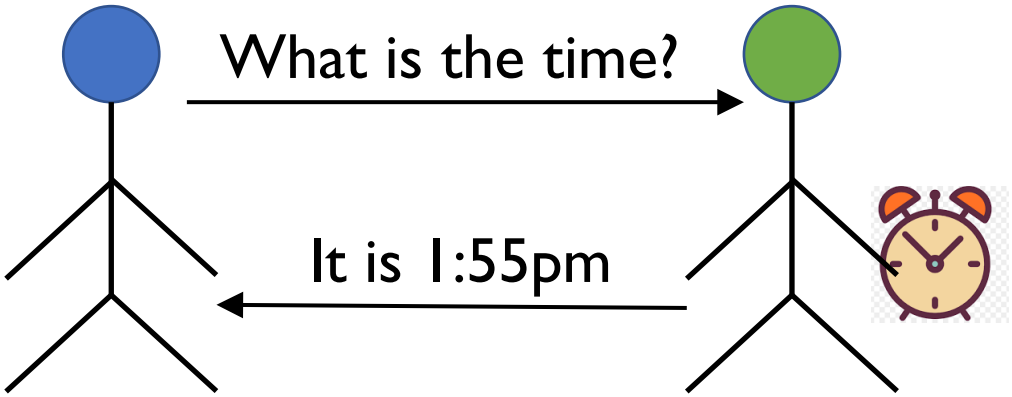
Distributed Systems

CS425/ECE428

Feb 3 2021

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

- VM clusters will be assigned by the end of the day today.
- HW1 will be released this Friday.

Today's agenda

- Failure Detection
  - Chapter 15.1
- Time and Clocks
  - Chapter 14.1-14.3

Recap

- Synchronous vs asynchronous systems.
- Failure detection via ping-ack and heartbeats.
  - Complete and accurate in synchronous system.
  - Impossible to achieve completeness and accuracy in asynchronous systems.
    - Setting a conservative timeout gives completeness but not accuracy.

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for previous ping from p to reach q)
  - Heartbeat: $T + \Delta_2 + \Delta$ (where Δ is time taken for last heartbeat from q to reach p)
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for previous ping from p to reach q)
  - Heartbeat: $T + \Delta_2 + \Delta$ (where Δ is time taken for last heartbeat from q to reach p)
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

Effect of decreasing T?

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for previous ping from p to reach q)
  - Heartbeat: $T + \Delta_2 + \Delta$ (where Δ is time taken for last heartbeat from q to reach p)
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

Decreasing T decreases failure detection time, but increases bandwidth usage.

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for previous ping from p to reach q)
  - Heartbeat: $T + \Delta_2 + \Delta$ (where Δ is time taken for last heartbeat from q to reach p)
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

Effect of increasing Δ_1 or Δ_2?

Metrics for failure detection

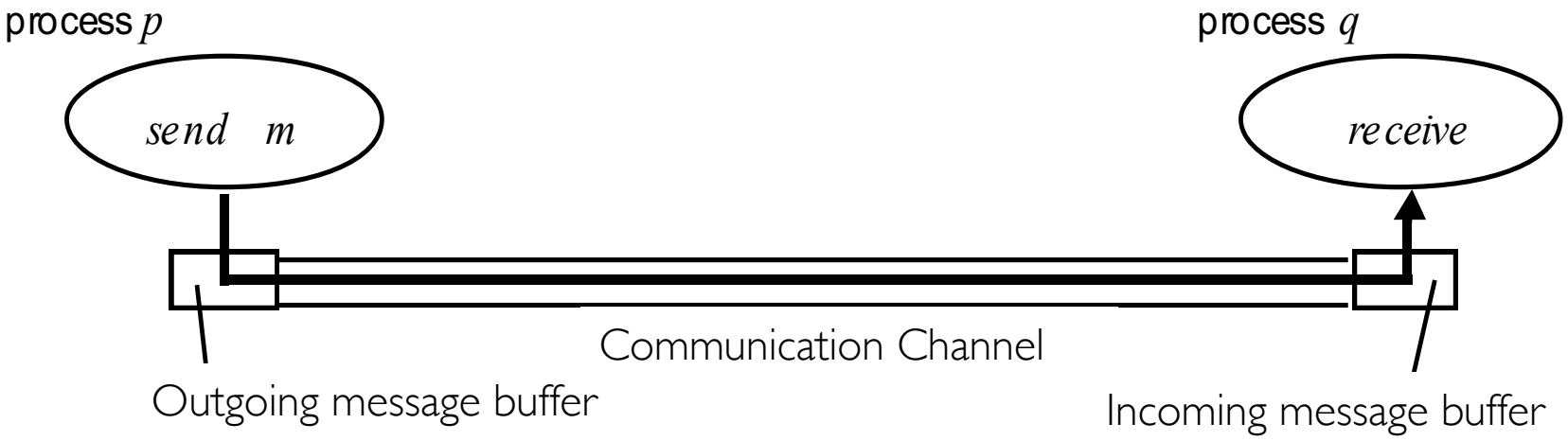
- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for previous ping from p to reach q)
  - Heartbeat: $T + \Delta_2 + \Delta$ (where Δ is time taken for last heartbeat from q to reach p)
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

Increasing Δ_1 or Δ_2 increases accuracy but also increases failure detection time.

Types of failure

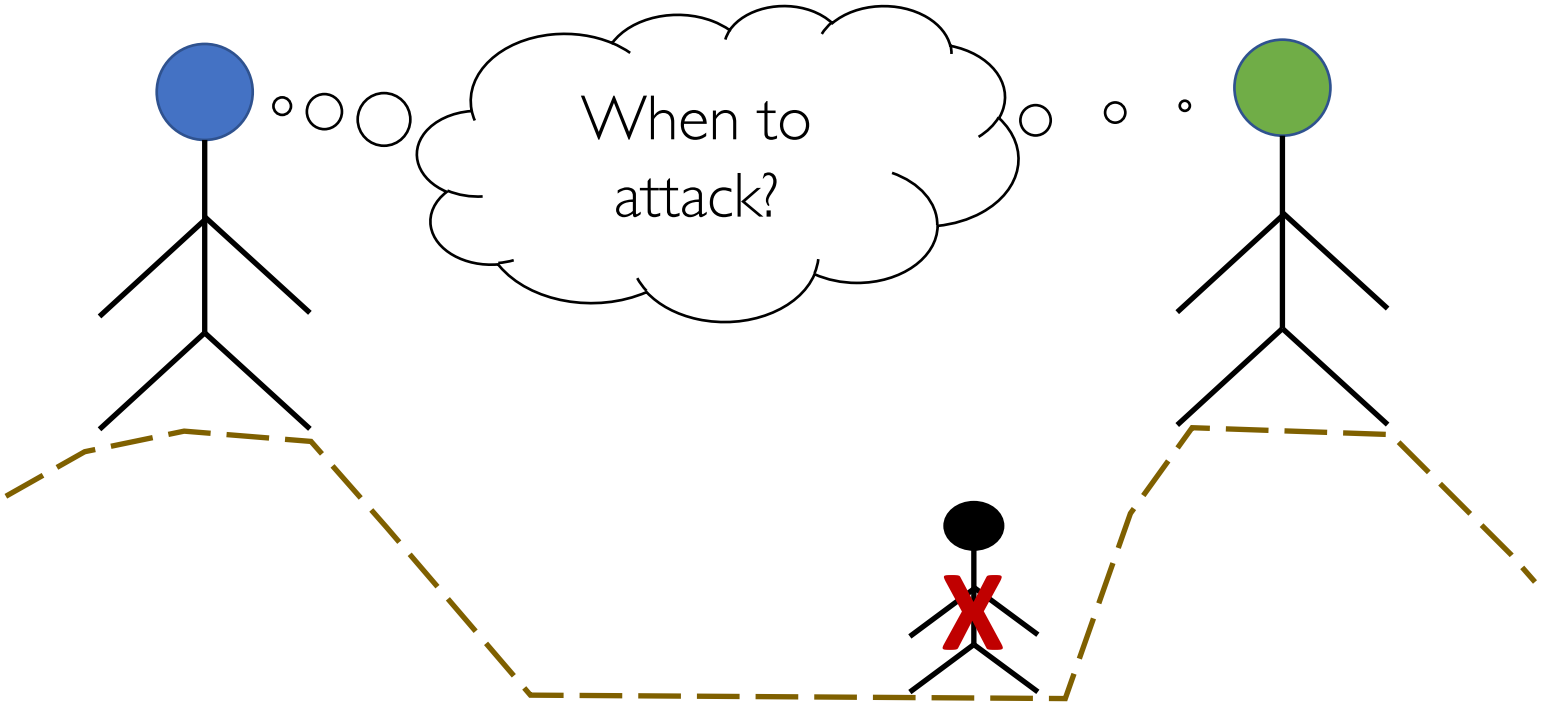
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.
  - **Fail-stop**: if other processes can certainly detect the crash.
  - **Communication omission**: a message sent by process was not received by another.

Communication Omission

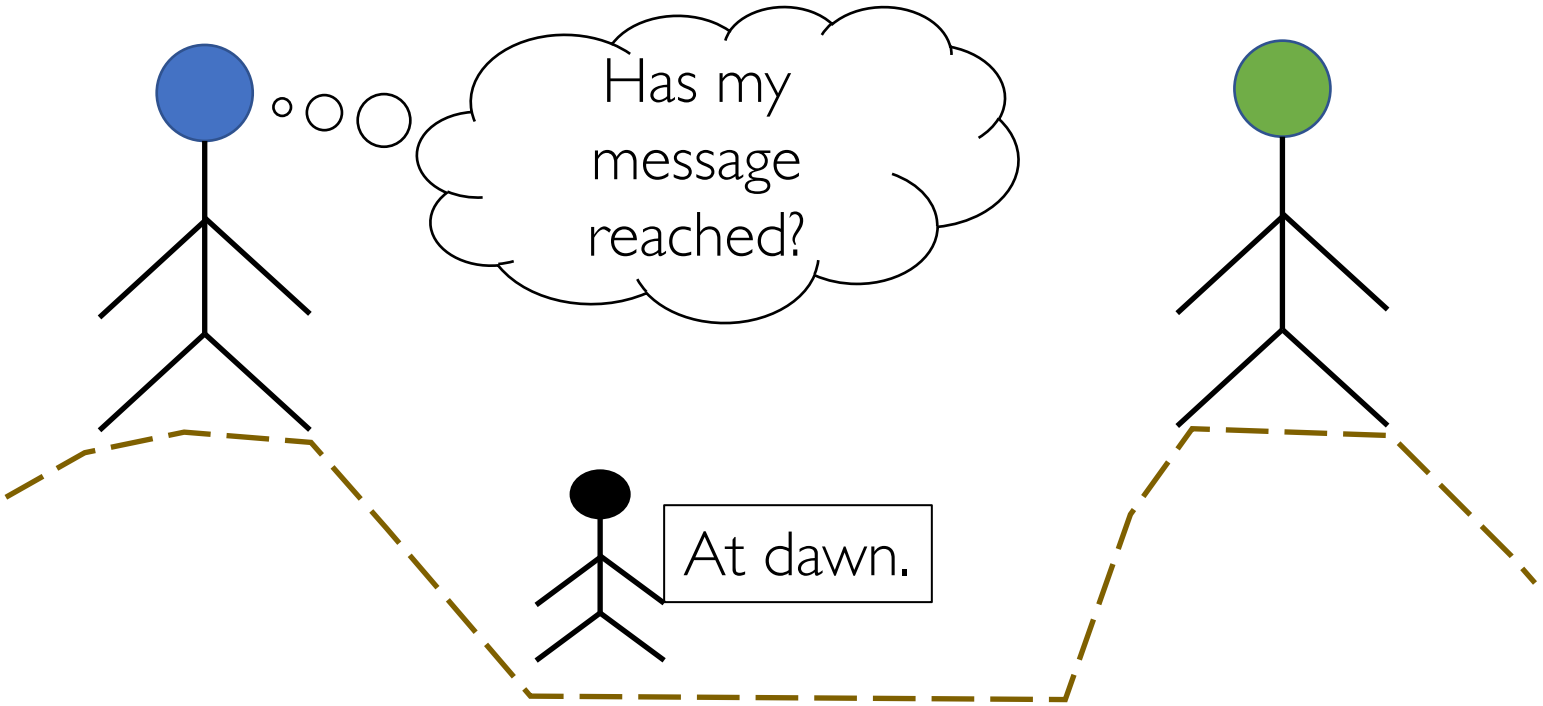


- Channel Omission: omitted by channel
- Send omission: process completes 'send' operation, but message does not reach its outgoing message buffer.
- Receive omission: message reaches the incoming message buffer, but not received by the process.

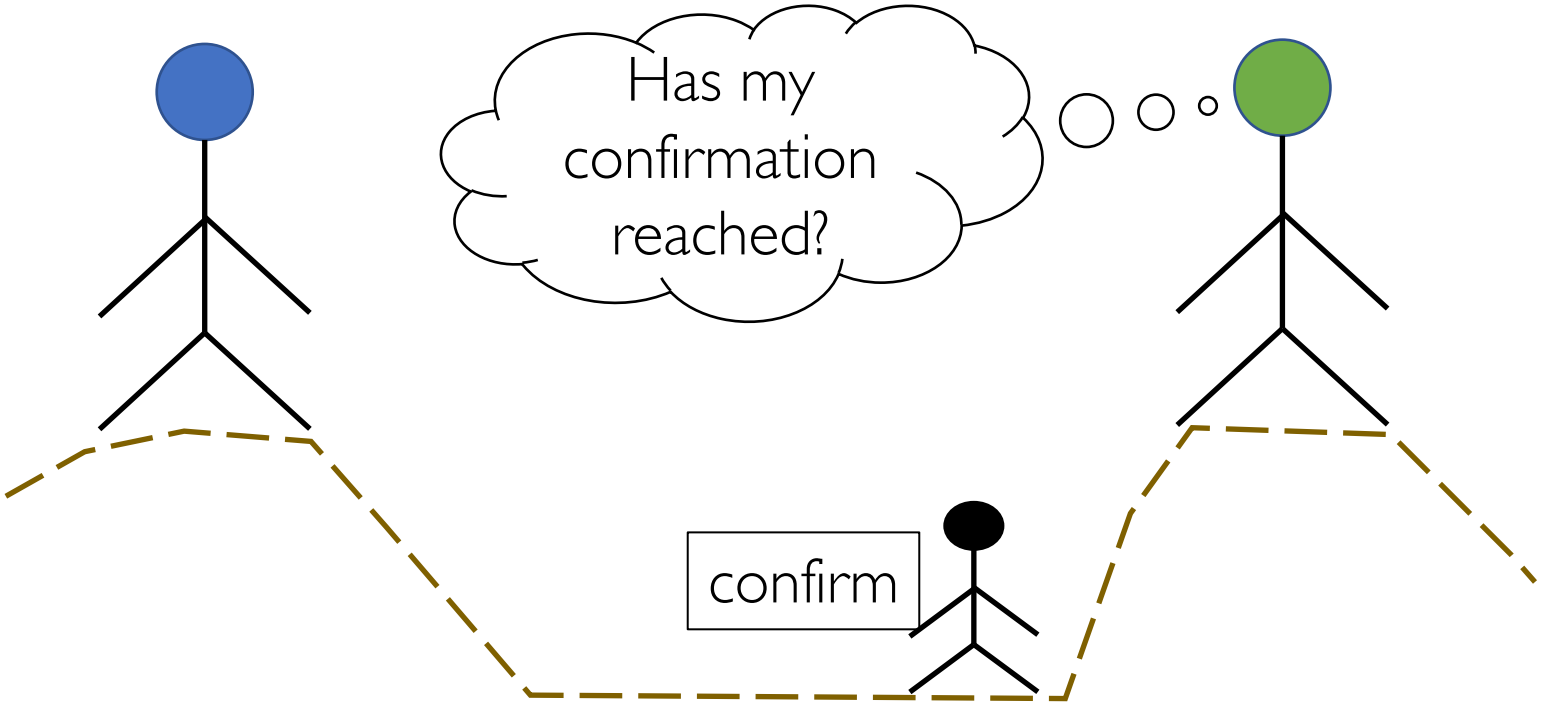
Two Generals Problem



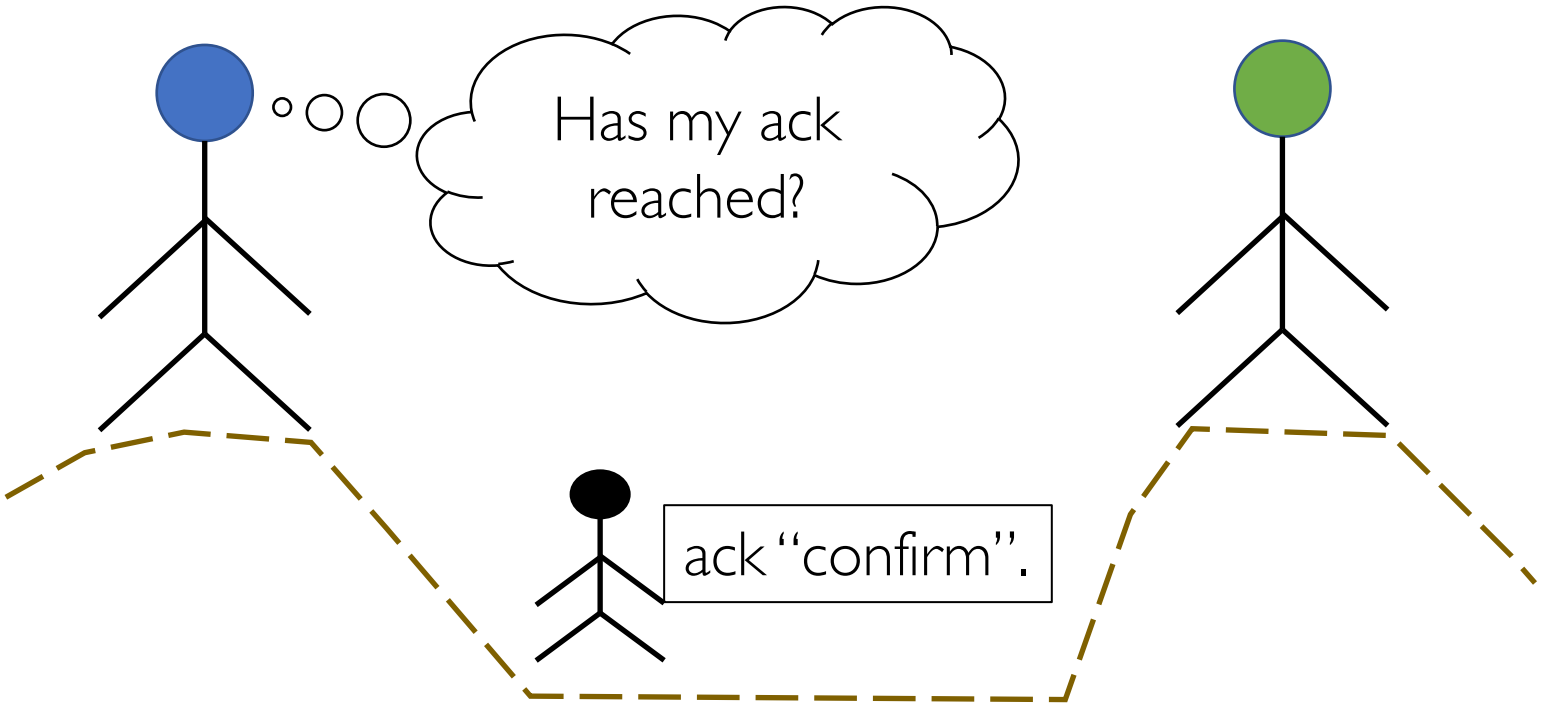
Two Generals Problem



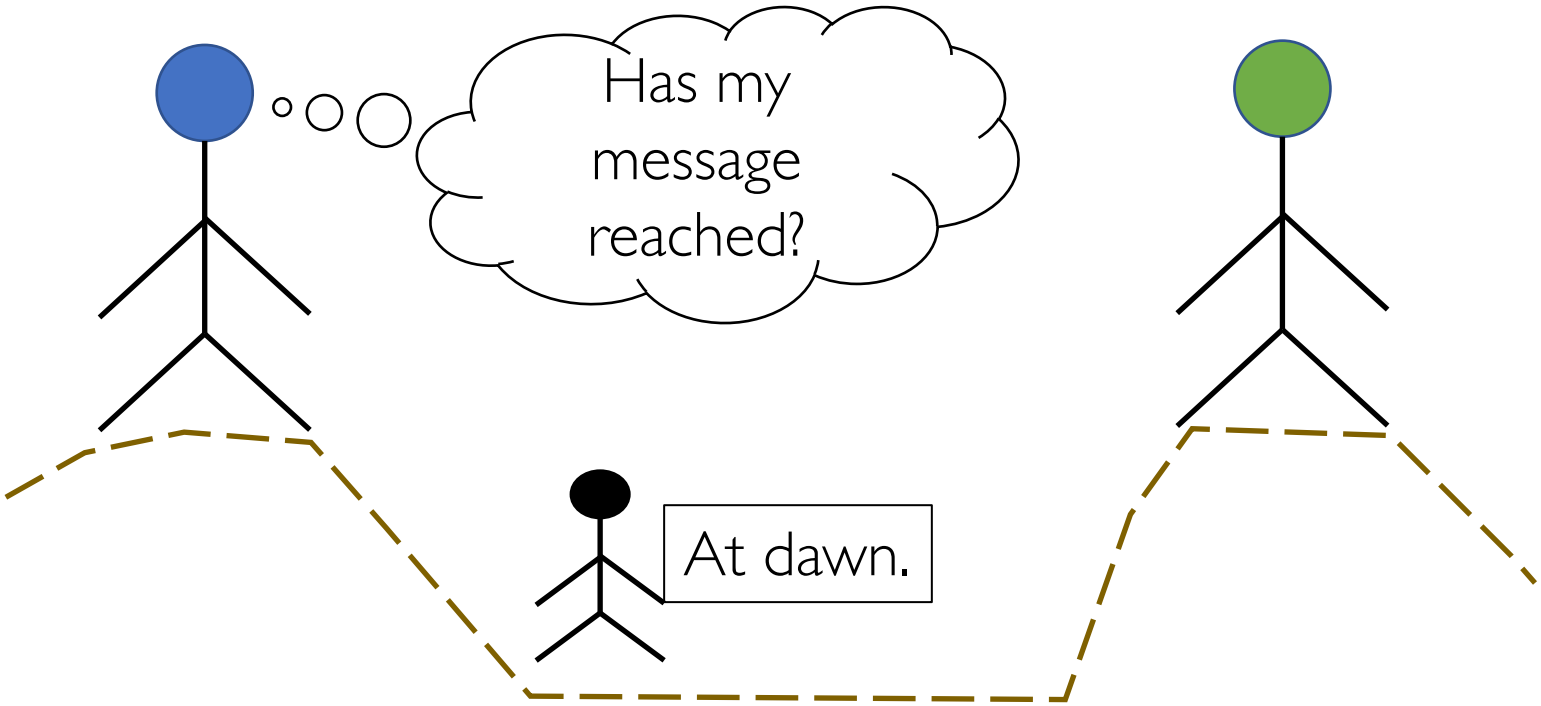
Two Generals Problem



Two Generals Problem

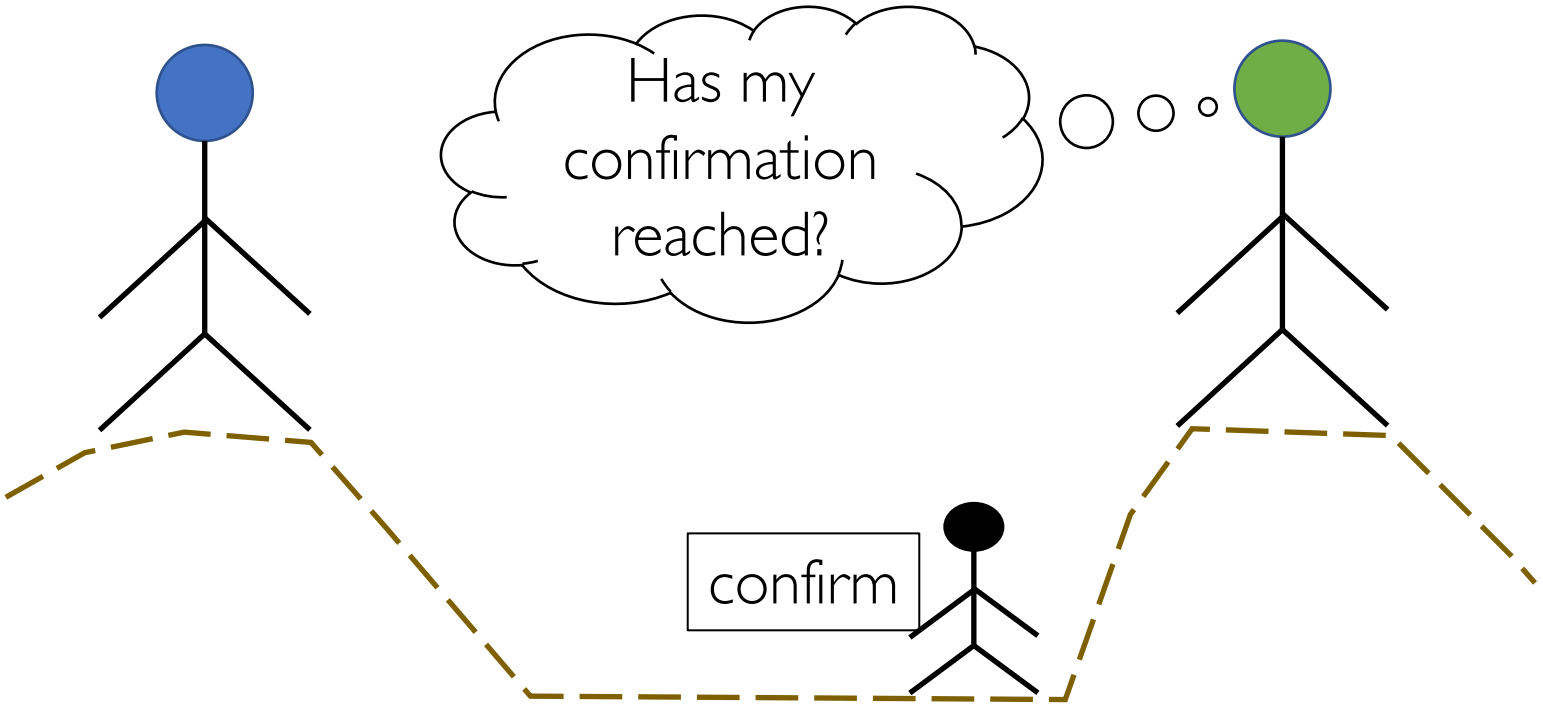


Two Generals Problem



Keep sending the message until confirmation arrives.

Two Generals Problem



Assume confirmation has reached in the absence of a repeated message.

Still no guarantees! But may be good enough in practice.

Types of failure

- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.
  - **Fail-stop**: if other processes can detect that the process has crashed.
  - **Communication omission**: a message sent by process was not received by another.

Message drops (or omissions) can be mitigated by network protocols.

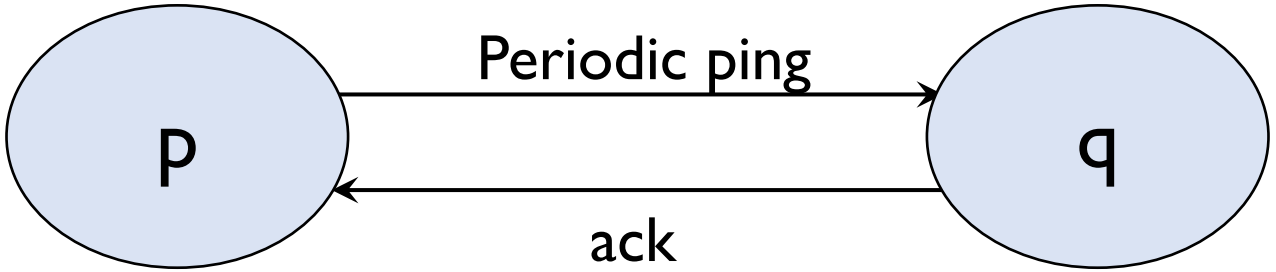
Types of failure

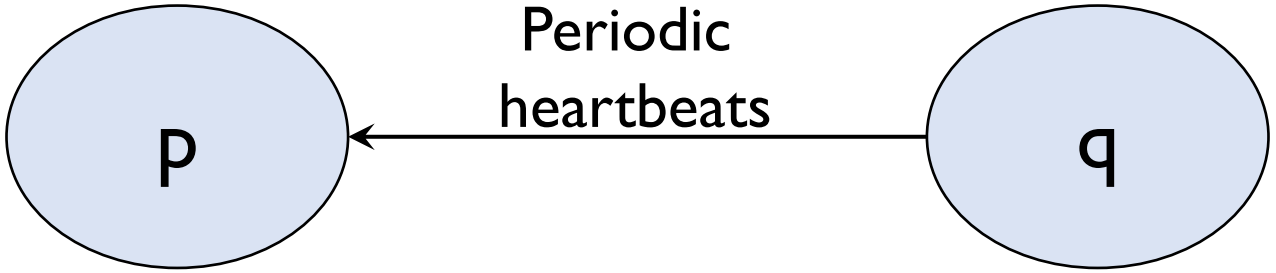
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do, e.g. process crash and message drops.

- **Arbitrary (Byzantine) Failures:** any type of error, e.g. a process executing incorrectly, sending a wrong message, etc.

- **Timing Failures:** Timing guarantees are not met.
  - Applicable only in synchronous systems.

How to detect a crashed process?





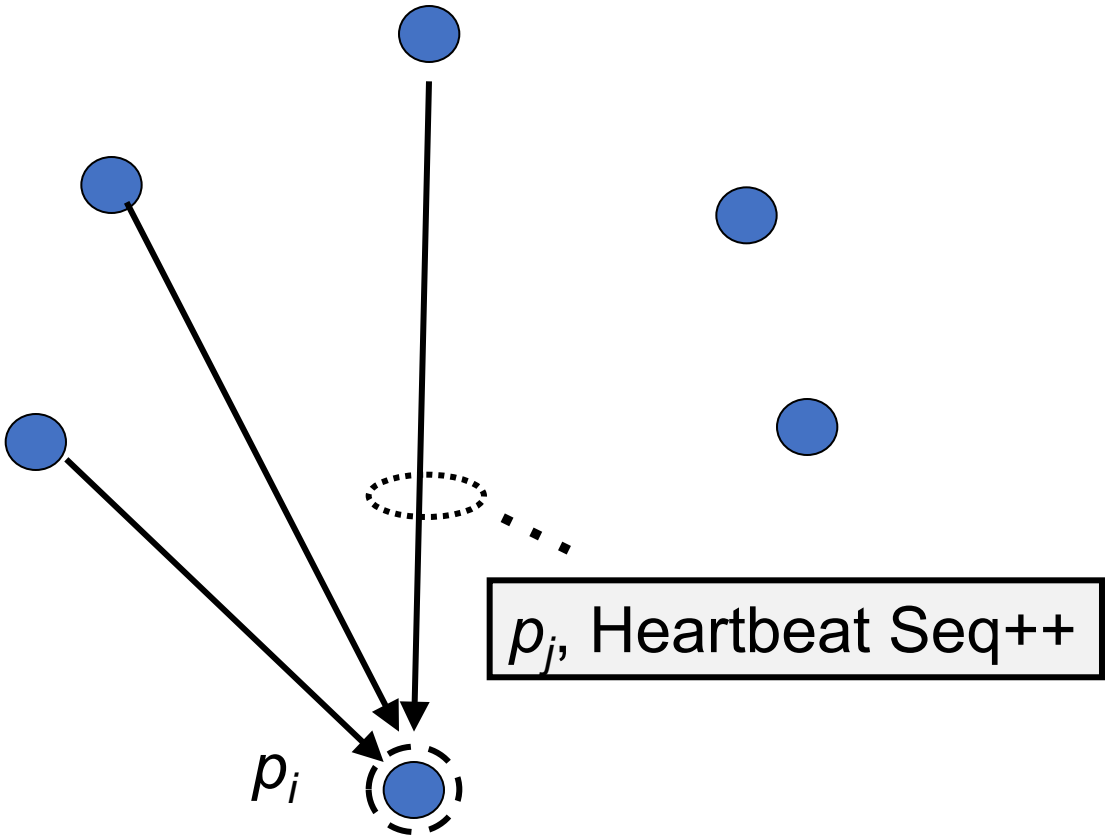
Extending heartbeats

- Looked at detecting failure between two processes.
- How do we extend to a system with multiple processes?

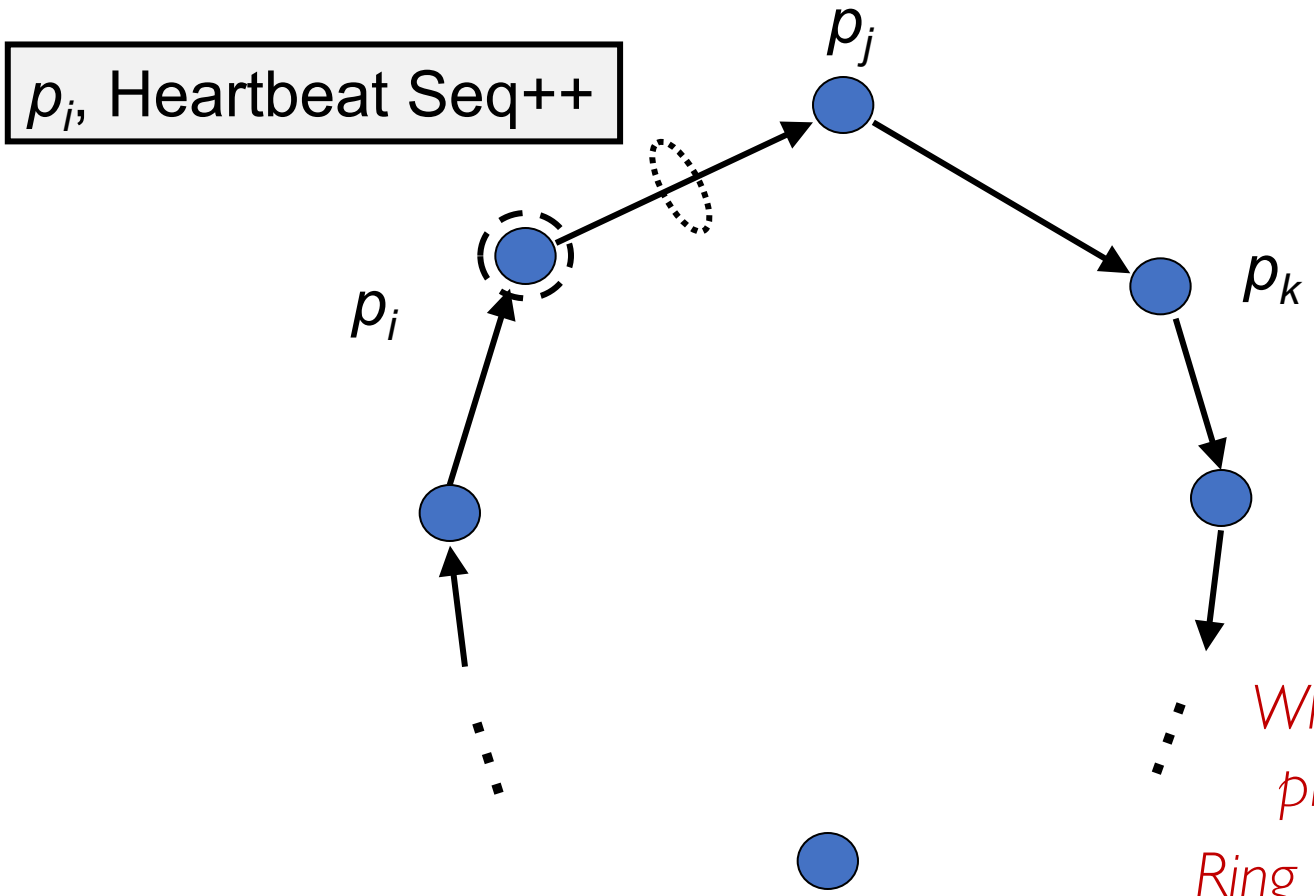
Centralized heartbeating

Downside:

What if p_i fails?



Ring heartbeating

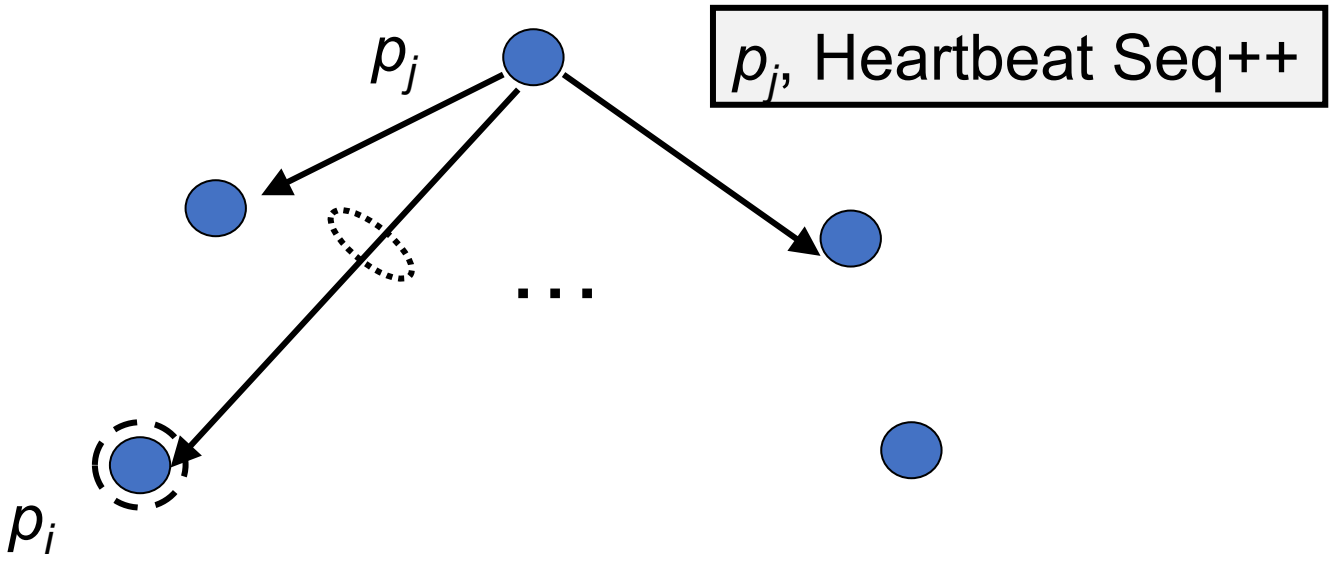


Downside:

What if multiple processes fail?

Ring repair overhead

All-to-all heartbeats



Everyone can keep track of everyone.

Downside:

Extending heartbeats

- Looked at detecting failure between two processes.
- How do we extend to a system with multiple processes?
  - Centralized heartbeating: *not complete.*
  - Ring heartbeating: *not entirely complete.*
  - All-to-all: *complete, but more bandwidth usage.*

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

- Failure Detection
  - Chapter 15.1
- Time and Clocks
  - Chapter 14.1-14.3

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.

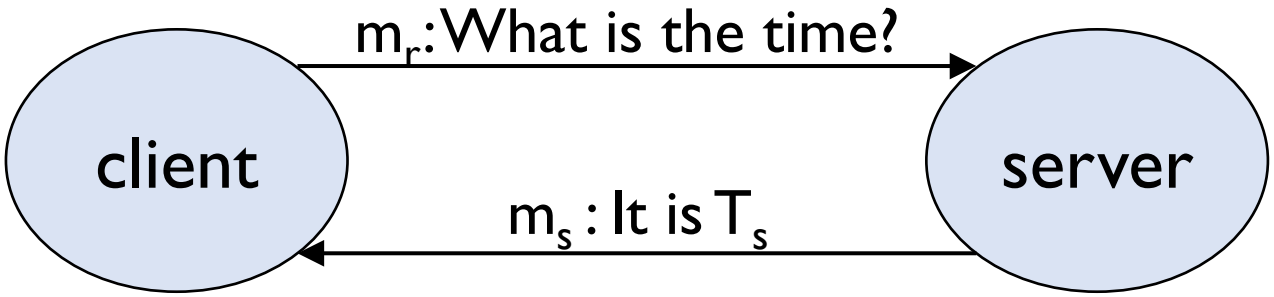
Synchronization Bound

- 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, B is *accurate* within a bound of *D*.

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?*

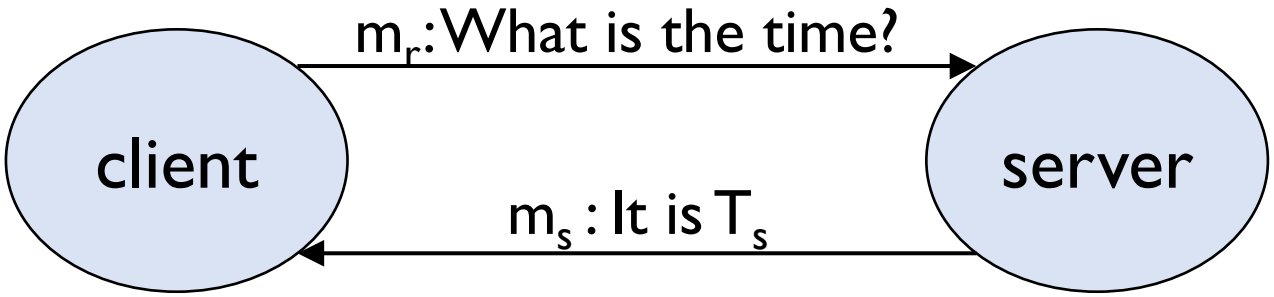
A: 2D. So the clocks are internally synchronized within a bound of 2D.

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 $T_c = T_s$, skew(client, server) $\leq$ *max*.

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

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

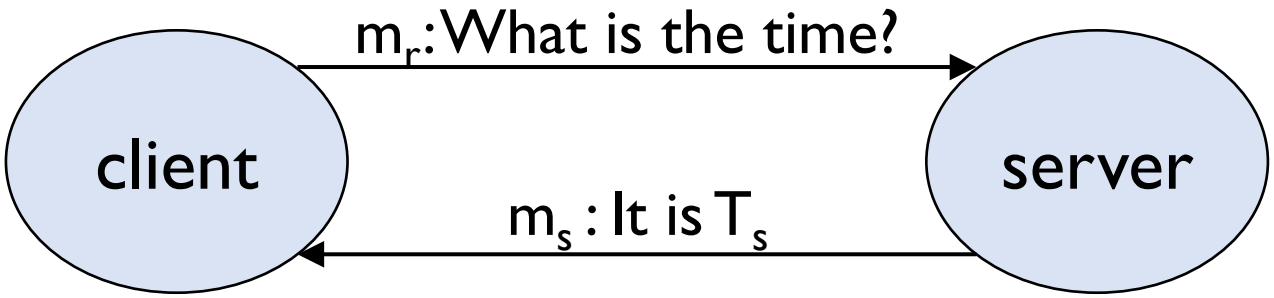
If $T_c = (T_s + (\mathbf{min + max})/2)$, skew(client,server) $\leq$ *(max – min)/2*

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

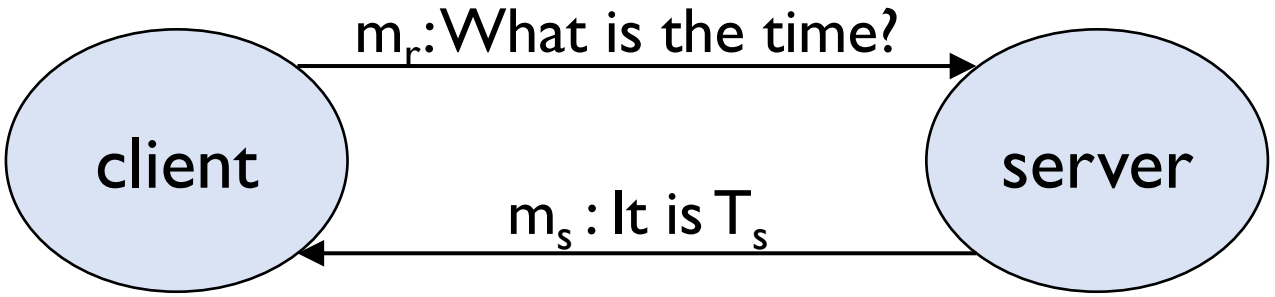
$\leq (T_{round} / 2)$

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

Cristian Algorithm



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

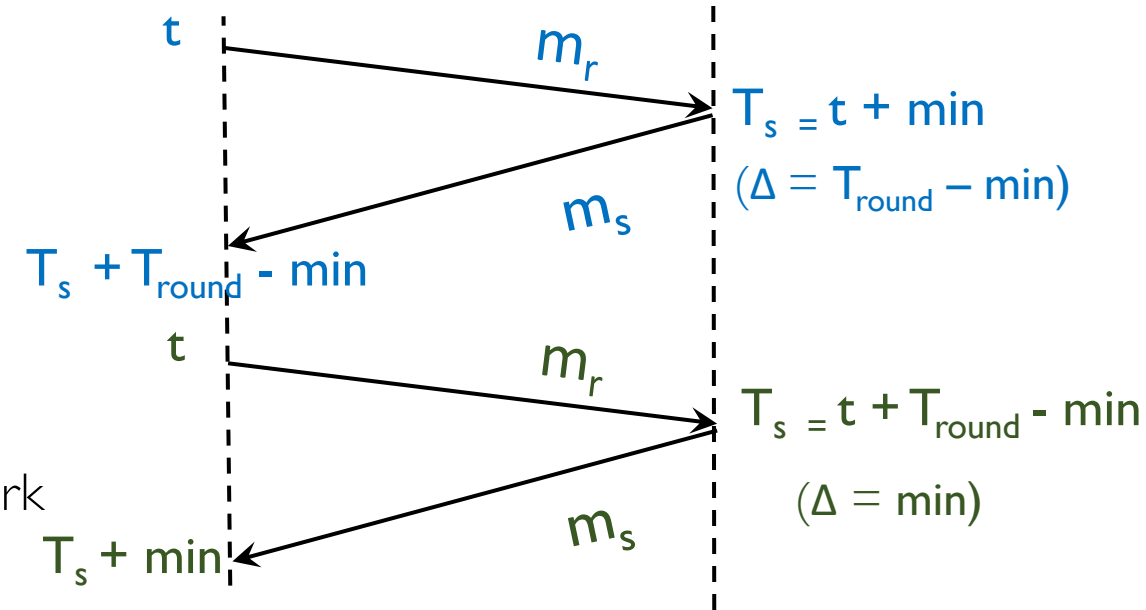
Client measures the round trip time (T_{round}).

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

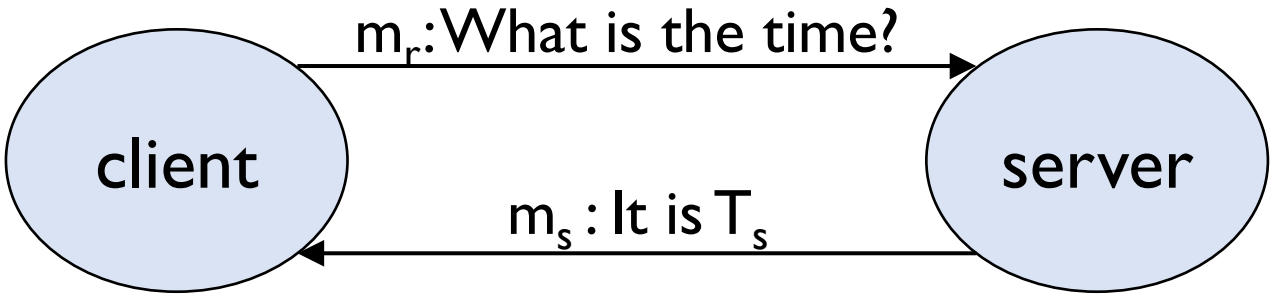
skew $\leq (T_{round} / 2) - min$

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

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



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}).

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

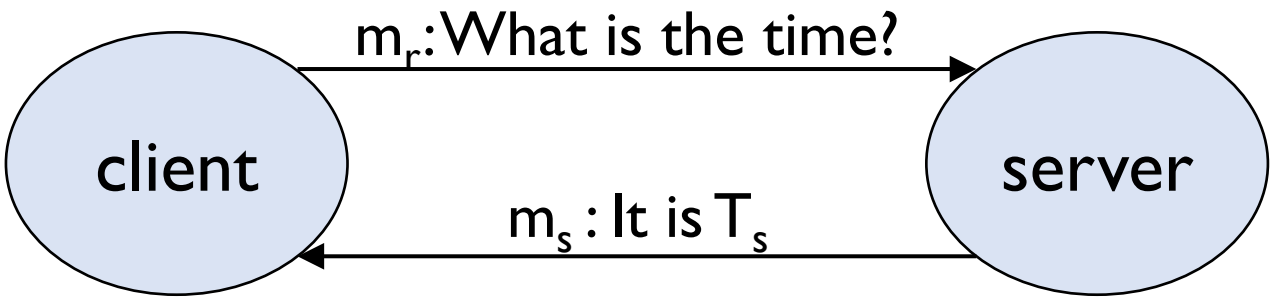
skew $\leq (T_{round} / 2) - min$
$\leq (T_{round} / 2)$

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

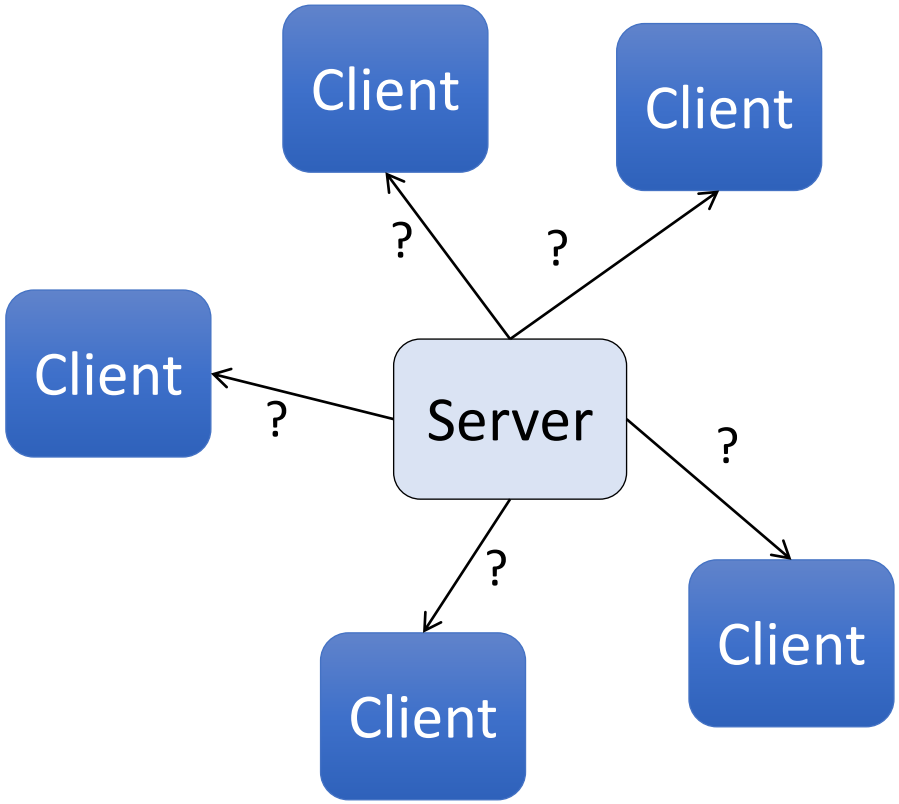
$\leq (T_{round} / 2)$

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

Cannot handle faulty time servers.

Berkeley Algorithm

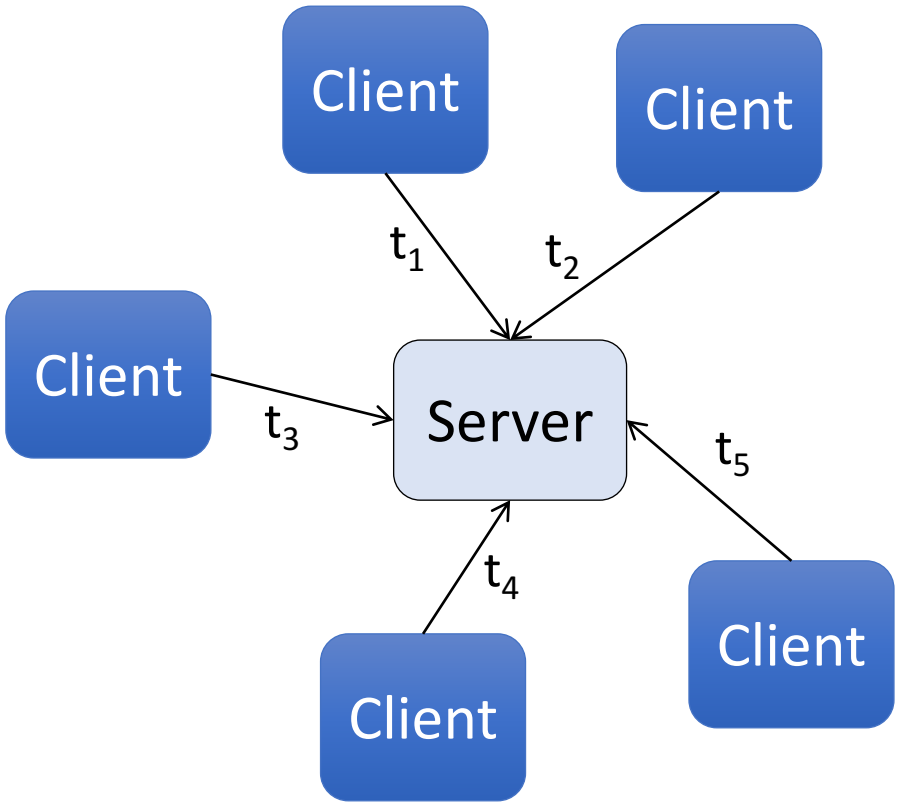
Only supports internal synchronization.



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

Berkeley Algorithm

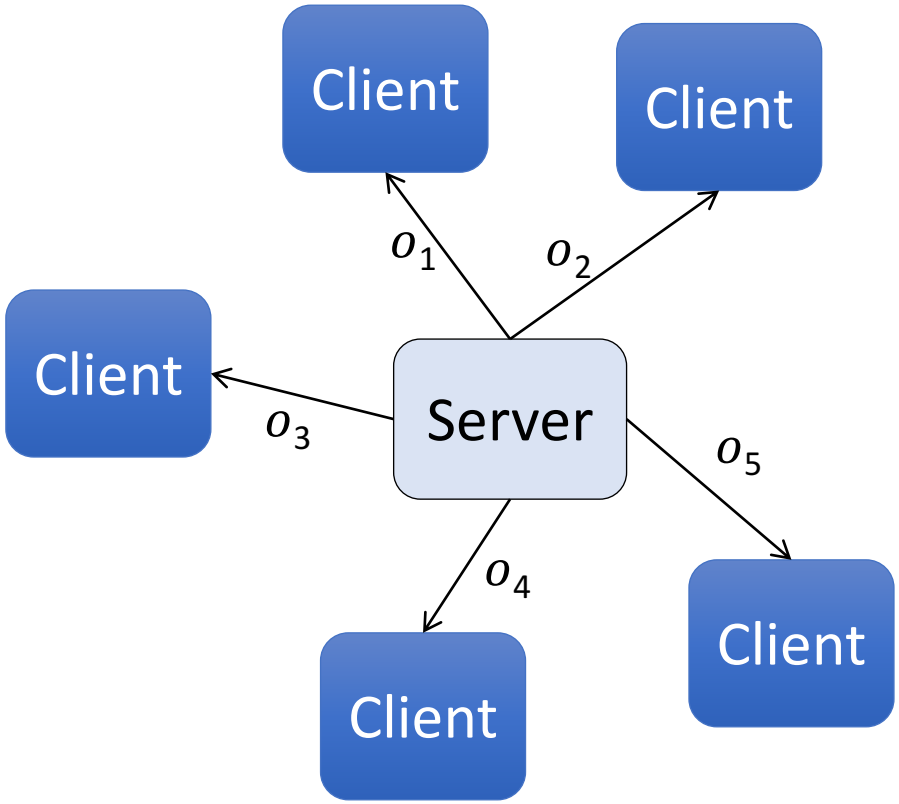
Only supports internal synchronization.



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

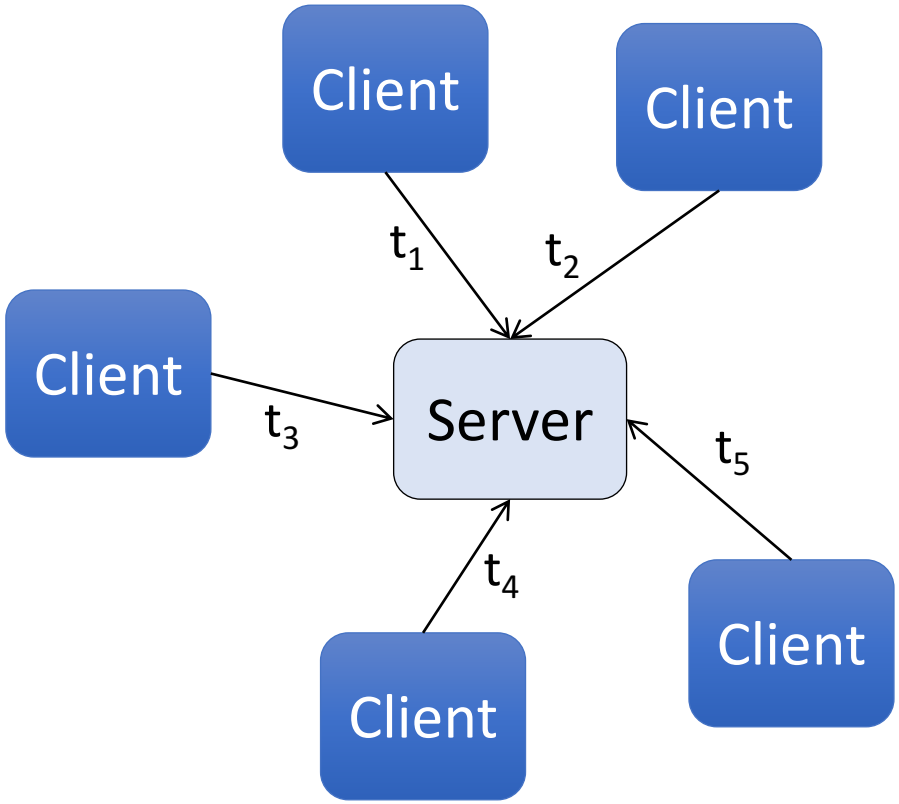
Only supports internal synchronization.



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.

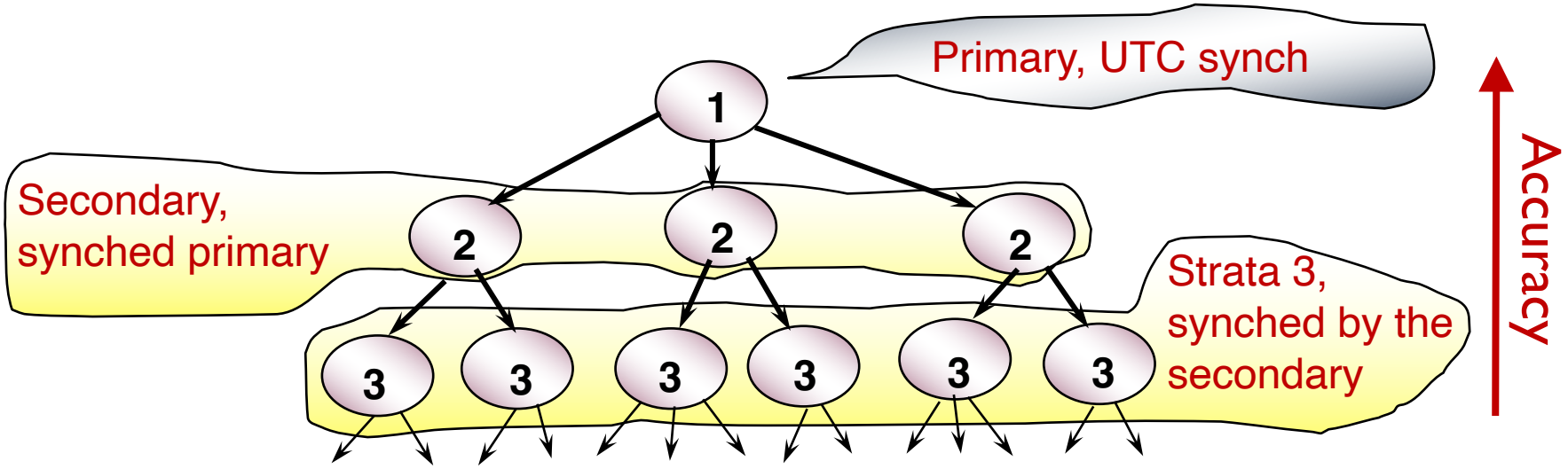


Handling faulty processes:
Only use timestamps within some threshold of each other.

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

Network Time Protocol

Time service over the Internet for synchronizing to UTC.



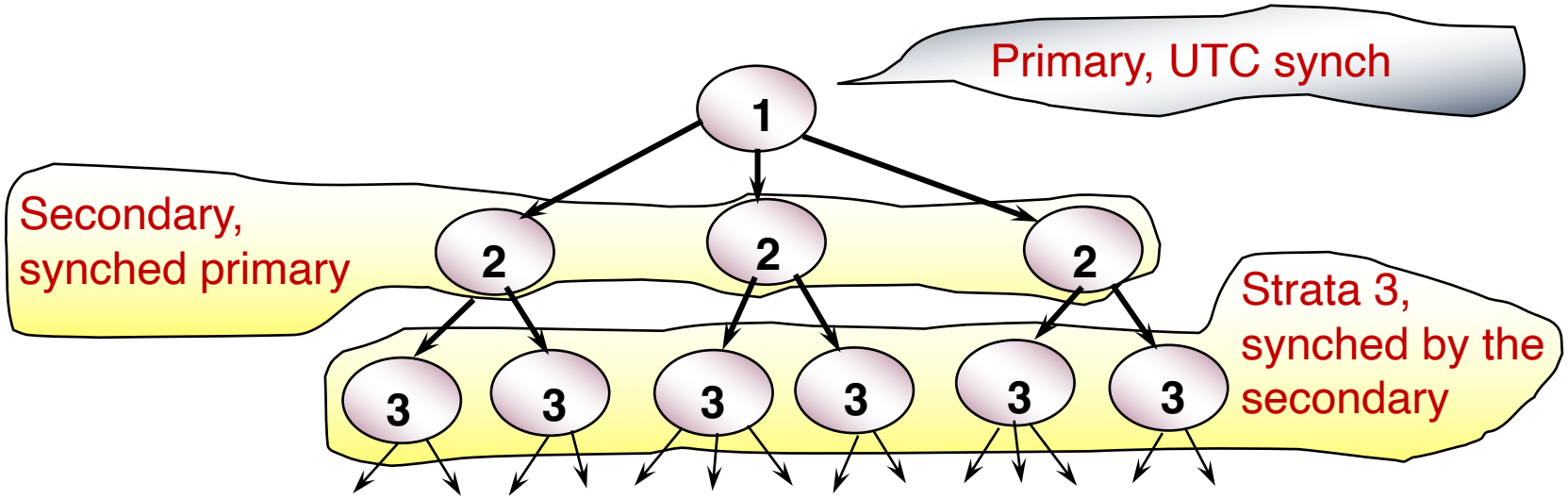
Hierarchical structure for *scalability*.

Multiple lower strata servers for *robustness*.

Authentication mechanisms for *security*.

Statistical techniques for better *accuracy*.

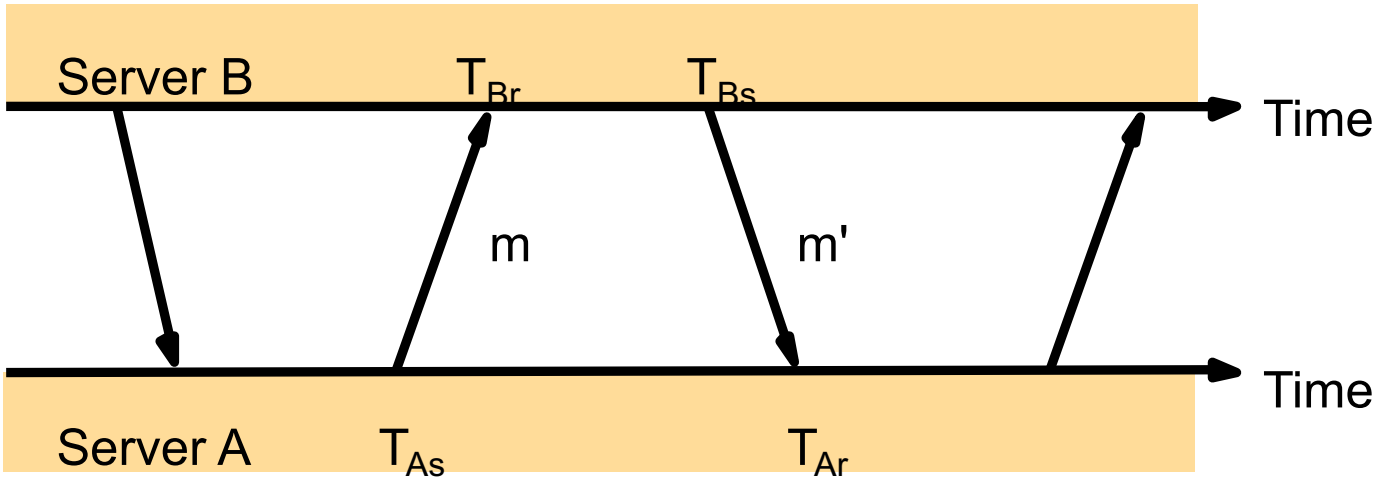
Network Time Protocol



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*.

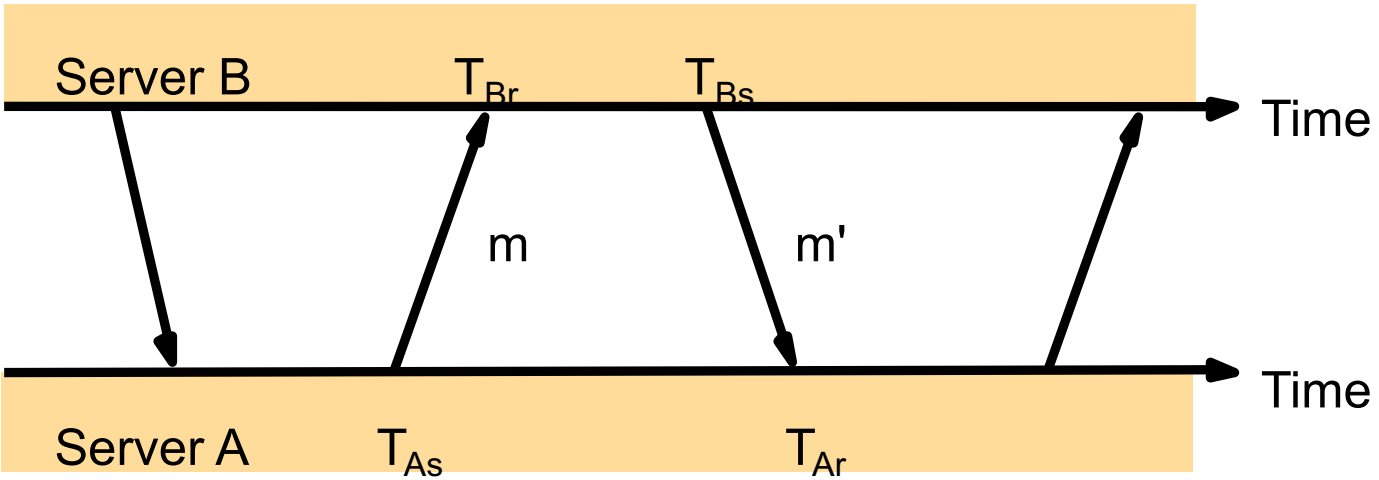
NTP Symmetric Mode



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

Use these timestamps to compute offset with respect to one another (o_i).

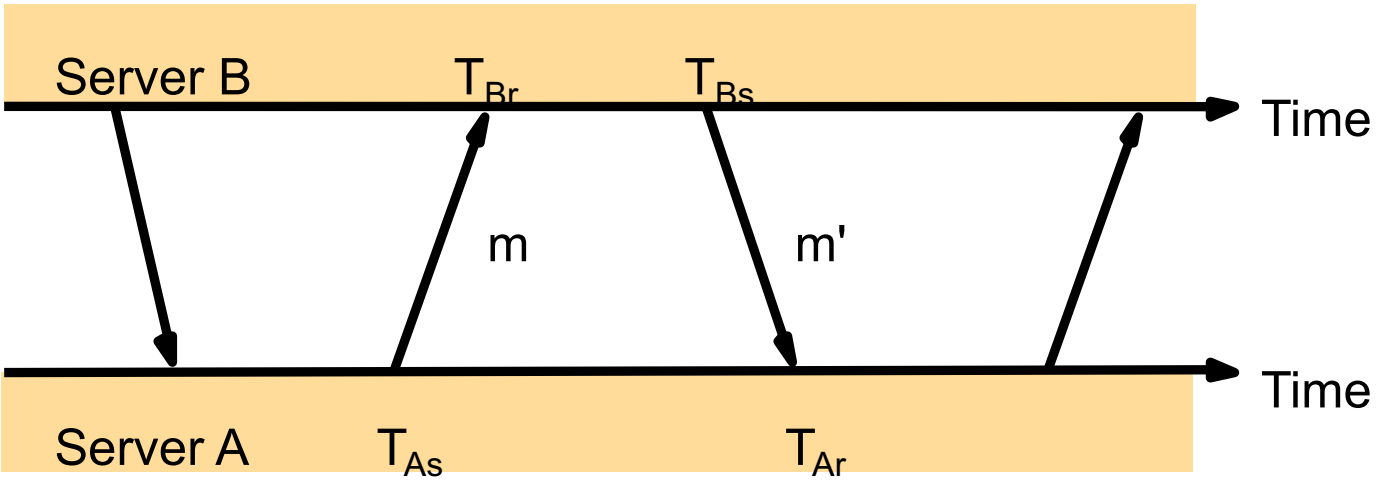
NTP Symmetric Mode



- 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'

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

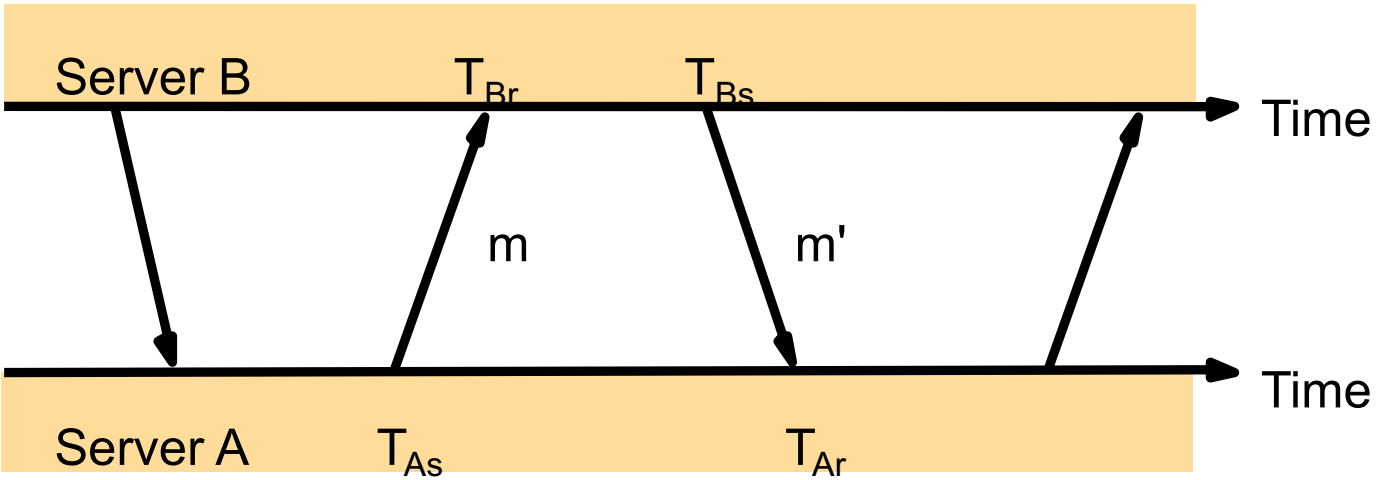
NTP Symmetric Mode



- t and t': actual transmission times for m and m'(unknown)
- o: 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'

$$T_{Br} = T_{As} + t + o$$
$$T_{Ar} = T_{Bs} + t' - o$$
$$o = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) + (t' - t))/2$$
$$\mathbf{o_i = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}))/2}$$
$$o = o_i + (t' - t)/2$$
$$\mathbf{d_i = t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs})}$$
$$(o_i - d_i / 2) \leq o \leq (o_i + d_i / 2) \quad \text{given } t, t' \geq 0$$

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.
  - Symmetric mode synchronization for higher accuracy.