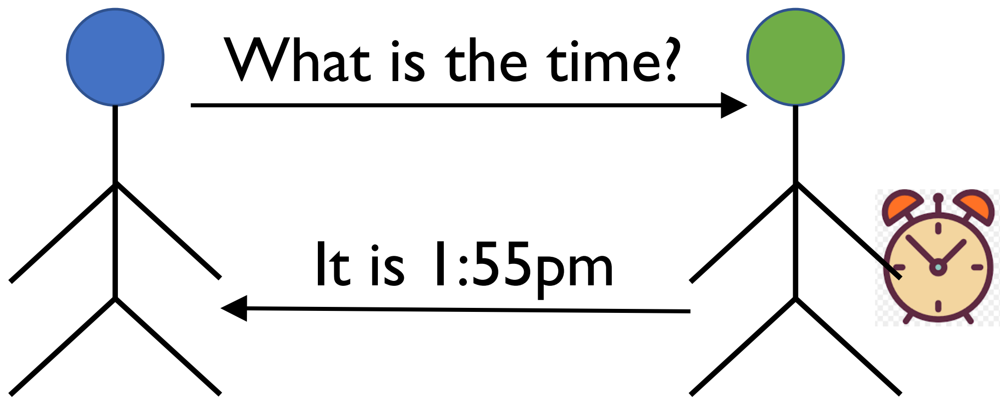
Distributed Systems

CS425/ECE428

Jan 25, 2023

Instructor: Radhika Mittal

While we wait…



Bluey does not own a clock, and wants to know the time.
He sends a message to Greeny asking the time, and Greeny sends a response as soon as he receives the request.
Bluey records that it took 6 minutes for him to receive Greeny's response after sending his request.

Given this information, what time should Bluey assume it actually is when he receives Greeny's message? Can he be totally accurate?

Logistics Related

- Make sure you are on CampusWire.
  - Email Manoj (gmk6) to get access if you are not already on it.

- All registered students have been added to Gradescope.
  - If you have registered late / plan on registering when a slot opens up, you can email Manoj (gmk6) to get added to Gradescope.
    - Please wait a week before doing so.

- MP0 released today
  - Will discuss in more details at the end of the class.

Recap: Failures

- Three types
  - omission, arbitrary, timing.

- Failure detection (detecting a crashed process):
  - Send periodic ping-acks or heartbeats.
  - Report crash if no response until a timeout.
  - Timeout can be precisely computed for synchronous systems and estimated for asynchronous.
  - Metrics: *completeness, accuracy, failure detection time, bandwidth.*
  - Failure detection for a system with multiple processes:
    - Centralized, ring, all-to-all
    - Trade-off between completeness and bandwidth usage.

Today's agenda

- Time and Clocks
  - Chapter 14.1-14.3
- Logical Clocks and Timestamps (if time)
  - Chapter 14.4

Why are clocks useful?

- How long did it take my search request to reach Google?
  - Requires my computer's clock to be *synchronized* with Google's server.

- Use timestamps to order events in a distributed system.
  - Requires the system clocks to be *synchronized* with one another.

- At what day and time did Alice transfer money to Bob?
  - Require *accurate* clocks (*synchronized* with a global authority).

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.

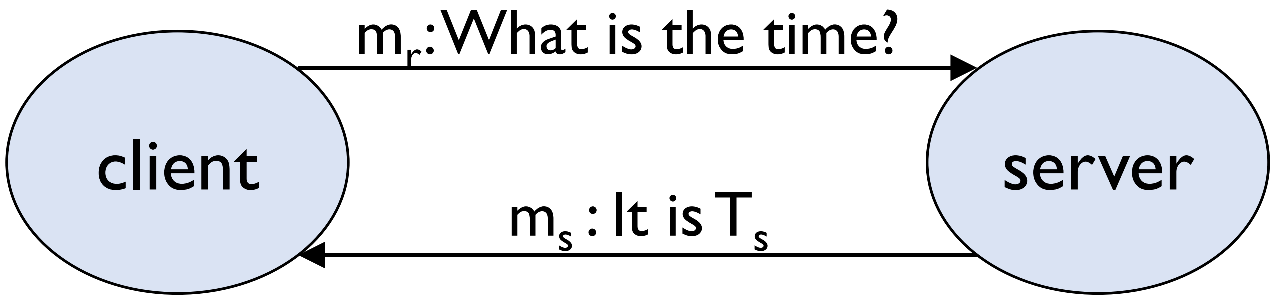
Synchronization Bound

- 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?*

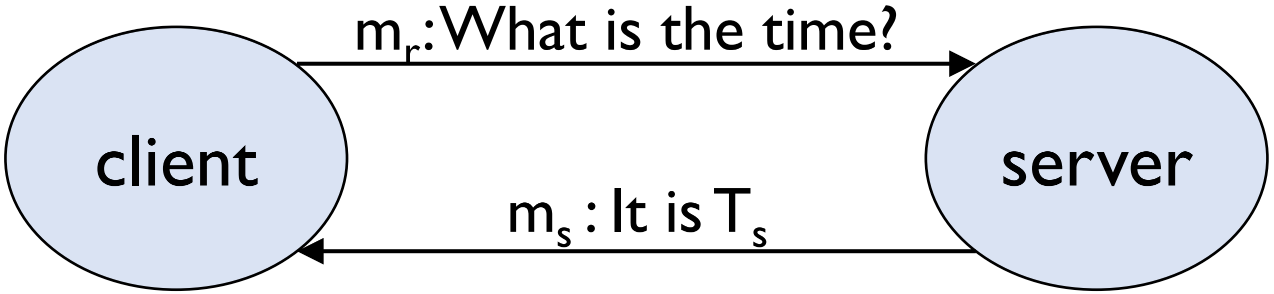
A: 2D. So the clocks are internally synchronized within a bound of 2D.

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.

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)*

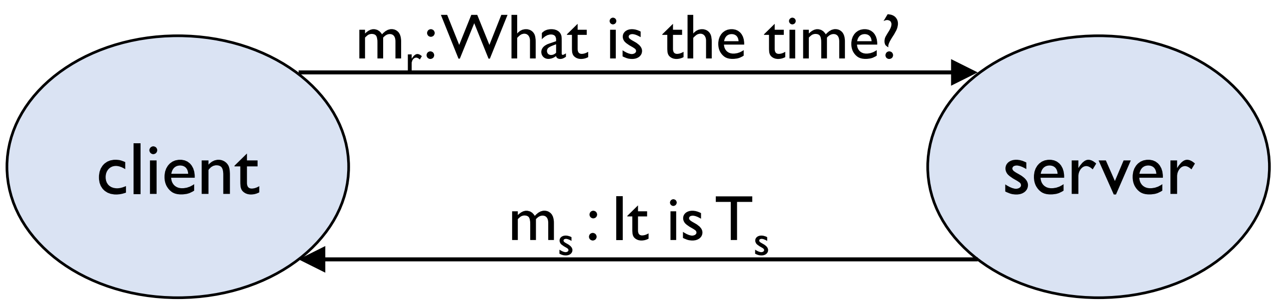
If $\mathbf{T_c = (T_s + (min + max)/2)}$, skew(client,server) $\leq$ *(max – min)/2*

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 (**T_{round}**).

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

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

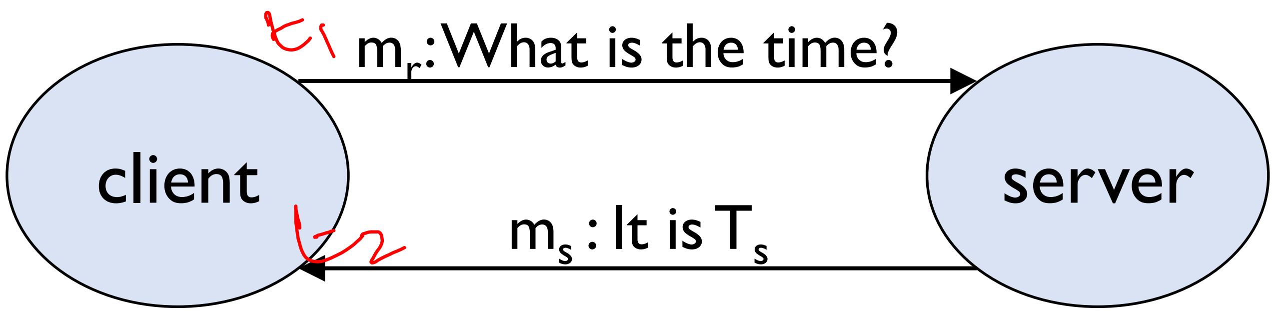
$\leq (T_{round} / 2)$

(*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* **Δ** *= $(T_{round}/2)$. How off can it be?*

Cristian Algorithm



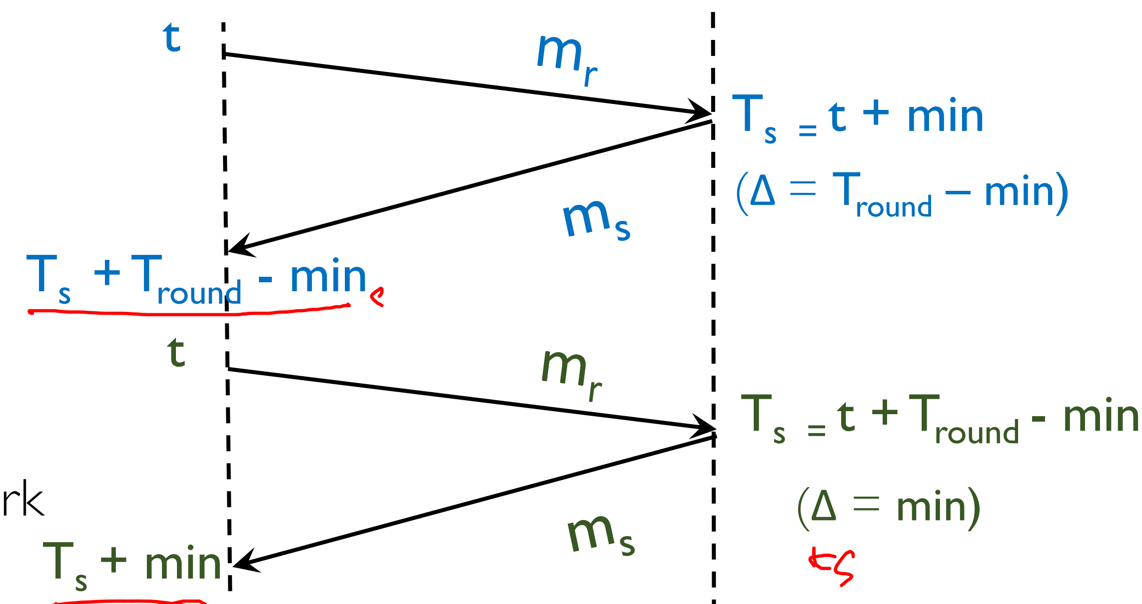
What time T_c should client adjust its local clock to after receiving m_s ?

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

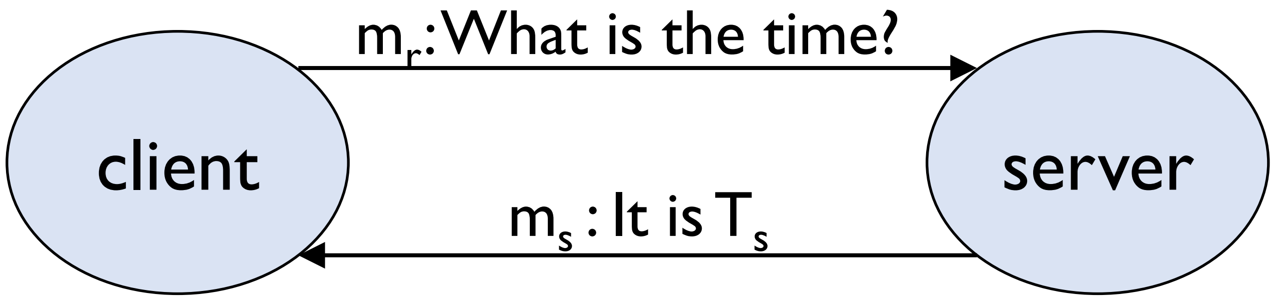
$$T_c = T_s + (T_{round} / 2)$$

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

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



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

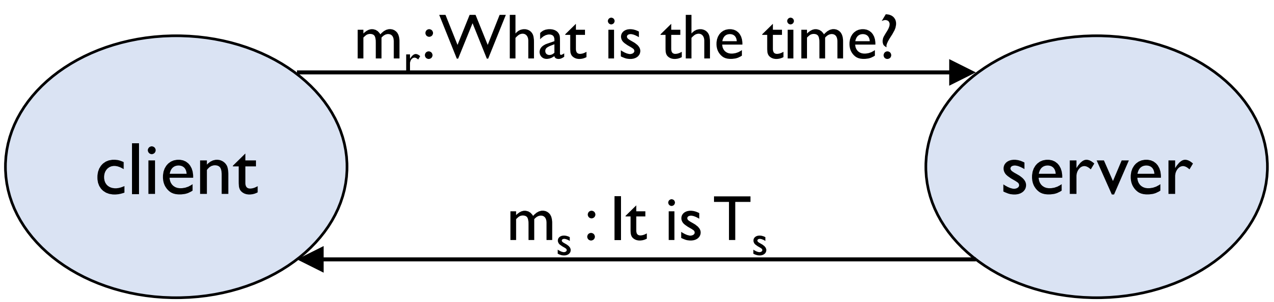
$\leq (T_{round} / 2)$

(*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}).

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

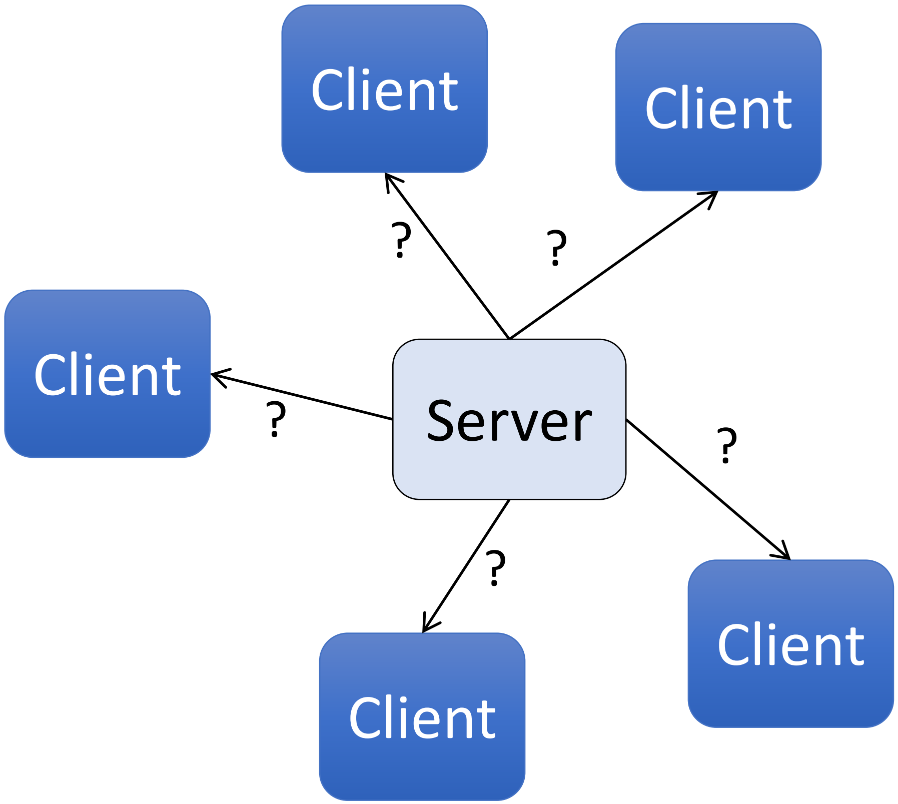
$$\text{skew} \leq (T_{round} / 2) - min$$
$$\leq (T_{round} / 2)$$

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

Cannot handle faulty time servers.

Berkeley Algorithm

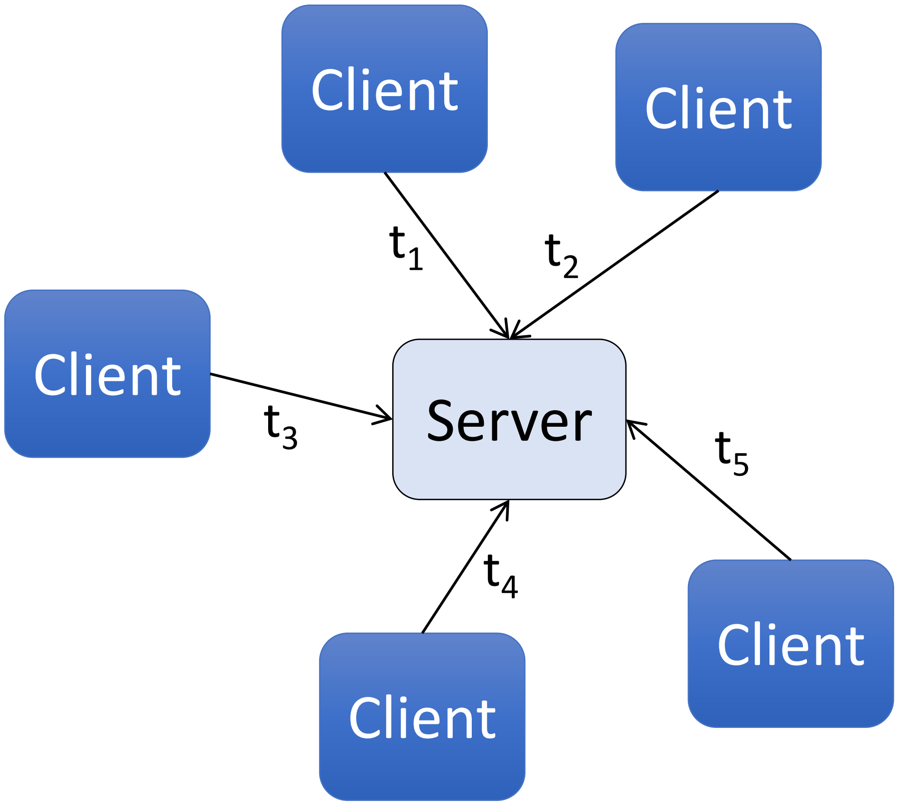
Only supports internal synchronization.



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

Berkeley Algorithm

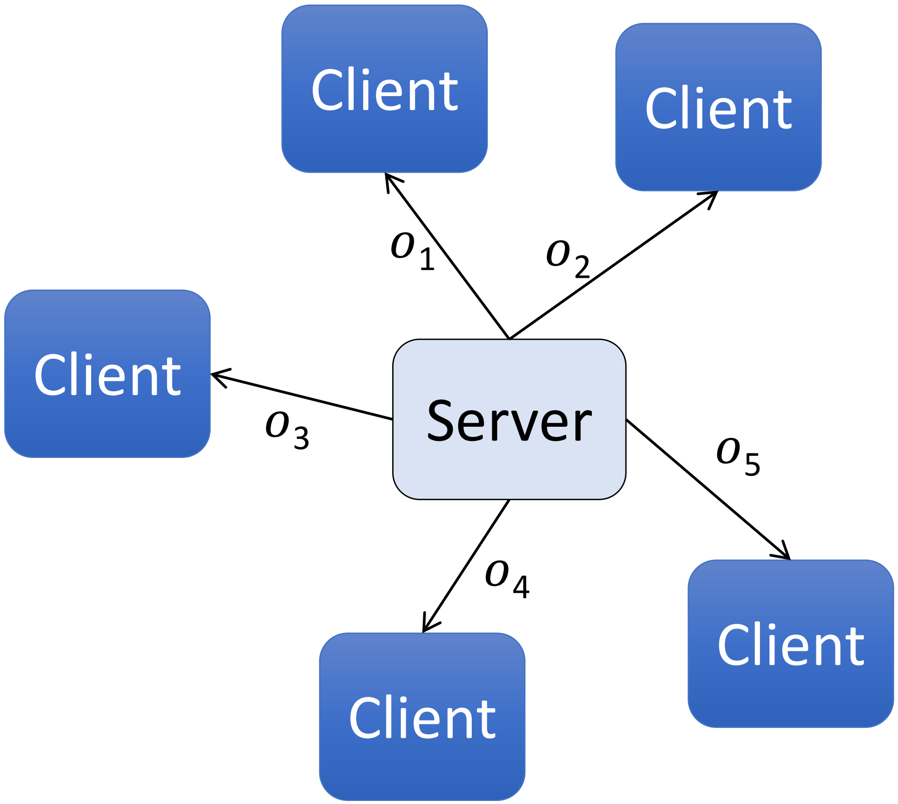
Only supports internal synchronization.



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

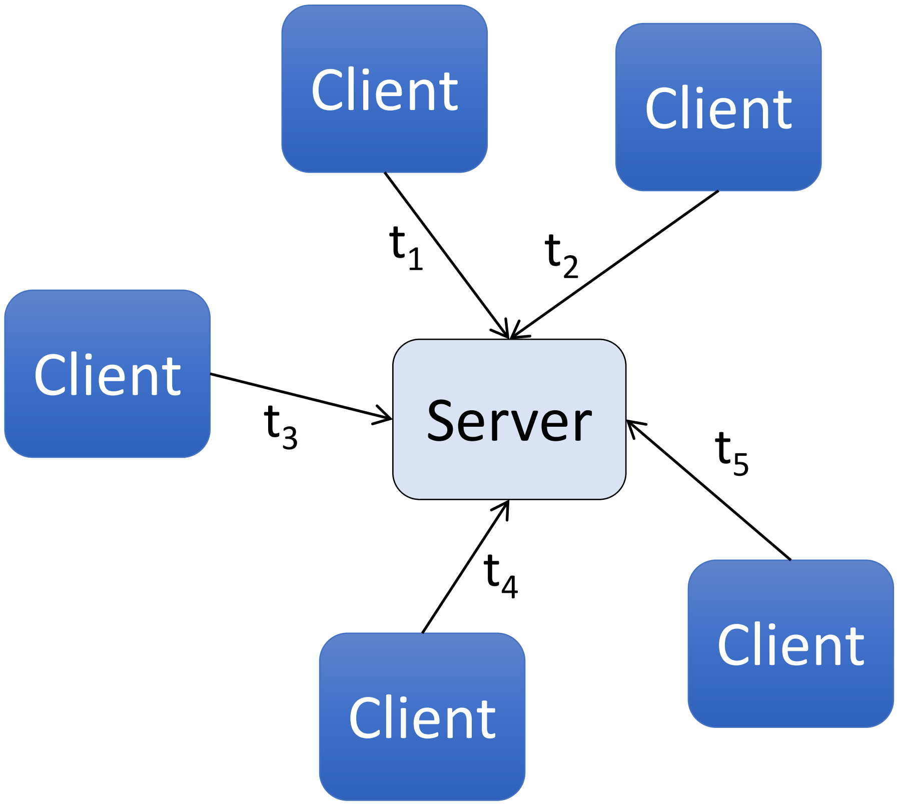
Only supports internal synchronization.



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.

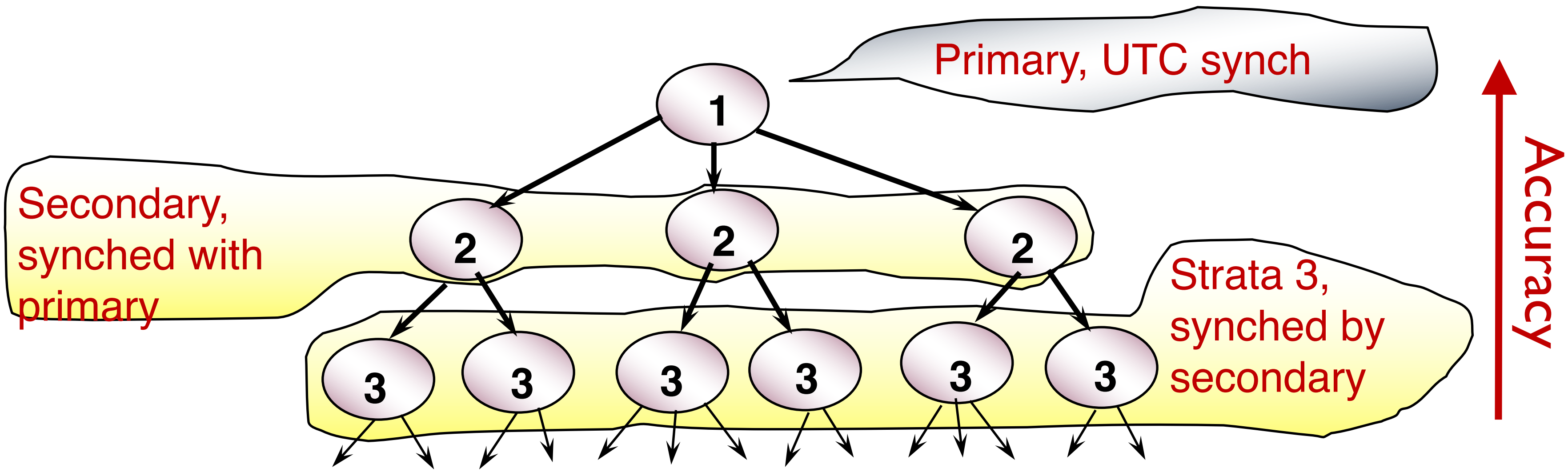


Handling faulty processes:
Only use timestamps within some threshold of each other.

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

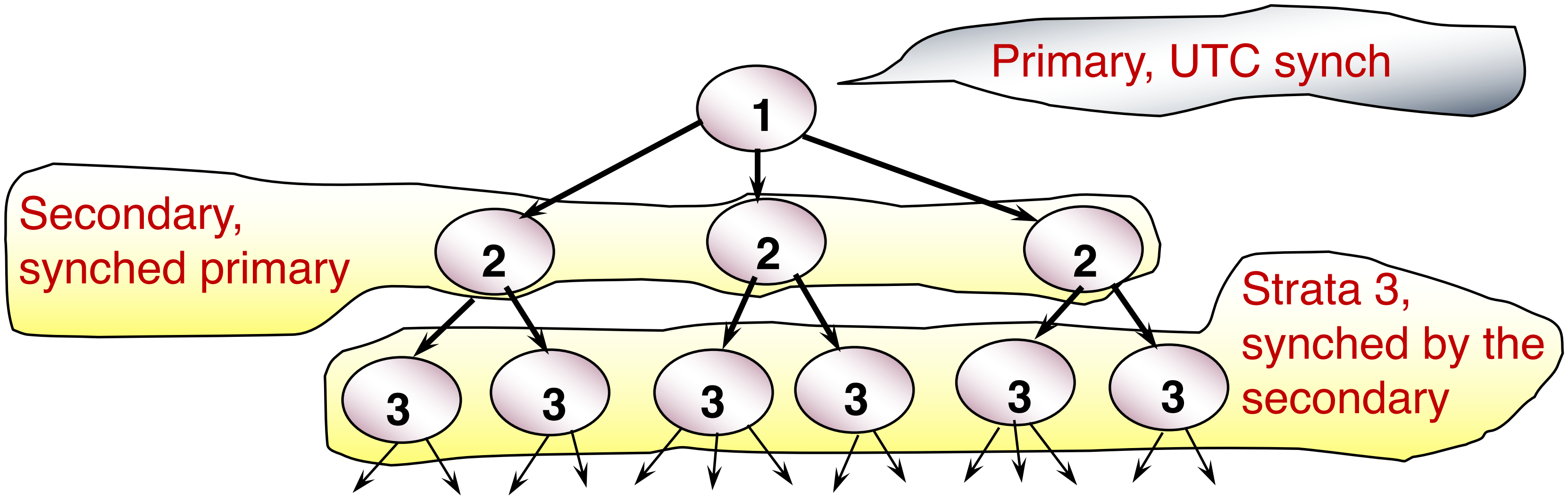
Network Time Protocol

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*.

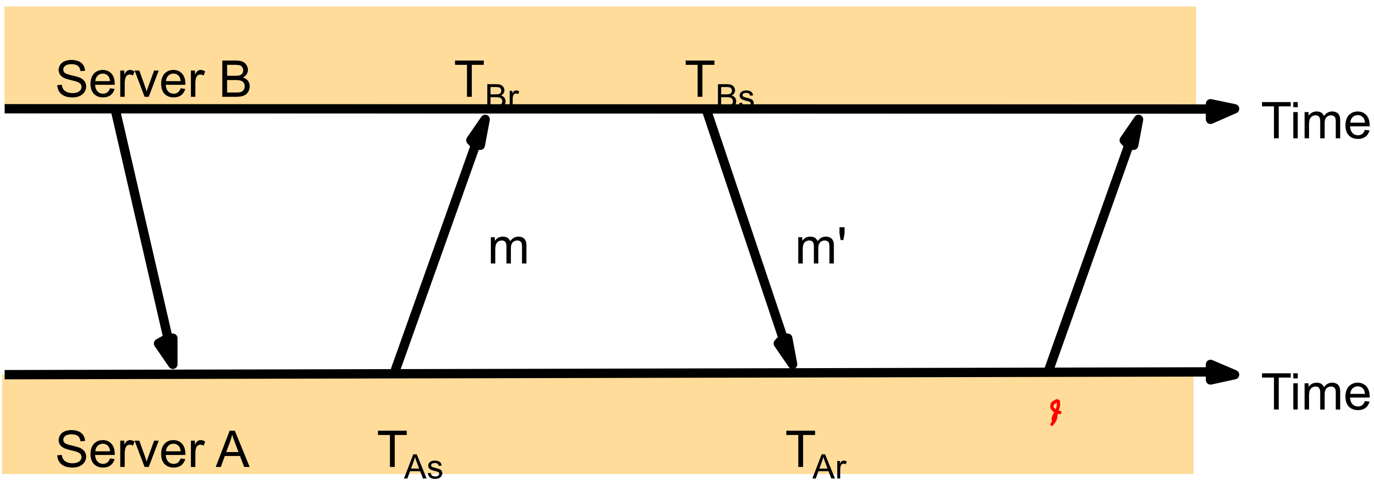
Network Time Protocol



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*.

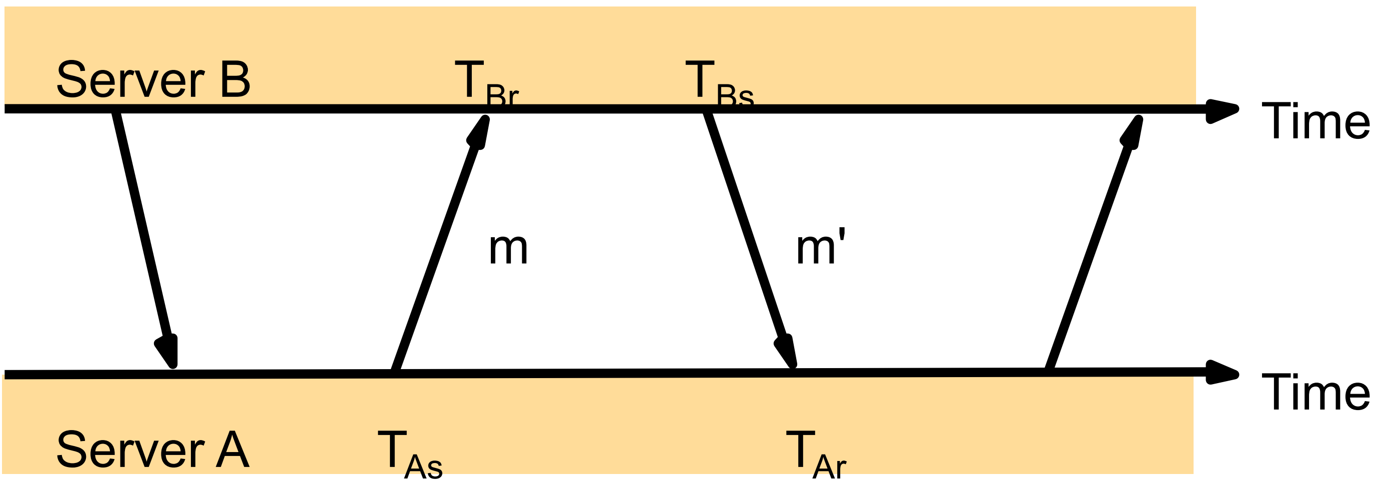
NTP Symmetric Mode



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

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

NTP Symmetric Mode

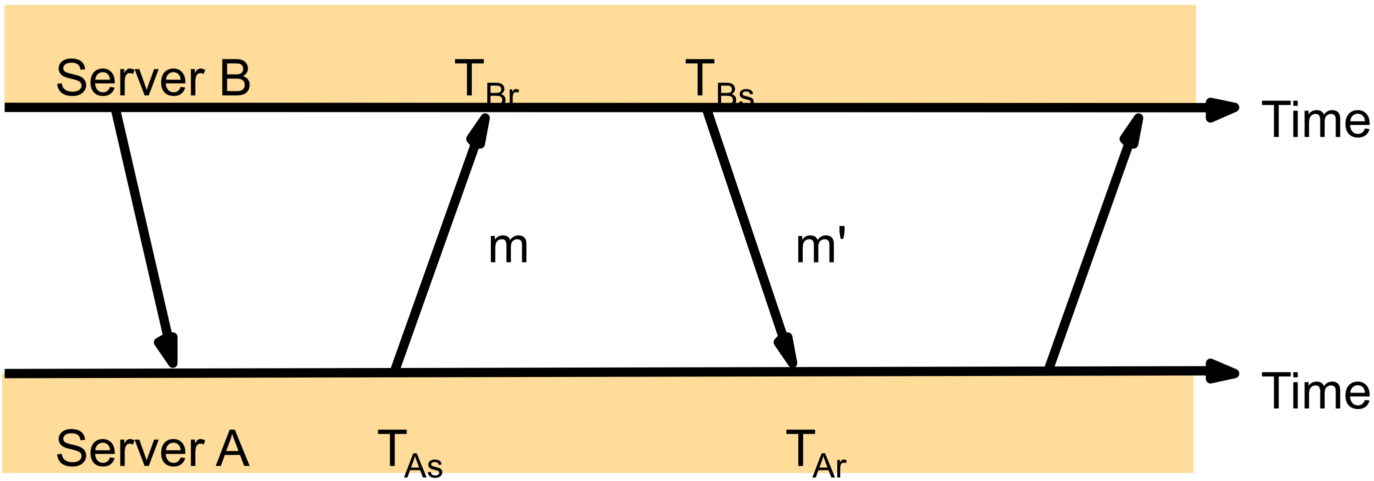


- t and t': actual transmission times for m and m'(unknown)
- o: true offset of clock at B relative to clock at A (unknown)
- o_i: estimate of actual offset between the two clocks
- d_i: estimate of accuracy of o_i ; total transmission times for m and m'. d_i=t+t'

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

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

NTP Symmetric Mode



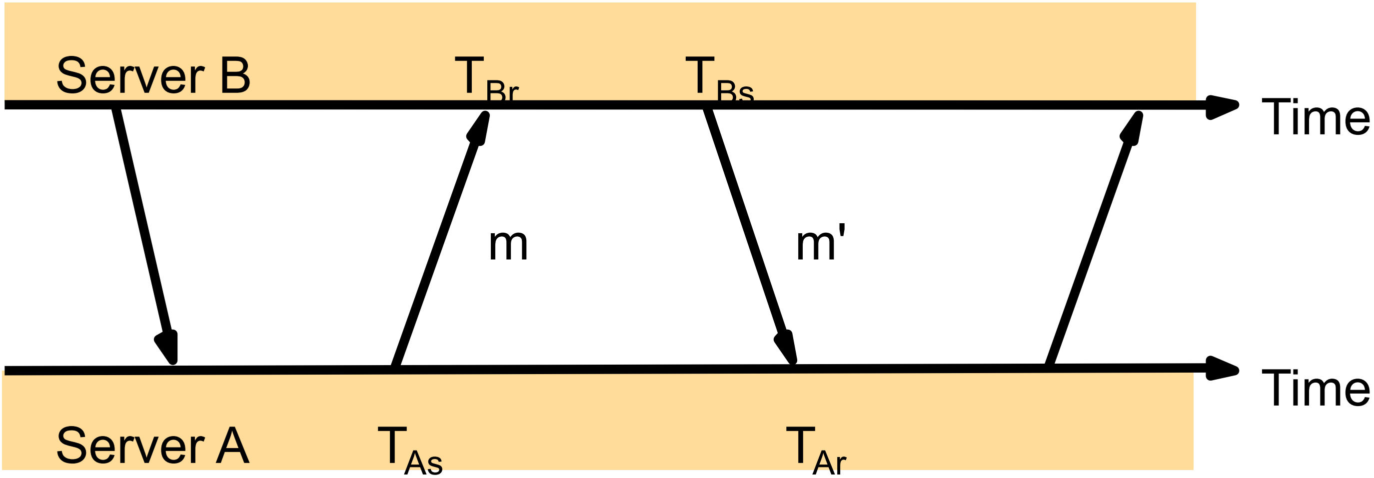
- t and t': actual transmission times for m and m'(unknown)
- o: <u>true</u> offset of clock at B relative to clock at A (unknown)
- o_i: <u>estimate</u> of actual offset between the two clocks
- d_i: estimate of <u>accuracy</u> of o_i ; total transmission times for m and m'. $d_i = t + t'$

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

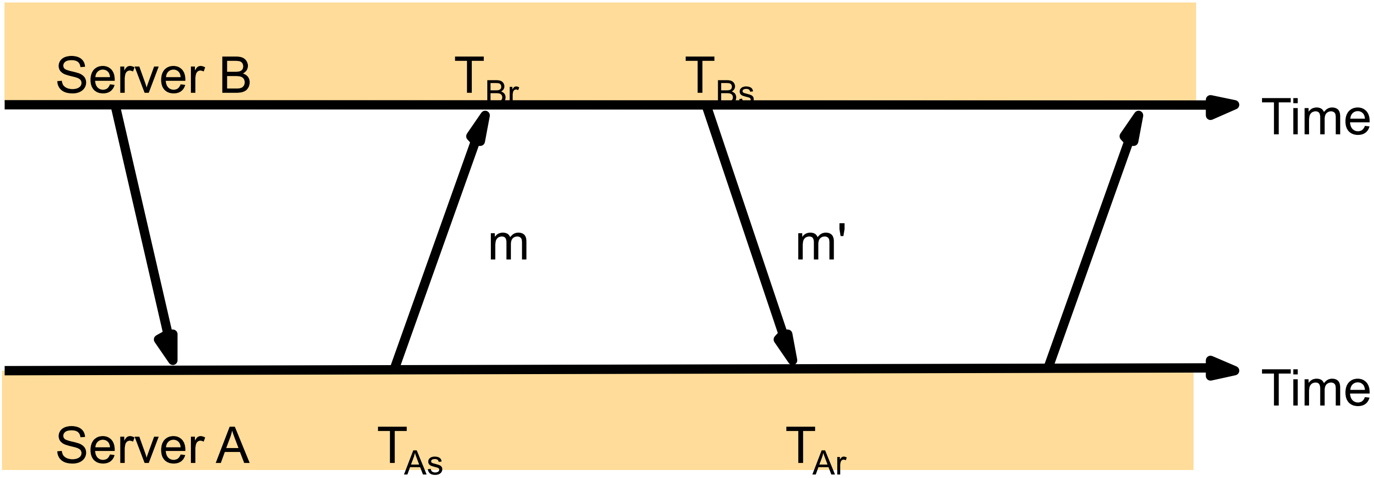
$$o = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) + (t' - t))/2$$
$$\mathbf{o_i = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}))/2}$$
$$o = o_i + (t' - t)/2$$
$$\mathbf{d_i = t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs})}$$

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

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.

MP0: Event Logging

- https://courses.grainger.illinois.edu/cs425/sp2023/mps/mp0.html
- Lead TA: Manoj Girija

- Task:
  - Collect events from distributed nodes.
  - Aggregate them into a single log at a centralized logger.

- Objective:
  - Familiarize yourself with the cluster development environment.
  - Practice distributed experiments and performance analysis.
  - Build infrastructure that might be useful in future MPs.

MP0: Event Logging

- We provide you with a script that generates logs

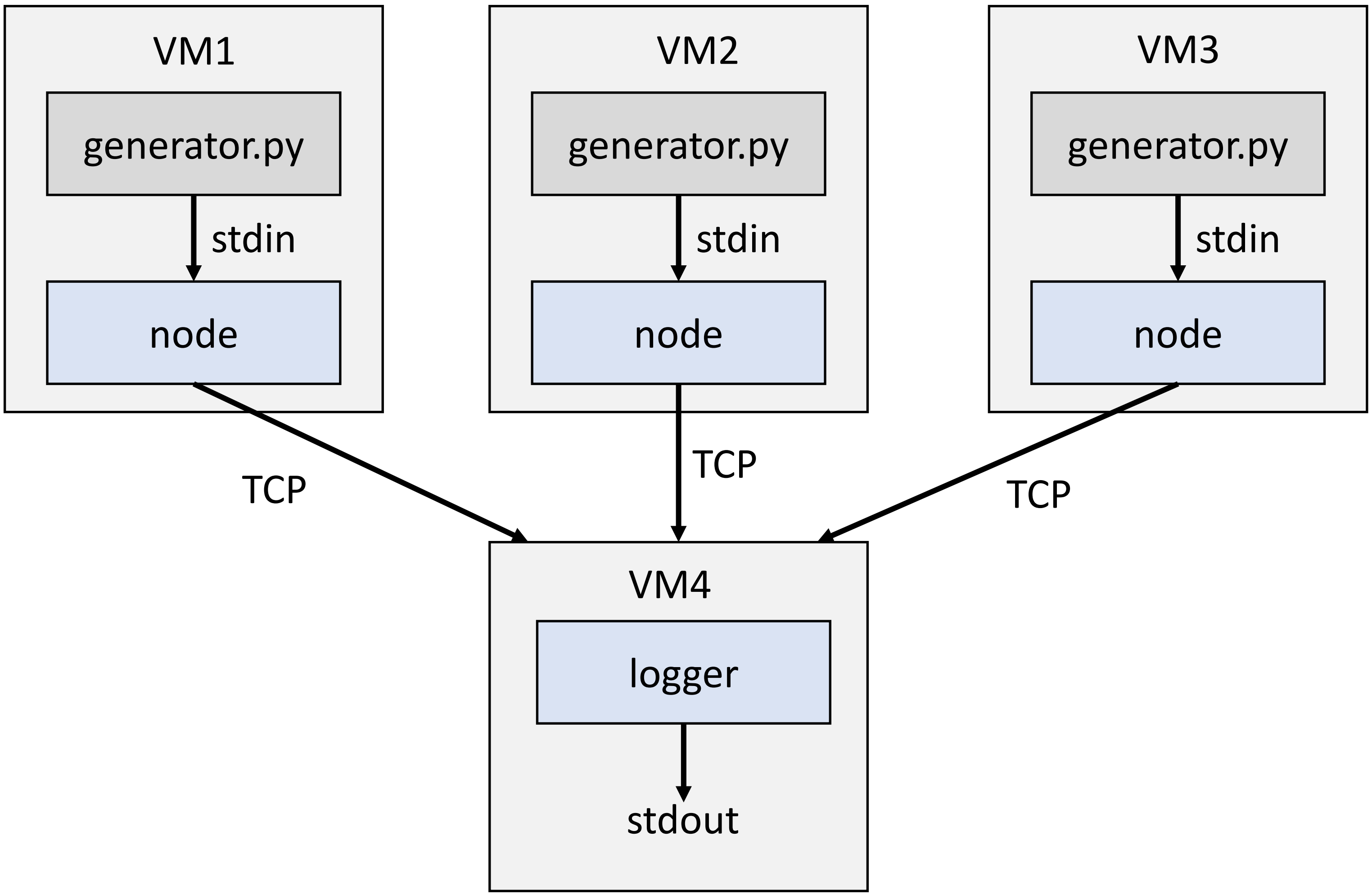
generator.py

Timestamp

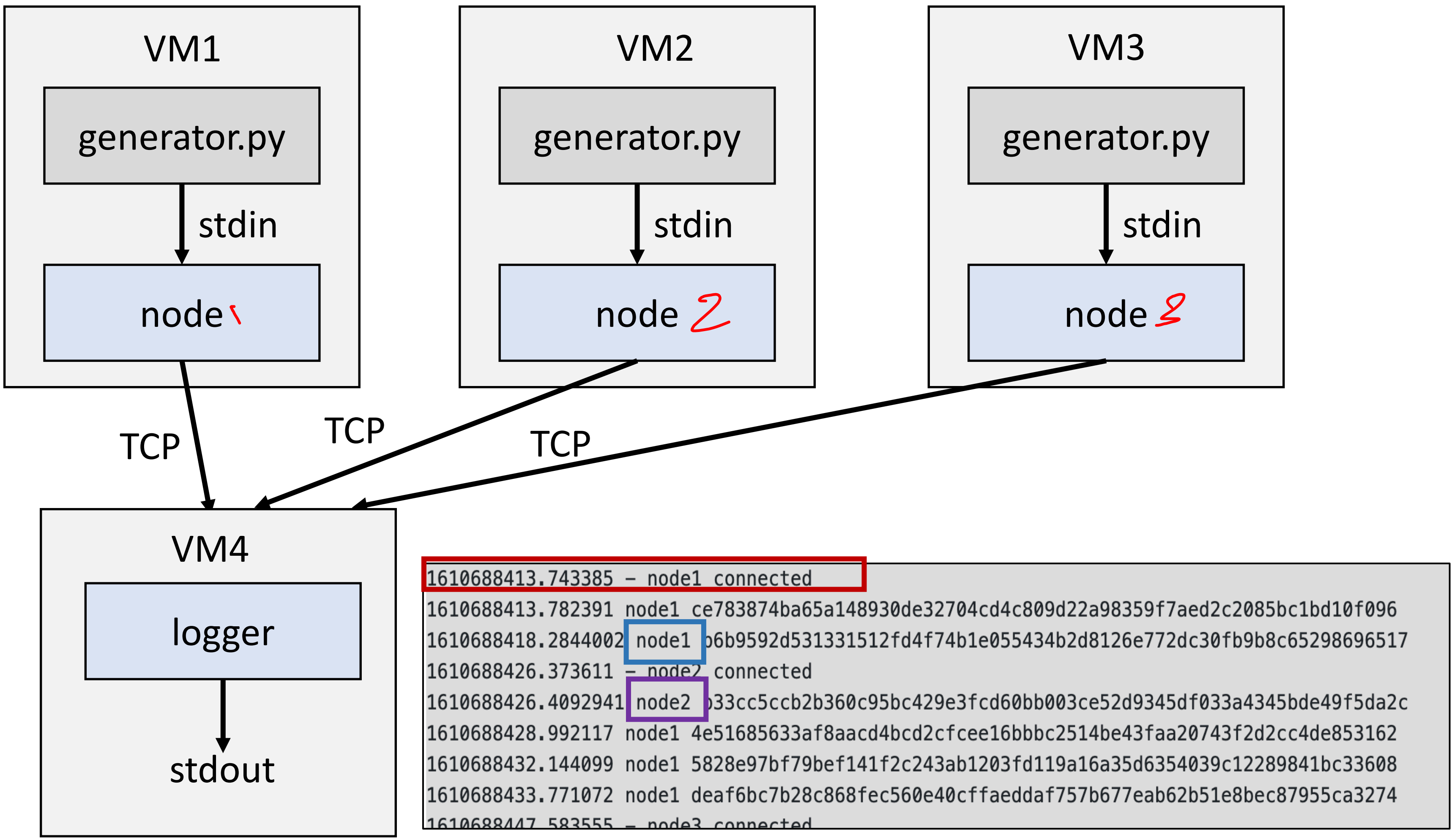
Event name (random)

```
% python3 generator.py 0.1
1610688413.782391 ce783874ba65a148930de32704cd4c809d22a98359f7aed2c2085bc1bd10f096
1610688418.2844002 b6b9592d531331512fd4f74b1e055434b2d8126e772dc30fb9b8c65298696517
1610688428.992117 4e51685633af8aacd4bcd2cfcee16bbbc2514be43faa20743f2d2cc4de853162
1610688432.144099 5828e97bf79bef141f2c243ab1203fd119a16a35d6354039c12289841bc33608
1610688433.771072 deaf6bc7b28c868fec560e40cffaeddaf757b677eab62b51e8bec87955ca3274
1610688449.1301062 ca6e5225e2ea02c1174701dd0320954fbfffb51dbcd9d15717e11d7e40556efb
1610688455.484428 ed4b1eb8a7bd980a1f0da41f5d6513e919e2bf201ba9ec9f9c05201bd777af94
1610688455.813278 3b014179e1cc1d2cc9cf553441492ad4f054634d2f0f0b66d0185c60fc4355da
1610688463.543133 8110f0cc37404a10989bfe14ae83224a73e642bb676ded625b08ed7d3e439706
```

MP0: Event Logging



MP0: Event Logging



MP0: Event Logging

- Run two experiments
  - 3 nodes, 2 events/s each
  - 8 nodes, 5 events/s each

- Collect graphs of two metrics:
  - Delay between event generation at the node and it appearing in the centralized log.
  - Amount of bandwidth used by the central logger.
  - Need to add instrumentation to your code to track these metrics.

MP0: Event Logging

- Due on Feb 8, 11:59pm
  - Late policy: Can use part of your 168hours of grace period accounted per student over the entire semester.

- Carried out in groups of 1-2
  - Same expectations regardless of group size.
  - Fill out form on CampusWire to get access to cluster.
    - Getting cluster access may take some time.
    - But you can start coding now!

- Can use any language.
  - Supported languages are C/C++, Go, Java, Python.