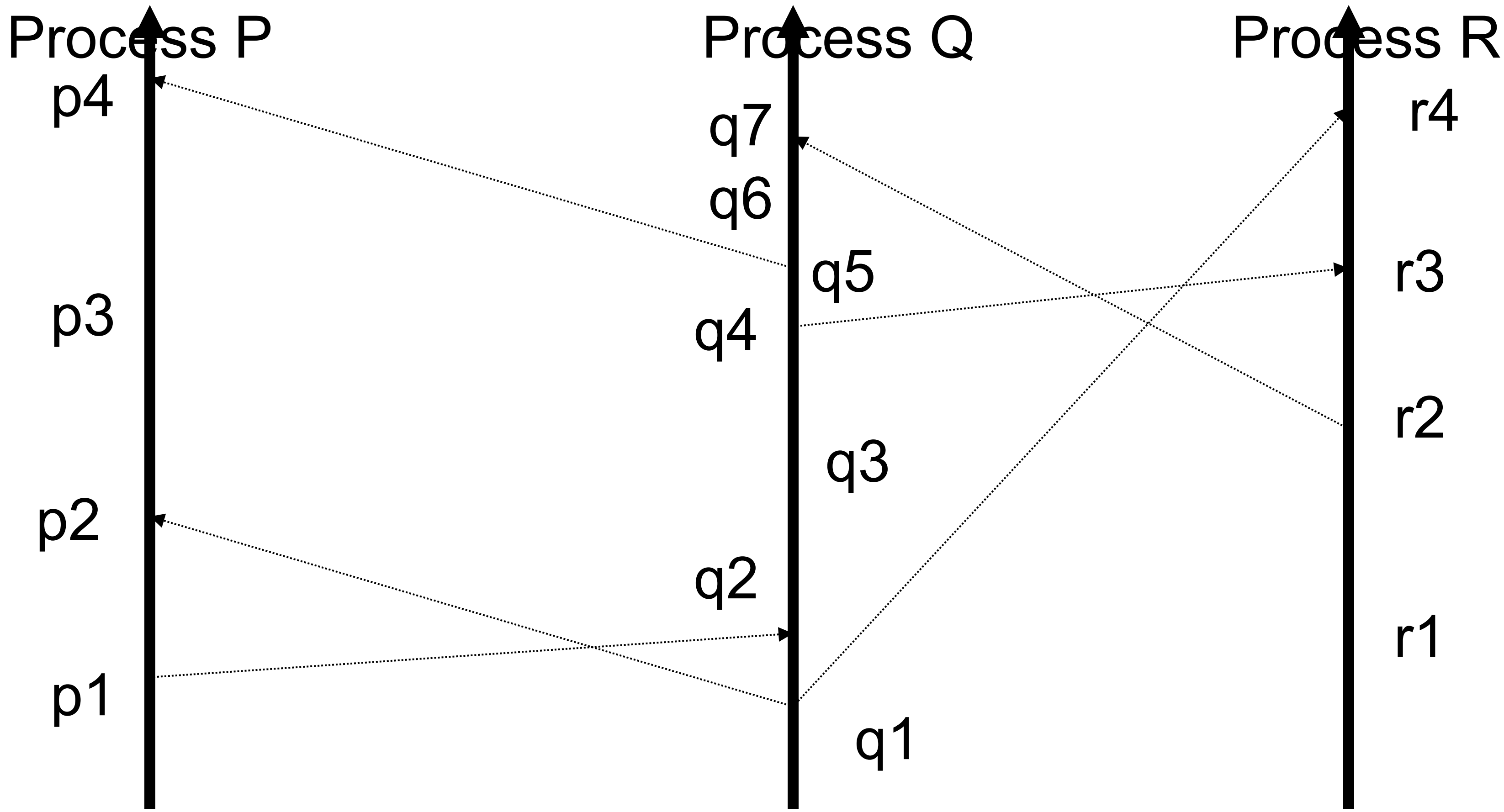
Distributed systems

- **Time, clocks, and the ordering of events in a distributed system Leslie Lamport. Communications of the ACM, 21(7):558-565, July 1978**
  - **Formal description of distributed systems concepts**
- **The ABCDs of Paxos, Butler W. Lampson, PODC 2001**
  - **Consensus among a group of unreliable processors**

Partial ordering

- Assume that the system is made of a number of processes. Each process consists of a sequence of events.
- “happens before” relationship:
  - If a and b are events in the same process, a comes before b, then a happens before b.
  - If a is the sending a message and b is the receipt of it, then a happens before b.
  - If a happens before b and b happens before c, then a happens before c
- If a does not happen before b and b does not happen before a, a and b are concurrent

Partial Ordering



- p3 and q3 are concurrent.

Logical clocks

- Clock is just a way of assigning a number to an event, number is thought of the time at which the event occurred.
- Clock C for each process P is a function that assigns a number Ci<a> to any event a.
- Clock condition:
  - For any event a, b: if a happens before b, then C(a)<C(b)
- Happens before condition holds if:
  - a and b are events in process P, and a comes before b, then C(a) < C(b)
  - a is the sending of a message and b is the receipt then C(a)<C(b)

Implementable clock condition

1. Each process P, increments C between any two successive events

2. If event a is the sending of a message m by process Pi, then the message m contains a timestamp Tm=Ci(a). Upon receiving a message m, process Pj sets Cj greater than or equal to its present value and greater than Tm.

Total ordering of events

- We can use a system of clocks satisfying the clock condition to place a total ordering on the set of all system events. We order events by the times at which they occur. To break ties, we use any arbitrary total ordering of the processes
  - If a is an event in Pi and b is an event in Pj, then $a \Rightarrow b$ iff
    - $Ci(a) < Cj(b)$ or
    - $Ci(a) = Ci(b)$ and $Pi < Pj$
- Total ordering depends on the clocks (C). Partial ordering is absolute

Application

- Algorithm for granting a resource which satisfies:
  1. A process which has been granted the resource must release it before it can be granted to another process
  2. Different requests for the resource must be granted in the order in which they are made
  3. If event process which is granted the resource eventually releases it, then every request is eventually granted

Central server based approaches that use the time received to grant resources does not work if two request take different times to reach the service

Physical clocks

- To synchronize clocks:
  - Sender sends message with time stamp
  - Receiver receives responses. The difference in expected and unexpected delay is the clock drift.

- They derive a bound on time taken to synchronize clocks.