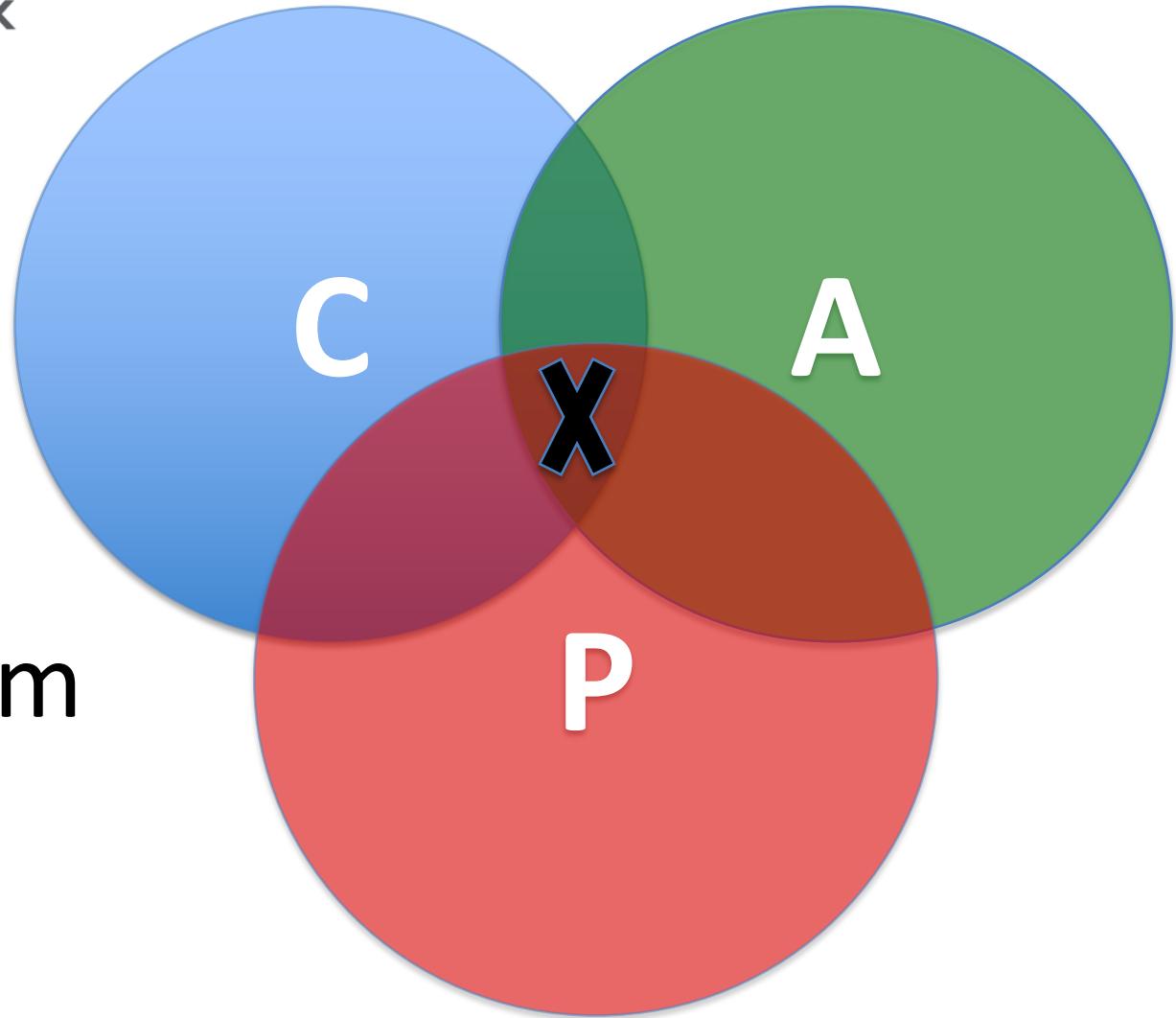


CAP Theorem



Thanks to Arvind K., Dong W., and Mihir N. for slides.

CAP Theorem

- “It is impossible for a web service to provide these *three guarantees at the same time (pick 2 of 3):*
 - **(Sequential) Consistency**
 - **Availability**
 - **Partition-tolerance”**
- Conjectured by Eric Brewer in ’00
 - Proved by Gilbert and Lynch in ’02
 - But with definitions that do not match what you’d assume (or Brewer meant)
- Influenced the NoSQL mania
- Highly controversial: “*the CAP theorem encourages engineers to make awful decisions.*” – Stonebraker
 - Many misinterpretations

CAP Theorem

- Consistency:
 - Sequential consistency (a data item behaves as if there is one copy)
- Availability:
 - Node failures do not prevent survivors from continuing to operate
- Partition-tolerance:
 - The system continues to operate despite network partitions
- CAP says that “*A distributed system can satisfy any two of these guarantees at the same time **but not all three***”

C in CAP != C in ACID

- They are different!
- CAP's C(onsistency) = **sequential consistency**
 - Similar to ACID's A(tomicity) = Visibility to all future operations
- ACID's C(onsistency) = Does the data satisfy schema constraints

Sequential consistency

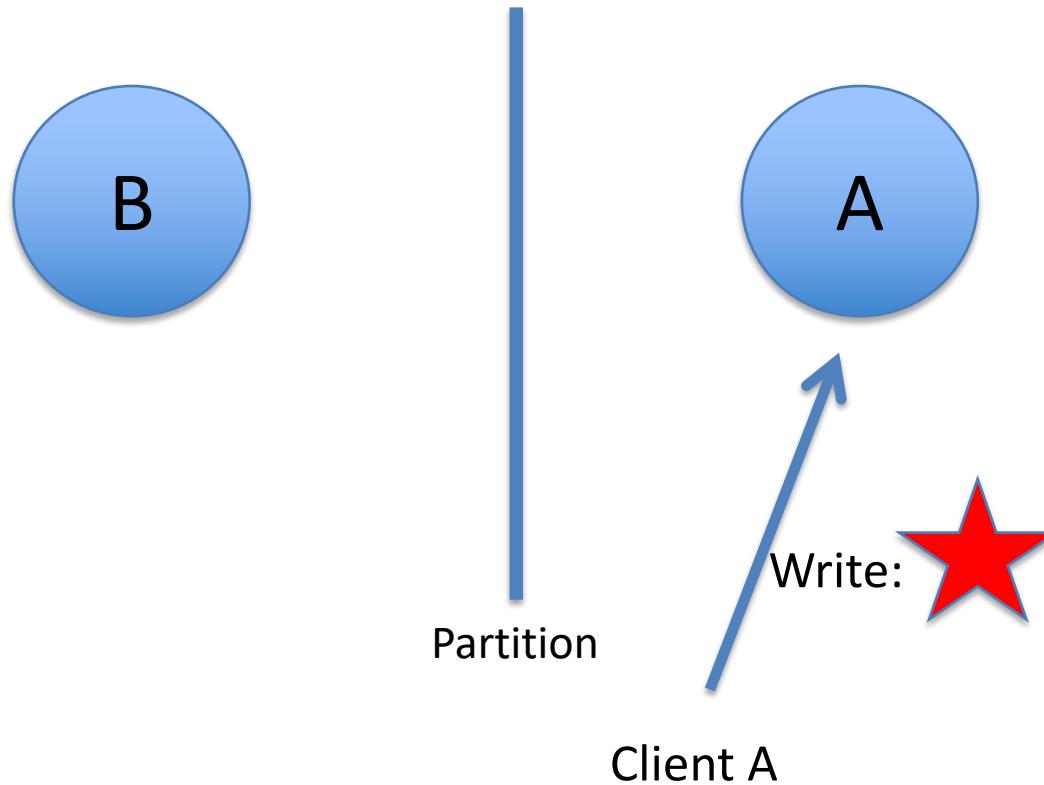
- Makes it appear as if there is one copy of the object
 - Strict ordering on ops from same client
 - A single linear ordering across client ops
 - If client a executes operations {a1, a2, a3, ...}, client b executes operations {b1, b2, b3, ...}
 - Then, globally, clients observe some serialized version of the sequence
 - e.g., {a1, b1, b2, a2, ...} (or whatever)
- Notice how a1 precedes a2, b1 precedes b2, etc

Sequential consistency

- Makes it appear as if there is one copy of the object
- Strict ordering on ops from same client
- A single linear ordering across client ops
 - If client a executes operations {a1, a2, a3, ...}, client b executes operations {b1, b2, b3, ...}
 - Then, globally, clients observe some serialized version of the sequence
 - e.g., {a1, b1, b2, a2, ...} (or whatever)
Notice how a1 precedes a2, b1 precedes b2, etc
- Where have we seen this before?

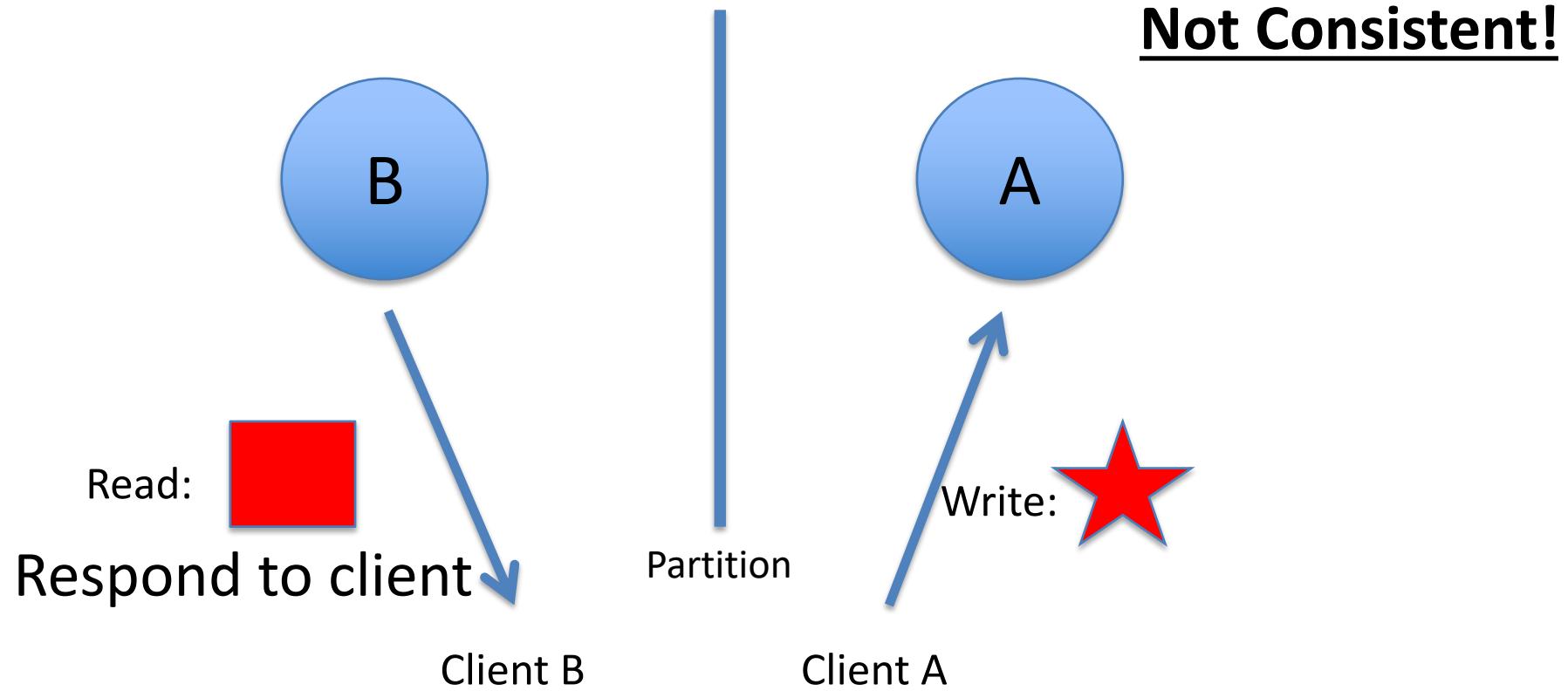
CAP Theorem: Proof

- A simple proof using two nodes:



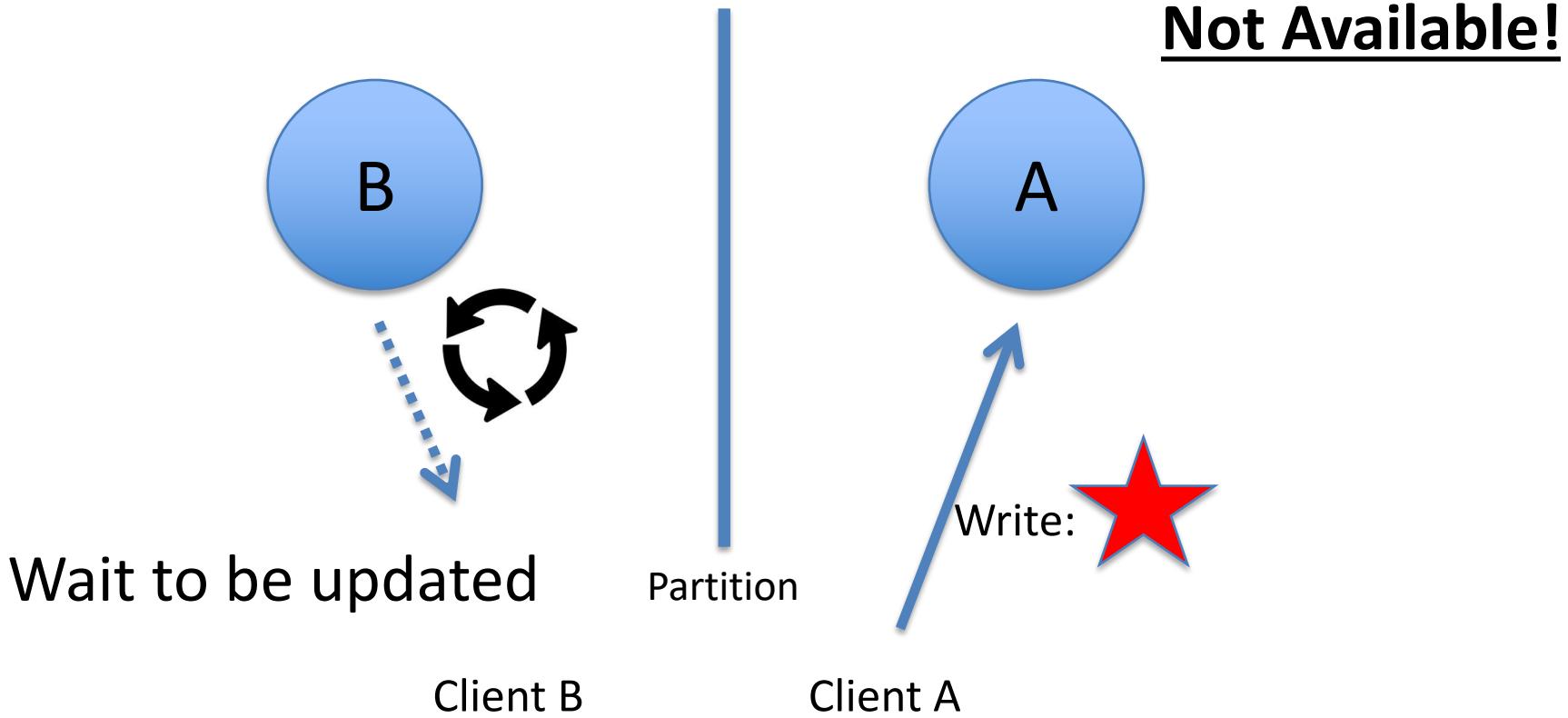
CAP Theorem: Proof

- A simple proof using two nodes:



CAP Theorem: Proof

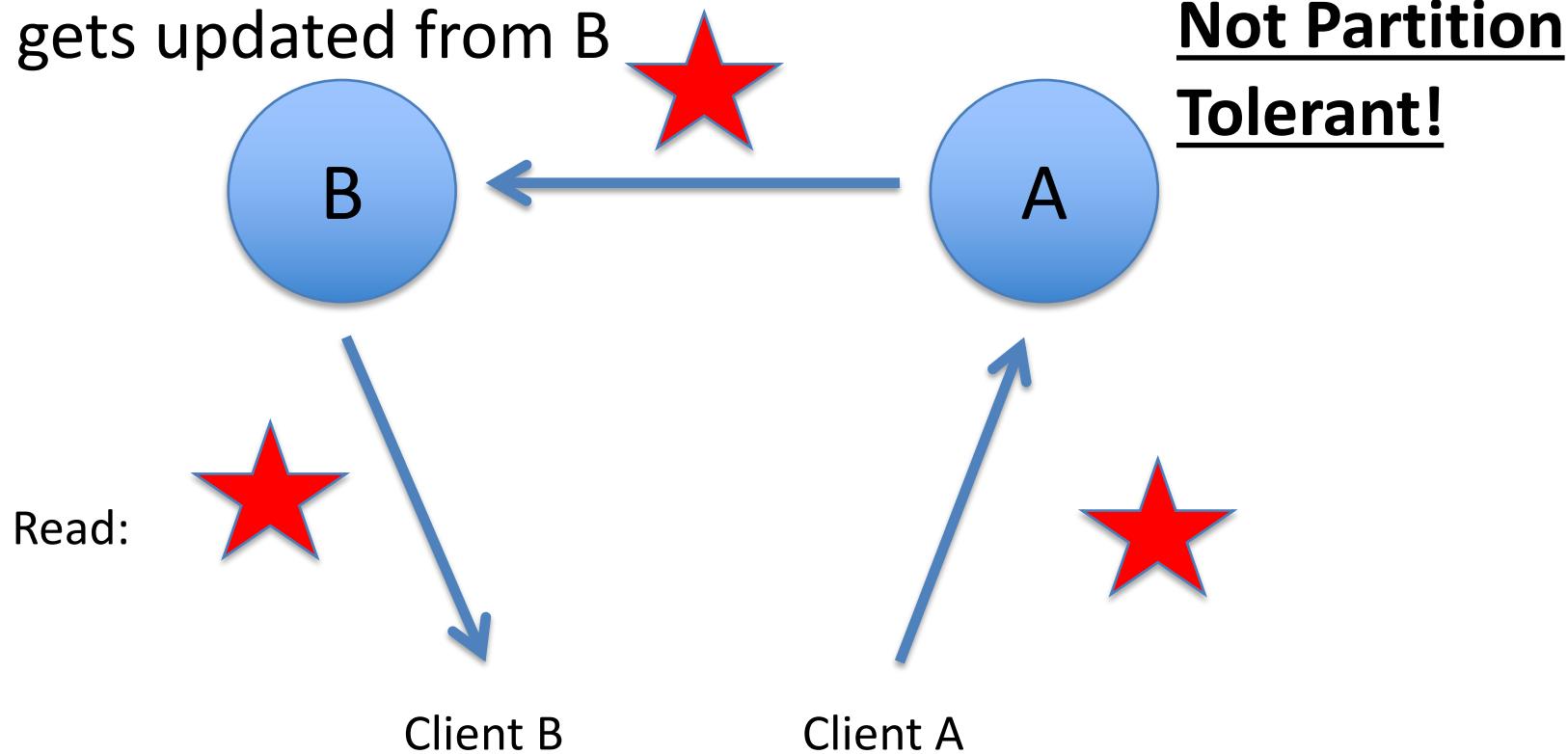
- A simple proof using two nodes:



CAP Theorem: Proof

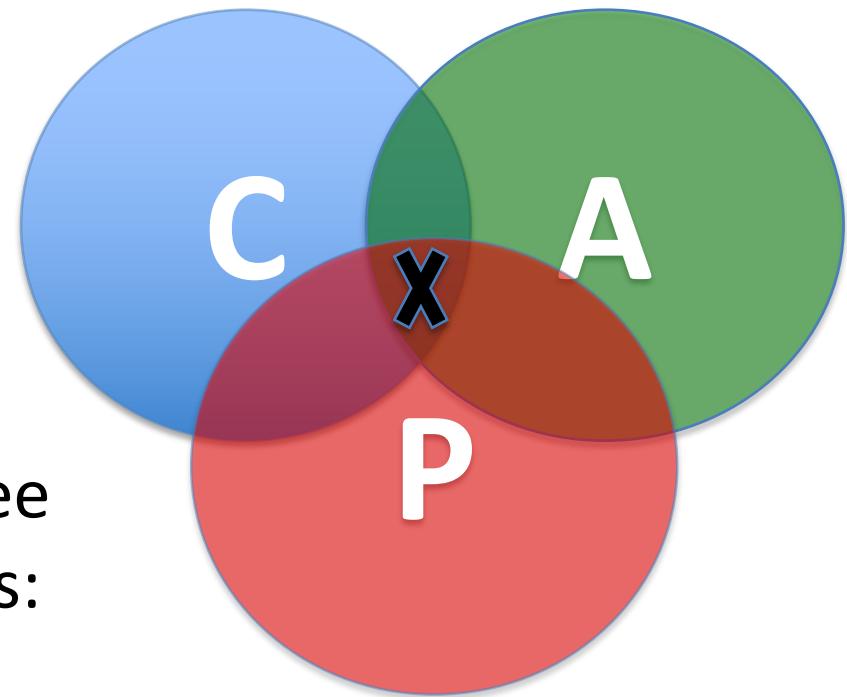
- A simple proof using two nodes:

A gets updated from B



CAP => 3 types of systems

- 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



Issues with CAP

- What does it mean to choose or not choose partition tolerance?
 - P is a property of the environment, C and A are goals
 - In other words, what's the difference between a "CA" and "CP" system? both give up availability on a partition!
- Better phrasing: "*if the network can have partitions, do we give up on consistency or availability?*"

Witnesses: P is unavoidable

- Coda Hale, Yammer (Microsoft?) 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/>

Witnesses: P is unavoidable

- 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.”*



Witnesses: P is unavoidable

- Daneil Abadi (UMD), Co-founder of Hadapt; Vertica, VoltDB contributor
 - *"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>

Witnesses: P?

Who cares about P!?

- Michael Stonebraker
 - [VoltDB, TuringAward'14]
 - *"In my experience, **network partitions do not happen often**. Specifically, they occur less frequently than the sum of bohrbugs [deterministic DB crashes], application errors, human errors and reprovisioning events. So it doesn't much matter what you do when confronted with network partitions. **Surviving them will not "move the needle" on availability** because higher frequency events will cause global outages. **Hence, you are giving up something (consistency) and getting nothing in return.***"



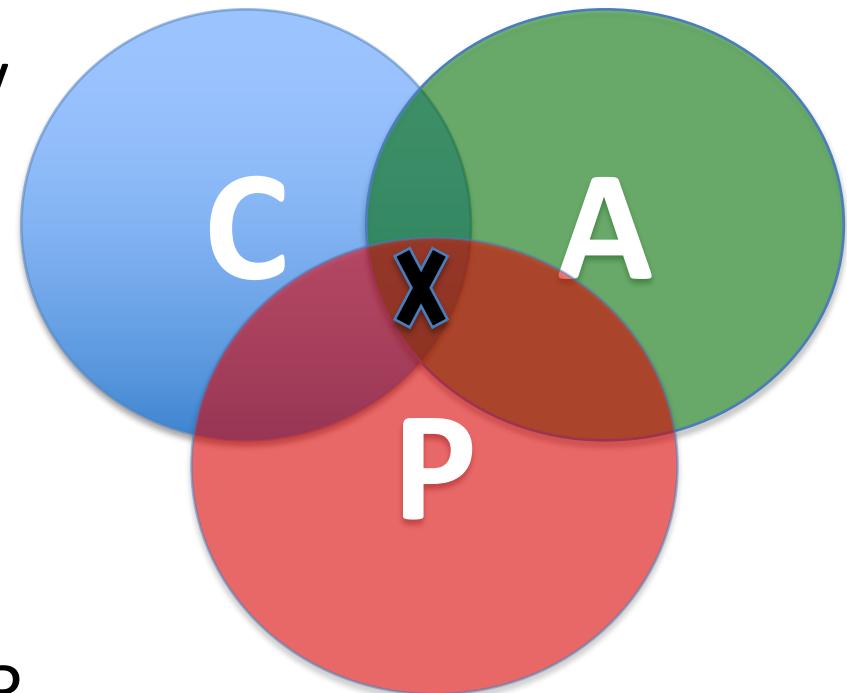
CAP Theorem 12 year later

- Eric Brewer: father of CAP
 - “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.”



Consistency or Availability

- *Consistency and Availability is not a “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

- Examples:
 - CDNs / Web caches
 - DNS
 - BlockChain
 - CRDTs
- Traits:
 - Optimistic concurrency control
 - Expiration/Time-to-live
 - Conflict resolution

CP: Best Effort Availability

- Examples:
 - Majority protocols (Paxos, Raft)
 - Distributed Locking (Google Chubby Lock service)
- Traits:
 - 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 its 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

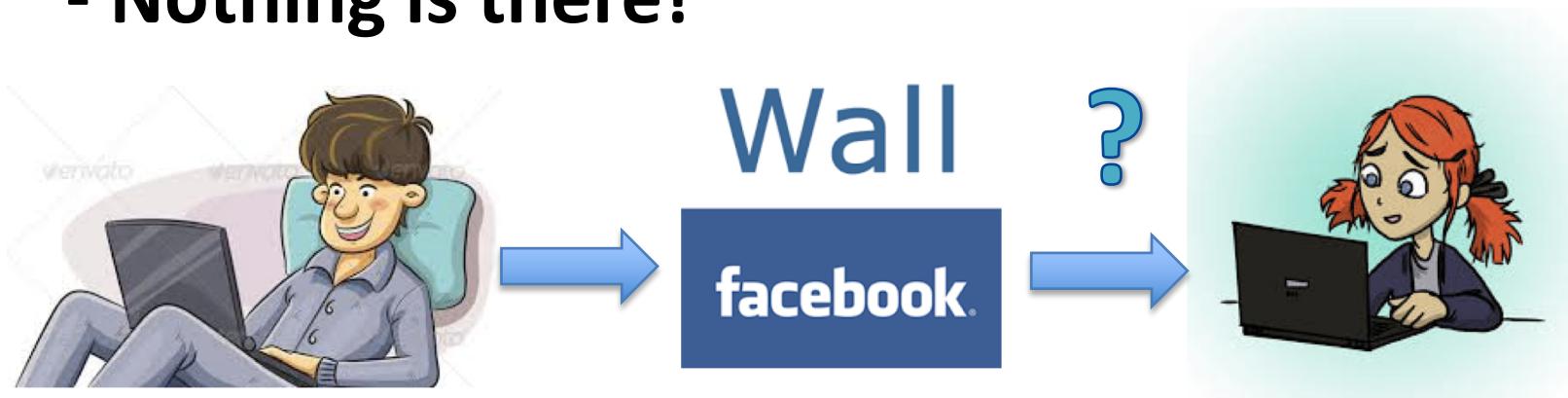
Eventual Consistency Variations

- 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

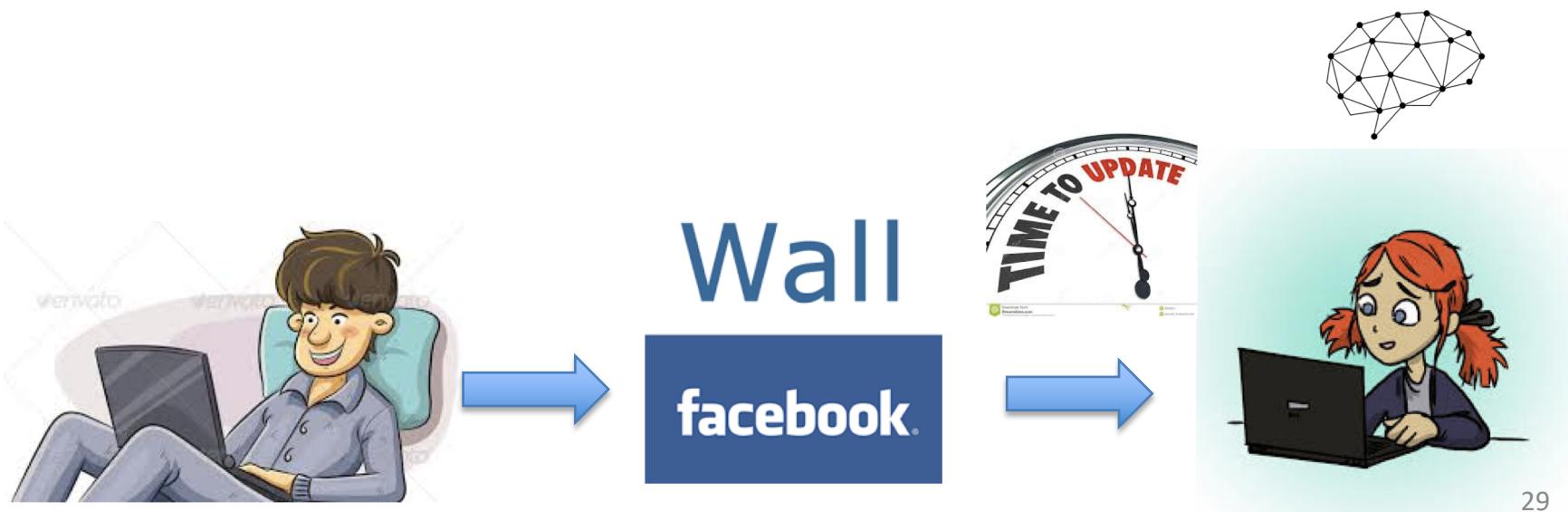
- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:
 - Nothing is there!



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 Cambridge Analytica story Bob shared with her!



Eventual Consistency

- A Facebook Example

- Reason: it is possible because Facebook uses an **eventual consistent model**
- Why would Facebook choose an 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**

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 to 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

Partitioning Strategies

- Data Partitioning
- Operational Partitioning
- Functional Partitioning
- User Partitioning
- Hierarchical Partitioning
- Idea: provide differentiated guarantees
depending on X {data/op/func/user/component}

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 (microservices)
- 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

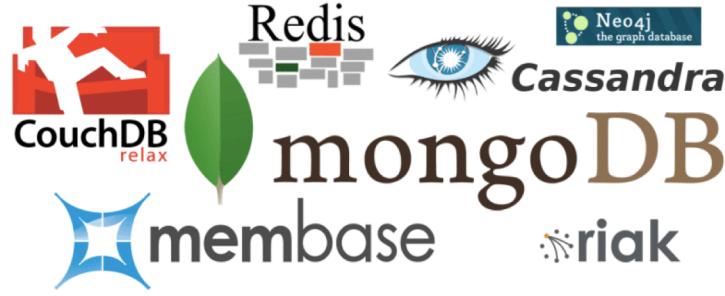
User Partitioning

- Try to keep related data close together to assure better performance
- Example: Craigslist
 - 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 close to them
 - Poorer performance if a Vancouver user queries Craglist in San Francisco

Partitioning Examples

Hierarchical (node) 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
 - Systems that do this: DNS, NTP



Take-aways

- CAP is a tool for thinking about trade-offs in distributed systems
- Misinterpreted + contentious
- The devil (in designing distributed systems) is often in the details: real systems cannot be classified into one of CA/AP/CP
- Many eventual consistency variants, widely adopted by popular systems

C

P
41