System Design Basics

-> Domain Name → Human Readable form of Ip address

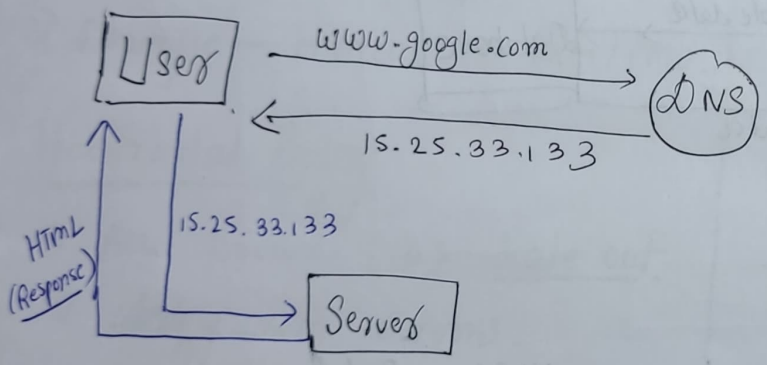
-> Domain Name → www.leetcode.come

-> Ip address → 15.13.125.....

-> Servers only understand Ip addresses they dont understand Domain Names.

-> Conversion of Domain Name → Ip addresses is done by Domain Name System (DNS) → phonebook of internet.

Single Server Setup → Only 1 server caters the request of users



-> Traffic → denotes the number of requests to a server.

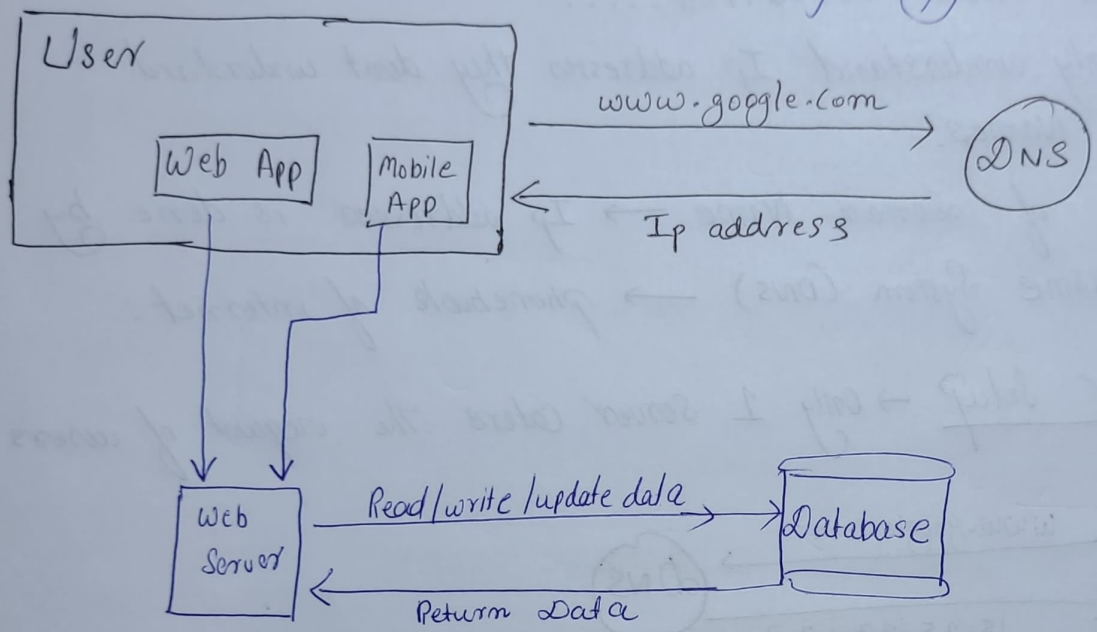
↳ Web applications → { Server Side languages + Client Side languages }

↳ Mobile apps → Communicates with server through HTTP protocol. Response recieved in API JSON format.

② With growth of users we need more servers to cater the request of many users.

Database and Multiple Servers

- Maintain 1 server for handling the traffic
- Maintain another server for storing the database
- So that we can scale each server independently.



Which database to Use?

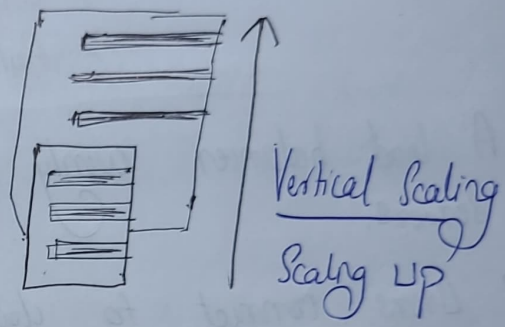
Relational Databases (RDBMS)	NOSQL Databases
Represent data and store it in tabular form.	Grouped in 4 categories –
Can perform JOIN operations.	Graph Stores, Key-value Stores
Eg:- MySql, Oracle, Postgresql	Column Stores, Document Stores.
Used by most developers.	Join operations not supported.
	Cassandra, Amazon DynamoDB

When to use NoSQL databases?

- Your application requires super low latency.
- When your data is unstructured
- When you want to store massive amount of data.
- You only need to serialize and deserialize the data for eg- (JSON, XML, YAML etc) → No schema needed.

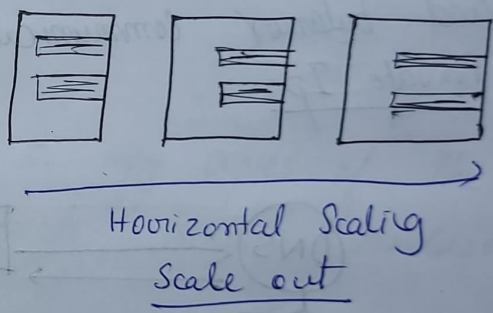
Vertical Scaling

- Also known as Scale up.
- Adding more power to your servers
- Example – increase → Ram / CPU.



Horizontal Scaling

- Also known as Scale out.
- Adding more servers.



- When to use Vertical & Horizontal Scaling

Vertical Scaling	Horizontal Scaling
- When traffic is low	- When traffic is very high
- Simplicity is its advantage	- For large scale applications
	- Complex to implement & costly.

Limitations of Vertical Scaling

- Impossible to add unlimited CPU / Ram to a single server.
- No backup if single server goes down.

When to use NoSQL databases?

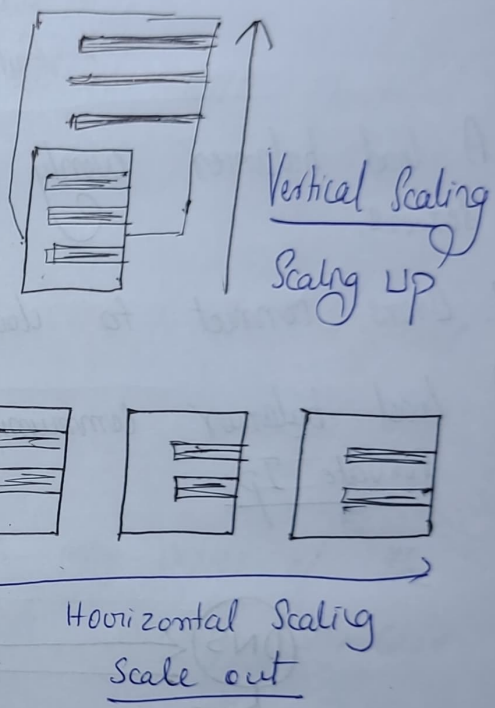
- Your application requires super low latency.
- When your data is unstructured
- When you want to store massive amount of data.
- You only need to serialize and deserialize the data for eg- (JSON, XML, YAML etc) → No schema needed.

Vertical Scaling

- Also known as Scale up.
- Adding more power to your servers
- Example – increase → Ram / CPU.

Horizontal Scaling

- Also known as Scale out.
- Adding more servers.



- When to use Vertical & Horizontal Scaling

Vertical Scaling	Horizontal Scaling
- When traffic is low	- When traffic is very high
- Simplicity is its advantage	- For large scale applications
	- Complex to implement & costly.

Limitations of Vertical Scaling

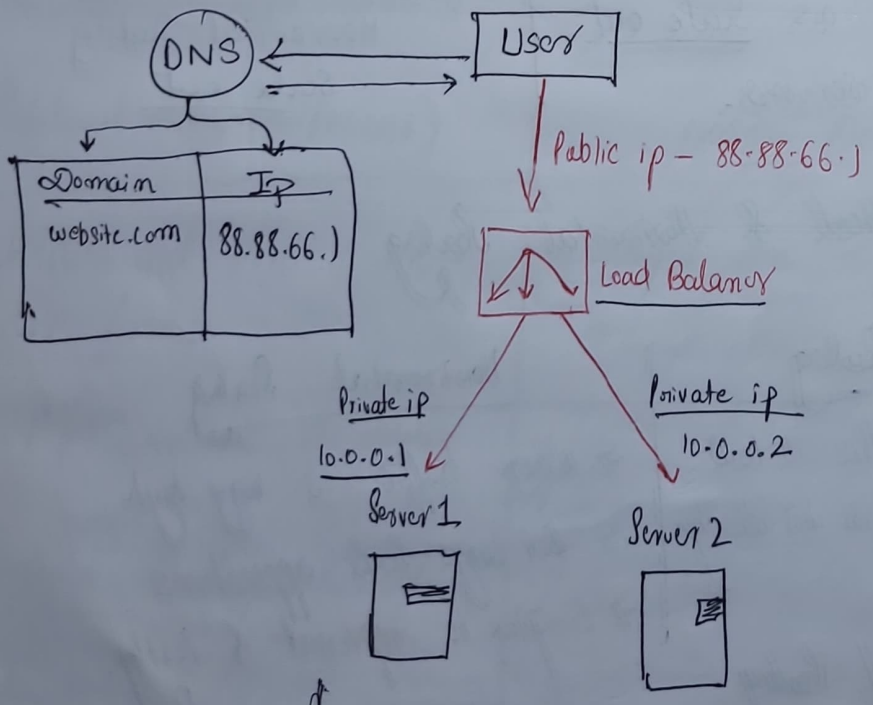
- Impossible to add unlimited CPU / Ram to a single server.
- No backup if single server goes down.

Load Balancer

- In our previous design users connected directly to the server.
- When server is offline what will happen then?
- Also when many users access at the same time :-

Problems ⟶
- Web server load limit.
- Slow response
- Fail to connect to server.

- A load balancer evenly distributes web traffic to the multiple servers.
- Users connect to load balancer via the public Ip.
- Load balancer communicates with web servers via the Private Ip.



- Users access via the public Ip, they don't have access to private ip.

(x) If Server 1 goes offline, Server-2 will get all traffic so that service will not go down.

(x) We can add a new web server afterwards to balance the load.

(x) If website grows rapidly 2 servers are not enough to handle traffic

(x) Add one more server and load balancer will start sending traffic to that server.

(x) Still one Issue — Database is only one, what happens if the database is gone ??

Database Replication → Maintain a copy copies of the databases to provide backup in case main database goes down.

Master - Slave Architecture

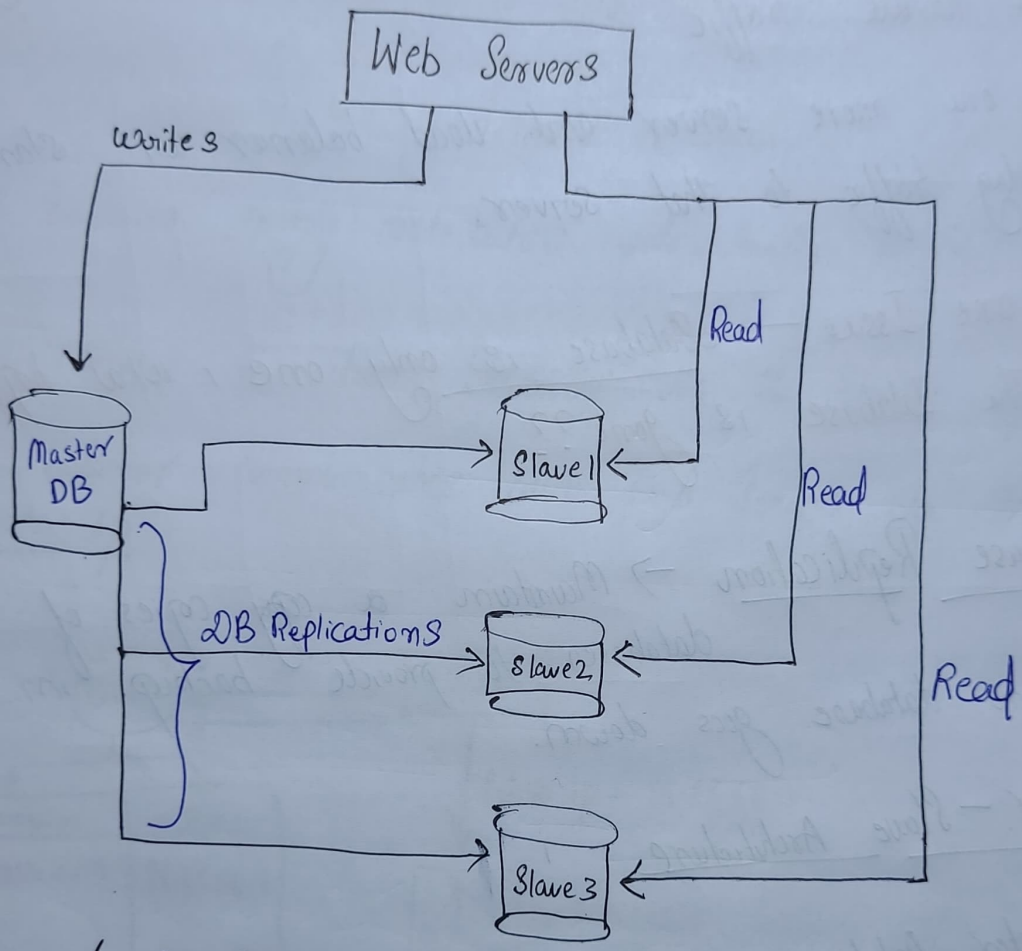
Master Database → Original database

Slave Databases → Copy / Replic databases.

Master → Only write operations :) Insert / update / delete operations.

Slave → Only Read operations :) Gets copy of data from master.
:) Only Read

* Most applications performs read operations only so than slave database server those requests. So [Slaves > Masters]
* All the updates made in master database are copied in slave databases but not immediately.

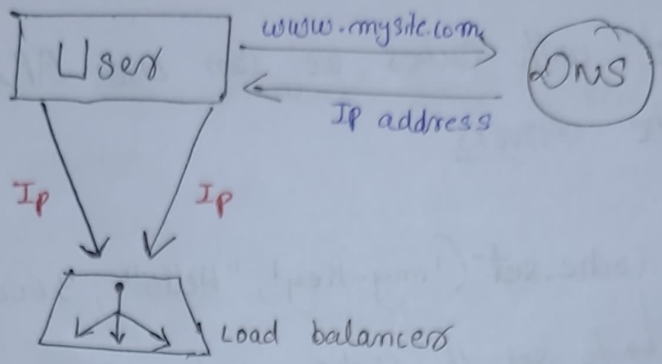


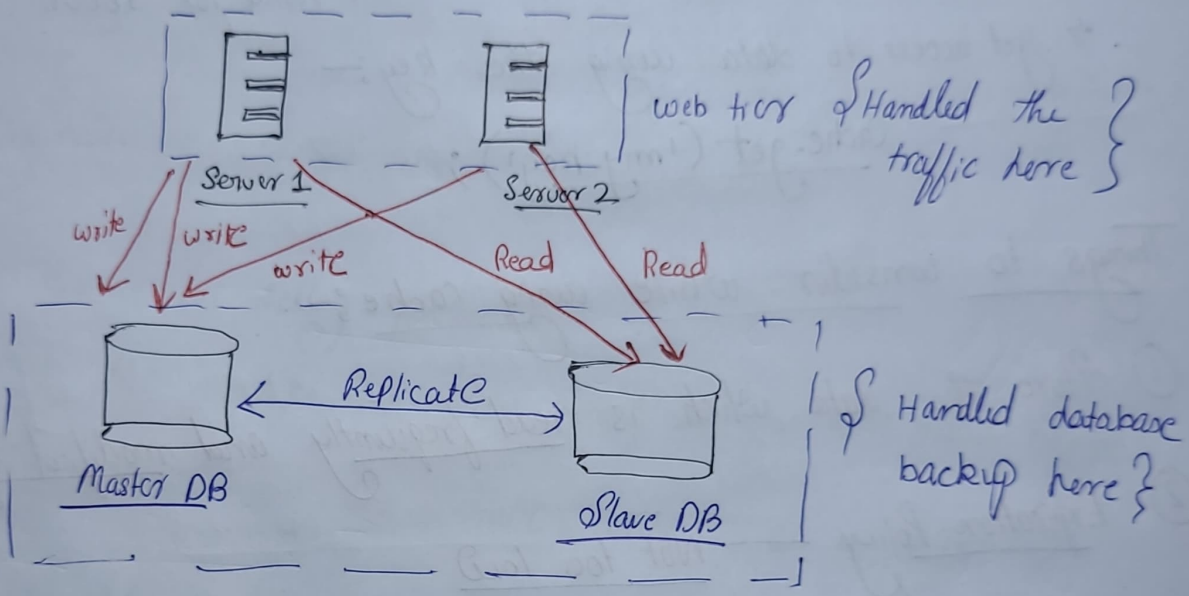
Advantages of DB Replication

① Better performance ② Reliability ③ High Availability
④ Parallel query executions.

* When any Server (slave) is down we redirect request to another server.
* Data will be never lost as we have multiple copies.
@ If master DB goes down, promote any slave to master but this is very complex

Now our system looks like this

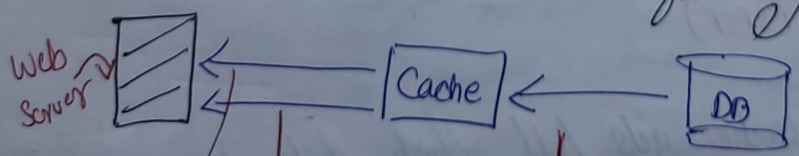




Caching → Used to improve the response time

- Temporary Storage
- Stores → Result of expensive responses
  - Frequently accessed data.

- We don't need to reach database for every request if we use cache.



↳ If data, don't exists in cache Save data to cache

Return data to web server

↳ If data exists in cache read data from cache.

Interacting with Cache

- To interact with caches we can use APIs provided by the cache servers

Example :-

```
Cache.set('my-key', "Hello", 3600 sec);
```

↳ to set the cache, cache will reside for 3600s.

- get access to data using the key :-

```
Cache.get('my-key');
```

Things to consider while using cache :-

1. Store the data which is read frequently and modified infrequently
2. Expiration Policy — Not too low
   or we will have to reload data from DB.
   ↳ Not too high :- Stale data
   ↳ old data
3. Consistency between DB & cache should be maintained.
4. Multiple Cache Servers across different data centres to avoid Single Point of Failure (SPOF)
5. Eviction Policy → When cache gets full which data to replace??
   - → LRU :- Least Recently Used.
   - → LFU :- Least Frequently Used.
   - → FIFO :- First in First Out.

Content Delivery Network (CDN)

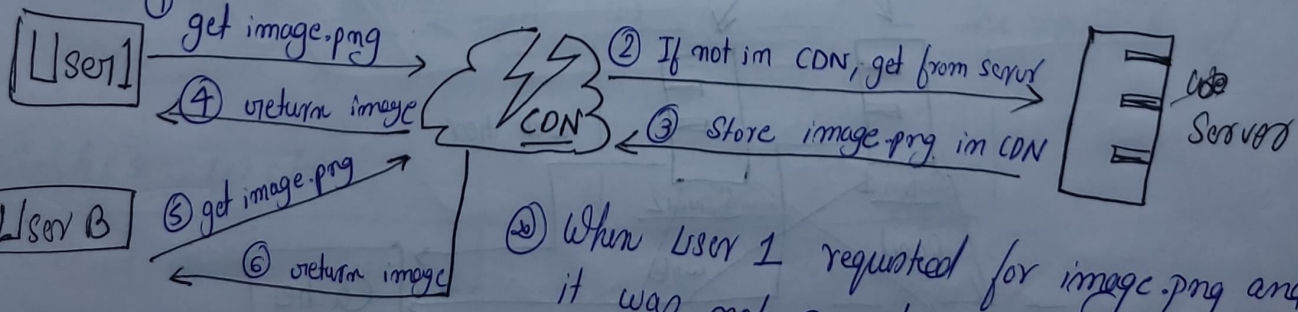
- To understand CDN we must know about static and dynamic content.
- Any website has two type of content :-
  1. Static Contents (Image, Videos, CSS, JS - logic)
  2. Dynamic Contents (changes according to user input)

Example - Website for selling ebooks -
- All the ebooks are static content as they will never change.

CDN ⟶ Stores the static contents of a site.

How CDN works ??

1. User visits a website
2. CDN Server closest to the user (geographically) Serves the static content.
3. The more further the CDN server is, the website will load more slowly.

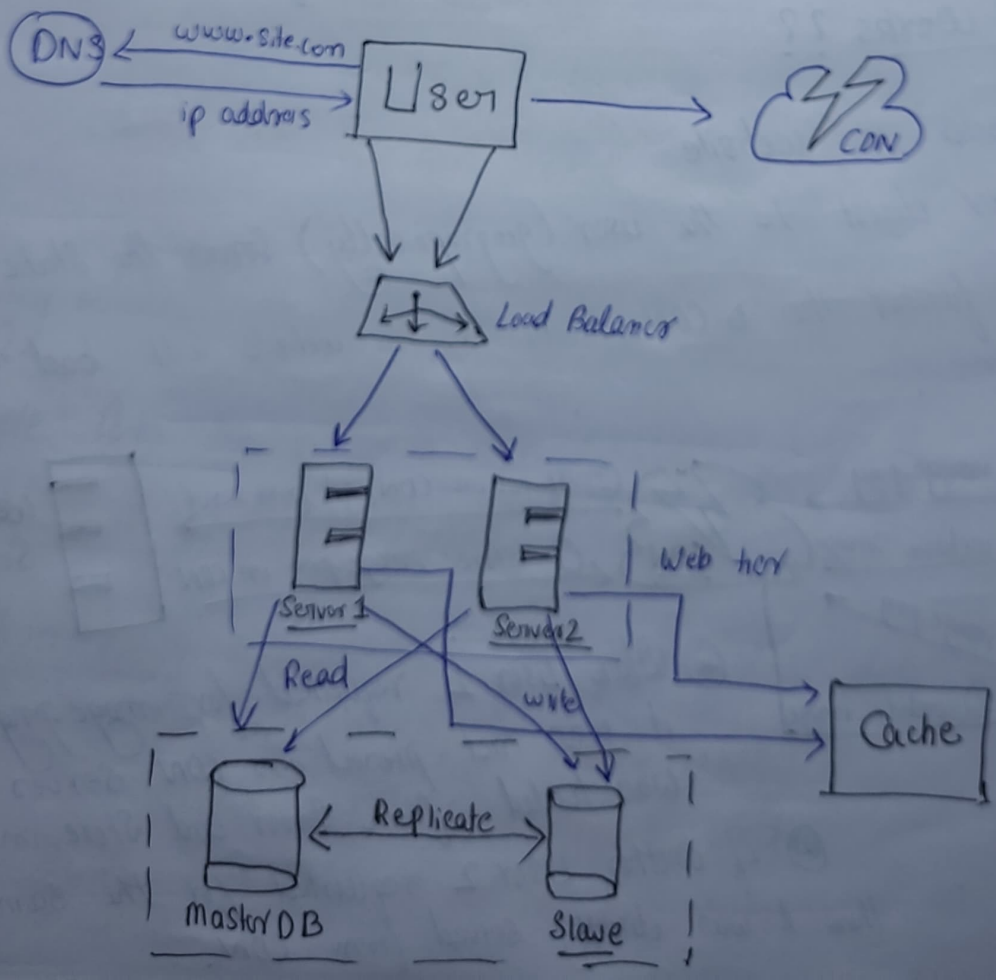


5. When User 1 requested for image.png and it was not present in CDN servers it was loaded from server and store in CDN
6. If another User 2 requested for the same data then it was directly served from CDN.

Things to Consider for using CDN :-

- **Cost** → Get rid of infrequent data to reduce costs.
- **Expiry time** → Should be moderate not too low not too high
- **CDN Fallback / Backup** :- Mechanism to fetch data directly from servers if CDN server fails.
- **Invalidating files** → ① APIs are provided by CDN vendors to invalidate a file which is no longer needed.
  ② **Versioning** — mention version names to invalidate previous versions. [img.png?V=2]

Now our System looks like this

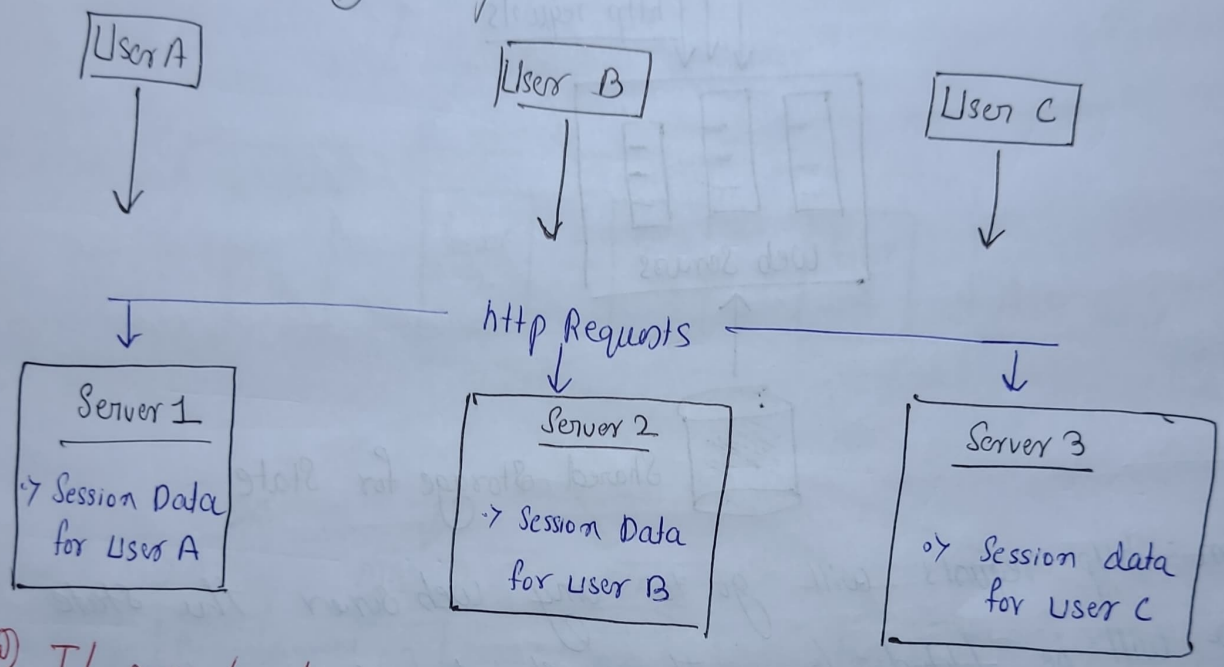


Stateless & Stateful Web architecture

(W) **Session Data / State** : Data that represents User's session on web.

- Why State is needed → Identify a user.
  → Authenticate a user.

- Consider the following example



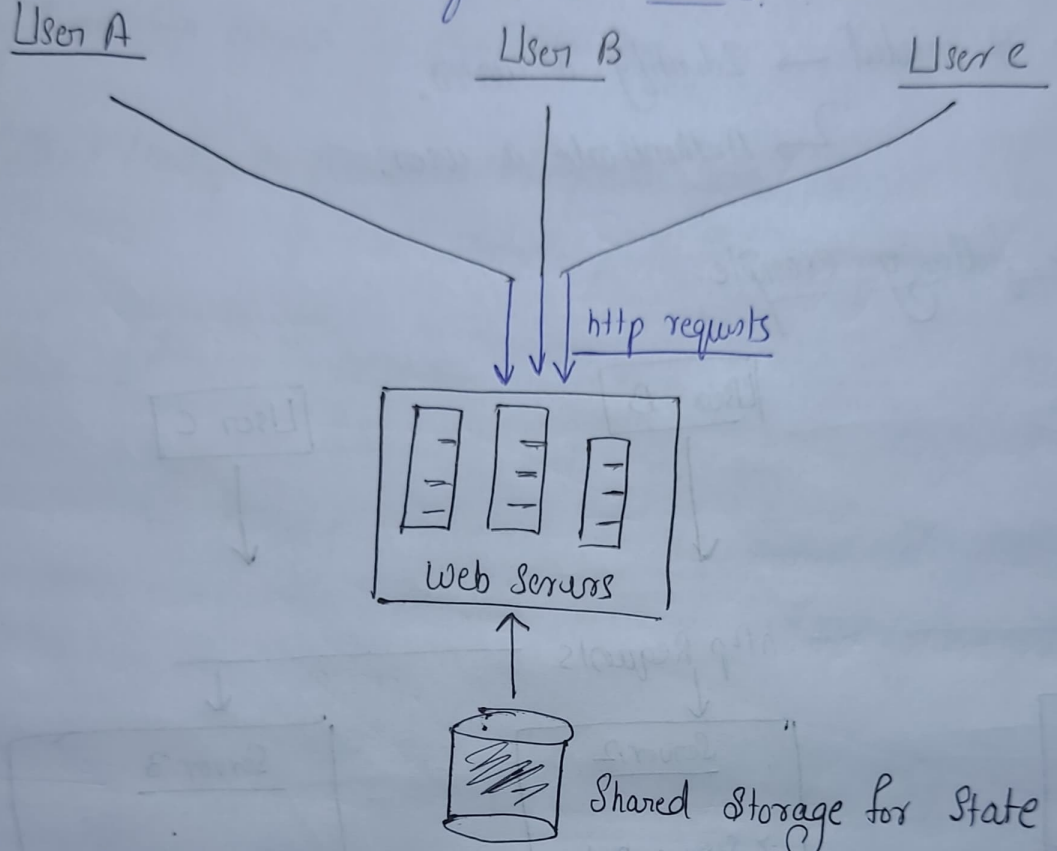
(W) If request of User (B) goes to Server 1 or request of user (A) goes to Server 3 it will not be Server or allowed as the session data won't match

- So this is a overhead for us to redirect the request of specific users to specific Servers only.

Stateless Web	Stateful Web
Does not remember client data	Remembers client data
Does not store state information	Stores state information
Simple & Robust	Overhead, not Simple.

Stateless architecture

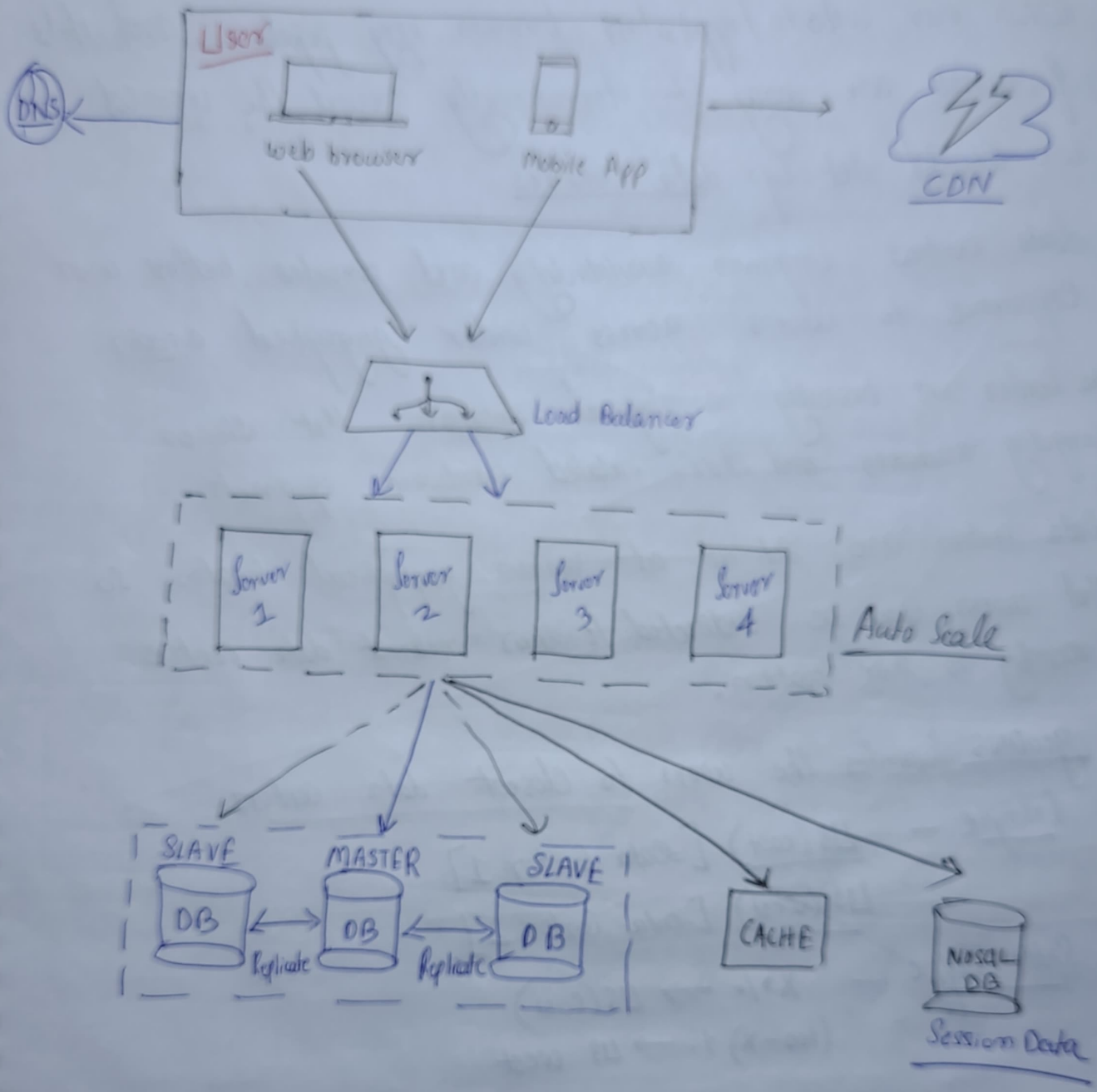
- Maintain a specific storage to store state data only.
- Store state data out of web servers.



→ When http requests will go to any web server the state data will be fetched from the shared storage to validate the users.

→ Since we are storing the state data separately we can auto scale our web servers.

Now our system looks like this



Data Centers

- When our website / application becomes very popular and lots of users are using it internationally around the world we should set up data centres
- Data centres improves availability and provides better user experience to users across wider geographical areas

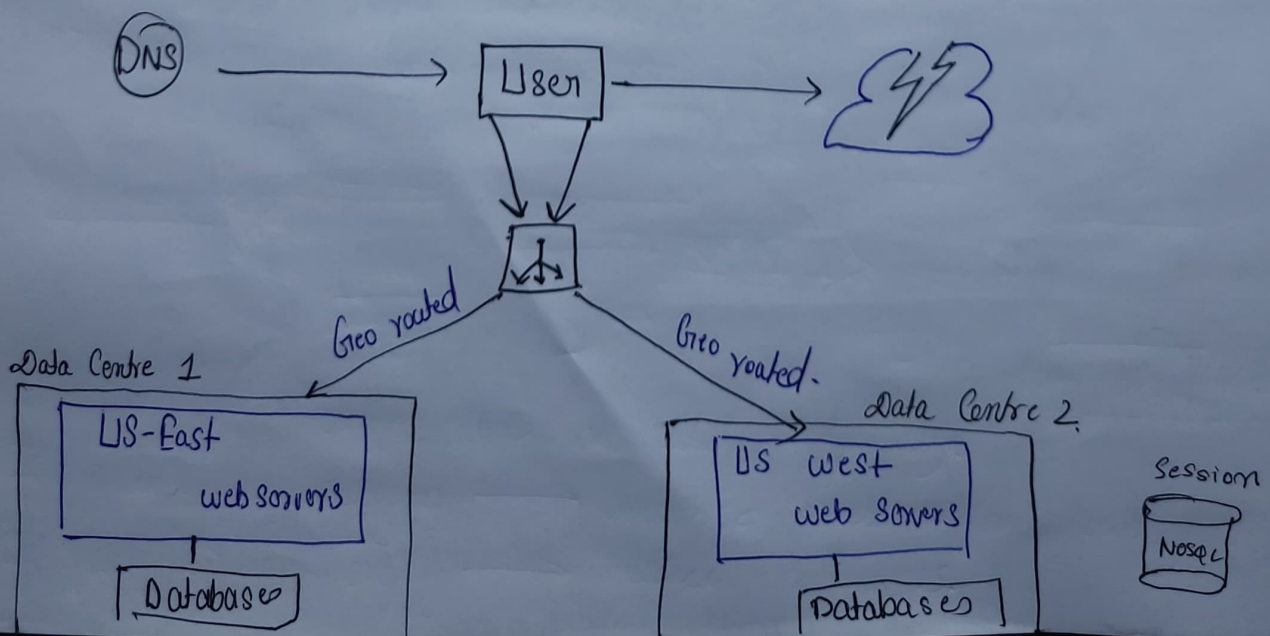
Data Center is basically a physical location that stores computing machines and their related hardware equipments.

- Data centres are set up at various geographical locations so that users can be redirected to their nearest data centre according to their location.

① geoDNS routes the users to closest data centers.

Example — US (east) [Data Center 1]
US (west) [Data Center 2]

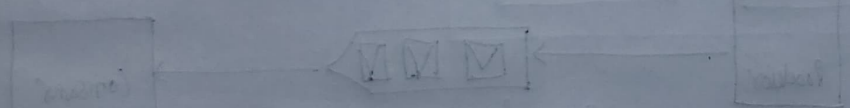
Split Traffic — X % → US (East)
(100 − X) % → US west



- The geoDNS system will accept the domain name from the user and return an ip address based on the user's location.
  - So if a user from US-west request to geoDNS it will get the Ip address of US-west servers.
- If any of the data centre is down then 100% traffic will be redirected to the working data centre by the geoDNS.

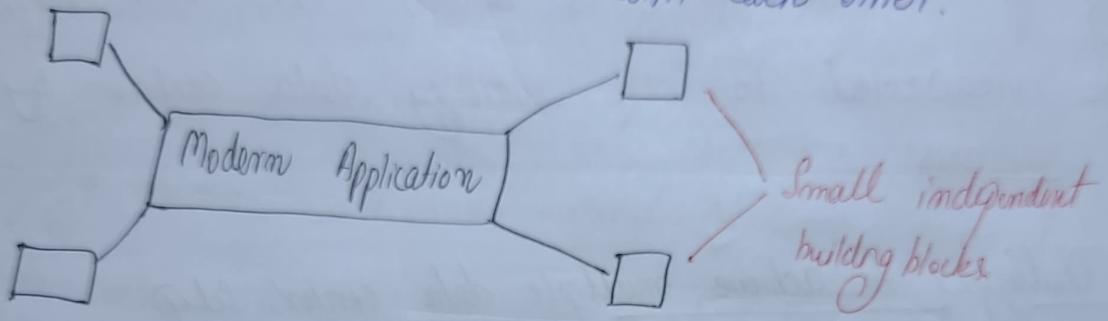
Technical Challenges to achieve multiple data center setup

- Traffic Redirection → Can be done by GeoDNS.
- Data Synchronization → Replicate data across multiple data centers so that any data center can serve requests in case any one goes down.
- Test & Deployment - Test your website/application at diffent locations.

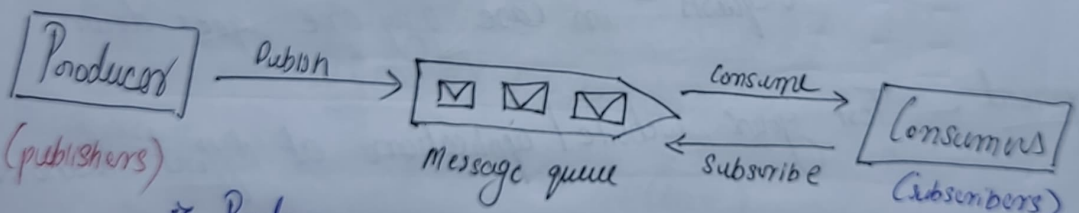


Message Queues

- When our application grows big in size, we can divide the various functionalities of the application into smaller chunks.
- Message queue provides a way through which these smaller chunks can communicate with each other.

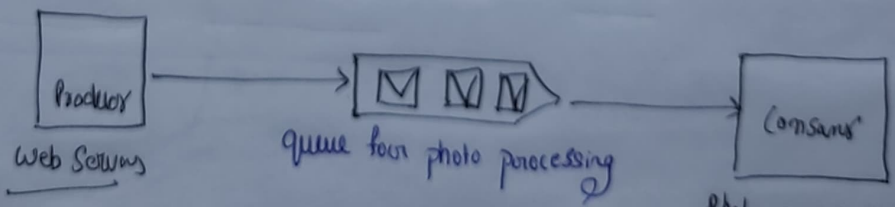


- Message queues provides communication & coordination between these smaller independent building blocks
- Message queue supports asynchronous communication.



- Producers can produce a message to a queue even when consumer is unavailable.
- Consumers can read message from queue even when producer is not available.

Eg- Photo processing app



- Producers and consumers can be scaled independtly.
- If queue size is large ⟶ increase consumers for processing.
- If queue is mostly empty ⟶ Reduce consumers.

Logging, Metric & Automation

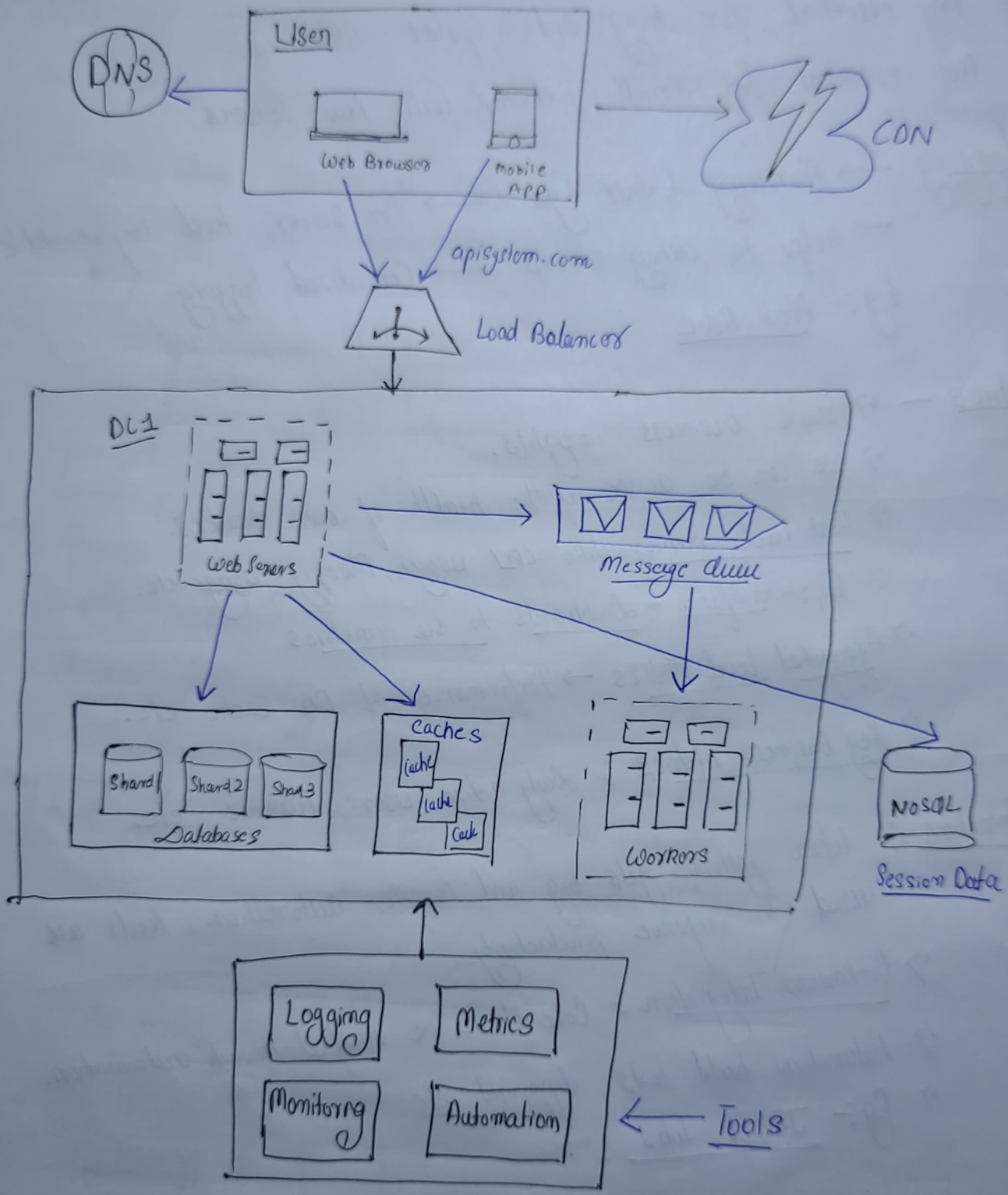
- Very essential for busy and popular websites.
- Not essential for small websites with few servers.

Logging → Monitoring Error Logs → Per server level implementable
→ Helps to debug errors → Centralized logging.
Eg:- New Relic

Metrics —
- Gain business insights.
- We can be aware of the health of our services.
- Host level metrics like CPU usage, Memory usage etc.
  Eg:- Grafana - Dashboards to see metrics
- Aggregated Level Matrics → Performance of DB, Cache etc.
- Key Business Metrics :- Daily active users, Revenue etc.

Automation :- When system gets big and complex automation tools are used to improve productivity.
- Continuous Integration :- Code check in verified through automation.
- Automations build tests, deployment etc.
- Eg:- Jenkin Jobs

Now Ours System Looks like this



Database Scaling

Vertical Scaling
① More power to current systems (CPU, RAM, Disk)
② RDS (Amazon Relational DB Service)
→ 24 TB of RAM

⁃ 2013 → Stackoverflow had 10 million unique visitors everyday with only one master DB.

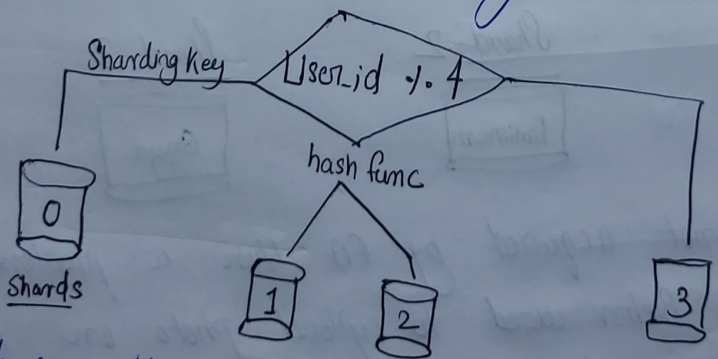
Drawbacks
(i) Hardware limits.
(ii) Single Point of failure.
(iii) Very costly

Horizontal Scaling → Create multiple copies of the server and redirect the users equally

① **Sharding** → method for distributing data across multiple machines.

Large DB → Shard 1, Shard 2, Shard 3, Shard 4

⁃ Hash function determines data will go to which Shard.



⁃ Carefully choose the sharding key so that data is distributed evenly.

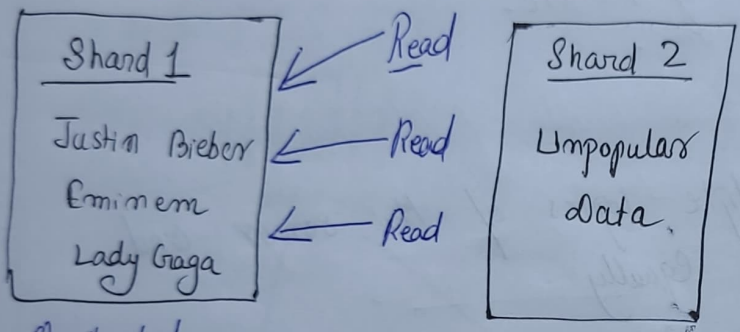
⁃ Horizontal partitioning of data is called Sharding.

Drawbacks of sharding

⑥ Resharding ⟶ When a shard can no longer hold more data.
- Shard exhaustion due to uneven distribution.
- It will require updating shard functions and the move the data around.

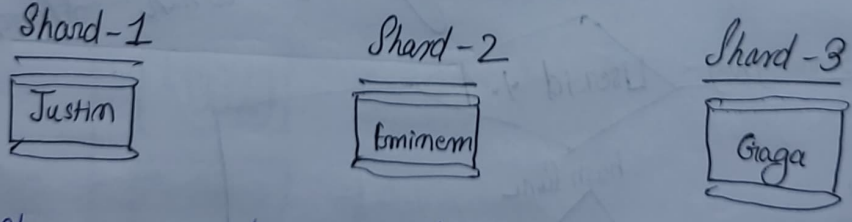
Celebrity Problem (Hotspot Problem)

⑦ Excessive access to a particular shard can cause shard overload.



Overloaded
- When certain data is stored in a shard which is accessed more frequently than the particular shard will be overloaded by the requests.

Solution ⟶ Assign one celebrity to Each Shard :-



- When Insta was not acquired by FB then a problem was faced. Whenever Justin Bieber used to upload photo on insta then insta used to go slow down as huge number of read and write operations like — likes & comments were performed.
- FB found a Solution ⟶ They started storing the data of likes for a post in a separate server.

(x) Sharding makes it difficult to perform Join operations

Common fix :- Denormalize the database, Store all of the data in a single ~~database~~ table so that queries can be done in a single table.

Normalization & Denormalization in RDBMS

Normalization :- Breaking down large tables into smaller related tables and establishing relationships between them.

- Minimizes redundancy and simplify the system.
- Dont overnormalize the data.

Example

Id	Name	Department	Location
1	Alice	HR	A
2	Bob	IT	B
3	Carol	HR	A

Redundant data

Normalize the table

Employee Table

1	Alice	HR
2	Bob	IT
3	Carol	HR

Department Table

Department	Location
HR	A
IT	B

• Find the department location of employee Id 2?

① Select E.employeeId, E.EmployeeName, D.DepartmentLocation
from Employee E Join Department D on
E.department = D.Department.

• When we normalize the data the operation will become a little bit complex to fetch the required data.

Denormalization

• Deliberately introducing redundancy into a database table
• Opposite of normalization.

Why denormalization ??

① Improves query performance as no joins are required.
② Simplified query
③ Reduce Joins
④ Best for Read intensive scenarios.
↳ Scenarios when read operations occurs frequently.

Disadvantages of Denormalization :-

① Redundancy → memory waste.
② Inconsistent data — as updates must be applied at multiple records what if we forget to update data at all places
③ Extremely slow update and write operations → As we will have to update in multiple places.