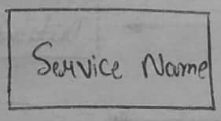
SYSTEM DESIGNING

☆ Introduction :-

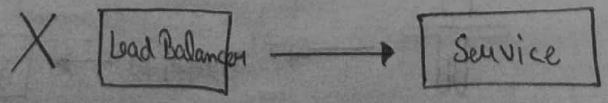
- To Prepare you for System Design Interviews in top Companies
- To help you to make better architectural Design Decisions at your Job.
- To improve your skills of analysing architectural Diagrams.

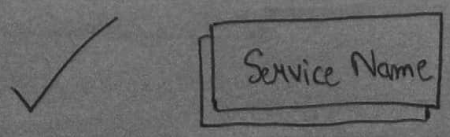
☆ Diagram Building Blocks :-

① Service :-



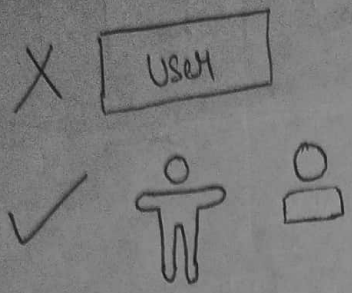
② Load Balanced Service :- Sometimes we don't run a Single Service but multiple Services at same time by putting Load balancer behind it, but it is wrong.



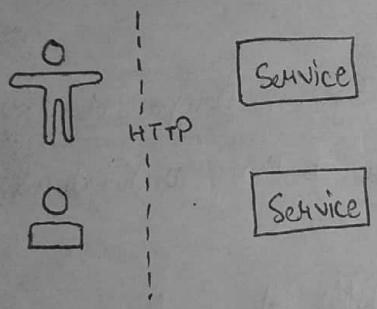


(ie cluster of Services)

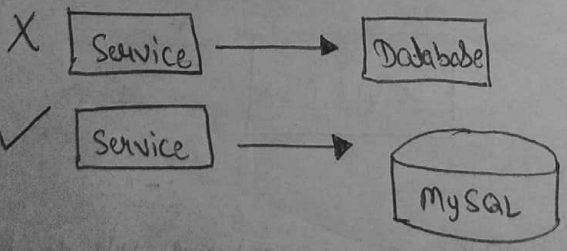
③ User :- feel free to draw them, but different symbol for different User Categories.



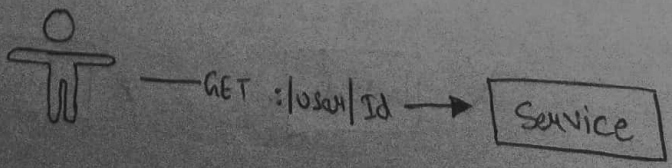
④ Great Network Divider :- Now Users generally Communicate over a Server and it is good to put divider b/w User and Service by giving protocol name, that which protocol we use.



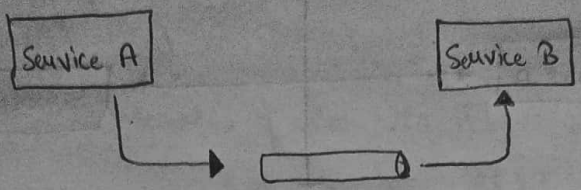
⑤ Database :- It is better to draw the diagram of database with Name.



⑥ Synchronous channel :- These are http Calls drawn by Solid Arrow.

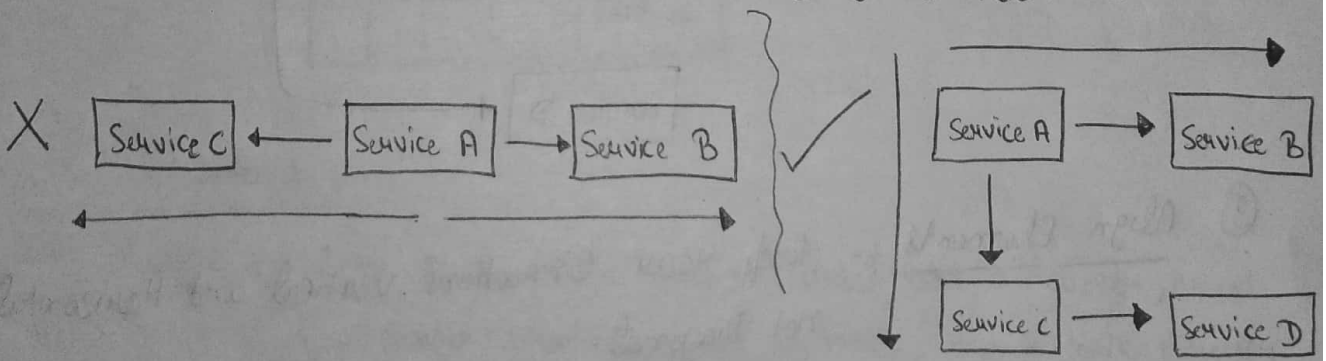


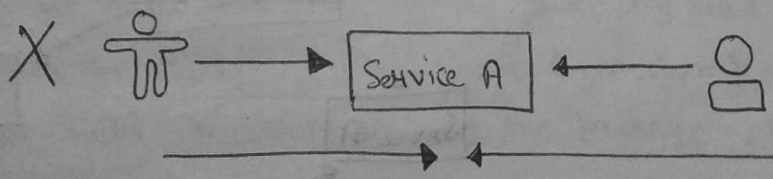
Queue :- It can be represented via dotted line or using barell.

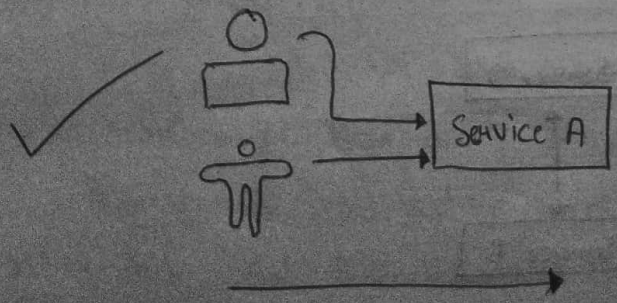


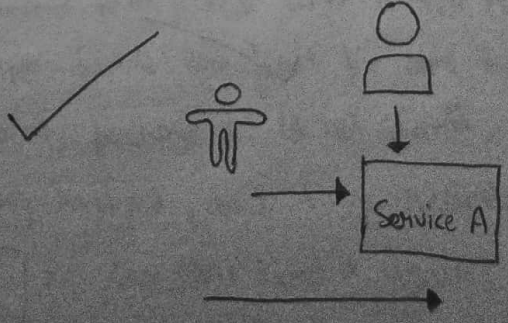
☆ Diagram Flow :-

① **Maintain Direction** :- Make the flow of your diagram one direction, left to right and top to bottom bcz it will be easier to understand.

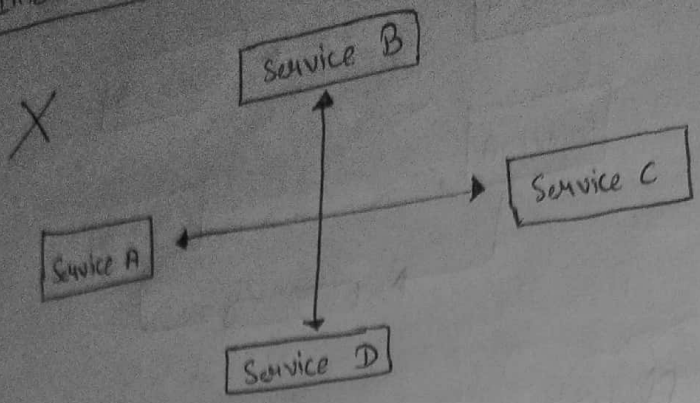


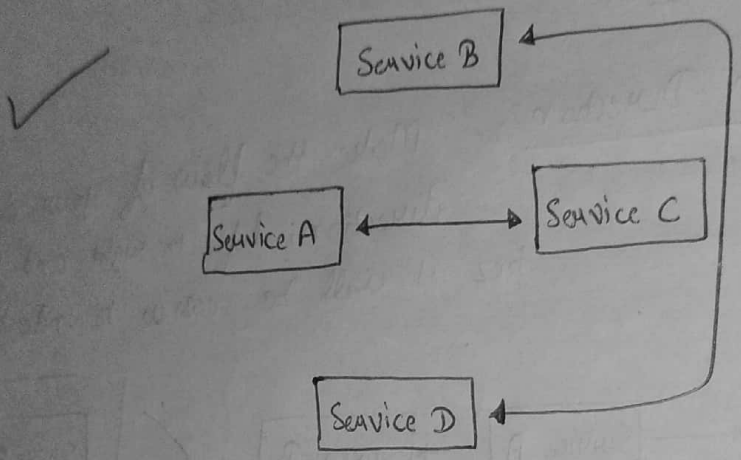




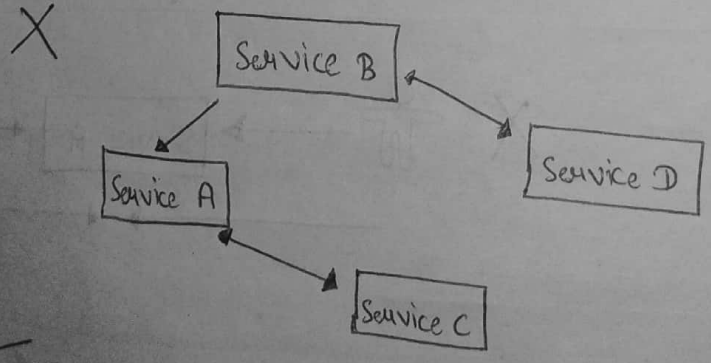


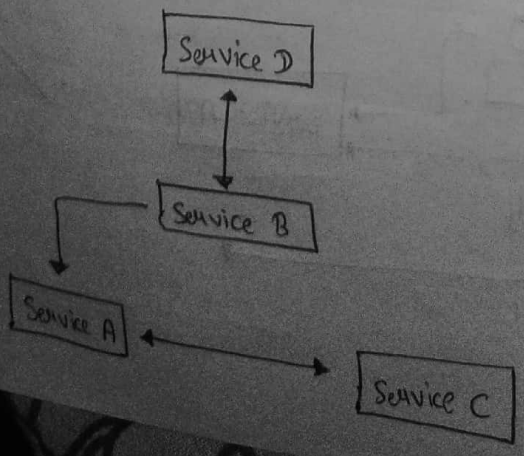
② Avoid Intersections :-





③ Align Elements :- keep your Connections Vertical and Horizontal not Diagonal.

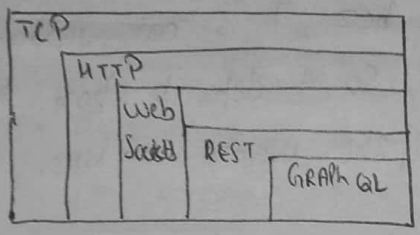




Networks :-

① **TCP** :- Transmission Control Protocol and it is a most common protocol used in Distributed Systems and we mostly used TCP, it means
- if we use HTTP, we use TCP
- if we use REST, We use TCP
- if we use GraphQL, We use TCP
- if we use WebSockets, We use TCP

that means TCP is everywhere.



- **TCP Advantages** :-

(i) **Reliable** :- Lets suppose we have a Sender and a Reciever, Now if a Sender wants to send an image, than it can't directly send the image, bcz first the payload will breakk into pieces (i.e. image), Now we will send the image piece by piece, and if Reciever does receive the package due to some Connection problems, TCP will send it again until receiver recieves the package, that makes it reliable.

(ii) **ordered** :- Now Let the payload divided into pieces with order no. 1, 2, 3, 4 and when Sender sends the first pic and than Second pic than due to some reasons package 2 is not sent, than it will send package 3 as nothing happens, but than Reciever asked that i got # 3 but dont get # 2, So can you send it again?, Hence it helps us to recognize which packets we missed, which makes it ordered.

(iii) Error-checked :- TCP is also has a checksum for each

Now let us suppose we have 4 packets and

Sandar sends the #2 package of size 21B but due to some

Reasons it get corrupted, than TCP checks the 'checksum' of

both the packages (i.e. before and After Sending) and if it is

Corrupted than Recievar will say, I got 8B image, but you

me it should be 21B, Can you Send it again?, This is how

helps in error checking.

(2) UDP :- User Datagram Protocol, it is fast as Comp

to TCP, bcz it is not reliable and ordered b

it is fast, So it depends upon the System designe

which protocol he/she wants to use.

Where UDP used :- It is used for Constant stream of data

or Sending a lot of data fast in less time

(i) Monitoring metrics.

(ii) Video Streaming.

(iii) Gaming.

Imp (iv) Stocks Exchange.

Imp ★ Note :- Can you Afford Losing Some data?, than Surely you can use

UDP, bcz it is fast as Compared to TCP.

TCP	UDP
(1) Expects Confirmation from Receiver.	(1) Just Sends packets of information.
(2) Numbers each message (i.e ordered).	(2) Unordered
(3) Add checksums for each message.	(3) Good, when there is Constant Stream of data.
(4) Slow as Compared to other protocols.	(4) it is fast.

☆ Load Balancing :-

- Load Balancer is a machine that runs a reverse proxy software.
- Goal of the software is to distribute the requests b/w multiple servers that host the actual application.

☆ Interview :-

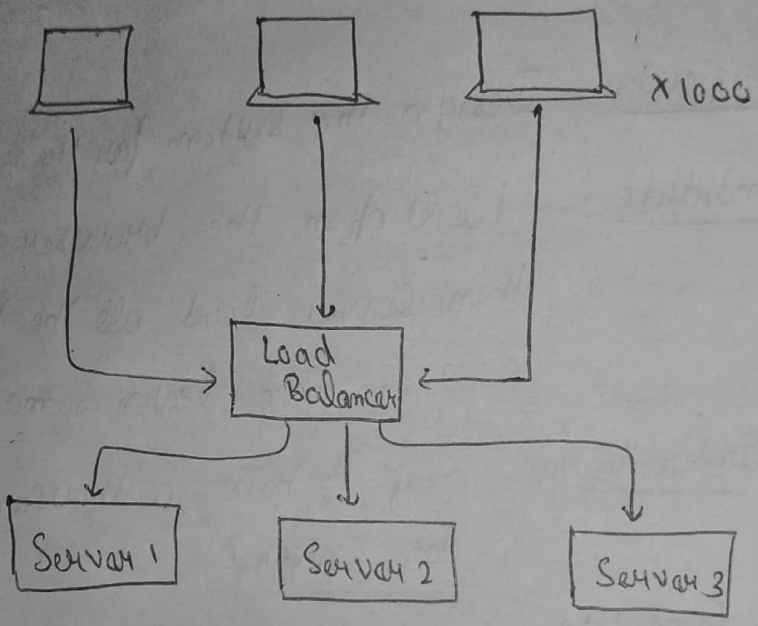
- Interviewer:- Design the system for how Udemy works?
- Candidate :- i will open the browser and type Udemy.com and then server load all the pages and videos and then a user can learn something New!
- Interviewer:- Say u have a thousand requests, will one server be enough?
- Candidate :- Of course not, we have multiple servers.
- Interviewer :- Then how user knows the tasks of each server?
- Candidate :- Yeah well, i will placed one server in the middle of client and all the servers which will accept all the requests from the client and it is known as Load Balancer.

Now, Is Load Balancer knows which request sent to which server? and for this, Strategies come, and Every Load Balancer different Strategies.

- Round Robin
- Least Connections
- Resource Based
- Weighted variants of the above.
- Random

Now we will understand the most common algo... i.e "Round Robin"

⇒ In this strategy, Load Balancer will distribute the load among different servers in Round Robin format.



⇒ Types of Load Balancers :-

① Level 4 :-
- Transport Layer Load Balancer
- Has Access to: TCP or UDP, IP, Port
- Low Level

② Level 7 :-
- Application Layer Load Balancer,
- Has Access to everything layer 4 has
- In Addition, has Access to HTTP headers, Cookies, Payload.

Examples of open Source Load Balancer :- NGINX, HAPROXY, traefik, etc.

dvantage :- (i) Resilience :- When if our one of the Server is down (i.e. fails to load) then load Balancer will route the incoming request to another Server.

(ii) Scalability :- Load Balancer makes your System Scalable, it means Suppose we have 1000 clients and 3 Servers, but suddenly our clients gets double (i.e 2000) So, you can simply add more Servers connecting to same Load Balancer, which we called Horizontal Scaling.

★ CDN :- We will understand by [redacted] giving an example, let Suppose we are going to open Udemy.com, but it will take different time in different Countries to open it, bcz like Udemy Server is located in USA, then users from USA will experience good Quality but like from india, users can't get good Quality, bcz data has to travel more for india.

- So, we can Solve this [redacted] issue by using more Servers, like Servers in every Country where we want to give our Service but it will be Very expensive.
- So we will use CDN, CDN stands for "Content Delivery Network." its ideology is very Simple, that a user will get data/Content from the nearest Server ASAP.

It is used to deliver Static Content :- Images, CSS, HTML, Javascript.

⇒ Two Types of CDN :-

(i) Push :- It is used to just upload a data to all [redacted] Servers and then CDN distribute to all the Servers.

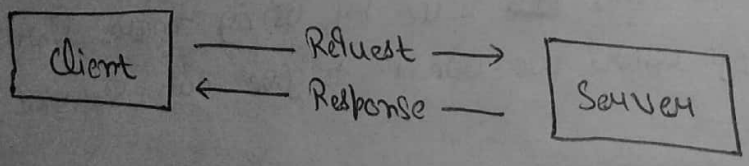
(ii) Pull :- When users opens your site and wants to download some data then if CDN server has data, then it will Serve it to user.

Note :- • CDN's allow placing your static assets
• Reduce Costs
• Decreases Latency
• Increases Complexity of your System.

Sometimes, it increases the Complexity of our System bcz before using CDN, we will directly send data from the server, but now we have to push it to CDN or make sure CDN approved these data files, so it will increase Complexity of our System.

★ Protocols :-

① HTTP :- (i) HyperText Transfer Protocol
(ii) HyperText → text with links to other Documents
(iii) Based on TCP
(iv) Request-Response Protocol
(v) Layer 7 – Application Layer



⇒ Request :- (i) Method
(ii) URL
(iii) Headers
(iv) Body (optional)

→ Response :- (i) Status
(ii) Headers
(iii) Body (optional)

⇒ HTTP Methods :-

GET /superheroes/123 ≠ DELETE /superheroes/123

(i) GET → read
(ii) DELETE → delete
(iii) POST → create
(iv) PUT → Update
(v) PATCH → partial Update

GET :- /superheroes/123

⇐

{ "id" : 123, "first name" : "Bruce", "Last Name" : "Way" }

Post :- /superheroes

{ "firstName" : "Bruce", "Last Name" : "Wayne" }

DELETE :- /superheroes/123

PUT :- /superheroes/123

"for Updation"

Note :- PUT is similar to PATCH, but PATCH is used for partial Updation, like if we just want to update first name, than PATCH will work.

⇒ HTTP Status Codes :-

- 100 - 199 Informational
- 200 - 299 Succesful
- 300 - 399 Redirection
- 400 - 499 Client Error
- 500 - 599 Server Error

⇒ 200 - 299 Succesfull Codes :- • 200 ok
• 201 created

⇒ 400 - 499 Client Error Codes :- • 401 Unauthorized (un Auth
• 403 Forbidden (No Perm
• 404 Not found (no Page

⇒ 500 - 599 Sarver Error Codes :- • 500 Internal Sarver Error
• 503 Service Unavailabible

⇒ 300 - 399 Redirection Codes :- • 301 Moved Permanantly

(2) REST :- It is not actually a Protocol but it is just a guidline to how to structure your API.

Representational State Transfer

- Built on HTTP
- Standard URL structure
- Standard use for HTTP Verbs.
- HTTP status codes to indicate errors
- JSON as body format.

⇒ Before REST :- So many ways To Represent Same Action

Example for Deletion :-

GET | ?id = 123 & action = delete & type = user

Post |
Body = "delete"; type : user;"
Cookie = "userId : 123"

DELETE | ? id = 123 & type = user

⇒ After REST :- DELETE | user | 123

⇒ Restfulness :- RESTful Verbs

- GET
- POST
- PUT
- PATCH

RESTful URL's :- METHOD | [resource | id]

GET /users/123 and not /user/123 or /getuser/123

RESTful URL's :- Nested Resources

METHOD | [resource | id] | [resource | id]

DELETE /users/123/books/567 → return book that was borrowed by user 123

GET /users/123/books → fetch all the books borrowed by this user

RESTful URL's :- state

PUT | [resource | id] | [action]

Example :- PUT /users/567/enable

PUT /users/567/disable

RESTful URL's :- idempotency

PUT /users/123/online → we can set it true/false, so it will be idempotent (i.e. action)

POST /users/123/likes → it is not idempotent

So, we can see where to use PUT and where to use POST

RESTful URL's :- Pagination

GET /[resourceid]? limit = x & offset = y

Example

GET /books? limit = 50 & offset = 100

We can provide pagination in our application using Query in API

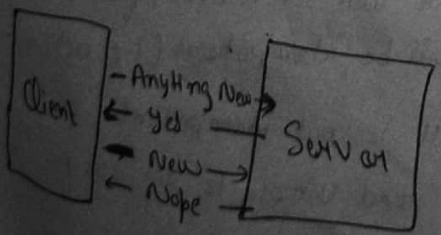
Imp Note :-
- Use pairs of resource-name / resource-id to structure your URL's.
- GET shouldn't change Anything
- Use PUT to switch State.
- Provide Pagination.

③ Web Sockets :-

Now, we know what Protocol aims to solve? it gets the request and then give response, (i.e- User - request - response model) that means Client have to give request and then wait for the Server to give response to terminate the connection.

Now, HTTP is great when we want someone to send a message and then get a response from the Server that msg is recieved but what about receiving messages.,

So, To solve this, Web Sockets came to an action, it is a duplex protocol, when the connection establish b/w Client and Server then messages can be simultaneously.



HTTP

Web Sockets

Advantages :- • Connection is established only once.
• Real time message delivery to the client.

⇒ Disadvantages :- • More Complicated to implement than HTTP. it means if there is network Error then we have write code to make Connection established
• May not have best support in some Languages.
• Load Balancer may have some troubles.
• Unlike REST, need to reinvent "the Protocal" every time.

Imp Note :- • Built on top of TCP, Same as HTTP
• Duplex (two way) Protocol.
• Persistant Connection.
• More efficient than polling with HTTP.

☆ Concurrency :- Parallelism ≠ Concurrency

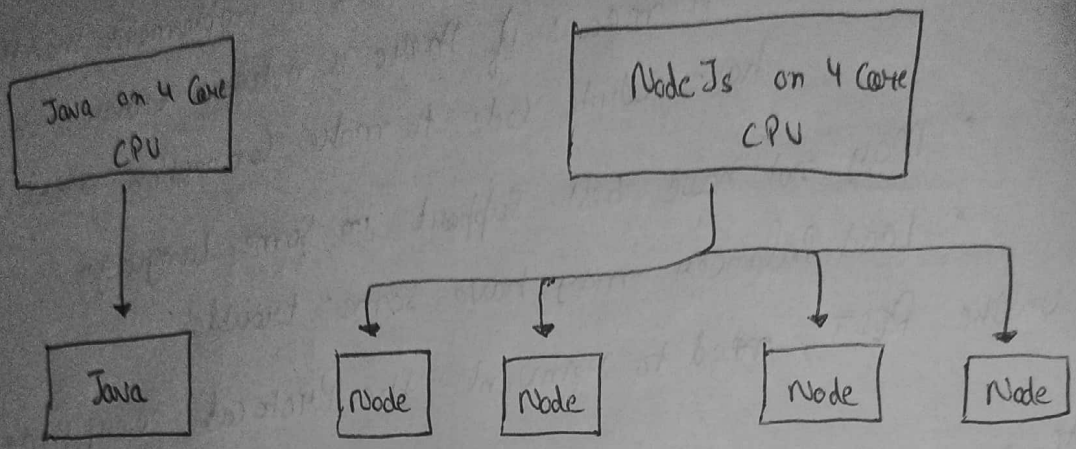
• Parallelism :- Doing more than one thing at the same time.
• Concurrency :- Providing an illusion of doing more than one thing at the same time.

Example :- ① Suppose you and your friend are in a kitchen, you are making a cup of tea and your friend is making a Sandwich on same time, both works are different and can do parallelly

② And if you are alone, you will starting making Cofee, then go to make Sandwich then again go back to check the Cofee and you will do this in parallel way by switching yourself from Cofee to Sandwich. Just like that our CPU works by switching b/w the processes.

★ Processes :-

- Each has Separate Memory Space



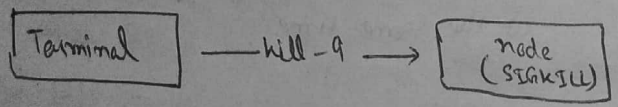
⇒ Interprocess Communication :- if processors cannot share memory, then Communication built

- File (also memory mapped file)

When Servers don't want to share, we will create a single file and put all the data coming from Servers in that file.

- Signal

$ kill -9 818



There are other more method but we will learn these two

★ Threads :- we already done this topic in OS & Java in very detail.

DATA BASES :-

⇒ **Indexes** :- Why do we need indexes?

⇒ To get a information at particular Column/Row. else we have to search entire table row by row.

What type of data structure an index is?

⇒ A index should be a hashmap but not usually, bcz in a hash map, we can find value at particular level/point. but if we want to search b/w two points then like,

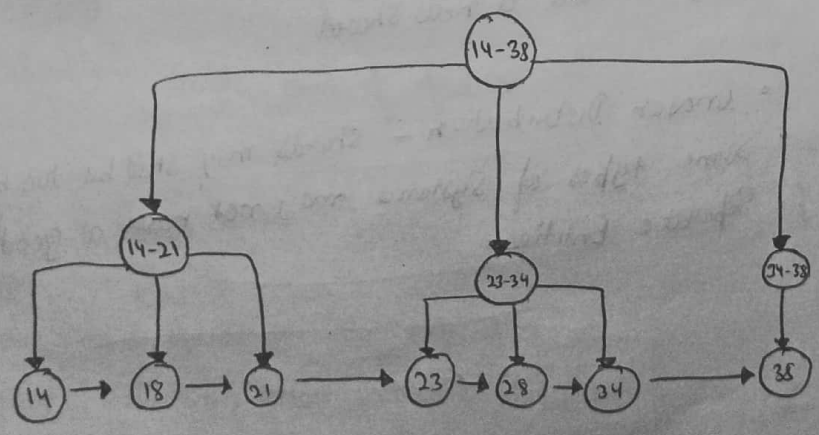
Select * from videogame_characters where age between 25 and 30,

So, we can say hashmap is not suitable, than "Tree can be a data structure for it", but which tree? it is B^+ tree,

Why B^+ tree?

⇒ Select * from members where age between 20 and 30?

What makes B^+ tree suitable for it, Is Arrows using in it.



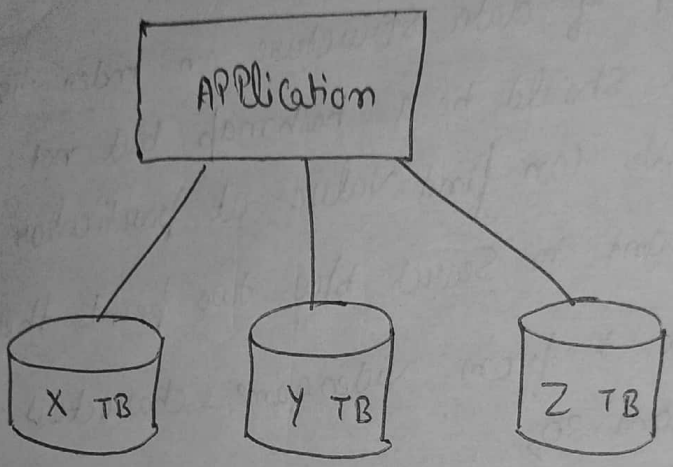
All leaves are ordered and inter-connected.

Use of these Arrows :- Like if we want 21 and 28, than we will go on Left side, search for 21, than we dont have to go back, We can use these Arrows to get 28

⇒ Sharding :-

Suppose when our projects grows and then we have to do sharding (ie. break big database into small pieces).

Different Database for different Services



Now, The Problem is, How our Application knows to use which database, for this we have two Strategies.

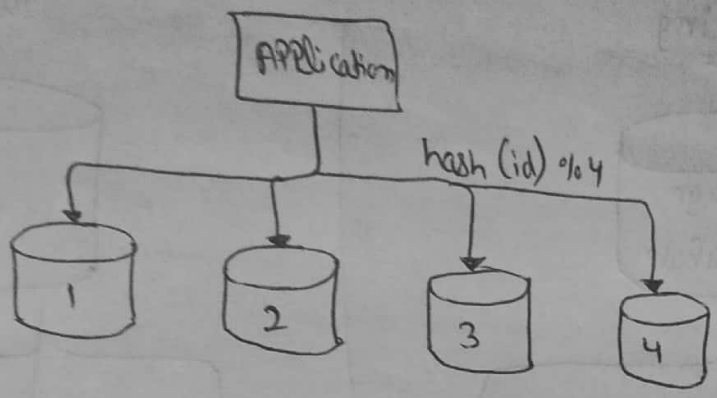
(i) Tenant Based Sharding :- it is used for that Systems which have partition b/w Entities.

Eg :- We do in Taxi-Hailing APP.

Pros :-
- Easy to understand
- Easy to add a new shard

Cons :-
- Uneven Distribution - shards may still be too big to handle.
- Some types of Systems may not have a good way to Separate Entities.

Hash Based Sharding :- We will perform Hashing before distribution into the Databases.



Pros :- • Even Distribution
• Works well for key-value Data

Cons :- • Adding new shards is Difficult, bcz each time we have to change the hash function
• Weaker Consistency (No foreign keys)

Shard LookUP :- To Do this, we need a Locator, that will be known as Shard Locator, in which we program it to do sharding.

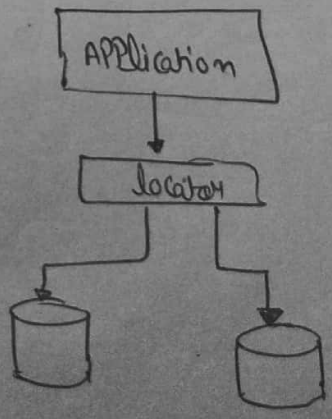
Pros :- Simplifies adding new shards, bcz we don't need to keep the logic in our application.

Cons :- Single point of failure

• **Down sides of sharding** :-

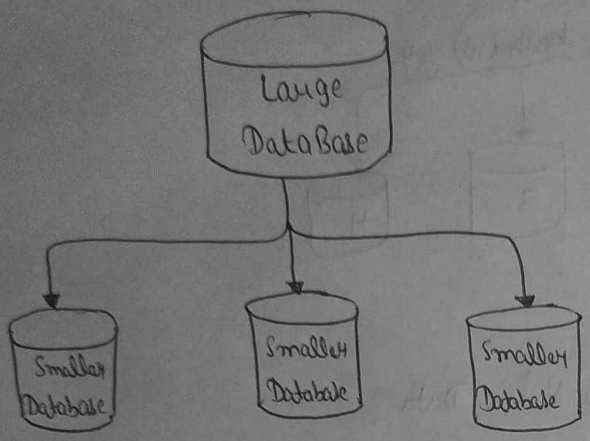
(i) Added Complexity
(ii) Cross shard operations are expensive
(iii) Data still may become unbalanced.

Like :- if we have even distribution of users but users may differ Some are Writers and else are viewers which ends up with different size shards.

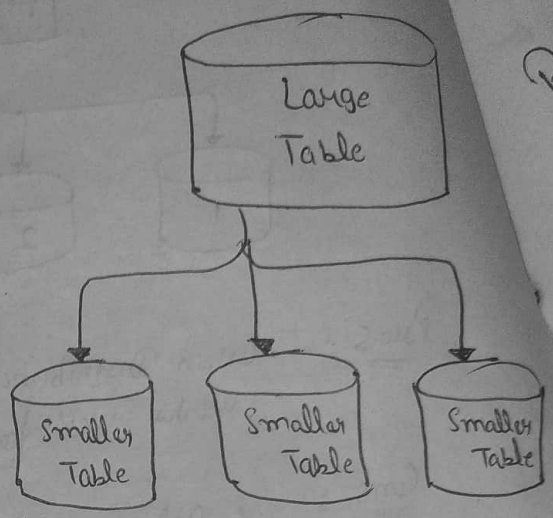


⇒ Partitioning :-

Sharding



Partitoning



Pros :- (i)

Cons :- (ii)

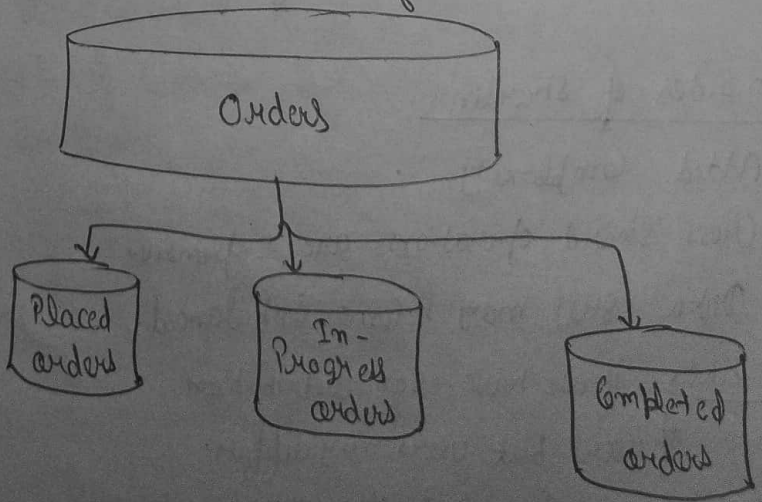
Partiot

Benefits of Partitoning :-

- Smaller files == faster queries (ie. users will be happy)
- Smaller indexes can fit into memory
- Dropping Partitions is fast.

Partioning Strategies :-

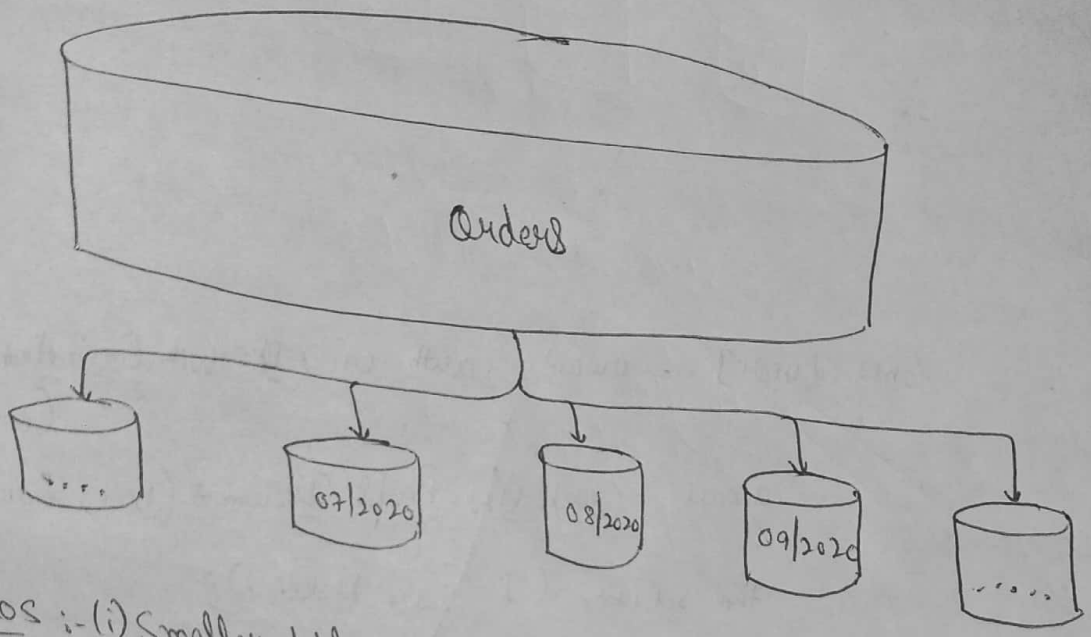
(i) List of Values :- Partioning by list of values



Smaller tables = faster Queries

Cons :- (i) Uneven data distribution.
(ii) Need to move data between tables.

Partitioning By Range of Dates :-



Pros :- (i) Smaller tables = faster Queries
(ii) Great if you want to get rid from old data

Cons :- Still, Uneven key Distribution.

Partioning by hash of a key :- As same as Hash-Based Sharding

Pros :- Even key distribution

Cons :- Changing no. of partitions is hard.

Down sides of Partioning :-

(i) Complexity of maintainance
(ii) Scanning all Partions is Expensive
(iii) Harder to main Uniqueness; Each Partion has its own Separate table.

Design a Taxi Hailing APP (AKA UBER)

★ Problem Introduction :-

Main Topics :- (i) Marketplace, (ii) Geolocation, (iii) Geosharding

Interviewer:- Design a Uber App for me? (i.e. user Perspective)

Candidate :- A user will open up the application then do Authentication and then he/she can select a drive and book it to reach the destination.

Now, **Marketplace Aspects :-** (i) Demand / Supply
(ii) Atleast two very Different kinds of users (i.e. riders and drivers)

Drivers Perspective :- We may send request to one driver at to multiple drivers on the same time at we can send request to that driver which is nearby.

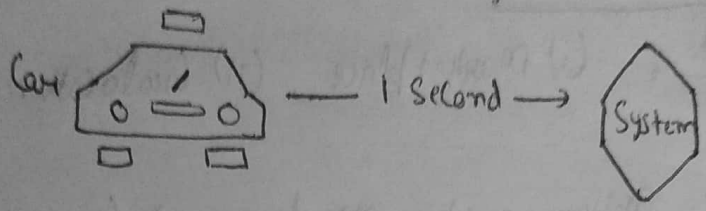
Note :- During Interview, we have to all the options to the interview, bcz it tell us about, how far we think.

Market place: Riders :- (i) 100 M Riders in total
(ii) 10 M active Riders
(iii) 10 rides per month on Average

Market place : Drivers :- (i) 1M Drivers in total
(ii) 500k active Drivers
(iii) Average shift : 6 Hours

★ Drivers Locations :-

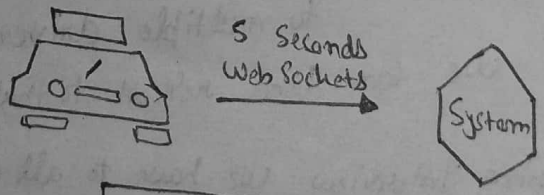
- **Frequency of location updates** :- We need the current locat
  of Driver after each 5 sec



- **Communication** :-
  - REST (Not Neccesarily)
  - Web Sockets
  - UDP (much better)

  We can use either Web Sockets | UDP, but we will stick to Web Sockets, bcz it will be more preferable for future requirements bcz it allows duplex communication.

- **Payload** :- Interviewer may ask, which type of data you will send



latitude	float
longitude	float
direction?	float
driver_id?	uuid

Through Put :- (i) 500k Active Drivers

(ii) Average Shift :- 6 Hours

(iii) 4 "Shifts" a day

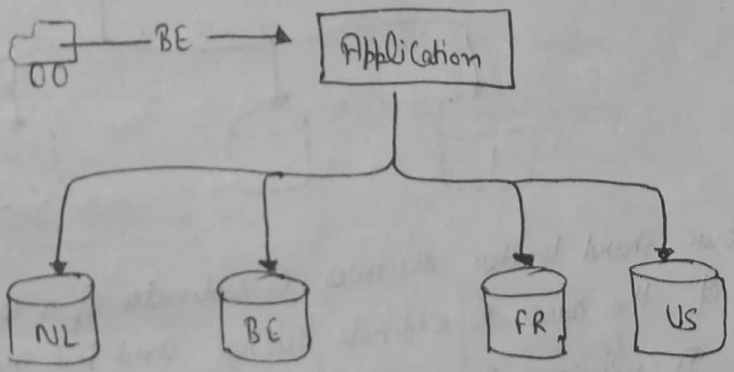
(iv) 125 k drivers at every moment on Average

(v) Message every 5 seconds (i.e. Current position of driver)

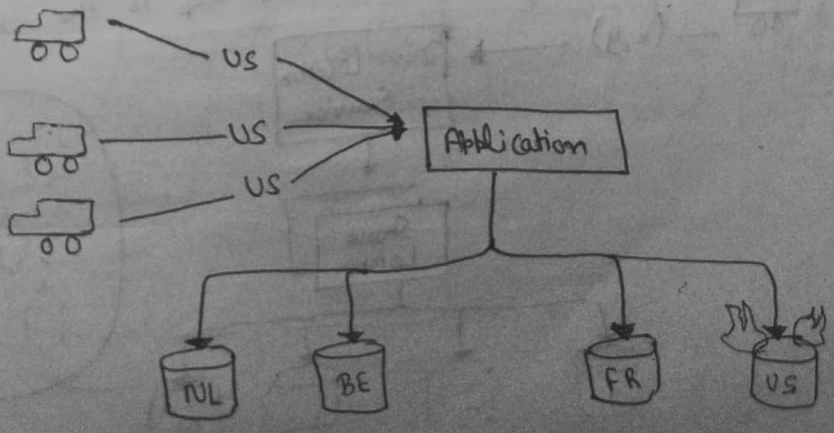
(vi) 25 k requests per second

★ **Storing Locations** :-

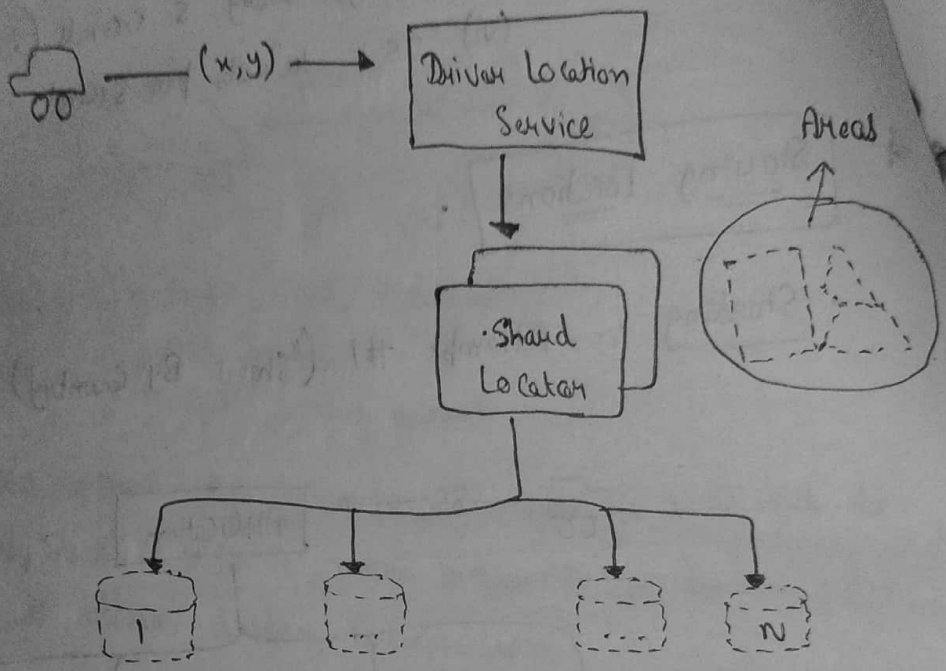
- Sharding :- Attempt #1 ("Shard By Country")



This is a most Common Approach (i.e. shard by Country), we will pick the shard for that Country and write location in it, but it also has a Disadvantage, if most of our drivers are from single shard and it is called hotspot (i.e. most of the requests sent to one shard).



⇒ **Attempt #2** :- Alternative would be used Custom Areas, ... work if our Company has lots of employees. In ... We will divide the map into shapes [redacted] as big as ... or small as States.

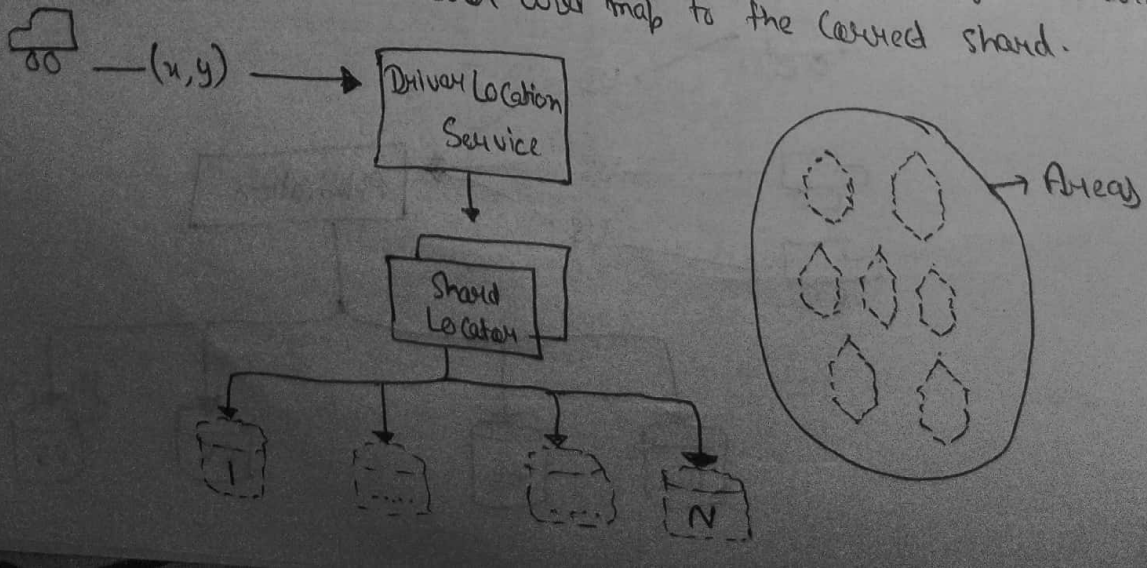


When our shard locator recieves Co-ordinates of a Driver, it check in which of the areas Co-ordinate belongs and but them in correct shard.

Problem :- It requires lots of maintainance.

⇒ **Attempt - #3** :- In the we can use Geospacial Libraries like Uber h3, Google S2.

Uber h3 :- Divide the world into hexagon shapes having same area, and then shard locator will map to the correct shard.

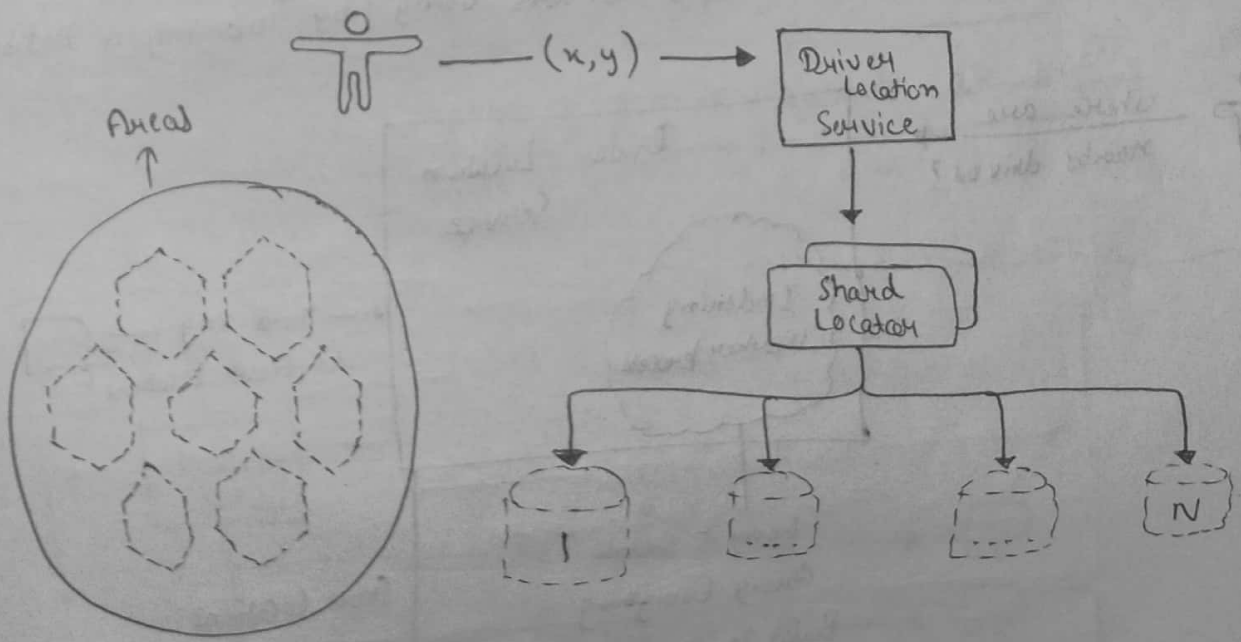


Taxi Around You :-

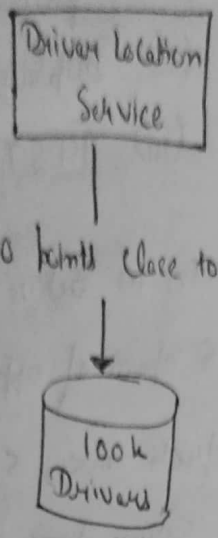
- **User Perspective :-**
  (i) Refresh every 5 seconds
  (ii) Display 10 drivers almost
  (iii) REST Based

- **Through Put :-**
  (i) 10 m active Riders
  (ii) 5 opens of app a day per user = 50m opens a day Total
  (iii) Refresh every 5 seconds.
  (iv) Average Session time is 1 minute = 12 refreshes
  (v) 50m * 12 = 600m requests per day
  (vi) 600m / 24 = 25m requests per hour
  (vii) 25m / 60 = 400k requests per minute
  (viii) 400k / 60 = ~7k requests per second

- **Getting Drivers Positions :-**

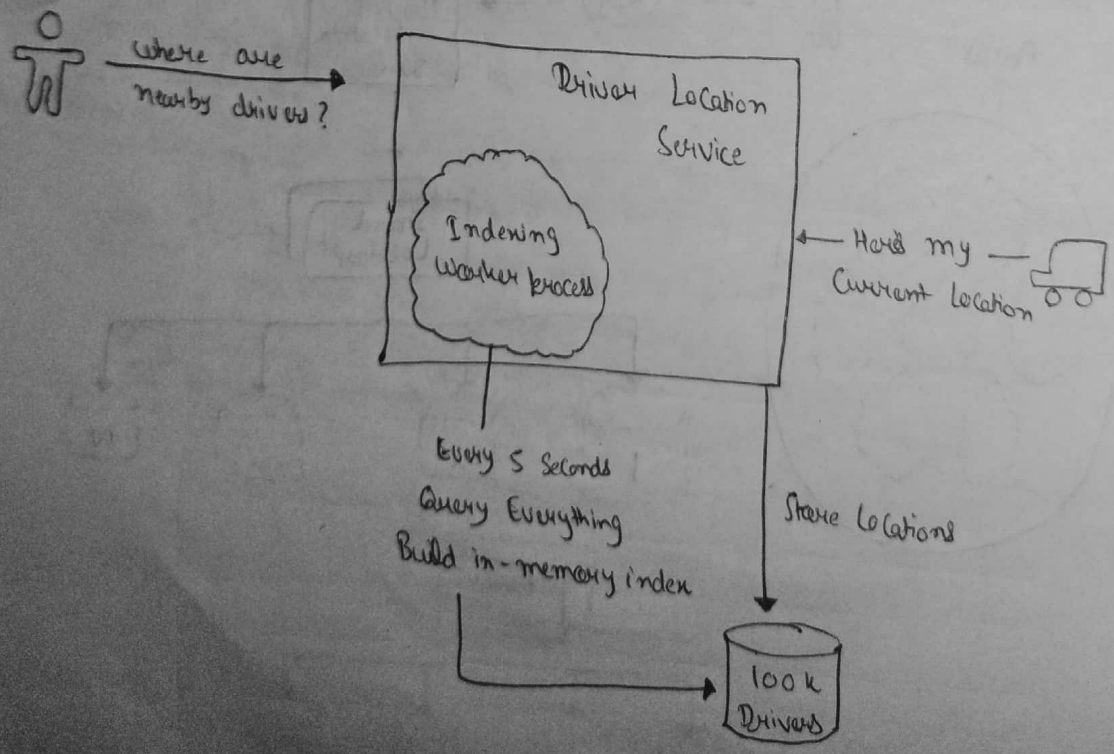


Option #1 :- Geospatial Search in DB → Database that can do for locations search (i.e. MongoDB, MySql,

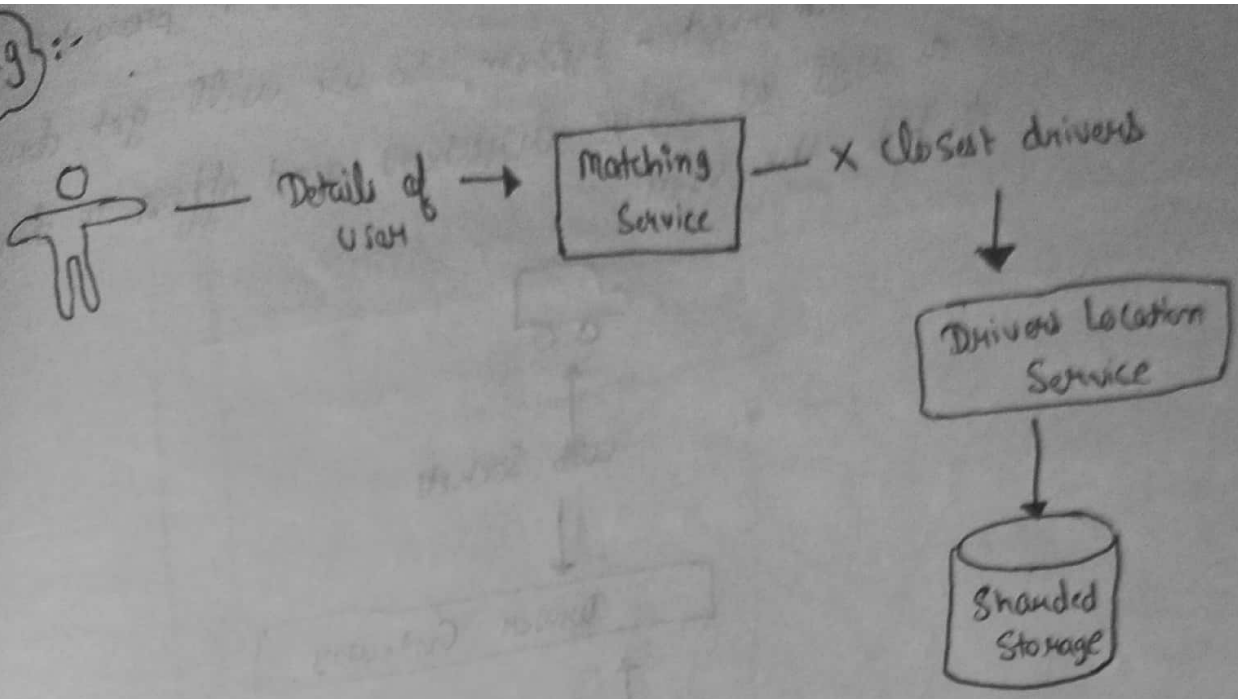


Option #2 :- In-Memory Geospatial Index (Safest approach)

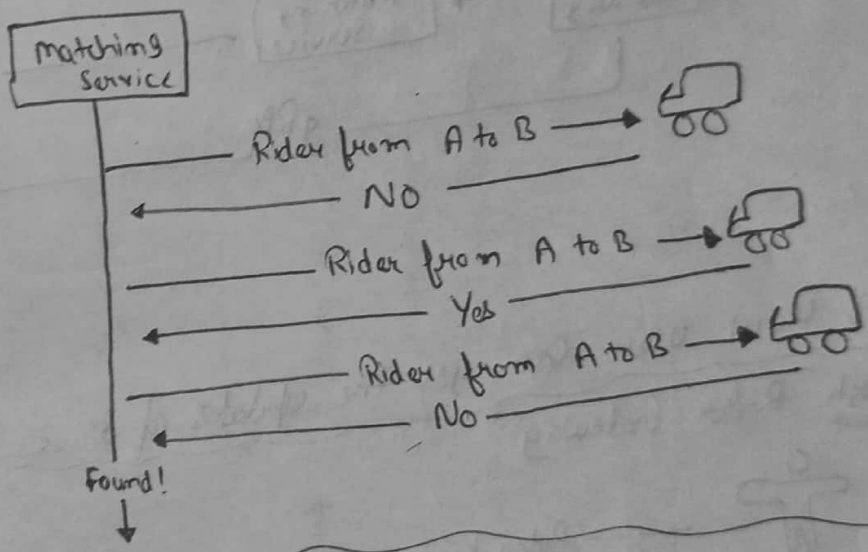
In this Driver Location Service will get the location of the Driver and then store it in the Correct shard but on a particular index using Indexing Worker process (We can say) and then User gets nearby Drivers using / by Querying a Database



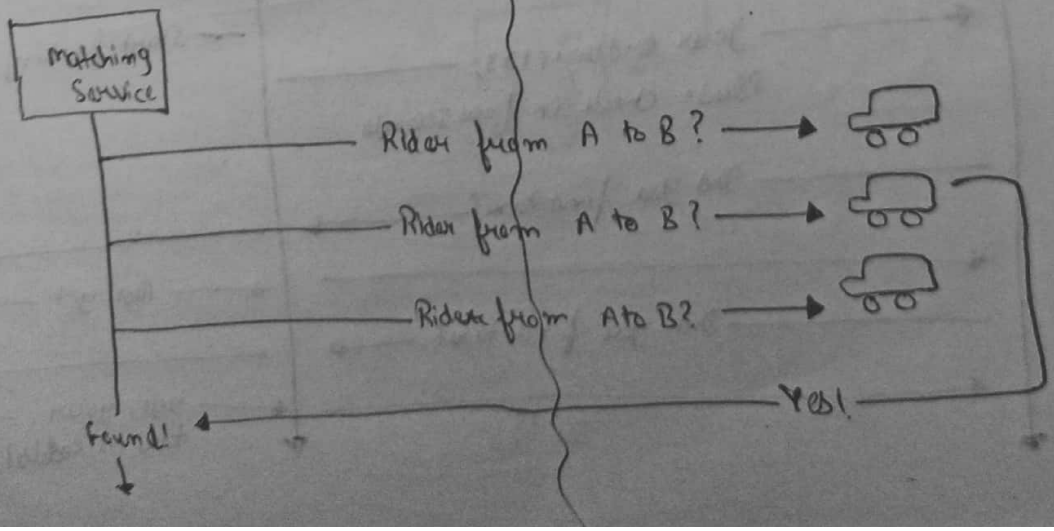
Matching :-



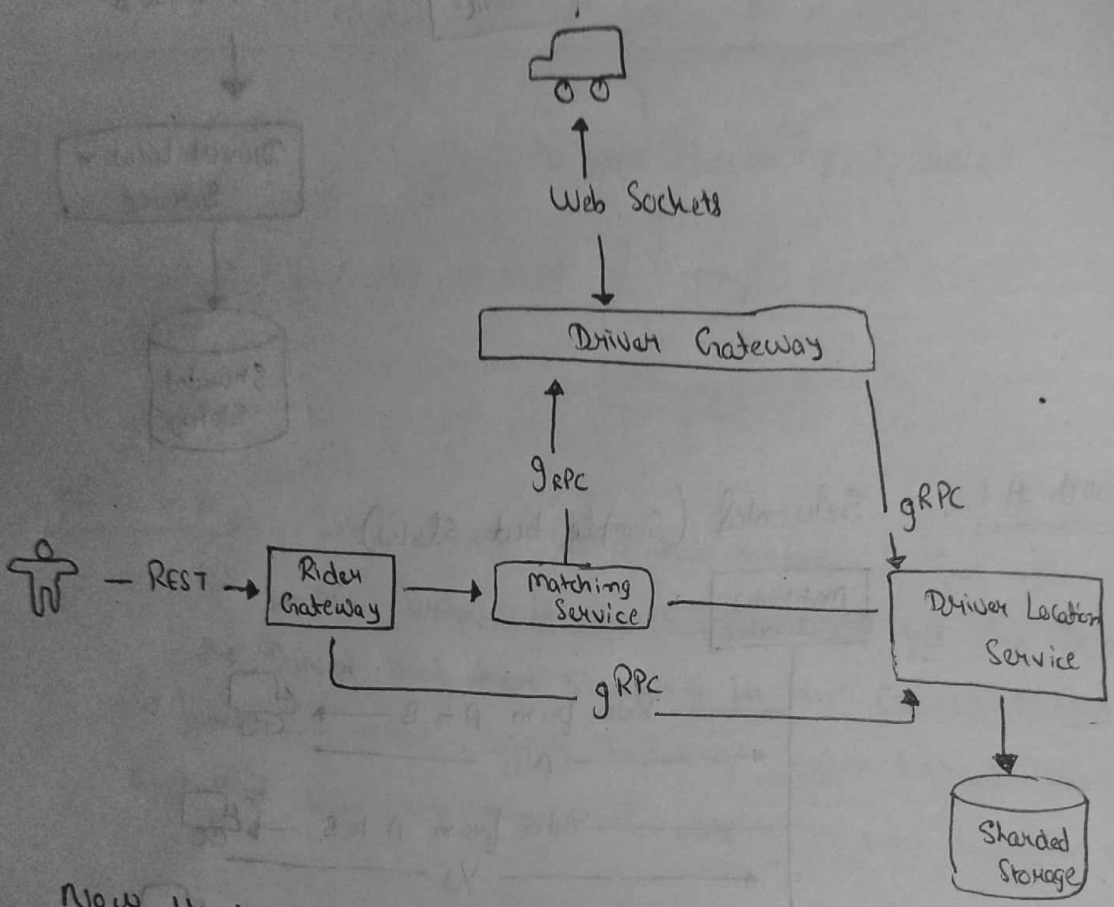
Approach #1 :- Sequential (simple but slow)



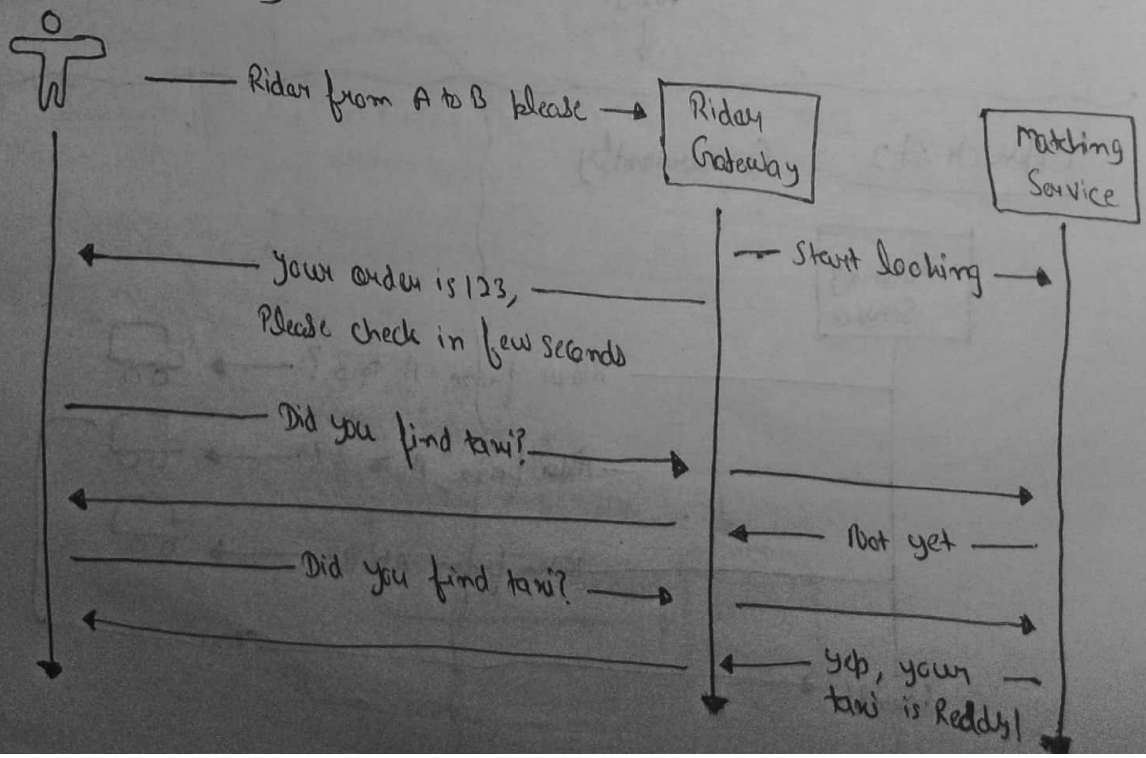
Approach #2 :- Concurrently

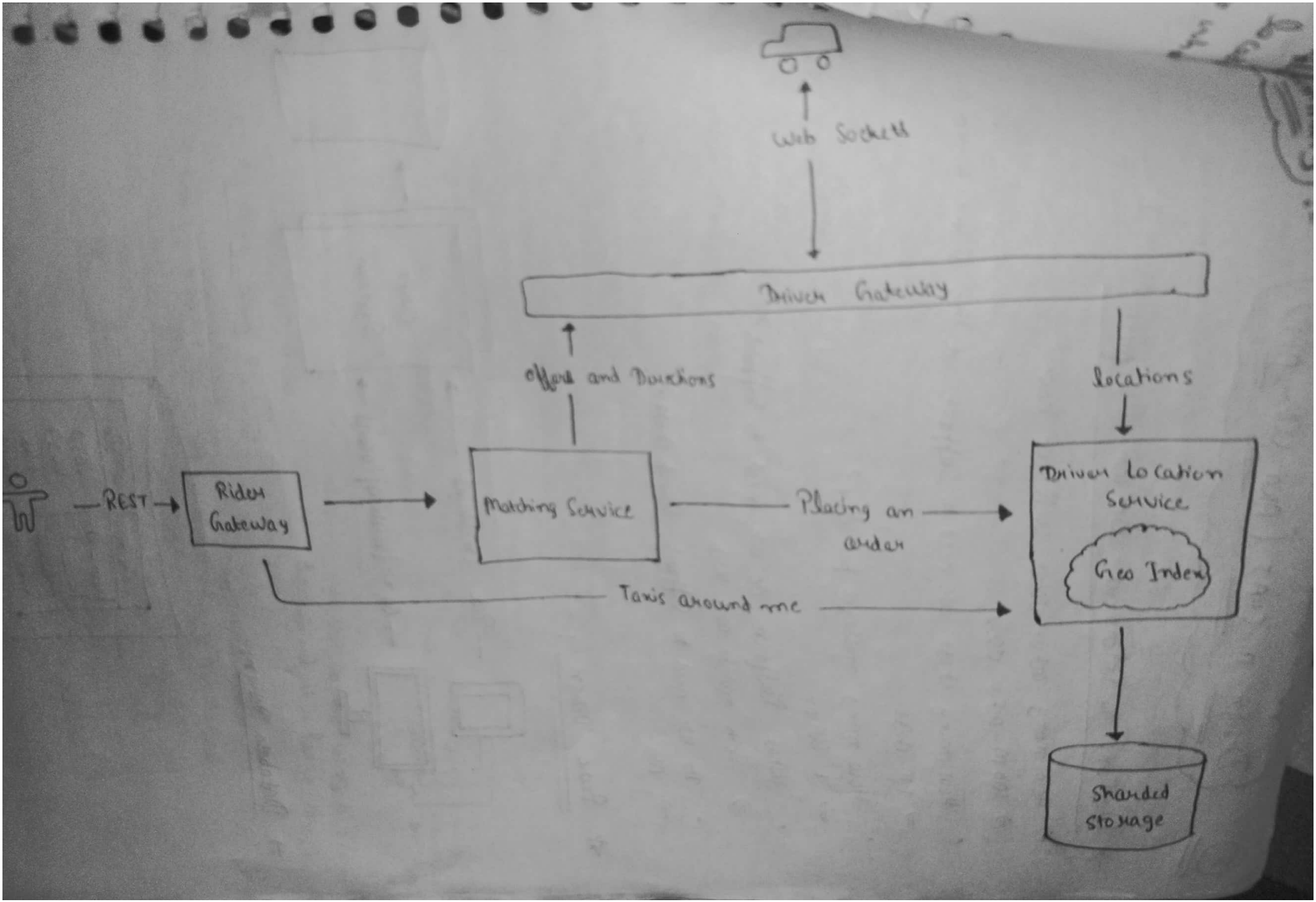


Driver Gateway:- Will use Web Sockets, bcz it provides the ... Communication System, so we will get drivers ... as well as give directions and offers to driver to ... to pick the user.



Now, How user can get the updates of driver, it can be acheived through Rider Gateway



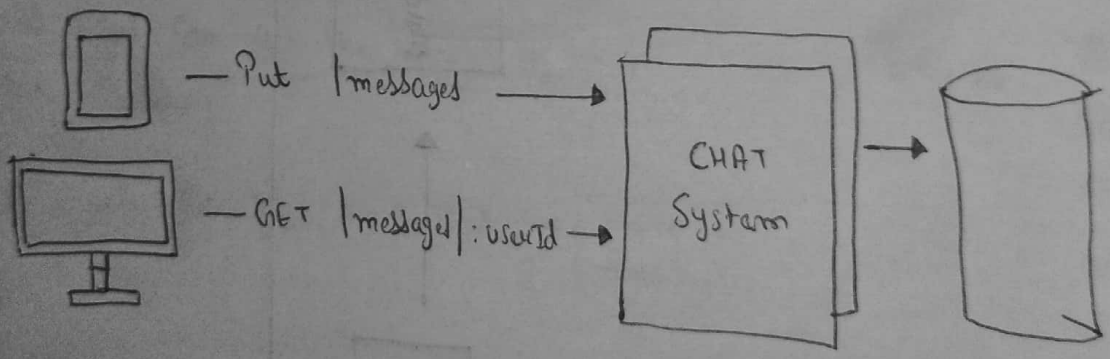


② Design a CHAT (AKA WhatsAPP)

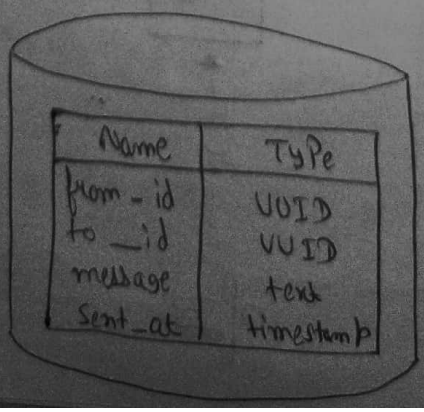
☆ Questions to Interview We can ask first?

- How many active Users will we have?
  ⇒ 100 M active Users
- How many chats we need to support at the same time?
  ⇒ 1 m chats
- How many messages per day?
  ⇒ 1 Billion
- What Platform do we need to Support?
  ⇒ Both mobile and Web
- Do we need to Support Group chats?
  ⇒ No

☆ Basic Design :-



⇒ Database Design:-



Name	Type
from_id	UUID
to_id	UUID
message	text
sent_at	timestamp

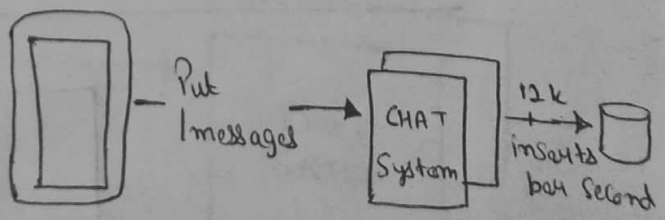
```
Select * from messages
Where (from_user = 'Neo' and to_user = 'Trin')
OR (from_user = 'Trin' and to_user = 'Neo')
Order by sent_at desc
Limit 10
```

- **Indexing**

```
Create index on messages (from_user_id, to_user_id)
Select * from messages
Where (from_user_id = ? and to_user_id = ?)
OR (from_user_id = ? and to_user_id = ?)
Order by sent_at desc
Limit 10
```

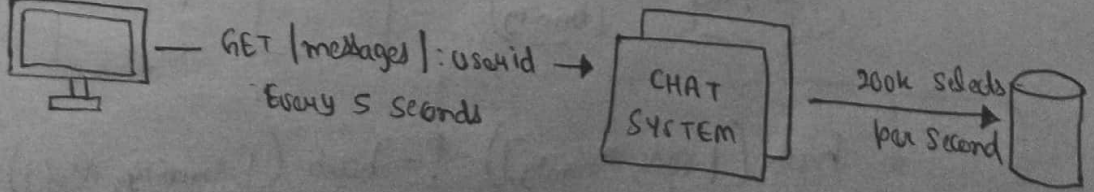
⇒ **Identifying Bottlenecks (Basic Requirement) :-**

1 Billion messages a day
= 40 M messages per hour
= 700k messages a minute
= 12 k messages a second



1 million open chats
Refreshing every 5 seconds
= 1000000 * (60/5) / 60 = 200 k Selects/Requests per second

We can do this using a single database if it is large enough.



⇒ it doesn't goot, actually it is a pretty-bit, so we will look at it.

★ Scaling Reads :-

Now, Earlier, we can see that there is a Load on Database

So, to solve this issue we can introduce a Load Balancer and

DB Replicas.

In this, we will put the DB Replica behind the Dedicated Load Balancer.

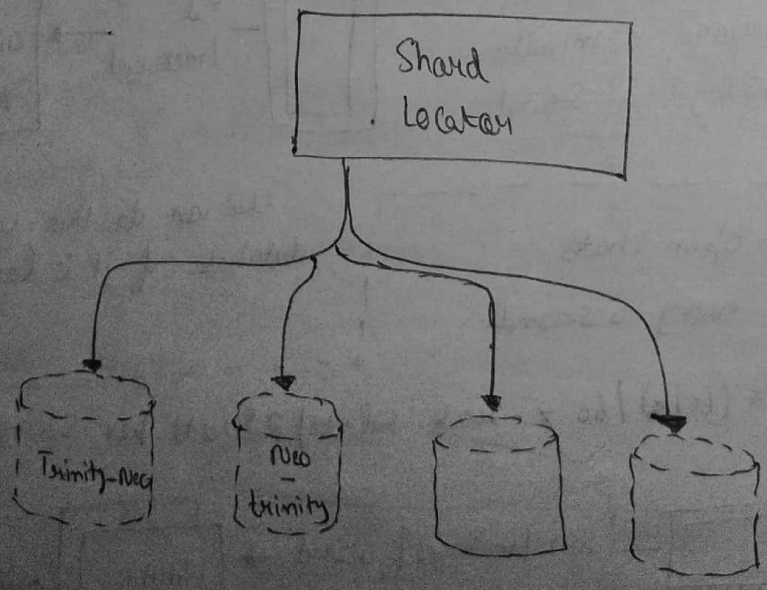
But it also has 2 Disadvantages.

(i) Costly, due to more Databases. (Copies/ Replicas of Same Database)

(ii) As we want to deliver a msg in every 5 seconds, but it will take around 1 minute bcz DB Replicas may face some problems So, it will inconsistant.

★ Sharding chat Messages :-

⇒ Appending Sender and Reciever :-



hash ([Neo, Trinity]) != hash ([Trinity, Neo])

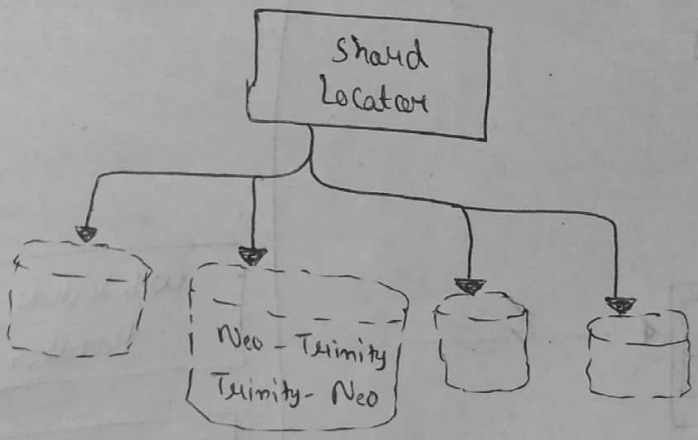
Now, if we sort before Appending

Sort ([Neo, Trinity]) = [Neo, Trinity]

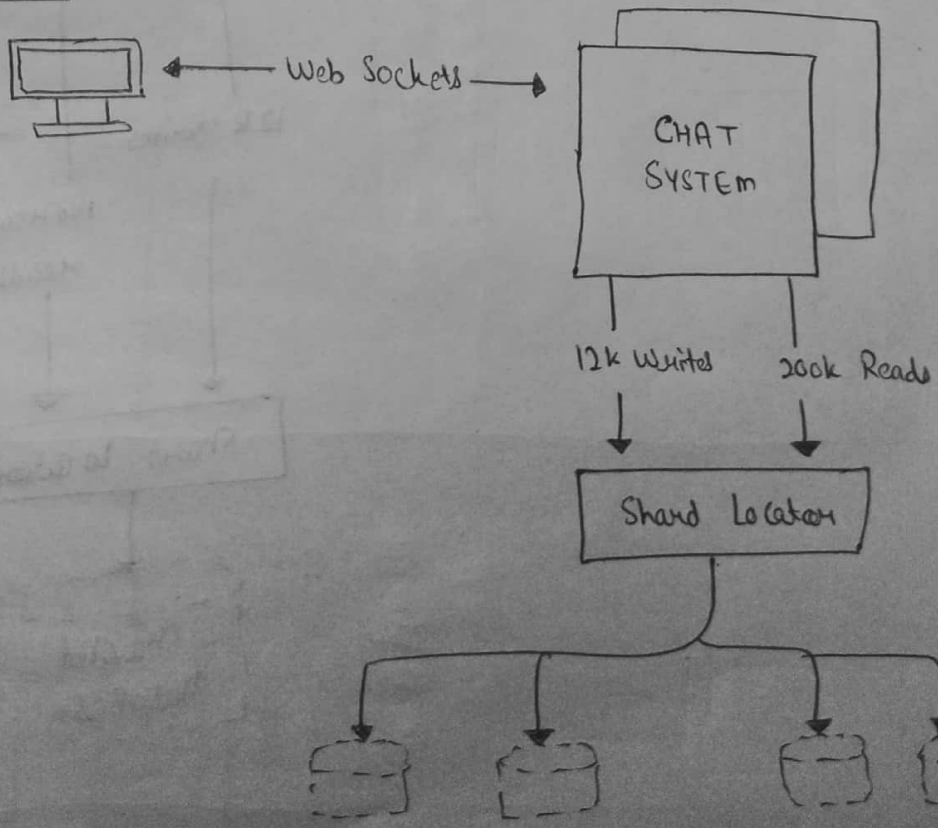
Sort ([Trinity, Neo]) = [Neo, Trinity]

hash (sort [[Neo, Trinity]]) == hash (sort ([Trinity, Neo]))

⇒ Picking Correct Shard key :-



Design - Upto - Here :-



- Our Previous Design is great but if there are million users chat at the same time, So we have to Queue the messages. Each WebSocket Handler is Subscribed to that Queue and will send the message to reciever whenever message Comes.

So, Ideal Design :-