

Time, Clocks and Ordering of Events in Distributed Systems

Leslie Lamport

- Dist s-m -> impossible to have absolute real time to determine exact event order
- Does exact event order matter?
- When does it matter?
- Objective of paper:
 - Establish valid ordering of events
 - Assign logical clocks/timestamps which reflect order
 - Determine properties of physical clocks (drift, rate change) so that still appear synchronous (i.e., events ordered according to these physical clocks are still correctly ordered)

Partial Ordering

- Establish order between causally related events
 - e.g., send msg \rightarrow receive msg
 - e.g., two consecutive events in same sequential process
- Determines a partial order:
 - “ \rightarrow ” – *happens before*
- If $a \not\rightarrow b$ – doesn't mean that a didn't occur before b

Logical Clocks

- What must be satisfied for a set of clocks to be valid logical clocks?

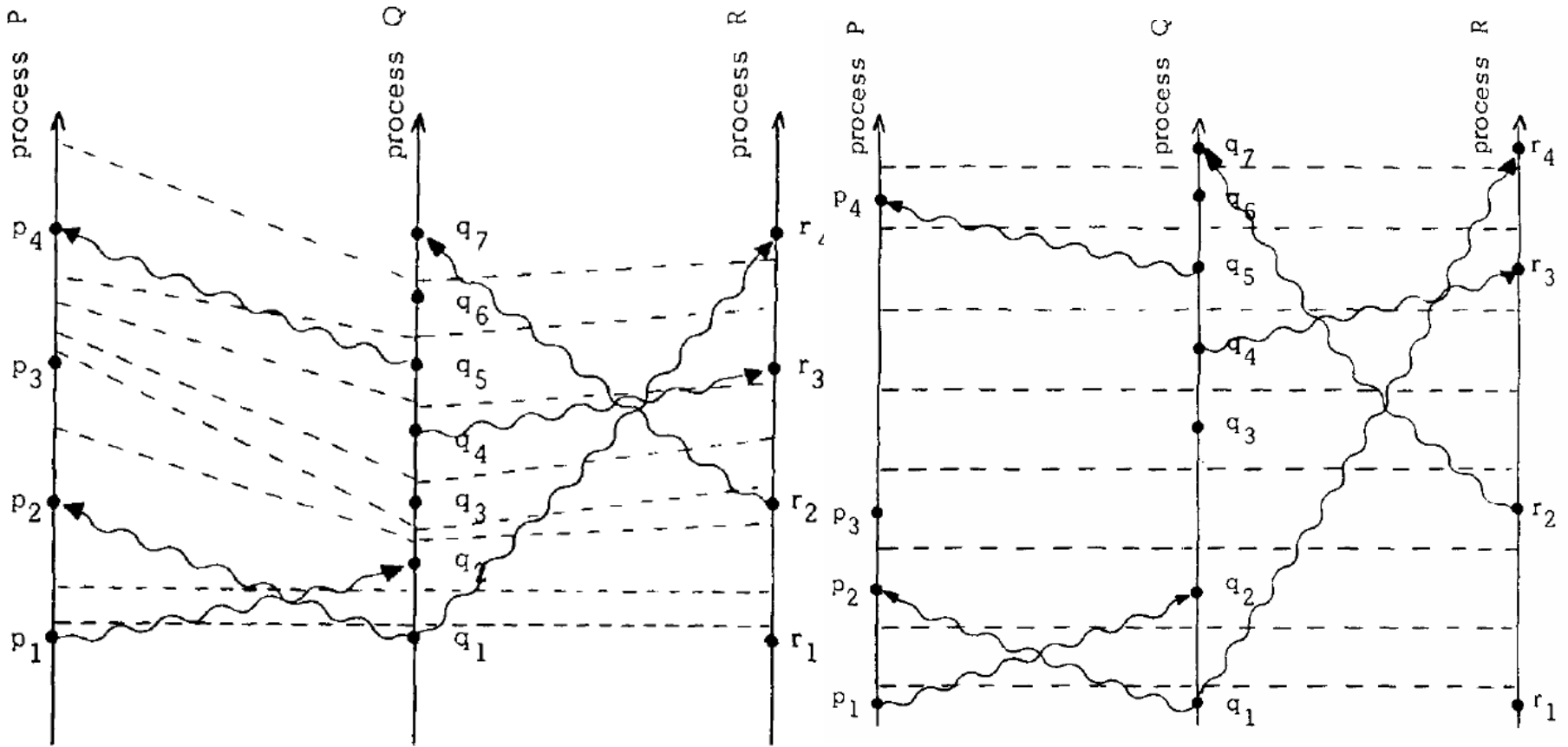
For events a, b : if $a \rightarrow b$ then $C(a) < C(b)$

- If a and b in same process P_i : $C_i(a) < C_i(b)$
- If a in P_i sends msg and P_j rcvs it at events b : $C_i(a) < C_j(b)$

Implementation

- Local clock must be incremented between successive events
- Msg must contain timestamps, and on rcv clock must be updated accordingly

Possible alternatives



From Partial to Total Ordering

- If $C_i(a) = C_j(b)$ arbitrarily break ties \rightarrow total order (" \Rightarrow ") of events is established...
 - a and b not causally related, so it doesn't matter, and there is no way to know exactly which one happened before
 - To establish total ordering (\Rightarrow) assume unique (though arbitrary) ordering of processes $P_i < P_j \dots$

Why is Total Order important:

- Distributed Coordination
- Mutual exclusion example
 - Must give access to resource in order in which requests are made
 - P1 msg to sched P0, P1 msg to P2, then P2 msg to P0, P0 receives P2's msg first
- Distributed implementation:
 - Send timestamped msg to everyone, also put it in your request queue
 - Get timestamped acks from everyone that they are aware of your request
 - When your request is on front of your queue, you have access
 - When done with resource – send timestamp release msg to everyone
 - On release msg everyone removes corresponding request from personal queue
- Assumptions:
 - Ordered, lossless communication!
 - No faulty processors
 - No external communication channels
 - Or explicit update if external comm exists

Condition for Physical Clocks

- Physical clocks may have a drift e and change in rate k
- Lamport establishes relationship between e and k relative to the minimal delay of a message μ : $e \leq \mu - \mu k$

Implementation rules:

- At each process, in absence of msgs C_i is differentiable and slope > 0 (i.e., will increase)
- When msg with timestamp T_m is received at time t' , $C_j = \max(C_j(t'-0), T_m + \mu)$ (i.e., explicit update)
- Msgs must be sent every τ sec, and must have unpredictable delay less than ξ (i.e., how often should we synchronize)