CE/CZ4052
Cloud Computing

CAP Theorem

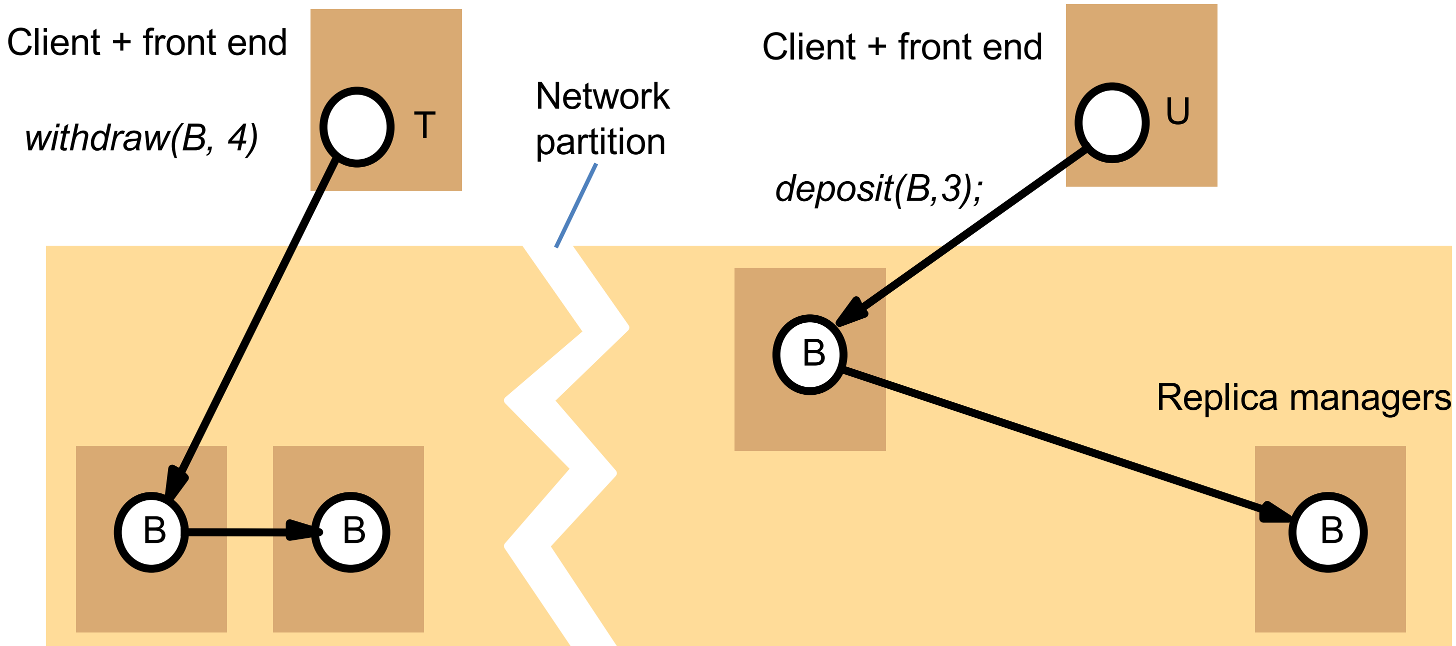
Dr. Tan, Chee Wei

Email: cheewei.tan@ntu.edu.sg

Office: N4-02c-104

Tradeoffs in Distributed System

- Let's just do best effort to make things consistent.
- Eventual consistency
  - Popularized by the CAP theorem.
  - The main problem is network partitions.



CAP Theorem

- Conjectured by Prof. Eric Brewer at PODC (Principle of Distributed Computing) 2000 keynote talk

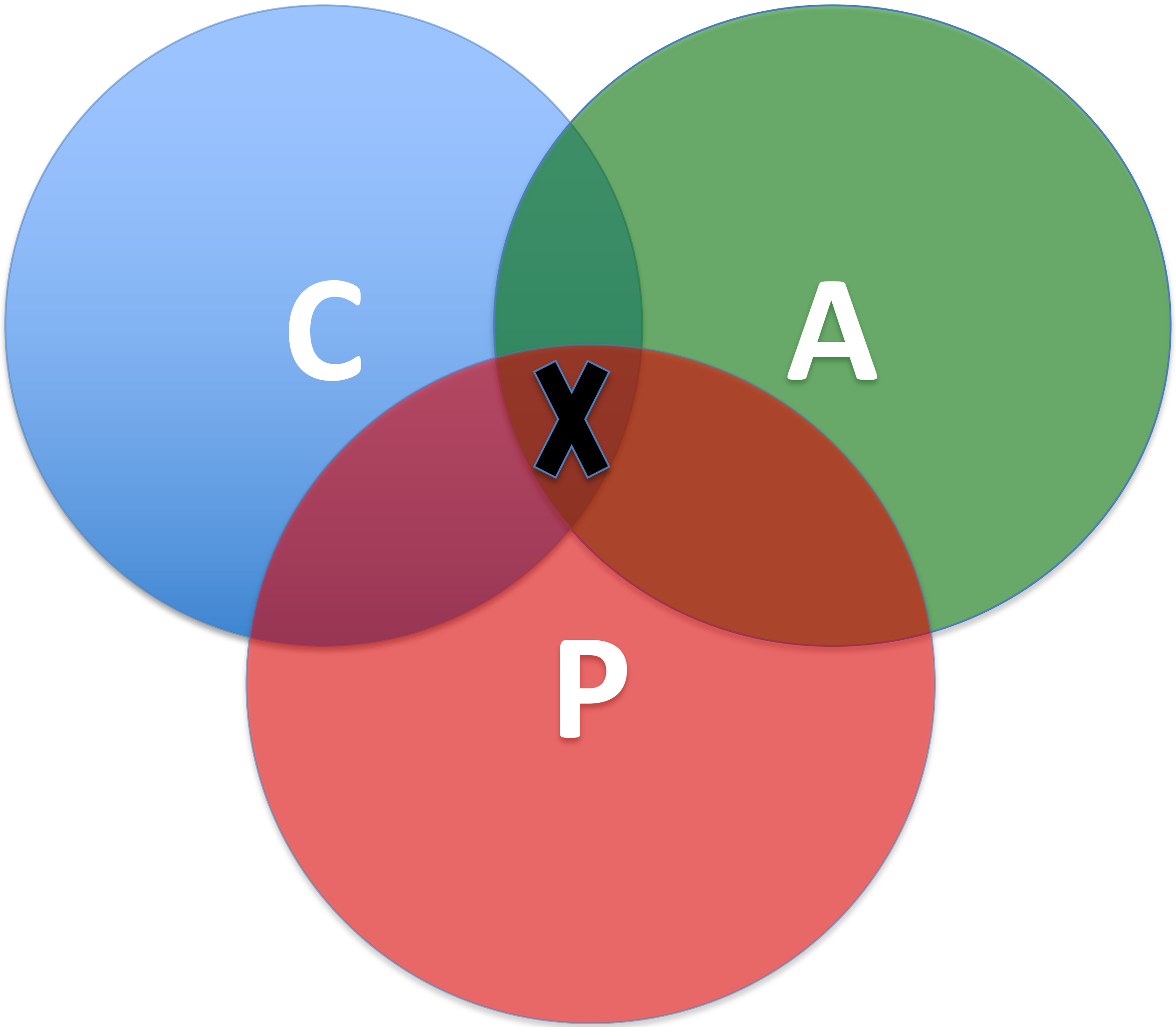
- Described the *trade-offs involved in distributed system*

- It is impossible for a web service to provide following *three guarantees at the same time*:
  - **Consistency**
  - **Availability**
  - **Partition-tolerance**

CAP Theorem

- **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
- A distributed system can satisfy any two of these guarantees at the same time **but not all three**

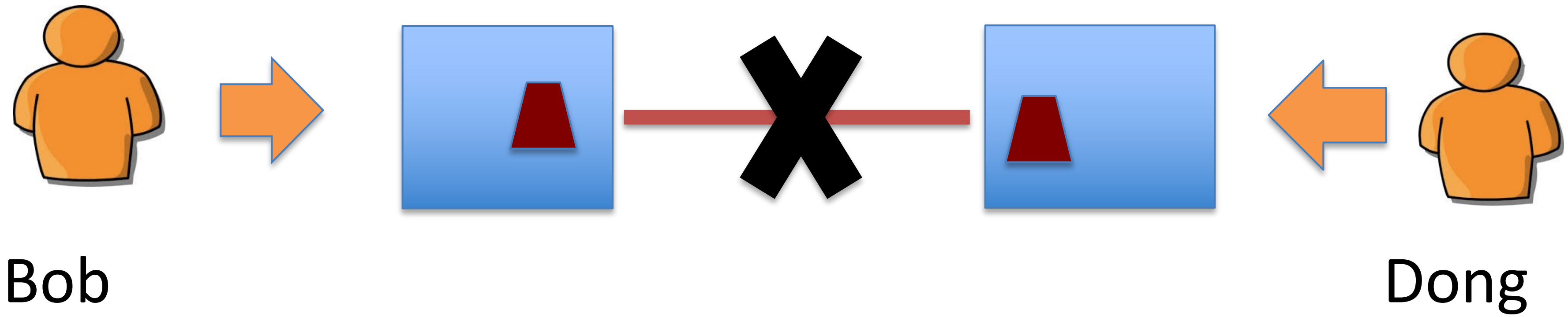
CAP Theorem



CAP Theorem

- A simple example:

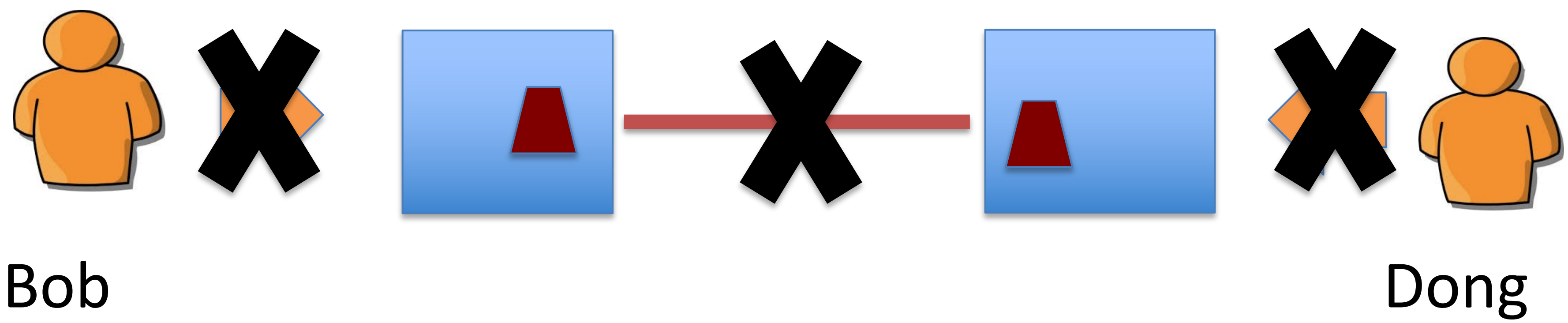
Hotel Booking: are we double-booking the same room?



CAP Theorem

- A simple example:

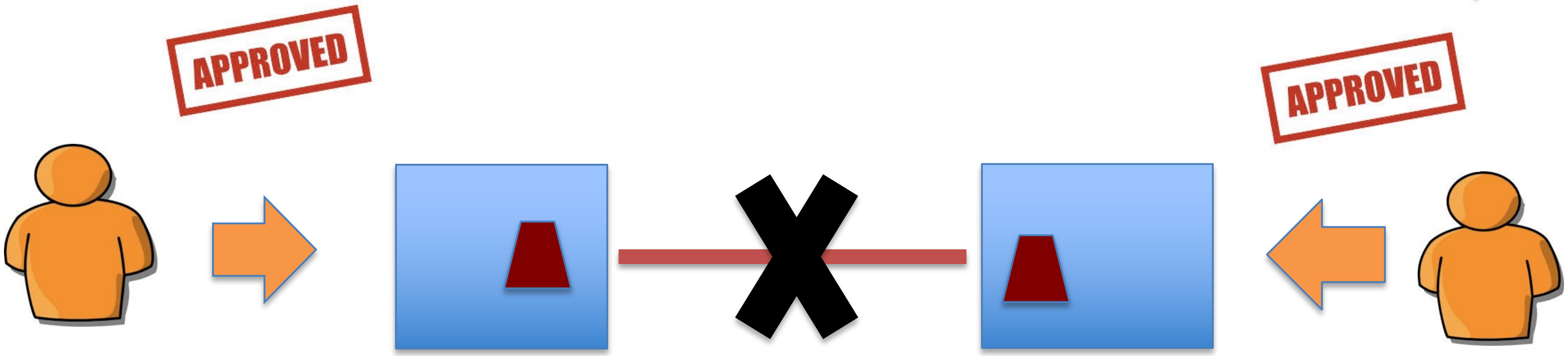
Hotel Booking: are we double-booking the same room?



CAP Theorem

- A simple example:

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



Bob

Dong

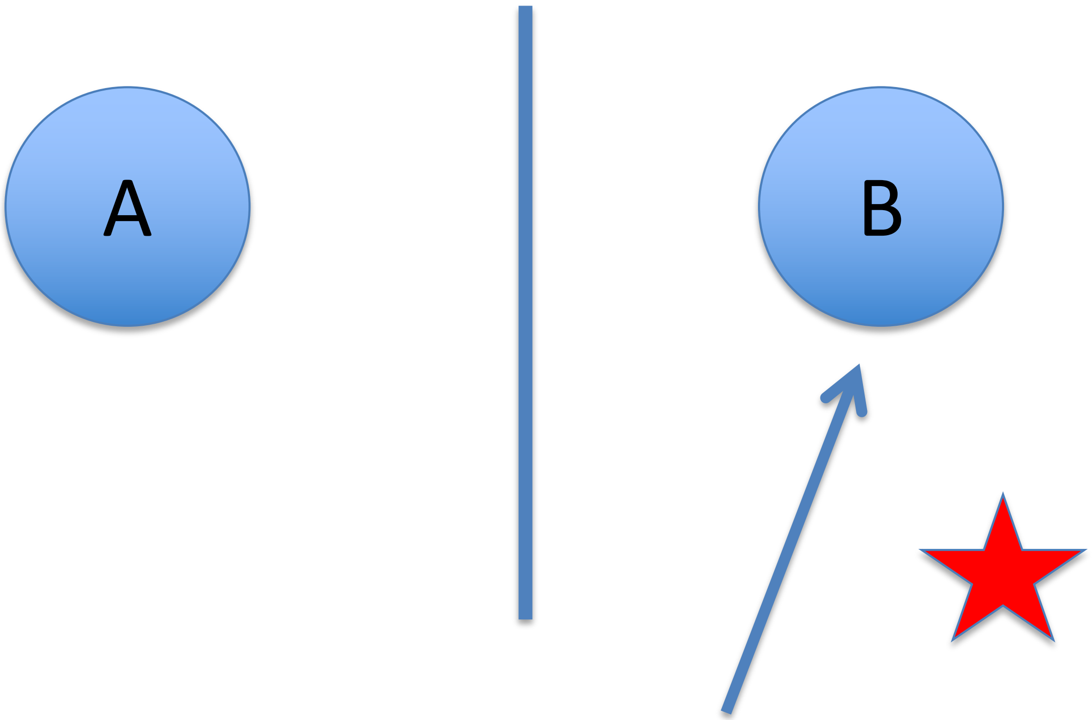
CAP Theorem: Proof

- 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.

CAP Theorem: Proof

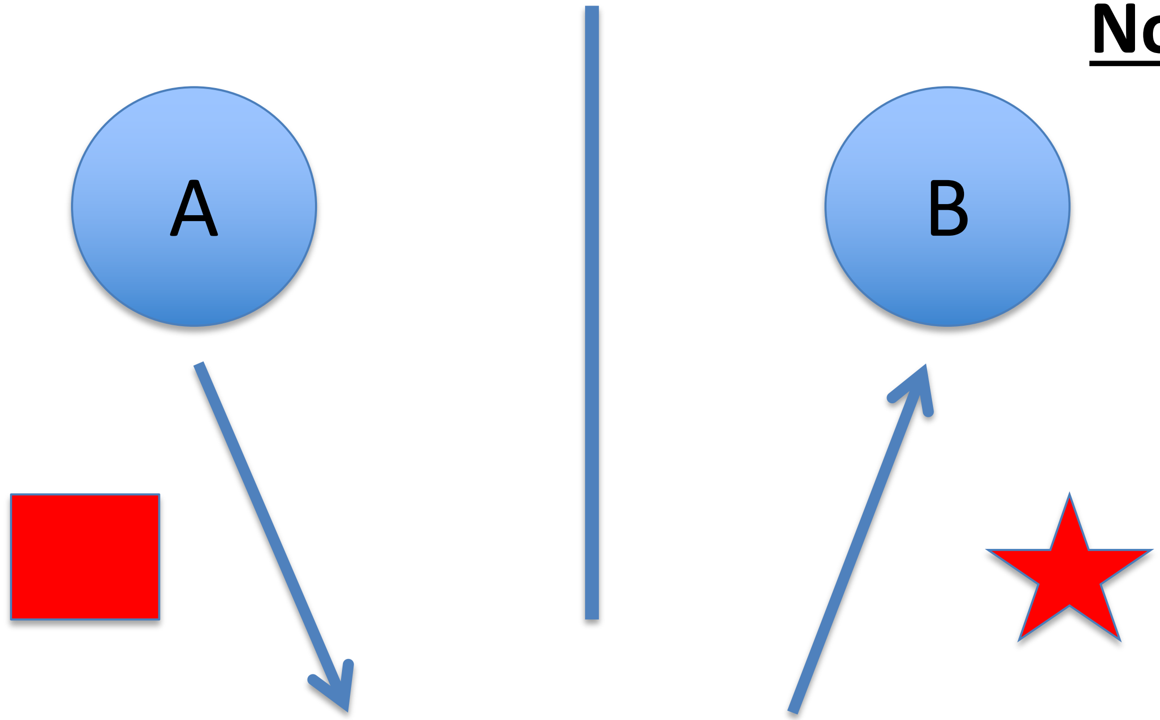
- A simple proof using two nodes:



CAP Theorem: Proof

- A simple proof using two nodes:

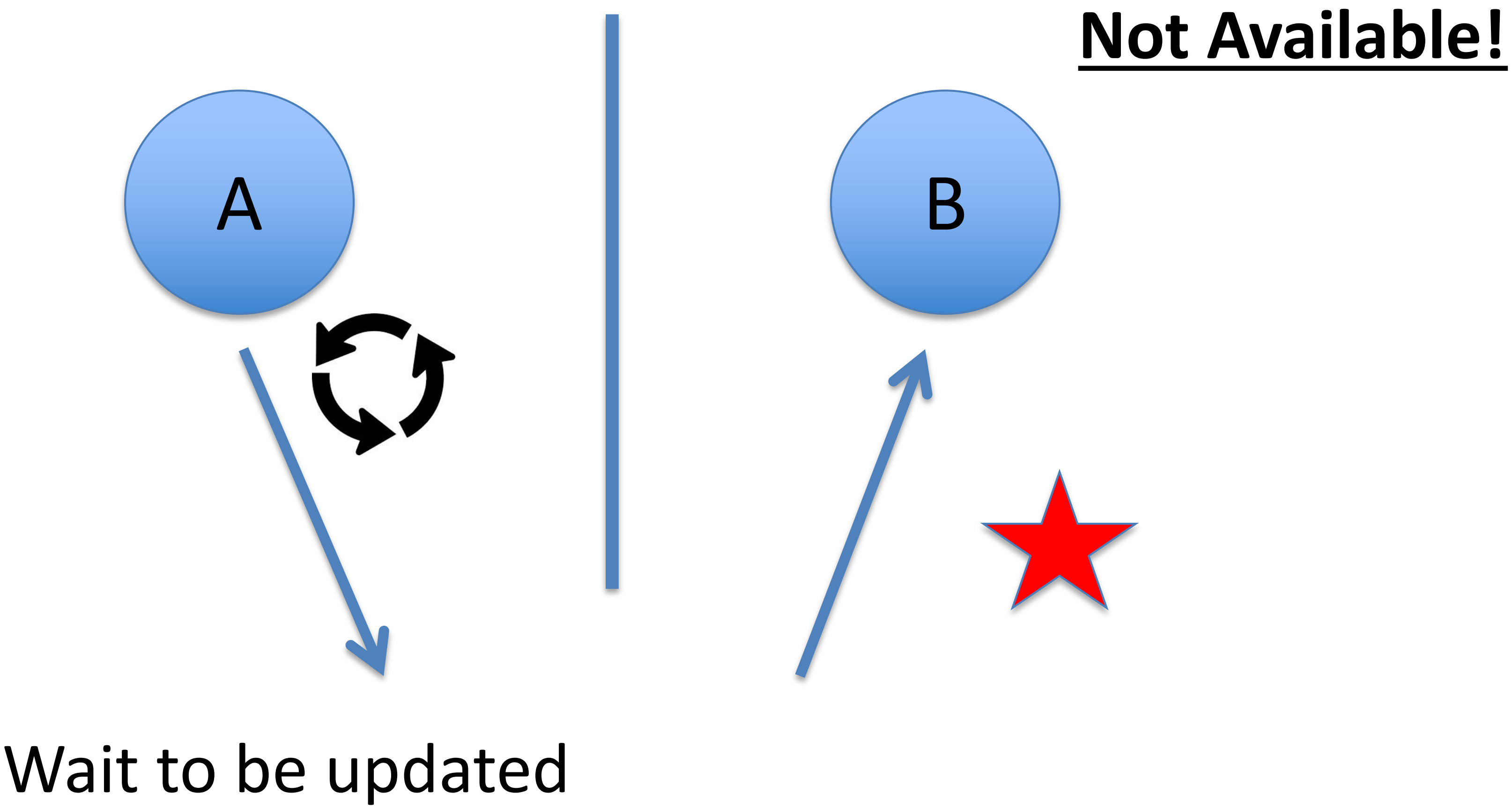
Not Consistent!



Respond to client

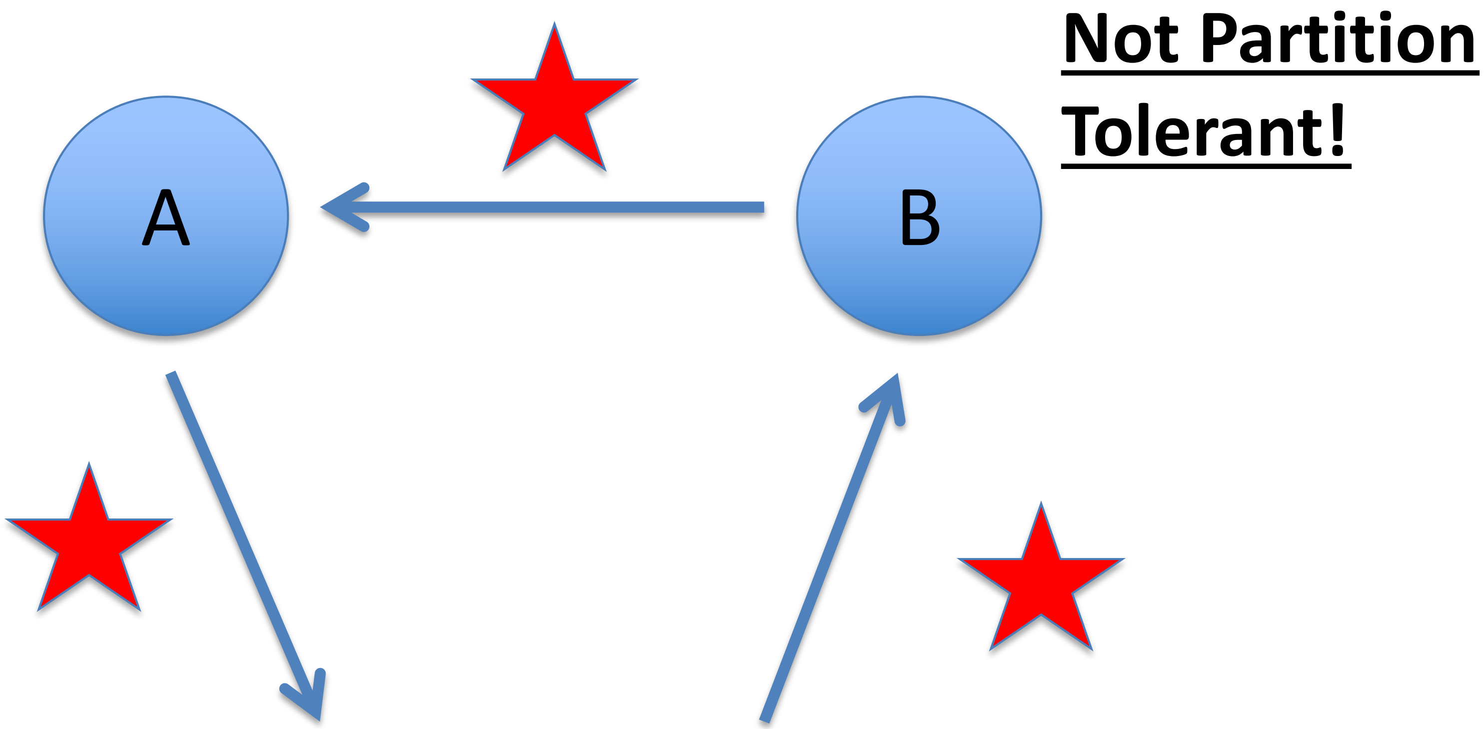
CAP Theorem: Proof

- A simple proof using two nodes:



CAP Theorem: Proof

- A simple proof using two nodes:



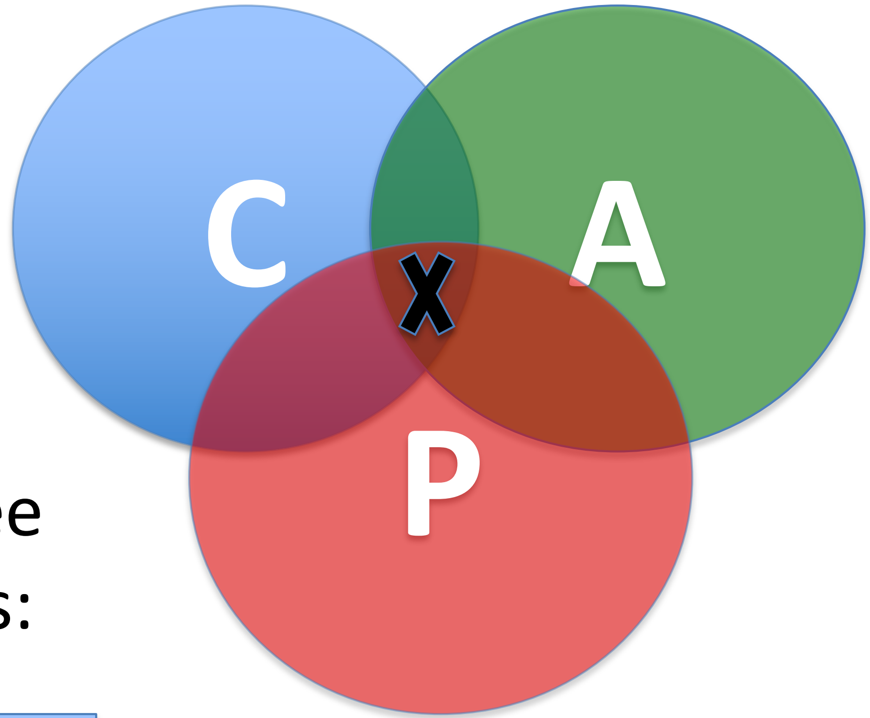
A gets updated from B

Why this is important?

- The future of cloud computing 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 system **design**
- Misunderstanding can lead to **erroneous or inappropriate** design choices

Revisit CAP Theorem

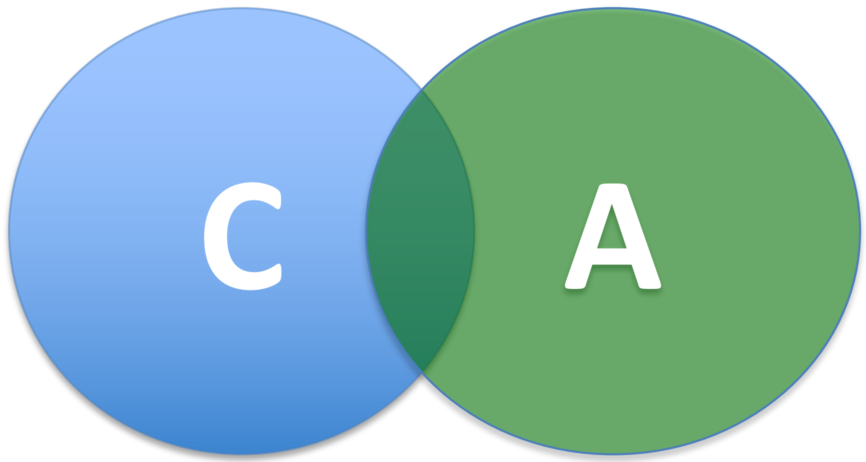
- Of the following three guarantees potentially offered a by distributed systems:
    - Consistency
    - Availability
    - Partition tolerance
- Pick two
- This suggests there are three kinds of distributed systems:
    - CP
    - AP
    - CA



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/

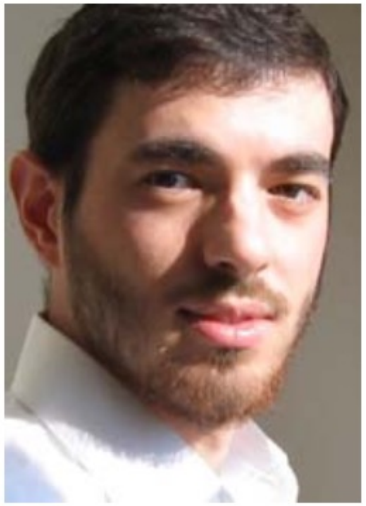
A few witnesses

- 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

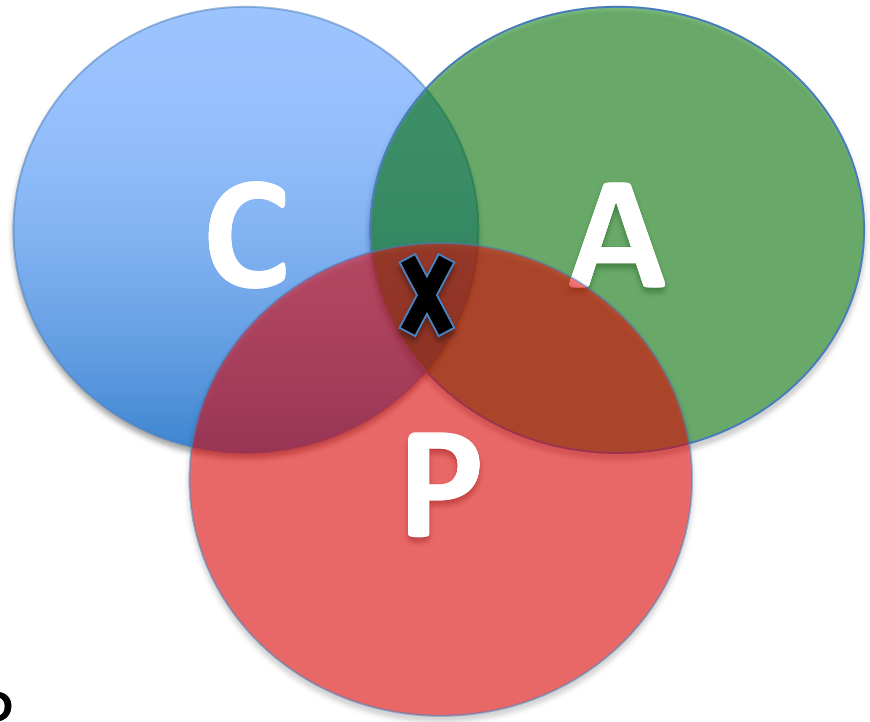
- Daneil Abadi, Co-founder of Hadapt
  - So in reality, there are only two types of systems ... I.e., if there is a partition, **does the system give up availability or consistency?**



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

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

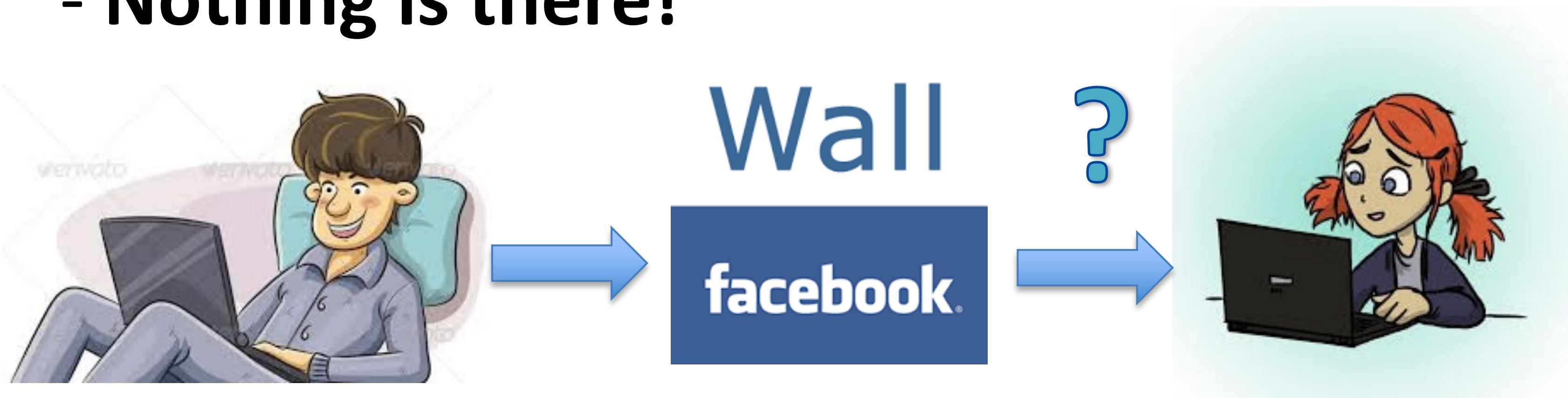
- Example:
  - Majority protocols
  - Distributed Locking (Google Chubby Lock service)
- Trait:
  - Pessimistic locking
  - Make minority partition unavailable

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*)

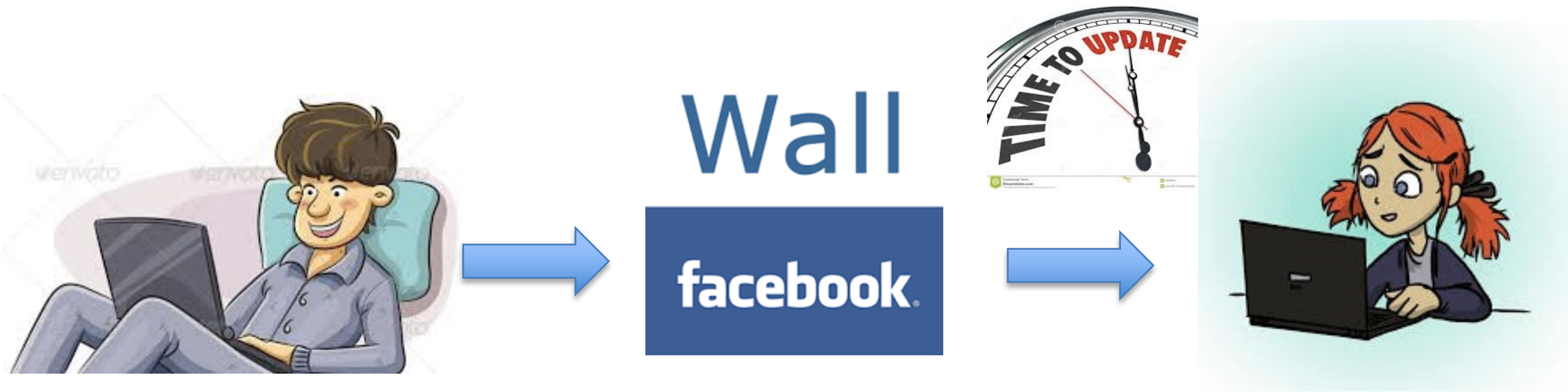
Eventual Consistency
- A Facebook Example

- 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

Discussion

- In a cloud computing system (e.g., Amazon, Google cloud, 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.)

Summary of CAP Theorem

- In the presence of a network partition:
- In order to keep the replicas consistent, you need to block.
  - From an outside observer, the system appears to be unavailable.
- If we still serve the requests from two partitions, then the replicas will diverge.
  - The system is available, but no consistency.
- The CAP theorem explains this dilemma.