Replication and Consistency

Chapter 6

Replication

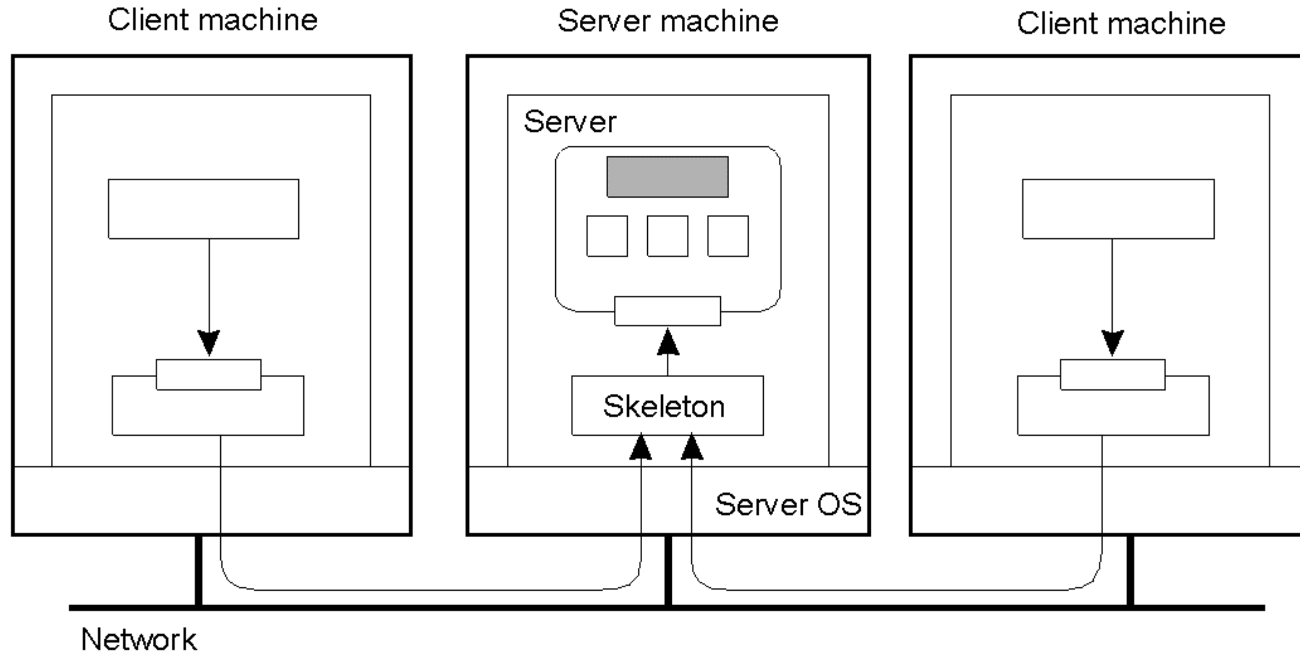
Data **Replication** is used for

- improving performance
- enhancing reliability

But data replicas must be kept **consistent.**

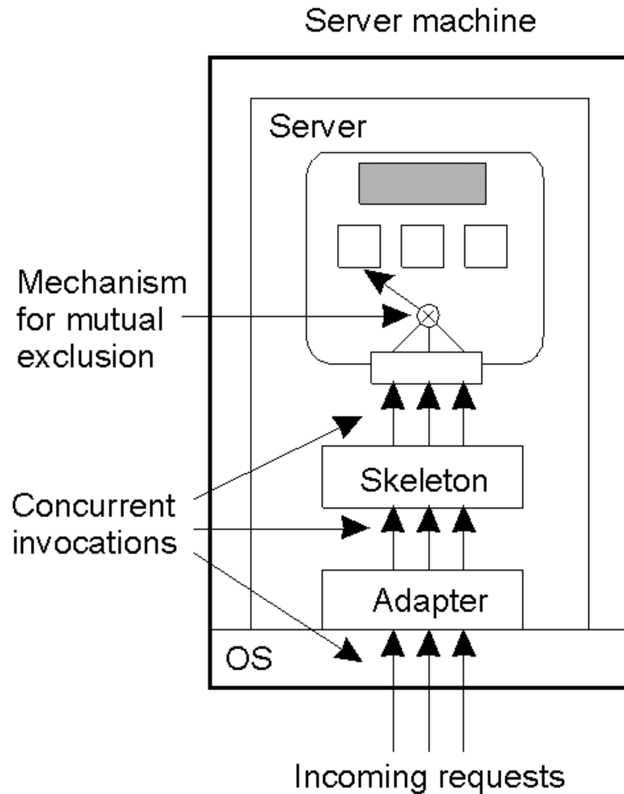
Efficient consistency models are hard to be implemented.

Object Replication (1)

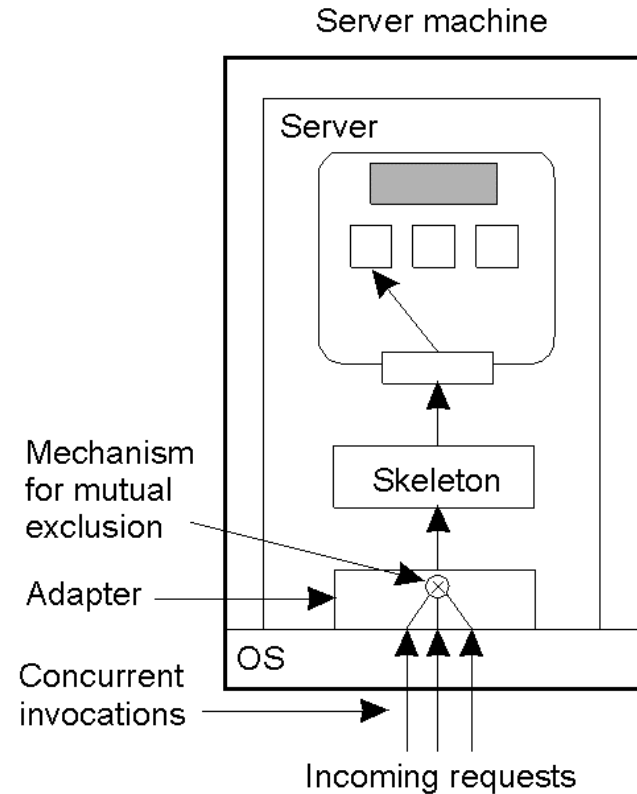


- Organization of a distributed remote object shared by two different clients.
- How access to a shared object ?

Object Replication (2)



(a)

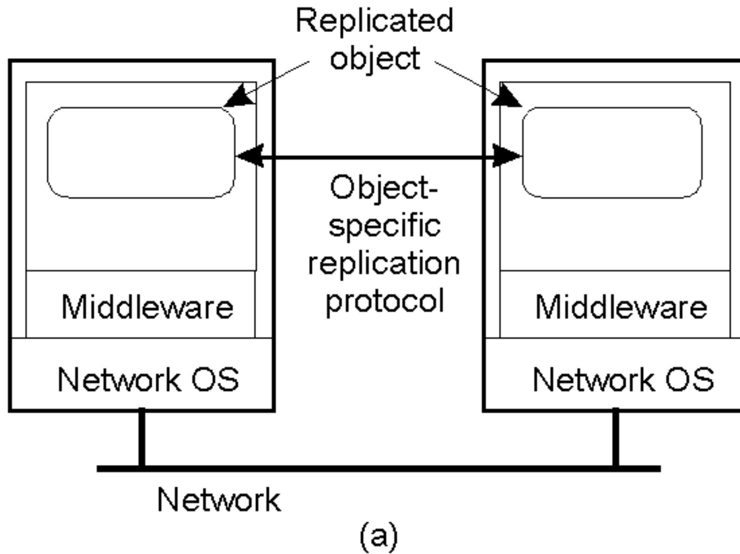


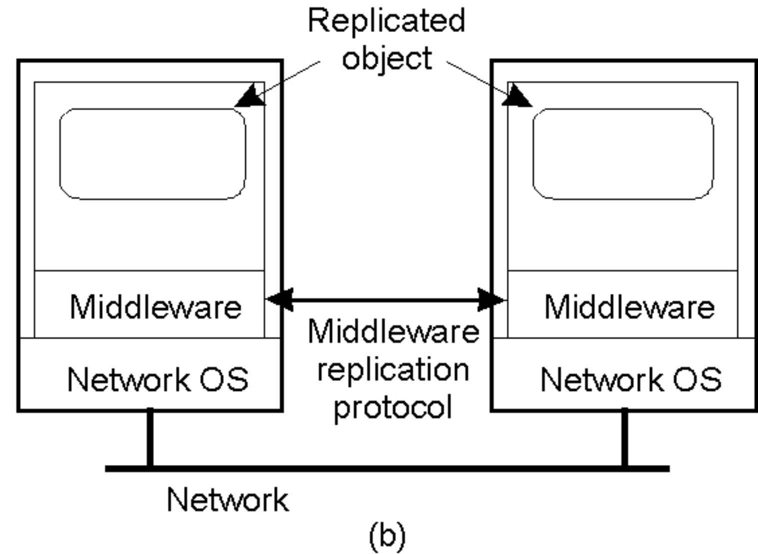
(b)

(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

Object Replication (3)





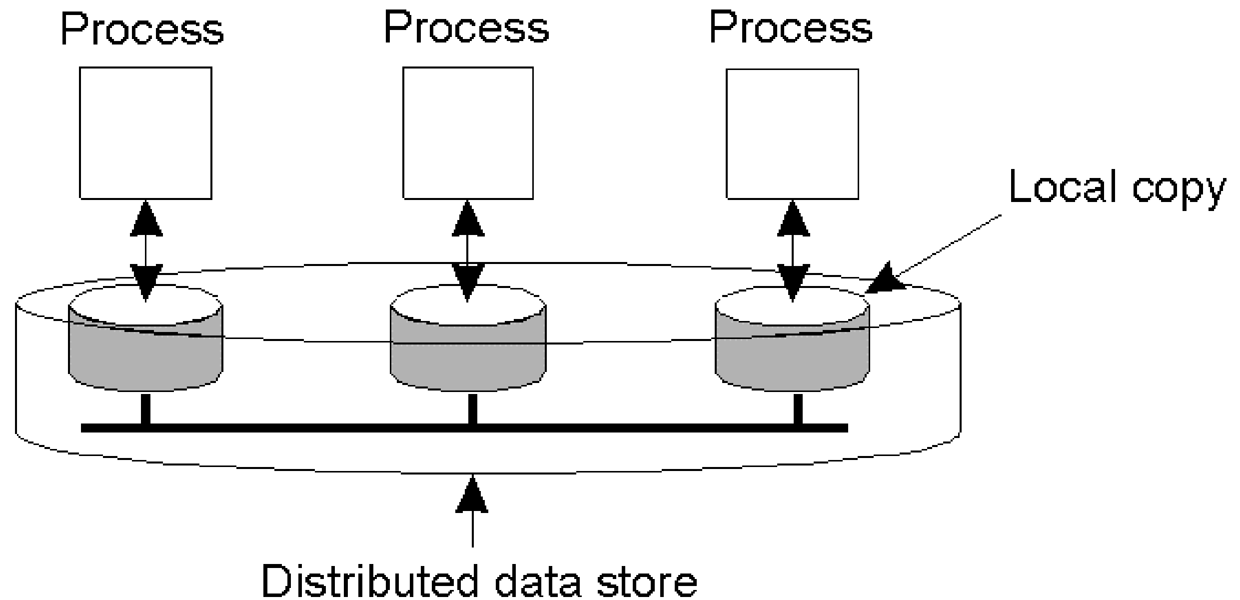
How to manage replics synchronizations ?

(a) A distributed system for replication-aware distributed objects (*Globe*, *SOS*).

(b) A distributed system responsible for replica management (*Piranha*).

Data-Centric Consistency Models

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



A **consistency model** is a contract between processes and the data store: if processes obeys certain rules the store contains correct values.

Strict Consistency

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

Global time is needed.

P1:	W(x)a		
P2:			R(x)a

(a)

P1:	W(x)a		
P2:		R(x)NIL	R(x)a

(b)

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

(a) A strictly consistent store.

(b) A store that is not strictly consistent.

In S.C. writes are instantaneously visible to all processes.

Sequential Consistency

The result of any execution is the same as if the 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.

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

(a)

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

(b)

(a) A sequentially consistent data store.

(b) A data store that is not sequentially consistent.

Linearizability (1)

The result of any execution is the same as if the 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 this program. In addition, if $ts_{OP1}(x) < ts_{OP2}(y)$*, then operation OP1(x) should precede OP2(y) in this sequence.*

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 (2)

(a)

```
x = 1;
print ((y, z);
y = 1;
print (x, z);
z = 1;
print (x, y);
```

Prints: 001011

Signature: 001011

(b)

```
x = 1;
y = 1;
print (x,z);
print(y, z);
z = 1;
print (x, y);
```

Prints: 101011

Signature: 101011

(c)

```
y = 1;
z = 1;
print (x, y);
print (x, z);
x = 1;
print (y, z);
```

Prints: 010111

Signature: 110101

(d)

```
y = 1;
x = 1;
z = 1;
print (x, z);
print (y, z);
print (x, y);
```

Prints: 111111

Signature: 111111

Four valid execution sequences for the processes.

Causal Consistency (1)

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

If two processes simultaneously write two variables are not potentially causally related (concurrent writes).

A read followed later by a write can be potentially causally related.

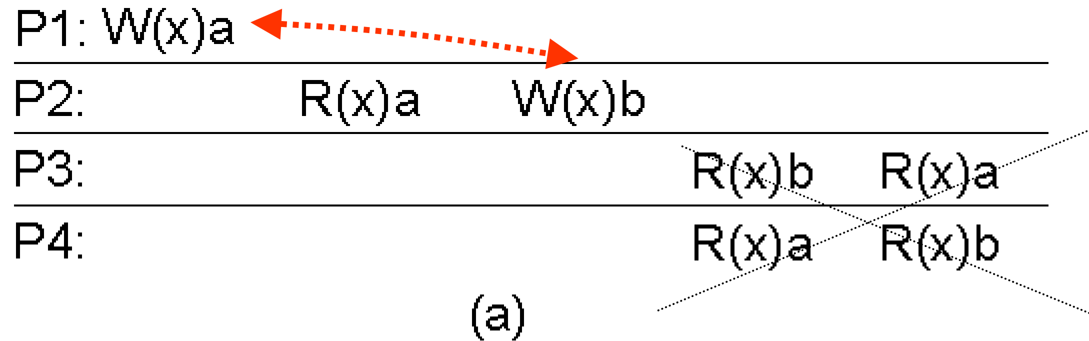
Causal Consistency (2)

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

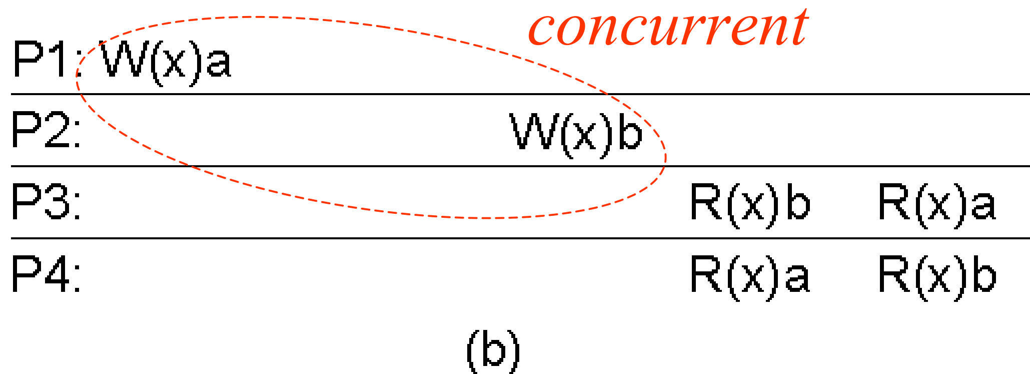
concurrent

This sequence is allowed with a causally-consistent store, but not with sequentially or strictly consistent store: ***concurrent writes can be seen in a different order***.

Causal Consistency (3)



(a) A violation of a casually-consistent store.



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

FIFO Consistency (1)

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.

Two or more writes from the same process must be seen in order. Writes from different processes can be seen in different order.

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 is easy to be implemented.

FIFO Consistency (3)

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

(a)	(b)	(c)
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

Statement execution as seen by the three processes.

The statements in bold are the ones that generate the output shown.

FIFO Consistency (4)

x and y are initialized to 0.

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

Two concurrent processes that can be killed with FIFO consistency.

Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
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 issued. Writes from different processes may not always be seen in that order.

Weak Consistency (1)

Use of **synchronized variables** that allow to synchronize all local copies of the data store.

synchronize(S)

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)

Weak consistency enforces consistency of a group of operations not on individual reads or writes.

```
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. When **f** is executed **a** and **b** must be put in memory.

Weak Consistency (3)

P1:	W(x)a	W(x)b	S			
P2:				R(x)a	R(x)b	S
P3:				R(x)b	R(x)a	S

(a)

A valid sequence of events for weak consistency.

P1:	W(x)a	W(x)b	S		
P2:				S	R(x)a

(b)

An invalid sequence for weak consistency.

Release Consistency (1)

Two operations are defined:

acquire(L) and **release(L)**

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).*

Release Consistency (2)

P1:	Acq(L)	W(x)a	W(x)b	Rel(L)				
P2:					Acq(L)	R(x)b	Rel(L)	
P3:								R(x)a

A valid event sequence for release consistency.

P3 does not do an acquire before reading data, so returning **a** is allowed.

Entry Consistency (1)

- Each shared data must be associated with some synchronization variable.
- Lists of shared data items are associated to a synchronization variable.
- Acquire and Release are then more efficient.
- Access to disjoint sets of shared data items can be concurrent.

Entry Consistency (2)

Rules:

1. *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.*

- This means that at an acquire all remote changes to the guarded data must be visible.

Entry Consistency (3)

Rules:

2 *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.*

- That is before updating a a shared data item, a process must enter a critical region in exclusive mode

Entry Consistency (4)

Rules:

3 *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.*

- That is if a process wants to enter a critical region in nonexclusive mode it must check with the owner of the synch variable to get updated copies of data.

Entry Consistency (5)

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 issued. 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.

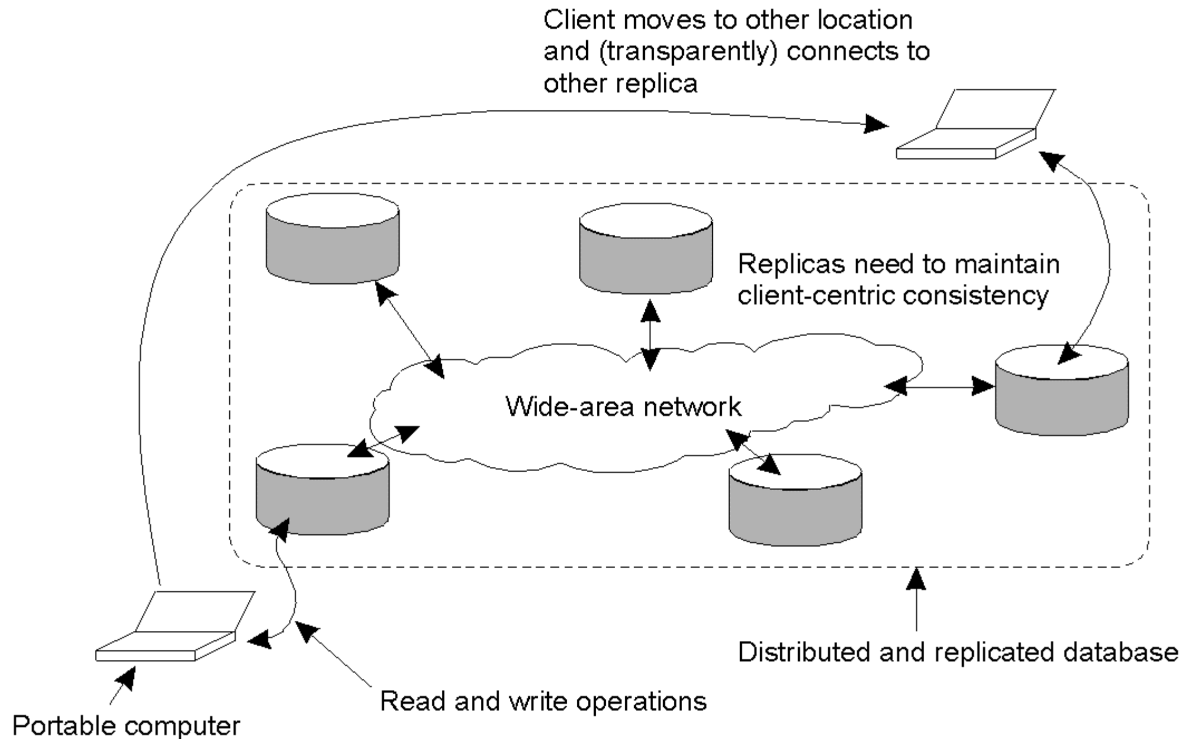
(b)

(a) Consistency models not using synchronization operations.

(b) Models with synchronization operations.

Eventual Consistency

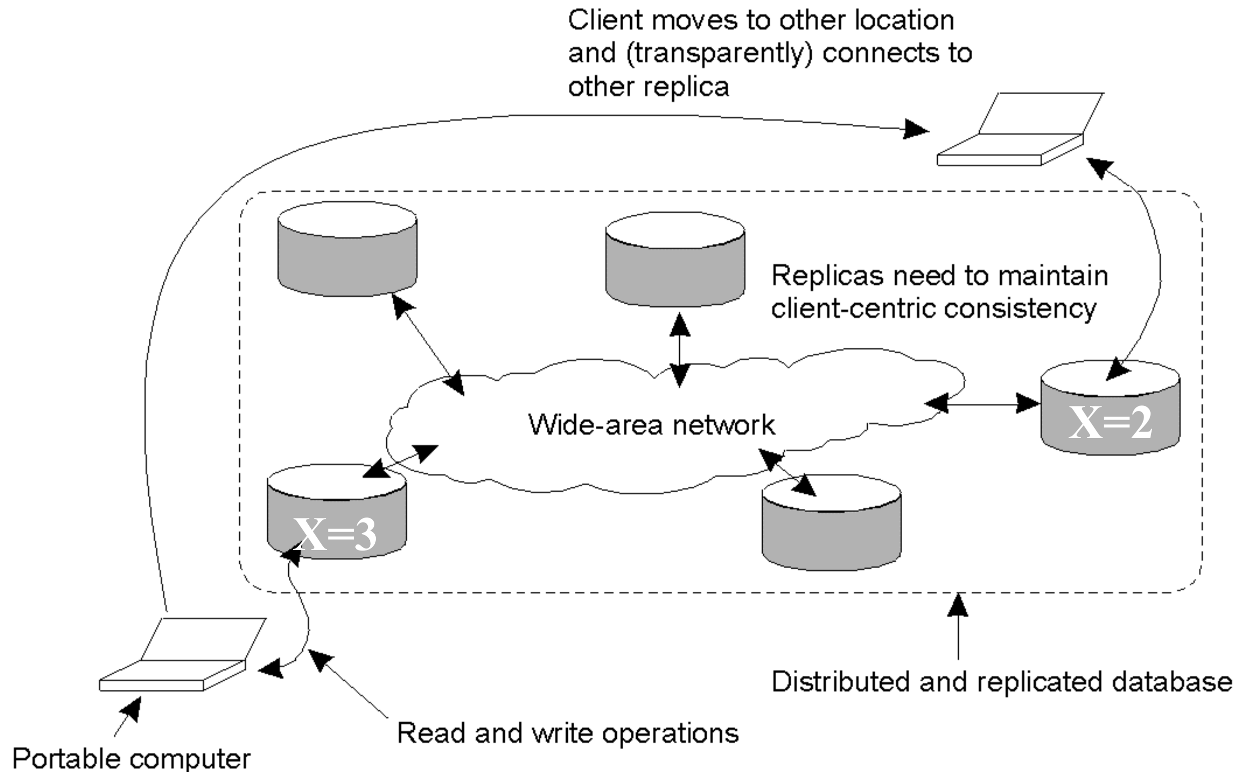
If update do not take place for a long time, all replicas will become inconsistent.



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

Client-centric Consistency

If a user moves can access inconsistent data.



Client-centric consistency models can be used.

Monotonic Reads

Successive reads by a process of a data item x return the same value or a more recent value.

L1:	$WS(x_1)$		$R(x_1)$
L2:		$WS(x_1;x_2)$	$R(x_2)$

(a)

L1:	$WS(x_1)$		$R(x_1)$		
L2:		$WS(x_2)$		$R(x_2)$	$WS(x_1;x_2)$

(b)

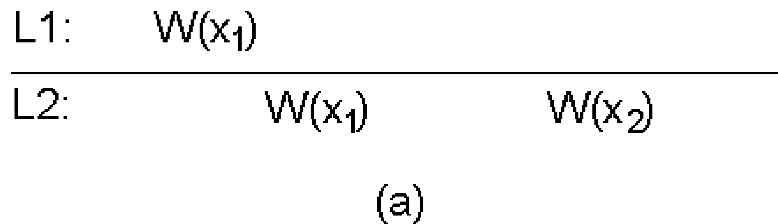
The read operations performed by a single process *P* at two different local copies of the same data store.

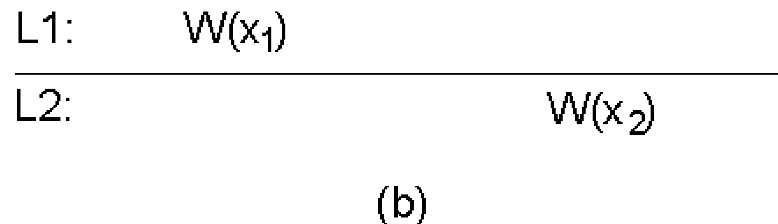
(a) A monotonic-read consistent data store

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

Monotonic Writes

A write operation by a process on a data item x is completed before any successive write on x by the same process.





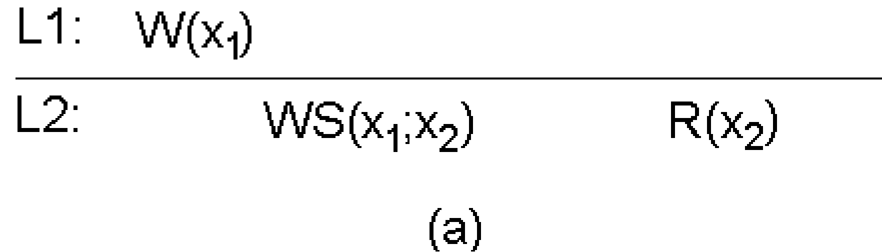
The write operations performed by a single process *P* at two different local copies of the same data store

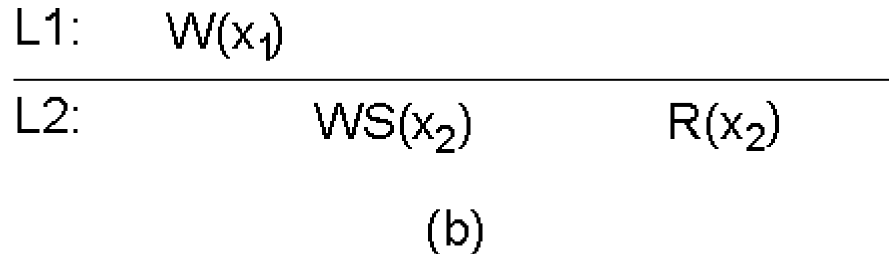
a) A monotonic-write consistent data store.

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

Read Your Writes

The effect of a write operation of a process on a data item x will always be seen by a successive read on x by the same process.

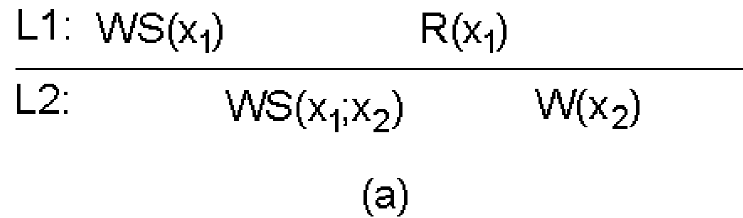


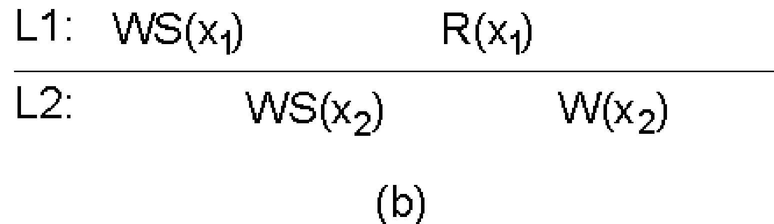


(a) A data store that provides read-your-writes consistency.
(b) A data store that does not.

Writes Follow Reads

A write operation by a process on a data item x following a previous read operation on x takes place on the same or a more recent value of x that was read.

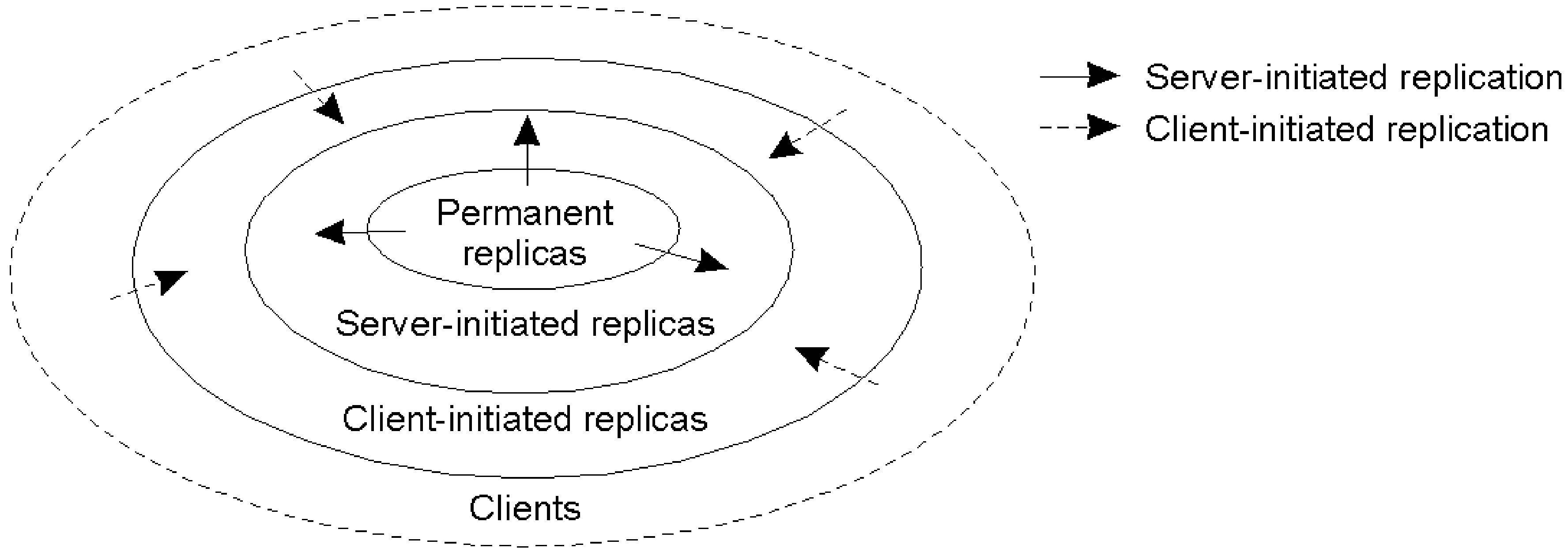




(a) A writes-follow-reads consistent data store

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

Replica Placement



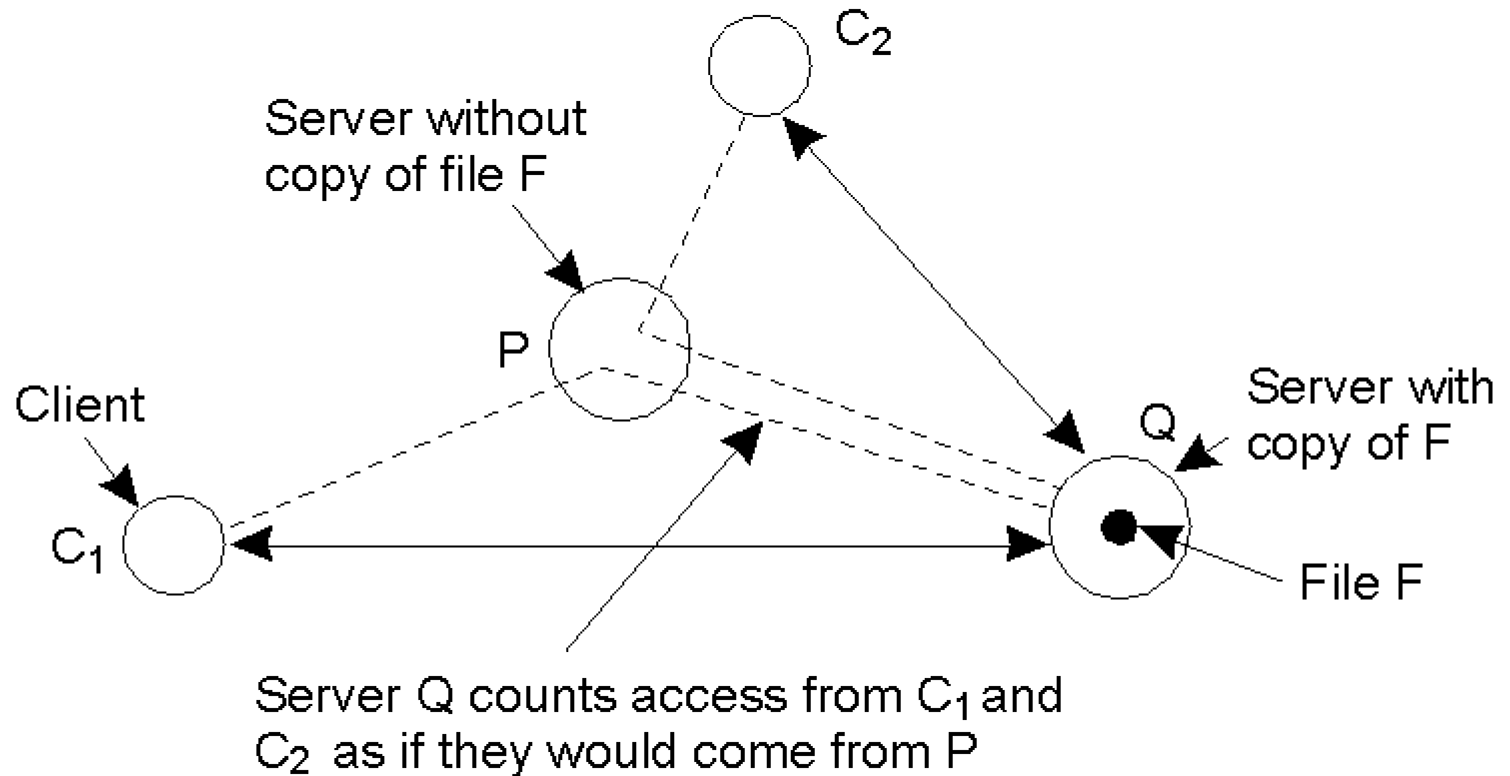
The logical organization of different kinds of copies of a data store into three concentric rings.

Permanent Replicas

Basic form of replicas.

- Web replication on a single local area network
- Web mirroring
- DB replication on a COW

Server-Initiated Replicas



Counting access requests from different clients.

Client-Initiated Replicas

- Client caches are used to temporarily store a copy of data it has just requested.
- Caches can be local or close to the client node.
- The server does not worry about data consistency.
- More than one client can share a cache.

Update Propagation

- How updated values are propagated ?
- Status versus operations
- Pull versus push protocols
- Unicasting versus multicasting

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 Protocols

- A **consistency protocol** defines an implementation of a specific consistency model.
- Primary-based protocols
- Replicated-write protocols
- Cache-coherence protocols.