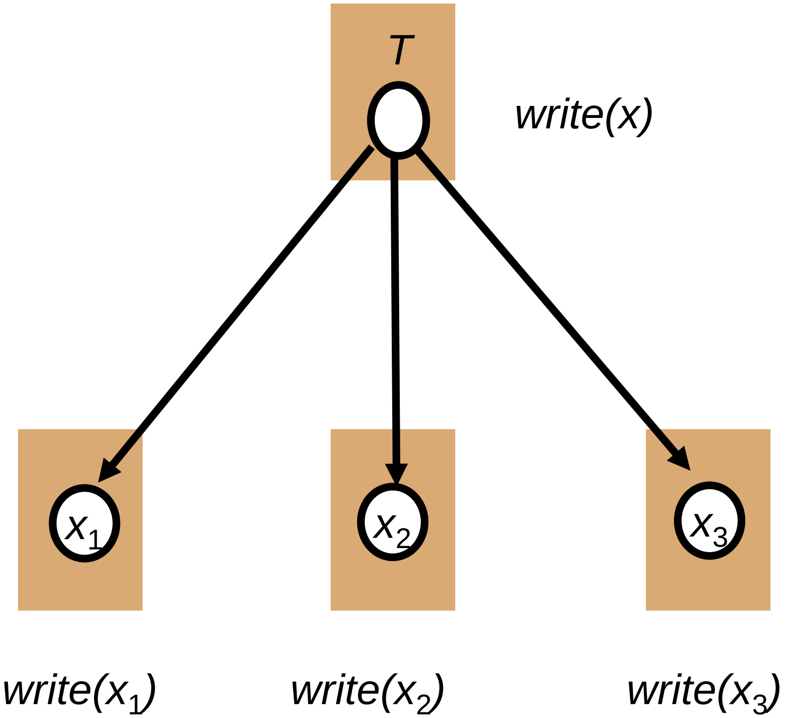
Module 7 - Replication

Replication

- Why replicate?
  - Reliability
    - → Avoid single points of failure
  - Performance
    - → Scalability in numbers and geographic area
- Why not replicate?
  - Replication transparency
  - Consistency issues
    - → Updates are costly
    - → Availability *may* suffer if not careful

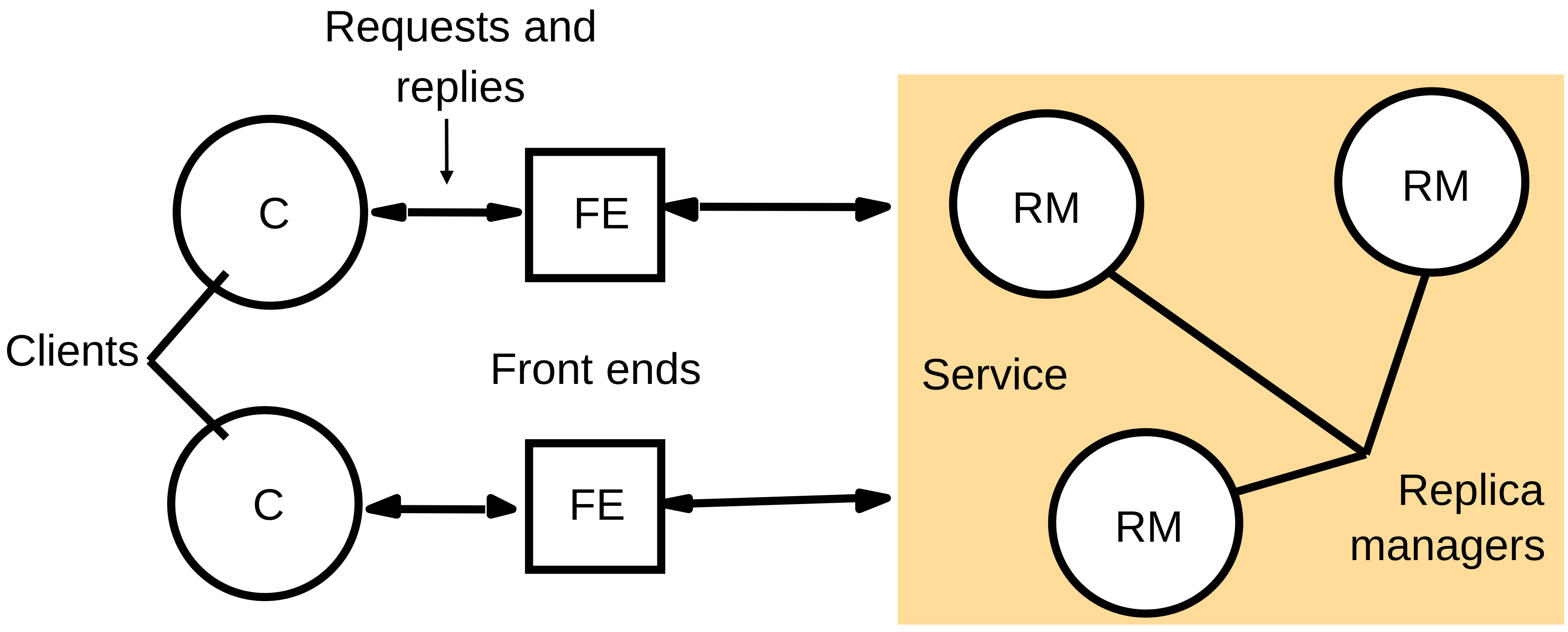
Logical vs Physical Objects

- There are physical copies of logical objects in the system.
- Operations are specified on logical objects, but translated to operate on physical objects.



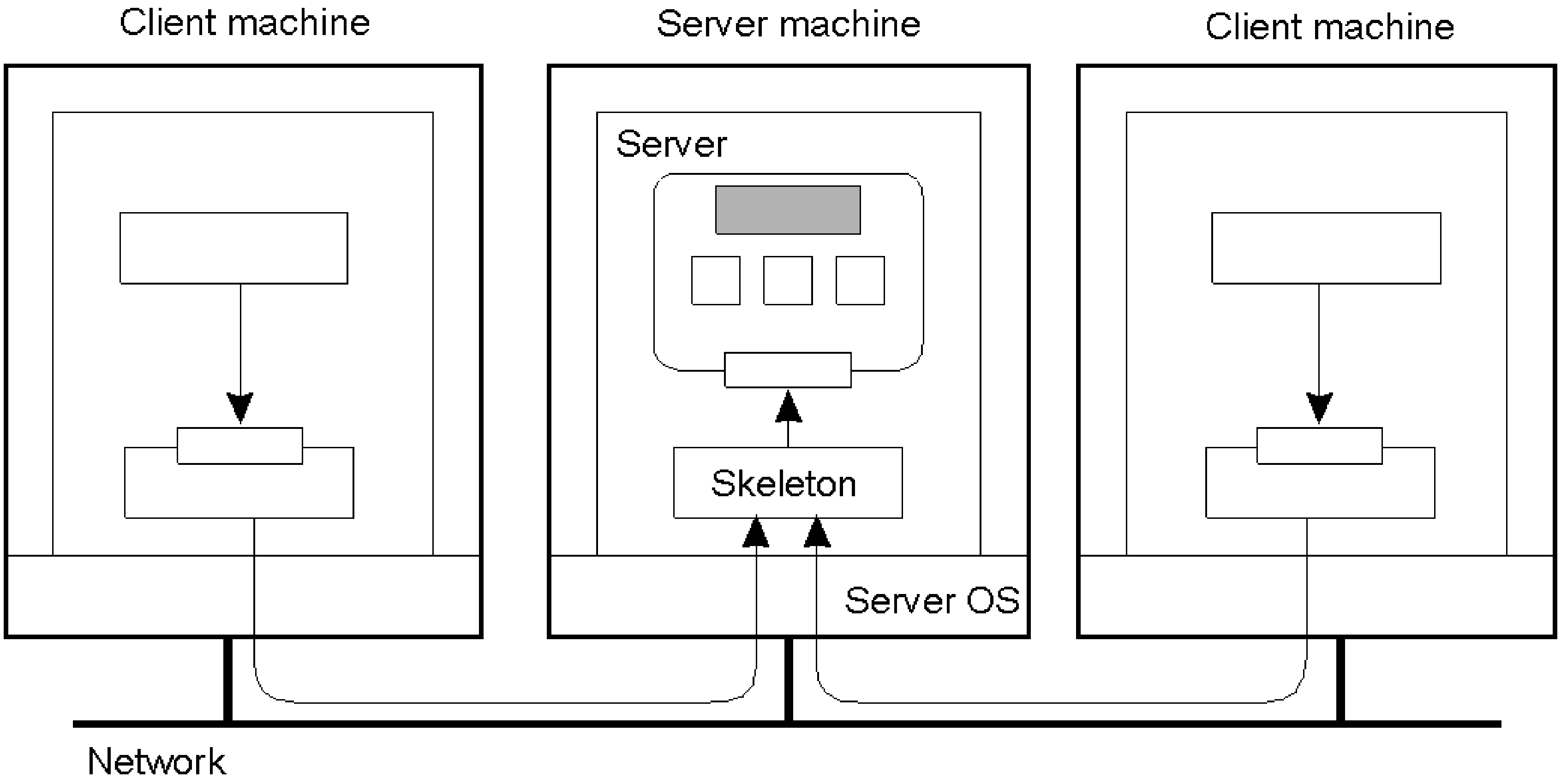
From Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 3rd ed.

Replication Architecture

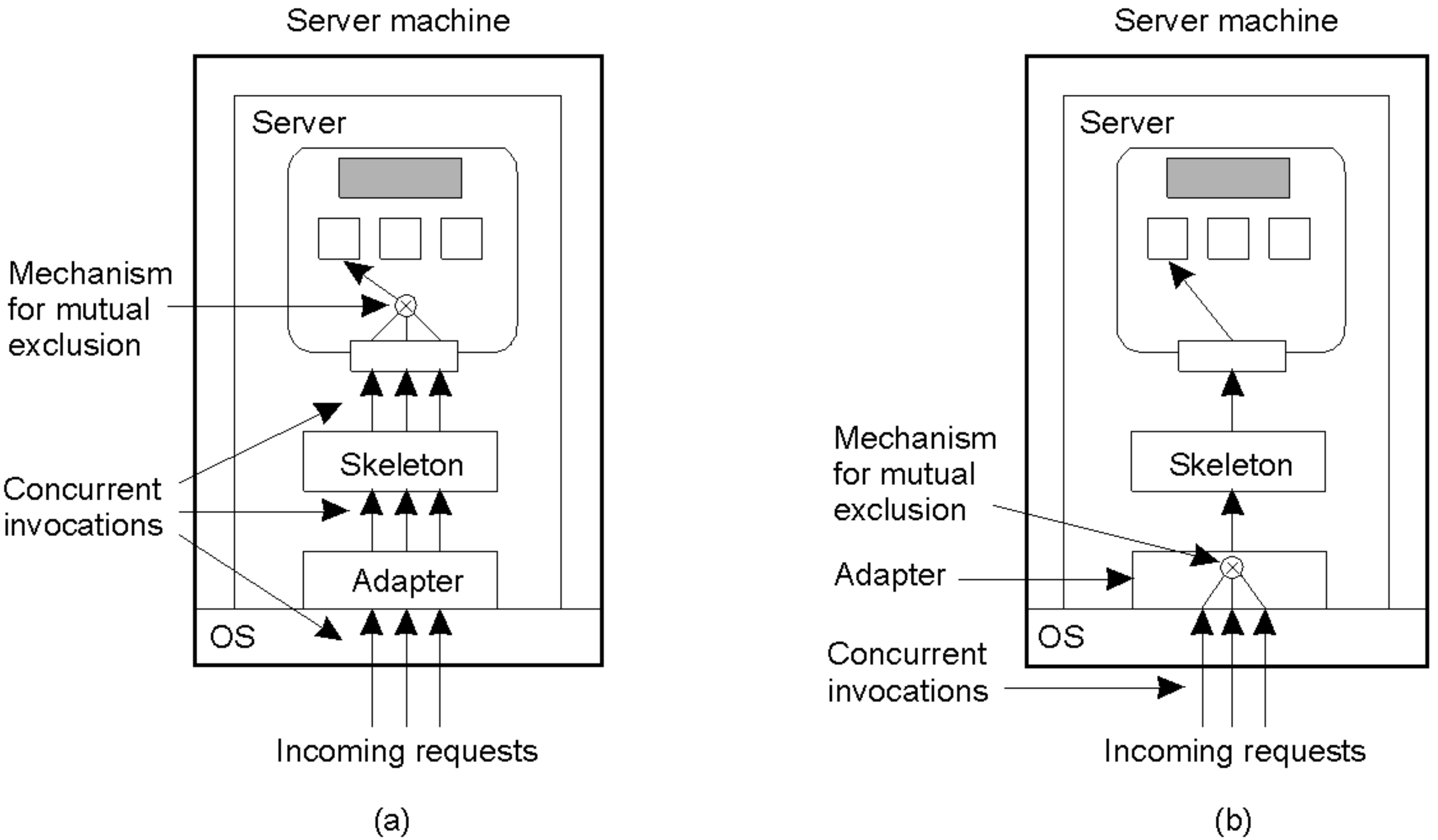


From Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 3rd ed.

Object Replication (1)



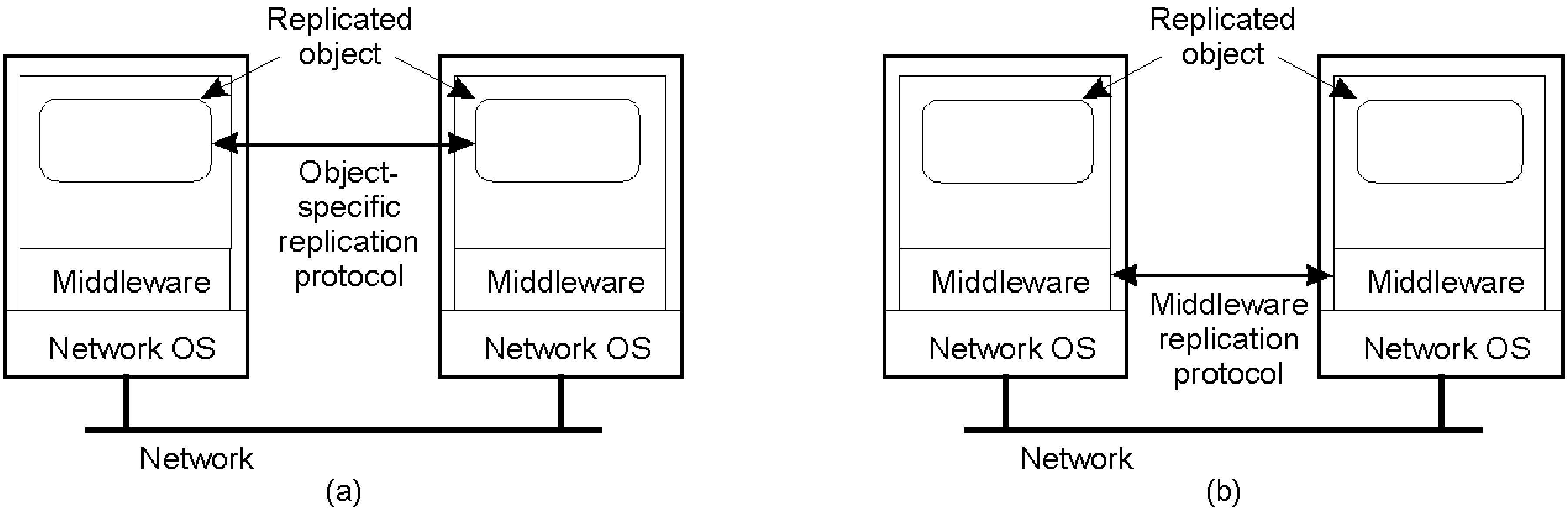
Object Replication (2)



a) A remote object capable of handling concurrent invocations on its own.
b) A remote object for which an object adapter is required to handle concurrent invocations

From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

Object Replication (3)



a) A distributed system for replication-aware distributed objects.

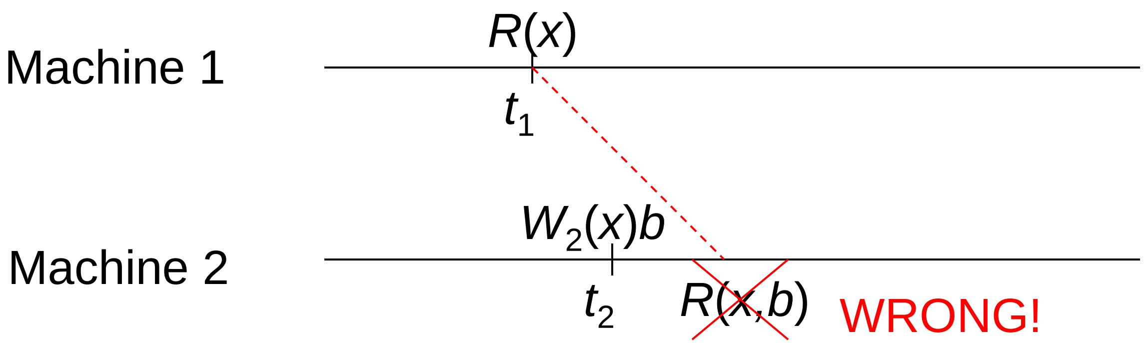
b) A distributed system responsible for replica management

What will we study

- Consistency models - How do we reason about the consistency of the "global state"?
  - Data-centric consistency
    - → Strict consistency
    - → Linearizability
    - → Sequential consistency
  - Client-centric consistency
    - → Eventual consistency
- Update propagation - How does an update to one copy of an item get propagated to other copies?
- Replication protocols - What is the algorithm that takes one update propagation method and enforces a given consistency model?

Strict Consistency

- Any *read*(*x*) returns a value corresponding to the result of the most recent *write*(*x*).



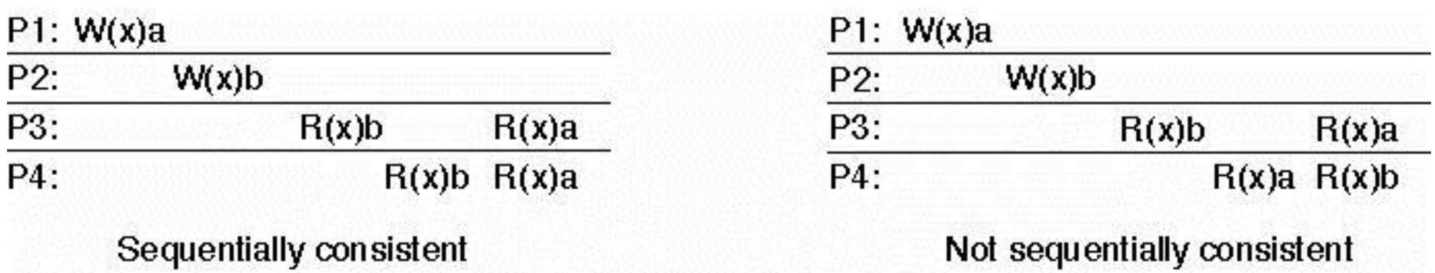
- Relies on absolute global time; all writes are instantaneously visible to all processes and an absolute global time order is maintained.
- Cannot be implemented in a distributed system

Linearizability

- The result of the execution should satisfy the following criteria:
  - Read and write by all processes were executed in some serial order and each process's operations maintain the order of specified;
  - If $ts_{op_1}(x) < ts_{op_2}(y)$ then $op_1(x)$ should precede $op_2(y)$ in this sequence. This specifies that the order of operations in itnerleaving is consitent with the real times at which the operations occurred in the actual implementation.
- Requires synchronization according to timestamps, which makes it expensive.
- Used only in formal verification of programs.

Sequential Consistency

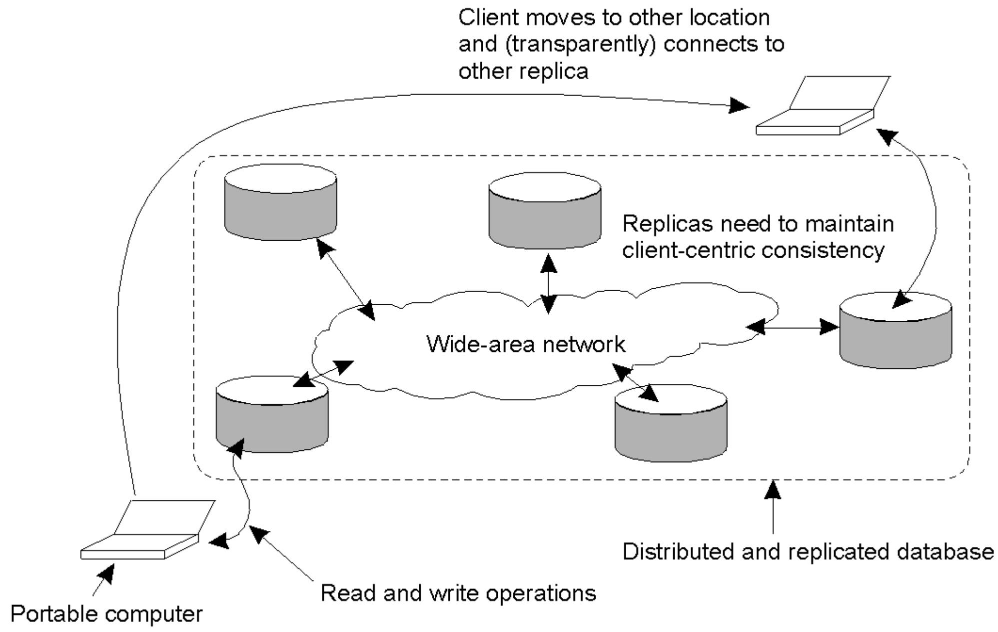
- Similar to linearizability, but no requirement on timestamp order.
- The result of execution should satisfy the following criteria:
  - Read and write operations by all processes on the data store were executed in some sequential order;
  - Operations of each individual process appear in this sequence in the order specified by its program.
- These mean that all processes see the same interleaving of operations ⇧ similar to serializability.



From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

Client-Centric Consistency

- More relaxed form of consistency ➔ only concerned with replicas being eventually consistent (eventual consistency).
- In the absence of any further updates, all replicas converge to identical copies of each other ➔ only requires guarantees that updates will be propagated.
- Easy if a user always accesses the same replica; problematic if the user accesses different replicas.
  - Client-centric consistency: guarantees for a single client the consistency of access to a data store.

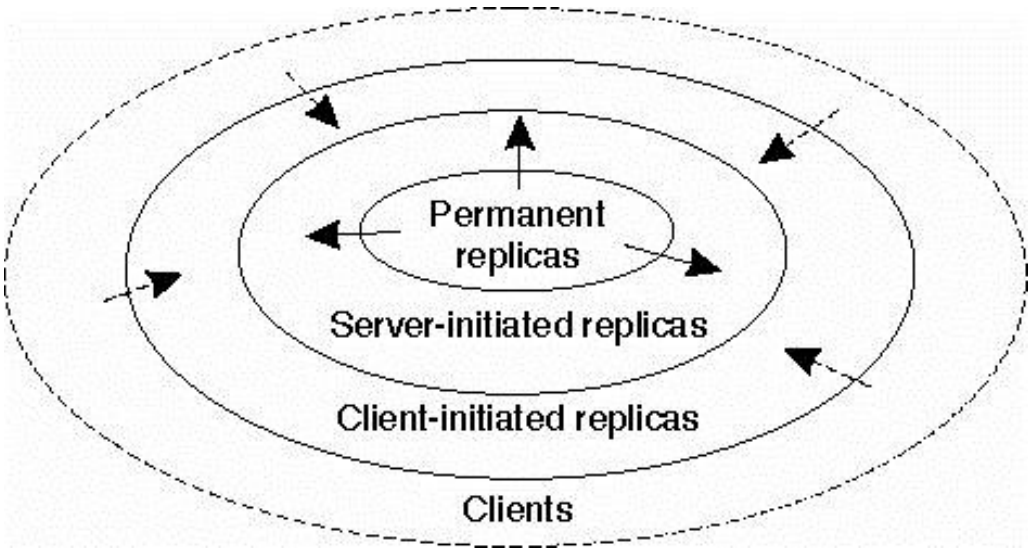


From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

Client-Centric Consistency (2)

- Monotonic reads
  - If a process reads the value of a data item *x*, any successive read operation on *x* by that process will always return that same value or a more recent value.
- Monotonic writes
  - A write operation by a process on a data item *x* is completed before any successive write operation on *x* by the same process.
- Read your writes
  - The effect of a write operation by a process on data item *x* will always be seen by a successive read operation on *x* by the same process.
- Writes follow reads
  - A write operation by a process on a data item *x* following a previous read operation on *x* by the same process is guaranteed to take place on the same or more recent value of *x* that was read.

Replica Placement Alternatives



- Permanent replicas
  - Put a number of replicas at specific locations
  - Mirroring
- Server-initiated replicas
  - Server decides where and when to place replicas
  - Push caches
- Client-initiated replicas
  - Client caches

Update Propagation

- What to propagate?
  - Propagate only a notification
    - → Invalidation
  - Propagate updated data
    - → Possibly only logs
  - Propagate the update operation
    - → Active replication
- Who propagates?
  - Server: push approach
  - Client: pull approach
- Epidemic protocols
  - Update propagation in eventual-consistency data stores.

Pull versus Push Protocols

Issue	Push-based	Pull-based
State at server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

Epidemic Protocols

- Based on the spreading of infectious diseases
  - Epidemic protocols try to “infect” all nodes with new information as fast as possible.
  - A node is “infected” if it holds data that it is willing to spread to other nodes.
  - Nodes that haven’t seen this data are “susceptible”
  - A node that is unwilling or unable to spread that data is called “removed”

Epidemic Protocols

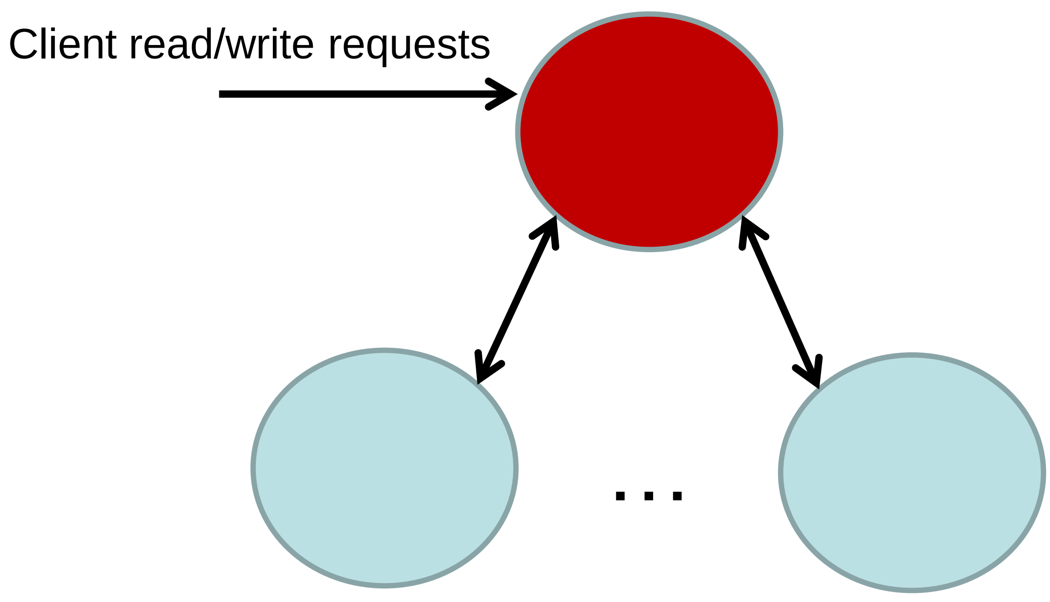
- Anti-entropy
  - Node P picks another node Q at random and exchanges updates with Q.
  - P pushes and/or pulls updates to/from Q.
  - Number of rounds to propagate a single update to all nodes is O(log N)
    - → A round is the period where every node would have at least once initiated an exchange with another node.
- Gossip
  - Node P has just been updated with data item x
  - It tries to push the update to a random node Q
    - → If Q has already heard the rumor, then P may lose interest in spreading the rumor (assume with probability 1/k).
  - Probabilistic protocol:
    - → $s = e^{-(k+1)(1-s)}$ will remain susceptible but will never receive the update.
    - → For k =4, then s is less than 0.007
- How do you remove data?

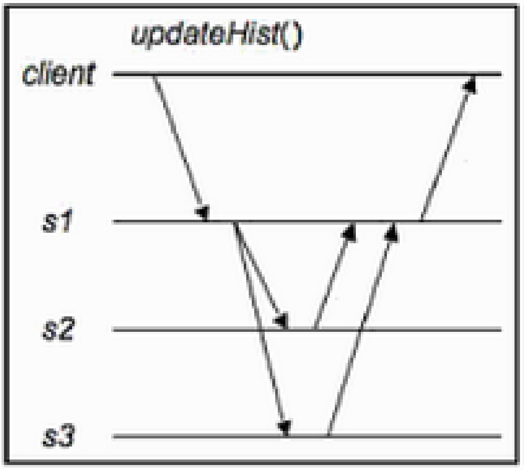
Replication Protocols

- Linearizability
  - Primary-Backup
  - Chain Replication
- Sequential consistency
  - Primary-based protocols
    - → Remote-Write protocols
    - → Local-Write protocols
  - Replicated Write protocols
    - → Active replication
    - → Quorum-based protocols

Primary-Backup

- To ensure linearizability, all reads and writes are sent to the primary node
  - Operations ordered by the primary.
  - Primary node forwards all writes to the backup nodes





Primary-Backup

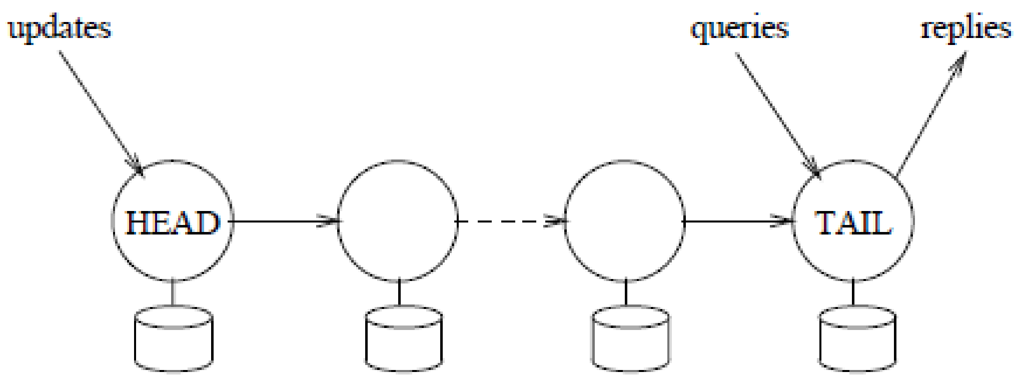
- Should the primary wait until receiving responses from the backups before responding to the client?
- What actions are required when the primary fails?

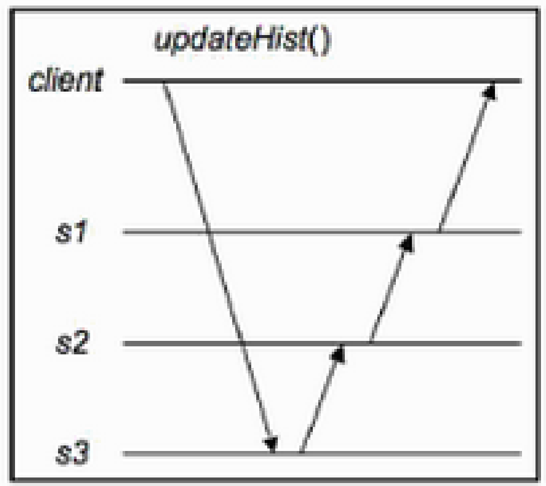
Primary-Backup

- Should the primary wait until receiving a responses from the backups before responding to the client?
  - Decision is a tradeoff between write latency and fault tolerance.
  - If the primary does not wait, it may report write success to the client and then fail before propagating the writes, resulting in data loss.

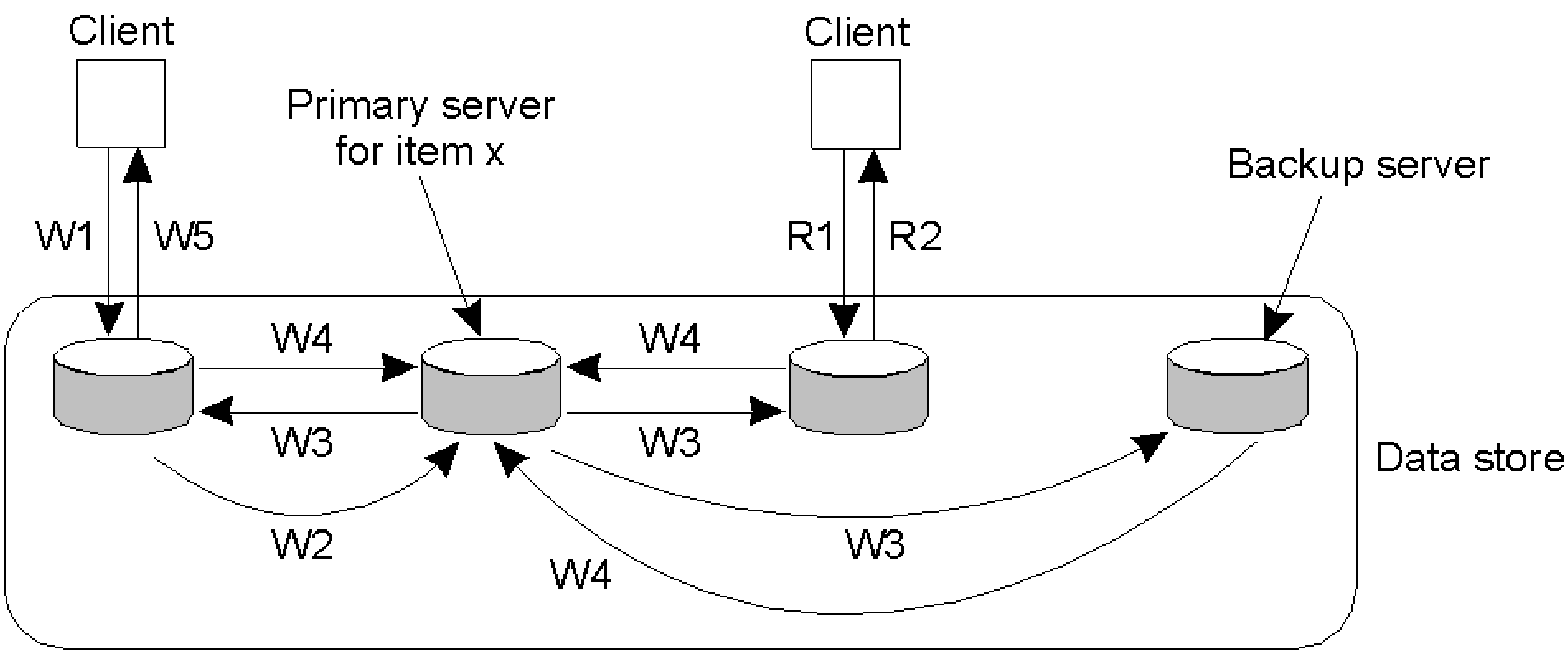
- What actions are required when the primary fails?
  - Elect a new primary
  - New primary sends a “sync” message to all backups to ensure the backups share the same history as the primary.
  - Assume that clients resend failed requests using the same request ID.

Chain Replication





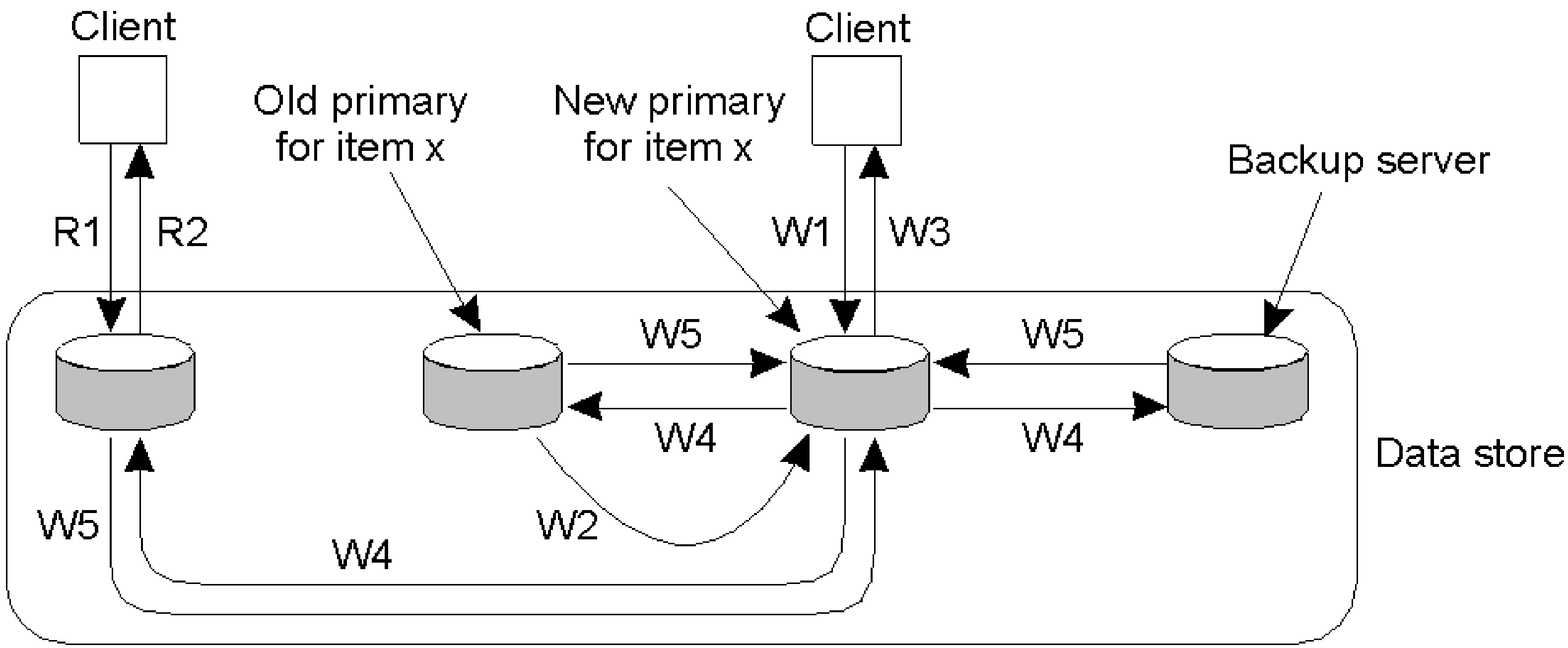
Primary Copy Remote-Write Protocol



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

Primary Copy Local-Write Protocol



W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

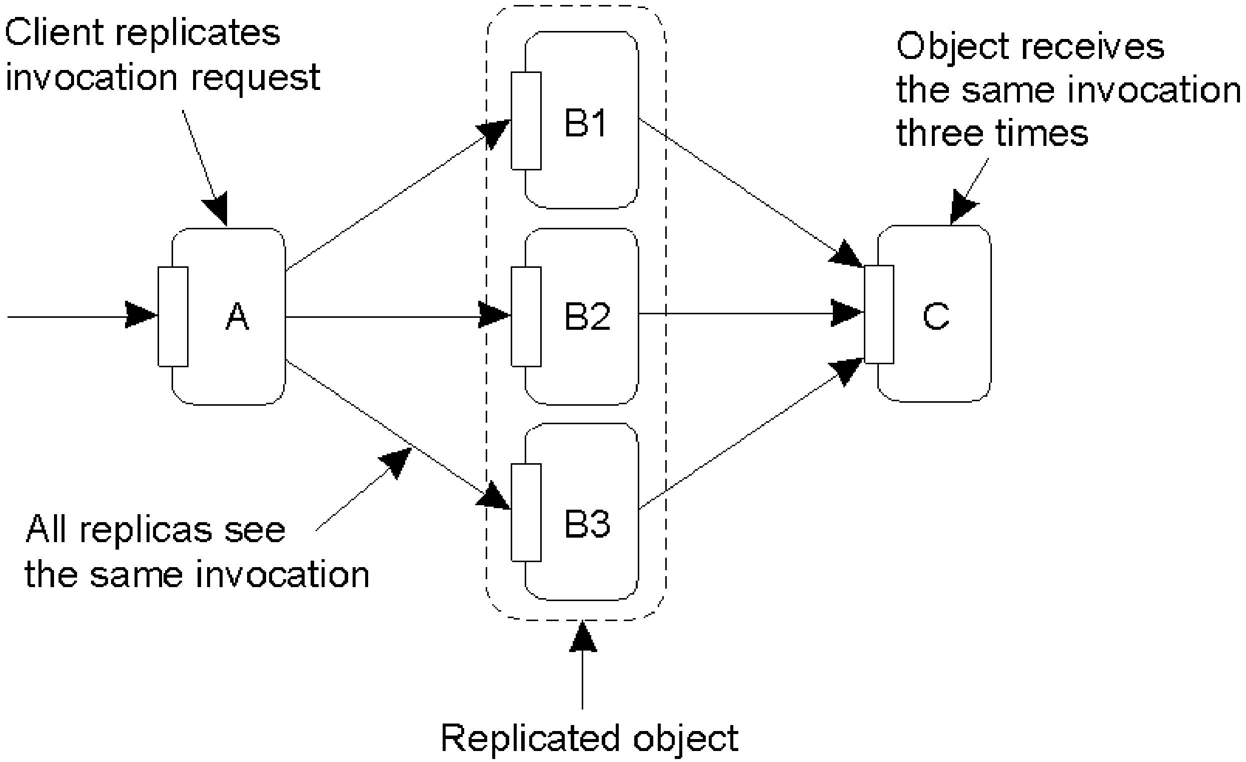
R1. Read request
R2. Response to read

From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

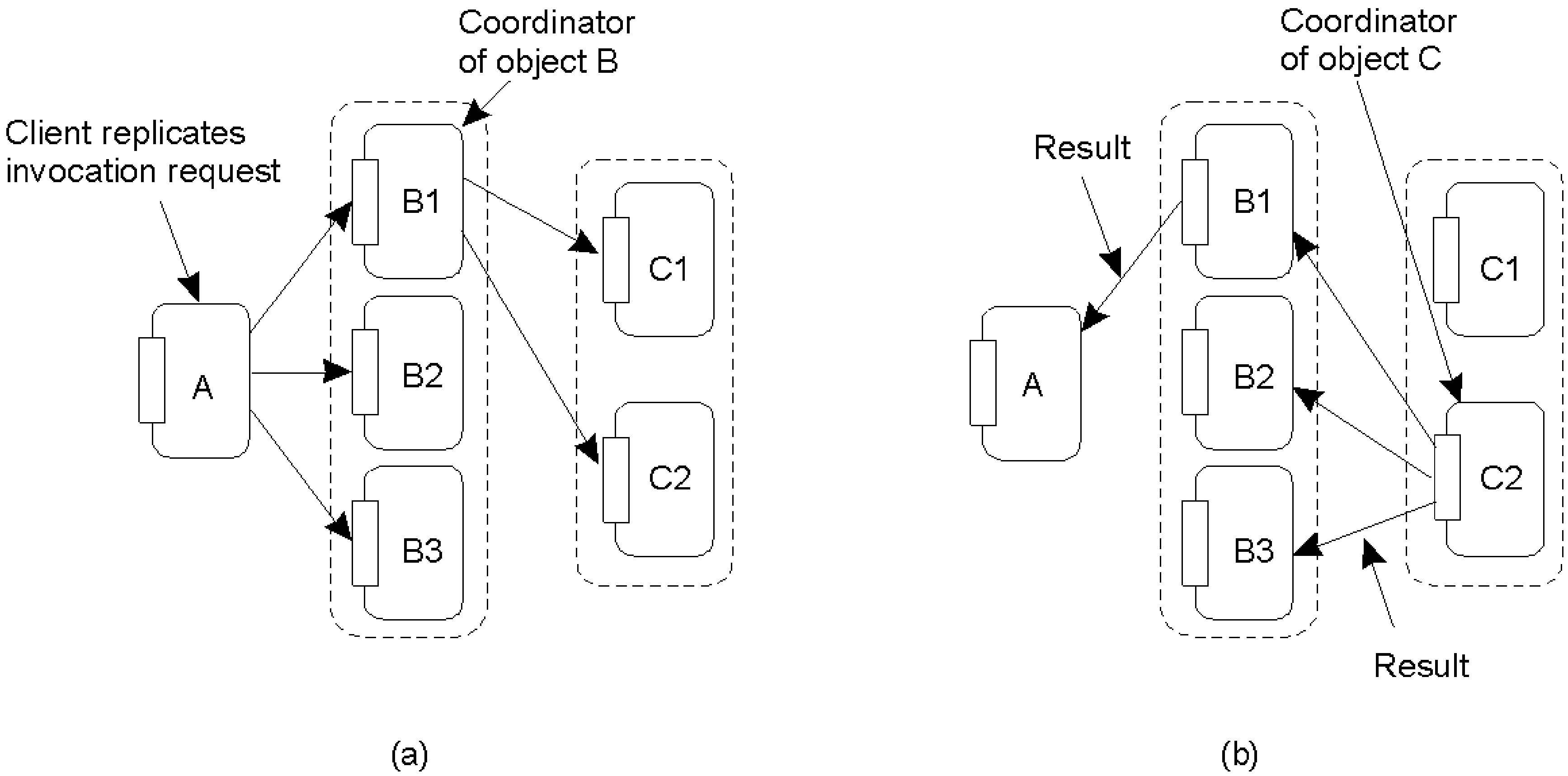
Active Replication

- Requires a process, for each replica, that can perform the update on it
- How to enforce the update order?
  - Totally-ordered multicast mechanism needed
  - Can be implemented by Lamport timestamps
  - Can be implemented by sequencer
- Problem of replicated invocations
  - If an object *A* invokes another object *B*, all replicas of *A* will invoke *B* (multiple invocations)

Replicated Invocations Problem



Solution to Replicated Invocations



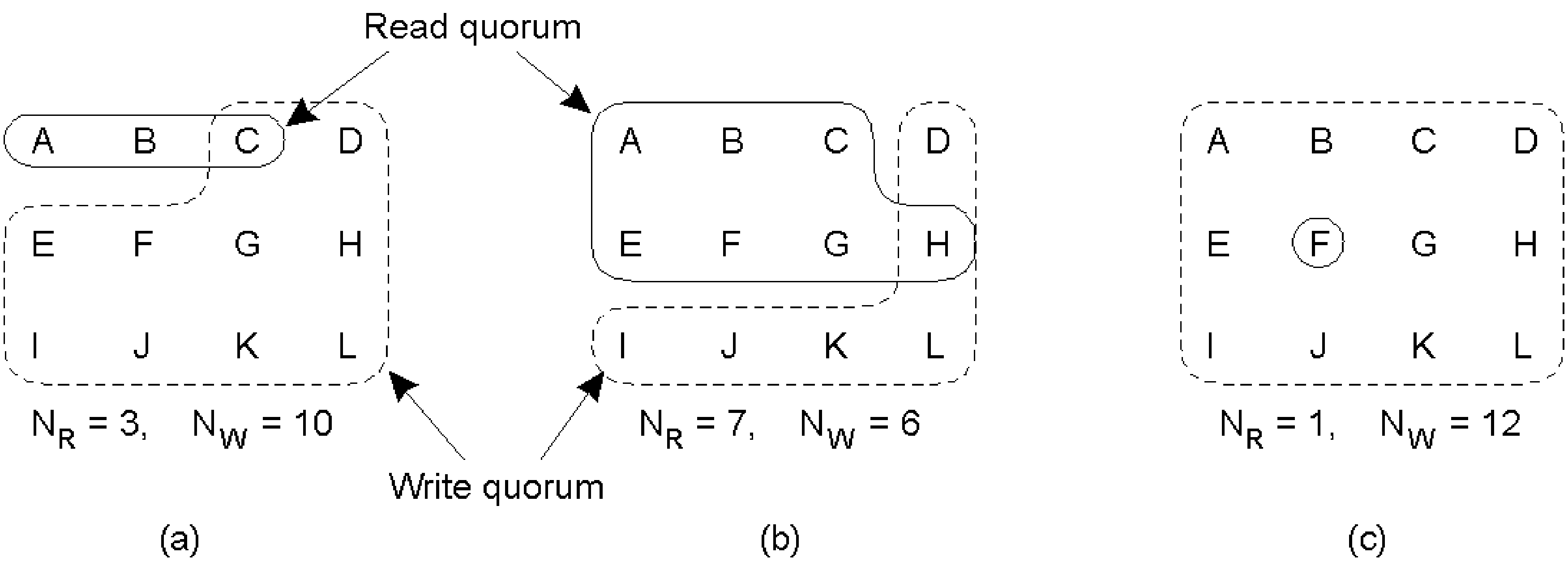
a) Forwarding an invocation request from a replicated object.

b) Returning a reply to a replicated object.

Quorum-Based Protocol

- Assign a vote to each *copy* of a replicated object (say V_i) such that $\sum_i V_i = V$
- Each operation has to obtain a *read quorum* (V_r) to read and a *write quorum* (V_w) to write an object
- Then the following rules have to be obeyed in determining the quorums:
  - $V_r + V_w > V$ — an object is not read and written by two transactions concurrently
  - $V_w > V/2$ — two write operations from two transactions cannot occur concurrently on the same object

Quorum Example



Three examples of the voting algorithm:

a) A correct choice of read and write set

b) A choice that may lead to write-write conflicts

c) ROWA