Exam and Term Project

- Exam
  - Exam, Monday, April 8, 2024
  - Time: 3:05pm – 5:05pm; Room XXxxx
  - Close book, Close note, no phone, calculator, or computer or any internet access
- Term Project
  - Presentation, Monday and Tuesday, April 22
  - Term project report, April 25, 2024
  - Progress report March 20

Chapter 7 (continuation)

- Data Consistency Model
  - Client-centric models
  - Eventual consistency and Epidemic protocols
- Distribution protocols
  - Invalidate versus updates, Push versus Pull, Cooperation between replicas
- Implementation issues (consistency protocols)
  - Primary-based, Replicated-write, Cache-coherence
- Putting it all together
  - Final thoughts
- Replica placement

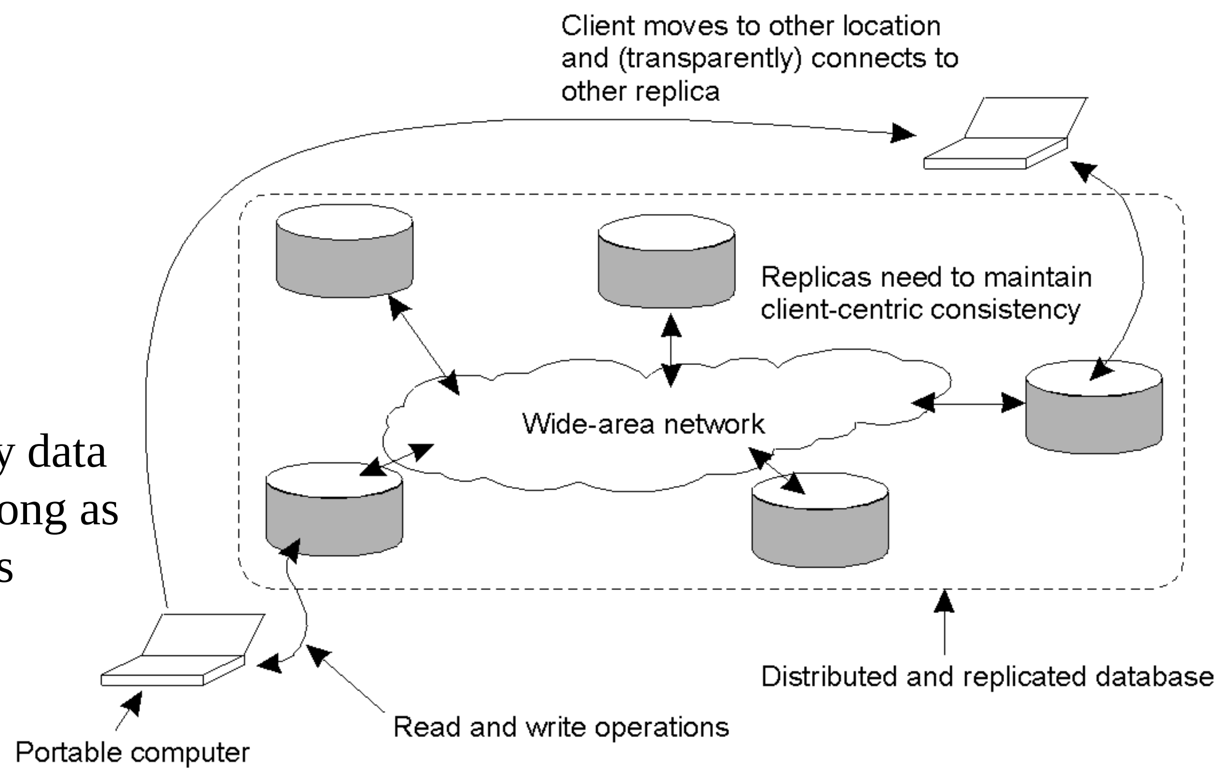
Eventual Consistency

- Assume a replicated database with few updaters and many readers
- *Eventual consistency*:
  - Definition: if no more updates, all replicas will gradually become consistent
  - Only requirement: guaranteed propagate
  - Cheap to implement: ?
  - Things work fine as long as user accesses the same replica
  - What if they don’t? Mobile users?

Client-Centric Consistency

- Eventually consistent stores require **only** that updates are guaranteed to propagate to all replicas. Conflicts due to concurrent writes are often easy to resolve. Cheap implementation
- Problems arise when different replicas are accessed by the same process at different time → Consistency for a single client

Eventual consistency data stores work fine as long as clients always access the same replica

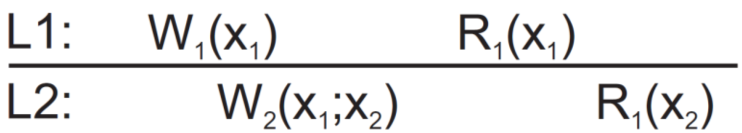


Models of Client-centric Consistency

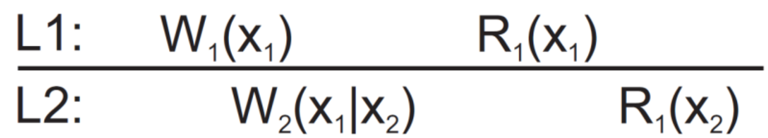
- Monotonic-reads
  - If a process reads the value of a data item x, any successive read on x by that process will never return an older value of x.
  - E.g. Mail read from a mbox in SF will also be seen in mbox of NY
- Monotonic-writes
  - A write by a process on a data item x is completed before any successive write on x by the same process.
  - Similar to FIFO consistency, but monotonic-write model is about the behavior of a single process.
- Read Your Writes
  - A write on data item x by a process will always be seen by a successive read on by the same process.
  - E.g. Integrated editor and browser
- Writes follow reads
  - A write on x by a process following its previous read of x is guaranteed to take place on the same or a more recent value of x that was read

Monotonic Reads

- Successive read return the same or a more recent value
- email



(a)



(b)

(a) A monotonic-read consistent data store
(b) No, not sure $W(x_1)$ is a part of $W(x_2)$

Monotonic Writes

- Write performed by the processor must keep in order (in any where)
- Software library

L1: $W_1(x_1)$

L2: $W_2(x_1;x_2)$ $W_1(x_2;x_3)$

(a)

L1: $W_1(x_1)$

L2: $W_2(x_1|x_2)$ $W_1(x_1|x_3)$

(b)

L1: $W_1(x_1)$

L2: $W_2(x_1|x_2)$ $W_1(x_2;x_3)$

(c)

L1: $W_1(x_1)$

L2: $W_2(x_1|x_2)$ $W_1(x_1;x_3)$

(d)

(a) A monotonic-write consistent data store

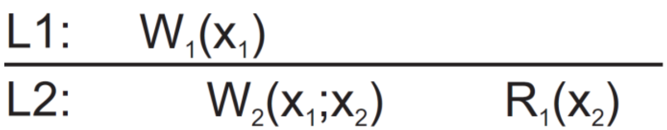
(b) Does not support monotonic-write consistent

(c) Not,

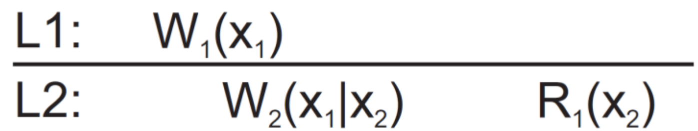
(d) Yes

Read Your Writes

- The write results should see by any successive read by the same process
- Web HTML



(a)



(b)

(a) A data store that provides read-your-writes consistency
(b) Not

Writes Follow Reads

- Write by the same process is guaranteed to take place on the same or a more recent value
- News group

L1:	$W_1(x_1)$		$R_2(x_1)$
L2:		$W_3(x_1;x_2)$	$W_2(x_2;x_3)$

(a)

L1:	$W_1(x_1)$		$R_2(x_1)$
L2:		$W_3(x_1 \mid x_2)$	$W_2(x_1 \mid x_3)$

(b)

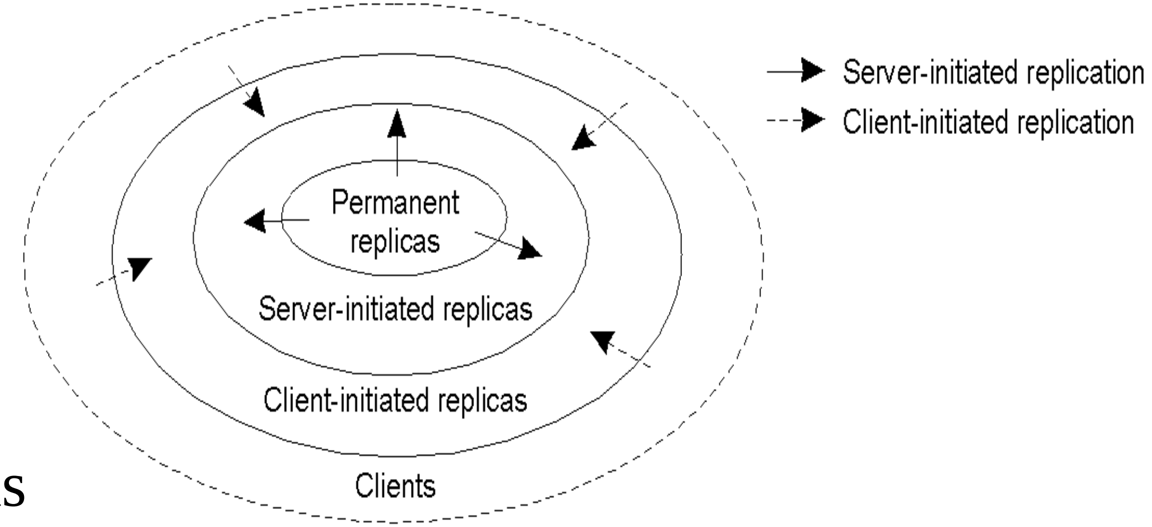
(a) A wites-follow-reads consistent data store
(b) Not

Outline

- Object Replication
- Data Consistency Model
- Implementation Issues
  - Update Propagation
    - Distributing updates to replicas, independent of consistency model
  - Other Consistency Model Specific Issues
- Case Studies

Replica Placement

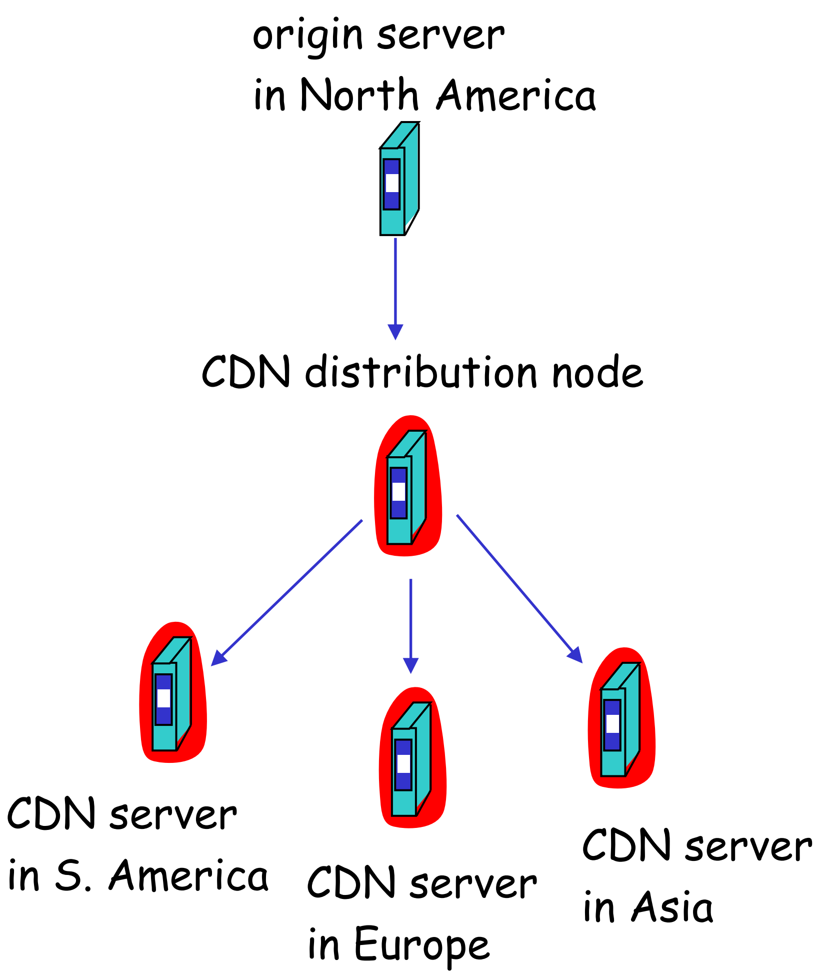
- A major design issue is to decide where, when, and by whom copies of the data store are to be placed.



- Permanent Replicas
  - Initial setup of replicas that constitute a distributed store
  - For examples,
    - Distributed web servers:
    - server mirrors,
    - distributed database systems (shared nothing arch vs federated DB)

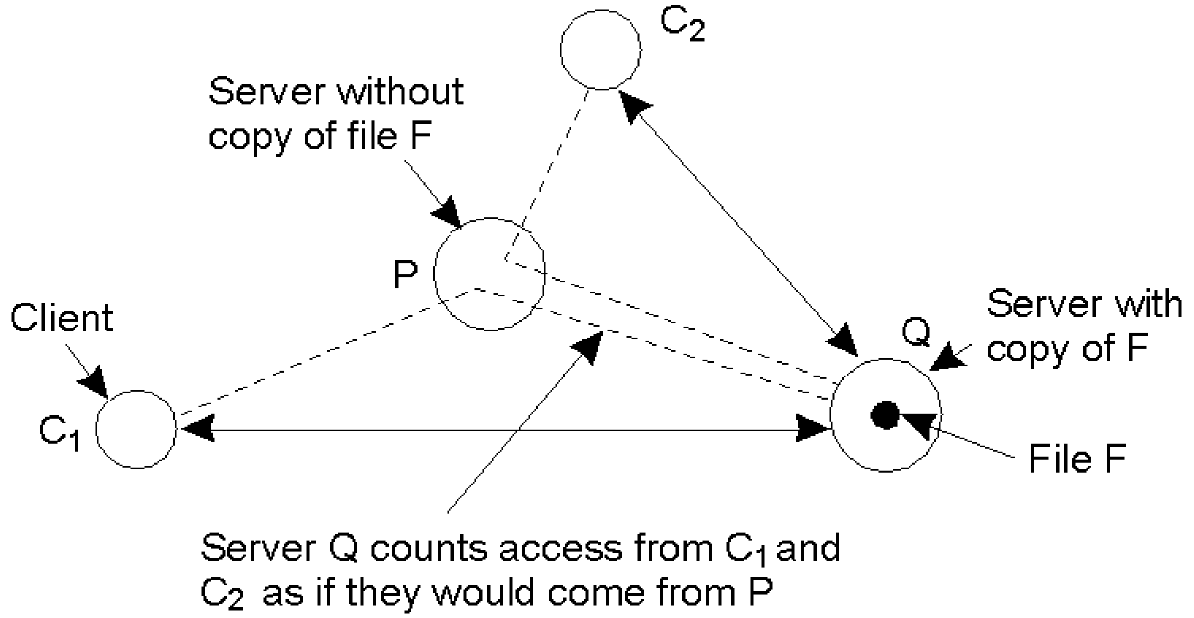
Replica Placement: server-initiated

- Dynamic replica placement of replica is a key to content delivery network (CDN).
  - CDN company installs hundreds of CDN servers throughout Internet
  - CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers
- Key issue: When and where replicas should be created or deleted



CDN Content Placement

- An algorithm (Rabinvoich et al 1999)
  - Each server keeps trace of access counts per file, and where access requests come from.
  - Assume give a client C, each server can determine which of the servers is closest to C.

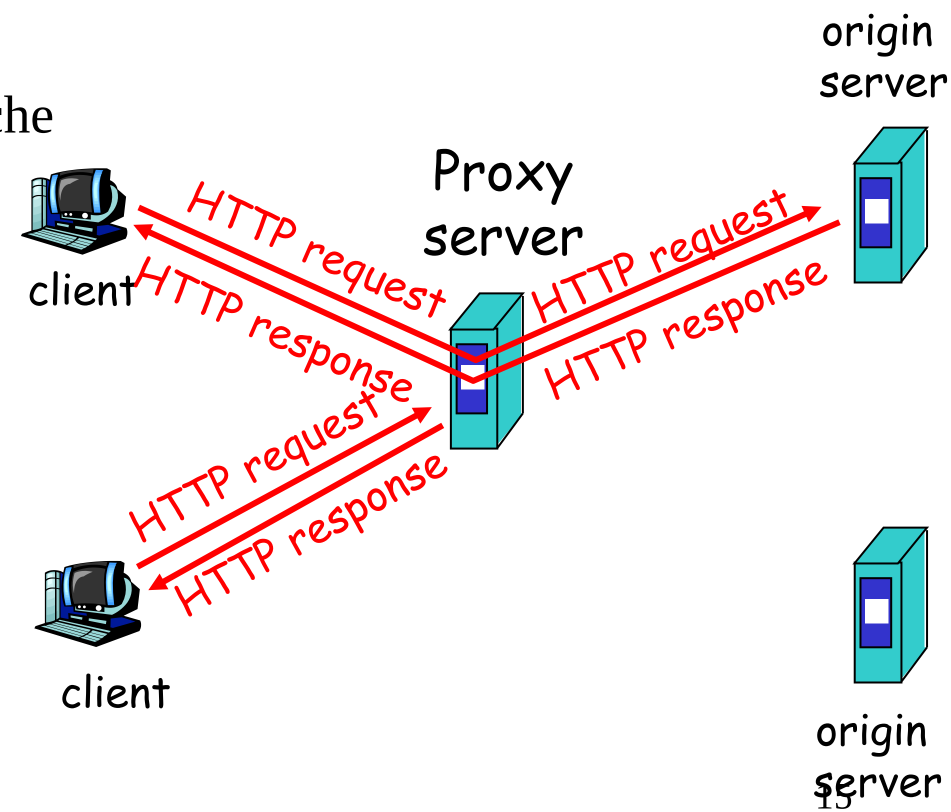


Server-initiated Replication (cont')

- An algorithm (Rabinvoich et al 1999)
  - Initial placement
  - Migration or replication of objects to servers in the proximity of clients that issue many requests for the objects
    - Counting access requests to file F at server S from different client
    - Deletion threshold, replication threshold
  - If #access(S,F) <= del(S,F), remove the file, unless it is the last copy
  - If #access(S,F)>= rep(S,F), duplicate the file somewhere
  - If del(S,F) < #access(S,F) < rep(S,F), migrate

Replica placement: Client-Initiated

- Cache: a local storage to temporarily store a copy of the recently accessed data mainly for reducing request time and traffic on the net
  - Client-side cache (forward cache in Web)
  - Proxy cache
  - Proxy cache network

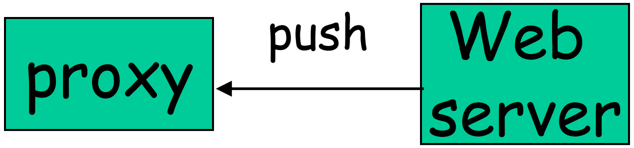


Consistency Issues

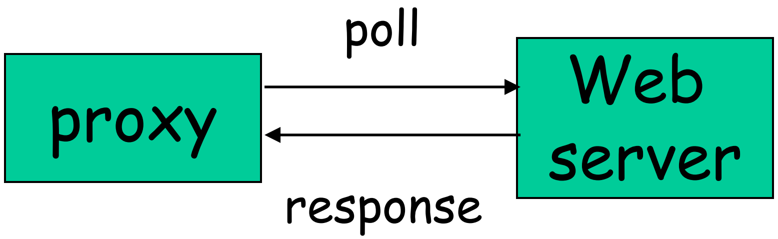
- Web pages tend to be updated over time
  - Some objects are static, others are dynamic
  - Different update frequencies (few minutes to few weeks)
- How can a proxy cache maintain consistency of cached data?
  - Send invalidate or update
  - Push versus pull

Push-based Approach

- Server tracks all proxies that have requested objects
- If a web page is modified, notify each proxy
- Notification types
  - Notification only
  - Transfer the new results
  - Propagate the operation carried
- How to decide between invalidate and updates?
  - Pros and cons?
  - Alternative approach: ?



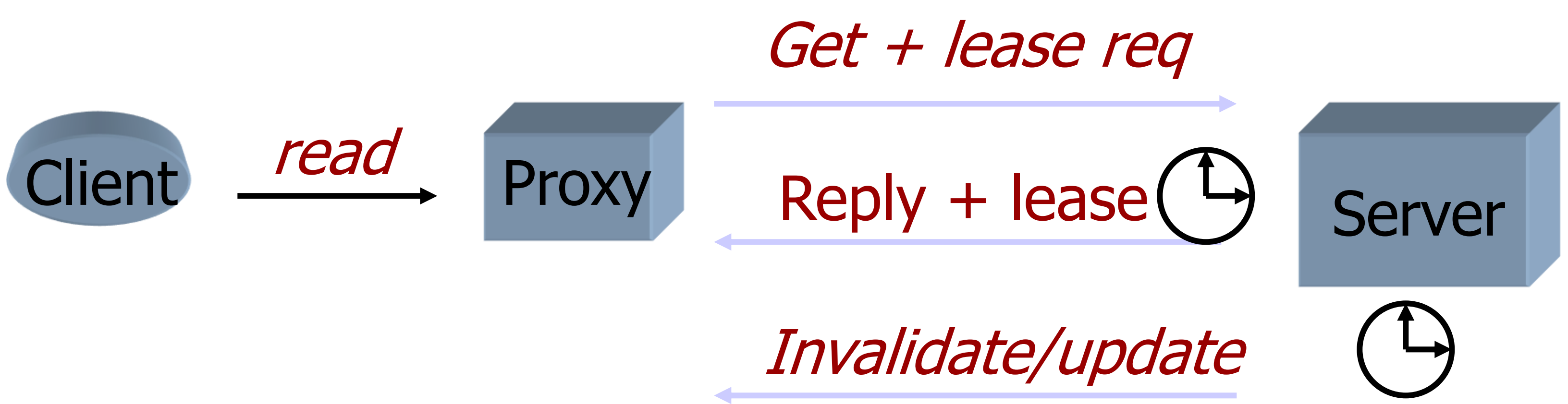
Pull-based Approaches



- Proxy is entirely responsible for maintaining consistency
- Proxy periodically polls the server to see if object has changed
  - Use if-modified-since HTTP messages
- Key question: when should a proxy poll?

A Hybrid Approach: Leases

- Lease: duration of time for which server agrees to notify proxy of modification
- Server issues lease on first request and sends notification until expiry
  - Need to renew lease upon expiry
- Efficiency depends on the *lease duration*
  - Zero duration => ?
  - Infinite leases => ?
  - *Criterion: age, client activity, server utilization*



Update Propagation

- Invalidation vs Update Protocols
  - In invalidation protocol, replicas are invalidated by a small message
  - In update protocol, replicas are brought to up to date by providing with modified data, or specific update operations (active replication)
- Pull vs Push Protocols
  - Push-based (server-based) for appl with high read-to-update ratios (why?)
  - Pull-based (client-based) is often used by client caches
- Unicasting versus Multicasting

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

Push vs Pull in a multiple-clients-single-server system

Lease-based Update Propagation

- [Gray&Chariton'89]: A lease is a server promise that updates will be pushed for a specific time. When a lease expires, the client is forced to
  - pull the modified data from the server, if exists, or
  - requests a new lease for pushing updates
- [Duvvuri et al'90]: Flexible lease system: the lease period can be dynamically adapted
  - age-based lease, based on the last time the item was modified. Long lasting leases to inactive data items.
  - renewal-frequency based lease. Long term lease be granted to clients whose caches need to be refreshed often.
  - Based on server-side state-space overhead. Overloaded server should reduce the lease period so that it needs to keep track of fewer clients as leases expire more quickly.
- [Yin et al'99] Volume lease on objects as well as volumes (i.e. collections of related objects)

The Rest of Chapter 7

- Implementation issues (consistency protocols)
  - Primary-based
  - Replicated-write
  - Cache-coherence
- Putting it all together
  - Final thoughts
- Replica placement

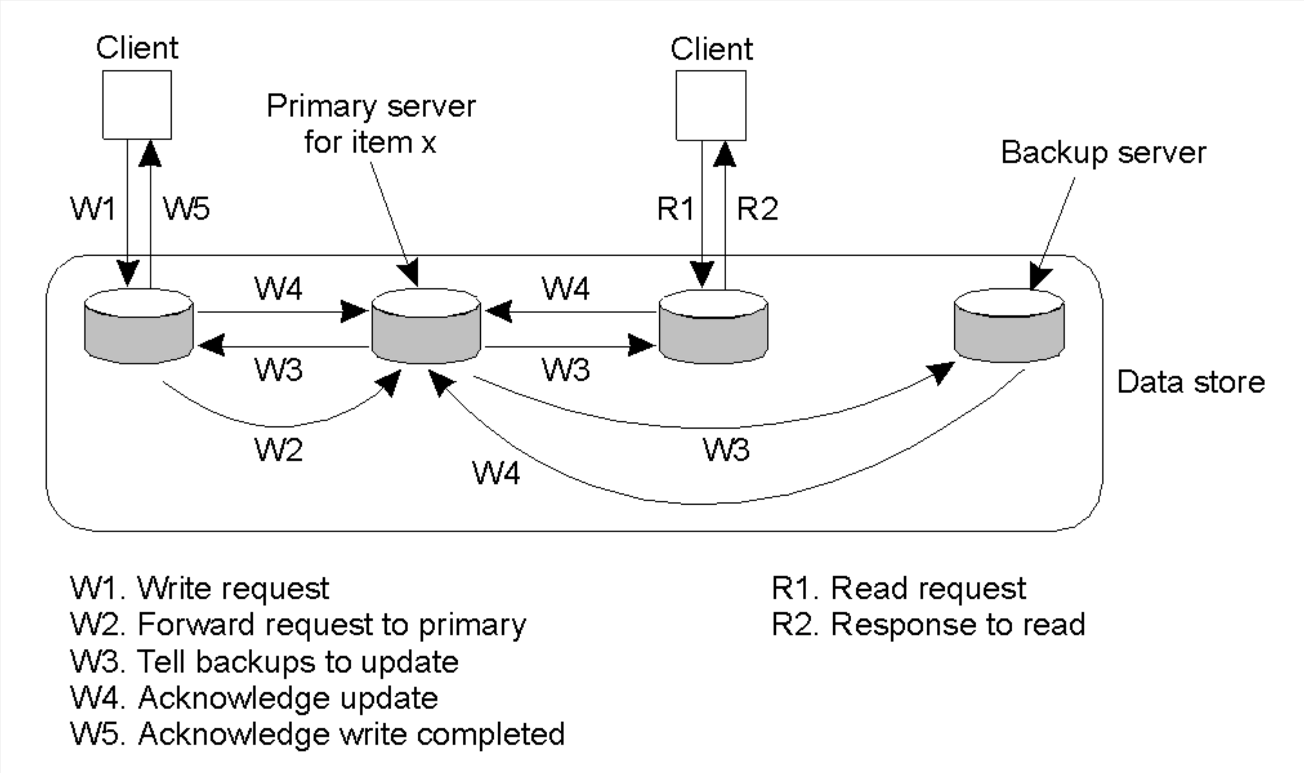
Impl. Issues of Consistency Model

- Passive Replication (Primary-Backup Organization)
  - Single primary replica manager at any time and one or more secondary replica manager
  - Writes can be carried out only the primary copy
  - Two Write Strategies:
    - Remote-Write Protocols
    - Local-Write Protocols
- Active Replication:
  - There are multiple replica managers and writes can be carried out at any replica
- Cache Coherence Protocols

Consistency Model of Passive Replication

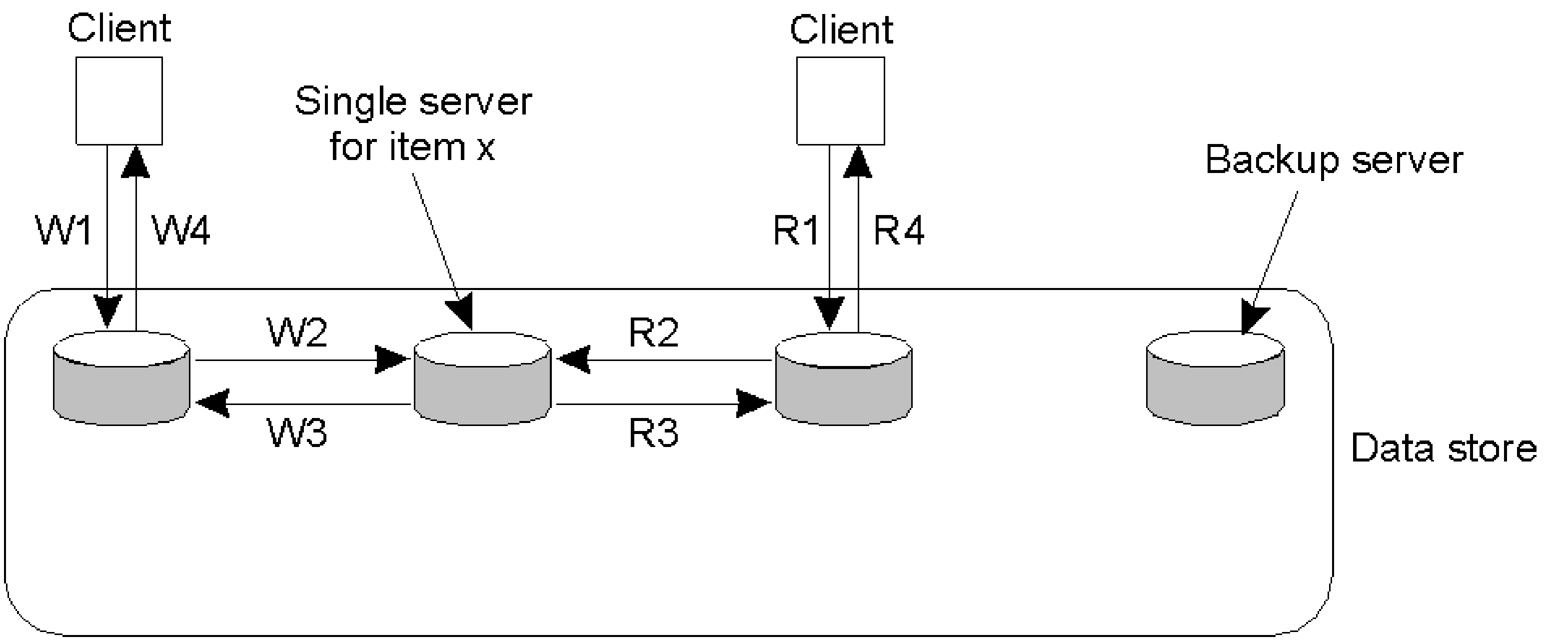
- Blocking update, waiting till backups are updated
  - Blocking update of backup servers must be atomic so as to implement sequential consistency as the primary can sequence all incoming writes and all processes see all writes in the same order from any backup servers.
- Nonblocking, returning as soon as primary is updated
  - what happens if backup fails after update is acknowledged
  - Consistency model with non-blocking update ??
- Atomic multicasting in the presence of failures!
  - Primary replica failure
  - Group membership change
  - Virtual Synchrony Implementation

Passive Replication: Remote Write



- Write is handled only by the remote primary server, and the backup servers are updated accordingly; Read is performed locally.
- Performance (blocking) and fault tolerance (non-blocking)
- E.g. Sun Network Information Service (NIS, formerly Yellow Pages)

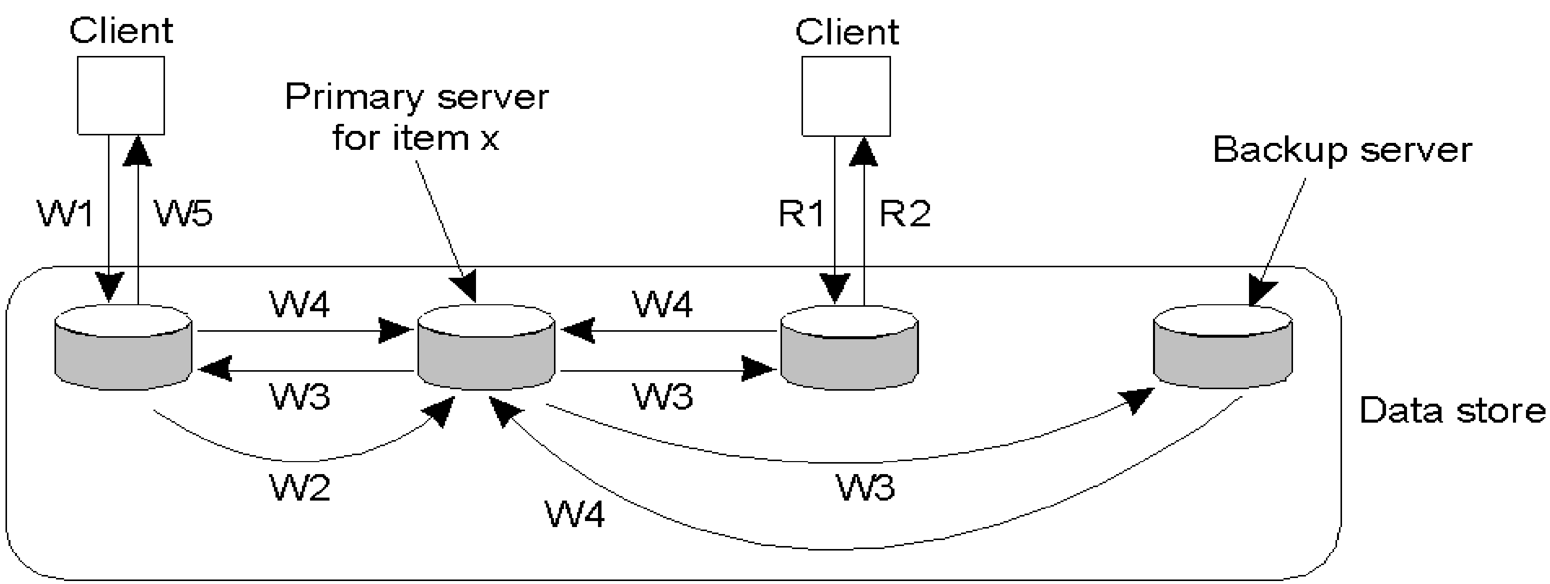
Remote-Write Protocols (non-blocking)



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

Remote-Write Protocols (blocking)

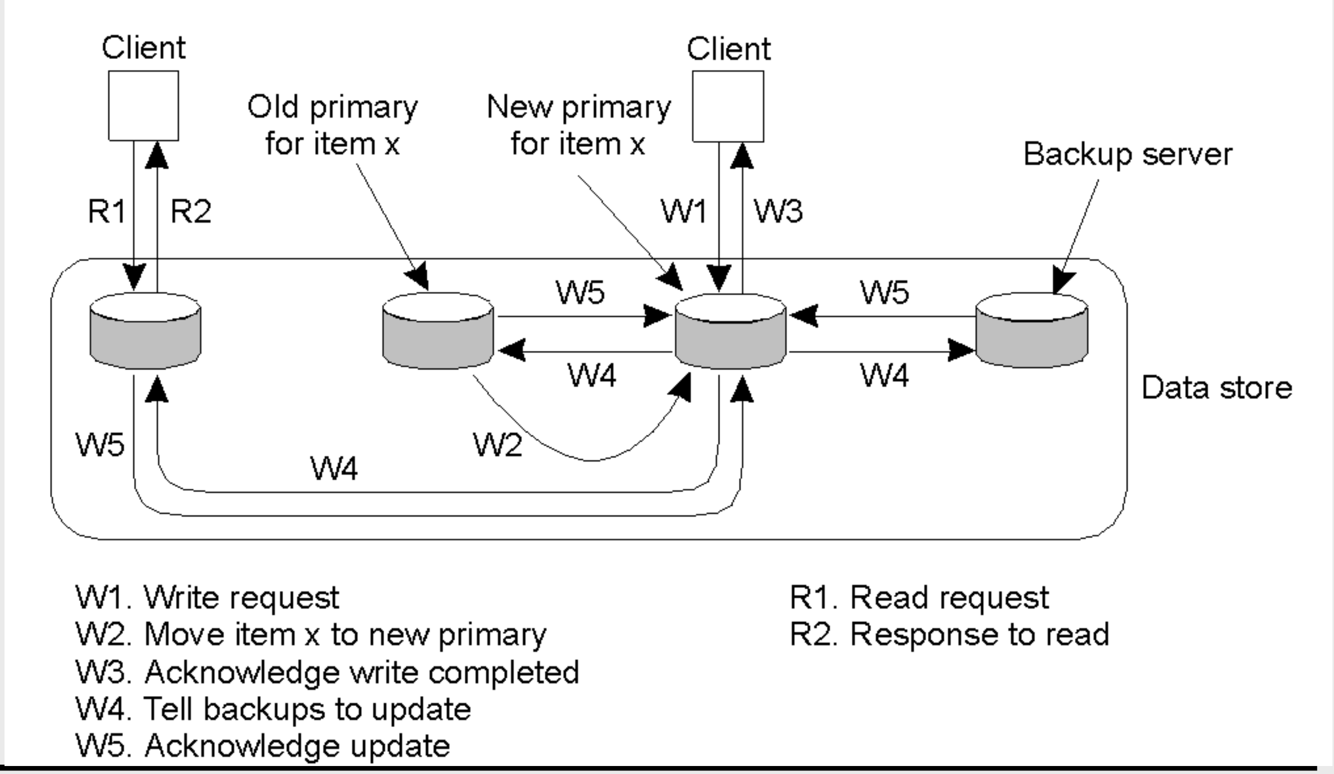


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

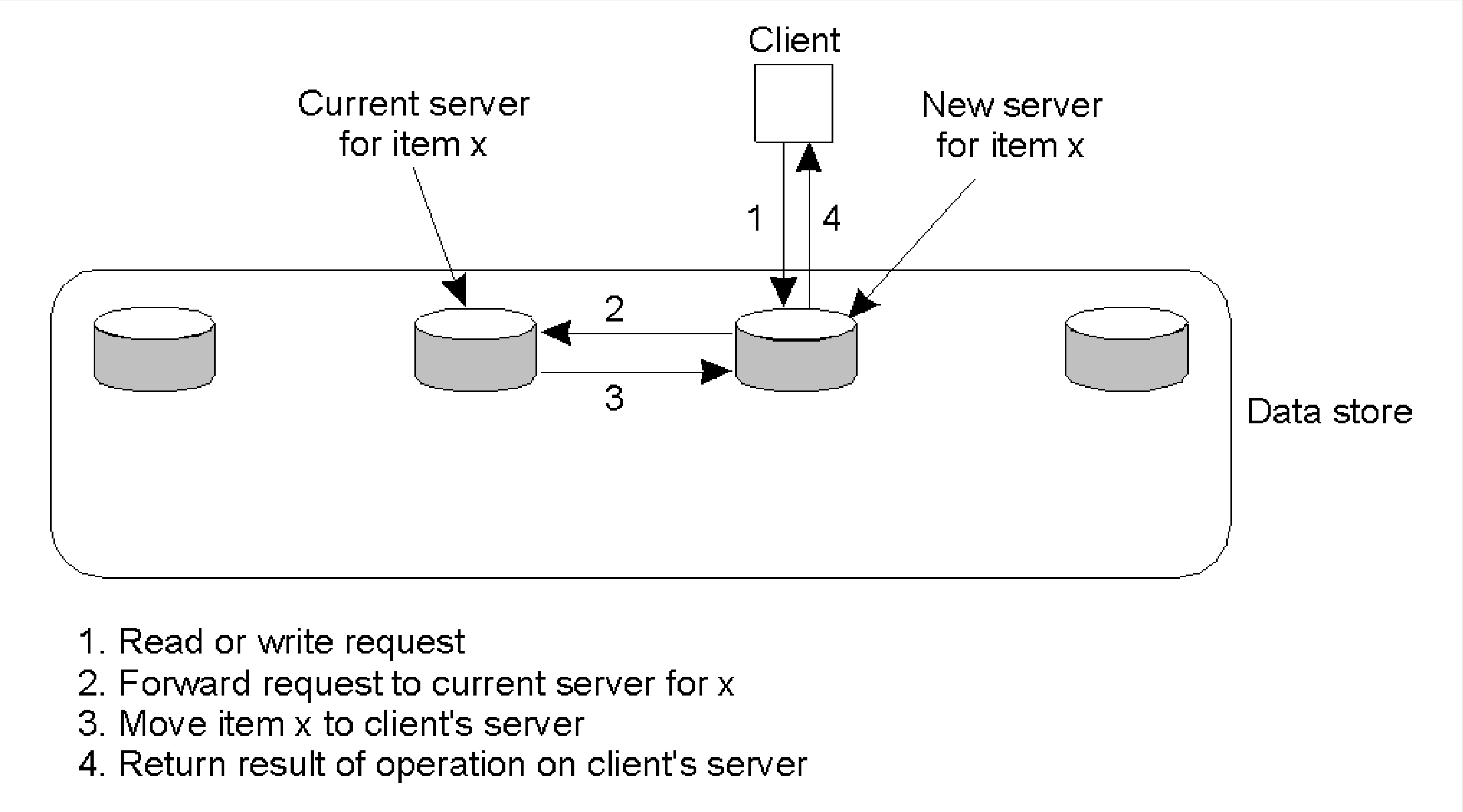
R1. Read request
R2. Response to read

Passive Replication: Local-Write

- Separate read with write, multiple copies for read, single for write
- Based on object migration
  - Multiple successive writes can be carried out locally
- How to locate the data item?
  - broadcast, home-based, forwarding pointers, or hierarchical location service



Local-Write Protocols (non-blocking)



- Limitation: ?

Local-Write Protocols (blocking)