

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:

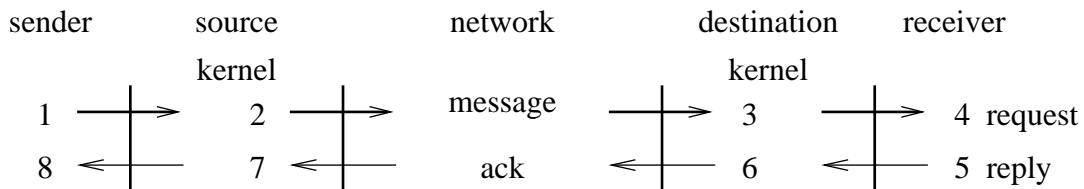
send(destination, message)

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

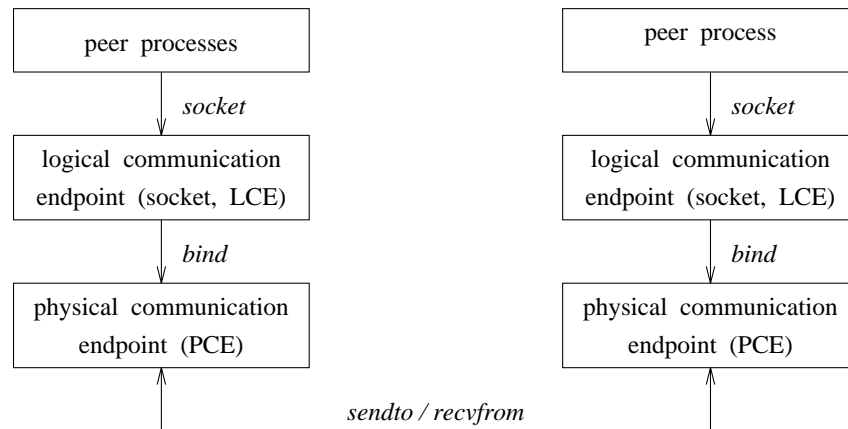


1. **Nonblocking send, $1+8$** : Sender process is released after message has been composed and copied into sender's kernel.
2. **Blocking send, $1+2+7+8$** : Sender process is released after message has been transmitted to the network.
3. **Reliable blocking send, $1+2+3+6+7+8$** : Sender process is released after message has been received by the receiver's kernel.
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.
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.

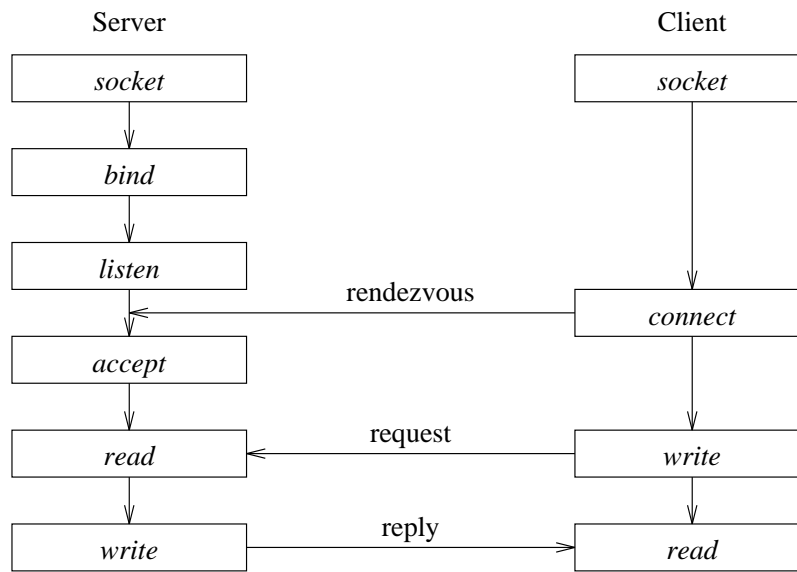
Message passing API

- *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
- *Socket*: A logical communication endpoint for communication between autonomous domains

Connectionless socket communication

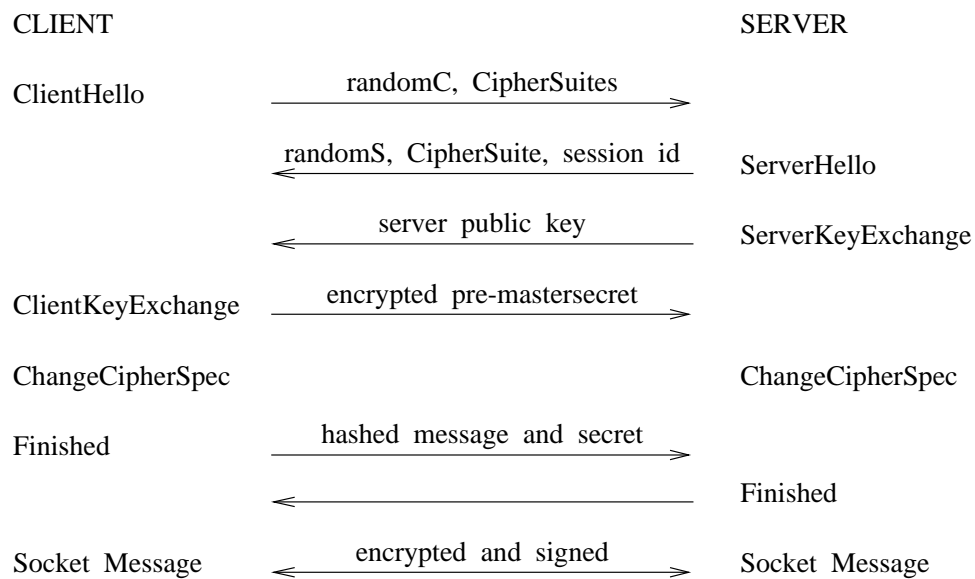


Connection-oriented socket communication



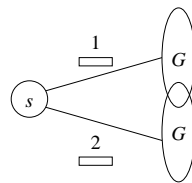
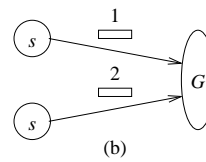
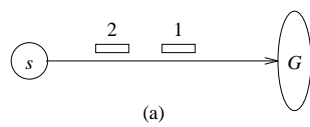
Secure Socket Layer protocol

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

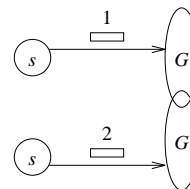


Group communication and multicast

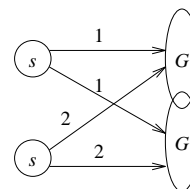
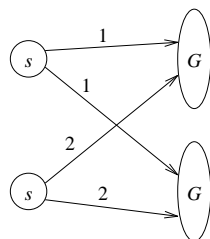
- *Best effort*
- *All or none*
- *Orderly delivery*
 - FIFO
 - Causal order
 - Total order



(c)



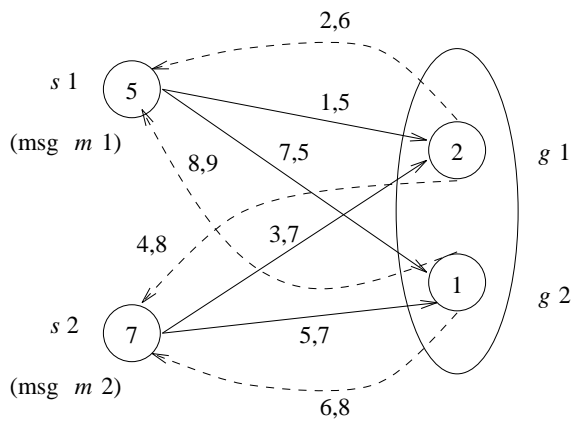
(d)



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 $T_i \leq S_i$.

Total order

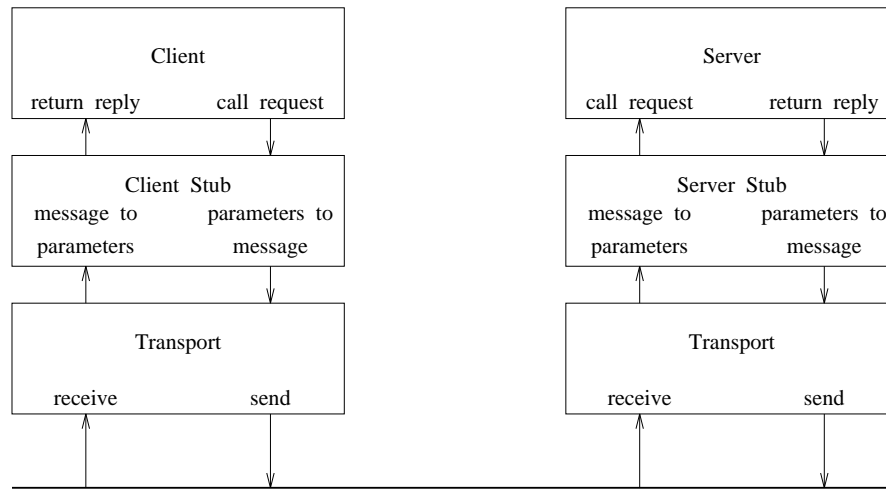


Multicast Message	Acknowledge Time	Commit Time
m_0	2	delivered
m_1	6	9
m_2	8	8
m_3	10	pending

Buffer management in the communication handler

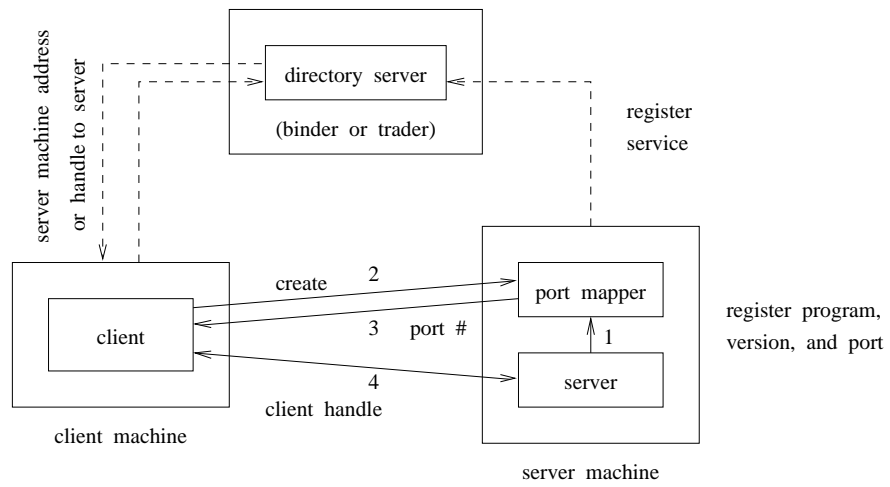
Request/reply communication

Remote Procedure Calls (RPCs)

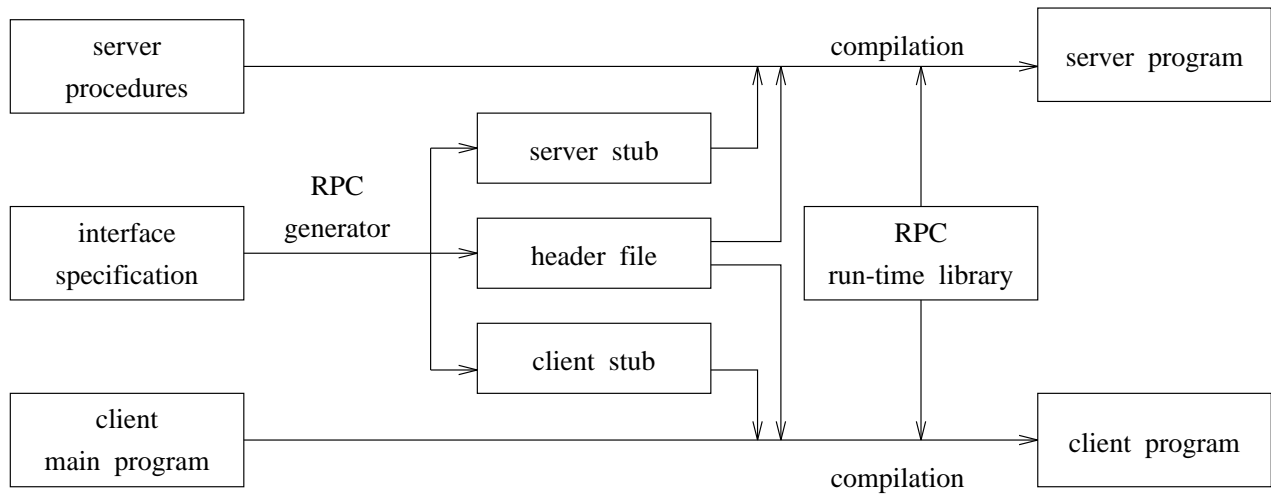


- *Parameter passing and data conversion*
- *Binding*
- *Compilation*
- *Exception and failure handling*
- *Security*

RPC Binding



RPC compilation



RPC exception and failure

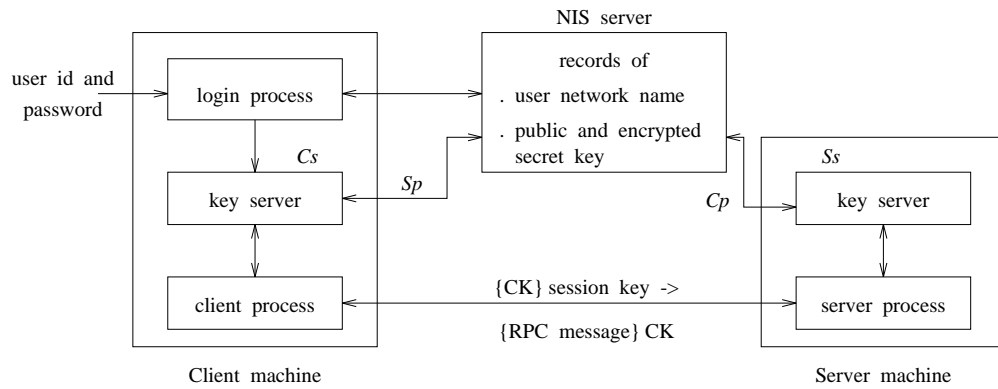
- *Exception*: in-band or out-band signaling
- *Link failure*: retransmission, sequence number and idempotent requests, use of transaction id *xid*
- *Server crash*:
 - *at least once*: server raises an exception and client retries
 - *at most once*: server raises an exception and client gives up
 - *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.
- $C_p = \alpha^{C_s} \bmod M$, and $S_p = \alpha^{S_s} \bmod M$, where α and M are known constants.

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

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

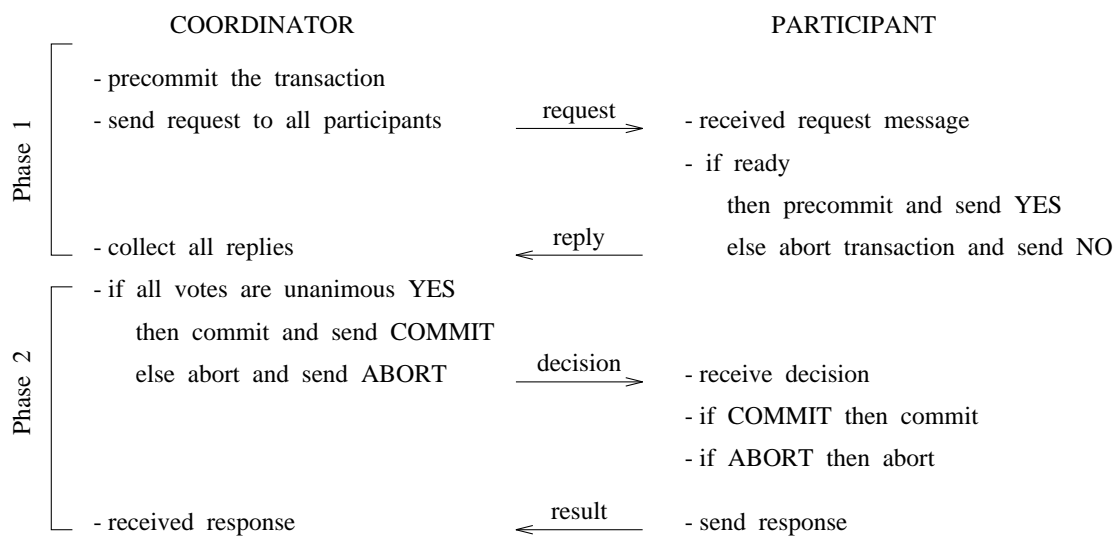


Transaction Communication

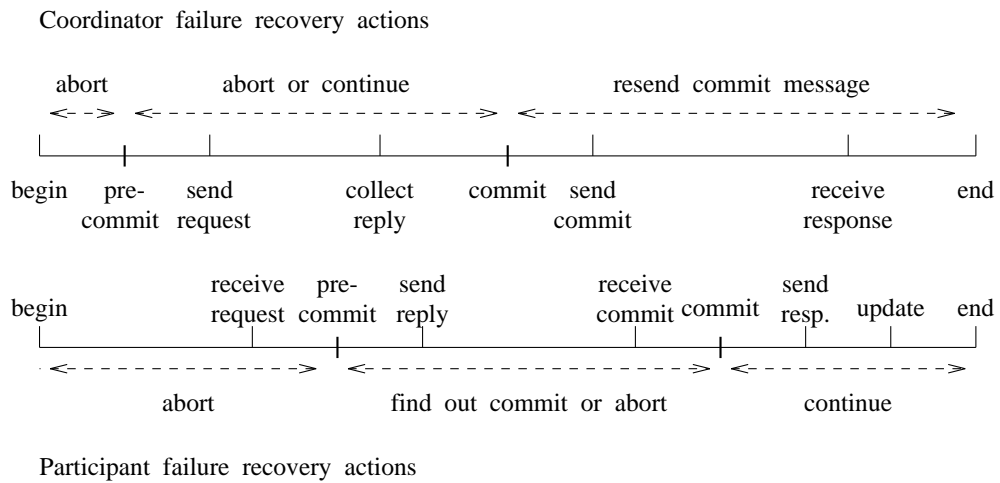
ACID properties

- *Atomicity*
- *Consistency*
- *Isolation*
- *Durability*

Two-phase commit protocol



Failure and recovery of the 2PC protocol

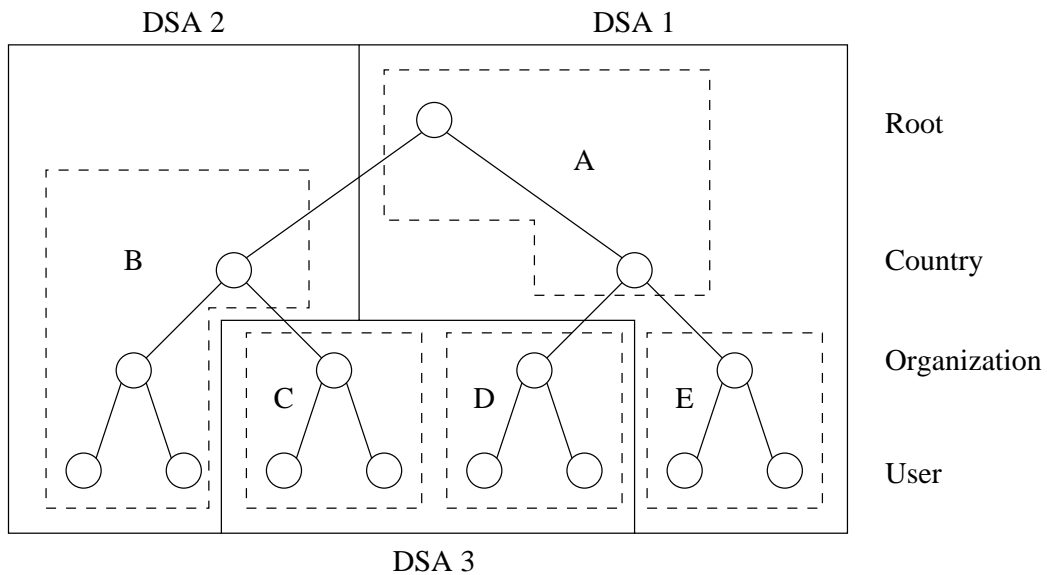


Name and Directory Services

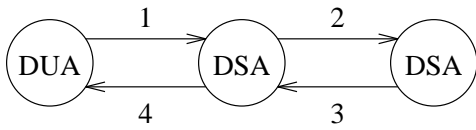
Object attributes and name structures

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

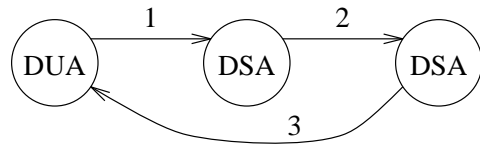
Name space and information base



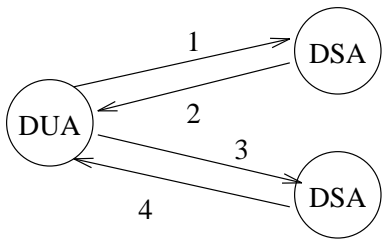
Name resolution



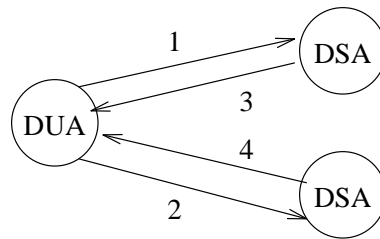
Recursive chaining



Transitive chaining



Referral

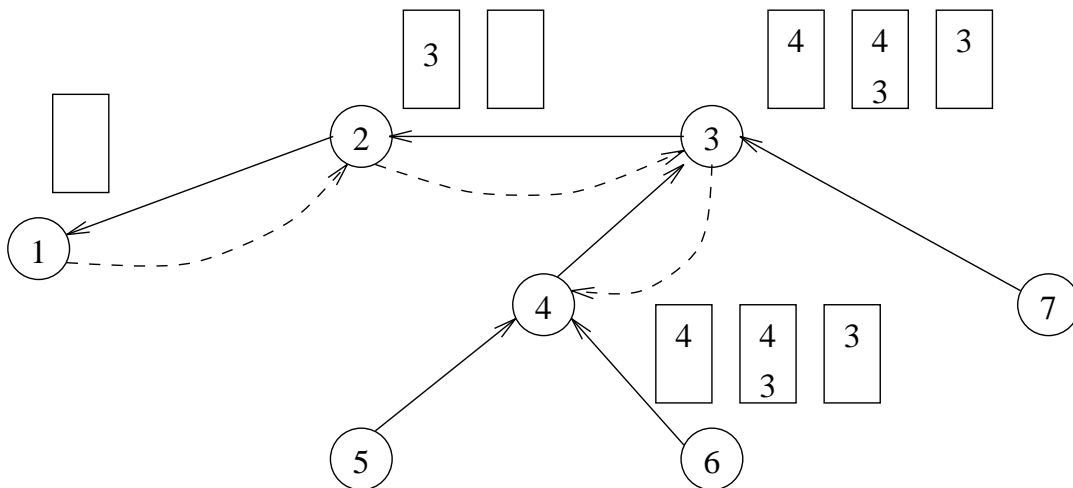


Multicast

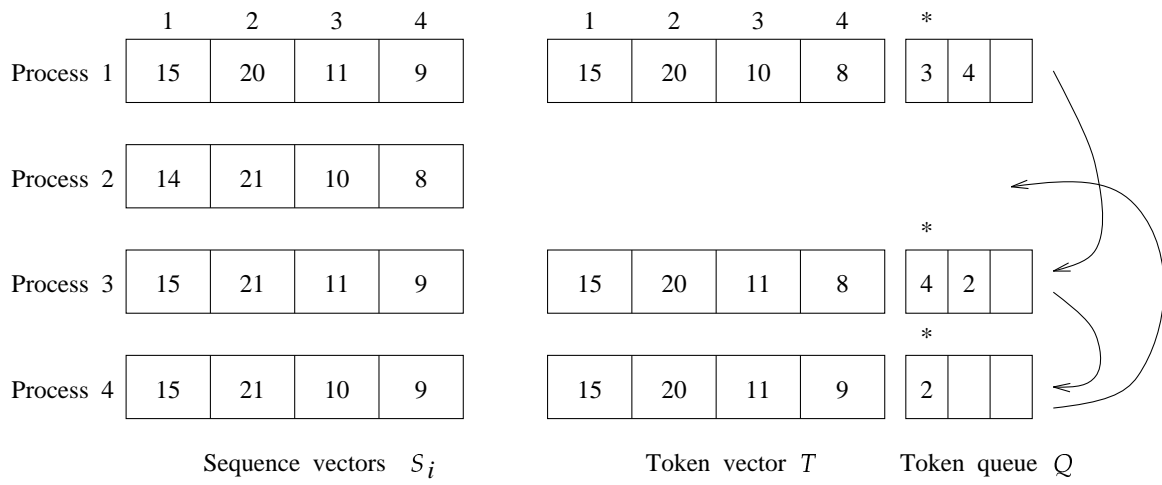
Distributed Mutual Exclusion

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

Tree-structure token passing



Broadcast structure token passing



Leader Election

Complete topology

The Bully algorithm

Logic ring topology

The initiator node sets participating = 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

Tree topology