CAP Theorem of Distributed Systems

School of Information Studies
Syracuse University

Distributed Systems

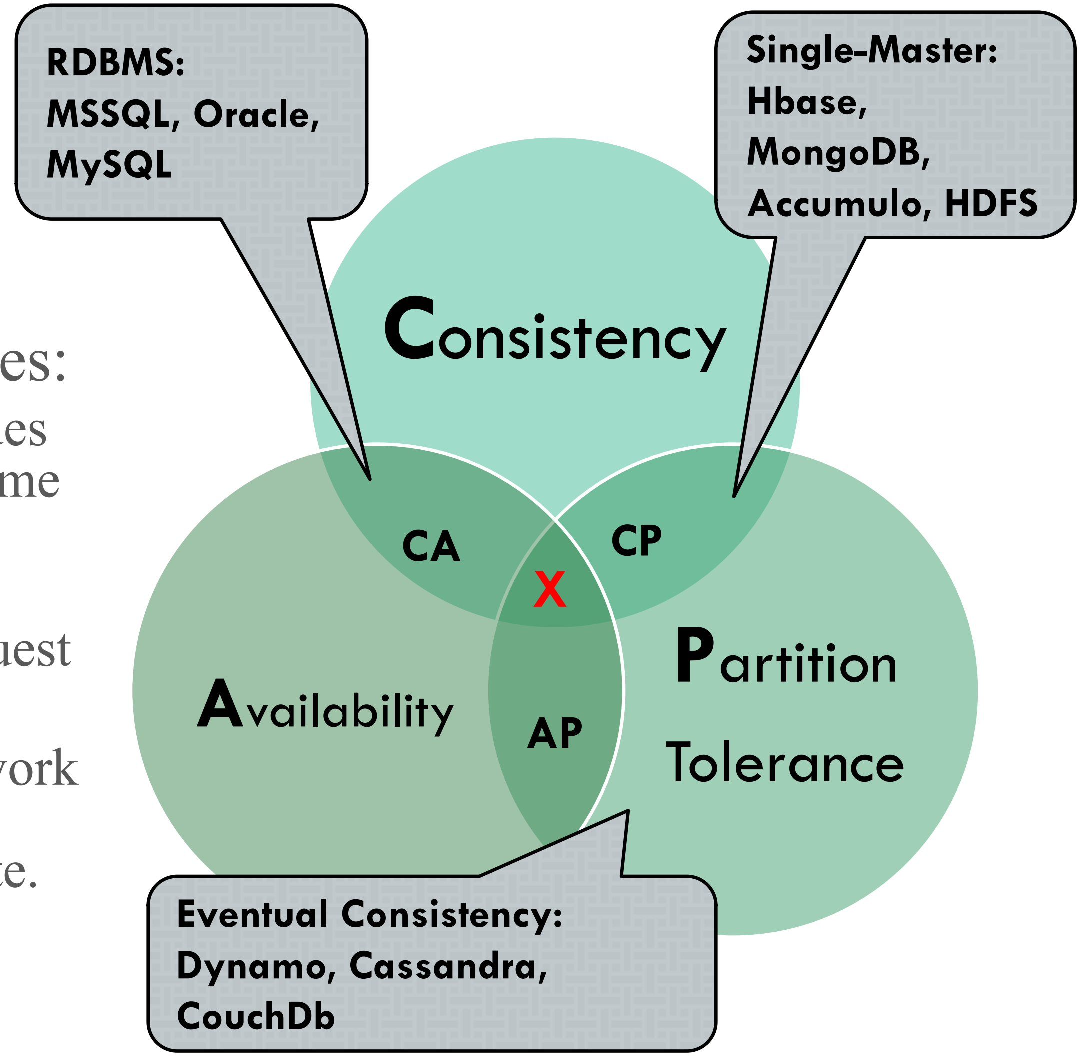
When the data volume is too large for a single system, and you can no longer scale up…

… you scale out.

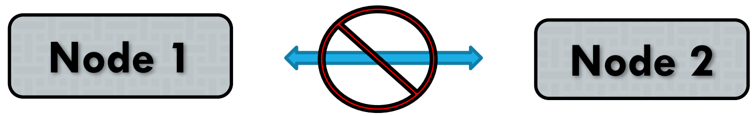
CAP Theorem of Distributed Systems

At the same time, you can have only two of the following three guarantees:

- **Data consistency**: All nodes see the same data at the same time.
- **Data availability**: Assurances that every request can be processed.
- **Partition tolerance**: Network failures are tolerated; the system continues to operate.



Why Can't You Have All Three? *



A counterexample:

Suppose we lose communication between nodes:

- We must ignore any updates the nodes receive, or sacrifice **consistency**, or we must deny service until it becomes ***available*** again.

If we guarantee ***availability*** of requests, despite the failure:

- We gain ***partition tolerance*** (the system still works) but lose ***consistency*** (nodes will get out of sync).

If we guarantee **consistency** of data, despite the failure:

- We gain ***partition tolerance*** (again, system works) but lose ***availability*** (data on nodes cannot be changed failure is resolved).

* You can have all three, just not at the same time.

CAP: Database Systems for Every Need

RDBMSs like Oracle, MySQL and SQL Server:

- Focus on consistency and availability (ACID principles), sacrificing partition tolerance (and thus they don't scale well horizontally).
- Use cases: business data, when you don't need to scale out.

Single-master systems like MongoDb, Hbase, Redis, and HDFS:

- Provide consistency at scale but data availability runs through a single node.
- Use cases: read-heavy DW, caching, document storage, product catalogs.

Eventual consistency systems like CouchDb, Cassandra, and Dynamo

- Provide availability at scale but do not guarantee consistency.
- Use cases: write-heavy, isolated activities, e.g., shopping carts, orders, tweets.