15-712:
Advanced Operating Systems & Distributed Systems

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

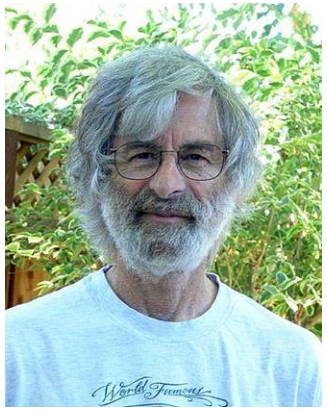
Prof. Phillip Gibbons

Spring 2020, Lecture 4

Today's Reminders / Announcements

- **Andrew has graded one Summary from each of you**
  - Will soon post a few model Summaries
- **I'm still working on the semester schedule…**
  - Our week of Lamport "fun" continues on Friday, with HoF paper on Distributed Snapshots

"Time, Clocks, and the Ordering of Events in a Distributed System"
Leslie Lamport 1978

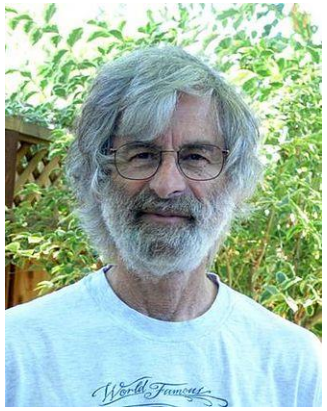


- **ACM Turing Award 2013:**
  - "For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency."

- **IEEE John von Neumann Award 2008**
  - Other winners: Brooks, Lampson
- **Dijkstra Prize (test-of-time award for distributed computing) in 2000 (this paper), 2005, and 2014**

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

Leslie Lamport 1978



SigOps HoF citation (2007):

Perhaps the first true “distributed systems” paper, it introduced the concept of “causal ordering”, which turned out to be useful in many settings. The paper proposed the mechanism it called “logical clocks”, but everyone now calls these “Lamport clocks.”

Happened Before

"A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process."

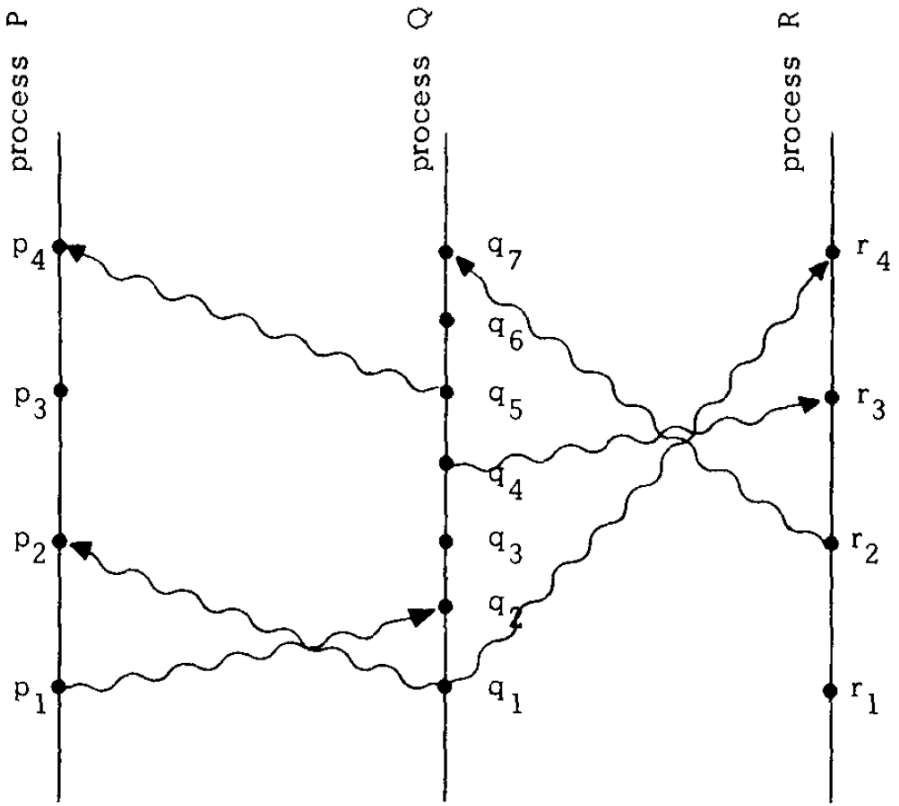
- **"Happened before" is only a partial ordering of events**

- **Must define without using physical clocks. Why?**
  - System specification may not include real clocks
  - Real clocks do not keep precise physical time (clock skew)

Happened Before Definition

The smallest relation satisfying:

- Two events on same process are ordered
- Message receipt ordered after associated message send
- Transitivity: $a \rightarrow b$ and $b \rightarrow c$ implies $a \rightarrow c$



Logical Clocks (aka. Lamport Clocks)

Clock Condition: If $a \rightarrow b$ then Clock(a) < Clock(b)

- **Satisfied if two conditions hold:**
  - **C1:** If a and b are events in process P_i, and a comes before b, then $Clock_i(a) < Clock_i(b)$
  - **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 $Clock_i(a) < Clock_j(b)$

- **An implementation using timestamps:**
  - **IR1:** Each process P_i increments C_i between any two successive events
  - **IR2:** (i) Send $T_m = C_i\langle a \rangle$ as part of the event a's message m
    (ii) Upon receiving that message, P_j sets its C_j to be ≥ its present value and > T_m

Total Order of the Events

- **Order events by the Lamport clock values; Breaking ties arbitrarily (e.g., using process ids)**
  - Fairness issues in breaking ties…

Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

Use in Distributed Mutual Exclusion

- **Goals:**
  I. Must release granted resource before can be granted again
  II. Grant resources in order they are made
  III. Every request is eventually granted
  (assuming no process fails to release a granted resource)

- **Straightforward centralized solution fails**

<P_0 is granting process. Draw figure: P_1 sends request to P_0, P_1 sends message to P_2, P_2 receives message then sends request to P_0, but P_2's request arrives at P_0 before P_1's request. Granting P_2's request first violates (II)>

- **Assume one resource, initially granted to one process, in-order delivery of messages from P_i to P_j, all messages eventually received**

Use in Distributed Mutual Exclusion

- **Request resource:** P_i **sends** T_m: P_i requests resource **to every other process & puts in its local request queue**

- **When receive** T_m: P_i requests resource**, place it on local request queue & send timestamped ack to** P_i

- **Release resource:** P_i **removes any** T_m: P_i requests resource **message from local queue & sends timestamped** P_i releases resource **message to every other process**

- **When receive** P_i releases resource **message, remove any** T_m: P_i requests resource **message from local queue**

- P_i **is granted the resource when (i)** T_m: P_i requests resource **in local queue is ordered (by total order) before any other request in local queue and (ii)** P_i **has received a message from every other process that is timestamped LATER than** T_m

Problem & Generalization

- **Problem: System halts if one process fails**
  - With logical time, no way to distinguish a failed process from a paused/delayed/slow process

- **Generalization: Works for any synchronization that can be specified in terms of a State Machine $\langle C, S, e: C \times S \rightarrow S \rangle$**
  - E.g., C is all possible requests/releases resource commands,
    S is the queue of waiting request commands,
    e is the transition function
  - Run same basic algorithm: A process can execute a command timestamped T when it has learned of all commands issued by all other processes with timestamps ≤ T

Anomalous Behavior

- **With respect to out-of-band communication**
  - Issue request A, call friend to have him issue request B
  - Yet B can get lower timestamp than A

- **Strong Clock Condition: $a \Rightarrow b$ implies $C(a) < C(b)$, where $\Rightarrow$ denotes happened-before when also include out-of-band events**

"One of the mysteries of the universe is that it is possible to construct a system of physical clocks which, running quite independently of each one another, will satisfy the Strong Clock Condition."

Discussion: Summary Question #2

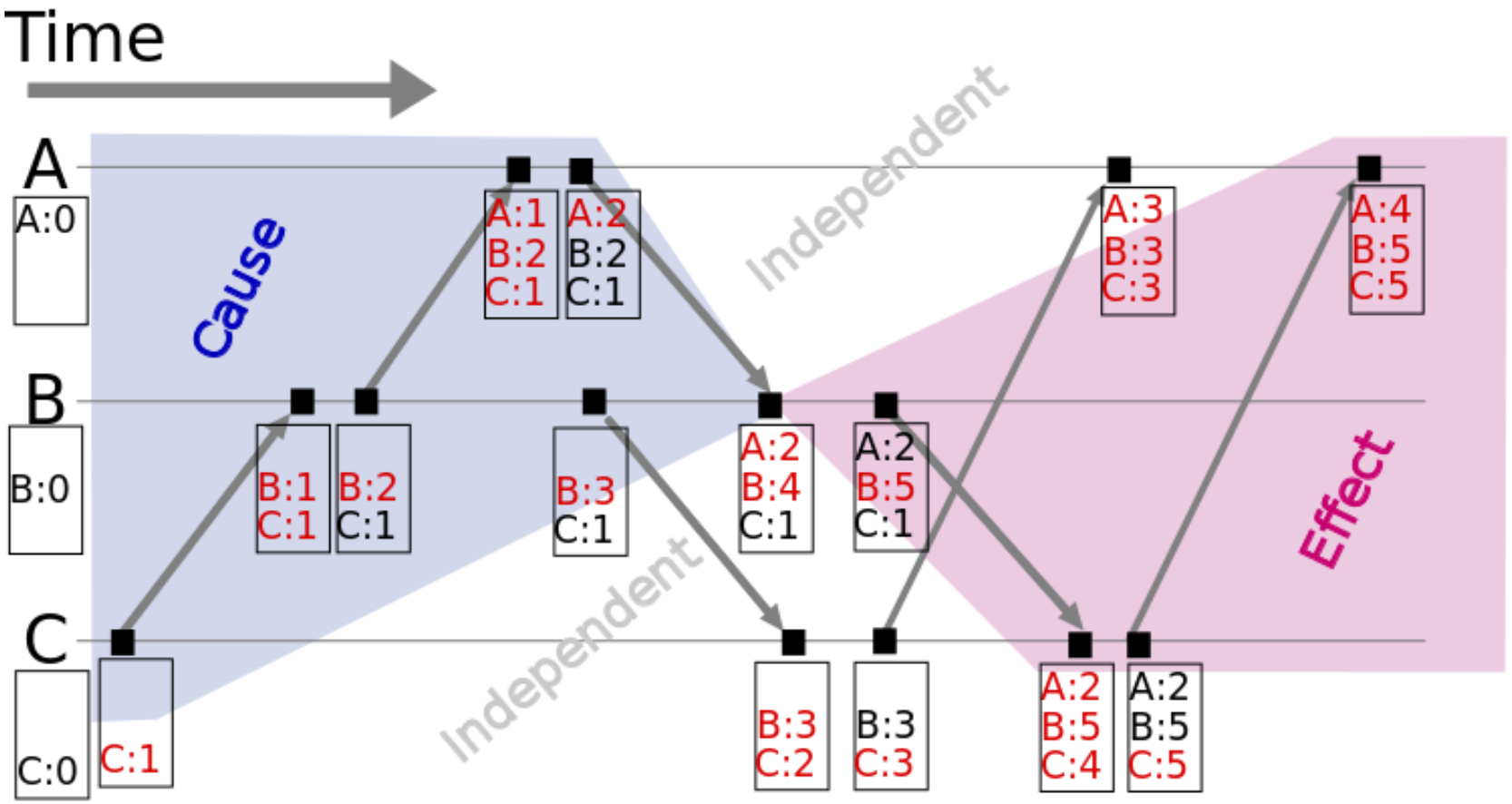
- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Aside: Vector Clocks

- **Each local clock is a vector of N values for N processes**
- **P_i increments i'th value of local clock on internal event**
- **Include entire vector clock when send message**
- **When P_j receives a message with clock V:**
  - Increment j'th value of local clock
  - Set local clock to be elementwise max of its local clock and V

Vector Clocks

On Receive of V: $V_j[j]$++; V_j = elementwise-max(V, V_j)



V < V′ if ≤ on each element and < on at least one element

Vector Clocks satisfy Clock Condition?

Clock Condition: If $a \rightarrow b$ then Clock(a) < Clock(b)

- **Satisfied if two conditions hold:**
  - **C1:** If a and b are events in process P_i, and a comes before b, then $Clock_i(a) < Clock_i(b)$
  - **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 $Clock_i(a) < Clock_j(b)$

- **Answer: Yes!**
  - (v1,…,vi,…vN) < (v1,…,vi+1,…vN)

$V_i(a)$ **sent to j**

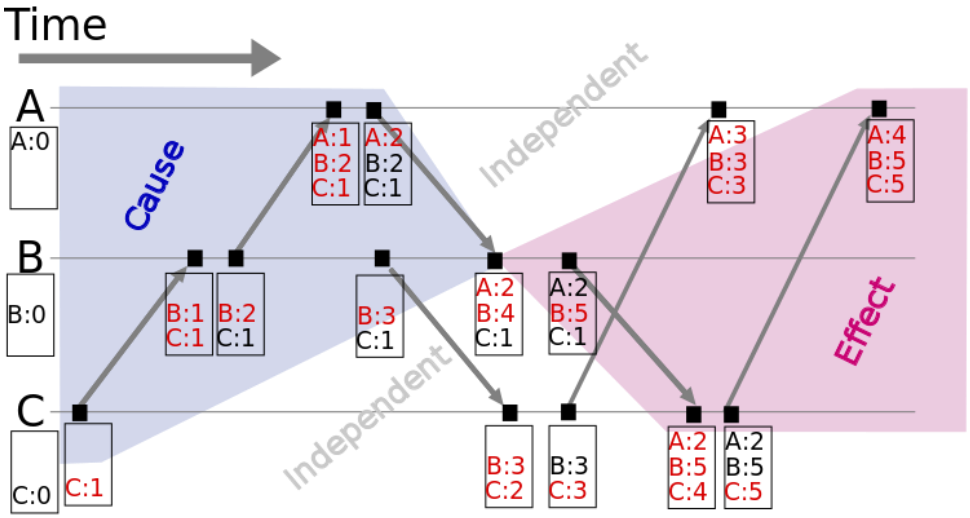
new $V_j(b)$

  - (v1,…,vN) < (max(v1,x1),…,max(vj,xj+1),…max(vN,xN))

V < V' if ≤ on each element and < on at least one element

Vector Clock Properties

- **Just showed: $a \rightarrow b$ implies $V(a) < V(b)$**
- **Not hard to show: $V(a) < V(b)$ implies $a \rightarrow b$**



- **Pros and Cons of Vector Clocks vs. Lamport's timestamps?**
  - Pro: more precise (iff)
  - Cons: much larger clocks, more complex

Physical Clocks

- **Let $C_i(t)$ denote the reading of clock C_i at physical time t**
  - Assume $C_i(t)$ is a continuous, differentiable function of t, except for isolated jump discontinuities where clock is reset
  - PC1: [assumed upper bound on rate of clock drift]
    There exists constant $\kappa \ll 1$ s.t. for all i: $\left|\frac{dC_i(t)}{dt} - 1\right| < \kappa$

- **Goal: Bound pairwise clock skew to at most ϵ**
  - PC2: For all i, j: $\left|C_i(t) - C_j(t)\right| < \epsilon$ for small constant ϵ

- **How small must κ, ϵ be to avoid anomalous behavior?**
  - Let μ be the minimum physical time needed to transmit out-of-band communication
  - Can ensure that $C_i(t + \mu) - C_j(t) > 0$ if we have $\frac{\epsilon}{1-\kappa} \leq \mu$

Physical Clocks

- **A distributed implementation:**
  - **IR1':** If P_i does not receive a message at physical time t then $\frac{dC_i(t)}{dt} > 0$
  - **IR2':** (i) P_i sends $T_m = C_i(t)$ along with its message.
    (ii) Upon receiving that message at time t',
    P_j sets $C_j(t') = \max\left(\lim_{\delta \to 0} C_j(t' - |\delta|), T_m + \mu_m\right)$

    where μ_m is the minimum delay for any message

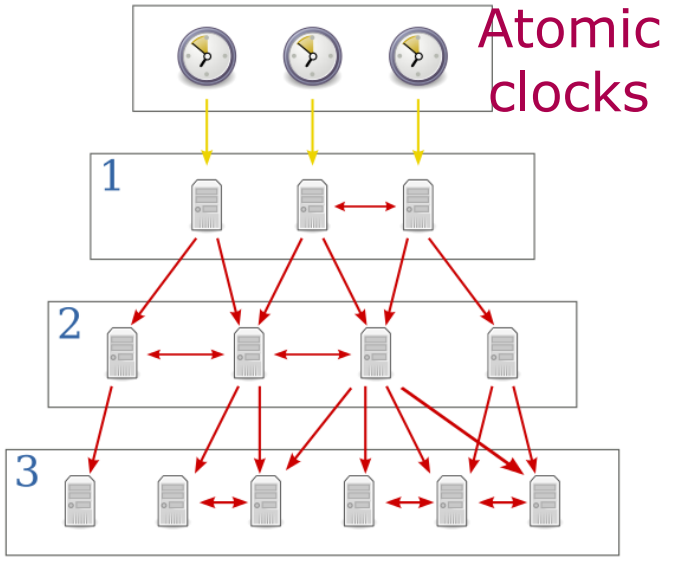
- **Theorem: Max clock skew is bounded by $d(2\kappa\tau + \xi)$**

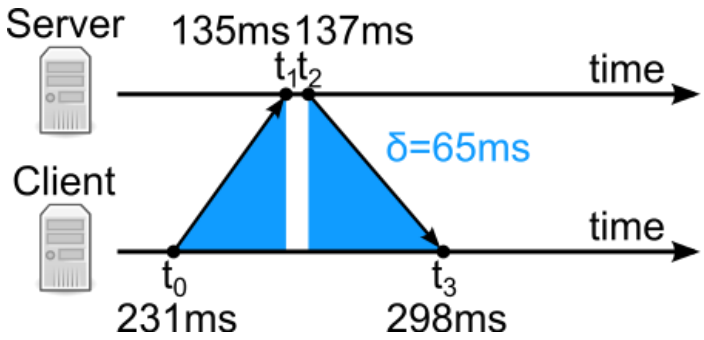
  i.e., (max number of hops) x [2 x (rate of clock skew) x (max time between point-to-point messages) + (max unpredictable message delay)]

Pros: No need for reference clocks; Clocks never set backwards
Cons: Skew versus real time; Frequent neighbor communications

Network Time Protocol (NTP)

- **In operation since before 1985**
- **Hierarchy of stratums**
- **Roundtrip delay δ**
- **Want $t_0 + \theta = t_1 - \frac{\delta}{2}$ and $t_3 + \theta = t_2 + \frac{\delta}{2}$**
- **Solve to get $\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}$**
- **Add θ to current clock**
  - $\theta = -128.5$, so clock at t_3 = 169.5 ms
- **Typically sync within 10s of millisecs on public internet**





Discussion: Summary Question #3

- **Describe what conclusion you draw from the paper as to how to build systems in the future.** Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.

Friday's Paper

"Distributed Snapshots: Determining Global States of Distributed Systems"

K. Mani Chandy & Leslie Lamport