# Lecture 6

# Consistency and Replication

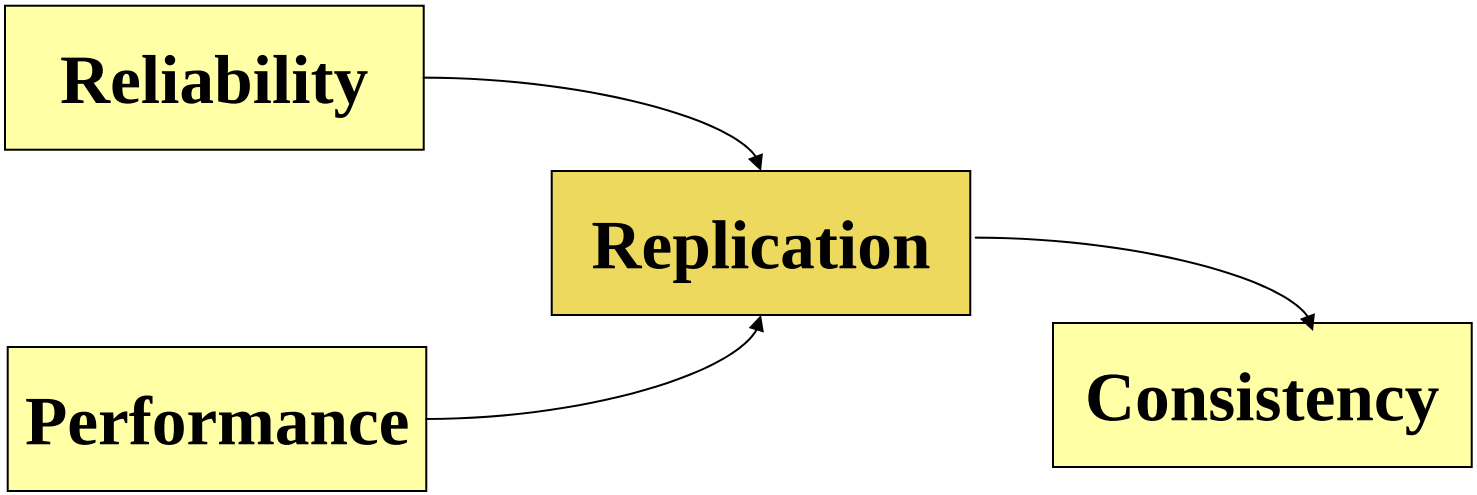
Prof. Wilson Rivera

University of Puerto Rico at Mayaguez
Electrical and Computer Engineering Department

# Outline

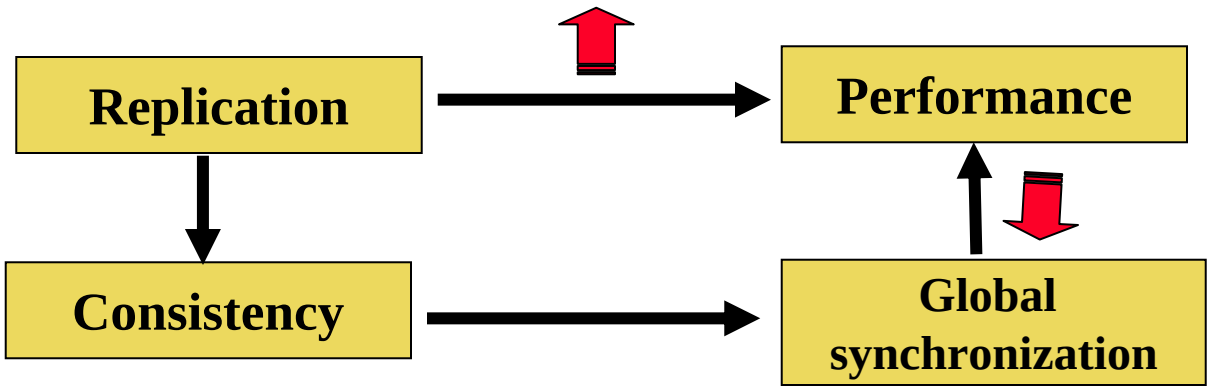
- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols

# Replication




If objects (or data) are shared, we need to do something about concurrent accesses to guarantee state consistency.

# Performance and scalability




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

# CAP Theorem

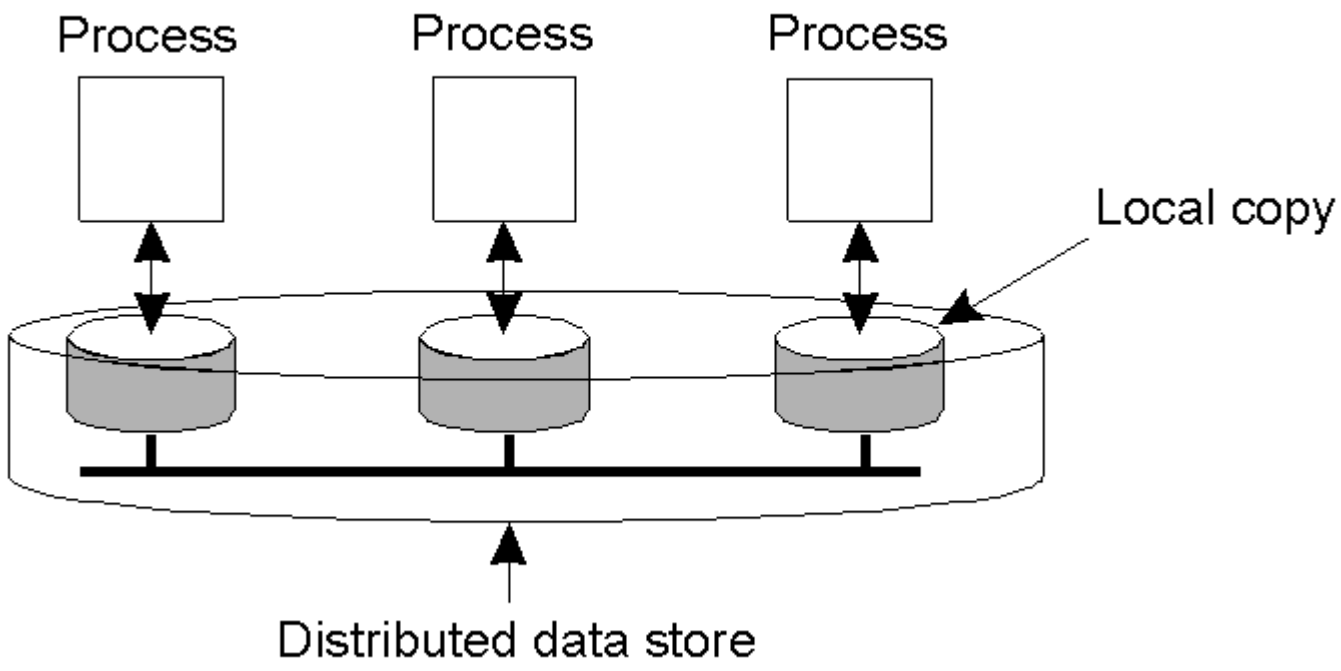
- **CAP Theorem; Brewer (2000)**
  - **States that of three properties of shared-data systems:**
    - » **data consistency,**
    - » **system availability, and**
    - » **tolerance to network partition**
  - **only two can be achieved at any given time.**

W. Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.

# Consistency Models

- 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.
  - Data-centric consistency models
  - Client-centric consistency models

# Data-Centric Consistency Models (1)




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

# 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
  - Entry consistency
- The weaker the consistency model, the easier it is to build a scalable solution.

# Strict Consistency (1)

- *Any read to a shared data item X returns the value stored by the most recent write operation on X.*
  - It doesn't make sense to talk about "the most recent" in a distributed environment.
  - Strict consistency is what you get in the normal sequential case, where your program does not interfere with any other program.
  - All writes are instantaneously visible to all processes and absolute global time is maintained

# Strict Consistency (2)

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

(a)

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

(b)

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

a) A strictly consistent store.

b) A store that is not strictly consistent.

# Sequential Consistency (1)

- *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.*

# Sequential Consistency (2)

| | | | | |
|---|---|---|---|---|
| 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)

a) A sequentially consistent data store.

b) A data store that is not sequentially consistent.

# Causal Consistency (1)

- *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes (not causally related) may be seen in a different order by different processes.*

# Causal Consistency (2)

| | | | | | | |
|---|---|---|---|---|---|---|
| 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.
- W(x)b & W(x)c are concurrent so it is not required to all processes see them in the same order

# Causal Consistency (3)

| | | | | |
|---|---|---|---|---|
| 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 (W(x)b depends on W(x)a since the value of a is read to compute b).

b) A correct sequence of events in a causally-consistent store (R(x)a is removed then W(x)a & W(x)b are concurrent).

# FIFO Consistency (1)

- *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.*

# FIFO Consistency (2)

| | | | | | | |
|---|---|---|---|---|---|---|
| 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

# Weak Consistency (1)

- Introduce synchronization variables
  - Accesses to synchronization variables associated with a data store are sequentially consistent
  - No access on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere
  - No access to data items are allowed to be performed until all previous operations to synchronization variables have been performed.
- You don't care that reads and writes of a *series* of operations are immediately known to other processes. You just want the *effect* of the series itself to be known.
- Here sequential consistency is enforced between groups of operations instead of between individual operations

# Weak Consistency (2)

| | | | | | | |
|---|---|---|---|---|---|---|
| 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) A valid sequence of events for weak consistency.

b) An invalid sequence for weak consistency.

# Entry Consistency (1)

- 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.

# 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.

# Summary of Consistency Models

| Consistency | Description |
|---|---|
| Strict | Absolute time ordering of all shared accesses matters. |
| 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 |
| | **SYNCHRONIZATION VARIABLES** |
| Weak | Shared data can be counted on to be consistent only after a synchronization is done |
| Entry | Shared data pertaining to a critical region are made consistent when a critical region is entered. |

# Client-Centric Consistency Models

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

- Eventual consistency
  - If no updates take place for a long time, all replicas will gradually become consistent (a relatively high degree of inconsistency).
  - Works fine as long as clients always access the same replica (e.g. mobile user accessing a distributed data base)

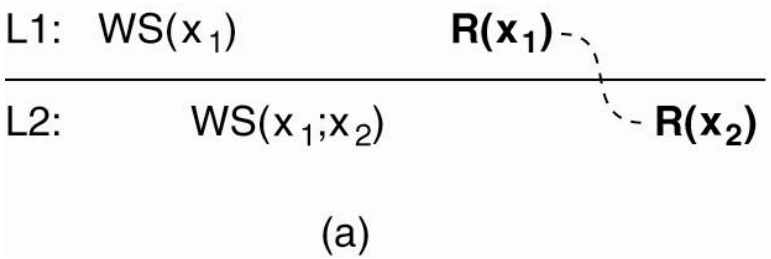
- Client centric consistency provides guarantee for a single client concerning the consistency of access to a data store by that client. No guarantees are given concerning access by different clients
  - Monotonic reads
  - Monotonic writes
  - Read-your-writes
  - Write-follows-reads

# Monotonic Reads (1)

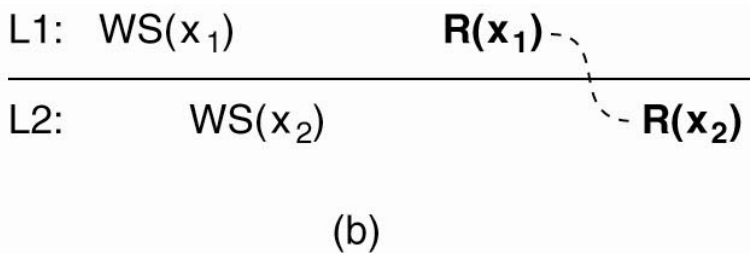
- *If a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value.*
  - **Example**: Each time you connect to a different email server, the server fletches all the updates form the server you previously visited.
  - **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.

- *WS(Xi)* denotes the set of operations at location Li that leads to version *Xi* of *X* (at time *t*)
- *WS(Xi,Xj)* denotes that operations in *WS(Xi)* has been propagated to location *Lj (that is WS(Xi) is part of WS(Xj))*

# Monotonic Reads (2)




(a)




(b)

- The read operations performed by a single process *P* at two different local copies of the same data store.

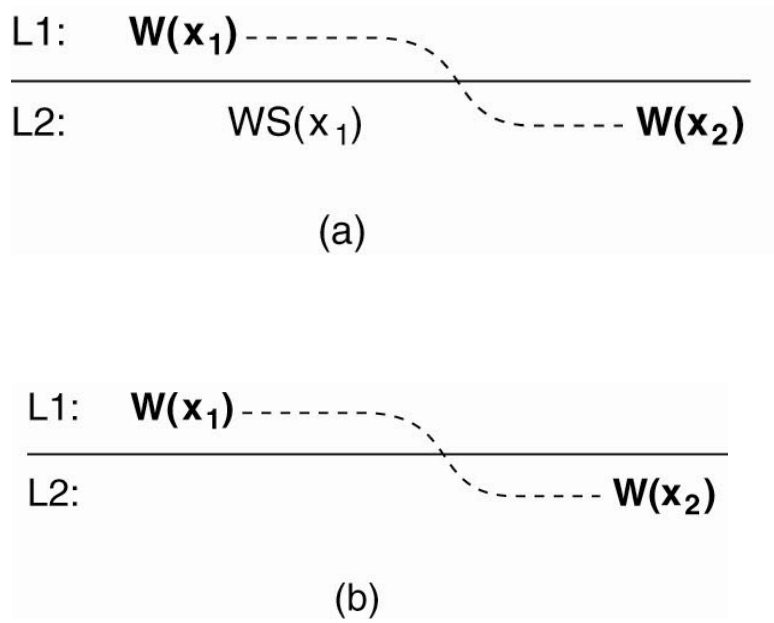
a) A monotonic-read consistent data store

b) A data store that does not provide monotonic reads.

# Monotonic Writes (1)

- *A write operation by a process on a data item x is completed before any successive write operation on x by the same process.*
  - **Example:** Updating a program at server *S*2, and ensuring that all components on which compilation and linking depends, are also placed at *S*2.
  - **Example**: Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

# Monotonic Writes (2)




- The write operations performed by a single process *P* at two different local copies of the same data store

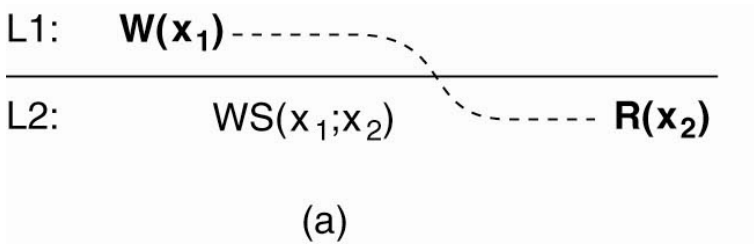
a) A monotonic-write consistent data store.

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

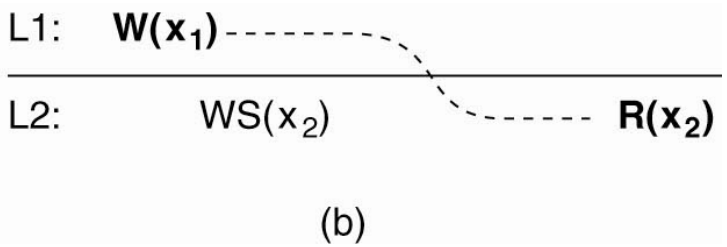
# Read Your Writes (1)

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

# Read Your Writes (2)




(a)




(b)

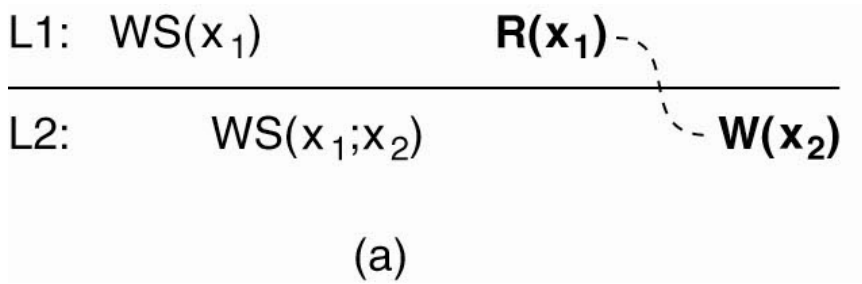
a) A data store that provides read-your-writes consistency.

b) A data store that does not.

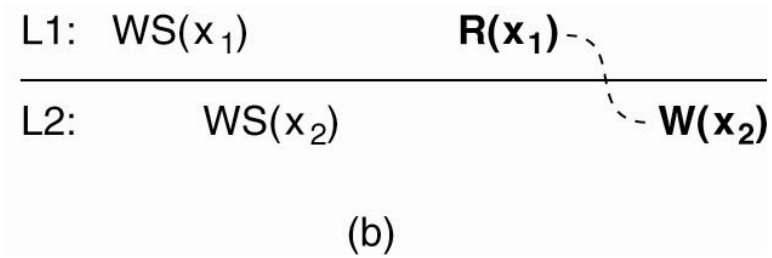
# Writes Follow Reads (1)

- *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.*
  - **Example:** See reactions to posted articles only if you have the original posting (a read “pulls in” the corresponding write operation).

# Writes Follow Reads (2)




(a)




(b)

a) A writes-follow-reads consistent data store

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

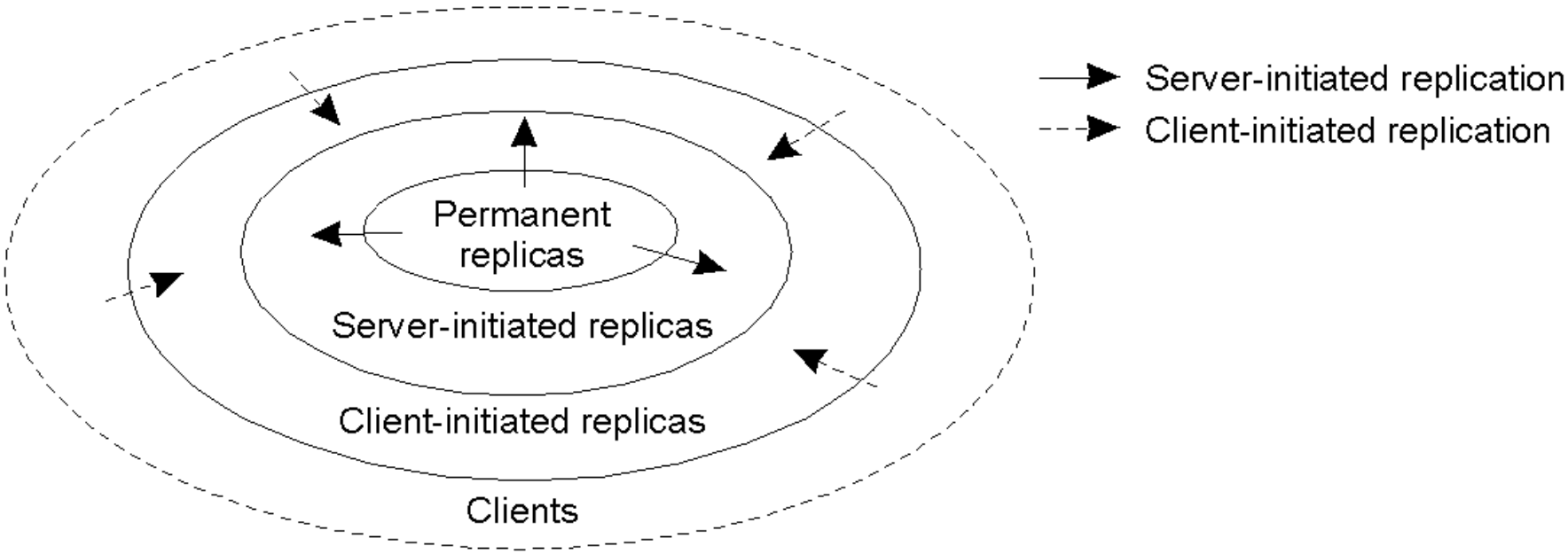
# Replica Management

- Replica Placement
- Update Propagation
- Epidemic Protocols

# Replica Placement (1)

- **Model:** We consider objects (and don't worry whether they contain just data or code, or both)
- **Distinguish different processes:** A process is capable of hosting a replica of an object or data:
  - **Permanent replicas:** Process/machine always having a replica
  - **Server-initiated replica:** Process that can dynamically host a replica on request of another server in the data store
  - **Client-initiated replica:** Process that can dynamically host a replica on request of a client (client cache)

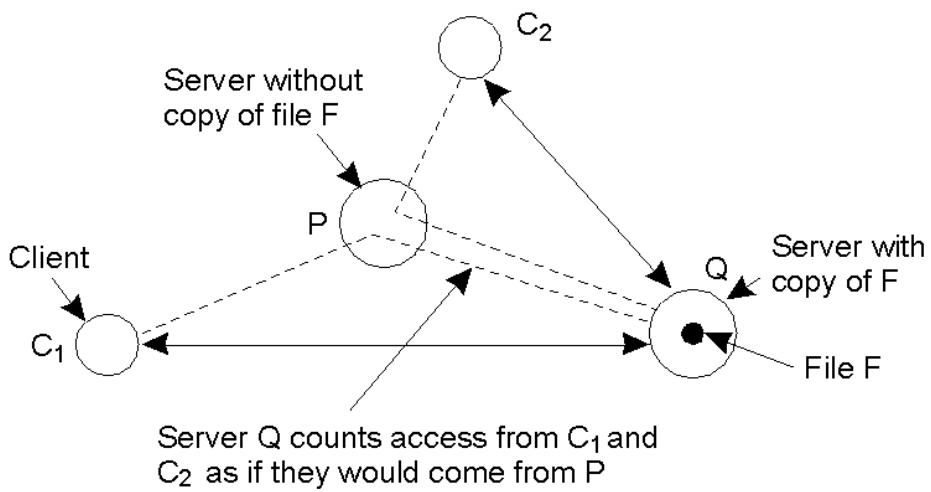
# Replica Placement (2)




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

# Server-Initiated Replicas

- Keep track of access counts per file, aggregated by considering server closest to requesting clients
  - Number of accesses drops below threshold *D* → delete file
  - Number of accesses exceeds threshold *R* → replicate file
  - Number of access between *D* and *R* → migrate file



# Update Propagation (1)

- Propagate only notification (invalidation protocol)
  - Small read-to-write ratio
  - Little network bandwidth required
- Transfer data from one copy to another
  - High read-to-write ratio
- Propagate the update *operation* (active replication)
  - Sending only parameter values the update operation needs could save bandwidth
  - However, more processing power may be required by each replica

# Update Propagation (2)

- Pushing updates: server-initiated approach, in which update is propagated regardless whether target asked for it.
  - High degree of consistency due to relatively high read-to-write ratio at each replica
- Pulling updates: client-initiated approach, in which client requests to be updated
  - Low degree of consistency
  - Response time increases in case of cache misses

| 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 |

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

# Update Propagation (3)

- We can dynamically switch between pulling and pushing using **leases**
  - A contract in which the server promises to push updates to the client until the lease expires.
- Make lease expiration time dependent on system's behavior (adaptive leases)
  - Age-based leases
    - » An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
  - Renewal-frequency based leases
    - » The more often a client requests a specific object, the longer the expiration time for that client will be
  - State-based leases
    - » The more loaded a server is, the shorter the expiration times become

# Consistency protocols

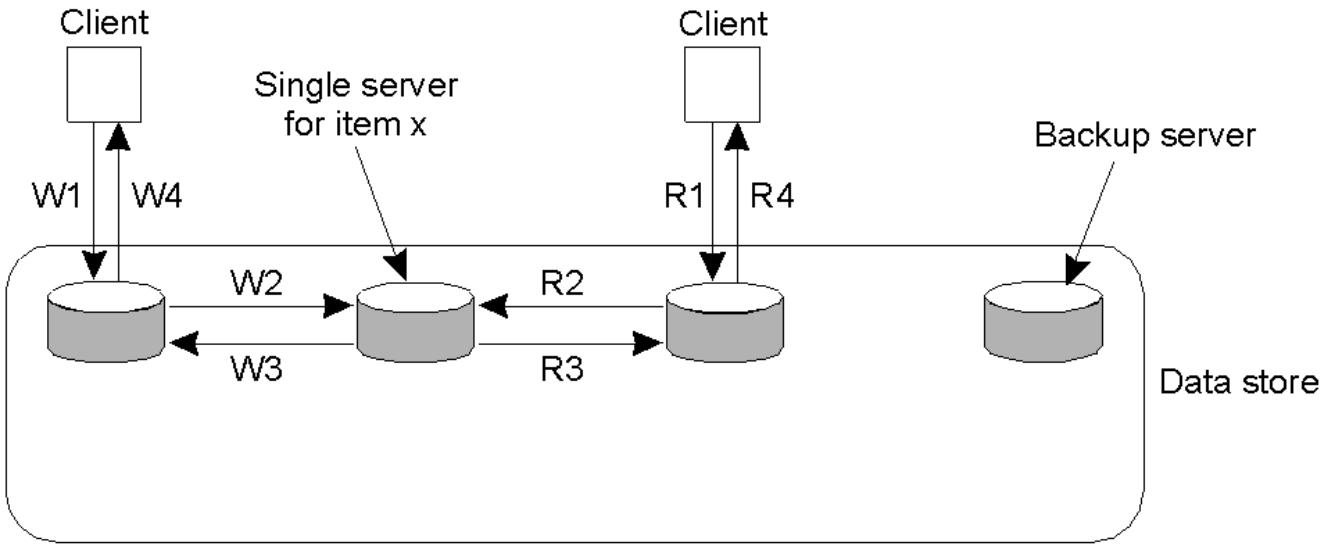
- Primary-based protocols
- Replicated-write protocols

- By and large, consistency models in which operations are globally serialized are the most important and widely applied modes. This include sequential consistency, weak consistency with synchronization variables as well as atomic transactions

# Primary Based Protocols (1)

- Each data item X in the data store has an associated primary, which is responsible for coordinating write operations on X
  - **Primary-based, remote-write, fixed server**
    - All read and write operations are carried out a a remote single server
    - Used in traditional client-server systems that do not support replication.
  - **Primary-backup protocol**
    - Allow processes to perform read operations on a locally available copy but should forward write operations to a (fixed) primary copy
    - Traditionally applied in distributed databases and file systems that require a high degree of fault tolerance. Replicas are often placed on same LAN.
  - **Primary-based, local-write protocol**
    - Establishes only a fully distributed, non-replicated data store.
    - Useful when writes are expected to come in series from the same client (e.g., mobile computing without replication)
  - **Primary-backup protocol with local writes**
    - The primary copy migrates between processes that wish to perform a write operation
    - Distributed shared memory systems

# Primary Based Protocols (2)

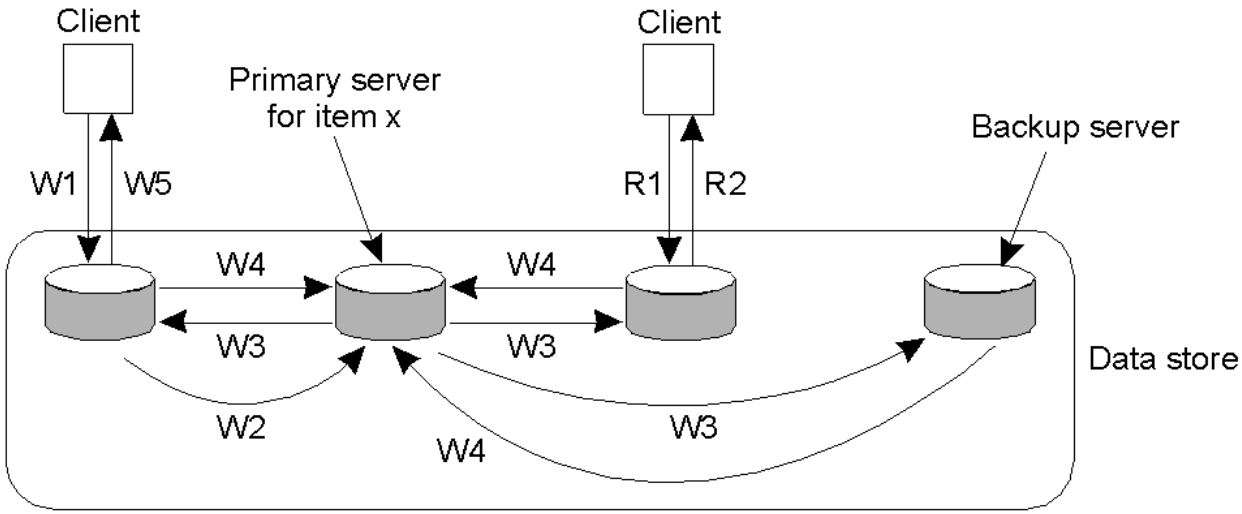



W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

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 Protocols (3)

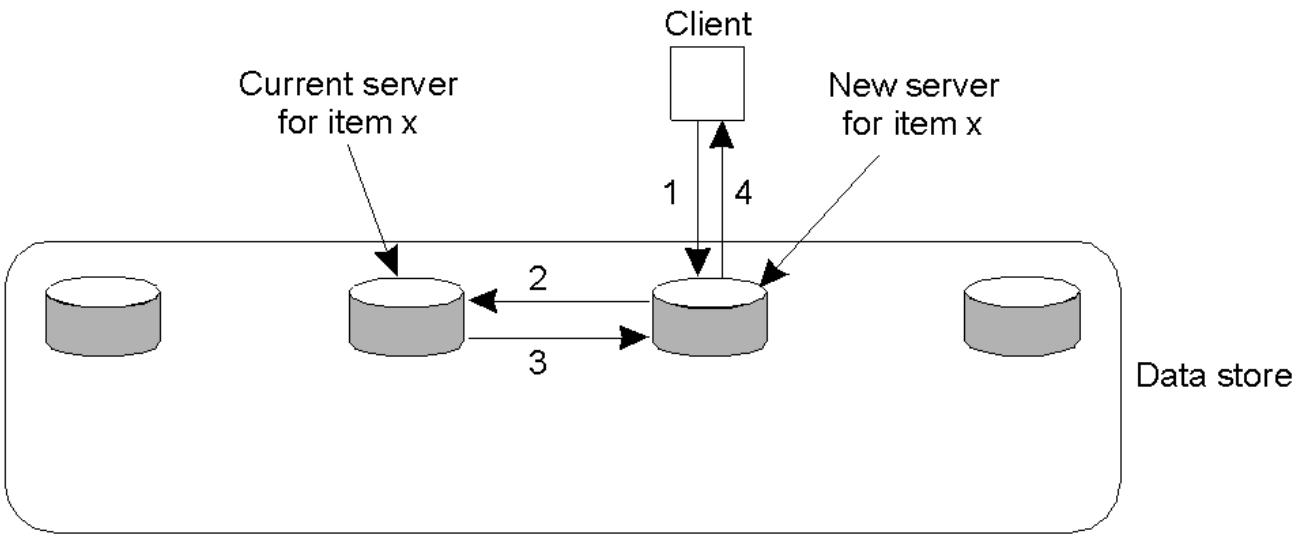



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

- The principle of primary-backup protocol.

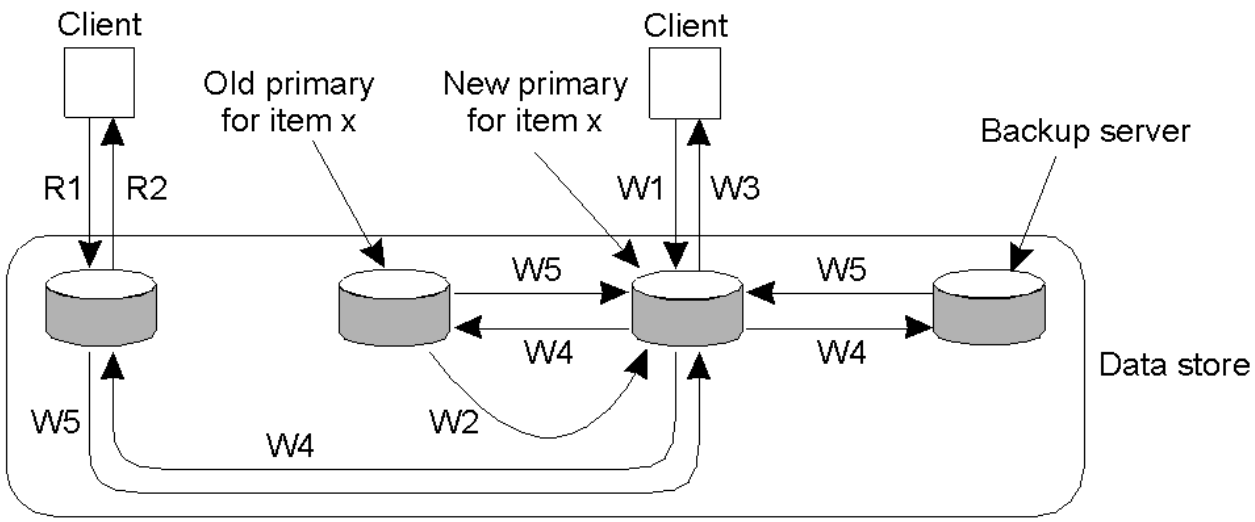
# Primary Based Protocols (4)




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.

# Primary Based Protocols (5)




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.

# Replicated Write Protocols

- Write operations can be carried out at multiple replicas instead of only one, as in the case of primary-based replicas
  - **Active replication**
    - » Updates are forwarded to multiple replicas, where they are carried out.
    - » Operations need to be carried out in the same order everywhere
    - » The solution to this problem is to assign a coordinator on each side (client and server), which ensures that only one invocation, and one reply is sent
  - **Quorum-based protocols**
    - » Ensure that each operation is carried out in such a way that a majority vote is established
    - » Distinguish read quorum and write quorum

# Summary

- Data-centric consistency
  - Strict, sequential, causal, FIFO
  - Weak consistency (synchronization)
- Client-centric consistency
  - Eventual consistency: monotonic read, monotonic write, read-your-writes, write-follows-read
- Replica management
  - Replica placement: permanent, server, client
  - Update propagation: invalidation protocol, transfer data, active replication
  - Update propagation: pushing, pulling
- Consistency protocols
  - Primary based protocol: primary backups
  - Replicated write protocols: active replication, quorum-based