بسم الله الرحمن الرحیم

تکنولوژی کامپیوتر

جلسه‌ی هجدهم

الگوریتم‌هایی در سیستم‌های توزیع‌شده

جلسه‌ی گذشته

این جلسه

Introduction

- We’ve looked at the building blocks, and last chapter discussed all of the things that can go wrong in distributed systems
- This chapter talks about some of the ways to address consensus, one of the hardest problems, and we’ll see that it’s very tricky, and approaches can be difficult or expensive

Consistency Guarantees

- Eventual Consistency / Convergence

Consistency Guarantees

- Eventual Co

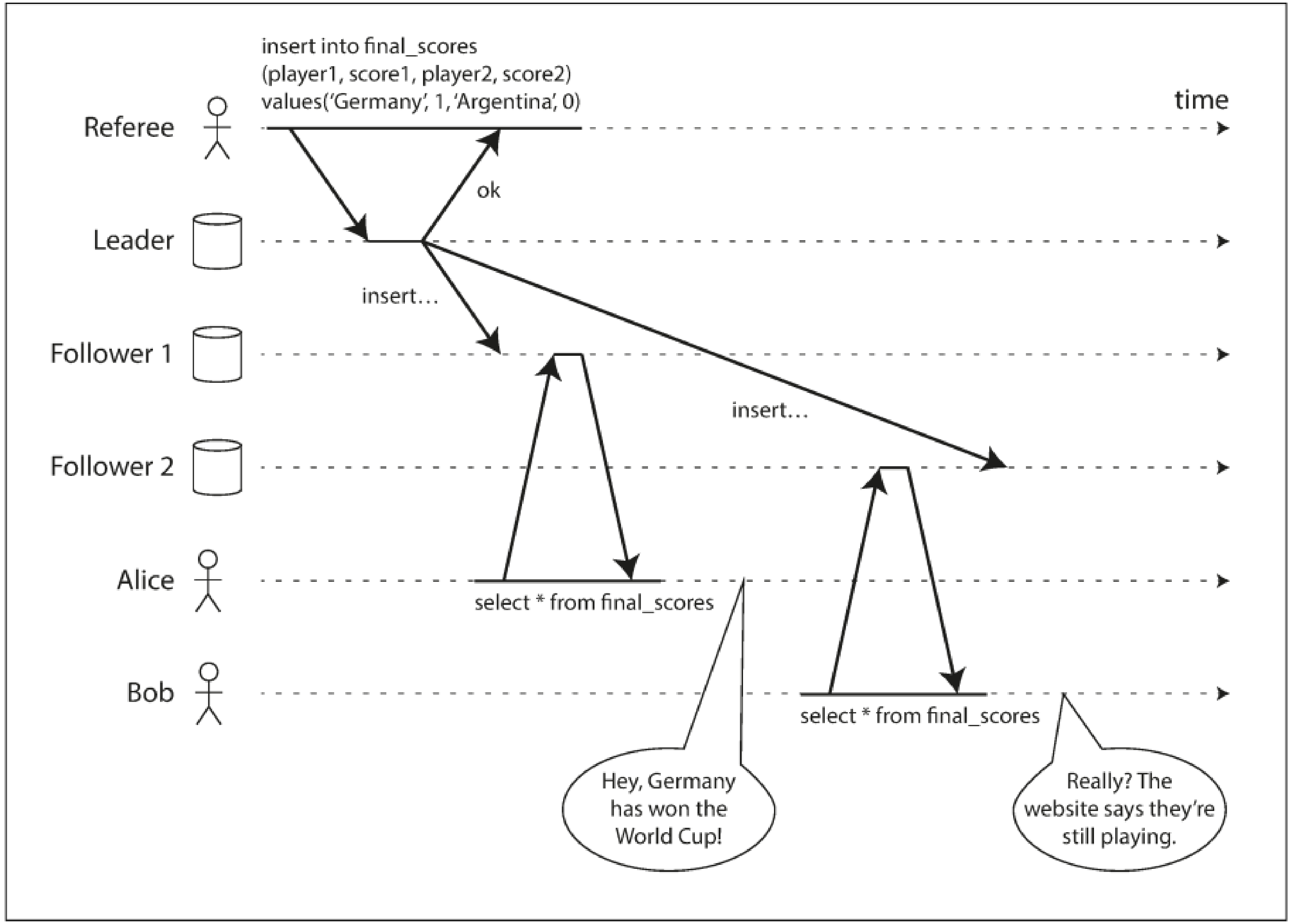

Figure 9-1. This system is not linearizable, causing football fans to be confused.

Linearizability

- Or strong consistency
- Instead of getting different answers from different replicas, linearizability abstracts away implementation details, and provides the illusion of a single replica
- A recency guarantee

Linearizability

- After ok, every one can read?

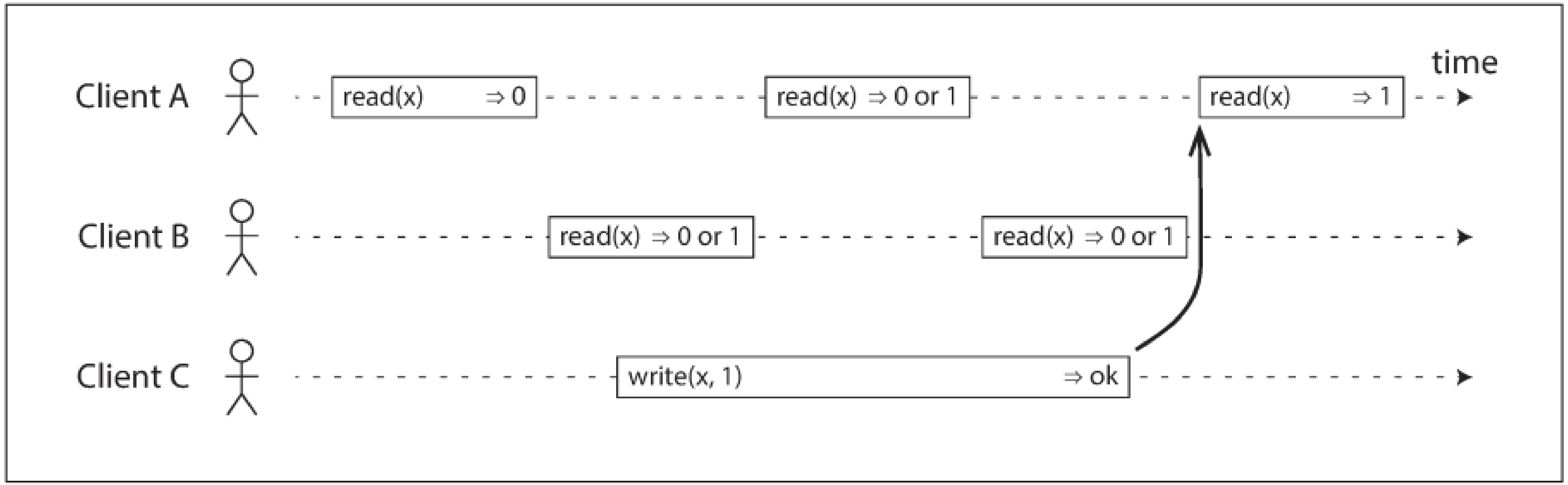

Figure 9-2. If a read request is concurrent with a write request, it may return either the old or the new value.

Linearizability

- After ok, every one can read?
- Not enough

Linearizability

- After ok, every one can read?
- Not enough

Linearizability

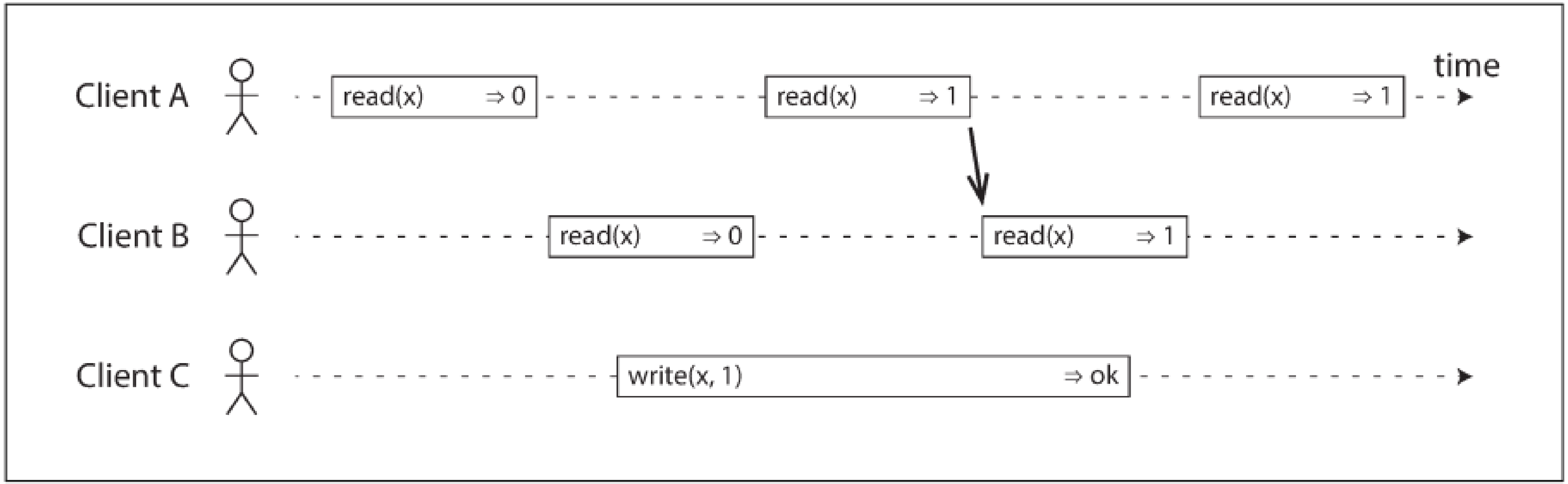

Figure 9-3. After any one read has returned the new value, all following reads (on the same or other clients) must also return the new value.

Linearizability

- New operation:
  - *cas(x, v_old, v_new)* $\Rightarrow$ *r*
  - *Compare and set*

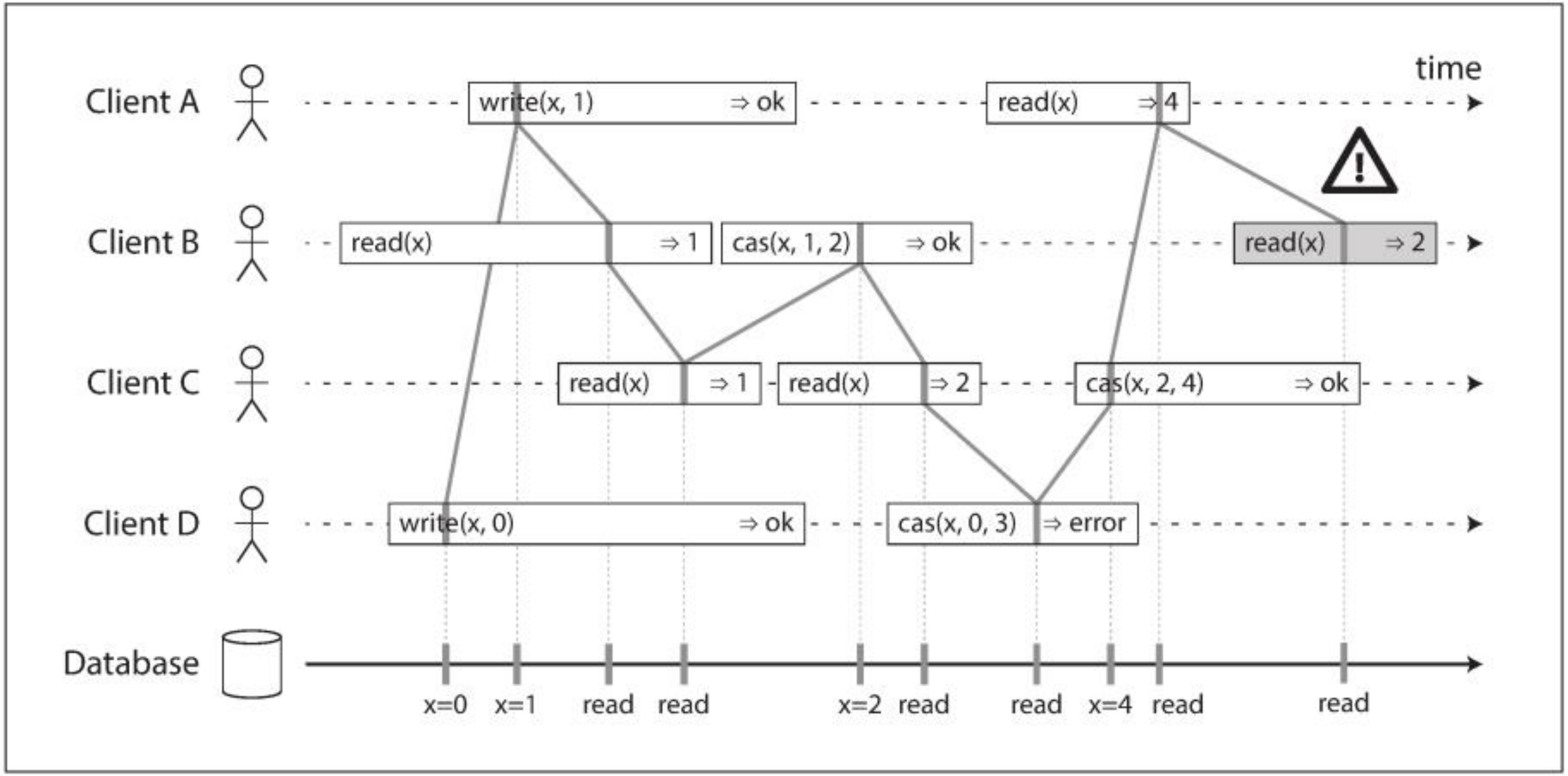

Figure 9-4. Visualizing the points in time at which the reads and writes appear to have taken effect. The final read by B is not linearizable.

Linearizability Versus Serializability

Linearizability -> Leader Election

Linearizability -> uniqueness Constraint

Cross-channel timing dependencies

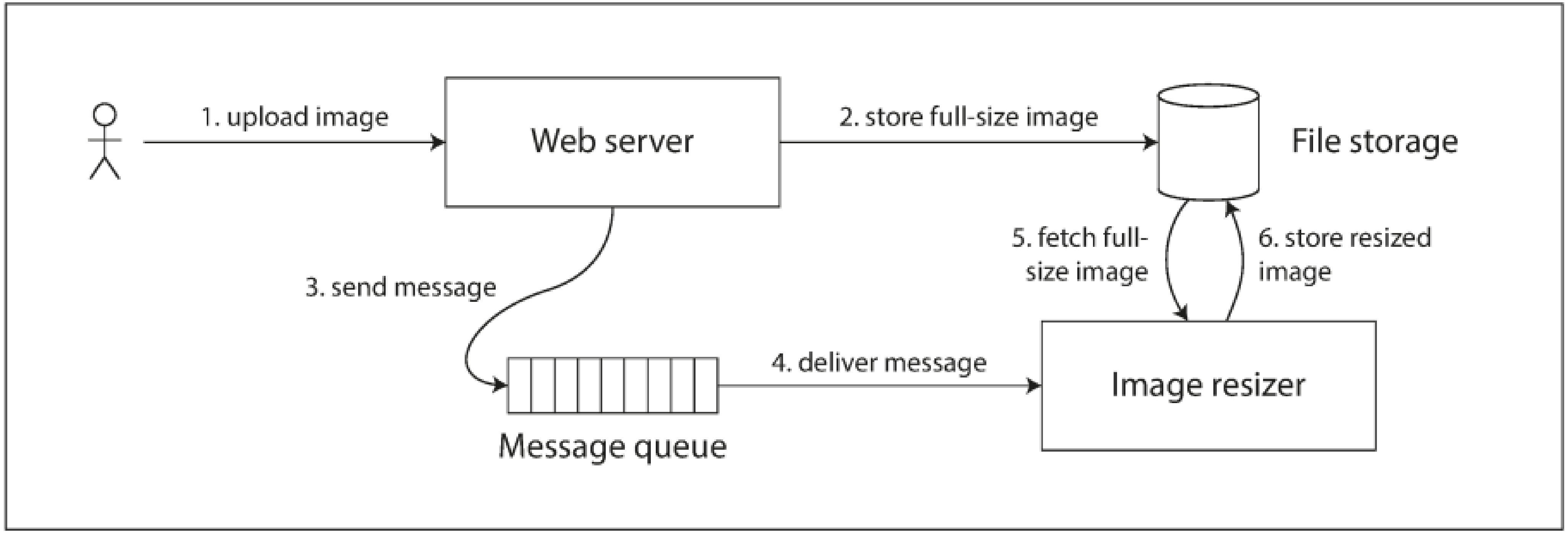

Figure 9-5. The web server and image resizer communicate both through file storage and a message queue, opening the potential for race conditions.

IMPLEMENTING LINEARIZABLE SYSTEMS

Linearizability without replication!

Single-leader replication

- potentially linearizable

Consensus algorithms

Multi-leader replication

- not linearizable

Leaderless replication

- w + r > n
  - *No!*☹

Leaderless replication

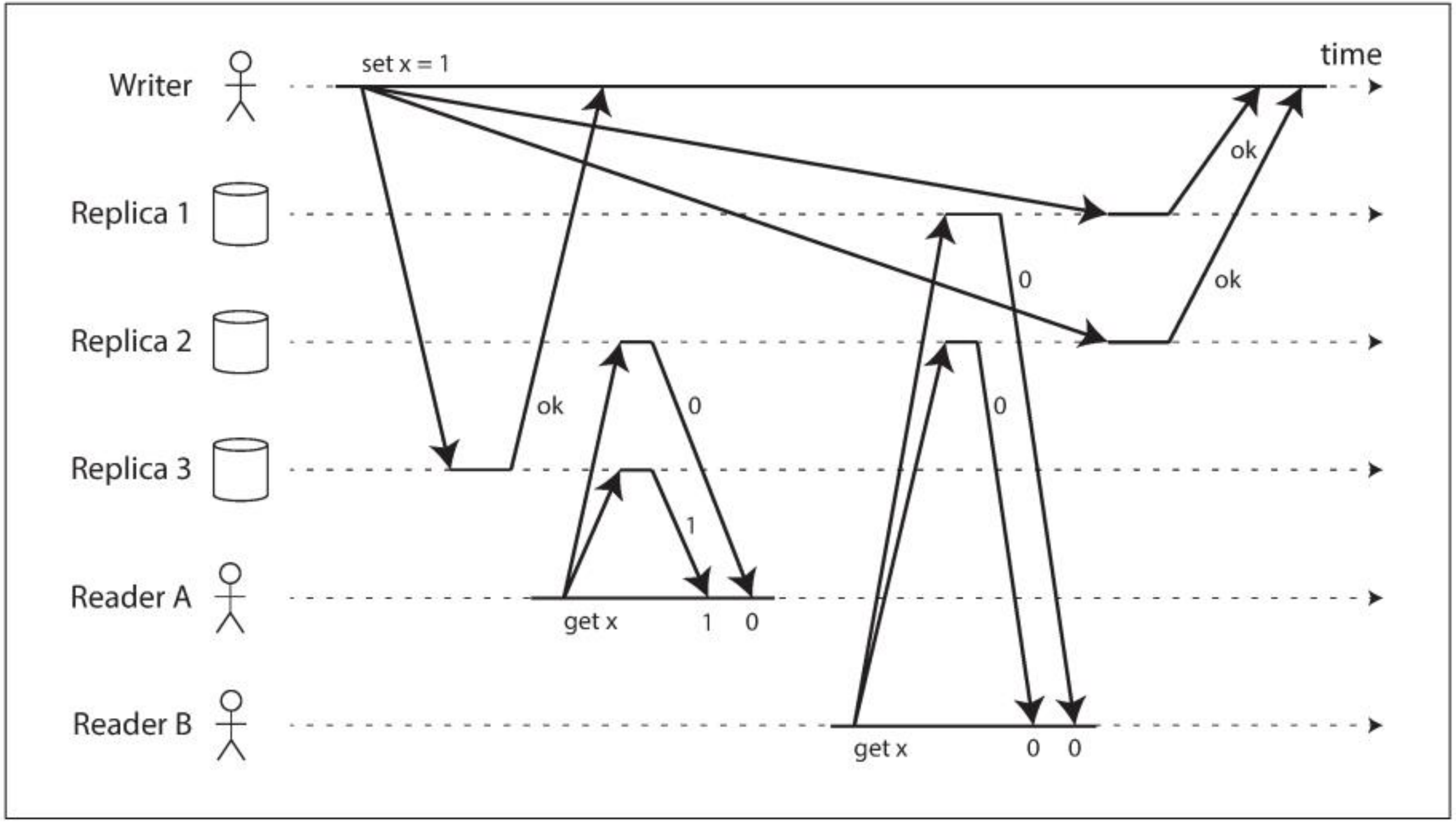

Figure 9-6. A nonlinearizable execution, despite using a strict quorum.

Leaderless replication

- Can have linearizability with:
  - *Synchronized read repair from reader*
  - *Writer must read the latest state before send its write*
- But only linearizable read and write operations can be implemented in this way; a linearizable compare-and-set operation cannot

THE COST OF LINEARIZABILITY

CAP Theorem

- Consistency
- Availability
- Partition Tolerance

- you can only guarantee two out of these properties

CAP Theorem

- Consistency
  - *Every read operation receives the most recent write or an error.*
  - *= Linearizability*
  - *It is **not** Consistency in ACID*

CAP Theorem

- Availability
  - *Every request to non-failing node receives a valid response*
  - *though not necessarily the most recent data*
  - *without guarantee that it contains the latest write.*

CAP Theorem

- Partition Tolerance
  - *The system continues to function (perhaps with limited capability) even if there is a "partition"*
  - *Network partition = a network fault that prevents some nodes from communicating with others*
  - *It's not partitioning in chapter 6*

CAP Theorem

- In network partition
  - *We need choice between linearizability or availability*

Limits of CAP Theorem

Costs of Linearizability

- Single computers may be not linearizability
- Many distributed databases that choose not to provide lineariza- ble guarantees: they do so primarily to increase performance, not so much for fault tolerance [46].
- Linearizability is slow—and this is true all the time, not only during a network fault.

ORDERING GUARANTEES

Causality

- Consistent Prefix Reads
- Detecting Concurrent Writes
- Snapshot Isolation
- …

Causality

- Is a partial order

Causal Consistency

Causal Consistency

- linearizability implies causality

Try to implement causal consistency

- Sequence Number Ordering
  - *Each node can generate its own independent set of sequence numbers.*
  - *You can attach a timestamp from a time-of-day clock (physical clock) to each operation*
  - *You can preallocate blocks of sequence numbers.*

Try to implement causal consistency

- Lamport timestamps
  - *(counter, node Id)*

Try to implement causal consistency

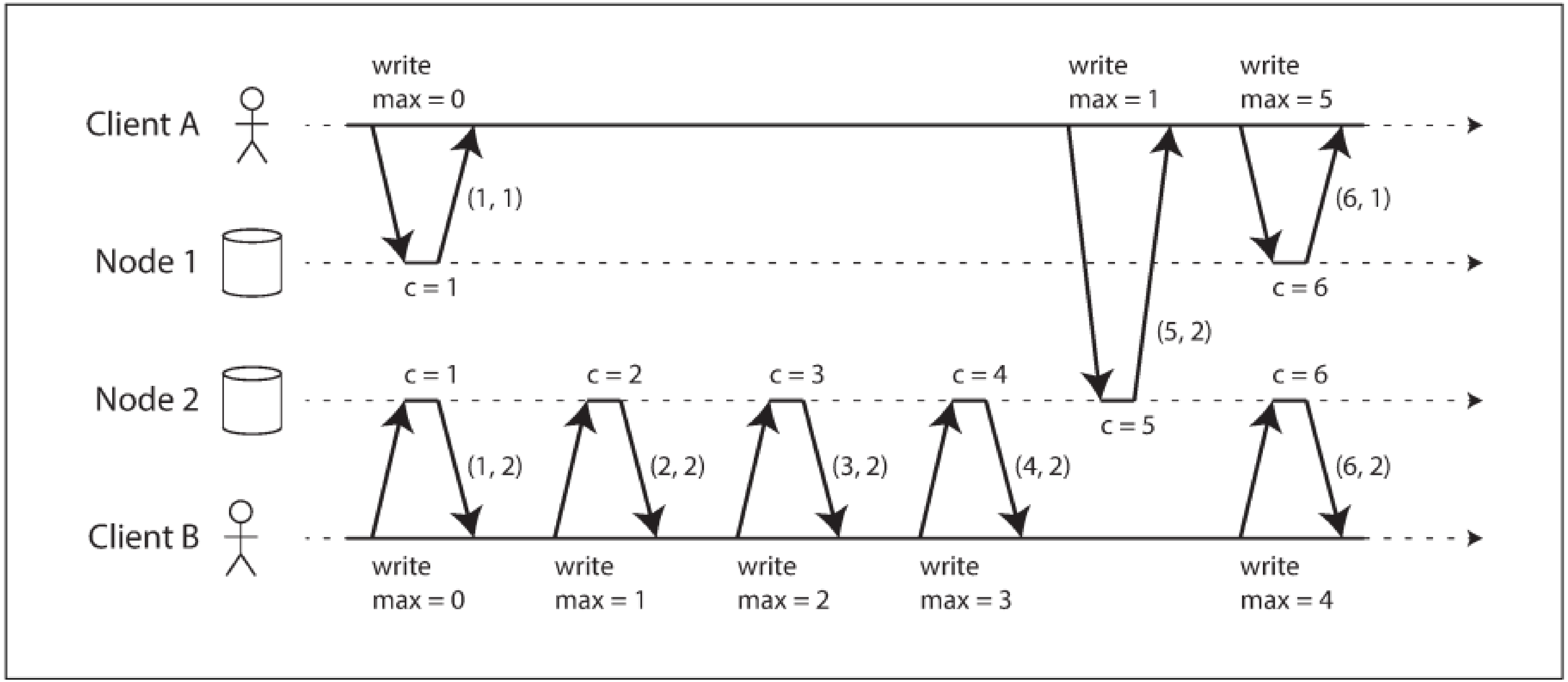

Figure 9-8. Lamport timestamps provide a total ordering consistent with causality.

Try to implement causal consistency

- Lamport timestamps Vs. Version Clock

Is timestamp ordering is enough?

- Username uniqueness constraint
- Try to solve with total order broadcast

TOTAL ORDER BROADCAST

Total order broadcast

- Reliable delivery
  - *No messages are lost: if a message is delivered to one node, it is delivered to all nodes.*
- Totally ordered delivery
  - *Messages are delivered to every node in the same order.*

Total order broadcast Vs. Linearizability

- Total order broadcast is asynchronous:
  - *messages are guaranteed to be delivered reliably in a fixed order, but there is no guarantee about when a message will be delivered (so one recipient may lag behind the others).*
- By contrast, linearizability is a recency guarantee:
  - *a read is guaranteed to see the latest value written*

معادل بودن مسئله‌ها

Total order broadcast => Linearizability

- implement such a linearizable compare-and-set operation using total order broadcast

Linearizability => Total order broadcast

- Assume we have an atomic increment-and-get operation with linearizable register

Linearizability => Consensus

DISTRIBUTED TRANSACTION

Atomicity

Two phase commit

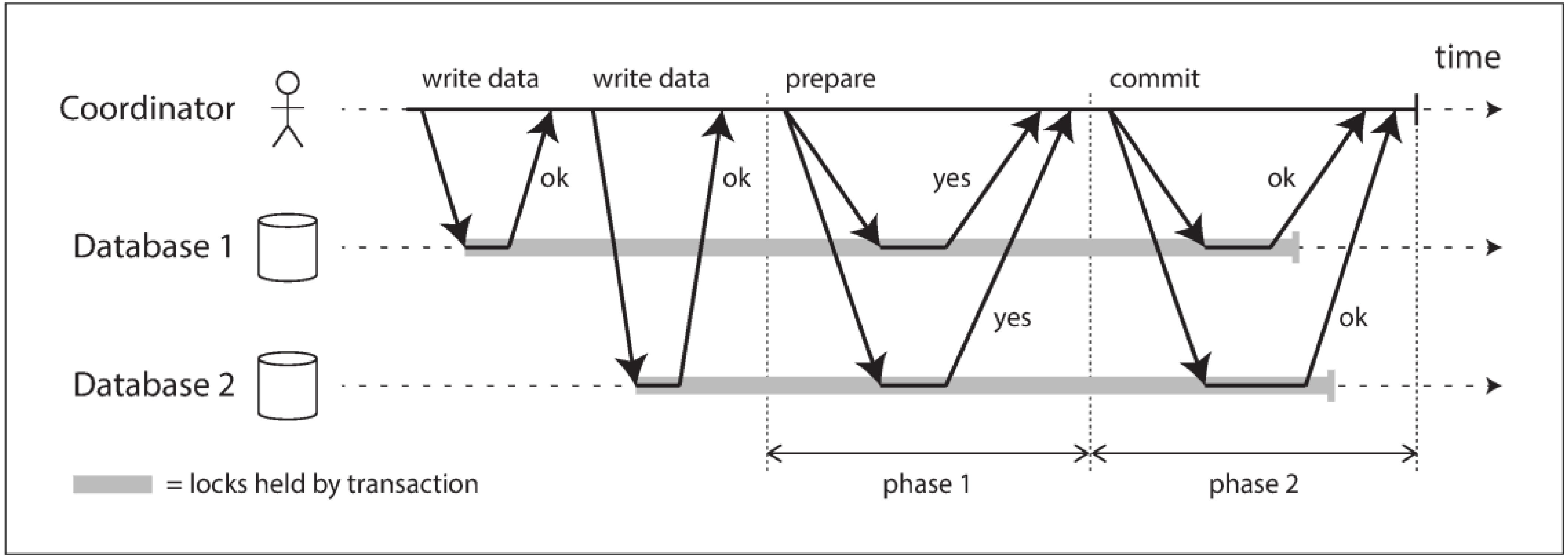

Figure 9-9. A successful execution of two-phase commit (2PC).

Two phase commit - promise

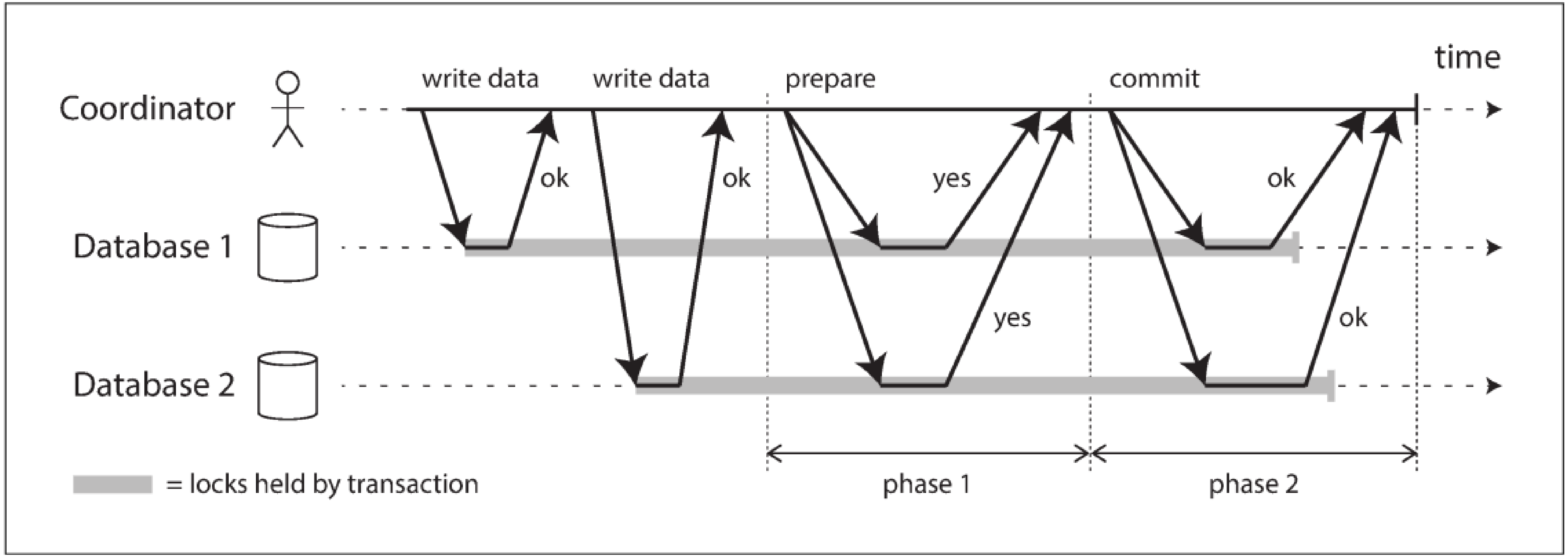

Figure 9-9. A successful execution of two-phase commit (2PC).

Two phase commit

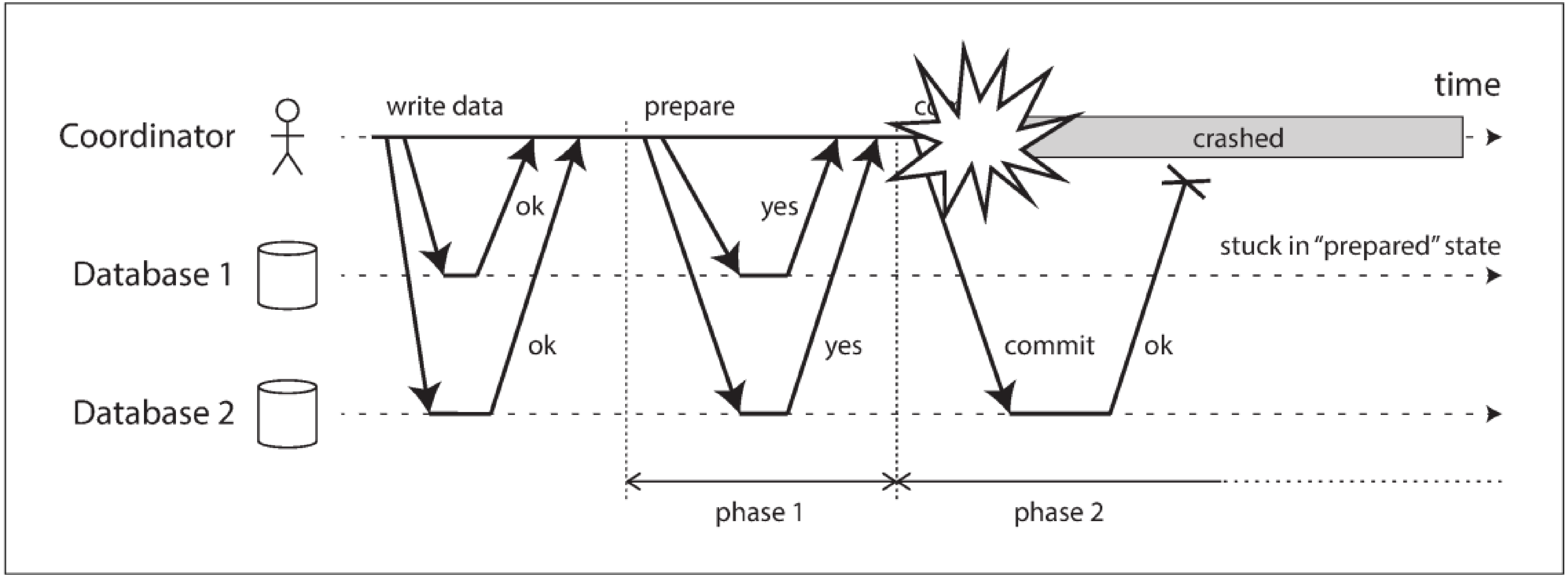

Figure 9-10. The coordinator crashes after participants vote "yes." Database 1 does not know whether to commit or abort.