Causal Consistency

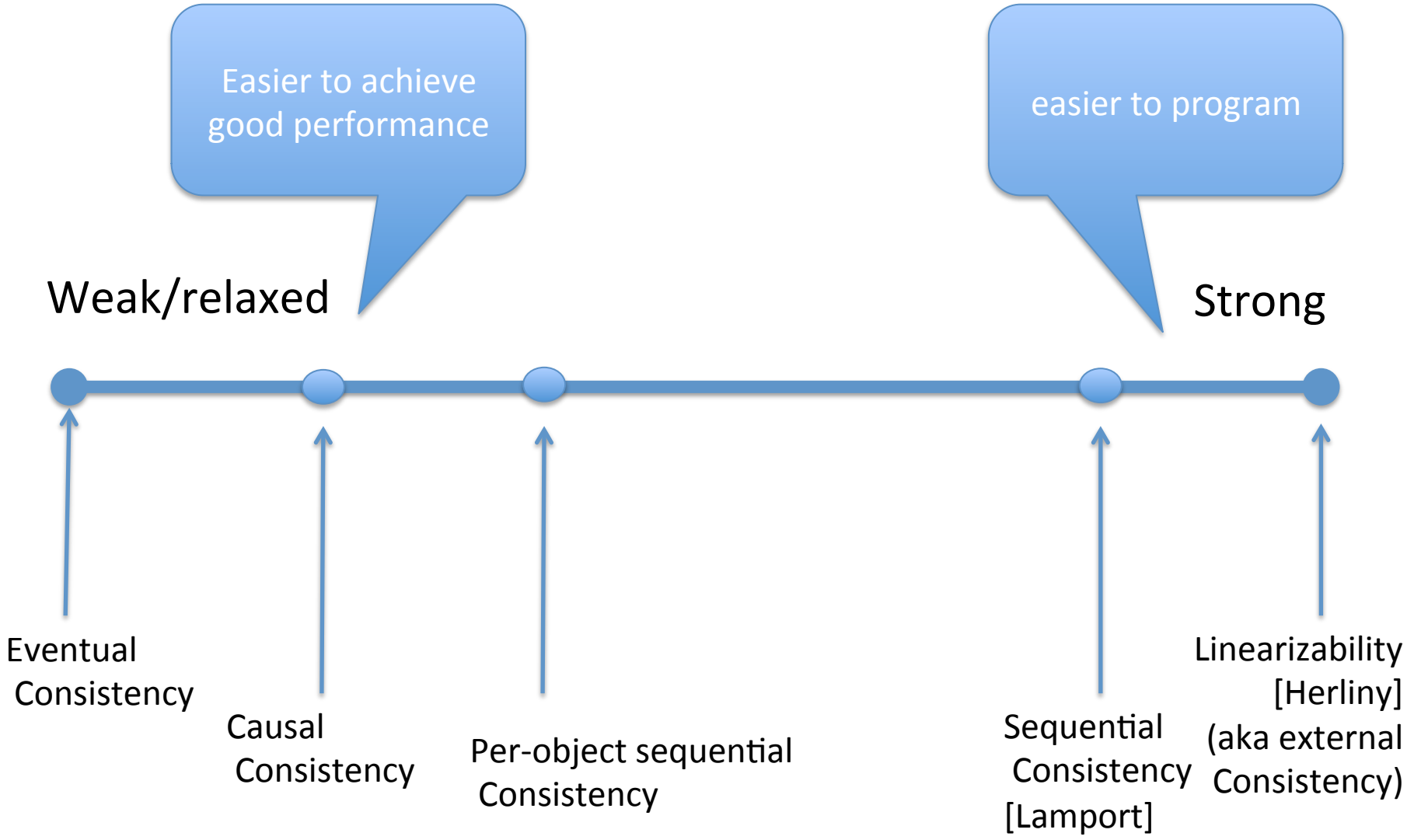
Jinyang Li

some Bayou slides are adapted from Kyle Jamieson's lecture

What we've learnt so far

- Linearizability:
  - equivalence to a serial ordering of events that preserves global completion-to-issue order.
- Fault-tolerant implementation of linearizable replication systems
  - Paxos (MultiPaxos)
  - Raft
  - Viewstamped replication

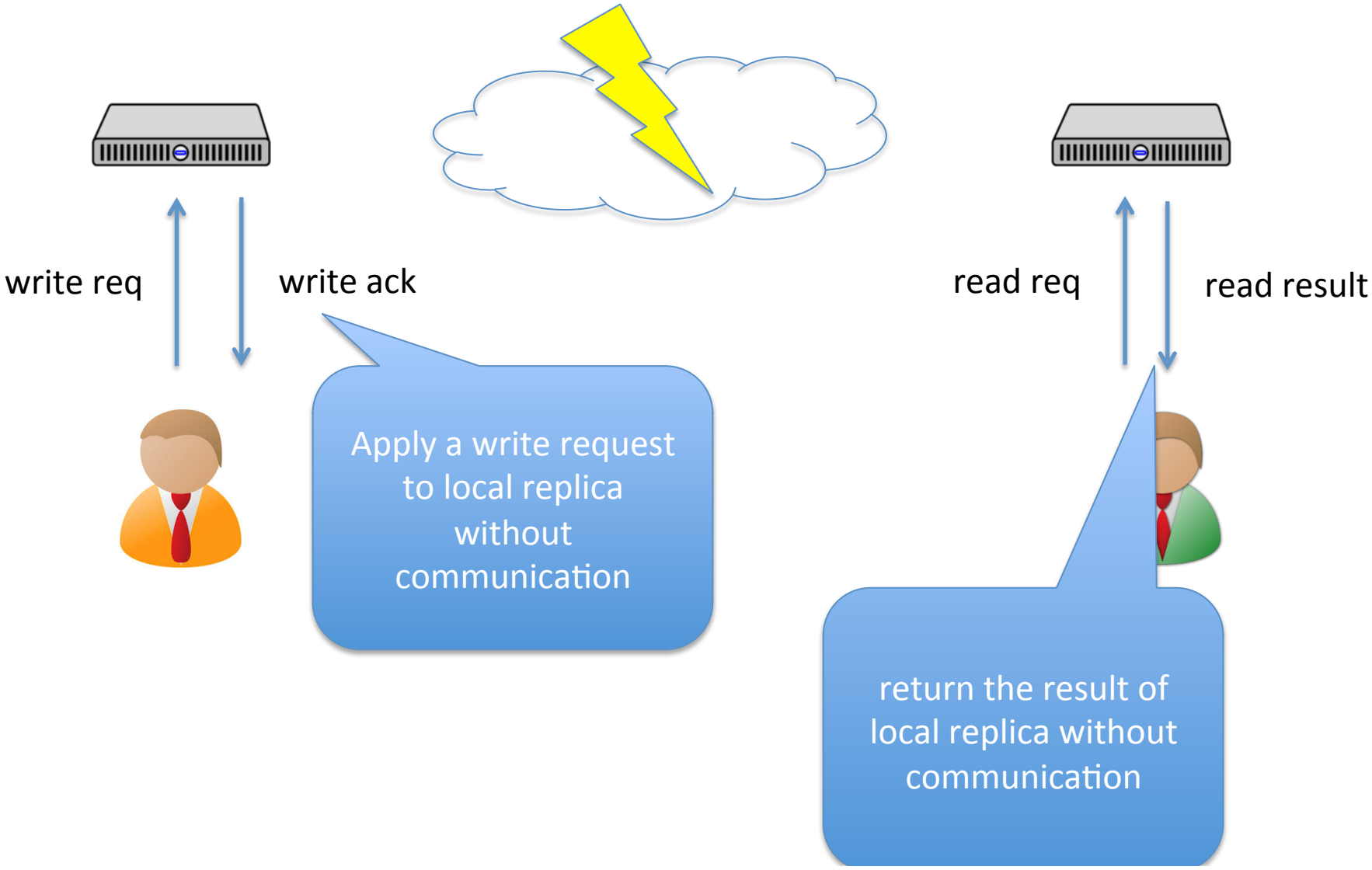
Spectrum of Consistency Models



The downsides of linearizability

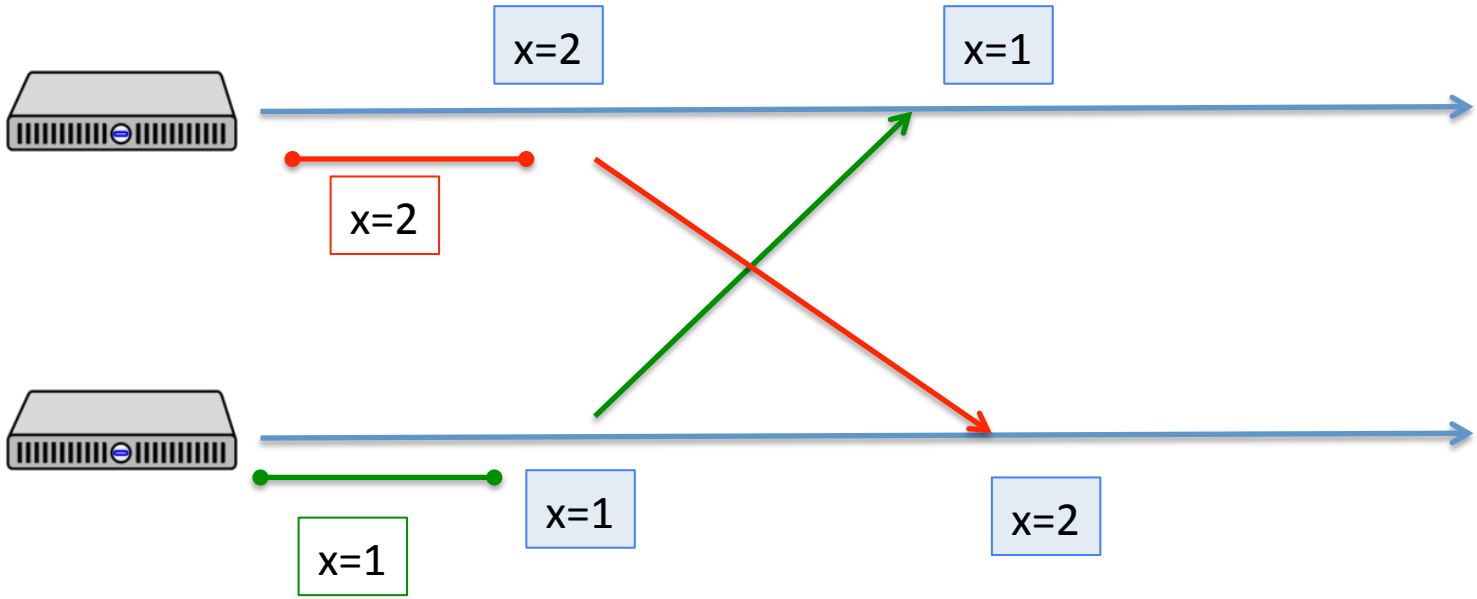
- Linearizable replication needs good network connectivity
- Write latency
  - must wait for a majority of servers to respond
- Read latency
  - w/o read lease: must contact a majority of servers
  - w/ read lease: must renew lease with a majority of servers
- If a replica is disconnected?
  - Can no longer be used

Relaxed consistency can have lower latency & better availability



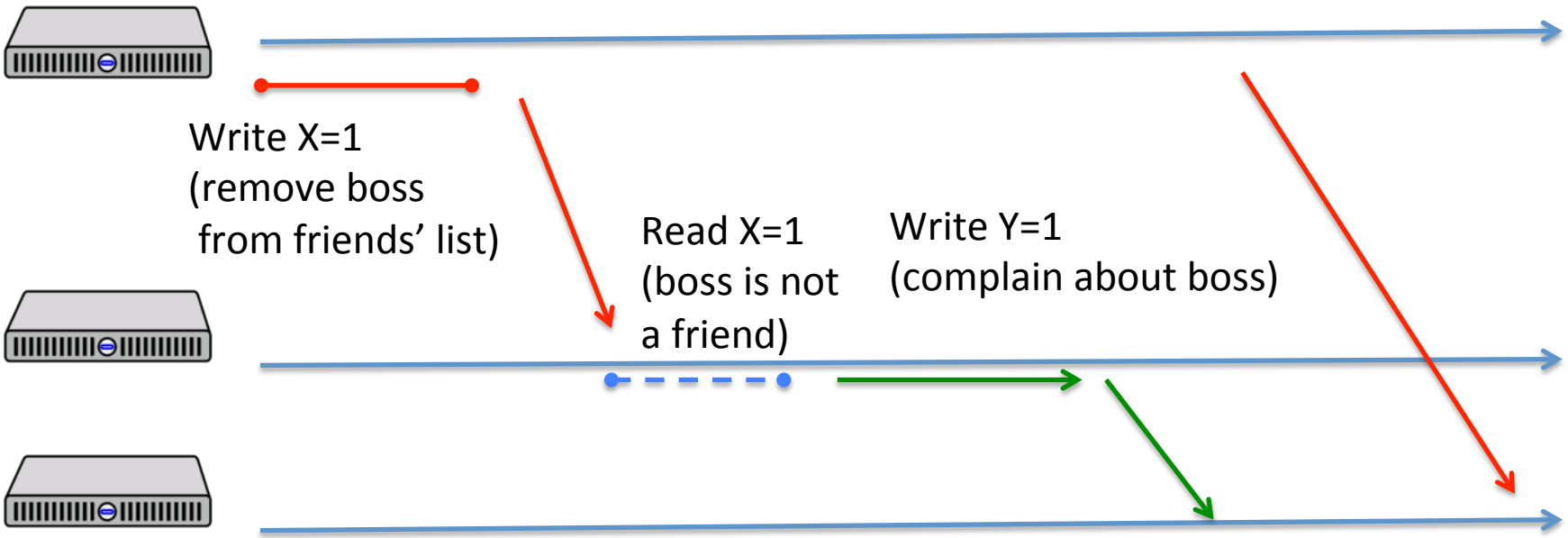
What can go wrong?

1. Replica divergence
   – replicas execute writes in different order



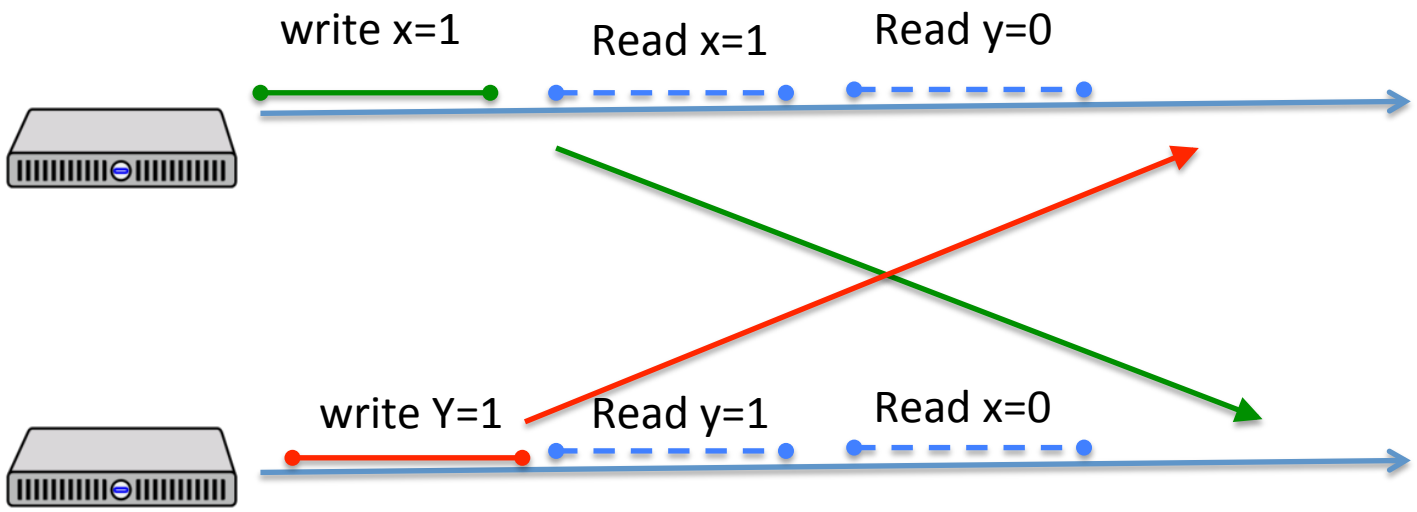
What can go wrong?

2. Causality violation
   – replicas execute writes out of their natural cause-effect order



What can go wrong?

3. Non-linearizable reads



What “wrongs” can be fixed

1. Divergent replicas
2. Causality violation
3. Non-linearizable reads

Causal ordering

- Definition of causal relationship: X → Y iff
  - X, Y created by same site and X is before Y.
  - X, Y created by site-i and j, and X "causes" Y, examples:
    - X is a send event at site-i, Y is the corresponding receive event at site-j.
    - X is the write op at site-i, Y is the execution of X at site-j
  - There exists Z, such that X→Z→Y

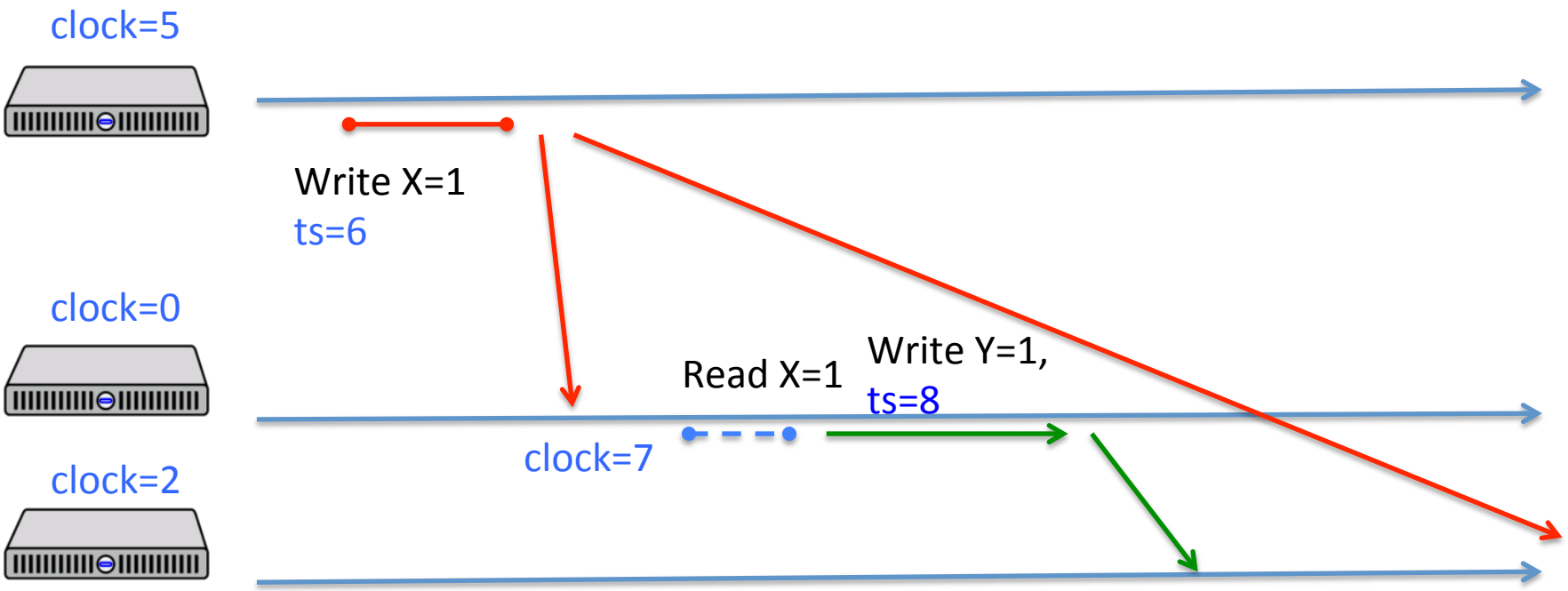
Causal ordering is a partial ordering

- If X→ Y, X happens before Y
- If neither X→Y, or Y→X, then X and Y are "concurrent"

Lamport clock preserves causal ordering

- A logical clock used to assign timestamps to events at each node
- Requirements:
  - If X, Y are created at same site with X before Y, TS(X) < TS(Y)
    - A node advances logical clock, C = C+1, with each new event
  - If X "causes" Y, then TS(X) < TS(Y) →
    - Upon receiving X with TS_X, a node adjusts logical clock upward, C = max(TS_X+1, C)

Lamport clock: the algorithm



Lamport clock can be used for total ordering

- All events can be totally ordered according to tuple [lamport-ts, server-id]
  - X < Y iff X.ts < Y.ts or X.ts = Y.ts and X.server-id < Y.server-id
  - This total ordering preserves causality
- Why total ordering?
  - If replicas apply writes according to total ordering, then replicas converge

Lamport clock question

- If TS(X) < TS(Y), what does it say about X (created by site-i) and Y (created by site-j)?
  1. X occurred at a physical time earlier than Y
  2. Site-i must have communicated with Site-j
  3. if X is a send event from site i to j, then X must have occurred at a physical time earlier than Y
  4. if X is a send event from process i to k (k≠j), then X must have occurred at a physical time earlier than Y

Bayou: A Weakly Connected Replicated Storage System

- Motivating application: Meeting room calendar
- Allow room reservation from any computer
- Want everyone to see the same set of valid reservations, eventually
  - Eventually, no rooms are double booked

Paper context

- Early ’90s when paper was written: Dawn of PDAs, laptops, tablets
  - H/W clunky but showing clear potential
- No wireless connections for devices.
- This problem has not gone away!
  - Devices might be off, not have network access
  - iPhone sync, Dropbox sync, collaborative editing

Bayou idea #1: Replicate a log of writes

- Each server replicates a log of all writes.
- Servers sync with each other to learn of each other's writes
- The alternative:
  - Each server directly modifies its state
  - Servers sync state
  - Why not alternative?

- more expensive to transfer state
- expensive to discover state difference
- Harder to resolve conflicts

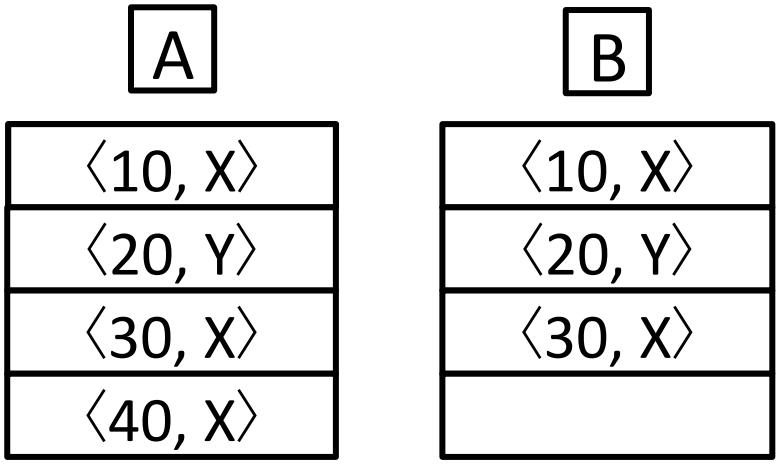
Bayou idea #2: eventual consistent ordering of log entries

- Replica convergence, guaranteed if:
  - Eventually, all servers have all log entries
  - Log entries follow the same total ordering at all servers
  - Log entries correspond to deterministic ops
- Causality preservation:
  - Guaranteed if the total order preserves causality (e.g. Lamport clock)

Bayou synchronization

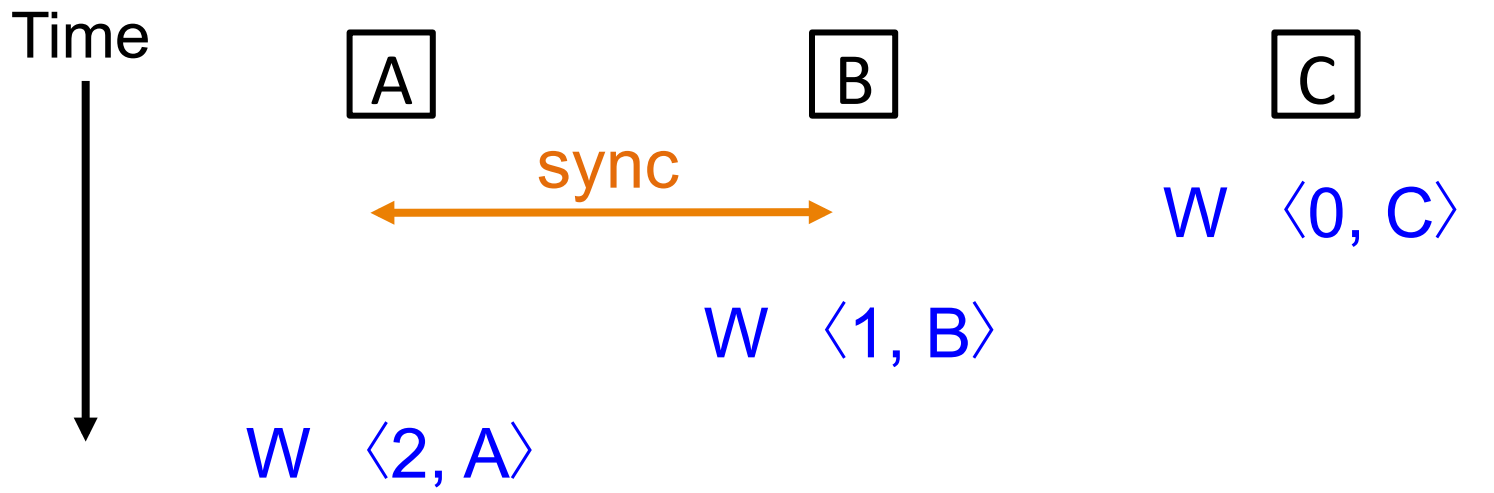
- Each node maintains a log of Lamport clock timestamped writes
- Pair-wise synchronization: X and Y, exchange writes so that their logs become identical after each sync.

How to sync, quickly?

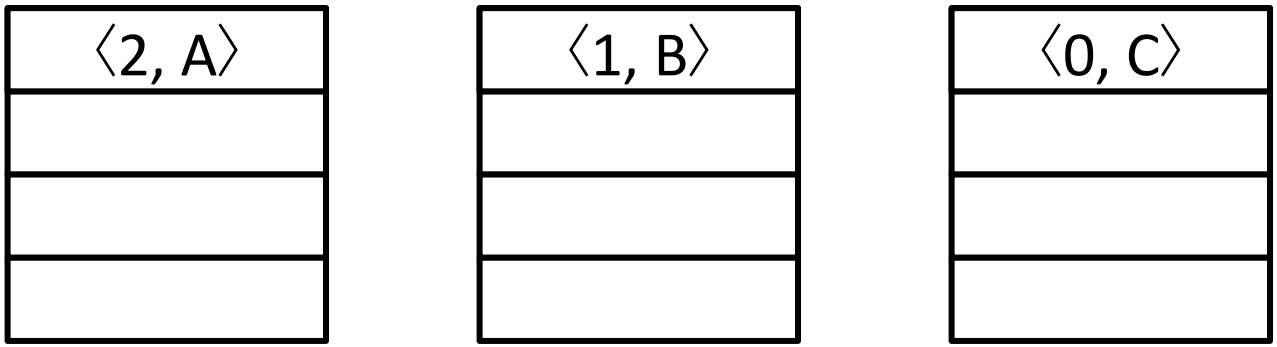


- To sync B with A, B tells A the highest TS it has seen for each other node
  - *Version vector:* [X:30, Y:20]
  - In response, A sends all X's updates after 〈30,X〉, all Y's updates after 〈20,X>

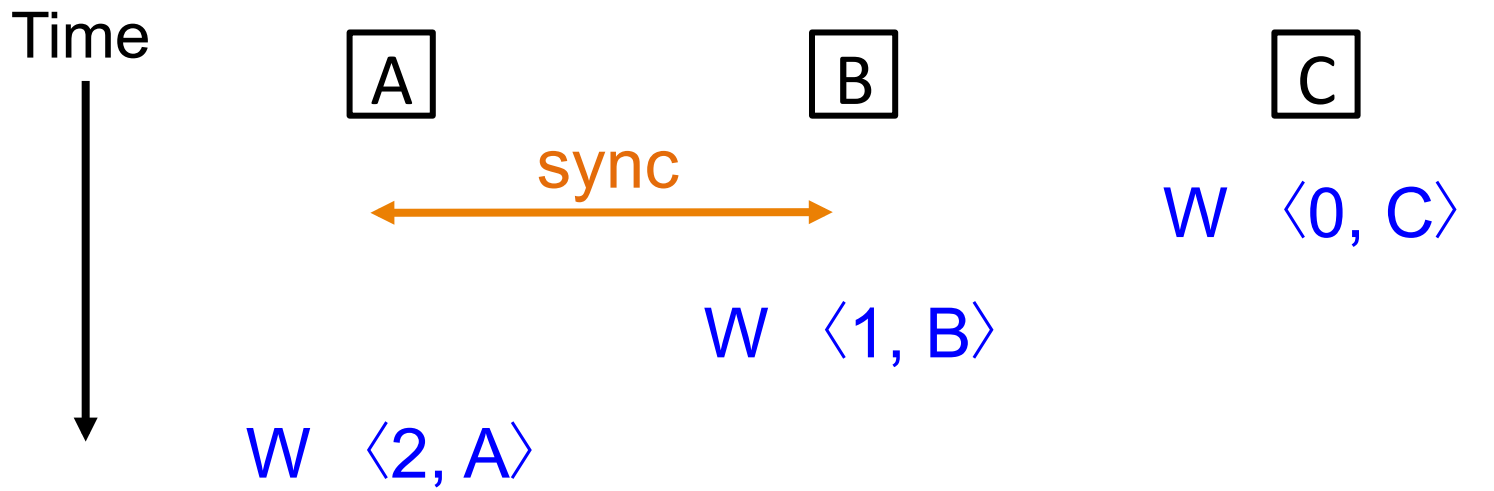
Server synchronization: an example

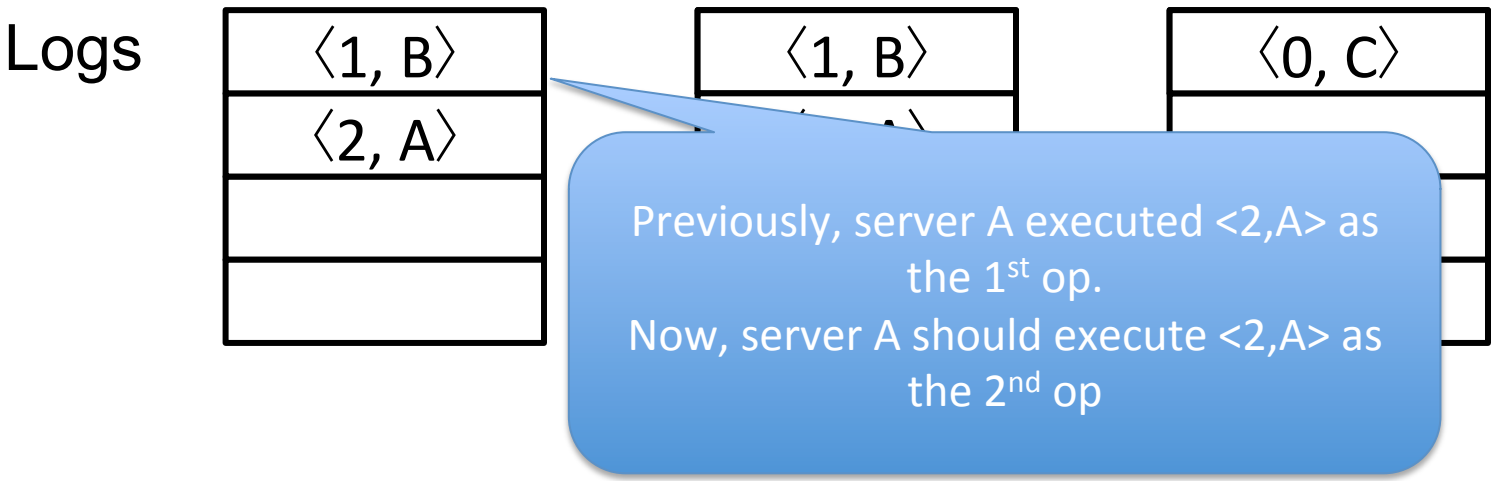


Logs



Server synchronization: an example





Problems of eventual ordering

- User expects to see the effects of his own writes
  - must execute local writes immediately without communication
- Upon sync, servers receive writes whose timestamps are earlier than writes that it has already executed

Solution: Roll back and replay

- Server A needs to "roll back" the DB, and re-run both ops in the lamport order.
- Each server's log consists of 2 portions:
  - stable writes, followed by
  - tentative writes
- write W is stable iff:
  - No entries will have a lamport timestamp < W.
- To be useful, writes should become stabilized soon-ish

How to stabilize writes?

- Decentralized approach: Update 〈10, A〉 is stable if all nodes have seen all updates with TS ≤ 10

- If a node has seen updates with TS > 10 from every node then it'll never again see a timestamp < 〈10, A〉
  - So 〈10, A〉 is stable

- Why doesn't Bayou do this?
  - Any server that remains disconnected would prevent writes from stabilizing
    - So many writes may be rolled back on re-connect

How Bayou commits writes

- Bayou uses a primary commit scheme
  - One designated node (the primary) commits updates
- Servers sync with the primary
- Primary marks each write it receives with a permanent CSN (commit sequence number)
  - That write is committed
  - Complete timestamp = 〈CSN, local TS, node-id〉

Advantage: As long as the primary server is up, writes can be committed (stablized)

How Bayou commits writes (2)

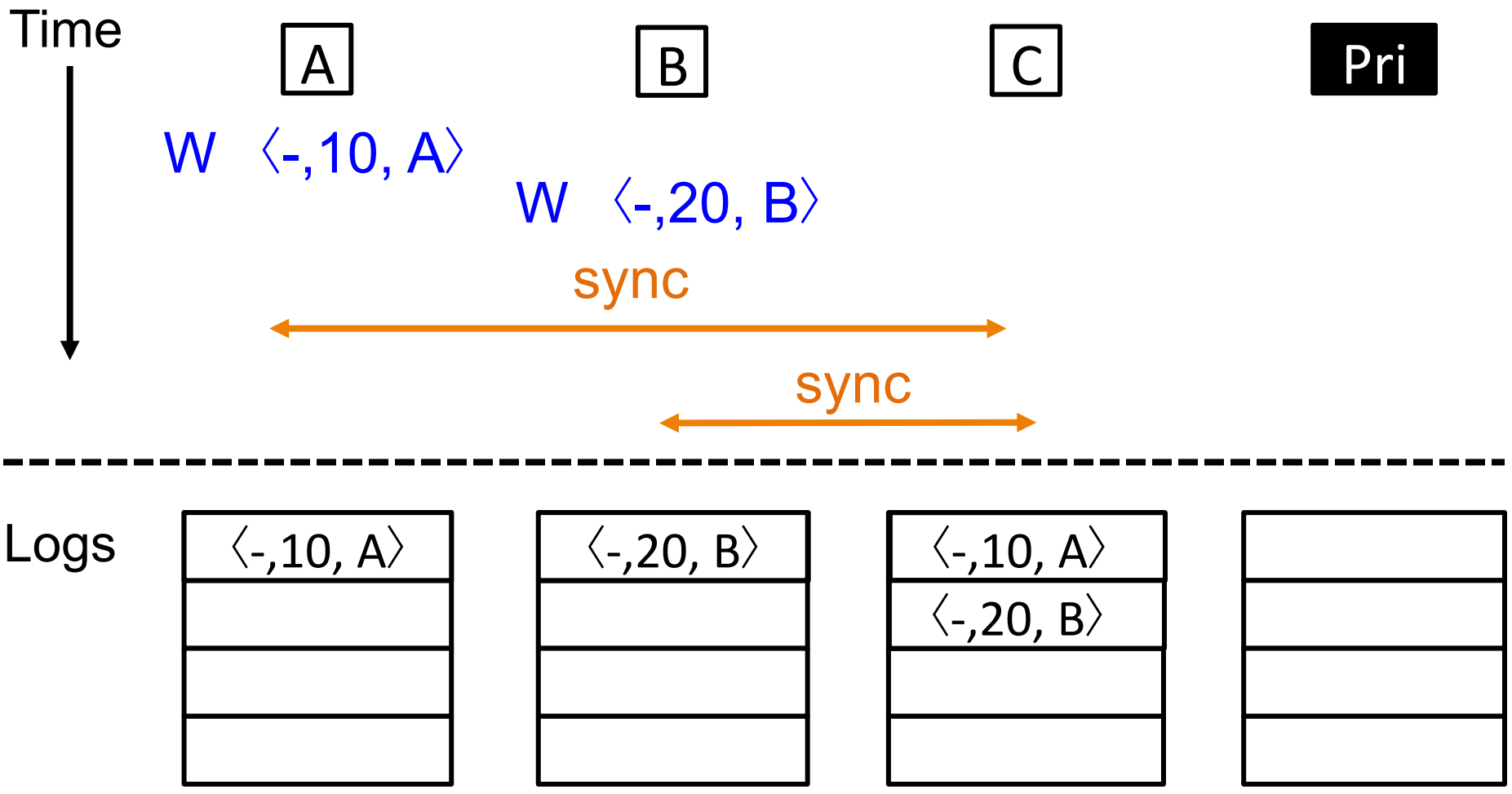
- Nodes exchange CSNs when they sync with each other

- CSNs define a total order for committed writes
  - All nodes eventually agree on the total order
  - Tentative writes come after all committed writes

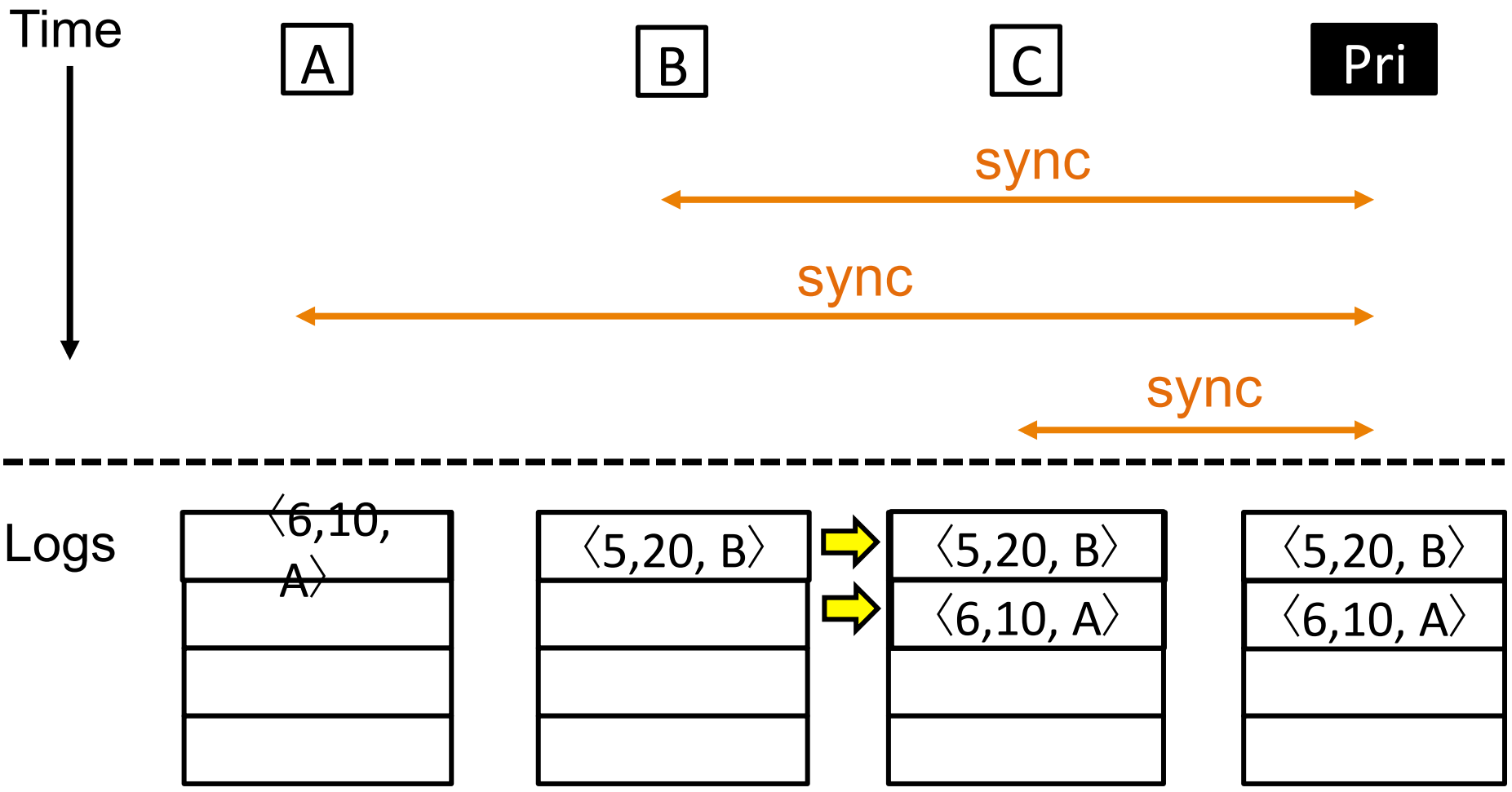
Does CSN order preserve causality?

- Yes
- A server asks the primary to assign CSN for all tentative writes (include those received from others)

Tentative order ≠ commit order



Tentative order ≠ commit order



Trimming the log

- When nodes receive new CSNs, can **discard** all committed log entries seen up to that point
  - Update protocol → **CSNs received in order**
- Keep copy of whole database as of highest CSN
- Result: No need to keep years of log data

Syncing with trimmed logs

- Suppose nodes discard all writes in log with CSNs
  - Just keep a copy of the **"stable" DB**, reflecting discarded entries
- Must not apply writes already in stable DB
  - Remember writes with highest CSN in "stable" DB.
  - If receiving a write with lower CSN, discard

Bayou idea #3: Application-specific conflict resolution

- Roll-back and re-apply writes ensures replica convergence
- But replica “convergence” is not sufficient for application-level “consistency”.

What's in a write?

- Suppose calendar update takes form, to reserve 1-hour meeting
  - Write(key="Room410@10am", value="DS meeting")
- What happens if Alice and Bob both reserve 10am slot?

Bayou's application-specific conflict resolution

- No blind write
- Bayou's write is an app-specific **update function**
  "1-hour meeting at 10 AM if Room 410 is free, else at 11am, else at noon, else raise an error"

Replicas converge if all servers execute same instructions in same order, eventually

Bayou's application user interface

- Displayed meeting room calendar entries are "Tentative" at first
  - A user sees his meeting tentatively scheduled at 10 AM, later it might be moved to 11 AM
- Once committed, room reservation is secured
- Is this an intuitive interface?

Let's step back

- *Is eventual consistency a useful idea?*
- Yes: people want fast writes to local copies iPhone sync, Dropbox, collaborative editing

- *Are update conflicts a real problem?*
- Yes—conflict resolution is application dependent

Is Bayou's complexity necessary?

- Much complexity is due to peer-to-peer sync
- Nowadays, a common sync model is with the cloud
  - keep the primary in the cloud
  - every server syncs with the primary only

What are Bayou's take-away ideas?

1. Update functions for automatic application-specific conflict resolution

2. Ordered update log is the real truth, not the DB

3. Application of Lamport logical clocks for causal consistency