بسم الله الرحمن الرحیم

تکنولوژی کامپیوتر

جلسه‌ی هفدهم

مصائب سیستم‌های توزیع‌شده

این جلسه

سیستم‌های توزیع‌شده، تفاوت و مشکلاتش نسبت به یک کامپیوتر، مدل ریاضی مسئله، خطی‌سازی

FAULT VS PARTIAL FAULT

Single Computer

- Deterministic
- Fault
  - *Blue Screen*
  - *Usually a computer fully crash*
  - *In single-system applications with proper code an application either works or it doesn’t*

Distributed systems

- Partial Fault
  - *We know individual systems will fail or groups of systems will fail but we need the overall system to continue working*
  - *it may sometimes work and sometimes unpredictably fail.*
- You may not even know whether something succeeded or not, as the time it takes for a message to travel across a network is also nondeterministic!

Building a Reliable System from Unreliable Components??

- TCP from unreliable IP
  - *But cannot remove delays in the network*
- Error-correcting codes
  - *But only can deal with small number of single-bit errors*

UNRELIABLE NETWORKS

shared-nothing systems

Asynchronous packet networks

- The internet and most internal networks in datacenters (often Ethernet) are asynchronous packet networks
- one node can send a message (a packet) to another node
  - *no guarantees as to when it will arrive*
  - *Or no guarantees whether it will arrive at all.*

Asynchronous packet networks

- If you send a request and expect a response, many things could go wrong
  - *Your request may have been lost*
  - *Your request may be waiting in a queue and will be delivered later*
  - *The remote node may have failed*
  - *The remote node may have temporarily stopped responding*
  - *The remote node may have processed your request, but the response has been lost on the network*
  - *The remote node may have processed your request, but the response has been delayed and will be delivered later*

Asynchronous packet networks

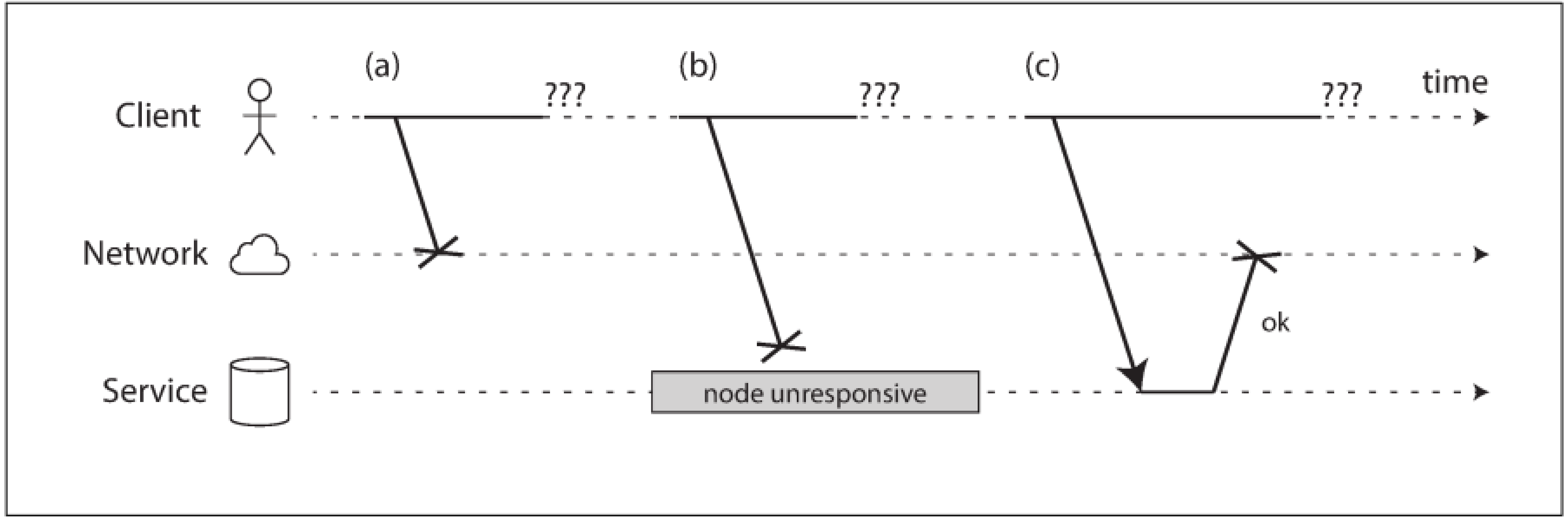

Figure 8-1. If you send a request and don't get a response, it's not possible to distinguish whether (a) the request was lost, (b) the remote node is down, or (c) the response was lost.

Asynchronous packet networks

- the only information you have is that you haven’t received a response yet!
- If you send a request to another node and don’t receive a response, it is <u>impossible</u> to tell why
- The usual way of handling this issue is a **timeout**

Network Faults in Practice

- Network problems can be surprisingly common! ☹
- Network partitions
  - *When one part of the network is cut off from the rest due to a network fault*
  - *called a network partition or netsplit*

Network Faults in Practice

- Handling network fault?
  - *Tolerating them?*
  - *If your network is normally fairly reliable, a valid approach may be to simply show an error message to users while your network is experiencing problems*

Detecting Faults

- Automatically detect faulty nodes
- Why?
  - *A load balancer needs to stop sending requests to a node that is dead (i.e., take it out of rotation)*
  - *In a distributed database with single-leader replication, if the leader fails, one of the followers needs to be promoted to be the new leader*

Detecting Fault?

- If the system is still up but application has run into a problem then possible to get a return from the system saying it is present but non-operational
- No way of knowing what requests it may have received and left unanswered or what data was sent to it and lost
- Possible for OS to directly announce it has failed so other systems don’t have to wait for timeout
- Possible to use remote management tools to check on systems if you are closely networked

Timeouts and Unbounded Delays

- How long should the timeout be?
  - *There is unfortunately no simple answer.* ☹
  - *Long Timeout …*
  - *Short timeout …*

When delay happen?

- Often delays are caused by queuing of some sort
  - *Multiple systems trying to send info to the same system, all have to be handled in whatever order they are received, but some will have to wait*
  - *If all CPU cores of a system are busy it will have to wait for cycles to address the information*
  - *In virtual systems they can be temporarily frozen while other VMs are doing things*
- Possible to have a noisy neighbor that is causing delays for you by using up networking capacity

- Have to determine appropriate timeout lengths experimentally

Synchronous Versus Asynchronous Networks

- Distributed systems would be a lot simpler if we could rely on the network to deliver packets with some fixed maximum delay, and not to drop packets
- traditional fixed-line telephone network
  - *is extremely reliable*
  - *delayed audio frames and dropped calls are very rare*

Synchronous Versus Asynchronous Networks

- When you make a call over the telephone network
  - *it establishes a* <u>*circuit*</u>
  - *a fixed, guaranteed amount of bandwidth is allocated for the call, along the entire route between the two callers.*

Synchronous Versus Asynchronous Networks

- Can we not simply make network delays predictable?
- The answer is that they are optimized for bursty traffic.
- Latency and Resource Utilization

UNRELIABLE CLOCKS

Unreliable Clocks

- 1. Has this request timed out yet?
- 2. What’s the 99th percentile response time of this service?
- 3. How many queries per second did this service handle on average in the last five
- minutes?
- 4. How long did the user spend on our site?
- 5. When was this article published?
- 6. At what date and time should the reminder email be sent?
- 7. When does this cache entry expire?
- 8. What is the timestamp on this error message in the log file?

Clocks in your computer

- Time-of-day clocks
  - *Time-of-day clocks are usually synchronized with NTP*
  - *System.currentTimeMillis() in java*
- Monotonic clocks
  - *System.nanoTime() in java*
  - *NTP may adjust the frequency at which the monotonic clock moves forward*

Clock Synchronization and Accuracy

- NTP synchronization can only be as good as the network delay
- …

Relying on Synchronized Clocks

- they seem simple and easy to use But …
- a day may not have exactly 86,400 seconds!
- time-of-day clocks may move backward in time
- time on one node may be quite different from the time on another node
- -> robust software needs to be prepared to deal with incorrect clocks.

Timestamps for ordering events

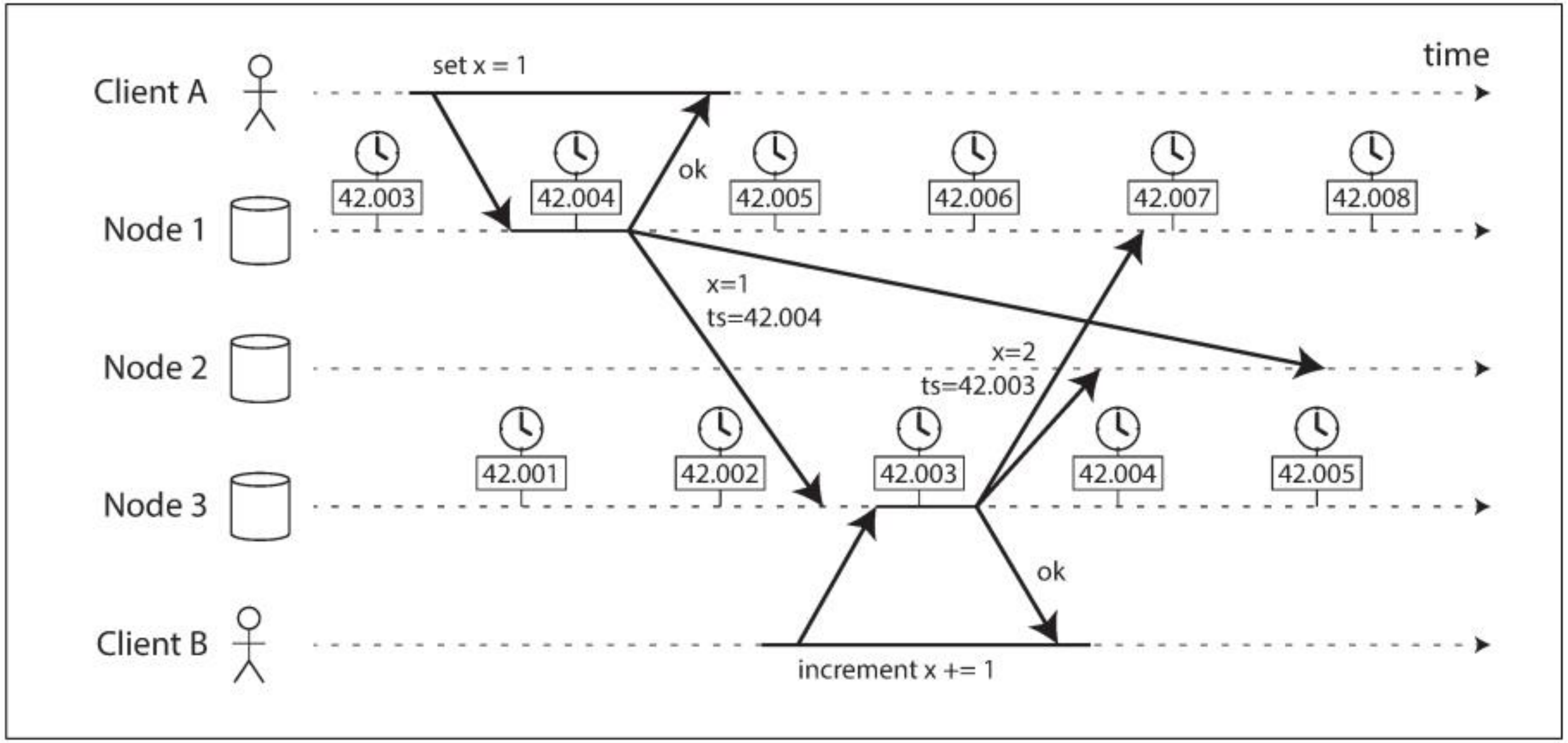

Figure 8-3. The write by client B is causally later than the write by client A, but B's write has an earlier timestamp.

Clock readings have a confidence interval

- You may be able to read a machine's time-of-day clock with microsecond or even nanosecond resolution
  - *Is it mean that the value is actually accurate to such precision?*
  - *No* ☹

Clock readings have a confidence interval

- Unfortunately, most systems don’t expose this uncertainty
  - *for example, when you call clock_gettime(), the return value doesn’t tell you the expected error of the timestamp,*
  - *you don’t know if its confidence interval is five milliseconds or five years.*

Clock readings have a confidence interval

- The uncertainty bound can be calculated based on your time source…
- If
  - *GPS receiver*
  - *atomic (caesium) clock*
- directly attached to your computer

Clock readings have a confidence interval

- Google's Spanner TrueTime API
  - *which explicitly reports the confidence interval on the local clock*
  - *you get back two values: [earliest, latest]*
    - the earliest possible
    - And the latest possible timestamp.

Synchronized clocks for global snapshots

- Snapshot isolation...
  - *Monotonically increasing transaction ID*
  - *To implement MVVC*
- Can we use timestamp for transaction id in distributed system?
- Spanner implementation of snapshot isolation

Process Pauses

```
while (true) {
    request = getIncomingRequest();

    // Ensure that the lease always has at least 10 seconds remaining
    if (lease.expiryTimeMillis - System.currentTimeMillis() < 10000) {
        lease = lease.renew();
    }

    if (lease.isValid()) {
        process(request);
    }
}
```

Process Pauses

- Garbage collector
- Suspend in virtual machines
- Suspend in laptops
- Context switching
- Stop process using stop signal

Preventing garbage collector pauses?

- Short lived processes
- Restart using rolling upgrade

KNOWLEDGE, TRUTH, AND LIES

Knowledge

- A node in the network cannot know anything for sure
- it can only make guesses based on the messages it receives (or doesn't receive) via the network
- If a remote node doesn't respond…
  - *Is node has a problem? Which node?*
  - *Is network has a problem?*

The truth?

- Example
  - *Node receive message from others*
  - *But can’t send message to others*
  - *Other nodes think that it is dead*
  - *the semi-disconnected node is dragged to the graveyard, kicking and screaming “I’m not dead!”*

- Example
  - *Node with long stop-the-world GC*

The Truth Is Defined by the Majority

- the quorum is an absolute majority of more than half the nodes
- it is safe, because there can only be only one majority in the system
- … consensus algorithms …

The leader and the lock

- Lock for distributed systems
  - *Only one node is allowed to be the leader for a database partition, to avoid split brain*
  - *Only one transaction or client is allowed to hold the lock for a particular resource or object, to prevent concurrently writing to it and corrupting it.*
  - *Only one user is allowed to register a particular username, because a username must uniquely identify a user.*

The leader and the lock

- even if a node believes that it is “the chosen one”
- Is it mean that quorum of nodes agrees?
  - *No!*
  - *Example…*

The leader and the lock

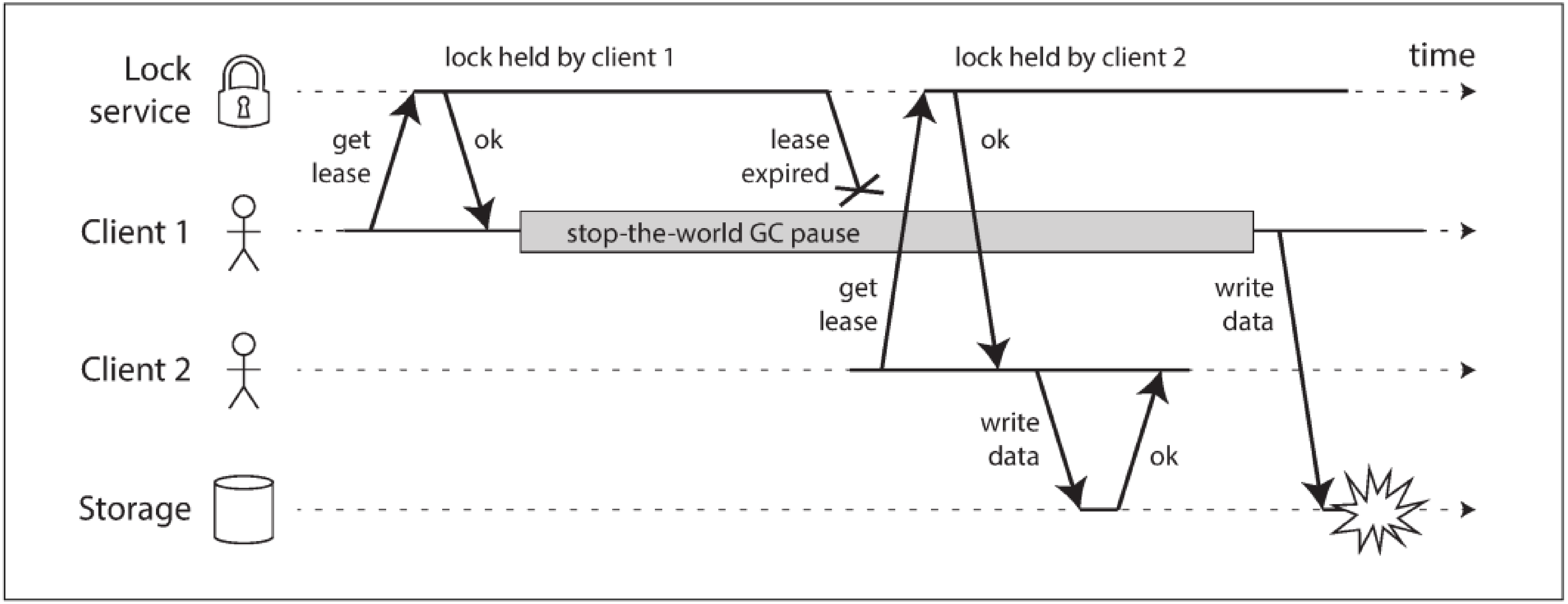

Figure 8-4. Incorrect implementation of a distributed lock: client 1 believes that it still has a valid lease, even though it has expired, and thus corrupts a file in storage.

The leader and the lock - Fencing tokens

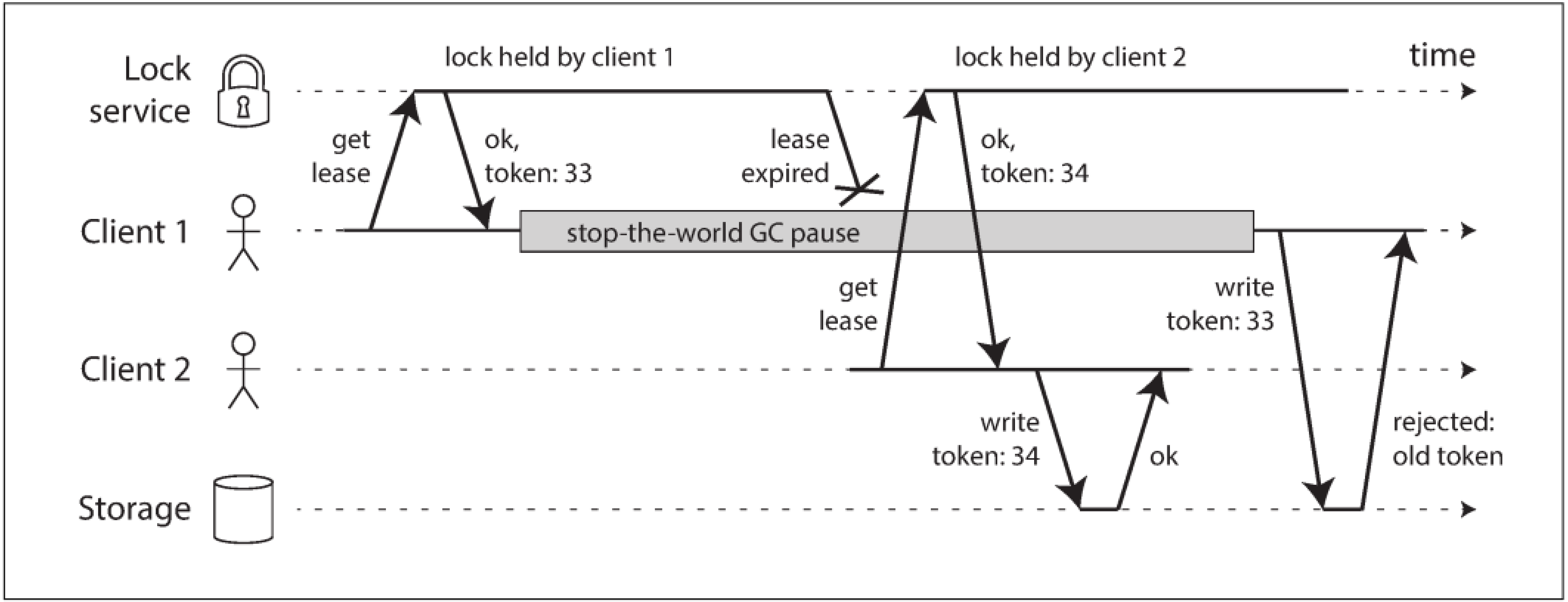

Figure 8-5. Making access to storage safe by allowing writes only in the order of increasing fencing tokens.

Byzantine Faults

SYSTEM MODEL AND REALITY

Timing assumptions

- Synchronous model
  - *bounded network delay, bounded process pau- ses, and bounded clock error*
- Partially synchronous model
  - *system behaves like a synchronous system most of the time, but it sometimes exceeds the bounds for network delay, process pauses, and clock drift*
- Asynchronous model
  - *an algorithm is not allowed to make any timing assumption*

Node failures

- Crash-stop faults
- Crash-recovery faults
- Byzantine (arbitrary) faults

Correctness of an algorithm?

- Example on fencing token
- Uniqueness
  - *No two requests for a fencing token return the same value.*
- Monotonic sequence
  - *If request x returned token tx , and request y returned token ty , and x completed before y began, then tx < ty*
- Availability
  - *A node that requests a fencing token and does not crash eventually receives a response*

Correctness of an algorithm?

- Safety and liveness
- Safety
  - *nothing bad happens*
- Liveness
  - *something good eventually happens*

LINEARIZABILITY

Consistency Guarantees

- Eventual consistency

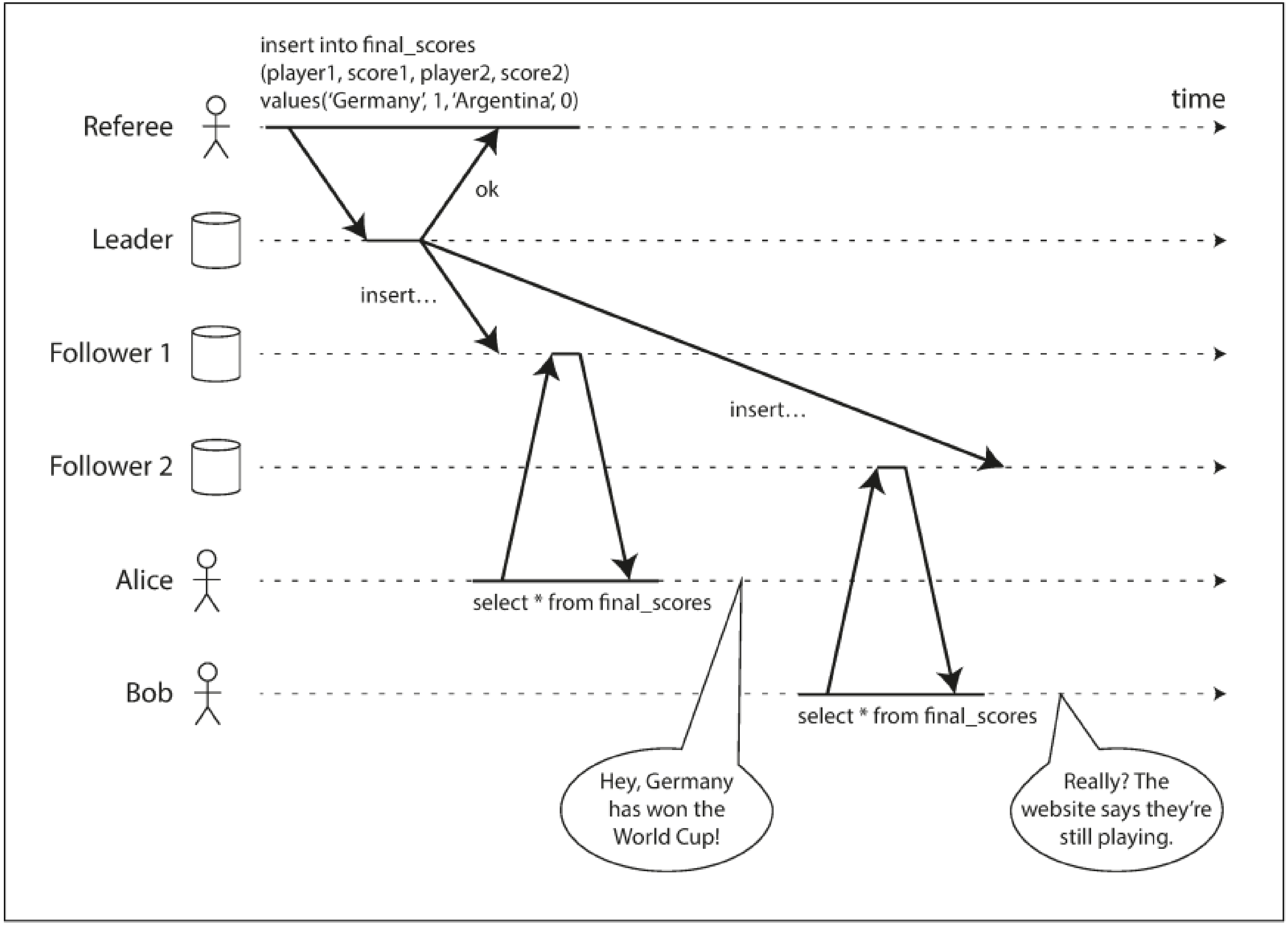

Figure 9-1. This system is not linearizable, causing football fans to be confused.

Linearizability

- atomic consistency
- strong consistency
- immediate consistency
- external consistency

Linearizability

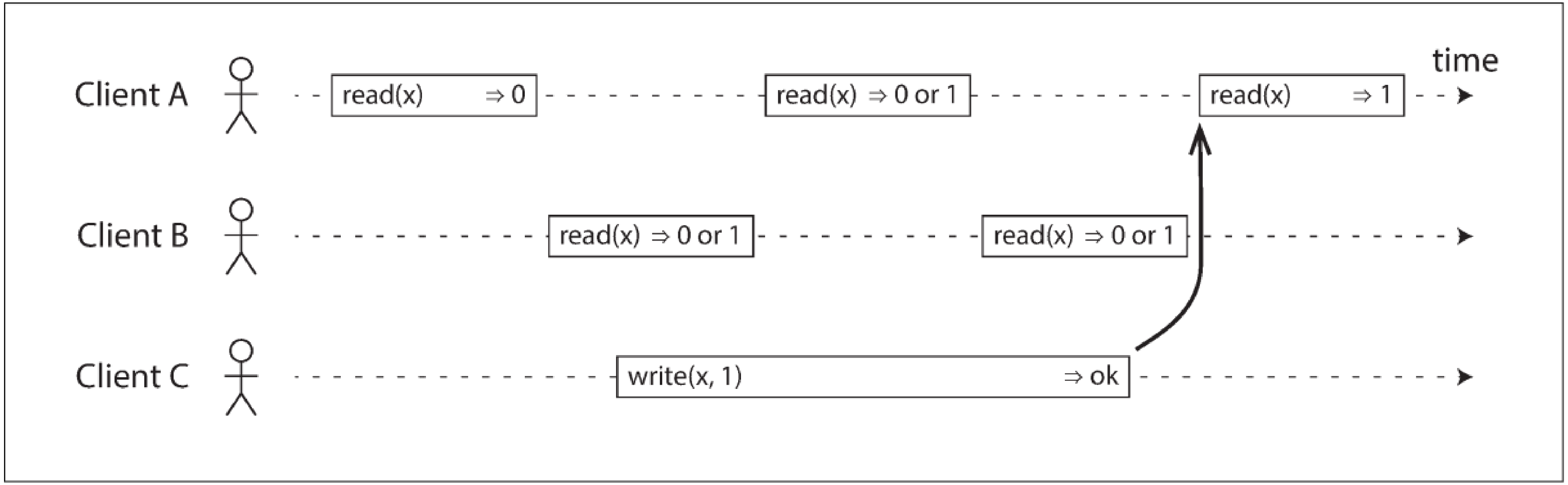

Figure 9-2. If a read request is concurrent with a write request, it may return either the old or the new value.

Linearizability

To make the system linearizable, we need to add another constraint, illustrated in Figure 9-3.

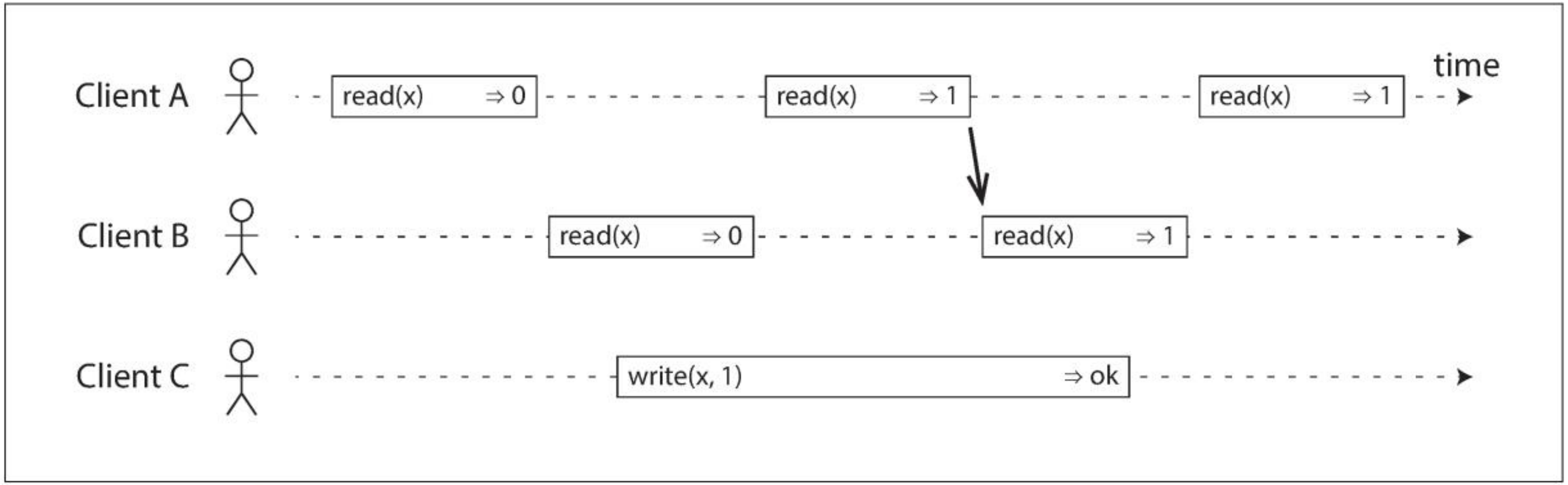

Figure 9-3. After any one read has returned the new value, all following reads (on the same or other clients) must also return the new value.

Linearizability

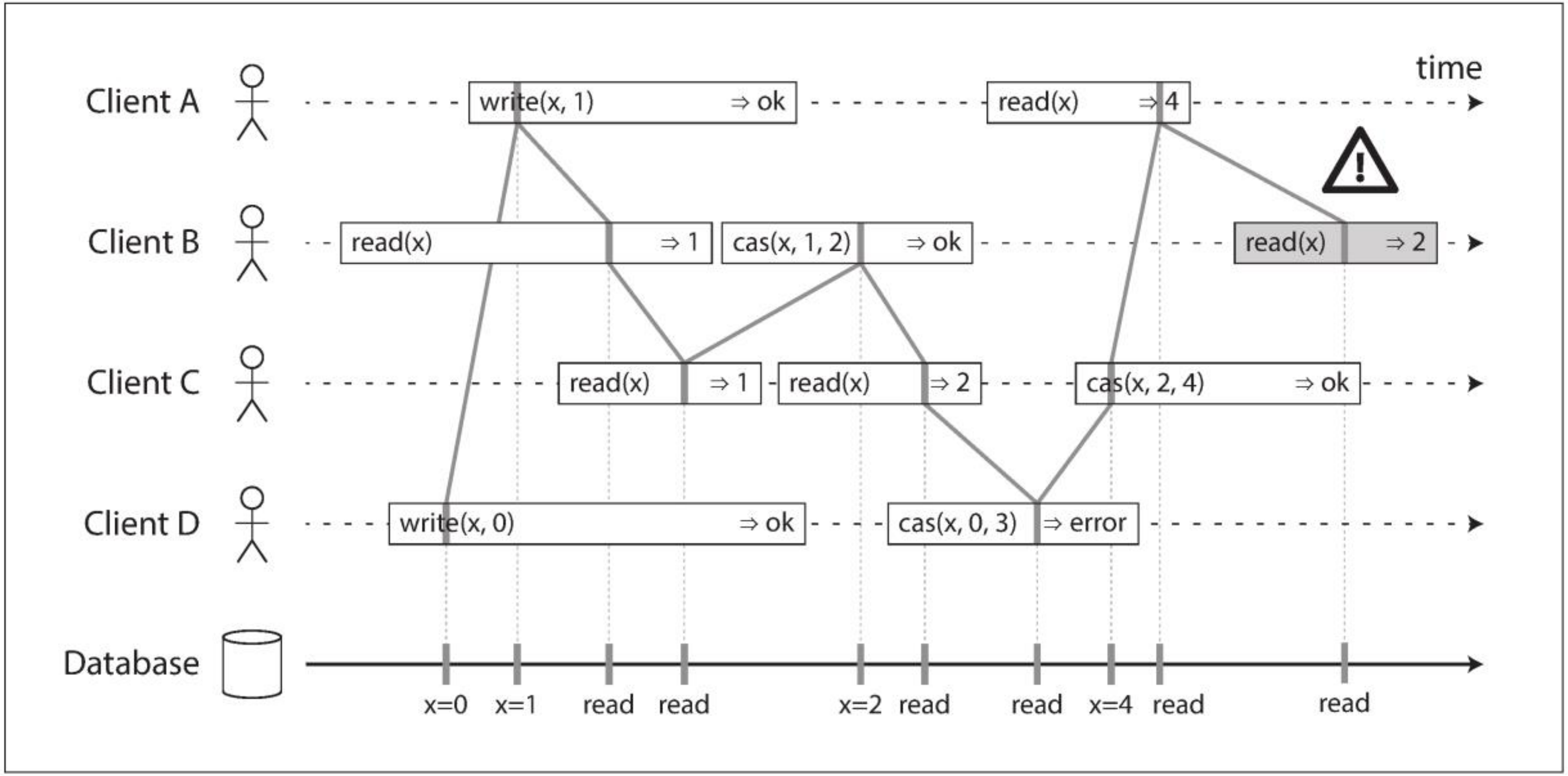

Figure 9-4. Visualizing the points in time at which the reads and writes appear to have taken effect. The final read by B is not linearizable.

Linearizability Versus Serializability

جلسه‌ی بعد