* Scale from 0 to 1M users *

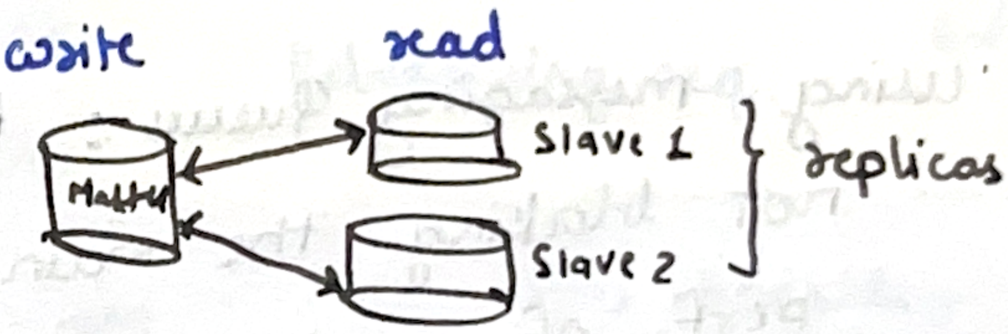
① ⇒ seperate out Mid tier (application servers) & DB to Let them scale independently

↓

② ⇒ Bring in LB and multiple App Servers
- if an app server fails the redistribute traffic amongothers
- ↳ also brings security as LB & app servers talks on private IP.

↓

③ ⇒ Database Replication



⇒ if master DB fails then one of the slave DB becomes master.

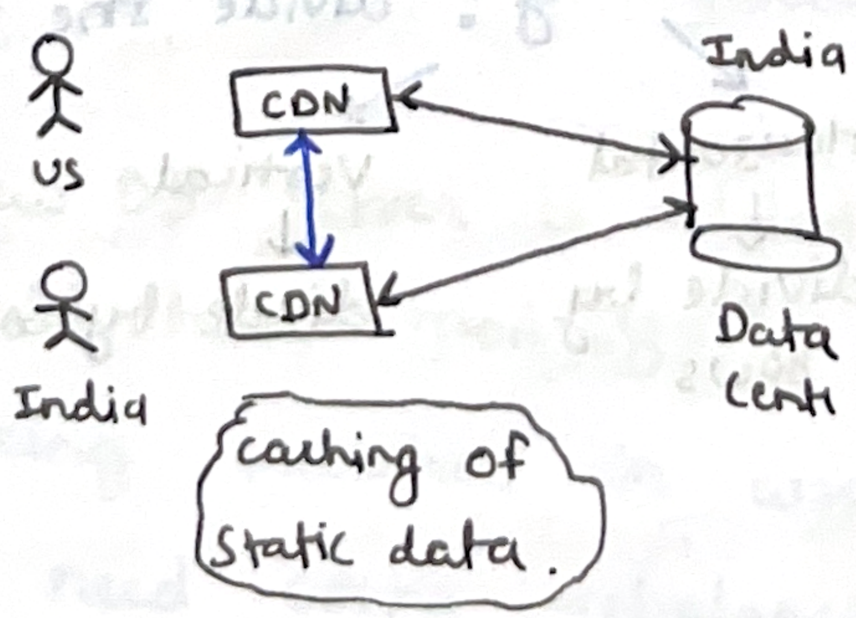
↓

④ ⇒ APP ↔ DB are expensive because it's an network call,

to overcome this we introduce cache → TTL (time to live) ⇒ this will make data from cache to be false.

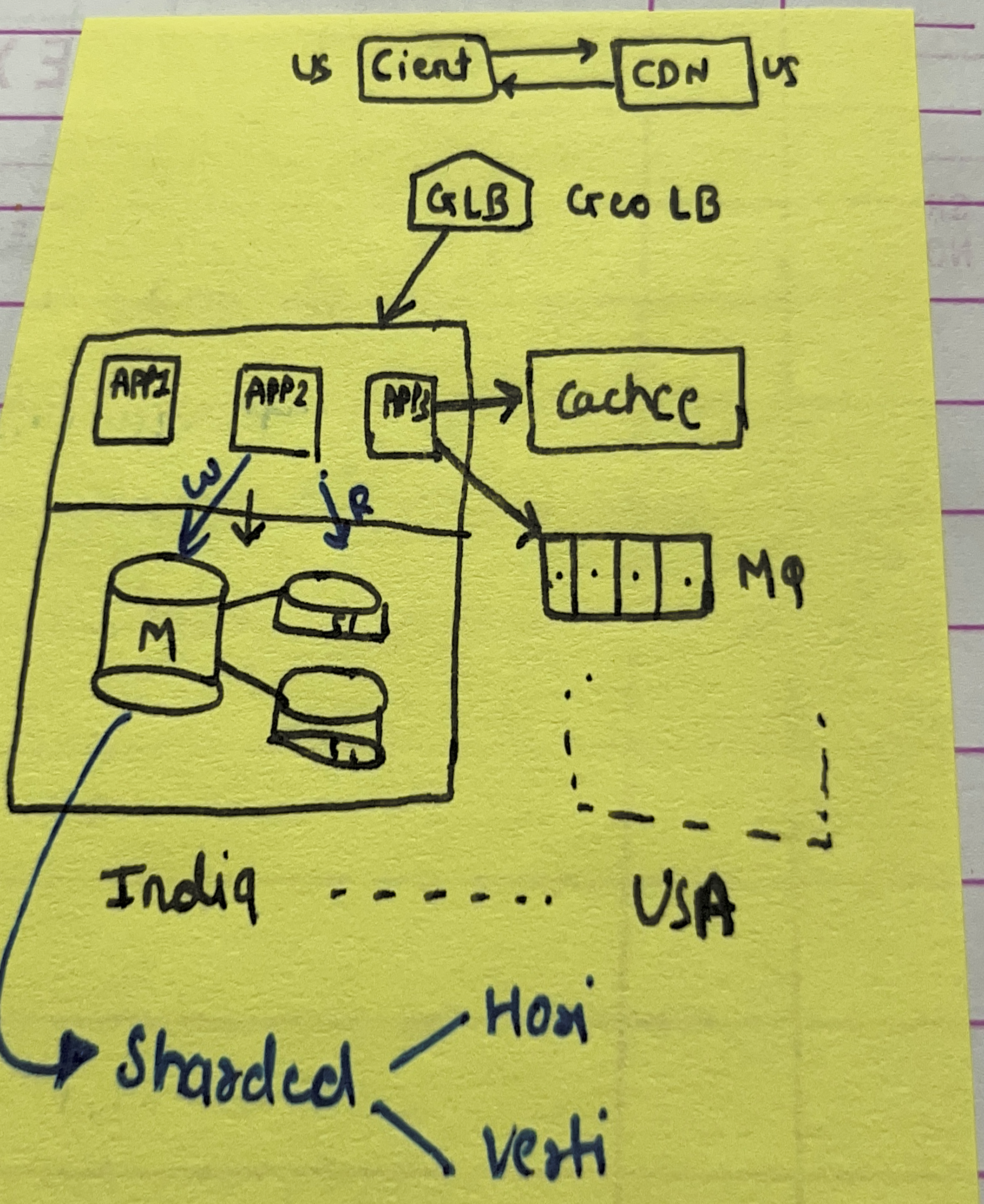
↓

⑤ ⇒ CDN: content delivery network. (cache)



CDN closer to user reduces latency for static content, if cache miss in CDN then go to nearest other CDN if miss then go to original DB.

TITLE	PAGE NO.	TEACHER'S SIGN



↓

⑥ multiple datacenters based on users location

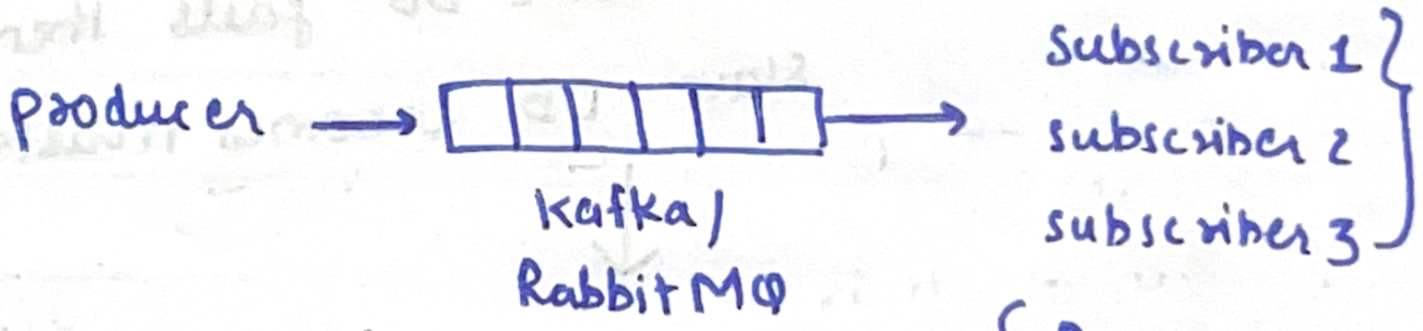
↳ same replica of application & database

⇒ here we have Geo Load Blancing, based upon latency to reach server.

✱ there is a replication bet^n zones & regions to support Disaster recovery.

↓

⑦ Using messaging Queues : for async exec of work & not blocking the main behaviour, this can improve perf of main service



↓

{ Processes messages Async till successful

Help to reduce load on main thru

⑧ DB scaling :

→ Vertical : ↑CPU ↑RAM (limited)

→ Horizontal : add more nodes

↳ Sharding : divide the tables.

↙ Horizontal ↓ divide by rows

↘ Vertical ↓ divide by columns

Drawbacks of Sharding:

- again and again re-sharding might be required.
- join can cause latency problems, so prefer denormalized table to avoid join.
- can be solved by consistent Hashing.