CSE 486/586 Distributed Systems

Time and Synchronization

Slides by Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time

- Synchronous and asynchronous systems
- Failure detection
  - Properties: completeness and accuracy
  - Perfect accuracy impossible in asynchronous systems
  - Simple protocols: heartbeating and ping-ack
  - Failure detection for large groups
    - Centralized heartbeat
    - Ring heartbeat
    - All-to-all heartbeat
  - Metrics: bandwidth, detection time, scalability, accuracy

The Next Two Lectures

- Time
- One of the two fundamental challenges in DS:
  - Failure, Ordering
- Recall that we even used time for failure detection!
- In an ideal world:
  - We know exactly when something happens
  - Everyone agrees on that time
- How do we agree on time?
- Why is it hard?

Today

- Servers in the cloud need to timestamp events
- Servers A and B have different clock values
  - You buy an airline ticket online
  - It’s the last airline ticket available on that flight
  - Server A timestamps your purchase at 9h:15m:32.45s
  - What if someone else also bought the last ticket (via server B) at 9h:20m:22.76s?
  - What if Server A was > 10 minutes ahead of server B? Behind?
  - How would you know what the difference in clocks was?

Physical Clocks & Synchronization

- Some definitions: Clock Skew versus Drift
  - Clock Skew: Relative difference in clock *values* of two processes
  - Clock Drift: Relative difference in clock *frequencies* of two processes
- *Clock drift will cause skew to continuously increase.*
- Real-life examples
  - Ever seen "make: warning: Clock skew detected. Your build may be incomplete."?
  - It's reported that in the worst case, there's 1 sec/day drift in modern HW.
  - Almost all physical clocks experience this.

Time Standards

- Time is a big deal in a lot of disciplines.
- Consequently, there are many physical solutions:
  - Frequency standards: Rb, Cs, OCXO, …
  - Time services: WWV/WWVB, GPS, …
- It turns out agreeing on physical time is quite difficult
  - Propagation delays (speed of light!)
  - Relativistic effects
- Absolute time has broadly settled on Coordinated Universal Time (UTC)

Synchronizing Physical Clocks

- $C_i(t)$: the reading of the software clock at process *i* when the real time is *t*.
- External synchronization: For a synchronization bound *D>0*, and for source S of UTC time,

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

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

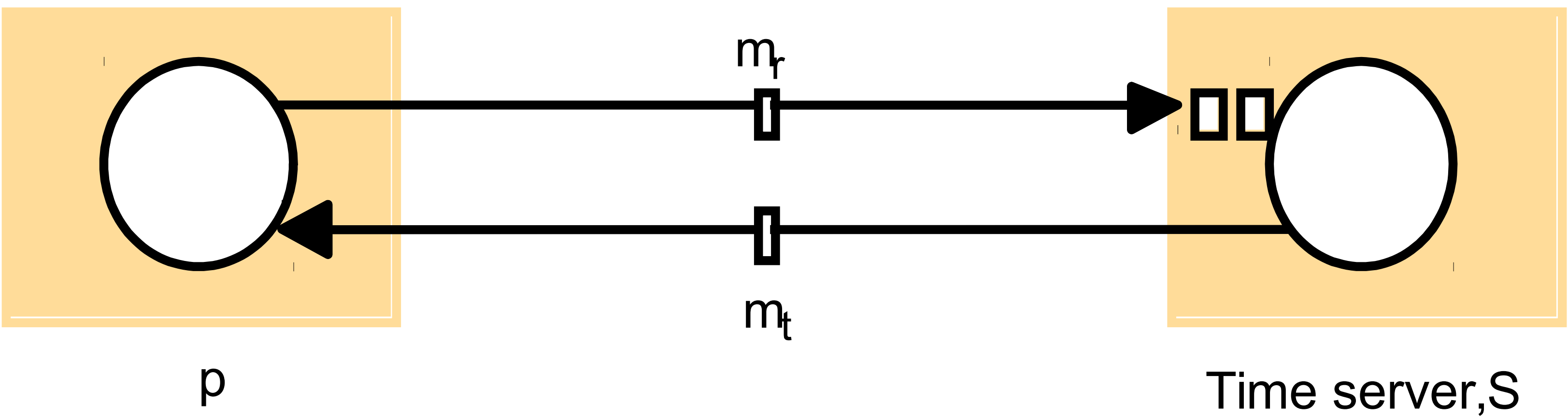
  Clocks C_i are accurate to within the bound *D*.
- Internal synchronization: For a synchronization bound *D>0*,

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

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

  Clocks C_i agree within the bound *D*.
- External synchronization with *D* ⇒ Internal synchronization with *2D*
- Internal synchronization with *D* ⇒ External synchronization with ??

Clock Synchronization Using a Time Server



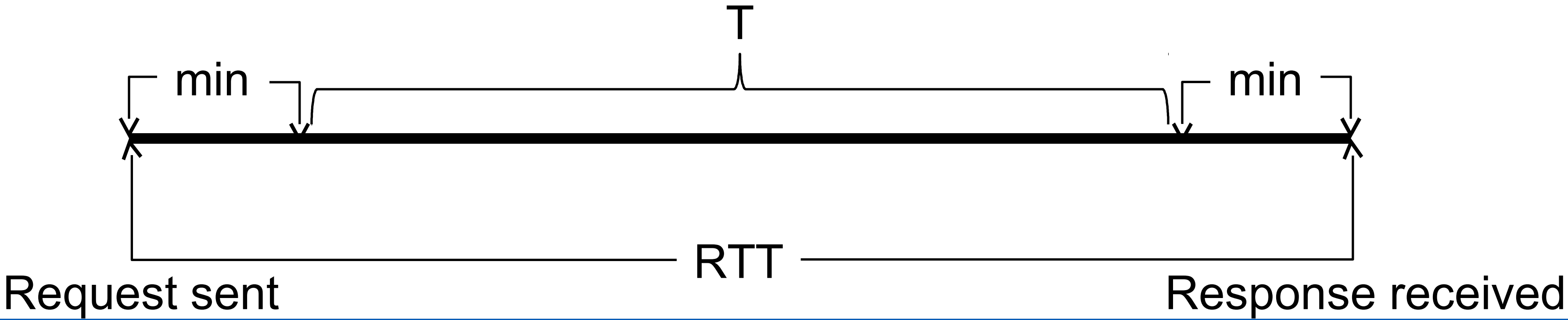
- Client: "What time is it?"
- Server: "It's *t*."
- Any difficulty?

Cristian's Algorithm

- Uses a *time server* to synchronize clocks
- Mainly designed for LAN
- Time server keeps the reference time (say UTC)
- A client asks the time server for time, the server responds with its current time T, and the client uses the received value T to set its clock
- Network round-trip time introduces error.
- So what do we need to do?
  - Estimate one-way delay

Cristian's Algorithm

- Let *RTT* = *response-received-time* – *request-sent-time* (measurable at client)
- Also, suppose we know
  - The minimum value *min* of the client-server one-way transmission time [Depends on what?]
  - That the server timestamped the message at the last possible instant before sending it back
- Then the actual time is between [T+min,T+RTT— min]

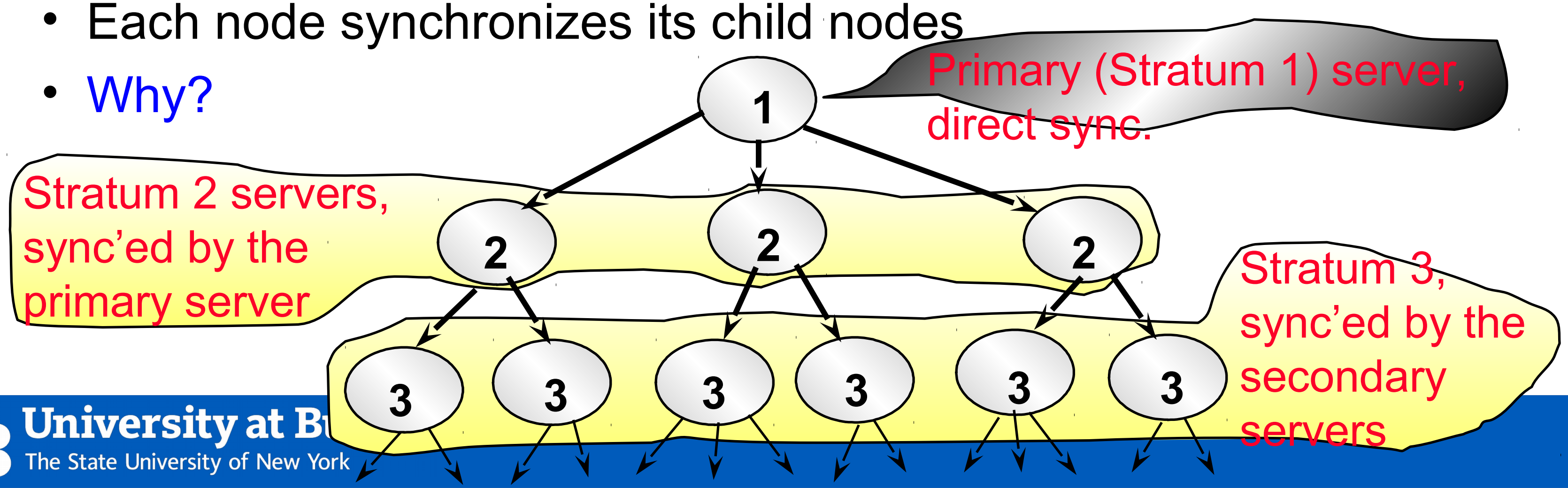


Cristian’s Algorithm

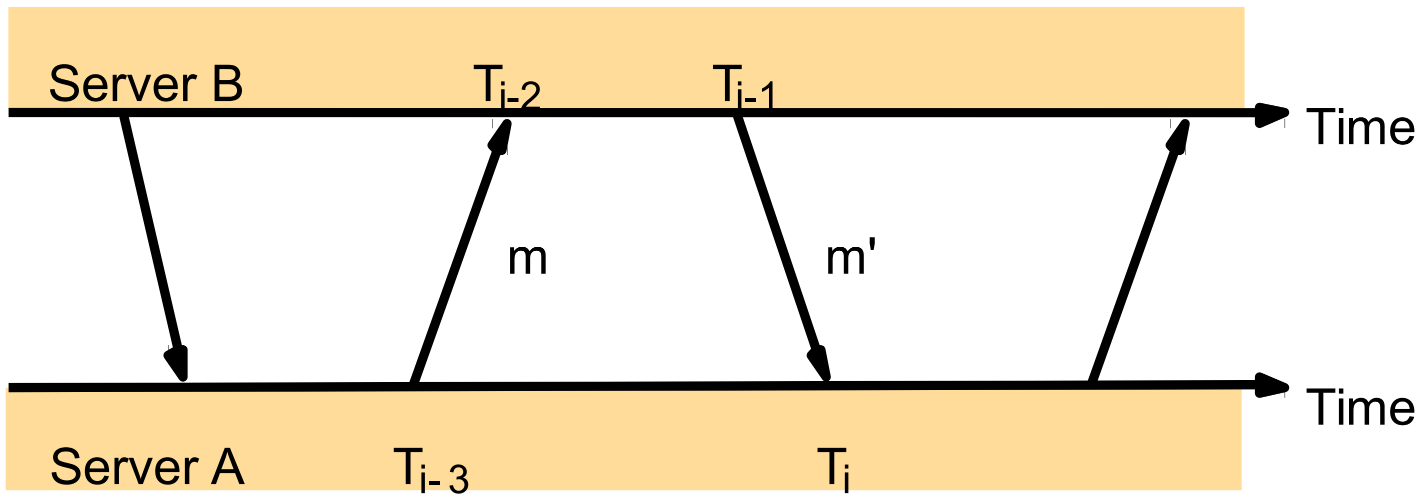
- (From previous slide), the accuracy is: +-(RTT/2 – min)
- Cristian’s algorithm [1]
  - A client asks its time server.
  - The time server sends its time T.
  - The client estimates the one-way delay and sets its time.
    - It uses T + RTT/2
- Want to improve accuracy?
  - Take multiple readings and use the minimum RTT: tighter bound
  - Ignore unusually long RTTs and repeat the request: remove outliers

The Network Time Protocol (NTP)

- Uses a network of time servers to synchronize all processes on a network.
- Designed for the Internet
  - Why not Cristian's algorithm?
- Time servers are organized into a tree
  - The root is disciplined by UTC
  - Each node synchronizes its child nodes
  - Why?

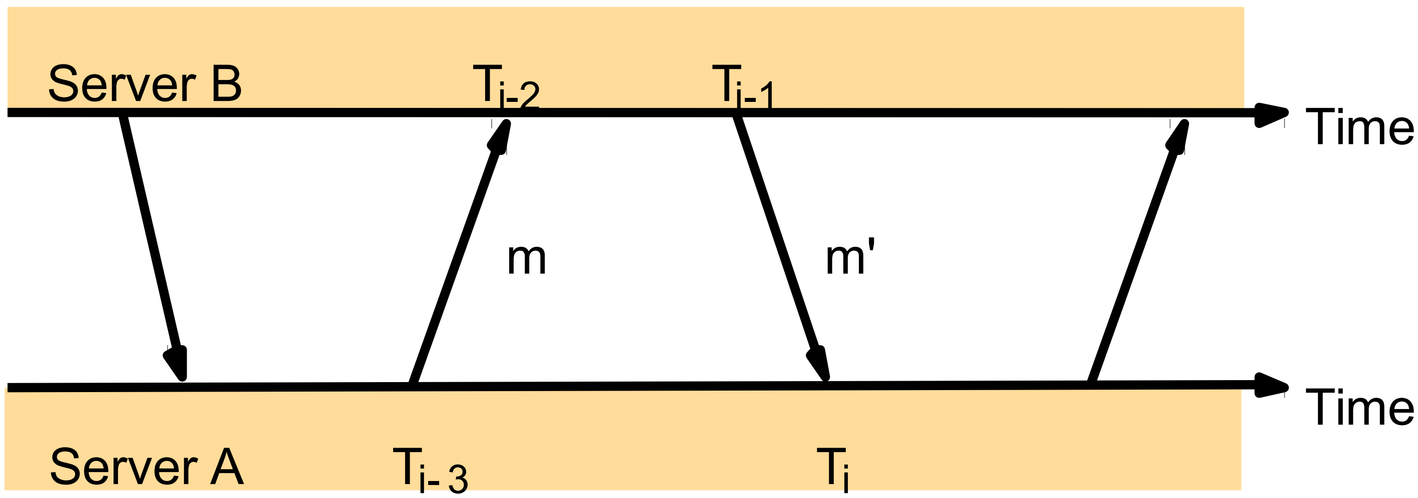


NTP Peer Message Exchange



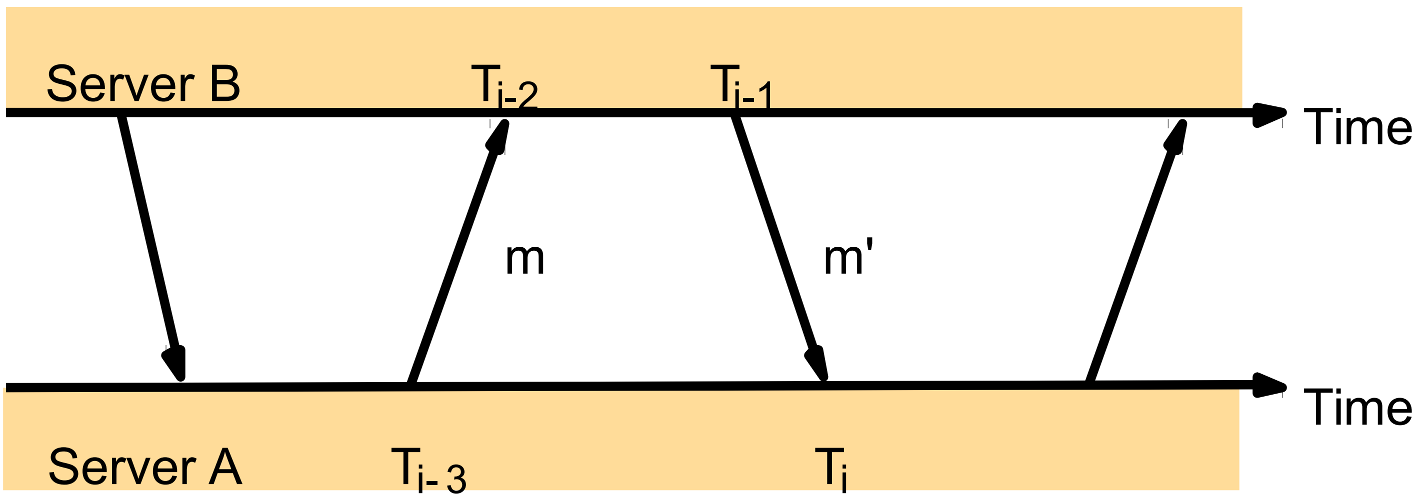
- Each message bears timestamps of recent message events: the local time when the previous NTP message was sent and received, and the local time when the current message was transmitted.

The Protocol



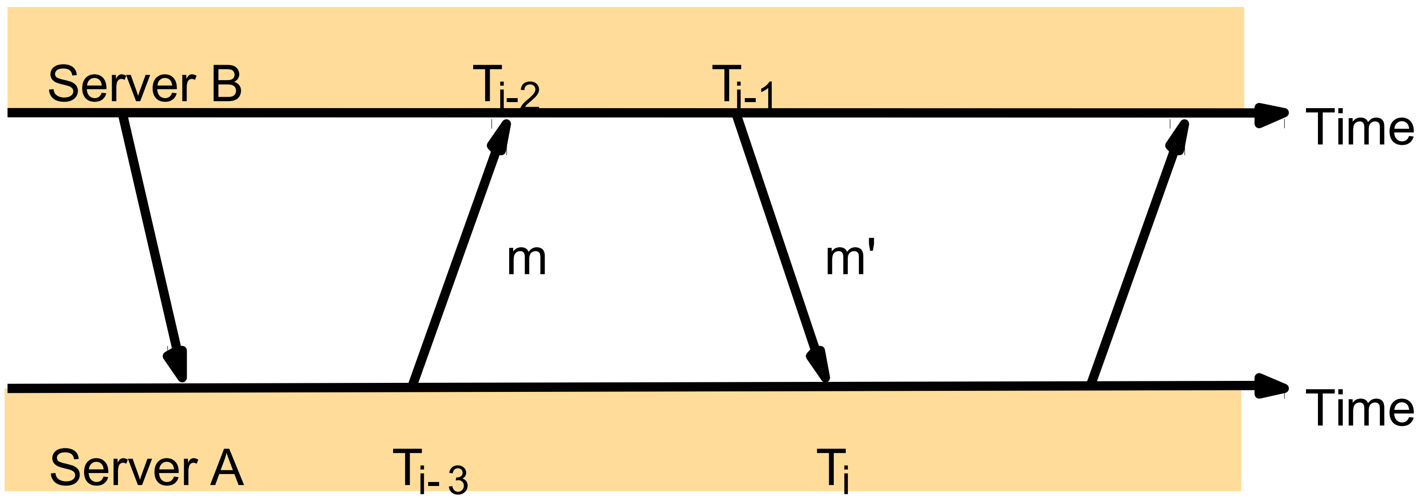
- Compute round-trip delay: $(T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$
- Take the half of the round-trip delay as the one-way estimate: $((T_i - T_{i-3}) - (T_{i-1} - T_{i-2}))/2$

The Protocol [2]



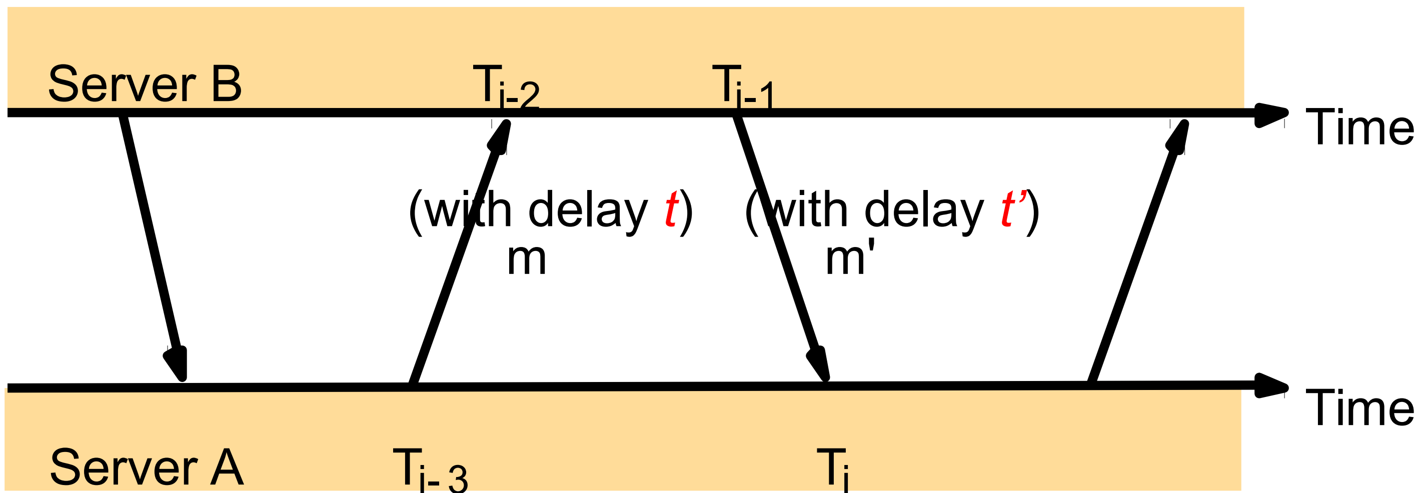
- Compute offset:
  - T_{i-1} + (one-way estimate) - T_i = $((T_{i-2} - T_{i-3}) + (T_{i-1} - T_i))/2$
- Do this with not just one server, but multiple servers.
- Do some statistical analysis, remove outliers, and apply a data filtering algorithm.
  - Out of scope of this lecture

Theoretical Base for NTP



- o_i: estimate of the actual offset between the two clocks
- d_i: estimate of accuracy of o_i ; total transmission times for *m* and *m'*; $d_i = t + t'$

Theoretical Base for NTP



First, let's get o:

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

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

$$\Rightarrow o = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2 + (t' - t)/2$$

Then, get the bound for $(t' - t)/2$:

$$-t' - t \le t' - t \le t' + t \text{ (since } t', t \ge 0)$$

Finally, we set:

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

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

Then we get:

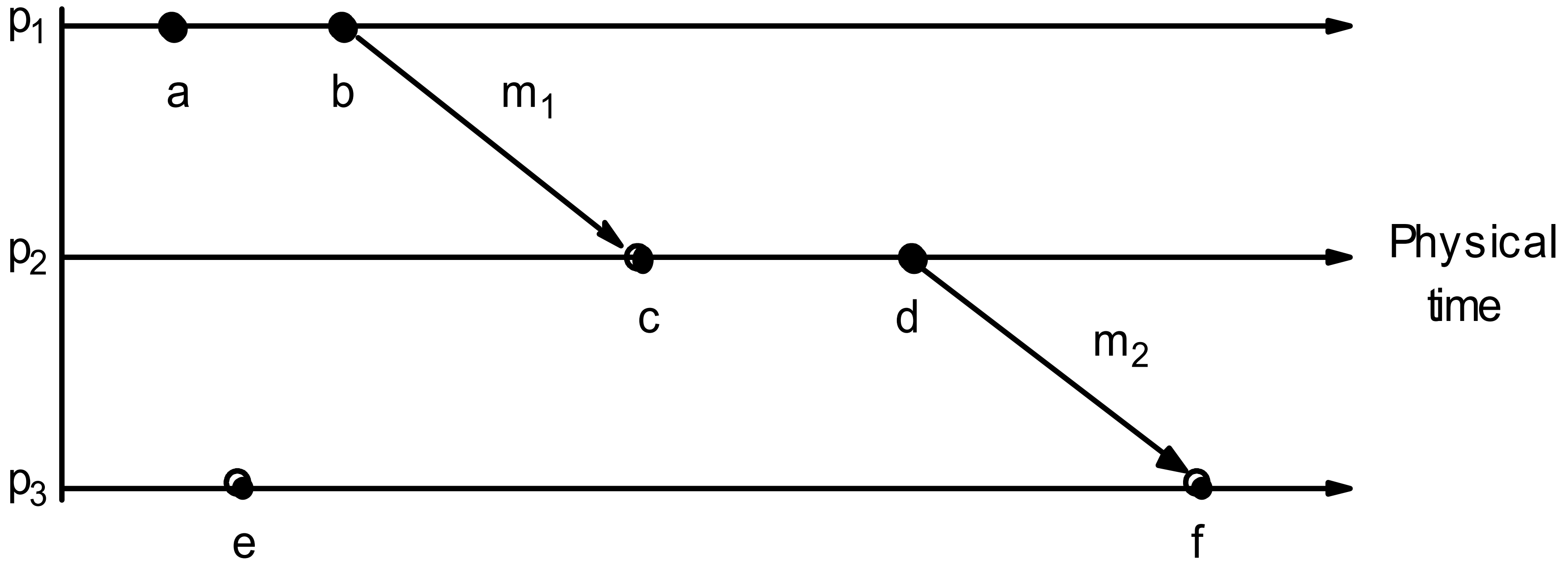
$$o_i - d_i/2 \le o \le o_i + d_i/2.$$

Then a Breakthrough…

- We cannot sync multiple clocks perfectly.
- Thus, if we want to order events happening in different processes, we cannot rely on physical clocks.
  - Remember the ticket reservation example?
- Then came logical time.
  - First proposed by Leslie Lamport in the 70s [2]
  - Based on causality of events
  - Defines relative time, not absolute time
- Critical observation: time (ordering) only matters if two or more processes interact, *i.e.*, send/receive messages.

Time, Clocks, and the Ordering of Events in a Distributed System [2] is required reading.

Events Occurring at Three Processes



Summary

- Time synchronization important for distributed systems
  - Cristian's algorithm
  - NTP
- Relative order of events is sufficient for many purposes
  - Lamport's logical clocks

Next time:

- More logical clocks

References

[1] *Probabilistic clock synchronization.* Flaviu Cristian. Distributed Computing Vol 3 No 3. September 1989.

[2] *Time, Clocks, and the Ordering of Events in a Distributed System.* Leslie Lamport. Communications of the ACM Vol 21 No 7. July 1978. **Required Reading.** https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Time-Clocks-and-the-Ordering-of-Events-in-a-Distributed-System.pdf

[3] *Internet Time Synchronization: The Network Time Protocol.* Dave L. Mills. RFC 1128. October 1989. https://www.rfc-editor.org/rfc/rfc1128.ps

[4] Textbook section 14.4. **Required Reading.**

Acknowledgements

- These slides are by Steve Ko, lightly modified by Ethan Blanton and used with permission.
- These slides contain material developed and copyrighted by Indranil Gupta at UIUC.