Agenda

Completing Theory of Caching

1. How to handle cache being full (Cache Eviction Policies)
2. Case Study: Scaler Question Submission Platform
3. Case Study: Scaler Contest Leaderboard Platform

Mini Case Studies

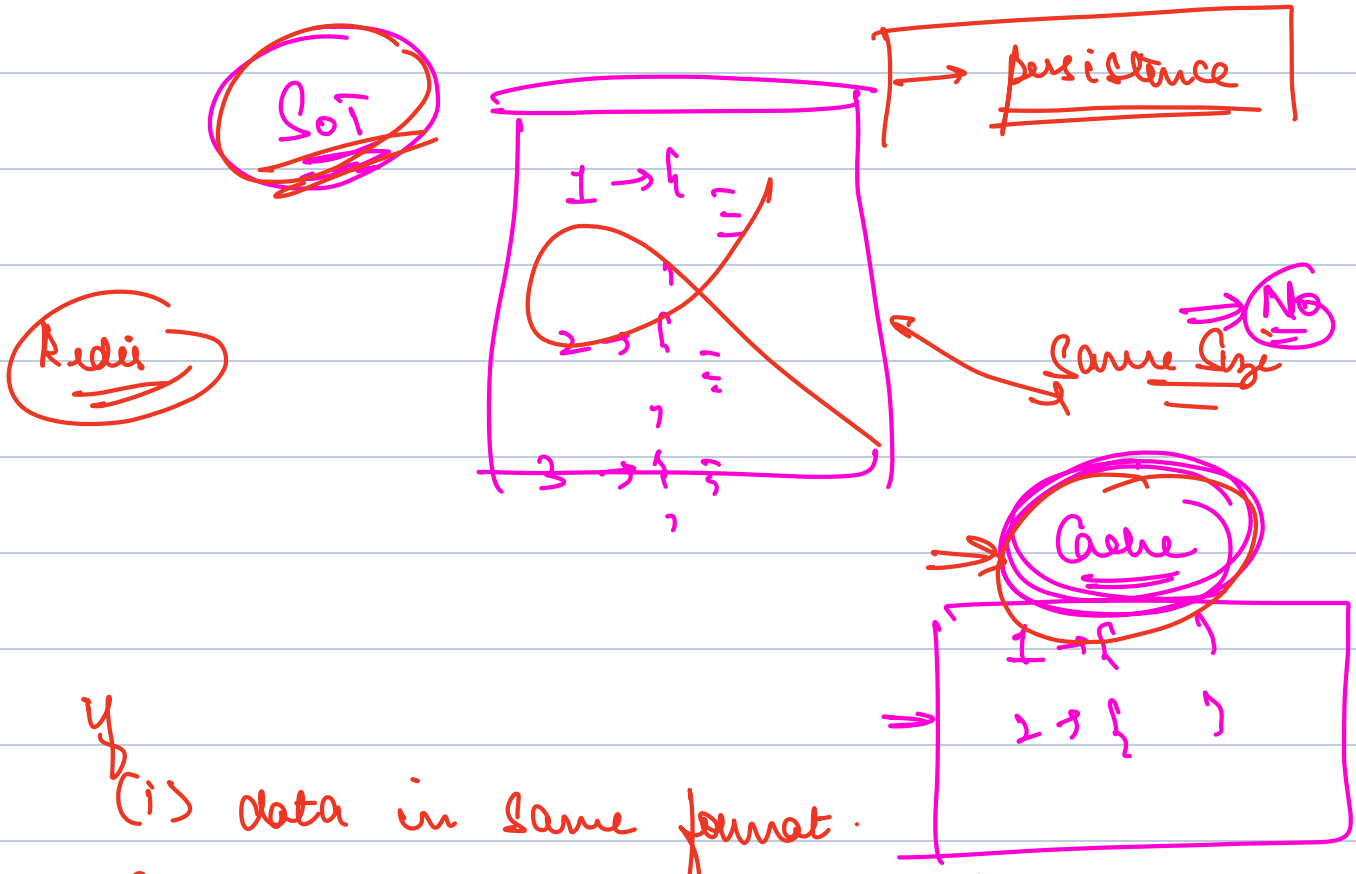
Require Caching to optimize

will start at 9:10 PM

Handle Cache being full

1. Act as a secondary place where we are storing data (faster to access than source of truth)
2. Store pre-processed data

⇒ we might decide to keep our cache machines as big as needed.

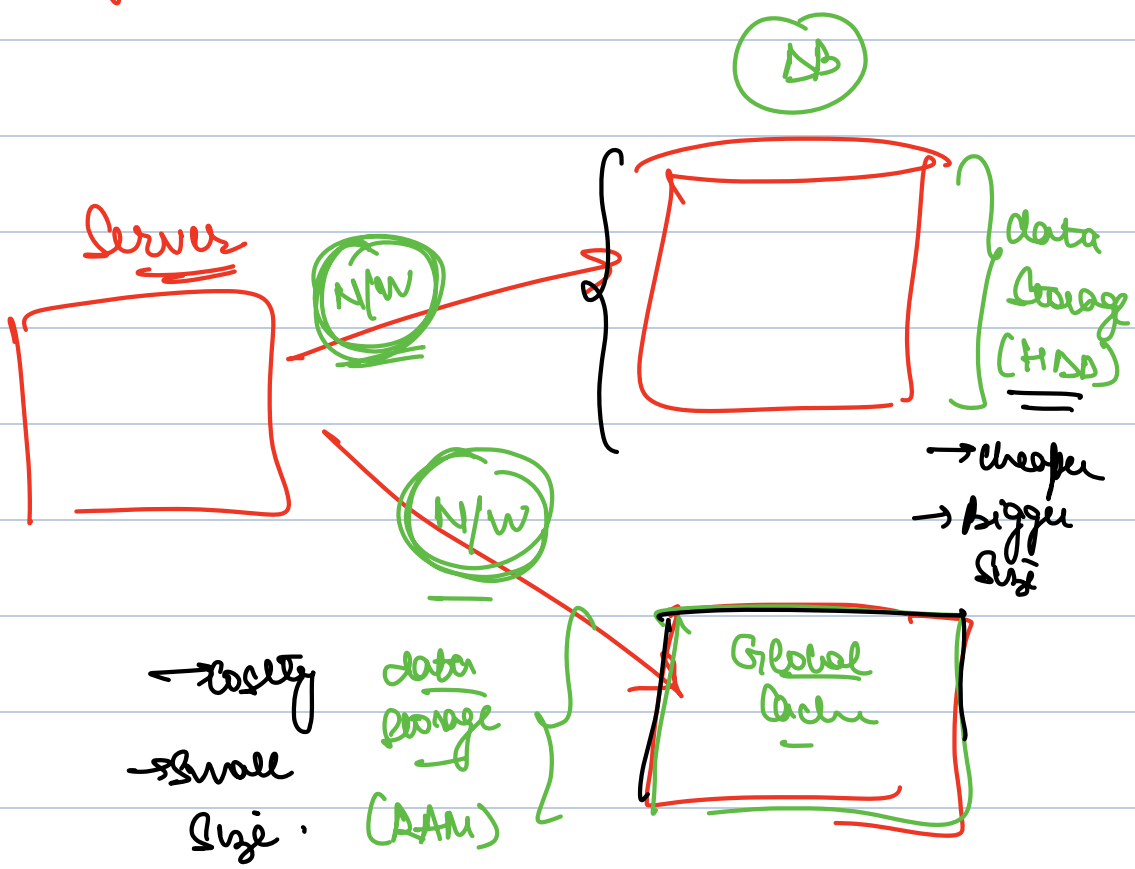


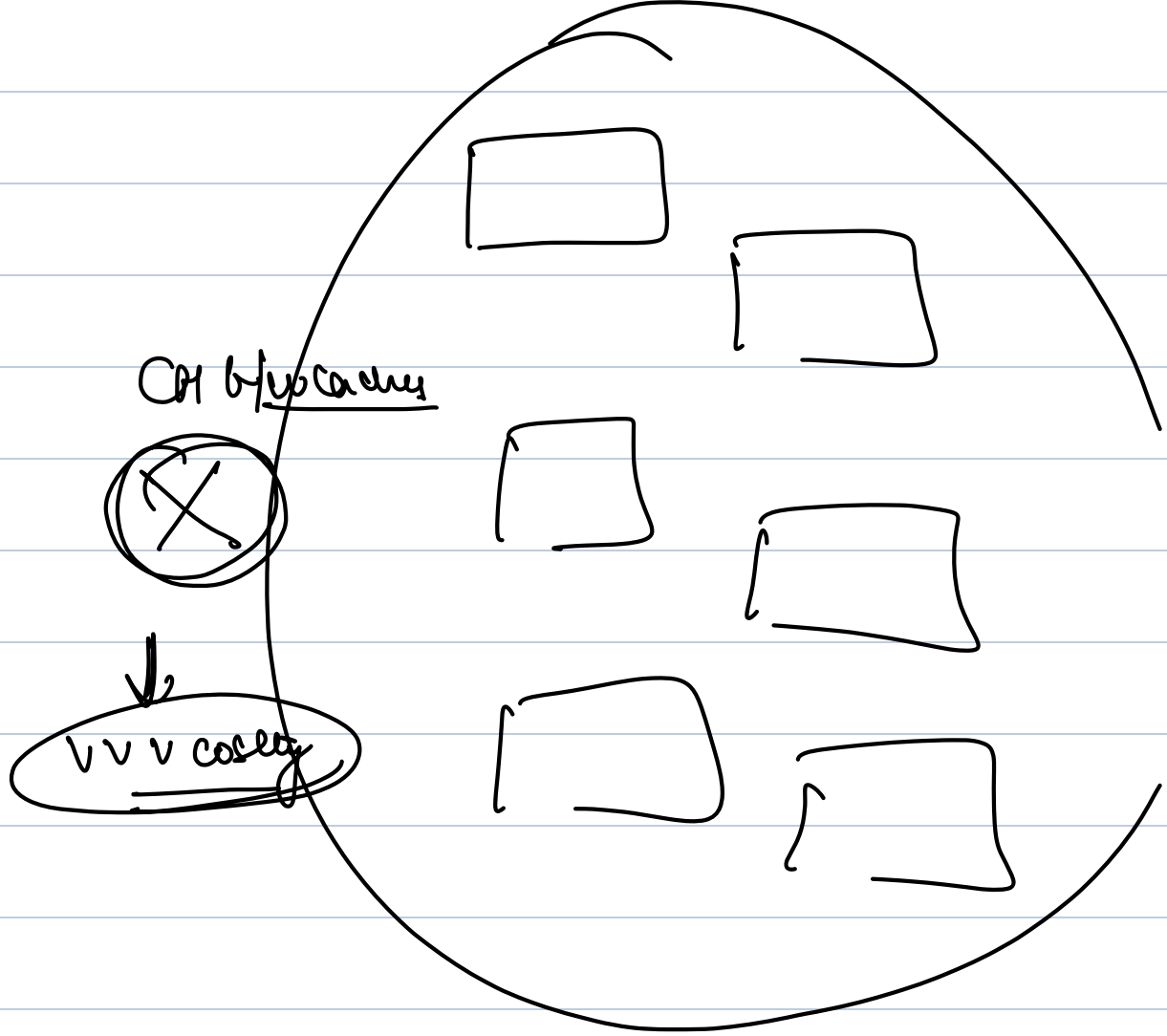
(i) data in same format.

(ii) one is faster to access ⇒ (faster to access)

⇓

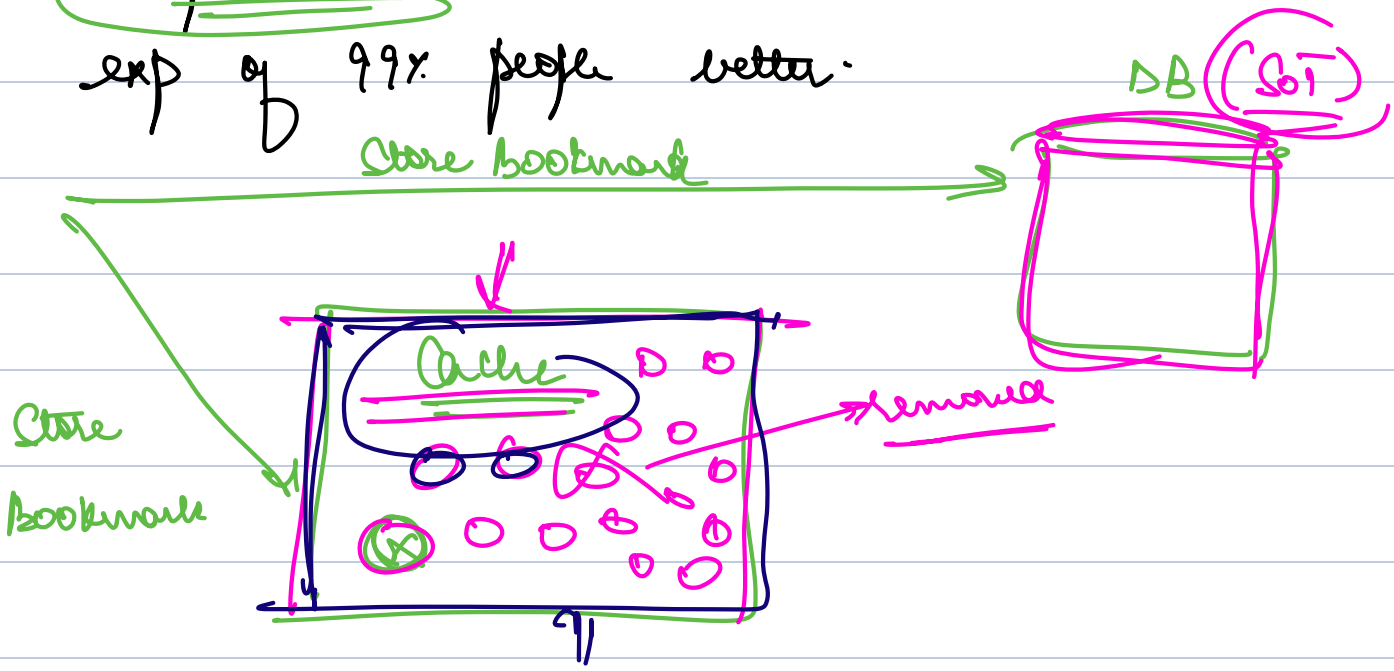
then why even keep other?





→ we don't store all the data in cache

→ we store data that we feel is important to be stored to make exp of 99% people better.



as I decided to store bookmark,
cache became full
⇒ I decide to remove one of the existing bookmark.

How do I decide what entry to remove?

→ CACHE EVICTION POLICIES

eg

Queues → ① FIFO (First In First Out)
↳ remove the entry that is existing in cache since longest amount of time

Default

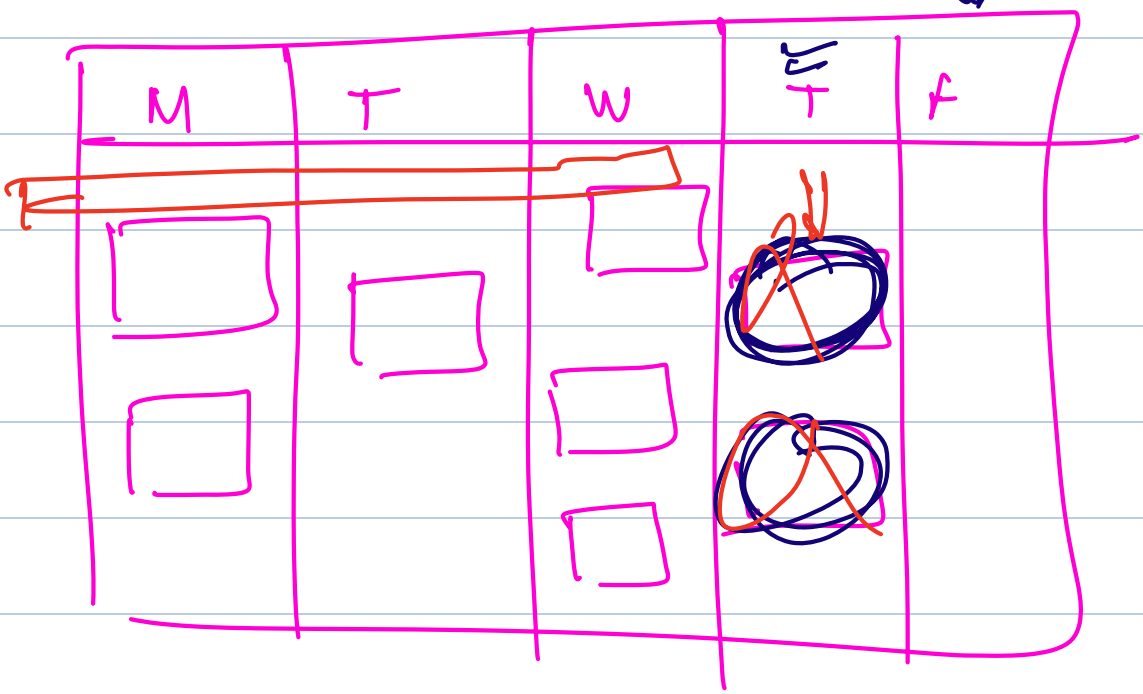
LU+HM

② LRU (Least Recently Used)
↳ remove the entry that has not been accessed since the longest amount of time

```
cache.write(k, v) {
    ⇒ if my size is full:
        → { Eviction Policy() }
```

```
{insert (k,v); }
}
```

- (3) MRU → Most Recently Used
- (4) LFU → Least Frequently Used



⇒ Reduce cache misses

↳ when you try to read data from cache

↓

Cache Miss → ↳ But cache doesn't have data

↳ So you have to go to DB

Slow

Cache.get() ⇒ null

Cache

Size 4

Eviction ⇒ FIFO / LRU

key accesses ⇒ 1, 2, 3, 1, 2, 4, 5, 7, 1, 2, 8, 1, 7

Cache Misses

Cache' → FIFO

⇒ | 5 | 7 | 1 | 2 |

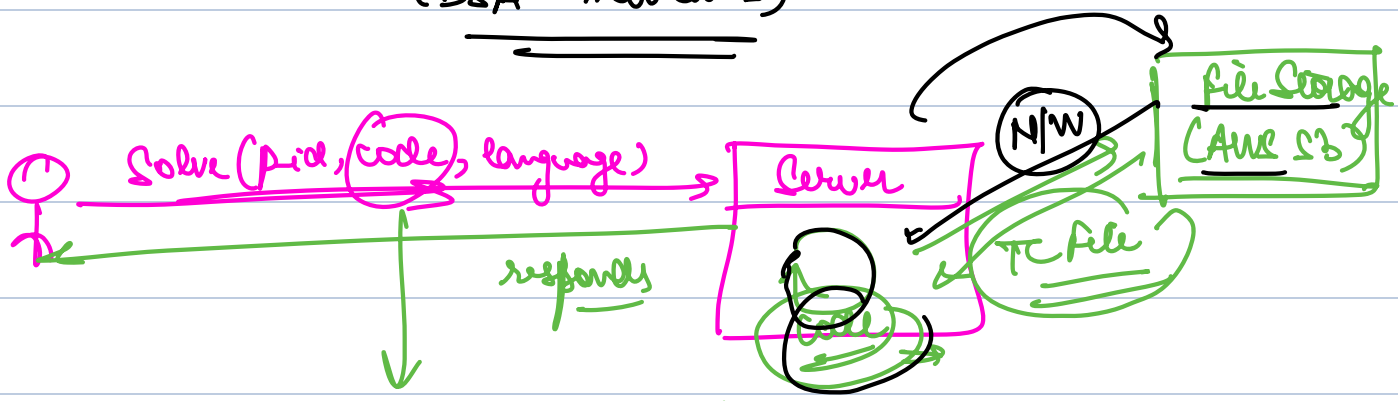
(5) →

Cache Misses

⇒ 9

Scaler Problem Solving Submission

(DSA Problems)



Multiple Test Cases

~100

(Big) ⇒

⇒ 10^6 elements in array

~~Array~~

N ⇒ 10^6

10^6 integers

⇒ $4B \times 10^6$

⇒ 4MB ≈ 10MB

TC File of 1 Prob ⇒ 100 MB

CodeAgon

80K people who solve problems

$80 \times 10^3 \times 5 \times 3$ ⇒ 24hr contests

1200×10^3

⇒ 1.2M submission ⇒ 24hrs

$$\frac{1.2 \times 10^6}{120 \wedge 100} \Rightarrow \frac{1.2 \times 10^{\cancel{12}\,6}}{\cancel{1.2 \times 10^{\cancel{4}\,1}}}$$

$\Rightarrow$ 100 submissions / sec

1000 submissions / sec

1B

100 $\Rightarrow$ 3B

$A_i \Rightarrow 10^9$

1000000000

9 chars

$\Rightarrow$ 9B

array

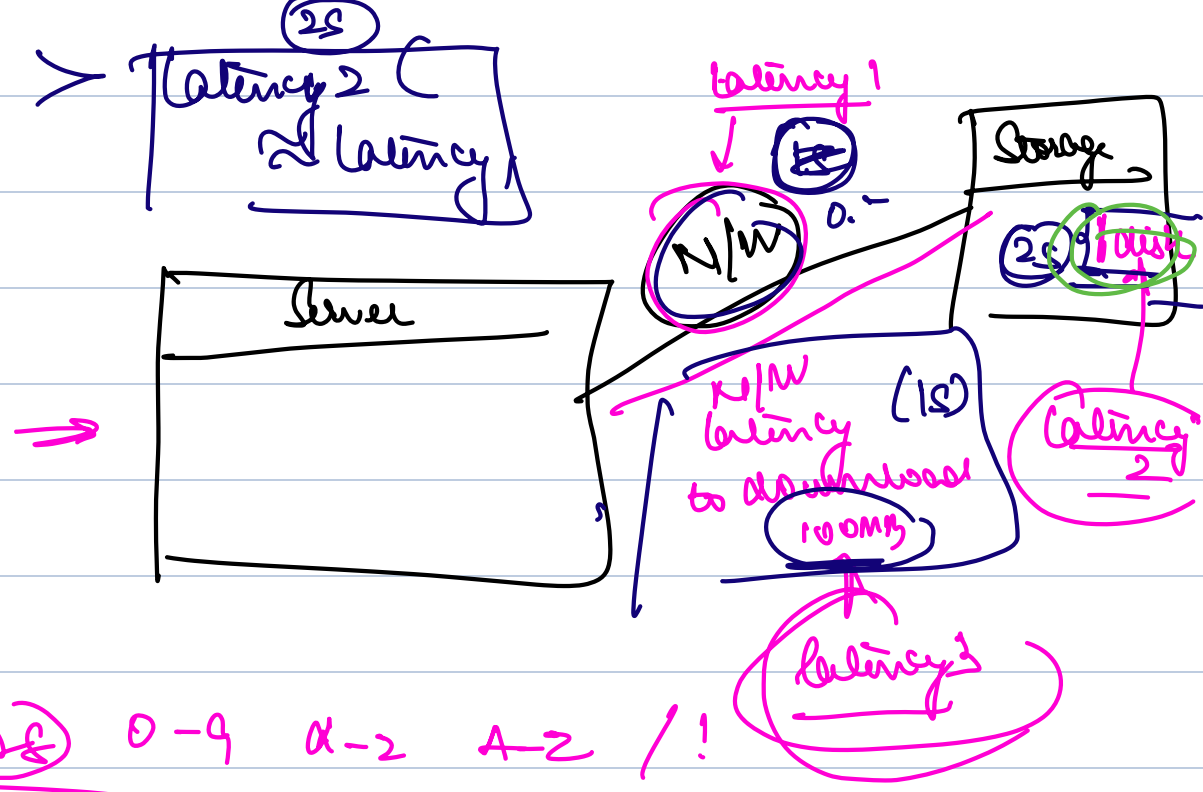
$N \Rightarrow 10^6$

$A_i \Rightarrow -10^9 \longrightarrow 10^9$

$A_i \Rightarrow$ 1000000009

10 chars $\Rightarrow$ 10B

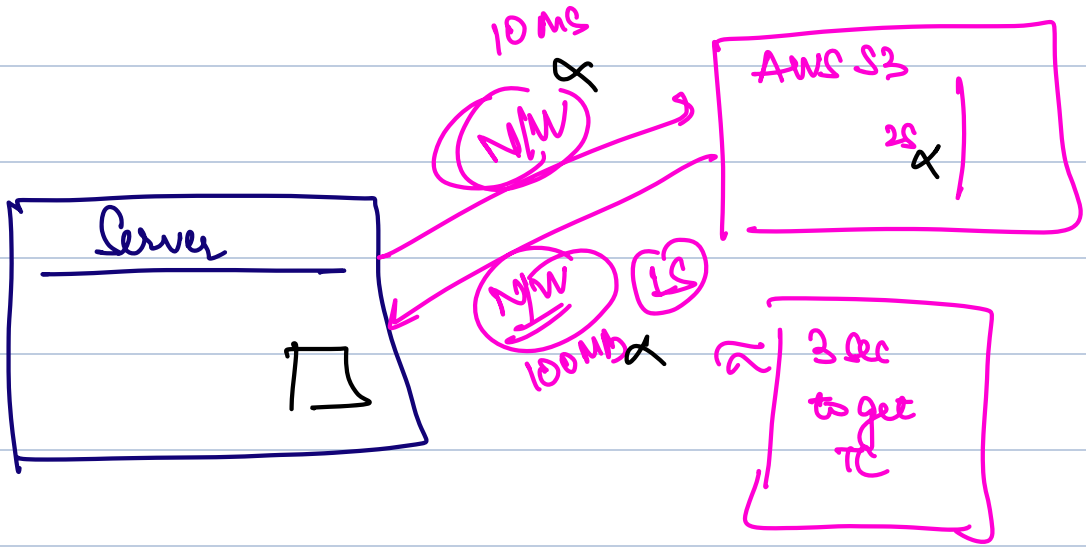
Latency 3 > Latency 2 > Latency 1



1 ASCII (1B) 0-9 a-z A-Z / !

Unicode → Emojis

latency 3 > latency 2 > latency 1



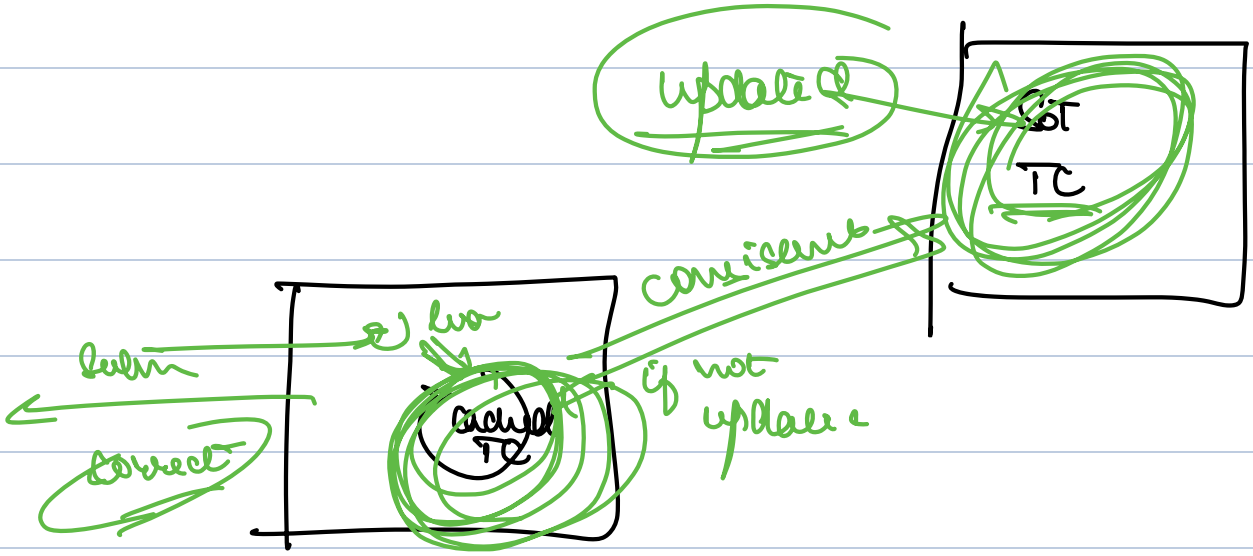
Why can't I cache the files on the server

Thought ⇒ cache test cases

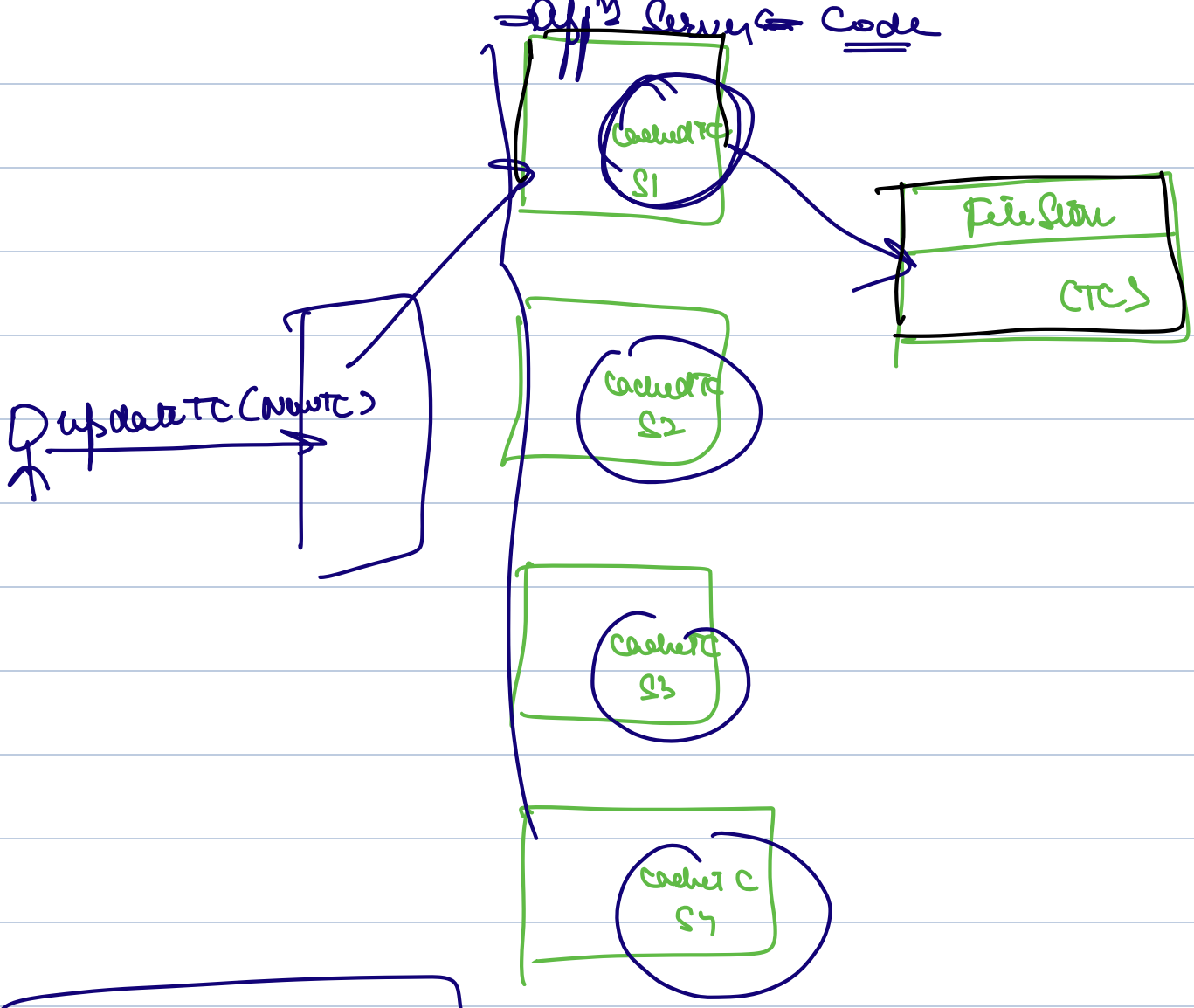
- TC may change (rare but can happen)
- when TC have been updated everyone should be evaluated against new TC

⇓

consistent

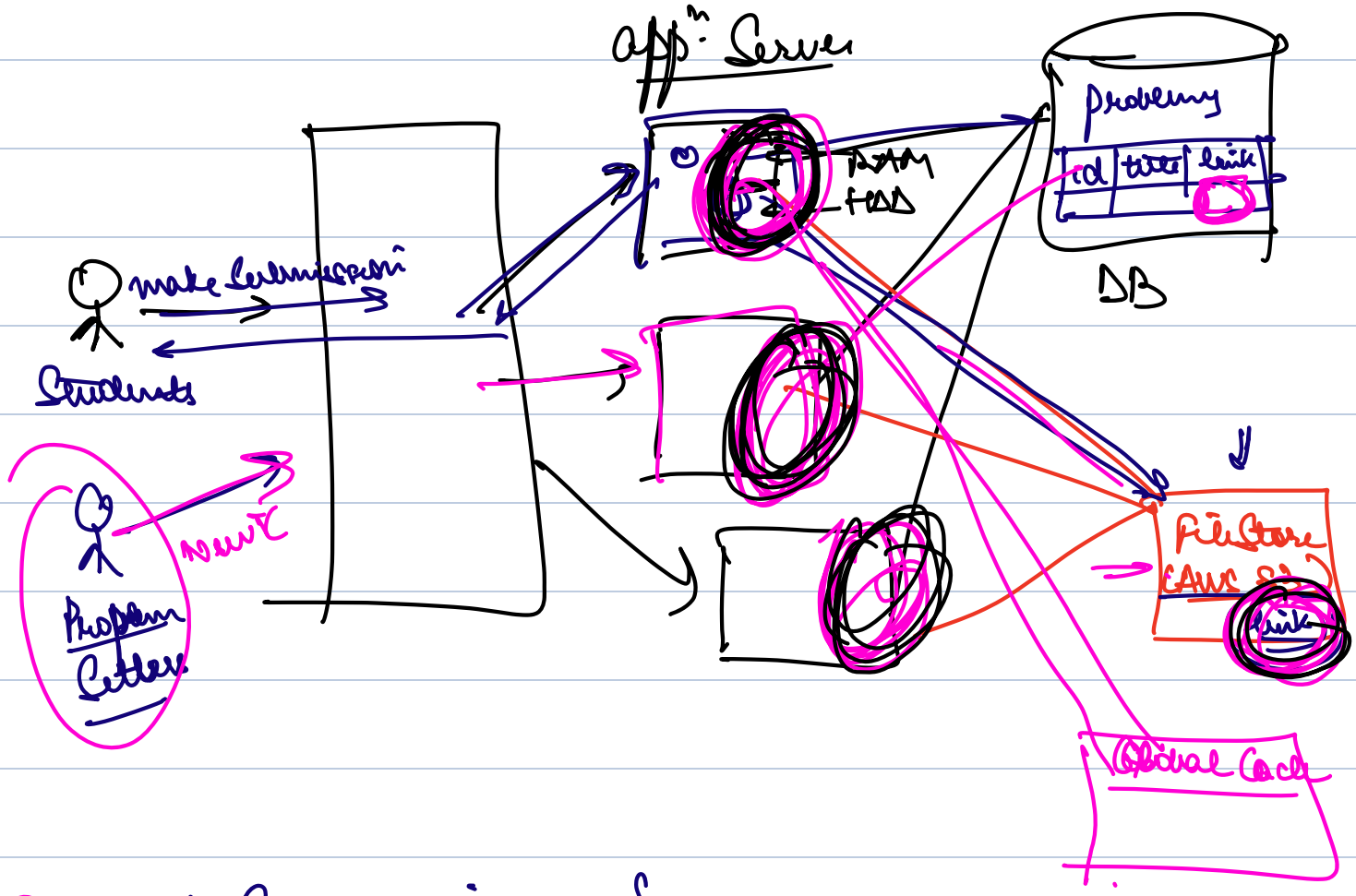


How? =



WTC ⇒ global Cache

TTL Approa



handleSubmission() {

① check if TC of that problem in

② if no:

go to DB

get the link to TC

download TC from that link

cache it on my machine

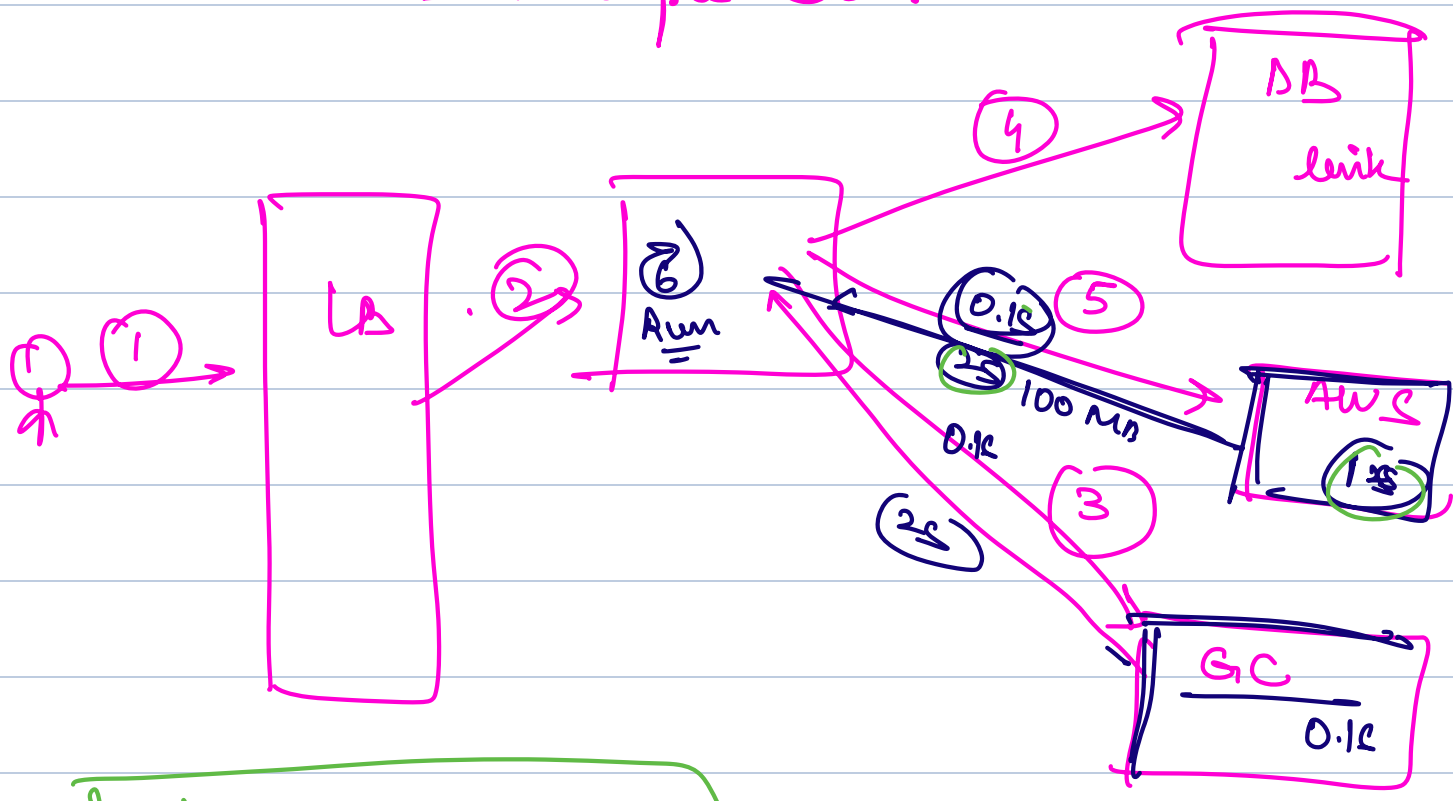
③ evaluate Submission()

}

Global Cache

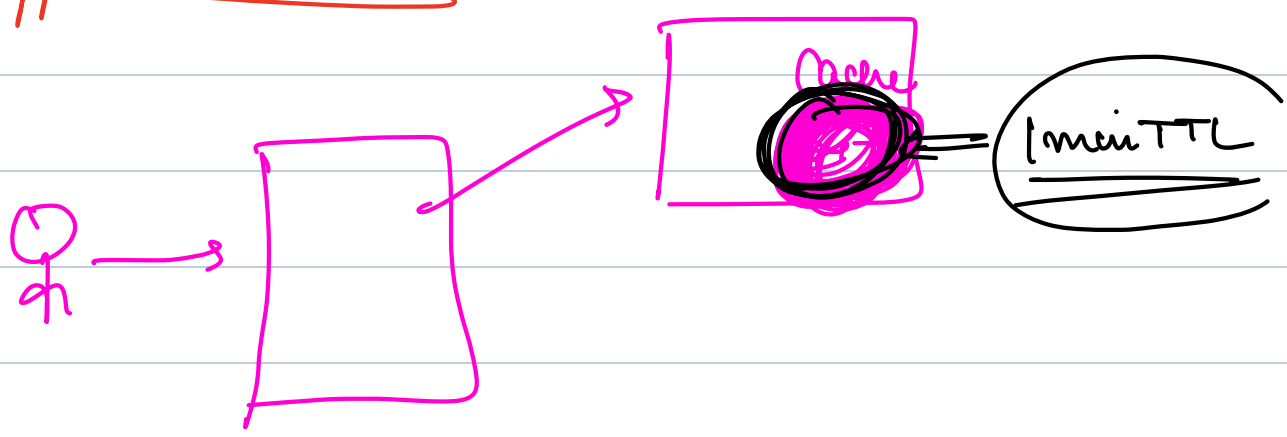
isn't really helping make our sys much fast

→ if we are caching in appⁿ servers, we can't do W.T.C as multiple caches



3.1 sec v/s 2.2 sec

Application Server

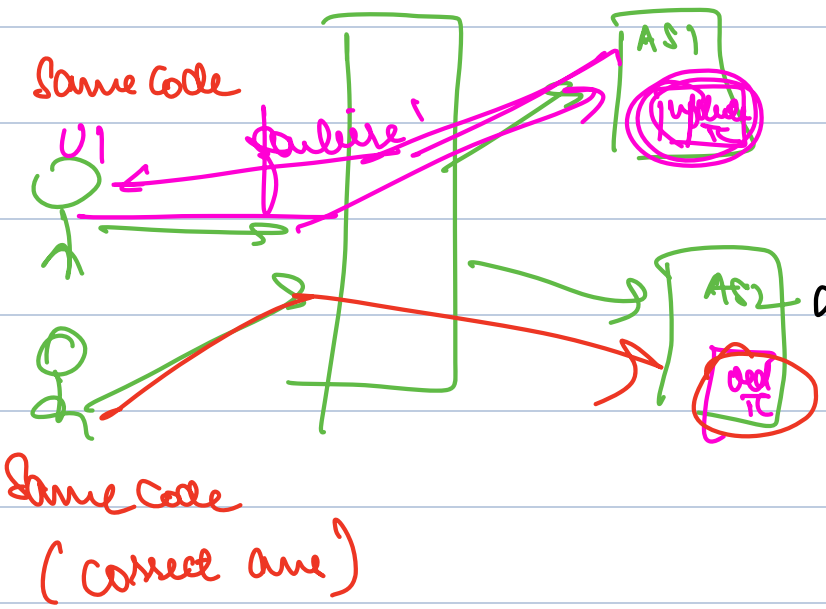


TTL (Time to Live Approach)

→ whenever I store TC in my cache I set TTL of 1 min

→ after 1 min, regardless of whether TC have been updated or not I refetch the TC from F.S.

~ Still inconsistent for 1 min

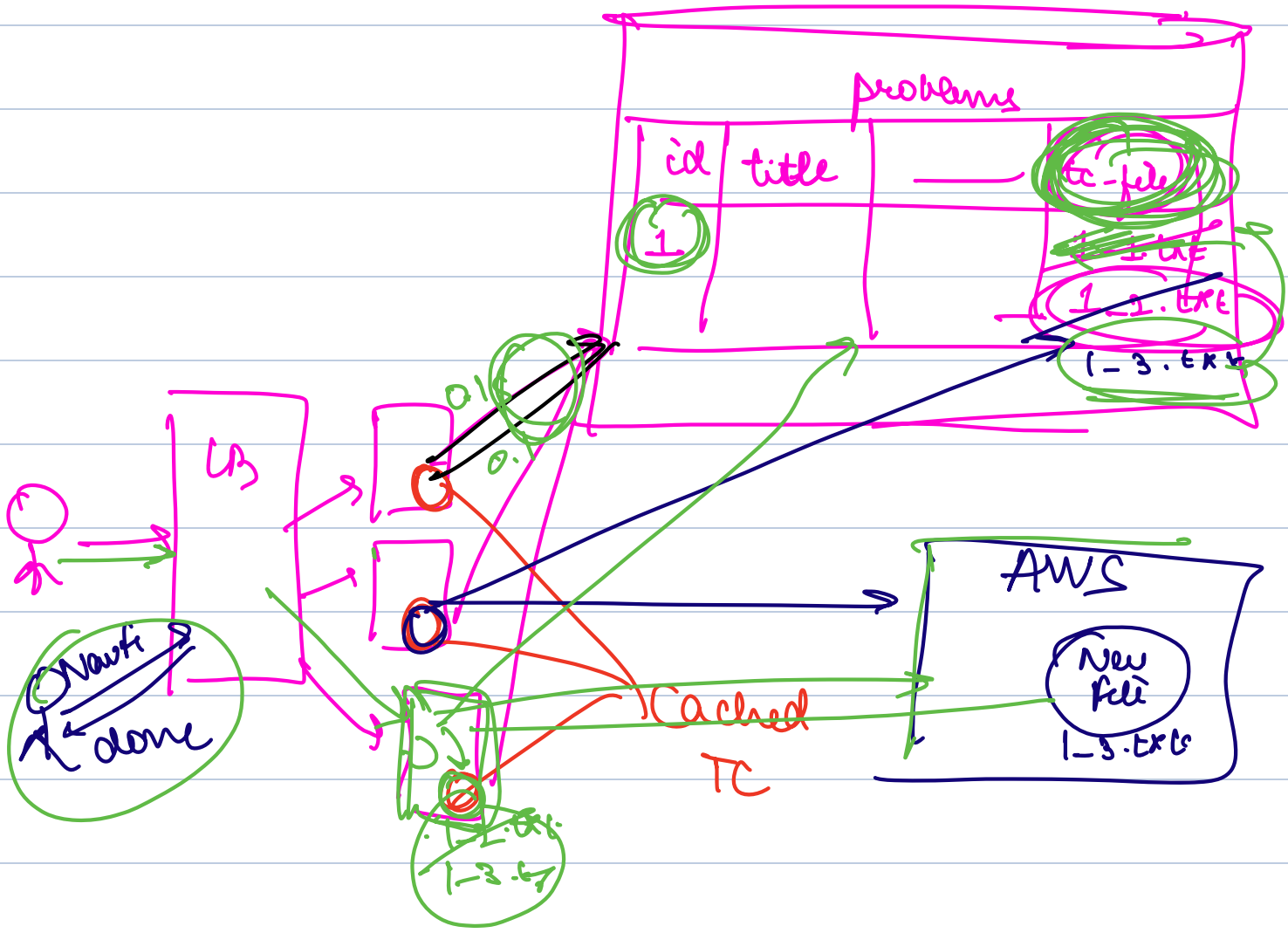


As soon as TC are updated every submission should be evaluated with latest TC

KISS

Keep it Simple!!

Auxiliary Info to identify if something has changed



↳ we store new TC with a diff fileName

evaluate Problem(id, code, lang) {

fName = go To DB and get TC file name of that prob.

if fName exists in my machine:
    done:

else:
    → go to AWS and download that file.
    → cache that file in me

evaluate Problem()

}

App^n Server

Consistent H/S

0.2 sec 1/S 2.2 sec 4/S 3.1 sec

1.5 sec

In app^n Server, where to cache

→ disk =

→ RAM

→ consider

4/5 problem

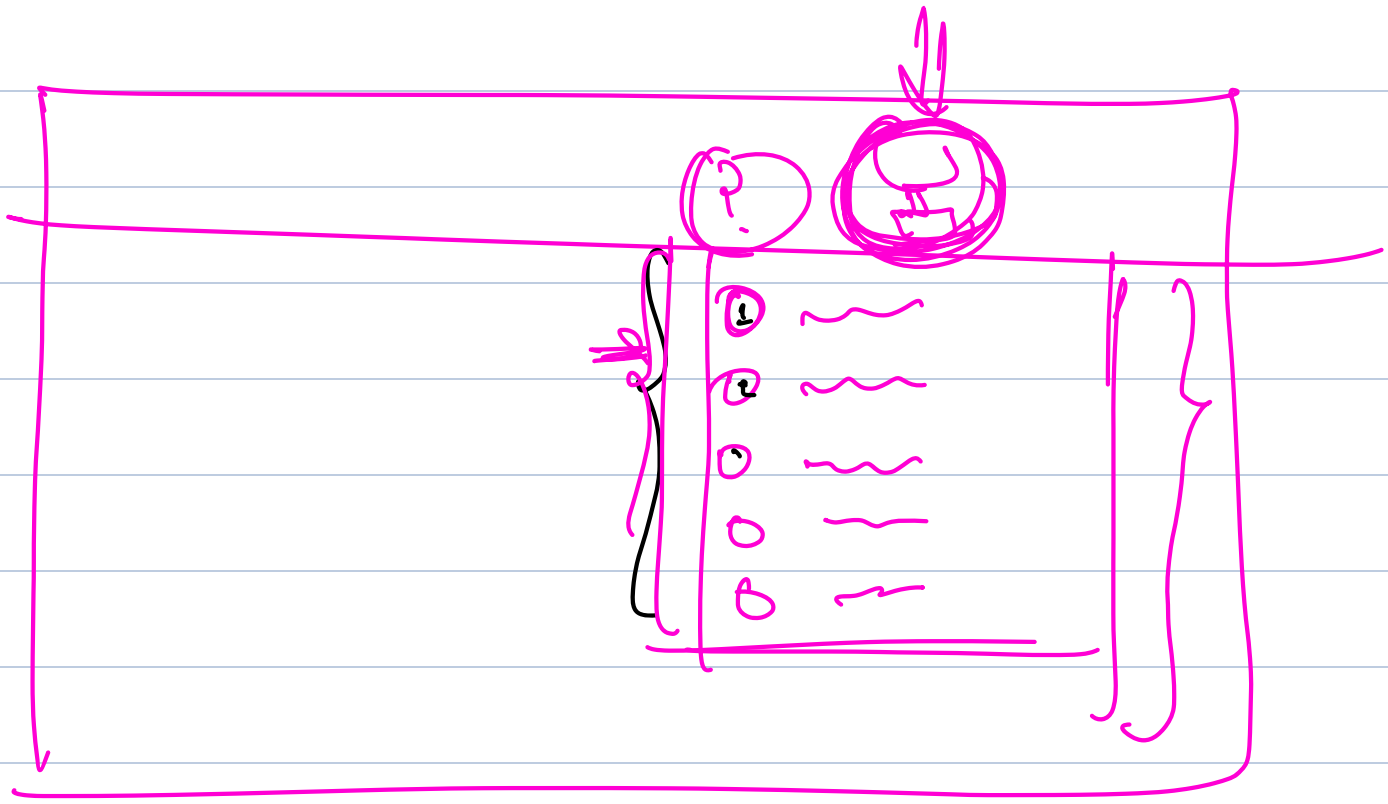
100 * 5 * 100 MB → disk

500 * 100

50 000 MB

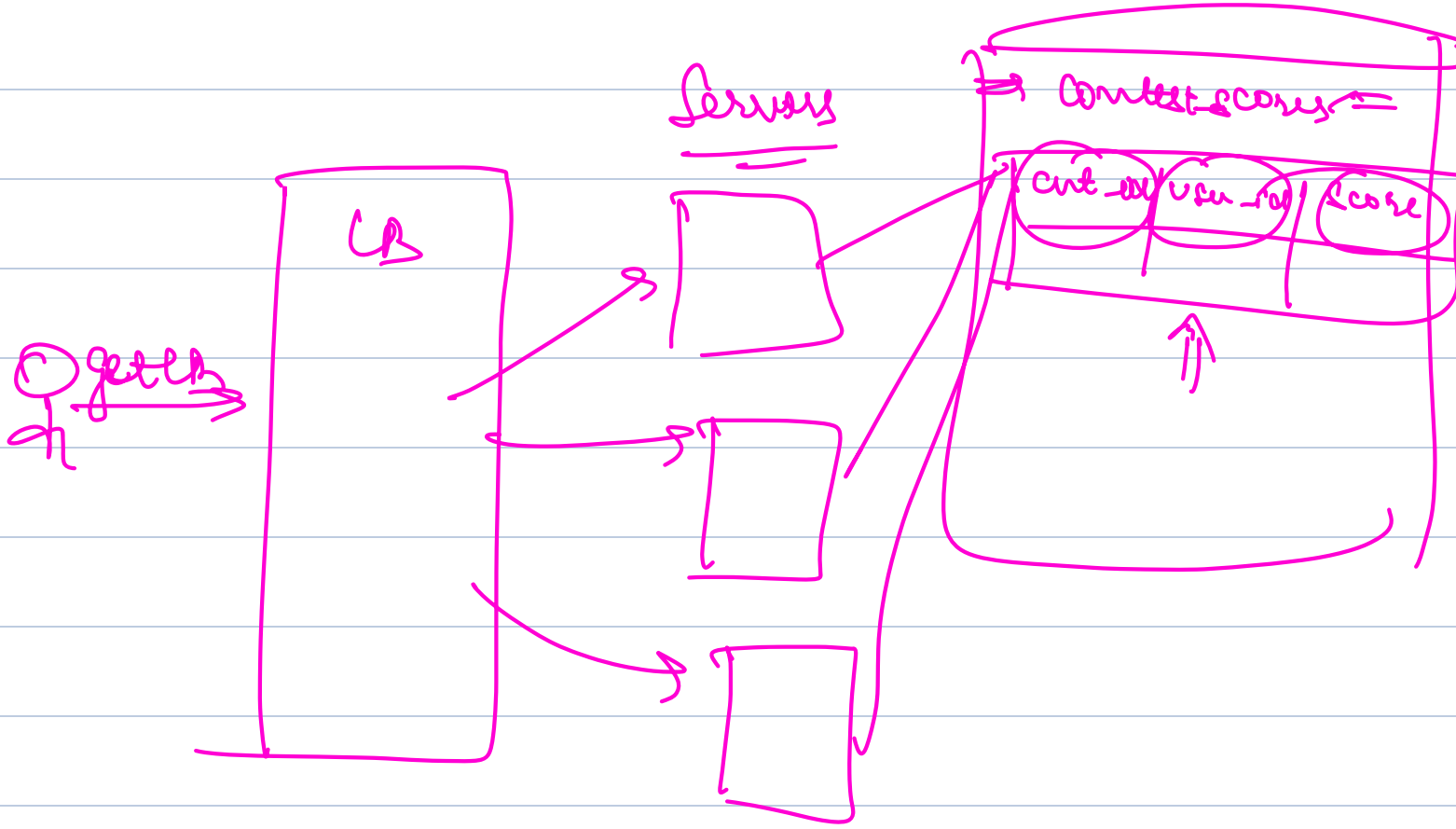
→ 50 GB

Case Study : Scale Contest Leaderboard



- 50K participants
- every submission may change leaderboard
- 1000 times / sec

how will we show LB




```
select *
from contest-score cs
    join users u
    on cs.user-id = u.id
where contest_id = { }
order by score desc
```

$O(N \log N)$

Put Huge Load on DB

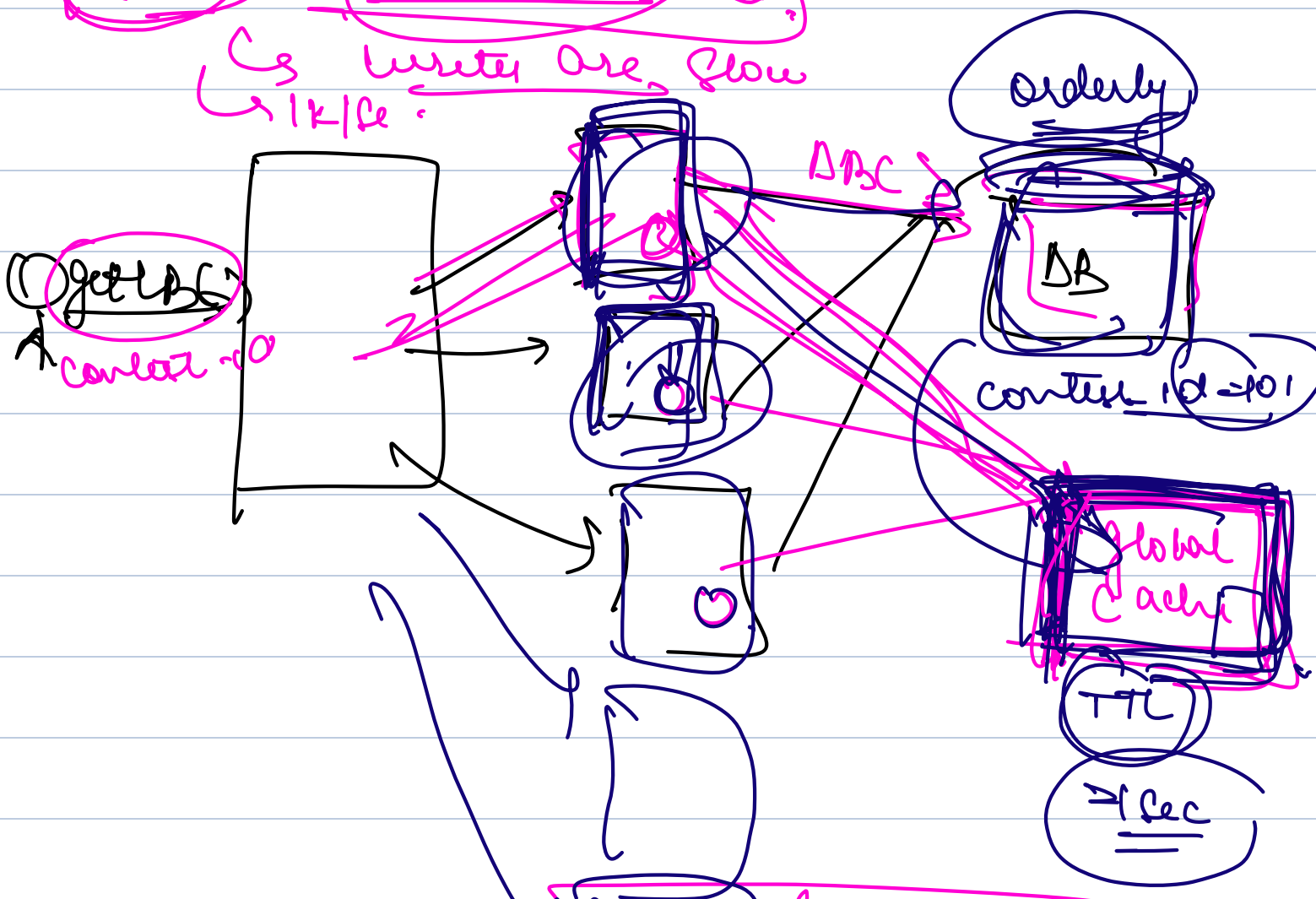
⇒ It is okay to have LB delayed By 1min

WTC ⇒ consistent ✗

TTL

↳ writes are slow

↳ 1K/sec



1min

Context-id	LB
1	[0, 0, 0, 0]

≈ 10–20sec

WTC → TTL

	DB Queri
appⁿ caching and	# Servers/min
global caching	1/min