CHAPTER 4: INTERPROCESS COMMUNICATION AND COORDINATION

Chapter outline

- Discuss three levels of communication: basic message passing, request/reply and transaction communication based on message passing
- Discuss name services for communication
- Show examples of process coordination using message passing

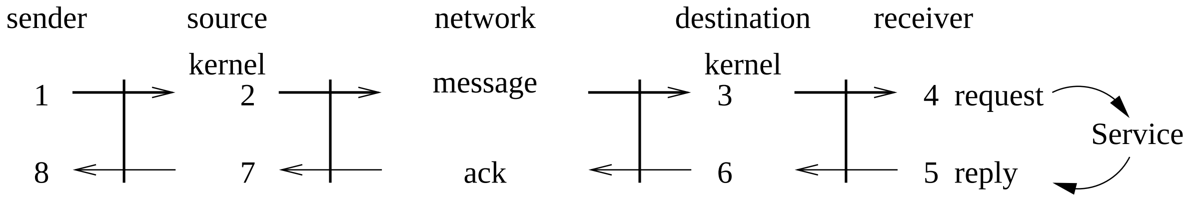
Basic message passing communication

Communication primitives:

send(destination, message)
receive(source, message)
channel naming = process name, link, mailbox, port

- *direct communication*: symmetric/asymmetric process naming, link
- *indirect communication*: many-to-many mailbox, many-to-one port

Message buffering and synchronization



1. **Nonblocking send,** *1+8* : Sender process is released after message has been composed and copied into sender's kernel (local system call)

2. **Blocking send,** *1+2+7+8* : Sender process is released after message has been transmitted to the network

3. **Reliable blocking send,** *1+2+3+6+7+8* : Sender process is released after message has been received by the receiver's kernel (kernel receives network ACK).

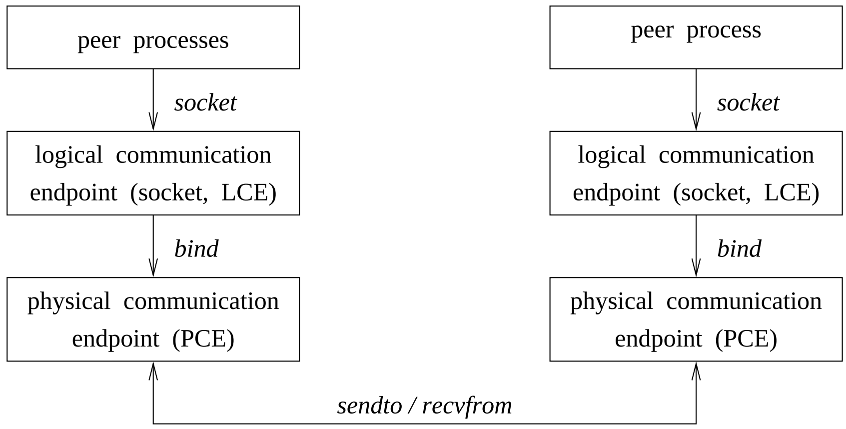
4. **Explicit blocking send,** *1+2+3+4+5+6+7+8* : Sender process is released after message has been received by the receiver process (kernel receives kernel delivery ACK)

5. **Request and reply,** *1-4, service, 5-8* : Sender process is released after message has been processed by the receiver and response returned to the sender

Message passing API

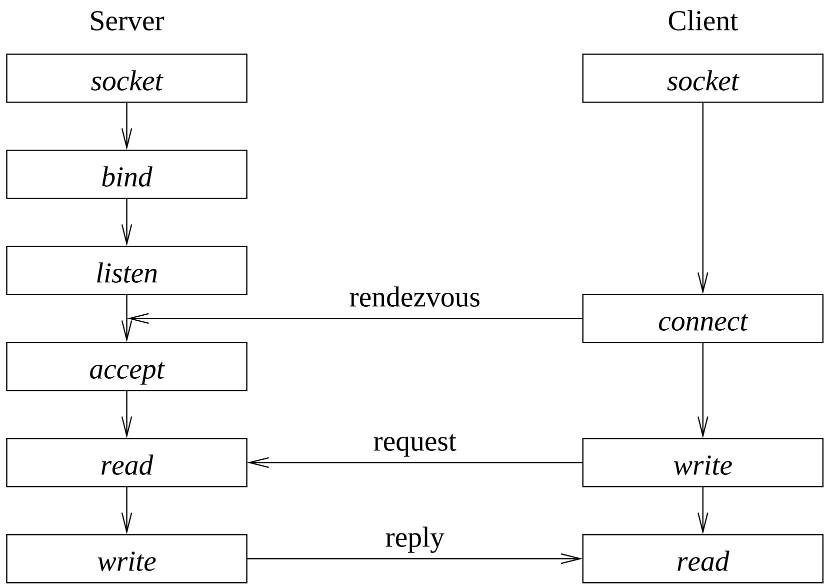
- *Pipe*: A FIFO byte-stream unidirectional link for related processes
- *Message queue*: A structured variable length message queue
- *Named Pipe*: A special FIFO file pipe using path name for unrelated processes under the same domain (explicitly created and accessed)
- *Socket*: A logical communication endpoint for communication between autonomous domains (bound to physical communication endpoint)

Connectionless socket communication



- *peer process*: application-level process - application protocol
- *LCE*: Logical Communication Endpoint - established with socket call
- *PCE*: Physcial Communication Endpoint - (a.k.a. endpoint in network) (Transport TSAP/L4SAP, Network NSAP/L3SAP) pair bound to LCE with *bind()* call
- *Network*: Accessed by *sendto()/recvfrom()* primitives

Connection-oriented socket communication



Asymmetric - Client and Server

Server Starts first:

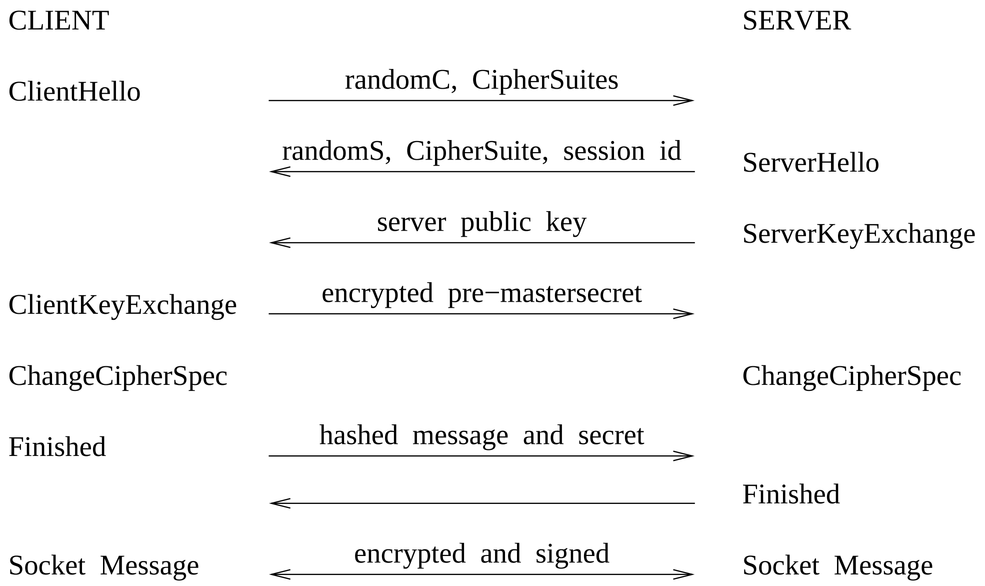
- *Server process*: application-level process - server protocol
- *LCE*: Logical Communication Endpoint - established with socket call
- *PCE*: Physcial Communication Endpoint - (a.k.a. endpoint in network) (Transport TSAP/L4SAP, Network NSAP/L3SAP) pair bound to LCE with *bind()* c all
- *Listen*: Server waits for incoming connection request
- *Accept*: Server accepts connection request, initializes connection
- *Read*: Server reads incoming segment(s) of request
- *Write*: Server writes reply segment(s)
- *Close*: Server terminates connection when reply is received

Client starts after Server:

- *Client process*: application-level process - client protocol
- *LCE*: Logical Communication Endpoint - established with socket call
- *PCE*: Physcial Communication Endpoint - (a.k.a. endpoint in network) (Transport TSAP/L4SAP, Network NSAP/L3SAP) pair bound to LCE with *connect()* call, which also initializes connection to Server PCE
- *Write*: Client writes request segment(s)
- *Read*: Client reads incoming segment(s) of reply
- *Close*: Client terminates connection when reply is received and acknowledged

Secure Socket Layer protocol

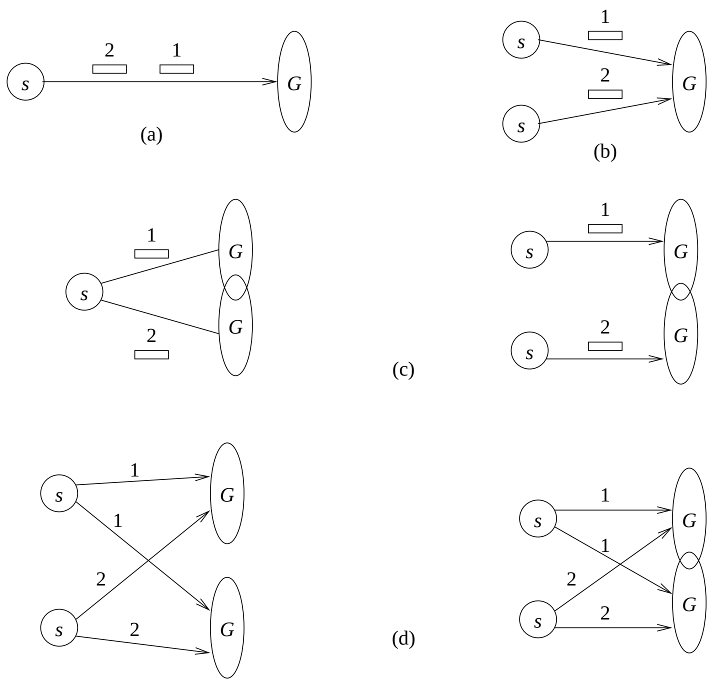
- *Privacy*: use symmetric private-key cryptography
- *Integrity*: use message integrity check
- *Authenticity*: use asymmetric public-key cryptography



- Server accepts connection, selects cipher suite both can use (if any), provides its public key in a signed certificate
- Client verifies Server public key certificate
- Client and Server exchange public information to establish shared secret
- Client and Server initialize hash key, session encryption key
- Either Client or Server may terminate secure connection

Group communication and multicast

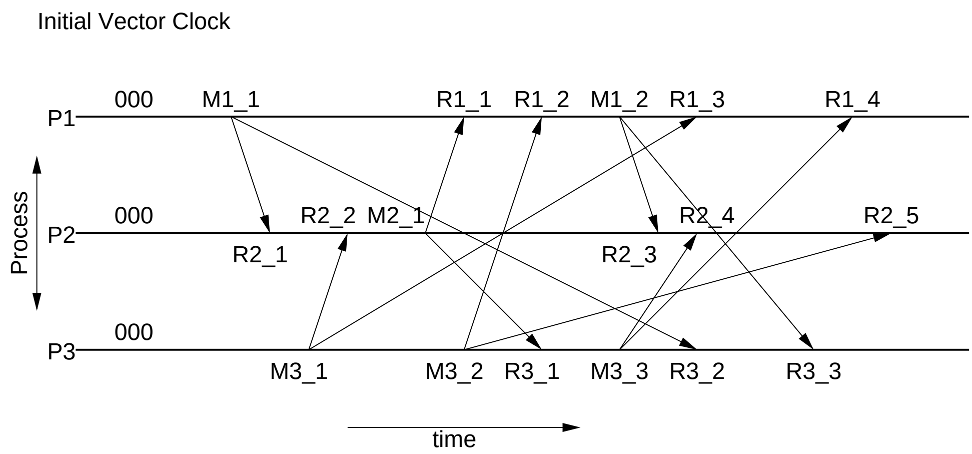
- Reliability of message delivery
  - *Best effort*
  - *Duplicate detection*
  - *Omission detection/recovery per receiver*
  - *All or none (atomic) to all receivers*
- *Orderly delivery*
  - FIFO (per sender)
  - Causal order
  - Total order



- (a) Single sender/single group - reliable, ordered deliver (FIFO)
- (b) Multiple senders/single group - order between senders' messages?
- (c-L) Single sender/overlapping groups - consistency of order of messages sent to different groups for nodes in intersection
- (c-R) Multiple, single group senders/overlapping groups - consistency of order of messages for nodes in intersection
- (d-L) Multiple, multi-group senders/independent groups - issues of (b) plus consistency of order in Group 1 and Group 2
- (d-R) Multiple, multi-group senders/overlapping groups - issues of (d-L) plus consistency of order for nodes in intersection of Group 1 and Group 2

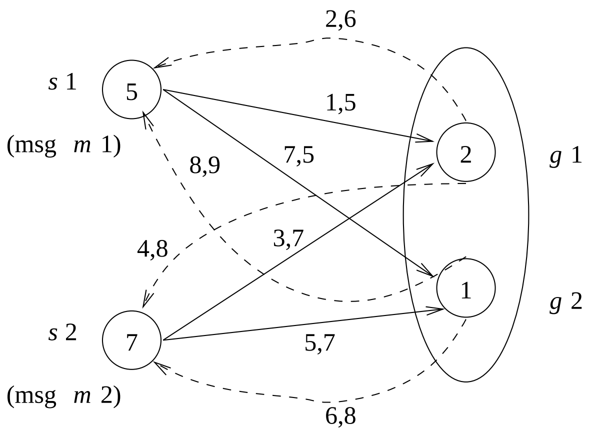
Causal order

- Accept message m if $T_i = S_i + 1$ and $T_k \leq S_k$ for all $k \neq i$.
- Delay message m if $T_i > S_i + 1$ or there exists a $k \neq i$ such that $T_k > S_k$.
- Reject the message if $T_i \leq S_i$.



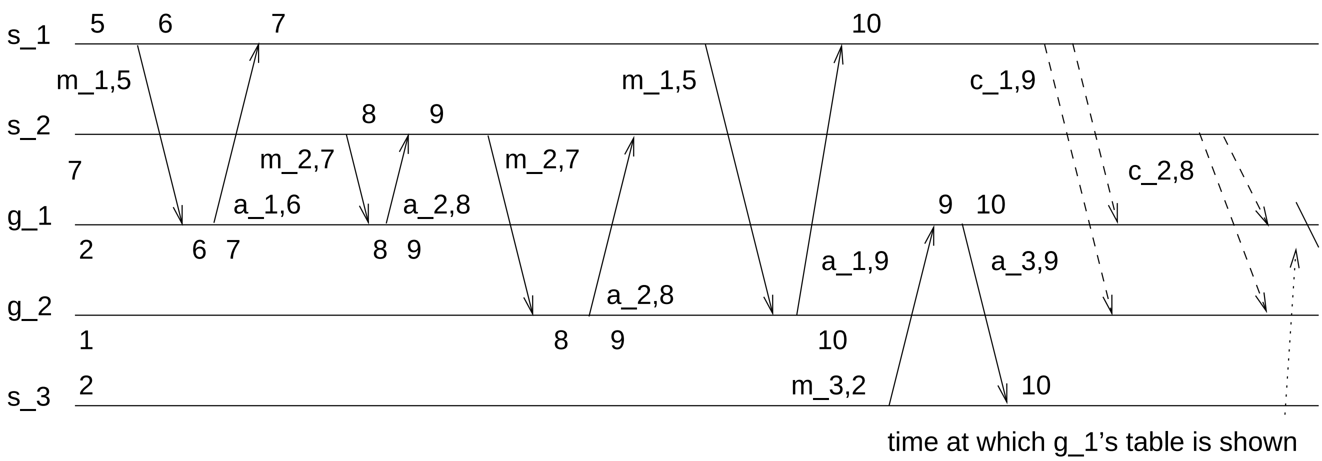
Message	Tx Timestamp	Rx Event	Action(s)	VLC after Rx
M1_1		R1_1		
M1_2		R1_2		
M2_1		R1_3		
M3_1		R1_4		
M3_2		R2_1		
M3_3		R2_2		
		R2_3		
		R2_4		
		R2_5		
		R3_1		
		R3_2		
		R3_3		

Total order



Multicast Message	Acknowledge Time	Commit Time
m 0	2	delivered
m 1	6	9
m 2	8	8
m 3	10	pending

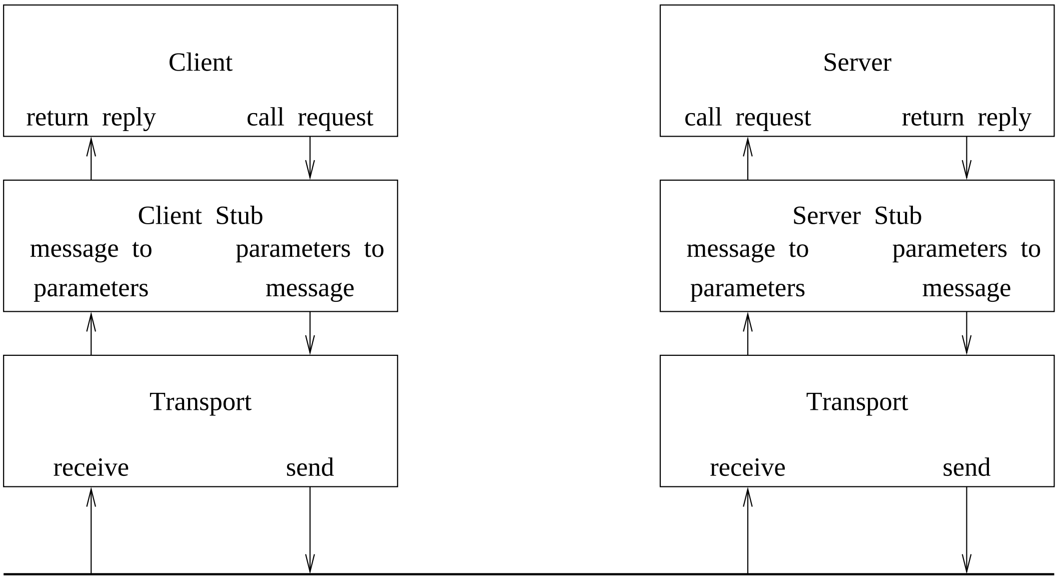
Buffer management in the communication handler



Total Order Multicast Example: Time−Space Diagram

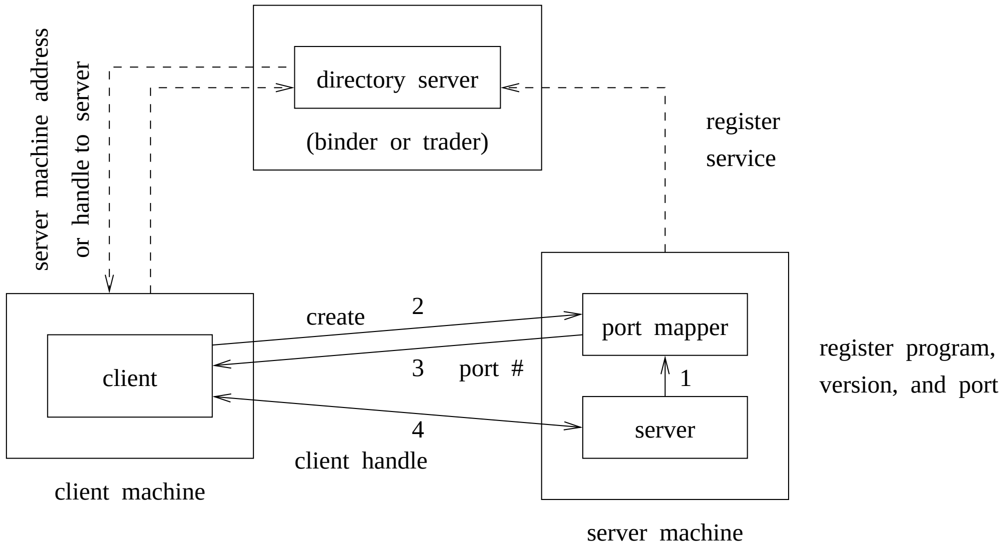
Request/reply communication

Remote Procedure Calls (RPCs)

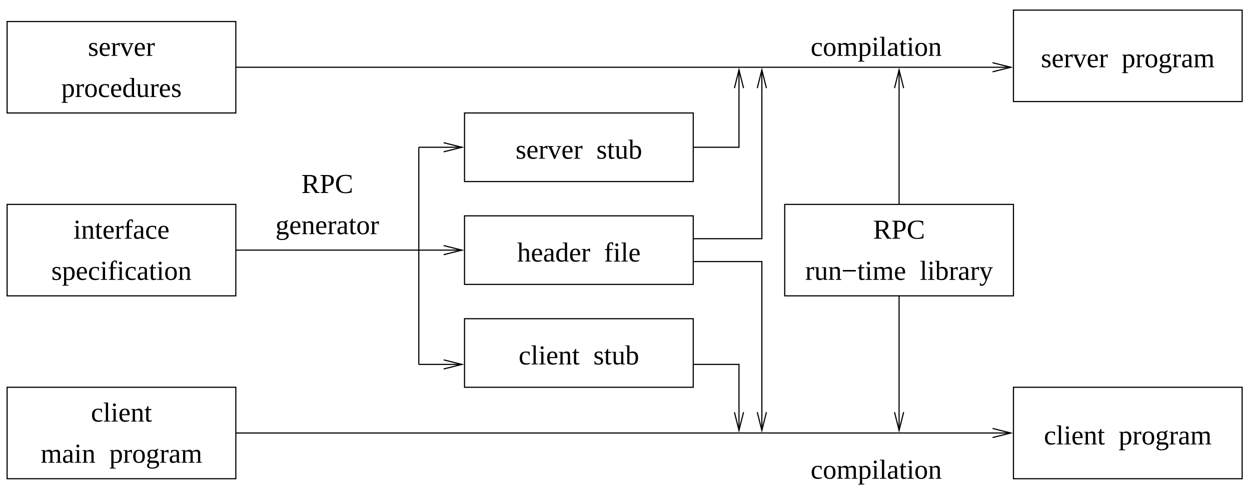


- *Parameter passing and data conversion*
- *Binding*
- *Compilation*
- *Exception and failure handling*
- *Security*

RPC Binding



RPC compilation



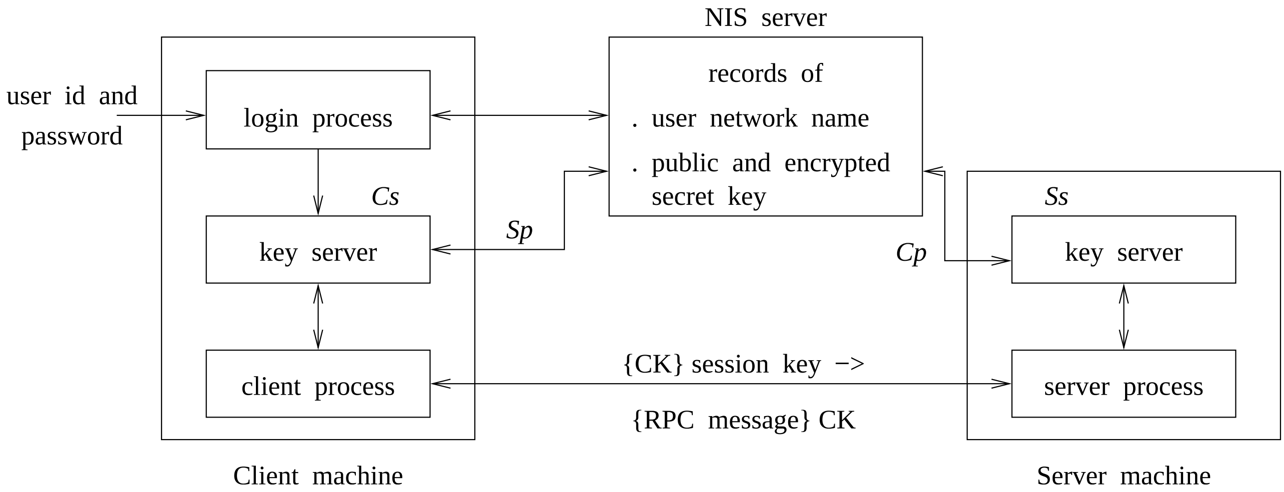
RPC exception and failure

- *Exception*: in-band or out-band signaling
- *Link failure*: retransmission, sequence number and idempotent requests, use of transaction id *xid*
- *Server crash*:
  - *at least once*: server raises an exception and client retries
  - *at most once*: server raises an exception and client gives up
  - *maybe*: server raises no exception and client retries
- *Client crash*:
  - orphan killed by client
  - orphan killed by server
  - orphan killed by expiration

Secure RPC

- C_s and S_s are 128-bit random numbers.
- $C_p = \alpha^{C_s}\ modM$, and $S_p = \alpha^{S_s}\ modM$, where α and M are known constants.

$$SK_{cs} = S_p^{C_s} = (\alpha^{S_s})^{C_s} = \alpha^{S_s * C_s}$$
$$SK_{sc} = C_p^{S_s} = (\alpha^{C_s})^{S_s} = \alpha^{C_s * S_s}$$



Transaction Communication

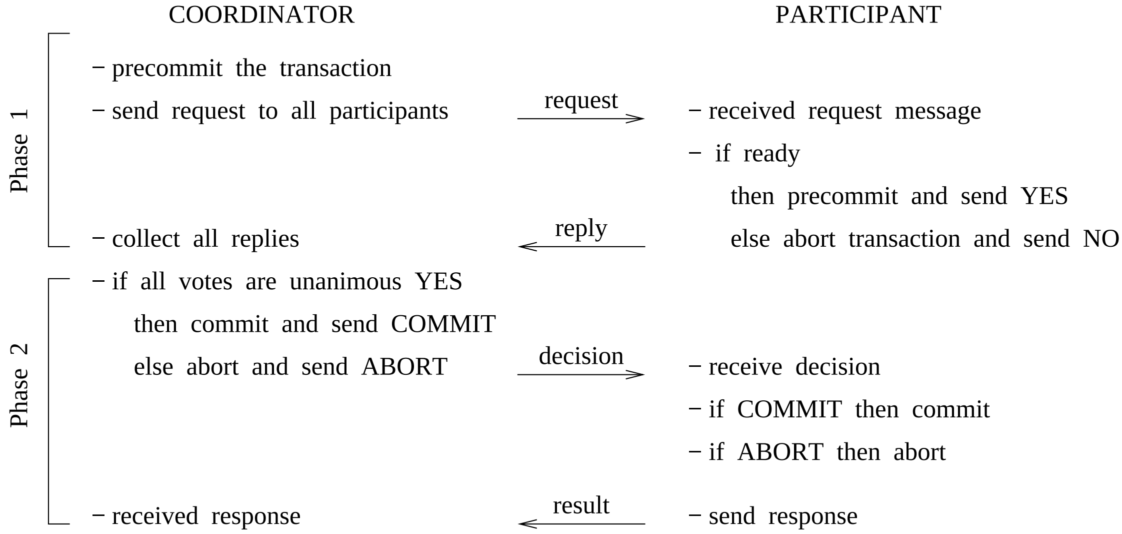
Must handle

- multiple operations
- many participants
- message failures
- node failures

ACID properties

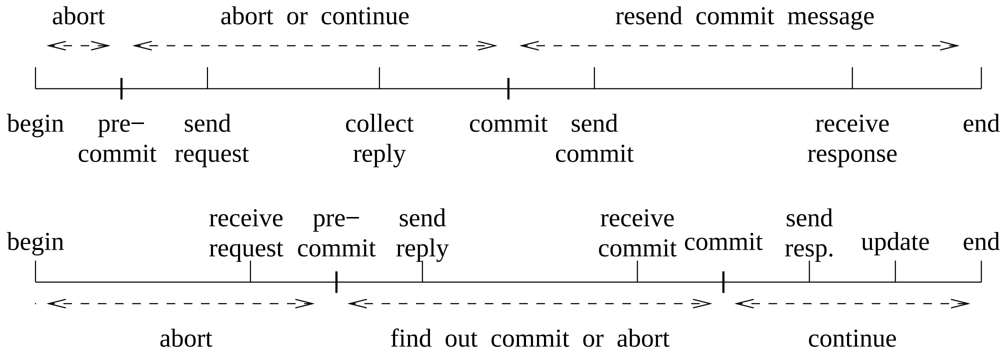
- *Atomicity* - either all operations take place or none of them,
- *Consistency* - the results of interleaved execution of operations of multiple transactions is equivalent to serial execution of the transactions
- *Isolation* - partial results of a transaction in progress are not visible to others
- *Durability* - once a transaction is committed, its results will be made permanent

Two-phase commit protocol



Failure and recovery of the 2PC protocol

Coordinator failure recovery actions



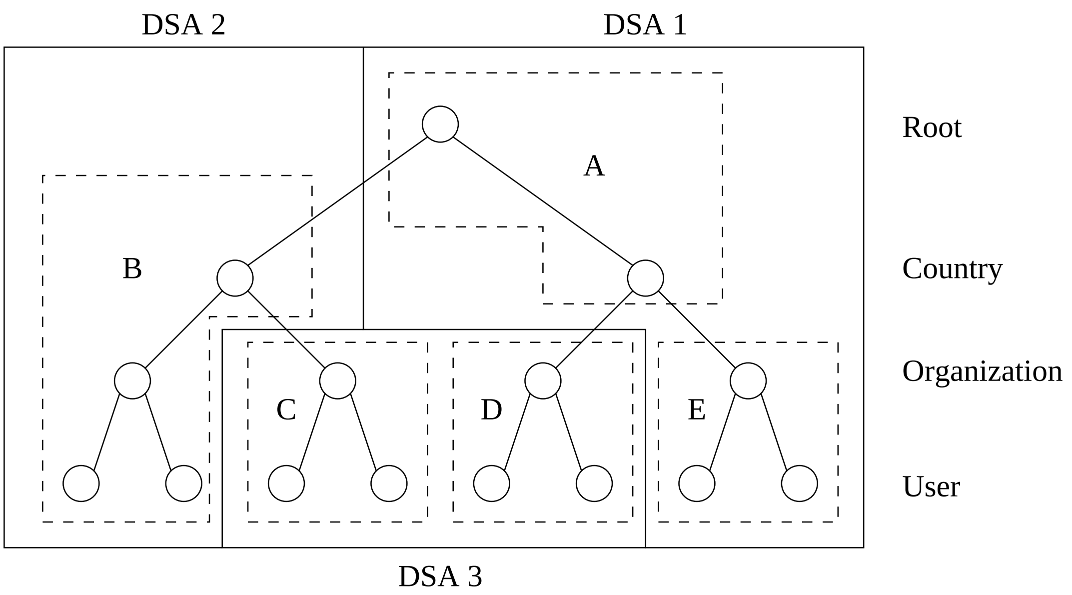
Participant failure recovery actions

Name and Directory Services

Object attributes and name structures

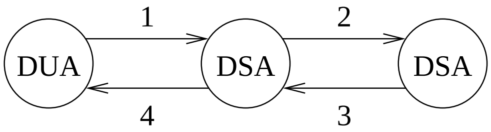
Service /object Attributes	Name Structures	Attribute Partitioning
< attributes > < name, attributes, address > < name, type, attributes, address >	flat structure hierarchical structure name−based resolution (e.g., white pages) structure−free attribute−based resolution (e.g., yellow pages)	physical organizational functional

Name space and information base

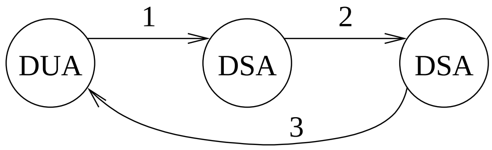


Five naming contexts of Directory Info Tree in three Directory Service Agents

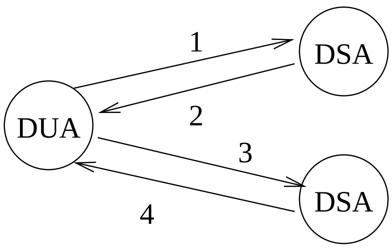
Name resolution



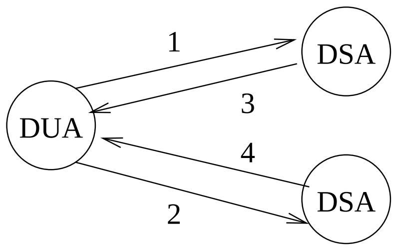
Recursive chaining



Transitive chaining



Referral



Multicast

Points of comparison:

- Simplicity at DUA
- Stateless DSA (with regard to requests)
- Ability of DSA to cache
- Message efficiency
- Response time at DUA

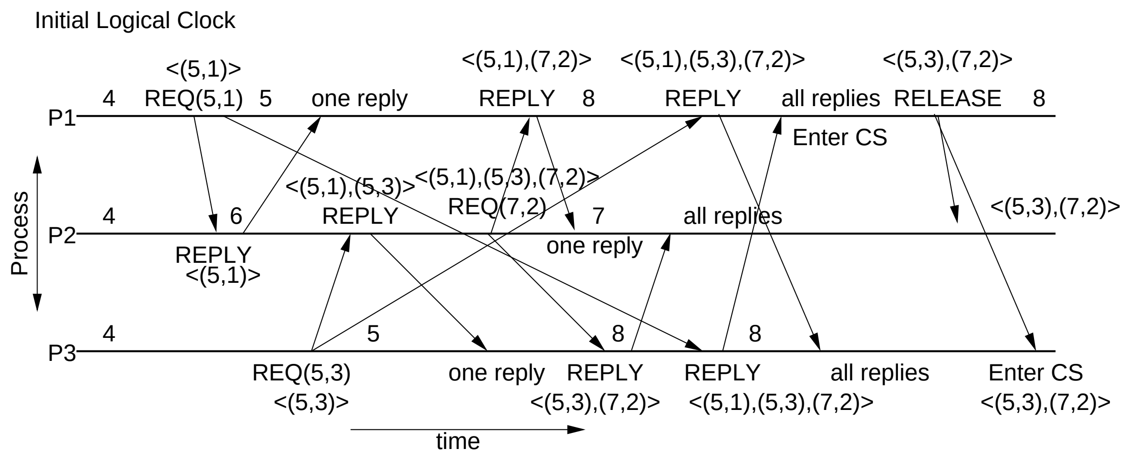
Distributed Mutual Exclusion

- *Contention-based*
  - Timestamp prioritized
  - Voting
- *Control (Token)-based*
  - Ring structure
  - Tree structure
  - Broadcast structure

Contention-based DME

- Timestamp prioritized
  - Lamport:
    Send REQUEST with Lamport Timestamp to all processes
    Wait for REPLY from all processes
    When own REQUEST at front of queue, enter CS
    When complete CS, sent RELEASE to all processes

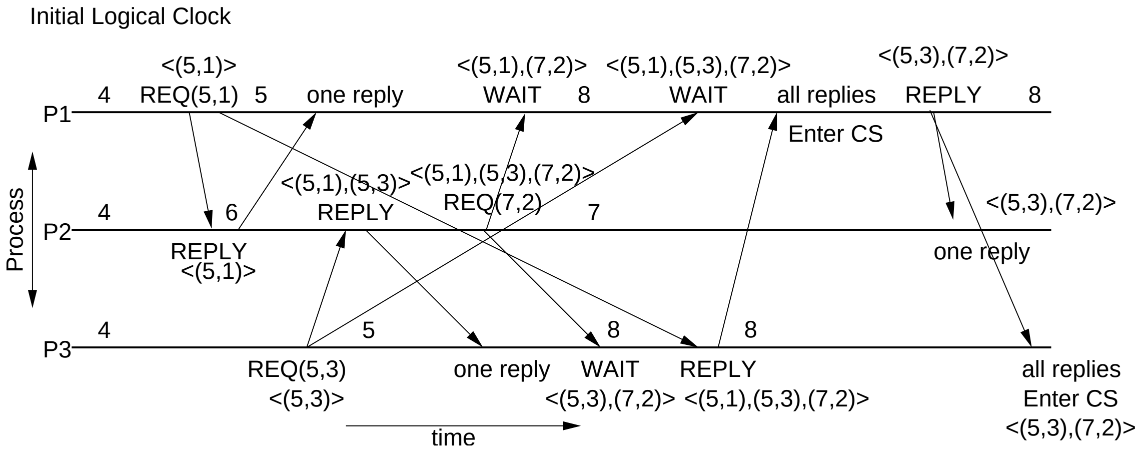
    When receive REQUEST from S, enqueue REQUEST and send REPLY to S
    When receive RELEASE from S, dequeue its REQUEST
  - $3(N-1)$ messages



Lamport Timestamp DME

Contention-based DME (cont)

- Ricart & Agrawala:
  Like Lamport, but only send REPLY if
  Not in CS, and
  REQUEST is ahead of mine (if any) in queue
- $2(N-1)$ messages

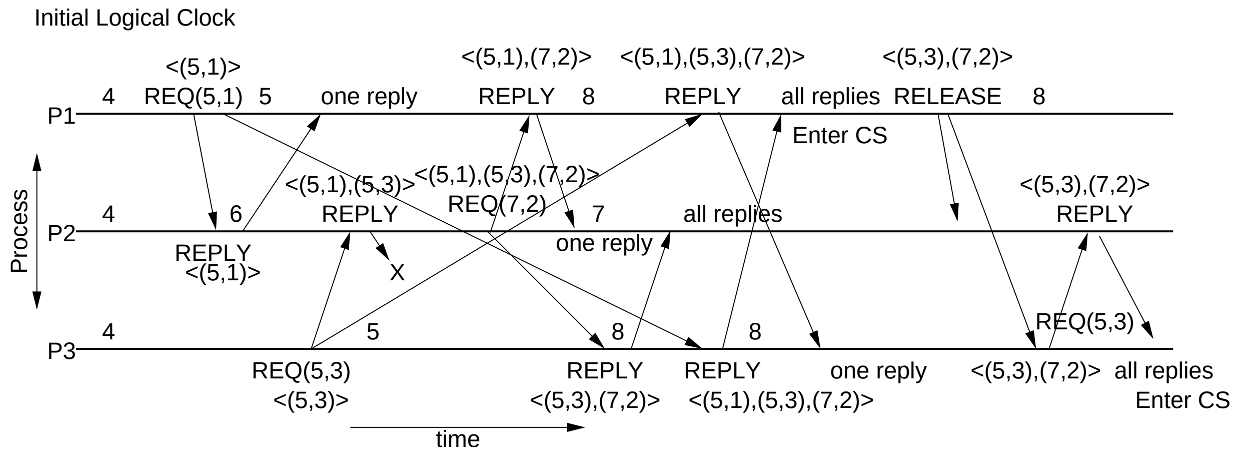


Ricart & Agrawala Timestamp DME

Contention-based DME (cont)

- Message failure:

  - Node may resend request if no reply
  - Receiving any request, node replies
  - Duplicate request just not entered into queue
  - Node may ask head of queue what is the hold up



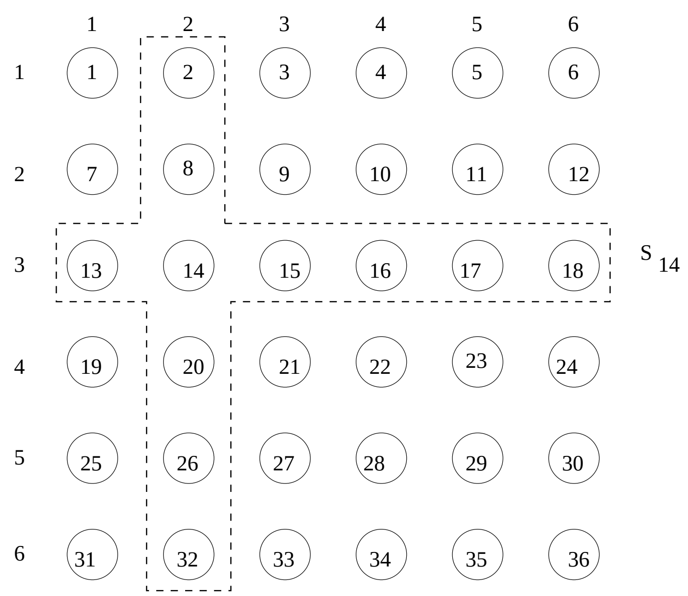
Lamport Timestamp DME with message failure

- Node failure:

  - More severe problem
  - Must detect failed node, recover

Contention-based DME (cont)

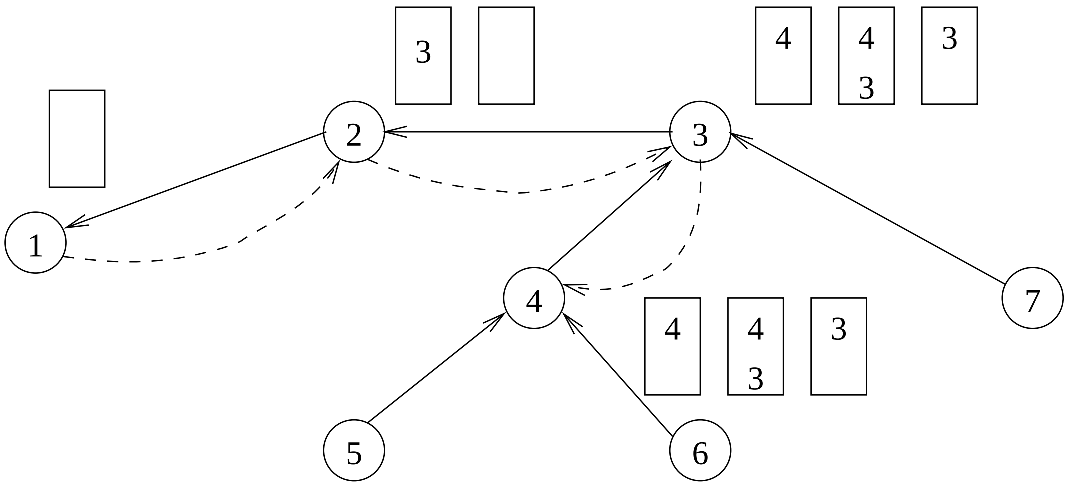
- Voting
  - Simple Majority Voting:
    Candidates solicit votes
    Voters can vote for only one candidate
    At most one gets majority
    Candidate returns votes when done with CS
    Problem: deadlock!
  - Majority Voting with Rescension:
    Voter can ask for vote back
    Candidate not in CS must return vote if asked
    Voters always vote for 'best 'candidate
  - Coterie-based Voting:
    Each process P_i has its request set S_i of voters (quorum)
    P_i must have vote from every member of S_i to enter CS
    $\forall i, j, \;\; S_i \cap S_j \neq \emptyset$



Maekawa $O(\sqrt{N})$ Quorum Method

Control (Token)-based DME

- Ring structure
  - Pass DME token around logical ring
  - Must wait for DME token to enter CS
  - Delay
  - Overhead
  - Priority scheme possible
- Tree structure (Raymond)
  - Only send messages when token is requested
  - Send messages along edges of logical tree
  - Send requests toward root (current token holder)
  - Maintain distributed queue with self, neighbors on it
  - Send token toward first request in queue when done
  - When receive token and first in queue, enter CS

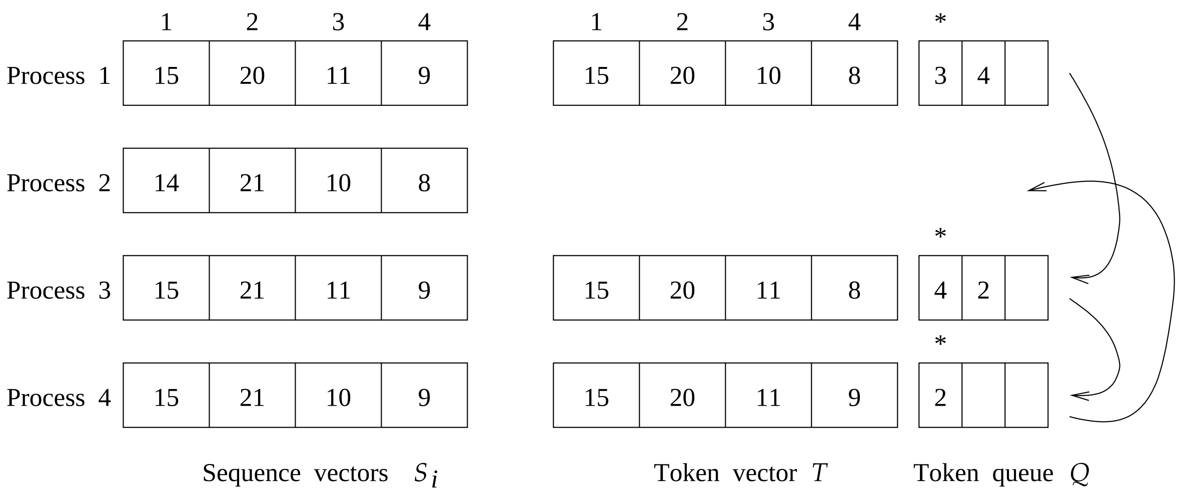


Tree Structure Distributed FIFO Queue Token Passing

Control (Token)-based DME (cont)

- Broadcast structure (Suzuki & Kasami)

  - Token has Token Vector T and Request Queue Q
  - T indicates completed CSs for each process
  - Q indicates pending requests, in request order
  - Each process maintains a sequence number for its requests
  - Broadcast REQUEST with incremented request sequence number
  - Process i maintains vector S_i of largest request sequence numbers it has seen for each process
  - Receive REQUEST, update S_i; if holding token, append to Q
  - If hold token and idle, send to first process in Q, if any
  - If receive token, update Q and enter CS

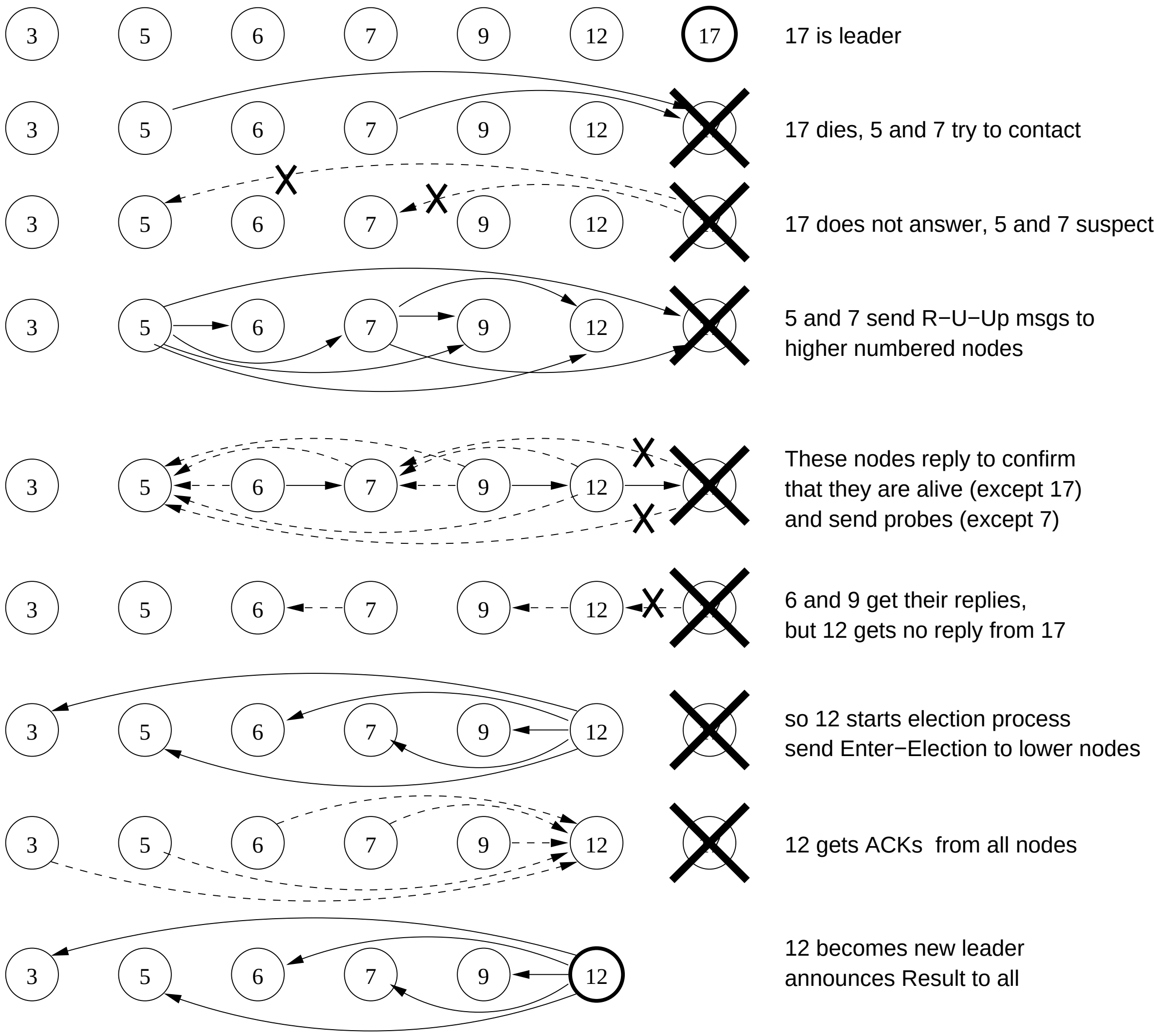


Broadcast Structure Token Passing

Leader Election

Complete topology

The Bully algorithm
1. Are-U-Up to higher numbered nodes
2. If highest alive, Enter-Election to lower nodes
3. When ACK or TRO for all lower nodes, send Result
4. Enter-Election received: transient state until Result



Leader Election (cont)

Tree topology

Distributed MST formation
Galleger, Humbelt & Spira (Sollin's Method in parallel)

Leader election
Last node to yield and merge
Timestamp protocol in tree

Logical ring topology

The initiator node sets partcipating = true and
send (id) to its successor node;

For each process node ,

```
receive (value);
case
    value > id : participating := true,    send (value);
    value < id and participating == false : participating := true,    send (id);
    value == id : announce leader;
end case
```

Logical ring topology