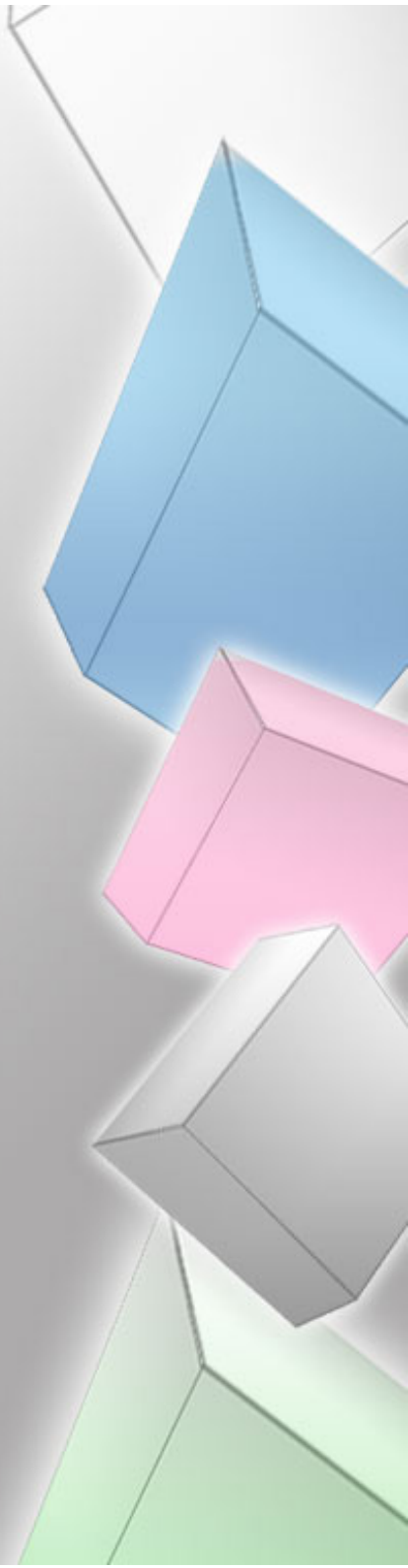
Advanced Topics in Operating Systems

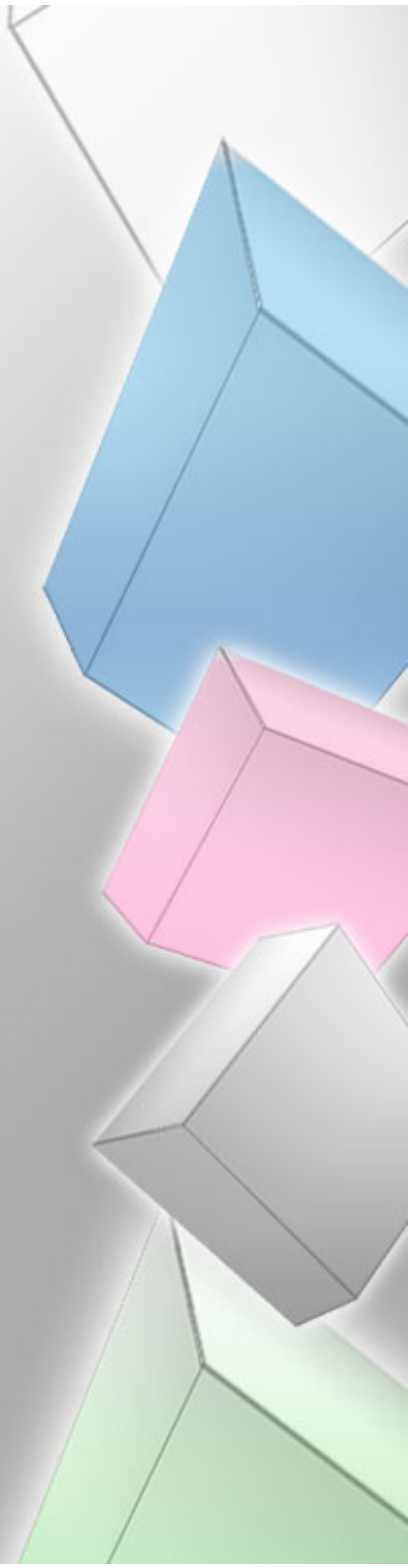
MSc in Computer Science
UNYT-UoG

Dr. Marenglen Biba
8-9-10 January 2010



Lesson 7

01: Introduction
02: Architectures
03: Processes
04: Communication
05: Naming
06: Synchronization
07: Consistency & Replication
08: Fault Tolerance
09: Security
10: Distributed Object-Based Systems
11: Distributed File Systems
12: Distributed Web-Based Systems
13: Distributed Coordination-Based Systems



Reasons for Replication

- Data are replicated to increase the reliability of a system.
- Replication for performance
  - Scaling in numbers
  - Scaling in geographical area
    - Place copies near the process that is using them
- Important features
  - Gain in performance
    - The process near the copy perceives good performance
  - Cost of increased bandwidth for maintaining replication (updating all the replicas!!!)
- Price to be paid: Consistency

Access-to-update ratio

- Consider a process *P* that accesses a local replica *N* times per second, whereas the replica itself is updated *M* times per second.
- Assume that an update completely refreshes the previous version of the local replica. If $N < M$, that is, the access-to-update ratio is very low, we have the situation where many updated versions of the local replica will never be accessed by *P*, rendering the network communication for those versions useless.

Consistency

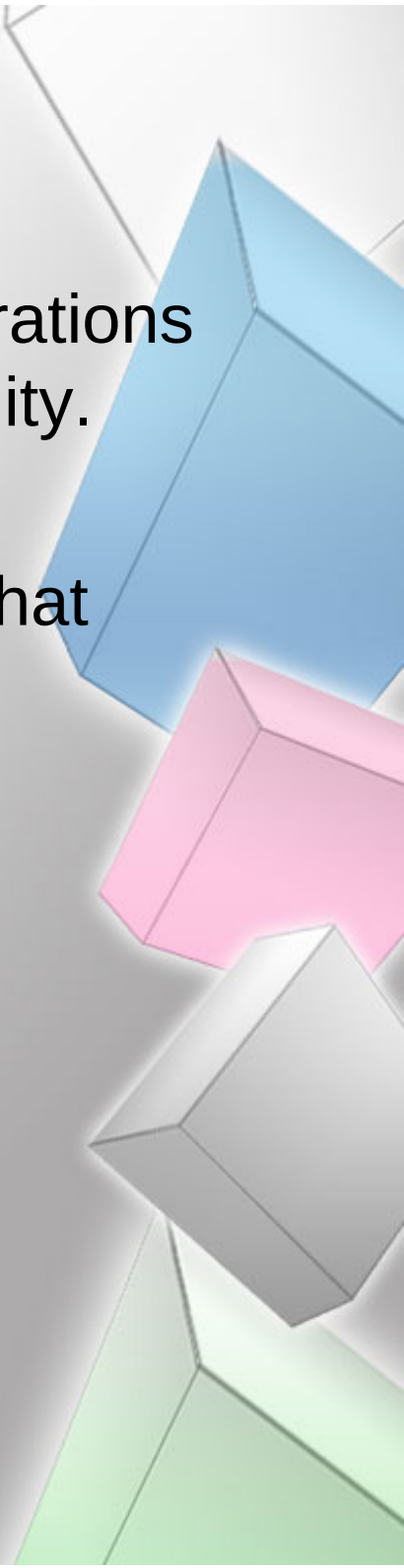
- A collection of copies is consistent when the copies are always the same.
- This means that a read operation performed at any copy will always return the same result.
- Consequently, when an update operation is performed on one copy, the update should be propagated to all copies before a subsequent operation takes place, no matter at which copy that operation is initiated or performed.

- Update as single atomic operation or transaction.

Softening Constraints

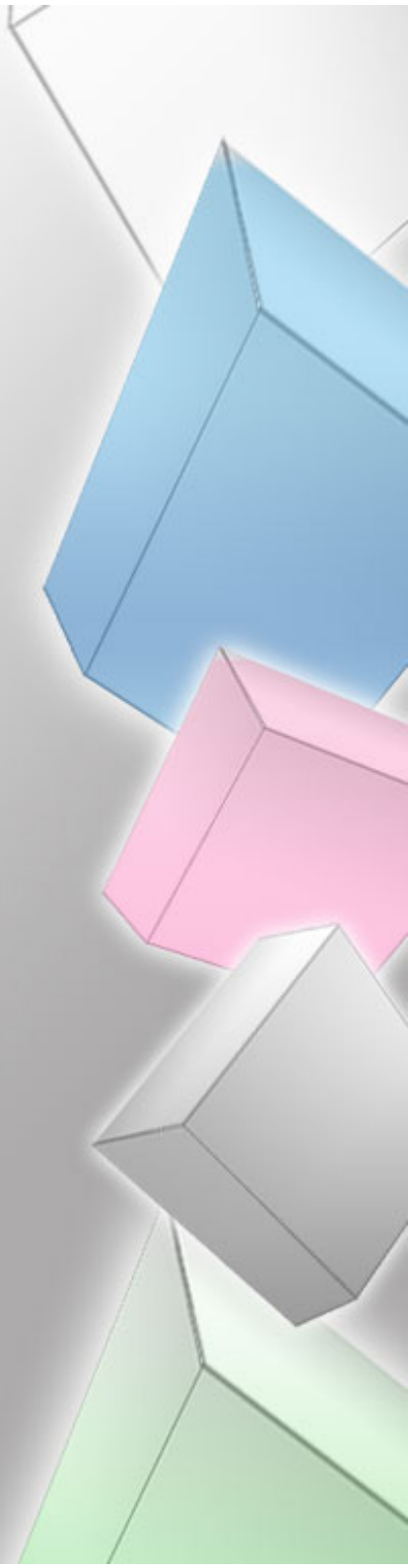
- Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability.

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

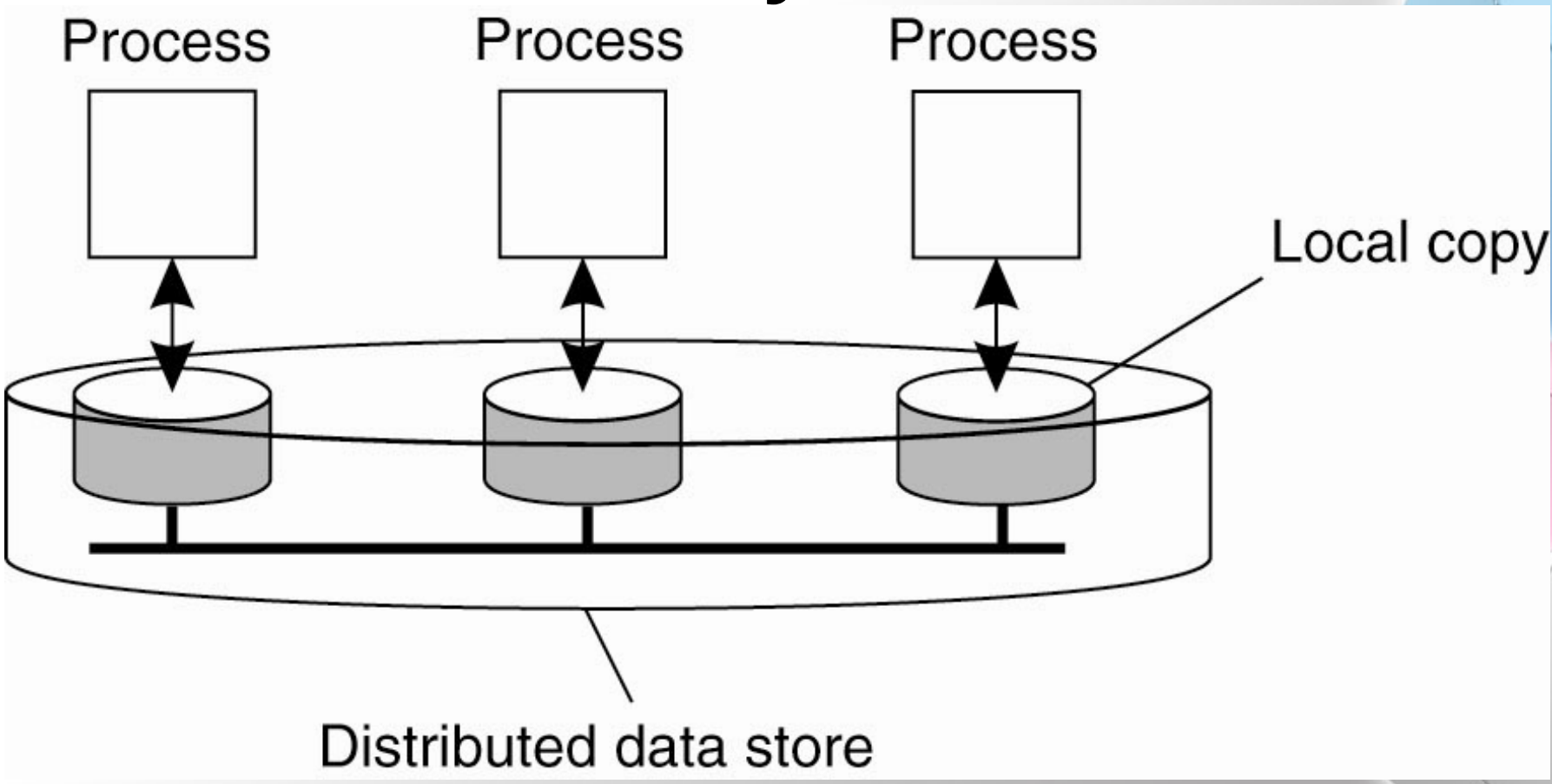


Content

- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols



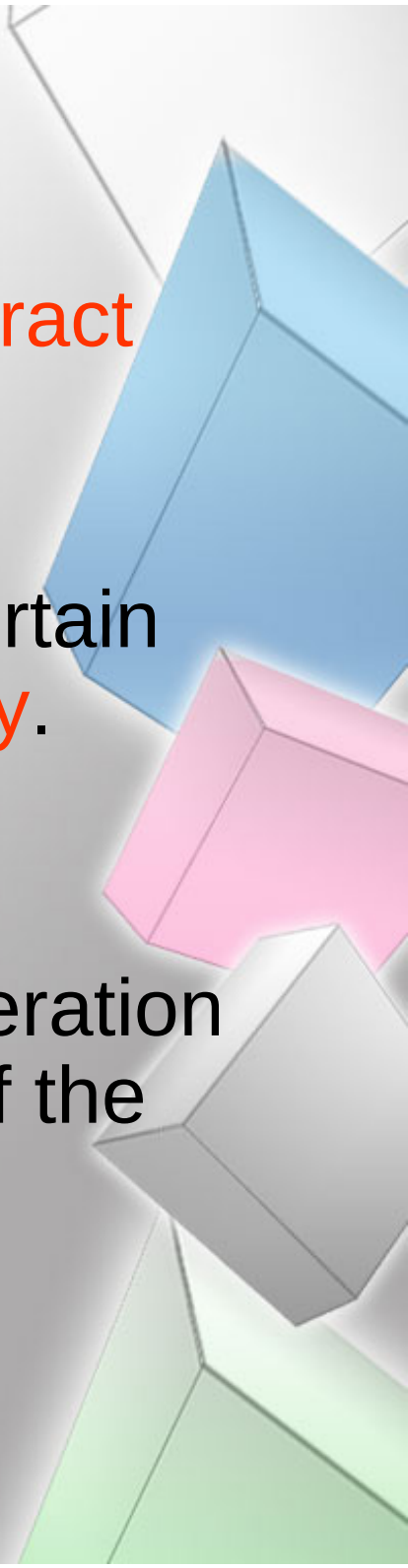
Data-centric Consistency Models



- Figure 7-1. The general organization of a logical data store, physically distributed and replicated across multiple processes.

Consistency Model

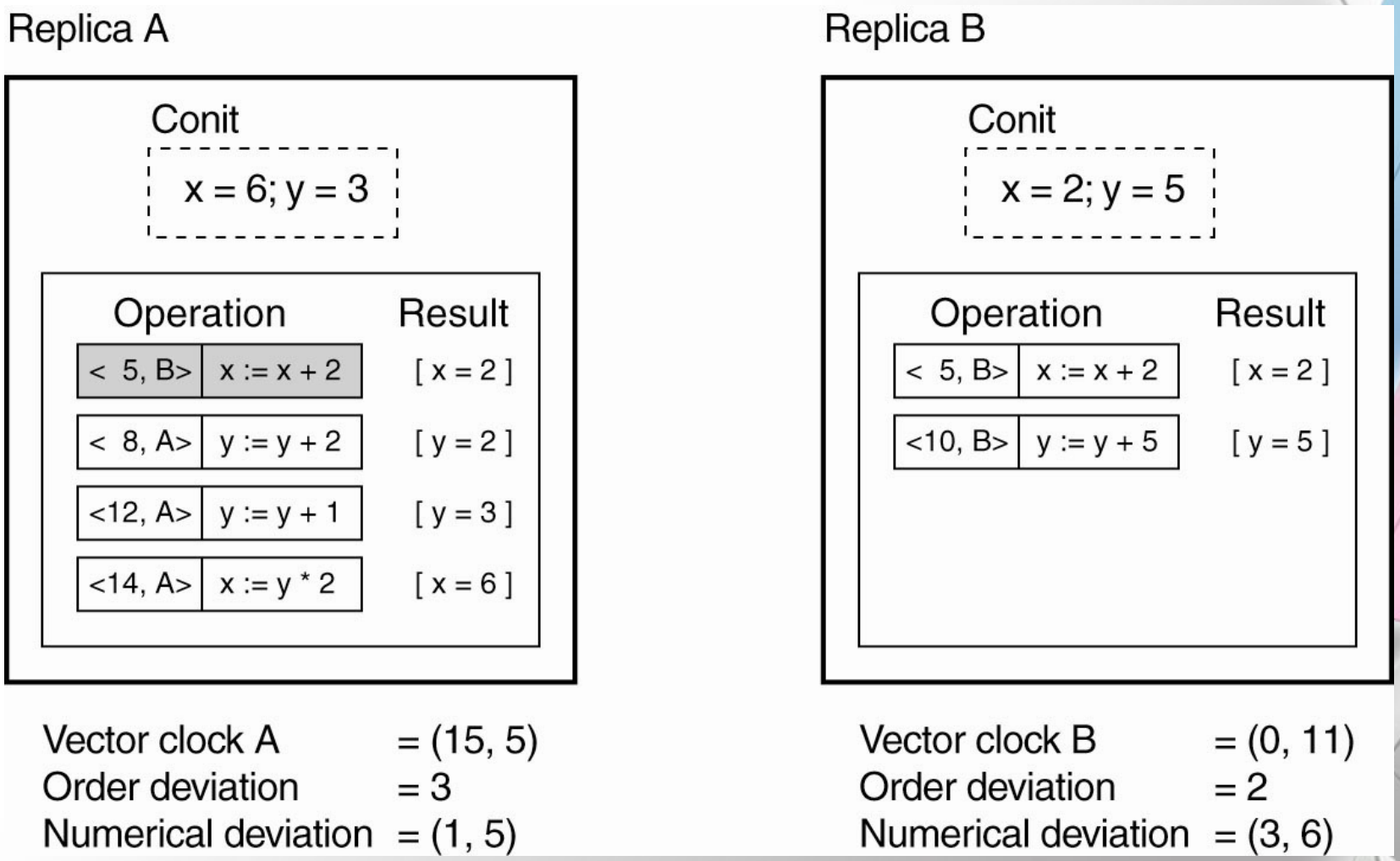
- A consistency model is essentially a contract between processes and the data store.
- It says that if processes agree to obey certain rules, the store promises to work correctly.
- Normally, a process that performs a read operation on a data item, expects the operation to return a value that shows the results of the last write operation on that data.



Continuous Consistency

- Degree of consistency:
  - Replicas may differ in their numerical value
  - Replicas may differ in their relative staleness
  - There may be differences with respect to (number and order) of performed update operations
- These deviations form continuous consistency ranges.
- **Conit**
  - Consistency unit => specifies the data unit over which consistency is to be measured.

Continuous Consistency



- Figure 7-2. An example of keeping track of consistency deviations [adapted from (Yu and Vahdat, 2002)].

Sequential consistency

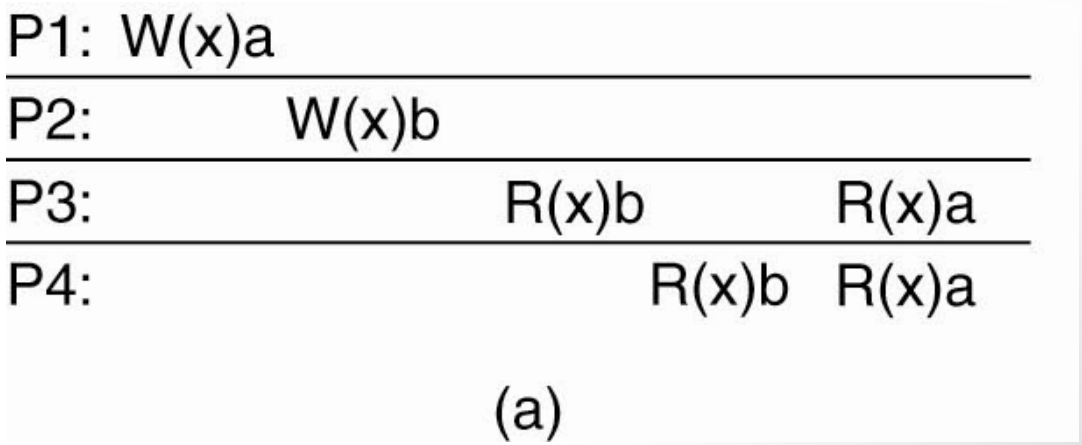
A data store is sequentially consistent when:

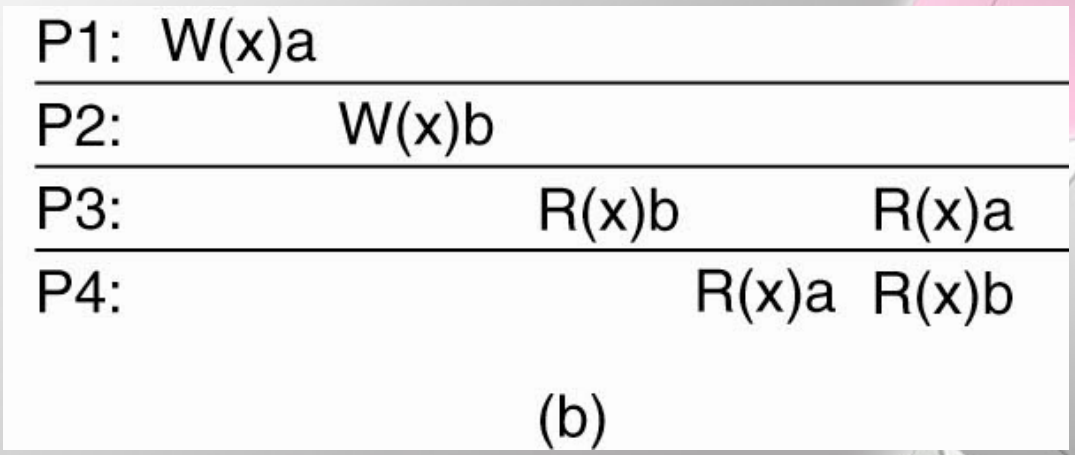
The result of any execution is the same as if the (read and write) operations by all processes on the data store …

- 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 other words all processes see the same interleaving of write operations.

Sequential Consistency





- Figure 7-5. (a) A sequentially consistent data store. (b) A data store that is not sequentially consistent.

Causal Consistency

- The causal consistency model represents a weakening of sequential consistency in that it makes a distinction between events that are potentially causally related and those that are not.
- We already came across causality when discussing vector timestamps.
  - If event *b* is caused or influenced by an earlier event *a*, causality requires that everyone else first see *a*, then see *b*.

Causal consistency (1)

For a data store to be considered causally consistent, it is necessary that the store obeys the following condition:

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 spontaneously and simultaneously write two different data items, these are not causally related.
- Operations that are not causally related are said to be **concurrent**.

Causal Consistency (2)

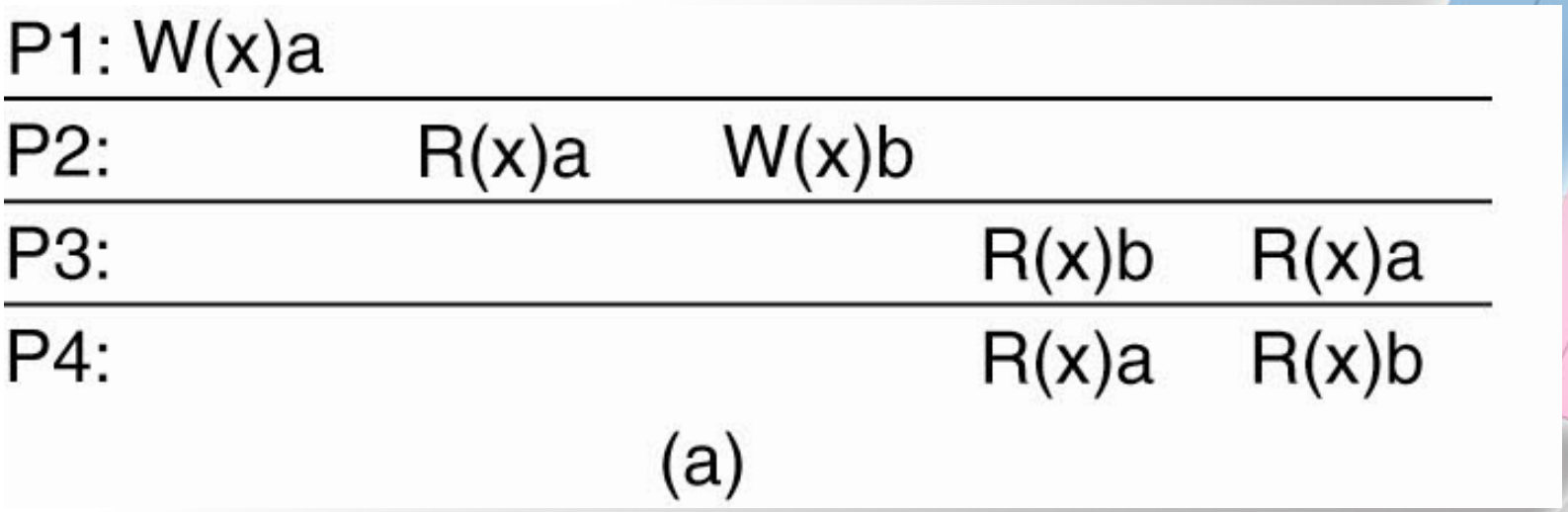
Writes $W_2(x)b$ and $W_1(x)c$ are concurrent, so it is not required that all processes see them in the same order

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

- Figure 7-8. This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store.

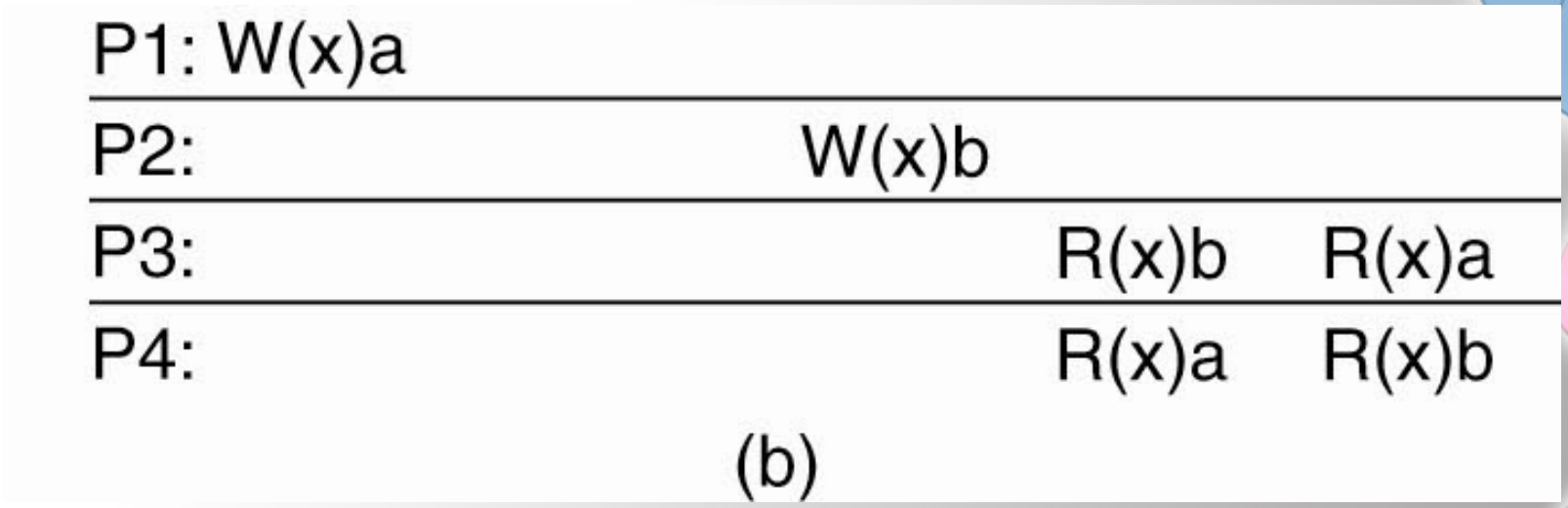
Causal Consistency (3)

- Figure 7-9. (a) A violation of a causally-consistent store.



$W_2(x)b$ potentially depending on $W_1(x)a$ because the *b* may be a result of a computation involving the value read by *R(x)a.* The two writes are causally related, so all processes must see them in the same order.

Causal Consistency (4)



(b)

- Figure 7-9. (b) A correct sequence of events in a causally-consistent store.

Grouping Operations (1)

- Critical Section
  - ENTER_CS and LEAVE_CS

- **Synchronization variables**
  - Each synchronization variable has a current owner, namely, the process that last acquired it. The owner may enter and exit critical sections repeatedly without having to send any messages on the network.
  - A process not currently owning a synchronization variable but wanting to acquire it has to send a message to the current owner asking for ownership and the current values of the data associated with that synchronization variable.

Grouping Operations (2)

Necessary criteria for correct synchronization:

- An acquire access of a synchronization variable, is not allowed to perform until all updates to 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' next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.

Grouping Operations (3)

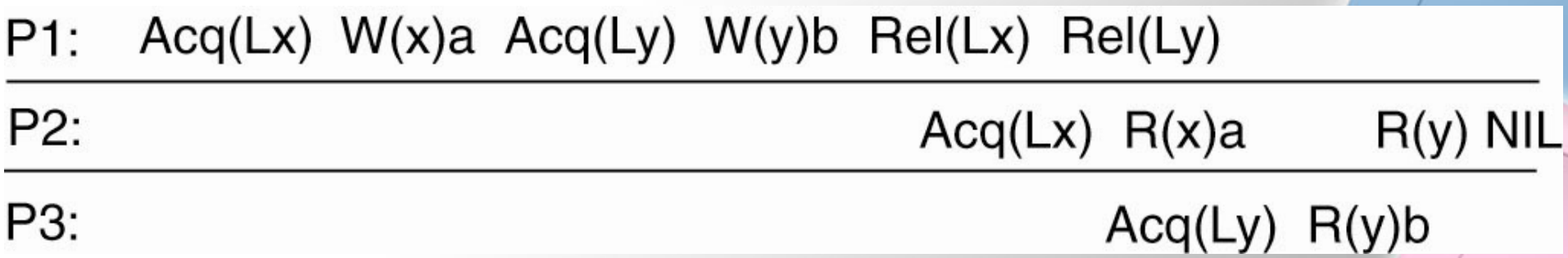


Figure 7-10. A valid event sequence for entry consistency.

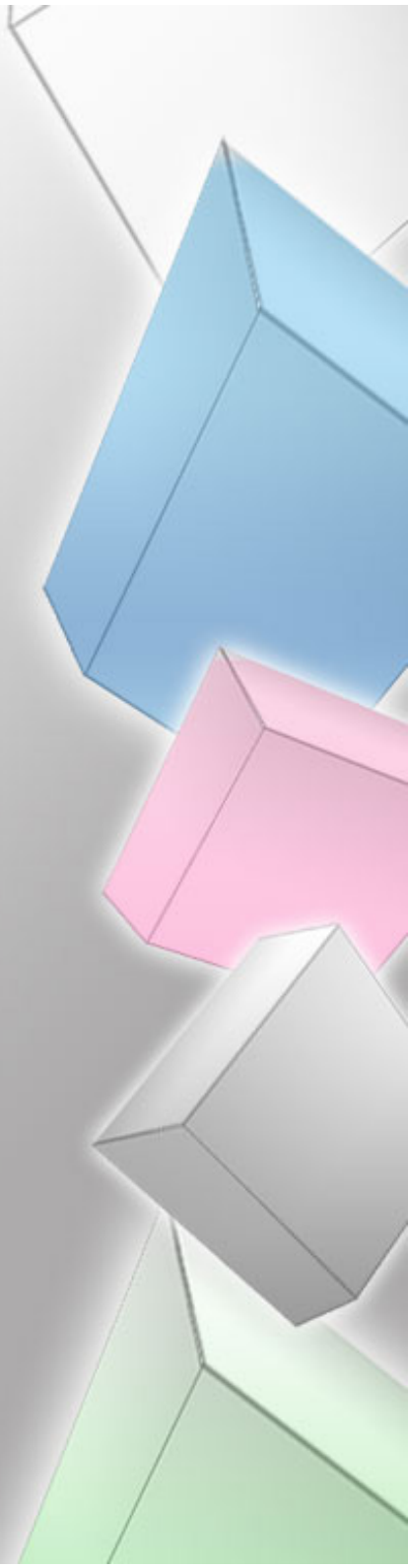
Consistency Vs. Coherence

- The models discussed so far all deal with the fact that a number of processes execute read and write operations on a set of data items.
  - A consistency model describes what can be expected with respect to that set when multiple processes concurrently operate on that data. The set is then said to be consistent if it adheres to the rules described by the model.

- While data consistency is concerned with a set of data items, coherence models describe what can be expected to only a single data item. In this case, we assume that a data item is replicated at several places;
  - It is said to be coherent when the various copies abide to the rules as defined by its associated coherence model.

Content

- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols



Eventual Consistency

- In many cases of (large-scale) distributed and replicated databases, these tolerate a relatively high degree of inconsistency.
- They have in common that if no updates take place for a long time, all replicas will gradually become consistent. This form of consistency is called **eventual consistency**.

Client-centric consistency

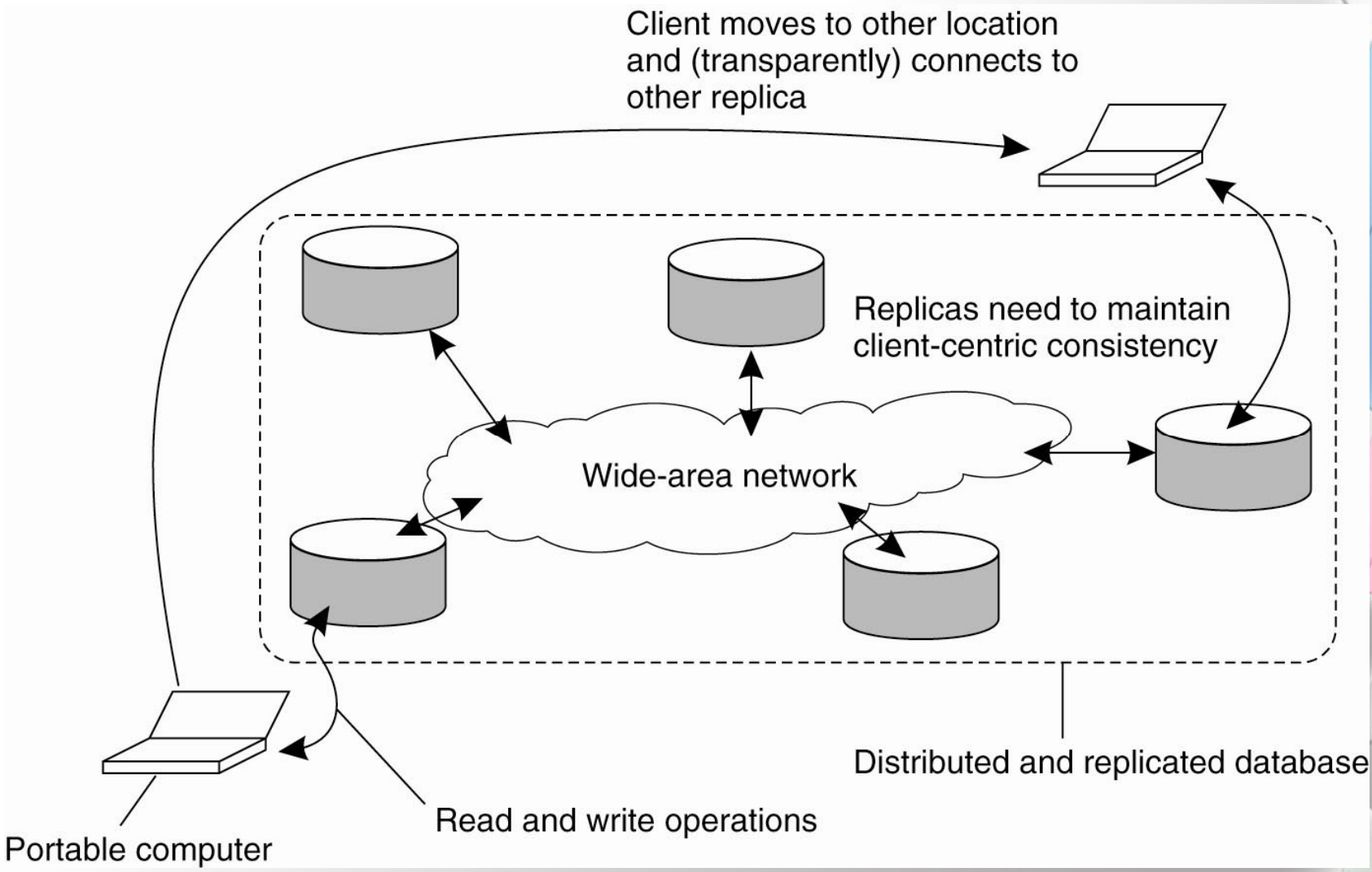


Figure 7-11. The principle of a mobile user accessing different replicas of a distributed database.

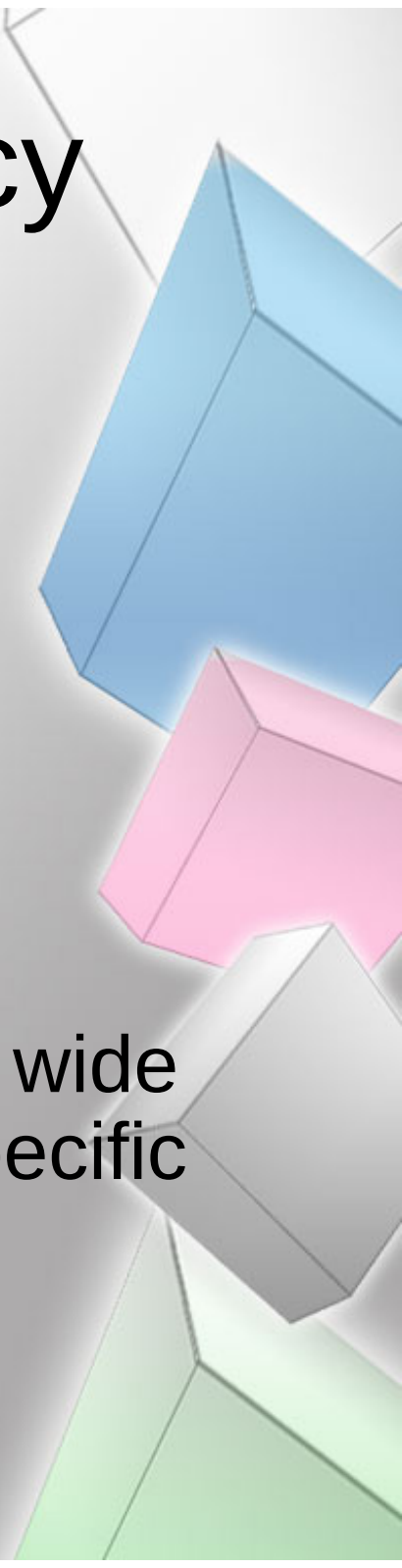
Client-centric consistency

Overview

- System model
- Monotonic reads
- Monotonic writes
- Read-your-writes
- Write-follows-reads

Goal

- Show how we can perhaps avoid system wide consistency, by concentrating on what specific clients want, instead of what should be maintained by servers.



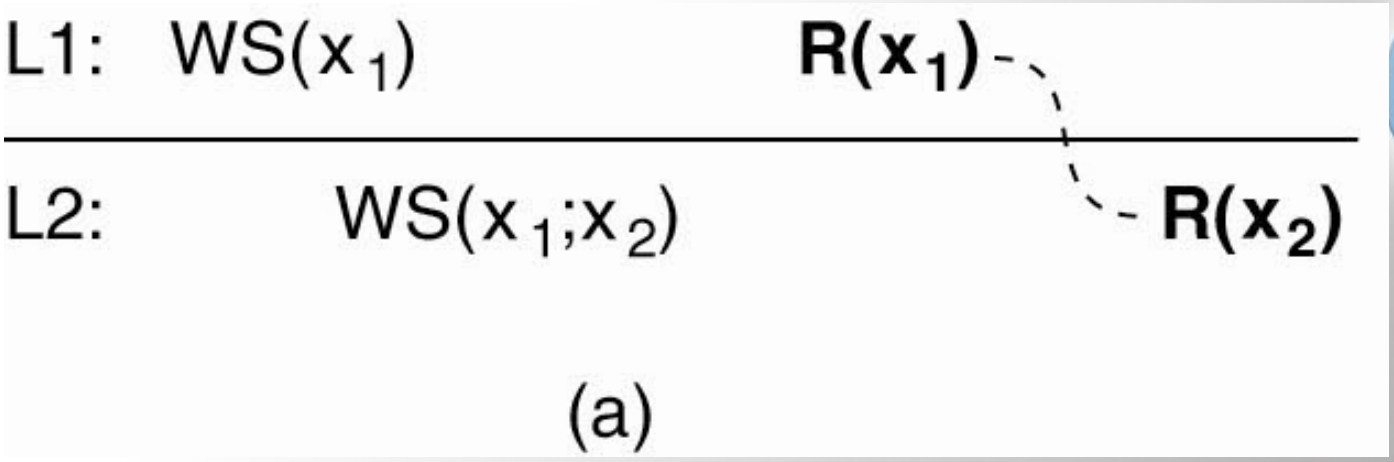
Monotonic Reads (1)

A data store is said to provide monotonic-read consistency if the following condition holds:

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 value
- or a more recent value.

Monotonic Reads (2)



- Figure 7-12. 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.

Monotonic Reads (3)

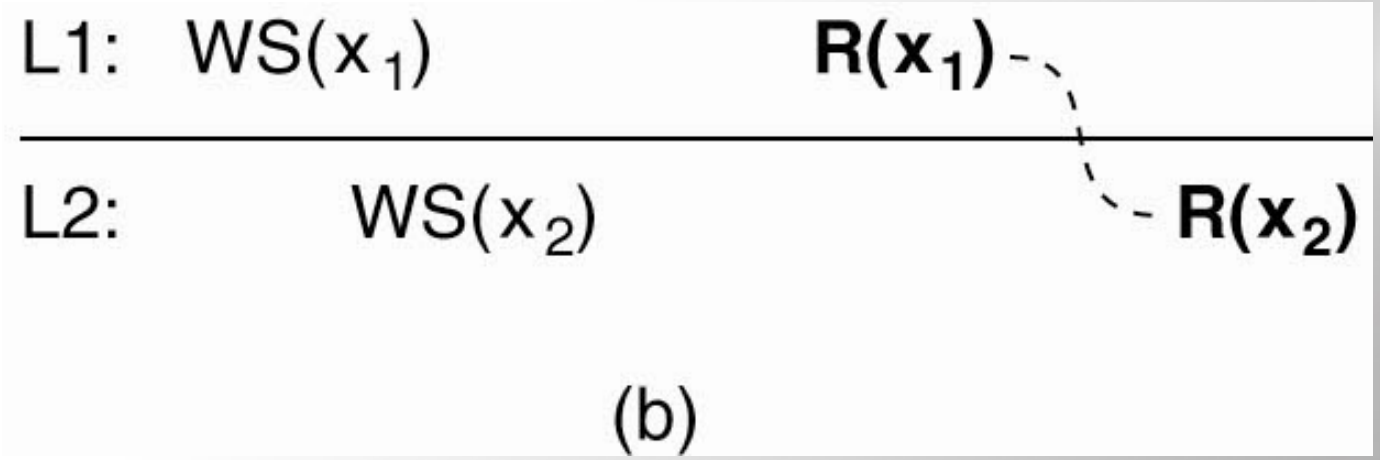


Figure 7-12. The read operations performed by a single process P at two different local copies of the same data store. (b) A data store that does not provide monotonic reads.

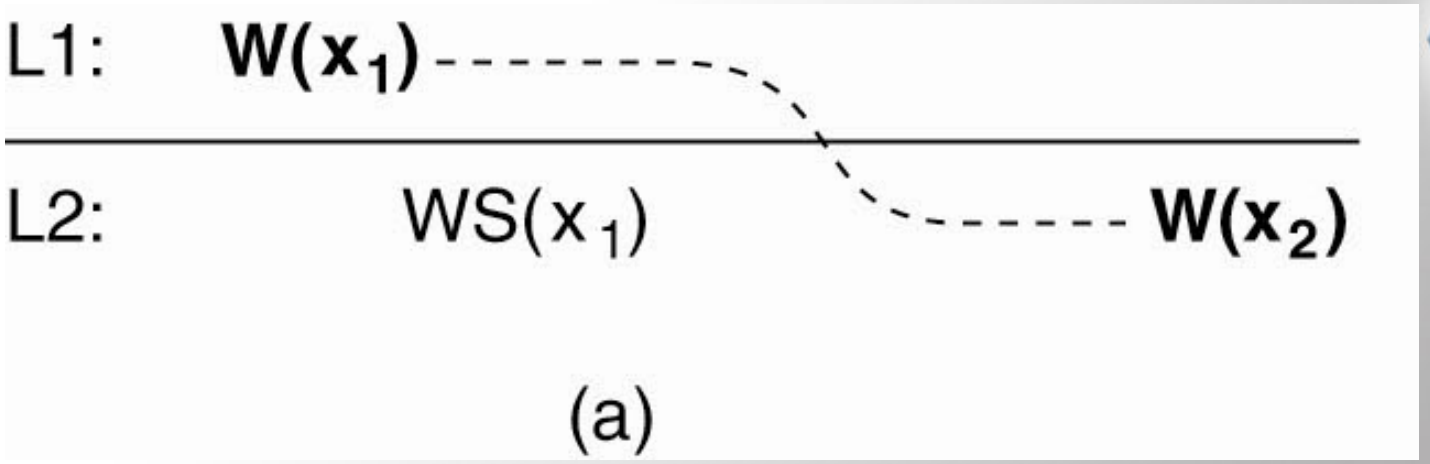
Monotonic Writes (1)

In a monotonic-write consistent store, the following condition holds:

A write operation by a process on a data item x …

- is completed before any successive write operation on x
- by the same process.

Monotonic Writes (2)



- Figure 7-13. 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.

Monotonic Writes (3)

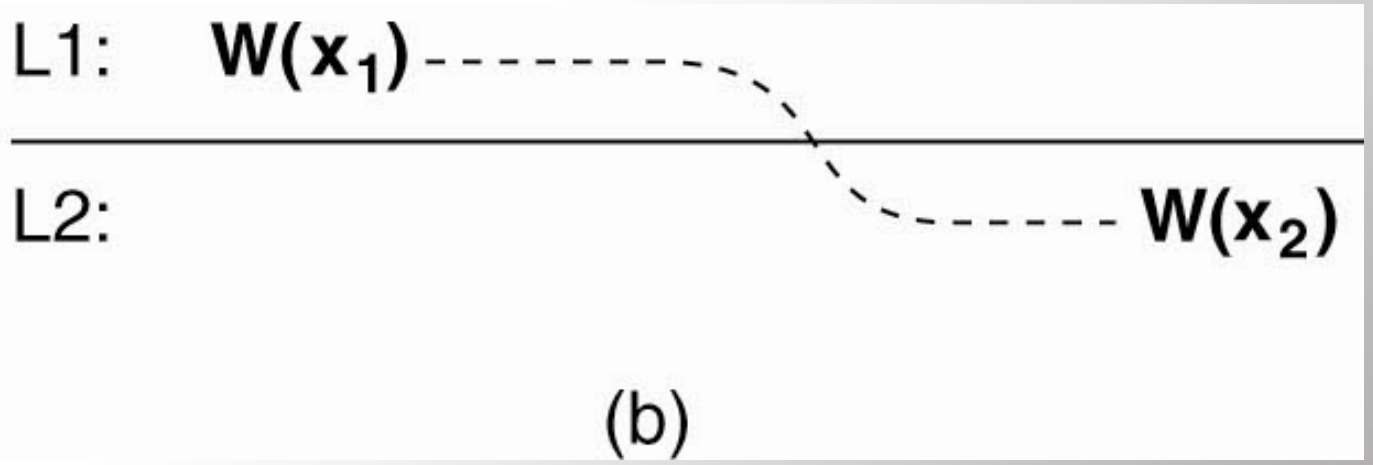


Figure 7-13. The write operations performed by a single process P at two different local copies of the same data store. (b) A data store that does not provide monotonic-write consistency.

Read Your Writes (1)

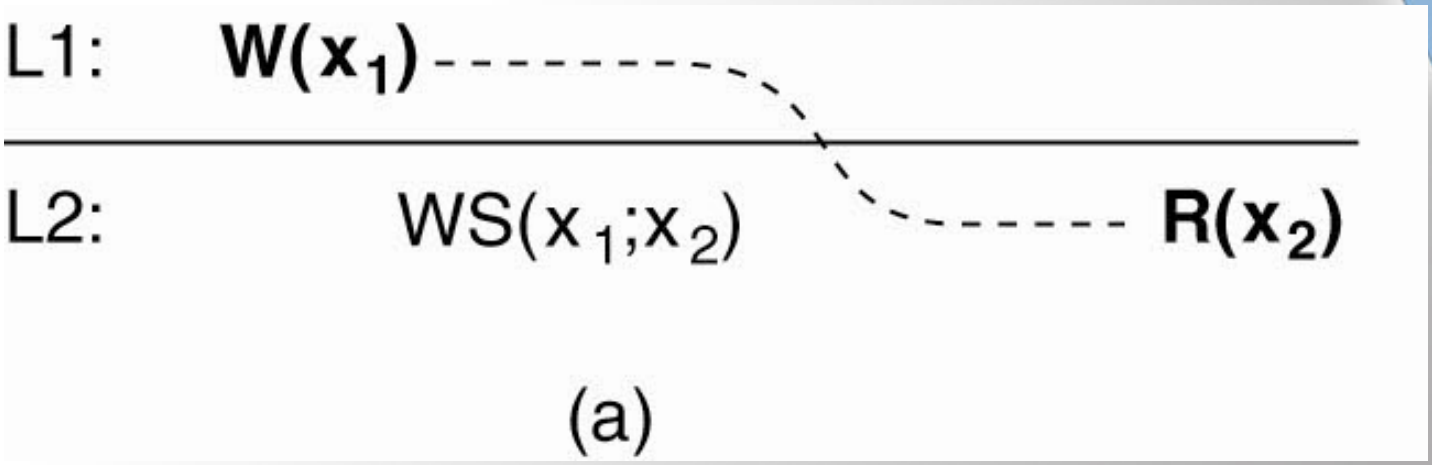
A data store is said to provide read-your-writes consistency, if the following condition holds:

The effect of a write operation by a process on data item x …

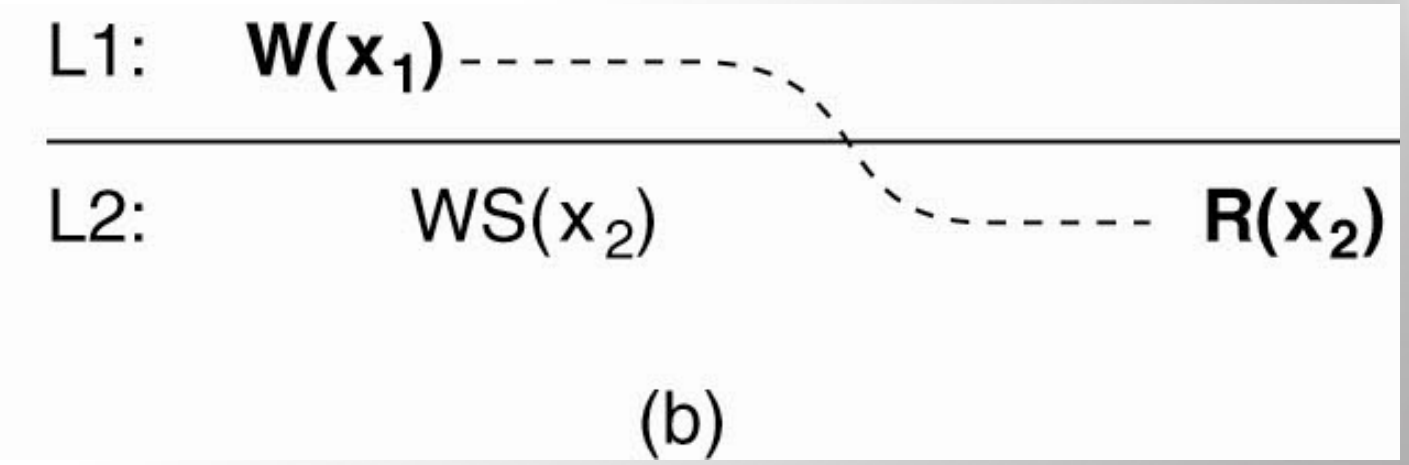
- will always be seen by a successive read operation on x
- by the same process.

Read Your Writes (2)

- Figure 7-14. (a) A data store that provides read-your-writes consistency.



Read Your Writes (3)



(b)

- Figure 7-14. (b) A data store that does not provide read-your-writes consistency. .

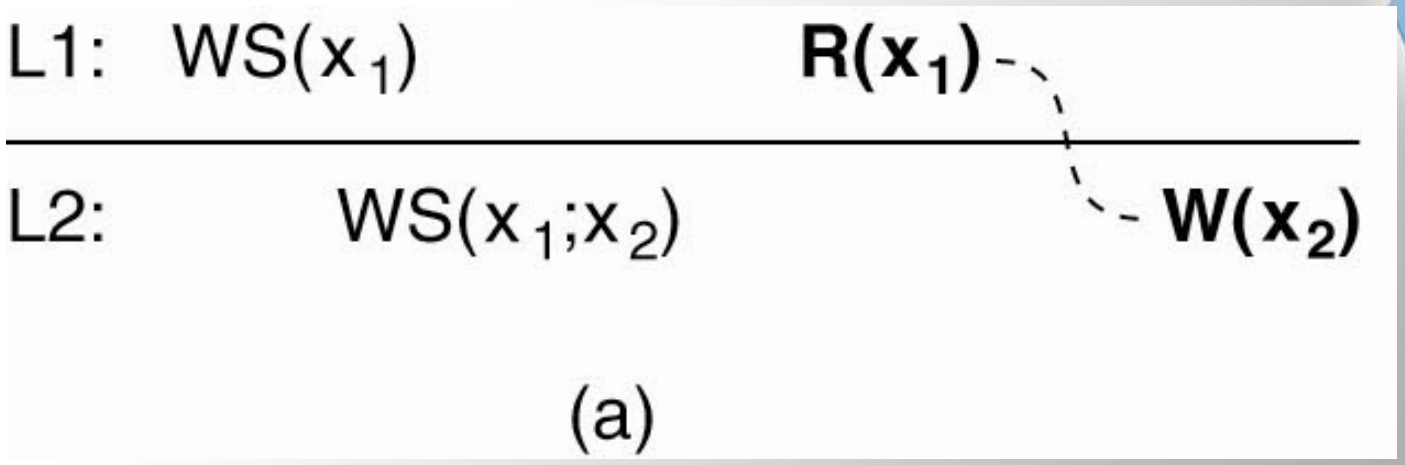
Writes Follow Reads (1)

A data store is said to provide writes-follow-reads consistency, if the following holds:

A write operation by a process …

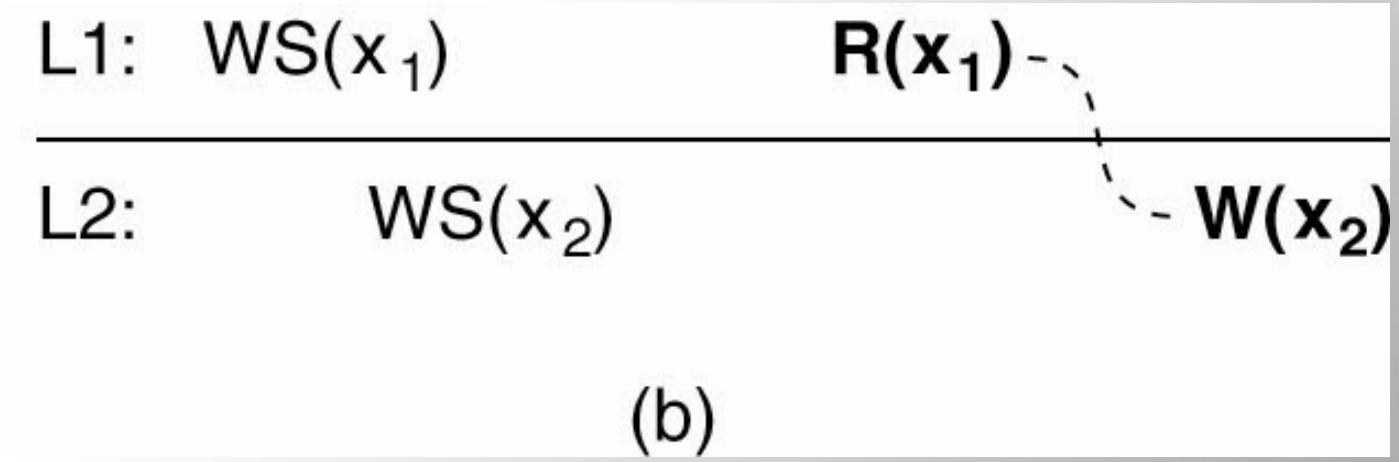
- on a data item x following a previous read operation on x by the same process …
- is guaranteed to take place on the same or a more recent value of x that was read.

Writes Follow Reads (2)



- Figure 7-15. (a) A writes-follow-reads consistent data store.

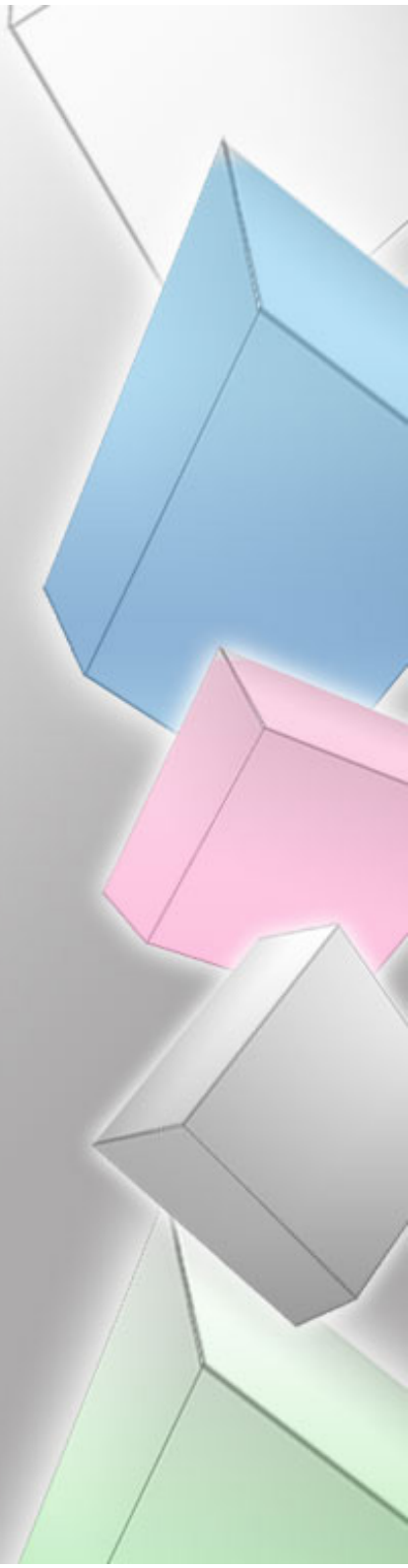
Writes Follow Reads (3)



- Figure 7-15. (b) A data store that does not provide writes-follow-reads consistency.

Content

- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols



Replica Management

- A key issue for any distributed system that supports replication is to decide where, when, and by whom replicas should be placed, and subsequently which mechanisms to use for keeping the replicas consistent.
- The placement problem itself should be split into two subproblems: that of placing *replica servers,* and that of placing *content.*
- The difference is a subtle but important one and the two issues are often not clearly separated.
  - Replica-server placement is concerned with finding the best locations to place a server that can host (part of) a data store.
  - Content placement deals with finding the best servers for placing content.

Replica-Server Placement

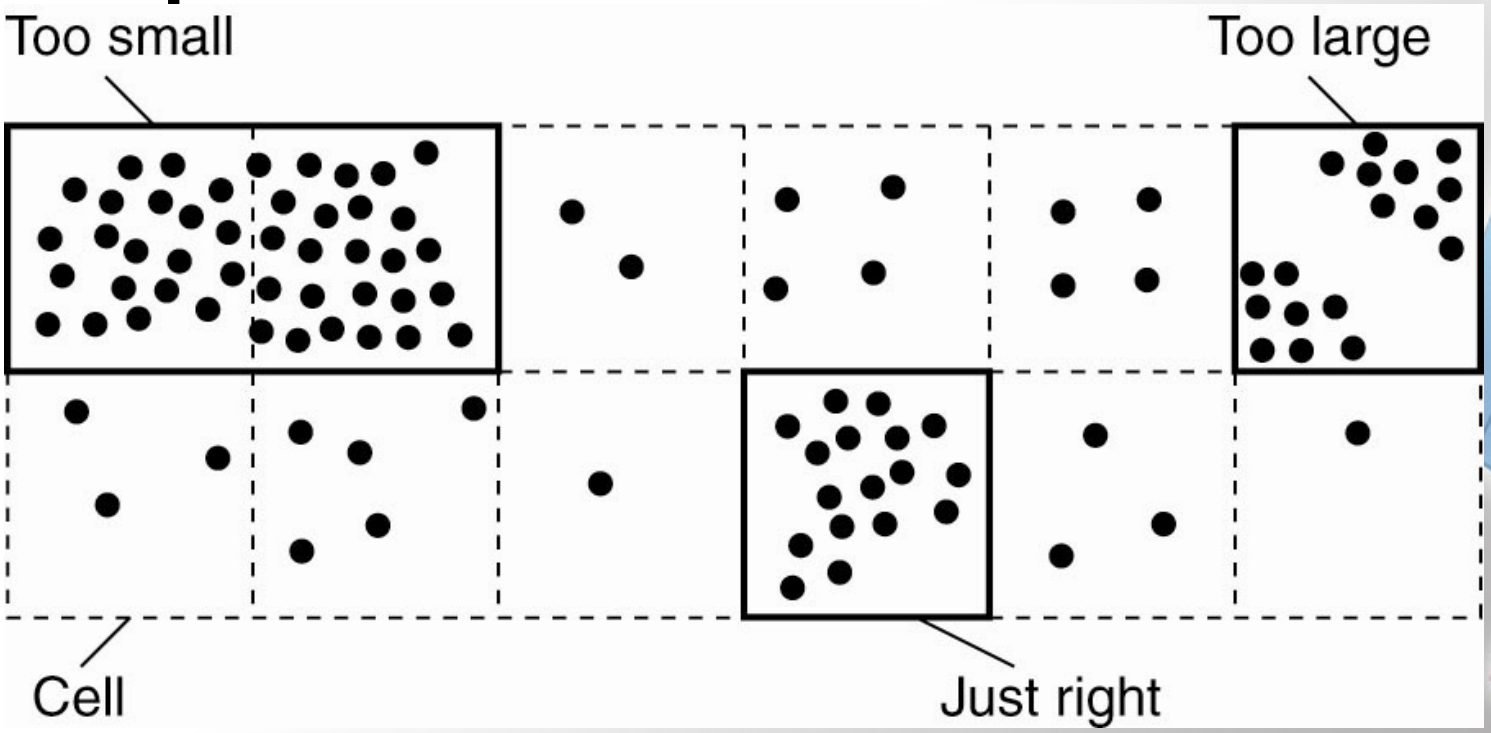


Figure 7-16. Choosing a proper cell size for server placement.

A region is identified to be a collection of nodes accessing the same content, but for which the internode latency is low.
The goal of the algorithm is first to select the most demanding regions - that is, the one with the most nodes - and then to let one of the nodes in such a region act as replica server.

Content Replication and Placement

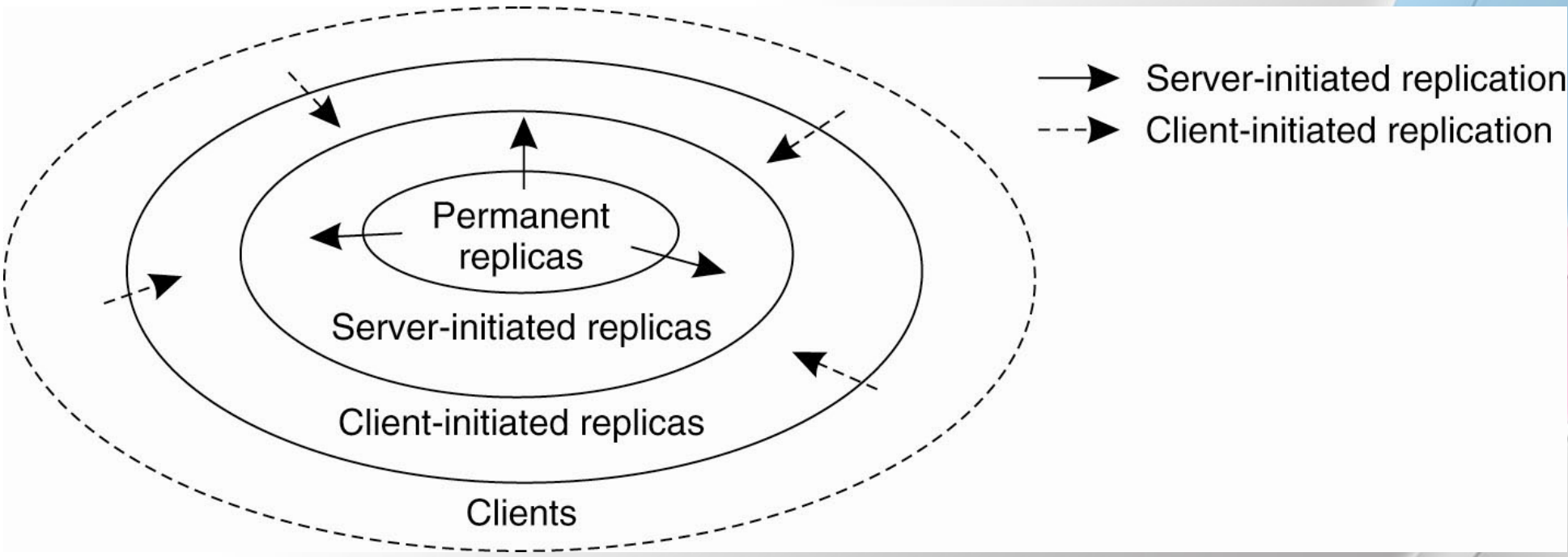


Figure 7-17. The logical organization of different kinds of copies of a data store into three concentric rings.

Permanent Replicas

- Permanent replicas can be considered as the initial set of replicas that constitute a distributed data store. In many cases, the number of permanent replicas is small.
- Consider, for example, a Web site. Distribution of a Web site generally comes in one of two forms:
  1. The first kind of distribution is one in which the files that constitute a site are replicated across a limited number of servers at a single location. Whenever a request comes in, it is forwarded to one of the servers, for instance, using a round-robin strategy.
  2. The second form of distributed Web sites is what is called mirroring. In this case, a Web site is copied to a limited number of servers, called mirror sites, which are geographically spread across the Internet. In most cases, clients simply choose one of the various mirror sites from a list offered to them.

Server-Initiated Replicas

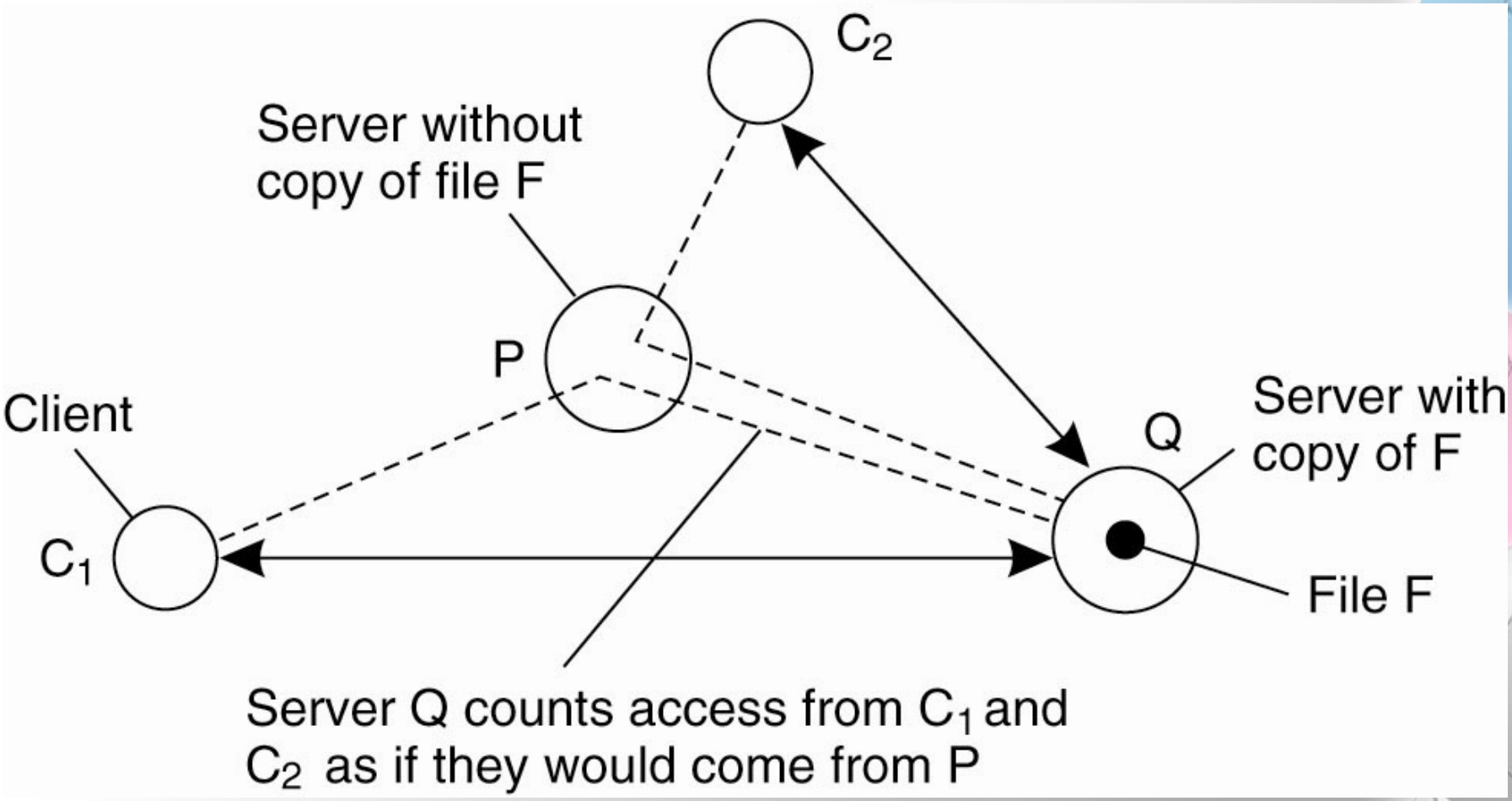


Figure 7-18. Counting access requests from different clients.

Deletion and replication thresholds

Keep track of access counts per file, aggregated by considering server closest to requesting clients

- Number of accesses drops below threshold D => drop file
- Number of accesses exceeds threshold R => replicate file
- Number of access between D and R => migrate file

Client-Initiated Replicas

- An important kind of replica is the one initiated by a client. Client-initiated replicas are more commonly known as (client) caches.
- Data are generally kept in a cache for a limited amount of time
- Placement of client caches is relatively simple: a cache is normally placed on the same machine as its client, or otherwise on a machine shared by clients on the same local-area network.
- However, in some cases, extra levels of caching are introduced by system administrators by placing a shared cache between a number of departments or organizations, or even placing a shared cache for an entire region such as a province or country.

Content distribution

Model

Consider only a client-server combination:

- Propagate only notification/invalidation of update (often used for caches)
- Transfer data from one copy to another (distributed databases)
- Propagate the update operation to other copies (also called active replication)

Note

- No single approach is the best, but depends highly on available bandwidth and read-to-write ratio at replicas.

Push and pull-based protocol

- Push-based (or server-based protocol)
  - Updates are propagated to other replicas without those replicas asking for the updates.
  - Applied when replicas need to maintain high degree of consistency

- Pull-based (or client-based protocol)
  - A server or client requests another server or client to send it any updates it has at that moment.
  - Efficient when the read-to-update ratio is relatively low.

Pull versus Push Protocols

Issue	Push-based	Pull-based
State at 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

Figure 7-19. A comparison between push-based and pull-based protocols in the case of multiple-client, single-server systems.

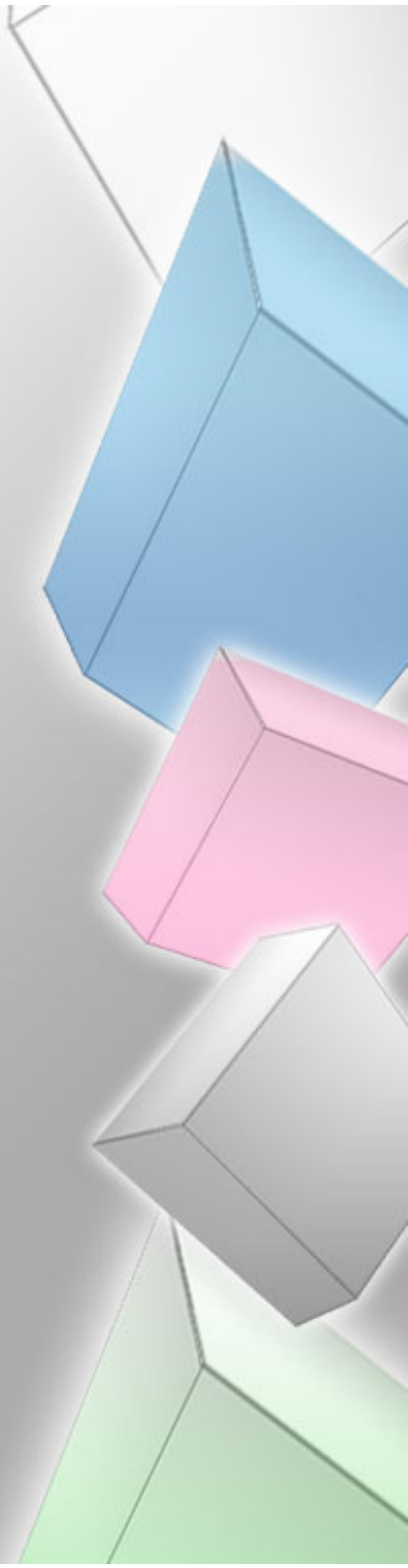
Lease

- A **lease** is a promise by the server that it will push updates to the client for a specified time.

- When a lease expires, the client is forced to poll the server for updates and pull in the modified data if necessary. An alternative is that a client requests a new lease for pushing updates when the previous lease expires.

Content

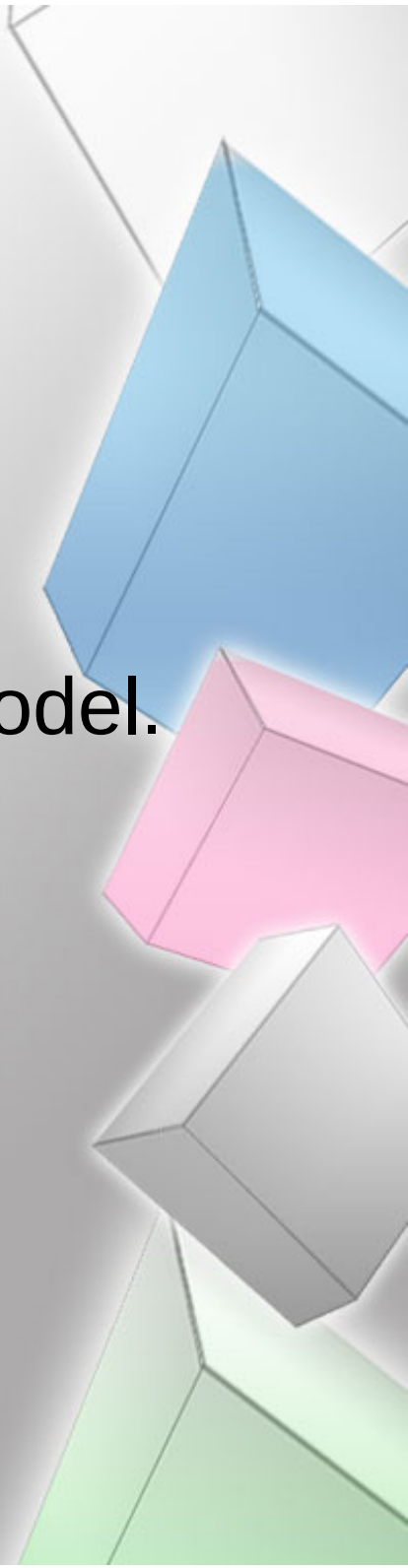
- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols



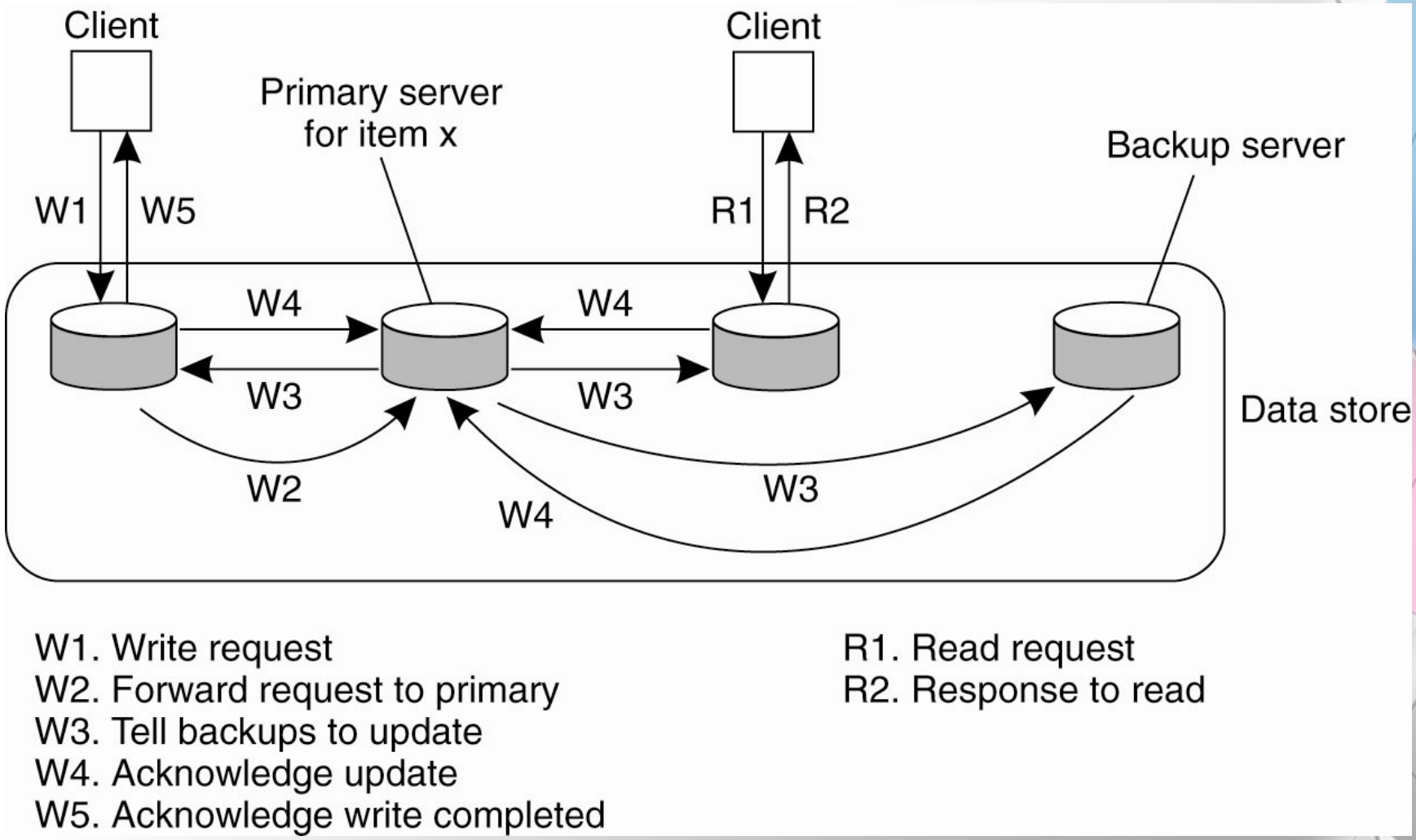
Consistency Protocols

A consistency protocol describes an implementation of a specific consistency model.

- Primary-based protocols
- Replicated-write protocols



Remote-Write Protocols



All write operations need to be forwarded to a **fixed single server**.

Read operations can be carried out locally. Such schemes are also known as **primary-backup protocols**

Local-Write Protocols

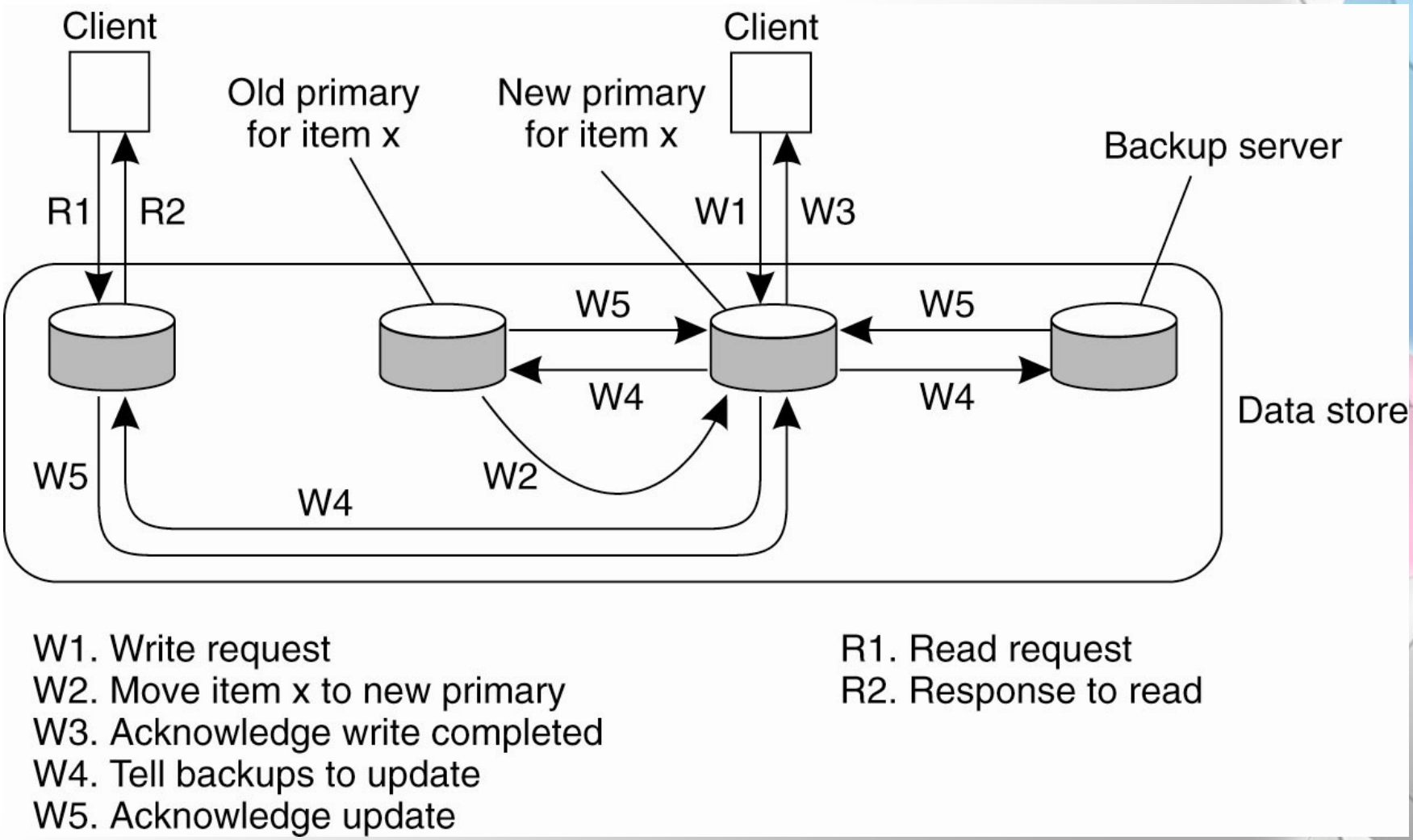


Figure 7-21. Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

Active Replication

- In replicated-write protocols, write operations can be carried out at multiple replicas instead of only one, as in the case of primary-based replicas.

- In active replication, each replica has an associated process that carries out update operations.

- In contrast to other protocols, updates are generally propagated by means of the write operation that causes the update. In other words, the operation is sent to each replica.

Replicated-write protocols

Quorum-Based Protocols

- Clients have to request and acquire the permission of multiple servers before either reading or writing a replicated data item.
- Majority scheme
- Gifford's scheme, two constraints:
  1. $N_R + N_W > N$
  2. $N_W > N / 2$

  Where N is the number of replicas, N_R is the read quorum, N_W is the write quorum.
- To read or write a file, N_R and N_W must be achieved.

Quorum-Based Protocols

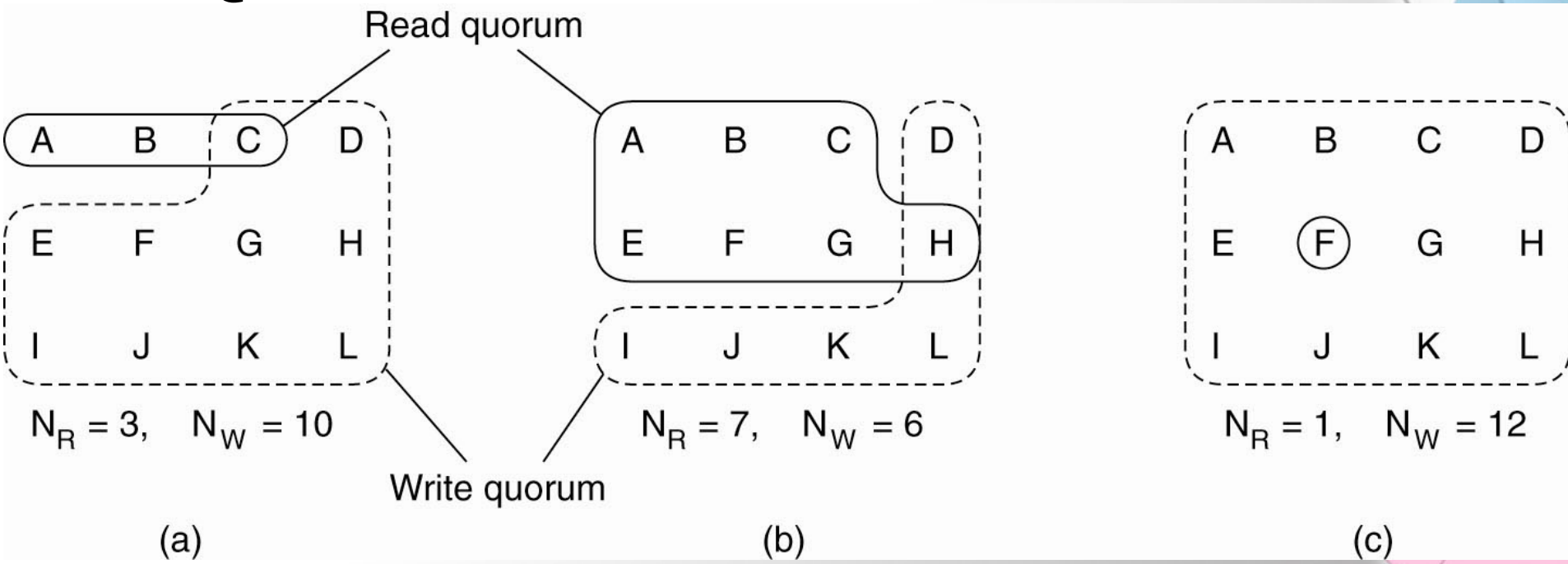


Figure 7-22. Three examples of the voting algorithm.

(a) A correct choice of read and write set.

(b) A choice that may lead to write-write conflicts. (one client chooses {A,B,C,E,F,G,H} as write set and another chooses {D,H,I,J,K,L}). Both writes will be accepted.

(c) A correct choice, known as ROWA (read one, write all).

End of Lesson 7

- Readings
  - Distributed Systems, Chapter 7, except 7.5.1, 7.5.4 and 7.5.5.