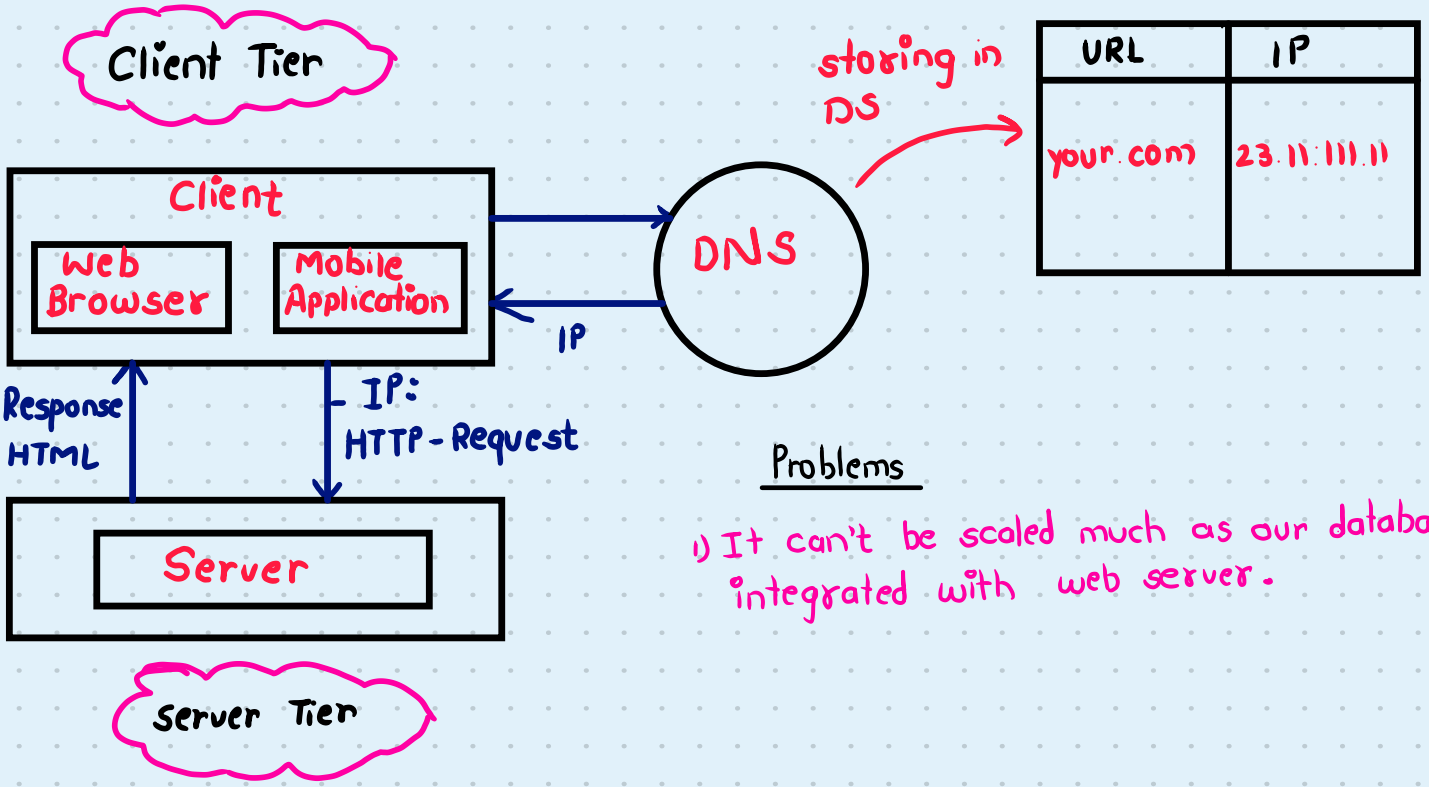
Lecture 2: Scaling from 0 to million user

* Remember about HLD
  ↳ Anything that we will do in HLD is to make our application scale better

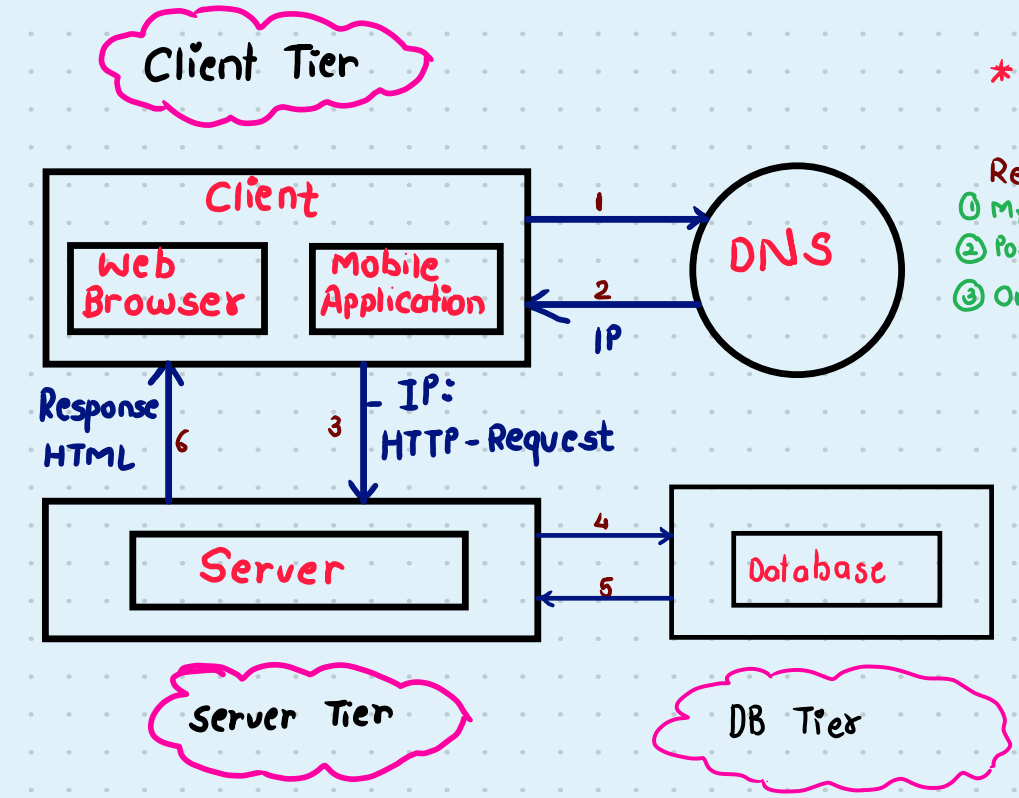
* Simple Client - Server Architecture



Problems

1) It can't be scaled much as our database is integrated with web server.

* Improved version - Introduced DB Layer



⊙ Discussion on DB

* Which DB to use ?

Relational
1. MySQL
2. PostgreeSQL
3. Oracle SQL

Non-Relational
1. key-value store
   ↳ Amazon Dynamo DB
2. Document based store
   ↳ Mongodb
3. Column based store
   ↳ Casandra
4. Graph based store
   ↳ GraphQL
   ↳ store relationship data

Ex: Insta ⇒ following each other (bidirectional graph)

Parameters to decide which db is great ?

① Can we perform query on our db ⟹ Yes ⟶ relational

② Faster execution of request / Less latency ⟹ Non-relational

③ But data is not structured ⟹ Non-relational

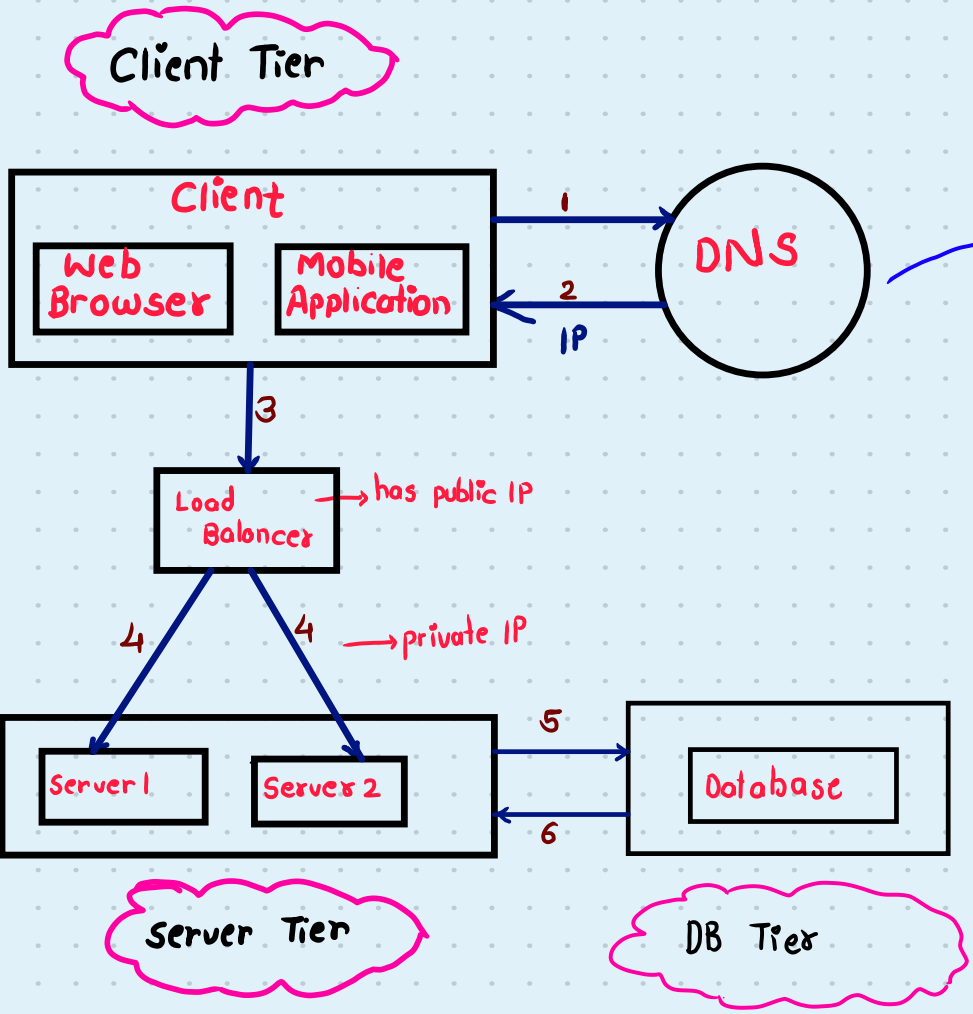
Problems : 1) It can only handle few thousands user ⇒ we need to scale up but vertical scaling will lead to SPOF (single point of failure) ∴ we need horizontal scaling

✱ Improved Version : Multiple Server + Load Balancer

1) We created multiple server to handle request

2) But how will DNS / we decide when to hit request on which server

3) To solve this we will introduce Load Balancer



⊙ Load Balancer

↳ balances the load on server and redirect requests to server less busy / idle

Now DNS will not store IP address of server instead it will store of load balancer

↳ And then we will host our server on private IP so that no one can hit req. or check our server

• Problem -

1) Now, we have scaled servers

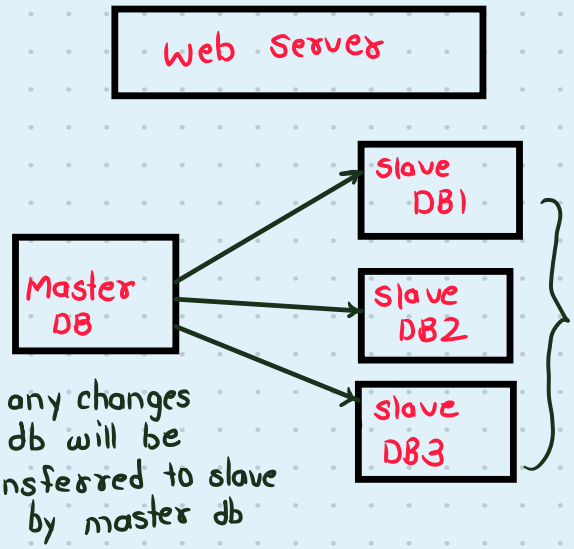
2) But our database is also a SPOF ⟶ we need to improve it to avoid physical damage on db and have data properly integrated

↓

damage like flood, tsunami near / in data center

* Improved Version : DB - Scaling
  ↳ To acheive this we will use Master Slave Architecture

Master - Slave architecture



Queries that can be performed
- Read (select) : slave db
- Write
- Insert
- Update
- Delete

} master db

↳ any changes in db will be transferred to slave db by master db

↳ after update it will update its own read query
↳ which slave server will give data
  ↳ we will use Load balancer to decide it
↳ later on discussion in detail

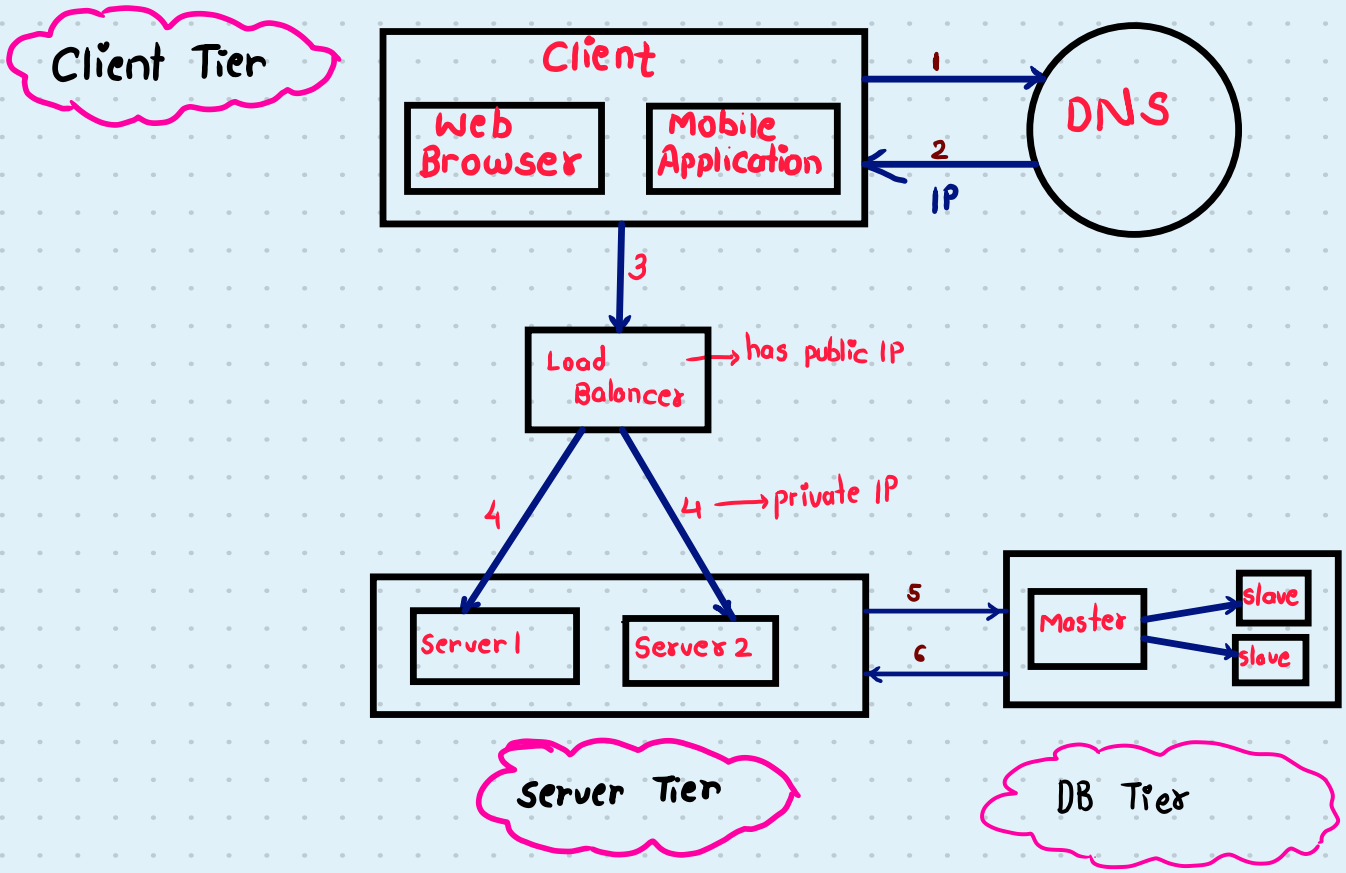
What problems are solved?

① If any slave db is crashed we will have option and data store in master db

② What if master db crashed?
  ↳ In that case, we will make any of the slave db as master till old master db is up or new master db is created

Which slave? => multiple algorithms are used to decide it most common is voting algorithm

③ What if we made changes in db like added insta post & before data is transferred to slave db, master db crashes → H.W → find ans



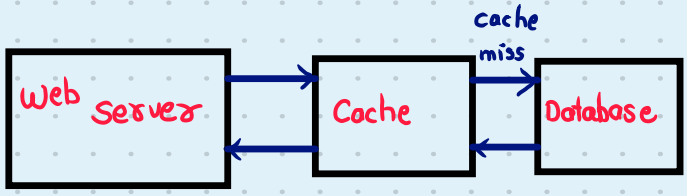
Que] Why do we need server code or language like Java/Python in our application when HTML + CSS + JS are enough to handle any logic or processing

Inshort : Why server side code ?

Ans If we did not introduced server side code then our database will be exposed and it will lead to data loss, wrong data injection, security issues and no authentication.

* Improved Version : Cache

↳ Cache helps in reducing query requests on db



↳ cache has many algorithms and internal implementation.

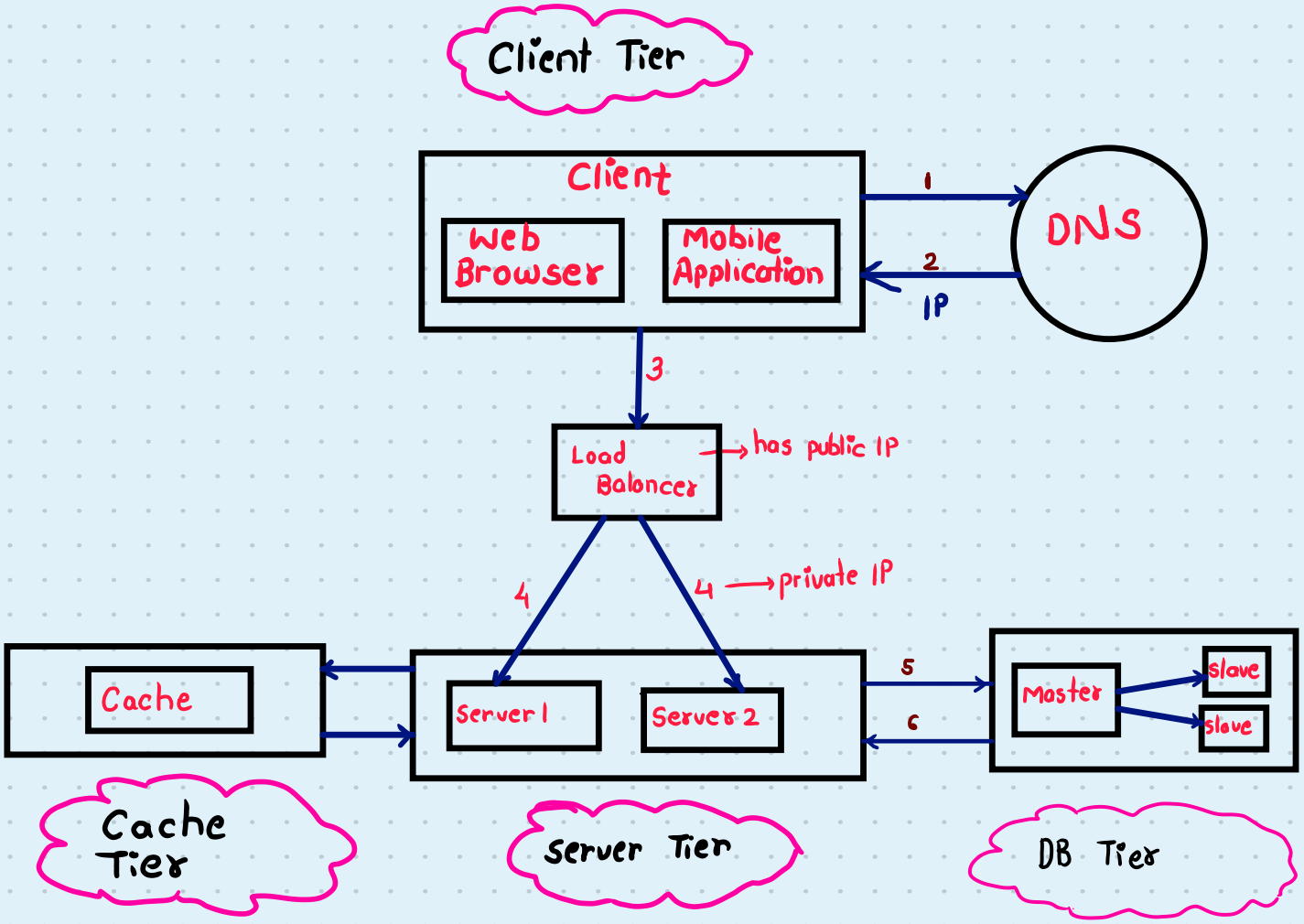
Cache Eviction Algorithm - decides which and when data is removed when cache is removed

Cache

↳ It is faster than DB any operation like read and write are fastly executed on cache.

↳ Cache will store frequently asked query and then if we user requests it then it will check if it is not stored in cache or not

↳ helped improve latency



Improved Version : Static Content → CDN

- If there are too much static content in our website then it is waste to refresh and bring all data from backend / server
- Therefore to avoid this we use CDN - Content Delivery Network

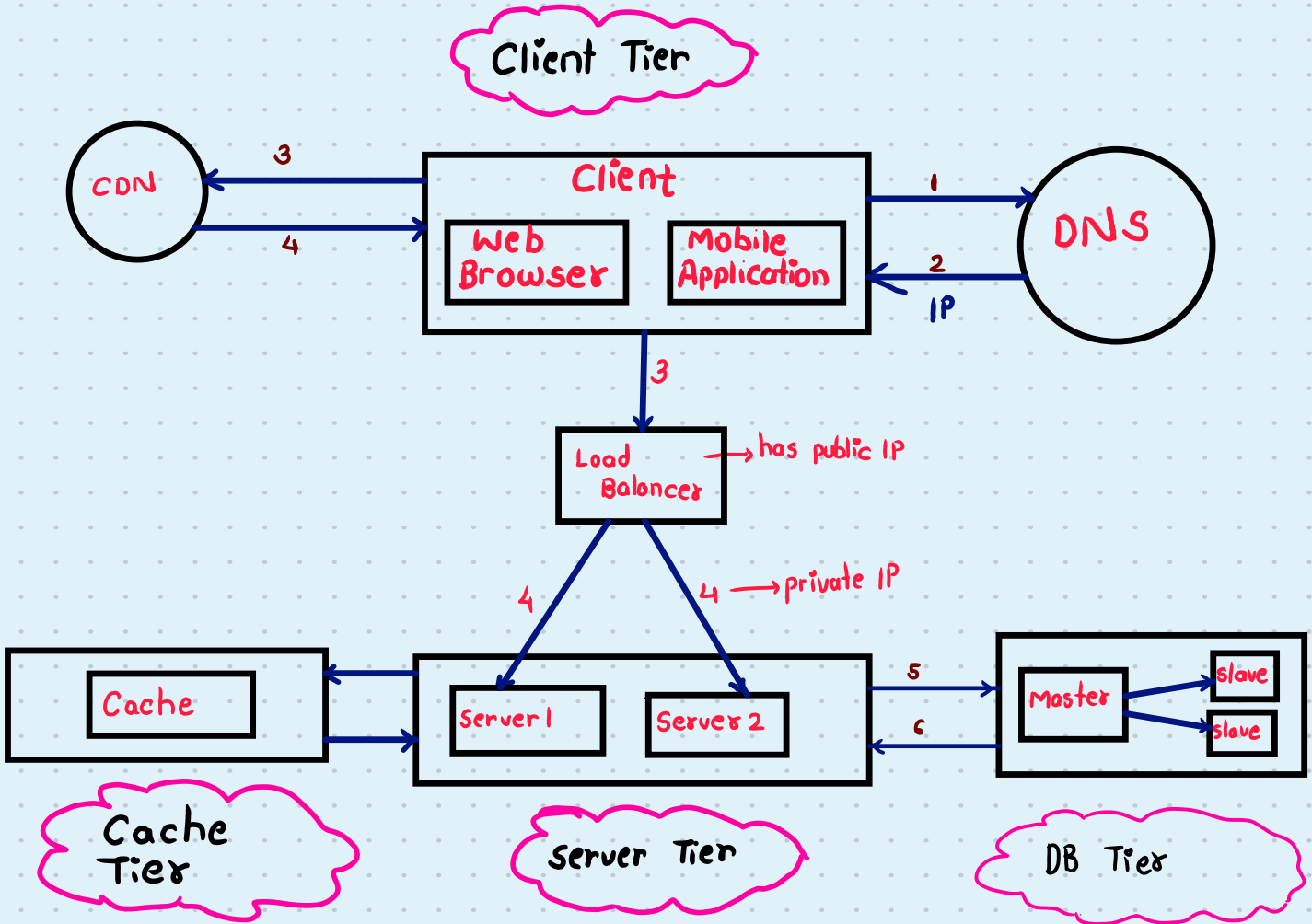
What is CDN ?

- It is a 3rd party vendor like DNS. CDN acts like proxy of web server.
- It's task is to load static data without making the process go through load balancer → cache → db

Available options for CDN

1) Amazon : CLOUDFRONT
2) AKAMAI

- Geolocation of CDN matters the most
- If client requests for data, CDN checks all the available server and selects the one which is closer
- DNS mein ab humare CDN ki ip address store hoga



Improved Version - Auto Scaling of Servers

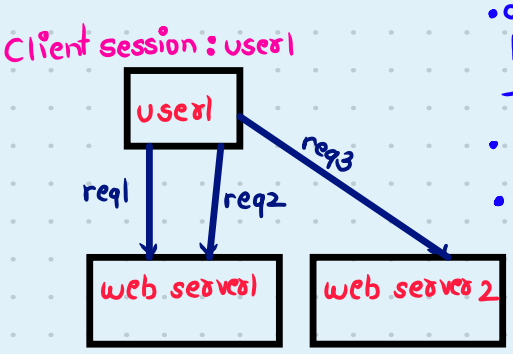
Why do we need auto-scaling ?

① Agr humara application pr kaam/less users aa rahe hai toh bahoot saare servers and cdn up rakhkar loss hoga ⇒ ∴ depending on traffic of users system/server scale down ho jaane chahiye

② Incase, users bad gaye toh instead of latency and server crash, more server should be available i.e scale up hona chahiye

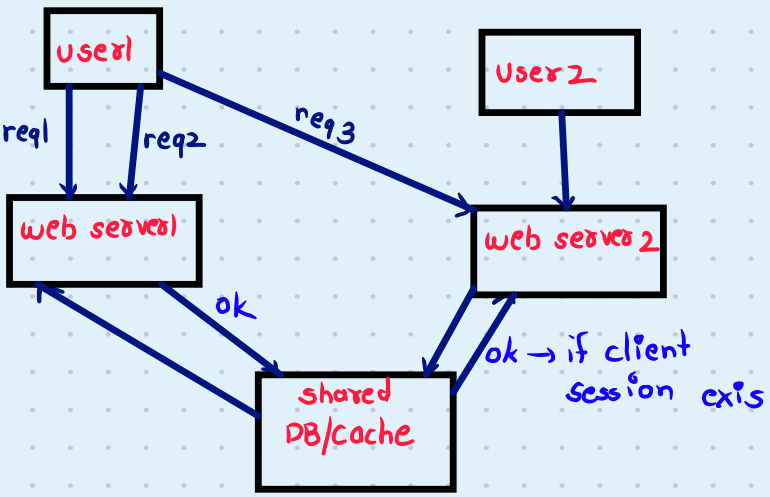
Before understanding auto-scalling we need to understand about client sessions

⊙ Suppose user1 ne login kiya on webserver1 and uske baad usne kuch request kiya before fulfilling request usne verify krna na ki kya user login hai i.e. uska client session active hai



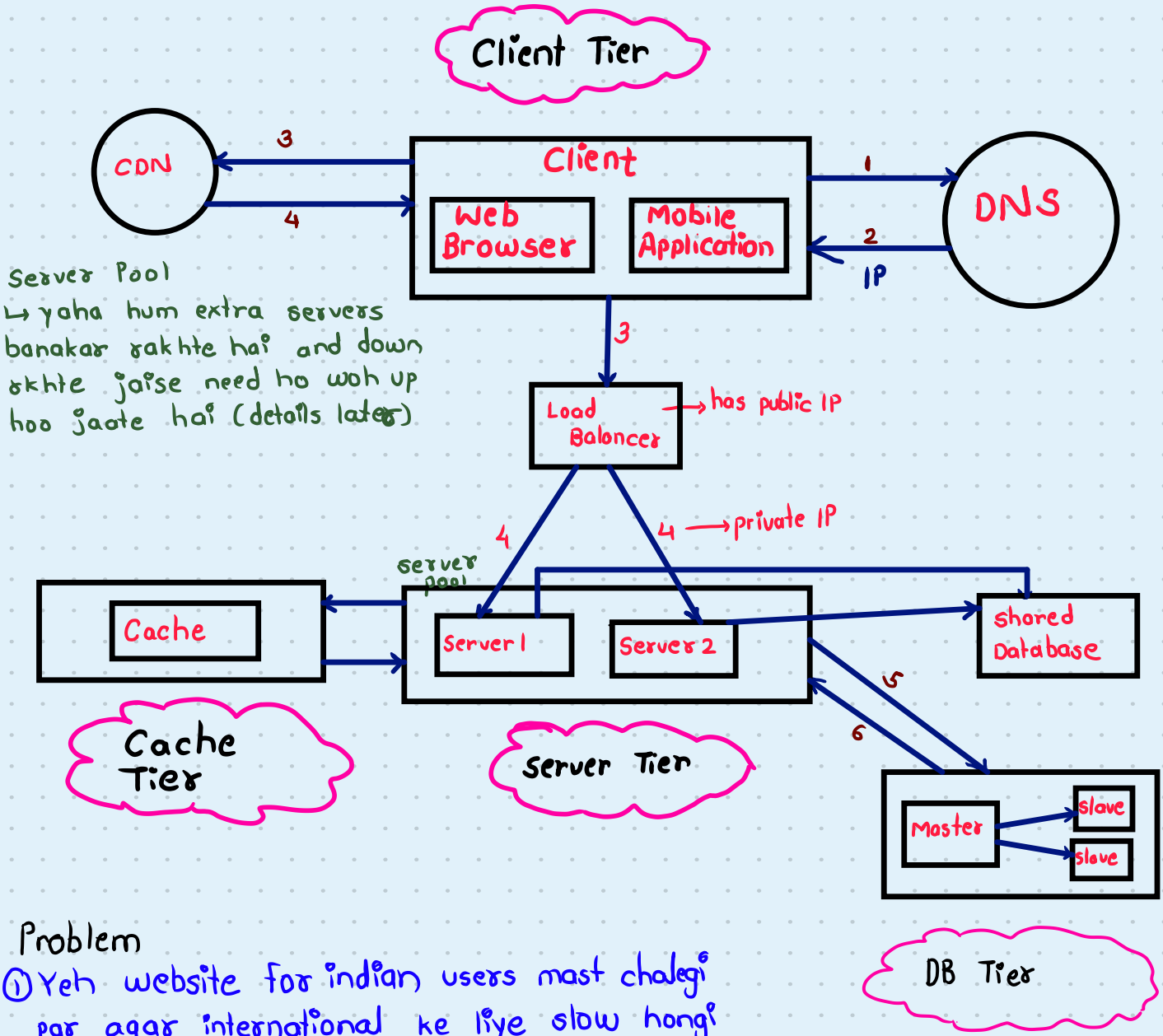
- ab aisa ho sakta hai load balancer next request webserver 2 par bhej de par uske pass user ko client session ki information hi nhi hogi
- so server2 req ka response nhi dega
- isse kehte hai **sticky sessions / stateful architecture**
  ↳ bekar!!

⊙ Solution :
- hum architecture from stateful se stateless le jaaye
- Kaise? ek separate db introduce krenge joh store karega client sessions ko and before webservers responds to any request woh db mein check karenge that whether client session exists or not



- HTTP Protocol **stateless** hote hai i.e. it does not require information of client session
  ↳ server ke liye har request unique hoti hai

- For autoscaling of servers we will use stateless architecture by having shared db for client session info



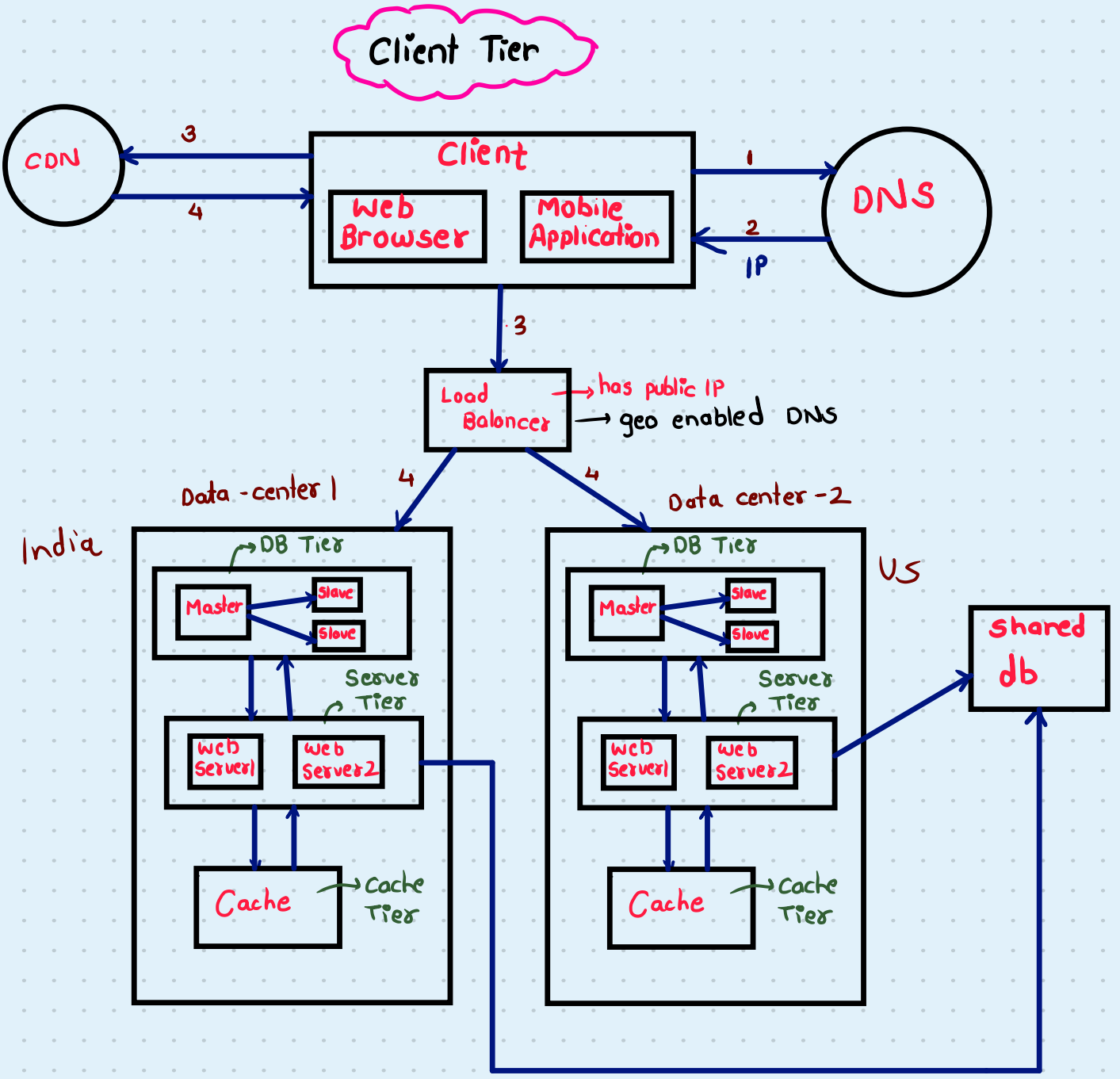
Problem

① Yeh website for indian users mast chalegi par agar international ke liye slow hongi coz uske saare datacenters India mein honge

② And data packets ko travel krne mein bahoot time lagega.

✱ Improved Version : Multiple Data Center

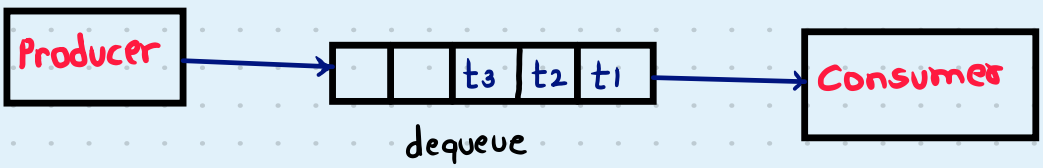
- Ismein ek hi shared db hota hai joh alag alag geo-locations ke data center se coordinate karta hai
- Load balancer geo enabled dns par kaam karta hai



⊛ Improved version : Messaging Queue

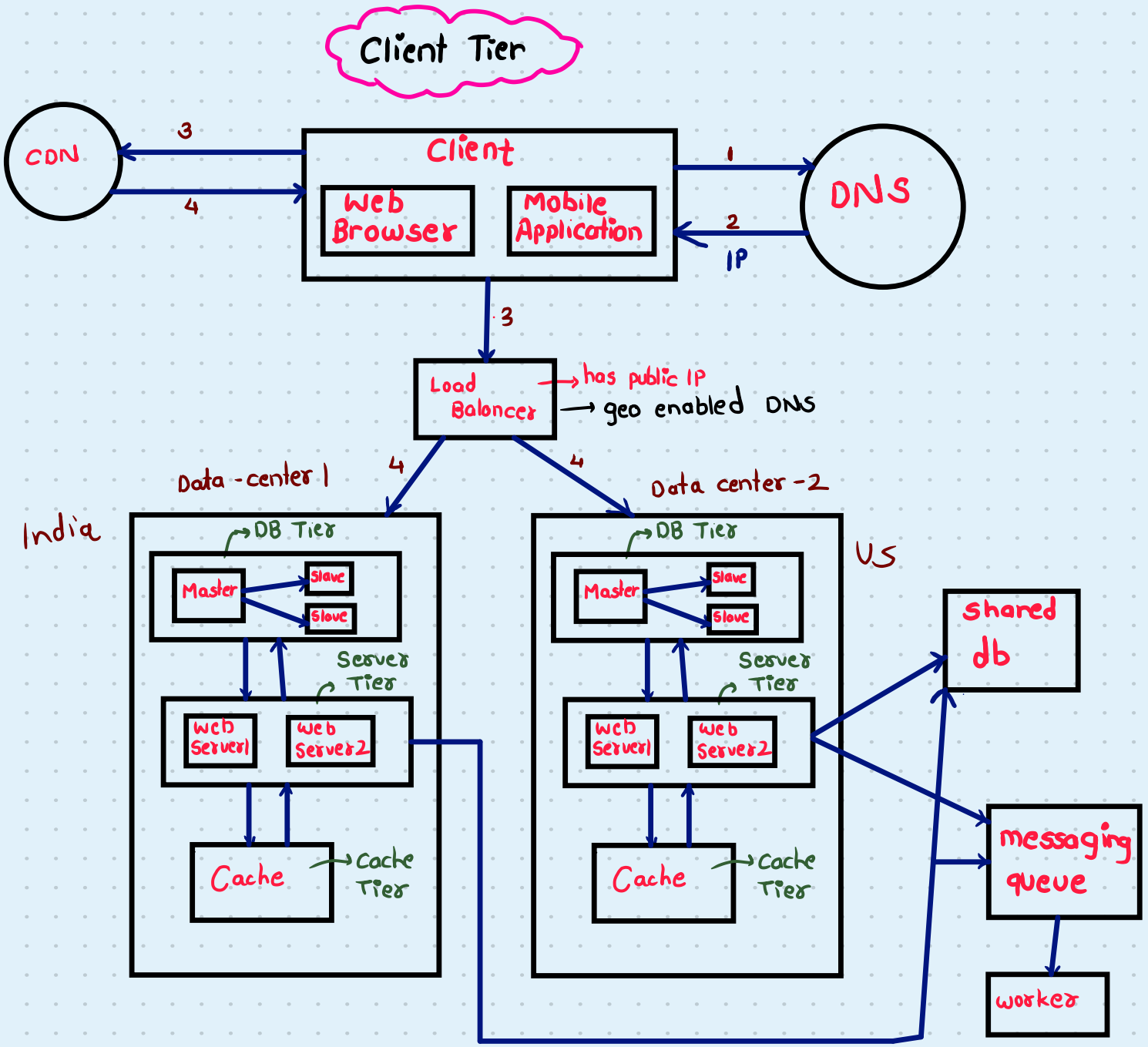
① Jaise hi humara LB koi req server par bhejta hai toh server busy hoo gaya uss request ko satisfy karne mein - aise hi bahoot saare kaam agaye toh isse latency ↑se ho jayegi ⇒ solution : messaging queue

Messaging queue
↳ two components
→ producer - adds task in queue
↳ consumer - takes task from queue to execute



• isse hum publisher - subscriber model bhi bolte hai

Types of Messaging Queue -
① kafka
② RabbitMQ



✳ Final Improvement : DB Sharding / Partitioning

↳ horizontal scaling

① Now suppose our db m can store 10 Million users but we need more so we can increase ram, still not enough.

② Then we consider horizontal scaling i.e. dividing data into multiple database

Ex of SQL db

we divided data based on mod 3

uid	name
1	saksham
2	abhay
3	sau
4	sakshi
5	avi

mod % 3 → coz dividing in 3 db

DB0: sau

DB1: saksham, sakshi

DB2: abhay, avi

- mod %3 on uid → known as sharding key
- db0, db1, db2 → known as shards
- Now if user wants uid=12 it will just mod %3 & search in that shard/db
- Now what if all shards are full we add one more shard / db but we need to divide data and the process is known as resharding
- Resharding is quite resource extensive, time taking process and can be achieved by data de-normalization

Note:

More details of each topic will be covered in separate upcoming lectures