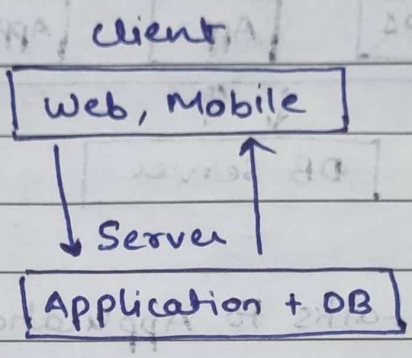
Date: __/__/____

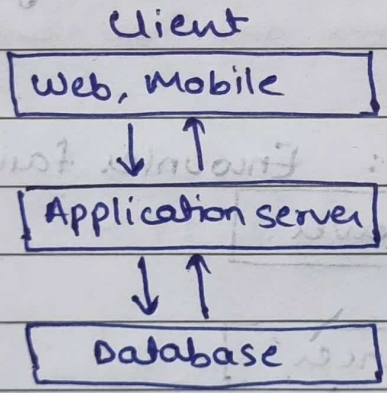
Scale from ZERO to MILLION Users in Detailed:

1) Single Server: When we have zero users, the we only need 1 server.



(Basic Project)

2) Application and DB Server Separation:



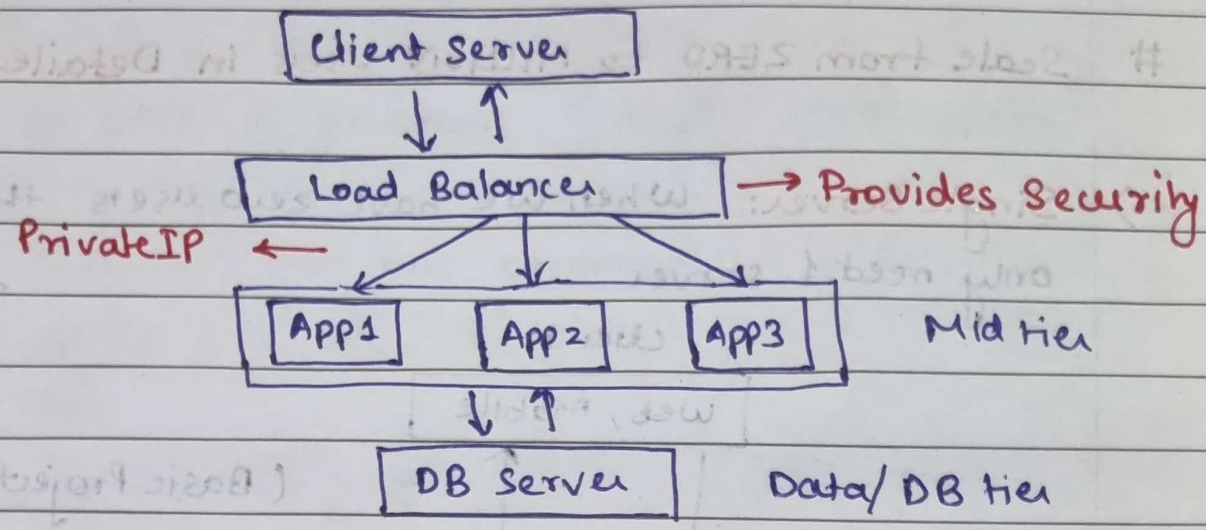
Mid tier (Business logics implementation)

Data/DB tier

→ Use: we can independently scale application server and Database server.

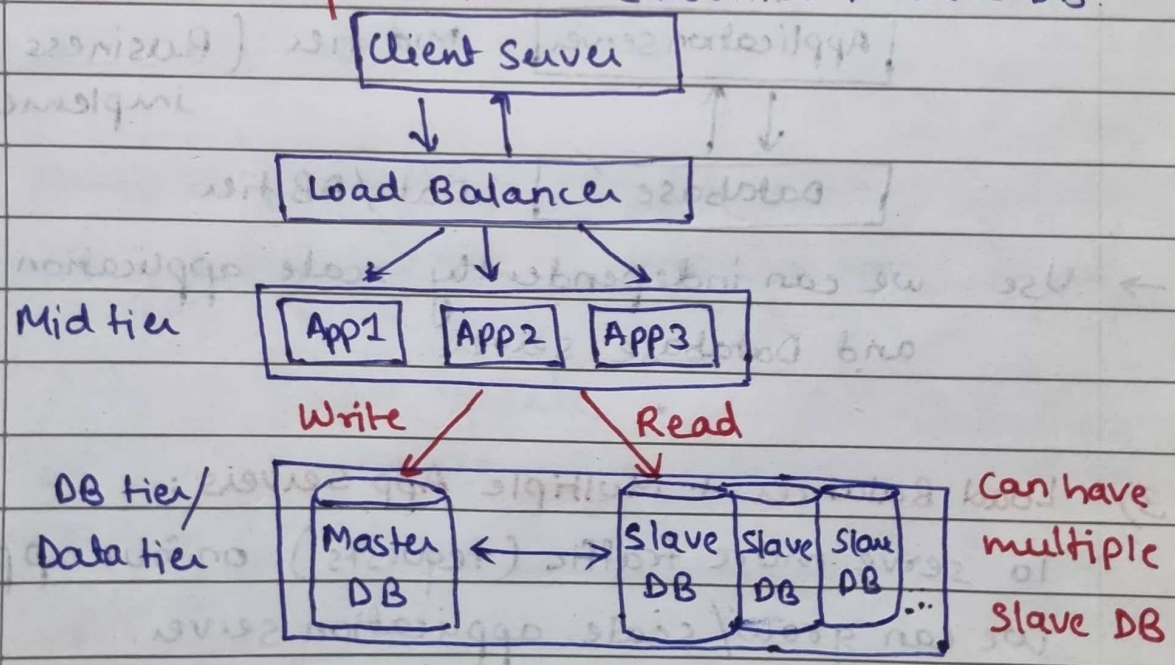
3) Load Balancer + Multiple App Servers:
To serve more traffic (requests) on our application we can grow/scale application server.

Date: __/__/____



- Load Balancer talks to Application Server using Private IP ⇒ hence LB brings Security.
- Also LB distributes traffic equally on each of the servers.

** 4) Database Replication: Encounter failure DB.



Date: __/__/____

- Master-Slave Architecture:

→ All write requests will go to Master DB and all read operation will go to slave

→ If one of the slave DB is failed, then another replica slave DB will be up, and will handle the requests.

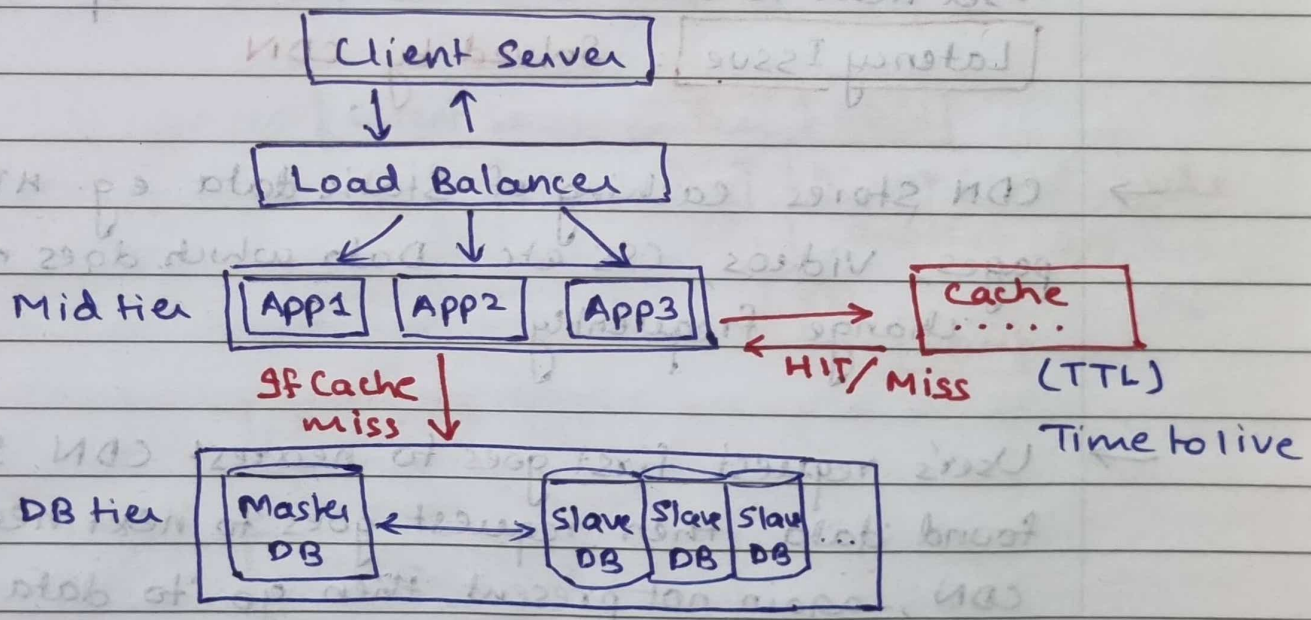
→ If master DB will fail, then anyone of the slave DB will get promoted to master DB.

5.) Cache:

→ Whenever our Application server talks with DB, this will be a network call and it is an expensive call.

→ DB operations are very very very expensive.

→ Increase performance / we have save DB calls

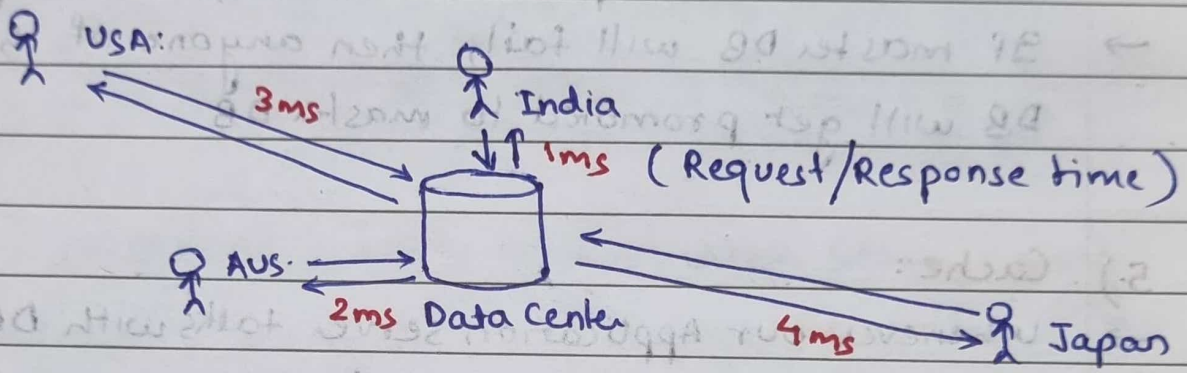


Date: __/__/____

** 6) CDN: Content Delivery Network

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

→ CDN has more functionalities apart from caching

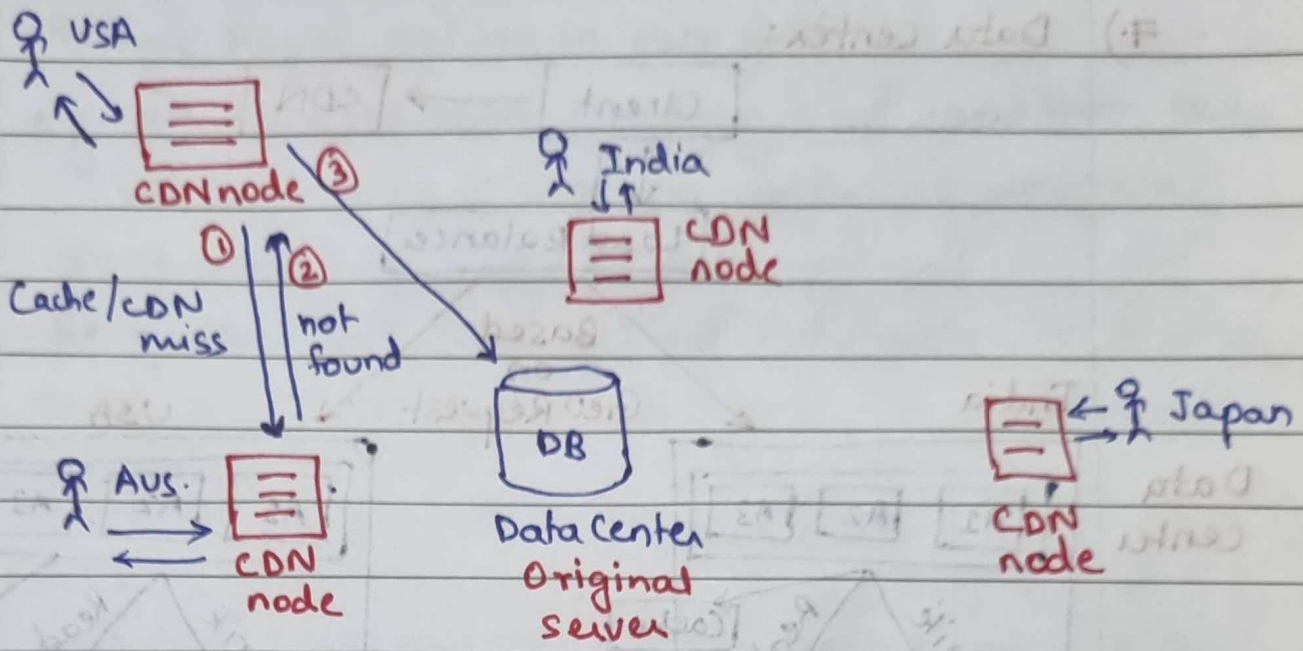


→ Based on user's location time for request/response has increased.
User near to data center has fastest response.
Latency Issue: Solved by CDN

→ CDN stores caching of static data. e.g. HTML pages, Videos, CSS etc. Data which does not change frequently.

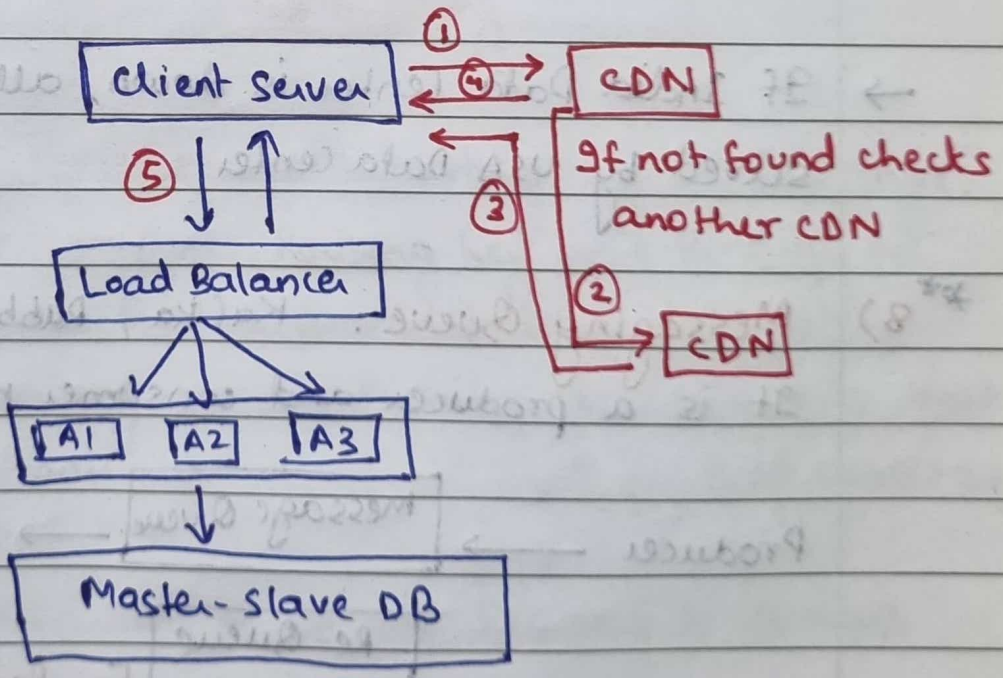
→ User's request first goes to nearest CDN. If not found data, then request goes to next nearest CDN, again not present then go to data center and stores response in CDN.

Date: __/__/____



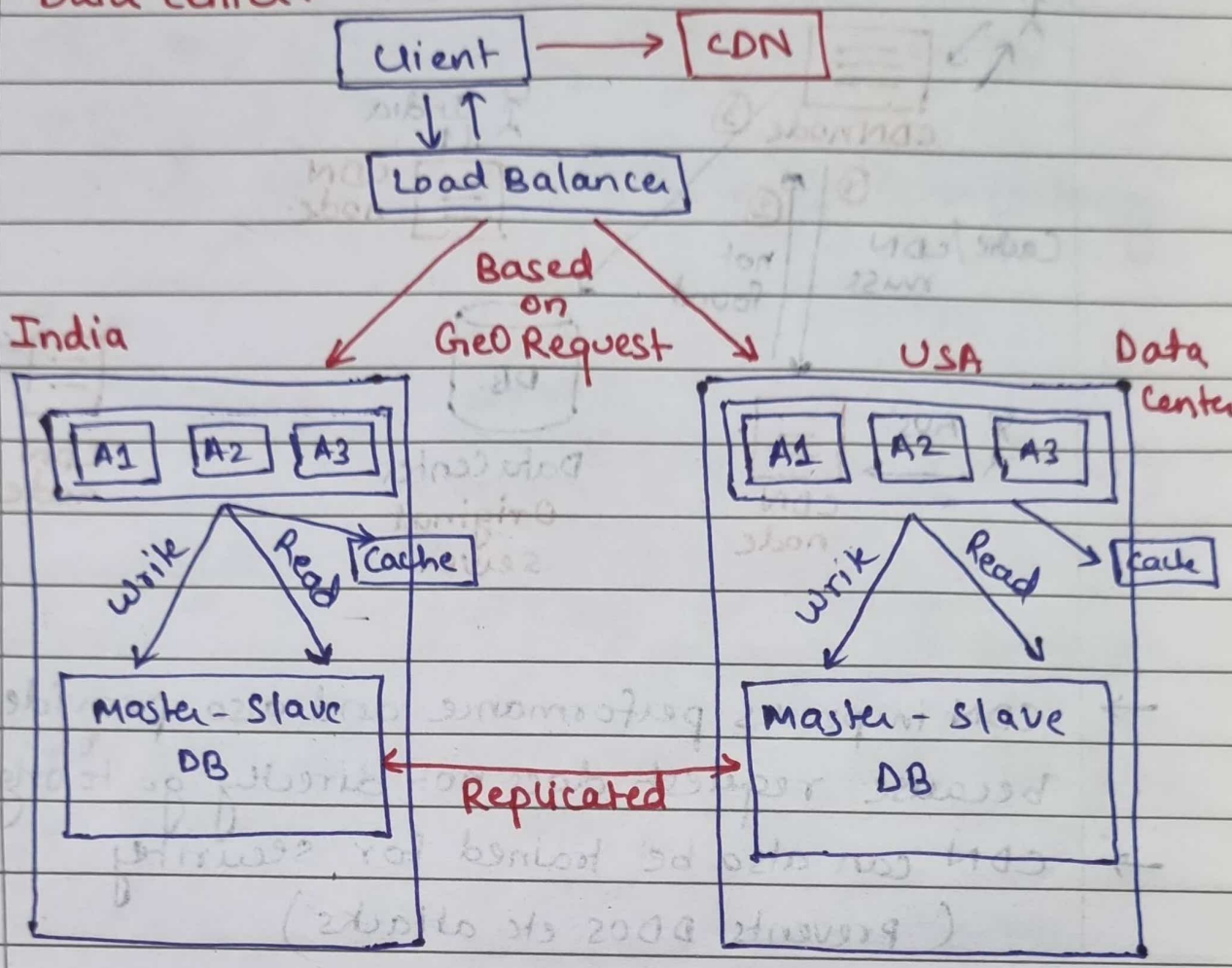
→ CDN improves performance and also provides security because request does not directly go to original DB

→ CDN can also be trained for security
(Prevents DDOS etc. attacks)



Date: __/__/____

7.) Data center:



→ If India Data Center is down, all requests will be served by USA Data Center.

** 8) Messaging Queue: Kafka, Rabbit MQ

It is a producer and consumer mechanism.

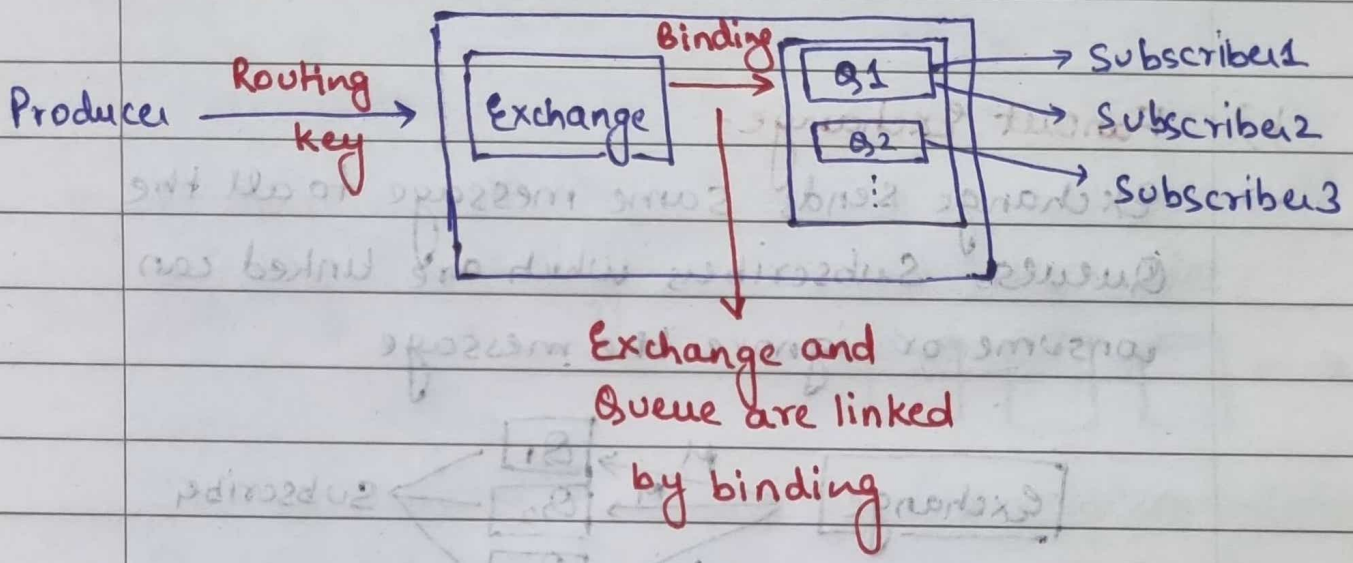
Producer → Message Queue → Subscriber

Re-Queue — For failed messages

Date: __/__/____

→ Brings Async nature in our codebase
e.g. we have to perform heavy load operation, then we can't keep on waiting until it's completed. So it should be Asynchronous

→ Re-queue will store failed messages, and after some time subscriber can again consume those messages.



Exchange: Producer sends routing key to Exchange. Exchange compares Routing key and Binding to decide which Queue needs to sends data to. Now, depending on which subscriber listens to that Queue, notification will be sent to that subscriber.

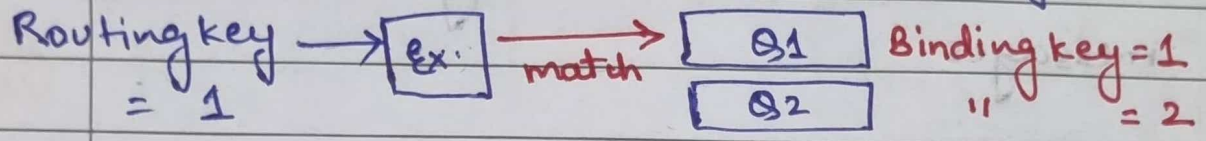
Exchange ⟶ sends Request to Queues.

Date: __/__/____

Types of Exchange:

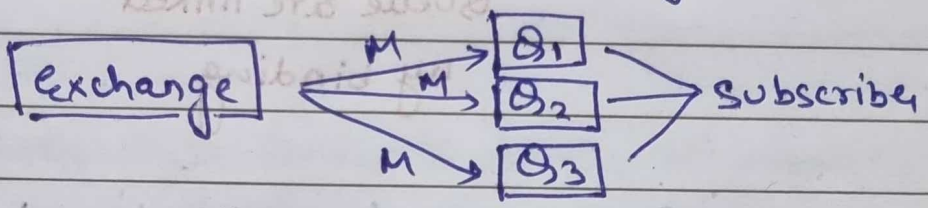
a) Direct Exchange - Exact match

→ (Routing key = = Binding key) → Sends to matched Binding key Queue



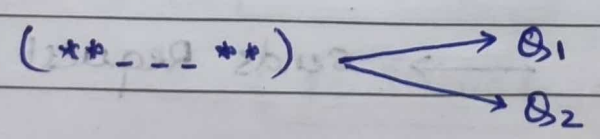
b) Fanout Exchange -

→ Exchange sends same message to all the Queues. Subscribers which are linked can consume or ignore the message.



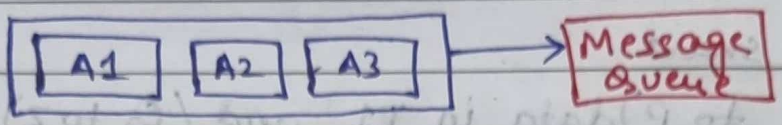
c) Topic Exchange: Regex Match

Based on regex match, request will be sent to that particular queue OR multiple queues matching with regex routing key.



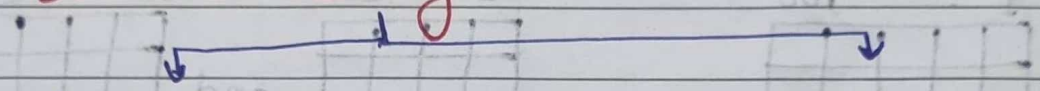
Date: __/__/____

→ Messaging Queue Present at Application Server level.



→ With Async nature, we can process the request faster.

**9) Database Scaling:



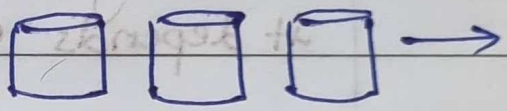
Vertical Scaling ⇐ Horizontal Scaling

i) It says increase the capacity of existing DB.

Capacity means
→ RAM increase ↑
→ CPU increase ↑

ii) But there is always a limit.

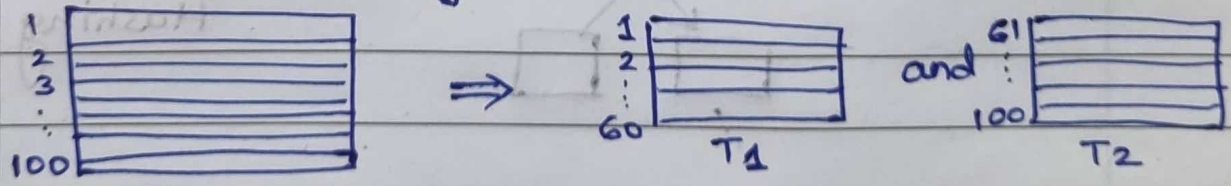
i) We can add more Database nodes.



ii) It's implementation is: SHARDING.

SHARDING: Horizontal Shading

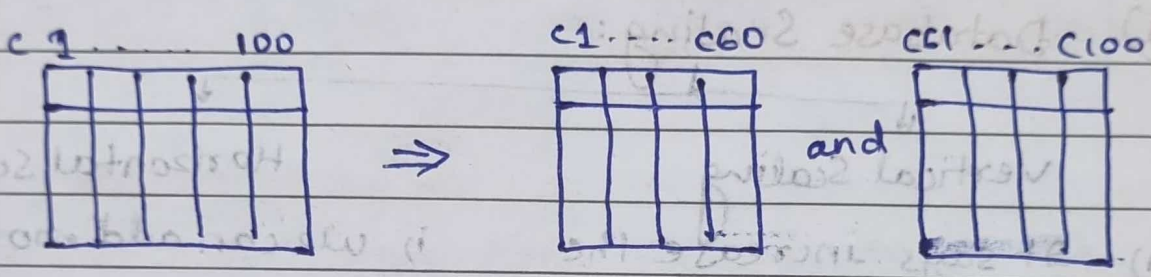
Divide the large table row wise:



Date: __/__/____

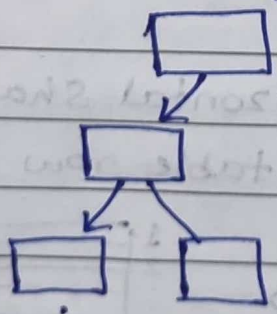
→ Based on some mechanism we can decide which rows can move to T1 shard and which rows to T2 shard.
e.g. (A to P) data in T1 and (Q to Z) in T2

Vertical Sharding: Data will be divided column wise.



→ Generally Horizontal Sharding is better, but it depends on the requirements.

Drawback of Sharding:
e.g. If we have A named data in millions, then T1 needs to sharding again.
Similarly a tree dependency can be created



⇒ Resolved by Consistant Hashing

Date: __/__/____

ii) Second issue is join Query: It can be solved by de-normalizing the tables.

Hence we covered all 9 steps which were required for 0 to million scaling.