Consistency and Replication

Today

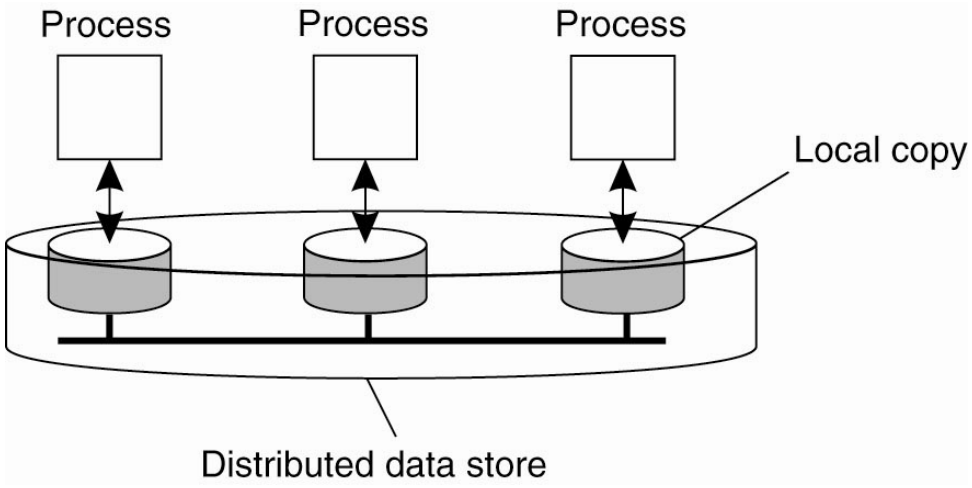
- Reasons for replication
- Consistency models
- Replica management
- Consistency protocols

Performance and scalability

- Replications for reliability and performance
- To keep replicas consistent, ensure that all conflicting operations are done in the same order everywhere
- Conflicting operations: from the world of transactions
  - Read–write conflict: a read & write operation act concurrently
  - Write–write conflict: two concurrent write operations
- 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

Data-centric consistency models

- A data store is a distributed collection of storages accessible to clients

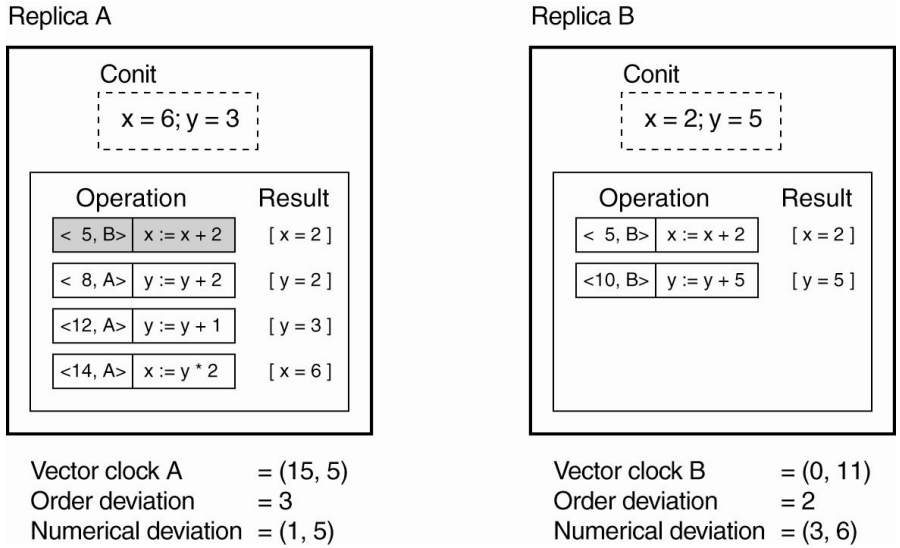


- Consistency model: a contract between a (distributed) data store and processes, where data store specifies precisely what the results of read/write operations are in the presence of concurrency

Continuous consistency

- We can actually talk a about a degree of consistency:
  - Replicas may differ in their numerical value (relative or absolute difference – your account balance)
  - Replicas may differ in their relative staleness
  - There may differences with respect to (number and order) of performed update operations
- Conit: consistency unit ⇒ specifies the data unit over which consistency is to be measured
  - A single stock in the stock exchange example
  - The account balance
- Too fine-grained conits, more to manage; too coarse-grain conits, false sharing …

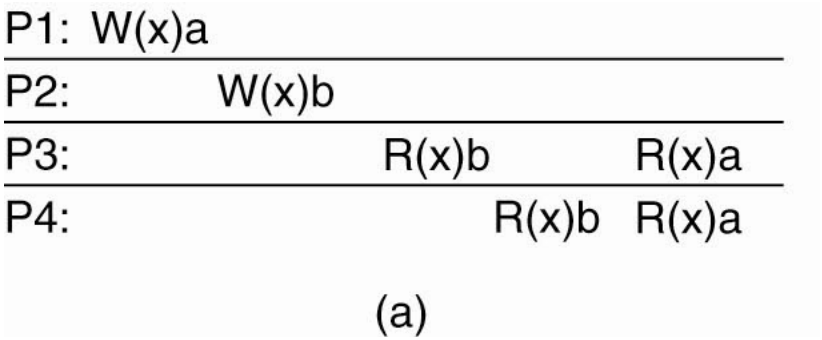
Example - conit

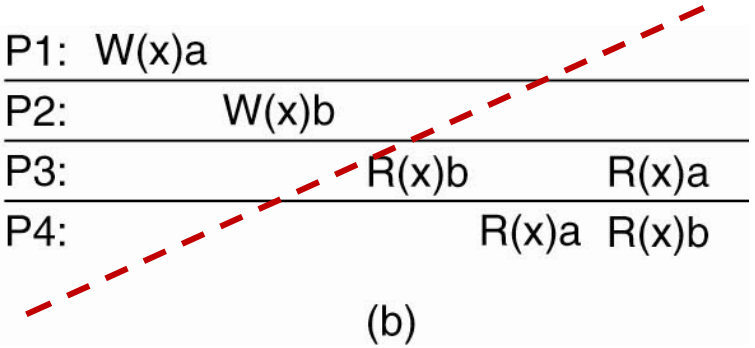


- Conit: contains the variables *x* and *y*
  - Two replicas, each maintains a vector clock
  - B sends A operation [5,B: x := x + 2]; A has made this operation permanent (cannot be rolled back)
  - A has three pending operations ⇒ order deviation = 3
  - A has missed one operation from B (y := y + 5), yielding a max diff of 5 units ⇒ (1,5)

Sequential consistency

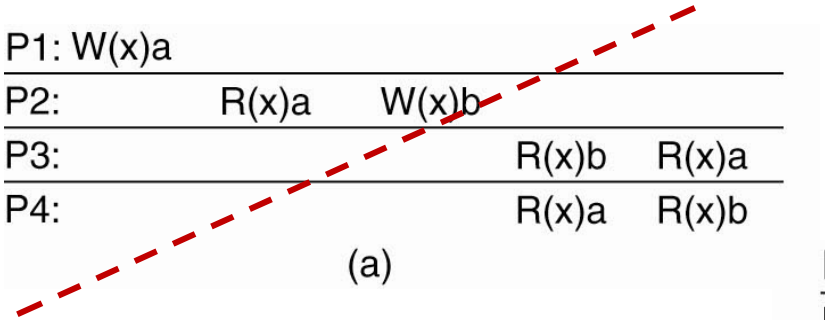
- *The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each process appear in this sequence in the order specified by its program*
- Any valid interleaving of operations is OK, but all processes see the same interleaving



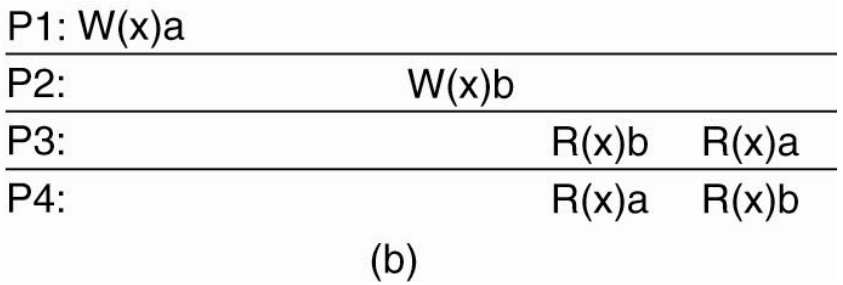


Causal consistency

- *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in different order by different processes*



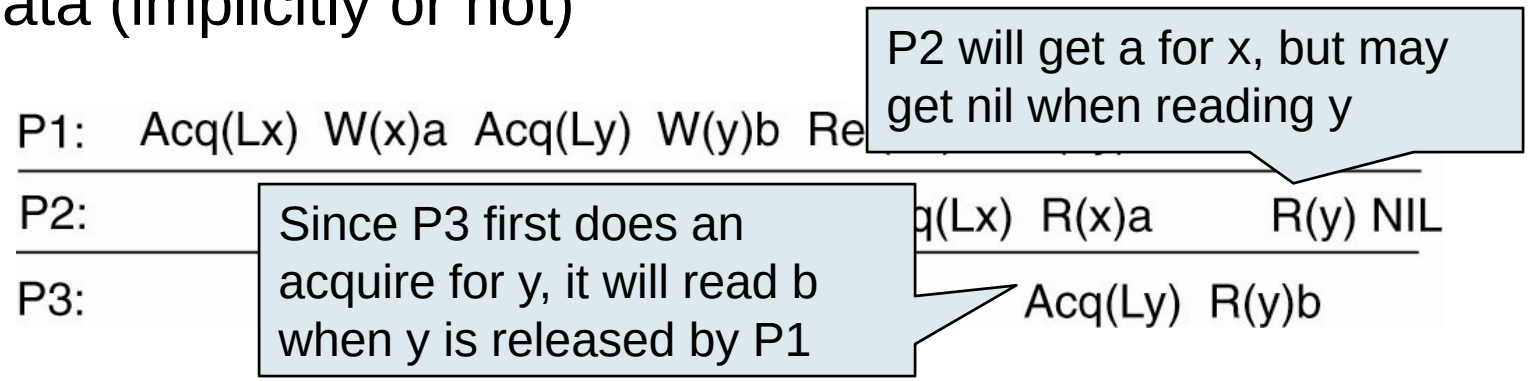
(a)



(b)

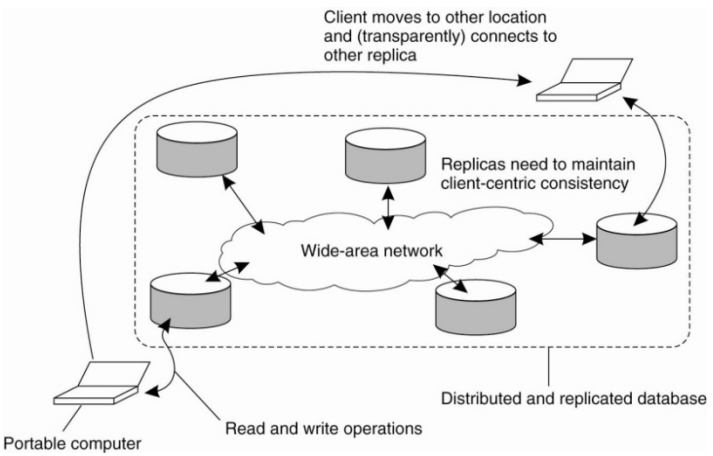
Grouping operations – entry consistency

- Basic idea: don't care that reads and writes of a series of ops are immediately known to other processes; just want the effect of the series itself to be known
  - Accesses to synchronization variables are sequentially consistent
  - No access to a synchronization variable is allowed until all previous writes have completed everywhere
  - No data access is allowed until all previous accesses to synchronization variables have been performed
- Weak consistency implies that we need to lock/unlock data (implicitly or not)



Client-centric consistency models

- For some distributed data stores with rare simultaneous updates, eventual consistency is enough
  - DNS, WWW, distributed email
- Problems may show up if the same user access data from different replicas
- Consider a distributed database to which you have access through a notebook (that acts as a front end)
  - At location A you access the database doing reads/updates.
  - At B you continue working, but …

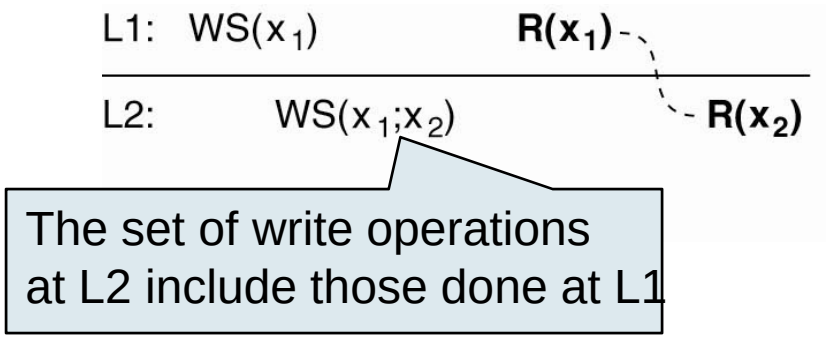


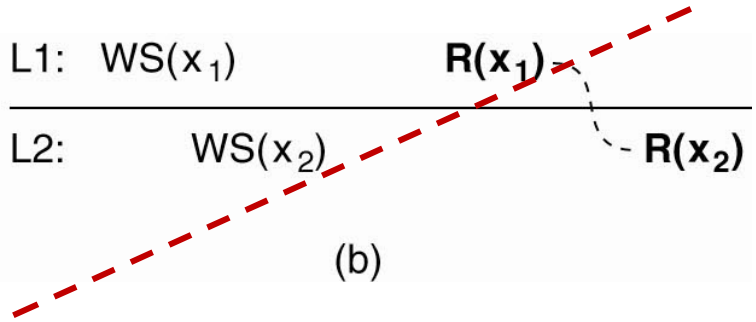
Client-centric consistency models

  - … but unless you access the same server as when at A, you may detect inconsistencies:
    - your updates at A may not have yet been propagated to B
    - you may be reading newer entries than the ones available at A
    - your updates at B may eventually conflict with those at A
- All you want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will seem consistent to you
- Client-centric consistency – consistency for a single client, *nothing about concurrent access by different clients*

Monotonic reads

- *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 or a more recent value*
- If you've seen a value of x at time t, you'll never se anything older at a later time

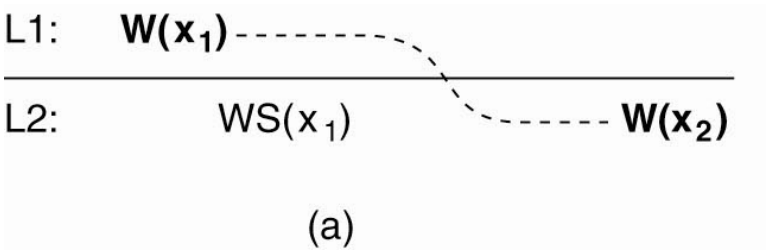


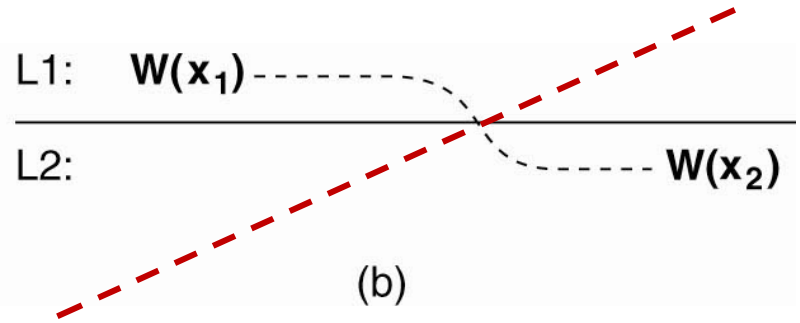


- Examples:
  - Reading your personal calendar updates from different servers.
  - Reading (not modifying) incoming mail in the move

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*
- i.e. a write on x is performaced only if that copy has been brought up to date

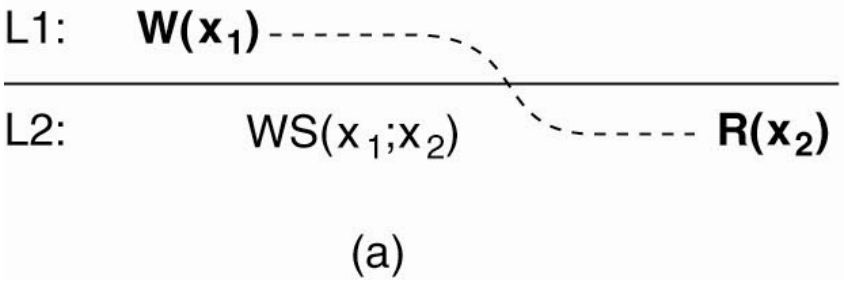


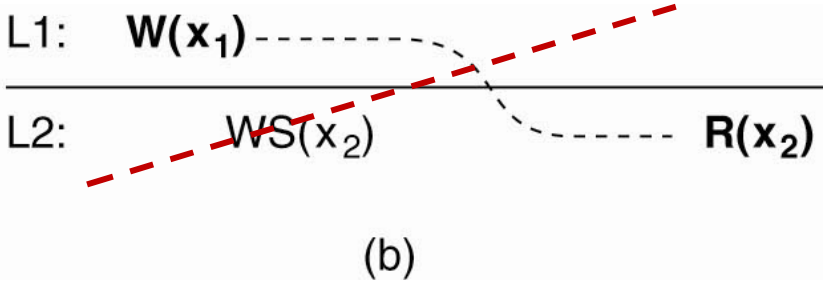


- Example
  - Maintaining versions of replicated files in the correct order everywhere (CVS-like)

Read your writes

- *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*
- i.e. a write is always completed before a successive read by the same process, no matter where the read is

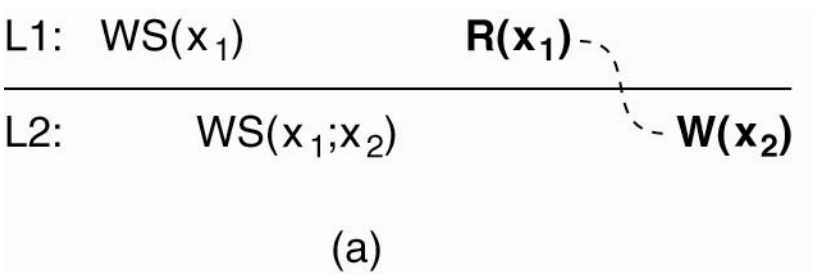


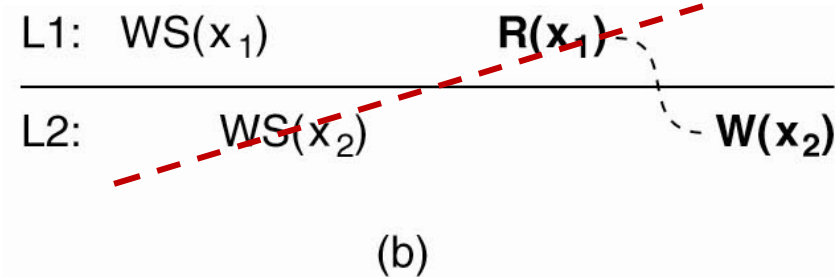


- Example:
  - Changing your password in dylan and try to login into zappa too soon after

Writes follows reads

- *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*





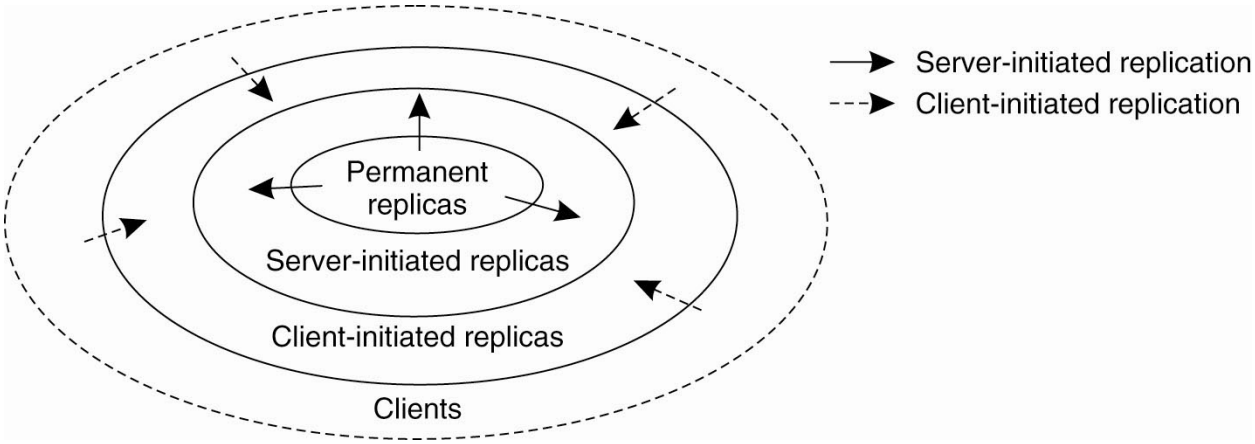
- Example:
  - See reactions to posted articles only if you have seen the original posting (a read "pulls in" the corresponding write operation)

Replica placement

- What are the best K out of N possible locations for a replica
  - Select one server at a time so to minimize the average distance between clients and replicas. Computationally expensive.
  - Look at the Internet topology, in terms of Autonomous Systems (AS). Select the K-th largest AS and place a server at the best-connected host. Computationally expensive.
  - Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. To compute the region size use the average distance between two nodes and K. Computationally cheaper.

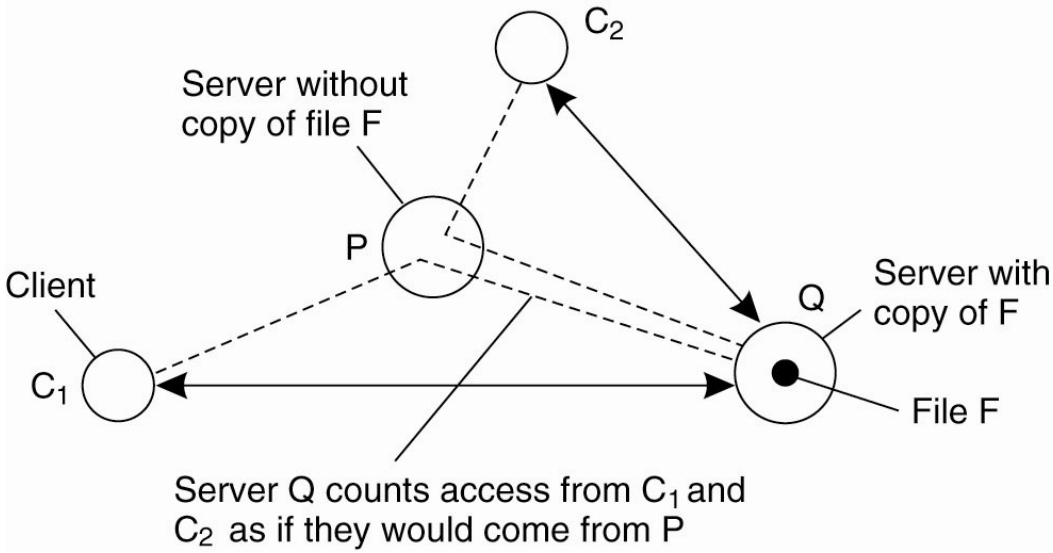
Content replication

- Distinguish different processes: A process is capable of hosting a replica of an object or data:
  - Permanent replicas: Process/machine always having a replica, think of it as the initial set
  - Server-initiated replica: Process that can dynamically host a replica on request of another server in the data store (remember you already have the replica servers placed)
  - Client-initiated replica: Process that can dynamically host a replica on request of a client (client cache)



Server-initiated replicas

- Keep track of access counts per file, aggregated by considering server closest to requesting clients
- Number of accesses < threshold D ⇒ drop file
- Number of accesses > threshold R ⇒ replicate file
- Number of access between D and R (and more requests at P than at Q) ⇒ migrate file to P



Content distribution

- Consider only a client-server combination
  - Propagate only notification/invalidation of update
  - Transfer data from one copy to another
  - Propagate the update operation (aka active replication)
- No single approach is the best, but depends on available bandwidth and read-to-write ratio at replicas
- Pushing/pulling updates:
  - Push - server-initiated, update is propagated regardless whether target asked for it
  - Pulling - client-initiated, client requests to be updated

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

Content distribution

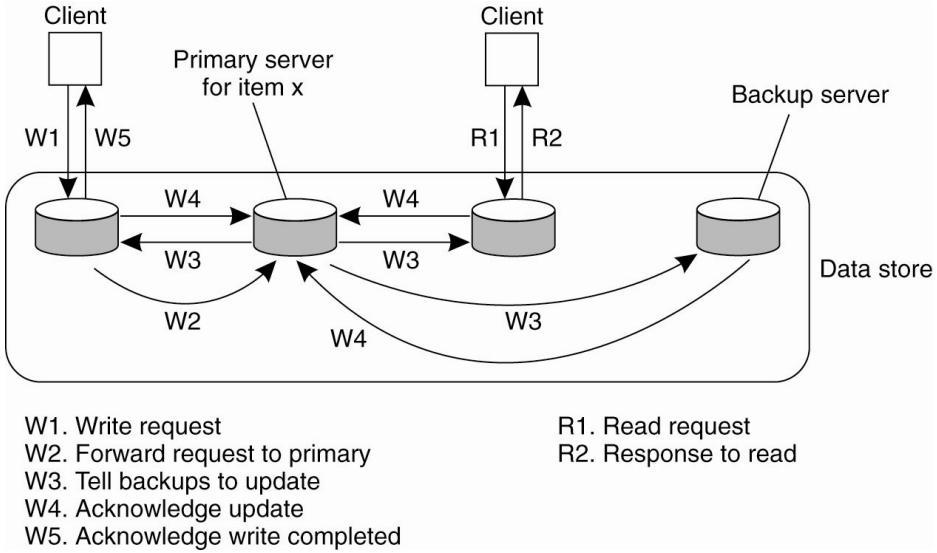
- *Leases* to dynamically switch bet/ pulling and pushing
  - A contract in which the server promises to push updates to the client until the lease expires
- Make lease expiration time dependent on system's behavior (adaptive leases):
  - Age-based: An object that hasn't changed for a long time, will not change in the near future, provide a long-lasting lease
  - Renewal-frequency based: The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
  - State-based: Higher load at servers, shorter expiration times
- Unicasting or multicasting
  - With push-based, multicasting may be a good idea
  - With pull-based, unicast is your only reasonable model

Consistency protocols – continuous

- Bounding numerical deviations
  - Replicas help to keep other replicas within bounds by pushing updates, looking at what they think everybody has seen
- Staleness can be done analogously, by essentially keeping track of what has been seen last from S_i
  - Replica starts pulling writes soon as time diff. is exceeding some limit
- Bounding ordering deviations
  - Caused by the fact that replica servers tentatively apply updates submitted to them; each server has a local queue of tentative writes, keep the length bounded
  - When reaching limit, stop accepting writes and try to commit the tentative writes by agreeing on some globally consistent order

Primary-based protocols

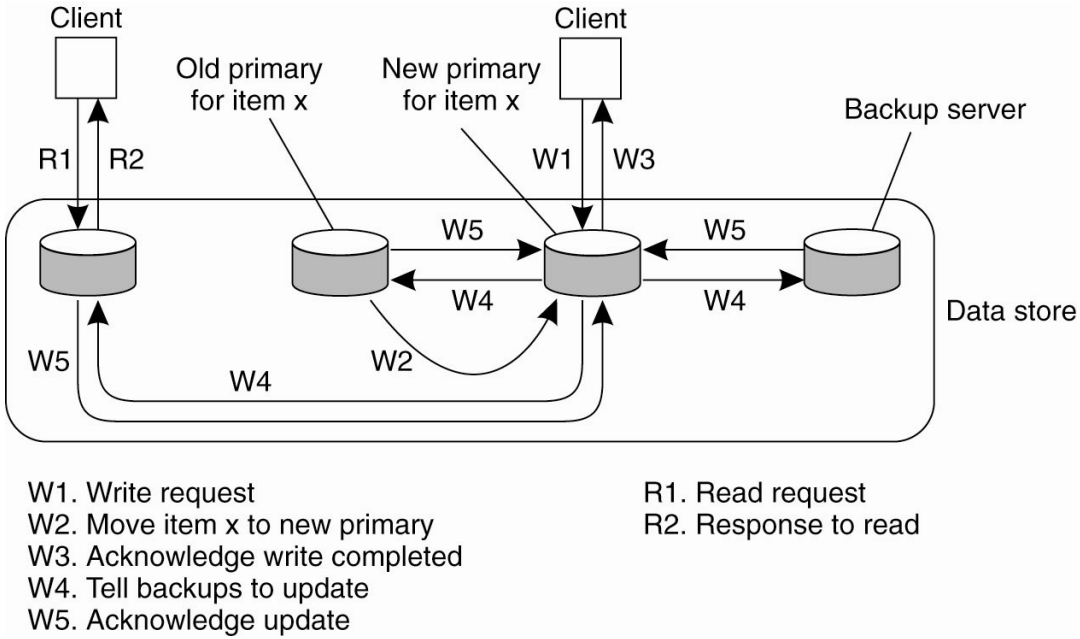
- Primary-backup protocol – all writes are blocking, forwarded to primary server; reads are local



- The process that does the write may block for a long while; but this is fault tolerant and easy to implement
- A non-blocking approach trades fault tolerance for performance

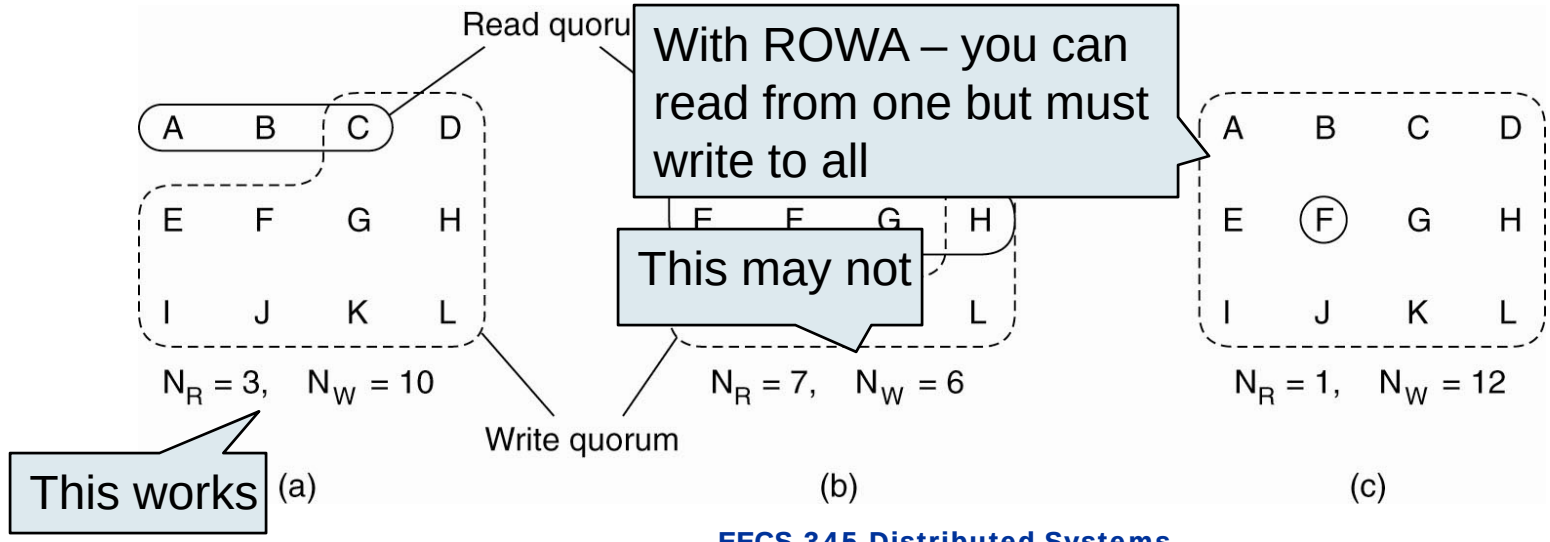
Primary-based protocols

- Primary-backup protocol with local writes – migrate primary copy between processes that want to write
- Multiple successive writes can be done locally
- Can be applied to mobile computing, for operation while being disconnected



Replicated-write protocols

- Write operations can be done at multiple replicas
- Ensure that each operation is carried out in such a way that a majority vote (quorum) is established; distinguish read quorum and write quorum:
- File is replicated on N servers
  - Nr – read quorum; Nw – write quorum
  - Nr + Nw < N (to prevent read/write conflicts); Nw > N/2 (to prevent write-write conflicts)



Summary

- Again, we use replication for performance and reliability
- Replication, however, introduces a few issues
  - The problem of consistency, which we may pay in terms of performance
  - The “details” of placement and management