Distributed Systems
COMP 212

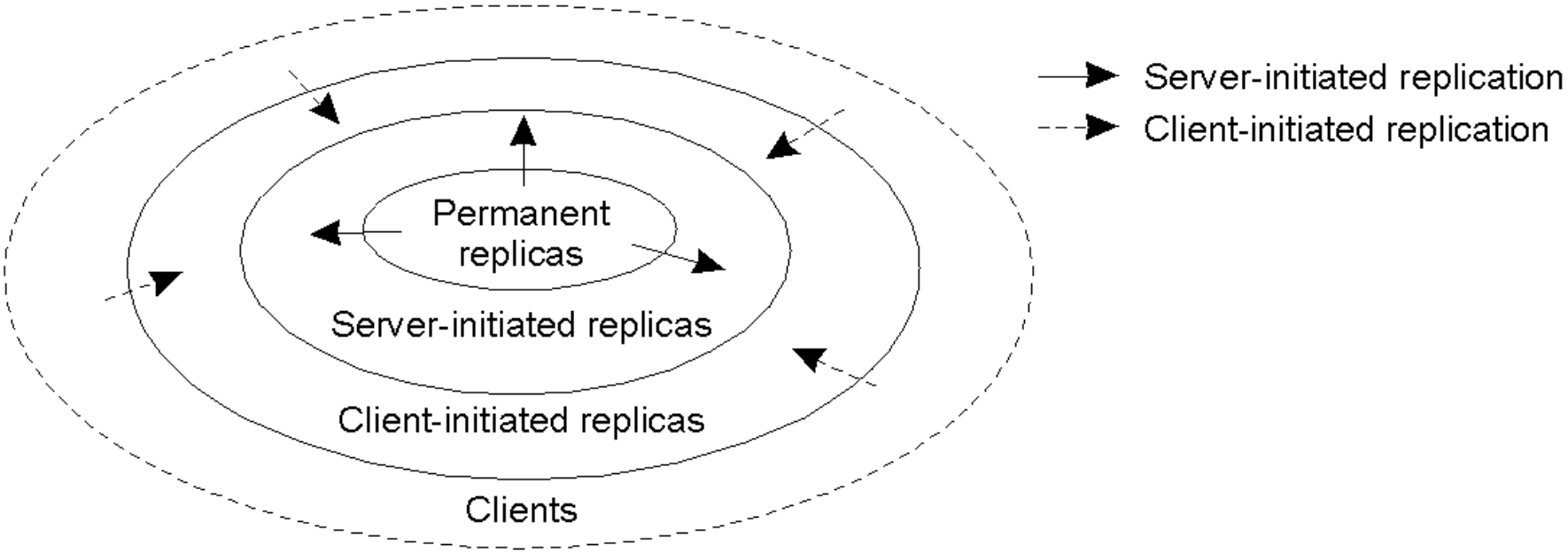
Lecture 24

Othon Michail

UNIVERSITY OF
LIVERPOOL

Distribution Protocols

- *Regardless of which consistency model is chosen, we need to decide* ***where***, ***when*** and ***by whom*** copies of the data-store are to be placed



Replica Placement Types

There are three types of replicas:

1. ***Permanent replicas***: tend to be small in number, organised as COWs (Clusters of Workstations) or mirrored systems. (Think of Web-servers or backup database servers)
2. ***Server-initiated replicas***: used to enhance performance at the initiation of the owner of the data-store. Typically used by web hosting companies to geographically locate replicas close to where they are needed most. (Often referred to as "push caches").
3. ***Client-initiated replicas***: created as a result of client requests – think of browser caches. Works well assuming, of course, that the cached data does not go stale too soon.

Update Propagation

1. Propagate notification of the update to other replicas
   - Often used for caches
   - Invalidation protocols
   - Works well when the read-to-write ratio is small
2. Transfer the data from one replica to another
   - works well when the read-to-write ratio is relatively high
   - Log the changes, aggregate updates
3. Propagate the update operation (aka active replication)
   - Less bandwidth, more processing power

Push vs. Pull Protocols

- Another design issue relates to whether the updates are *pushed* or *pulled*

1. ***Push-based/Server-based Approach***: sent "automatically" by server, the client does not request the update
   - Useful when a high degree of consistency is needed
   - Often used between permanent and server-initiated replicas

2. ***Pull-based/Client-based Approach***: used by client caches (e.g., browsers), updates are requested by the client from the server
   - No request, no update!

Push vs. Pull Protocols: Trade Offs

Issue	Push-based	Pull-based
What information should the server maintain	List of client replicas and caches	None
Messages that need to be sent between server and clients	Update (and possibly fetch update later in case the original update was just an invalidation)	Poll and update
Response time at client	Immediate (or fetch-update time in case of an invalidation initially)	Fetch-update time

- A comparison between push-based and pull-based protocols in the case of multiple client, single-server systems

Hybrid Approach: Leases

- A *lease* is a *contract* in which the server promises to push updates to a client until the lease expires
- Make lease expiration time-depended on system behaviour (adaptive leases)
- Age-based leases: an object that has not changed for a while probably will not change in the near future; provide longer lease
- Renewal-frequency leases: the more often a client requests a specific object, the longer the expiration time for that client (that object)
  - Server keeps updating those clients where its data are popular
- State-space overhead leases: the more loaded a server is, the shorter the expiration times become

Leases and Synchronisation Problems

- What if client's and server's clocks are not tightly synchronised?
  - The client may take a "pessimistic" view concerning the level at which its clock is synchronised with the server's clock and attempt to obtain a new lease before the current one expires

Epidemic Protocols

- This is an interesting class of protocols that can be used to implement Eventual Consistency
- The main concern is the propagation of updates to all the replicas in *as few a number of messages as possible*
- Of course, here we are spreading updates, not diseases!
- With this "update propagation model", the idea is to "infect" as many replicas as quickly as possible.
  - Removing data can be problematic
  - Old copies, from which the data has not been deleted yet may be interpreted as something new by a copy from which those data have been already deleted

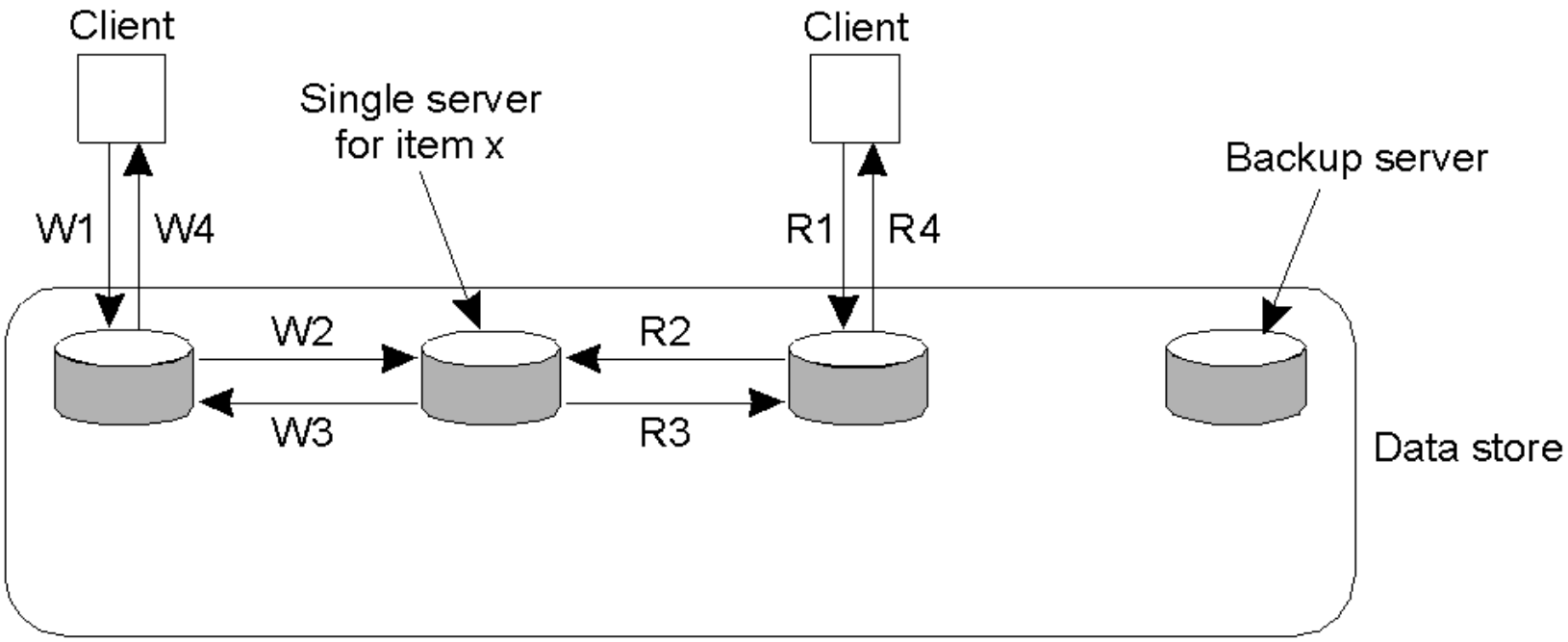
Consistency Protocols: Primary-Based Protocols

- Each data item is associated with a "primary" replica
- The primary is responsible for coordinating writes to the data item

- There are two types of Primary-Based Protocol:
  1. Remote-Write
  2. Local-Write

Remote-Write Protocols

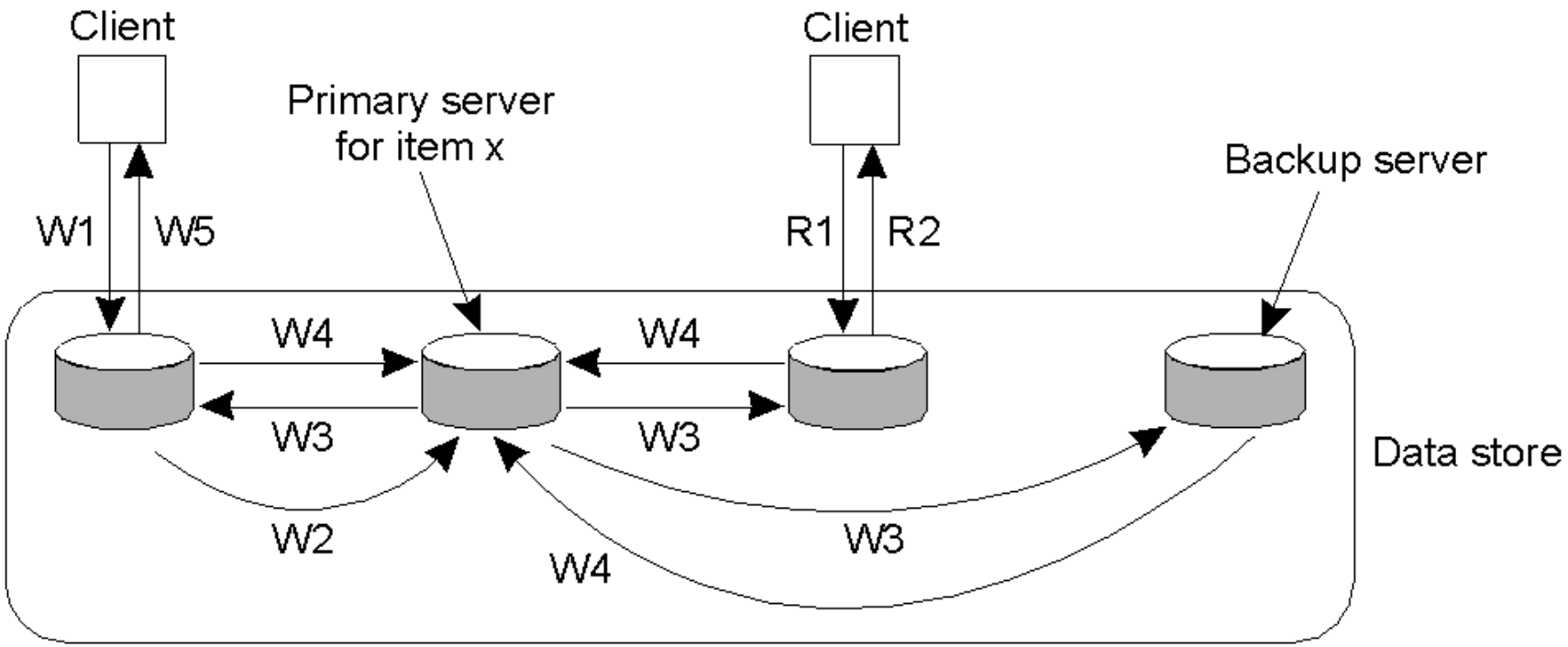
- With this protocol, all writes are performed at a single (remote) server
- This model is typically associated with traditional client/server systems



W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

R1. Read request
R2. Forward request to server for x
R3. Return response
R4. Return response

Primary-Backup Protocol: A Variation



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

- Writes are still centralised, but reads are now distributed
- The primary coordinates writes to each of the backups

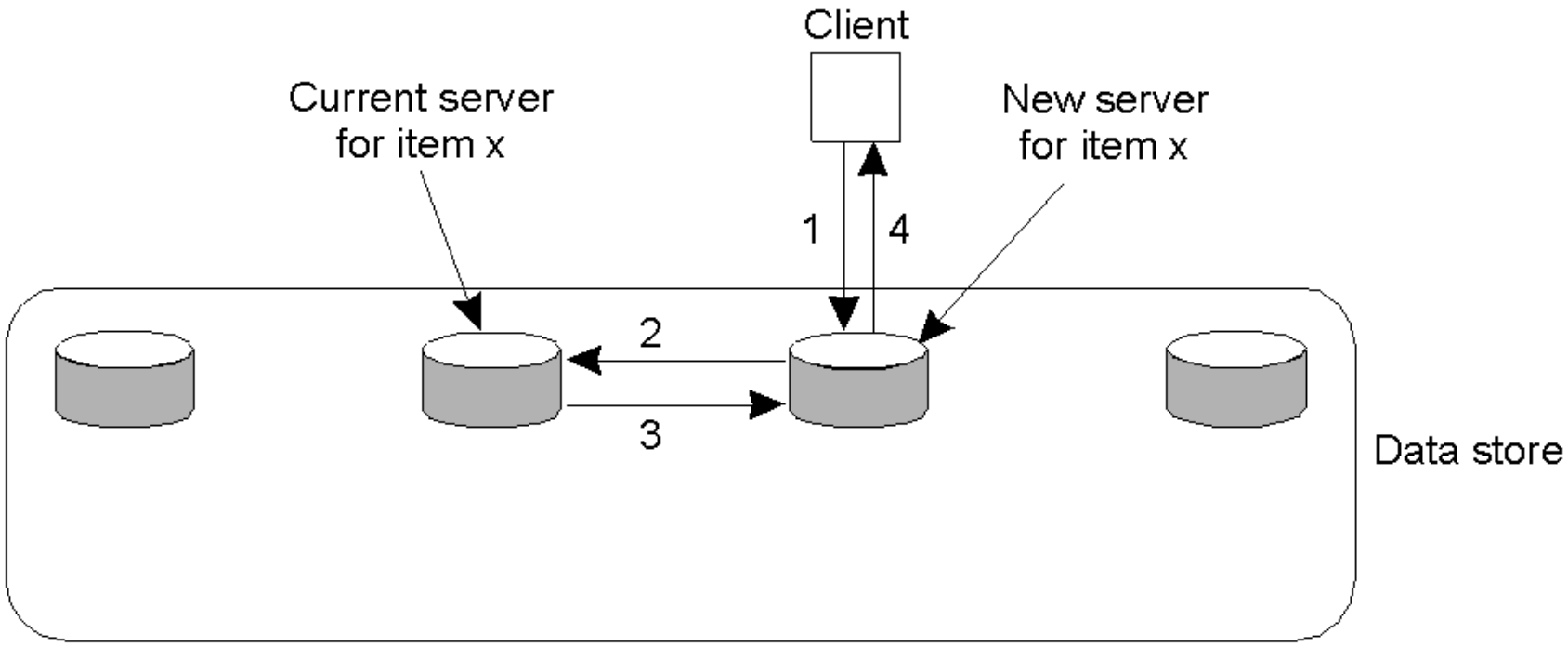
The Bad and Good of Primary-Backup

- Bad: Performance!
  - *All of those writes can take a long time (especially when a "blocking write protocol" is used)*
- Good: Using a non-blocking write protocol to handle the updates
  - But can lead to the client not knowing whether the update is backed up
- Good: The benefit of this scheme is, as the *primary* is in control, all writes can be sent to each backup replica *IN THE SAME ORDER*, making it easy to implement *sequential consistency*
  - The benefits of a centralised/coordinated approach

Local-Write Protocols

- In this protocol, a single copy of the data item is still maintained
- Upon a write, the data item gets transferred to the replica that is writing
- That is, the status of *primary* for a data item is *transferrable*
- This is also called a "fully migrating approach"

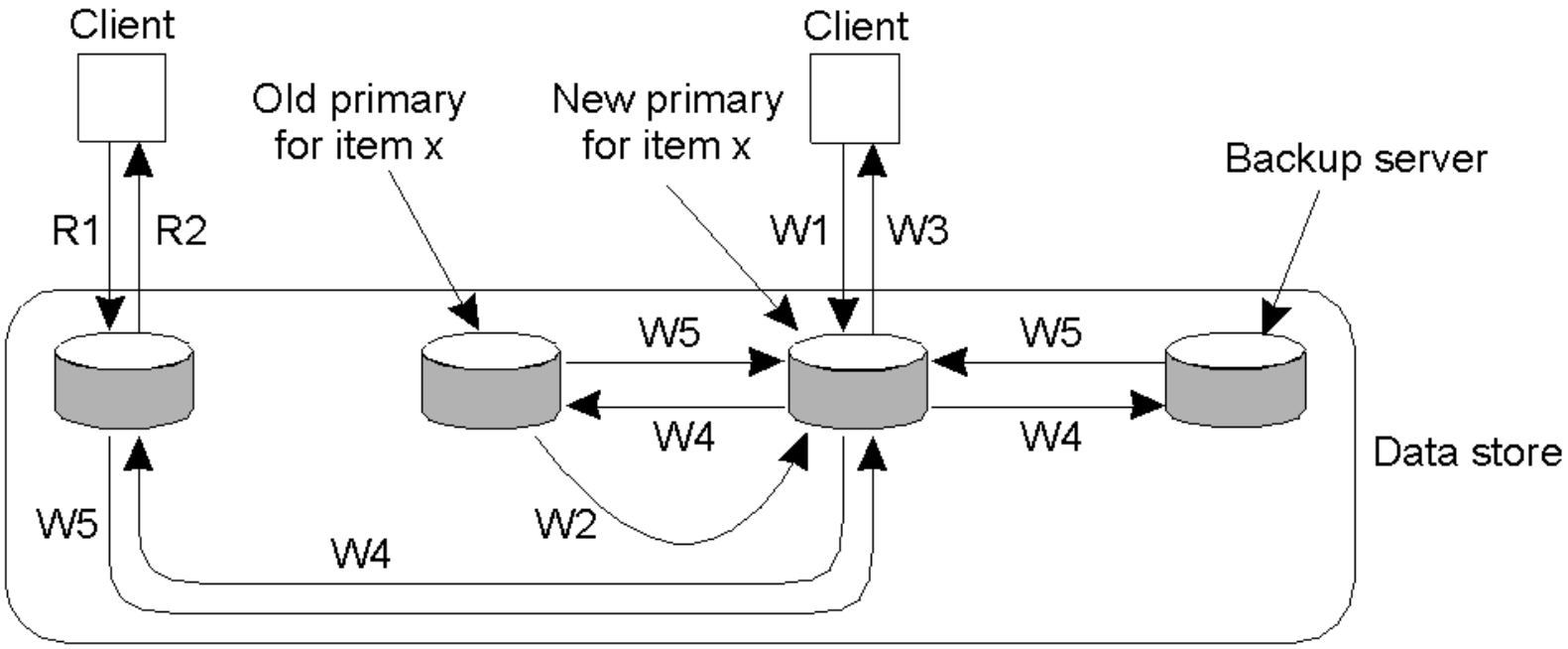
Local-Write Protocols Example



1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

- Primary-based local-write protocol in which a single copy is migrated between processes (prior to the read/write)

Local-Write Protocols: A Variation



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

- Primary-backup protocol in which the primary migrates to the process wanting to perform an update, then updates the backups
- Consequently, reads are much more efficient

Local-Write Issues

- The big question to be answered by any process about to read from or write to the data item is:
  - *"Where is the data item right now?"*

- It is possible to use some of the *dynamic naming technologies,* but scaling quickly becomes an issue

- Processes can spend more time actually locating a data item than using it!

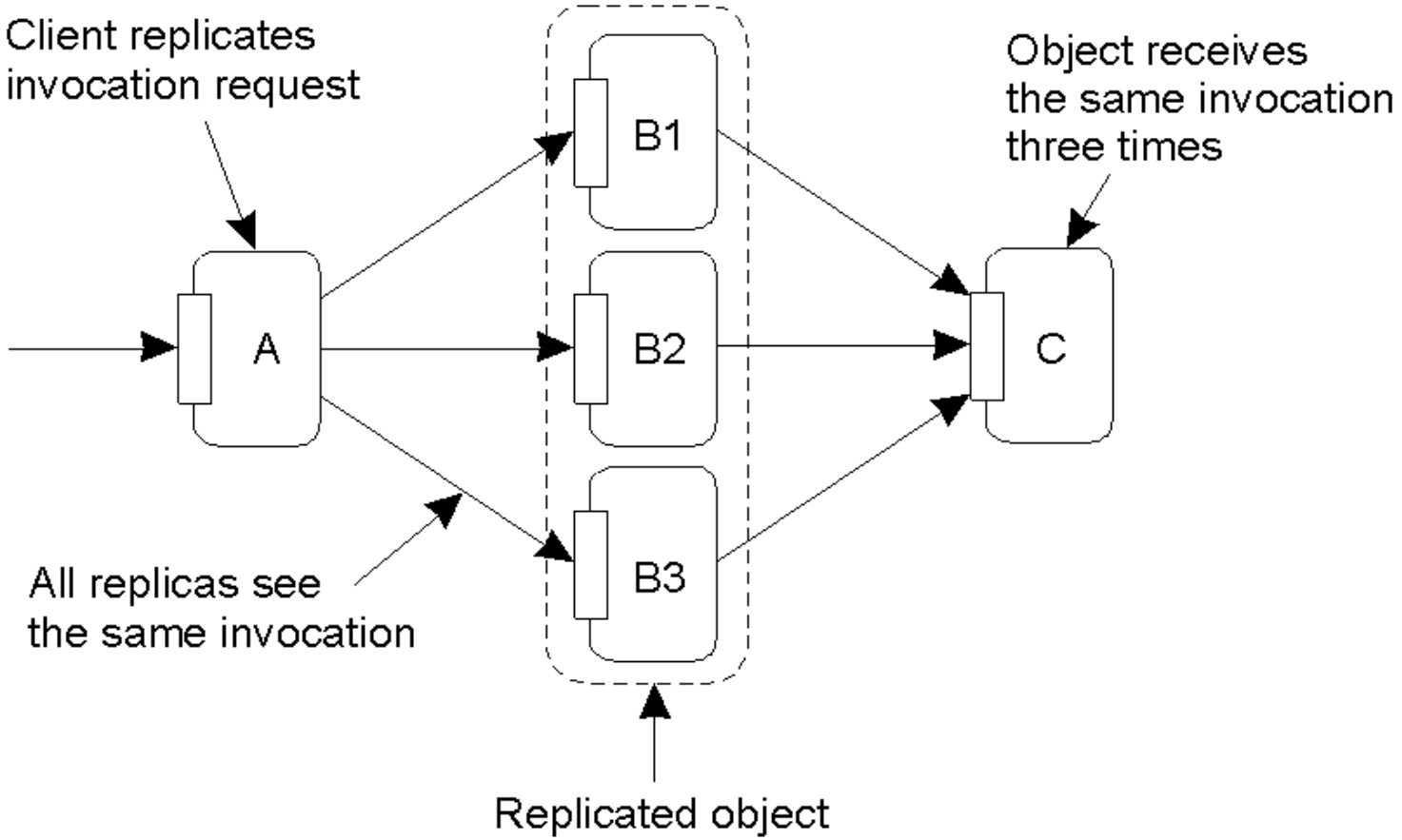
Consistency Protocols: Replicated-Write Protocols

- With these protocols, writes can be carried out at *any* replica.
- Another name might be: "Distributed-Write Protocols"

Example: Active Replication

- A special process carries out the update operations at each replica
  - Lamport's timestamps can be used to achieve total ordering, but this does not scale well within Distributed Systems
- An alternative/variation is to use a *sequencer*, which is a process that assigns a unique ID# to each update, which is then propagated to all replicas
- This can lead to another problem: *replicated invocations*

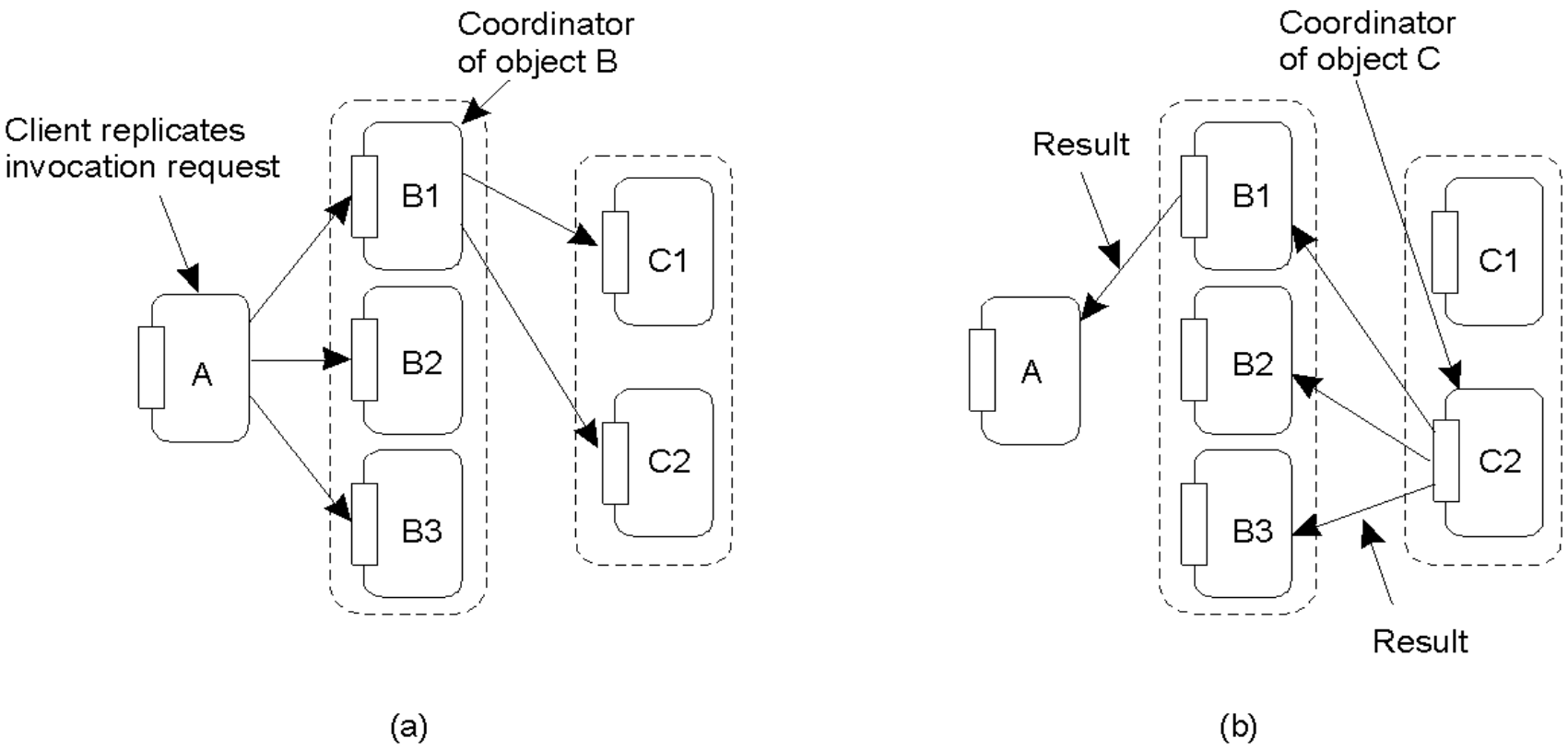
Active Replication: The Problem



- The problem of replicated invocations – ‘B’ is a replicated object (which itself calls ‘C’). When ‘A’ calls ‘B’, how do we ensure ‘C’ isn’t invoked three times?

Active Replication: Solutions

- Using a replication-aware communication layer on top of which replicated objects execute



a) Using a coordinator for 'B', which is responsible for forwarding an invocation request from the replicated object to 'C'

b) Returning results from 'C' using the same idea: a coordinator is responsible for returning the result to all 'B's. Note the single result returned to 'A'.

Summary

- To distribute (or “propagate”) updates, we draw a distinction between *WHAT* is propagated, *WHERE* it is propagated and by *WHOM*
  - Permanent, server-initiated, client-initiated replicas
  - Push vs Pull and leases
  - Epidemic protocols
  - Primary-based protocols
  - Active replication