



NTNU

Det skapende universitet



TDT 4225 Very Large, Distributed Data Volumes

Chapter 14 (Coulouris et al)
Time and global states

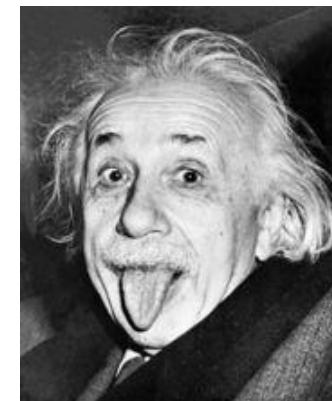
Svein Erik Bratsberg / Jon Olav Hauglid

Part 1: Time

- Why is time important?
- Physical time
 - Skew, drift, UTC
 - External and internal synchronization
 - NTP: Network Time Protocol
- Logical time
 - Definitions
 - Logical clocks
 - Vector clocks
- Part 2: Global state

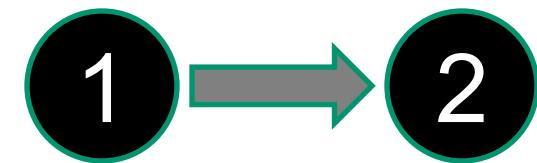
Why is time important?

- Time is used «everywhere»
 - When did something happen?
 - What happened first and what happened later?
 - Bank transactions, e-mail, ...
- Trivial for a single computer
- More difficult in a distributed system
 - Communication takes time
 - What is the time?
 - No universally correct time



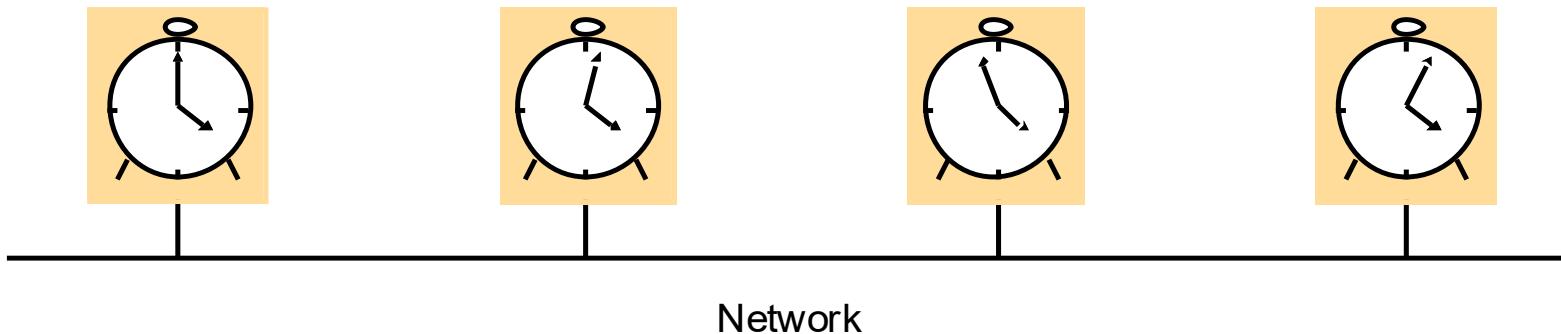
Physical and logical time

- Physical time
 - Timestamp of event
 - Can derive order of events
 - Requires synchronized clocks
- Logical time
 - Order of events
 - Focused on cause and effect
 - Typically a counter incremented for each event



Physical time

- Demand for physical time
- Ordering of events based on timestamp
- Can we trust clocks?
 - «Skew» - difference between clocks at a point in time
 - «Drift» – skew changes over time



Network

UTC – Coordinated Universal Time

- Highly accurate international time standard
- Based on atomic clocks
- An extra second is sometimes inserted due to Earth's rotation slowing down (leap seconds)
- Negative leap seconds may appear as well
- Time zones are relative to UTC
 - We are at UTC+1 (UTC+2 in the summer)
- Transmitted using
 - Ground based stations (~1 ms accuracy)
 - Satellites GPS (~1 μ s accuracy)

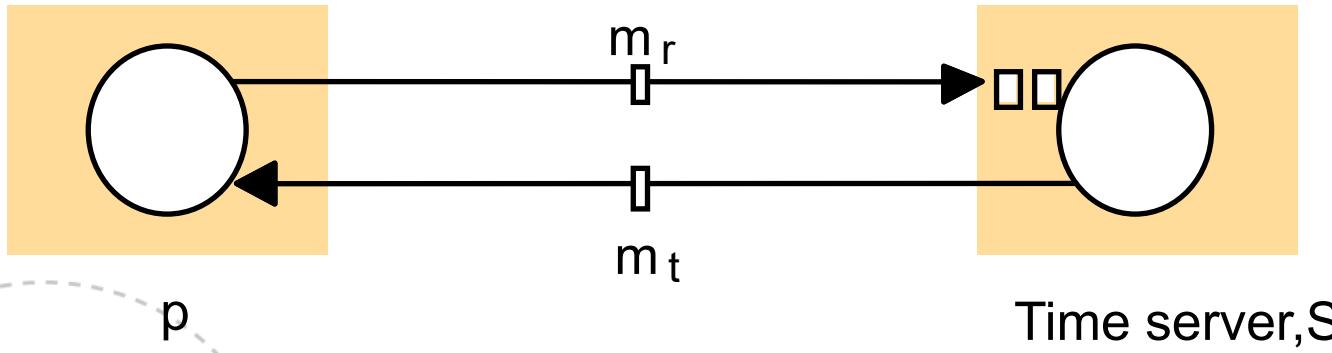
Physical clock synchronization

- External synchronization
 - Clocks synchronized against an external time source
 - Christian's algorithm
- Internal synchronization
 - Synchronization of clocks internally in a distributed system
 - Not necessarily the «correct» time
 - The Berkeley algorithm
- Basic problem: Communication takes time

Christian's algorithm

External time server, S (UTC)

1. p sends message m_r to S
2. S replies with its time t in message m_t
3. When p receives m_t , it sets its clock to $t + \text{half of the time passed since } m_r \text{ was sent}$



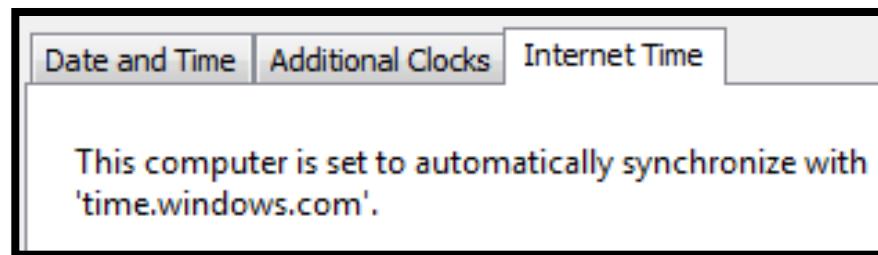
The Berkeley algorithm

One node selected as *master*

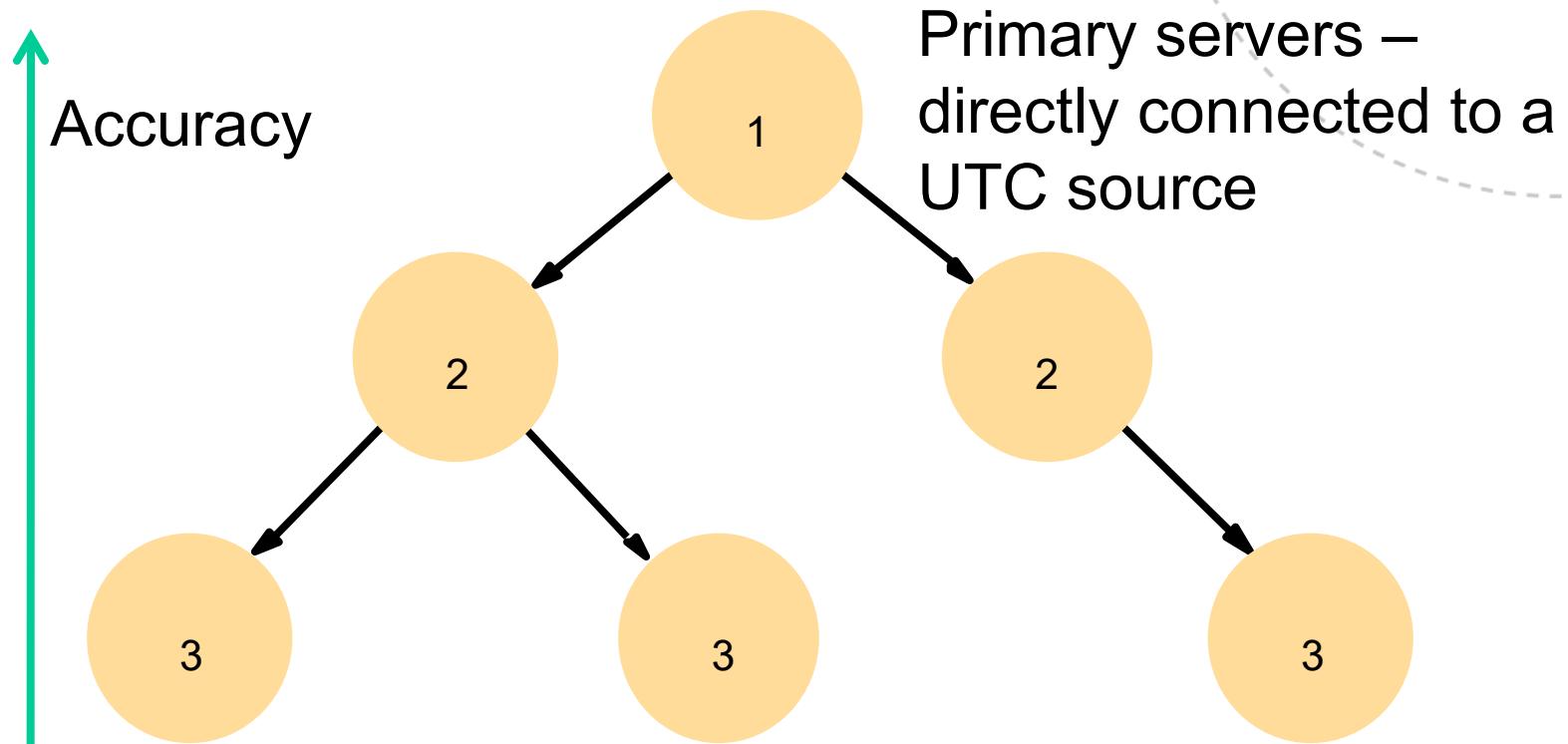
1. The master polls all other nodes (*slaves*)
2. Slaves reply with their local time
3. Master calculates average time
 - Message latency considered
 - Ignores outliers
4. Master sends individual differences to each slave (why differences?)

NTP: Network Time Protocol

- Protocol for synchronizing clocks connected to the Internet
- Uses UTC
- Focus on:
 - Scalability – hierarchy of servers
 - Correctness – handle clock drift
 - Reliability – dynamic reconfiguration
 - Security – authentication etc.



NTP: Server hierarchy (logical)



NTP: Synchronization

Three synchronization modes:

1. Multicast (LAN)

- Assumes fixed message latency
- Periodic multicast (not on demand)

2. Procedure-call (e.g. Christian's algorithm)

3. Symmetric mode (high accuracy)

- Servers communicate in pairs
- One server can be in multiple pairs
- Estimates *offset* and *delay*

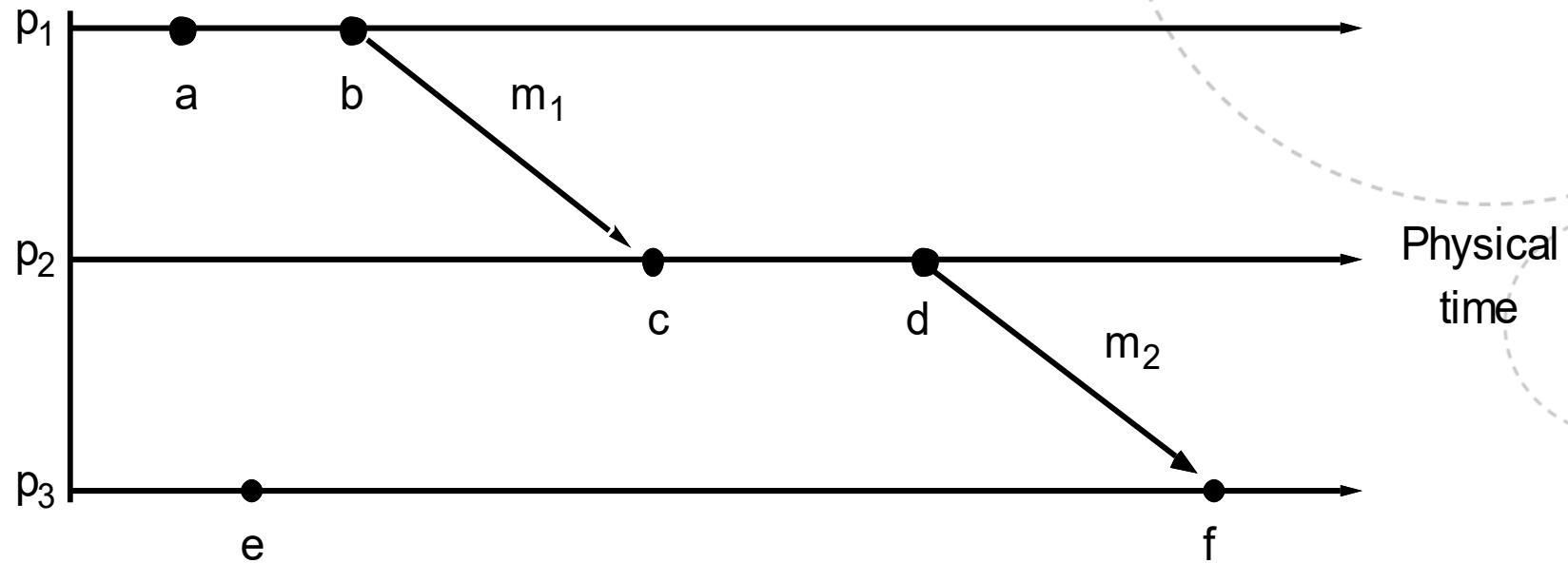
Logical time

- If we only need to order events, physical time is overkill
- Also, perfect synchronization of physical clocks is impossible!
- Local time – focus on event order
 - Two local events happened in the order observed by the process executing them
 - A message must be sent before it can be received
 - I.e. cause → effect

Definitions

- Collection of processes $p_i, i = 1, 2, \dots N$
- Every process p_i has a state s_i
 - E.g. the values of all variables
- Processes communicate using messages
- Events
 - State change
 - Sending or receiving a message

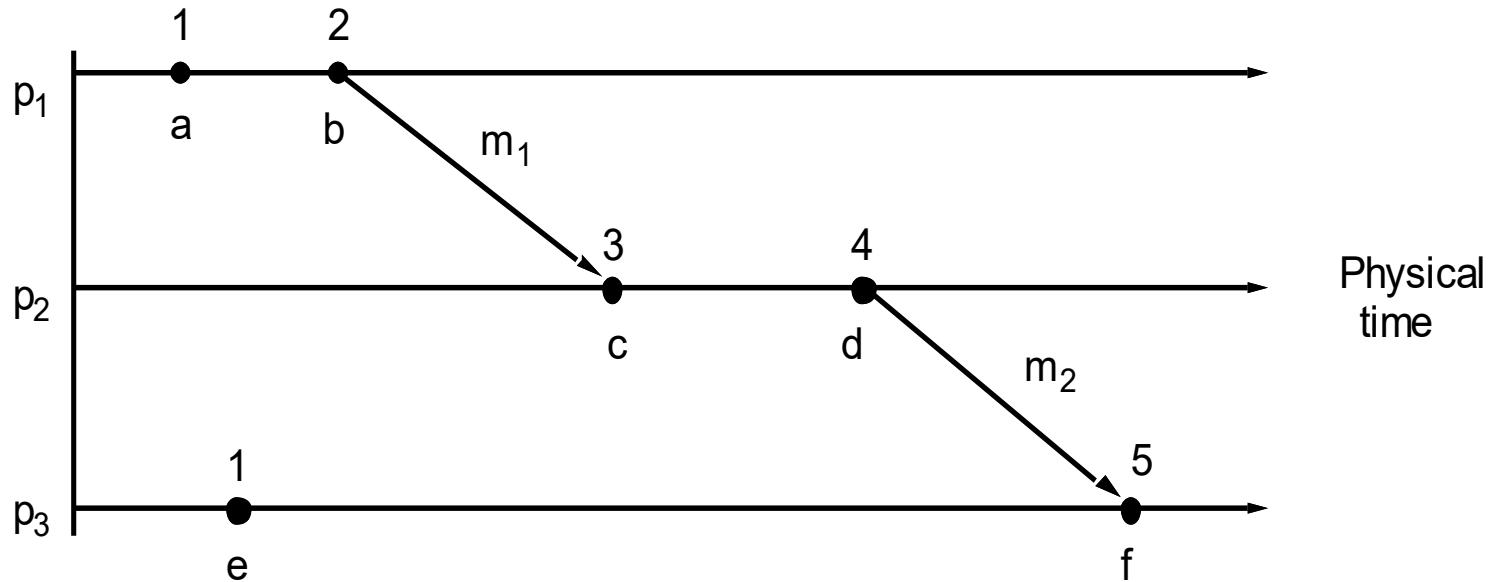
Happened-before (\rightarrow)



- Local events: $a \rightarrow b$; $c \rightarrow d$; $e \rightarrow f$
- Messages: $b \rightarrow c$; $d \rightarrow f$
- Derived: $a \rightarrow c$; $a \rightarrow f$; $b \rightarrow d$
- Concurrent: $a \parallel e$; $b \parallel e$; $c \parallel e$; $d \parallel e$

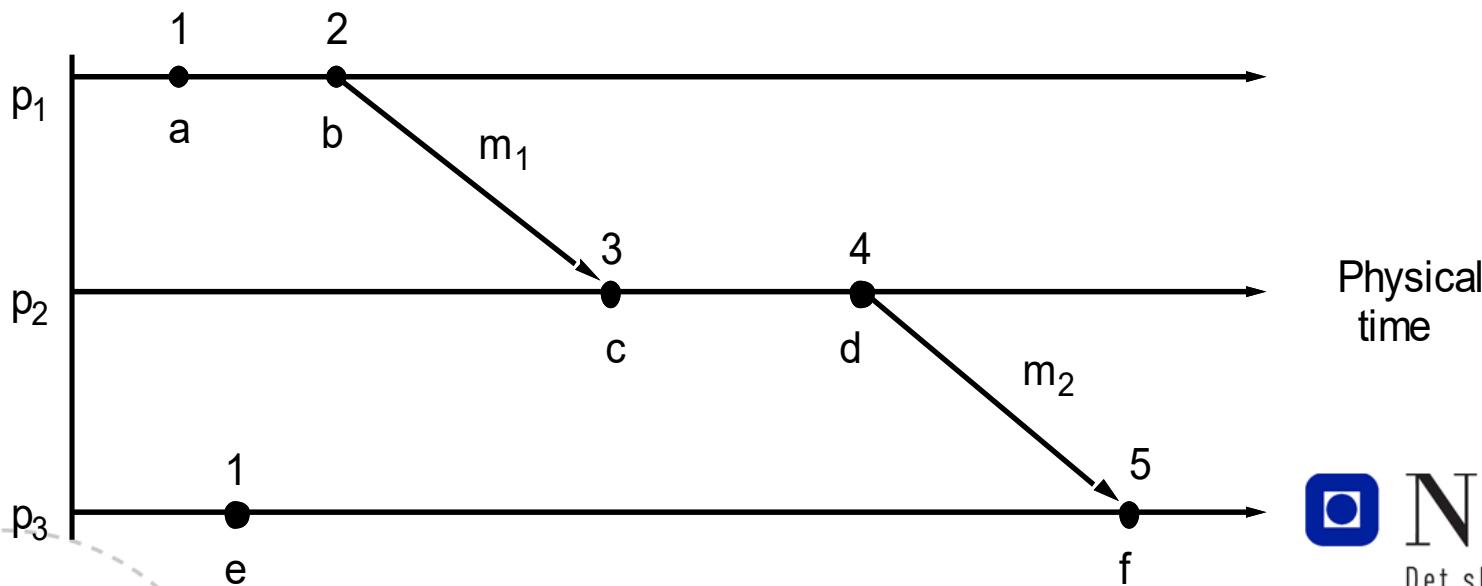
Logical clocks (Lamport) (1/2)

- Every process has a logical clock (counter) L_i
- Before every event: $L_i = L_i + 1$
- Attach clock value to every message, $t = L_i$
- When receiving, $L_j = \max(L_j, t) + 1$

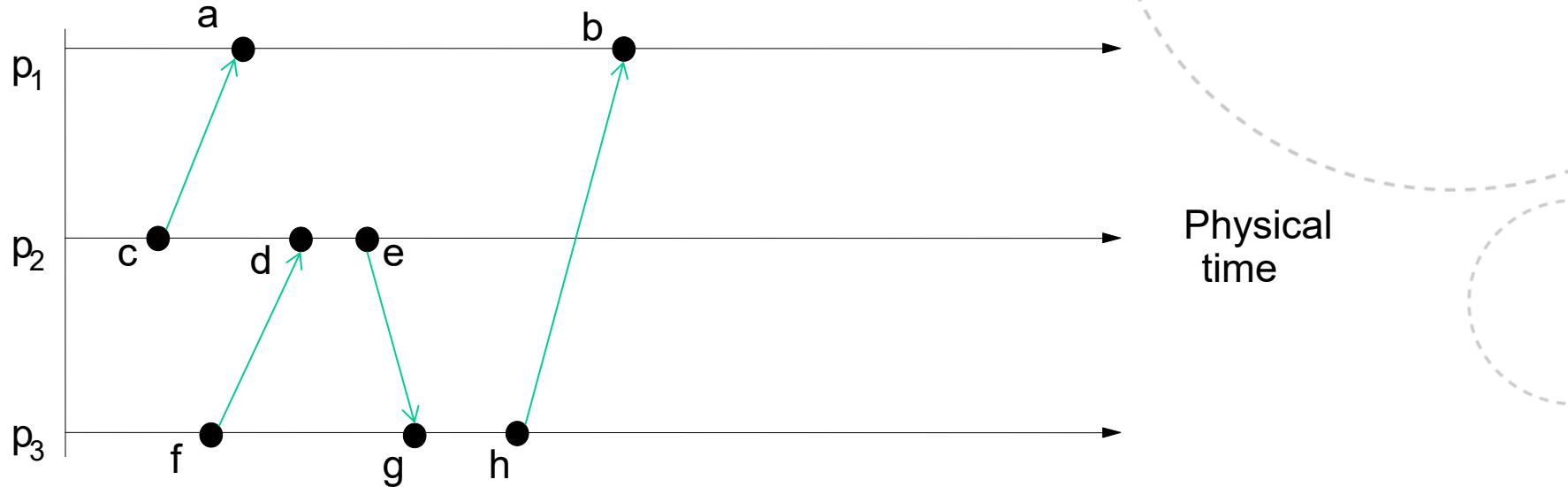


Logical clocks(Lamport) (2/2)

- If $e \rightarrow e'$ then $L(e) < L(e')$
- But the reverse is not true!
 - $L(e) < L(b)$ without $e \rightarrow b$
 - But $L(e') > L(e) \rightarrow (\text{not } e' \rightarrow e)$



Try yourselves!



- What are the logical clock values?
- Is $d \rightarrow g$?
- Is $a \rightarrow g$?

Vector clocks (1/3)

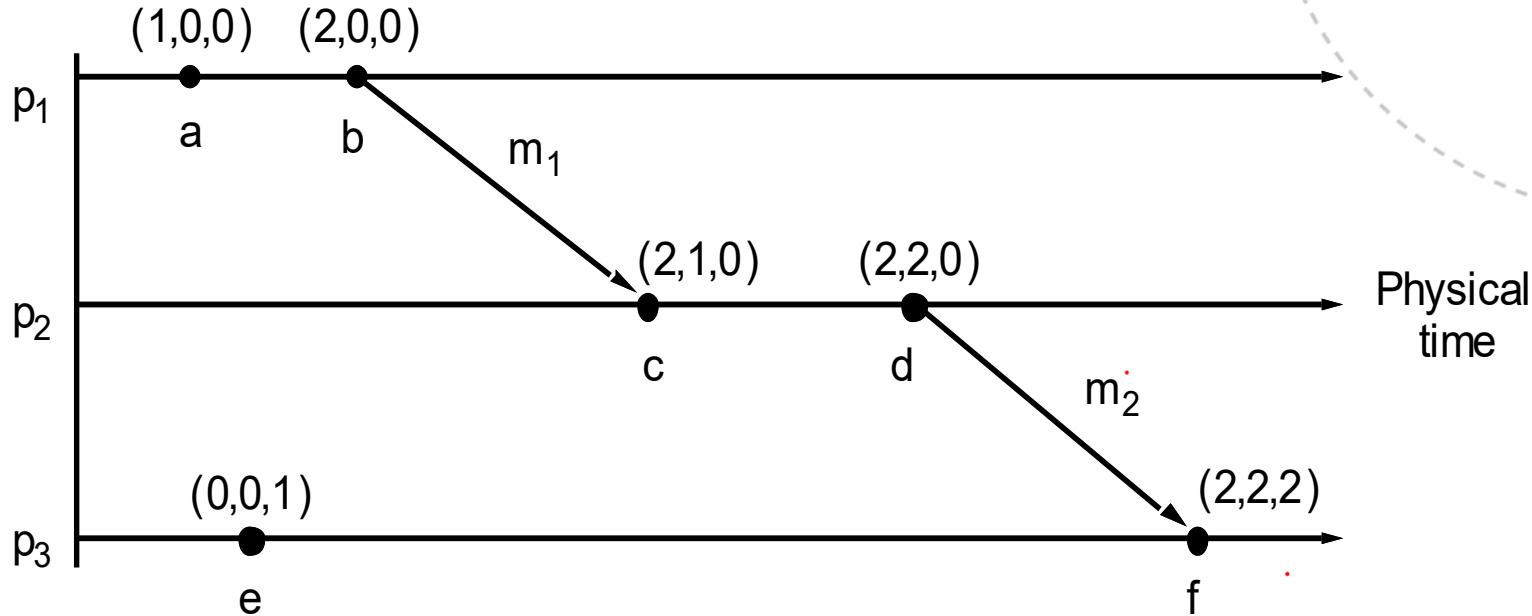
- Logical clocks only get you so far
- To figure out more about event order, we need to store/transfer more info
- Vector clock, given N processes
 - Every process has a vector of N elements
 - Contains the number of events from each process that the given process can have been affected by

Vector clocks (2/3)

Definitions:

- V_i – vector at process i
- Initially all vector elements = 0
- Before each event at p_i :
$$V_i[i] = V_i[i] + 1$$
- p_i attaches $t = V_i$ to all messages
- When p_i receives a message,
$$V_i[j] = \max(V_i[j], t[j]),$$
 for $j = 1, 2 \dots N$
(Also $V_i[i] = V_i[i] + 1$)

Vector clocks (3/3)



- If $e \rightarrow e'$ then $V(e) < V(e')$
- If $V(e) < V(e')$ then $e \rightarrow e'$
 - $V < V'$ iff $V \leq V'$ and $V \neq V'$
 - \leq and $=$ must hold for all pairs of vector elements

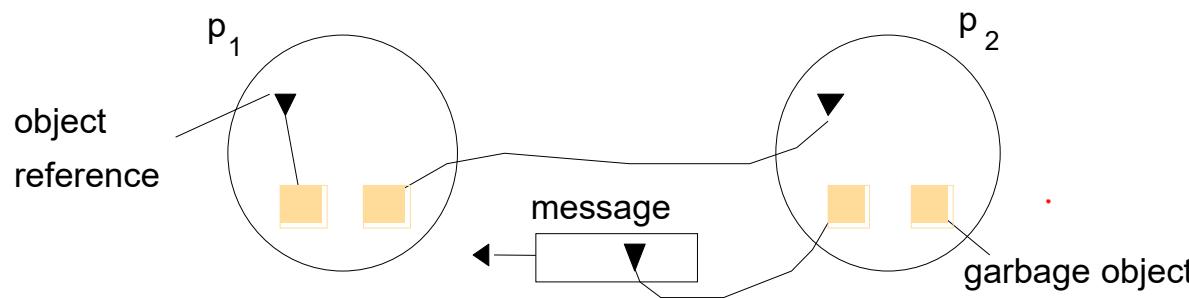
As before
New!

Part 2: Global states

- What and why?
 - Distributed garbage collection
 - Distributed deadlock detection
 - Distributed debugging
- How?
 - Cuts and globally consistent states

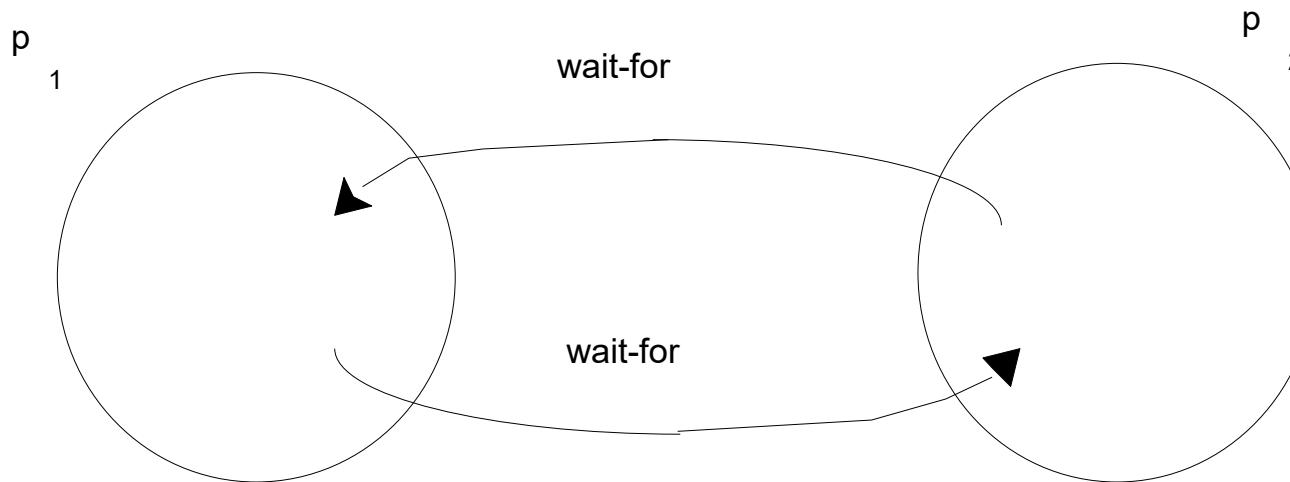
Distributed garbage collection

- Garbage: Objects without active references
- References can be:
 - Local
 - At other processes/nodes (new)
 - In messages (new)
- Need global state (including messages in transit)



Distributed deadlock detection

- Distributed waits-for cycle
- Need global state to detect this



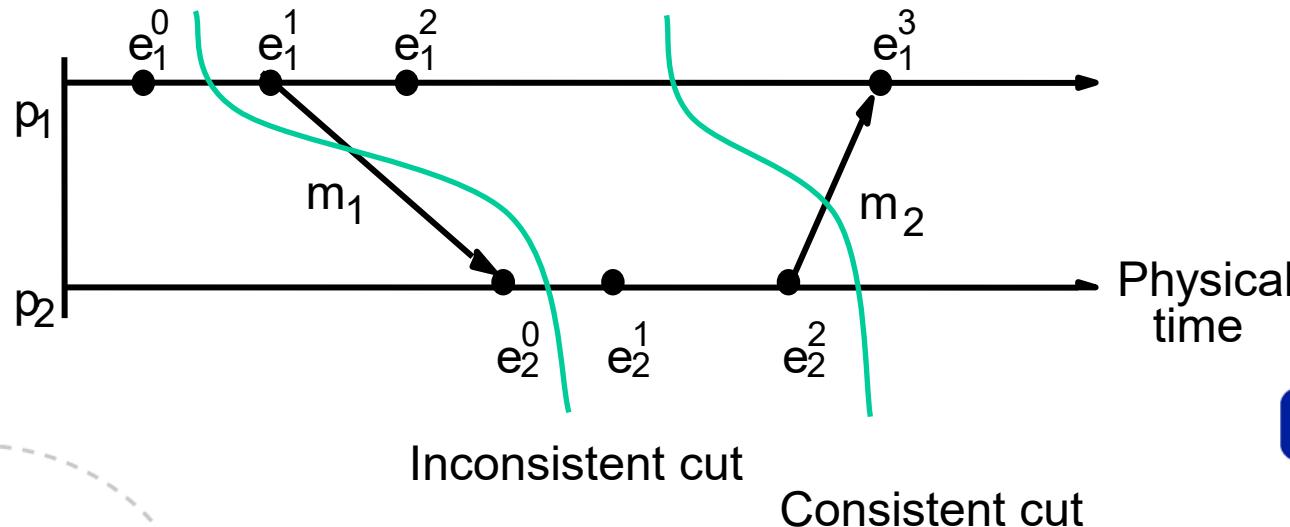
Distributed debugging

- How did variables change at runtime?
 - Example: Has $|x_1 - x_2|$ ever been > 50 ?
- Variables can be located at different processes/nodes
- Problem: Global consistent view of variable values



Cuts (1/2)

- Local history: Events at one process
- Global history: Union of all local histories
- Cut: Subset of global history (local prefix)
- What is the problem?
 - How to find consistent cuts without global time?



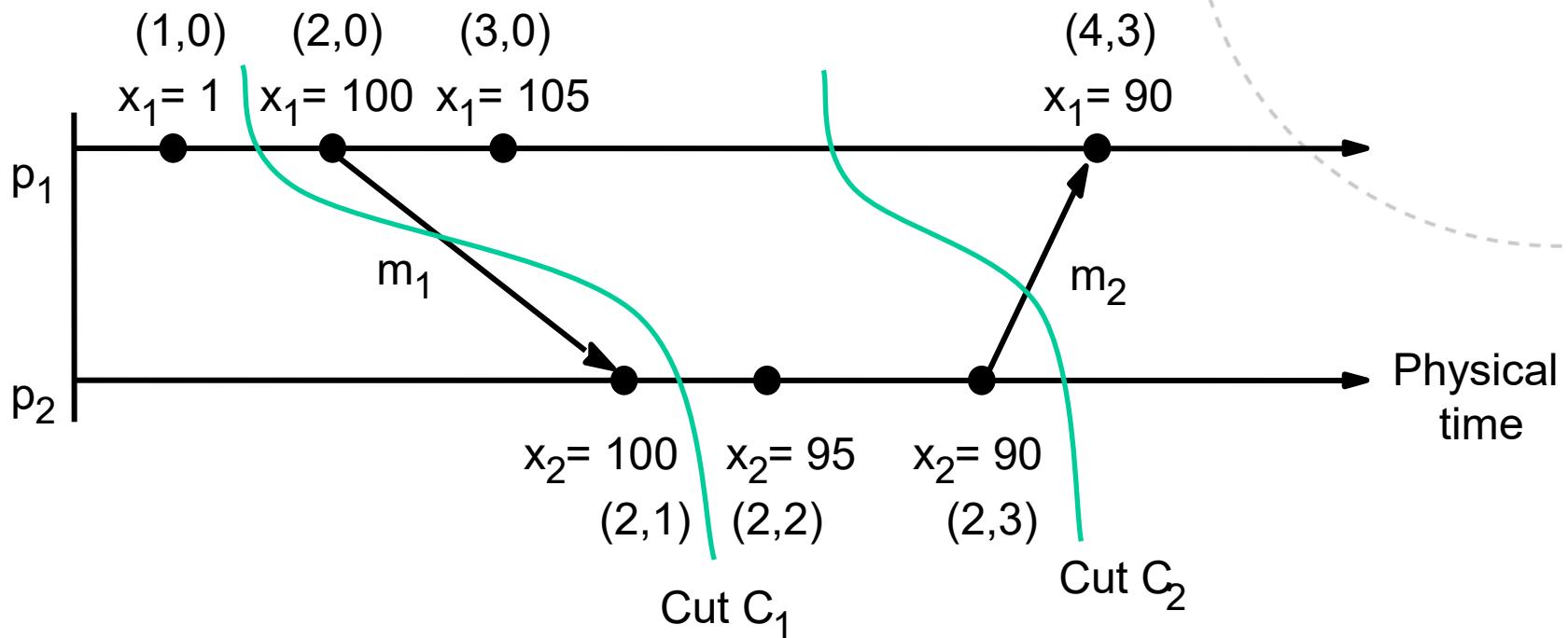
Cuts (2/2)

- A cut C is consistent if
 - For all events $e \in C$, $f \rightarrow e \Rightarrow f \in C$
 - Inconsistent: The system could never have been in this state
 - Consistent \Rightarrow Global consistent state
- Run
 - Global history where the order satisfies all local histories
- Consistent run / linearization
 - All global states passed through are consistent
- Reachable
 - S' is reachable from S if there is a consistent run between them

Distributed debugging

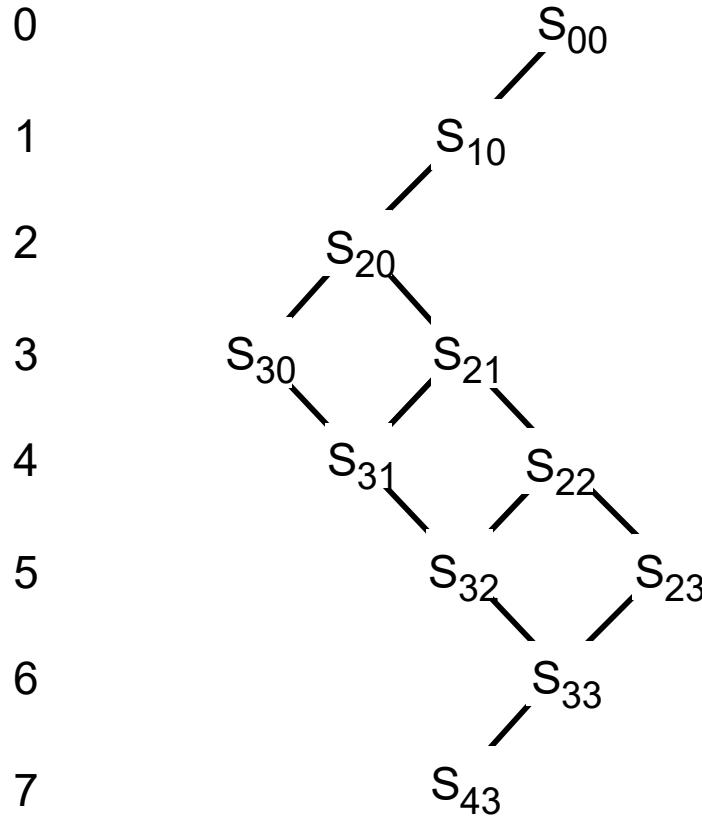
- After the fact: Was a condition true during execution?
- All participants send information about state changes to an external observer
- Global state predicate ϕ
 - Possibly ϕ
 - At least one consistent run passes through a global consistent state where $\phi = \text{true}$
 - Definitely ϕ
 - All consistent runs pass through a global consistent state where $\phi = \text{true}$

Observing global state



- The observer is notified about all state changes
- Uses vector clocks to find global consistent states
- Has $|x_1 - x_2|$ ever been > 50 ?

Alternative consistent runs



S_{ij} = global state after i events at process 1
and j events at process 2

Possibly ϕ
Definitely ϕ

Passes S where $\phi = \text{true}$
Can't avoid S where $\phi = \text{true}$