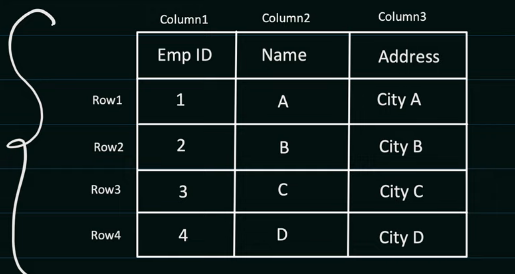
System Design Concepts

# Database Indexing

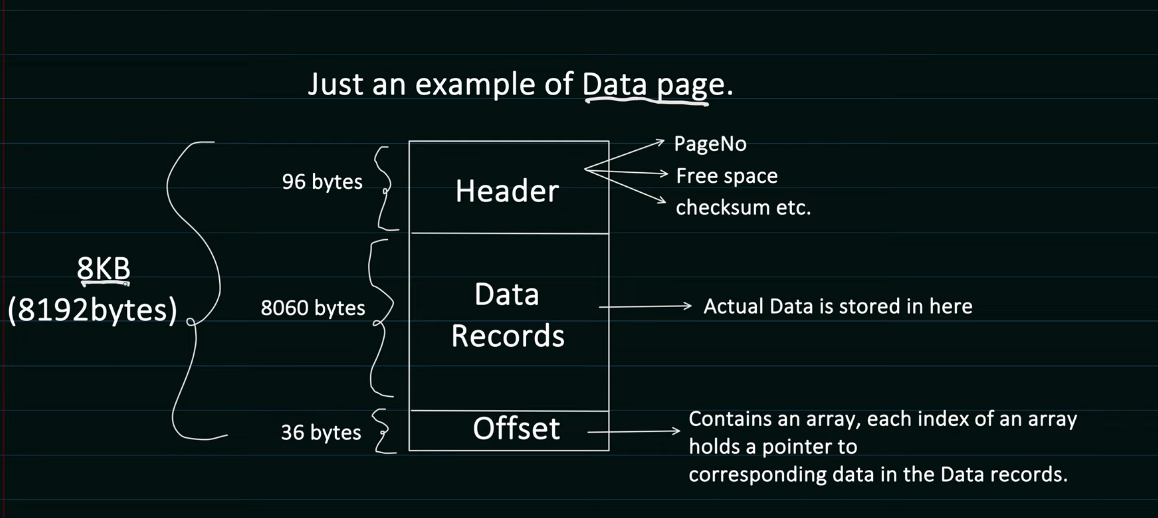
## How table data store into the table?



This is just a logical representation of the table. Actually table is not stored in this form

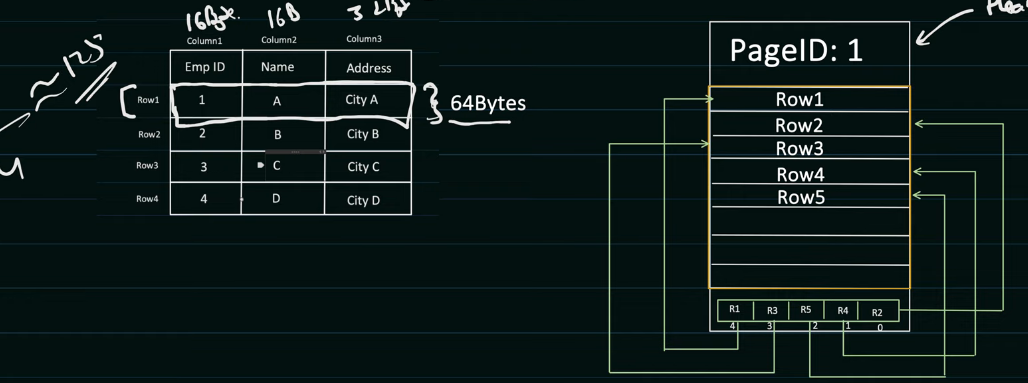
DBMS Creates “data pages” (Generally its 8kb but depends upon DB to DB)

Each data page can can store multiple table rows In it.



Datapages is of total 8kb and 8060bytes is data records

Let say one row is of size 64Bytes then 8060/64 = 125 So here we can say that 1 data page can hold upto 125 db rows

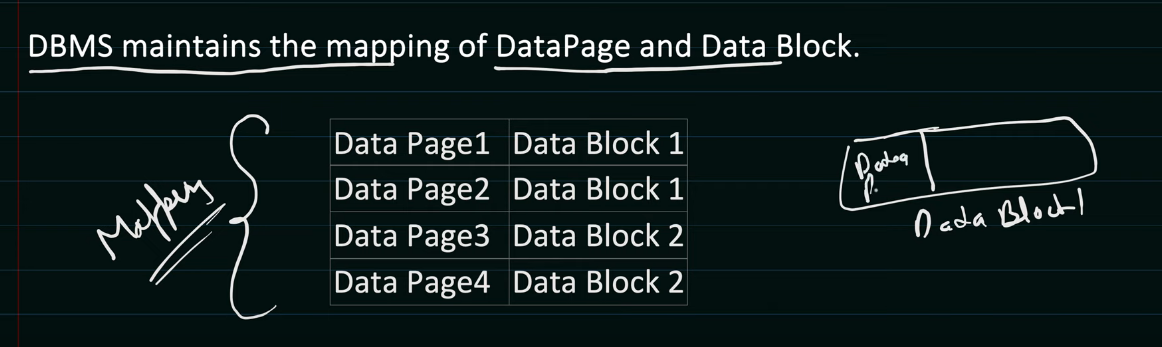


Here last line is representing the OFFSET array it’s and pointer to the actual data present in the data page it’s not in ordered but more of

These Data pages unlimited get stored in the Data Block in Physical memory like disk.

## What is Data Block ?

* Data Block is the minimum amount of data which can be read/write by an I/O operationsI/O operations
* Minimum amount of data that we can fetch from the memory
* It is managed by Disk or Secondary Memory
* Data block size can be range from 4Kb to 32kB



Here we have the mapping of Data Page & Data Block and mapping is done once we read the data from the offset then we will fetch the Data page from the data block Or from the Disk according to the mapping.

* Data Blocks can be scattered on the DISK there is no Sequential data stored in the DISK.

## What type of Indexing present in RDBMS ?

Indexing of 2types

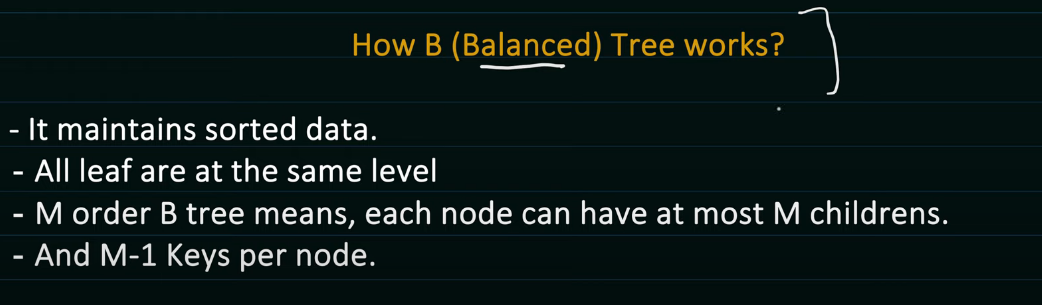
* Cluster Indexing
* Non-Cluster Indexing

Indexing used to increase the performance of the data query. So that data can be fetch faster.

Without Indexing DBMS has to iterate the data over all the rows to search the data

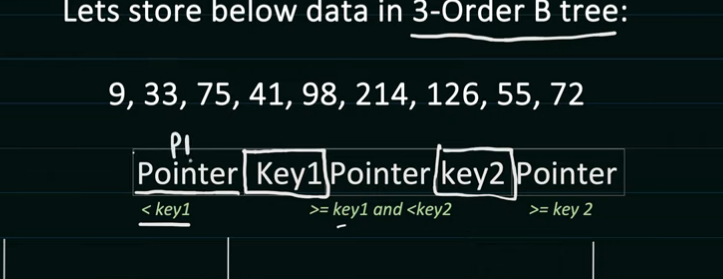
O(n) time will be required to fetch the data from the DBMS

**B+ tree Data structure Provides better time complexity for insertion, searching & Deletion.**

****

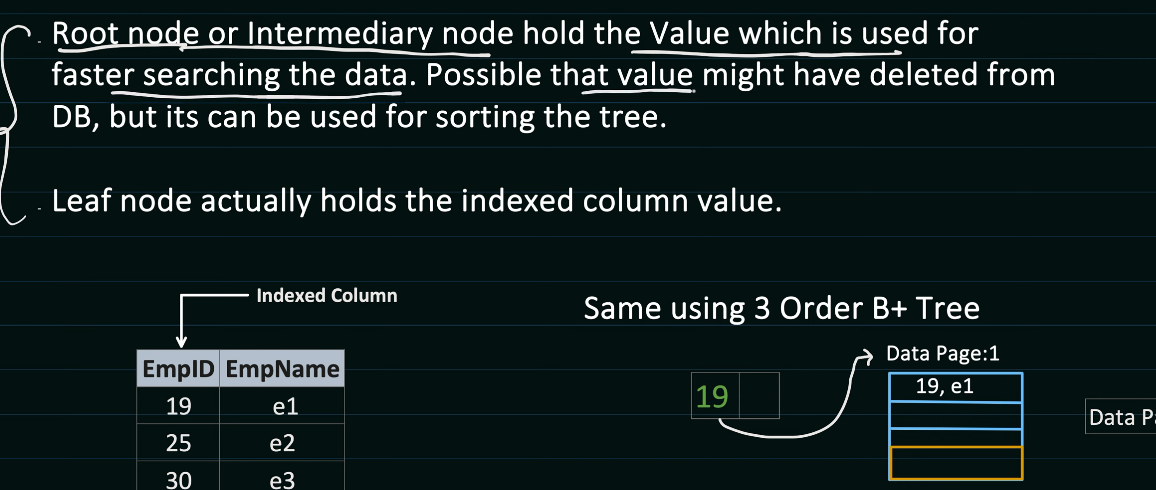
**M order B tree means each node can have atmost m child in the tree**

**And m – 1 key per node**

****

B+ tree are same as B tree but additional feature like child nodes are also linked to each other.

DBMS Uses the B+ tree to manage it’s data pages and rows with in the pages



We Have Data pages where our data is stored and indexes maintain in the form of B+ Tree which will point the data pages where we store the data into the page and same hierarchy will follow

And datapages store into the DATA BLOCKS each page have its own capacity to store the data into the data pages

Ex : Let say we have 1 row of 8kb and data page if 64KB then we can store the 8 records into the

# Indexing Types:

1. Cluster Indexing : as soon as we apply the primary key to the schema then automatically that can behave as sorted order and store into the sorted form in data pages.
2. Non Cluster Indexing



CAP Theorem:

Consistency Availability Partition Tolerance

These three things we cannot guarantee at the same time we can make only 2 happens rest we have to compromise with other

In this Partition Tolerance is the thing that we have to maintain we can compromise with Availability or Consistency.

We have three scenario

1. CA
2. CP
3. AP
4. CAP

All three of them is not guaranteed if partition between 2 node get breaks then we will transfer the load to one node then availability is there but we have to compromise with consistency as one node have updated data but another one don’t have it

# Consistence Hashing

Hashing with normal values

If we have to insert some data value into the hashtable then we will calculate the hash for that

Value and insert into the HashTable.

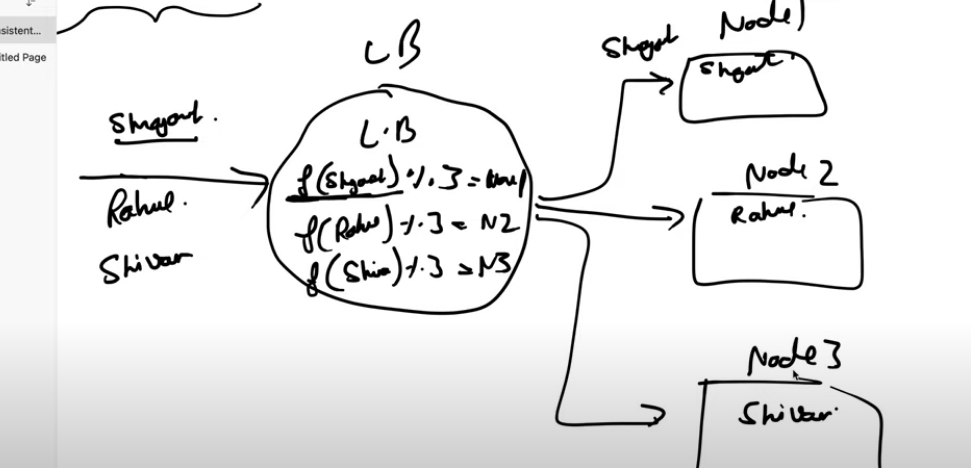
What if we have a fixed size of table in hash

Let say a example of LOAD BALANCER



Here we have a LB and a key we are getting the value from the client and on the basis of modulo we will divide the value **f(value) % 3 == server id**

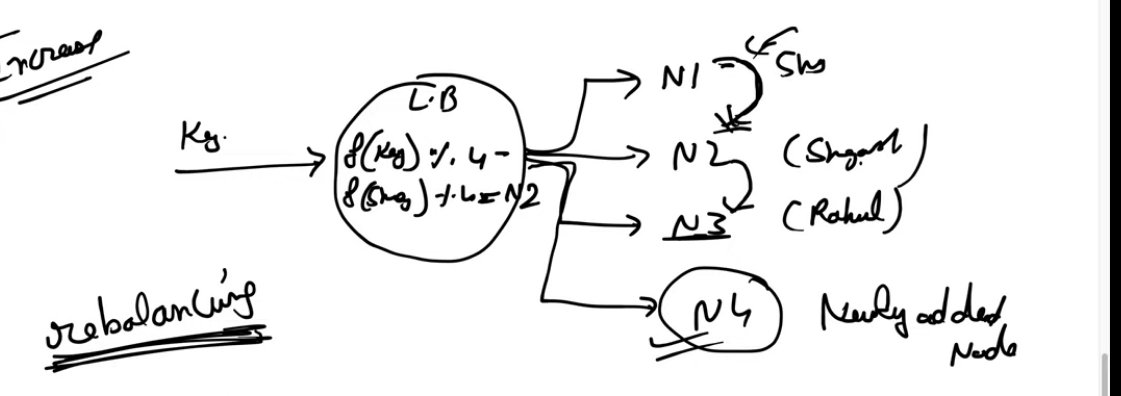
But what if we are getting same hash value again & Again then it will increase the load of the one server



These are the server we have what if

1. Server increase or we have the new node then our function will be f() % 4 if we add one more server Now if try to fetch the value the it will not guaranteed that F(Shreyansh) will present in Node1 only it might give us the another index of the server

Solution: We need to perform the rebalancing



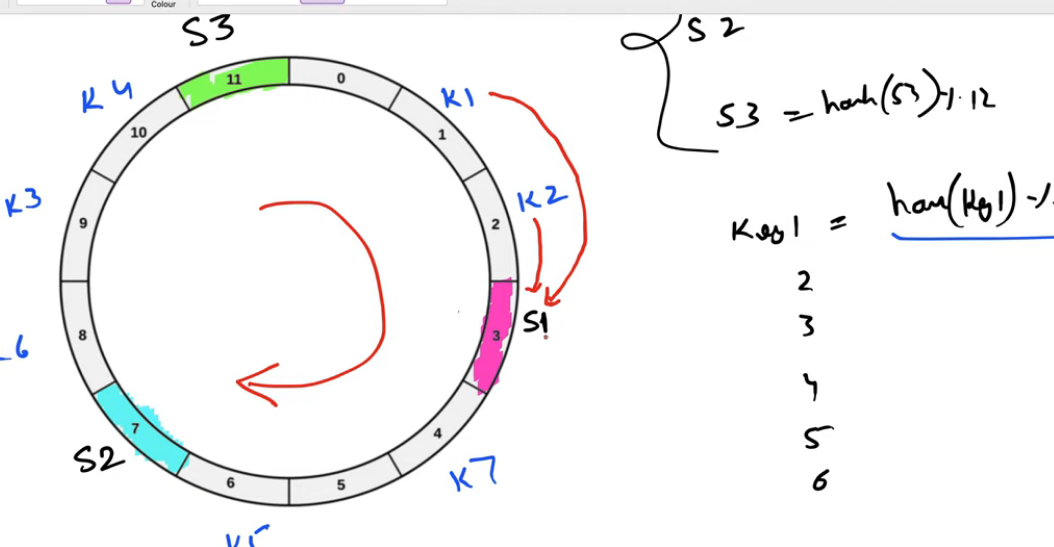
In db we have millions of entry and what if we have to rebalance for the millions of entry at this time

Solution for this is **Consistence Hashing**

Consistence Hashing says that If we add the node or remove the node then the total rebalancing that happens internally is **(1/n% of total number of keys).**

**Work flow of consistence Hashing:**

1. **Virtual Ring** 
   1. We have 3 server s1,s2,s3

****

**Here we have the keys that we need to store onto the node those are present on S1,S2,S2**

**So what we will do here we will move into the clock wise direction**

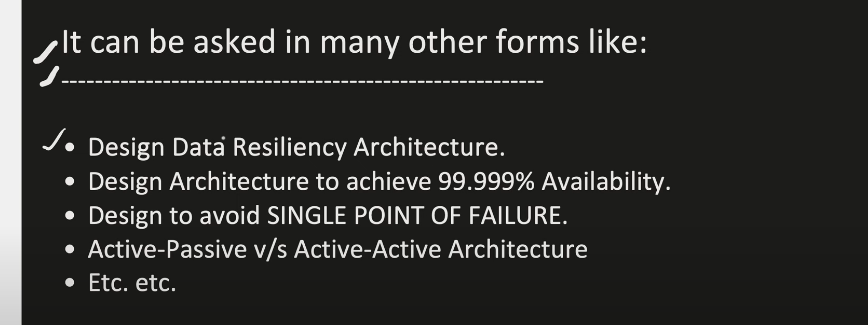
**So for K1 we will move clock wise then we will get the server 1 value**

**If we move clock wise for k2 then it will go to the S1**

**How it will handle if a new server is add on like if we add one more server s4 then what we will do**

**Id we need to rebalance only one key and rest other is point to the same if we move to clock wise direction.**

High Availability



Client

Load balancer

It’s a single point of failure and it’s doesn’t guarantee the 99.999% architecture

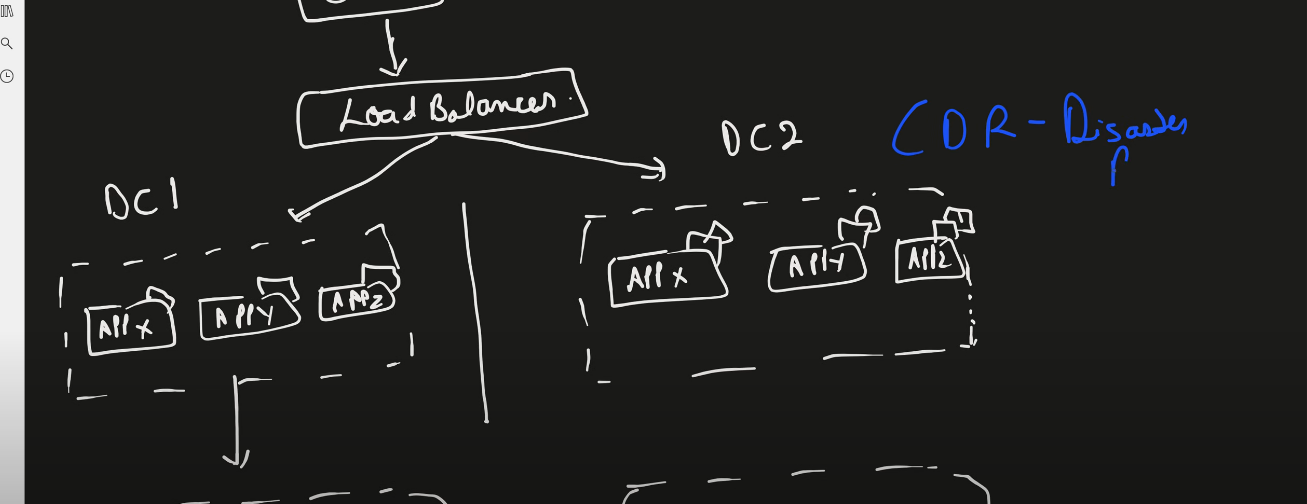
Here we have multi Nodes

1. Active – Passive
2. Active – Active

## Active Passive Architecture:

We have 2 data center

DC1 (Mumbai) DC2(Pune)



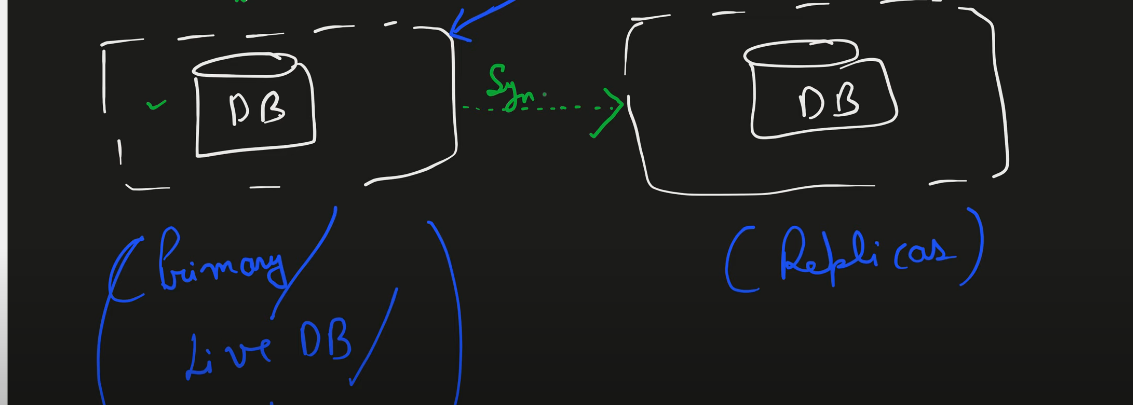
Here we have active passive architecture

One is the DC1 is the active one or primary one we have the load balancer which moves the request to DC1

If load balancer shifted the request to DR servers then it will transfer the request to primary DB

Below we have 2 db one is primary and other one is replica set

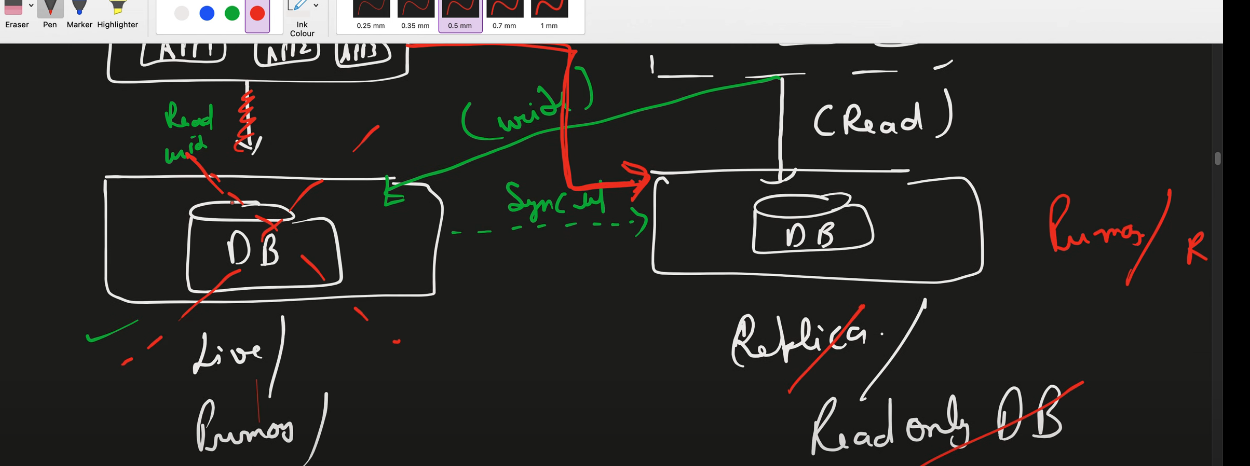
And both of them are synch up in nature to process the latest data into the both db



## 

If read request comes in then we can read it from the replica set and if write request comes in then it goes to the primary db

As Oracle,Mysql,Postgre SQL is not multi master



If the db is down then we can say that traffic will route to the replica set

At the time of break link then making replica connection is a data loss for the time being.

## ACTIVE – ACTIVE Architecture:

Same design as above one two data center with micro services and all of them is live

And here data base is in sync up in nature and both of them is live and active!!

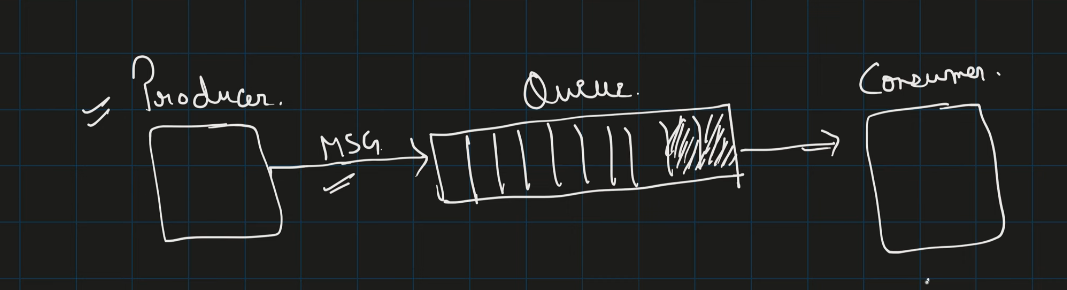
If any of the data center get request then both the db is in synch up form

ACTIVE ACTIVE can handle more traffic as both the db of data center are Read/Write.

We can go Active active HA where we need the High Writing Activities and should guarantee the Availability of the server and consistency should be maintained

System Design Distributed Messaging Queue.

# What is Messaging Queue?

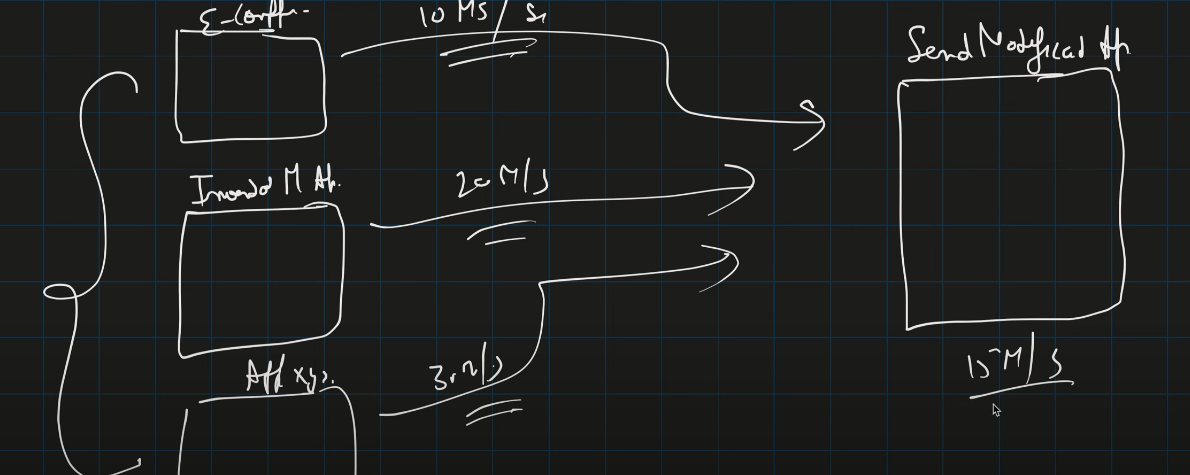


We have a producer consumer architecture

Use case:

1. Asynchronous in nature.

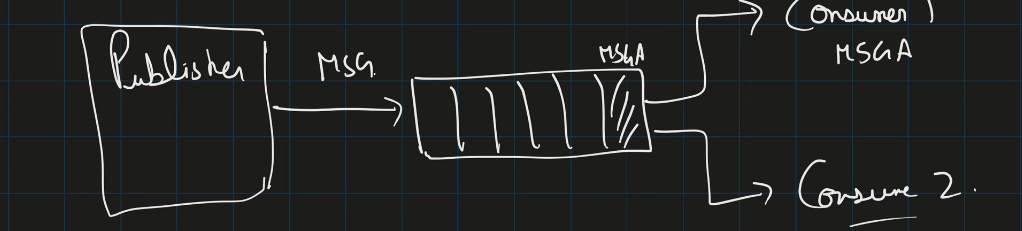
If we are doing some work then in between we need to send the notification to the user we will produce message send notification to the user and Asynchronous it will send it we don’t have to wait for the notification.



Here in this diagram we can see that different services produce messages at different pace but client side or consumer side it processing at 15 message/second.

## What is p2p & Pub sub ??

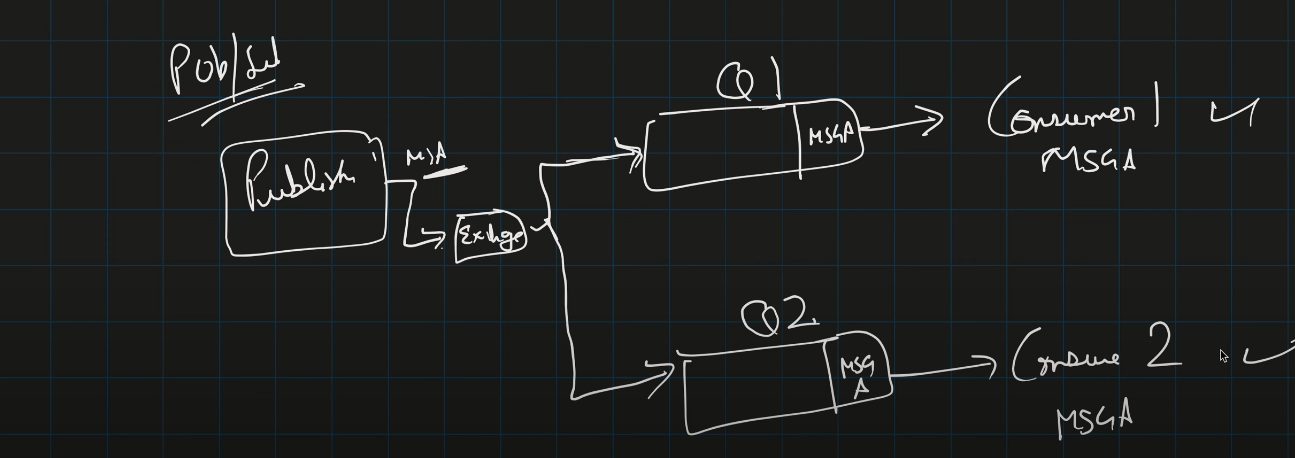
Publisher is a message in a queue and consumer consume that msg



One message is consume by one consumer in p2p architecture

P2p says that if we have an msg in queue then only one Consumer is allowed to do this.

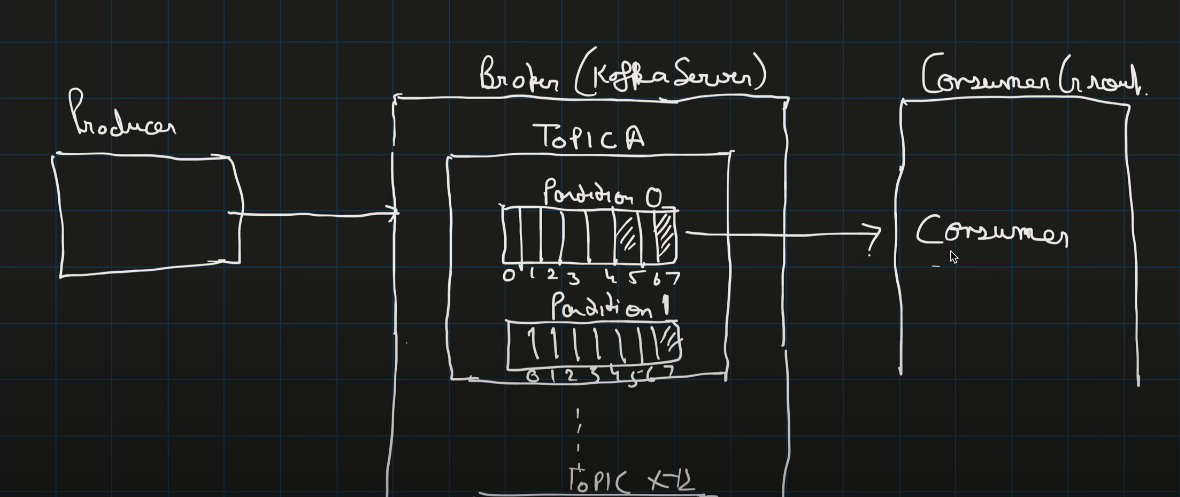
In pub Sub :

In pubsub pattern producer will produce the message for the queue and it will broadcast to all the queue in the system and any number of consumer can consume it 

# How Messaging Queue Works? Kafka :

1. Producer
2. Consumer
3. Consumer Group
4. Topics
5. Partition
6. Offset
7. Broker(Nothing but kafka Server)
8. Cluster
9. Zookeeper

## Architecture Of Kafka:



As you can see that We have Broker which is nothing but a Kafka server In that we have n number of topics like Topic A to Topic N

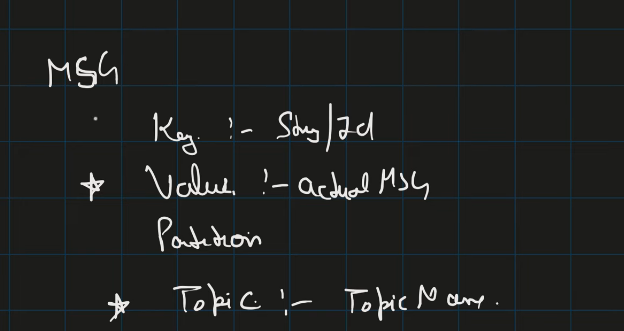
And in topics We have number of partition from partition 0 to partition n having numbers like 0 to 9

And we have consumer Group in that n number of consumers can be there.

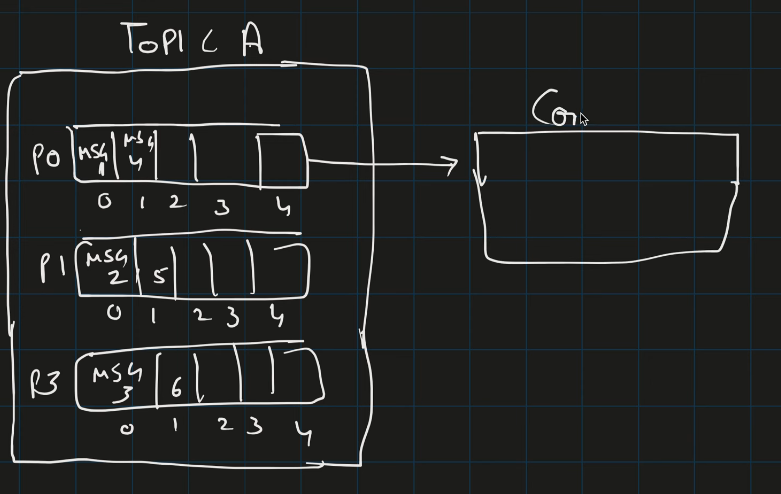
**All the broker talk with zookeeper**

Message Strucure;

In msg structure it’s not important to send key or partition and both



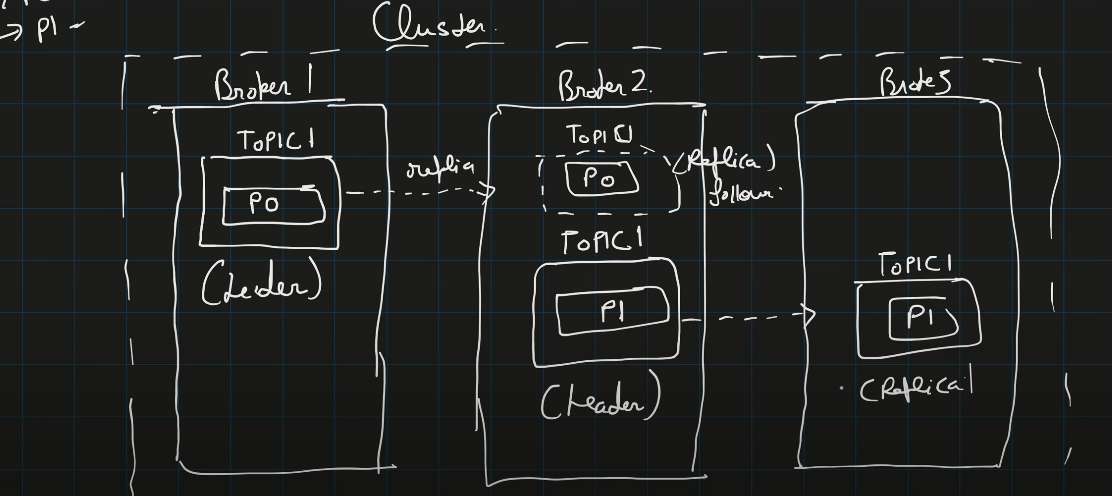
If key is empty then it will check the partition and if it is define the it will go to the define the partition

If none of them is not present key & Partition then in round robin fashion it will store the value of the msg. 

Commited Offset tell us that How much data is read by the consumer group consumer 1 from the partition0 so we will set the offset value as 3 now next time if we want to read the p0 value then consumer will start reading from offset 4.

## When Queue Size is at limit then what happen:

We will have different broker

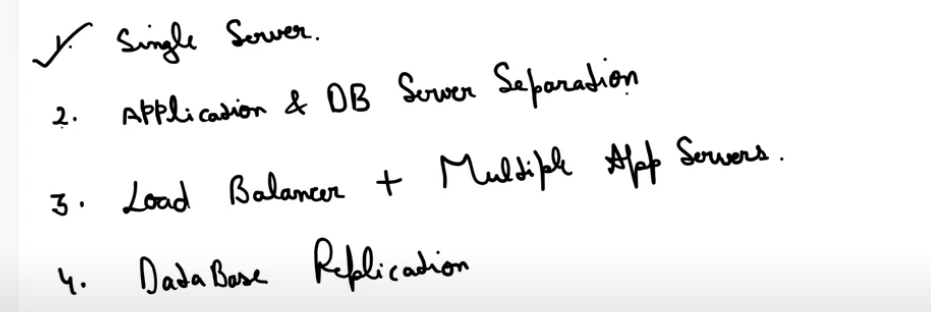


As we can see that we have 3 different broker or Kafka server we can say that and each of them having some topics in it

And we have created the replica of the topic into another broker so the main topic of that broker which is not replica is called **Leader.**

Read / Write happens in Leader only

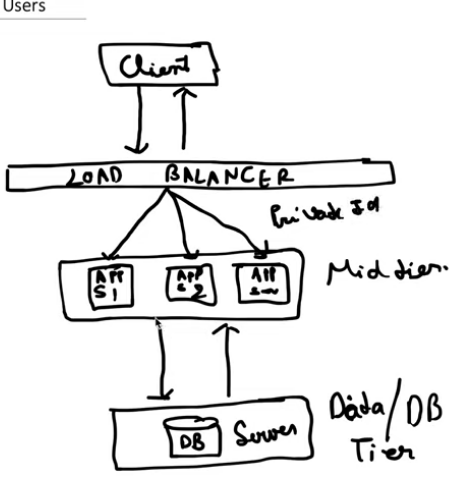
“Zero to Million Users scale in Detailed”



Some Steps we need to follow to scale the project from scratch

CLIENT => Web Server where application is there => DB Server / DB Tier

Now we need to achieve the LB to distribute the load in between the different machines



Our client will never send the request directly to the Web services As our backend services are host on the private IP and these IP are exposed to load balancer and load balancer IP is white list to the API Gate way.

So in general our client send the request to the api gateway and then there we will distributed the load to the different machines.

## Now we will introduce the database replication

Master – Slave Architecture for DB’s

All the write goes to master and read will be done from slave and both of them is in th synch nature.

1. If master db fails in any case then slave db will promote to master and handle the concurrency.

## Use of Cache:

Application server talks with db then it’s a n/w call then read and write operation are expensive

Instead of calling the db then it will call the Cache

Two cases:

1. Cache Hit
2. Cache Miss

And it’s increase the performance of the cache

In cache there is always TTL (Time to live)

## CDN(Content Delivery Network)

CDN does caching but all those who does caching are not CDN

* How CDN Works
  + We have Different users country wide
  + India,USA, Japan
  + The data access through our country will be costly as it takes more time 2ms, 3ms, 4ms

Now if we get the request then from usa then it will go to nearest CDN

Now here the everything got increased performance and security and load of networks call and cost cutting we don’t need DB servers that much

Distributed Cache & Stratagies:

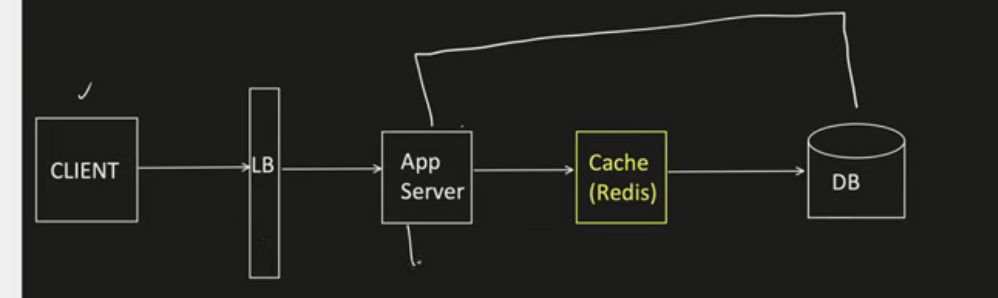
# Cache:

## What is caching:

It’s a technique to store the frequently used data to store the data into cache rather than the Secondary Memory to remove the slow access memory

* There Are Different type of caching :
* Client Side Caching
* CDN(Used to store the static Data)
* Load Balancer
* Server Side application(use of Redis) etc.

## Design:



It’s in between the application and the Database.

## Distributed Caching:

Let say we 3 App Server and each of them is accessing the same cache server so the problem we can face:

* Scalability
* Single Point of Failure

Solution of this is distributed caching

We will create on Cache Pool having more than 1 cache server and we will have one Cache client to access the server from the cache pool below is the architecture define.



Problem : How the client will select which cache server we need to select from Cache pool

Solution: Consistent Hashing We have study about this One ring will be form around the server and divide it into the server we will travel around the ring for each client when we get the server we will allocate it to that same

What we will do we will calculate the hash value let say if we want to put the value into the cache

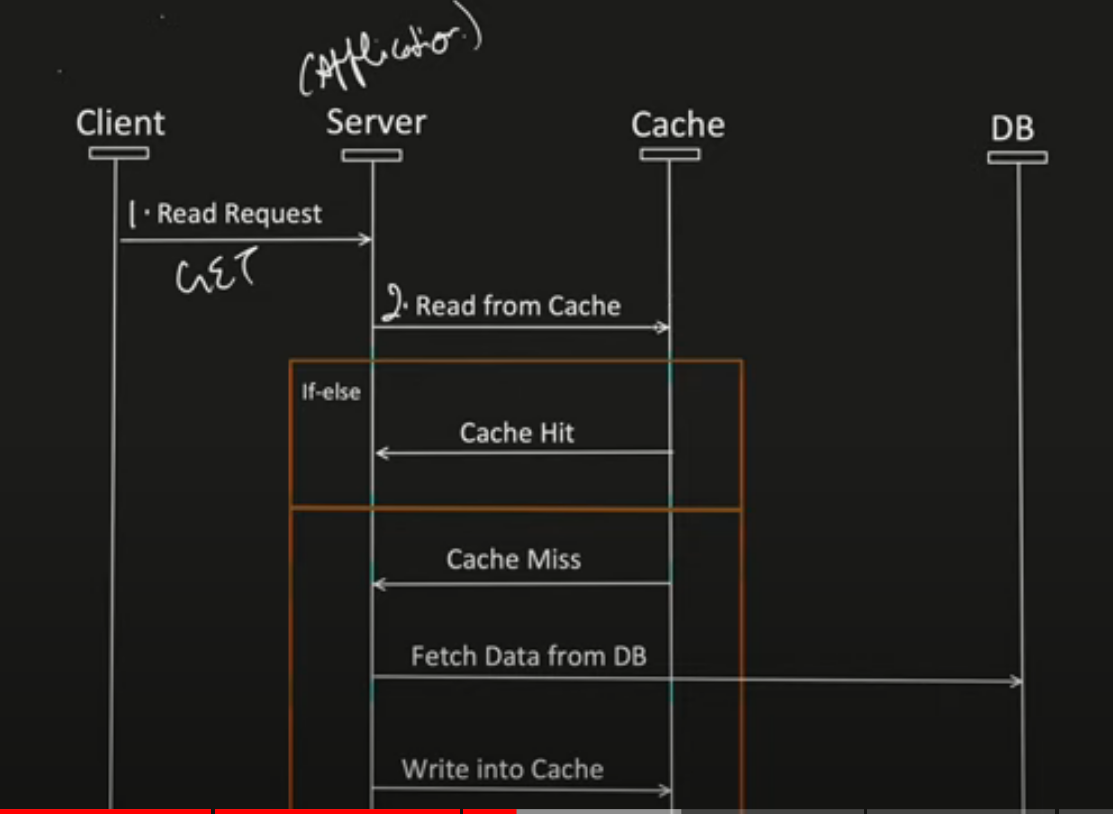
And we have 6 cache server in cache pool

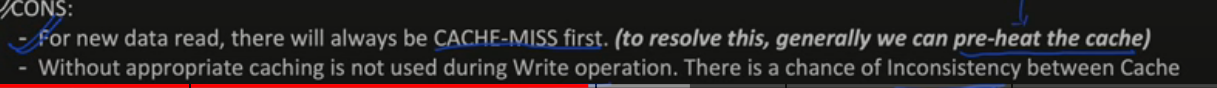
Then => **Function(value) % 6 == key after getting the key we will move in the clock wise direction and the first encounter server will be the one where we will save our value;**

## Cache Strategies:

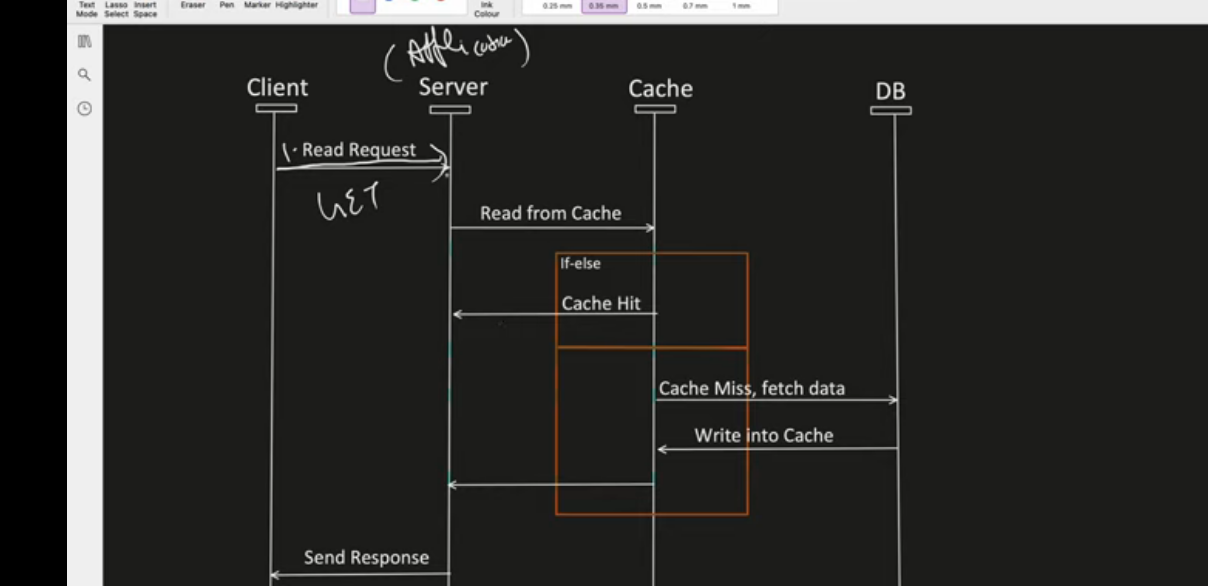
1. Cache Aside

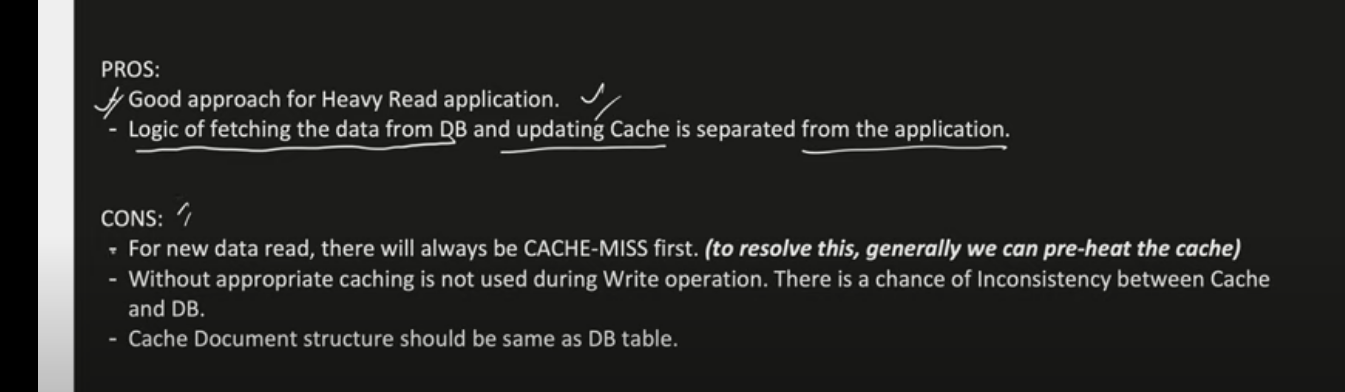
* First if we get the request on the sever it will check the cache if we found the data the cache hit otherwise cache miss.
* Application fetch the data from the db and put it into the cache for the next time.



* Pros:
  + Good approach for heavy read
  + Even cache is down request will not fail as it will fetch the data from the DB.
* Cons:
  + If write operation perform directly on the DB then we will perform the get operation from the cache then the data can be inconsistent on both the places/
  + 

1. Read Through Cache:
   * Application first check the cache
   * If data is found in the cache it will return the data to the client
   * If data Is not found in the cache the first it will fetch the data from the db and sotred it into the Cache then return the data to the client.

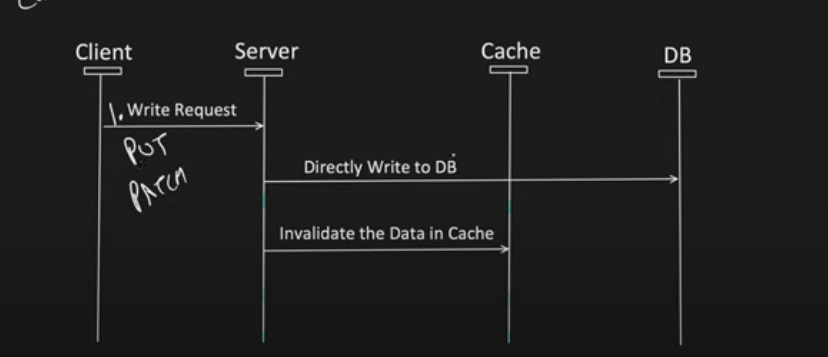




1. Write Around Cache:

Directly write data into the DB

It do not update the cache;



First we will directly into the db PUT request come and change the data value into the db

One dirtly flag to be set when we write the data and once get call will be there then it will check the flag is dirty or not

PROS:

* + Good For heavy read application
  + Resolved inconsistency between cache and DB

CONS

* + For new data read there will be cache- miss because we are directly writing into the db(to resolve this generally we can pre hit the cache).
  + If DB is down, Write operation will fail.

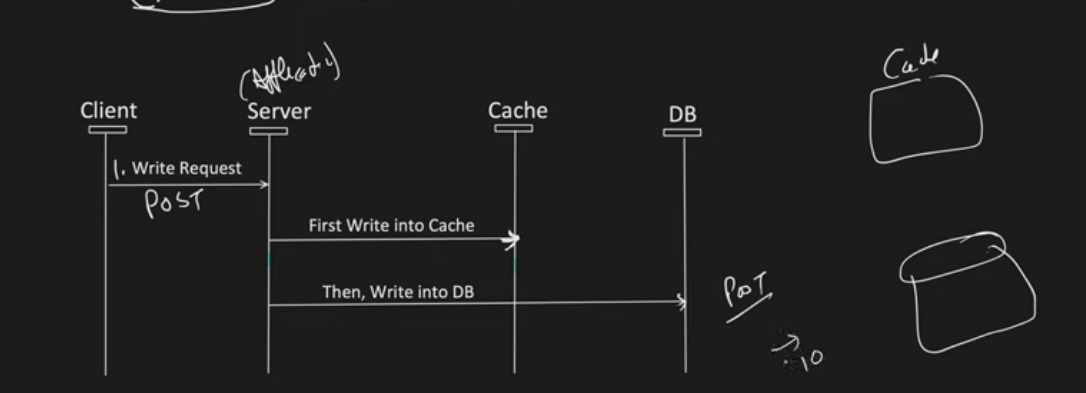
1. Write through Cache:

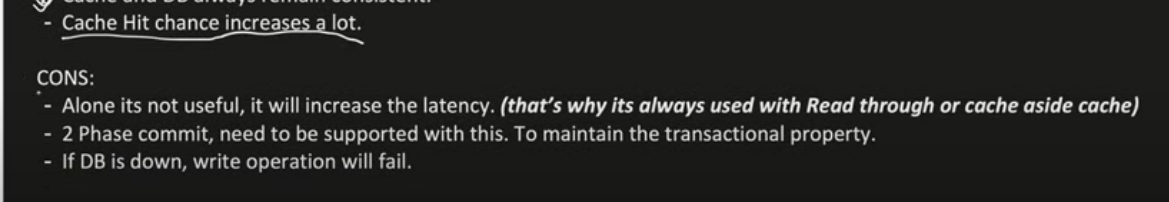
This will pass through the data

* First write into the cache and then synchronize with the db.
* If any of the insertion is fail then we have to fail the whole transactions.

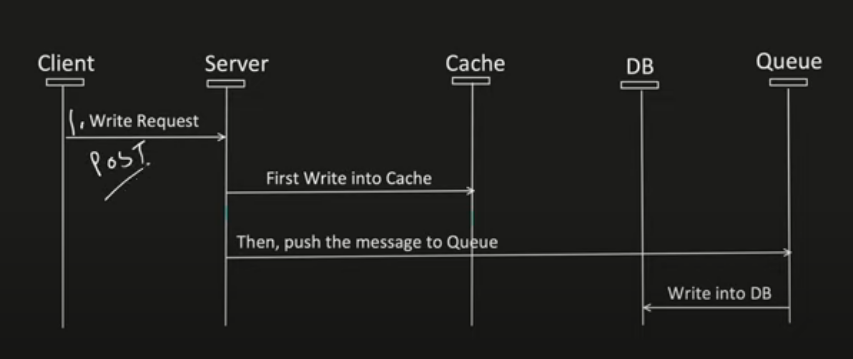
PRO:

* Cache & DB always remain consistent.





1. Write Back or (Behind) cache Strategy:



Here what we will do we will write the data into the cache not into the db and and apparantely generates one push message or asynchronous message to the Queue that will directly write into the db and the latency of writing and waiting for writing is save as of now.

PROS:

* We are not dependent on to the DB any more
* If DB Is down we will not have to roll back the writes as we have TAT(turn around time) of cache 48 hours that will store that data into the cache only once the DB is up then we will write the data into it and clear the queue.
* Cache Hit chance a lot of increase

CONS:

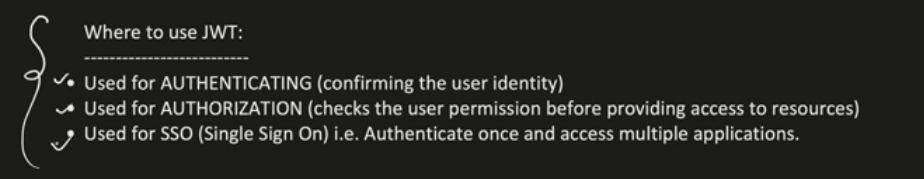


# JWT Explained:

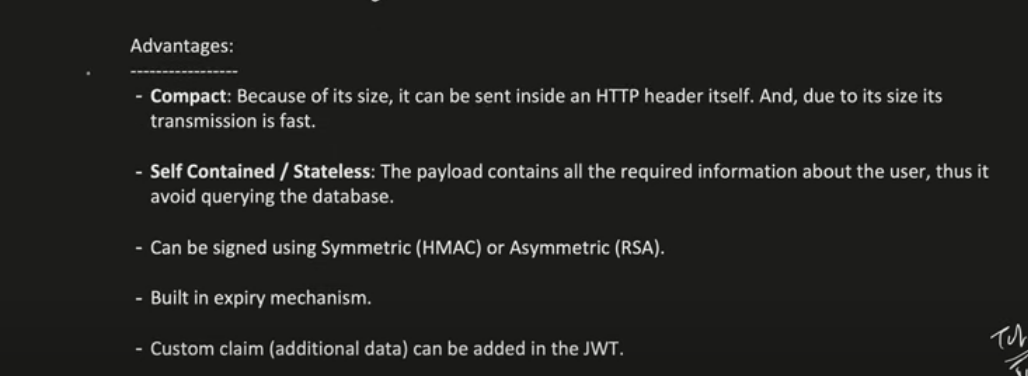
It’s provides a secure way of transforming information between parties as a JSON object.

This info can be verified because it’s digitally signed using RSA(Public / Private KEY) etc.

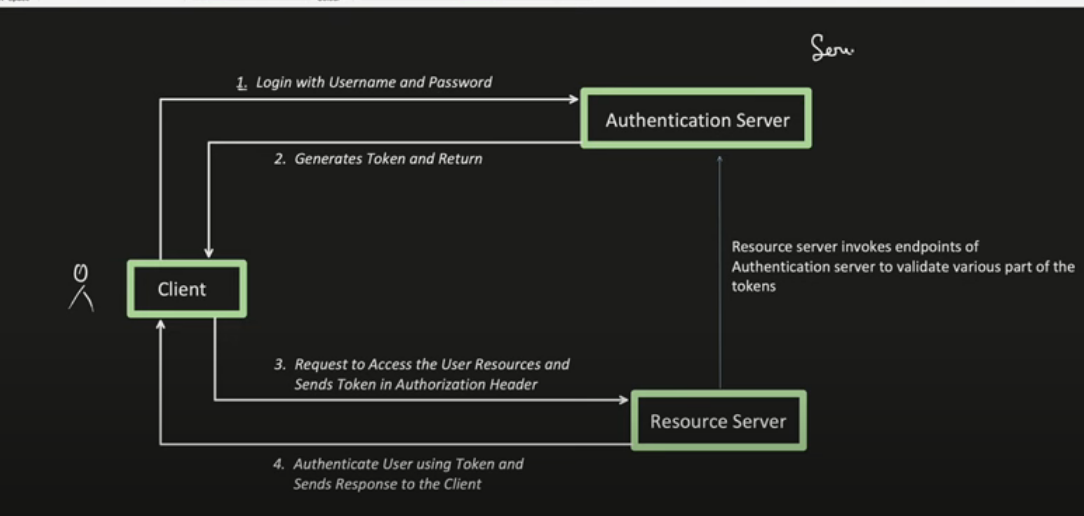
HMACSHA256() signature



It will check the permission or role based authorization weather I am allowed to access that data/api or not



## JWT Flow Diagram:

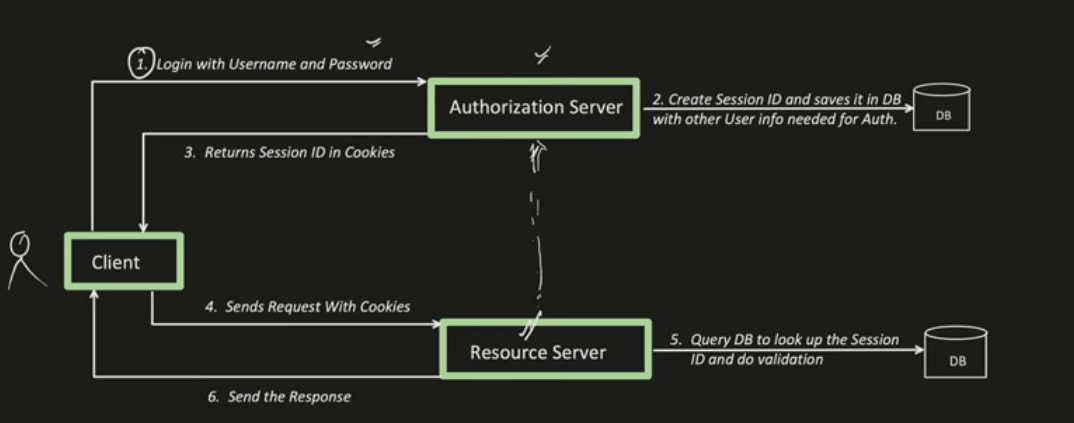


We have one application and we have one authentication service CLIENT tries to login on our server using login id and auth server will give the JWT TOKEN

Now if Client tries to access any api then it has to pass the token to access the api

## What was popular before JWT and what are the problems in that ?

**Session Id or JsessionID:**

****

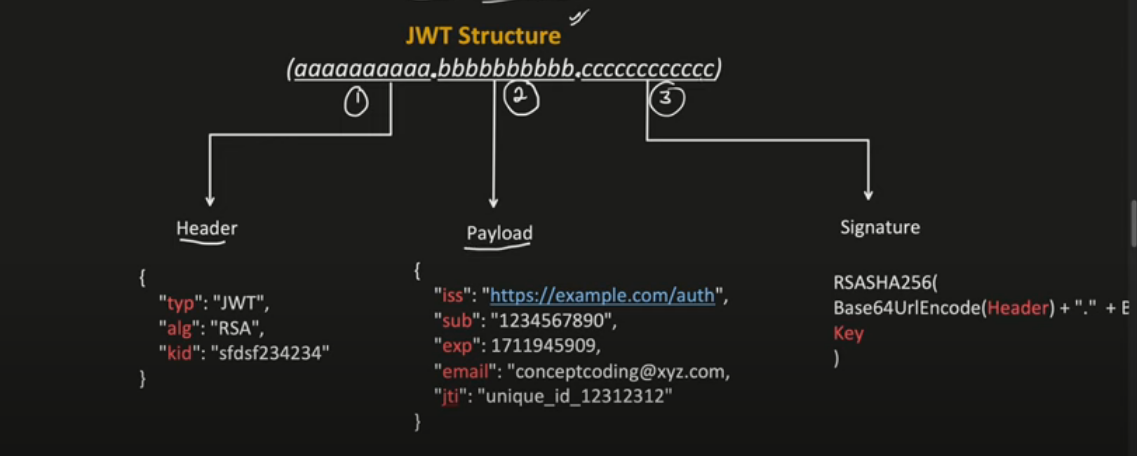
Flow is same Client tries to login and it will generate the session and store into the “DB” with all the information and return unique Id “SessionId” returns back to the client.

Now if we have to hit any api then first we have to hit the DB to look up the session and validate all the information

Major Issue:

1. Each time we are hitting the db which increase the latency of the network call
2. It’s stateful because we are maintaining the state of the user and storing the data into the DB
3. It just a unique number for which DB have to perform query to search the value for the same.

## JWT Structure:



1st is header 2nd is Payload and 3rd is Signature

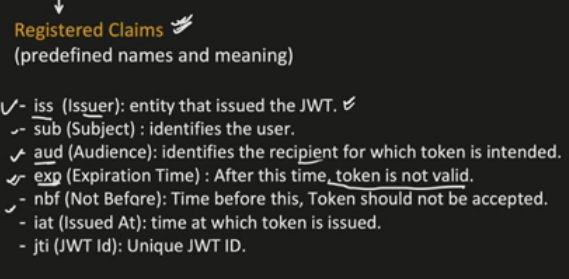
Header:

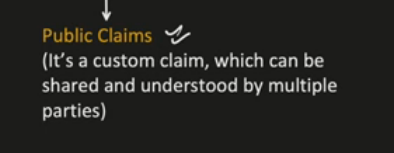
* Contains metadata information of the token
* Type of the token generally JWT always we added here
* Alg : Signing algorithm use like RSA or Hmac etc.

Payload:

Claims

Registered claims are reserved for the JWT token



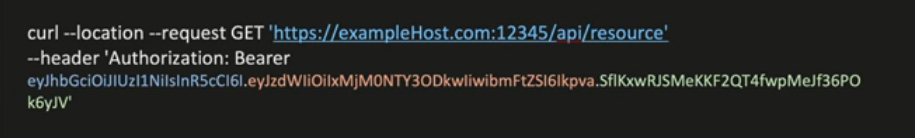


It’s a custom claim which can be seen by anyone

After this we have the private Claims which are indented for internal use only and not standardized

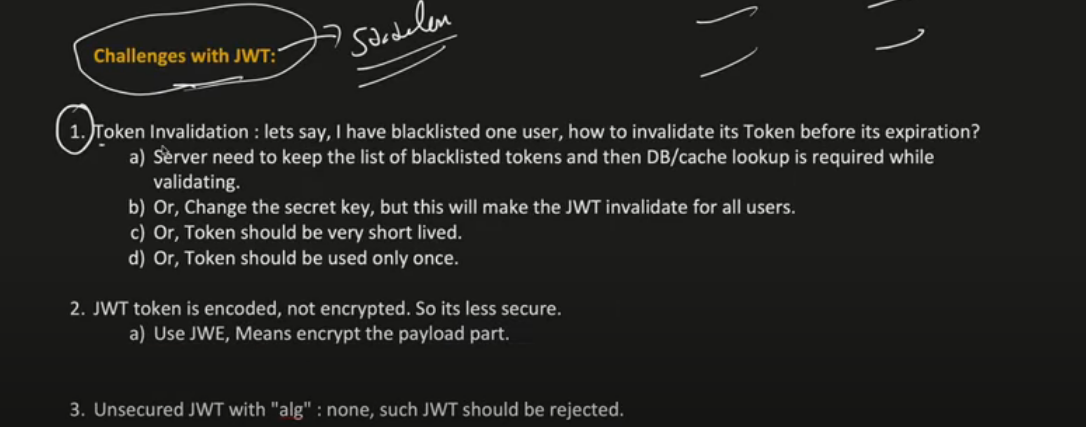
1. Signature:
   1. Encode Jwt Header and payload separately using base64 encoding
   2. Concatenate encoded string of header and payload string using “.”;
   3. User(RSA) or (HMAC)to create a digital signature

Sample resource request of passing header be like:



Base64(Header).Base64(payload).Base64(Signature);

# Challenges with JWT:



This is it for the JWT Token From security Concern to the use of JWT Effectively.

# Oauth2.0 Explained with Architecture:

Oauth – stands for open Authorization