CS 425/ECE 428
Distributed Systems

Lecture 2

Time & Synchronization

Reading: 11.1-11.4

Klara Nahrstedt

Acknowledgement

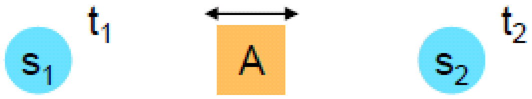
- **The slides during this semester are based on ideas and material from the following sources:**
  - **Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.**
  - **Slides from Professor S. Gosh's course at University o Iowa.**

Why synchronize clocks?

- **You want to catch the 10 Gold West bus at the Illini Union stop at 6.05 pm, but your watch is off by 5 minutes**
  - **What if your watch is Faster by 5 minutes?**
  - **What if your watch is Late by 5 minutes?**

- **Two sharpshooters in a multiplayer online game kill the same target. Who gets the point?**

- **Object A is observed by S1 and S2 at local times t1 and t2. Which way is A moving? How fast?**



- **Synchronizing clocks helps us**
  - **Time-stamping events (provide 'Fairness')**
  - **Ordering events (provide 'Correctness')**

Plan for today

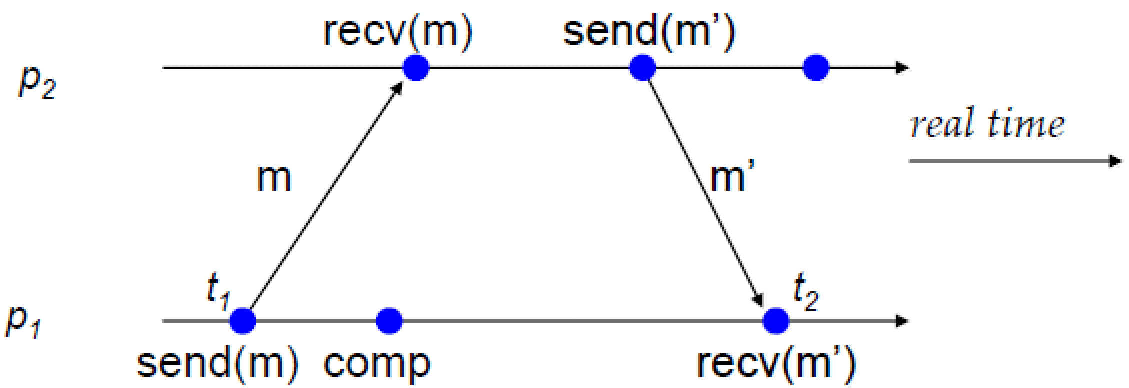
- **Sources of time**
- **How to synchronize clocks?**
- **Can we define sequence of events without physical clocks?**

Time Sources

- De Facto Primary Standard – International Atomic Time (TAI)
  - Keeping of TAI started in 1955
  - 1 atomic second = 9,192,631,770 orbital transitions of Cs^{133} (Caesium)
  - 86400 atomic seconds = 1 solar day – 3 ms
- Coordinated Universal Time (UTC) – International Standard
  - Keeping of UTC started 1961
  - Derived from TAI by adding leap seconds to keep it close to solar time
  - UTC source signals are synchronized
  - UTC time is re-transmitted by GPS satellites
- Local clocks are based on oscillators

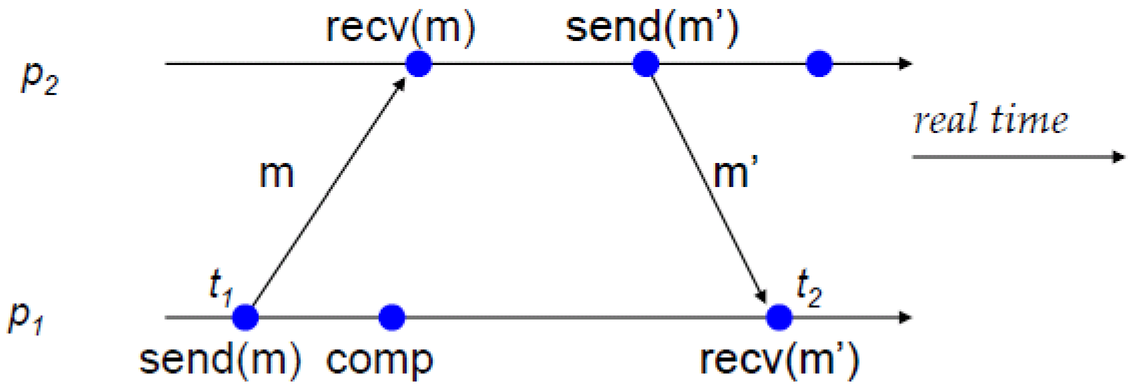
Terminology

- Distributed System (DS) consists of N processes $p_1, p_2, .. p_N$
- Process p_i, $i \in \{1,\ldots N\}$
  - State: values of local variables including time
    - » $C_i(t)$: the reading of the local clock at process i when the real time is t
  - Actions: send message [*send(m)*], receive message [*recv(m)*], compute [*comp*]
- Occurrence of an action is called an ***event***



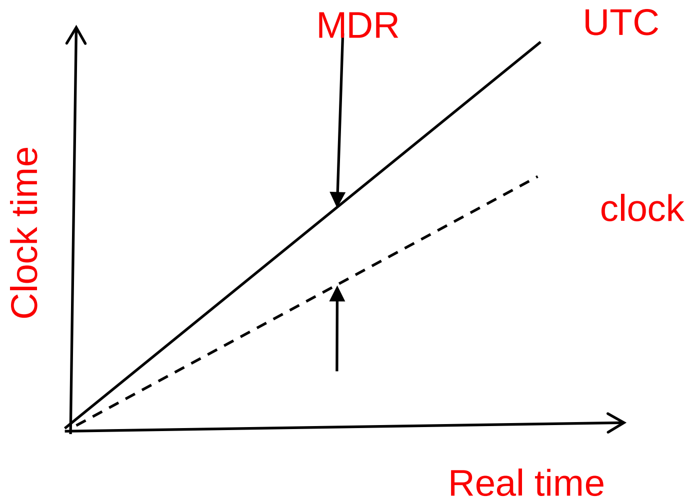
Terminology

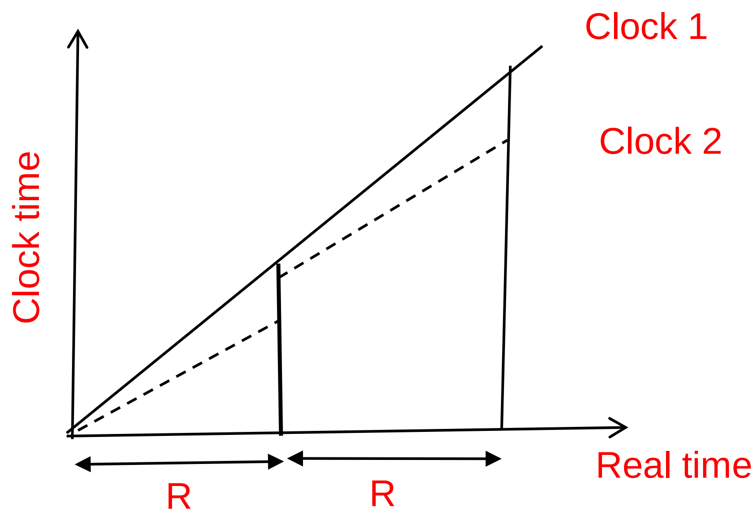
- Events within process p_i can be assigned timestamp and thus ordered
- Events across different processes in DS need to be ordered, but
- □ Clocks in DS across processes are not synchronized
  - Process clocks can be different
- Need algorithms for either
  - time synchronization or
  - telling which event happened before which



Definitions

- **Skew:**
  - $s(t) = C_i(t) - C_j(t)$
- **Maximum Drift Rate (MDR)** ρ
  - $|t - C_i(t)| \leq \rho t$
  - $(1-\rho) \leq dC_i(t)/dt \leq (1+\rho)$
- **Synchronization interval** R **and synchronization bound** D
  - $| C_i(t) - C_j(t) | \leq 2\rho t$
  - $| C_i(R) - C_j(R)| \leq 2\rho R \leq D$
  - $R \leq D/2\rho$
  - **This calculation ignored propagation delays**





Synchronizing Physical Clocks

- **External synchronization: For a synchronization bound *D > 0*, and for source *Source(t)* of UTC time,**

$$\left| Source(t) - C_i(t) \right| < D,$$

for *i=1,2,...,N* and for all real times *t* .

Clocks C_i are accurate within the bound *D*.

- **Internal synchronization: For a synchronization bound *D>0*,**

$$\left| C_i(t) - C_j(t) \right| < D$$

for *i, j=1,2,...,N* and for all real times *t* .

Clocks C_i agree within the bound *D*.

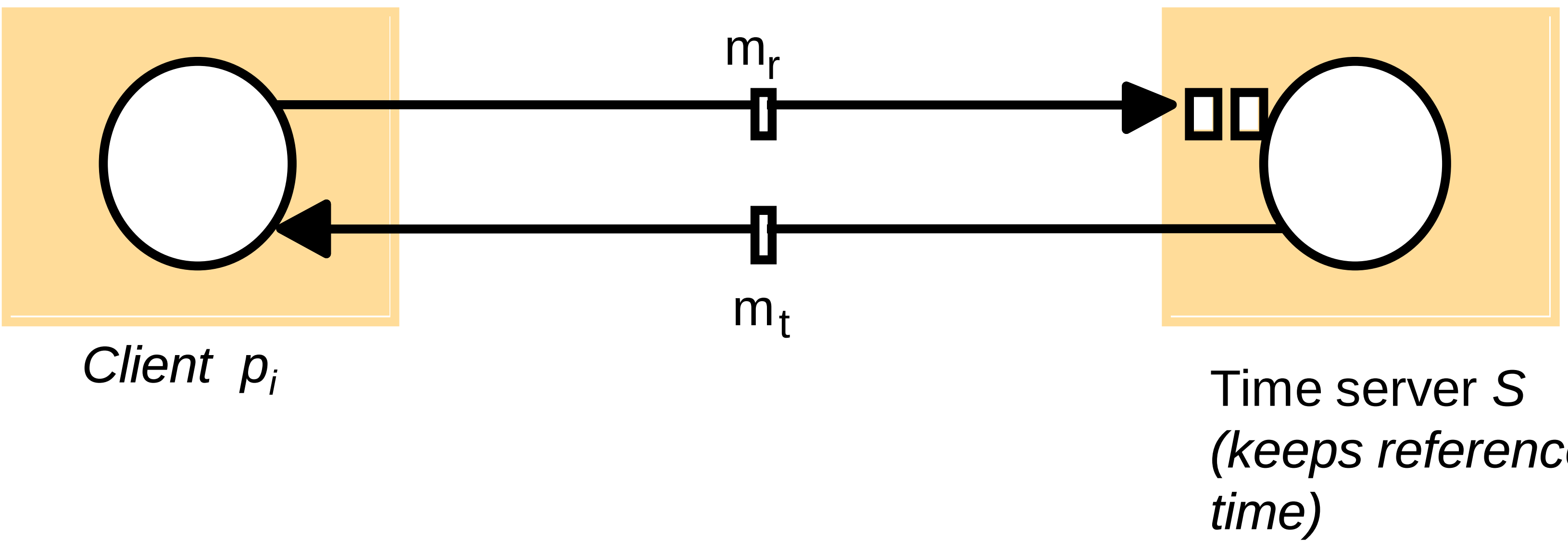
Internal Synchronization: Berkeley Algorithm

- Use *elected leader process* to ensure maximum skew is ρ among clients
- *Elected leader* broadcasts to all machines for their time,
  - adjusts times received for transmission delay & latency,
  - averages times after removing outliers
  - tells each machine how to adjust.
- In some systems multiple time leaders are used.

☹Averaging client's clocks may cause the entire system to drift away from UTC over time

☹Failure of the leader requires some time for re-election, so accuracy cannot be guaranteed

External Synchronization: Cristian's Method

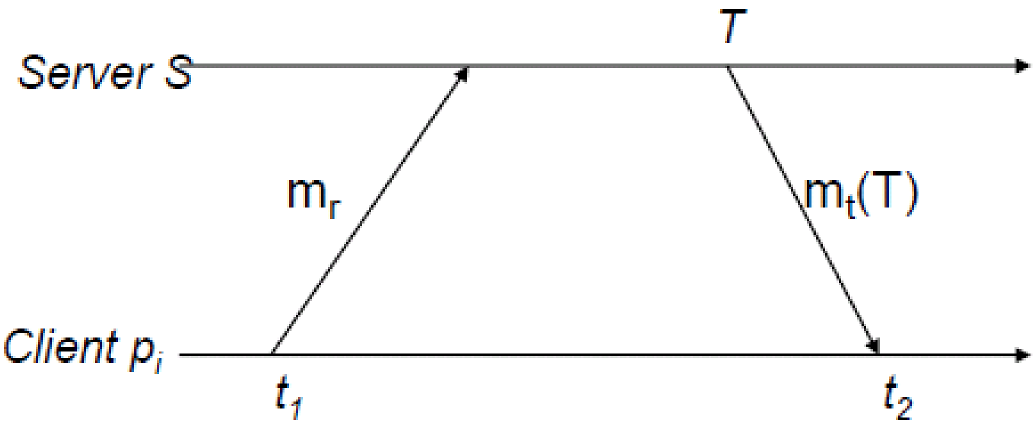


m_r - message where client p_i asks time server S for time

m_t - message where time server responds with its current time T

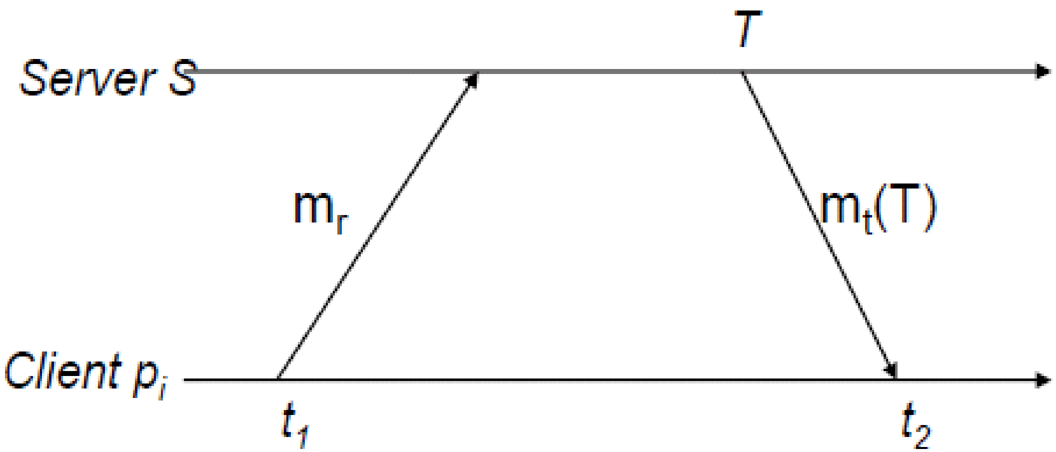
Client p_i uses T in m_t to set its clock

External Synchronization: Cristian's Method



- $RTT = t_1 - t_2$ **(Round Trip Time)**
- **Client sets its clock to** $T + RTT/2$
- **Assumptions:**
  - **RTT measured accurately**
  - **Transmission and computation delays are symmetric**
  - **Server time-stamped the message at the last possible instant before sending it back**

External Synchronization: Cristian's Method

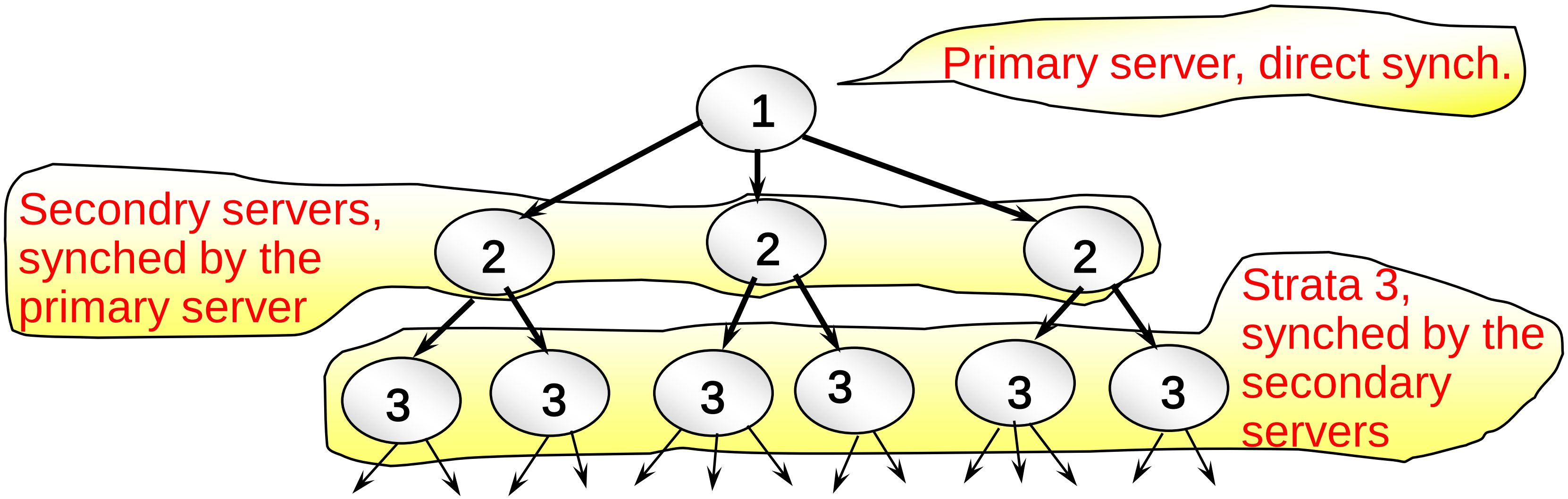


- **Suppose we know the minimum client-server one way transmission delay:** *min*
- **Then => actual time at client could be between**

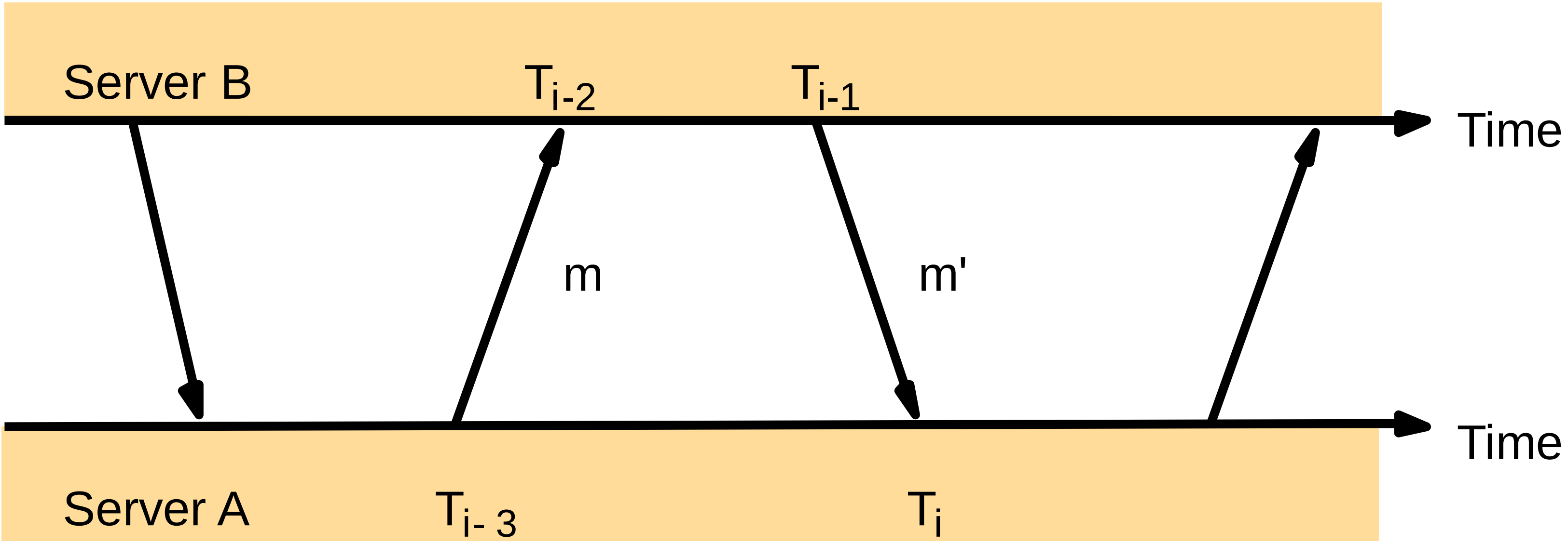
  [T + min, T + RTT - min]
- **Accuracy:** *RTT/2 – min*

The Network Time Protocol (NTP)

- **Provides UTC synchronization service across Internet**
- **Uses time servers to sync. networked processes.**
- **Time servers are connected by sync. subnet tree.**
- **The root is adjusted directly.**
- **Each node synchronizes its children nodes.**



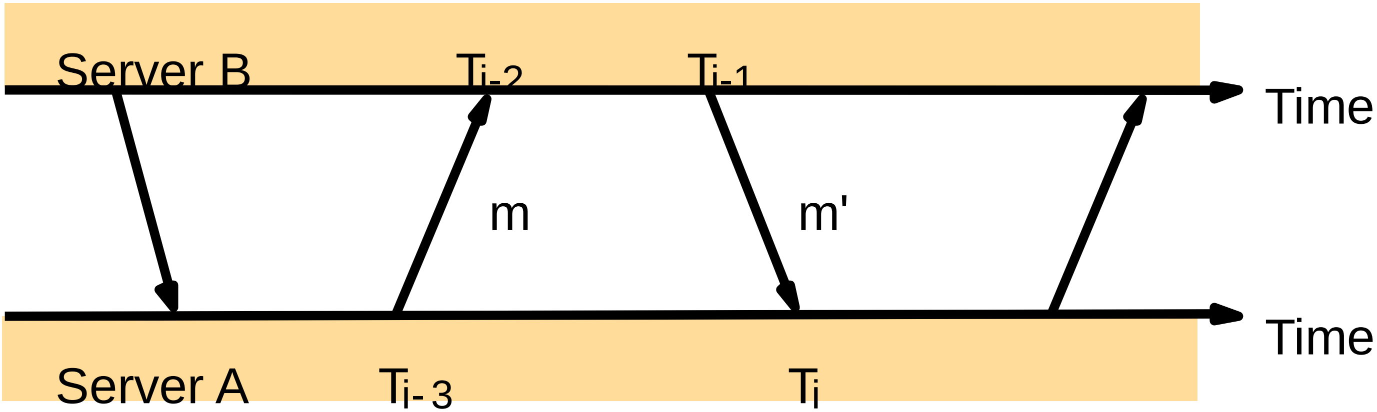
Messages Exchanged Between a Pair of NTP Peers (Connected Servers)



Each message (e.g., *m, m'*) bears **timestamps** of recent message events:

- the local time when the previous NTP message (*m*) was sent (T_{i-3}) and received (T_{i-2}), and
- the local time when the current message (*m'*) was transmitted (T_{i-1}).

Theoretical Base for NTP



$$T_{i-2} = T_{i-3} + t + o$$

$$T_i = T_{i-1} + t' - o$$

adding

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

substracting

$$o = o_i + (t' - t)/2, \text{ where}$$

$$o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2.$$

It can also be shown that

$$o_i - d_i/2 \le o \le o_i + d_i/2.$$

- o_i: estimated skew
- o: true skew of clock at B relative to that at $A = C_A - C_B$
- t and t': actual transmission times for m and m'
- d_i: estimate of accuracy of o_i; total transmission times for m and m'; $d_i = t + t'$

Summary on Physical Clocks

- **Several algorithms for internal and external clock synchronization**
- **Accuracy depends on uncertainty in message delays**
- **We have not discussed failures**
  - **There are algorithms for synchronization with a fraction of failed clock (e.g., Lamport)**
- **Next: Logical Clocks**

Logical Clock

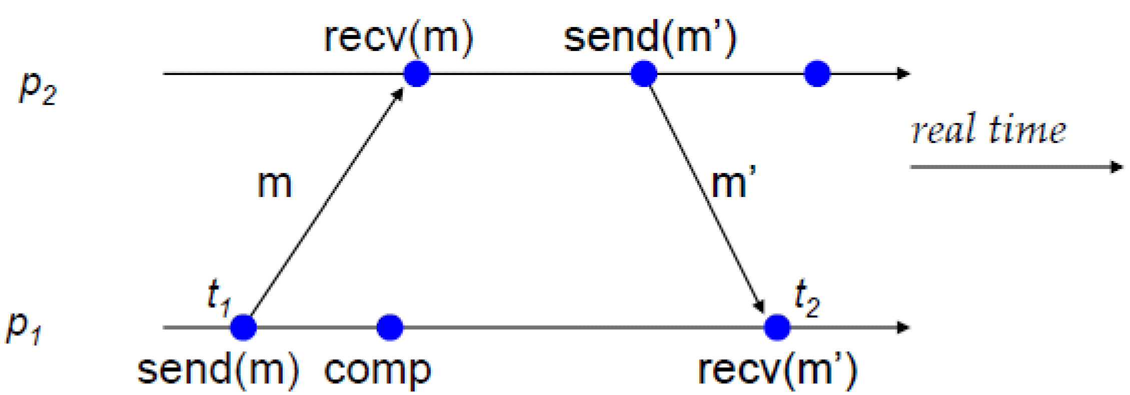
- **Is it always necessary to give *absolute time* to events?**

- **Suppose we can assign *relative time to events*, in a way that does not violate their causality**

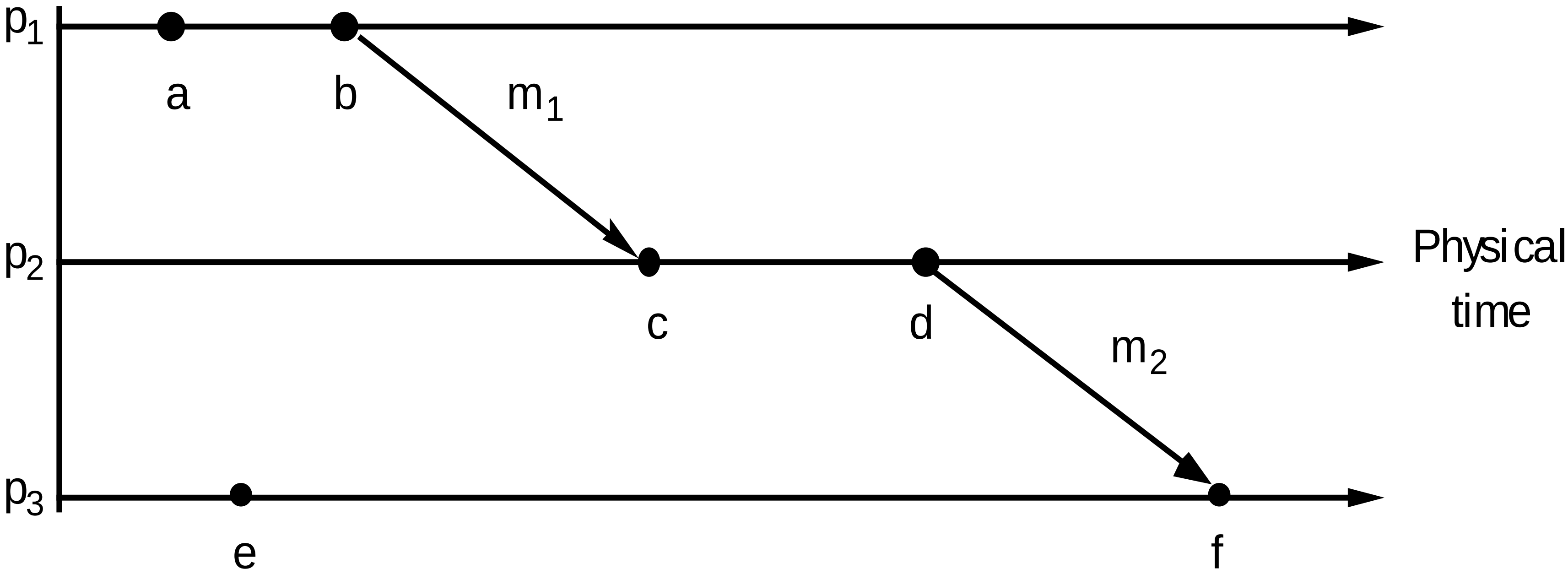
Happens-Before Relation on Events

- Define a relation → on the events as follows
  - On the same process: *a → b iff time (a) < time (b)*
  - If p_1 sends *m* to p_2: *send(m) → recv(m)*
  - Transitivity: *if a → b and b → c then a → c*
- → is called the Happens-Before Relation
- Events *a* and *b* are concurrent if

  not (a → b or b→ a)



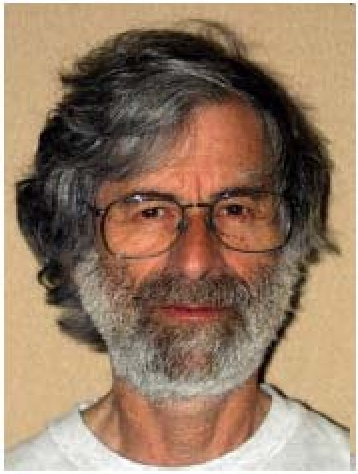
Events Occurring at Three Processes



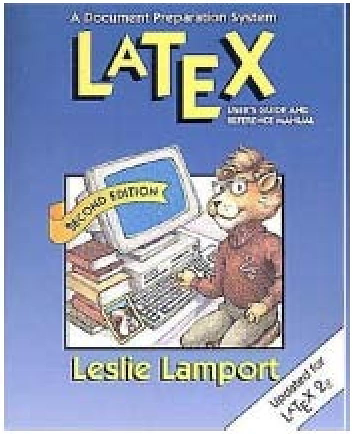
How to construct the happen-before relation in a distributed system?

Lamport's Logical Clock

- **First proposed by Leslie Lamport in 70's**
- **Lamport algorithm assigns logical timestamps to events**



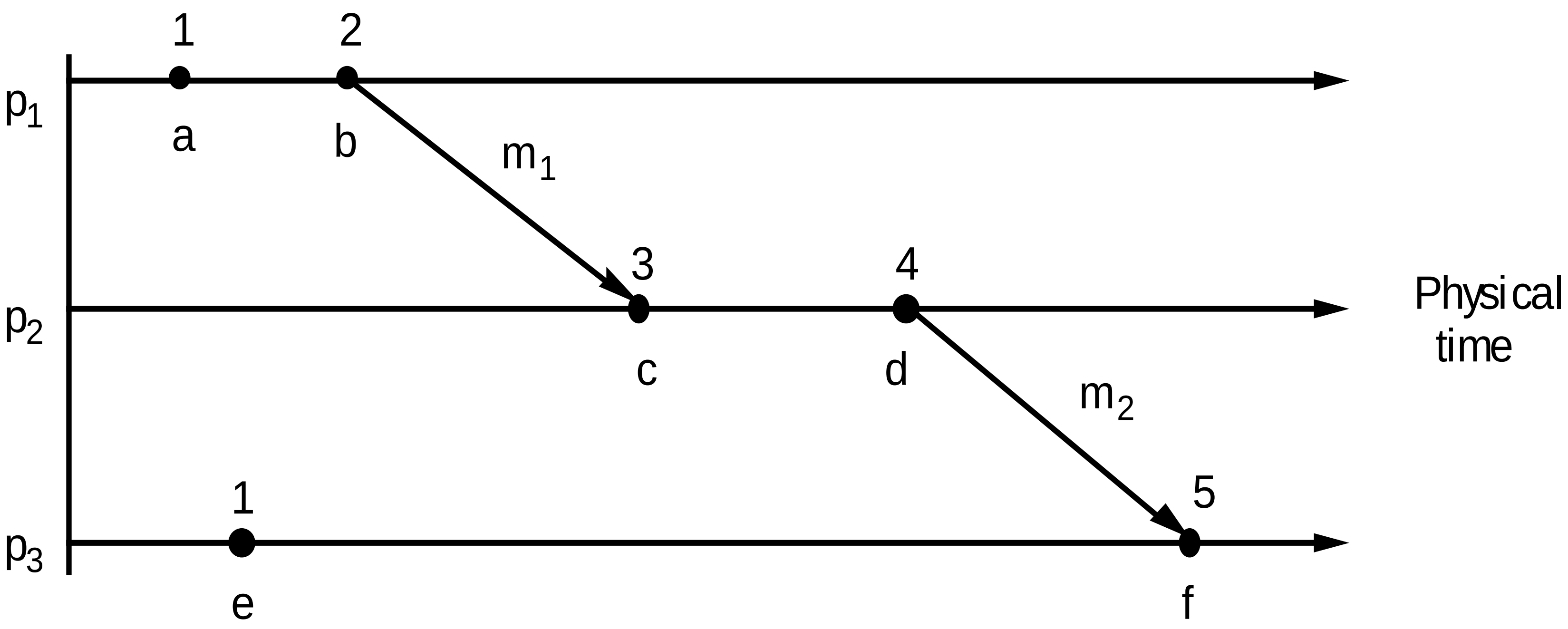
Leslie Lamport



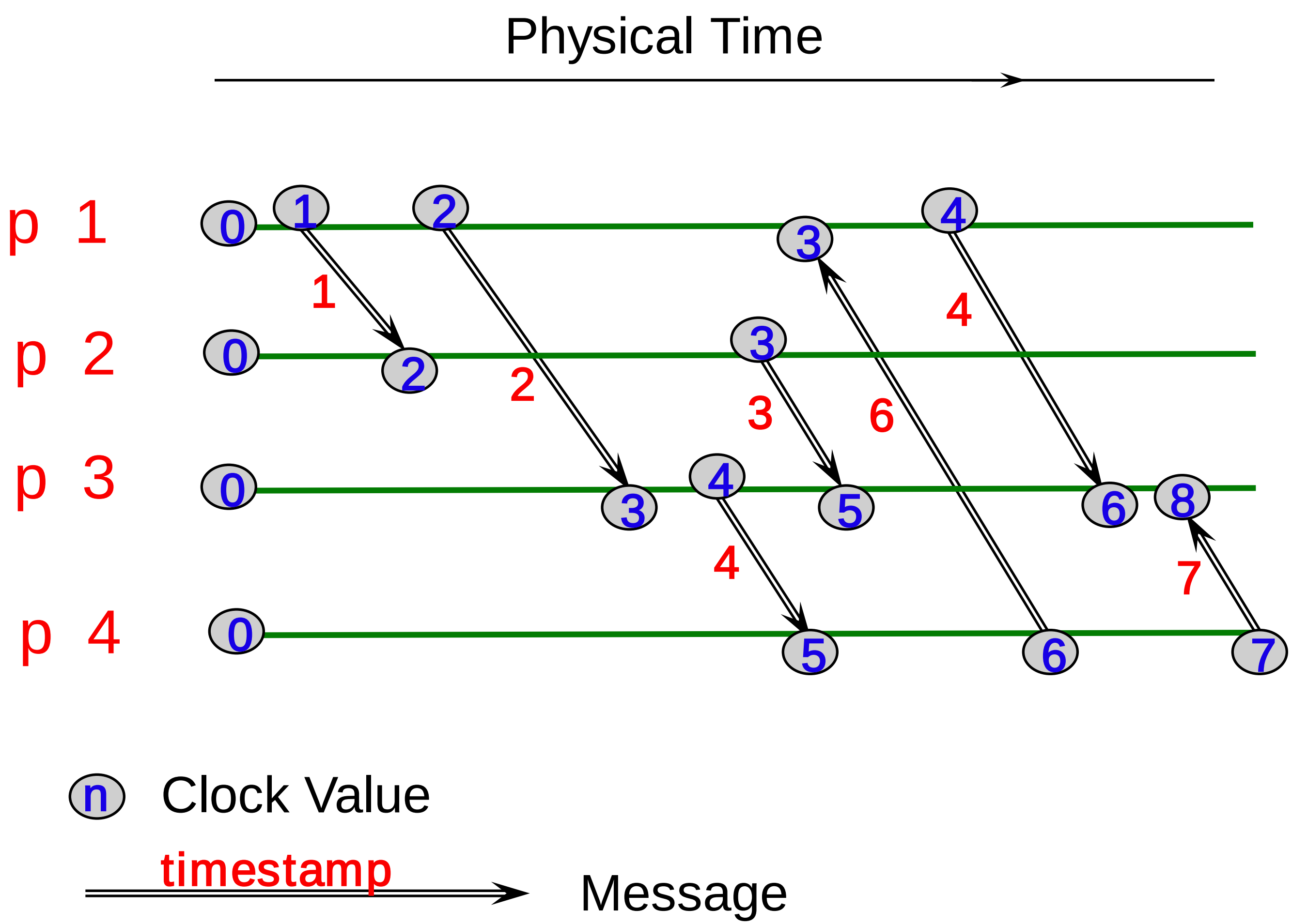
- **Each process has a counter (logical clock)**
- **Initially logical clock is set to 0**
- **Process increments its counter when a *send* or computation (*comp*) step is performed**
- **Counter is assigned to event as its timestamp**
- **send(message) event carries its timestamp**
- **On recv(message) event, the counter is updated by**

max(local clock, message timestamp) + 1

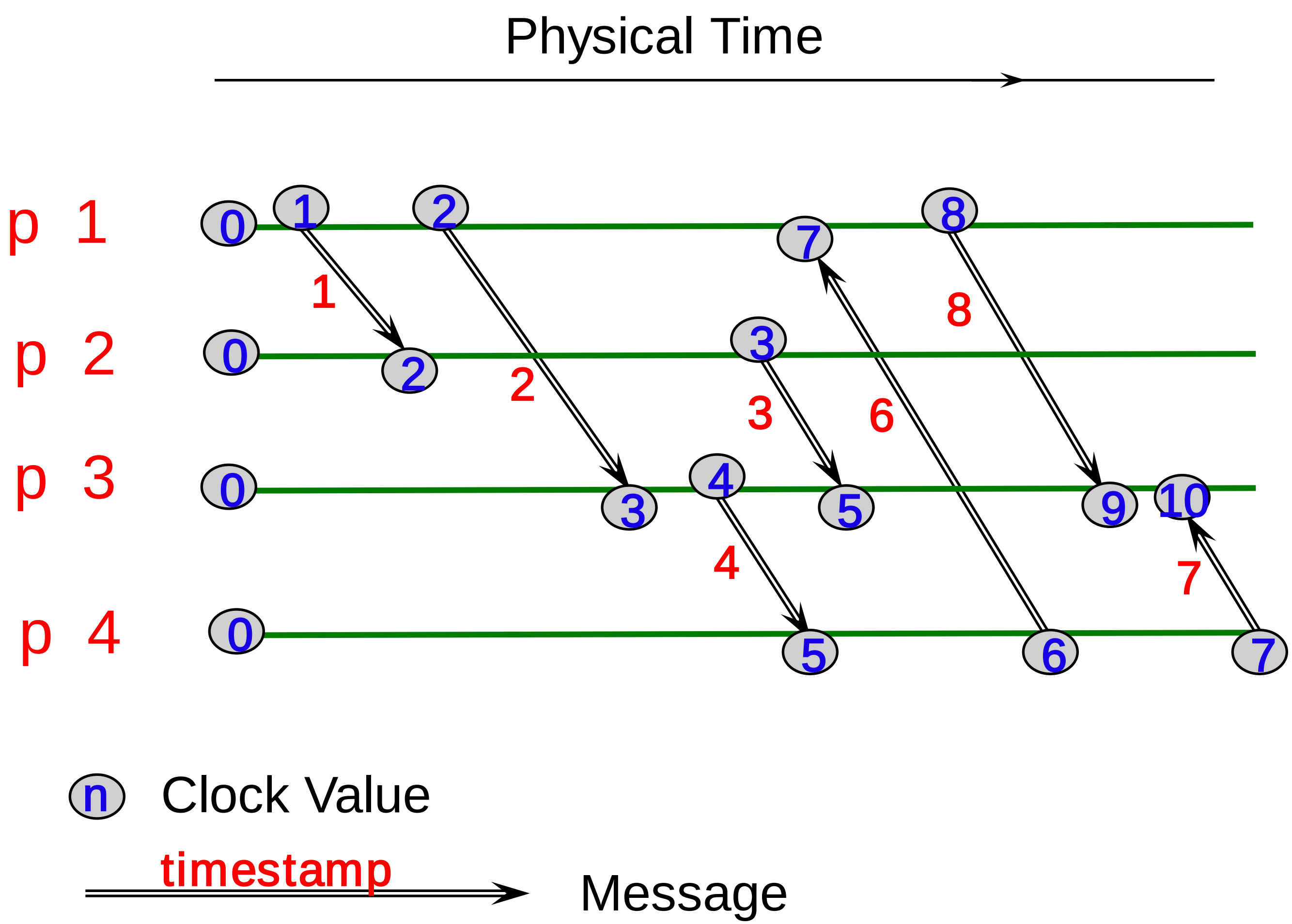
Lamport Timestamps



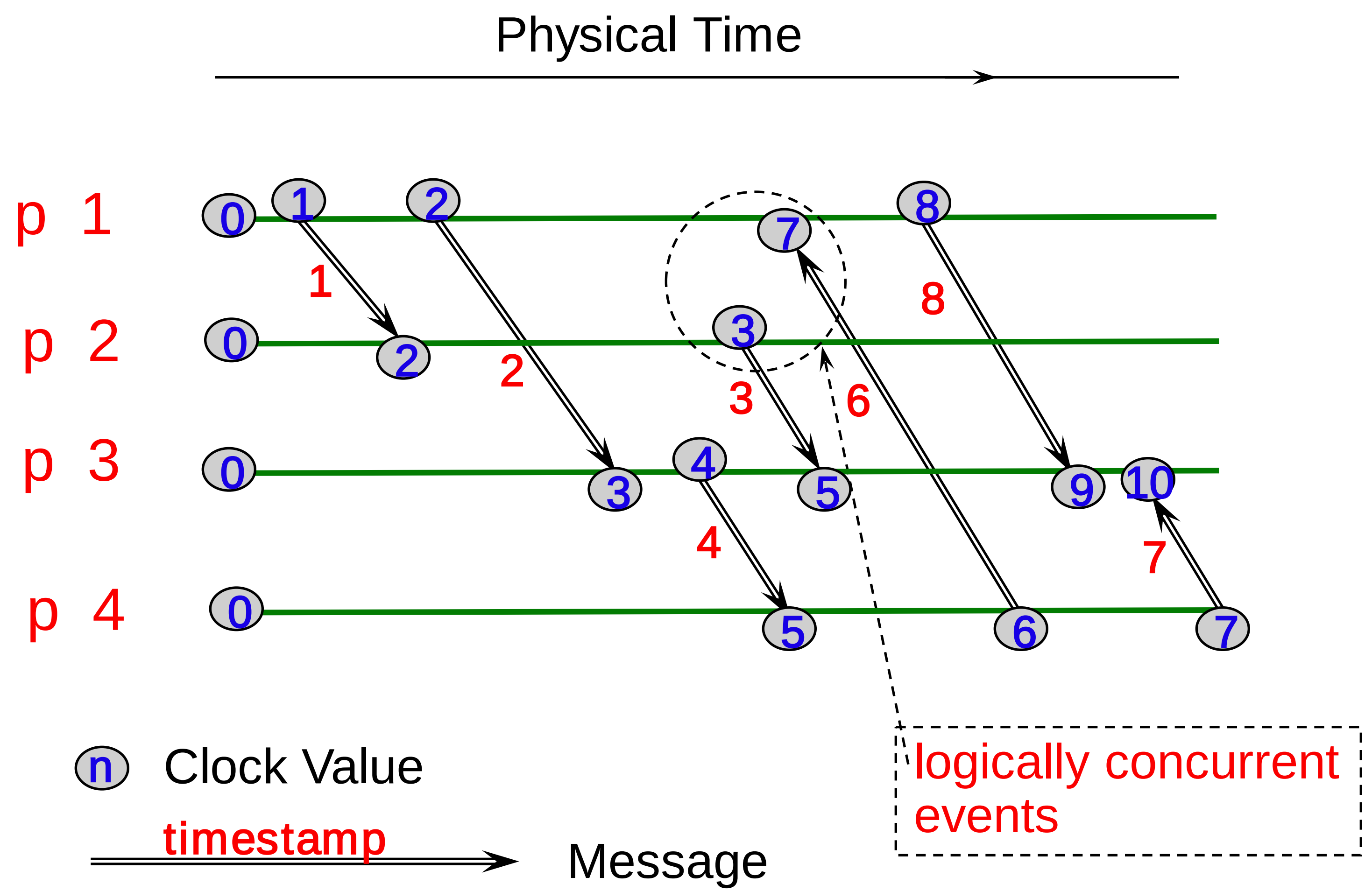
Spot the Mistake



Corrected Example: Lamport Logical Time



One thing to Notice ...



Summary

- Time synchronization for distributed systems
  - Berkeley algorithm
  - Cristian's algorithm
  - NTP
- Relative order of events enough for practical purposes
  - Lamport's logical clocks
- Next class: Logical Clock (Vector Clock) and Global States/Snapshots.
  - Reading: 11.4 and 11.5