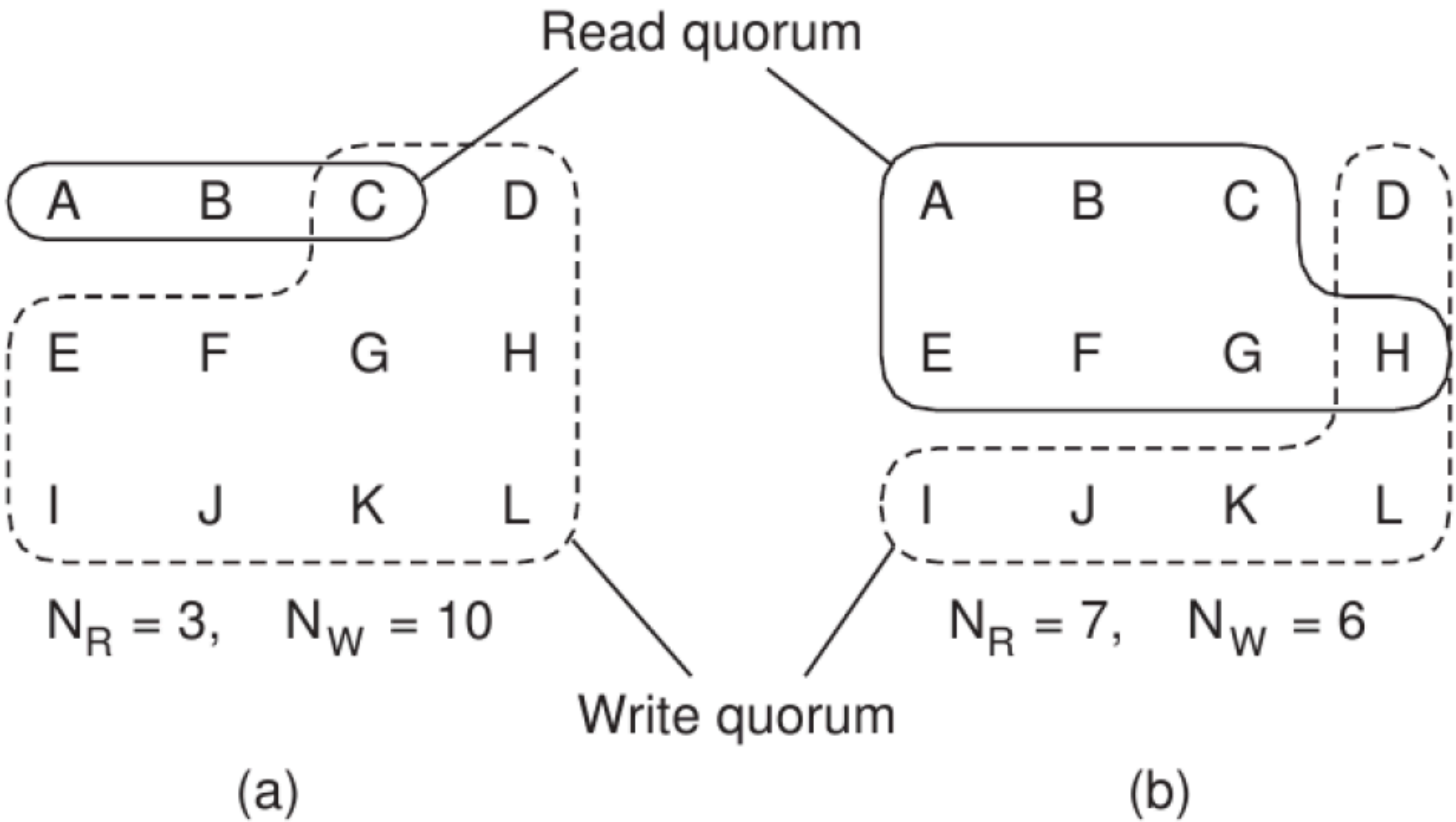
# Quorum Protocols

- Proposed by Gifford in 1979
- **Quorum-based protocols** guarantee that each operation is carried out in such a way that a *majority vote* (a quorum) is established.
  - *Write quorum W*: the number of replicas that need to acknowledge the receipt of the update to complete the update
  - *Read quorum R*: the number of replicas that are contacted when a data object is accessed through a read operation

# Quorum Systems

- Formally, a **quorum system** $S = \{S_1, \ldots, S_N\}$ is a collection of **quorum sets** $S_i \subseteq U$ such that two quorum sets have at least an element in common
- For replication, we consider two quorum sets, a **read quorum** R and a **write quorum** W
- **Rules**:
    1. Any read quorum must overlap with any write quorum
    2. Any two write quorums must overlap
- U is the set of replicas, i.e., $|U| = N$

# Quorum Examples



# Quorum Examples

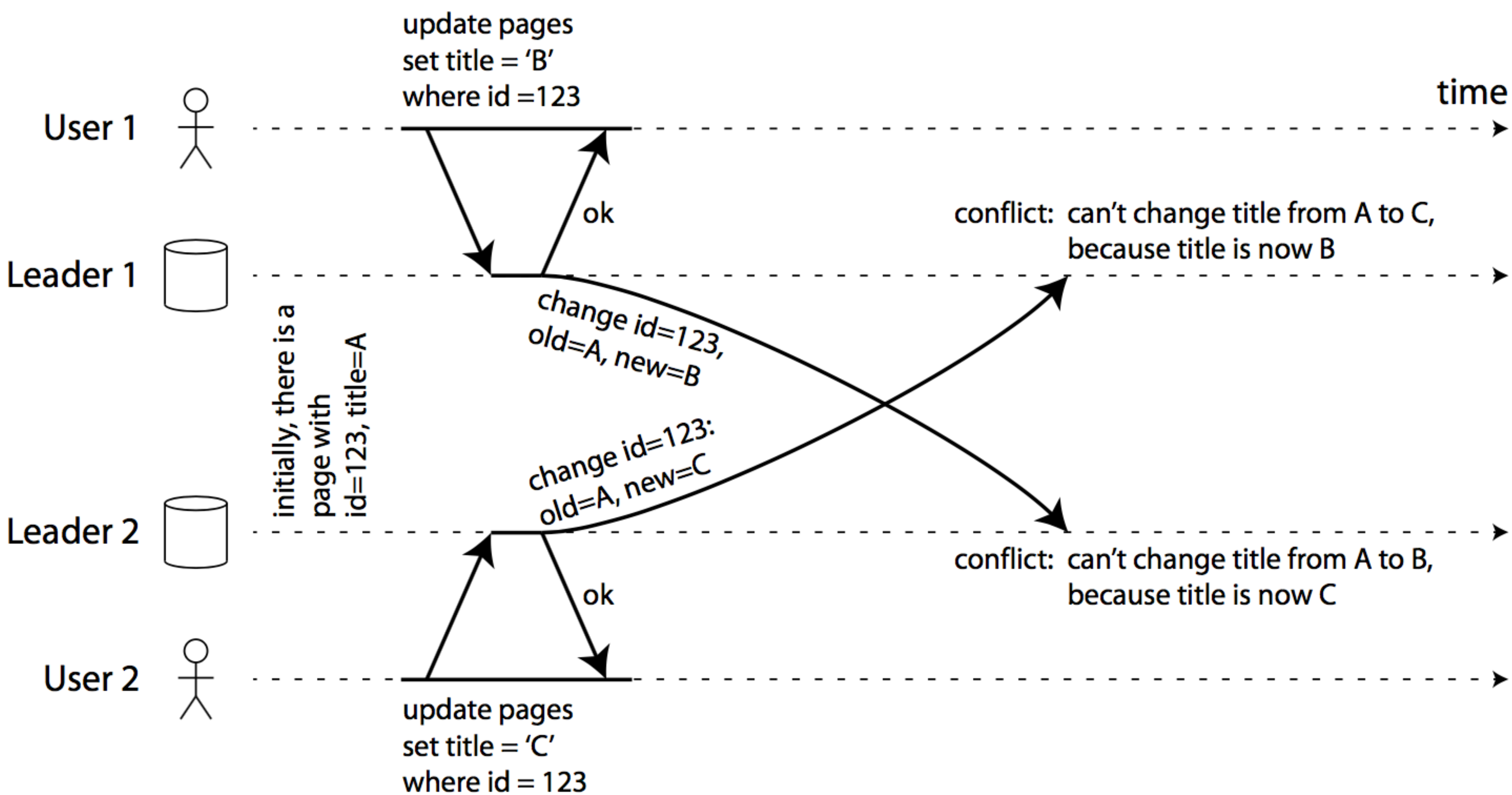
- <u>Read rule</u>: |R| + |W| > N ⇒ read and write quorums overlap
- <u>Write rule</u>: 2 |W| > N ⇒ two write quorums overlap
- The quorum sizes determine the costs for read and write operations
- Minimum quorum sizes for are

$$\min |W| = \left\lfloor \frac{N}{2} \right\rfloor + 1 \qquad \min |R| = \left\lceil \frac{N}{2} \right\rceil$$

  - Write quorums requires majority
  - Read quorum requires at least half of the nodes
- ROWA (R,W,N) = (N = N, R = 1, W = N)
- Amazon’s Dynamo (N = 3, R = 2, W = 2)
- Linkedin's Voldemort (N = 2 or 3, R = 1, W = 1 default)
- Apache's Cassandra (N = 3, R = 1, W = 1 default)

# Write Conflicts

The biggest problem with active replication is that write conflicts can occur, which means that conflict resolution is required.



# Handling Write Conflicts

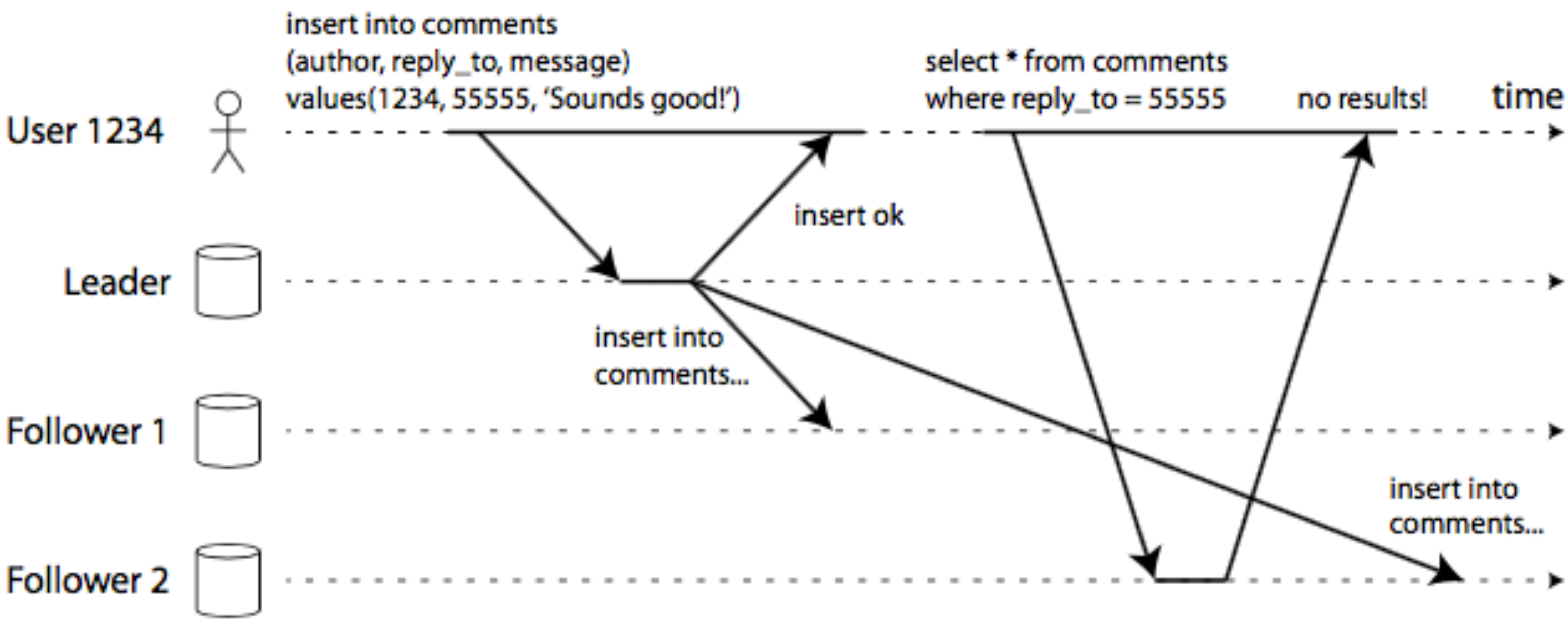
- Avoid them by '**normally**' using a **single leader**, and change leader for exceptional conditions only.
- **Converge** towards a consistent state
  - Give each write a unique ID, pick the write with the highest ID as the winner, and throw away the other writes. If a timestamp is used, this technique is known as **last write wins** (LWW). Although this technique is popular, it is dangerously prone to data loss.
  - Give each replica a unique ID, and let writes that originated at a **higher-numbered replica** always take precedence over writes that originated at a lower-numbered replica. This also implies data loss.
  - Record the conflict in an explicit data structure that preserves all information, and write application code which resolves the conflict at **some later time** (perhaps by prompting the user).
- Use custom logic
- Use automatic logic (e.g., conflict-free replicated data types, CRDTs)

# Replication Lag

- If an application reads from asynchronous followers, it may see outdated information if the follower has fallen behind.
- This leads to apparent inconsistencies in the database
  - if you run the same query on the leader and a follower at the same time, you may get different results, because not all writes have been reflected in the follower.
- This inconsistency is just a temporary state
  - if you stop writing to the database and wait a while, the followers will eventually catch up and become consistent with the leader.
- For that reason, this effect is known as eventual consistency.

# Read Your Writes Consistency

if the user views the data shortly after making a write, the new data may have not yet reached the replica.



# Client-centric Consistency Models

- **Each WRITE operation** is assigned a unique identifier
  - Done by the replica manager where the operation is requested
- For each replica manager, we keep track of:
  - Write set *WS* : contains the write operations executed so far
- For each client *c*, we keep track of:
  - Read set $WS_R$ : contains write operations relevant to the read operations performed by *c*
  - Write set $WS_W$ : contains write operations relevant to the write operations performed by *c*

# Read-Your-Writes Implementation

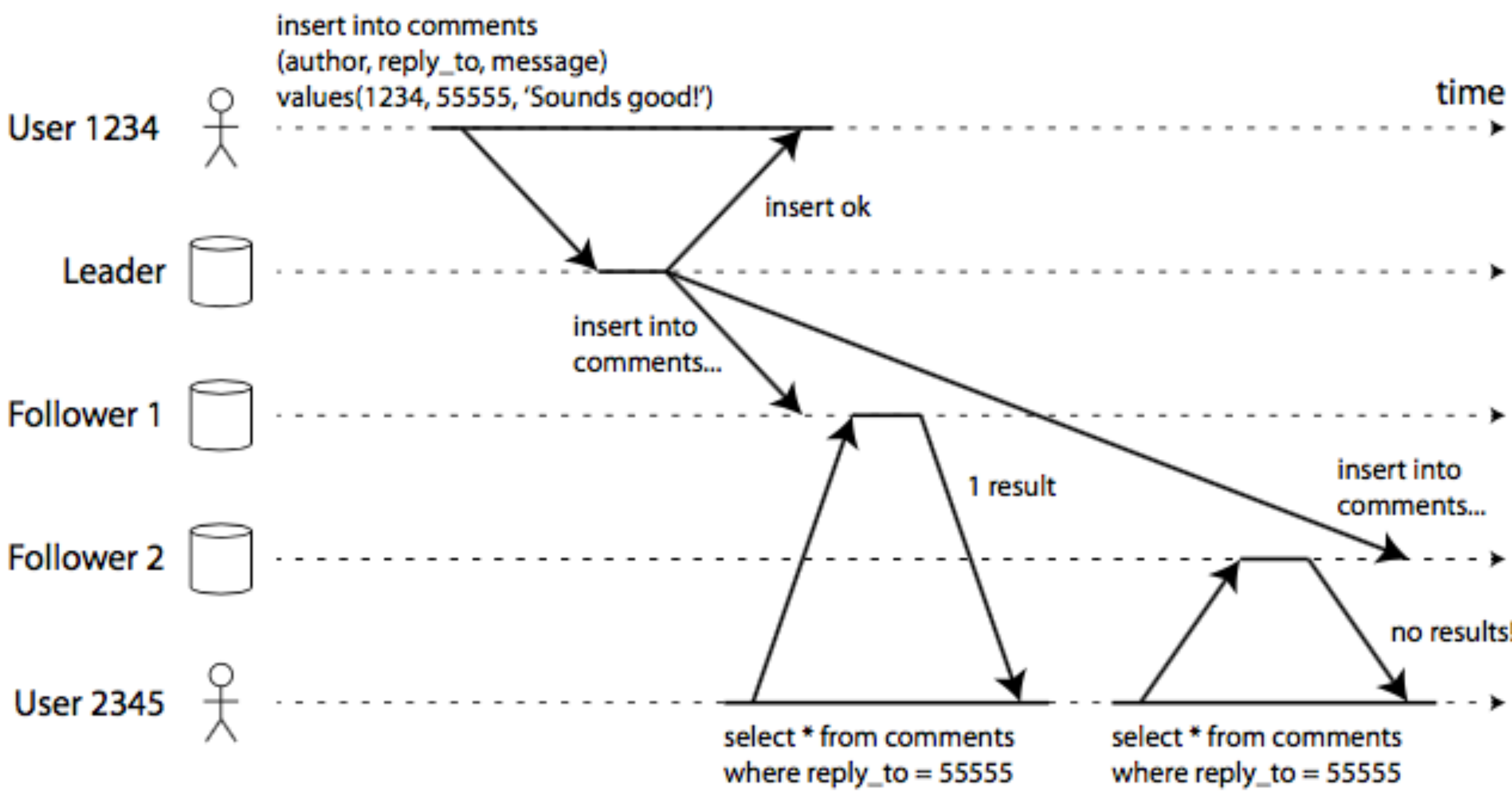
- To perform a READ:
  - A client
    - sends READ and its $WS_W$ to a replica manager S.
  - The replica manager S:
    - Checks if the $WS_W \subseteq WS$, i.e., all the WRITES seen from the client have been applied by the replica manager
    - If not, asks the other replica managers the missing WRITES
    - Applies the missing WRITES locally and update its WS
    - Return the requested value to the client

# Read-Your-Write Implementation

- To perform a WRITE:
  - A client
    - sends WRITE and adds it to its $WS_W$
  - The replica manager S:
    - Perform the WRITE
    - adds it to its WS

# Monotonic Reads Consistency

if a user makes several reads from different replicas, it's possible for a user to see things moving backwards in time.



# Monotonic-Read Implementation

- To perform a READ:
  - A client
    - sends READ and its $WS_R$ to a replica manager S.
  - The replica manager S:
    - Checks if the $WS_R \subseteq WS$, i.e., all the WRITES seen from the client have been applied by the replica manager
    - If not, asks the other replica managers the missing WRITES
    - Applies the missing WRITES locally and update its WS
    - Return the requested value and WS to the client
  - The client
    - adds WS to its $WS_R$

# Monotonic-Read Implementation

- To perform a WRITE:
  - A client
    - sends WRITE
  - The replica manager S:
    - Perform the WRITE
    - adds it to its WS

# Extra: Monotonic-Write

- In a monotonic-write consistent data storage system, a write operation by a client on a data item is completed before any successive write operation on the same object by the same client

# Extra: Writes Follow Reads

- In a writes-follows-reads consistent data storage system, a write operation by a client on a data item following a previous read operation on the same item by the same client is guaranteed to take place on the same or a more recent value of the item that was read

# Additional References


- D. Terry et al., *Session Guarantees for Weakly Consistent Replicated Data*, https://www.cis.upenn.edu/~bcpierce/courses/dd/papers/SessionGuaranteesPDIS.ps

- D. Terry, *Replicated Data Consistency Explained Through Baseball*, http://research.microsoft.com/pubs/157411/ConsistencyAndBaseballReport.pdf