Lecture 1 :- Introduction to system design Lecture 2 :- component of system design Lecture 3 – client server architecture Lecture 4- Proxies

Lecture 5: data and data flow

Lecture 6 : Databases types: SQL, NoSQL, Lecture 7 Anatomy of applications and services lecture 8 – Application programming interface

lecture 9 : Caching | Cache Patterns | Cache Invalidation & Eviction Lecture 10 What is a REST API

Lecture 11 : Message Queues | Producer Consumer Model | lecture 12 Pub Sub Messaging

Lecture 13 : pub- sub part 2 lecture 14- Performance metrics

lecture 15 Performance Metrics of components in a software lecture 16 : Fault and Failure in distributed systems

lecture 17 scaling

lecture 18 Database replication

Lecture 19 CAP | Consistency, Availability and Partitioning

Lecture 20 : What is CAP Theorem? Degrees of CAP theorem, use cases

**Lecture 21 :What is Database Sharding, Logical and Physical Shards, Dynamic vs Algorithmic Lecture 22** Key based Sharding

Lecture 23 Range based sharding | Advantages and disadvantages Lecture 24 :- Directory based sharding

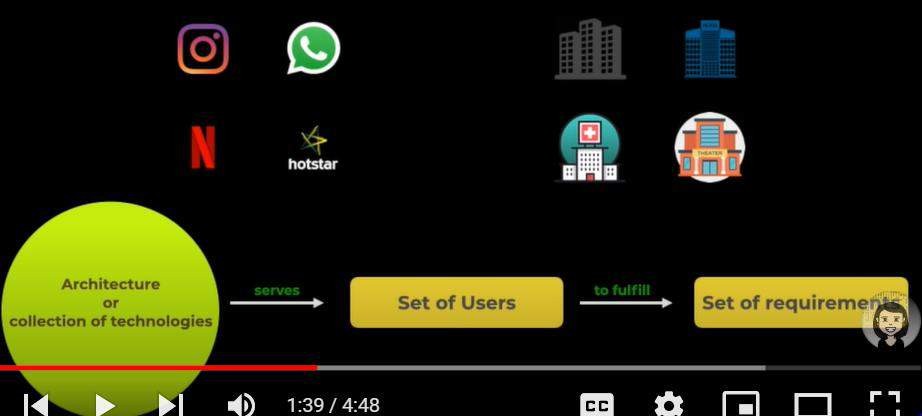
**Lecture 25** A simple introduction to basics of hashing Lecture 26 Basics of consistent hashing

**Lecture 27** Foundation of System Design Interview

**Lecture 28 :** How to solve capacity estimation problems faster

Lecture 1 :- Introduction to system design

System design is collection of technologies which serve to certain set of user to fulfill certain set of requirement





Design is the process of understanding the user requirement and selecting the components, Modules and software technologies how they are interacting with each other to serve the need of the system.

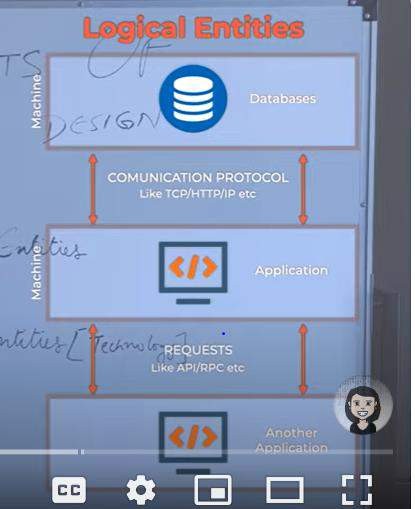
Lecture 2 :- component of system design Component is the basic building block of system design

**2 types :**

Logical entities (databases)and tangible entities (technologies) Note :

Data is store in d/b and there is something to interact with data and that is called application layer like

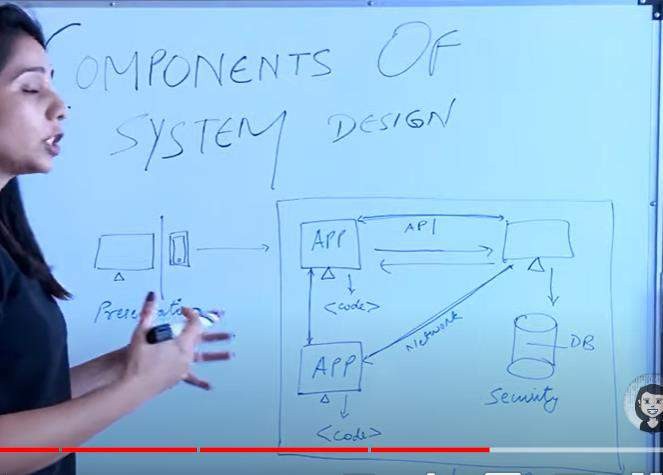
**desktop , mobile app, website**



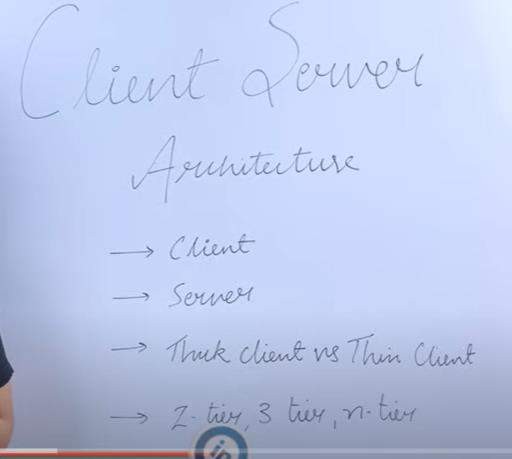
Application interact with each other in the form of request



Presentation layer ( front end application like angular, react)



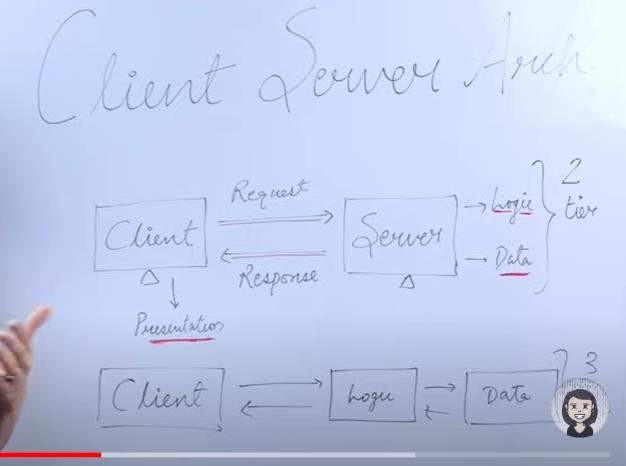
Lecture 3 – client server architecture



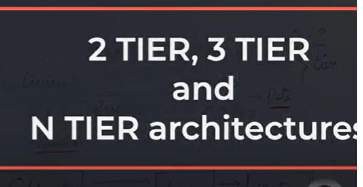
Client request for data from server and server response it back using some logic done on server side .

**Thick client :-** logic sit on client site (Microsoft outlook)

**Thin client** : logic sit on server side ( hotstar)



**Depending on where processing of data and logic happen, then can be thin or thick client**



**If data is huge:-**

**Client presentation layer (front end)**

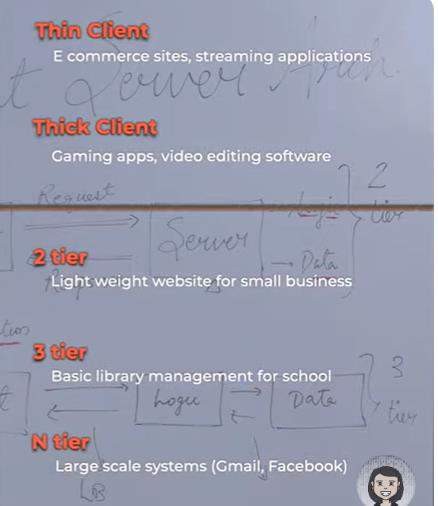
**Logic application layer**

**Data database layer**

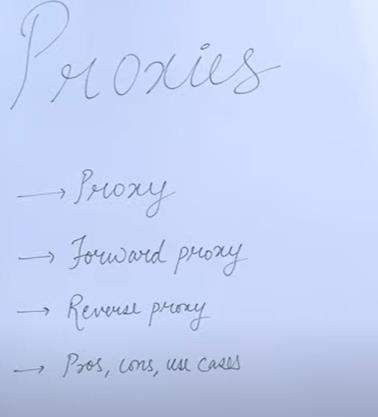
The two-tier DB architecture is a client-server architecture. The three-tier DB architecture is a type of web- based application. It contains mainly two layers- the Data Tier (Database Tier), and the Client Tier. It mainly contains three layers- the Data Layer, the Business Layer, and the Client Layer.

**In some cases (** complex system these layer are not enough)

So there is **caching** in b/w logic and data and LB or proxies in b/w client and application layer. These are the examples of n tier application



Lecture 4- Proxies

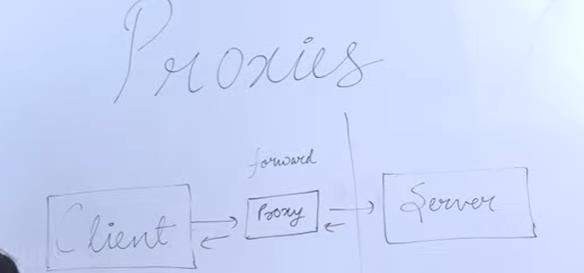


Proxies : - always think on behalf There are two types of proxy :

**Forward proxy :** proxy on behalf of client talking to the server machine sit b/w client and server towards the client side

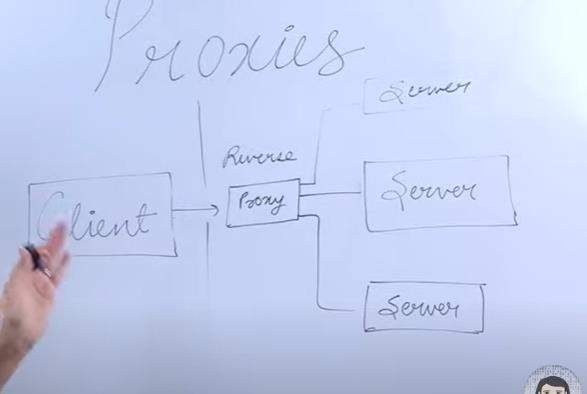
server doesn’t know about ip of client, server knows only ip of proxy.

This is useful in blocking the access of certain site and also traffic control if multiple client is there.



A forward proxy, also known simply as a proxy server, acts as an intermediary between a client (such as a web browser) and the internet.

**Reverse proxy :** proxy on behalf of server talking to client (example – spring cloud gateway)



Client doesn’t know about the ip of any server

**Useful :**

**for load balancing**

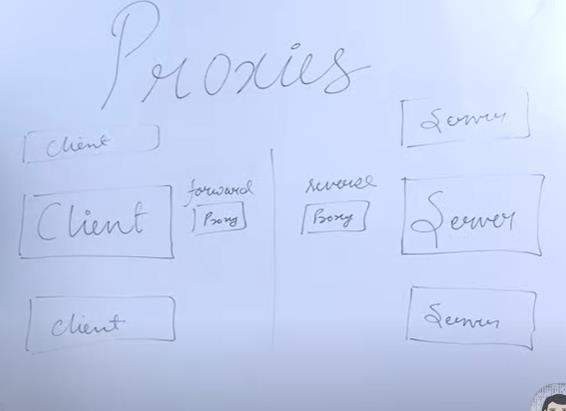
**caching the response from server**

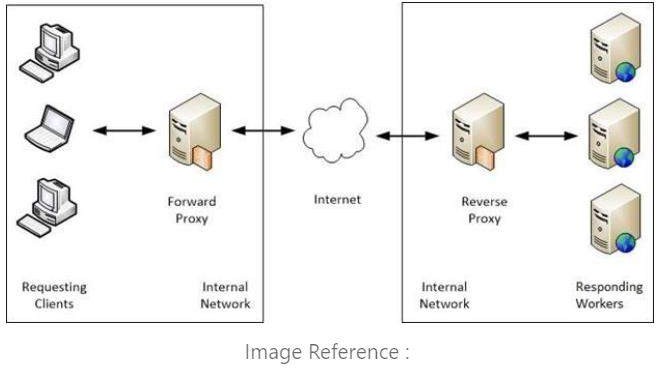
**only proxy server expose to outside world**

Spring cloud gateway

**Client side load balancing**

1. **Spring Cloud Gateway**: Spring Cloud Gateway is a lightweight, developer-friendly way to route requests from external clients to your backend services. It's built on top of the Spring Framework and offers features like routing, filtering, and load balancing, making it suitable for implementing reverse proxy functionality in Java applications.

**Proxies :** useful for security, traffic control d/a :- proxy single point of failure



Forward proxies and reverse proxies serve opposite functions and are used in different contexts:

1. **Forward Proxy**:

|  |  |
| --- | --- |
|  | * **Function**: A forward proxy acts on behalf of clients to retrieve resources from servers. Clients connect to the forward proxy and request resources from the internet, and the forward proxy forwards those requests to the internet on behalf of the clients. * **Usage**: Typically used within internal networks to control access to the internet, provide   anonymity to clients, and cache frequently accessed resources.   * **Example**: Corporate networks often use forward proxies to restrict access to certain websites, enforce content filtering policies, and improve network performance by caching frequently accessed content. |
| 2. **Reverse Proxy**: | |
|  | * **Function**: A reverse proxy sits in front of servers and acts as an intermediary between clients and servers. Clients connect to the reverse proxy, which then forwards requests to backend servers. The reverse proxy can perform various functions such as load balancing, SSL termination, caching, and security filtering before forwarding requests to the backend servers. * **Usage**: Typically used to improve security, scalability, and performance of web servers by offloading tasks such as SSL termination, load balancing, and caching. * **Example**: Websites and web applications often use reverse proxies to distribute incoming traffic across multiple servers, handle SSL encryption/decryption, and protect servers from direct   exposure to the internet by serving as a barrier. |

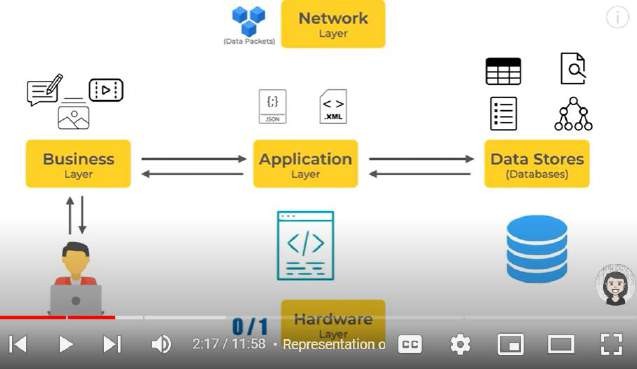
In summary, the main difference between forward proxies and reverse proxies lies in their directionality and the role they play in the client-server communication. Forward proxies sit between clients and the internet, while reverse proxies sit between clients and servers. Forward proxies are used to retrieve resources from the internet on behalf of clients, while reverse proxies are used to distribute client requests across backend servers and provide various services to improve server performance, security, and

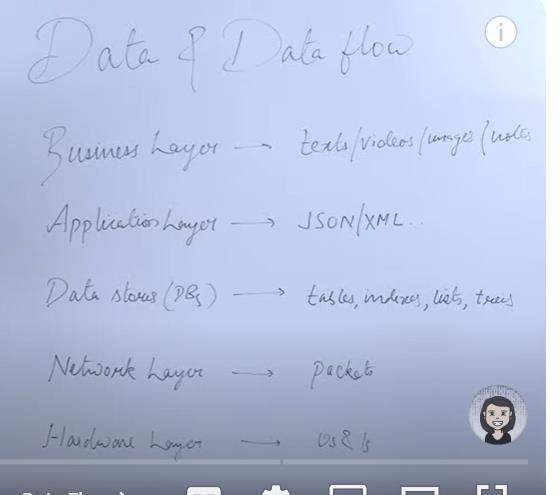
scalability.

Lecture 5: data and data flow



How data is represented in different layer in diff format.





Note :

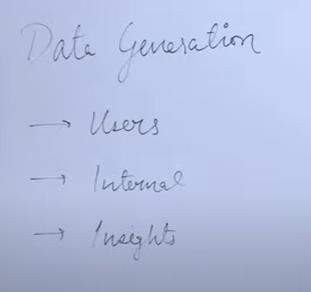
Always think wrt data.

Now understand different data stores and data flow



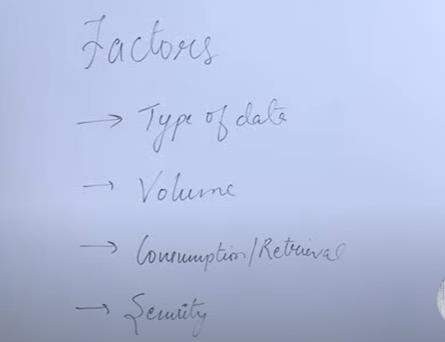


Data generation



Users :

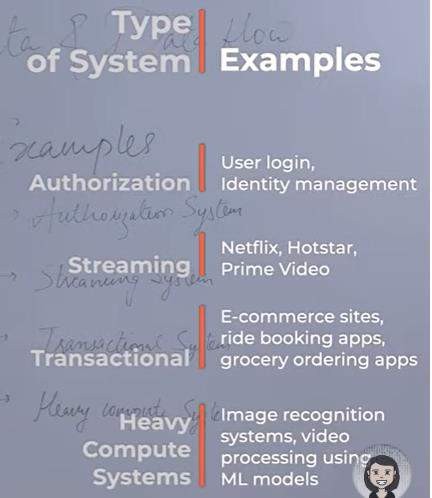
Internal : like saving logs (system populates of their own)

Insight: user buying something invoice will be generated. , in your tube history details Note :

1. **Knowing the type of data(like text,video) is important because it will tell which database we are going to use**
2. **system** which support terabyte of data is completely diff which support some GB of data.

Type of system and example :

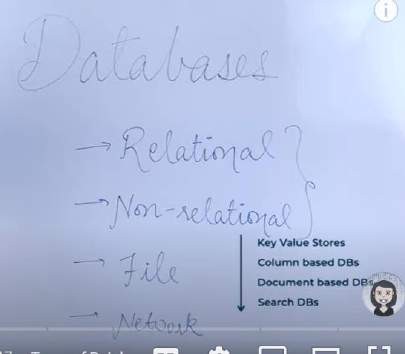
Authorization :- might not required more data(data volume is low) but level of security is high Streaming : data volume and data retrieval is high



**Lecture 6 : Databases types: SQL, NoSQL,**

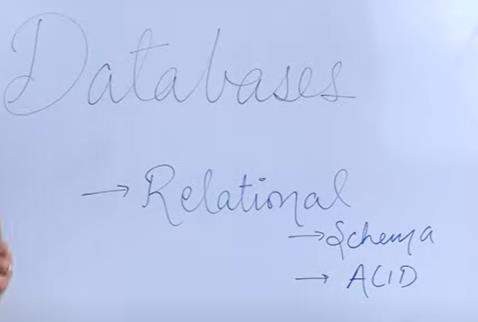


**some** popular d/b



Non –relational d/b again divided into 4 types

**Key value stores , column based, document based, search DB**



**There** are two factors schema and ACID which help to decide whether we have to select relational db Lets talks about schema

**Schema : refers to how data is going to be structured**

In relational db we have table and rows which store the data





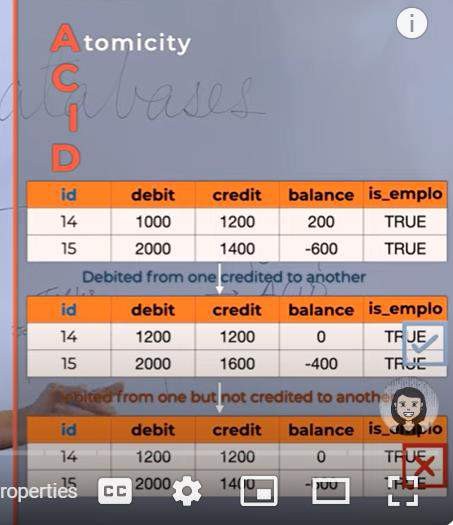
Advantages of relational d/b



ACID

A- Atomicity :-- transaction in d/b either happen completely or doesn’t happen it.

Example : transfer money from one account to another.



**Consistency**: at any given point of time the **state of d/b should be consistent**

If two request trying to read account balance , one read 200 and other read 300 this is not possible because d/b is consistent

**Isolation :** the two isolation does not know each other.

**Durability** : guarantee that transaction completed will survive permanently

1. **Atomicity**: Atomicity ensures that a transaction is treated as a single, indivisible unit of work. This means that either all operations within the transaction are successfully completed, or none of them are applied. If any part of the transaction fails, the entire transaction is rolled back to its original state, ensuring that the database remains consistent.
2. **Consistency**: Consistency ensures that the database remains in a valid state before and after each transaction. In other words, the execution of a transaction must preserve all integrity constraints, data validations, and business rules defined for the database. If a transaction violates any of these constraints, it is not applied, and the database remains unchanged.
3. **Isolation**: Isolation ensures that the execution of one transaction is isolated from the execution of other concurrent transactions. Each transaction appears to be executed in isolation, without interference or dependency on other transactions running concurrently. This prevents concurrency-related issues such as dirty reads, non-repeatable reads, and phantom reads.
4. **Durability**: Durability ensures that the effects of a committed transaction persist even in the event of system failures, crashes, or restarts. Once a transaction is committed, its changes are stored permanently in the database, and they cannot be lost or undone. This is typically achieved through mechanisms such as write-ahead logging, transaction logging, or database journaling

Note :-

If we have fixed schema which will not change too much and also transaction is there then will select relational db.

**Where we cant use this :**

If we are not sure about the column as the product evolve. **Vertical means :** we can increase the storage of one machine Horizontal scaling becomes difficult

In such of d/b schema is not fixed

k/v store : this is quite fast because most of the data are in memory. And this can be use in multiple caching mechanism

A key-value NoSQL database is a type of database management system (DBMS) that stores data in a schema-less manner, where each data item (value) is associated with a unique identifier (key). Key-value databases are designed for high-speed access to data and are often used for caching, session management, and other use cases where fast read and write operations are critical. Here are some examples of key-value NoSQL databases:

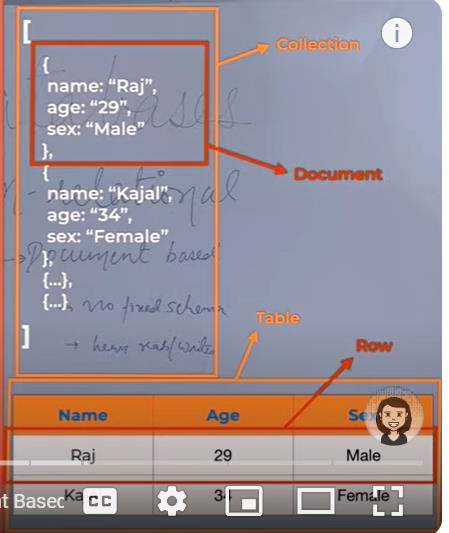
**example :- redis , memcached document based :**

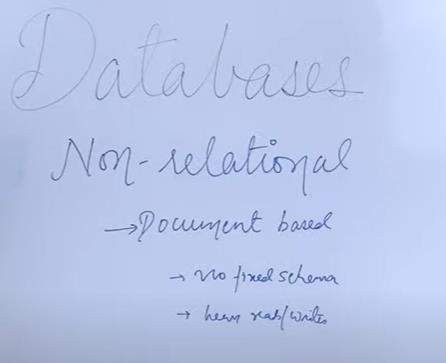
**no fixed schema,**

we can support heavy read and write and keep flexibility of dynamic data

example :- want to save products details in any ecommerce site

**table like collection and rows like document**





**Drawback :-**

We don’t have schema so we can have empty value or null values Do not provide ACID transaction sometime update become complex Benefits

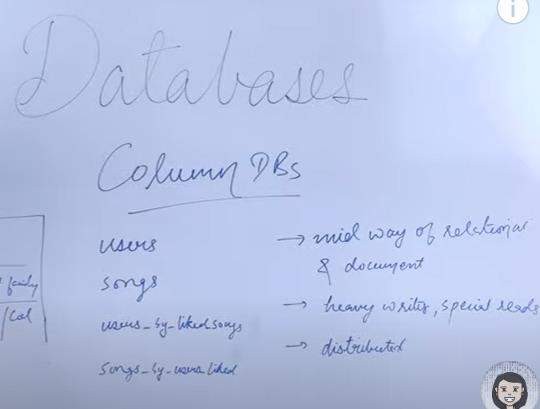
Example : mongo db

Column db : support huge number of write

**Example tracking**

Sort of mid way of relational and document db in a way there is fixed column and schema but this not support ACID transaction and it is use like song going on and we liking it so that some analytics can be done on it.

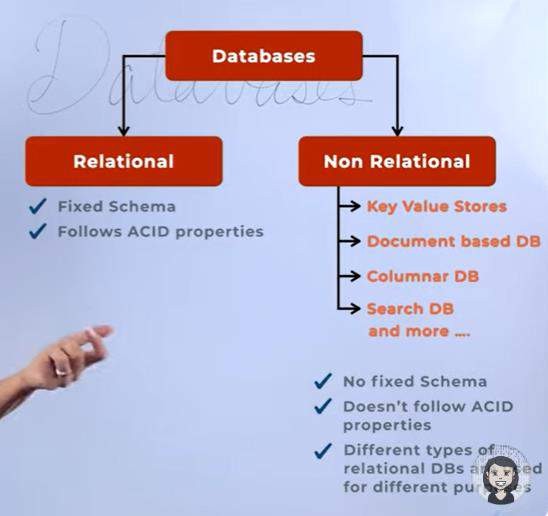
Sensor are sending data continuously in every 10 second.



**Example : Cassandra**



Images, video s3



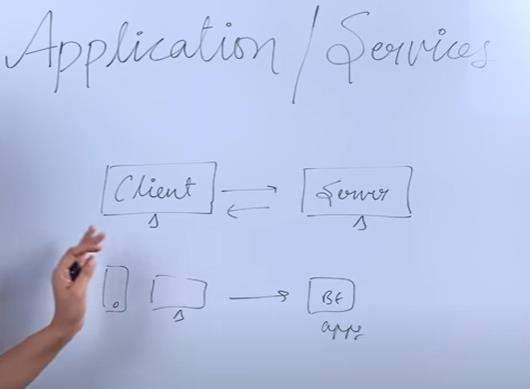
Relation : schema/acid No sql :-

Document based :- mongo db Search based:- elastic search

Column based :- metrics cassandra

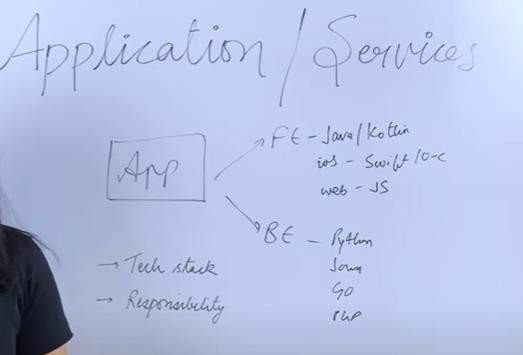
Key, value based redis

**Lecture 7 Anatomy of applications and services**

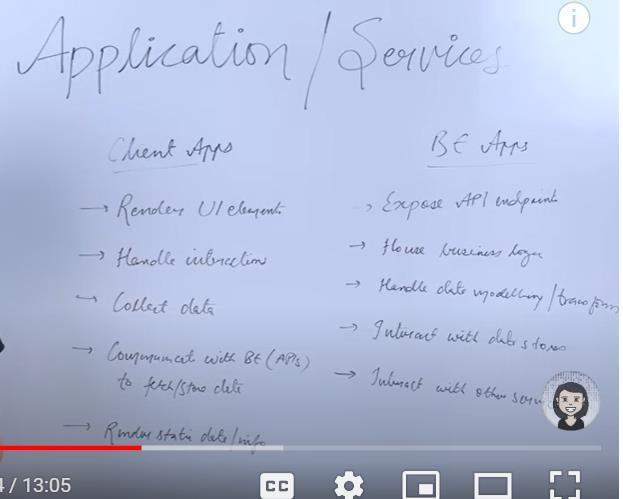


Application /services perform certain responsibilities like elevator in building. Application at diff layer perform diff responsibilities.

**Application is a piece of code** that is written in certain language



Application/framework



Design of parking slot

**First collect requirement** :--- parking slot is available

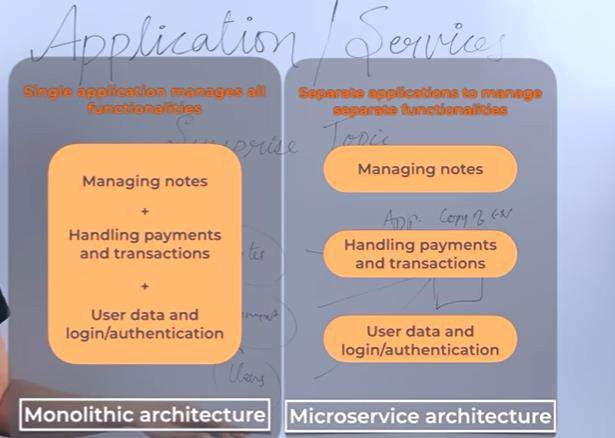
**Layer** mobile app, dektop app, web app

**Tech Stack** : depending on layer choose tech stack

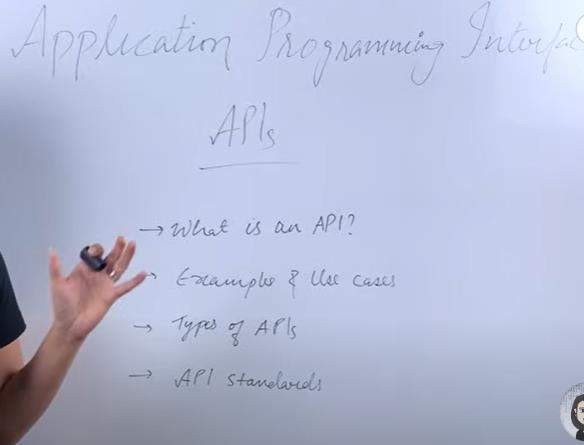
**Code structure/ design pattern :-** which we are going to use implement logic

**Performace/cost**



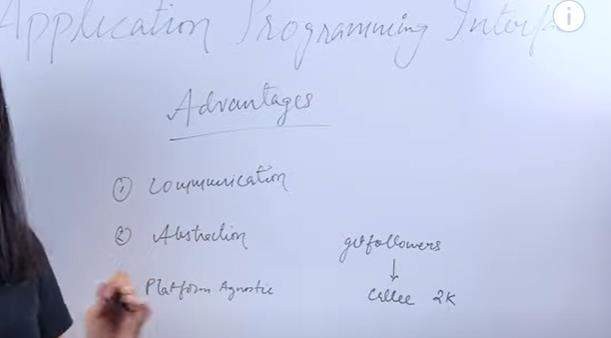


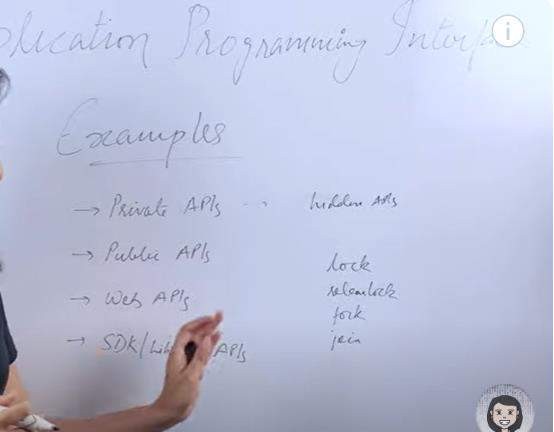
**lecture 8** – **Application programming interface**



**Application : a** piece of code have to interact with another piece of code Interface : think about abstraction

**Advantages of api**

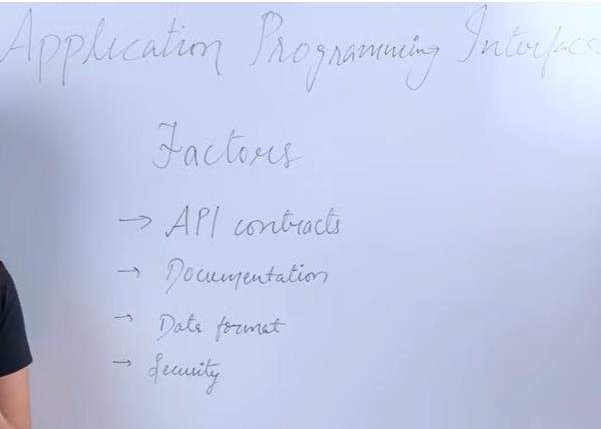




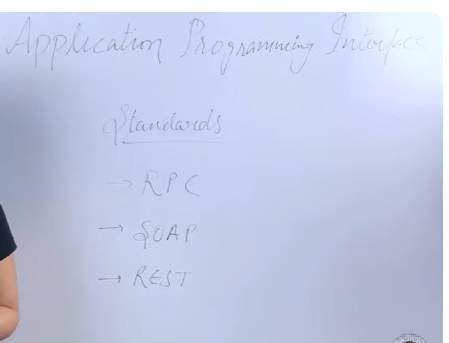
**Private api** : like in phone for payment Public api : google api

Web api :- superset of private and public like getFollower from instagram

Sdk library-----lock ----------thread based main advantage abstraction



**API standard**

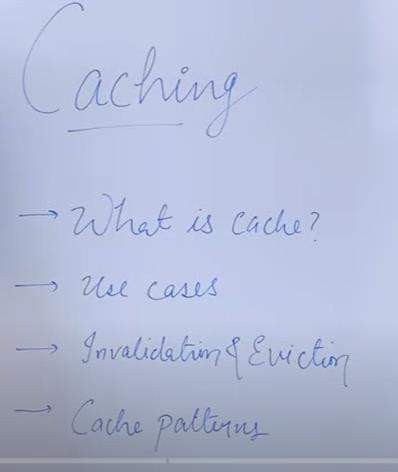


lecture 9 : **Caching | Cache Patterns | Cache Invalidation & Eviction**

**Cache hit Cache miss**

**Cache invalidate handle ttl**

**Cache evict**



**Cache :-** a h/w or s/w component which help in serving the data which is either frequently requested or so cache store the computed response and help in saving the expensive operation.



Saving the image in cache either in server side or client side

**Another example**

Client request to server and server get data from db and perform some computation and save in cache.

As request and response id. Another time request will come and it will check in cache if it is found it will called **cache hit** . suppose a new entry came and not found in cache then it need to fetch from db and it called **cache miss**





**What is cache invalidation and why it is needed ?**

It is needed because the data we keep in cache is going to change at some point of time. The data is changing and we have to update the cache as well and process removing old value and updating with new value is called **cache invalidation**

**How we will decide when we have to invalidate cache ? There are multiple method**

1 **cache expiry** ( and it is done through ttl)---time to live

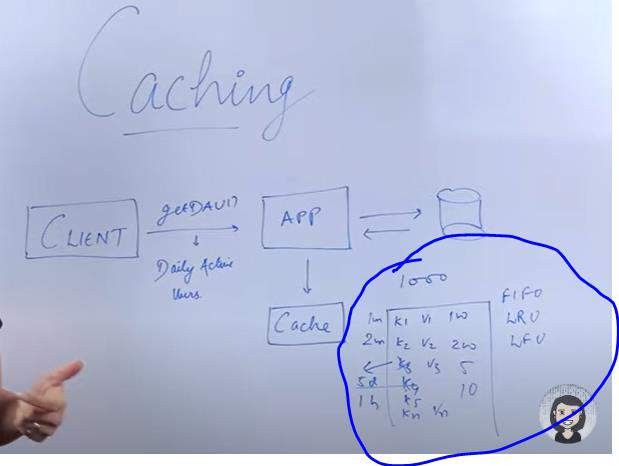
**Ttl will decide depending on the use case**

**2- :) if** cache miss happen then it will fetch from db and will update cache or whenever update occue in db then it will update in cache in this way cache miss will not occur

**What will happen if cache have limit of 1000 key and new key need to get update ?**

**In this case** any of the existing key has to be **evicted** so that there is a room for new key can be update. This is called **cache evict**

**There are multiple ways in which cache evict can occur ?**

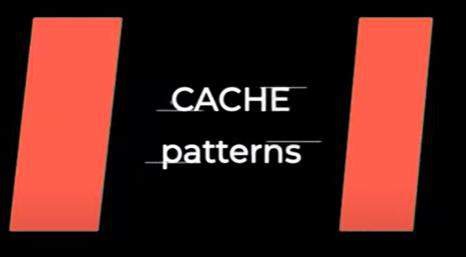


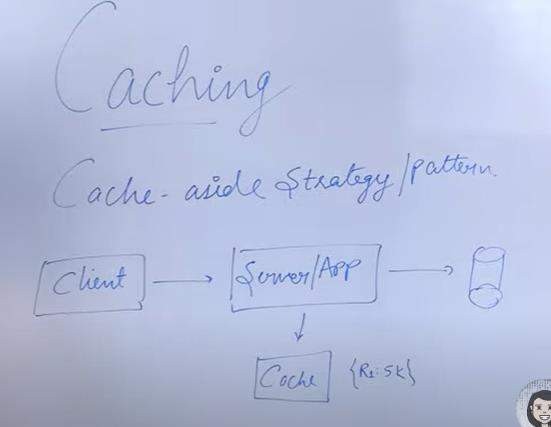
**FIFO :** first in first out

**LRU :** least recently used

**LFU :** lease frequently used

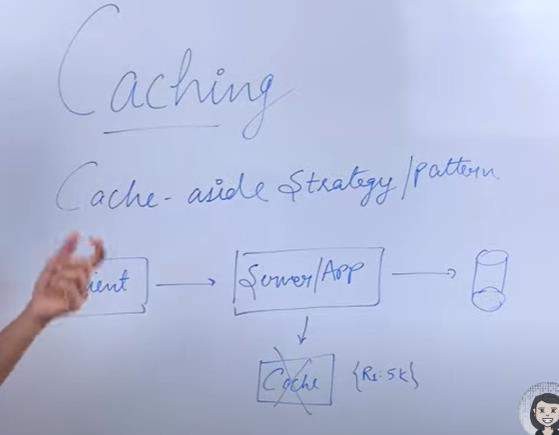
**Note : different type of cache provider implement different strategy of cache eviction**





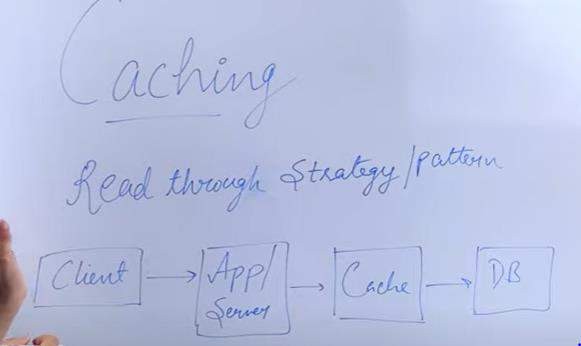
1. **Cache aside strategy pattern** : application always talk to cache and cache never talks to db Note :

If cache down whole system will not go down

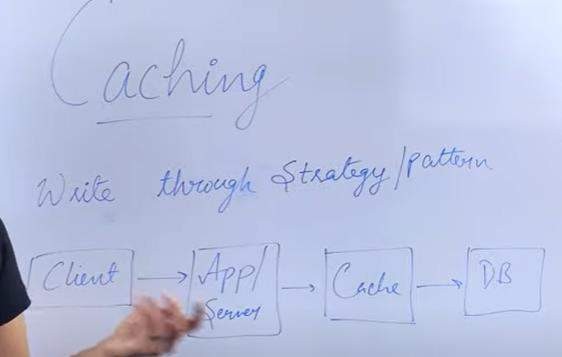


1. **Read through strategy pattern**

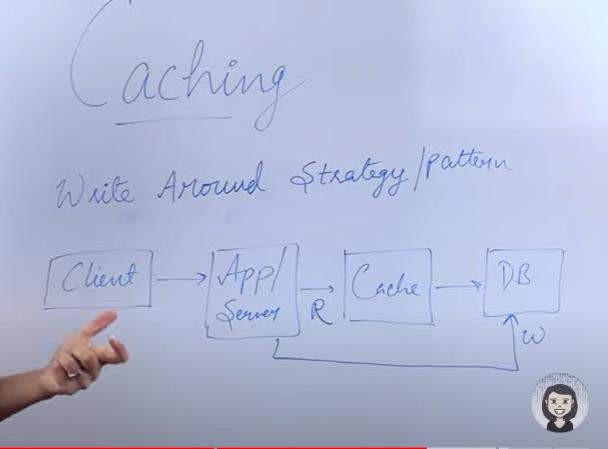
Application always talk to cache and application never talk to db so whenever first request will come there will be cache miss. Only cache will talk to db.



**Data modeling between cache and db is to be similar where in case of** cash aside strategy pattern modeling could be different since application keep data.



1. **Write through strategy pattern** : application read from cache and write to cache

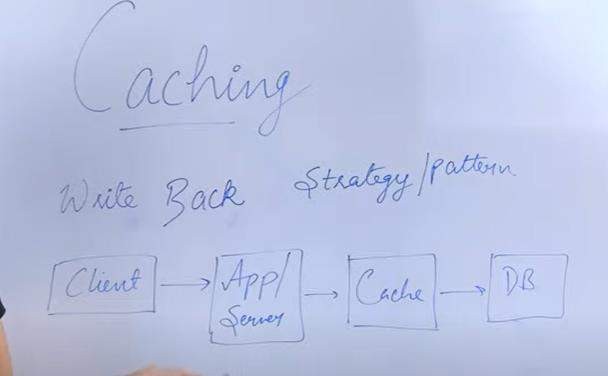


1. **Write around strategy pattern**

Application write to d/b while reading it will read from cache.

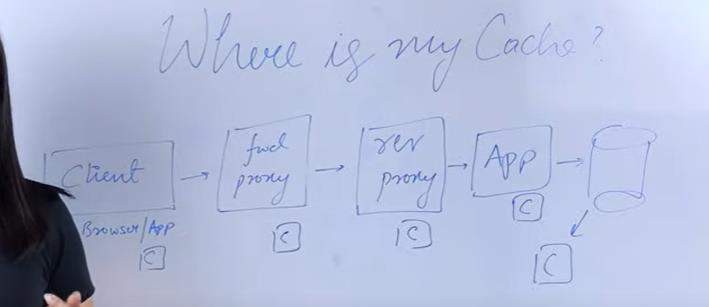
This can be use when there are lot number of writes and less number of read.

1. **Write back strategy pattern**



**Write data in cache in bulk then from cache write data in db in bulk**





**Note :**

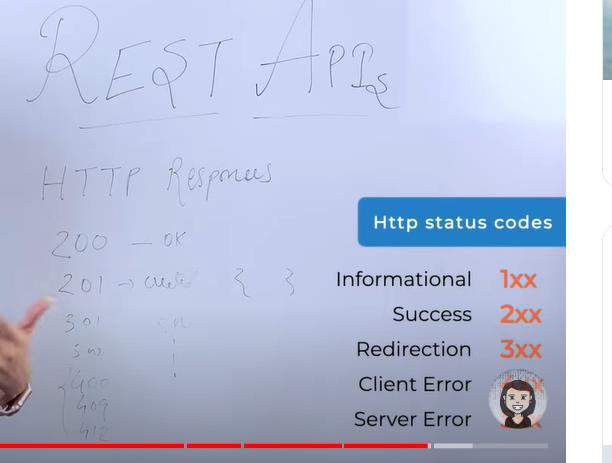
Depending on the requirement cache can be sit in any of the one or more than one

**Lecture 10 What is a REST API?**

**Stateless:-**

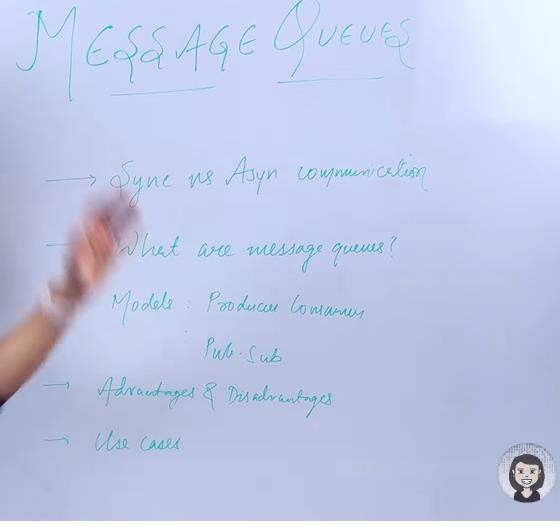
Suppose 3 client requesting the server to get date then server will treat all client in the same manner i.e it is stateless. Server should not know state of client in any point of time.

**http status codes**



**Lecture 11 : Message Queues | Producer Consumer Model |**

**What will discuss**





Synchronous : example talking over a phone

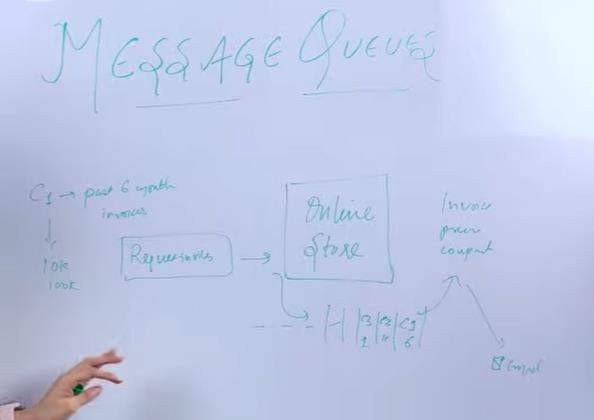
Asynchronous: example sending mail

Synchronous : -- client request to server and server need to response immediately

**Asynchronous** : there is also a requirement when client send a request to server for processing and server process that request in asynchronous manner but don’t need to response immediately.



Think in term of bank . 25 customer are there will get a token and when our token display then will go. This is asynchronous communication not immediate response we need to wait in queue .



Customer request for 6 month invoice, online store put it into queue and invoice component pick it up from queue and send to customer using email.

One component pushing the messages into queue . here is the customer want invoice please download it and other component reading the message from queue.

So this is how message queue works



Queue could be a process which has data structure in memory to store the messages. It could be a process running on same machine or different machine.

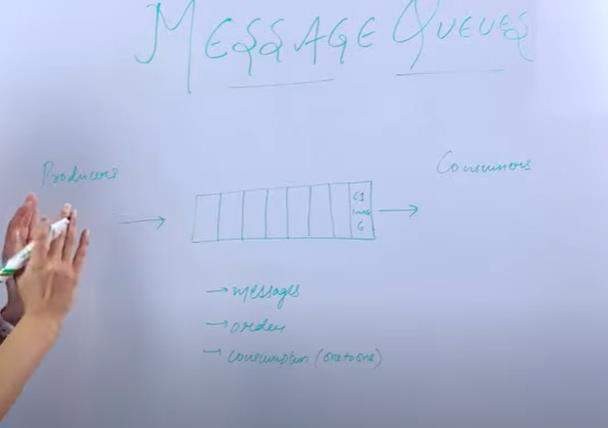
**Ex : SQS, kafka :-** build for handling highly scalable and complex system

**Queue consist of messages :**

**Messages** are short size data which jsut tell what has to be done , what task has to be accomplished.

**Producer :** the component which add or send message to the queue is called producer. Producer are the component which tell what has to be done.

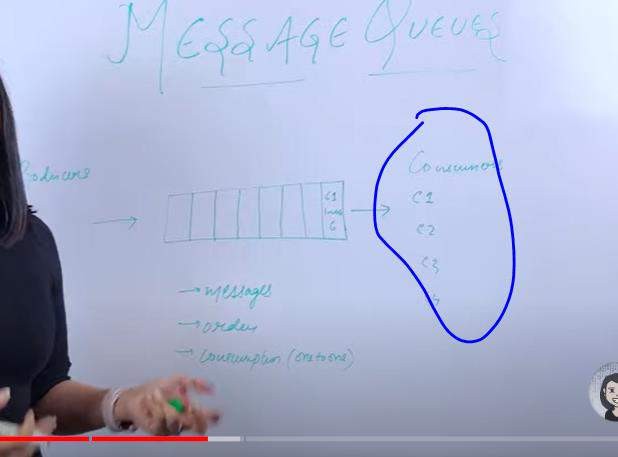
**Consumer** : will pick up the message from queue and do processing



**Advantages :**

**We can take a lot of request and keep it in queue :** means we can handle a lot of load

This can be scaled . When producer sending lot of messages in the queue then we can increase the number of consumer



Suppose consumer is down then actual data or processing never loss it will saved in the queue when consumer come up it will process again



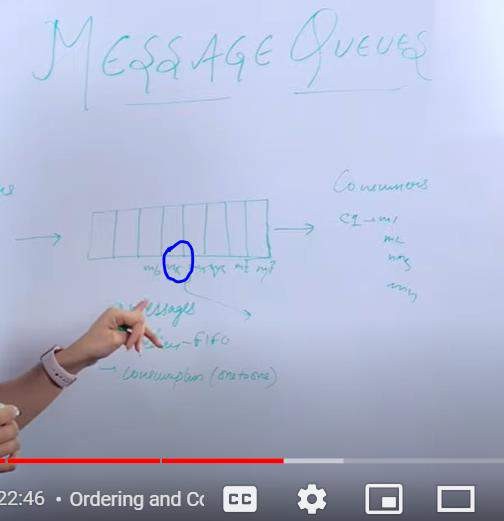
The message push by the producer in the queue will be consume only once or will be processed only once. So once a consumer consume the message it will be removed from the queue.

**One to one : this is one of the model of message queue Note :**

**Ordering** of message depend on use case **for example :- in chat use case** order is matter.

In case of invoice generation order doesn’t matter.

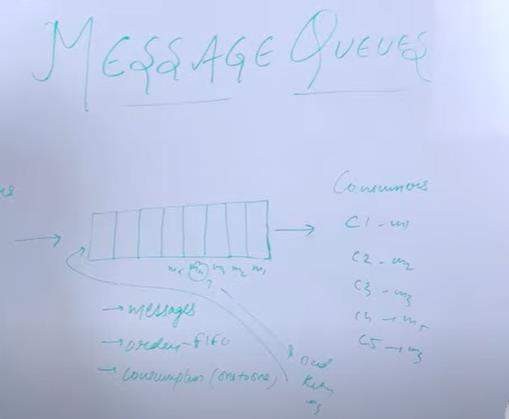
**Let see how consumption take place in order and unordered message ? Order :---** consumption take place in FIFO



Suppose due to some reason m5 is not processed **in order queue** then it will blocked and other message will not processed it until m5 process it.

**Let see what happens if the queue is unordered**

**M3** not processed it will move to DLQ and failure of message does not stop the consumption of the messages.



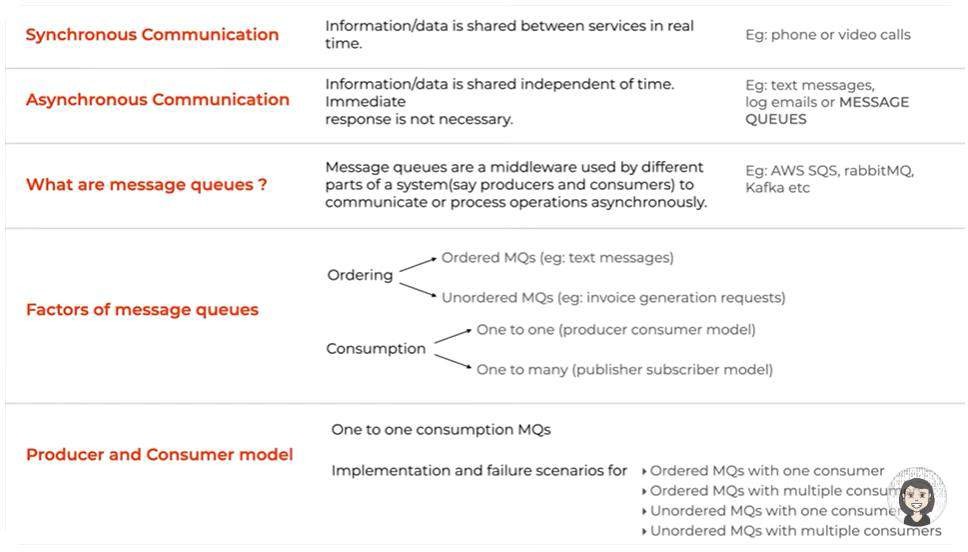
When we want to handle lot of load we can use it messaging queue

**example**

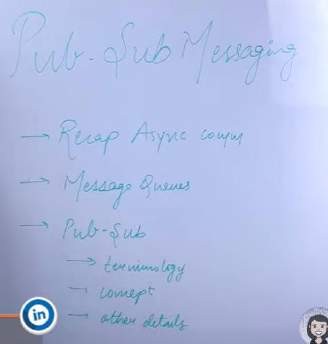
Sending an email

Order has been placed sending an invoice.

**Producer- consumer model the message will be consume only once**



lecture 12 **Pub Sub Messaging**

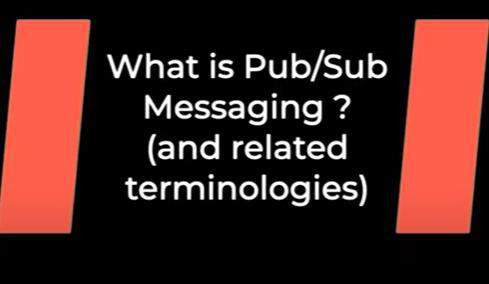


**Pub-sub** is one of the model of message queue

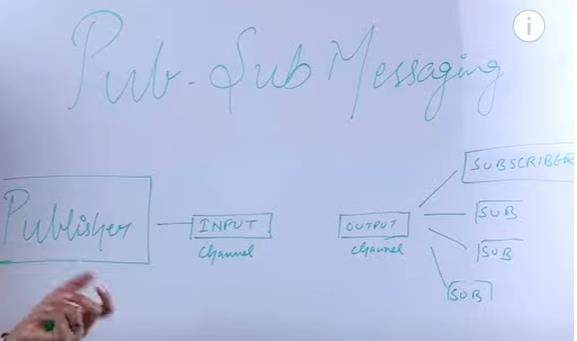


**Event will be consumed by either a group of people or particular group of people or all of them**

Example – think wrt hospital an emergency case come



Publisher :-- tell about the event whoever take the responsibility bringing from outside to into the system that is called publisher.

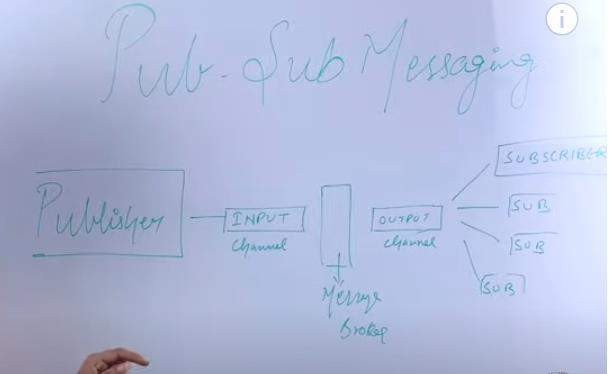


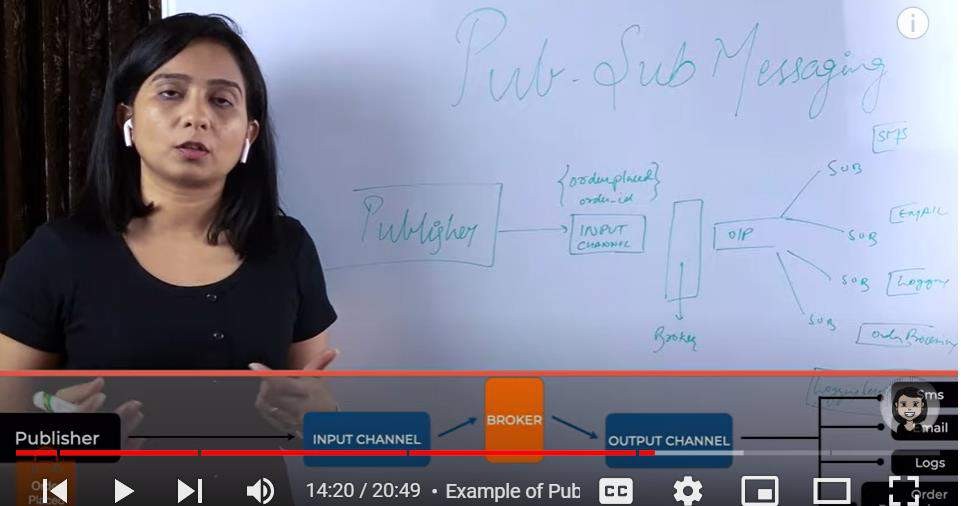
Publisher publish the event into input channel subscriber subscribe the output channel and they react accordingly. There can be diff output channel for diff subscriber .

There is a need of **message broker** which takes care of multiple responsibilities

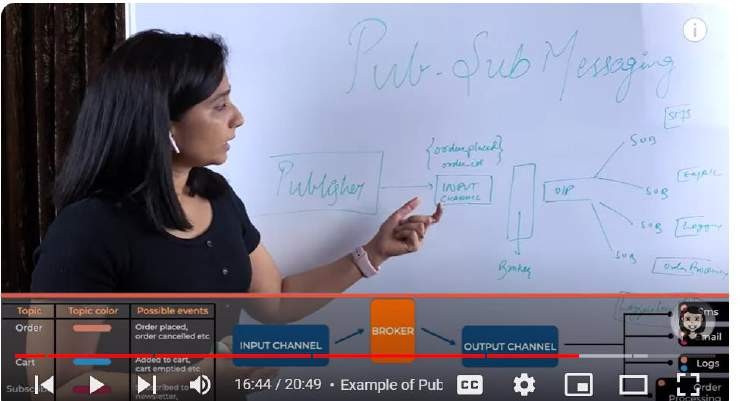
1. so one of the responsibilities publisher is publish the event in some format may be that data or that format need to be modify or it has to be enriched with some other data and then it has to be pass in output channel so message broker take care of that responsibility.
2. message broker can divide the message into different partition

Example : some emergency came so publisher publish the message(there is event of emergency case) but they don’t know where to send it so message broker route the message and subscriber are like doctor.





When order is placed publisher send to input channel and broker will add some more info like timestamp once it is placed in output channel then subscriber will react accordingly

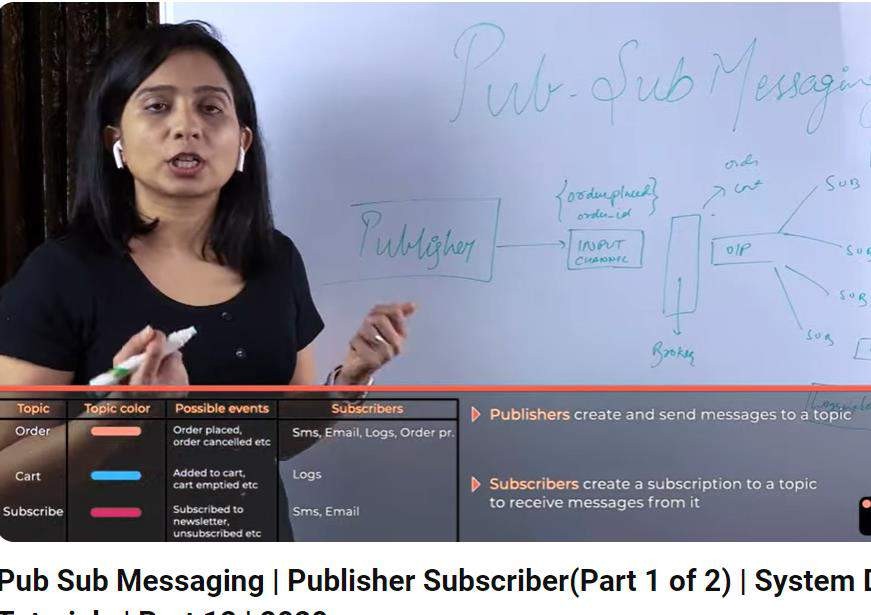


Subscriber don’t need to subscribe all output channel that’s why all these event divided into different

topic . think like **that topic is subset of messages**

**On these** topic there could be different output channel on which these subscriber subscribed.

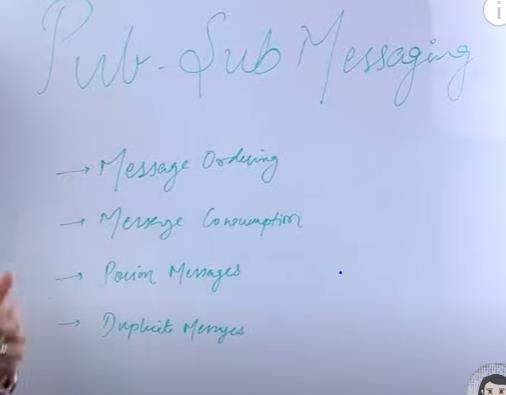
**There could be order topic, cart topic etc**



**These subscriber perform in asynchronous manner.**

This allow scaling of the system very easily.





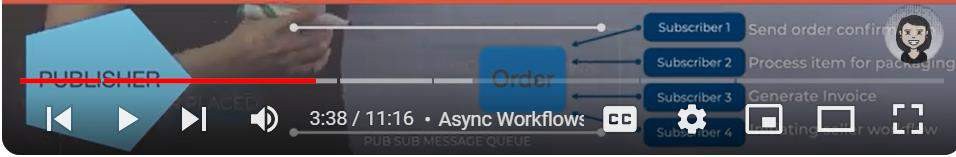
Ordering is not guarantee.

We can achieve ordering using priority queue

Lecture 13 : pub- sub part 2

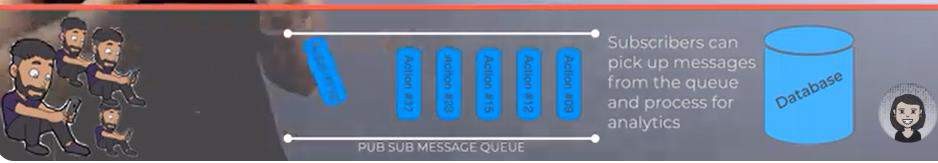




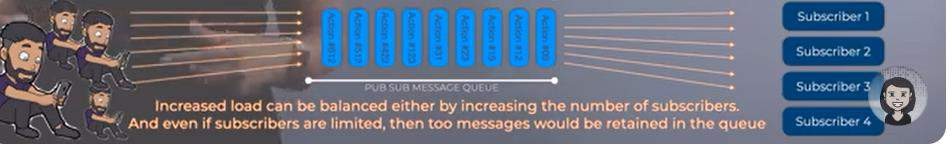




**Website want to track every click and every action taken by user in website** and want to save data in backend for some analytics



Message queue help in load balancing





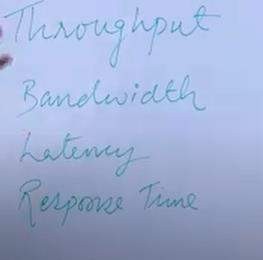
Like uploading all invoice in s3 at the end of the day.



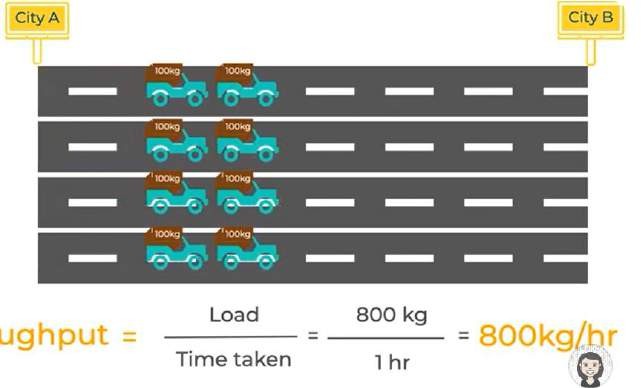
Note :

Same pattern can be coupled with other pattern (like acknowledgement from subscriber)

lecture 14- Performance metrics

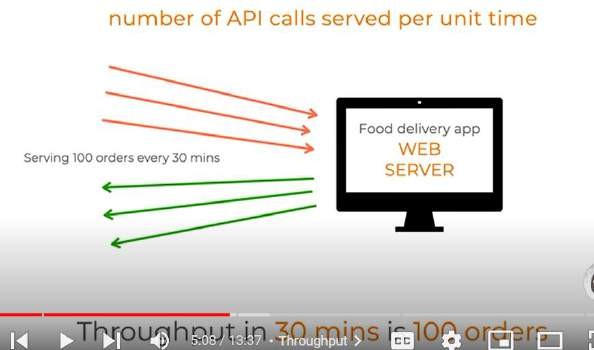


Throughtput : some amount of work done in a particular time Example : chop 20 onion in one hour ( this is throughput) **Throughtput =load/time taken or work done/time taken**



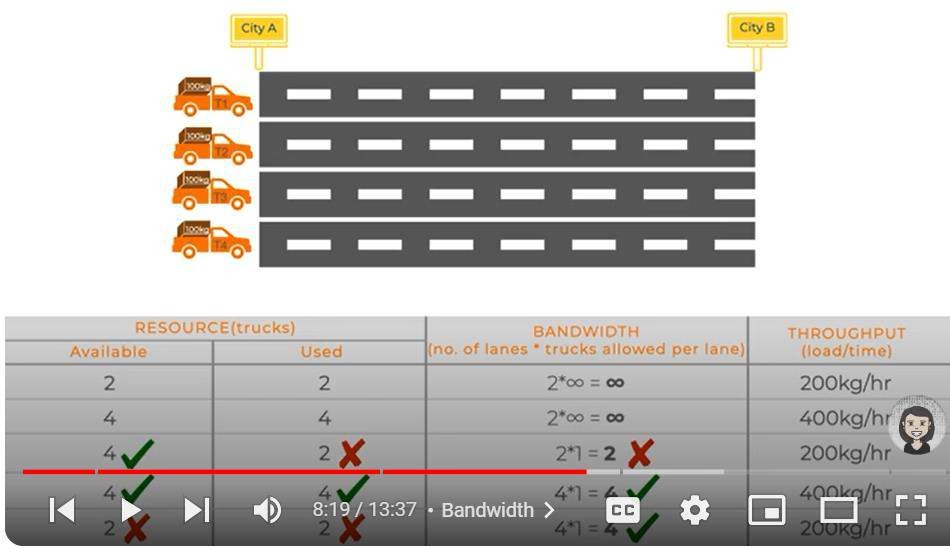
**In terms of system design throughput :** number of api calls served per unit time

**Example :-**



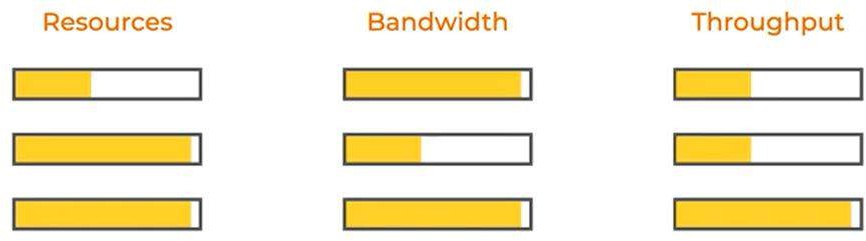
Bandwidth : come into play when data getting transferred over network for multiple location. Example : for Netflix streaming has to happen over diff location

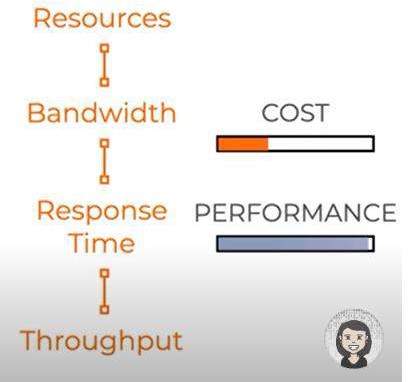
**Network bandwidth** is a measure of the data transfer rate or capacity of a given network.



Note :

If we have higher bandwidth and resources to use it then we have higher throughput.





**API response time** is the total amount of time it takes for an API to receive a request, process it and send a response back to the client

**Diff b/w latency and response time**

**Latency refers** to the delay in the system, while response time includes both the delays and the actual processing time.

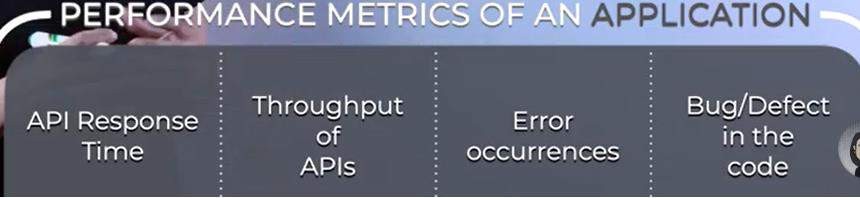
**each resource** is treated as a separate entity that can be accessed and manipulated independently

**lecture 15 Performance Metrics of components in a software will** talk performances of diff components of a system.



**Will use diff metrics to know the performance of diff component of a system.**

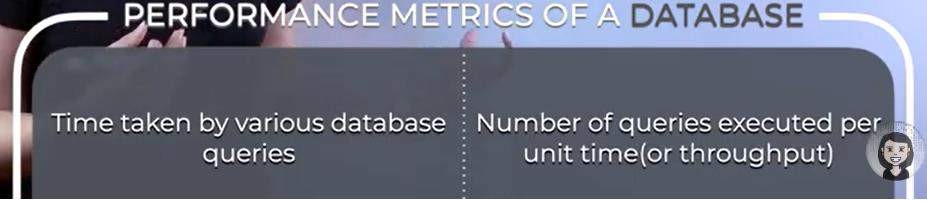
**Also** above component(like application,dbc ,caches,messages queue) running in instances



**Good performance of application :**

Response time is low Latency is low





If d/b can perform **huge number of read and write** means it performance is good.

In relational d/b we need to design a **good schema** and write **less complex query**(using join more)

If data is more in db then snapshot need to be taken and replica need to create in order to reduce the load on db



**Cache** is in memory key value store fetching the data from the caches quick Latency of writing to cache is low





Here we need to make balanced b/w throughput and number of consumer.

More failed message should not be there in queue otherwise throughput will decrease





Every component running on some instances

**2 basic requirement**

**Application : Memory and cpu**

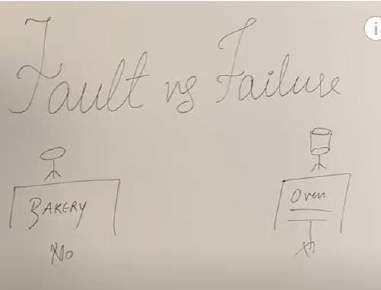
**Databases : memory and cpu (** databases take lot of memory)

Lot of data coming in system , we need to suppose delete old data but not deleting it so memory usage will go high then change disk size



**lecture 16 : Fault and Failure in distributed systems**





We ordered cake in bakery and in evening we went to collect it but oven broke so this is system fault and bakery person fail to give us cake so this is failure

**Note :**

**Fault : is** something which is beyond our control and in result of that fault a **failures** occur where the customer cannot be served.

**Fault is the cause , failures is its effect**

Example : - wire cut light gone ( wire cut – that is system fault)

**Now what could be done here in order to avoid this problem**

Even if oven not work and bakery person do something so customer doesn’t go empty hand and the birthday party

still go on. Bakery person will give alternate choice sorry we didn’t serve your order but here some other cakes. What could be done to avoid fault , there could be multiple oven

**Performance can be improved :---- 1:- tolerant of fault**

**Let understand fault and failures in context of distributed system**

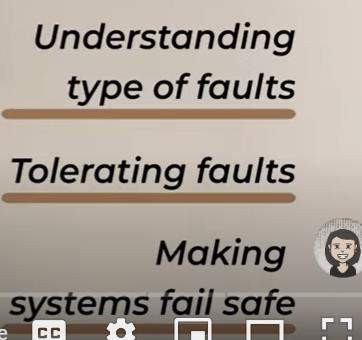
**Examples of fault**

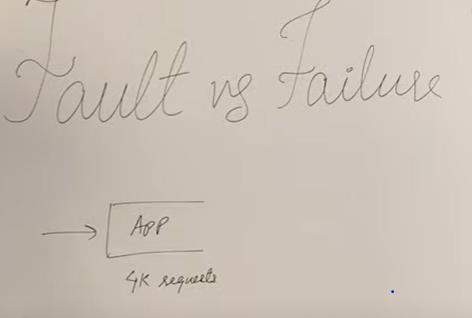
Network fault :- network time out

machine crashes , h/w issues , memory issue system is running out of memory and human errors like software

bug

because of this system as a whole get infected





There can be multiple type of fault:- Hardware fault, bug

**Suppose this app serving 4k request and instance goes down or cpu usage high because of this fault , failures occur**

**Amount of data sending in network that is huge and network time out** that is another type of fault Imp

**In order to deal with h/w fault** : we have multiple instances so that other instances can serve it **Now** what happen if there is a bug in a code , whether we have 3 or 4 instances it doesn’t matter. **Solution : test** it and give graceful message to the client

The same issue can happen in cache, db etc

**Solution : replication is one of the way**

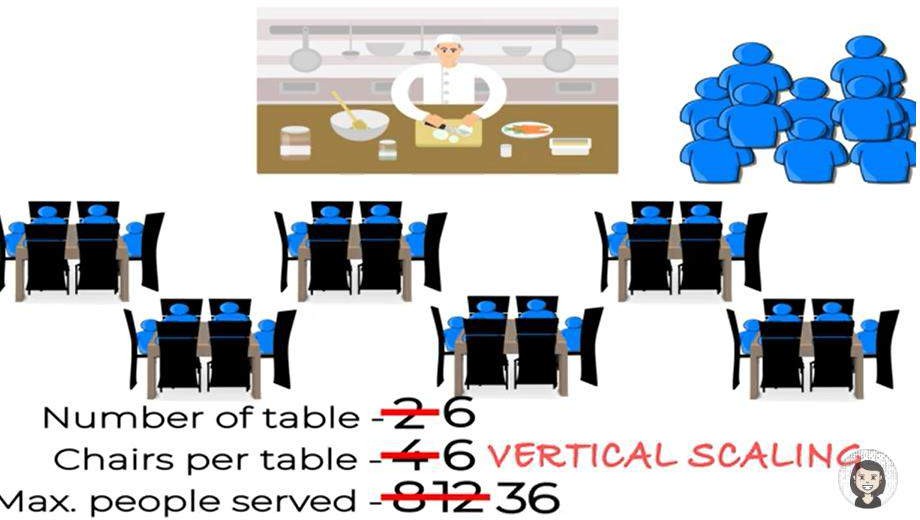


If plane have to land in emergency then live jacket is there which can save life so this is fall safe mechanism. Using that in mindset we have to build system

Fault tolerance is a process that enables an operating system to respond to a failure in hardware or software. This fault-tolerance definition refers to the system's ability to continue operating despite failures or malfunctions.

lecture 17 :--- scaling





**Vertical scaling** : increase the capacity of existing resource

**Horizontal scaling :** increase the number of resources

**Scaling** : in term of business ( every year business will increase so we should have the resources)

**Building scalable system means** : serving more customer, system should not be complex and performance should not be decrease

**Scalable :------------ Increase load**

**Not complex**

**Performance should be high( performance should not take a hit)**

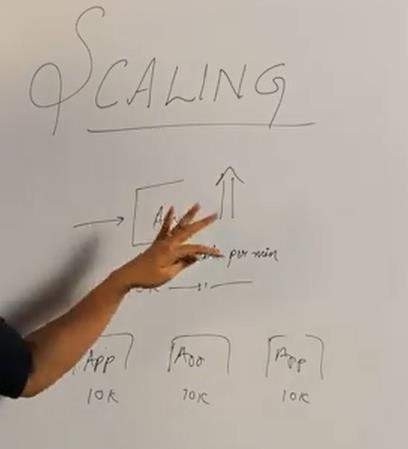
If all above box ticked then we can say scalability is maintained

**In terms of system**

**Increase or upgraded the power of resource when request increase ( this is vertical scaling) increase memory**

**and cpu**

**We increase the power and we can increase multiple instance**



**Business is less :** vertical scaling is less complex

Horizontal scaling : slightly complex , need to handle load balancer Note :

**Scaling is a concept** it is applicable for application, db , cache etC

lecture 18 Database replication

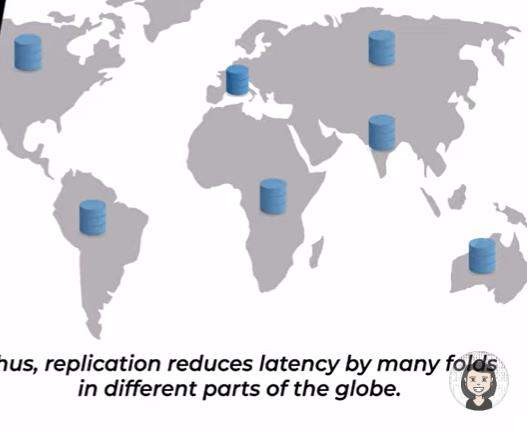


**Replication means :** to have a copy **Primary/master** : the db which has data **Secondary/slave/replica** : copy data of primary

advantage

**Note** :

1. :) if master db goes down then replica db will take over
2. :) having replica reduce latency



3) we can use replica for read one

**But there are lot of complexities come while handling the replicas**

**Replication lag : is the time it takes to copy value from primary db to secondary db or the replica**

**But if replication lag is more then there is a problem if there are n number of replicas then how we will solve it. ?**



Write : at t1 Read : at t2

**If replication lag > t2-t1** in that case if someone is reading from replica then it will get inconsistent data.

If someone reading from master then they will get x=1 but if someone reading from replica then they will get x=2 so **this is the state of inconsistent**

**Now the challenge is to solve this consistency issues**

**The way these problem solve are known as consistency model or consistency algorithms One of the consistency algorithm Read after write consistency**

In this case **replication lag is** zero because master is saying to all replicas to update the values. This is also known as **synchronous replication**

**Advantages of synchronous replication :**

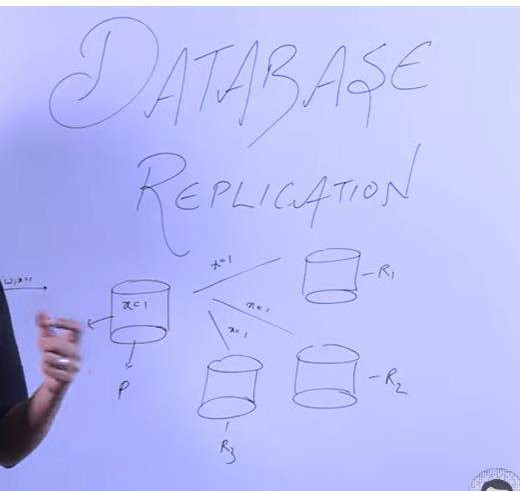
**Replication lag is zero**

**Data is always consistent in the system Downside**

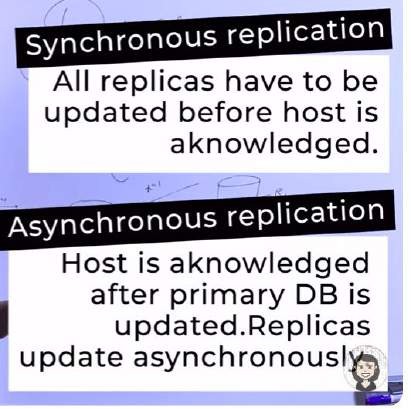
**Performance might take a hit (** because for every write replicas need to get updated as well as for acknowledge) The latency of whole write system is going to be higher

But id one replica fail to update value then is failure scenario **drawback of synchronous replica**

**How to solve this problem Asynchronous replicas**



Master db will not wait for acknowledgement but if some replica fail it will go into inconsistent state



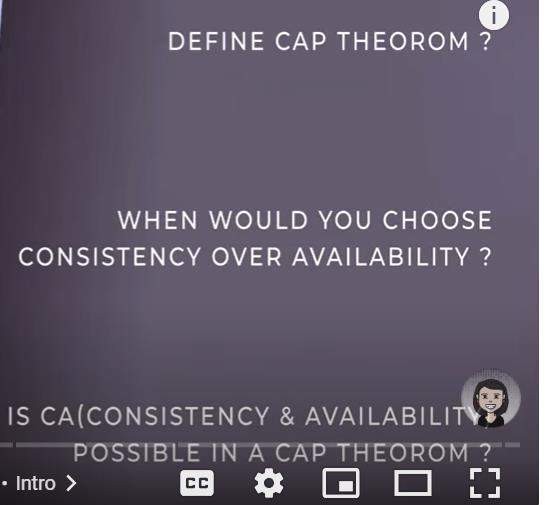
Hybrid replicas

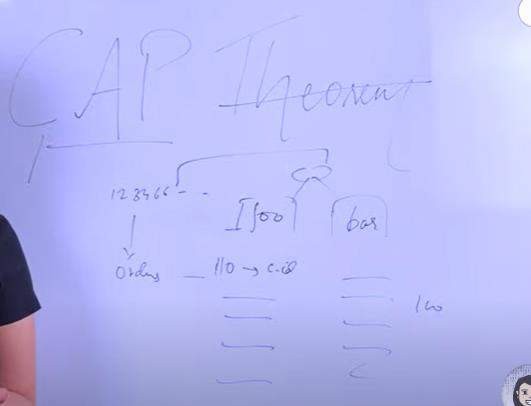
Master db will wait for only one replica to acknowledge that write is done

**Snapshot : is state of database** :

If database goes wrong we can go to one of the state Replica : is happening periodically

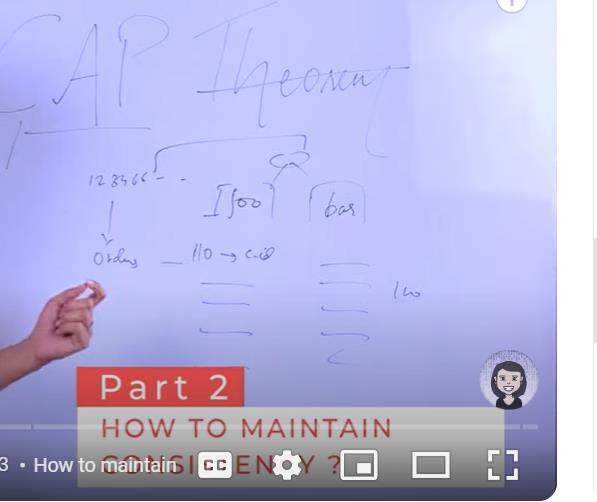
**Lecture 19 CAP | Consistency, Availability and Partitioning**





Example :

In lockdown we start a business for delivery food . two person taking order in same phone number but don’t know list of customer of each other so this is inconsistency but cap theorem told **consistency** should be there in the system



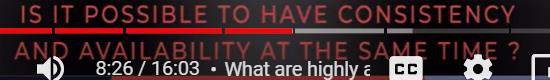
How to maintain consistency between me and partner

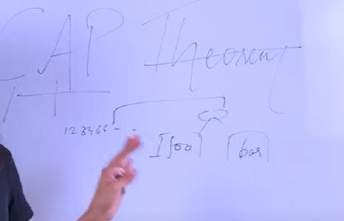
**Using replica we maintain consistency problem**

**After** fixing consistency problem what do u mean by availability If one person got sick then service is not available

Note : if we have more partner then overall availability will increase

Suppose 10 person are there to take order now our system is highly available but **what about consistency**





For now let there are 2 partner both are available , consistency is there both are passing order to each other but two partner passing info to each other by some communication channel which is **broken** this is called **partitioning** .

But if we can tolerate this partition after this consistency and availability maintain then this is **called partition tolerant**

**Due to** communication link availability and consistency maintain



**Very important**

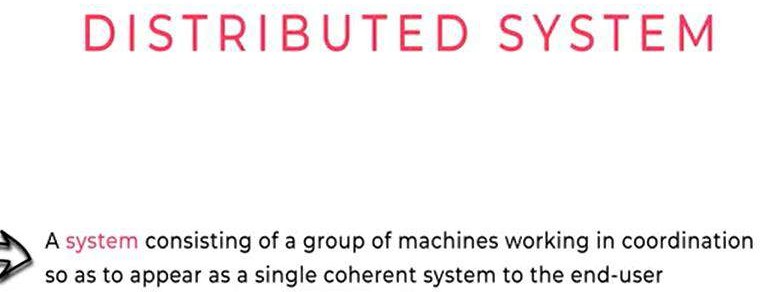
**If partitioning is there then system will have either consistency or availability**

**Lets talk CAP in terms of system**

Database and replica deployed in different instance which are connected with each other via some network . if network break (partitioning occur) still availability is there but consistency is not there.

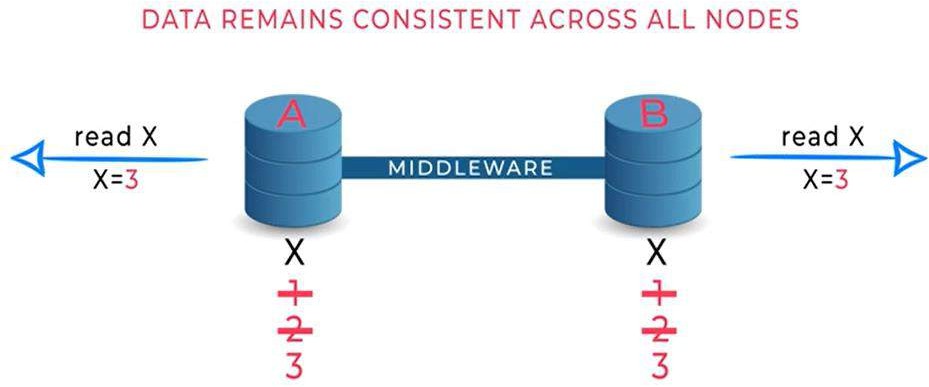
To provide consistency replica need to closed.

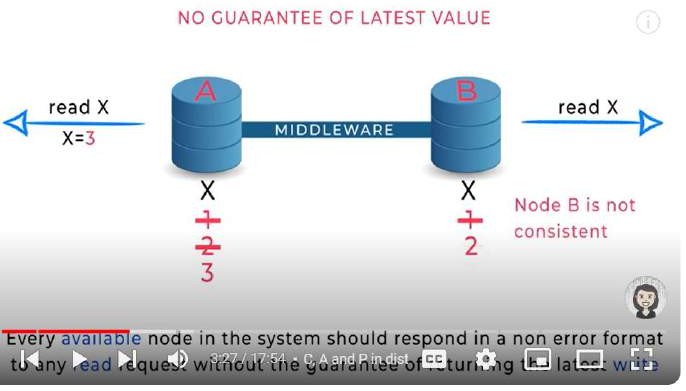
**Lecture 20 : What is CAP Theorem? Degrees of CAP theorem, use cases**

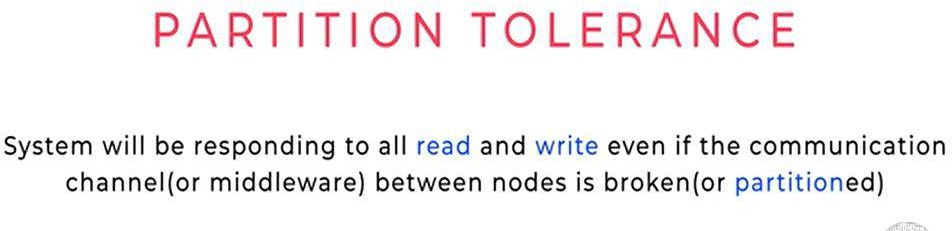


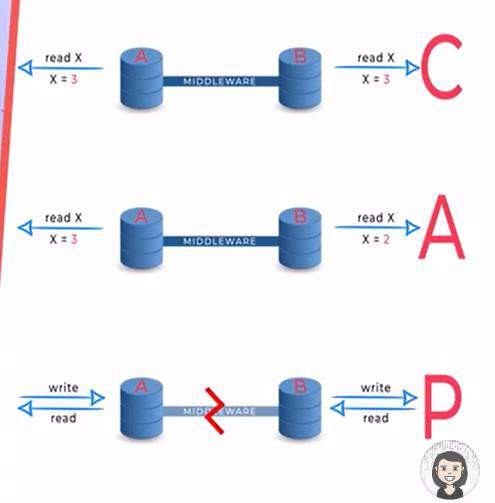
**Consistency is defined as :**











Consistency :-- return latest value Availability : not guarantee latest will read

Partitioning : even channel break , read and write will not stop

**The CAP theorem** says that a distributed system can deliver on **only two of three desired characteristics: consistency, availability and partition tolerance**.

Note :

**Partition tolerance** is something which happen due to network failure. Partition tolerance is mandatory

Property hence we can compromise with partition tolerance. Now we need to choose either consistency or availability



**Degree** : means even we cant provide 100 percent consistency but we can provide some level of consistency.

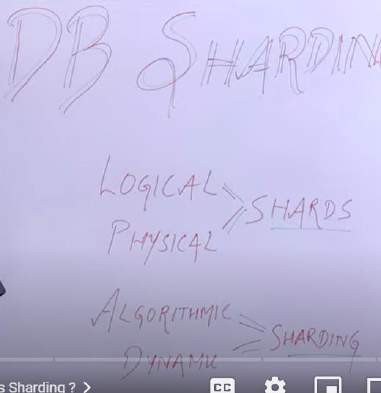
Totally depend on use case :---

**Atm machine : - consistency is important**

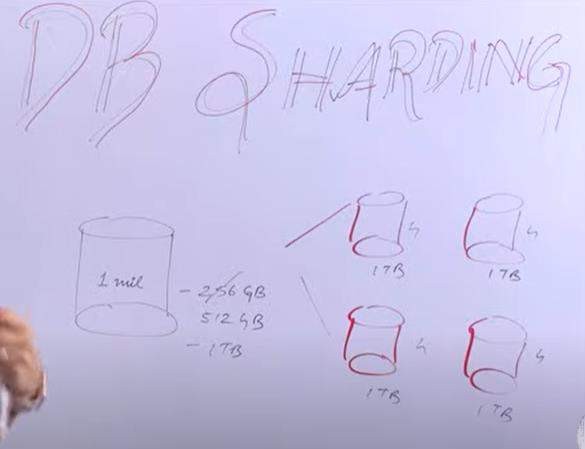
**Youtube :** availability we can choose

**Lecture 21**

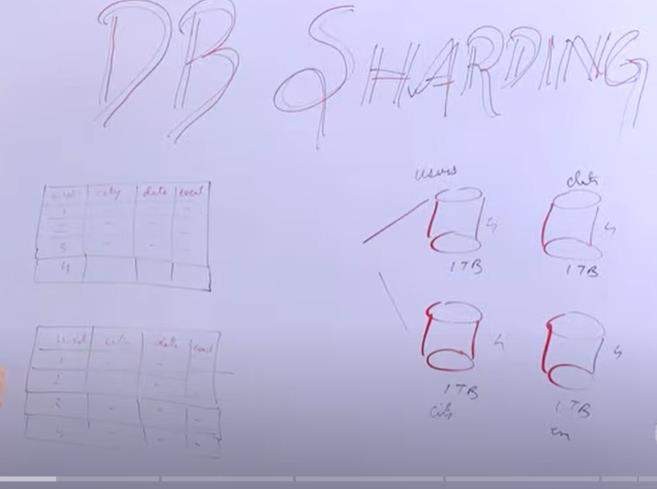
**What is Database Sharding, Logical and Physical Shards, Dynamic vs Algorithmic Sharding**



**Data is sitting in 4 diff machine .**



Example : Relational database



In 4 database we are saving one one column :

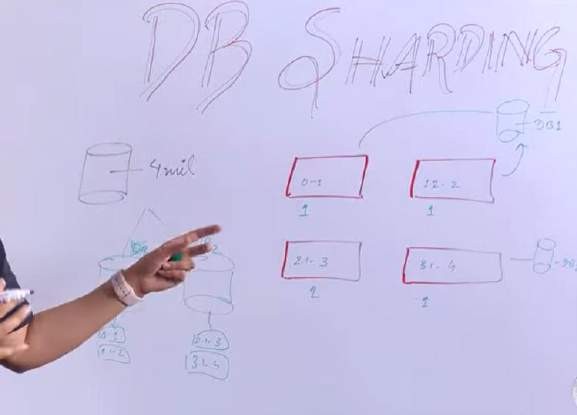
**So partition on the basic of column is called vertical partitioning Storing different rows called horizontal partitioning**

**Horizontal partitioning is known as sharding ( sharding means partition)**

What is sharding in a database?

Database sharding splits a single dataset into partitions or shards. Each shard contains unique rows of information that you can store separately across multiple computers, called nodes. All shards run on separate nodes but share the original database's schema or design.

**Sharding means : partition of data or subset of whole data That** subset could be logical or physical



**Shards. The partitioned data chunks are called logical shards.**

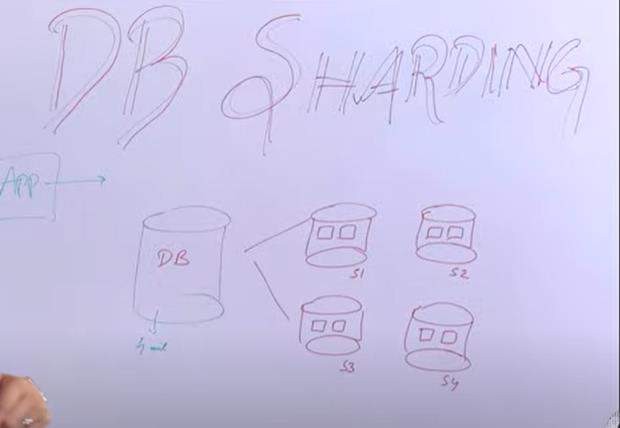
**The machine that stores the logical shard is called a physical shard or database node. Advantages of sharding**

Searching of query will fast

We can put d/b sharding in diff physical location.

**How we will decide on what basics will do partition ? There** are various sharding strategies

**Note : there are two sharding** :-----

-

**algorithm sharding** : there is some function in app which knows in which sharding read and write request need to be routed.



and **dynamic sharding :app will ask to module**

**algorithm sharding :** app knows

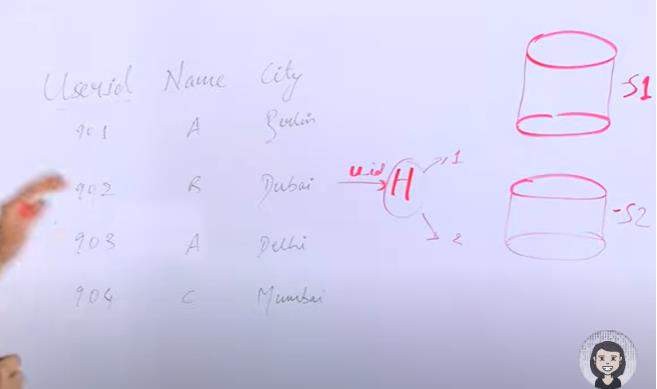
**dynamic sharding** : app will ask to someone else where need to write or read

In **dynamic sharding, a separate locator service tracks the partitions amongst the nodes**. **Drawback** : sharding is complex

**After sharding come back to non sharded data is very complex All the d/b do not provides sharding**

Lecture 22 **Key based Sharding**

User id will go to hash function and will get value on basic of that it will go shard 1 or shard 2



**Now this is algorithm sharding**: since hashing function logic is there in application

**Advantages**

**Data is evenly distributed ( machine** know what type of load is there)

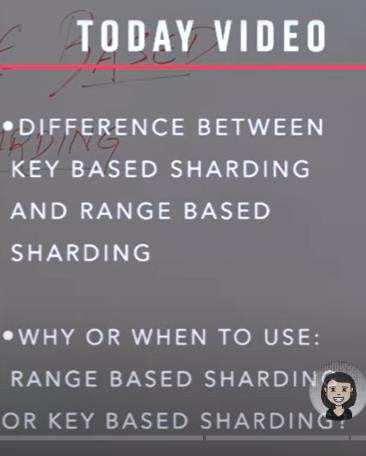
**Disadvantages :**

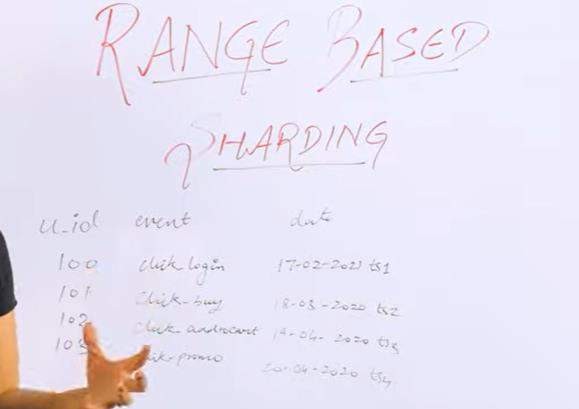
When data increases then hashing function need to get change and move the data from old shard to new shard. This problem occur when we want to reduce the data also. This is more complex.

**How to choose a sharding key**

Key is static in nature , we can choose combination of column as well

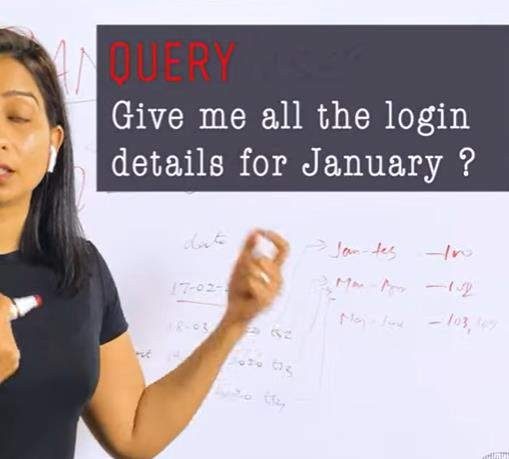
**Lecture 23 Range based sharding | Advantages and disadvantages**





Need to store data for analytics purpose ( in terms of date or month) Note: we can divide this data on the basic of date or month **requirement**

Total 6 month :- 2 month data in one shard



Another example : for e commerce site we can save product in sharding on the basic of price range Advantages :

We have same database shema

There is no hashing function we can add more data Disadvantage :

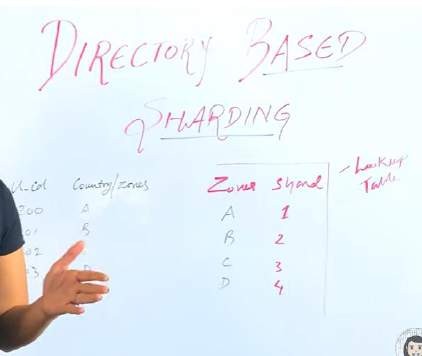
In jan/feb :---- data is more----- those shard is hotspot slow response time

Data is not uniformly distributed Based on use case : it is use more

**Lecture 24 :- Directory based sharding Example of dynamic sharding**



Key and shard is store somewhere else



**benefit**

**We can add more zone and also can remove in that way this is dynamic Disadvantage**

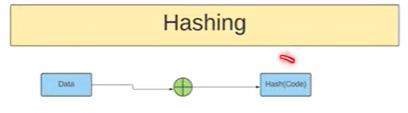
**Increase latency**

**Lookup table down or corrupted**

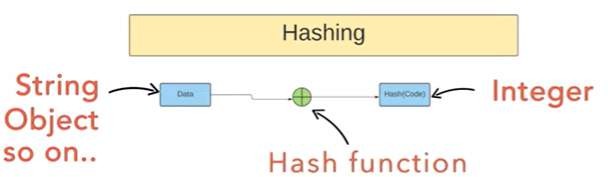
Lecture 25 **A simple introduction to basics of hashing Let** see what is hashing and its use case which solve by consistent hashing

**What is hashing ?**

**U put** data into some type of function or computing which give u result called hashcode.



Hashcode usually a integer value but it can be anything else also



When u put data into hash function it generally generate a integer

**Now how this is useful ?**

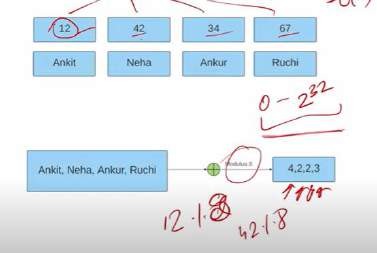
**The** accessing of the element become faster while using hash



Hashing is an important technique in java that helps improve the performance of certain operations. When data is stored in a hash table, it can be accessed quickly and efficiently. Hash tables are typically used to store large amounts of data, such as databases or caches

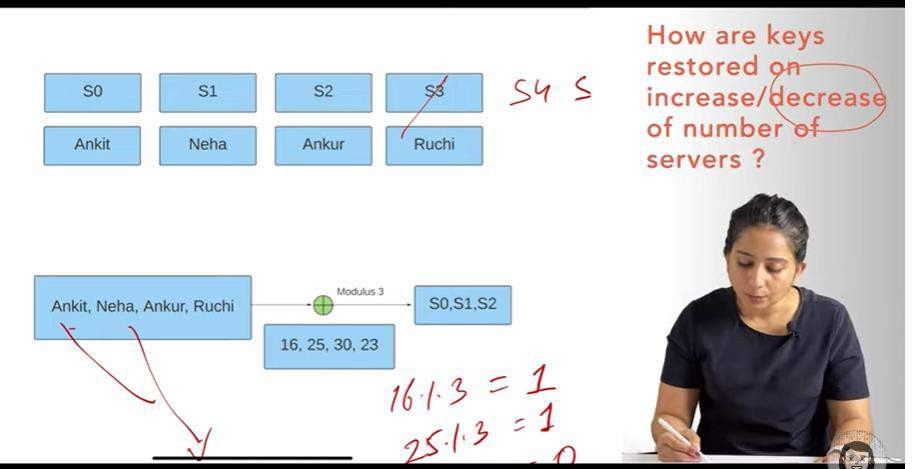
**What is the range of hash value ? 0 to 2 ki power 32**

That’s why we will take mod after calculating hash. Will take mod on all these values and get single digit Now all the value sit at 2,3,4 index





**Two value in same index is called collision**

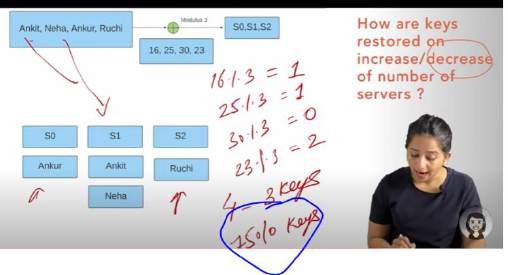


**Example :**

Redis is a key value pair , so instead of array mapping value with server

When one server got removed due to low load then hashing function got changed and the value which was there in s4 then need to reassigned in other server.

**Since we change our hashing function , keys got reassigned when one server got removed with low load ( redis)** 75 percent of key reassign and we don’t want this



So this is problem with normal hashing using modulus approach

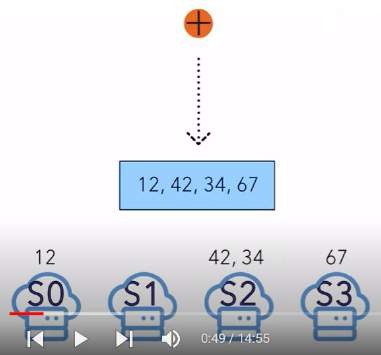
Note : if number of server increasing or decreasing we have to minimum number of movement in hashing

**And how we will do that ? Solution :** consistent hashing

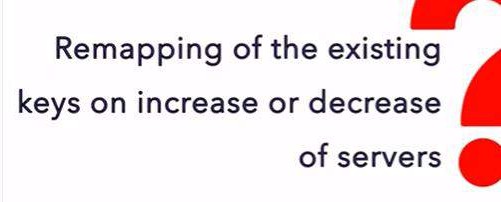
**Good hash function** : collision should be less.

**Lecture 26 Basics of consistent hashing**

We map the number with diff server



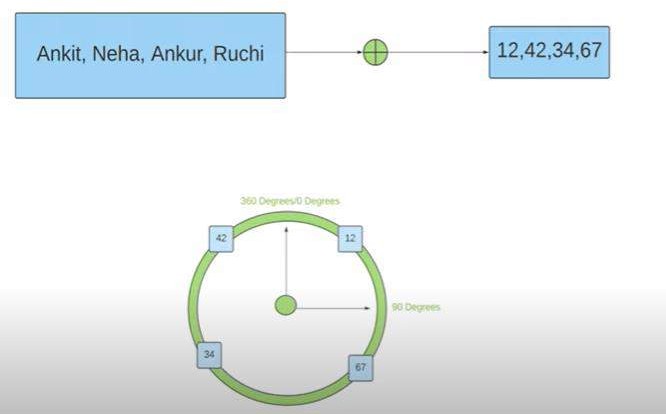
**Problem**



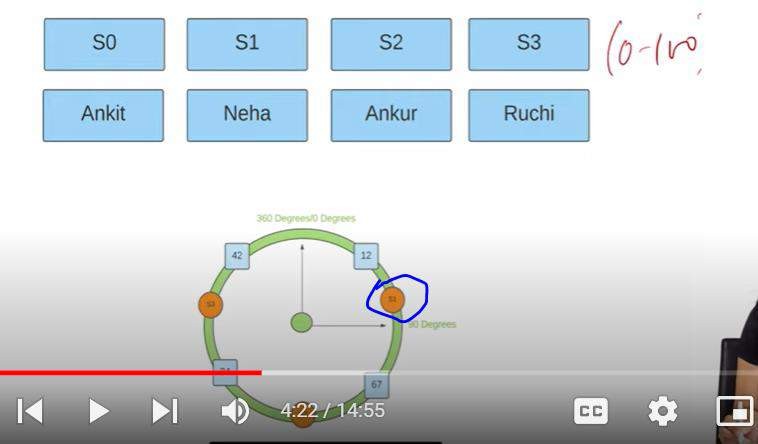
We have to solve the problem reducing number of key that have to be remapped whenever a server added or removed from the cluster



We will put hash generated value on this circle which is 360 degree

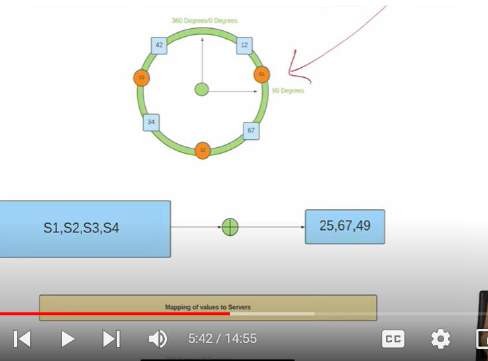


Every value will mapped to certain angle and that value will be represented on the circle



**Note :** for server also hash will generate which will lie on same circle

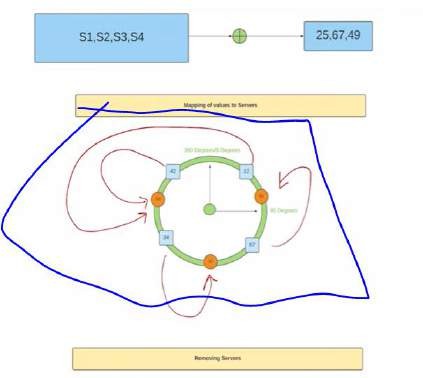
Very imp : the server hash and the data hash share the same output range **(hash value calculated after input)**



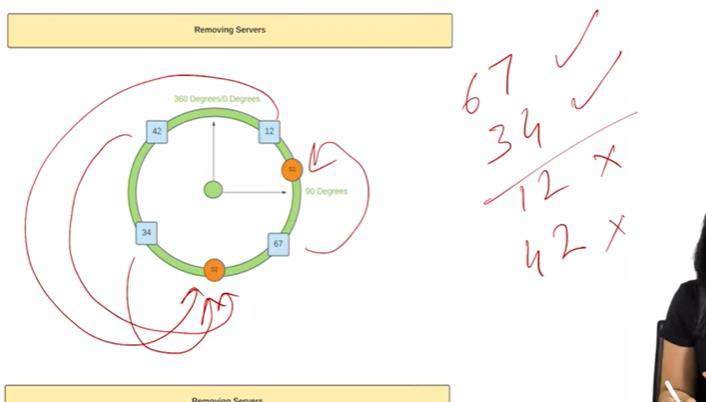
**Mapping of value to server**

How we will know which server has which value ?

We can go clockwise or anticlockwise ( in this way we map value to server)



**Actual problem :** when we remove server from this circle



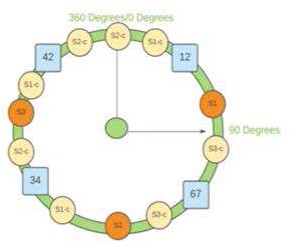
Here the movement of key become less, only less number of key remapped

**Now what is the problem with this approach ?**

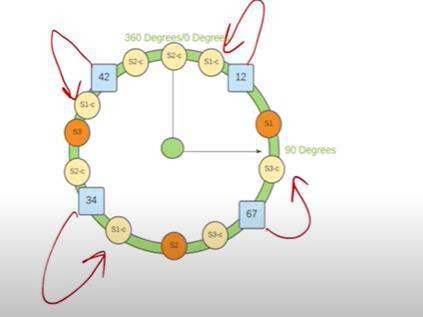
**Here :** s2 got overloaded then distribution of key is uneven

**How we will solve this problem** ?

We solve the problem of less number of key movement but now we encounter into another problem that is key not uniformly distributed.

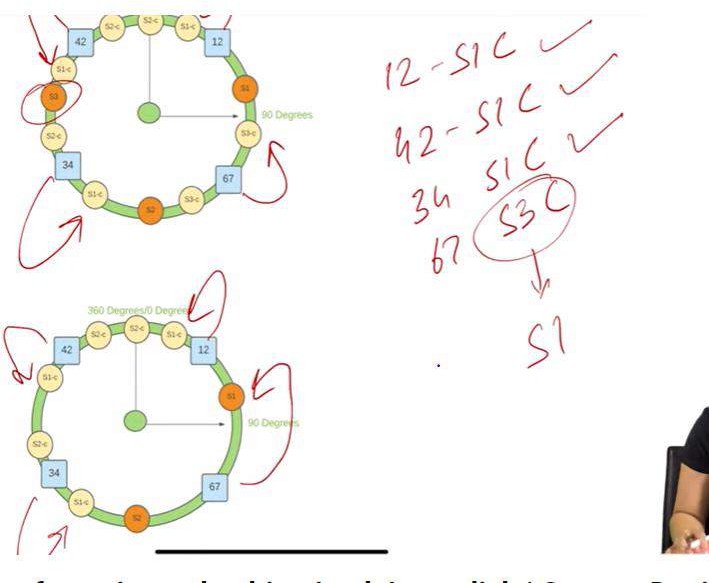


We will going to create replica of server



**Now get rid of s3**

**When we use replica we need to move 25 percentage**



**What is Consistent Hashing?** Consistent hashing is a technique used in computer systems to distribute keys (e.g., cache keys) uniformly across a cluster of nodes (e.g., cache servers).

Lecture 27 **Foundation of System Design Interview**



**First step :** to find functional and non functional requirement

**Functional requirement** think wrt user



**Now coming wrt system**



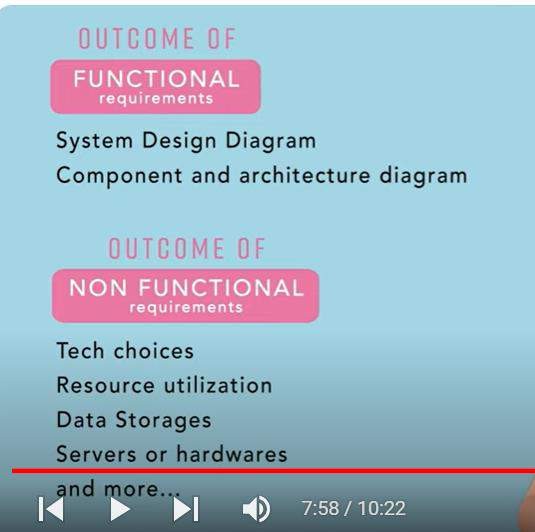
**Step 2 :**

**Once we understand functional requirement then break it into component try to design component( api**

**,messaging, events, worker)**







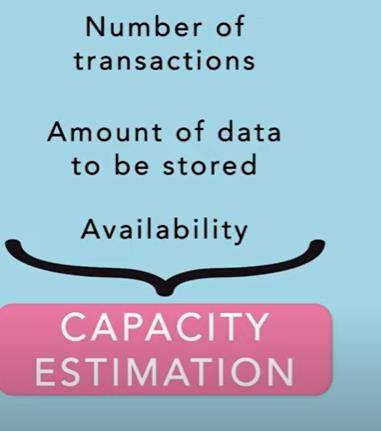
**What is a Functional Requirement?**

In software engineering, a **functional requirement** defines a system or its component. It describes the functions a software must perform

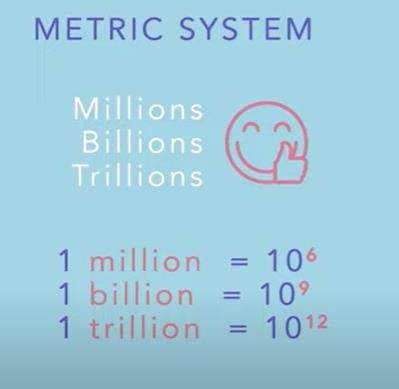
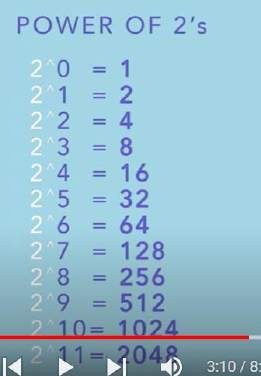
Types of **Non-functional requirements** are Scalability Capacity, Availability, Reliability, Recoverability, Data Integrity, etc

**Non functional : defines performance of the system**

Lecture 28 : **How to solve capacity estimation problems faster**



**Capacity estimation** problem come from non functional requirement











**Five mistake :--**

1. **Not understanding the requirement properly** ( ask functional and non functional requirement)
2. **Not talking about capacity estimation ( load of the system is going to have)**

Most of the system design done for **scalablity and interviewer** want to know what u know about scaling system

**We need to estimate, traffic ,load user 3 – trade off**

What is the meaning of trade-off?

A trade-off is a kind of compromise that involves giving up something in return for getting something else. 4-**Not finding fault with the solution**

1. Not understanding the requirements properly
2. Not talking about capacity estimation
3. Not discussing the trade-offs
4. Not finding faults with the solutions
5. Better to say "I don't know"