Concepts of Distributed Systems 2006/2007

Consistency & replication

Johan Lukkien

Contents

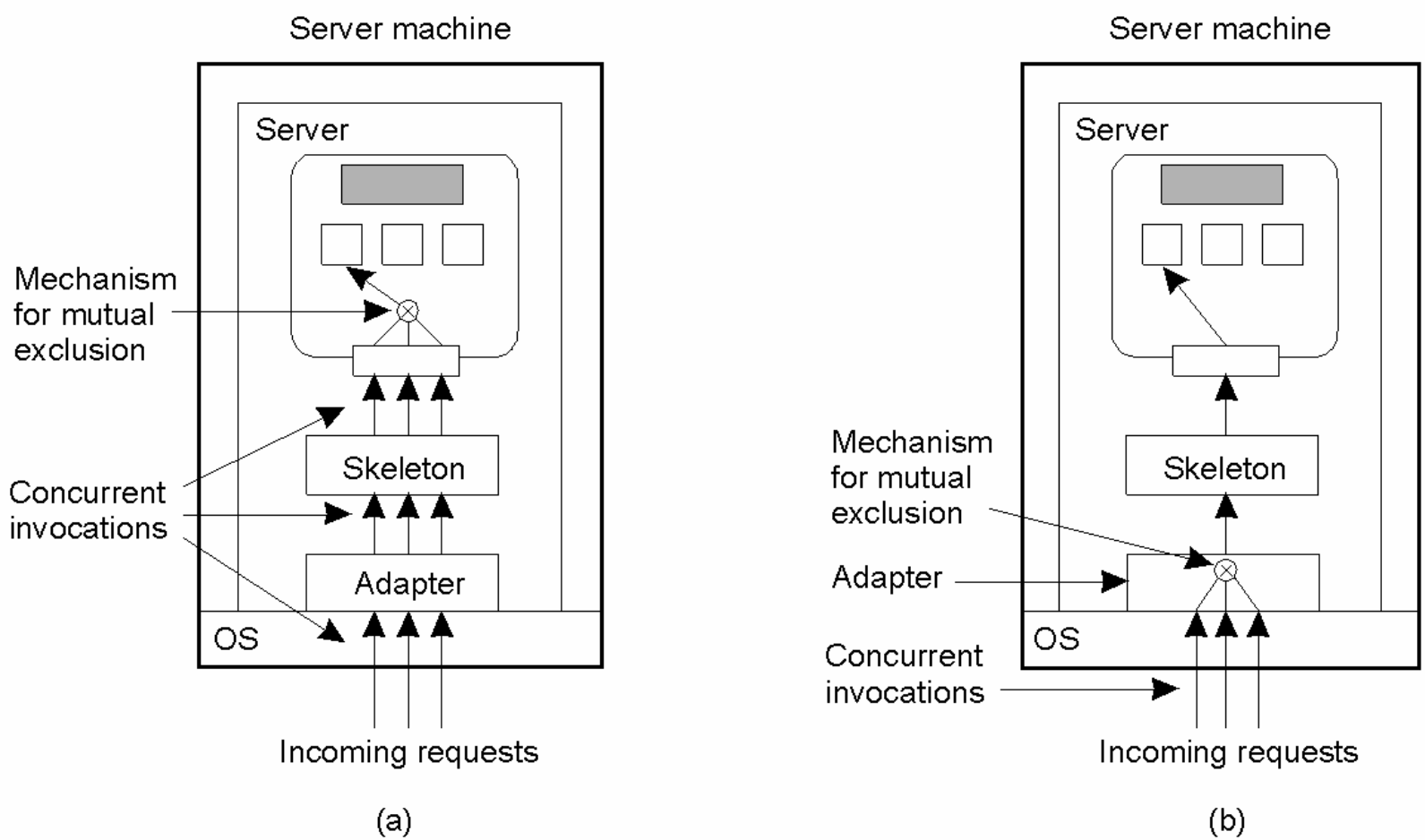
- Replication, background and motivation
- Consistency models
  - data centric
    - strong and weak
  - client centric
- Implementation
  - replication strategies
  - consistency protocols

Consistency and concurrency control

- Shared objects need concurrency control to *maintain consistency* (state invariants)
  - thread safety
  - exclusion, synchronization

- Responsibility
  - object itself
    - concurrency aware
  - in the access path, e.g. serialize access
    - server, adapter
    - general middleware layer

Concurrency (un)aware



Replication

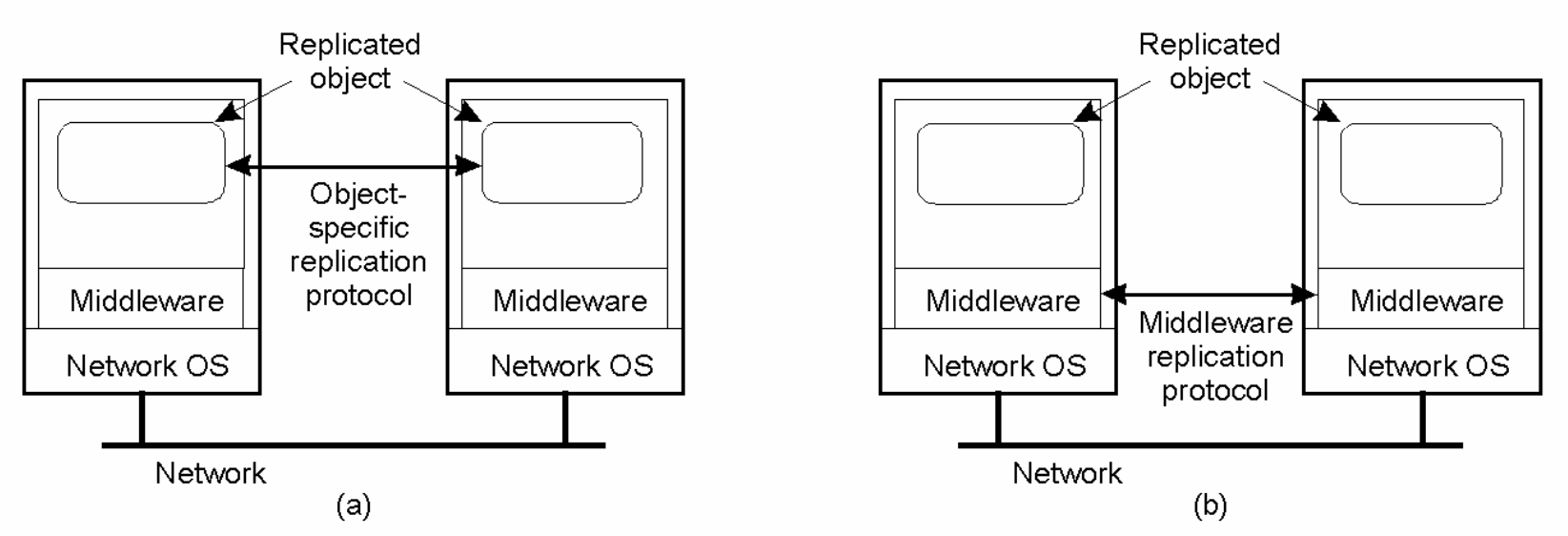
- Of data (objects), servers
- For performance (scalability)
  - caching
  - replicating servers
- ... or reliability
  - fault tolerance
- Two issues
  - additional concurrency control: consistency of replica's
  - replication strategy

Caching

- Benefit:
  - reduce retrieval latency
  - reduce (peak, average) bandwidth use
  - reduce server load
    - from memory module to web server

- Needed:
  - locality of reference
    - spatial & temporal
  - *overhead cost less than cost of retrieval*

Replication (un)aware



a) replication-aware distributed objects

b) middleware responsible for replica management

Trade efficiency for transparency

Replication and scalability

- Scalability is a strong reason for replication
- However, consistency takes bandwidth and resources as well
- Try to relax the consistency requirements
- Trade-off, depending on
  - the consistency requirements
  - the update frequency
- Consistency itself may scale badly!!
  - not all scenario's lend themselves to replication

Contents

- Replication, background and motivation
- Consistency models
  - data centric
    - strong and weak
  - client centric
- Implementation
  - replication strategies
  - consistency protocols

Consistency models

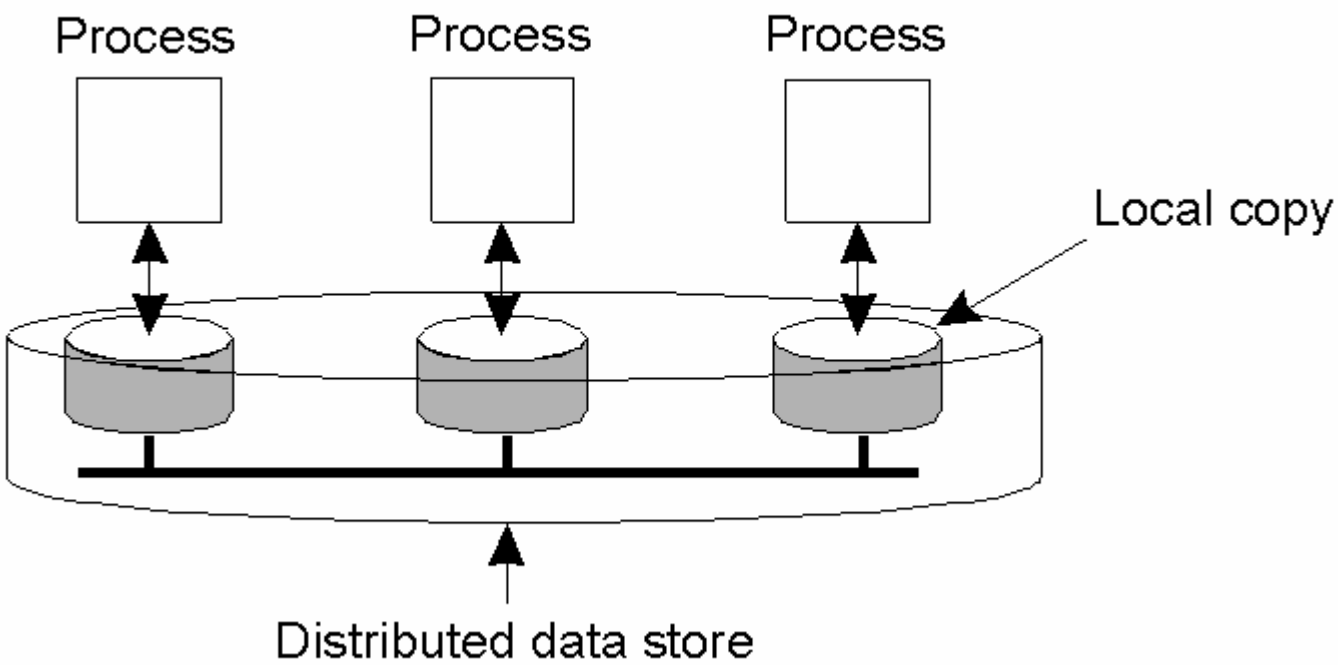
- A *consistency model* describes assumptions on the replicated data set on which an application bases its operation
  - serves as specification for an implementation
  - “contract” between application and a data store (global memory)
- Data-centric
  - expressed in terms of operations on a global memory
- Client-centric
  - expressed from the point of view of a (mobile) client operating on a global memory
- The model (implicitly) describes how to handle *conflicts*
  - read-write and write-write

Data centric consistency models

- Strong models:
  - except for the model assumptions, the client is unaware of replication
  - strict, sequential, causal and FIFO consistency
    - decreasing order of strictness

- Weak models: limit the time that consistency can be assumed
  - specify a validity period (actually sacrifices consistency)
    - lease, time-to-live
  - group operations after which consistency is enforced
    - consistency aware
    - need some mechanism to enforce synchronization
      - consistency primitives

In a picture


Process
Process
Process
Local copy
Distributed data store

Global memory model

- Read's and write's on a global memory
  - $W_i(x)a$: *i* writes value *a* in variable *x*
  - $R_i(x)b$: *i* read value *b* from variable *x*
- Read's and write's are performed on local copies
- Write's are propagated to other copies
- The consistency model describes the effects of sequences of these operations

Strict Consistency

- *Any read on a data item x returns the value of the most recent write on x*
  - requires immediate write effects
  - uses a global notion of time

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

(a)

correct

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

(b)

wrong

Sequential consistency (Lamport)

- *An execution (sequence of read's and write's) is according to a possible interleaving*
  - all processes see the same interleaving!
  - must be able to reconstruct this interleaving....

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)

Linearizable

- *An execution is sequentially consistent and obeys the partial order of a timestamping according to Lamport's timestamps.*

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Example

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

- Initial values 0

Interleavings

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	Signature: 101011	Signature: 110101	Signature: 111111
(a)	(b)	(c)	(d)

Causal consistency

- *Write's that are (potentially) causally related must be seen by all processes in the same order*
  - says nothing about unrelated writes

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

Causality violation

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

(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): violation; (b): ok

FIFO Consistency

- *Write's by a single process are seen in the same order by all processes*
  - i.e., restricted to a single process the sequence of write's is valid
  - not much better than: messages from a single source are delivered in order

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

Exercise

- What are possible behaviors with
  - FIFO consistency?
  - Sequential consistency?
  - (initially, both *x* and *y* are 0)

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

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Weak Consistency

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

Synchronization enforced

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)

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

(b)

- (a): correct; (b): wrong

Critical sections: acquire/release

- 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 complete, all previous reads and writes by the process must have completed
  - Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).
- Called 'release consistency' – makes distinction in the use of a synchronization primitive
  - release: export changes
  - acquire: import changes

Entry Consistency

- Rules:
  - 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.
- Per data item
- Exclusive access through ownership

Examples

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

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

Summary of consistencies

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)

Contents

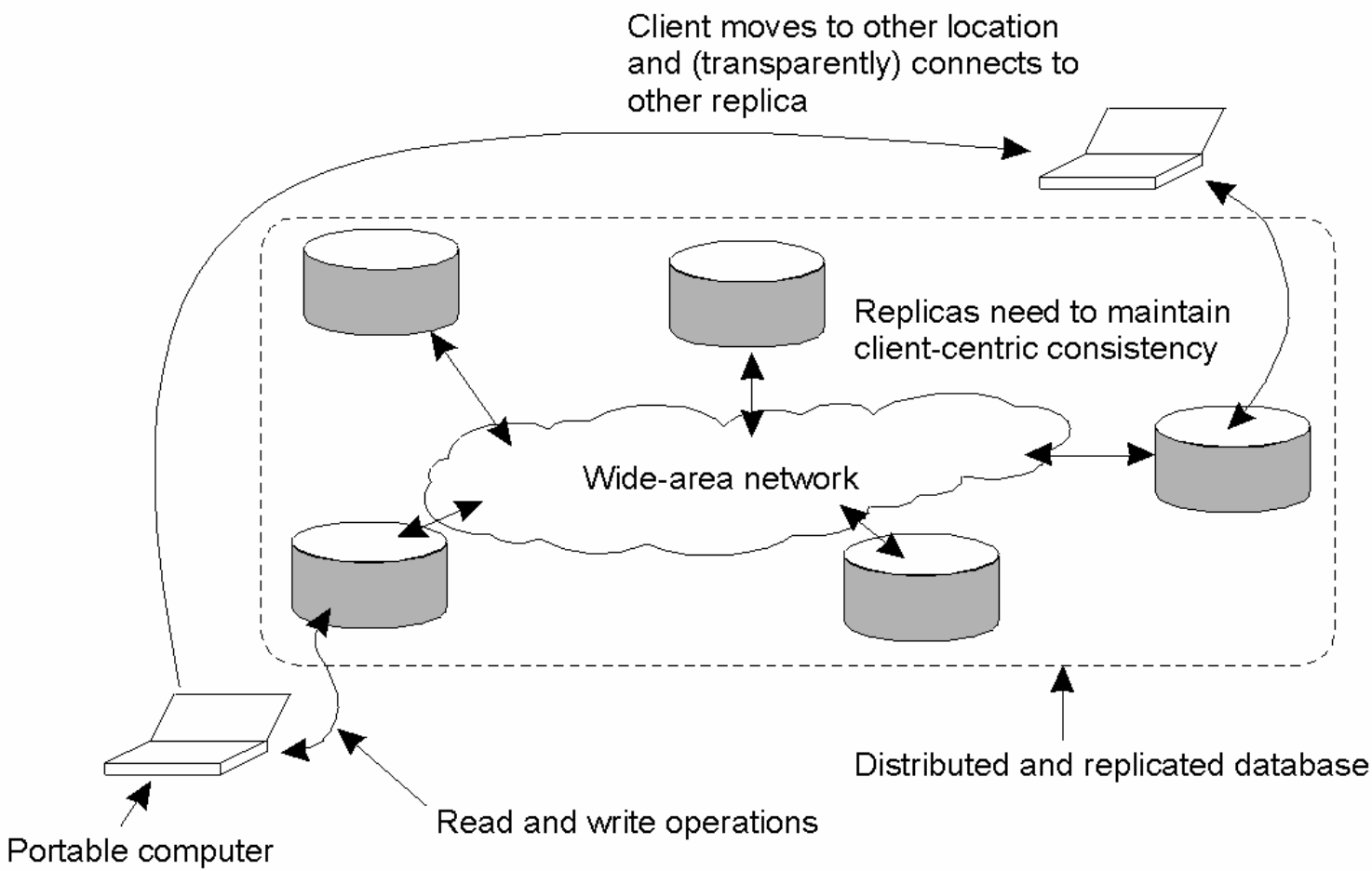
- Replication, background and motivation
- Consistency models
  - data centric
    - strong and weak
  - client centric
- Implementation
  - replication strategies
  - consistency protocols

Client-centric consistency

- Focus on what specific clients want
  - may be more efficient

- **Eventual consistency**: *when left to itself, the system converges to a state in which all replica's have the same value*
  - commonly used in distributed systems
    - e.g. news, www, DNS, ...
  - usually reasonable if client sticks with same replica
  - mobile client: need to specify how this client sees the data

Mobility and eventual consistency

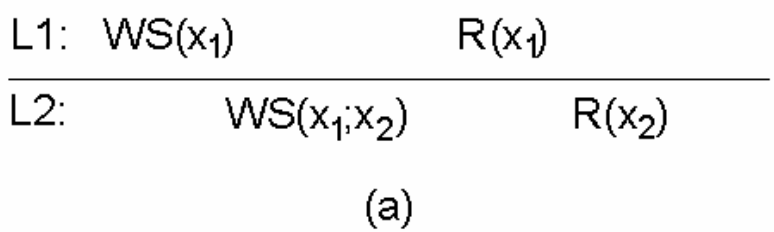


Notation

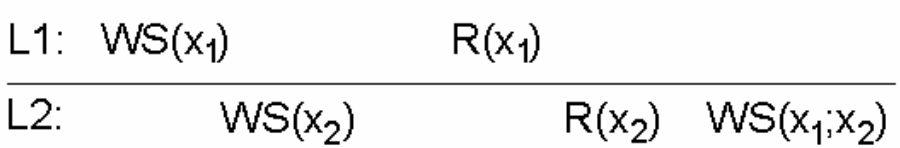
- Single process accesses data at different locations L_i
- $x_i[t]$ denotes data item x at time t at location L_i
- $R(x_i[t])$, $W(x_i[t])$ *respectively* denote a read and a write operation on $x_i[t]$
- $WS(x_i[t])$ denotes a series of write operations on x at L_i since initialization, resulting in $x_i[t]$
- $WS(x_i[t1];x_j[t2])$ denotes a series of writes at location i followed by a series of writes at location j
- When it is clear, t is omitted

Monotonic Reads

- *A read always returns the same or a more recent result than the previous read*



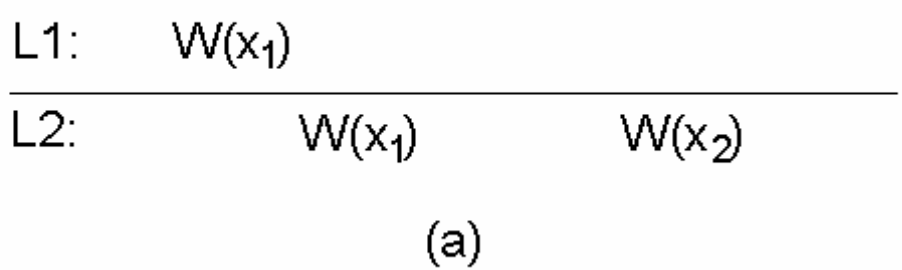
(a)



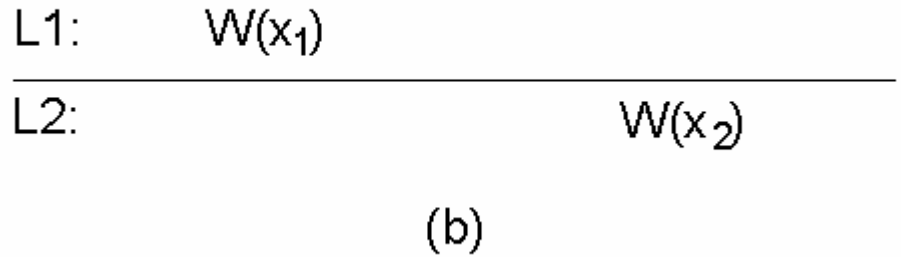
(b)

Monotonic Writes

- *A write operation completes before a successive write operation (at possibly another location)*



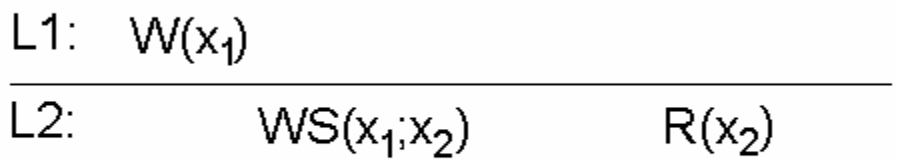
(a)



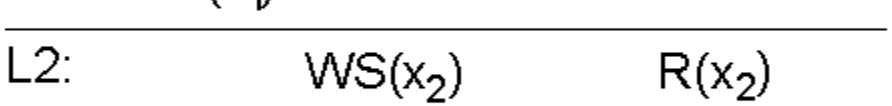
(b)

Read Your Writes

- *A write is effective before a successive read*
  - e.g. edit and compile



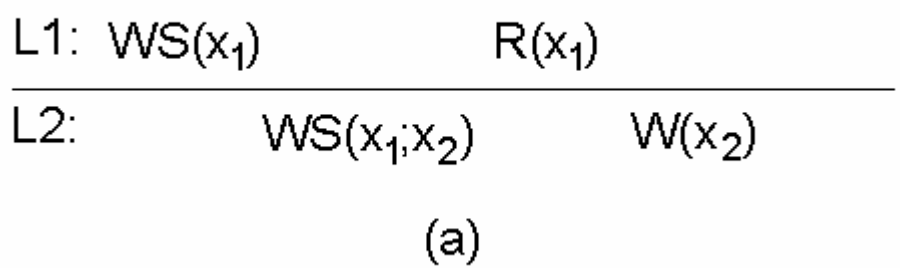
(a)



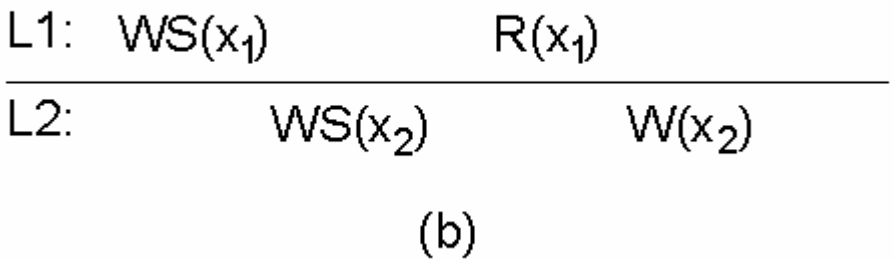
(b)

Writes Follow Reads

- *A write following a read will operate on the same or a more recent value*



(a)



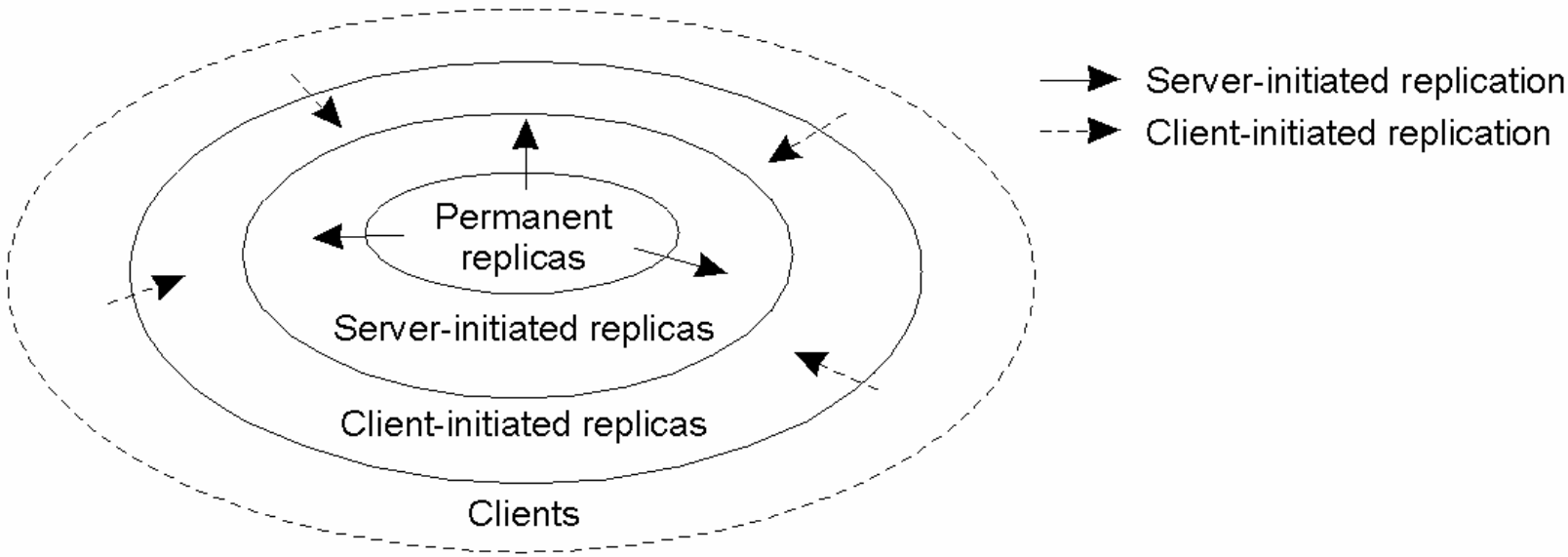
(b)

Contents

- Replication, background and motivation
- Consistency models
  - data centric
    - strong and weak
  - client centric
- Implementation
  - replication strategies
  - consistency protocols

Replica Placement

- Why, where, when, who?
- Types of copies



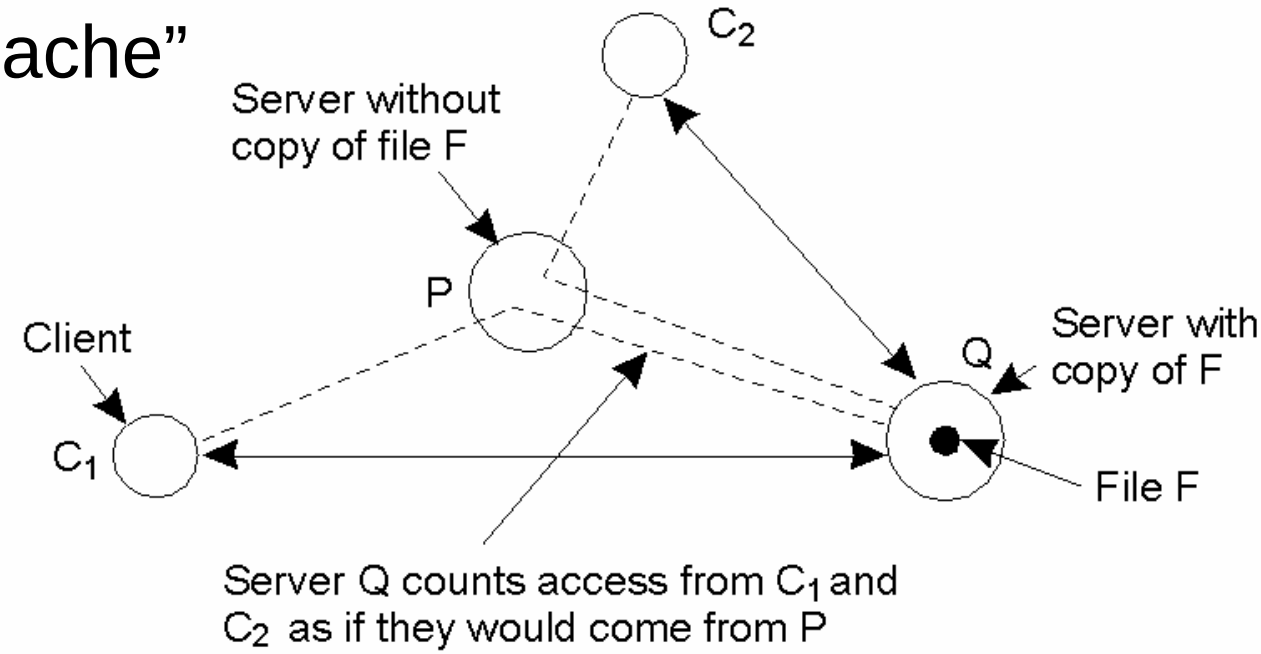
Replica's

- Permanent: "Initial configuration"
  - mirrors
  - copies of databases across network of workstations
    - e.g. round robin access

- Client initiated: "cache"

Server-Initiated Replica's

- Dynamic creation to increase performance
  - move data to the place where it is needed
  - load balancing
- "push cache"



Update propagation

- Notify/invalidate: for caches
- Copy: for databases
- Propagate the operation:
  - trade processing for bandwidth

- Push: send from server to client without request
- Pull: obtain actively from server
- In between: use a lease

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

Contents

- Replication, background and motivation
- Consistency models
  - data centric
    - strong and weak
  - client centric
- Implementation
  - replication strategies
  - consistency protocols

Algorithms for “eventual consistency”

- Epidemic
  - some sort of floodfill
  - not immediate; protocol executes at regular times
  - not total
  - needs time-stamping
- Examples
  - Gossiping:
    - tell a random neighbor about changes
    - if it knows, stop with a certain probability
  - Anti-entropy: exchange the state with a random partner
- Question: how to remove a data item?

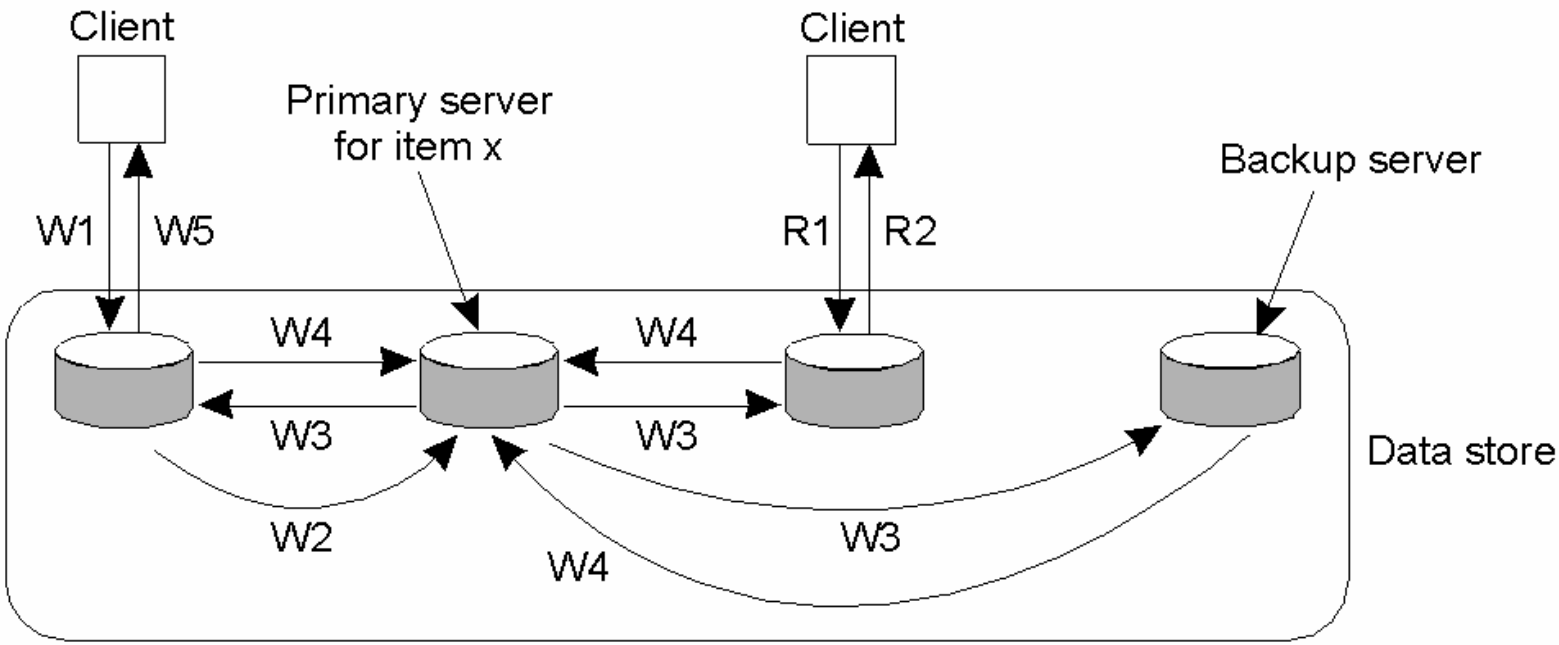
Consistency protocols

- Implement the models
- Protocol types
  - Based on a primary copy
  - Replicated write's
  - Cache coherence

Primary-based

- One (server holding a) particular copy of the data is in charge of maintaining it
  - remote read/write: just single copy at server
  - primary backup: read local, write remote
    - to primary server
    - that forwards it to backup servers
    - and after that acknowledges the write
    - implements sequential consistency
      - with blocking operations
  - local-write: put the primary copy where it is used
    - same problems as with migrating objects (see: naming)

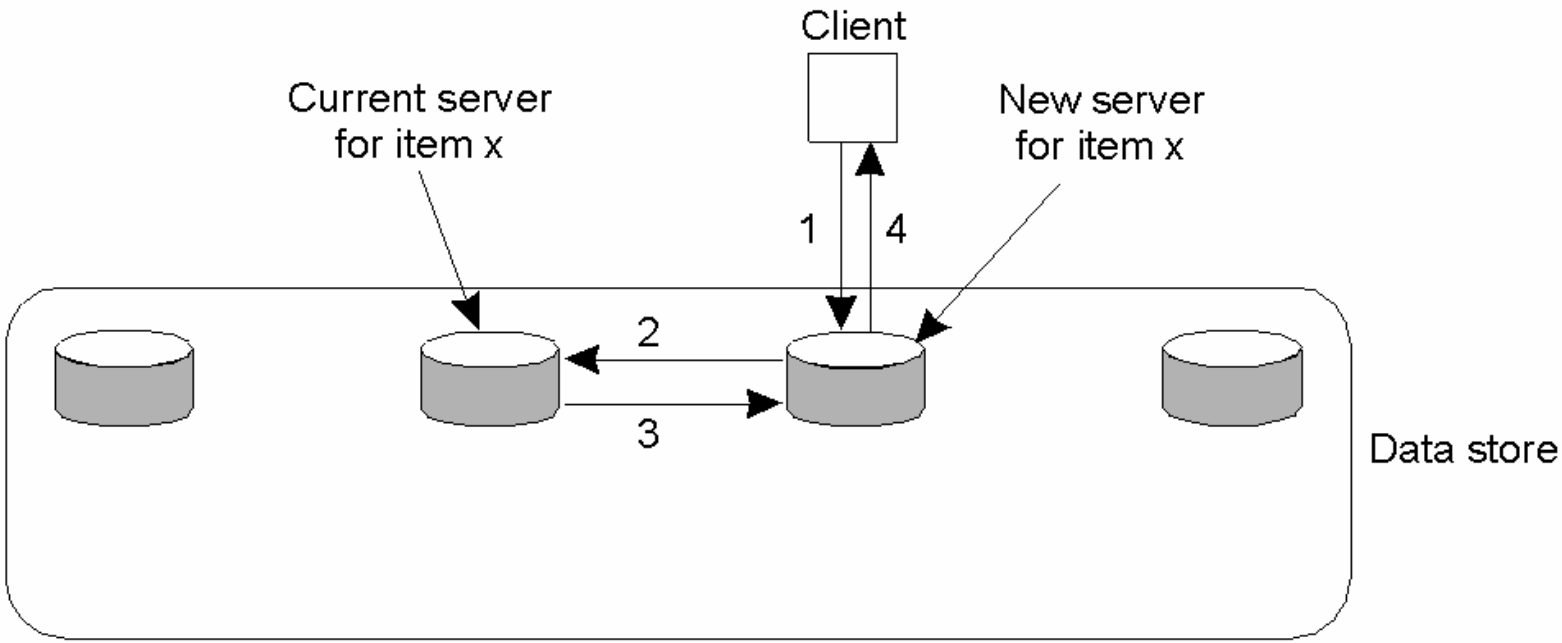
Primary backup



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

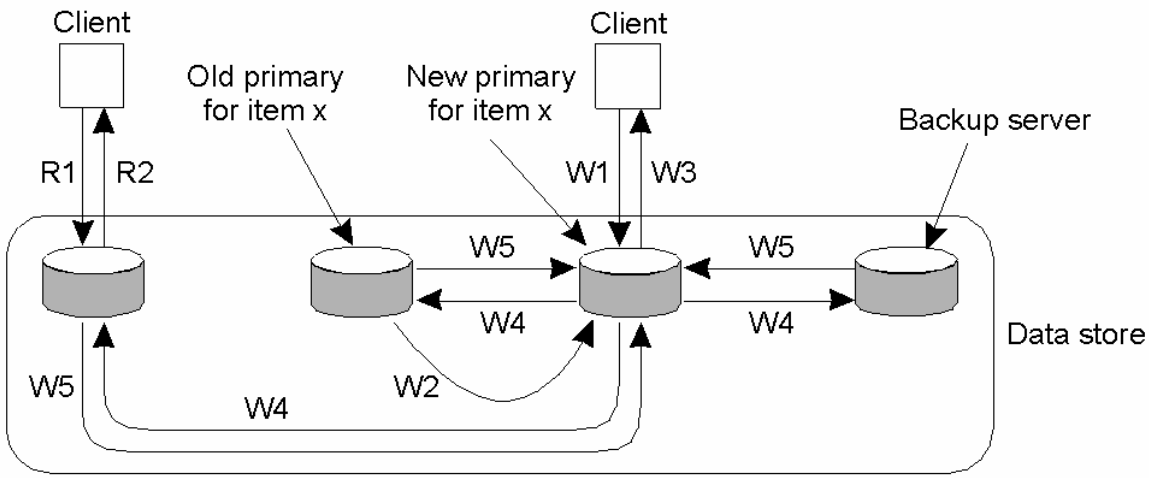
Local-write



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

Local-write

- Combine with primary backup & non-blocking updates
- Can use for mobile devices: postpone updates



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

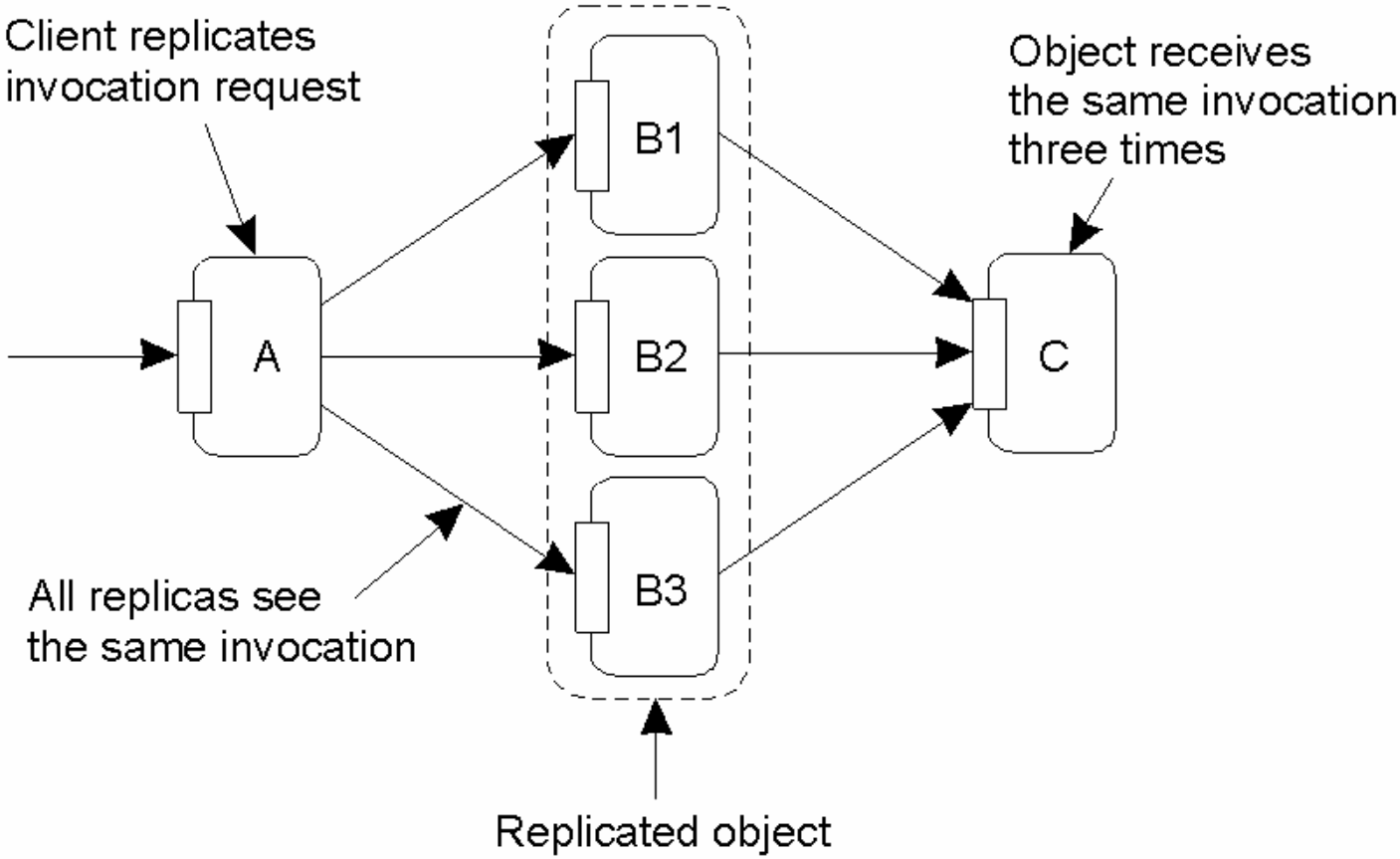
R1. Read request
R2. Response to read

Replicated write

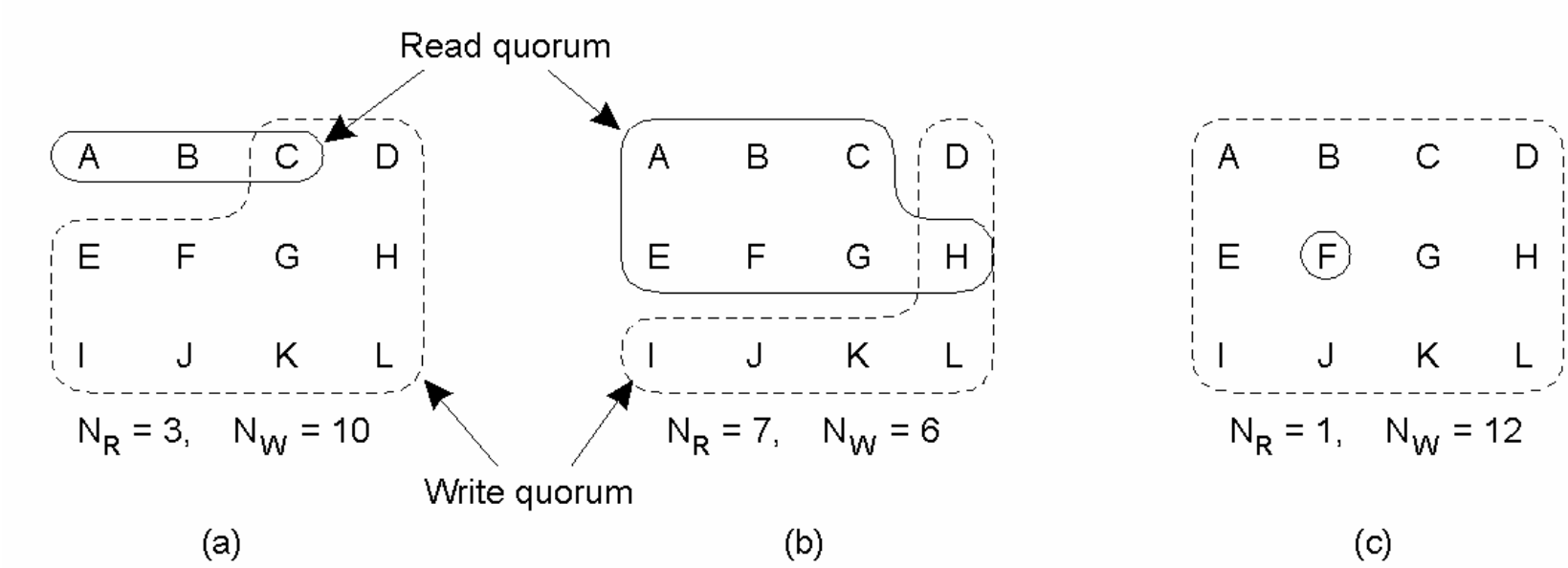
- Active replication
  - multiply the write itself rather than the update
    - need totally ordered multicast or sequencer
      - similar as primary-based
    - beware of events that result from these write's
      - these should not be multiplied

- Quorum-based: voting
  - obtain permission of a server majority to write a data item
    - read quorum N_R, write quorum N_W
    - $N_R+N_W>N$; $N_W>N/2$

Repeated invocation



Quorum-Based Protocols



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

- Client-control
- Two issues
  - coherence detection strategy
    - static: compiler
    - dynamic: check+blocking, check+optimistic, just optimistic
  - coherence enforcement strategy
    - send invalidation
    - propagate updates
    - write-through cf. primary based local write
    - write-back cf. distributed file systems