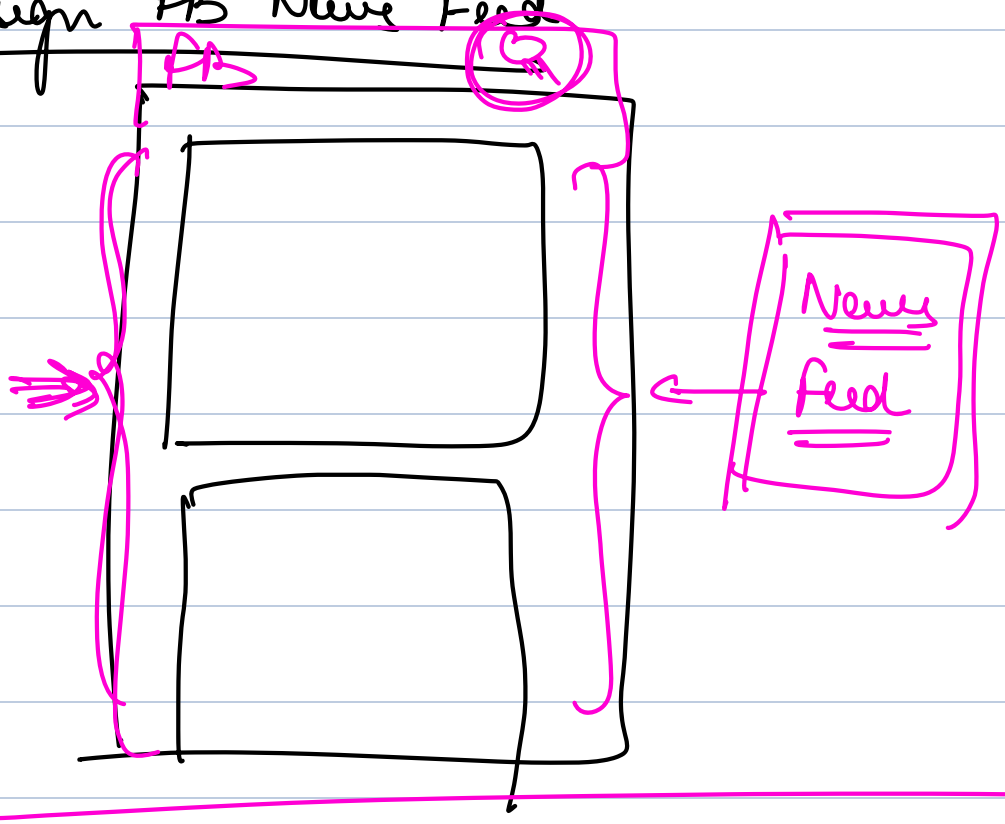
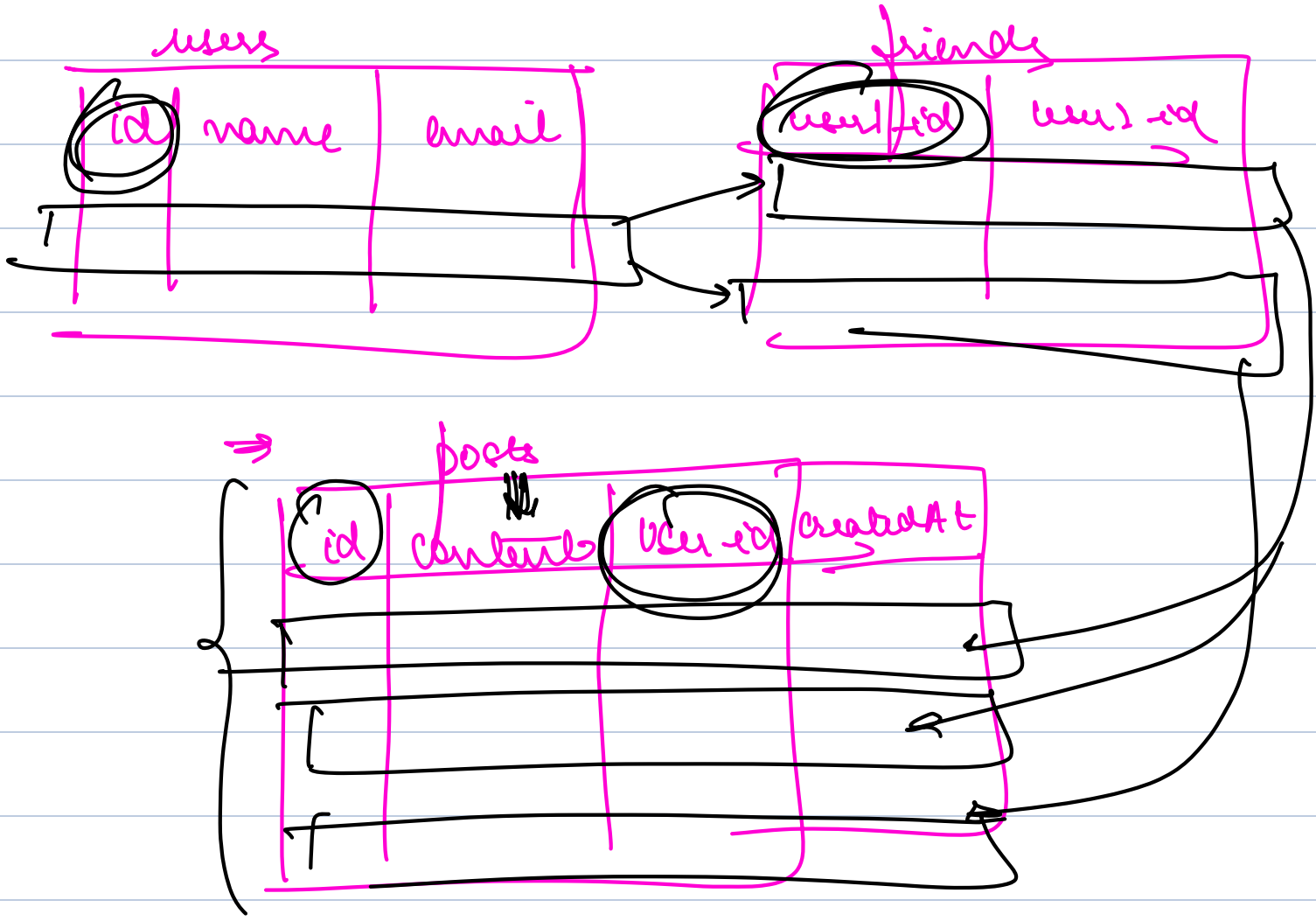
Will start at 9:10

1. Design FB News Feed ✓ — Caching Complete
2. CAP Theorem ✓
3. PACELC Theorem ✓

Design FB News Feed

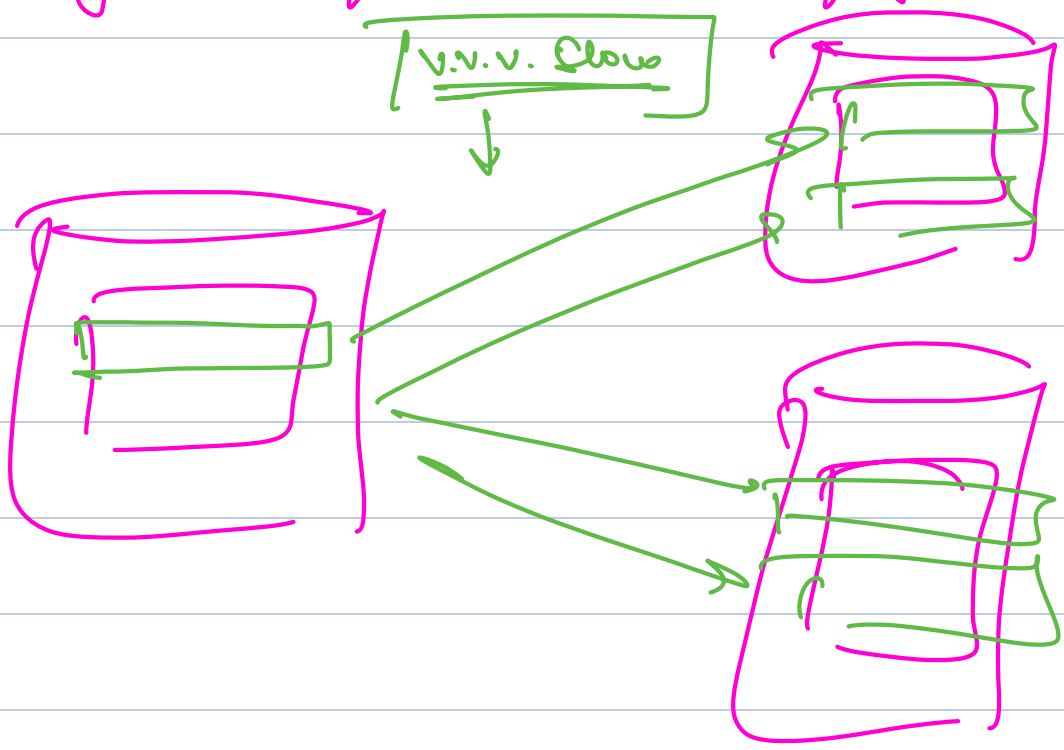


2004: Chronologically ordered set of posts made by your friends




```
select *
    from users u
        join friends f
            on f.user1_id = u.id
        join posts p
            on f.user2_id = p.user_id
order by createdAt DESC;
```

Sharding nullifies benefits of joins



doing joins across mfw v.v. slow

Assume we shard by user id

Users

id
1

friends

user_id	user2_id
1	2
1	7
1	8

posts

id	content	U1
1	Hello	1
31	I am going	1
507	Hey!!	1

⇒ we need to pre compute data

↳ Caching

Sugg^n 1 Pre compute the news feed of every user

Cache

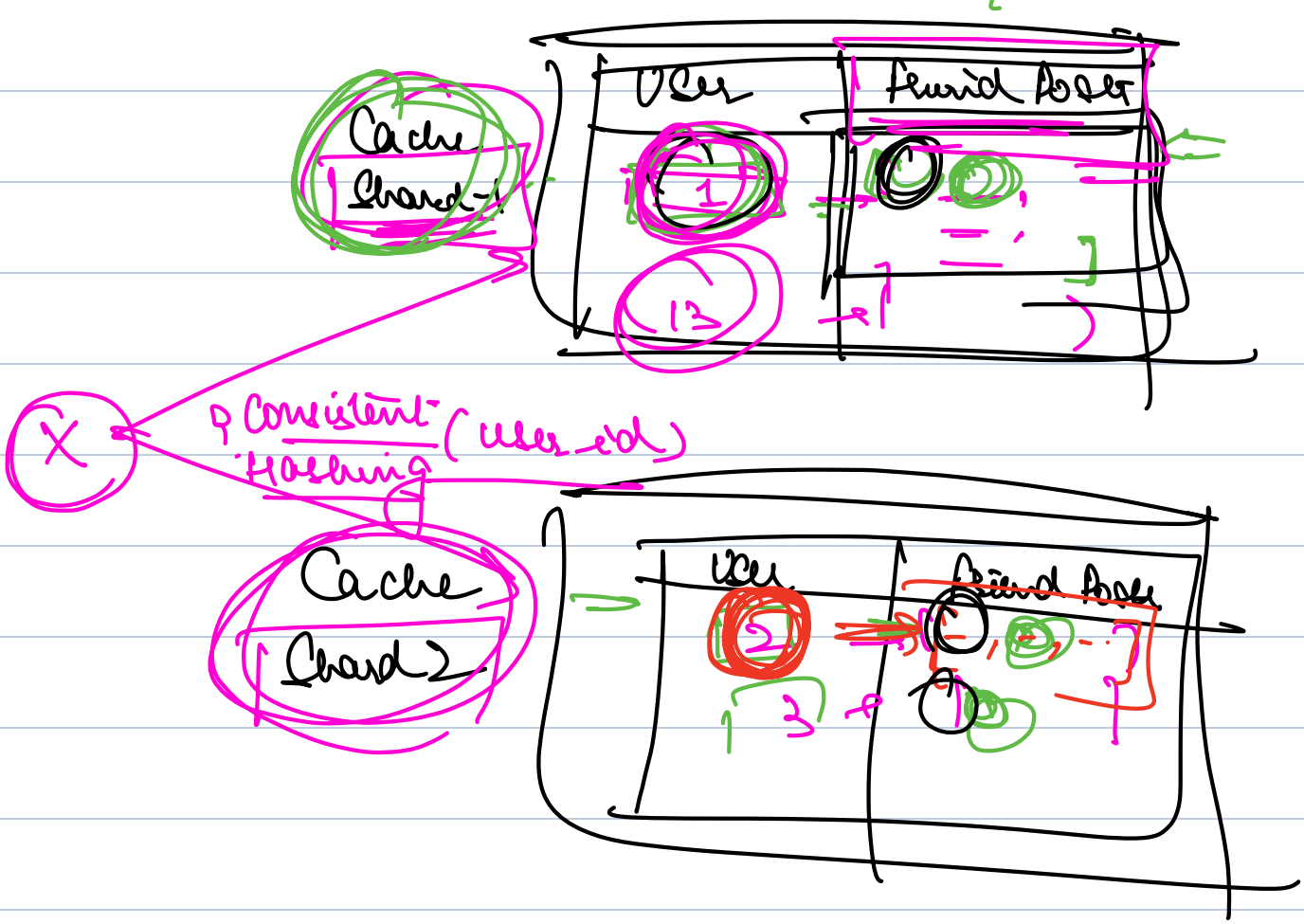
Key	Value
UserID	posts made by friends of that user
1	[0, 0, 0, 0]
2	[0, 0, 0, 0, 0]
3	[0]

Shard

30 day posts

or

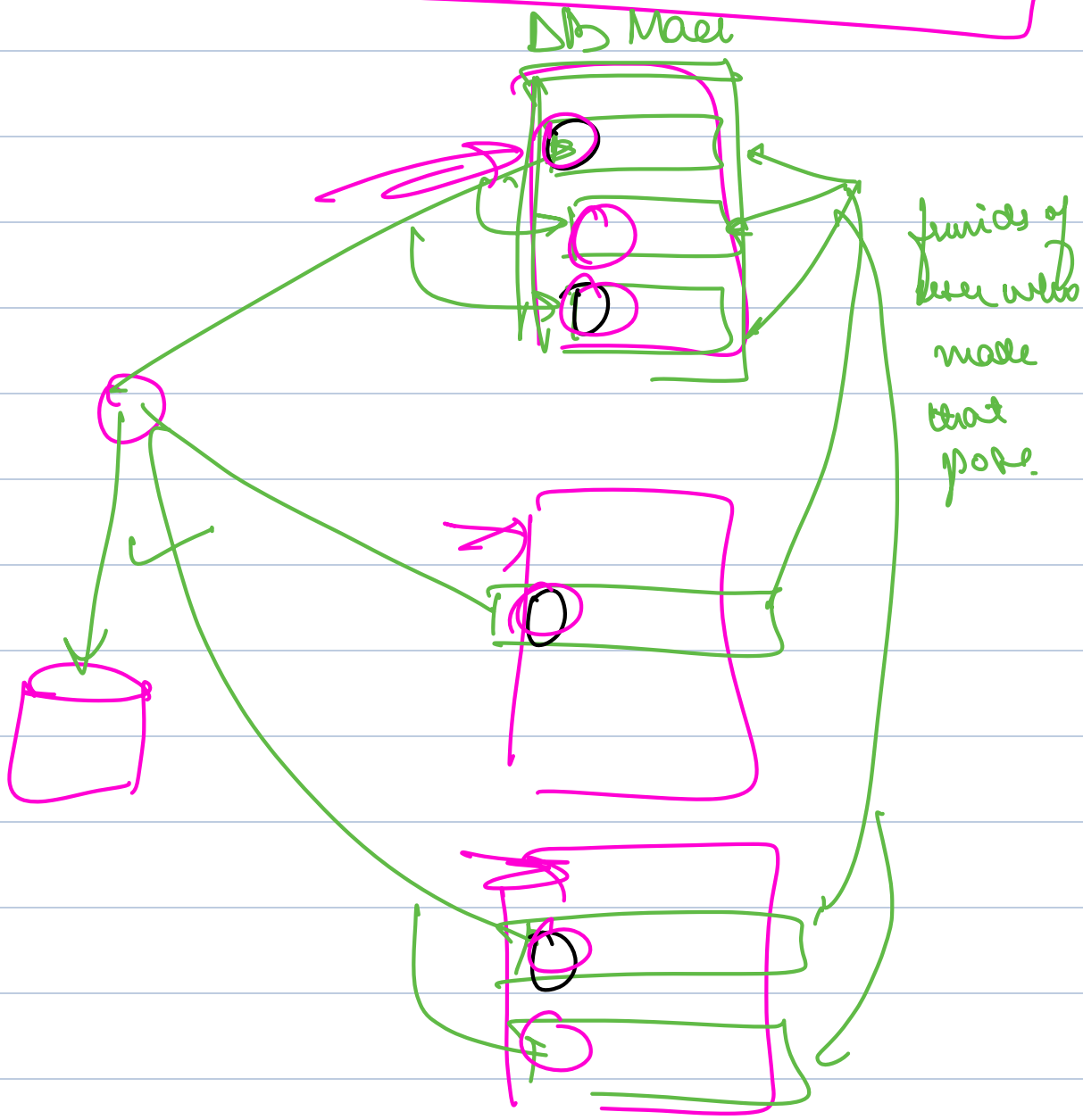
10000



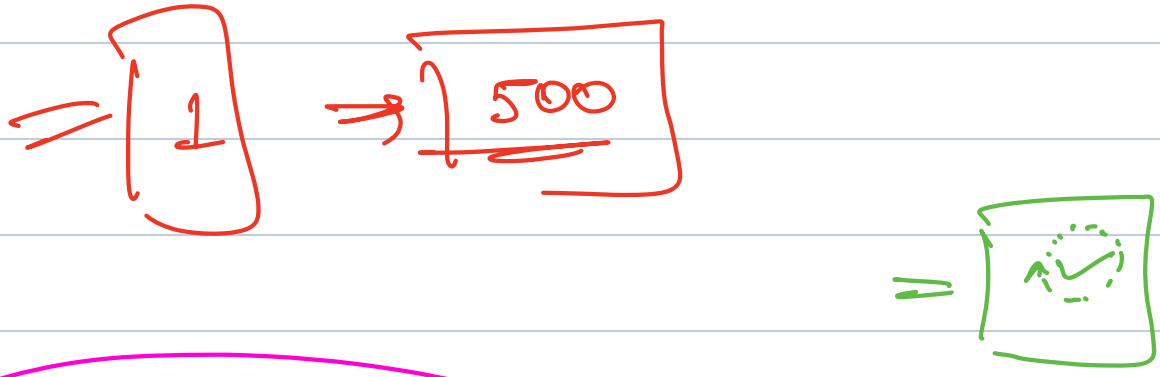
Get Shard (Sharding Key) → Shard #

```
Create Post (UserID, content) {
    db.save (in posts table)
    List<Friends> db.get (friends of UserID)
    for each friend:
        S = get Shard (friend)
        S.get (friend).append (post)
```

}




```
getNewsFeed(userID) {
    l = getBoard(userID)
    l.get(userID)   <= O(1)
}
```



CELEBRITY PROBLEM

Narendra Modi makes a post

100 Mn

(s) will have to insert the post in 100Mn keys

→ DB overload !!!

↳ fail the DB

Option 2

Last 30 day posts

⇒ 99% of people will never scroll beyond posts made in last 30 days

↳ even if it takes more time to load posts > 30 day old, we are okay

FB ⇒ 2B DAU

⇒ 1% of them make post

⇒ 20 M posts a day

⇒ $20 \times 10^6 \times 500B$

⇒ $10000 \times 10^6 B$

⇒ $10 \times 10^9 B$ → 10GB

Post

→ id ⇒ 8

→ user ID ⇒ 8

500B

→ created At → 8

→ content → 300
(300 char)

→ 30 days

= 300 GB post

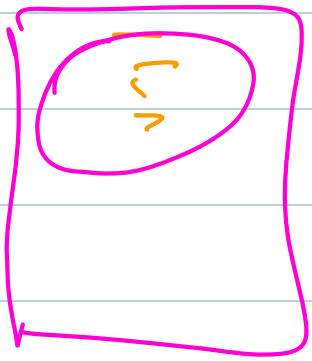
= 15 TB hard disk

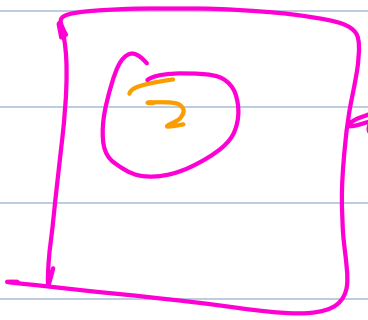
256 GB RAM

Cache (High Memory Machine)

500 GB
1 TB

each machine has all posts made in last 30 days





CPU
→ 3GHz → 8 core

GPU
{ 400 MHz
500 cores }

=

(10)

(DB)

writes ⇒ 1 + 1 per cache machine

Create Post (Post content)

1 → db.save(post) ← 1

1 → cache.put(user ID, post)

↳ cache is "Replicated" across many machine

get Feed (user ID) {

⇒ List<Friend> db.getFriends(user ID)

(Randomly go to 1)

cache.get

⇒ [k.value where k in friends]

}

sort() and reverse

Cache

user	posts by user in 30 day
U_1	[ ]
U_2	[ ]
U_3	[ ]

all users

all posts in last 30 day

$\boxed{O(\#\text{friends})}$

↳ can't have more than 5000 friends

$n = \boxed{5000} \times 30 = \boxed{5000}$

$\boxed{\text{mergn} \neq 10\text{ms}}$

Caching whole feed for every user's

{ user ID → [posts of all my friends]

exp of people who have a lot of followers

Now, caching posts of a user

user ID → [posts made by user]

5000 { people who have a lot of friends their exp will be bad.

Instagram | Twitter

Hy brid

Create Post (user ID, content) {

if user ID is a celebrity.

op = 2

else :

op = 1

}

}

2B DAU

→ avg 20 times

→ 40B news feed

→ $\frac{40 \times 10^9}{10^5}$ → 400000

CAP Theorem

→ Fundamental constraint of dist systems

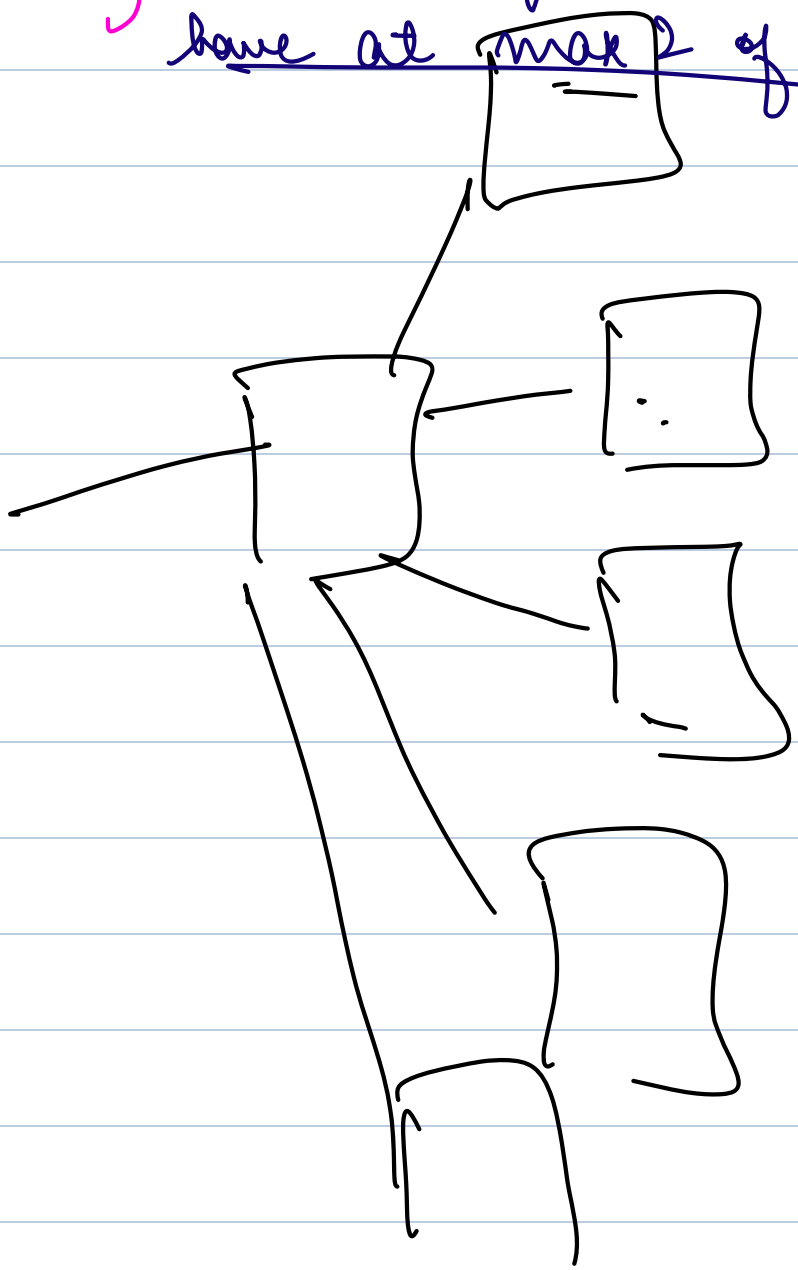
C → Consistency

A → Availability

P → Partition Tolerance

⇒ No S/W system can have all of these 3

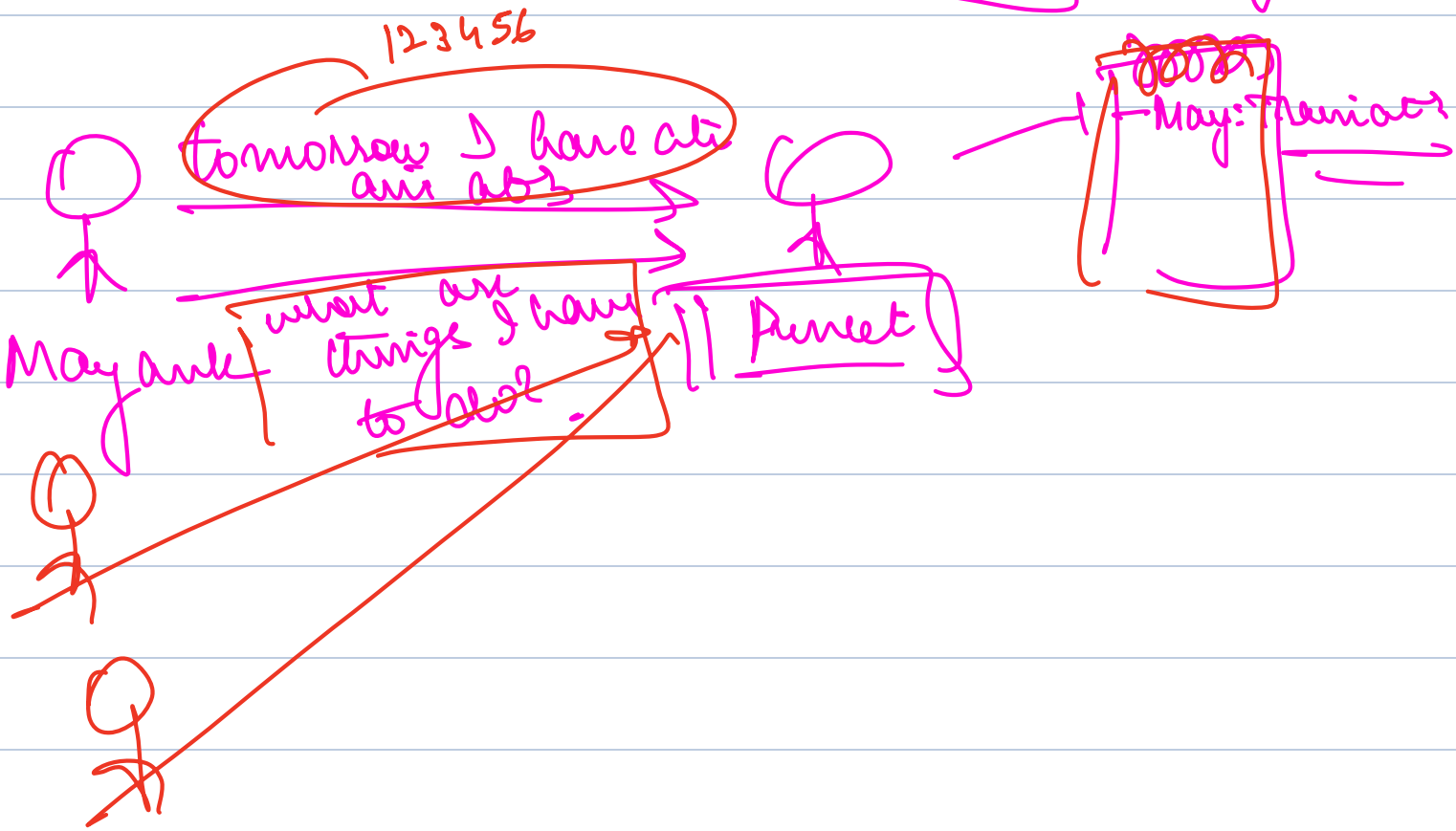
⇒ A S/W system can have at max 2 of these



1960

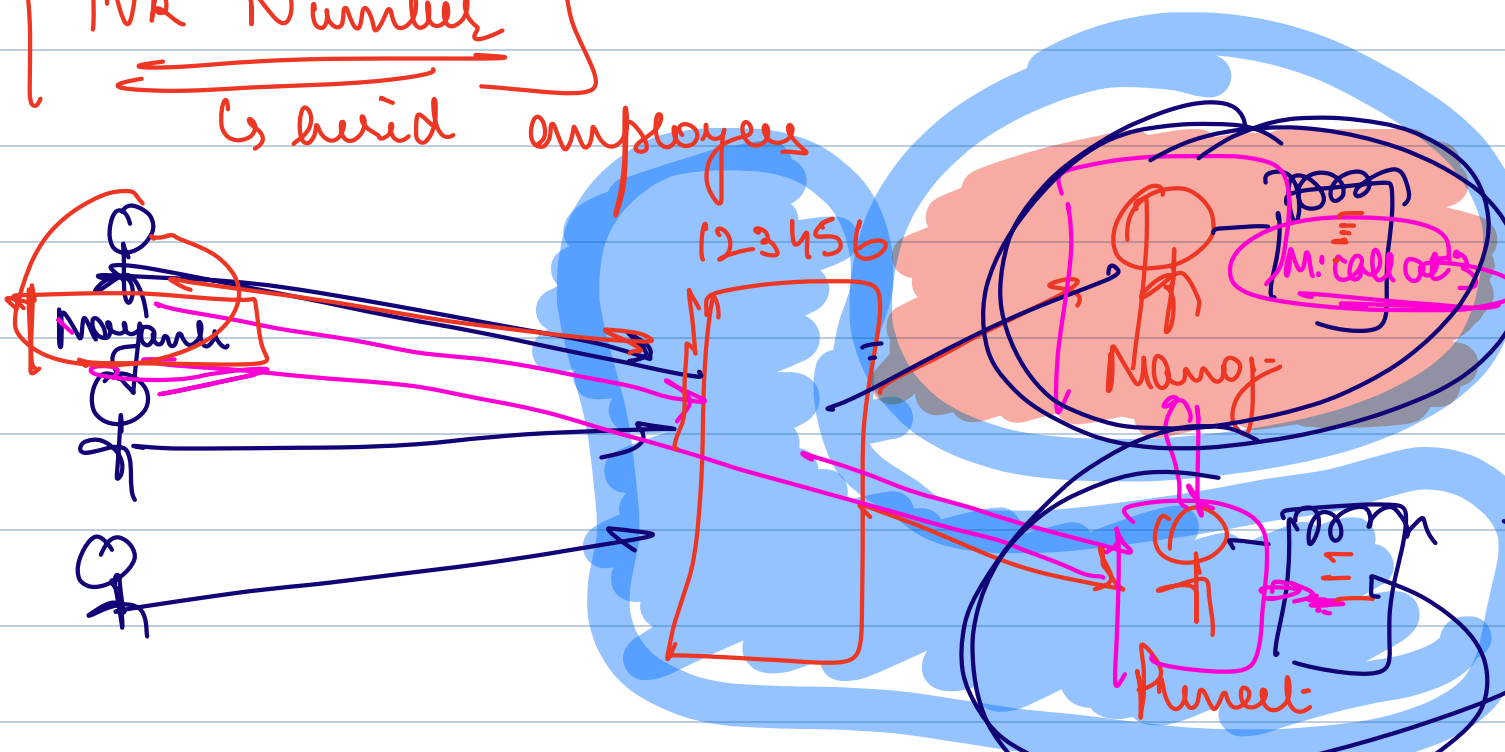
Puneet

↳ Remind Me Company



IVR Number

↳ hired employees



Manoj had a figure

2 Parts

Mayank calls → call goes to Puneet
→ anything for me?

2 opⁿ for Puneet — No reminder for you

Q ⇒ (i) he gives the partial answer (Basis what he knows)

⇒ (ii) he says : Sorry! Something is wrong. Please connect later.

(i) ⇐ PB News Feed

Company was available.
But company may not give correct info.

(ii) ⇐ Remind Me, Bank

Company is less available.

But always complete, correct info.

C → Consistency : Complete, correct data

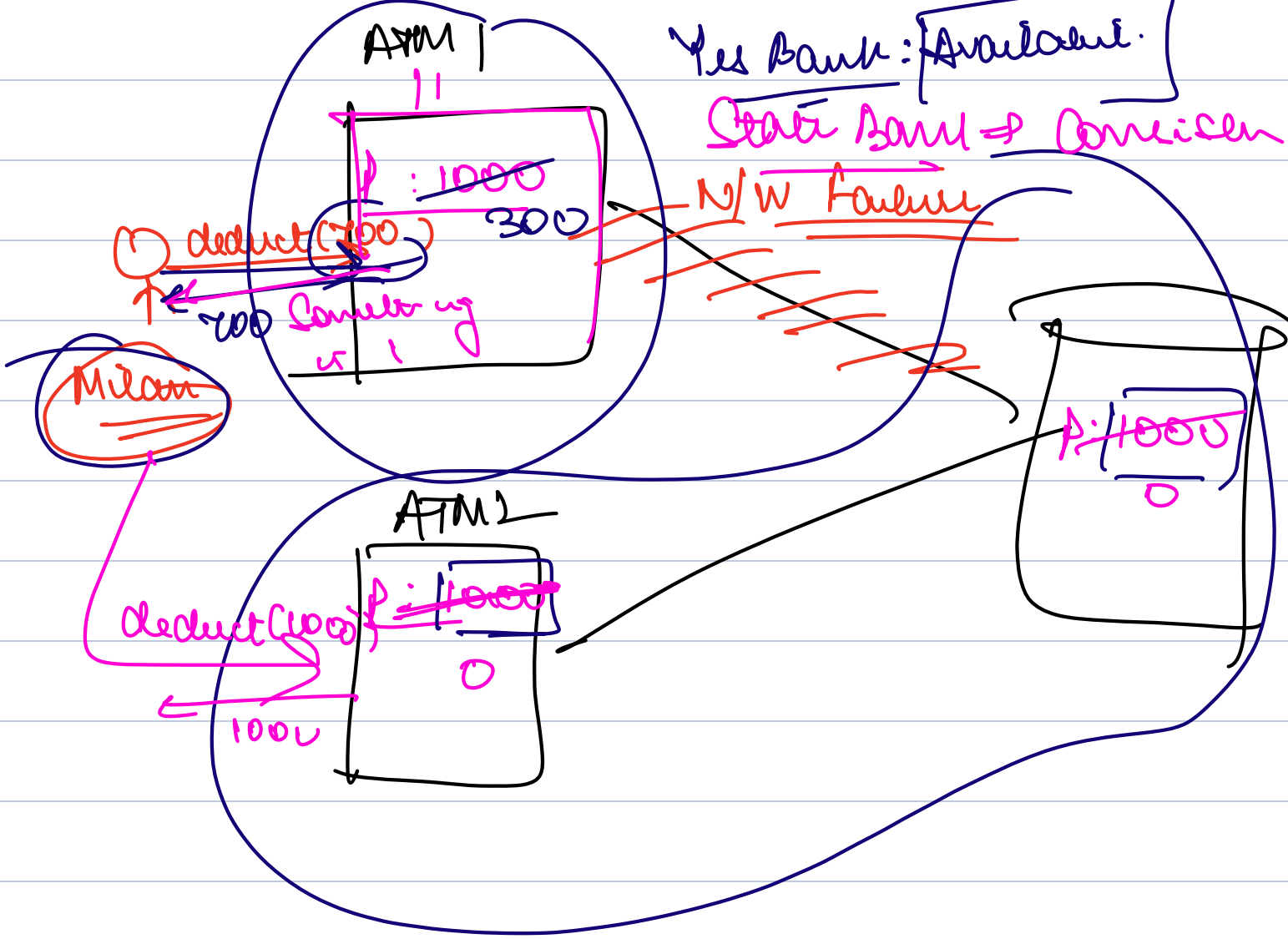
A → Availability : some data.

P → what should system do when it has developed parts that are not able to connect to each other.

Internet

Can be unreliable

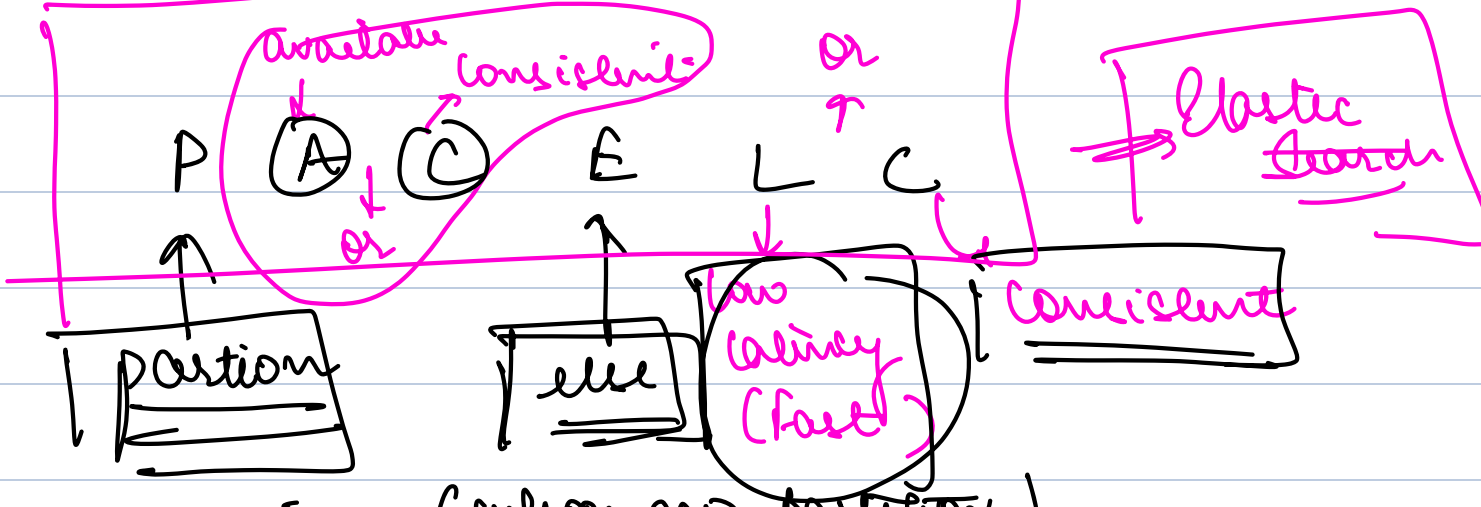
2 Parts where data is stored and both aren't able to reach each other



CAP

PACELC

opp of CAP

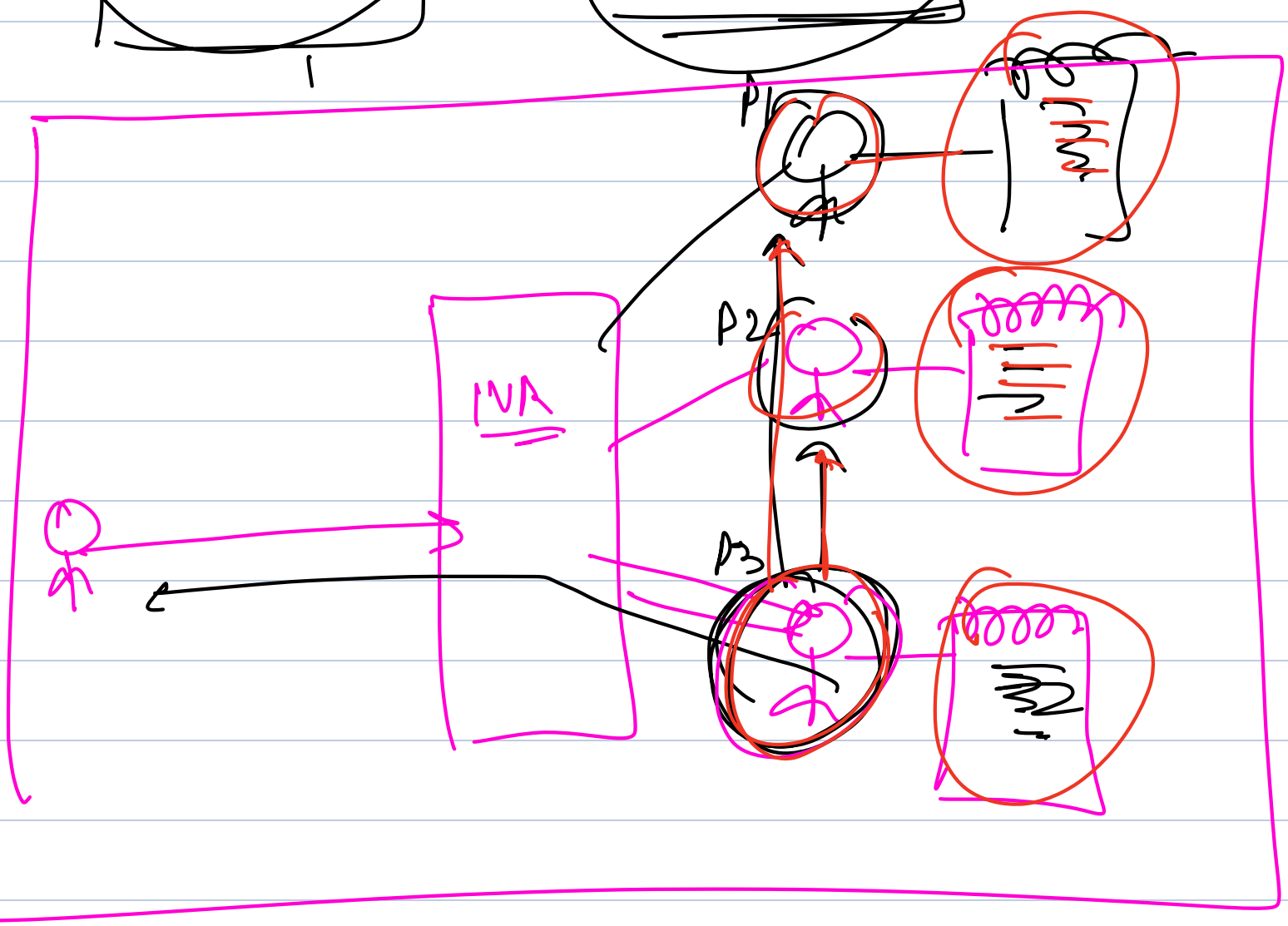


(when no partition)

Gravitation law

$$\frac{G M_1 M_2}{r^2} \rightarrow \frac{K q_1 q_2}{r^2}$$

Coulomb's law



When partition:

available

or

consistent

Else (when no partition):

low latency → fast

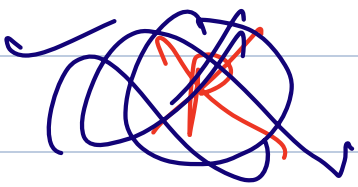
or

Consistent

↳ can't be consistent as well as v. fast at same time

C

A



When your sys is working fine:

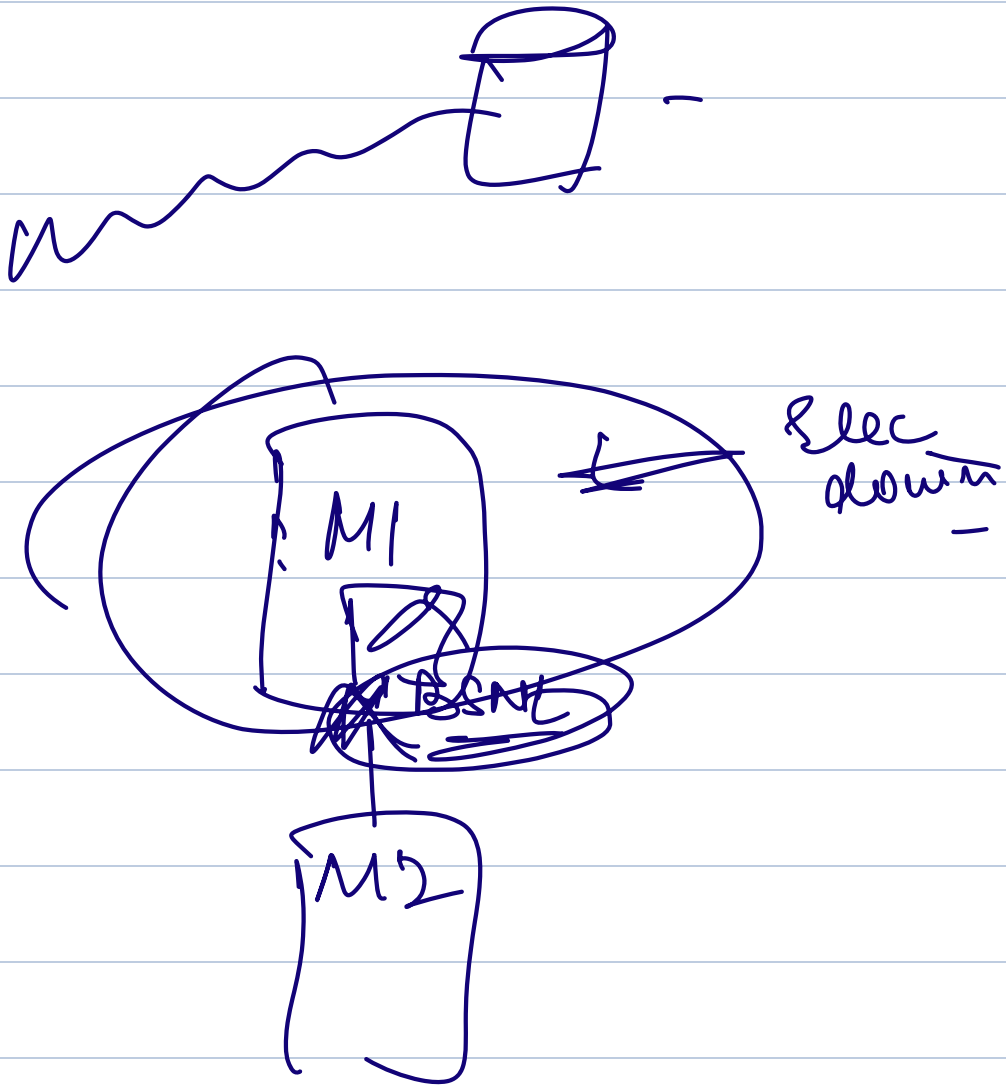
cons and avail

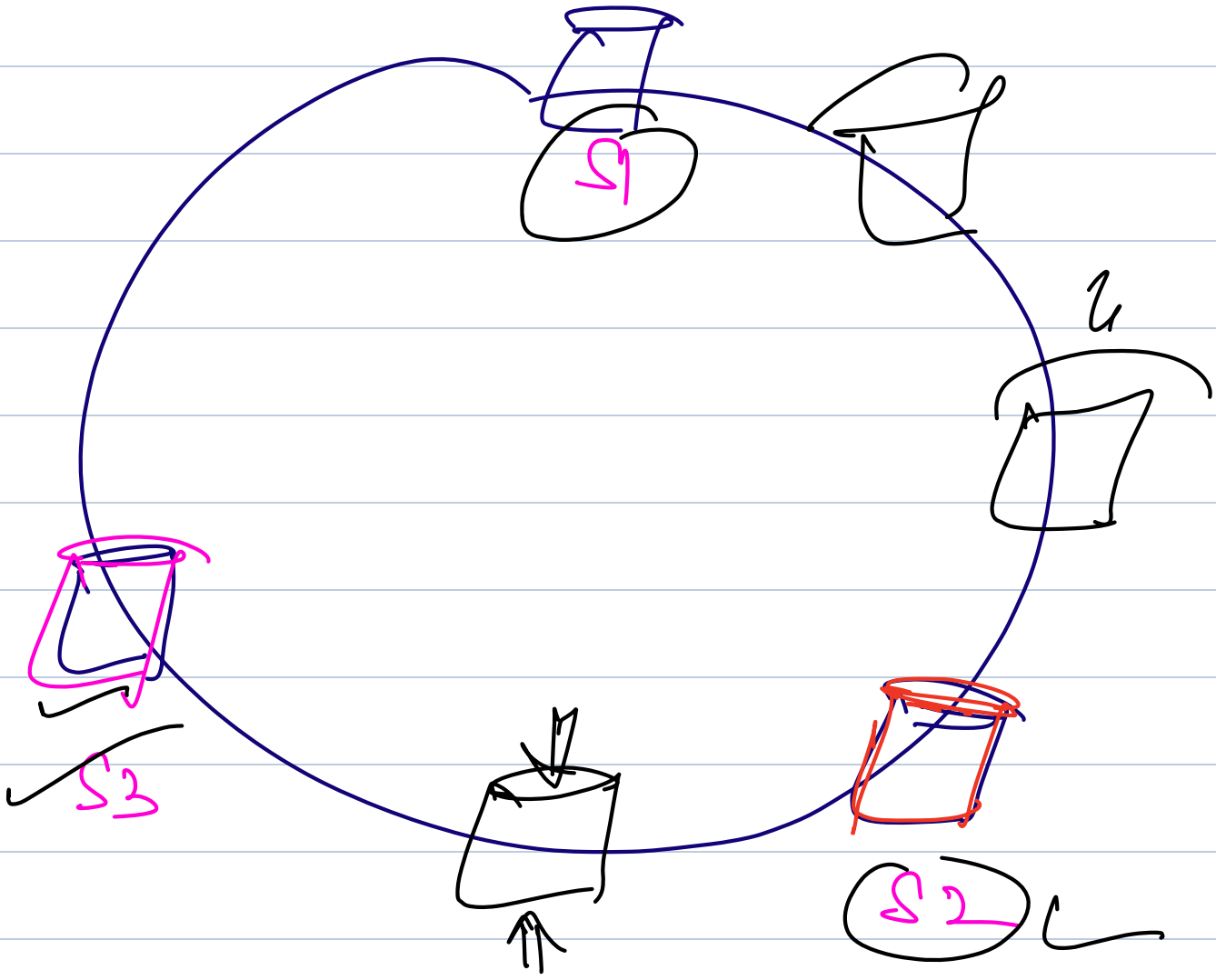
But now q^n is:

fast

or

consistent


M1
BSNL
M2
Elec down


S1
S2
S3