



Operating System Design

Dr. Jerry Shiao, Silicon Valley University

Distributed Coordination

- Overview
- Apply Centralized Synchronization mechanisms to a Distributed Environment.
- In Distributed Environment:
 - Event Ordering
 - Mutual Exclusion
 - Atomicity
 - Concurrency Control
 - Deadlock Handling
 - Schemes for handling Deadlock Prevention, Deadlock Avoidance, and Deadlock Detection.
 - Election Algorithms
 - Reaching Agreement

Distributed Coordination

- Event Ordering
- Centralized System: Order two events occur controlled through single common memory and clock.
 - Resource Allocation: Resource used after resource is granted.
- Distributed System: No common memory or clock.
 - Which two events occurred first?
- Happened-Before Relation: Partial Ordering of events in Distributed System.
 - Partial Ordering need to be extended to a consistent Total Ordering of all events in the system.
 - In Sequential Process: All events executing in single process is totally ordered. Based on Law of Causality.
 - Law of Causality: A message can be received ONLY after it has been sent.

Distributed Coordination

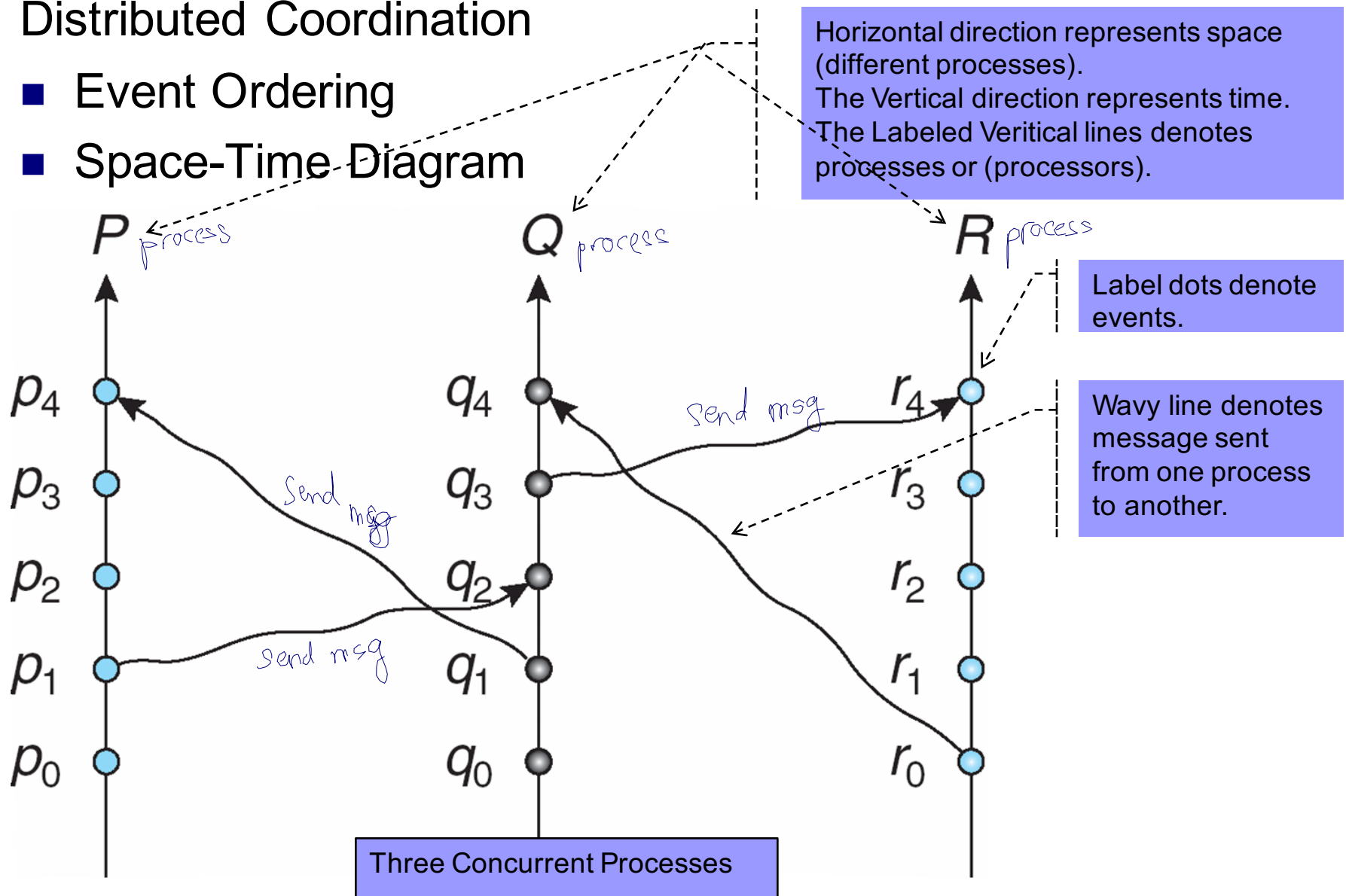
■ Event Ordering

■ Happened-Before Relation:

- Denoted by “ \rightarrow ” on a set of events (Sending and Receiving Message constitutes an Event):
- If A and B are events in the same process, and A was executed before B , then $A \rightarrow B$
- If A event of sending a message by one process and B event of receiving that message by another process, then $A \rightarrow B$
- If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
- Since event cannot happen before itself, the “ \rightarrow ” relation is an irreflexive (binary relationship in a set where no element is related to itself) partial ordering.
- Two events, A and B , not related by the “ \rightarrow ” relation (A did not happen before B , and B did not happen before A), then the two events were executed concurrently.

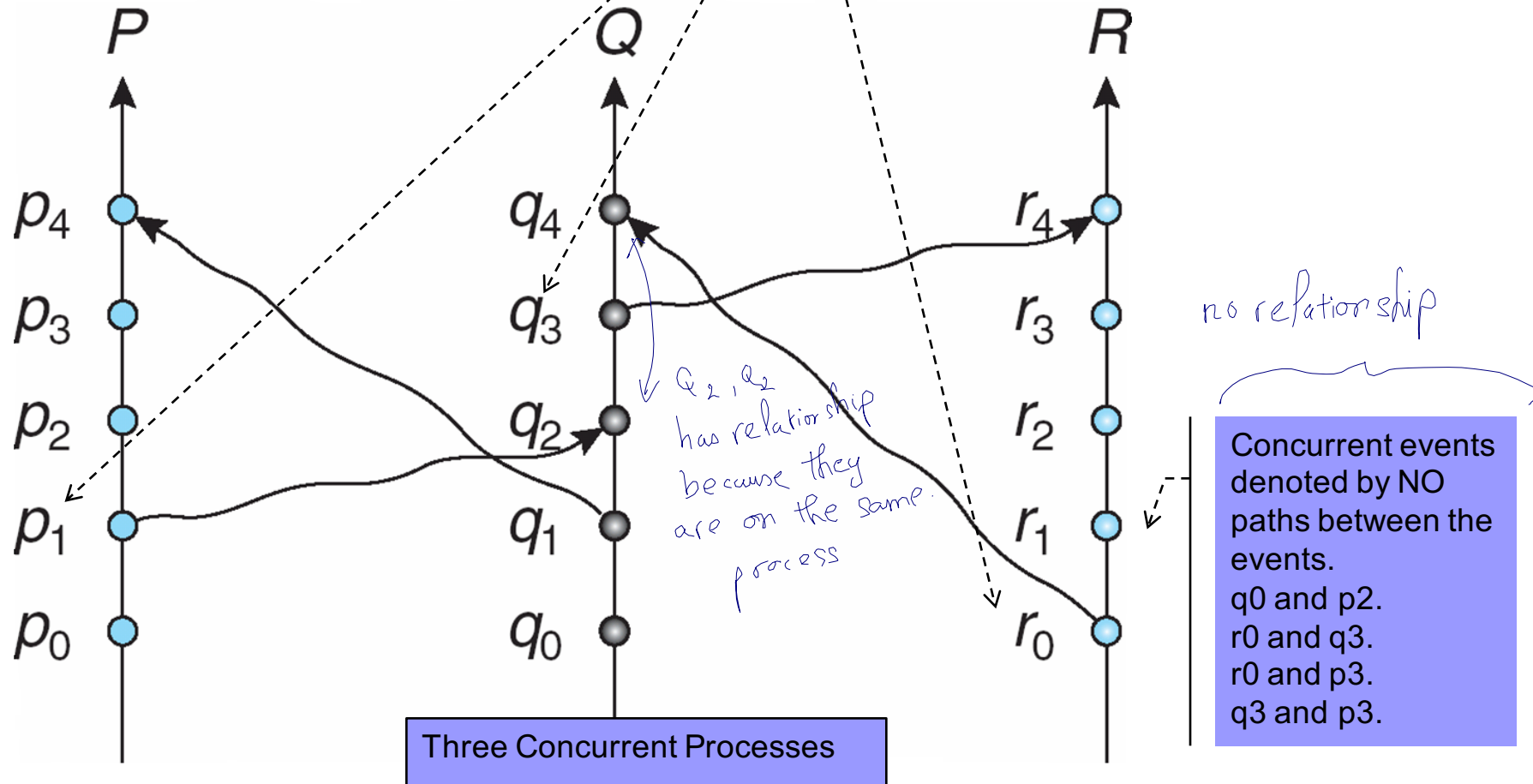
Distributed Coordination

- Event Ordering
- Space-Time Diagram



Distributed Coordination

- Event Ordering
- Space-Time Diagram



Distributed Coordination

Q: which false abt happen before?
Ans: $A \rightarrow B, B \rightarrow C$, A does not have relationship with C.

■ Implementation

■ How to determine Event A happened before Event B without common clock (enforce the global ordering requirement in a Distributed Environment)?

- Associate each system event Timestamp.

- $A \rightarrow B$: Timestamp A less than Timestamp B.

■ Within each process P_i a **logical clock**, LC_i is associated with system event.

artificial timing
to identify the
relationship (order)
between events

- The logical clock is a simple counter, incremented between two successive events executed within a process.

- Logical clock is **monotonically increasing**.

counter of events
receiver when receives a msg from
sender will adjust its clock to
be far with sender's logical clock
if its clock is $<$ timestamp of the
msg (sd equal incr)

Distributed Coordination

- Implementation
- Global Ordering Requirement Across Processes.
 - Unsynchronized Clocks in processes.
- 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

Distributed Coordination

■ Mutual Exclusion

has to be able to communicate with others

■ Maintaining Critical Sections in Distributed System.

■ Centralized Environment: *← 1 coordinator decides the order → overhead, SPOF*

- One Process in the system coordinates the entry into Critical Section.
- Each Process wanting to invoke Mutual Exclusion sends Request Message to the Coordinator.
- The Coordinator checks if another Process is in its Critical Section.
 - If another Process is in its Critical Section, Coordinator queues the Request Message and the Process waits.
 - If no Process is in its Critical Section, Coordinator sends Reply Message to the Process.
- Process receives the Reply Message, the process enter Critical Section.
- When Process completes its Critical Section, the process sends a Release Message to the Coordinator.
- The Coordinator will send Reply Message from its queue (i.e. FCFS Scheduling) when it receives a Release Message.
- Three Messages per Critical-Section Entry: Request, Reply, Release.

Distributed Coordination

■ Mutual Exclusion

has to send a msg to all computer systems to reserve the slot to Critical System

■ Fully Distributed Environment:

- Distribute the decision making across the entire system.

■ 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.

■ Two algorithms to ensure the mutual exclusion execution of processes in their critical sections:

- Using Event Ordering and Time Stamp Approach.
- Using Token Passing Approach.

{ QN

Distributed Coordination

■ Mutual Exclusion

- Approach: Event Ordering and Time Stamp) Q14
- 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. *P_j already in CS so it delays replying until it finishes the CS.*
- 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.

Distributed Coordination

■ Mutual Exclusion

■ Approach: Event Ordering and Time Stamp

■ 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) · newer request gives way to older request.
↓
newer has larger timestamp than older.
 - Otherwise, the reply is deferred

Distributed Coordination

■ Mutual Exclusion

■ Approach: Event Ordering and Timestamp

■ Desirable Behavior:

- 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.

Qn: if a process receives a msg & about to enter its CS, what it should do?

Ans: check its timestamp & compare before decide to reply or defer.

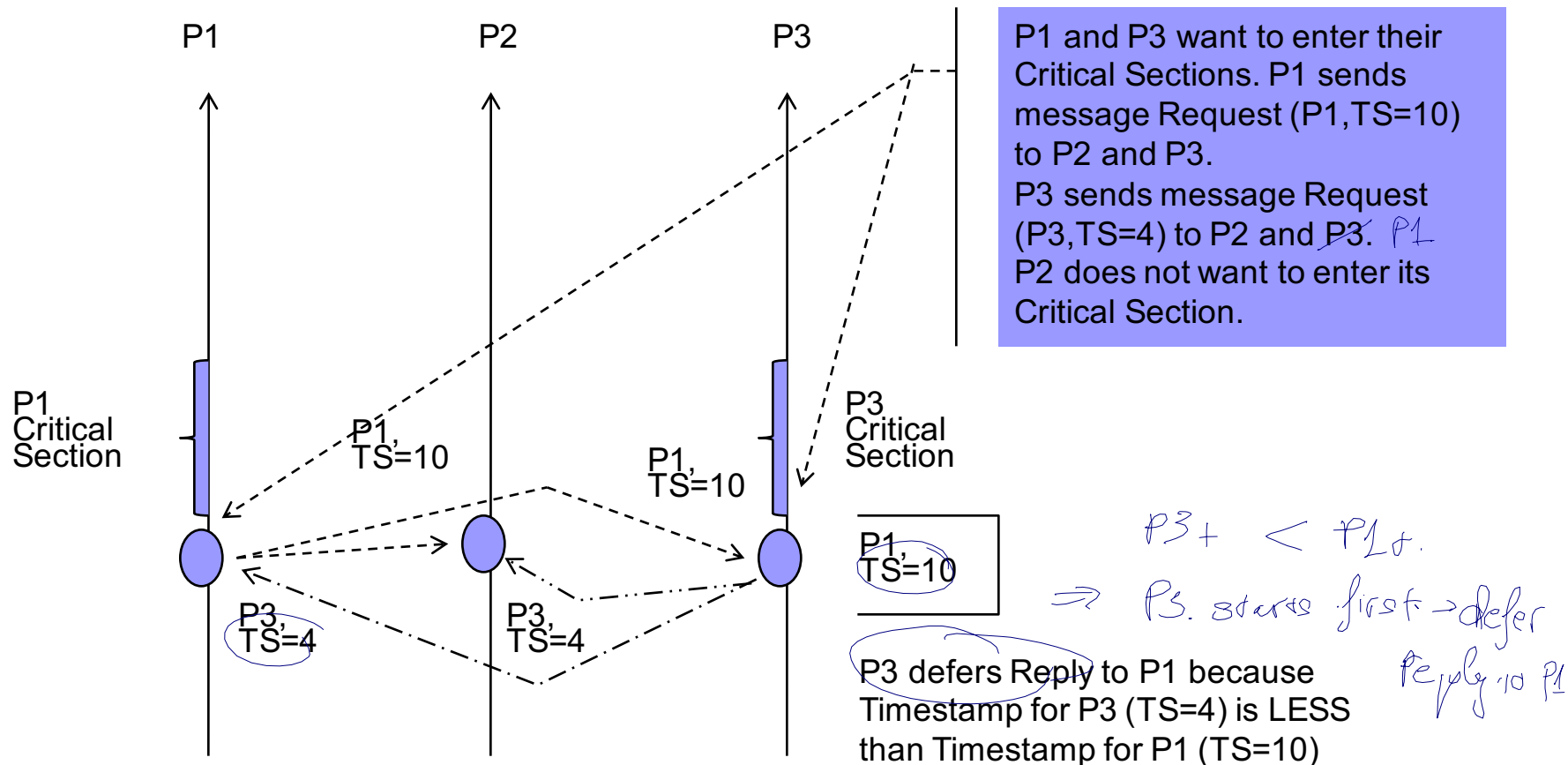
we use timestamp to prevent the Circular Wait; which is a condition of Deadlock.

3 processes: send H, receives H.
3 immediate.
1 delay

Distributed Coordination

- Mutual Exclusion
- Approach: Event Ordering and Timestamp

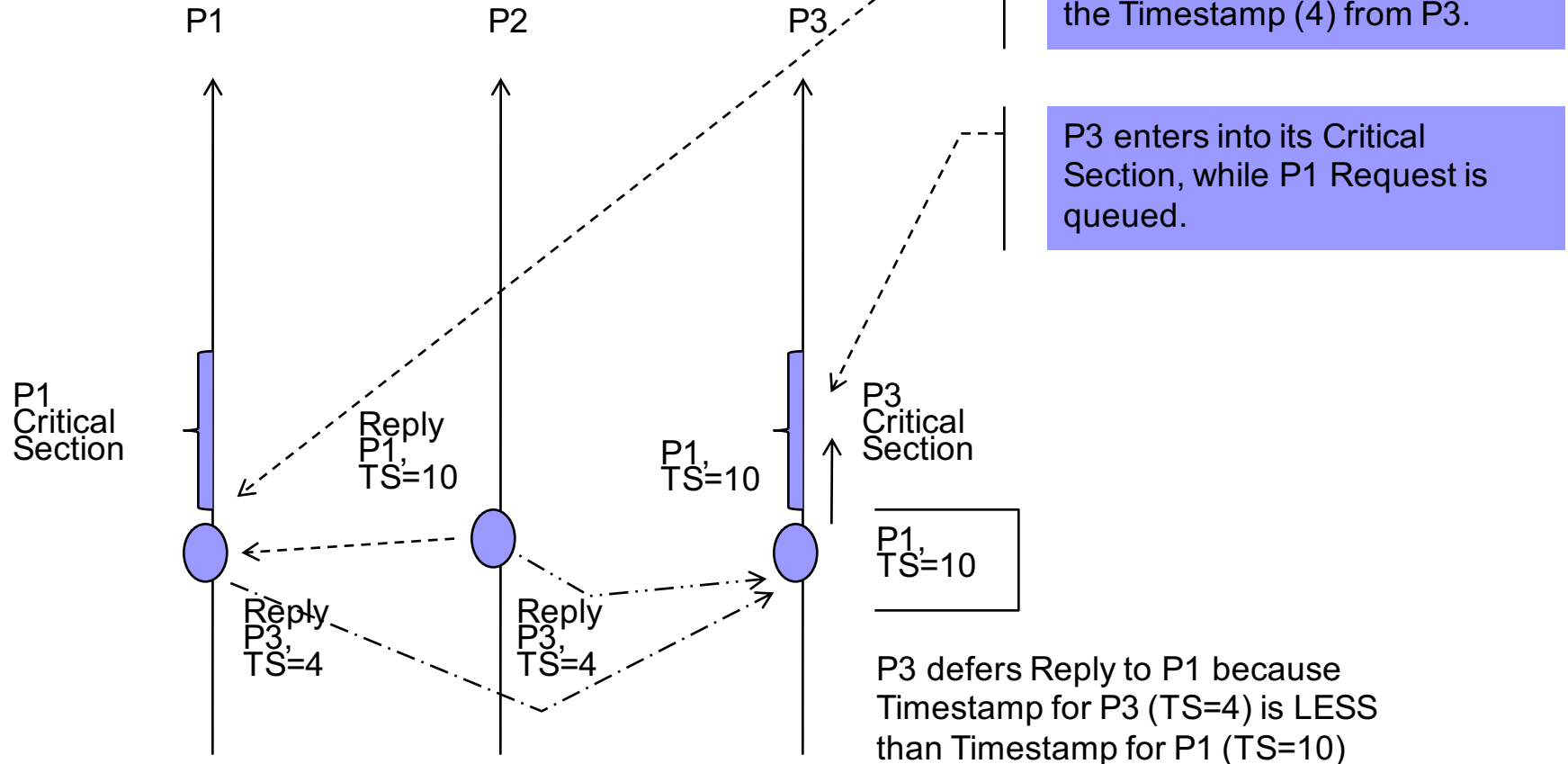
QN



Distributed Coordination

■ Mutual Exclusion

■ Approach: Event Ordering and Timestamp



Distributed Coordination

■ Mutual Exclusion

■ Approach: Event Ordering and Timestamp

■ UnDesirable Behavior

- 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. *has to maintain status of all other processes.*
- 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.

Distributed Coordination

event ordering is used more popular

■ Mutual Exclusion

Q: which algorithm uses a coordinator & receives requests from all processes
Ans. centralized approach.

■ Approach: Token-Passing

■ Circulate a token among processes in system.

- ☐ **Token** is special type of message.
- ☐ Possession of Token entitles holder to enter Critical Section.
- ☐ After Process exits its Critical Section, the Token is passed to next process in the ring.
- ☐ If Process receiving Token does not enter its Critical Section, it passes the Token to the next Process.
- Processes *logically* organized in a **ring structure**.
- Unidirectional ring guarantees freedom from starvation.
- Two types of failures:) Q N
 - ☐ Lost token – election must be called.
 - ☐ Failed processes – new logical ring established.

Distributed Coordination

■ Atomicity

- Atomic Transaction: Program unit where all operations associated with it are executed to completion (i.e. uninterruptable).
- Distributed System: Atomicity of transaction may require several sites participating in execution of a single transaction.
- 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

↑
transaction → 1 fail all reverted.

Distributed Coordination

- Atomicity
- Two-Phase Commit Protocol
- Assumes fail-stop model
- Execution of the protocol is initiated by the Transaction Coordinator that initiated the transaction, after the last step of the transaction has been reached.
- When the protocol is initiated, the transaction may still be executing at some of the local sites.
- The protocol involves all the local sites at which the transaction executed.
- Example: Let T be a transaction initiated at site S_i and let the transaction coordinator at S_i be C_i

Distributed Coordination

- Atomicity
- Two-Phase Commit Protocol
- Example: Transaction T completes at the site S_i (initiated the T).
- Phase 1:
 - C_i adds $\langle \text{prepare } T \rangle$ record to the log.
 - C_i sends $\langle \text{prepare } T \rangle$ message to all sites.
 - When a site receives a $\langle \text{prepare } T \rangle$ message, the transaction manager determines if it can commit the transaction.
 - If no: add $\langle \text{no } T \rangle$ record to the log and respond to C_i with $\langle \text{abort } T \rangle$.
 - If yes:
 - Add $\langle \text{ready } T \rangle$ record to the log.
 - Force *all log records* for T onto stable storage.
 - Send $\langle \text{ready } T \rangle$ message to C_i .

if receiver
can't complete
the tx.

Distributed Coordination

- Atomicity
- Two-Phase Commit Protocol
- Phase 1 (Cont):
- Coordinator collects responses:
 - All respond “ready”,
decision is *commit*.
 - At least one response is “abort”,
decision is *abort*.
 - At least one participant fails to respond within time out period,
decision is *abort*.
 - Site at anytime, can send unconditional <abort T> to the coordinator.
 - Coordinator site can also unconditionally <abort T>, since it is also executing Transaction T.

Distributed Coordination

- Atomicity
- Two-Phase Commit Protocol
- Phase 2:
- Coordinator adds a decision record
 $\langle \text{abort } T \rangle$ or $\langle \text{commit } T \rangle$
to its log and forces record onto stable storage.
notable to change, final.
- Once that record reaches stable storage it is irrevocable (even if failures occur).
- Coordinator sends a message to each participant informing it of the decision (commit or abort).
- Participants take appropriate action locally, logs and forces $\langle \text{commit } T \rangle$ onto stable storage.
- Sites sends an $\langle \text{acknowledge } T \rangle$ to the coordinator, at the end of Phase 2.

Distributed Coordination

- Atomicity
- Two-Phase Commit Protocol **Failure** Handling
- Failure of a Particular Site: After recover at the Site:
- The log contains a $\langle \text{commit } T \rangle$ record
 - In this case, the site executes **redo**(T)
- The log contains an $\langle \text{abort } T \rangle$ record
 - In this case, the site executes **undo**(T)
- The contains a $\langle \text{ready } T \rangle$ record; consult C_i
 - The C_i has requested site commit the Transaction T result:
Consult C_i whether Transaction T was committed or aborted.
 - If C_i is up, C_i sends back whether committed or aborted.
 - If C_i is down, site sends **query-status** T message to the other sites whether committed or aborted.
- The log contains no control records concerning T
 - In this case, the site executes **undo**(T)

look at the logs to recover

Distributed Coordination

- Atomicity
- Two-Phase Commit Protocol Failure Handling
- Failure of the Coordinator: Sites must decide the state of T .
- If an active site contains a $\langle \text{commit } T \rangle$ record in its log, the T must be committed.
- If an active site contains an $\langle \text{abort } T \rangle$ record in its log, then T must be aborted.
- If some active site does *not* contain the record $\langle \text{ready } T \rangle$ in its log then the failed coordinator C_i did not decide to commit T .
 - Rather than wait for C_i to recover, it is preferable to abort T .
- All active sites have a $\langle \text{ready } T \rangle$ record in their logs, but no additional control records ($\langle \text{abort } T \rangle$ or $\langle \text{commit } T \rangle$).
 - In this case we must wait for the coordinator to recover.
 - Blocking problem – T is blocked pending the recovery of site S_i .
The sites can hold locks on data until C_i recovers.

Distributed Coordination

■ Concurrency Control

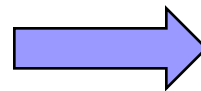
- **Centralized Concurrency:** Multiple Atomic Transaction can execute in a manner where these transactions execute serially in some arbitrary order.

- Concurrency Control algorithms allow Transactions to overlap their execution, but maintain serializability.

can be concurrent, but need to be serializable.

T0	T1
Read (A)	
Write (A)	
Read (B)	
Write (B)	
	Read (A)
	Write (A)
	Read (B)
	Write (B)

Serial Schedule: Transaction
T0 followed by T1



T0	T1
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
Write (B)	
	Read (B)
	Write (B)

Concurrent Serializable Schedule:
Transaction T0 and T1 does NOT
access same data item.

Distributed Coordination

- Concurrency Control
- Distributed Environment Concurrency: Concurrency Control algorithms allow Transactions to overlap their execution, but maintain serializability.
- Modify the centralized concurrency schemes to accommodate the distribution of Transactions.
- Transaction Manager coordinates execution of Transactions (or subtransactions) that access data at local sites.
- Local transaction only executes at that site.
- Global transaction executes at several sites.

Distributed Coordination

- Concurrency Control
- Locking Protocols
- Can use the two-phase locking protocol in a distributed environment by changing how the lock manager is implemented.
- Nonreplicated Data Scheme – Each site maintains a local Lock Manager which administers lock and unlock requests for those data items that are stored in that site.
 - When Transaction wishes to lock data item Q at site S_i , it sends a message to the Lock Manager at site S_i .
 - At site S_i , the request is granted ONLY if data item Q is unlocked or locked in a compatible mode.
 - Site S_i Lock Manager sends Reply Message to requestor.
 - Simple implementation involves two message transfers for handling lock requests, and one message transfer for handling unlock requests.
 - Deadlock handling is more complex.

Distributed Coordination

- Concurrency Control

- Locking Protocols

- Single-Coordinator Approach:

- A single Lock Manager resides in a single chosen site, all lock and unlock requests are made at that site.

- Simple implementation.

- Simple deadlock handling.

- Disadvantage:

 - Possibility of bottleneck.

 - Vulnerable to loss of concurrency controller if single site fails.

- **Multiple-Coordinator Approach:** Distributes Lock Manager function over several sites.

 - Each Lock Manager administers lock/unlock Requests for a subset of the data items.

 - Reduces degree of bottleneck, but multiple sites complicates deadlock handling.

Distributed Coordination

■ Concurrency Control

■ Locking Protocols

■ Majority Protocol Approach:

does not need to wait for ALL reply, just wait for majority.

■ Maintains a Lock Manager at each site.

- Each Lock Manager controls all data or replicas of data at that site.

■ Transaction locking a replicated data item Q, sends Lock Request to greater than half of the sites in which Q is stored.

- Transaction operates on Q ONLY when it has obtained Lock from majority of the replicas.

■ Avoids drawbacks of central control by dealing with replicated data in a decentralized manner.

■ Drawbacks:

- More complicated to implement: Need $2(n/2 + 1)$ messages for Lock Requests and $(n/2 + 1)$ messages for unlock Requests.
- Deadlock-handling algorithms must be modified; possible for deadlock to occur even in locking only data item.

Distributed Coordination

■ Concurrency Control

■ Locking Protocols

■ Biased Protocol Approach:

■ Maintains a Lock Manager at each site.

- Each Lock Manager controls all data at that site.

- Shared and Exclusive Locks are handled differently.

- Shared Locks: Transaction needs lock on shared data item Q, Request ONLY a lock on Q at one site with replica of Q.

- Exclusive Locks: Transaction needs lock on exclusive data item Q, Request lock from ALL sites with replica of Q.

- Similar to majority protocol, but requests for shared locks prioritized over requests for exclusive locks.

- Less overhead on read operations (shared data items) than in majority protocol; but has additional overhead on writes (exclusive data items).

- Like majority protocol, deadlock handling is complex.

Qn: which locking protocol is NOT concurrency control?

Ans: Exclusive Protocol (no such thing called Exclusive)

readable.

exclusive, even read.

Distributed Coordination

- Concurrency Control
- Locking Protocols
- Primary Copy Approach:
 - One of the sites at which a data item replica resides is designated as the Primary Site.
 - Request to lock a data item is made at the primary site of that data item.
- Concurrency control for replicated data handled in a manner similar to that of unreplicated data.
 - Primary Site can handle multiple data item replicas.
 - Single site to send Lock Request.
- Simple implementation, but if Primary Site fails, the data item is unavailable, even though other sites may have a replica.

Distributed Coordination

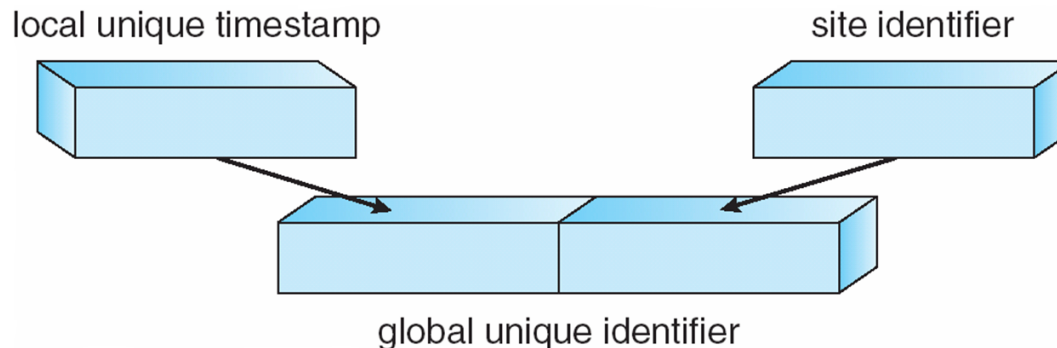
■ Concurrency Control

■ TimeStamping

■ Unique TimeStamp is used to decide the serialization order.

■ Generate unique timestamps in distributed scheme:

- Each site generates a unique local timestamp.
- The global unique timestamp is obtained by concatenation of the unique local timestamp with the unique site identifier.



- Use a *Logical Clock* defined within each site to ensure the fair generation of timestamps.
- Site advances its Logical Clock whenever Transaction T_i with TimeStamp greater than the current Logical Clock.

Distributed Coordination

■ Deadlock Handling

- Distributed Algorithms for Deadlock-Prevention, Deadlock-Avoidance, and Deadlock-Detection.

- **Resource-Ordering Deadlock-Prevention** – Define a global ordering among the system resources. *prevent Circular Wait from happening.*

- ☐ Assign a unique number to all system resources.
- ☐ A process may request a resource with unique number i only if it is not holding a resource with a unique number greater than i .
- ☐ Simple to implement; requires little overhead.

- **Banker's algorithm** – Designate one of the processes in the system as the process that maintains the information necessary to carry out the Banker's algorithm.

- ☐ Also implemented easily, but high overhead, every Resource Request must be channeled through the Banker.

Distributed Coordination

■ Deadlock Handling

■ DeadLock-Prevention using TimeStamp with Resource Preemption:

- Priority numbers assigned to each Process.
- Priority numbers are used to decide whether a process P_i should wait for a process P_j ; otherwise P_i is rolled back.
- The scheme prevents deadlocks
 - For every edge $P_i \rightarrow P_j$ in the wait-for graph, P_i has a higher priority than P_j .
 - Thus a cycle cannot exist.
- Possibility of Starvation. Process with low priority could always be rolled back.
- TimeStamp for each process in the system.

priority

timestamp

older request has higher priority than younger one.

Distributed Coordination

■ Deadlock Handling

■ Wait-Die NonPreemptive Scheme:

□ Based on a nonpreemptive technique.

□ If P_i requests a resource currently held by P_j , P_i is allowed to wait only if it has a smaller timestamp than does P_j (P_i is older than P_j).

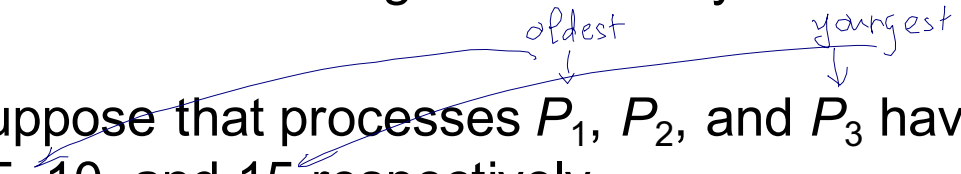
■ Otherwise, P_i is rolled back (dies): A younger process is killed before it can be caught in a wait cycle.

□ Example: Suppose that processes P_1 , P_2 , and P_3 have timestamps 5, 10, and 15 respectively

older can wait
- younger rolled back

■ if P_1 request a resource held by P_2 , then P_1 will wait ← older waits can get Q.

■ If P_3 requests a resource held by P_2 , then P_3 will be rolled back ← no younger waits for older, dropped.



Distributed Coordination

■ Deadlock Handling

■ Wound-Wait Preemptive Scheme.

- Based on a preemptive technique; counterpart to the wait-die system
- If P_i requests a resource currently held by P_j , P_i is allowed to wait only if it has a larger timestamp than does P_j (P_i is younger than P_j). Otherwise P_j is rolled back (P_j is wounded by P_i).
- Example: Suppose that processes P_1 , P_2 , and P_3 have timestamps 5, 10, and 15 respectively
 - If P_1 requests a resource held by P_2 , then the resource will be preempted from P_2 and P_2 will be rolled back.
 - If P_3 requests a resource held by P_2 , then P_3 will wait.
- The Wound-Wait algorithm preempts the younger process. When the younger process re-requests resource, it has to wait for older process to finish. This is the better of the two algorithms.

- older preempt younger.
- younger wait for older.

Distributed Coordination

Qn: favors younger one rolled back then.

Ans: Wait-Die.

■ Deadlock Handling

■ Differences in Wait-Die Scheme and Wound-Wait Scheme.

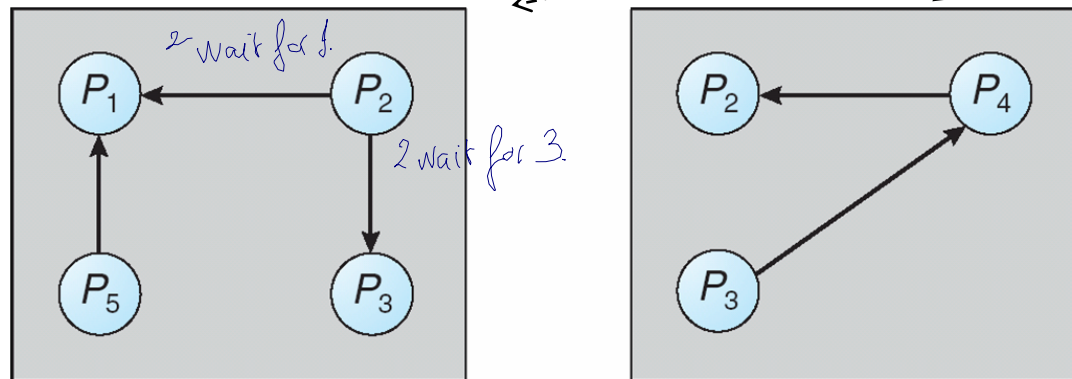
- In Wait-Die Scheme: Older process will wait for a younger process to release its resource. Younger process will not wait for older process and younger process will roll-back.
- In Wound-Wait Scheme: Older process will preempt the younger process for the resource and younger process rolled-back. Younger process will wait for older process.
- Younger process will tend to be rolled-back more in Wait-Die Scheme because the rolled-back Younger process will request the same resource again and get rolled-back.
- Younger process will be rolled-back less in Wound-Wait Scheme because the Younger process is allowed to wait for the resource held by older process.

Distributed Coordination

- Deadlock Handling
- Deadlock Detection
- Using a Wait-For Graph to describe the Resource-Allocation State, a cycle in the graph would represent a Deadlock.
 - How to maintain Wait-For Graph for Distributed System?
 - Local wait-for graphs at each local site. The nodes of the graph correspond to all the processes that are currently either holding or requesting any of the resources local to that site
 - May also use a global wait-for graph. This graph is the union of all local wait-for graphs.

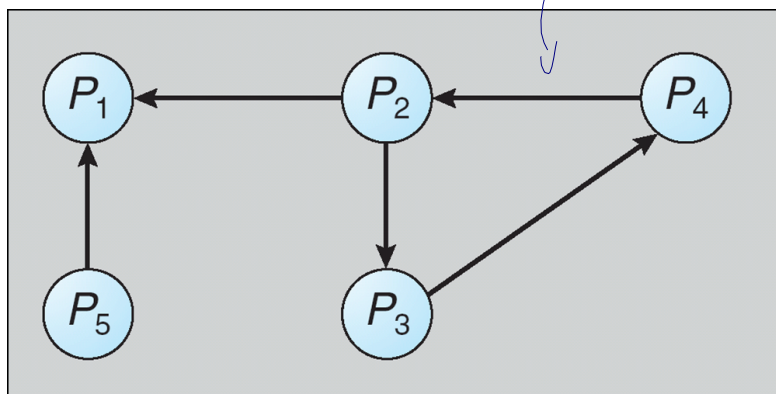
Distributed Coordination

- Deadlock Handling
- Deadlock Detection



site S_1

site S_2



Local Wait-For Graphs for Local Processes and Resources. Process P3 in Site S1 needs a Resource held by Process P4 in Site S2, a Request Message is sent by P3 to Site S2. This is represented by the edge: P3 → P4 in Site S2.

No cycles in Local Wait-For Graphs does NOT mean there are no Deadlocks. The UNION of all Local Wait-For Graphs must be acyclic to imply the Distributed System cannot be in a Deadlock State.

Distributed Coordination

■ Deadlock Handling

- Deadlock Detection: Centralized Approach
- Each site keeps a local wait-for graph.
- A global wait-for graph is maintained in a single coordination process, the Deadlock-Detection Coordinator.
 - Two Types of Wait-For Graphs because of communication delays in the system.
 - Real Graph: Describes the real, but unknown state of the system at any point in time.
 - Constructed Graph: Approximation generated by the Coordinator during execution of its algorithm.
 - If a deadlock exists, then it is reported properly as a cycle.
 - If a deadlock is reported, then the system is in a Deadlocked State.

Distributed Coordination

■ Deadlock Handling

■ Deadlock Detection: Centralized Approach

■ There are three different options (points in time) when the Wait-For Graph may be constructed:

1. Whenever a new edge is inserted or removed in one of the local Wait-For Graphs.

Local site sends a message to the Coordinator: Coordinator updates its Global Wait-For Graph.

2. Periodically, when a number of changes have occurred in a Wait-For Graph.

Local site sends a number of changes in a single message to the Coordinator.

3. Whenever the coordinator needs to invoke the cycle-detection algorithm.

Distributed Coordination

- Deadlock Handling
- Deadlock Detection: Centralized Approach
- Invoking the Deadlock-Detection Algorithm uses the Global Wait-For Graph.
 - Cycle found, a victim is selected to be rolled-back.
 - Coordinator notifies ALL sites of the victim: Sites roll-back the victim process.
- Unnecessary rollbacks may occur as a result of false cycles.

false positive.

Distributed Coordination

- Deadlock Handling
- Deadlock Detection: Centralized Approach
- Unnecessary Rollbacks: Occur as a result of false cycles.
 - **False cycles** may exist in Global Wait-For Graph.
 - Caused by unsynchronized arrival of messages to Delete and Insert graph edges in the Global Wait-For Graph: Coordinator receives the Insert edge before the Delete edge message.
 - Process has been aborted (unrelated to the Deadlock) and removes a cycle in the Global Wait-For Graph, but the Coordinator already acting on the detected cycle and roll-back a process unnecessarily.

Distributed Coordination

■ Deadlock Handling

■ Deadlock Detection: Centralized Approach

■ Coordinator Requests ALL Local Wait-For Graphs Approach: Detects Deadlocks and does not detect False Deadlocks.

- Append unique identifiers (timestamps) to requests from different sites.
 - Local Requests do not need identifiers.
- When process P_i , at site A , requests a resource from process P_j , at site B , a request message with timestamp TS is sent.
- The edge $P_i \rightarrow P_j$ with the label TS is inserted in the local wait-for of A . The edge is inserted in the local wait-for graph of B only if B has received the request message and cannot immediately grant the requested resource

Distributed Coordination

■ Deadlock Handling

■ Deadlock Detection: Centralized Approach

■ Coordinator Requests ALL Local Wait-For Graphs Approach: Detects Deadlocks and does not detect False Deadlocks.

□ Detection Algorithm:

1. The controller sends an initiating message to each site in the system.
2. On receiving this message, a site sends its local Wait-For Graph to the coordinator.
3. When the controller has received a reply from each site, it constructs a global Wait-For Graph as follows:
 - (a) The constructed graph has a vertex for every process system.
 - (b) The graph has an edge $P_i \rightarrow P_j$ if and only if
 - (1) there is an edge $P_i \rightarrow P_j$ in one of the Wait-For Graphs, or
 - (2) an edge $P_i \rightarrow P_j$ with some label TS appears in more than one Wait-For Graph.

If the constructed graph contains a cycle \Rightarrow deadlock

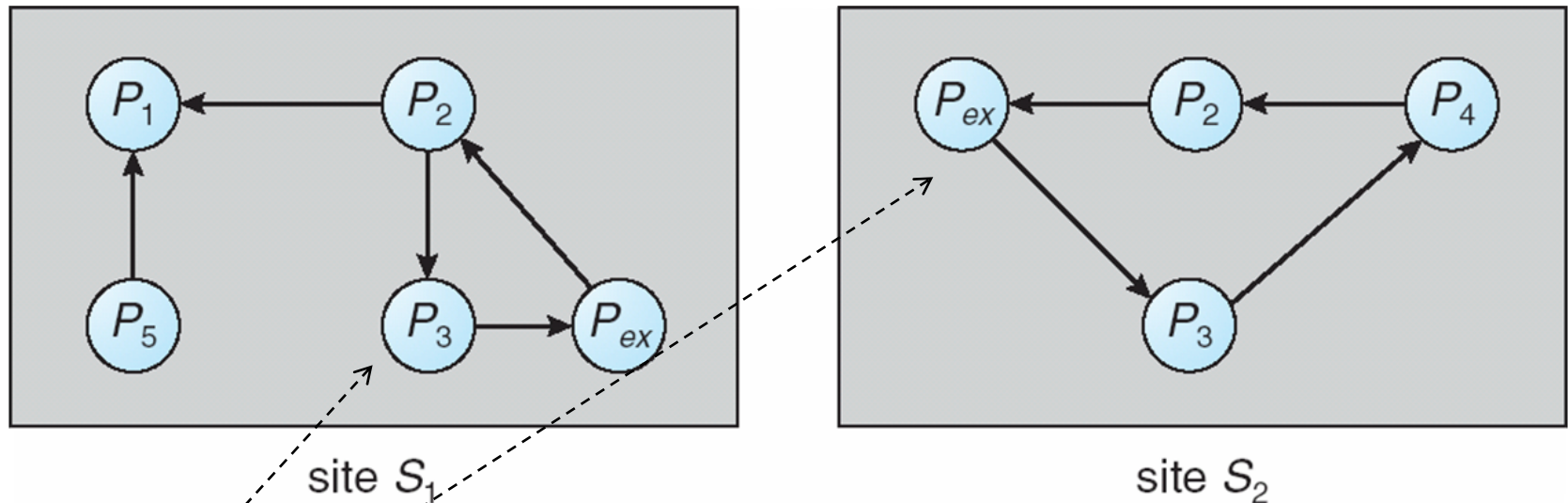
Distributed Coordination

■ Deadlock Handling

- Deadlock Detection: Fully Distributed Approach
- All controllers share equally the responsibility for detecting deadlock.
- Every site constructs a Wait-For Graph that represents a part of the total graph. *builds its own Wait For*
- We add one additional node P_{ex} to each local Wait-For Graph.
 - An edge $P_i \rightarrow P_{ex}$ exists if P_i is waiting for a data item in another site being held by *any* process.
 - An edge $P_{ex} \rightarrow P_j$ exists if P_{ex} (Process at another site) is waiting to acquire a resource held by P_j in the local site.
- If a local Wait-For Graph contains a cycle that does not involve node P_{ex} , then the system is in a Deadlock state.
- A cycle involving P_{ex} implies the possibility of a deadlock.
 - To ascertain whether a deadlock does exist, a distributed deadlock-detection algorithm must be invoked.

Distributed Coordination

- Deadlock Handling
- Deadlock Detection: Fully Distributed Approach



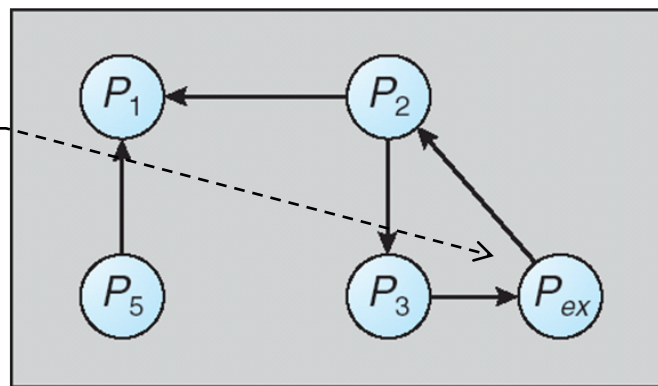
An edge $P_i \rightarrow P_{ex}$ exists if P_i is waiting for a data item in another site being held by *any* process. An edge $P_{ex} \rightarrow P_j$ exists if P_{ex} (Process at another site) is waiting to acquire a resource held by P_j in the local site. If a local graph contains a cycle involving P_{ex} , then this implies the possibility of a deadlock.

Distributed Coordination

■ Deadlock Handling

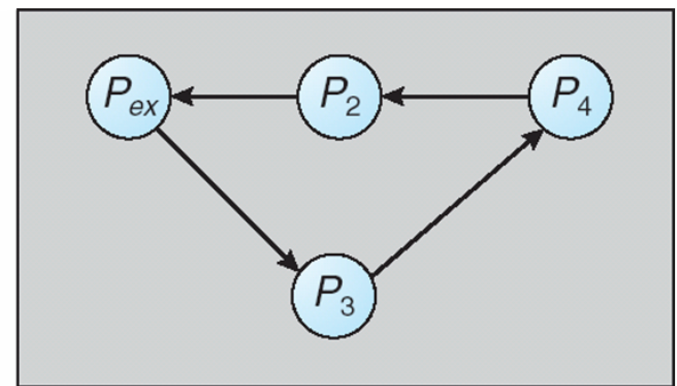
- Deadlock Detection: Fully Distributed Approach
- Distributed Deadlock-Detection Algorithm: At site, the local Wait-For Graph must contain a cycle involving P_{ex} .
- $P_{ex} \rightarrow P_{k1} \rightarrow P_{k2} \rightarrow \dots \rightarrow P_{kn} \rightarrow P_{ex}$
- Process P_{kn} in site S_i is waiting to acquire a data item located at some other site, S_j .
- Site S_i sends site S_j the Deadlock-Detection Message and site S_j updates its local Wait-For Graph and checks for cycle involving P_{ex} .

Site S_1 discovers the cycle:
 $P_{ex} \rightarrow P_2 \rightarrow P_3 \rightarrow P_{ex}$
Site S_1 sends message to S_2 describing the cycle.



site S_1

Site S_1 : $P_{ex} \rightarrow P_2 \rightarrow P_3 \rightarrow P_{ex}$.



site S_2

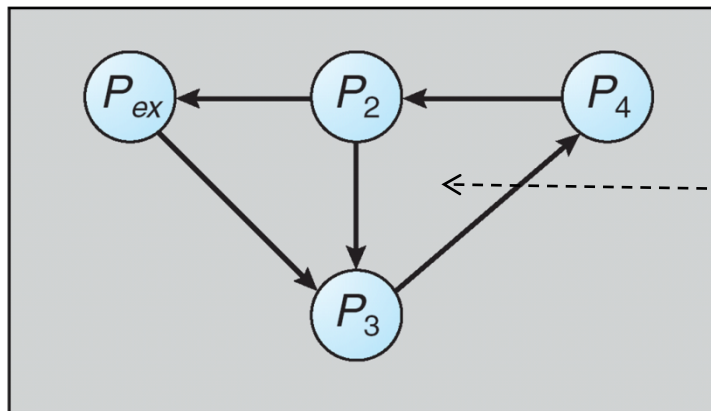
Site S_2 : $P_{ex} \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_{ex}$.

Distributed Coordination

■ Deadlock Handling

- Deadlock Detection: Fully Distributed Approach
- Distributed Deadlock-Detection Algorithm: (Cont)
- S2 receives the Deadlock-Detection Message, it updates its local Wait-For Graph, obtaining the graph containing the cycle that does NOT include P_{ex} .

$P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2$.



site S_2

Local Wait-For Graph contains a cycle that does NOT contain P_{ex} .
A recovery scheme is invoked at S2 to remove the cycle.

Distributed Coordination

■ Deadlock Handling

■ Deadlock Detection: Fully Distributed Approach

■ Distributed Deadlock-Detection Algorithm: (Cont)

■ Reduce Deadlock-Detection Message traffic:

- Multiple Deadlock-Detection Message can be sent, when sites discovers the cycle with *Pex*.
- Assign each process *P_i* a unique identifier.
- Local Wait-For Graph with a Pex cycle, will send Deadlock-Detection Message ONLY when:
 $ID(P_{k1}) < ID(P_{k2})$

Relationship between processes could be:

$ID(P_1) < ID(P_2) < ID(P_3) < ID(P_4)$

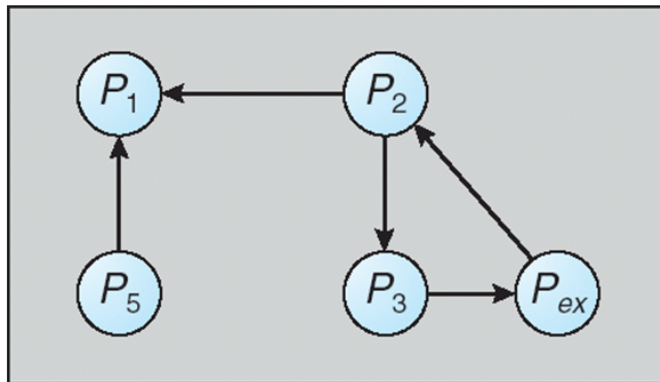
Distributed Coordination

■ Deadlock Handling

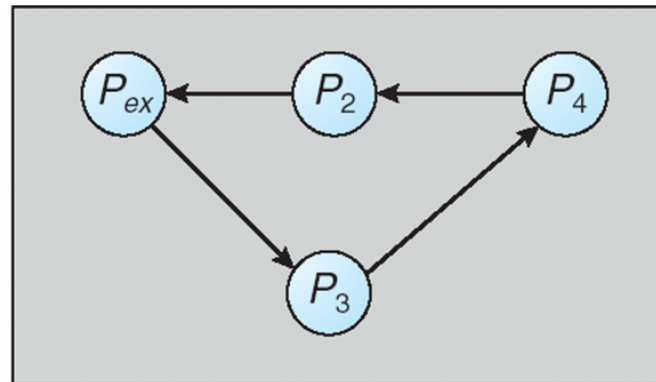
■ Deadlock Detection: Fully Distributed Approach

■ Distributed Deadlock-Detection Algorithm: (Cont)

■ Reduce Deadlock-Detection Message traffic: (Cont)



site S_1



site S_2

■ Both sites discovers cycle in local Wait-For Graph:

□ Site S_1 : $P_{ex} \rightarrow P_2 \rightarrow P_3 \rightarrow P_{ex}$

■ The edge with P_{ex} is $P_3 \rightarrow P_{ex} \rightarrow P_2 = ID(P_3) > ID(P_2)$

□ Site S_2 : $P_{ex} \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_{ex}$

■ The edge with P_{ex} is $P_2 \rightarrow P_{ex} \rightarrow P_3 = ID(P_2) < ID(P_3)$. Send Deadlock-Detection Message.

Distributed Coordination

■ Election Algorithms

■ Coordinator process used in Distributed Systems to:

- Enforce Mutual Exclusion.
- Maintaining Global Wait-For Graph for Deadlock Detection.
- Replacing lost Token.
- Controlling an input or output device on the system.

■ Failure of Coordinator requires restarting the Coordinator at some OTHER site.

- Algorithm used to determine where new copy of coordinator is restarted are called Election Algorithms.

■ Election Algorithms uses an unique priority number, associated with each active process in the system: Assume that the priority number of process P_i is I .

- Coordinator is always the process with the largest priority number.
- When Coordinator fails, Election Algorithm just elect the active process with the largest priority number.
- The elected Process's number must be sent to ALL processes in the system.

Distributed Coordination

■ Election Algorithms

- Two algorithms, the Bully Algorithm and a Ring Algorithm, can be used to elect a new coordinator in case of failures.
- Bully Algorithm:
 - Applicable to systems where every process can send a message to every other process in the system.
 - If process P_i sends a request that is not answered by the coordinator within a time interval T , assume that the coordinator has failed; P_i tries to elect itself as the new coordinator.
 - P_i sends an election message to every process with a higher priority number, P_i then waits for any of these processes to answer within T .

Distributed Coordination

■ Election Algorithms

- Two algorithms, the Bully Algorithm and a Ring Algorithm, can be used to elect a new coordinator in case of failures.
- Bully Algorithm:
 - Applicable to systems where every process can send a message to every other process in the system.
 - If process P_i sends a request that is not answered by the coordinator within a time interval T , assume that the coordinator has failed; P_i tries to elect itself as the new coordinator.
 - P_i sends an election message to every process with a higher priority number, P_i then waits for any of these processes to answer within T .

Distributed Coordination

■ Election Algorithms

- Two algorithms, the Bully Algorithm and a Ring Algorithm, can be used to elect a new coordinator in case of failures.
- Bully Algorithm:
 - If no response within T , assume that all processes with numbers greater than i have failed; P_i elects itself the new coordinator.
 - If answer is received, P_i begins time interval T' , waiting to receive a message that a process with a higher priority number has been elected.
 - Some higher priority process is electing itself coordinator.
 - If no message is sent within T' , assume the process with a higher number has failed; P_i should restart the algorithm.

Distributed Coordination

■ Election Algorithms

■ Bully Algorithm: (Cont)

- If P_i is not the coordinator, then, at any time during execution, P_i may receive one of the following two messages from process P_j .
 - P_j is the new coordinator ($j > i$). P_i , in turn, records this information.
 - P_j started an election ($j < i$). P_i , sends a response to P_j and begins its own election algorithm, provided that P_i has not already initiated such an election.
- After a failed process recovers, it immediately begins execution of the same algorithm.
- If there are no active processes with higher numbers, the recovered process forces all processes with lower number to let it become the coordinator process, even if there is a currently active coordinator with a lower number (reason algorithm named “Bully Algorithm”).

Distributed Coordination

■ Election Algorithms

■ Ring Algorithm expects links between processes are unidirectional.

- Each process sends its message to the neighbor on the right.
- Each process has Active List: List containing the priority number of all active processes in the system, after Ring Algorithm completes.

■ Ring Algorithm:

- If process P_i detects a coordinator failure, it creates a new active list that is initially empty. It then sends a message $\text{elect}(i)$ to its right neighbor, and adds the number i to its active list.
- If P_i receives a message $\text{elect}(j)$ from the process on the left, it must respond in one of three ways:

Distributed Coordination

■ Election Algorithms

■ Ring Algorithm: (Cont)

- If P_i receives a message $elect(j)$ from the process on the left, it must respond in one of three ways:
 1. If this is the first *elect* message it has seen or sent, P_i creates a new active list with the numbers i and j
 - ✦ It then sends the message $elect(i)$, followed by the message $elect(j)$
 2. If $i \neq j$, the message received does not contain P_i 's number, then P_i adds j to its active list and forwards message to the right.
 3. If $i = j$, then P_i receives the message $elect(i)$
 - ✦ The active list for P_i contains all the numbers of the active processes in the system
 - ✦ P_i can now determine the largest number in the active list to identify the new coordinator process



Distributed Coordination

■ Reaching Agreement

- System reliability require mechanism that allows a set of process to agree on a common “value”.

- Such agreement may not take place due to:
 - Faulty communication medium.
 - Faulty processes:
 - Processes may send garbled or incorrect messages to other processes.
 - A subset of the processes may collaborate with each other in an attempt to defeat the scheme.

Distributed Coordination

■ Reaching Agreement

■ Unreliable Communications:

- Process P_i at site A , has sent a message to process P_j at site B ; to proceed, P_i needs to know if P_j has received the message

■ Detect failures using a time-out scheme

- When P_i sends out a message, it also specifies a time interval during which it is willing to wait for an acknowledgment message from P_j
- When P_j receives the message, it immediately sends an acknowledgment to P_i
- If P_i receives the acknowledgment message within the specified time interval, it concludes that P_j has received its message
 - If a time-out occurs, P_i needs to retransmit its message and wait for an acknowledgment
- Continue until P_i either receives an acknowledgment, or is notified by the system that B is down

Distributed Coordination

■ Reaching Agreement

■ Unreliable Communications: (Cont)

- Suppose that P_j also needs to know that P_i has received its acknowledgment message, in order to decide on how to proceed.
 - In the presence of failure, it is not possible to accomplish this task.
 - It is not possible in a distributed environment for processes P_i and P_j to agree completely on their respective states.
 - P_i sends a message to P_j .
 - P_j sends an acknowledgement to P_i .
 - P_i receives the acknowledgement, but P_j cannot be certain that P_i has received the acknowledgement, because of the possibility of failure on communication medium.

Distributed Coordination

■ Reaching Agreement

■ Faulty Processes:

- Assuming communication medium is reliable, processes can fail in unpredictable ways.

- Consider a system of n processes, of which no more than m are faulty

- Suppose that each process P_i has some private value of V_i

- Devise an algorithm that allows each nonfaulty P_i to construct a vector $X_i = (A_{i,1}, A_{i,2}, \dots, A_{i,n})$ such that::

- If P_j is a nonfaulty process, then $A_{ij} = V_j$.

- If P_i and P_j are both nonfaulty processes, then $X_i = X_j$.

- For multiple rounds of exchanges to create multiple vectors, if at least 2 of 3 values in the vectors agree, then the process can make a decision if the process is trustworthy.

Distributed Coordination

■ Summary

- Synchronizing actions in a Distributed System is problematic because of no common memory and no common clock.
- Order of events in a Distributed System can be aligned partially using the Happened-Before Relationship:
 - Happened-Before Relationship with a Timestamp for total event ordering.
- Critical Sections in a Distributed System using Mutual Exclusion:
 - Centralized Approach: One of the processes in the system coordinate the entry to Critical Section.
 - Fully Distributed Approach: Decision making distributed across the system using Request/Reply Messaging and Timestamp.
 - Token-Passing Approach: Circulates a Token to allow a process to enter its Critical Section.
- Atomicity in a Distributed System needs all processes to complete operations associated with the same program unit.
 - Requires a Transaction Coordinator at each site.
 - A Transaction either commits at all sites or aborts at all sites: Using Two-Phase Commit Protocol (2PC Protocol).

Distributed Coordination

■ Summary (Cont)

■ Concurrency-Control Schemes in Distributed System:

- Transaction Manager maintains log and participates in the Concurrency-Control Scheme to coordinate concurrent execution of Transactions.
- Single-Coordinator with Lock Manager for delaying transactions on shared sites.
- Global Timestamp with Local Timestamp and site identifier for Deadlock Handling.
- Deadlock-Prevention Scheme using Wait-Die Nonpreemptive Scheme and Would-Wait Preemptive Scheme.
- Deadlock Detection can be managed using a Global Wait-For Graph consolidated from the Local Wait-For Graphs from the sites.
 - Methods for organizing the Wait-For Graph are Centralized Approach and Fully Distributed Approach.
- Coordinator used extensively in Distributed System, but require mechanisms for recovering failed Coordinator.
 - Bully Algorithm and Ring Algorithm used to elect new Coordinator in case of failure.