**System Design** is about designing a system’s components, architecture, and modules and how data will be flowing from one element to another. In System Design, We focus on solving a problem by specifying all necessary components for a sound running system. System Design is also a part of problem-solving on a larger scale.

5 Important steps to design a reliable system:

1. **Designing Interfaces (GUI, API):** - Making signatures for API Designing (Rest API or 1 GraphQL ) and how many screens are required to complete the Application. The interviewer checks how broadly you are thinking about your application
2. **Capacity Approximation**: We ask the interviewer about scenarios like how much our application is scaled, how many DB Calls per second would be there.
3. **Selection of the right Database and Data Structure**: Choosing the right DB, files, RAM based on our storage and server type requirement is the next step.
4. **High-Level Detailed Design**: show the flow of data from Client to Server and vice-versa using boxes and flow diagrams. You need to provide the number of nodes (hosts) of the server, Load Balancing, Caching, DB persistence, and all other details required to design a better possible solution.
5. **Optimization**: Check for Point of failures, if backup requirements are there, bottleneck conditions.

**Important Factors to be considered while designing a system:**

**Scalability:** It simply means making the system flexible enough to accommodate new users and handle efficiently increased data rate and traffic load. The performance and management of the system and complexity shouldn’t be affected while scaling.

**Throughput:** It is defined as the amount of work done by a machine in a given particular time. Throughput is one of the significant metrics for Network performance. Generally, we measure the throughput of the server how many API Calls it serves in a unit of time. Generally, we measure throughput in GBps, MBps, KBps. How to increase throughput?

* Simple answers can be paying for the good server if higher throughput is needed
* By increasing the capacity of load taken by the server.
* By increasing the speed of work done by the system.

Bandwidth: Maximum data that can be transferred over different Networks is known as Bandwidth.

**Response Time**: Time taken by any API in response to API Call is response time. Let’s say we sent a request to API at t1, and you got a response at t2 that the time difference between request and response is response time.

**Latency**: Latency is an important measure for the performance of the system. It is defined as how long a system takes to transmit data from one point to another point in the system.

* A Network request latency is the time taken by a request sent by the client and server sending the same request’s response.
* Time is taken for reading data from a file on Server.
* If we perform ten types of operations in our system, latency can differ for different functions.

**Availability:** It simply means how fault tolerant a system is. If any failure occurs, then how will the system work? That is how resistant a system is in adverse conditions and how it repairs itself whenever needed.

**Consistency:** When two or more nodes are supposed to share the same data, any node can manipulate data first, and the other node is also trying to access, while node A writes something at t1 time, and node B read before t1, but the data got updated in meanwhile, leading to inconsistency. To avoid inconsistency, we can use Master-Slave architecture, which can become peer to peer in some cases while two replica share data at the same level.

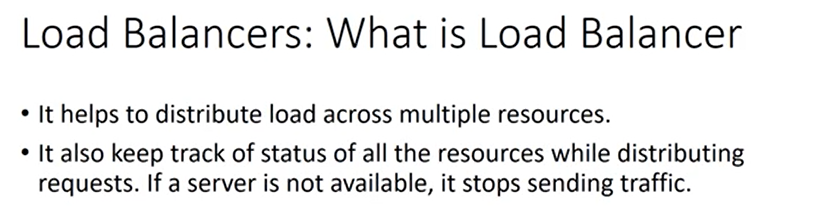
**What is the difference between Weak, Strong, and Eventual Consistency?**

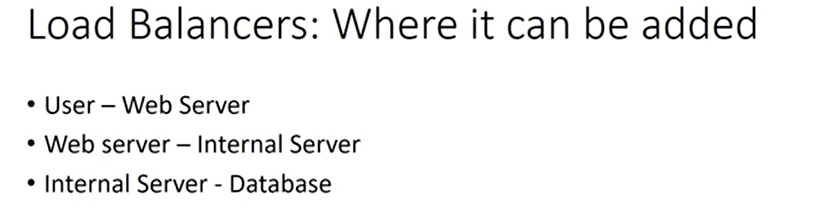
Eventual Consistency: This simply means that the data must be highly available and partially consistent. In case of and overlapping read with an edit, the read might be inconsistent at first but later it gets updated to a consistent value. Here availability is the priority over consistency but none the less consistency is also achieved**.**

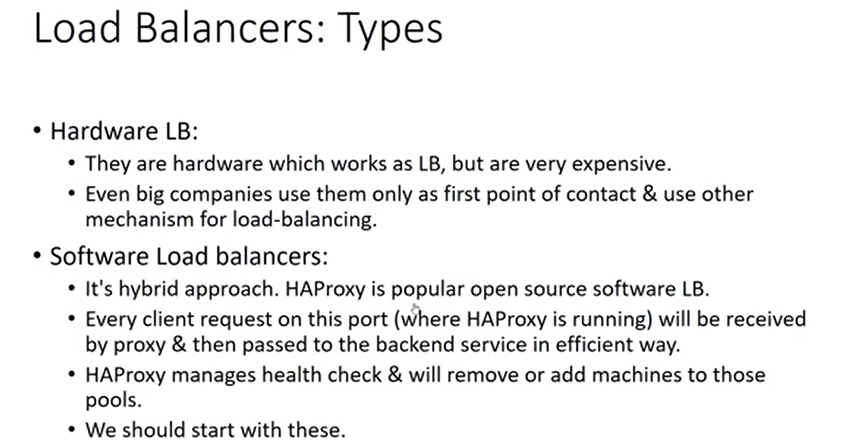
Strong Consistency: Strong Consistency simply means the data must be strongly consistent at all times. All the server nodes across the world should contain the same value as an entity at any point in time. And the only way to implement this behaviour is by locking down the nodes when being updated.

**What are some examples of availability patterns?** H/W redundancy for application servers and DB Master-slave model and replication.

**Load Balancer:**

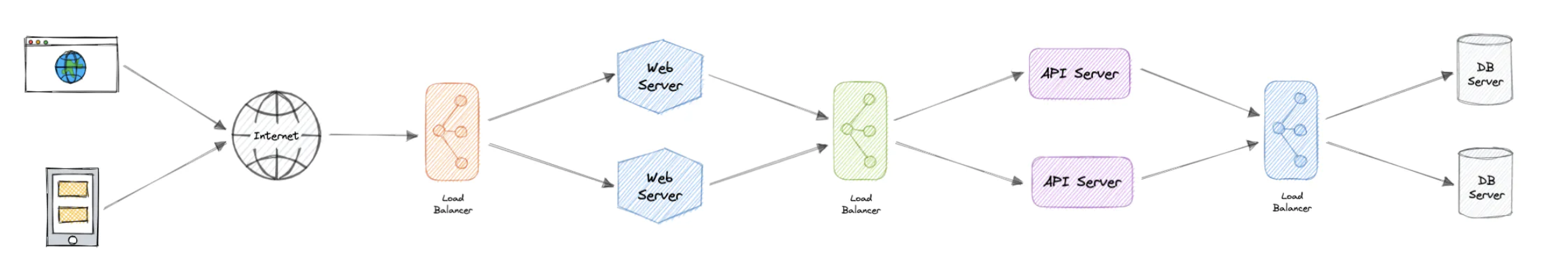


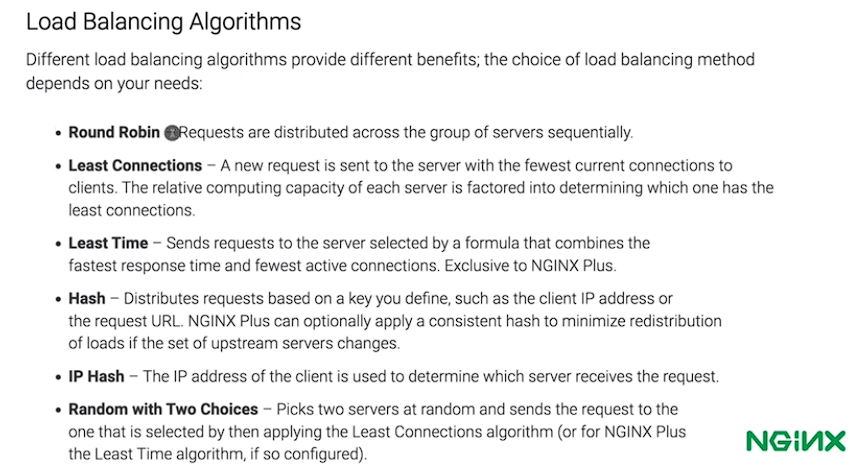




**Load Balancing** helps to/attempts to evenly distribute User request traffic among multiple application servers (distribute processes across multiple Processing units).  **Ensures High availability and Reliability.** Servers can be weighted(higher/lower specs), requests can be session-based and application servers are loosely coupled and can be add or removed from the system.

Hardware load balancers include proprietary firmware that requires maintenance and updates as new versions, and security patches are released. Software ones are provided by Cloud servives and are flexible and cost effective.





**In round robbing**, the requests are assigned to responding server sequentially in a cyclic fashion. This approach fails when sessions come into picture or when servers are weighted. In **weighted Round robin,**  we before hand assign request bucket to a server based on the server handling capacity. Least response time algorithm considers both average response time and least number of active connections. The server with least active connections -> lowest average response time gets the new client request.

Placement of the load balancer.: Layer 4 and 7. (Transport and Application layer in OSI model)

Layer 4 load balancing does look up the data/ message content. It forms a single TCP connection and is simpler, fast and more secure. But it does not allow caching, smart load balancing/micro-service support and enforce sticky segment. This uses Network address translation (NAT) to change destination IP on packet to actual application server’s address and source IP to its own IP. Where as in case of Layer 7 load balancing, we discover the message content and has >2 TCP connections which allows smart load balancing ideal for microservice support and caching. But it is somewhat slower, expensive and decrypts which is not secured if compromised at load balancing level.

**A load balancer can sit in front of the servers and route client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization. This ensures that no single server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts sending requests to it.**

**Why use redundant Load balancers? SPF**

**Supported Load balancer features:** Healthcheacks, Auto scaling, sticky session(Assigne same user to same resource to maintain session), caching, Compression and encrryption. Logging, Redirect. Allow persistent connection.

**Clustering**

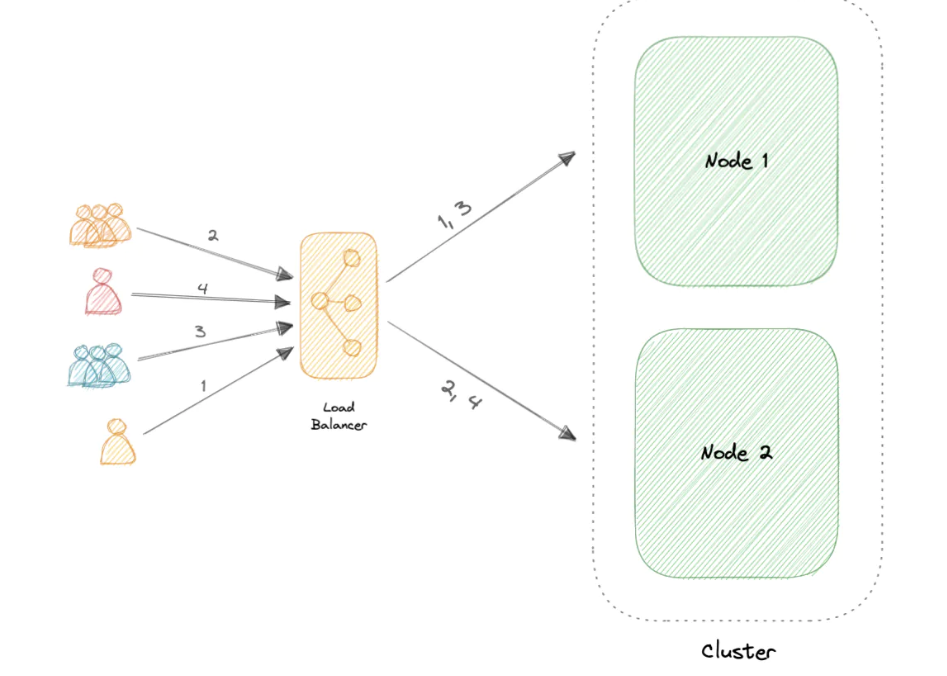
A computer cluster is a group of two or more computers, or nodes, that run in parallel to achieve a common goal. This allows workloads consisting of a high number of individual, parallelizable tasks to be distributed among the nodes in the cluster. As a result, these tasks can leverage the combined memory and processing power of each computer to increase overall performance. This also allows to increase system availability.

Typically, at least one node is designated as the leader node and acts as the entry point to the cluster. The leader node may be responsible for delegating incoming work to the other nodes and, if necessary, aggregating the results and returning a response to the user.

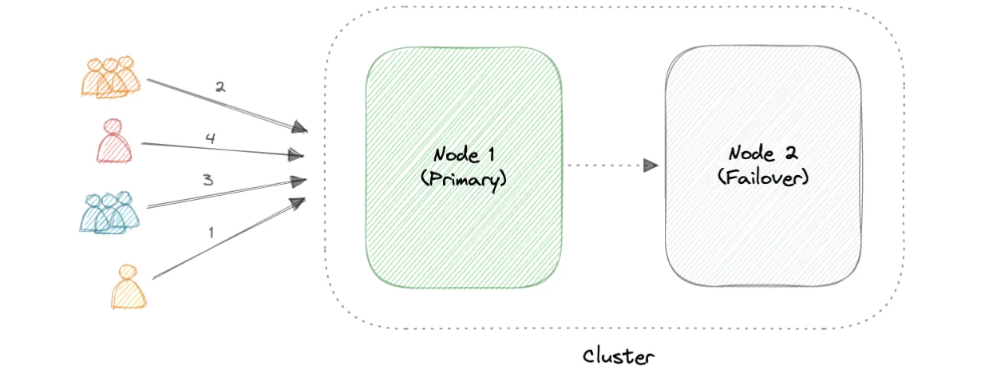
a cluster functions as if it were a single system. A user accessing the cluster should not need to know whether the system is a cluster or an individual machine. Furthermore, a cluster should be designed to minimize latency and prevent bottlenecks in node-to-node communication.

**Cluster Configuration:**

**Active-Active (Primarily to achieve performance and load sharing)**



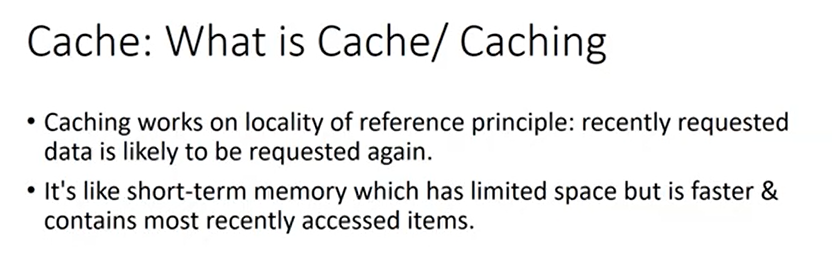
**Active Passive (Primarily to achieve Fault talerance and availability. Not all nodes are active)**



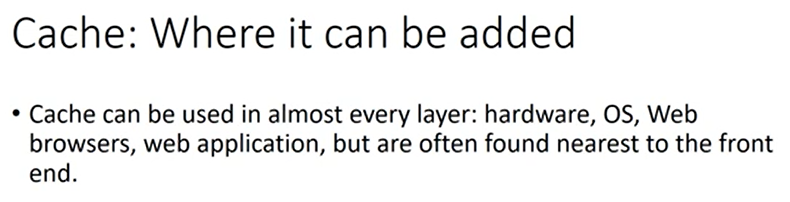
**Caching/Cache memory:**

Cache is defined as a Hardware or Software component which helps in serving the data which is either frequently requested or is hard to fetch/complex or expensive to compute on. A cache's primary purpose is to increase data retrieval performance by reducing the need to access the underlying slower storage layer. Trading off capacity for speed, a cache typically stores a subset of data transiently, in contrast to databases whose data is usually complete and durable.

Caches take advantage of the locality of reference principle "recently requested data is likely to be requested again".



Cache can be built up on reverse proxy org server which processes the request in absence of cache or can be built up on forward proxy server which distributes or routs the data to the application server.



**Cache hit and Cache miss.**

* A cache hit can also be described as cold, warm, or hot. In each of these, the speed at which the data is read is described. Hot cache hit is the hit in L1/fastest level cache, whereas warm cache hit is the one which involves hit from any intermediary level of cache. A cold cache hit is the one where hit still occurs but in the later stages/levels of cache.

**Cache Invalidation:**

Data stored in cache is volatile and does not exist forever. Cache invalidation techniques are needed when the stored data is changed/updated depending on the time. The cache’s K-V would be needed to be updated or removed in case that particular data is changed/updated in persistent storage (warming up of cache memory). Hence faulty or invalid data members of cache are needed to either be updated or removed from the cache storage.

**Few factors deciding Cache efficiency.**

* TTL (time of cache expiration) (if TTL is too short, no point of using a cache and if the TTL is too long, value inconsistency might arise) hence deciding on TTL value is a non-deterministic problem and is totally use case dependent.
* Another way is by taking help of Application server. (This is only possible if cached data is on reverse proxy server). Either application server can check if the write request is on a data object whose key is present in cache and remove it or it can check for the same and update the K-V pair in Persistent storage simultaneously. We can use the Hybrid of TTL and Application layer handling as well.

**Cache Eviction**: If the cache is full, it is needed to be handled in a way that no useful data is evicted and hence eviction policies are used to make space for the new entries.

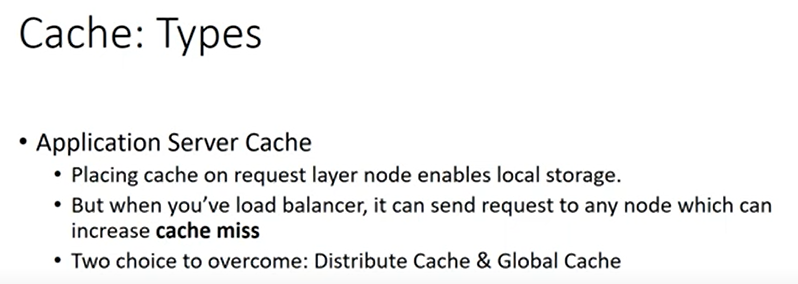
Following are some of the most common cache eviction policies:

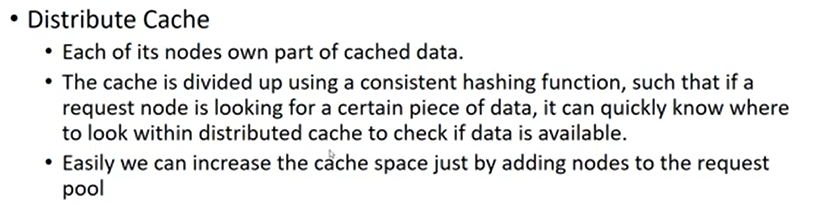
1. **First In First Out (FIFO):** The cache evicts the first block accessed first without any regard to how often or how many times it was accessed before.
2. **Last In First Out (LIFO):** The cache evicts the block accessed most recently first without any regard to how often or how many times it was accessed before.
3. **Least Recently Used (LRU):** Discards the least recently used items first.
4. **Most Recently Used (MRU):** Discards, in contrast to LRU, the most recently used items first.
5. **Least Frequently Used (LFU):** Counts how often an item is needed. Those that are used least often are discarded first.
6. **Random Replacement (RR):** Randomly selects a candidate item and discards it to make space when necessary.

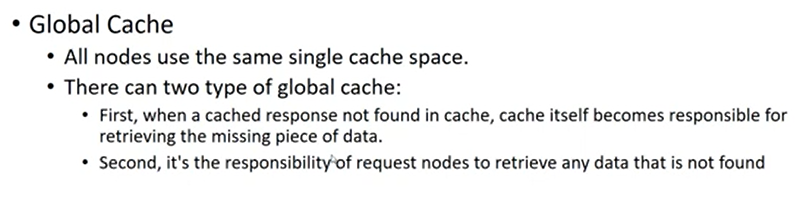
**Cache Retention policies:** No, cache eviction policy and cache retention policy are not the same thing.

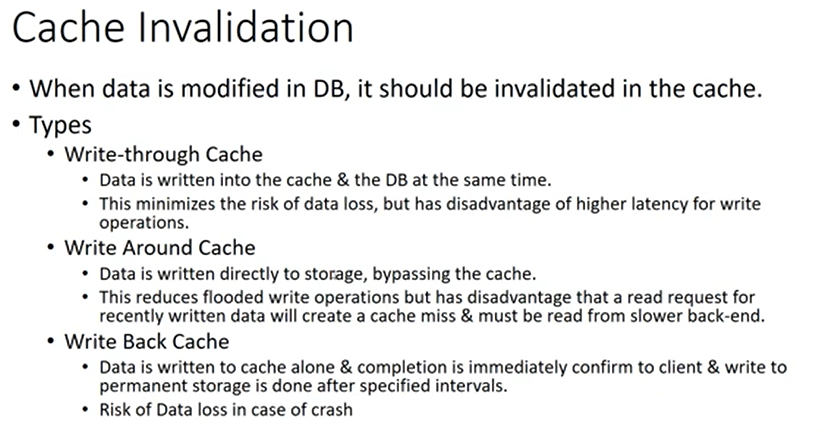
Cache retention policy, on the other hand, refers to the strategy used by a cache to decide which items to keep in the cache and for how long. The retention policy determines how long an item should remain in the cache before it is removed, regardless of whether the cache is full or not.

A cache retention policy may consider various factors such as the frequency of access to an item, the importance of the item, or the expiration time of the item. Some common cache retention policies include time-based eviction, size-based eviction, and frequency-based eviction.











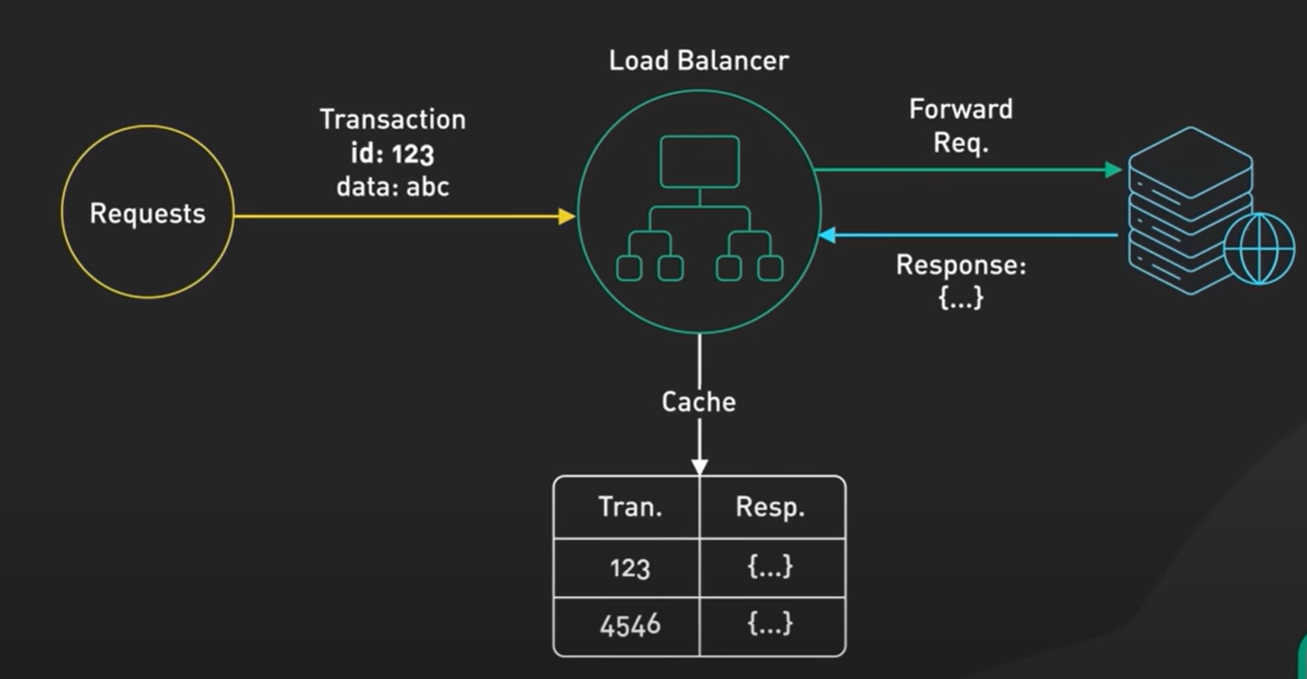
Applications: **DNS, API request and CDN**

**When not to use caching solution: When moer updates/writes than reads. Cache hit time is equivalent to DB retrieval time.**

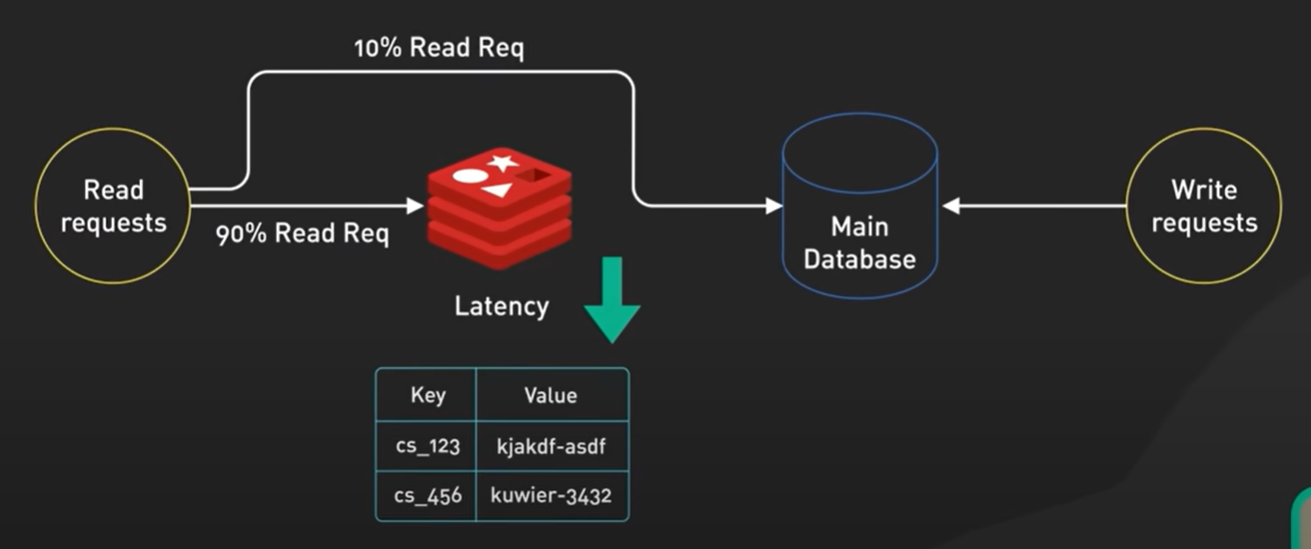
**OS Based caches:**

1. **Page cache:** Controlled by OS to store recently used Disk Blocks in memory.
2. **File System cache:** is a type of cache that stores frequently accessed file system data, such as directory structures and file metadata. The file system cache is maintained by the operating system and is used to speed up file operations.

**Load Balancer can also be integrated with cache to reduce load on application server:**



**How Distributed Cache looks like?**



**What is inverted index?**

**Used in Elastic Search (Text / Fuzzy search solution).** An inverted index is a data structure used in information retrieval systems to allow for fast full-text searches of large collections of documents. It is called "inverted" because it inverts a mapping between documents and the terms they contain, to a mapping between terms and the documents in which they appear. The inverted index is often used in search engines, document management systems, and other applications that involve searching large amounts of text data.

**Example of how it works:**

The process of creating an inverted index involves several steps. First, the text in each document is tokenized and pre-processed to extract individual terms and remove stop words, punctuation, and other non-relevant information. Then, the terms are sorted and indexed, with each term associated with a list of documents in which it appears. Finally, the inverted index is stored in memory or on disk for fast access during search operations.

**Content Delivery Network**

A content delivery network (CDN) is a geographically distributed group of servers that work together to provide fast delivery of internet content. Generally, static files such as HTML/CSS/JS, photos, and videos are served from CDN.

**Why CDN?**

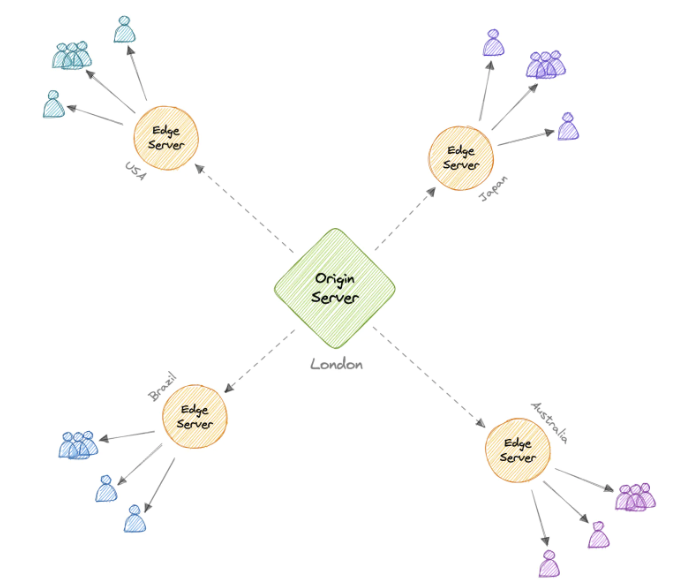
It increases content availability and redundancy while reducing bandwidth costs and improving security. Serving content from CDNs can significantly improve performance as users receive content from data centers close to them and our servers do not have to serve requests that the CDN fulfills.

**How?**

In a CDN, the origin server contains the original versions of the content while the edge servers are numerous and distributed across various locations around the world.

To minimize the distance between the visitors and the website's server, a CDN stores a cached version of its content in multiple geographical locations known as edge locations. Each edge location contains several caching servers responsible for content delivery to visitors within its proximity.

Once the static assets are cached on all the CDN servers for a particular location, all subsequent website visitor requests for static assets will be delivered from these edge servers instead of the origin, thus reducing the origin load and improving scalability.



**Types:**

Push CDN: Gets updated data from the server. Content is uploaded only when it is new or changed, minimizing traffic, but maximizing storage.Sites with a small amount of traffic or sites with content that isn't often updated work well with push CDNs. Content is placed on the CDNs once, instead of being re-pulled at regular intervals.

Pull CDN: Gets updated data on client request. When the client sends a request that requires static assets to be fetched from the CDN if the CDN doesn't have it, then it will fetch the newly updated assets from the origin server and populate its cache with this new asset, and then send this new cached asset to the user.

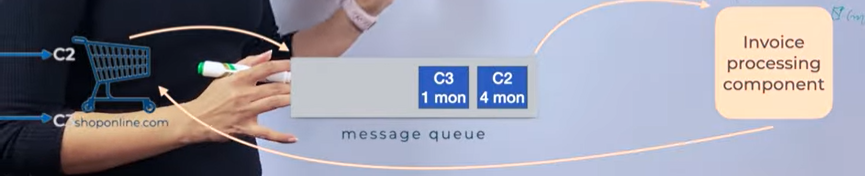
Contrary to the Push CDN, this requires less maintenance because cache updates on CDN nodes are performed based on requests from the client to the origin server. Sites with heavy traffic work well with pull CDNs.

**Message Queue:**

**Synchronous Communication**: Connected (generally 2) components communicated over a dedicated line to exchange messages in a synchronous way. There is no lag at either components for processing the request. Processing is spontaneous and fast and hence the exchange goes sequentially. (Eg: Phone call)

**Asynchronous Communication**: Connected components have a communication lag/defer to process a certain request while not necessarily holding the seeker. Spontaneous response in not expected as the requested process might take some time. Hence in case of async-await there is a function lag to process and utilize the result or output of the requested process while maintaining consistency. (Eg: text and mails.)

Message Queue is a Queue data structure that is utilised to achieve a highly scalable asynchronous communication. Example: (Order Invoice range request by multiple users)



Here whole communication is asynchronous. Request are time intensive and need more time produce the desired result. It provides ease to the system to process multiple requests in a non-simultaneous fashion plus it makes the system request scalable. Here the 2 components can communicating are online store and invoice processing component. Examples: Kafka / Rabbit MQ / Active MQ.

Message could be a data structure to store upcoming requests or it could be a process running on either same machine or different machine that is needed to be contacted for processing the stored requests. This queue contains messages that are nothing but short sized data which just tells what task is needed to be done. Now what are the components that communicate using this queue? These are producers and subscribers. The components that add message on to the queue are producers while the components that carry out the required task or aid in the desired process are called consumers.

**Advantages**: Handle multiple (lot of) requests and decouple the system into multiple producers and consumers according to the load thereby providing scalability. Message queue is also reliable in case if the components lose / faces an unwanted failure, queue still holds the requested attributes to once again start the processing.

**Ordering**: Once the consumer processes the front request and acknowledges the result as required, then only the request is dequeued from the queue.

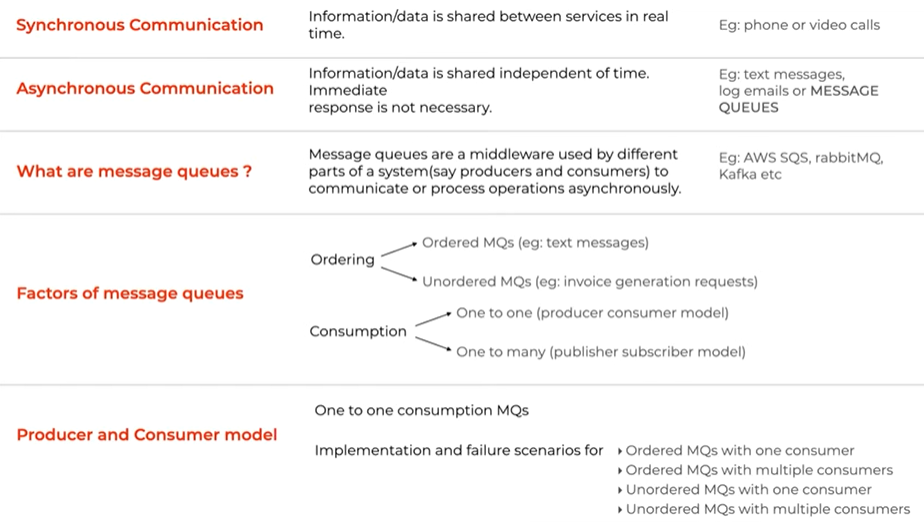
**Ordered Message queues**: Amazon SQS, ordering is imperative and done in FIFO manner. In case if some request is failed to be processed, queue is blocked until the front() message is not processed and acknowledged to the queue. This ensures ordering. Example (Chat application). This happens similarly in case of multiple consumer presence. Functioning gets blocked until the expected consumer do not completes the unfinished request.

**Unordered Message Queue**: Ordering isn’t important here. In case of failure in a request process, the failed message is pushed onto a DEAD Letter Queue (DLQ) and is pushed again into the Message queue for re-processing.

**Message Queue Models**:

Producer and Consumer: Consumption is 1 to 1, One message is consumed only once by one consumer.

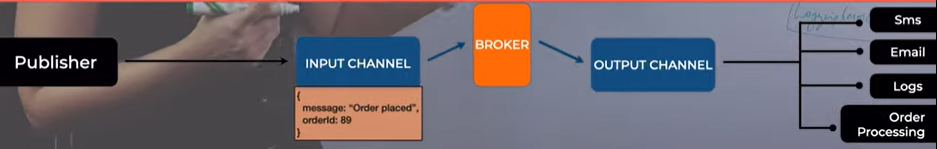
Publish Subscribe: Consumption is 1 to Many: For one message multiple processes have to do something in response to that message.

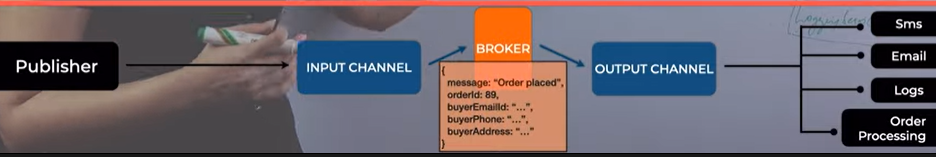


Publish Subscribe is also a pattern for asynchronous communication. Analogy: “Hospital announcements analogy”.

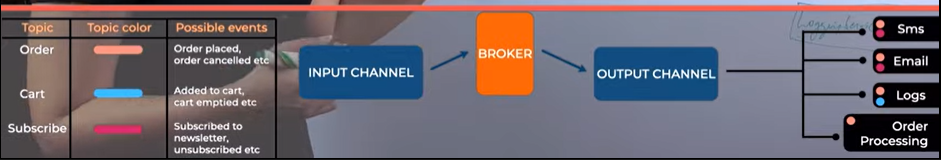
In any system, a component that informs about certain event that has happened is called a Publisher. It pushes the messages informing what event has happened in the system. An Input channels is a physical component where publisher will push all the event messages. Output channel is where the subscribers who reacts to the respective event messages subscribe. Now there could be multiple output channels depending on events concerning certain subscribers (A subscriber need not to be informed about all the published event messages, hence this subscriber would need a separate output channel which will inform it about only the concerned events). Message broker in the middle has to hold on with multiple responsibilities like:

1. Enriching/modifying messages as per the requirement by the subscribers.
2. Message broker can divide the messages into different topics. Relating to our previous analogy, an ER announcement (topic: ER, ER announcement: event message) is needed to be heard by everyone whereas speciality announcement (childcare dept. doctors) are concerned and done only on specific floor(s) (this event messages gets routed onto the concerned output channels).









Publish Subscribe model allows decoupling of components and scaling of the system very easily. It also increases system performance. Ordering here is not decided and consumption is 1 to many. In order to achieve a certain ordering we would need to priority to every event message and push it into a priority queue instance. Repeated and 3rd party generated ill-messages are needed to be taken care for.

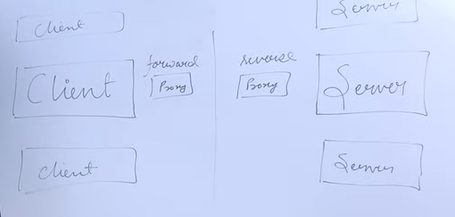
**Proxy**

**Proxy server:** Analogy (broker (asset) between 2 parties for communication and abstraction for security);

**Forward Proxy**: Think about proxy server as a machine which sits between the client and the request handling server, but on the client side which talk to the server on behalf of the client. Useful for anonymity. This is useful when you have multiple clients and all the traffic is to be monitored and controlled via forward proxy. This can also be used when certain access permissions are to be granted within a general group of requests. (Can also be used for blocking access to certain site or Caching user requesting on the proxy site). In certain scenarios this type of proxy can be used for illegal accesses.

**Reverse Proxy**: When the proxy server sits in between client and the request handling servers but on the server side and client only has the knowledge about the proxy server which in turn distributes the request onto different request handling servers. This setup is majorly used for security purpose (like request and data SSL encryption) and abstracting actual server details and it can also be used as corporation’s Load balancing server. It becomes a bottleneck when we have a possibility to have a SPF on reverse proxy site.

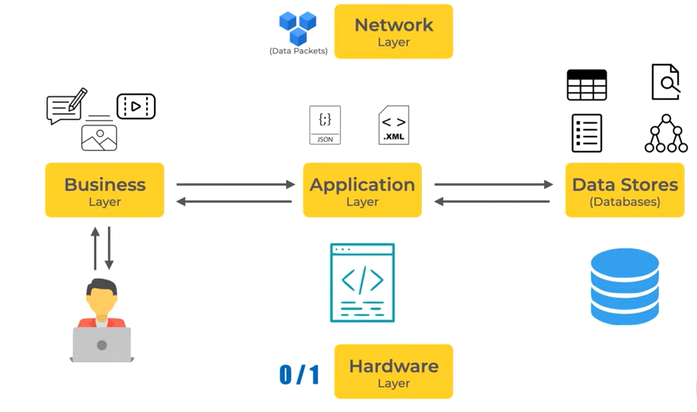
Mixed Server Architecture:



**Load balancer vs Reverse Proxy**

Wait, isn't reverse proxy similar to a load balancer? Well, no as a load balancer is useful when we have multiple servers. Often, load balancers route traffic to a set of servers serving the same function, while reverse proxies can be useful even with just one web server or application server. A reverse proxy can also act as a load balancer but not the other way around.

**Data Flow and representations:**



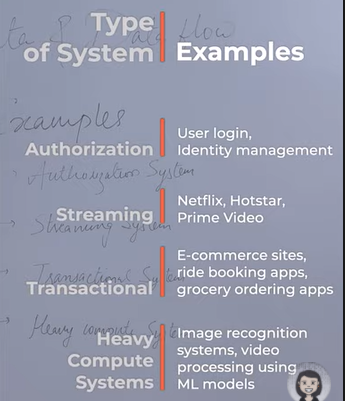
Data Stores: Databases, Indexes, Queues, Cache. (Places where data is needed to be stored):

Dataflow methods: Methods like User event calls, Messages (in real time applications) and APIs etc…



Data generator: User/Internal data (meta-data)~system data/Insights (user data populated by system)

Data Factors Affecting System design: Type of data/Volume/Consumption and retrieval and security.



**Database:**

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

**DBMS**

A database typically requires a comprehensive database software program known as a Database Management System (DBMS). A DBMS serves as an interface between the database and its end-users or programs, allowing users to retrieve, update, and manage how the information is organized and optimized. Facilitates Admin authority and data security.

**Schema**: The role of a schema is to define the shape of a data structure, and specify what kinds of data can go where. Schemas can be strictly enforced across the entire database, loosely enforced on part of the database, or they might not exist at all.

**Aims of Database**:

1. Allow real time access and management.
2. Allow increased volume of data processing.
3. Ensure Security
4. Easy management of database

**SQL (Structured Query Language):** SQL database is a collections of items with predefined hard definitions. Organized as a set of interlink tables. Single table represents an object information (logical entity’s metadata). Note: Each row in a table could be marked with a unique identifier called a primary key, and rows among multiple tables can be made related using foreign keys.

+ Consistency and Easy to visualize

- Difficult to scale (Horizontally) and complex schema evolution

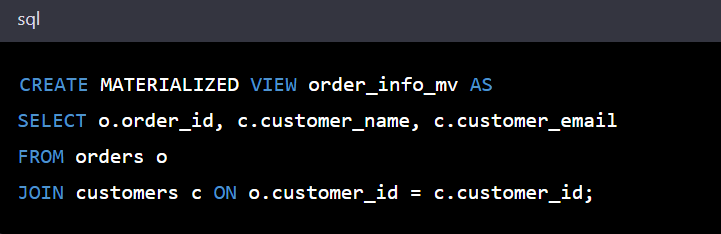
- Performace hit (Joins and denormalization as schema is enforced)

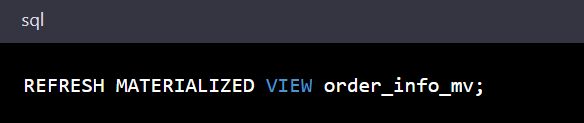
**Materialized View**:

A materialized view is a pre-computed data set derived from a query specification and stored for later use. Because the data is pre-computed, querying a materialized view is faster than executing a query against the base table of the view. This performance difference can be significant when a query is run frequently or is sufficiently complex. Used when we have frequent queries from **table subset or complex joined tables or we need a table replication.**

**But how is it different from View?**

A view is a virtual table that is based on the result of a SQL query. It does not store data on disk, but instead retrieves data from the underlying tables every time it is queried. A materialized view, on the other hand, is a physical copy of a view that is stored on disk. It is essentially a table that is populated by the result of a SQL query, and it is refreshed periodically to keep its data up to date. (Increase performance by reducing I/O access time for frequent complex queries).





**N+1 Query Problem:**

The N+1 query problem happens when the data access layer executes N additional SQL statements to fetch the same data that could have been retrieved when executing the primary SQL query. The larger the value of N, the more queries will be executed, the larger the performance impact. **Happens in GraphQL or ORM tools.**

Can be addressed by optimizing the SQL query or using a dataloader that batches consecutive requests and makes a single data request under the hood.

**NoSQL database:**

NoSQL is a broad category that includes any database that doesn't use SQL as its primary data access language. These types of databases are also sometimes referred to as non-relational databases. **Unlike in relational databases, data in a NoSQL database doesn't have to conform to a pre-defined schema.**

**Types of Databases:** Different databases depending on query requirements and properties of data like (type and volume) provided different ways and features to store data.

Popular types:

* Relational DB:

The 2 factors which helps us decide if one should have a relational DB in the system or not are ACID properties (-) and Schema (How your data is going to be structured). Also the data relationship is well known.

+Here Database with complex relation can easily be designed.

+Inconsistency has no chance here. Even Database constraints helps in putting strict guidelines on how and what should be needed.

+ACID

**When to Use RDBMS:**

Application involving frequent Transactions (requires ACID properties to hold anyway) like banking or monetary applications.

When you are certain that Table structures are future proof in terms of updation.

-Vertical Scaling is supported but not the horizontal one.

-When Table size grows JOINS operation can be an expensive one and can hamper the system performance.

* **Non-Relational DB**: Schema of stored data is not fixed.

1. **Key-Value Store**: Example Redis/Dynamo DB/Memcache. Just like a Hash-Map.

+Fast and Provide quick access as most of the data is in memory (performant).

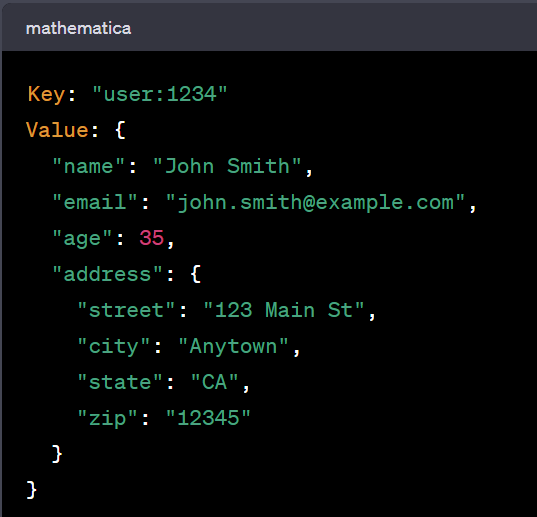
Is generally used by cache with request response pair as Key value pairs.

On technical side, key value store is generally used for in-memory data caching to speed up applications by minimizing reads and writes to (slower) disk-based systems.

+High performance and scalability.

It is not ideal for a scenario where there are heavy writes/updates. Not considered suitable for applications requiring frequent updates or for complex queries involving specific data values, or multiple unique keys and relationships between them.

In a key-value store, data is organized as a collection of key-value pairs. Each key-value pair consists of a unique identifier (the "key") and an associated value. The key is used to look up the corresponding value, allowing fast retrieval of data based on the key.



1. **Document DB**: They are usually used when Schema is not needed to be future proof and you want a flexibility of keeping dynamic data (might change over time) then this is the best option. Use cases fits both transactional and analytical applications.

Collection (Doc…)

+It provides high performance over complex join structure in RDBMS as it stores everything in one entity.

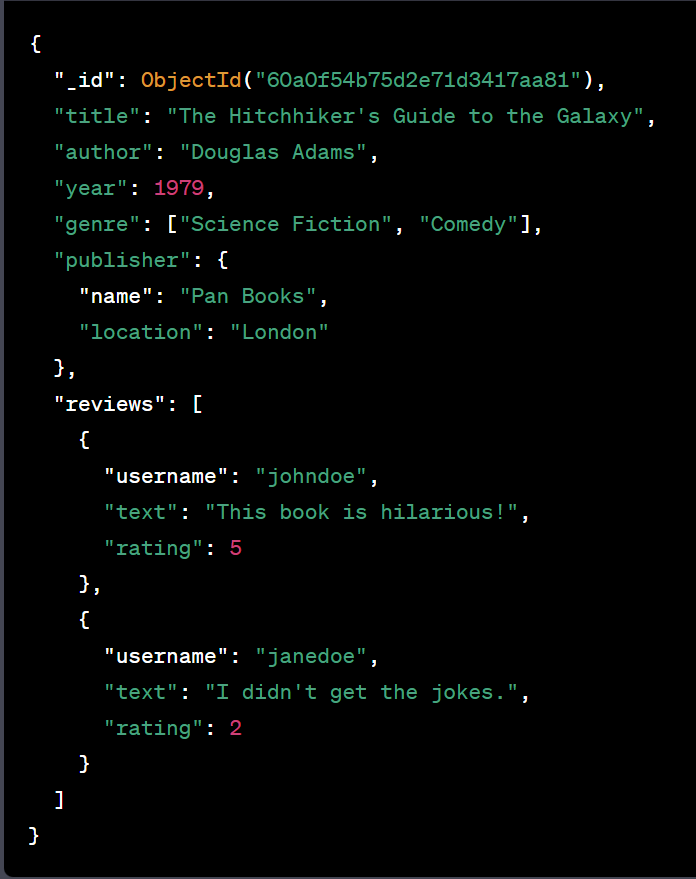
+Support heavy read and write.

-Data inconsistency and dirty data as ACID property is absent.

-Schemaless.

+Highly scalable (Horizontal)

Examples: MongoDB / CouchDB.



1. **Wide Column DB**: Sort of midway between RDBMS and Document DB. There is a fixed schema but do not support ACID properties. They are generally used when you have a application involving heavy writes of data like streaming data/event data. (IOT, commodity tracking). They do not support heavy read operation but special kind of reads. Cassandra is a good example of this type of DB. Column are good supporter of Distributed DBs.

Example: Cassandra

**SQL VS Wide column DB:**

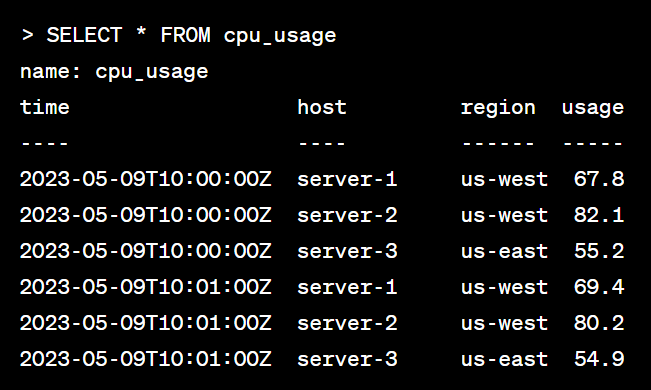
One of the main differences is the flexible schema design. In a traditional SQL database, the schema is usually defined before any data is inserted, and all rows in a table must adhere to the same schema. In contrast, a wide column NoSQL database like Apache Cassandra allows for different column families to have different sets of columns, and each column can have a different data type. This makes it easier to store and query large amounts of data with varying structures and types.

Another difference is that wide column NoSQL databases typically scale horizontally by adding more nodes to the cluster, rather than vertically by upgrading a single machine. This allows for better performance and fault tolerance, and makes it easier to handle large and rapidly growing data sets.

1. **Time-Series Database**: A time-series database is a database optimized for time-stamped, or time series, data. It is more optimized and focused towards writing data than to read. This is because they are designed to handle large volumes of time-stamped data, which often requires fast and efficient writes.

In a time series database, the data is typically organized by timestamp and may include metrics such as sensor readings, stock prices, or website traffic data.

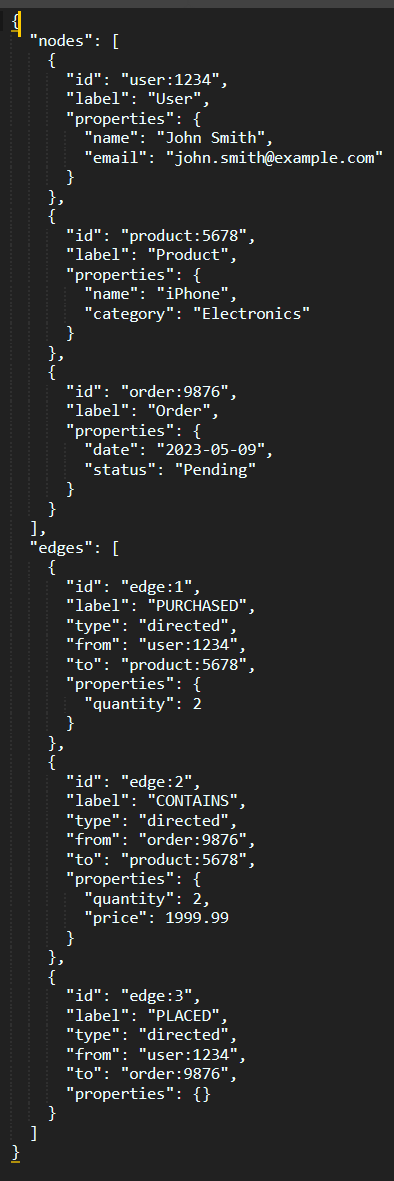
Example: Influx DB (for time series : Query language here is SQL like InfluxQL)



1. **Graph based NoSQL**: A graph database is a NoSQL database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data instead of tables or documents.

The graph relates the data items in the store to a collection of nodes and edges, the edges representing the relationships between the nodes. The relationships allow data in the store to be linked together directly and, in many cases, retrieved with one operation.

Example: Neo4J.



* File DB
* Network DB

**SQL vs NoSQL:**

|  |
| --- |
| **Conform to Schema Constraints** |
| **Query Languages** |
| **Scalability (**SQL databases are vertically scalable, which can get very expensive. Still it is possible to scale a relational database across multiple servers, but this is a challenging and time-consuming process. | InNoSQL databases horizontal scaling is possible, meaning we can add more servers easily to our NoSQL database infrastructure to handle large traffic**)** |
| **ACID compliancy** |

**When to use when:**

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| Strict Schema and relational data. | Flexible based on schema and non-relational data |
| No need for too much scalability | Very data intensive workload |
| Transaction oriented(need to hold ACID properties) | Very High throughput for IOPS. |

**Database Replication:**

Replication means to copy/or to have a copy, in this case- of a Database (DB). This process is done to avoid Single point of failure and data inconsistency. Main database which has the write authority is called the master / Leader DB and the one that continuously copies the master database and replicates the data from master DB is called slave or secondary DB and they are usually >1 and are used for read operation. How replication helps?

* Fault tolerance: Let us say, some mishap happens with one of the DB’s hardware. With replicated DB available, we can have all the data still available and this can now be as master DB.
* Reduce Latency: because distributed and replicated FS / DB makes the information access fast. A geographically close location will surely help us reduce delivery time as compared to a far located DB on the same network.
* Increase system performance and make it scalable as replica will help out in sorting multi-read operation’s burden.

How replication happens: One could be periodically send master DB snaps to the replicas or using CDC (Change Data capture) slight change in Master DB initiates a change forwarding process to update all replicas.

Replication Lag: When a read and write request occurs at the same time. Let us say if write request happens at T1 and a read request occurs at T2 and the systems replication lag is R, then,

* Everything is fine until T2-T1>R; Till the time T2 secondary DB would be updated as per the write request of T1. But if T2-T1<=R, data consistency is hampered, and dirty read will happen (**Eventual consistency**).

This is the challenge faced in a DB replication model. This is resolved by consistency algorithms like “read after write”; Let us say a new write request is issued at T4 on master DB (obviously). Now with this model of “read after write”, master will updated according to the request and send the update changes to all the replicas and wait for their acknowledgement(s). When it receives all the acknowledgements is will inform the request issuer that the change is updated in the database and hence Replication lag is 0 with full data consistency. This is called **Synchronous Replication**. The downsides here are that system might take a performance hit as the processing of the write request holds onto the other operations. Here in case of replica failure scenario during acknowledgement phase also takes a lot of consideration.

**Asynchronous Replication**: Primary DB will not wait for the replicas’ acknowledgements and the replica update process is done in the background. When the write request is issued at the master, master will first forward this request to the replica DBs and then will update itself according to the request. Advantage is of lowering the chance of performance hit of DB but there still might be some inconsistency in this system.

Semi-synchronous Replication: When a new write request is issued, master DB will forward the request to all the replicas but will wait for acknowledgement from only one replica. Then the Master DB will also update itself and informs the request issuer about the update.

Snapshot vs. Replica: Snapshot is a state of DB at a certain time whereas DB replica is the full copy of the master DB. Snapshot allows the DB system to roll back to a recently saved snapshot but it won’t allow you to reduce latency and scale in case of multi read operation thereby increase performance as the replica do.

**Database Sharding:** DB partitioning (Persistent storage scaling via DB or collection partitioning).

Vertical Partitioning: Store every column in different Shards.

Horizontal Partitioning (Actual Sharding): Store bunch of rows / collection’s Doc in 1 shard.

Physical and Logical Shards: Logical Shard is nothing but portioned data (Say out of 1 billion users table, we have 4 partitions (horizontal) each of 250 million based on some sort of key, these each partitions are called shards). Whereas Physical Shard is the actual machine where the logical shards resides.

Advantage of Sharding: Scale up accordance with the customer traffic. Also Increases system performance via query optimization (If we need to query something based on the key we know in which physical shard do we need to search and hence we do not need to run the query over all the records, but instead a shard). Avoids risk of SPF. Also geographically distributed data can be stored in equivalent geo-based DB server (Tinder).

Algorithmic Sharding (Application logic decides target shard) Dynamic Shard (A follow-up registry decides the target shard).

Drawbacks of Sharding: Data distribution logic is needed to be good enough so that even distribution of data is possible. (Otherwise might result in complications for system performance). Sharding is irreversible.

Key Based

Range Based

Directory Based

Hashing

Consistent Hashing

CAP Theorem

# N/W Basics:

* **IP**

An IP address is a unique address that identifies a device on the internet or a local network. IP stands for "Internet Protocol", which is the set of rules governing how devices can be made accessible for communication. The internet needs a way to differentiate between different computers, routers, and websites. IP addresses provide a way of doing so and form an essential part of how the internet works.

**IPv4:** a 32-bit numeric dot-decimal notation that only allows for around 4 billion IP addresses.

**IPv6:** a new protocol that was introduced in 1998. Deployment commenced in the mid-2000s and since the internet users have grown exponentially. This new protocol uses 128-bit alphanumeric hexadecimal notation. This means that IPv6 can provide about ~340e+36 IP addresses. That's more than enough to meet the growing demand for years to come.

Example: 2001 : 0db8 : 85a3 : 0000 : 0000 : 8a2e : 0370 : 7334

**Public**

A public IP address is an address where one primary address is associated with your whole network. In this type of IP address, each of the connected devices has the same IP address.

**Private**

A private IP address is a unique IP number assigned to every device that connects to your internet network

**Static**

A static IP address does not change and is one that was manually created, as opposed to having been assigned. These addresses are usually more expensive but are more reliable.

**Dynamic**

A dynamic IP address changes from time to time and is not always the same. It has been assigned by a Dynamic Host Configuration Protocol (DHCP) server. Dynamic IP addresses are the most common type of internet protocol address. They are cheaper to deploy and allow us to reuse IP addresses within a network as needed.

* **OSI** is a logical and conceptual model that defines network communication used by systems open to interconnection and communication with other systems. The Open System Interconnection (OSI Model) also defines a logical network and effectively describes computer packet transfer by using various layers of protocols.

Advantages: eaiser trouble shooting, role seperation, security first mindset and widely accepted.

**Layers and there major roles:**

**Application:** This is the only layer that directly interacts with data from the user. Software applications like web browsers and email clients rely on the application layer to initiate communication. Example: SMTP/HTTP.

**Presentation:** The functions of the presentation layer are translation, encryption/decryption, and compression.

**Session:** This is the layer responsible for opening and closing communication between the two devices. The time between when the communication is opened and closed is known as the session. Maintains sessions and synchronizes data exchange via checkpoints.

**Transport:** responsible for end-to-end communication between the two devices. This includes taking data from the session layer and breaking it up into chunks called segments. It aslo reassemnbles the data on receiving device. MAC addess is attached in this layer. TCP/UDP.

**Network:** The network layer is responsible for facilitating data transfer between two different networks. The network layer breaks up segments from the transport layer into smaller units, called packets, on the sender's device, and reassembles these packets on the receiving device. The network layer also finds the best physical path for the data to reach its destination this is known as routing. If the two devices communicating are on the same network, then the network layer is unnecessary.

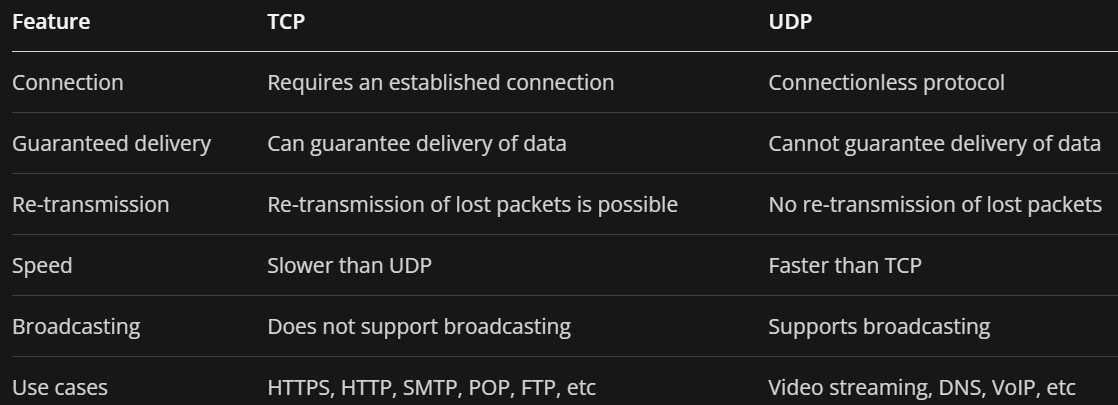
**Data-link:** The data link layer is very similar to the network layer, except the data link layer facilitates data transfer between two devices on the same network. The data link layer takes packets from the network layer and breaks them into smaller pieces called frames. Manages Access control (Pure ALOHA, Slotted ALOHA and CSMA), Flow control (Simplex, stop and wait, stop & wait ARQ, Selective repeat and Go back sliding window.) and Error control (VRC, CRC and checksum) is also its responsibility.

**Physical:** This layer includes the physical equipment involved in the data transfer, such as the cables and switches. This is also the layer where the data gets converted into a bit stream, which is a string of 1s and 0s.

**Scatter Net: A collection of pico nets is typically called a "scatternet". In Bluetooth technology, a piconet refers to a network of up to 8 Bluetooth devices connected in a master-slave fashion. When multiple piconets overlap, they can form a scatternet. A scatternet allows devices to communicate with each other even if they are not directly connected in the same piconet, by using devices that act as bridges between the different piconets.**

* **TCP and UDP**

TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol. A key difference between TCP and UDP is speed, as TCP is comparatively slower than UDP. Overall, UDP is a much faster, simpler, and more efficient protocol, however, retransmission of lost data packets is only possible with TCP.



* **DNS**

Domain Name System (DNS) is a hierarchical and decentralized naming system used for translating human-readable domain names to IP addresses.



**What happens when you query in a search engine?**

**DNS lookup (DNS resolution) involves the following eight steps:**

A client types example.com into a web browser, the query travels to the internet and is received by a DNS resolver.

1. The resolver then recursively queries a DNS root nameserver.
2. The root server responds to the resolver with the address of a Top-Level Domain (TLD).
3. The resolver then makes a request to the .com TLD.
4. The TLD server then responds with the IP address of the domain's nameserver, example.com.
5. Lastly, the recursive resolver sends a query to the domain's nameserver.
6. The IP address for example.com is then returned to the resolver from the nameserver.
7. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.
8. Once the IP address has been resolved, the client should be able to request content from the resolved IP address. For example, the resolved IP may return a webpage to be rendered in the browser.

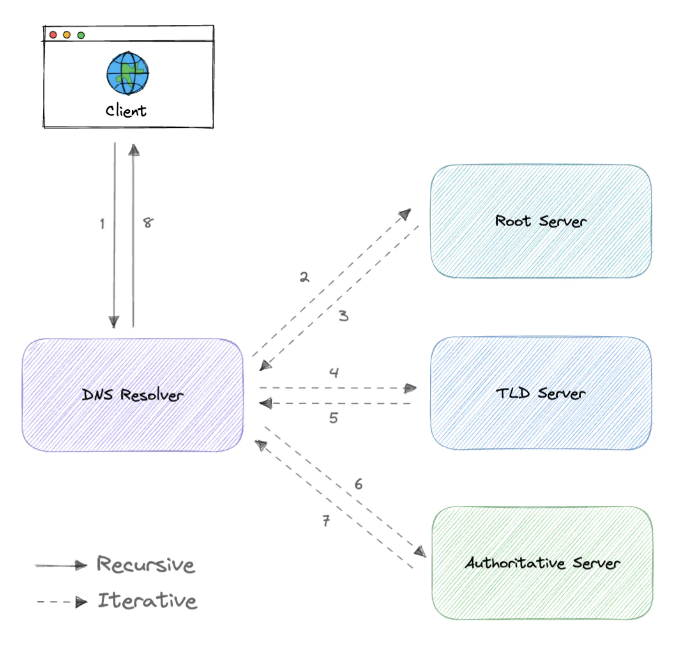
The recursive resolver acts as a middleman between a client and a DNS nameserver. After receiving a DNS query from a web client, a recursive resolver will either respond with cached data, or send a request to a root nameserver, followed by another request to a TLD nameserver, and then one last request to an authoritative nameserver.

A root server accepts a recursive resolver's query which includes a domain name, and the root nameserver responds by directing the recursive resolver to a TLD nameserver, based on the extension of that domain (.com, .net, .org, etc.). The root nameservers are overseen by a nonprofit called the Internet Corporation for Assigned Names and Numbers (ICANN).

There are 13 DNS root nameservers known to every recursive resolver. There are 13 types of root nameservers, but there are multiple copies of each one all over the world, which use Anycast routing to provide speedy responses.

The TLD server holds TLD to domain name mapping. These servers are manages by IANA and are of 2 types: Generic TLD holder and country code TLD holders.

The authoritative nameserver contains information specific to the domain name it serves (e.g. google.com) and it can provide a recursive resolver with the IP address of that server.



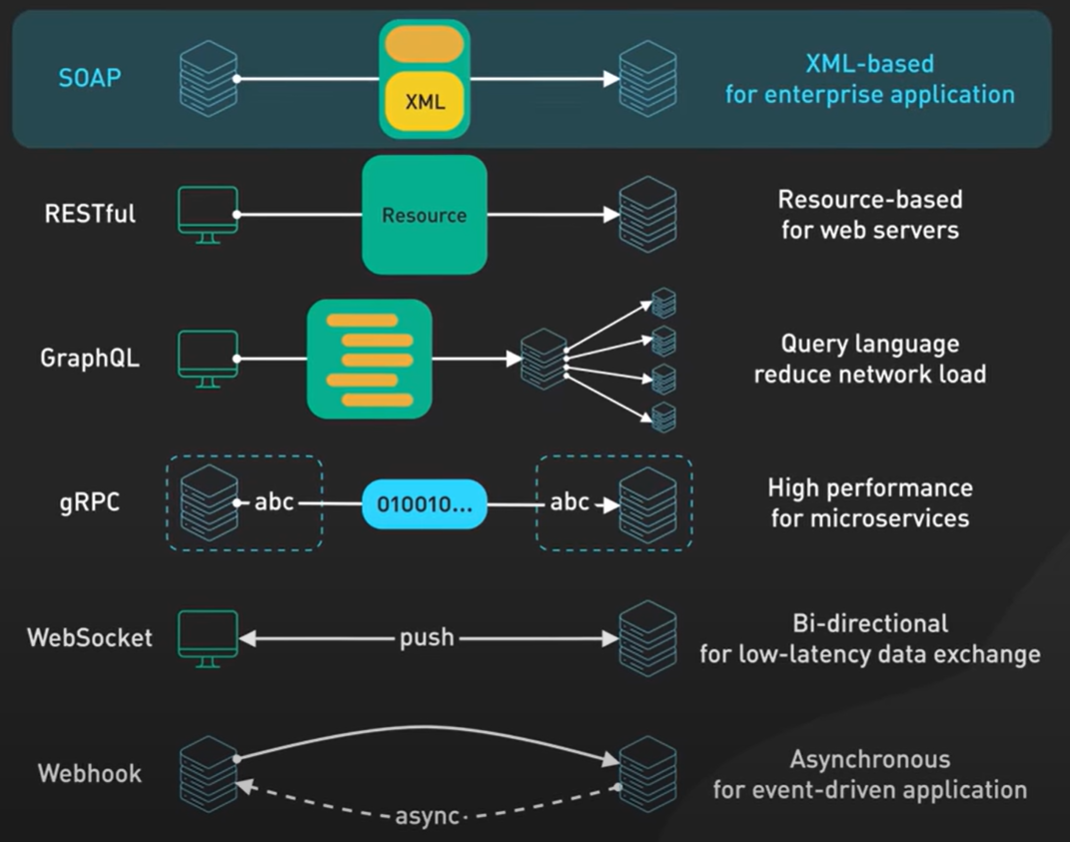
**DNS caching**

A DNS cache is just a memory of recent DNS lookups that our computer can quickly refer to when it's trying to figure out how to load a website. The Domain Name System implements a time-to-live (TTL) on every DNS record. TTL specifies the number of seconds the record can be cached by a DNS client or server.

**Reverse DNS resolving:**

Here we, query for the domain name associated with a given IP address. Reverse lookups are commonly used by email servers. Email servers check and see if an email message came from a valid server before bringing it onto their network.

* **Clustering**
* **Content Delivery Network (Static Content Caching Solution)**
* **Availability in depth**
* Scalability in depth
* Storage Solution
* **API Architectures**: top six most popular API architecture styles, including SOAP, RESTful, GraphQL, gRPC, WebSocket, and Webhook. Each architecture style has its advantages and disadvantages, and their popularity depends on the specific use cases. For example, SOAP is reliable and secure and is commonly used in financial services and payment gateways {For a light weight application SOAP is considered to an overkill}, RESTful build on top of HTTP methods are fast, popular and easy to implement {not ideal for highky connected data model or real time data}, while GraphQL allows for efficient communication {no more over fetching and underfetching} and faster responses with a steep learning curve {more processing on server side}. Additionally, gRPC is modern, fast and efficient and used for microservices architecture {limited browser support}, while WebSocket is great for bidirectional-low latency data exchange, {if real time data transfer is not needed then this can cause unecessary network overhead} and Webhook is event-driven and allows for synchronous operations but might not provide immediate response or synchronous communication. (https://www.youtube.com/watch?v=4vLxWqE94l4)



# DESIGNS

# **URL Shortener:**