FILESYSTEMS:- DISTRIBUTED FILE SYSTEMS.

1. Requirement: Durable Data storage + High Availability.

3 File Systems I've picked up out of many.

(1) GFS. (2) Colossus. (3) Facebook Tectonic.

(1) GFS. (old)
* most powerful.
* seminal (original)

+ uses centralized Control plane (manager). simple.
- Over the years, GFs single Manager approached its scalability limits. However GFs cluster could survive.

So, they invented GFS- colossus.

(2) GFS Colossus:- Evolution from GFs to meet
(1) scalability beyond petabytes.
(2) enable more....

↓

example of evolution of system Design

Principle: "Make the system only as complex as needed for present."

(3) Fb's Hyperscale Tectonic System:-
+ Resembles colossus.
+ Achieving high scalability of resource efficiency.

GFS:- Google file System

Before GFS, there were single-node file system, network-attached storage, storage area networks.

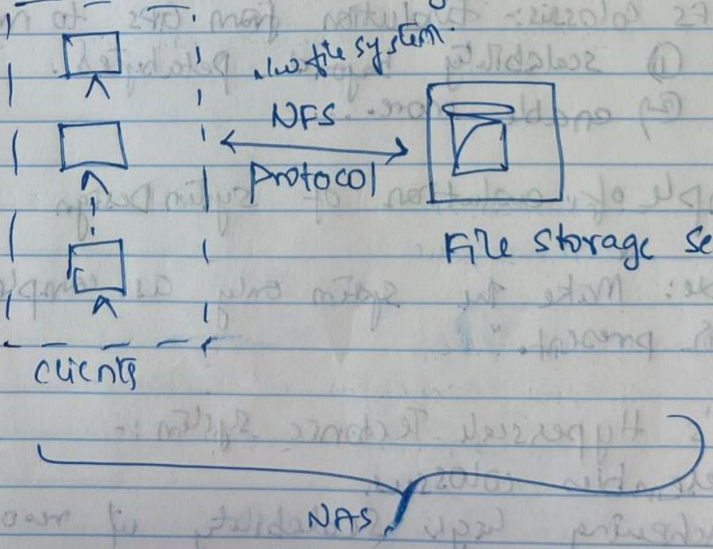
1. single-node file system:- runs on a single computer and manages storage on it.

Limitations: 1) storage capacity even if there is vertical scaling. 2) I/O operations on disk per second 3) power of processor 4) can become a single point of failure

Only advantage:- Low latency.

> But high throughput is more valuable than low latency.

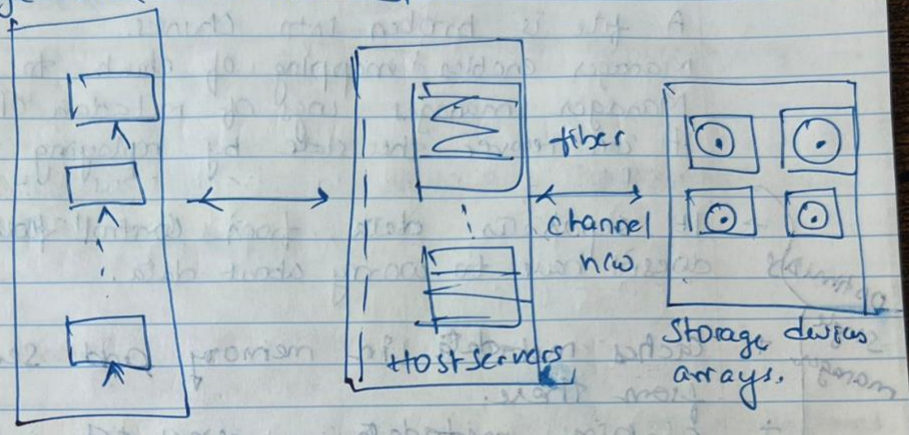
2. Network-attached storage (NAS)



Data stored over network into a file server.

Same limitations as single-node file system.

3. Storage Area Network: SAN.



Adv: Easy to scale by adding more storage devices

Disadv: Difficult to manage the second n/w - i.e Fiber channel.

- Difficult to troubleshoot any failure.
- Very costly.

4. GFS: + Scalability
+ Availability
+ Fault Tolerance
+ Durability
+ Easy managing it
+ performance optimization
+ Relaxed consistency model.

GFS Architecture :-

2 major components :- ① manager node
② Chunk servers.

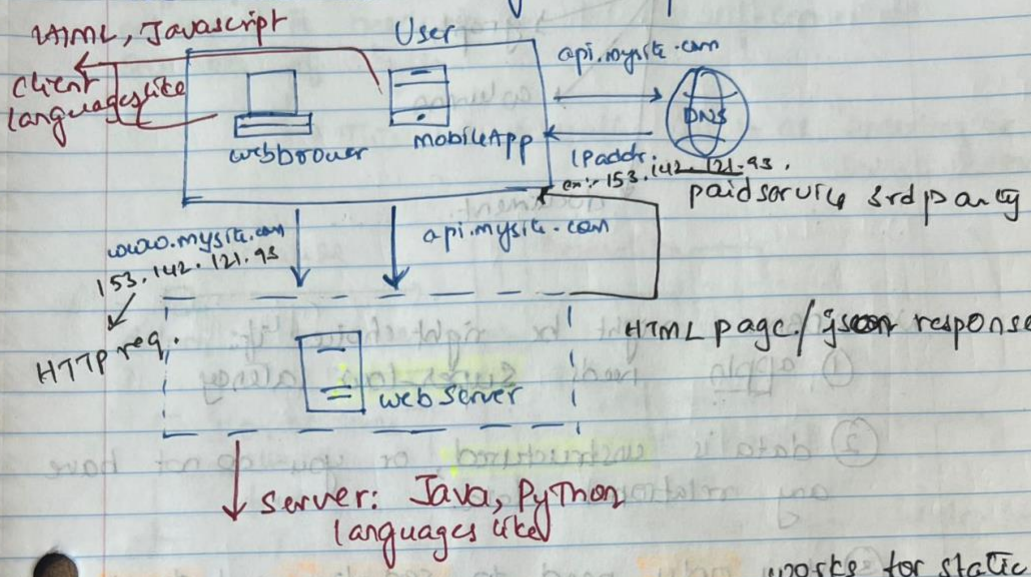
A file is broken into chunks.
Manager enables mapping of chunk to file.
Manager manages logs of metadata changes, so it can recover the state by replaying the log.

optimizations single manager
+ It separates data from control flow. So manager doesn't have to worry about data.
+ Caches metadata in memory and serving clients from there.
+ Caching metadata on client side.

Q. How GFS handles singe manager failure?
A: Can recover using operation log.
↓
this will be in persistent storage

Q. How is the time to recover low of operation log?
A: It checkpoints the manager state when it grows (log) beyond certain threshold size. Then during recovery it loads the latest checkpoint & replays the logs after that checkpoint

We could achieve by a simple architecture, however,



works for static pages

The above is the simplest approach. However, with the growth of user base, one server is not enough. So, we will separate DB from web server so that web/mobile traffic & DB can scale independently.

We have choices in which DB to use:-

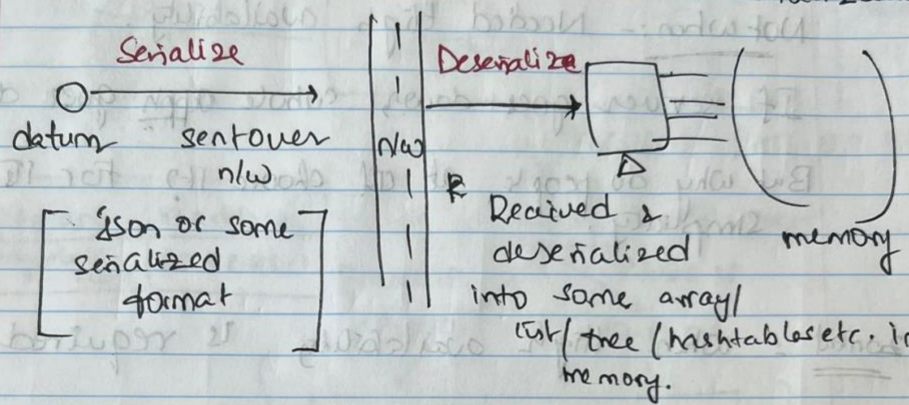
1) Traditional RDBMS 2) Non-R DBMS.

1) RDBMS: ex:- MySQL, Oracle DB, PostgreSQL
   → rows & columns.

2) Non-R DBMS: CouchDB, Neo4j, Cassandra, HBase, Amazon Dynamo DB, etc.

But if we're sending data to a file or over a network, it needs to be in a self-contained sequence of bytes.

↳ This we usually call it as "encoding or Serialization".



2 data Representations:-

① Serialized/ Deserialized ② in-memory form.

aka

Encoding / Decoding.

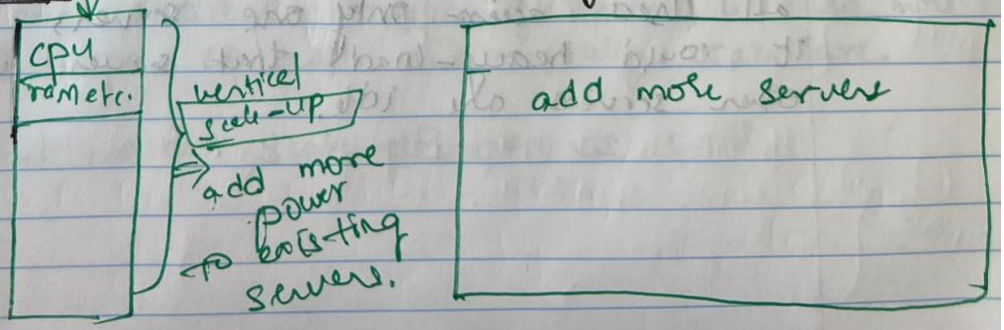
SCALING

SCALING

vertical = scale up.

horizontal = scale out

Vertical vs. Horizontal scaling.



* "Join" operations are generally not supported in non-RDBMS.

Non-RDBMS → keyvalue
→ graph
→ columns
→ document.

Non-RDBMS might be right choice if:

① appln needs super-low latency

② data is unstructured, or you do not have any relational data.

③ You only need to serialize and deserialize data (JSON, XML, YAML, etc.).

④ You need to store massive amount of data.

Above we mentioned "Serialization" & "Deserialization".

Serialization/ Deserialization: When we write data into memory, it is usually in some arrays, lists or hashtables etc.

In-memory, data is in arrays, lists or hashtables, trees etc. Why? = so that the data access is efficient & cpu can manipulate them.

When To Choose which scaling vertical vs. horizontal

Vertical Scaling = when traffic is low

Not when:- Needed High availability.

If server goes down, whole appn goes down.

But why do people at all choose it? For its simplicity.

Horizontal : when high availability is required.

Till now we studied: So, now design:- Have a web server. + dB | scale it as needed

Now, when many users across web simultaneously access the web server, it reaches the web server's load limit, then it might give slower response.

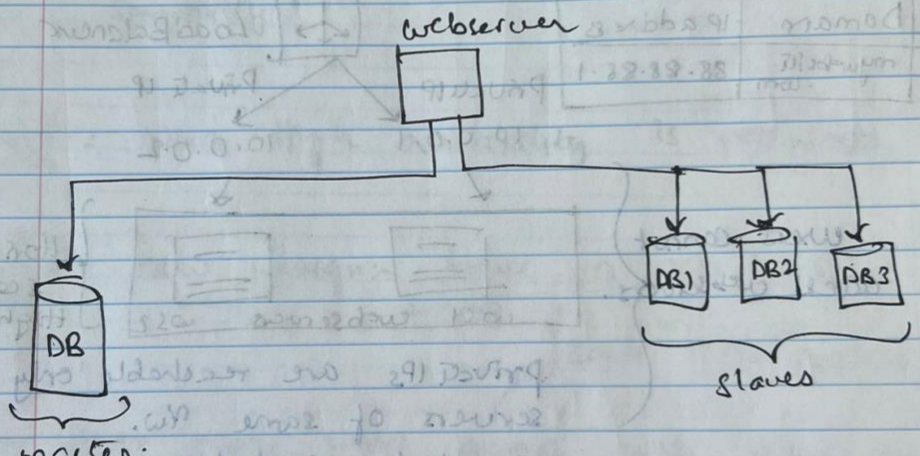
Even if ppl servers are there, what if all users access only one server, then it could heavy-load that server. while other servers are idle.

DATABASE REPLICATION :-

1. usually master/slave relationship.

Diff :- Master = only writes usually. = insert, update, delete.
Slave = usually only read operations. ↳ select.

∴ Most applns. have high read : write ratios, mple slaves are used.



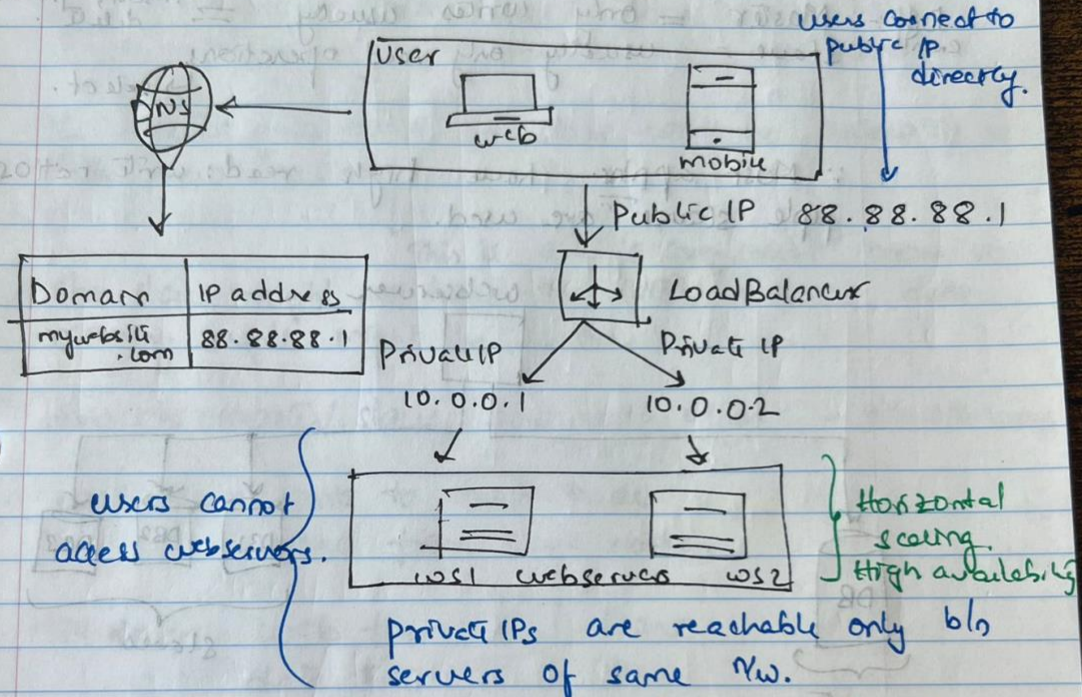
Advantage :- 1. all writes & updates happen in master node. All reads in slave nodes.

① Better performance ↳ This improves performance ∴ more queries in parallel.

② Reliability. If one DB is destroyed physically. There are other servers available. So no data loss

To avoid this problem, we use a Load Balancer.

↳ evenly divides the ~~workload~~ traffic among web servers.



Advantages:-

1) If one server (WS1) goes down, we have WS2 which is still up.
2) LB can distribute traffic. If needed, we add more servers. LB will automatically take care.

Now we can scale dB tier as well.

high availability

same as above.

If all slaves go down, master takes care of the role temporarily till new slave(s) are added. (performs)

If master goes down, a slave will be promoted as the new master.

↓ This is a little complicated process as the slave might not have the latest data. ∴ data-recovery scripts might be needed.

How do master/slave know each other? — Need to know.

> webservers write to master node.
> read from slave nodes.

Till Now, web tier ☑ Done
data tier ☑ Done

Now, let us improve load/response time.
For that we will add CACHE LAYER. &
shift static content (JS/CS/image/video files) to CDN: Content Delivery Network.

CACHE:

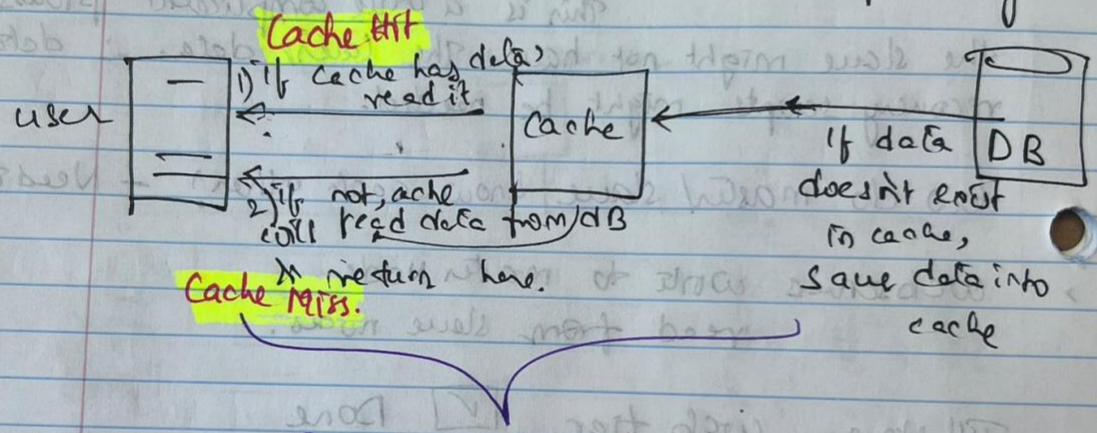
Temporary storage area.

Stores results of expensive responses or frequently accessed data.

Why? Better cache system performance,

ability to ↓ DB workloads,

ability to scale cache tier independently.



Read-through cache strategy.

Q what other cache strategies exist?

Interacting with cache servers is simple. ∴ most cache servers provide APIs.

"CACHES are meant for PERFORMANCE."

Cache is a temporary data storage that can serve data faster by keeping data entries in memory.

caches only store the most frequently accessed data.
↳ non persistent data - keeps repeatedly read & written data
⇓
To give lower latency.

DISTRIBUTED CACHE: multiple caches server - so it's scalable & high degree of availability.

CACHES :
1. minimize user-perceived latency by already calculating & storing the results
2. store user session temporarily,
3. serve temp data even if data store down for some time.
4. Reduce n/w costs by serving data from local resources.

Cache is then further performed at different layers

Layers

1. Web — HTTP cache headers, web accelerators, k-v data stores, CDNs etc.
2. Appln — Local cache & (k-v) data store
3. DB — DB cache, buffers, (k-v) data store

It can also be put in DNS x client side technologies like browsers or end-devices,

Design considerations:-

① Scaling: spof (single point of failure). to avoid

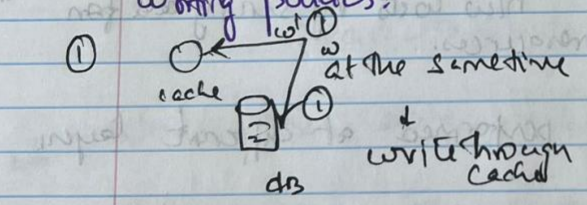
② If scaling main challenge is consistency of data in dB

③ Eviction policy: Remove unwanted data so the cache can hold the effective data

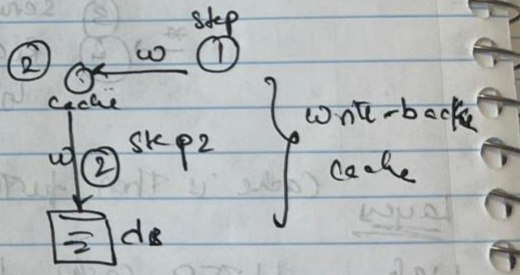
④ Expiration policy: If Data expiration is not set, it will store permanently. So, exp date should be set. Not too long or too short.

⑤ writing policies :- How to make cache have the data

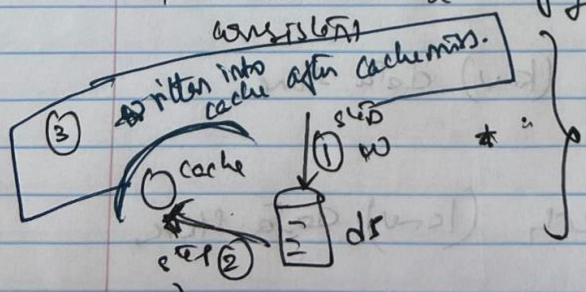
writing policies:-

① 

+ writing time is more
+ But data will be strongly consistent

② 

+ writing time is less.
+ But may have inconsistent data

③ 

a.k.a Read Through cache

+ Doesn't reflect recently updated data

ex:- (github, fb)

- Eviction policies:- LRU, MRU, LFU, FIFO, MFU.

> Data structures while designing cache:-

Think what it can be? we need O(1) read/write access

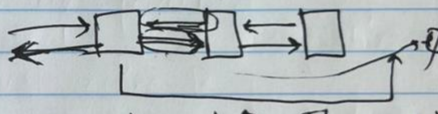
eviction policy ⇒ some order ⇒ LRU / LFU ← Doubly Linked List + Hash

Why Doubly Linked List (DLL) & not SLL?

Ans In cache, you might want to move an element to the front (or end) to indicate its most recent use.

With a DLL, we can just update the pointer

after putting at end. just say
head.prev = tail
head.next = null



just update the pointers.

Other wise in SLL, we would have to traverse all the way in the list to find the previous node - O(n)

Both SLL & DLL can be used.

↳ has adv: for constant - removal & insertion at both ends. O(1).

Overall: Cache Design Includes:-

① Hash Map ② DLL ③ Eviction Policy.

Both are open-source.

MemCache vs. Redis

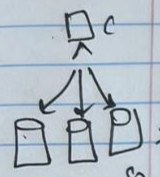
Both have client/server model

① MemCache :- (k, v) store.

Both "k" & "v" are strings. So, MemCache cannot manipulate it. This also means any data that is to be stored has to be serialized.

client/server :- ½ logic done by client

& Other ½ - by servers.



Each server follows "shared-nothing" architecture.

∴ O(1) query speed. ↑ throughput

↓ low latency

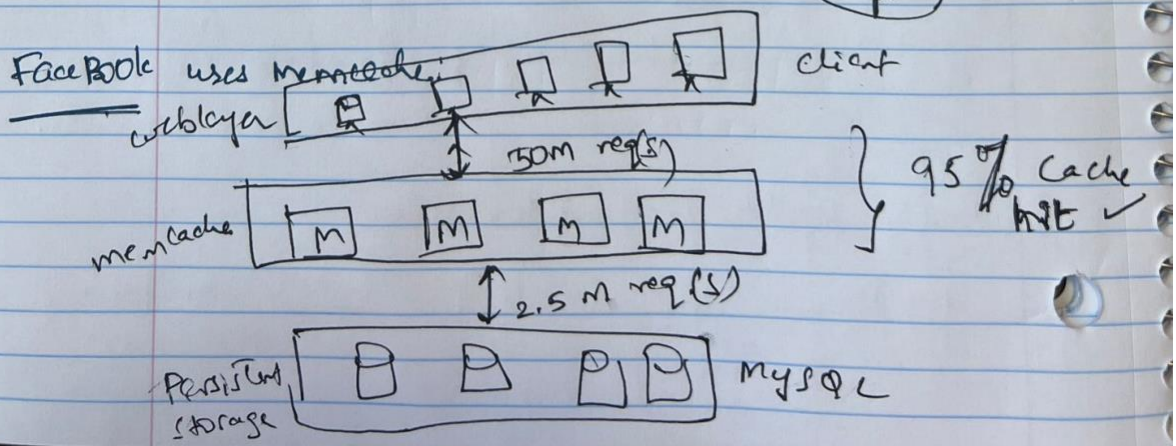
✓ good for horizontal scaling

ex: code in memcache: > cache.set("mykey", "Hello", 3600* seconds)

> cache.get("mykey")

resp.

FaceBook uses memcache:



Redis: Both Redis & Memcache belong to NoSQL family → Cache / database(k,v) / message broker

→ Can modify data w/ wasting n/w bandwidth. doesn't need serialization/deserialization

→ stores data in (data structures)

in-memory storage → No common b/n cache servers.

Memcache is preferred for smaller, simple read read-heavy systems. where as Redis is useful for both read & write-heavy systems.

good for streaming videos as well.

Memcache	vs.	Redis
(data stored as strings only)		(data stored in data structures)
+ can't modify data ∴ n/w bandwidth wasted in uploading & downloading. wastes		+ can modify data ∴ n/w BW not wasted
+ saves time & effort in serialize/deserialize.		+ saves time & effort by avoiding serialize/deserialize.
+ servers do not know each other.		

CDN: CONTENT DELIVERY NETWORK

CDN: is a n/w of geographically dispersed servers to deliver static content. like images, videos, CSS, JS files etc

cache of pages → static (images, videos, CSS, JS)

→ Dynamic (based on req. params, we don't cache query strings etc)

CDNs are caches of static content of web pages.

"Globally dispersed servers."

⊖ The closer the CDN server is to the user, the faster the response is.

⊕ The further away slower response

Amazon provides CDN. Akamai also provides CDN.

Adv + Static assets (JS, CSS, images etc) are fetched from CDN

+ Hence DB is lightened the load.

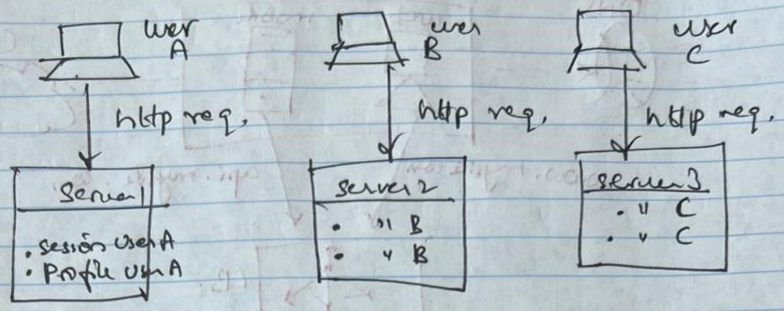
STATEFUL VS. STATELESS:- State ex:- user session data

① Stateful server: remembers client data (state) from one req. to other.

Stateless: keeps no. info. It is usually stored in persistent storage ∋ RDBMS or NoSQL

Q. Who does it refer to remember / not remember?
Ans: Server.

ex:- Stateful :-

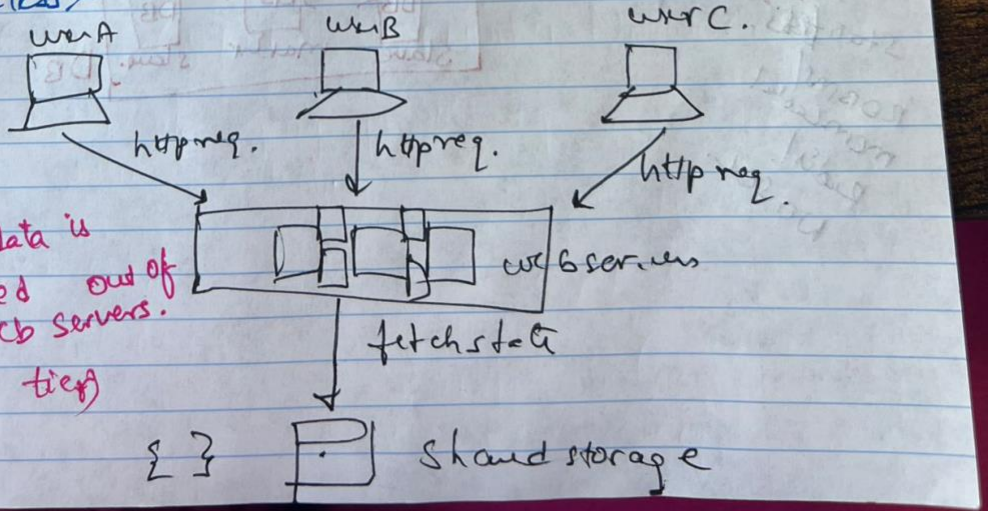


① Problem is :- If UserB by chance (Via LB) sends req. to Server1, it doesn't contain the data. Then either it fails or needs to be rerouted to server2.
Overhead.

② What if all requests come from UserB? Then only Server2 is kept on working, while rest are idle.

∴ Stateless.

Stateless :-



State data is moved out of web servers.
(web tier)

All we learnt till now.