

CAP Theorem

CAP theorem = fundamental principle in distributed systems.

It says: when systems are distributed (multiple nodes, across network), you cannot guarantee

Consistency, Availability, and Partition tolerance all at once.

Important for real-world systems (databases, storage, large-scale apps).

The 3 Properties

C (Consistency): All nodes show the same data. Every read = latest write.

A (Availability): System always responds (success or failure). No downtime.

P (Partition Tolerance): System works even if network between nodes breaks.

The Trade-off

Consistency OR Availability

In presence of a partition (P), we must choose between C and A.

Meaning: during network issues → either system rejects requests to stay consistent OR accepts requests but risks inconsistency.

You can only have 2 out of 3 (not all 3).

CAP Theorem (with Analogy)

Suppose Vaibhav is an Entrepreneur & he notices that people often need reminders,

so Vaibhav builds a **Reminder Service**

{ Anyone can call to his company **ABC** having mob.no' 1234 1234 to tell what they want to remember



now, everything is working smoothly, whenever a person calls, Vaibhav notes his reminder & remind him later

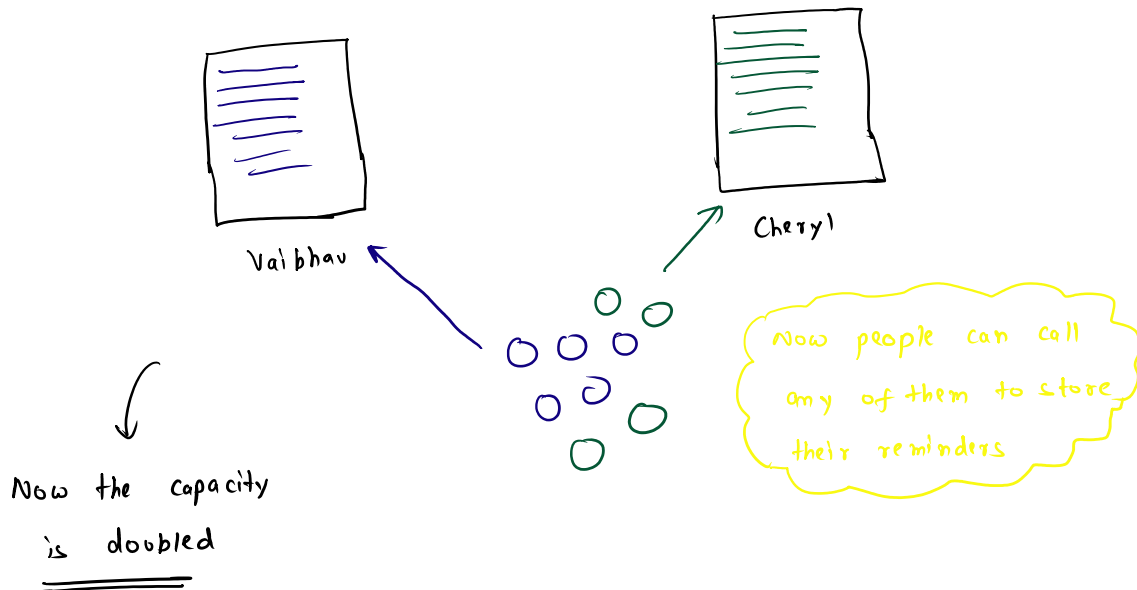
But, Vaibhav notices that with this model he can't scale

Although there is Consistency & Availability both but

(As there is only single node (notebook), so we will not talk about partition at this stage)

So, Vaibhav realized that there needs to be some other person also to manage because he is getting lots of calls & not able to manage alone.

To solve this, Vaibhav asks his girlfriend Cheryl to also get involved



But after few days, a problem arise

Mr. X calls Vaibhav to ask whether he has any reminder of him &

Vaibhav said (NO)

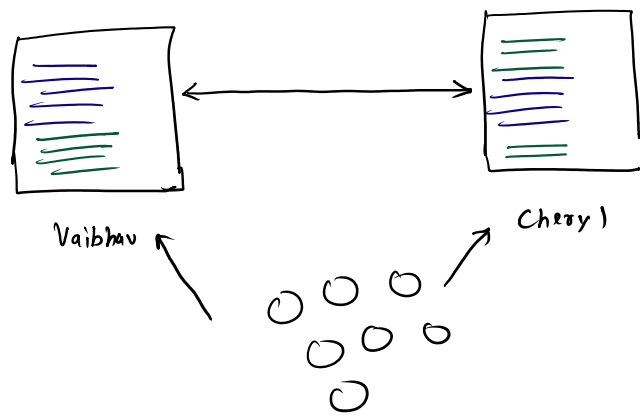
Mr. X becomes very angry because he had called previously to store his reminder

Later, Vaibhav realized that Mr. X had previously called Cheryl (so reminder was in Cheryl's notebook) & then after he calls Vaibhav (he said NO) because the reminder was not in Vaibhav's notebook

Consistency Issue:

- User stores reminder with Cheryl, later calls Vaibhav.
- If Vaibhav doesn't know about it → inconsistent answer.

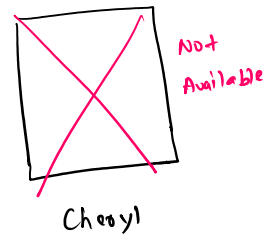
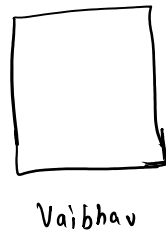
Now, for solving this issue, Vaibhav devises a solⁿ that whenever a call came to anyone (Vaibhav or Cheryl) then they contact each other & write reminder in both notebooks



now the consistency issue is solved because now both notebooks have every reminder

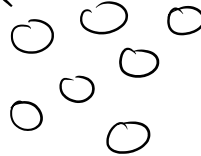
Now, things were going well but one day Cheryl calls Vaibhav that she will be unavailable because she was sick

But now, Vaibhav can't exchange their notes with Cheryl because she is not available



So for staying consistent, Vaibhav also has to reject the calls

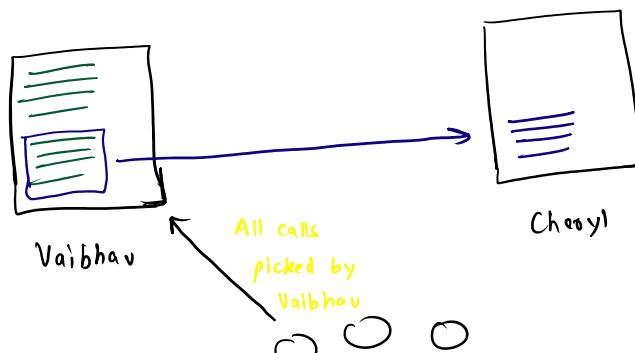
All calls picked by Vaibhav



Availability Issue:

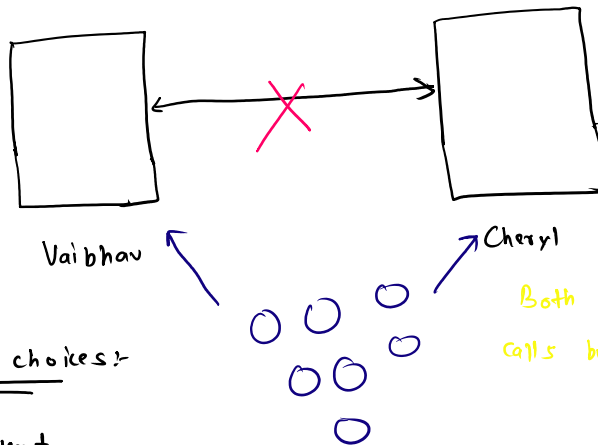
- Cheryl is sick (unavailable).
- If we enforce strict consistency, system rejects new reminders (not available).

Now, for solving this problem Vaibhav has a soln that when Cheryl is sick, only Vaibhav picks the call & write reminders & the next day when Cheryl returns, she first has to copy all the previous day reminders before answering any new call





Now, one day Vaibhav & Cheryl had a fight & now they are not talking to each other
 So the notes (reminders) can't be synced (Partition Tolerance issue)



Now, there are 2 choices:-

↳ Stay Consistent

↳ Stay Available

Partition Tolerance Issue:

- Vaibhav & Cheryl have a fight (no communication = network partition).
- System must choose:
 - Stay consistent → reject new reminders (low availability).
 - Stay available → accept reminders separately but risk inconsistency.

Now, the system faces a choice:

1. Remain Consistent (C):

- Vaibhav says: "I won't take any new reminder unless I confirm with Cheryl."
- Since he can't talk to her → both stop accepting new reminders.
- Result: Consistent but not available.

2. Remain Available (A):

- Vaibhav and Cheryl each keep accepting reminders separately.
- But because they're not syncing, answers may differ.
- Result: Available but inconsistent.

So, in case of partition tolerance I have to choose b/w Consistency & Availability

CP

AP

Eventual Consistency Idea:

Later, a "clerk" reconciles both diaries (Vaibhav's & Cheryl's) → they become consistent eventually.

PACELC Extension

CAP handles trade-off during partitions.

PACELC adds: Even when no partition exists (Else) → trade-off is Consistency vs Latency.

Example:

- Strong consistency = more network confirmations → higher latency.
- Relaxed consistency = low latency but temporary inconsistency.

Remember the problem when we devise a solⁿ to sync notebooks for each call

So the customer has to wait at each call till the entry is written in both

- Strong consistency = more network confirmations → higher latency.
- Relaxed consistency = low latency but temporary inconsistency.

So the customer has to wait at each call till the entry is written in both notebooks
 ↳ Latency

1. CP Systems (Consistency + Partition tolerance)

- Prioritize correct data over always being available.
- During partitions → may reject requests (less available).

Examples:

- Relational databases with strong consistency
 - PostgreSQL, MySQL (with synchronous replication mode)
- HBase
 - Ensures consistency, but can become unavailable if partition happens.
- MongoDB (with "majority write concern")
 - Waits for multiple replicas to acknowledge → may sacrifice availability.
- Zookeeper / etcd
 - Used for leader election, configuration storage — consistency is critical, so they reject requests if partitioned.

👉 Think: banking, stock trading, inventory management — correctness > availability.

2. AP Systems (Availability + Partition tolerance)

- Prioritize system always responding over strict consistency.
- During partitions → system accepts requests, but data may be temporarily inconsistent (eventual consistency).

Examples:

- Cassandra
 - Always available; uses eventual consistency to sync later.
- DynamoDB (Amazon's Dynamo model)
 - Highly available, but may show stale data.
- CouchDB / Riak
 - Focus on availability with replication & conflict resolution.
- DNS (Domain Name System)
 - Always available globally, but records may take time to update → eventual consistency.

👉 Think: social media feeds, e-commerce recommendations, messaging apps — availability > strict consistency.