Time, Clocks, and the Ordering of Events in a Distributed System (1978) - Leslie Lamport

Presented by,
Bhargav Sundararajan

The Problem

Which request was made first ?

Request A

Request B

NODE A

NODE B

The Problem

Which request was made first ?

Solution: Physical clocks?

Request A

Request B

12:00 AM

12:01 AM

NODE A

NODE B

Overview

- Partial Ordering
- Total Ordering
- Anomalous Behavior
- Physical clocks
- Conclusion

Partial Ordering

- The system is composed of a collection of processes
- Each process consists of a sequence of events (instructions/subprogram)

Process P :
```
instr1
instr2
instr3 ... (Total Order)
```

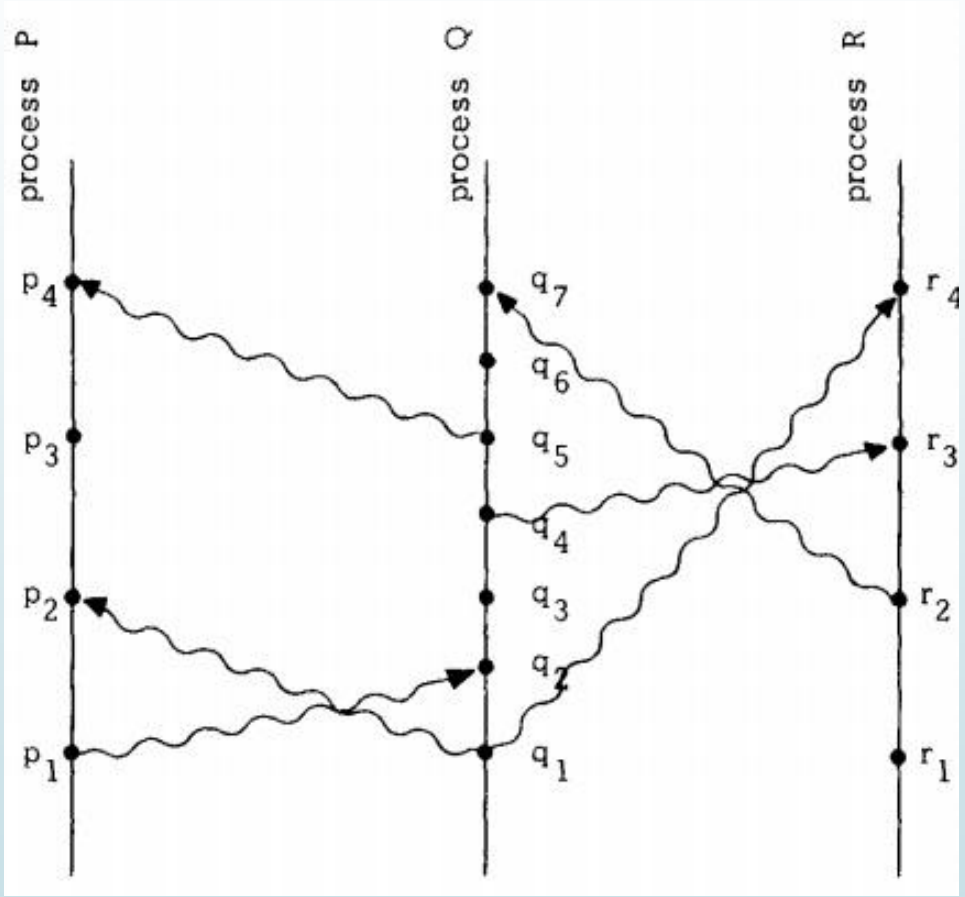
- 'Sending' and 'Receiving' messages among processes
  - 'Send' : an event
  - 'Receive' : an event

‘Happened Before’ relation

- a ‘*happened before*’ b is denoted as ‘$a \rightarrow b$’
- If ‘*a*’ and ‘*b*’ are events in the same process, and ‘*a*’ comes before ‘*b*’, then $a \rightarrow b$.
- If ‘*a*’ sends a message and ‘*b*’ received it, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Two distinct events a and b are said to be *concurrent if a -/->b and b -/->a*

Space-Time Diagram

- Vertical lines represent process, Dots represent events and wavy lines represent messages.
- Horizontal direction represents space.
- Vertical direction represents time.



Logical Clocks

- A clock C_i for each process P_i to be a function which assigns a number $C_i(a)$ *to any* event a in that process.
- Logical Clock (C_i) has no relation with the Physical Clock.

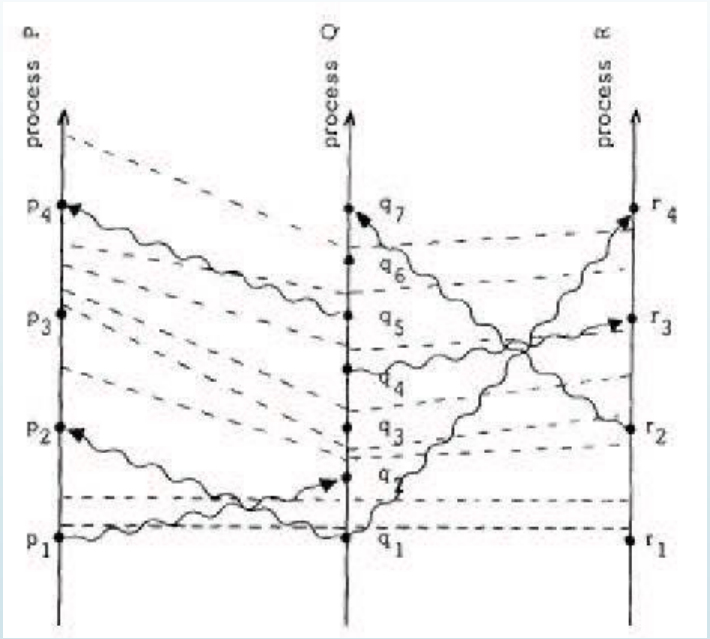
Clock Condition:
For any event a, b:
If a→b, then C(a) < C(b)

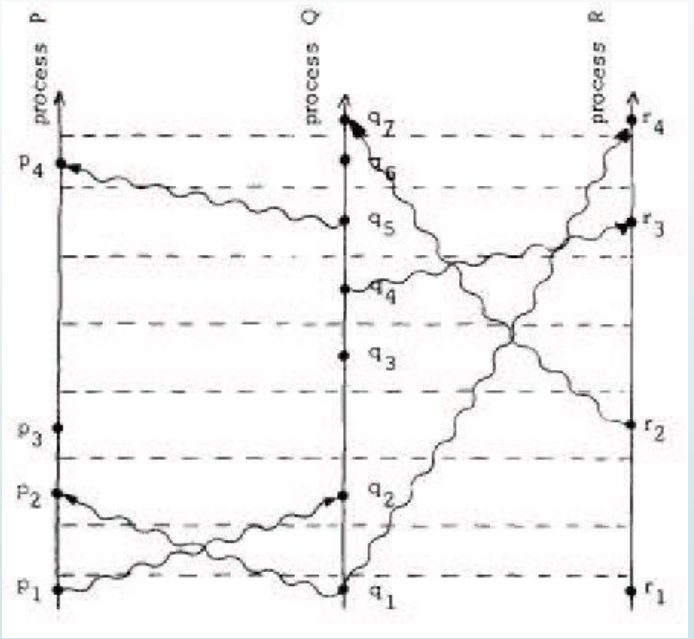
- Clock Condition is satisfied if :

C1. If *a* and *b* are events in process P_i, and a comes before *b*, then $C_i(a) < C_i(b)$.

C2. If *a* is the sending of a message by process P_i and *b* is the receipt of that message by process P_j, then
$C_i(a) < C_j(b)$

Space-Time Diagram





- Dashed lines denotes a clock tick
- The clock tick happens between two events

Implementation rules of Logical Clocks

- Following implementation rules are proposed to satisfy the clock condition:

IR1. Each process P_i increments *Ci* between any two successive events.

IR2. (a) If event a is the sending of a message *m* by process P_i, then the message *m* contains a timestamp
$T_m = C_i(a)$.
(b) Upon receiving a message m, process P_i sets C_i greater than or equal to its present value and greater than T_m.

Total Order of Events

- A system of clocks can be used to order the set of all events in a system
- To break ties, we define a new relation '<' known as the arbitrary total order of events
- The total order relation => is defined as:

a => b, if and only if

(i) $C_i(a) < C_j(b)$

(ii) $C_i(a) = C_j(b)$ *and* $P_i < P_j$

- This new relation, completes the 'happened-before' partial order into a total ordering

Mutual Exclusion Problem

- Multiple processes share the same resource
- Conditions of the problem:

1. A process using the resource must release it before it can be given to another process.

2. Requests for the resource must be granted in the order in which they were made.

3. If every process using the resource eventually releases it, then every request is eventually granted.

Solution to Mutual Exclusion Problem

- Every process maintains its own request queue.
- *The Algorithm:*

1. Resource request:
    a. Process P_i sends the message T_m : P_i requests resource to every other process.
    b. P_i also puts the request message on its request queue.
2. Resource request receipt:
    a. P_j receives P_i's request message.
    b. P_j then puts the message on its request queue
    c. P_j sends an acknowledgement to P_i (timestamped later)
3. Resource release:
    a. P_i removes request message T_m : Pi requests resource from its queue
    b. sends the release message P_i releases resource to every other process.

Solution to Mutual Exclusion Problem

4. Resource release receipt:

    a. P_j receive's P_i's resource release message.

    b. P_j removes the T_m : P_i requests resource from its request queue.

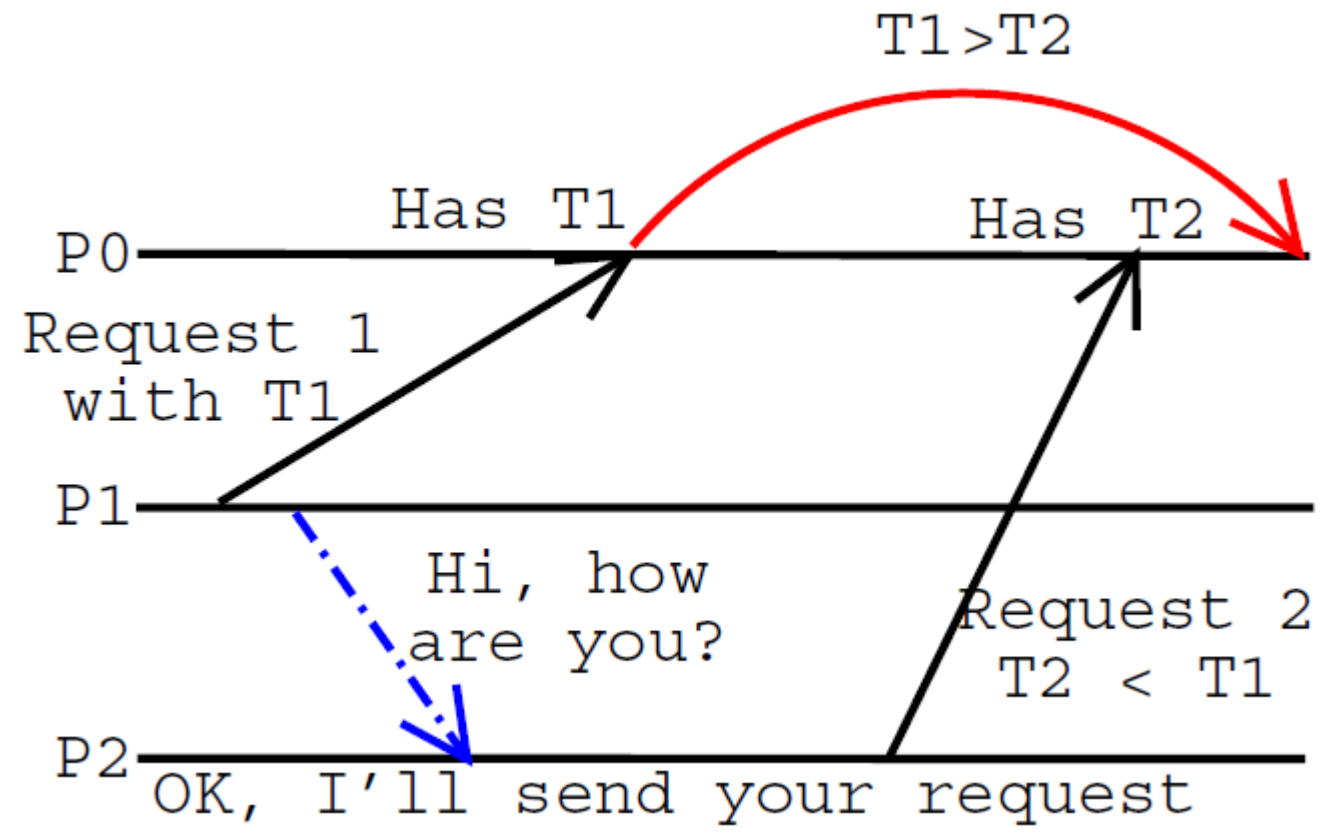
5. Resource allocation:

    P_i is allocated the recourse when,

    a. There is a T_m : P_i requests resource message in P_i 's request queue which is ordered before and other request in the queue.

    b. P_i has received messages from every other process timestamped later than T_m.

Anomalous Behavior



Anomalous Behavior

- Two possible ways to avoid such anomalous behavior:

1. The user can take the responsibility and assign a later timestamp to its own event. For eg. *b* could give itself a later timestamp than *a* before requesting for the resource.
2. Strong Clock Condition:

   Let S be a set of all system events.

   S' : Set that contains S + all external events.

   Then the strong clock condition is that,

   For any events a; b in S' : if a → b then C(a) < C(b)

3. One can construct physical clocks, running quite independently, and having the Strong Clock Condition, therefore eliminating anomalous behavior.

Physical Clocks

- $C_i(t)$ is the reading of the clock at physical time t.
- We assume that the clock is continuous and does not have discrete ticks. Hence, $dC_i(t)/dt \approx 1$. Now, we can assume a following condition such as:

  PC1: $| dC_i(t)/dt - 1 | < \kappa$, **where** $\kappa << 1$
- It is not only enough for the clocks to run at the same rate, we need them to be synchronized as well. There we can assume another condition such as:

  PC2: ***For all i,j :*** $|C_i(t) - C_j(t)| < \varepsilon$, **where** $\varepsilon \approx 0$
- As two clocks do not run at the same rate all the time, the difference tends to get bigger and bigger over time.
- Hence, we need to devise an algorithm such that PC2 always holds.

Physical Clocks

- Let μ be a number such that if *a* and *b* are two processes and $a \rightarrow b$. If a occurs at time *t*, then as *b* occurs after *a*, it should occur at time *t*+μ.
- To avoid anomalous behavior, we need:

$$C_i(t+\mu) - C_j(t) > 0$$

- Combining this with PC1 and 2 allows us to relate the required smallness of κ and ε to the value of μ.
- Using PC2, it is then easy to deduce that $C_i(t+\mu) - C_j(t) > 0$ if the following inequality holds,

$$\varepsilon/(1-\kappa) \leq \mu$$

Algorithm

- Let us assume that there is a minimum delay in transmission of a message and is denoted by μ_m, where $\mu_m \geq 0$
- The Implementation rules IR1 and 2 can be specialized for physical clocks as follows,

IR1' : For each *i*, C_i is differentiable at t and $dC_i(t)/dt > 0$
IR2' : (a) P_i sends a message at *t*, timestamp $T_m = C_i(t)$. (b)Upon receipt of m, P_j sets $C_j(t')$ to $\max(C_j(t'), T_m + \mu_m)$

Conclusion

- Introduced the concept of ‘happens before’ and how it defines an invariant partial ordering of events in a distributed system.
- Described an algorithm for extending that partial ordering to a somewhat arbitrary total ordering.
- Total ordering can sometimes result in an anomalous behavior.
- Prevented this behavior using synchronized physical clocks and devised an algorithm to show the same.