Strong Consistency & CAP Theorem

CS 475: Concurrent & Distributed Systems (Fall 2021)

Lecture 11

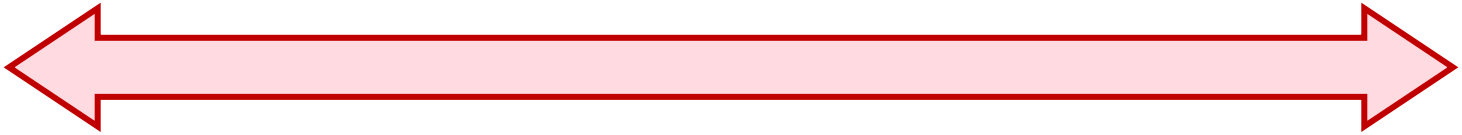
Yue Cheng

Some material taken/derived from:

- Princeton COS-418 materials created by Michael Freedman.
- MIT 6.824 by Robert Morris, Frans Kaashoek, and Nickolai Zeldovich.

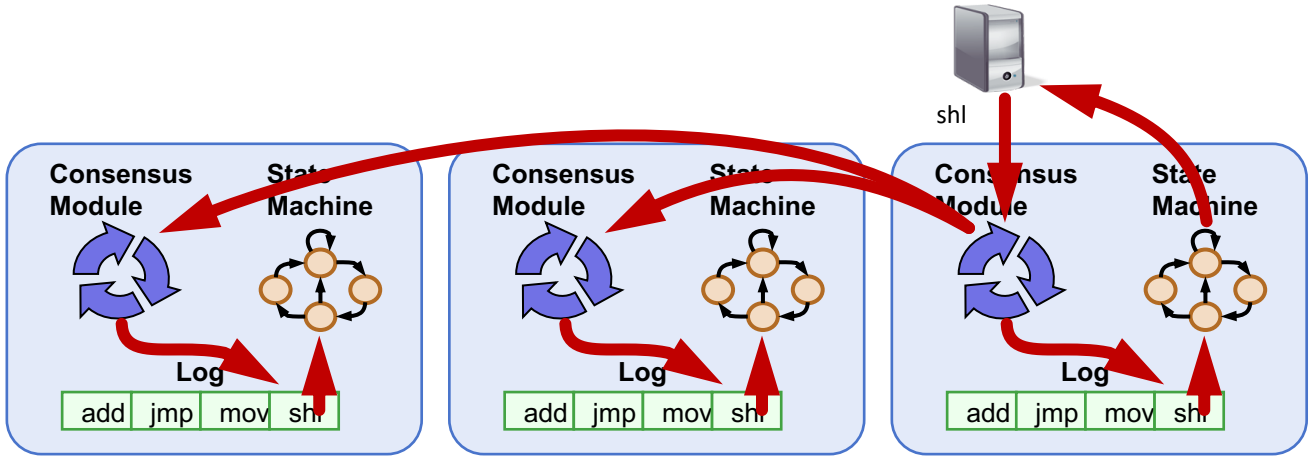
Consistency models

2PC / Consensus **Eventual consistency**



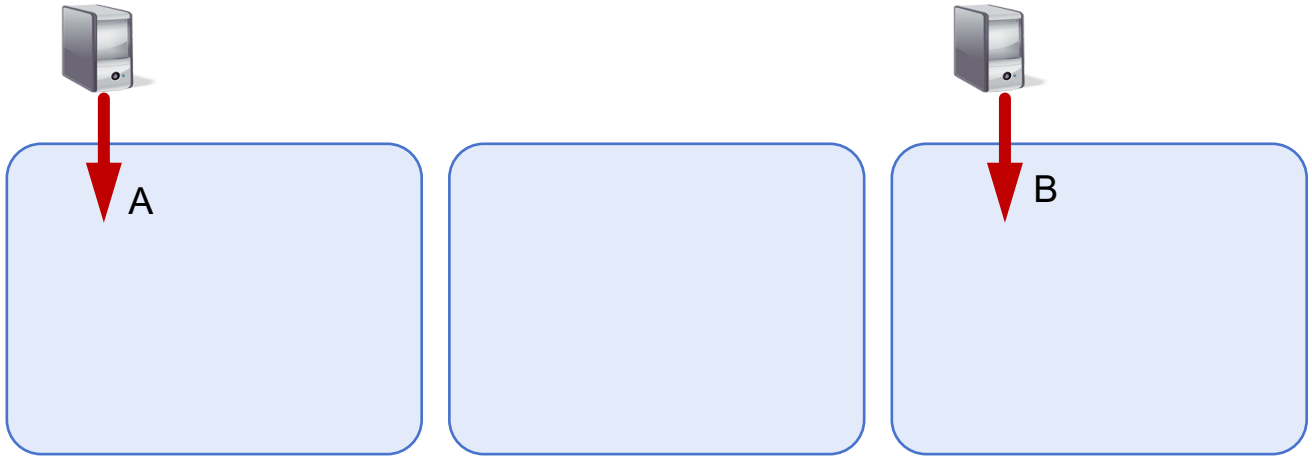
Paxos / Raft **Dynamo**

Consistency in Paxos/Raft



- Fault-tolerance / durability: Don't lose operations
- Consistency: Ordering between (visible) operations

Correct consistency model?



- Let's say A and B send an op.
- All readers see A → B ?
- All readers see B → A ?
- Some see A → B and others B → A ?

Paxos/Raft has strong consistency

Paxos/Raft has strong consistency

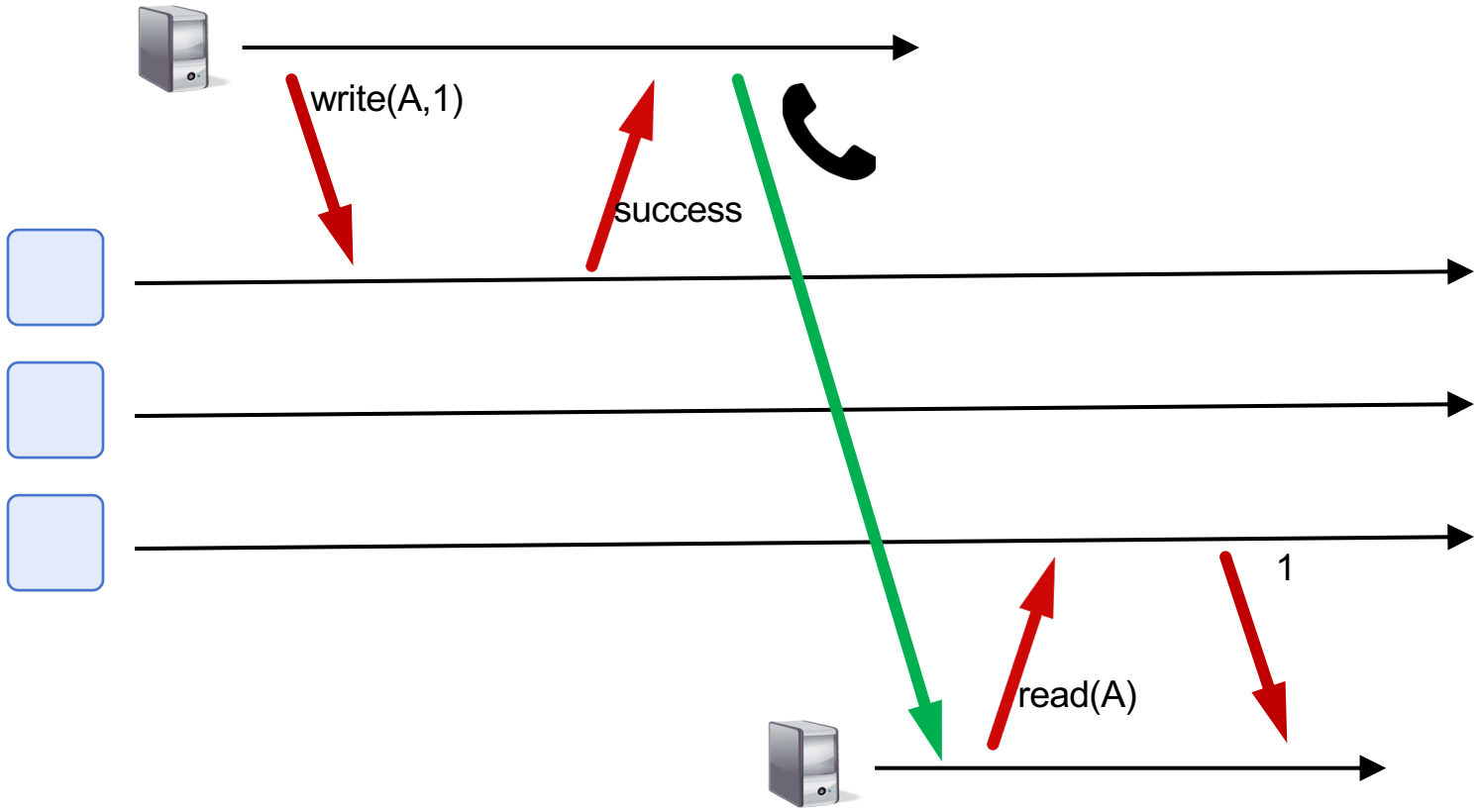
- Provide behavior of a single copy of object:
  - Read should return the most recent write
  - Subsequent reads should return same value, until next write

Paxos/Raft has strong consistency

- Provide behavior of a single copy of object:
  - Read should return the most recent write
  - Subsequent reads should return same value, until next write

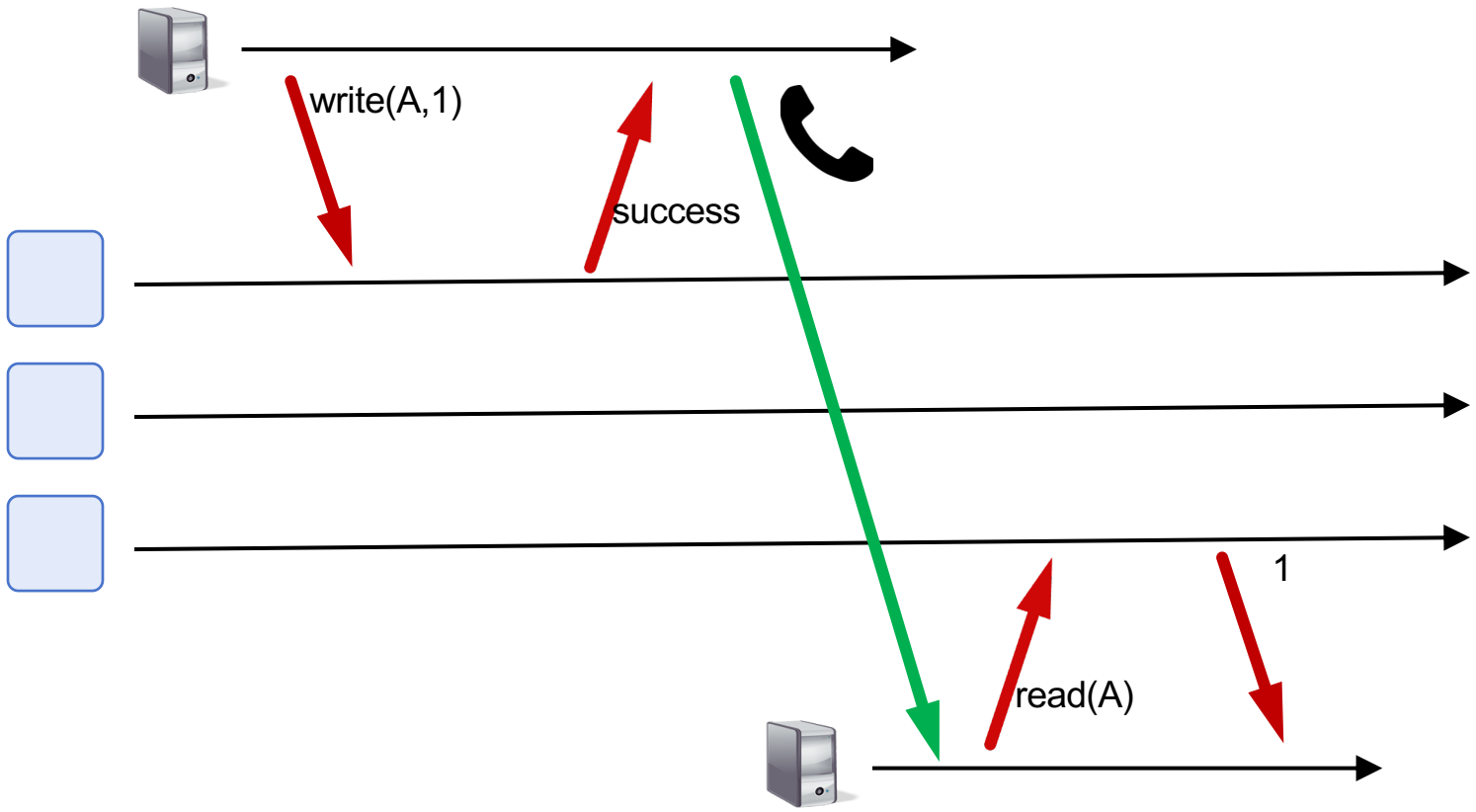
- Telephone intuition:
  1. Alice updates Facebook post
  2. Alice calls Bob on phone: “Check my Facebook post!”
  3. Bob read’s Alice’s wall, sees her post

Strong Consistency?



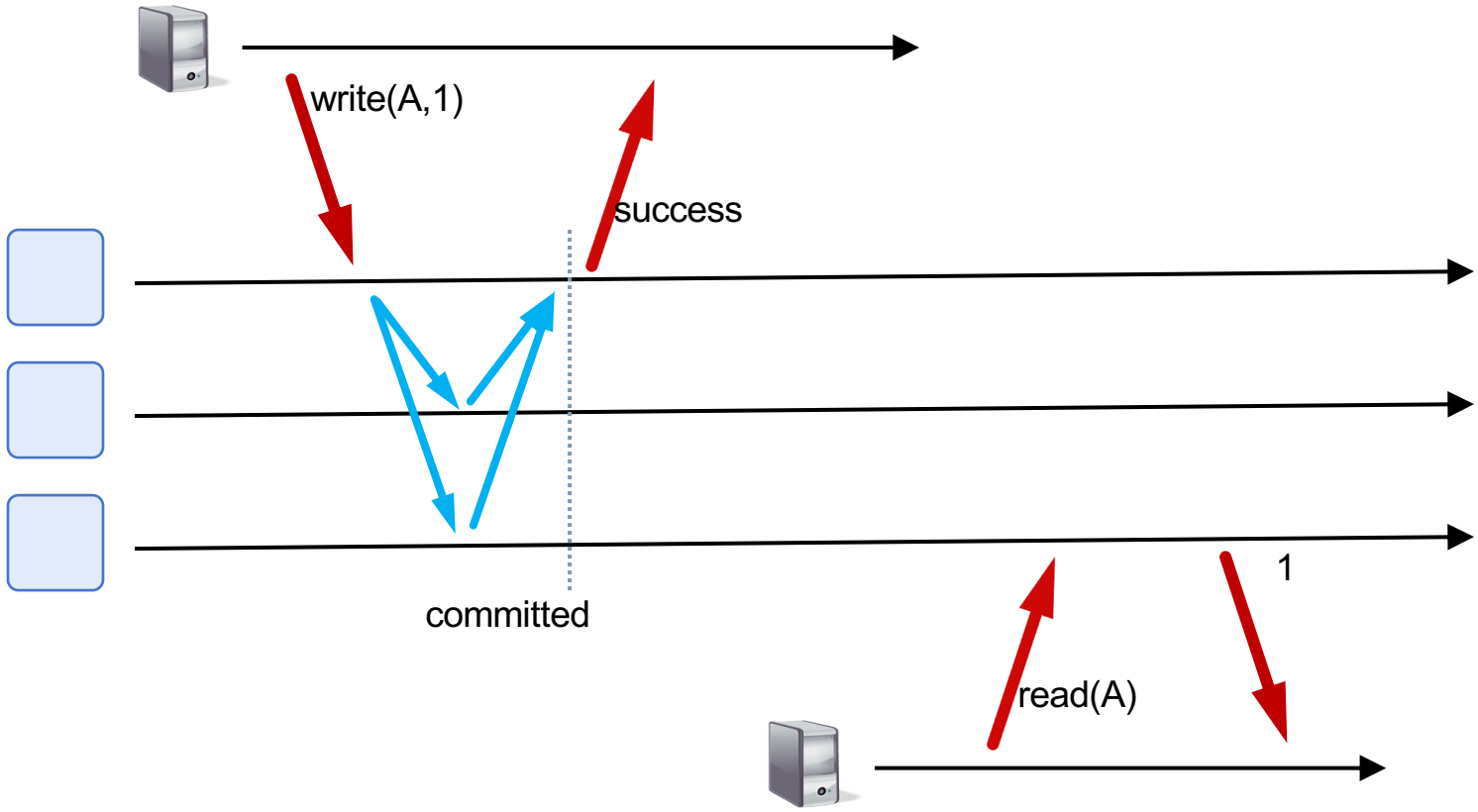
Phone call: Ensures *happens-before* relationship,
even through "out-of-band" communication

Strong Consistency?



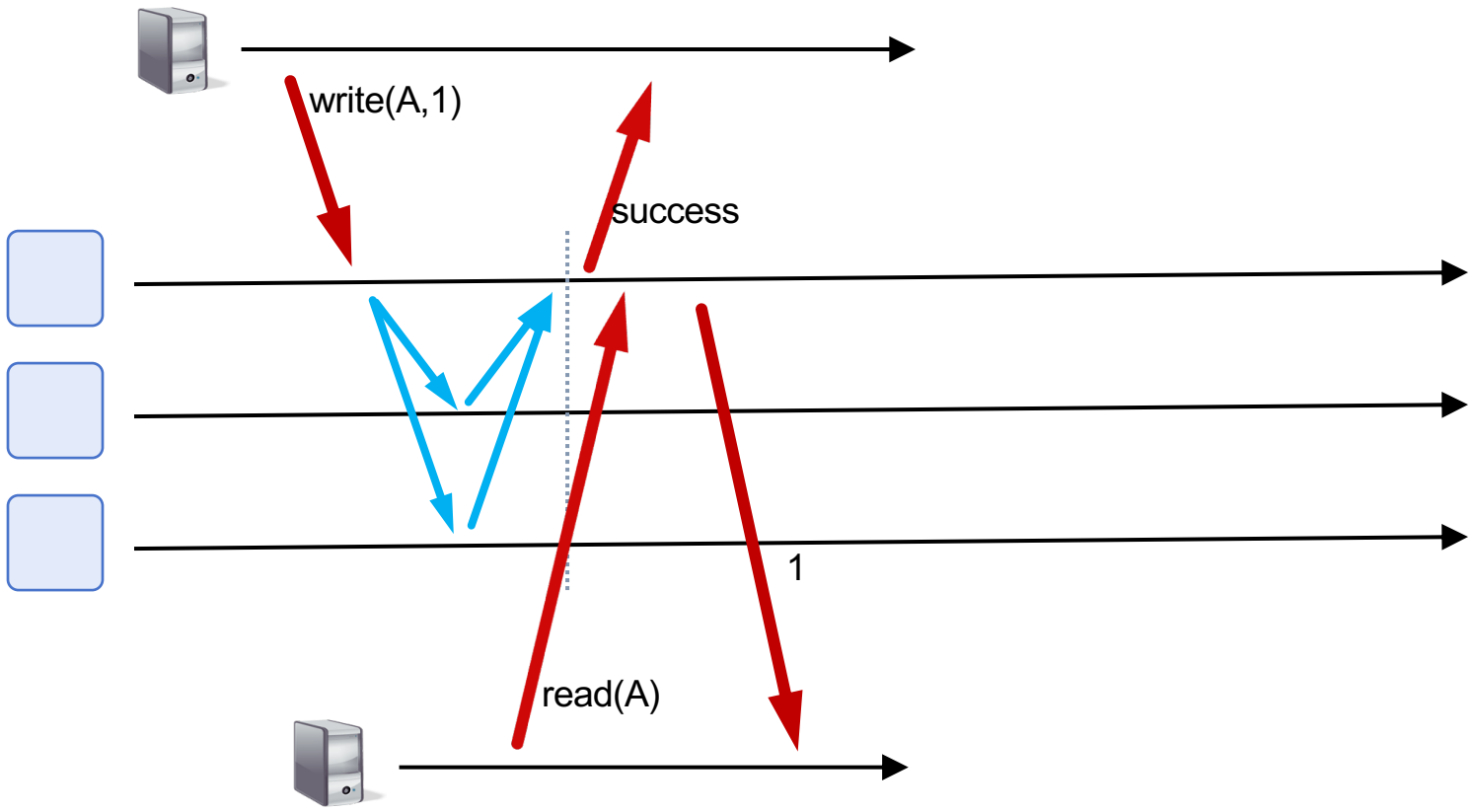
One cool trick: Delay responding to writes/ops until properly committed

Strong Consistency? This is buggy!



- Isn't sufficient to return value of third node:
  It doesn't know precisely when op is "globally" committed
- Instead: Need to actually *order* read operation

Strong Consistency!



Order all operations via (1) leader, (2) consensus

Strong consistency = linearizability

- Linearizability (Herlihy and Wing 1991)
  1. All servers execute all ops in *some* identical sequential order
  2. Global ordering preserves each client's own local ordering
  3. Global ordering preserves **real-time guarantee**

Informally, linearizability specifies that each concurrent operation **appears** to occur **instantaneously** and **exactly once** at some point in time between its invocation and its completion.

Strong consistency = linearizability

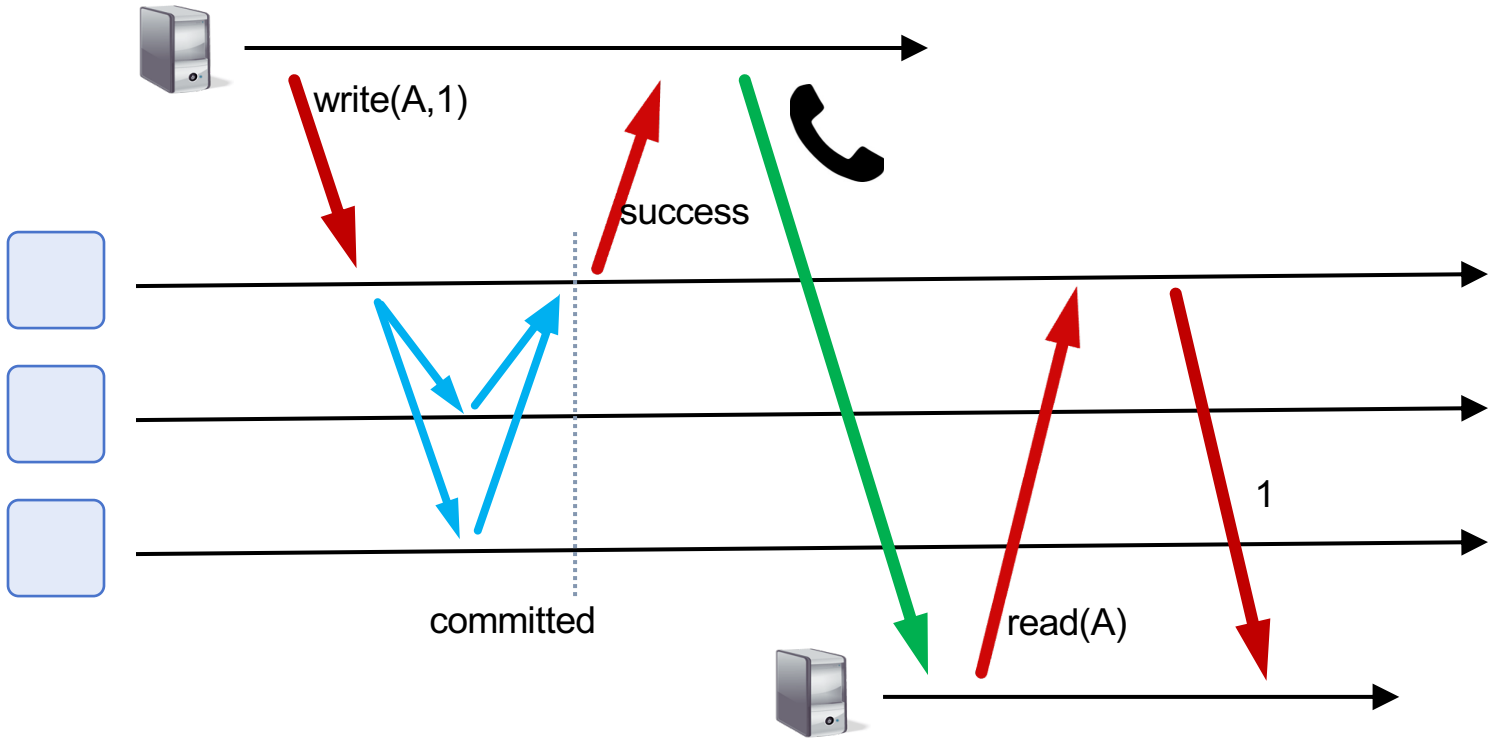
- Linearizability (Herlihy and Wing 1991)
  1. All servers execute all ops in *some* identical sequential order
  2. Global ordering preserves each client's own local ordering
  3. Global ordering preserves **real-time guarantee**
     - All ops receive global time-stamp using a sync'd clock
     - If $ts_{op1}(x) < ts_{op2}(y)$, OP1(x) precedes OP2(y) in sequence

Strong consistency = linearizability

- Linearizability (Herlihy and Wing 1991)
  1. All servers execute all ops in *some* identical sequential order
  2. Global ordering preserves each client's own local ordering
  3. Global ordering preserves **real-time guarantee**
     - All ops receive global time-stamp using a sync'd clock
     - If $ts_{op1}(x) < ts_{op2}(y)$, OP1(x) precedes OP2(y) in sequence

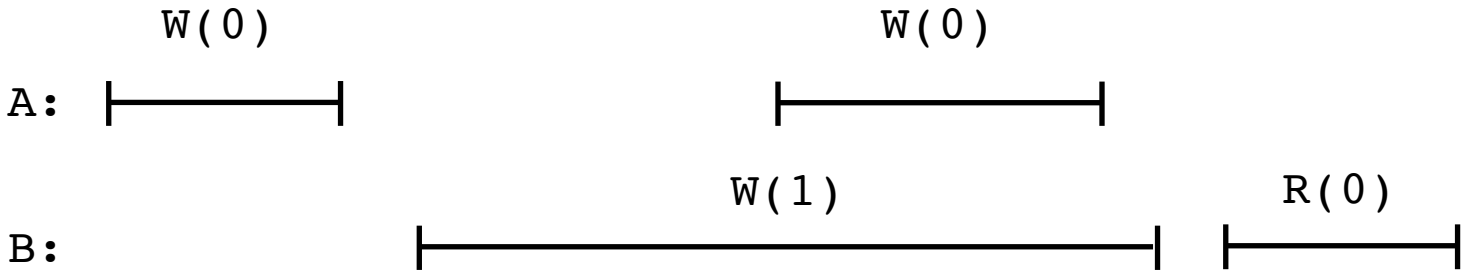
- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

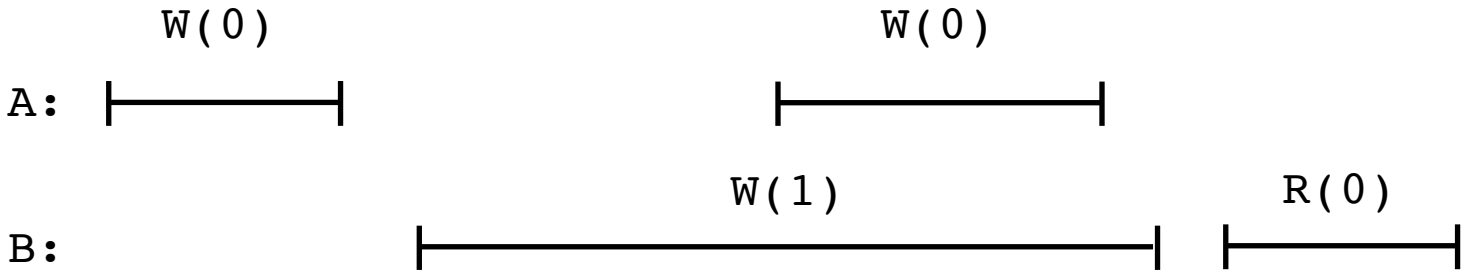
Intuition: Real-time ordering

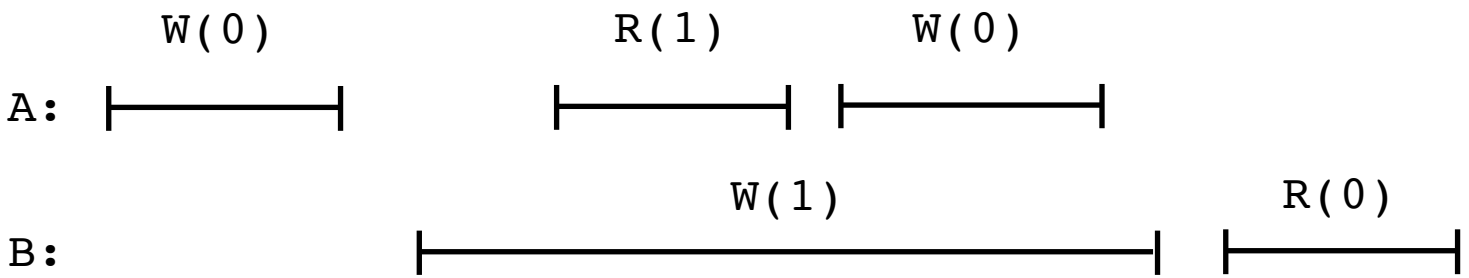


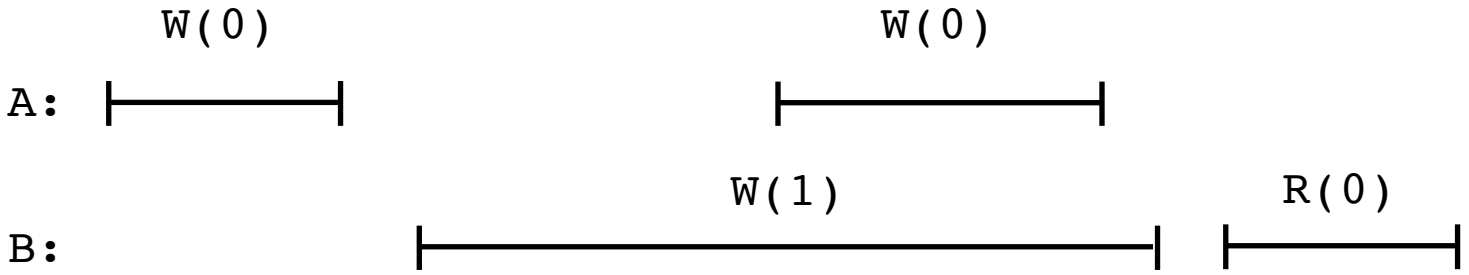
- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

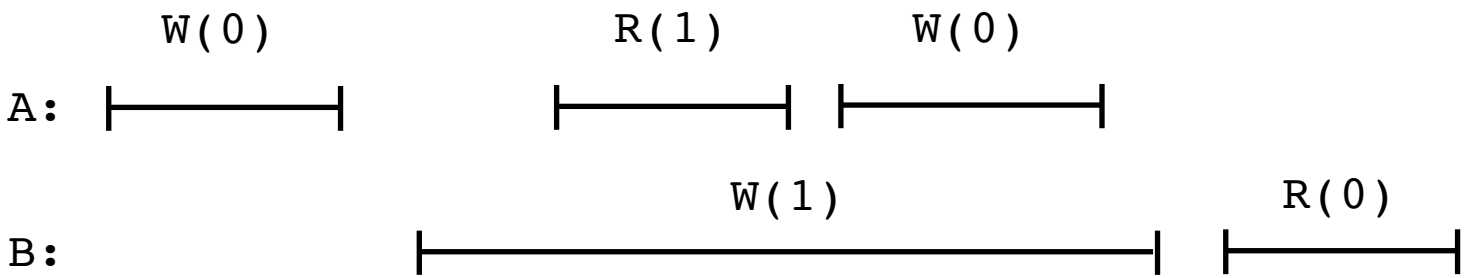
Real-time ordering examples

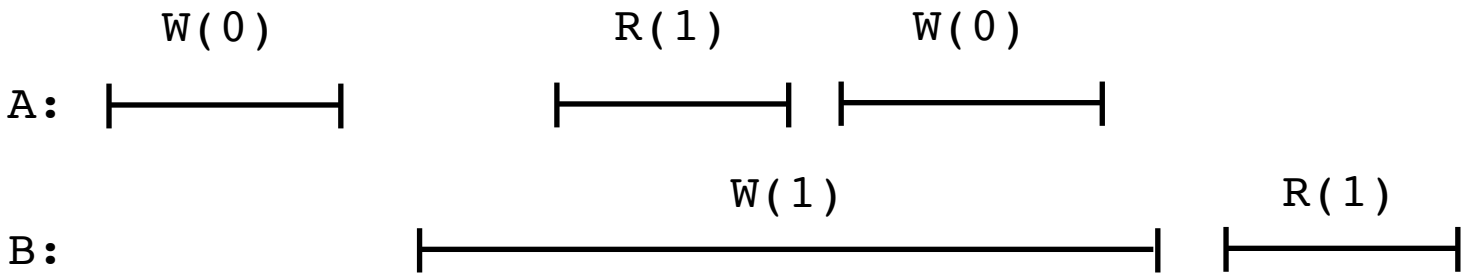

W(0)
W(0)
A:
W(1)
R(0)
B:


W(0)
W(0)
A:
W(1)
R(0)
B:

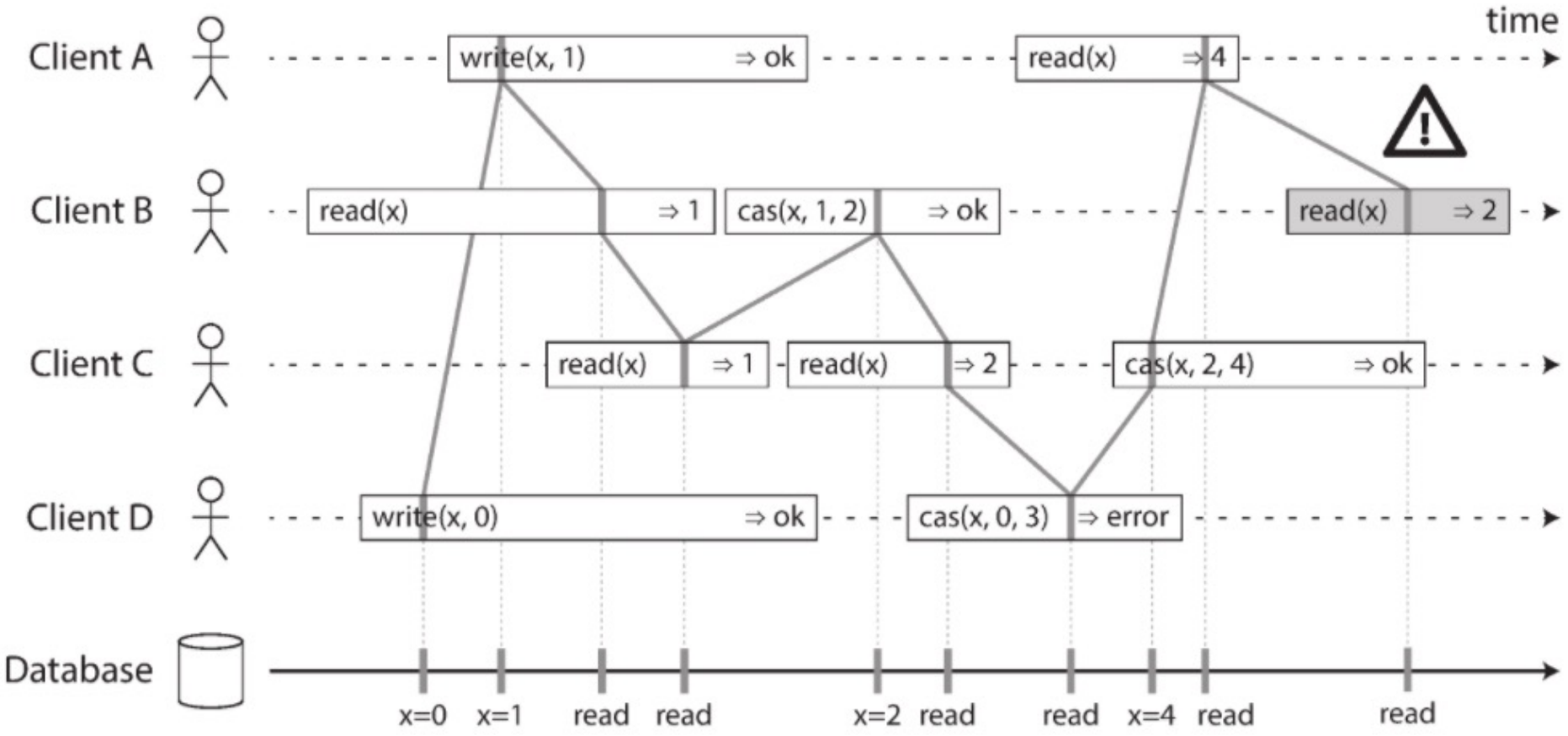

W(0)
R(1)
W(0)
A:
W(1)
R(0)
B:


W(0)
W(0)
A:
W(1)
R(0)
B:


W(0)
R(1)
W(0)
A:
W(1)
R(0)
B:


W(0)
R(1)
W(0)
A:
W(1)
R(1)
B:

Real-time ordering examples



*: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ (Page 328)

Weaker: Sequential consistency

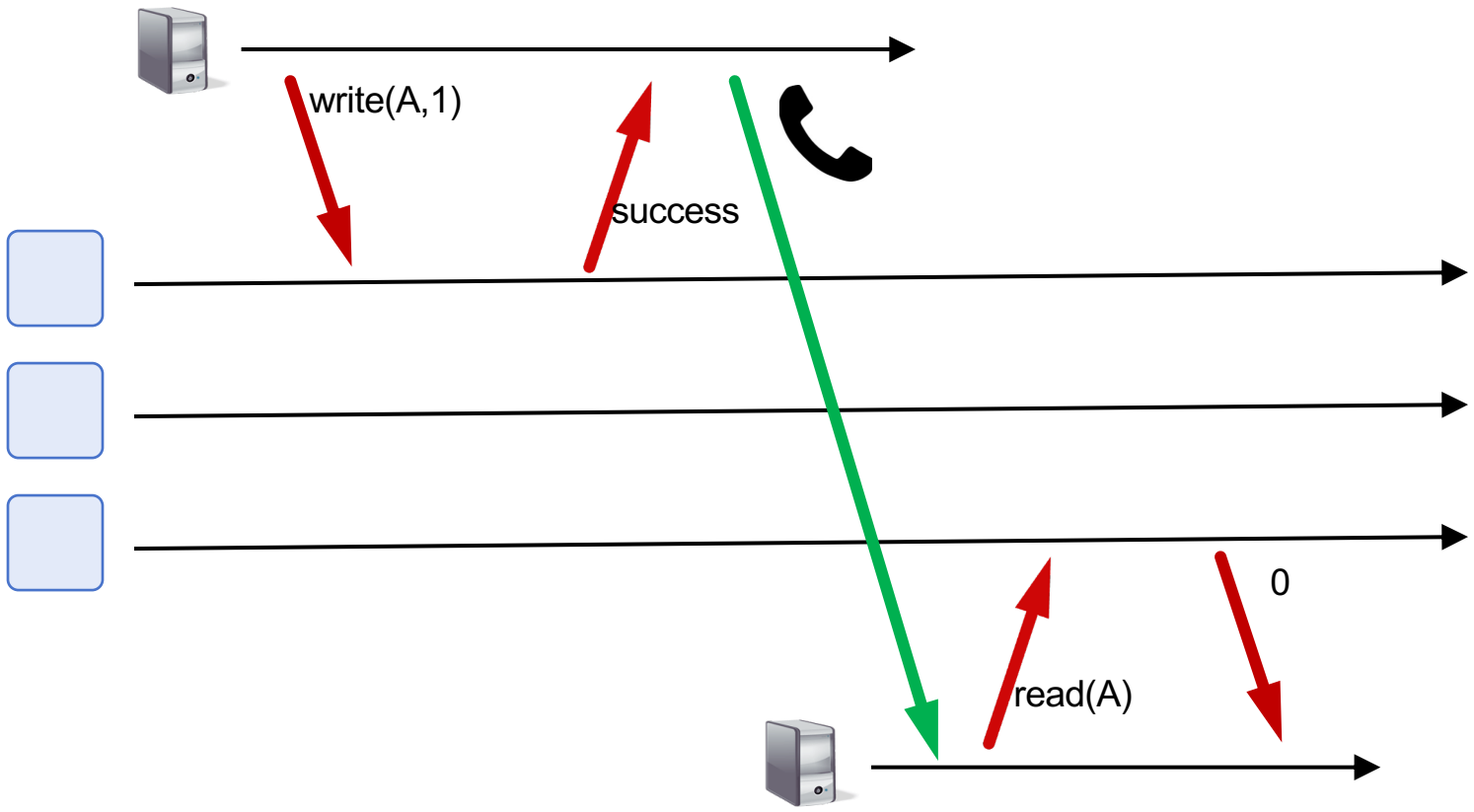
- Sequential = Linearizability – real-time ordering
    1. All servers execute all ops in *some* identical sequential order
    2. Global ordering preserves each client's own local ordering

Weaker: Sequential consistency

- Sequential = Linearizability – real-time ordering
  1. All servers execute all ops in *some* identical sequential order
  2. Global ordering preserves each client's own local ordering

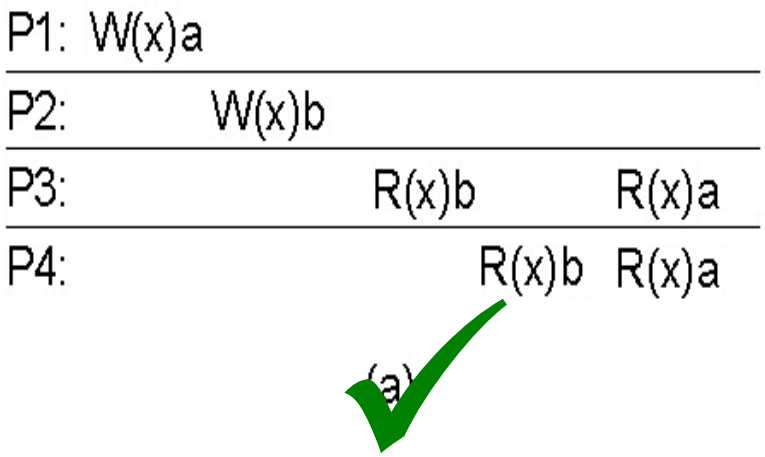
- With concurrent ops, "reordering" of ops (w.r.t. real-time ordering) acceptable, but all servers must see same order
  - e.g., linearizability cares about time
    sequential consistency cares about program order

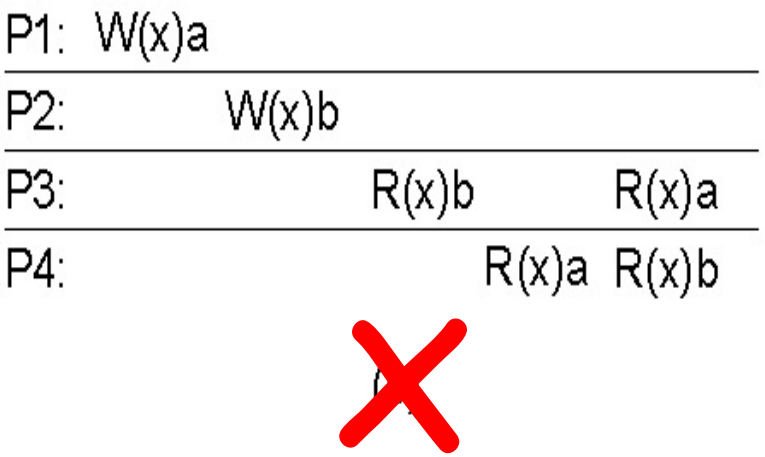
Sequential Consistency



In example, system orders read(A) before write(A,1)

Valid Sequential Consistency?

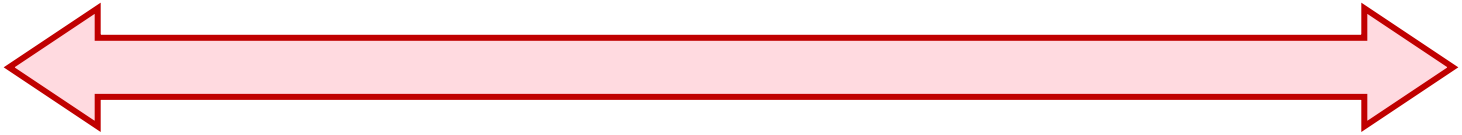




- Why? Because P3 and P4 don't agree on order of ops. Doesn't matter when events took place on diff machine, as long as proc's AGREE on order.
- What if P1 did both W(x)a and W(x)b?
  - Neither valid, as (a) doesn't preserve local ordering

Tradeoffs are fundamental?

2PC / Consensus **Eventual consistency**



Paxos / Raft **Dynamo**

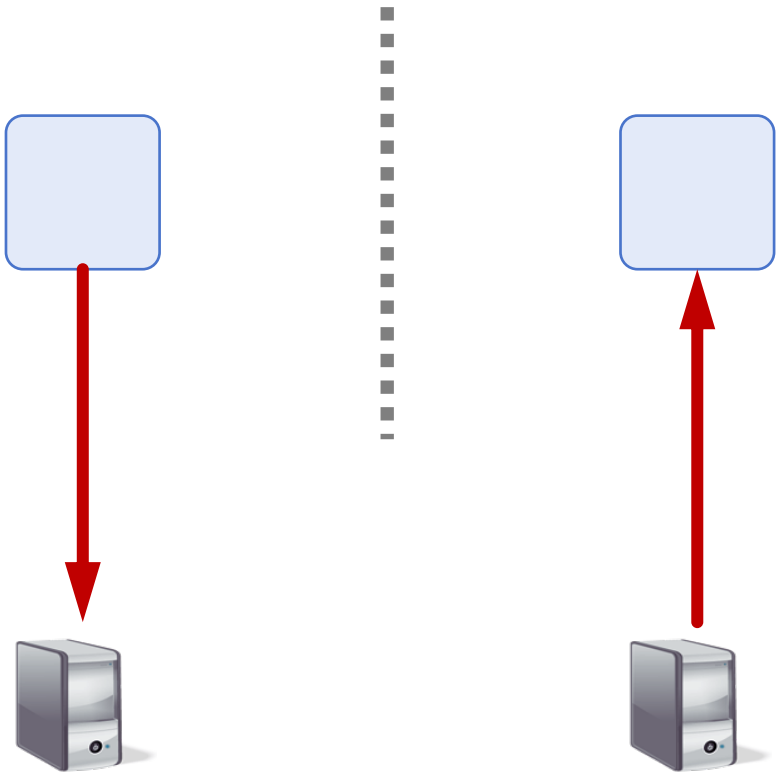
"CAP" conjection for distributed systems

- From keynote lecture by Eric Brewer (2000)
  - History: Eric started Inktomi, early Internet search site based around "commodity" clusters of computers
  - Using CAP to justify "BASE" model: Basically Available, Soft-state services with Eventual consistency

"CAP" conjection for distributed systems

- From keynote lecture by Eric Brewer (2000)
  - History: Eric started Inktomi, early Internet search site based around "commodity" clusters of computers
  - Using CAP to justify "BASE" model: Basically Available, Soft-state services with Eventual consistency
- Popular interpretation: 2-out-of-3
  - Consistency (Linearizability)
  - Availability
  - Partition Tolerance: Arbitrary crash/network failures

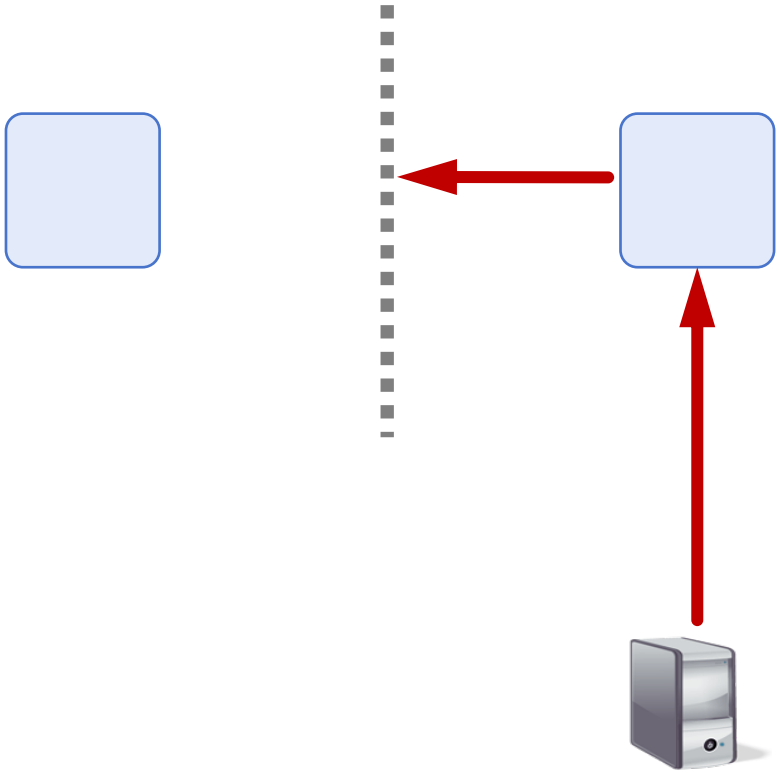
CAP Theorem: Proof



Not consistent

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.

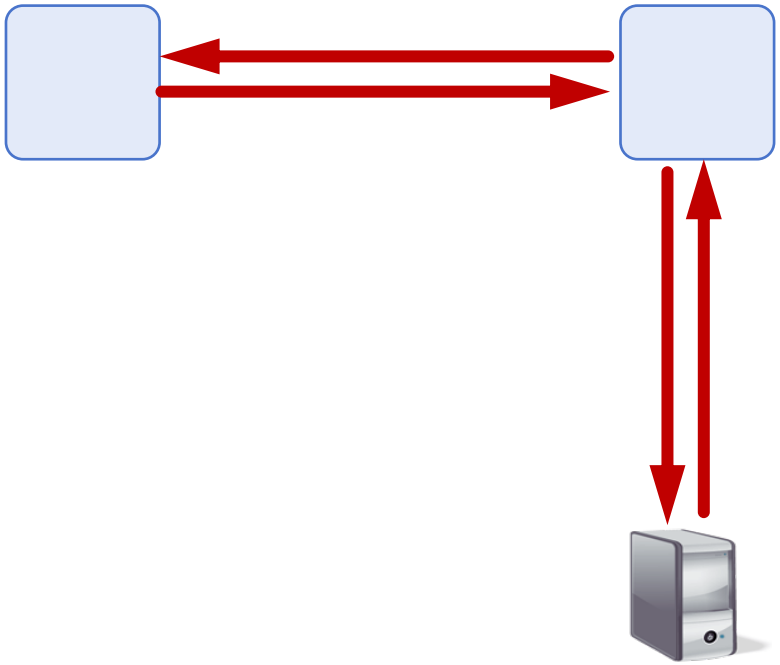
CAP Theorem: Proof



Not available

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.

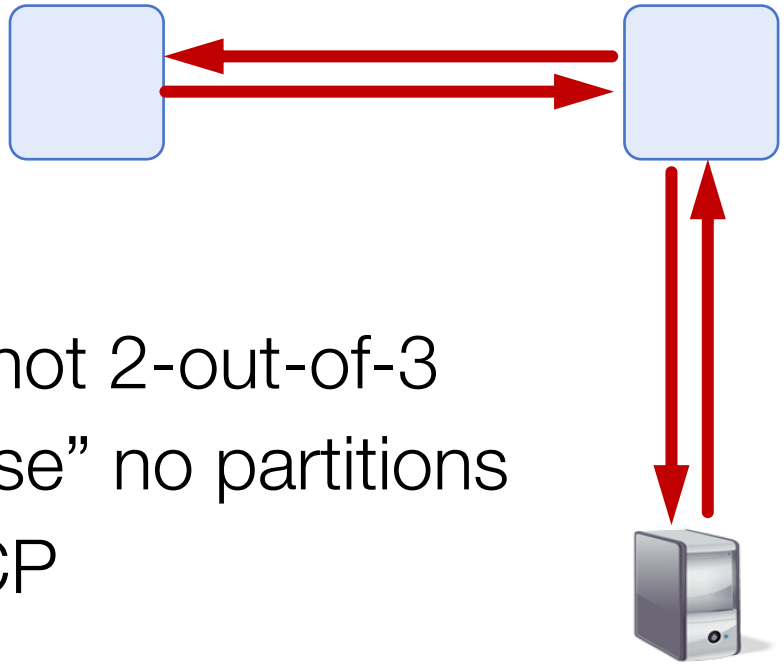
CAP Theorem: Proof



Not partition tolerant

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: AP or CP



Criticism: It's not 2-out-of-3

- Can't "choose" no partitions
- So: AP or CP

Not partition tolerant

More tradeoffs L vs. C

- Low-latency: Speak to fewer than quorum of nodes?
  - 2PC: write N, read 1
  - Raft: write ⌊N/2⌋ + 1, read ⌊N/2⌋ + 1
  - General: $|W| + |R| > N$

More tradeoffs L vs. C

- Low-latency: Speak to fewer than quorum of nodes?
  - 2PC: write N, read 1
  - Raft: write ⌊N/2⌋ + 1, read ⌊N/2⌋ + 1
  - General: $|W| + |R| > N$
- L and C are fundamentally at odds
  - "C" = linearizability, sequential, serializability (more later)

PACELC

- If there is a partition (P):
  - How does system tradeoff A and C?
- Else (no partition)
  - How does system tradeoff L and C?

PACELC

- If there is a partition (P):
  - How does system tradeoff A and C?
- Else (no partition)
  - How does system tradeoff L and C?

- Is there a useful system that switches?
  - Dynamo: PA/EL
  - “ACID” DBs: PC/EC

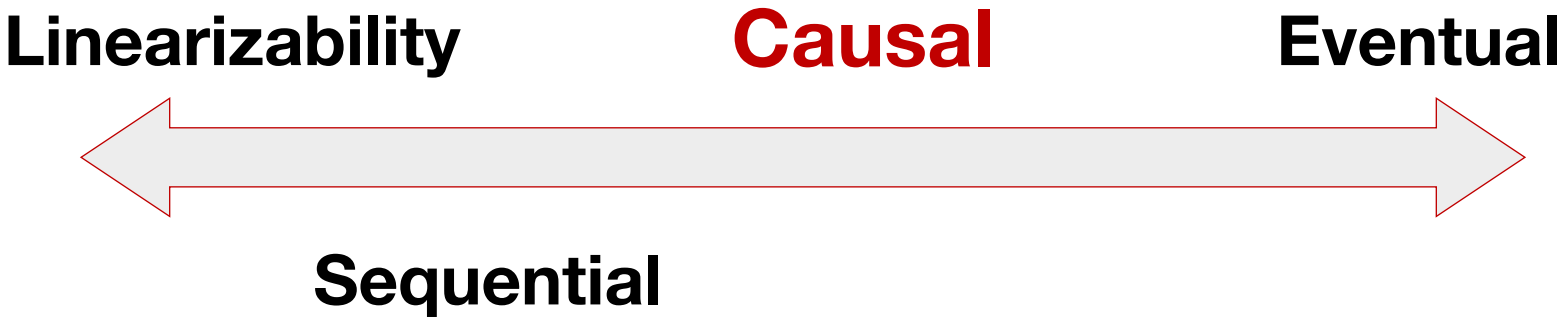
PACELC

- If there is a partition (P):
  - How does system tradeoff A and C?
- Else (no partition)
  - How does system tradeoff L and C?

- Is there a useful system that switches?
  - Dynamo: PA/EL
  - “ACID” DBs: PC/EC

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

Consistency models



Recall use of logical clocks (lec N?)

- Lamport clocks: $C(a) < C(z)$ Conclusion: None
- Vector clocks: $V(a) < V(z)$ Conclusion: $a \rightarrow \ldots \rightarrow z$

Recall use of logical clocks (lec N?)

- Lamport clocks: $C(a) < C(z)$ Conclusion: None
- Vector clocks: $V(a) < V(z)$ Conclusion: $a \rightarrow \ldots \rightarrow z$

- Distributed bulletin board application
  - Each post gets sent to all other users
  - Consistency goal: No user to see reply before the corresponding original message post
  - Conclusion: Deliver message only **after** all messages that **causally precede** it have been delivered

Causal Consistency

Causal Consistency

1. Writes that are *potentially* causally related must be seen by all machines in same order.

Causal Consistency

1. Writes that are *potentially* causally related must be seen by all machines in same order.

2. Concurrent writes may be seen in a different order on different machines.

Causal Consistency

1. Writes that are *potentially* causally related must be seen by all machines in same order.

2. Concurrent writes may be seen in a different order on different machines.

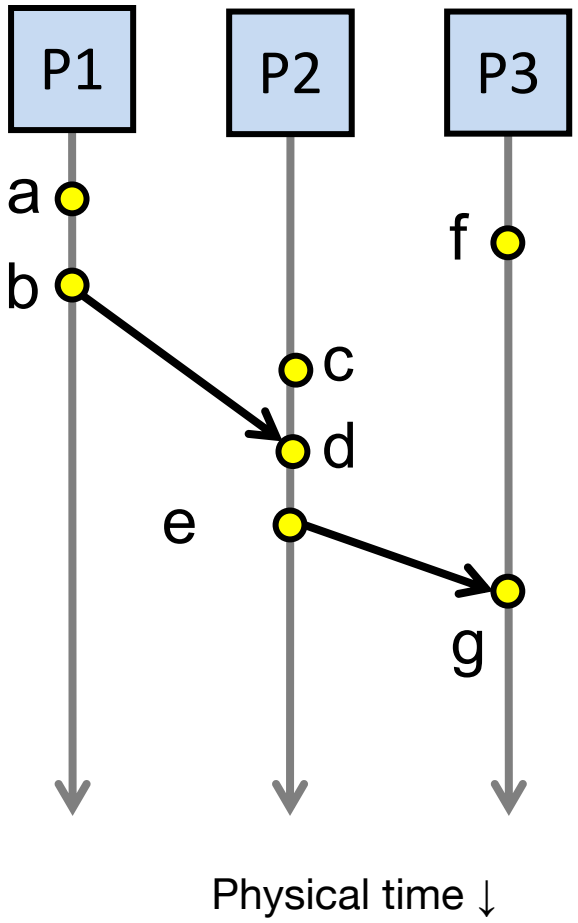
- Concurrent: Ops not causally related

Causal Consistency

1. Writes that are *potentially* causally related must be seen by all machines in same order.

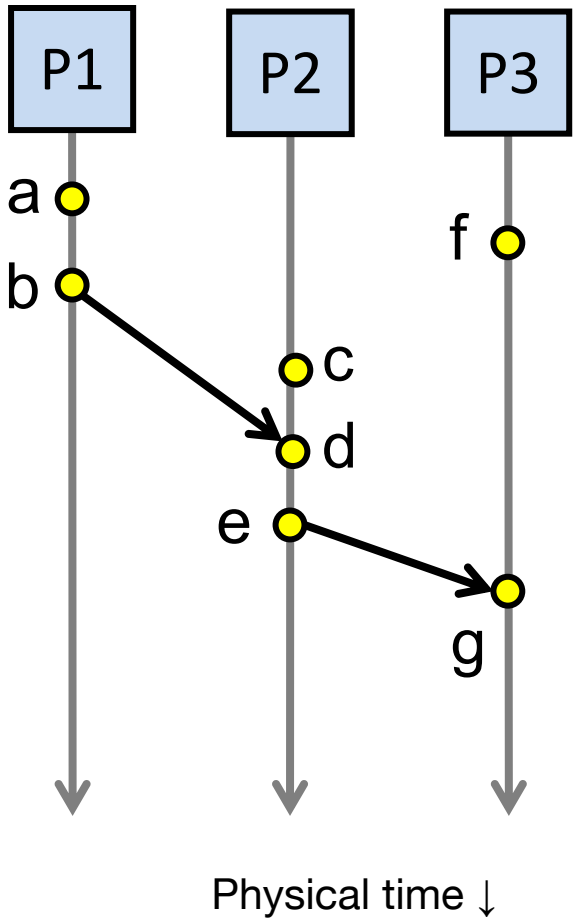
2. Concurrent writes may be seen in a different order on different machines.

- Concurrent: Ops not causally related



Causal Consistency

Operations	Concurrent?
a, b	
b, f	
c, f	
e, f	
e, g	
a, c	
a, e	



Causal Consistency

Operations	Concurrent?
a, b	**N**
b, f	**Y**
c, f	**Y**
e, f	**Y**
e, g	**N**
a, c	**Y**
a, e	**N**