Car-rental services like:-
Uber, ola → these follow one pattern of system design

e-commers services like amazon, flipkart, mynthra... etc., → follow another pattern of system design

proxiy's have → one type of distributed system

Chatting services like what's app, snapchat fb-messenge. → follow another pattern of system design

OTT's like netflix, prime → follow another pattern of system design

search Engine's → follow another pattern of system design.

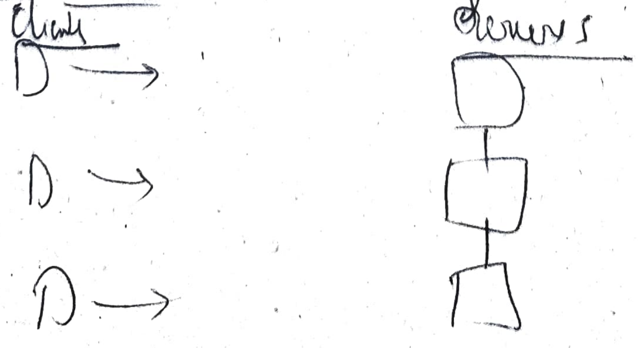
peer to peer → follow another pattern of system design

Scalability:-

based on the no of users, our configuration to host the application should increase or decrease.

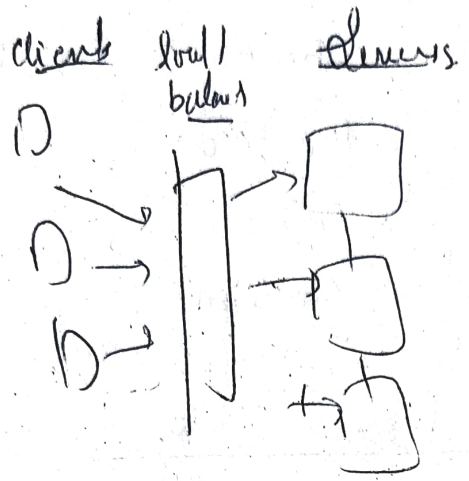
1.) Vertical scaling:-
so this scenario where people are changing the configuration again and again is known as vertical scaling.

~~to~~ load balancer.



when there many clients are coming when had to figure out which server will take how many no. of client requests.

so this brought a concept called load balancer



load balancer: load balancer basically ~~will~~ which basically balances the load for you.

Q) what is load balancer good at?

→ client is not making request to your server directly, it will first go through your load balancer

→ load balancer knows where your read API's should go, and where your write API's should go

→ load balancer is aware of where are the more no. of request are going to which server and where are the less no. of requests are going to which server.

→ let's say if one of the server is down due to an unexpected technical issue, then our load balancer will understand and stops sending any kind of load to the broken server.

HORIZONTAL SCALING

① In this we use load-balancing

Advantages:-

→ if one server is down still not an issue

→ scales well

VERTICAL SCALING

② we don't use load-balancing in this

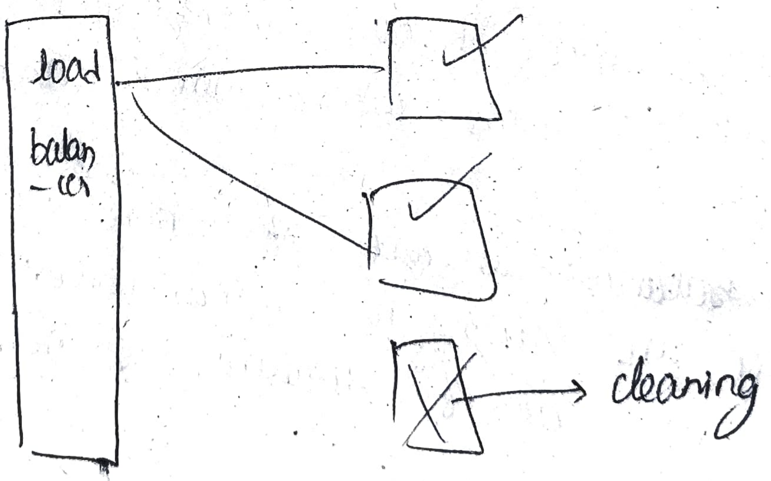
Advantages:-

→ fast

→ consistent

→ In real world we use the combination of both scaling types.

→ it is always recommended to follow the odd rule i.e, to have example :- 3 servers



→ so that if one of them went for cleaning / patching, remaining available servers will handle the burden

→ In this case, where as per our example of 3 servers,

~~B~~ 1 - went for patching
2nd & 3rd are taking the load now.

if one of these is broken,
at least you will have 1 - left as a back-up.

i.e., why odd ~~rules~~ rule works!

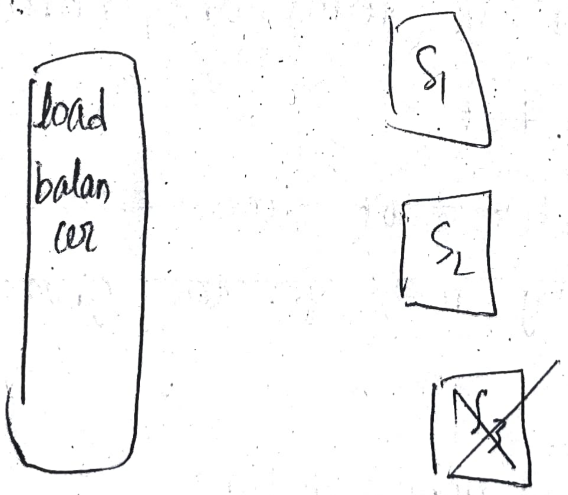
→ So, for production, you can go for 3 - server
~~otherwise if its dev, or stage you can go for 1 - server~~

∴ a horizontal scaling with a minimum of 3 is recommended

→ for dev you can use lower 3 configured servers
→ for prod you can use 3 high configured servers

Q) How, what happen in the patching scenario of a server?

Ans:-



→ initially these 3 servers were registered in the load balancer,

→ Now if you want one of them server to go for cleaning/patching,

→ we tell this to our load balancer.

→ "load balancer will first lets the server to full fill all its clients requests on the server and then puts it down for patching."

→ load balancer blocks all the in-gress policy for the server;

and after all the out-gress policy is finished then this server will go under patching.

→ now this server will be taken out of the rotation, i.e., we will not send any kind of traffic toward it.

Availability:-

↳ it is calculated in terms or %percentage.

~~availability is nothing but~~

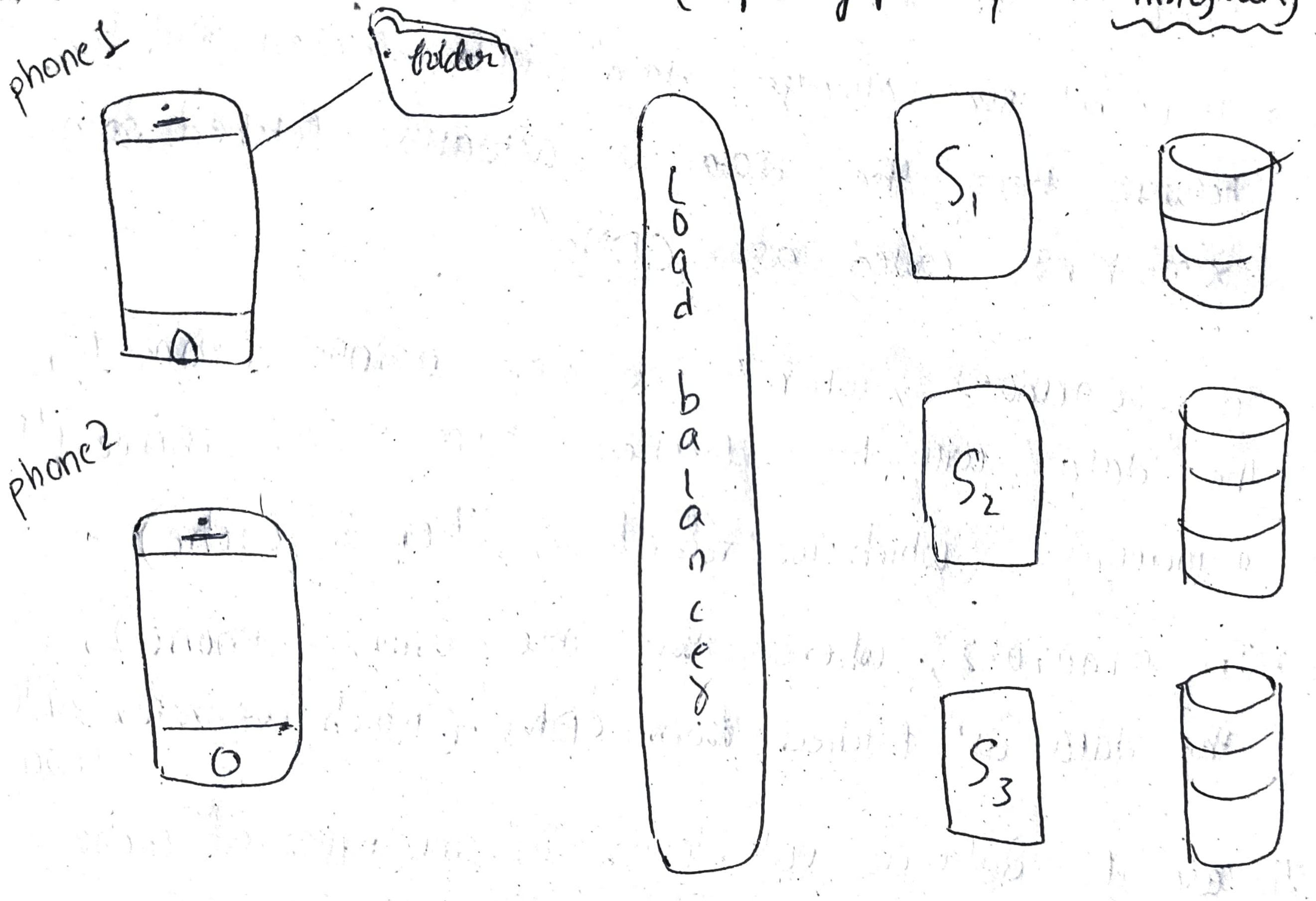
(*) ~~for a given interval of time, the no of request served by me, for the given no of request~~

(*) → for a given interval period of time, if i received "n" requests, how many requests was i succesfully able to serve.

Cache :-

→ for any request sent from client, the response that can be gotten faster when compared to the original source of truth (Databases, Servers) is cache

→ let's look at a scenario:- (updating profile pic in "instagram")



→ for every application you install the mobile creates its own [• Folder] for that particular application.

→ let's say you have updated your profile.

→ this profile's data, two ~~copies~~ copies will be there
  ↳ one copy will be stored in "DataBase"
  ↳ another copy will be stored in [• folder]

→ let's say that you are login in using another new mobile

→ this time you install instagram app. and again [.folder] will be created.

↳ now in this scenario ~~gener~~. when you look into "my profile", generally the data is fetched from Database but no

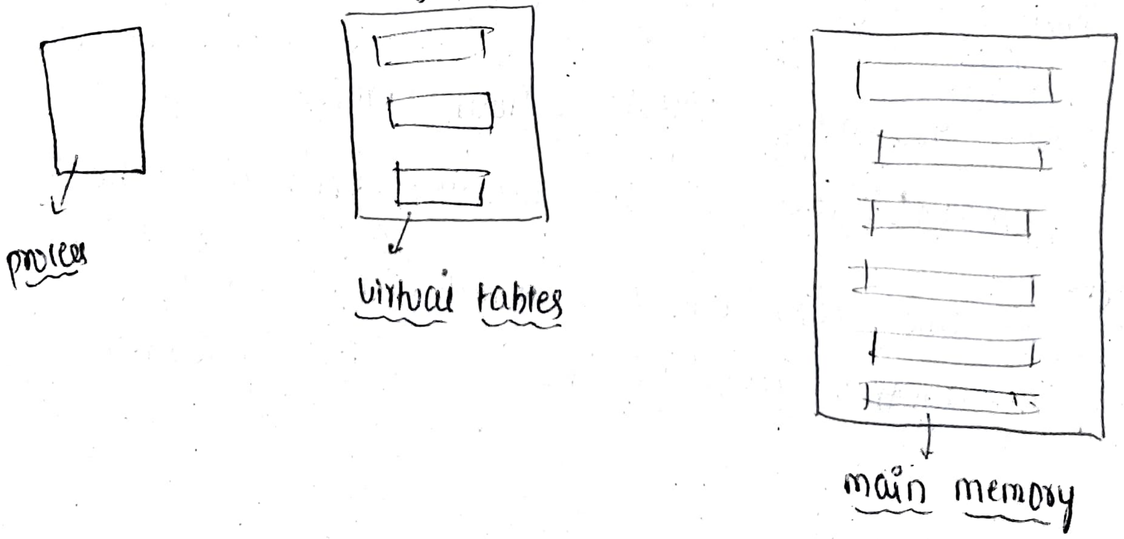
→ It is not like always data will be fetched from DB, because here the data is actually fetched from something called as "CDNs"

↳ In scenario-1, where you were using Phone-1, the data will be fetched from the Phone-1's in-memory (which we refer to as Tier-1 cache)

↳ In scenario-2, where you are using phone-2, the data is fetched from "CDNs" (which we refer as Tier-2 cache)

In case if data is not found in any type of cache then data is fetched from databases.

⊛ Virtual Memory :-



→ Example of main memory :- SSD, HDD, whatever...

→ ~~In~~ The data is present in Main Memory, whenever any data is frequently used, it will be stored in virtual tables,

→ virtual tables will have pointers of whatever data we need, so that we can fastly fetch our data.

→ if we do not find our data through virtual tables, then again data fetching will happen from main memory. (This is known as cache miss).

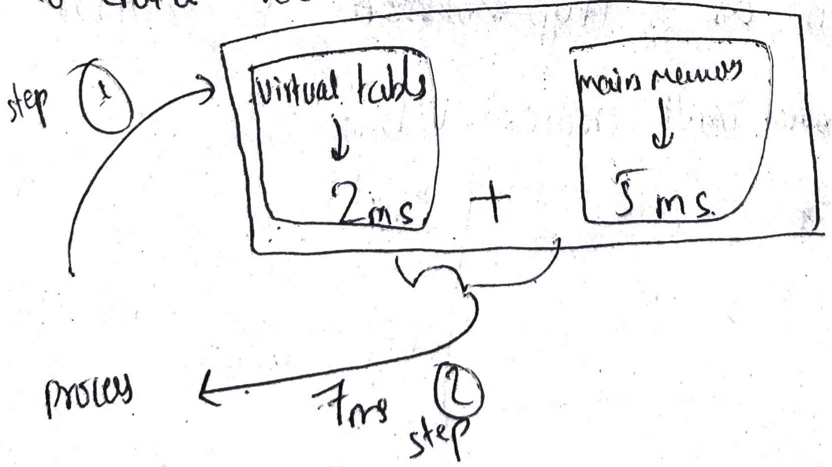
let's say for example, let's assume that

virtual tables take - 2ms time to give you your data,

mainmemory takes - 5 ms time to give you your data.

Now worst case scenario is, when you find nothing in virtual tables, ~~so now you~~ so data now will be fetched from main memory.

so worst case scenario of total time taken ~~is~~, when no data found in virtual tables (cahe miss) is :-



Storage :-

→ Types
① Files
② Blob
③ Queues
④ Tables

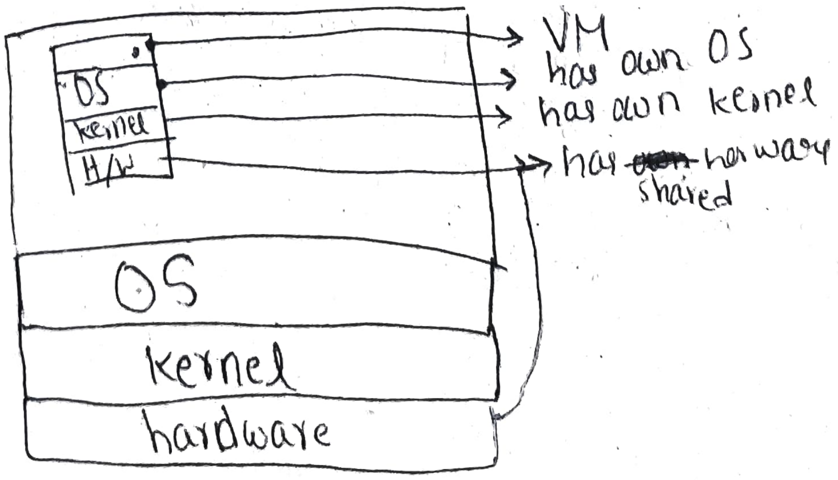
Databases :-

→ let's say that there is an entity which represents a real-life object, and
now if you are trying to store these entities you ~~will~~ might be storing this in different - different ways,
either in tables formatt (SQL)
either in documents formatt (NoSQL)
either in Graphical formatt (GraphQL)

VMs / Containers :-

⊛ VMs :-
→ To create a Virtual machine we need some extra hardware known as ~~[illegible]~~ "Hypervisor"

→ without hypervisor you can't create VMs.



→ To create VMs we have various software available in the market such as:-

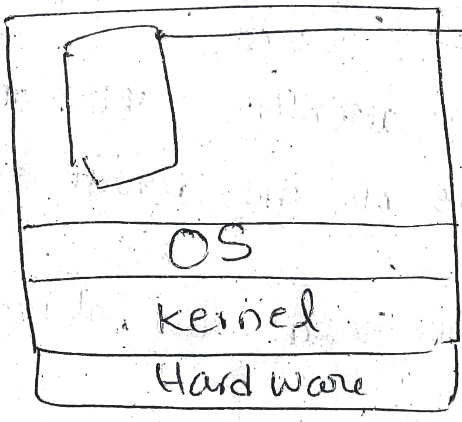
1) Oracle (virtual box)
2) VMware

→ virtual Machines, when they created will have own kernel

Note:- storage will be shared :- If your main system has 16GB ram we can say VMs to utilize 4GB out of it.

* Containers :-

Note :- Kernel for windows is diff,
kernel for linux is diff, ...
each OS will have diff kernel.



→ In containers "kernel" will be shared
∴ windows will have only windows containers
linux will have only linux containers

Topic / Queue

→ Queue follows FIFO property.