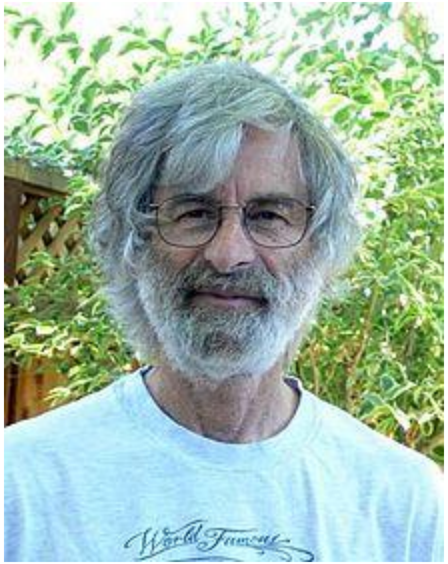
Time, Clocks, and the Ordering of Events in a Distributed System

Ann Zhang
October 8, 2024

Authors



- Leslie Lamport 1978
  - “A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable”
  - Received 2013 Turing Award for “fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency”
  - Was a researcher at Massachusetts Computer Associates at the time of this paper’s publication, later spent time at SRI International, Compaq, Microsoft Research

Problem/motivation

- There is no real notion of time in a distributed system
- There is only some observable ordering
- This makes consistency in distributed systems hard

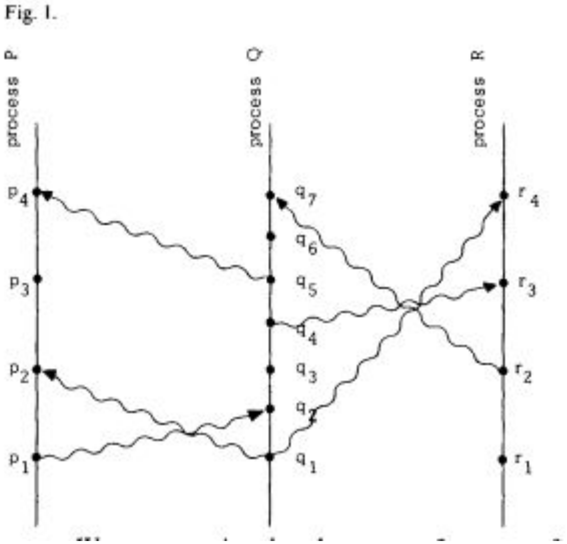
Outline

- Partial ordering
- Logical clocks
- Total ordering
- Anomalous behavior and physical clocks

Partial ordering

- Events in a single process are ordered
- ‘Happened before’ relation ‘→’ defined as:
  - If *a* and *b* are events in the same process and *a* comes before *b*, then $a \rightarrow b$
  - If *a* is the sending of a message by one process and *b* is the receipt by another process, then $a \rightarrow b$
  - If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Gives a partial ordering of events

Partial ordering



- Horizontal direction represents space, vertical represents time
- $p_1 \rightarrow r_4$
- p_3 and q_3 are concurrent

Outline

- Partial ordering
- Logical clocks
- Total ordering
- Anomalous behavior and physical clocks

Logical clocks

- $C(a)$: Mapping from event a to time $C(a)$
- C_i for process P_i
- Clock condition: For any events a, b, if $a \rightarrow b$ then $C(a) < C(b)$
  - Note that the converse is not necessarily true
- Equivalently:
  - If a and b are events in process P_i and a comes before b, then $C_i(a) < C_i(b)$
  - 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 $C_i(a) < C_j(b)$

Logical clocks

Fig. 2.

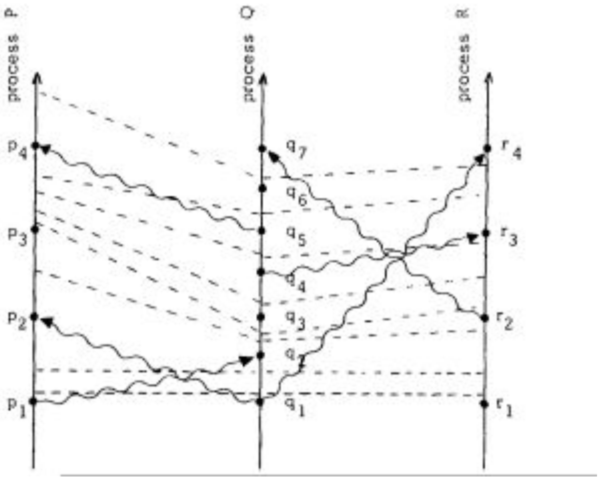
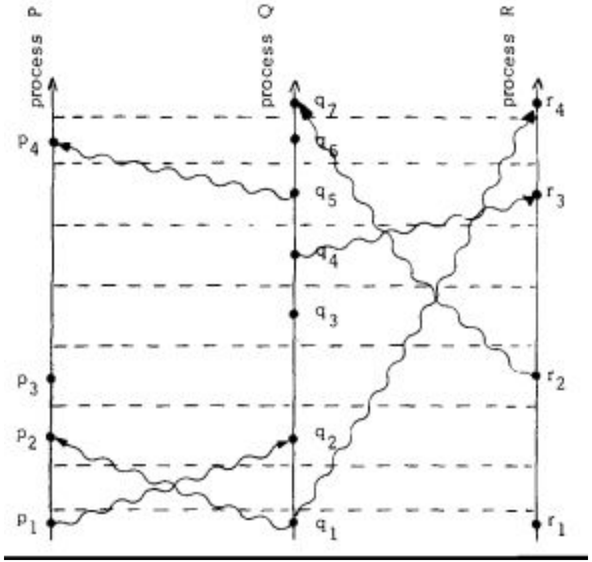


Fig. 3.



Logical clocks

- When should we increment the clock so that it follows the given invariant?
  - For each process, increment local clock every time an event occurs
  - When a message is received, set clock to some value which is at least max(message sent timestamp+1, current clock value)

Outline

- Partial ordering
- Logical clocks
- Total ordering
- Anomalous behavior and physical clocks

Total ordering

- Can construct total ordering of events from system of logical clocks
- It is possible to have ties; break these by arbitrary process ordering
- Formally, total ordering relation '⇒' given by:
  - If *a* is an event in process P_i and *b* is an event in process P_j, then $a \Rightarrow b$ iff
    - $C_i(a) < C_j(b)$, or
    - $C_i(a) = C_j(b)$ and priority of P_i is higher than priority of P_j

Total ordering

- Total ordering is useful for synchronization in a distributed system
- Example: processes which share a single resource
  - Requirements:
    - A process which has been granted the resource must release it before another process can have it
    - Requests are granted in the order they are made
    - If every process which is granted the resource eventually releases it, then every request is eventually granted

Mutual exclusion

- Process P_i sends message T_m:P_i *requests resource* to every other process and puts it on its request queue

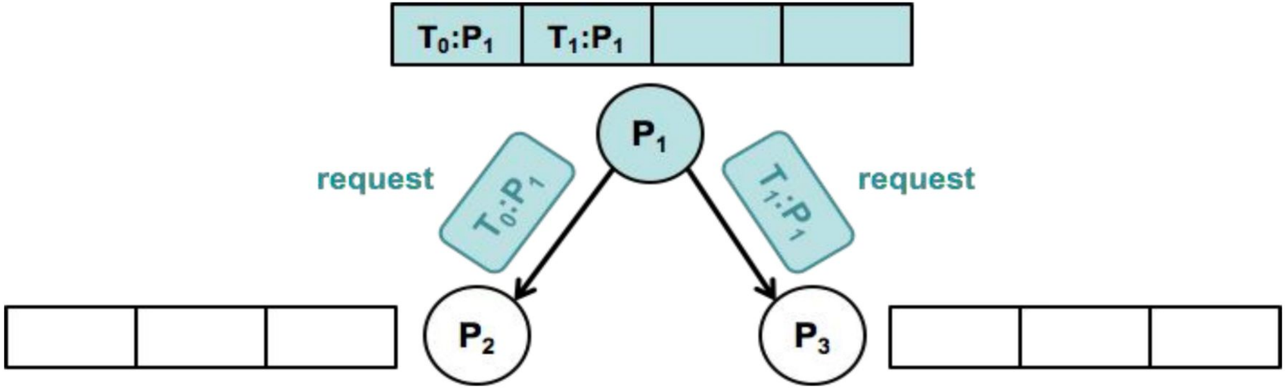


Image taken from CS 6410 fall 2017 slides

Mutual exclusion

- When process P_j receives a *request resource* message, it adds it to its request queue and sends an ack

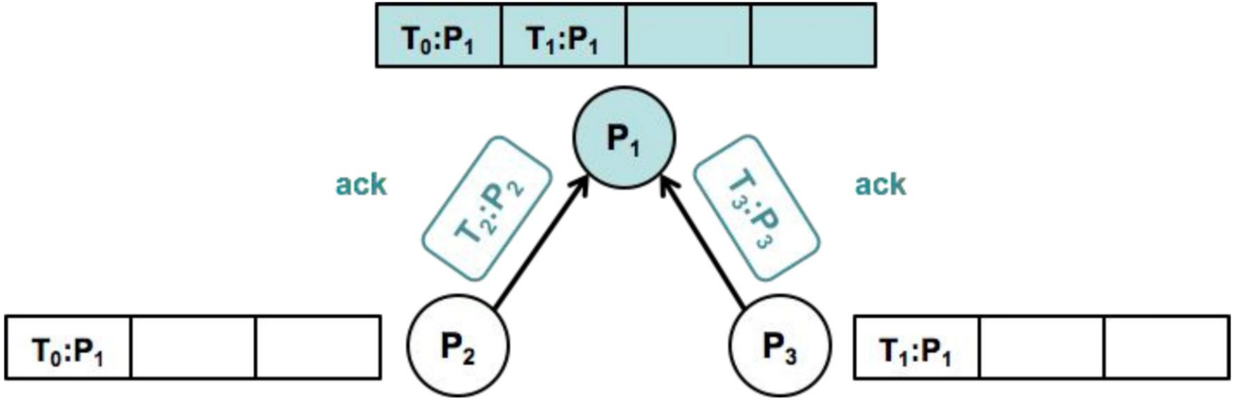


Image taken from CS 6410 fall 2017 slides

Mutual exclusion

- To release a resource, P_i removes its own *request resource* messages from its queue and sends a P_i *releases resource* message to every other process

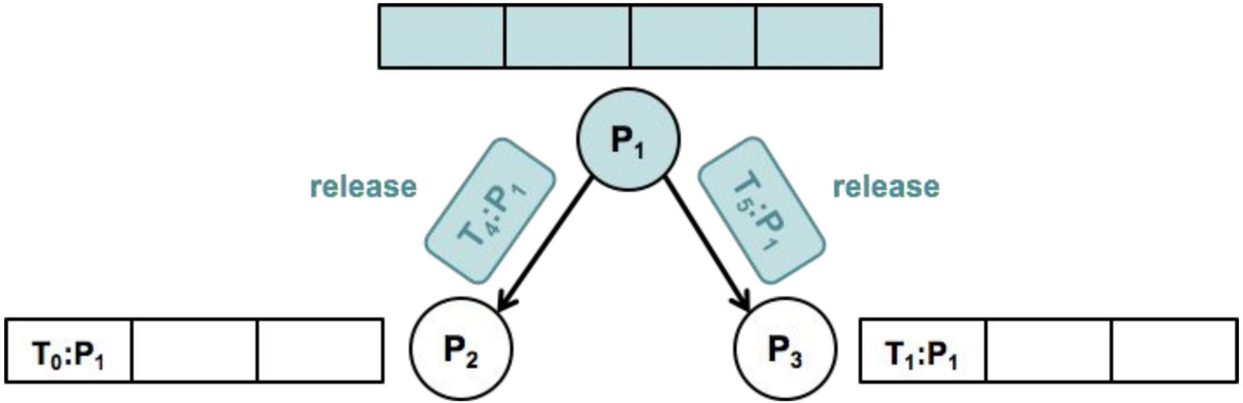


Image taken from CS 6410 fall 2017 slides

Mutual exclusion

- When process P_j receives a P_i *releases resource* message, removes any P_i *requests resource* messages from request queue

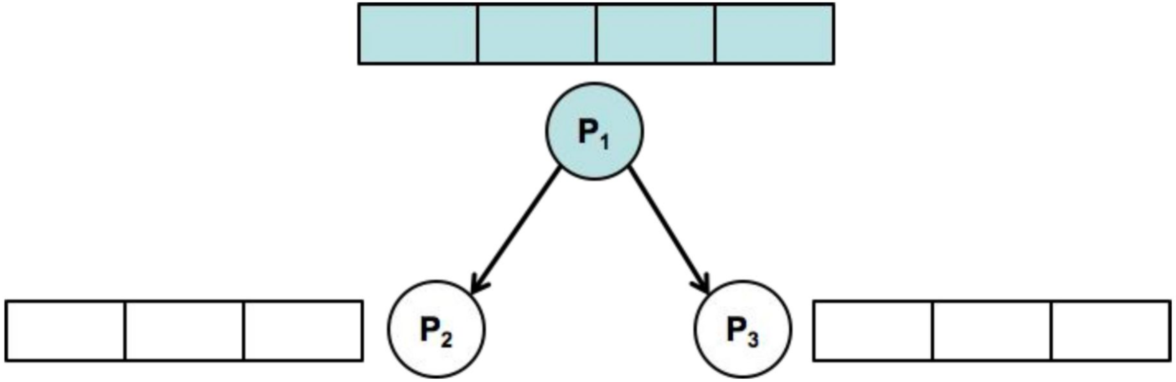


Image taken from CS 6410 fall 2017 slides

Mutual exclusion

- If each process independently follows this protocol, correctness is achieved
- How generalizable is this?
- What are potential problems/limitations of this approach?
    - What happens if there's a failure?

Outline

- Partial ordering
- Logical clocks
- Total ordering
- Anomalous behavior and physical clocks

Anomalous behavior and physical clocks

- Anomalous behavior can occur if information which is not observable by the system governs user-perceived ordering of events
- This can be solved by either:
    - Introducing this information into the system, or
    - Using physical clocks
- If we are relying on physical clocks, they must:
    - Individually run at approximately the correct rate
    - Be synchronized so that all clocks report the same time within some epsilon

Why is this important?

- Notion of time in a distributed system
- State machine replication
- Your thoughts?

Related work

- State machine replication (Fred Schneider)
- Distributed snapshots (Chandy & Lamport 1985)
- Paxos
- Cool but not comprehensive tool:
    - https://www.connectedpapers.com/main/593619c2a69391454eae1f5ebe75fb8fc7e77e9d/graph?utm_source=share_popup&utm_medium=copy_link&utm_campaign=share_graph