Multicore Computing
Lecture23 – CAP Theorem

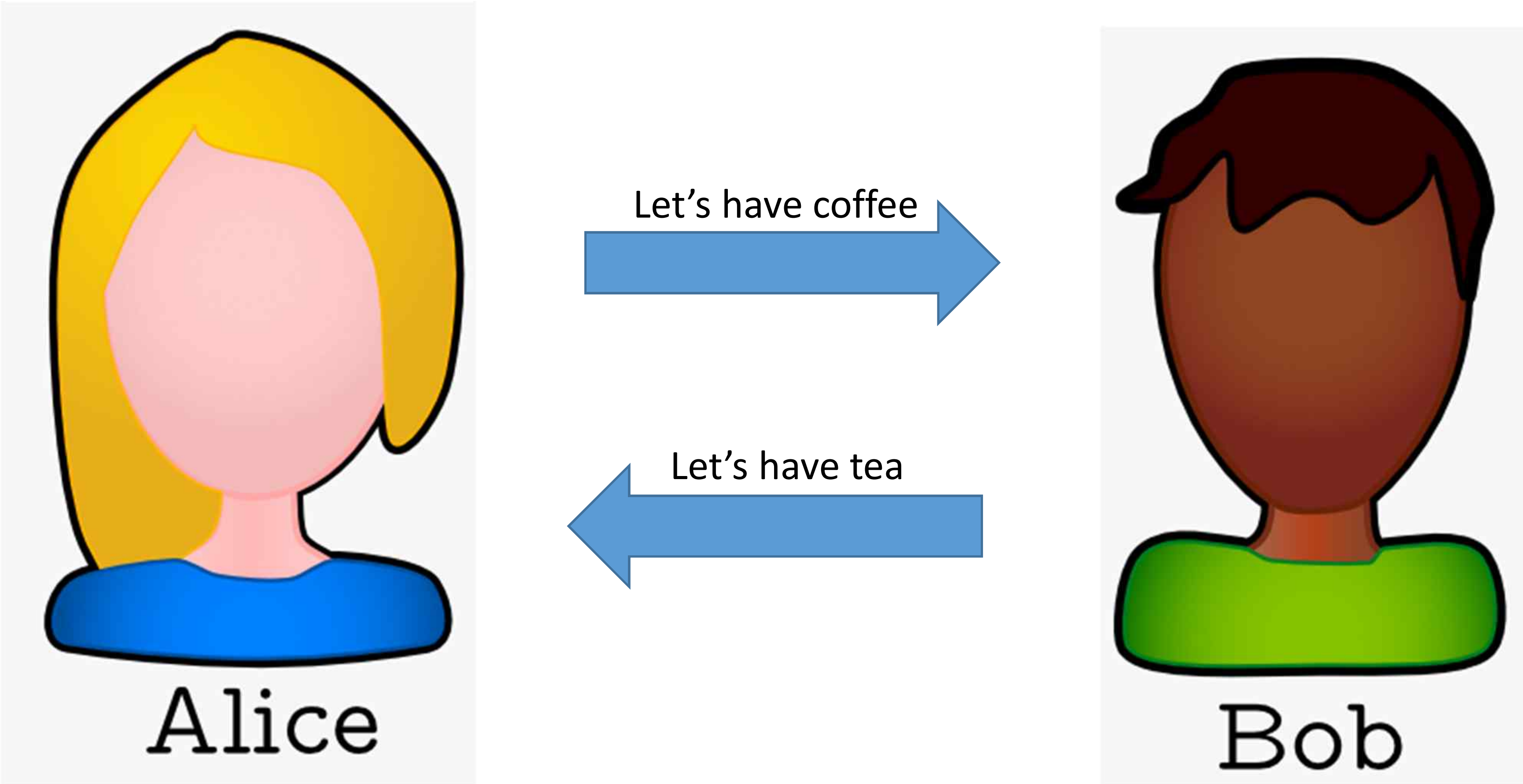
SUNG KYUN KWAN UNIVERSITY

남 범 석

bnam@skku.edu

Slides from prof Dong Wang (U. Notre Dame)
https://www3.nd.edu/~dthain/courses/cse40822/fall2014/slides/cse40822-CAP.pptx

Distributed Consensus

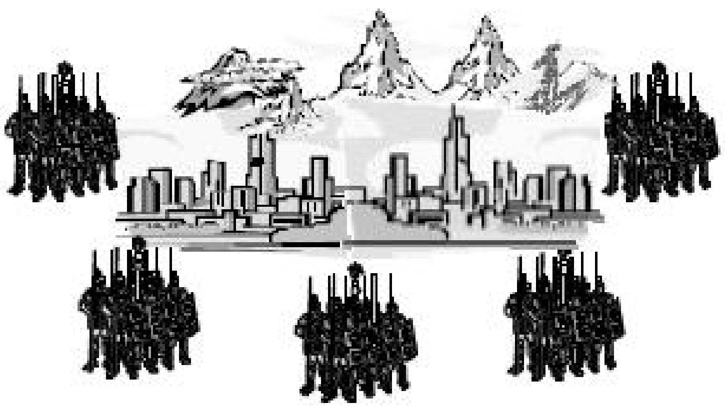


Agreement in Faulty Systems

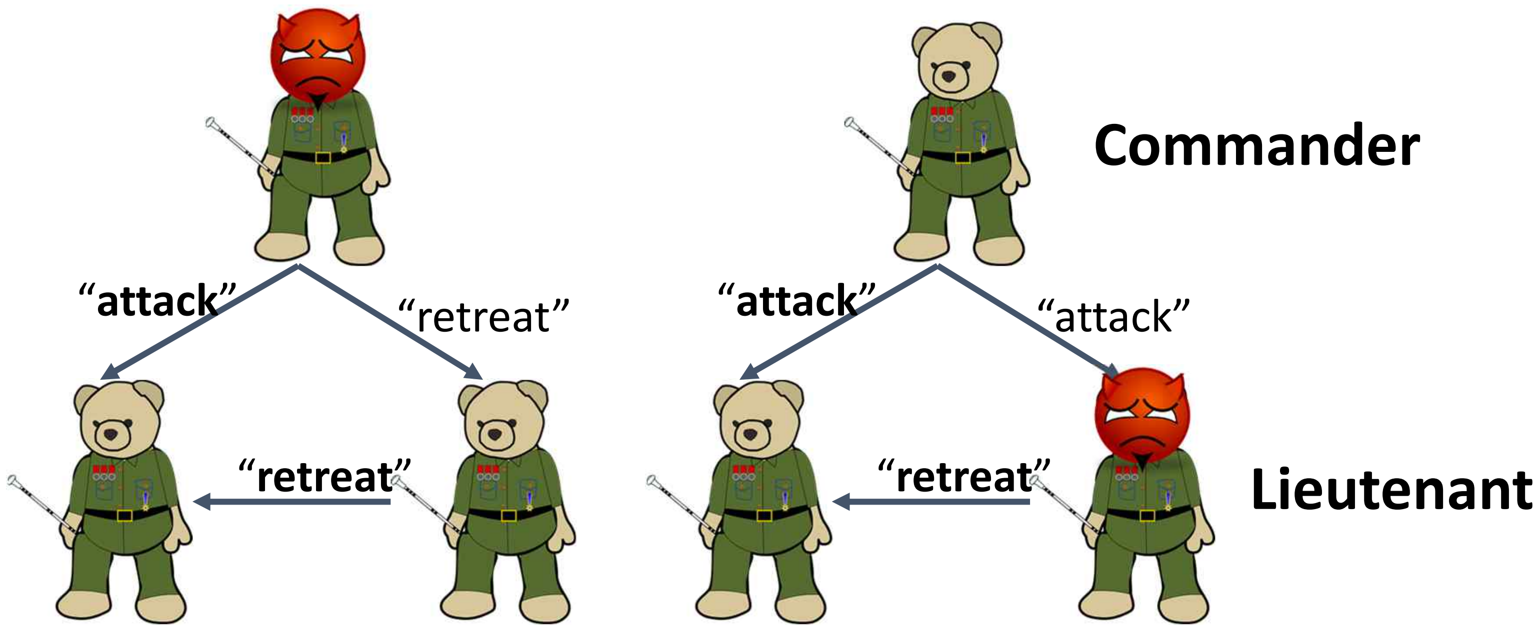
- Reaching agreement using E-mail becomes even harder.
  - Alice → Bob : Let's meet at noon on the 2nd floor
  - Alice ← Bob : Ok!
  - Alice (What if Bob doesn't know that I received his message?)
  - Alice → Bob : I received your message, so it's ok.
  - Bob (What if Alice doesn't know that I received her message?)
  - …

Agreement in Faulty Systems

- Byzantine Agreement (Pease, Shostak, Lamport, 1980)
  - The Byzantine generals announce their troop strengths.
  - There are loyal generals and traitors as well.
  - Goal: Each troop learns the true strengths sent by each of the loyal generals.



Agreement in Faulty Systems (n=3)



**There is no solution for
3 Generals, 1 Traitor.**

CAP Theorem

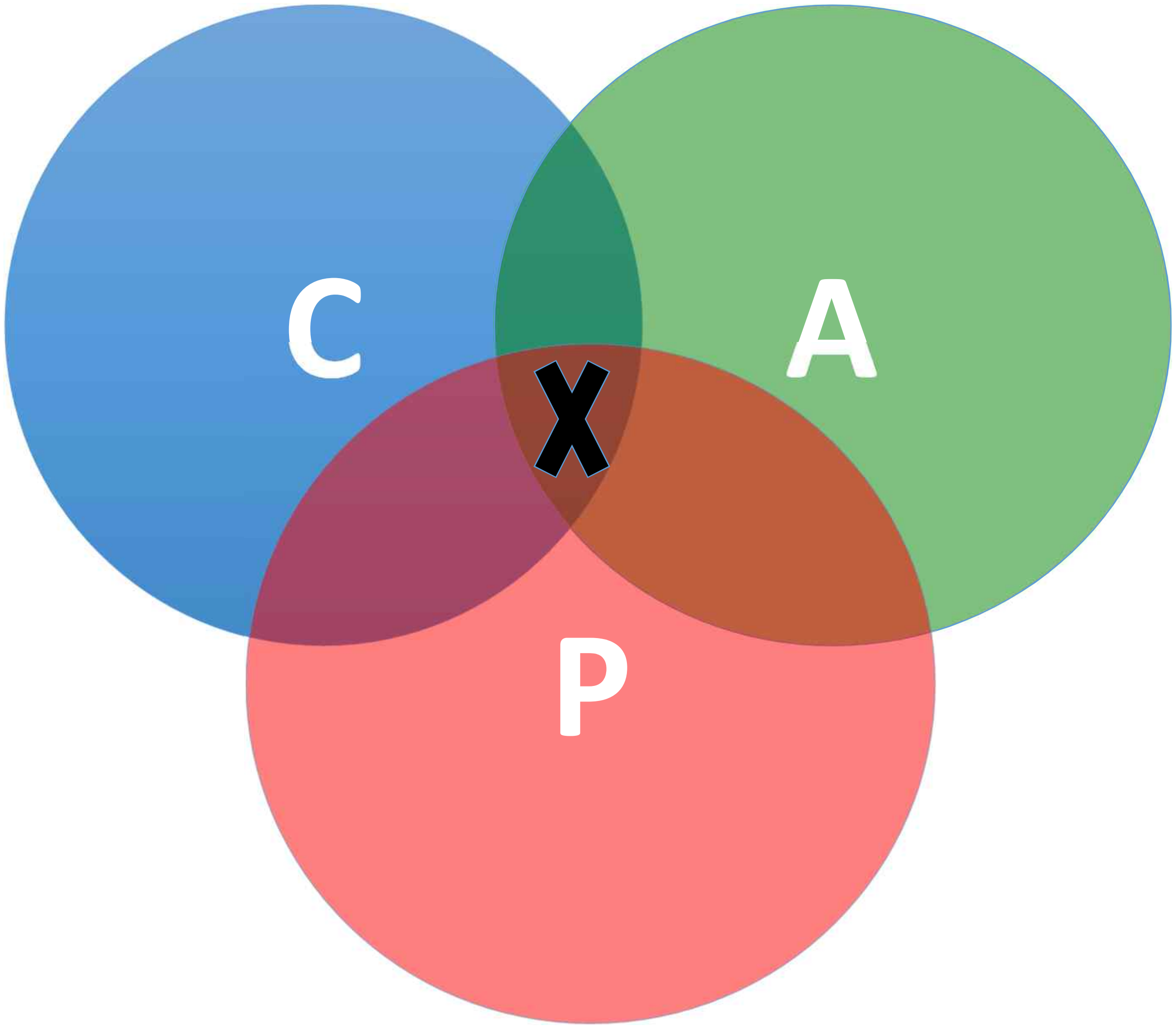
- It is impossible for a web service to provide following *three guarantees at the same time*:
  - **Consistency**
  - **Availability**
  - **Partition-tolerance**

- Prof. Eric Brewer (U.C Berkeley and Google)

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**

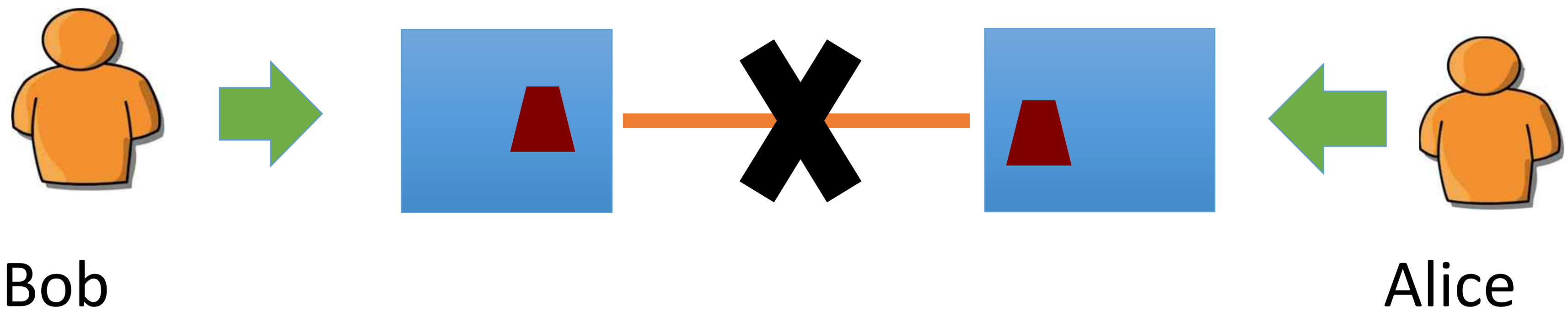
CAP Theorem



CAP Theorem

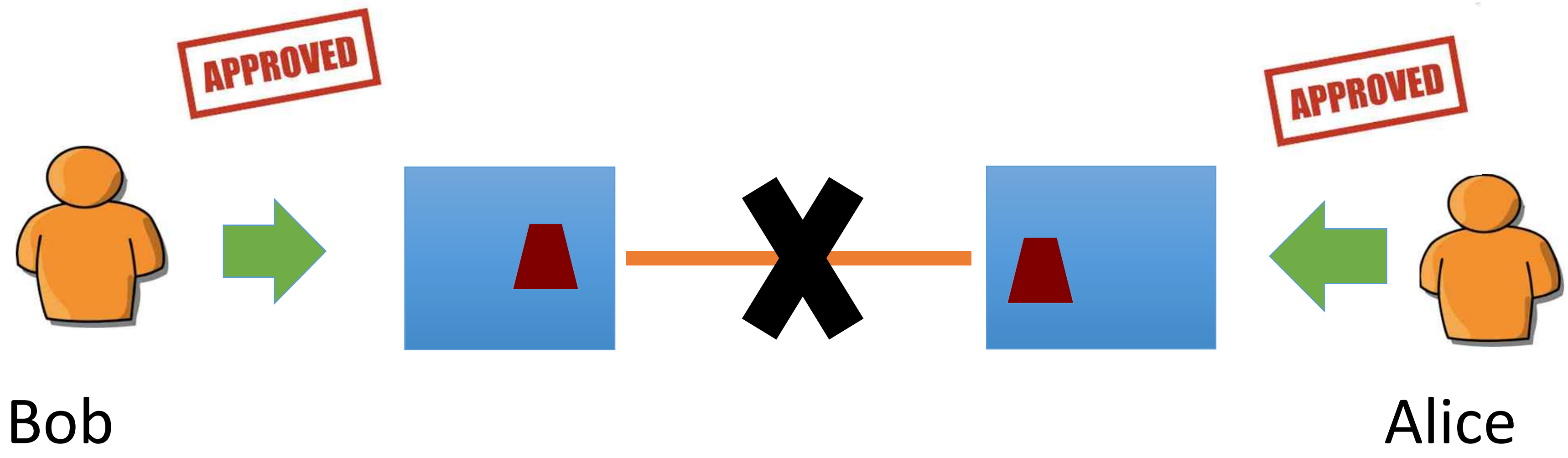
- A simple example:

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



- A simple example:

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



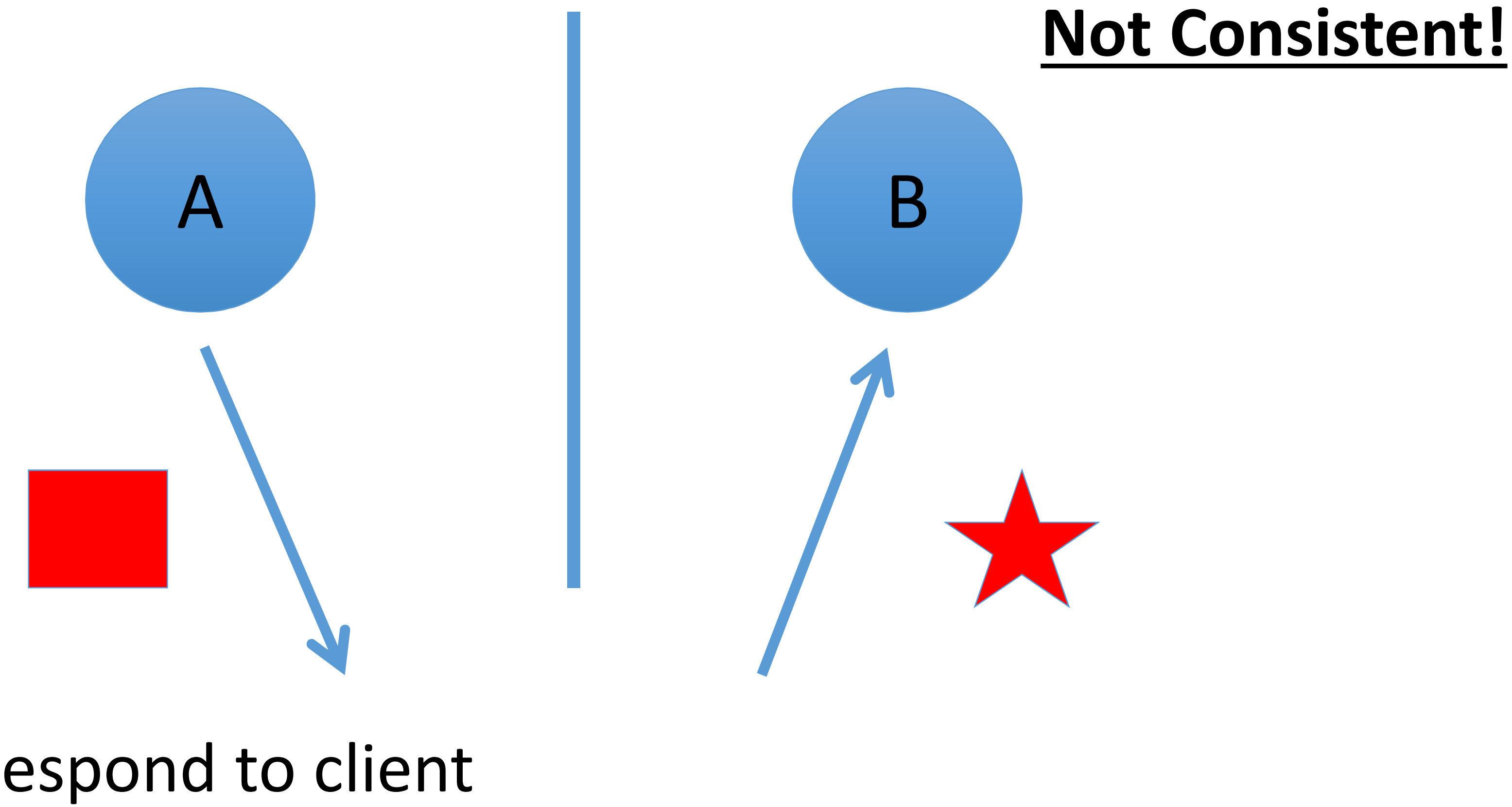
CAP Theorem: Proof

- 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.

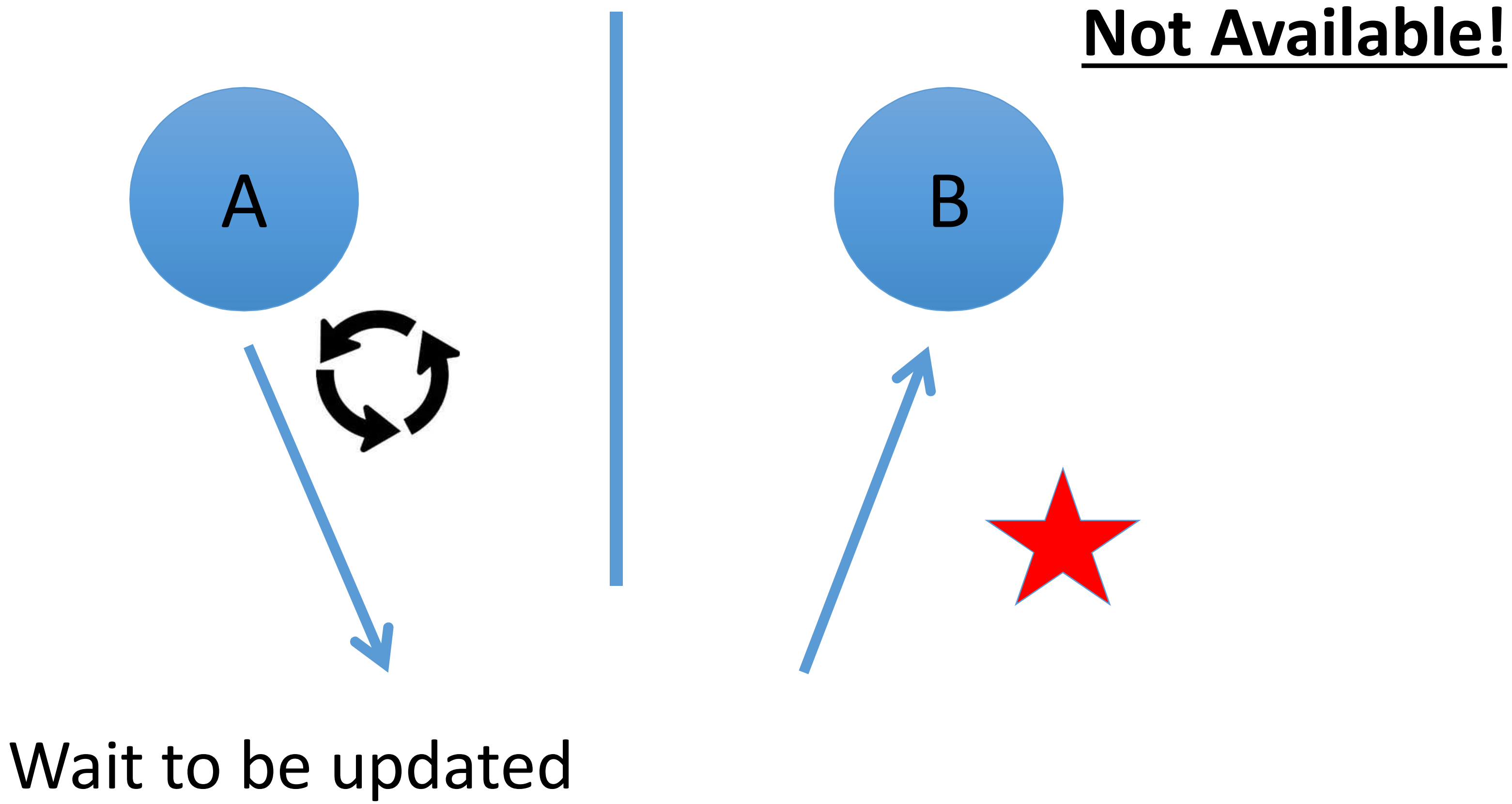
CAP Theorem: Proof

- A simple proof using two nodes:



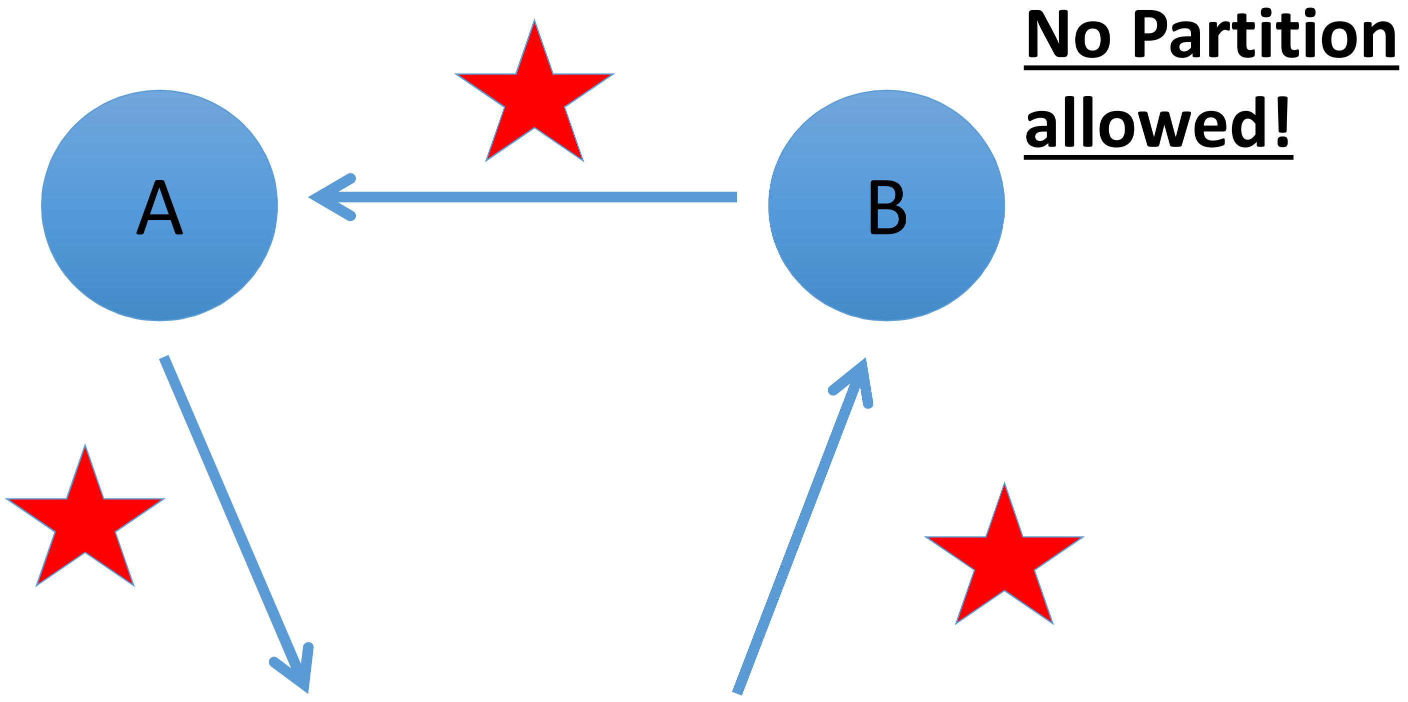
CAP Theorem: Proof

- A simple proof using two nodes:



CAP Theorem: Proof

- A simple proof using two nodes:



A gets updated from B

Why this is important?

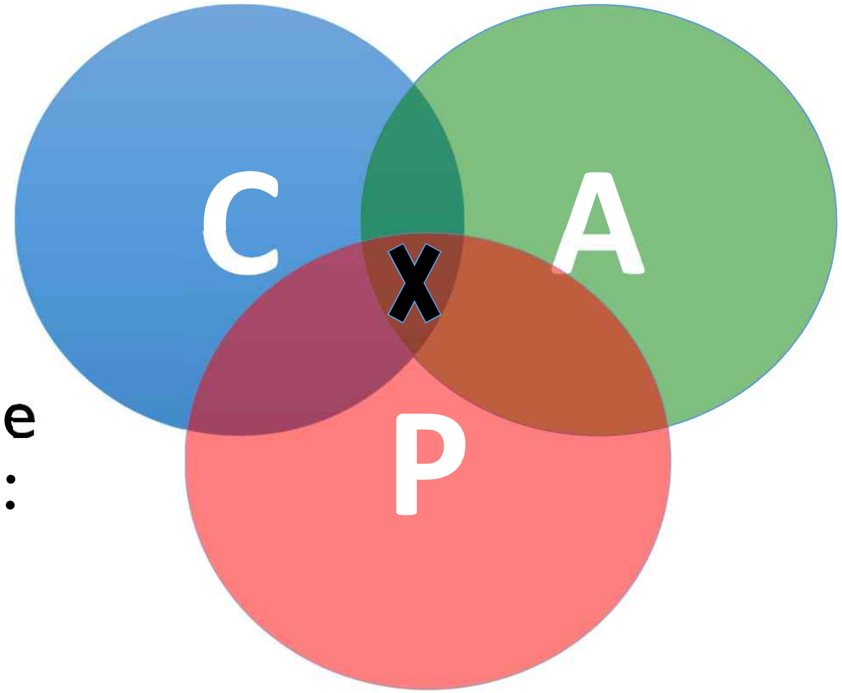
- The future of databases is **distributed** (Big Data Trend, etc.)
- CAP theorem describes the **trade-offs** involved in distributed systems
- A proper understanding of CAP theorem is essential to **making decisions** about the future of distributed database **design**
- Misunderstanding can lead to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

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



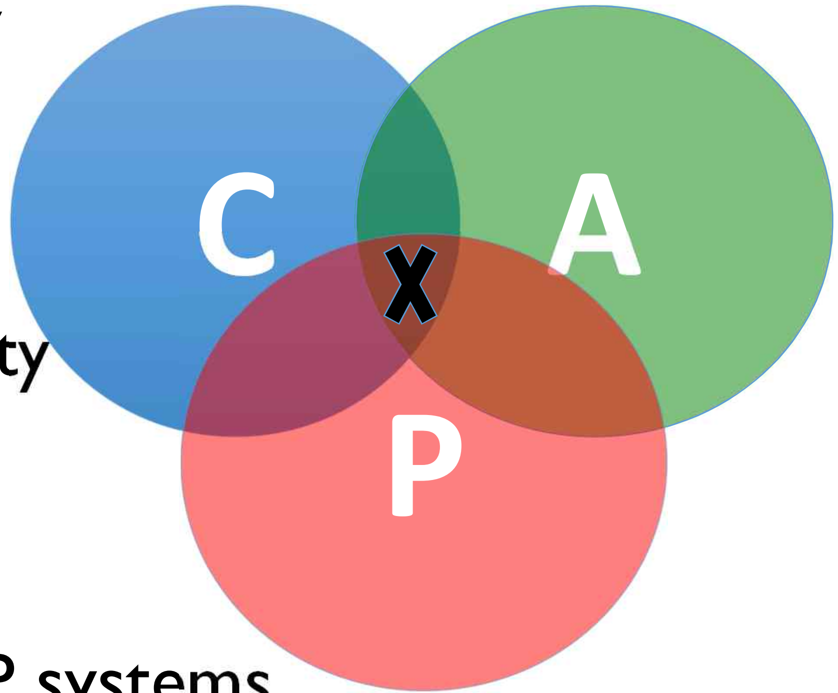
CAP Theorem 12 year later

- Prof. 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

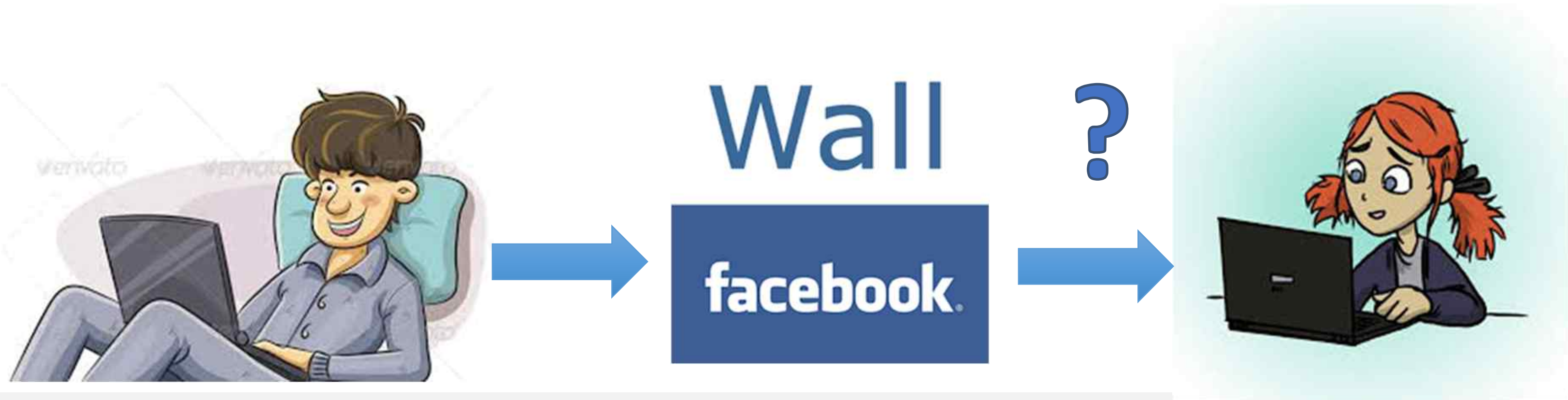
- Example:
  - Majority protocols
    - Paxos, Raft
  - Distributed Locking
    - Google's Chubby: Coarse-grained lock service for loosely-coupled systems
- Feature:
  - Pessimistic locking
  - Make minority partition unavailable
    - E.g.) elect a new leader if the leader is not reachable from majority

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

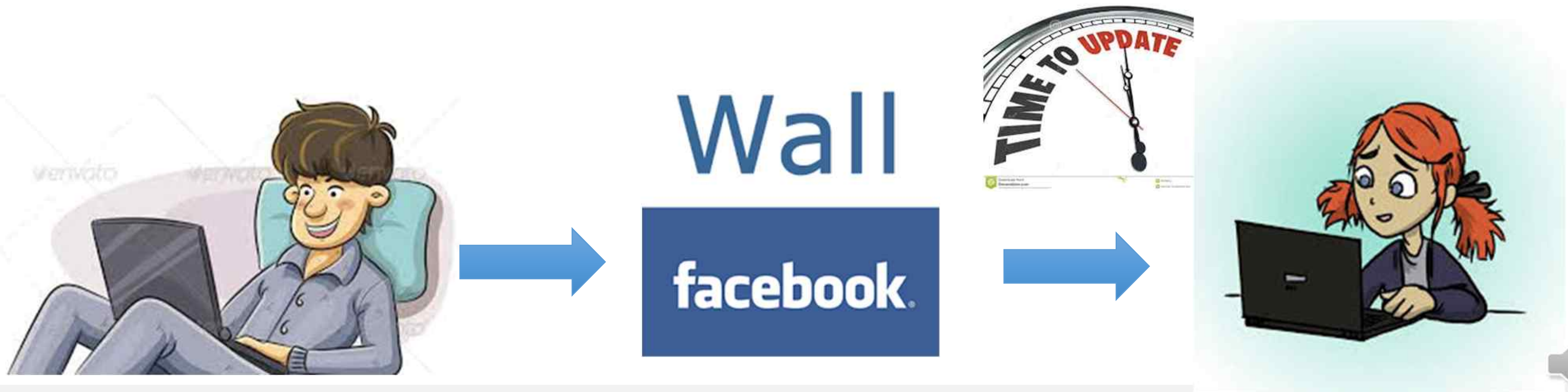
- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds nothing is there.



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, Gmail, and many web services use an **eventual consistent model**
- Why Facebook chooses eventual consistency 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 consistency 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

- 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

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 Theorem

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

PACELC