Distributed Systems (5DV147)

Replication and consistency

Fall 2013

Replication

What is replication?

- Make different copies of data ensuring that all copies are identical
  - Immutable data – trivial
  - Often updated data – tricky, can be expensive to maintain copies identical
- Replication requirements
  - Replication transparency (illusion of single copy)
  - Consistency

Why replication?

- Reliability
  - Fault tolerance (redundancy, switching to other replica)
  - Protection against corrupted data (majority)
  - Availability
- Performance
  - Scalability (divide the work)
  - Load balancing
  - Reducing access latency (data closer to process)
    - Caching

Problems that you may find

- Problems if multiple clients access replicas
  - Concurrent access, rather than exclusive
  - Operations are *interleaved*
    - How do we ensure correctness?
- Replica placement
  - Servers
  - Content
- Overhead required to keep replicas up to date
  - Global synchronization (Atomic operations)
    - Relaxed atomicity constraint, but copies will not always be the same
      - Depends on access and update patterns of data

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

Correctness of interleaving

- “Basic” correctness property
  - An interleaved sequence of operations must meet the specification of a single correct copy of the object(s)
- Sequential consistency property
  - Order of operations is consistent with the program order in which each individual process executed them
- Linearizability property
  - Order of operations is consistent with the real times at which the operations occurred during execution

Example of interleaved operations for 2 clients:

C1: A, B, C

C2: d, e, f

Real Order during execution: A, B, d, C, e, f

An interleaving with sequential consistency:

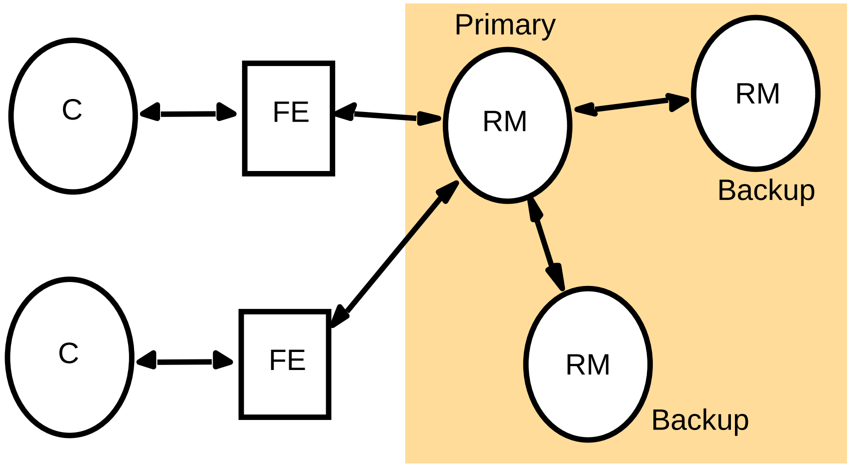
A, B, d, e, f, C

Interleaving with linearizability:

A, B, d, C, e, f

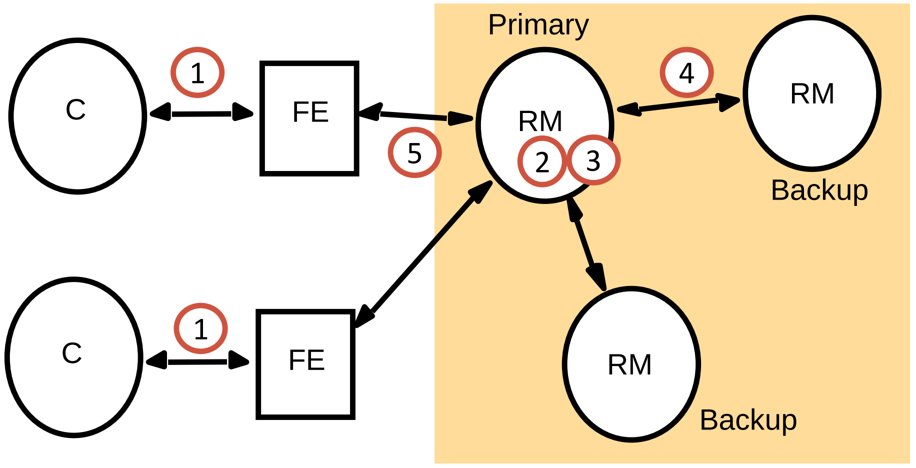
Passive replication

- One *primary* replica manager, many backup replicas
  - If primary fails, backups can take its place (election!)
- Implements linearizability if:
  - A failing primary is replaced by a unique backup
  - Backups agree on which operations were performed before primary crashed
    - View-synchronous group communication!



Steps of passive replication

1. Request
   – Front end issues request with unique ID
2. Coordination
   – Primary checks if request has been carried out, if so, returns cached response
3. Execution
   – Perform operation, cache results
4. Agreement
   – Primary sends updated state to backups
5. Response
   – Primary sends result to front end, which forwards to the client



What happens if the primary RM crashes?

- Before agreement
- After agreement

Active replication

- More distributed
- All replica managers carry out all operations
- Requests to RM are totally ordered
- Front ends issue one request at a time (FIFO)
- Implements sequential consistency

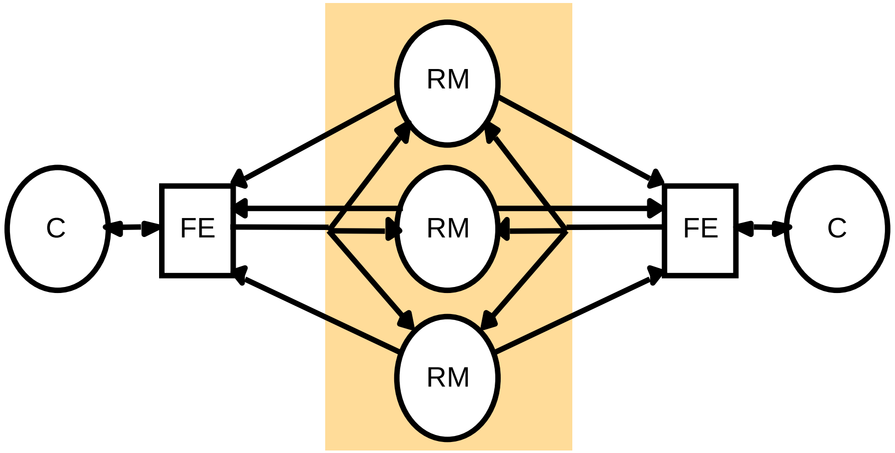
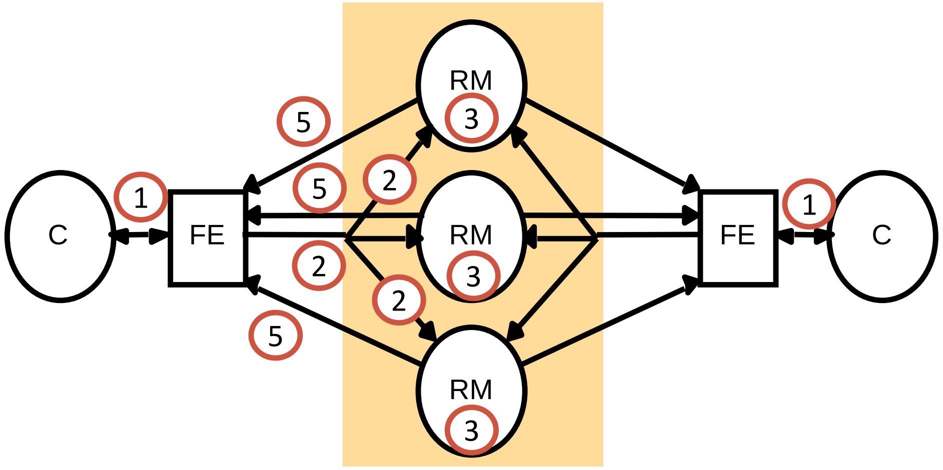


Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – based on **Figure 18.4**

Steps of active replication

1. Request
   - Front end adds unique identifier to request, multicasts to RMs
2. Coordination
   - Totally ordered request delivery to RMs
3. Execution
   - Each RM executes request
4. Agreement
   - Not needed
5. Response
   - All RMs respond to front end, front end interprets response and forwards response to client



Comparing active and passive replication

- Both handle crash failures (but differently)
- Only active can handle arbitrary failures
- Optimizations?
  - Send “reads” to backups in passive

    Lose linearizability property!
  - Send “reads” to specific RM in active

    Lose fault tolerance
  - Exploit commutativity of requests to avoid ordering requests in active

Consistency

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

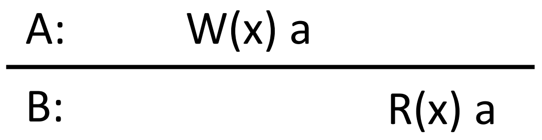
- Is a contract between processes and a data store
  - if processes agree to obey certain rules, the store promises to work correctly (Tanenbaum and van Steen, 2002)
  - What to expect when reading and updating shared date (while others do the same)
  - Data-centric models (system-wide)
  - Client-centric models (single client)

Data-centric consistency models

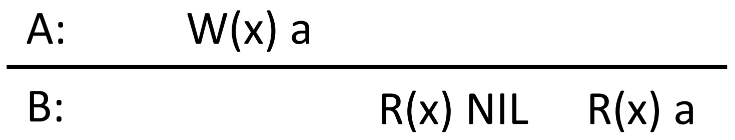
Strict consistency

- 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

Strict consistency example:



Strictly consistent



Not strictly consistent

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

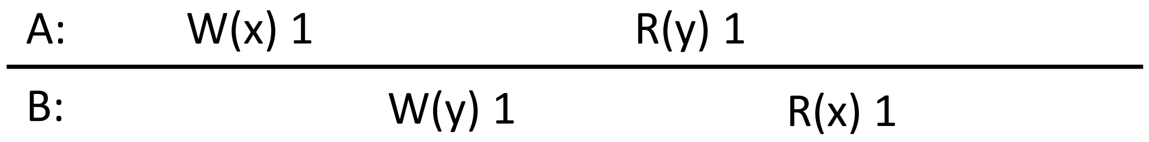
(regardless on the number of replicas of x)

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
  - 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 $t_1 < t_2$

Linearizability example:

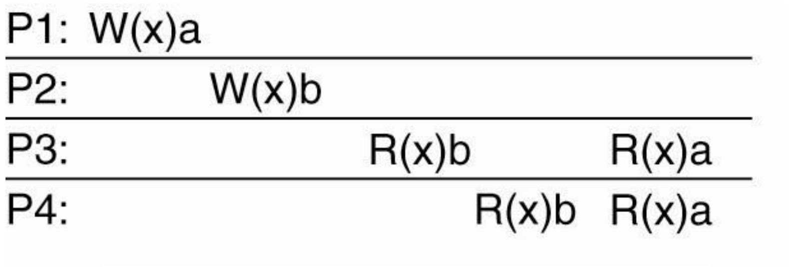


The linearizable trace is A:W(x,1), B:W(y,1), A:R(y,1), B:R(x,1)

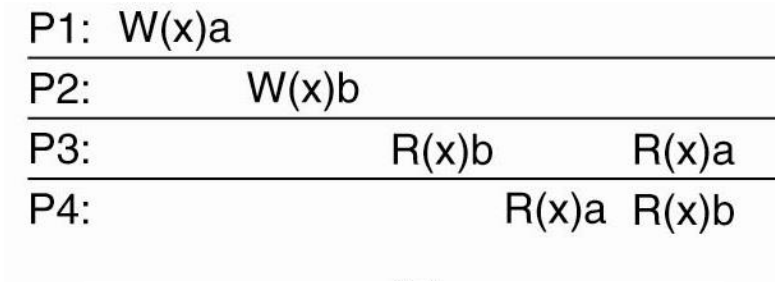
Sequential consistency

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

Sequential consistency example



Sequentially consistent



Not sequentially consistent

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

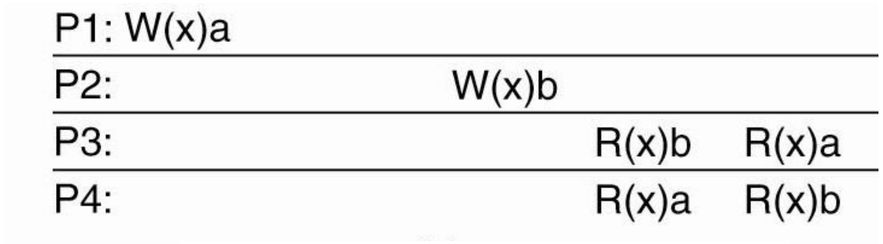
W_2 (x)b, R_3(x)b, ???

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

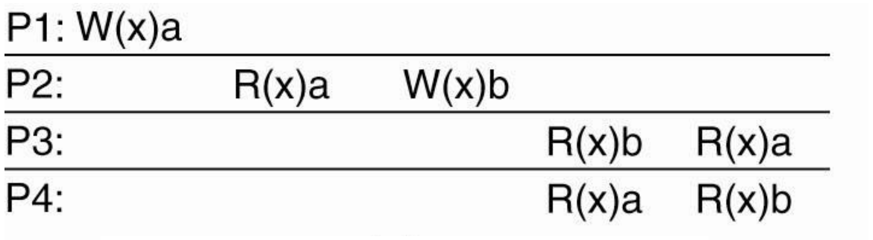
Causal consistency

- All writes that are 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



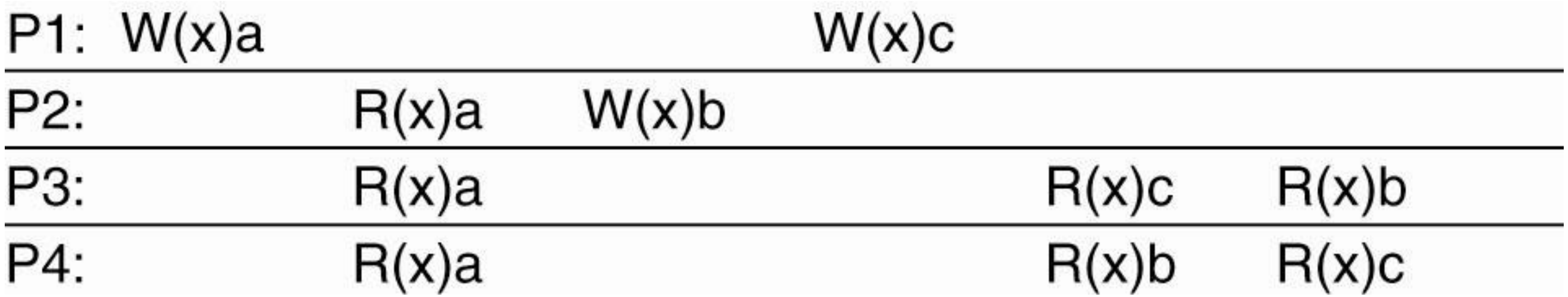
Causally consistent



Not causally consistent

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

Example



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

Causally consistent

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

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**

Update propagation and consistency protocols

Update propagation

- Price of replication → Keep replicas consistent
- Replica updates
  - Atomic updates (all replicas need to be identical), maintain all replicas equal
  - Maintaining replicas consistent may also generate scalability problems

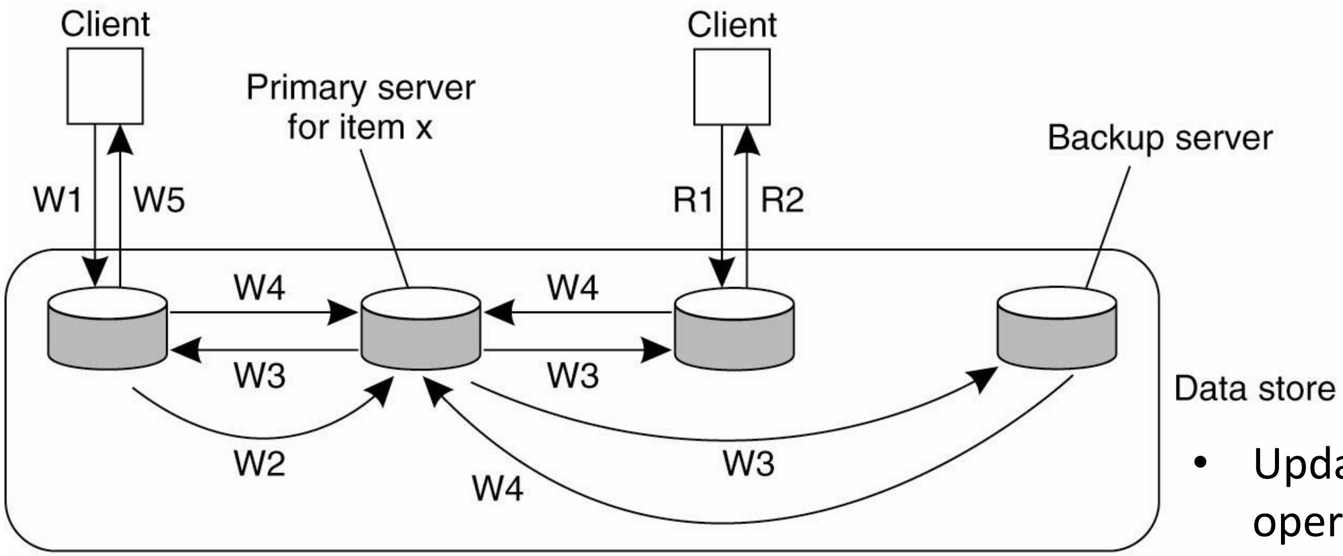
Update propagation

- Update notifications
  - Good for low read-to-write ratios
- Transfer data from one copy to another
  - Good for high read-to-write-ratios
- Propagate the update operation to other copies
  - Active replication
- Push
  - Propagation initiated by server
  - Good for high read-to-write ratios
- Pull
  - Client requests server to send updates
  - Good for low read-to-write ratios

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 → quorum-based protocols

Primary-based protocol: remote-write



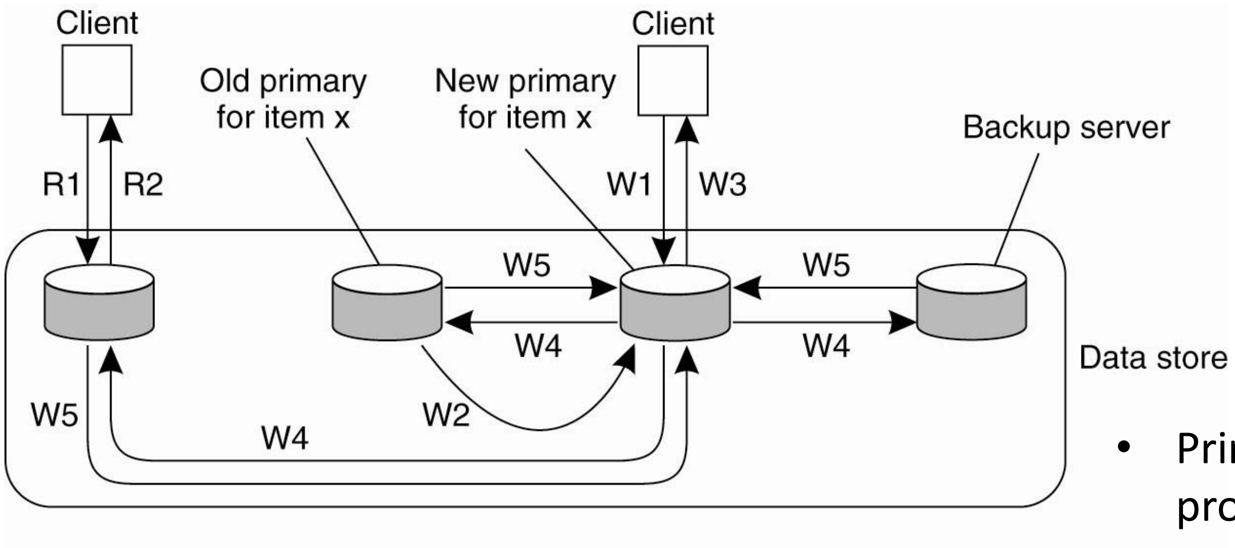
W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

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

problem → Fault tolerance

Primary-based protocol: local-write



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

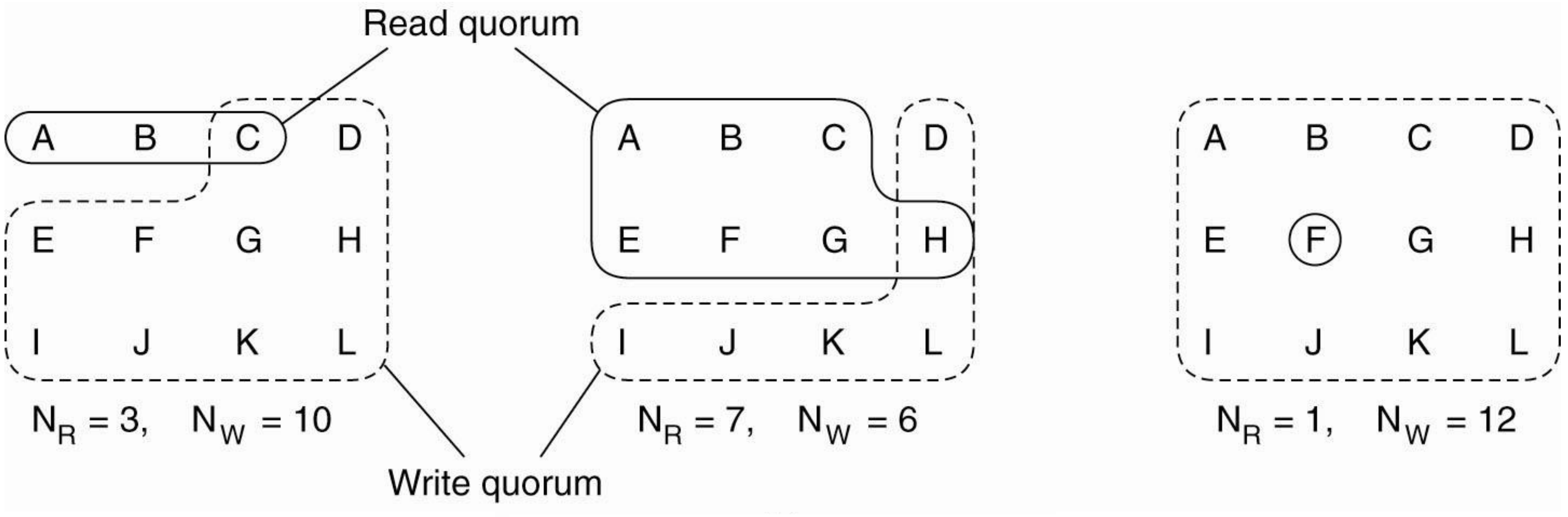
- Primary migrates between processes that wish to perform an operation
- Optimization → carry out multiple successive writes locally

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**

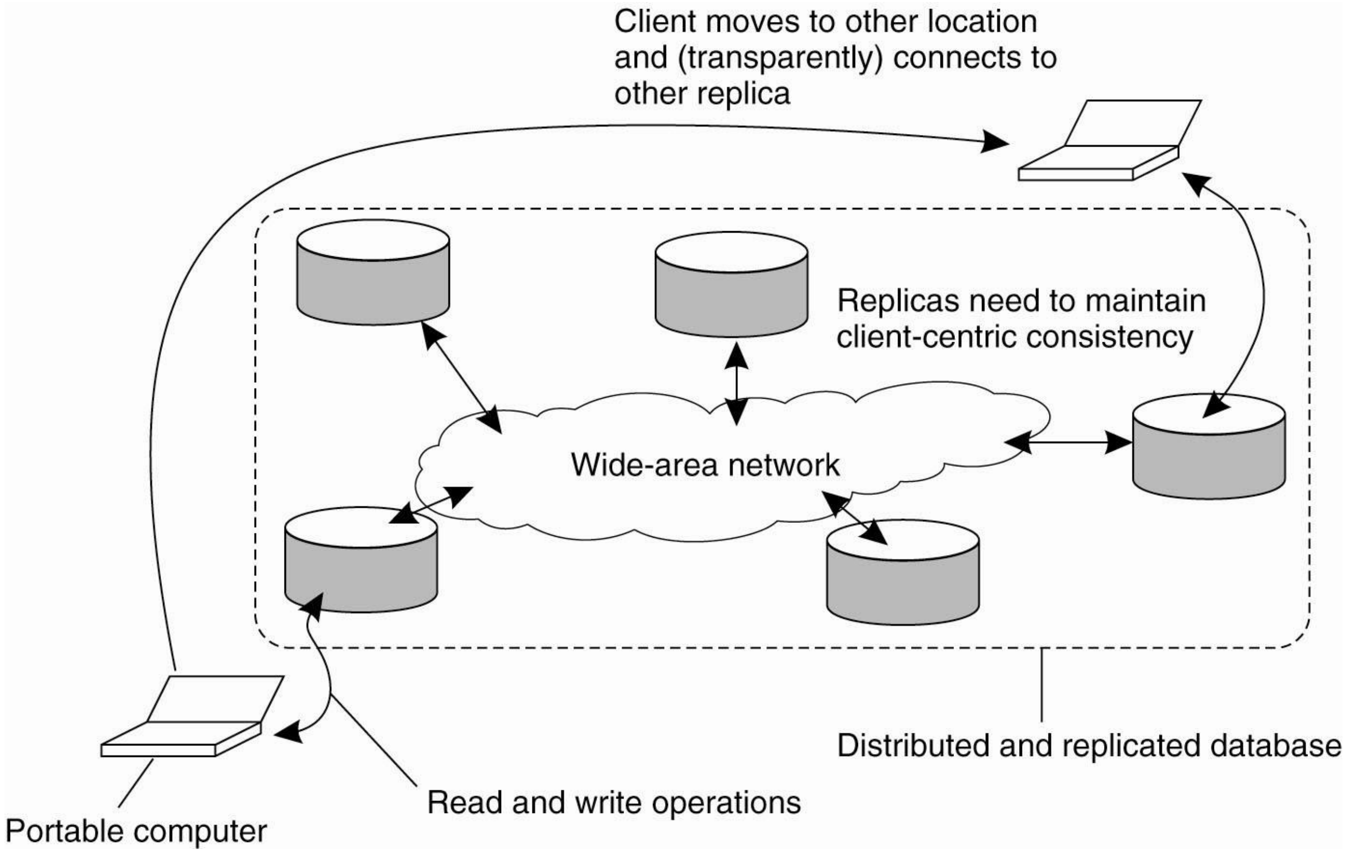
Client-centric consistency models

Eventual consistency

- Maintains consistency for *individual clients,* not considering that data may be shared by several clients
- If updates are infrequent, eventually all replicas will obtain the update and become identical
    - Good if clients always access the same replica
- Delay resolving conflicts
- Assume that clients connect to different replicas and that differences in those replicas should be transparent

Several variations …

Client centric consistency


Client moves to other location and (transparently) connects to other replica
Replicas need to maintain client-centric consistency
Wide-area network
Distributed and replicated database
Read and write operations
Portable computer

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

Monotonic-write consistency

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

Read-your-writes consistency

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

Write-follow-reads consistency

- When writing 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

… more about groups

Group views

- Contain the set of group members at a given point in time
  - Failed identified processes are not in the view
- Events occur *in views*
- Views are delivered when membership changes
- View-synchronous group communication
  - Based on view delivery, we can know which messages must have been delivered (within a view)

View delivery requirements

- Order
  - View changes always occur in the same order at all processes
- Integrity
  - If a process delivers a view then that process is part of that view
- Non-triviality
  - Processes that have joined a group and communicate indefinitely are members of the same group
  - Membership service should eventually reflect network partitions

View-synchronous group communication

- Correct processes deliver the same set of messages in any given view
- Messages are delivered at most once
- Correct processes always deliver messages they send:
  - If delivering to *q* fails, the next view excludes *q*

Summary (1)

- What is replication and why is it necessary
- Correctness of interleaving
  - Basic, sequential consistency, and linearizability
- General replication phases
  - Request, coordination, execution, agreement, response
- Types of ordering adapted to replication
  - FIFO, Causal, Total

Summary (2)

- Passive replication (implements linearizability)
  - A single primary replica manager and one or more backup replica managers
- Active replication (implements sequential consistency)
  - Independent replica managers executing all operations
- Updates propagation
  - Update notifications, data, or operations, and push vs. pull
- Consistency protocols (implementation of consistency model)
  - Primary-based protocols (passive)
    - Remote-write
    - Local-write
  - Quorum-based protocols (active)

Summary (3)

- Consistency models
  - Data-centric models (strict, linearizability, sequential, causal), differ …
    - In how restrictive they are
    - How complex their implementations are
    - Ease of programming
    - Performance
  - Client-centric models
    - Eventual consistency
      - Monotonic reads, monotonic writes, read your writes, writes follow reads

Summary (4)

- More about groups
  - Static vs. dynamic groups and primary partition vs. partitionable groups
- Group views
  - Current list of members
  - Events occur in views
  - Views are delivered when membership changes
  - Requirements for delivering views
- View synchronous group communication

Next Lecture

Cassandra