Distributed Systems

Monsoon 2024

Lecture 19

International Institute of Information Technology

Hyderabad, India

From RDBMS to NoSQL

- Recall our discussion on distributed databases using RDBMS models and NoSQL models.
- The RDBMS models ensure ACID property
- NoSQL models ensure BASE property
- From a user perspective, three guarantees matter:

From RDBMS to NoSQL

- From a user perspective, three guarantees matter:
- Consistency:
  - All replicas contain the same version of data
  - Client always has the same view of the data (no matter what node)
- Availability
  - System remains operational on failing nodes
  - All clients can always read and write
- Partition tolerance
  - System works well across physical network partitions

From RDBMS to NoSQL

- Consistency:
  - All replicas contain the same version of data
  - Client always has the same view of the data (no matter what node)
- Availability
  - System remains operational on failing nodes
  - All clients can always read and write
- Partition tolerance
  - System works well across physical network partitions

- Can any system support all three of Consistency, Availability, and Partition Tolerance?

From RDBMS to NoSQL

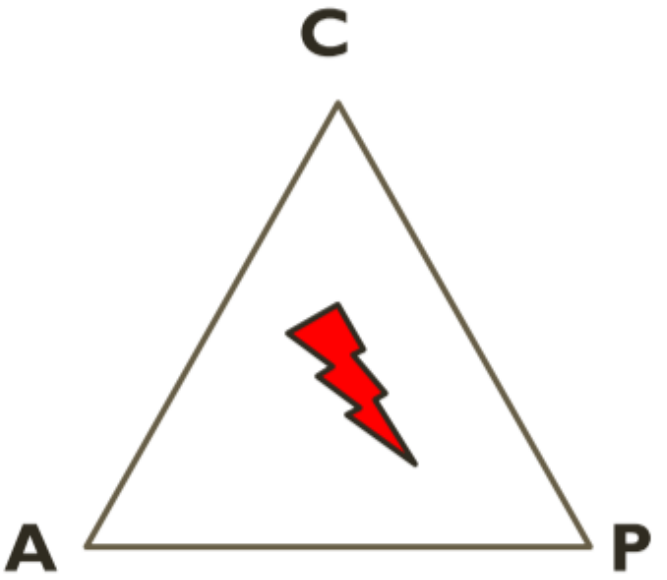
- Why does not any of them support all three of Consistency, Availability, and Partition Tolerance?

- Fundamentally, there is a limit.
- Informally called as the CAP Theorem.
- Any distributed system can ensure only two of the above three properties!

The CAP Theorem

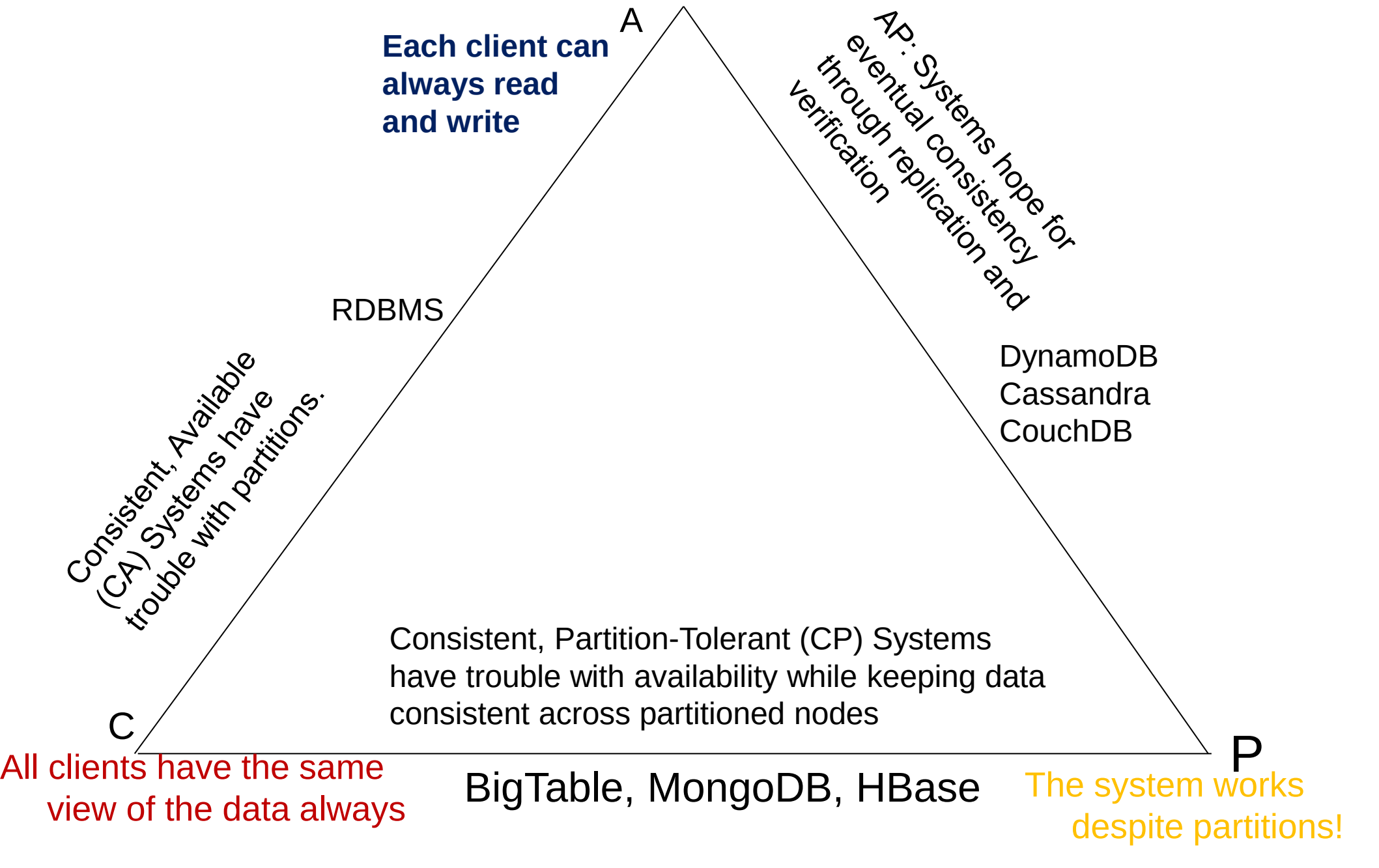
Given a distributed system with many nodes and replication of data:

- Consistency:
  - All replicas contain the same version of data
  - Client always has the same view of the data (no matter what node)
- Availability
  - System remains operational on failing nodes
  - All clients can always read and write
- Partition tolerance
  - multiple entry points
  - System works well across physical network partitions



CAP Theorem:
satisfying all three at the same time is impossible

The CAP Theorem and Systems



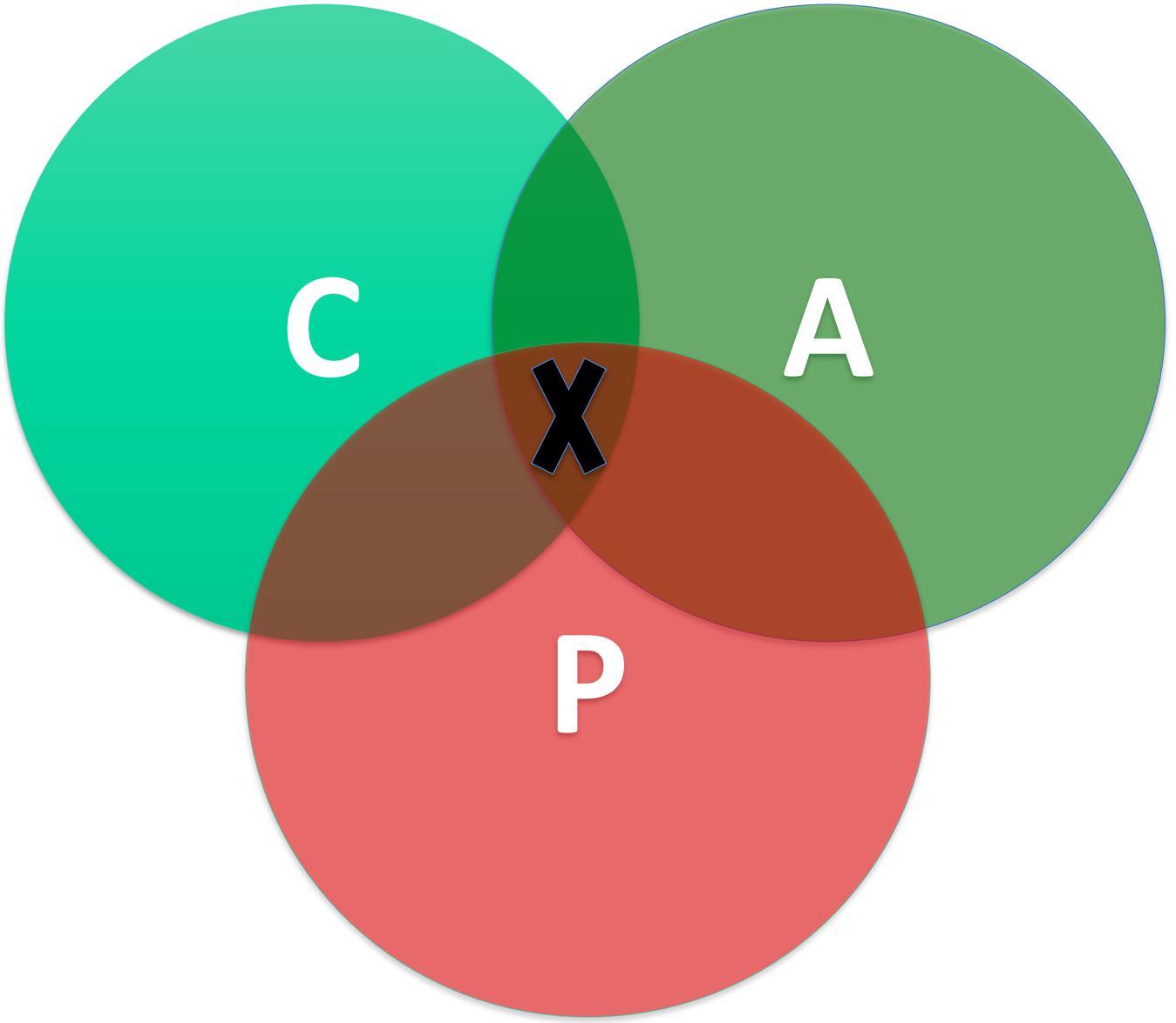
CAP Theorem

- The observation that got its name as the CAP theorem is made by Eric Brewer in 2001.
- If you cannot limit the number of faults and requests can be directed to any server and you insist on serving every request you receive then you cannot possibly be consistent.

CAP Theorem

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

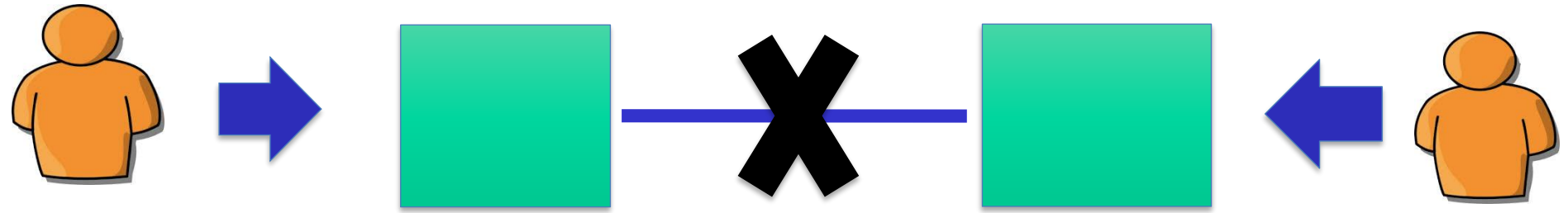
CAP Theorem



CAP Theorem

- A simple example:

  Booking a Seat: double-booking the same seat?



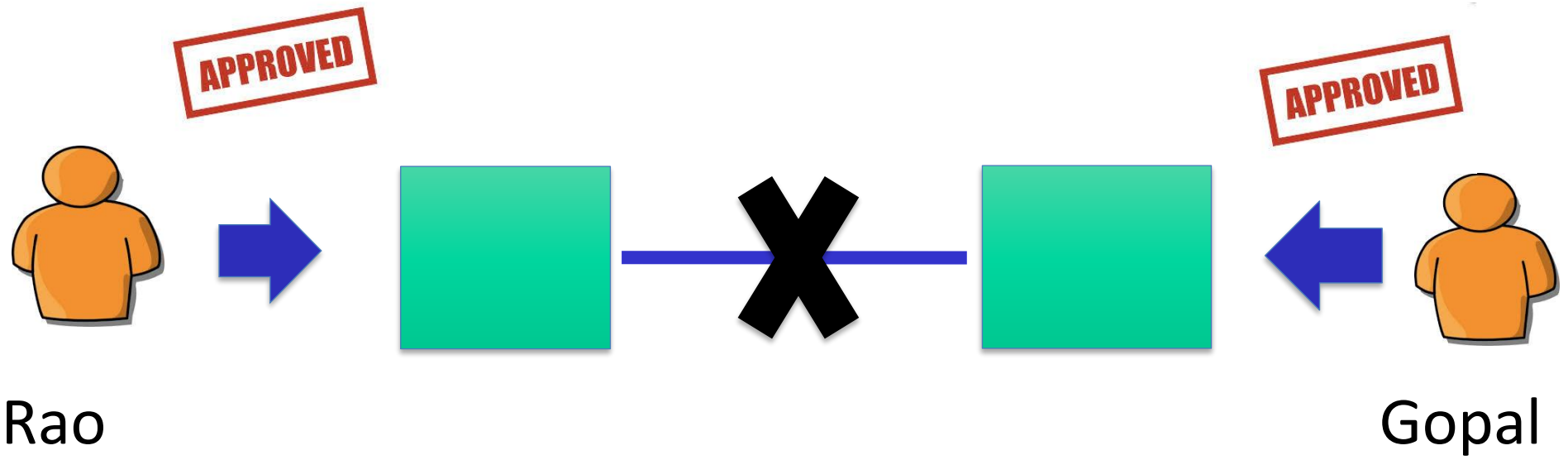
Rao

Gopal

CAP Theorem

- A simple example:

  Booking a Seat: double-booking the same seat?



CAP Theorem: Proof

- 2002: Proven by research conducted by Nancy Lynch and Seth Gilbert at MIT

Gilbert, Seth, and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." ACM SIGACT News 33.2 (2002): 51-59.

Why this is important?

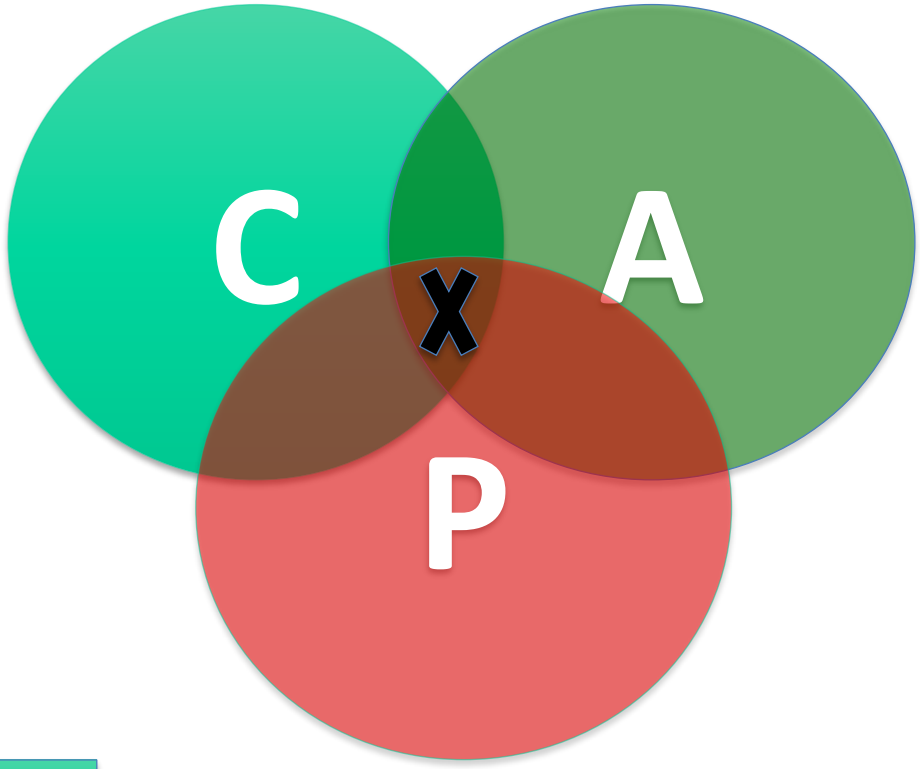
- The future of databases is **distributed** (Big Data Trend, NoSQL, …)
- CAP theorem describes the **trade-offs** involved in distributed systems
- A proper understanding of CAP theorem is essential to **making decisions** about the future of distributed database **design**
- Misunderstanding can lead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

Revisit CAP Theorem

- Of the following three guarantees potentially to be offered by distributed systems:
  - Consistency
  - Availability
  - Partition tolerance



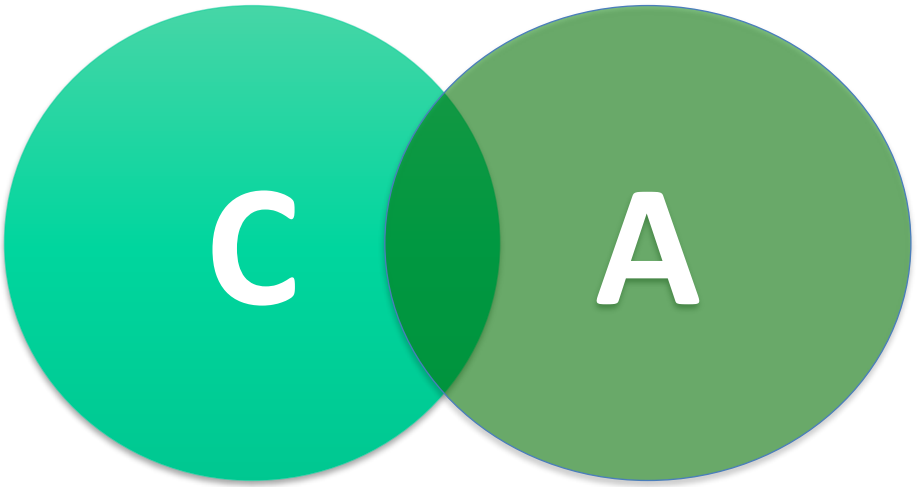
- Pick two

- This suggests there are three kinds of distributed systems:
  - CP
  - AP
  - CA

Any problems?

A popular misconception: 2 out 3

- How about CA?
- Can a distributed system (with unreliable network) really be not tolerant of partitions?



A few witnesses

- Coda Hale, Yammer software engineer:
  - "Of the CAP theorem's Consistency, Availability, and Partition Tolerance, **Partition Tolerance is mandatory in distributed systems**. You cannot not choose it."

http://codahale.com/you-cant-sacrifice-partition-tolerance/

A few witnesses

- Werner Vogels, Amazon CTO
  - "An important observation is that in larger distributed-scale systems, network partitions are a given; therefore, **consistency and availability cannot be achieved at the same time**."

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

A few witnesses

- Daneil Abadi, Co-founder of Hadapt
  - So in reality, there are only two types of systems ... I.e., if there is a partition, **does the system give up availability or consistency?**

http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html

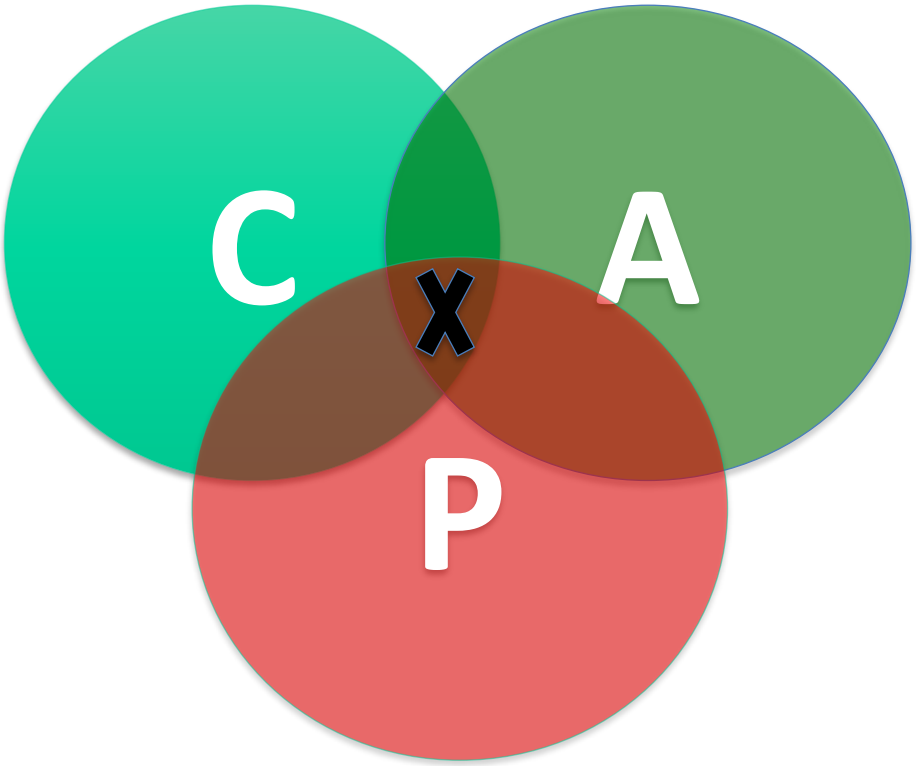
CAP Theorem 12 year later

- Eric Brewer: father of CAP theorem
  - "The "2 of 3" formulation was always **misleading** because it tended to oversimplify the tensions among properties. ...
  - **CAP prohibits only a tiny part of the design space**: *perfect availability and consistency in the presence of partitions*, which are rare."

http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

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

CP: Best Effort Availability

- Example:
  - Majority protocols
  - Distributed Locking (Google Chubby Lock service)
- Trait:
  - Pessimistic locking
  - Make minority partition unavailable

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 it's 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

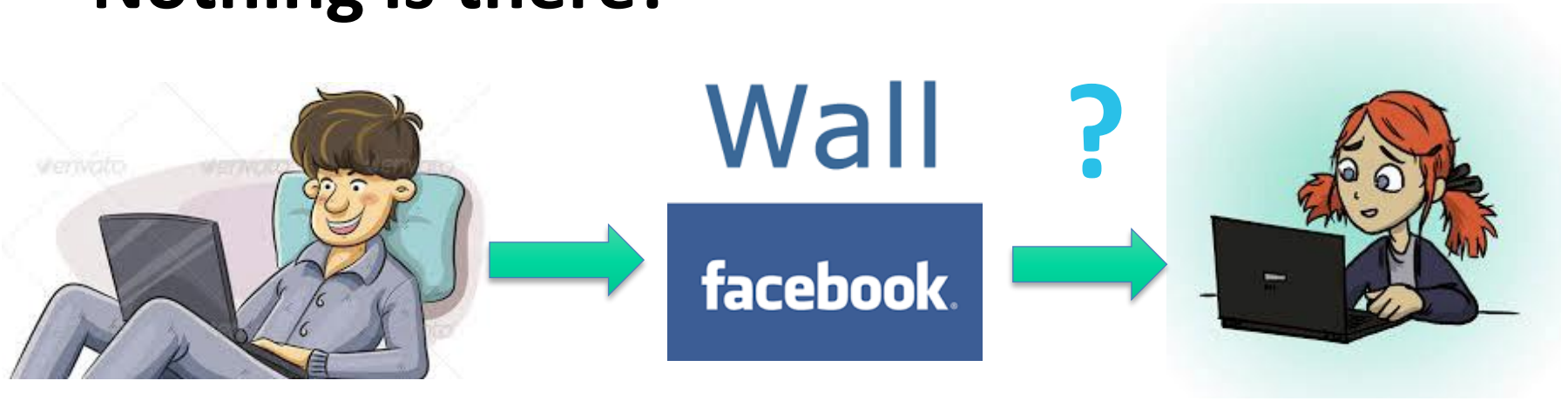
Eventual Consistency Variations

- 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
  - 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 natural 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 can put **a limit** on the amount of withdraw. The bank might also charge you a fee when an 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

Heterogeneity: Segmenting **C** and **A**

- No single uniform requirement
  - Some aspects require strong consistency
  - Others require high availability
- Segment the system into different components
  - Each provides different types of guarantees
- Overall guarantees neither consistency nor availability
  - Each part of the service gets exactly what it needs
- Can be partitioned along different dimensions

Discussion

- In an e-commerce system (e.g., Amazon, Flipkart, etc), what are the trade-offs between consistency and availability you can think of? What is your strategy?
- Hint -> Things you might want to consider:
  - Different types of data (e.g., shopping cart, billing, product, etc.)
  - Different types of operations (e.g., query, purchase, etc.)
  - Different types of services (e.g., distributed lock, DNS, etc.)
  - Different groups of users (e.g., users in different geographic areas, etc.)

Partitioning Examples

- Data Partitioning
- Operational Partitioning
- Functional Partitioning
- User Partitioning
- Hierarchical Partitioning

Partitioning Examples

Data Partitioning

- Different data may require different consistency and availability
- Example:
  - Shopping cart: high availability, responsive, can sometimes suffer anomalies
  - Product information need to be available, slight variation in inventory is sufferable
  - Checkout, billing, shipping records must be consistent

Partitioning Examples

Operational Partitioning

- Each operation may require different balance between consistency and availability
- Example:
  - Reads: high availability; e.g.., “query”
  - Writes: high consistency, lock when writing; e.g., “purchase”

Partitioning Examples

Functional Partitioning

- System consists of sub-services
- Different sub-services provide different balances
- Example: A comprehensive distributed system
  - Distributed lock service (e.g., Chubby) :
    - Strong consistency
  - DNS service:
    - High availability

Partitioning Examples

User Partitioning

- Try to keep related data close together to assure better performance
- Example: Craglist
  - Might want to divide its service into several data centers, e.g., east coast and west coast
    - Users get high performance (e.g., high availability and good consistency) if they query servers closet to them
    - Poorer performance if a New York user query Craglist in San Francisco

Partitioning Examples

Hierarchical Partitioning

- Large global service with local “extensions”
- Different location in hierarchy may use different consistency
- Example:
  - Local servers (better connected) guarantee more consistency and availability
  - Global servers has more partition and relax one of the requirement

Can Any 2-out-of-3 be Achieved?

- We may want to get some convincing answers on this question.