Strong Consistency & CAP Theorem

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS 240: Computing Systems and Concurrency
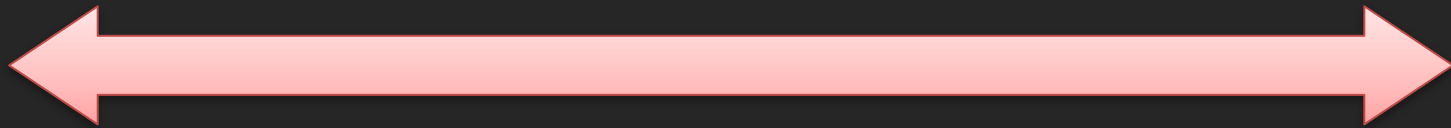## Lecture 14

Marco Canini

Credits: Michael Freedman and Kyle Jamieson developed much of the original material.

Consistency models
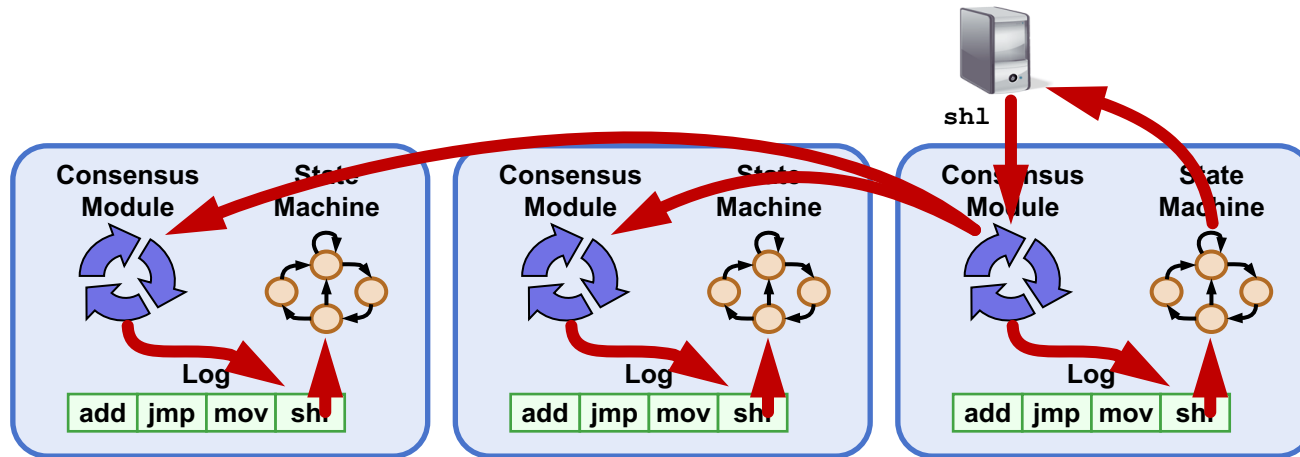
2PC / Consensus

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 $A \rightarrow B$?

- All readers see $B \rightarrow A$?

- Some see $A \rightarrow B$ and others $B \rightarrow A$?

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

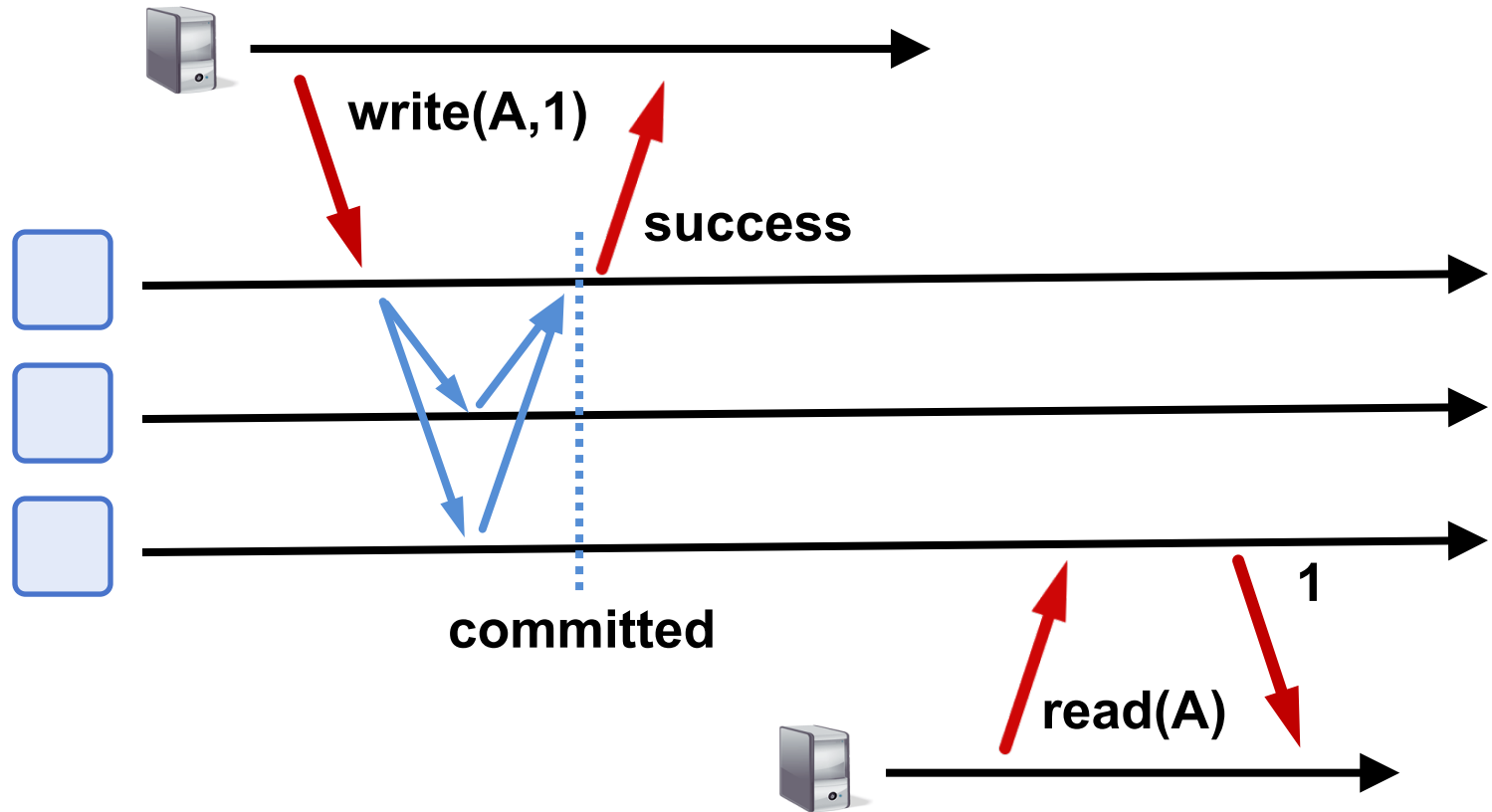# Strong Consistency?
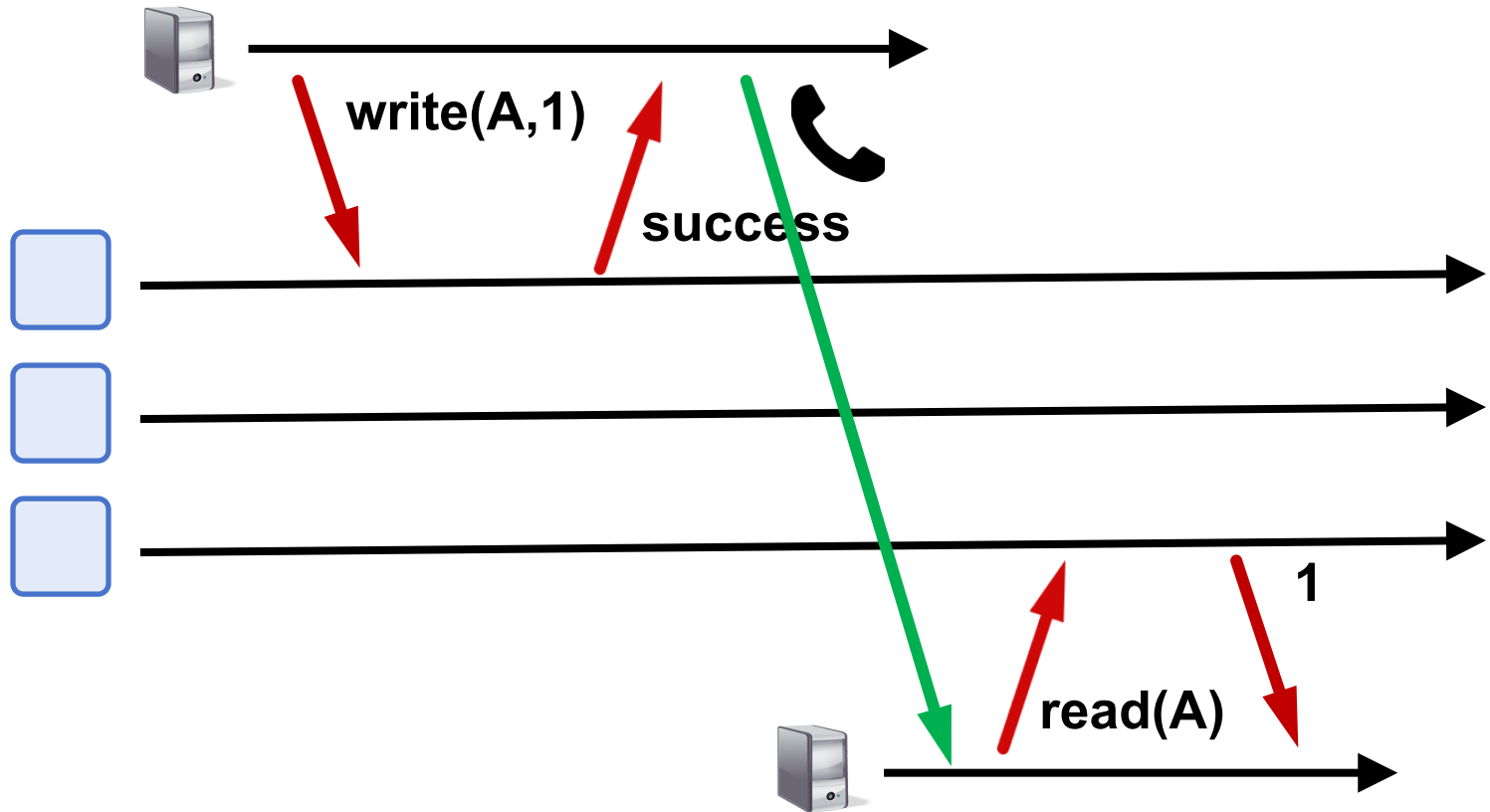
Phone call: Ensures *happens-before* relationship,
even through "out-of-band" communication

Strong Consistency?

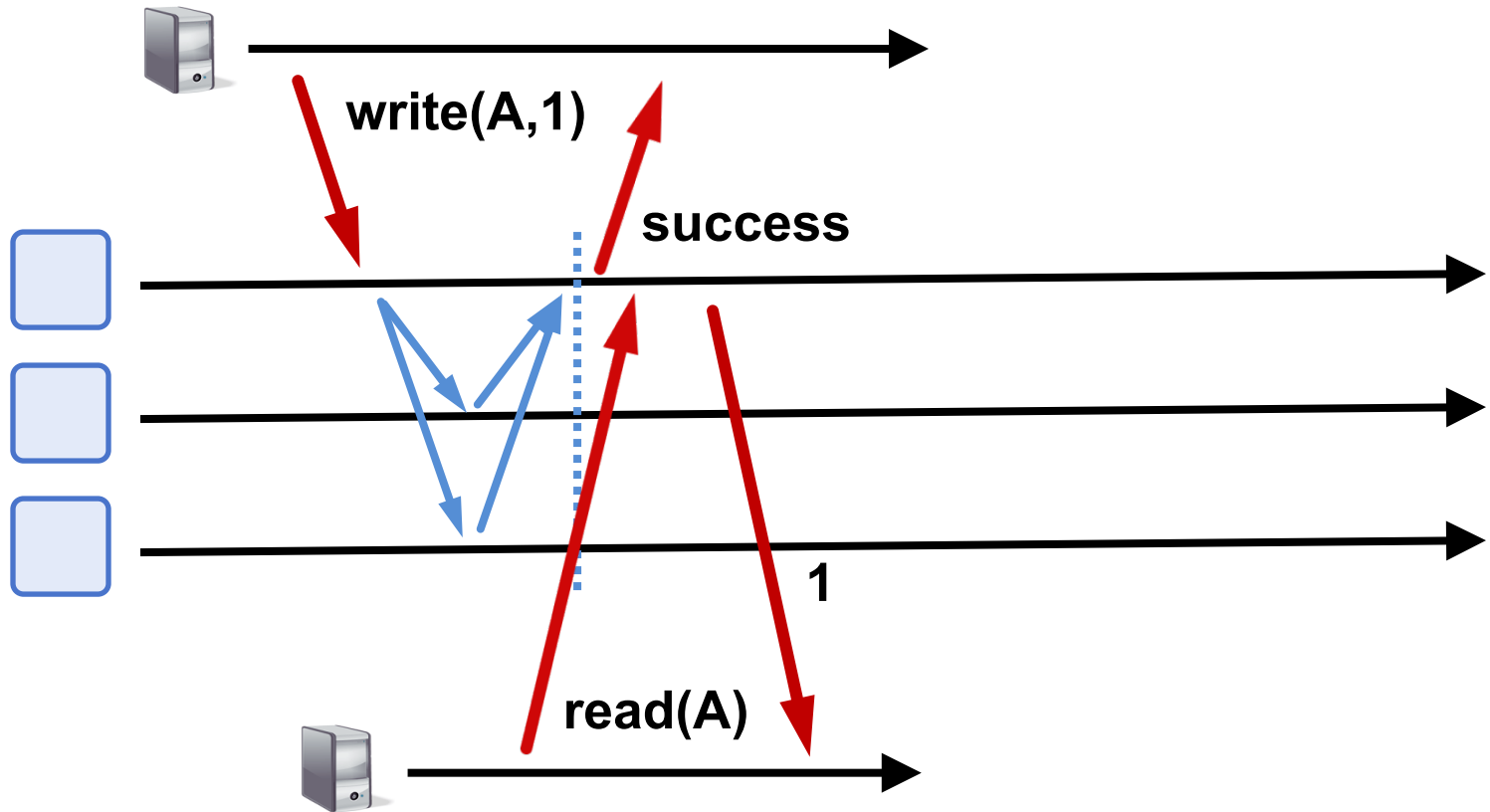write(A,1)

success

read(A)

1
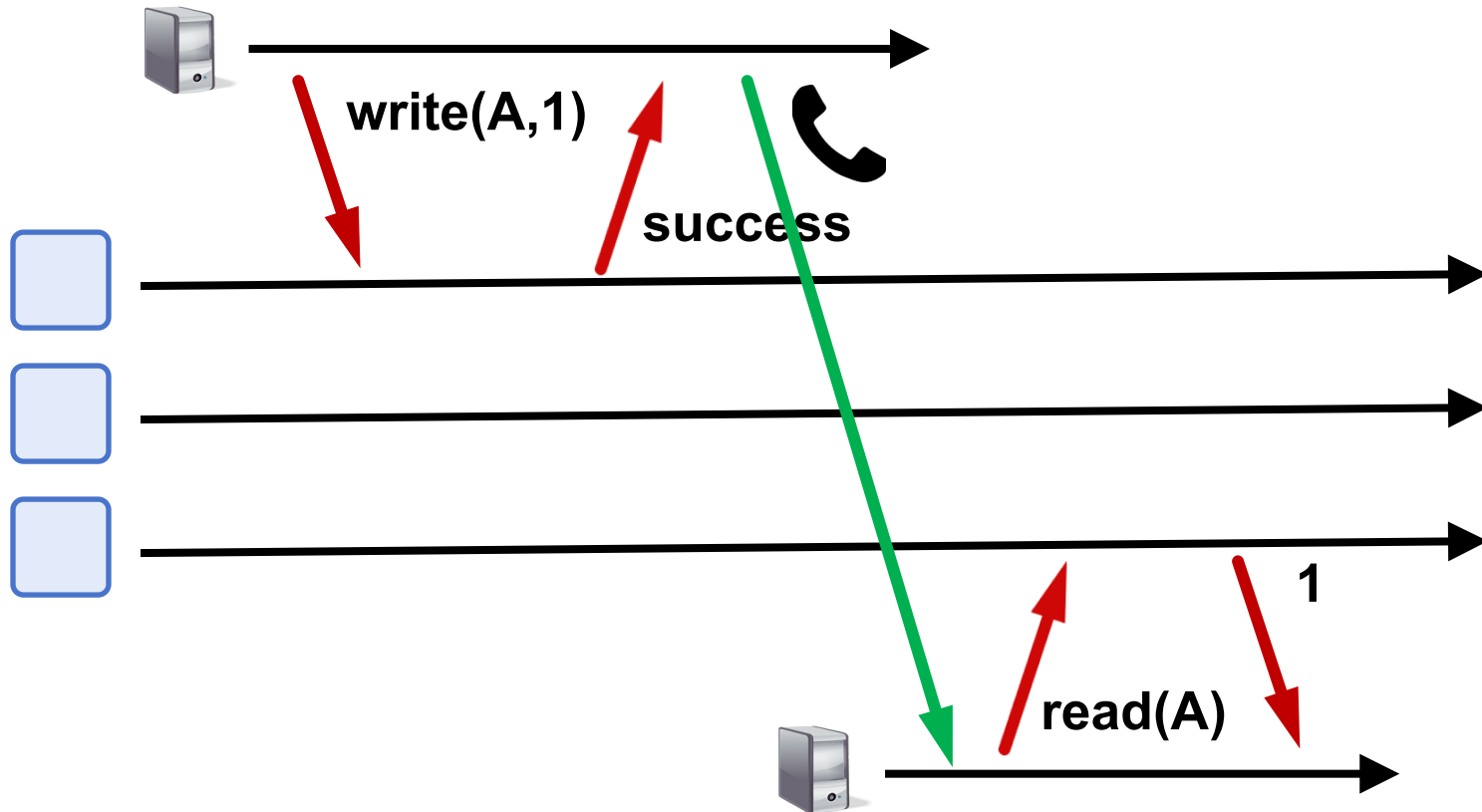
One cool trick: Delay responding to writes/ops until properly committed

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!


write(A,1)

success

1

read(A)

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

Strong consistency = linearizability

- Linearizability (Herlihy and Wang 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
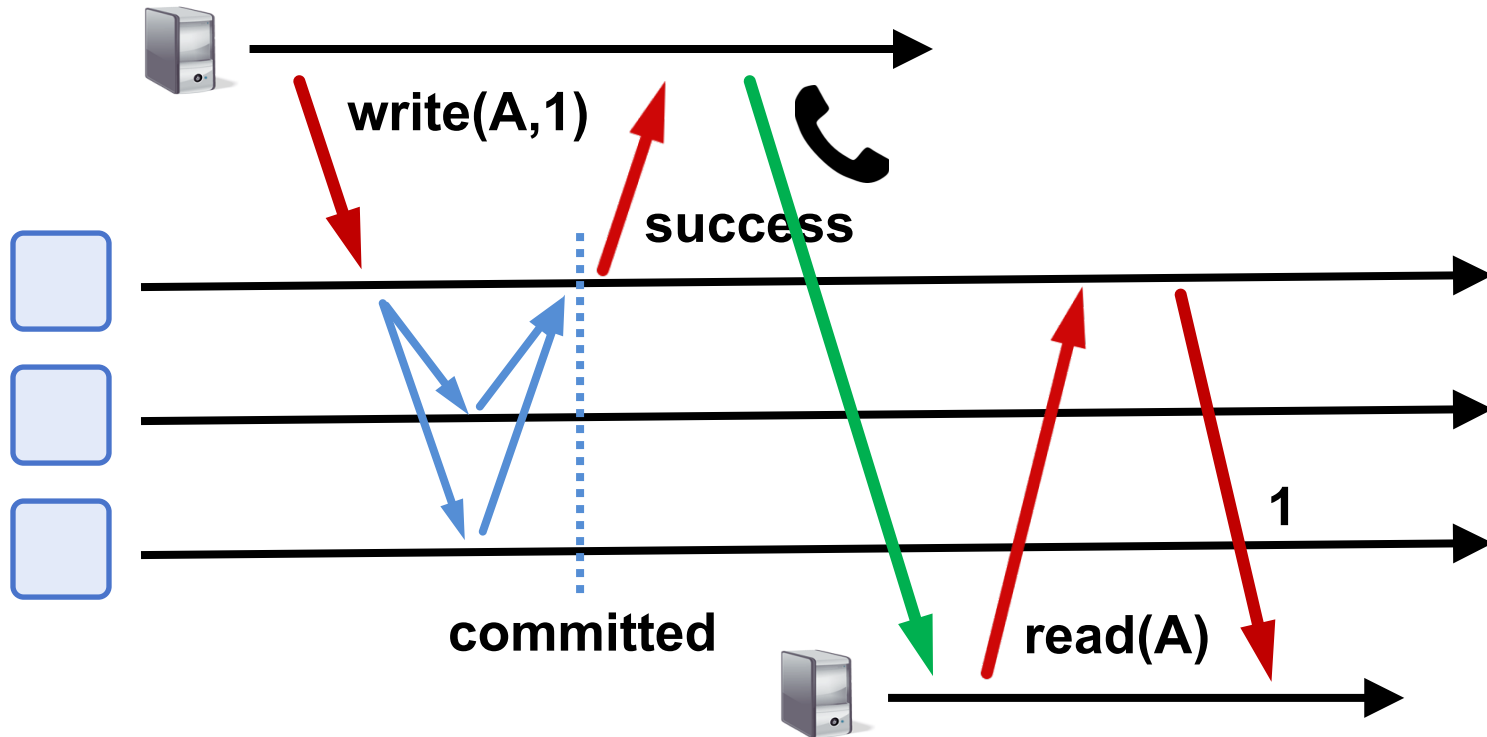
- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.

- Once read returns particular value, all later reads should return that value or value of later write.

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

write(A,1)

success

read(A)

0
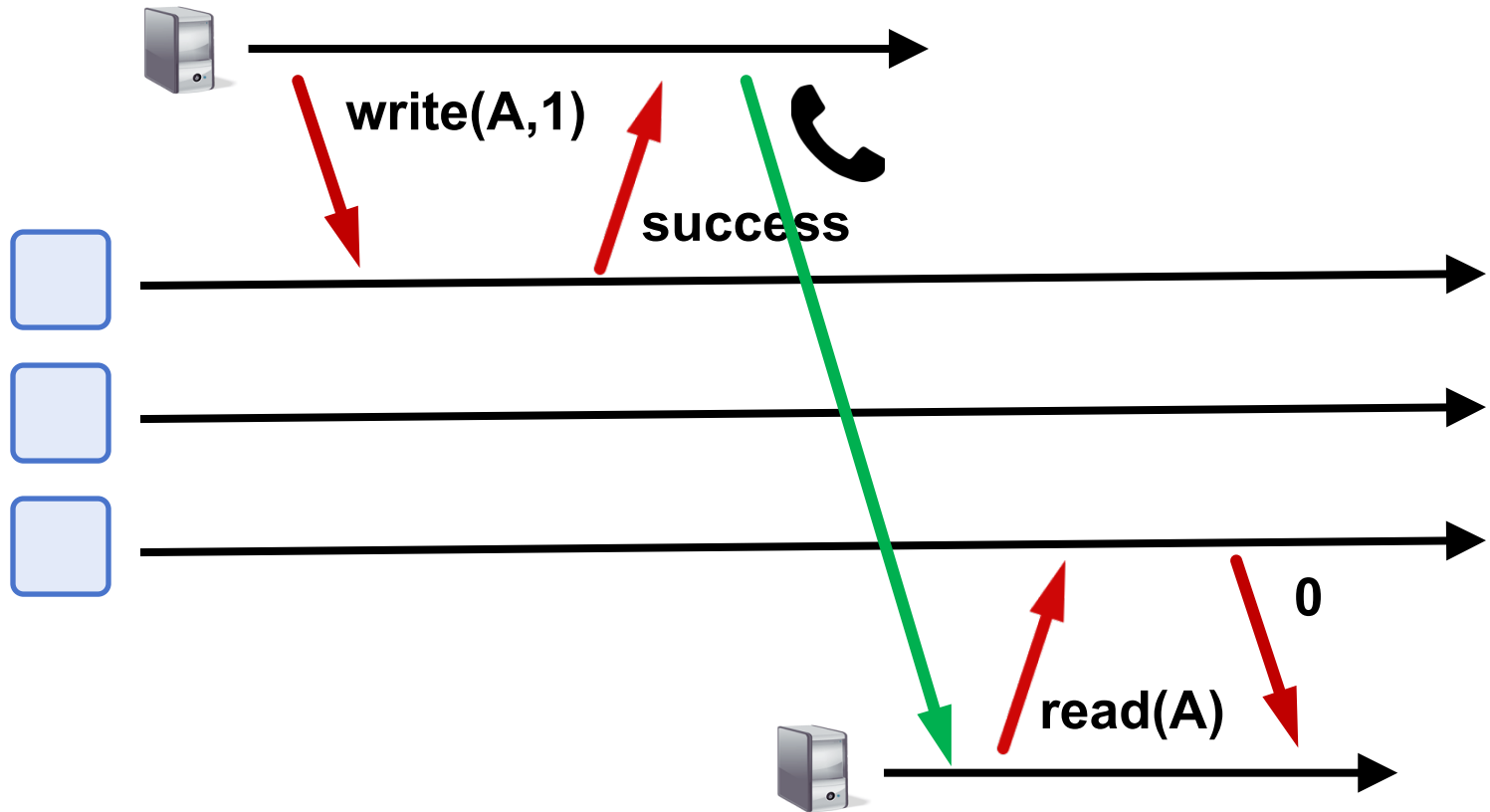
In example, system orders read(A) before write(A,1)

Valid Sequential Consistency?

P1: W(x)a

P2:   W(x)b

P3:   R(x)b   R(x)a

P4:   R(x)b R(x)a

✔

P1: W(x)a

P2:   W(x)b

P3:   R(x)b   R(x)a
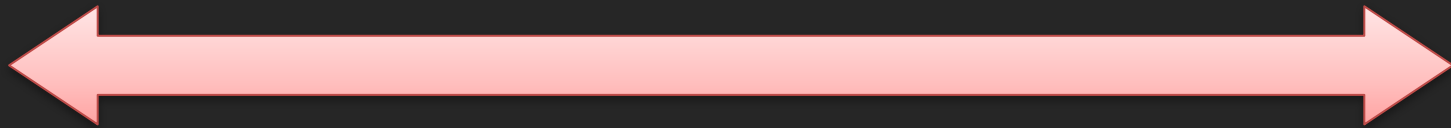
P4:   R(x)a R(x)b

✘

- 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?
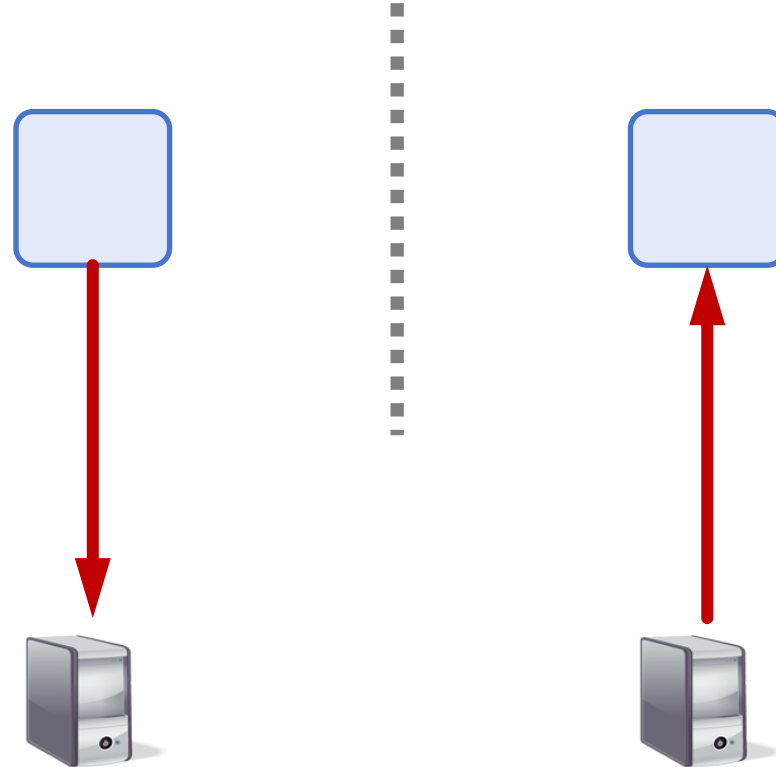
2PC / Consensus   **Eventual consistency**

Paxos / Raft   **Dynamo**
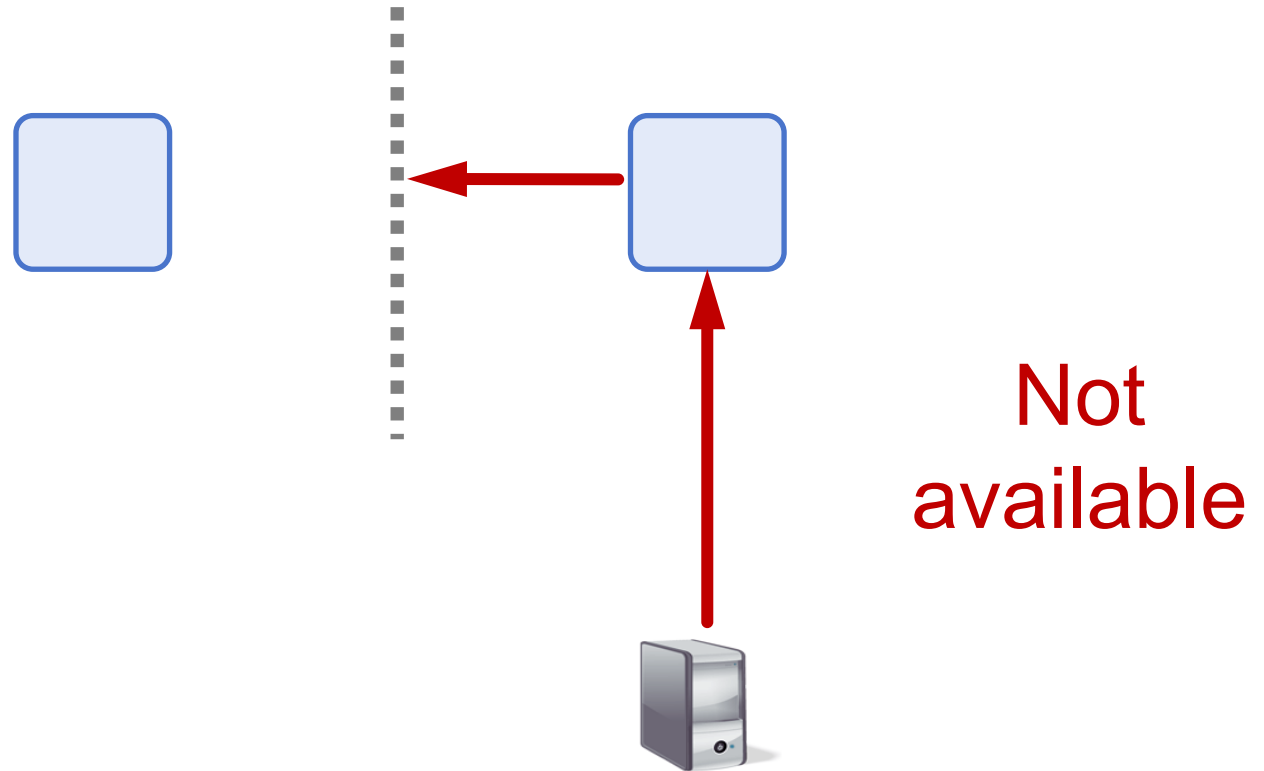
"CAP" Conjection for Distributed Systems

- 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

- Popular interpretation: 2-out-of-3

  - Consistency (Linearizability)

  - Availability

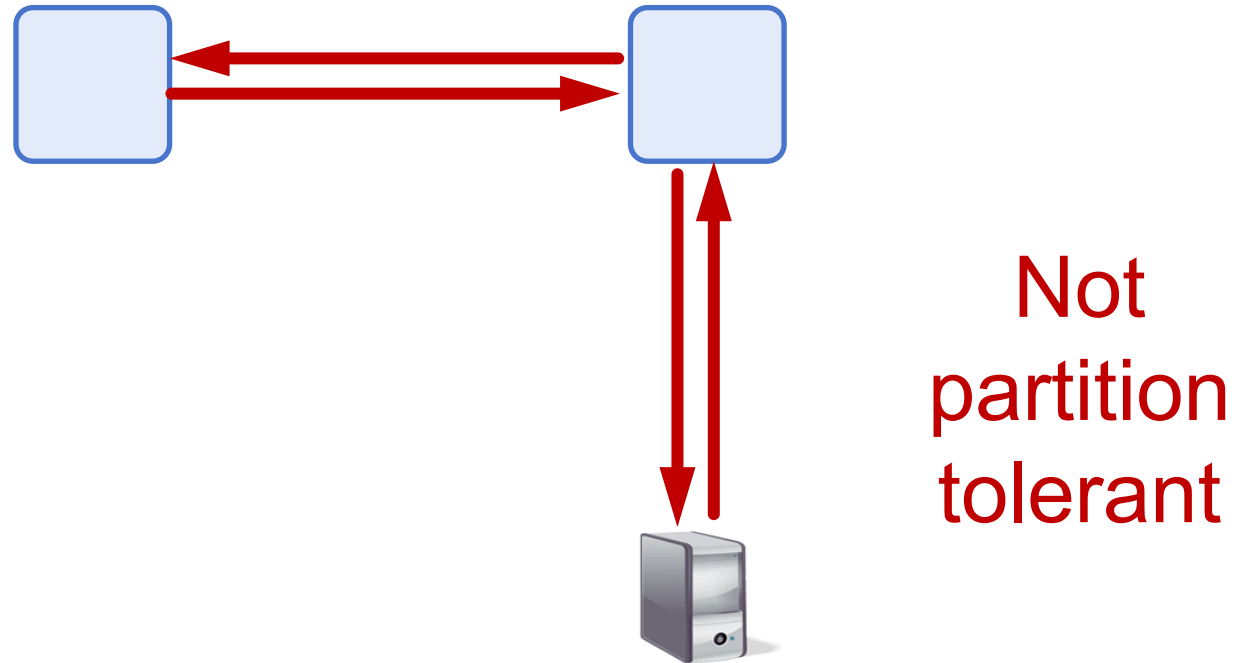 - Partition Tolerance: Arbitrary crash/network failures

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

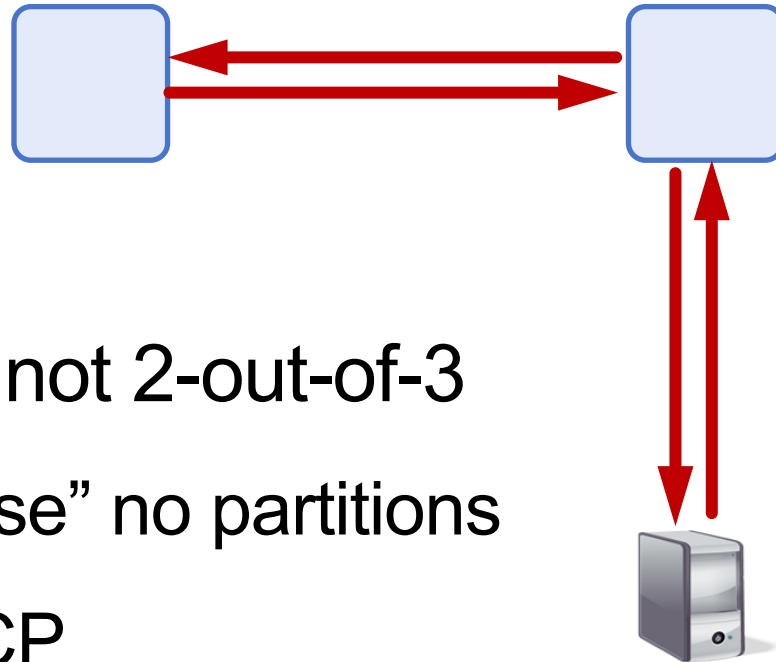Gilbert, Seth, and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." ACM SIGACT News 33.2 (2002): 51-59.

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
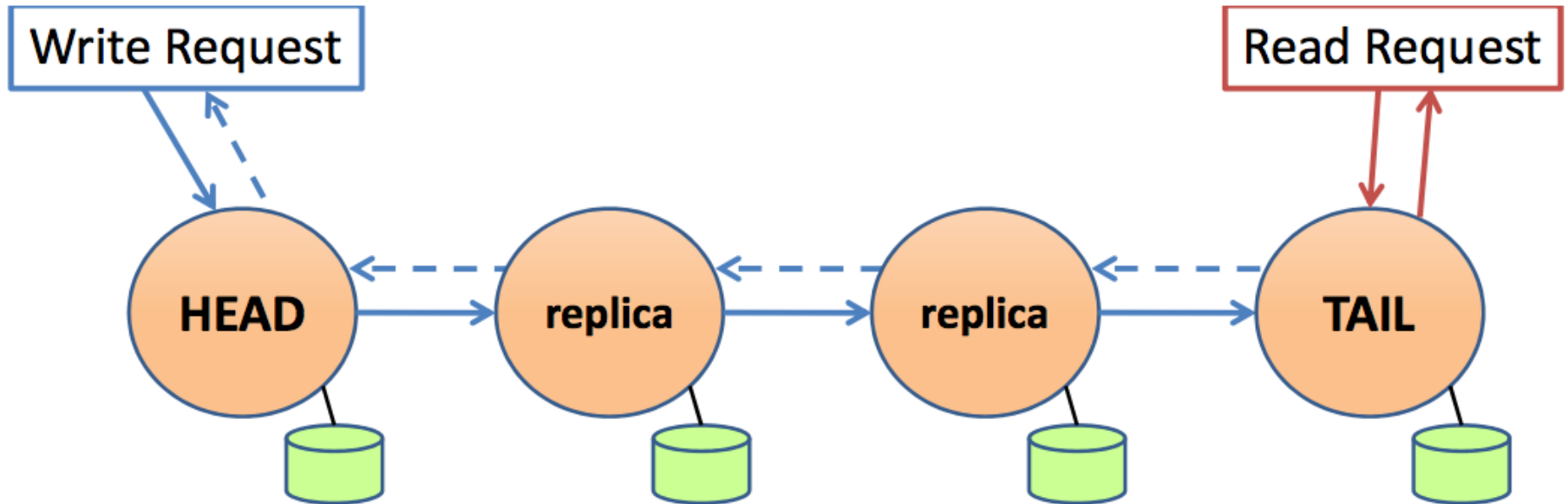
More tradeoffs L vs. C

- Low-latency: Speak to fewer than quorum of nodes?

  - 2PC: write N, read 1

  - RAFT: write $\lfloor N/2 \rfloor + 1$, read $\lfloor N/2 \rfloor + 1$

  - 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?

- 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
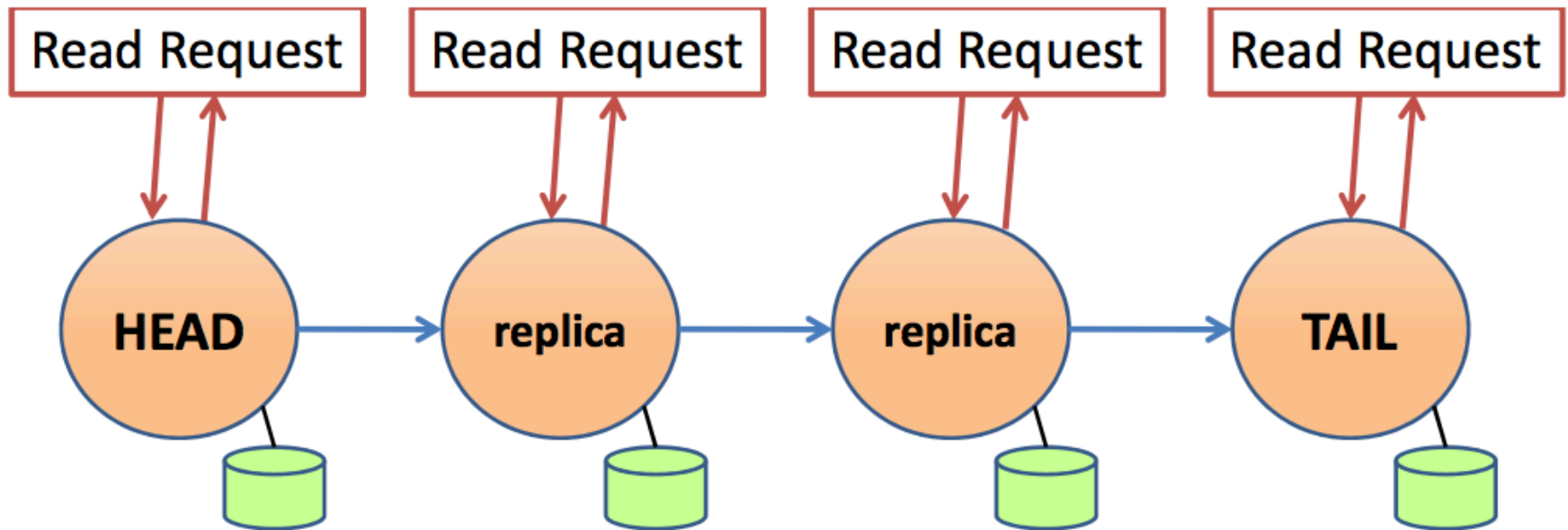
More linearizable replication algorithms
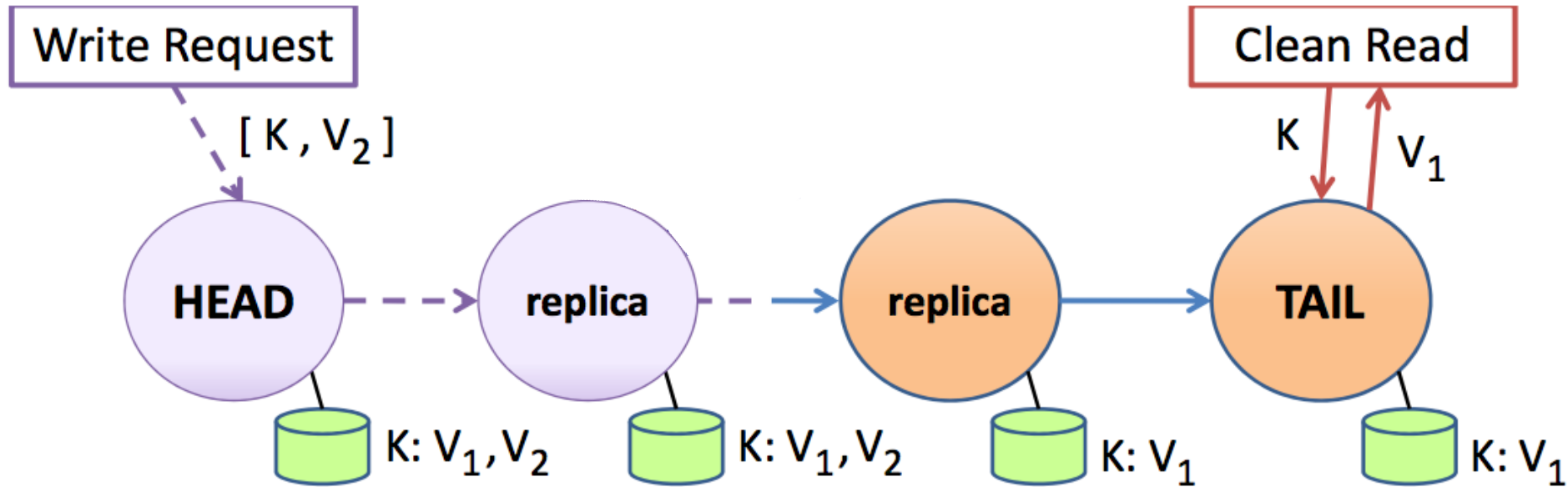
Chain replication

- Writes to head, which orders all writes

- When write reaches tail, implicitly committed rest of chain

- Reads to tail, which orders reads w.r.t. committed writes

Chain replication for read-heavy (CRAQ)

- Goal: If all replicas have same version, read from any one

- Challenge: They need to *know* they have correct version

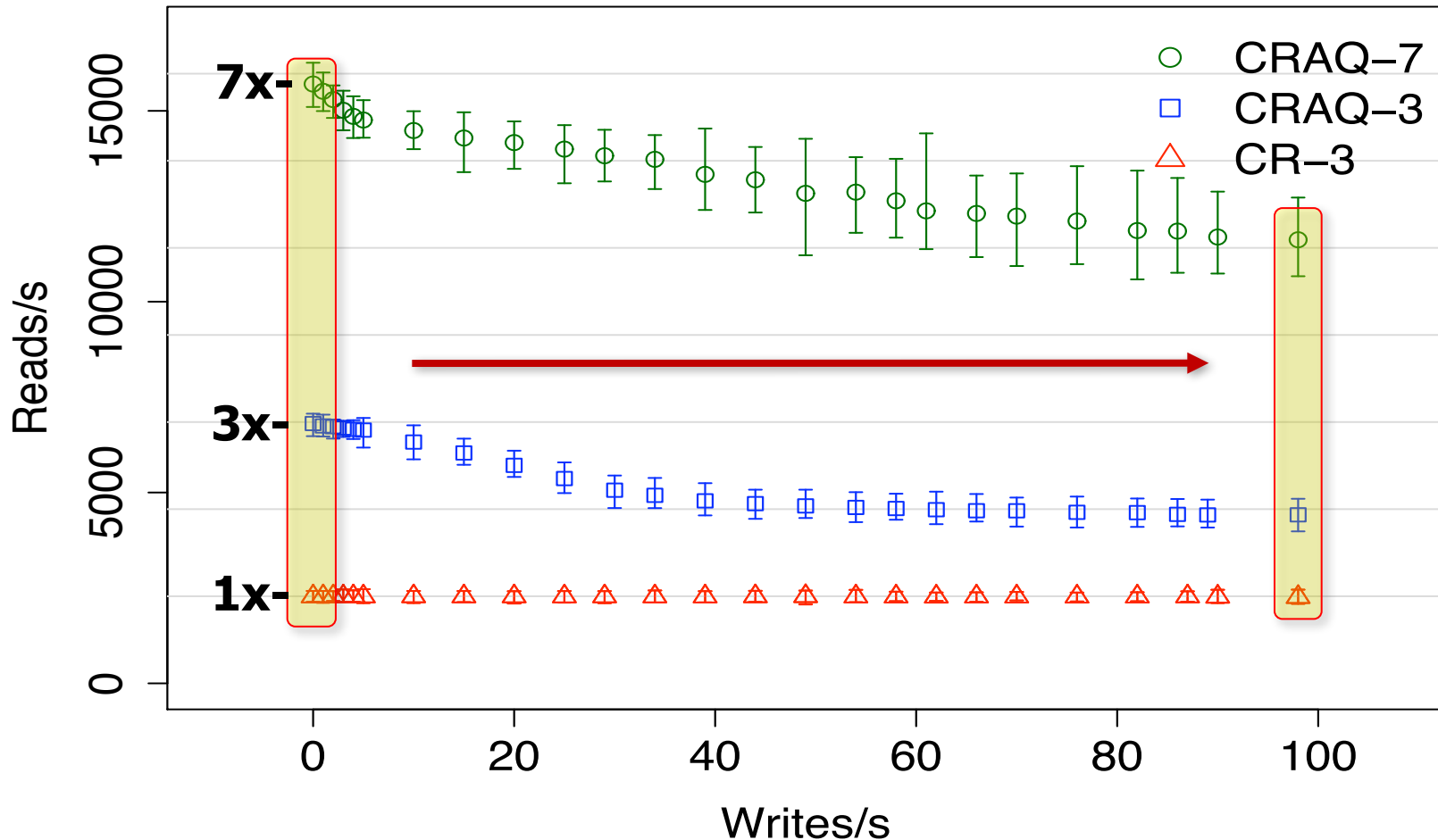# Chain replication for read-heavy (CRAQ)

- Replicas maintain multiple versions of objects while "dirty", i.e., contain uncommitted writes

- Commitment sent "up" chain after reaches tail

Chain replication for read-heavy (CRAQ)
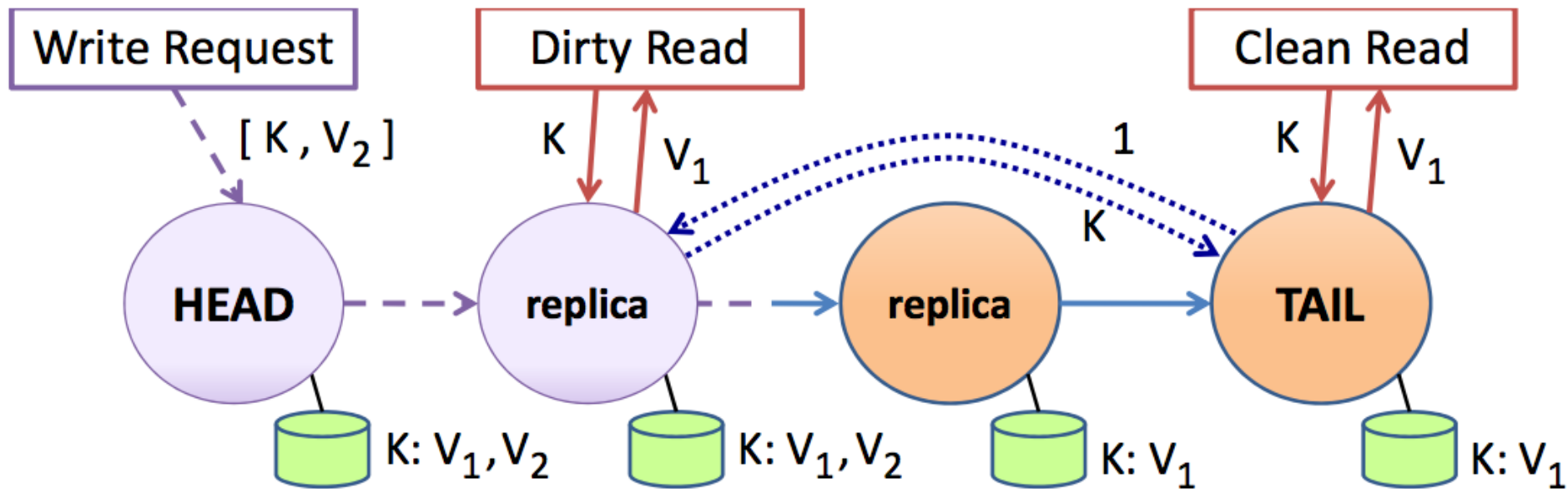
- Read to dirty object must check with tail for proper version

- This orders read with respect to global order, regardless of replica that handles

Performance: CR vs. CRAQ

R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. OSDI 2004.

J. Terrace and M. Freedman. Object Storage on CRAQ: High-throughput chain replication for read-mostly workloads. USENIX ATC 2009.

Wednesday lecture

Causal Consistency