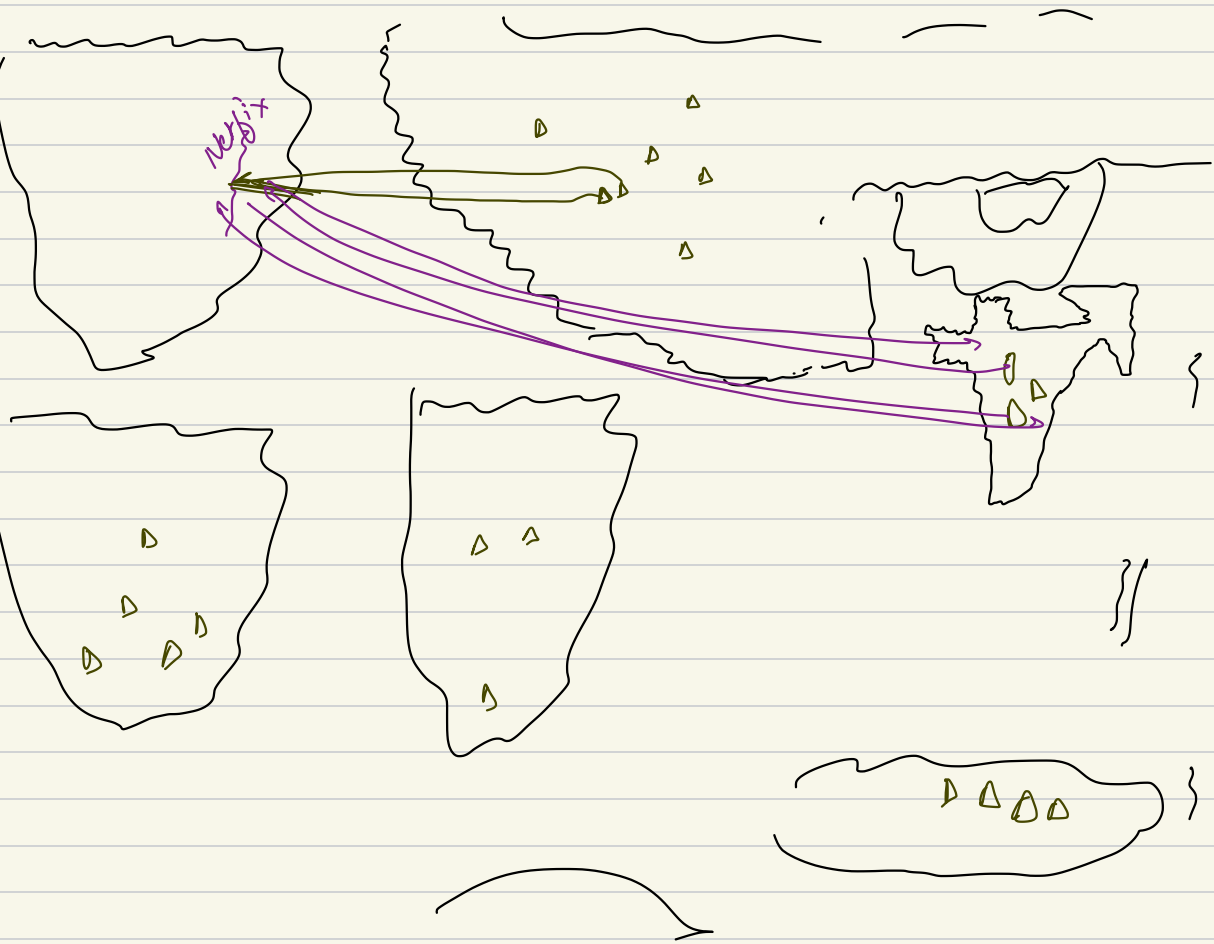
23/Nov/2023

Caching - 2

(4) Content Delivery Networks



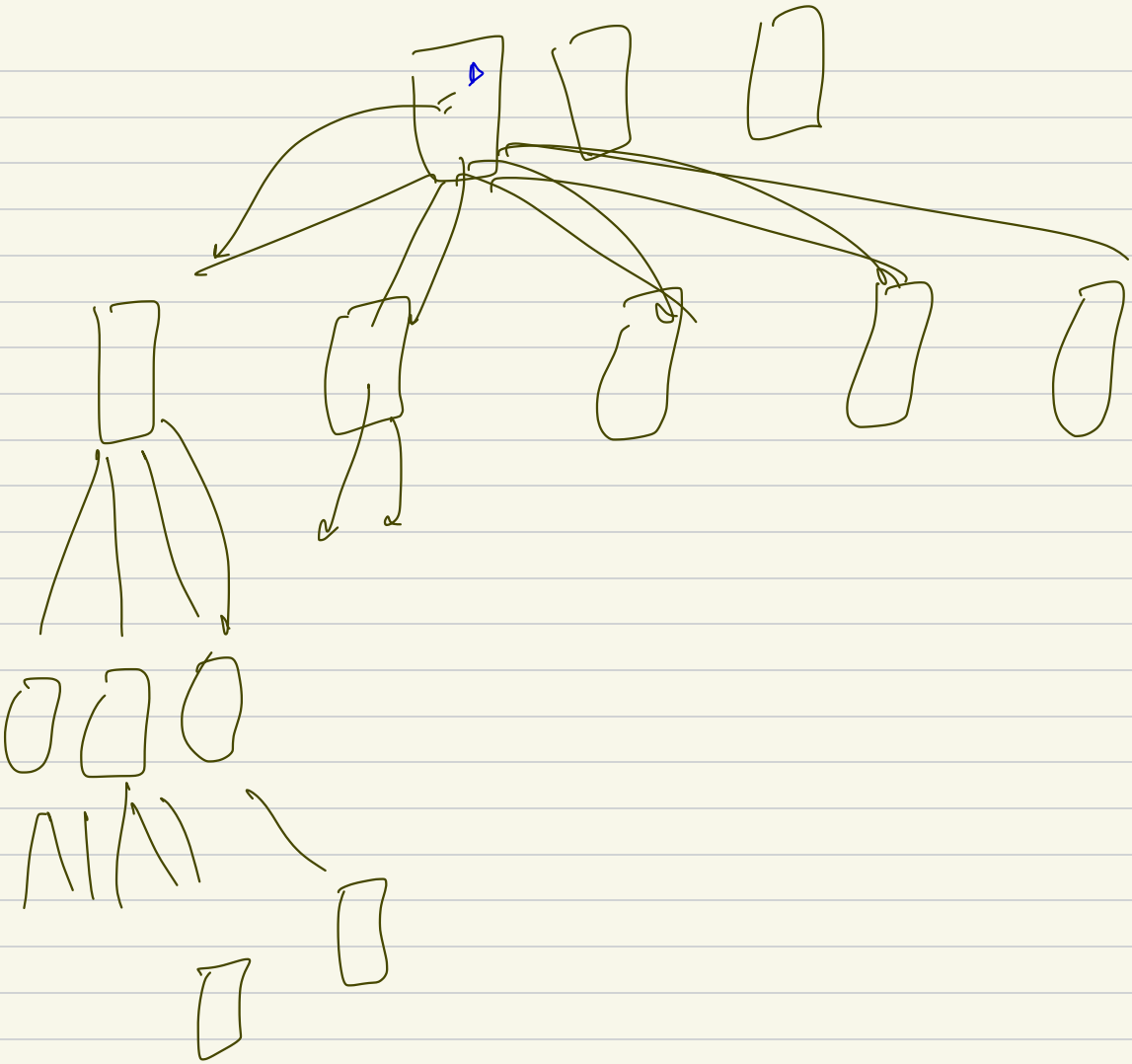
Big static media content (like Videos, Photos) etc. need to be transferred again and again for every individual user.

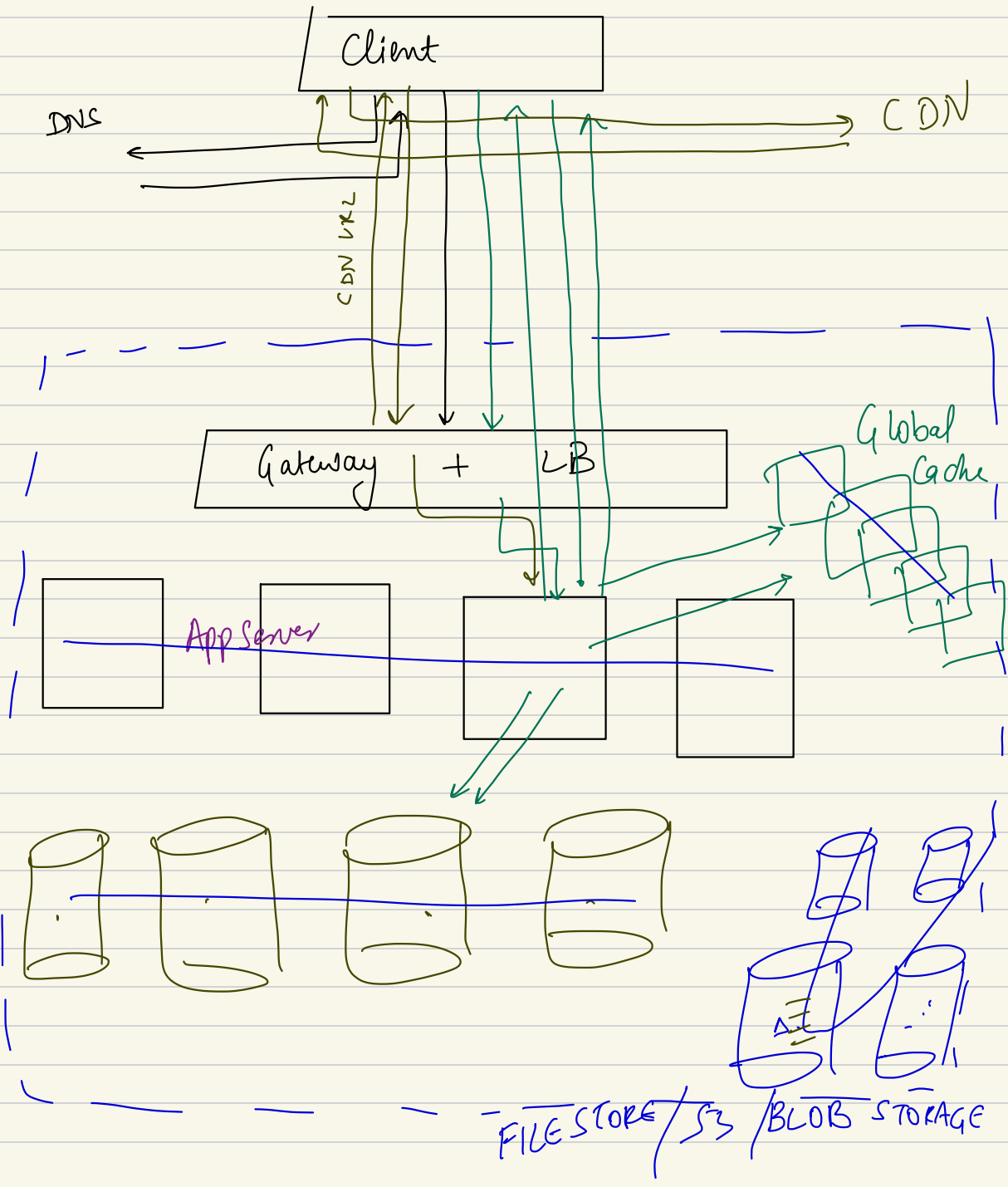
→ MBs, GBs...!!!

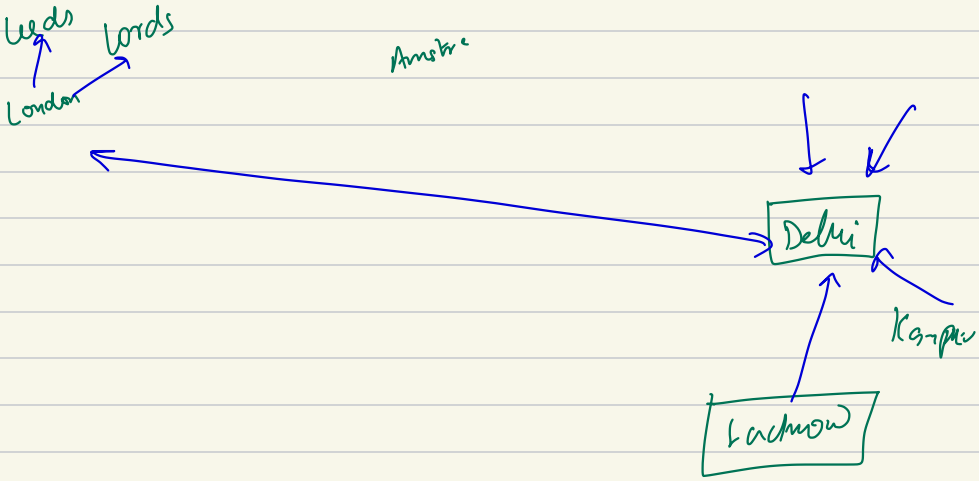
1. Netflix / Source datacentre is going to be clogged.
2. Duplicate data transfer for every individual user.
3. Latency of data transfer & Distance

Content Delivery Networks

CDN like Akamai




Client
DNS
CDN
CDN URL
Gateway + LB
Global Cache
App Server
FILE STORE / S3 / BLOB STORAGE


Leeds
Lords
London
Amritsar
Delhi
Kanpur
Lucknow

Cache - is just a copy

[can go out of sync]

Cache ① is NOT the ultimate store of data

hence it doesn't have the source of truth ☺

② Cache is limited in size

|cache| << << |DB|

you can't cache every data point and data update.

CACHE DATA CAN BECOME INCONSISTENT ☺ ☺

Cache Invalidation Strategy

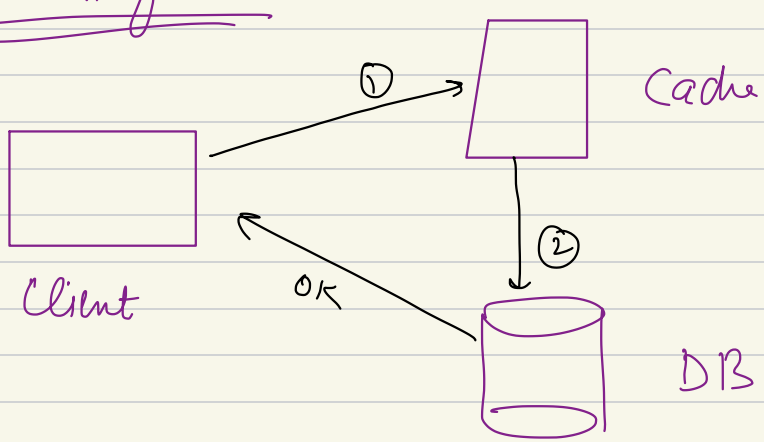
TTL (Time to Live)

→ +data (TTL = 10:38 pm 23/Nov/2023)

→ +data (TTL = 10:45pm 23/Nov/23)

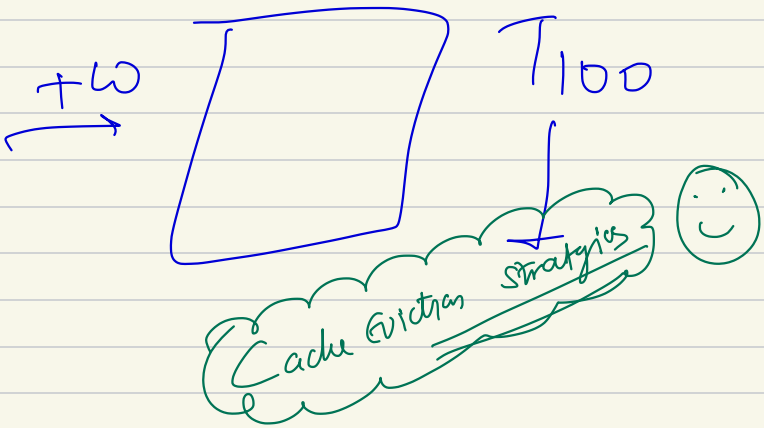
Strategies used for writing to Cache

(a) Write Through cache



detour

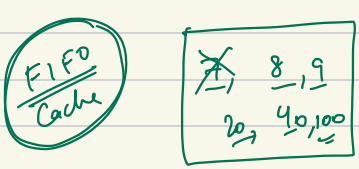
Given caches are smaller, they can get filled quickly....



- FIFO
- LRU
- LIFO
- MRU

(Most Recently Used)

ex



T8 → W(100)

T9 R(7) → cache miss

T1	W(7)	→
T2	W(8)	
T3	Read(7)	→ Cache hit ☺
T4	W(9)	
T5	W(20)	
T6	W(40)	
T7	R(40)	→ cache hit ☺

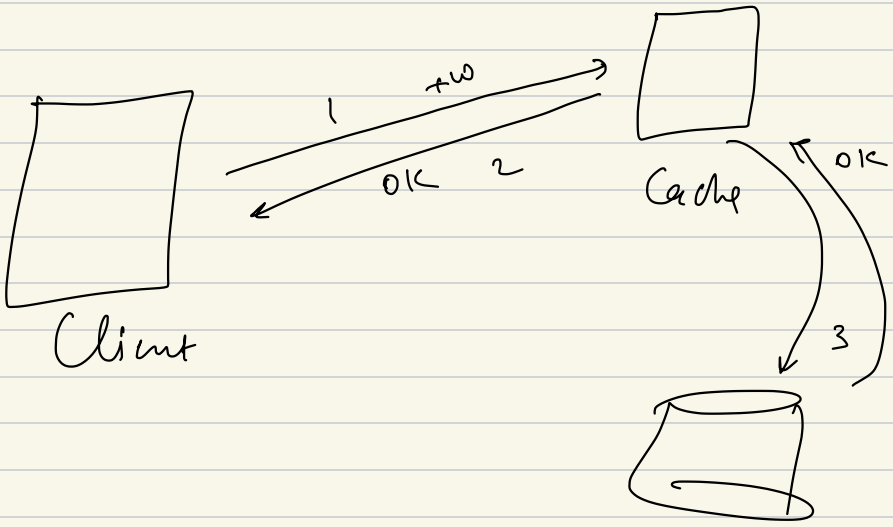
CON

1. Increase in write latency ↑↑↑
2. Given your cache size is limited, and in this strategy, you're trying to write everything to the cache; it will have loads of cache evictions...

PRO:

1. data consistency
2. If your system is Read heavy, then this system will speed up your reads....

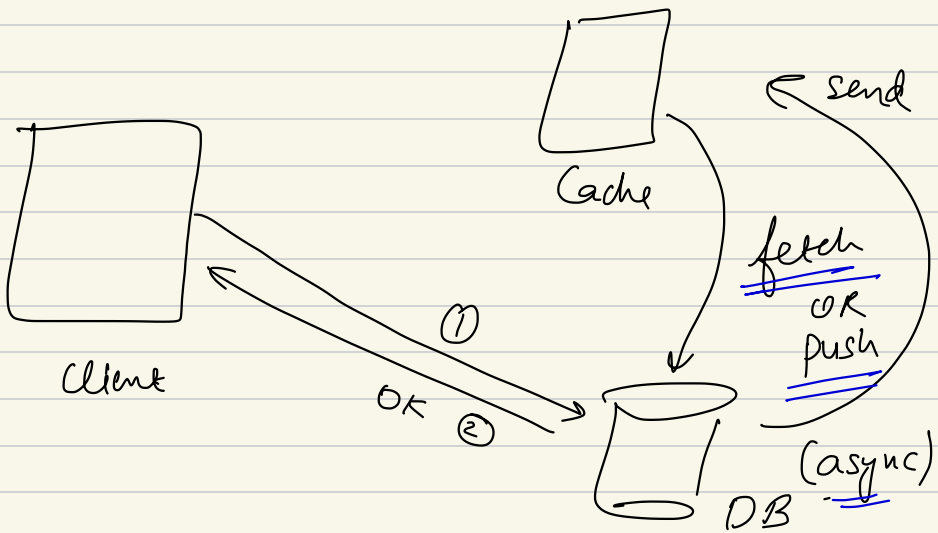
(2) WRITE BACK CACHE



PRO:
- very fast Write
- DB can handle writes in batches

CONS
→ data is inconsistent
→ data can get lost forever.

③ WRITE AROUND CACHE



PRO:

① data is always persisted

②

CON:

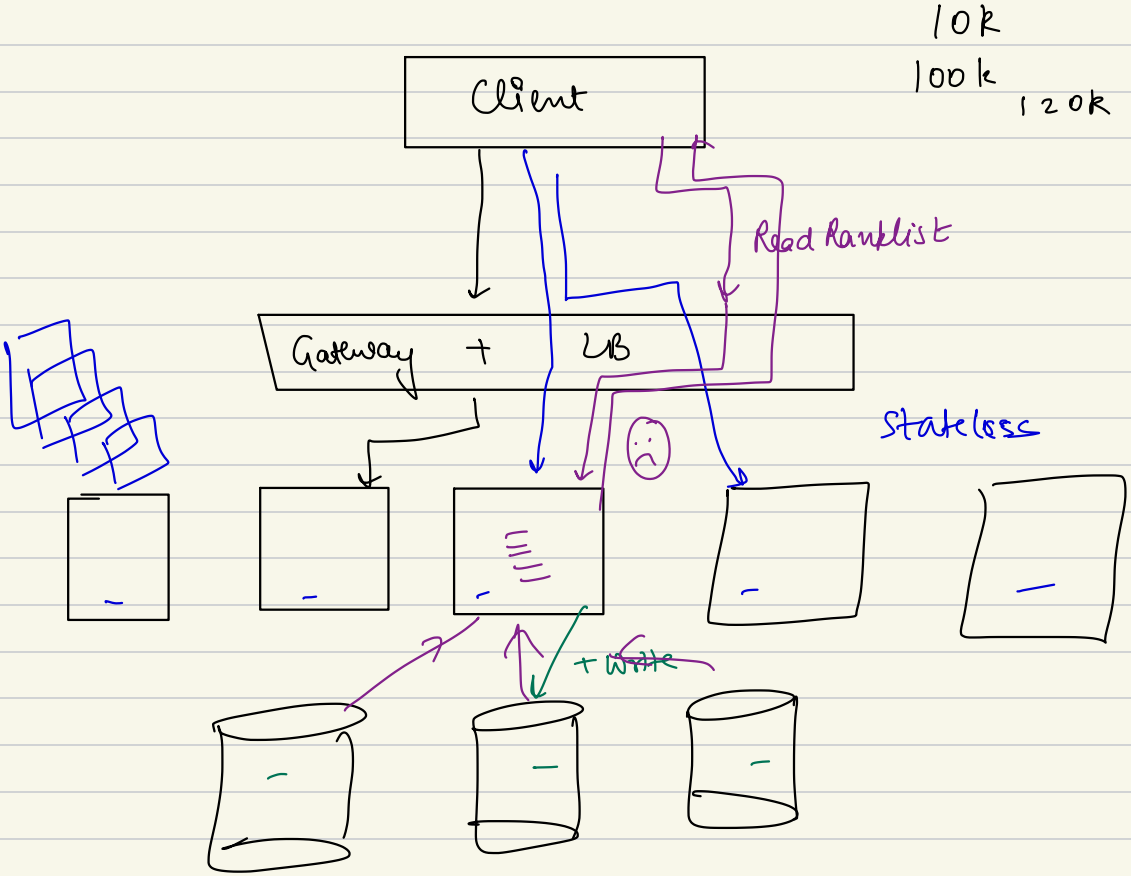
① high latency

② data is inconsistent

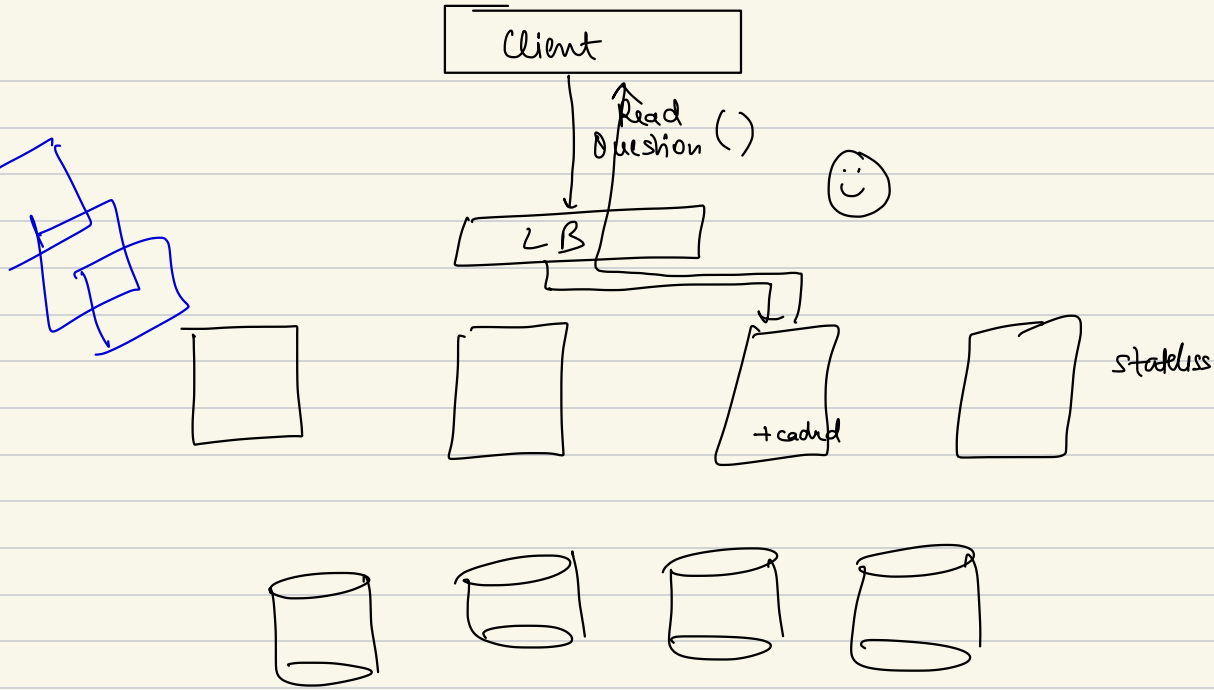
11:06 – 11:20pm Break

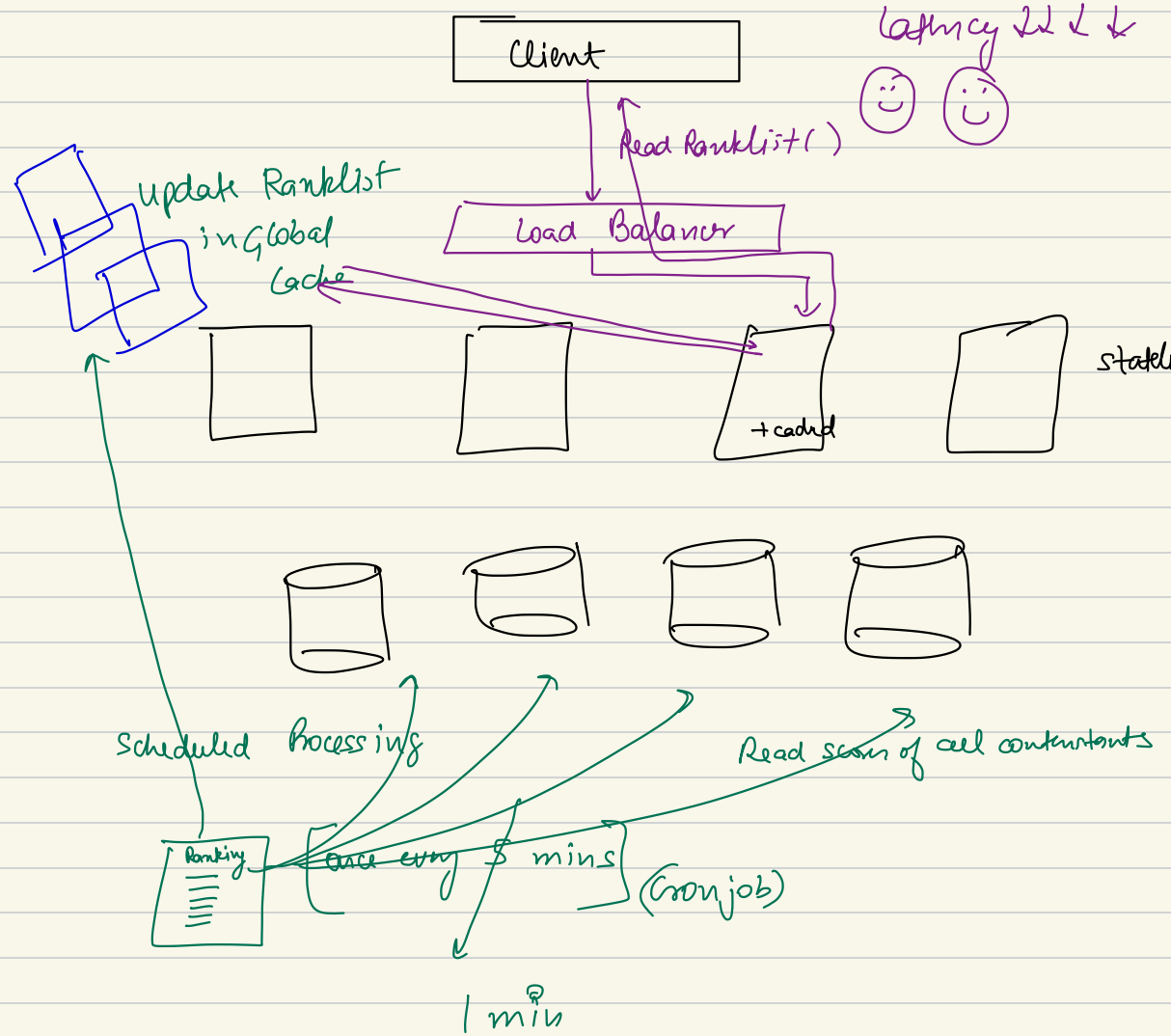
Real Life Examples

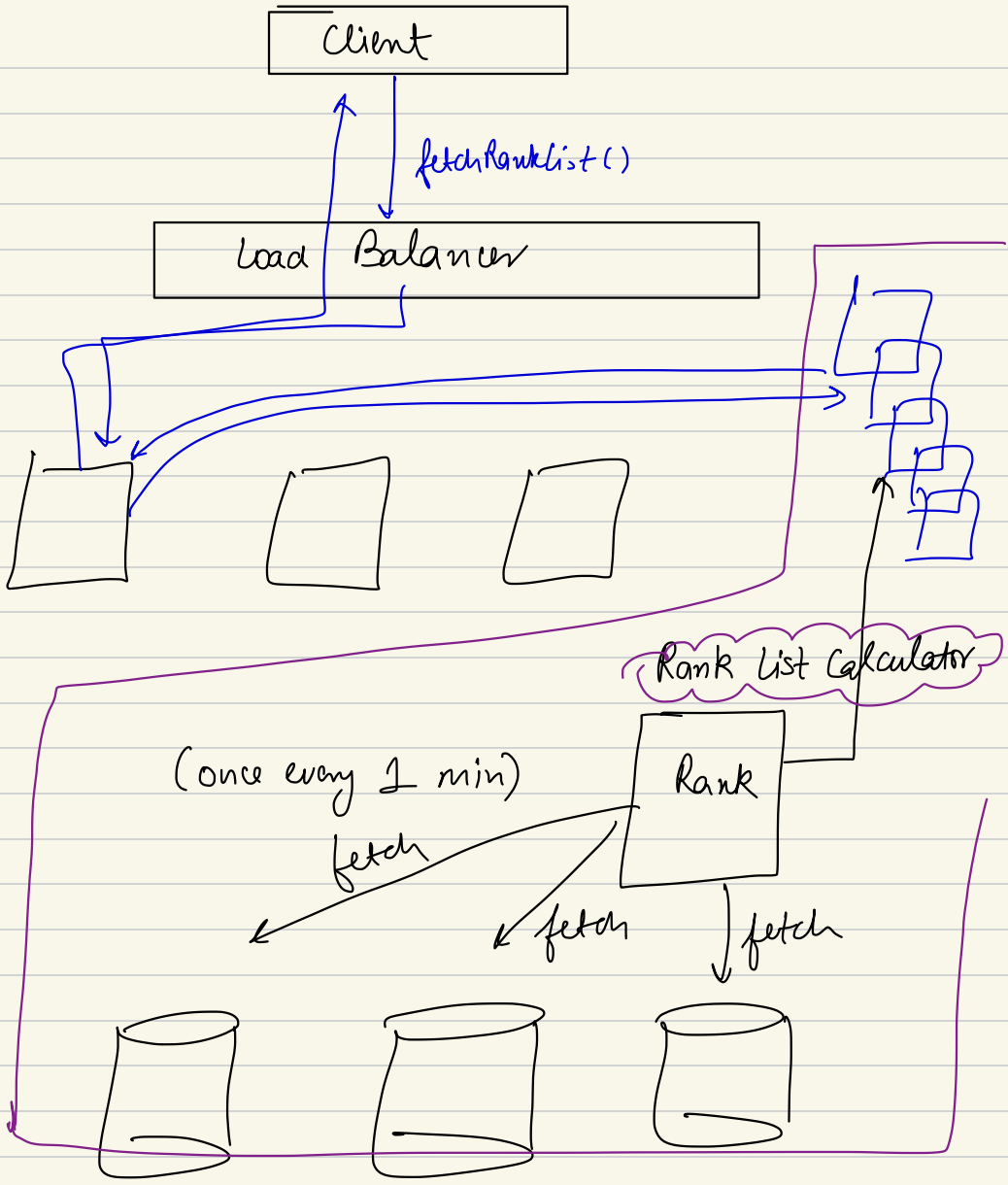
Ranklist of a contest



At every read ranklist query, collecting data and generating the ranklist afresh is a bad idea. This wont scale...

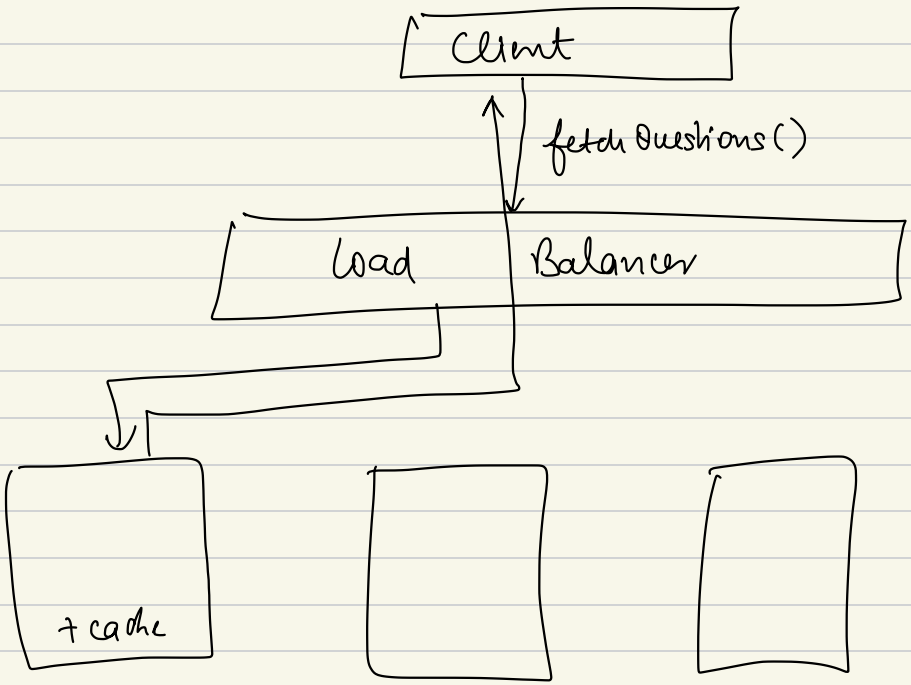

Client
Read Question ()
LB
+ cached
stateless

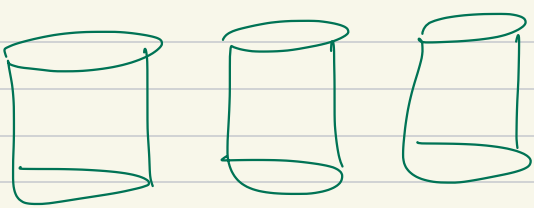

Client
Latency ↓↓↓
Read Ranklist()
Load Balancer
Update Ranklist in Global Cache
+cached
Scheduled Processing
Read scan of all contestants
Ranking
Once every 5 mins
(Cronjob)
1 min

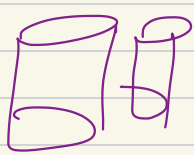

Client
fetchRankList()
Load Balancer
Rank List Calculator
(once every 1 min)
Rank
fetch
fetch
fetch

Code Evaluation

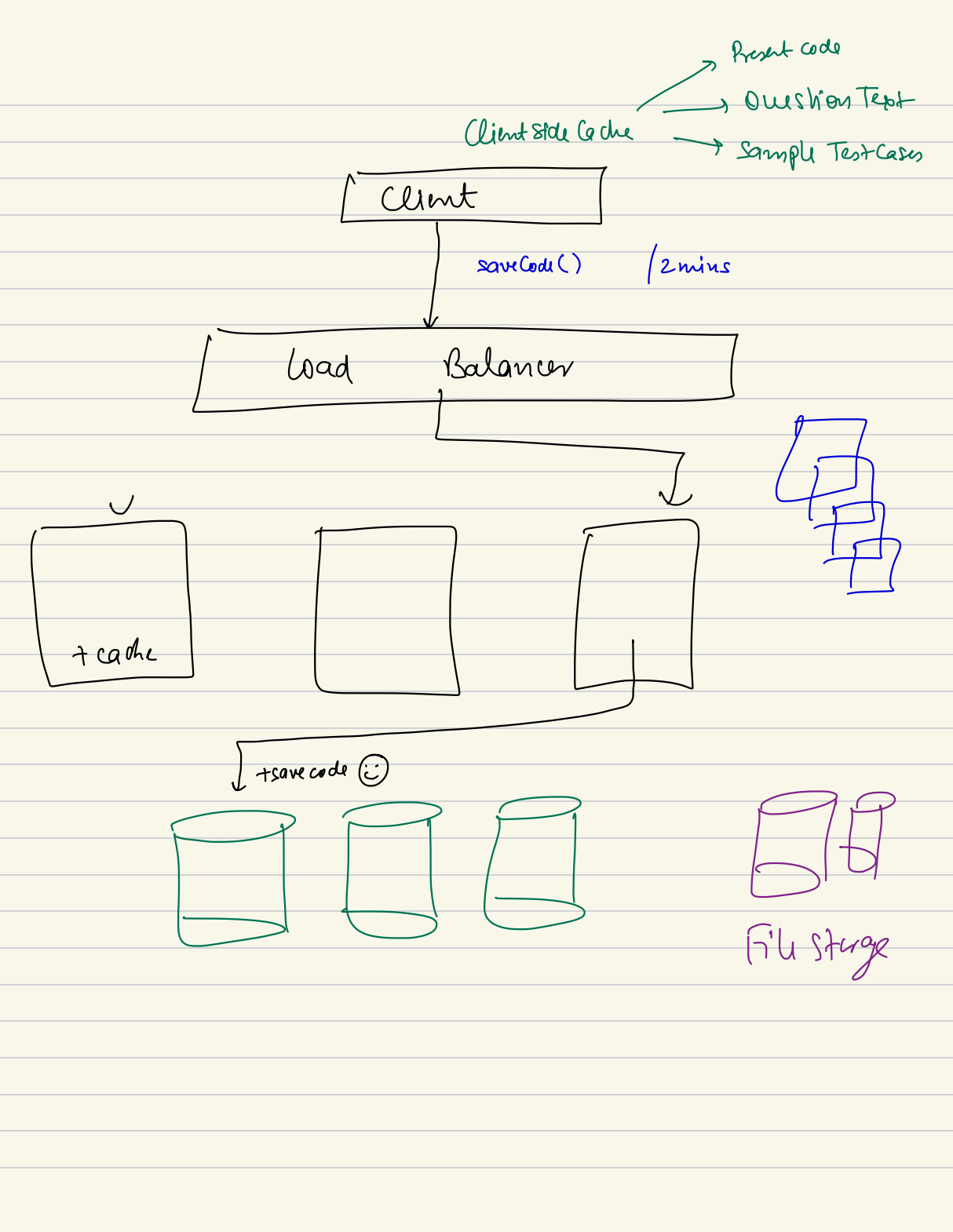
Contest System

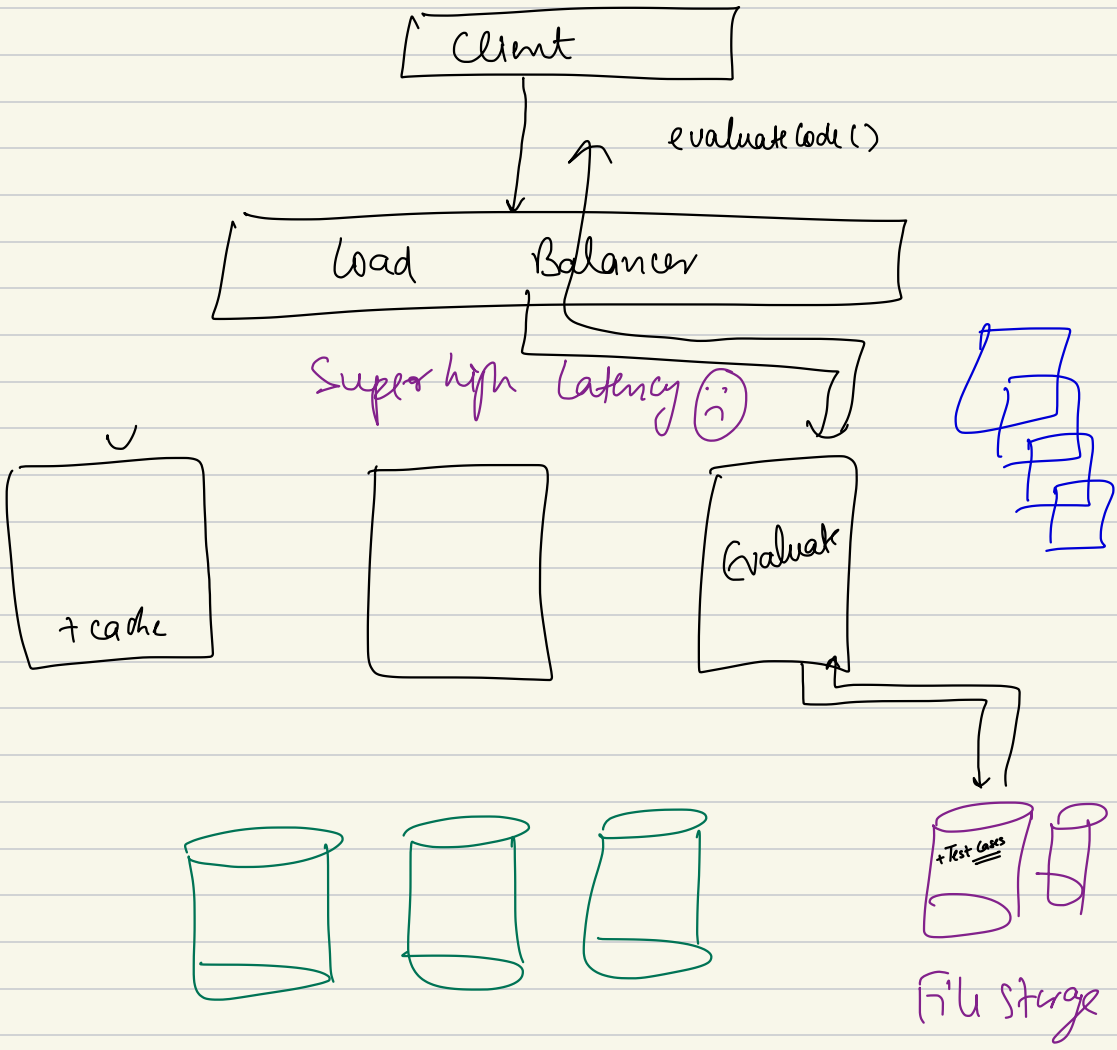




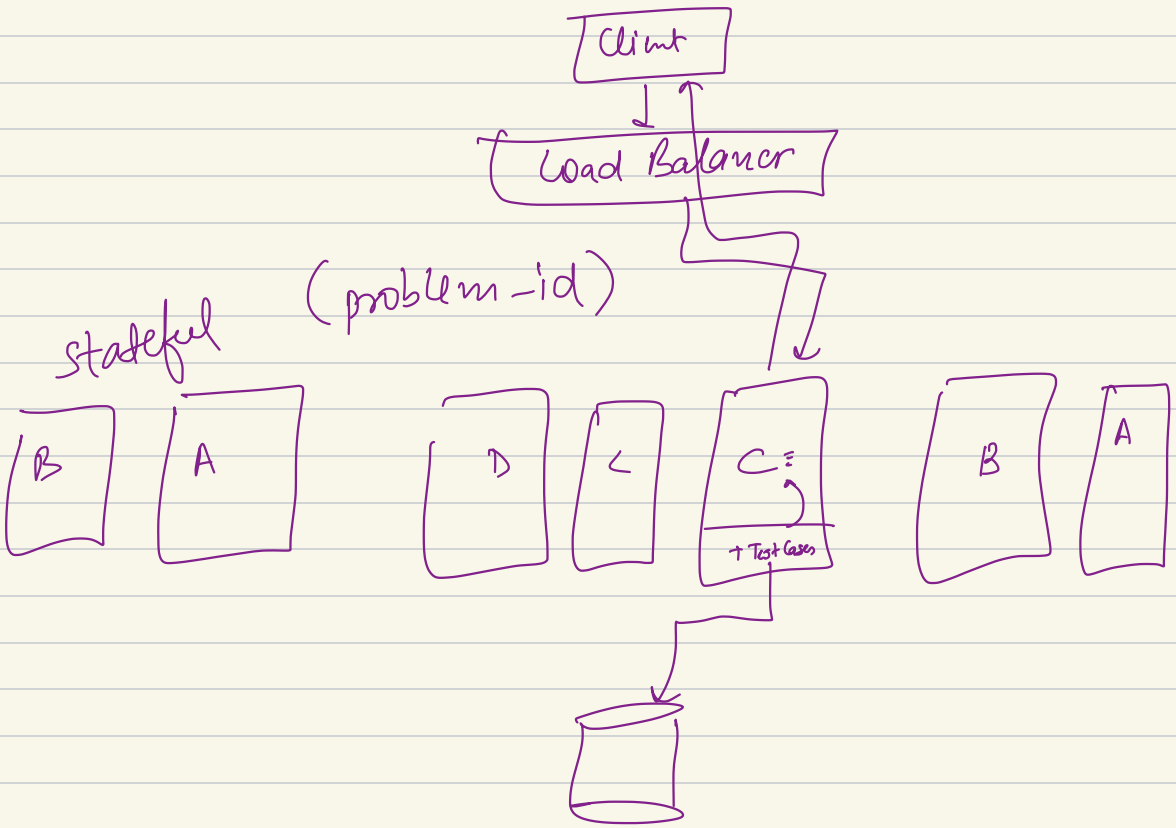


File storage


Recent code
Question Text
Client side Cache
Sample Test Cases
Client
saveCode()
/2mins
Load Balancer
+cache
+save code
File Storage

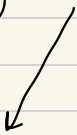

Client
evaluate(code)
Load Balancer
Super high Latency
+ cache
Evaluator
+ Test Cases
File Storage

Code Evaluation Service



Microservices

Big System



Evaluation Service

App Server → Stateful

Stateful Load Balancing

- Consistent Hashing (1)
- State Tracker System (2)