Strong Consistency

CS 475: Concurrent & Distributed Systems (Fall 2021)

Lecture 11

Yue Cheng

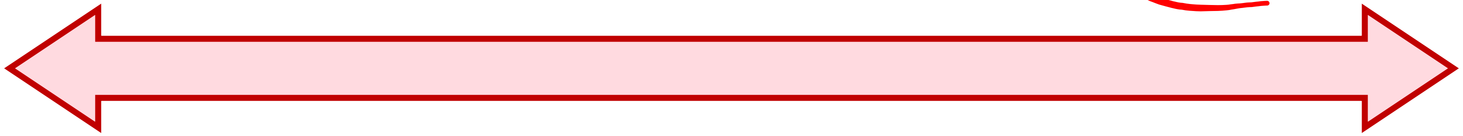
Some material taken/derived from:

- 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

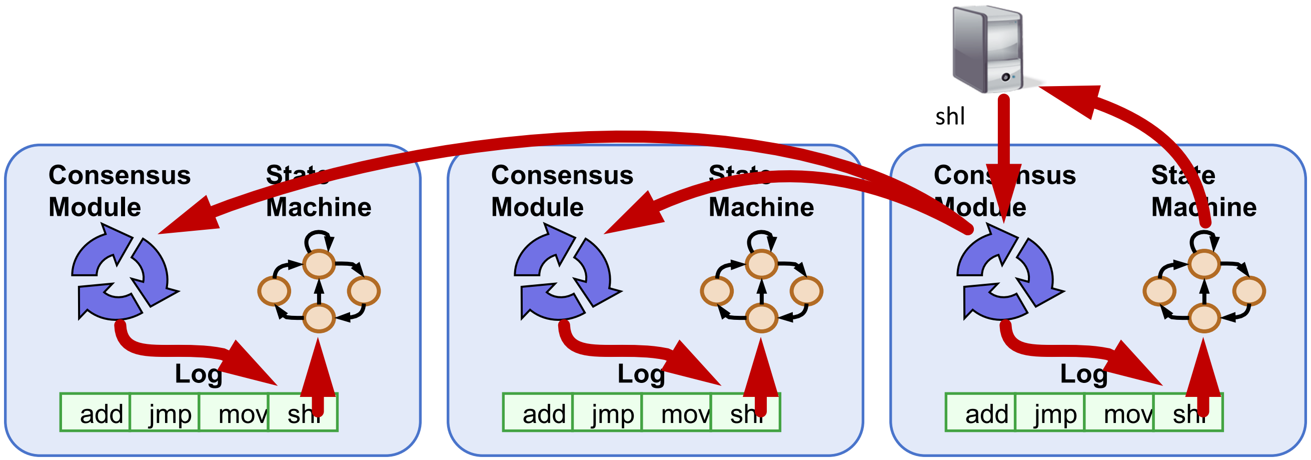


Paxos / Raft

Dynamo

Strong.

Consistency in Paxos/Raft

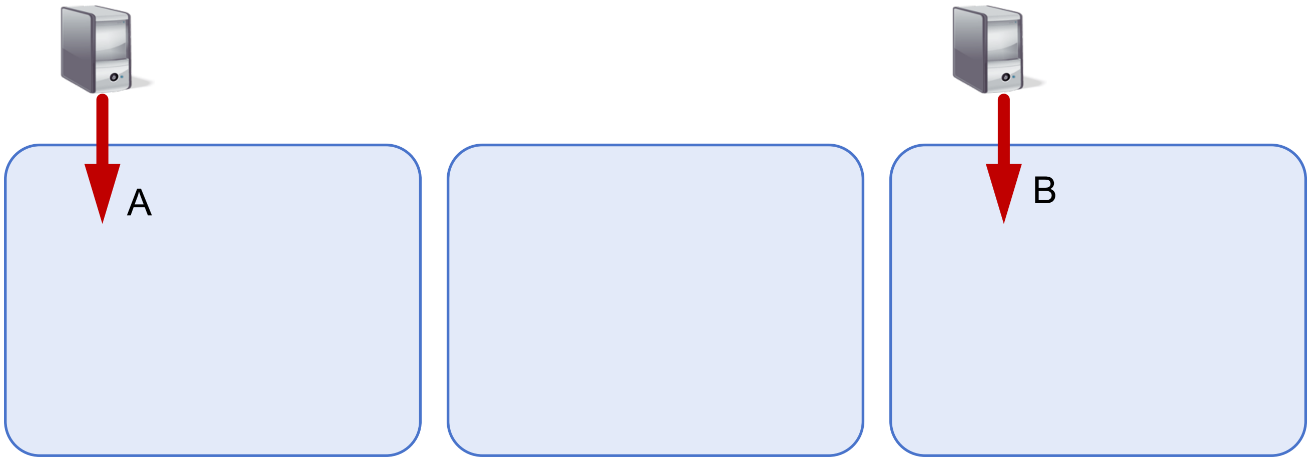


liveness

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

safety

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

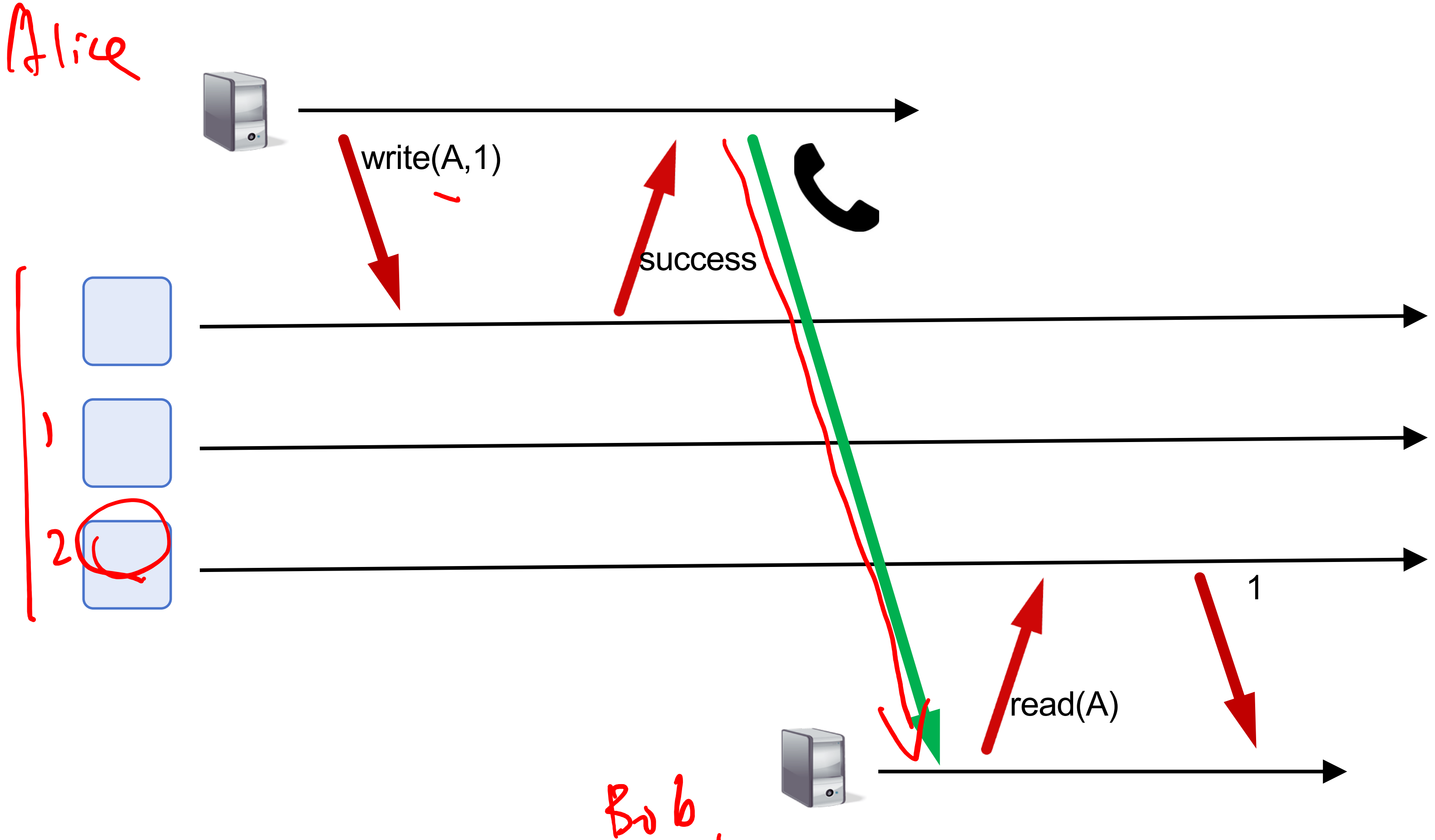
- Provide behavior of a single copy of object:
  - Read should return the most recent write
  - Subsequent reads should return same value, until next write

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

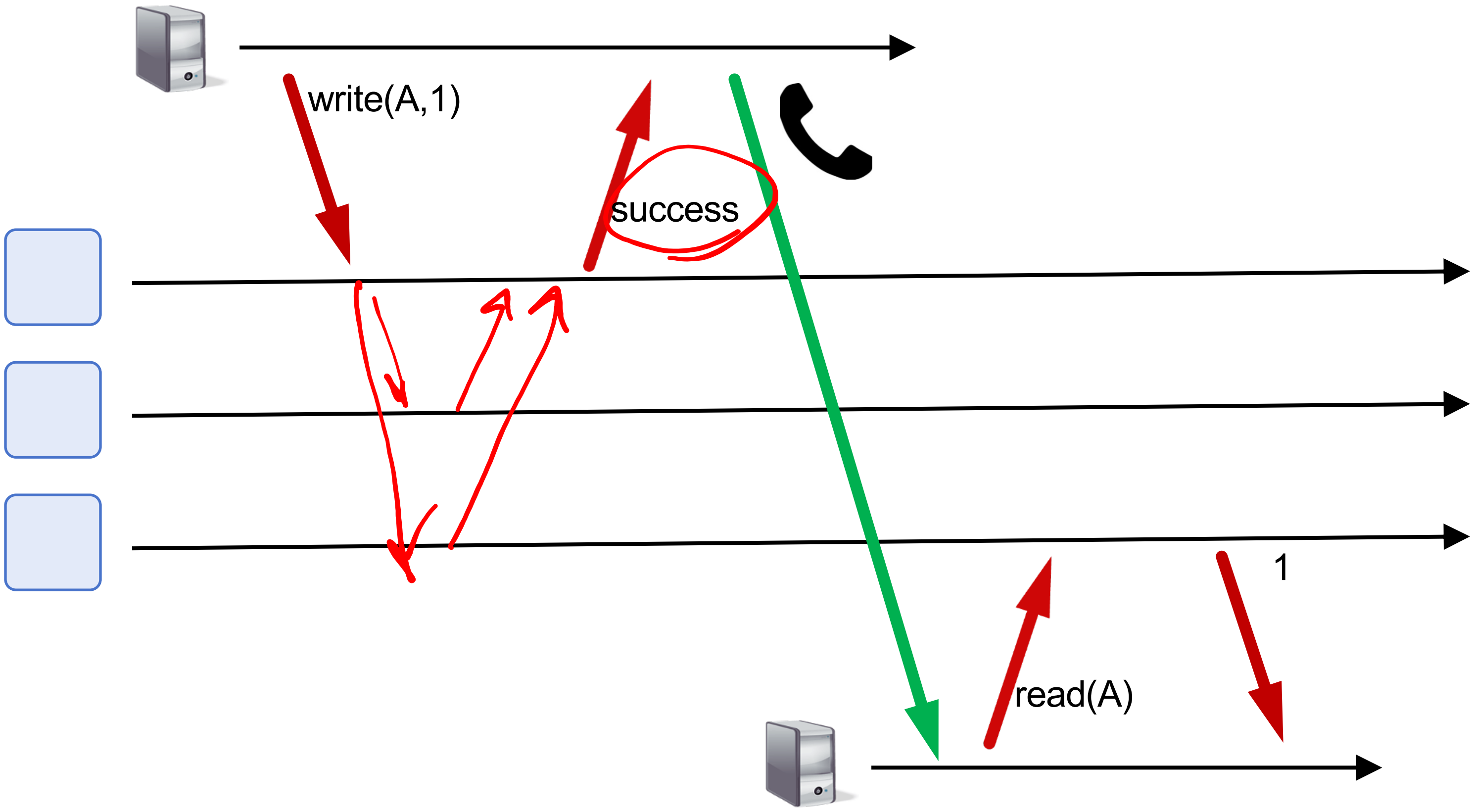
- Telephone intuition:
  1. Alice updates Facebook post
  2. Alice calls Bob on phone: “Check my Facebook post!”
  3. Bob read’s Alice’s wall, sees her post

Strong Consistency?



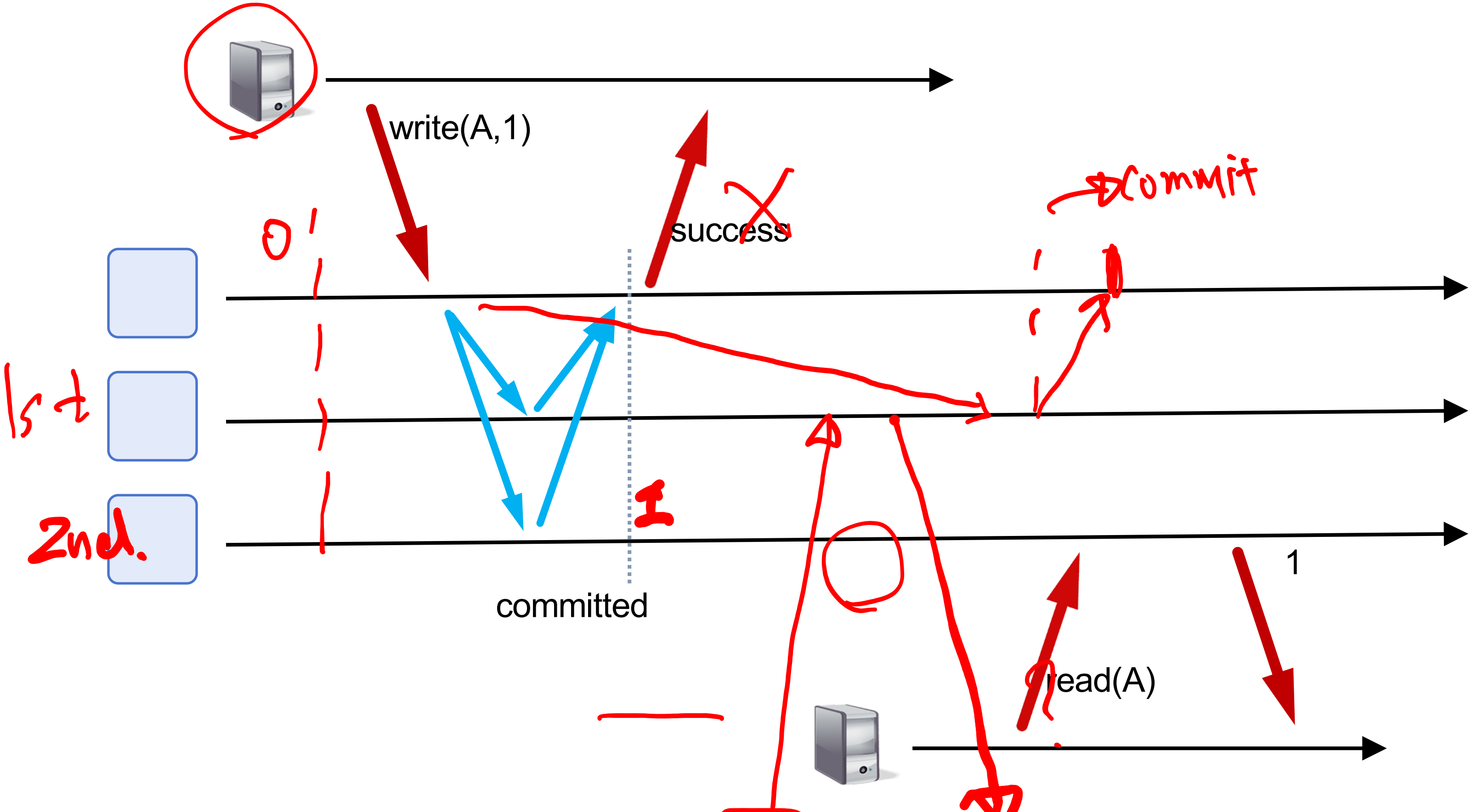
Phone call: Ensures *happens-before* relationship, even through "out-of-band" communication

Strong Consistency?



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

Strong Consistency? This is buggy!

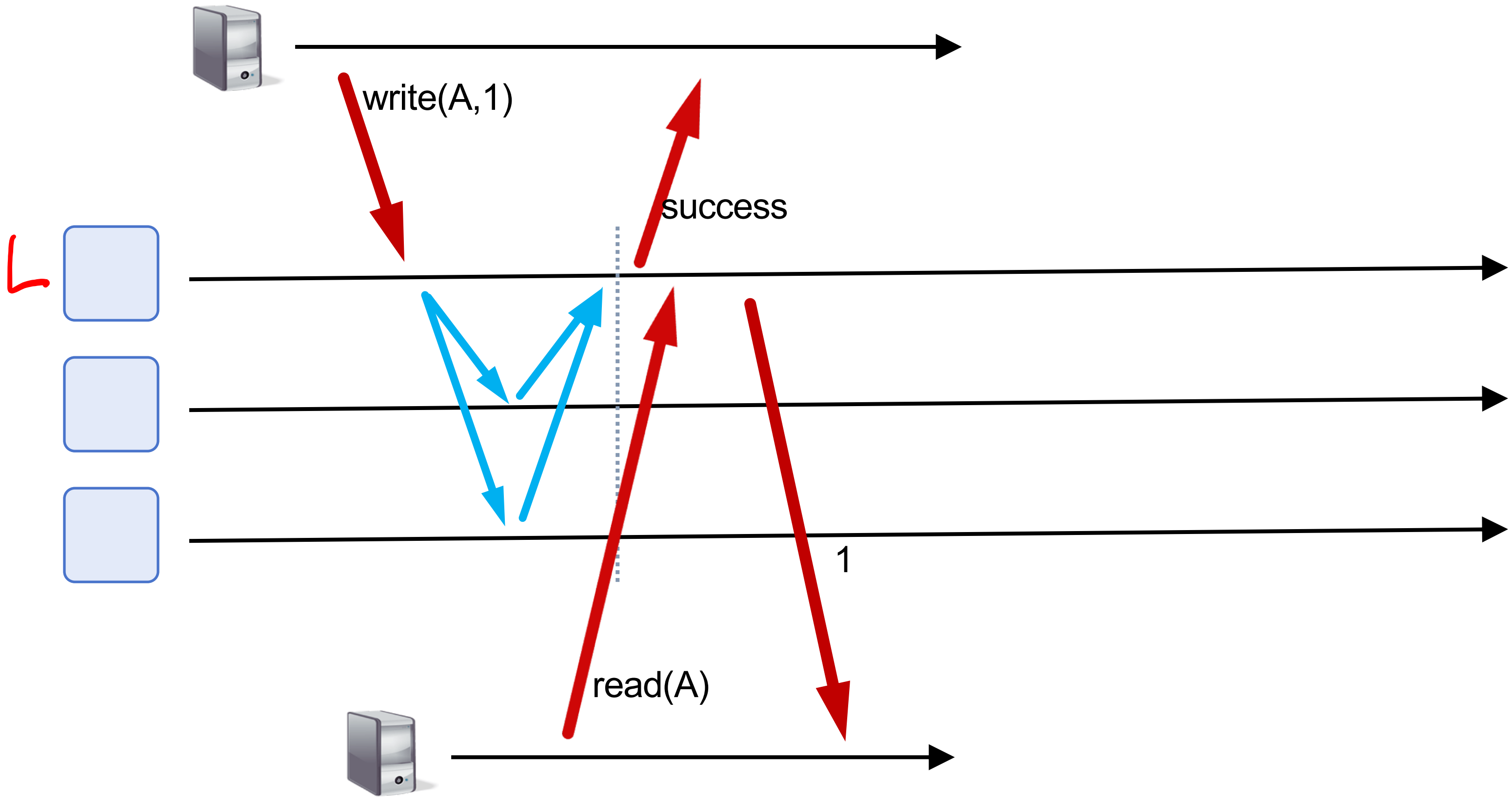


- Isn't sufficient to return value of third node:
  It doesn't know precisely when op is "globally" committed

1 ≠ 0

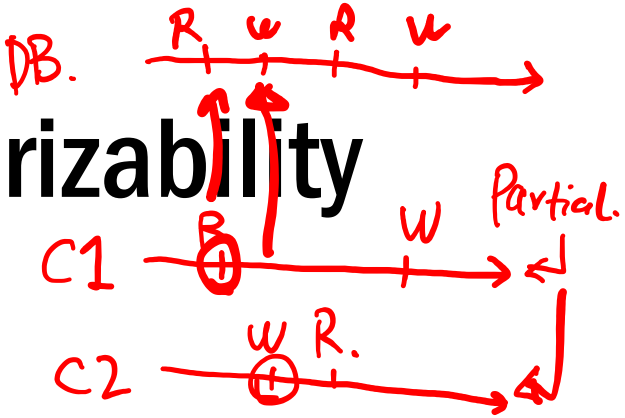
- Instead: Need to actually *order* read operation

Strong Consistency!



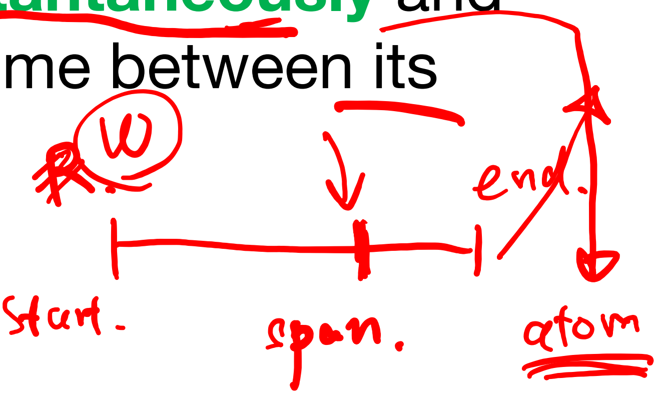
Order all operations via (1) leader, (2) consensus

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 $\underline{ts_{op1}(x)} < \underline{ts_{op2}}(y)$, OP1(x) precedes OP2(y) in sequence

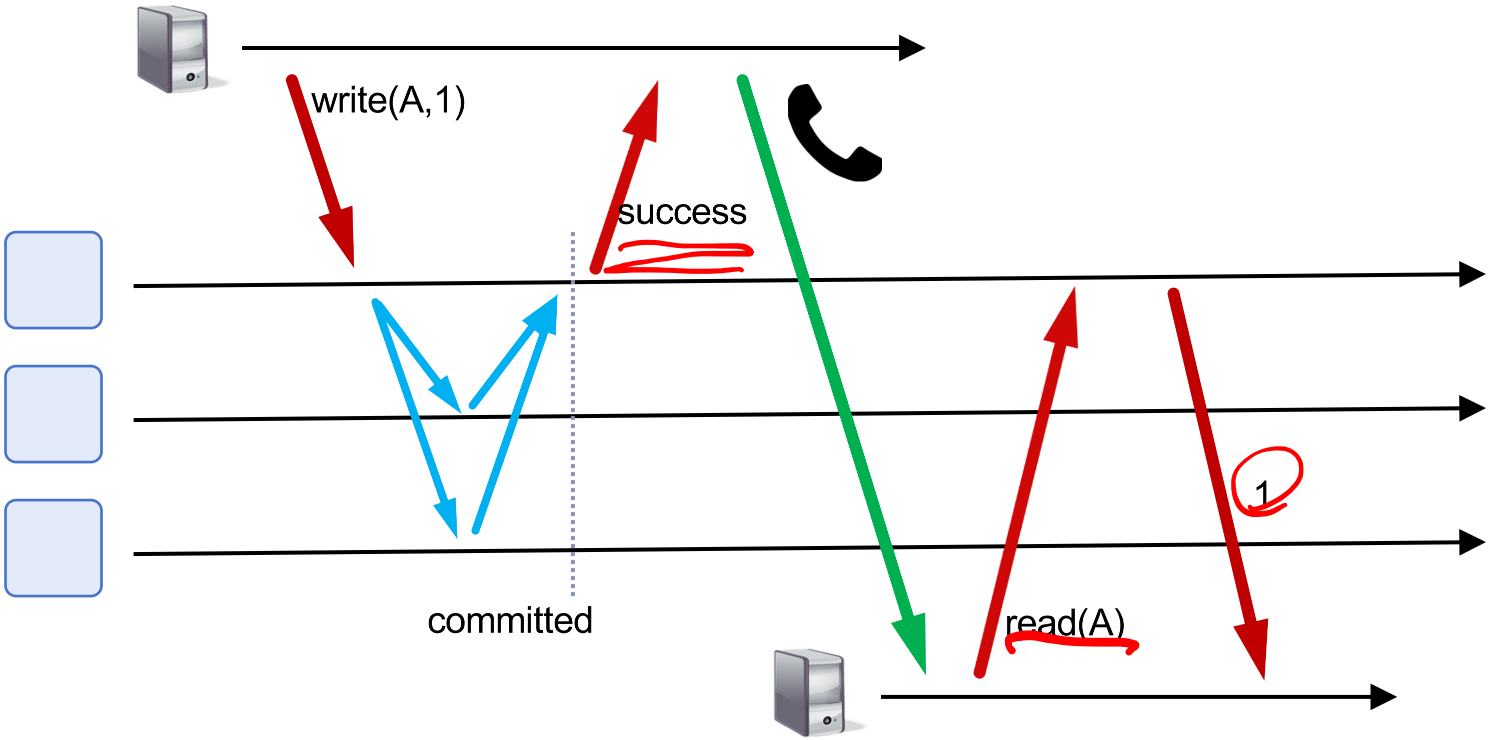
Strong consistency = linearizability

- 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.

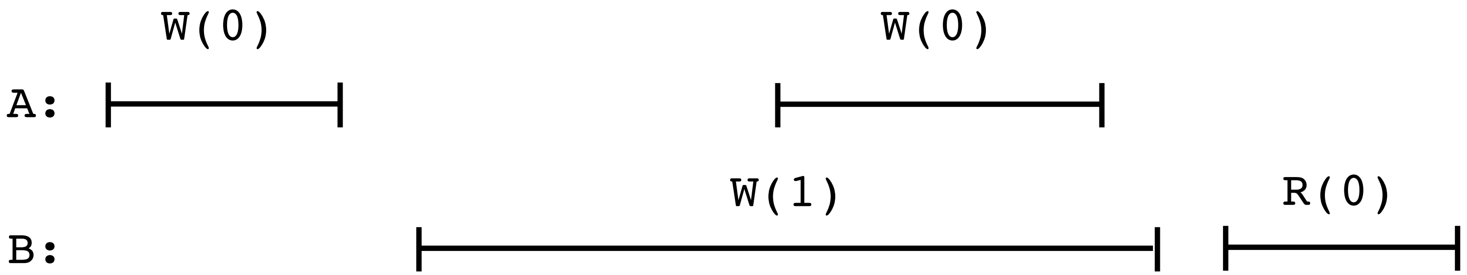
- Once read returns particular value, all later reads should return that value or value of later write.

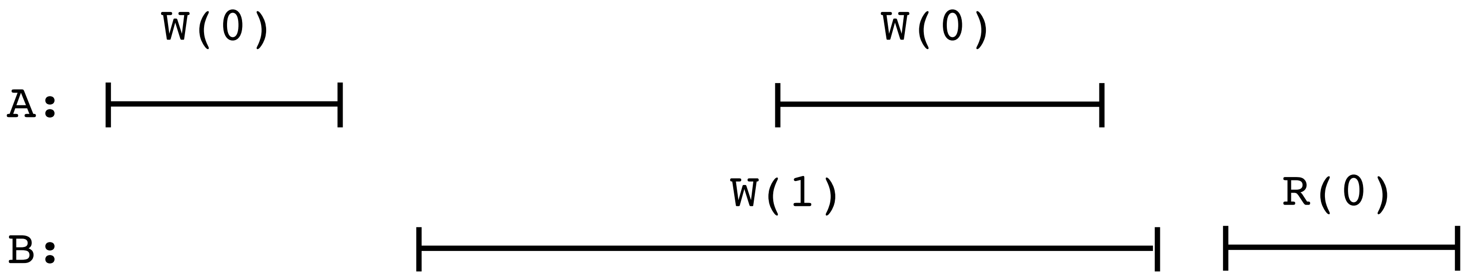
Intuition: Real-time ordering

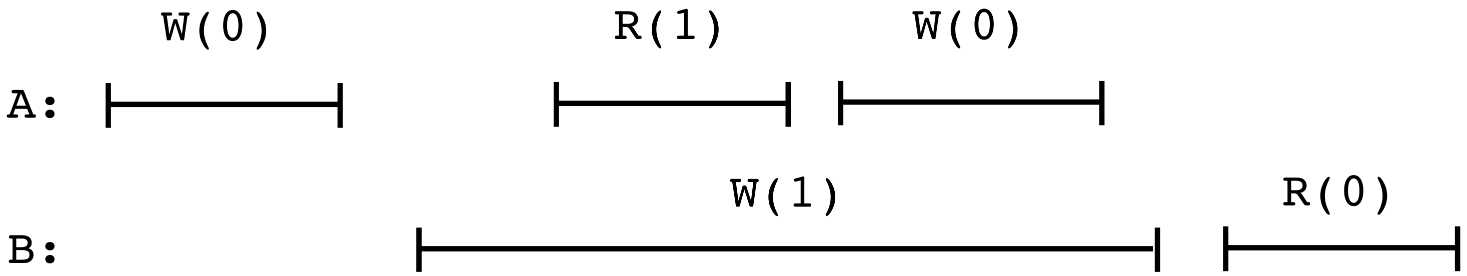


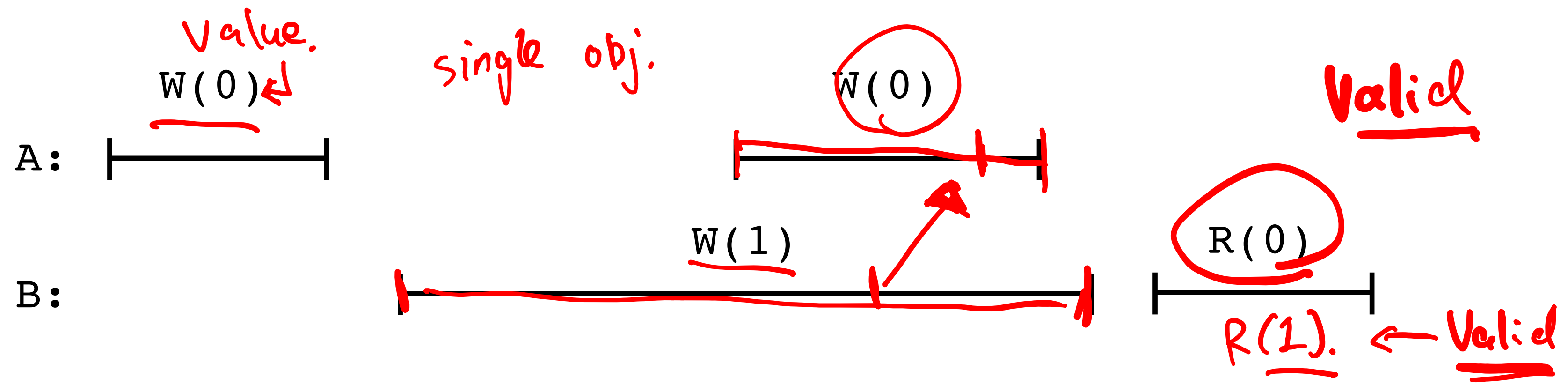
- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

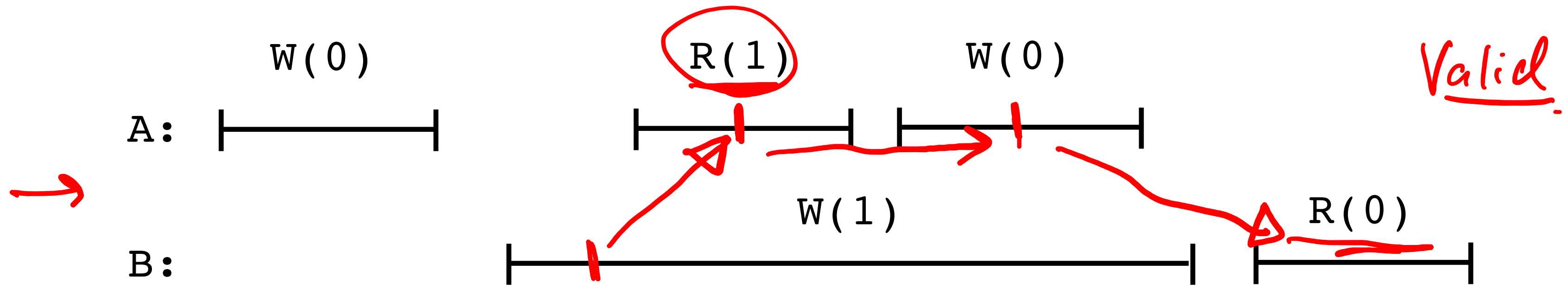
Real-time ordering examples

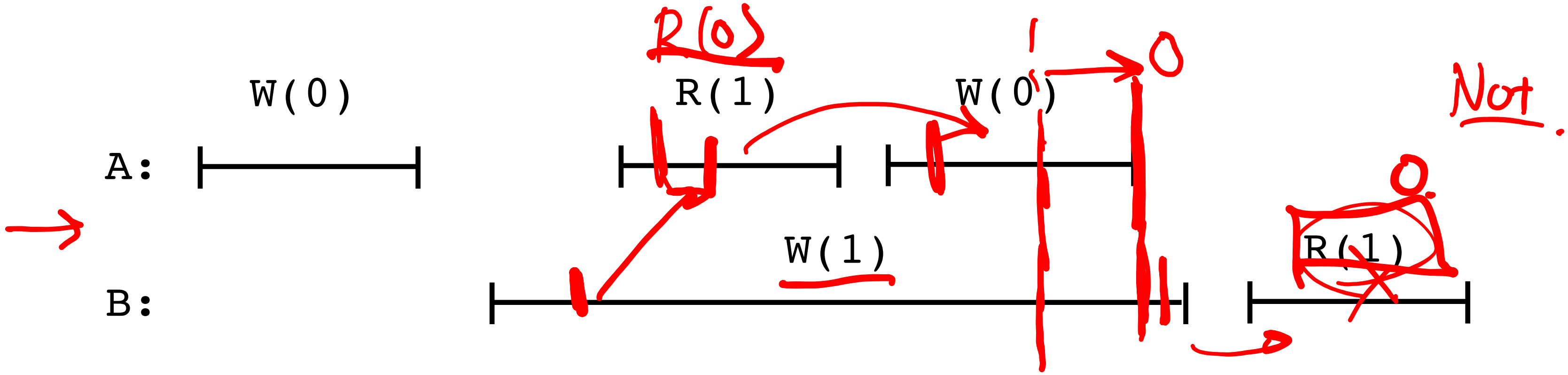

W(0)
W(0)
A:
W(1)
R(0)
B:


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

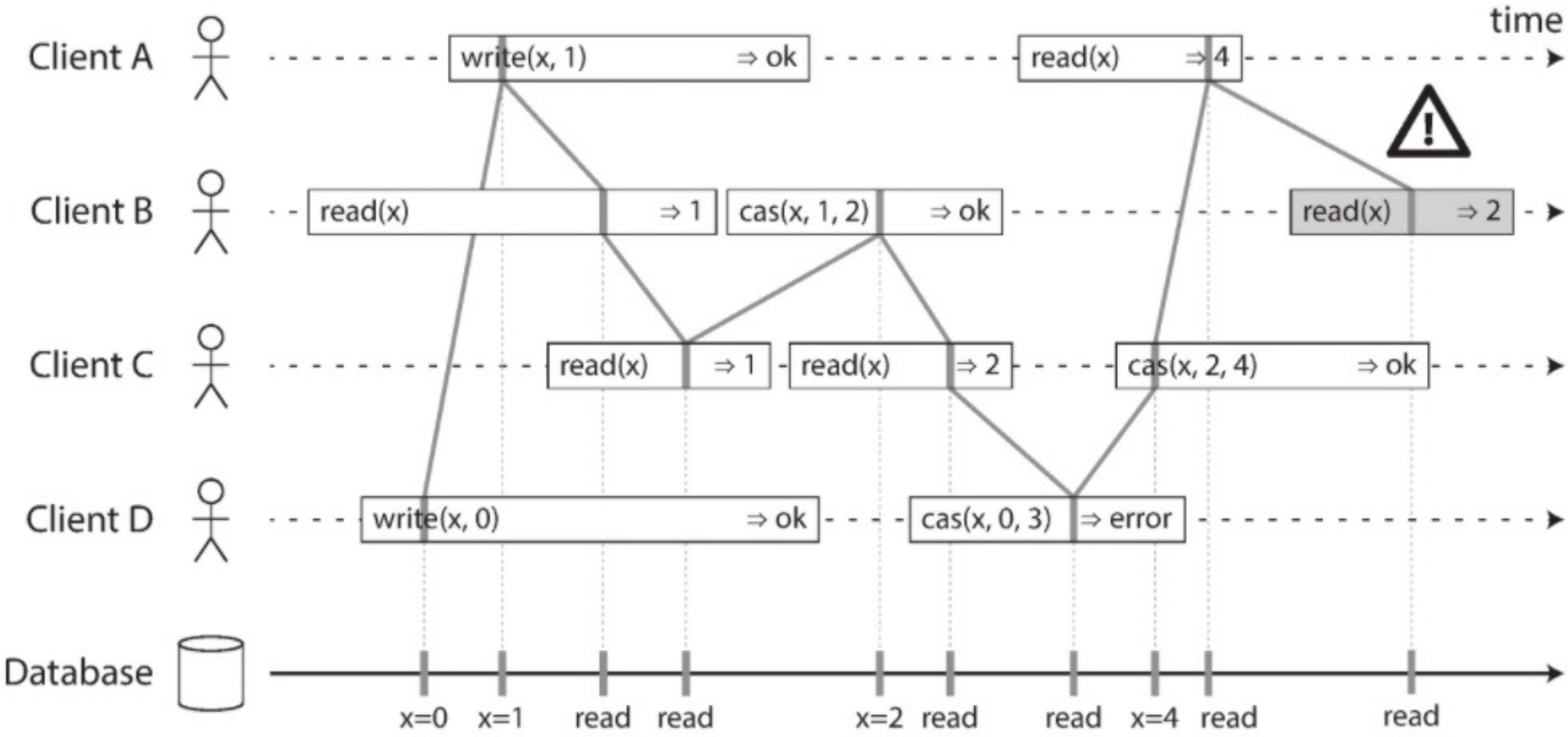

W(0)
R(1)
W(0)
A:
W(1)
R(0)
B:


Value.
W(0)
single obj.
W(0)
Valid
A:
W(1)
R(0)
B:
R(1).
Valid


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


R(0)
W(0)
R(1)
1
W(0)
0
Not.
A:
0
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

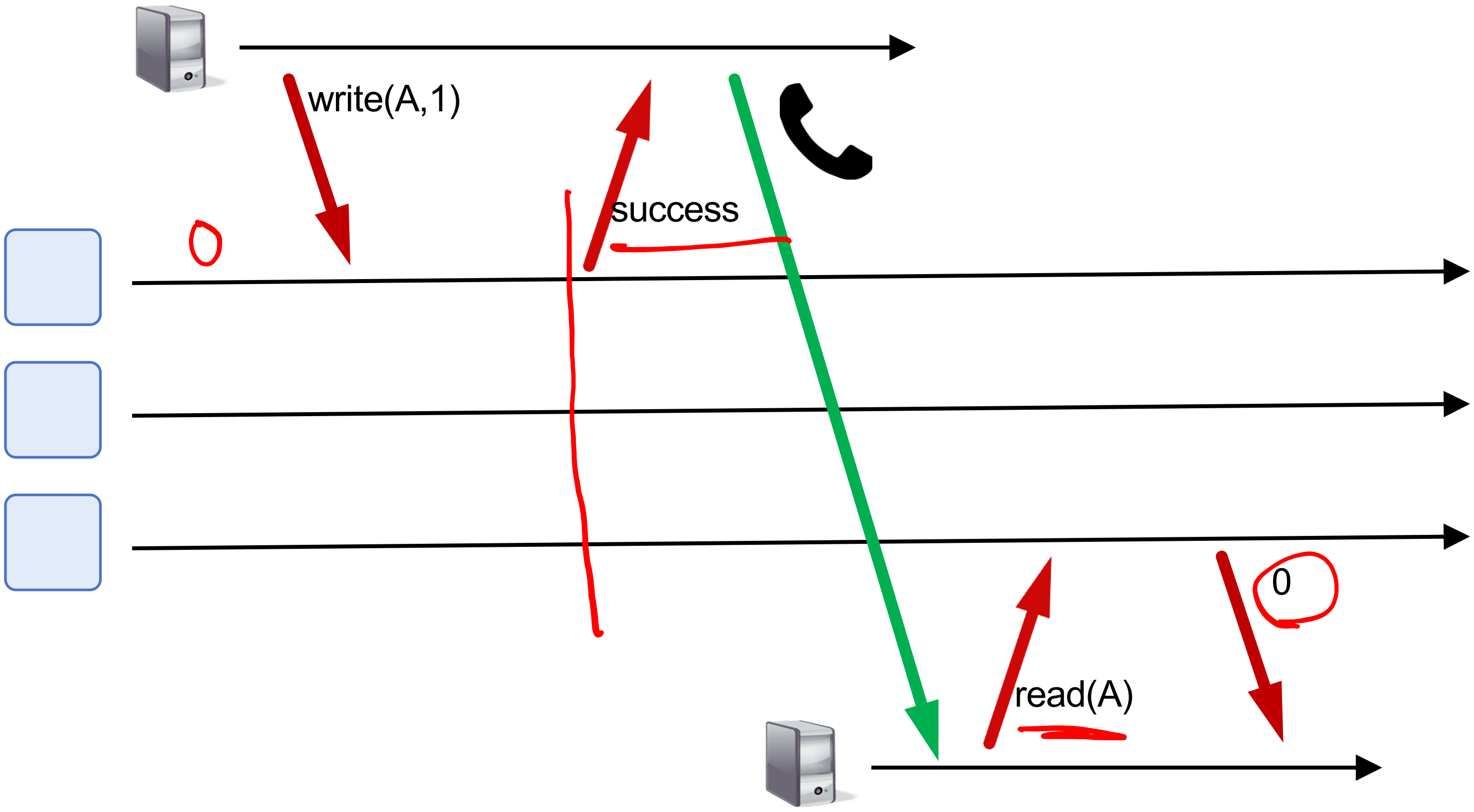
- Sequential = Linearizability – real-time ordering
  1. All servers execute all ops in *some* identical sequential order
  2. Global ordering preserves each client's own local ordering

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

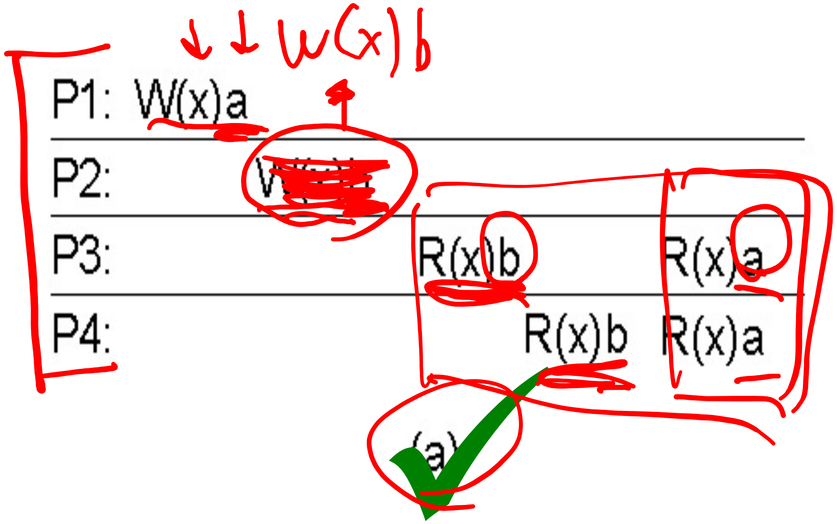
- With concurrent ops, "reordering" of ops (w.r.t. real-time ordering) acceptable, but all servers must see same order
  - e.g., linearizability cares about time
    sequential consistency cares about program order

Sequential Consistency



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

Valid Sequential Consistency?



W(x)b

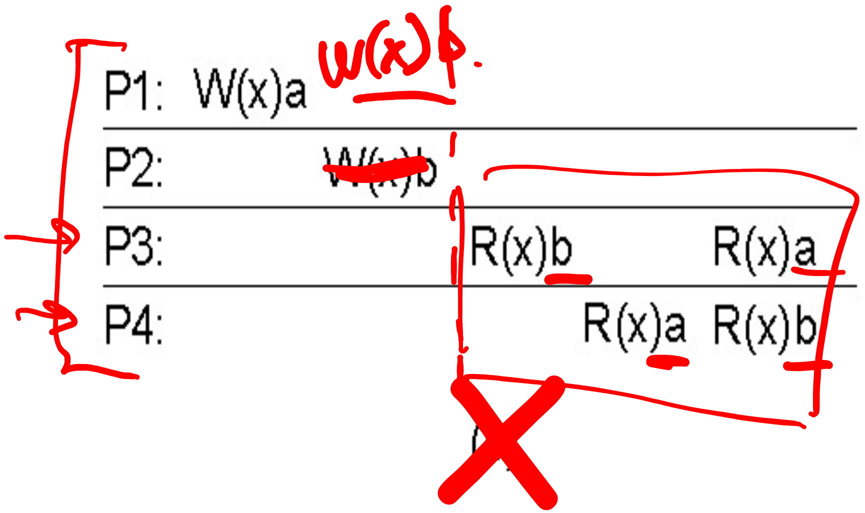
P1: W(x)a

P2:

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

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

(a)



P1: W(x)a W(x)b.

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