Lamport’s Clocks

David Eckhardt
Bruce Maggs

Original Document

Operating Systems

R. Stockton Gaines
Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

Communications of the ACM

July 1978
Volume 21
Number 7

15-410 Gratuitous Quote of the Day

"There have been members of the Maggs family in south east Suffolk since the great subsidy of 1327 but they were of no great distinction either then or afterwards."

-- from Allan Farquar Bottomley, "Introduction," in the Southwold Diary of James Maggs, 1818-1876, edited by Allan Farquar Bottomley, Volume I - 1818-1848, (Suffolk: Published for the Suffolk Records Society by the Boydell Press, 1983), p.1.

Life Made Simple

- Global clocks simplify protocol design.
- E.g., first-come first-serve resource allocation.
- Bruce: My watch is synchronized to the U.S. atomic clock!

Timing is Everything

- “Time is relative, or did I misread Einstein?” - Dan Bern (irreverent songwriter)
- Even in one inertial reference frame, can’t built an “arbiter”.

Which button was pressed first?

Distributed System

- A collection of processes that exchange messages.
- A process consists of a sequence of events.
- Sending and receiving messages are two types of events.

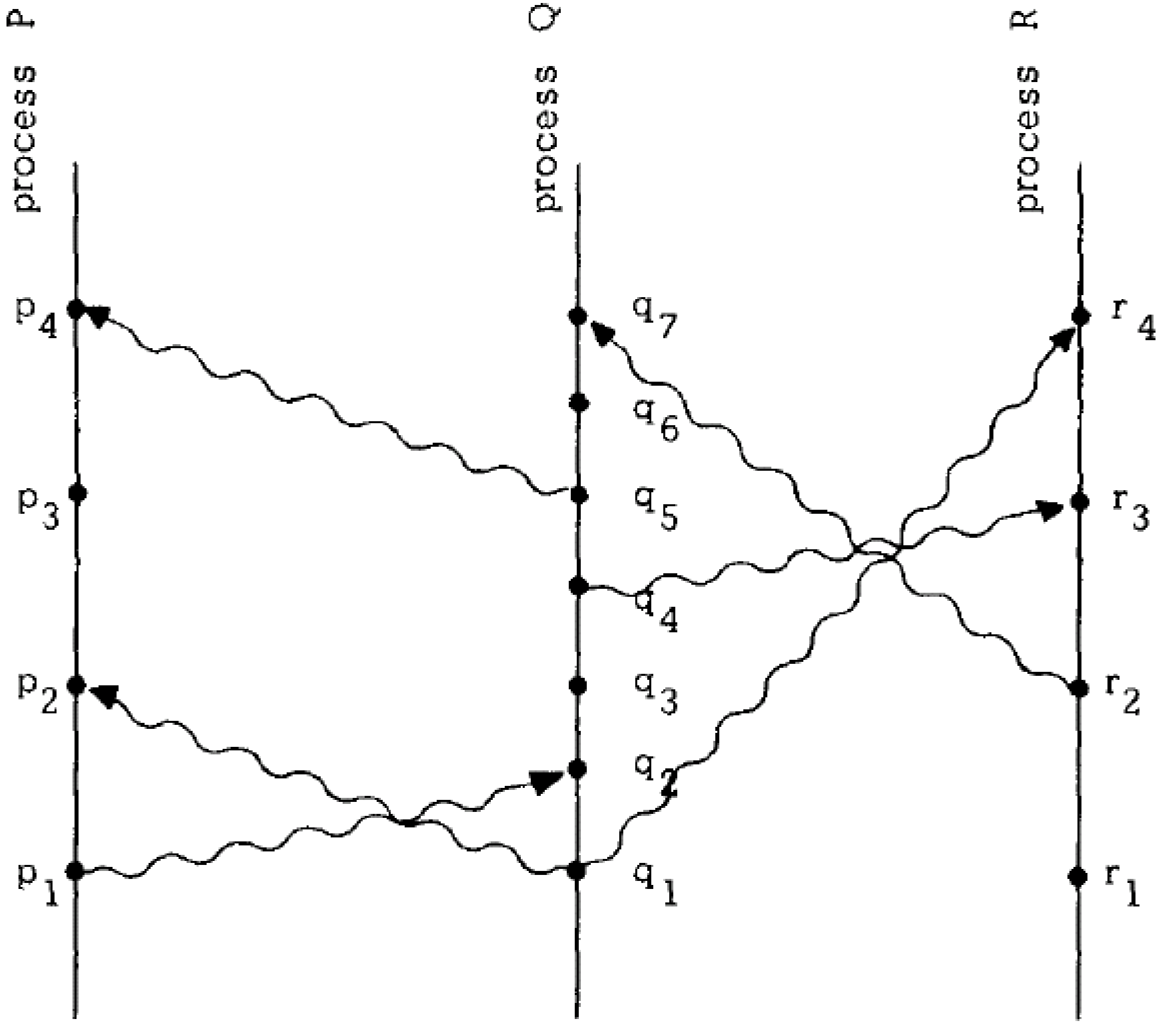
Happened Before ($\rightarrow$) Partial Order

1. If a and b are events in the same process and a occurs before b, then $a \rightarrow b$
2. If a is the sending of a message by one process and b is the receiving of a message by another, then $a \rightarrow b$
3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Concurrency

- Two events a and b are said to be concurrent if a $\not\to$ b and b $\not\to$ a

Space-Time Diagram



Clock Condition

- If a $\rightarrow$ b then C(a) < C(b).

- Notice that converse cannot hold: p_2, p_3, and q_3 are concurrent in space-time diagram, would all have to happen at same time. But $p_2 \rightarrow p_3$.

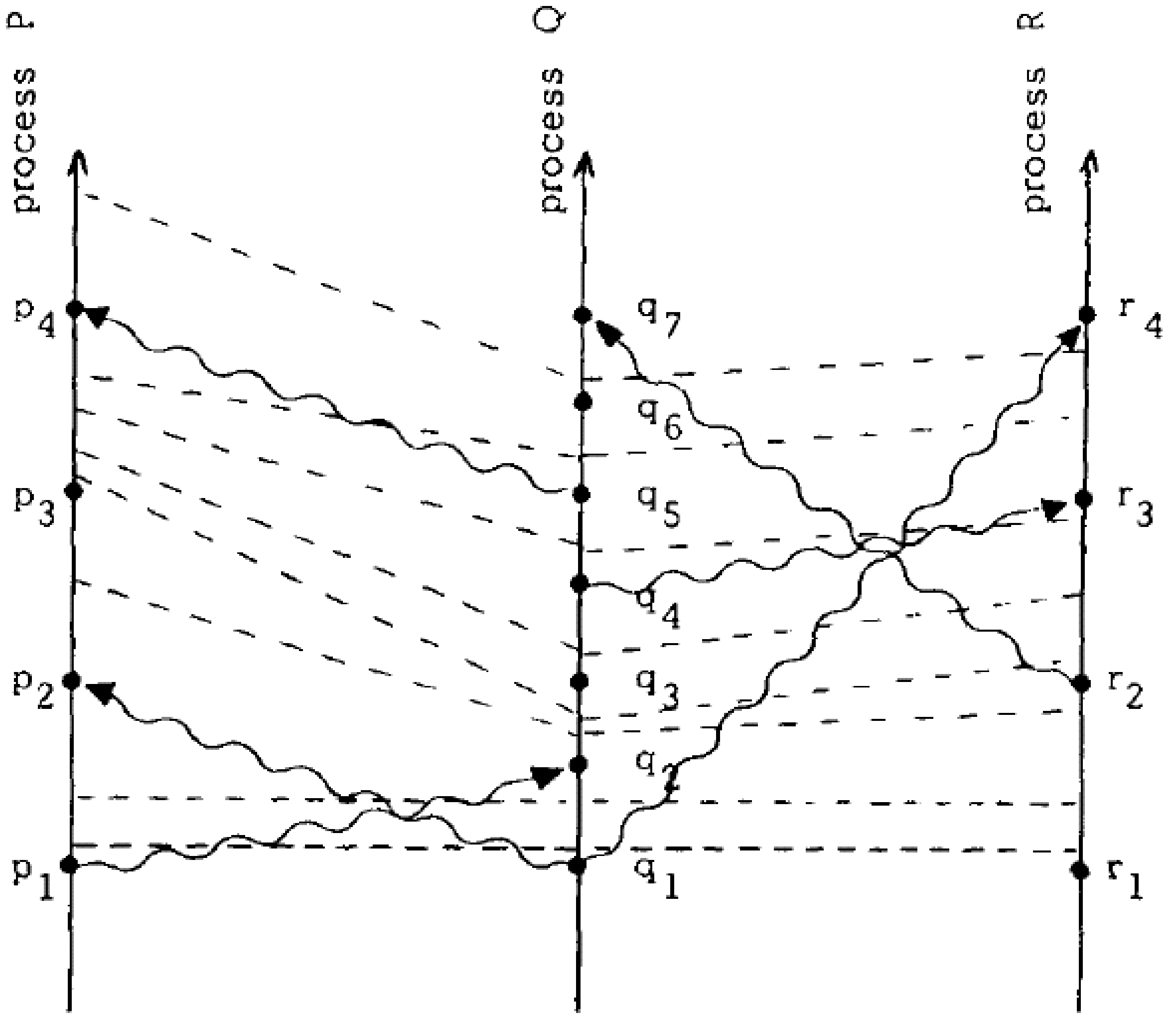
Local Clocks

- Counter C_i at process P_i
- $C_i(a)$ is value of C_i when a occurs at P_i
- $C(a) = C_i(a)$ if a occurs at process P_i

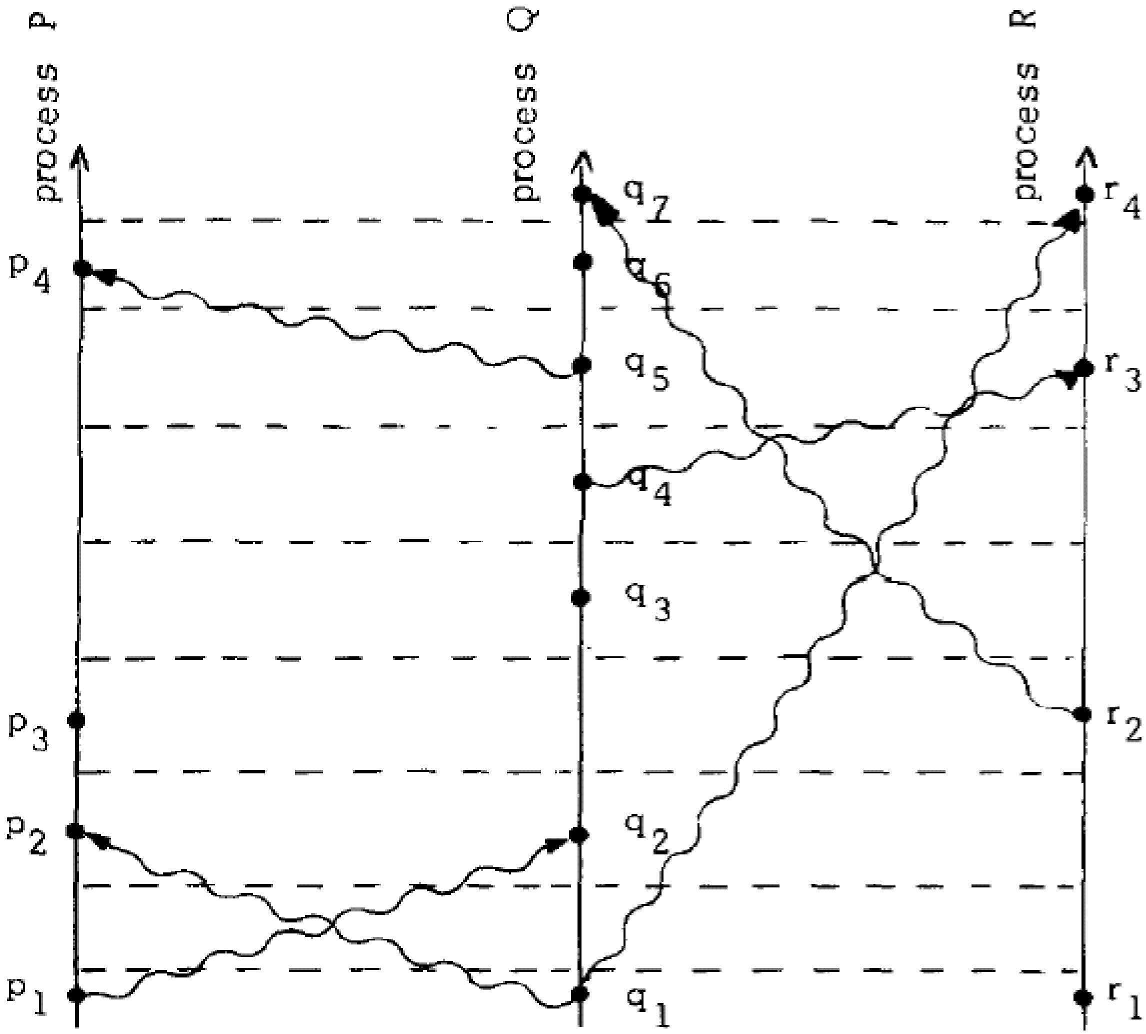
Sufficient Subconditions

1. If a and b are events in P_i and a comes before b, then $C_i(a) < C_i(b)$.
2. If a is the sending of a message from P_i and b is the receiving of the message by P_j, then $C_i(a) < C_j(b)$.

Clock “Ticks”


process P
process Q
process R
p_1
p_2
p_3
p_4
q_1
q_2
q_3
q_4
q_5
q_6
q_7
r_1
r_2
r_3
r_4

Redrawing


process P
process Q
process R
p_4
p_3
p_2
p_1
q_7
q_6
q_5
q_4
q_3
q_2
q_1
r_4
r_3
r_2
r_1

15-410 Gratuitous Quote of the Day

- "First, computer software and hardware are the most complex and rapidly developing intellectual creations of modem man." - p. iii, Internet and Computer Law, P. B. Maggs, J. T. Soma, and J. A. Sprowl, 2001

Timestamps

- When process P_i sends a message, it attaches a *timestamp* T_m.

Implementation Rules

- Each process P_i increments C_i between any two successive events.
- Regarding messages,
  - (a) if event a is the sending of a message m by process P_i, then m contains timestamp $T_m = C_i(a)$.
  - (b) upon receiving m, process P_j sets C_j greater than or equal to its own value and greater than T_m

Total Ordering

- If a is an event in P_i and b in P_j, then a $\Rightarrow$ b if and only if either
  - $C_i(a) < C_j(b)$ or
  - $C_i(a) = C_j(b)$ and $P_i < P_j$
- (Assume an ordering on the processes.)

Mutual Exclusion Example Goals

- A process that has been granted a resource must release it before it can be granted to another process.
- Different requests for the resource must be granted in the order (with respect to $\rightarrow$) in which they are made.
- If every processes that is granted a resource eventually releases it, then every request is eventually granted.

Centralized Scheduling Fails!

- P_1 issues a resource request to P_0.
- P_1 tells P_2, “I just issued a resource request.”
- P_2 receives the message.
- P_2 issues a resource request to P_0.
- P_2’s message arrives first, P_0 grants request to P_2.

Flooding Algorithm

- Broadcast every request to every process.
- Assume all messages reach their destinations.
- Assume in-order delivery of messages.

Algorithm

- To request resource, P_i sends "T_m:P_i requests resource" to all other processes, puts request on local queue.
- On receiving "T_m:P_i requests resource", P_j puts on queue and sends timestamped acknowledgement
- To release resource, P_i removes from queue and sends "P_i releases resource" to all other processes
- When P_j receives "P_i releases resource", removes P_i requests from queue
- P_i granted resource when T_m:P_i request in queue is ordered by $\Rightarrow$ before any other request, and P_i has received a message with time stamp larger than T_m from all others

State Machines

- All processes can simulate identical state machines.
- Inputs are ordered resource requests and releases.
- State indicates which process (if any) has resource.