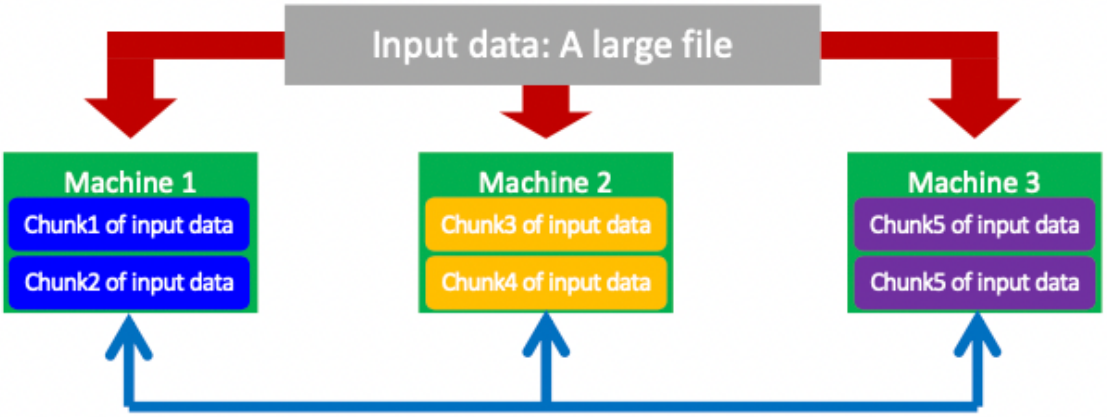
Cap theorem

Scaling Traditional Databases

- Traditional RDBMSs can be either scaled:
    - Vertically (or up).
        - Can be achieved by hardware upgrades (e.g., faster CPU, more memory, or larger disk).
        - Limited by the amount of CPU, RAM, and disk that can be configured on a single machine.
    - Horizontally (or out).
        - Can be achieved by adding more machine.
        - Requires database sharding and probably replication.
        - Limited by the Read-to-Write ratio and communication overhead.

Data sharding

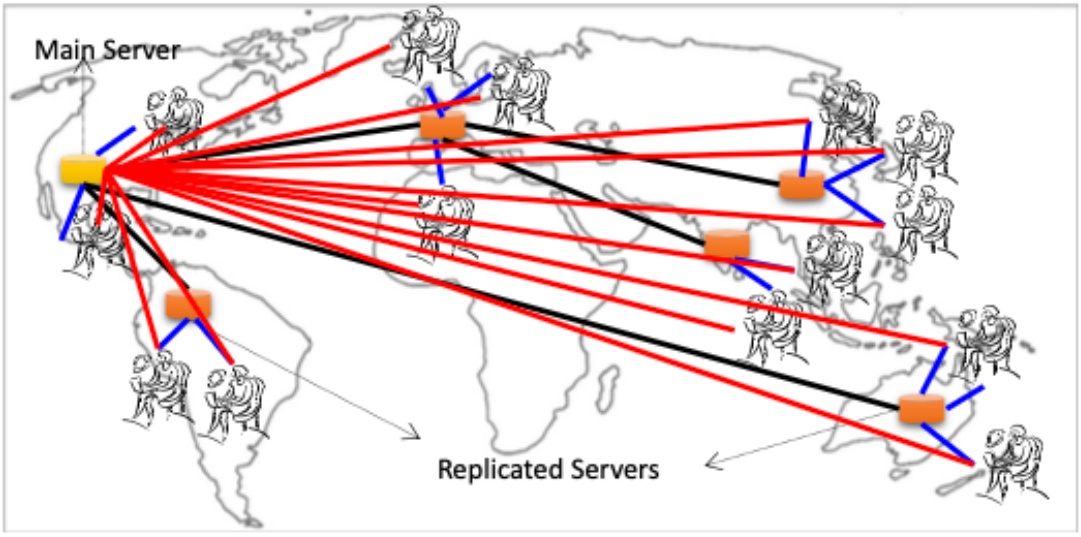
- Data is typically sharded (or striped) to allow for concurrent/parallel accesses.
- Will it scale for complex query processing?



E.g., Chunks 1, 3 and 5 can be accessed in parallel

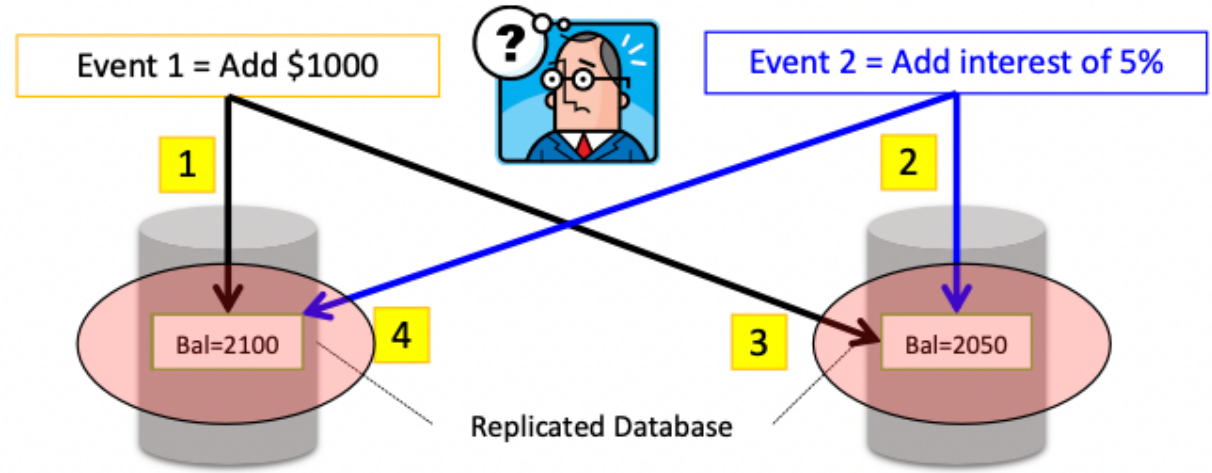
Data replicating

- Replicating data across servers helps in:
    - Avoiding performance bottlenecks.
    - Avoiding single point of failures.
    - And, hence, enhancing scalability and availability.



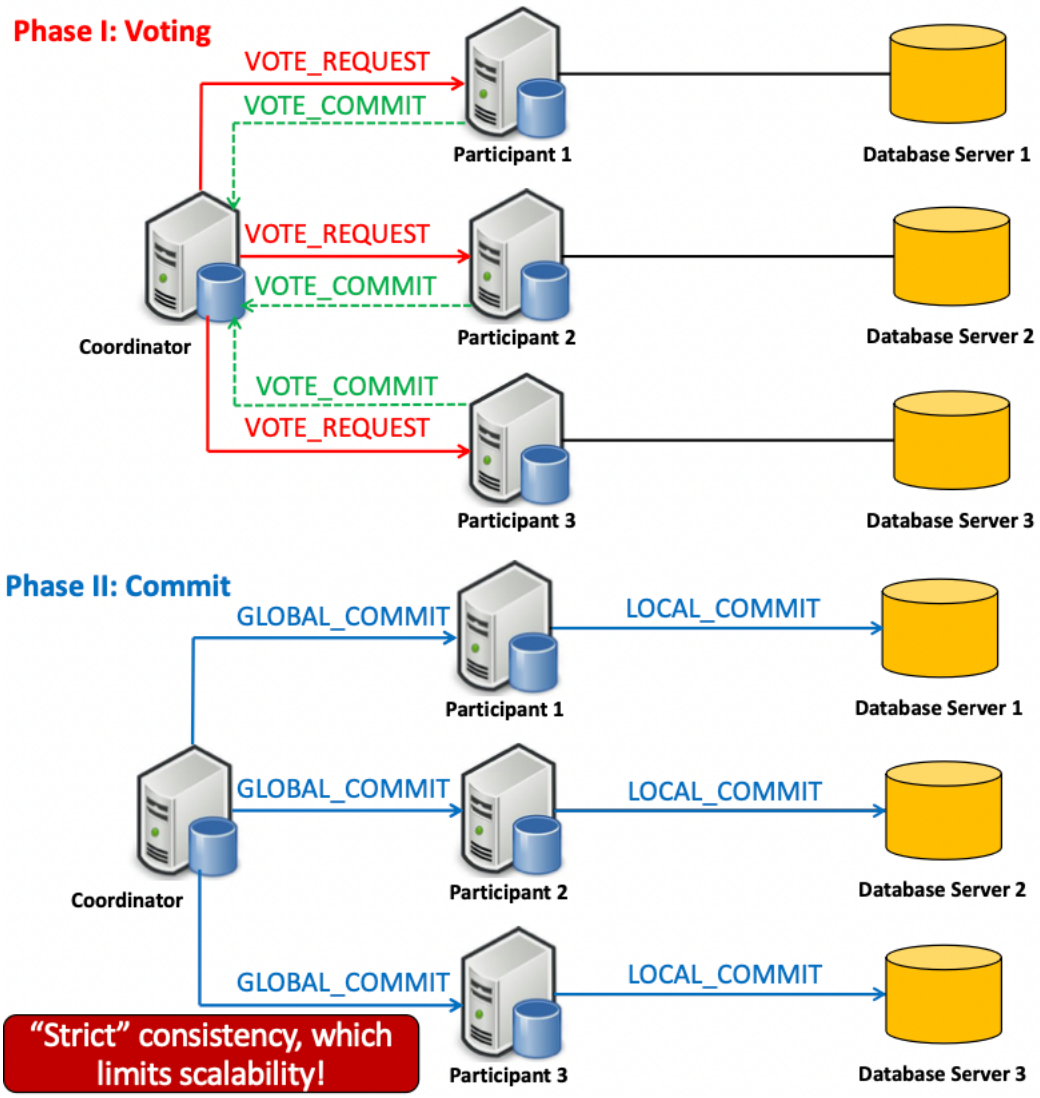
But Consistency Becomes a Challenge

- Example:
  - In an e-commerce application, the bank database has been replicated across 2 servers.
  - Maintaining consistency of replicated data is a challenge.



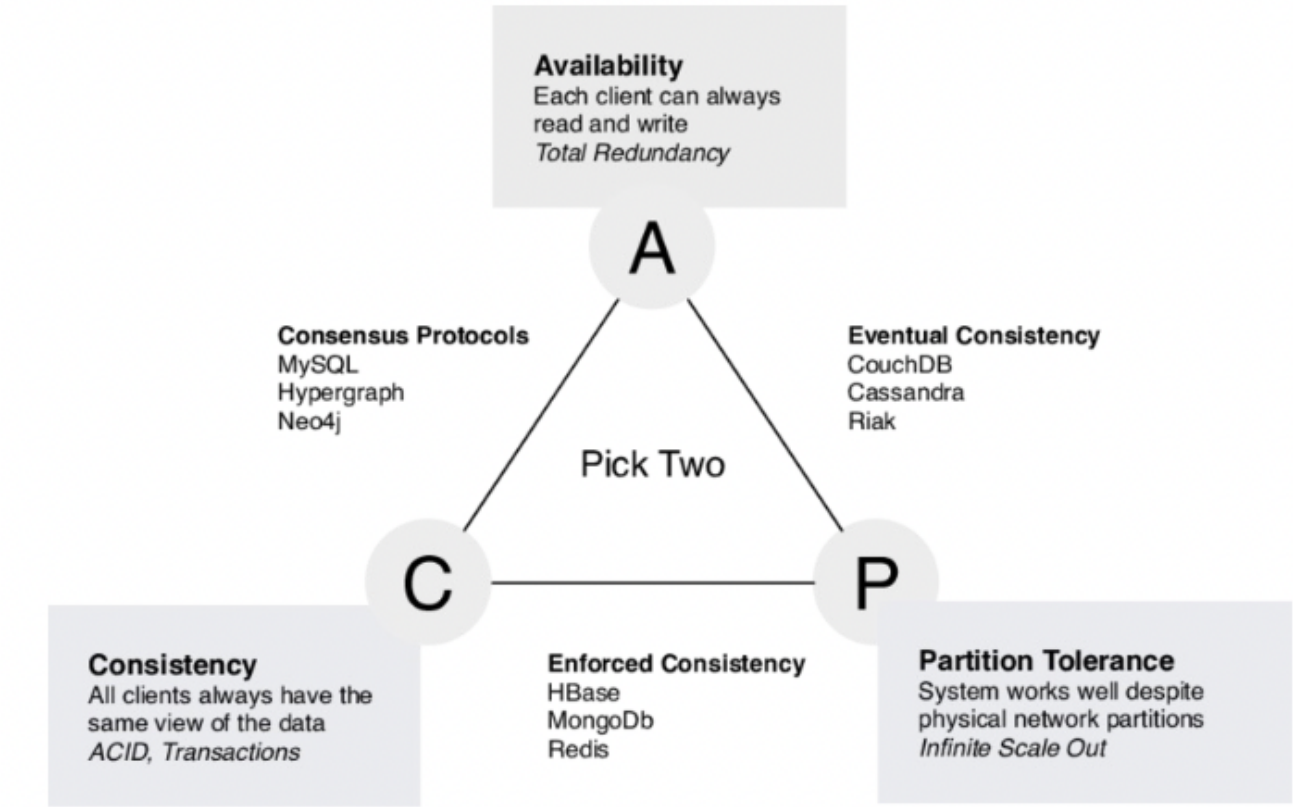
The Two-Phase Commit Protocol

- The two-phase commit protocol (2PC) can be used to ensure atomicity and consistency.



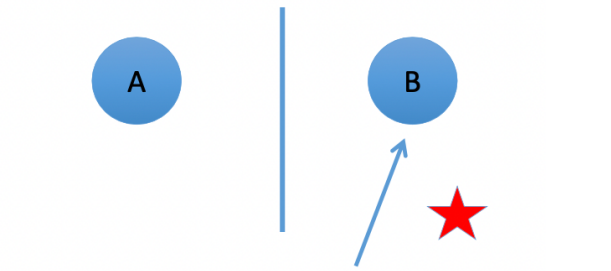
The CAP theorem

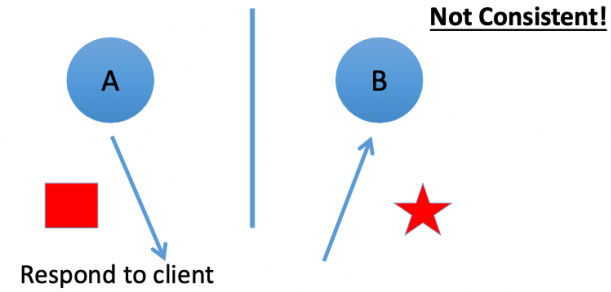
- The limitations of distributed databases can be described in the so called the CAP theorem.
    - Consistency: every node always sees the same data at any given instance (i.e., strict consistency).
    - Availability: the system continues to operate, even if nodes in a cluster crash, or some hardware or software parts are down due to upgrades.
    - Partition Tolerance: the system continues to operate in the presence of network partitions.
- ***CAP theorem: any distributed database with shared data can have at most two of three desirable properties, C, A, or P. These are trade-offs involved in distributed system by Eric Brewer in PODC 2000.***

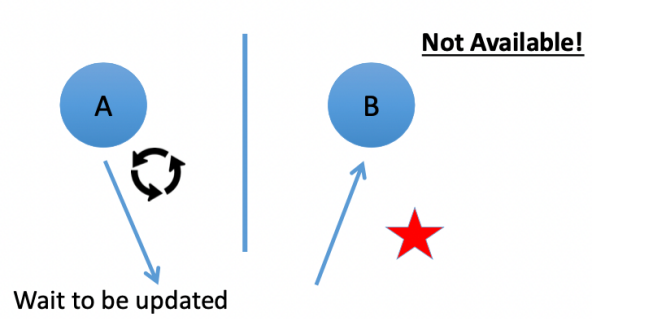


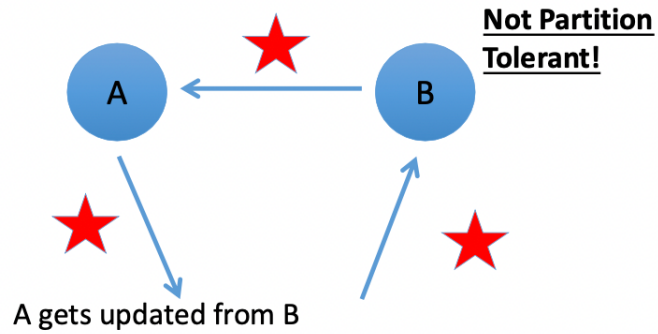
The CAP theorem: Proof

• A simple proof using two nodes:









Scalability of relational databases

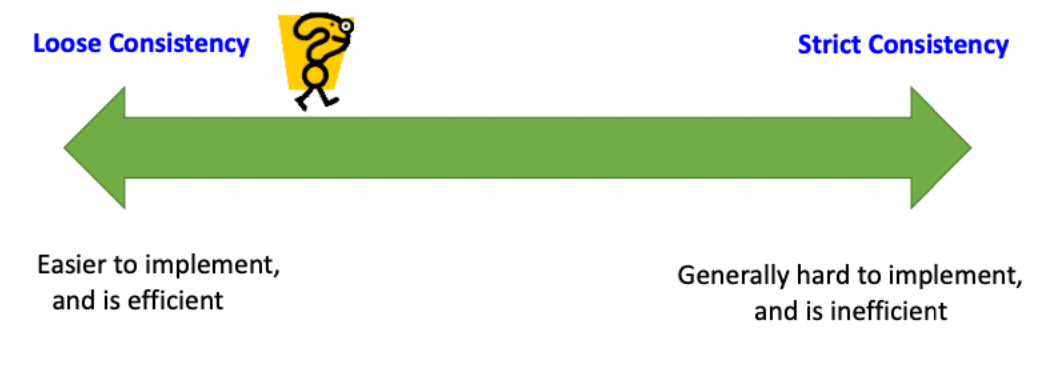
- The Relational Database is built on the principle of ACID (Atomicity, Consistency, Isolation, Durability).
- It implies that a truly distributed relational database should have availability, consistency, and partition tolerance.
- Which unfortunately is impossible.

Large-Scale Databases

- When companies such as Google and Amazon were designing large-scale databases, 24/7 Availability was a key.
    - A few minutes of downtime means lost revenue.
- When horizontally scaling databases to 1000s of machines, the likelihood of a node or a network failure increases tremendously.
- Therefore, to have strong guarantees on Availability and Partition Tolerance, they had to sacrifice “strict” Consistency (implied by the CAP theorem).

Trading-Off consistency

- Maintaining consistency should balance between the strictness of consistency versus availability/scalability.
    - Good-enough consistency depends on your application.

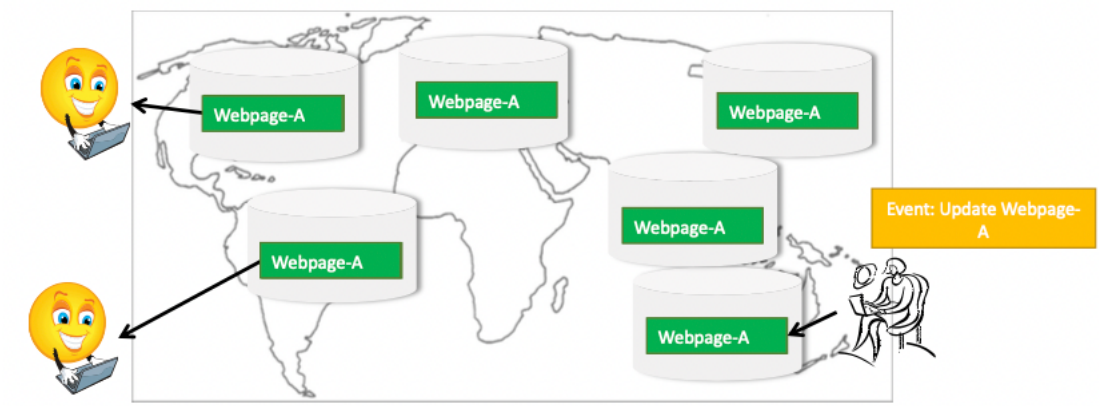


The BASE Properties

- The CAP theorem proves that it is impossible to guarantee strict Consistency and Availability while being able to tolerate network partitions.
- This resulted in databases with relaxed ACID guarantees.
- Such databases apply the BASE properties:
    - Basically Available: the system guarantees Availability.
    - Soft-State: the state of the system may change over time.
    - Eventual Consistency: the system will eventually become consistent.

Eventually Consistency

- A database is termed as Eventually Consistent if:
    - All replicas will gradually become consistent in the absence of new updates.



- But what if the client accesses the data from different replicas?
    - Protocols like Read Your Own Writes (RYOW) can be applied!

Table of Contents