DOUBT SESSION

- → Design of FB Messenger ✓
  - a.) How to choose DB
- → Bloom filters ✓
  - a.) VV Beautiful Algo
- → Doubts

45mins

✓ This Saturday we won't be taking class.

Design of FB Messenger

Choosing DB

R : W → 1 : 1

→ Both read as well as write heavy

→ Try to reduce one of them

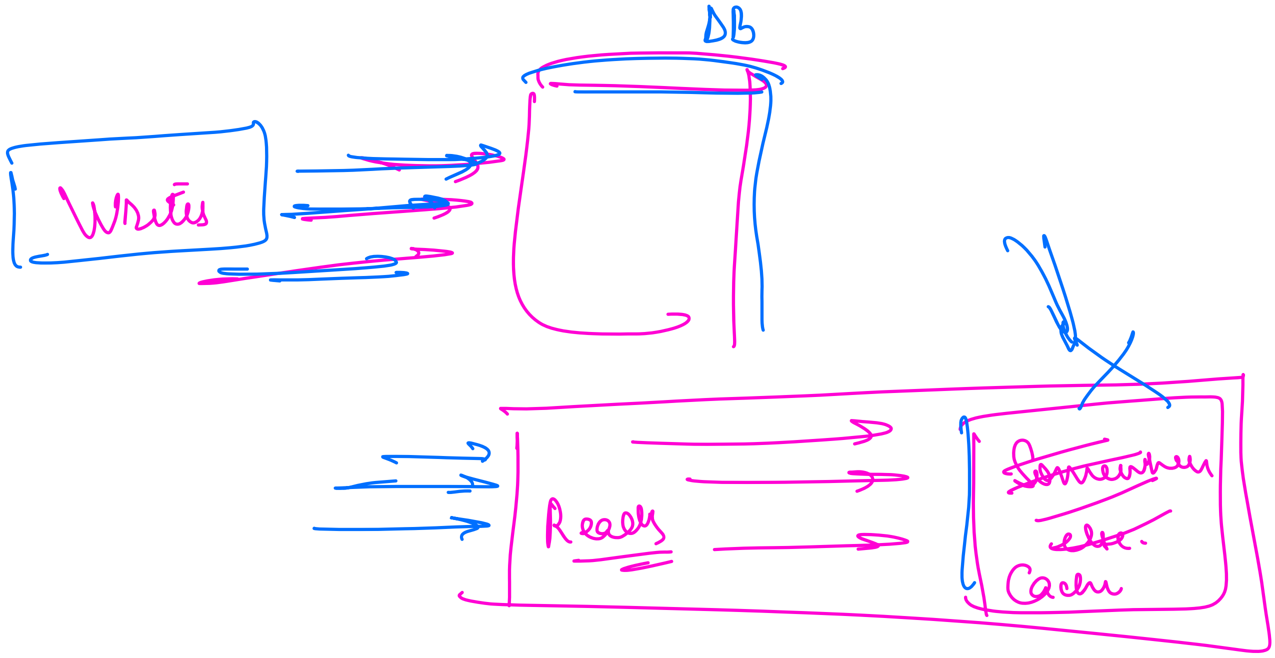
Let's try to reduce write

eg. Google Typeahead.

→ Batched write

- ⇒ This won't work in FB Messenger
- ⇒ Consistency is very very important here.

Let's try to reduce reads



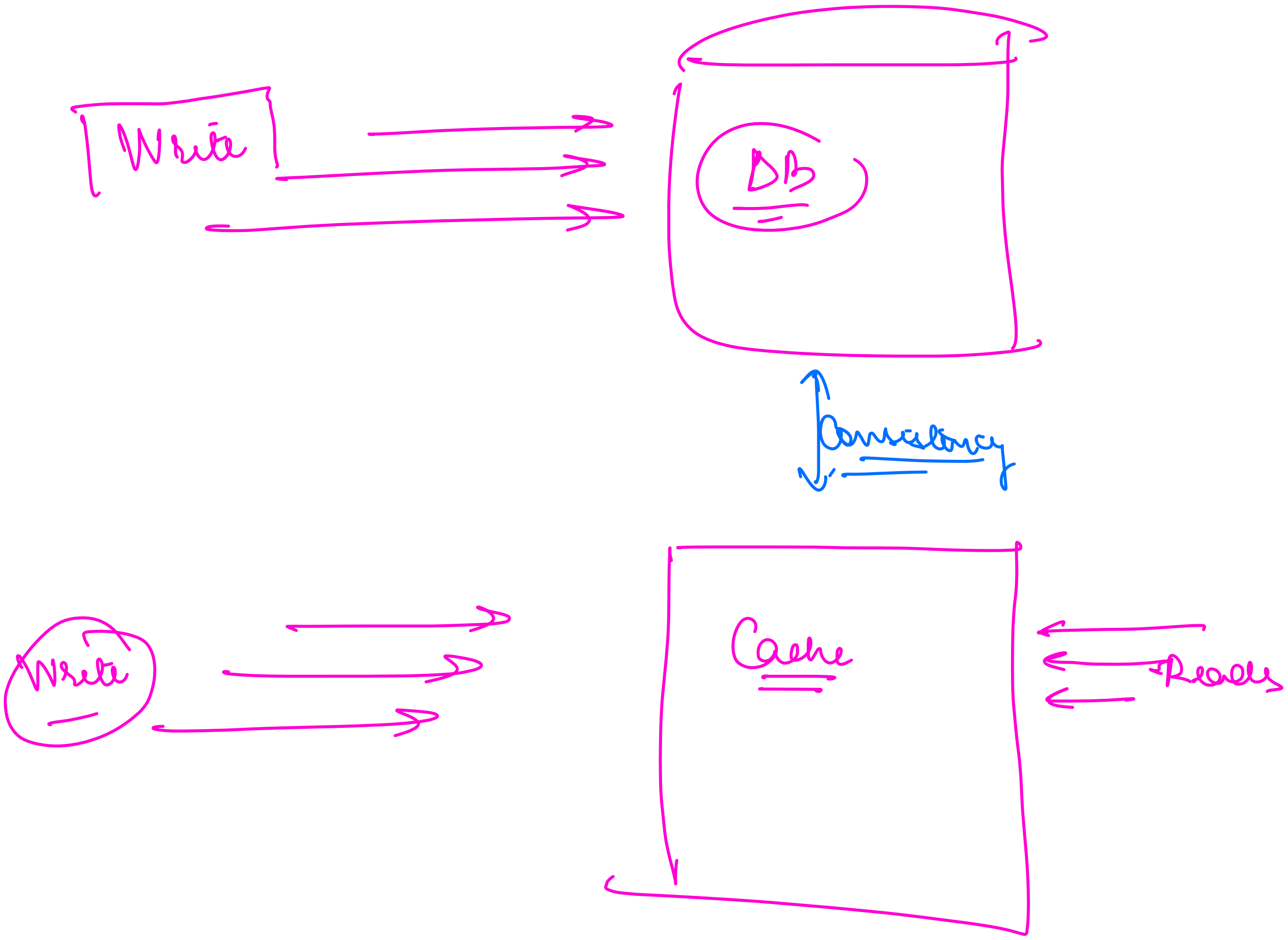
⇒ Because I have to maintain consistency all data in DB should also be present in 'X'

- ⇒ X should be consistent with DB
- ⇒ X should be able to handle large # of writes

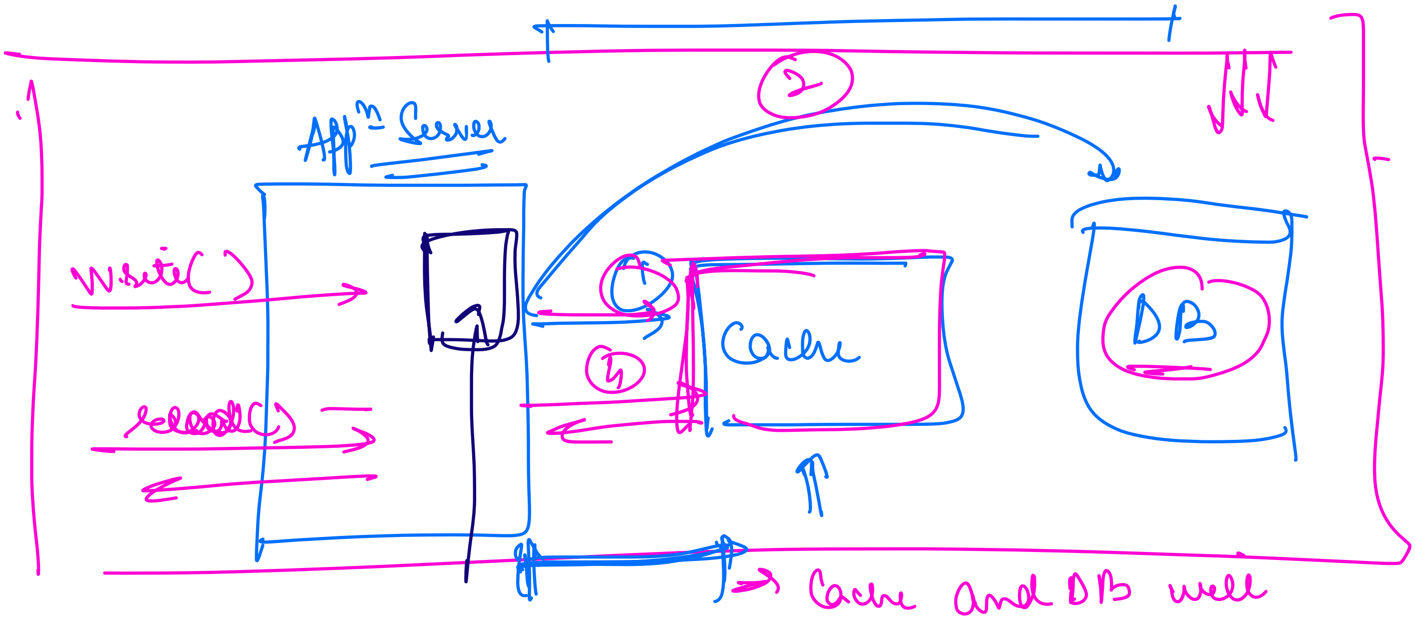
=> X should be able to handle large # of reads.

=> X should be data other than disk

=> X should be storing data in RAM.



→ Write Through Cache



→ Cache and DB will always be consistent

Can I optimize this further

Latency while writing

(i) app^n server and cache

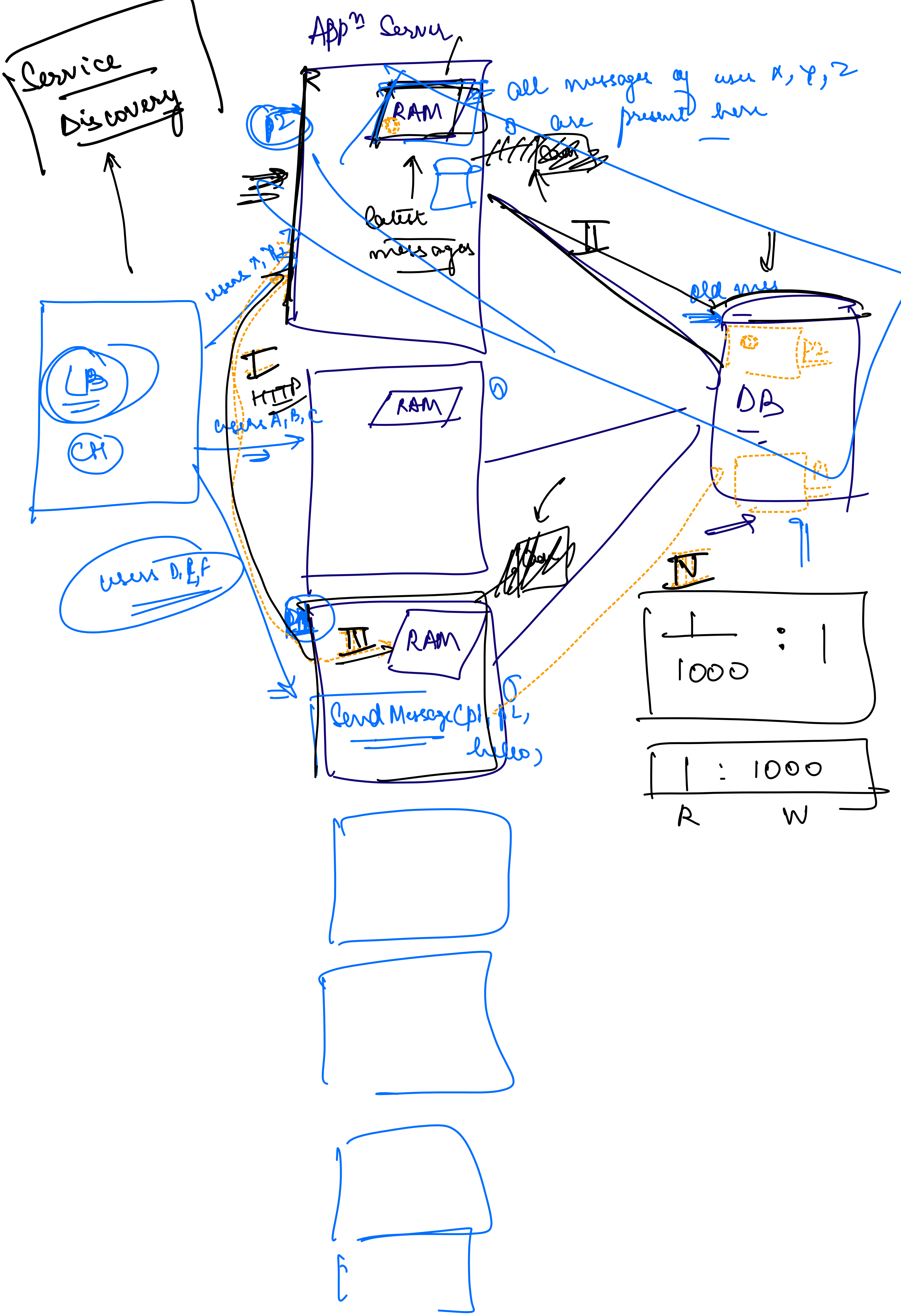
(ii) app^n server and DB

⇒ slow system

Latency while reading

(i) app^n server and cache

What if I cache within the app^n server


Service Discovery
Appⁿ Server
R
p2
RAM
all messages of user x, y, z are present here
latest messages
II
old msgs
DB
p1
p2
p1
LB
CH
users x, y, z
I
HTTP
users A, B, C
RAM
users D, E, F
III
RAM
Send Message(p1, p2, hello)
IV
1/1000 : 1
1 : 1000
R
W

Send Message (p1, p2, hello) {

⇒ we have to put data in shards of p1 and p2

}

What DB ?

→ Handle large # of writes

→ NoSQL DB like Cassandra / HBase

→ |R| , |W|

small value of w

Document DB ?

unstructured data

message

id	sent by	time	—	—	—

conversations

user_id	conversation
1	[{ auth: 141, latest: "Hey" } { auth: 123, latest: 'Bye' }]

Bloom Filters

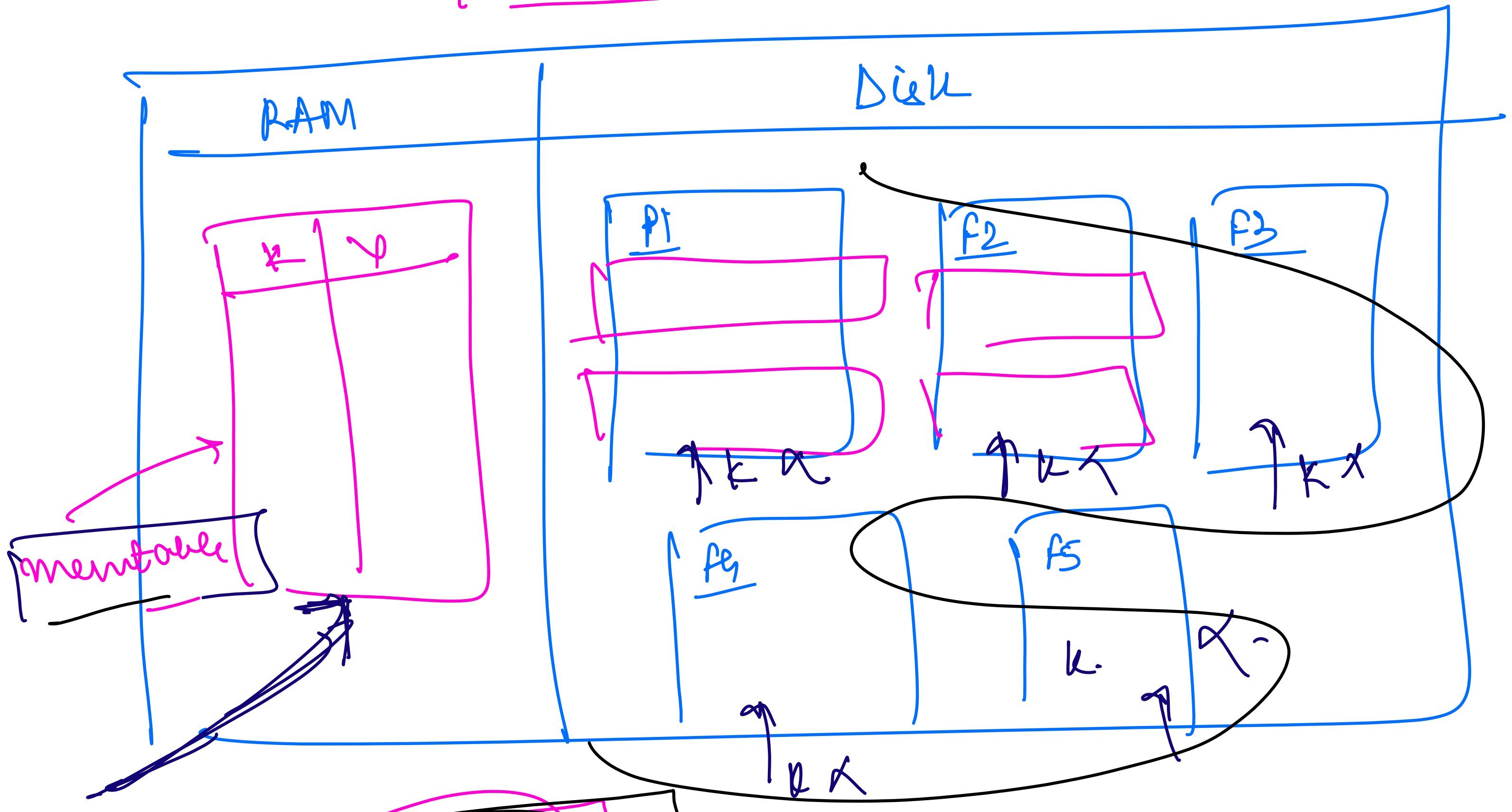
- → to Search for Something you have to go to DB
- → to check if Something exists or not, we need to go to DB.

⇒ latency

⇒ time DB will take to execute query

NoSQL Internals

→ LSM Trees



find (k)

→ Go to memtable: if yes: return

if no: find one by one in each file from latest to oldest.

if state not there ⇒ doesn't exist

⇒ { Searching key that doesnt exist is going to be most time consuming }

{ "I have to spend a lot of time to check if something doesnt exist" }

Can I get to know if something doesnt exist fast ⇒ Bloom filters

0.1% : false positives — reduce as much as possible

exists (key)

- true : may or may not exist
- false : definitely doesnt exist

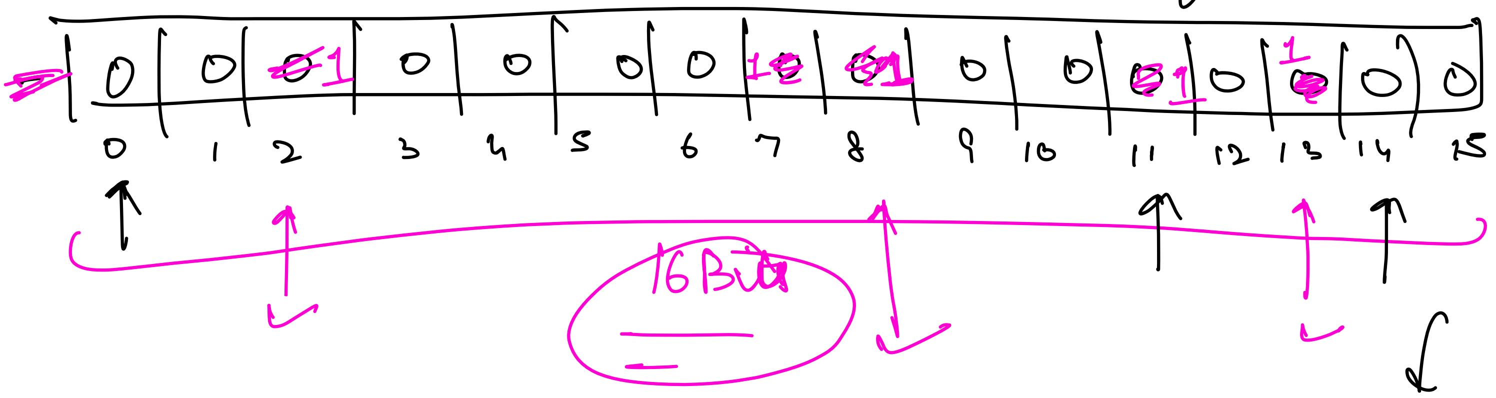
⇒ 99.9% of times if a key doesnt exist ⇒ done

⇒ 0.1% chance where I will still check

How Bloom Filters Work

① Bit Array → initially everything 0

② Bigger a bloom filter, lower false positive
The more the hash f^n, lower false positive



③ Multiple hash f^n, each selecting from 0 to 15

h1( )

h2( )

h3( )

Whenever someone inserts a key:

Compute the hash value of that key for all hashes.

K1

h1(key) → 2

h2(key) → 8

h3(key) ⇒ ~~17~~

Set 1 in bit array at all these place.

Insert (K2)

h1(K2) → 8

h2(K2) → 7

h3(K2) → 13

O(1)

When someone searches a key

Search (Key)

⇒ compute value of all hash f^n for that key

search (K3) → h1(K3) ⇒ 0

h2(K3) ⇒ 11

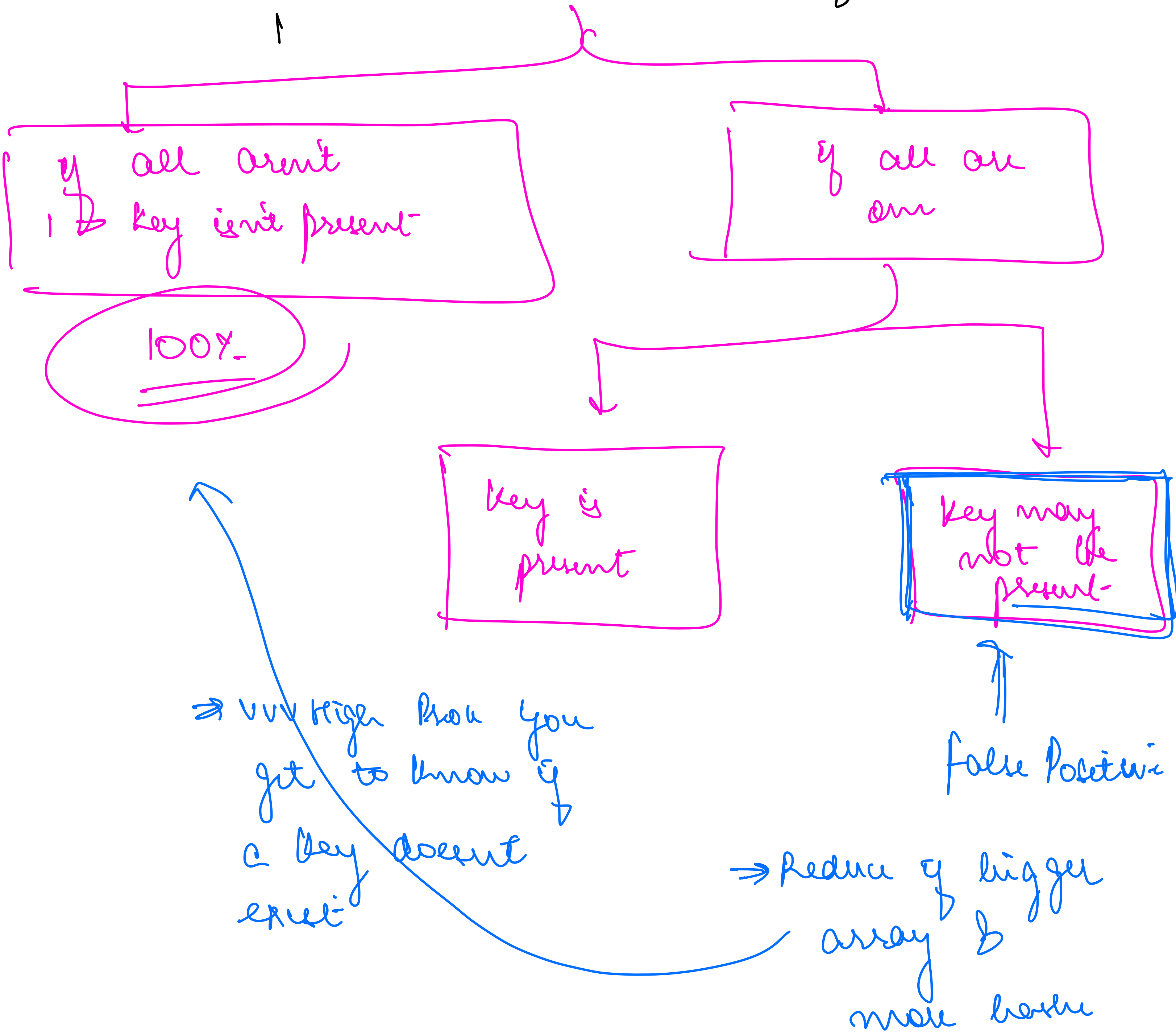
h3(K3) ⇒ 14

⇒ 2

⇒ 13

⇒ 8

→ Check if in bitarray all of these are 1



If all aren't 1 & key isn't present

100%

If all are one

key is present

key may not be present

False Positive

→ VVV High Prob you get to know if a key doesn't exist

→ Reduce if bigger array & more hashes

```
get (key) {
    exists = bf. exists (key) ;
✓ 99.9%   if (! exists) {
              return -1
          } else {                                   0.1 %
              search (memtable)
              search file [illegible] ?
          }
}

insert (key) {
    bf. insert (key) → O(1)
}
```

[100000] By [2^ key]

1 Billion → 32 hash f^n

10 Million

2^10 million

db.put(1, { }) ⇒ Cassandra

delete (key) → don't do anything

⇒ every day reconstruct

delete (k4)

h1(k4) → 4 → 0

h2(k4) → 2 → 0

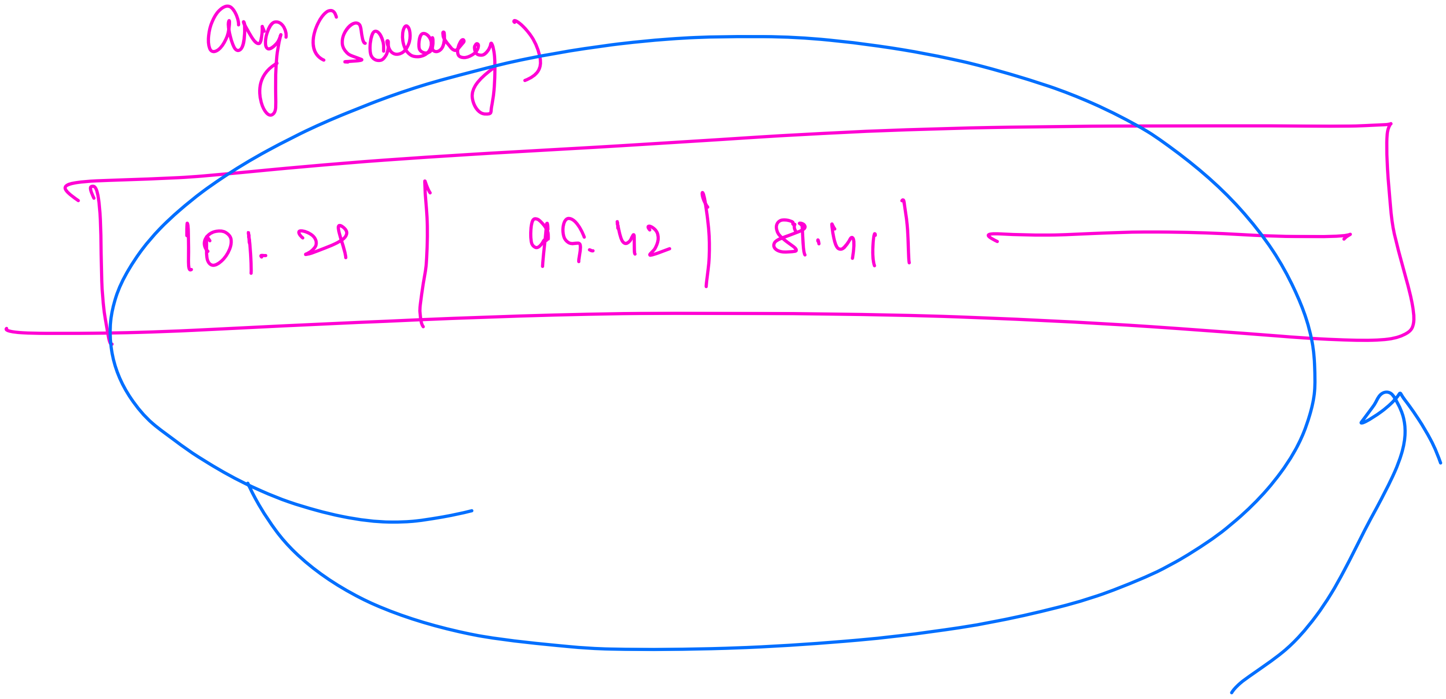
h3(k4) → 8 → 0

n!(k2) →

2×10^9 ops / sec

$$\frac{50}{2 \times 10^9} \text{ sec}$$

avg (salary)



101.21	99.42	81.41	———