whether a given element is a member of a set. It achieves this by using multiple hash functions to map elements to a bit array. While false positives are possible, false negatives are not. This makes Bloom filters valuable in scenarios where memory is constrained, and a slight risk of false positives is acceptable, such as in caching systems and network routing tables.

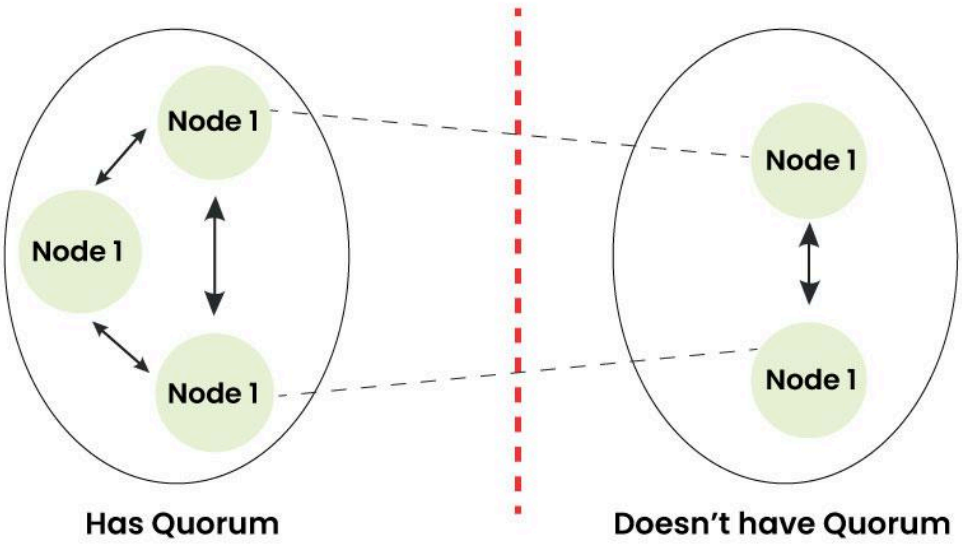
7.2 Consistent Hashing



Consistent Hashing | System design bootcamp

It is a technique used in distributed systems to efficiently distribute data across a changing set of nodes. Unlike traditional hash functions, consistent hashing minimizes the impact of adding or removing nodes, ensuring that most keys remain mapped to the same nodes. This is particularly useful in scenarios like distributed caching and load balancing, where maintaining a stable mapping despite node changes is essential for performance and data integrity.

7.3 Quorum

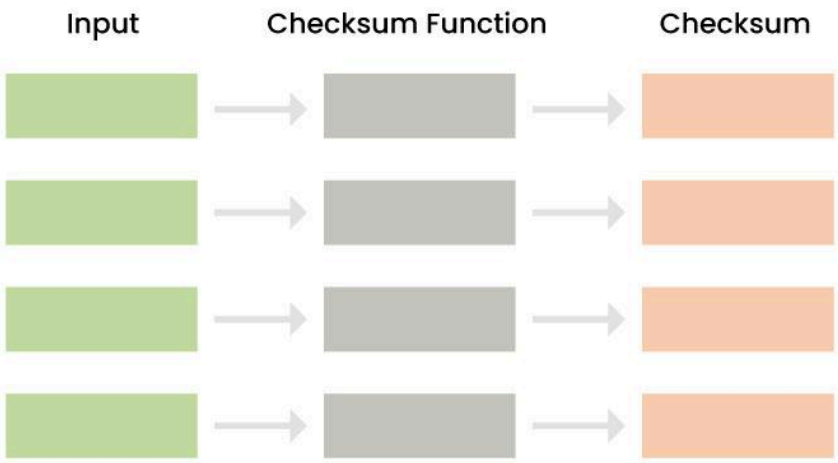


Quorum | System design bootcamp

In distributed systems, a quorum is a strategy to achieve consensus among a majority of nodes. It helps in ensuring that a certain number of nodes must agree on an operation for it to be considered successful. Quorums are crucial for maintaining data consistency and

availability, especially in scenarios prone to network partitions. Variations like the “two-thirds” or “majority” quorum systems are common in databases, providing a balance between fault tolerance and system responsiveness.

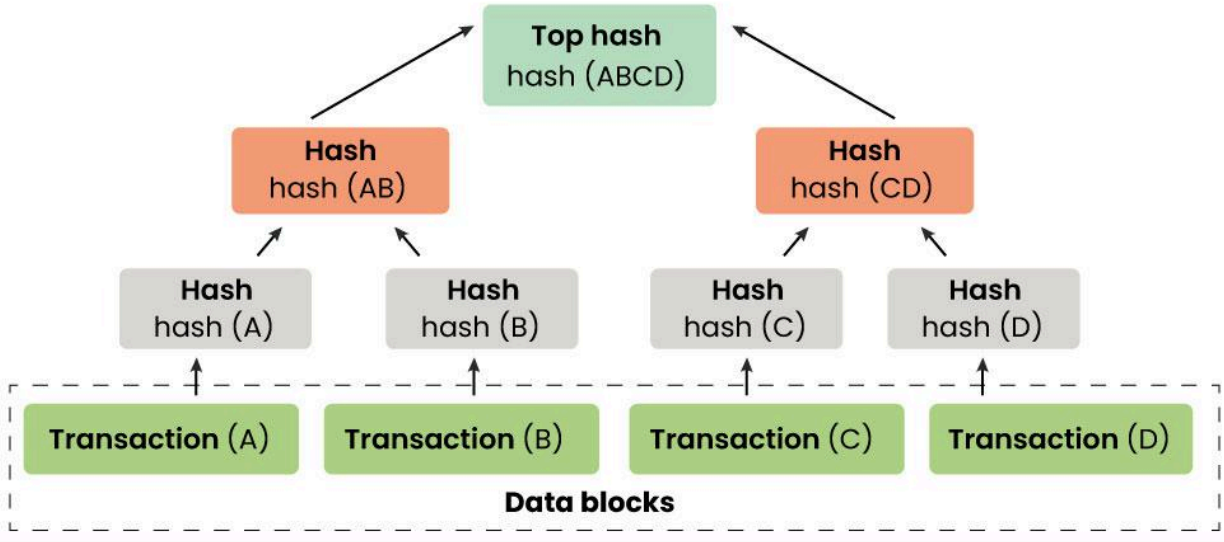
7.4 Checksum



CheckSum | System design bootcamp

A checksum is a value derived from the data in a file or message, used to verify its integrity. Various algorithms, like CRC or Adler-32, calculate checksums, and if the checksum of the received data matches the calculated checksum, it indicates that the data is likely intact. Checksums are widely used in data transmission and storage to detect errors, ensuring data reliability and preventing the propagation of corrupt information.

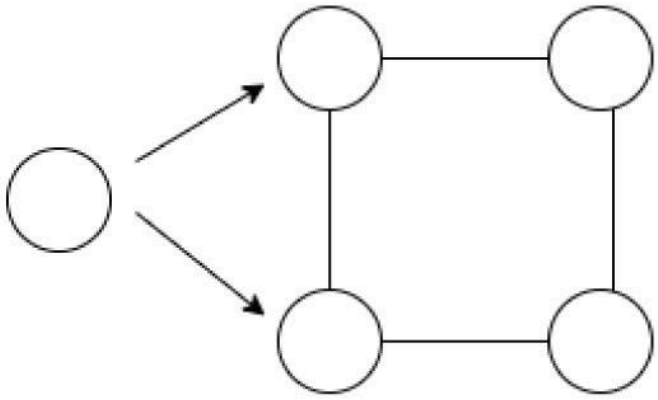
7.5 Merkle Trees



Merkel Tree | System design bootcamp

It is a hierarchical data structure that facilitates efficient verification of large datasets. It works by recursively hashing pairs of data until a single hash, known as the Merkle root, is obtained. If any part of the data changes, it only affects the path from the altered data to the root, simplifying verification. Merkle trees are commonly used in distributed systems and cryptocurrencies to ensure data consistency and integrity without transmitting the entire dataset.

7.6 Leader Election



Leader Election | System design bootcamp

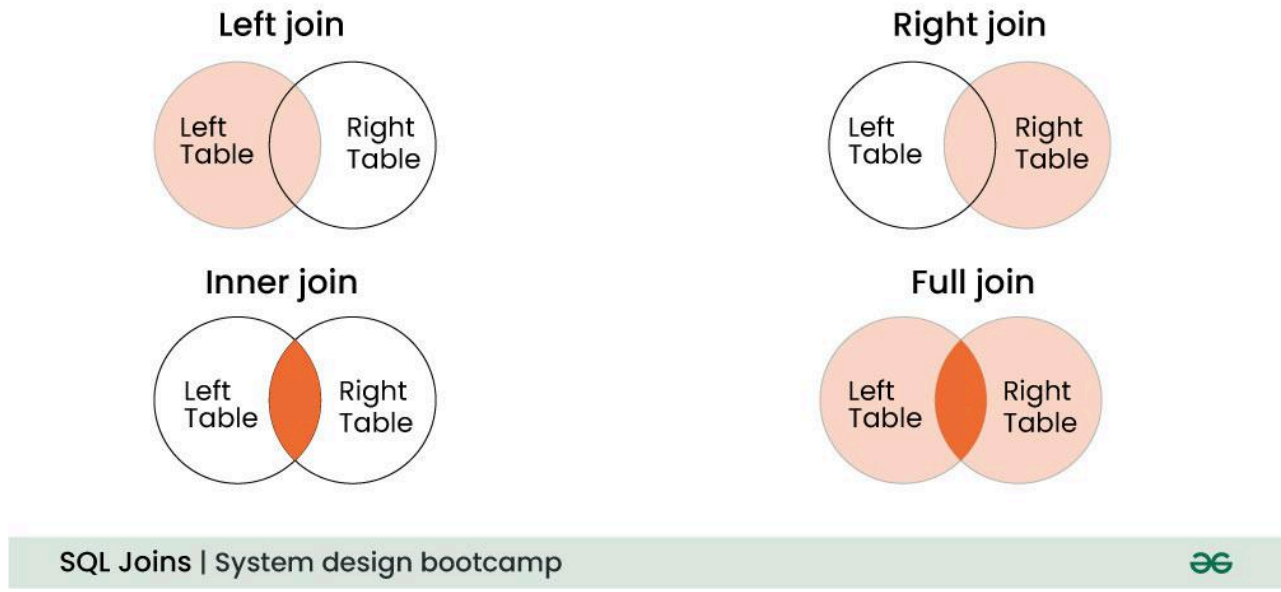
It is a crucial concept in distributed systems where nodes must select a leader to coordinate and manage the distributed activities. Algorithms like Paxos and Raft are often employed for leader election, ensuring that one node takes charge while others follow. This leadership structure simplifies decision-making, improves coordination, and enhances the efficiency of distributed systems.

8. Databases in System Design

8.1 Relational databases

- **MySQL:** It is an open-source Relational Database Management System that stores data in a structured format using rows and columns. MYSQL language is easy to use as compared to other programming language like C, C++, Java, etc. By learning some basic commands we can work, create and interact with the Database.
- **PostgreSQL:** PostgreSQL is an advanced and open-source relational database system and is used as a database for many web applications, mobile and analytics applications. It supports both SQL (relational) and JSON (non-relational) querying

- **SQL Joins** are operations used to combine rows from two or more tables based on a related column between them. Common types of joins include:
    - **Inner Join:** Returns rows when there is a match in both tables.
    - **Left (Outer) Join:** Returns all rows from the left table and matching rows from the right table.
    - **Right (Outer) Join:** Returns all rows from the right table and matching rows from the left table.
    - **Full (Outer) Join:** Returns all rows when there is a match in either table.
    - **Cross Join:** Returns the Cartesian product of rows from both tables (all possible combinations).
    - **Self Join:** Joins a table with itself based on a related column.



8.2 Non-relational databases

MongoDB: The most popular NoSQL database, is an open-source document-oriented database. The term ‘NoSQL’ means ‘non-relational’. It means that MongoDB isn’t based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON

8.3 How to choose a database?

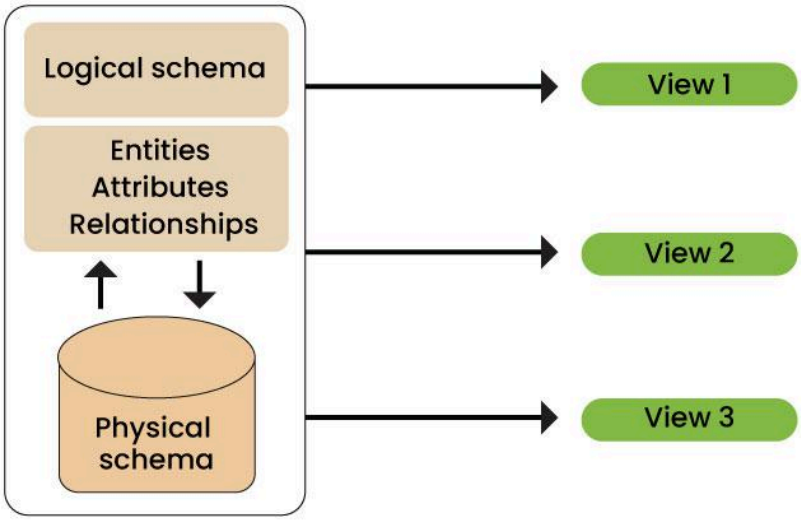
There are so many databases are available and picking up one database over another is a complicated decision. Well, there is no real formula you can follow but there are a few things you should think about. irstly set aside the idea that you are going to find the one true database that is better than everything else. Now ask a few important questions related to your project:

- How much data do you expect to store when the application is mature?
- How many users do you expect to handle simultaneously at peak load?

- What availability, scalability, latency, throughput, and data consistency does your application need?
- How often will your database schemas change?
- What is the geographic distribution of your user population?
- What is the natural “shape” of your data?
- Does your application need online transaction processing (OLTP), analytic queries (OLAP), or both?
- What ratio of reads to writes do you expect in production?
- What are your preferred programming languages?
- Do you have a budget? If so, will it cover licenses and support contracts?
- How strict are you with invalid data being sent to your database? (Ideally, you are very strict and do server-side data validation before persisting it to your database)

Note: Also check SQL vs NoSQL Database

8.4 Database Schemas



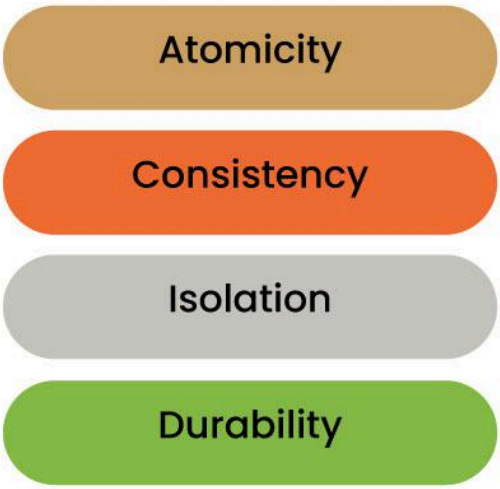
Database Schemas | System design bootcamp

A database schema is a **logical representation of data** that shows how the data in a database should be stored logically. It shows how the data is organized and the relationship between the tables. Database schema contains table, field, views and relation between different keys like primary key, foreign key.

Data is stored in the form of files which is unstructured in nature which makes accessing the data difficult. Thus to resolve the issue the data are organized in structured way with the help of database schema.

Database Queries: Queries are used to retrieve and manipulate data from databases using SQL or other query languages.

8.5 ACID Properties

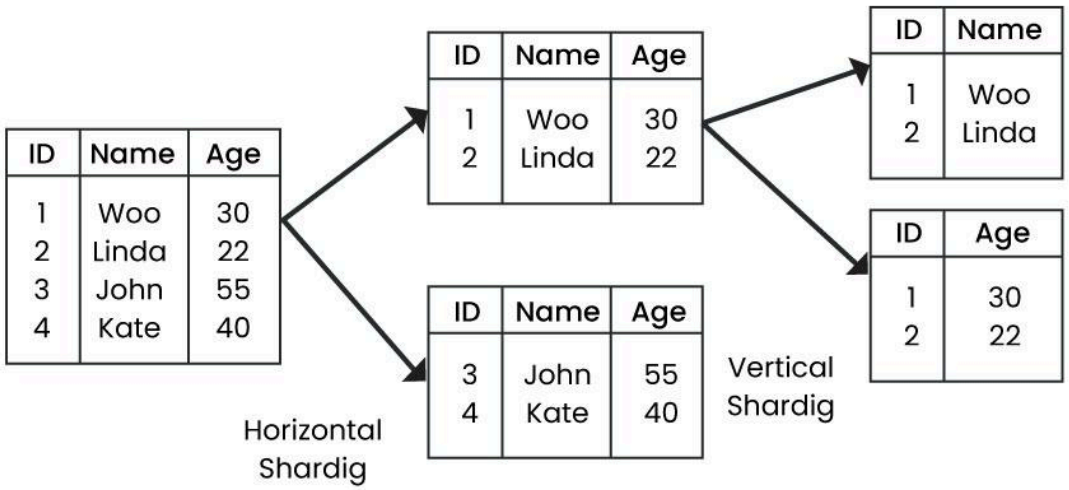


ACID Properties | System design bootcamp

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

- **Atomicity:** By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially.
- **Consistency:** This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.
- **Isolation:** This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference.
- **Durability:** This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.

8.6 Sharding and Partitioning



Database Sharding and Partitioning | System design bootcamp

Sharding

Sharding represents a technique used to enhance the scalability and performance of database management for handling large amounts of data. This approach involves fragmenting the extensive dataset into smaller, self-contained segments known as **shards**. These shards are then allocated to separate servers or nodes, facilitating parallelism in data processing. As a result, query response times are improved, high traffic loads can be accommodated, and bottlenecks are mitigated.

Partitioning

Partitioning is an optimization technique in databases where a single table is divided into smaller segments called partitions. These partitions hold subsets of the table's data based on specific criteria like value ranges or categories. This strategy enhances query performance by reducing the amount of scanned data, resulting in faster retrieval times. Furthermore, partitioning simplifies maintenance tasks such as backup and indexing since they can be focused on individual partitions.

8.7 Database Indexing

Indexing improves database performance by minimizing the number of disc visits required to fulfill a query. It is a data structure technique used to locate and quickly access data in databases. Several database fields are used to generate indexes. The main key or candidate key of the table is duplicated in the first column, which is the Search key.

To speed up data retrieval, the values are also kept in sorted order. It should be highlighted that sorting the data is not required. The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk

block where that particular key value can be found.

9. What is Low-Level Design?

> **LLD,** as the name suggests, stands for **low-level design**. It is a component-level design process that follows step by step refinement process. The input to LLD is HLD.

LLD describes class diagrams with the help of methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document. It provides us with the structure and behavior of class as different entities have different character sets. From this design, it is easy for a developer to write down logic and henceforth the actual code for it.

10. What are distributed systems?

Distributed System is a collection of autonomous computer systems that are physically separated but are connected by a centralized computer network that is equipped with distributed system software. The autonomous computers will communicate among each system by sharing resources and files and performing the tasks assigned to them.

10.1 Characteristics of Distributed System

- **Resource Sharing:** It is the ability to use any Hardware, Software, or Data anywhere in the System.
- **Openness:** It is concerned with Extensions and improvements in the system
- **Concurrency:** It is naturally present in Distributed Systems, that deal with the same activity or functionality that can be performed by separate users who are in remote locations. Every local system has its independent Operating Systems and Resources.
- **Scalability:** It increases the scale of the system as a number of processors communicate with more users by accommodating to improve the responsiveness of the system.
- **Fault tolerance:** It cares about the reliability of the system if there is a failure in Hardware or Software, the system continues to operate properly without degrading the performance the system.
- **Transparency:** It hides the complexity of the Distributed Systems to the Users and Application programs as there should be privacy in every system.
- **Heterogeneity:** Networks, computer hardware, operating systems, programming languages, and developer implementations can all vary and differ among dispersed system components.

11. Distributed System Failures

- **Method failure:** In this type of failure, the distributed system is generally halted and unable to perform the execution. Sometimes it leads to ending up the execution resulting in an associate incorrect outcome. Method failure causes the system state to deviate from specifications, and also method might fail to progress.
- **System failure:** In system failure, the processor associated with the distributed system fails to perform the execution. This is caused by computer code errors and hardware issues. Hardware issues may involve CPU/memory/bus failure. This is assumed that whenever the system stops its execution due to some fault then the interior state is lost.
- **Secondary storage device failure:** A storage device failure is claimed to have occurred once the keep information can't be accessed. This failure is sometimes caused by parity error, head crash, or dirt particles settled on the medium.
- **Communication medium failure:** A communication medium failure happens once a web site cannot communicate with another operational site within the network. it's typically caused by the failure of the shift nodes and/or the links of the human activity system.

11.1 Failure Models

- **Timing failure:** Timing failure occurs when a node in a system correctly sends a response, but the response arrives earlier or later than anticipated. Timing failures, also known as performance failures, occur when a node delivers a response that is either earlier or later than anticipated.
- **Response failure**: When a server's response is flawed, a response failure occurs. The response's value could be off or transmitted using the inappropriate control flow.
- **Omission failure:** A timing issue known as an "infinite late" or omission failure occurs when the node's answer never appears to have been sent.
- **Crash failure:** If a node encounters an omission failure once and then totally stops responding and goes unresponsive, this is known as a crash failure.
- **Arbitrary failure :** A server may produce arbitrary response at arbitrary times.

12. Distributed System Fundamentals

12.1 MapReduce

Map Reduce | System design bootcamp

MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages with various different-different optimizations.

The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster network and to reduce the processing power. The MapReduce task is mainly divided into two phases Map Phase and Reduce Phase.

12.2 Stateless and Stateful systems

12.2.1 Stateless Systems

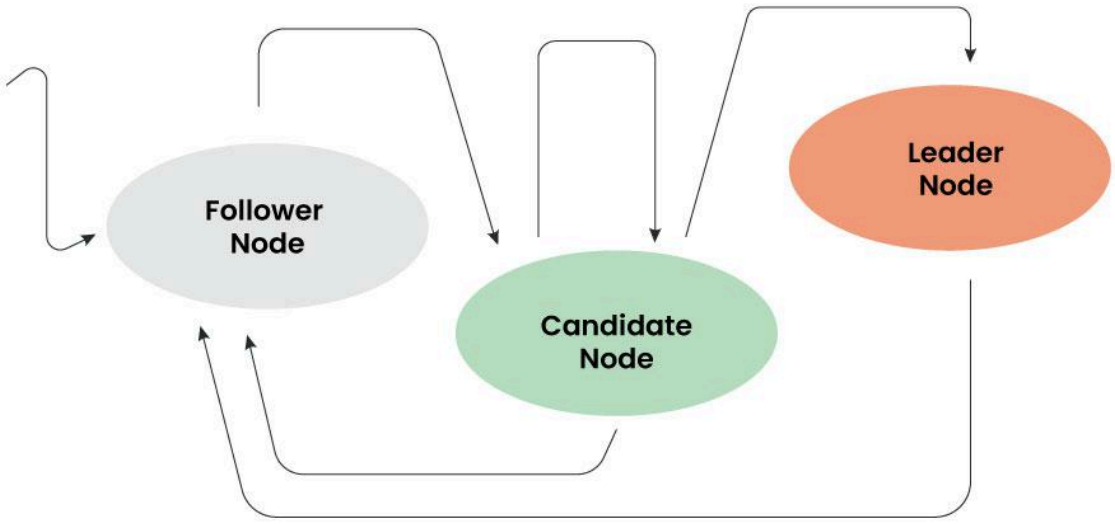
In a stateless system, each request from a client to a server is independent and self-contained. The server does not retain any information about the client's previous requests or state. Every request is treated as new, and the server doesn't store any data or context about the client between requests.

This design philosophy simplifies scalability since any server can handle any request from the client. Stateless systems are often more fault-tolerant and easier to scale horizontally because each request is isolated, and there are no dependencies on previous interactions.

12.2.2 Stateful systems

In a stateful system, the server keeps track of the state or context of the client across multiple requests. The server retains information about the client's interactions, allowing for a continuous and personalized experience. This is particularly useful for applications that involve sessions, user authentication, or complex workflows where the server needs to remember past actions or maintain a certain state. Stateful systems offer advantages in terms of convenience and efficiency.

12.3 Raft



Raft | System design bootcamp

It is a consensus algorithm designed to manage a replicated log in a distributed system. Developed by Diego Ongaro and John Ousterhout, Raft aims to provide a straightforward yet efficient approach to distributed consensus. The primary use case for Raft is ensuring that a group of nodes (servers) can agree on a sequence of entries in a log, even if some nodes may fail or behave unpredictably.

13. UML Diagrams for System Design

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system

Different Types of Diagrams:

- **Component Diagrams:**
  They are used to show code modules of a system in Unified Modeling Language (UML). They are generally used for modeling subsystems. It represents how each and every component acts during execution and running of a system program.

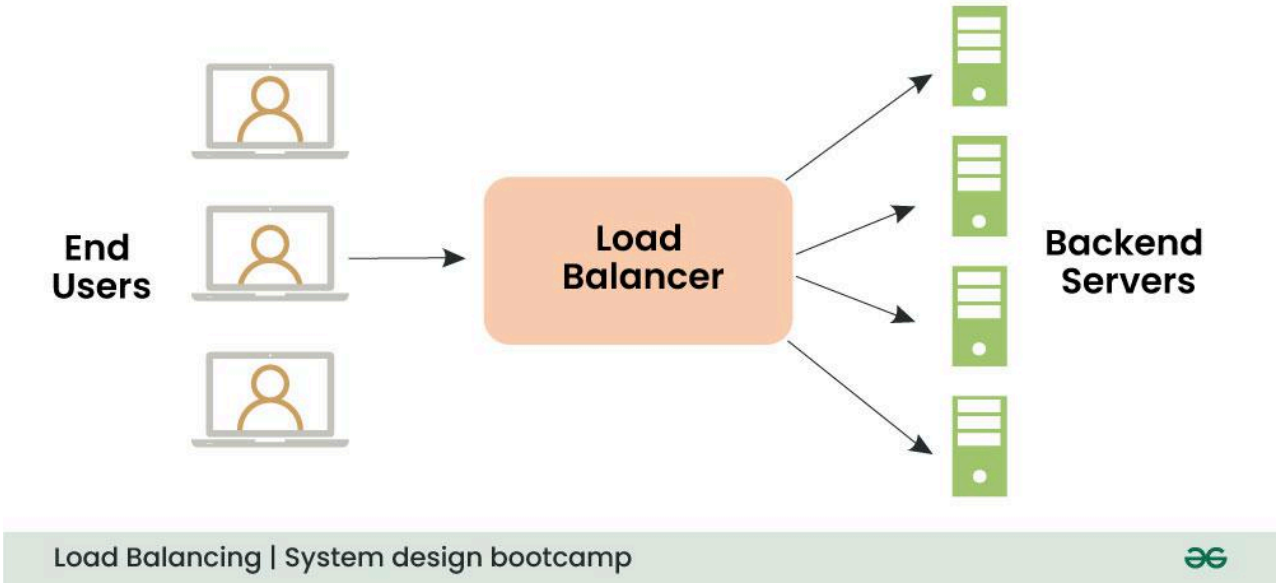
- **Activity Diagrams:**
  We use them to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram.
- **Use Case Diagram:**
  It is a vital tool in this System Design Interview Bootcamp, it provides a visual representation of how users interact with a system. It serves as a blueprint for understanding the functional requirements of a system from a user's perspective, aiding in the communication between stakeholders and guiding the development process.
- **Sequence diagram:**
  It is the most commonly used interaction diagram. An interaction diagram is used to show the interactive behavior of a system. Since visualizing the interactions in a system can be a difficult task, we use different types of interaction diagrams to capture various features and aspects of interaction in a system
- **Data Flow Diagram:**
  The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present.
- **Entity Relational Model:**
  Itis a model for identifying entities to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.
- **Package diagram:**
  It is a type of UML diagram mainly used to represent the organization and the structure of a system in the form of packages. A package is used as a container to organize the elements present in the system into a more manageable unit. It is very useful to represent the system's architecture and design as a cohesive unit and a concise manner.

14. Scalable web applications

14.1 DNS

Domain Name System (DNS) is a hostname for **IP address** translation service. DNS is a distributed database implemented in a hierarchy of name servers. It is an application layer protocol for message exchange between clients and servers. It is required for the functioning of the Internet.

14.2 Load Balancer



Load Balancing | System design bootcamp

A **load balancer** works as a “traffic cop” sitting in front of your server and routing client requests across all servers. It simply distributes the set of requested operations (database write requests, cache queries) effectively across multiple servers and ensures that no single server bears too many requests that lead to degrading the overall performance of the application. A load balancer can be a physical device or a virtualized instance running on specialized hardware or a software process.

14.3 N-tier Applications

An **N-tier architecture** (also known as multi-tier or layered architecture) is a design approach for software applications that divides the entire application into a set of interconnected and logically separated layers or tiers. Each tier represents a specific functionality or set of related functionalities.

This architecture enhances modularity, scalability, and maintainability by organizing the application into distinct layers, each with a specific role and responsibility. The “N” in N-tier represents the number of tiers or layers in the architecture, and it commonly refers to three-tier or four-tier architectures.

14.4 HTTP and REST

14.4.1 HTTP

It is the foundation of data communication on the World Wide Web. It is an application-layer protocol used for transmitting hypermedia documents, such as HTML. Developed primarily to facilitate communication between web browsers and servers, HTTP follows a client-server model, where the client initiates requests, and the server provides responses.

14.4.2 REST

REST is an architectural style for designing networked applications. It was introduced by Roy Fielding in his doctoral dissertation and emphasizes a stateless client-server communication model. RESTful systems adhere to a set of constraints to achieve simplicity, scalability, and uniformity.

14.4 Stream processing

It is a computing paradigm that involves the continuous processing of data streams in real-time. Unlike batch processing, where data is collected, stored, and processed in chunks, stream processing deals with data continuously as it is generated. This approach is crucial in scenarios where low-latency, real-time insights, and immediate actions on data are essential.

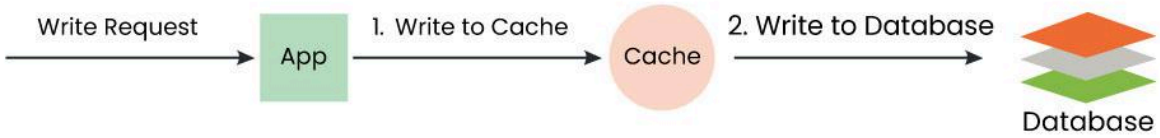
15. Caching

Caching is a system design concept that involves storing frequently accessed data in a location that is easily and quickly accessible. The purpose of caching is to improve the performance and efficiency of a system by reducing the amount of time it takes to access frequently accessed data.

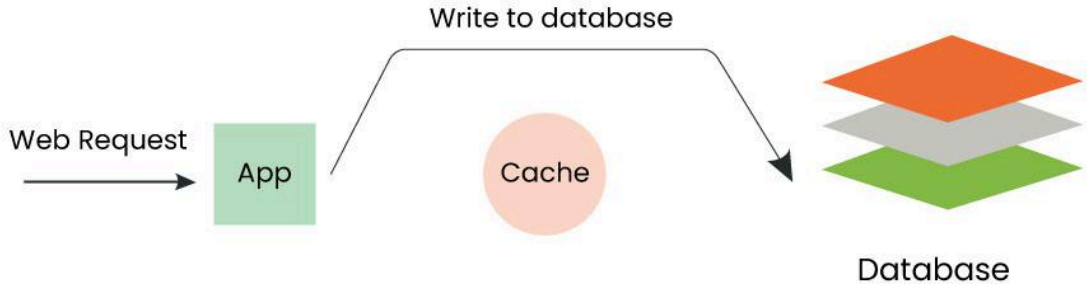
Caching can be used in a variety of different systems, including web applications, databases, and operating systems. In each case, caching works by storing data that is frequently accessed in a location that is closer to the user or application. This can include storing data in memory or on a local hard drive.

15.1 Cache invalidation

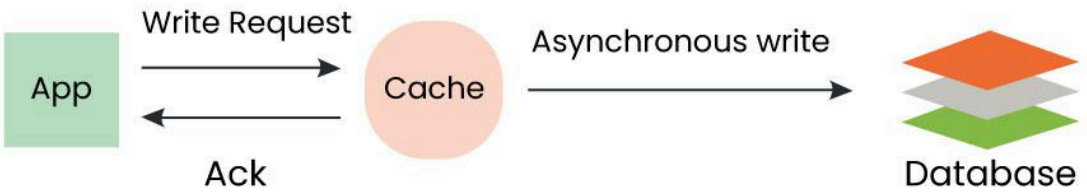
Cache invalidation is a state where we push away the data from the cache memory when the data present is outdated so do we perform this operation of pushing back/flushing the cache otherwise this still data will result in inconsistency of data. When cached data gets stale or inaccurate, cache invalidation is the process of removing or updating it. The terms “purge,” “refresh,” and “ban” are often used in content delivery networks (CDNs), web proxies, and application caches as cache invalidation techniques.



Write Through cache | System design bootcamp



Write around cache | System design bootcamp



Write back cache | System design bootcamp

15.2 Cache eviction

Cache eviction is the process of removing data from a cache when the cache becomes full or when the data is no longer needed. There are several different types of cache eviction algorithms used by computer systems, including:

- Time-Based Eviction
- Count-Based Eviction
- Query Result Eviction
- Cache Region Clearing

16. Essential Security Measures in System Design

In this System Design Interview Bootcamp, ensuring the security of the systems is a top-notch priority. This article will deep into the aspects of why it is necessary to build secure systems and maintain them. With various threats like cyberattacks, Data Breaches, and other Vulnerabilities, it has become very important for system administrators to incorporate robust security measures into their systems.

Some of the ways to ensure the security of your system are:

- **Authentication:**
  It is a crucial step or way to ensure the security of a system, it is very necessary to identify the person who is using that certain device, and to do so users need to authenticate themselves before using the machine.
- **Authorization:**
  It is a process in which the authority of the particular user trying to access a system is checked. It is as important as the authentication process, in this process the users are verified and their authority to access the system is being checked. It is done after the Authentication process.
- **Data Encryption:**
  It is also a crucial step to ensure the safety of any system.
- **Secure Coding Practices**:
  It is necessary to safeguard a system from various types of Cyber Threats. By following these practices, it is possible to safeguard the sensitive an confidential data, restrict unauthorized access, maintain the integrity of the system etc. Proper validation of the input and access control mechanism are important to stop common cyber attacks.

- **Network Security:**
  - It is also a important security measure in case of System Design, as it helps in maintaining the Confidentiality, Integrity and Availability of the data stored in the system.
  - Network Security comes into play when there is a need to transfer data or for any kind of communication purpose between two systems, having a strict network security policy will create a safe passage to transfer data or information keeping their intergrity and security.
- **Secure Communication Protocols:**

  They are vital for System Design when it comes to data transmission. These protocols help in ensuring the information remains confidential, integral and safe from any unauthorized access. Using secure protocols like SSL/TLS and HTTPS while transferring data is highly recommended to securely transmit sensitive data, most of the industries use this.
- **Third-Party Risk Management:**

  It is essential to learn in this System Design Interview Bootcamp to reduce the vulnerabilities caused by external factors like partners, vendors or services. Modern Systems rely on various Third Party applications like APIs and Cloud Services, so it is easy to get affected by any of them.

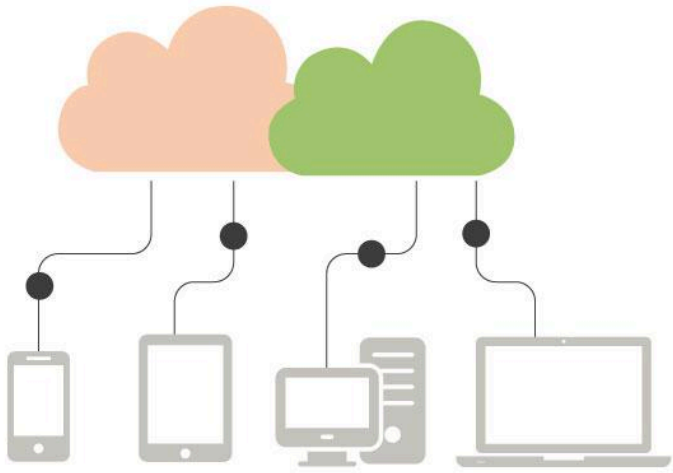
17. Machine Learning and System Design

System design in machine learning is vital for scalability, performance, and efficiency. It ensures effective data management, model deployment, monitoring, and resource optimization, while also addressing security, privacy, and regulatory compliance. A well-designed system enables seamless integration, adaptability, cost control, and collaborative development, ultimately making machine learning solutions robust, reliable, and capable of real-world deployment.

18. Containerization and System Design

Containerization is a lightweight form of virtualization that allows applications and their dependencies to be packaged and run consistently across different computing environments. Containers encapsulate an application, its runtime, libraries, and other dependencies, ensuring that it runs consistently regardless of the environment in which it is deployed. Docker, a widely used containerization platform, popularized this approach, but other containerization technologies exist, such as containerd and Podman.

19. The cloud and System Design



Cloud and System Design | System design bootcamp

Cloud computing is like renting tools or services over the internet instead of owning them on your own computer. Big companies, called cloud providers, take care of these services in their special buildings called data centers. This way, you don't have to worry about fixing problems like you would with your own computer. It's also cheaper, easier to use, and can grow or shrink based on what you need.

There are different companies that offer various services in the cloud, like storing your files, making sure things are secure, and managing who gets access to what. These services help you create flexible and efficient systems. The way you use these cloud services can be different, like using multiple clouds at once, mixing them with your own system, or just sticking to one. It's like having different options for how you want to organize your stuff in the cloud