Distributed Information Processing

11th Lecture

Eom, Hyeonsang (엄현상)
Department of Computer Science & Engineering
Seoul National University

©Copyrights 2016 Eom, Hyeonsang All Rights Reserved

Outline

- Other Topics in Distributed Systems
  - Consistency
  - Replication
  - Fault Tolerance
- Q&A

Summary of Data-Centric Consistency Models

Models Not Using Synchronization Operations

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
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

Summary of Data-Centric Consistency Models

Models Using Synchronization Operations

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

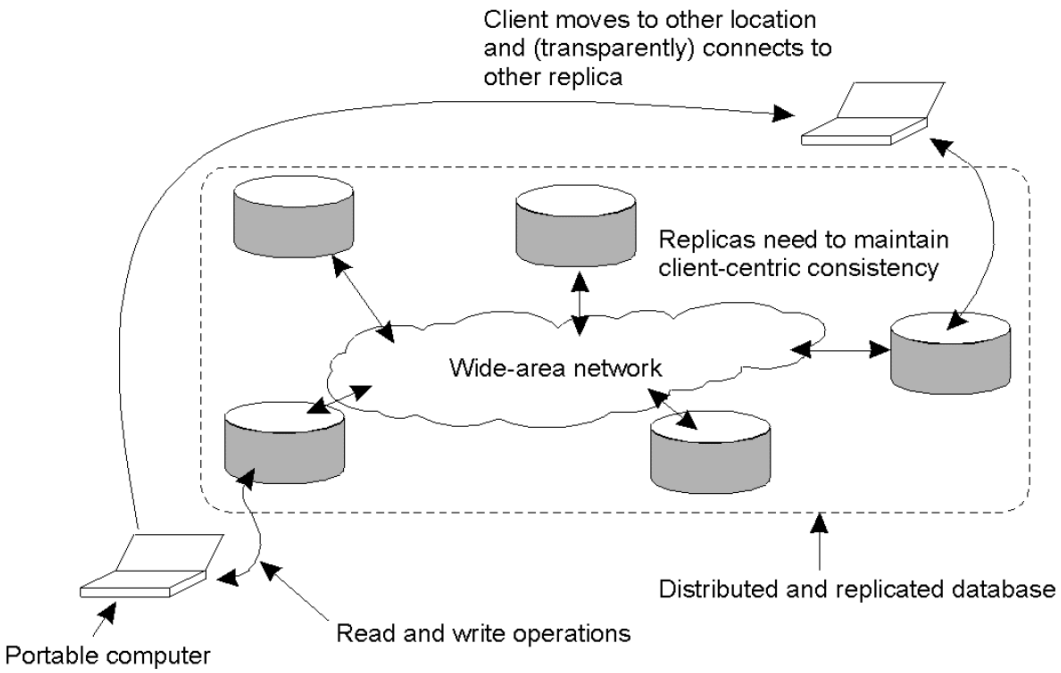
Client-Centric Consistency Models

- Eventual Consistency
- Monotonic Reads
- Monotonic Writes
- Read Your Writes
- Writes Follow Reads
- Implementation

Eventual Consistency

- All Replicas Will Gradually Become Consistent

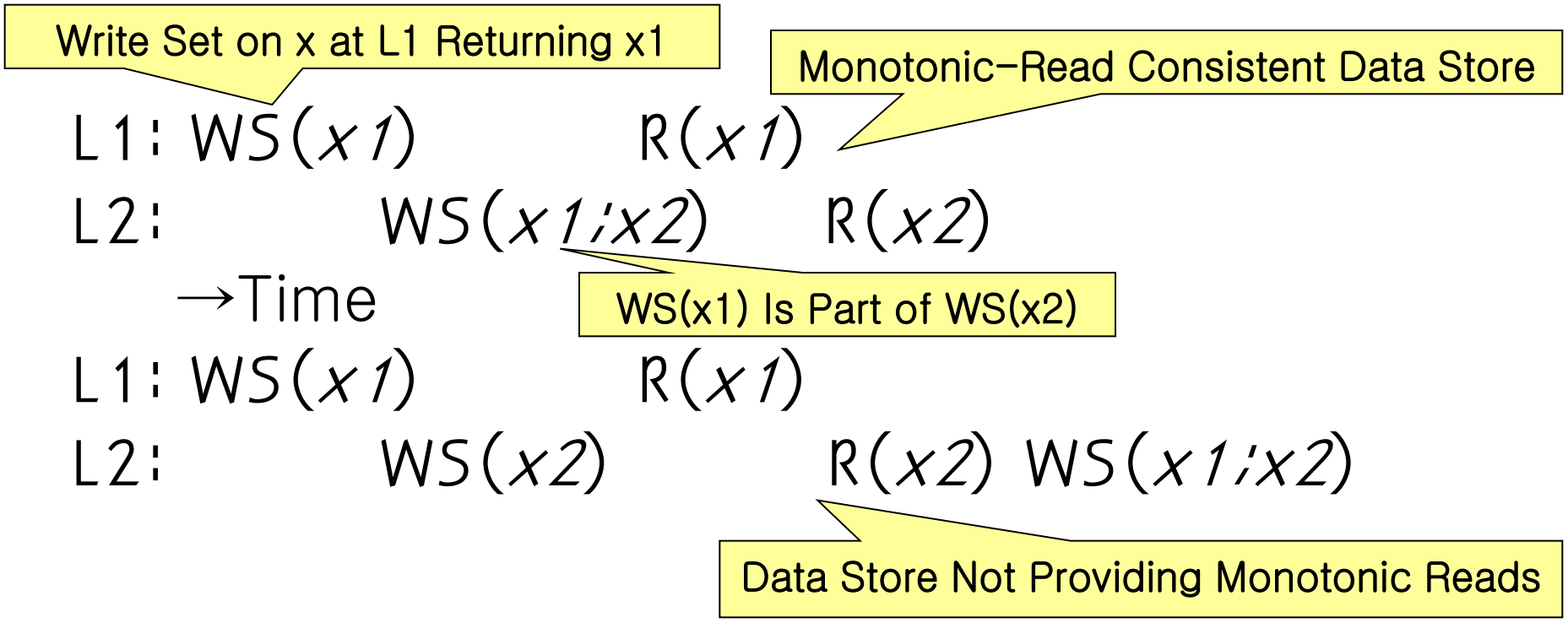
Distributed Data Stores Characterized by the Lack of Simultaneous Updates: e.g., DNS and WWW



The principle of a mobile user accessing different replicas of a distributed database.

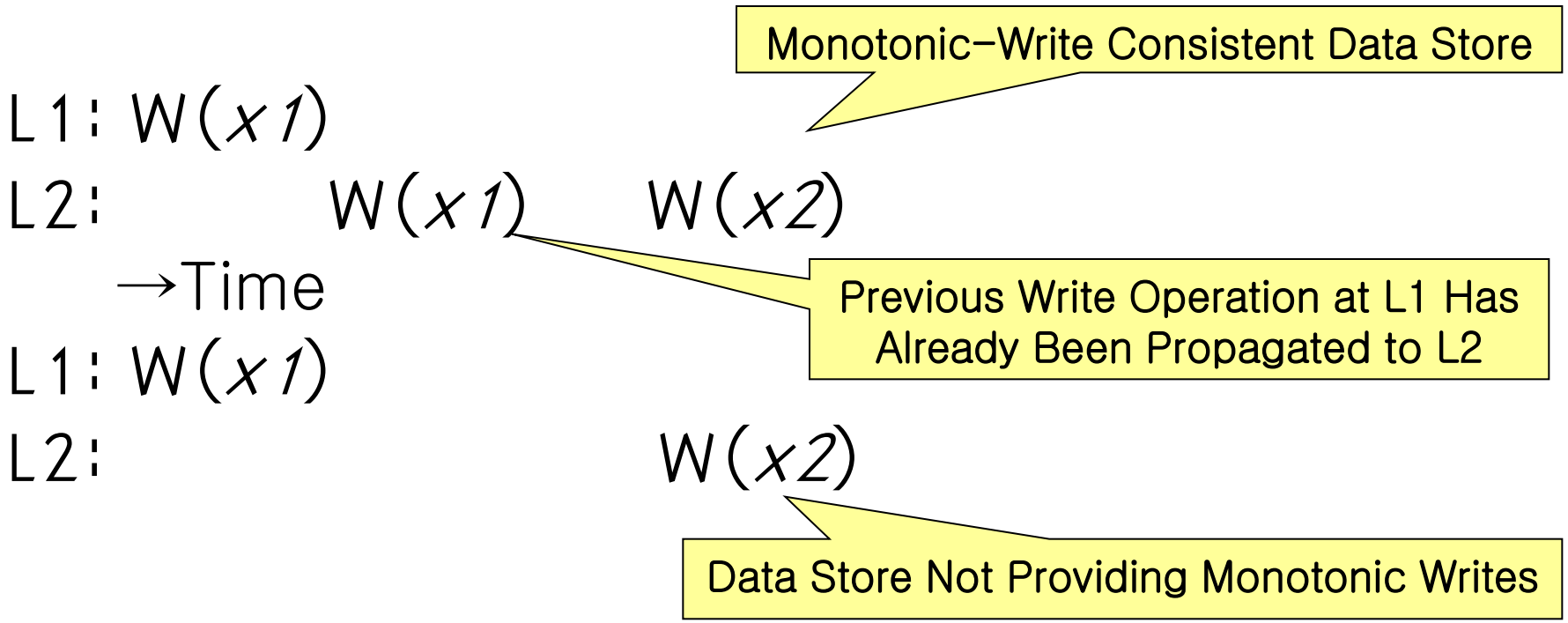
Monotonic-Read Consistency

- 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 Recent Value



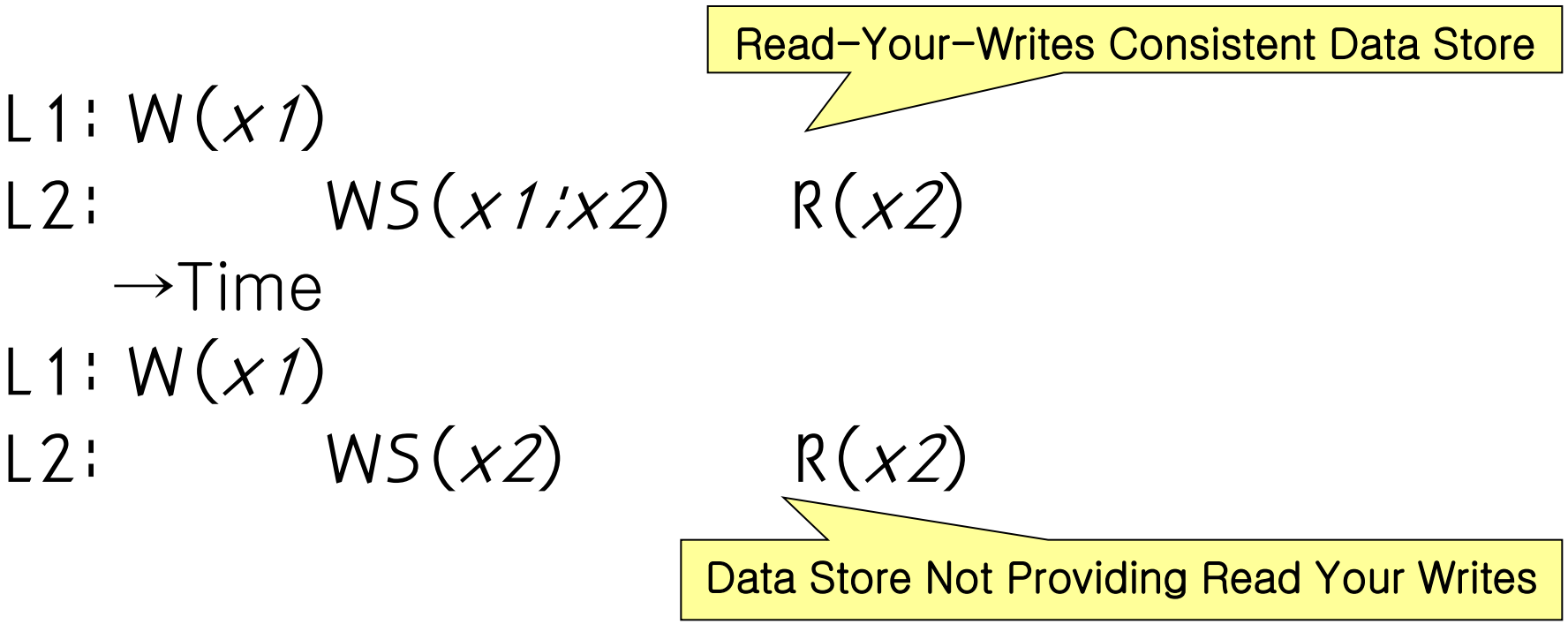
Monotonic-Write Consistency

- A Write Operation by a Process on a Data Item x, Is Completed before Any Successive Write Operation on x by the Same Process



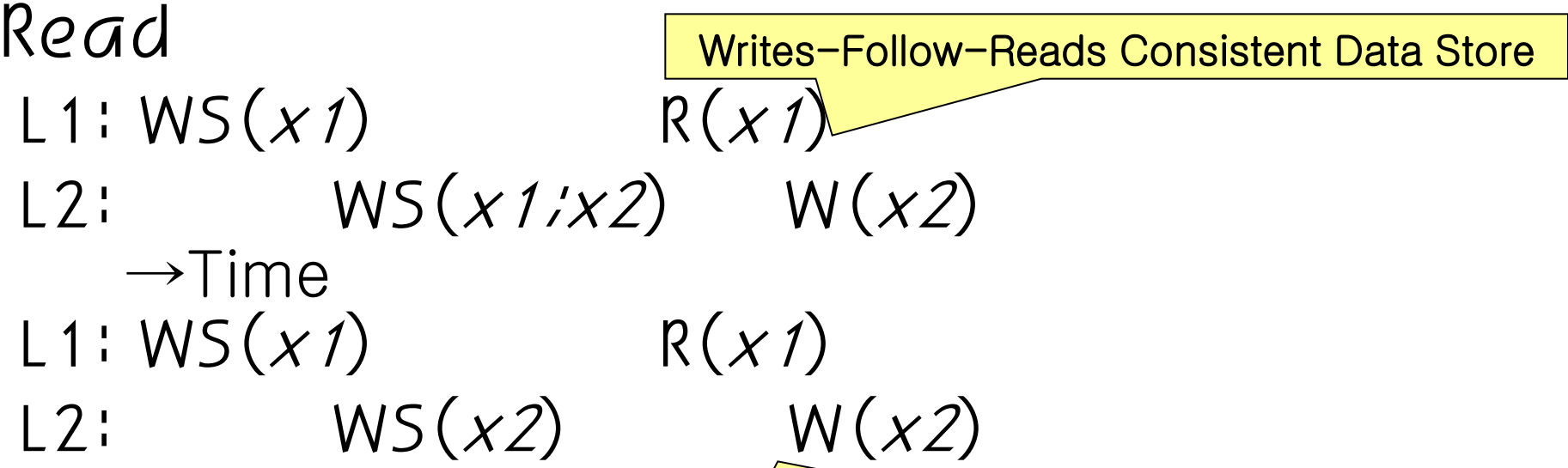
Read-Your-Writes Consistency

- 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



Writes-Follow-Reads Consistency

- A Write Operation by a Process on 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



Implementation

- Assumptions
  - Each write operation is assigned a globally unique identifier by the server that accepts the operation for the first time (i.e., the operation is initiated at that server)
  - Read set for a client consists of the write identifiers relevant for the read operations performed by a client
  - Write set consists of the writes performed by the client

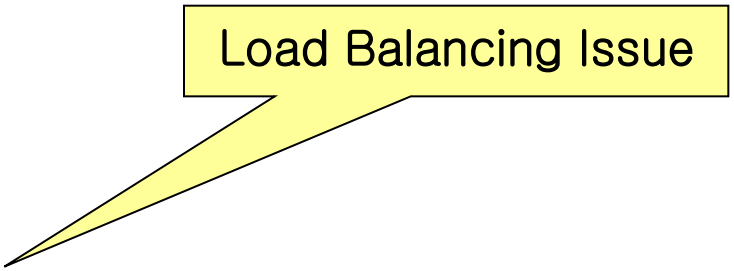
Implementation (Cont'd)

- Monotonic-Read Consistency
  - When a client performs a read at a server, that server is handed the client's read set to check whether all the identified writes have been taken place locally
  - If not, the read is postponed
    - Alternatively, the read is forwarded to a server where the writes have already taken place
- Monotonic-Write Consistency
  - When a client performs a write at a server, that server is handed the client's write set to check whether all the identified writes have been performed first and in correct order

Implementation (Cont'd)

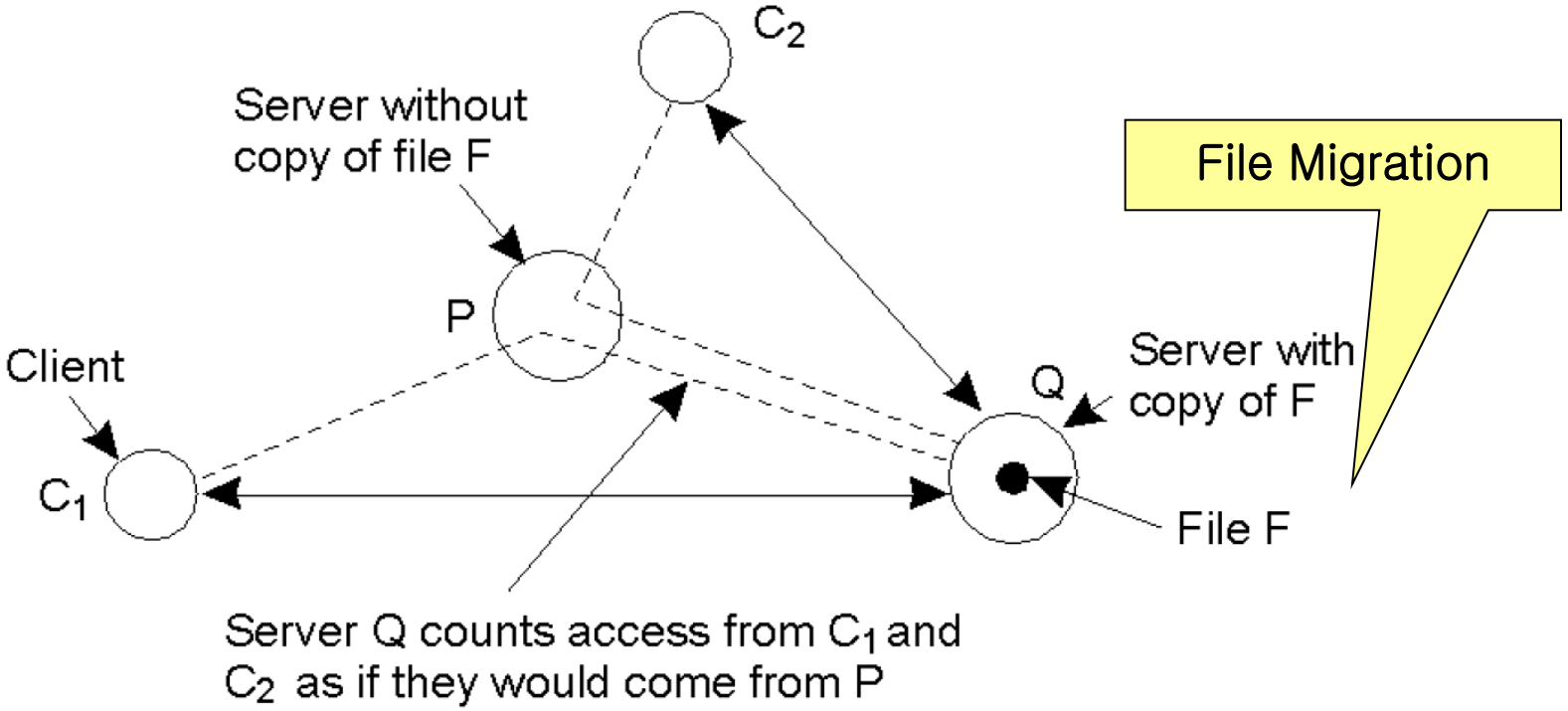
- Read-Your-Writes Consistency
  - When a client performs a read at a server, that server is handed the client's write set to check whether all the identified writes have been taken place locally
- Writes-Follow-Reads Consistency
  - When a client performs a write at a server, that server is handed the client's read set to check whether all the identified writes have been taken place locally

Replica Placement



- Permanent Replicas
  - Distribution of a Web site
    - Replication across servers
    - Mirroring
- Server-Initiated Replicas
- Client-Initiated Replicas
  - (Client) caches

Server-Initiated Replicas



Counting access requests from different clients.

Pull vs Push Protocols

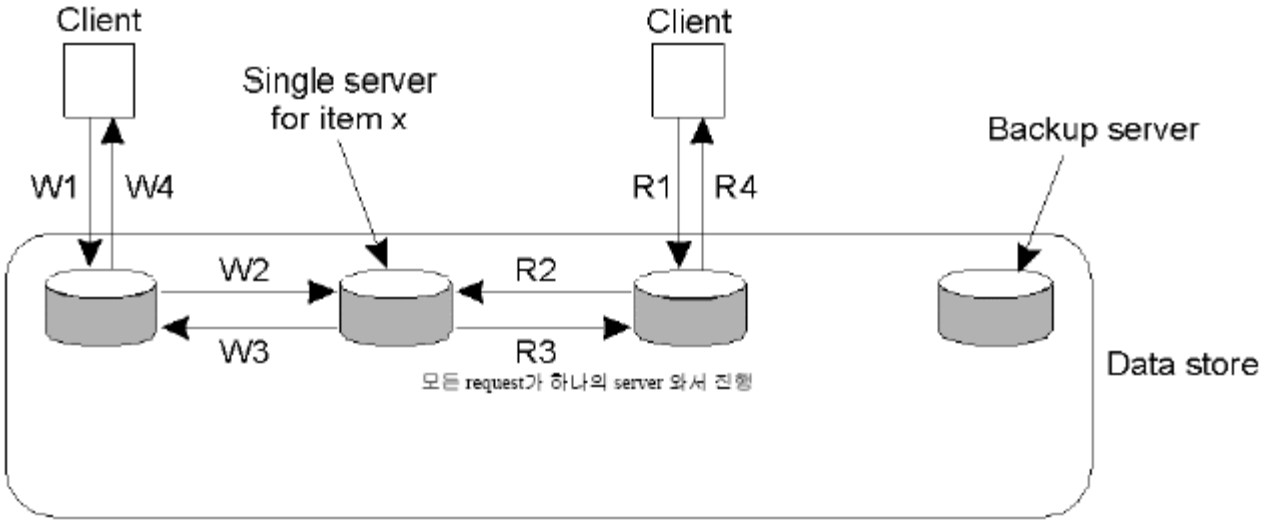
Issue	Push-based	Pull-based
State of server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

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

Consistency Protocols

- Protocols for Sequential Consistency Classified by Whether or Not There Is a Primary Copy of data to Which All Write Operations Should Be Forwarded
  - Primary-Based Protocols
  - Replicated-Write Protocols
    - A write operation can be initiated at any replica

Remote-Write Protocols

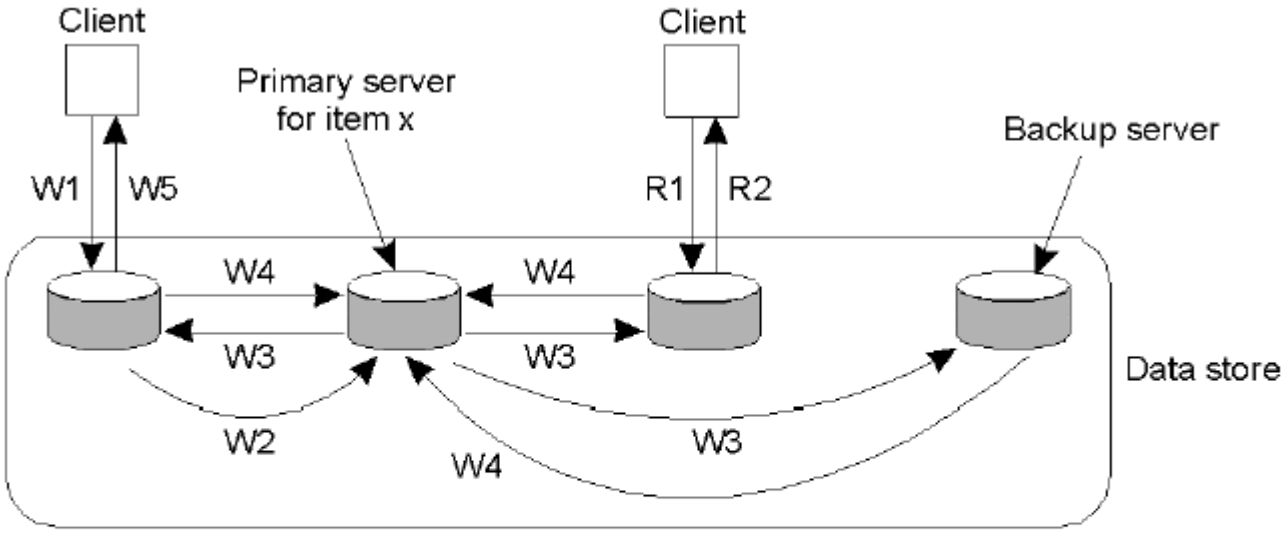


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

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.

Remote-Write Protocols (Cont'd)

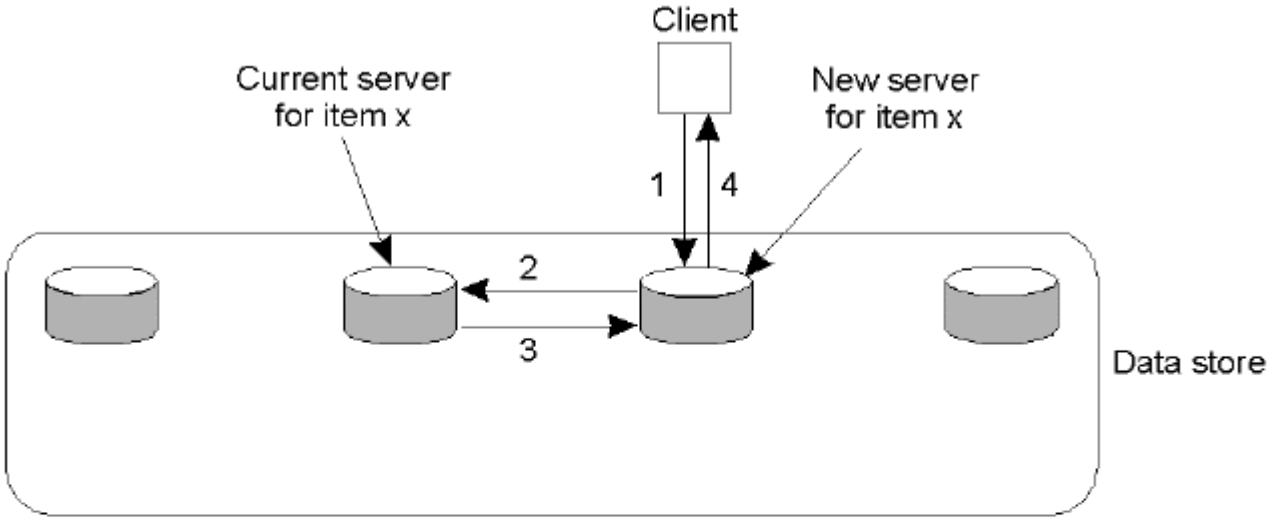


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

The principle of primary-backup protocol.

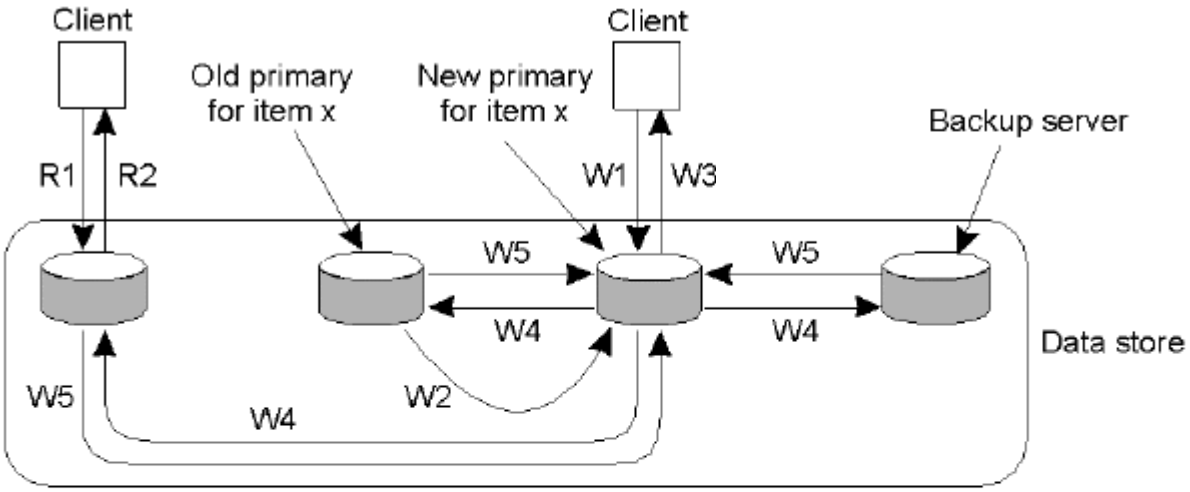
Local Write Protocols



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.

Local Write Protocols (Cont'd)



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

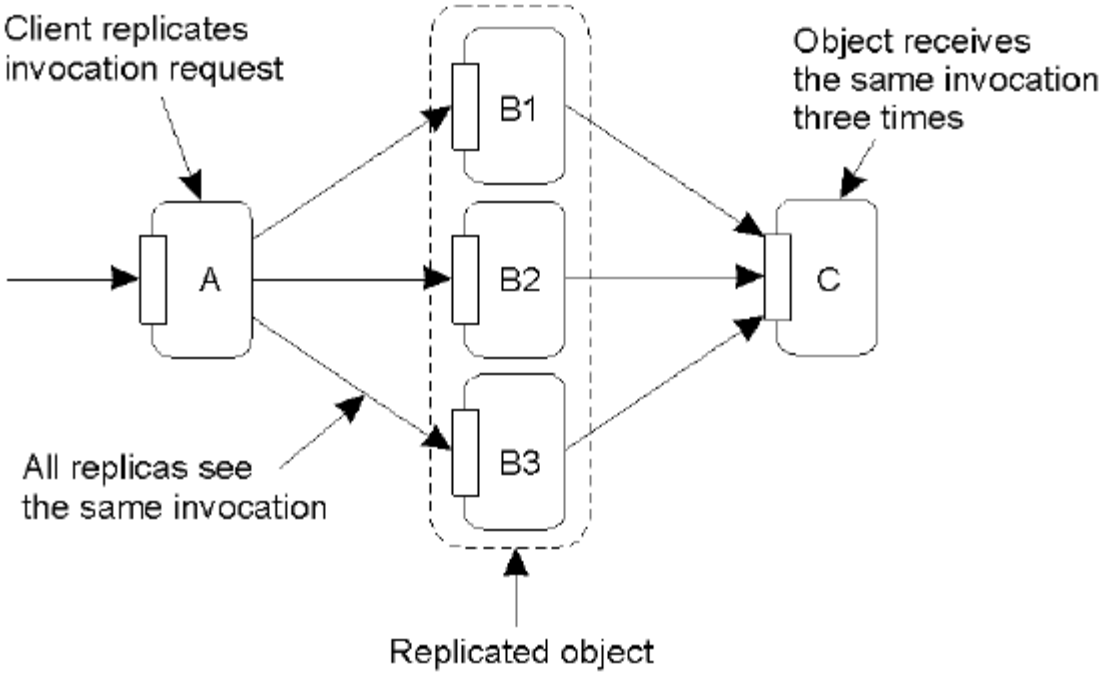
Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

Active Replication

- An Operation Is Forwarded to Any Replica with an Associated Process Carrying Out Update Operations
  - Potential Problems
    - Operations need to be carried out in the same order everywhere
      - Totally ordered multicast mechanism
      - Sequencer, a central coordinator
    - Invocations can be replicated

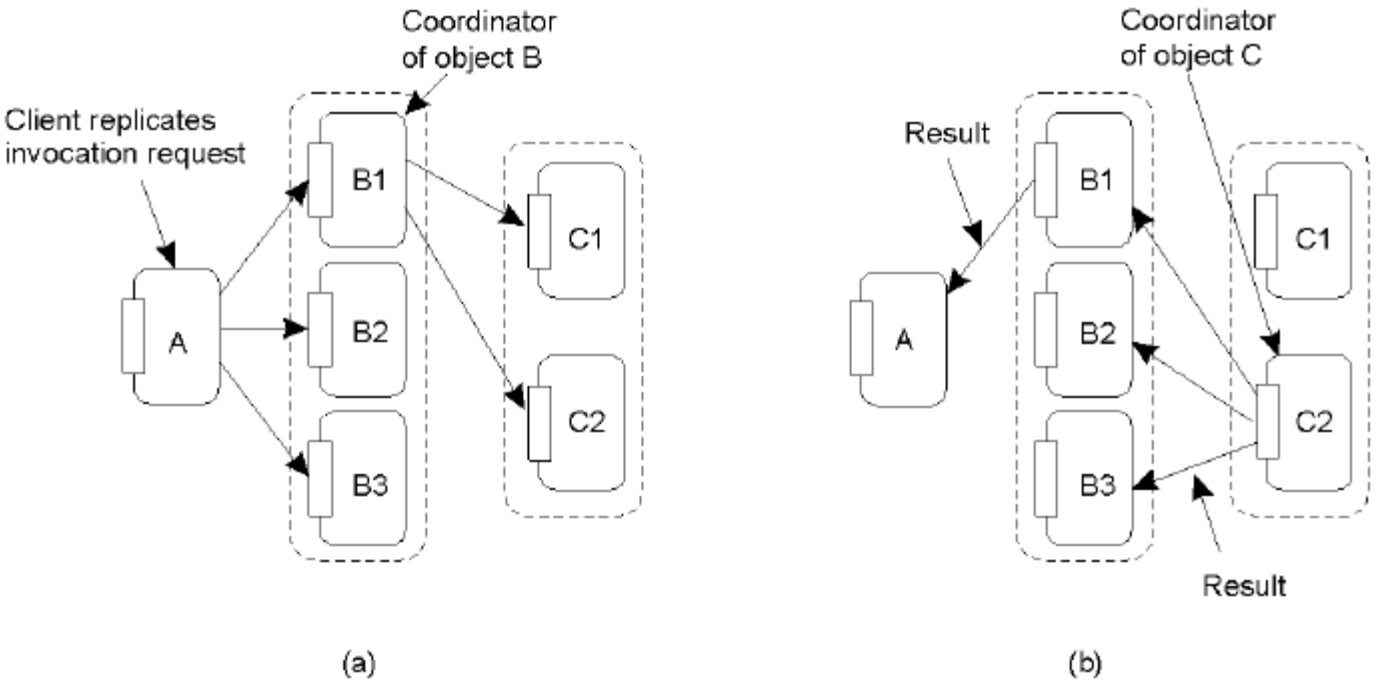
Scalability Problem!

Replicated Invocation Problem



The problem of replicated invocations.

Replicated Invocation Solution



a) Forwarding an invocation request from a replicated object.
b) Returning a reply to a replicated object.

Quorum-Based Protocol

- Idea: To Require Clients to Request and Acquire the Permission of Multiple Servers before Carrying Our Any Operation
  - To update a file, a client must first contact at least half the server plus one (a majority) and get them to agree to do the update
  - To read a replicated file, a client must also contact at least half the servers plus one and ask them to send the version numbers associated with the file

Fault Tolerance

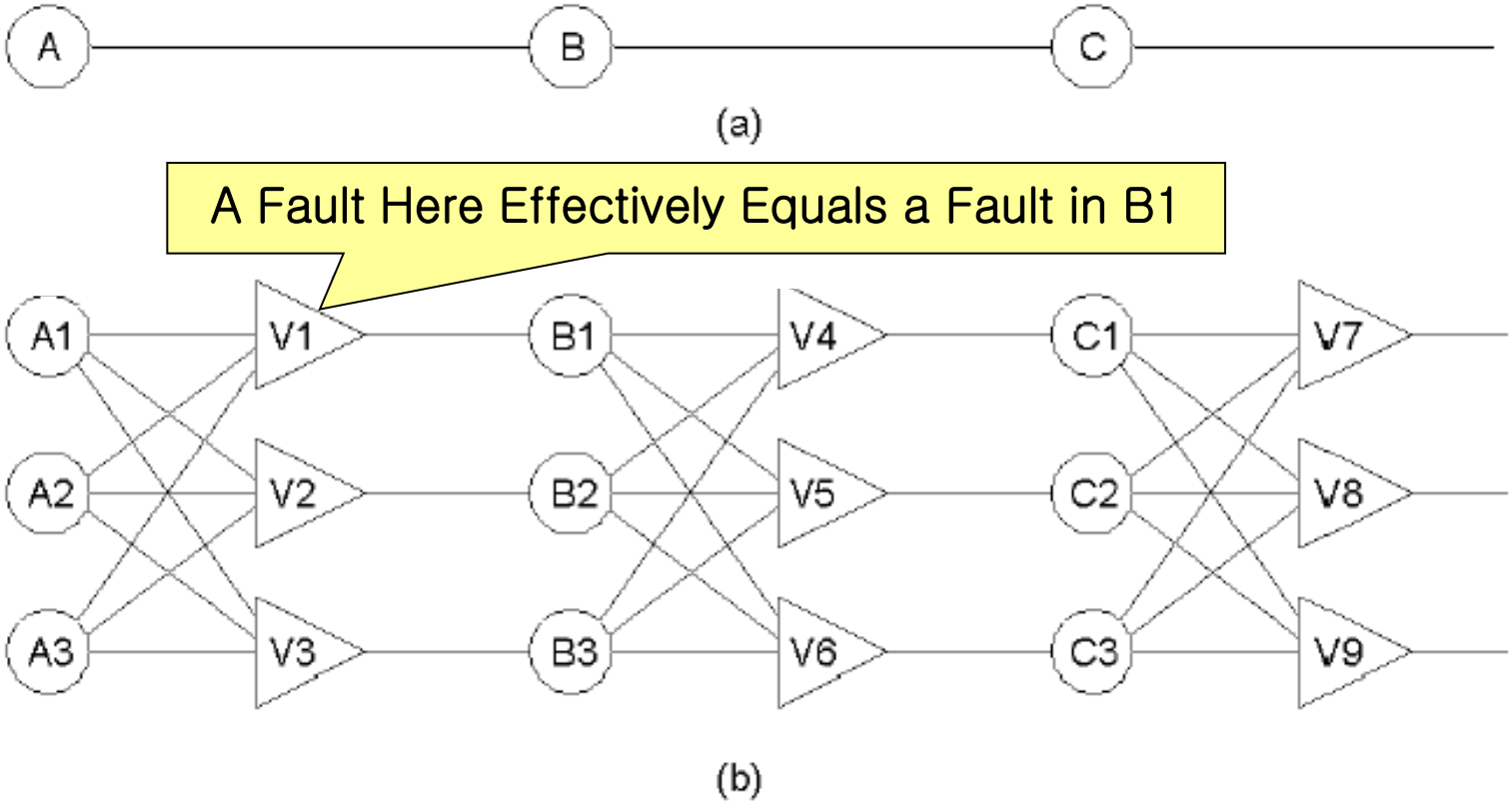
- Dependability
  - Availability
    - Ready to be used immediately
  - Reliability
    - Running continuously without failure
  - Safety
    - Not leading to catastrophe in the case of temporary failure
  - Maintainability
    - How easy a failed system can be repaired

Failure Modes

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Different types of failures.

Failure Masking by Redundancy

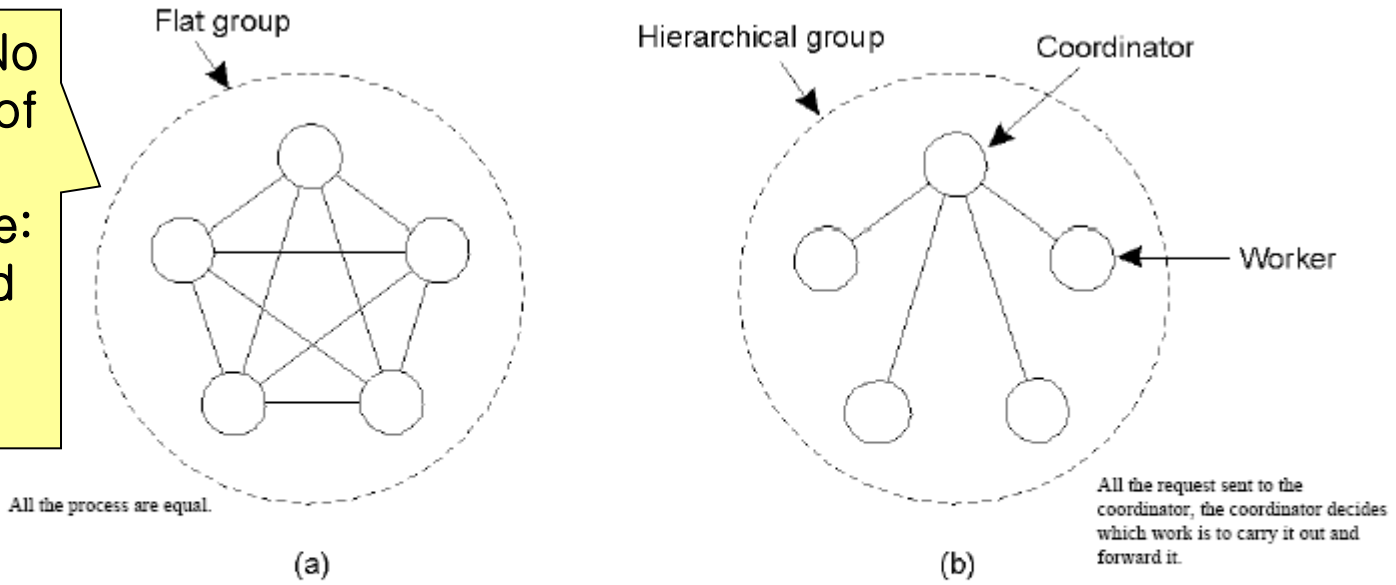


Triple modular redundancy.

Process Resilience

- Flat Groups vs Hierarchical Groups

Advantage: No Single Point of Failure
Disadvantage: Complicated Decision Making

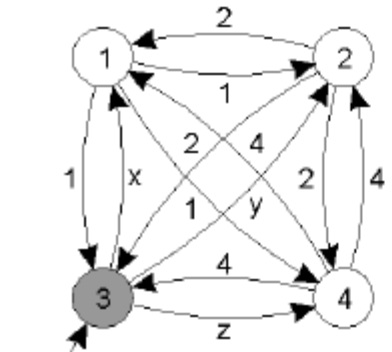


a) Communication in a flat group.
b) Communication in a simple hierarchical group

Agreement in Faulty Systems

- Two-Army Problem: Agreement Even between TWO Processes Is Not Possible in the Face of Unreliable Communication
- Byzantine Generals Problem
  - Whether to reach an agreement with loyal generals and traitors

Agreement in Faulty Systems (Cont'd)



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b) The result of (a) are collected together, then broadcast

Examines the ith element of received vectors,if no value has a majority, marked UNKNOWN.

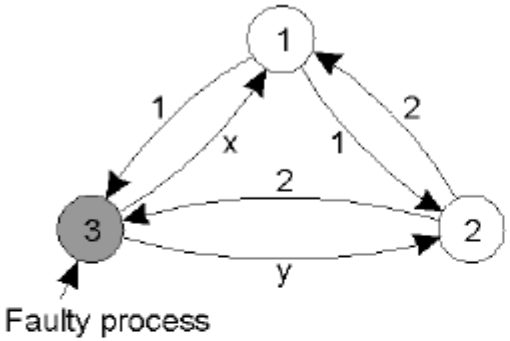
1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c) 1,2,4 all come to agreement on (1,2,UNKNOWN,4), so traitor 3 can't corrupt the information and can't gum up the works.

The Byzantine generals problem for 3 loyal generals and1 traitor.

a) The generals announce their troop strengths (in units of 1 kilosoldiers).
b) The vectors that each general assembles based on (a)
c) The vectors that each general receives in step 3.

Agreement in Faulty Systems (Cont'd)



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

All of the elements marked UNKNOWN,so, the algorithm has failed to produce agreement.

The same as in previous slide, except now with 2 loyal generals and one traitor.

A System with m Faulty Processes, Agreement Can Be Achieved Only If 2m+1 Correctly Functioning Processes Are Present, for a Total of 3m+1