Distributed Systems

CS425/ECE428

01/29/2020

Logistics

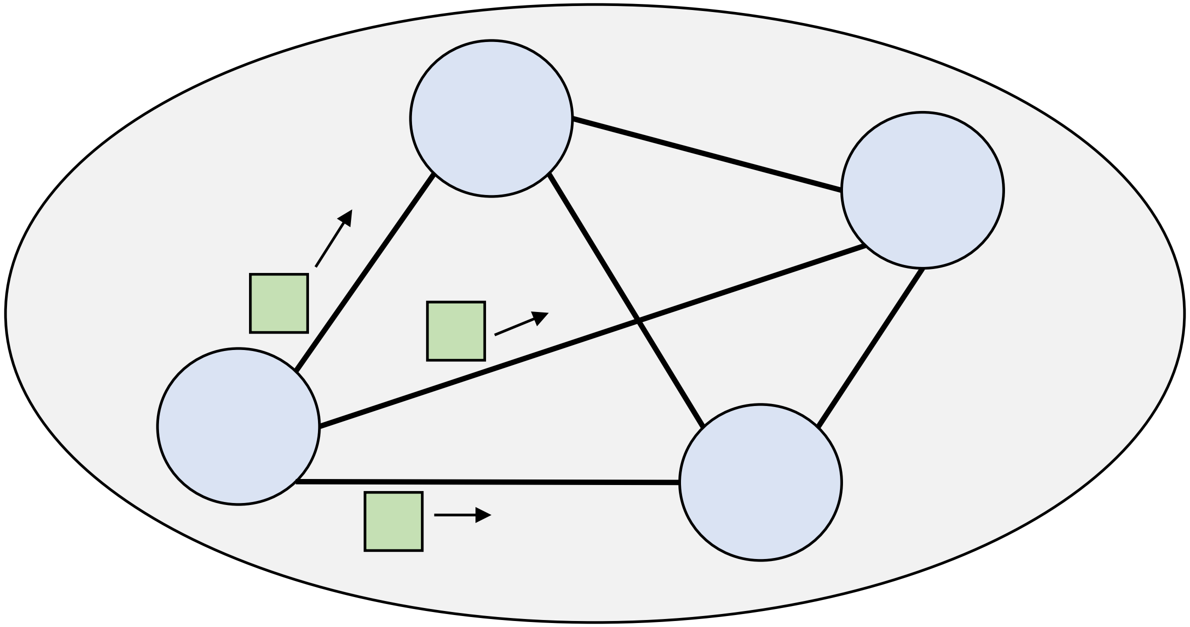
- **Slide policy:**
  - Lecture slides v1
    - By noon the day of the lecture.
  - Lecture slides v2
    - By 6pm on the day of the lecture.

- **MP0:** Please sign up for groups if you have not already done so.

Today's agenda

- Wrap up failure model and detection
  - Chapter 2.4 (except 2.4.3), Chapter 15.1
- Time and Clocks
  - Chapter 14.1-14.3

Recap: What is a distributed system?



Independent processes that are **connected by a network** and communicate by **passing messages** to achieve a common goal, appearing as **a single coherent system**.

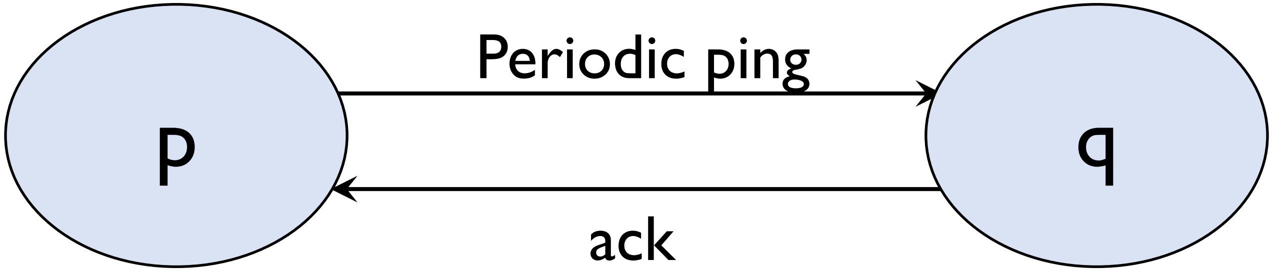
Recap from last class

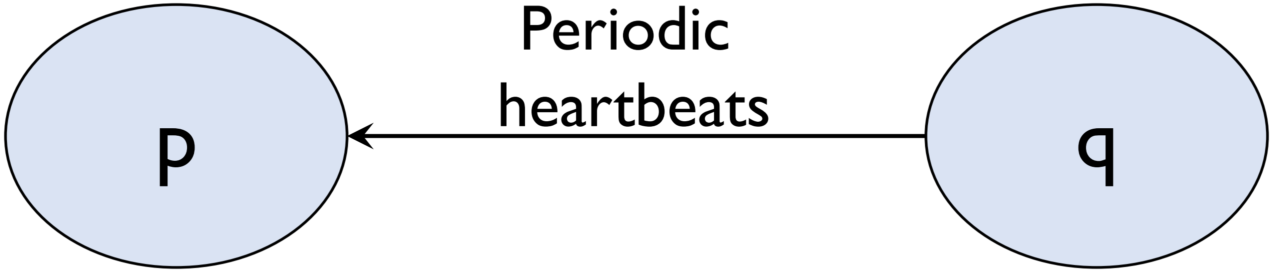
- Relationship between processes
  - Client-server and peer-to-peer
- Sources of uncertainty
  - Communication time, clock drift rates
- Synchronous vs asynchronous models.
- Failure model and detection.

Types of failure

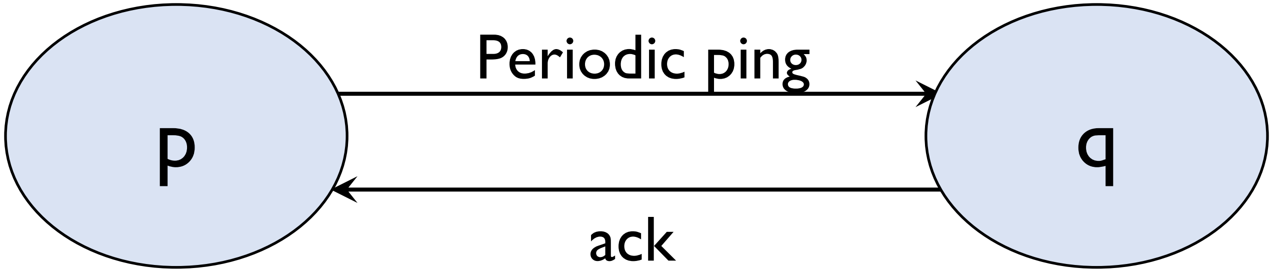
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.

How to detect a crashed process?





How to detect a crashed process?



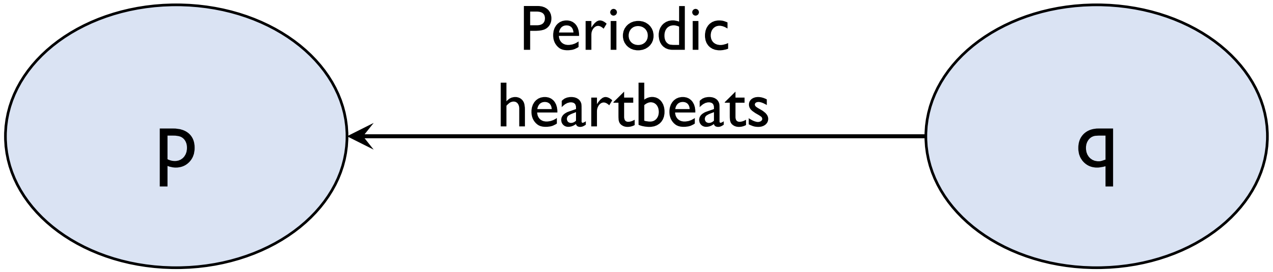
Pings are sent every T seconds.

Δ_1 time elapsed after sending ping, and no ack, report crash.

If synchronous, Δ_1 = 2(max network delay)

If asynchronous, Δ_1 = k(max observed round trip time)

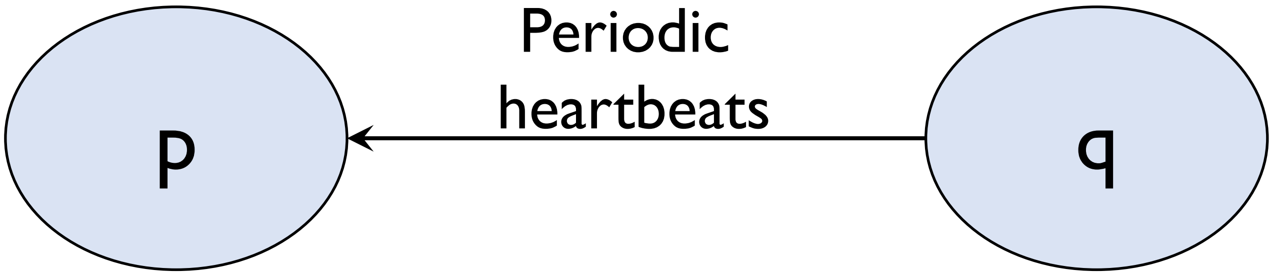
How to detect a crashed process?



Heartbeats are sent every T seconds.
$(T + \Delta_2)$ time elapsed since last heartbeat, report crash.

If synchronous, Δ_2 = max network delay – min network delay
If asynchronous, Δ_2 = k(observed delay)

How to detect a crashed process?

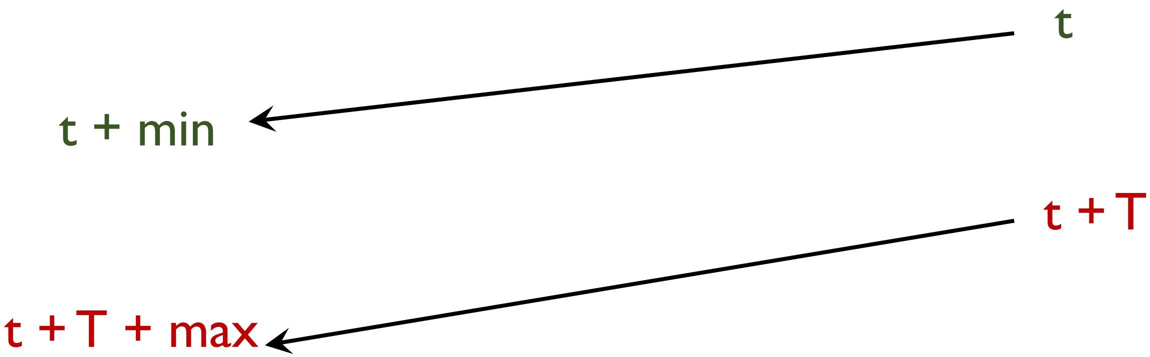


p

Periodic heartbeats

q

$(T + \Delta_2)$ time elapsed since last heartbeat.



t

t + min

t + T

t + T + max

Correctness of failure detection

- Completeness
    - Every failed process is *eventually* detected.
- Accuracy
    - Every detected failure corresponds to a crashed process (no mistakes).

Correctness of failure detection

- Characterized by **completeness** and **accuracy**.

- Synchronous system
  - Failure detection via ping-ack and heartbeat is both complete and accurate.

- Asynchronous system
  - *Our strategy for ping-ack and heartbeat is complete.*
  - Impossible to achieve both completeness and accuracy.
  - Can we have an accurate but incomplete algorithm?
    - *Never report failure.*

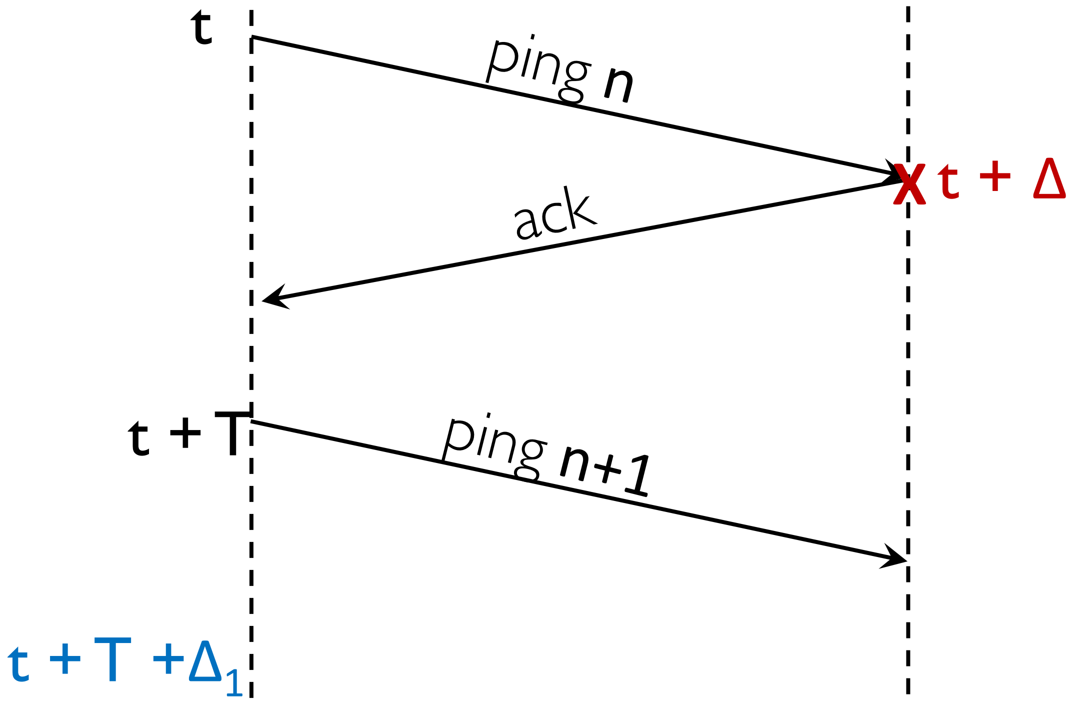
Metrics for failure detection

- Worst case failure detection time

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_l - \Delta$, where Δ is time taken for previous ping from p to reach q

    T is the time period for pings, and Δ_l is timeout value.



Worst case failure detection time:

$t + T + \Delta_1 - t + \Delta$

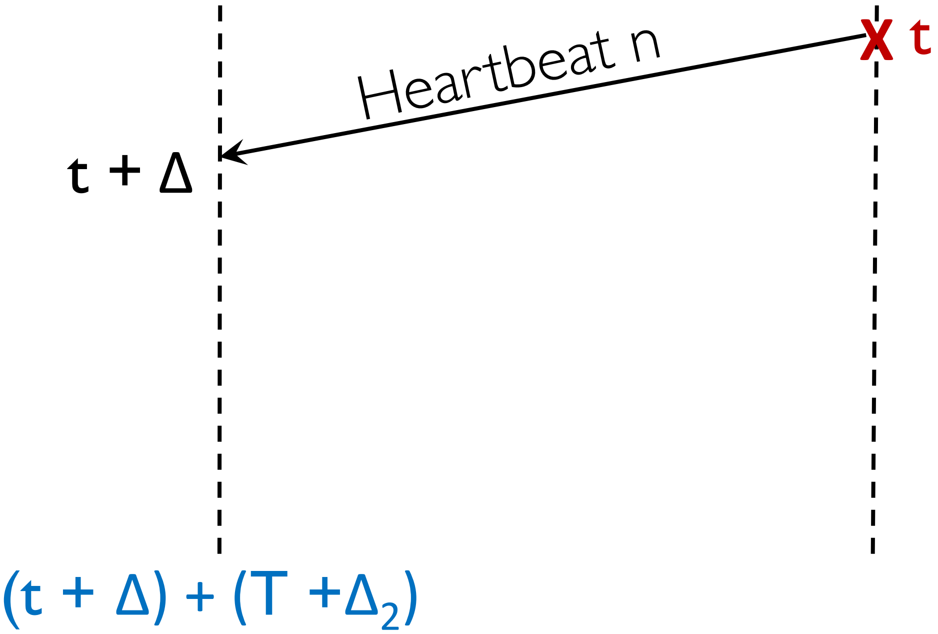
$= T + \Delta_1 - \Delta$

Q: What is worst case value of ***Δ*** for a *synchronous system?*

A: min network delay

Metrics for failure detection

- Worst case failure detection time
  - Heartbeat: $\Delta + T + \Delta_2$ where Δ is time taken for last heartbeat from q to reach p
    T is the time period for heartbeats, and $T + \Delta_2$ is the timeout.

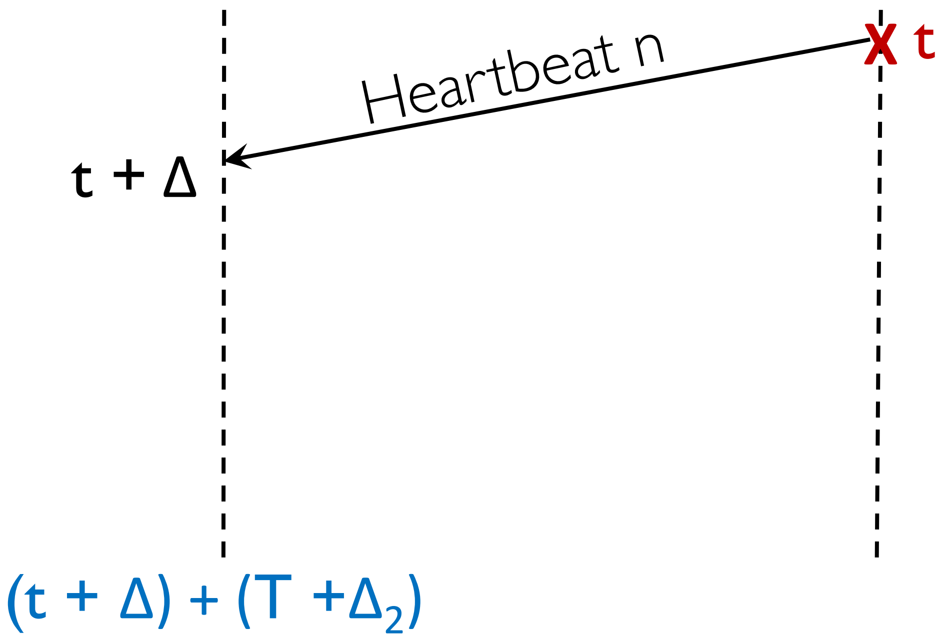


Worst case failure detection time:
$(t + \Delta) + (T + \Delta_2) - t$
$= T + \Delta_2 + \Delta$

Q: What is worst case value of ***Δ*** in a *synchronous system?*
A: max network delay

Metrics for failure detection

- Worst case failure detection time
  - Heartbeat: $\Delta + T + \Delta_2$ where Δ is time taken for last heartbeat from q to reach p
    T is the time period for heartbeats, and $T + \Delta_2$ is the timeout.

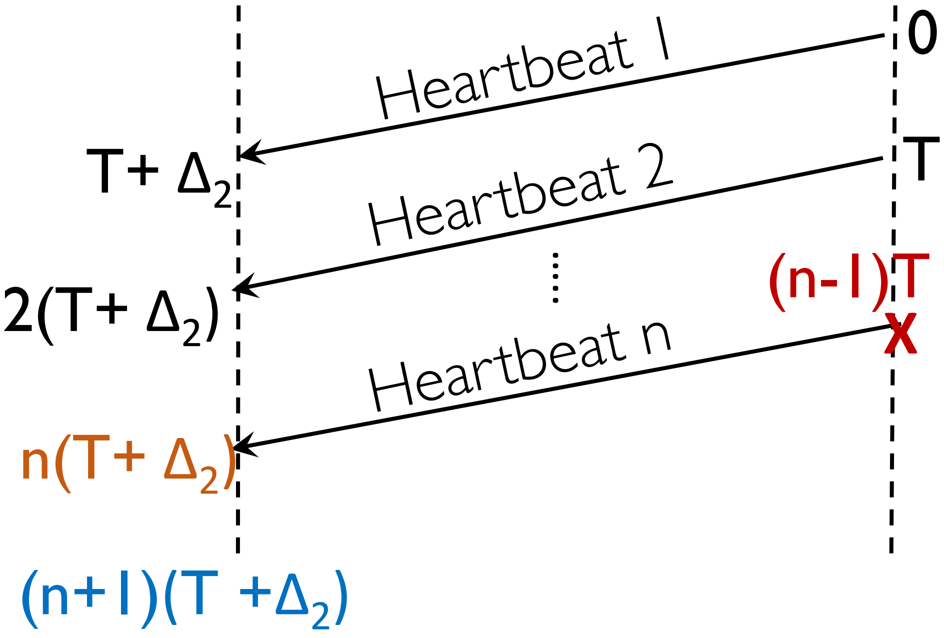


Worst case failure detection time:
$(t + \Delta) + (T + \Delta_2) - t$
$= T + \Delta_2 + \Delta$

Q: What is worst case value of Δ in an *asynchronous system?*

Metrics for failure detection

- Worst case failure detection time
  - Heartbeat: $\Delta + T + \Delta_2$ where Δ is time taken for last heartbeat from q to reach p
    T is the time period for heartbeats, and $T + \Delta_2$ is the timeout.



Worst case failure detection time:
$(t + \Delta) + (T + \Delta_2) - t$
$= T + \Delta_2 + \Delta$

Q: What is worst case value of Δ in *an asynchronous system?*

Worst case $\Delta = T + n\Delta_2$

Worst case detection time

$= 2T + (n+1)\Delta_2$

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: $\Delta + T + \Delta_2$ (where Δ is time taken for last heartbeat from q to reach p)

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: $\Delta + T + \Delta_2$ (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: $\Delta + T + \Delta_2$ (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: $\Delta + T + \Delta_2$ (where Δ is time taken for last heartbeat from q to reach p)

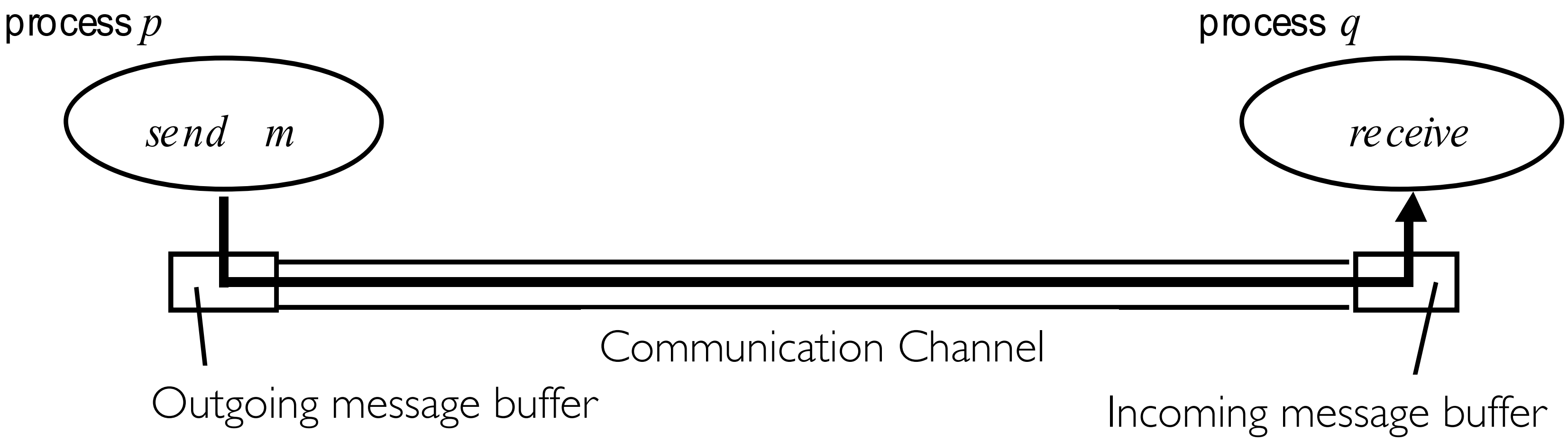
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

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

Types of failure

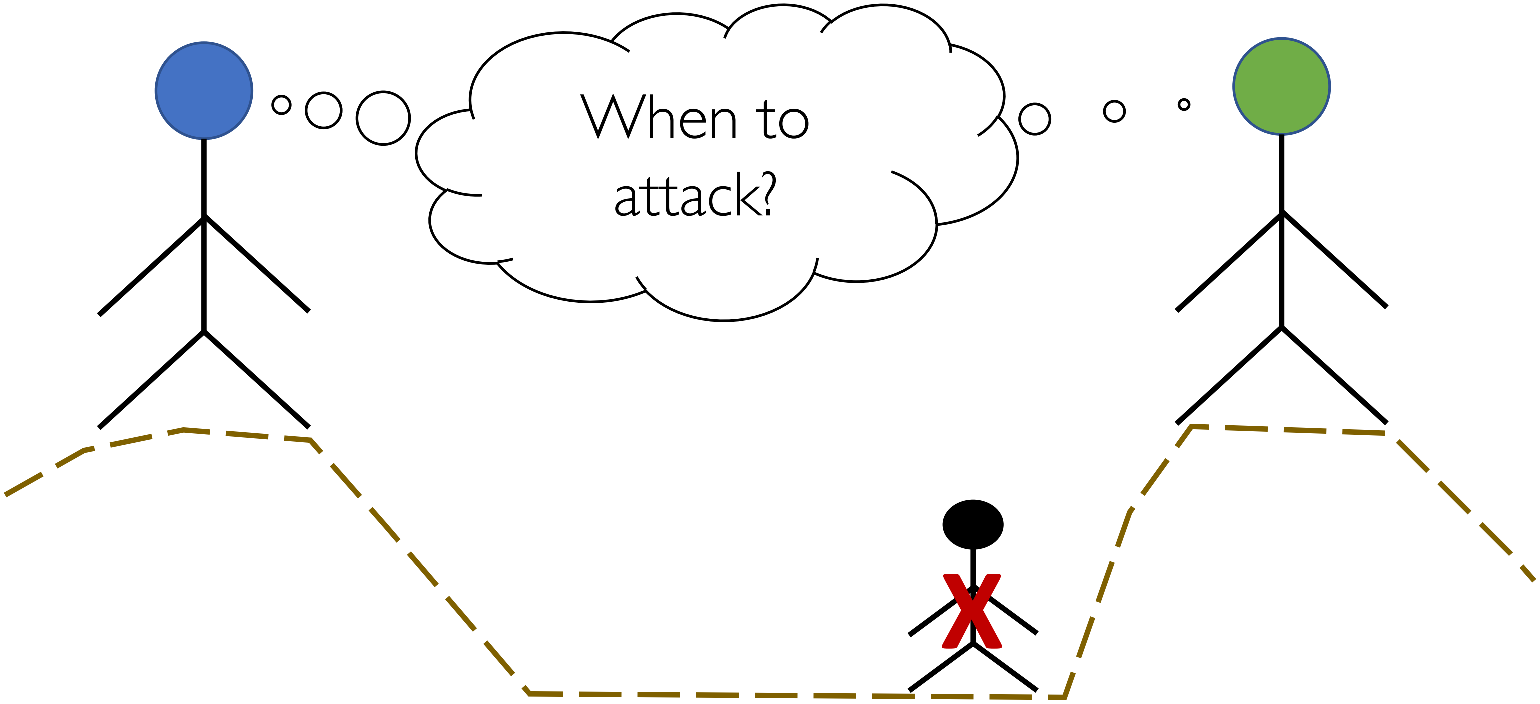
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.
  - **Fail-stop**: if other processes can certainly detect the crash.
  - **Communication omission**: a message sent by process was not received by another.

Communication Omission

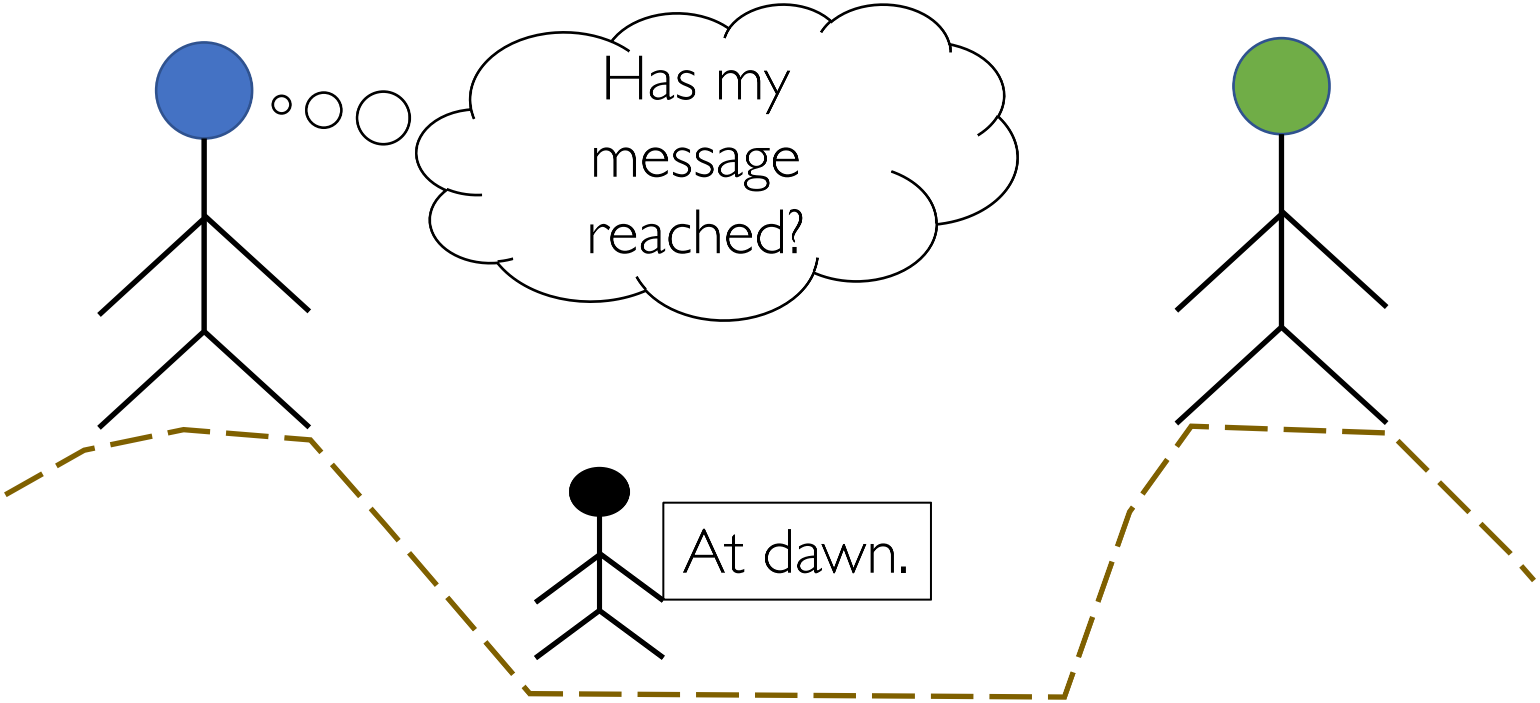


- **Channel omission:** omitted by channel
- **Send omission:** process completes 'send' operation, but message does not reach its outgoing message buffer.
- **Receive omission:** message reaches the incoming message buffer, but not received by the process.

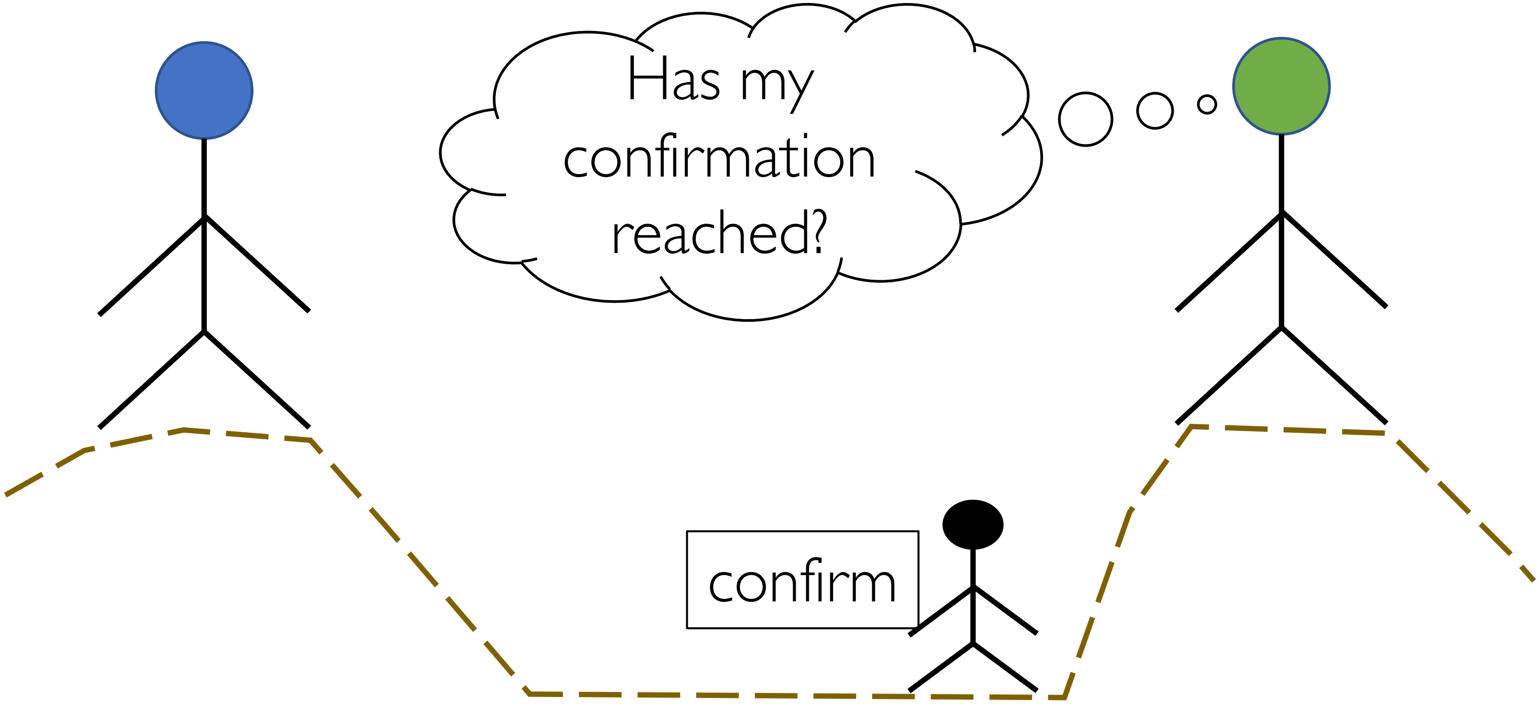
Two Generals Problem



Two Generals Problem



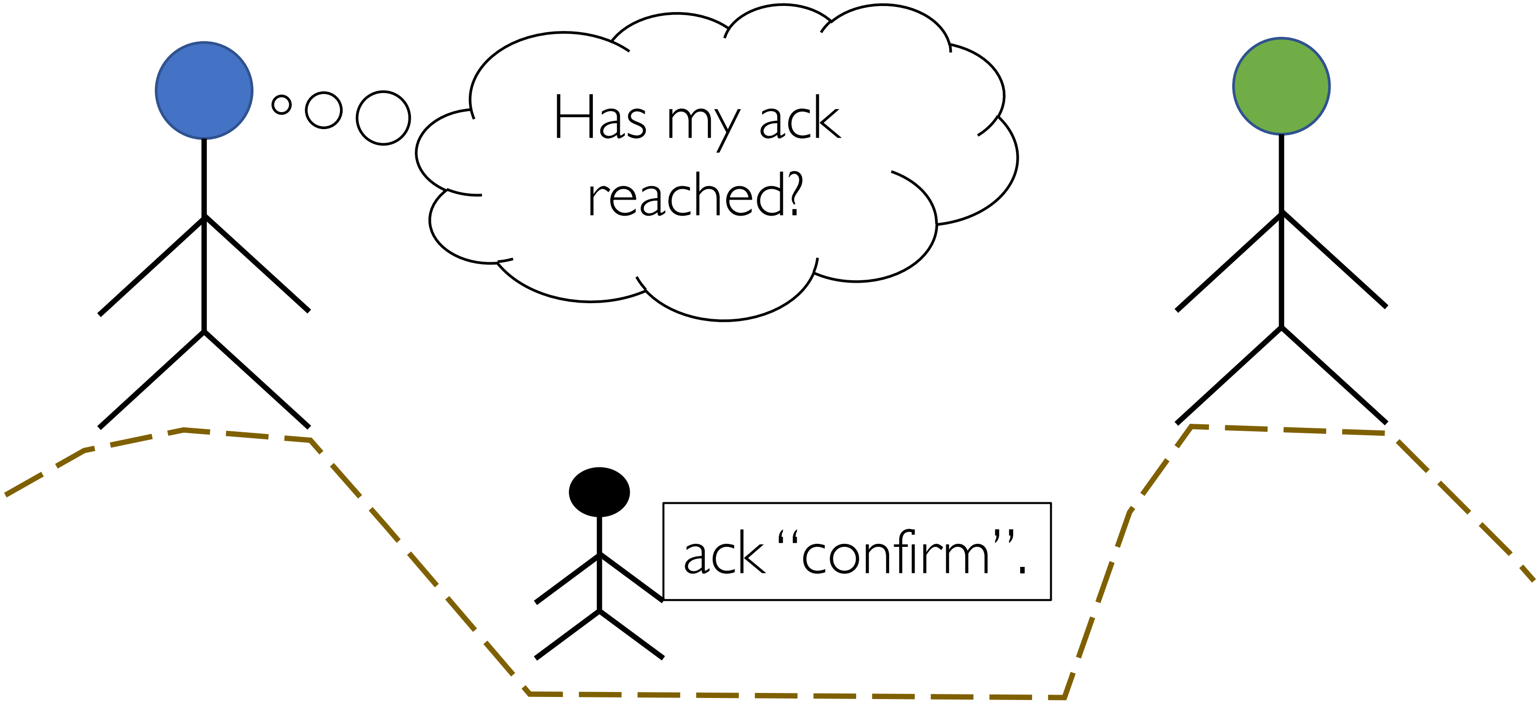
Two Generals Problem



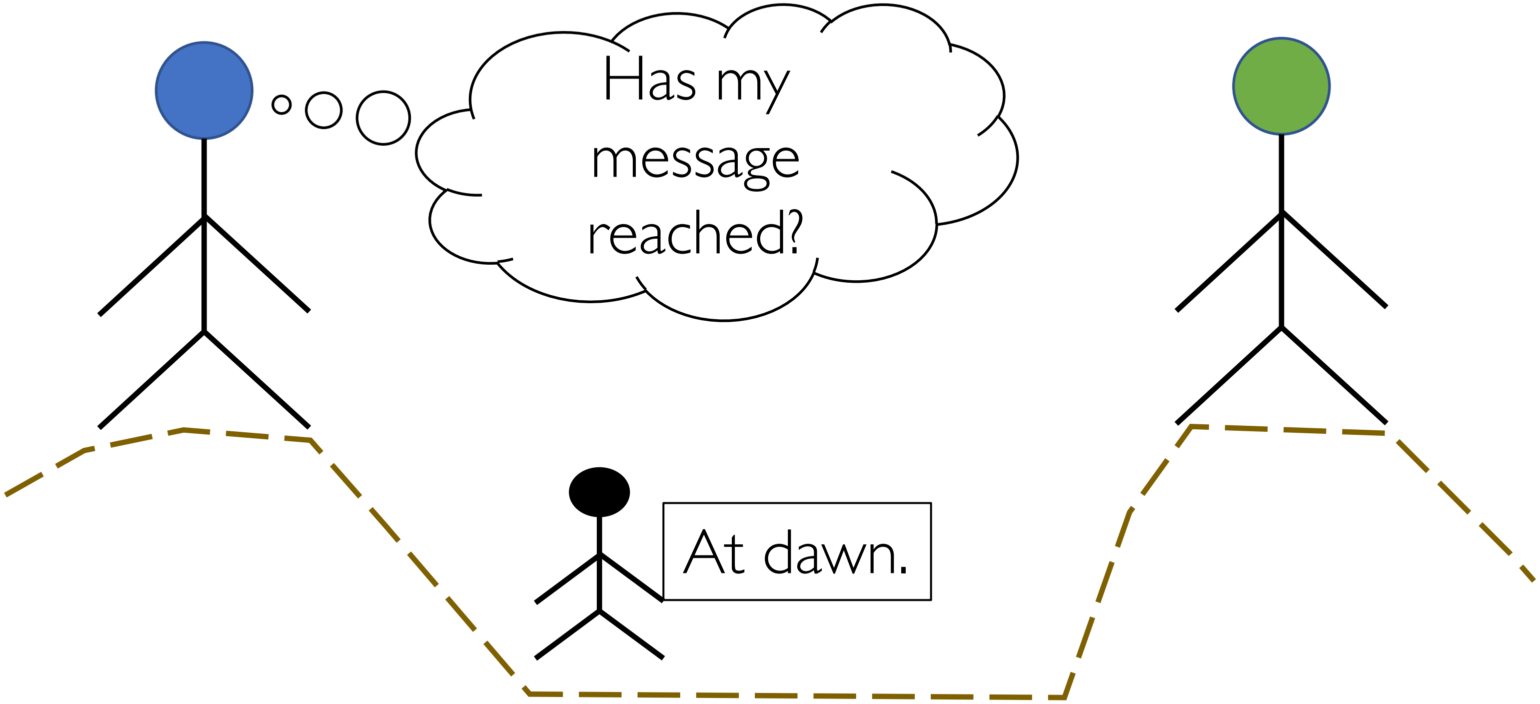
Has my confirmation reached?

confirm

Two Generals Problem

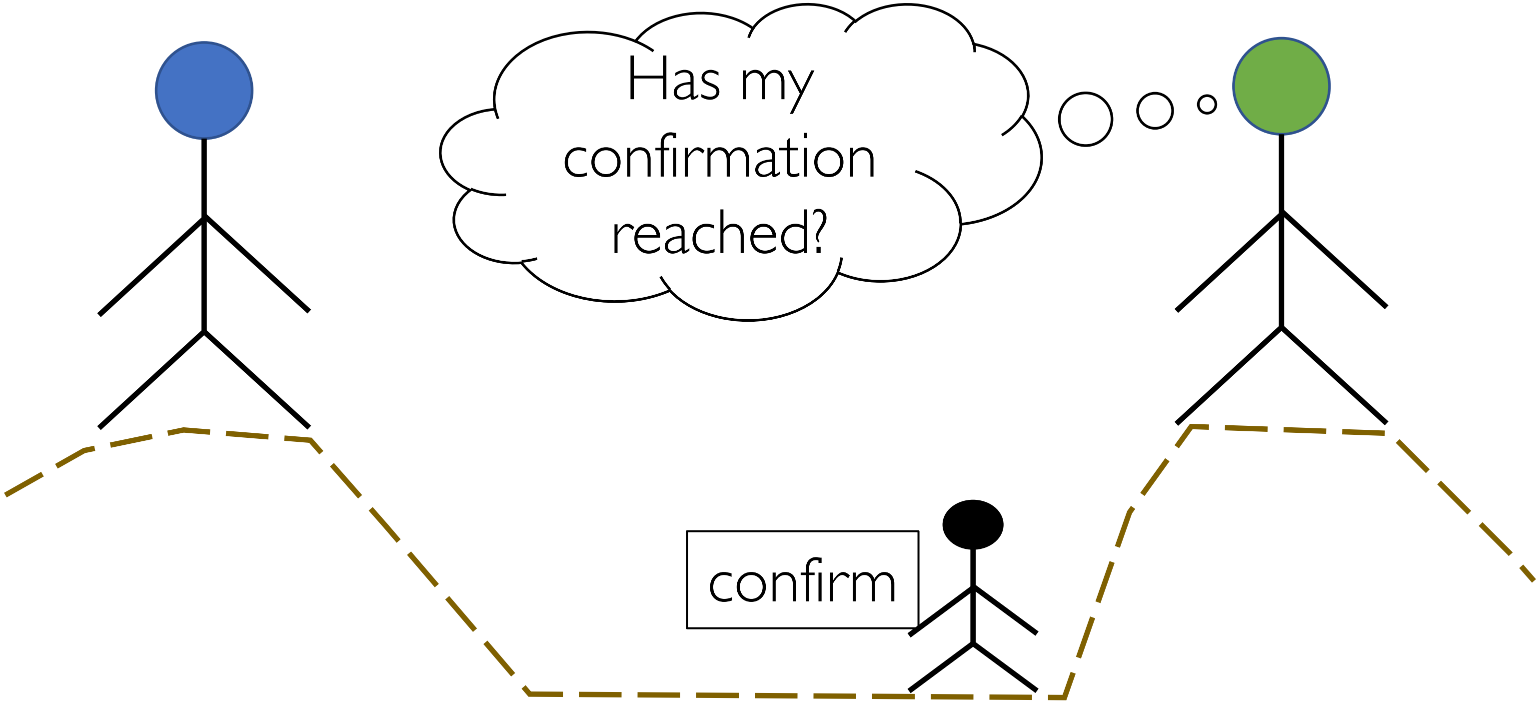


Two Generals Problem



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.

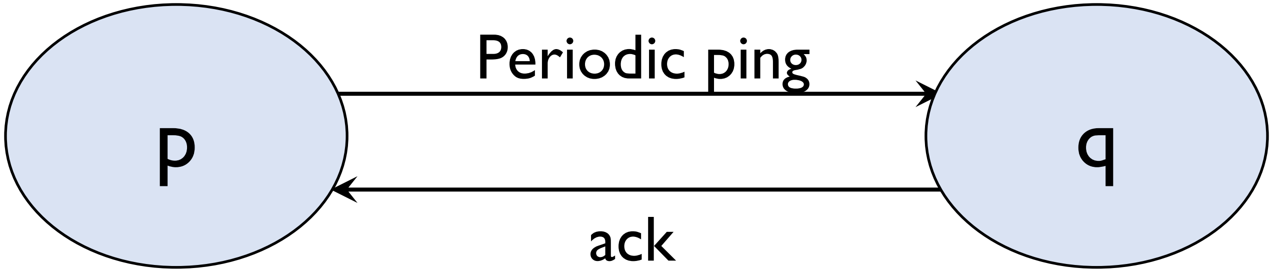
Types of failure

- **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.

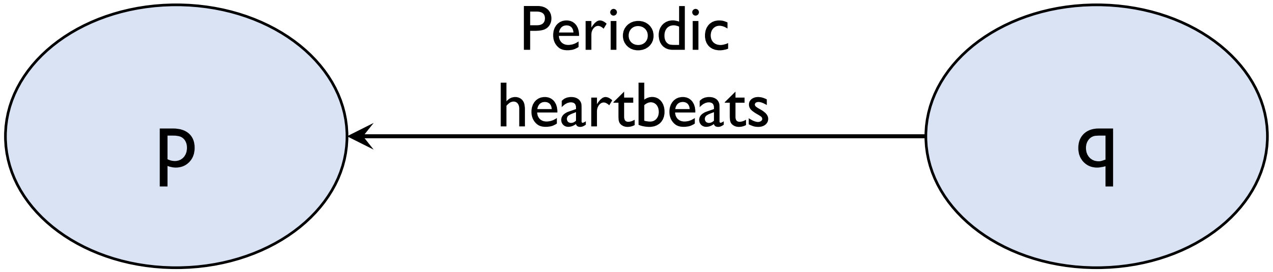
How to detect a crashed process?



Δ_1 time elapsed after sending ping, and no ack.

If synchronous, Δ_1 = 2(max network delay)
If asynchronous, Δ_1 = k(max observed roundtrip time)

How to detect a crashed process?



($T + \Delta_2$) time elapsed since last heartbeat.

If synchronous, Δ_2 = max network delay – min network delay
If asynchronous, Δ_2 = k(max observed delay)

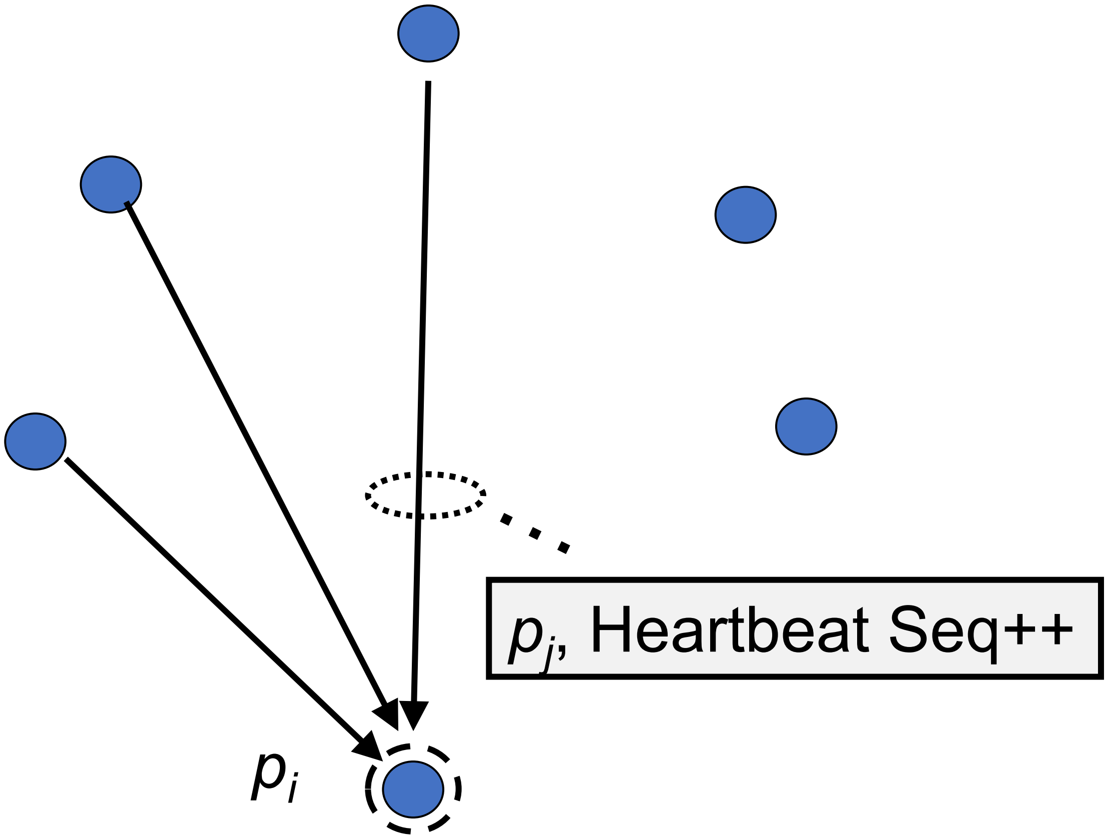
Extending heartbeats

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

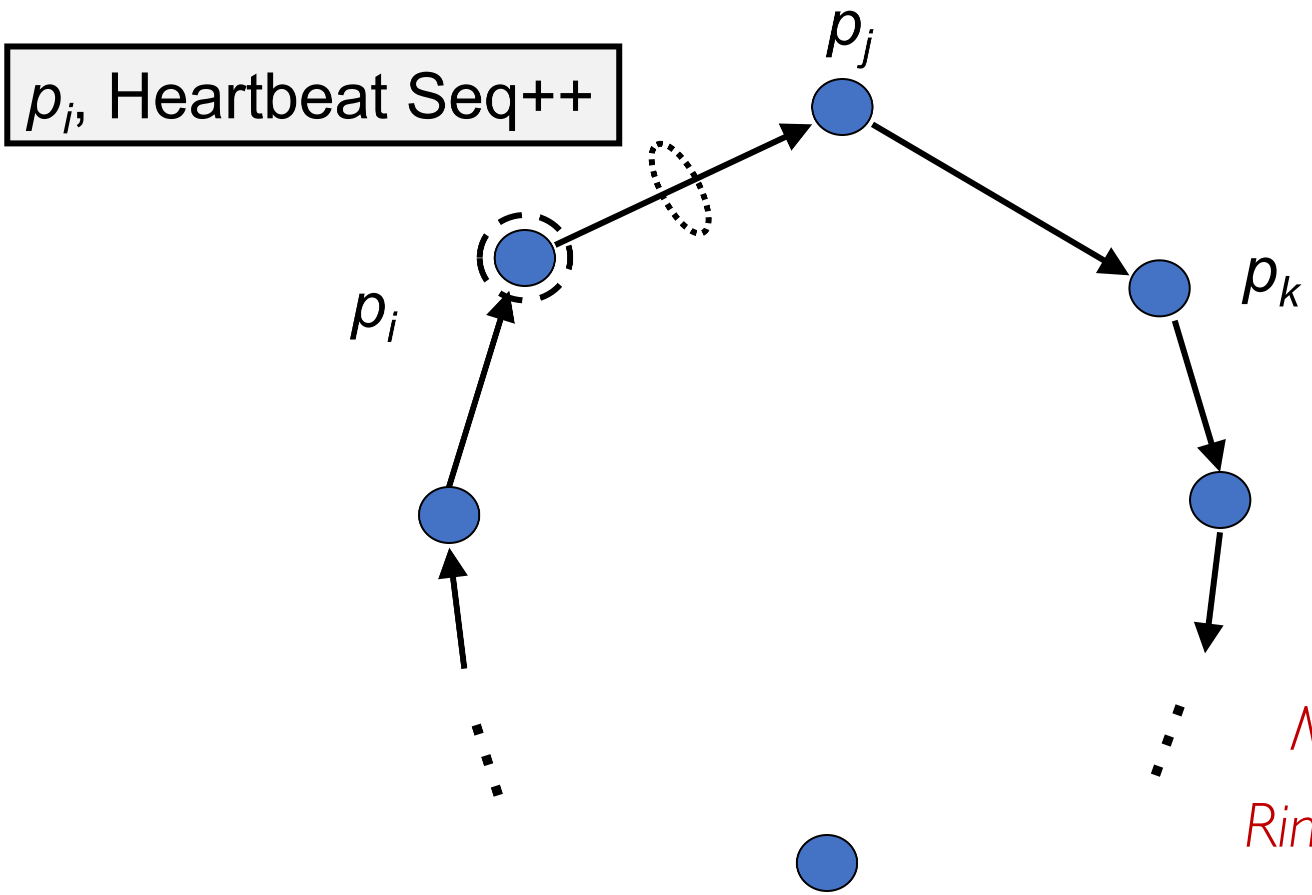
Centralized heartbeating

Downside:

What if p_i fails?



Ring heartbeating

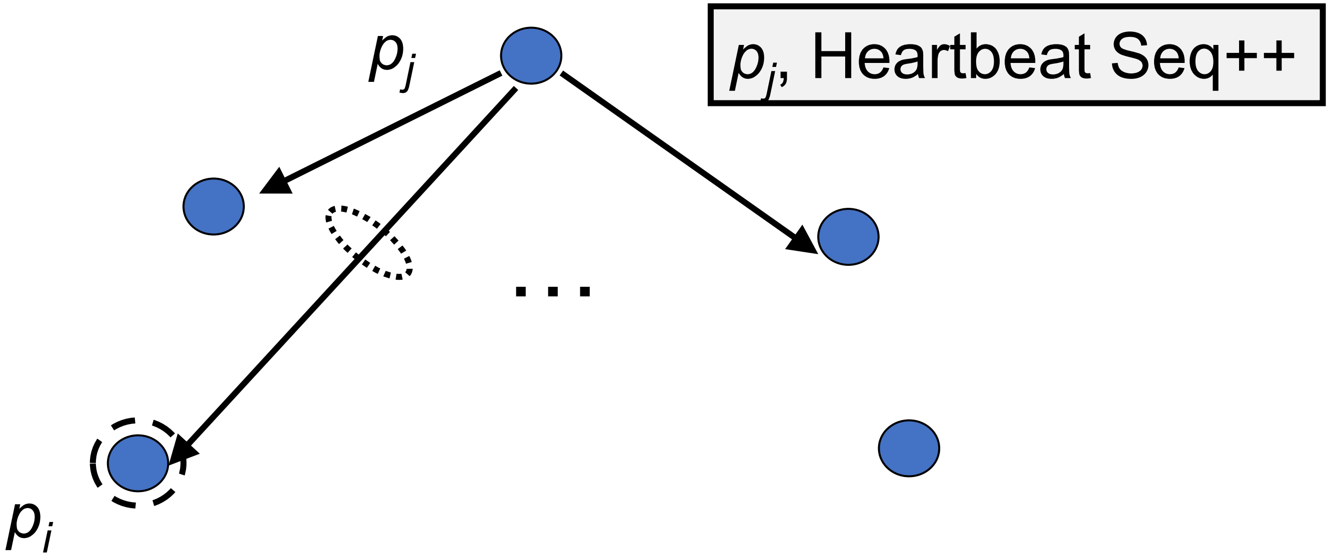


Downside:

Multiple failures

Ring repair overhead

All-to-all heartbeats



Everyone can keep track of everyone.

Downside: Bandwidth.

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

- Wrap up failure model and detection
  - Chapter 2.4 (except 2.4.3), Chapter 15.1
- Time and Clocks
  - Chapter 14

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.

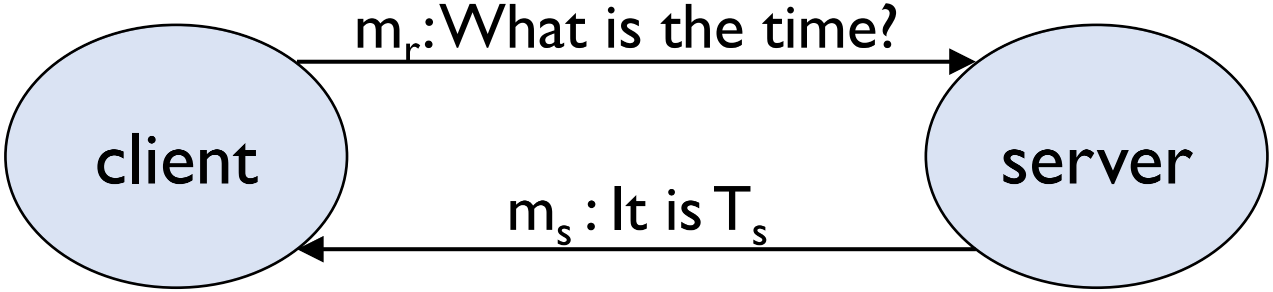
Synchronization Bound

- 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



m_r: What is the time?

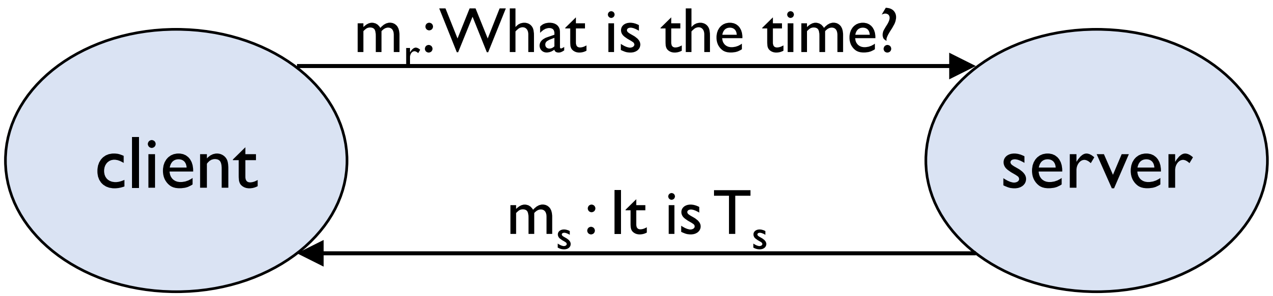
client

server

m_s : It is T_s

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

Synchronization in synchronous systems



m_r: What is the time?

client

server

m_s : It is T_s

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

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

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

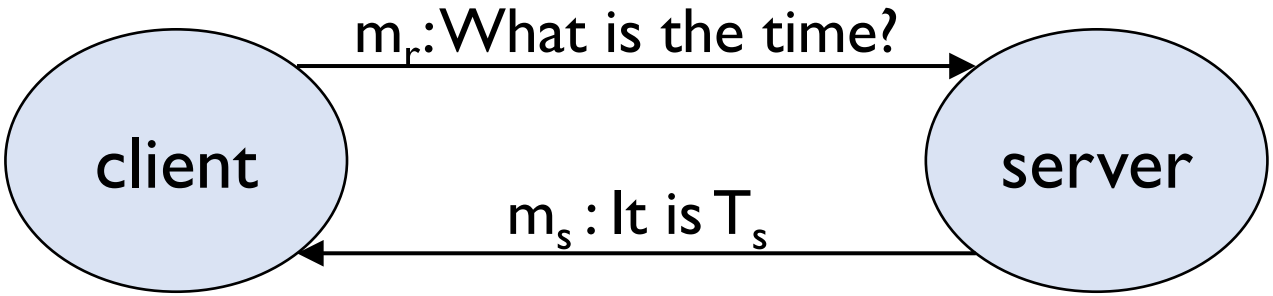
If $\mathbf{T_c} = \mathbf{(T_s + (min + max)/2)}$, skew(client,server) $\leq$ *(max – min)/2*

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 $\leq (T_{round} / 2) - min$ (*min* is minimum one way network delay).

Try deriving the worst case skew!

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

Next Class

- Wrap-up time synchronization:
    - Cristian algorithm, Berkeley algorithm, NTP
- Do we really need timestamps to reason about event ordering?
- How do we determine which events *happened before* a given event X?