Distributed Systems
Consistency and Replication

[2] Consistency and Replication

Consistency and replication

1. Introduction
2. Data-centric consistency models
3. Client-centric consistency models
4. Consistency protocols

[3] Introduction

Two primary reasons for replicating data:

- **reliability** – to increase reliability of a system,
- **performance** – to scale in numbers and geographical area.

Reliability corresponds to fault tolerance, performance/scalability corresponds to high availability.

The cost of replication:

- modifications have to be carried on all copies to ensure **consistency**,
- *when* and *how* modifications need to be carried out, determines the price of replication.

[4] Performance and Scalability

Main issue: To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the the same order everywhere.

Conflicting operations:

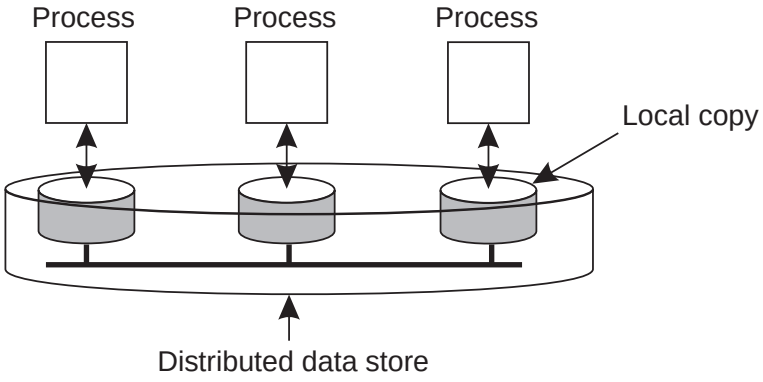
read–write conflict a read operation and a write operation act concurrently,

write–write conflict two concurrent write operations.

Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability.

Solution: weaken consistency requirements so that hopefully global synchronization can be avoided.

[5] Data-Centric Consistency Models (1)



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

Consistency model
A contract between a (distributed) data store and processes, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.

[6] Data-Centric Consistency Models (2)

Strong consistency models: Operations on shared data are synchronized:

- strict consistency (related to time),
- sequential consistency (what we are used to),
- causal consistency (maintains only causal relations),
- FIFO consistency (maintains only individual ordering).

Weak consistency models: Synchronization occurs only when shared data is locked and unlocked:

- general weak consistency,
- release consistency,
- entry consistency.

Observation: The weaker the consistency model, the easier it is to build a scalable solution.

[7] Strict Consistency

Strict consistency

> Any read to a shared data item X returns the value stored by the most recent write operation on X.

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

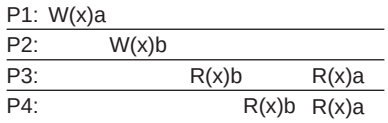
a. a strictly consistent store,

b. a store that is not strictly consistent.

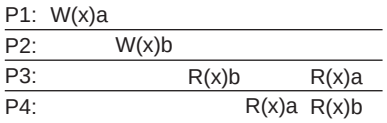
[8] Linearizability and Sequential Consistency (1)

Sequential Consistency

> The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program.



(a)



(b)

All processes should see the same interleaving of operations.

a. a sequentially consistent data store,

b. a data store that is not sequentially consistent.

[9] **Linearizability and Sequential Consistency (3)**

linearizable = sequential + operations ordered according to a global time.

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

```
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)

Four valid execution sequences for the presented processes. The vertical axis is time.

[10] **Causal Consistency (1)**

Causal consistency

> 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.

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 causally-consistent store, but not with sequentially or strictly consistent store.

[11] **Causal Consistency (2)**

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. a violation of a causally-consistent store,

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

[12] **FIFO Consistency (1)**

FIFO consistency

> 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.

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.

- **PRAM** consistency = pipelined RAM, writes from a single process can be pipelined,
- easy to implement by tagging each write operation with a (process, sequence number) pair.

[13] **FIFO Consistency (2)**

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

[14] **FIFO Consistency (3)**

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

Two concurrent processes.

Sequential vs. FIFO consistency:

- FIFO consistency: counterintuitive results – both processes can be killed,
- sequential consistency: none of interleavings results in both processes being killed,
- in sequential consistency, although the order is non-deterministic, at least all processes agree what it is. This is not the case in FIFO consistency.

[15] **Weak Consistency (1)**

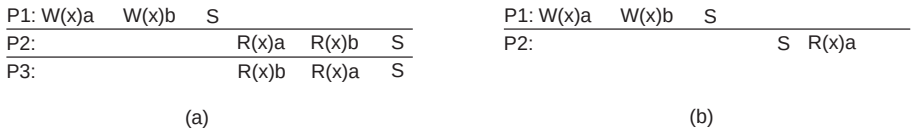
Weak consistency models

> Introduction of explicit synchronization variables. Changes of local replica content propagated only when an explicit synchronization takes place.

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.

[16] **Weak Consistency (2)**

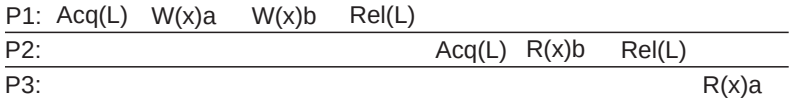


a. a valid sequence of events for weak consistency,

b. an invalid sequence for weak consistency.

Issue: The simplest method of weak consistency model implementation in case of replication with full replicas.

[17] **Release Consistency (1)**



A valid event sequence for release consistency.

[18] **Release Consistency (2)**

Release consistency properties:

- 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).

Additional issues:

- **lazy release consistency** versus eager release consistency,

– **barriers** instead of critical regions possible.

[19] Entry Consistency (1)

– with release consistency, all local updates are propagated to other copies/servers during release of shared data.

– with entry consistency, each shared data item is associated with a synchronization variable.

– when acquiring the synchronization variable, the most recent values of its associated shared data item are fetched.

Note: Where release consistency affects all shared data, entry consistency affects only those shared data associated with a synchronization variable.
Question: What would be a convenient way of making entry consistency more or less transparent to programmers?

[20] Entry Consistency (2)

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.

[21] Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses
Linearizability	All processes 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. Strong consistency models.

b. Weak consistency models.

[22] Client-Centric Consistency Models (1)

1. System model
2. Coherence models
   - monotonic reads,
   - monotonic writes,
   - read-your-writes,
   - write-follows-reads.

[23] Client-Centric Consistency Models (2)

Goal: Avoiding system-wide consistency, by concentrating on what specific clients want, instead of what should be maintained by servers.
Background: Most large-scale distributed systems (i.e., databases) apply replication for scalability, but can support only weak consistency:

DNS updates are propagated slowly, and inserts may not be immediately visible.

NEWS articles and reactions are pushed and pulled throughout the Internet, such that reactions can be seen before postings.

Lotus Notes geographically dispersed servers replicate documents, but make no attempt to keep (concurrent) updates mutually consistent.

WWW caches all over the place, but there need be no guarantee that you are reading the most recent version of a page.

[24] Consistency for Mobile Users

Example: Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.

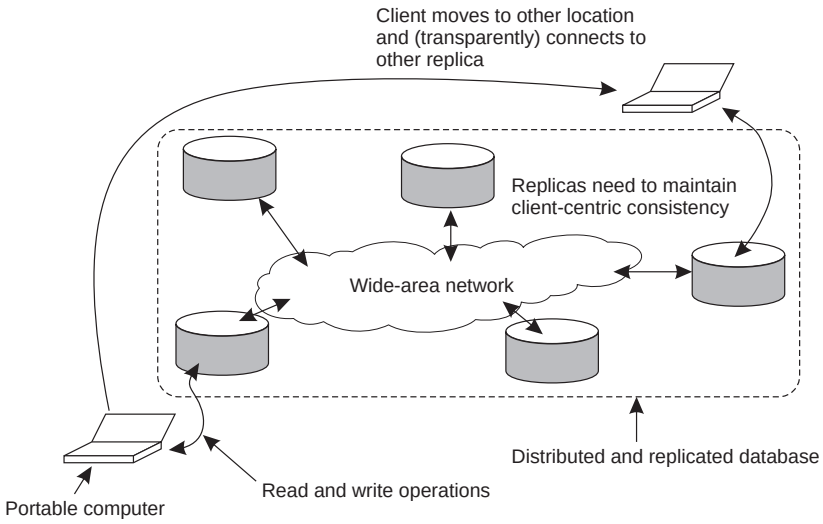
- at location A you access the database doing reads and updates.
- at location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies:
  - your updates at A may not have yet been propagated to B
  - you may be reading newer entries than the ones available at A
  - your updates at B may eventually conflict with those at A

Note: The only thing you really want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will appear to be consistent to you.

[25] Eventual Consistency

Eventual consistency

> Consistency model in large-scale distributed replicated databases that tolerate a relatively high degree of inconsistency. If no updates take place for a long time, all replicas gradually becomes consistent.

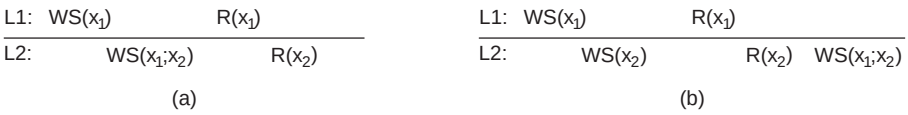


The principle of a mobile user accessing different replicas of a distributed database.

[26] **Monotonic Reads (1)**

Monotonic reads

> If a process reads the value of a data item x, any successive read operation on x by that process will always return that same or a more recent value.



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.

[27] **Monotonic Reads (2)**

Example

Automatically reading your personal calendar updates from different servers. Monotonic reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

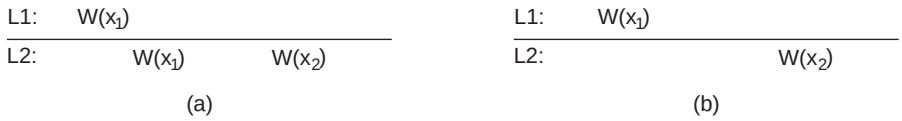
Example

Reading (not modifying) incoming mail while you are on the move. Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

[28] **Monotonic Writes (1)**

Monotonic writes

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

[29] **Monotonic Writes (2)**

Example

Updating a program at server S2, and ensuring that all components on which compilation and linking depends, are also placed at S2.

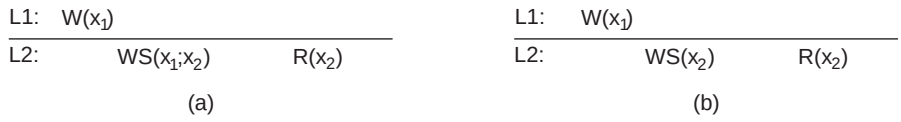
Example

Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

[30] **Read Your Writes**

Read your writes

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.



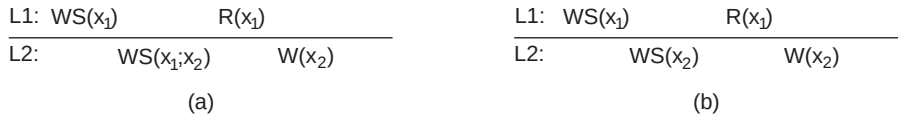
a. a data store that provides read-your-writes consistency.

b. a data store that does not.

[31] **Writes Follow Reads**

Writes follow reads

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.



a. a writes-follow-reads consistent data store,

b. a data store that does not provide writes-follow-reads consistency.

[32] **Examples**

Read-your-writes example

Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

Writes-follow-reads example

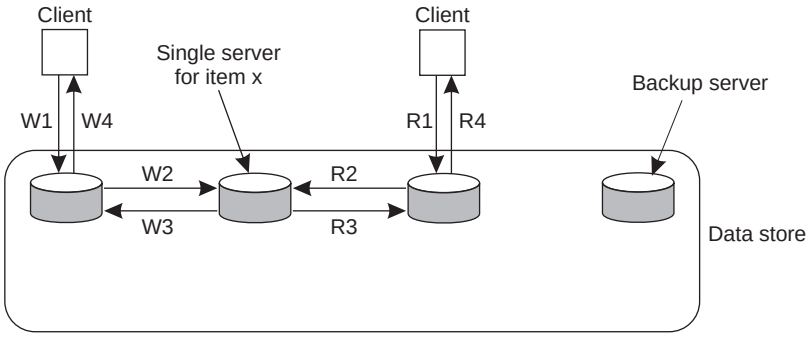
> See reactions to posted articles only if you have the original posting (a read "pulls in" the corresponding write operation).

[33] **Consistency Protocols**

Consistency protocol
Describes the implementation of a specific consistency model. We will concentrate only on sequential consistency.

- Primary-based protocols
  - remote-write protocols,
  - local-write protocols.
- Replicated-write protocols
  - active replication,
  - quorum-based protocols.
- Cache-coherence protocols (write-through, write-back)

[34] **Remote-Write Protocols (1)**

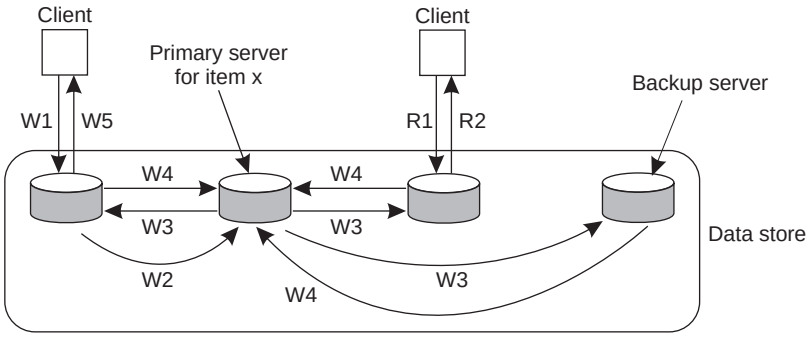


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

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.

[35] **Remote-Write Protocols (2)**

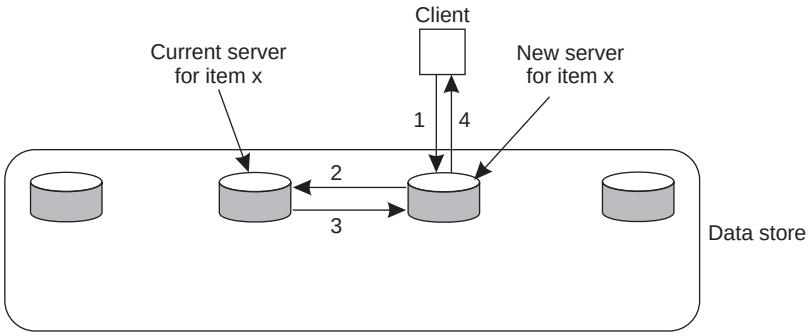


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

The principle of **primary-backup protocol**: read operations allowed on a locally available copy, write operations forwarded to a fixed primary copy.

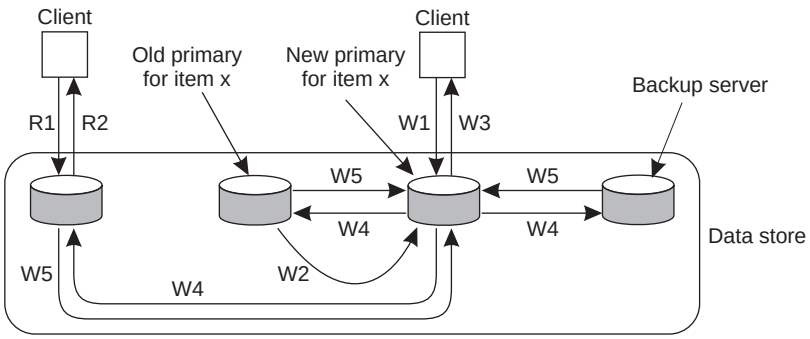
[36] **Local-Write Protocols (1)**



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.

[37] **Local-Write Protocols (2)**

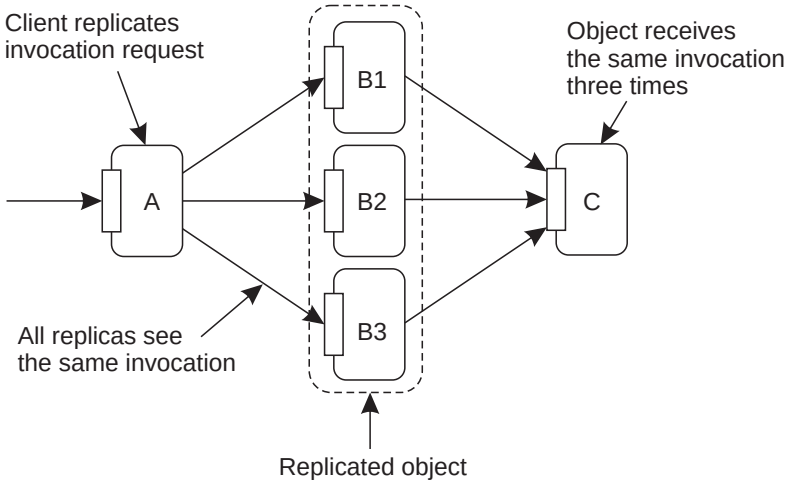


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

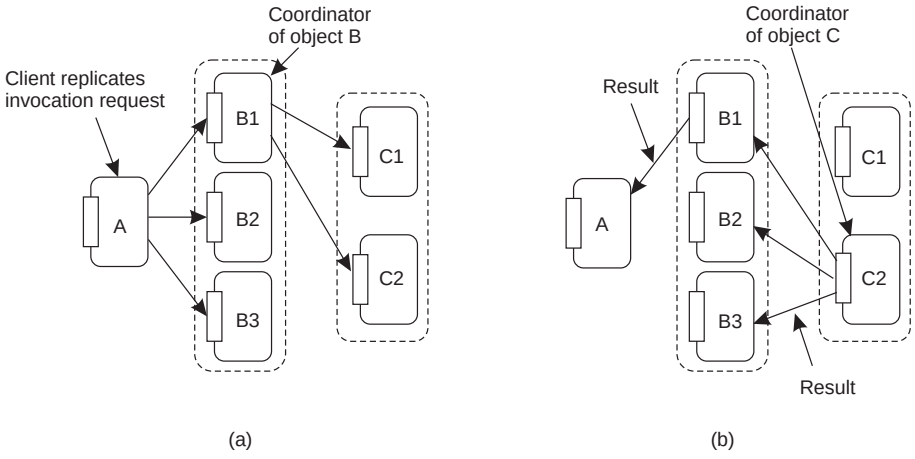
Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

[38] **Active Replication (1)**



The problem of replicated invocations.

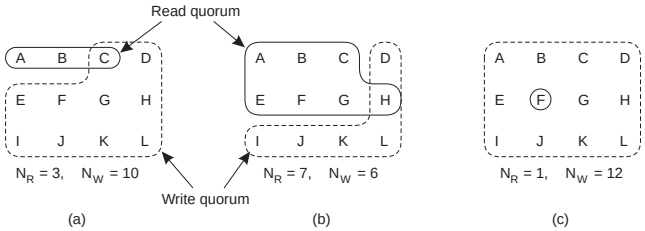
[39] **Active Replication (2)**



a. forwarding an invocation request from a replicated object,

b. returning a reply to a replicated object.

[40] **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).

Constraints: $N_R + N_W > N$ and $N_W > N/2$

[41] **Cache-Coherence Protocols**

Cache coherence strategies:

- coherence detection strategy - *when* inconsistencies are detected,
- coherent enforcement strategy - *how* caches are kept consistent with the copies stored at servers.

When processes modify data:

- read-only cache - updates can be performed only by servers,
- write-through cache - clients directly modify cached data and forward updates to servers,
- write-back cache - propagation of updates may be delayed by allowing multiple writes to take place before informing servers.