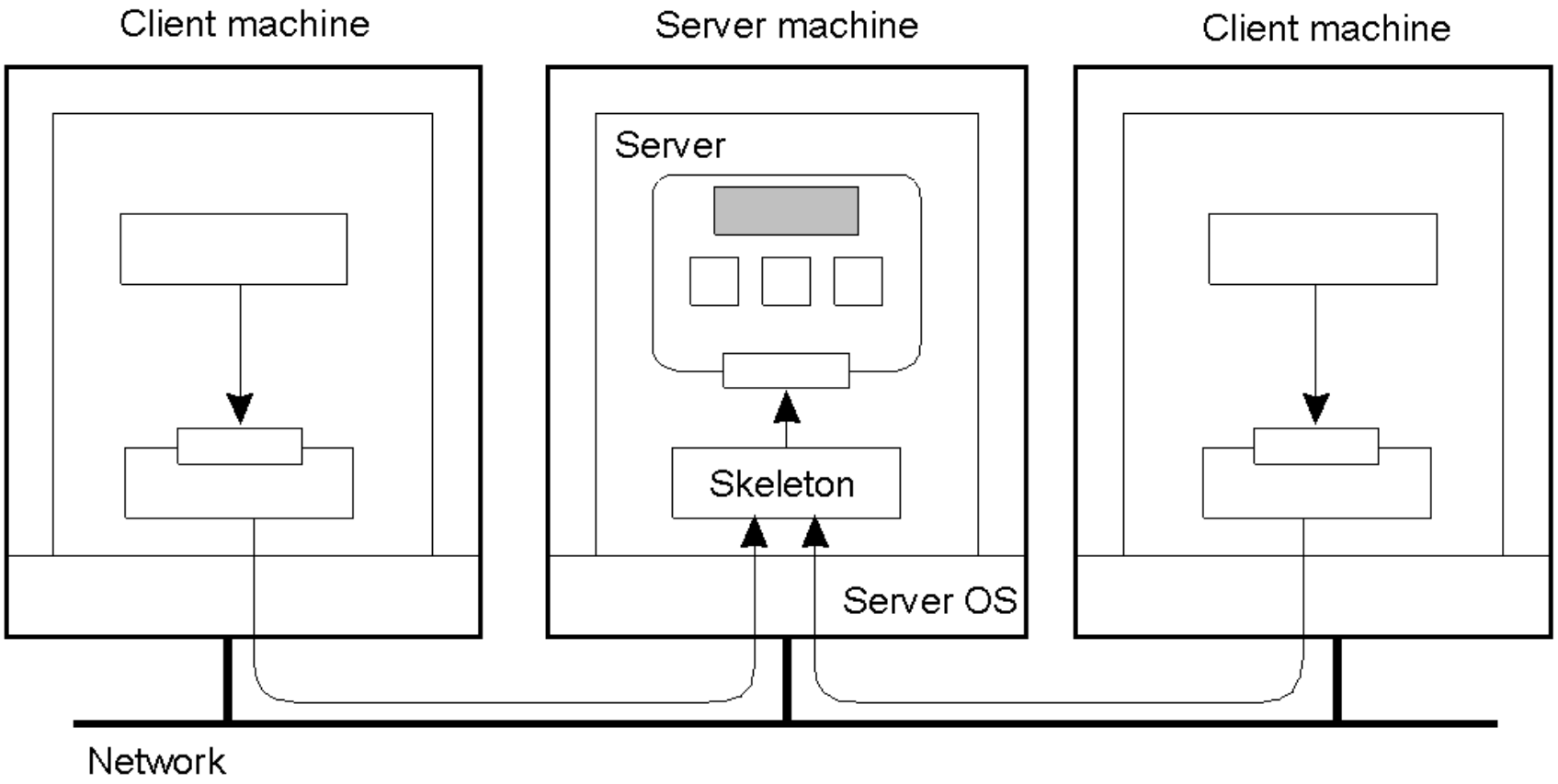
Consistency and Replication

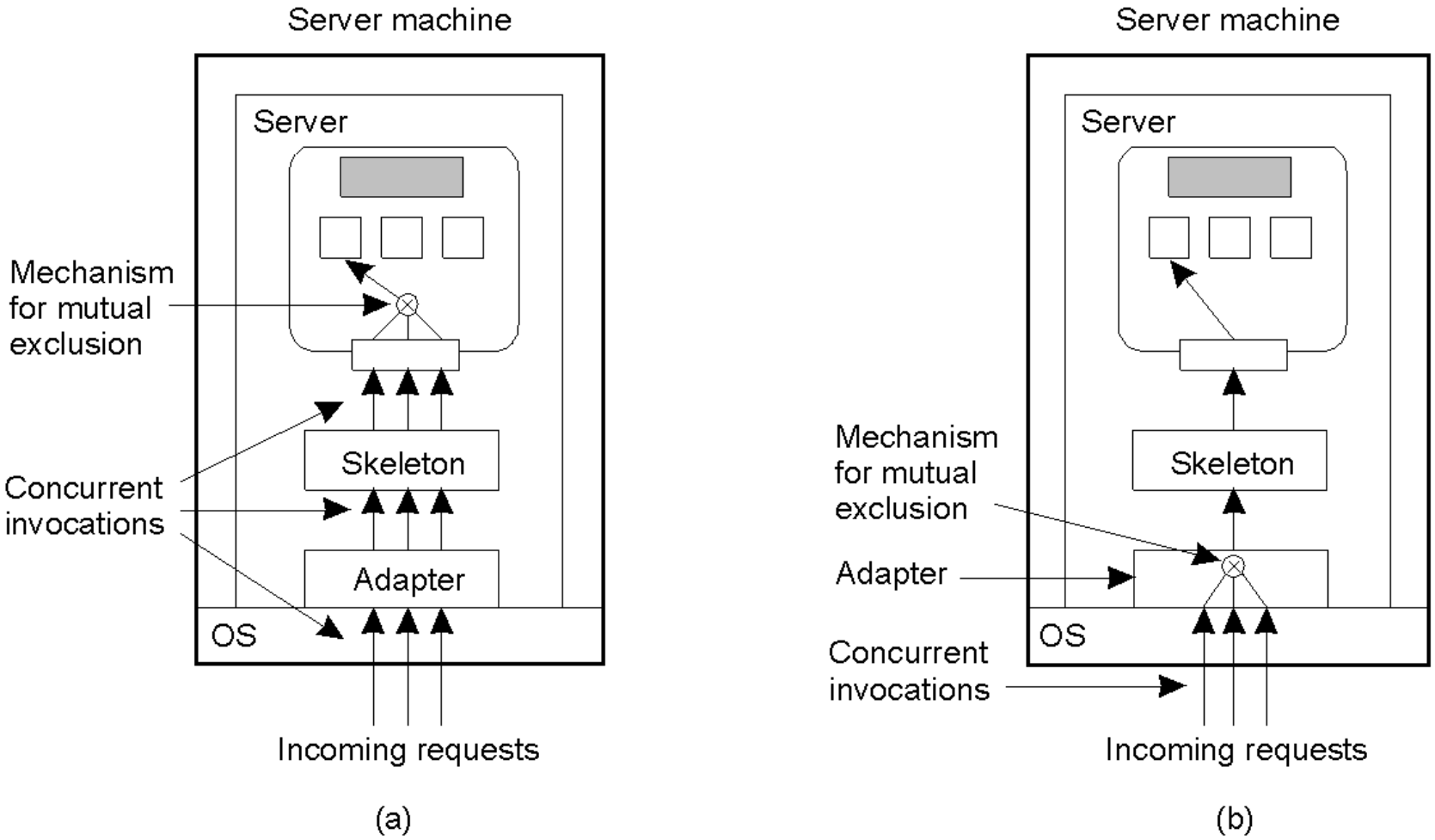
Chapter 6

Object Replication (1)



Organization of a distributed remote object shared by two different clients.

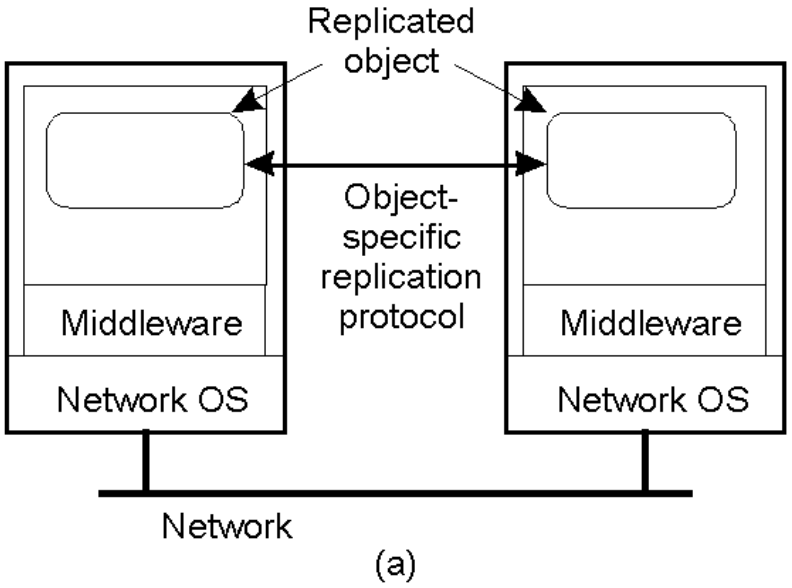
How to protect the object againts simultaneous access by multiple clients?

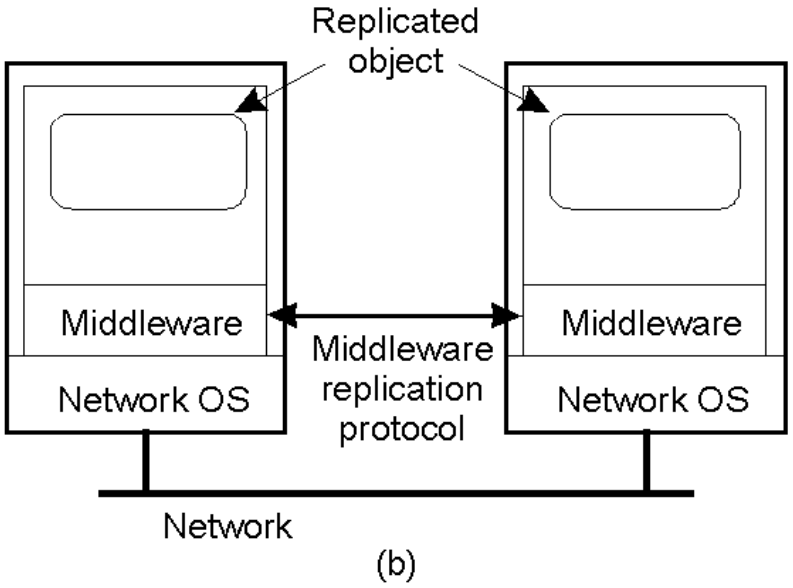


a) A remote object capable of handling concurrent invocations on its own.

b) A remote object for which an object adapter is required to handle concurrent invocations

Consistency problem when replicating a shared remote object without taking any special measures regarding the handling of concurrent invocation (Example : Synchronization problem was replicated bank-account database)

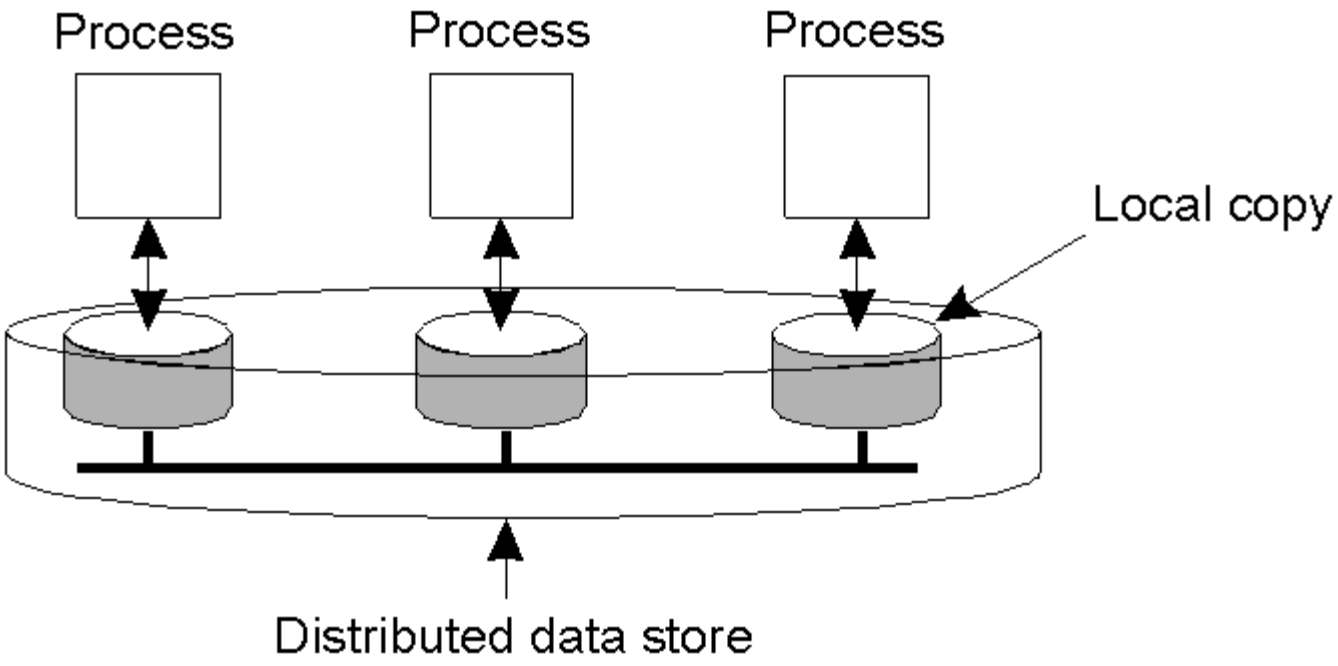




a) A distributed system for replication-aware distributed objects.

b) A distributed system responsible for replica management

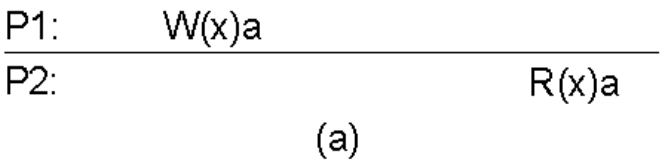
Data-Centric Consistency Models



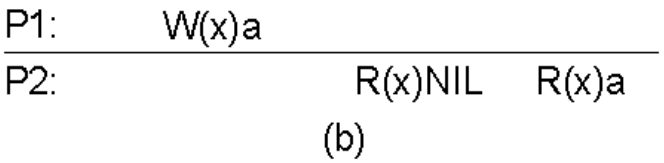
The general organization of a logical data store, physically distributed and replicated across multiple processes.

Strict Consistency

Any read on a data item x returns a value corresponding to the result of the most recent write on x



(a)



(b)

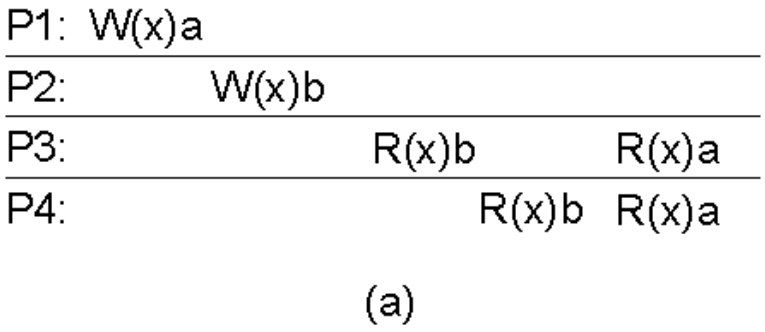
Behavior of two processes, operating on the same data item.

- A strictly consistent store.
- A store that is not strictly consistent.

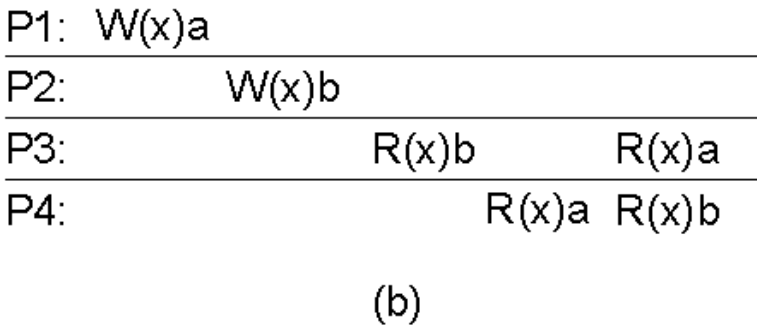
Linearizability and Sequential Consistency (1)

Sequential consistency : The Result of any execution is the same as if the (read & write) operation by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program.

Linearizable : the result of any execution is the same as if the (read & write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program. In addition, if $ts_{OP1}(x) < ts_{OP2}(y)$, then operation OP1(x) should precede OP2(y) in this sequence.



(a)



(b)

a) A sequentially consistent data store.

b) A data store that is not sequentially consistent.

Linearizability and Sequential Consistency (2)

Process P1	**Process P2**	**Process P3**
x = 1;	y = 1;	z = 1;
print (y, z);	print (x, z);	print (x, y);

Three concurrently executing processes.

Linearizability and Sequential Consistency (3)

x = 1;	x = 1;	y = 1;	y = 1;
print ((y, z);	y = 1;	z = 1;	x = 1;
y = 1;	print (x,z);	print (x, y);	z = 1;
print (x, z);	print(y, z);	print (x, z);	print (x, z);
z = 1;	z = 1;	x = 1;	print (y, z);
print (x, y);	print (x, y);	print (y, z);	print (x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011 (a)	Signature: 101011 (b)	Signature: 110101 (c)	Signature: 111111 (d)

Four valid execution sequences for the processes of the previous slide. The vertical axis is time.

Casual Consistency (1)

Necessary condition:

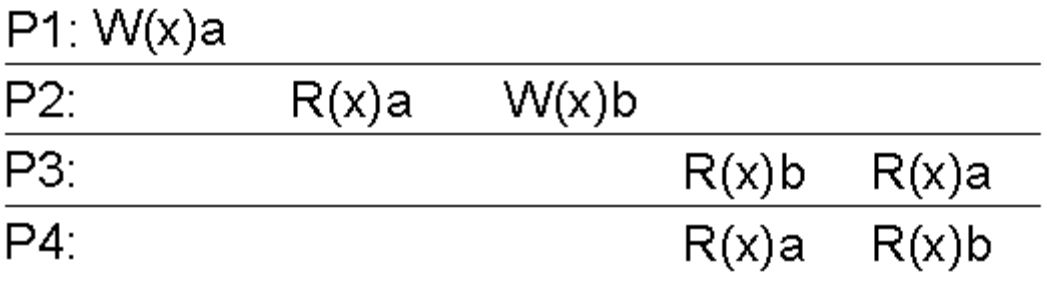
Writes that are potentially casually related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.

Casual Consistency (2)

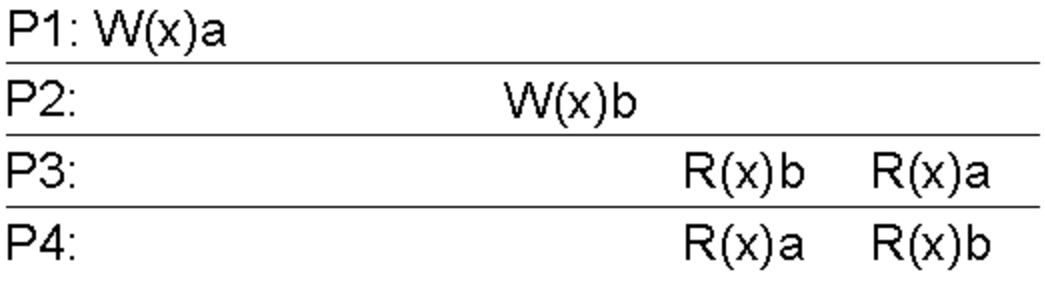
P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

This sequence is allowed with a casually-consistent store, but not with sequentially or strictly consistent store.

Casual Consistency (3)



(a)



(b)

a) A violation of a casually-consistent store.

b) A correct sequence of events in a casually-consistent store.

FIFO Consistency (1)

Necessary Condition:

Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.

FIFO Consistency (2)

P1: W(x)a						
P2:	R(x)a	W(x)b	W(x)c			
P3:				R(x)b	R(x)a	R(x)c
P4:				R(x)a	R(x)b	R(x)c

A valid sequence of events of FIFO consistency

FIFO Consistency (3)

x = 1;	x = 1;	y = 1;
print (y, z);	y = 1;	print (x, z);
y = 1;	**print(x, z);**	z = 1;
print(x, z);	print (y, z);	**print (x, y);**
z = 1;	z = 1;	x = 1;
print (x, y);	print (x, y);	print (y, z);
Prints: 00	Prints: 10	Prints: 01
(a)	(b)	(c)

Statement execution as seen by the three processes from the previous slide. The statements in bold are the ones that generate the output shown.

FIFO Consistency (4)

Process P1	**Process P2**
x = 1;	y = 1;
if (y == 0) kill (P2);	if (x == 0) kill (P1);

Two concurrent processes.

Weak Consistency (1)

Properties:

- Accesses to synchronization variables associated with a data store are sequentially consistent
- No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere
- No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.

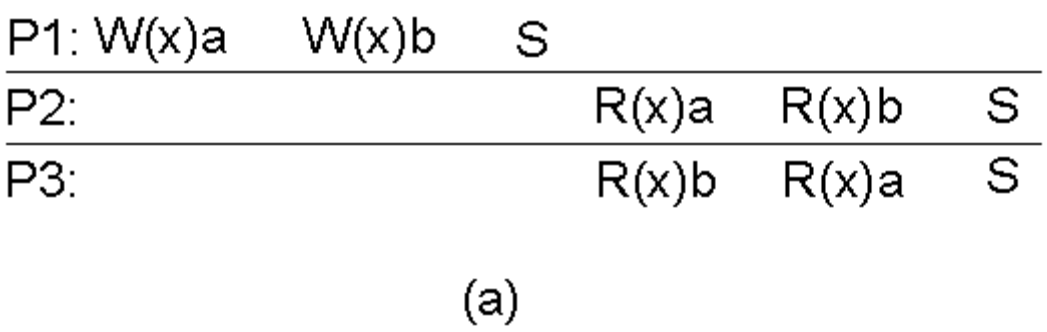
Weak Consistency (2)

```
int a, b, c, d, e, x, y;          /* variables */
int *p, *q;                       /* pointers */
int f( int *p, int *q);           /* function prototype */

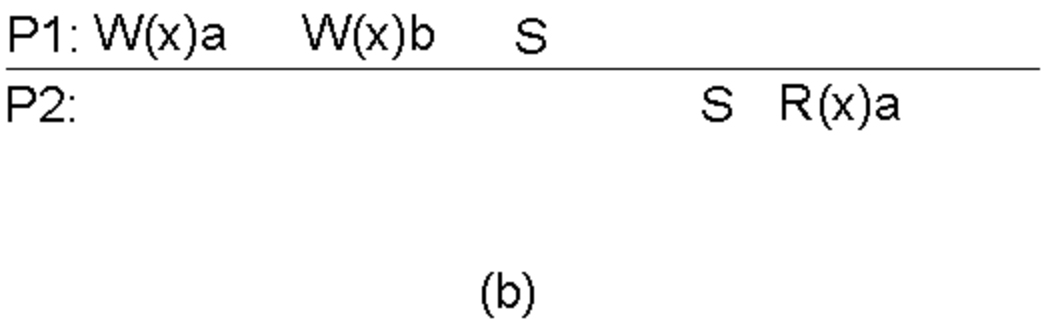
a = x * x;                        /* a stored in register */
b = y * y;                        /* b as well */
c = a*a*a + b*b + a * b;          /* used later */
d = a * a * c;                    /* used later */
p = &a;                           /* p gets address of a */
q = &b                            /* q gets address of b */
e = f(p, q)                       /* function call */
```

A program fragment in which some variables may be kept in registers.

Weak Consistency (3)



(a)

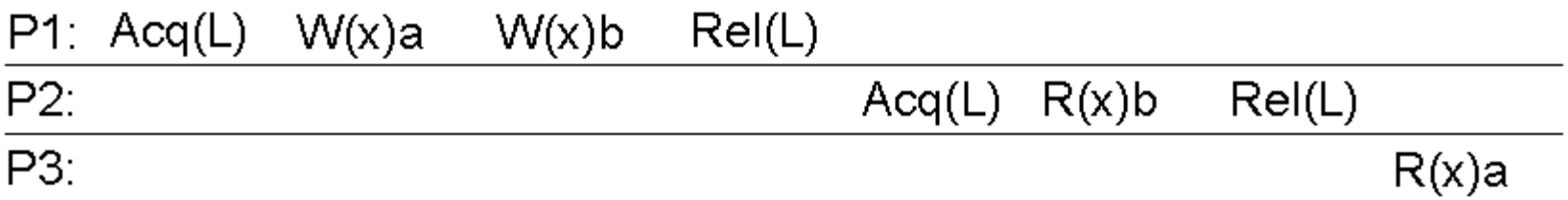


(b)

a) A valid sequence of events for weak consistency.

b) An invalid sequence for weak consistency.

Release Consistency (1)



A valid event sequence for release consistency.

Release Consistency (2)

Rules:

- Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully.
- Before a release is allowed to be performed, all previous reads and writes by the process must have completed
- Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).

Entry Consistency (1)

Conditions:

- An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process.
- Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode.
- After an exclusive mode access to a synchronization variable has been performed, any other process's next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.

Entry Consistency (1)

P1:	Acq(Lx)	W(x)a	Acq(Ly)	W(y)b	Rel(Lx)	Rel(Ly)						
P2:							Acq(Lx)	R(x)a				R(y)NIL
P3:									Acq(Ly)	R(y)b		

A valid event sequence for entry consistency.

Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.
FIFO	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order

(a)

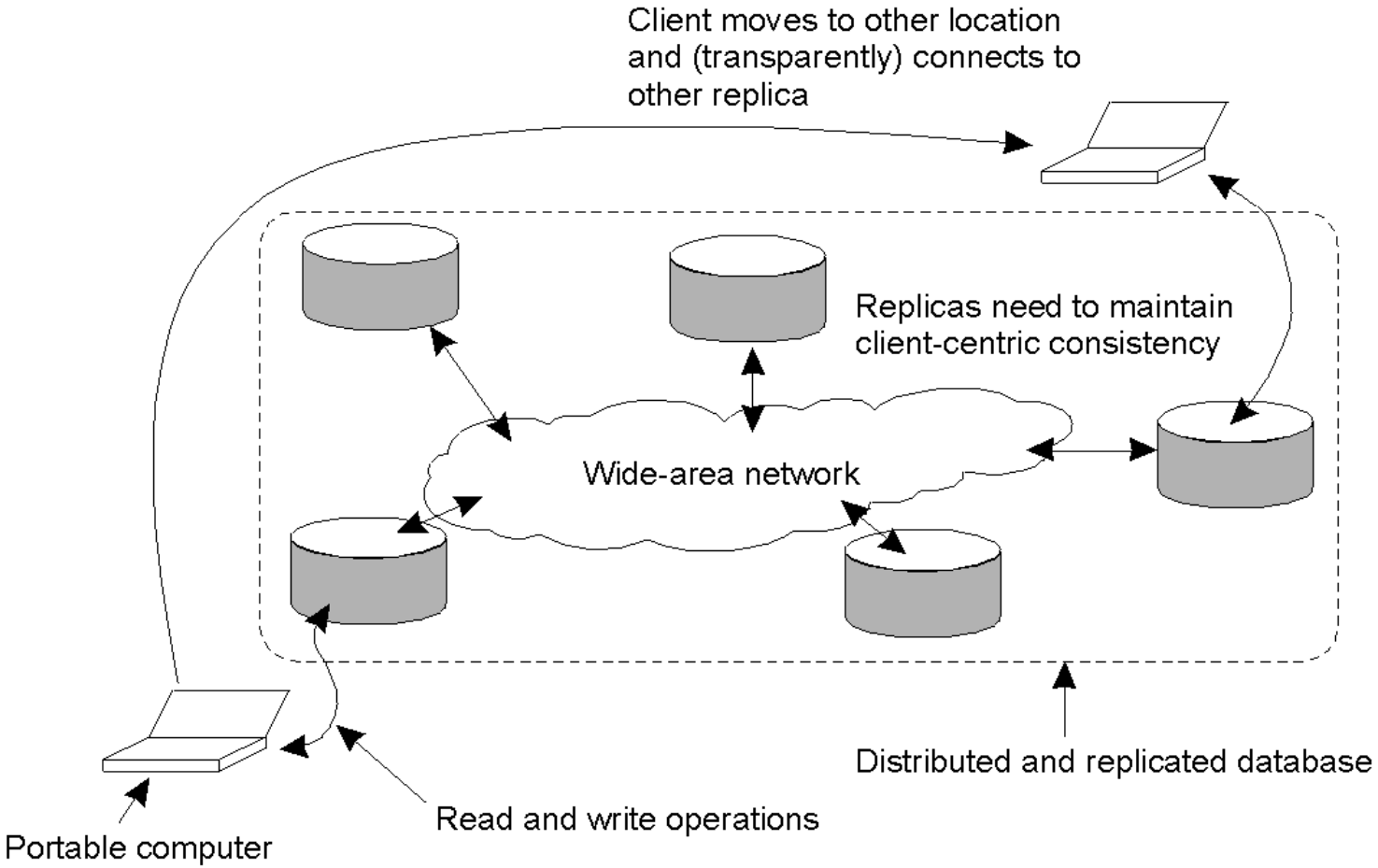
Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b)

a) Consistency models not using synchronization operations.

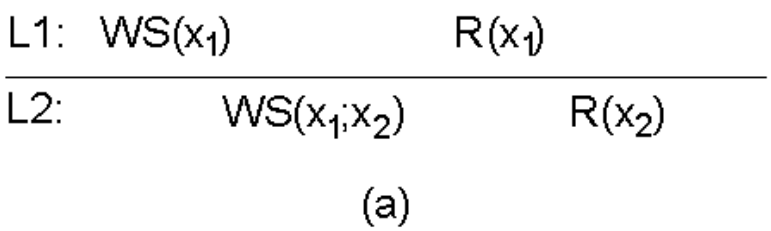
b) Models with synchronization operations.

Client-Centric Eventual Consistency

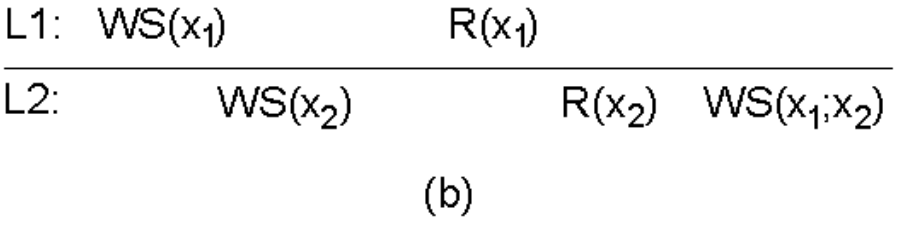


The principle of a mobile user accessing different replicas of a distributed database.

Monotonic Reads



(a)



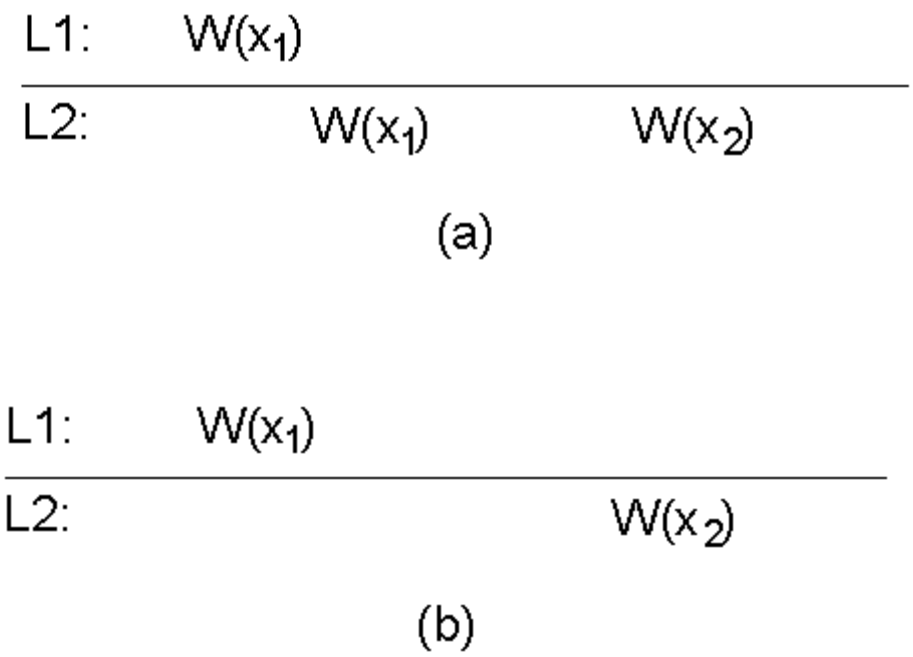
(b)

The read operations performed by a single process *P* at two different local copies of the same data store.

a) A monotonic-read consistent data store

b) A data store that does not provide monotonic reads.

Monotonic Writes

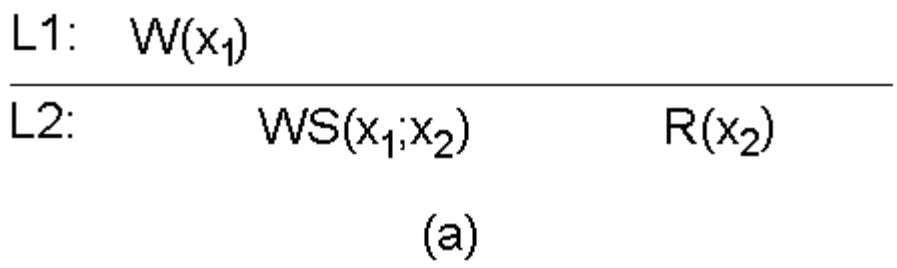


The write operations performed by a single process *P* at two different local copies of the same data store

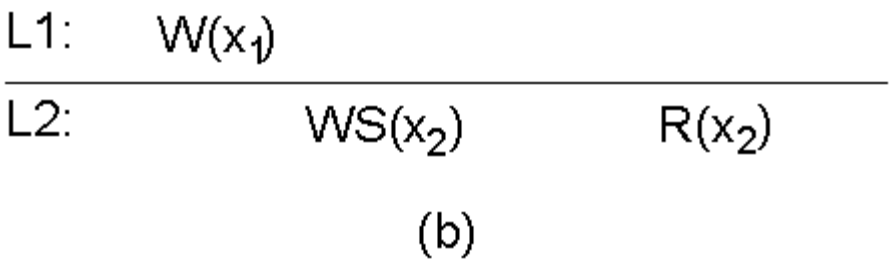
a) A monotonic-write consistent data store.

b) A data store that does not provide monotonic-write consistency.

Read Your Writes



(a)

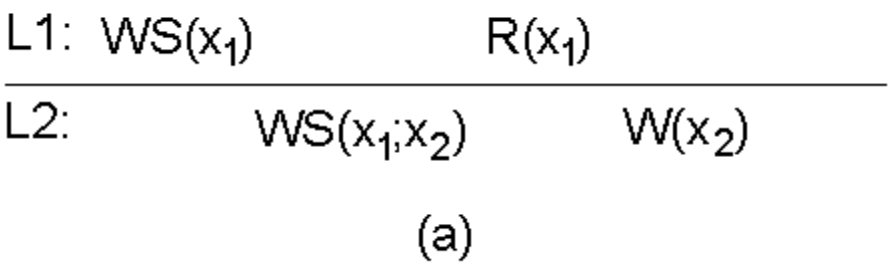


(b)

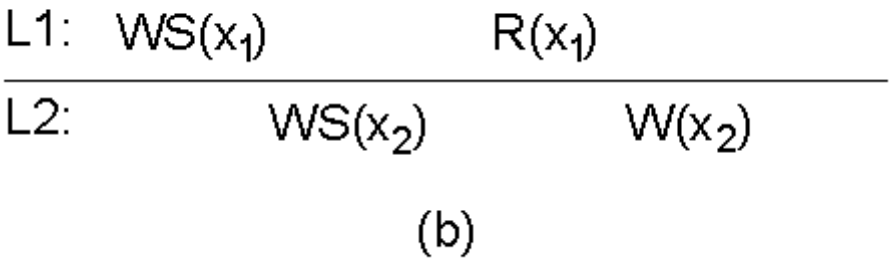
a) A data store that provides read-your-writes consistency.

b) A data store that does not.

Writes Follow Reads



(a)

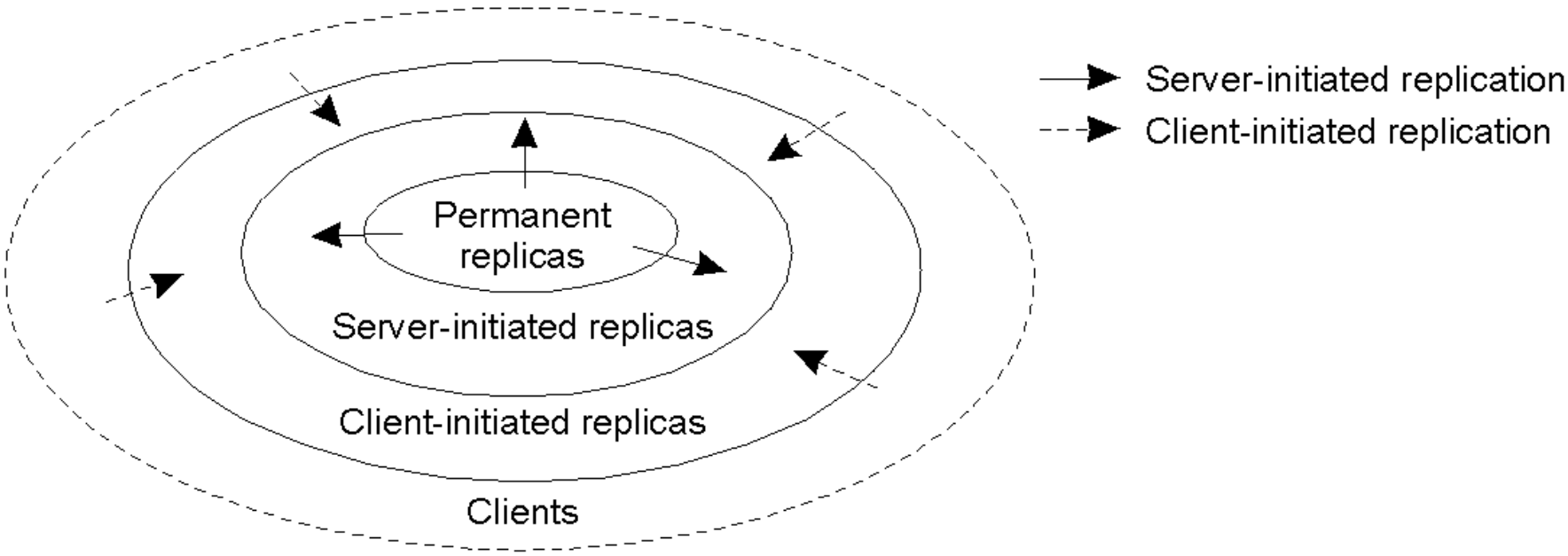


(b)

a) A writes-follow-reads consistent data store

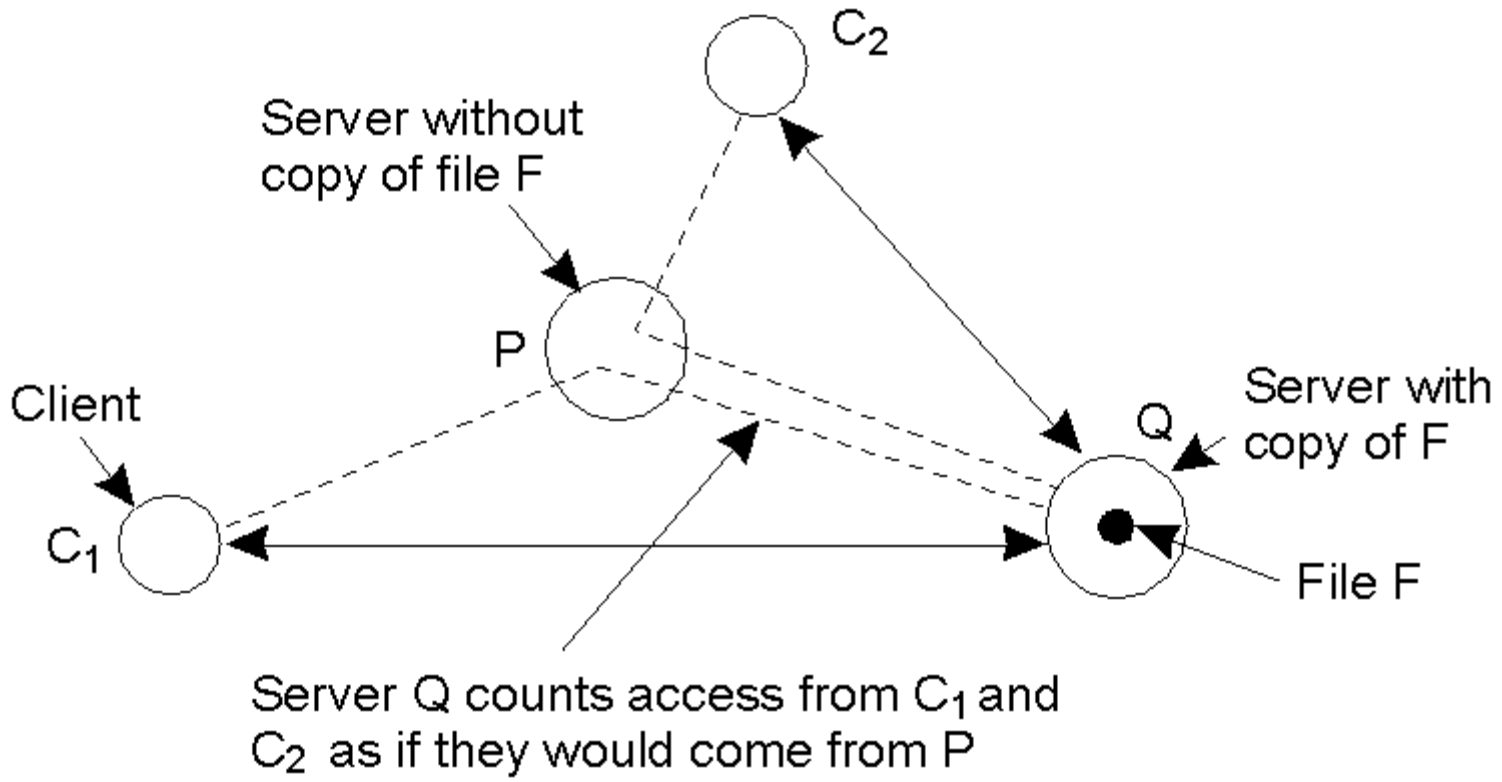
b) A data store that does not provide writes-follow-reads consistency

Replica Placement



The logical organization of different kinds of copies of a data store into three concentric rings.

Server-Initiated Replicas



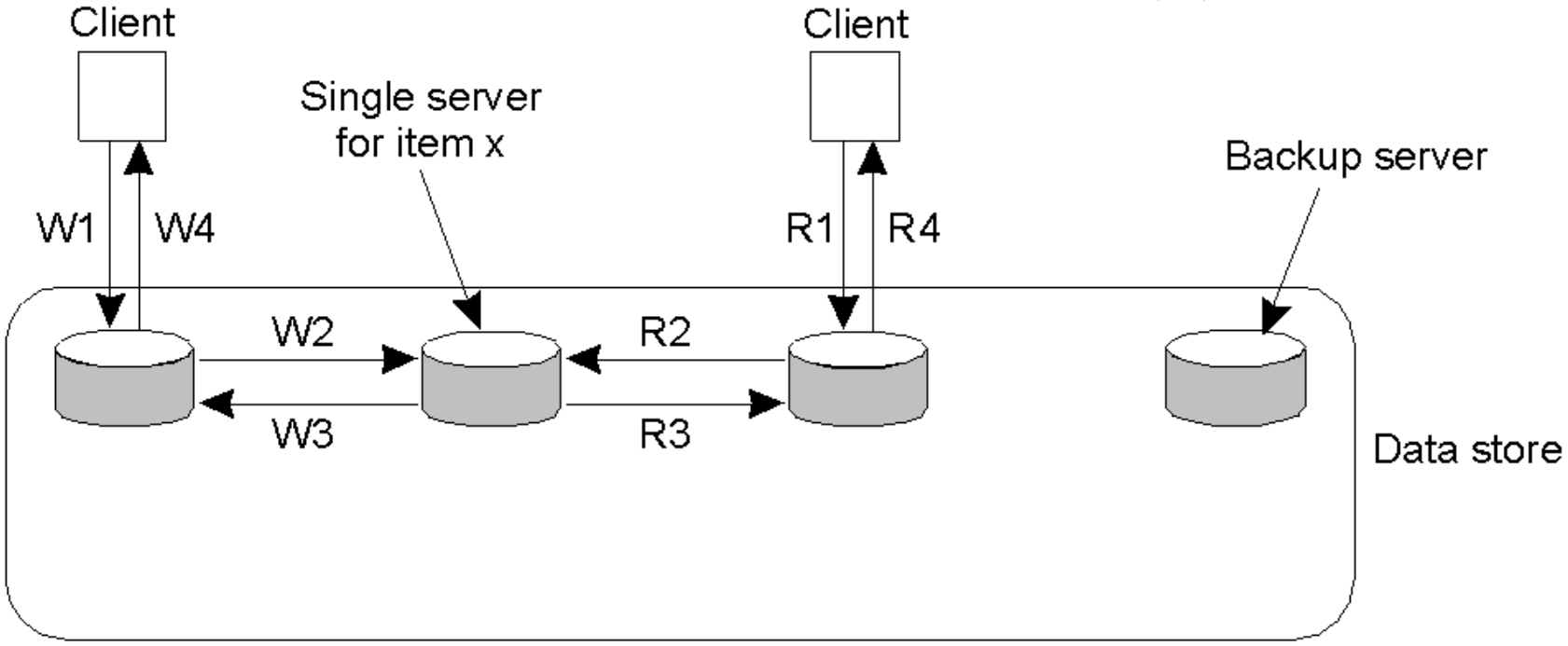
Counting access requests from different clients.

Update Propagation
Pull versus Push Protocols

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

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

Consistency Protocol
Primary Based Protocol
Remote-Write Protocols (1)

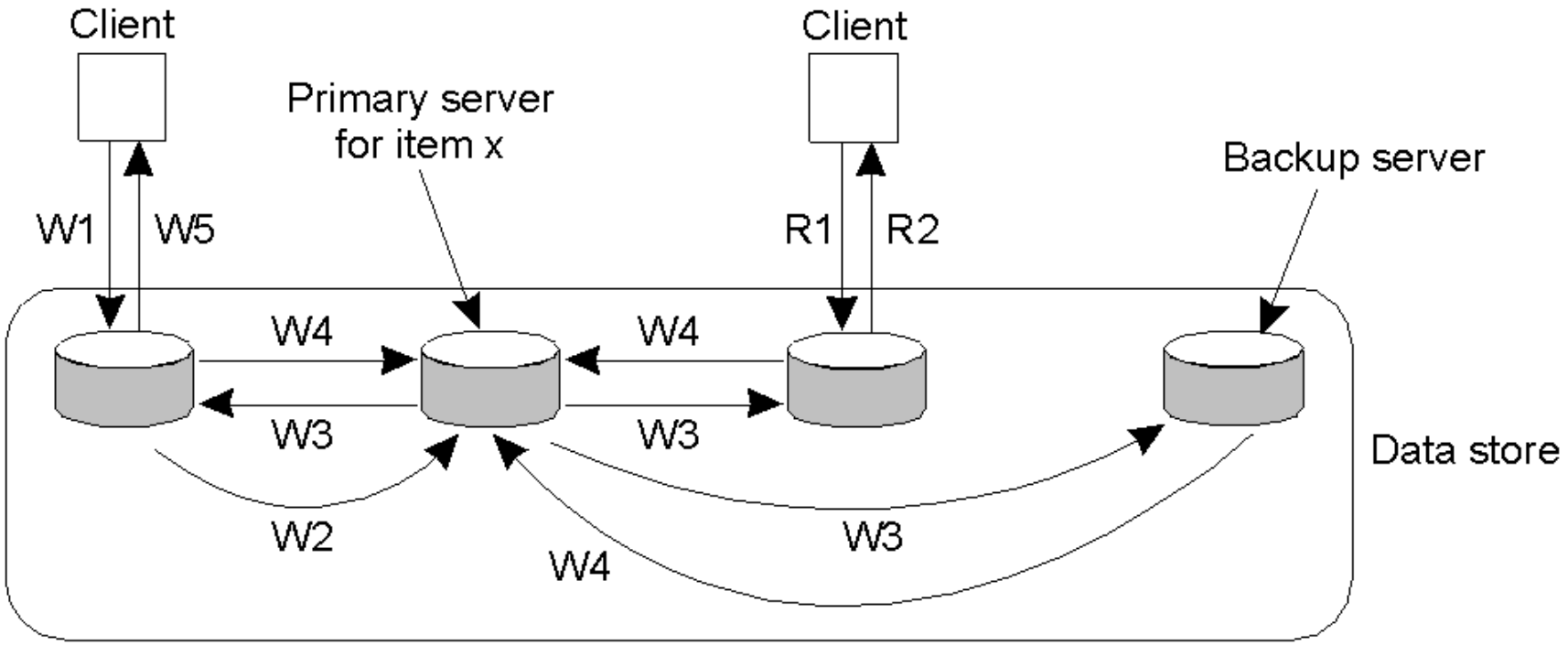


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

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

Primary-based remote-write protocol with a fixed server to which all read and write operations are forwarded.

Remote-Write Protocols (2)

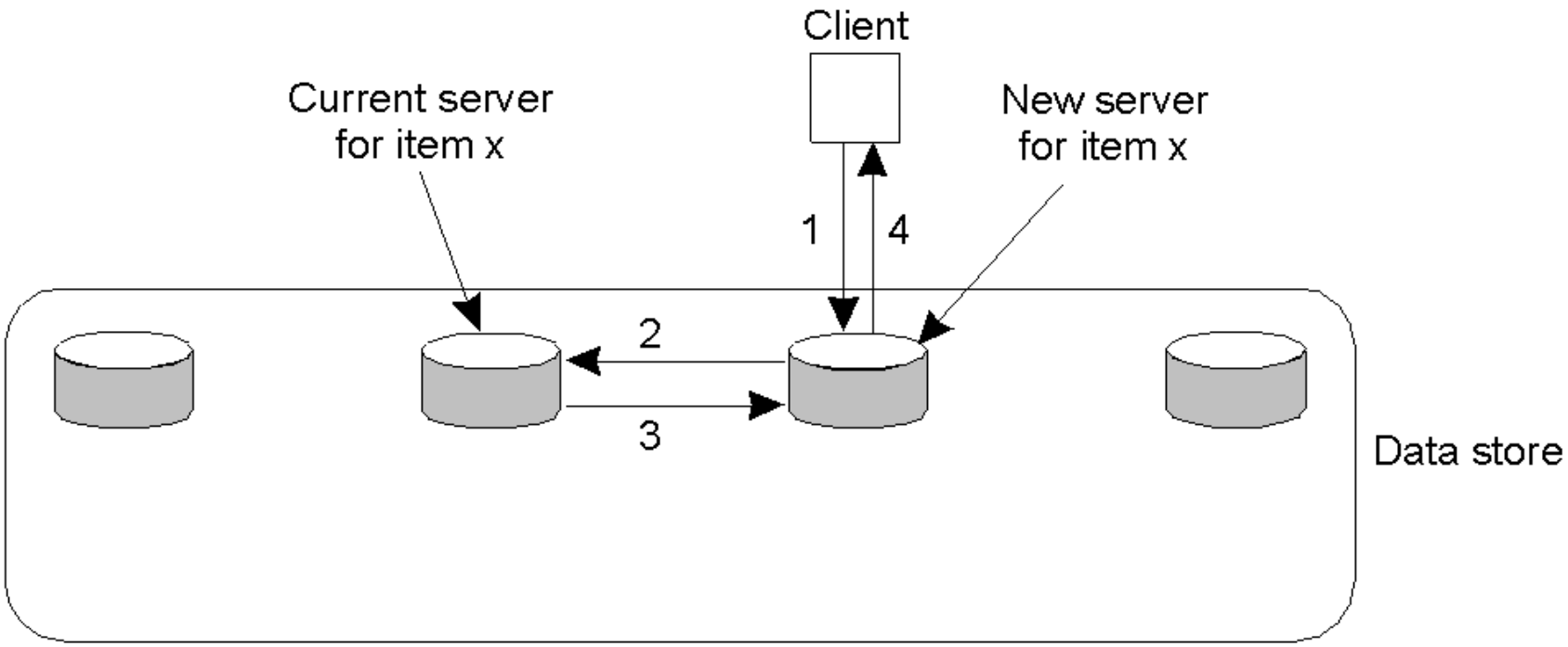


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

The principle of primary-backup protocol.

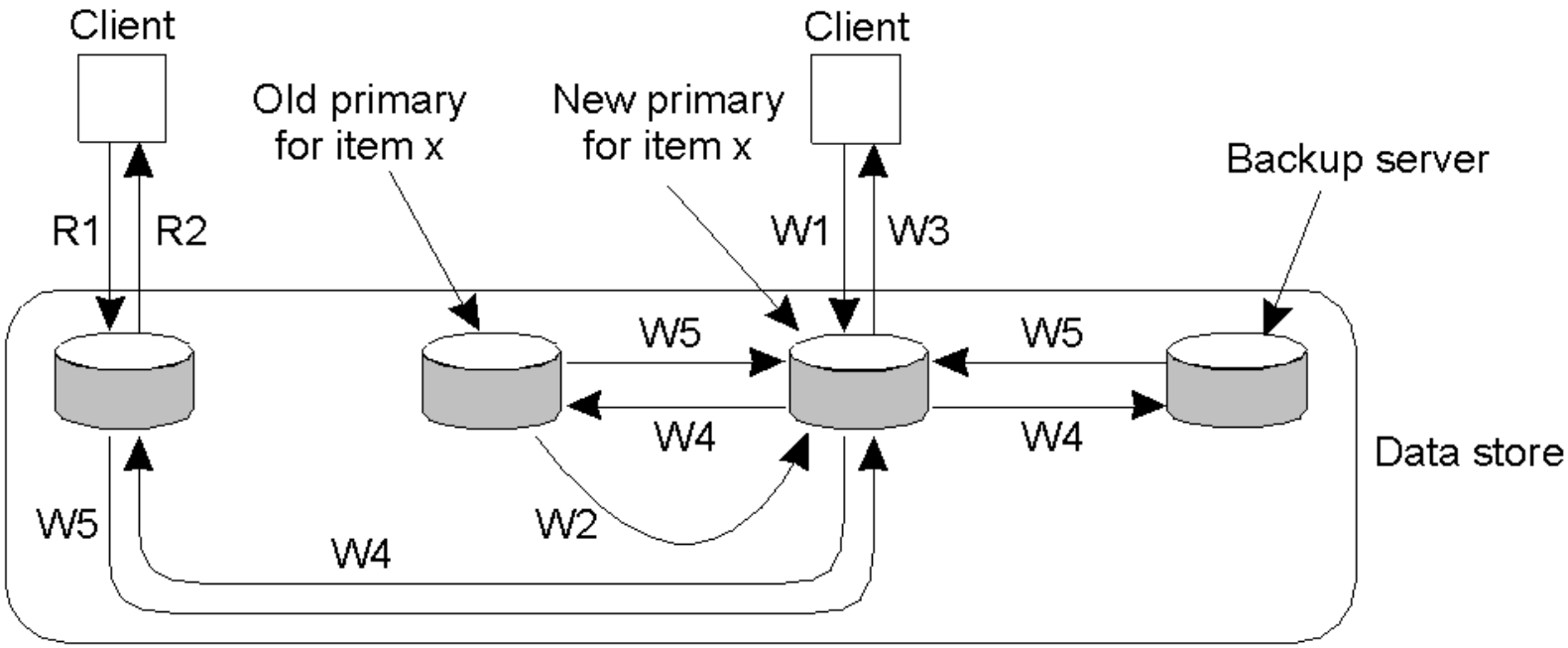
Local-Write Protocols (1)



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.

Local-Write Protocols (2)

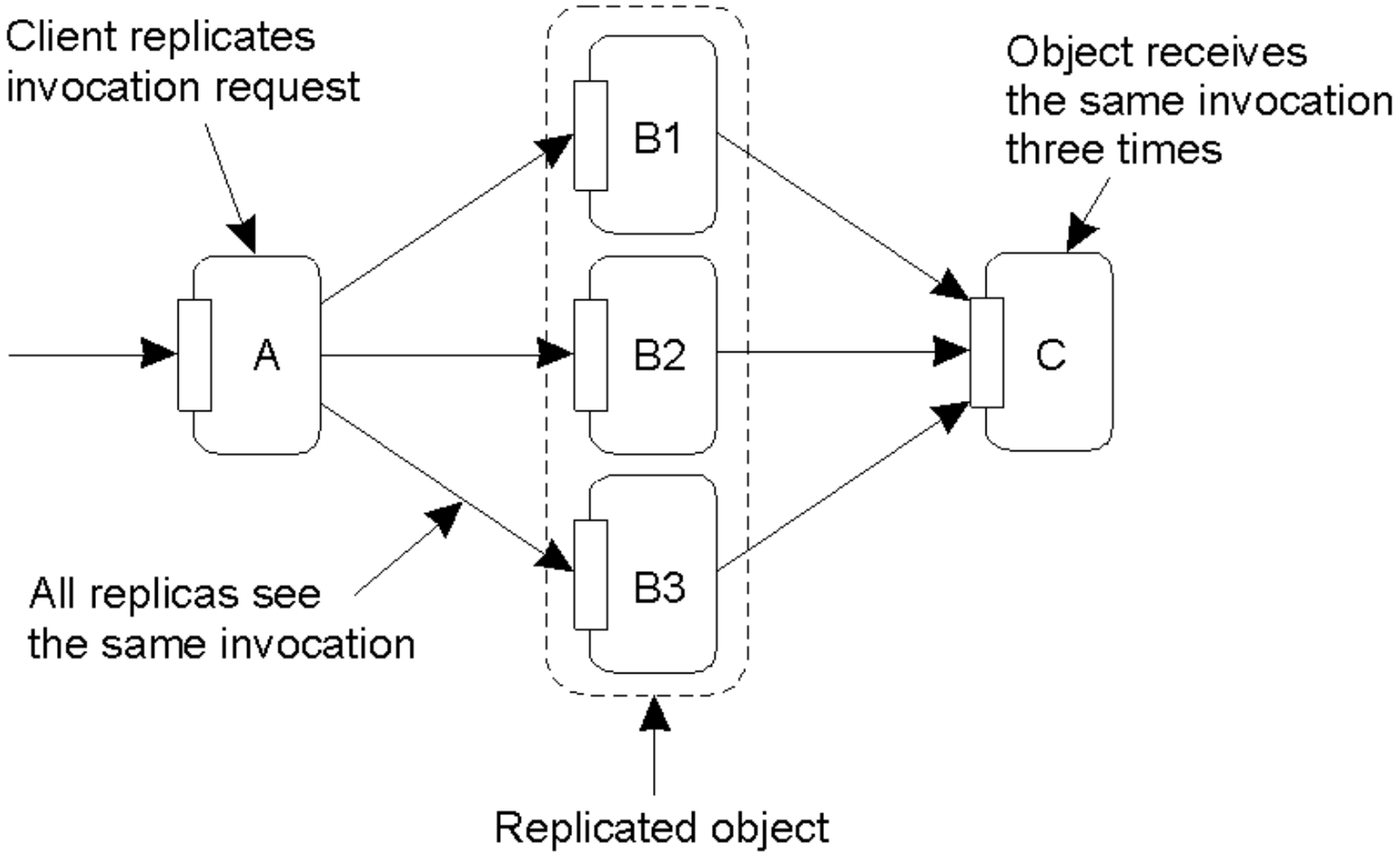


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

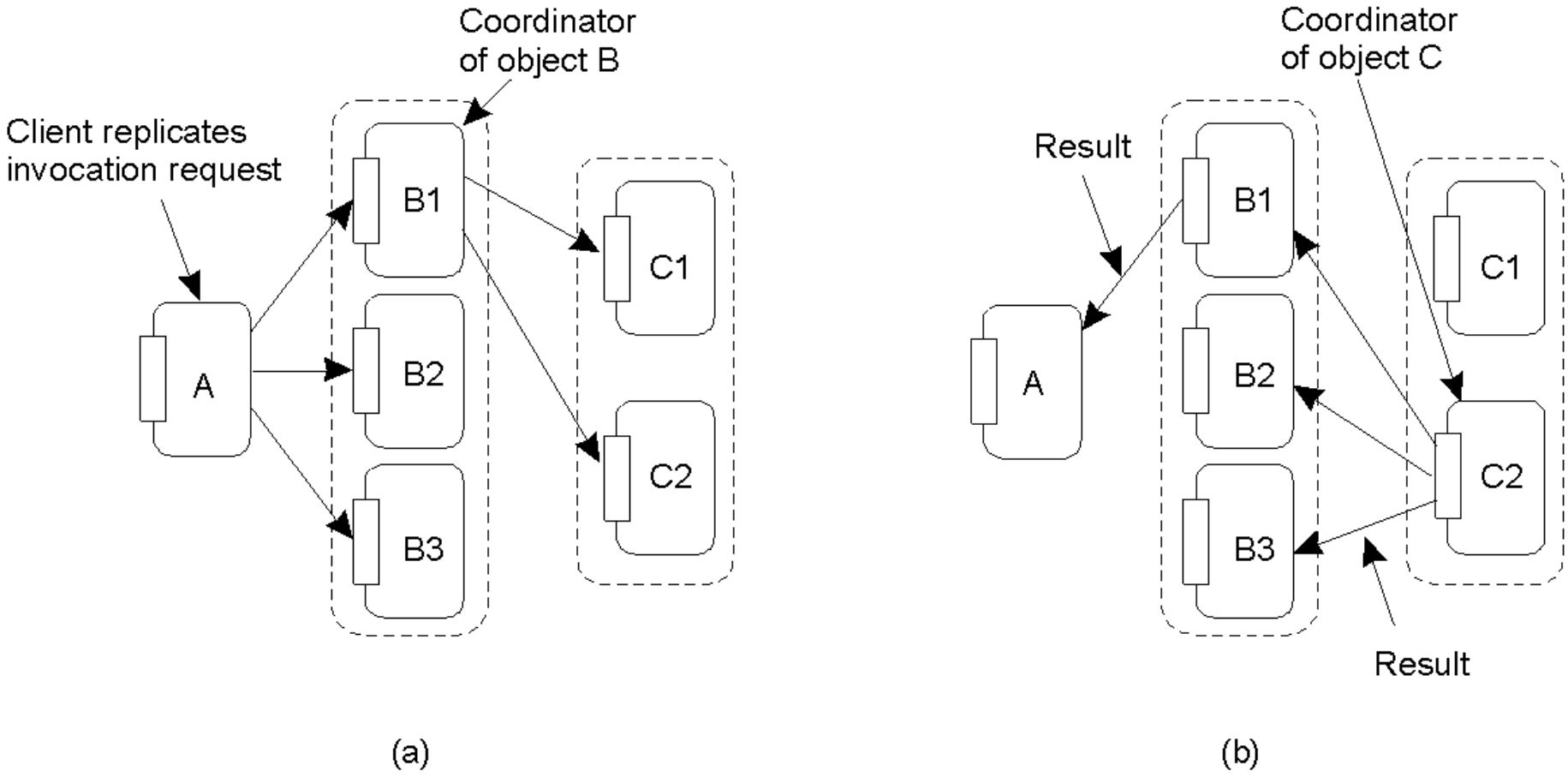
Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

Replicated-Write Protocols
Active Replication (1)



The problem of replicated invocations.

Active Replication (2)



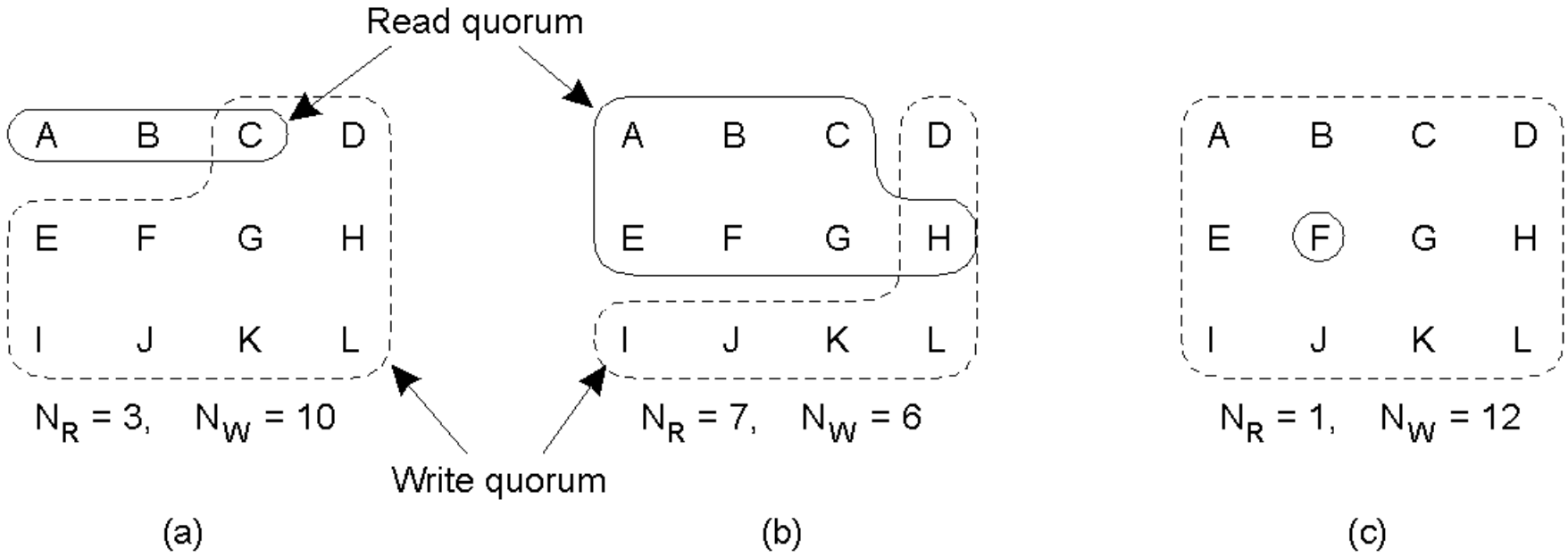
a) Forwarding an invocation request from a replicated object.

b) Returning a reply to a replicated object.

Quorum-Based Protocols

Two Constraint : 1. $N_R + N_W > N$

2. $N_W > N / 2$



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)

Orca

Orca is a Programming system thar allows processes on different machines to have controlled access to a distributed shared memory consisting of protected objects

Thre Orca Systems consists of the Orca Language, compiler, and runtime system that actually manage the shared objects during execution.

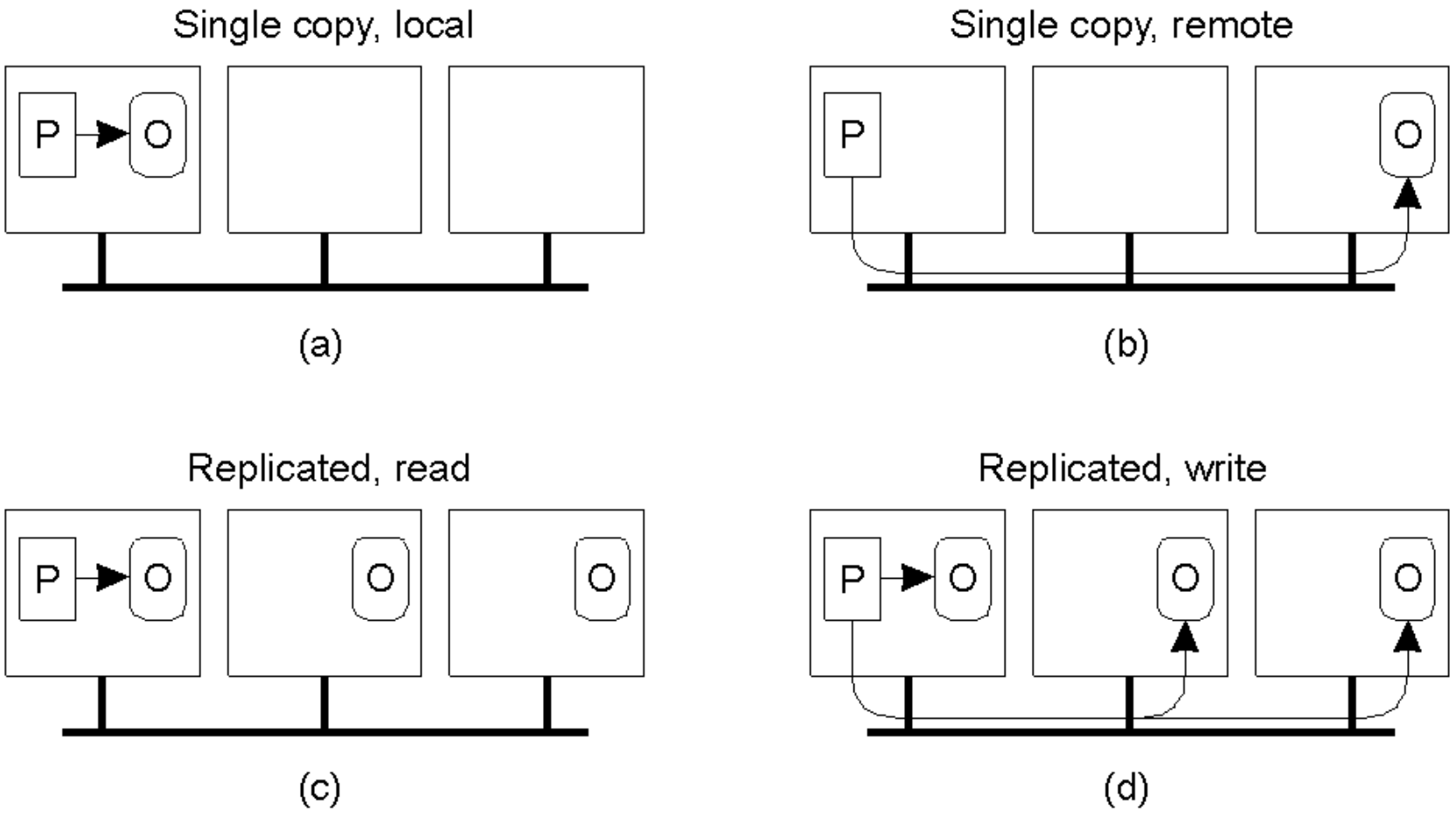
```
OBJECT IMPLEMENTATION stack;
  top: integer;                                  # variable indicating the top
  stack: ARRAY[integer 0..N-1] OF integer        # storage for the stack

  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;
```

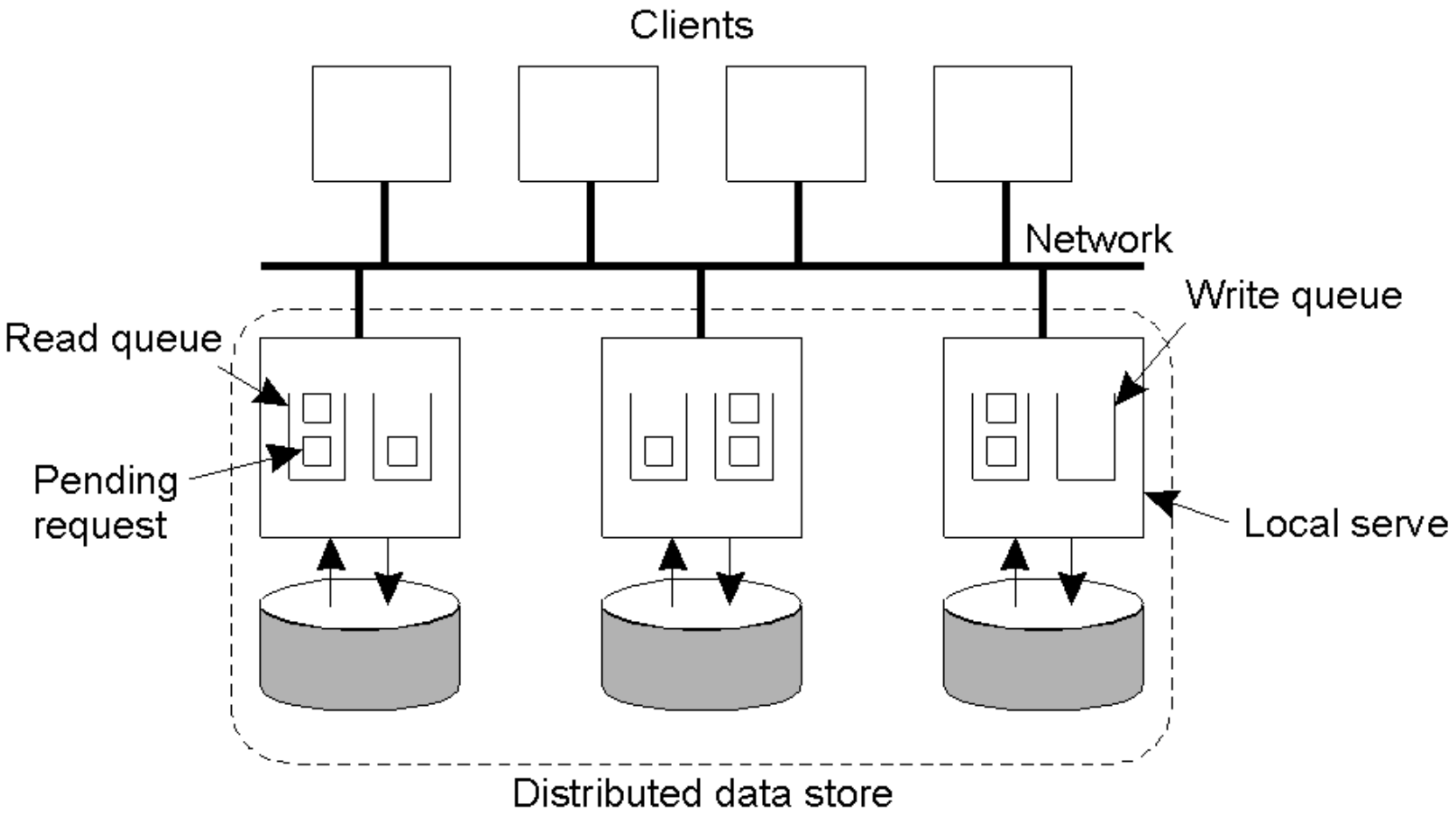
A simplified stack object in Orca, with internal data and two operations.

Management of Shared Objects in Orca



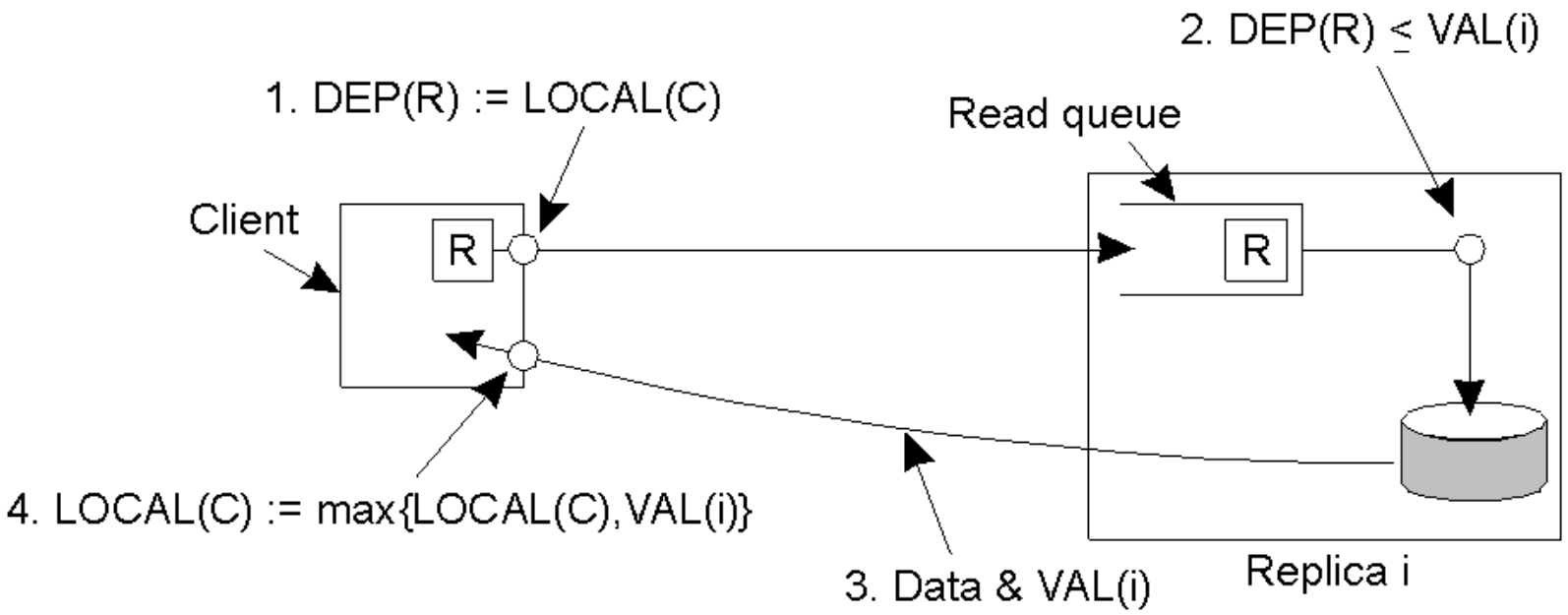
Four cases of a process *P* performing an operation on an object *O* in Orca.

Casually-Consistent Lazy Replication



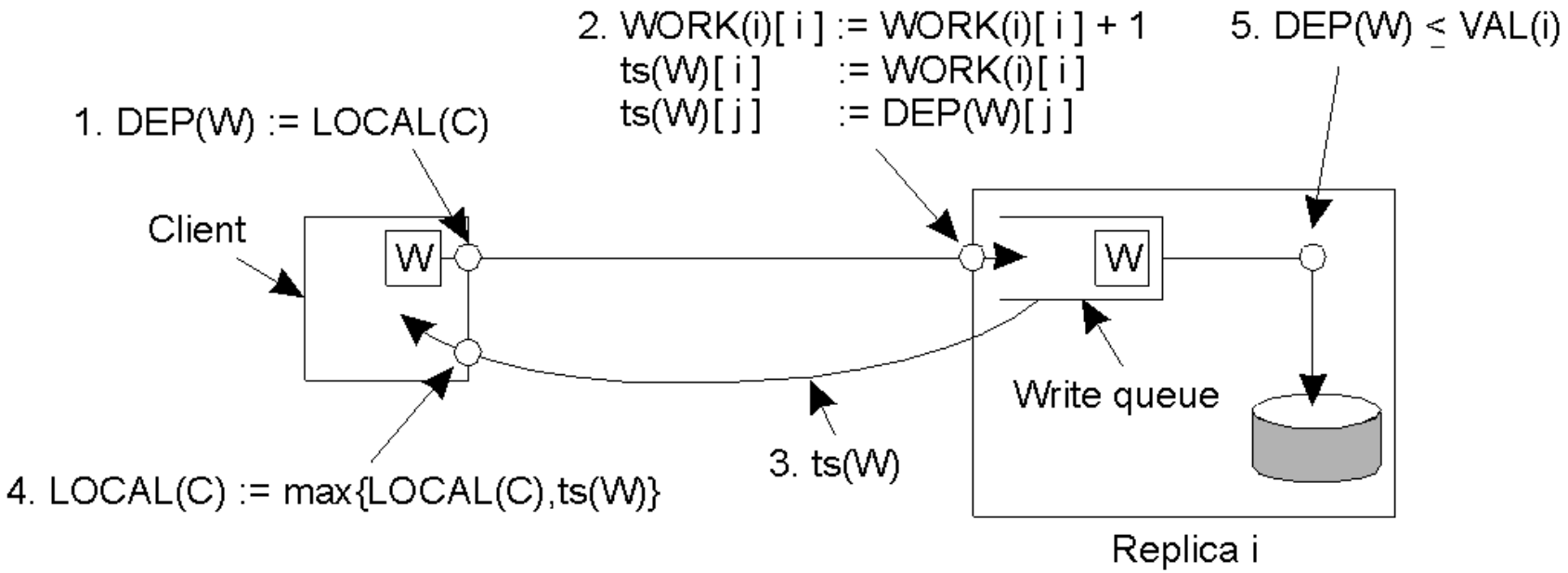
The general organization of a distributed data store. Clients are assumed to also handle consistency-related communication.

Processing Read Operations



Performing a read operation at a local copy.

Processing Write Operations



Performing a write operation at a local copy.