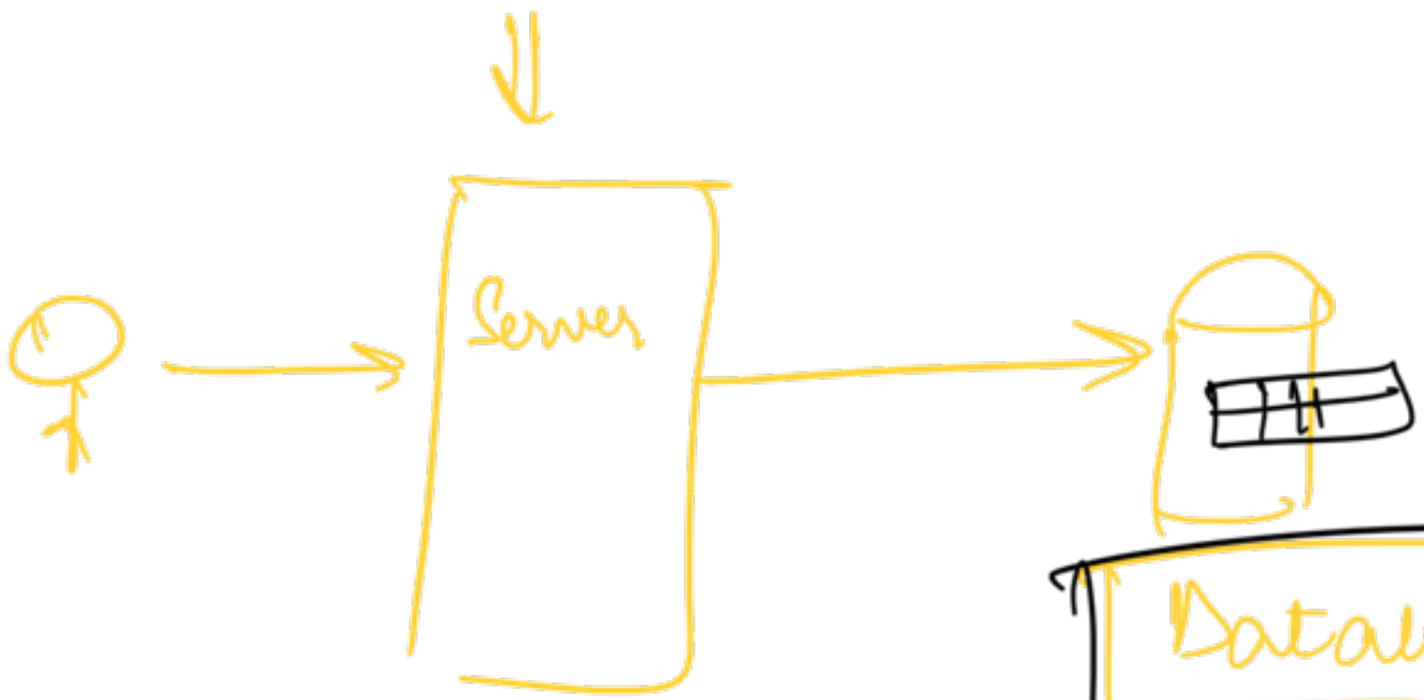
Design a Distributed Cache



Disks =

(over the file system)

Caches ⇒ temporary storage

Make computations more efficient

Browser Cache

| google.com | 192.168.1.1 |

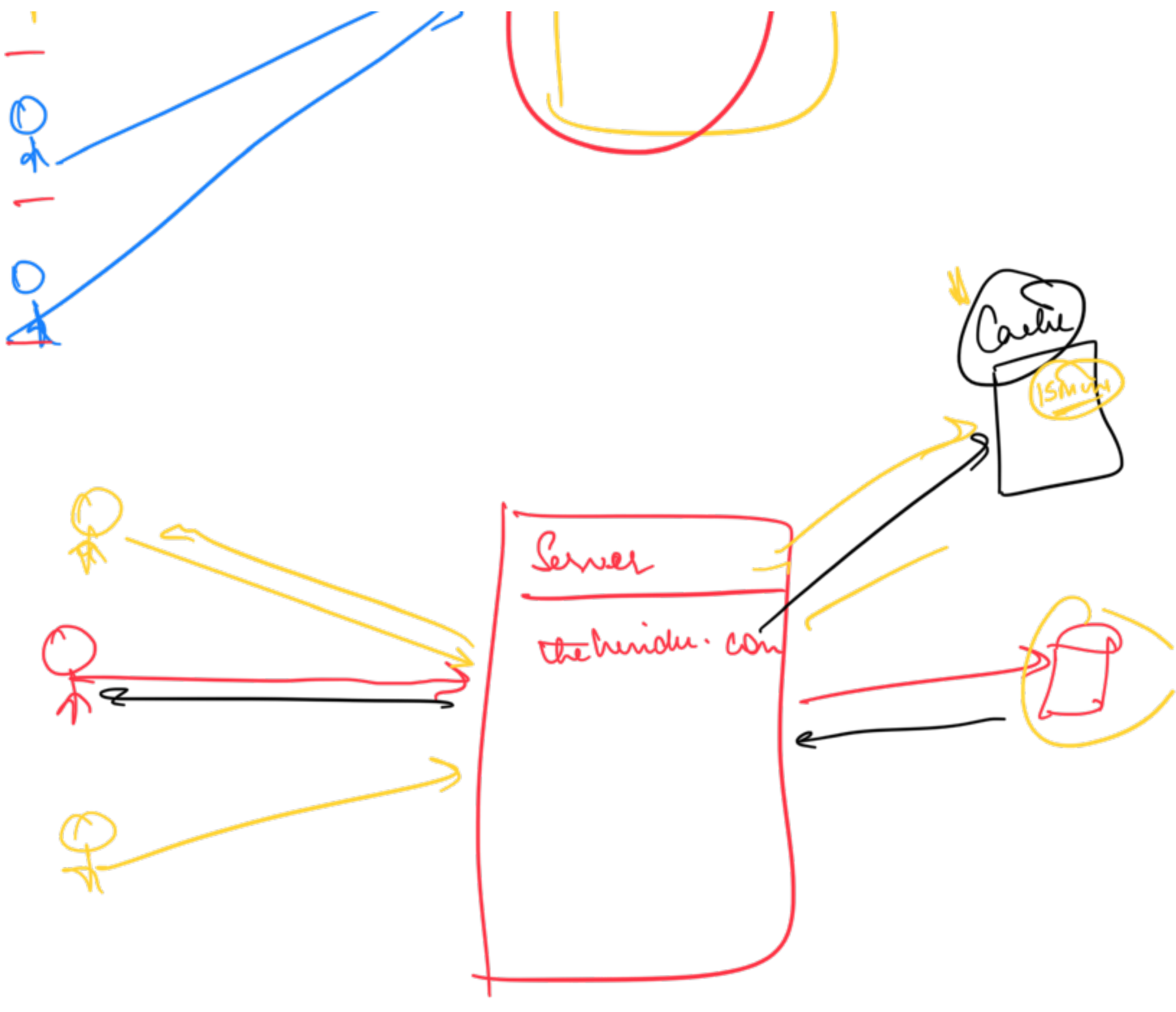
DNS Server

IP

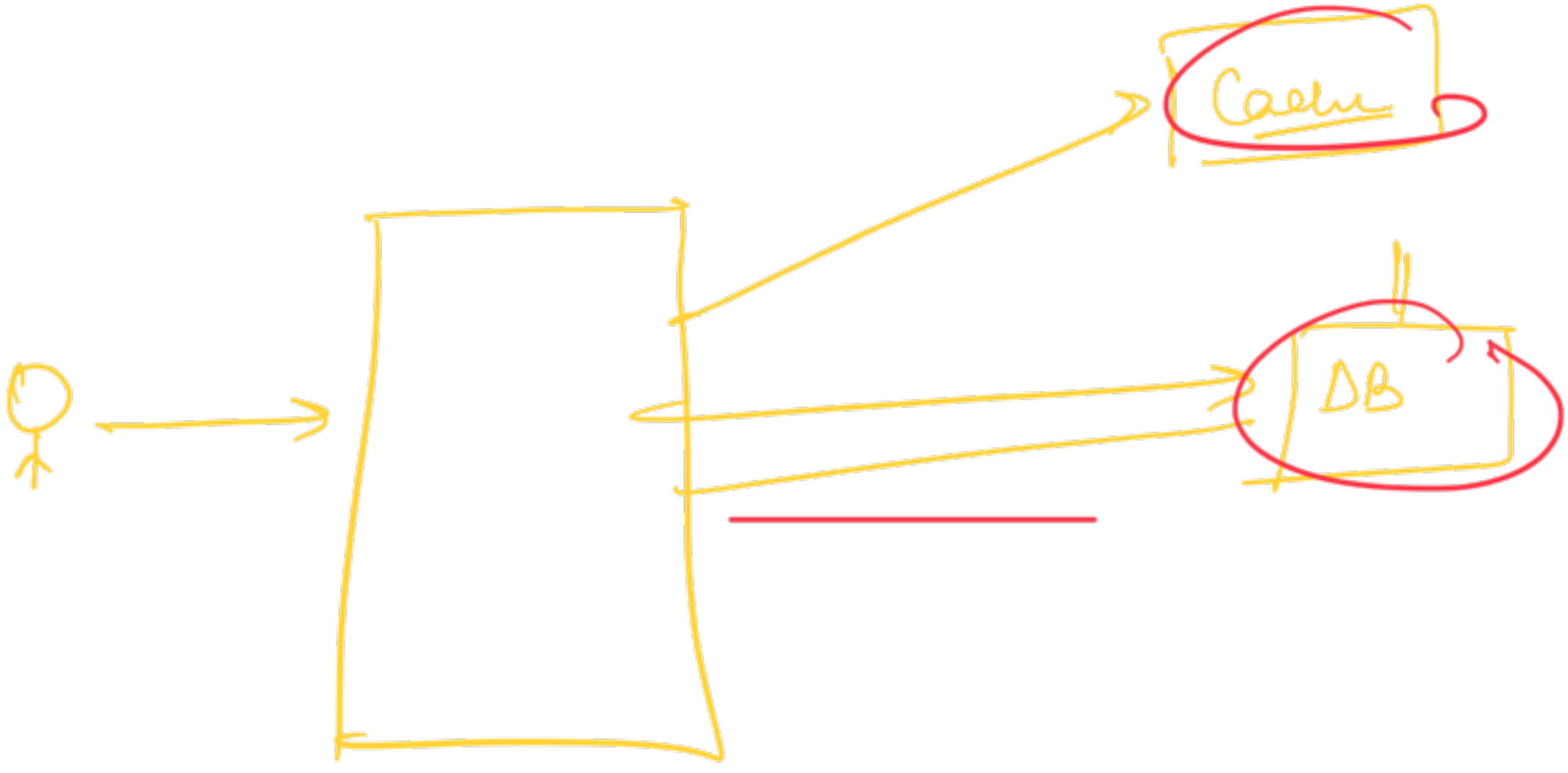
Website

google.com

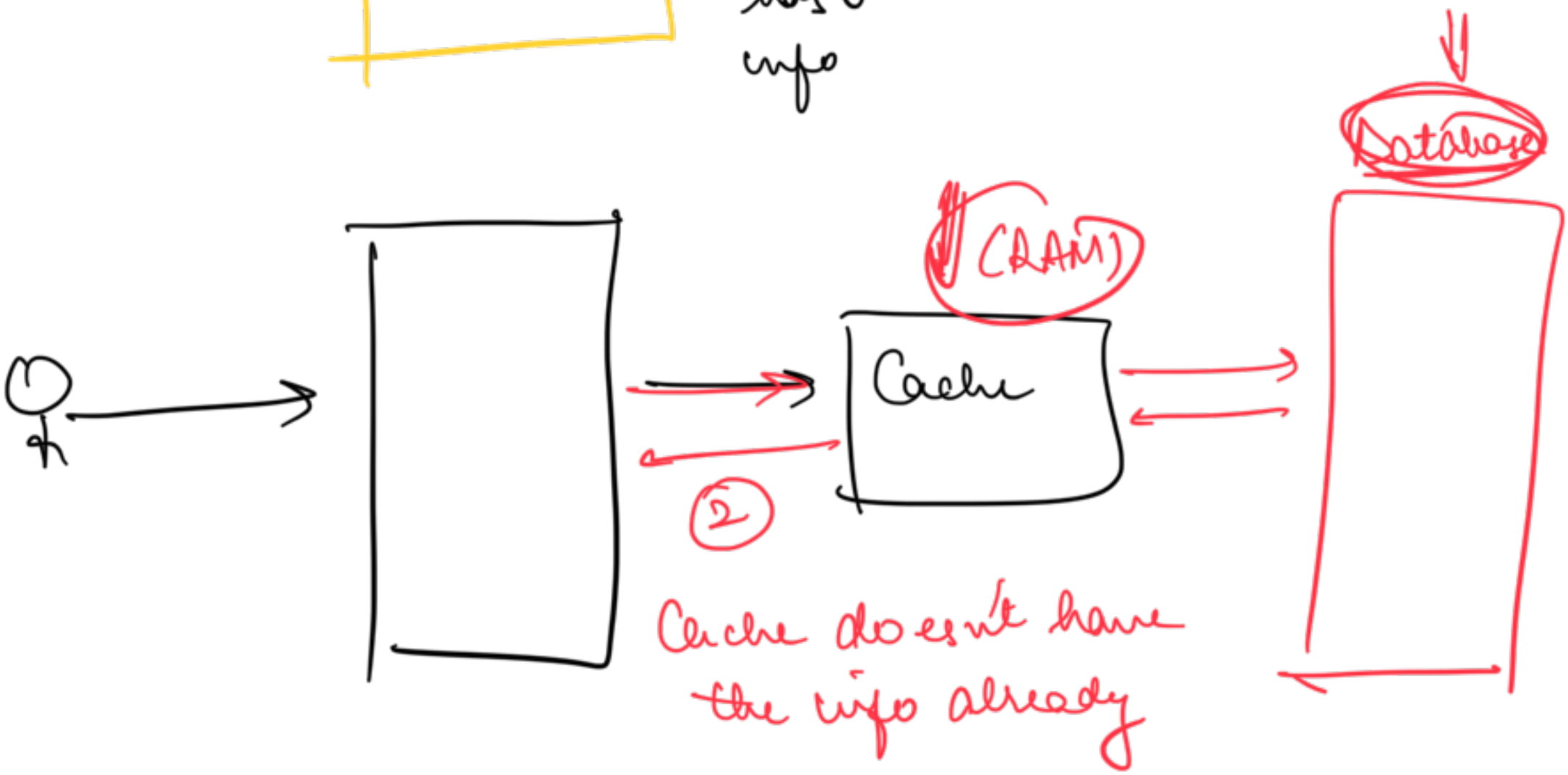
IP address of Google


Server
thehindu.com

Distributed Cache




Cache
1
already has info


Database
(RAM)
Cache
2
Cache doesn't have the info already

RAM

16GB

(Expensive)

low storage

large ⇒ high mem
medium cpu

SSD/HD

512 GB

(Cheap)

high storage
low cpu
low mem

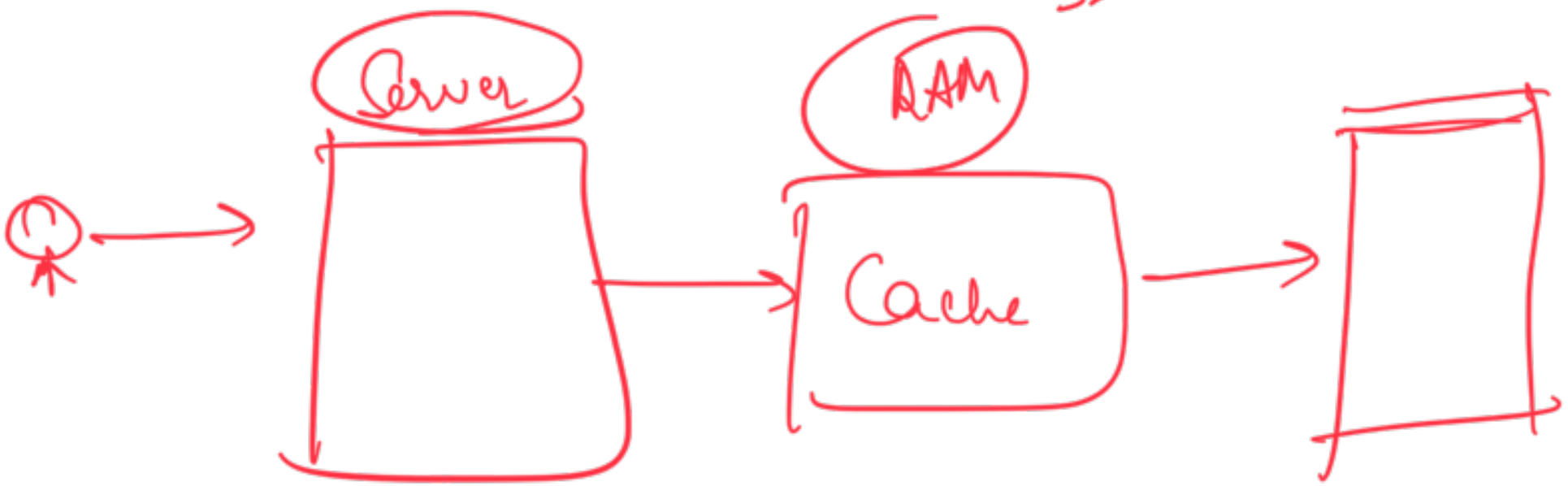
portals

VM

RAM ⇒ 512GB

RAM ⇒ 128 GB
Storage = 2TB
CPU

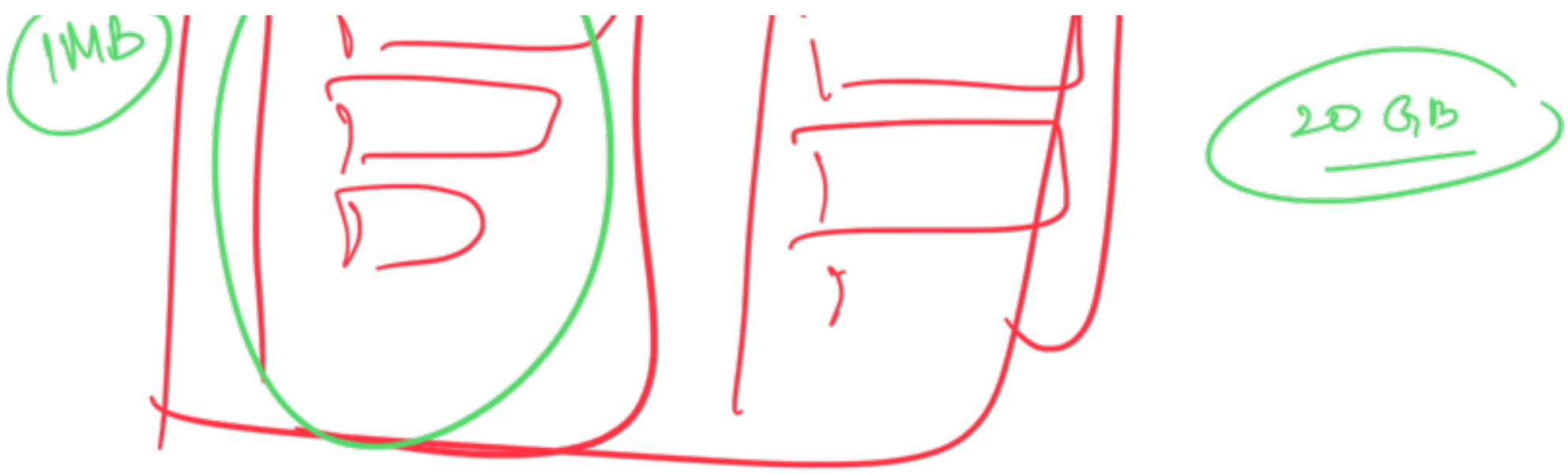
32 GB



Distributed Cache

2 Billion Users

$2 \times 10^9 \times 10$ KB


1MB
20 GB


Cache

C1

(A – E)

C2

(F – J)

C3

(K – O)

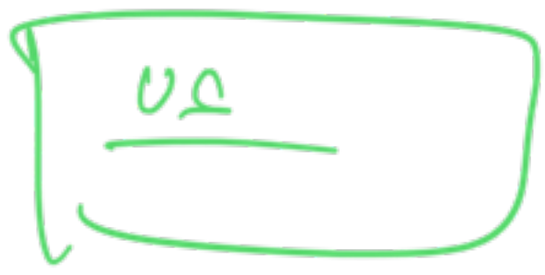
C4

(P – T)

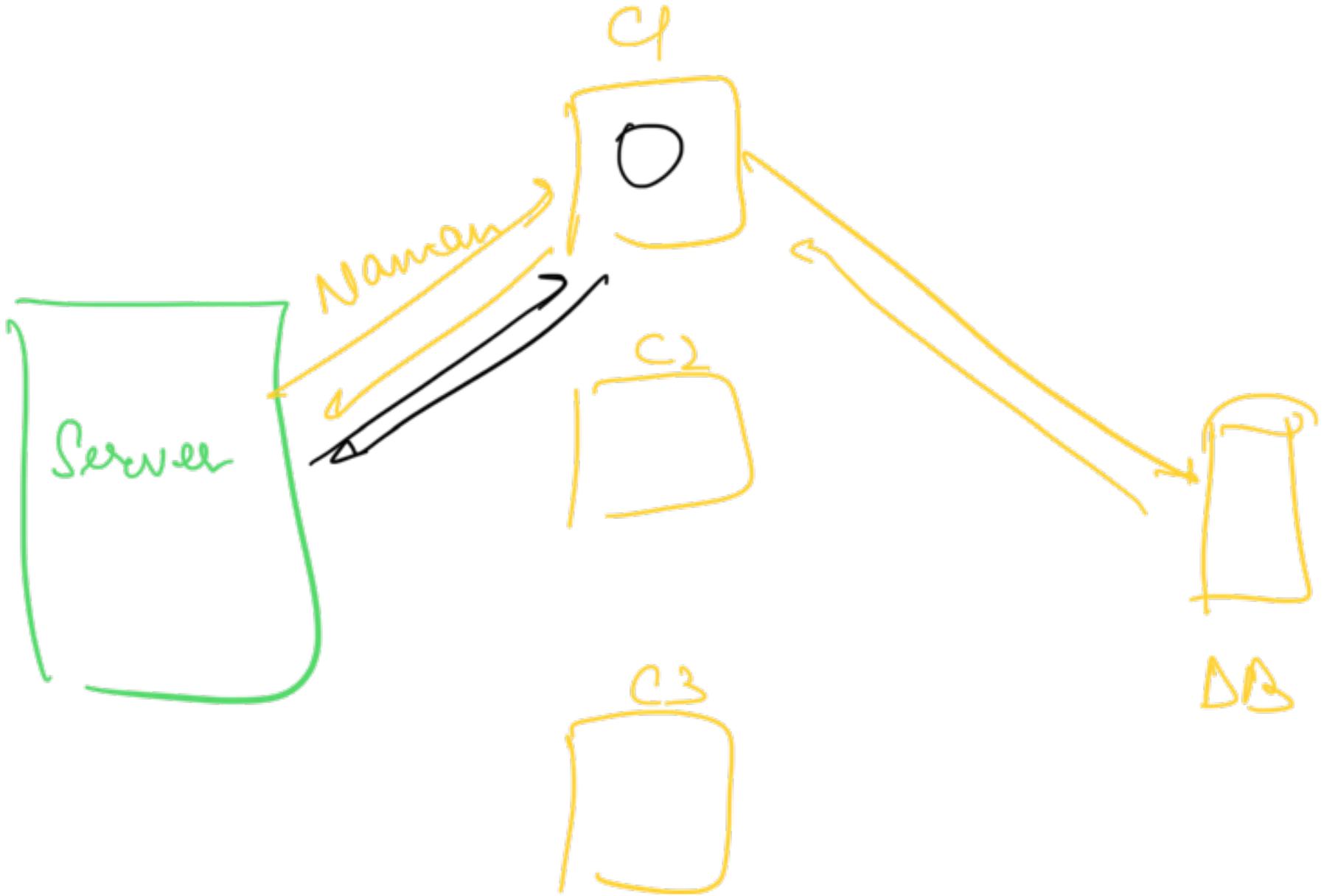
C5

(U – Z)

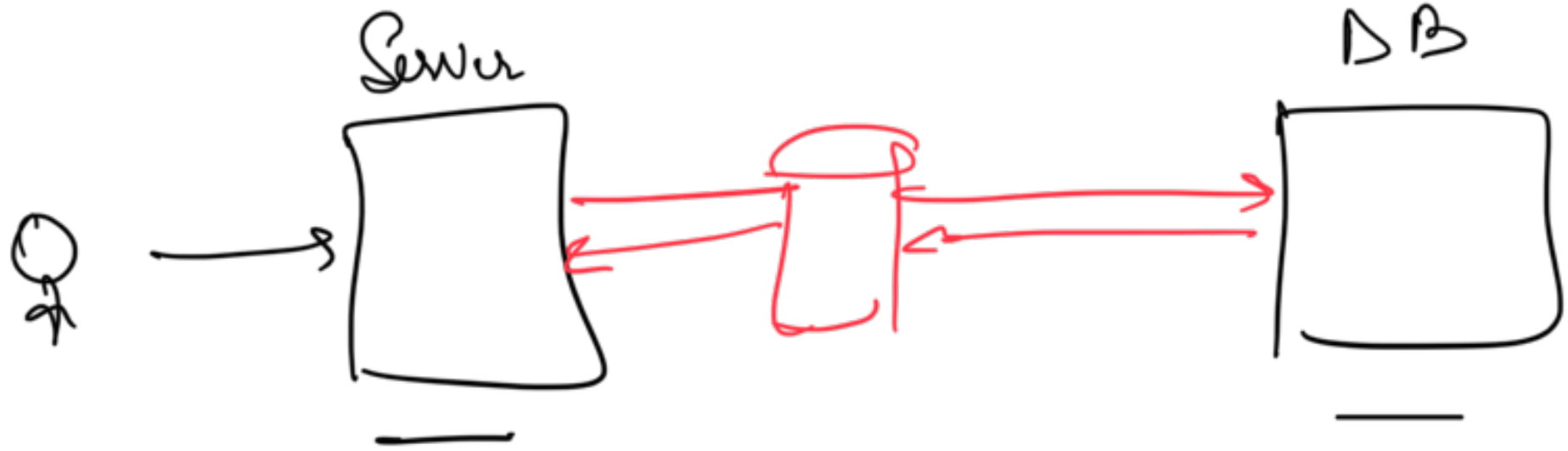
Geo sharding — distribute based on location



UK



Design a Distributed Cache



Requirements

handle

① Should support CRUD

C → Create / Add
R → Read / fetch
U → update
D → Delete

③ Limit on max Size of cache & Support for diff eviction strategy

④ Support for TTL

⑤ Hot Loading (Warming up the Cache)

⑥ Handle Concurrency & Thread Affinity

⑦ Request Collapsing

CACHE

DB

create(X,20)

(X,20)

@Getter

@Setter

@DI

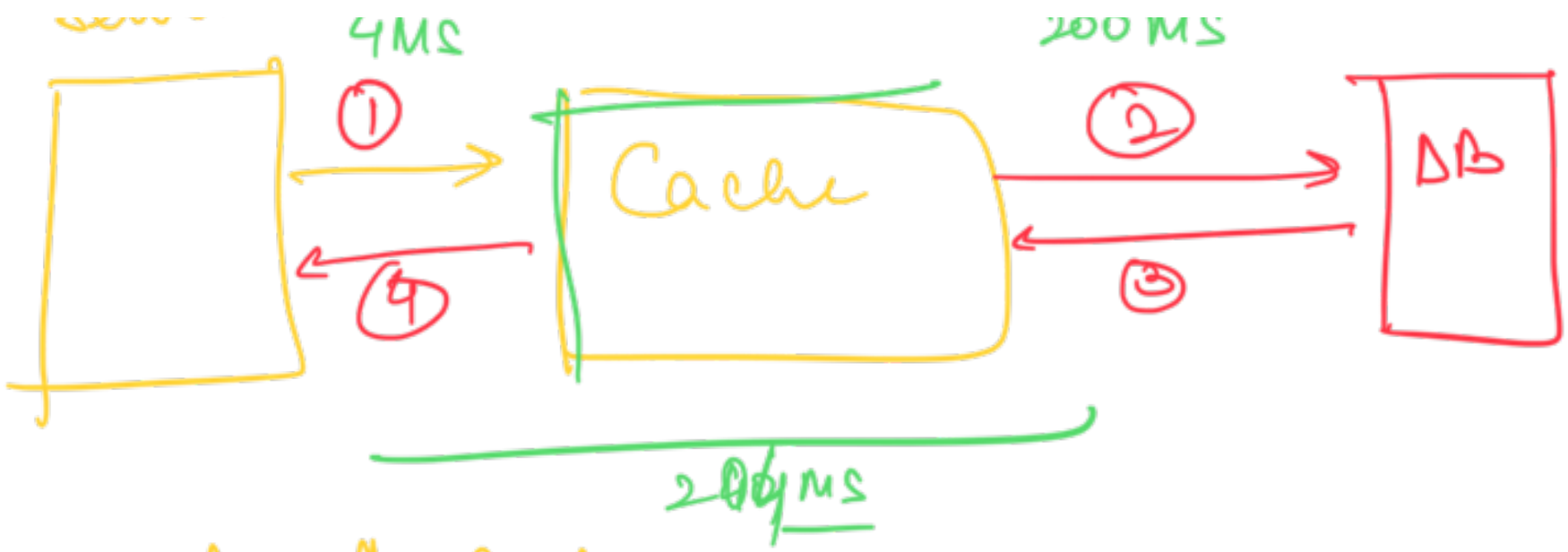
RAM

⇒ POJO

Create

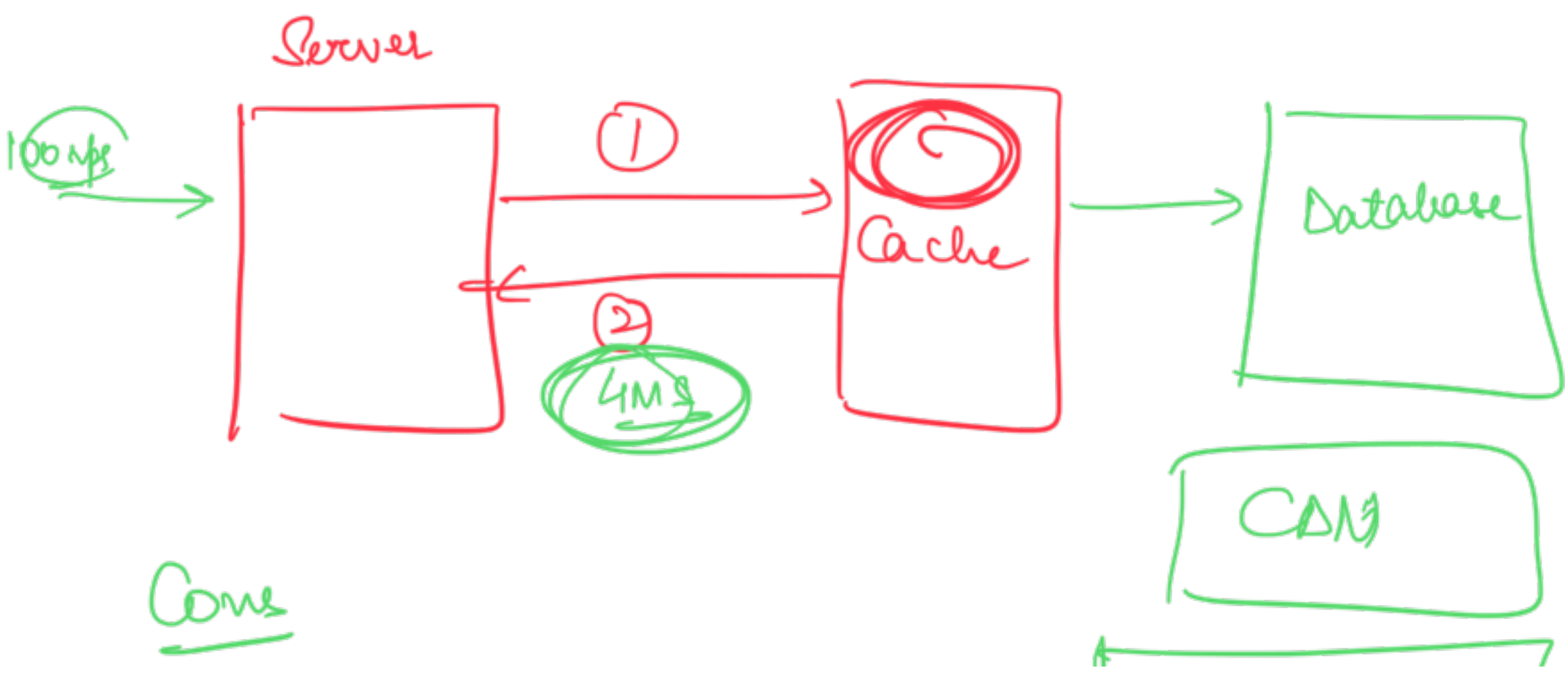
2 Types of Cache Based on How a Write is Stored

① Write Through Cache



② Write Back Cache

Twitter Newsfeed



① Data loss

Pros

① Efficiency (Fast)

Write Heavy

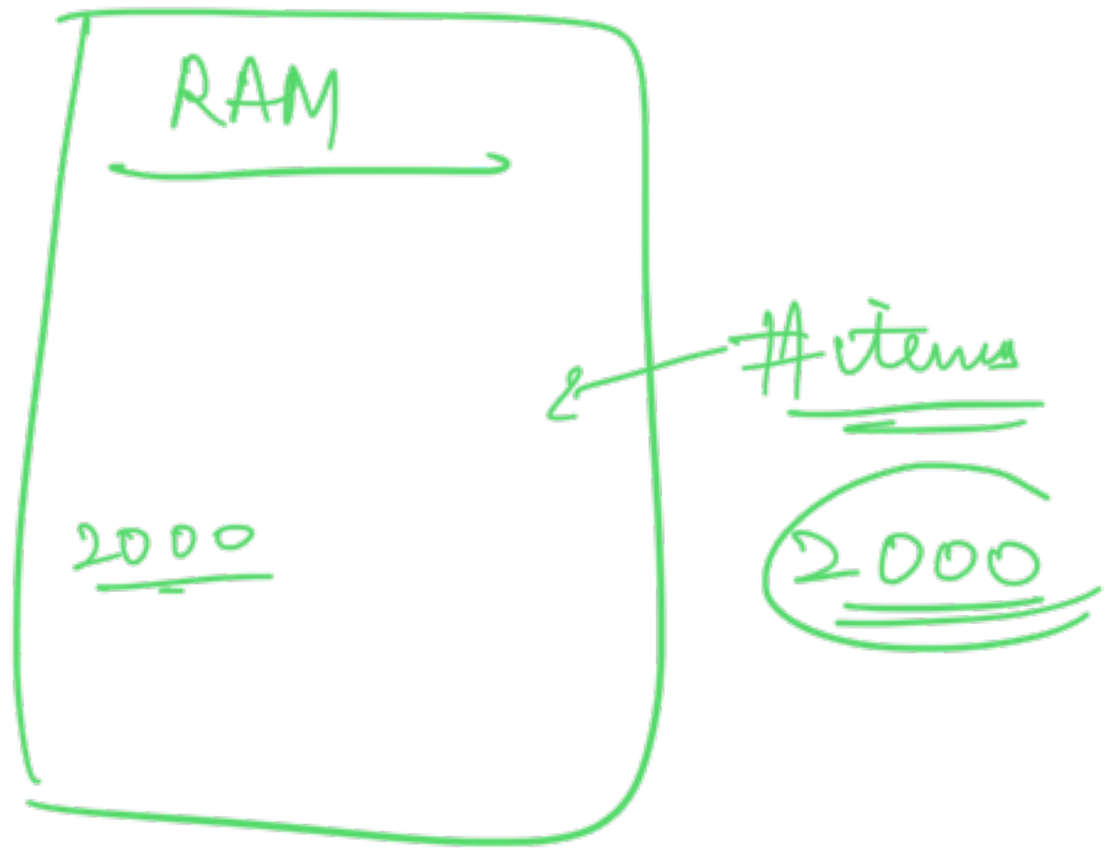
400

300

730

Twitter → Delete Your Account

+1 day Twitter → Reactivate


RAM
#items
2000
2000


(A, 1)
(B, 3)
(C, 5
(D, 62
Cache
(A, 7)
(C, 5)
X
15 items

What to do when you add something after cache is full ?

1. Remove one of the existing item
2. Add the new item

1. Random
2. LRU (Least Recently Used)

② LFU Remove the item that is used the least # of times

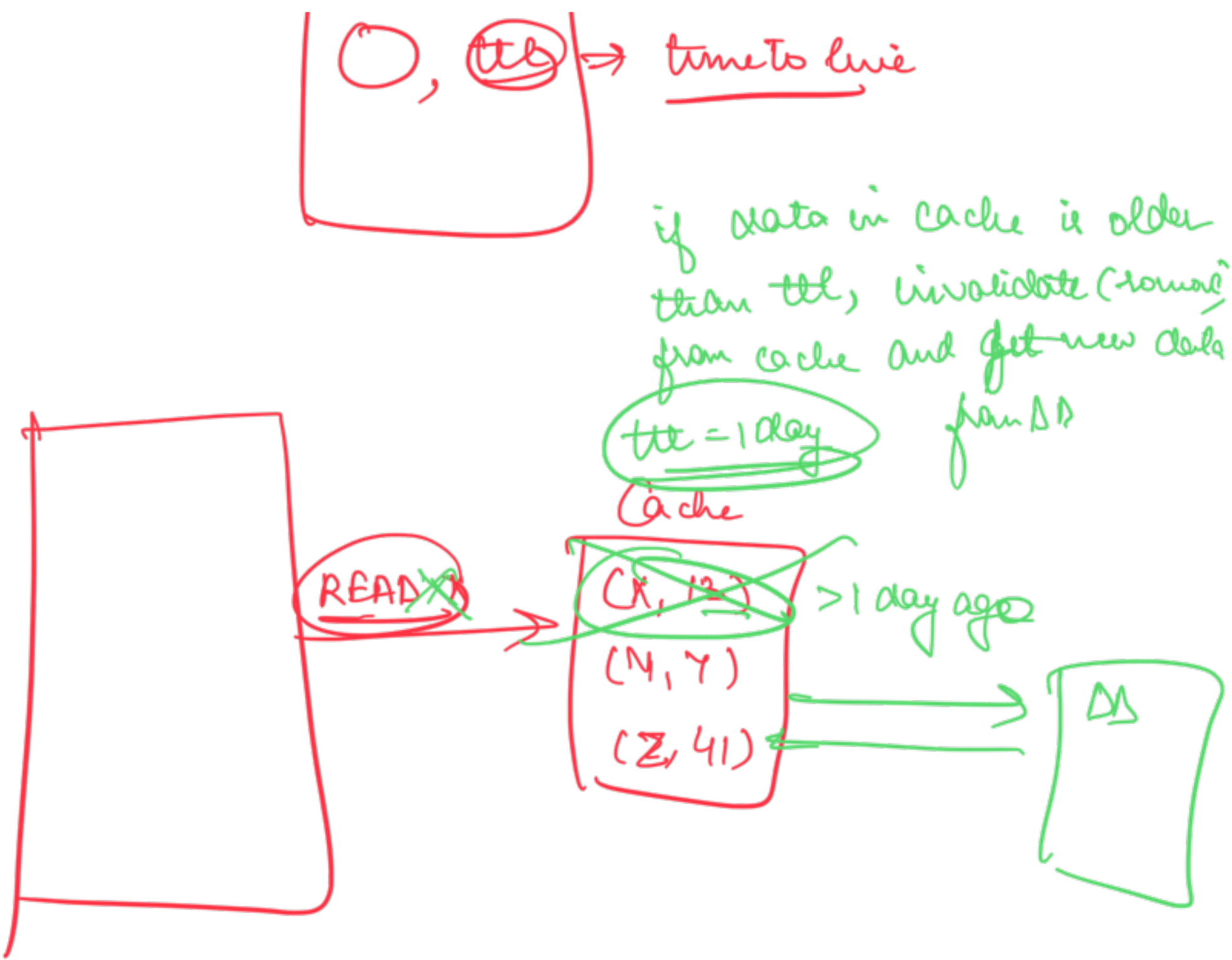
Quick Sort Pivot: left, Middle, right

Randomized Quick Sort

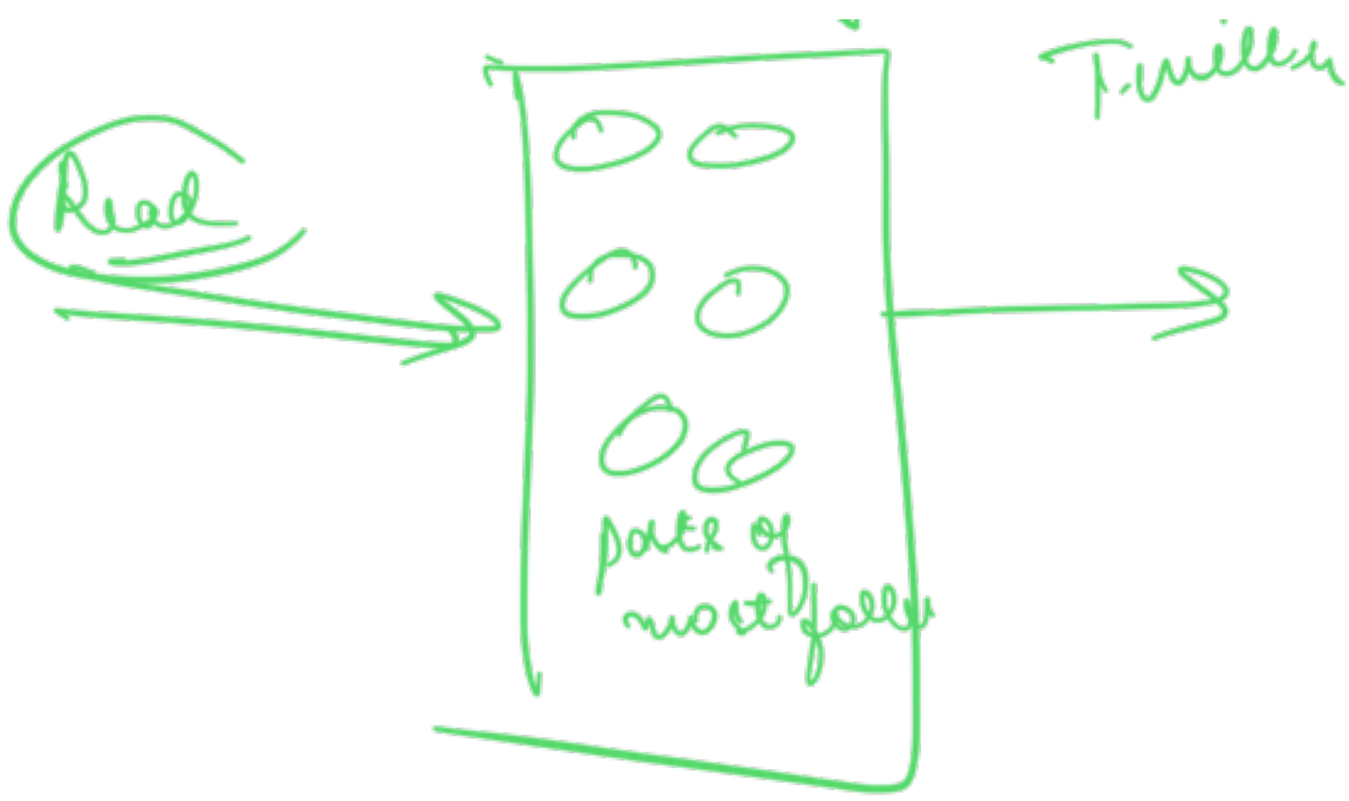
pivot: random

Read the proof of probabilistic TC of randomized QS ⇒ O(klogN)

Cache

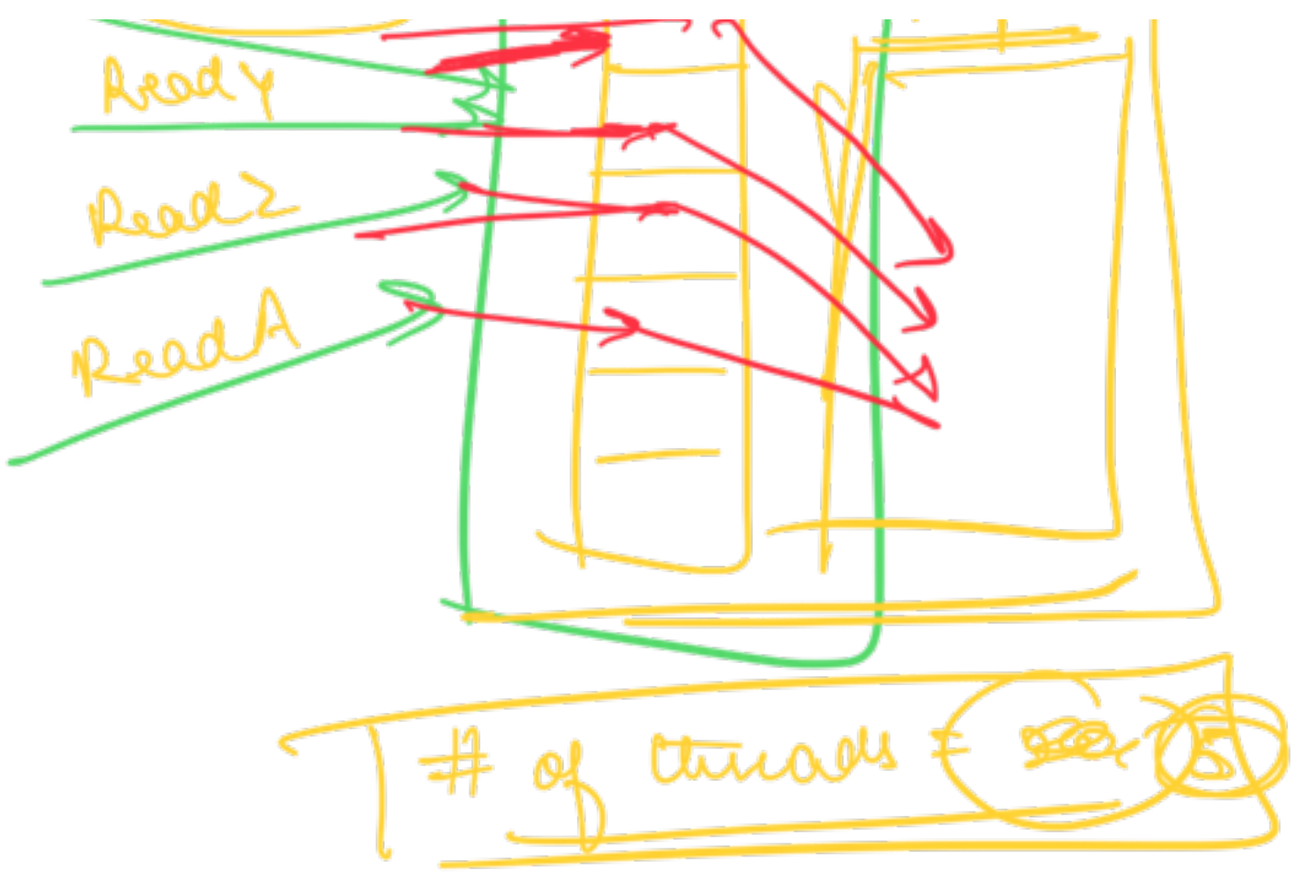


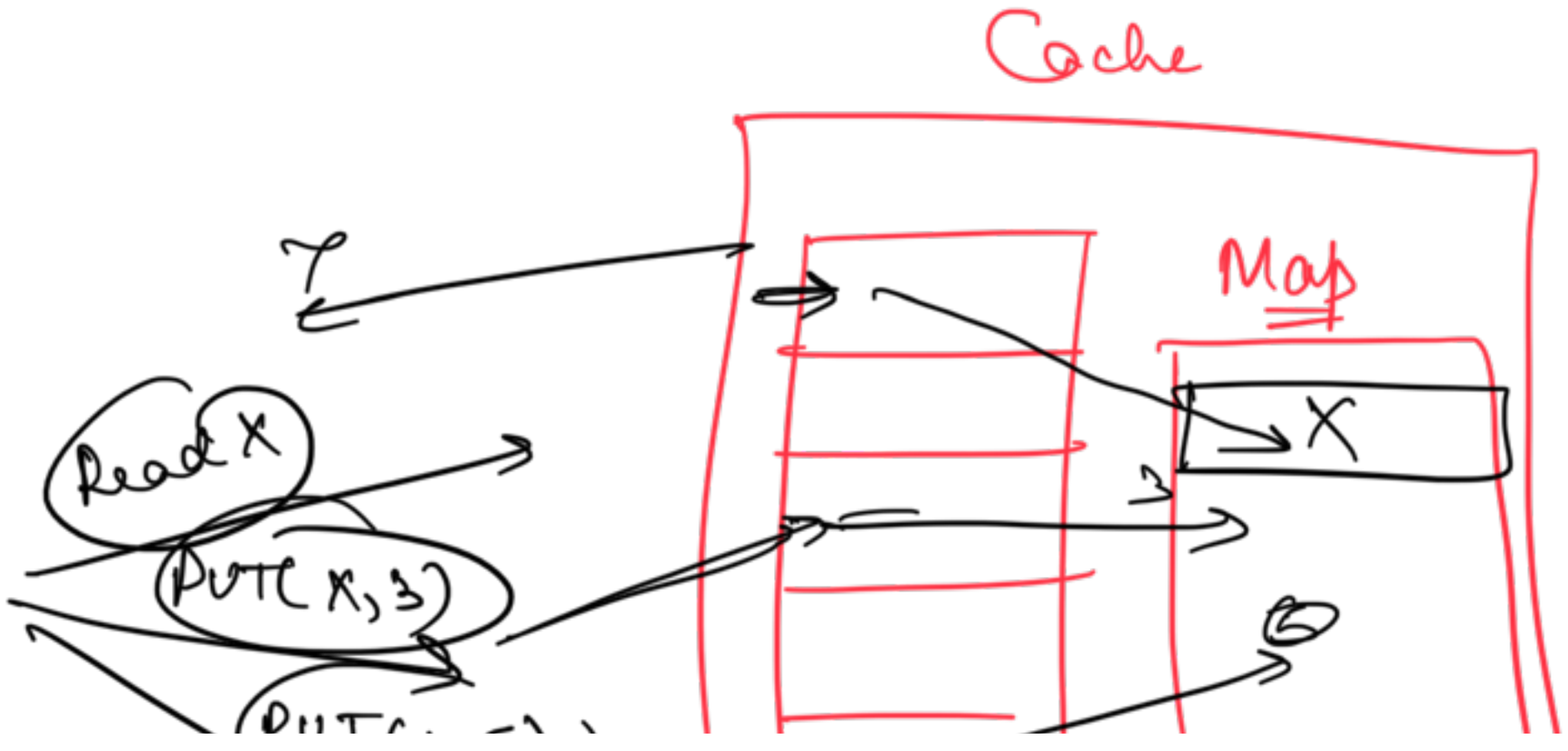
initially cache will be slower



Hot loading

Cache Warming

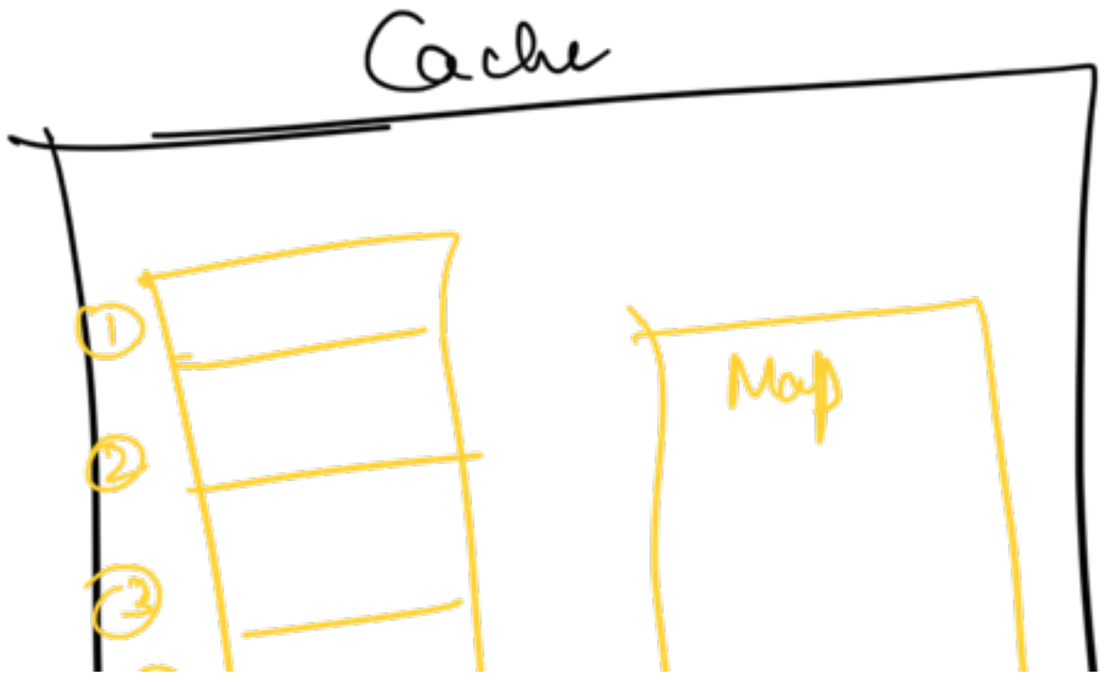

Read Y
Read 2
Read A
# of threads


Cache
Y
Map
X
Read X
PUT(X,3)

Sol^n

(1) Lock → Slow

(2)



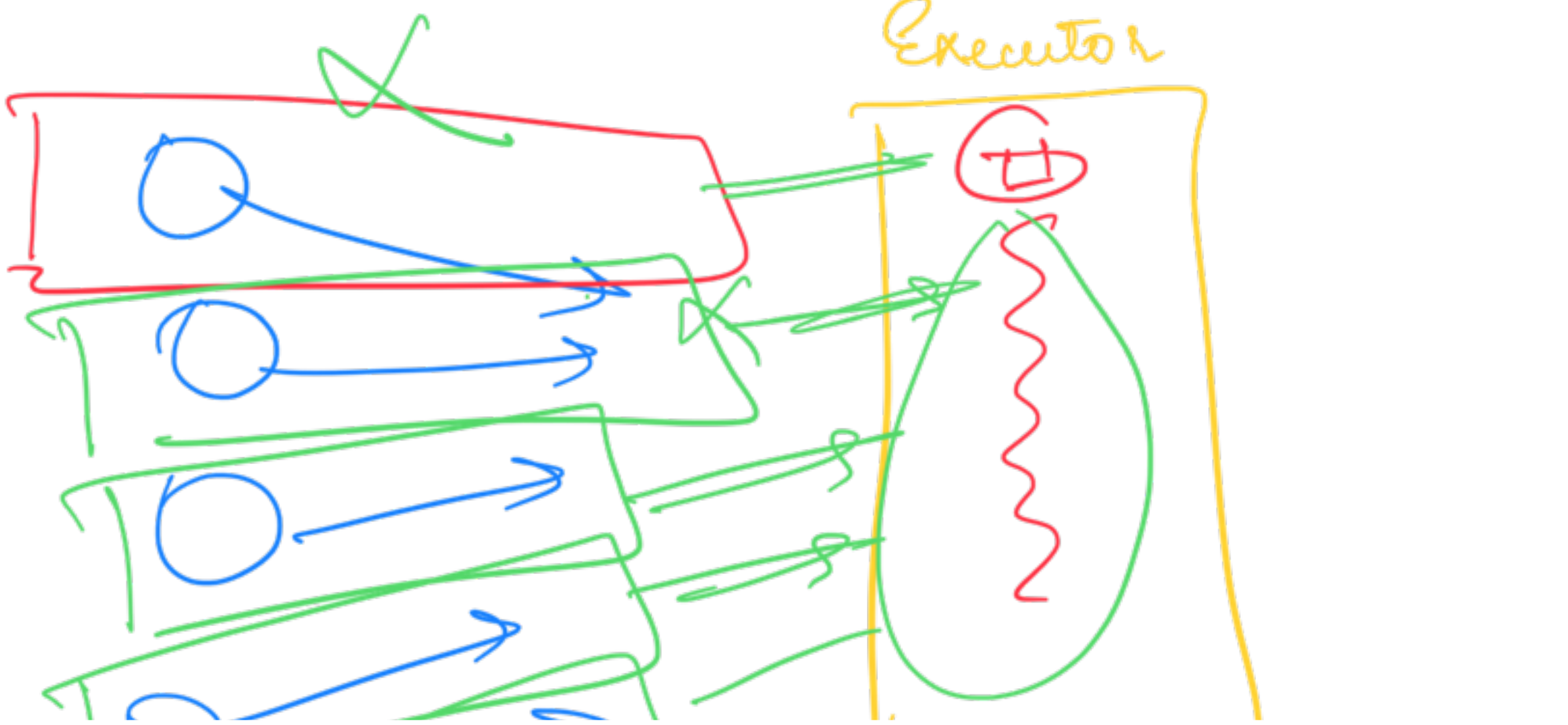
Concurrency issues because multiple threads same key

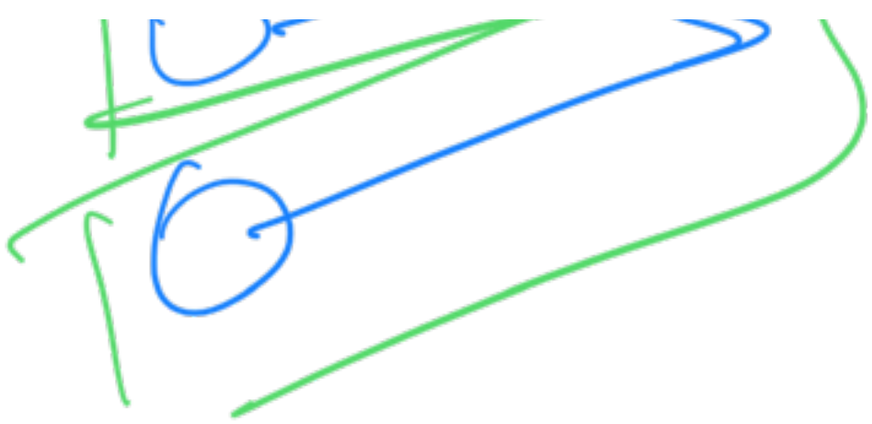
What if I say that all the locks to a particular key will happen from 1 thread

fixed Thread Pool

Cached Thread Pool

Single Thread Executor





[57]

No of kernels = 16

Cache {

1Byte 10^9 1GB

Executors [15] ⇒ [STE, STE, STE, STE, STF]

save (x) 20) {

exit

Executors [x % 5]. execute (saveU)

}

}

Still faster than lock

~~A%5~~

F%5

H%5

J%5

Thread Affinity

set / get / update / delete

A → A%5

B → B%5

delete (x)

get (x)

X%5

So create (X, 10)

get (X)

update (X, 30)

delete (X)

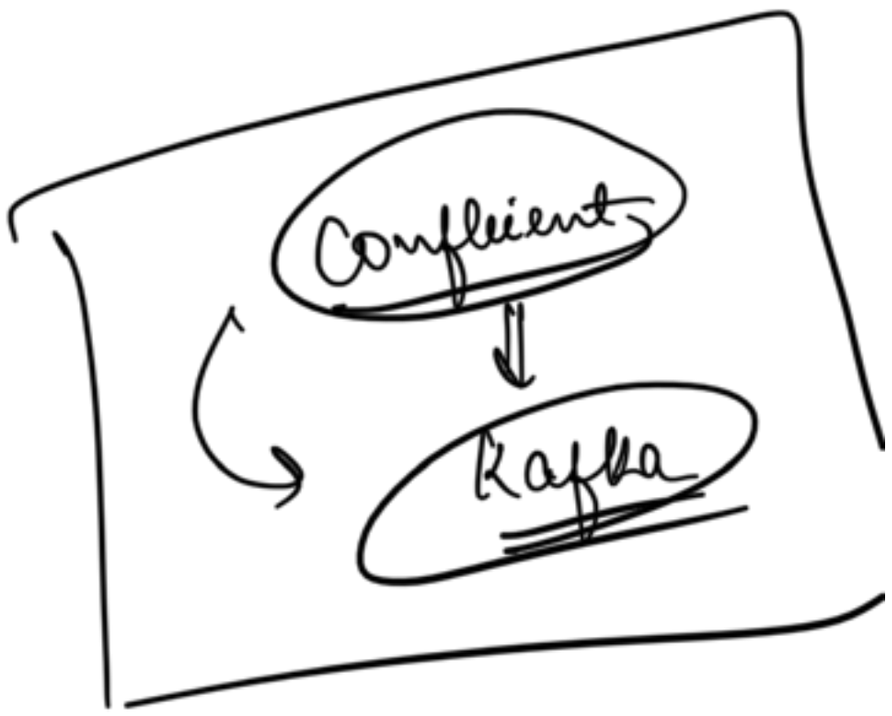
if your op^{v} are IO intensive → More

if your op^{v} are cpu intensive → less

Redis
Single Thread
Redis Server
Memory


Redis 1

App
Serv
middle
vs
dist-logic
A - M
N - W
X - Z
Redis 2
Redis 3



Dist Cache

Concurrency

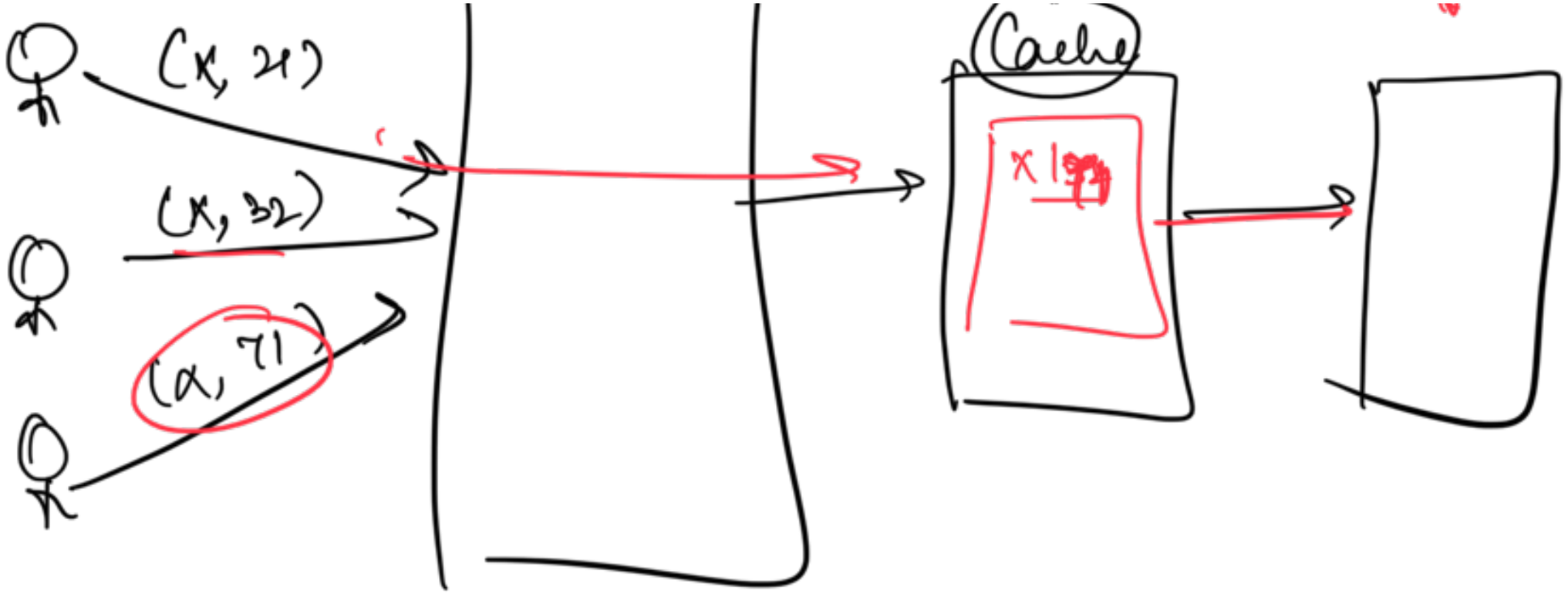
Dist Cache

each op^{n} → fetching from dist^{n}

Request Collapsing

3 write

1 write

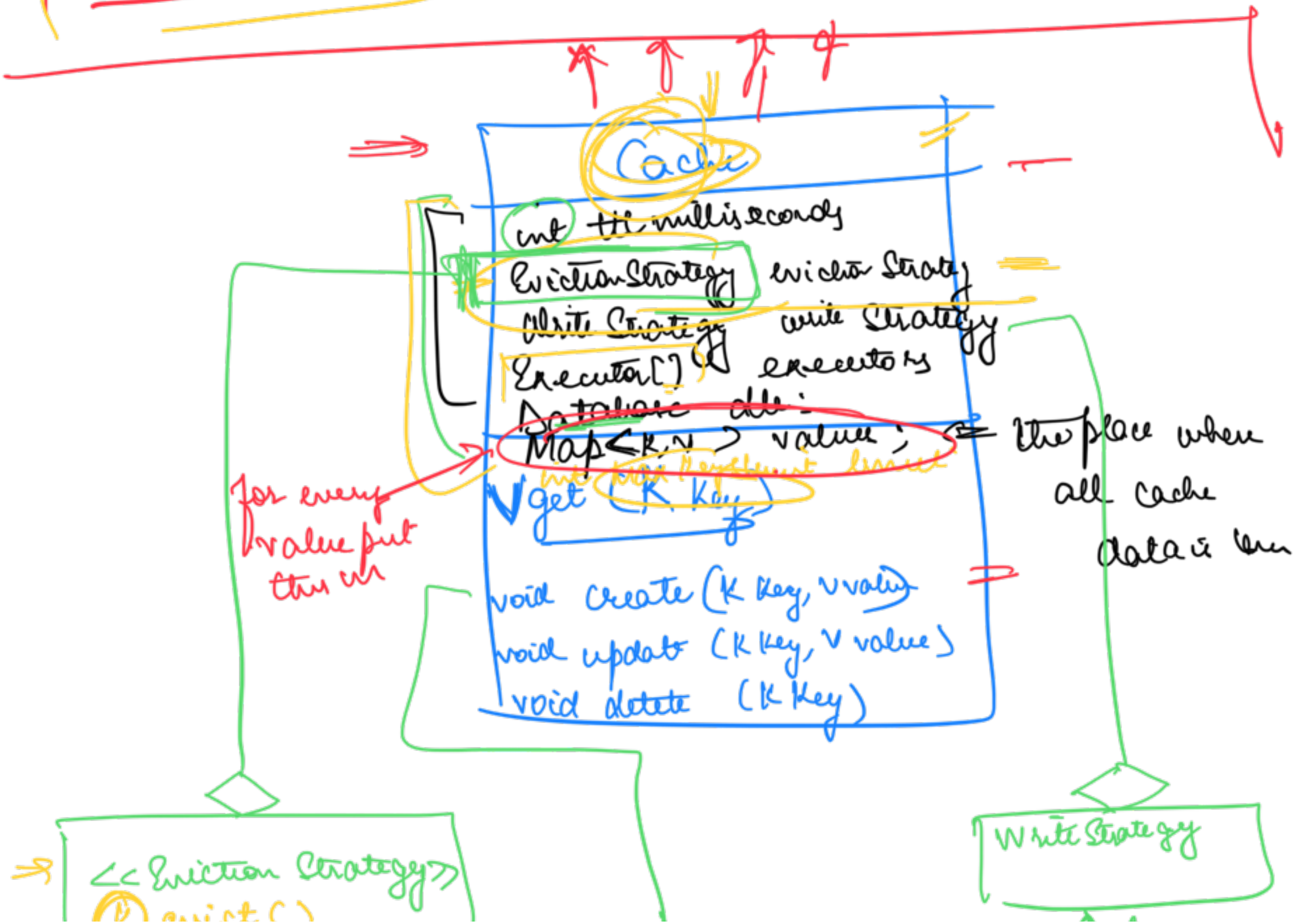


Request Collapsing

→ write Back Cache

→ Reduced # of Calls on DB

Class Diagram



LRU Eviction Strategy

K evict()

Needs to know the last used time of every key

LFU Eviction Strategy

Needs to know the count of every key used

<< Datastore >>

get (k)

create (k, V)

delete (k)

update (k, V)

Write back Cache Strategy

Write through Cache Strategy

Recording

MySQL Adapter

MongoDB Adapter

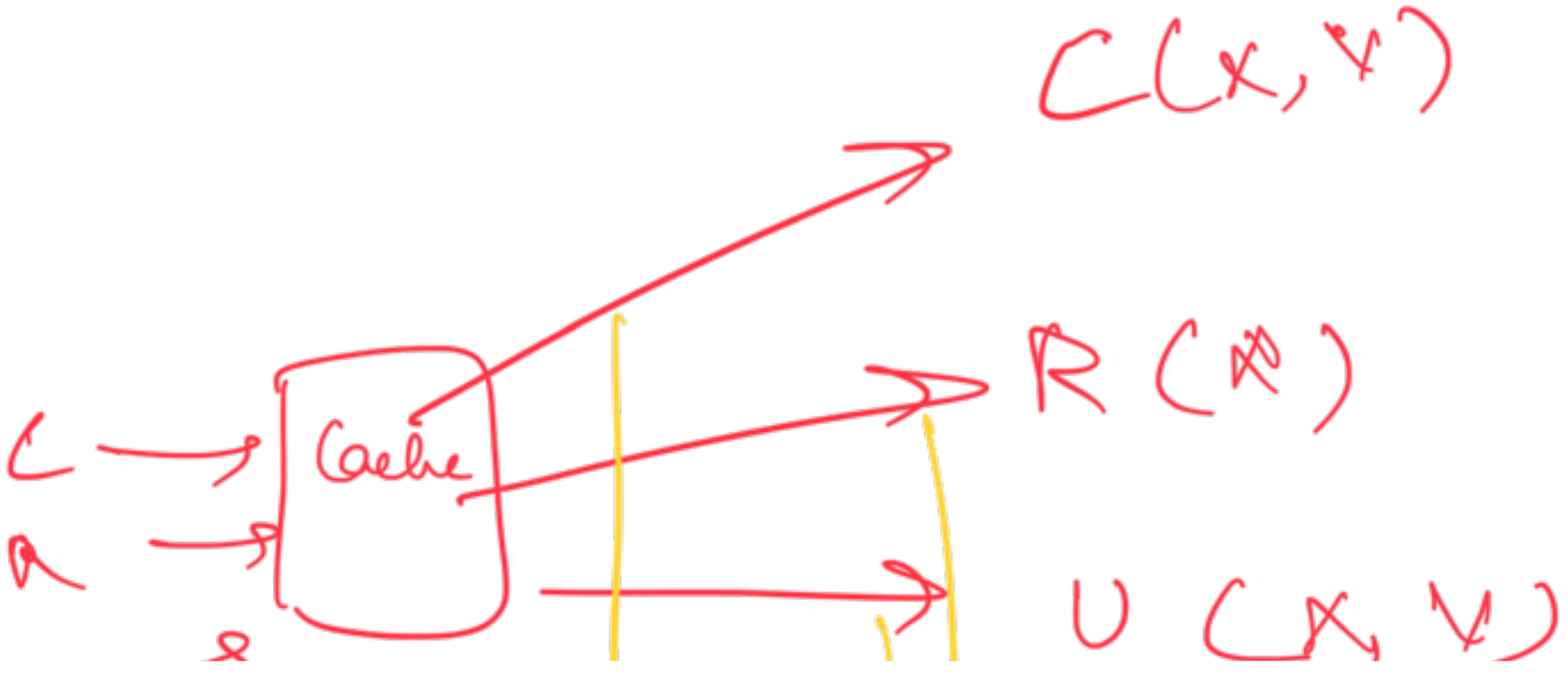
① Cache class shouldn't know what every eviction strategy needs

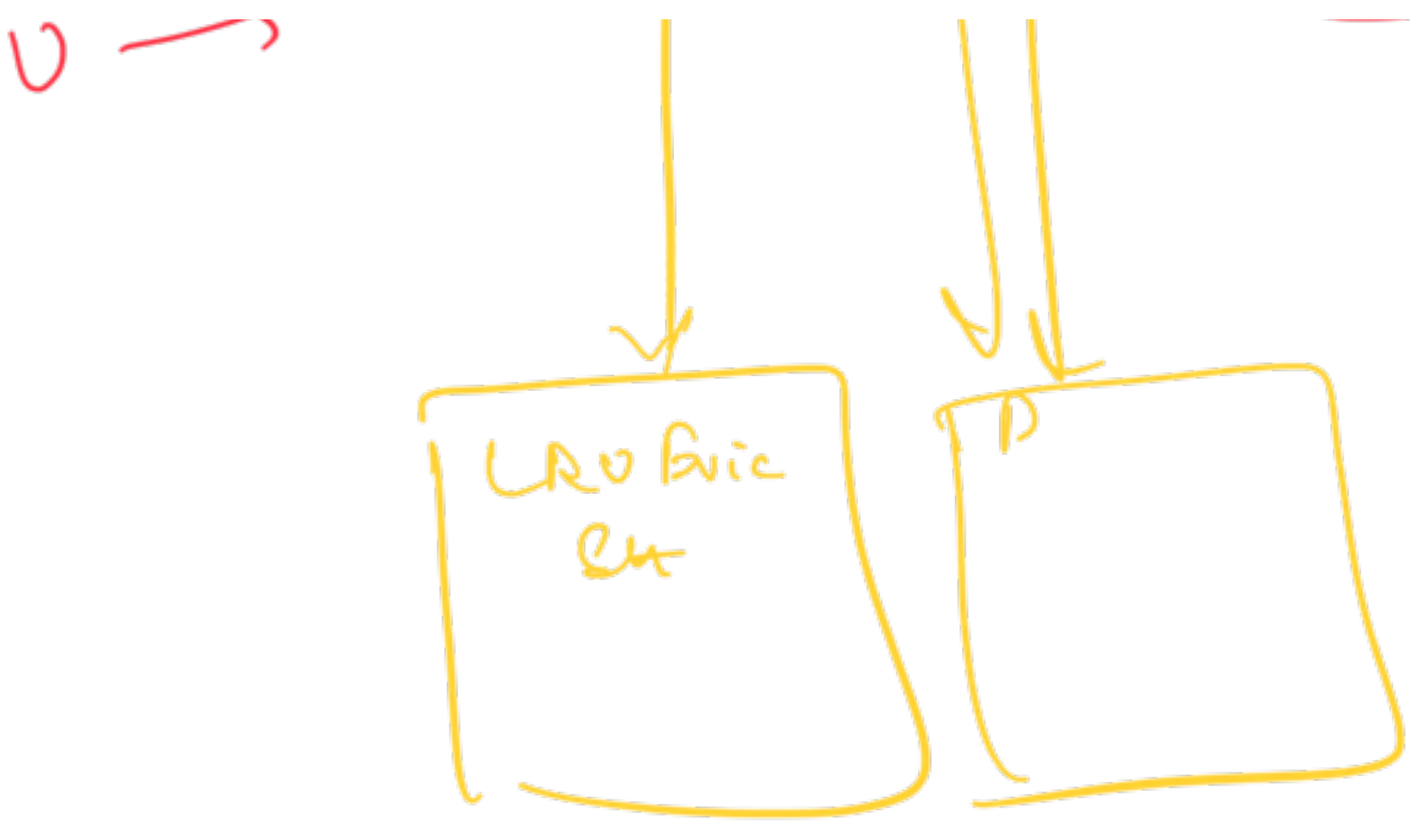
② Eviction strategy should get the data that is there with cache

Cache

Eviction

Strategy




Producer
Enum
- Create
- Read
- update
Subscribe (Event, Subscriber)
mp. event. add (sub
notify (Event) }

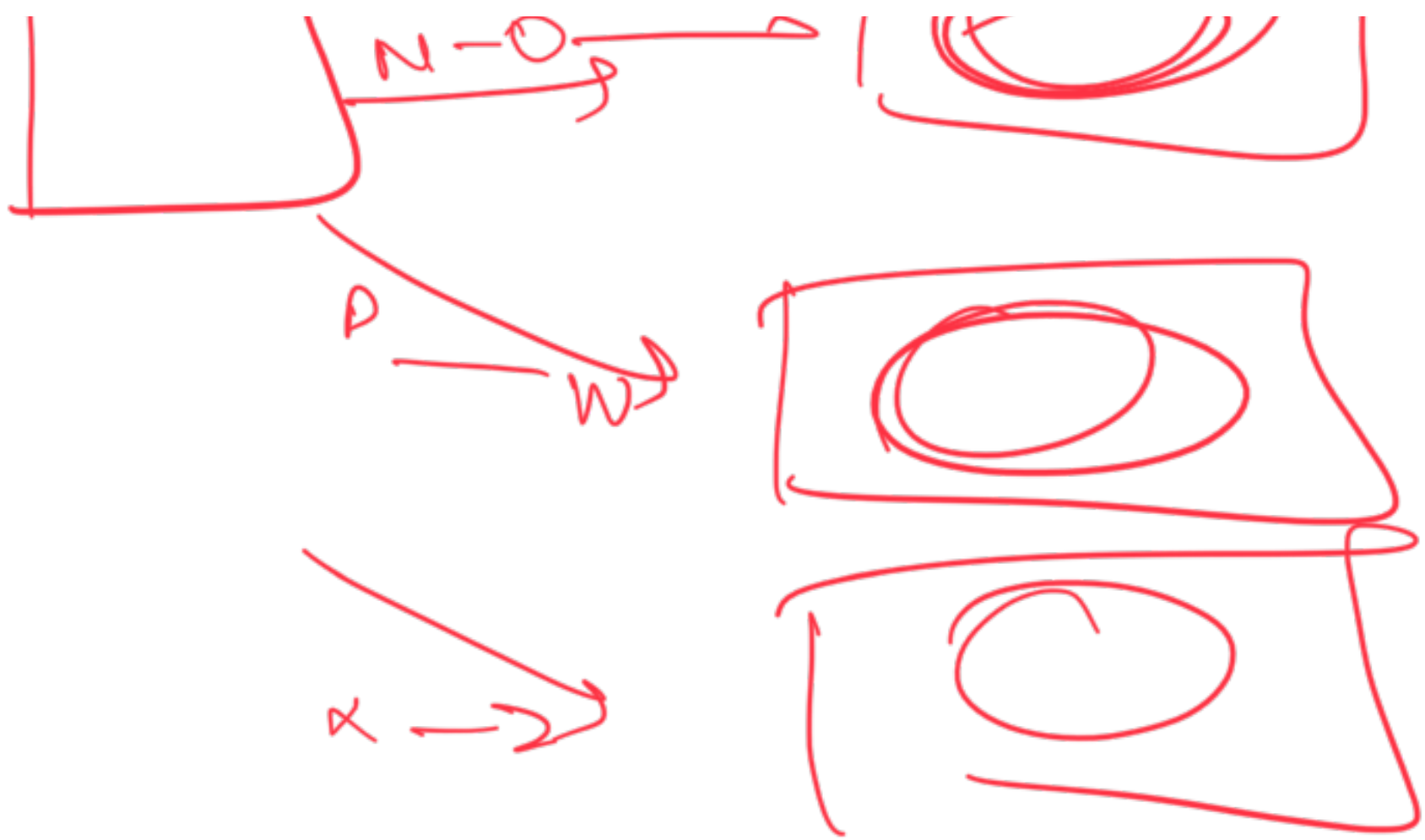
- Debit -

Code

→ All the knowledge of conc classes

↳ Concurrent DT

↳ Futures



Factory

DB

getDB with Name (String name)

if Name == mySql

return MySQL DBC

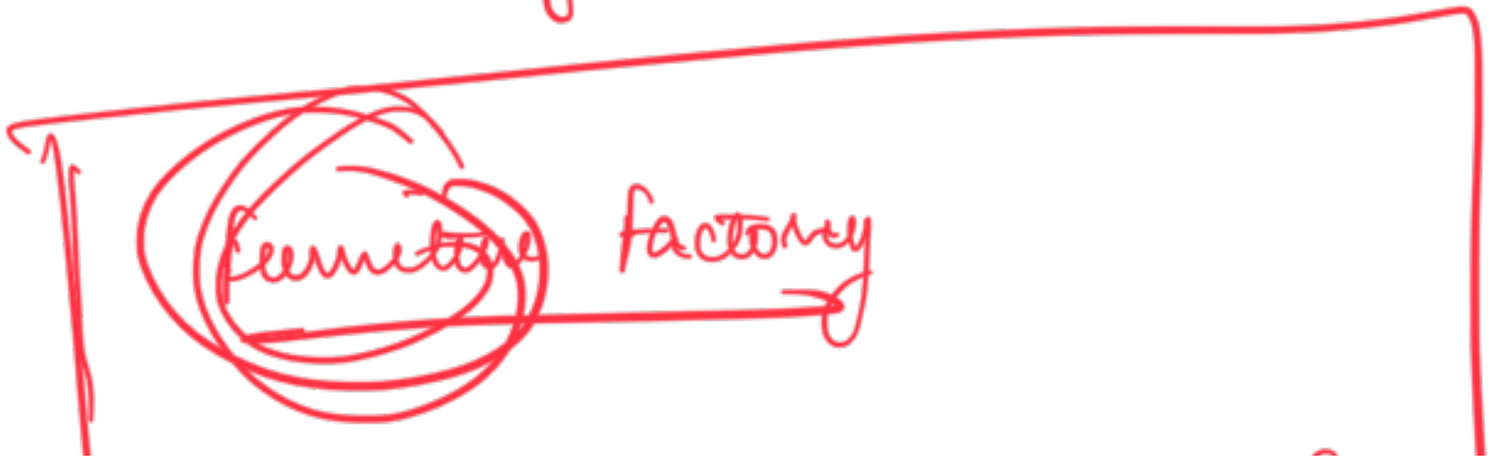
== pSQL

psQL are

BB

getDB DfType (Type)

Abs factory : related object



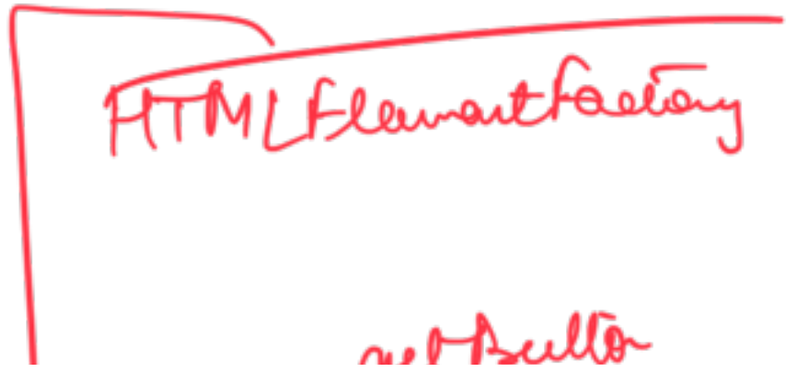


Table getTable();

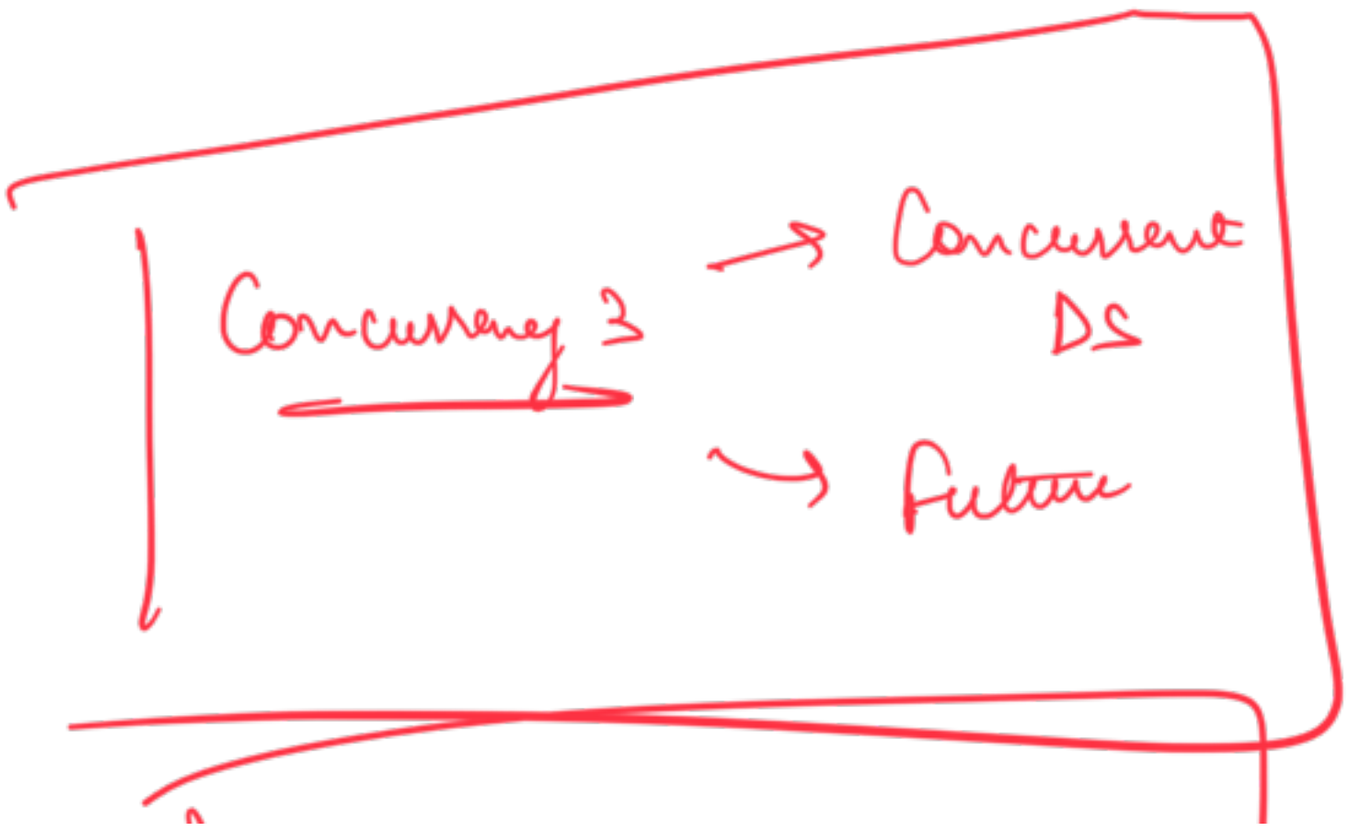
getMenu

}

Chair getChair() {

getDropdown

}



Tom

Recordings of 2 SDP

Dec

Adapter

Facade

Flyweight

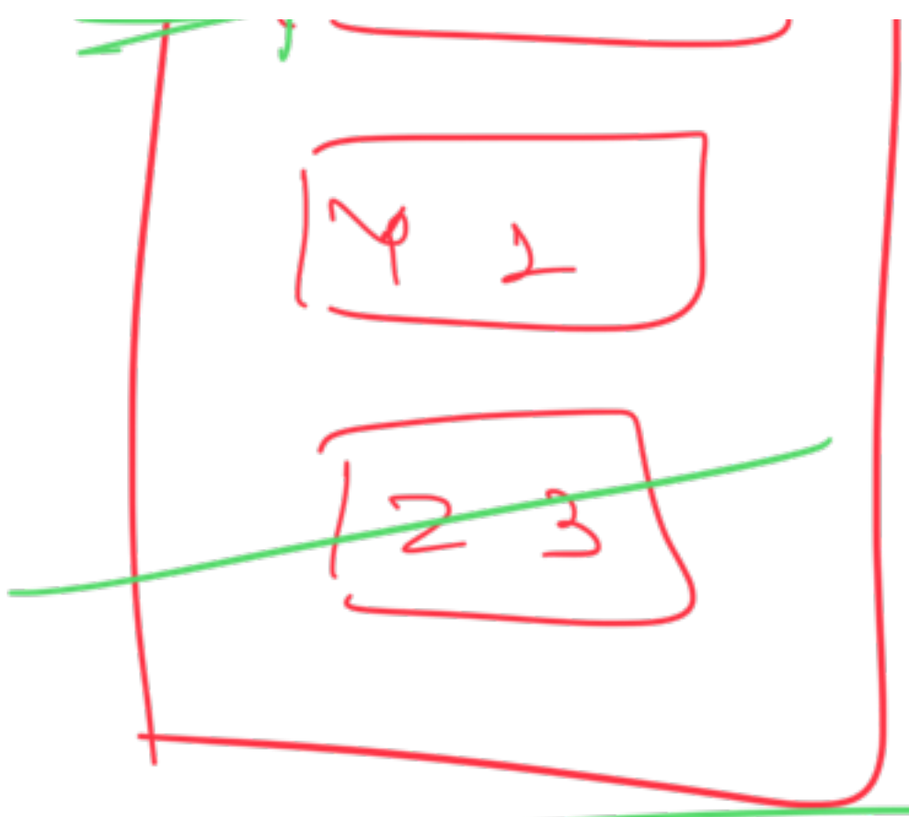
Search Controller {

Search() {

Search

Search Controller

Search Service



① Check at time of access if the key is expired

② Have a parallel thread which periodically removes expired keys.