Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Logistics Related

- We have shared the VM mappings with Eng-IT.
  - We'll update you once the clusters have been assigned.

- My pace is way faster than last year!
  - Please feel free to ask questions.

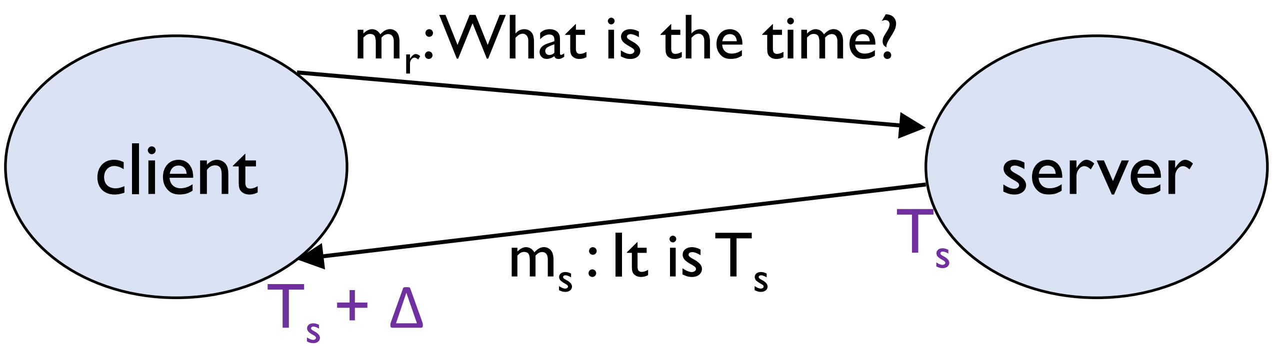
Today's agenda

- Time and Clocks
  - Chapter 14.1-14.3
- Logical Clocks and Timestamps
  - Chapter 14.4

Clock Skew and Drift Rates

- Each process has an internal clock.
- Clocks between processes on different computers differ:
  - Clock skew: relative difference between two clock values.
  - Clock drift rate: change in skew from a perfect reference clock per unit time (measured by the reference clock).
    - Depends on change in the frequency of oscillation of a crystal in the hardware clock.
- Synchronous systems have bound on maximum drift rate.

Synchronization in synchronous systems



What time T_c should client adjust its local clock to after receiving m_s ?

Let *max* and *min* be maximum and minimum network delay.

If $\mathbf{T_c = T_s}$, skew(client, server) $\leq$ *max*.

If $\mathbf{T_c = (T_s + max)}$, skew(client, server) $\leq$ *(max – min)*

If $\mathbf{T_c = (T_s + min)}$, skew(client, server) $\leq$ *(max – min)*

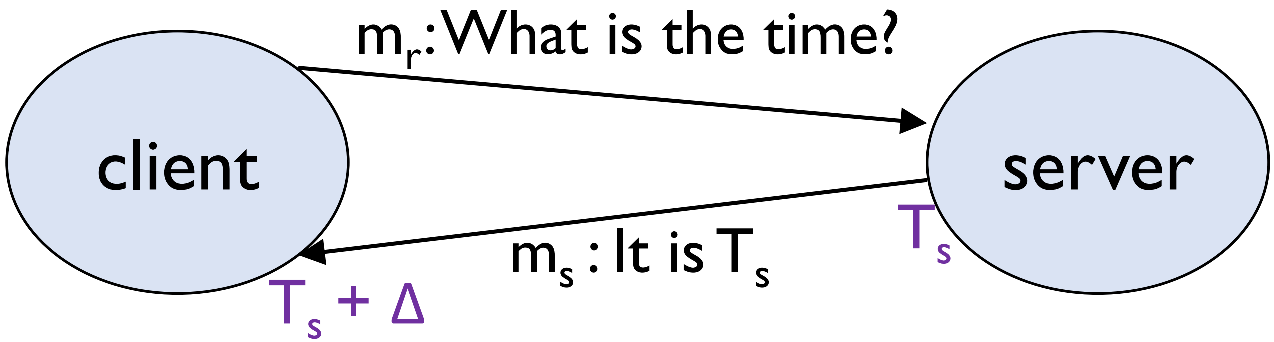
If $\mathbf{T_c = (T_s + (min + max)/2)}$, skew(client,server) $\leq$ *(max – min)/2*

Provably the best you can do!

Synchronization in asynchronous systems

- Cristian Algorithm
- Berkeley Algorithm
- Network Time Protocol

Cristian Algorithm



What time T_c should client adjust its local clock to after receiving m_s ?

Client measures the round trip time ($\mathbf{T_{round}}$).

$\mathbf{T_c = T_s + (T_{round} / 2)}$

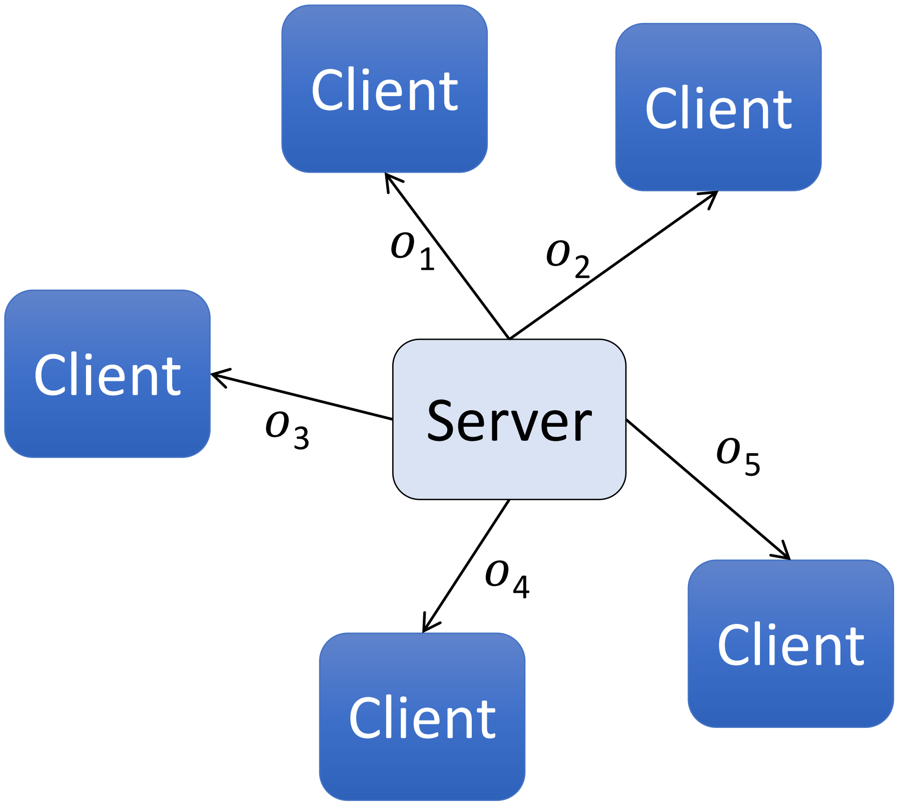
skew $\leq (T_{round} / 2) - \mathit{min}$
$\leq (T_{round} / 2)$
(*min* is minimum one way network delay which is atleast zero).

Improve accuracy by sending multiple spaced requests and using response with smallest T_{round}.

Server failure: Use multiple synchronized time servers.

Berkeley Algorithm

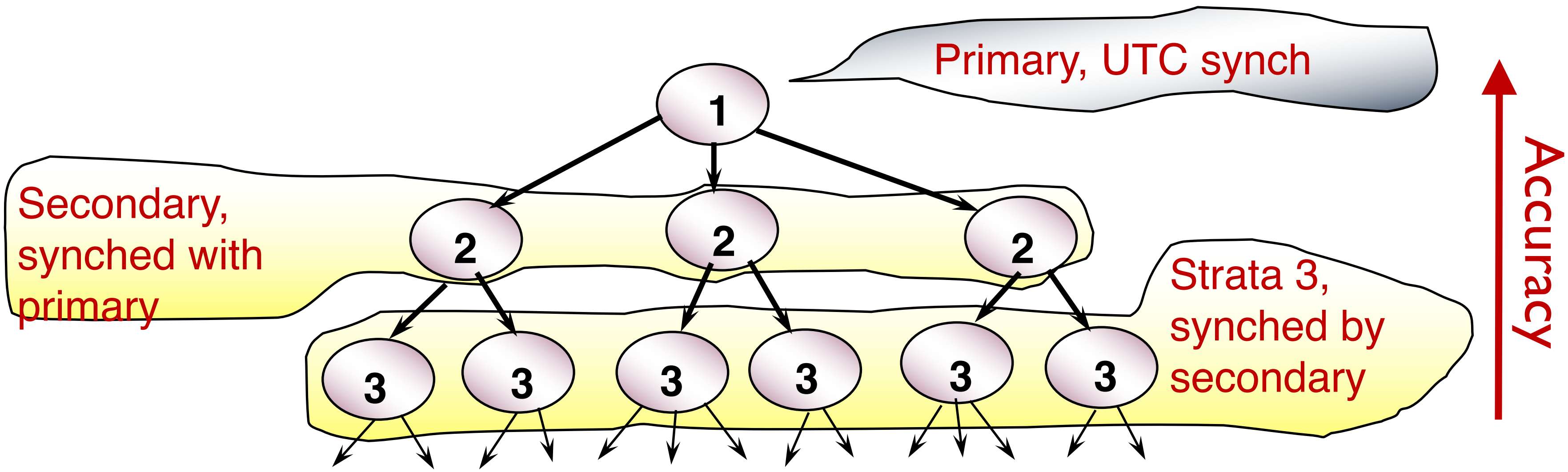
Only supports internal synchronization.



1. Server periodically polls clients: *"what time do you think it is?"*
2. Each client responds with its local time.
3. Server uses Cristian algorithm to estimate local time at each client.
4. Average all local times (including its own) – use as updated time.
5. Send the offset (amount by which each clock needs adjustment).

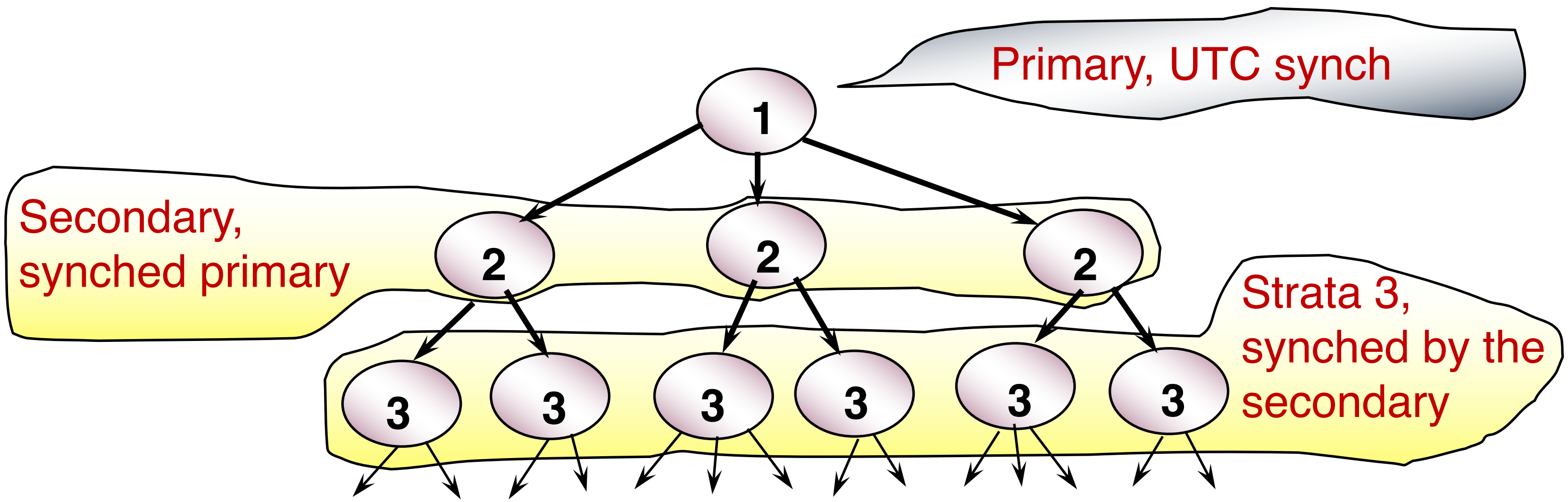
Network Time Protocol

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*.

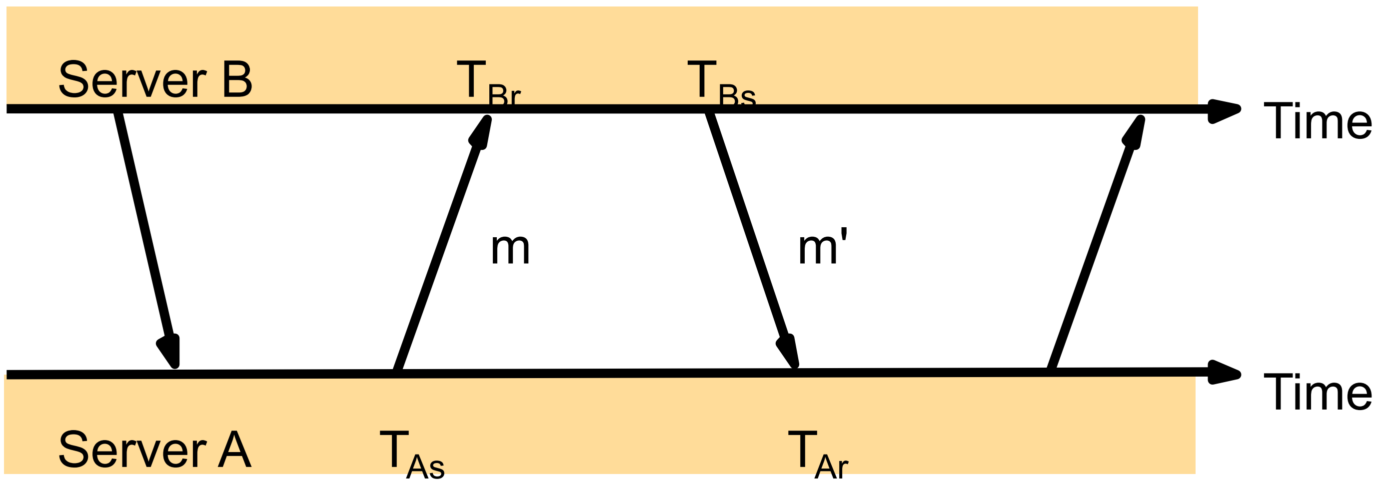
Network Time Protocol



How clocks get synchronized:

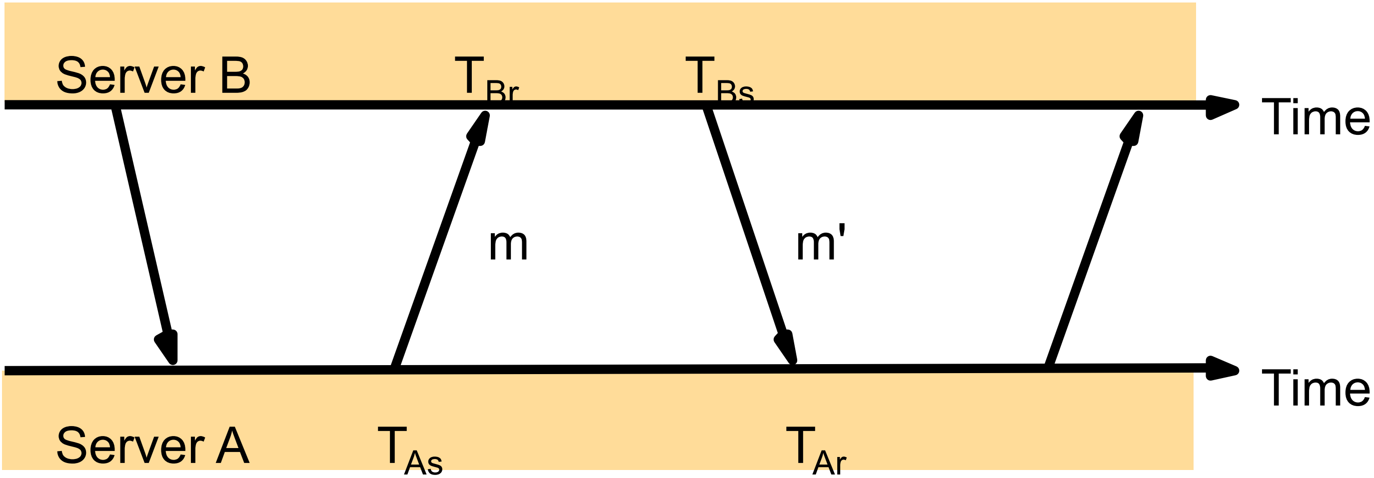
- Servers may *multicast* timestamps within a LAN. Clients adjust time assuming a small delay. *Low accuracy*.
- *Procedure-call* (Cristian algorithm). *Higher accuracy.*
- *Symmetric mode* used to synchronize lower strata servers. *Highest accuracy*.

NTP Symmetric Mode



- A and B exchange messages and record the send and receive timestamps.
  - T_{Br} and T_{Bs} are local timestamps at B.
  - T_{Ar} and T_{As} are local timestamps at A.
  - A and B exchange their local timestamp with eachother.
- Use these timestamps to compute offset with respect to one another.

NTP Symmetric Mode



- t and t': actual transmission times for m and m'(unknown)
- o: <u>true</u> offset of clock at B relative to clock at A (unknown)
- o_i: <u>estimate</u> of actual offset between the two clocks

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

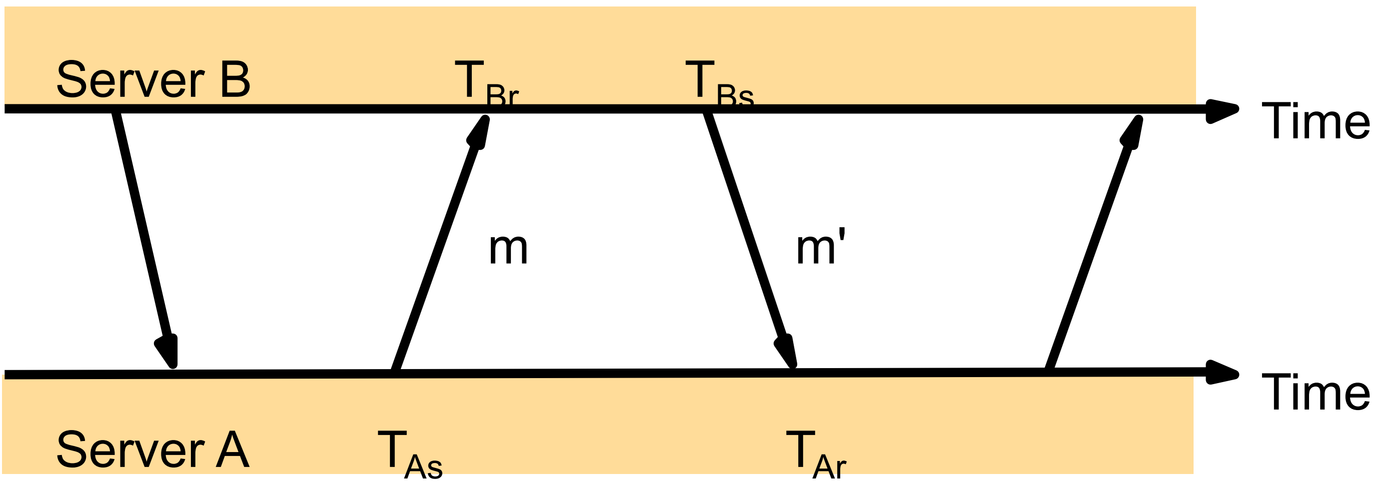
$$\mathbf{T_{Ar} = T_{Bs}} + t' - o$$

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

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

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

NTP Symmetric Mode

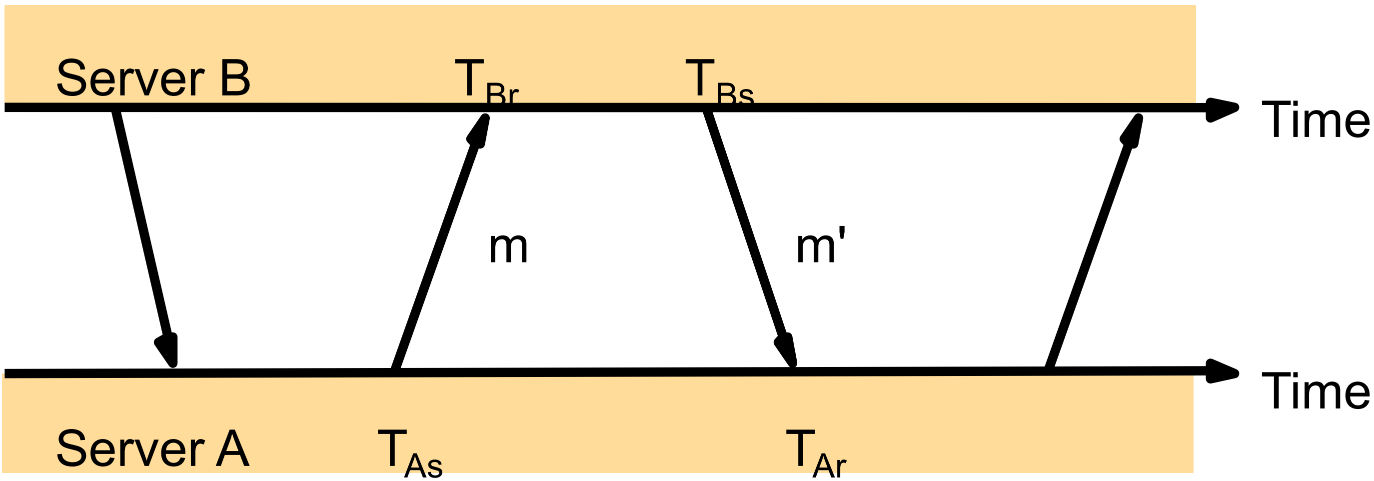


- t and t': actual transmission times for m and m'(unknown)
- o: true offset of clock at B relative to clock at A (unknown)
- o_i: estimate of actual offset between the two clocks
-

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

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

NTP Symmetric Mode



- t and t': actual transmission times for m and m'(unknown)
- o: <u>true</u> offset of clock at B relative to clock at A (unknown)
- o_i: <u>estimate</u> of actual offset between the two clocks
- d_i: estimate of <u>accuracy</u> of o_i ; $d_i = t + t'$
- $d_i/2$: synchronization bound

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

How off can o_i be?

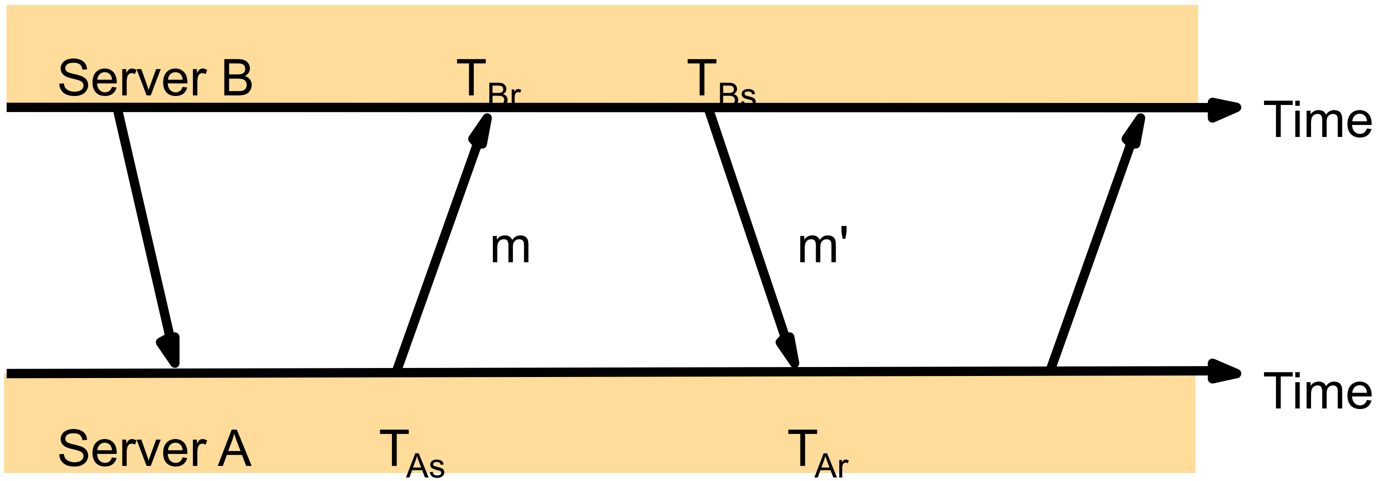
- We do not know, t, t' or (t'-t)
- We do not know max or min delays.
- We know (t + t'), $t \geq 0$, $t' \geq 0$

$$\mathbf{d_i = t + t'}$$

- (t'-t) ~= (t + t'), if t ~= 0 (one extreme)
- (t'-t) ~= - (t + t'), if t' ~= 0 (other extreme)

$$(o_i - d_i / 2) \leq o \leq (o_i + d_i / 2)$$

NTP Symmetric Mode



- t and t': actual transmission times for m and m'(unknown)
- o: <u>true</u> offset of clock at B relative to clock at A (unknown)
- o_i: <u>estimate</u> of actual offset between the two clocks
- d_i: estimate of <u>accuracy</u> of o_i ; $d_i = t + t'$
- $d_i/2$: synchronization bound

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

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

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

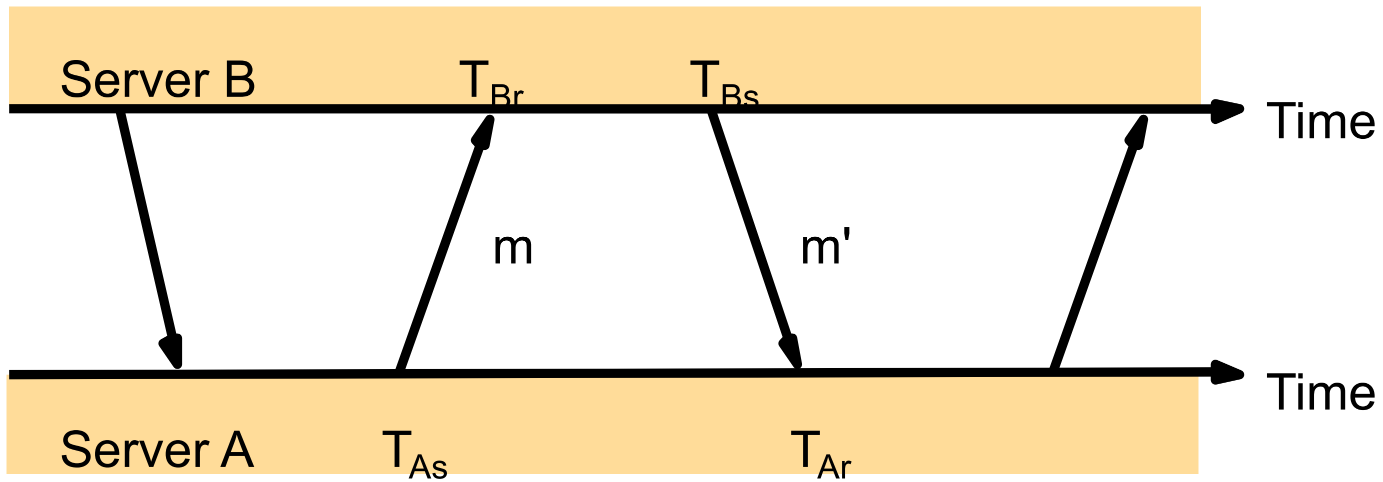
$$\mathbf{o_i = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}))/2}$$

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

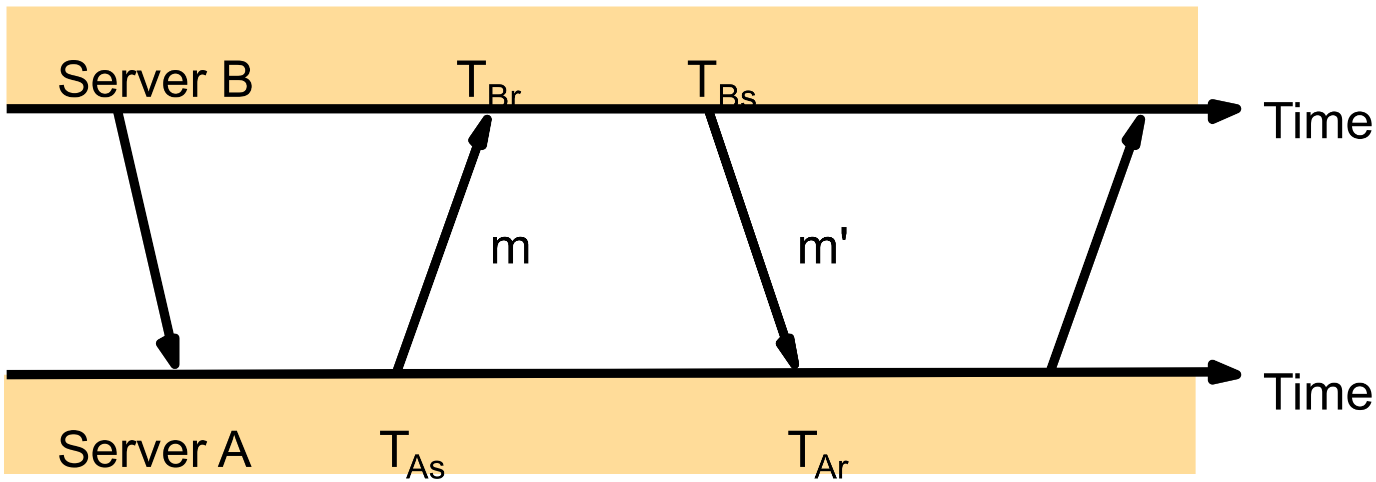
$$\mathbf{d_i = t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs})}$$

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

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.

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
  - Hierarchical time synchronization over the Internet.

Today's agenda

- Time and Clocks
  - Chapter 14.1-14.3
- Logical Clocks and Timestamps
  - Chapter 14.4

Event Ordering

- A usecase of synchronized clocks:
  - Reasoning about order of events.
- Why is it useful?
  - Debugging distributed applications
  - 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.*
  - ….
- Can we reason about order of events without synchronized clocks?

Process, state, events

- Consider a system with **n** processes: $<p_1, p_2, p_3, \ldots., p_n>$
- Each process $\mathbf{p_i}$ is described by its *state* $\mathbf{s_i}$ that gets transformed over time.
  - State includes values of all local variables, affected files, etc.
- $\mathbf{s_i}$ 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 $\mathbf{p_i}$, 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 $\rightarrow$.
  - $\mathbf{e} \rightarrow \mathbf{e'}$ means **e** *happened before* **e'**.

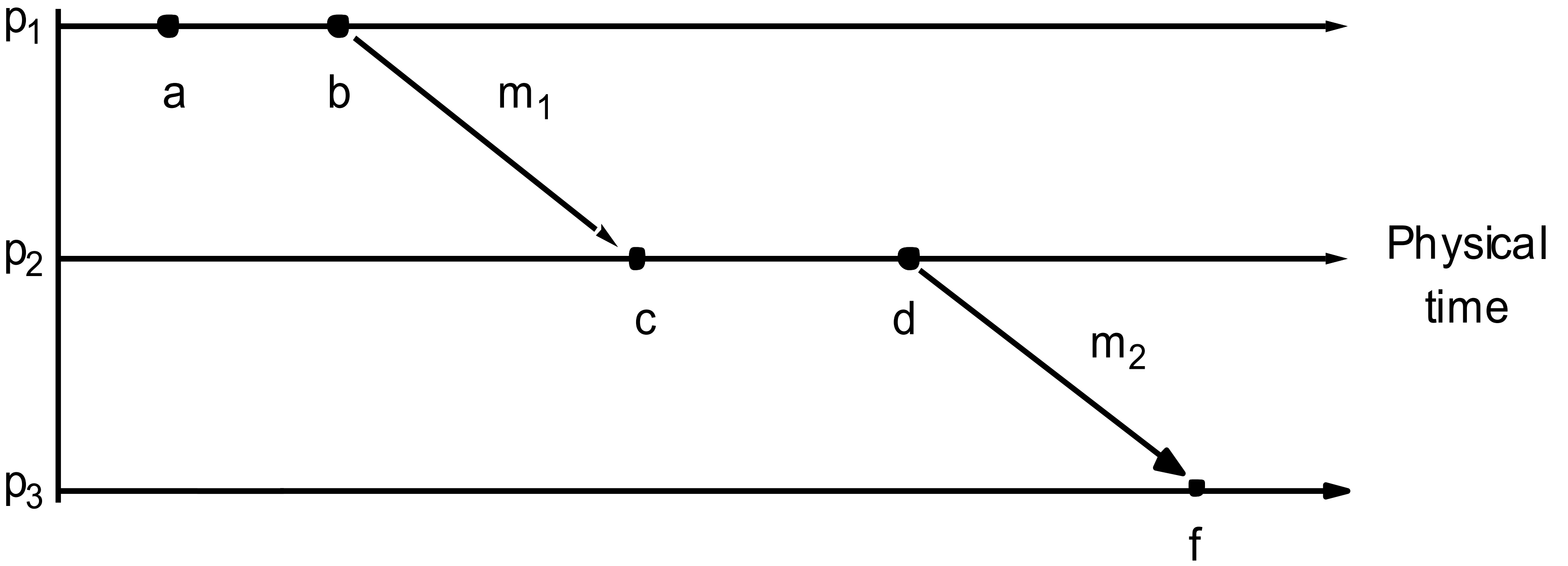
Happened-Before Relationship

- *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 $\mathbf{p_i}$.

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

- Also called *"causal" or "potentially causal"* ordering.

Event Ordering: Example

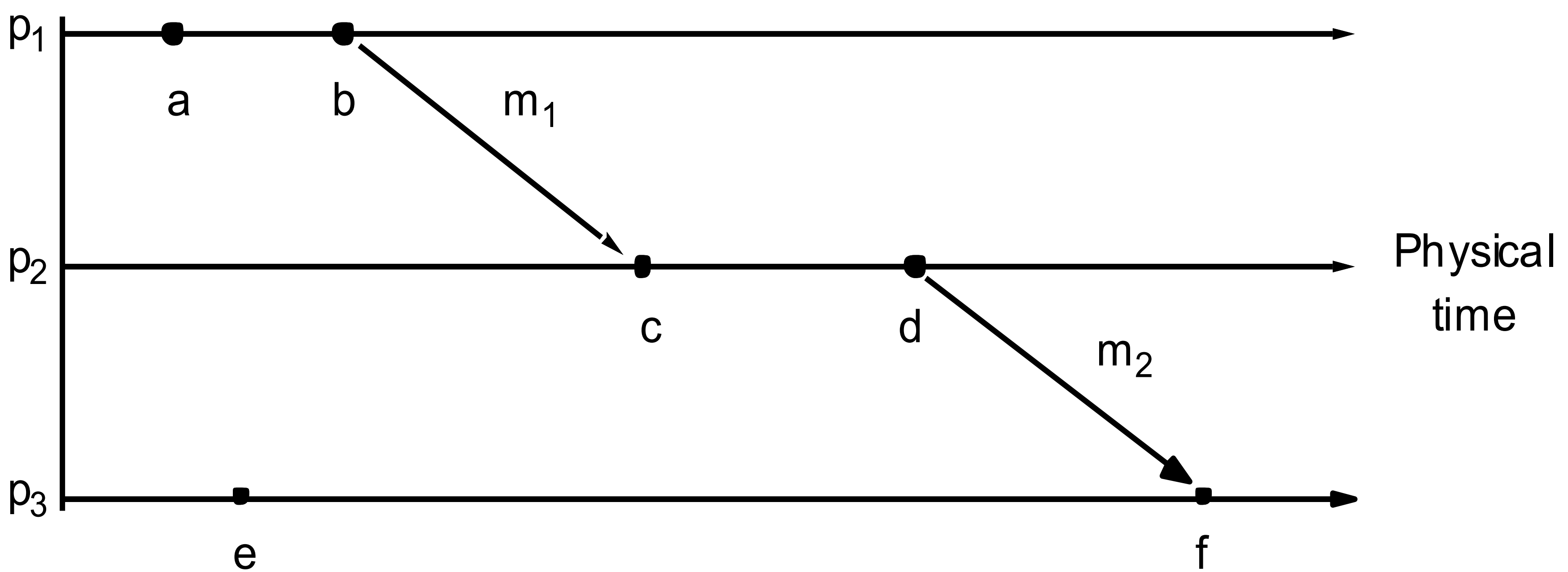


Which event happened first?

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**?

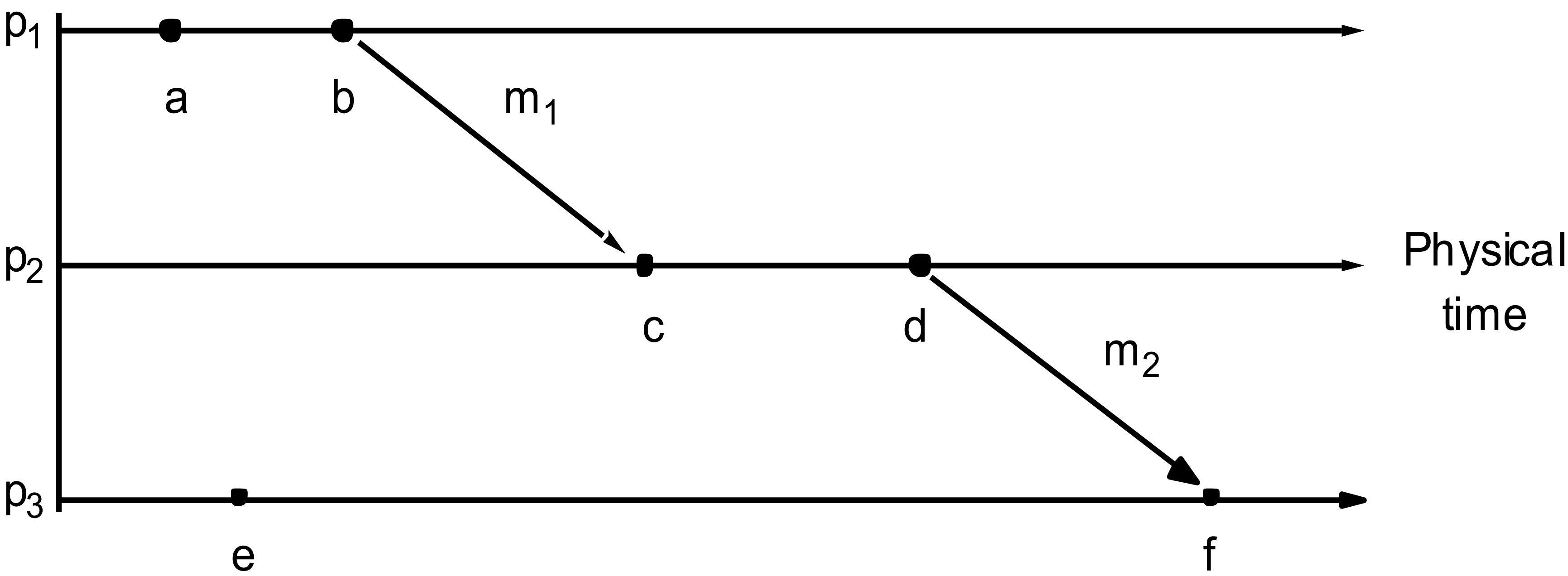
e $\rightarrow$ **f**

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

a || **e**

a and **e** are *concurrent*.

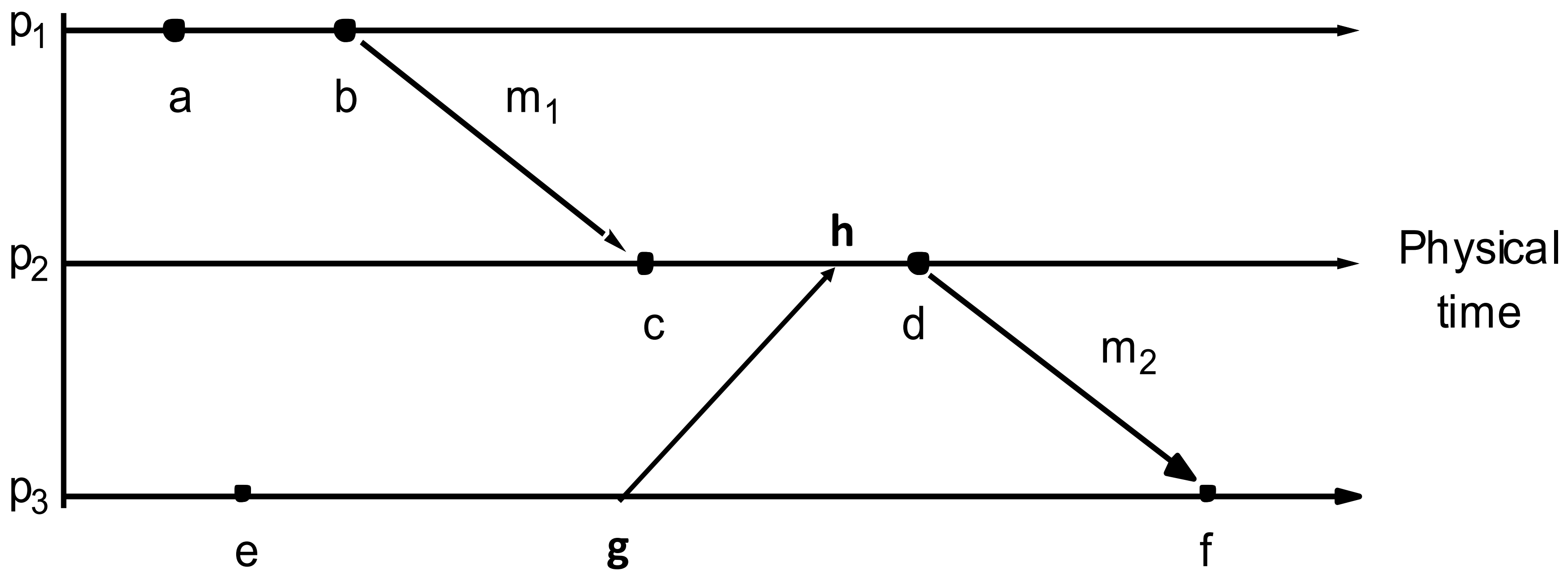
Event Ordering: Example



What can we say about **e** and **d**?

e || d

Event Ordering: Example



What can we say about **e** and **d**?

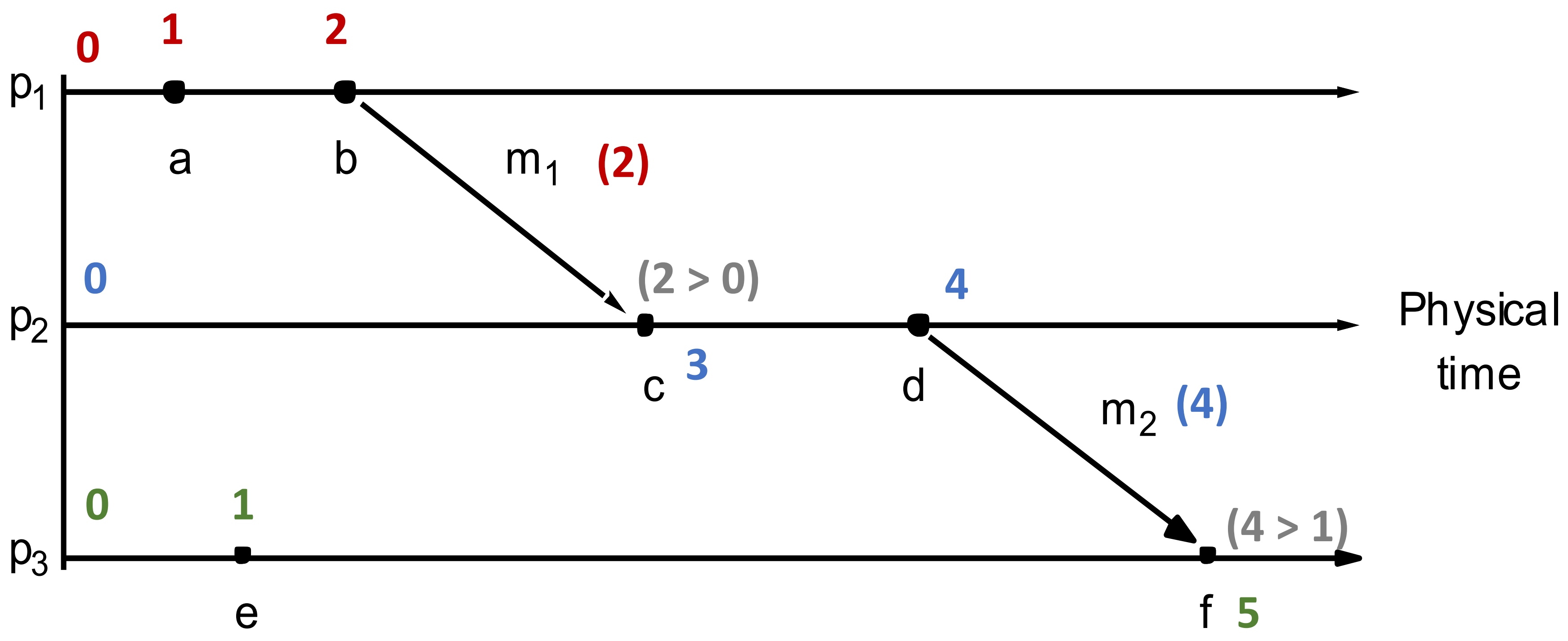
e → d

Lamport's Logical Clock

- Logical timestamp for each event that captures the *happened-before* relationship.

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

Logical Timestamps: Example

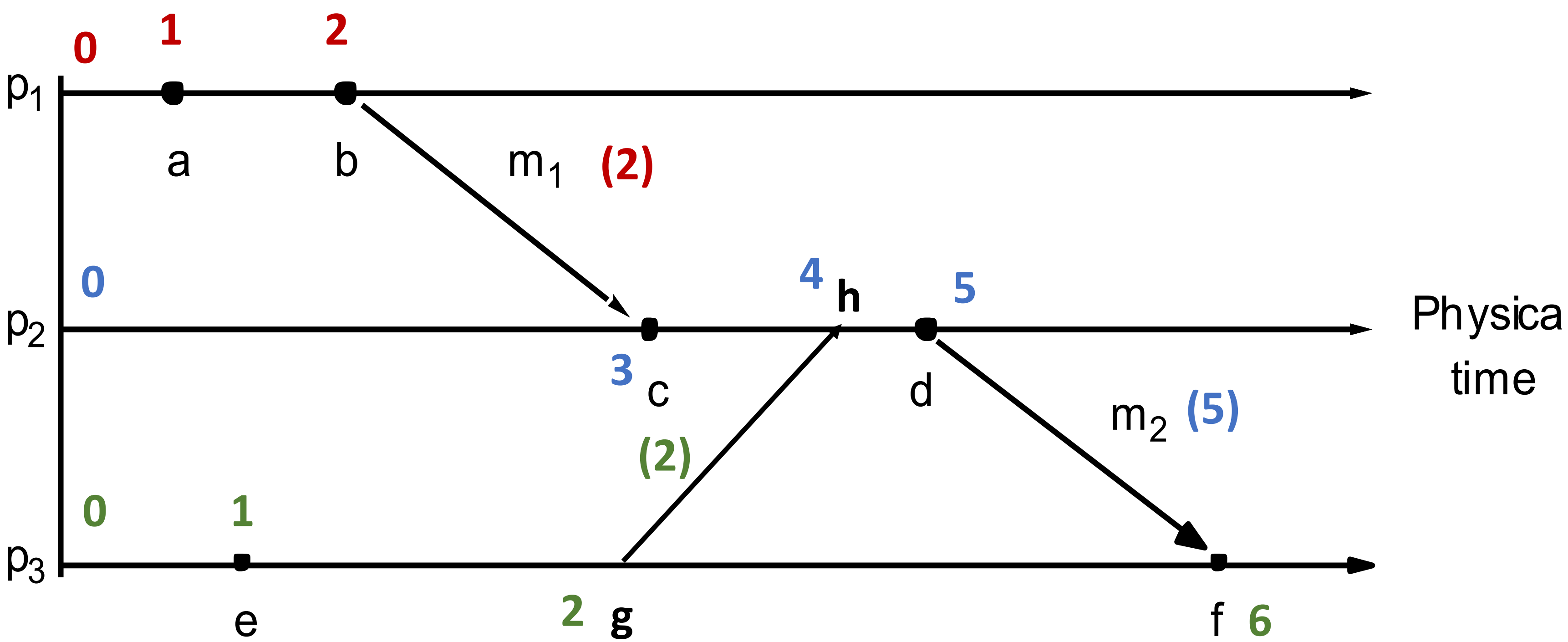


Lamport's Logical Clock

- Logical timestamp for each event that captures the *happened-before* relationship.

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

Logical Timestamps: Example



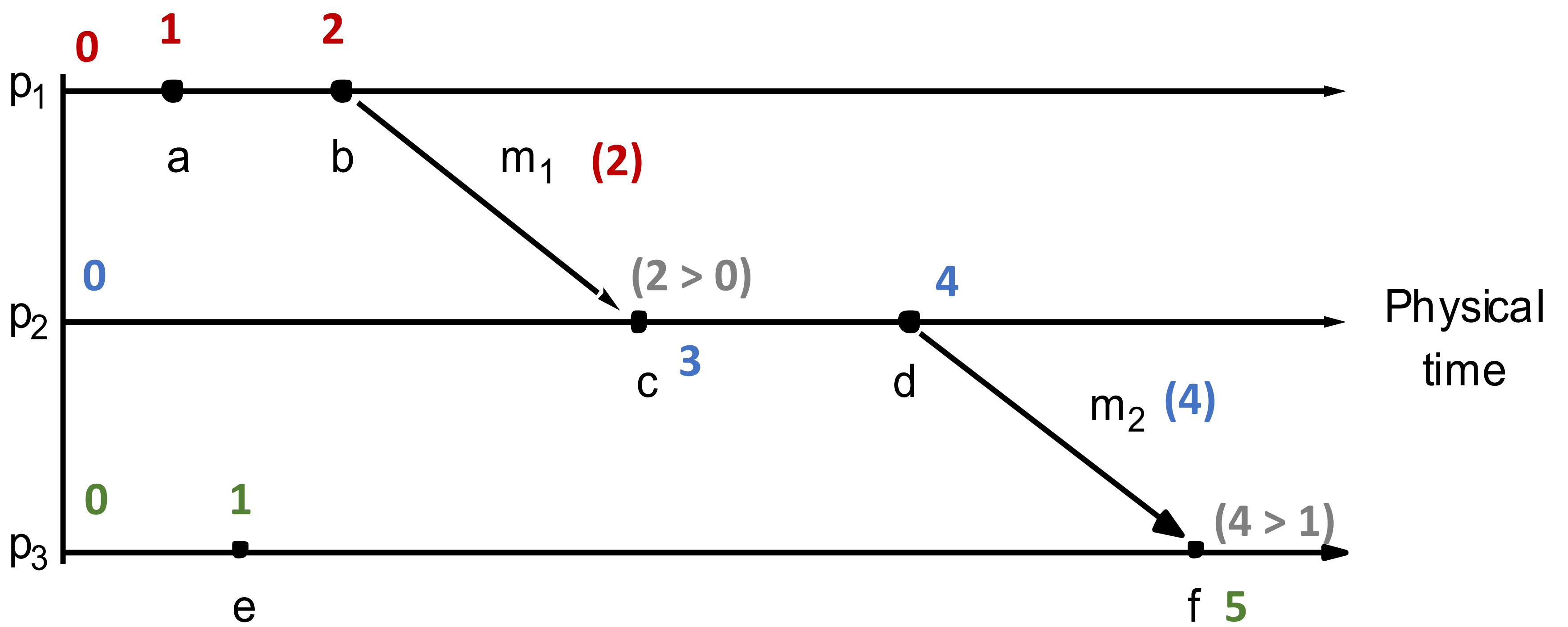
Lamport's Logical Clock

- 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** || **e'**

Logical Timestamps: Example



L(e) < L(d), e || d

L(e) < L(f), e → f

Vector Clocks

- Next class….