

# Understanding the CAP Theorem in Distributed Systems

Hitkar Miglani



# Table of Contents

3	What is CAP Theorem?	8	CAP Trade-Offs
4	What is Consistency?	9	Real World Examples
5	What is Availability?	10	Use Cases
6	What is Partition Tolerance?	11	Recap & Summary
7	Importance Of All Factors	13	Questions

# What is The Cap Theorem ?

The CAP theorem is a fundamental concept in distributed systems theory that was first proposed by Eric Brewer in 2000 and subsequently shown by Seth Gilbert and Nancy Lynch in 2002. It asserts that all three of the following qualities cannot be concurrently guaranteed in any distributed data system:

- Consistency
- Availability
- Partition Tolerance

# What is Consistency?

Consistency means that every read receives the most recent write or an error.

- After a successful write operation, any subsequent read operation will return that written data.
- All nodes (servers) reflect the same, up-to-date view of the data at any given time.

Suppose you update your profile picture:

- With Consistency: Every user around the world immediately sees your new picture (or gets an error if the system can't guarantee it).
- No stale or old data is allowed to be returned.

# What is Availability?

Availability means that every request (to a non-failing node) receives a response, regardless of whether it is the latest data.

- The system remains operational and continues to serve read/write requests even during network partitions.
- It prioritizes responding quickly, even if the data may be stale or not fully consistent.
- The focus is on uptime and responsiveness, not on correctness of the data.

Imagine a distributed e-commerce application:

- You add a product to your cart, and even if the latest data hasn't propagated, the system still responds with a valid confirmation.
- It avoids timeouts or errors — you always get a response.

# What is Partition Tolerance ?

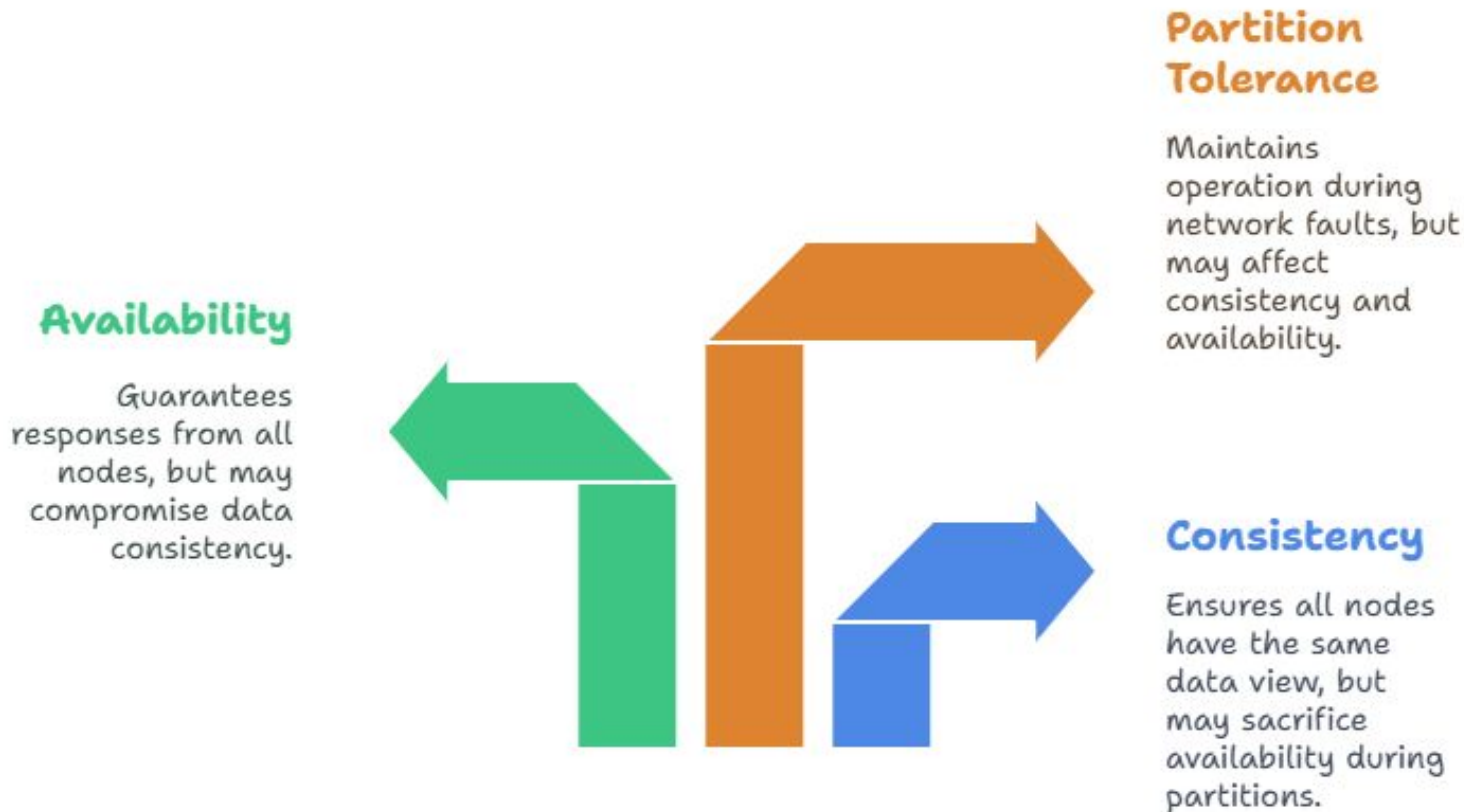
Partition Tolerance means that the system continues to operate correctly even if network communication is lost between some parts of the system.

- The system can tolerate communication failures (partitions) in the network.
- Nodes may be unable to talk to each other, but the system does not crash.
- It ensures data availability or consistency choices are made when partitions occur.

Consider a distributed database split across two data centers:

- If a network issue prevents them from syncing, the system still runs independently in both zones.
- Depending on your design, it might:
  - Allow reads/writes with eventual consistency (AP system)
  - Reject requests to maintain strong consistency (CP system)

# Importance of All Factors:



# CAP Trade-Offs :

System Type	Guarantees	Sacrifices	Use Cases
CA	Consistency + Availability	Partition Tolerance	Single-server databases (e.g., MySQL)
CP	Consistency + Partition Tolerance	Availability	Financial systems (e.g., MongoDB)
AP	Availability + Partition Tolerance	Consistency	Social media (e.g., Cassandra)



## Real World Examples:



MongoDB (CP)




Cassandra (AP)



RDBMS (CA)

# Use Cases →



### CP System

Ensures consistency and partition tolerance, suitable for banking transactions where data accuracy is crucial.



### AP System

Prioritizes availability and partition tolerance, ideal for social media newsfeeds where immediate access is key.



### Hybrid System

Balances availability and consistency, suitable for online shopping carts where different stages require different priorities.



# Recap

- What is CAP Theorem?
- What is Consistency?
- What is Availability?
- What is Partition Tolerance?
- Importance of All Factors
- CAP Trade-Offs
- Real World Examples
- Use Cases

# Summary

- CAP Theorem shapes the architecture of all distributed databases and systems.
- No system can achieve all three: Must choose the best trade-off for your needs.
- Consistency is critical for accuracy, but may reduce uptime.
- Availability ensures users can always access the system, but data may be temporarily inconsistent.
- Partition tolerance is a must in real-world distributed systems due to possible network failures.
- Understand your application's needs before choosing a database or architecture.
- Key takeaway: Always balance Consistency, Availability, and Partition Tolerance based on your project's priorities.



**Questions?**



**Thank You**