416 Distributed Systems

Time Synchronization

(Part 2: Lamport and vector clocks)

Jan 27, 2016

Important Lessons

- Clocks on different systems will always behave differently
  - Skew and drift between clocks
- Time disagreement between machines can result in undesirable behavior
- Clock synchronization
  - Rely on a time-stamped network messages
  - Estimate delay for message transmission
  - Can synchronize to UTC or to local source
  - Clocks never exactly synchronized
- Often inadequate for distributed systems
  - might need totally-ordered events
  - might need millionth-of-a-second precision

Today's Lecture

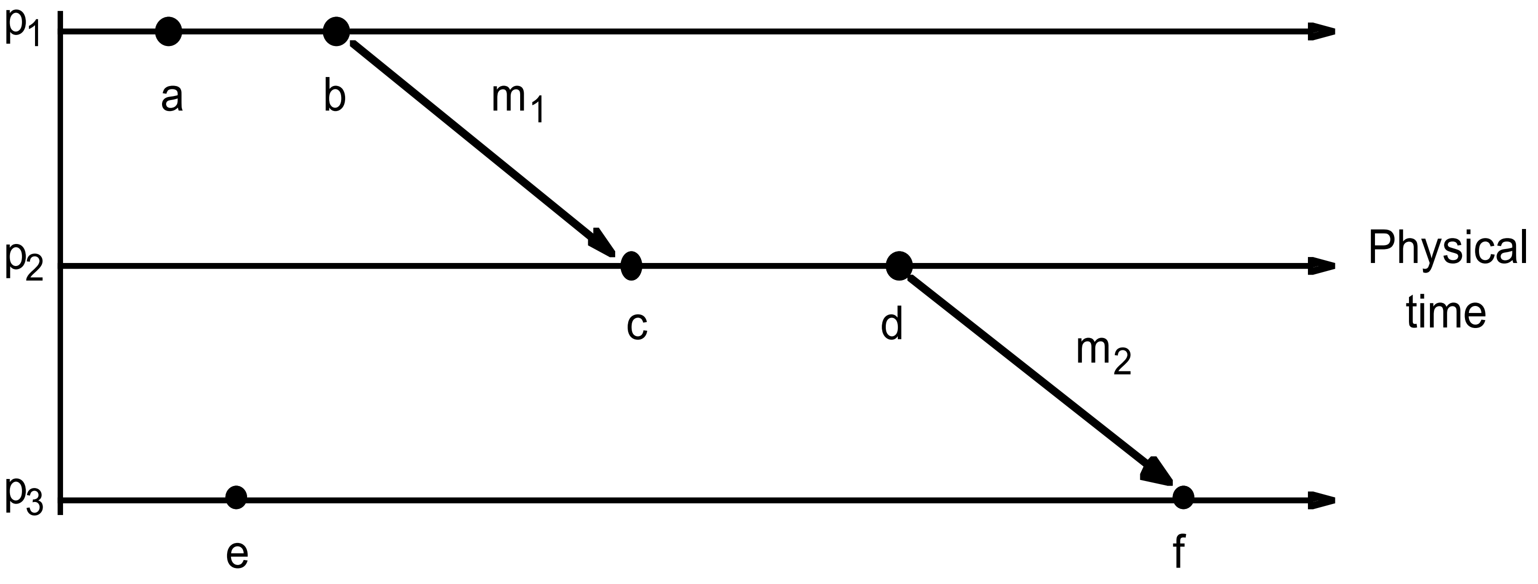
- Need for time synchronization
- Time synchronization techniques
- Lamport Clocks
- Vector Clocks

Logical time

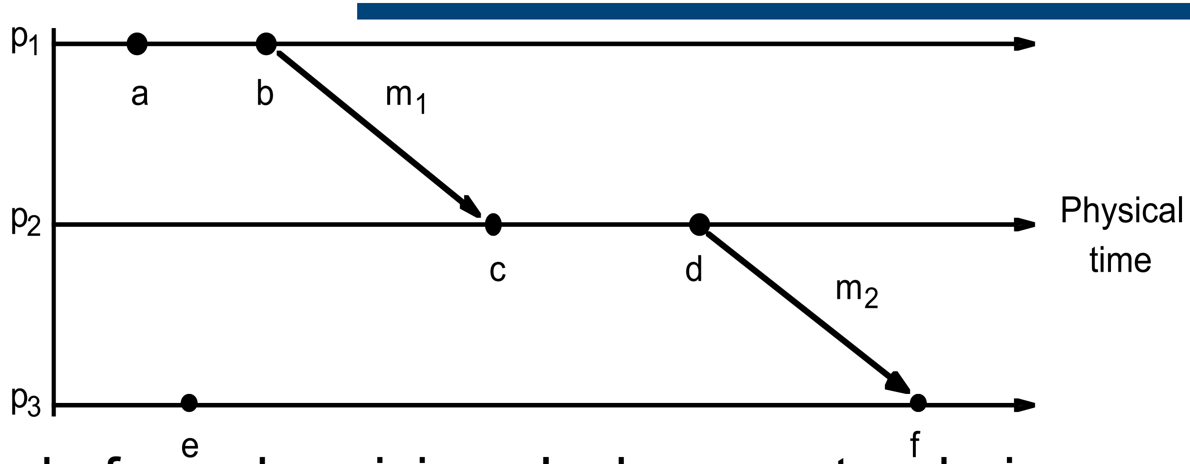
- Capture just the “happens before” relationship between events
  - Discard the infinitesimal granularity of time
  - Corresponds roughly to causality

Logical time and logical clocks (Lamport 1978)

- Events at three processes



Logical time and logical clocks (Lamport 1978)

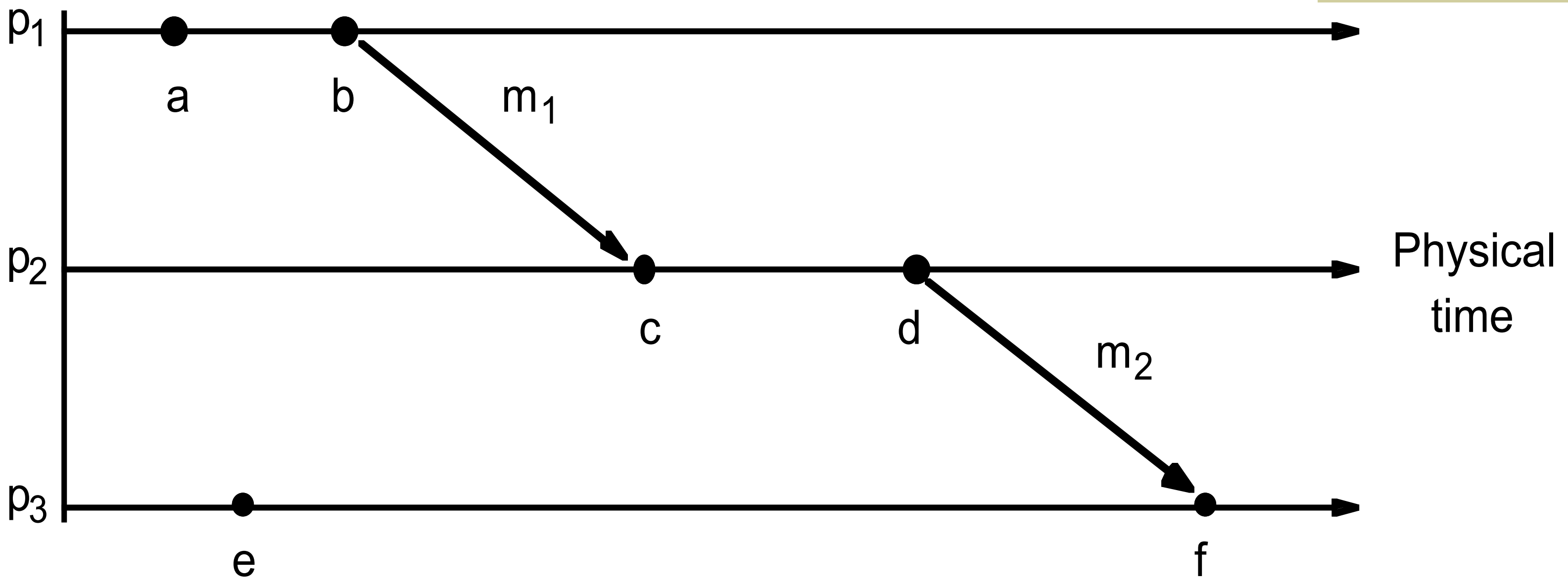


- Instead of synchronizing clocks, event ordering can be used

1. If two events occurred at the same process p_i (i = 1, 2, … N) then they occurred in the order observed by p_i, that is the definition of: $\rightarrow_i$
2. When a message, m is sent between two processes, send(m) happens before receive(m)
3. The happened before relation is transitive

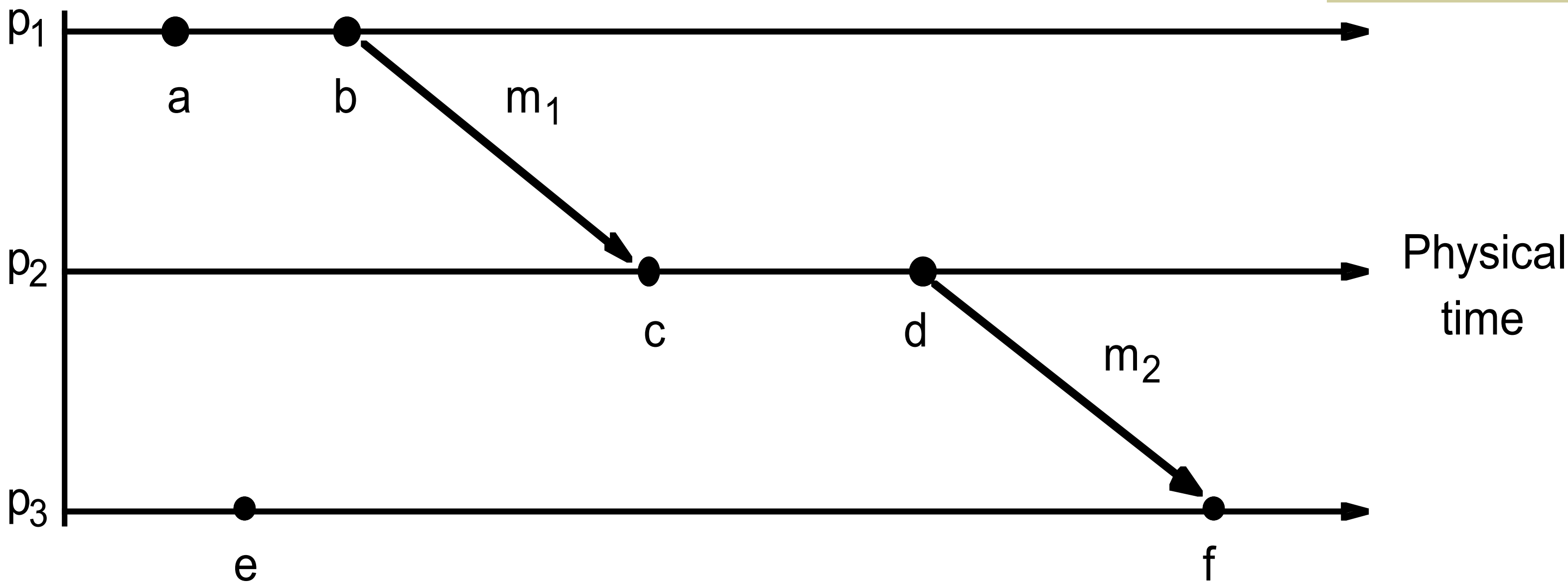
- The happened before relation is the relation of causal ordering

Logical time and logical clocks (Lamport 1978)



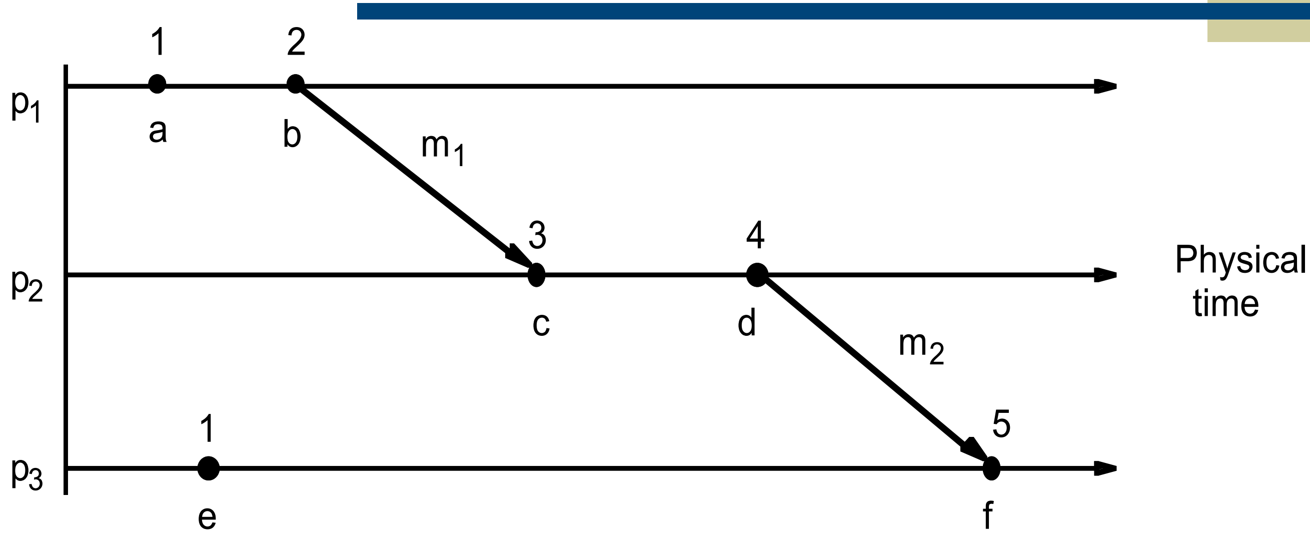
- $a \rightarrow b$ (at p_1) $c \rightarrow d$ (at p_2)
- $b \rightarrow c$ because of m_1
- also $d \rightarrow f$ because of m_2

Logical time and logical clocks (Lamport 1978)



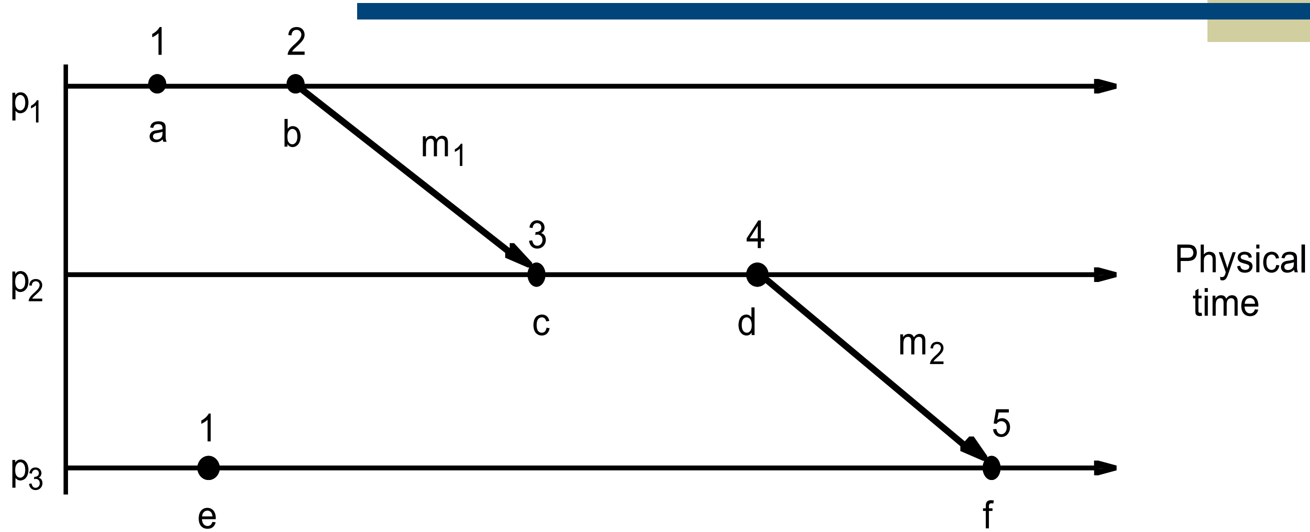
- Not all events are related by →
- Consider *a* and *e* (different processes and no chain of messages to relate them)
  - they are not related by → ; they are said to be concurrent
  - written as *a* || *e*

Lamport Clock (1)



- A logical clock is a monotonically increasing software counter
  - It need not relate to a physical clock.
- Each process p_i has a logical clock, L_i which can be used to apply logical timestamps to events
  - Rule 1: L_i is incremented by 1 before each event at process p_i
  - Rule 2:
    - (a) when process p_i sends message *m*, it piggybacks $t = L_i$
    - (b) when p_j receives *(m,t)* it sets $L_j := max(L_j, t)$ and applies rule 1 before timestamping the event *receive* (*m)*

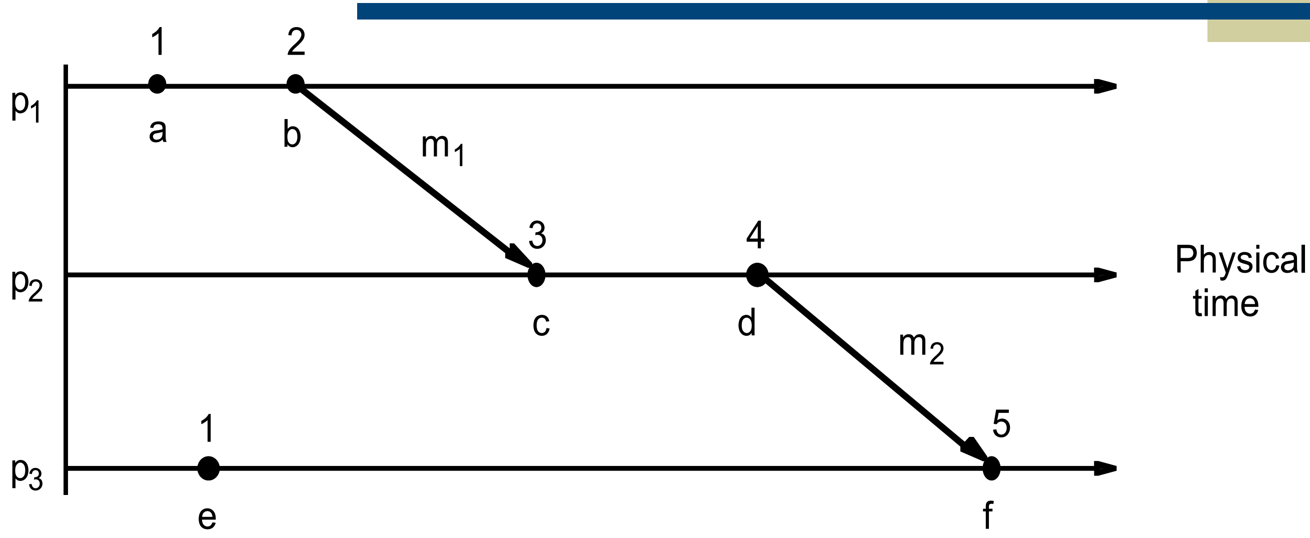
Lamport Clock (1)



- each of p_1, p_2, p_3 has its logical clock initialised to zero,
- the clock values are those immediately after the event.
- e.g. 1 for a, 2 for b.

- for m_1, 2 is piggybacked and c gets max(0,2)+1 = 3

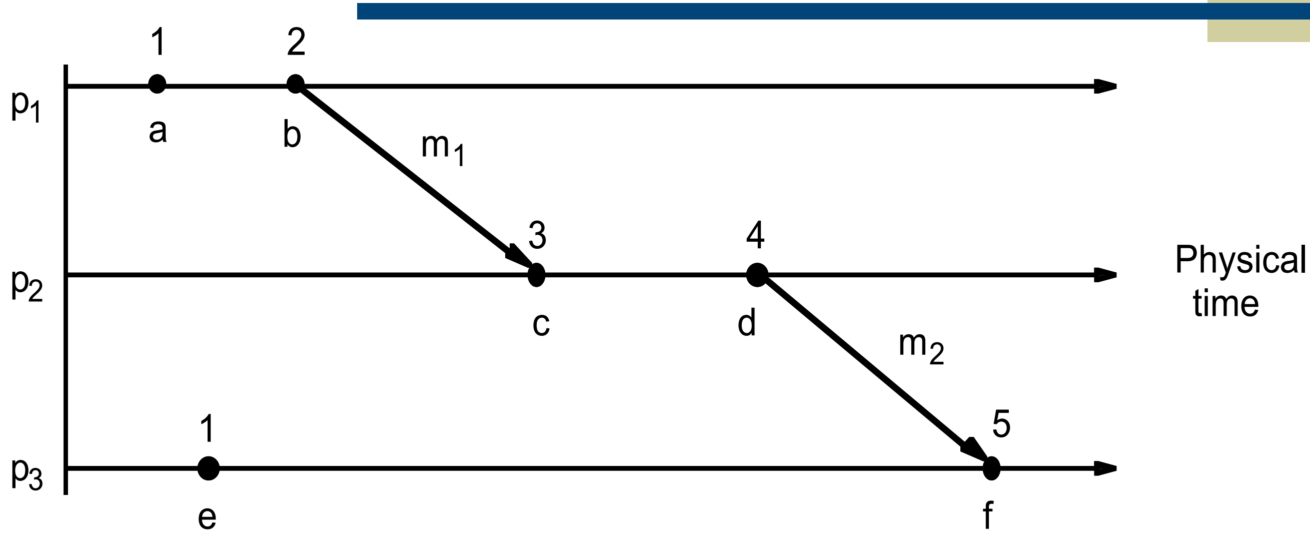
Lamport Clock (1)



- $e \rightarrow e'$ implies $L(e) < L(e')$

- The converse is not true, that is $L(e) < L(e')$ does not imply $e \rightarrow e'$. What's an example of this above?

Lamport Clock (1)



- $e \rightarrow e'$ implies $L(e)<L(e')$
- The converse is not true, that is $L(e)<L(e')$ does not imply $e \rightarrow e'$
  - e.g. $L(b) > L(e)$ but $b \parallel e$

Lamport logical clocks

- Lamport clock L orders events consistent with logical "happens before" ordering
  - If $e \rightarrow e'$, then $L(e) < L(e')$
- But not the converse
  - $L(e) < L(e')$ does not imply $e \rightarrow e'$

- Similar rules for concurrency
  - $L(e) = L(e')$ implies $e \| e'$ (for distinct e,e')
  - $e \| e'$ does not imply $L(e) = L(e')$
  - i.e., Lamport clocks arbitrarily order some concurrent events

Total-order Lamport clocks

- Many systems require a total-ordering of events, not a partial-ordering
- Use Lamport's algorithm, but break ties using the process ID; one example scheme:
  - $L(e) = M * L_i(e) + i$
    - M = maximum number of processes
    - i = process ID

Today's Lecture

- Need for time synchronization
- Time synchronization techniques
- Lamport Clocks
- Vector Clocks

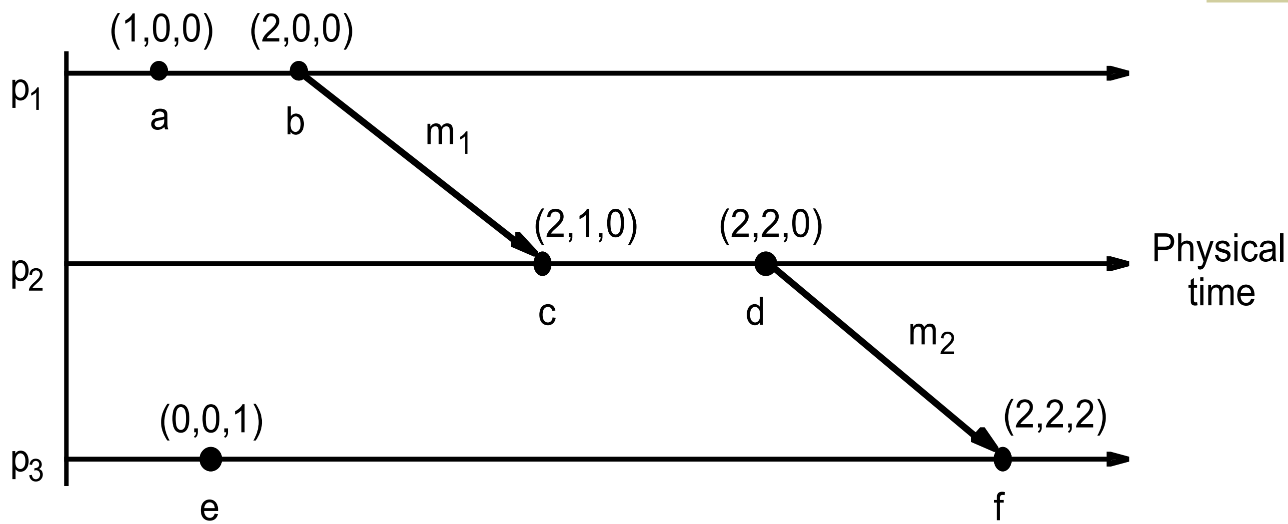
Vector Clocks

- Vector clocks overcome the shortcoming of Lamport logical clocks
  - *L*(e) < *L*(*e*') does not imply *e* happened before *e*'
- Goal
  - Want ordering that matches causality
  - V(e) < V(e') if and only if e → e'
- Method
  - Label each event by vector V(e) [c_1, c_2 …, c_n]
    - c_i = # events in process i that causally precede e

Vector Clock Algorithm

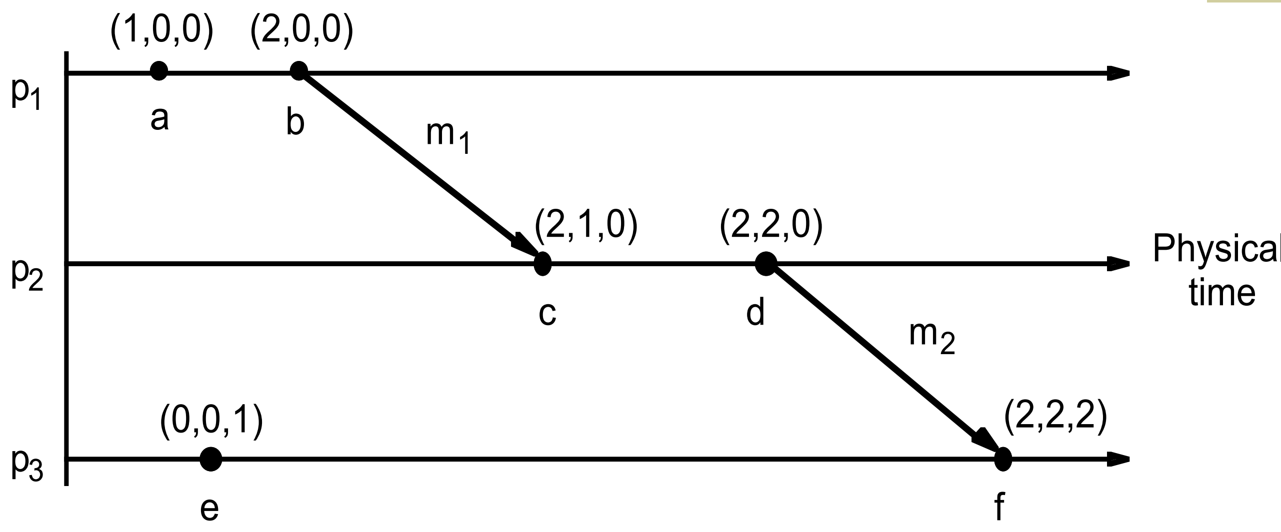
- Initially, all vectors [0,0,…,0]
- For event on process i, increment own c_i
- Label message sent with local vector
- When process j receives message with vector $[d_1, d_2, \ldots, d_n]$:
  - Set local each local entry k to $\max(c_k, d_k)$
  - Increment value of c_j

Vector Clocks



- At p_1
  - *a occurs at* (1,0,0); *b* occurs at (2,0,0)
  - piggyback (2,0,0) on m_1
- At p_2 on receipt of m_1 use *max* ((0,0,0), (2,0,0)) = (2, 0, 0) and add 1 to own element = (2,1,0)
- Meaning of =, <=, max etc for vector timestamps
  - compare elements pairwise

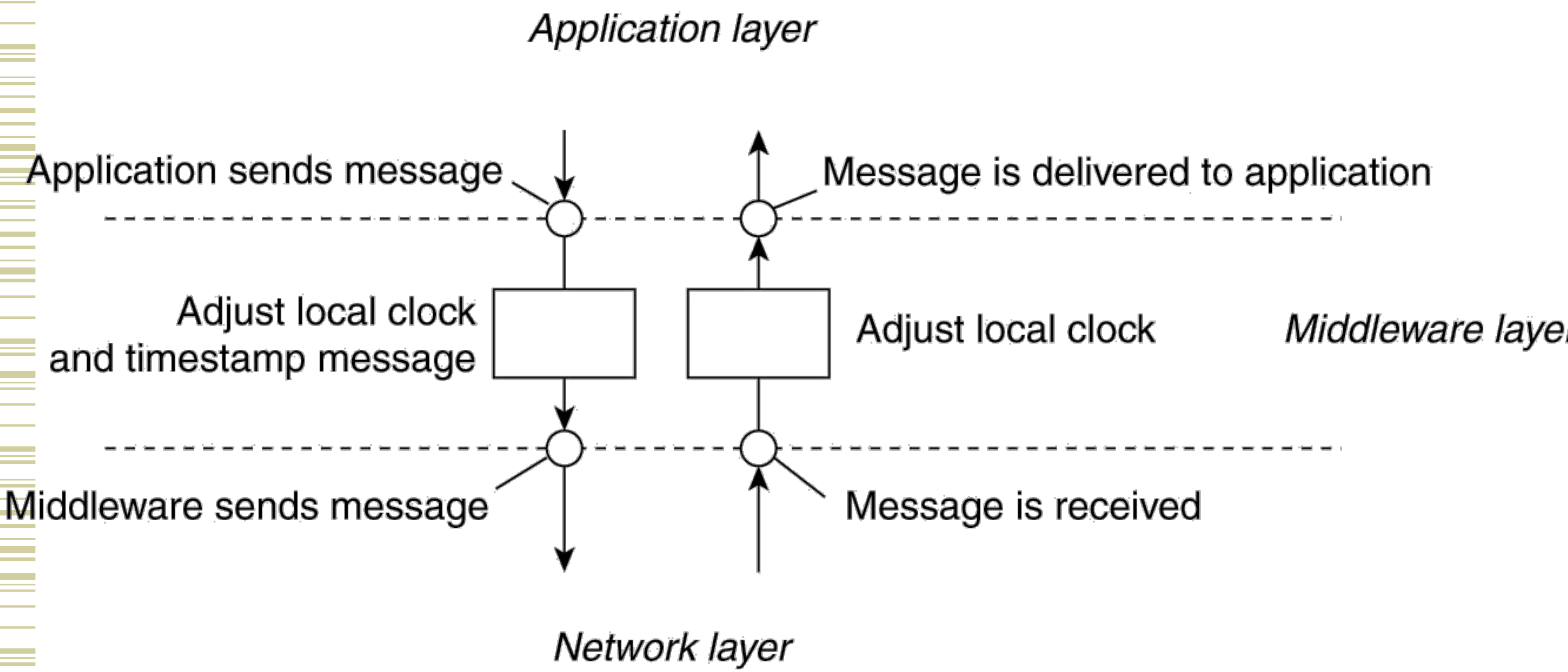
Vector Clocks



- Note that e →e' implies V(e)<V(e'). The converse is also true
- Can you see a pair of parallel events?
  - *c* || *e* (parallel) because neither *V*(c) <= *V*(*e*) nor *V*(*e*) <= *V*(*c*)

Implementing logical clocks

- Positioning of logical timestamping in distributed systems.



Distributed time

- Premise
  - The notion of time is well-defined (and measurable) at each single location
  - But the relationship between time at different locations is unclear
    - Can minimize discrepancies, but never eliminate them
- Reality
  - Stationary GPS receivers can get global time with < 1μs error
  - Few systems designed to use this

Important Points

- Physical Clocks
  - Can keep closely synchronized, but never perfect
- Logical Clocks
  - Encode causality relationship
  - Lamport clocks provide only one-way encoding
  - Vector clocks precedence necessary for causality (but not sufficient: could have been caused by some event along the path, not all events)
- Assignment 4 will require you to use vector timestamps compatible with ShiViz:
- http://bestchai.bitbucket.org/shiviz/ (DEMO!)