Caltech

CS 142: Lecture 4.1
Time, Clocks, and Synchronization

Richard M. Murray

21 October 2019

Goals:

- Asynchronous computation → distributed (multi-agent) computation
- Introduce logical and vector clocks to keep track of event ordering

Reading:

- P. Sivilotti, *Introduction to Distributed Algorithms*, Chapter 5
- L. Lamport, "Time, Clocks, and the Ordering of Events in Distributed Systems", CACM 21(7) p.558–565, 1978.

Where We Are In the Course

Weeks 1-3: UNITY programs

- Predicate calculus, equivalence, quantification [HW #1]
- Program execution (UNITY semantics) [HW #2]
- Stability properties (next, stable, invariant, ~~unless~~) [HW #2]
- Progress properties (transient, ensures, leadsto) [HW #3]
- Induction (metrics) and proofs of correctness [HW #3, 4]

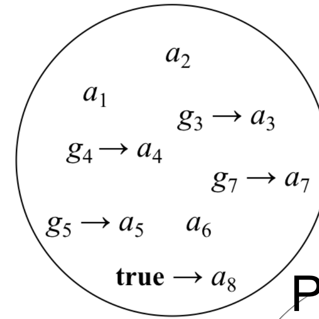
Week 4: Intro multi-agent systems

- Logical clocks and vector clocks [HW #4]
- Diffusing computations [HW #4]

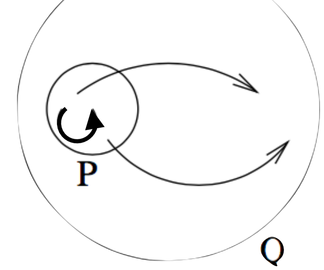
Midterm

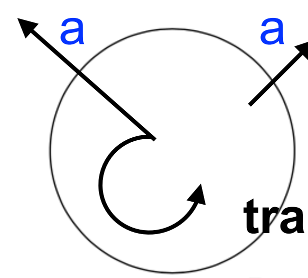
Week 5: Mutual exclusion (not covered on the midterm)

- Restrict access to a resource to a single process
- User processes + control protocols (composition)



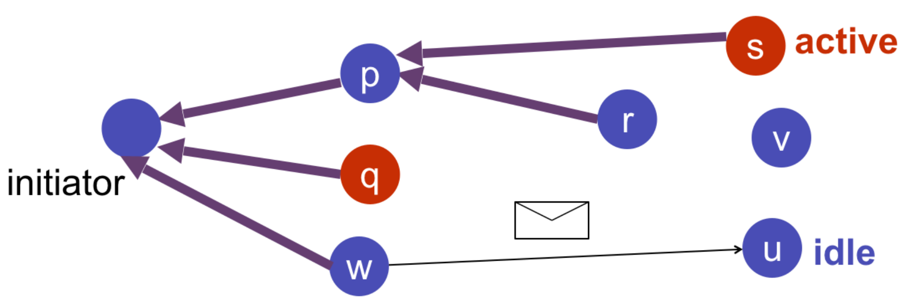
P **next** Q

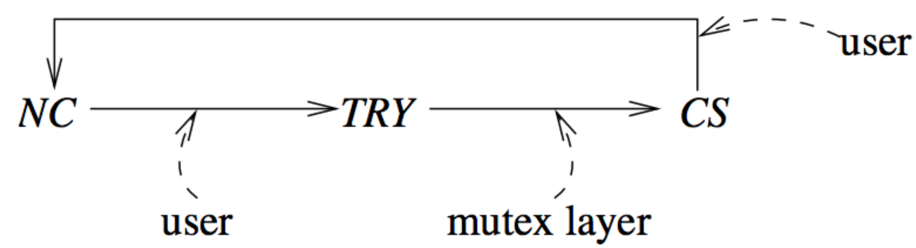




transient(Q)

$$(\forall a : a \in G : \{P\}\ a\ \{Q\})$$



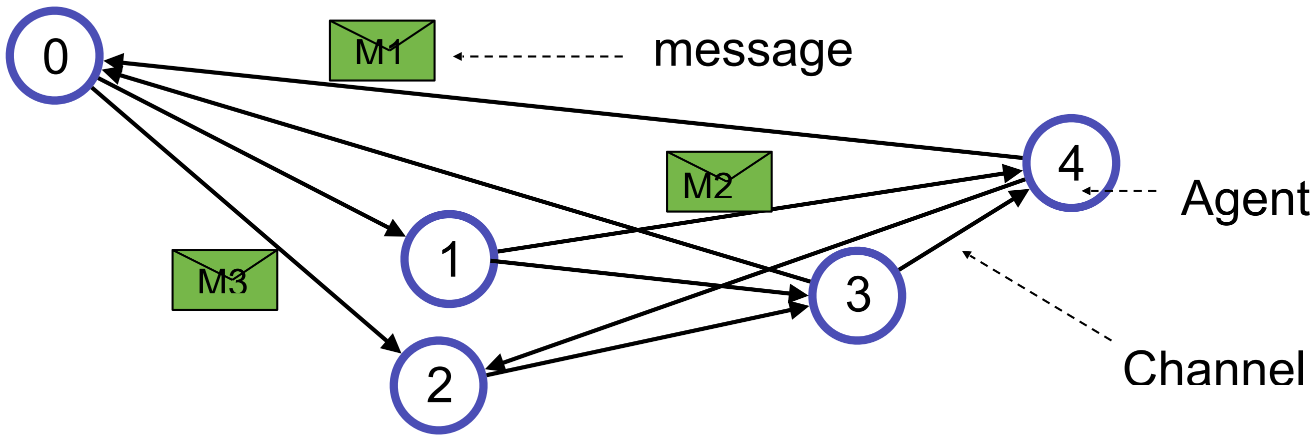


Distributed (Multi-Agent) Systems

Models for distributed systems

- Fixed set of agents (processes) + fixed set of directed *channels*
- Represent by a directed graph: vertex = agent, edge = (directed) channel

Agents and channels



- Agent: a message-passing automaton
- Channel: sequence of messages, initially empty
- State transitions:
  - Change state without sending or receiving a message
  - Change agent state and send a message on a channel
  - Receive a message on a channel and change agent state
- Channel can be FIFO, TCP-like (out of sequence), or UDP-like (dropped messages)

Challenges:

- How do we write programs that require "synchronization" (agreement on time/order)?
- How do we reason about correctness of a program with channels?

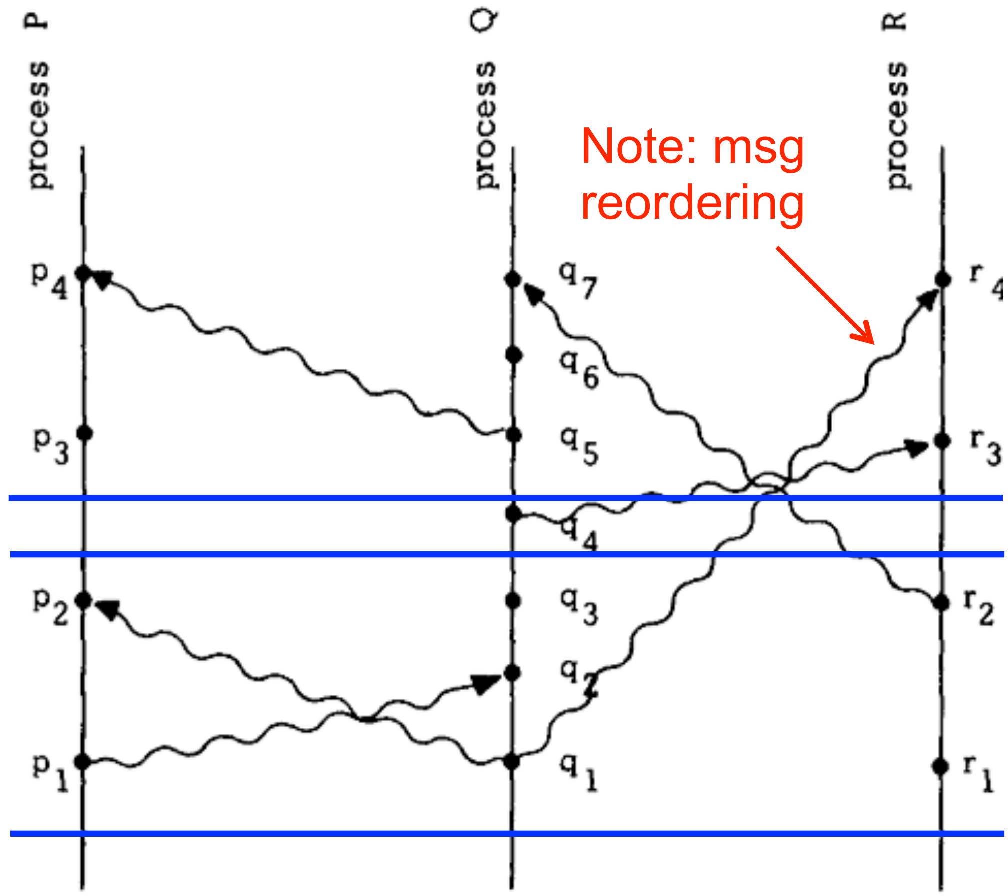
System State for a Multi-Agent System

Definition of system state: $\Sigma_1 \times \cdots \times \Sigma_N \times Q_{c_1} \times \cdots \times Q_{c_M}$

- Global state = state of each agent + state of each channel (current messages in queue)

Possible actions on this state

- Local change of state: agent i updates its state = assignment action
- Sending a message: agent i sends message to agent j:
  - Local agent updates its state (eg, msg pending $\rightarrow$ msg sent)
  - Channel adds msg to queue
- Receiving a message: agent j receives message from agent i:
  - Channel removes msg from queue
  - Receiving agent updates local state (eg, store message info)



“Space-time diagram”

Execution semantics

- Any enabled action can be executed at any time
- Receive action is enabled whenever a channel message queue is not empty

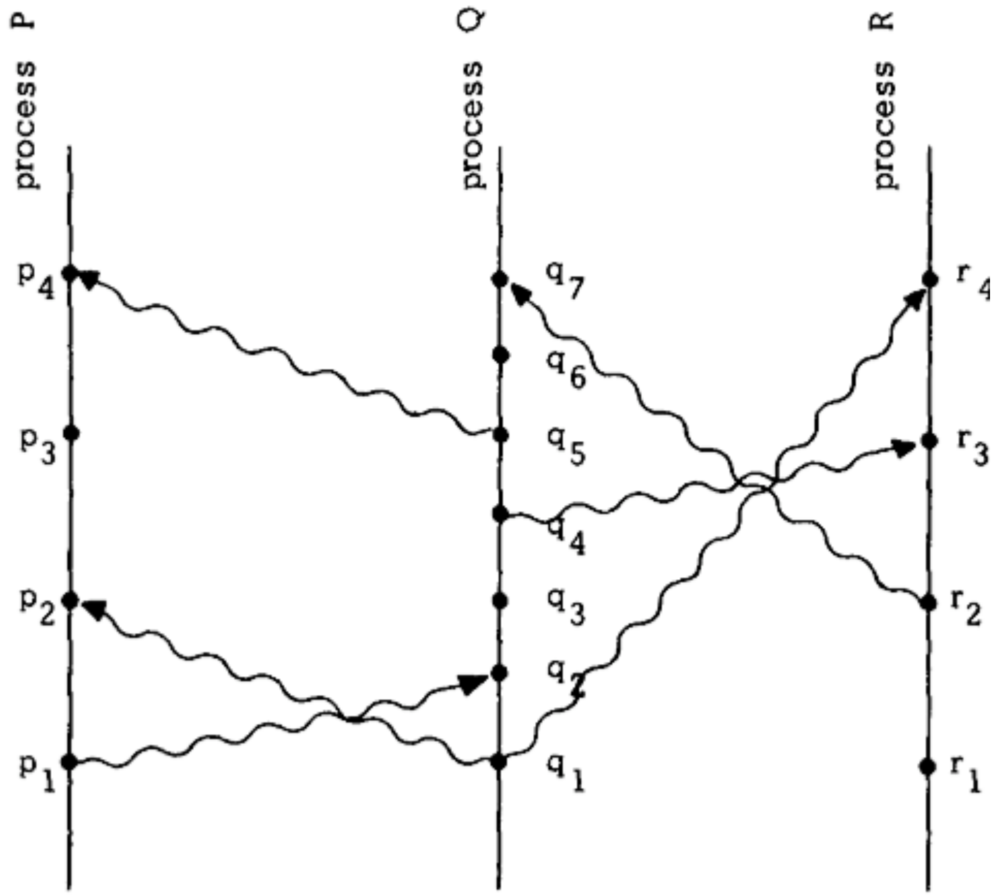
Causality in Distributed Communications (Lamport, '78)

Partial ordering: A → B ("happened before")

- If A and B are events in the same process, then A → B if A occurs first
- If A is the sending of a message by one process and B is the receipt of the same message by another process, then A → B
- Some events cannot be ordered

Logical clocks (Lamport notation)

- Let $C_i\langle \mathrm{A} \rangle$ be a clock for process P_i that assigns a number to an event A
- Define $C\langle \mathrm{B} \rangle = C_j\langle \mathrm{B} \rangle$ if B is event in P_j
- *Clock condition:* for any two events A, B: if A → B then $C\langle \mathrm{A} \rangle < C\langle \mathrm{B} \rangle$



Remarks

- Events are partially ordered: can compare some events but not all events
- Example from diagram: p1 → q3 but p3 and q3 are not related
- Clocks are not unique (can choose any set of integers with appropriate relations)
- If A → B and B → C then A → C; interpret A → B as "A can causally effect B"

UNITY Channel Modeling (and Properties)

Basic channel model: FIFO queue (messages delivered in order sent)

Safety properties

- Basic invariant: messages in a channel are delivered *in the order the were sent*
- Sequence interpretation: let c.sent and c.received be the sequence of messages sent and received (respectively) on a channel
- Property 1: c.received is an initial prefix of c.sent

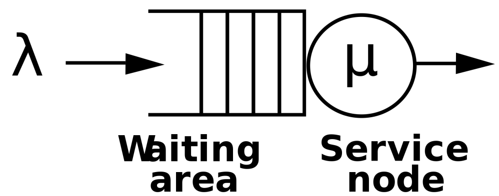
symbol for initial prefix

$$\textbf{invariant}\ (\text{c.received} \sqsubseteq \text{c.sent})$$

- Property 2: Messages are never lost:

$$\textbf{stable}\ (\text{sequence} \sqsubseteq \text{c.sent})$$

- Example: c.sent = [10, 20, 30, 40] and c.received = [10, 20]



https://en.wikipedia.org/wiki/File:Mm1_queue.svg

Liveness

- All messages eventually get sent:

$$(\text{sequence} \sqsubseteq \text{c.sent}) \leadsto (\text{sequence} \sqsubseteq \text{c.received})$$

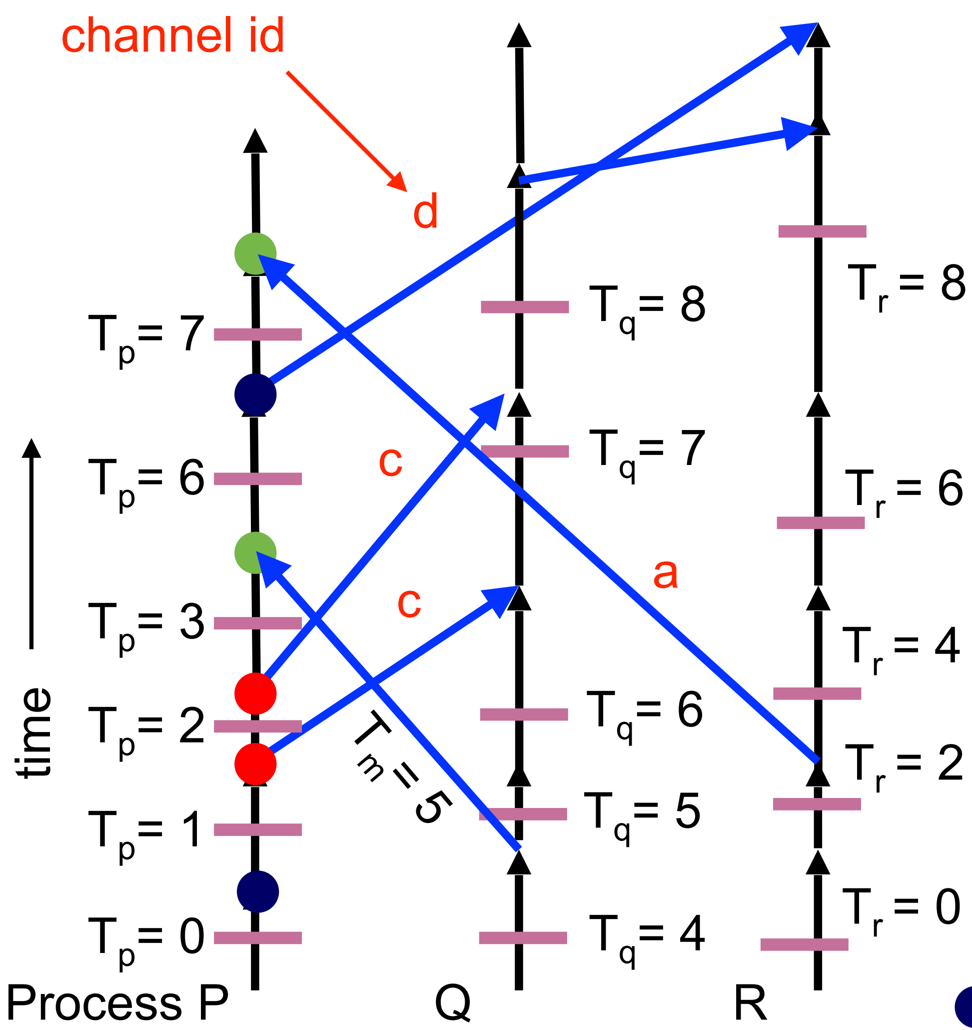
Comments

- Message-passing systems versus shared variables: once a message is sent, it cannot be deleted
- Channel model assumes TCP-like protocol; UDP is also possible (how?)

Logical Clocks

Basic idea: “synchronize” clocks between processes

- Make use of the fact that a message arrives after it was sent
- Attach local clock to each message => get information about other processes



Space-time diagram (= “timeline”)

- Graph must be acyclic (why: __________)

Logical clocks

- Increment local clock on each action (including send and receive)
- Timestamp each message with the local clock time of the sender
- Receiver ensures its local clock is greater than the timestamp of any received msg

Properties

- If A → B then clock(A) < clock (B)
- If A and B are not causally related, then clocks cannot be compared

Internal event   Send event   Receive event

Algorithm for Implementing Logical Clock

Program running on each agent

Program $\mathit{LogicalClock}\ \mathrm{j}$ program for agent j (instantiate for all agents)

var j, k : processes,
$ch.j.k$: channel from j to k,
$clock.j$: logical time of j,
$time.A$: logical time of A,

initially $clock.j = 0$

assign

sequential execution

treat "send" as new event

$$
\begin{array}{llll}
 & \text{local event } A & \longrightarrow & clock.j := clock.j + 1 \\
 & & ; & time.A := clock.j \\
[\!] & \text{send event } A & \longrightarrow & clock.j := clock.j + 1 \\
 & (\text{to } k) & ; & time.A, time.m := clock.j, clock.j \\
 & & ; & ch.j.k := ch.j.k \mid m \\
[\!] & \text{rcv event } A & \longrightarrow & clock.j := \mathbf{max}\,(time.m, clock.j) + 1 \\
 & (m \text{ from } k) & ; & time.A, ch.j.k := clock.j, tail(ch.j.k)
\end{array}
$$

add message to queue

treat "rcv" as new event

remove message from queue

Invariant:

$$
\begin{array}{ll}
 & (\forall A, j \,:\, A \text{ occurs at } j \,:\, time.A \le clock.j\,) \\
\wedge & (\forall m, j, k \,:\, m \in ch.j.k \,:\, (\exists A \,:\, A \text{ occurs at } j \,:\, time.A = time.m\,)\,) \\
\wedge & (\forall A, B \,::\, A \longrightarrow B \Rightarrow time.A < time.B\,)
\end{array}
$$

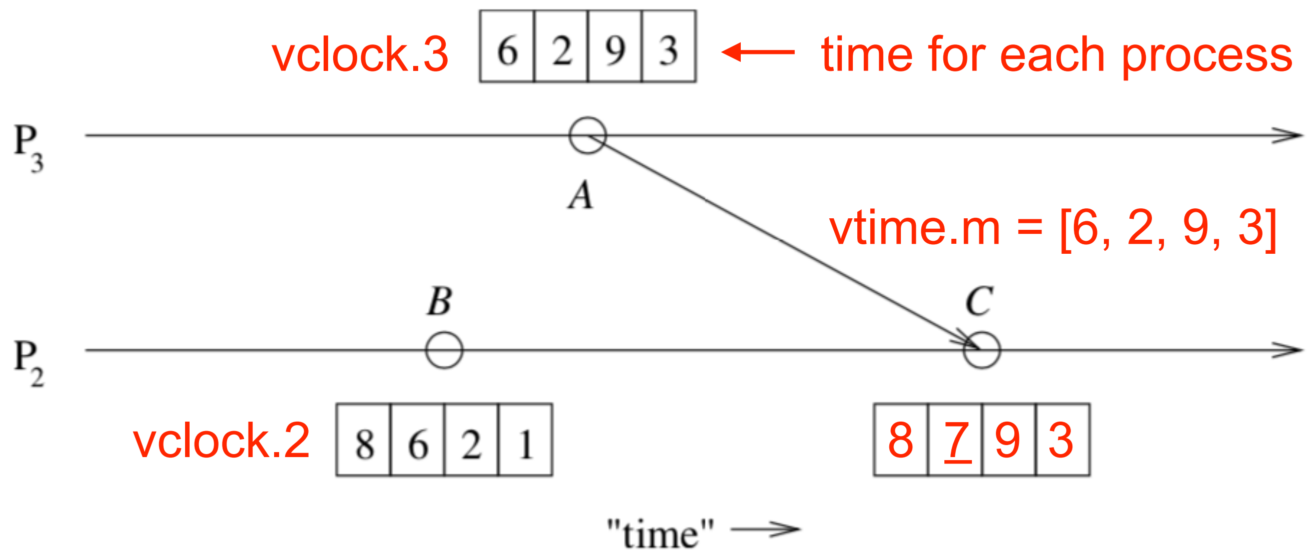
Vector Clocks

Goal: try to find a protocol such that $time(A) < time(B) \implies A \to B$

- Do this by creating a *partial order*: won't be possible to order *every* pair of events

Basic idea: each process keeps track of the last time it heard from other processes

- Each clock maintains vector of timestamps for each process in the system
- Increment local timestamp for every event (including send and receive)
- Local update: just update local (logical) clock
- Received message: take max between local clock and timestamp; update local clock entry



Note: P1 and P4 not shown

Properties of vector clocks

- Can define partial order within each process

$$vtime.A = vtime.B \equiv (\forall i :: vtime.A.i = vtime.B.i)$$

$$vtime.A \leq vtime.B \equiv (\forall i :: vtime.A.i \leq vtime.B.i)$$

$$vtime.A < vtime.B \equiv vtime.A \leq vtime.B \wedge vtime.A \neq vtime.B$$

Algorithm for Implementing a Vector Clock

Program $\quad VectorClock\, \mathrm{j}$

var $\quad j, k$: processes,

$\quad ch.j.k$: channel from j to k ,

$\quad m$: message,

$\quad A$: event,

$\quad vclock.j$: vector time of j ,

$\quad vtime.A$: vector time of A ,

initially $\quad vclock.j.j = 1$

$\quad \wedge \quad (\forall\, k \;:\; k \neq j \;:\; vclock.j.k = 0\ )$

assign

local event $A \longrightarrow \quad vclock.j.j := clock.j.j + 1$

$\quad ; \quad vtime.A := vclock.j$

$[\!]$ send event $A \longrightarrow \quad vclock.j.j := vclock.j.j + 1$

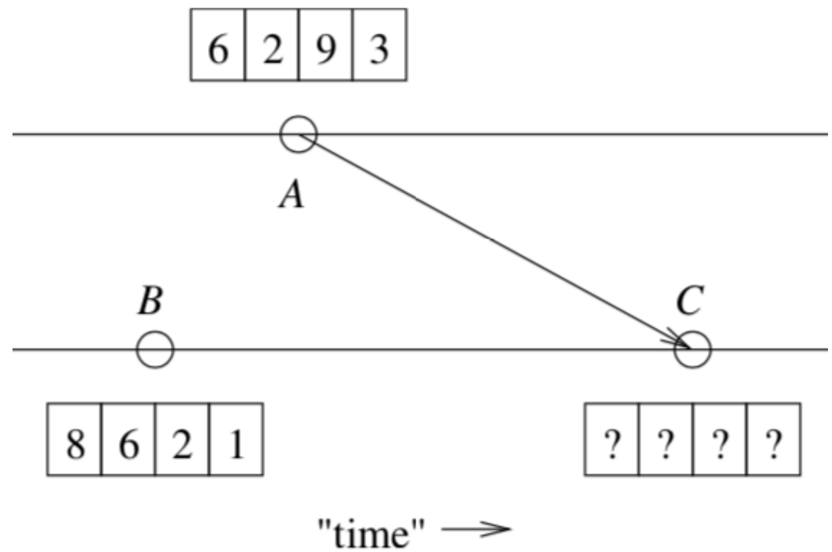
(to k) $\quad ; \quad vtime.A, vtime.m := vclock.j, vclock.j$

$\quad ; \quad ch.j.k := ch.j.k \mid m$

$[\!]$ rcv event $A \longrightarrow \quad vclock.j := \mathbf{max}\ (vtime.m, vclock.j)$ element-wise max

(m from k) $\quad ; \quad vclock.j.j := vclock.j.j + 1$

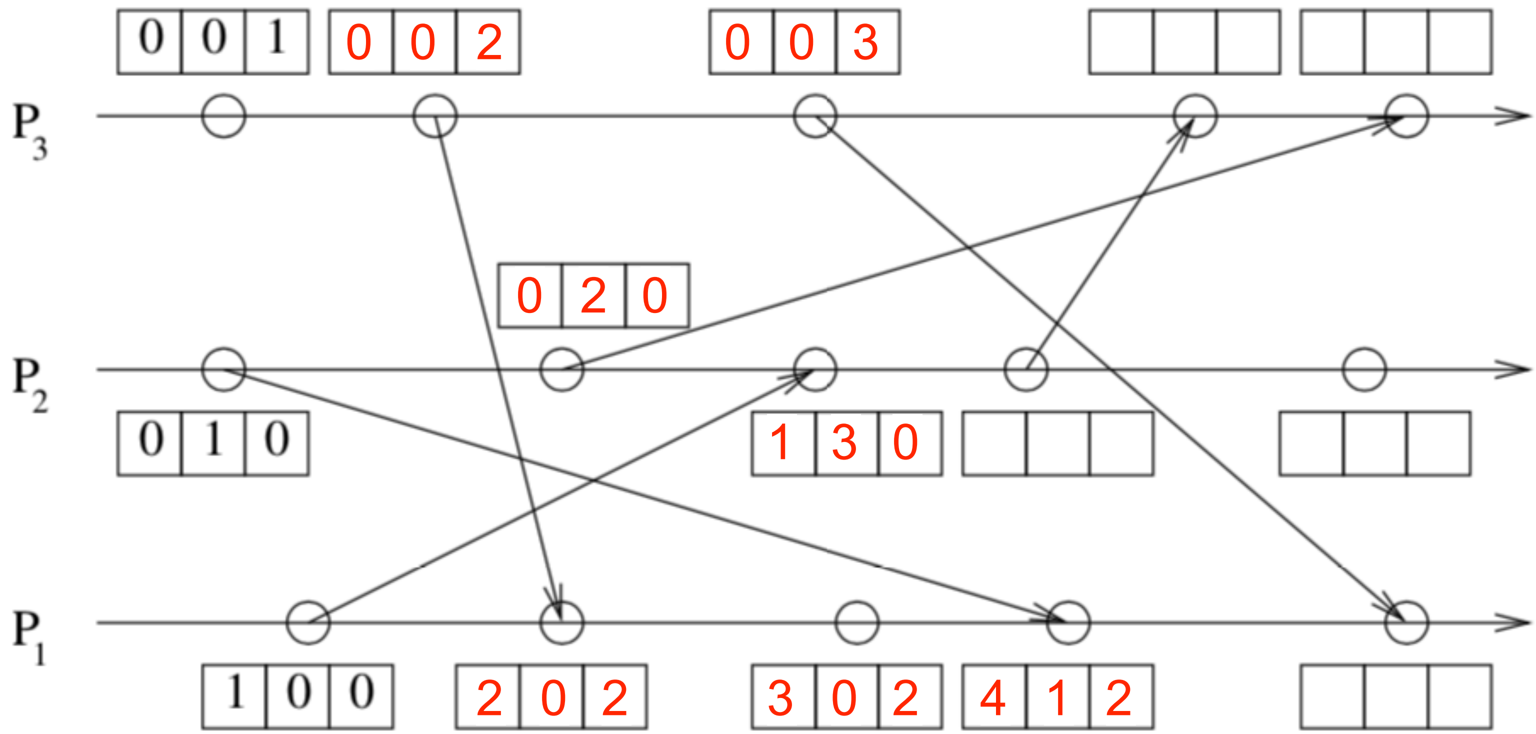
$\quad ; \quad vtime.A, ch.j.k := vclock.j, tail(ch.j.k)$



vclock.i = vclock[i]= ith processes vector clock

vclock.i.j = (vclock[i])[j] = jth entry of ith processes vector clock

Vector Clock Example (from Sivilotti)



Invariant: the following sets of properties are always true

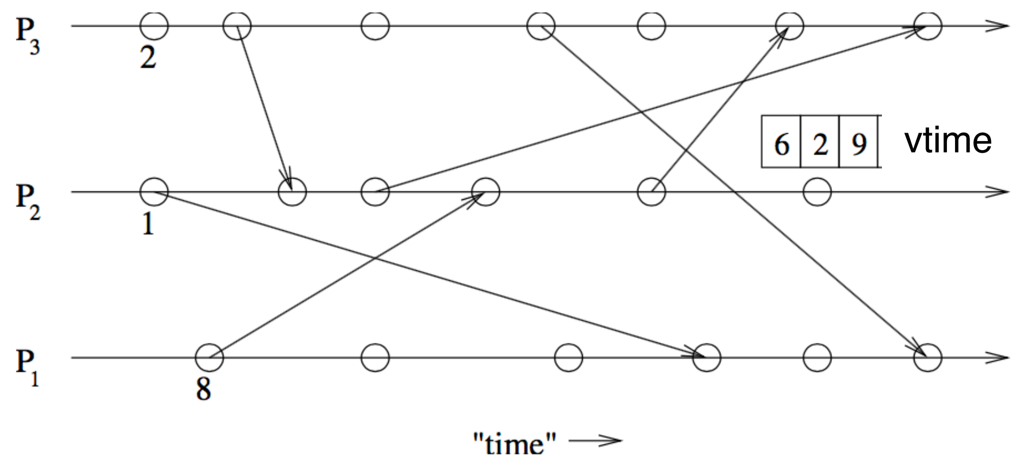
$$
\begin{aligned}
& (\forall\, j, k \;::\; vclock.k.j \le vclock.j.j \;) \\
\wedge\quad & (\forall\, j, k, m_j \;::\; vtime.m_j.k \le vclock.j.k \;) \\
\wedge\quad & (\forall\, A_j, B_k \;::\; A_j \longrightarrow B_k \equiv vtime.A_j < vtime.B_k \;) \\
\wedge\quad & (\forall\, A_j, B_k \;::\; A_j \longrightarrow B_k \Leftarrow vtime.A_j.j \le vtime.B_k.j \;)
\end{aligned}
$$

local clock for process is always bigger than remote estimates of clock

m_j = message from jth process

Summary: Time, Clocks, and Synchronization

Channel model: FIFO, lossless, directed



Events, system timelines and logical time

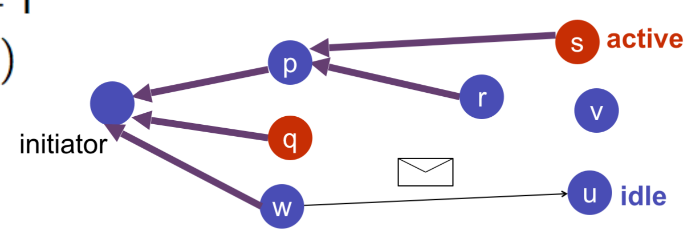
- Can't assume process clocks agree
- Make use of "logical time"

$$A \longrightarrow B \;\Rightarrow\; time.A < time.B$$

Algorithm for setting logical time

$$
\begin{array}{lll}
 & \text{local event } A \longrightarrow & clock.j := clock.j + 1 \\
 & \qquad ; & time.A := clock.j \\
[\!] & \text{send event } A \longrightarrow & clock.j := clock.j + 1 \\
 & (\text{to } k\,) \qquad ; & time.A, time.m := clock.j, clock.j \\
 & \qquad ; & ch.j.k := ch.j.k \mid m \\
[\!] & \text{rcv event } A \longrightarrow & clock.j := \mathbf{max}\,(time.m, clock.j) + 1 \\
 & (\,m \text{ from } k\,) \qquad ; & time.A, ch.j.k := clock.j, tail(ch.j.k)
\end{array}
$$

Wed: gossip algorithms



Properties

$$
\begin{array}{ll}
 & (\forall A, j \;:\; A \text{ occurs at } j \;:\; time.A \le clock.j\,) \\
\wedge & (\forall m, j, k \;:\; m \in ch.j.k \;:\; (\exists A \;:\; A \text{ occurs at } j \;:\; time.A = time.m\,)\,) \\
\wedge & (\forall A, B \;::\; A \longrightarrow B \Rightarrow time.A < time.B\,)
\end{array}
$$

Vector clocks: $A \longrightarrow B \;\equiv\; vtime.A < vtime.B$