Event Ordering

- **Happened-before** relation (denoted by $\rightarrow$)
  - If $A$ and $B$ are events in the same process (assuming sequential processes), 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 $A \rightarrow C$
  - If two events $A$ and $B$ are not related by the $\rightarrow$ relation, then these events are executed **concurrently**.
Relative Time for Three Concurrent Processes

Which events are concurrent and which ones are ordered?

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: Centralized 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 it sends that process a reply message
- 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: 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 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 (Cont.)

- 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

- Example: P1 sends a request to P2 and P3 (timestamp=10)
  P3 sends a request to P1 and P2 (timestamp=4)

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
Distributed Deadlock Handling

• Resource-ordering deadlock-prevention
  => define a *global* ordering among the system resources
    - 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

• Timestamp-ordering deadlock-prevention
  => unique Timestamp assigned when each process is created

  1. wait-die scheme -- non-reemptive
  2. wound-wait scheme -- preemptive

Prevention: Wait-Die Scheme

• non-preemptive approach
• 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 - releases resources)

• Example: Suppose that processes \(P_1\), \(P_2\), and \(P_3\) have
  timestamps 5, 10, and 15 respectively
  - if \(P_1\) request a resource held by \(P_2\), then \(P_1\) will wait
  - If \(P_3\) requests a resource held by \(P_2\), then \(P_3\) will be rolled back

• The older the process gets, the more waits
Prevention: Wound-Wait Scheme

- Preemptive approach, 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 rolled-back process eventually gets the smallest timestamp.

Comparison

- Both avoid starvation, provided that when a process is rolled back, it is not assigned a new timestamp

- In wait-die, older process must wait for the younger one to release its resources. In wound-wait, an older process never waits for a younger process.

- There are fewer roll-backs in wound-wait.
  - $P_i$->$P_j$; $P_i$ dies, requests the same resources; $P_i$ dies again...
  - $P_j$->$P_i$; $P_i$ wounded. requests the same resources; $P_i$ waits.
Deadlock Detection

Two Local Wait-For Graphs

Global Wait-For Graph
Deadlock Detection - Centralized Approach

- Each site keeps a local wait-for graph
  - 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
- A global wait-for graph is maintained in a single coordination process; this graph is the union of all local wait-for graphs
- 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
  2. Periodically, when a number of changes have occurred in a wait-for graph
  3. Whenever the coordinator needs to invoke the cycle-detection algorithm
- Option 1: unnecessary rollbacks may occur as a result of false cycles

Local and Global Wait-For Graphs
Detection Algorithm Based on Option 3

- Append unique identifiers (timestamps) to requests from different sites
- 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

Algorithm: Option 3

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 graph as follows:
   (a) The constructed graph contains a vertex for every process in the system
   (b) The graph has an edge $P_i \rightarrow P_j$ if and only if
       - there is an edge $P_i \rightarrow P_j$ in one of the wait-for graphs, or
If the constructed graph contains a cycle $\Rightarrow$ deadlock

*To avoid report of false deadlocks, requests from different sites appended with unique ids (timestamps)*
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
- We add one additional node $P_{ex}$ to each local wait-for graph
  - $P_i \rightarrow P_{ex}$ exists if $P_i$ is waiting for a data item at another site being held by any process
- 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

Augmented Local Wait-For Graphs
Augmented Local Wait-For Graph in Site S2

Distributed File Systems

- Distributed file system (DFS) - a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources over a network

- A DFS manages set of dispersed storage devices

- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces

- There is usually a correspondence between constituent storage spaces and sets of files
DFS Structure

- **Service** - software entity running on one or more machines and providing a particular type of function to a priori unknown clients

- **Server** - service software running on a particular machine

- **Client** - process that can invoke a service using a set of operations that forms its *client interface*

- A client interface for a file service is formed by a set of primitive *file operations* (create, delete, read, write)

- Client interface of a DFS should be transparent, i.e., not distinguish between local and remote files

Naming and Transparency

- **Naming** - mapping between logical and physical objects

- Multilevel mapping - abstraction of a file that hides the details of how and where on the disk the file is actually stored

- A *transparent* DFS hides the location where in the network the file is stored

- For a file being replicated in several sites, the mapping returns a set of the locations of this file’s replicas; both the existence of multiple copies and their location are hidden
Naming Structures

- **Location transparency** - file name does not reveal the file's physical storage location
  - File name still denotes a specific, although hidden, set of physical disk blocks
  - Convenient way to share data
  - Can expose correspondence between component units and machines

- **Location independence** - file name does not need to be changed when the file’s physical storage location changes
  - Better file abstraction
  - Promotes sharing the storage space itself
  - Separates the naming hierarchy form the storage-devices hierarchy