Distributed Systems

CS425/ECE428

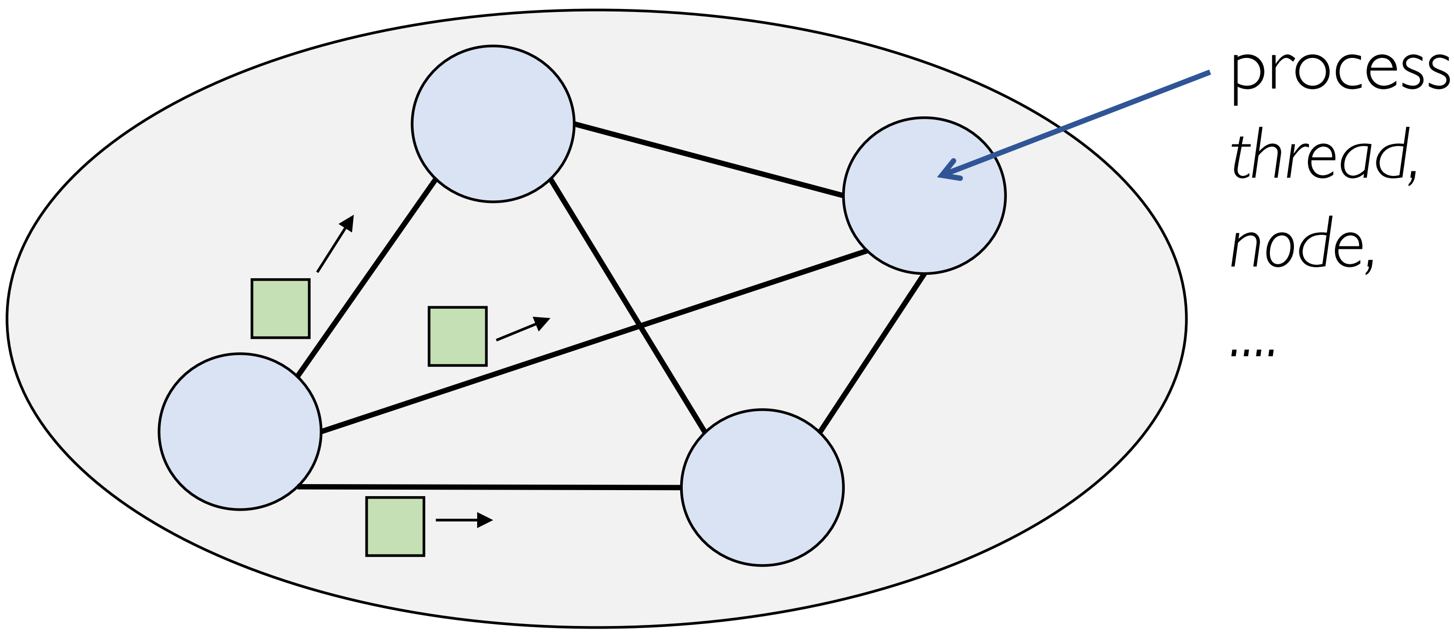
Logistics Related

- Undergraduates switching from T3 to T4
  - Please email Heather Mihaly and Elsa Gunter (hmihal2@illinois.edu, egunter@illinois.edu) with the request and your UIN.

Today's agenda

- System Model
  - Chapter 2.4 (except 2.4.3), parts of Chapter 2.3
- Failure Detection
  - Chapter 15.1

What is a distributed system?



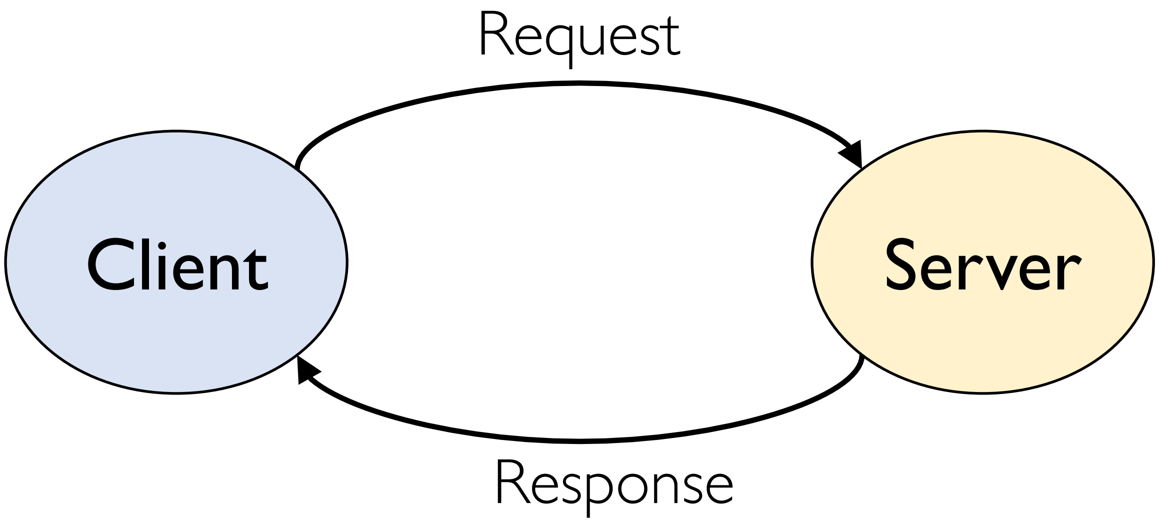
Independent components that are **connected by a network** and communicate by **passing messages** to achieve a common goal, appearing as **a single coherent system**.

Relationship between processes

- Two main categories:
  - Client-server
  - Peer-to-peer

Relationship between processes

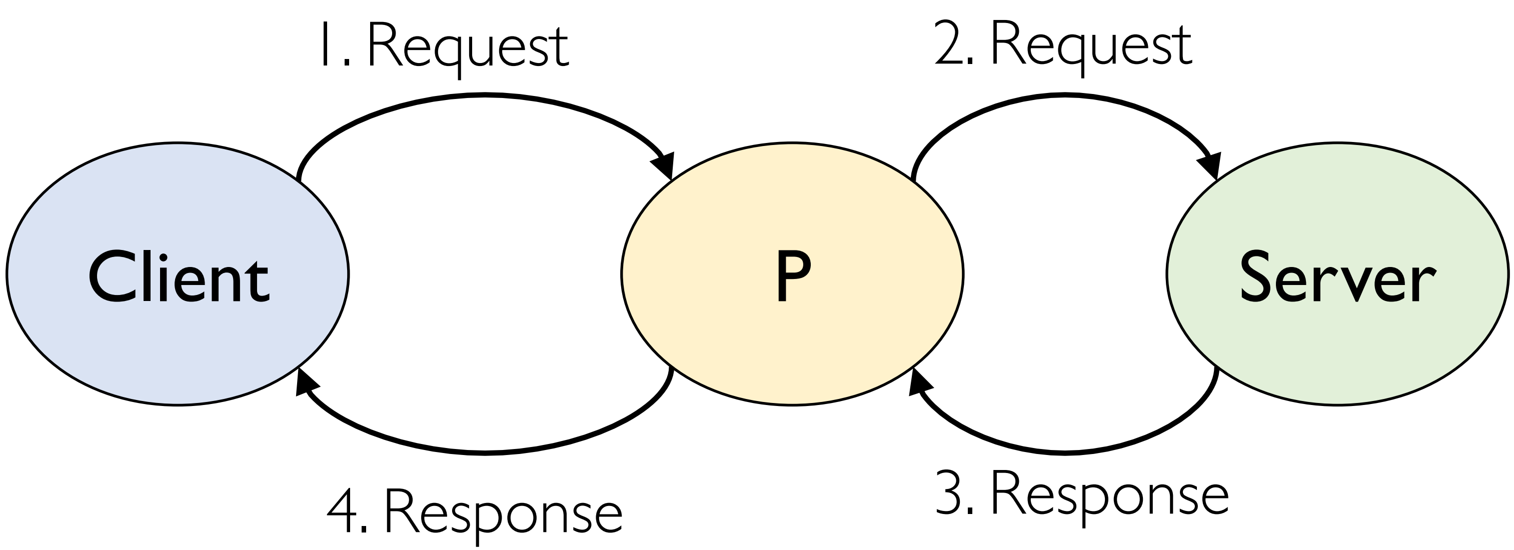
- Client-server



Clear difference in roles.

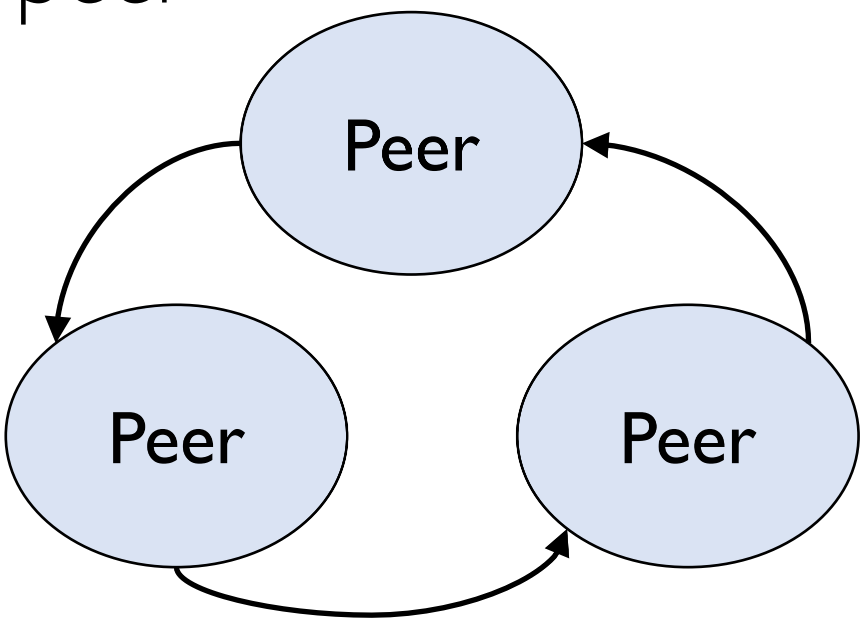
Relationship between processes

- Client-server



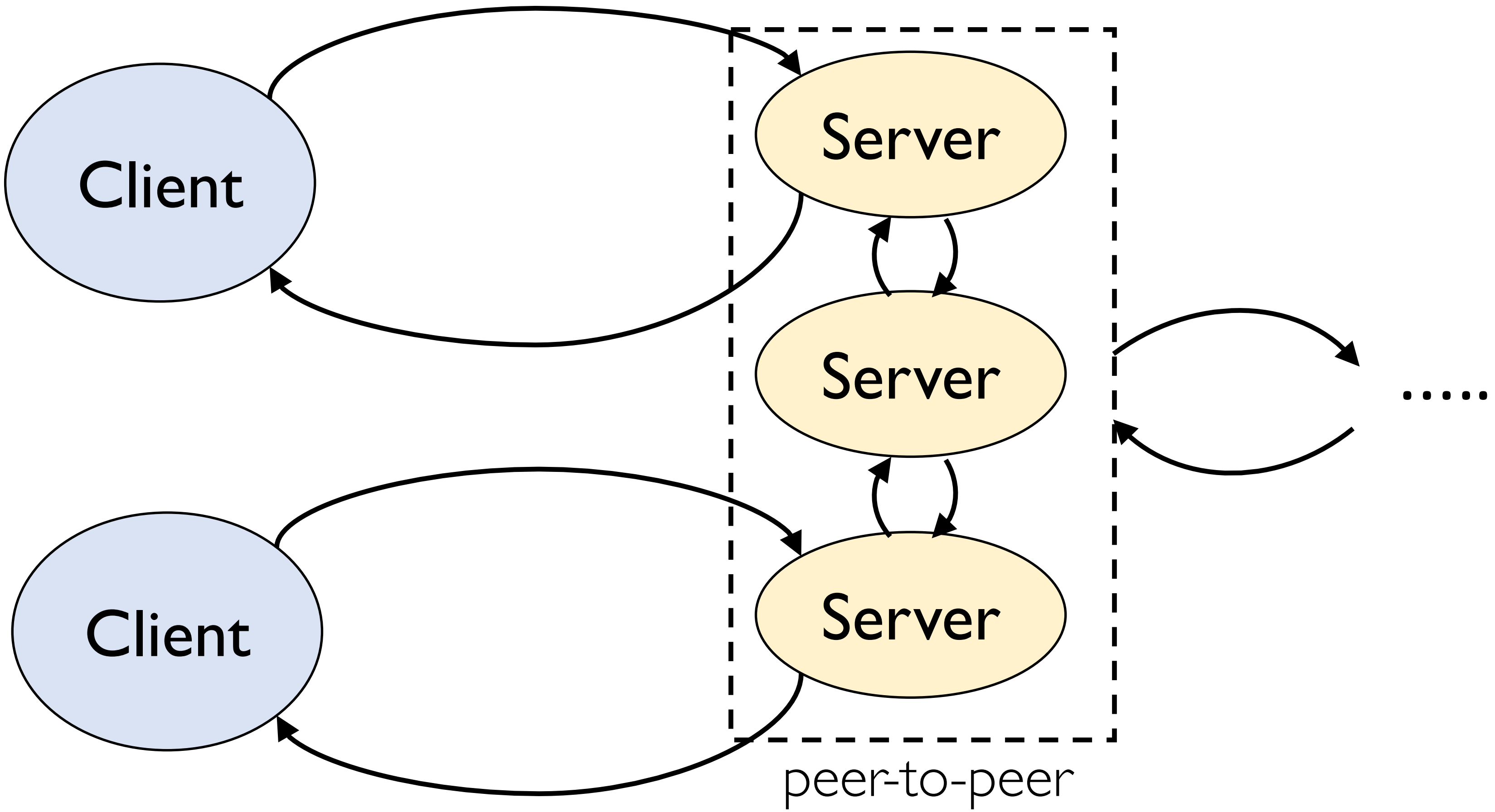
Relationship between processes

- Peer-to-peer



Similar roles.
Run the same program/algorithm.

Relationship between processes



Relationship between processes

- Two broad categories:
  - Client-server
  - Peer-to-peer

Distributed algorithm

- Algorithm on a single process
  - Sequence of steps taken to perform a computation.
  - *Steps are strictly sequential.*

- Distributed algorithm
  - Steps taken by each of the processes in the system (including transmission of messages).
  - *Different processes may execute their steps concurrently.*

Key aspects of a *distributed* system

- Processes must communicate with one another to coordinate actions. Communication time is variable.

- Different processes (on different computers) have different clocks!

- Processes and communication channels may fail.

Key aspects of a *distributed* system

- Processes must communicate with one another to coordinate actions. Communication time is variable.

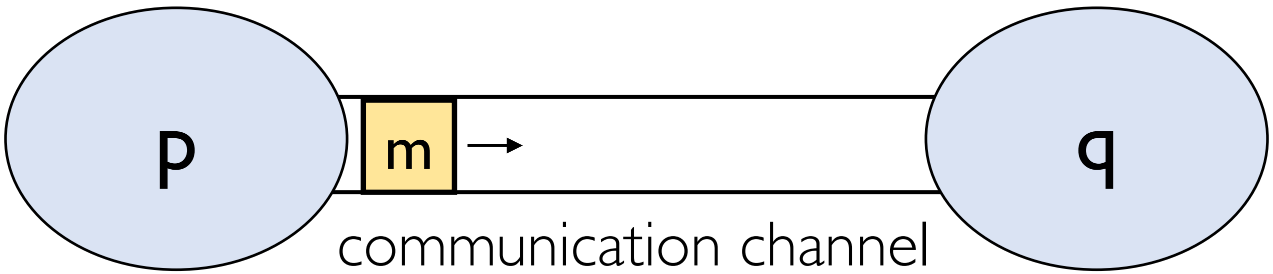
- Different processes (on different computers) have different clocks!

- Processes and communication channels may fail.

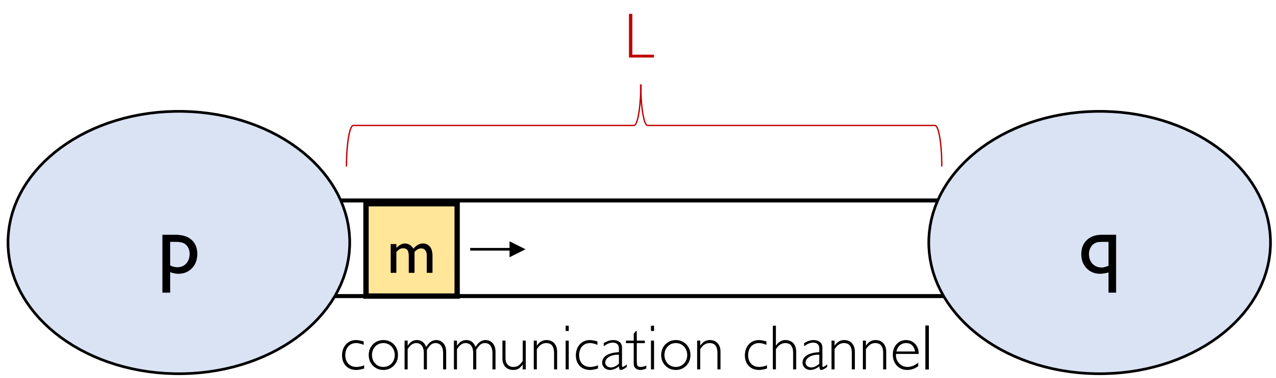
How processes communicate

- Directly using network sockets.
- Abstractions such as remote procedure calls, publish-subscribe systems, or distributed share memory.
- Differ with respect to how the message, the sender or the receiver is specified.

How processes communicate

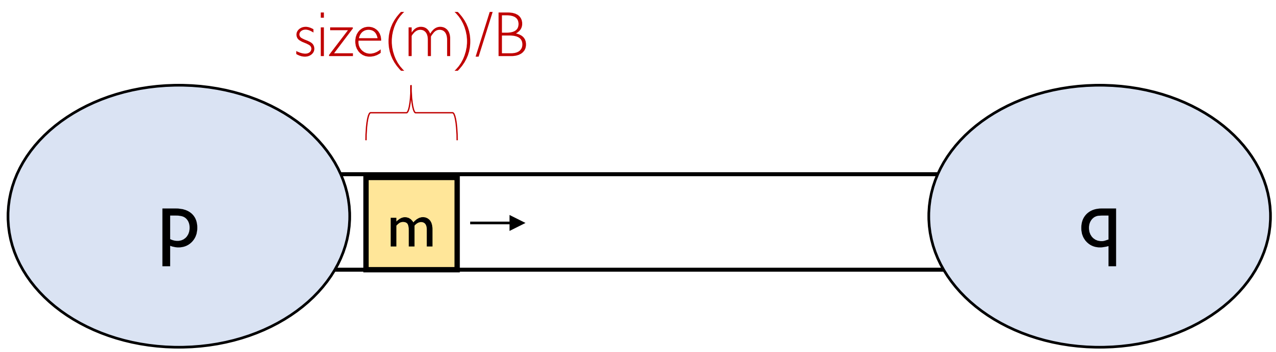


Communication channel properties



- Latency (L): Delay between the start of **m**'s transmission at **p** and the beginning of its receipt at **q**.
  - Time taken for a bit to propagate through network links.
  - Queuing that happens at intermediate hops.
  - Delay in getting to the network.
  - Overheads in the operating systems in sending and receiving messages.
  - …..

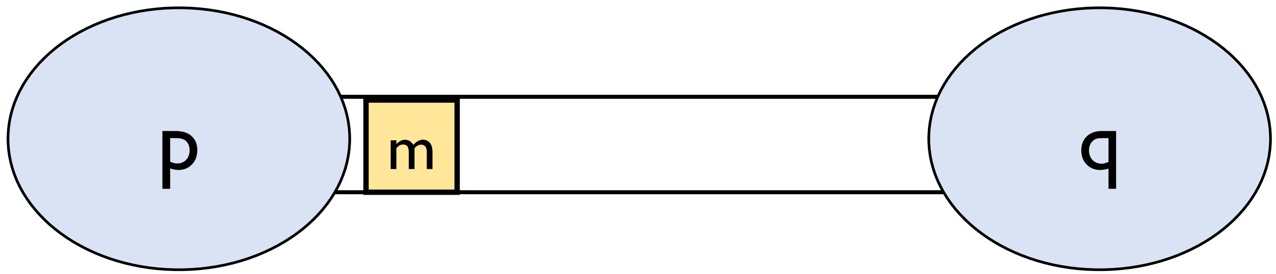
Communication channel properties



- Latency (L): Delay between the start of **m**'s transmission at **p** and the beginning of its receipt at **q**.

- Bandwidth (B): Total amount of information that can be transmitted over the channel per unit time.
  - Per-channel bandwidth reduces as multiple channels share common network links.

Communication channel properties



- Total time taken to pass a message is governed by latency and bandwidth of the channel.
- Both latency and available bandwidth may vary over time.

Key aspects of a *distributed* system

- Processes must communicate with one another to coordinate actions. Communication time is variable.

- Different processes (on different computers) have different clocks!

- Processes and communication channels may fail.

Differing clocks

- Each computer in a distributed system has its own internal clock.

- Local clock of different processes show different time values.

- Clocks *drift* from perfect times at different rates.

Key aspects of a *distributed* system

- Processes must communicate with one another to coordinate actions. Communication time is variable.

- Different processes (on different computers) have different clocks!

- Processes and communication channels may fail.

Two ways to model

- Synchronous distributed systems:
  - Known upper and lower bounds on time taken by each step in a process.
  - Known bounds on message passing delays.
  - Known bounds on clock drift rates.
- Asynchronous distributed systems:
  - No bounds on process execution speeds.
  - No bounds on message passing delays.
  - No bounds on clock drift rates.

Synchronous and Asynchronous

- Most real-world systems are asynchronous.
  - Bounds can be estimated, but hard to guarantee.
  - Assuming system is synchronous can still be useful.

- Possible to build a synchronous system.

Key aspects of a *distributed* system

- Processes must communicate with one another to coordinate actions. Communication time is variable.

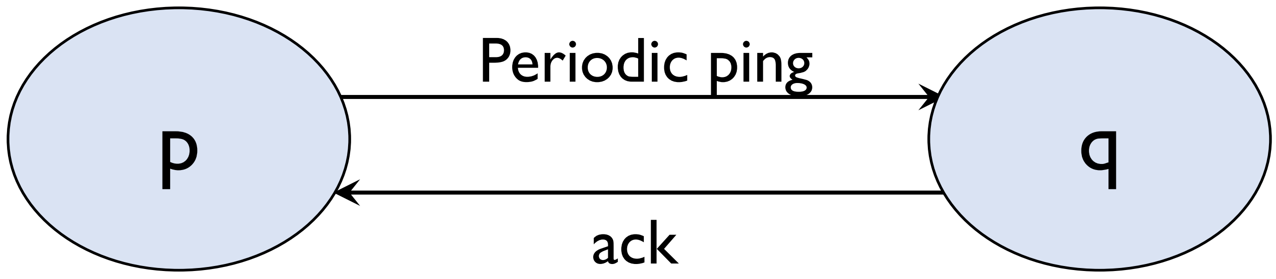
- Different processes (on different computers) have different clocks!

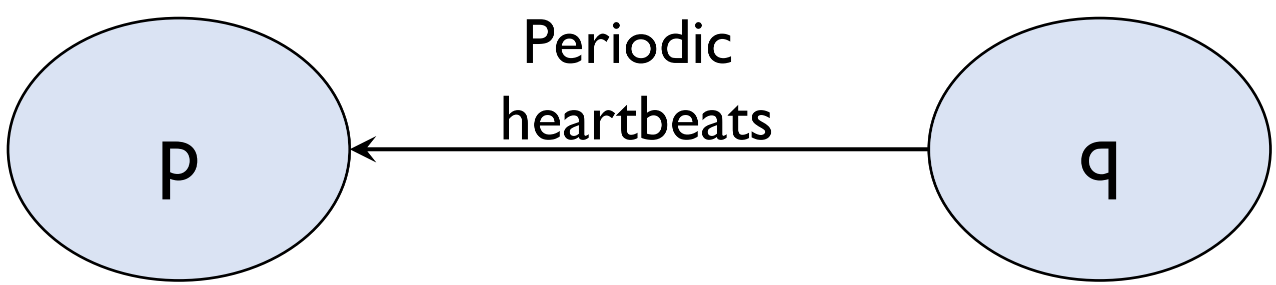
- Processes and communication channels may fail.

Types of failure

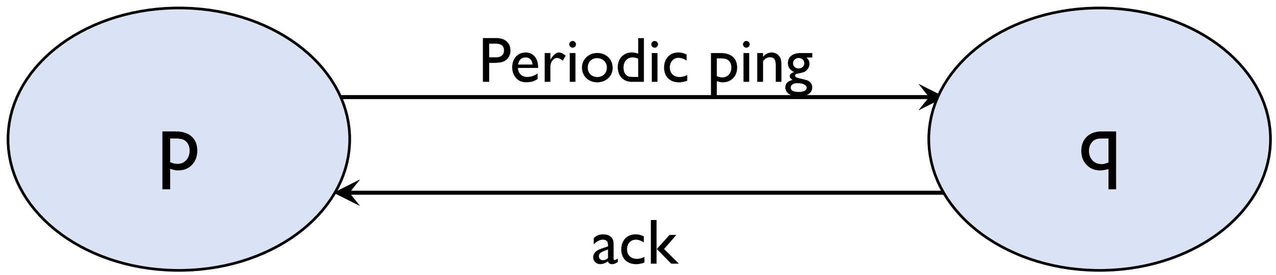
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.

How to detect a crashed process?





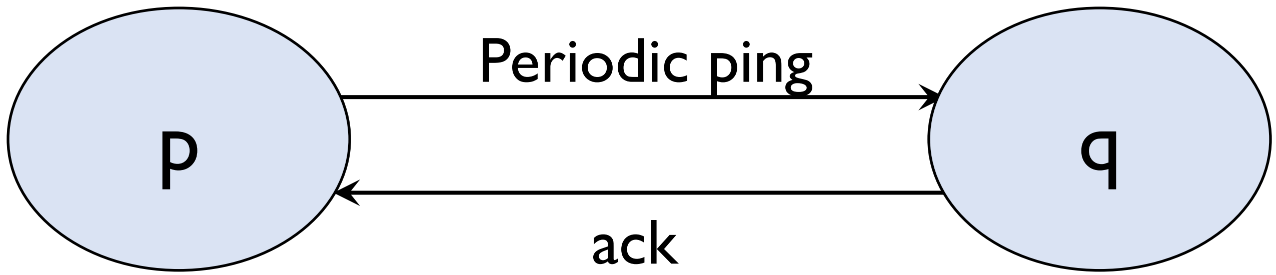
How to detect a crashed process?



Δ_1 time elapsed after sending ping, and no ack.

If synchronous, Δ_1 = 2(max network delay)
If asynchronous, Δ_1 = (max observed round trip time)

How to detect a crashed process?



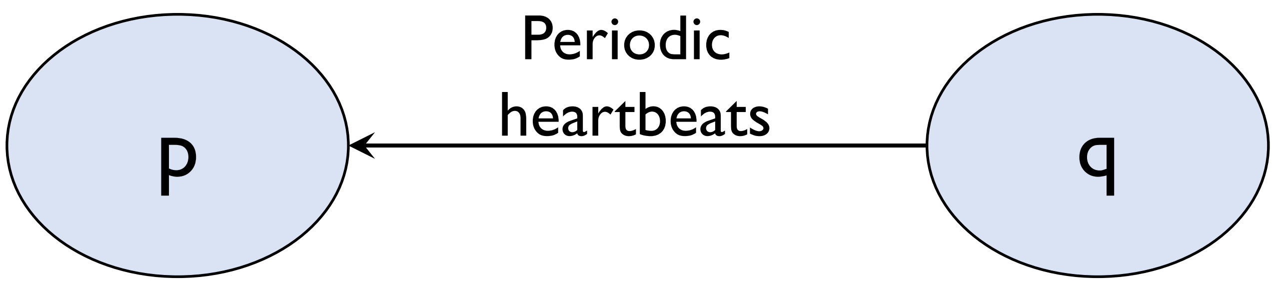
Pings are sent every T seconds.

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

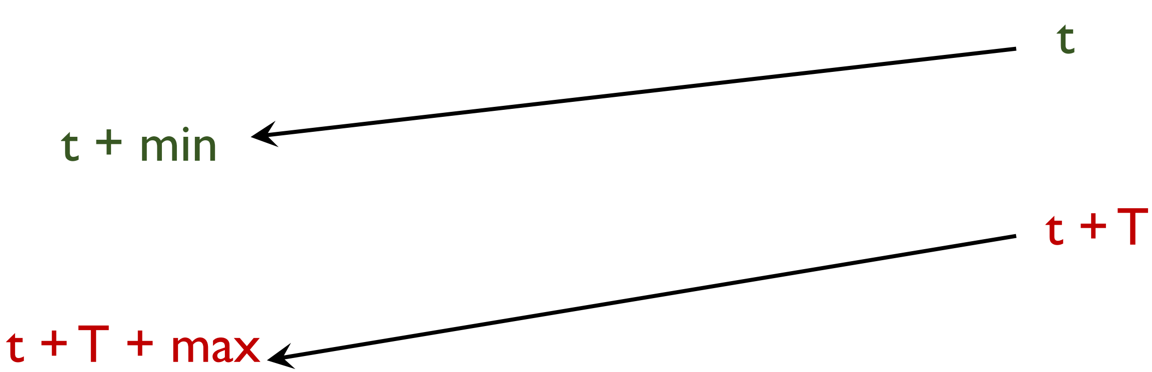
If synchronous, Δ_1 = 2(max network delay)

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

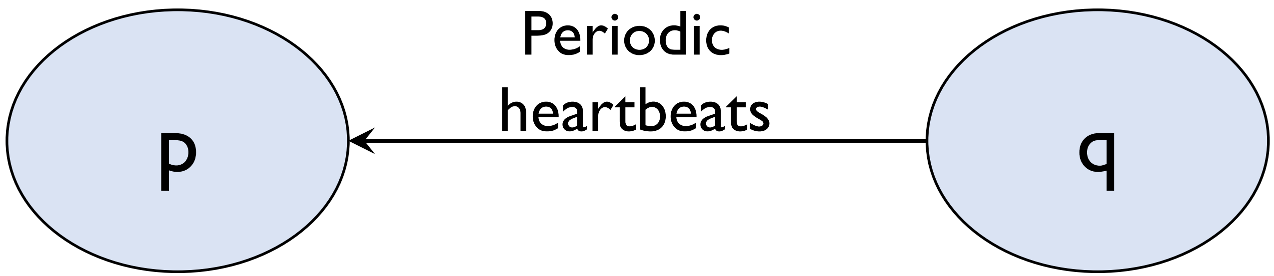
How to detect a crashed process?



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



How to detect a crashed process?



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

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

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
  - Ping-ack:

Metrics for failure detection

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

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for last ping from p to reach q)
  - Heartbeat: $\Delta + T + \Delta_2$ (where Δ is time taken for last message 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 last ping from p to reach q)
  - Heartbeat: $\Delta + T + \Delta_2$ (where Δ is time taken for last message from q to reach p)

Try deriving these before next class!

Metrics for failure detection

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

Metrics for failure detection

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

Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where Δ is time taken for last ping from p to reach q)
  - Heartbeat: $\Delta + T + \Delta_2$ (where Δ is time taken for last message 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 last ping from p to reach q)
  - Heartbeat: $\Delta + T + \Delta_2$ (where Δ is time taken for last message 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 last ping from p to reach q)
  - Heartbeat: $\Delta + T + \Delta_2$ (where Δ is time taken for last message from q to reach p)

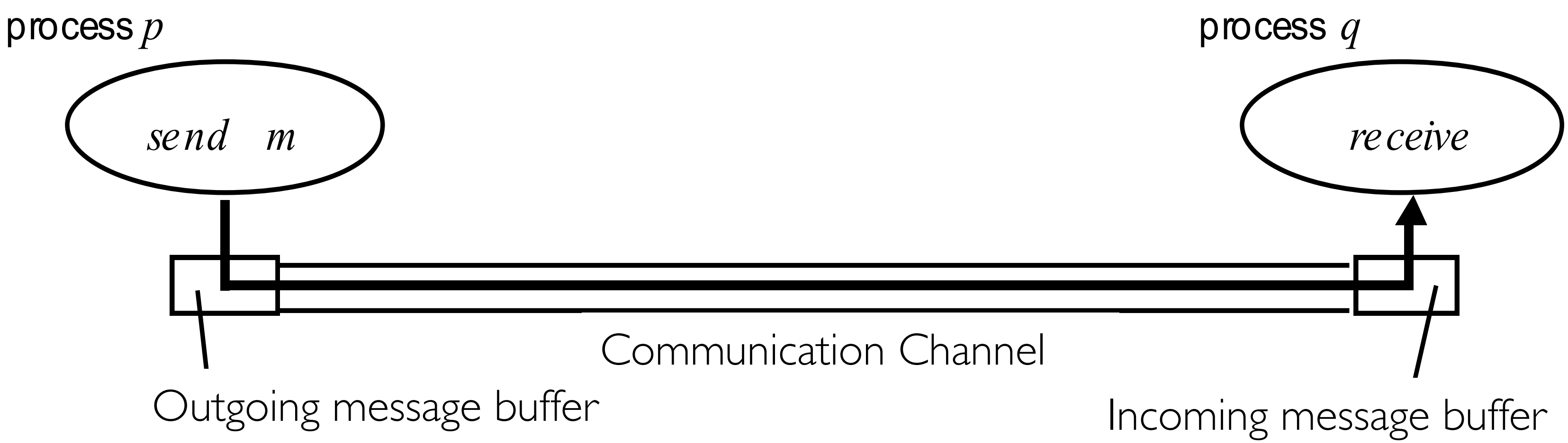
- Bandwidth usage:
  - Ping-ack: 2 messages every T units
  - Heartbeat: 1 message every T units.

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

Types of failure

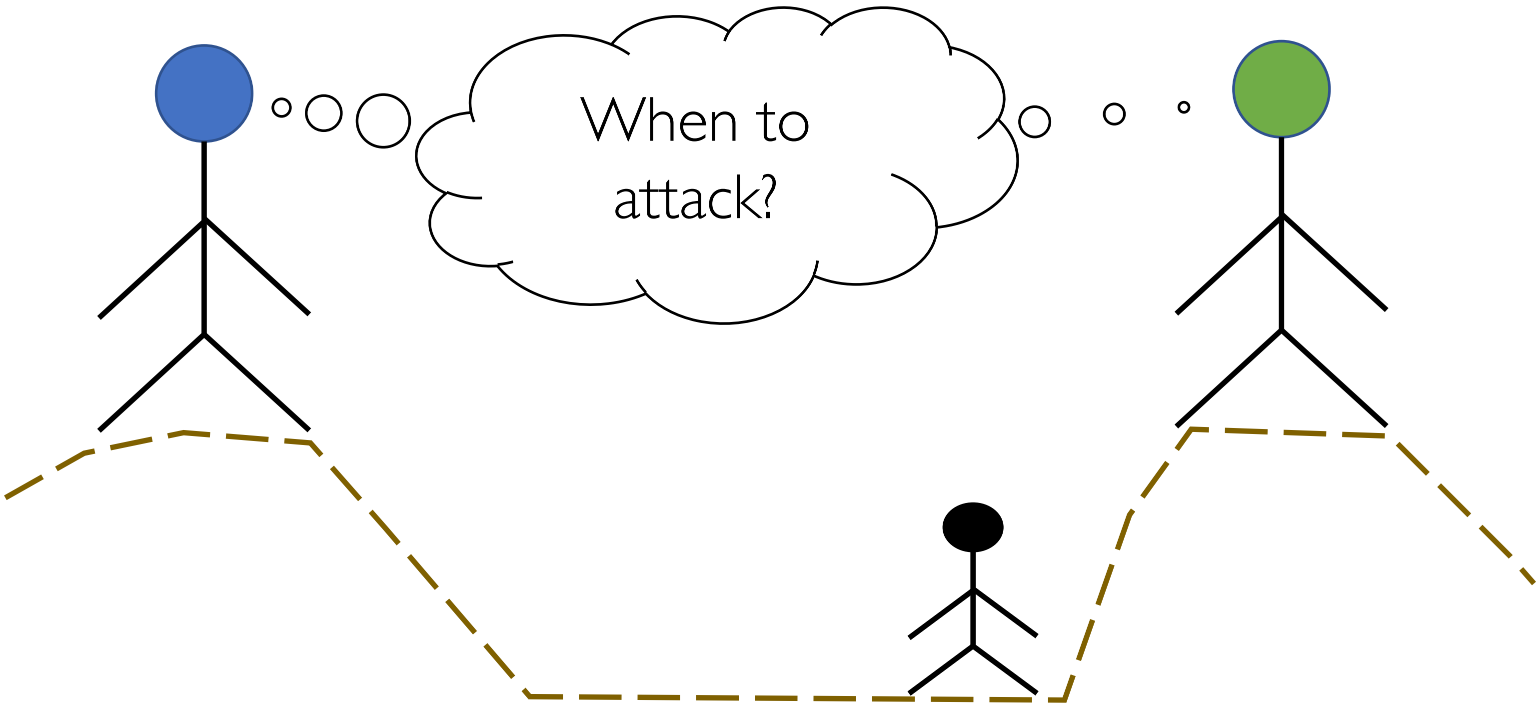
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.
  - **Fail-stop**: if other processes can certainly detect the crash.
  - **Communication omission**: a message sent by process was not received by another.

Communication Omission



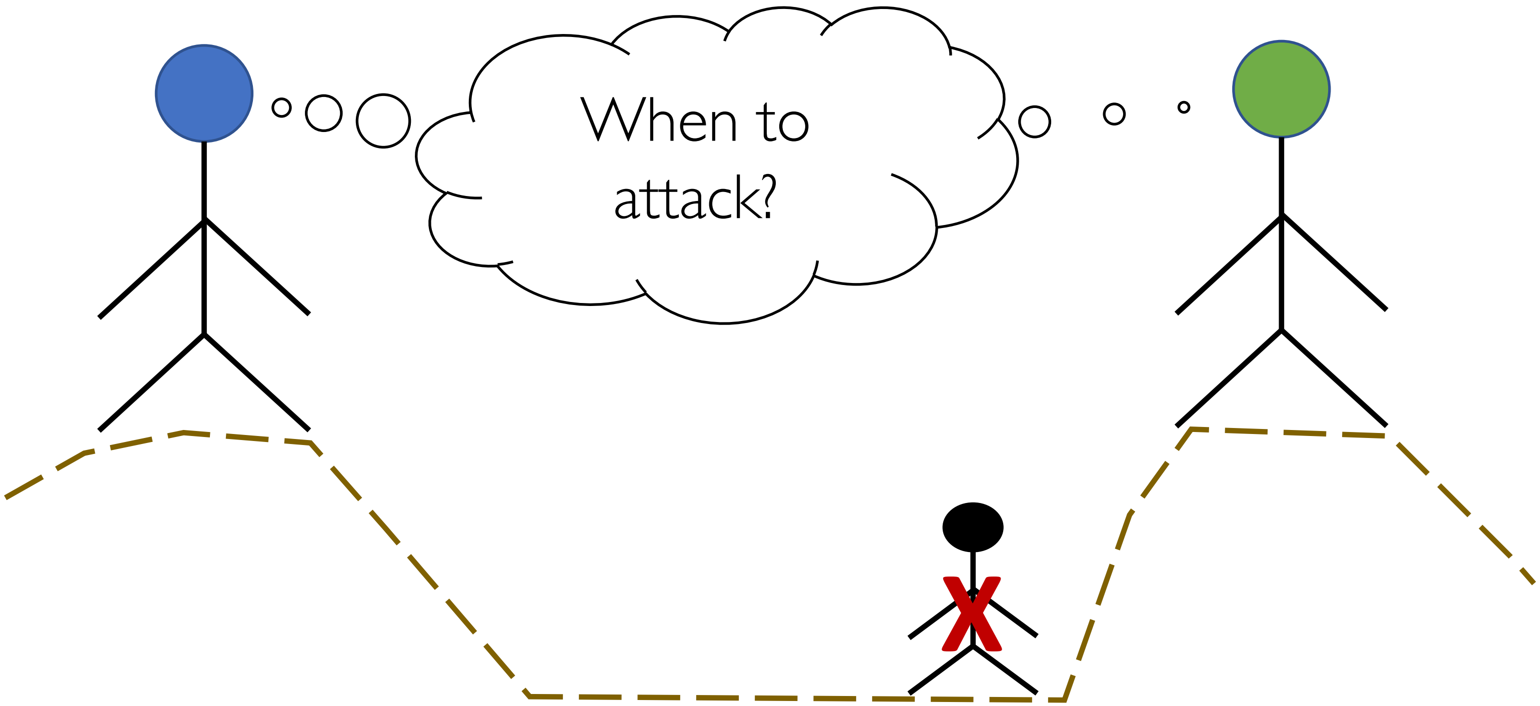
- Channel Omission: omitted by channel
- Send omission: process completes 'send' operation, but message does not reach its outgoing message buffer.
- Receive omission: message reaches the incoming message buffer, but not received by the process.

Two Generals Problem



How do the two general coordinate their time for attack?

Two Generals Problem



What if their messengers may get shot on the way?

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.

Summary

- Relationship between processes
  - Client-server and peer-to-peer
- Sources of uncertainty
  - Communication time, clock drift rates
- Synchronous vs asynchronous models.
- Types of failures: omission, arbitrary, timing
- Detecting failed a process.