Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Today's agenda

- Failure Detection (wrap up)
  - Chapter 15.1
- Time and Clocks
  - Chapter 14.1-14.3
- Logical Clocks and Timestamps (if time)
  - Chapter 14.4

Why are clocks useful?

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.

Ordinary and Authoritative Clocks

- Ordinary quartz crystal clocks:
  - Drift rate is about 10^{-6} seconds/second.
  - Drift by 1 second every 11.6 days.
  - Skew of about 30minutes after 60 years.

- High precision atomic clocks:
  - Drift rate is about 10^{-13} seconds/second.
  - Skew of about 0.18ms after 60 years.
  - Used as standard for real time.
  - Universal Coordinated Time (UTC) obtained from such clocks.

Two forms of synchronization

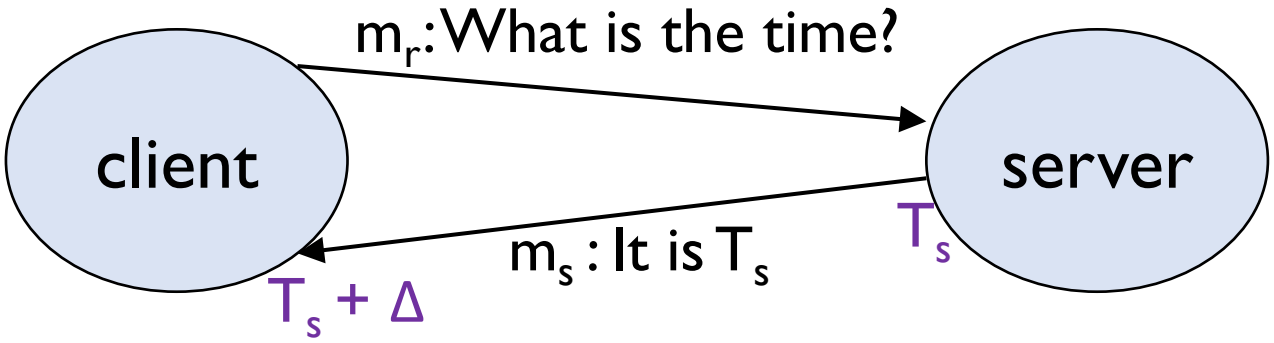
- External synchronization
  - Synchronize time with an authoritative clock.
  - When accurate timestamps are required.
- Internal synchronization
  - Synchronize time internally between all processes in a distributed system.
  - When internally comparable timestamps are required.
- If all clocks in a system are externally synchronized, they are also internally synchronized.

Synchronization Bound

- Synchronization bound (D) between two clocks A and B over a real time interval I.
  - $|A(t) - B(t)| < D$, for all t in the real time interval I.
    - Skew(A, B) < D during the time interval I.
    - A and B *agree* within a bound *D*.
  - If A is authoritative, D can also be called *accuracy bound*.
    - B is *accurate* within a bound of *D*.
- Synchronization/accuracy bound (D) at time 't'
  - worst-case skew between two clocks at time 't'
    - Skew(A, B) < D at time t

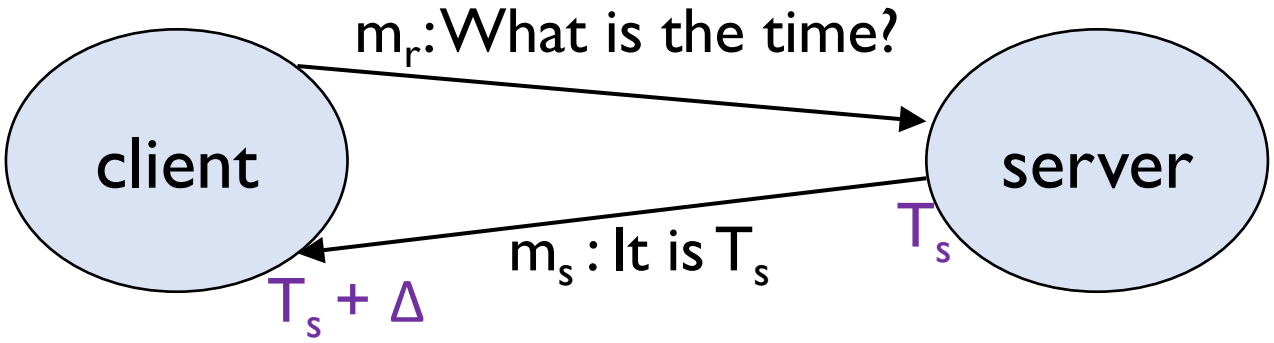
Q: *If all clocks in a system are externally synchronized within a bound of D, what is the bound on their skew relative to one another?*

Synchronization in synchronous systems



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

Synchronization in synchronous systems



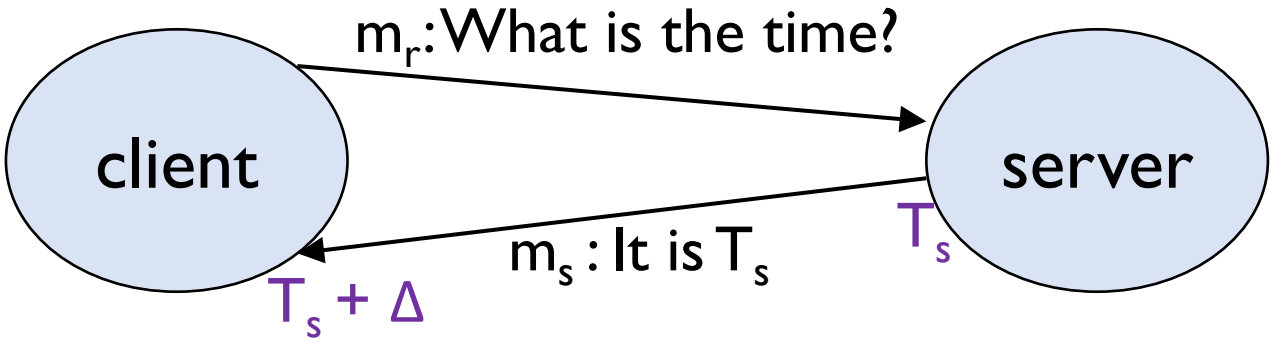
What time T_c should client adjust its local clock to after receiving m_s ?

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

Synchronization in asynchronous systems

- Cristian Algorithm
- Berkeley Algorithm
- Network Time Protocol

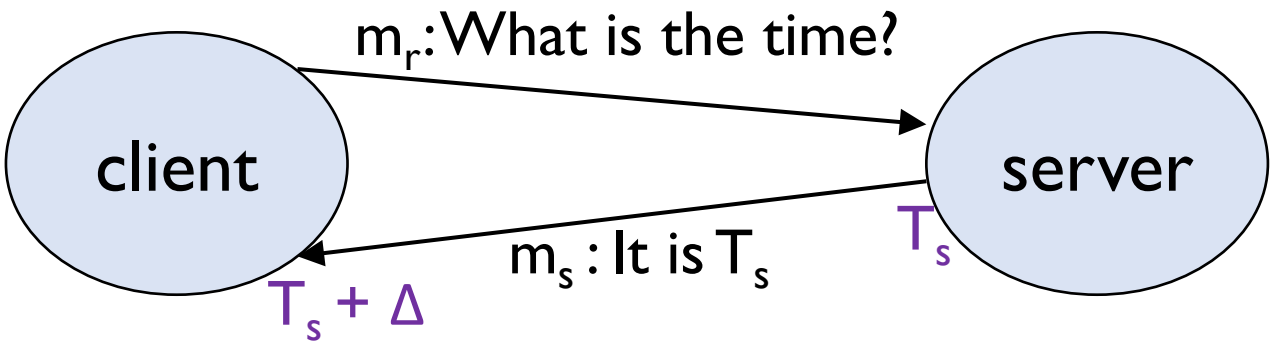
Cristian Algorithm



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

Client measures the round trip time (T_{round})
= time difference between when client sends m_r and receives m_s.

Cristian Algorithm



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

Client measures the round trip time (T_{round})

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

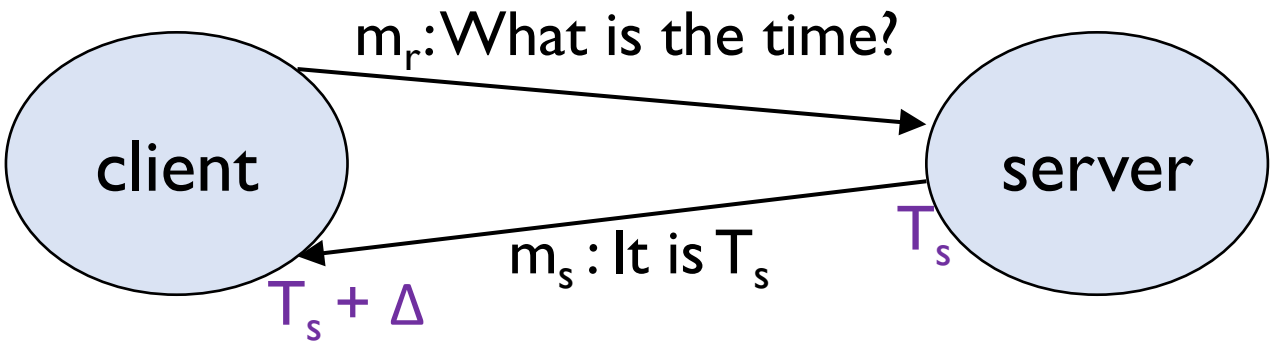
skew $\leq (T_{round} / 2) - min$
$\leq (T_{round} / 2)$

(*min* is minimum one way network delay which is atleast zero).

Try deriving the worst case skew!

Hint: *client is assuming its one-way delay from server is* $\Delta = (T_{round}/2)$. *How off can it be?*

Cristian Algorithm



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

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

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

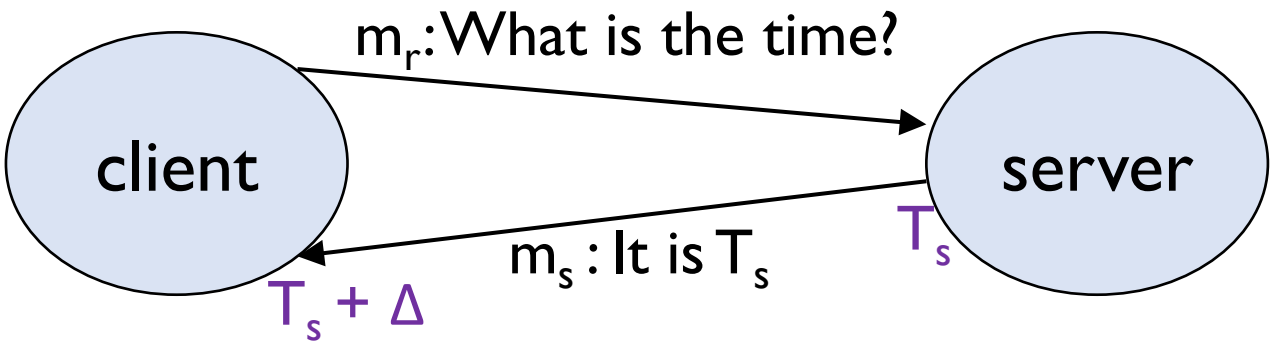
skew $\leq (T_{round} / 2) - min$
$\leq (T_{round} / 2)$

(*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.

Cristian Algorithm



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

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

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

skew $\leq (T_{round} / 2) - min$

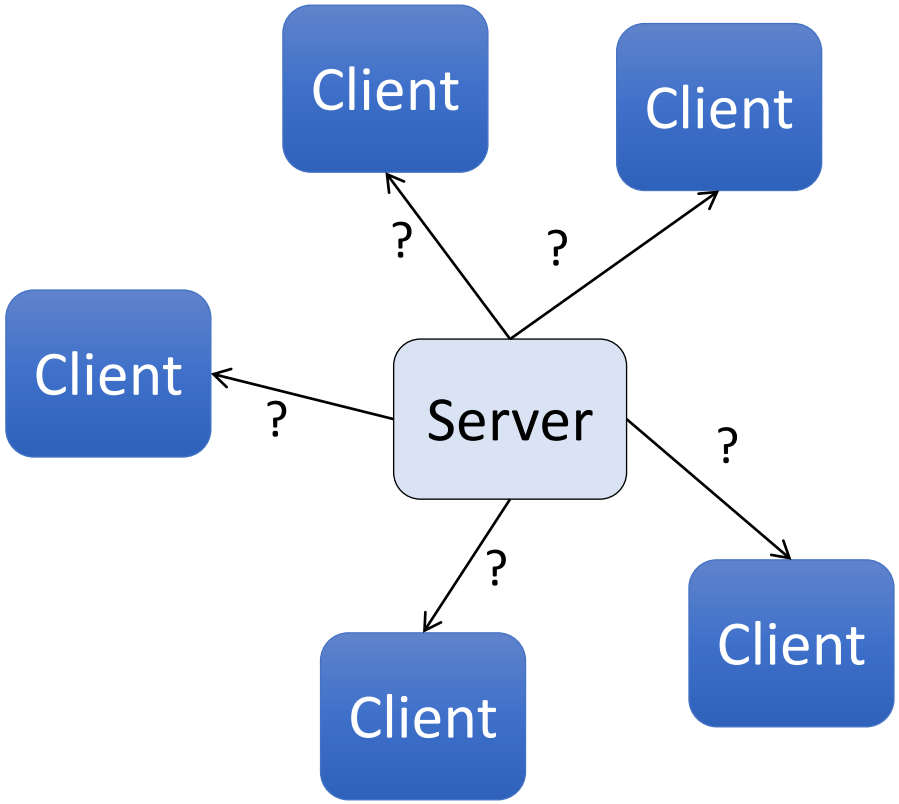
$\leq (T_{round} / 2)$

(*min* is minimum one way network delay which is atleast zero).

Cannot handle faulty time servers.

Berkeley Algorithm

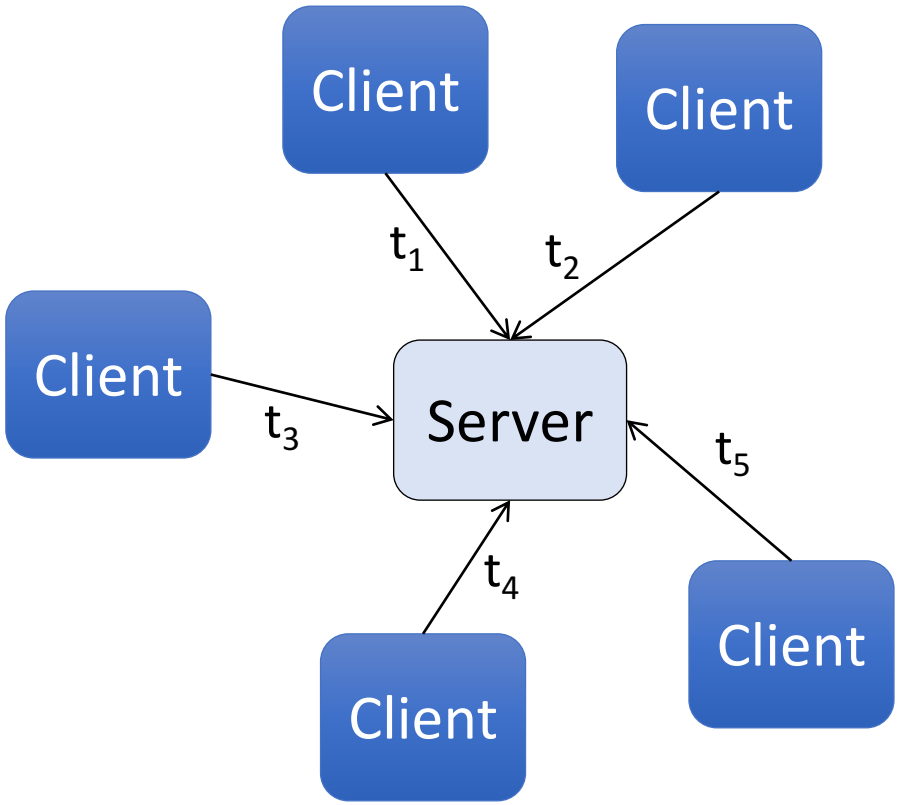
Only supports internal synchronization.



1. Server periodically polls clients:
   "what time do you think it is?"

Berkeley Algorithm

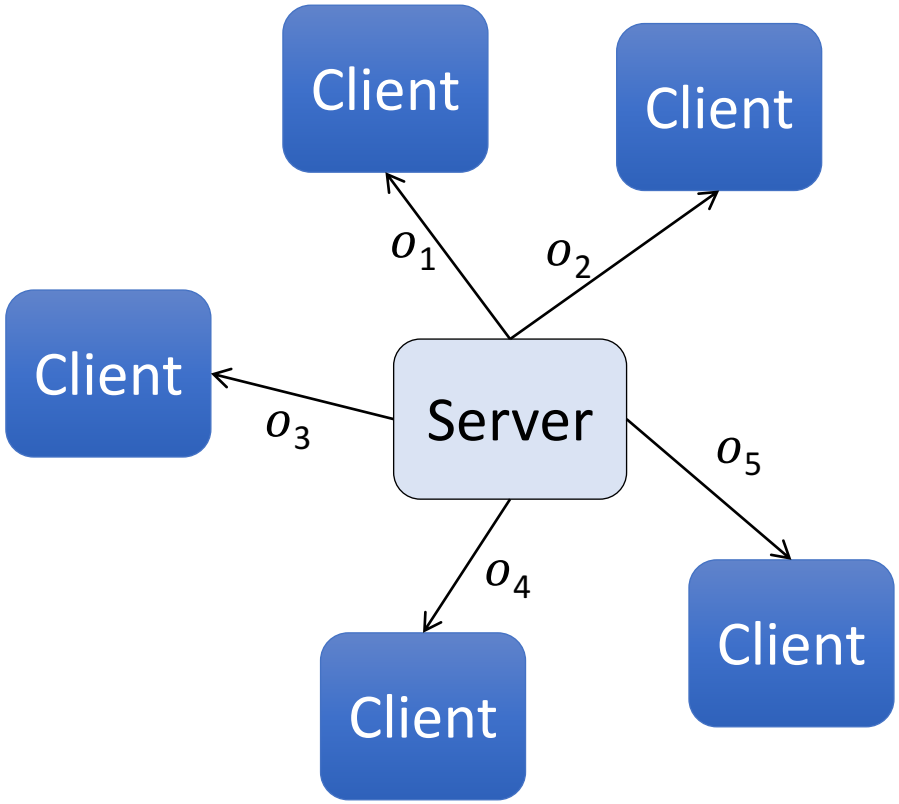
Only supports internal synchronization.



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.

Berkeley Algorithm

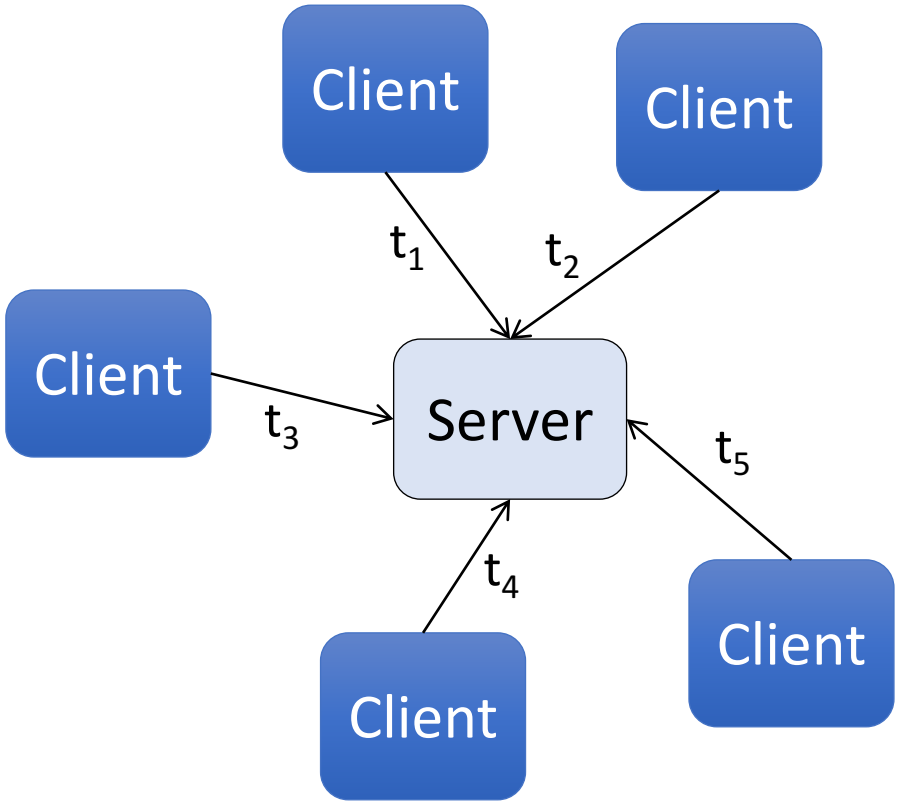
Only supports internal synchronization.



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).

Berkeley Algorithm

Only supports internal synchronization.

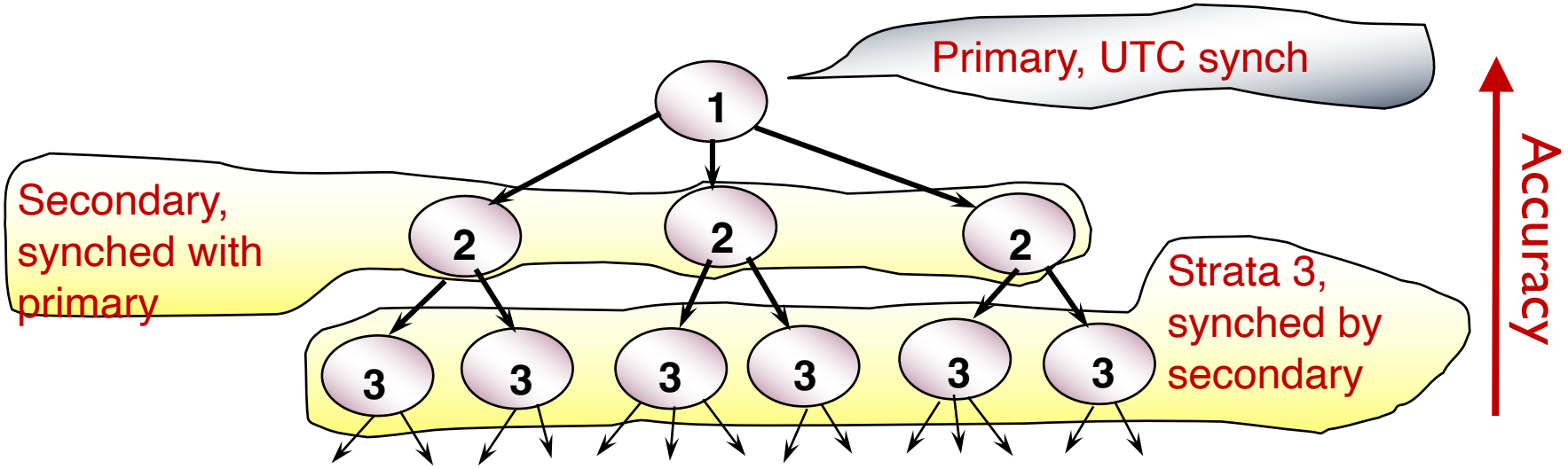


Handling faulty processes:
Only use timestamps within some threshold of each other.

Handling server failure:
Detect the failure and elect a new leader.

Network Time Protocol

Time service over the Internet for synchronizing to UTC.



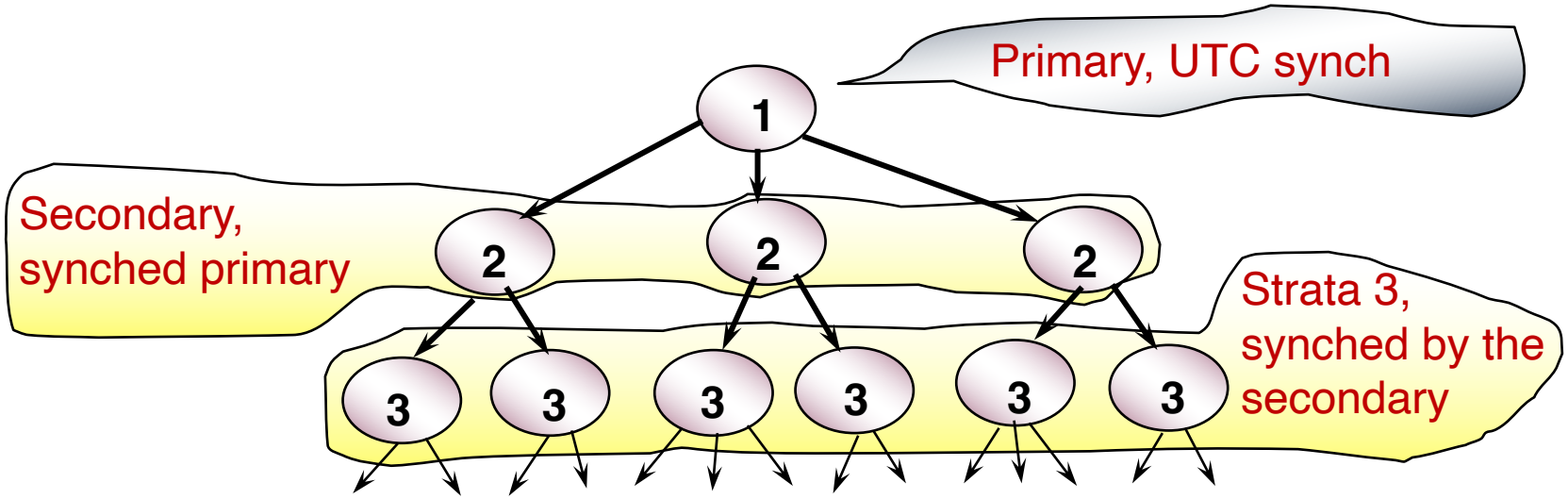
Hierarchical structure for *scalability*.

Multiple lower strata servers for *robustness*.

Authentication mechanisms for *security*.

Statistical techniques for better *accuracy*.

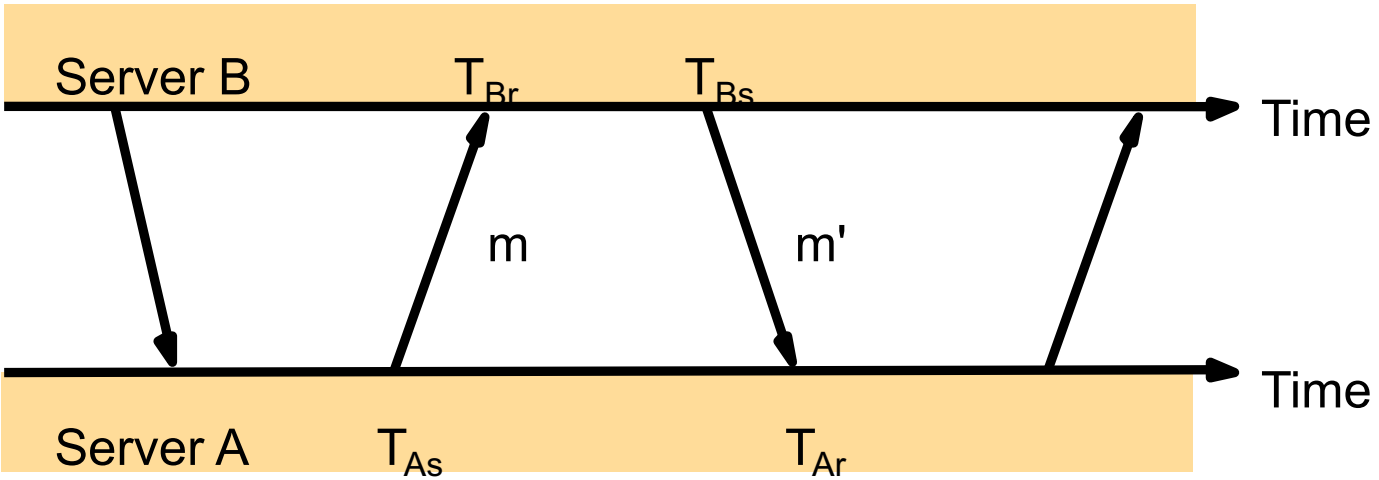
Network Time Protocol



How clocks get synchronized:

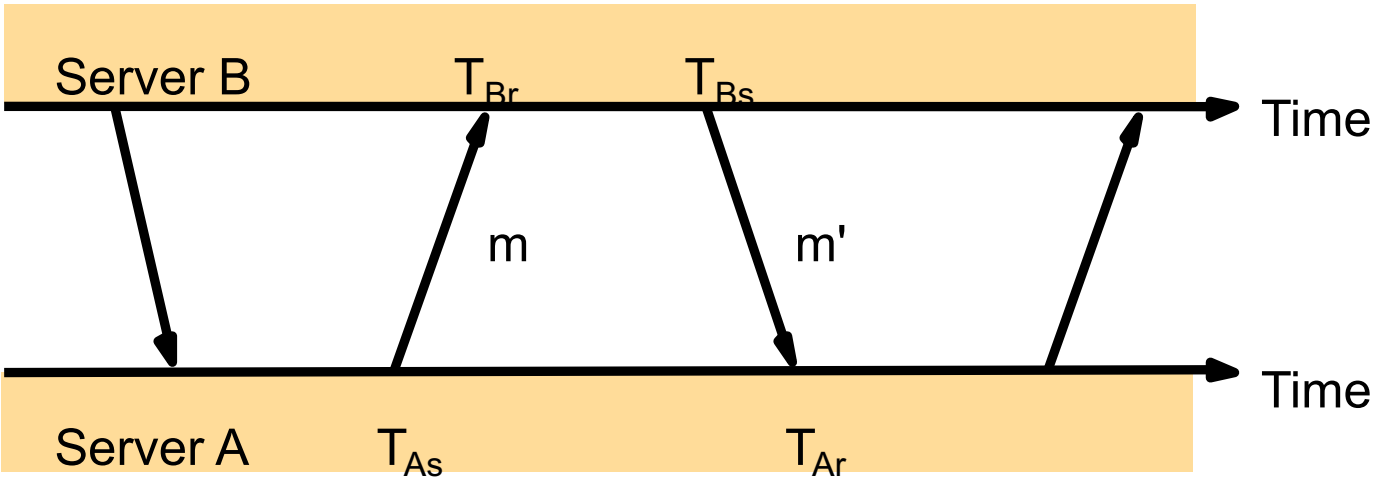
- Servers may *multicast* timestamps within a LAN. Clients adjust time assuming a small delay. *Low accuracy*.
- *Procedure-call* (Cristian algorithm). *Higher accuracy.*
- *Symmetric mode* used to synchronize lower strata servers. *Highest accuracy*.

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: true offset of clock at B relative to clock at A (unknown)
- o_i: estimate of actual offset between the two clocks

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

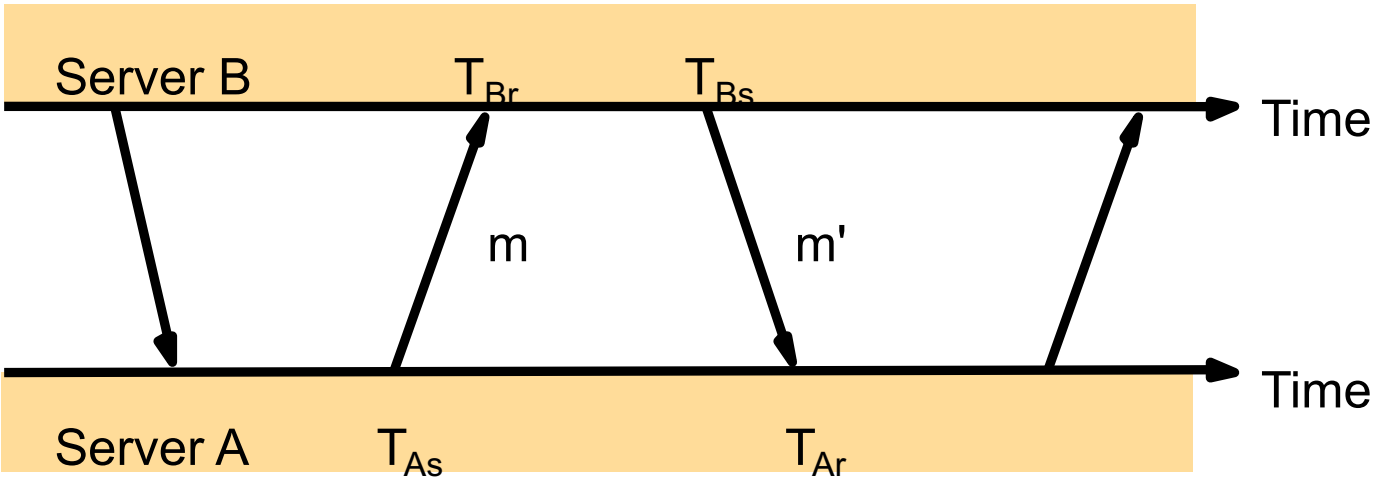
$$T_{Ar} = T_{Bs} + t' - o$$

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

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

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

NTP Symmetric Mode



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

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

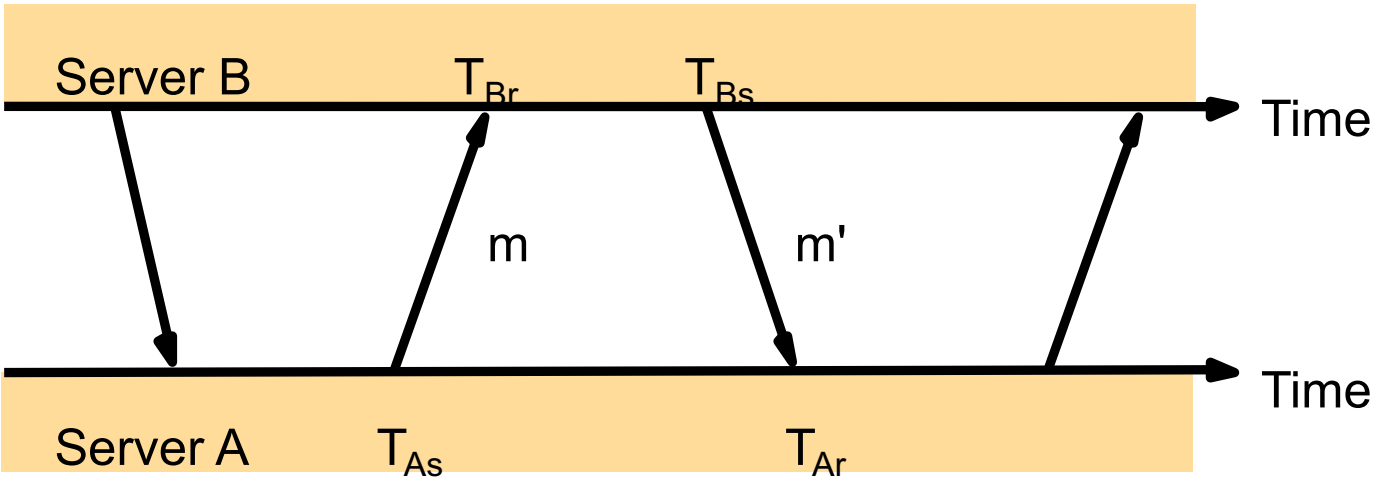
$$T_{Ar} = T_{Bs} + t' - o$$

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

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

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

NTP Symmetric Mode



- t and t': actual transmission times for m and m'(unknown)
- o: 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 ; d_i=t+t'
- $d_i/2$: synchronization bound

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

How off can o_i be?

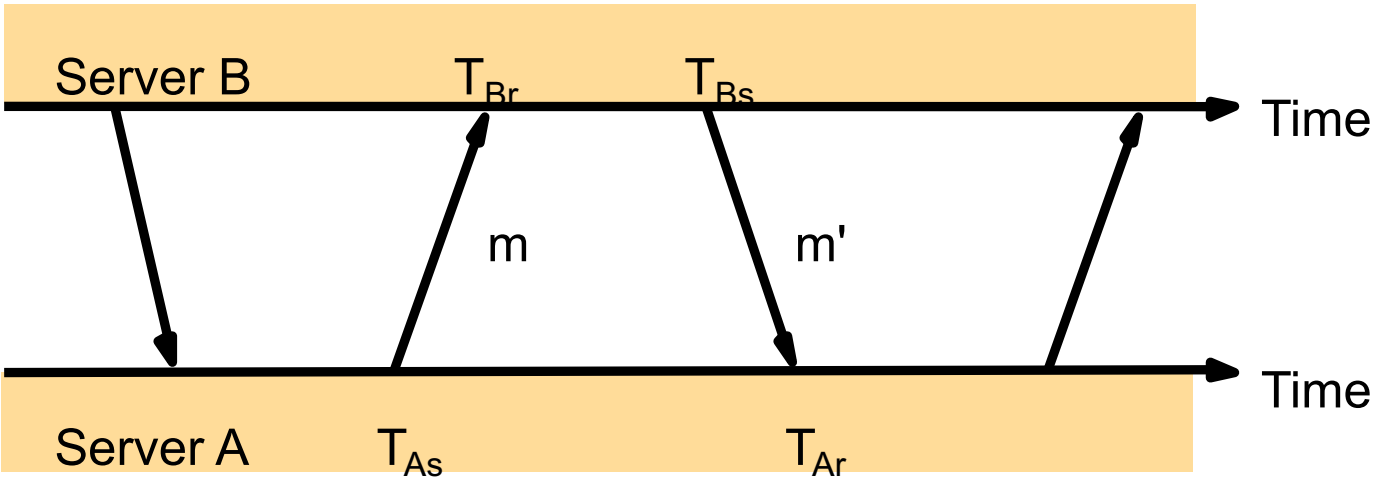
- We do not know, t, t' or (t'-t)
- We do not know max or min delays.
- We know (t + t'), $t \geq 0$, $t' \geq 0$

$d_i = t + t'$

- (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

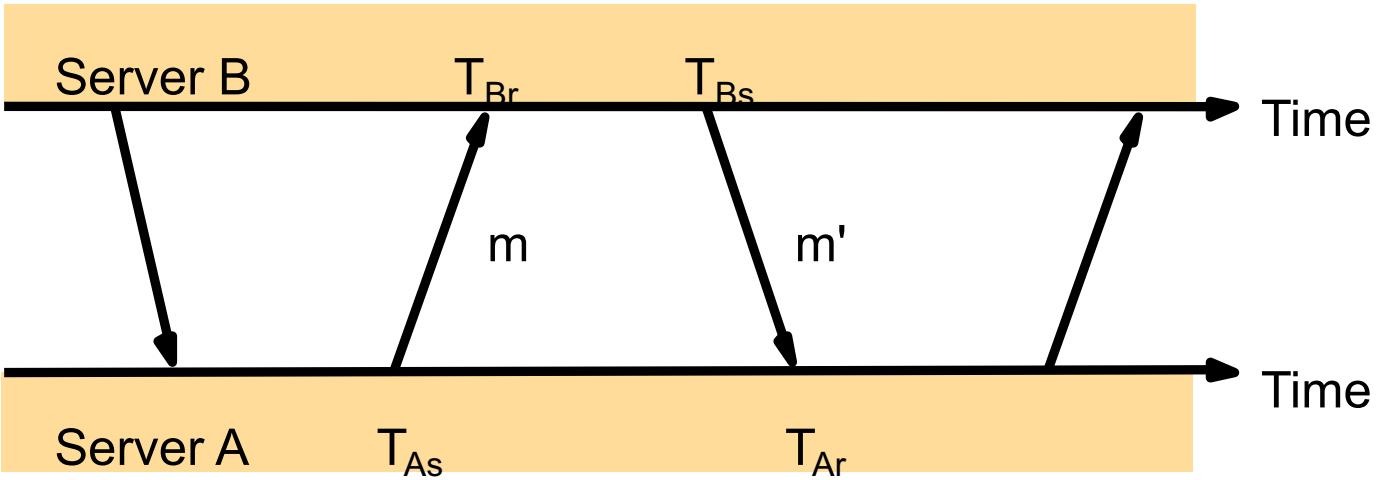


- t and t': actual transmission times for m and m' (unknown)
- o: true offset of clock at B relative to clock at A (unknown)
- o_i: estimate of actual offset between the two clocks
- d_i: estimate of accuracy of o_i ; $d_i = t + t'$
- $d_i/2$: synchronization bound

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

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

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 (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.

Next on today's agenda

- Logical Clocks and Timestamps
  - Chapter 14.1-14.3

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: $\langle p_1, p_2, p_3, \ldots, p_n \rangle$
- 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 $\mathbf{p_i}$, 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 →.
  - **e** → **e'** means **e** *happened before* **e'**.

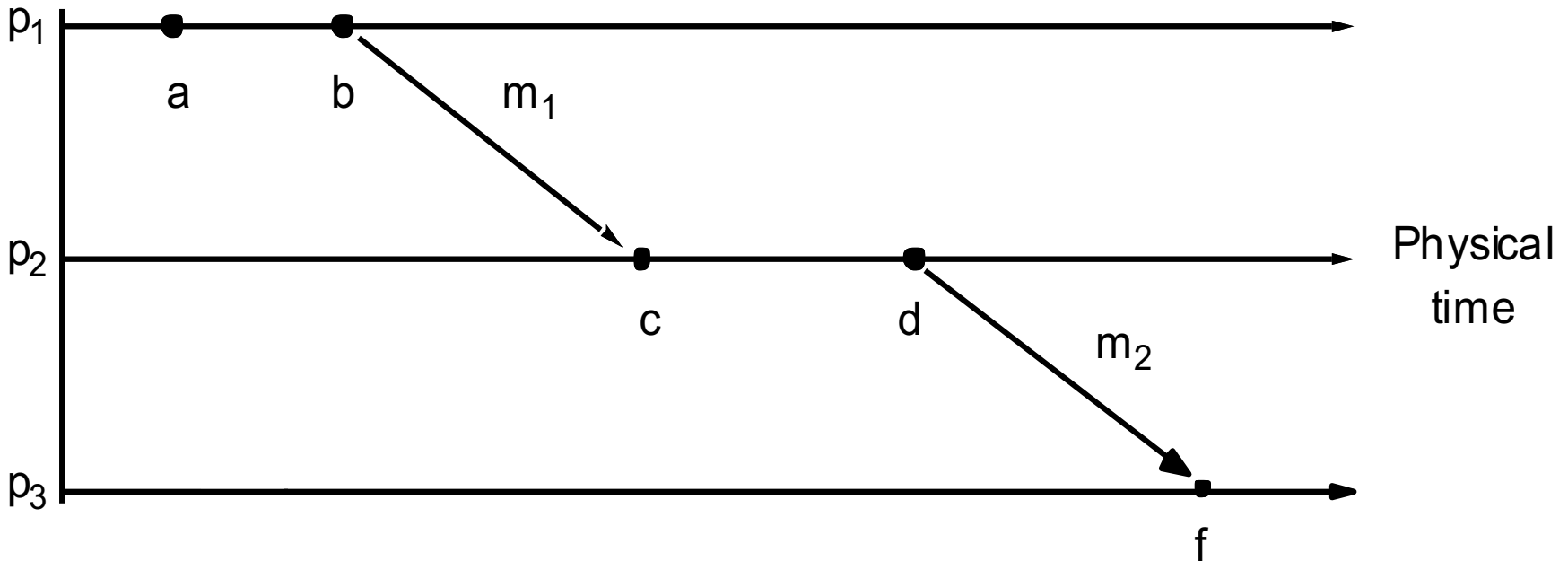
Happened-Before Relationship

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

- HB rules:
  - If ∃ p_i , **e** $\rightarrow_i$ **e'** then **e** → **e'**.
  - For any message m, **send(m)** → **receive(m)**
  - If **e** → **e'** and **e'** → **e"** then **e** → **e"**

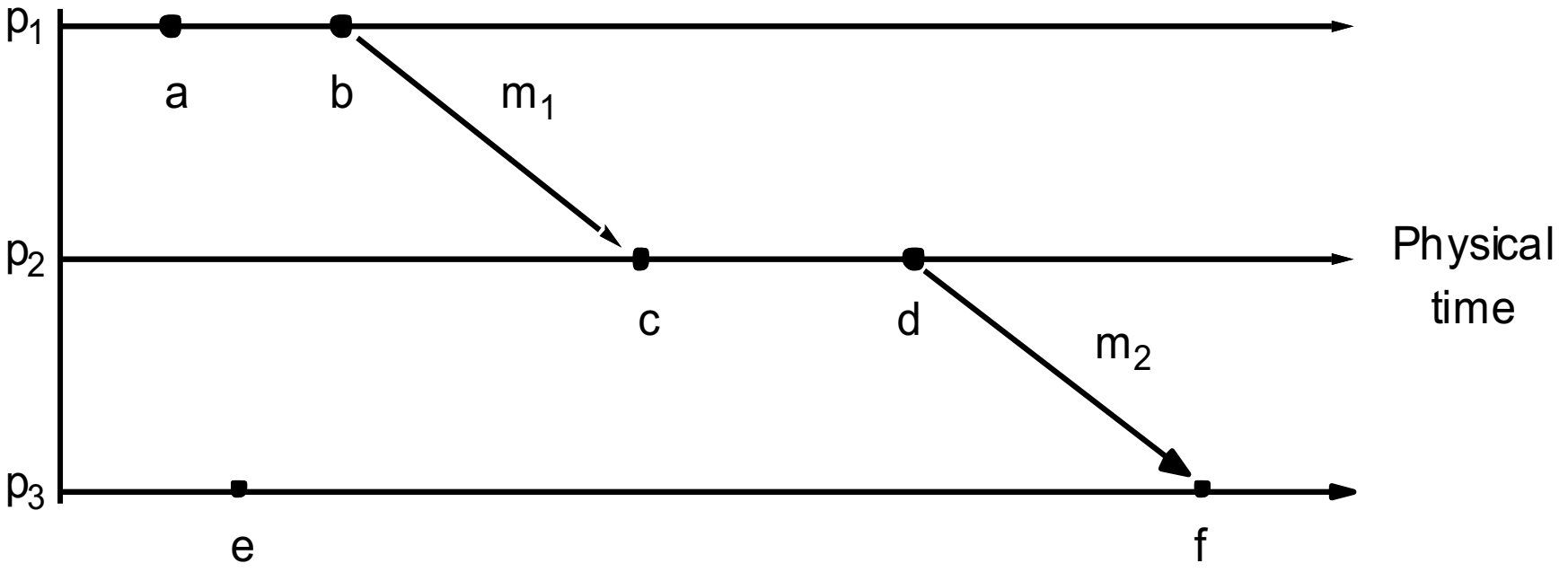
- Also called *"causal" or "potentially causal"* ordering.

Event Ordering: Example



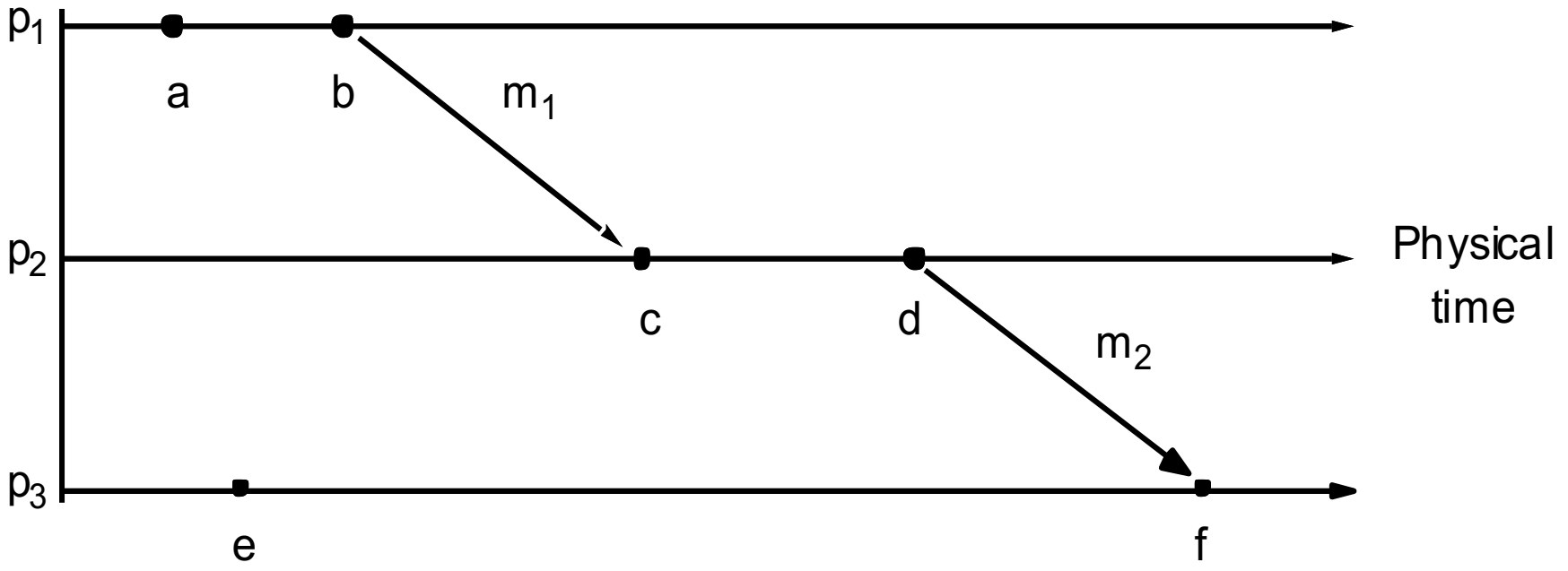
Which event happened first?

Event Ordering: Example



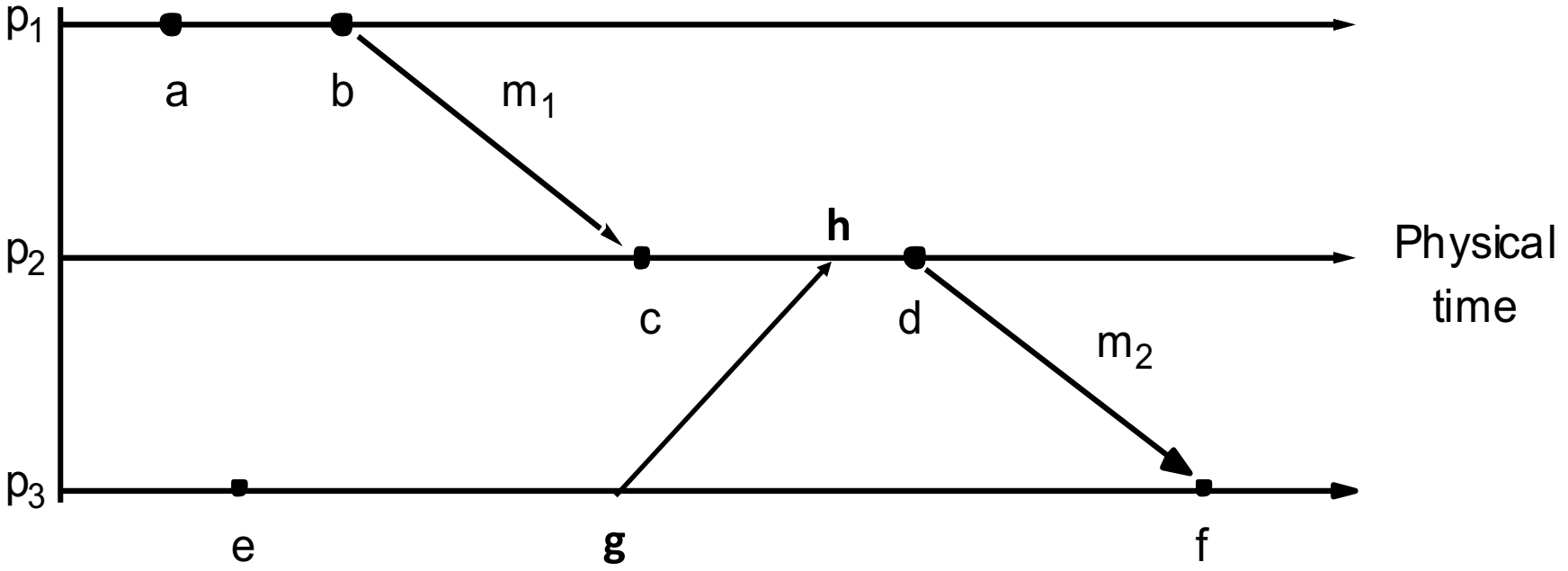
What can we say about **e**?

Event Ordering: Example



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

Logical Timestamps: Example



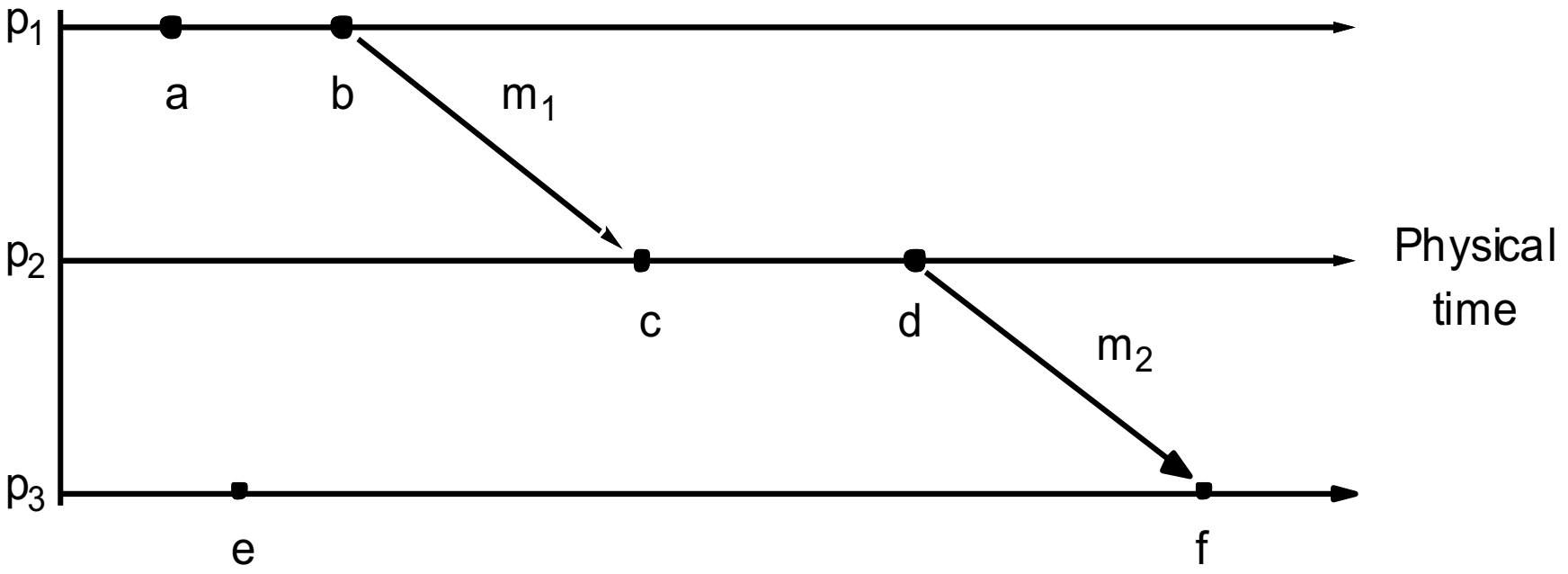
What can we say about **e** and **d**?

Lamport's Logical Clock

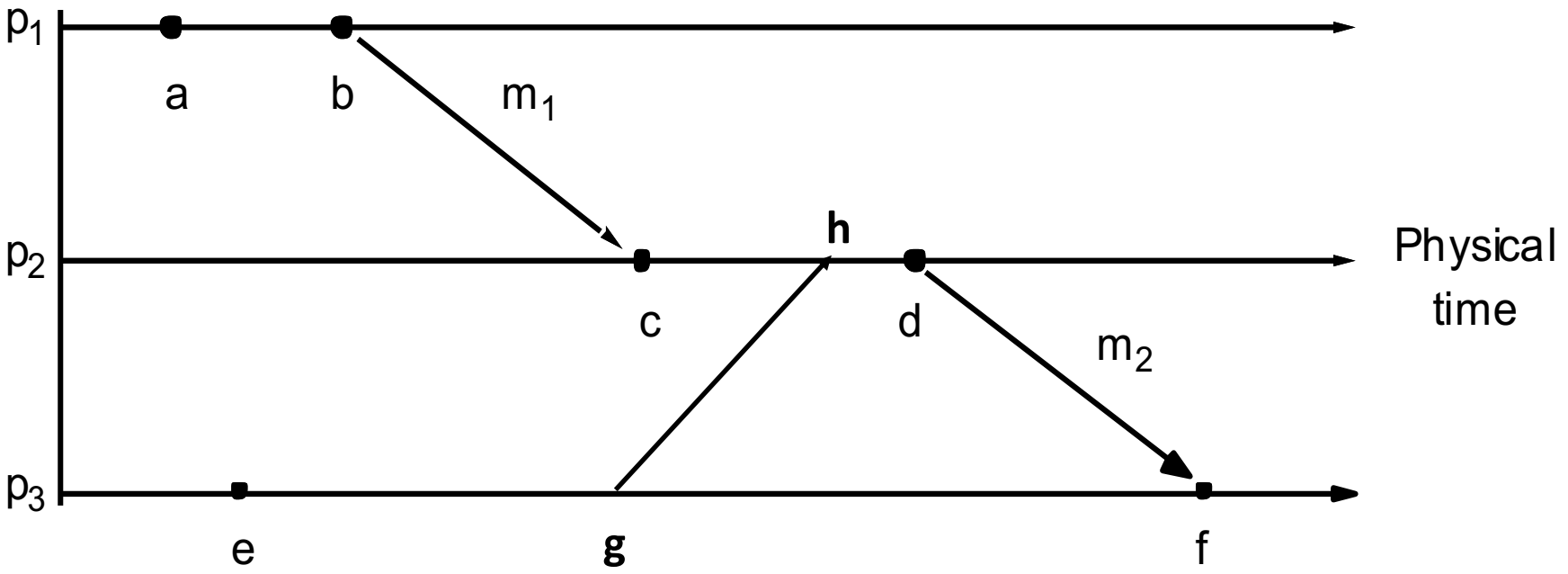
- Logical timestamp for each event that captures the *happened-before* relationship.

- *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.

Logical Timestamps: Example



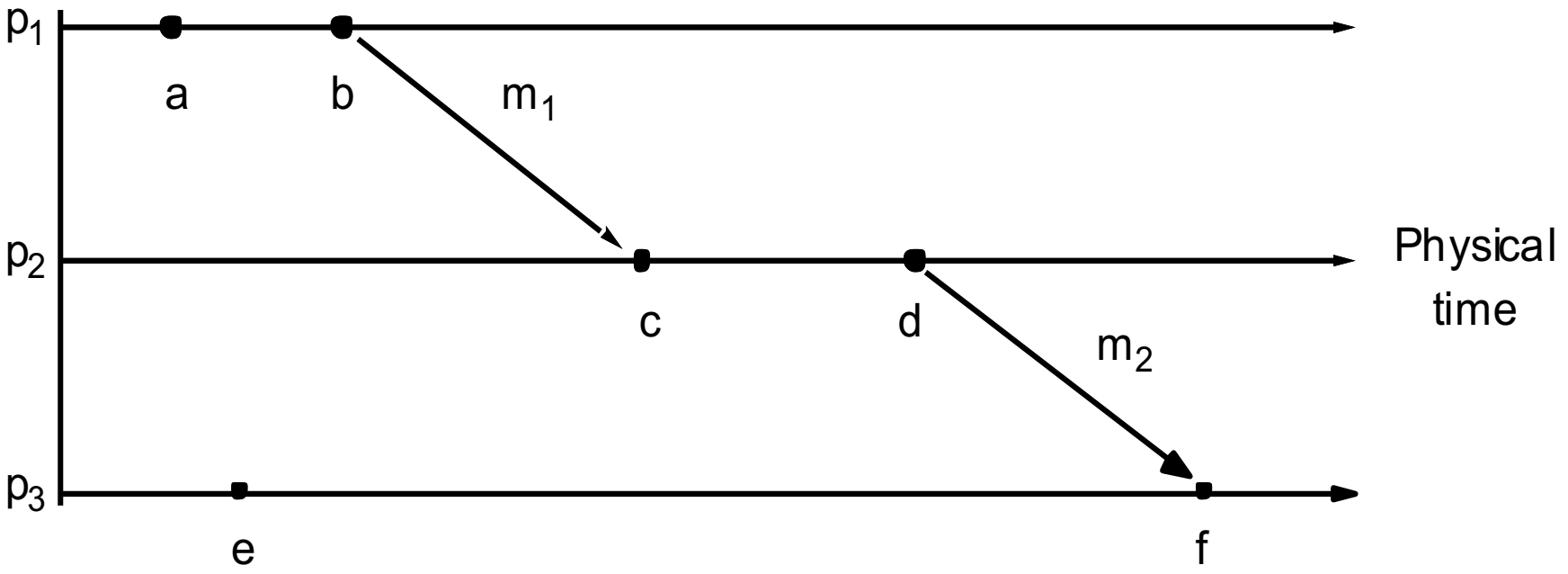
Logical Timestamps: Example



Lamport's Logical Clock

- Logical timestamp for each event that captures the *happened-before* relationship.
- If **e → e'** then **L(e) < L(e')**
- What if **L(e) < L(e')**?

Logical Timestamps: Example

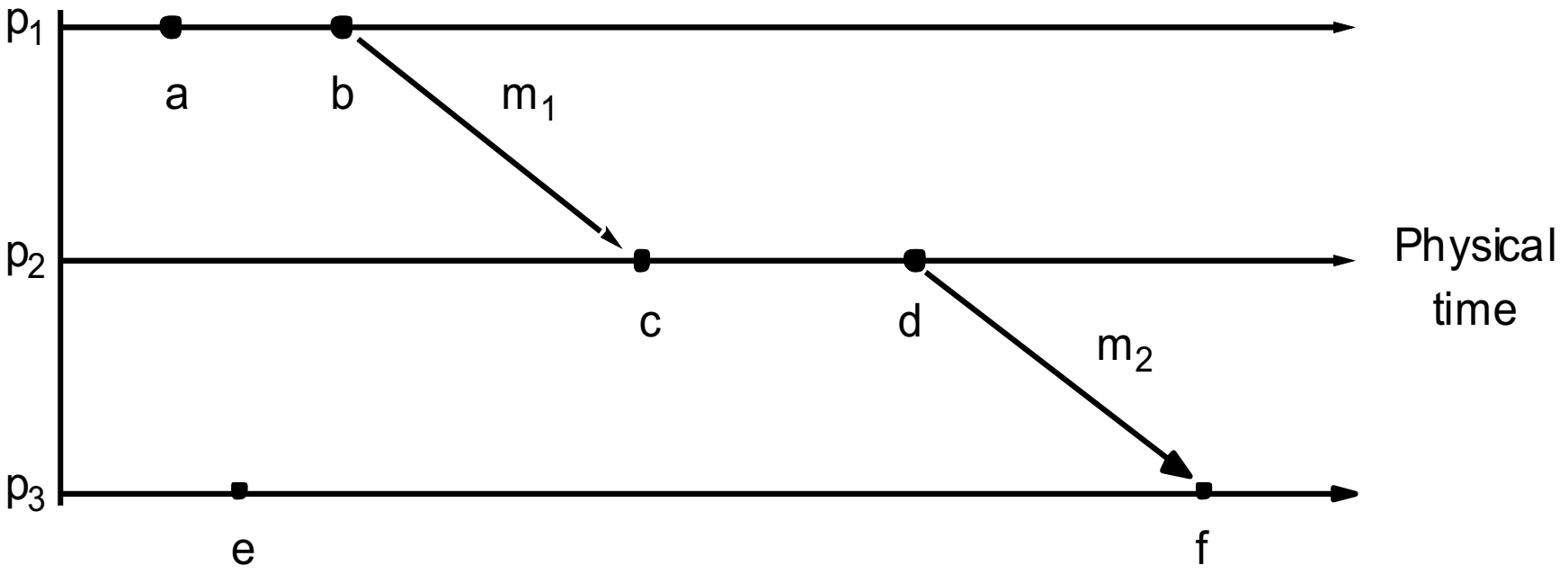


Vector Clocks

- Each event associated with a vector timestamp.
- Each process maintains vector of clocks V_i
  - $V_i[j]$ is the clock for process p_j

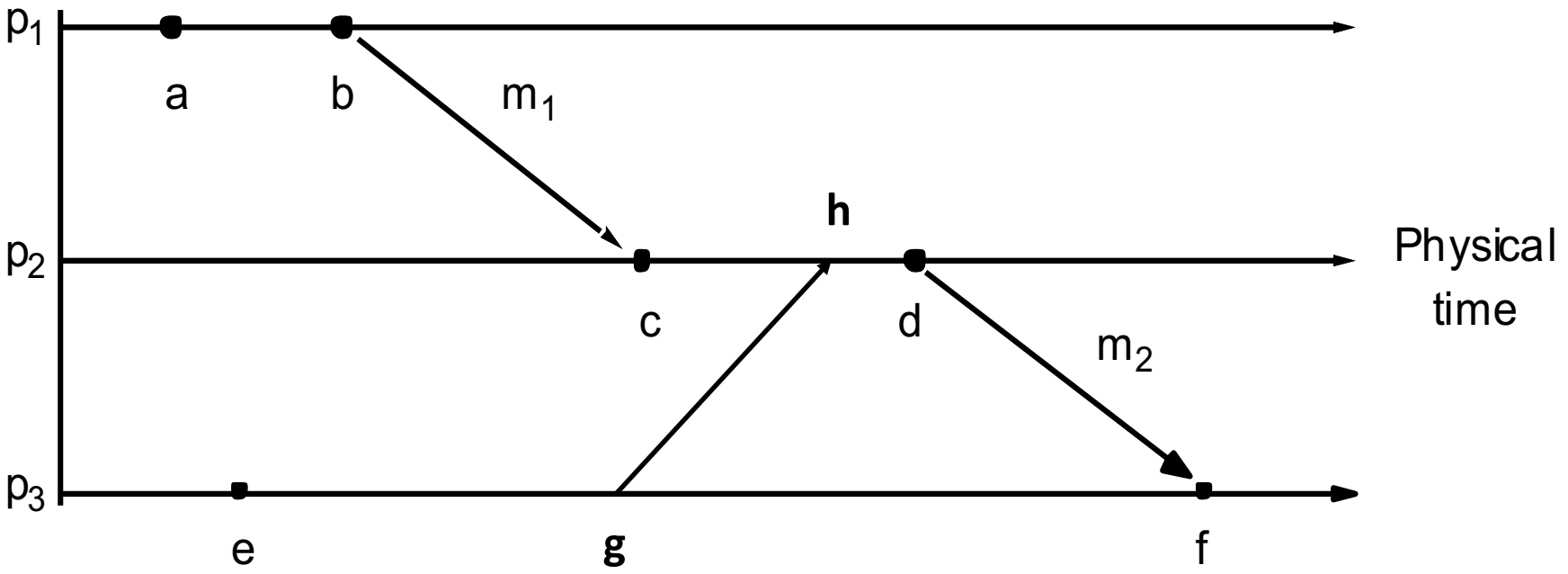
- Algorithm: each process p_i:
  1. initializes local clock $V_i[j] = 0$
  2. increments $V_i[i]$ before timestamping each event.
  3. piggybacks V_i when sending a message.
  4. upon receiving a message with clock value t
     - sets $V_i[j] = \max(V_i[j], t[j])$ for all j=1…n.
     - increments $V_i[i]$ (as per point 2).

Vector Timestamps: Example



Assign a vector timestamp to each event!

Vector Timestamps: Example



Assign a vector timestamp to each event!

Comparing Vector Timestamps

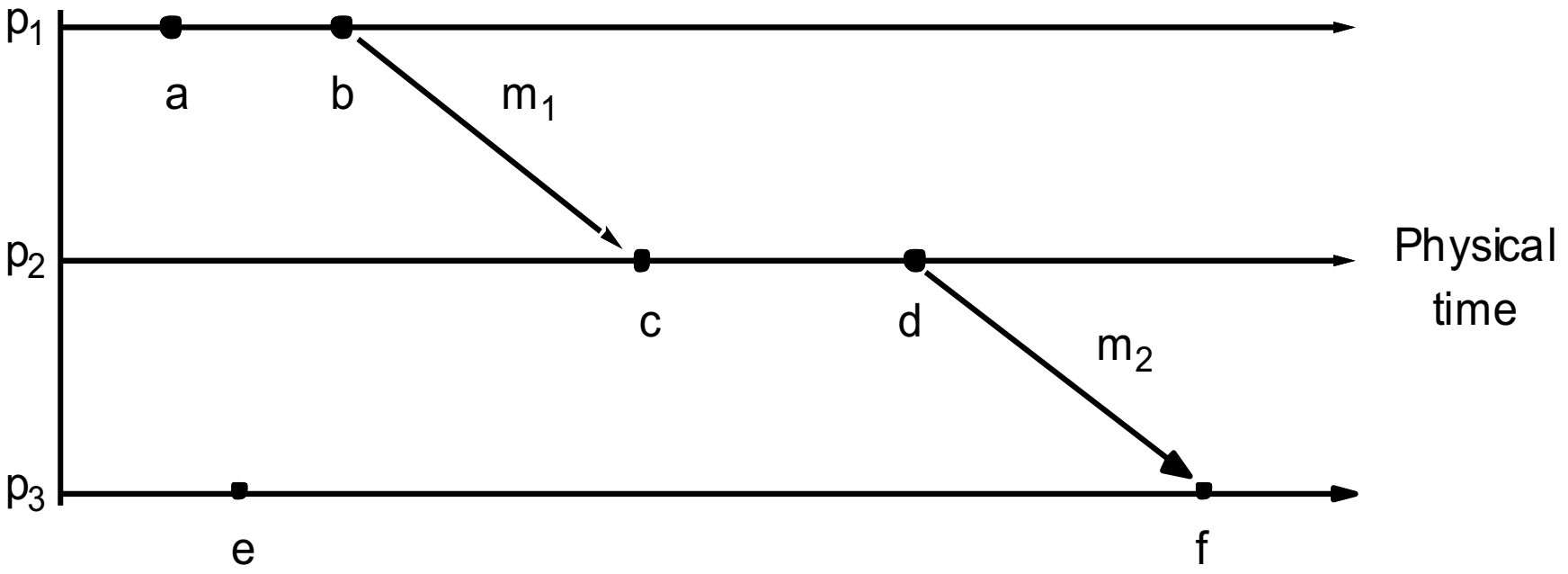
- Let $V(e) = V$ and $V(e') = V'$

- $V = V'$, iff $V[i] = V'[i]$, for all $i = 1, \ldots, n$
- $V \leq V'$, iff $V[i] \leq V'[i]$, for all $i = 1, \ldots, n$
- $V < V'$, iff $V \leq V'$ & $V \neq V'$

  iff $V \leq V'$ & $\exists$ j such that $(V[j] < V'[j])$

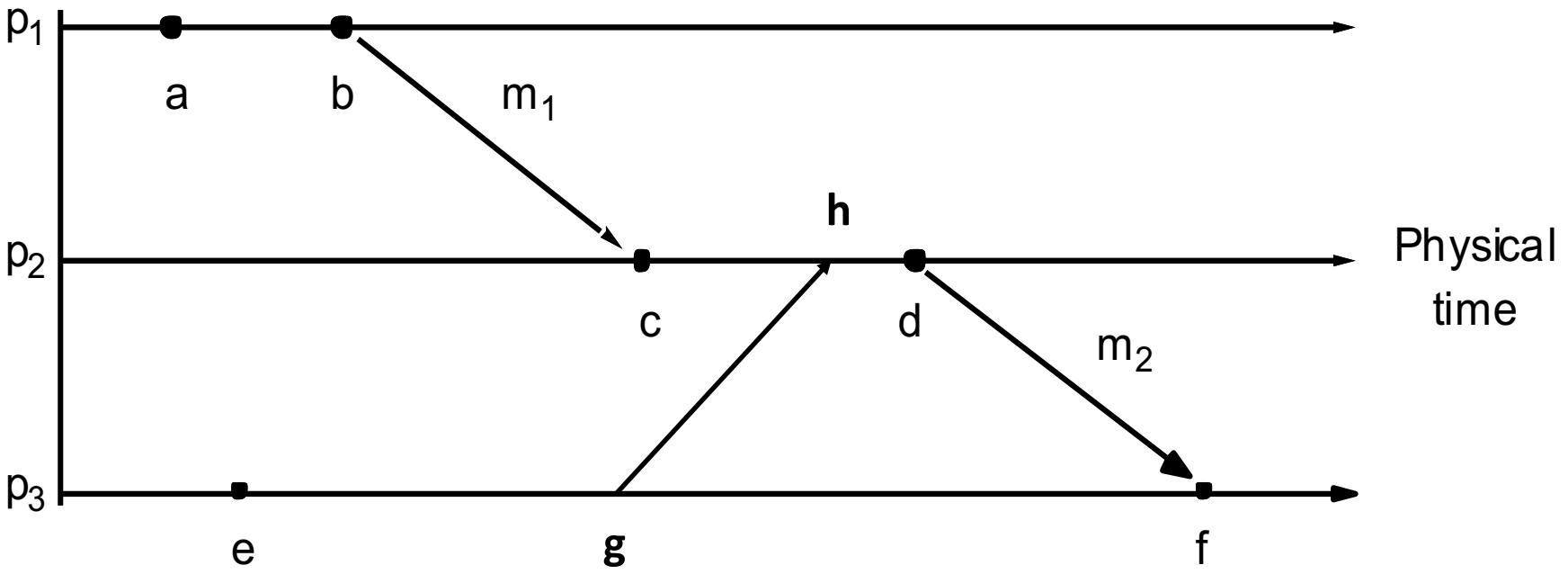
- $e \rightarrow e'$ iff $V < V'$
  - ($V < V'$ implies $e \rightarrow e'$) and ($e \rightarrow e'$ implies $V < V'$)
- $e \parallel e'$ iff ($V \not< V'$ and $V' \not< V$)

Vector Timestamps: Example



Compare vector timestamps between e & f and e & d.

Vector Timestamps: Example



Compare vector timestamps between e & f and e & 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.

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.