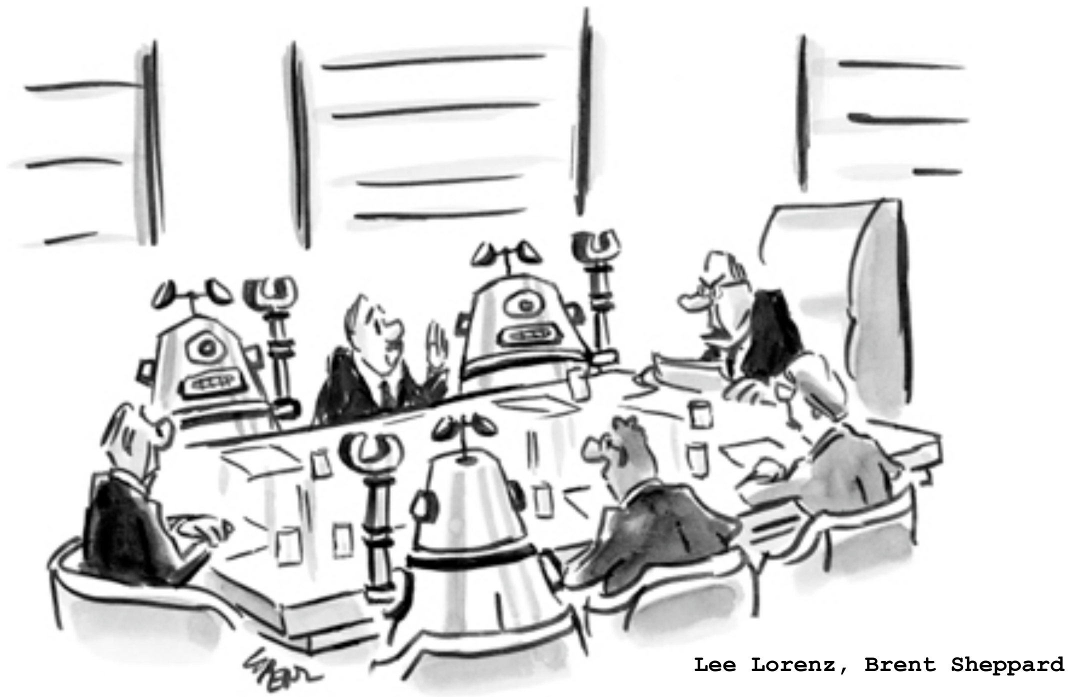


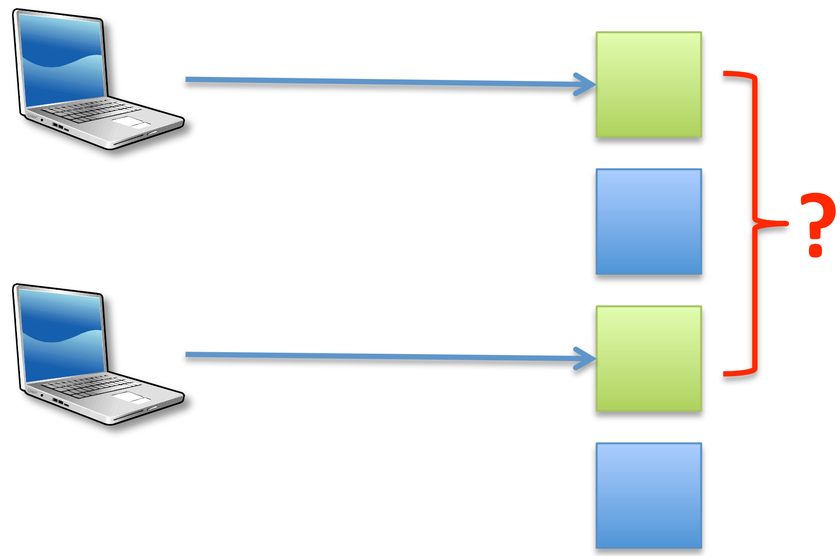
Jenkins, if I want another yes-man, I'll build one!

Strong Consistency and Agreement

COS 461: Computer Networks
Spring 2011

Mike Freedman
http://www.cs.princeton.edu/courses/archive/spring11/cos461/

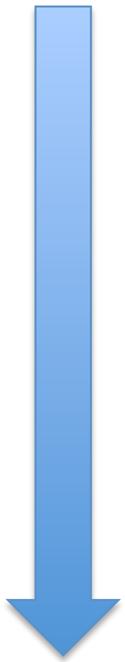
What consistency do clients see?



- Distributed stores may store data on multiple servers
  - Replication provides fault-tolerance if servers fail
  - Allowing clients to access different servers potentially increasing scalability (max throughput)
  - Does replication necessitate inconsistencies? Harder to program, reason about, confusing for clients, …

Consistency models

- Strict
- Strong (Linearizability)
- Sequential
- Causal
- Eventual

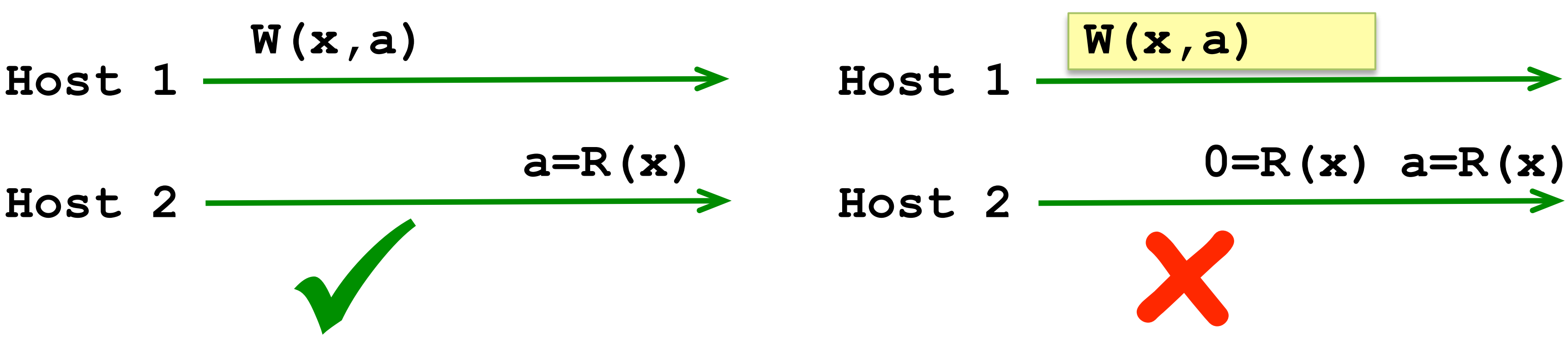


Weaker Consistency Models

These models describes when and how different nodes in a distributed system / network view the order of messages / operations

Strict Consistency

- Strongest consistency model we'll consider
  - Any read on a data item X returns value corresponding to result of the most recent write on X
- Need an absolute global time
  - "Most recent" needs to be unambiguous
  - Corresponds to when operation was issued
  - Impossible to implement in practice on multiprocessors



Sequential Consistency

- Definition:

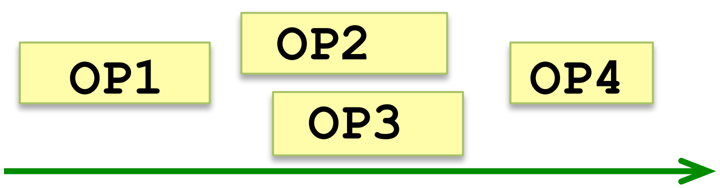
  All (read and write) operations on data store were executed in *some* sequential order, and the operations of each individual process appear in this sequence

- Definition: When processes are running concurrently:
  - Interleaving of read and write operations is acceptable, but all processes see the same interleaving of operations

- Difference from strict consistency
  - No reference to the most recent time
  - Absolute global time does not play a role

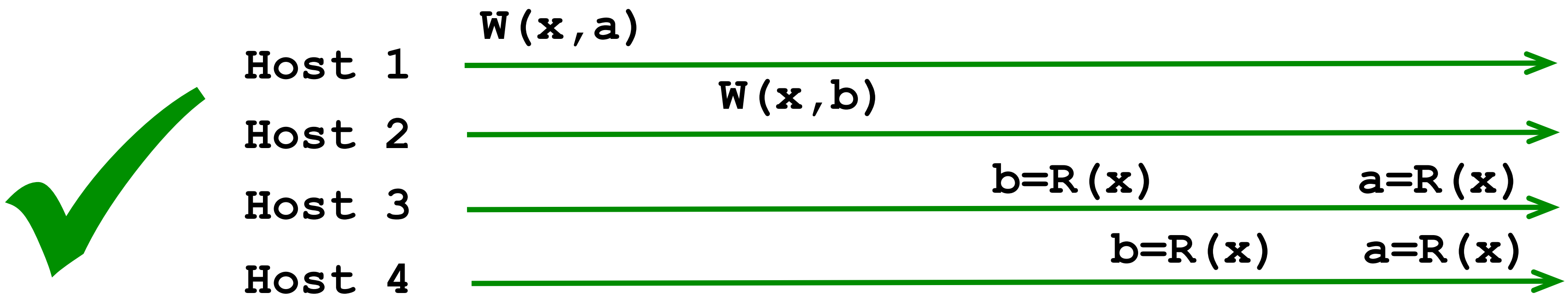
Implementing Sequential Consistency

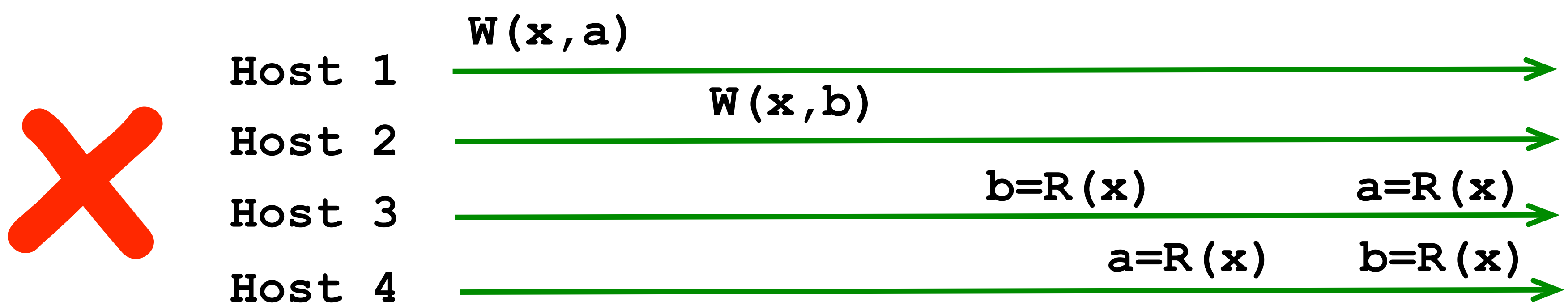
- Nodes use vector clocks to determine if two events had distinct happens-before relationship
  - If timestamp (a) < timestamp (b) $\Rightarrow$ a $\rightarrow$ b
- If ops are concurrent (\exists i,j, a[i] < b[i] and a[j] > b[j])
  - Hosts can order ops a, b arbitrarily but consistently



Valid:		**Valid**		**Invalid**	
Host 1:	OP 1, 2, 3, 4	Host 1:	OP 1, 3, 2, 4	Host 1:	OP 1, 2, 3, 4
Host 2:	OP 1, 2, 3, 4	Host 2:	OP 1, 3, 2, 4	Host 2:	OP 1, 3, 2, 4

Examples: Sequential Consistency?





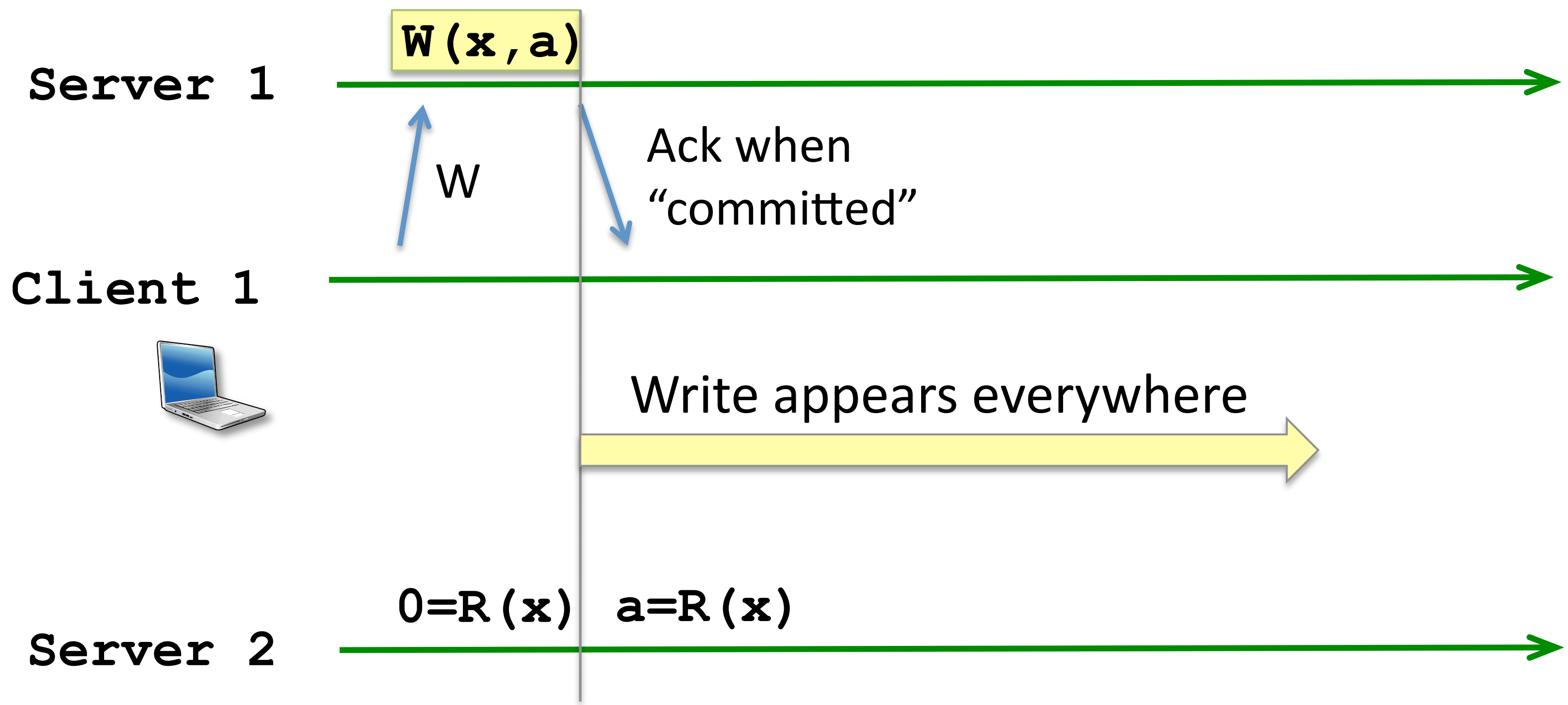
(but is valid causal consistency)

- Sequential consistency is what allows databases to reorder "isolated" (i.e. non causal) queries
- But all DB replicas see same trace, a.k.a. "serialization"

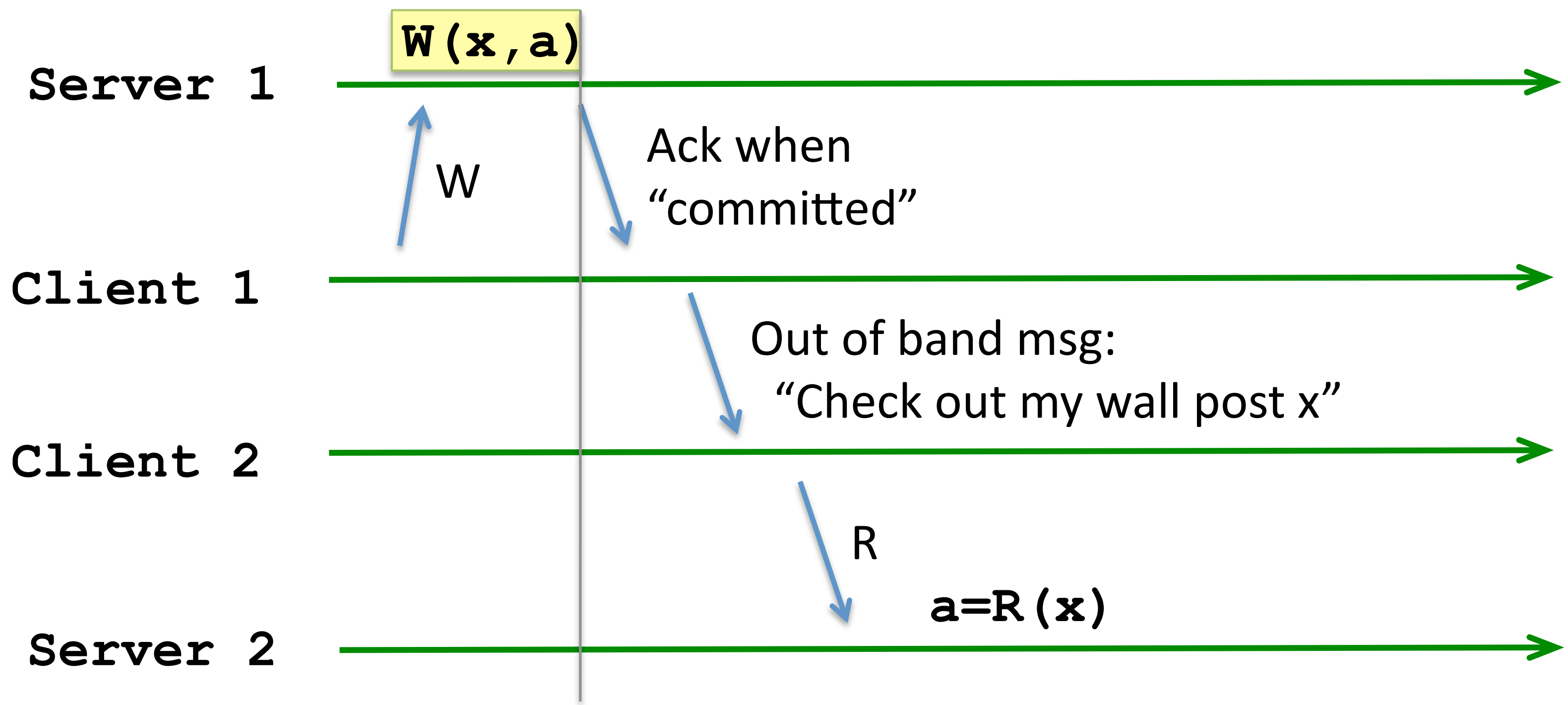
Strong Consistency / Linearizability

- Strict > Linearizability > Sequential
- All operations (OP = read, write) receive a global time-stamp using a synchronized clock sometime during their execution
- Linearizability:
  - Requirements for sequential consistency, plus
  - If $ts_{op1}(x) < ts_{op2}(y)$, then OP1(x) should precede OP2(y) in the sequence
  - "Real-time requirement": Operation "appears" as if it showed up everywhere at same time

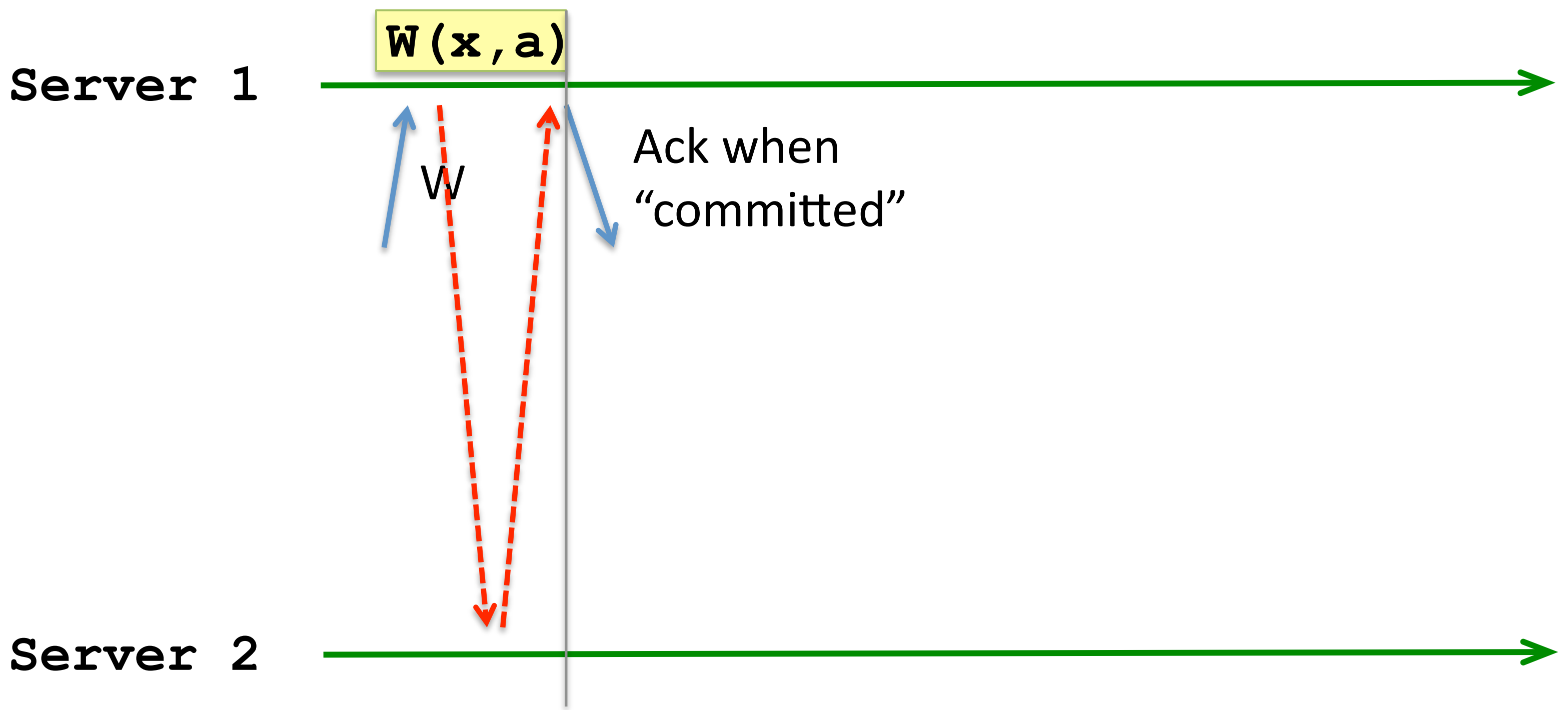
Linearizability



Implications of Linearizability

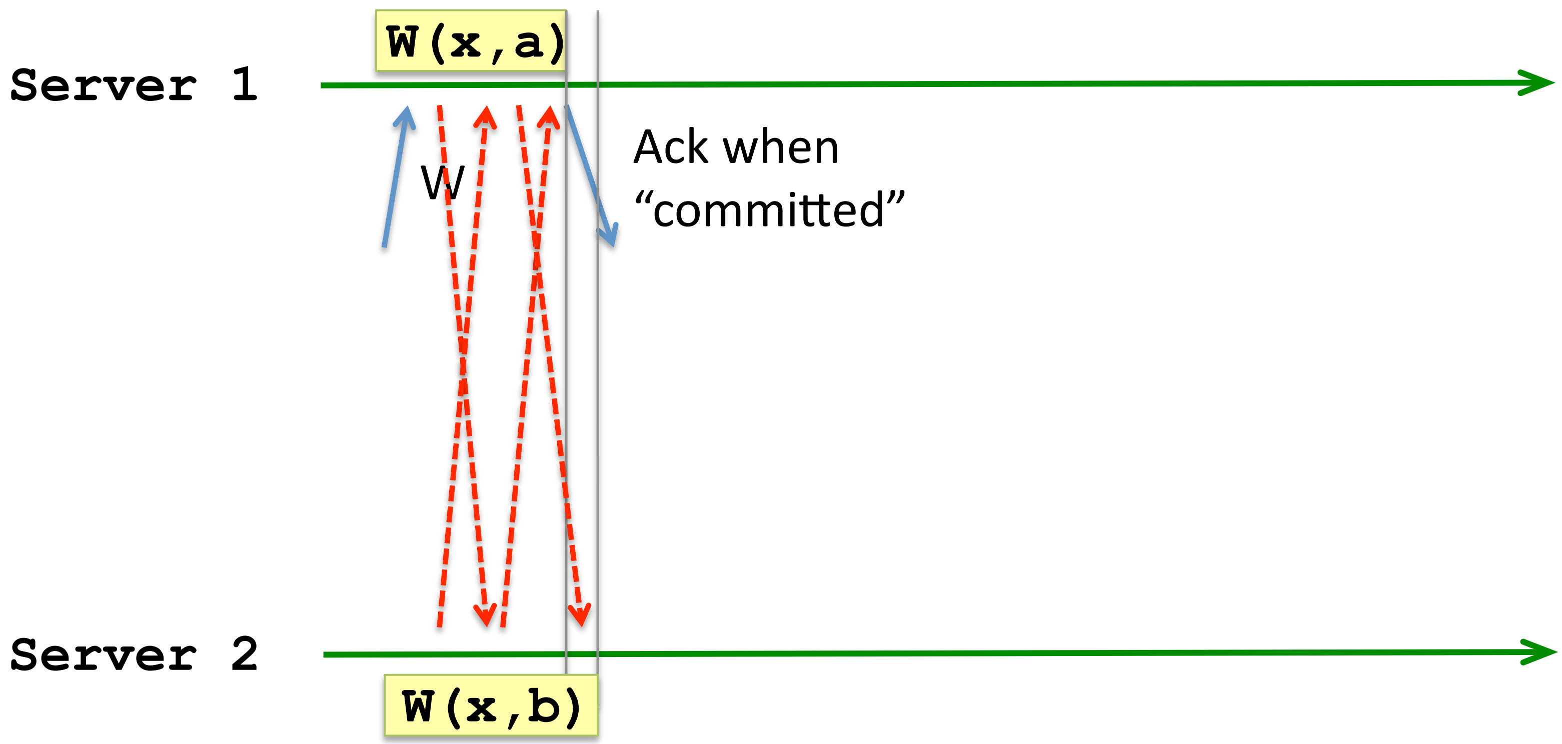


Implementing Linearizability



- If OP must appear everywhere after some time (the conceptual "timestamp" requirement) ⇒ "all" locations must locally commit op before server acknowledges op as committed
- Implication: Linearizability and "low" latency mutually exclusive
  - e.g., might involve wide-area writes

Implementing Linearizability



- Algorithm not quite as simple as just copying to other server before replying with ACK: Recall that all must agree on ordering
  - Both see either a → b or b → a , but not mixed
  - Both a and b appear everywhere as soon as committed

Consistency

+

Availability

Data replication with linearizability

- Master replica model
  - All ops (& ordering) happens at single master node
  - Master replicates data to secondary

- Multi-master model
  - Read/write anywhere
  - Replicas order and replicate content before returning

Single-master: Two-phase commit

• Marriage ceremony	Do you? I do. Do you? I do.	**Prepare**
	I now pronounce…	**Commit**
• Theater	Ready on the set? Ready!	
	Action!	
• Contract law	Offer Signature	
	Deal / lawsuit	

Two-phase commit (2PC) protocol

What about failures?

- If one or more acceptor (≤ F) fails:
  - Can still ensure linearizability if $|R| + |W| > N + F$
  - "read" and "write" quorums of acceptors overlap in at least 1 non-failed node
- If the leader fails?
  - Lose availability: system not longer "live"
- Pick a new leader?
  - Need to make sure everybody agrees on leader!
  - Need to make sure that "group" is known

Consensus / Agreement Problem

- Goal: N processes want to agree on a value
- Desired properties:
  - Correctness (safety):
    - All N nodes agree on the same value
    - The agreed value has been proposed by some node
  - Fault-tolerance:
    - If ≤ F faults in a window, consensus reached *eventually*
    - Liveness not guaranteed: If > F failures, no consensus
    - Given goal of F, what is N?
      - "Crash" faults need 2F+1 processes
      - "Malicious" faults (called Byzantine) need 3F+1 processes

Paxos Algorithm

- Setup
  - Each node runs *proposer (leader)*, *acceptor,* and *learner*
- Basic approach
  - One or more node decides to act like a leader
  - Leader proposes value, solicits acceptance from acceptors
  - Leader announces chosen value to learners

Why is agreement hard?

(Don't we learn that in kindergarten?)

- What if >1 nodes think they're leaders simultaneously?
- What if there is a network partition?
- What if a leader crashes in the middle of solicitation?
- What if a leader crashes after deciding but before broadcasting commit?
- What if the new leader proposes different values than already committed value?

Strawman solutions

- Designate a single node X as acceptor
  - Each *proposer* sends its value to X
  - X decides on one of the values, announces to all learners
  - Problem!
    - Failure of acceptor halts decision ⇒ need multiple acceptors
- Each proposer (leader) propose to all acceptors
  - Each acceptor accepts first proposal received, rejects rest
  - If leader receives ACKs from a majority, chooses its value
    - There is at most 1 majority, hence single value chosen
  - Leader sends chosen value to all learners
  - Problems!
    - With multiple simultaneous proposals, may be no majority
    - What if winning leader dies before sending chosen value?

Paxos' solution

- Each acceptor must be able to accept multiple proposals
- Order proposals by proposal #
  - If a proposal with value v is chosen, all higher proposals will also have value v
- Each node maintains:
  - t_a, v_a: highest proposal # accepted and its corresponding accepted value
  - t_{max}: highest proposal # seen
  - t_{my}: my proposal # in the current Paxos

Paxos (Three phases)

Phase 1 (Prepare)

- Node decides to become leader
  - Chooses $t_{my} > t_{max}$
  - Sends *<prepare,* t_{my}*>* to all nodes
- Acceptor upon receiving *<prep, t>*

  If $t < t_{max}$

  reply <prep-reject>

  Else

  $t_{max} = t$

  reply <prep-ok, t_a, v_a>

Phase 2 (Accept)

- If leader gets *<prep-ok, t, v>* from majority
  - If v == null, leader picks v_{my}. Else v_{my} = v.
  - Send *<accept,* t_{my}, v_{my}*>* to all nodes
- If leader fails to get majority, delay, restart
- Upon *<accept, t, v>*

  If $t < t_{max}$

  reply with <accept-reject>

  Else

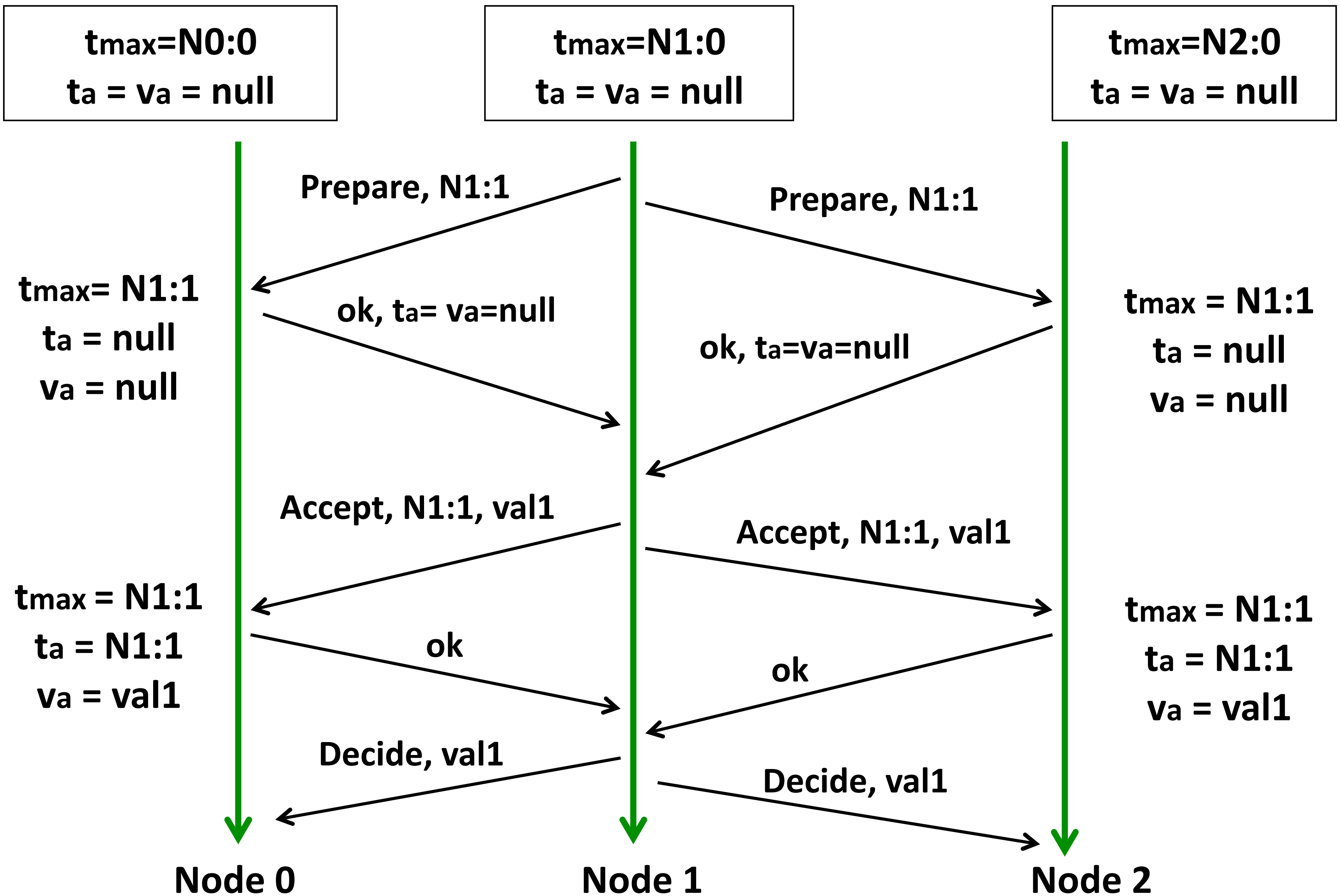
  $t_a = t$; $v_a = v$; $t_{max} = t$

  reply with <accept-ok>

Phase 3 (Decide)

- If leader gets *acc-ok* from majority
  - Send *<decide,* v_a*>* to all nodes
- If leader fails to get accept-ok from majority
  - Delay and restart

Paxos operation: an example



Combining Paxos and 2PC

- Use Paxos for view-change
  - If anybody notices current master unavailable, or one or more replicas unavailable
  - Propose view change Paxos to establish new group:
    - Value agreed upon = <2PC Master, {2PC Replicas} >
- Use 2PC for actual data
  - Writes go to master for two-phase commit
  - Reads go to acceptors and/or master
- Note: no liveness if can't communicate with majority of nodes from previous view

CAP Conjecture

- Systems can have two of:
  - C: Strong consistency
  - A: Availability
  - P: Tolerance to network partitions

…But not all three

- Two-phase commit: CA
- Paxos: CP
- Eventual consistency: AP