Distributed Systems (CS 543)

Replication & Fault Tolerance

Dongman Lee

Dept of CS

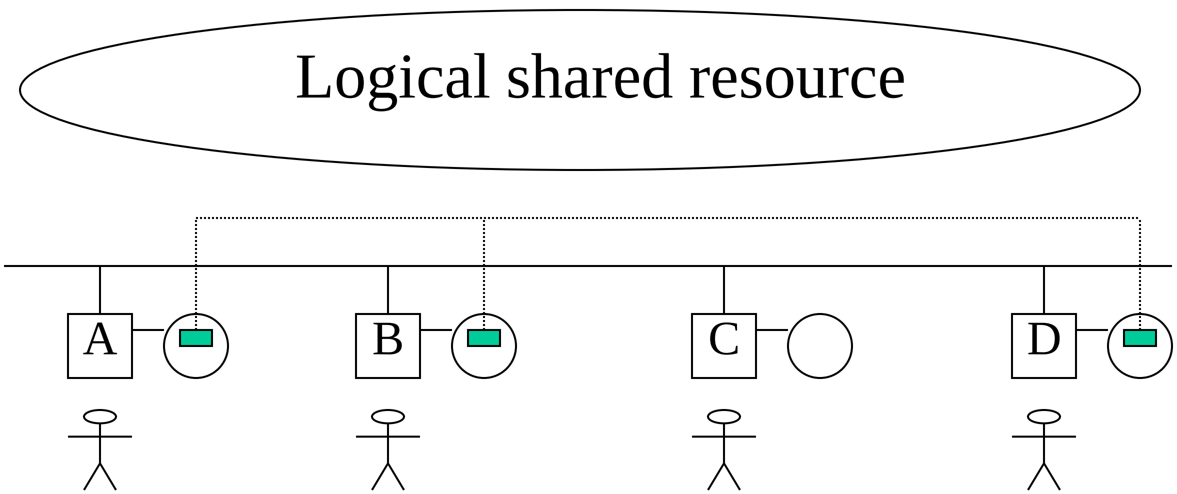
KAIST

Class Overview

- Why Replication?
- Replication Model
- Replication Consistency Protocols
- What is Fault?
- Fault Model
- Fault Tolerant Approaches
  - state machine
  - primary-backup

Why Replication?

- Purpose
  - increase availability, dependability and/or performance without knowledge of replica visibility
- Replication transparency
  - hiding the replication of state in a system
    - active vs. primary/stand-by replicas
    - generic functions: active and passive replication mechanisms



Replication Model

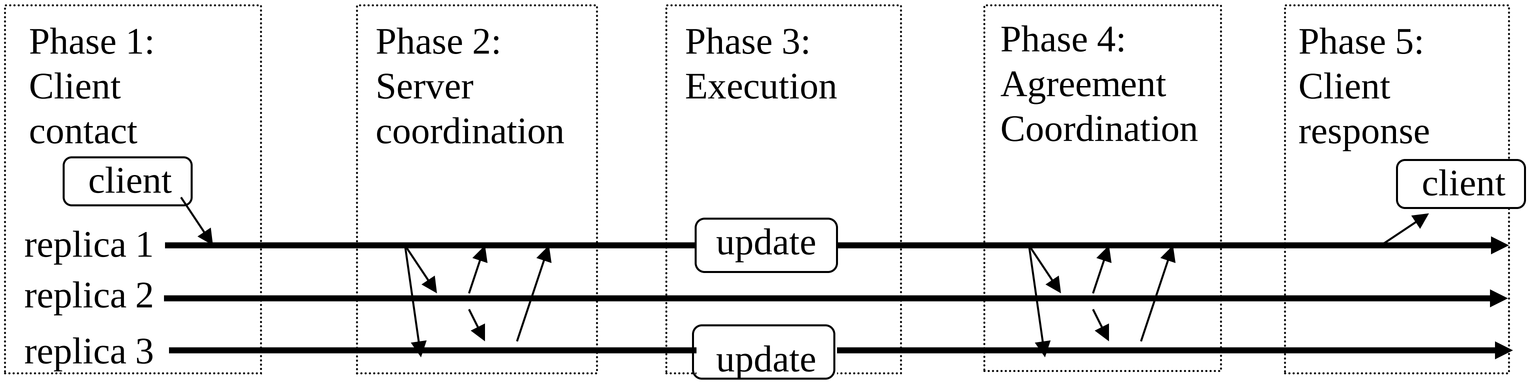
- Replication model spectrum
  - consistency
    - totally synchronous model
      - complete synchronization among replicas
    - asynchronous model
      - asynchronous update of replicas - that is, allow temporal inconsistency among replicas
    - most replication models are somewhere between these two models
  - purpose
    - performance improvement
      - reduction of delay by caching or replicating a server near clients
    - availability
      - make the service accessible (close to 100%) in the presence of process and network failures (partition and disconnection)
    - fault tolerance
      - guarantee strictly correct behavior despite of failures (byzantine and crash)

Replication System Model (cont.)

- Replication model
  - Active replication
    - deterministic execution
    - request sent to replicas using atomic totally ordered multicast
    - no need of agreement
  - Passive replication
    - non-deterministic execution
    - view synchronization
    - no need of server coordination
  - Semi-active replication
    - non-deterministic execution
    - request sent to replicas using atomic totally ordered multicast
    - leader informs followers of its choice using view synchronization
  - Semi-passive replication
    - same as passive without view synchronization
    - allow for aggressive time-outs values and suspecting crashed processes without incurring too high cost for incorrect failure suspicions

Replication System Model [Weismann]

- Replication protocol model
  - Request phase
    - active replication
    - passive replication
  - Server coordination
    - message ordering: FIFO, causal, total
  - Execution
  - Agreement coordination
    - necessary in database while ordering guarantee is enough for distributed systems
  - Client response
    - synchronous vs. lazy or asynchronous

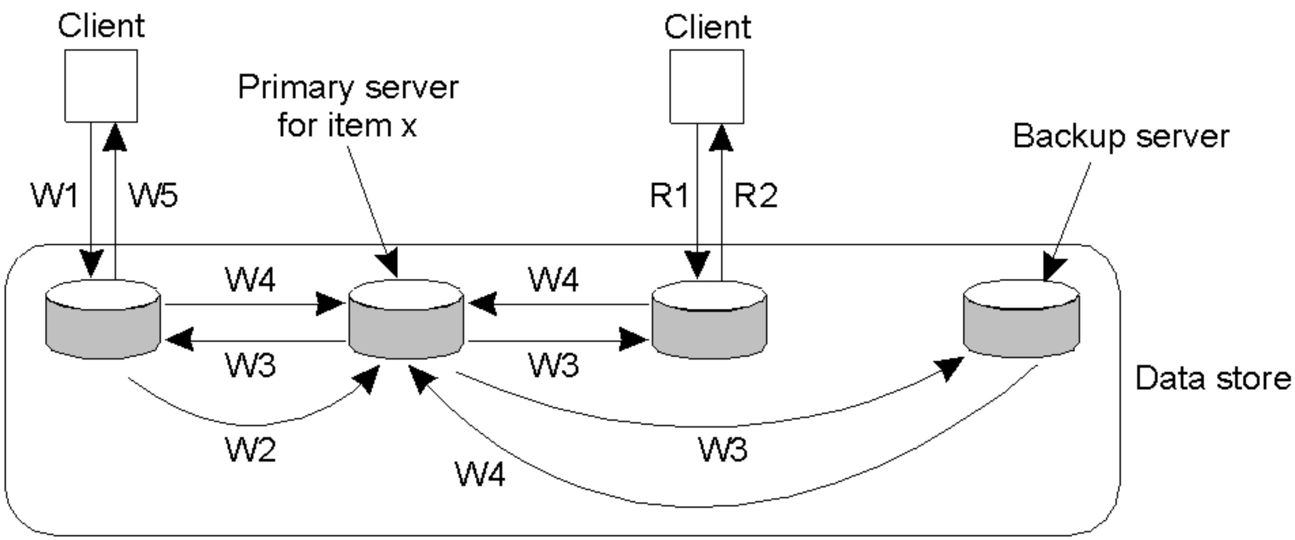


Replication Consistency Protocols

- Description
  - describe an implementation of a specific consistency model
- Classification
  - primary-based protocols
    - remote-write protocols
    - local-write protocols
  - replicated-write protocols
    - active replication
    - quorum-based protocols

Primary-based Remote-Write Protocols

- All write operations are performed at a (remote) fixed server
  - read operations are allowed on a local copy while write operations are forwarded to a fixed primary copy



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

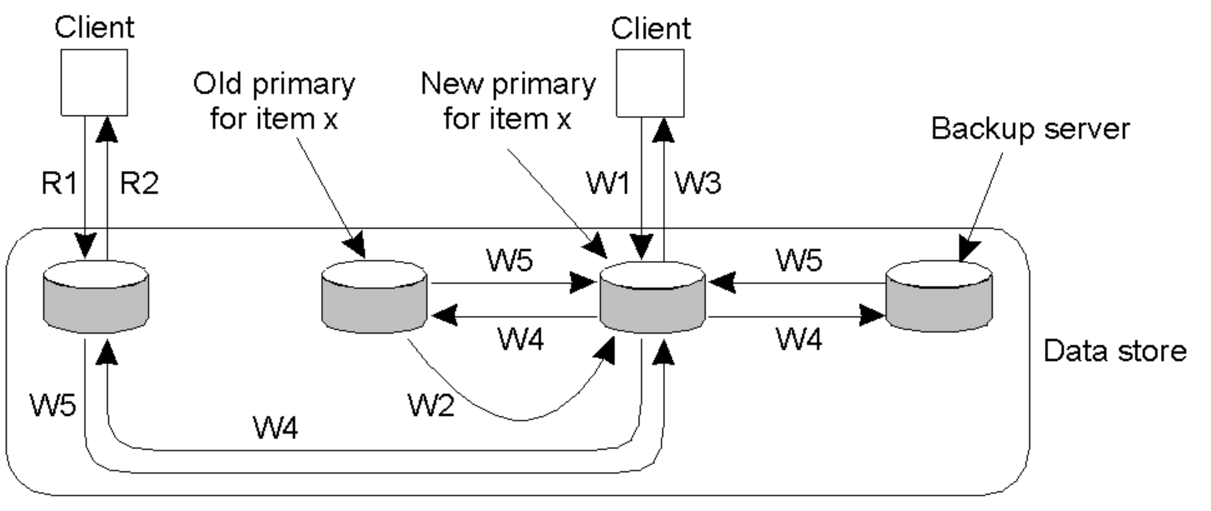
R1. Read request
R2. Response to read

Primary-based Remote-Write Protocols (cont.)

- Issues
  - update can be a performance bottleneck if implemented as a blocking operation
    - but guarantees sequential consistency (most recent write as the result of a read)
    - if implemented as a non-blocking, the protocol provides no guarantee of sequential consistency and fault tolerance

Primary-based Local-Write Protocols

- All write operations are performed locally and forwarded to the rest of replicas
  - primary copy migrates between processes that wish to perform a write operation
  - Multiple, *successive* writes can be done locally (via non-blocking protocol)
  - can be exploited in mobile computing

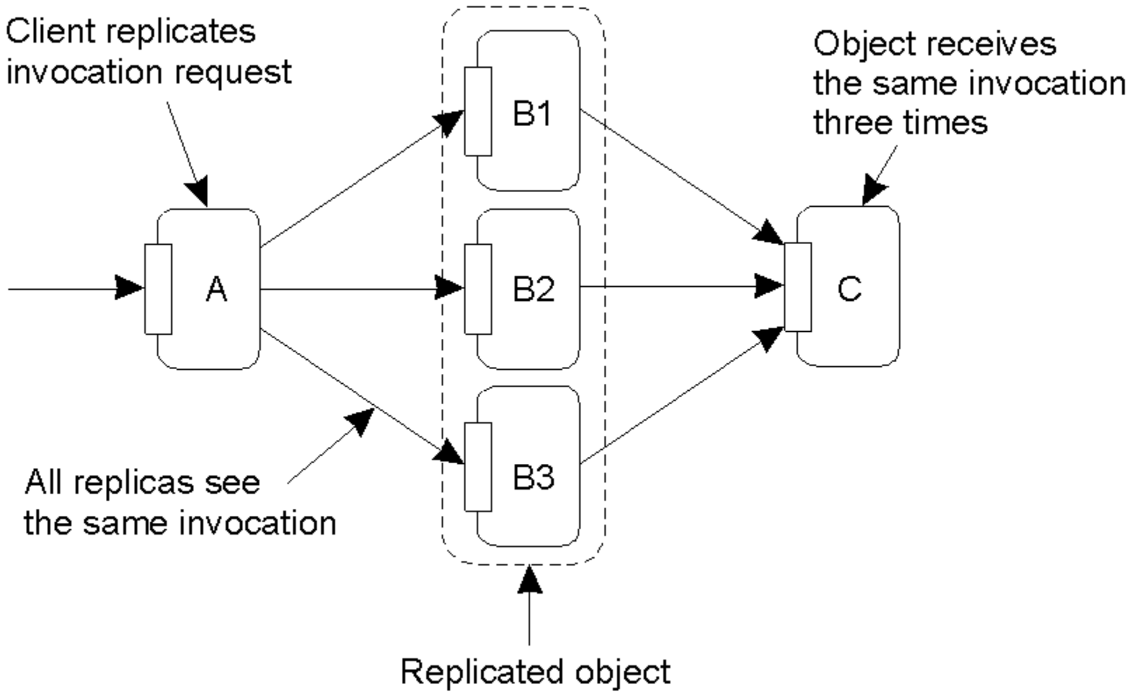


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

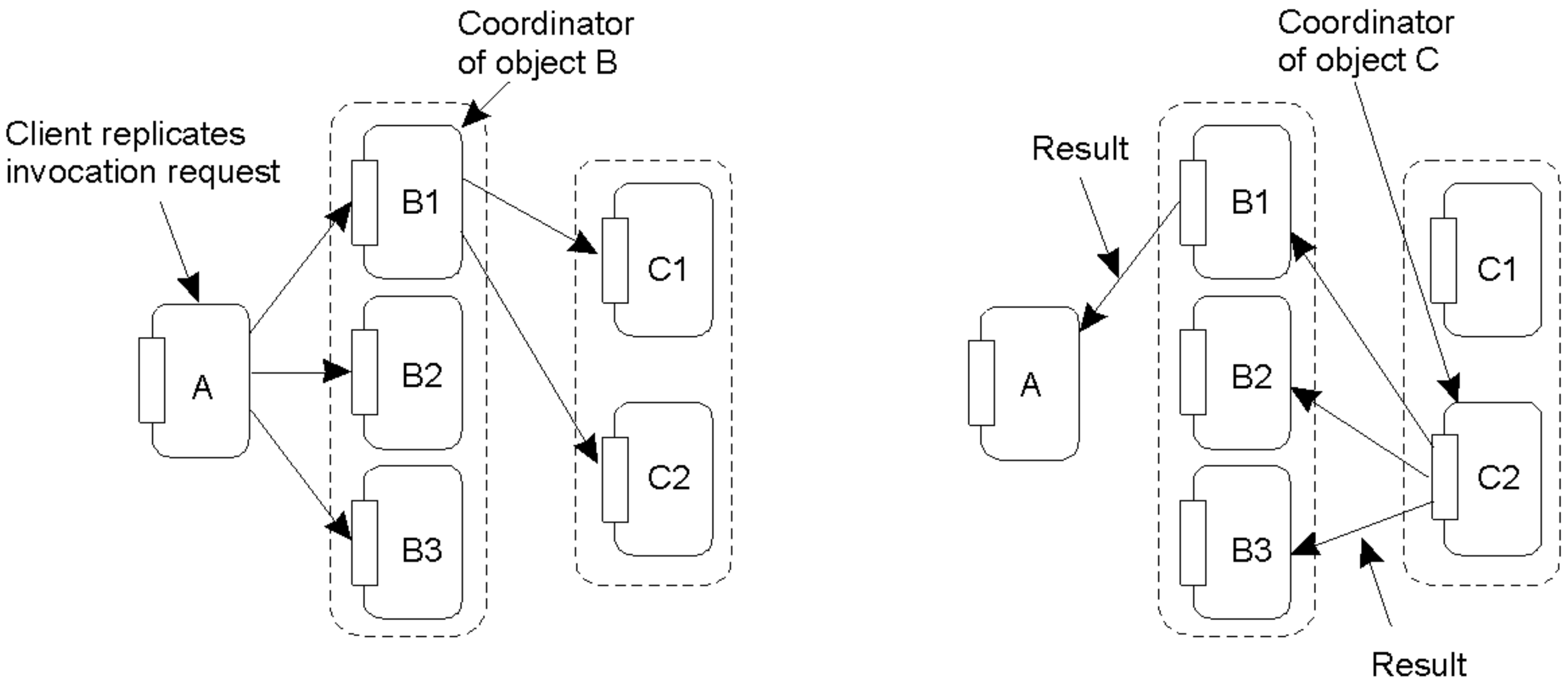
Active Replication

- Each replica performs update operations and propagates them (or the results) to the others
  - requires totally ordered multicast
- Replicated invocation problem



Active Replication (cont.)

- Solutions to the replicated invocation problem
  - group coordinator
  - sender-driven vs. receiver-driven

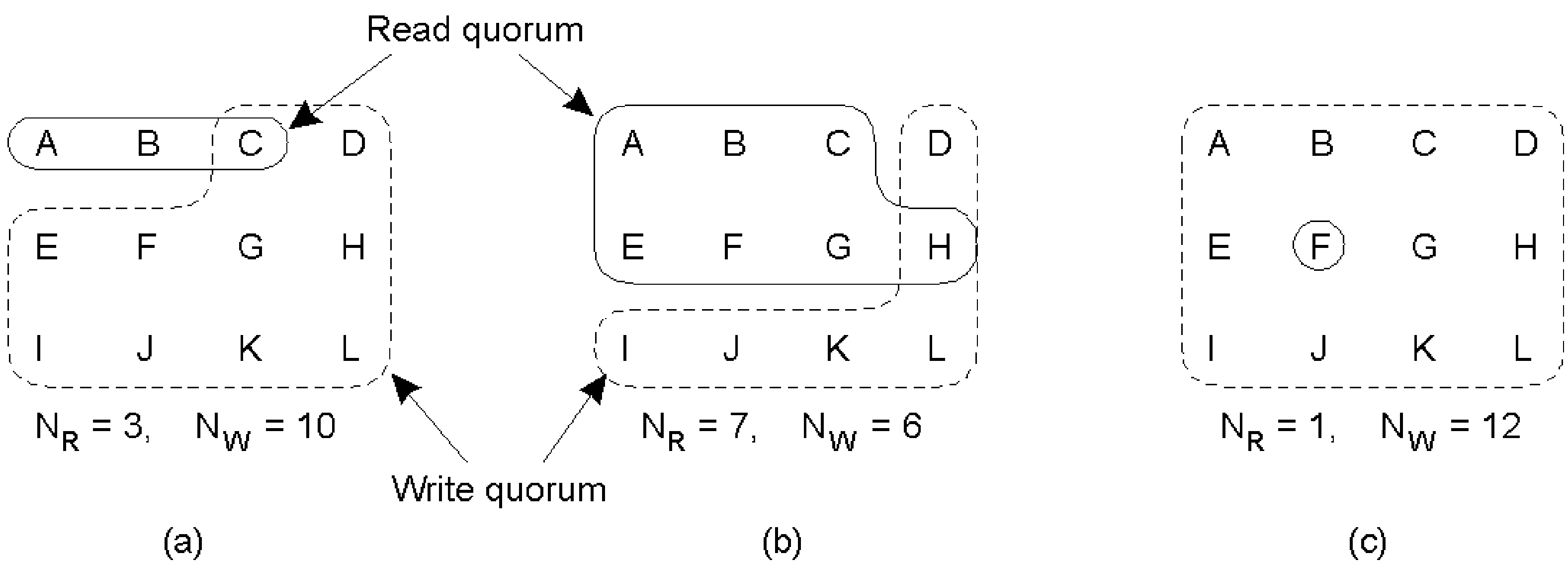


Quorum-based Protocols

- Require clients to request and acquire the permission of multiple servers before any operation on replicas
  - quorum set
    - W > half the total votes
    - R + W > total number of votes for group
      - any pair of read quorum and write quorum must contain common copies, so no conflicting operations on the same copy
    - read operations
      - check if there is enough number of copies >= R
      - perform operation on up-to-date copy
    - write operations
      - check if there is enough number of up-to-date copies >= W
      - perform operation on all replicas

Quorum-based Protocols (cont.)

- Examples



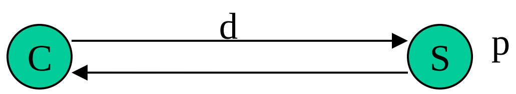
a) A correct choice of read and write set
b) A choice that may lead to write-write conflicts since W <= N/2
c) A correct choice, known as ROWA (read one, write all)

What is Fault?

- Definition
  - system is considered *faulty* once its behavior is no longer consistent with its specification [Schneider]
- Separation property of distribution systems lead to *partial failure property*
  - components that one component depends on may fail to respond due to various reasons
    - system or network failure
    - system or network overload
- Dependability
  - availability: readiness of use
  - reliability: continuity of service delivery
  - safety: low chances of catastrophes
  - maintainability: repairability

Failure Model

- Failure semantics
  - description of the ways in which a service may fail



  - recovery actions depend on the likely failure behavior of a server when its failure is detected
  - designers should ensure that the behavior of a server conforms to a specified failure semantics
    - e.g. network with omission/time failure semantics
      - need to guarantee detection of message corruption such as checksum
    - stronger failure semantics costs more in general
  - adequacy of failure semantics would require preliminary stochastic analyses

Failure Model (cont.)

- Representative faulty behavior
  - Fail-stop failures
    - when system fails, it changes to a state that allows others to detect its failure and then stops
  - Byzantine failures
    - system exhibits arbitrary and malicious behavior which may collude with other systems

Fault-Tolerant Approaches

- Fault tolerance
  - can detect a fault and either fail predictably or mask the fault from users
  - hiding the occurrence of errors in system components and communications
  - ➲ incorporate *redundant* processing component to achieve fault tolerance
- *k-resilient/fault-tolerant*
  - a set of systems satisfies its specification if no more than *k* systems become faulty
  - k is chosen based on statistical measures of system reliability
    - Arbitrary failure: 2k+1 (identical group); 3k+1(non-identical; Byzantine failure)
    - fail-stop failure: k+1

Fault-Tolerant Approaches (cont.)

- Two approaches to support fault tolerance (fault masking)
  - hierarchical failure masking
    - hierarchical failure and recovery management
      - error detection in layered communication protocols
      - various levels of error abstraction in OS
  - group failure masking
    - state-machine approach
    - primary-backup approach
- Fault tolerance support can be done
  - hardware
    - stable storage
  - software
    - replicated servers

State-Machine Approach

- Requirements for k fault-tolerant state machine
  - all replicas receive and process the same sequence of requests
    - agreement: every non-faulty replica receives every request
      - specify the interaction behavior of a client with state machine replicas
      - relaxed for read-only request in fail-stop failures
    - order: every non-faulty replica processes requests it receives in the same relative order
      - specify the behavior of state machine replicas in term of how to process requests from clients
      - relaxed for commutative requests

State-Machine Approach (cont.)

- Agreement requirement
  - to satisfy agreement requirement, state-machines should support a message broadcasting protocol which conforms to
    - IC1: all non-faulty processors agree on the same value
    - IC2: if sender of request is non-faulty, then all non-faulty processors use its value as the one on which they agree
  - message broadcasting protocol is called Byzantine agreement protocol or reliable broadcast protocol

State-Machine Approach (cont.)

- Order requirement
  - to implement order requirement requires
    - assignment of unique identifier to each message
    - stability (a request is ready to be delivered once all the previous requests have been delivered) test
  - assumptions on order requirement
    - O1: requests issued by a single client to a given state machine *sm* are processed by *sm* in the order they were issued
    - O2: if the fact that a request *r* was made to a state machine *sm* by a client *c* could have caused a request *r'* to be made by a client *c'* to *sm*, then *sm* processes *r* before *r'*
  - three approaches
    - logical clock-based
    - synchronized real-time clock-based
    - replica-generated identifiers-based

State-Machine Approach (cont.)

- Order requirement: logical clock-based
  - only for failstop failures
  - unique id assignment: logical clock
    - LC1: timestamp is incremented after each event at p
    - LC2: upon receipt of a message with timestamp t, process p resets its timestamp T_p to max(T_p, t)+1
  - stability test
    - a request is stable at replica sm_i if a request with larger timestamp has been received by sm_i from every client running on a non-faulty processor
      - messages between a pair of processors are delivered in the order sent
      - processor p detects that a failstop process q has failed only after p has received q's last message sent to p

State-Machine Approach (cont.)

- Order requirement: synchronized physical clock-based
  - unique id assignment
    - no client makes two or more requests between successive clock ticks => every message will have greater timestamp than its previous message (satisfies O1)
    - degree of clock synchronization is better than minimum message delivery time => timestamps of two causally related messages issued by two clients will be such that earlier one should have lower timestamp than later one (satisfies O2)
  - stability test
    - request r is *stable* if local clock reads T and uid(r) < T-d (d: worst case message delivery time)
    - request r is *stable* if a request with larger uid has been received from every client

State-Machine Approach (cont.)

- Order requirement: replica-generated identifiers-based
  - 2 phases are used
    - phase 1: replicas propose uid as part of agreement protocol (SEEN)
    - phase 2: one of candidates is selected and becomes uid (ACCEPTED)
  - stability test
    - request r that has been accepted by sm_i is *stable* if there is no request that has
      - been seen by sm_i,
      - not been accepted by sm_i, and
      - for which $cuid(sm_i, r) <= uid(r)$ holds

        where $cuid(sm_i, r) = \max(SEEN_i, ACCEPT_i) + 1 + i$

        $SEEN_i$: largest $cuid(sm_i, r)$ assigned to any request r so far seen by sm_i

        $ACCEPT_i$: largest uid(r) assgined to any request r so far accepted by sm_i

        $uid(r) = \max_{smj \in NF} (cuid(sm_j, r))$ where *NF* be the set of replicas from which candidate unique identifiers(*cuid*'s) were received

Primary-Backup Approach

- Cost metrics of primary-backup protocols
  - degree of replication
    - # of servers for fault tolerance
  - blocking time
    - worst cast period between a request and its response in any failure-free execution
  - failover-time
    - worst-case period during which requests can be lost because there is no primary

  ⇒ Smallest degree of replication, blocking time, failover-time for k-fault-tolerance?

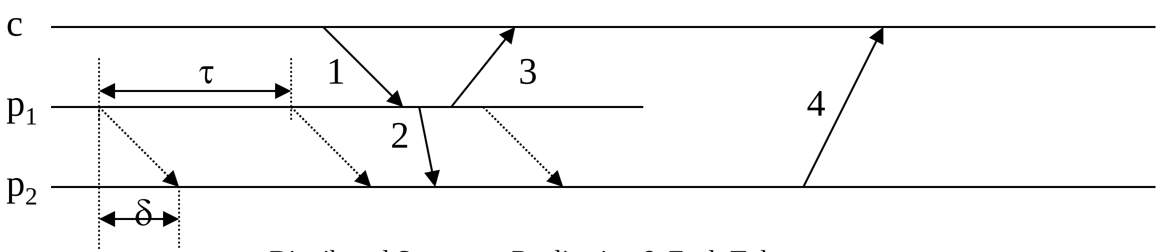
Primary-Backup Approach (cont.)

- Protocol properties
  - Pb1: there is at most one server whose state satisfies a condition being a primary
    - no more than one server is the primary at a time
  - Pb2: each client maintains a server identity to which the client can send a message
    - a client sends a request to the service by sending it to the server it believes to be the primary
  - Pb3: if a client request arrives at a server that is not a primary, then that request is not enqueued (thus, not processed)
    - messages to a backup are ignored
  - Pb4: there exist fixed value k and Δ such that the service behaves like a single (k, Δ)-bofo server*

    * (k, Δ)-bofo server (bounded outage, finitely often) : all server failures can be grouped into at most k intervals of time with each interval having length at most Δ

Primary-Backup Approach (cont.)

- Simple primary-backup protocol
  - assumption
    - one primary server p_1 and one backup server p_2, connected via a communication link (message delivery time upper bound: δ)
    - operations when p_1 receives a request from a client
      - processes the request and updates its state
      - send update info to p_2 (a state update message)
      - send a response to the client without waiting for ack from p_2
    - P1 sends a dummy message every τ seconds; If p_2 does not receive a dummy message for $\tau + \delta$ seconds, p_2 becomes a primary
  - spec conformance
    - Pb1: (p_1 has not crash) ^ (p_2 has not received a message from p_1 for $\tau + \delta$)=false
    - Pb2: client c sends a message to p_1
    - Pb3: requests are not sent to p_2 until after p_1 has failed
    - Pb4: a single $(1, \tau+4\delta)$-bofo server



Primary-Backup Approach (cont.)

- Failure models

  1. crash failures
     - permanent halt - once a server halts, it never recovers
  2. crash + link failures: 1+ link may lose messages
     - links do no delay, duplicate or corrupt messages
  3. receive-omission failures: 1+ failed to receive some of messages
  4. send-omission failures: 1 + failed to send some of replies
  5. general-omission failures: 3 + 4

State-machine vs. Primary-backup

- Comparison

	State-machine	Primary-backup	Remarks
Arbitrary Failure support	Yes	No	2k+1/3k+1 replication for k-resilience
Request loss	No	Possible	Loss happens when a primary fails
Failure handling	Voting	Failover	
Request copy	as many servers as k-resilience suffices	Only to primary	2k+1/3k+1 for arbitrary k+1 for fail-stop
Overall cost	expensive	cheap	Primary-backup approach is more popular in commercial applications