РАЗВОЈ СОФТВЕРА 2

САР Теорема

КОНЗИСТЕНТНОСТ, ДОСТУПНОСТ И ТОЛЕРАНЦИЈА НА ПАРТИЦИОНИСАЊЕ

CAP THEOREM

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

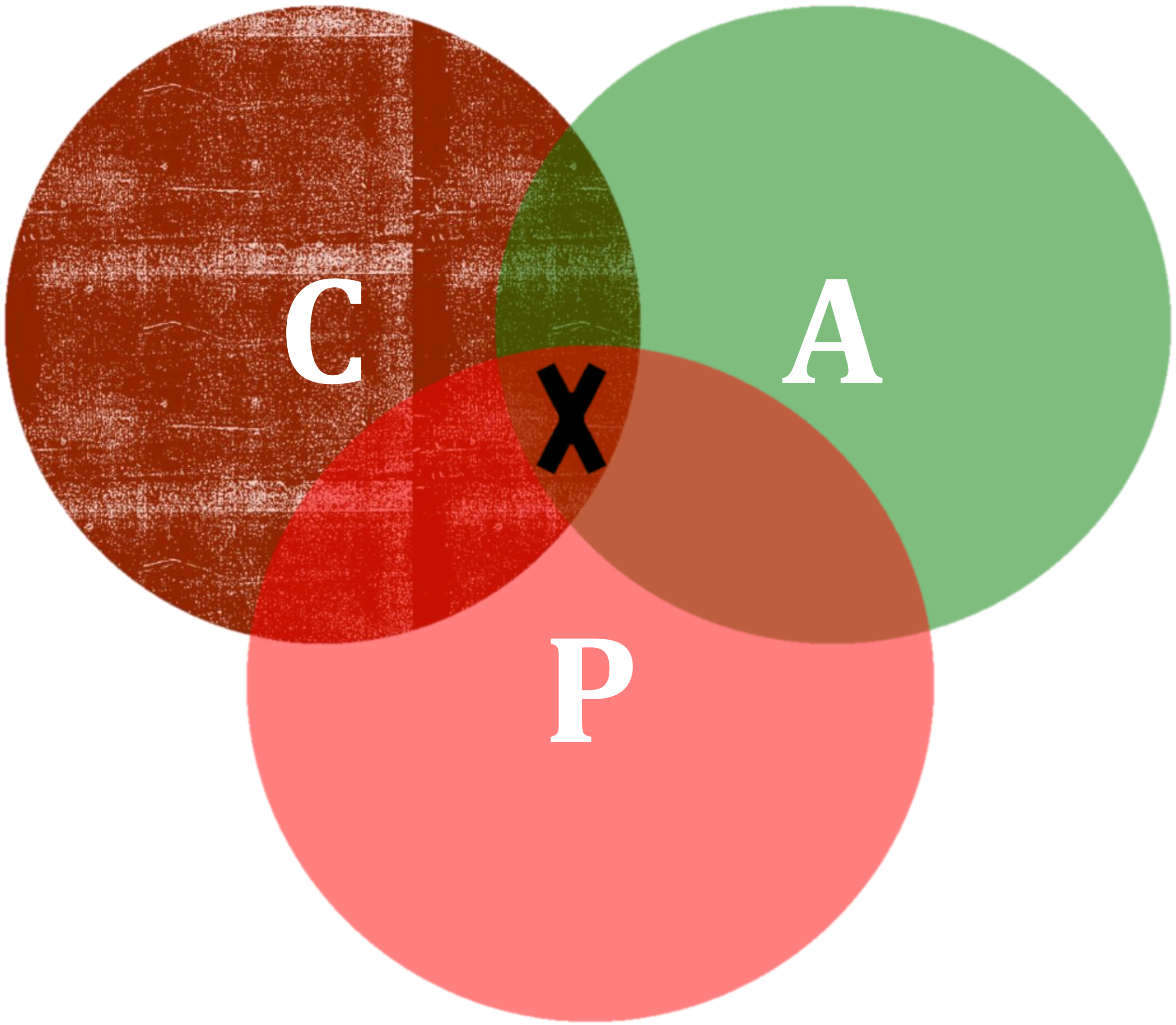
- Described the *trade-offs involved in distributed system*

- 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**

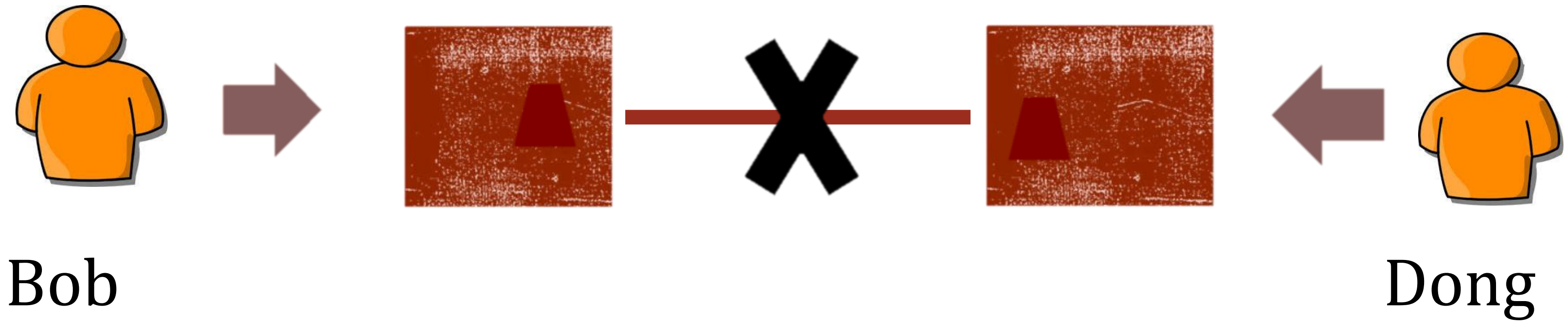
CAP THEOREM



CAP THEOREM

- A simple example:

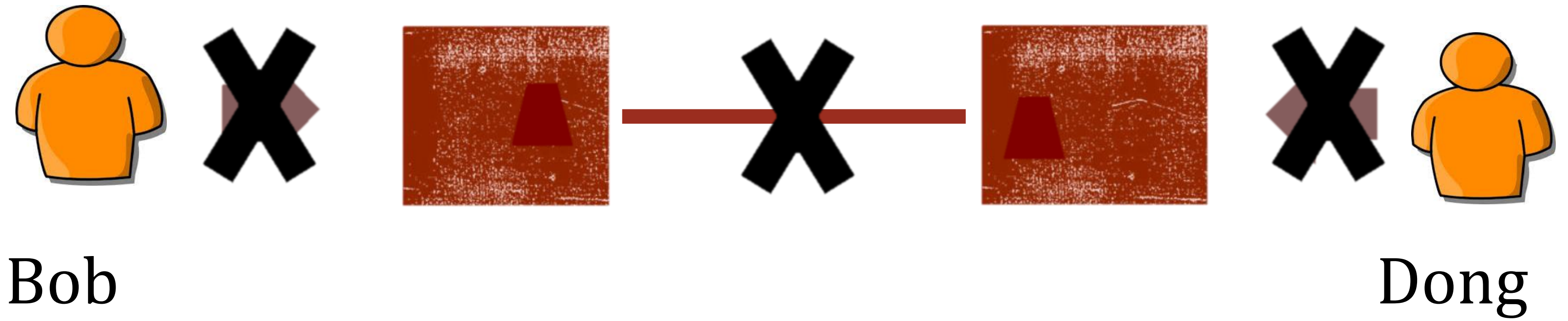
Hotel Booking: are we double-booking the same room?



CAP THEOREM

- A simple example:

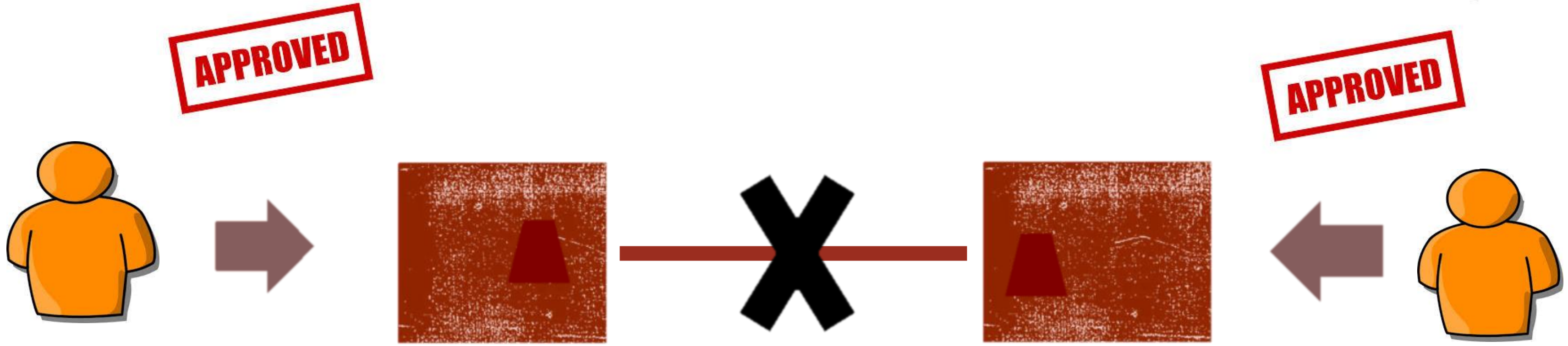
Hotel Booking: are we double-booking the same room?



CAP THEOREM

- A simple example:

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



Bob

Dong

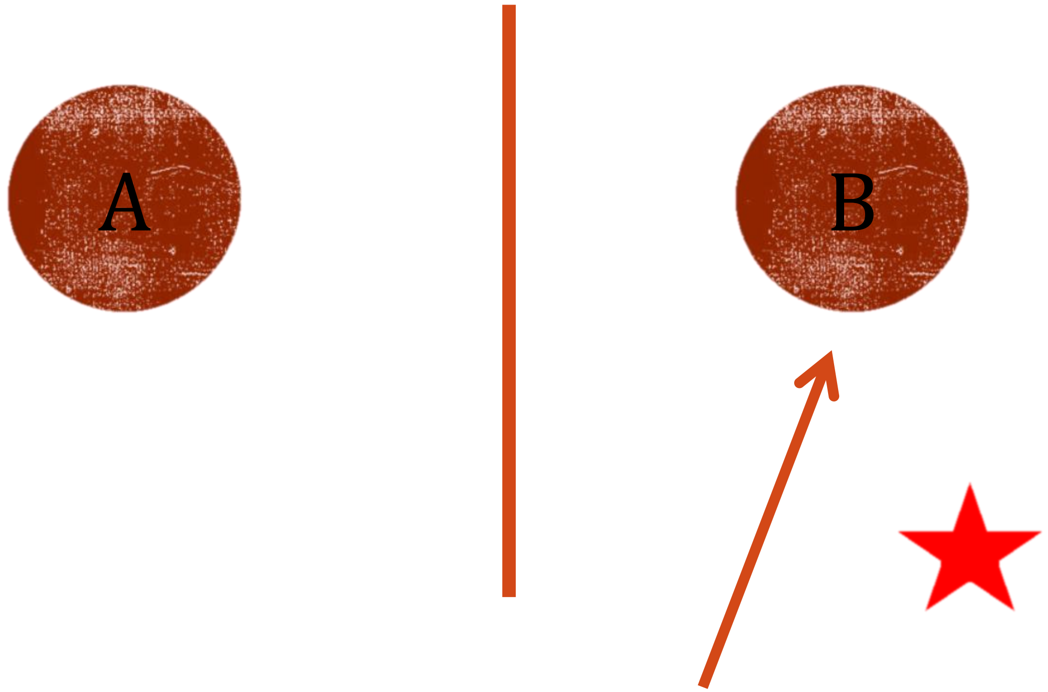
CAP THEOREM: PROOF

- 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.

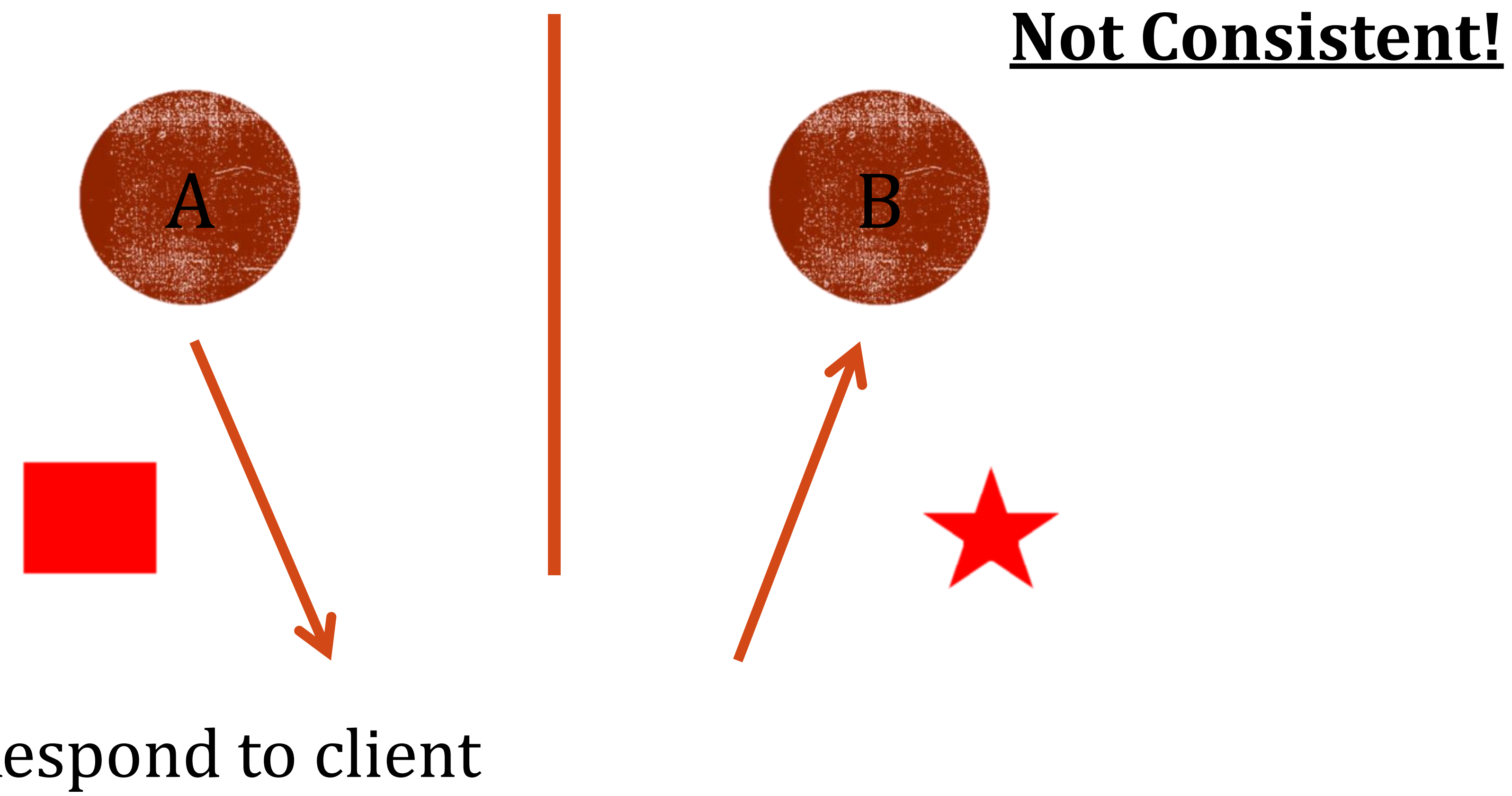
CAP THEOREM: PROOF

- A simple proof using two nodes:



CAP THEOREM: PROOF

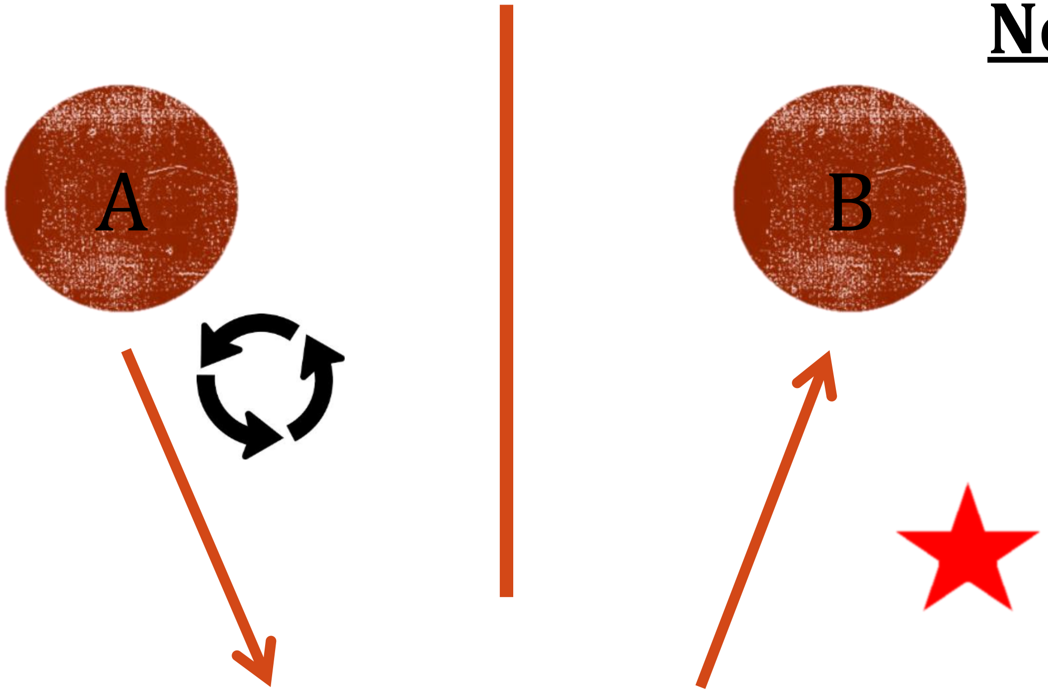
- A simple proof using two nodes:



CAP THEOREM: PROOF

- A simple proof using two nodes:

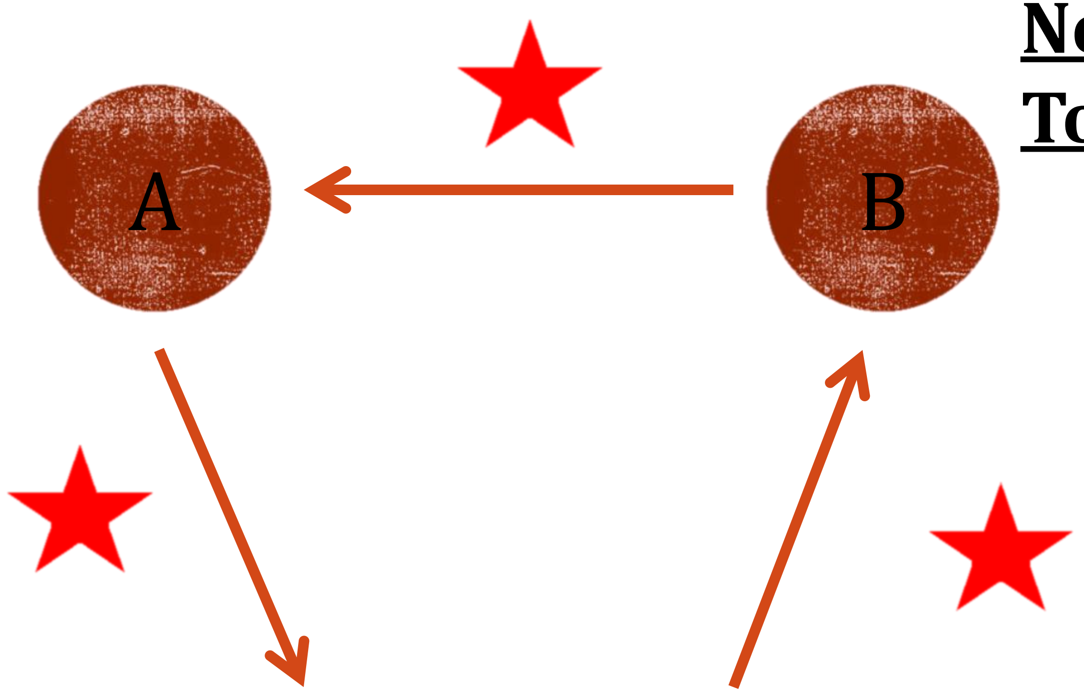
<u>Not Available!</u>



Wait to be updated

CAP THEOREM: PROOF

- A simple proof using two nodes:



<u>Not Partition Tolerant!</u>

A gets updated from B

ПОСЛЕДИЦЕ CAP ТЕОРЕМЕ

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

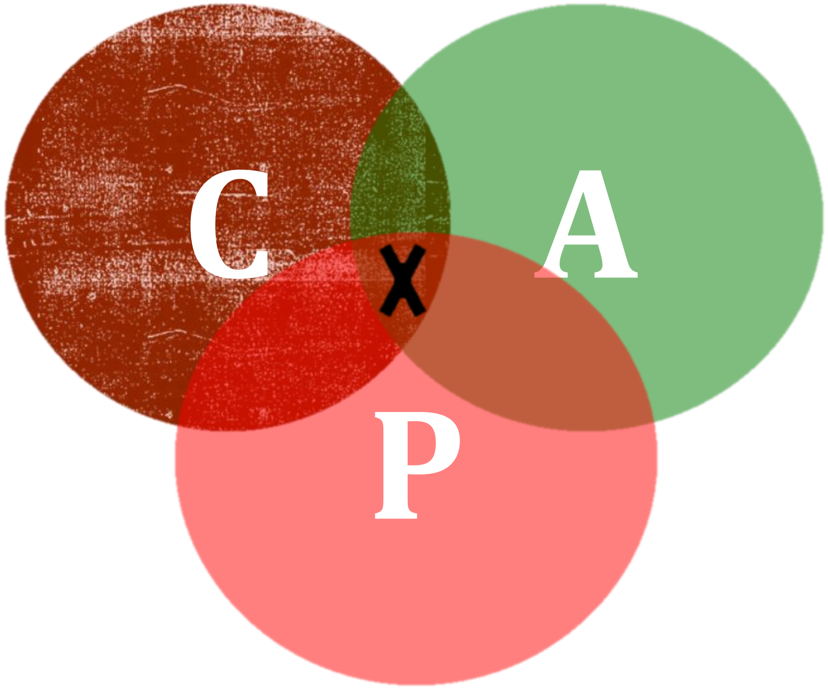
- The Relational Database is built on the principle of **ACID** (Atomicity, Consistency, Isolation, Durability)
- It implies that a truly distributed relational database should have **availability, consistency and partition tolerance**.
- Which unfortunately is **impossible** …

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



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/

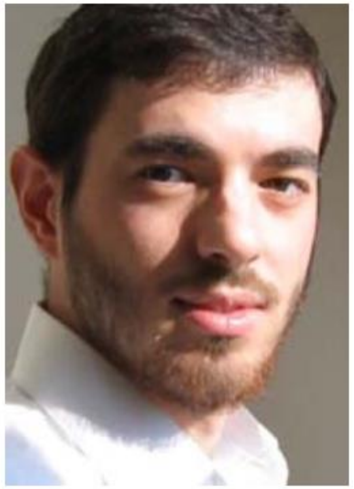
A FEW WITNESSES

- 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

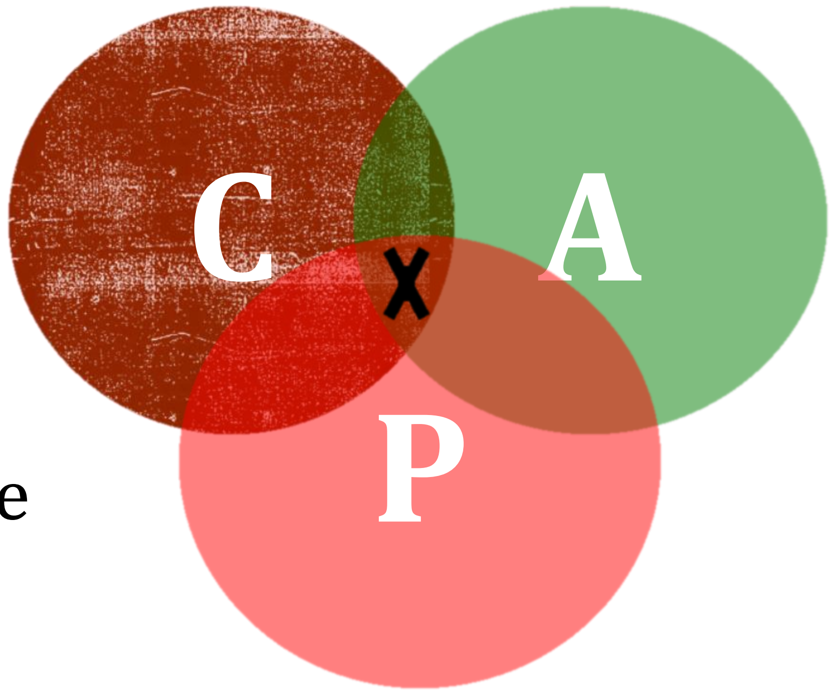
CAP THEOREM 12 YEAR LATER

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

ВРСТЕ КОНЗИСТЕНЦИЈЕ

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

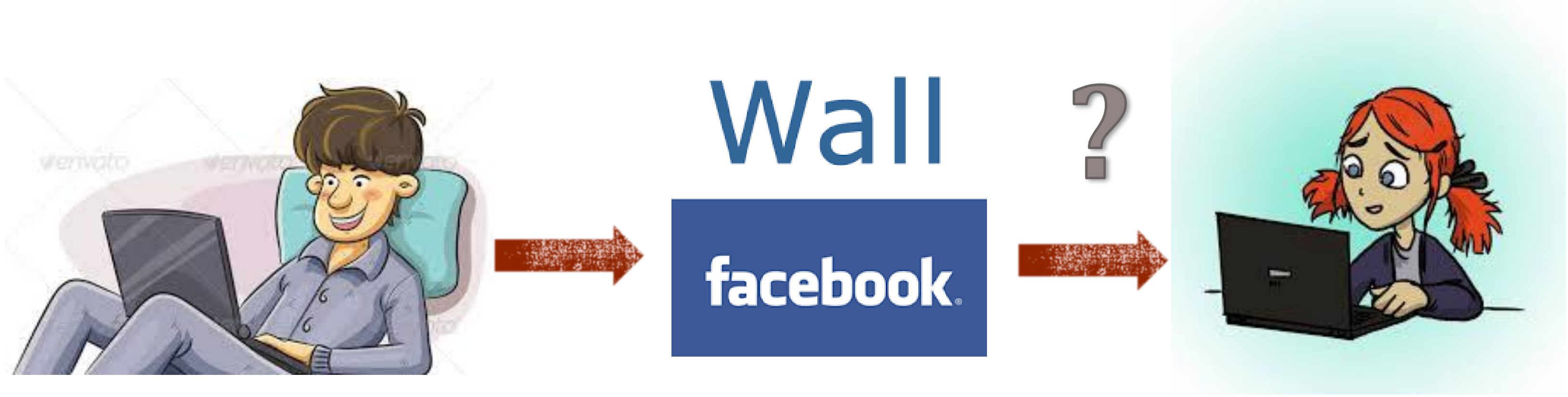
EVENTUAL CONSISTENCY VARIATIONS

- 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 (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

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

ПРИМЕРИ ПАРТИЦИОНИСАЊА

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

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

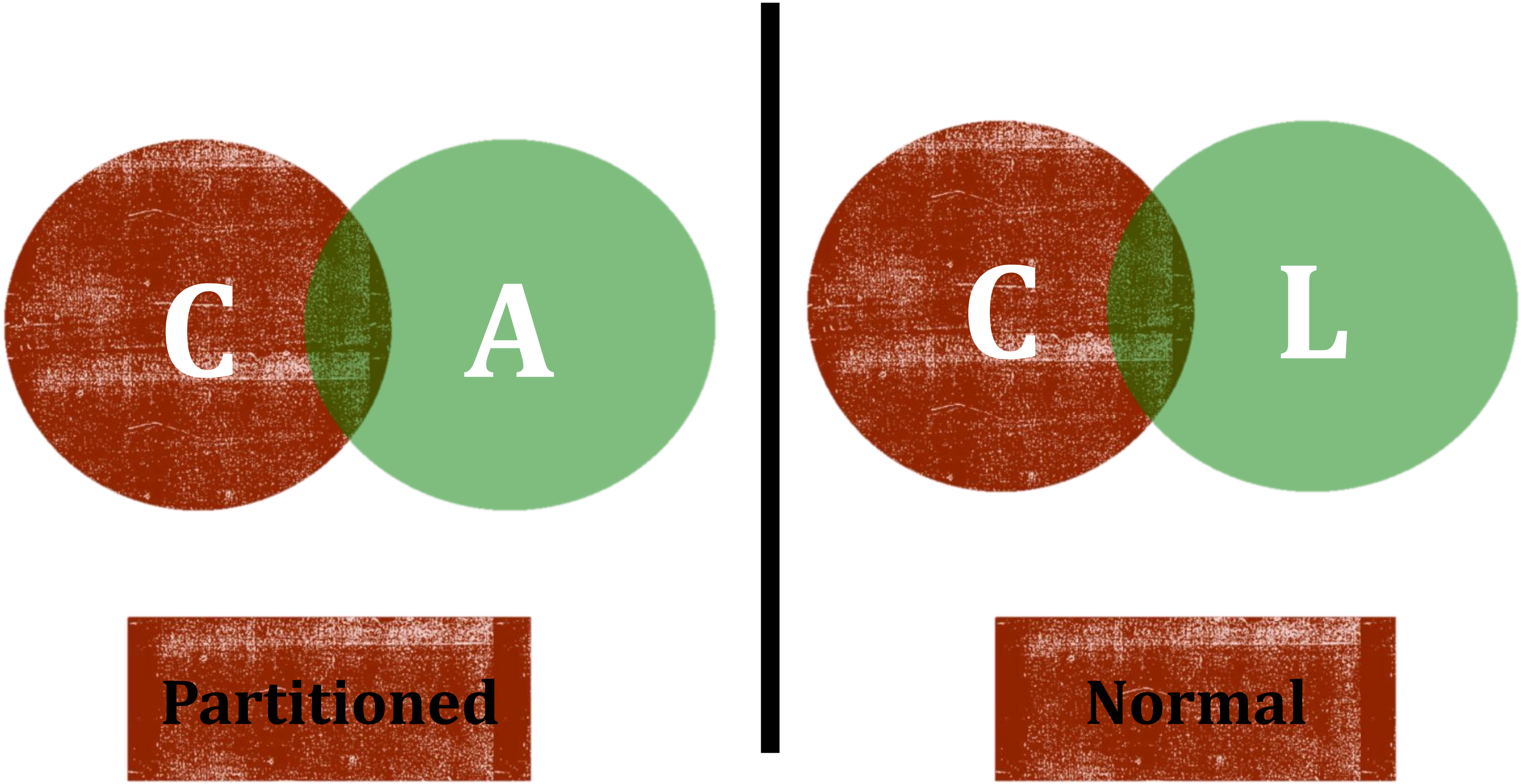
РАСЕЛС ТЕОРЕМА

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.

PACELC



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

НАПОМЕНА

Највећи део материјала ове презентације је преузет из презентације
CAP Theorem, аутора Dong Wang, која је доступна на адреси:
https://www3.nd.edu/~dthain/courses/cse40822/fall2014/slides/cse40822-CAP.pptx