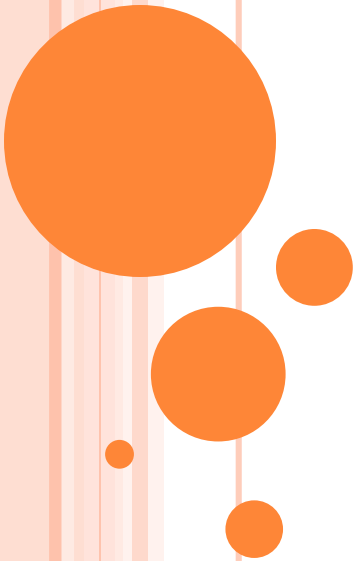


Consistency & Replication

Why replicate?

- Data replication: common technique in distributed systems
- **Reliability**
  - If one replica is unavailable or crashes, use another
  - Protect against corrupted data
- **Performance**
  - Scale with size of the distributed system (replicated web servers)
  - Scale in geographically distributed systems (web proxies)
- **Key issue: need to maintain consistency of replicated data**
  - If one copy is modified, others become inconsistent

What is a Consistency Model?

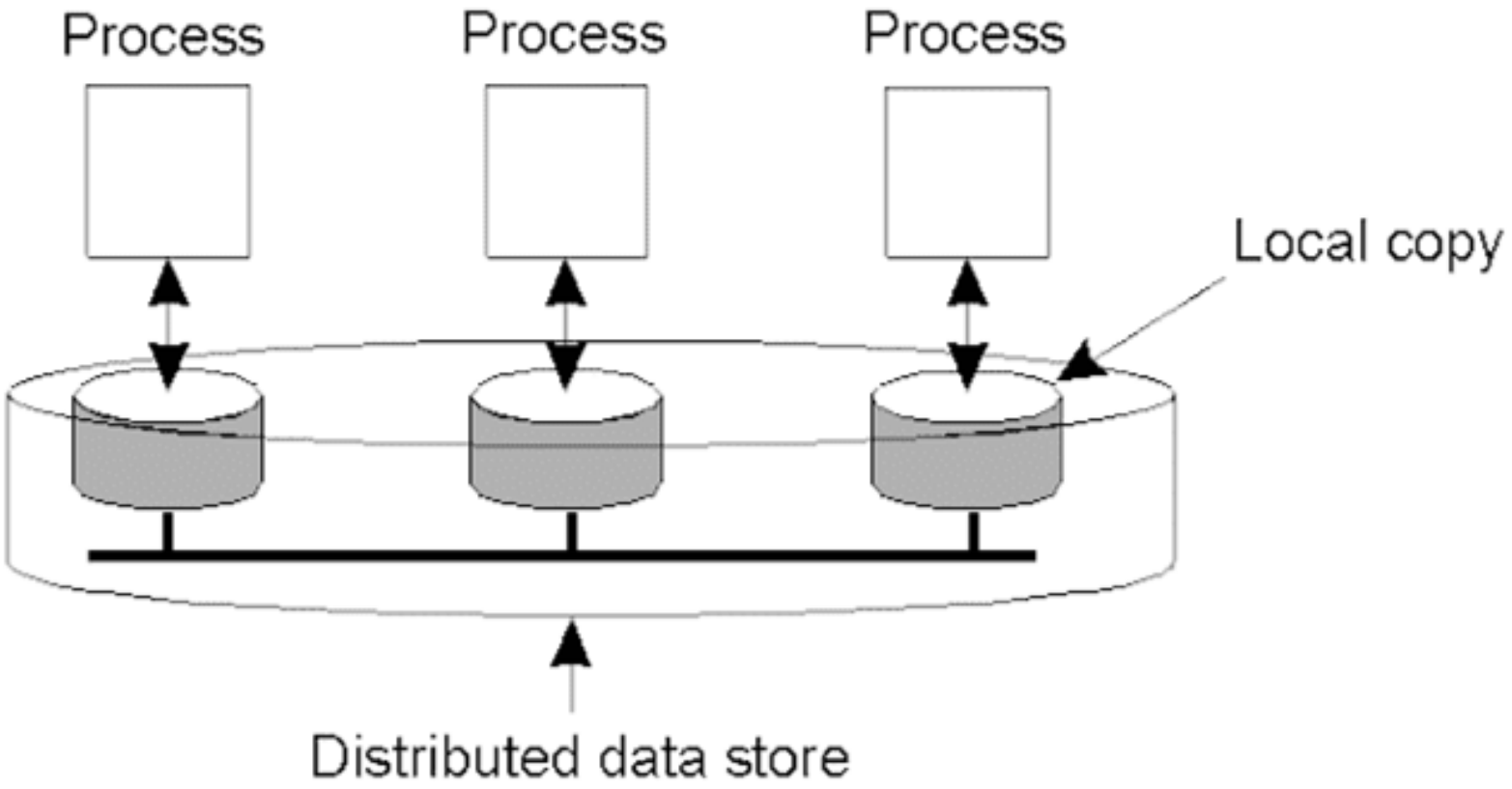
- A “consistency model” is a CONTRACT between a DS data-store and its processes.

- If the processes agree to the rules, the data-store will perform properly and as advertised.

Types of Consistency Models

- Data Centric
- Client Centric

Data-Centric Consistency Models



Data-centric Consistency Models

- Strict Consistency
- Linearizability and Sequential Consistency
- Causal Consistency
- FIFO Consistency
- Weak Consistency
- Release Consistency
- Entry Consistency

Consistency Model Diagram Notation

- $W_i(x)a$ – a write by process 'i' to item 'x' with a value of 'a'. That is, 'x' is set to 'a'.

  (Note: The process is often shown as 'P_i').

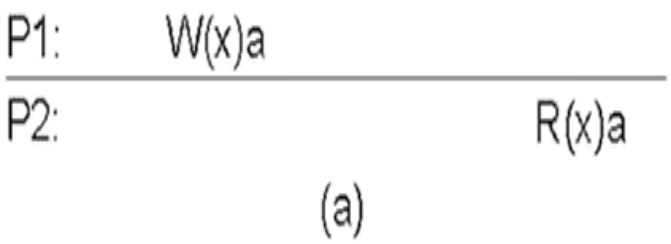
- $R_i(x)b$ – a read by process 'i' from item 'x' producing the value 'b'. That is, reading 'x' returns 'b'.

- Time moves from left to right in all diagrams.

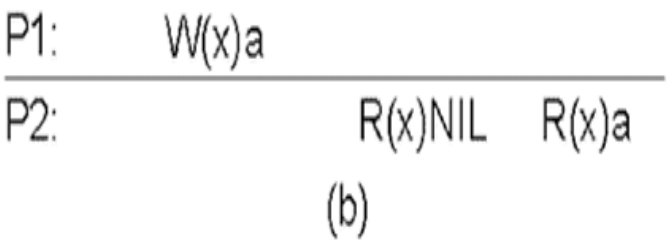
Strict Consistency Diagrams

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

Disadvantage: Assume the existence of absolute global time.



Permitted

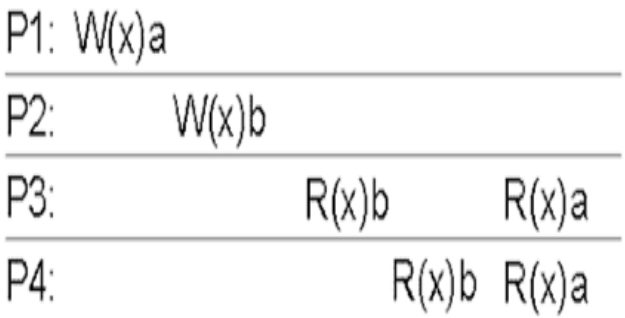


Not permitted

Sequential Consistency

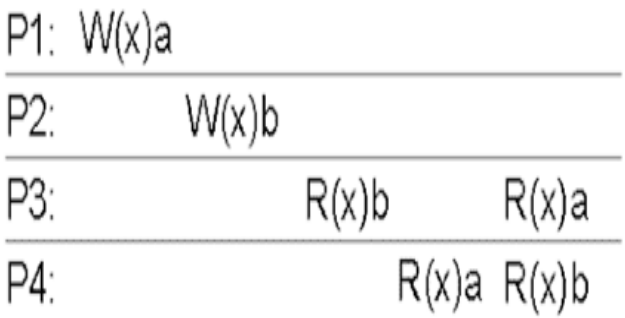
Sequential consistency: weaker than strict consistency

- Assumes all operations are executed in some sequential order and each process issues operations in program order
  - Any valid interleaving is allowed
  - All agree on the same interleaving
  - Each process preserves its program order
  - Nothing is said about "most recent write"



(a)

Permitted



(b)

Not permitted

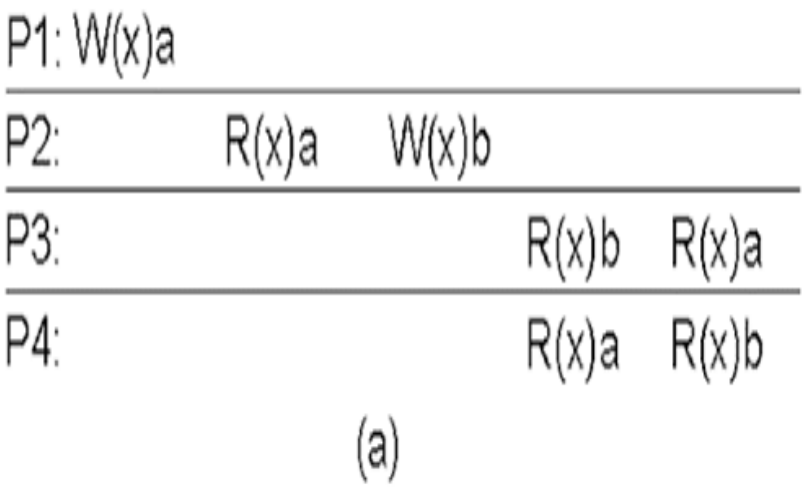
- The result of any execution is the same as if the read and the write .
- Operations by all processes on the data store were executed in some Sequential order and the operations of each individual process appear in this sequence in the order specified by its program.

Causal consistency

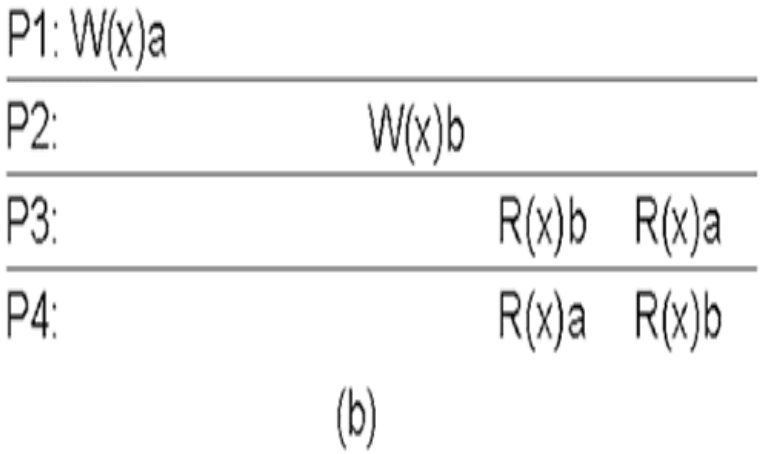
- This model distinguishes between events that are "causally related" and those that are not.
- *If event B is caused or influenced by an earlier event A, then causal consistency requires that every other process see event A, then event B.*
- Operations that are not causally related are said to be *concurrent*.

CAUSAL CONSISTENCY

- Causally related writes must be seen by all processes in the same order.
- Concurrent writes may be seen in different orders on different machines

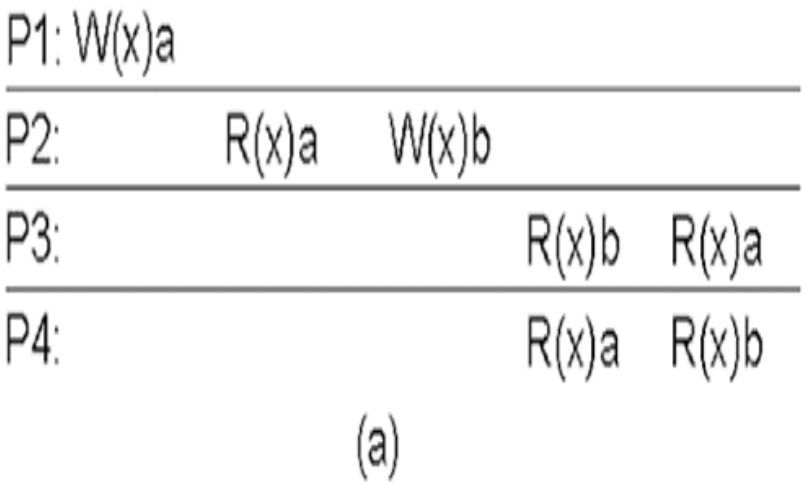


Not permitted



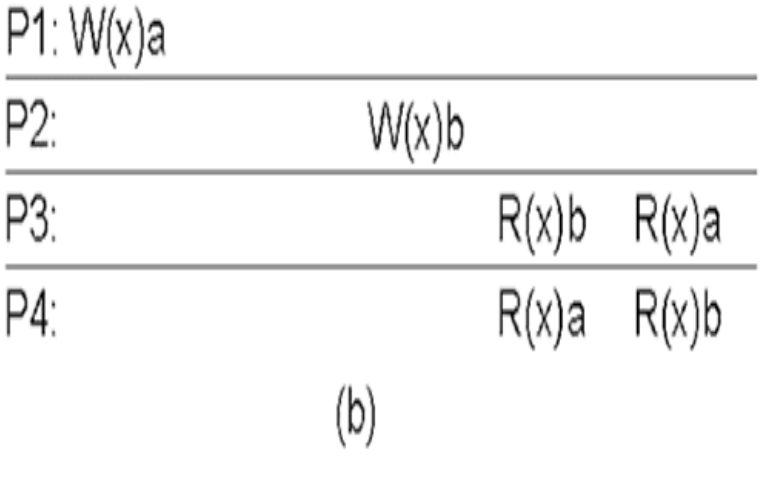
Permitted

Causal consistency



(a)

Not permitted



(b)

Permitted

a) Violation of causal-consistency – P2's write is related to P1's write due to the read on 'x' giving 'a' (all processes must see them in the same order).

b) A causally-consistent data-store: the read has been removed, so the two writes are now *concurrent*. The reads by P3 and P4 are now OK.

Causal consistency

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.

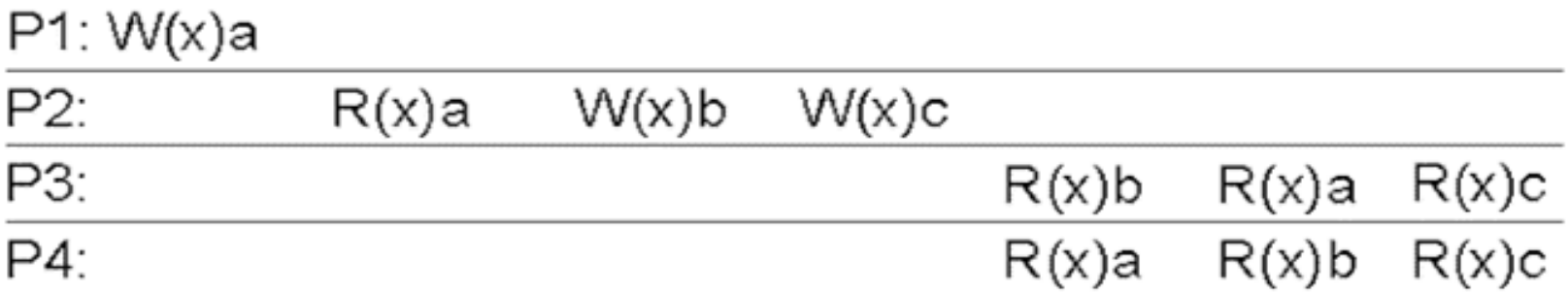
FIFO Consistency

Necessary Condition:

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.

- This is also called "PRAM Consistency" – Pipelined RAM.
- The attractive characteristic of FIFO is that it is easy to implement. There are no guarantees about the order in which different processes see writes – except that two or more writes from a single process must be seen in order.

FIFO Consistency



A valid sequence of events for FIFO consistency

- A valid sequence of FIFO consistency events.
- Note that none of the consistency models studied so far would allow this sequence of events.

Weak Consistency

- Weak consistency model says that not all applications need to see all writes – the criteria for same order is relaxed.
- This model ensures consistency on a *group of operations, as against* individual reads and writes as seen in the other consistency models.
- It introduces the concept of a synchronization variable "S" which is associated with a particular datastore.
- The process acquiring this synchronization variable is allowed to access the critical region and when it leaves the variable should be released.
- The synchronization variable for a datastore assures the process that the value it has got is the most recently written value of that datastore.

Weak Consistency

Properties:

- Accesses to synchronization variables associated with a data store are sequentially consistent.

(All processess see all ops on synch. Variable in the same process in the same order.)

- No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere.

(Syn. Forces all writes that are in progress or partially completed at some local copies but not others to complete everywhere.)

- No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.

(when a data item is accessed either for reading or for writing all the syncs. have to be completed)

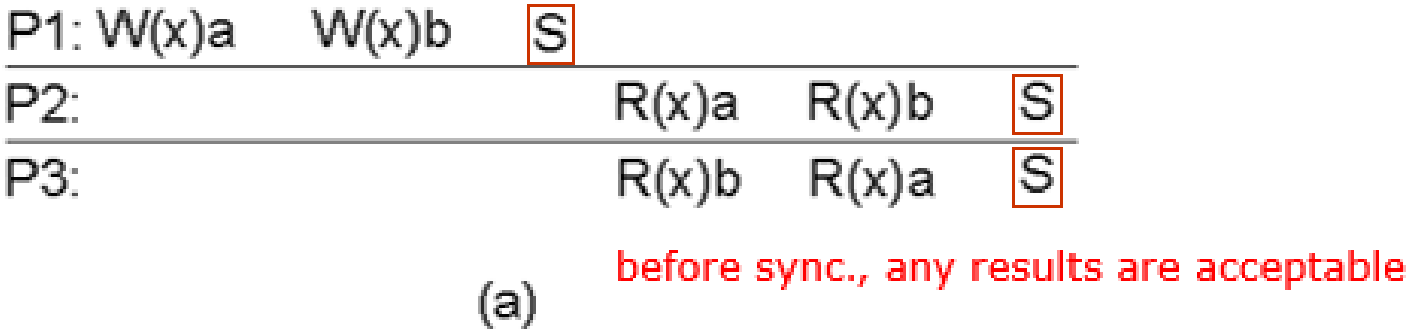
Weak Consistency

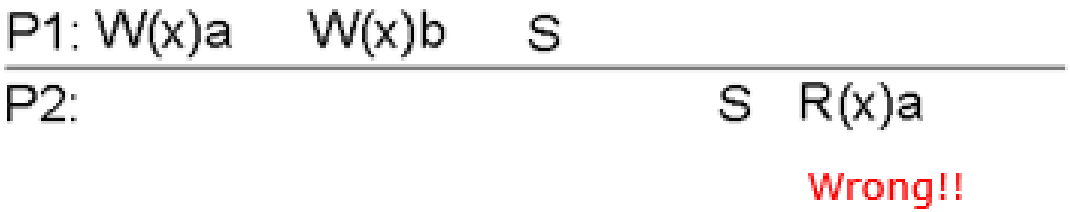
Following criteria must be met:

- 1. Acquisition of a synchronization variable cannot be performed for the process until all updates to the shared data have been performed.

- 2. No other process is allowed to hold the synchronization variable, even not in non-exclusive mode, before an exclusive access to synchronization variable is allowed to perform.

- 3. After an exclusive mode access to a synchronization variable has been performed, any other process' next non-exclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.





a) A valid sequence of events for weak consistency. This is because P2 and P3 have yet to synchronize, so there's no guarantees about the value in 'x'.

b) An invalid sequence for weak consistency. P2 has synchronized, so it cannot read 'a' from 'x' – it should be getting 'b'.

Release Consistency

- In weak consistency model no distinction is made for acquiring synchronization variable for read and write operations on a datastore.

- Release consistency introduces two sync variables, acquire and release:

  1. Acquire: This variable assures that the local copies are consistent with the remote ones.

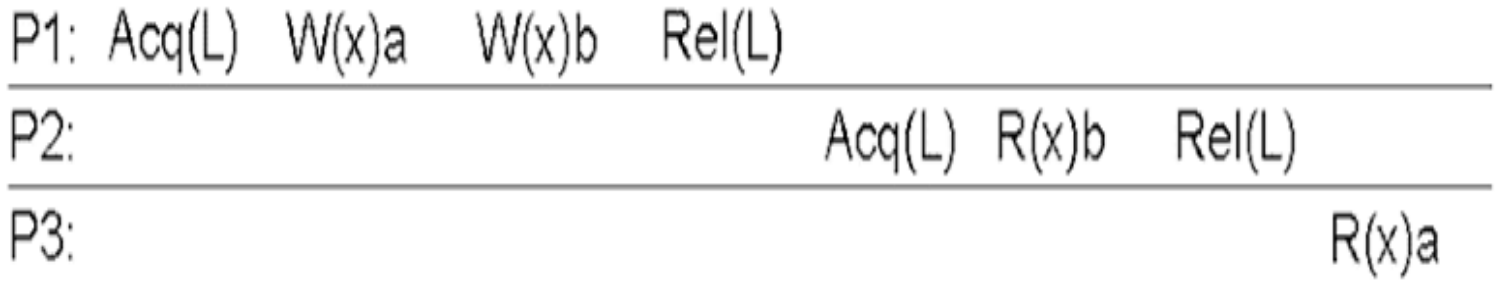
  2. Release: This variable assures that updates to a datastore are propagated to all local copies.

Release Consistency

Rules:

- Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully.
- Before a release is allowed to be performed, all previous reads and writes by the process must have completed
- Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).

Release Consistency



A valid event sequence for release consistency.

- Process P3 has not performed an *acquire*, so there are no guarantees that the read of 'x' is consistent. The data-store is simply not obligated to provide the correct answer.
- P2 does perform an *acquire*, so its read of 'x' is consistent.

Entry Consistency

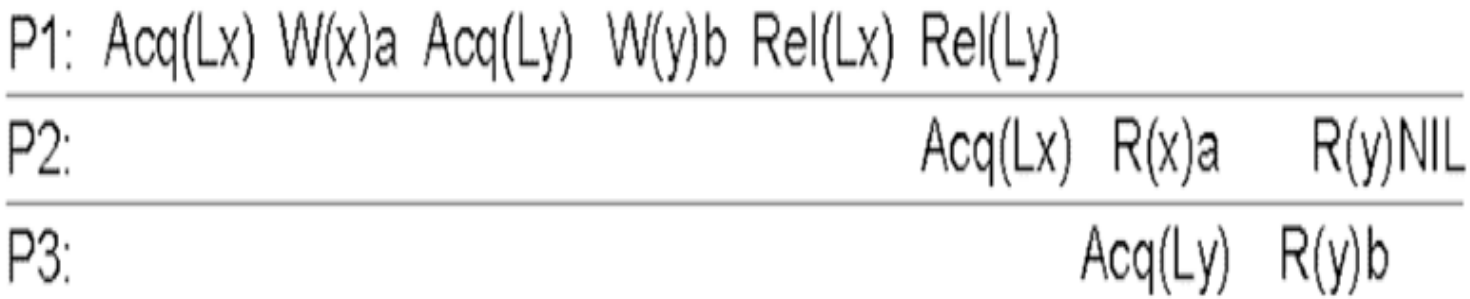
- In entry consistency, an acquire and a release are used for individual data items rather than entire datastore.

- The data consistency is offered at the entry. It should meet the following two main conditions:

  1. At an *acquire, all remote changes to data item must be brought up to date with remote ones.*

  2. Before a write to a data item, a process must take care that no other process is performing write *at the same time; thus the synchronization variable should be held in exclusive mode.*

Entry Consistency



A valid event sequence for entry consistency

Locks associate with individual data items, as opposed to the entire data-store.

Note: P2's read on 'y' returns NIL as no locks have been requested.

Summary of Data-centric 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

(a)

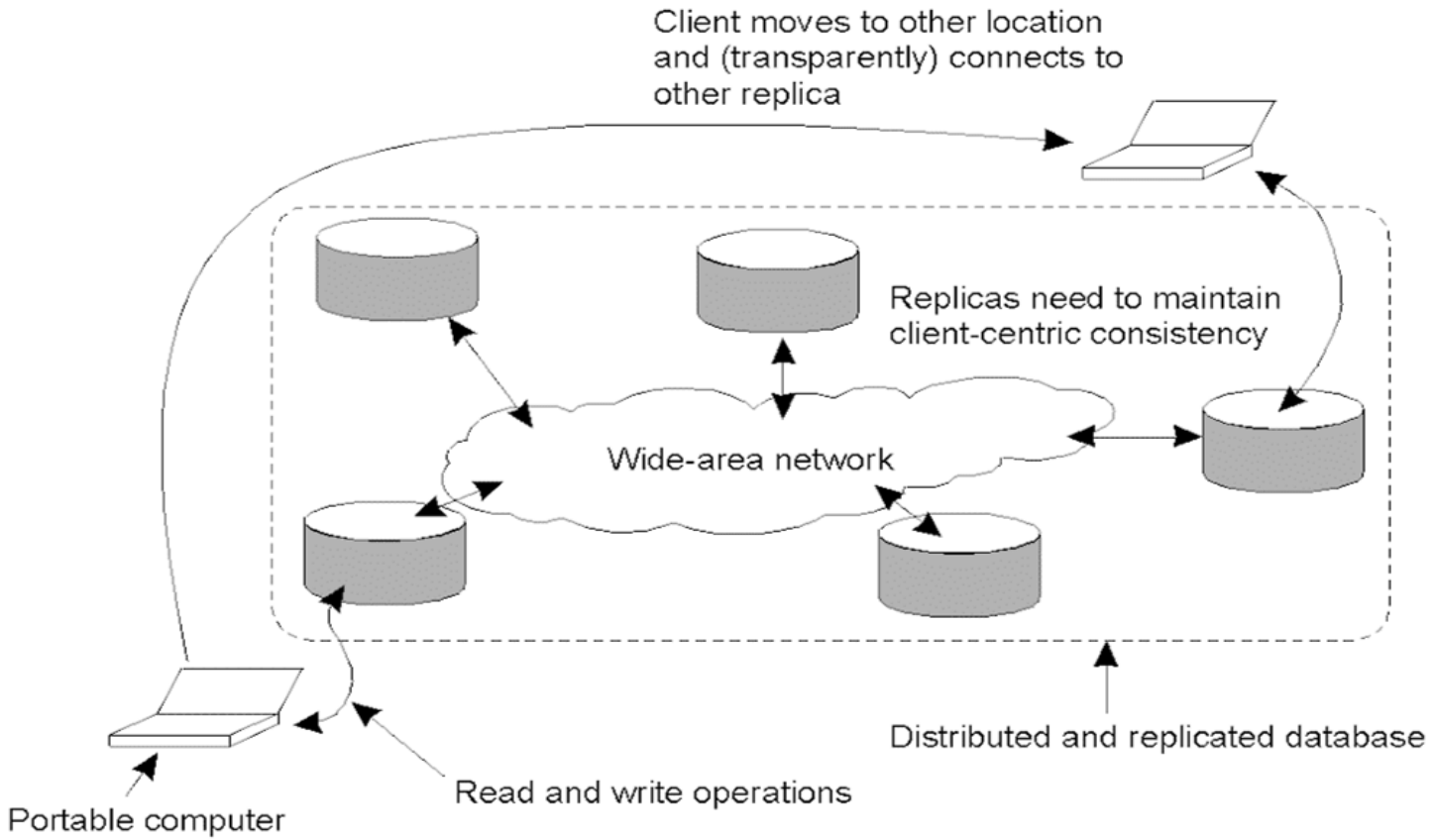
Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b)

Client-centric Consistency Models

- The data stores characterized by the lack of simultaneous updates,
- Most operations involve reading data. These data stores offer a very weak consistency model, called eventual consistency.
- In many database systems, most processes hardly ever perform update operations; they mostly read data from the database. Only one, or very few processes perform update operations.
- The question then is how fast updates should be made available to only reading processes.
- another example is the World Wide Web. In virtually all cases, Web pages are updated by a single authority,

Eventual Consistency



A mobile user accessing different replicas of a distributed database has problems with eventual consistency

Client-Centric Consistency

Four models in client-centric consistency:

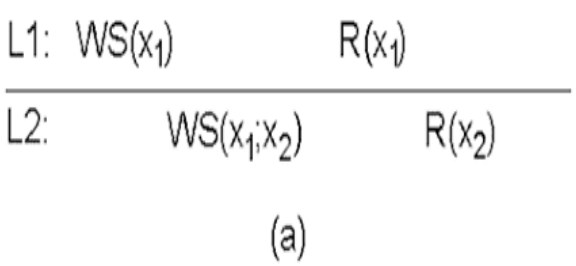
- Monotonic Read Consistency
- Monotonic Write Consistency
- Read-your-writes Consistency
- Writes-follows-reads Consistency

Client-Centric Consistency

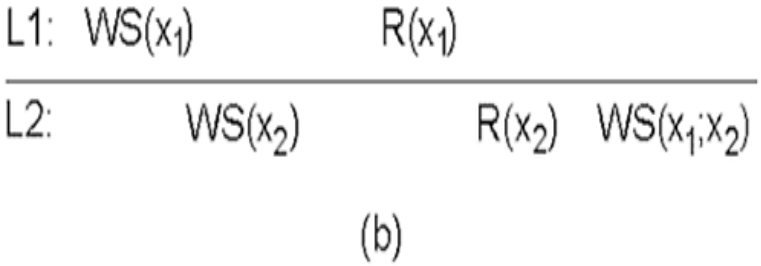
- xi[t] -version of data x at copy Li at time t
- WS(xi[t]) is a set of write operations at Li which have led to version xi of x
- When these operations later (t2) are performed to x at copy Lj ,it is written as WS(xi[t1]; xj[t2])

Monotonic Reads

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



(a)



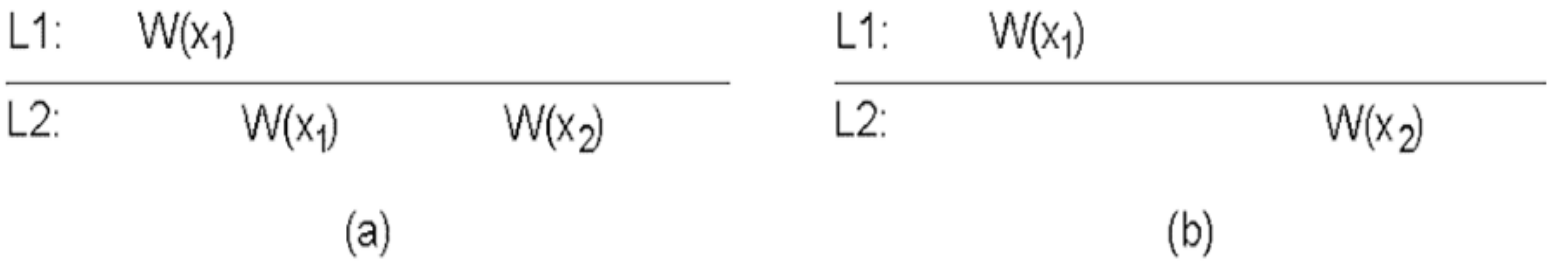
(b)

a) A monotonic-read consistent data store
b) A data store that does not provide monotonic reads.

Example: distributed email database

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

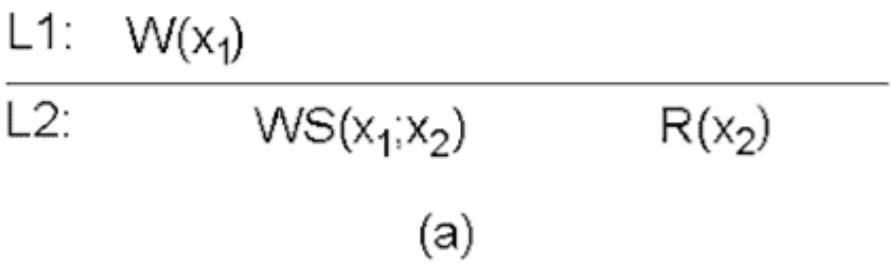


a) A monotonic-write consistent data store.
b) A data store that does not provide monotonic-write consistency.

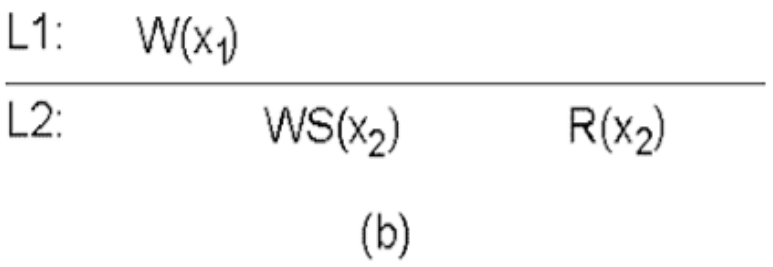
Example: The MW-guarantee could be used by a text editor when editing replicated files

Read Your Writes

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



(a)



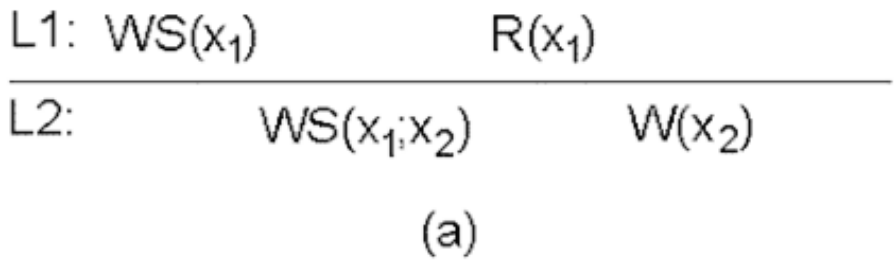
(b)

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

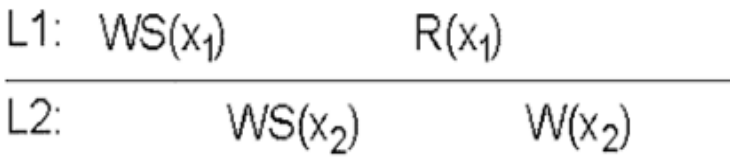
Example: updating passwords

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, it is guaranteed to take place on the same or a more recent value of x that was read.



(a)



(b)

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

Example: replicated bulletin board database

Client centric model-Summary

- Monotonic read

  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 write

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

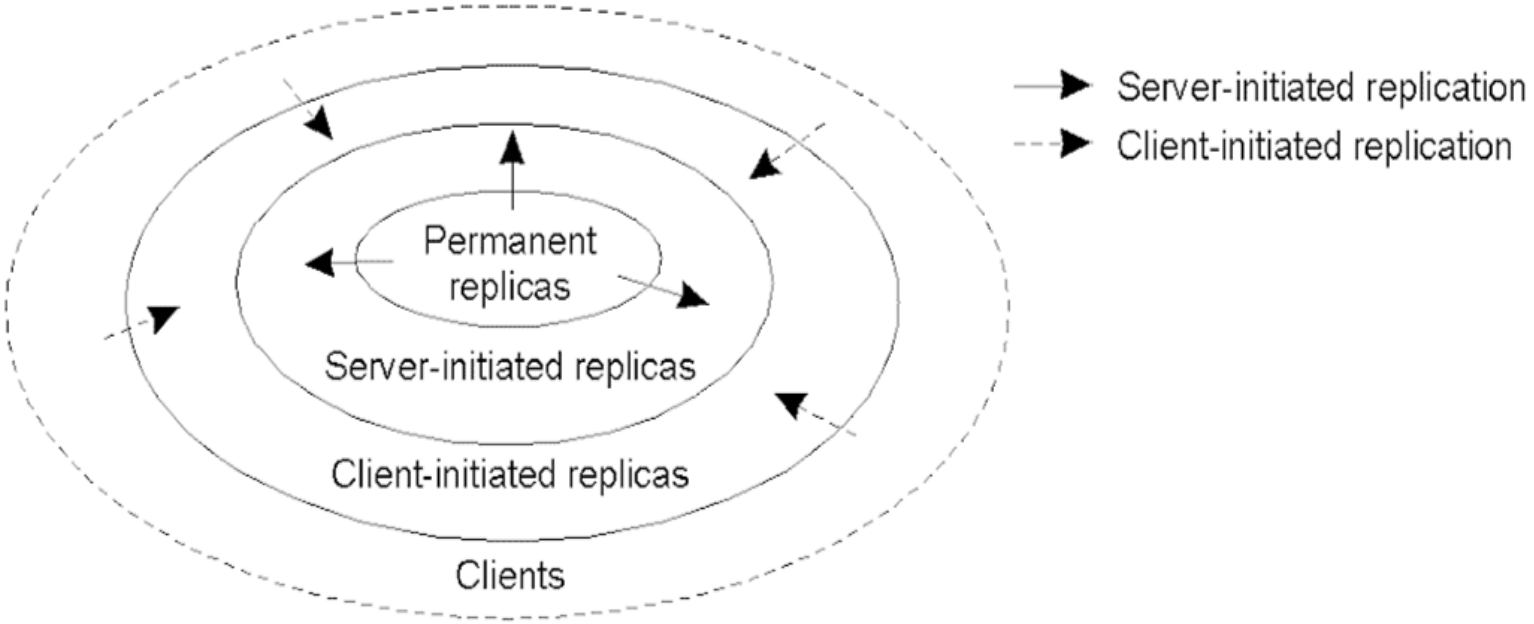
- Read your writes

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

- 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 garanteed to take place on the same or more recent values of x that was read

Replica Placement



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

Replica Placement Types

There are three types of replica:

1. ***Permanent replicas***: tend to be small in number, organized as COWs (Clusters of Workstations) or mirrored systems.
2. ***Server-initiated replicas***: used to enhance performance at the initiation of the owner of the data-store. Typically used by web hosting companies to geographically locate replicas close to where they are needed most. (Often referred to as "push caches").
3. ***Client-initiated replicas***: created as a result of client requests – think of browser caches. Works well assuming, of course, that the cached data does not go *stale* too soon.

Replica Placement

- Permanent replicas
  - Initial set of replicas. Created and maintained by DDS-owner(s)
  - Writes are allowed
  - E.g., web mirrors
- Server-initiated replicas
  - Enhance performance
  - Not maintained by owner of DDS
  - Placed close to groups of clients
    - Manually
    - Dynamically
- Client-initiated replicas
  - Client caches
  - Temporary
  - Owner not aware of replica
  - Placed closest to a client
  - Maintained by host (often the client)

Permanent Replicas

- Permanent Replicas
  - Initial set of replicas that constitutes a distributed data store
  - Typically, the number of it is small

- Example: Web site

Server-Initiated Replicas

Server-Initiated Replicas

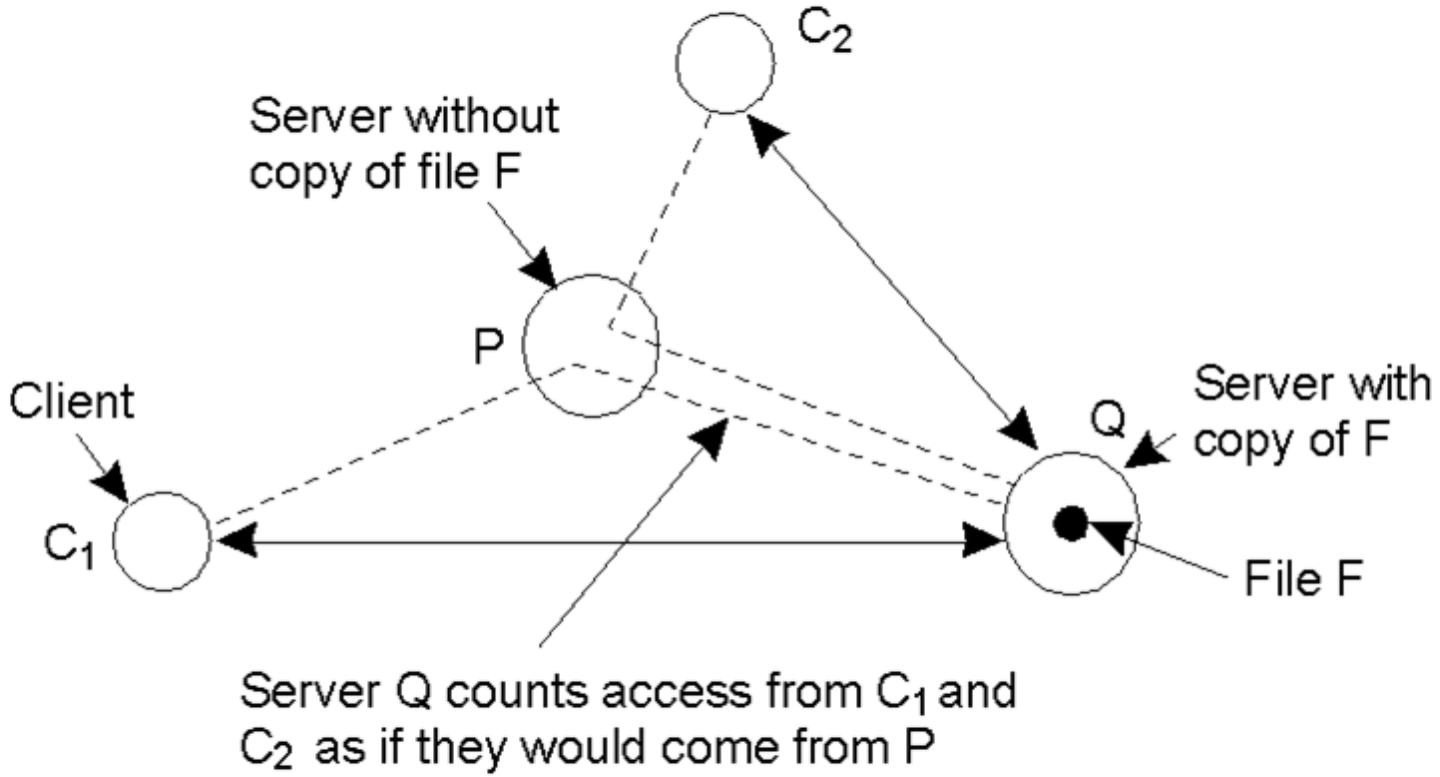
- Created at the initiative of the owner of the data store
- Exist to enhance performance

Work Scheme

How to decide where and when replicas should be created or deleted?

- Each server keeps track of access counts per file, and where access requests come from.
- When a server Q decides to reevaluate the placement of the files it stores, it checks the access count for each file.
- If the total number of access requests for F at Q drops below the deletion threshold del (Q,F), it will delete F unless it is the last copy.

Server-Initiated Replicas



1. Reduce load on servers
2. Specific files could be migrated.

Counting access requests from different clients.

Client-Initiated Replicas

- **Work Scheme**
  - When a client wants access to some data, it connects to the nearest copy of the data store from where it fetches the data it wants to read,
  - When most operations involve only reading data, performance can be improved by letting the client store requested data in a nearby cache.
  - The next time that same data needs to be read, the client can simply fetch it from this local cache.

- **Client-Initiated Replicas**
  - Created at the initiative of clients
  - Commonly known as client caches

Replica Placement

Replication Models

1. Master–Slave: **Slaves are read-only**, updates only in master, slaves only synchronize with master.

2. Client–Server: Just like master–slave model, but updates **can happen in any slave replica too**. Which are first pushed to server, and then to others.

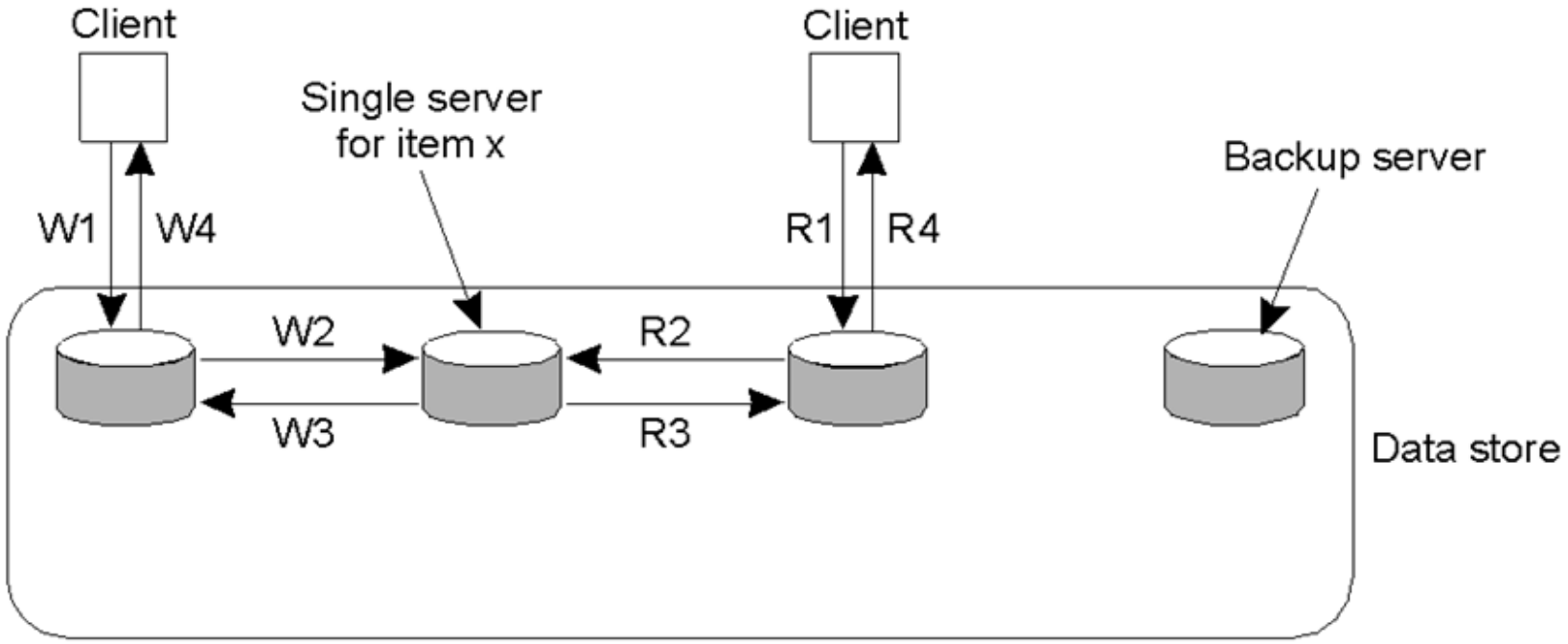
3. Peer–to–Peer: All replicas are of equal importance and are peer. Any replica can synchronize with any other besides any replica can propagate the update. All peers should have all the information

Replica Consistency

1. Optimistic:
2. Pessimistic:

Consistency Protocols: Remote-Write Protocols (1)a

Primary –based : data item x is associated with a primary responsible for co-ord. writes

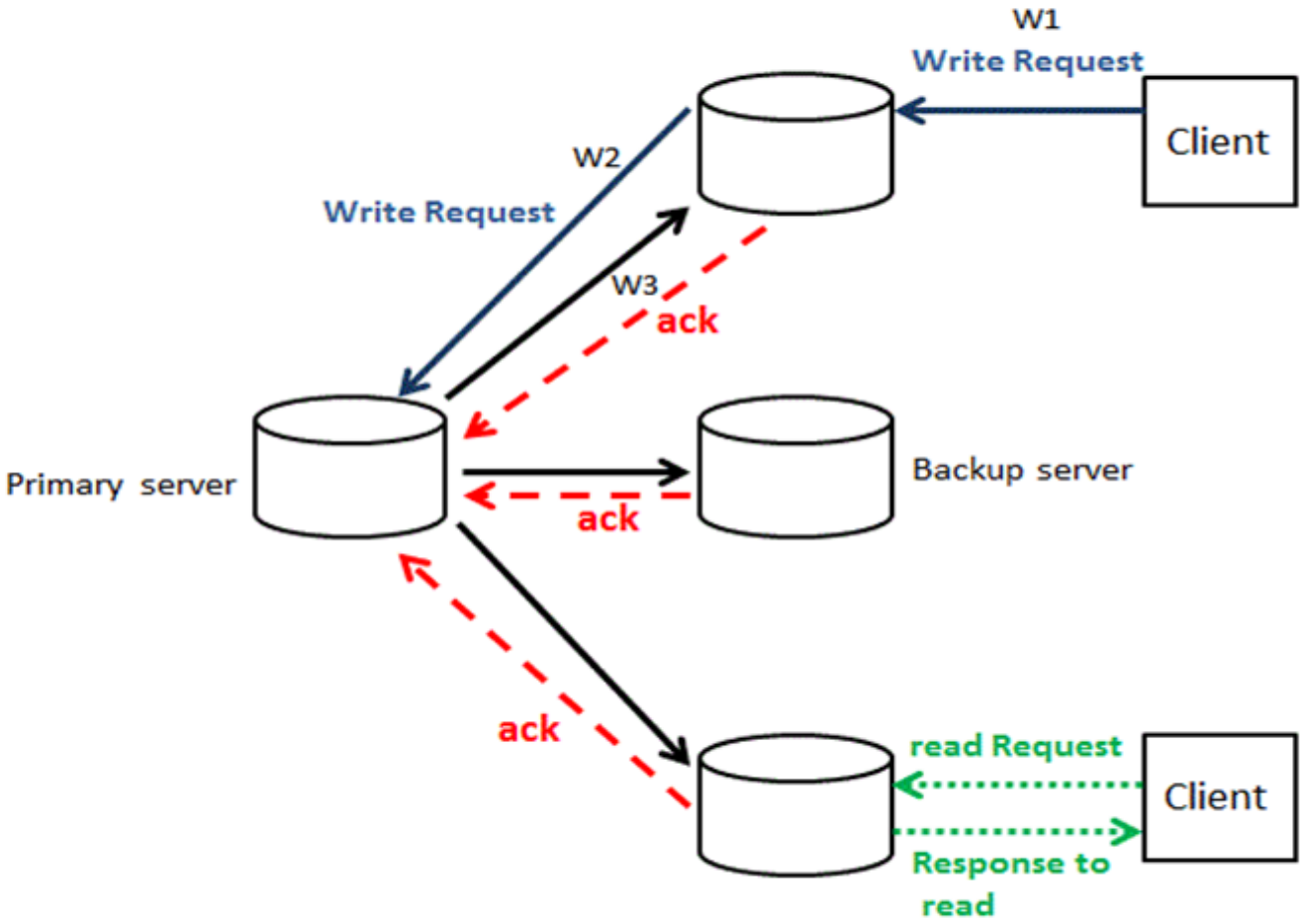


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

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.

Primary-Based replication Protocol



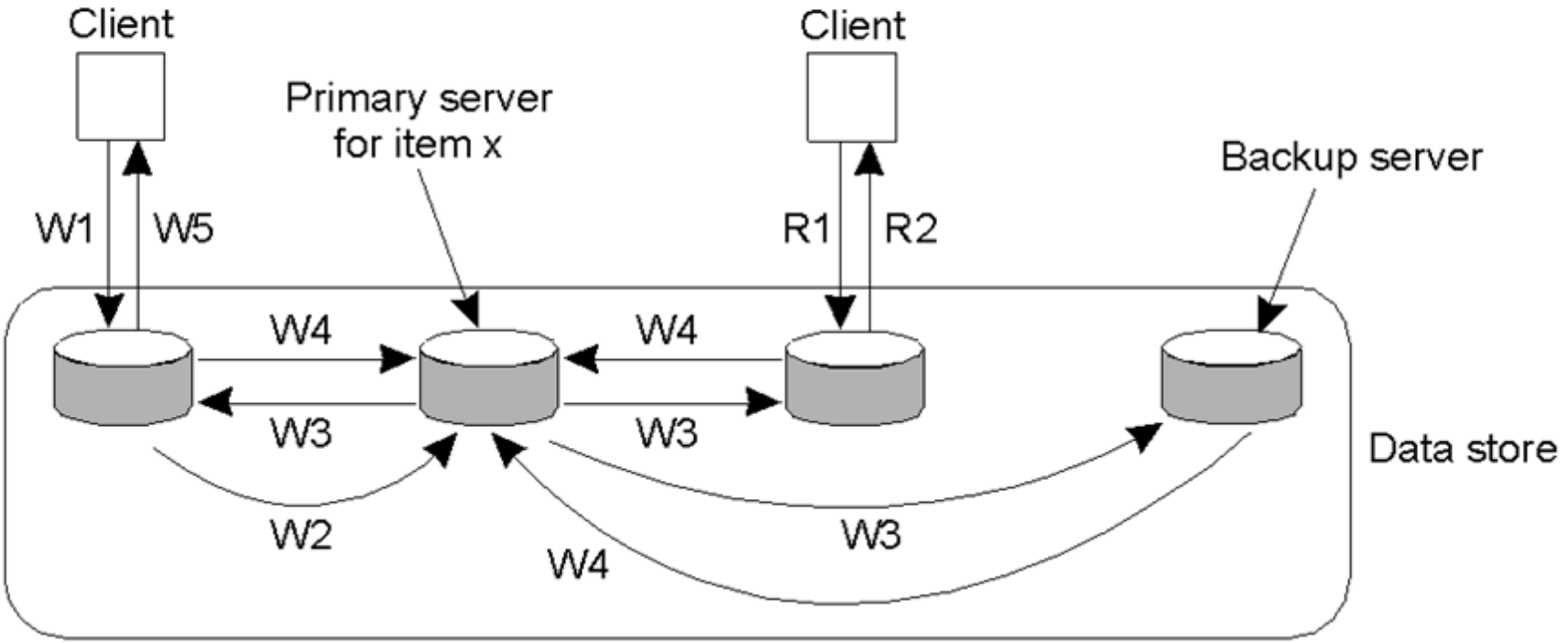
Primary-Based Remote-Write Protocols (cont.)

Replicated: primary-backup remote write protocol

–Primary copy and backups for each data item
–Read from local copy
–Write to the (remote) primary server
–Update backups

• Blocking vs. Non-Blocking update

Remote/Replicated -Write Protocols



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.