Distributed Systems

LECTURE 26

Consistency protocols

Consistency Protocols

- This is a specific implementation of a consistency model.
- The most widely implemented models are:

1. Sequential Consistency.
2. Weak Consistency (with sync variables).
3. Atomic Transactions.

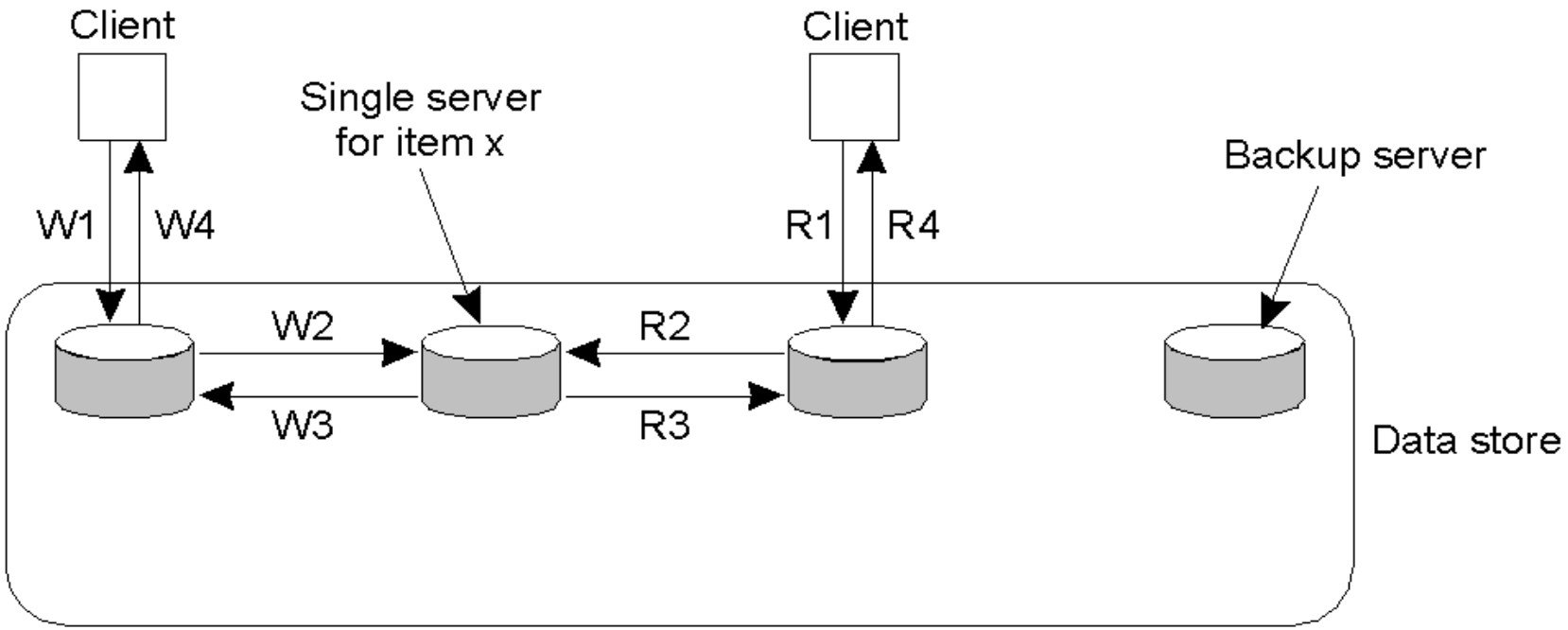
Primary-Based Protocols

- 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

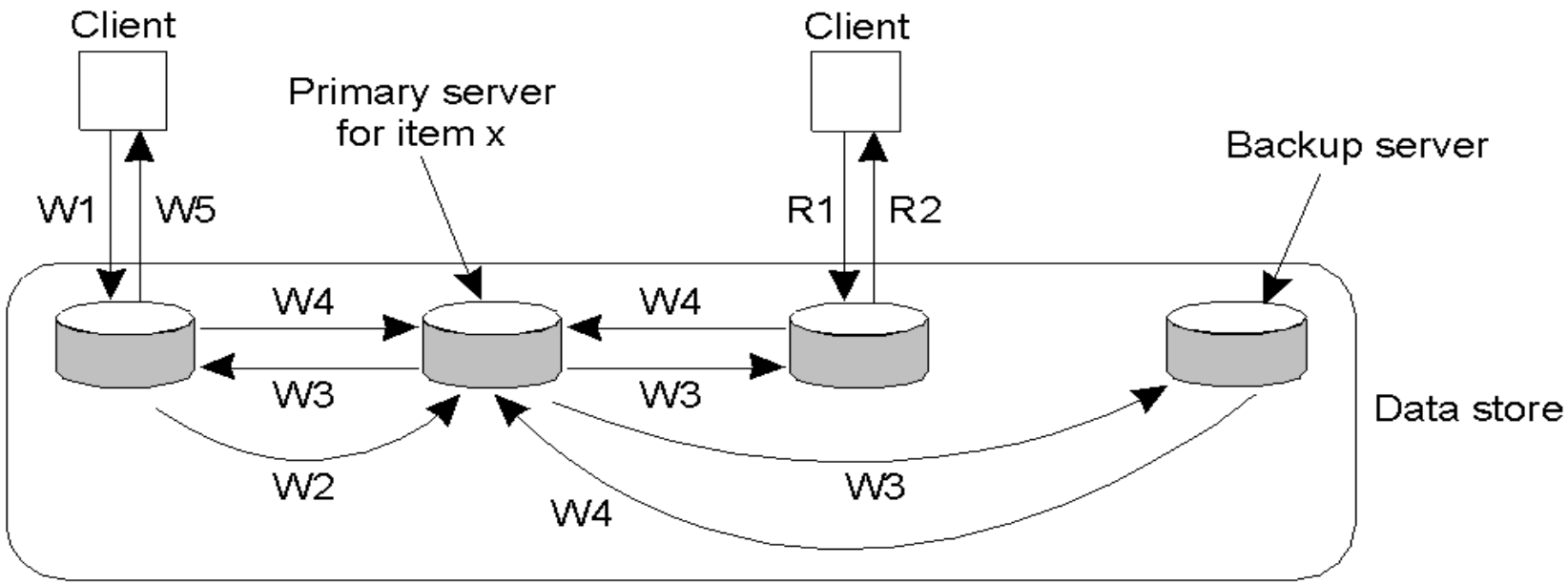
- With this protocol, all writes are performed at a single (remote) server.
- This model is typically associated with traditional client/server systems.



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.

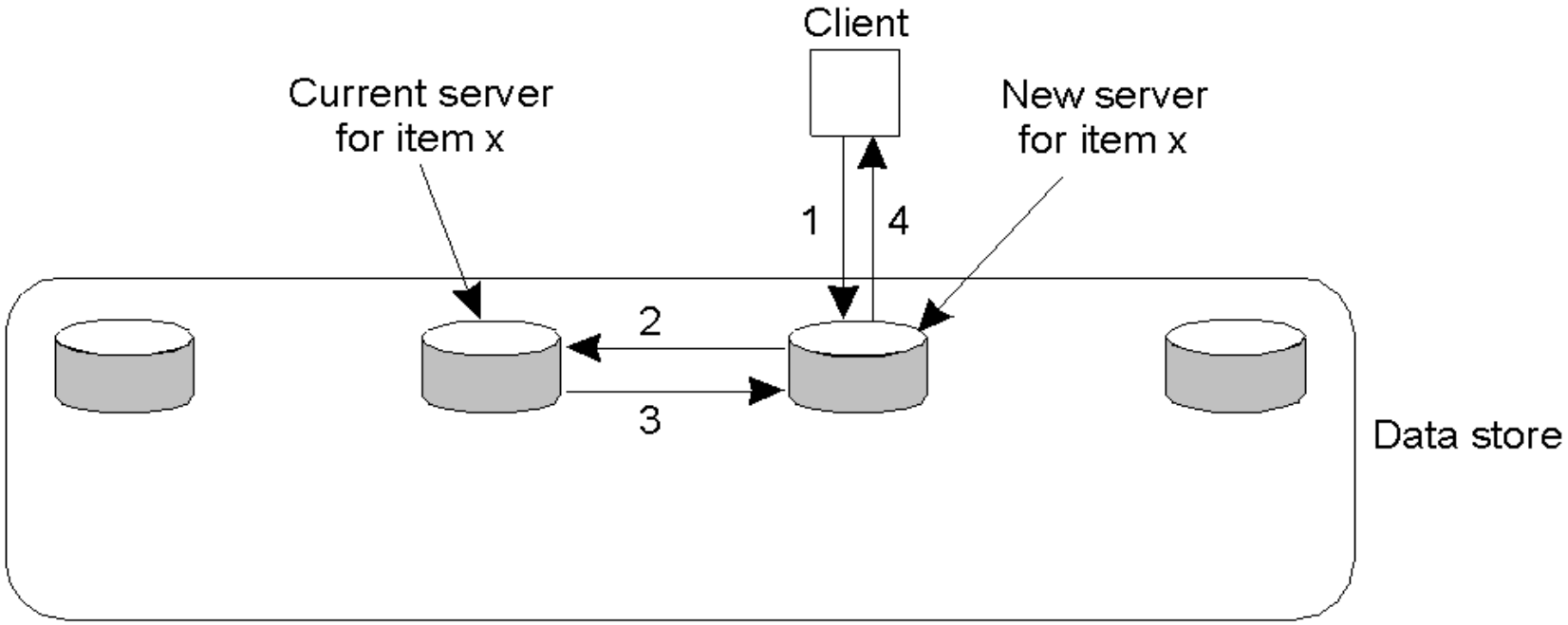
The Bad and Good of Primary-Backup

- Bad: Performance!
- *All of those writes can take a long time (especially when a "blocking write protocol" is used).*
- Using a non-blocking write protocol to handle the updates can lead to fault tolerant problems (which is our next topic).
- 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*.

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 *transferable*.
- This is also called a “fully migrating approach”.

Local-Write Protocols Example



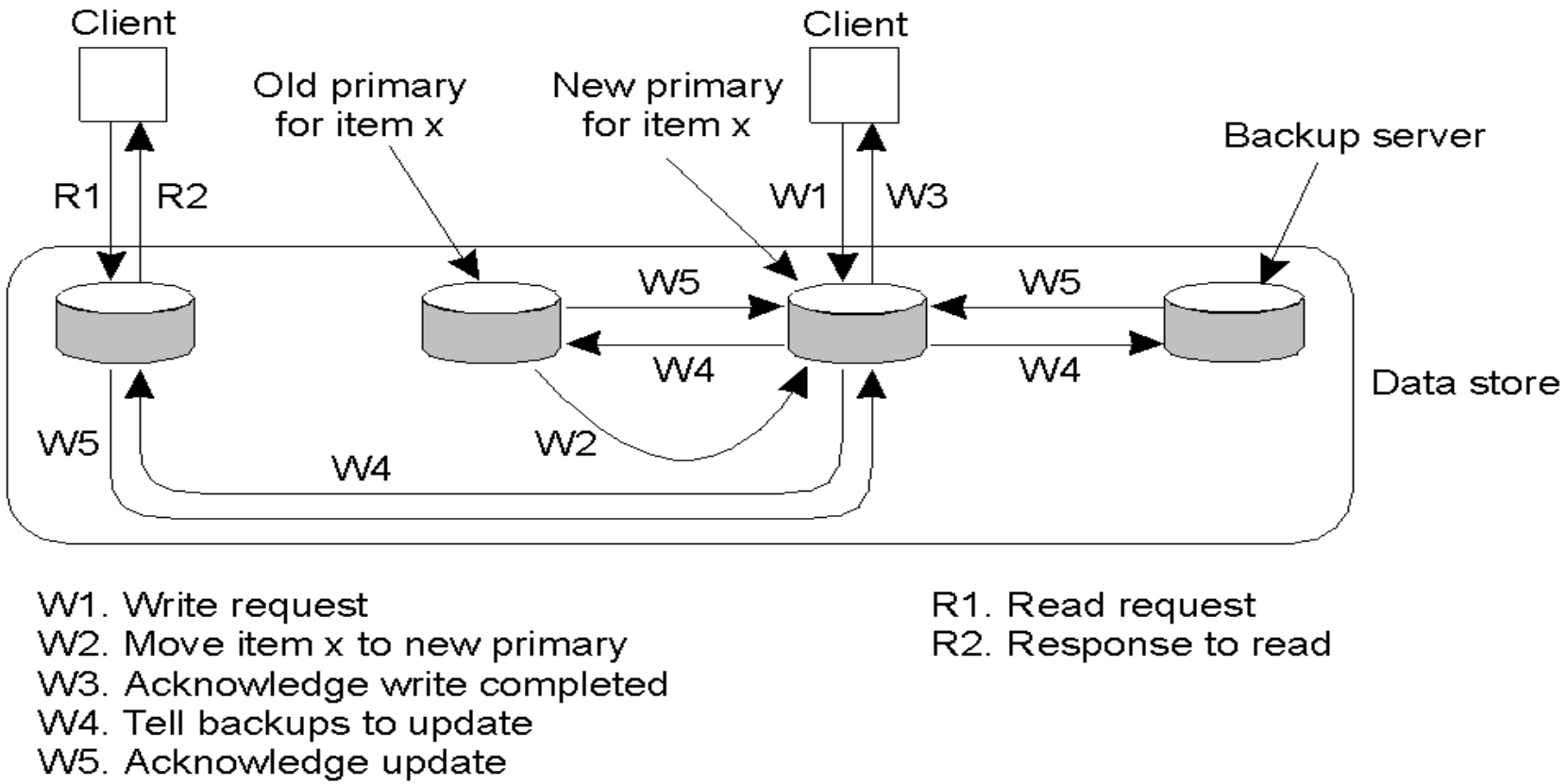
1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

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

Local-Write 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* studied earlier in this course, but scaling quickly becomes an issue.
- Processes can spend more time actually locating a data item than using it!

Local-Write Protocols: A Variation



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

Replicated-Write Protocols

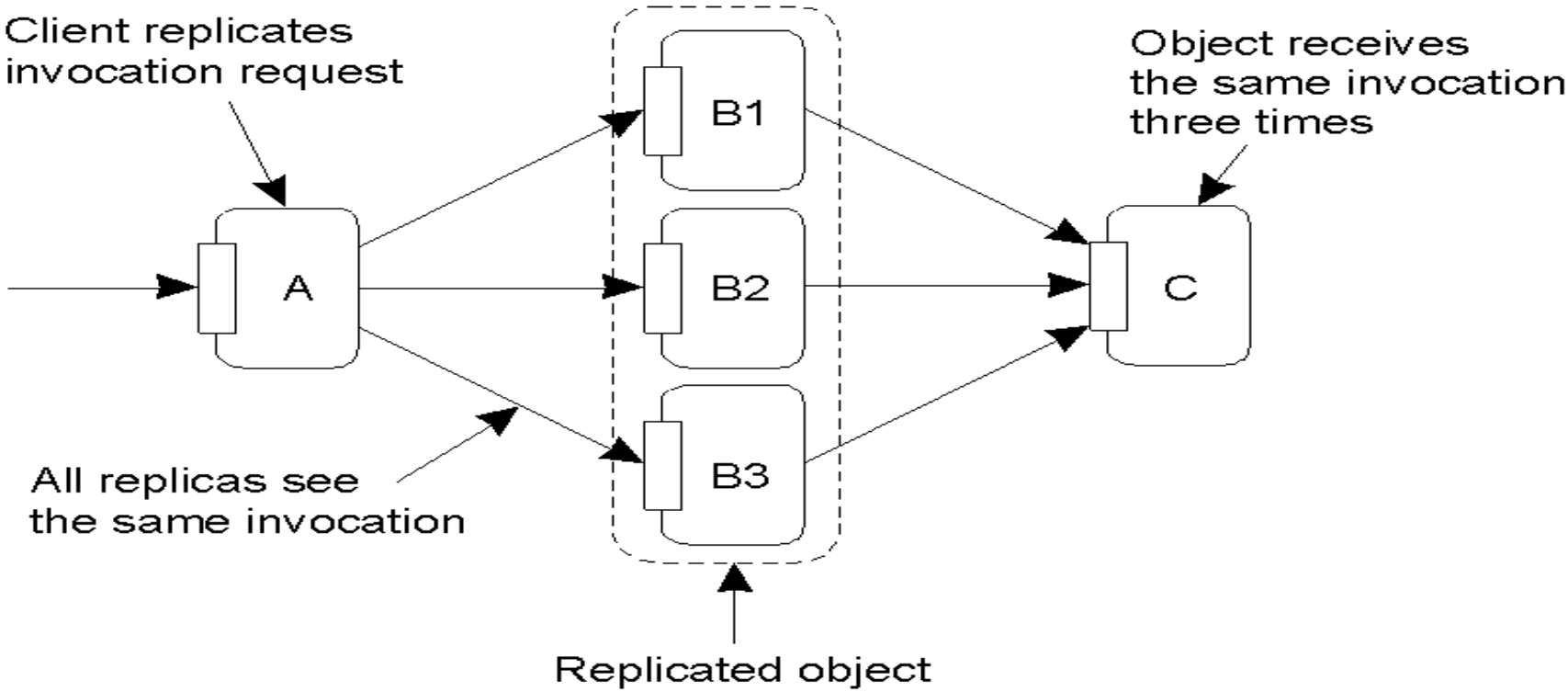
- With these protocols, writes can be carried out at *any* replica.
- Another name might be: “Distributed-Write Protocols”
- There are two types:

1. Active Replication.
2. Majority Voting (Quorums).

Active Replication

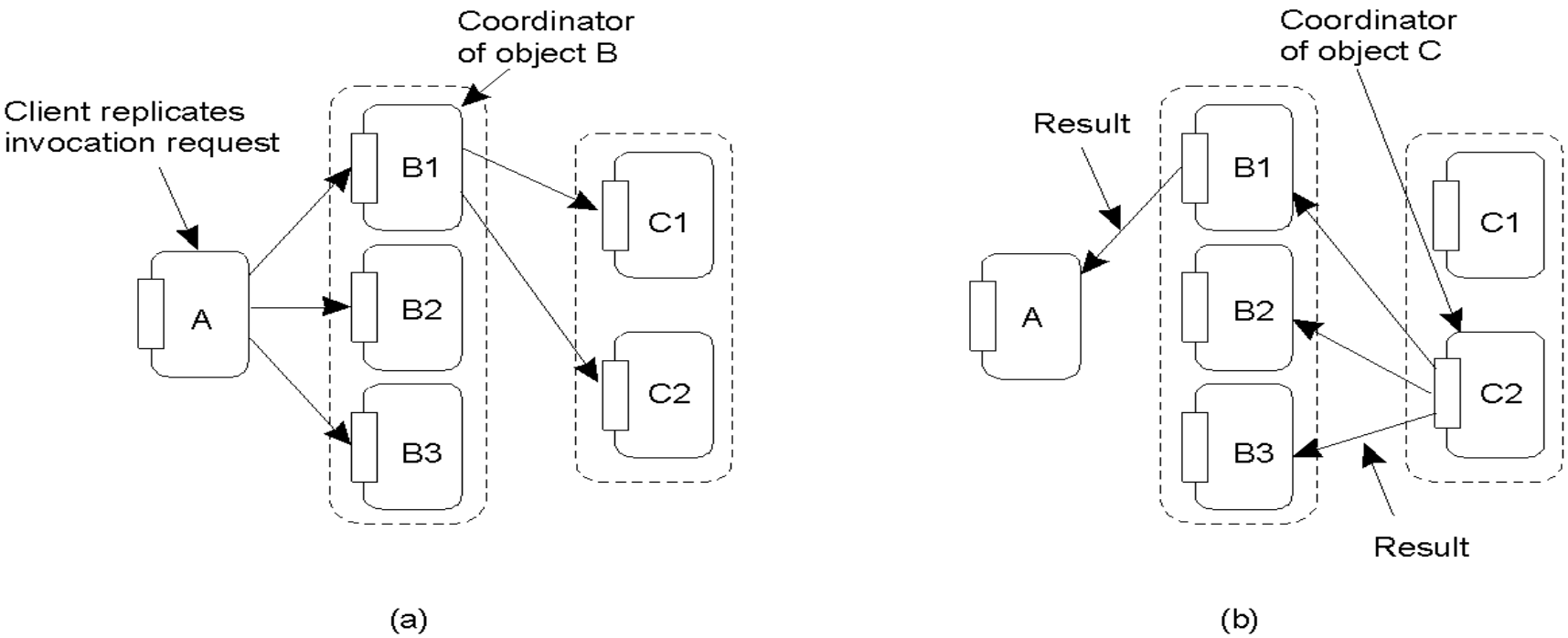
- A special process carries out the update operations at each replica.
- Lamport's timestamps can be used to achieve total ordering, but this does not scale well within Distributed Systems.
- An alternative/variation is to use a *sequencer*, which is a process that assigns a unique ID# to each update,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



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’.

Quorum-Based Protocols

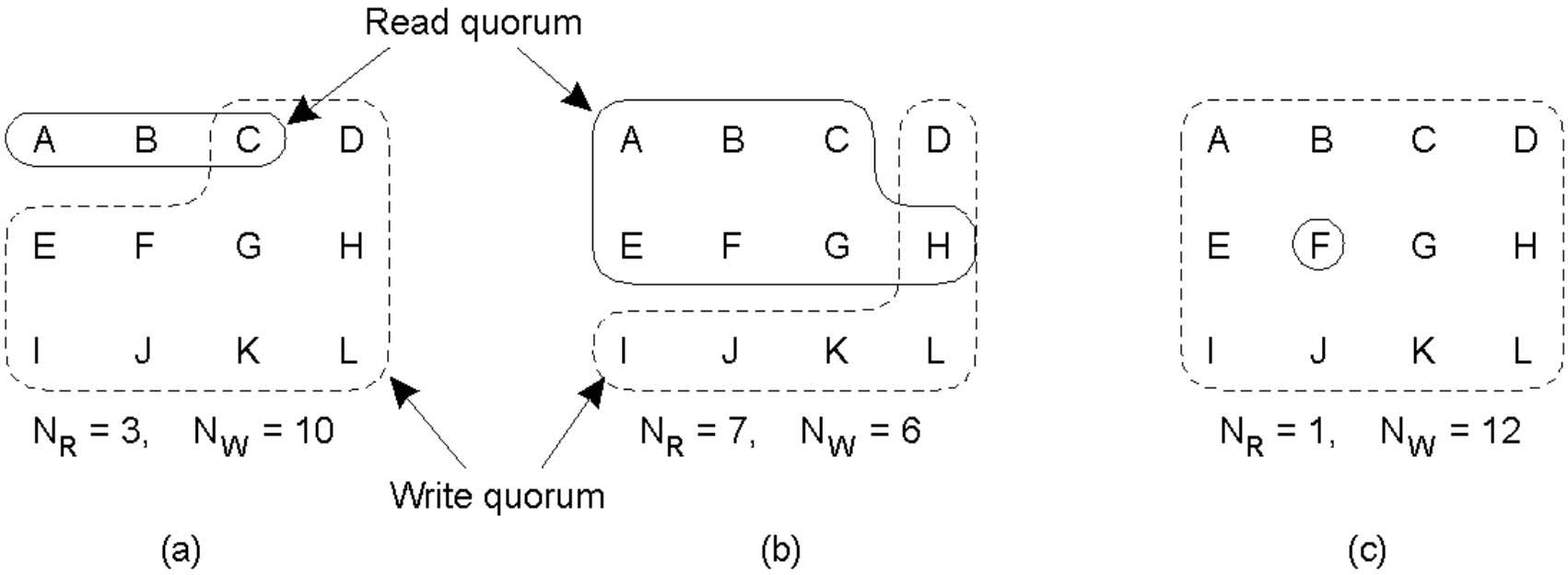
- Clients must request and acquire permissions from multiple replicas before either reading/writing a replicated data item.
- Consider this example:
  - A file is replicated within a distributed file system.
  - To update a file, a process must get approval from a majority of the replicas to perform a write. The replicas need to agree *to also perform the write*.
  - After the update, the file has a new version # associated with it (and it is set at all the updated replicas).
  - To read, a process contacts a majority of the replicas and asks for the version # of the files. If the version # is the same, then the file must be the most recent version, and the read can proceed.

Quorum Protocols: Generalisation

$$N_R + N_W > N$$

$$N_W > N/2$$

Quorum-Based Protocols



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) A correct choice, known as ROWA (read one, write all)

Cache-Coherence Protocols

- These are a special case, as the cache is typically controlled by the client *not* the server.
- *Coherence Detection Strategy*:
  - When are inconsistencies actually detected?
    - Statically at compile time: extra instructions inserted.
    - Dynamically at runtime: code to check with the server.
- *Coherence Enforcement Strategy*
  - How are caches kept consistent?
    - Server Sent: invalidation messages.
    - Update propagation techniques.
- Combinations are possible

Orca

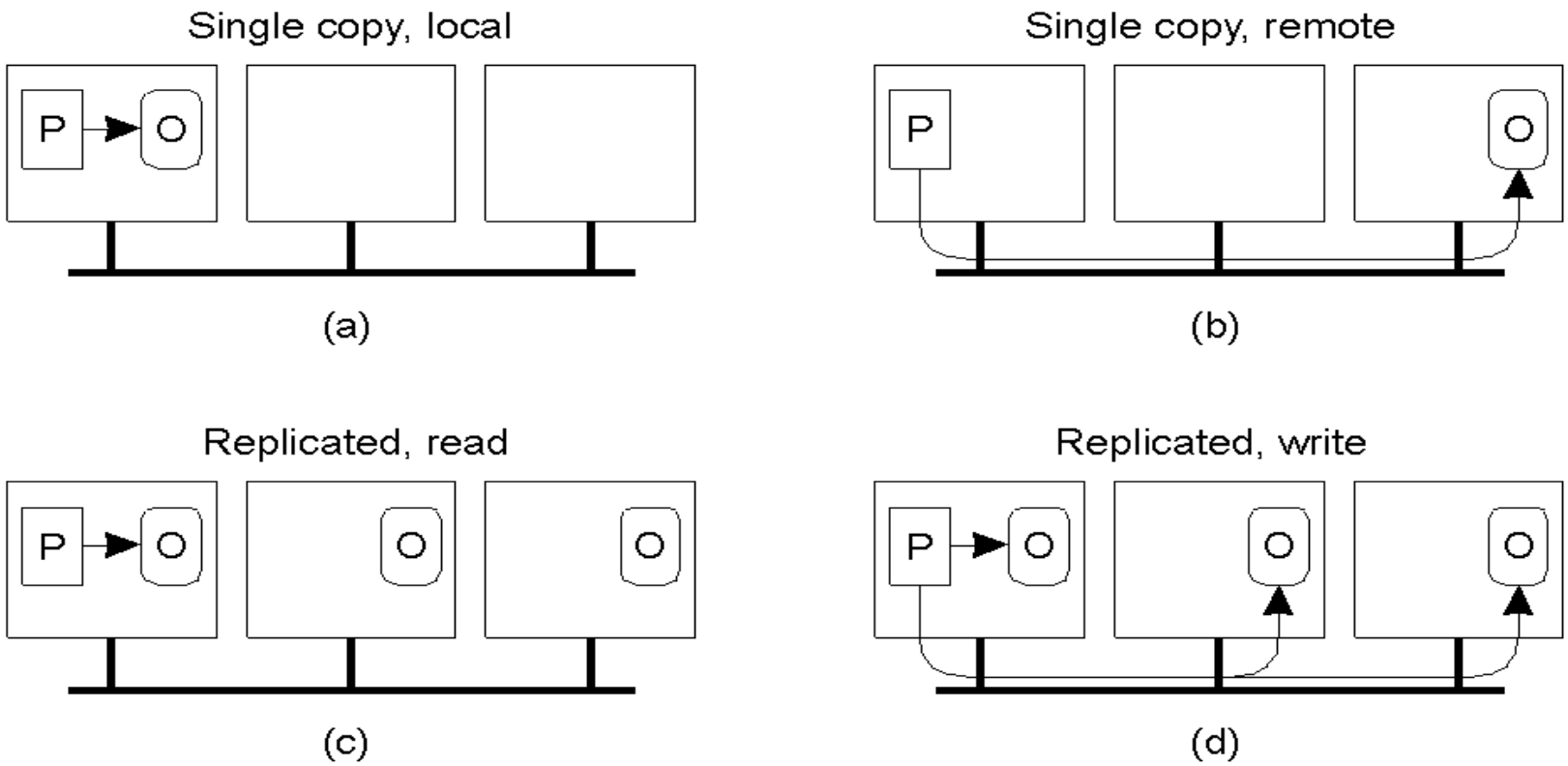
```
OBJECT IMPLEMENTATION stack;
  top: integer;                                  # variable indicating the top
  stack: ARRAY[integer 0..N-1] OF integer        # storage for the stack

  OPERATION push (item: integer)                 # function returning nothing
  BEGIN
    GUARD top < N DO
        stack [top] := item;                     # push item onto the stack
        top := top + 1;                          # increment the stack pointer
    OD;
  END;

  OPERATION  pop():integer;                      # function returning an integer
  BEGIN
    GUARD top > 0 DO                             # suspend if the stack is empty
         top := top – 1;                         # decrement the stack pointer
        RETURN stack [top];                      # return the top item
    OD;
  END;
BEGIN
  top := 0;                                      # initialization
END;
```

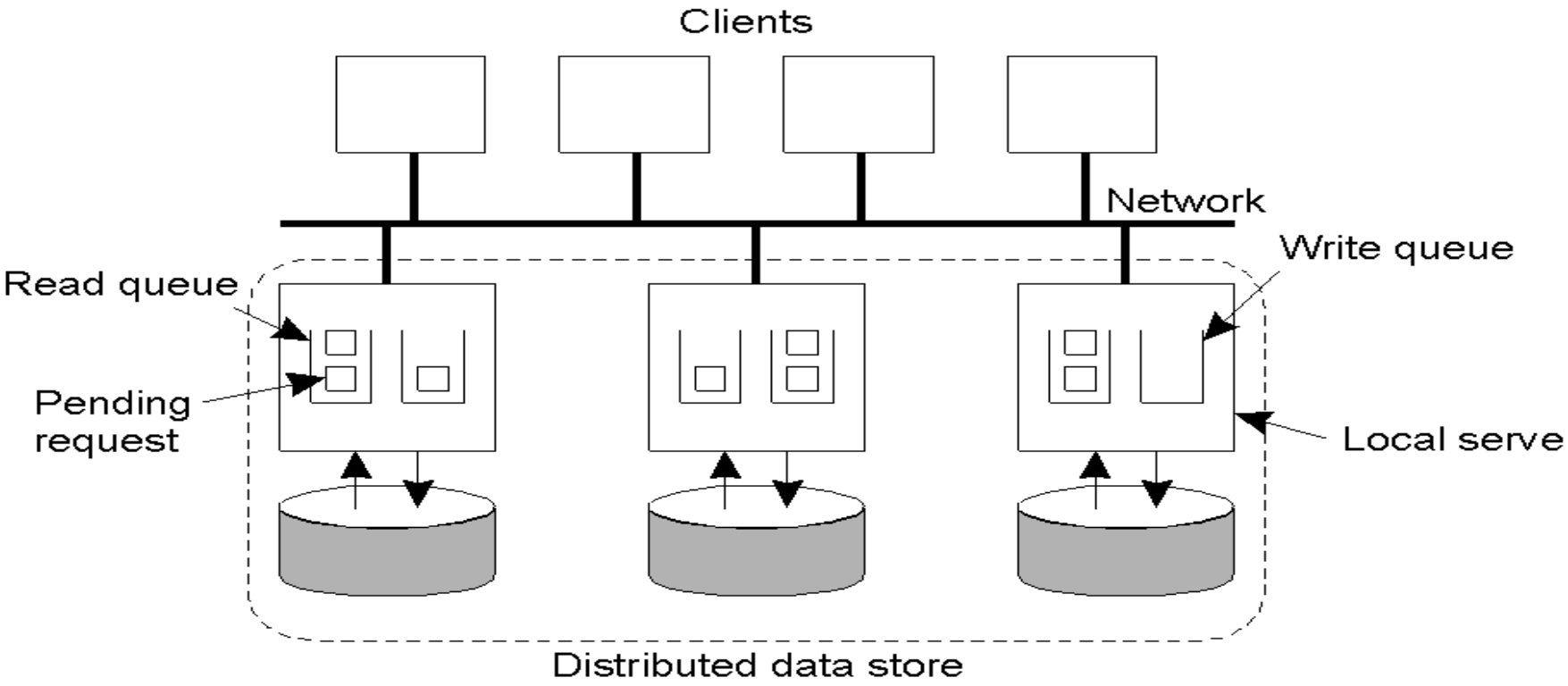
A simplified stack object in Orca, with internal data and two operations.

Management of Shared Objects in Orca



Four cases of a process *P* performing an operation on an object *O* in Orca.

Casually-Consistent Lazy Replication

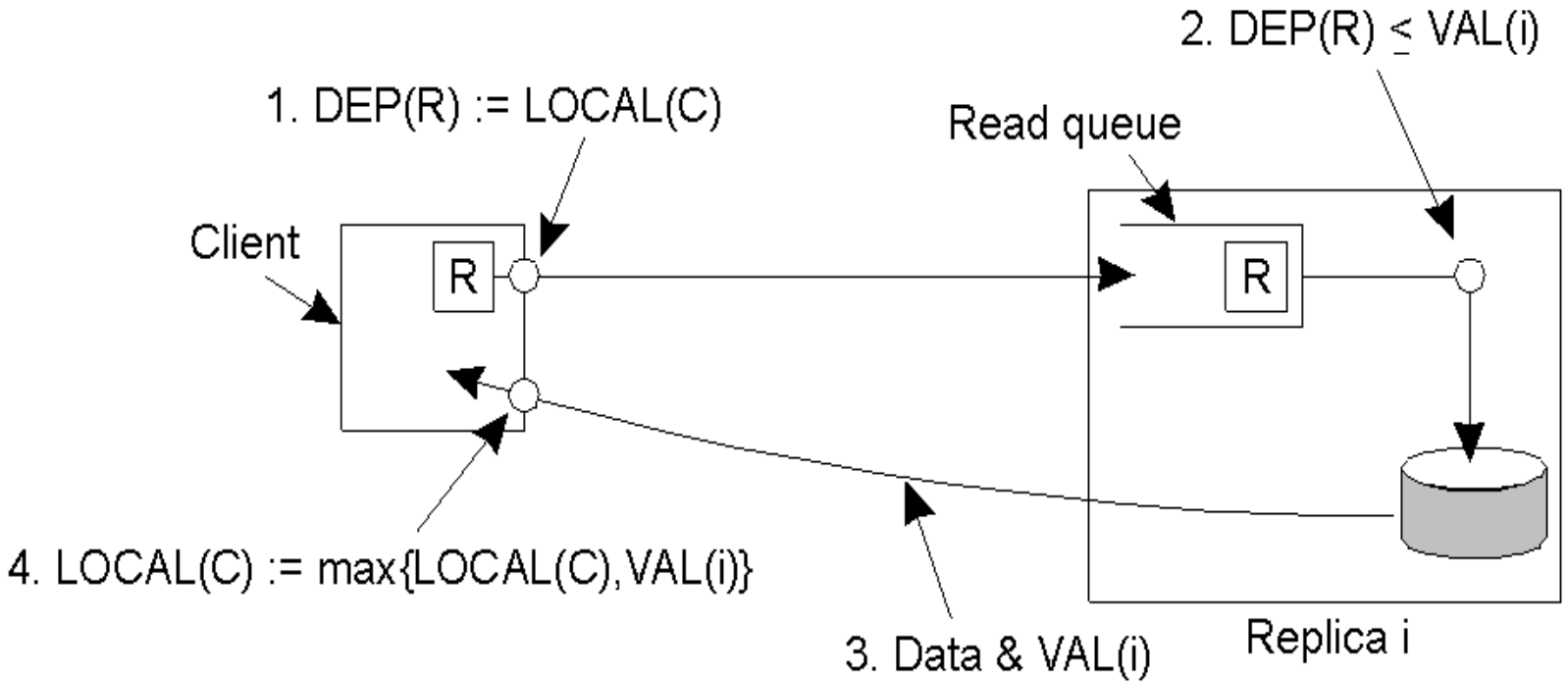


The general organization of a distributed data store. Clients are assumed to also handle consistency-related communication.

What about Writes to the Cache?

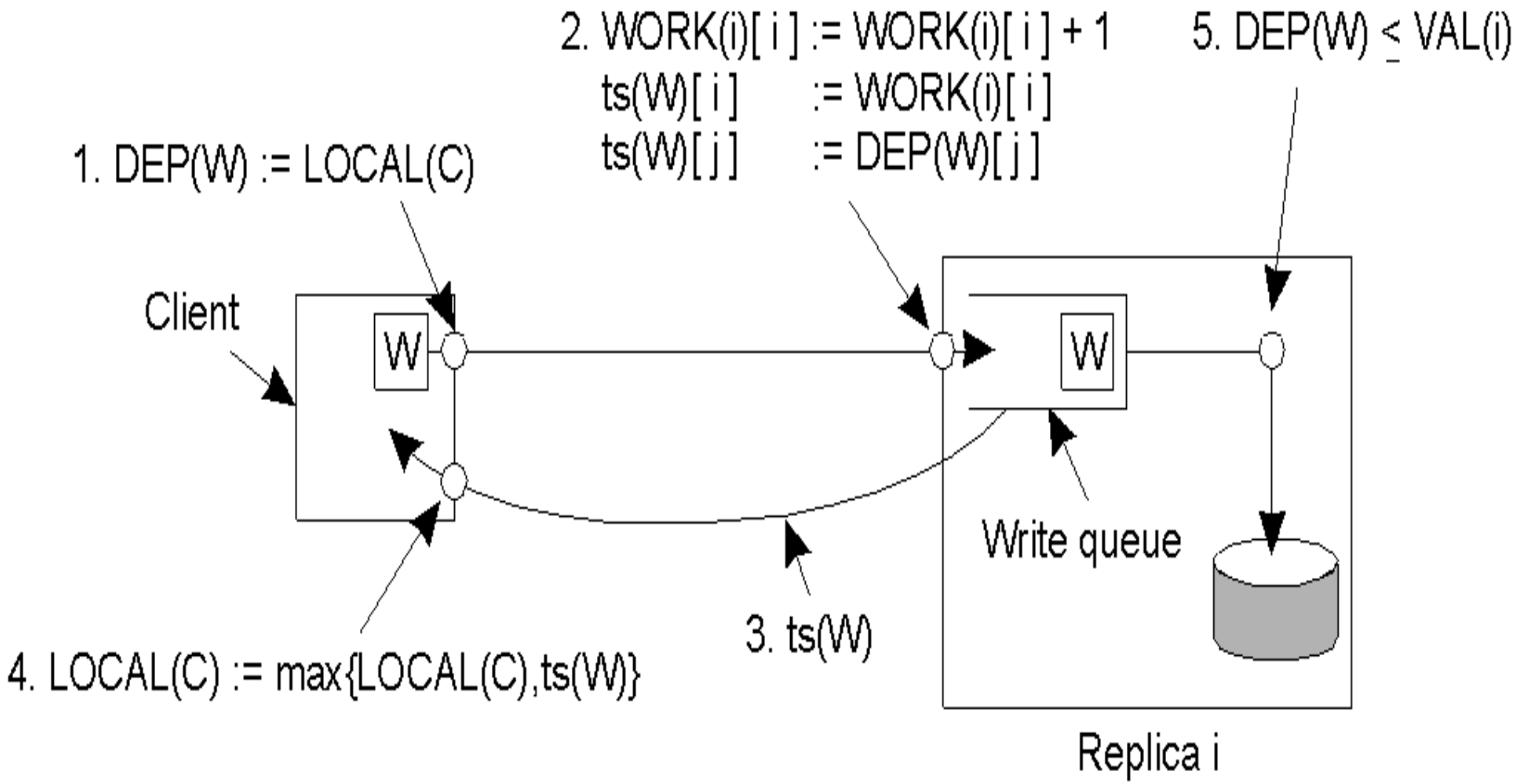
- *Read-only Cache*: updates are performed by the server (i.e., pushed) or by the client (i.e., pulled whenever the client notices that the cache is *stale*).
- *Write-Through Cache*: the client modifies the cache, then sends the updates to the server.
- *Write-Back Cache*: delay the propagation of updates, allowing multiple updates to be made locally, then sends the most recent to the server (this can have a dramatic *positive* impact on performance)

Processing Read Operations



Performing a read operation at a local copy.

Processing Write Operations



Performing a write operation at a local copy.

Summary (1) (Consistency and Replication)

- Reasons for replication: improved *performance*, improved *reliability*.
- Replication can lead to *inconsistencies* …
- How best can we *propagate updates* so that these inconsistencies are not noticed?
- With “best” meaning “without crippling performance”.
- The proposed solutions resolve around the relaxation of any existing consistency constraints.

Summary (2) (Consistency and Replication)

- Various consistency models have been proposed:
- *Strict*, *Sequential*, *Causal*, *FIFO* concern themselves with individual reads/writes to data items.
- Weaker models introduce the notion of synchronisation variables: *Release*, *Entry* concern themselves with a group of reads/writes.
- These models are known as “Data-Centric”.
- “Client Centric” models also exist:
  - Concerned with maintaining consistency for a single clients’ access to the distributed data-store.

Summary (3) (Consistency and Replication)

- To distribute (or “propagate”) updates, we draw a distinction between *WHAT* is propagated, *WHERE* it is propagated and by *WHOM*.
- We looked at various *Distribution Protocols* and *Consistency Protocols* designed to facilitate the propagation of updates.
- The most widely implemented schemes are those that support *Sequential Consistency* or *Weak Consistency* with *Synchronisation*

ASSIGNMENT

- Q: Explain all consistency protocols in detail.