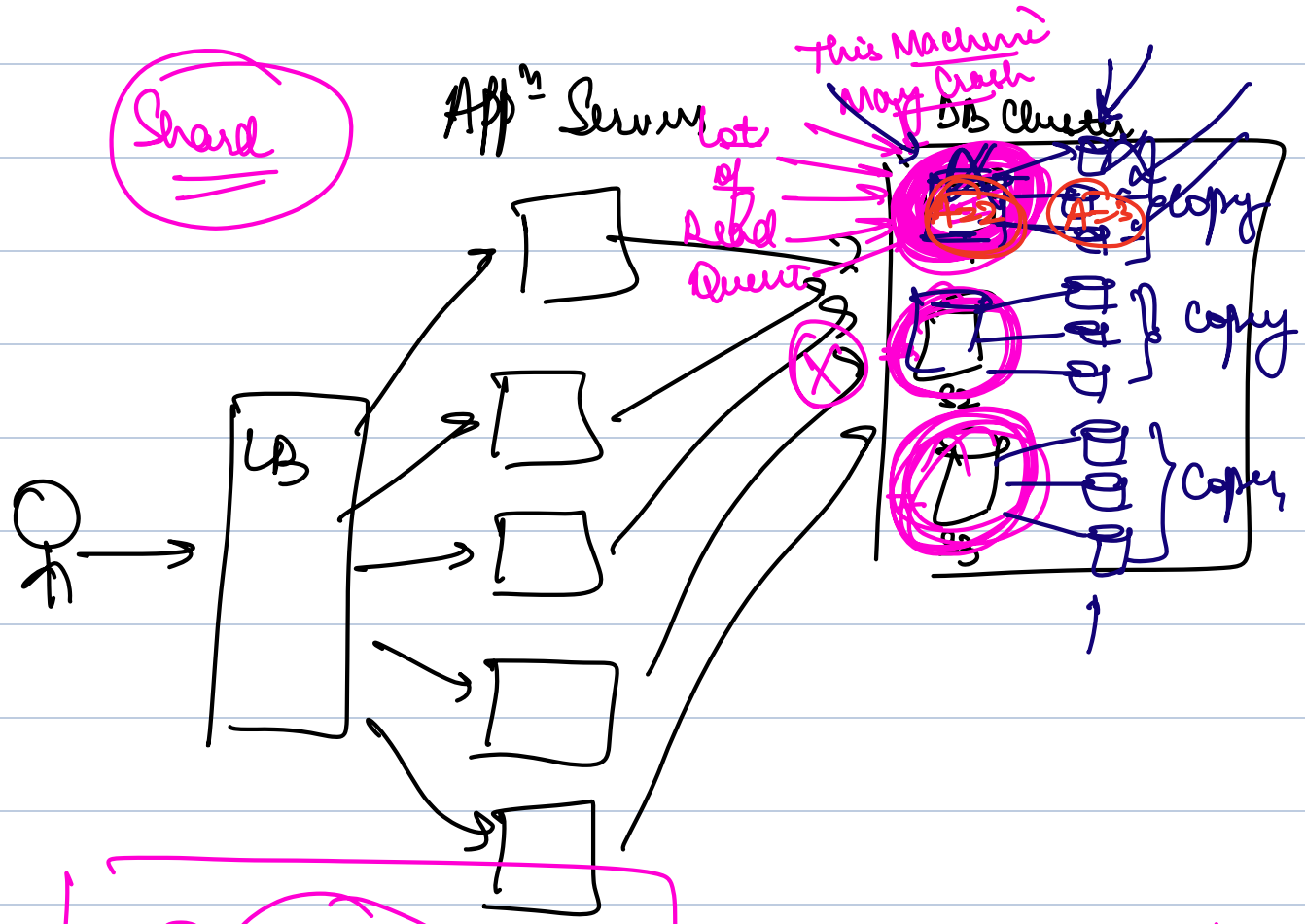
1. Master Slave Architecture
2. Intro to NoSQL Databases
   - ↳ Shortcomings of SQL DB
   - ↳ What are NoSQL DB
   - ↳ Types of NoSQL DB

Master Slave Architecture



1. SPOF
2. Bottleneck
   too many queries

Size of Data ↑ ⇒ More Shards

Queries ↑ ⇒ Replication

Replication

Instead of having data in 1 Machine, Put data in multiple machines.

→ No SPOF

→ Distribute My reads across multiple machine

Can I also dist write → No

→ ↳ all write go to 1 Machine

↳ that machine transfers

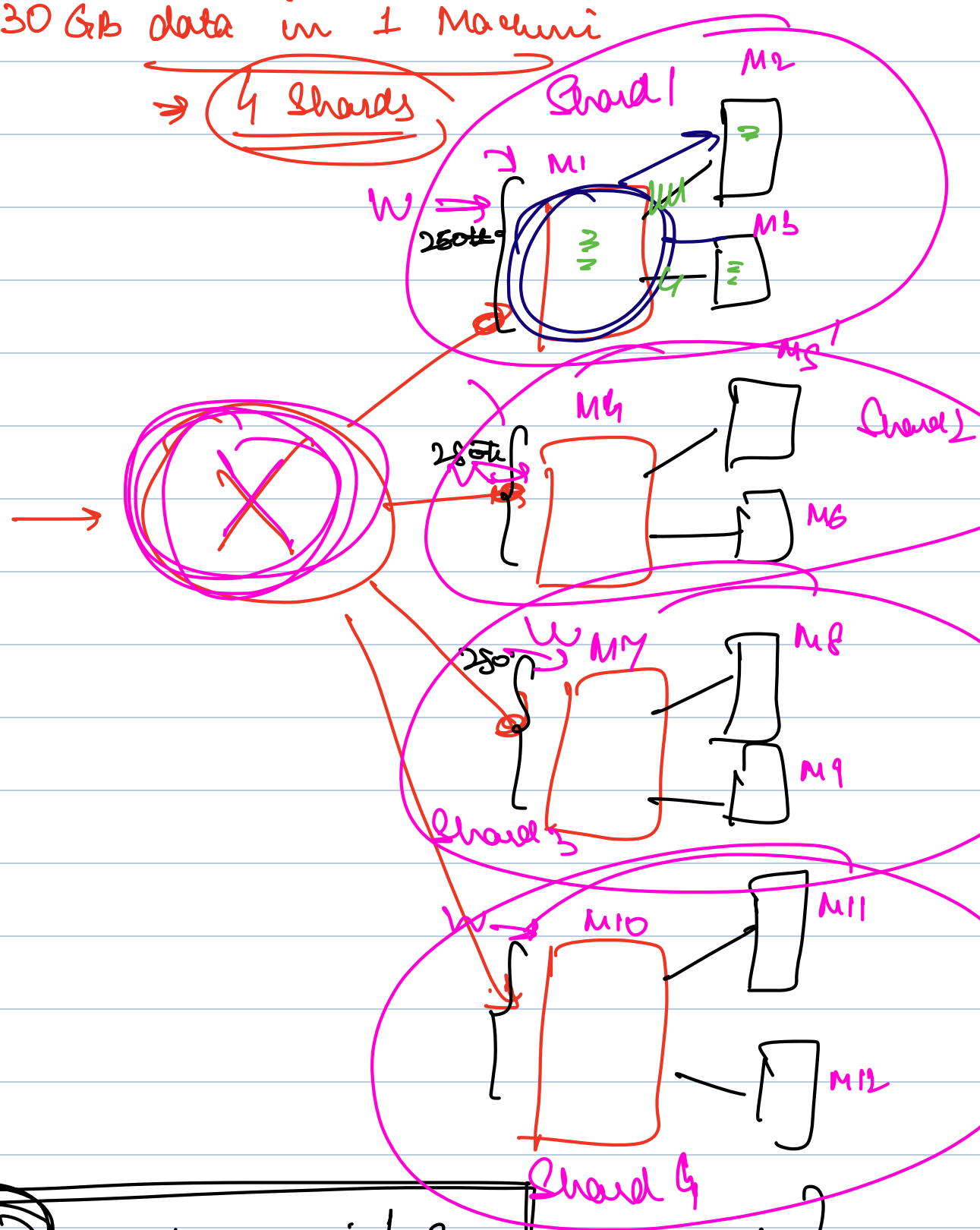
→ the data to all backups

30 + 30 + 30 + 30 = 120

100 GB data

30 GB data in 1 Machine

⇒ 4 Shards



⇒ 1M read queries / sec ⇐ total

1 Machine ⇒ 100 K reads / sec

$\Rightarrow$ $\boxed{250K \text{ reads / sec / shard.}}$

$$\frac{1000000}{4} \rightarrow \underline{250000}$$

atleast 3 machines / shard

$$\text{ceiling}\left(\frac{250K}{100k}\right) = \text{ceiling}(2.5) \Rightarrow 3$$

$\Downarrow$

$$\boxed{\#\text{ of } \underline{\text{Shards}} \Rightarrow \text{Max}^n\left(\text{as per } \underline{\text{size of}} \text{ data, as } \underline{\text{per write queries}}\right)}$$

$$\boxed{\#\text{ of replicas} \Rightarrow \text{max}(2, \text{ as per read seq})}$$

to avoid SPOF

MASTER / LEADER

⇒ machine that gets the write queries from the servers

⇒ and then distribute data to other machines

SLAVES / FOLLOWERS

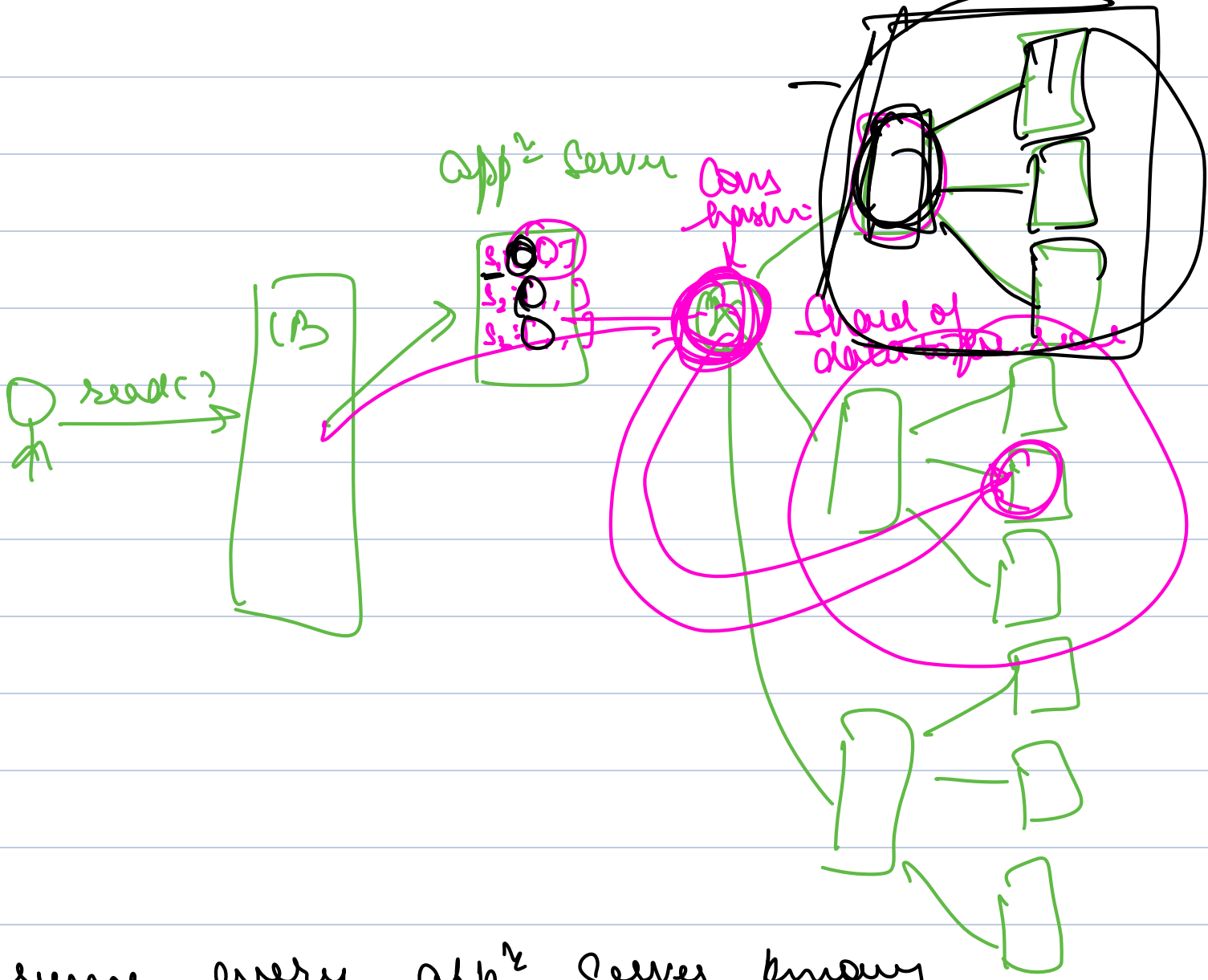
⇒ get the data from master

⇒ acting as redundancy to avoid SPOF

⇒ allow to parallelize reads.

How Masters and Slaves interact

```
read() {
    // appⁿ server knows i/p add of
       all machines in a shard
    ⇒ randomly picks one of them
    → reads from that machine
}
```



Assume every app^n server knows who master of each shard is

Write()

{

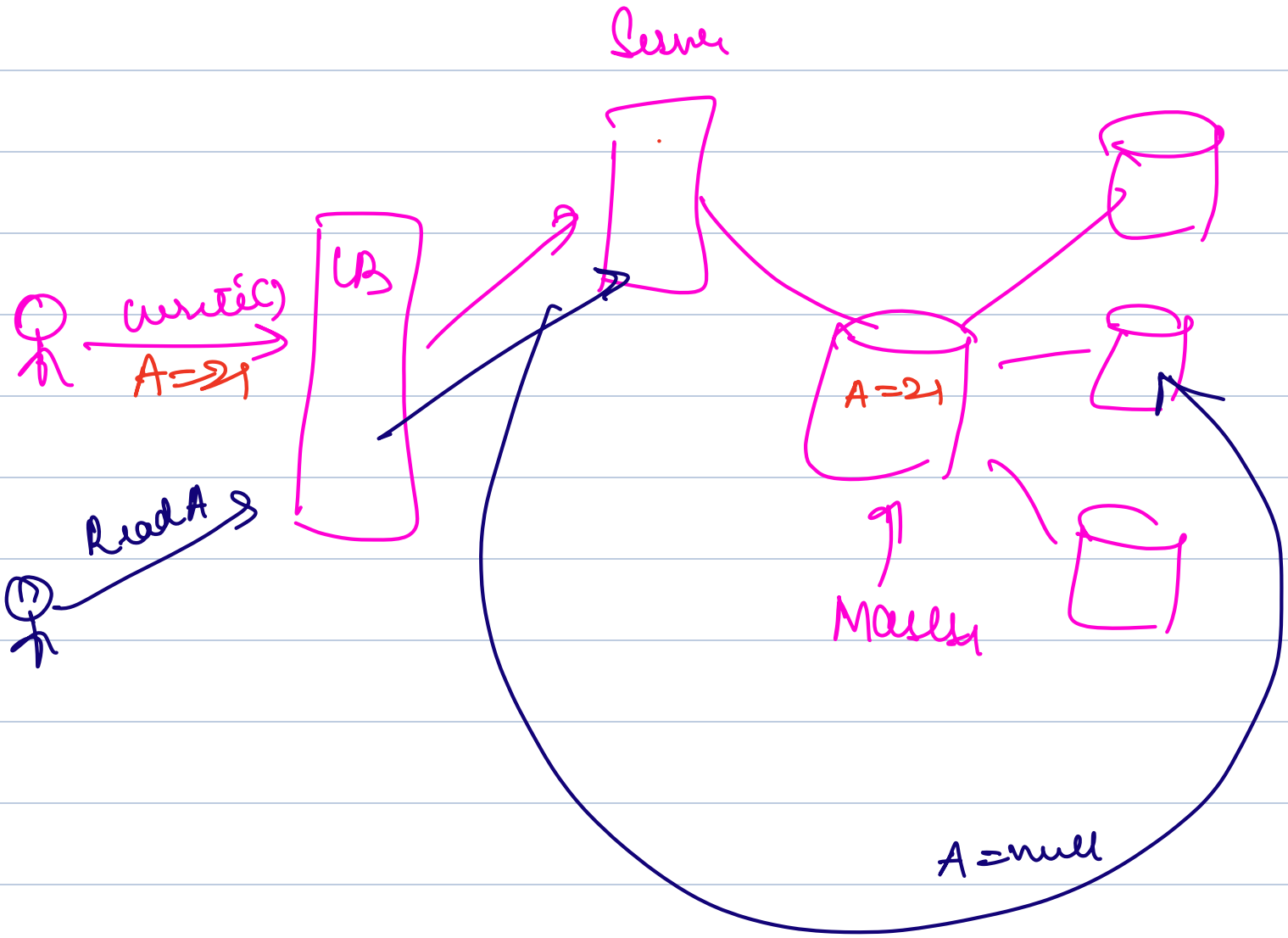
① using C-H. we get to know which shard to write to.

② write to master of that shard.

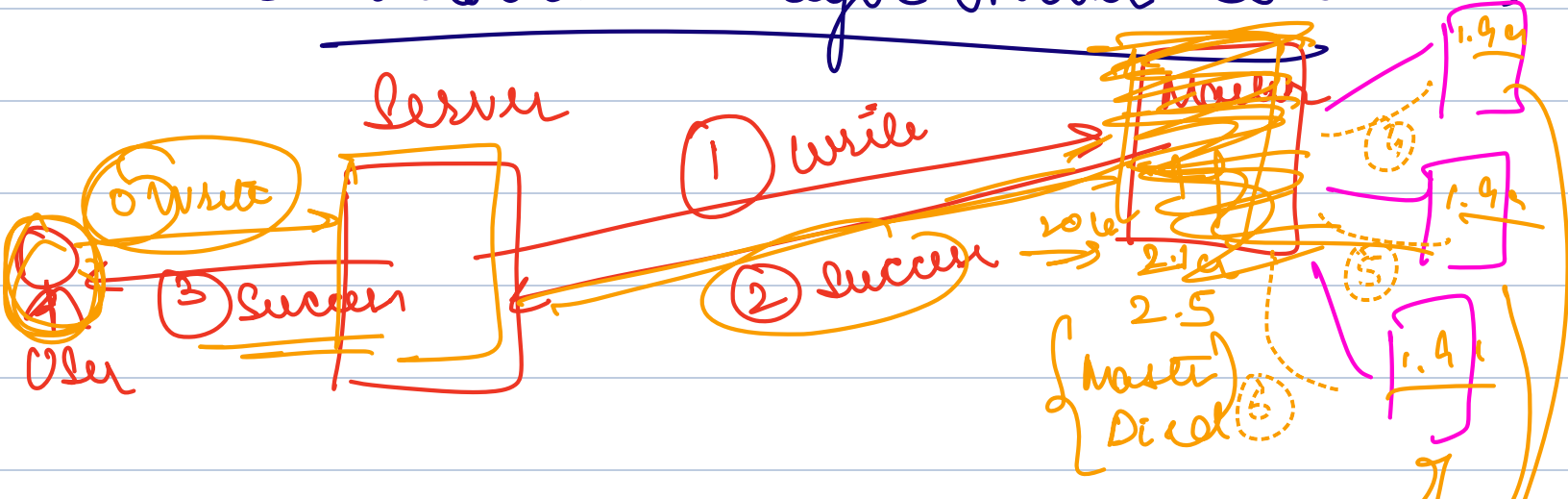
}

Leader Election Algo

ZOOKEEPER



(1) Available and don't care at all about consistency
(Available + Maybe Never Consistent)



Master will async write to slaves

Write ⇒ V.V. Fast.

Reads ⇒ Inconsistent
⇒ Might be never consistent
⇒ Might end up ~~losing data~~.

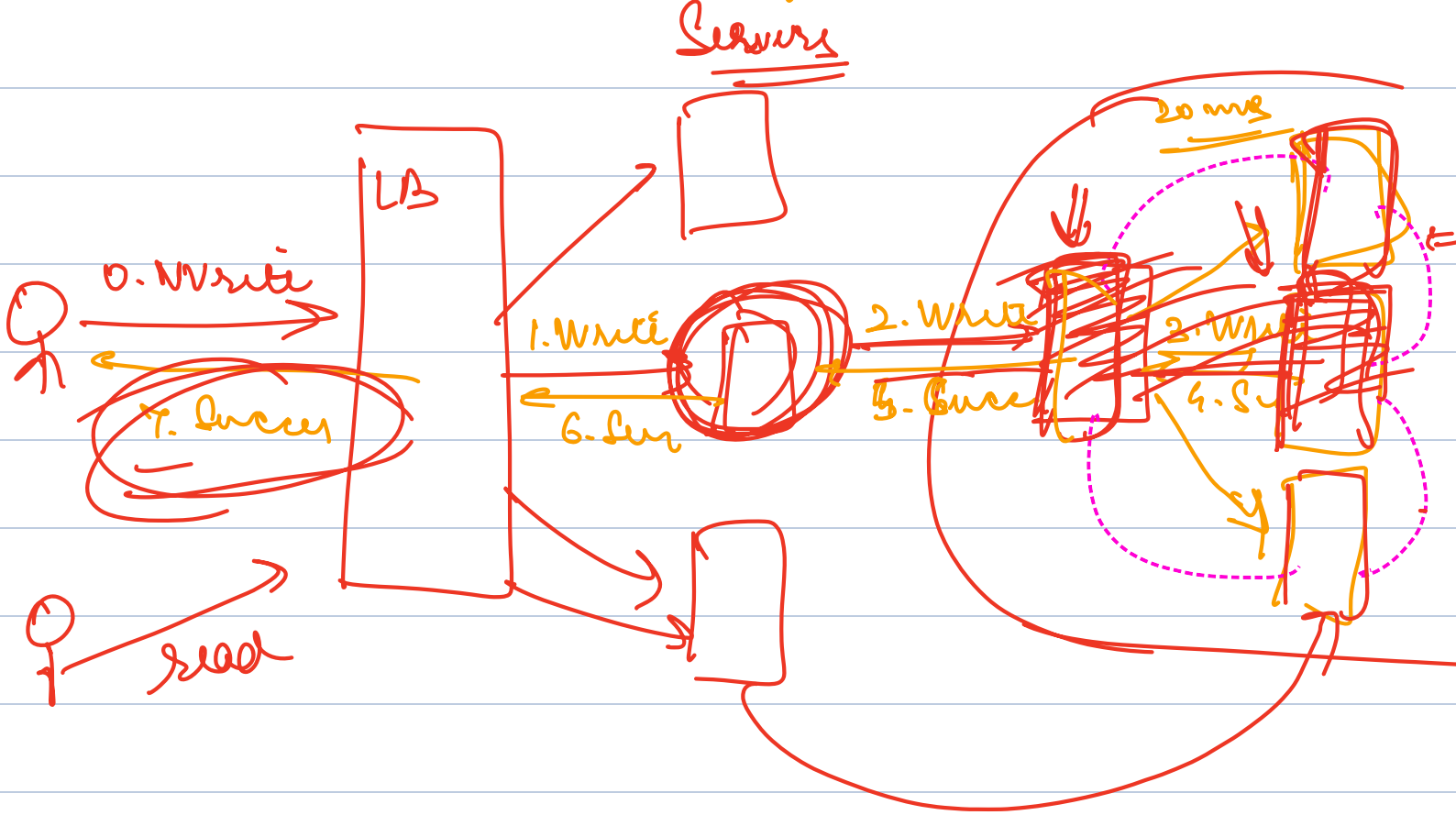
Use Cases :

Analytics
↳ Hotstar View Count
⇒ Care about high level numbers, not exactness

(2) Inconsistent
⇒ But eventually consistent ****

↳ Likes Count

**** : V.V.V. ~~V~~ High Prob.



⇒ Master waits to atleast ~~1/2/3~~ 2 slave before confirming the write to server

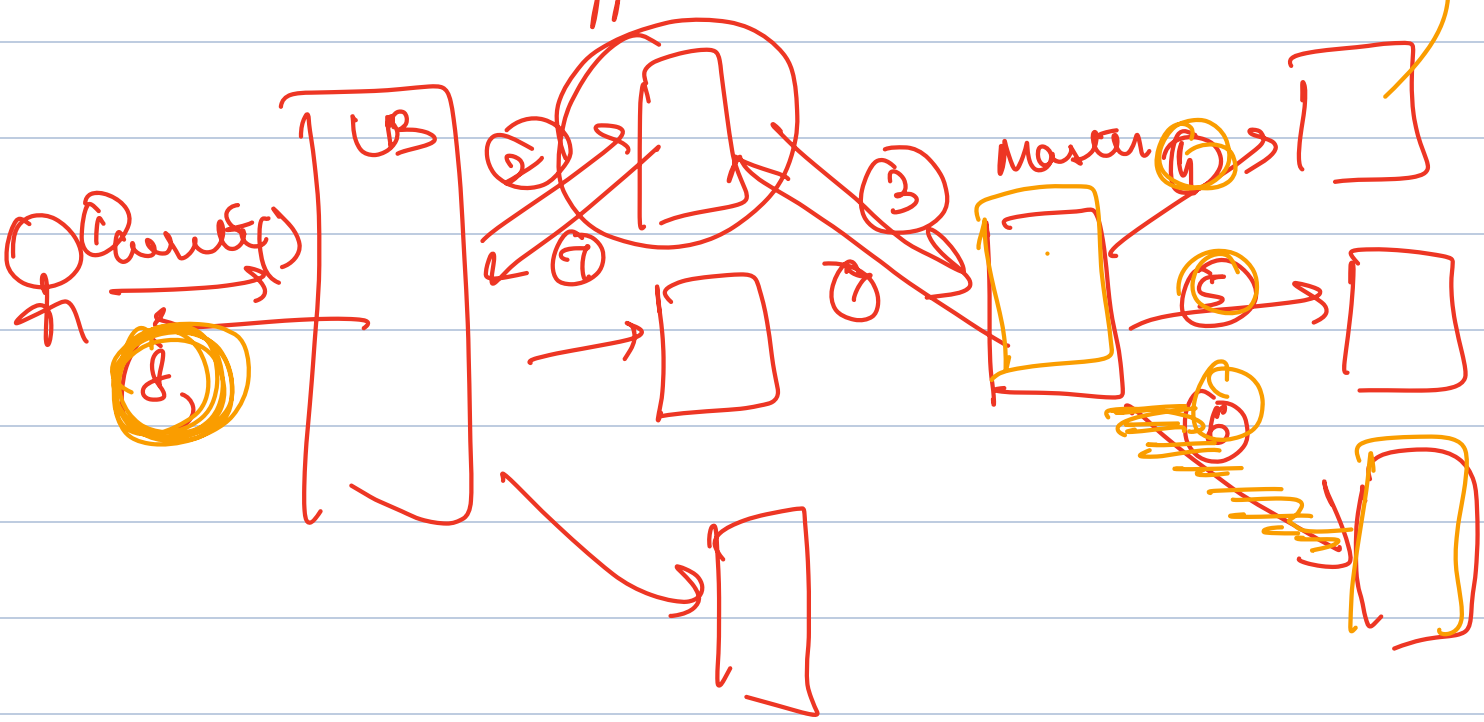
⇒ Async other slaves will sync.

0.0001

0.000001

3 Consistent

⇒ write to all slaves before returning success.



⇒ Never inconsistency

But

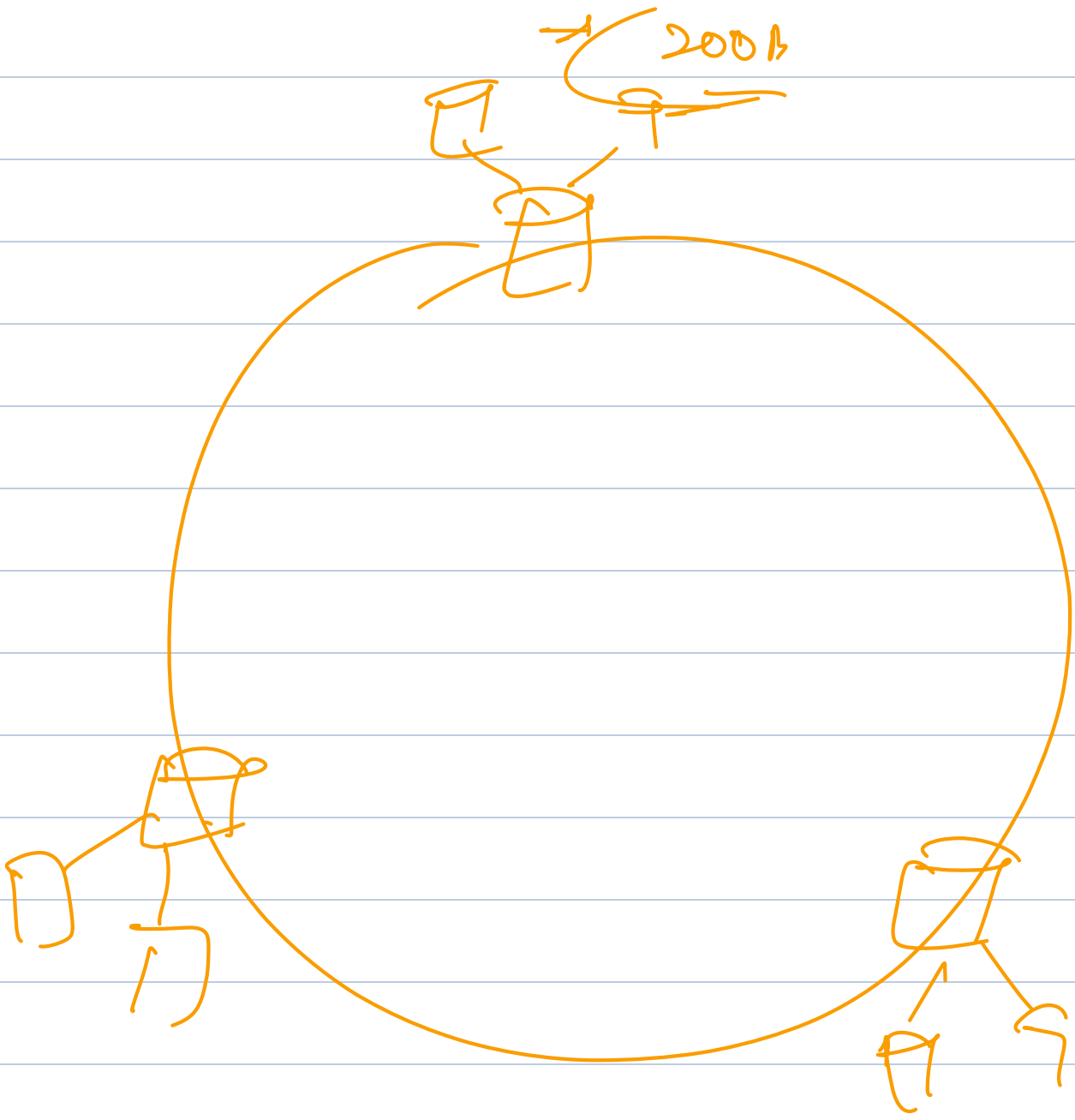
1.) High Latency

10 Shards

⇓

5 s/ Shard → 50 i/p add

→ 4 Bytes / i/p



Intro to NoSQL DB

SQL Databases

↳ Relational Databases

→ ↳ Structured Data

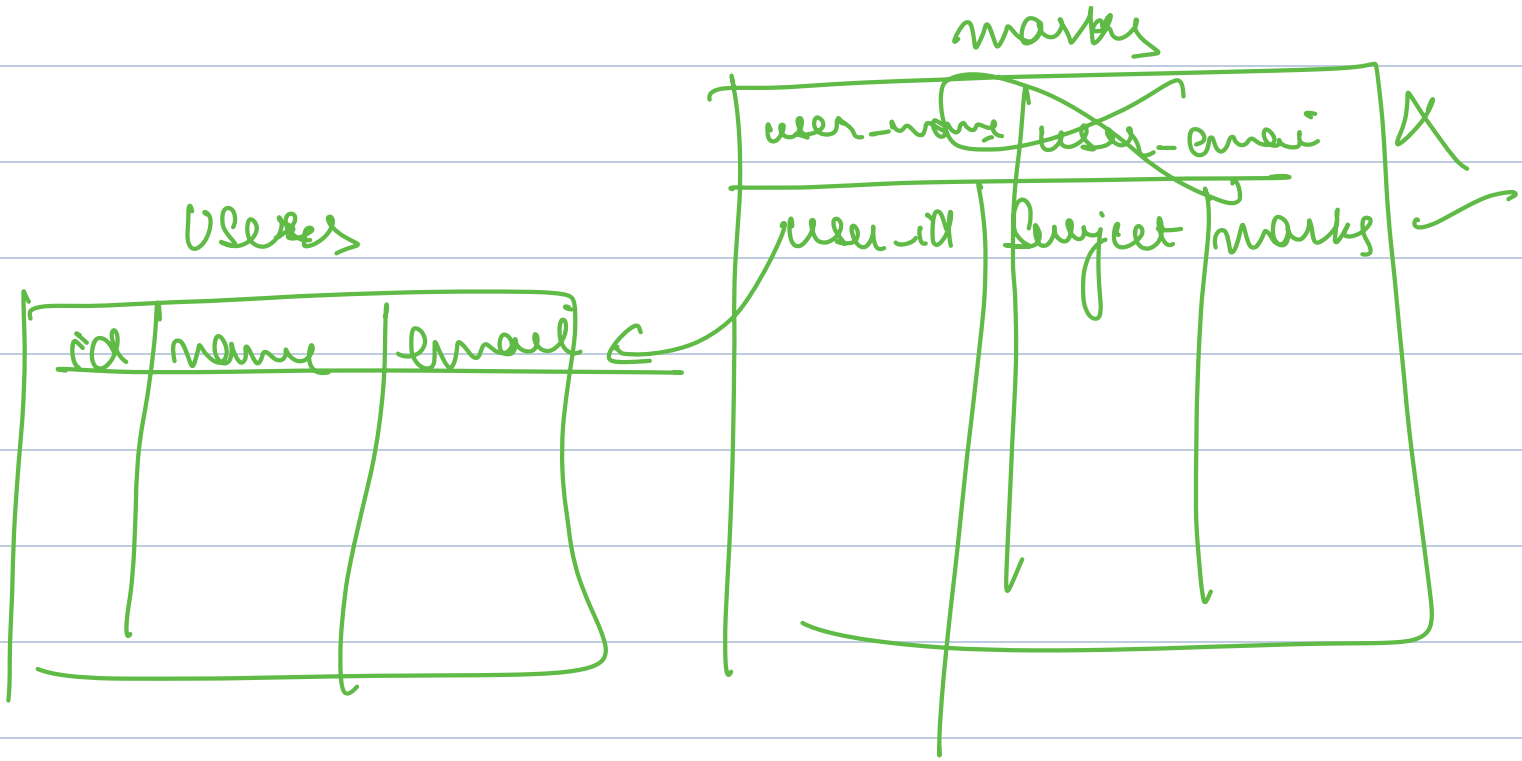
→ ↳ Relations/info entities.

↳ joins

when we store data in SQL,

we try to avoid redundancy

⇒ we store data in normalized form



1. Structured Data
2. Joins
3. Normalized Way
4. ACID Transac

does every use case need all of these?
↳ in some cases, these things may become a problem.

~1.9 a

Case 1 ECOM app^n

→ Products

id	title	desc	price	name	OS	fabric	brand	collar_type

currently

⇒ every prod-type may have its own set of attrs
→ putting all attrs will lead to wastage of storage
→ Sparse table.

Q^n 2

⇒ we will create diff table for diff types of products

10K categories

Clothes

phones

adapter

Shirts

pants

⇒ SQL DB aren't great in handling so many table.

⇒ queries become difficult.

Joins

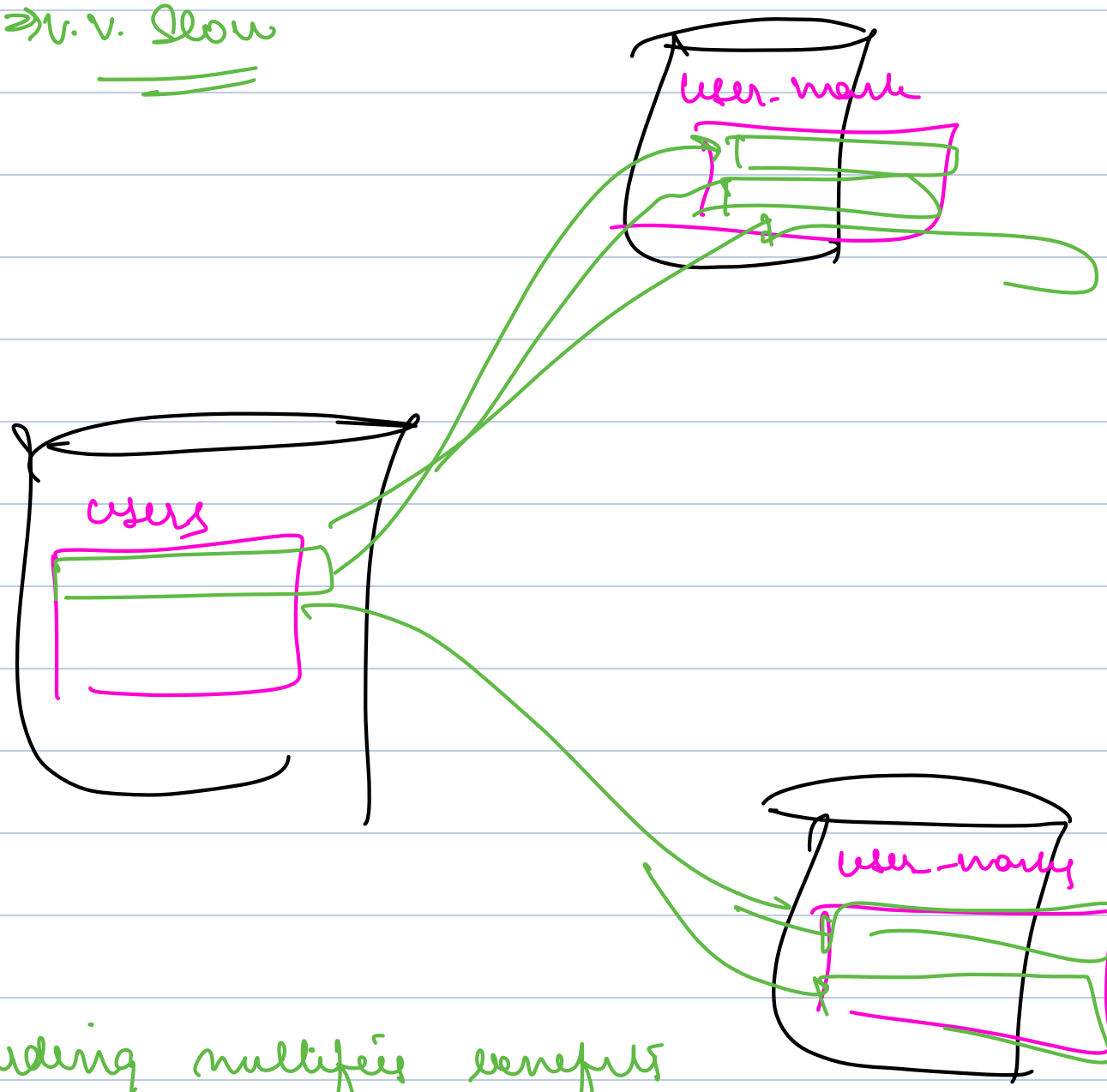
⇒ is data in 1 machine

⇓

Sharding

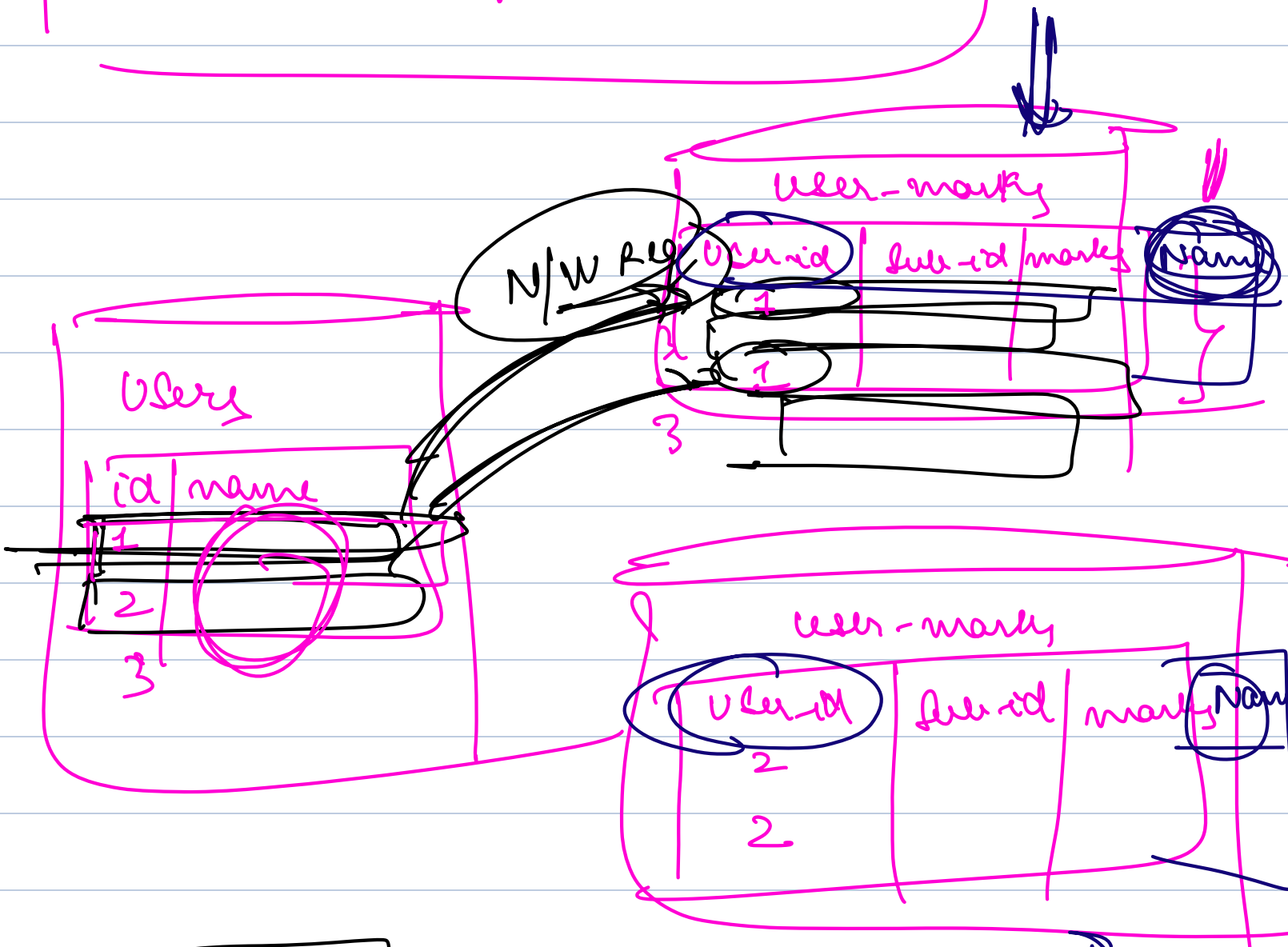
⇒ N/W req

⇒ v. v. slow



⇒ Sharding nullifies benefits of joins.

User-name	Avg (marks)



→ N/W req for each row of user

↓

N.V. Slow

```
Select name, avg (marks)
    from user-marks
    group by name
```

→ M1

UNION

```
Select name, avg (marks)
    from user-marks
    group by name.
```

→ M2

ACID

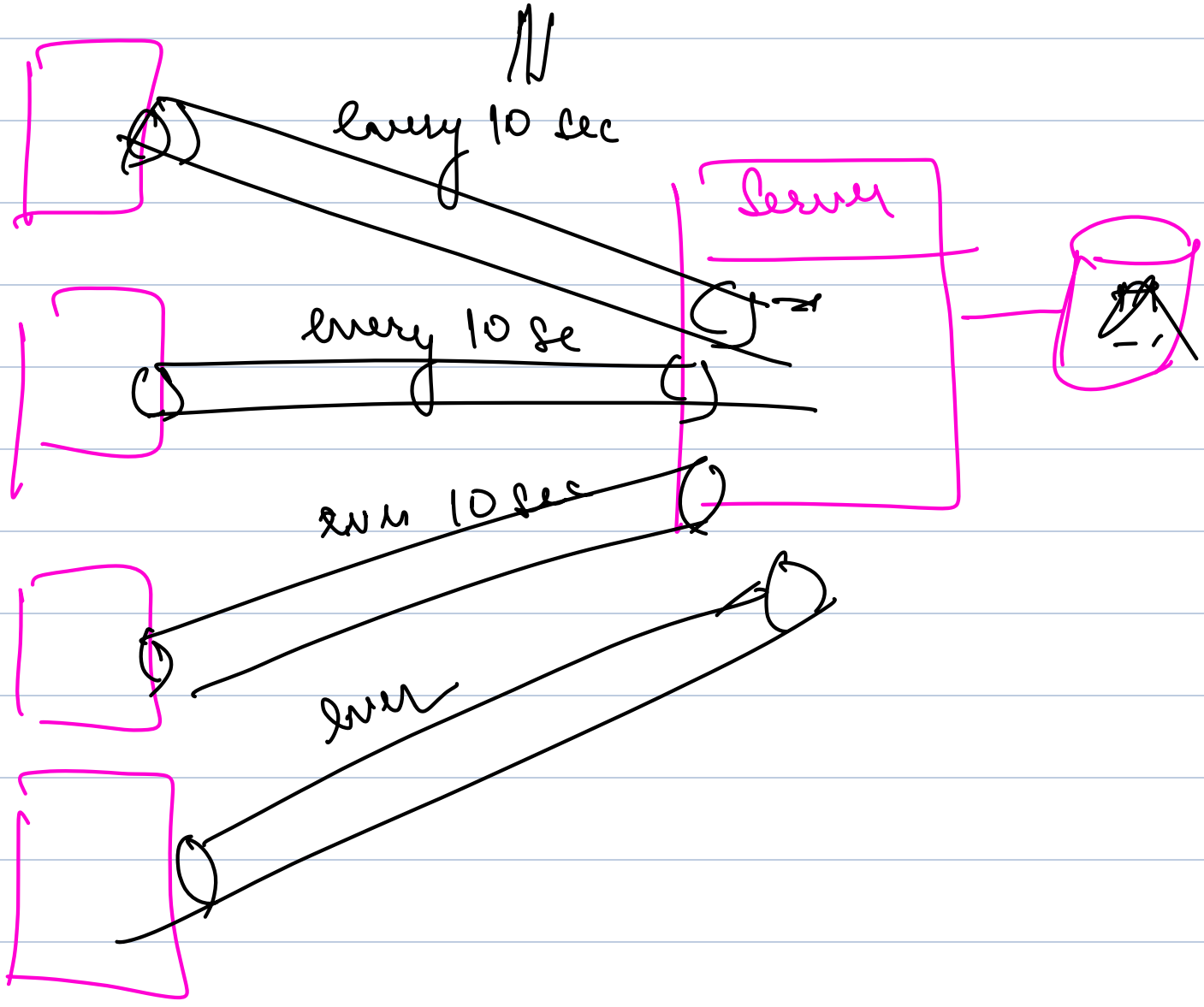
→ Disk → Slow

→ Durability

↳ once you have stored data, it is gonna stay

→ RAM → Fast

Live viewer counter



→ Some scenarios

→ only store data in cache

live counter

NewNode

New Node

c = 1 → 2

Locks → slow

exact Count = 3

Shown Count = 2

Q → Paid 5000

Company told failure

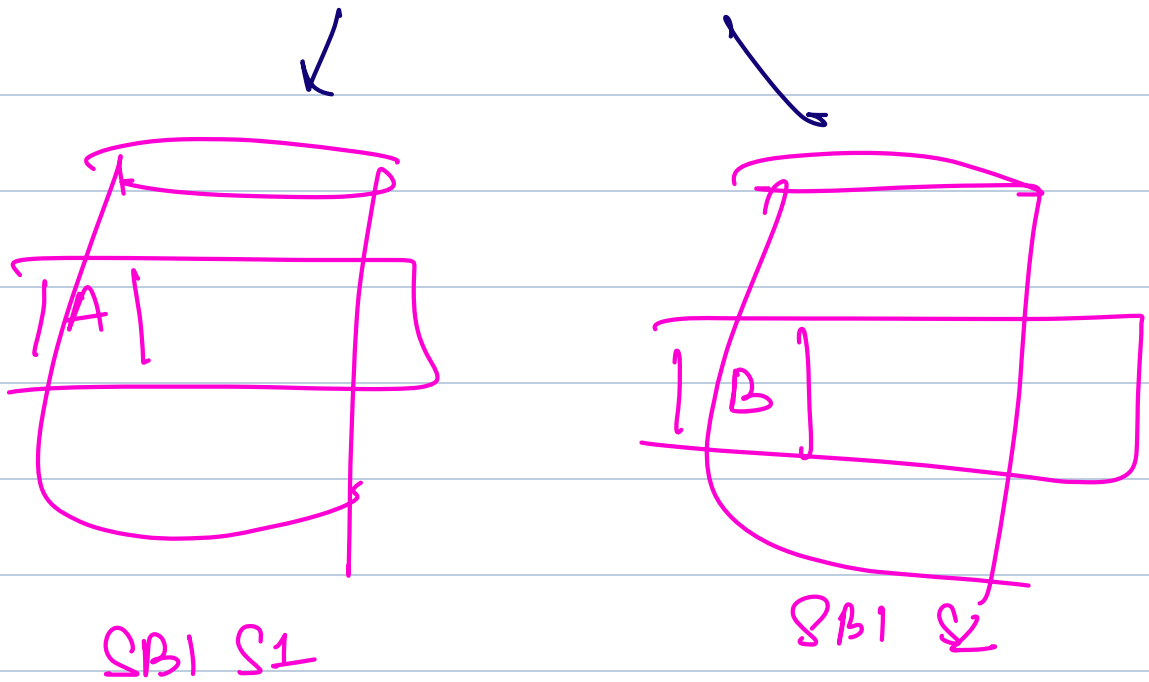
after 7 days → +5000

SAGA

everything will happen

or

nothing will happen



$\rightarrow$ | A $\rightarrow$ B $\rightarrow$ 50,000

a += 1

Read A $\rightarrow$ t

t = t + 1

write (A, t)

= always prefer SQL DB

unless your use case has some needs that SQL has disadvantages.

NoSQL

internally sacrifice one or more properties of SQL DB to give some other benefits

Redis ⇒ NoSQL DB

20MS

= lets go durability

↳ achieve fast speed =

Vicuner - Comket

match_views

match_id	count

99

```
update match_count
    set count = count+1
    where match_id = { }
```

Chatting Appⁿ

10k live chatting

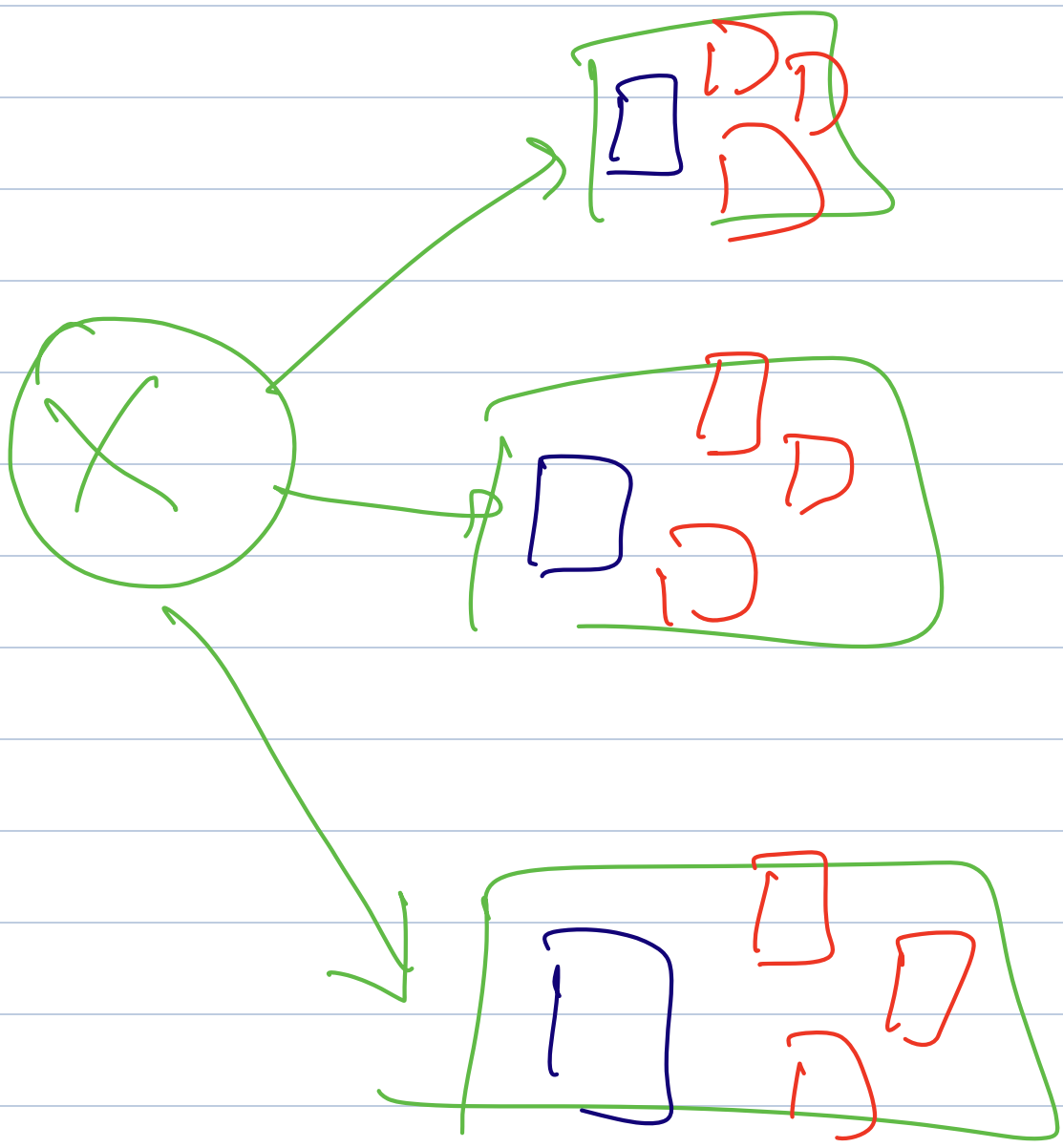
↳ PostgreSQL DB

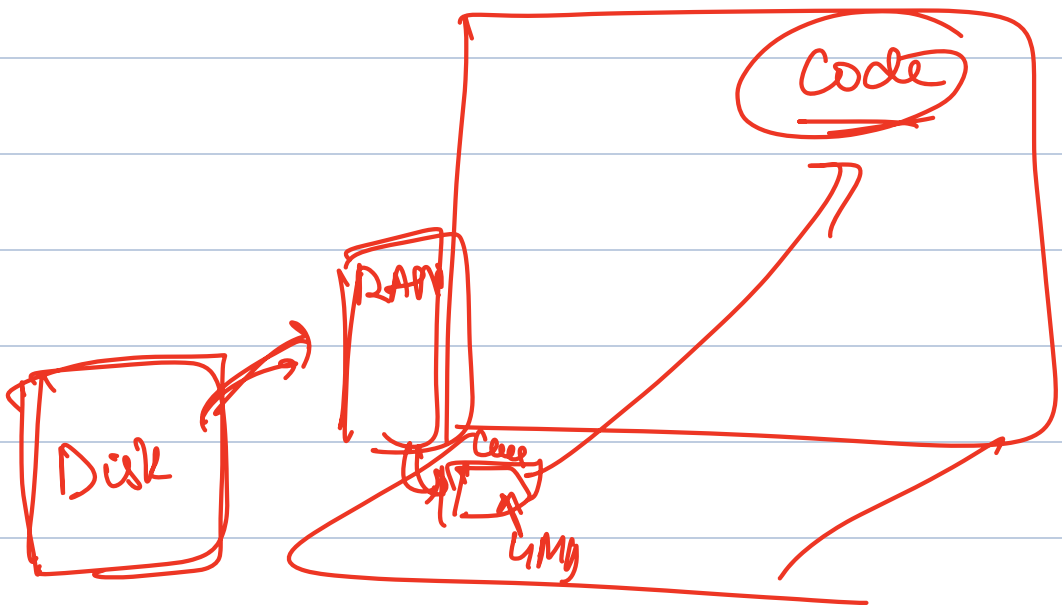
↳ Cassandra

print (time)

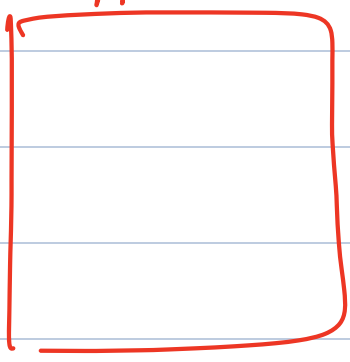
$\equiv$

print (time)



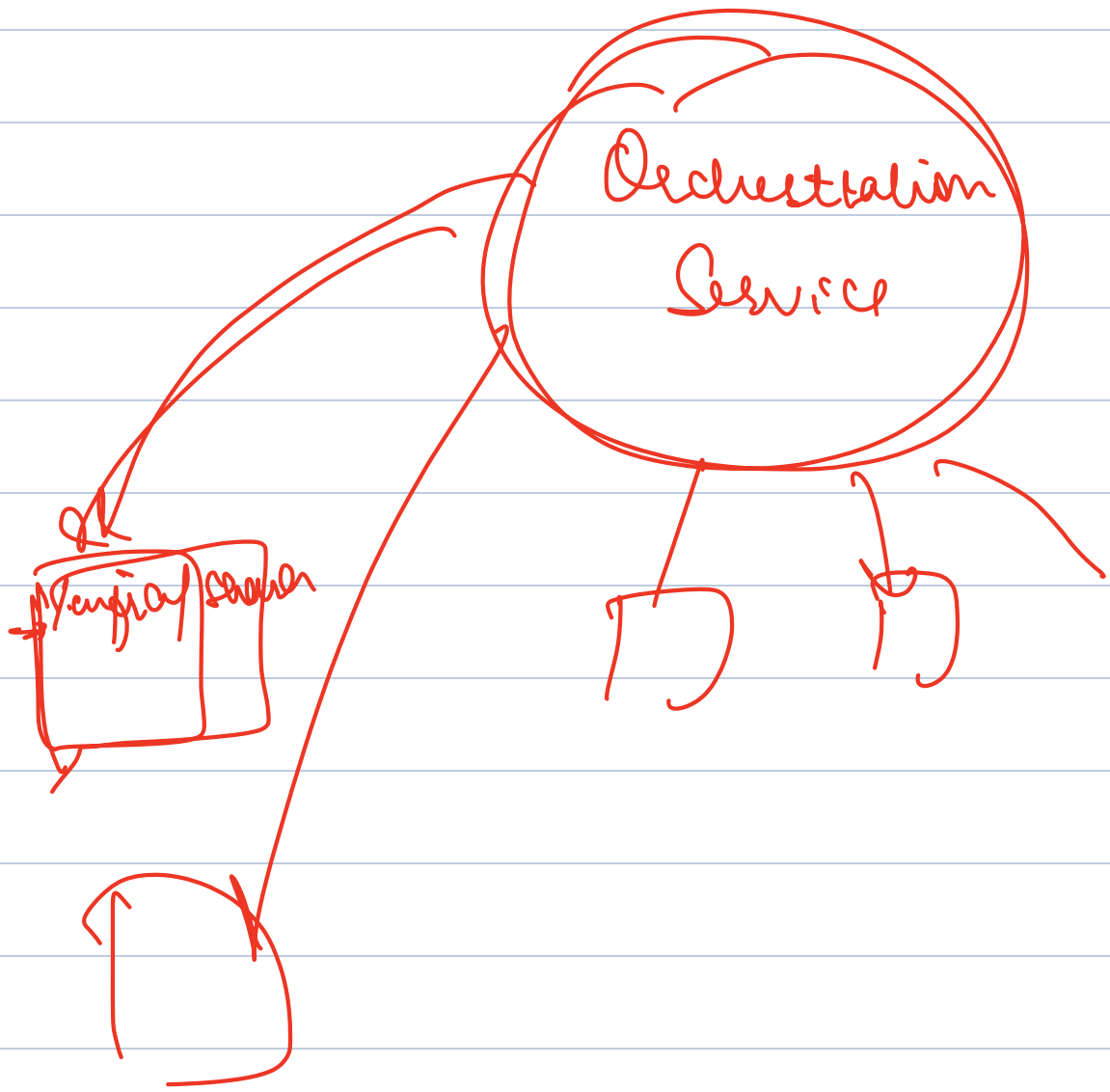


App Server



MySQL

```
write (data) {
    db. write (data)
    cache. write (data)
}
```



1st

last 30 day

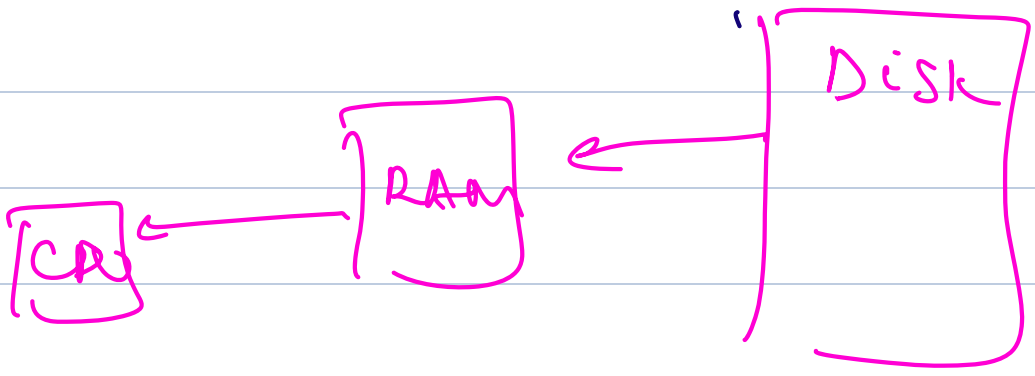
user →	[0, 0, 0, 0]

User	Feed
0, →	pid Content [0, 0, 0, 0]

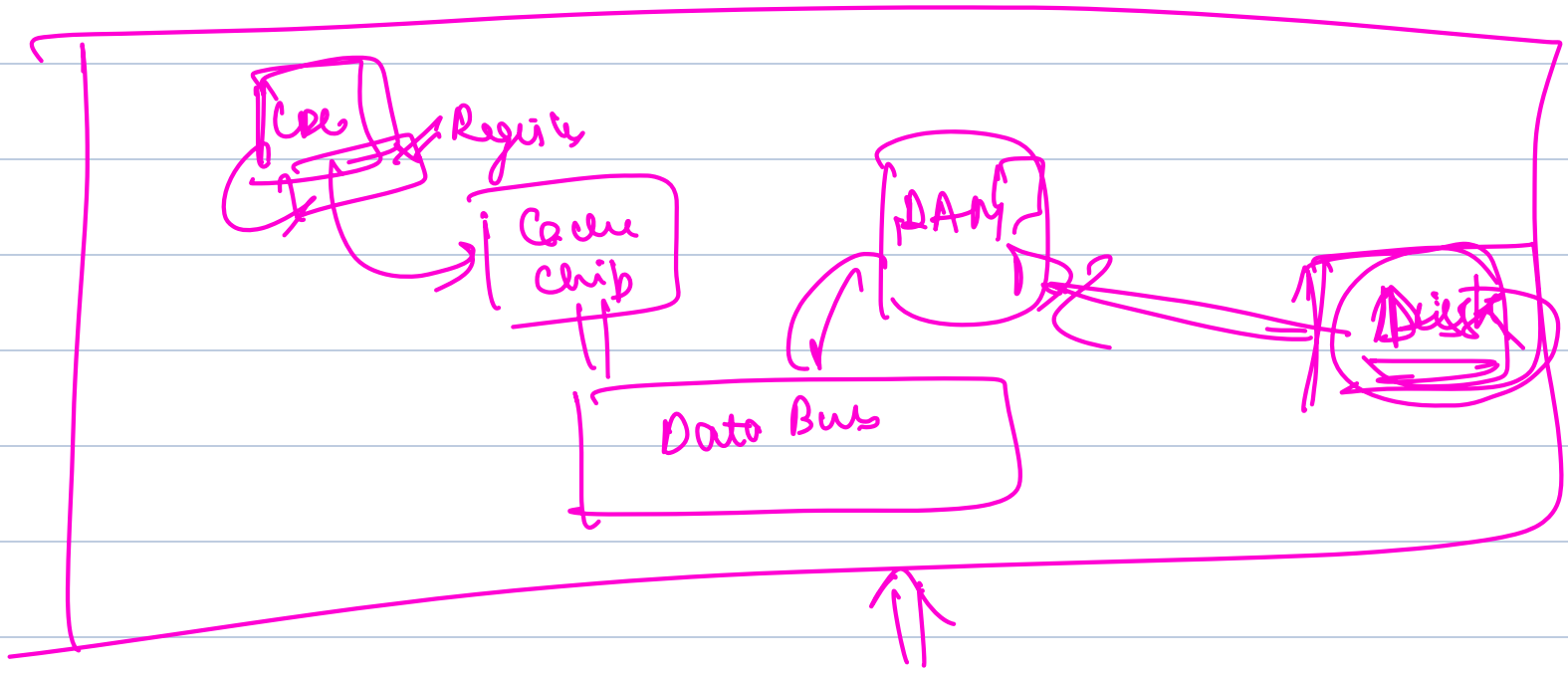
(pid, uID)

[0, 0, 0,]

IOS



2-SC


CPU
Registers
Cache chip
RAM
Disk
Data Bus