Réplication et cohérence de données (Data replication and consistency)

Gérald Oster <gerald.oster@loria.fr>

(support de Claudia-Lavinia Ignat)

Organisation

- 9h of lectures
- 3h dedicated to presentations of assigned papers
- 12h of labs
- Web site: http://members.loria.fr/goster/

Evaluation

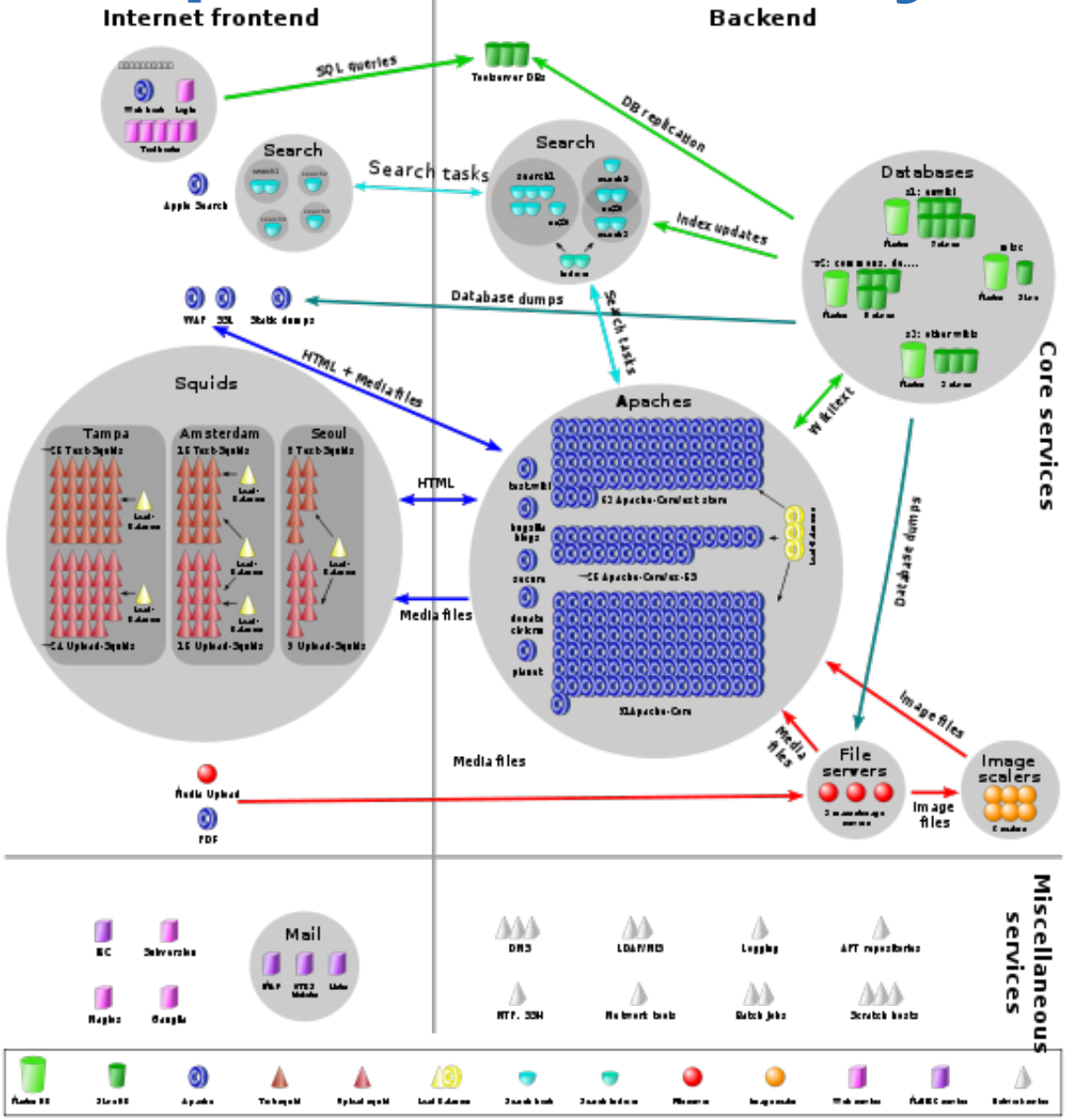
- Paper presentation (~25% of the final mark)
  - Groups of 1 or 2 students
  - 20min presentation + 5-10min questions
- Exam (75% of the final mark)

Course overview

- Introduction to replication
- Consistency models (*)
- Consistency protocols (*)
- Pessimistic replication vs. optimistic replication
- Optimistic replication approaches

(*) Andrew S. Tanenbaum, Maarten Van Steen, "Distributed Systems: Principles and Paradigms“, 2002

Replication is everywhere



Wikipedia

- 25000-60000 page requests/second
- Until 2004 a single server
- In december 2009 344 clusters of Linux servers

Replication is everywhere

- Facebook
  - Most used social network
  - >1.6 billion monthly active users
- Twitter
  - Social networking and microblogging service
  - >500 million users (>330 million active users)
  - 500 million tweets / day
  - 2.1 billion search queries /day
- Google
  - >1 million servers
  - >3.5 billion search requests/day
  - Synchronous data replication (Google Docs, Gmail, Google Sites, Calendar)
- Amazon
  - >278 million active customers account
  - >12 000$ revenue/minute

Introduction

- Distributed system
  - Sites (processing unit + storage device)
  - Communication links (bidirectional communication between 2 sites)
  - Messages (no guarantee to be delivered within a maximum delay)
- Failures of sites and links
  - An error leads to a faulty state
  - Errors: human mistakes or physical damage
  - Site failures: stopping, crash of a critical subsystem, malicious actions (byzantine failures)
  - Link failures: messages are no longer transmitted or excessively delayed, unidirectional transmission, byzantine communication failure
  - One consequence: network partition

Reliability (*Fiabilité*)

- Property of tolerating component failures for the longest time
- A system is perfectly reliable if it never fails
- A system is reliable if it fails rarely and almost always recovers from component failures and design faults s.t. its activity is resumed without perceptible interruption
- A system is reliable to the extent that it is able to successfully complete a service request once it accepts it

Availability (*Disponibilité*)

- Accessibility of system services to users
- A system is highly available if the fraction of its down-time is very small (failures are rare or it can restart very quickly after a failure)
- A system is highly available if denial of service request is rare

Reliability vs. Availability

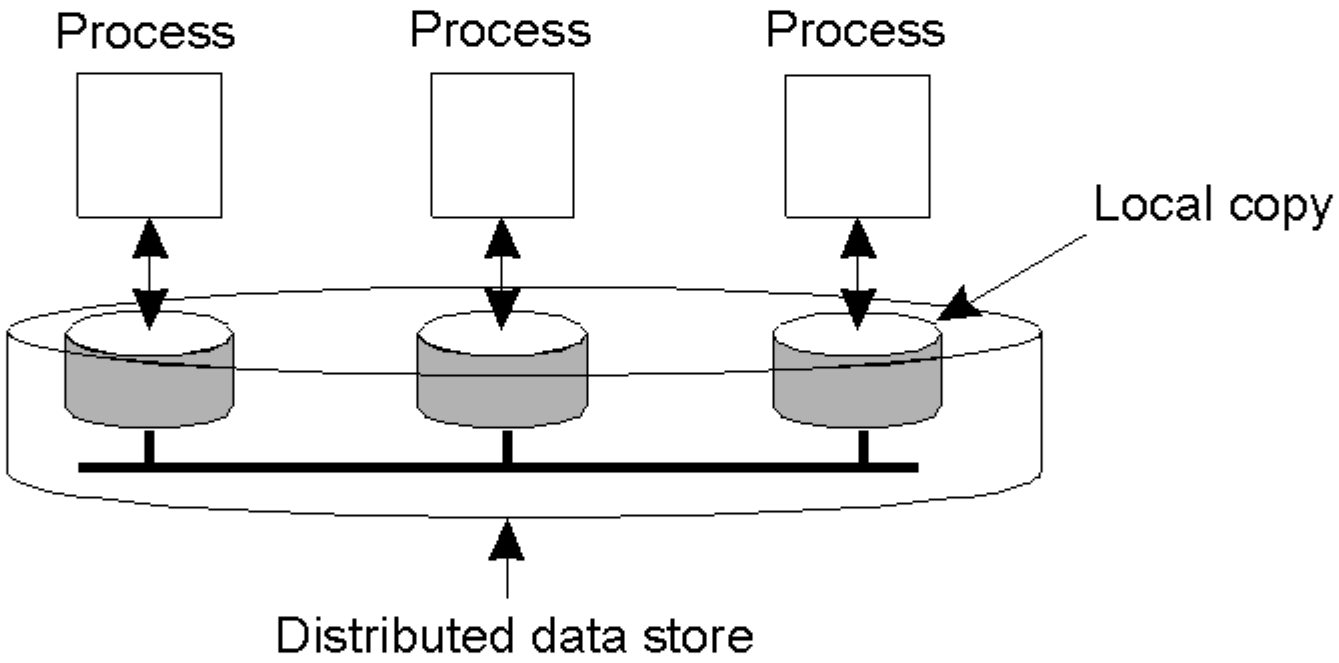
- Reliability: duration of time a system is expected to remain in continuous operation
- Availability: fraction of time instants where the system is operational
- A reliable system is not necessarily highly available
- A highly available system is not necessarily reliable
- Dependability=Reliability x Availability

Reasons for replication

- Reliability
  - System continues to work after one replica crashes
  - Masking failures + system reconfiguration
  - Better protection against corrupted data
- Performance
  - Scaling in numbers (no overloading of servers, i.e. replicated web servers)
  - Scaling with size of geographical area (reduced communication latency, i.e geo-replicated servers)
- Challenge: **How to maintain consistency between replicated data?**

Data-Centric Consistency Models

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



Data-Centric Consistency Models

- Write operation when it changes data, otherwise read operation
- Consistency model=contract between processes and data store
  - If processes agree to obey certain rules, data store promises to work correctly
- A process that performs a read expects a return value that shows results of last write
- Too strict criteria as lack of global clock, need for other consistency models
- Each model restricts the values that a read can return

Strict Consistency

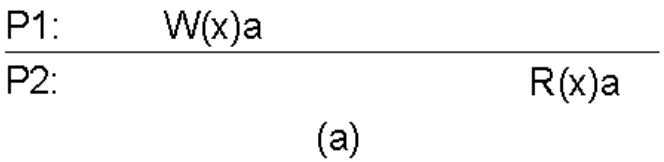
- *Any read on a data item x returns a value corresponding to the result of the most recent write on x*
- Assumes existence of absolute global time
- Two main issues:
  - Definition of “most recent event”
  - Instantaneous execution of operations
- Example:
  - X a data item stored at machine B
  - Machine A reads X at time T_1 and message sent to B to read X
  - At T_2 a process on B writes X
  - Read should return old value of X regardless of where machines are how close T_2 and T_1 is

Strict Consistency

- “most recent event” needs perfectly synchronised clocks (0 delay between any 2 sites)
- An operation that requires a remote access cannot be executed instantaneously (a local operation launched after a remote operation can be terminated before)
- Notation:
  - $W_i(x)a$ – process P_i writes x with value a
  - $R_i(x)b$ – process P_i reads x with value b

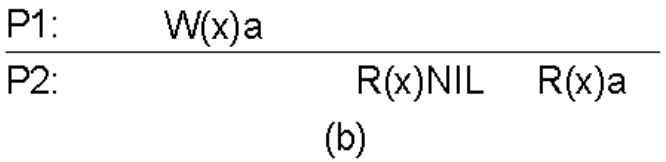
Strict Consistency

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



(a)

A strictly consistent store



(b)

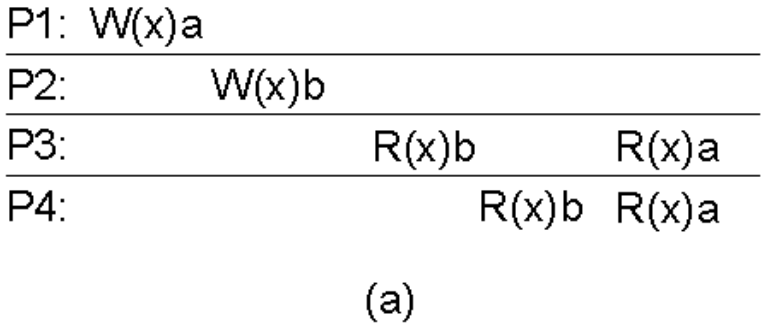
A store that is not strictly consistent.

- Strict consistency is an ideal model
- Need for relaxed consistency models

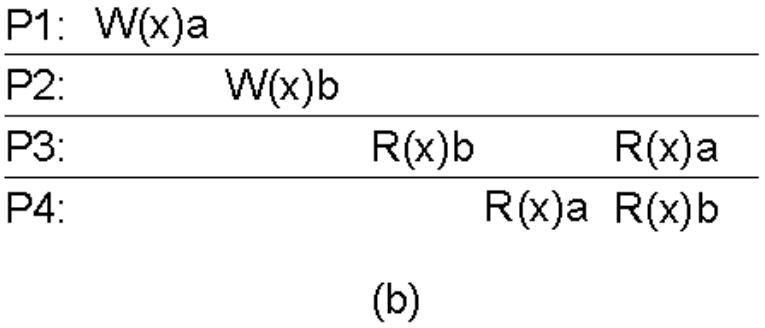
Sequential Consistency (1)

- Sequential consistency (defined by Lamport)
  - *The result of any execution is the same as if the (read and 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*
  - Any valid interleaving of operations is possible, but all processes see the same interleaving
  - There is no reference to the “most recent” write
  - A process sees writes from all processes but only its own reads

Sequential Consistency (2)



(a)



(b)

a) A sequentially consistent data store.

b) A data store that is not sequentially consistent.

Linearizability (1)

- Linearizability weaker than strict consistency but stronger than sequential consistency
  - $ts_{OP}(x)$ timestamp assigned to operation OP performed on data item x
  - *The result of any execution is the same as if the (read and 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 $OP_1(x)$ should precede $OP_2(y)$ in this sequence*
  - A linearizable data store is also sequentially consistent

Example Sequential Consistency (1)

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

Example Sequential Consistency (2)

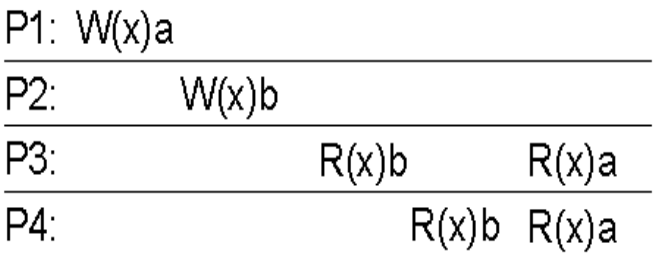
- Four valid execution sequences

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:	**Signature:**	**Signature:**	**Signature:**
001011	**101011**	**110101**	**111111**

- 000000 not permitted
- 001001 not allowed

Formal Expression Sequential Consistency (1)

- Each process P_i has an associated execution E_i of read and write performed on data store S

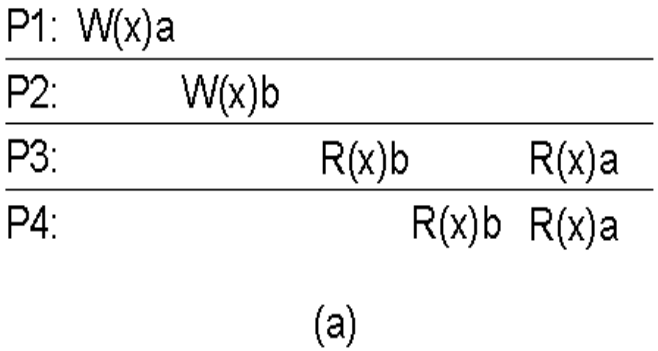


(a)

- E_1: $W_1(x)a$
- E_2: $W_2(x)b$
- E_3: $R_3(x)b$, $R_3(x)a$
- E_4: $R_4(x)b$, $R_4(x)a$
- Merge E_i into H s.t. each operation in E_i appears in H once

Formal Expression Sequential Consistency (2)

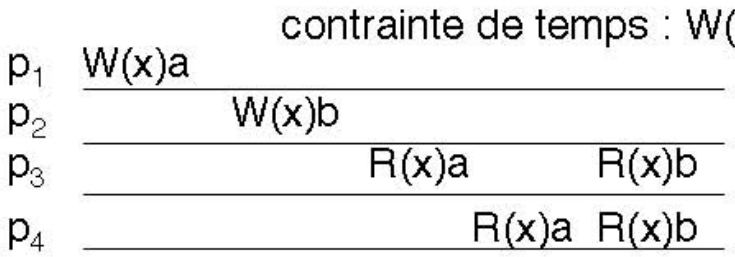
- Legal values for H must obey the constraints
  - Program order must be maintained: if OP_1 before OP_2 in E_i, then OP_1 before OP_2 in H
  - Data coherence must be respected: R(x) must return the value most recently written to x



(a)

- H=$W_2(x)b$, $R_3(x)b$, $R_4(x)b$, $W_1(x)a$, $R_3(x)a$, $R_4(x)a$

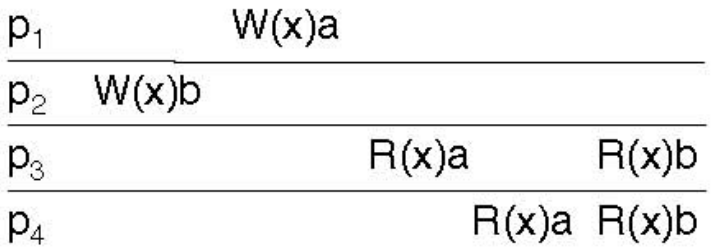
Linearizability and Sequential Consistency



Notation :
$R_i(x)a$: par p_i, lecture de x, résultat a
$W_i(x)a$: écriture de x, valeur a

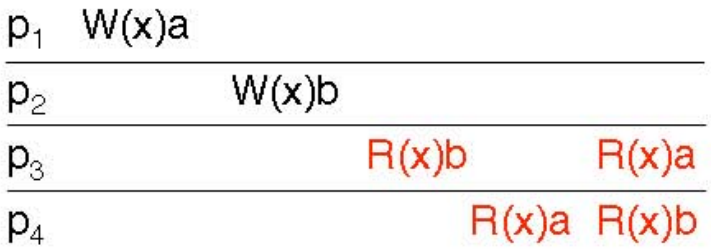
S = $W(x)a\ R_3(x)a\ R_4(x)a\ W(x)b\ R_3(x)b\ R_4(x)b$

linéarisable



S = $W(x)a\ R_3(x)a\ R_4(x)a\ W(x)b\ R_4(x)b\ R_3(x)b$

séquentiel, non linéarisable



non séquentiel

Sequential Consistency

- It is costly to be realised
  - t the minimal transfer time of a message between two sites
  - r the reading period
  - w the writing period
  - $r+w \geq t$
  - Gain on reading results in a lose of writing time

Causal Consistency (1)

- If event B is caused or influenced by an earlier event A, causality requires that everyone sees A and then B
- Concurrent operations = operations not causally related
- Examples:
  - If E_1: p writes x, then E_2: q reads x, then $E1 \rightarrow E2$
  - If E_1: p reads x, then E_2: p writes y, then $E_1 \rightarrow E_2$ (value of y depends on x)
  - If E_1:p writes x, then E_2: q writes y (independently), then $E_1 \| E_2$

Causal Consistency (2)

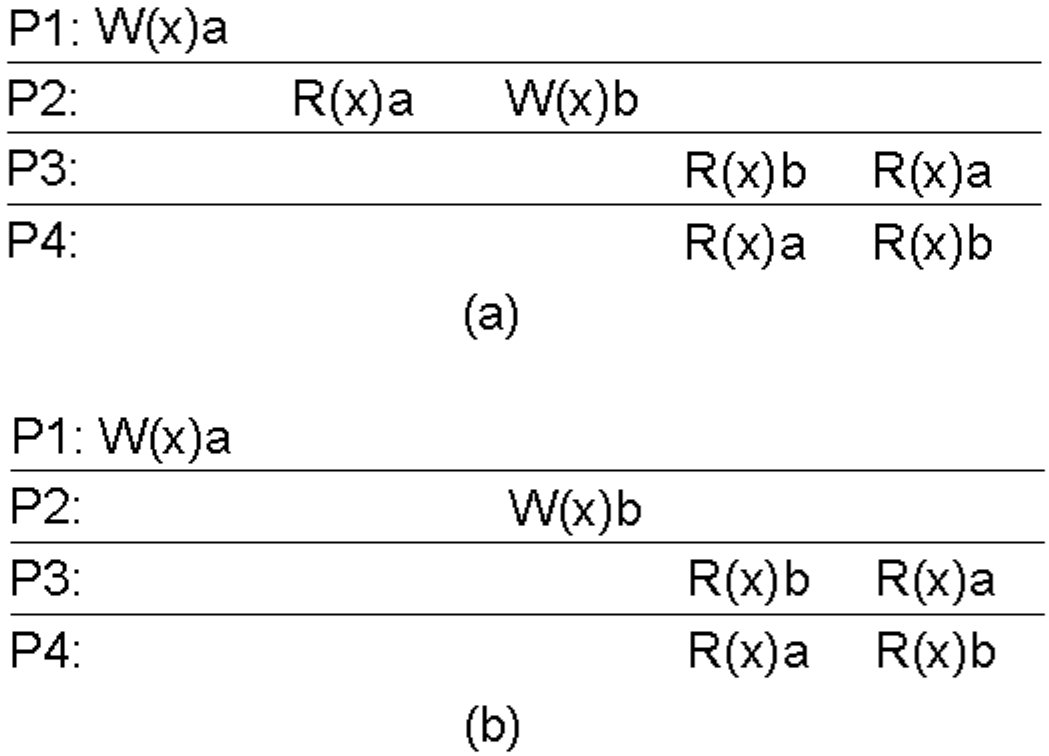
- *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines*.

Causal Consistency (3)

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

- Allowed with a causally-consistent store
- Not allowed with sequentially or strictly consistent store.

Causal Consistency (4)



$W_2(x)b$ depends on $W_1(x)a$

a) A violation of a causally-consistent store.

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

Causal Consistency (5)

- Implementation needs keeping track of which processes have seen which writes
- Construction of a dependency graph
- One solution based on vector timestamps

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.*

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

- valid 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)

- Result impossible to obtain with sequential consistency

FIFO Consistency (4)

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

- With FIFO consistency both processes P_1 and P_2 can be killed

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**

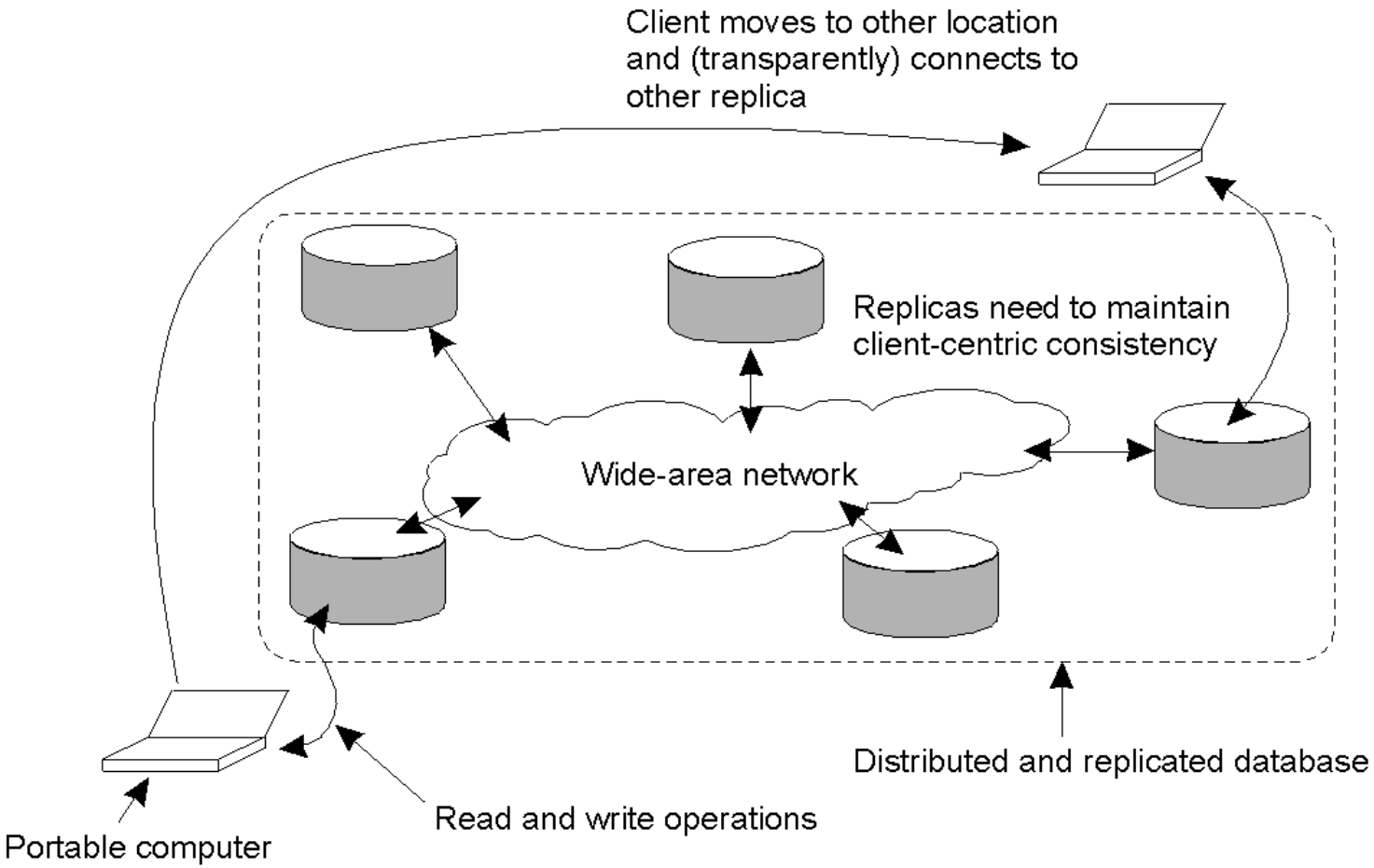
Data Centric vs. Client Centric Consistency Models

- Data-centric consistency model
  - Provides a systemwide consistent view on a data store
  - Assumption: concurrent processes may simultaneously updating the data store
- Client-centric consistency model
  - Assumption: lack of simultaneous updates or when updates happen they can easily been resolved
  - Most operations involve reading data
  - Data stores offer a very weak consistency model called eventual consistency

Eventual Consistency (1)

- Concurrency appears only in a restricted form
  - Database systems
    - most processes hardly perform updates, but only read data
  - DNS
    - domains assigned to naming authorities that are allowed to update their part of the name space
    - no write-write conflicts, only read-write conflicts
    - Acceptable to propagate an update in a lazy fashion
  - World wide web
    - Web pages updated by a single authority
    - No write-write conflicts
    - Design of local caches for efficiency
- If no updates take place for a long time, all replicas become consistent

Eventual Consistency (2)



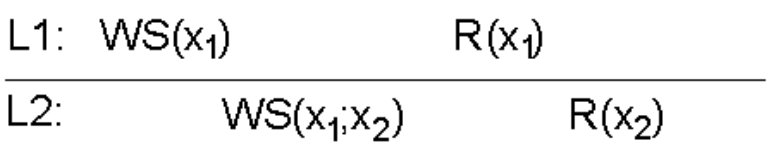
Client centric consistency

- Provides guarantees for a single client concerning consistency of accesses to a data store by that client
- A client connects to different replicas during a period of time and the differences should be made transparent
- Whenever a client connects to a new replica, that replica is brought up to date with the data that was manipulated by that client before and can reside at other replica sites
- Notations
  - $x_i[t]$ version of x at local copy L_i at time t
  - $WS(x_i[t])$ series of write operations at L_i that took place since initialization
  - $WS(x_i[t_1];x_j[t_2])$ if operations in $WS(x_i[t_1])$ have been performed at the local copy L_j at time t_2

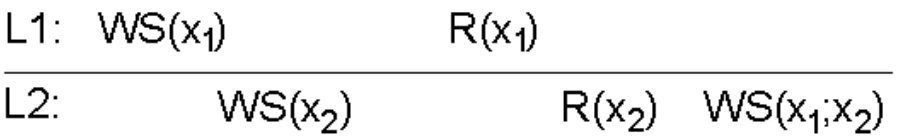
Monotonic Reads (1)

- *If a process reads the value of a data item x, any succesive read operation on x by that process will always return that same value or a more recent value*
- Example:
  - A distributed email database
  - Each user's mailbox may be distributed and replicated across multiple machines
  - Mail can be inserted at any location
  - Updates propagated in a lazy manner
  - Emails read (no remove, etc.) at location X are present when they are read later at location Y

Monotonic Reads (2)



(a)



(b)

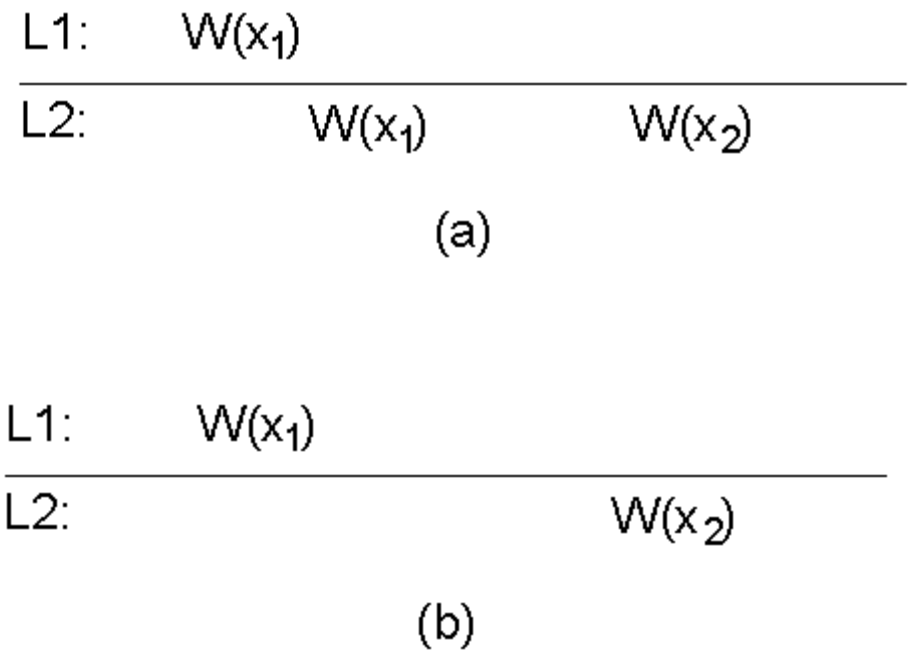
a) A monotonic-read consistent data store

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

Monotonic Writes (1)

- *A write operation by a process on a data item x is completed before any successive write operation on x by the same process*
- Similarity with FIFO consistency
- Example: updating a software library

Monotonic Writes (2)



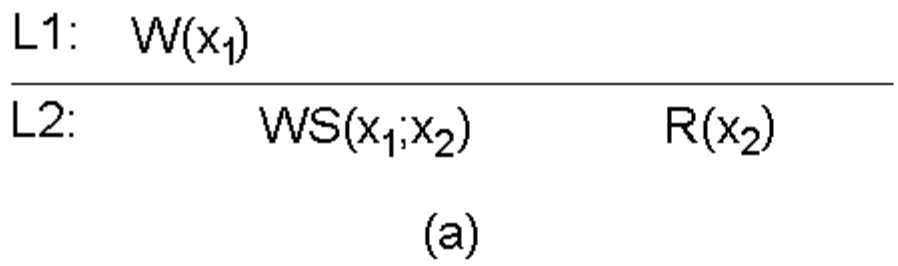
a) A monotonic-write consistent data store.

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

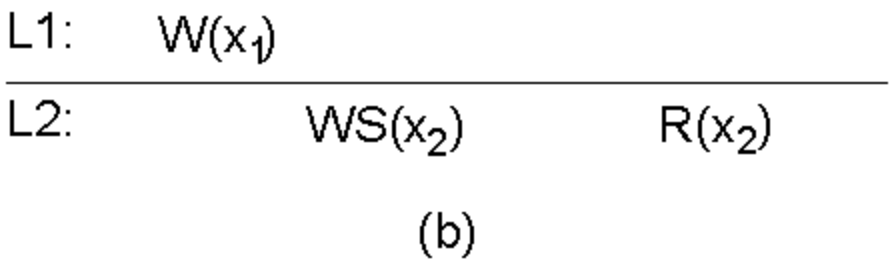
Read Your Writes (1)

- *The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process*
- Examples
  - Updating HTML pages
  - Updating passwords

Read Your Writes (2)



(a)



(b)

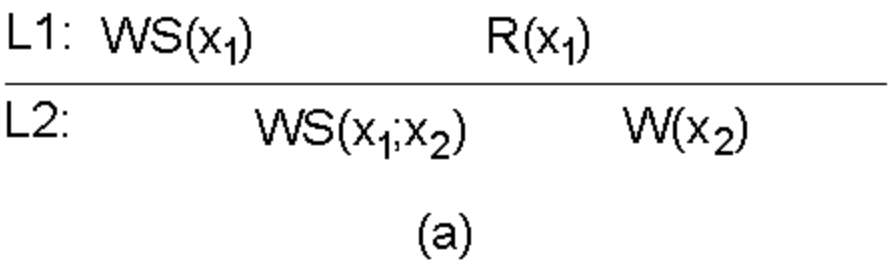
a) A data store that provides read-your-writes consistency.

b) A data store that does not.

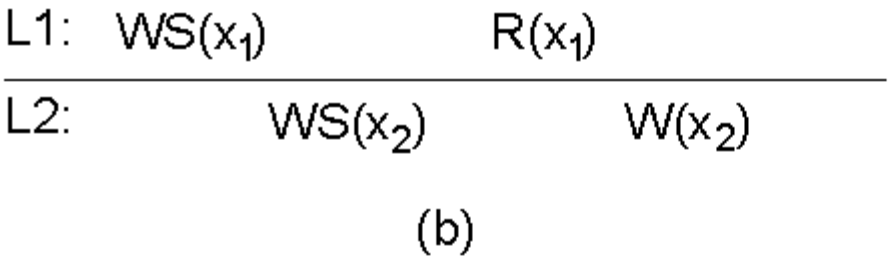
Writes Follow Reads (1)

- *A write operation by a process on a data item x following a previous read operation on x by the same process, is guaranteed to take place on the same or a more recent value of x that was read*
- Example:
  - Network newsgroup that see a posting of a reaction to an article only after they have seen the original article

Writes Follow Reads (1)



(a)



(b)

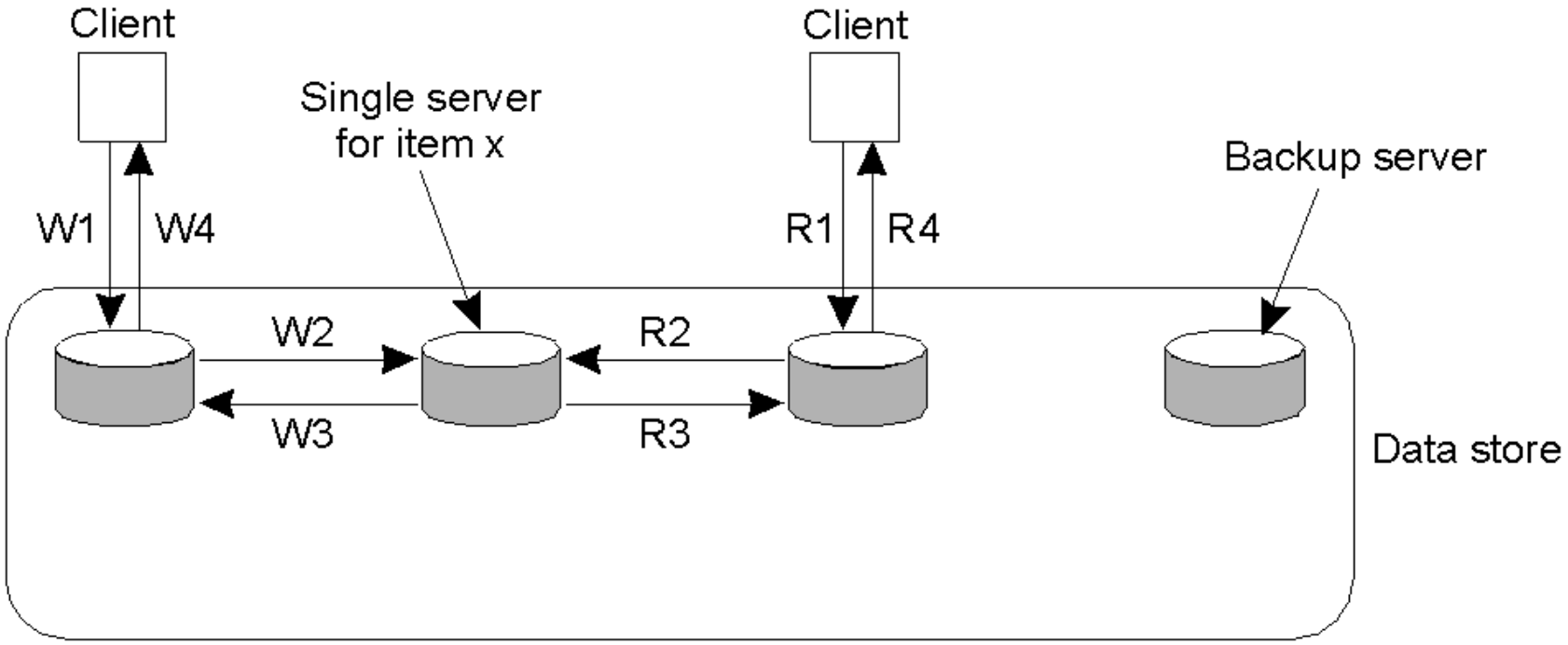
a) A writes-follow-reads consistent data store

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

Consistency protocols

- Describe implementation of a specific consistency model
- Primary-based protocols
  - Each data item x has an associated primary responsible for coordinating write operations on x
  - Remote-write protocols
  - Local-write protocols
- Replicated-write protocols
  - Write operations can be carried out at multiple replicas
  - Active replication
  - Quorum-based protocols

Remote-Write Protocols (1)

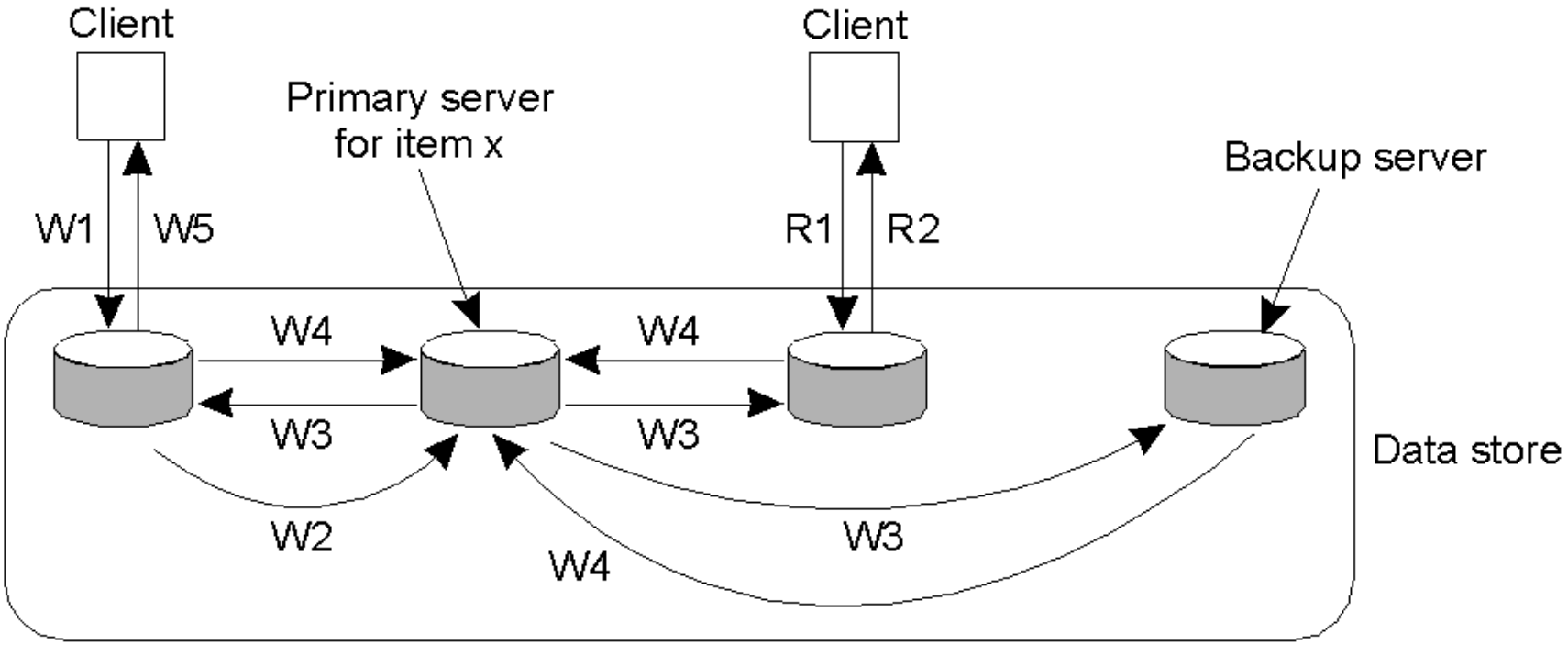


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

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

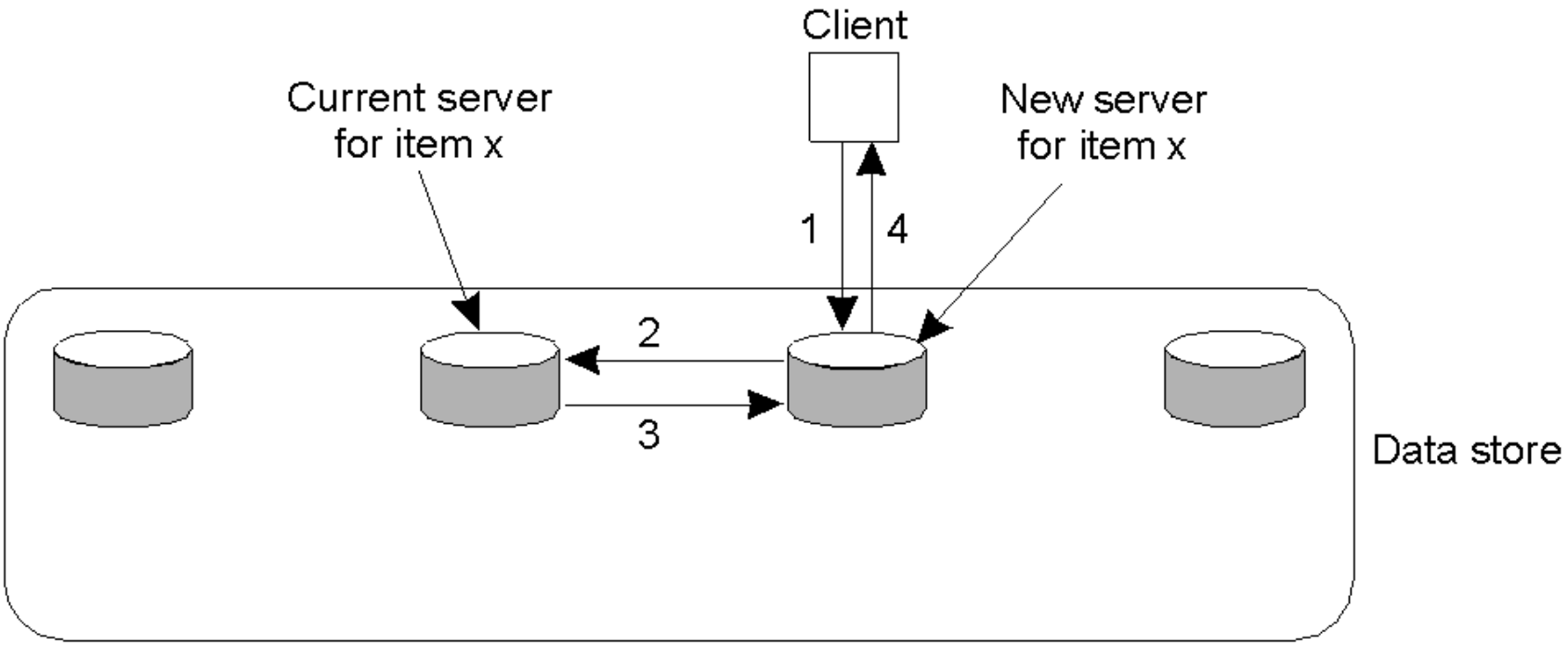
R1. Read request
R2. Response to read

- **The principle of primary-backup protocol.**

Remote-Write Protocols (3)

- Primary-backup protocol implements update as a blocking operation
- Alternative solution: non-blocking protocol
  - As soon as primary updated the local copy of x, it returns an acknowledgement
  - After that ask backup servers to perform the update
  - Fault tolerance concerns
- Implementation of sequential consistency

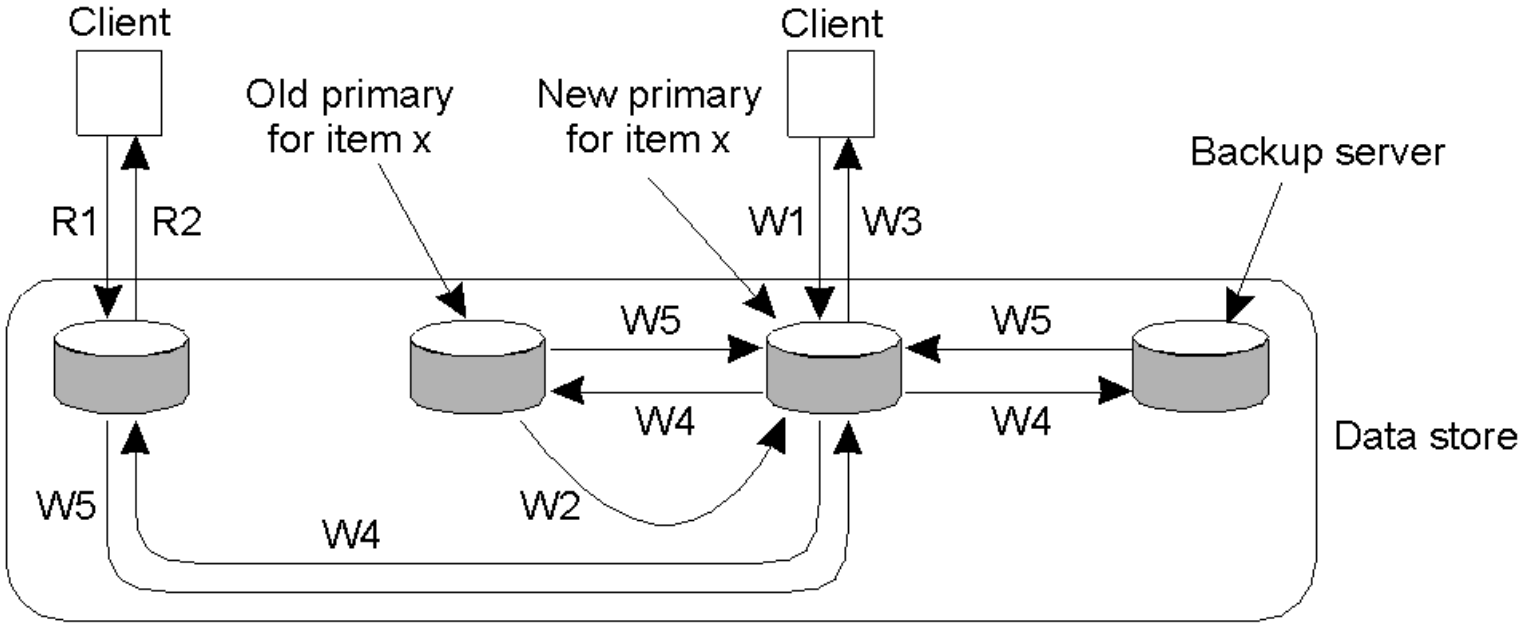
Local-Write Protocols (1)



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
- Disadvantage: keeping track where each data item currently is

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

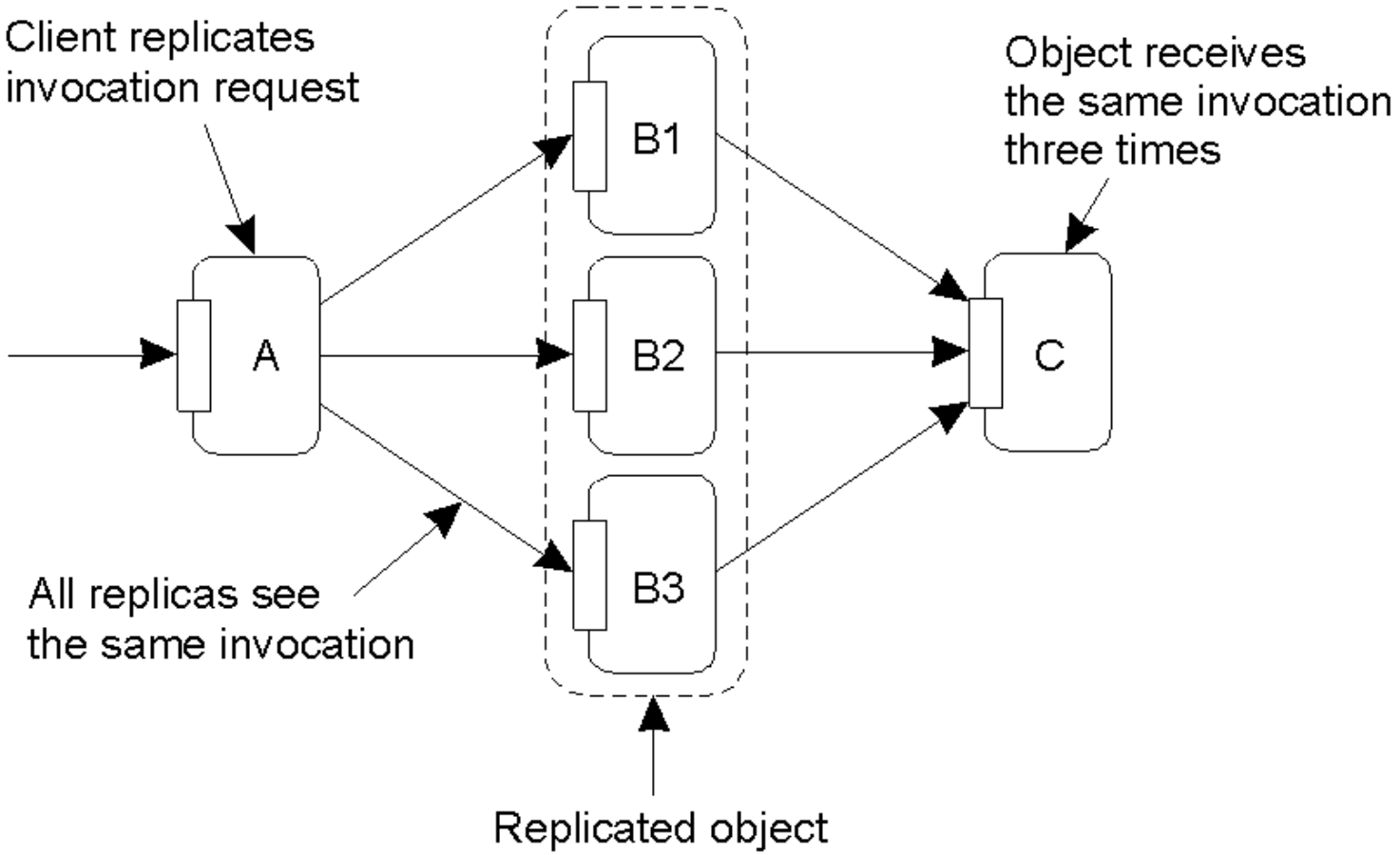
R1. Read request
R2. Response to read

- Primary-backup protocol in which the primary migrates to the process wanting to perform an update
- Advantage if nonblocking protocol: write operations carried locally, while reading can access local copies
- Protocol suitable for mobile computers

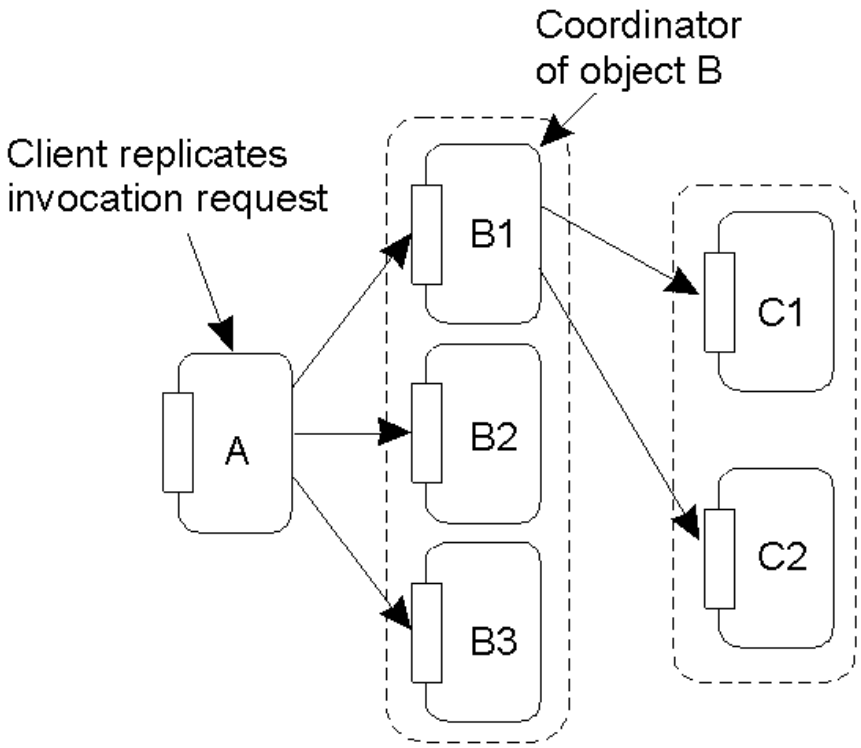
Replicated write protocols
Active Replication (1)

- Operations sent to each replica
- Operations have to be carried out in the same order everywhere
  - Need of totally-ordered multicast
    - Using Lamport timestamps
    - Using a central coordinator called sequencer
- Deal with replicated invocations

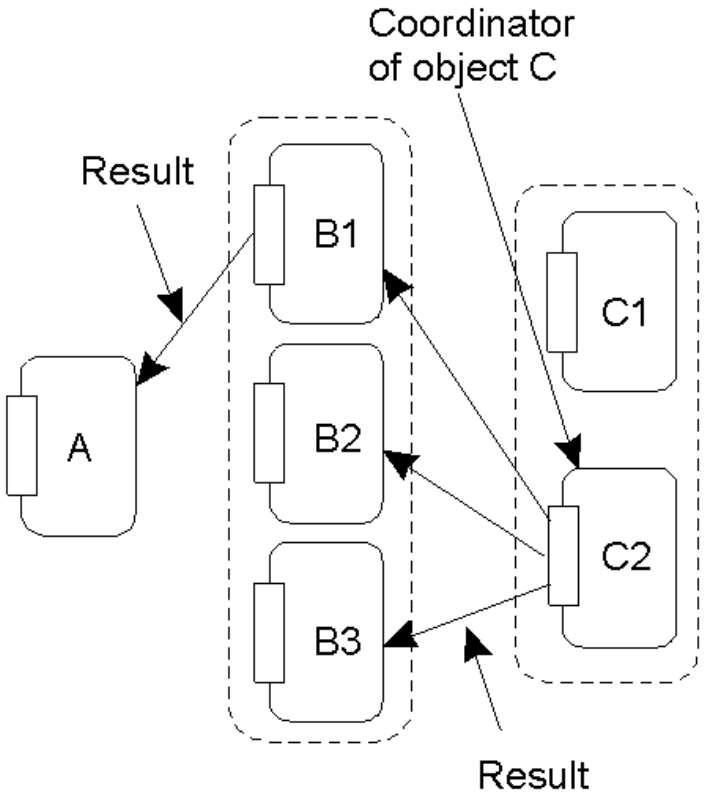
Active Replication (2)



Active Replication (3)



(a)



(b)

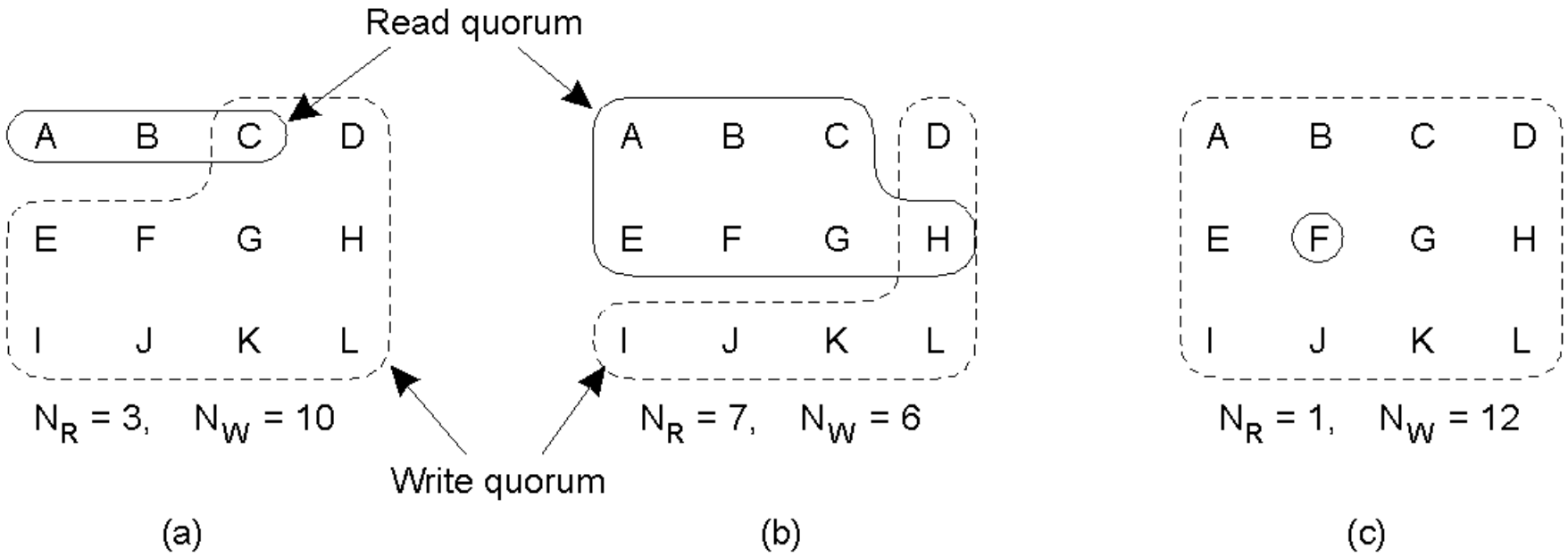
Quorum-Based Protocols (1)

- Use voting: clients request and acquire permission of multiple servers before reading/writing a replicated object
- Example distributed file system
  - File replicated on N servers
  - For an update a client must contact a majority of servers (half +1)
  - If agreement file changed and version number updated
  - For a read a client must contact at least half of servers+1 and ask them to send version numbers of the file
  - Choose the most recent version

Quorum-Based Protocols (2) -Gifford scheme

- A file with N replicas
- A read quorum (N_R servers) for reading the file
- A write quorum (N_W servers) for modifying the file
- $N_R + N_W > N$
- $N_W > N/2$

Quorum-Based Protocols (3)



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)