System Design Day 1: CAP Theorem

1. What is the **CAP Theorem**?

- **Consistency (C)**: Every read receives the most recent write.
- **Availability (A)**: Every request receives a non-error response (even if stale).
- **Partition Tolerance (P)**: The system continues operating despite network failures between nodes.

  In any distributed system where network partitions are possible (which is most real-world systems), you **must choose** between C and A. You cannot guarantee all three.

2. Choosing Between C and A

🔒 CP: Consistency + Partition Tolerance

- Maintains correctness by rejecting or delaying reads/writes during a partition.
- **Example**: Banking systems—better to delay a transaction than risk inconsistent balances.
  - Finance like share market
  - Ticketing Apps

☁️ AP: Availability + Partition Tolerance

- Always responds, even if some data is outdated (eventual consistency).
- **Example**: Social media feeds—users see posts even if some replicas lag.

  Practical systems: Cassandra and DynamoDB follow AP, while MongoDB (default) leans CP.

  Video Streaming Apps like Netflix

3. Real-World Tuning & Examples

- **DynamoDB**: Lets you choose durability vs speed — you can request **strongly consistent** or **eventual reads**. Splunk
- **SQL databases** (e.g., MySQL, PostgreSQL): often default to strong consistency (CP), especially with master-slave setups—writes may pause during failover. Wikipedia+3Exponent+3DEV Community+3
- **Cassandra & Dynamo**: Default to AP, but can be tuned per operation for quorum reads/writes
- **Own Example:** In almost all our projects we have used CP as we need data consistency.

4. Best Practices for Consistency in Spring Boot

1. Use `@Transactional` smartly, even for nested methods.
2. Avoid complex distributed transactions — prefer SAGA/eventual consistency.
3. Use strong types and validation to reduce data issues at boundaries.
4. Centralize audit/logging to track cross-service consistency.
5. Monitor using observability tools (e.g., Spring Sleuth, Zipkin).

Layer	How Spring Boot Ensures Consistency
Database	`@Transactional` , JPA flush modes
Microservices	SAGA, orchestration, retries
Caching	`@CacheEvict` , write-through, TTL
Messaging	Eventual consistency, deduplication
Replication	DB-specific configs (e.g., read quorum)