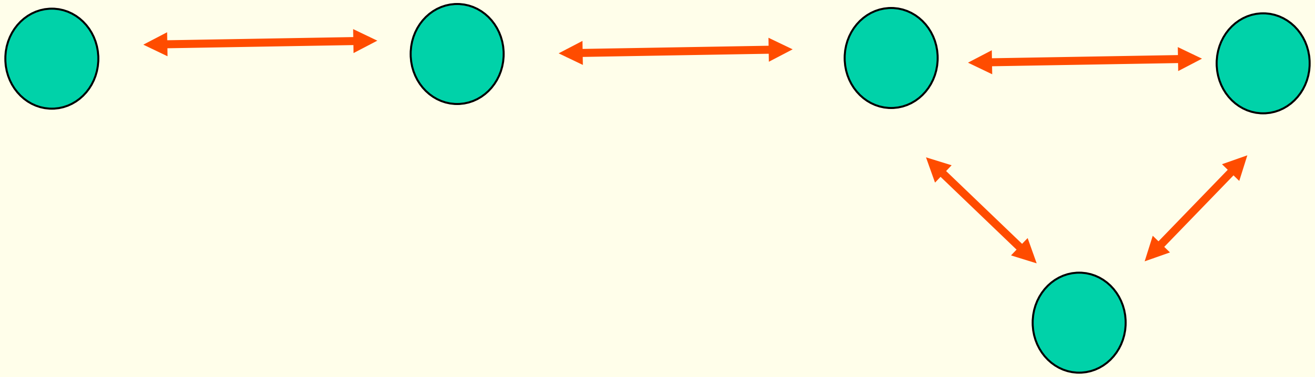
# Concurrent & Distributed Systems

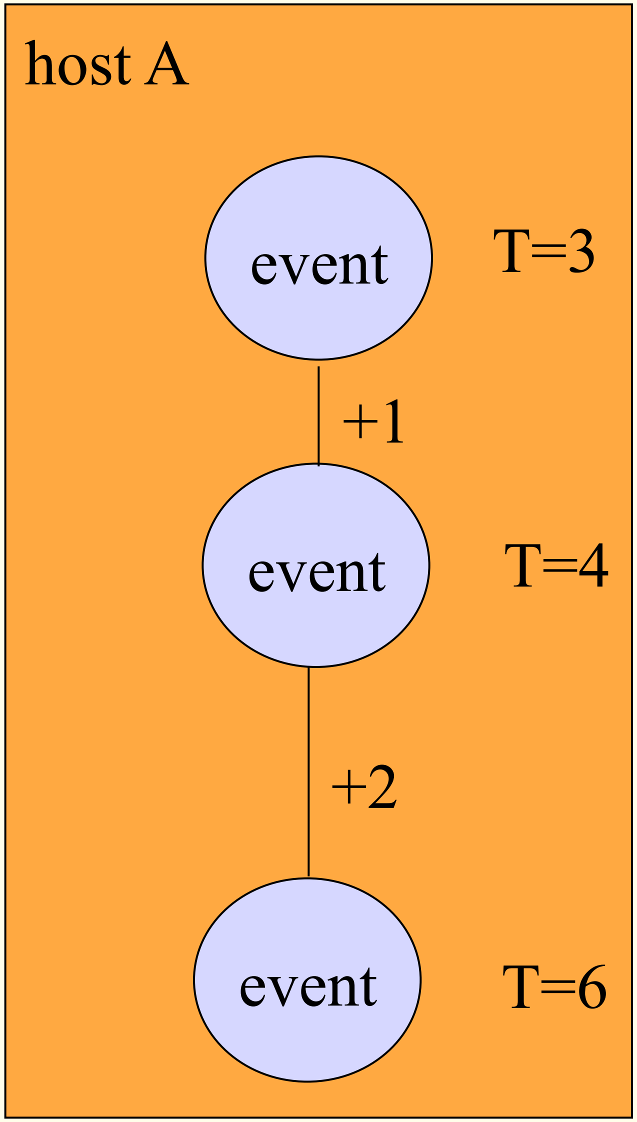
## distributed co-ordination 1

# distributed coordination

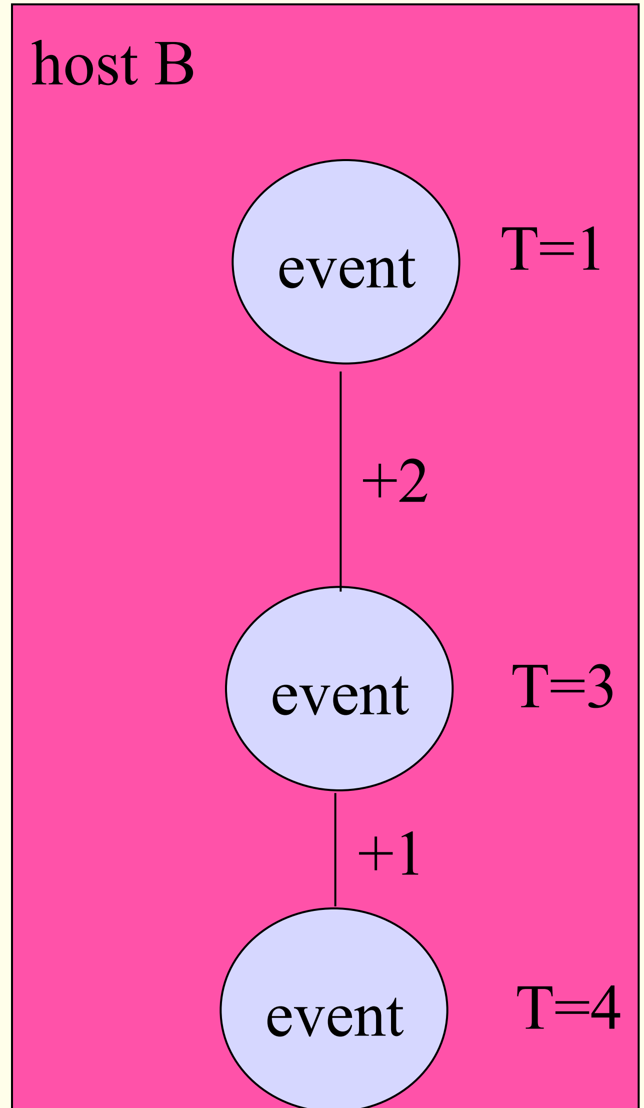
- event ordering
- mutual exclusion
- deadlocks
- election algorithms



# event ordering: partial Vs global




**How can we ensure we have a *global* ordering ?**



# event ordering

- ***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 $A \rightarrow C$.**
- **how can we ensure this is maintained across a number of hosts?**

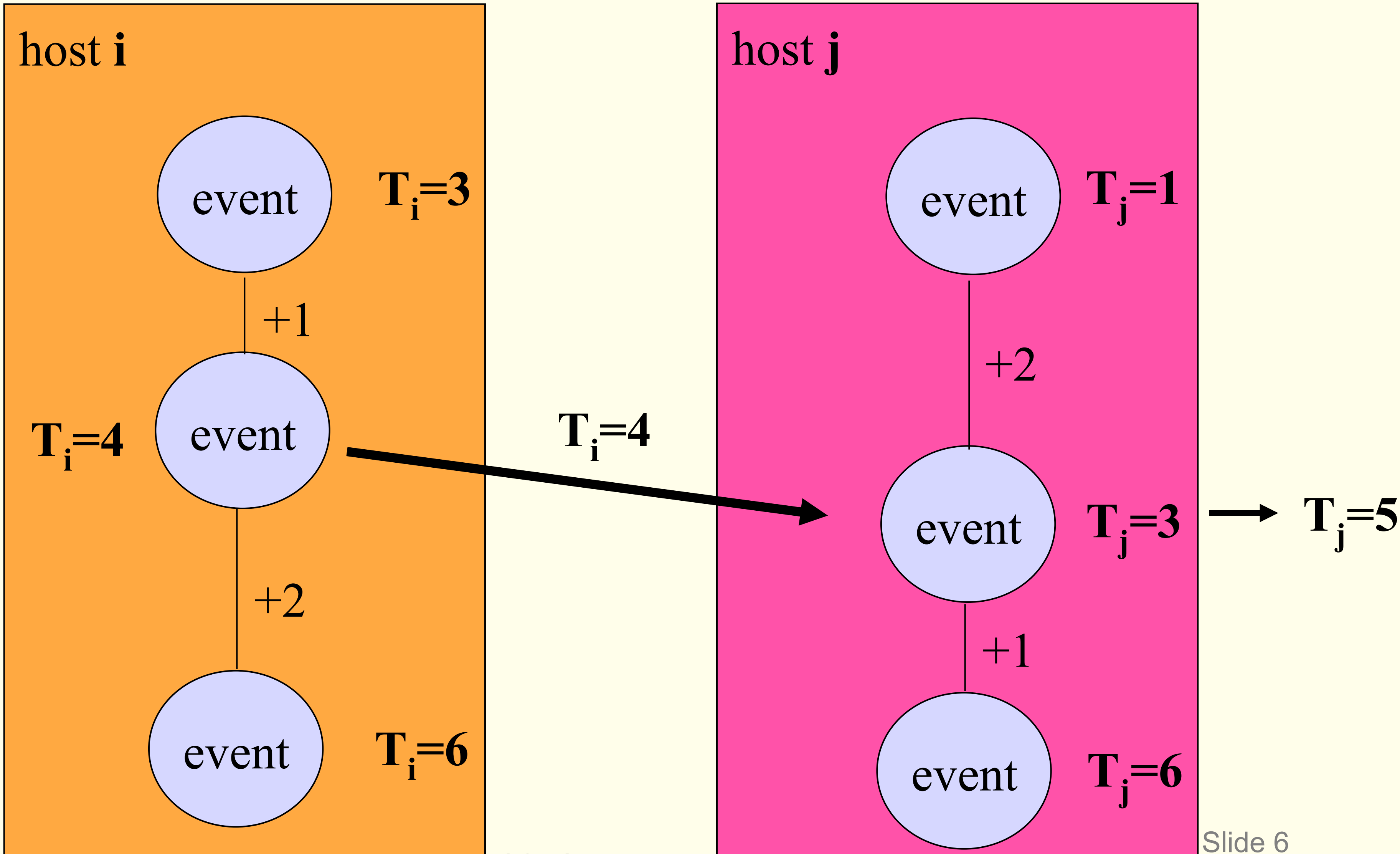
# implementation of $\rightarrow$

- **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 $P_i$ a *logical clock*, $T_i$ is associated. The logical clock can be implemented as a simple counter that is incremented between any two successive events executed within a process.**

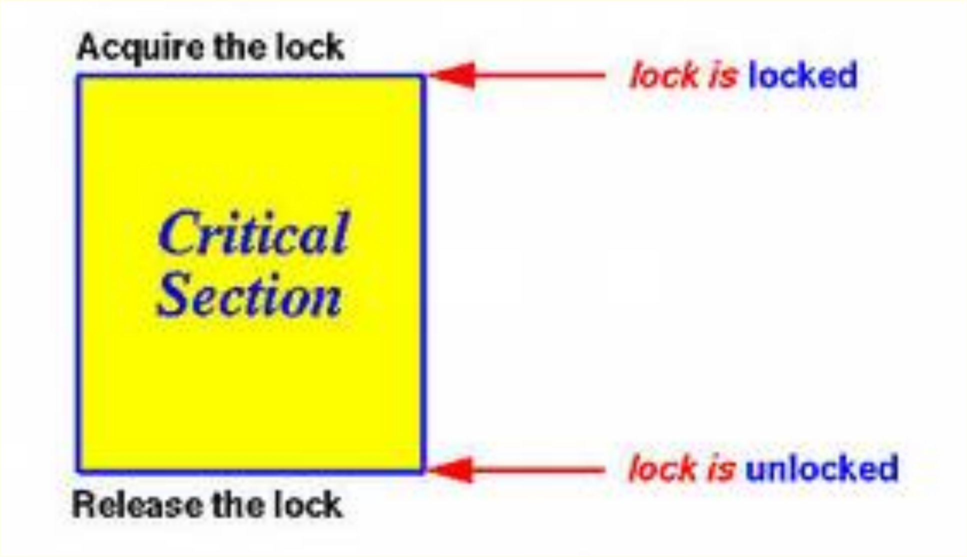
- **A process advances its logical clock when it receives a message whose timestamp is *greater or equal* than the current value of its logical clock. ( Must make it *less*; new_time = received_time + 1 )**
  - **we need to do this even if the times are equal as $A \rightarrow B$, means the receiver's time must be *more.***

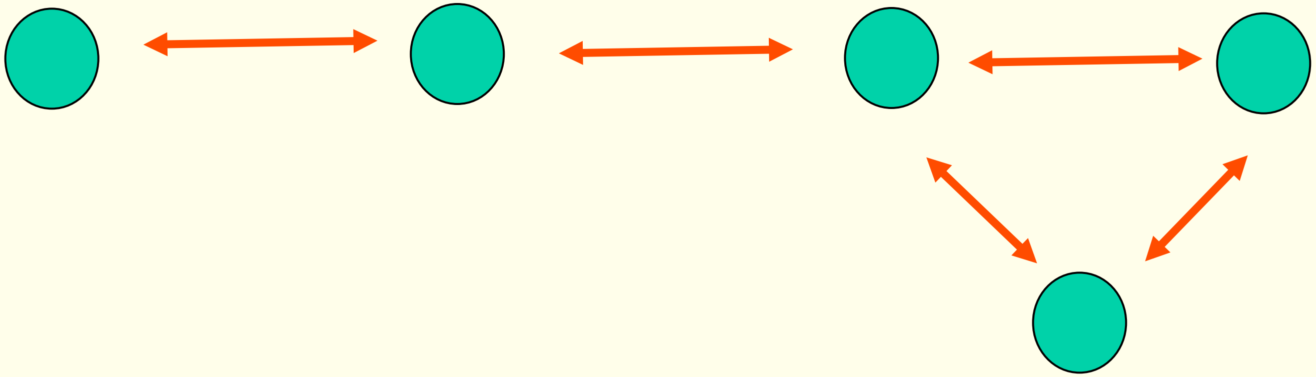
# event ordering: implementation



# distributed coordination

- event ordering
- mutual exclusion
- deadlocks
- election algorithms



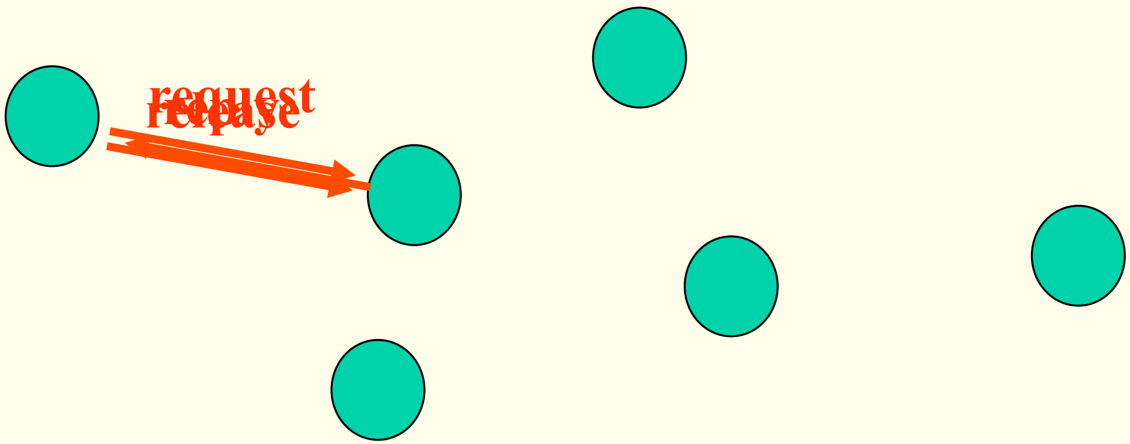
# Distributed Mutual Exclusion (DME)

- **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.**
- **We present two algorithms to ensure the mutual exclusion execution of processes in their critical sections.**

# DME: centralised approach

- **One of the processes in the system is chosen to coordinate the entry to the critical section.**
- **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. (It may have a number of requests queued.)**
- **When the process receives a *reply* 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**
  - **release**

# DME: centralised approach




- **This scheme requires three messages per critical-section entry:**
  - **request**
  - **reply**
  - **release**

# DME: Fully Distributed Approach

- **When process $P_i$ wants to enter its critical section, it generates a new timestamp, $T_i$, and sends the message *request* ( $P_i$, $T_i$ ) to all other processes in the system.**

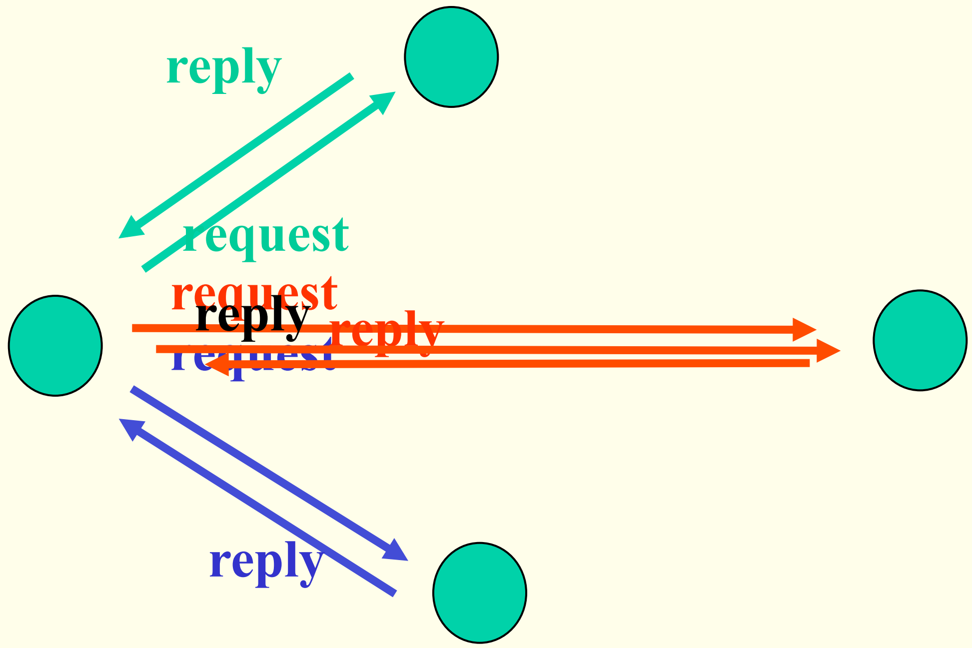
- **When process $P_j$ receives a *request* ( $P_i$, $T_i$ ) message, it may reply immediately or it may defer sending a reply back.**

  **(More on a later slide)**

- **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




- **After exiting its critical section, the process sends *reply* messages to all its deferred requests.**

# DME: fully distributed approach (continued)

- **The decision whether process $P_j$ replies immediately to a *request* ($P_i$ , $T_i$) 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 $P_j$ *does* want to enter its critical section but has not yet entered it, then it compares its own request timestamp ($T_j$ ) with the timestamp $T_i$**
    - **If its own request timestamp ($T_j$ ) is greater than $T_i$ , then it sends a *reply* immediately to $P_i$ ($P_i$ asked first).**
    - **Otherwise, the reply is deferred.**

# desirable behaviour of 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**

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

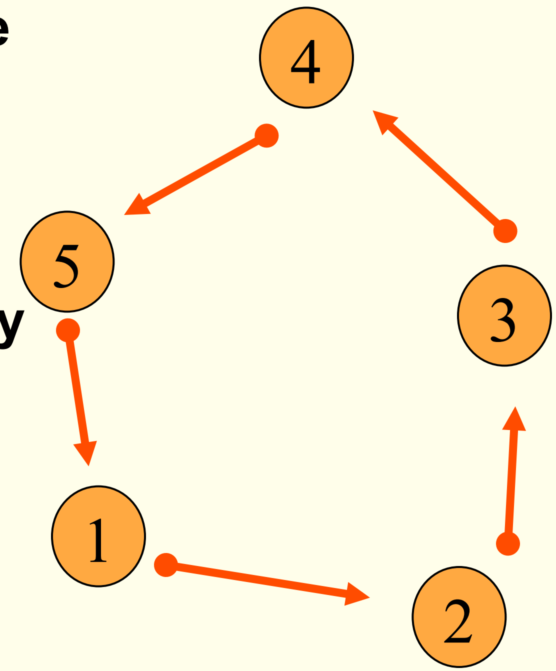
**This is the minimum number of required messages per critical-section entry when processes act independently and concurrently.**

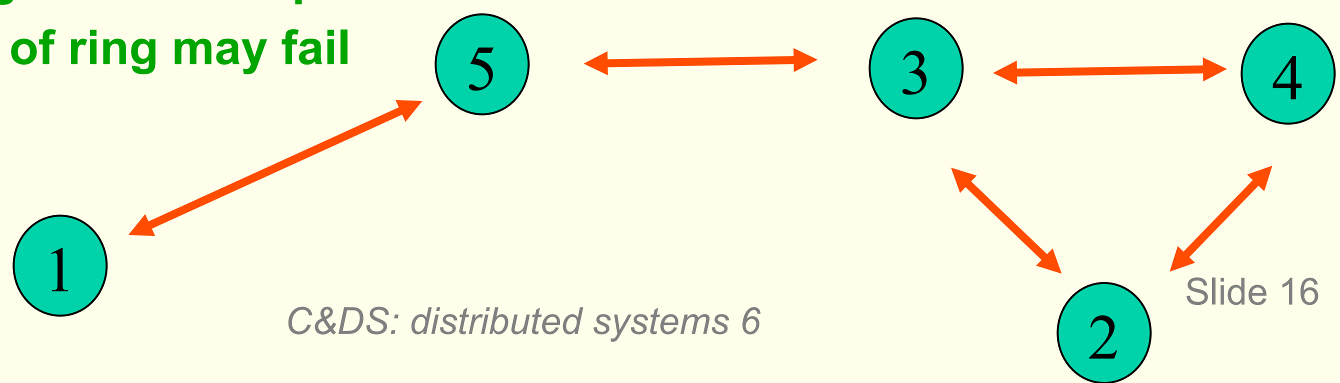
# three undesirable consequences

- 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.
- Processes that have not entered their critical section must pause frequently to assure other processes that they intend to enter the critical section.
- This protocol is therefore suited for small, stable sets of cooperating processes.

# token passing

- **host can only enter a critical region if it has the token**
- **once finished, token is released to next host**
- **forms a logical ring**
- **if not required - token is passed on immediately**
- **again,**
  - **fair, no starvation**
  - **works**
  - **no deadlock**
- **problems**
  - **token may get lost or duplicated**
  - **component of ring may fail**



# Next laboratory (lab 3)

- this laboratory will investigate the *centralised approach* to mutual exclusion (DME)
- we are going to share access to a resource (such as a file) in a way that is synchronised, i.e. *safe,* but also *fair*
- communications using RMI
- you are going to write 2 programs
  - one will act as the central controller or co-ordinator
    - 2 methods, *request*, & *release*
    - requests are queued (FIFO) !
  - the other will act as a client(s) accessing the shared file.
    - can *append* id and timestamp to file
    - implement *reply* method
- *Please* design first!