# CAS CS 552
# Intro to Operating Systems

Richard West

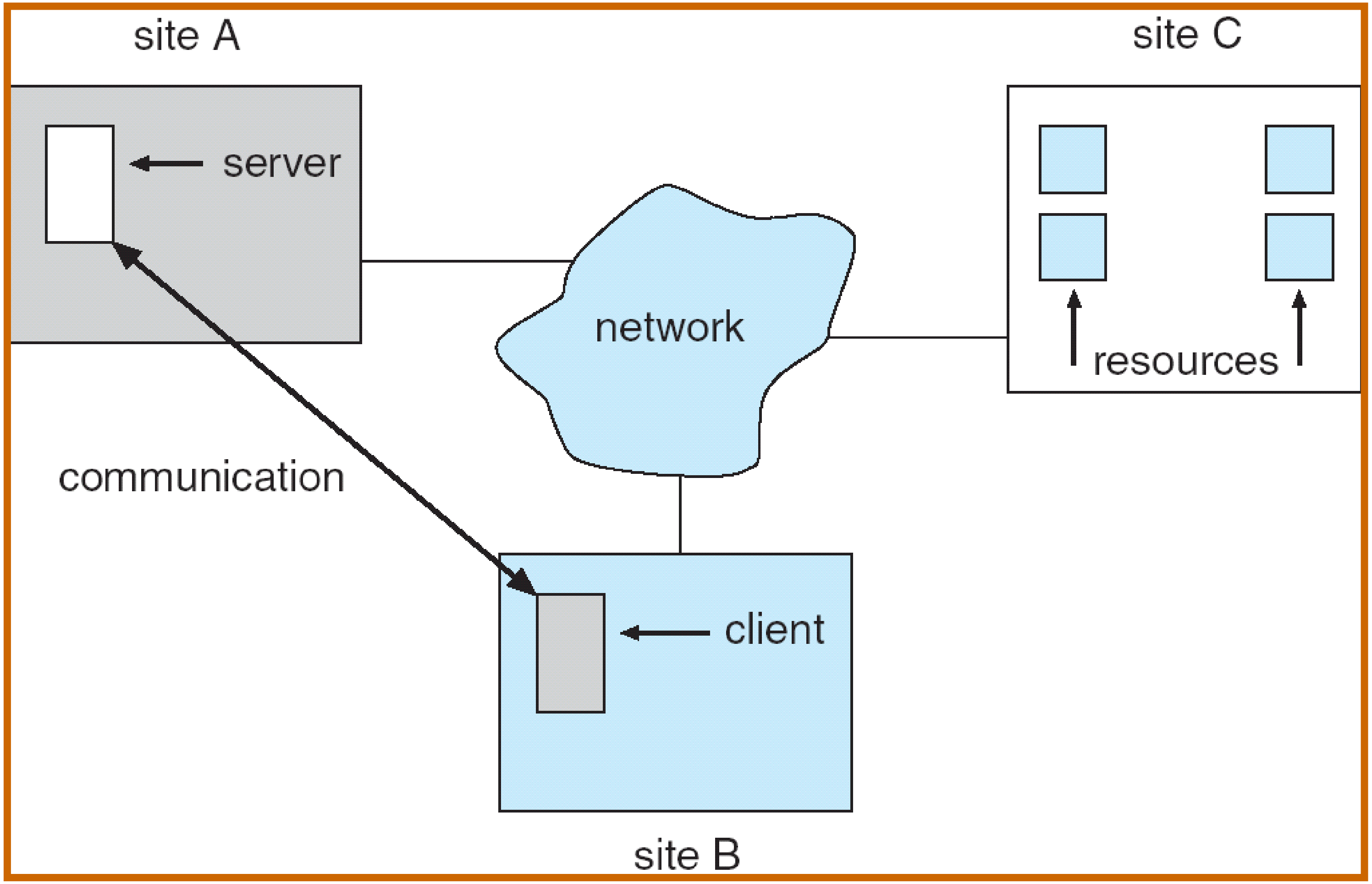
Distributed Systems Topics

# Distributed System

- **Distributed system** is a collection of loosely coupled processors interconnected by a communications network
  - No physically common clock or physically shared memory
  - More tightly coupled parallel systems have physically shared memory
- Processors variously called *nodes, computers, machines, hosts*

- Motivation:
  - Resource sharing
  - Computation speedup – **load sharing**
  - Reliability – detect and recover from site failure, function transfer, reintegrate failed site
  - Communication – message passing

# A Distributed System



# Distributed OS Types

- Network Operating Systems
- Distributed Operating Systems

# Network-Operating Systems

- Users are aware of multiplicity of machines.  Access to resources of various machines is done explicitly by:
  - Remote logging into the appropriate remote machine (telnet, ssh)
  - Remote Desktop (Microsoft Windows)
  - Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism

# Distributed-Operating Systems

- Users not aware of multiplicity of machines
  - Access to remote resources similar to access to local resources
  - **Transparency**

- Process migration for:
  - Load balancing
  - Computational speedup
  - Data access at remote site
  - Hardware/software specific reasons
- File/data sharing/caching

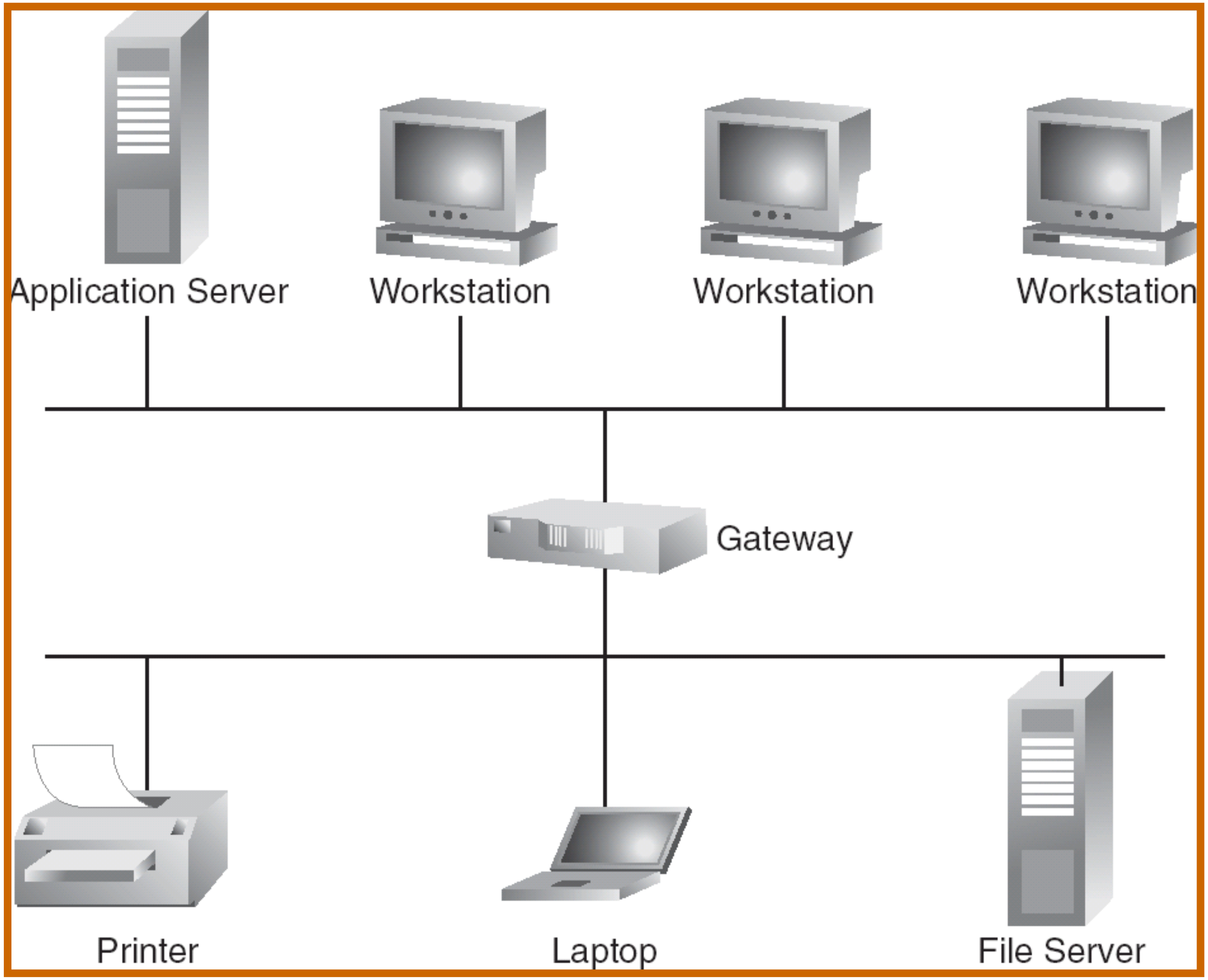
# Distributed-Operating Systems (Cont.)

- Process Migration – execute an entire process, or parts of it, at different sites
  - Load balancing – distribute processes across network to even the workload
  - Computation speedup – subprocesses can run concurrently on different sites
  - Hardware preference – process execution may require specialized processor
  - Software preference – required software may be available at only a particular site
  - Data access – run process remotely, rather than transfer all data locally

# Network Structure

- Local-Area Network (LAN) – designed to cover small geographical area.
  - Multiaccess bus, ring, or star network
  - Broadcast is fast and cheap
  - Nodes:
    - usually workstations and/or personal computers
    - possibly a few mainframes

# Depiction of typical LAN



# Network Types (Cont.)

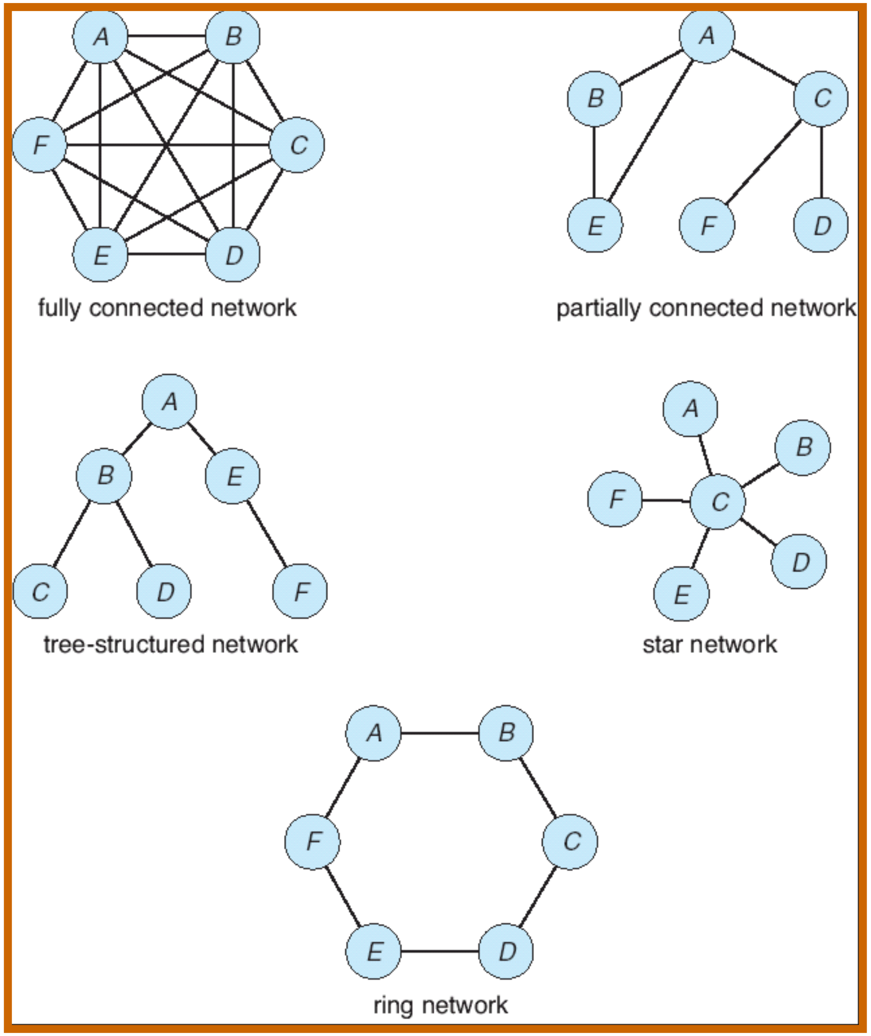
- Wide-Area Network (WAN) – links geographically separated sites
  - Point-to-point connections over long-haul lines (often leased from a phone company)
  - Broadcast usually requires multiple messages
  - Nodes:
    - usually a high percentage of mainframes

# Network Topology

- Nodes in the system can be physically connected in a variety of ways, based on factors such as:

  - **Basic cost** - How expensive is it to link the various sites in the system?
  - **Communication cost** - How long does it take to send a message from site *A* to site *B*?
  - **Reliability** - If a link or a site in the system fails, can the remaining sites still communicate with each other?

# Network Topology



# Communication Issues

- **Naming and name resolution** - How do two processes locate each other to communicate?
- **Routing strategies** - How are messages sent through the network?
- **Connection strategies** - How do two processes send a sequence of messages?
- **Contention -** The network is a shared resource, so how do we resolve conflicting demands for its use?

# Design Issues

- **Transparency** – the distributed system should appear as a conventional, centralized system to the user

- **Fault tolerance** – the distributed system should continue to function in the face of failure

- **Scalability** – as demands increase, the system should easily accept the addition of new resources to accommodate the increased demand

# Distributed Coordination / Synchronization

- How did we address synchronization (to enforce mutual exclusion) on a single-machine system?

- What about issues with distributed systems?
  - No physically common clock or shared memory

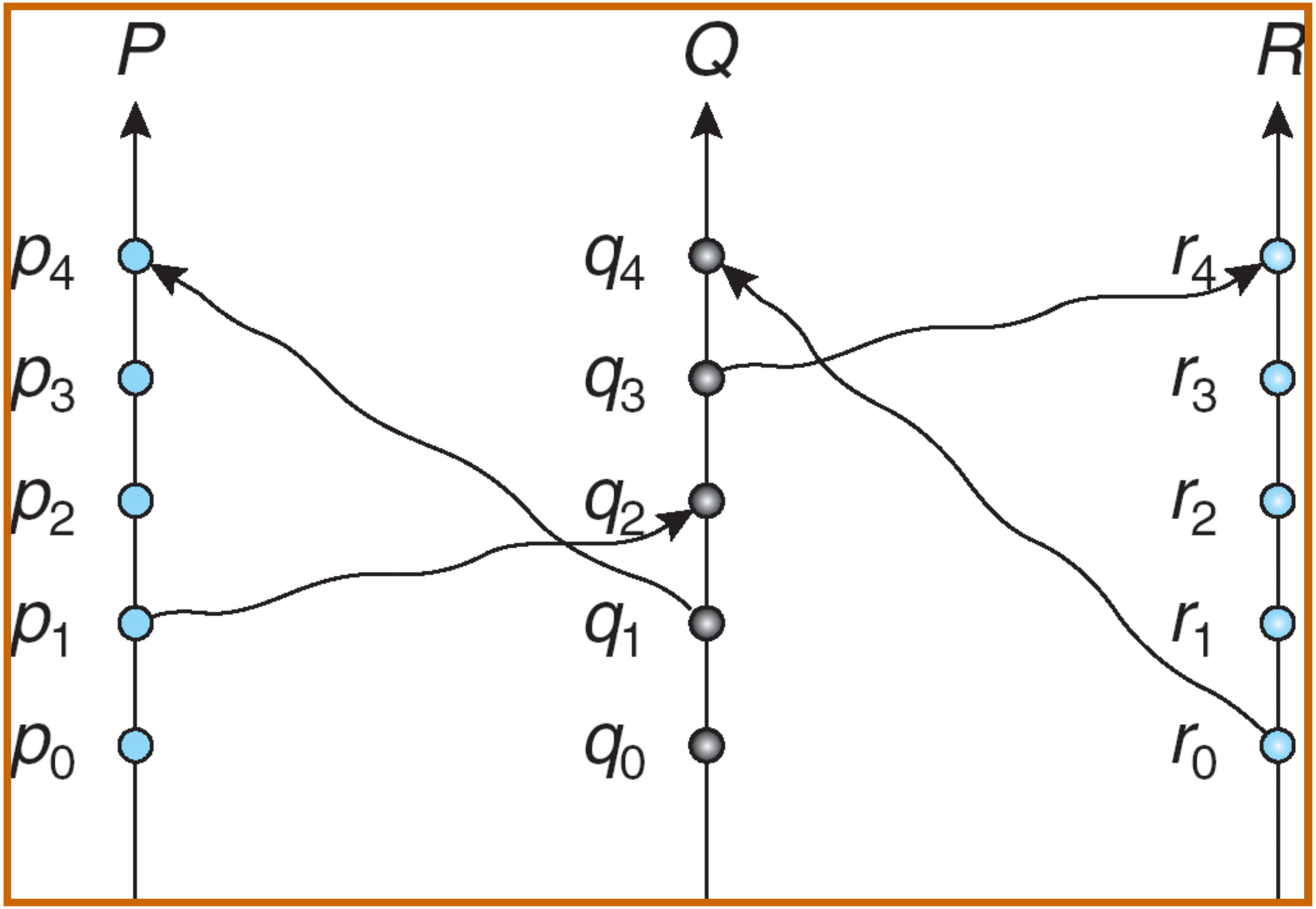
  - How do we guarantee global ordering (do we care about absolute global time ordering)?
    - Think: Einstein's special theory of relativity

# Event Ordering

- Only really care about ordering of events not when those events happened exactly in time

- *Happened-before* relation (denoted by $\rightarrow$)
  - If $A$ and $B$ are events in the same process, and $A$ was executed before $B$, then $A \rightarrow B$
  - If $A$ is the event of sending a message by one process and $B$ is the event of receiving that message by another process, then $A \rightarrow B$
  - If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$ *(transitivity)*
  - If neither $A \rightarrow B$ nor $B \rightarrow A$, A and B are *concurrent* events (i.e., $A \neq B$)

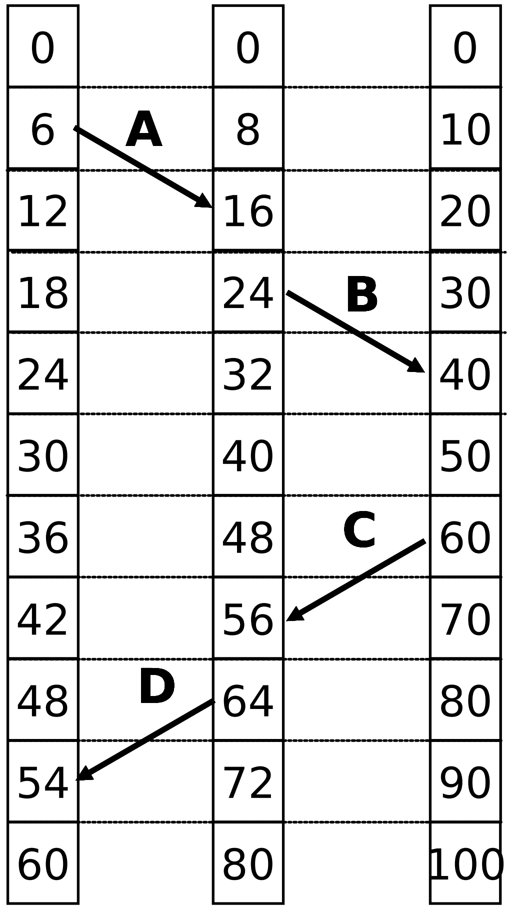
# Relative Time for Three Concurrent Processes



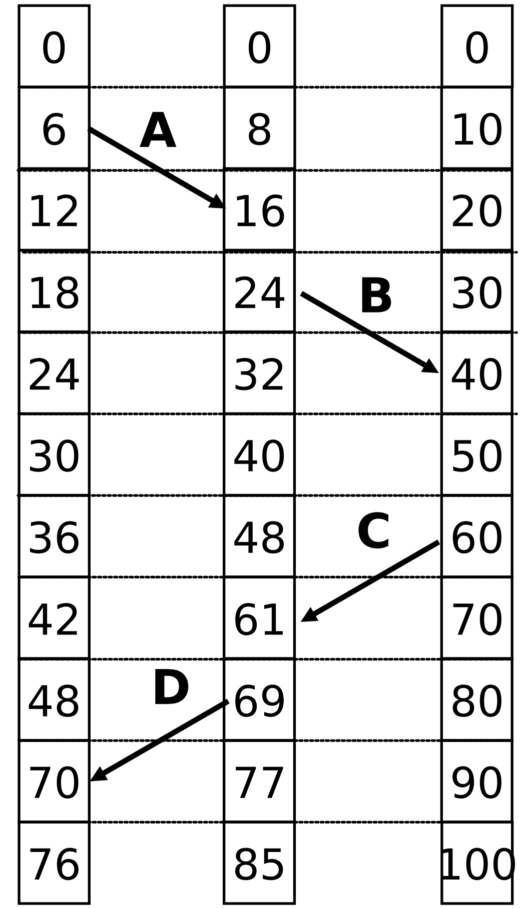
# Lamport Clocks

- Associate a timestamp with each system event
  - Require that for every pair of events A and B, if A $\rightarrow$ B, then the timestamp of A is less than the timestamp of B
- Within each process Pi maintain a **logical clock**, LCi
  - The logical clock can be implemented as a simple counter that is incremented between any two successive events executed within a process
    - LCi is **monotonically increasing**
- A process advances its logical clock when it receives a message whose timestamp is greater than the current value of its logical clock
- If the timestamps of two events A and B are the same, then the events are concurrent
  - We may use the process identity numbers to break ties and to create a total ordering

# Example




(a) 3 processes, each with own clock




(b) Lamport’s algorithm corrects clocks

# Distributed Mutual Exclusion (DME)

- Assumptions
  - The system consists of $n$ processes; each process $P_i$ resides at a different processor
  - Each process has a critical section that requires mutual exclusion

- Requirement
  - If $P_i$ is executing in its critical section, then no other process $P_j$ is executing in its critical section

# DME: Centralized Approach

- One of the processes in the system is the *coordinator*
- A process that wants to enter its critical section sends a request message to the coordinator
- The coordinator decides which process can enter the critical section next, and its sends that process a reply message
- When the process receives a reply (or grant) message from the coordinator, it enters its critical section
- After exiting its critical section, the process sends a release message to the coordinator and proceeds with its execution
- This scheme requires three messages per critical-section entry:
  - request
  - reply / grant
  - release

# DME: Fully Distributed Approach

- When process $P_i$ wants to enter its critical section, it generates a new timestamp, *TS*, and sends the message *request* ($P_i$,*TS*) to all other processes in the system
- When process $P_j$ receives a *request* message, it may reply immediately or it may defer sending a reply back
- When process $P_i$ receives a *reply* message from all other processes in the system, it can enter its critical section
- After exiting its critical section, the process sends *reply* messages to all its deferred requests

# DME: Fully Distributed Approach

- The decision whether process $P_j$ replies immediately to a *request*($P_i$, *TS*) message or defers its reply is based on three factors:
  - If $P_j$ is in its critical section, then it defers its reply to $P_i$
  - If $P_j$ does *not* want to enter its critical section, then it sends a *reply* immediately to $P_i$
  - If $P_j$ wants to enter its critical section but has not yet entered it, then it compares its own request timestamp with the timestamp *TS*
    - If its own request timestamp is greater than *TS*, then it sends a *reply* immediately to $P_i$ ($P_i$ asked first)
    - Otherwise, the reply is deferred

# DME: Fully Distributed Approach

- Freedom from Deadlock is ensured
- Freedom from starvation is ensured, since entry to the critical section is scheduled according to the timestamp ordering
  - The timestamp ordering ensures that processes are served in a first-come, first served order
- The number of messages per critical-section entry is

  $$2 \times (n - 1)$$

  This is the minimum number of required messages per critical-section entry when processes act independently and concurrently

# Undesirable Consequences

- The processes need to know the identity of all other processes in the system, which makes the dynamic addition and removal of processes more complex

- If one of the processes fails, then the entire scheme collapses
  - This can be dealt with by continuously monitoring the state of all the processes in the system

# Token-Passing Approach

- Circulate a token among processes in system
  - **Token** is special type of message
  - Possession of token entitles holder to enter critical section
- Processes *logically* organized in a **ring structure**
- Unidirectional ring guarantees freedom from starvation
- Two types of failures
  - Lost token – election must be called
  - Failed processes – new logical ring established

# Group Communication (Atomicity)

- Either all the operations associated with a program unit are executed to completion, or none are performed

- Ensuring **atomicity** in a distributed system requires a **transaction coordinator**, which is responsible for the following:
  - Starting the execution of the transaction
  - Breaking the transaction into a number of subtransactions, and distribution these subtransactions to the appropriate sites for execution
  - Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites