Chapter 18 - Distributed Coordination

Distributed coordination controls several **mechanisms** to ensure the Distributed Environment is working in order.

- Event ordering
- Mutual Exclusion
- Atomicity
- Concurrency Control
- Deadlock Handling
- Eelection Algorithms
- Reaching Agreement

1. Event Ordering

There are 2 ordering systems: *Centralized System, or Distributed System*.

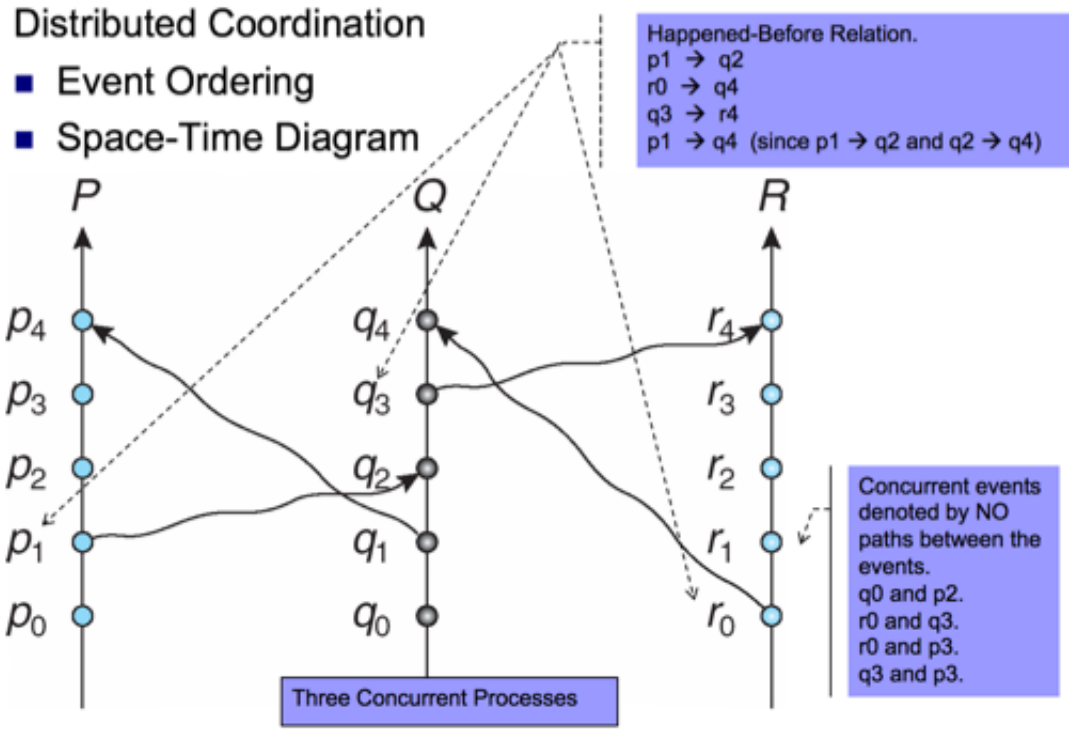
1.1. In Centralized System

> Control the order of events by a single common clock and memory.

1.2. In Distributed System

> **Happen-Before Relation**, denoted as `A –> B`
>
> - Process A & process B are in the same processor, their order is based on their timestamp (FCFS).
> - If A sends B a message, then `A -> B`
> - If `A -> B` AND `B -> C`, then `A -> C`
> - Otherwise, A and B has no relation, and thus can be executed concurrently.

Example:



1.2.1. Implementation

- Each process has a logical clock (LCi) that generates its own timestamp.
- If **A -> B**, then A timestamp must be **less than** B timestamp.
- If A sends B a message (with A's timestamp), and B logical clock was found smaller than that timestamp, B will adjust advances its logical clock to be greater than the timestamp.

2. Mutual Exclusion

> Control how a process can enter its Critical Section state in a mutual exclusive way.

2.1. In Centralized Environment

- A **Coordinator** process governs the access to Critical Section using **Request, Reply, and Release** messages.
  - A process sends a **Request** to the Coordinator to enter to its CS.
  - Coordinate checks and
    - if there is already a process currently in its CS, new process is queued.
    - if there is no process in its CS, Coordinate sends **Reply** message to let the process proceed to its CS.
  - When the process completes its CS, it sends a **Release** message so the Coordinator can allow the next process into its CS.

Message Count: 3 msgs per CS entry

2.2. In Distributed Environment

Using one of these **2 aproaches: event ordering & timestamp, token passing**.

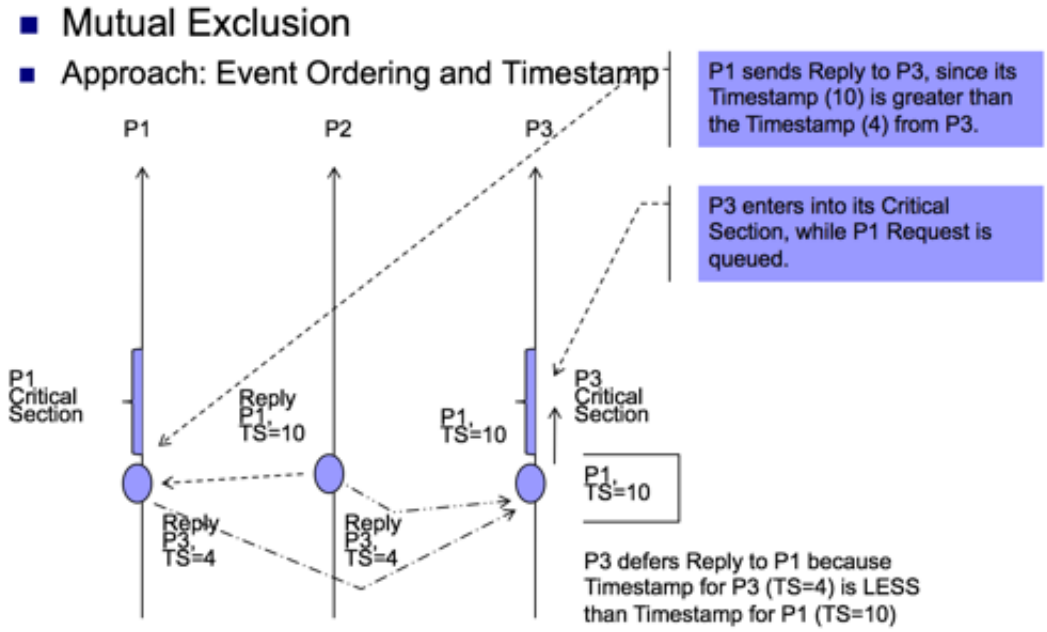
2.2.1. Event Ordering and Timestamp

How it works:

- a process Pi wants to enter its CS, it sends messages `<PID, timestamp>` to ALL other processes.
  - if it receives replies from ALL processes, then it proceeds to its CS, otherwise it will wait. The reply from other process is based on:
    - if other process does not want to enter CS, it replies immediately.
    - if other process is currently in CS, it will defer the reply.
    - if other process is about to enter its CS, it will compare its clock v/s the timestamp in the message.
      - if `its clock ≤ timestamp`, it proceeds to CS and defer replying.
      - if `its clock > timestamp`, it replies immediately to the requestor and defer going to CS.
  - after completing CS, it sends replies to all deferred processes waiting.

Message Count: `2*(n-1)` messages per CS entry.

Example:



2.2.2. Token Passing

How it works:

- A process can enter CS if it has the Token.
- Once done, the Token is passed to next process.

Potential issues:

- Lost token –> Re-election.
- Failed process –> form a new logical ring.

3. Atomicity

- Ensures all operations of a program unit are executed until completion.
- **Requires a Transaction Coordinator**.
- Using two-phase commit protocol.
  - Phase 1: Coordinator sends request for prepare to all sites and wait for responses from ALL sites. If timeout or one "aborts", the whole transactions will be aborted.
  - Phase 2: Coordinator makes a decision to commit or abort and inform all sites for their local recording. ACK is required from each site.
  - If Coordinator dies: sites decide the state of T base on their log.
  - If Site dies: it looks at it logs to recover (commit->redo, abort->undo, ready->ask C)

4. Concurrency Control

Uses **locking protocols** & **timestamp** to ensure multiple atomic transactions can execute in some "serial

order", but still allow concurrent execution.

4.1. Locking Protocols

- Single Coordinator Approach: single site is acting as the **central Coordinator**, all lock/unlock request go thru this site.
  - Pros: Simple implementation, simple deadlock handling.
  - Cons: possible bottleneck, SPOF.
- Multiple Coordinator Approach: **distribute coordinator to multiple sites**, each handle different sets of resources.
  - Pros: reduces bottleneck.
  - Cons: multisites complicate deadlock handling.
- Majority Protocol Approach: **Lock Manager at each site**, trx sends lock request to more than half of sites where the data is stored, when majority reply, lock is obtained.
  - Pros: avoid central control, SPOF
  - Cons: complicated to implement, 2(n/2+1) messages for lock req, and n/2+1 for unlock req.
- Biased Protocol Approach: **Lock Manager at each site**. Read: shared locks; Write: exclusive locks. Shared locks priority > exclusive locks.
  - Pros: less overhead on read, but additional overhead on writes.
  - Cons: complex deadlock handling.
- Primary Copy Approach: one of the sites chosen as Primary Site, for handling request to lock.
  - Pros: simple implementation.
  - Cons: if primary fails, data item is unavailable.

4.2. Timestamp

Use to decide the serialization order. Site advances it logical clock if timestamp greater than current logical clock.

5. Deadlock Handling

Deal with 3 areas: deadlock prevention, deadlock avoidance, and deadlock detection.

5.1. Deadlock Prevention

Main idea is to **order the resource**, to prevent Circular Wait to happen, using timestamp and/or priority.

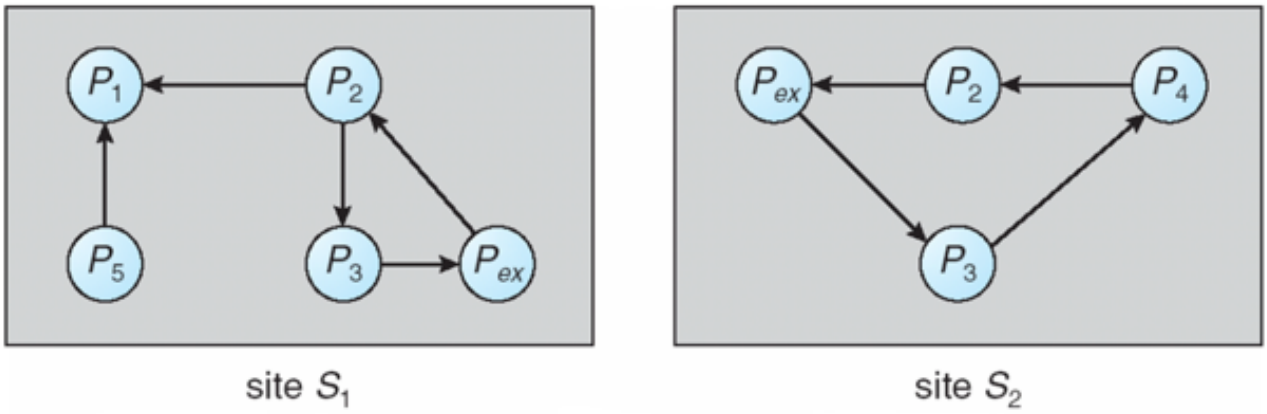
- **Wait-Die Non-Preemptive Scheme**: when a new resource request comes up
  - **older process** can wait for younger process to release its resource.

    - **younger process** will not wait for older process and thus rollback. Hence in this scheme the younger ones may tend to roll back more.
- **Wound-Wait Preemptive Scheme**: when a new resource request comes up
    - **older process** will preempt the younger one for resource, forcing it to roll-back.
    - **younger process** will wait for older process.

5.2. Deadlock Detection

> Using Wait-For Graph, both locally and remotely.

- **Centralized Approach**: uses a central deadlock-detection Coordinator maintaining global wait-for graphs.
    - Cons: as a result of false cycles (unsync arrival of msg), unecessary rollbacks may occur.
- **Fully Distributed Approach**: uses an algorithm developed by Chandy-Misra-Haas to circulate a probe message through out the systems with its process ID in the blocked field. If the originator of the message gets the message (after it was circulated) and sees its process ID in the blocked field, a cycle must have appeared and the deadlock exists. It may commit suicide or use some algorithm to resolve this situation.



- Both sites discovers cycle in local Wait-For Graph:
    - Site S1: *Pex* → *P2* → *P3* → *Pex*
        - The edge with *Pex* is *P3* → *Pex* → *P2* = ID(*P3*) > ID(*P2*)
    - Site S2: *Pex* → *P3* → *P4* → *P2* → *Pex*
        - The edge with *Pex* is *P2* → *Pex* → *P3* = ID(*P2*) < ID(*P3*). Send Deadlock-Detection Message.

6. Election Algorithm

> Election is used when the Coordinator fails. It promote active process with highest priority to become Coordinator.
> There are 2 Election algorithm:

- **Bully Algorithm**: process P discovers Coordinator is dead (no response after T time), and tries to elect

itself to become the new coordinator. It sends messages to all higher numbers processes, if no response, it will forces all process with lower number to let it become the coordinator process, even there is already an active coordinator with lower number.

- **Ring Algorithm**: circulate the active process (with priority number) to the right as candidate for election. The largest number in the active list is promoted to be the new Coordinator.

7. Reaching Agreement

> Guidlines for detecting failures due to unreliable communications or faulty processes, with a time-out scheme for exepcting response ACK.

Questions

Qn: Which is false about Happen-Before?
Ans: A->B, B->C, and A has no relationship with C (false statement).