Distributed Systems (5DV147)

Consistency

Fall 2014

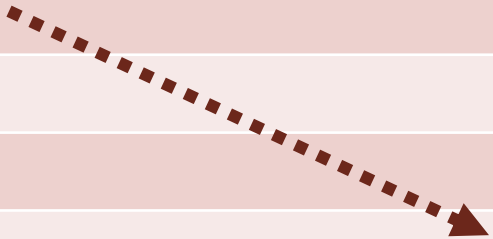
Intuition

- A replicated system is correct when:
  - It maintains execution despite failures
  - Clients can't tell the difference between the results from a system that uses replicated data from those obtained from a system with a single correct replica
  - In general we expect a read to return the last value written

    … but which is the last value written since we don't have a global clock?

Example

Client 1	Client2
$setBalance_B(x,1)$	
$setBalance_A(y,2)$	
	$readBalance_A(y) \rightarrow 2$
	$readBalance_A(x) \rightarrow 0$



- Local replica of Client 1 is B
- Local replica of Client 2 is A

Consistency problem

Replication improves reliability and performance

… but

when a replica is updated, it becomes different from the others

… so

we need to propagate updates in a way that temporal inconsistencies are not noticed

… however

this may degrade performance severely

Consistency models

Consistency model

- It is a contract between processes and a data store
  - Processes agree to follow certain rules, and the store promises to work correctly (Tanenbaum and van Steen, 2002)
  - What to expect when reading and updating shared data (while others do the same)
  - Models restrict the values that a read can return.
    - Minor restrictions' models are easy to use but have low performance
    - Major restrictions' models are hard to use but offer better performance
- Types of models
  - Data-centric models (system-wide)
  - Client-centric models (single client)

Data-centric consistency models

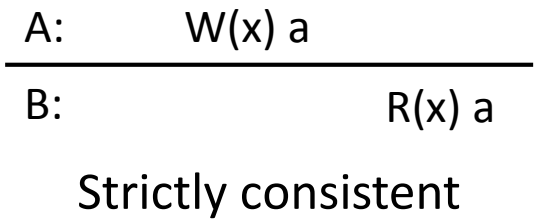
Introduction

- These models provide a system wide consistent view of the data store
  - Concurrent processes can simultaneously update the data store
  - A data store is distributed across a number of machines
  - Writes are propagated to other replicas
- These models are concerned with consistently ordering operations to the data store

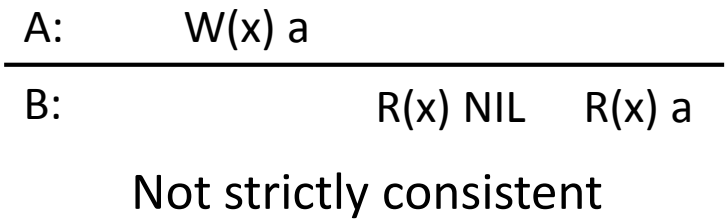
Strict consistency

- Every read of *x* returns a value corresponding to the result of the most recent write to *x*
- True replication transparency, every process receives a response that is consistent with the real time
- All writes are instantaneously visible to all process
- Assumes absolute global time
  - Due to message latency, strict consistency is difficult to implement

Example



Strictly consistent



Not strictly consistent

In general, A:write$_t$(x,a) then B:read$_{t'}$(x,a) ; t'>t

(regardless on the number of replicas of x)

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.5**

Linearizability

- Interleaving of reads and writes into a single total order that respects the local ordering of the operations of every process (i.e., program order must be maintained)
  - A trace is consistent when every read returns the latest write preceding the read
- A trace is linearizable when
  - It is consistent
  - If t_1, t_2 are the times at which p_i and p_j perform operations, and $t_1 < t_2$, then the consistent trace must satisfy the condition that $t_1 < t_2$

Example

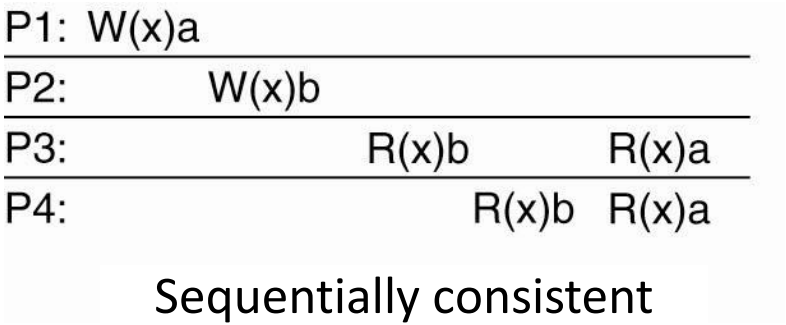
A:	W(x) 1		R(y) 1
B:		W(y) 1	R(x) 1

The linearizable trace is $W_A(x,1)$, $W_B(y,1)$, $R_A(y,1)$, $R_B(x,1)$

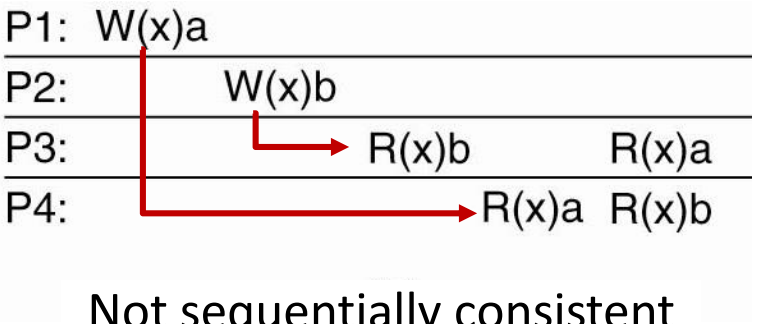
Sequential consistency

- *"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"* (Lamport 1979)
  - Is not concerned with real time
  - All processes see the same interleaving of operations
  - Requires that interleaving preserving local temporal order of reads and writes are consistent traces

Sequential consistency example



Sequentially consistent



Not sequentially consistent

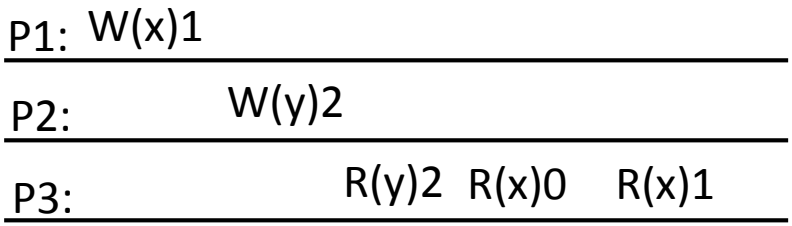
Sequence of operations:

$W_2(x)b$, $R_3(x)b$, $R_4(x)b$, $W_1(x)a$, $R_3(x)a$, $R_4(x)a$

Must be seen in the same order by all processes

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.6**

One more example



Sequence of operations:

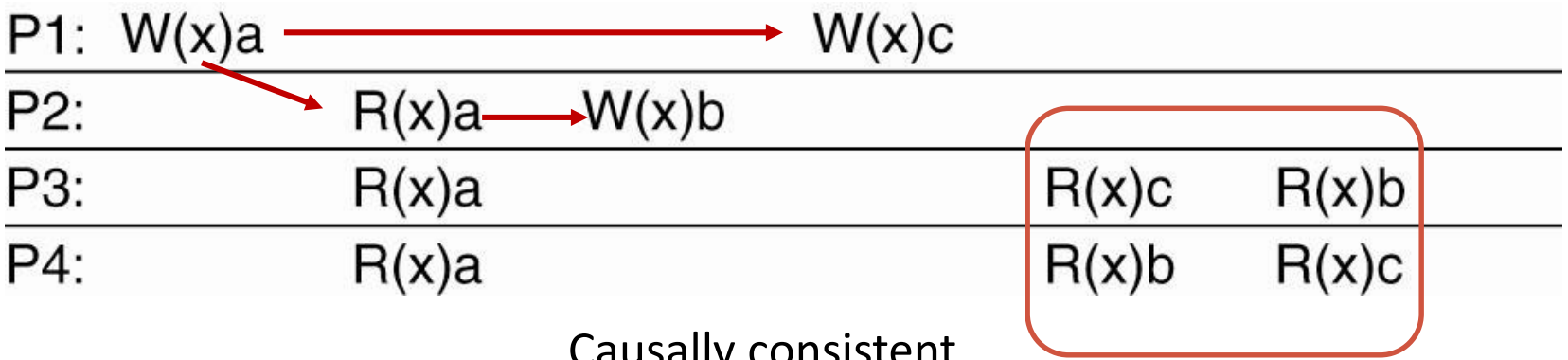
$R_3(x)b$, $W_2(y)2$, $R_3(y)2$, $W_1(x)1$, $R_3(x)1$

Notice that processors can see writes from other processors but they can only see their reads

Causal consistency

- All writes that are potentially causally related must be seen by every process in the same order, and reads must be consistent with this order
- Writes that are not causally related to one another *(concurrent)* can be seen in any order
- No constraints on the order of values read by a process if writes are not causally related

Example



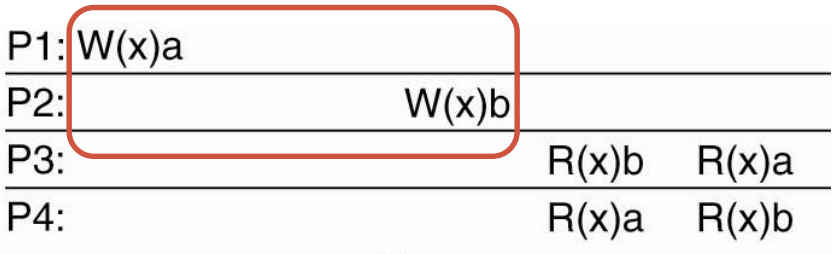
Causally consistent

$W_1(x)a \rightarrow R_2(x)a \rightarrow W_2(x)b$

$W_1(x)a \rightarrow W_1(x)c$

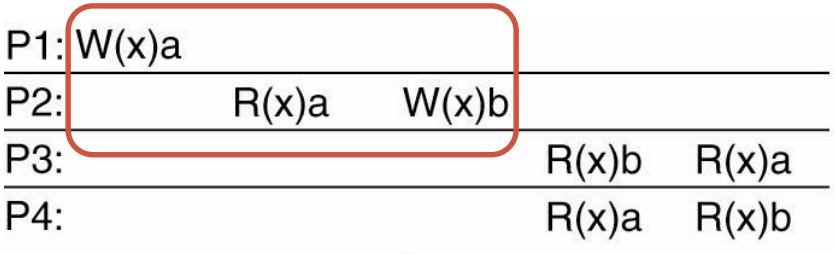
Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.9**

Example



Causally consistent

Causally consistent because $w_1(x)a$ and $w_2(x)b$ are concurrent and not causally related



Not causally consistent

Not causally consistent:

$W_1(x)a \rightarrow W_2(x)b$ so R(x)b must always precede R(x)b

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.10**

Consistency	Description
Strict	Absolute time ordering on all shared accesses, essentially impossible to implement it in distributed systems
Linearizability	All processes see all shared accesses in the same order. Accesses are ordered based on a global timestamp. Good for reasoning about correctness of concurrent programs but not really used for building programs
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time. Feasible and popular but has poor performance
Causal	All processes see causally-related shared accesses in the same order. There is no globally agreed upon view of the order of operations

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.18.a**

Client-centric consistency models

Motivation

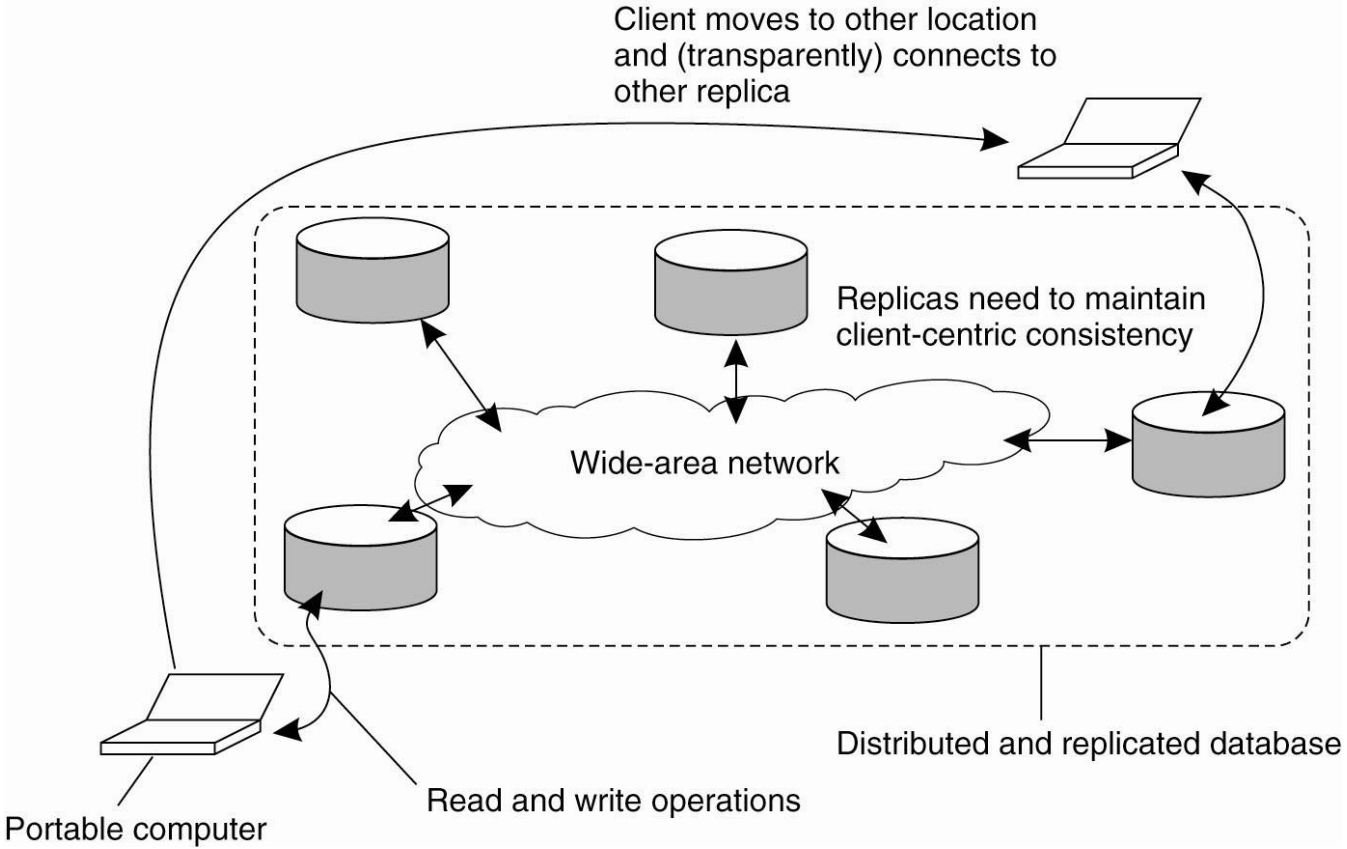
- Data stores characterized by lack of simultaneous updates (or updates that are easily resolved)
  - Read-Write, Write-Read, Read-Read, ~~Write-Write~~
  - Most operations involve reading data
- If updates are infrequent, eventually all replicas will obtain the update and become identical
  - Good if clients always access the same replica
- Predominant case for current large-scale Internet services
  - CAP theorem (Consistency, Availability, Partition tolerable)

    … more of this in the next lecture

Eventual consistency

- Maintains consistency for *individual clients,* not considering concurrent access by different clients
- Ensure that replicas are brought up to date with data that has been manipulated by a client and that probably resides at another replica sites
  - If there are no updates, eventually all replicas will be consistent
  - Easier if clients access a single replica (more difficult if clients access different replicas over a short period of time)
  - Delay resolving conflicts, but updates are guaranteed to propagate to all replicas

Several variations …



- Clients are unaware of which replica they are accessing
- Clients may access different replicas
- Updates need to be propagated or otherwise there is inconsistent behavior
- Avoid write-write conflicts if data objects have a single owner (that can update the object)

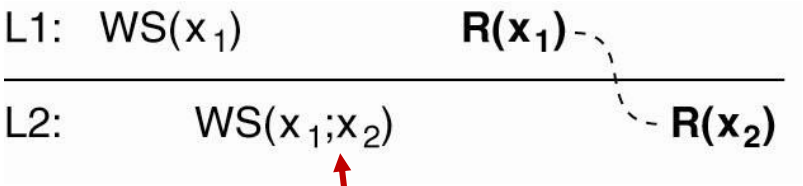
Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.19**

Notation

- $X_i[t]$: data item X, at replica L_i, at time t
- $WS(X_i[t])$: Writing set, i.e., series of write operations until X is at version [t]
- $WS(X_i[t_1]; X_j[t_2])$: Operations in $WS(X_i[t])$ were also made at replica copy j at time t_2

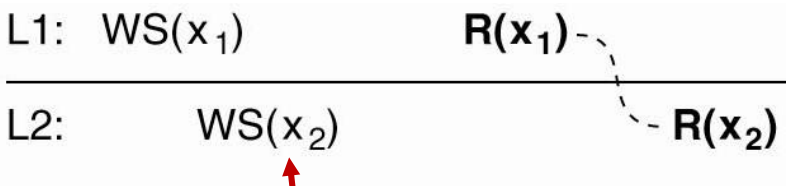
Monotonic-read consistency

- If a process has seen a value of (data item) *x* at a certain time, it will never see an older version of *x* at a later time



A monotonic-read consistent data store

The state has been copied to L2

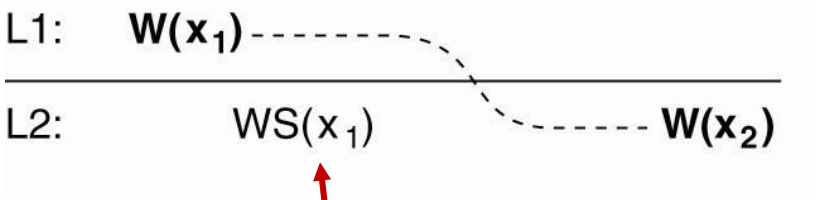


A not monotonic reads data store

Only the state in L2 is considered

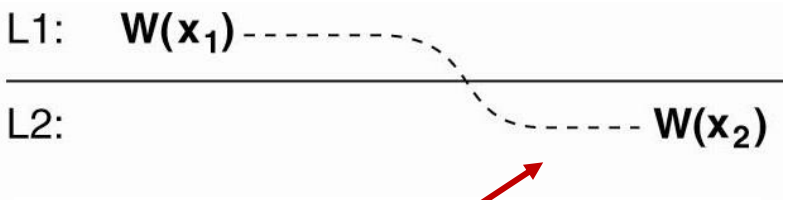
Monotonic-write consistency

- A write to data item *x* is completed before any successive write to *x* by the same process



A monotonic-write consistent data store

The last write is reflected at L2

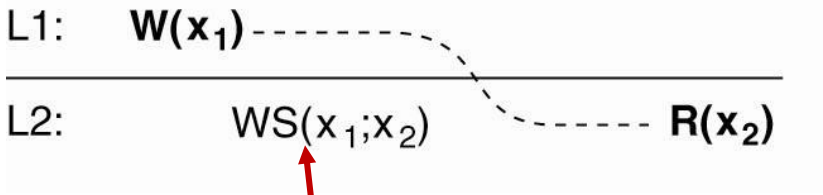


A Not monotonic-write data store

The latest write is not updated at L2

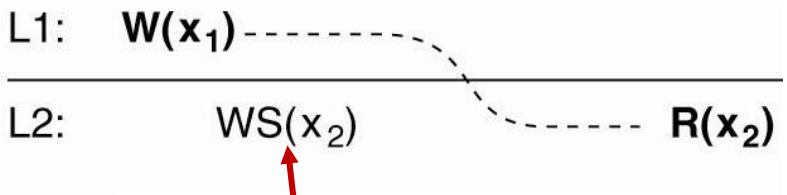
Read-your-writes consistency

- A process will never see a previous value of *x* after a write to that data item *x*



A data store that provides read-your-writes

L2 performs updates to the last write

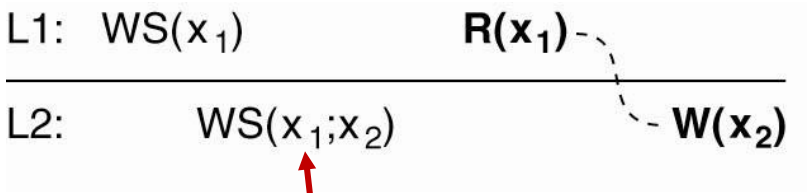


A data store that does not

L2 is not updated to the last write

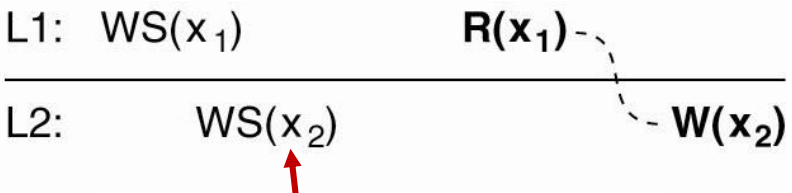
Write-follow-reads consistency

- A write to *x* following a previous read by the same process, is guaranteed to take place on the same or a more recent value of *x* that was read



A writes-follow-reads consistent data store

Write operations are moved to L2



A data store that does not

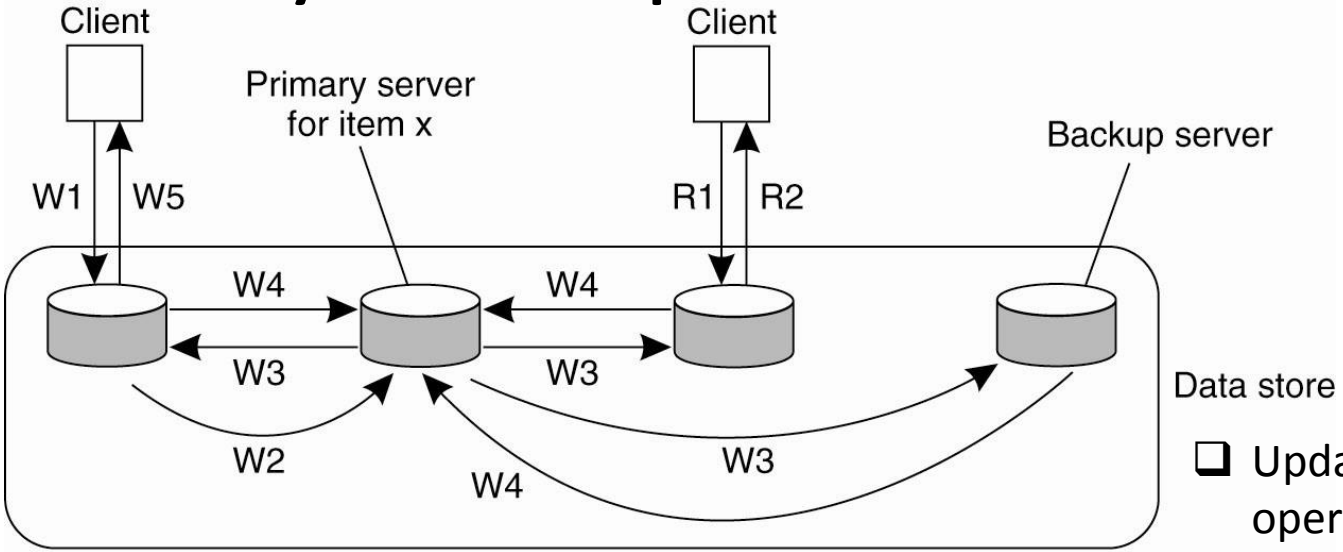
The writes from L2 are not consistent with those in L1

Consistency protocols

Consistency protocols

- Describes an implementation of a specific consistency model
- Sequential consistency
  - Passive replication → remote-write protocols and local-write protocols (primary-based)
  - Active replication → sequencer and quorum-based protocols

Primary-based protocol: remote-write

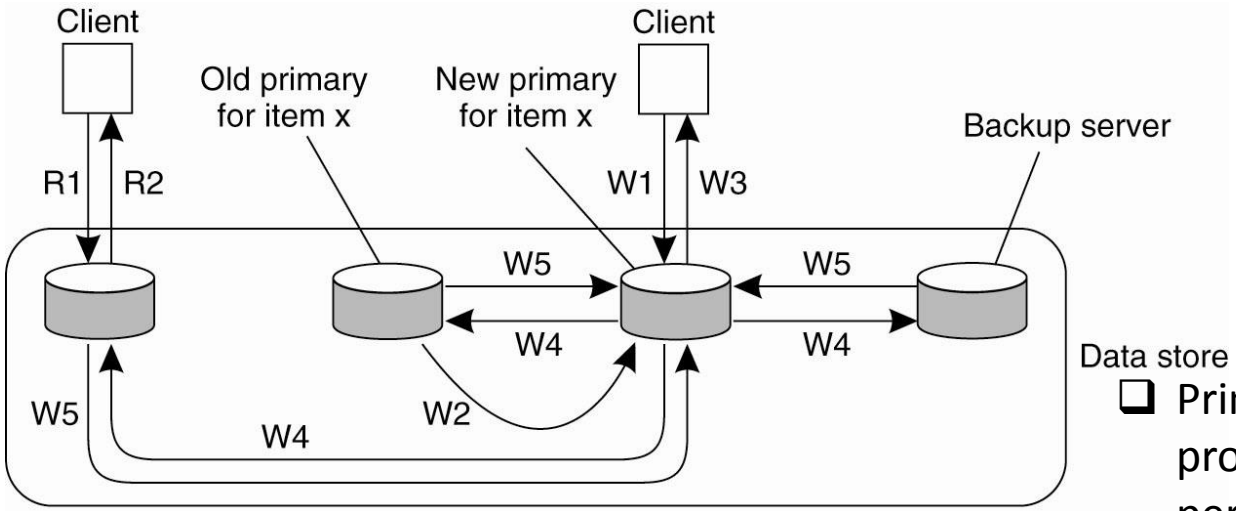


- Updates are blocking operations
  - non-blocking operations improve performance but,

problem → Fault tolerance

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.28**

Primary-based protocol: local-write



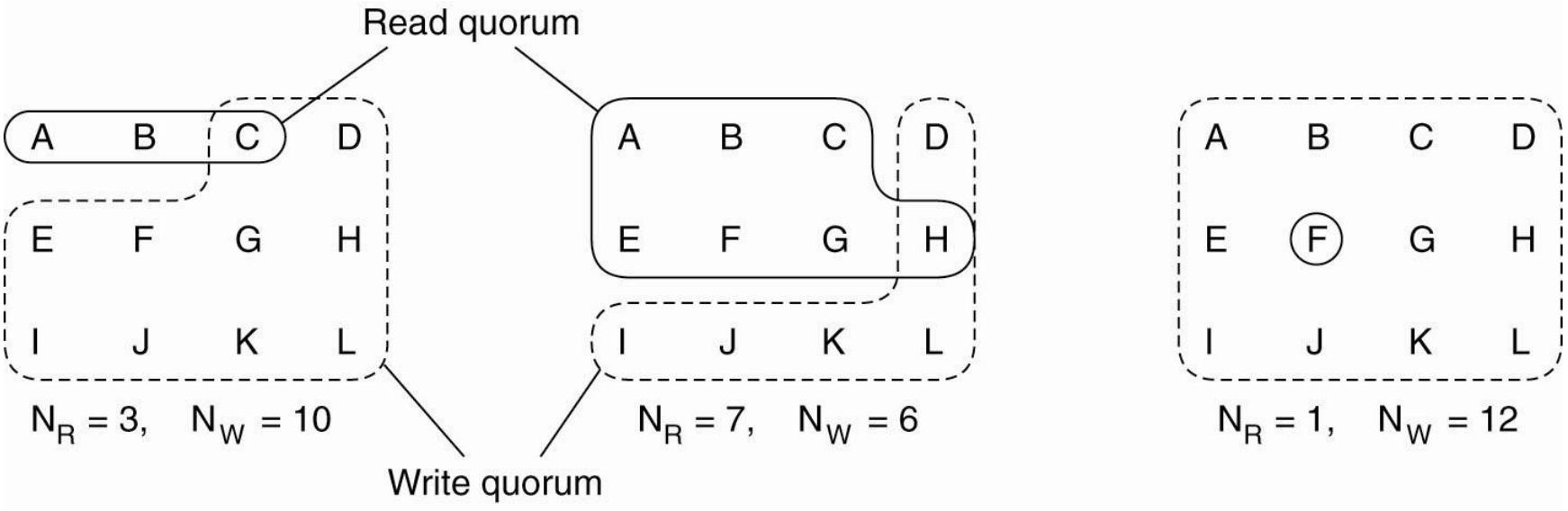
- Primary migrates between processes that wish to perform an operation
- Optimization → carry out multiple successive writes locally
  - But the requests need to be non-blocking

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.30**

Active replication: quorum-based

- Clients need to request and acquire permission from replicas before reading (*read quorum*) or writing (*write quorum*)
- Each data item contains a version number
- Read/write requires agreement of a majority
- Constraints for read (N_R) and write (N_W) quorums
  1. $N_R + N_W > N$
  2. $N_W > N/2$

Quorum-based example



Correct choice of N_R & N_W

write-write conflict

ROWA (read one, write all)

Figure adapted from Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, (c) 2002 Prentice-Hall, Inc.- based on Figure **6.33**

Summary

- Consistency models establish the rules on how a data store operates, models differ …
  - In how restrictive they are
  - How complex their implementations are
  - Ease of programming
  - Performance
- Implementation of stronger consistency models is expensive
- Weaker models have less constraints and are cheaper to implement
- Data-centric models
  - Strict, linearizability, sequential, causal

- Client-centric models
  - Eventual consistency
    - Monotonic reads, monotonic writes, read your writes, writes follow reads
- Consistency protocols describes an implementation of a consistency model
  - Primary-based protocols (passive)
    - Remote-write
    - Local-write
  - Quorum-based protocols (active)

Next Lecture

Cassandra