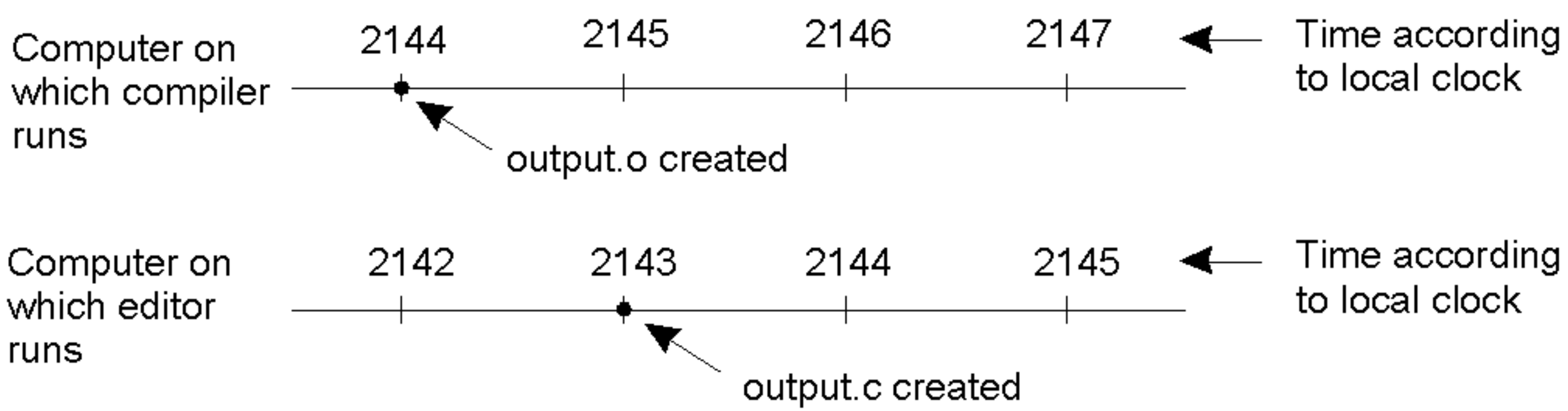
Physical and Logical Clocks

Yao Liu

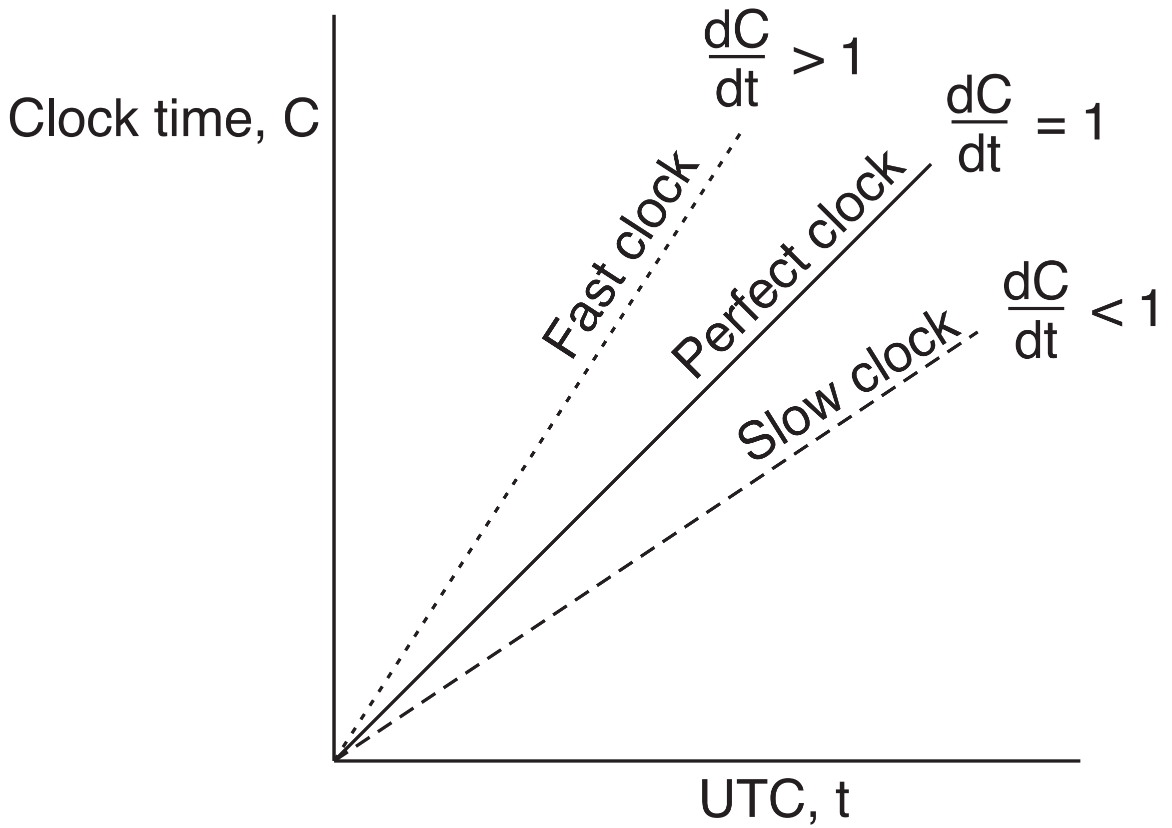
Each machine has its own clock

- An event that occurred after another event may nevertheless be assigned an earlier time.



Perfect, fast, and slow clocks

- The relation between clock time and UTC when clocks tick at different rates.

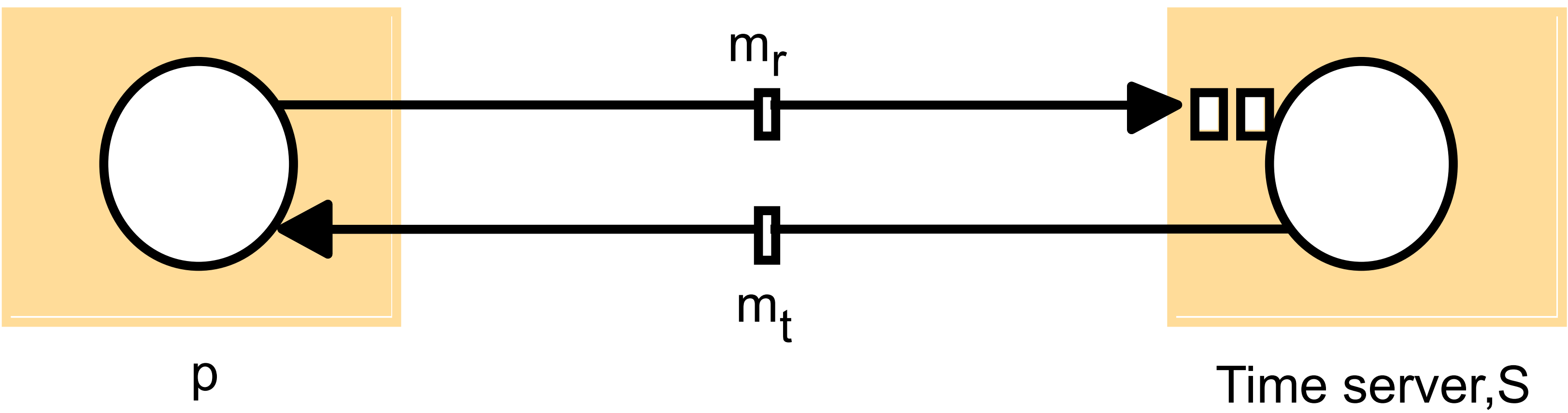


Clock synchronization

- Physical clocks drift, therefore need for clock synchronization algorithms
  - Clock synchronization algorithms – Cristian's algorithm, NTP, Berkeley algorithm, etc.

- However, since we cannot perfectly synchronize clocks across computers, we cannot use physical time to order events

Clock synchronization using a time server



Cristian's algorithm

- A process *p*
  - requests the time in a message m_r
  - receives the time value t in a message m_t

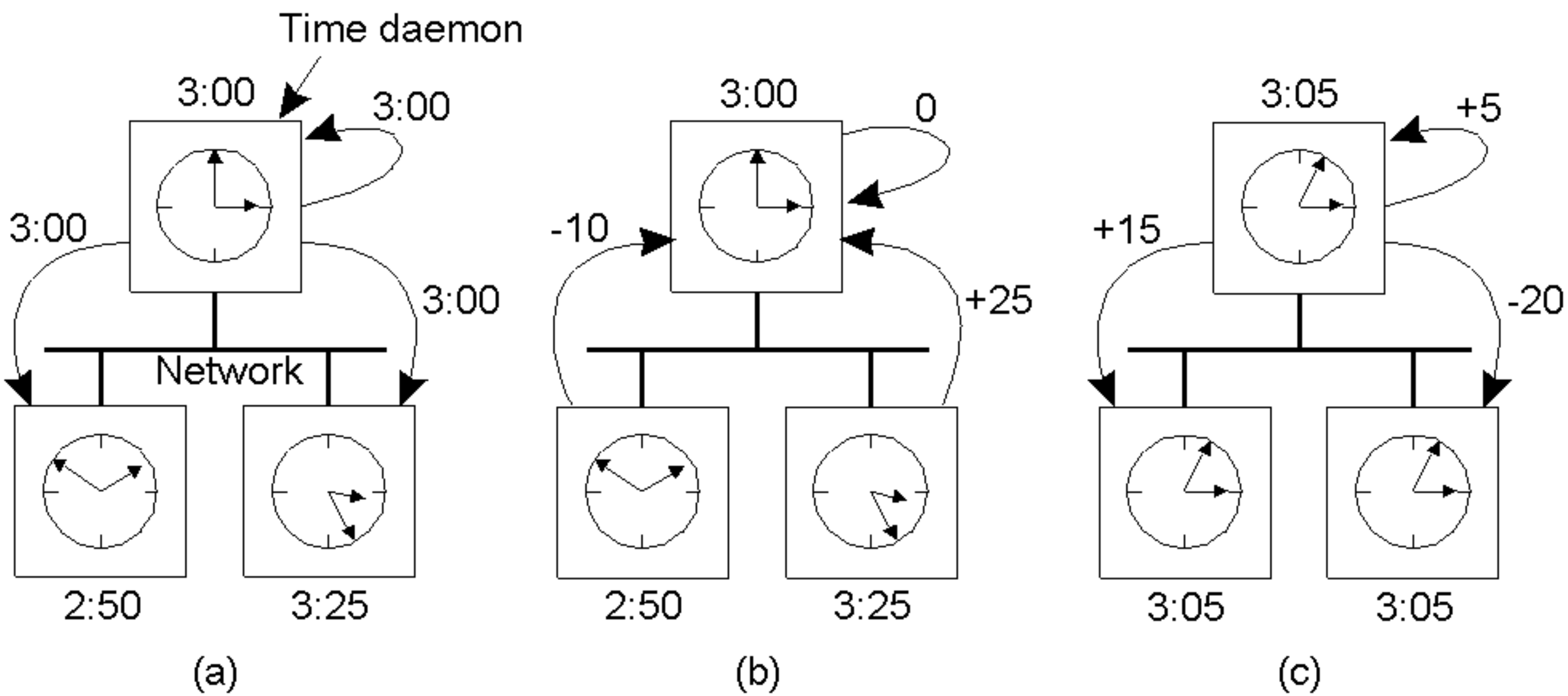
- *p* should set its time to $t + T_{round}/2$

Cristian's algorithm

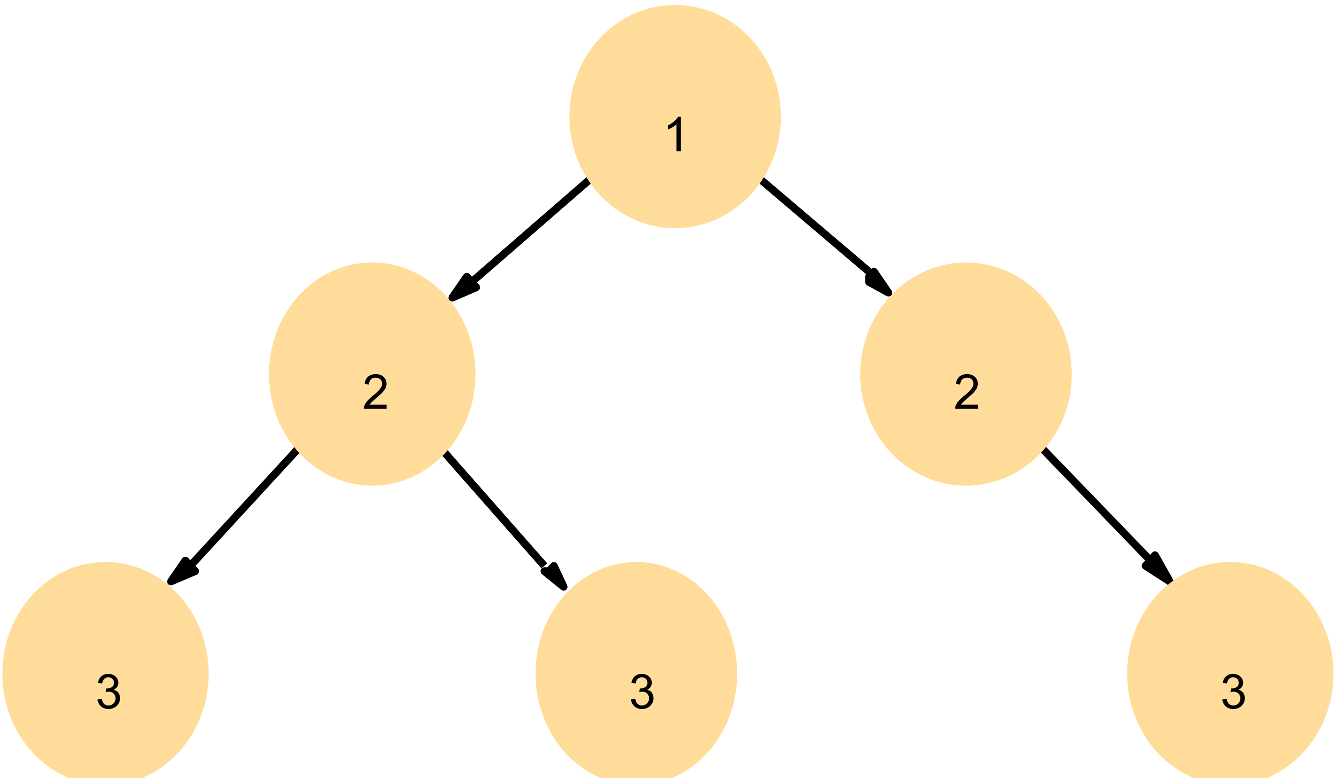
- If the value of the minimum transmission time *min* is known or can be conservatively estimated
  - Earliest time at which S could have placed its time in m_t was *min* after p dispatched m_r
  - Latest point at which it could do so was *min* before m_t arrived at p
  - Time by S's clock when message arrives at p is in range $[t + min, t + T_{round} - min]$
  - Accuracy $\pm(T_{round}/2 - min)$
- Further accuracy can be gained by making multiple requests to S and using the response with the shortest T_{round}.

The Berkeley algorithm

a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock



Network Time Protocol



Note: Arrows denote synchronization control, numbers denote strata.

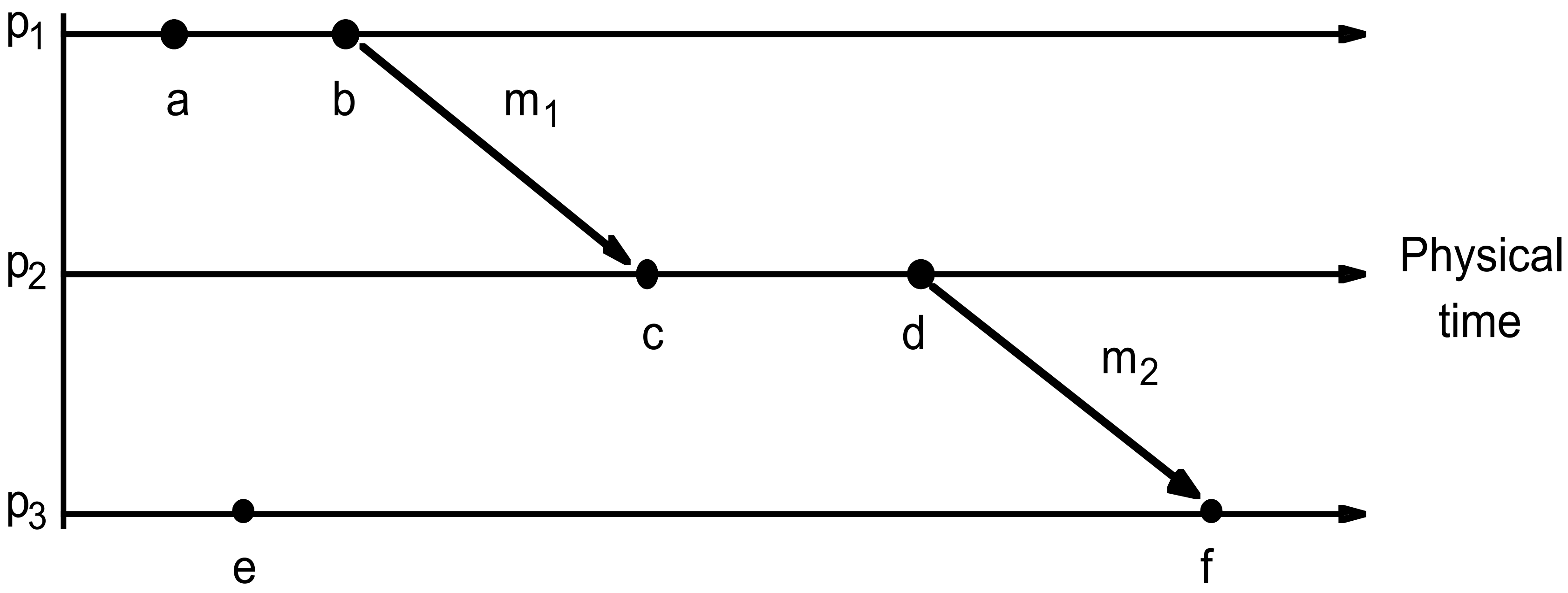
Real synchronization is not perfect

- Clocks never exactly synchronized
- Often inadequate for distributed systems
  - might need to find out the order of any arbitrary pair of events
  - might need millionth-of-a-second precision

Logical time & clocks

- Lamport proposed using logical clocks based upon the “happened before” relation
  - If two events occur at the same process, then they occurred in the order observed
  - Whenever a message is sent between processes, the event of sending occurred before the event of receiving
  - X happened before Y denoted by $X \rightarrow Y$

Events occurring at three processes



Concurrency

- $\rightarrow$ is only a partial-order
- Some events are unrelated
- We say e is concurrent with e' (e || e') if neither e $\rightarrow$ e' nor e' $\rightarrow$ e

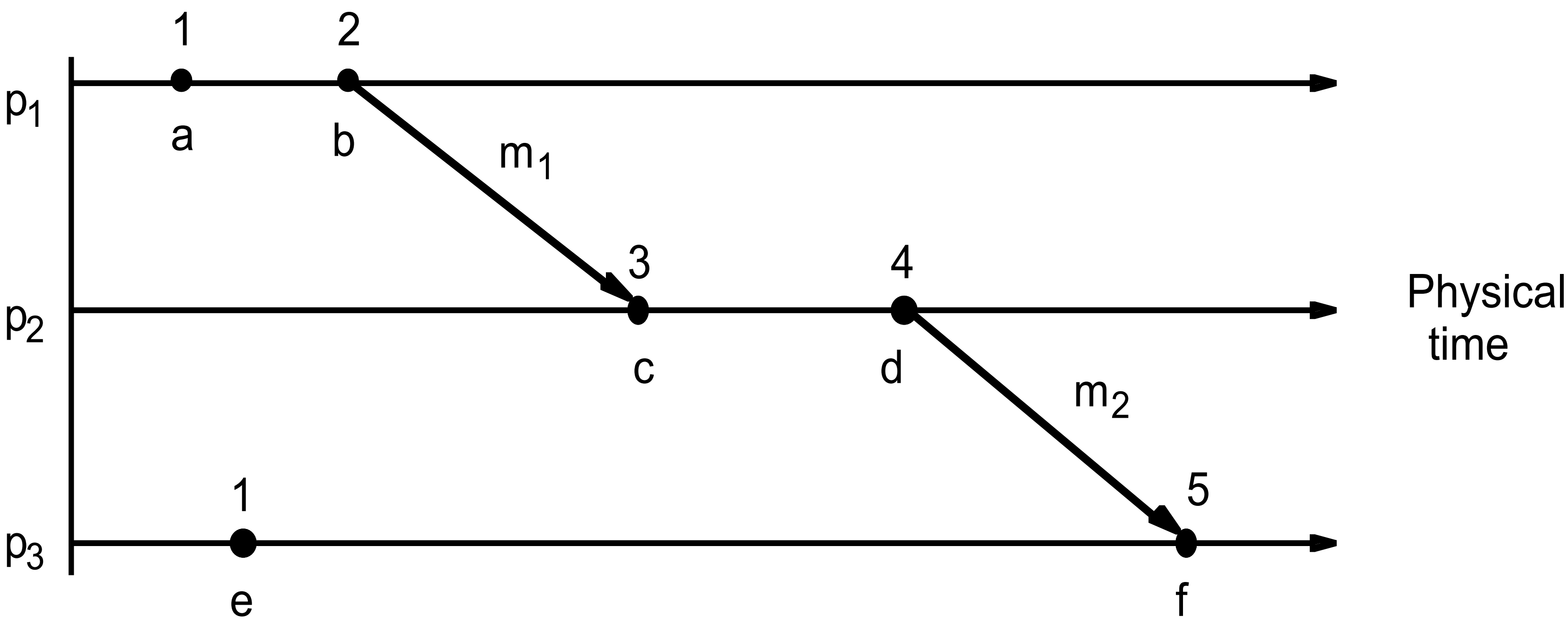
Lamport logical clocks

- Lamport clock orders events consistent with logical “happens before” ordering
- If e $\rightarrow$ e’, then *L(e)* < *L(e’)*

Lamport's algorithm

- Each process has its own logical clock
- Three rules:
  - C_p is incremented before each event at process *p*
  - When process *p* sends a message it piggybacks on it the value C_p
  - On receiving a message (*m*,t), a process *q* computes $C_q = \max(C_q, t)$ and then applies the first rule before timestamping the receive event

Lamport timestamps for the events



Totally ordered logical clocks

- Lamport logical clocks only impose partial ordering
- For total order, use (T_a, P_a) where P_a is process id
- $(T_a, P_a) < (T_b, P_b)$ if and only if either
  $T_a < T_b$ or $(T_a = T_b$ and $P_a < P_b)$
- This ordering has no physical significance, but it is sometime useful, e.g., to break a tie between two processes trying to enter a critical section

Vector timestamps

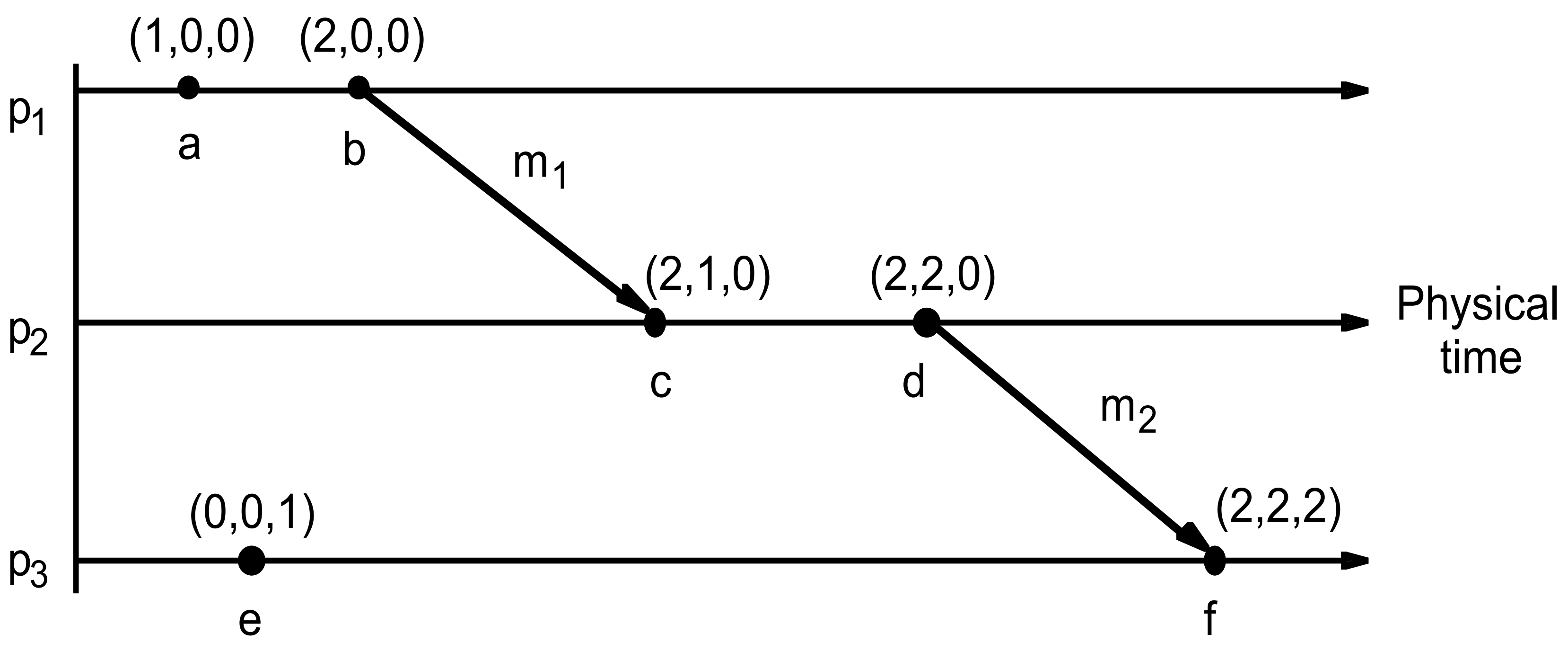
- Shortcoming of Lamport's clocks:
    - if $L(e) < L(f)$, we **cannot** conclude that $e \rightarrow f$

- Vector timestamps
    - A process keeps a vector of integer clocks, one for each process
    - Like Lamport timestamps, processes piggyback vector timestamps on messages they send each other

Assigning vector timestamps

- Initially, $\mathbf{V}_i[j] = \mathbf{0}$, for $i, j = 1, 2, \ldots, N$.
- Just before process *i* timestamps an event, it sets
  - $\mathbf{V}_i[i] = \mathbf{V}_i[i] + \mathbf{1}$
- Process *i* includes the value $t = \mathbf{V}_i$ in every message it sends.
- On receiving a message at process *i*:
  - $\mathbf{V}_i[i] = \mathbf{V}_i[i] + \mathbf{1}$
  - $\mathbf{V}_i[j] = \mathbf{max}(\mathbf{V}_i[j], \mathbf{t}[j])$ **for** $j \neq i$

Vector timestamps for the events



Inferring happened-before relationship

- $V = W$ *iff* $V[j] = W[j]$ for $j = 1, 2, \ldots, N$
- $V \leq W$ *iff* $V[j] \leq W[j]$ for $j = 1, 2, \ldots, N$
- $V < W$ *iff* $V \leq W$ and $V \neq W$
  - $V \rightarrow W$, V happened before W
- If neither $V \leq W$ nor $W \leq V$
  - V and W are concurrent

Reading

- Sections 6.1 and 6.2 of TBook
- Sections14.1-14.4 of CBook