



CAP Theorem



CAP Theorem

- Conjectured by Prof. Eric Brewer at PODC (Principle of Distributed Computing) 2000 keynote talk, proved in 2002 by Nancy Lynch and Seth Gilbert at MIT
- Described the *trade-offs involved in distributed system*
- It is impossible for a web service to provide following *three guarantees at the same time*:
 - **Consistency**
 - **Availability**
 - **Partition-tolerance**

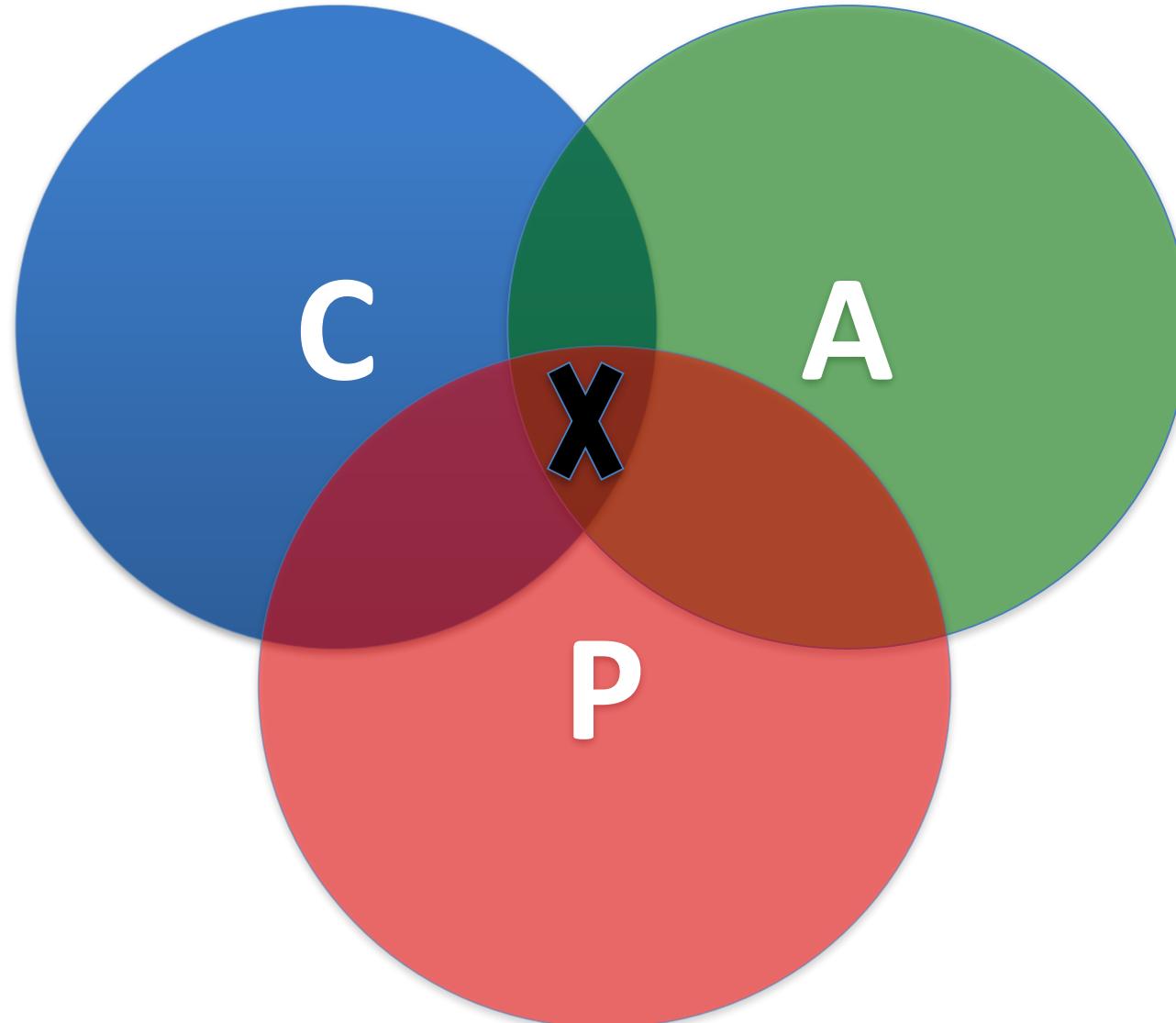


CAP Theorem

- Consistency:
 - All nodes should see the same data at the same time
- Availability:
 - Node failures do not prevent survivors from continuing to operate
- Partition-tolerance:
 - The system continues to operate despite network partitions
- A distributed system can satisfy any two of these guarantees at the same time **but not all three**



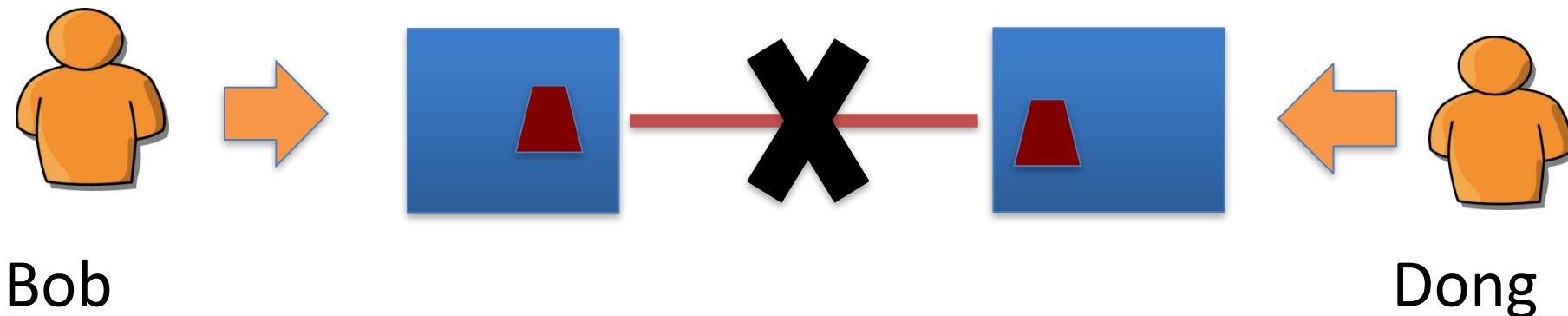
CAP Theorem



CAP Theorem

- A simple example:

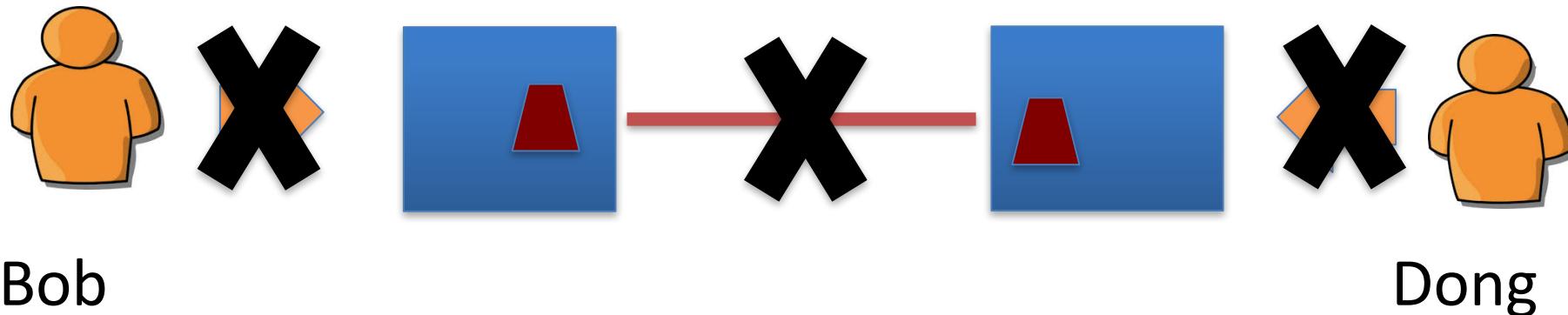
Hotel Booking: are we double-booking the same room?



CAP Theorem

- A simple example:

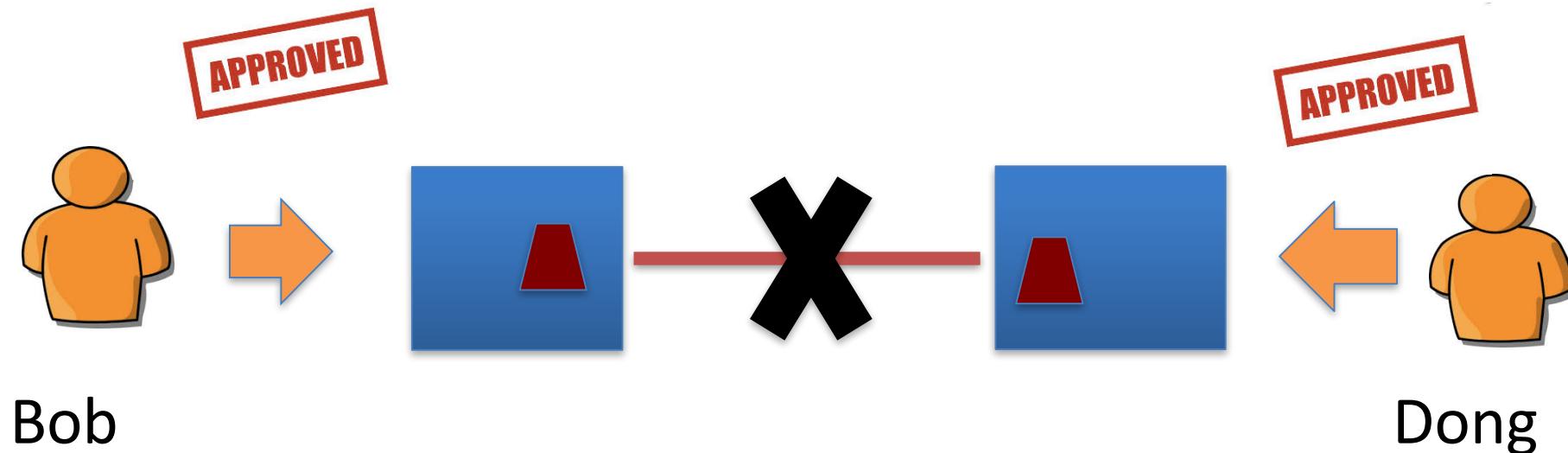
Hotel Booking: are we double-booking the same room?



CAP Theorem

- A simple example:

Hotel Booking: are we double-booking the same room?



Why this is important?

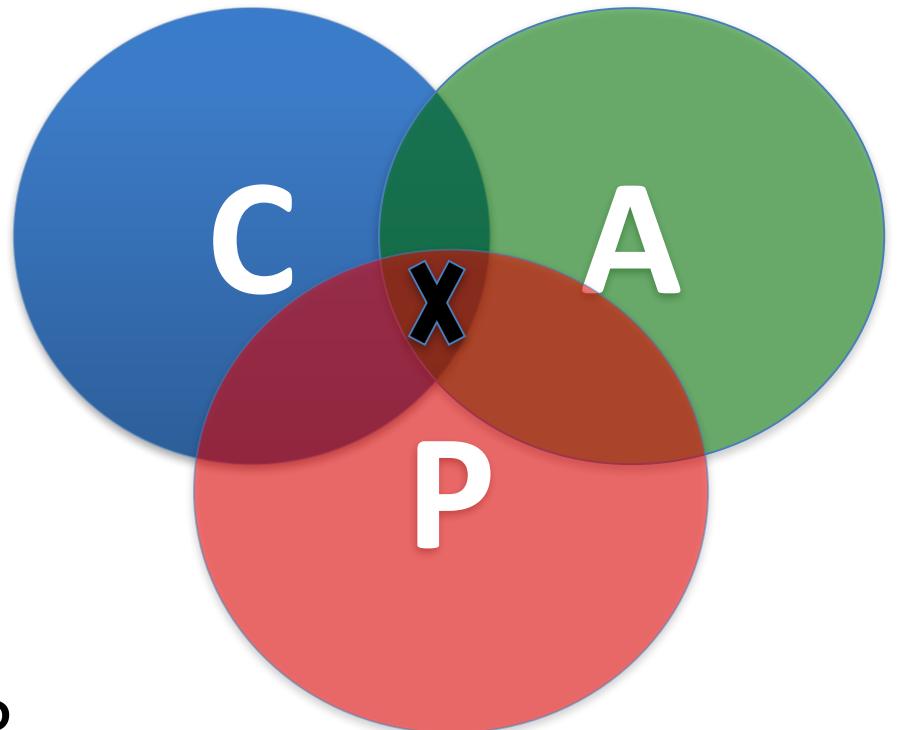
- The future of databases is **distributed** (Big Data Trend, etc.)
- CAP theorem describes the **trade-offs** involved in distributed systems
- A proper understanding of CAP theorem is essential to **making decisions** about the future of distributed database **design**
- Misunderstanding can lead to **erroneous or inappropriate** design choices

Problem for Relational Database to Scale

- 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** ...

Consistency or Availability

- Consistency and Availability is not “binary” decision
- AP systems relax consistency in favor of availability – but are not inconsistent
- CP systems sacrifice availability for consistency- but are not unavailable
- This suggests both AP and CP systems can offer a degree of consistency, and availability, as well as partition tolerance



AP: Best Effort Consistency

- Example:
 - Web Caching
 - DNS
- Trait:
 - Optimistic
 - Expiration/Time-to-live
 - Conflict resolution

Types of Consistency

- Strong Consistency
 - After the update completes, **any subsequent access** will return the **same** updated value.
- Weak Consistency
 - It is **not guaranteed** that subsequent accesses will return the updated value.
- **Eventual Consistency**
 - Specific form of weak consistency
 - It is guaranteed that if **no new updates** are made to object, **eventually** all accesses will return the last updated value (e.g., *propagate updates to replicas in a lazy fashion*)

Eventual Consistency Variations

- Causal consistency
 - Processes that have causal relationship will see consistent data
- Read-your-write consistency
 - A process always accesses the data item after its update operation and never sees an older value
- Session consistency
 - As long as session exists, system guarantees read-your-write consistency
 - Guarantees do not overlap sessions

Eventual Consistency Variations

- Monotonic read consistency
 - If a process has seen a particular value of data item, any subsequent processes will never return any previous values
- Monotonic write consistency
 - The system guarantees to serialize the writes by the *same* process
- In practice
 - A number of these properties can be combined
 - Monotonic reads and read-your-writes are most desirable

Eventual Consistency

- A Facebook Example

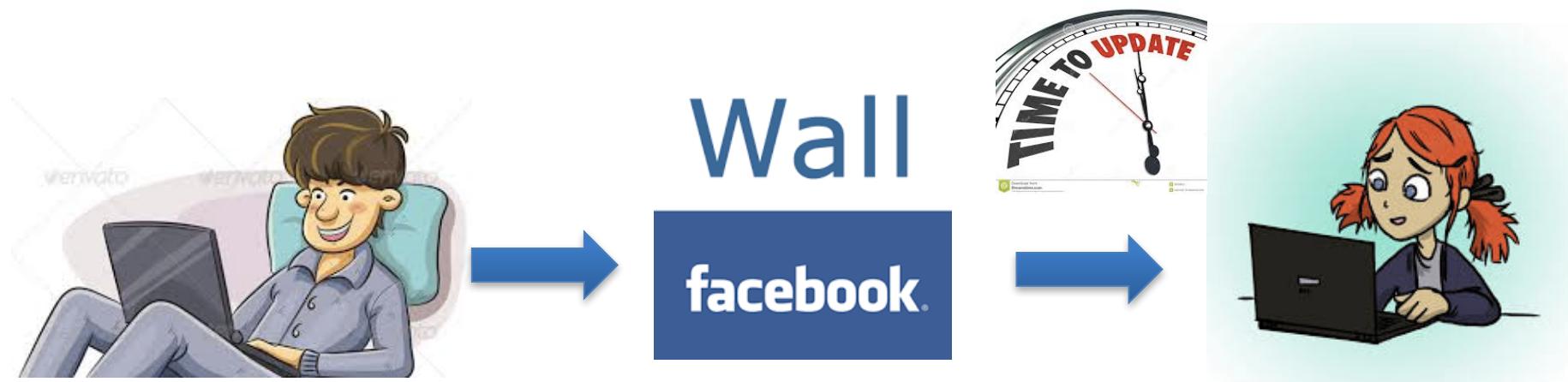
- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:
 - **Nothing is there!**



Eventual Consistency

- A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:
 - She finds the story Bob shared with her!



Eventual Consistency

- A Facebook Example

- Reason: it is possible because Facebook uses an **eventual consistent model**
- Why Facebook chooses eventual consistent model over the strong consistent one?
 - Facebook has more than 1 billion active users
 - It is non-trivial to efficiently and reliably store the huge amount of data generated at any given time
 - Eventual consistent model offers the option to **reduce the load and improve availability**

Eventual Consistency

- A Dropbox Example

- Dropbox enabled immediate consistency via synchronization in many cases.
- However, what happens in case of a network partition?



Eventual Consistency

- A Dropbox Example

- Let's do a simple experiment here:
 - Open a file in your drop box
 - Disable your network connection (e.g., WiFi, 4G)
 - Try to edit the file in the drop box: can you do that?
 - Re-enable your network connection: what happens to your dropbox folder?

Eventual Consistency

- A Dropbox Example

- Dropbox embraces eventual consistency:
 - Immediate consistency is impossible in case of a network partition
 - Users will feel bad if their word documents freeze each time they hit Ctrl+S , simply due to the large latency to update all devices across WAN
 - Dropbox is oriented to **personal syncing**, not on collaboration, so it is not a real limitation.

Eventual Consistency

- An ATM Example

- In design of automated teller machine (ATM):
 - Strong consistency appear to be a nature choice
 - However, in practice, **A beats C**
 - Higher availability means **higher revenue**
 - ATM will allow you to withdraw money *even if the machine is partitioned from the network*
 - However, it puts a **limit** on the amount of withdraw (e.g., \$200)
 - The bank might also charge you a fee when a overdraft happens



Dynamic Tradeoff between C and A

- An airline reservation system:
 - When most of seats are available: it is ok to rely on somewhat out-of-date data, availability is more critical
 - When the plane is close to be filled: it needs more accurate data to ensure the plane is not overbooked, consistency is more critical
- Neither strong consistency nor guaranteed availability, but it may significantly increase the tolerance of network disruption

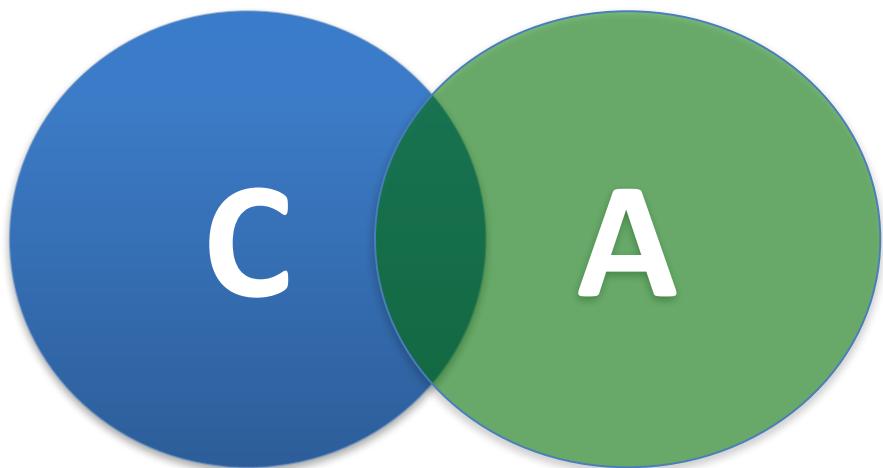
What if there are no partitions?

- Tradeoff between **Consistency** and **Latency**:
- Caused by the **possibility of failure** in distributed systems
 - High availability -> replicate data -> consistency problem
- Basic idea:
 - Availability and latency are arguably **the same thing**: unavailable -> extreme high latency
 - Achieving different levels of consistency/availability takes different amount of time

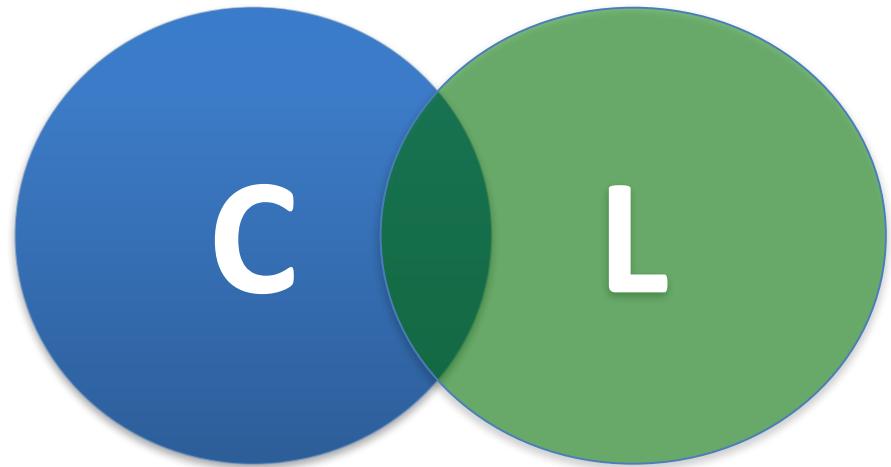
CAP -> PACELC

- A more complete description of the space of potential tradeoffs for distributed system:
 - If there is a **partition (P)**, how does the system trade off **availability and consistency (A and C)**;
else (E), when the system is running normally in the absence of partitions, how does the system trade off **latency (L) and consistency (C)**?

Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (2012): 37.



Partitioned



Normal

Examples

- **PA/EL Systems:** Give up both Cs for availability and lower latency
 - Dynamo, Cassandra, Riak
- **PC/EC Systems:** Refuse to give up consistency and pay the cost of availability and latency
 - BigTable, Hbase, VoltDB/H-Store
- **PA/EC Systems:** Give up consistency when a partition happens and keep consistency in normal operations
 - MongoDB
- **PC/EL System:** Keep consistency if a partition occurs but gives up consistency for latency in normal operations
 - Yahoo! PNUTS

Summary

CAP Theorem:

It is impossible for a distributed data store to offer **Consistency**, **Availability** and **Partition tolerance** at the same time.

RDBMS is based on the principle if ACID
(Atomicity, Consistency, Isolation, Durability)
=> Not possible to distribute

Thank you.



The
Center of
**Applied
Data Science**