بسم الله الرحمن الرحیم

تکنولوژی کامپیوتر

جلسه‌ی چهاردهم
رپلیکیشن بدون رهبر

جلسه گذشته

رپلیکیشن

رپلیکیشن

- Replication is keeping an entire copy of the data on multiple machines
- We will assume that all of data can be stored in a single machine

Leader / Follower

- Replica / Node
- Leader (Or Master / Primary)
  - *One of replicas will be Leader*
  - *All writes must go through the leader*
- Follower
  - *Catch data from leader*
  - *reads can come from any replica*

Multi-Leader Replication

- each leader also acts as a follower to the other leaders
- Use cases?
  - *Multi-data center operation*
  - *Clients with offline operation*
  - *Collaborative editing*

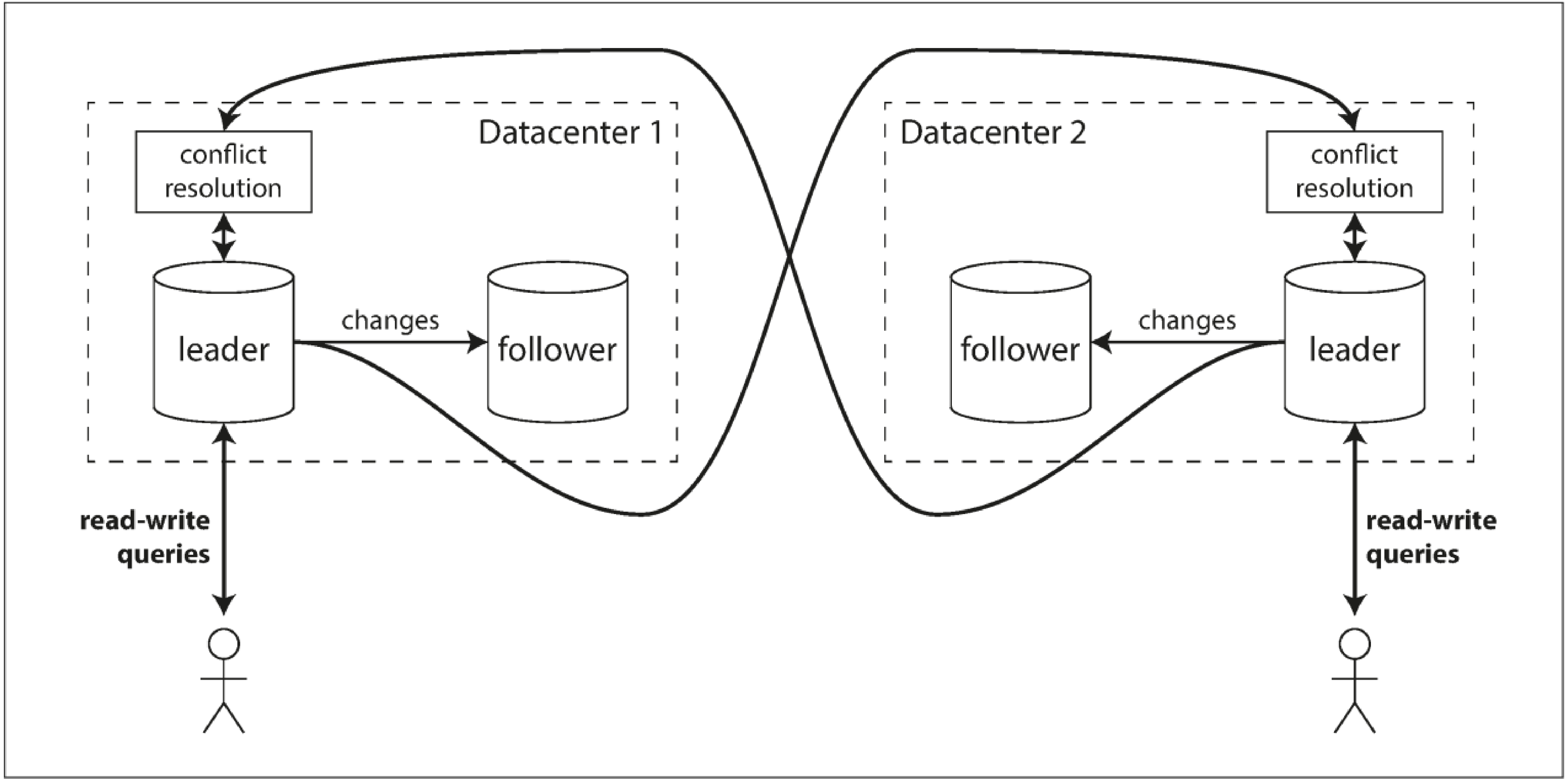


Figure 5-6. Multi-leader replication across multiple datacenters.

Handling Write Conflicts

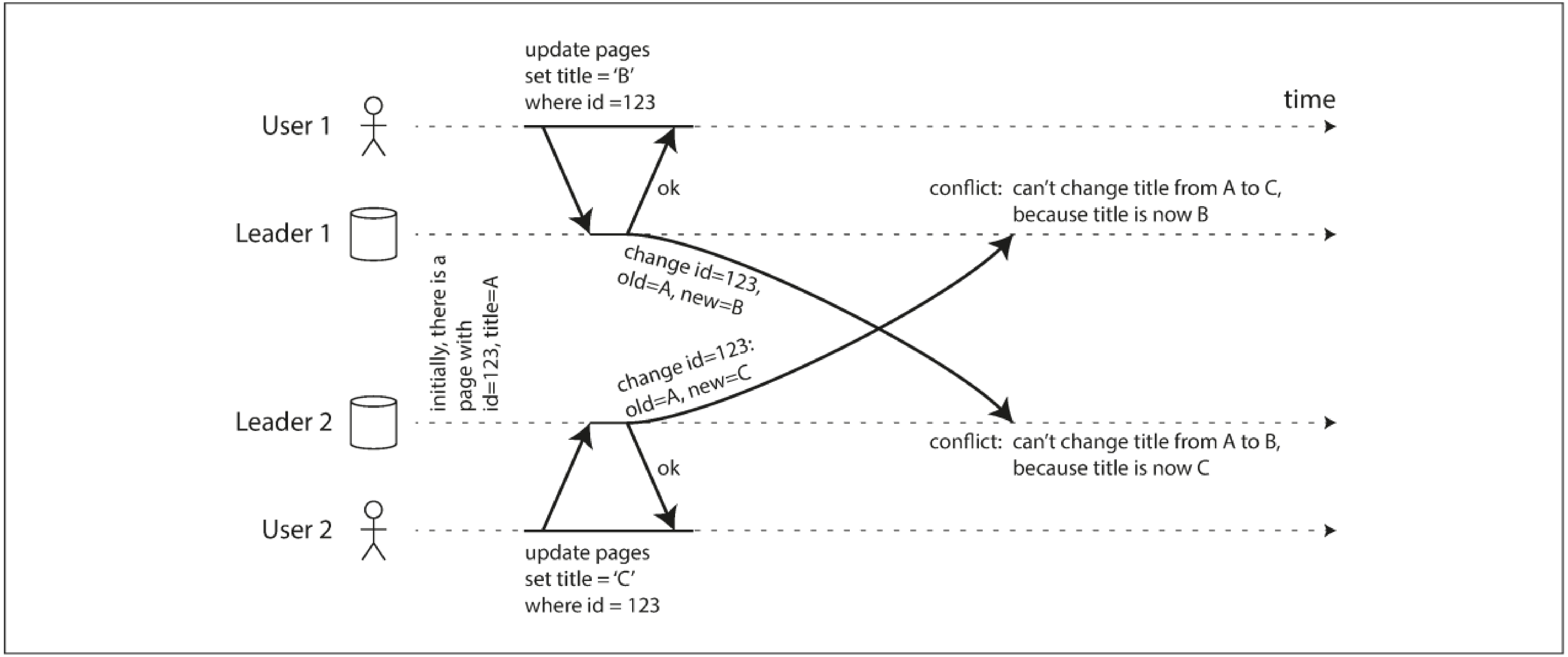

update pages
set title = 'B'
where id =123
time
User 1
ok
conflict: can't change title from A to C, because title is now B
Leader 1
change id=123, old=A, new=B
initially, there is a page with id=123, title=A
change id=123: old=A, new=C
Leader 2
conflict: can't change title from A to B, because title is now C
ok
User 2
update pages
set title = 'C'
where id = 123

Figure 5-7. A write conflict caused by two leaders concurrently updating the same record.

Handling Write Conflicts

- Downside of multi-leader is that write conflicts can happen, requiring conflict resolution
- Why it’s hard and have bad User Experience?
  - Conflicts are generally detected asynchronously, so too late to prompt user

Handling Write Conflicts

- Conflict avoidance
  - *Assigning each record a home datacenter works until a failure or they move*
  - *If you need to change the designated leader for a record -> conflict may happen* ☹

جلسه‌ی جدید

WRITE CONFLICTS

Handling Write Conflicts

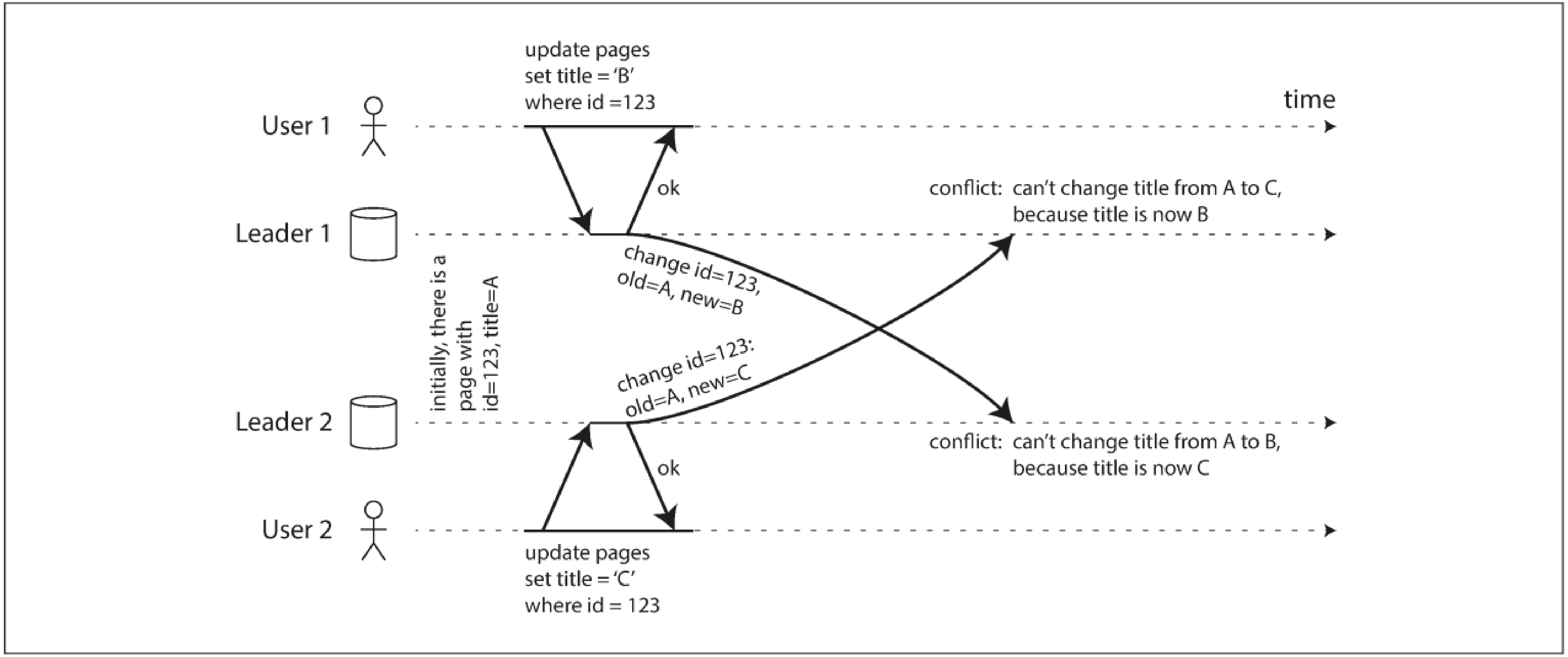


Figure 5-7. A write conflict caused by two leaders concurrently updating the same record.

Converging toward a consistent state

- Whan happen if each replica simply applied writes in the order that it saw the write?

Converging toward a consistent state

- Last Write Wins (LWW)
  - *Each write request contains a unique id/timestamp*
  - *The request with bigger id wins*
  - *It is dangerously prone to data loss*
  - *if there are several concurrent writes to the same key:*
    - all reported as successful to the client
    - But only one of the writes will survive and the others will be silently discarded

LWW

- some situations, such as caching, in which lost writes are perhaps acceptable
- If losing data is not acceptable, LWW is a poor choice for conflict resolution
- The only safe way of using a database with LWW is to ensure that a key is only written once and thereafter treated as immutable

Pre-define arbitrary replica precedence rules

- Give each replica a unique ID
- Writes that originated at a higher-numbered replica always take precedence over writes that originated at a lower-numbered replica.
- This approach also implies data loss.

Merge Values together?

- Somehow merge the values together
  - *e.g., order them alphabetically*
  - *then concatenate them*
- Example:
  - *Concurrent writes: B and C*
  - *Conflict resolution: B/C*

Record log and resolve later

Custom conflict resolution login

- On write
- On read

Automatic Conflict Resolution Datatypes?

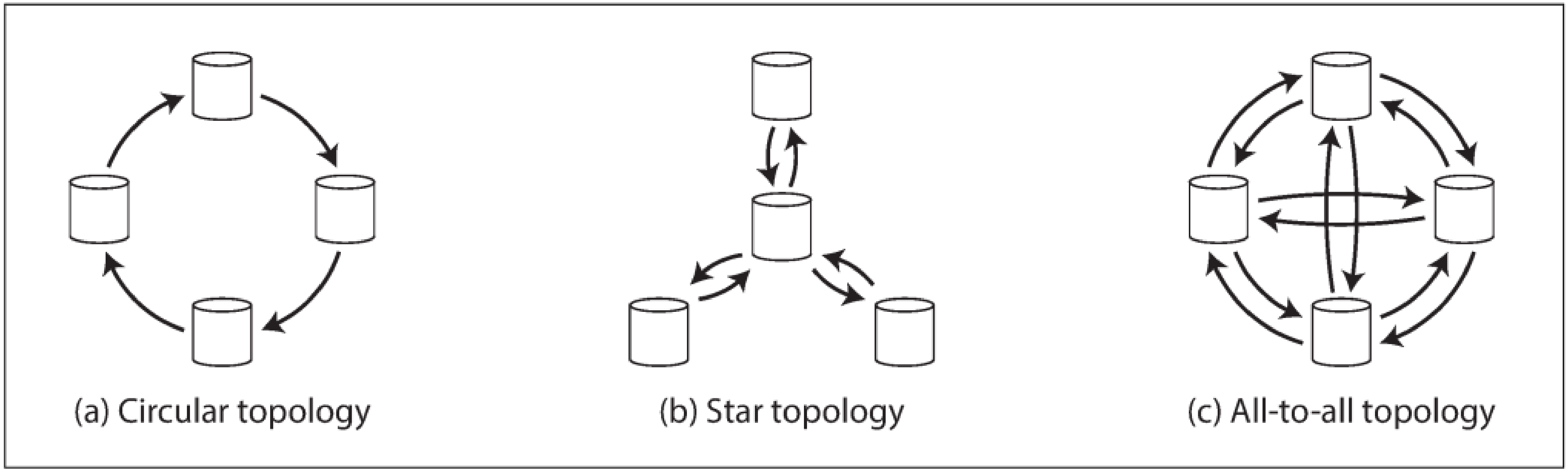
- Conflict-free replicated datatypes (CRDTs)
  - *Family of data structures for sets, maps, ordered lists, counters, … that can concurrently edited and automatically resolve conflicts in sensible ways.*
- Mergeable persistent data structures
  - *Track history and merge like git.*
- Operational transformation
  - *Conflict resolution of collaborative editing (google docs)*
  - *Concurrent editing of an ordered list of items*

فقط نوشتن همزمان یک فیلد کانفلیکت هست؟

- Meeting room booking system

- در جلسات آینده در مورد روش‌های حلش بیشتر حرف می‌زنیم.

Multi-Leader Replication Topologies



(a) Circular topology

(b) Star topology

(c) All-to-all topology

Figure 5-8. Three example topologies in which multi-leader replication can be set up.

Multi-Leader Replication Topologies

- In circular and star
  - *node receive writes from one node should forward it to other node.*
  - *How to prevent infinite replication loops?*
  - *What happed if one node fails?*

Multi-Leader Replication Topologies

- All-to-all topologies

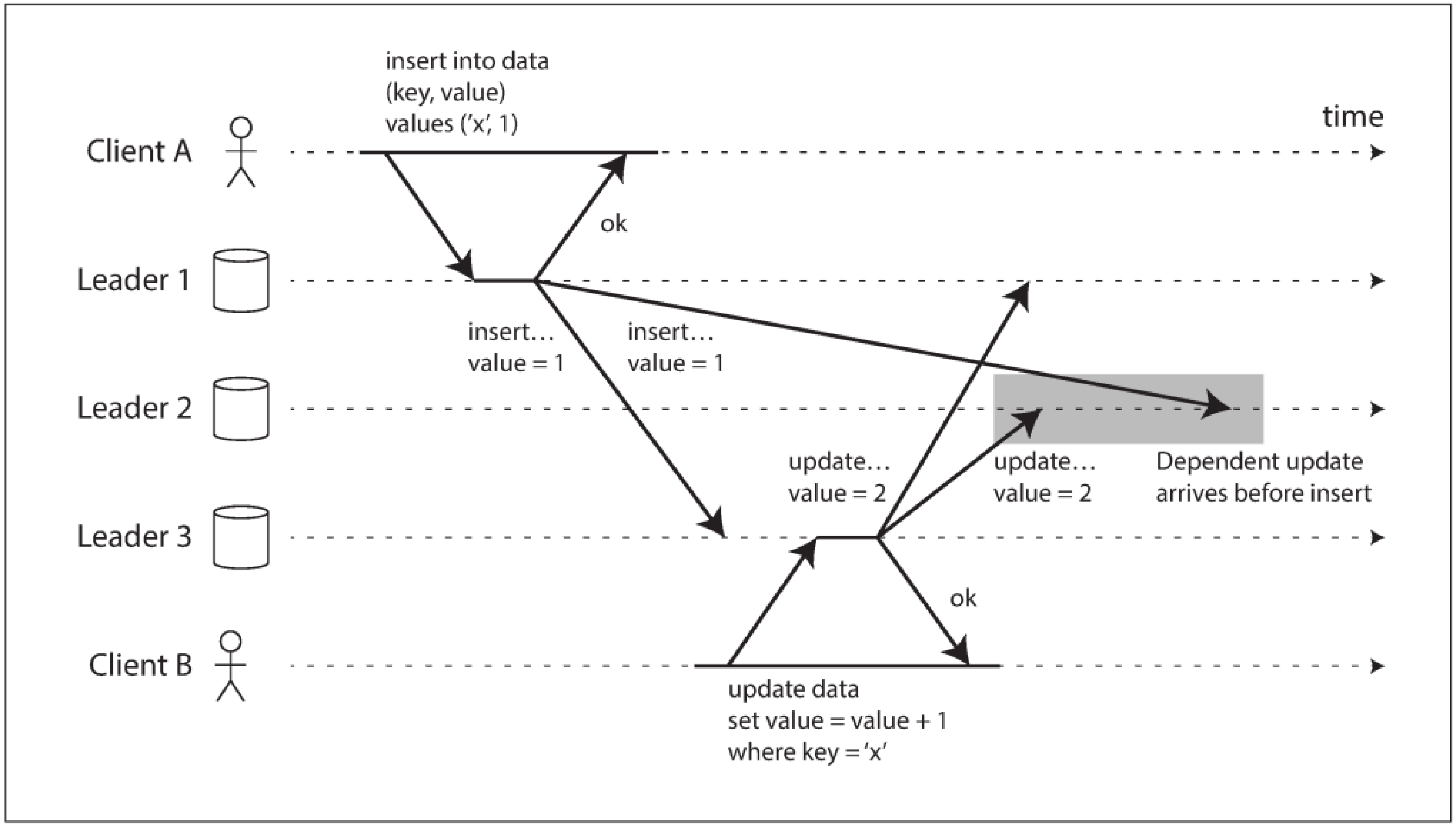


Figure 5-9. With multi-leader replication, writes may arrive in the wrong order at some replicas.

LEADERLESS REPLICATION

- Leader based replication: client send data to one node, and the database takes care of copying that write to the other replicas.
- Writes send to multiple replicas
  - *Client directly send writes to replicas*
  - *Or a coordinator handles that*

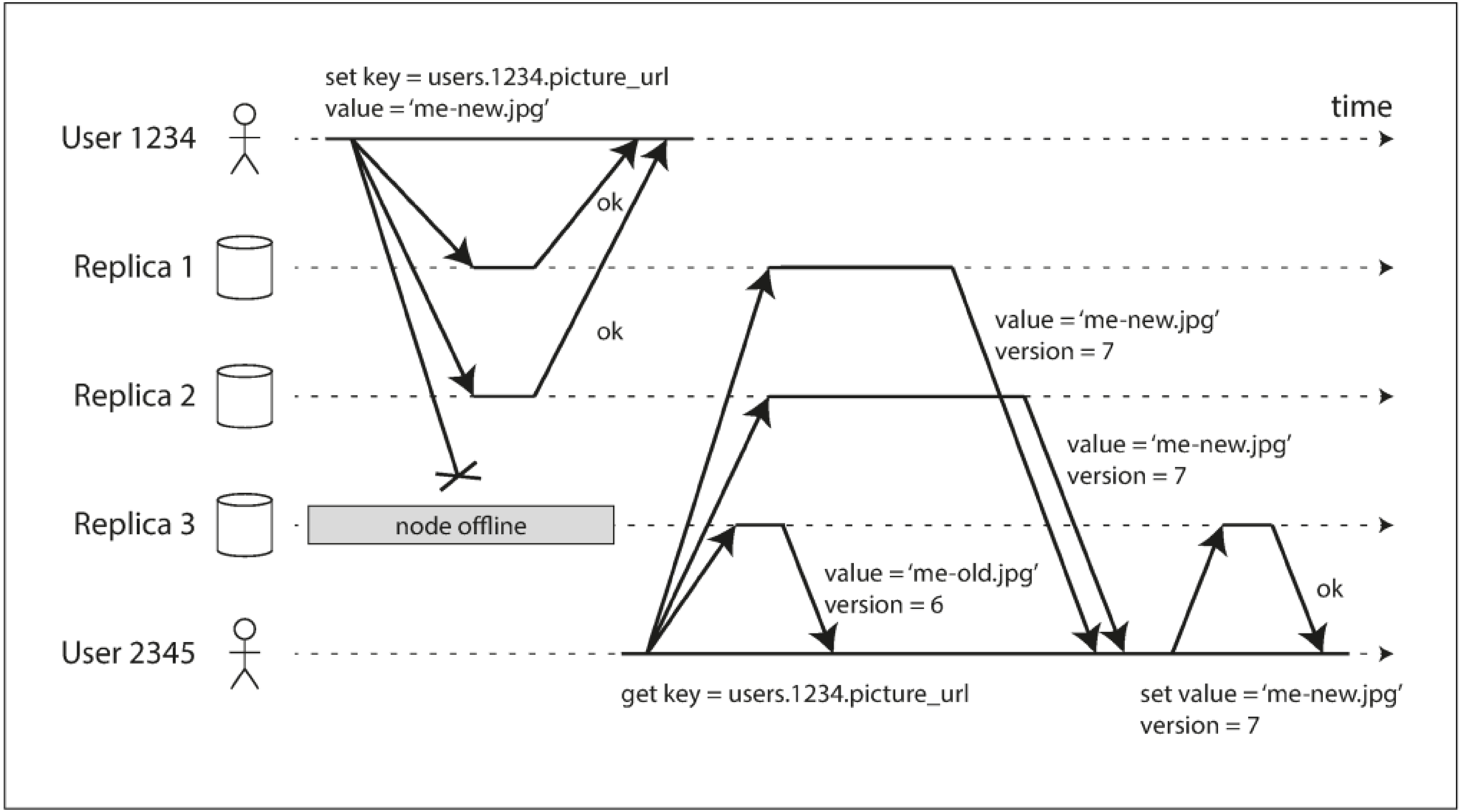


Figure 5-10. A quorum write, quorum read, and read repair after a node outage.

Writing to the Database When a Node Is Down

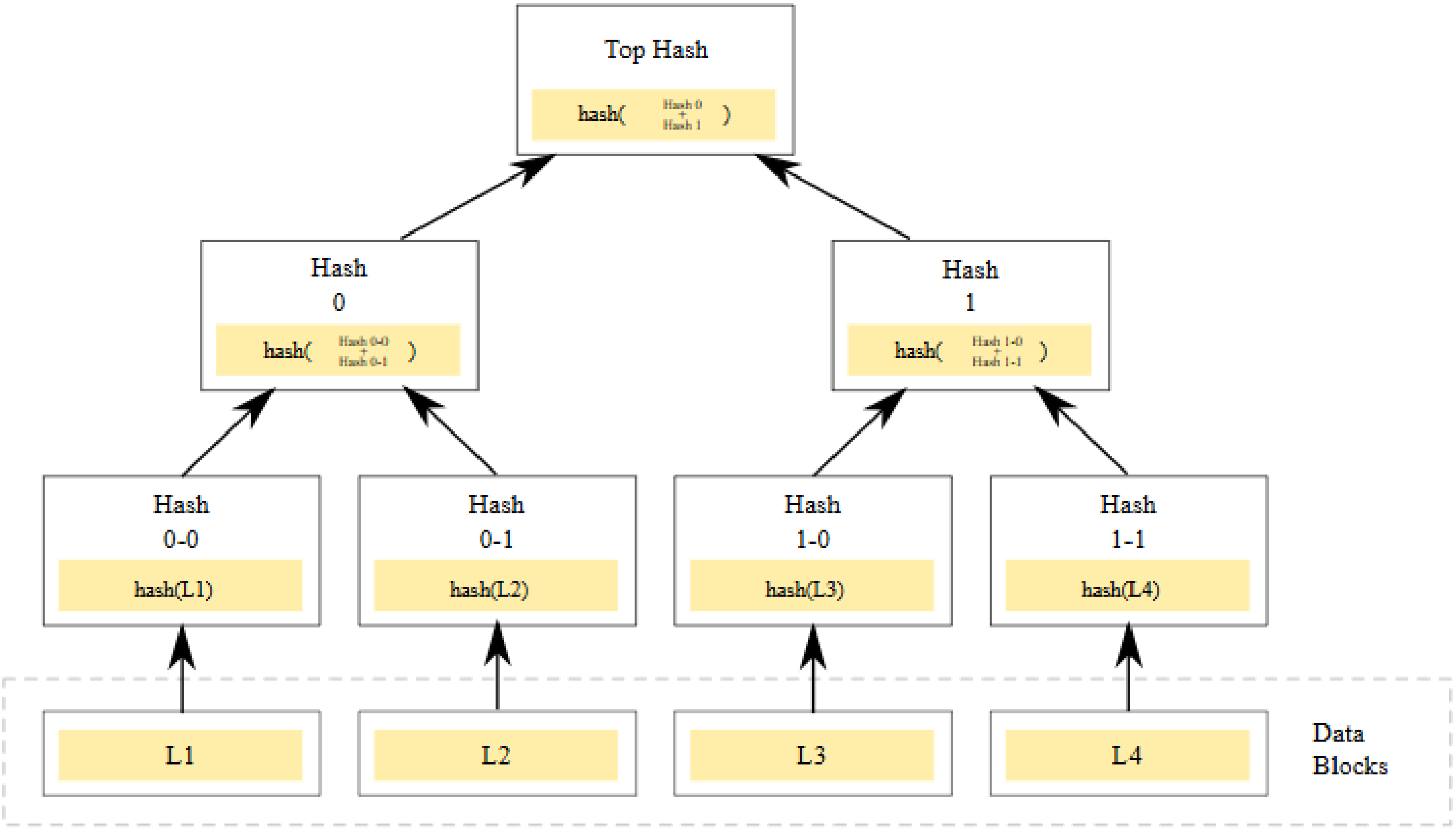
- you can successfully continue
  - *When nodes come back up, some of the data is stale*
- How we can detect stale on read?

Catching up on stale data from missed writes

- Read Repair
  - *when a client detects a stale read, it writes the newer value back*
- Anti-entropy process
  - *background process that searches for differences between replicas and corrects them*
  - *Merkle tree can be used to find differences*

Catching up on stale data from missed writes

- Anti-entropy process
  - *Merkle tree:*



Quorums for readings and writings

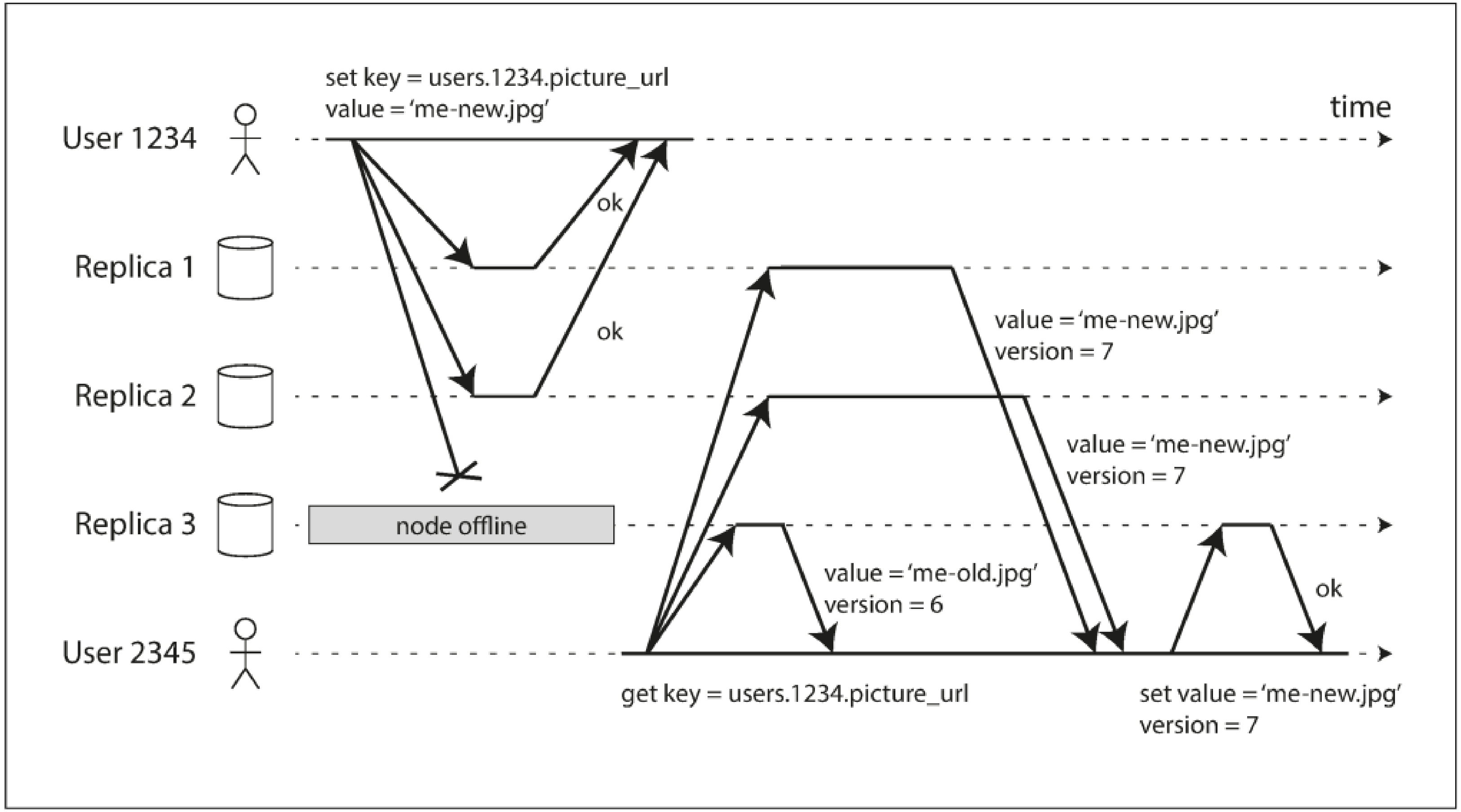


Figure 5-10. A quorum write, quorum read, and read repair after a node outage.

Quorums for readings and writings

- N: number of nodes
- n: number of replicas that particular data is stored
- w: minimum number of OKs that write considered as successful
- r: minimum number of OKs that read considered as successful
  - **$w + r > n$**

Quorums for readings and writings

- n, w, r?
- Usually they are configurable.
- A common choice:
  - *n an odd number*
  - *w, r = (n+1)/2*

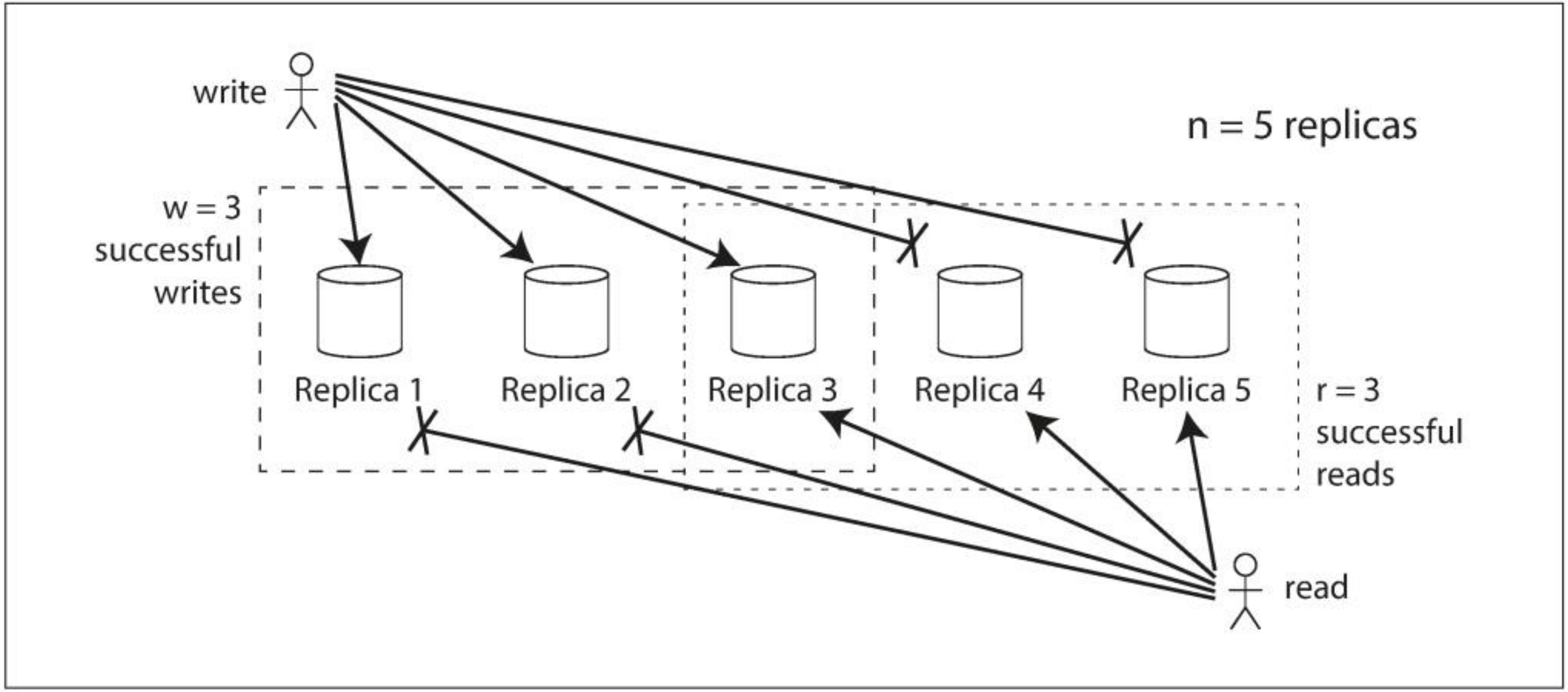


Figure 5-11. If w + r > n, at least one of the r replicas you read from must have seen the most recent successful write.

Limitations of Quorum Consistency

- Choosing w + r <= n risks stale reads.
- Sloppy quorums (next section)
- Concurrent writes
- If a write operation fails to get a quorum, a subset of nodes won’t roll back, so will incorrectly have the newer value
- If a failed node A is restored from node B, anything stale on B is now stale on A, and both A and B can contribute toward a read quorum

Sloppy Quorums and Hinted Handoff

- In a large cluster, a network interruption could easily cause there to be not enough nodes for a quorum
- Is it better to simply return errors, or to try a workaround?
- Suppose N > n and we don't have w ready nodes from n nodes.
  - ***Sloppy quorum:*** *use other N-n nodes for write.*
  - *When connectivity is restored, temporary nodes use **hinted handoff** to write the data back to nodes where it belongs.*

Multi-Datacenter Operation

- Given that communication between datacenters is expected to be slow
- Some configured to send cross-datacenter writes asynchronously
- Riak limits initial replication to be within a datacenter, and uses a multi-leader strategy between datacenters.

Concurrent Writes

- Can happen on multi-leader and leaderless replication
- Also during read repair and hinted handoff
- Last Write Wins (LWW)

Concurrent Writes

- What the meaning of concurrent write?
- If A was aware of write B, we say B happened before A.
- If B was aware of A, then A happened before B.
- In all other situations where neither A nor B were aware of the other, we define A and B as concurrent writes.

Version Vector – Single node Algorithm

- Server maintains a version with each key
- Clients must do a read, which includes version(s) and value(s), before they write
- Clients must merge multiple values read before doing a write
- Server increments max version each write
- Server receiving a write can discard data from that version or older, but keeps newer data

Version Vector – Single node Algorithm

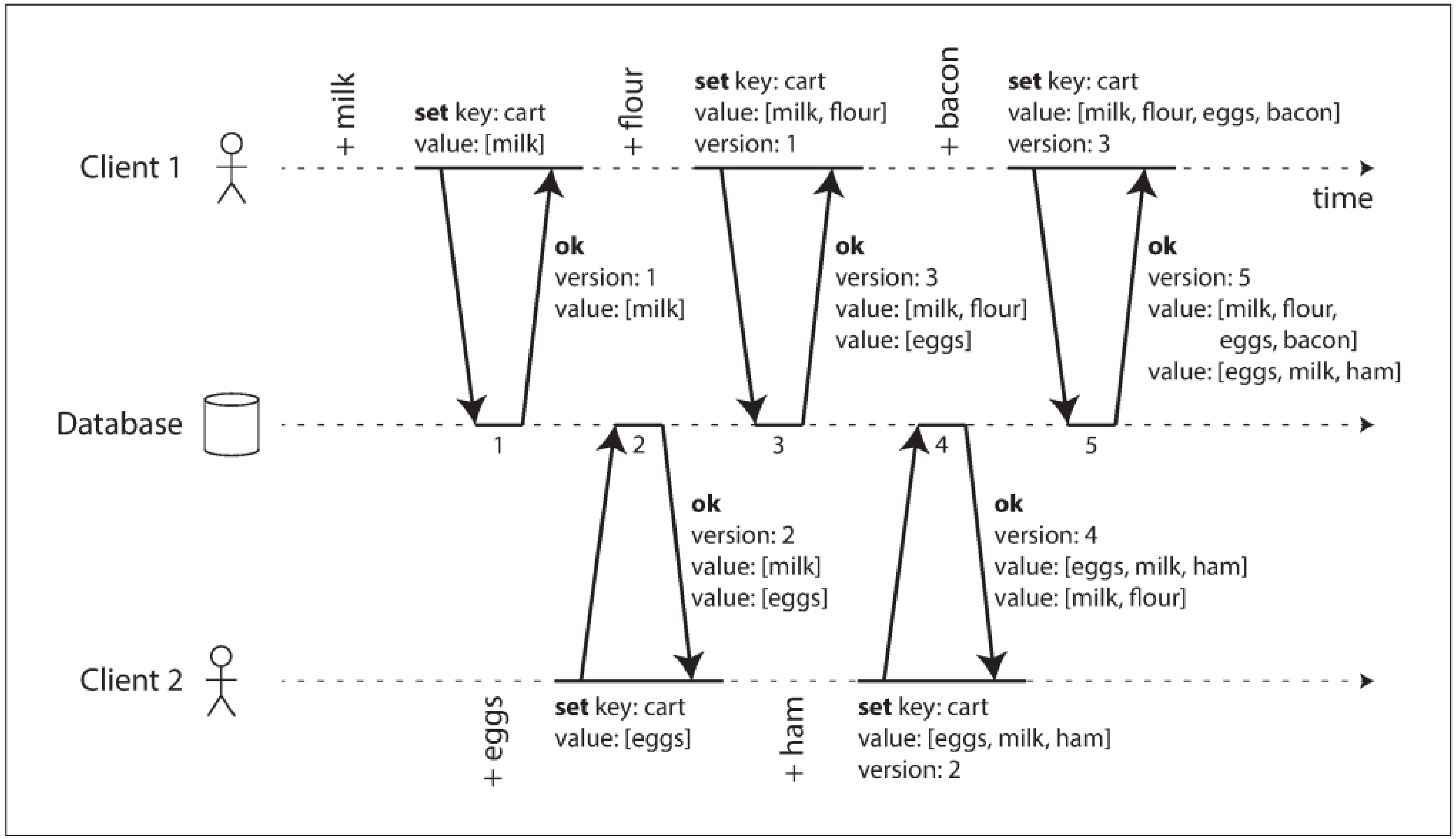


Figure 5-13. Capturing causal dependencies between two clients concurrently editing a shopping cart.

Version Vector – Single node Algorithm

- Siblings versions
- Merging concurrently written values?

Version Vector – Multiple replica

- Imagine you have N replicas: Replica A, Replica B, Replica C, …
- Each replica keeps track of a local integer counter.
- For example:
  - *Replica A has version counter vA.*
  - *Replica B has version counter vB.*
  - *Replica C has version counter vC.*

Version Vector – Multiple replica

- The version vector is a collection of each replica’s version counters
- instead of a single integer, you have a small list/tuple of integer counters—one counter for each replica.
- Example: if you have three replicas, your version vector might look like [vA=5, vB=3, vC=7].

پارتیشنینگ

Partitioning

- Why?
- Partition / shard / region / vnode

Partitioning on K/V store

- Partitioning by Key range

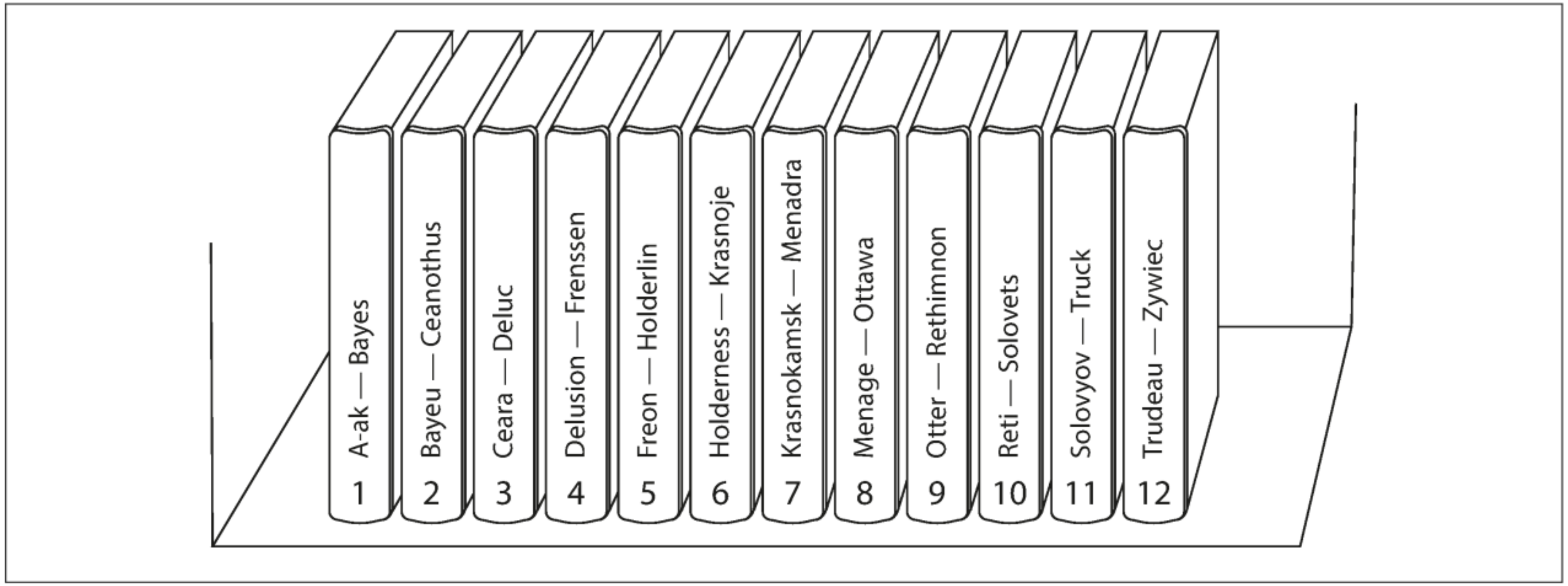


Figure 6-2. A print encyclopedia is partitioned by key range.

Partitioning on K/V store

- Partitioning by Key range
  - *Good for range queries*
  - *Example: an application that store sensors*
    - Key: timestamp
    - Value: measurement
  - *Hot Spots!*

Partitioning on K/V store

- Partitioning by Hash key

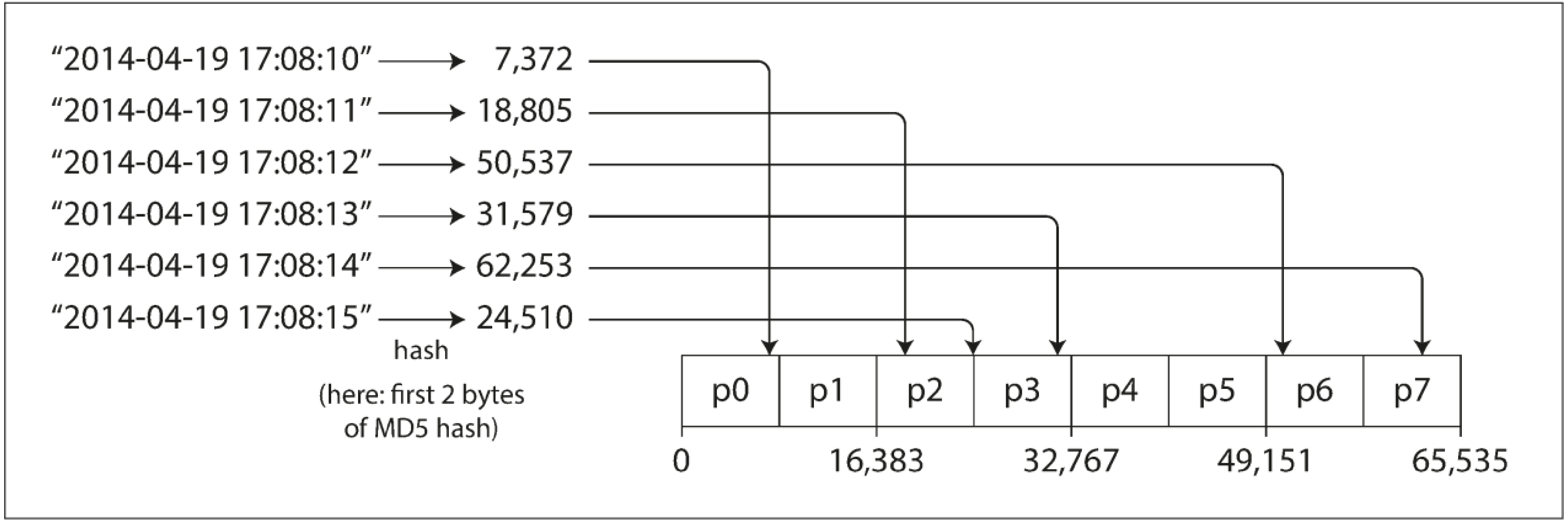


Figure 6-3. Partitioning by hash of key.

Partitioning on K/V store

- Partitioning by Hash key
  - *Hash to node?*
  - *What happen if we add new node?*