**CAP Theorem Explained**

The **CAP theorem**, introduced by Eric Brewer in 2000, provides a fundamental framework for understanding the **trade-offs** that must be made when designing distributed systems.

[[A diagram of different components of a company

AI-generated content may be incorrect.](https://substackcdn.com/image/fetch/$s_!vOjG!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F40850582-9eaf-48e2-a254-43817c4d60c0_951x792.png)](https://substackcdn.com/image/fetch/$s_!vOjG!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F40850582-9eaf-48e2-a254-43817c4d60c0_951x792.png" \t "_blank)

CAP stands for **Consistency**, **Availability**, and **Partition Tolerance**(সহনশীলতা), and the theorem states that:

**It is impossible for a distributed data store to simultaneously provide all three guarantees.**

* **Consistency (C)**: Every read receives the most recent write or an error.
* **Availability (A)**: Every request (read or write) receives a non-error response, without guarantee that it contains the most recent write.
* **Partition Tolerance (P)**: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

In this article, we will explore the the 3 pillars of CAP theorem, trade-offs, and practical design strategies for building resilient and scalable distributed systems.

**3 Pillars of CAP**

**1. Consistency**

Consistency ensures that **every read receives the most recent write** **or an error**. This means that all working nodes in a distributed system will return the same data at any given time.

[[A diagram of a computer system

AI-generated content may be incorrect.](https://substackcdn.com/image/fetch/$s_!y3bc!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Ff6cda484-672c-4009-9cda-69a8e36e95a3_1166x932.png)](https://substackcdn.com/image/fetch/$s_!y3bc!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Ff6cda484-672c-4009-9cda-69a8e36e95a3_1166x932.png" \t "_blank)

In a consistent distributed system, if you write data to node A, a read operation from node B will immediately reflect the write operation on node A.

Consistency is crucial for applications where having the most up-to-date data is critical, such as financial systems, where a balance inquiry must reflect the most up-to-date state of an account.

**2. Availability**

Availability guarantees that **every request (read or write) receives a response**, without ensuring that it contains the most recent write. This means that the system remains **operational** and **responsive**, even if the response from some of the nodes don’t reflect most up-to-date data.

[[A diagram of a computer network

AI-generated content may be incorrect.](https://substackcdn.com/image/fetch/$s_!3lf-!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F4d8ccf80-2bee-4ff6-8cc4-005a418da7ae_1188x868.png)](https://substackcdn.com/image/fetch/$s_!3lf-!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F4d8ccf80-2bee-4ff6-8cc4-005a418da7ae_1188x868.png" \t "_blank)

Availability is important for applications that need to remain operational at all times, such as online retail systems.

**3. Partition Tolerance**

Partition Tolerance means that the **system continues to function despite network partitions** where nodes cannot communicate with each other.

[[A diagram of a network

AI-generated content may be incorrect.](https://substackcdn.com/image/fetch/$s_!13Kz!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F73a83347-71c5-4d8f-9487-ff41b5bc1333_844x858.png)](https://substackcdn.com/image/fetch/$s_!13Kz!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F73a83347-71c5-4d8f-9487-ff41b5bc1333_844x858.png" \t "_blank)

A **network partition** occurs when a network failure causes a distributed system to split into two or more groups of nodes that cannot communicate with each other.

When there is a network partition, the system must choose between **Consistency** and **Availability**.

Partition Tolerance is essential for distributed systems because network failures can and do happen. A system that tolerates partitions can maintain operations across different network segments.

**The CAP Trade-Off: Choosing 2 out of 3**

The CAP theorem asserts that in the presence of a network partition, a distributed system must choose between **Consistency** and **Availability**.

Let's explore these scenarios:

**CP (Consistency and Partition Tolerance):**

These systems prioritize consistency and can tolerate network partitions, but at the cost of availability. During a partition, the system may reject some requests to maintain consistency.

Traditional relational databases, such as MySQL and PostgreSQL, when configured for strong consistency, prioritize consistency over availability during network partitions.

**Banking systems** typically prioritize **consistency over availability** since data accuracy is more critical than availability during network issues.

Consider an ATM network for a bank. When you withdraw money, the system must ensure that your balance is updated accurately across all nodes (consistency) to prevent overdrafts or other errors.

**AP (Availability and Partition Tolerance):**

These systems ensure availability and can tolerate network partitions, but at the cost of consistency. During a partition, different nodes may return different values for the same data.

NoSQL databases like Cassandra and DynamoDB are designed to be highly available and partition-tolerant, potentially at the cost of strong consistency.

**Amazon's** shopping cart system is designed to always accept items, prioritizing availability.

When you add items to your Amazon cart, the action almost never fails, even during high traffic periods like Black Friday.

**CA (Consistency and Availability):**

In the absence of partitions, a system can be both consistent and available. However, network partitions are inevitable in distributed systems, making this combination impractical.

**Example Systems:** Single-node databases can provide both consistency and availability but aren't partition-tolerant. In a distributed setting, this combination is theoretically impossible.

**Practical Design Strategies**

Designing distributed systems requires carefully balancing these trade-offs based on application requirements.

Here are some practical strategies:

**1. Eventual Consistency**

For many systems, strict consistency isn't always necessary.

Eventual consistency can provide a good balance where updates are propagated to all nodes eventually, but not immediately.

**Example:** Systems where immediate consistency is not critical, such as DNS and content delivery networks (CDNs).

**2. Strong Consistency**

A model ensuring that once a write is confirmed, any subsequent reads will return that value.

**Example:** Systems requiring high data accuracy, like financial applications and inventory management.

**3. Tunable Consistency**

Tunable consistency allows systems to adjust their consistency levels based on specific needs, providing a balance between strong and eventual consistency.

Systems like Cassandra allow configuring the level of consistency on a per-query basis, providing flexibility.

**Example:** Applications needing different consistency levels for different operations, such as e-commerce platforms where order processing requires strong consistency but product recommendations can tolerate eventual consistency.

**4. Quorum-Based Approaches:**

Quorum-based approaches use voting among a group of nodes to ensure a certain level of consistency and fault tolerance.

**Example:** Systems needing a balance between consistency and availability, often used in consensus algorithms like Paxos and Raft.

**Beyond CAP: PACELC**

While CAP is foundational, it doesn't cover all scenarios.

Daniel Abadi proposed the [**PACELC**](https://en.wikipedia.org/wiki/PACELC_theorem) theorem as an extension by introducing **latency** and **consistency** as additional attributes of distributed systems.

* If there is a partition (P), the trade-off is between availability and consistency (A and C).
* Else (E), the trade-off is between **latency (L)** and **consistency (C)**.

This theorem acknowledges that even when the system is running normally, there's a tradeoff between latency and consistency.

In conclusion, the CAP Theorem is a powerful tool for understanding the inherent trade-offs in distributed system design. It's not about choosing the "best" property, but rather about making informed decisions based on the specific needs of your application.

By carefully evaluating the CAP trade-offs, you can architect robust and resilient systems that deliver the right balance of consistency, availability, and partition tolerance.

Thank you for reading!

**Basic Questions**

1. **What does CAP stand for in the CAP theorem?**  
   **Answer:** Consistency, Availability, and Partition Tolerance.
2. **Who introduced the CAP theorem, and when?**  
   **Answer:** Eric Brewer proposed it in 2000, and it was later proven by Seth Gilbert and Nancy Lynch in 2002.
3. **What does the CAP theorem state?**  
   **Answer:** In a distributed system, you can only guarantee **two** of the three properties (Consistency, Availability, Partition Tolerance) at the same time — you can’t have all three fully.
4. **What is Consistency in CAP?**  
   **Answer:** Every read receives the most recent write (strong consistency) or an error.
5. **What is Availability in CAP?**  
   **Answer:** Every request receives a non-error response, even if it’s not the latest data.
6. **What is Partition Tolerance in CAP?**  
   **Answer:** The system continues to function despite network partitions or message loss between nodes.

**Intermediate Questions**

1. **Why can’t a distributed system achieve all three properties of CAP?**  
   **Answer:** In the presence of a network partition, you must choose between serving stale data (Availability) or refusing requests until consistency is restored (Consistency).
2. **Can a single-node database violate CAP?**  
   **Answer:** No — CAP applies to distributed systems with multiple nodes where network partitions can occur.
3. **Give an example of a CP system.**  
   **Answer:** HBase, MongoDB (with strong consistency mode), Zookeeper — they prioritize Consistency and Partition Tolerance.
4. **Give an example of an AP system.**  
   **Answer:** Cassandra, CouchDB, DynamoDB — they prioritize Availability and Partition Tolerance, allowing eventual consistency.
5. **Give an example of a CA system.**  
   **Answer:** A traditional single-node relational database (e.g., MySQL in standalone mode) — prioritizes Consistency and Availability but loses Partition Tolerance.

**Advanced Questions**

1. **How does eventual consistency relate to CAP?**  
   **Answer:** It’s a compromise used in AP systems — data may be temporarily inconsistent, but replicas will converge over time.
2. **If a system chooses Availability over Consistency during a partition, what’s the trade-off?**  
   **Answer:** Users may get stale or conflicting data, requiring conflict resolution later.
3. **If a system chooses Consistency over Availability during a partition, what’s the trade-off?**  
   **Answer:** Some requests will fail or be blocked until the partition is resolved, but data remains correct and synchronized.
4. **How does the PACELC theorem extend CAP?**  
   **Answer:** It states that in case of a Partition (P), you choose Availability (A) or Consistency (C); Else (E), when the system is running normally, you choose between Latency (L) and Consistency (C).

**Scenario-Based CAP Theorem Questions & Answers**

1. **Online Banking Transfer During Network Partition**  
   The system must either block transactions until the network recovers or risk inconsistent balances.
   * **Question:** Should it choose CP or AP?
   * **Answer:** **CP** — prioritize Consistency to prevent incorrect balances.
2. **Social Media Feed During Outage**  
   Users should still see posts, even if some are slightly old.
   * **Question:** Should it choose CP or AP?
   * **Answer:** **AP** — prioritize Availability, allow eventual consistency.
3. **E-Commerce Checkout**  
   Stock levels must be accurate to prevent overselling during a partition.
   * **Question:** CP or AP?
   * **Answer:** **CP** — Consistency is critical for inventory correctness.
4. **Video Streaming Platform**  
   During a node failure, continue streaming without blocking.
   * **Question:** CP or AP?
   * **Answer:** **AP** — Availability is more important than showing the absolute latest data.
5. **Distributed Caching System**  
   Serving stale cache data is acceptable for speed.
   * **Question:** CP or AP?
   * **Answer:** **AP** — prioritize speed and availability over strict consistency.
6. **Airline Ticket Reservation System**  
   Must prevent double booking of the same seat.
   * **Question:** CP or AP?
   * **Answer:** **CP** — prevent inconsistent seat allocations.
7. **Real-Time Stock Trading**  
   Prices and holdings must be 100% accurate.
   * **Question:** CP or AP?
   * **Answer:** **CP** — incorrect trades could cause huge losses.
8. **Ride-Hailing App Location Updates**  
   Showing a slightly outdated driver location is fine.
   * **Question:** CP or AP?
   * **Answer:** **AP** — prefer availability to keep tracking working.
9. **Weather Forecast App**  
   If a network partition occurs, still serve yesterday’s data.
   * **Question:** CP or AP?
   * **Answer:** **AP** — availability matters more than strict freshness.
10. **Online Multiplayer Game Leaderboard**  
    Leaderboard rankings should eventually sync but not block gameplay.
    * **Question:** CP or AP?
    * **Answer:** **AP** — players can keep playing with slightly stale leaderboard.
11. **Medical Record Update System**  
    Doctors must see the most recent test results before prescribing.
    * **Question:** CP or AP?
    * **Answer:** **CP** — medical safety requires consistency.
12. **IoT Smart Home Lighting**  
    Light switches should work even if the cloud server is down.
    * **Question:** CP or AP?
    * **Answer:** **AP** — prioritize availability for usability.
13. **Email Delivery Service**  
    Emails can be queued and sent later, but service must accept requests.
    * **Question:** CP or AP?
    * **Answer:** **AP** — allow temporary inconsistency.
14. **Blockchain Cryptocurrency Transfer**  
    Double-spending must be impossible even during partitions.
    * **Question:** CP or AP?
    * **Answer:** **CP** — strong consistency to prevent fraud.
15. **Content Delivery Network (CDN)**  
    Serve static assets globally, even if some edge servers have old copies.
    * **Question:** CP or AP?
    * **Answer:** **AP** — prioritize availability and speed.

**CAP Scenario Decision Table**

| **Scenario** | **Choice** | **Why** |
| --- | --- | --- |
| Online Banking Transfer | **CP** | Prevent inconsistent balances; block until partition resolves. |
| Social Media Feed | **AP** | Users can see slightly stale posts; keep app usable. |
| E-Commerce Checkout | **CP** | Avoid overselling by ensuring accurate stock. |
| Video Streaming Platform | **AP** | Keep streaming even if metadata is slightly outdated. |
| Distributed Cache | **AP** | Serve stale data for performance. |
| Airline Ticket Reservation | **CP** | Prevent double bookings of the same seat. |
| Stock Trading | **CP** | Ensure trades are executed with correct prices and holdings. |
| Ride-Hailing App Location Updates | **AP** | Small location delays are acceptable to keep service running. |
| Weather Forecast App | **AP** | Yesterday’s data is acceptable during outage. |
| Game Leaderboard | **AP** | Gameplay continues with slightly outdated rankings. |
| Medical Records | **CP** | Safety requires doctors see most recent data. |
| Smart Home Lighting | **AP** | Lights must work even without cloud connection. |
| Email Delivery Service | **AP** | Queue emails; accept requests immediately. |
| Blockchain Transfers | **CP** | Prevent double-spending; strong consistency needed. |
| CDN Static Assets | **AP** | Serve cached files globally for speed and uptime. |