|  |  |
| --- | --- |

# 

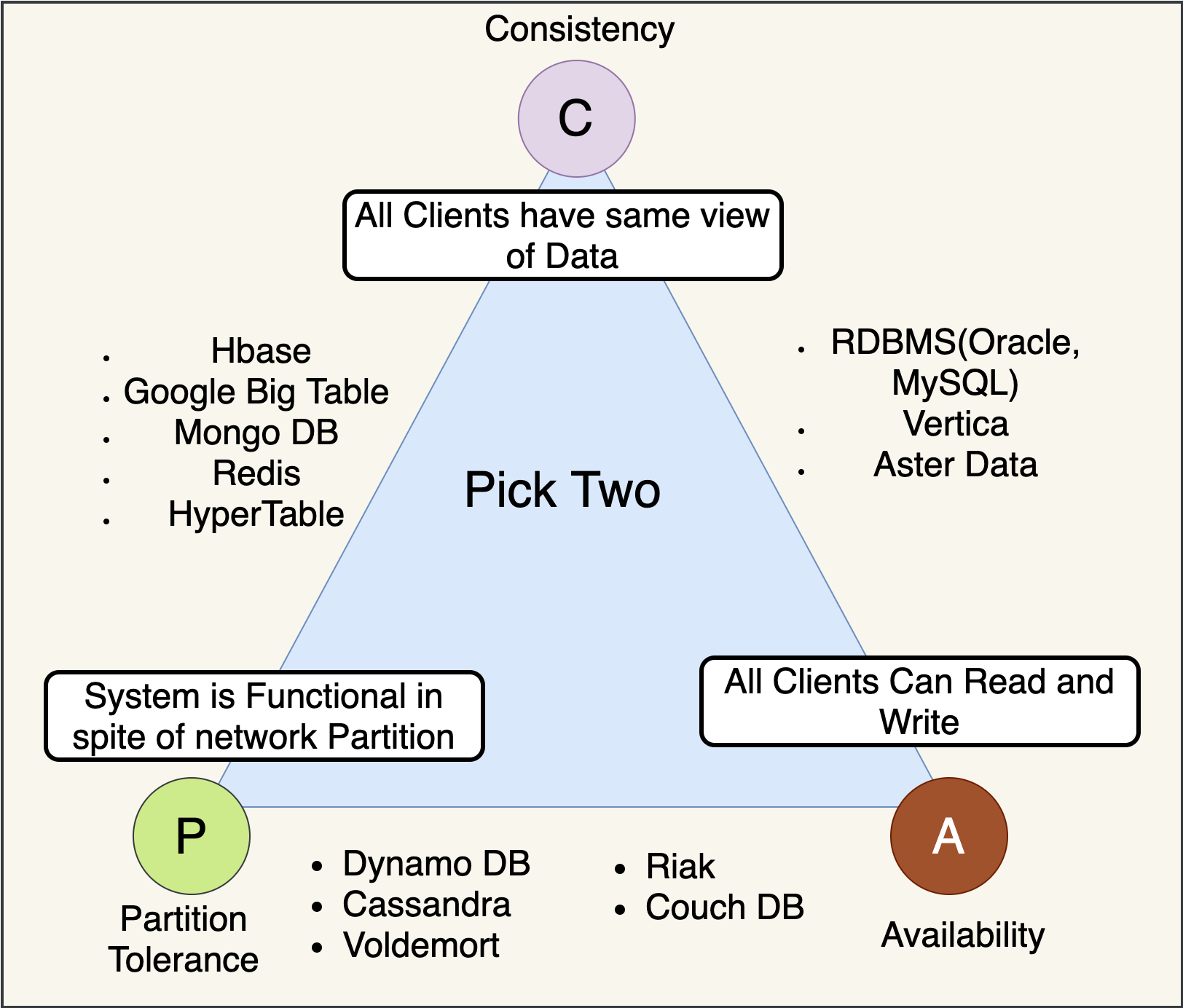
# Must know Full form

1. TLS = Transport Layer Security
2. SSL = Secure Sockets Layer
3. SMTP = Simple Mail Transfer Protocol
4. XMPP = Extensible Messaging and Presence Protocol
5. AMQP = Advanced Message Queuing Protocol
6. ELK Stack = Elasticsearch, Logstash, and Kibana
7. OLTP = Online Transaction Processing
8. OLAP = Online Analytical Processing
9. SIMT = Single Instruction, Multiple Threads
10. SIMD = Single Instruction, Multiple Data
11. JWT = JSON Web Token
12. IDP = Identity Provider
13. JSON = JavaScript Object Notation
14. HMAC =Hash-Based Message Authentication Code
15. SHA = Secure Hash Algorithm
16. GFS = Global File System
17. HDFS = Hadoop Distributed File System
18. DAU = Daily Active Users
19. RAID = Redundant Array of Independent Disks
20. SSH = Secure Shell
21. ISO = International Organization for Standardization
22. UML = Unified Modeling Language
23. TSDB = Time Series Database
24. RBAC = Role-Based Access Control
25. CQRS = Command and Query Responsibility Segregation
26. GDPR = General Data Protection Regulation
27. PCI = Payment Card Industry
28. ORM = Object-Relational Mapping
29. CSRF = Cross-Site Request Forgery
30. XSS = Cross-Site Scripting
31. DDoS = Distributed Denial of Service
32. SQLi = SQL Injection
33. PaaS = Platform as a Service
34. IaaS = Infrastructure as a Service
35. SaaS = Software as a Service
36. FaaS = Function as a Service
37. BCP/DR = Business Continuity Plan/Disaster Recovery
38. UTC = Coordinated Universal Time
39. MVCC = Multi-Version Concurrency Control

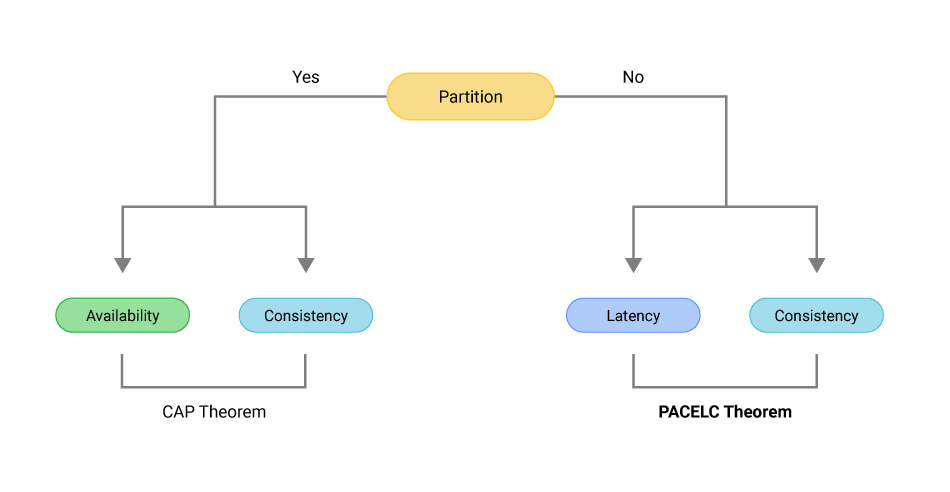
# 

# Must know Terms

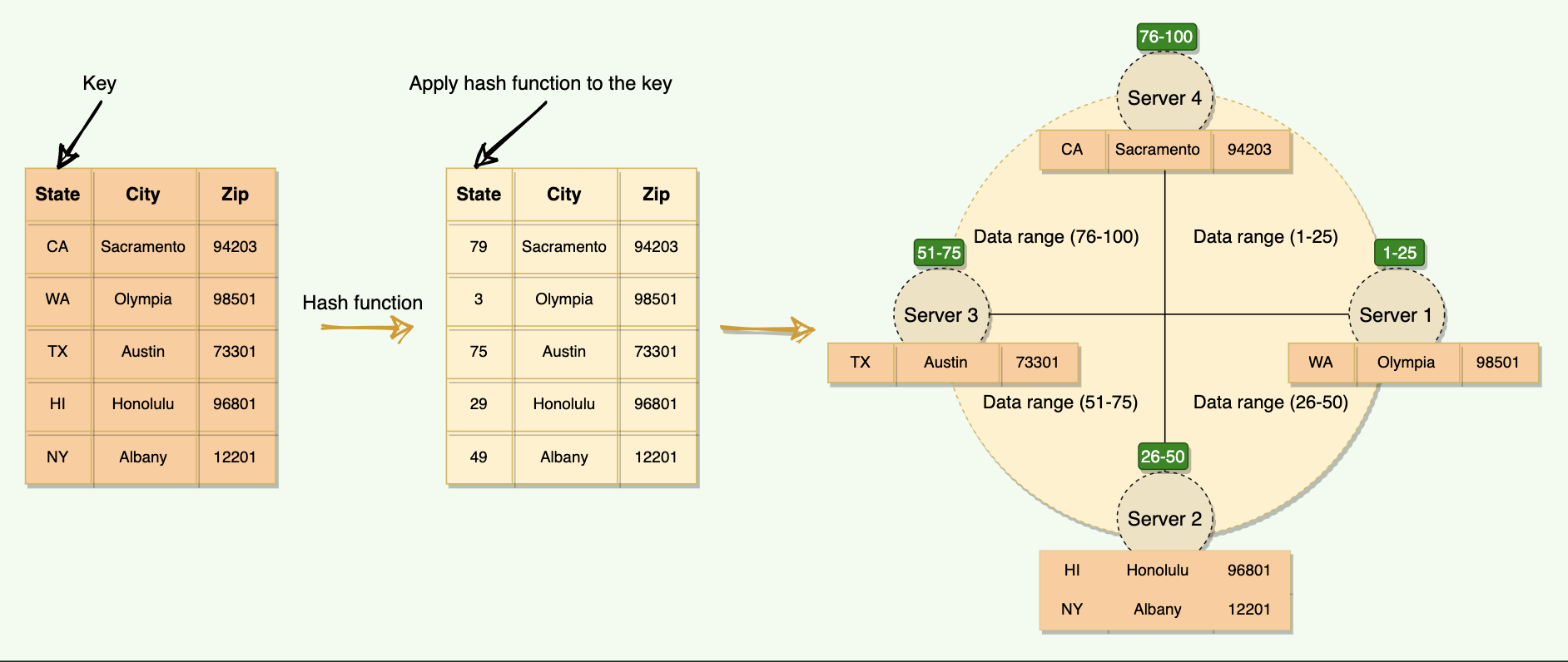
1. ***Note: Use*** [***https://chat.openai.com***](https://chat.openai.com) ***as a search engine.***
2. The Unix epoch is 00:00:00 Coordinated Universal Time (UTC) on 1 January 1970.
3. Database Design Principal
   * ACID Properties
     + ACID properties are typically associated with traditional relational databases and emphasize strong consistency.
     + Atomicity
       - Atomicity ensures that a transaction either succeeds completely or fails completely. if any part of a transaction fails, the entire transaction is rolled back, and the database is left unchanged.
     + Consistency
       - Consistency ensures that the database remains in a valid state before and after the execution of a transaction. It ensures any transaction will bring the database from one valid state to another, and prevents the database from entering an invalid state.
     + Isolation
       - Isolation ensures that the execution of multiple transactions concurrently does not interfere with each other.
     + Durability
       - Durability ensures that once a transaction is committed, it's permanently stored in the database and should not be lost even in the event of system failures (e.g., power outage, or hardware failure).
     + <https://www.educative.io/answers/what-are-acid-properties-in-a-database>
     + <https://www.scaler.com/topics/dbms/acid-properties-in-dbms/>
   * BASE
     + BASE principles are often employed in distributed systems and NoSQL databases, prioritizing availability and partition tolerance over strict consistency.
     + Basically Available
       - Focuses on providing availability over consistency, meaning that the system should remain operational even in the face of failures or network partitions.
     + Soft state
       - Allows for temporary inconsistencies or relaxed consistency requirements, particularly useful in distributed systems.
     + Eventually Consistent
       - Promotes the idea that system-wide consistency will eventually be achieved, given enough time and lack of further updates.
4. CAP Theorem
   * CAP theorem highlights the trade-offs in distributed systems.
   * Consistency
     + Consistency means all clients see the same data at the same time no matter which node they connect to.
   * Availability
     + Availability means any client which requests data gets a response even if some of the nodes are down.
   * Partition Tolerance
     + A partition indicates a communication break between two nodes. Partition tolerance means the system continues to operate despite network partitions.
   * <https://www.youtube.com/watch?v=BlkAOdFjGa8&list=PLGo1-Ya-AEQDFaT8RFh-lTQrh7RJCs4Ly&index=9>
   * <https://www.scaler.com/topics/cap-theorem-mongodb/>

[](https://www.nitendratech.com/database/cap-theorem/)

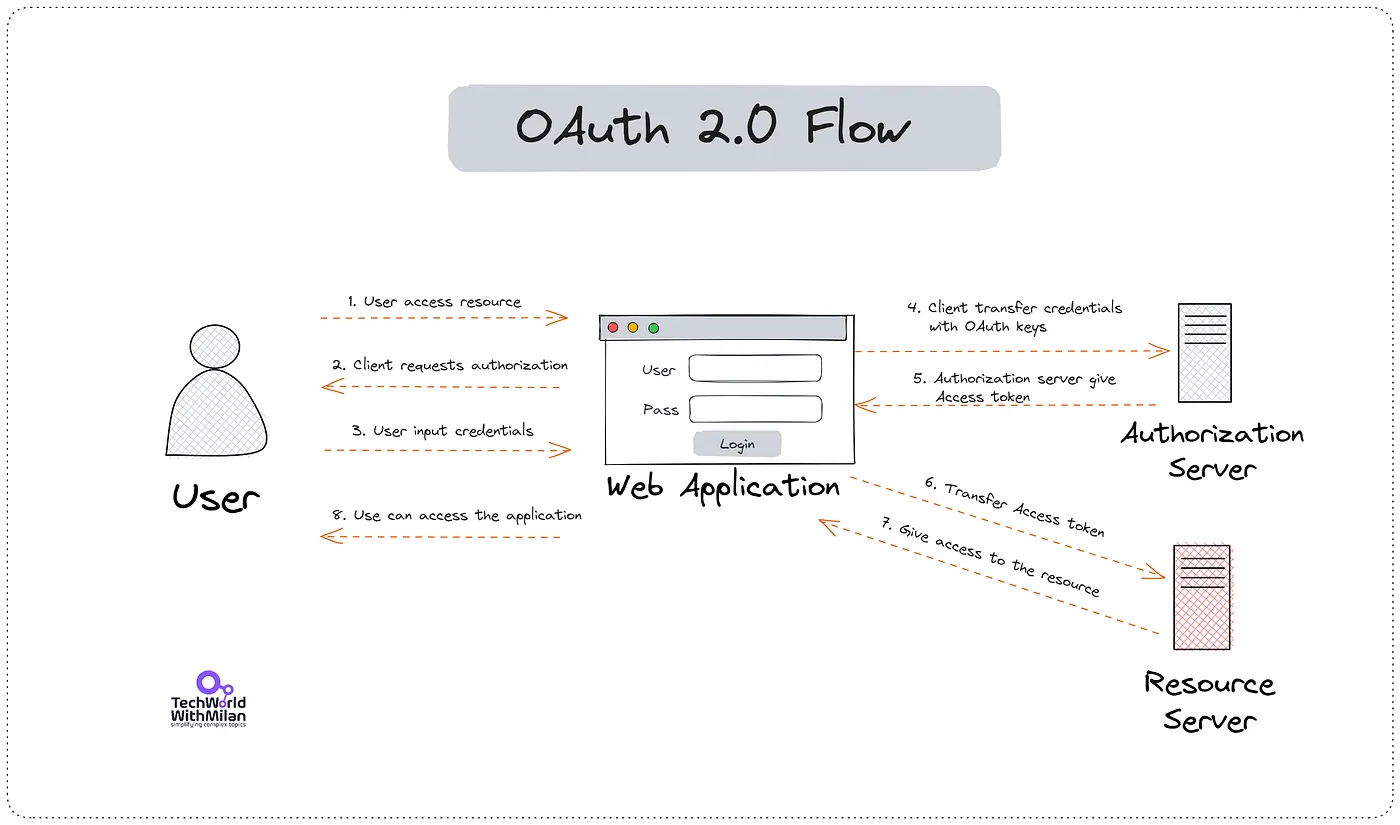
1. PACELC Theorem
   * It is an extension of the CAP theorem.
   * If there is a partition (P), a distributed system can tradeoff between availability (A) and consistency (C) else (E), when the system is running normally in the absence of partitions, the system can tradeoff between latency (L) and consistency (C).

[](https://www.scylladb.com/glossary/pacelc-theorem/)

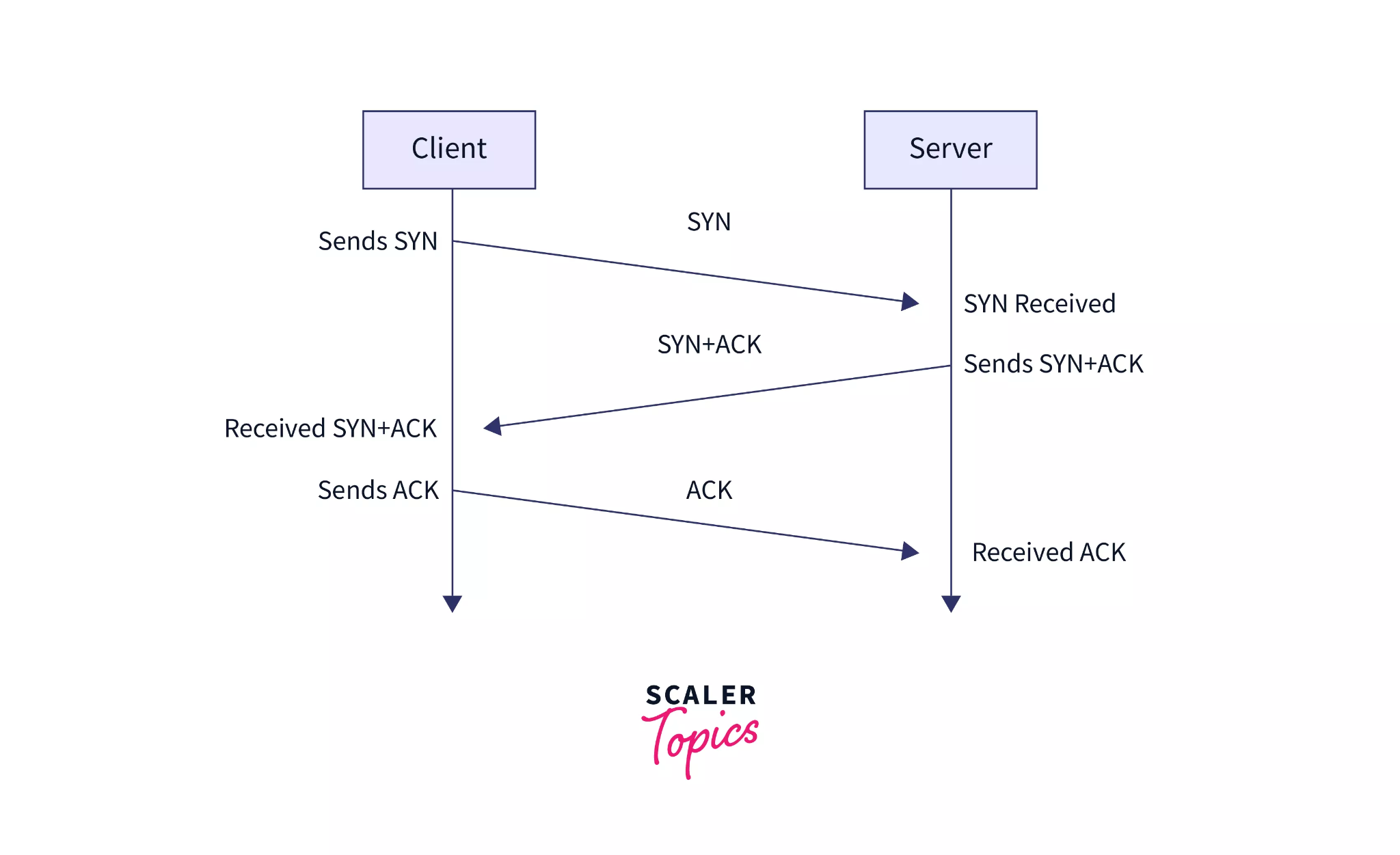
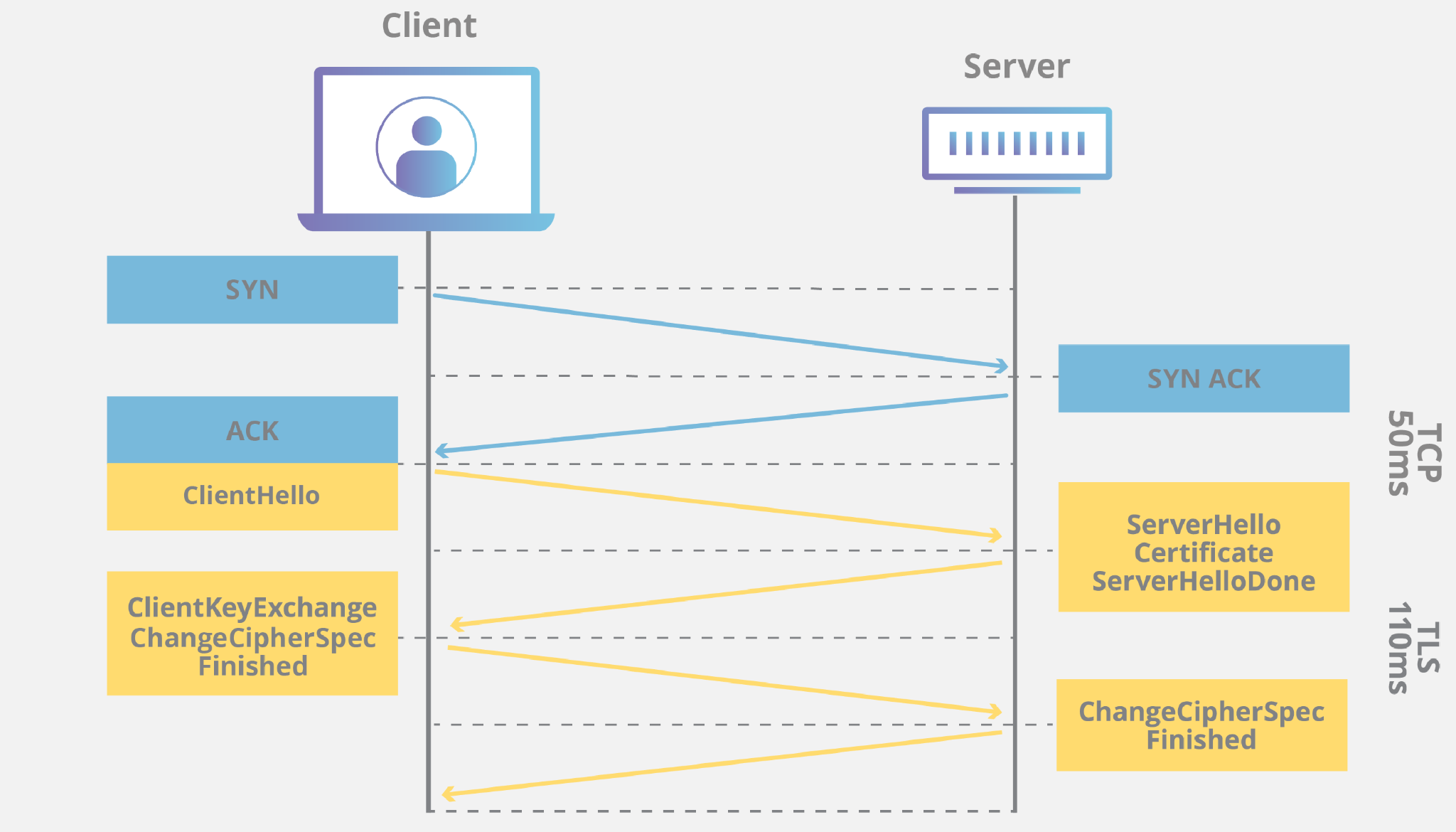
1. C in ACID vs CAP
   * <https://stackoverflow.com/questions/4813282/why-is-c-in-cap-theorem-not-same-as-c-in-acid/4813479#4813479>
2. Consistent Hashing
   * Consistent Hashing is a technique used in distributed systems to distribute data across multiple nodes while minimizing the need for data movement when nodes are added or removed from the system. The result of applying a hash function to information or data (e.g. *hash(data) = digest*) is called a message digest or digest, or hash value.
   * Whenever the system needs to read or write data, the first step it performs is to apply the hashing algorithm to the key. The output of this hashing algorithm determines within which range the data lies and hence, on which node the data will be stored.
   * Each key is assigned to a coordinator node (the node that falls first in the hash range), which first stores the data locally and then replicates it to N−1 clockwise successor nodes on the ring. If a client cannot contact the coordinator node, it sends the request to a node holding a replica.
   * Since there can be heterogeneous machines in the clusters, some servers might hold more Vnodes (Virtual Nodes) than others.
   * <https://www.educative.io/courses/grokking-the-system-design-interview/B81vnyp0GpY>
   * Internal Working
     + The entire hash value produced by a hash function is mapped onto a circular ring. Each server's hash i.e. hash(Node IP address) is positioned on this ring. when hashing a key i.e. hash(key), its resulting hash value is also placed on the ring, and stored to the nearest server or node in the clockwise direction.
     + <https://www.toptal.com/big-data/consistent-hashing>
     + <https://highscalability.com/consistent-hashing-algorithm/>
   * Implementation
     + Circular Array Ring - Store the hash range in a sorted array sorted by the start hash value. Use binary search to quickly locate the correct partition for a given hash value.
     + BST: Use a range query data structure tree like a segment tree.
     + Skip List (HashMap + DLL)
     + Hash table
     + <https://github.com/roottraveller/DSAlgo-HowToDoIt-learningcode/tree/master/consistent-hashing-implementation>
   * [Consistent Hashing | Algorithms You Should Know #1](https://www.youtube.com/watch?v=UF9Iqmg94tk&list=PLCRMIe5FDPseVvwzRiCQBmNOVUIZSSkP8&index=2)

[](https://www.designgurus.io/course-play/grokking-the-advanced-system-design-interview/doc/6376795982f3782df575c65f)

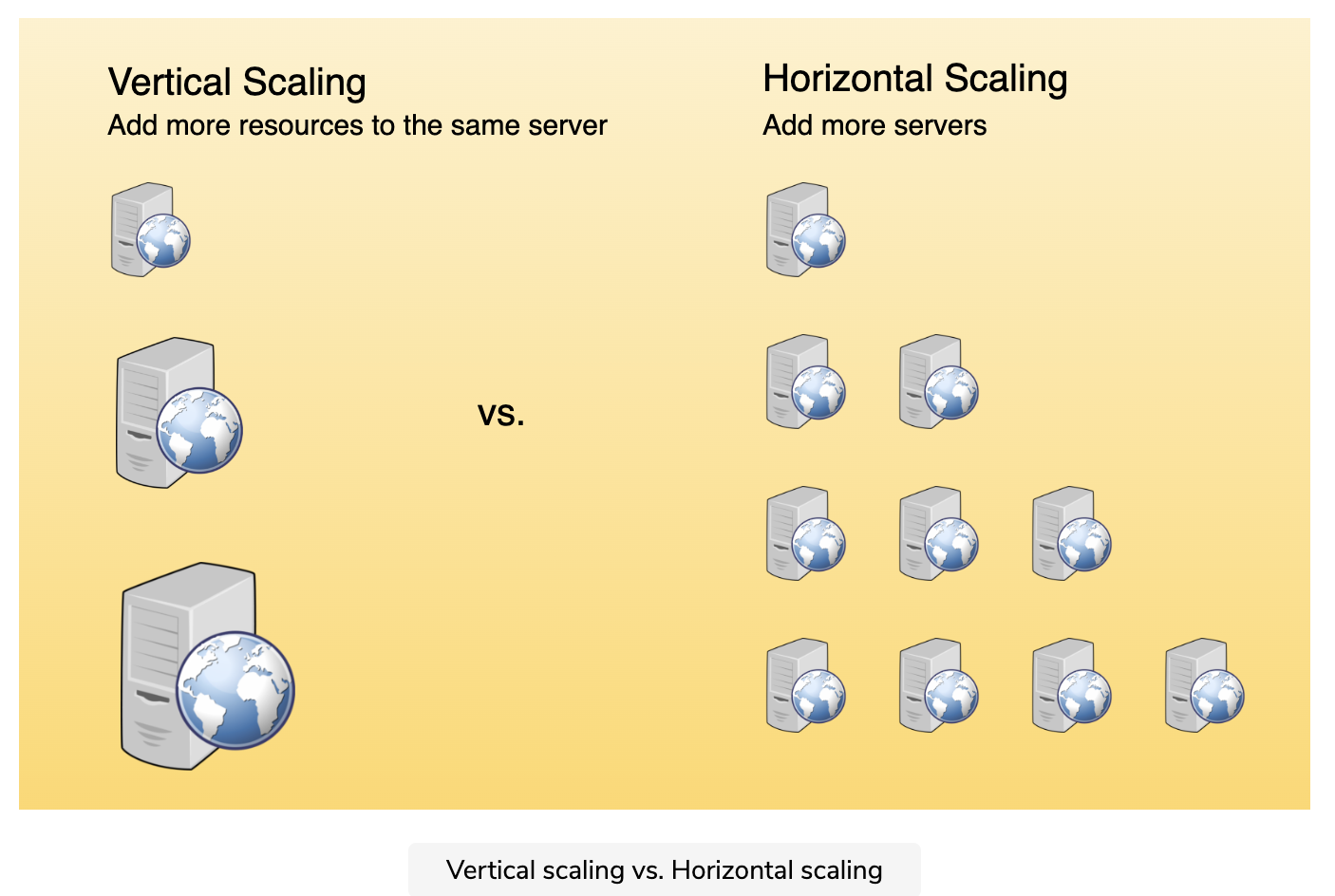
1. Load Balancer (LB)
   * LB is responsible for Traffic Distribution, High Availability, Scalability, Session Persistence, Health Checks, SSL Termination, Content-Based Routing, Rate Limiting, Logging and Monitoring etc. Examples are
     + HAProxy (High Availability Proxy),
     + Nginx,
     + AWS ELB (Elastic Load Balancing),
     + Azure Load Balancer,
     + Google Cloud Load Balancing etc.
   * If a server fails a health check, it is automatically removed from the pool, and traffic will not be forwarded to it until it responds to the health checks again.
   * Types of Load Balancing Technology
     + Hardware Load Balancers
     + Software Load Balancers
   * Types of Load Balancer
     + Application Load Balancer (L7 Balancer)
       - Provides advanced routing features and can make routing decisions based on HTTP headers, session information, request/response data, and content. Supports features like SSL termination, content-based routing, and host-based routing. Ideal for modern web applications with complex routing requirements and microservices architectures.
     + Network Load Balancer ((L4 Balancer))
       - Routes traffic based on IP address and port information only.
     + DNS Load Balancer
       - Distributes traffic across multiple servers using DNS (Domain Name System) resolution. Works by resolving domain names to multiple IP addresses, each corresponding to a different server.
   * Type of Load Balancing Algorithms
     + Dynamic Load Balancing Algorithms
       - Least Connection
       - Weighted Least Connection
       - Least Response Time
       - Resource-based
       - Adaptive Load Balancing
       - Dynamic Proximity-based Load Balancing
     + Static Load Balancing Algorithms
       - Round Robin
       - Weighted Round Robin
       - IP Hash (client's IP address hash value), This ensures that requests from the same client are always routed to the same server, which can be useful for maintaining session state.
       - Static Proximity-based Load Balancing (based on geographic regions)
       - Static Content-based Load Balancing
   * <https://www.scaler.com/topics/aws/load-balancing/>
   * <https://www.nginx.com/resources/glossary/load-balancing/>
   * <https://blog.bytebytego.com/p/ep47-common-load-balancing-algorithms#%C2%A7what-are-the-common-load-balancing-algorithms>
2. Consistent Hashing vs Load Balancing
   * Consistent hashing handles the distribution of data(data partitioning) across nodes and determines which node is responsible for each piece of data based on a consistent hashing scheme.
   * Load balancing algorithms decide which replica or node should handle a request among the replicas based on load balancing algorithms.
3. Multi-node Architecture
   * Active-Active vs Active-Passive (Hot/Cold) Setup
     + In an active-active configuration, all servers or resources in the system are actively serving traffic and processing requests simultaneously.
     + In an active-passive configuration, one server (or a group of servers) is designated as active (hot), while the remaining servers are passive (cold) or standby. When an active or hot server goes down, one of the passive or cold servers elects itself as a hot server.
     + <https://www.jscape.com/blog/active-active-vs-active-passive-high-availability-cluster>
4. API gateway
   * API GW serves as a centralized entry point for all incoming API requests, allowing clients to interact with multiple backend services through a single interface. It provides a wide range of functionality to manage, secure, and optimise API traffic i.e. Routing and Load Balancing, Authentication and Authorization, Rate Limiting and Throttling, Fault Tolerance and Circuit Breaking, Caching, Request and Response Transformation, Logging and Monitoring, SSL Termination, IP Whitelisting etc. Examples
     + NGINX API GW
     + Kong API GW
     + Amazon API Gateway
   * <https://stackoverflow.com/a/61210085/5167682>
   * <https://medium.com/@anoobblogger/api-gateway-definition-lb-caching-and-more-e6c764a485c0>
5. CDN
   * <https://www.imperva.com/learn/performance/cdn-architecture/>
   * <https://blog.bytebytego.com/p/how-does-cdn-work>
   * CDN Providers
     + Akamai
     + Cloudflare
     + Amazon CloudFront
     + <https://en.wikipedia.org/wiki/Content_delivery_network#Notable_content_delivery_service_providers>
6. Circuit Breaker
   * A circuit breaker is used in distributed systems to improve resilience and fault tolerance by preventing cascading failures and reducing the impact of system failures. Examples
     + Resilience4j Circuit Breaker
     + Netflix Hystrix Circuit Breaker
     + istio/istio-circuitbreaker
   * <https://resilience4j.readme.io/docs/circuitbreaker>
7. Rate Limiting
   * Rate limiting is used to control the rate at which clients or users can access a particular resource or service over a specified period of time. It helps prevent abuse, misuse, or overloading of the system by limiting the number of requests or transactions that can be processed within a given timeframe. Examples
     + Resilience4j Rate Limiter
     + Envoyproxy Rate Limiter
     + Guava Rate Limiter
     + Bucket4j
     + Spring Cloud Gateway Rate Limiting
   * The rate limiter returns a HTTP response code 429 - Too Many Requests with the following HTTP headers to the client.
     + X-Ratelimit-Remaining - The remaining number of allowed requests within the window.
     + X-Ratelimit-Limit - It indicates how many calls the client can make per time window.
     + X-Ratelimit-Retry-After - The number of seconds to wait until you can make a request again.
   * Implementation Strategies
     + Client-Side Rate Limiting
     + Server-Side Rate Limiting
     + Distributed Rate Limiting
   * Rate Limiting Algorithms
     + Token-Based
       - Use a fixed-size token. Each request is assigned a token if available else rate-limited. tokens are reset at a fixed interval.
     + Leaking Bucket
       - Use a virtual "bucket/queue" with a fixed capacity and a constant leak rate. requests are added to the bucket if not full else rate-limited.
     + Fixed Window Counter
       - Requests are counted within fixed time intervals, or windows. If the number of requests exceeds the allowed threshold, the request is rate-limited. the window is reset at a fixed time interval.
     + Rolling/Sliding Window (Production)
       - Requests within the current (rolling or sliding) window are counted, and if they exceed the threshold, further requests are rate-limited.
   * Hard vs Soft Rate Limiting
     + Hard - The number of requests cannot exceed the threshold.
     + Soft - Requests can exceed the threshold for a short period.
   * <https://tech.groww.in/rate-limiter-and-its-algorithms-with-illustrations-564455162935>
   * <https://www.system.design/SystemDesign/RateLimiter>
8. Authentication and Authorization
   * Authentication (Auth)
     + Authentication is the process of verifying the identity of a user or system attempting to access a resource or service. The goal of authentication is to ensure that the entity requesting access is who it claims to be.
     + Authentication Methods
       - Username and Password
       - Biometric Authentication
       - Multi-Factor Authentication (MFA)
     + Authentication Protocols
       - OpenID Connect
       - Kerberos
       - SAML (Security Assertion Markup Language)
       - LDAP (Lightweight Directory Access Protocol)
   * Authorization (Authz)
     + Authorization is the process of determining what actions a user or system is allowed to perform on a resource or within a system. The goal of authorization is to enforce access control policies (permissions or roles) and limit access to authorized users or systems.
     + Authorization Mechanisms
       - Coarse-grained Authorization
       - Fine-grained Authorization
     + Authorization Methods
       - Access Control Lists (ACLs)
         * ACL is Fine-grained and defines permissions at a very detailed level, such as which users can read, write, delete or execute a specific file or resource.
         * Libraries are Spring Security ACLs.
       - Role-Based Access Control (RBAC)
         * RBAC is Coarse-grained and assigns permissions to roles (Admin, User, Developer, Editor, Viewer etc) rather than individual users.
         * Libraries are Spring Security, Keycloak.
       - Attribute-Based Access Control (ABAC)
     + Authorization Protocols
       - OAuth 2.0
       - SAML (Security Assertion Markup Language)
       - LDAP (Lightweight Directory Access Protocol)
9. JWT (JSON Web Token)
   * JWT token can be used for both authentication and authorization purposes. A JWT token consists of three parts: a header (metadata), a payload (claims), and a signature. ​​
   * <Base64URL encoded header>.<Base64URL encoded payload>.<Base64URL encoded signature>
     + The header contains metadata about the token, such as the type of token (JWT) and the cryptographic algorithm used for signing.
     + The payload contains the claims, which are statements about an entity (typically the user) and additional data. claims include the subject (sub), issuer (iss), expiration time (exp), and custom user information.
     + The signature is created by taking the encoded header, the encoded payload, and a secret key known only to the server. This data is combined and then signed using a cryptographic algorithm specified in the header (e.g., HMAC SHA-256, RSA). The resulting signature is appended to the first two parts of the JWT.
   * What are the issues with JWT?
     + JWTs have long token expiration times which can pose a security risk if a token is compromised. Additionally, JWTs are base64-encoded but not encrypted by default, making sensitive information in the payload easily decodable. Furthermore, if JWTs contain a lot of claims, they can become large, increasing the size of HTTP headers and potentially impacting performance, particularly for mobile and low-bandwidth connections.
   * <https://www.scaler.com/topics/spring-boot/jwt-token/>
   * <https://blog.bytebytego.com/p/ep69-explaining-json-web-token-jwt>
   * <https://jwt.io/>
10. OAuth 2.0
    * OAuth 2.0 is an open standard authorization protocol that enables third-party applications to access protected resources on behalf of a resource owner (user) without disclosing the user's credentials. OAuth 2.0 defines a set of roles, grant types, endpoints, and protocols for authentication and authorization.
    * Authorization Server
      + The server that authenticates the resource owner and issues access tokens to the client after obtaining authorization. Before initiating the OAuth flow, the client application (e.g., a web or mobile app) must register (client ID, client secret, client scope or permissions) with the authorization server. Auth server sends 2 token access token and refresh token to client.
      + Access Token (Opaque Token)
        - Access tokens are typically opaque strings or JWT that contain information about the authorization and scope of access granted to the client. It is used by the client to access protected resources on the resource server. Access tokens are short-lived and typically have an expiration time.
      + Refresh Token
        - Refresh tokens are securely stored by the client application and are used to request new access tokens from the authorization server without requiring the user to re-authenticate. refresh tokens are long-lived.
    * <https://blog.bytebytego.com/p/ep72-oauth-20-explained-with-simple>

[](https://medium.com/@techworldwithmilan/how-does-oauth-2-0-work-bea67a760aa5)

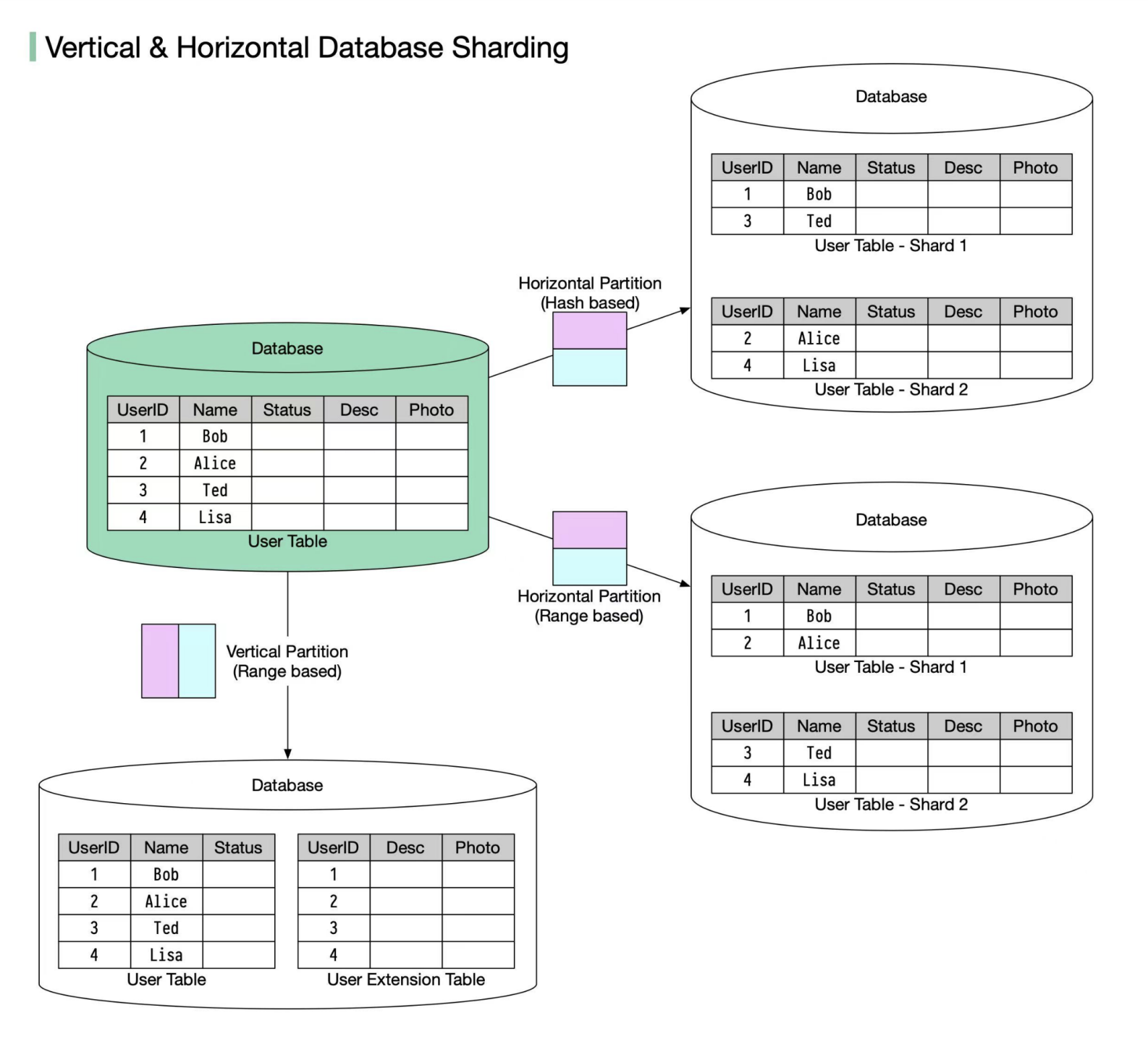
1. Application Server and Web Server
   * Web server's primary function is to handle HTTP requests from clients (such as web browsers) and serve static content (HTML pages, images, CSS files, etc.) to users over the internet. A web server may talk to an application server for data etc.
     + Examples - Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS) etc
   * Application servers execute the business logic of web applications, including processing user input, interacting with databases, performing calculations, generating dynamic content and often provide middleware services such as connection pooling, transaction management, messaging, and caching to support the development of complex web applications.
     + Examples - Apache Tomcat, Jetty, Python Flask, Nginx,
   * <https://stackoverflow.com/questions/936197/what-is-the-difference-between-application-server-and-web-server>
2. Stateless and Stateful Servers
   * Stateless Servers
     + A stateless server does not maintain any session state between requests. Each request is processed independently, with no reliance on past interactions or shared session state. This makes stateless servers highly scalable and fault-tolerant but may limit their ability to provide personalized responses or maintain context between interactions.
   * Stateful Servers
     + A stateful server maintains state information about client sessions. It keeps track of session-related data, such as connection status, context, and past interactions. This allows the server to provide continuity and personalized responses based on the client's history.
3. Stateful vs Stateless Connection
   * Stateful Connection
     + In a stateful connection, the server maintains information about the state of the communication session between the client and server. Each request builds upon the previous one, and the server remembers the context of the interaction.
   * Stateless Connection
     + In a stateless connection, each request from the client to the server is independent and self-contained. The server does not maintain any session state between requests and treats each request as new and unrelated to previous ones.
4. Network Protocols
   * Client Server Protocol
     + HTTP/HTTPS - Operates using a single TCP connection for unidirectional communication.
     + FTP - Operates using two connections, one for control and one for data, and lacks encryption.
     + SMTP - Often paired with IMAP or POP3 for email communication.
     + Websocket - Enables bidirectional communication.
   * Peer to Peer Protocol
     + WebRTC - Operates using UDP for real-time communication.
5. TCP Connection (TCP Socket)
   * A TCP connection or TCP socket, is a communication link established between two endpoints (client and server) over a TCP/IP network. It enables bidirectional data exchange between processes running on the client and server systems. TCP sockets are identified by a combination of IP addresses and port numbers.
   * Algorithms
   * 3-Way Handshake (TCP handshake)
     + 3-Way Handshake is a basic protocol used by TCP to establish a connection and does not involve encryption or security mechanisms. This is used by HTTP.
   * TLS (Transport Layer Security) Handshake (SSL handshake)
     + TLS Handshake is a more complex protocol used by TLS/SSL to establish a secure encrypted connection within the application layer. TLS handshake occurs on top of the TCP connection established through the 3-Way Handshake. It involves multiple steps to negotiate encryption parameters, authenticate the parties, and establish a shared secret key for secure communication. This is used by HTTPS.
   * TCP -> HTTP -> TLS/SSL -> HTTPS -> Websocket
   * [SSL, TLS, HTTPS Explained](https://www.youtube.com/watch?v=j9QmMEWmcfo&list=PLCRMIe5FDPseVvwzRiCQBmNOVUIZSSkP8&index=9)

[](https://www.scaler.com/topics/computer-network/tcp-3-way-handshake/) 

1. WebSockets
   * WebSockets is a communication protocol that provides full-duplex communication channels over a single TCP connection.
   * WebSockets use the WebSocket protocol, which is a standardized protocol for real-time, full-duplex communication between a client and a server over a single, long-lived connection. The WebSocket protocol is designed to provide low-latency, bidirectional communication between web browsers and web servers, enabling interactive and real-time web applications. It builds on top of the HTTP protocol and utilizes the same TCP ports (typically port 80 for unencrypted connections and port 443 for encrypted connections using TLS/SSL) for communication.
   * The client establishes a WebSocket connection through a process known as the WebSocket handshake. WebSockets use an initial HTTP handshake to establish the connection, after which the protocol is upgraded to the WebSocket protocol, allowing for efficient and continuous communication. WebSocket handshake is a 4-step process.
     + Client sends header “Connection: Upgrade”, “Upgrade: websocket” indicating that it wants to upgrade the connection to websocket.
     + Server responds with an HTTP 101 status code (Switching Protocols).
     + Client sends another HTTP request with headers "Upgrade: websocket", "Connection: Upgrade", and a unique "Sec-WebSocket-Key" header.
     + Server responds with the headers "Upgrade: websocket" and "Connection: Upgrade" confirming the upgrade.
   * Once a WebSocket connection is established between a client and a server, it remains open until either the client or the server explicitly closes the connection or until a network error or timeout occurs.
   * <https://www.scaler.com/topics/computer-network/websocket/>
2. HTTP Long Polling
   * HTTP Long Polling is a technique used in web development to simulate real-time communication between a client (usually a web browser) and a server over the HTTP protocol. It's often used when WebSocket support is not available or practical.
   * With long polling, the client requests information from the server with the expectation that the server may not respond immediately. Instead of immediately responding to the client's request, the server holds the request open, keeping the connection active and waits for new data or updates to become available.
   * Each Long-Poll request has a timeout. The client has to reconnect periodically after the connection is closed due to timeouts. The Connection header is used to ensures that the connection remains open for a longer period i.e. ‘Connection: keep-alive’.
   * <https://www.educative.io/answers/what-is-http-long-polling>
3. Connection Pooling
   * Instead of opening a new connection each time an application needs to interact with a resource, connection pooling allows the application to reuse existing connections from a pool of pre-established connections. This can significantly reduce the overhead of creating and tearing down connections, leading to improved performance and resource utilization.
4. Vertical Scaling vs Horizontal Scaling
   * According to Amazon RDS, you can get a database server with 24 TB of RAM. Stackoverflow.com runs on all data in the RAM model.
   * <https://aws.amazon.com/ec2/instance-types/high-memory/>

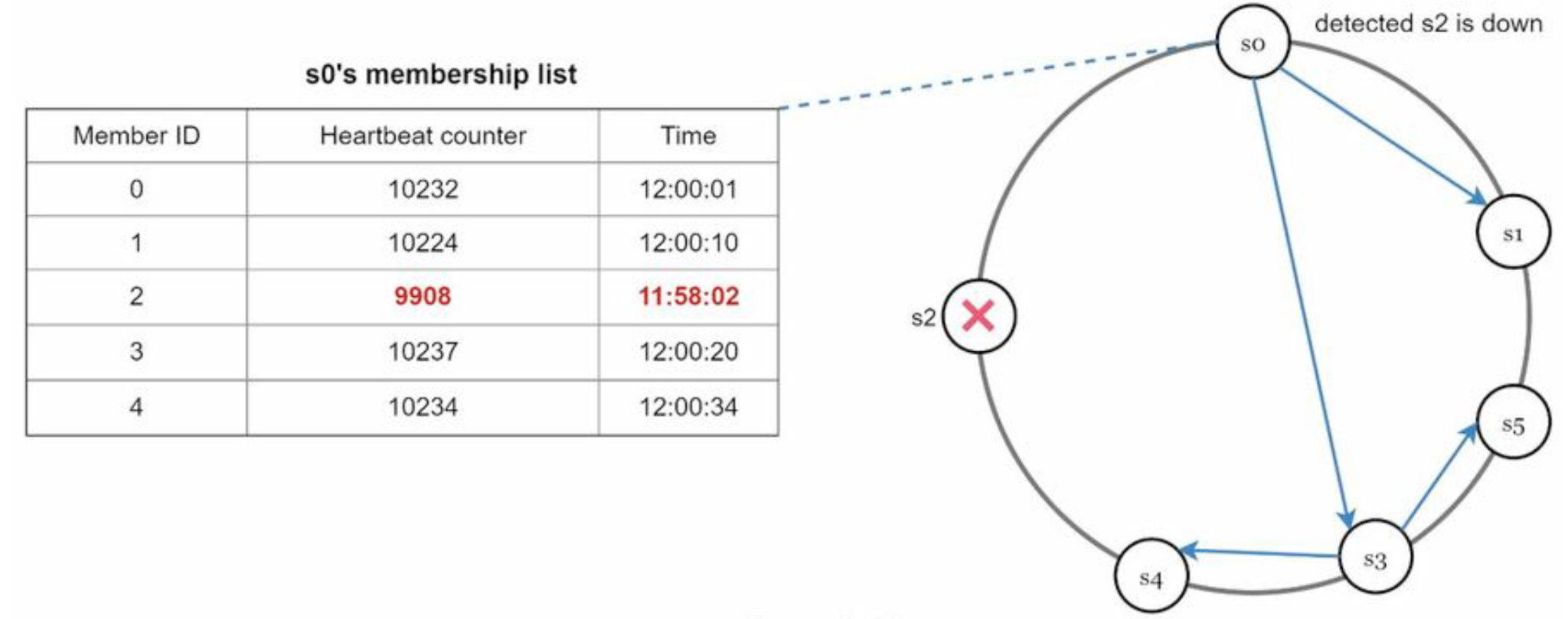
[](https://hackmd.io/@systemdesign/HkUTVbinu)

1. Elasticity vs Scalability
   * Scalability refers to the capability of a system to increase its capacity to handle increased loads. Elasticity refers to the ability of a system to automatically adjust its resources to handle varying workloads in real-time. It means dynamically allocating or deallocating resources as demand changes (easily scale up or down based on your needs).
2. Sharding and Partitioning
   * Database is sharded and data is partitioned. The most important factor to consider when implementing a sharding strategy is the choice of the sharding key. Sharding key (known as a partition key) consists of one or more columns that determine how data is distributed.
   * <https://stackoverflow.com/questions/20771435/database-sharding-vs-partitioning>
   * Data Partitioning
     + Horizontal Partitioning
       - Each partition contains a subset of rows (with all columns) from the original table.
     + Vertical Partitioning
       - Each partition contains a subset of columns (for all rows) from the original table. This partitioning is based on the nature of the data or access patterns.
   * Database Sharding
     + Horizontal Sharding
       - In horizontal sharding, the table is horizontally partitioned into smaller shards, and each shard is stored on a separate server or node.
     + Vertical Sharding
       - In vertical sharding, different columns of the table are stored on separate shards, and each shard contains a subset of columns for all rows.

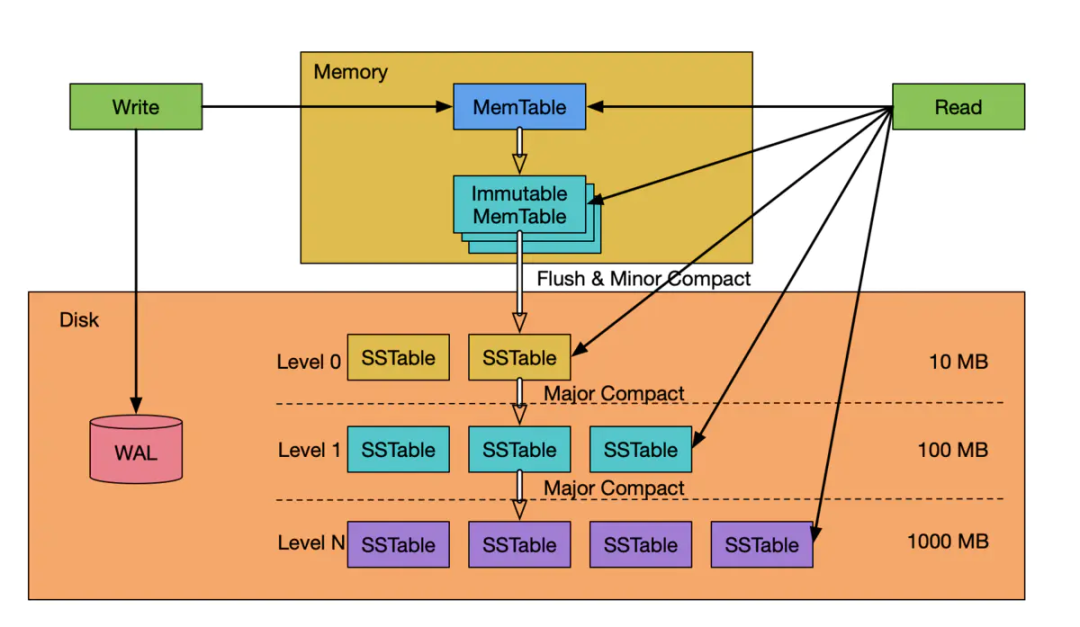
[](https://blog.bytebytego.com/p/vertical-partitioning-vs-horizontal)

* + Partitioning Criteria
    - Key or Hash-based Partitioning (Consistent Hashing)
    - Range Partitioning
    - Composite Partitioning
    - List-range Partitioning
    - Round-robin Partitioning
  + Celebrity problem (Hotspot/Hotkey Problem)
    - The hotspot or hotkey problem occurs in distributed databases when a specific shard or partition receives a disproportionately high volume of read or write requests compared to others. This imbalance can result from uneven data distribution, skewed access patterns, or particular keys being accessed more frequently than others. Hotkeys are specific keys or identifiers within a distributed dataset that experience significantly higher read or write operations, often due to popular data items, frequently queried records, heavily accessed user accounts, or frequently updated counters.
    - To solve this, we can utilize caching mechanisms, use read replicas to distribute read traffic etc.

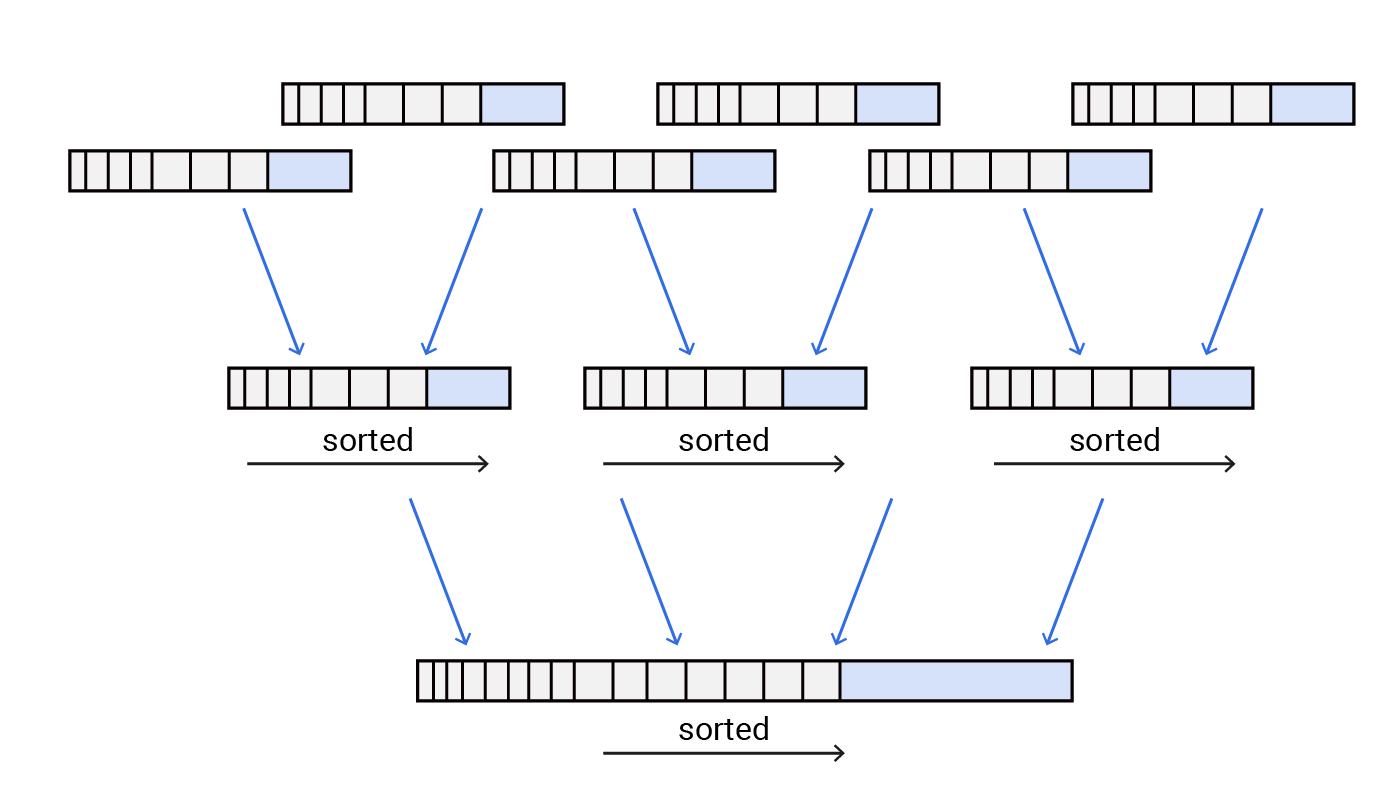
1. Hashing/Hash Function
   * Good hashes have 4 properties
     + it is easy to compute the hash value for any given message.
     + it is infeasible to generate a message with a given hash.
     + it is infeasible to modify a message without changing the hash.
     + it is infeasible to find two different messages with the same hash.
   * Hashing Algorithms
     + MD5 Hashing Algorithms (Message-Digest Method 5)
       - MD5 output length is 128 bits (32 chars on base16) i.e. “65a8e27d8879283831b664bd8b7f0ad4”.
       - <https://www.comparitech.com/blog/information-security/md5-algorithm-with-examples/>
     + RIPEMD-128, RIPEMD-160, RIPEMD-256
       - RIPEMD-160 output is 160 bits (40 chars) i.e. "98034a99f9f77f3bb93e740c9f4f2a6d92e85c63".
     + SHA-1, SHA-256, SHA-512 (Secure Hash Algorithms)
       - SHA-1 output is 160 bits.
       - SH256 output is 256 bits and SHA512 output is 512 bits i.e. "b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9".
     + MurmurHash
     + The output hashes are represented on base 16 (4 bits).
   * <https://stackoverflow.com/questions/415953/how-can-i-generate-an-md5-hash-in-java?rq=1>
2. Properties of a Distributed System
   * High Availability
     + Data replication increases system availability by distributing data copies across multiple nodes. This allows clients to access data from nearby replicas, reducing latency and improving responsiveness. In the event of node failures, clients can still access data from other available replicas, ensuring uninterrupted service.
   * Fault tolerance
     + Fault tolerance refers to the ability of a system to continue operating without interruption when one or more of its components fail. Data replication improves fault tolerance by ensuring that data remains accessible even in the event of node failures or network partitions. If a node hosting a copy of the data fails, other replicas can be used to serve read and write requests, preventing data loss or service disruption.
   * Eventually Consistency
     + Eventual consistency is a consistency model in distributed systems where data consistency is guaranteed to be achieved eventually, even in the presence of concurrent updates and network partitions.
   * Redundancy
     + Redundancy ensures that if one copy of the data becomes unavailable due to node failure or network issues, other copies can still be accessed to maintain system availability. redundancy is achieved by storing multiple copies of data across distributed nodes.
3. Characteristics of a Distributed Database
   * Distributed & Decentralized
   * Scalable
   * Highly Available
   * Data Replication
   * Eventually Consistent
   * Fault-Tolerant and Reliable
   * Durable
   * Partitioning and Sharding
   * Concurrency Control & Transaction Management
   * Cross-Datacenter Replication (XDCR)
4. Consistency models
   * Strong consistency
   * Weak consistency
   * Eventual consistency
5. Data Replication
   * Master-Slave Replication
     + There is one master server and one or more slave servers. The master server is responsible for receiving write operations and propagating these changes to the slave servers. Slave servers replicate data from the master server asynchronously in real time.
   * Multi-Master Replication
     + Master-Master Replication
       - All nodes typically act as "master" nodes, each capable of handling both read and write operations. Each master server can accept write operations independently, and changes made on one master server are asynchronously replicated to the other master servers. There are no slave nodes in this.
     + Peer-to-Peer Replication
       - In peer-to-peer replication, all nodes in the distributed system are equal peers, and each node can accept both read and write operations. Changes made on any node are propagated to other nodes in a peer-to-peer fashion, with each node acting as both a source and destination of data replication.
   * Snapshot Replication
     + Snapshot replication involves periodically taking snapshots of data from a source node and replicating these snapshots to other nodes. Snapshot replication is useful for creating backups, data migration, and disaster recovery purposes.
6. CDC (Change Data Capture)
   * CDC ensure that any changes in data due to operations such as inserts, updates, and deletions are identified, captured, and automatically applied to another data repository instance, or made available for consumption by applications and other tools.
7. Quorum
   * Quorum is a concept used in distributed systems to achieve consensus and make decisions among a group of nodes. It refers to the minimum number of nodes or replicas that must agree on a particular operation before it is considered successful or valid. It plays a crucial role in ensuring data consistency and fault tolerance in distributed environments.
   * <https://medium.com/double-pointer/replication-for-system-design-interview-11-quorum-variations-1972bb4df42f>
   * Type of Quorum
     + Read Quorum (R)
       - Read quorum specifies the minimum number of nodes that must agree on a read operation for it to be considered valid.
     + Write Quorum (W)
       - Write quorum specifies the minimum number of nodes that must agree on a write operation for it to be considered successful.
     + N is the total number of nodes or replicas in the system.
       - If W + R > N, strong consistency is guaranteed.
       - If W + R <= N, strong consistency is not guaranteed. System is eventual consistency.
       - If R = 1 and W = N, the system is optimized for a fast read.
       - If W = 1 and R = N, the system is optimized for fast write.
   * Variations of Quorum configurations
     + Strict Quorum
       - In a strict quorum configuration, a fixed number of nodes must be present and participate in the decision-making process for a quorum to be achieved.
     + Sloppy Quorum
       - In a sloppy quorum configuration, the required number of participating nodes can vary dynamically based on the current state of the system. A sloppy quorum allows for flexibility in the number of participating nodes, allowing decisions to be made even when a fixed quorum is not available due to node failures or network partitions.
8. Hinted Handoff
   * Hinted handoff is a technique used in distributed databases and storage systems to ensure high availability and fault tolerance in the case of node failures or network partitions. when a node is unreachable, a neighbouring node can accept writes on its behalf. The write is then kept in a local buffer and sent out once the destination node is reachable again. Apache Cassandra uses it.
   * <https://medium.com/double-pointer/replication-for-system-design-interview-11-quorum-variations-1972bb4df42f>
9. Communication in Distributed Systems
   * Gossip protocol
     + It is a peer-to-peer communication mechanism in which nodes periodically exchange state information (timestamps, node identifiers, version numbers, checksums (markle hash), heartbeat, routing information) with a small subset of other nodes, known as neighbours or peers. Upon receiving new information, nodes propagate it to their neighbours and so on.
     + Not all nodes talk directly to all other nodes. Instead, each node communicates with a subset of other nodes, and through the iterative exchange and propagation of information, the protocol ensures that eventually, most or all nodes receive the information.

[](http://pdfcoffee.com_system-design-interview-an-insiders-guidepdf-pdf-free.pdf)

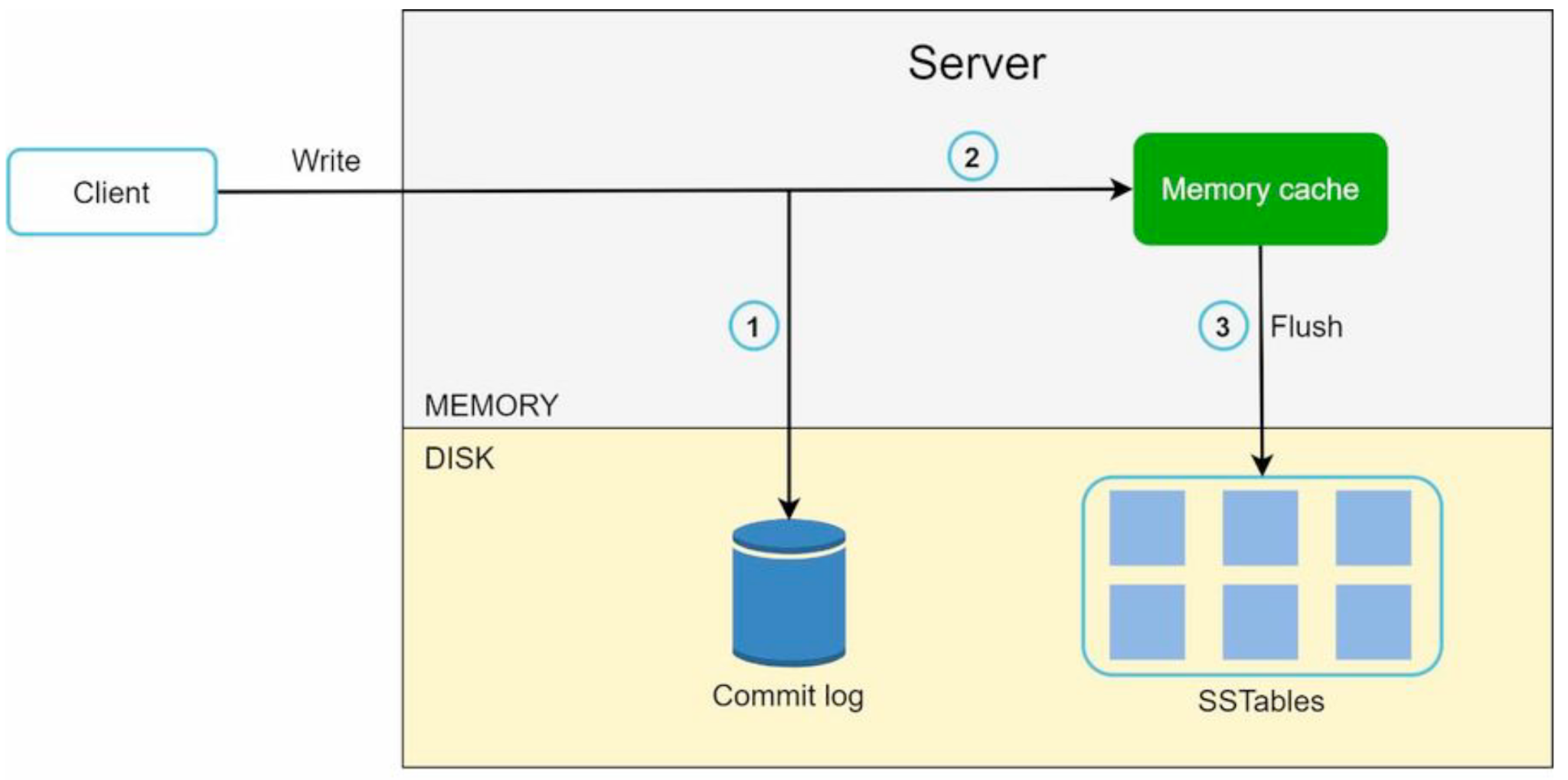
1. Commit Log
   * “Commit Log” logs all write operations before they are applied to the actual data store or Cache. The commit log is typically stored on disk and is append-only, meaning new write operations are appended to the end of the log file. In the event of a crash or failure, the commit log is used for recovery to replay the write operations and bring the database back to a consistent state.
2. Write Ahead Log (WAL)
   * The write-ahead log is a specific type of commit log used in database systems such as PostgreSQL, SQLite, Oracle Database, and Microsoft SQL Server etc.
   * When a transaction or operation modifies data in the system, instead of directly updating the main data store (such as a database), the changes are first recorded in a log file known as the Write-Ahead Log.
   * Write-ahead logs are typically written sequentially (appended to the end of the log file), which allows for efficient disk I/O operations.
   * This log file is usually stored on durable storage, such as disk or SSD, to ensure persistence even in the event of system failure. If the system crashes or experiences a failure, the changes can be replayed from the log to restore the data to a consistent state.
   * Additionally, Write-Ahead Logs are often used for replication purposes, allowing changes to be replayed on secondary replicas to keep them synchronized with the primary data store.
3. Memtable
   * Memtable is an in-memory data structure used in databases to store writes before they are persisted to disk. When a write operation occurs, the data is first written to a write-ahead log (WAL) for durability. Then, the data is written to the memtable, an in-memory structure (often implemented as a sorted data structure like a red-black tree or a skip list). Once the memtable reaches a certain size or after a certain period, it is flushed to disk as an SSTable (Sorted String Table). Over time, multiple SSTables are created. The compaction process merges these SSTables to reduce the number of files, reclaim space from deleted records, and improve read performance.
   * During read operations, the database needs to check the memtable first for the most recent data. If the requested data is not in the memtable, it then searches the SSTables on disk. Memtable is a fundamental component of log-structured merge (LSM) trees, which are used by many modern databases, such as Apache Cassandra, HBase, and LevelDB.

[](https://anyview.fun/2022/09/23/%E4%B8%80%E6%96%87%E4%BA%86%E8%A7%A3lsm-tree/)

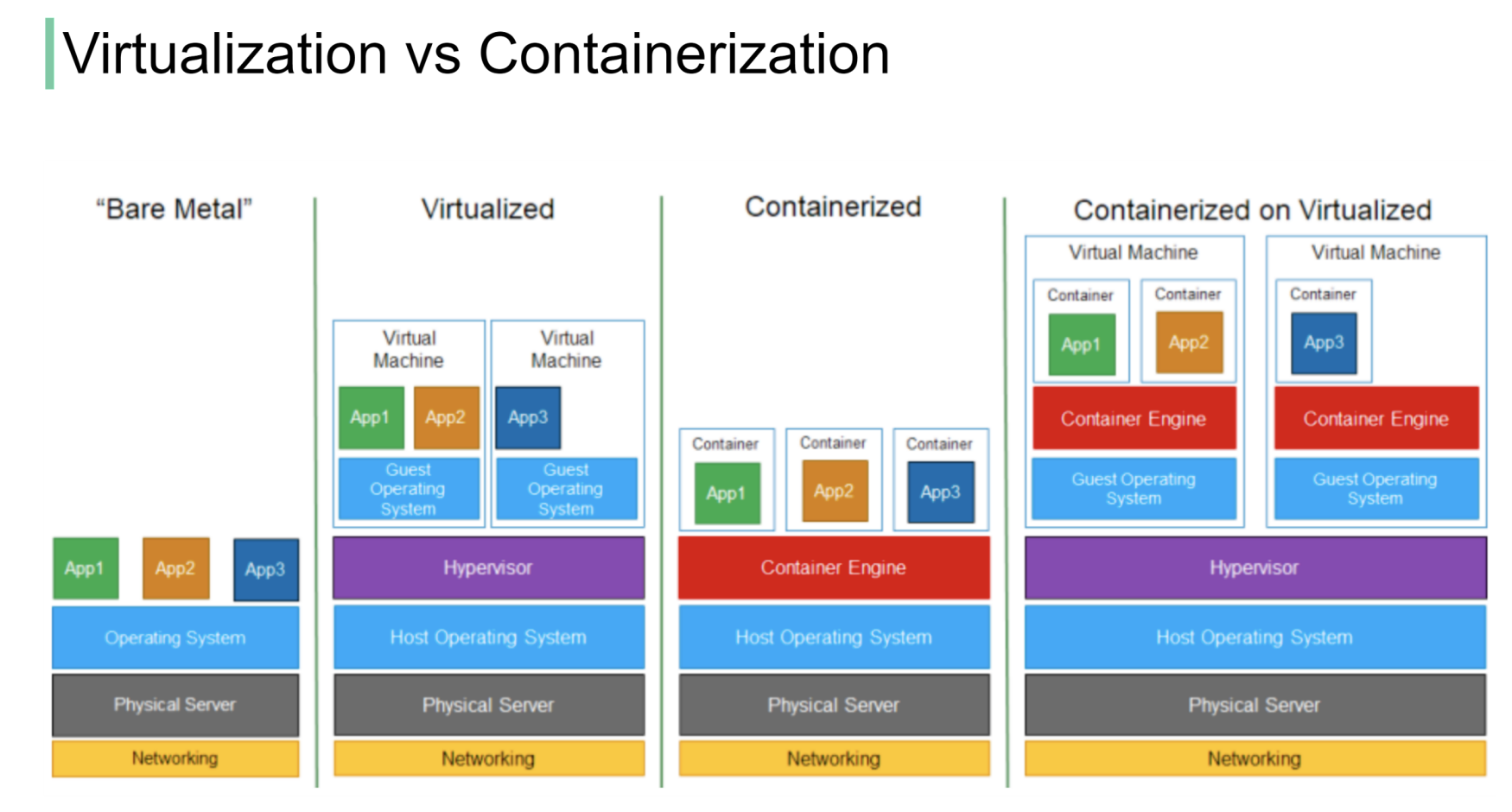
1. Log-structured merge tree (LSM tree)
   * An LSM tree is a data structure used for storing and managing key-value pairs, commonly employed in databases and storage systems and is at the heart of most storage systems. LSM trees handle both write and read operations efficiently with in-memory and disk-based storage structures.
   * LSM trees organize data into multiple levels with a sequence of increasingly larger, sorted components. This merge process periodically consolidates data into structures, facilitating faster lookups and efficient write operations.
   * Who uses it: Cassandra, Kafka, HBase.

[](https://www.scylladb.com/glossary/log-structured-merge-tree/)

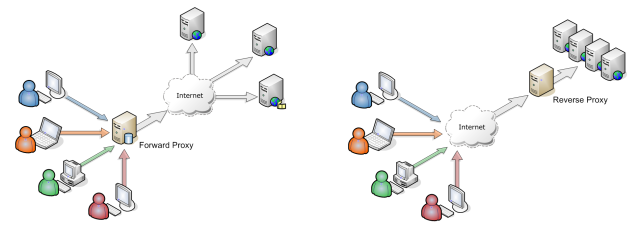
1. SSTable (Sorted String Tables)
   * SSTable is a data structure used in distributed databases, particularly in systems that implement the Log-Structured Merge-Tree (LSM tree) storage engine.
   * Each SSTable is typically stored as a file on disk and the data within an SSTable is organized into blocks, with each block containing a range of contiguous key-value pairs. These blocks are often of fixed or variable size and are used to efficiently store and retrieve data. SSTables may include metadata such as file format version, bloom filters, and index structures to accelerate key lookups.
   * Within each block, the key-value pairs are sorted by key in ascending order, which enables efficient read operations, including range queries and point lookups. Each SSTable typically represents a snapshot of the database state at a specific point in time.
   * SSTables are immutable, meaning once written, they cannot be updated or modified. They have fixed size and once full, new versions of SSTables are created for each write operation. For updates and deletions, SSTables use a technique called "compaction" to merge multiple SSTables and remove obsolete or overwritten data.

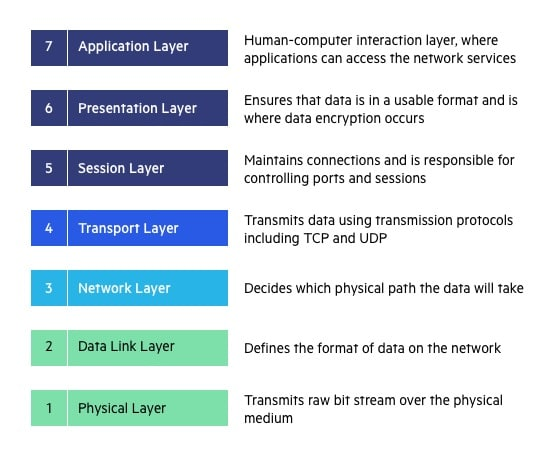
[](http://pdfcoffee.com_system-design-interview-an-insiders-guidepdf-pdf-free.pdf)

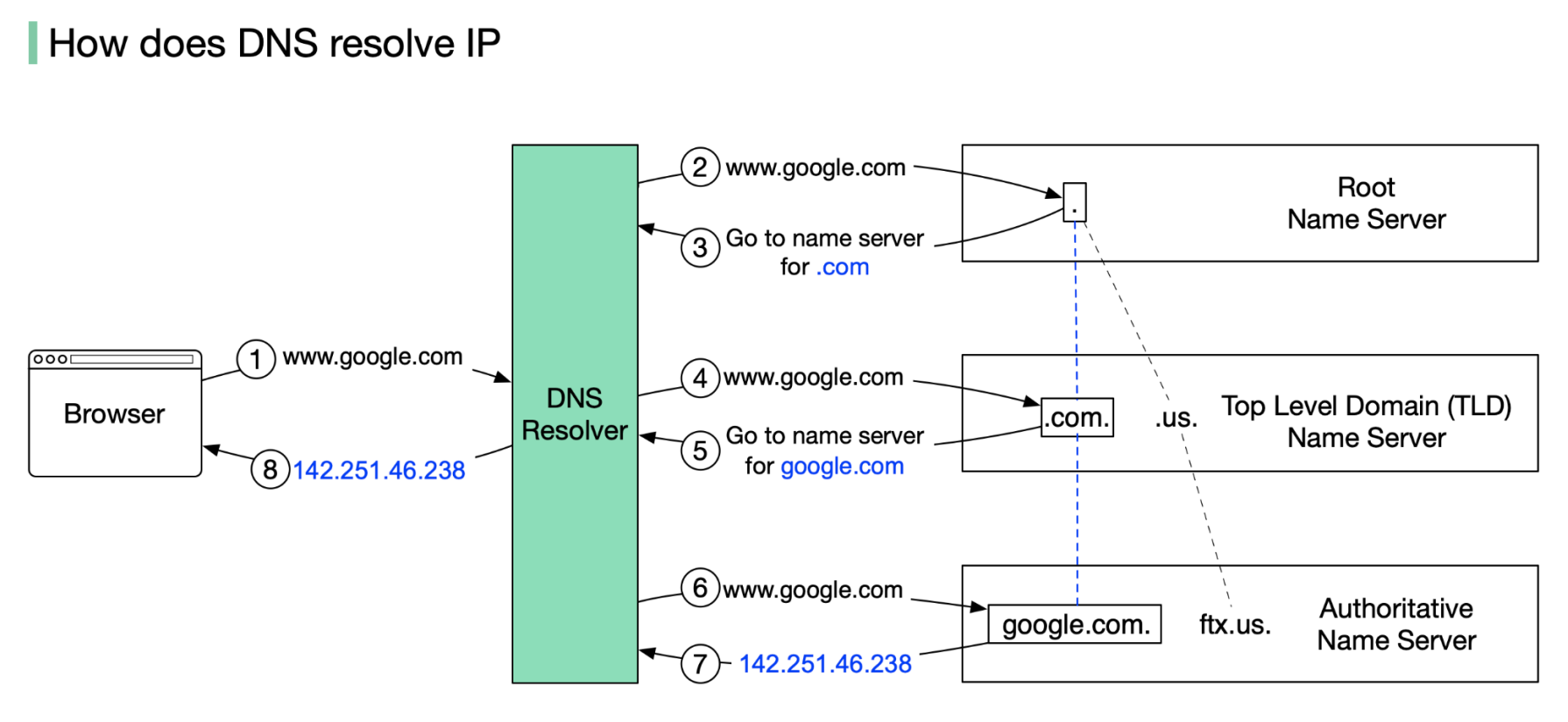
1. Merkle Tree (Binary Hash Tree)
   * A Merkle tree is a binary tree of hashes, where each internal node is the hash of its two children, and each leaf node is a hash of a portion of the original data.
   * Algorithm
   * At the bottom level of the tree are the leaf nodes, which represent individual blocks of data. Each leaf node contains the hash value of a single data block. Internal nodes are calculated recursively by hashing pairs of child nodes until a single root hash, called the Merkle root or root hash, is obtained.
   * Applications
     + Data Replication, Blockchain, Digital Signatures and Certificates,
   * <https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain>
2. B-Tree and B+ Tree
   * B-Tree and B+ Tree are balanced tree and in-memory data structures commonly used in databases and file systems for efficient indexing and retrieval of data.
   * Applications
     + Database Indexing, Storage Systems, File Systems Storage, Cache Management, Search Engines etc.
   * Example
     + MySQL, MongoDB, Cassandra, Redis, NTFS File System,
   * <https://www.programiz.com/dsa/b-tree>
   * <https://github.com/keyvanakbary/learning-notes/blob/master/books/designing-data-intensive-applications.md#b-trees>
   * Used mostly for all database indexing
3. Quad Tree
   * A Quadtree is used to represent spatial data efficiently. It recursively divides a 2D space into four quadrants until each division meets a specified criterion.
   * Applications
     + Image Compression, Spatial Indexing
   * Example
     + Google Map, Apple Map etc
4. Type of ACID Transaction
   * Single-row ACID transactions
   * Single-shard ACID transactions
     + Single-shard ACID occurs when all the rows involved in the transaction’s operations are located in a single shard of a distributed database. Since a single shard is always located inside a single node (or server), single-shard ACID doesn’t involve coordinating transaction operations across multiple nodes. Therefore, it is easier to implement in a distributed database.
   * Distributed ACID transactions
     + Distributed ACID transactions are transactions that impact a set of rows distributed across shards on multiple nodes distributed across a data center, region, or the world. Shards are spread across nodes when the cluster
5. Distributed Transaction
   * In distributed transactions, ACID properties must be adhered to.
   * Algorithms
   * Two-Phase Commit (2PC)
     + In the 2PC protocol, transactions are coordinated by a central coordinator, which initiates the transaction and coordinates commit or abort decisions among all participating resources.
     + First Phase (Voting/Prepare Phase)
       - A coordinator node will ask all the participants node whether they are ready to commit. If all participants vote "yes," the coordinator proceeds to the second phase. If any participant votes "no," the coordinator instructs all participants to abort.
     + Second Phase (Commit Phase)
       - The coordinator instructs all participants to either commit or abort the transaction based on the results of the first phase.
     + <https://www.educative.io/answers/what-is-the-two-phase-commit-protocol>
     + What happens if the coordinator fails before sending the commit/abort request?
       - All the participants can detect the failure using timeouts or heartbeats and can elect a new coordinator node or abort the transaction and release any locks or resources held.
     + What happens if a participant fails in the 2nd phase or a network partition occurs?
       - It may not receive the commit request from the coordinator. The coordinator can handle this by setting a timeout and considering non-responsive participants as failed. The coordinator can then decide to either resend the commit request or abort the transaction.
   * Three-phase commit (3PC)
     + 3PC is an extension of the 2PC protocol that adds an additional phase to address certain edge cases and improve fault tolerance.
     + First Phase (CanCommit Phase)
       - Similar to the first phase of 2PC.
     + Second Phase (PreCommit Phase)
       - The coordinator informs the participants that they will soon be asked to commit the transaction. Participants acknowledge the PreCommit message, indicating their readiness to commit. The purpose of this phase is to prepare participants for the commitment decision and to ensure that they are ready to commit when asked in the next phase. ACK is expected from all coordinator nodes.
     + Third Phase (DoCommit Phase)
       - Similar to the second phase of 2PC.
   * Try-Confirm/Cancel (TC/C)
     + In the TCC protocol, transactions are decentralized, and each resource or service manages its part of the transaction independently.
     + Try Phase
       - Similar to the first phase of 2PC.
     + Confirm Phase
       - The coordinator sends a commit message to all participants who responded positively during the try phase. Participants apply the commit and acknowledge back to the coordinator.
     + Cancel Phase
       - If any participant responds negatively during the try phase or if the coordinator detects failures during the confirm phase, it initiates the cancel phase. The coordinator sends abort messages to all participants, instructing them to roll back the transaction.
   * Saga Pattern
     + It’s the de-facto standard in microservices architecture. A saga is composed of a series of local transactions, each of which updates data within a single service or component. These local transactions are organized into a sequence of steps that collectively represent the entire business transaction. The Saga pattern typically follows one of two approaches:
     + Choreography-Based Saga
       - In this, each service participating in the Saga is responsible for coordinating its own local transaction and communicating with other services as needed. Services publish events to a message broker or event bus to notify other services about the outcome of their local transactions.
       - Other services subscribe to these events and react accordingly, either by executing their own local transactions or compensating for failures.
     + Orchestration-Based Saga
       - In this, there is a centralized coordinator (Saga orchestrator) responsible for coordinating the entire Saga. The orchestrator coordinates the execution of each step in the Saga, communicating with individual services to initiate local transactions and handle compensation in case of failures.
       - The orchestrator maintains the state of the Saga and makes decisions based on the outcomes of each step, ensuring that the overall transaction progresses correctly.
     + <https://medium.com/design-microservices-architecture-with-patterns/saga-pattern-for-microservices-distributed-transactions-7e95d0613345>
6. Distributed Consensus
   * Consensus in distributed systems refers to the process by which a group of nodes or processes agree on a single value or a sequence of values despite the presence of faulty nodes, message delays, and network partitions.
   * Consensus is typically required in distributed systems for tasks such as leader election, distributed locking, atomic commit, and replicated state machine replication.
   * Algorithms
   * Paxos Consensus
     + Phase 1 (Prepare)
       - A proposer selects a proposal value (also called a ballot number) and sends a prepared message to a majority of acceptors. Acceptors respond with a promise not to accept any proposal with a lower proposal number and the highest-numbered proposal they have accepted.
     + Phase 2 (Accept)
       - If a proposer receives promises from a majority of acceptors in Phase 1, it can send an accept message to a majority of acceptors. The accepted message includes the proposal identifier and the proposed value. Acceptors accept the proposal if it has the highest proposal number they have seen.
     + Phase 3 (Learn)
       - Once a proposer receives acceptance messages from a majority of acceptors, the value is considered chosen. The proposer informs all nodes or processes of the chosen value.
     + <https://medium.com/designing-distributed-systems/paxos-a-distributed-consensus-algorithm-41946d5d7d9>
     + <https://www.scylladb.com/glossary/paxos-consensus-algorithm/>
     + <https://medium.com/designing-distributed-systems/flexible-paxos-relaxing-the-quorum-constraint-89caec294083>
   * Raft Consensus
     + Raft is more popular in practice, especially in open-source projects and modern distributed systems. Raft consensus is used for leader election, transaction support (with 2PC) and data replication. Finally, the Raft consensus algorithm imposes the restriction that only the servers with the most up-to-date data can become leaders.
     + All nodes in a Raft cluster start in the follower state. Followers listen for messages from other nodes, including heartbeats from the leader. During an election, followers transition to the candidate state.
     + RequestVote RPC
       - The candidate node requests votes from other nodes in the cluster by sending a RequestVote RPC message. The request includes information such as the node's current term, its last log index and its candidate ID.
     + Granting Votes
       - If a follower receives a RequestVote RPC from a candidate with a higher term than its own, it updates its own term, transitions to the follower state, and grants its vote to the candidate. A node can vote for only one candidate per term. If a follower has already voted for a candidate in the current term, it will reject subsequent RequestVote RPCs.
     + Election Majority
       - A candidate becomes the leader if it receives votes from a majority of nodes in the cluster for the current term. Once a candidate receives votes from a majority of nodes, it transitions to the leader state and it periodically sends heartbeat messages to followers to maintain its leadership status and to inform followers of its current term and commit index.
     + <https://www.yugabyte.com/tech/raft-consensus-algorithm/>
7. Thundering Herd Problem
   * It can occur in distributed systems when multiple clients or processes simultaneously request the same resource that has recently become available or has been updated. This simultaneous surge of requests can overwhelm the system, leading to increased latency, resource contention, and potentially system instability.
   * Thundering Herd means a sudden surge of activity or requests directed at a resource or a system
8. Resource Starvation
   * Resource starvation occurs in distributed systems when certain processes or entities are unable to access the resources (CPU, Memory, Network, I/O, Lock etc) they need, leading to degradation or failure of the system.
   * <https://stackoverflow.com/questions/1162587/what-is-starvation>
9. Concurrency Control Problems in Database
   * Dirty Read
     + A transaction(T1) modifies data, and Transaction (T2) reads this data before T1 commits its changes to the database. If T1 rolls back its changes later, T2 will have read data that was never actually committed to the database.
   * Non-Repeatable Reads
     + Non-repeatable reads can occur, where a transaction(T1) might read the same row multiple times but get different results if another transaction(T2) modifies and commits that row in between.
   * Phantom Reads
     + A phantom read occurs when a transaction(T1) retrieves a set of rows that satisfy a certain condition, and then, during the course of the transaction, another transaction(T2) inserts or deletes rows that also meet the condition. Consequently, when the first transaction(T1) retrieves the same set of rows again, it encounters new rows (phantoms) or finds that some rows it previously retrieved are now missing.
     + Phantom - something (as a ghost) that seems to be there but is not real.
   * Lost Update
     + Lost Update occurs in a concurrent access scenario when one transaction (T1) updates a piece of data, and before it commits its changes, another transaction (T2) updates the same data based on the original (stale) version it read. As a result, when T1 commits its changes, they are overwritten by the changes made by T2, leading to the loss of T1's updates.
   * Dirty Write
     + It occurs when one transaction (T1) overwrites uncommitted data modifications made by another transaction (T2).
   * <https://www.scaler.com/topics/dbms/concurrency-control-in-dbms/>
10. Mitigate Concurrency Control Problems in Database
    * Approaches
      + Optimistic Locking
        - Optimistic locking assumes that conflicts between transactions are rare, so it allows transactions to proceed without acquiring locks initially. Before committing, the transaction checks whether any other transaction has modified the data it has read. If conflicts are detected (e.g., data has been updated by another transaction, i.e. version is updated), the transaction aborts and retries the operation.
      + Pessimistic Locking
        - Pessimistic locking assumes that conflicts between transactions are common, so it acquires locks on data items before allowing transactions to proceed. Other transactions may be blocked while a transaction holds locks on data items.
      + <https://stackoverflow.com/questions/129329/optimistic-vs-pessimistic-locking>
    * Deadlock Prevention Strategies
      + Lock Timeout
      + Timeouts for Resource Requests
      + Avoidance of Nested Locks
      + Resource Allocation Graph (RAG)
      + Transaction Rollback
    * Cascading Abort (cascading rollback or cascading cancellation)
      + Cascading abort refers to the automatic rollback or cancellation of dependent transactions when a higher-level transaction encounters an error or is aborted.
    * Algorithms
    * Two-Phase Locking (2PL)
      + It's based on the principles of transactions to acquire and release locks in two phases (growing phase and shrinking phase), ensuring that conflicting operations do not occur concurrently. Deadlock can occur.
      + Growing Phase
        - A transaction can acquire locks on data items as needed in any order but cannot release any locks until they have acquired all the locks they need for their entire transaction. Once a lock is acquired, it cannot be released until the transaction reaches the shrinking phase.
      + Shrinking Phase
        - When all the data changes are done, only then the locks are released. A transaction releases the locks they hold, but they cannot acquire any new locks. Once a lock is released, it cannot be reacquired.
      + 2PL Protocol
        - Shared Lock (i.e. Read Lock)
          * When a shared lock is applied to a data item, multiple transactions can hold the shared lock on that item simultaneously for concurrent read access. A shared lock does not block other shared locks. However, it prevents exclusive locks from being acquired on the same data item.
        - Exclusive Lock (i.e. Write Lock)
          * When an exclusive lock is applied to a data item, no other transactions can hold any type of lock (shared or exclusive) on that item. This ensures that no other transaction can read or write the data until the exclusive lock is released.
      + <https://www.scaler.com/topics/two-phase-locking-protocol/>
    * Timestamping
      + Assign a unique timestamp to each transaction when it begins execution. Use timestamps to determine the order of transaction execution and resolve conflicts based on timestamp ordering.
      + Timestamping Protocol
        - Timestamp Ordering Protocol
    * Multi-Version Concurrency Control (MVCC)
      + MVCC maintains multiple versions of each data item in the database to support concurrent read and write operations. When a transaction updates a data item, instead of overwriting the existing value, MVCC creates a new version of the data item with the updated value and timestamps it with the transaction's start time.
      + The DBMS ensures that the new version is not visible to other transactions until the updating transaction commits. This allows other transactions to continue reading the old version of the data item until the updating transaction completes. MVCC employs garbage collection mechanisms to remove obsolete versions of data items that are no longer needed.
11. Locality of reference
    * It refers to the tendency of programs to access data and instructions that are close to each other in time or space.
    * Spatial Locality
      + Spatial locality refers to the tendency of an execution process to access data elements whose addresses are close to each other, within a relatively short period. This behaviour is common in typical applications, such as when accessing elements of an array in a loop. Because of spatial locality, modern processors and memory architectures use caching strategies that, upon accessing a particular data element, also load surrounding data elements into the faster cache memory. This approach optimizes the access times for subsequent accesses to nearby data elements.
    * Temporal Locality
      + Temporal locality, on the other hand, describes the tendency of a process to access the same set of data or instructions repeatedly over a short period of time. An example of temporal locality is a loop that accesses the same variables repeatedly. Caching mechanisms leverage temporal locality by keeping recently accessed data in cache memory for some time, under the assumption that this data is likely to be accessed again soon. This reduces the need to access slower main memory and speeds up the execution time.
    * <https://stackoverflow.com/questions/7638932/what-is-locality-of-reference>
12. Distributed Denial of Service (DDoS)
    * In a DDoS attack, multiple compromised computers or devices, often referred to as "bots" or "zombies," are used to generate or amplify the attack traffic, overwhelming the target’s resources such as bandwidth, CPU, memory, or network connections.
    * To defend against DDoS attacks, organizations often use various mitigation techniques such as traffic filtering, rate limiting, IP whitelisting, blackholing, and deploying specialized DDoS protection services or appliances.
13. Probabilistic Data Structures (PDS)
    * False Positive (FP)
      + False - The model got it wrong.
      + Positive - The model incorrectly identified something as positive.
      + Thus the model falsely identified something as positive when it was actually negative.
    * False Negative (FN)
      + False - The model got it wrong.
      + Negative - The model incorrectly identified something as negative.
      + Thus the model falsely identified something as negative when it was actually positive.
    * True Positive (TP)
      + True - The model got it right.
      + Positive - The actual result was positive.
      + Thus the model correctly identified something as positive.
    * True Negative (TN)
      + True - The model got it right.
      + Negative - The actual result was negative.
      + Thus the model correctly identified something as negative.
    * Bloom Filters
      + The Bloom filter is a data structure that efficiently determines whether an element may be a member of a set, with a small probability of false positives, or definitely is not a member.
      + Algorithm
        - A Bloom filter is typically initialized as an array of bits, all initially set to 0. Several independent hash functions are chosen. Each hash function maps input elements to positions in the bit array.
        - To insert an element into the Bloom filter, it is hashed by each of the hash functions, and the corresponding positions in the bit array are set to 1.
        - To check if an element is in the set, it is hashed by the same hash functions. If any of the corresponding bit positions in the array are 0, the element is definitely not in the set. If all the corresponding bit positions are 1, the element is probably in the set, but there's a chance of false positives.
      + Example
        - CDNs, Database Systems, Search engines, Network routers, Google Chrome (to spot malicious URLs).
      + <https://blog.bytebytego.com/p/how-to-avoid-crawling-duplicate-urls>
      + <https://www.educative.io/answers/what-is-a-bloom-filter>
      + <https://systemdesign.one/bloom-filters-explained/>
      + <https://github.com/roottraveller/DSAlgo-HowToDoIt-learningcode/blob/master/java-code/src/main/java/BloomFilterDemo.java>
      + [Bloom Filters | Algorithms You Should Know #2 | Real-world Examples](https://www.youtube.com/watch?v=V3pzxngeLqw&list=PLCRMIe5FDPseVvwzRiCQBmNOVUIZSSkP8&index=3)
    * HyperLogLog (HLL)
      + HyperLogLog is a probabilistic data structure used for estimating the number of unique elements in very large datasets or data streams with high accuracy and minimal memory usage.
      + Algorithm
      + // Understand todo
      + <https://stackoverflow.com/a/35219704>
      + <https://www.waitingforcode.com/big-data-algorithms/hyperloglog-explained/read>
      + Example
        - Counting Unique Visitors on a website, routers, etc
    * Count-Min sketch
14. Cloud Computing
    * Types of Cloud Services
      + IaaS (Infrastructure-as-a-Service) - AWS EC2
      + PaaS (Platform-as-a-Service) - AWS Elastic Beanstalk
      + SaaS (Software-as-a-Service) - Microsoft Office 365
      + FaaS (Function-as-a-Service) or Serverless - AWS Lambda
    * <https://www.scaler.com/topics/cloud-computing/introduction-to-cloud-computing/>
    * <https://www.linkedin.com/pulse/iaas-vs-paas-whats-difference-christian-camacho/>
15. Bare Metal vs Virtualization (VMware) vs Containerization (Docker)
    * <https://www.redhat.com/en/topics/virtualization>
    * <https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization>

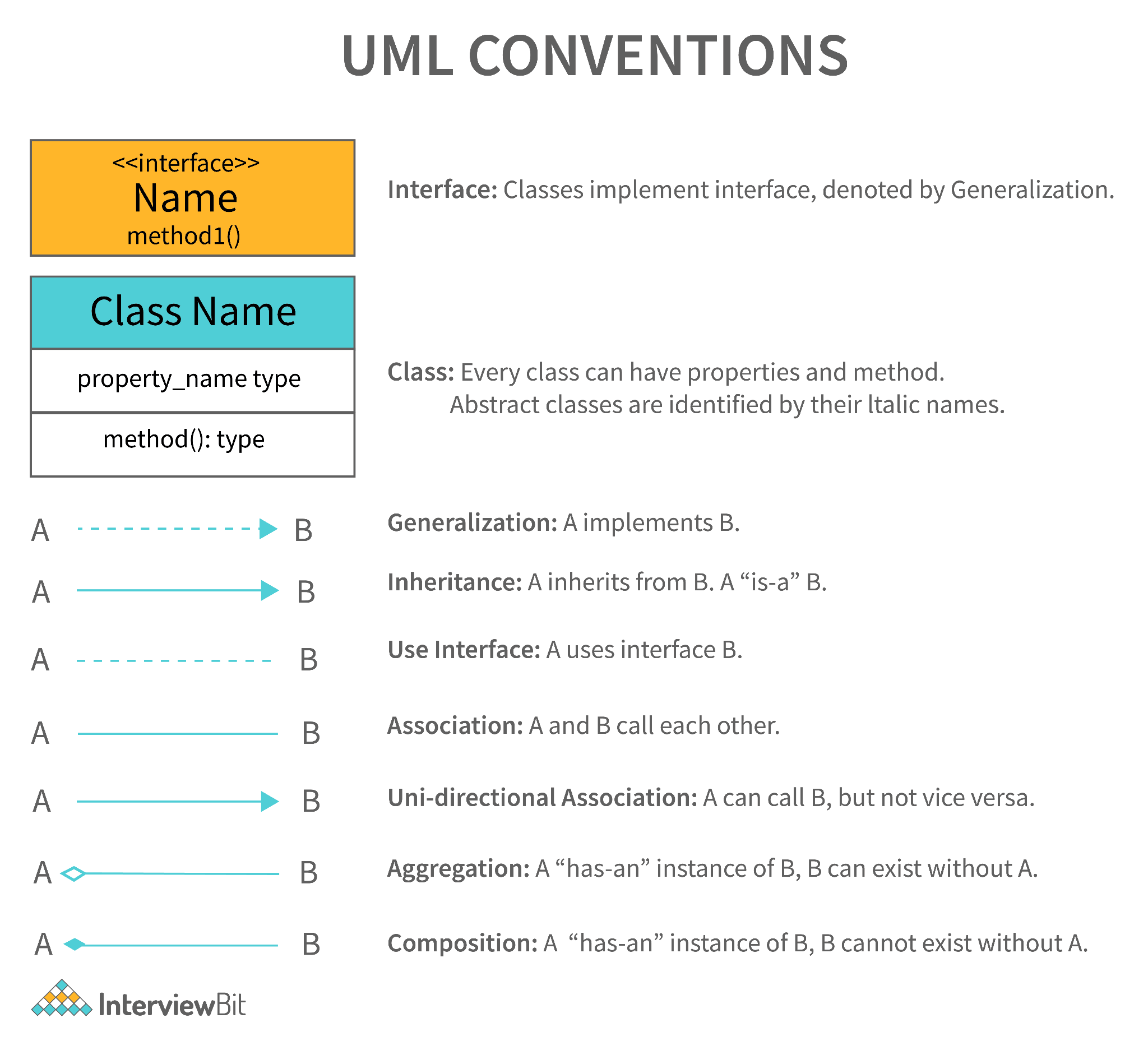
[](https://blog.bytebytego.com/p/what-are-the-differences-between)

1. Types of Databases
   * Relational database
     + MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, SQLite, MariaDB, Amazon RDS
   * NoSQL databases
     + Key-Value Store
       - Redis, Aerospike, Memcached, Amazon ElastiCache, Amazon DynamoDB
     + Document Store
       - MongoDB, CouchDB, Amazon DocumentDB
     + Column Store
       - Apache Cassandra, HBase, Google Bigtable, Amazon Redshift
     + Graph-Based
       - Neo4j
   * <https://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis/>
2. Cache Strategies
   * Reading data from Cache
   * Cache-Aside (Lazy Loading)
     + The application code is responsible for managing the cache. If the data is found in the cache, it's cache hit and data is returned to the application. If not, its cache miss, the application fetches the data from the underlying data store, stores it in the cache for future use, and then returns it to the application.
   * Read-Through
     + The cache itself is responsible for managing the cache and automatically fetching data from the underlying data store when it's not found in the cache. When the application requests data, the cache checks if it's available. If not, it automatically fetches it from the data store, populates the cache, and then returns the data to the application.
   * Writing data to Cache
   * Write-Around
     + The data is written directly to the underlying data store (e.g., SSD) , bypassing the cache. This strategy is useful for write-heavy workloads.
   * Write-Through
     + The data is written to both the cache and the underlying data store synchronously. This ensures that the data in the cache is always consistent with the data in the underlying data store but can introduce additional latency for write operations.
   * Write-Back
     + The data is written to the cache first and then asynchronously written to the underlying data store. This can improve write performance since writes are acknowledged as soon as they're written to the cache, but there's a risk of data loss if the cache fails before data is written to the data store. write-ahead-log (WAH) can be used for this.
   * Write-Behind
     + Similar to Write-Back, but the actual write to the data store is deferred until a later time, often during periods of low activity, and may involve batching or aggregating data before writing it to the data store.
   * <https://codeahoy.com/2017/08/11/caching-strategies-and-how-to-choose-the-right-one/>
   * <https://blog.bytebytego.com/p/top-caching-strategies>
   * Cache Eviction Policies/Algorithms
     + Least Recently Used (LRU)
     + Most Recently Used (MRU)
     + Least Frequently Used (LFU)
     + First In First Out (FIFO)
     + Last In First Out (LIFO)
     + Random Replacement (RR)
3. Network Architectures
   * Peer-to-Peer (P2P) Connection
   * Client-Server Architecture
   * Distributed Systems
   * Message-Oriented Middleware (MOM)
   * Hybrid Architectures
4. CORS (Cross-Origin Resource Sharing)
   * Web browsers enforce a security policy known as the Same-Origin Policy, which restricts web pages from making requests to resources (e.g., APIs, images, scripts) on domains other than the one that served the web page.
   * To enable cross-origin (i.e., CDN) requests, the server must include specific CORS headers in its response to indicate which origins are allowed to access the requested resource.
   * <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
5. Network Topology
   * Bus Topology
   * Star Topology
   * Ring Topology
   * Mesh Topology
   * Hybrid Topology
   * <https://www.studytonight.com/computer-networks/network-topology-types>
6. Switch vs Router
   * A switch is a layer 2 (Data Link layer) device that forwards data packets between devices within a LAN based on their MAC (Media Access Control) addresses. They use MAC address tables to learn which devices are connected to which switch ports and make forwarding decisions accordingly.
   * A router is a layer 3 (Network layer) device that connects multiple LANs or WANs networks and forwards data packets based on routing tables between them based on their IP (Internet Protocol) addresses.
7. Proxy Servers
   * A proxy server is an intermediate piece of software or hardware that sits between the client and the server, facilitating communication between them.
   * Forward Proxy
     + Forward proxy hides the client’s identity.
   * Reverse Proxy
     + Reverse proxy hides the server’s identity.
   * <https://www.scaler.com/topics/proxy-servers-and-web-caching/>
   * <https://stackoverflow.com/questions/224664/whats-the-difference-between-a-proxy-server-and-a-reverse-proxy-server>

[](https://github.com/dariubs/awesome-proxy/blob/master/readme.md)

1. Key Exchange Algorithms
   * Key exchange is the process by which cryptographic keys (private key) are securely exchanged between parties to enable secure communication and establish a shared secret key for encryption and decryption over an insecure communication channel
   * Diffie-Hellman Key Exchange
   * RSA Key Exchange
   * <https://www.vskills.in/certification/tutorial/private-and-public-key-exchange-3/>
2. Unicode Characters
   * Typically represented in hexadecimal format. The length of a Unicode character representation in bits depends on the encoding scheme used.
   * UTF-8: 8 bits (1 byte) per character, U+0000 to U+007F (127/2^8 - 1)
   * UTF-16: 16 bits (2 bytes) per character, U+0000 to U+FFFF (65535/2^16−1)
   * UTF-32: 32 bits (4 bytes) per character, U+0000 to U+10FFFF (4294967295/2^32-1)
   * <https://symbl.cc/en/unicode-table/#devanagari>
3. OSI (Open Systems Interconnection) Model
   * <https://www.scaler.com/topics/computer-network/osi-model/>
   * <https://www.geeksforgeeks.org/layers-of-osi-model/>   
     [](https://medium.com/@varkeythms/tcp-ip-and-osi-model-2a93f315b253)
4. TCP/IP Model
   * <https://www.scaler.com/topics/computer-network/tcp-ip-protocol-suite/>
   * <https://www.geeksforgeeks.org/tcp-ip-model/>
5. DNS (Domain Name System)
   * DNS Server
     + Resolve FQDN to IP address through DNS recursive resolver and authoritative DNS server. The authoritative server responds with the IP address associated with the FQDN.
   * DNS Load Balancer
     + If DNS load balancing is part of the domain's configuration, the authoritative DNS server will return multiple IP addresses, allowing the client to connect to one of the load-balanced servers. Each client might receive a different order of IP addresses based on Round Robin. the client chooses the first IP address from the list.
   * Type of ​​DNS Servers
     + Root DNS Servers
     + TLD (top-level domain) Name Servers
     + Authoritative Name Servers
     + Recursive DNS Servers
   * CNAME (Canonical Name)
     + It is a type of DNS resource record that allows multiple domain names to resolve to the same IP address.
   * GeoDNS (Geographic Domain Name System)
     + It is a DNS service that allows domain names to be resolved to IP addresses based on the location of a user. This is useful in case of multi DCs.
   * DNS Resolution with CDN
     + When a user requests a URL, the DNS server may be configured to return the IP address of the CDN edge server closest to the user instead of the origin server's IP address. This is accomplished through DNS-based global server load balancing techniques. The DNS server is typically configured with multiple IP addresses associated with the CDN edge servers.

[](https://blog.bytebytego.com/p/how-does-the-domain-name-system-dns)

1. HTTP vs HTTPS
   * HTTP requests and responses are typically sent and received over the network as plain text. The content of HTTP messages, including headers and message bodies, is human-readable and not encrypted by default. It typically uses port 80 for communication.
   * HTTPS is essentially HTTP, but secure. Before the actual data transfer begins, the client and server perform a TLS handshake. This handshake involves the server sending its SSL certificate to the client to verify the server's identity. The certificate contains the server's public key and is issued by a trusted Certificate Authority (CA). It uses port 443 for communication.
   * <https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/>
   * <https://blog.bytebytego.com/p/how-https-works-youtube-diagram-as>
2. HTTP vs HTTP2
   * HTTP 1.0 (1996)
     + Each request to the server required a new TCP connection. No persistent connections were allowed, resulting in increased latency and overhead due to repeated connection establishment.
   * HTTP 1.1 (1997)
     + Introduced persistent connections, allowing a single TCP connection to be reused for multiple requests. However, it did not fully address HOL (Head-of-Line) blocking issues, as subsequent requests still had to wait for earlier ones to complete on the same connection. HTTP is primarily unidirectional with workarounds for bidirectional communication,
   * HTTP 2.0 (2015)
     + Addressed HOL blocking through request multiplexing, allowing multiple HTTP exchanges (streams) to be multiplexed onto a single TCP connection. This eliminated HOL blocking at the application layer by enabling parallel processing of requests within the same connection. However, HOL blocking still persisted at the transport (TCP) layer, as a single slow request could delay the transmission of other requests on the same connection. HTTP/2 is designed to support native bidirectional communication
   * HTTP 3.0 (2020, draft):
     + Proposed successor to HTTP 2.0, aiming to further improve performance and address HOL blocking issues. Utilizes QUIC (Quick UDP Internet Connections) instead of TCP as the underlying transport protocol. QUIC operates at the transport layer and is designed to provide better performance, particularly for connections with high latency or packet loss.
   * <https://blog.bytebytego.com/p/http1-vs-http2-vs-http3-a-deep-dive>
3. TCP (Ack needed) vs. UDP
   * <https://www.educative.io/answers/tcp-vs-udp>
   * <https://www.scaler.com/topics/computer-network/tcp-vs-udp/>
4. HTTP Status Code
   * <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
5. HTTP Headers
   * <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>
   * ETag (Entity Tag) HTTP Header
     + ETag header is used by web servers and browsers to determine whether a resource (e.g., web pages, images, CSS files) has changed since it was last requested or cached.
     + Example - ETag: "686897696a7c876b7e"
     + <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag>
     + <https://www.educative.io/answers/what-is-an-etag-in-http>
   * Authorization HTTP Header
     + Authorization: <scheme> <credentials>
     + <scheme> specifies the authentication scheme being used e.g., "Basic", "Bearer", "Digest".
     + Basic
       - In Basic Authentication, the <credentials> consist of the base64-encoded(username:password).
       - Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
     + Bearer
       - In Bearer Authentication, the <credentials> consist of a token (JWT) provided by the client.
       - Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c
     + Digest
       - In Digest Authentication, the <credentials> contain various pieces of information, including the username, realm, nonce, URI, response, and other parameters.
       - Authorization: Digest username="username", realm="example", nonce="abc123", uri="/resource", response="d41d8cd98f00b204e9800998ecf8427e"
6. How Does HTTP Work?
   * <https://www.scaler.com/topics/computer-network/hypertext-transfer-protocol/>
7. URL vs URI vs URN
   * URI (Uniform Resource Identifier)
     + Identify a resource, either by its location, name, or both. Example, <mailto:user@example.com>, file:///home/user/file.txt
   * URL (Uniform Resource Locator)
     + locate a resource on the internet. It includes the complete address for locating a resource, typically consisting of a protocol, domain name, path, and possibly other components. Example, <http://example.com/mypage.html>, ftp://[example.com/download.zip](http://example.com/download.zip)
   * URN (Uniform Resource Name)
     + Identifies a resource by a unique and persistent name, but doesn't necessarily tell you how to locate it on the internet. Example
     + urn:isbn:0451450523 to identify a book by its ISBN.
     + urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66 a globally unique identifier.
     + urn:publishing:book - An XML namespace that identifies the document as a type of book.
8. BGP (Border Gateway Protocol)
   * It is the protocol that routers use to make decisions about how to route packets between different networks.
   * <https://www.scaler.com/topics/computer-network/bgp-border-gateway-protocol/>
9. Inside Look at Modern Web Browser
   * <https://developer.chrome.com/blog/inside-browser-part1/>
   * <https://developer.chrome.com/blog/inside-browser-part2/>
   * <https://developer.chrome.com/blog/inside-browser-part3/>
   * <https://developer.chrome.com/blog/inside-browser-part4/>
10. Checksum
    * <https://www.scaler.com/topics/computer-network/checksum/>
11. Serialization and Deserialization
    * Serialization refers to the process of converting data objects from their in-memory representation into a byte stream that can be transmitted over a network or stored in a file or database
    * Deserialization is the reverse process of serialization. It involves converting a byte stream back into its original in-memory data object.
    * Compression, encryption, and decryption are additional steps that can be applied on top of serialization and deserialization, depending on the specific requirements of the application.
    * Serialization Algorithms
      + Binary Serialization/Deserialization
      + JSON Serialization/Deserialization
      + XML Serialization/Deserialization
      + Protocol Buffers (protobuf) Serialization/Deserialization
12. Encoding and Decoding
    * Encoding is used to represent data in a specific format that is suitable for a particular purpose, such as storage, transmission, or presentation.
    * Encoding and Decoding are similar concepts to serialization and deserialization. However, serialization and deserialization are commonly used in software development to convert complex data structures or objects into a linear format.
    * Encoding Algorithms
      + Base64 Encoding/Decoding
      + UTF-8 Encoding/Decoding
      + UTF-16 Encoding/Decoding
13. Compression and Decompression
    * Before serialization, data can be compressed to reduce its size, which can lead to more efficient transmission over the network or storage in files or databases.
    * Algorithms
      + Huffman Coding
      + Run-Length Encoding (RLE)
    * Compression Library
      + Gzip
      + Snappy
14. Encryption and Decryption
    * To ensure data security during transmission or storage, encryption can be applied either before or after serialization. Encryption scrambles the data using a cryptographic algorithm and a key, making it unreadable to unauthorized parties.
    * When encrypted data is received or retrieved, it needs to be decrypted before deserialization to restore it to its original form. Decryption requires the appropriate cryptographic key and algorithm to reverse the encryption process.
    * Encryption Algorithms
      + RSA (Rivest-Shamir-Adleman)
      + AES (Advanced Encryption Standard)
      + Diffie-Hellman
      + ECDHE
    * Types of Encryption
      + Symmetric Encryption
        - A single key is used for both encryption and decryption.
      + Asymmetric Encryption (Public Key Encryption)
        - A public key for encryption and a private key for decryption is used.
15. Fragmentation and Defragmentation
    * Fragmentation
      + Fragmentation occurs when files are stored in non-contiguous blocks of memory or disk space.
      + Internal Fragmentation
      + External Fragmentation
    * Defragmentation
      + Defragmentation rearranges fragmented data on the disk so that files are stored contiguously in sequential blocks. This reduces seek times and improves data access speed and increasing storage capacity efficiency by minimizing wasted space.
      + File System Defragmentation
      + Disk Defragmentation
    * <https://www.scaler.com/topics/fragmentation-in-os/>
16. UML Diagram  
    
17. Paging in OS
    * <https://www.scaler.com/topics/paging-in-os/>
    * <https://www.scaler.com/topics/memory-management-in-operating-system/>
    * <https://www.scaler.com/topics/operating-system/virtual-memory-in-os/>
18. Ephemeral Port
    * Ephemeral ports are allocated dynamically by the operating system when a client application initiates a connection to a server. These are intended for temporary usage and are typically associated with short-lived communication sessions. Once the communication session is terminated, the ephemeral port is released back to the pool of available ports for reuse.
    * Ports with numbers [0-1023] are called systems or well-known ports. These ports are reserved for system services and well-known applications and are standardized by the Internet Assigned Numbers Authority (IANA).
      + Port 80: HTTP (Hypertext Transfer Protocol)
      + Port 443: HTTPS (HTTP Secure)
      + Port 8080: HTTP Alternate - Often used as a secondary HTTP port.
      + Port 8443: HTTPS Alternate - Often used as a secondary HTTPS port.
      + Port 21: FTP (File Transfer Protocol)
      + Port 22: SSH (Secure Shell)
      + Port 23: Telnet
      + Port 25: SMTP (Simple Mail Transfer Protocol)
      + Port 53: DNS (Domain Name System)
      + Port 110: POP3 (Post Office Protocol v3)
      + Port 123: NTP (Network Time Protocol)
      + Port 143: IMAP (Internet Message Access Protocol)
    * Ports with numbers [1024-49151] are called user or registered ports. They are not standardized by IANA but are assigned by system administrators or software developers for specific applications.
      + Example: A gaming company selects a port range within the user or registered ports range (1024-49151) for their game. Let's say they choose a port range 30000-30010 for their game servers.
      + Port 3306: Default port for MySQL server
      + Port 9092: Default port for Kafka brokers
      + Port 2181: Default port for Zookeeper
      + Port 6379: Default port for Redis
      + Port 3000: Default port for Aerospike service
    * Ports with numbers [49152-65535] are called dynamic, private or ephemeral ports.
19. QPS vs RPS
    * QPS is often used in the context of database systems or data storage systems. It represents the number of queries or database transactions executed per second.
    * RPS is more commonly associated with web services, APIs, and network-related scenarios. It represents the number of HTTP requests or network transactions processed per second.
20. Sticky Sessions
    * Also known as session affinity or session persistence, are a method used in web server load balancing to ensure that subsequent requests from a particular client are always sent to the same server in a cluster.
    * <https://dev.to/gkoniaris/why-you-should-never-use-sticky-sessions-2pkj>
21. 32-bit vs 64 bit System Architecture
    * In a 32-bit system, memory addresses are represented using 32 bits, which allows for a maximum of 2^32 (approximately 4 GB) unique memory addresses. Therefore, even if a computer is running with more than 4 GB of physical RAM installed, the 32-bit architecture can only address up to 4 GB of RAM directly.
    * In a 64-bit system, memory addresses are represented using 64 bits, which allows for a maximum of 2^64 (approximately 18 EB) unique memory addresses. Therefore, a 64-bit architecture can address significantly more memory compared to a 32-bit architecture. With this increased address space, a computer running a 64-bit architecture can access and utilize much larger amounts of physical RAM, making it suitable for handling larger datasets and memory-intensive applications.
    * Memory Scale
      + Kilobyte (KB): 2^10 bytes (10^3)
      + Megabyte (MB): 2^20 bytes (10^6)
      + Gigabyte (GB): 2^30 bytes (10^9)
      + Terabyte (TB): 2^40 bytes (10^12)
      + Petabyte (PB): 2^50 bytes (10^15)
      + Exabyte (EB): 2^60 bytes (10^18)
      + Zettabyte (ZB): 2^70 bytes (10^21)
      + Yottabyte (YB): 2^80 bytes (10^24)
22. Latency Numbers Every Programmer Should Know
    * 0.5 ns - L1 cache reference
    * 5 ns - Branch mispredict
    * 7 ns - L2 cache reference
    * 100 ns - Mutex lock/unlock
    * 100 ns - Main memory reference
    * 0.01 ms - Compress 1K bytes with Zippy
    * 0.02 ms - Send 2K bytes over 1 Gbps network
    * 0.25 ms - Read 1 MB sequentially from memory
    * 0.5 ms - Round trip within the same datacenter
    * 10 ms - Disk seek
    * 10 ms - Read 1 MB sequentially from network
    * 30 ms - Read 1 MB sequentially from disk
    * 150 ms - Send packet CA->Netherlands->CA
    * <https://colin-scott.github.io/personal_website/research/interactive_latency.html>
23. Deployment strategies
    * Blue/Green Deployment
    * Canary Deployment
24. Chaos Engineering
    * Chaos Engineering involves deliberately injecting failures or adverse conditions into a system in a controlled manner to observe how the system responds. The purpose is to uncover systemic weaknesses that might not be apparent through traditional testing methods.
    * Tools for Chaos Engineering
      + Netflix Chaos Monkey
      + Chaos Toolkit (open source)
      + K8 Litmus
25. Circular Buffer
26. Colocation (i.e. colo)
27. Geocoding
    * Converting addresses into geographic coordinates (latitude, longitude).
28. Geofencing
29. Geohashing
    * Divides the world map into a grid of cells (2X2) until each grid reaches a certain size threshold. This is used for storage purposes. With each divide, we move to the next zoom level. Note, that we can divide the ocean, forest, and mountain grid to only a few zoom levels to save space.
30. Idempotency
    * Idempotency is a property of operations where the result of applying an operation multiple times is the same as applying it once. This property is important in distributed systems, network protocols, and transaction processing to ensure consistency and reliability.
31. BCP/DR (Business Continuity Plan/Disaster Recovery)
    * RTP(Recovery Time Objective) - represents the amount of time an application can be down and not result in significant damage to a business and the time that it takes for the system to go from loss to recovery.
    * RPO(Recovery Point Objective) - is the maximum amount of data that can be lost before causing detrimental harm to the organization.
32. Types of Recovery
    * Backup and Restore
      + Full, incremental, and differential backups for periodic recovery.
    * Log-Based Recovery
      + Use transaction logs to restore to the latest committed state.
    * Point-in-Time Recovery (PITR)
      + Restore to any specific moment within a retention period.
    * Snapshot Recovery
      + Use point-in-time snapshots for quick restoration.
33. Service Discovery
    * Service discovery is a critical component in distributed systems and microservices architectures, facilitating the dynamic and automatic detection of services and their endpoints within a network.
    * Service Registry
      + A service registry is a centralized database or directory that stores information about available services and their locations. It maintains a dynamic and up-to-date catalog of service instances, including their metadata and status. Today, service registry capability sits with load balancer and hence no need to maintain a separate service.
      + Examples are ZooKeeper, etcd, and Netflix Eureka.
    * Types of Service Discovery
      + Client-Side Discovery
        - Client applications are responsible for querying a service registry (i.e. discovery service) to obtain information about available running instances. Service Registry returns one of the instance details and clients use this to interact with the server. Clients typically use libraries or SDKs to interact with the service registry and dynamically discover service endpoints.
        - Examples - Netflix Eureka.
      + Server-Side Discovery
        - In this, a dedicated component or proxy acts as a middleware layer between clients and services, handling service discovery on behalf of clients. Clients communicate with the discovery component or proxy (i.e., Load Balancer) to request information about available service endpoints and discovery component or proxy return one of the instances of service.
        - Examples - NGINX.
      + Service Registry-Based Discovery
        - It relies on a centralized database or directory, known as a service registry, to store and manage information about available service instances. Service instances register themselves with the service registry upon startup or when they become available, and clients query the registry to discover service endpoints.
        - Examples - etcd, ZooKeeper.
      + DNS-Based Discovery
        - DNS-based discovery leverages DNS to resolve service names to their corresponding IP addresses or endpoints. Service instances (i.e., applications) register their information (e.g., IP address, port) as DNS records with a DNS server, which clients query to resolve service names.
        - Examples - Kubernetes DNS-based service discovery.
34. Strangler Pattern
    * It is used in the context of monolith or legacy system migration to microservice. rather than attempting a complete overhaul all at once, we enable traffic migration gradually i.e. 10% then 20% etc. This approach allows for the seamless integration of new microservices while minimizing disruptions to the existing system.
35. Type of Microservice architecture
    * Domain-Driven Design (DDD)
      + In this pattern, microservices are organized around business domains or subdomains. Each microservice is responsible for implementing the business logic and data related to a specific domain such as customer management, product catalog, payment or order processing.
    * Event-Driven Architecture (EDA)
      + Microservices in this pattern communicate with each other asynchronously through events. Events represent state changes or meaningful occurrences within the system, and microservices can publish or subscribe to events to react to changes in real-time.
36. Single Point of Failure (SPOF)
37. Loosely Coupled and Tightly Coupled
38. IP Whitelisting
39. Cross-Site Request Forgery (CSRF)
40. Command Query Responsibility Segregation (CQRS)
    * CQRS is a design pattern that separates the operations of reading data (queries) from writing data (commands) in a system. This separation allows for optimizing each side independently and can lead to better performance, scalability, and maintainability.
41. SLA vs SLO
    * SLO targets specific measurable characteristics such as availability, throughput, response time at service level and SLA is the contract with the customer promising certain SLOs.
42. Linearizability
    * Linearizability is one of the strongest single-key consistency models. It implies that every operation appears to take place atomically and in some total linear order. This means it’s consistent with the real-time ordering of those operations. In other words, the following should be true of operations on a single key:
      + Operations can execute concurrently, but the state of the database at any point in time must appear to be the result of some ordered, sequential execution of operations.
      + If operation A completes before operation B begins, then B should logically take effect after A.
43. Blocking vs Non-blocking IO
    * In Blocking IO the process is blocked until data is available and in non-blocking the process continues even if the data is not available.
44. Graceful Shutdown
    * Properly terminating threads and releasing resources when shutting down an application.
45. Fail-safe vs Fail-fast Design Principle
    * A fail-safe system is designed to continue operating, possibly at a reduced level, rather than failing completely when some part of the system fails. Some characteristics are Graceful Degradation and basic functionality over full operational capabilities. Examples are traffic lights, nuclear reactors, backup systems etc.
    * A fail-fast system is designed to immediately report any failure or error, halting further execution to prevent potential issues from propagating. Some characteristics are Early Termination and Immediate Error Reporting. Examples are Input Validation, assertions, network protocols.
46. North Star Metric (NSM)

# 

# System Design HLD

<https://docs.google.com/document/d/1jVUeQGD_xFlnVPBEdSHrxka-CESJBe_xuV-IA-kKxOs/edit>

# System Design LLD

<https://docs.google.com/document/d/195UBBc7vSewe9BWUHLAvQFClR4HTLloozVhjdgZ54H8/edit>

# Java Quick Reference

<https://docs.google.com/document/d/1SyPSJoIh_HhOw3mbTpCQIV1b_TCdkrOTHyeHSr5CwGc/edit>

# Kafka & Zookeeper

<https://docs.google.com/document/d/15qOi-rMsaZqla6QWYMwhUrR3_P6L70bhEtmQDNRvpXA/edit>

# RabbitMQ

<https://docs.google.com/document/d/1sC_mCfLuC-rjFxp55jZhH1Uhr1mEDldV8HmDoCai6KI/edit>

# Redis

<https://docs.google.com/document/d/1jmnqvk5vSYTLMAY_Spt_bL2id-pgl8UbpiPDALuw2uI/edit>

# Aerospike

<https://docs.google.com/document/d/1l_KCZMUax4Jpd_lYCOzizWXmL7BSz8jTV_ATmj_uHwY/edit#heading=h.30j0zll>

# Memcached

<https://docs.google.com/document/d/1MfvRW8VZrBtI6cT-HXNr2eHlCyrkyc5iZVC2cZx-voc/edit>

# MongoDB

<https://docs.google.com/document/d/1mNctWd0uz5DV0LhTO5qjxz0y0d7rpjaTpi8B_Wi7Kew/edit>

# Neo4J

<https://docs.google.com/document/d/189gAod69aaGEaETgPgpog6np_eNnX0T7Imtuz5qBoHQ/edit>

# MySQL & SQL

<https://docs.google.com/document/d/18ZS-wyJsSDk1jJV6Lm_S9NqmSOWsvZ7sb7OpydsQnRI/edit>

# Cassandra & HBase

//Todo

# Docker & Container

<https://docs.google.com/document/d/1NQS1UuN5D4RSd7fXW1pcy61sDYSk8oZ8J-C1swZQJV8/edit>

# Kubernetes (K8s)

<https://docs.google.com/document/d/1Stp3XzKefriVcbYGYTZZlVXLyujBeZqHUPaVYdxAdZI/edit>

# AWS

<https://docs.google.com/document/d/1DAIvB9gwgxVFdREiDVtf9PeUw6rokjUN-EZg5bQsOnA/edit#heading=h.2o9wnwjcjek8>

# DEVCON-24 Wrap-up

<https://docs.google.com/document/d/1cn6ENS_ZtV_6fhNDWyw0dTLd2S6FDa-DTxnk3EES5mY/edit#heading=h.u7wekvvj0h36>

# Random Flow Diagrams

<https://docs.google.com/document/d/1eGp677BRd46b3yPsGXRDVllUtHr6HGbW7KWCUF1T31A/edit>