Distributed Systems Lecture 9

Professor Louise E. Moser

Spring 2013

Client-Centric Consistency Models

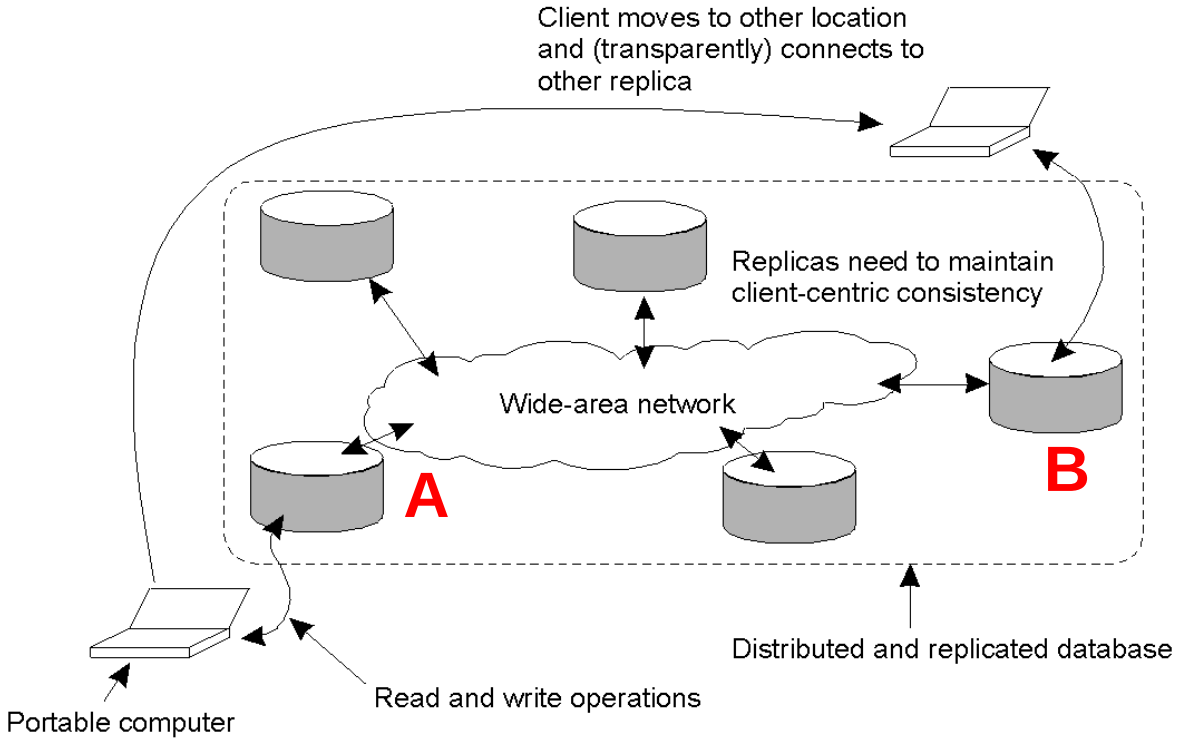
- **Background:** Large-scale distributed systems often use replication for scalability, but support only weak consistency
  - **DNS: Updates are propagated slowly, and inserts might not be immediately visible**
  - **NEWS: Articles and reactions are pushed and pulled through the Internet, and might be seen on some Web sites before postings on other Web sites**
  - **WWW: Caches exist all over the Internet, but there is no guarantee that you're reading the most recent version of a Web page**
- **Question:** How (perhaps) can we avoid system-wide consistency, by focusing on what individual *clients* want, instead of what servers should maintain?

Consistency for Mobile Users

- **Example:** Consider a distributed database to which you have access from your laptop. Assume that your laptop acts as a front-end to the database
- At location *A*, you access the database at A doing reads and updates
- At location *B*, you continue your work with the database at B; however, you might detect inconsistencies, because:
  - **Your updates at *A* might not yet have been propagated to *B***
  - **You might be reading newer entries at B than the entries at *A***
  - **Your updates at *B* might eventually conflict with those at *A***

Consistency for Mobile Users

- All you really want is that the entries you read and/or updated in the database at *A* are in the database at *B* the way you left them in the database at *A*
  - **In that case, the database will appear to be consistent to *you***



Client-Centric Consistency Models

- **Monotonic Reads**
- **Monotonic Writes**
- **Read Your Writes**
- **Writes Follow Reads**

Monotonic – One after the other in succession

Monotonic Reads

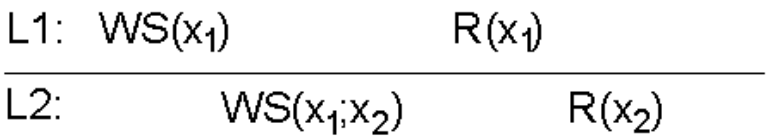
- If a process reads the value of a data item x, any successive read operation on x **(on the same or a different data store)** by the same process returns the same or a more recent value of x
- **Examples:**
  - Reading (not modifying) incoming e-mail while you are on the go. Each time you connect to a different e-mail server, that e-mail server fetches (at least) all updates from the e-mail server you previously used
  - Reading updates to your personal calendar from different computers. No matter which computer is used for reading, you see the same updates
- **Example Violation:** You receive and read an email message with a timestamp 3:00pm before you receive and read an email message with a timestamp 2:55pm

Monotonic Reads

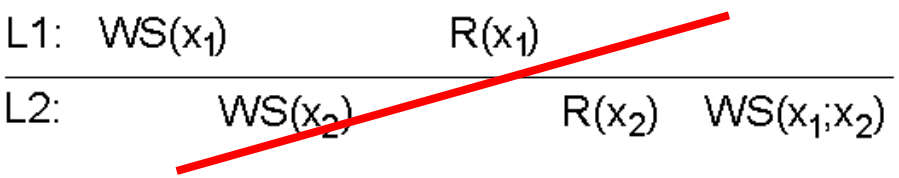
- The read operations performed by a process on two different local copies (L1 and L2) of the same data store

(a) A valid example of monotonic read consistency:
In L1, only x_1 is read; in L2, the read of x_2 occurs after the propagation of x_1 from L1 to L2 and after the write of x_2

(b) An invalid example of monotonic read consistency:
In L1, only x_1 is read; in L2, the read of x_2 occurs before the propagation of x_1 to L2



(a)



(b)

Note: $WS(x_i)$ is the sequence of write operations in a local copy L that lead to version x_i of x. $WS(x_i{:}x_j)$ indicates that $WS(x_i)$ precedes $WS(x_j)$ in a local copy L. Think of x_1 as an earlier value of x, and x_2 as a later value of x that depends on x_1.

Monotonic Writes

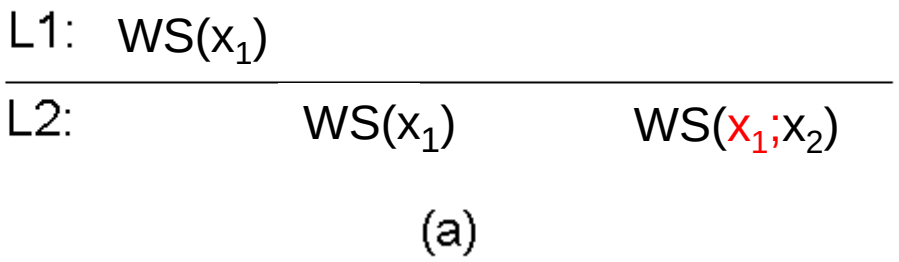
- If a process writes to a data item x, that operation is completed before any successive write operation to x by the same process
- **Examples:**
  - Maintaining different versions of replicated files in the same order on several computers (propagate the previous versions to another computer before sending the next version)
  - Modifying different modules of a program on one or more laptops, and compiling and linking them on a server, making sure that all of the updated modules are placed on the server before compiling and linking
- **Example Violation:** You send a message "Let's get together on Thursday" and then send a second message "Oops, I meant Friday" but the second message is received before the first message

Monotonic Writes

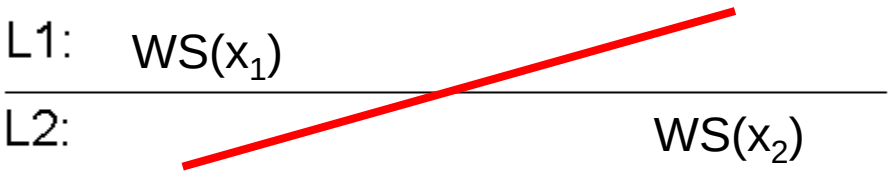
- The write operations performed by a process on two different local copies of the same data store

(a) A valid example of monotonic write consistency:
In L1, x_1 is written but x_2 is not
In L2, x_1 is propagated from L1 to L2 and then x_2 is written

(b) An invalid example of monotonic write consistency:
In L1, x_1 is written but x_2 is not
In L2, x_2 is written without first propagating x_1 from L1 to L2



(a)



(b)

Read Your Writes

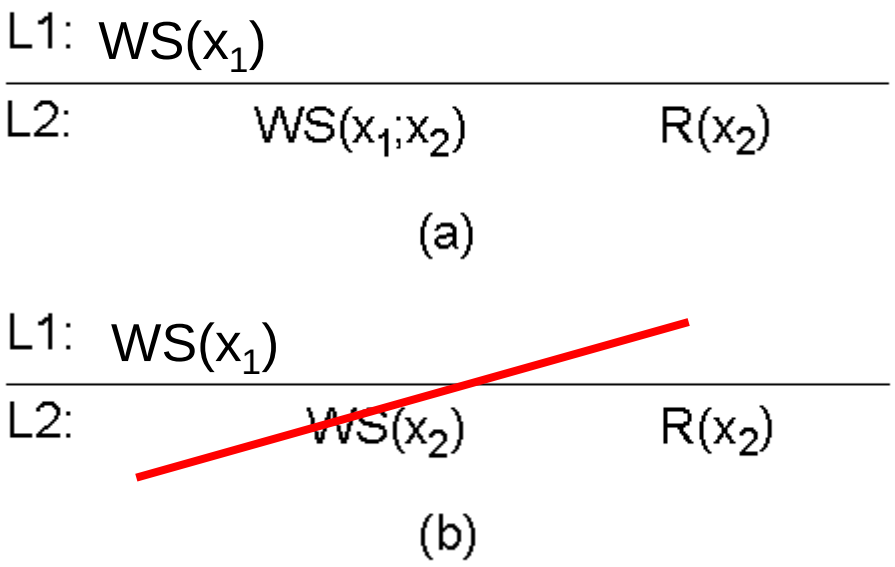
- The effect of a write operation by a process on a data item x is seen by a successive read operation on x by the same process
- **Example:** Updating your Web page and guaranteeing that your Web browser shows the newest version of the Web page, instead of an old cached copy
- **Example Violation:** After you send an email message, you don't see the message in your Inbox or Sent mail for a few seconds

Read Your Writes

- The operations performed by a process on two different local copies of the same data store

(a) A valid example of read-your-writes consistency:
In L1, the write of x_1 is propagated to L2;
In L2, the read of x_2 occurs after x_1 and x_2 are written

(b) An invalid example of read-your-writes consistency:
The read of x_2 in L2 does not see the write of x_1 in L1



Writes Follow Reads

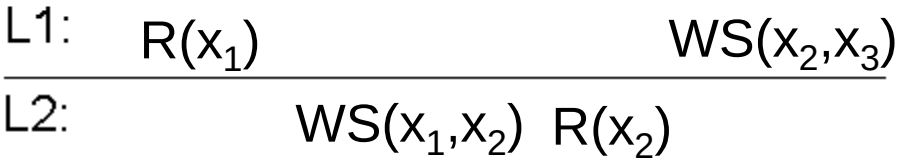
- A write operation on a data item x, that follows a previous read operation on x by the same process, depends on the same or a more recent value of x
- **Example:** Reactions to posted articles on the Web follow the original posting (a read operation "pulls in" the corresponding write operation)
- **Example Violation:** You get two email messages from Tom, read both, and then reply to the first message as if you had not read the second message

Writes Follow Reads

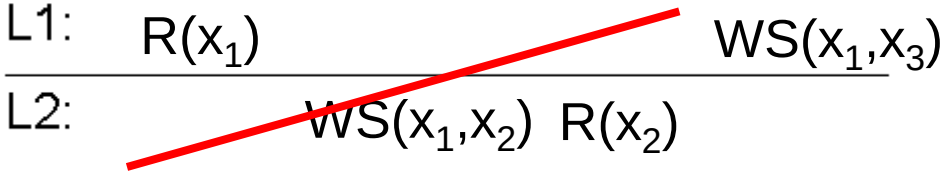
- The operations performed by a process on two different local copies of the same data store

(a) A valid example of writes-follow-reads consistency:
The write of x_3 in L1 follows the read of x_2 in L2, which is propagated to L1 before x_3 is written

(b) An invalid example of writes-follow-reads consistency:
The write of x_3 in L1 does not follow the read of x_2 in L2, because x_2 is not propagated from L2 to L1



(a)



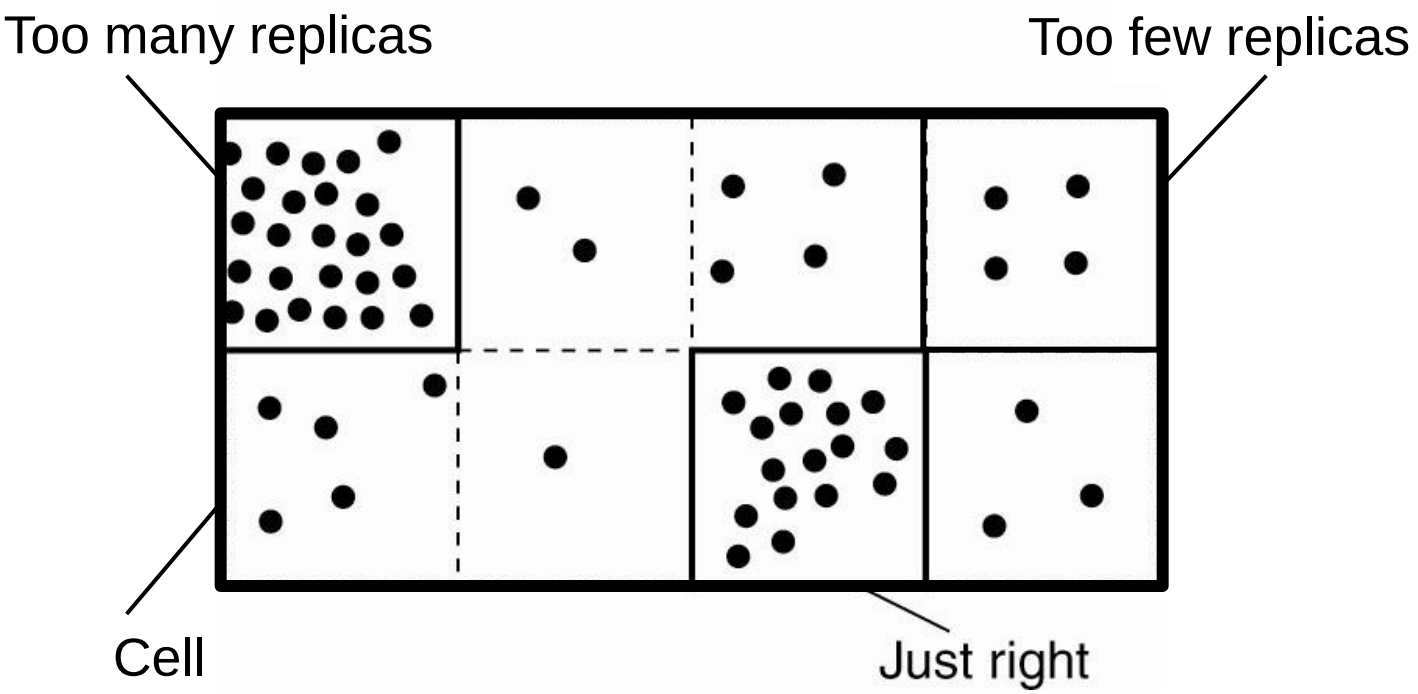
(b)

Replica Management

- **Replica-Server Placement**
- **Content Replication and Placement**
- **Content Distribution**

Replica-Server Placement

- Choosing an appropriate number of replicas and placing them appropriately

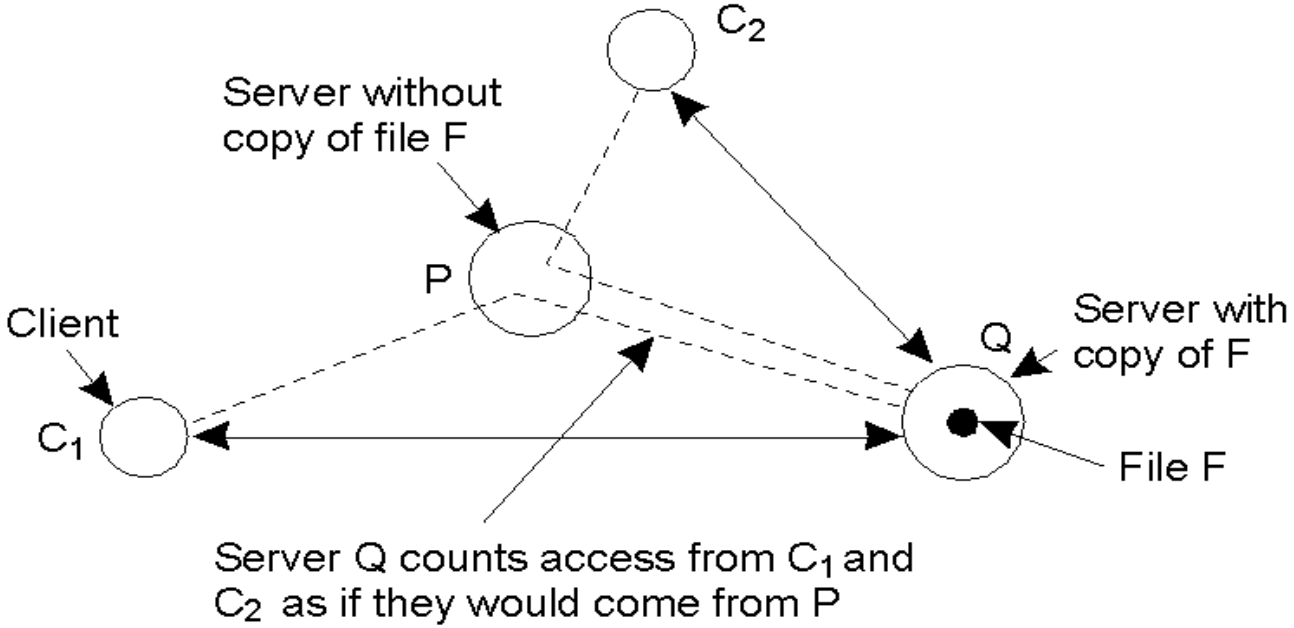


Content Replication and Placement

- **Different kinds of replicas, i.e., copies of objects, files, databases, processes**
  - **Permanent replicas:** Placed on some number of servers at a single location, or on some number of servers spread across the Internet (*mirroring*), typically for **fault tolerance**
  - **Server-initiated replicas:** Created at the initiative of the owner of the objects, to reduce the **load** on the server, or to reduce the **bandwidth** required by placing the replicas in the proximity of the clients
  - **Client-initiated replicas:** Created at the initiative of a client, to reduce the **access time** by placing the objects on the same machine as the client (e.g., in a client cache) or, if shared with other clients, on a machine within the same local-area network

Server-Initiated Replicas

- For each file F, keep track of **access counts** and aggregate them, considering the server closest to the requesting clients
  - **Number of accesses drops below a threshold *D* : Drop the file**
  - **Number of accesses exceeds a threshold *R* : Replicate the file**
  - **Number of accesses is between *D* and *R* : Migrate the file**



Content Distribution

- **State vs. Operations**
  - **Propagate only the notification / invalidation of an update (often used for caches)**
  - **Transfer data from one copy to another (used for distributed databases)**
  - **Propagate the update operation to other copies (also called *active replication*)**
- **Observation:** No one approach is the best
  - **The approach to use depends on the available *bandwidth* and the *read-to-write ratio* at the replicas**

Content Distribution

- **Push-Based vs. Pull-Based Updates**
  - **Push-Based:** Server-initiated approach in which the server propagates the update, regardless of whether or not the target asked for it
  - **Pull-Based:** Client-initiated approach in which the client requests that the update be sent to it

Issue	Push-Based	Pull-Based
State of 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

Content Distribution

- **Observation:** We can dynamically switch between pulling and pushing using a **lease**, i.e., a contract in which the server promises to *push* updates to the client until the lease expires
- **Adaptive lease:** Make the lease expiration time dependent on the system's behavior
  - **Age-based lease: An object that hasn't changed for a long time, will not change in the near future, so provide a lease with a long expiration time**
  - **Renewal-frequency-based lease: The more often a client requests a specific object, the longer the expiration time for that object is for that client**
  - **State-based lease: The more loaded a server is, the shorter the expiration time becomes**

Consistency Protocols

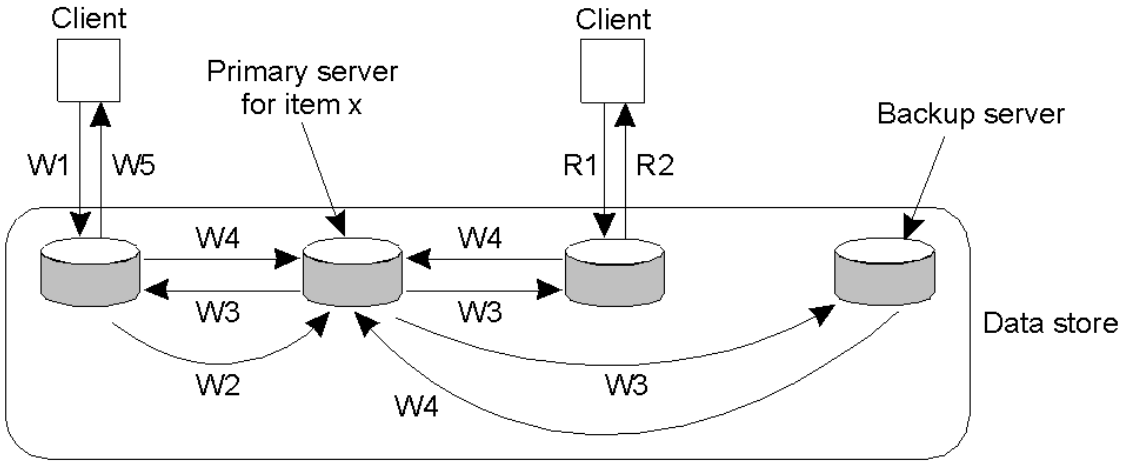
- **Consistency protocol:** Implements a specific consistency model
- We will focus on sequential consistency
  - **Primary-based protocols**
    - Remote write
    - Local write
  - **Replicated-write protocols**
    - Active replication
    - Quorum-based
  - **Cache-coherence protocols**

Primary-Based Remote-Write Protocols

- All **write** operations on a data item are forwarded to a single fixed server, called the **primary**
  - **The primary is responsible for performing all the write operations on the data item**
- The other servers are called the **backups**; read operations can be performed locally at the backups
- Typically, the primary and the backups are in the same local-area network
- Used in distributed database and file systems that require a high degree of fault tolerance
- Primary-backup protocols implement sequential consistency, because the primary orders all incoming writes in a globally unique order

Primary-Based Remote-Write Protocols

- Write operations on a data item are sent to the primary that is responsible for updating the data item
- **Note:** This protocol is *blocking*, which hampers performance, but is necessary to ensure sequential consistency
  - **It is blocking because a write must be propagated and acknowledged by all replicas before the next write can take place**



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

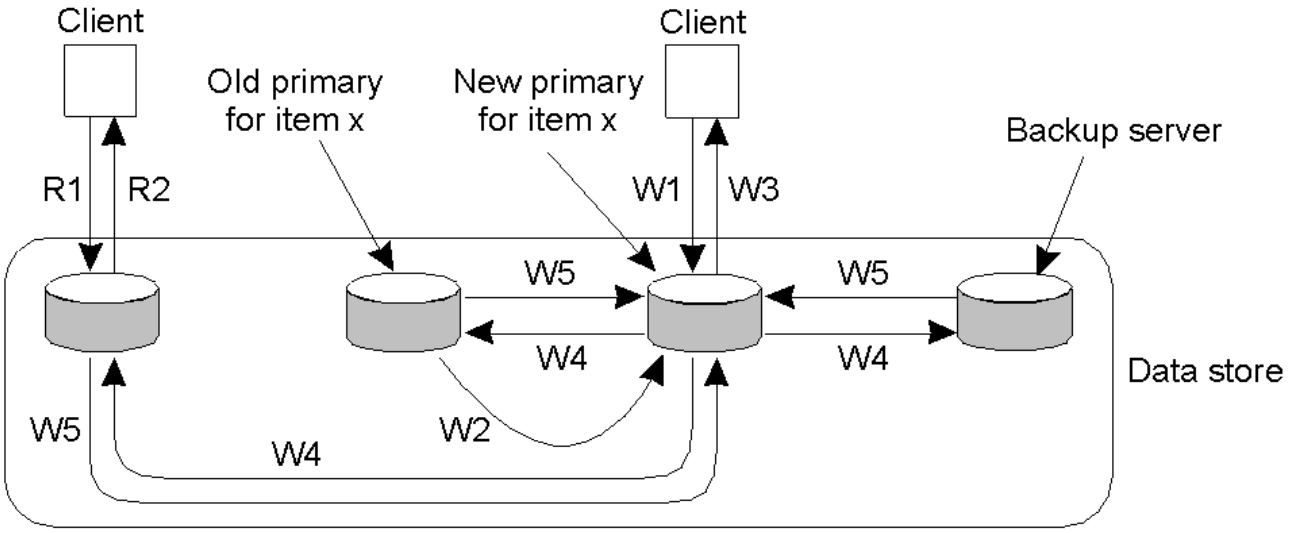
R1. Read request
R2. Response to read

Primary-Based Local-Write Protocols

- When a process wants to update a data item, it locates the primary copy and **migrates the data item to its own location**
  - **The process then updates the data item locally**
  - **It may perform several successive writes locally (which reduces communication costs)**
- Other processes may read the data item but may not update it
- In *non-blocking* mode, the process propagates an update as soon as it has performed the update
- Used in mobile computing systems, where a mobile node
  - **Migrates the data item to itself before it becomes disconnected**
  - **Performs updates on the data item while it is disconnected**
  - **Communicates the updates to the other nodes after it is reconnected**

Primary-Based Local-Write Protocols

- Data items are moved to a new primary so that it can update the data item locally

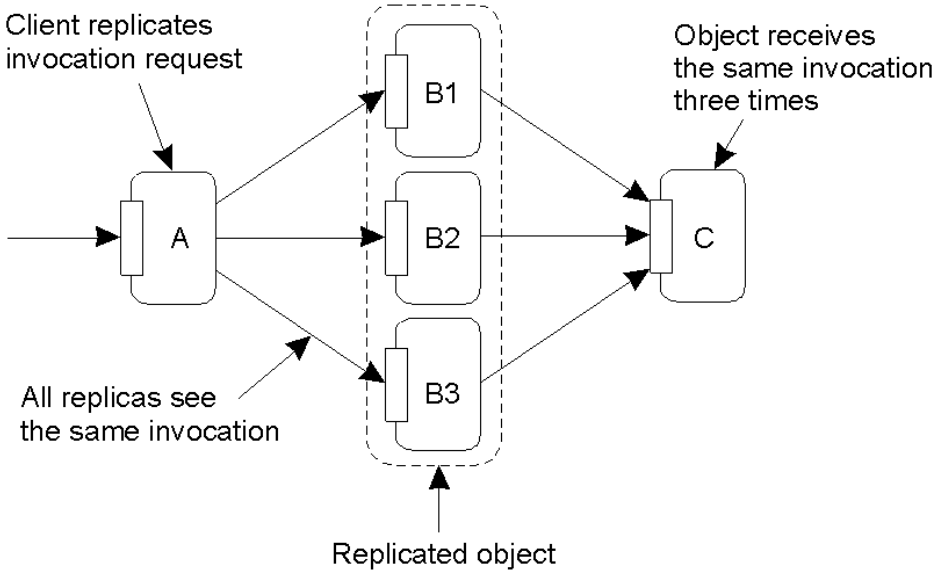


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

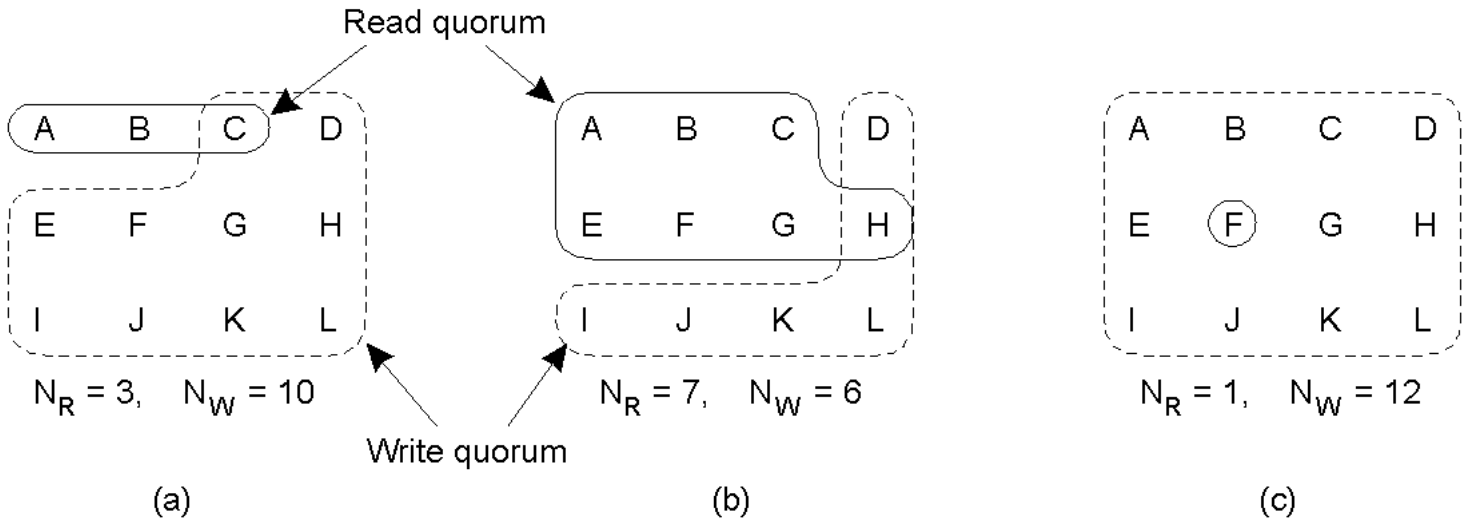
Replicated-Write Protocols

- **Active Replication Protocols:** *Operations* are sent to multiple *process replicas*, each of which performs the operations
  - **Operations must be performed in the same order at all process replicas, which requires totally-ordered multicast**
    - Centralized sequencer (Amoeba: Tanenbaum, et al.)
    - Rotating sequencer (Totem: Moser, Melliar-Smith, et al.)
  - **Duplicate operations must be detected and suppressed**



Replicated-Write Protocols

- **Quorum-Based Protocols:** Each operation is performed such that a *majority vote* is established
  - The set of replicas **written** must constitute a **majority**
  - The set of replicas **read** must intersect the set of replicas **written**



- Three examples of the voting algorithm
  (a) A correct choice of read and write set
  (b) An incorrect choice that may lead to write-write conflicts
  (c) A correct choice, known as Read One Write All (ROWA)

Cache-Coherence Protocols

- Caches are typically controlled by clients rather than servers, and differ in their coherence detection strategy and their coherence enforcement strategy:
- **Coherence detection strategy**
  - **Static: Compiler determines which operations might lead to inconsistencies (because the data are cached), and inserts instructions to avoid inconsistencies**
  - **Dynamic: Runtime system detects inconsistencies, e.g., by checking with the server to see whether a cached data item has been modified since it was cached**

Cache-Coherence Protocols

- **Coherence enforcement strategy:** Determines how a client cache is kept consistent with copies at the servers
  - **Do not allow clients to cache shared data; keep shared data only at the servers**
  - **Allow clients to cache shared data; when a server modifies a data item:**
    - **Server sends an invalidation to all client caches**
    - **Server propagates the update to all client caches**

Cache-Coherence Protocols

- **Question:** What happens when cached data are modified?
- With **read-only caches**, update operations are performed only by the server
- With **write-through caches**, the client can modify cached data and then forward the updates to the server
  - **Used in distributed file systems**
  - **Similar to primary-based local-write protocols**
  - **For *sequential consistency*, the client must first obtain *exclusive-write* access; otherwise, write-write conflicts can occur**
  - **Improves performance, because operations can be performed locally**
- With **write-back caches**, the client delays the propagation of updates by performing several writes before informing the servers
  - **Also used in distributed file systems**