

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**



C and Latency Tradeoff

- Amazon claims that just an extra one tenth of a second on their response times will cost them 1% in sales.
- Google said they noticed that just a half a second increase in latency caused traffic to drop by a fifth.

CAP Theorem

- Consistency
- Availability
- Partition Tolerance
- Choose two

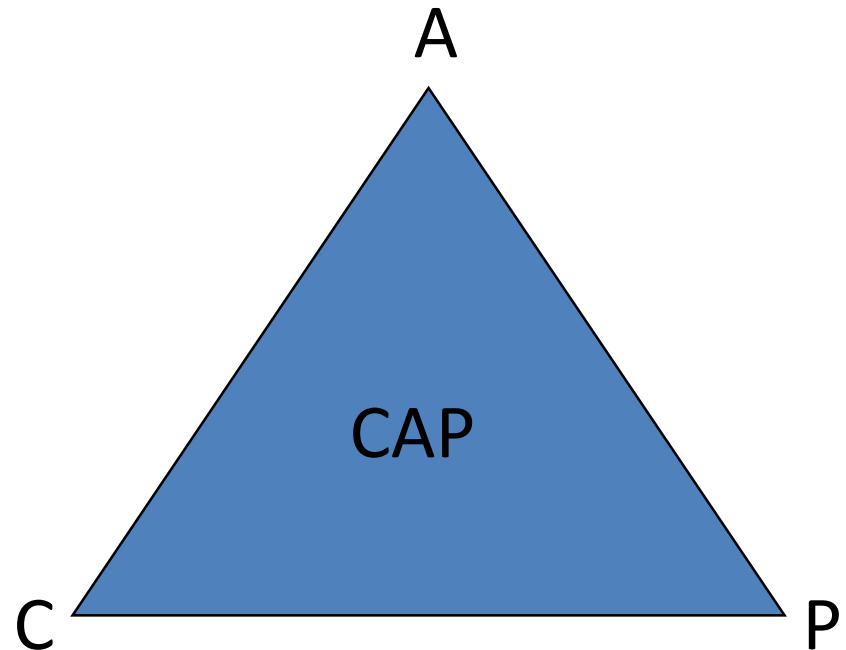
[CAP Twelve Years Later: How the Rules Have Changed \(Eric Brewer\)](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6133253)

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6133253>

[The CAP Theorem's Growing Impact](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6155651)

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6155651>

(Simon Shim)



The CAP Theorem

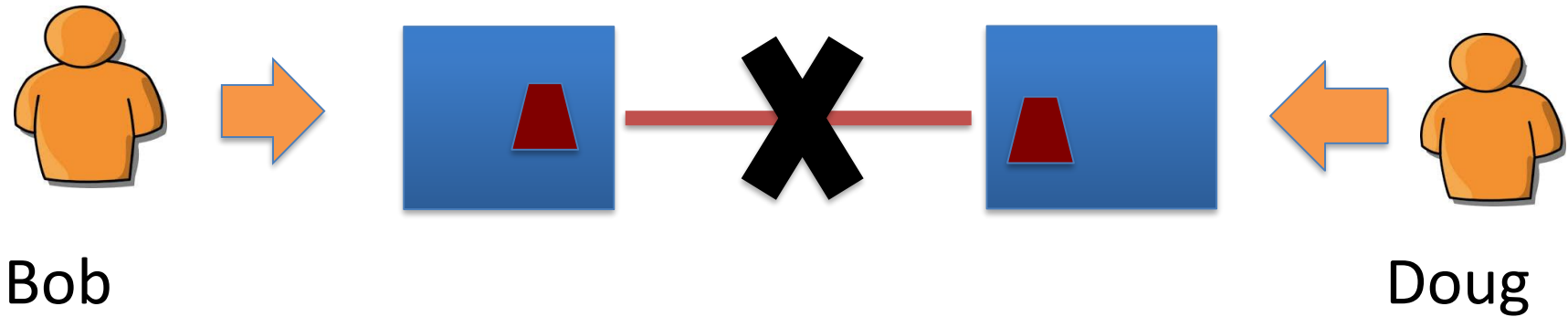
- The limitations of distributed databases can be described in the so called the **CAP theorem**
 - **Consistency**: every node always sees the same data at any given instance (i.e., strict consistency)
 - **Availability**: the system continues to operate, even if nodes in a cluster crash, or some hardware or software parts are down due to upgrades
 - **Partition Tolerance**: the system continues to operate in the presence of network partitions

CAP theorem: any distributed database with shared data, can have at most two of the three desirable properties, C, A or P

CAP Theorem

- A simple example:

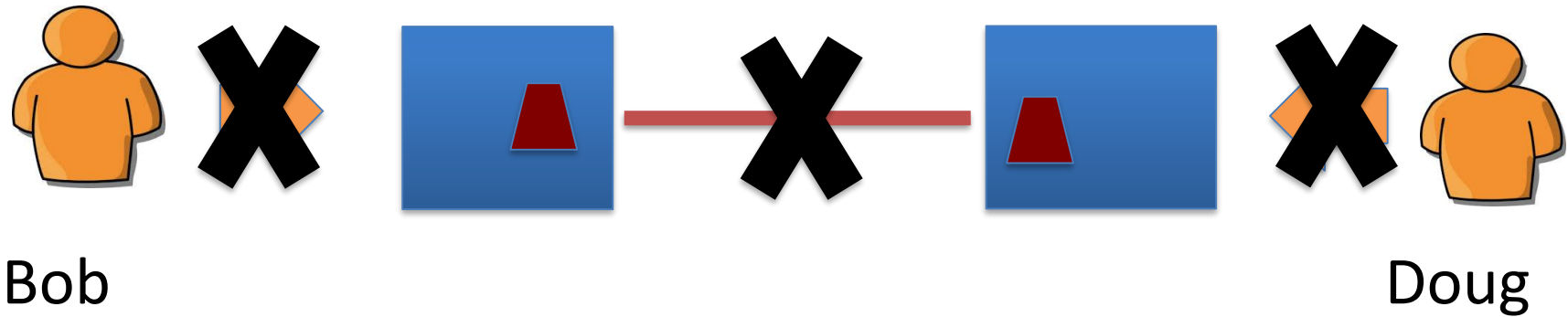
Hotel Booking: are we double-booking the same room?



CAP Theorem

- A simple example:

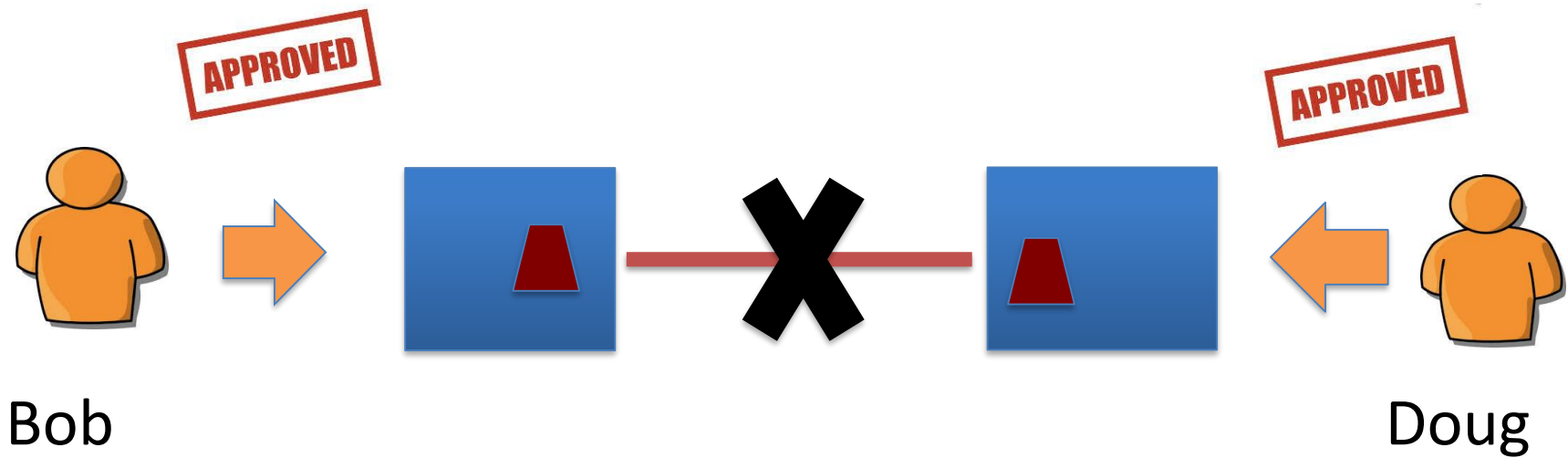
Hotel Booking: are we double-booking the same room?



CAP Theorem

- A simple example:

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



CAP Theorem: Proof

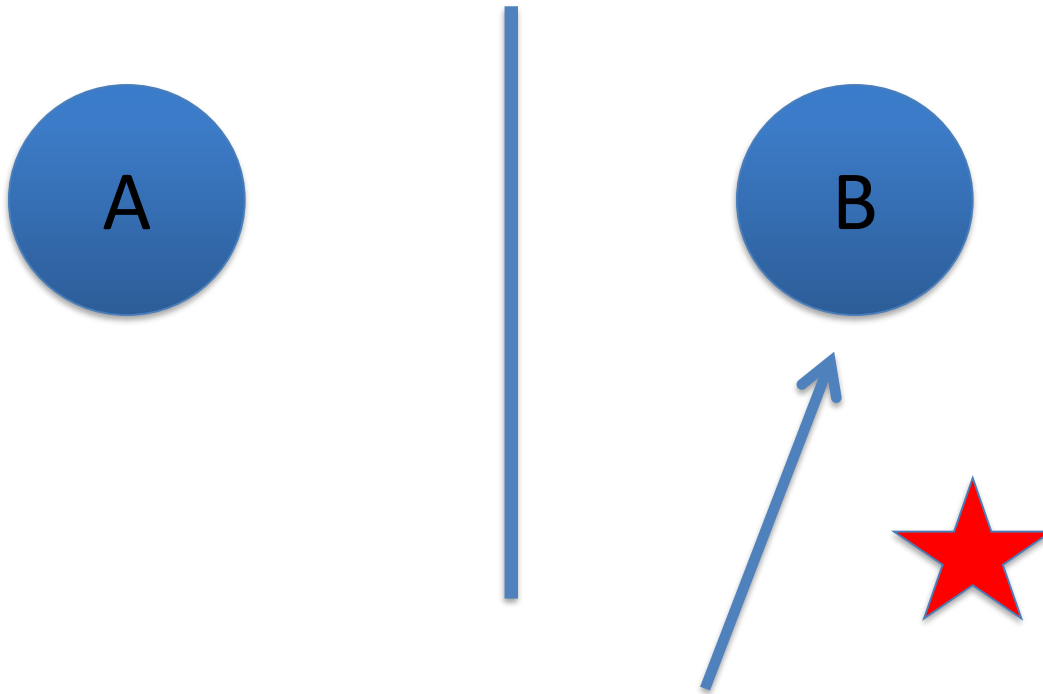
- 2002: Proven by research conducted by Nancy Lynch and Seth Gilbert at MIT

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.



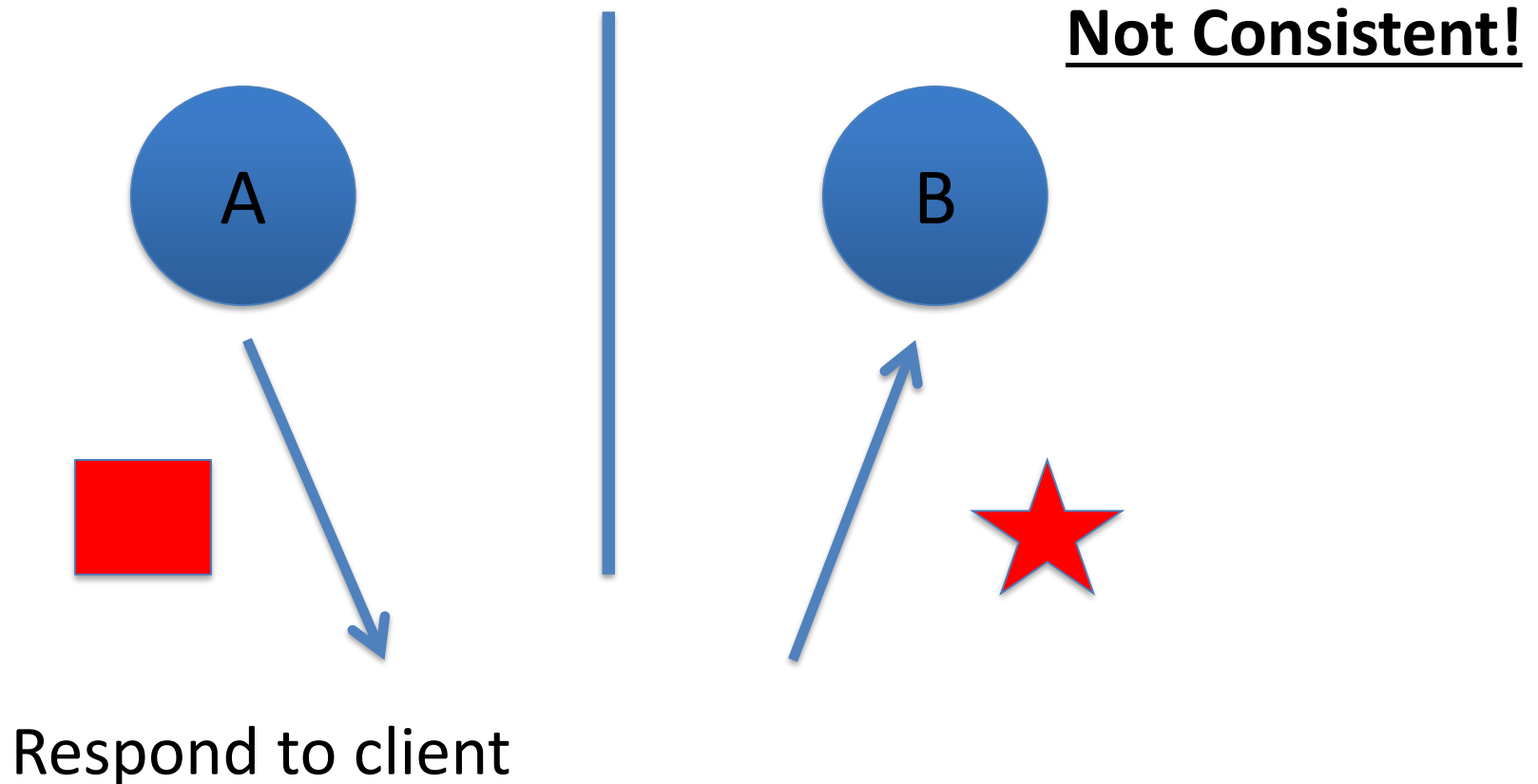
CAP Theorem: Proof

- A simple proof using two nodes:



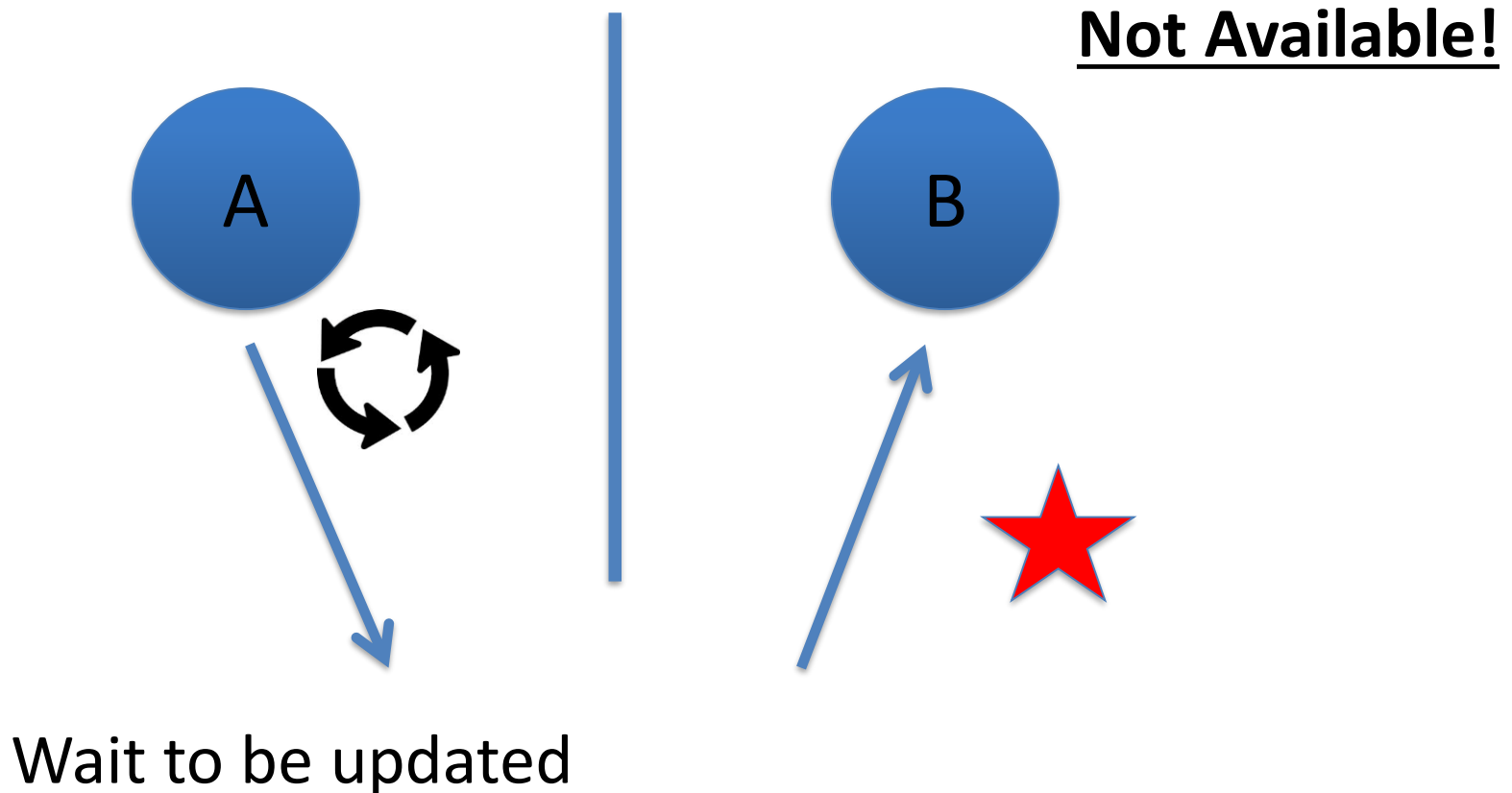
CAP Theorem: Proof

- A simple proof using two nodes:



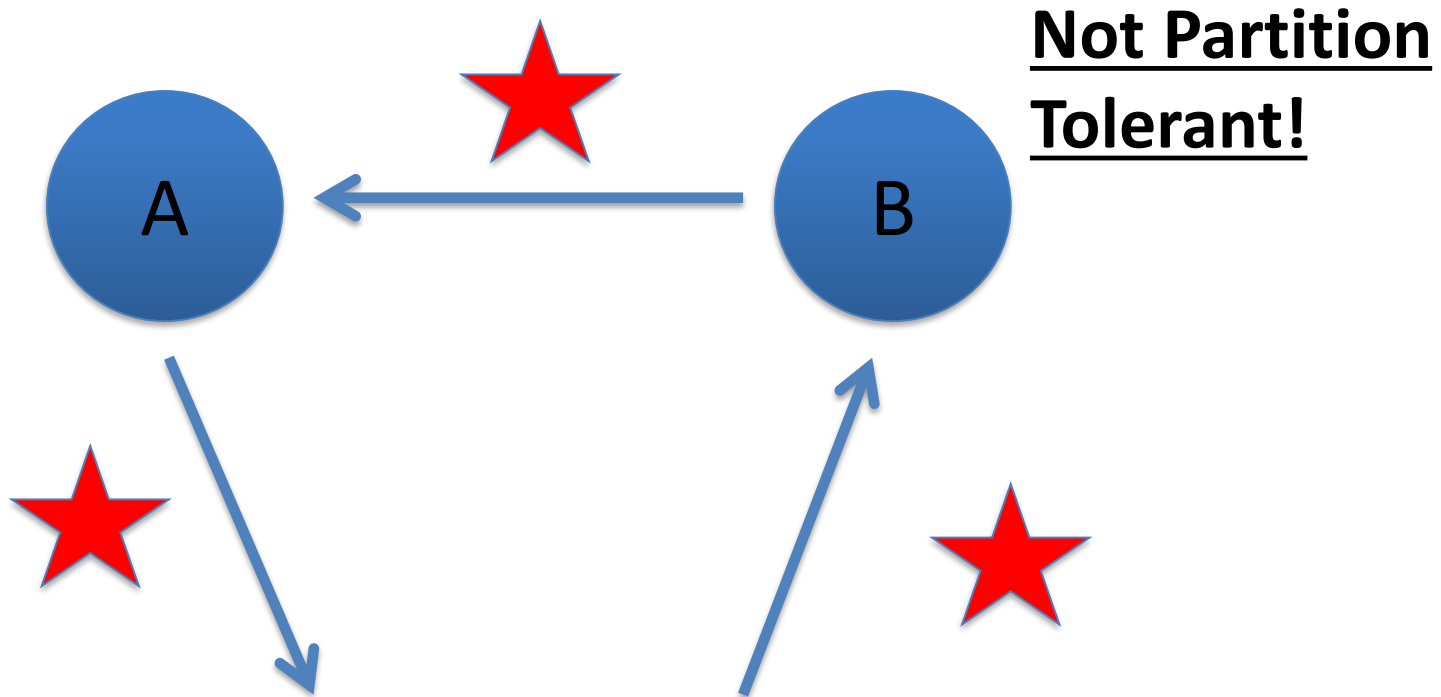
CAP Theorem: Proof

- A simple proof using two nodes:



CAP Theorem: Proof

- A simple proof using two nodes:



A gets updated from B

Questions?

- Which one would you choose when network partition?
(a) C (b) A
- Which of CAP is essential for a distributed system?
(a) C (b) A (c) P (d) none of the above

(see more details p.21)

CAP

- Dynamo does not guarantee C by default
- The event of P forces systems to decide on reducing C or A
- What is the probability of P?
 - Local network
 - Wide area network

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**

Why this is important?

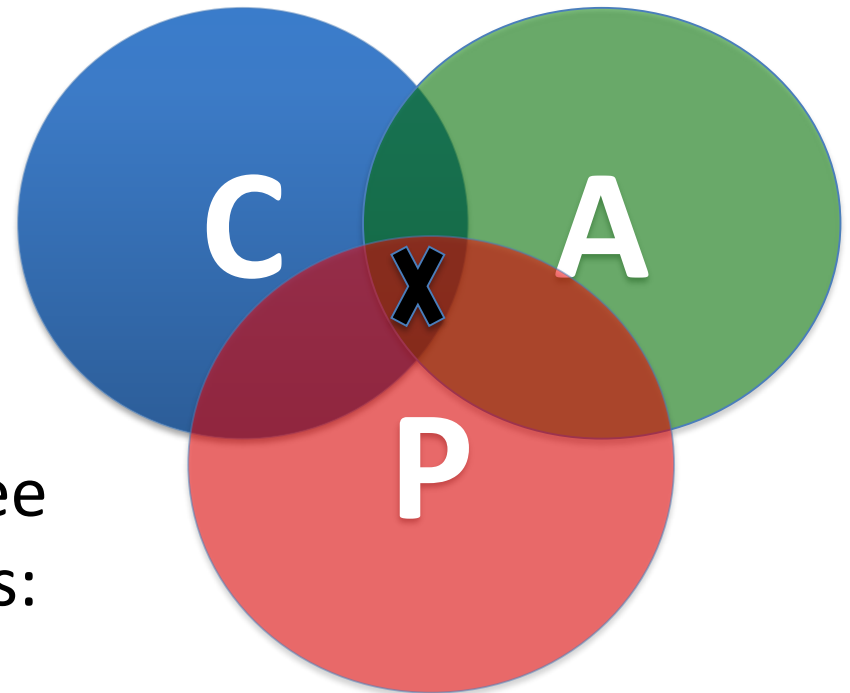
- The future of databases is **distributed** (Big Data Trend, etc.)
- CAP theorem describes the **trade-offs** involved in distributed systems
- A proper understanding of CAP theorem is essential to **making decisions** about the future of distributed database **design**
- Misunderstanding can lead to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

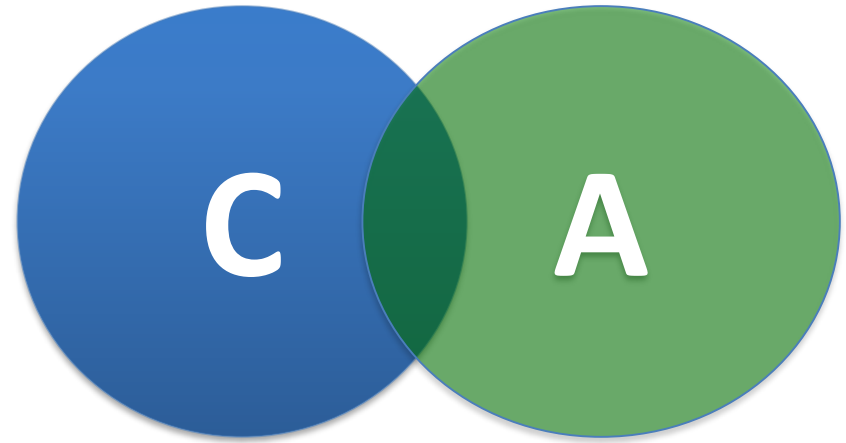
Revisit CAP Theorem

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



A popular misconception: 2 out of 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.**”



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?**



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.”



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?



www.bigstock.com - 30744092



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

Discussion

- In an e-commercial system (e.g., Amazon, e-Bay, 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.)

Partitioning Examples

- Data Partitioning
- Operational Partitioning
- Functional 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

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?

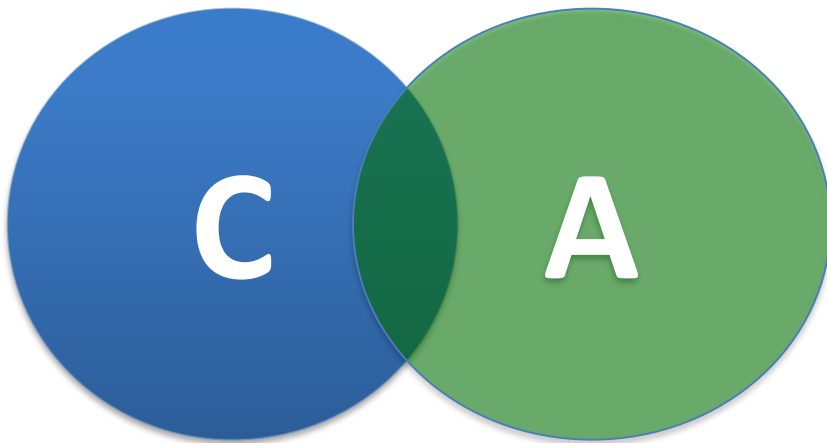
- Tradeoff between **Consistency** and **Latency**:
- Caused by the **possibility of failure** in distributed systems
 - High availability -> replicate data -> consistency problem
- Basic idea:
 - Availability and latency are arguably **the same thing**: unavailable -> extreme high latency
 - Achieving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

CAP -> PACELC

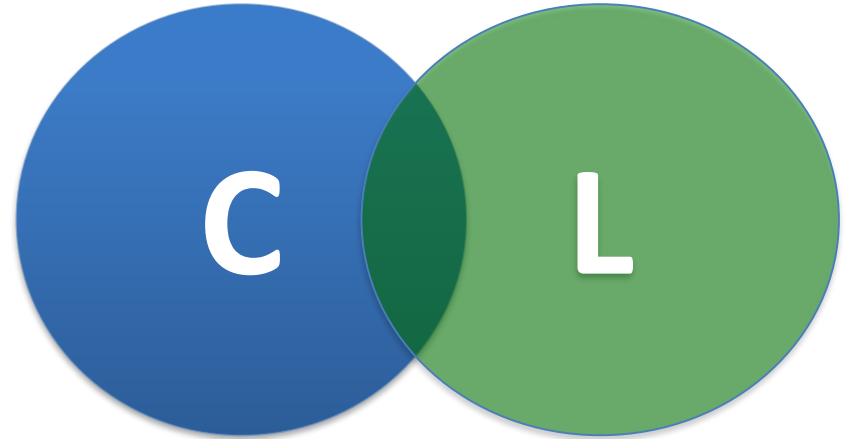
- A more complete description of the space of potential tradeoffs for distributed system:
 - If there is a **partition (P)**, how does the system trade off **availability and consistency (A and C)**; **else (E)**, when the system is running normally in the absence of partitions, how does the system trade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



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