

Distributed Systems

CS425/ECE428

Jan 27 2022

Instructor: Radhika Mittal

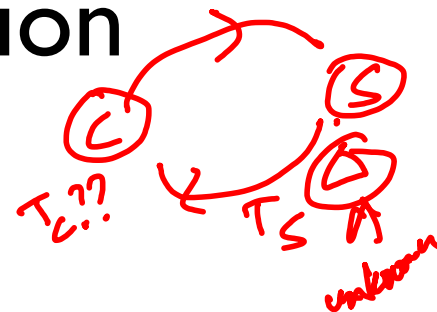
Logistics Related

- Lecture links on website fixed!
 - Was initially pointing to last year's -- logistics differ.
 - Sorry for the confusion and inconvenience.
- HWI has been released today.
 - Four questions in total (each with a few subparts).
 - You should be able to solve the first two questions by the end of today's class.
 - You might need to wait until next week's classes to solve the remaining questions.
- VM clusters have been assigned. Stay tuned for instructions on Campuswire.

Today's agenda

- **Wrap up clock synchronization**
 - Chapter 14.1-14.3
- **Logical Clocks and Timestamps**
 - Chapter 14.4
- **Global State**
 - Chapter 14.5

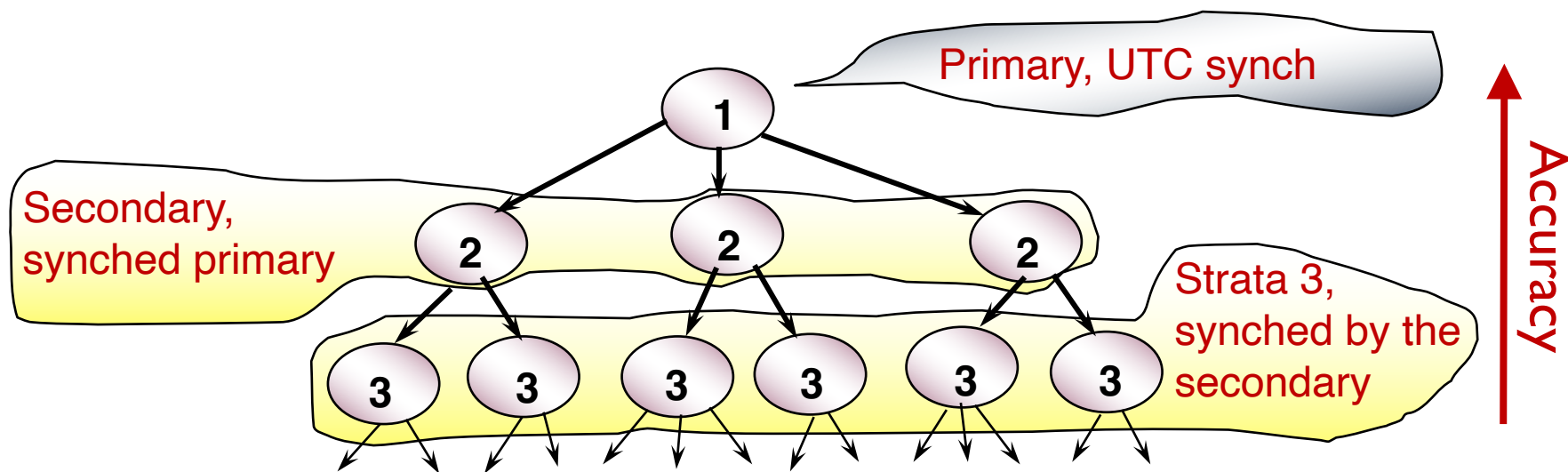
Quick Recap: Clock Synchronization



- Synchronization in synchronous systems:
 - Synchronization bound = $(max - min) / 2$
- Synchronization in asynchronous systems:
 - Cristian Algorithm: Synchronization between a client and a server.
 - Synchronization bound = $(T_{round} / 2) - min \leq T_{round} / 2$
 - Berkeley Algorithm: internal synchronization between clocks.
 - A central server picks the average time and disseminates offsets.

Network Time Protocol

Time service over the Internet for synchronizing to UTC.



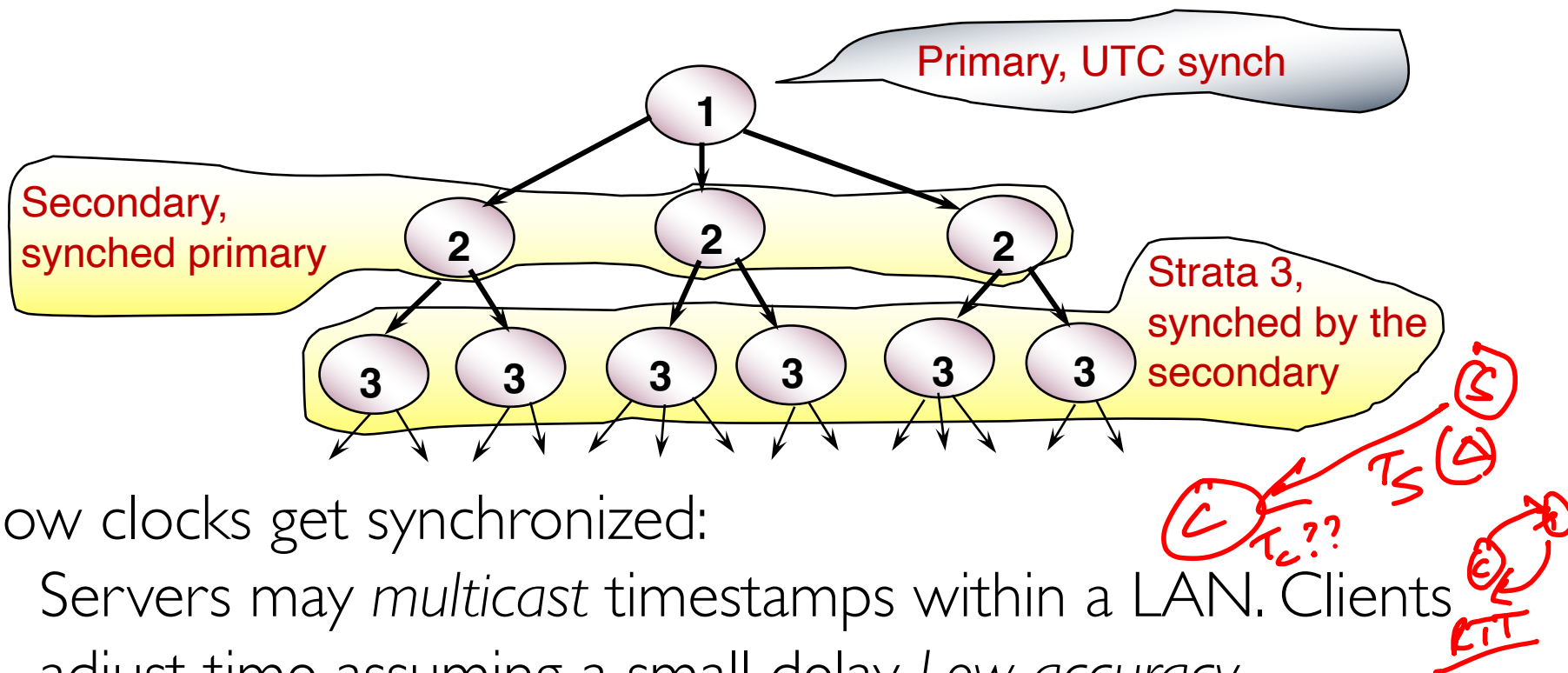
Hierarchical structure for *scalability*.

Multiple lower strata servers for *robustness*.

Authentication mechanisms for *security*.

Statistical techniques for better *accuracy*.

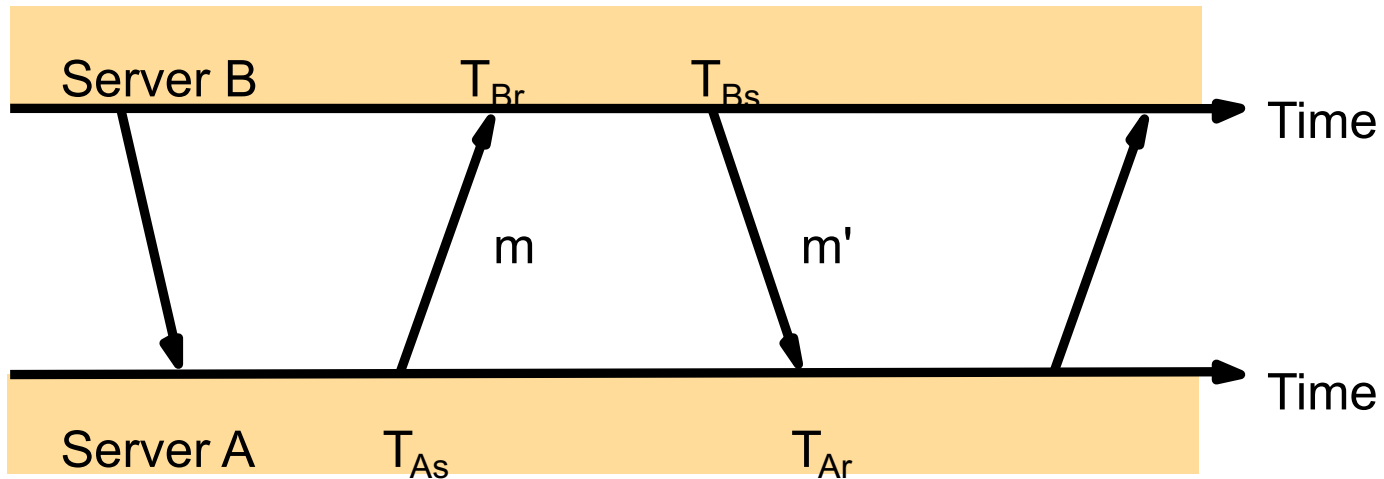
Network Time Protocol



How clocks get synchronized:

- Servers may *multicast* timestamps within a LAN. Clients adjust time assuming a small delay. *Low accuracy.*
- *Procedure-call* (Cristian algorithm). *Higher accuracy.*
- *Symmetric mode* used to synchronize lower strata servers. *Highest accuracy.*

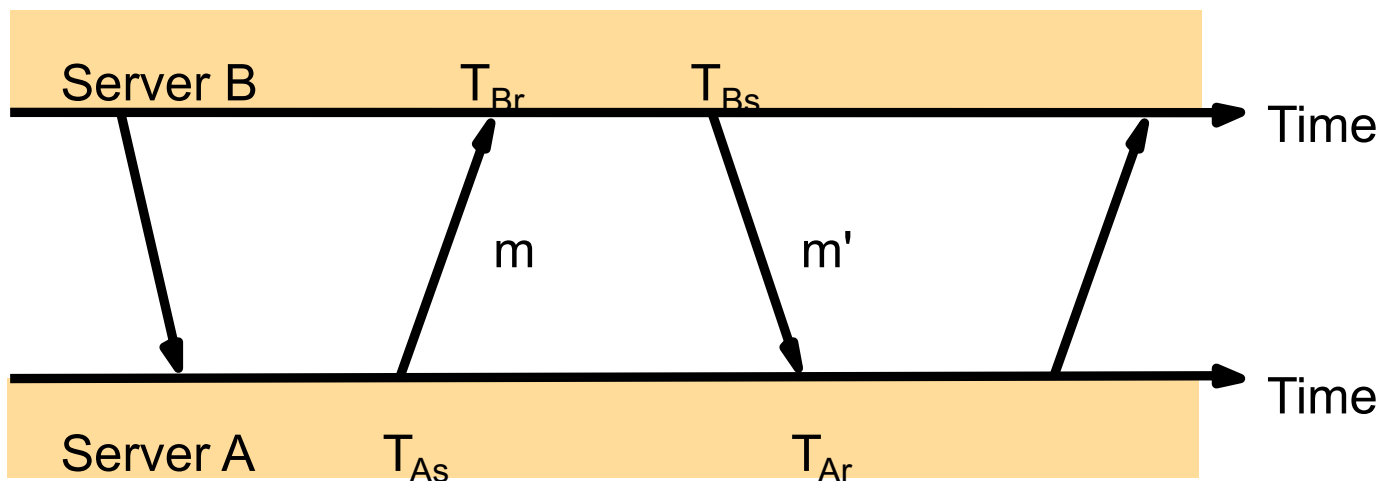
NTP Symmetric Mode



A and B exchange messages and record the send and receive timestamps.

Use these timestamps to compute offset with respect to one another (\mathbf{o}_i).

NTP Symmetric Mode



- t and t' : actual transmission times for m and m' (unknown)
- o : true offset of clock at B relative to clock at A (unknown)
- o_i : estimate of actual offset between the two clocks
- d_i : estimate of accuracy of o_i ; total transmission times for m and m' . $d_i = t + t'$

$$T_{Br} = T_{As} + t + o$$

$$T_{Ar} = T_{Bs} + t' - o$$

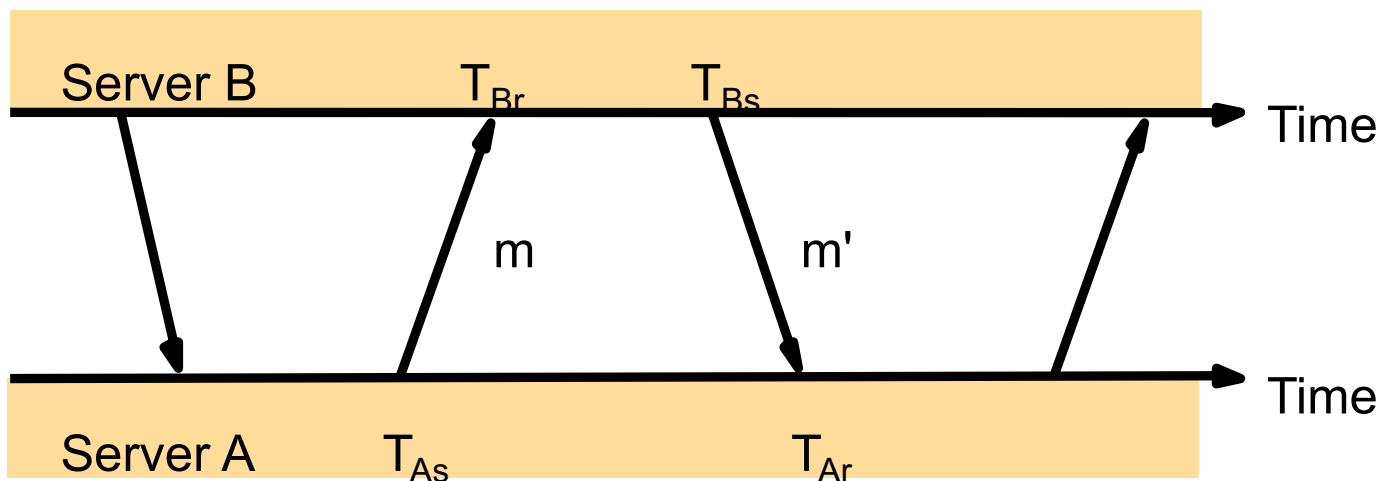
$$o = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) + (t' - t)) / 2$$

$$o_i = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs})) / 2$$

$$o = o_i + (t' - t) / 2$$

$$d_i = t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs})$$

NTP Symmetric Mode



- t and t' : actual transmission times for m and m' (unknown)
- o : true offset of clock at B relative to clock at A (unknown)
- o_i : estimate of actual offset between the two clocks
- d_i : estimate of accuracy of o_i ; total transmission times for m and m' . $d_i = t + t'$

$$T_{Br} = T_{As} + t + o$$

$$T_{Ar} = T_{Bs} + t' - o$$

$$o = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) + (t' - t)) / 2$$

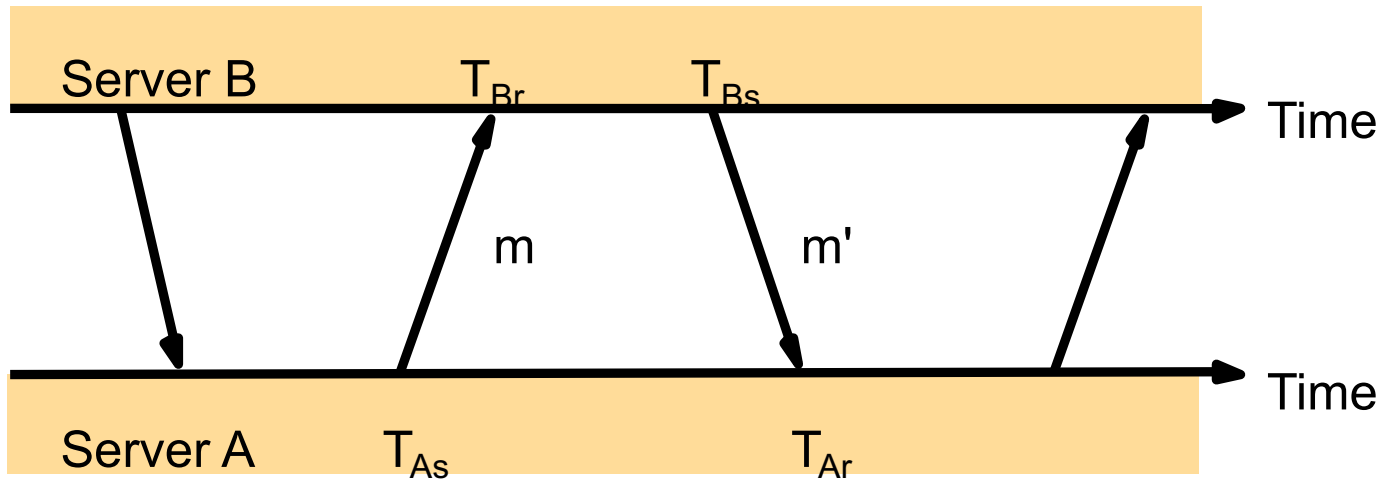
$$o_i = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs})) / 2$$

$$o = o_i + (t' - t) / 2$$

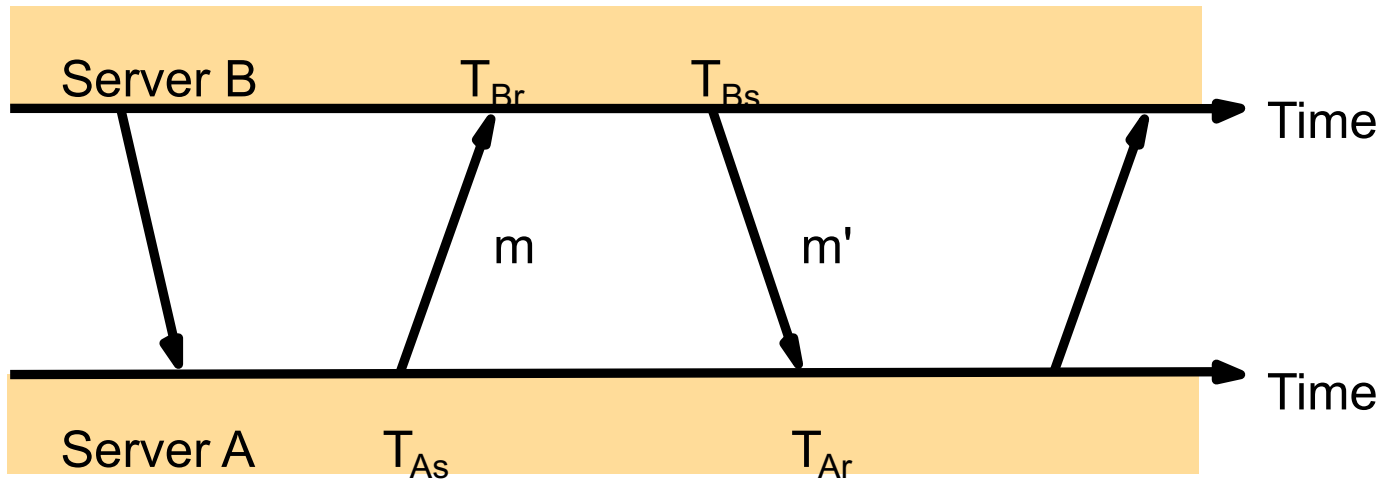
$$d_i = t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs})$$

$$(o_i - d_i / 2) \leq o \leq (o_i + d_i / 2) \text{ given } t, t' \geq 0$$

NTP Symmetric Mode



NTP Symmetric Mode



A and B exchange messages and record the send and receive timestamps.

Use these timestamps to compute offset with respect to one another (\mathbf{o}_i).

A server computes its offset from multiple different sources and adjust its local time accordingly.

Today's agenda

- **Wrap up clock synchronization**
 - Chapter 14.1-14.3
- **Logical Clocks and Timestamps**
 - Chapter 14.4
- **Global State**
 - Chapter 14.5

Event Ordering

- A usecase of synchronized clocks:
 - Reasoning about order of events.
- Can we reason about order of events without synchronized clocks?

Process, state, events

- Consider a system with n processes: $\langle p_1, p_2, p_3, \dots, p_n \rangle$
- Each process p_i is described by its *state* s_i that gets transformed over time.
 - State includes values of all local variables, affected files, etc.
- s_i gets transformed when an *event* occurs.
- Three types of events:
 - Local computation.
 - Sending a message.
 - Receiving a message.

Event Ordering

- Easy to order events within a single process p_i , based on their time of occurrence.
- How do we reason about events across processes?
 - A message must be *sent* before it gets *received* at another process.
- These two notions help define *happened-before* (HB) relationship denoted by \rightarrow .
 - $e \rightarrow e'$ means e *happened before* e' .

Happened-Before Relationship

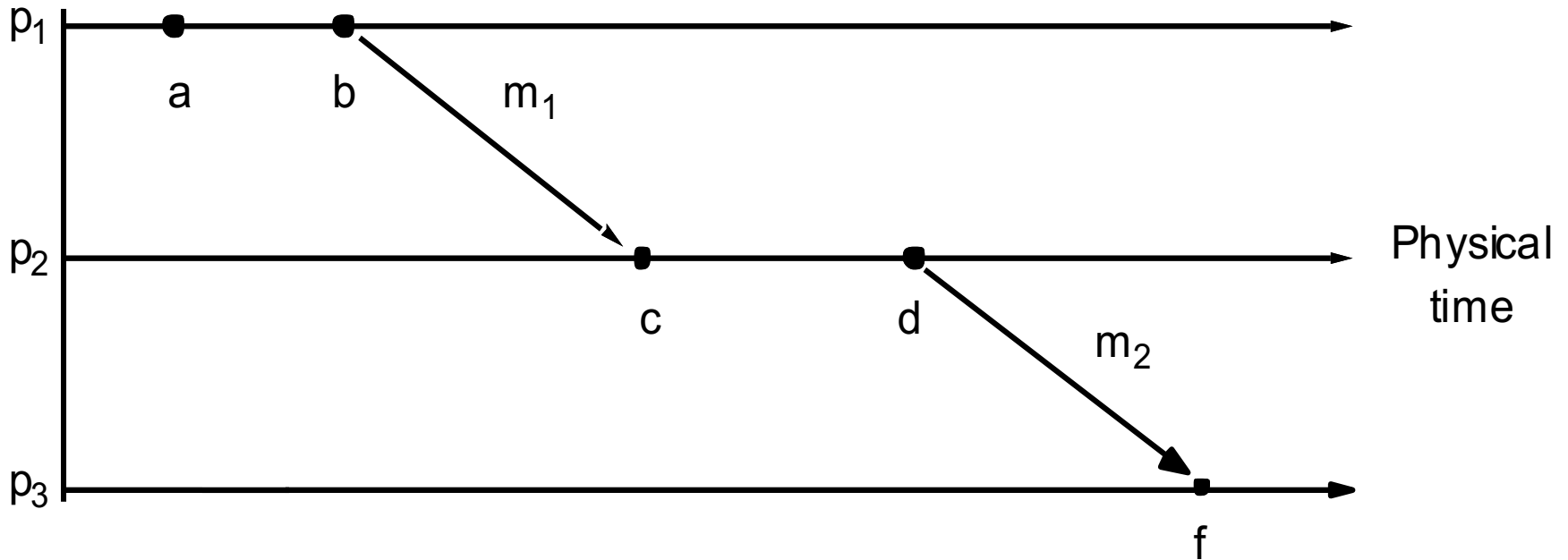
- *Happened-before* (HB) relationship denoted by \rightarrow .
 - $e \rightarrow e'$ means e happened before e' .
 - $e \rightarrow_i e'$ means e happened before e' , as observed by p_i .

- HB rules:
 - If $\exists p_i$, $e \rightarrow_i e'$ then $e \rightarrow e'$. exists
 - For any message m , $\text{send}(m) \rightarrow \text{receive}(m)$
 - If $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$

\exists (there exists)
 \forall (for all)

- Also called “causal” or “potentially causal” ordering.

Event Ordering: Example



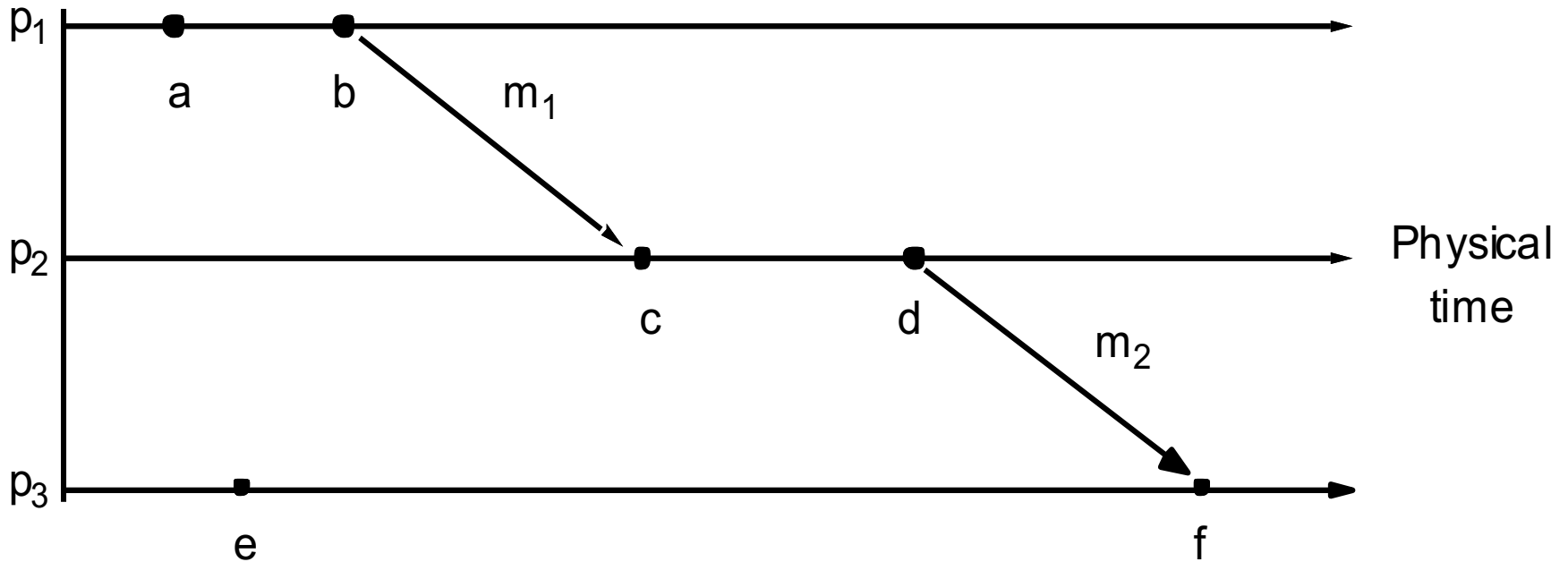
a → b
⇒ a → c

Which event happened first?

a → **b** and **b** → **c** and **c** → **d** and **d** → **f**

a → **b** and **a** → **c** and **a** → **d** and **a** → **f**

Event Ordering: Example



What can we say about e ?

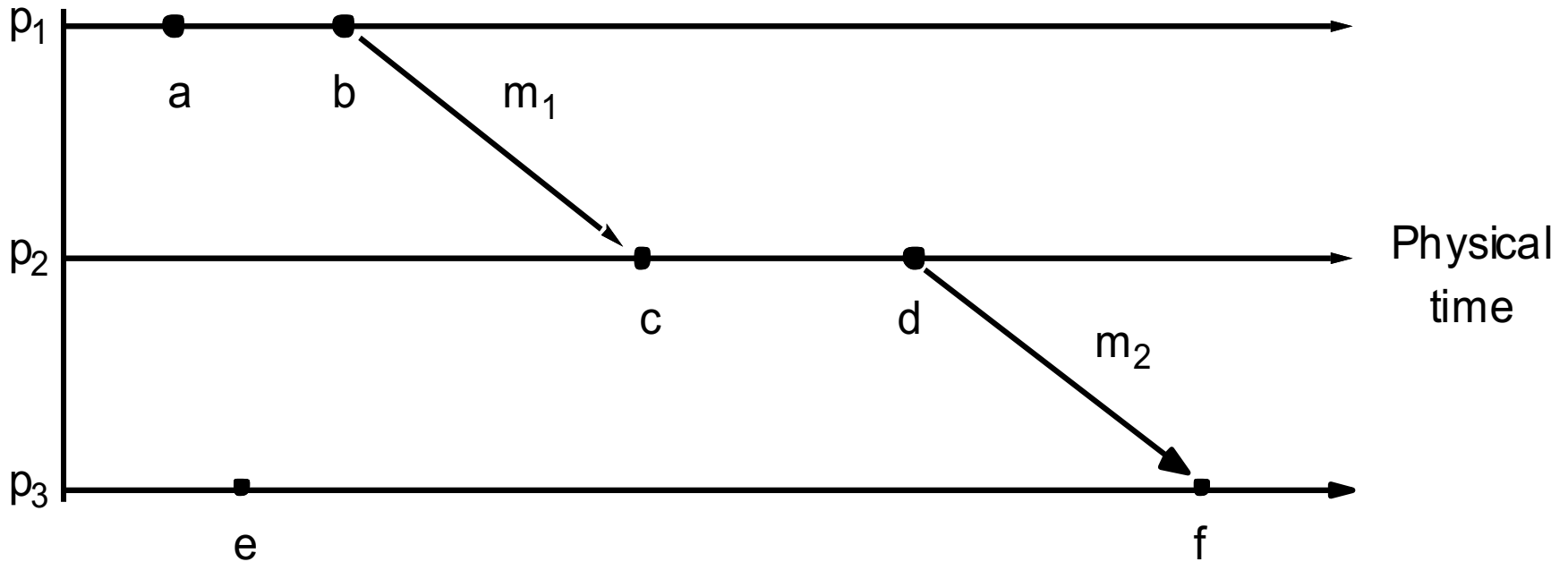
$e \rightarrow f$

$a \not\rightarrow e$ and $e \not\rightarrow a$

$a \parallel e$

a and e are concurrent.

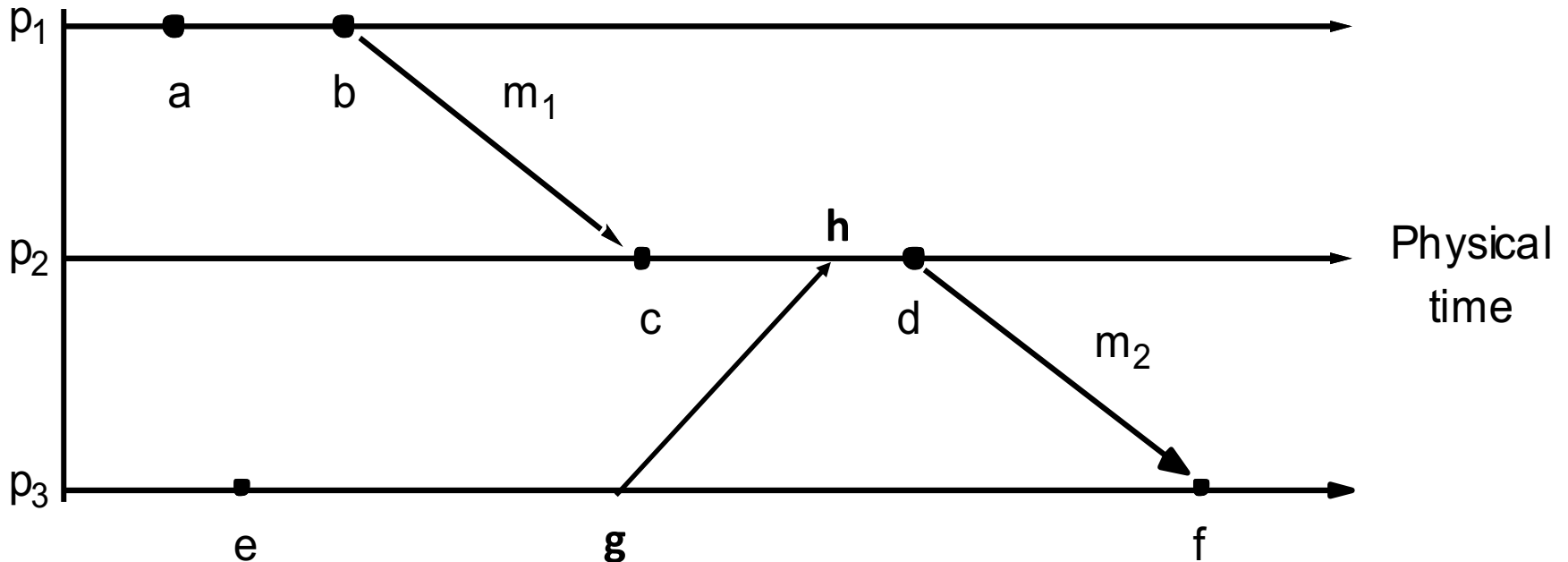
Event Ordering: Example



What can we say about e and d ?

$e \parallel d$

Logical Timestamps: Example



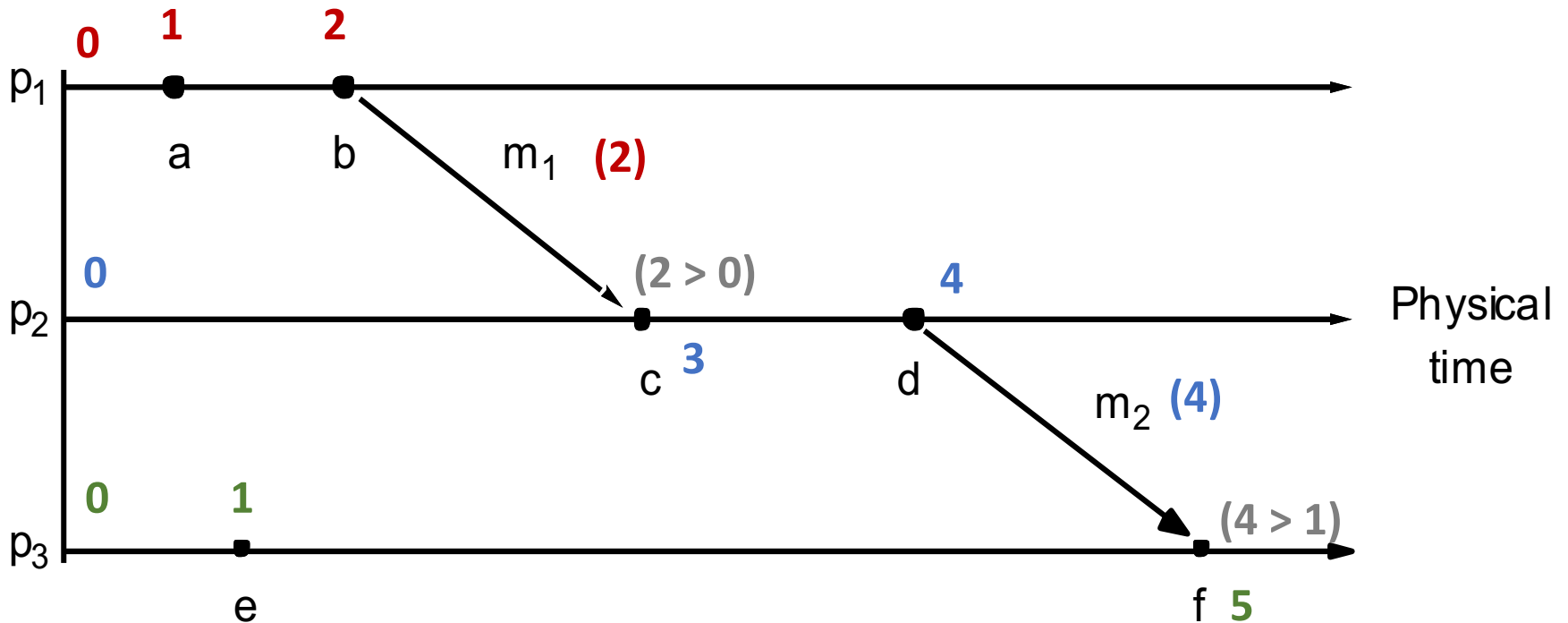
What can we say about e and d ?

$e \rightarrow d$

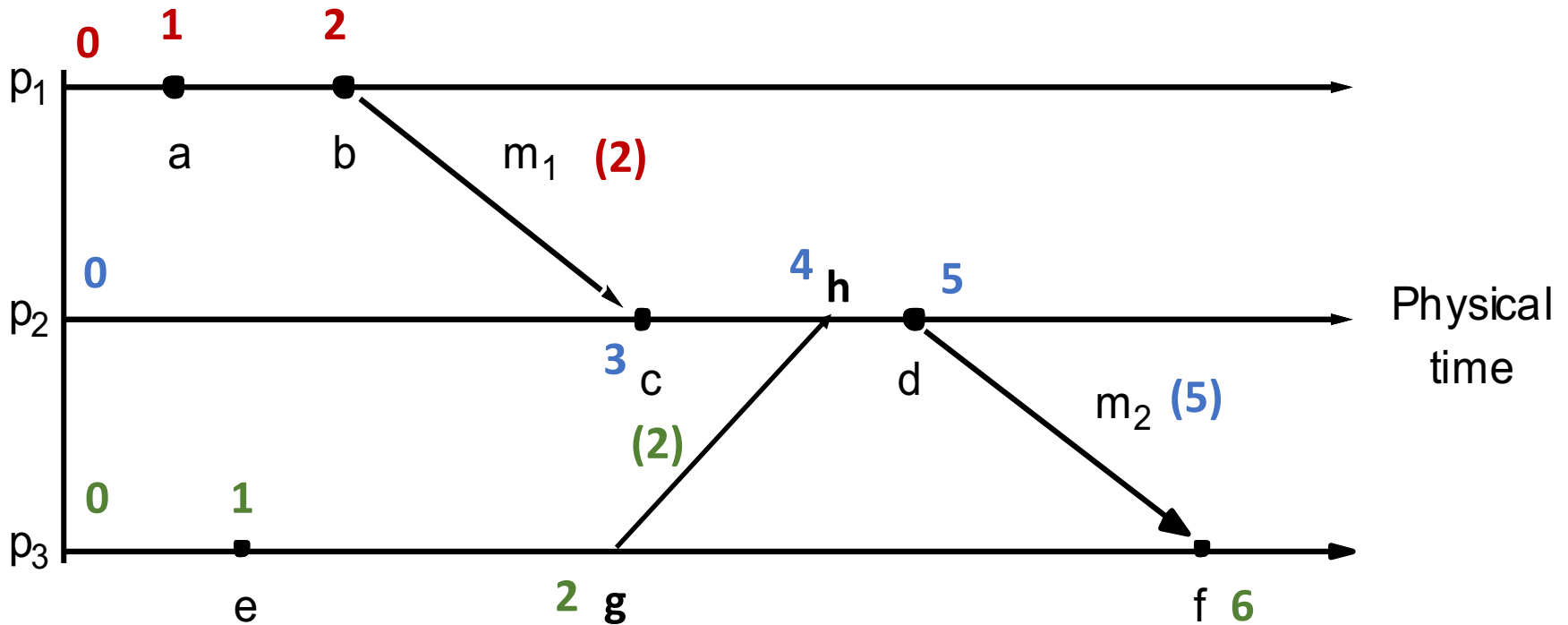
Lamport's Logical Clock

- Logical timestamp for each event that captures the *happened-before* relationship.
- *Algorithm:* Each process p_i
 1. initializes local clock $L_i = 0$.
 2. increments L_i before timestamping each event.
 3. piggybacks L_i when sending a message.
 4. upon receiving a message with clock value t
 - sets $L_i = \max(t, L_i)$
 - increments L_i before timestamping the receive event (as per step 2).

Logical Timestamps: Example



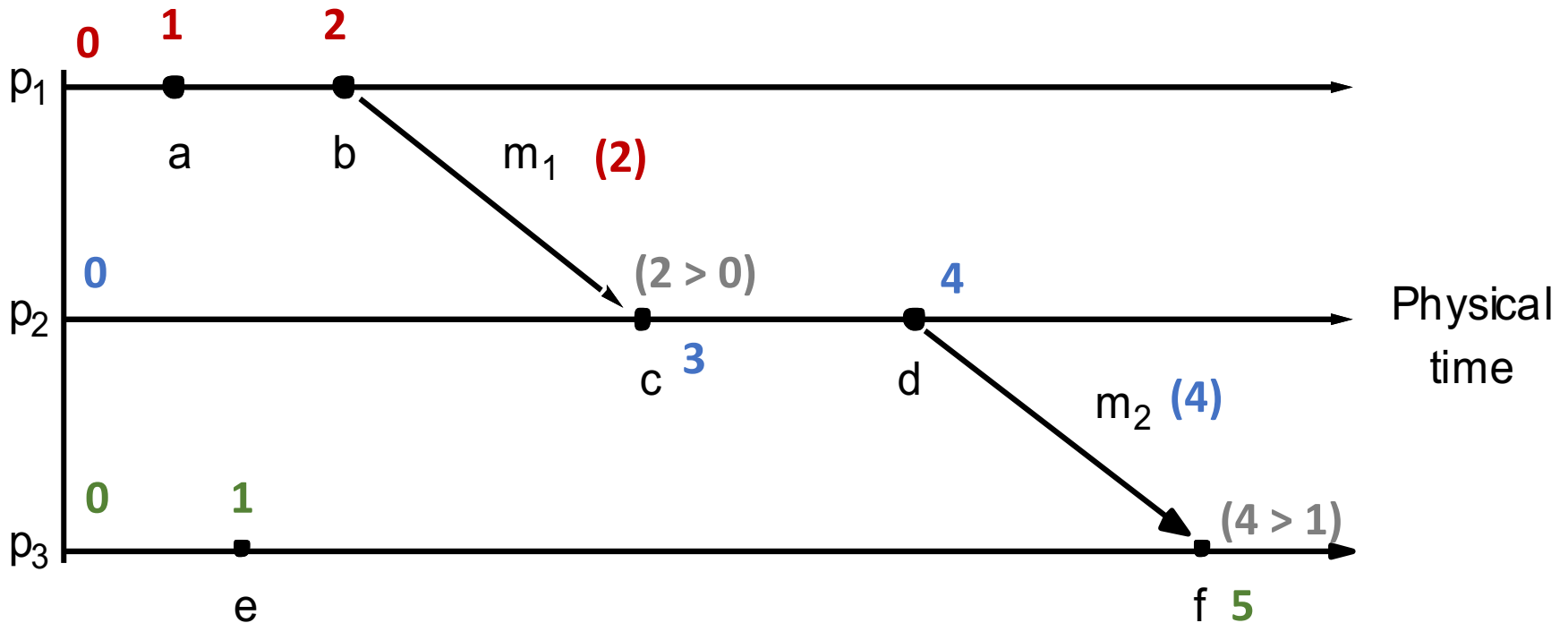
Logical Timestamps: Example



Lamport's Logical Clock

- Logical timestamp for each event that captures the *happened-before* relationship.
- If $e \rightarrow e'$ then $L(e) < L(e')$
- What if $L(e) < L(e')$?
 - We cannot say that $e \rightarrow e'$
 - We can say: $e' \nrightarrow e$
 - Either $e \rightarrow e'$ or $e \parallel e'$

Logical Timestamps: Example



$L(e) < L(d), e \parallel d$

$L(e) < L(f), e \rightarrow f$

Vector Clocks

- Each event associated with a vector timestamp.
- Each process p_i maintains vector of clocks V_i
- The size of this vector is the same as the no. of processes.
 - $V_i[j]$ is the clock for process p_j as maintained by p_i
- Algorithm: each process p_i :

Vector Clocks

- Each event associated with a vector timestamp.
- Each process p_i maintains vector of clocks V_i
- The size of this vector is the same as the no. of processes.
 - $V_i[j]$ is the clock for process p_j as maintained by p_i
- Algorithm: each process p_i :
 1. initializes local clock $V_i[j] = 0$

Vector Clocks

- Each event associated with a vector timestamp.
- Each process p_i maintains vector of clocks V_i
- The size of this vector is the same as the no. of processes.
 - $V_i[j]$ is the clock for process p_j as maintained by p_i
- Algorithm: each process p_i :
 1. initializes local clock $V_i[i] = 0$
 2. increments $V_i[i]$ before timestamping each event.

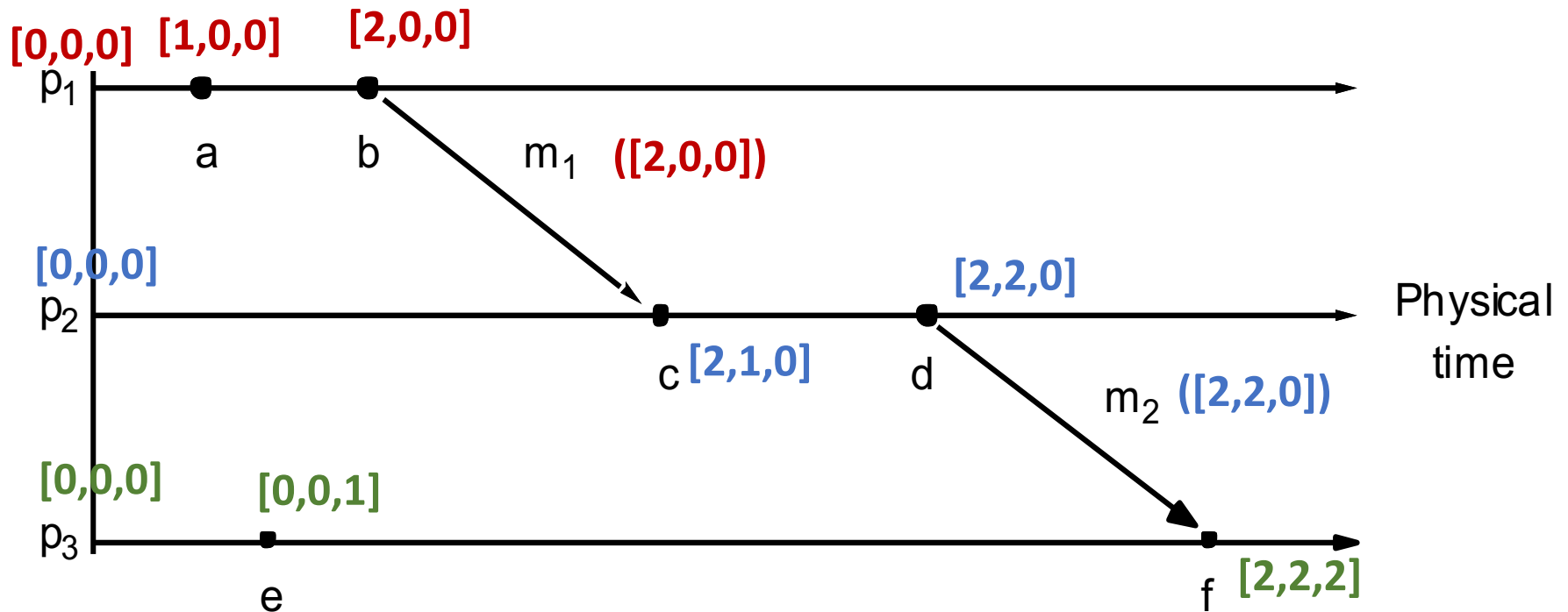
Vector Clocks

- Each event associated with a vector timestamp.
- Each process p_i maintains vector of clocks V_i
- The size of this vector is the same as the no. of processes.
 - $V_i[j]$ is the clock for process p_j as maintained by p_i
- Algorithm: each process p_i :
 1. initializes local clock $V_i[i] = 0$
 2. increments $V_i[i]$ before timestamping each event.
 3. piggybacks V_i when sending a message.

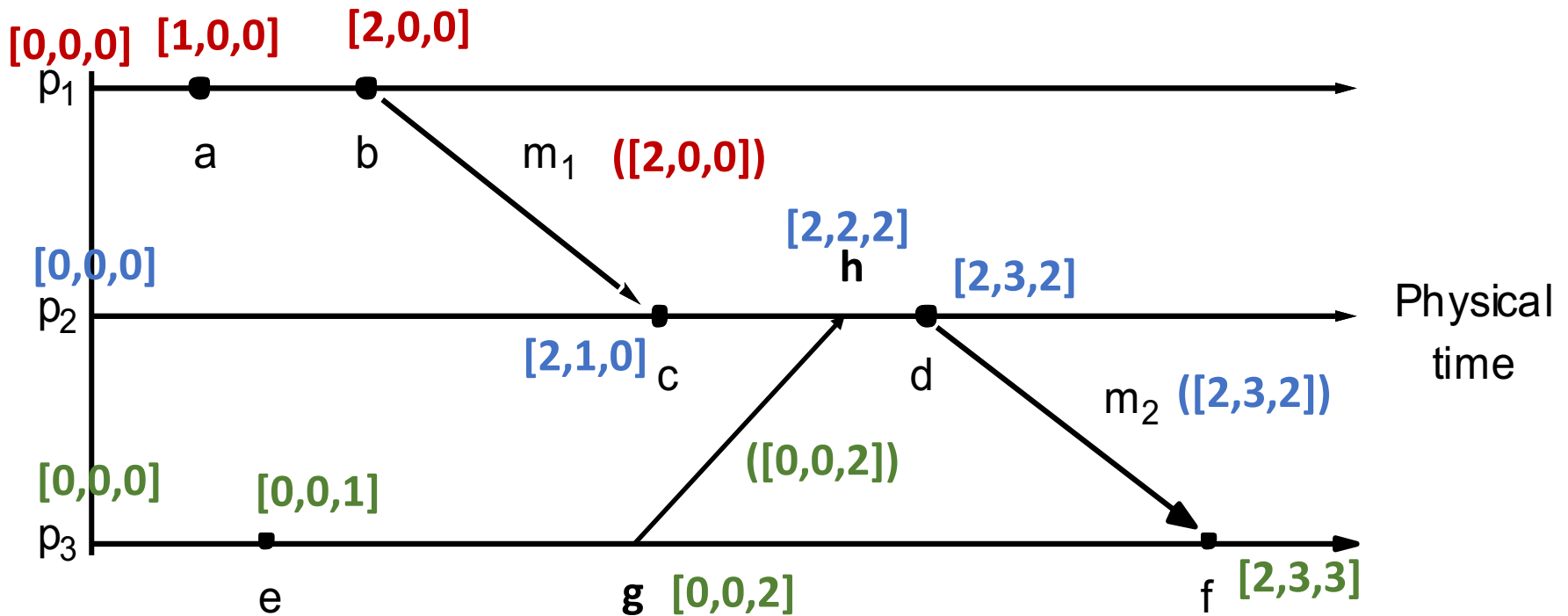
Vector Clocks

- Each event associated with a vector timestamp.
- Each process p_i maintains vector of clocks V_i
- The size of this vector is the same as the no. of processes.
 - $V_i[j]$ is the clock for process p_j as maintained by p_i
- Algorithm: each process p_i :
 1. initializes local clock $V_i[i] = 0$
 2. increments $V_i[i]$ before timestamping each event.
 3. piggybacks V_i when sending a message.
 4. upon receiving a message with vector clock value v
 - sets $V_i[j] = \max(V_i[j], v[j])$ for all $j=1 \dots n$.
 - increments $V_i[i]$ before timestamping receive event (as per step 2).

Vector Timestamps: Example



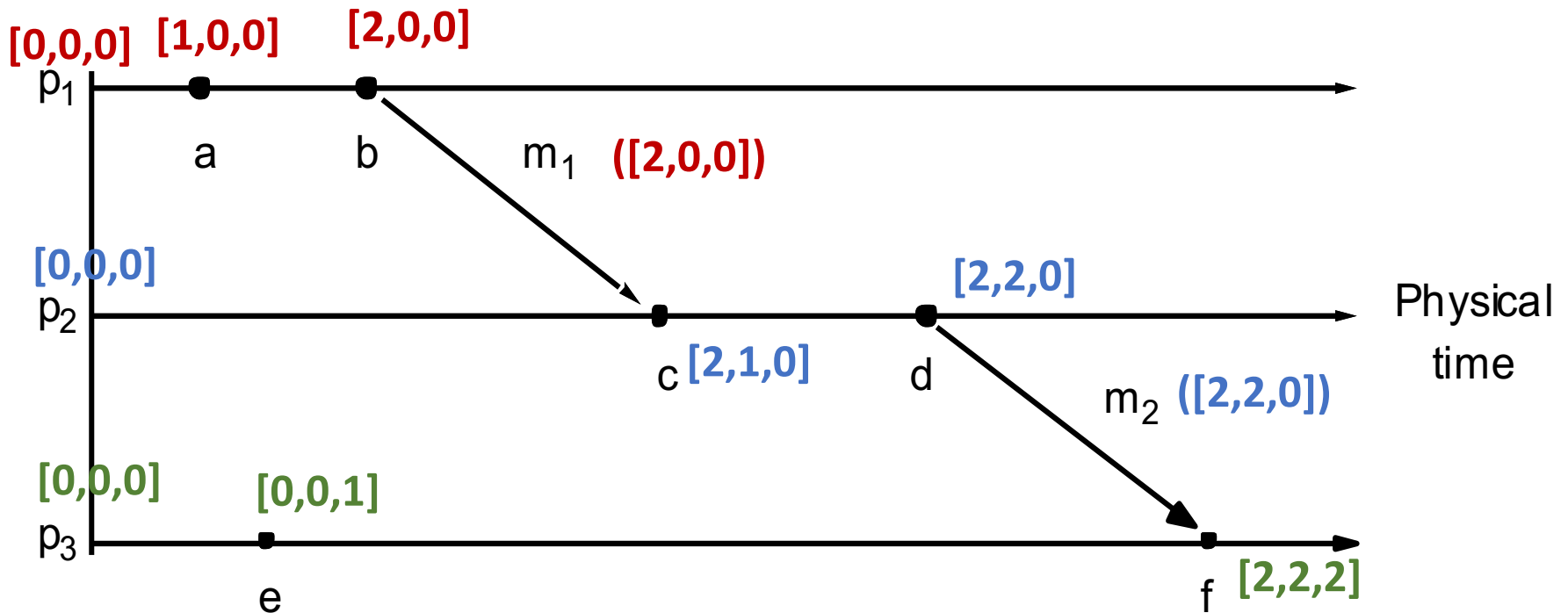
Vector Timestamps: Example



Comparing Vector Timestamps

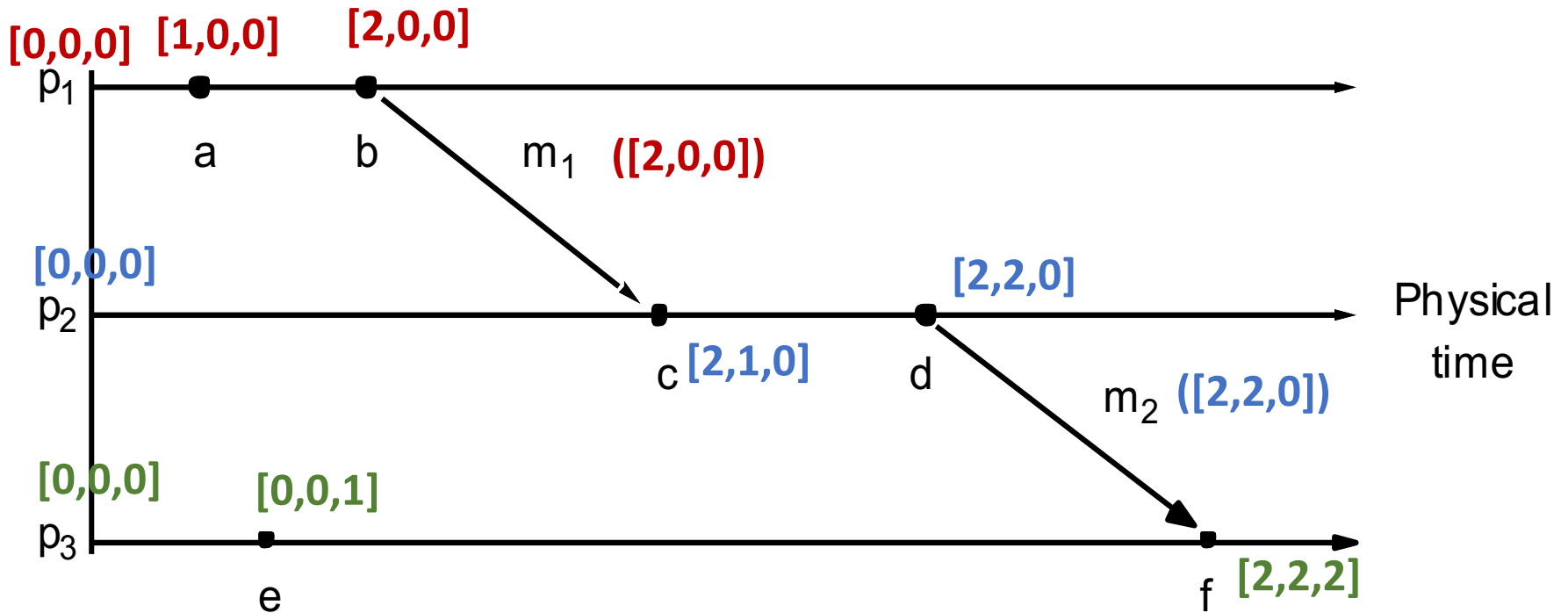
- Let $V(e) = V$ and $V(e') = V'$
- $V = V'$, iff $V[i] = V'[i]$, for all $i = 1, \dots, n$
- $V \leq V'$, iff $V[i] \leq V'[i]$, for all $i = 1, \dots, n$
- $V < V'$, iff $V \leq V' \ \& \ V \neq V'$
iff $V \leq V' \ \& \ \exists j$ such that $(V[j] < V'[j])$
- $e \rightarrow e'$ iff $V < V'$
 - $(V < V'$ implies $e \rightarrow e'$) and $(e \rightarrow e'$ implies $V < V')$
- $e \parallel e'$ iff $(V \not< V' \ \& \ V' \not< V)$

Vector Timestamps: Example



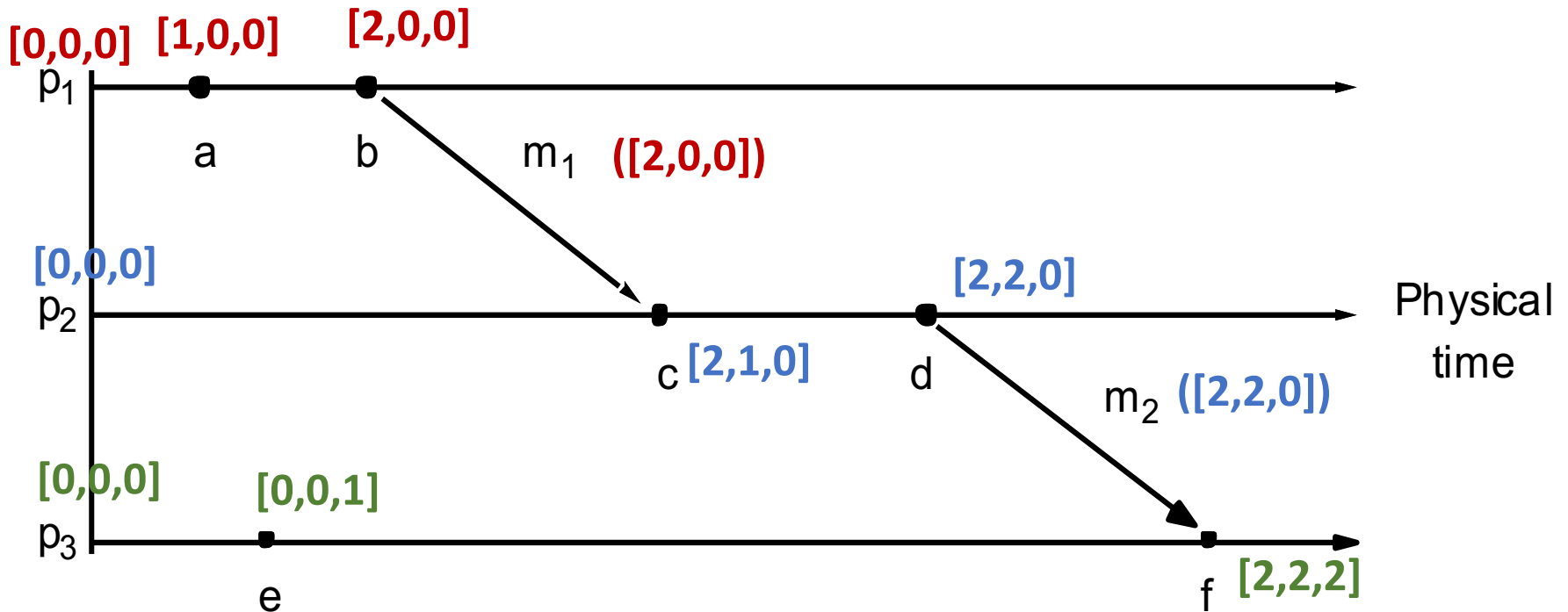
What can we say about e & f based on their vector timestamps?

Vector Timestamps: Example



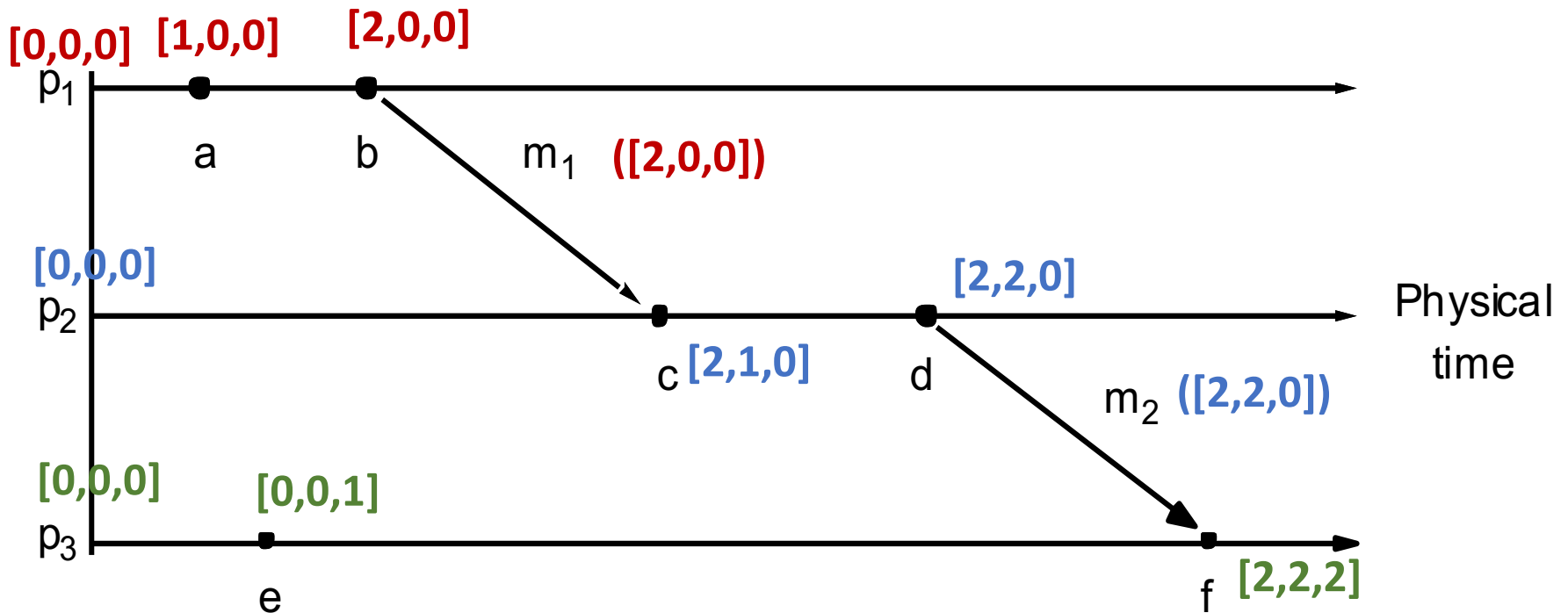
$$V(e) < V(f), e \rightarrow f$$

Vector Timestamps: Example



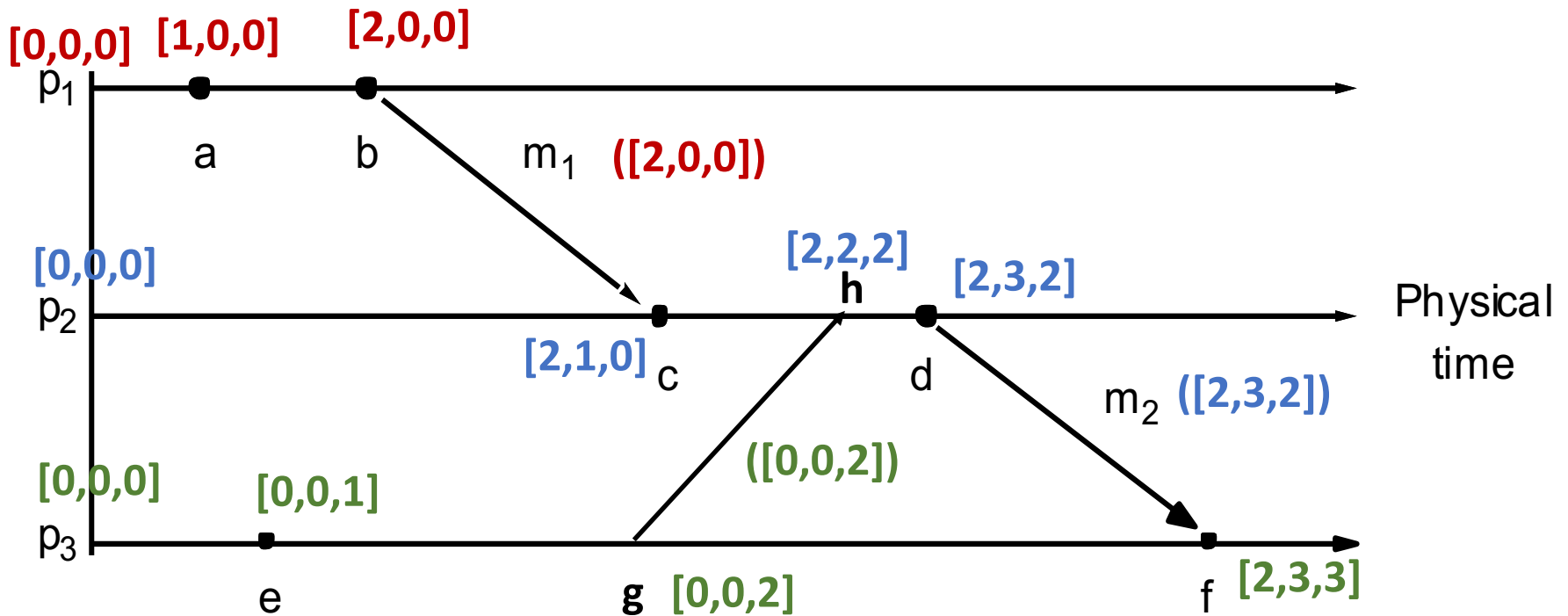
What can we say about e & d based on their vector timestamps?

Vector Timestamps: Example



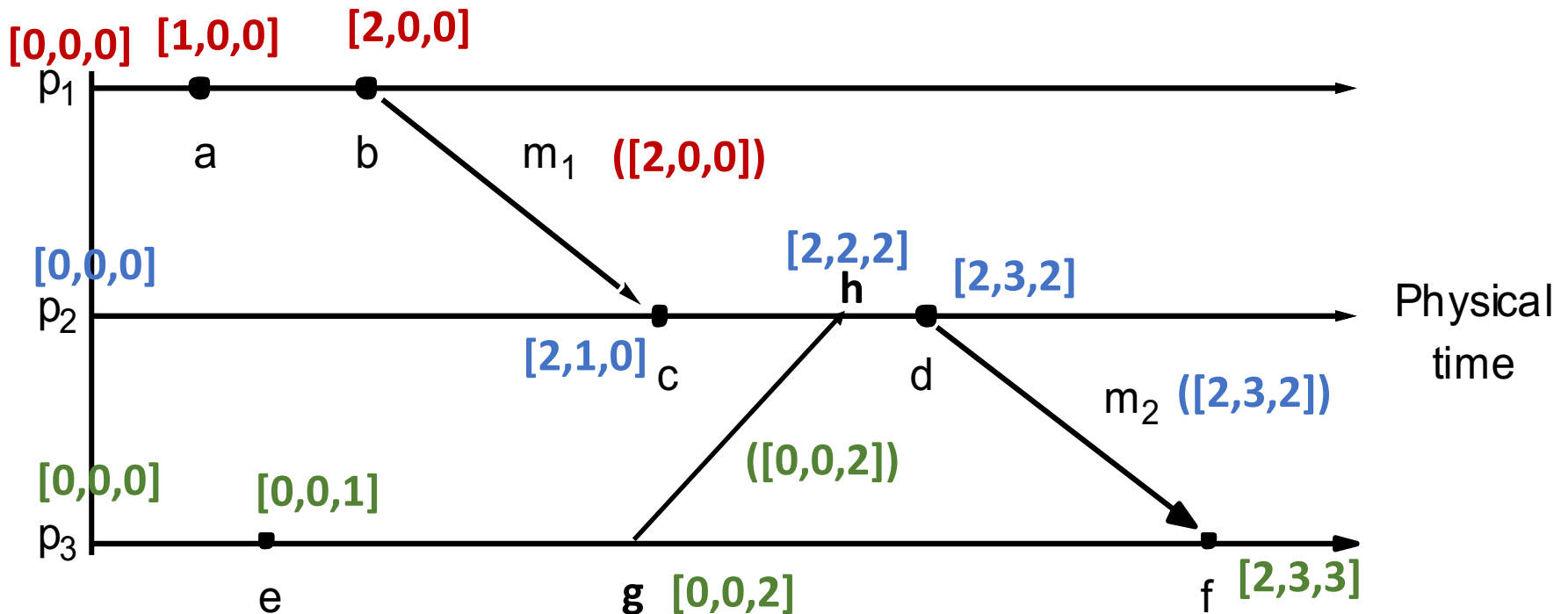
$V(e) \not\leq V(d)$ and $V(d) \not\leq V(e)$, $e \parallel d$

Vector Timestamps: Example



How about now?

Vector Timestamps: Example



$$V(e) < V(f), e \rightarrow f$$

$$V(e) < V(d), e \rightarrow d$$

Timestamps Summary

- **Comparing timestamps across events is useful.**
 - Reconciling updates made to an object in a distributed datastore.
 - Rollback recovery during failures:
 1. Checkpoint state of the system;
 2. Log events (with timestamps);
 3. Rollback to checkpoint and replay events in order if system crashes.
- **How to compare timestamps across different processes?**
 - **Physical timestamp:** requires clock synchronization.
 - Google's Spanner Distributed Database uses "TrueTime".
 - **Lamport's timestamps:** cannot fully differentiate between causal and concurrent ordering of events.
 - Oracle uses "System Change Numbers" based on Lamport's clock.
 - **Vector timestamps:** larger message sizes.
 - Amazon's DynamoDB uses vector clocks.