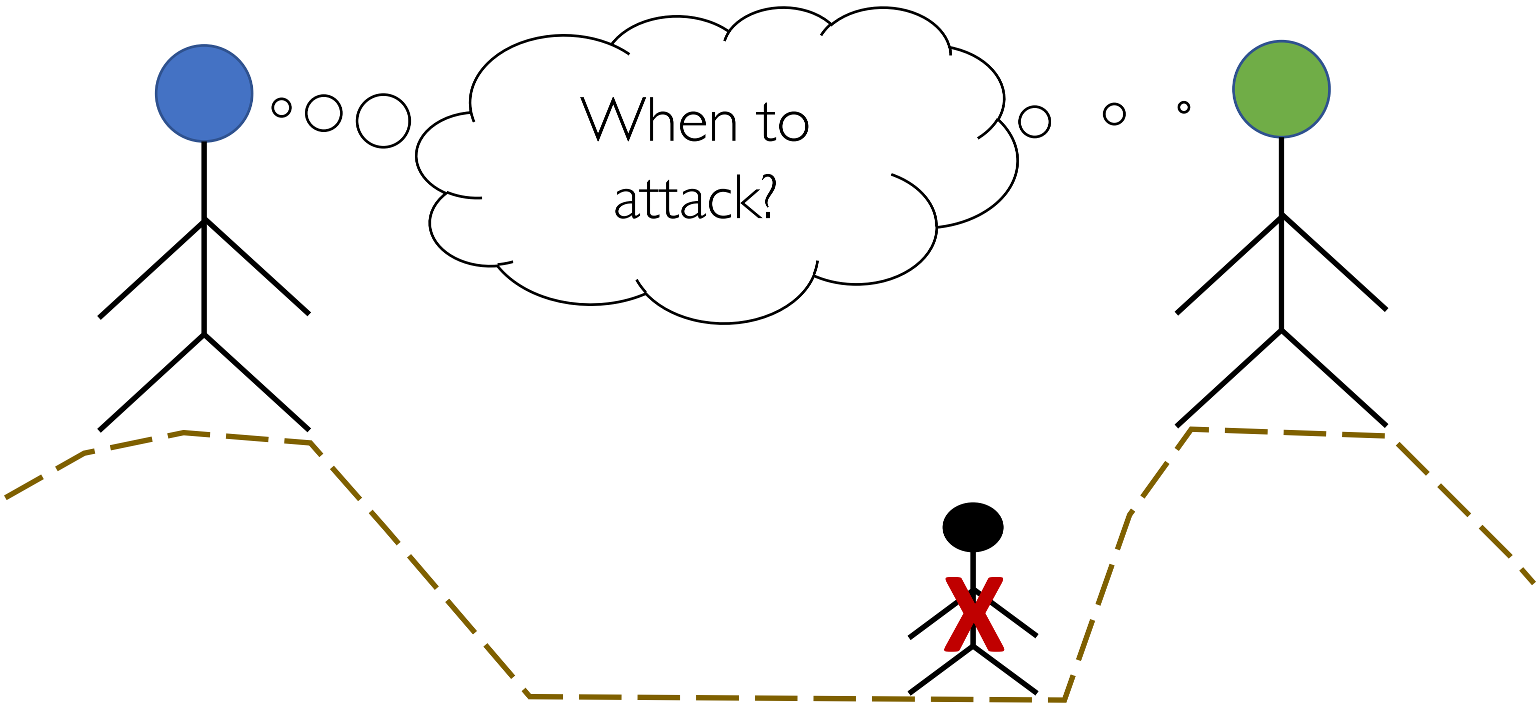
# Distributed Systems

## CS425/ECE428

*Instructor: Radhika Mittal*

# While we wait…..




Two generals must agree on a time to attack the enemy base.
They can communicate with each-other by sending messengers.
But, a messenger may get killed by the enemy along the way.
Thankfully, they have unlimited no. of messengers at their disposals.
**How can the two generals agree on a time to attack?**

# Logistics Related

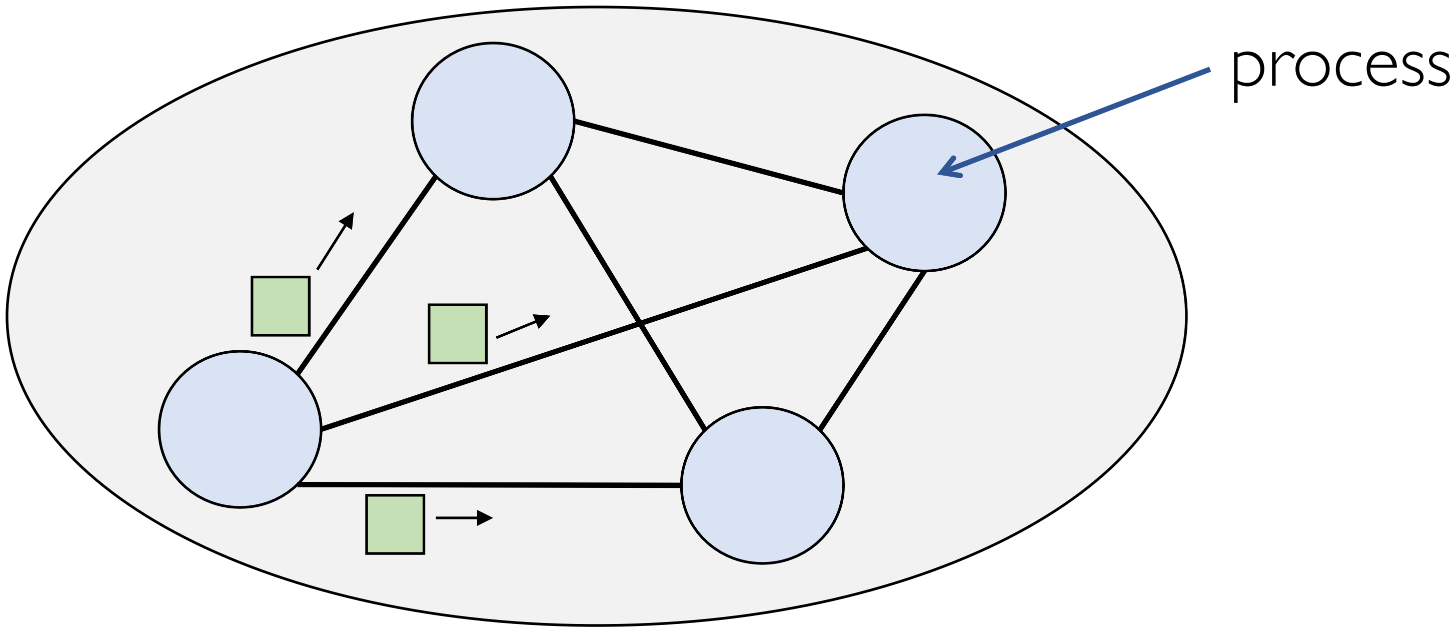
- Sign-up forms for VM clusters is available on CampusWire.
  - Please fill it up by Jan 24th, Monday, 11:59pm.

- MP0 has been released! Due in two weeks.

# 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

# 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.

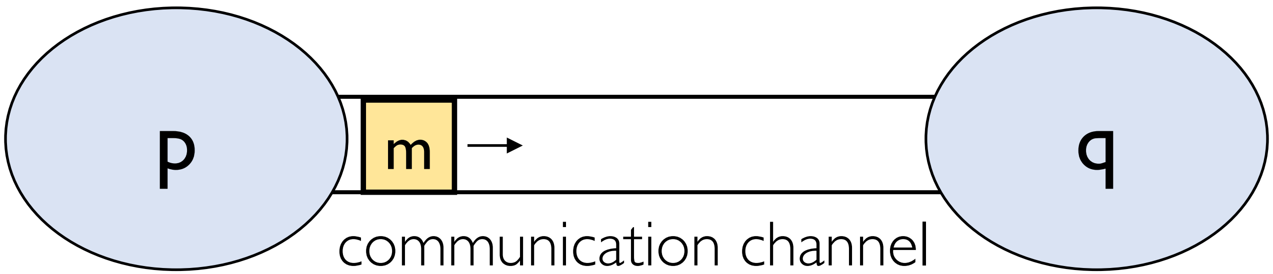
# How processes communicate

- Directly using network sockets.

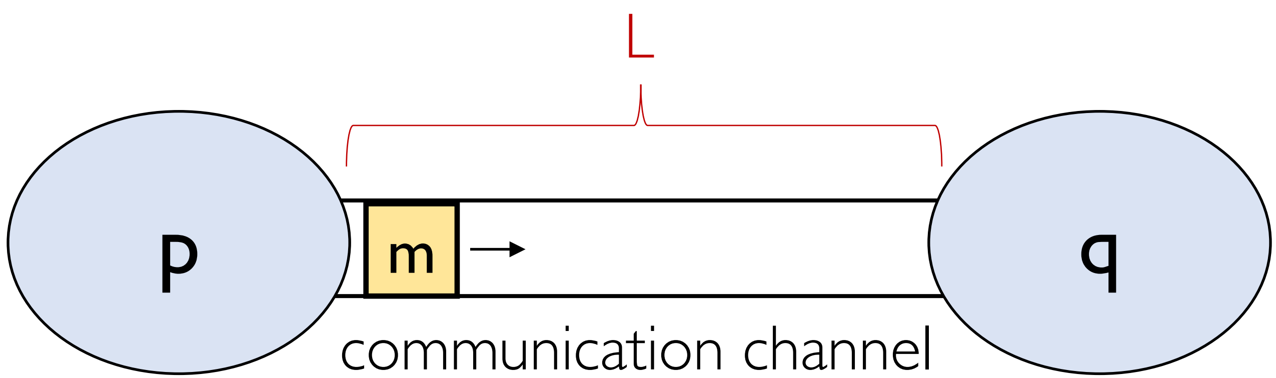
- Abstractions such as remote procedure calls, publish-subscribe systems, or distributed share memory.

- Differ with respect to how the message, the sender or the receiver is specified.

# How processes communicate

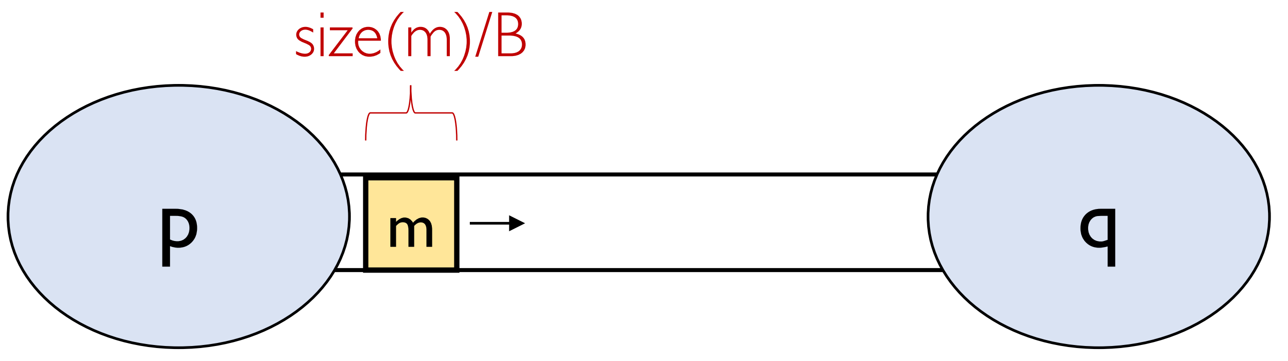


# Communication channel properties




- 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.
  - Overheads in the operating systems in sending and receiving messages.
  - …..

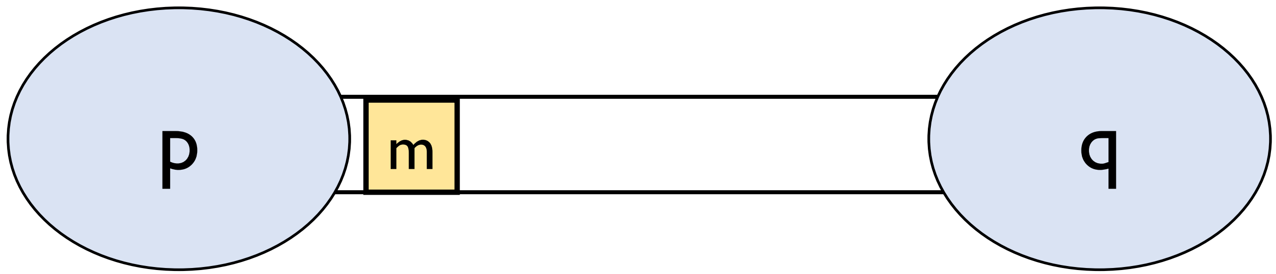
# Communication channel properties




- 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.

# 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.

- *Sometimes useful to measure "bandwidth usage" of a system as amount of data being sent between processes per unit 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.

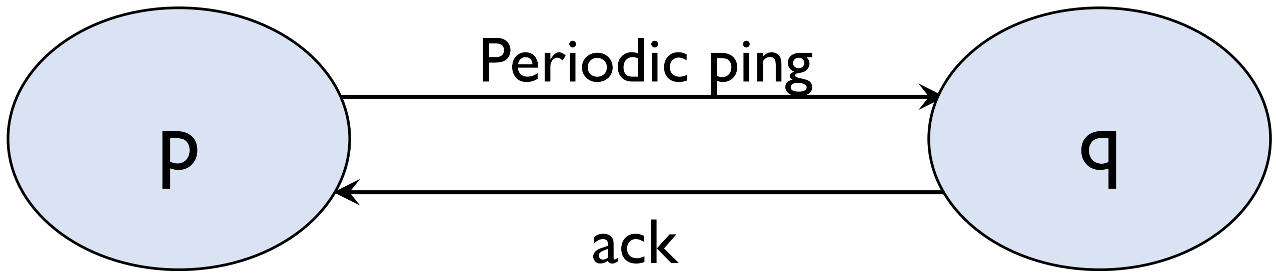
- Different processes (on different computers) have different clocks!

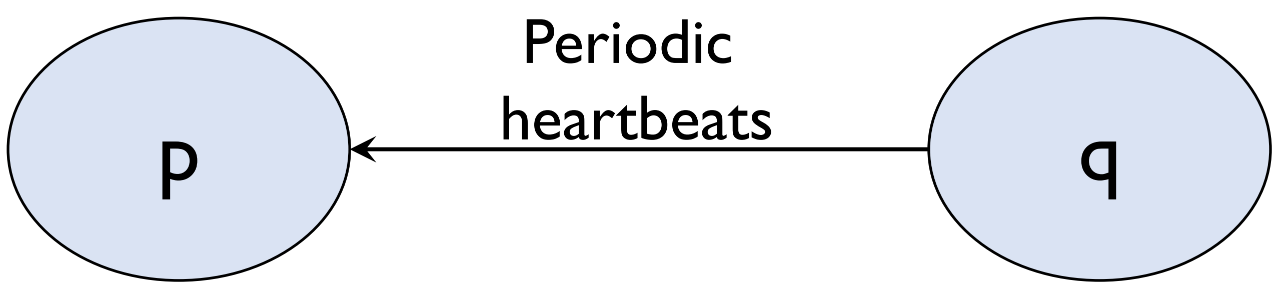
- Processes and communication channels may fail.

# Types of failure

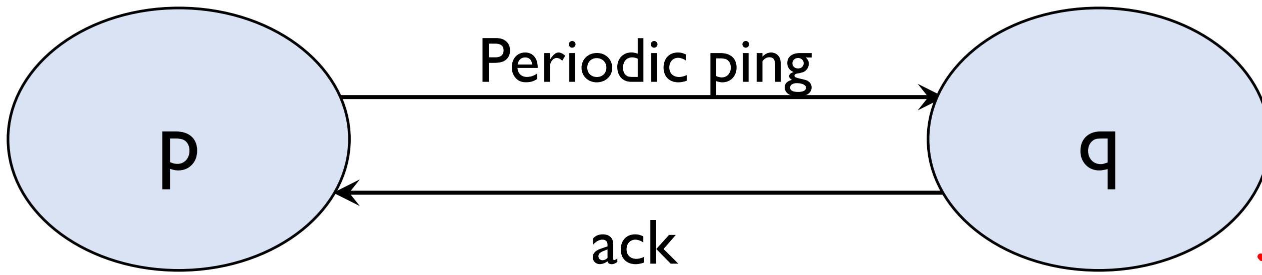
- **Omission:** when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.

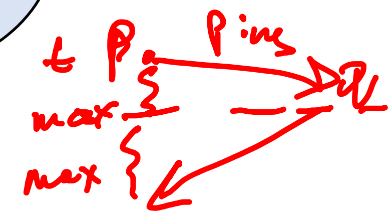
# How to detect a crashed process?



# How to detect a crashed process?




p sends pings to q every T seconds.

$\Delta_1$ is the *timeout* value at p.

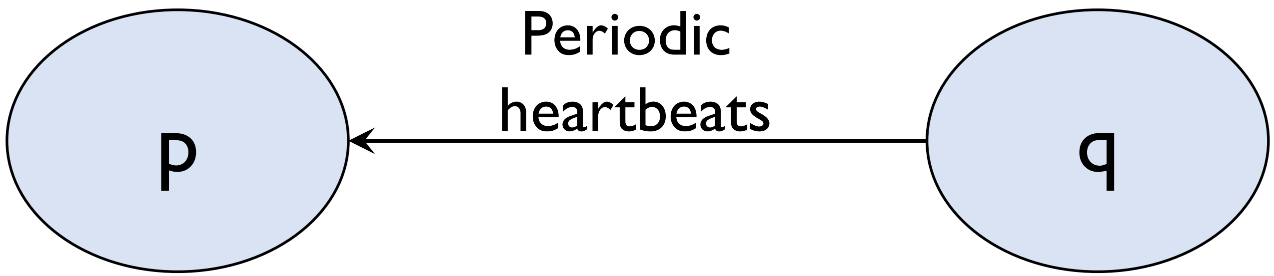
If $\Delta_1$ time elapsed after sending ping, and no ack, report q crashed.

If synchronous, $\Delta_1$ =

If asynchronous, $\Delta_1$ =

>1

# How to detect a crashed process?




q sends heartbeats to p every T seconds.

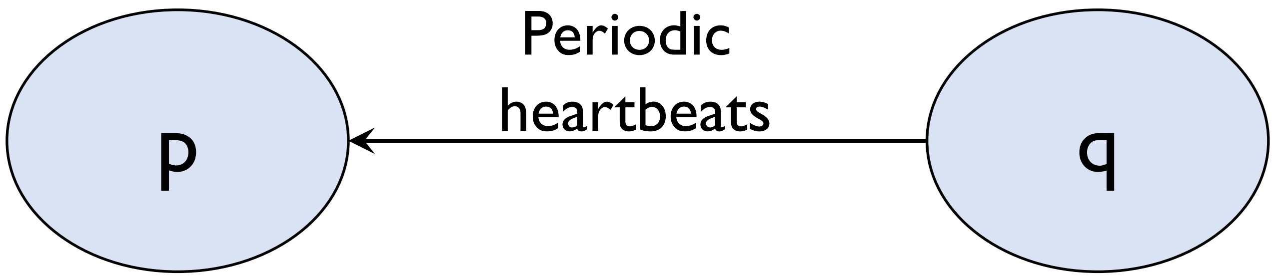
$(T + \Delta_2)$ is the *timeout* value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.

If synchronous, $\Delta_2 =$

If asynchronous, $\Delta_2 =$

# How to detect a crashed process?



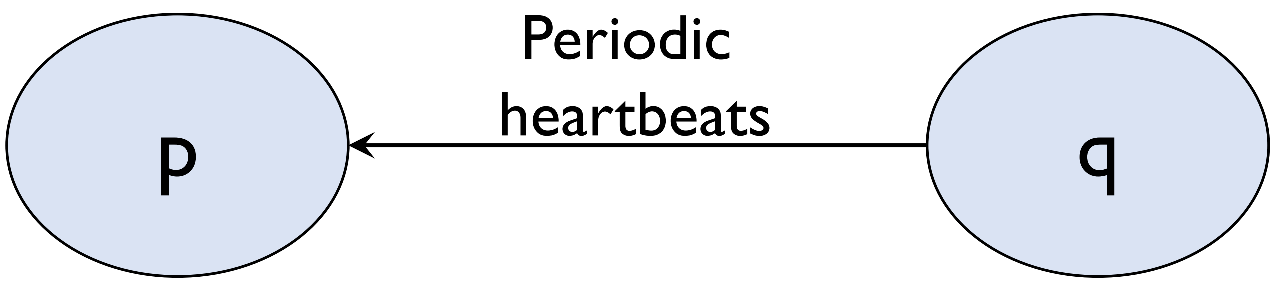

q sends heartbeats to p every T seconds.

$(T + \Delta_2)$ is the *timeout* value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.

If synchronous, $\Delta_2 =$

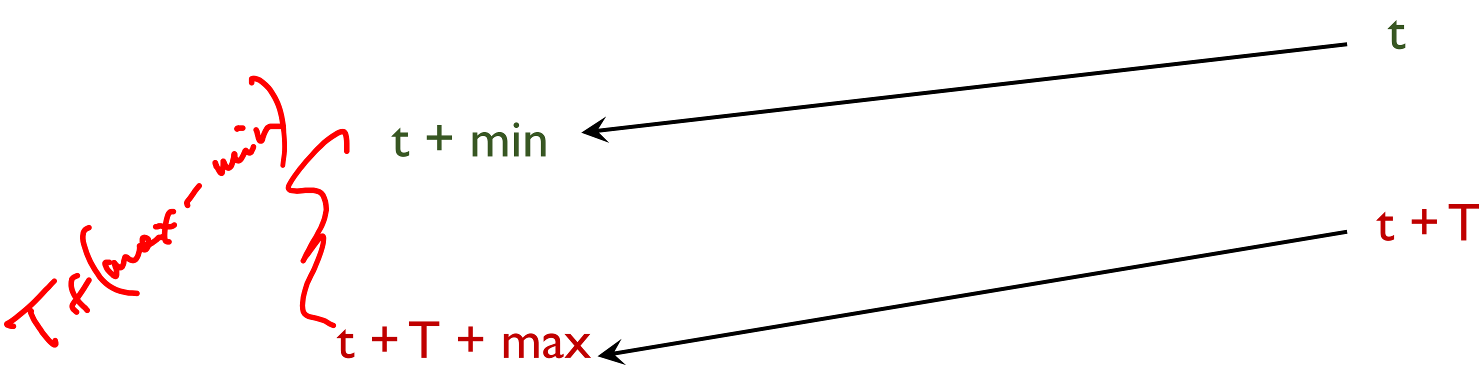
# How to detect a crashed process?



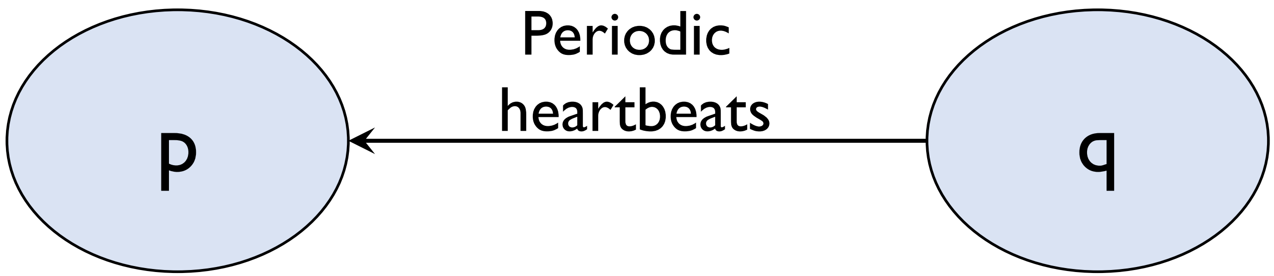

q sends heartbeats to p every T seconds.

$(T + \Delta_2)$ is the *timeout* value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.



# How to detect a crashed process?




q sends heartbeats to p every T seconds.

$(T + \Delta_2)$ is the *timeout* value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.

If synchronous, $\Delta_2$ = max network delay – min network delay

If asynchronous, $\Delta_2$ = 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*
  - Impossible to achieve both completeness and accuracy.
  - Can we have an accurate but incomplete algorithm?
    - *Never report failure.*

# Metrics for failure detection

Try deriving these before next class!

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

# Metrics for failure detection

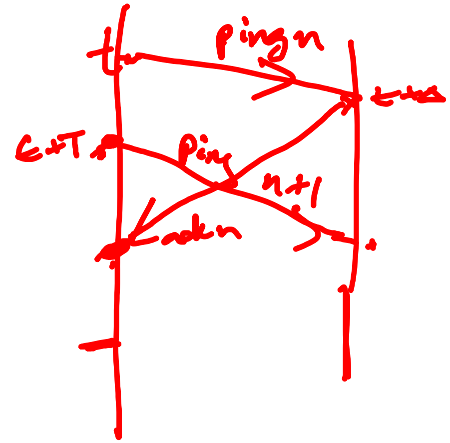
- Worst case failure detection time
  - After a process crashes, how long does it take for the other process to detect the crash in the worst case?

# Metrics for failure detection

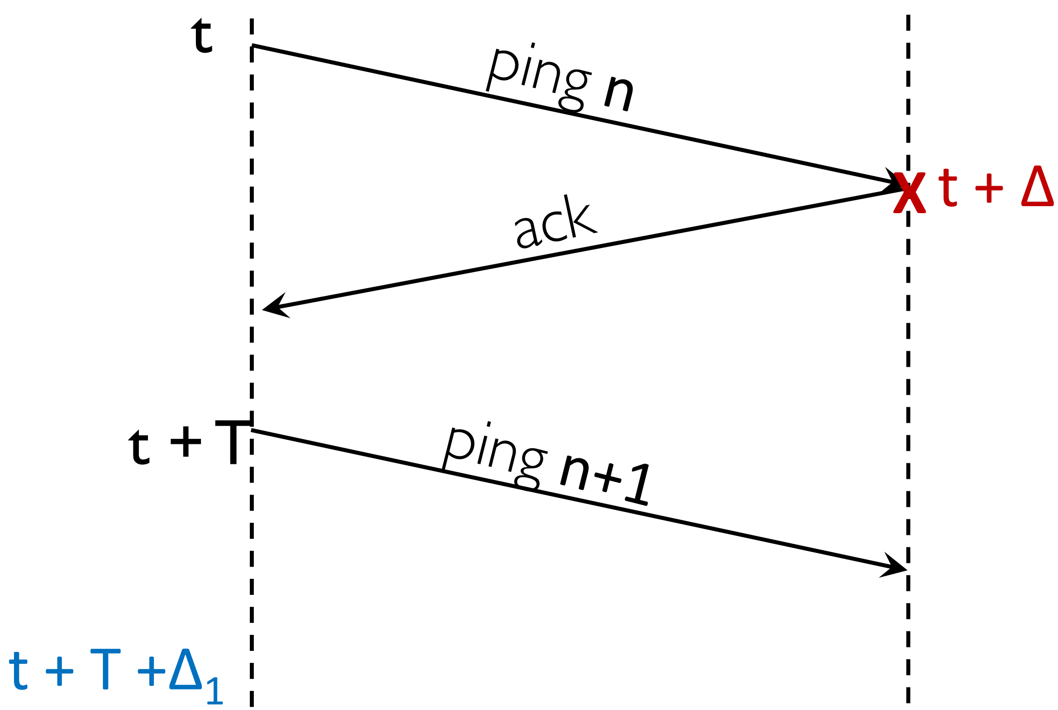
- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ where $\Delta$ is time taken for the last ping from p to reach q before q crashed. T is the time period for pings, and $\Delta_1$ is timeout value.

Try deriving this!

# Metrics for failure detection




- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ where $\Delta$ is time taken for the last ping from p to reach q before q crashed. T is the time period for pings, and $\Delta_1$ 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* $\Delta$ for a *synchronous system?*
*A:* min network delay

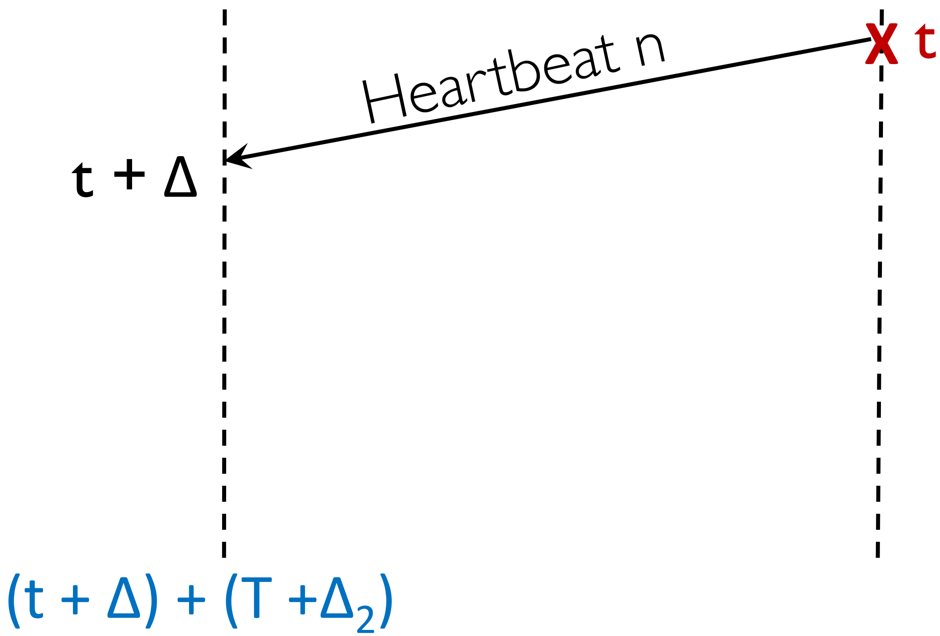
# Metrics for failure detection

- Worst case failure detection time
  - Heartbeat: $T + \Delta_2 + \Delta$ where $\Delta$ 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.

Try deriving this!

# Metrics for failure detection

- Worst case failure detection time
  - Heartbeat: $T + \Delta_2 + \Delta$ where $\Delta$ 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* $\Delta$ in a *synchronous system?*
*A:* max network delay

# Metrics for failure detection

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

# Metrics for failure detection

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

# MP0: Event Logging

- https://courses.grainger.illinois.edu/cs425/sp202~~1~~2/mps/mp0.html
- Lead TA: Sanchit Vohra

- 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

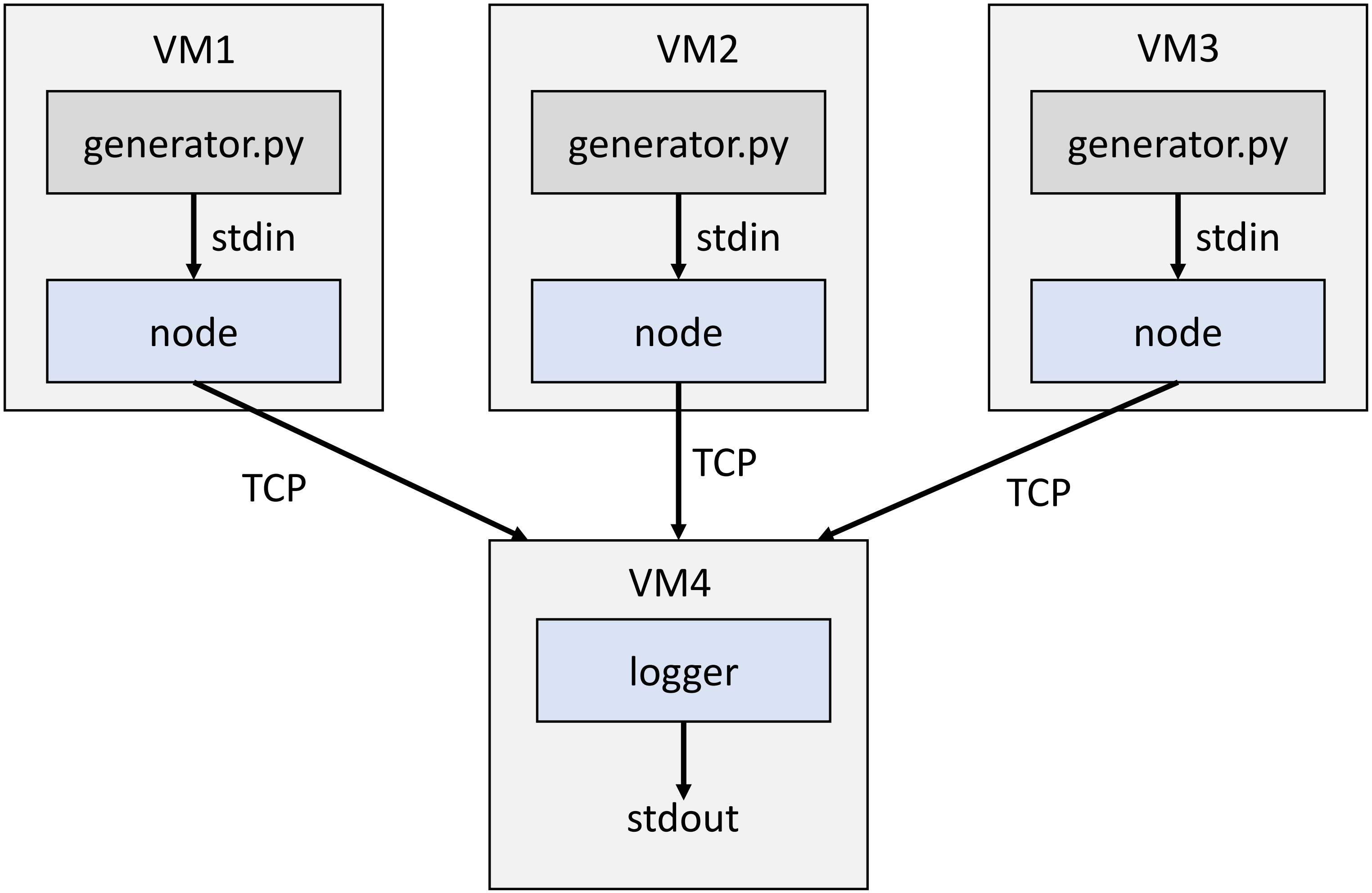
generator.py

Timestamp

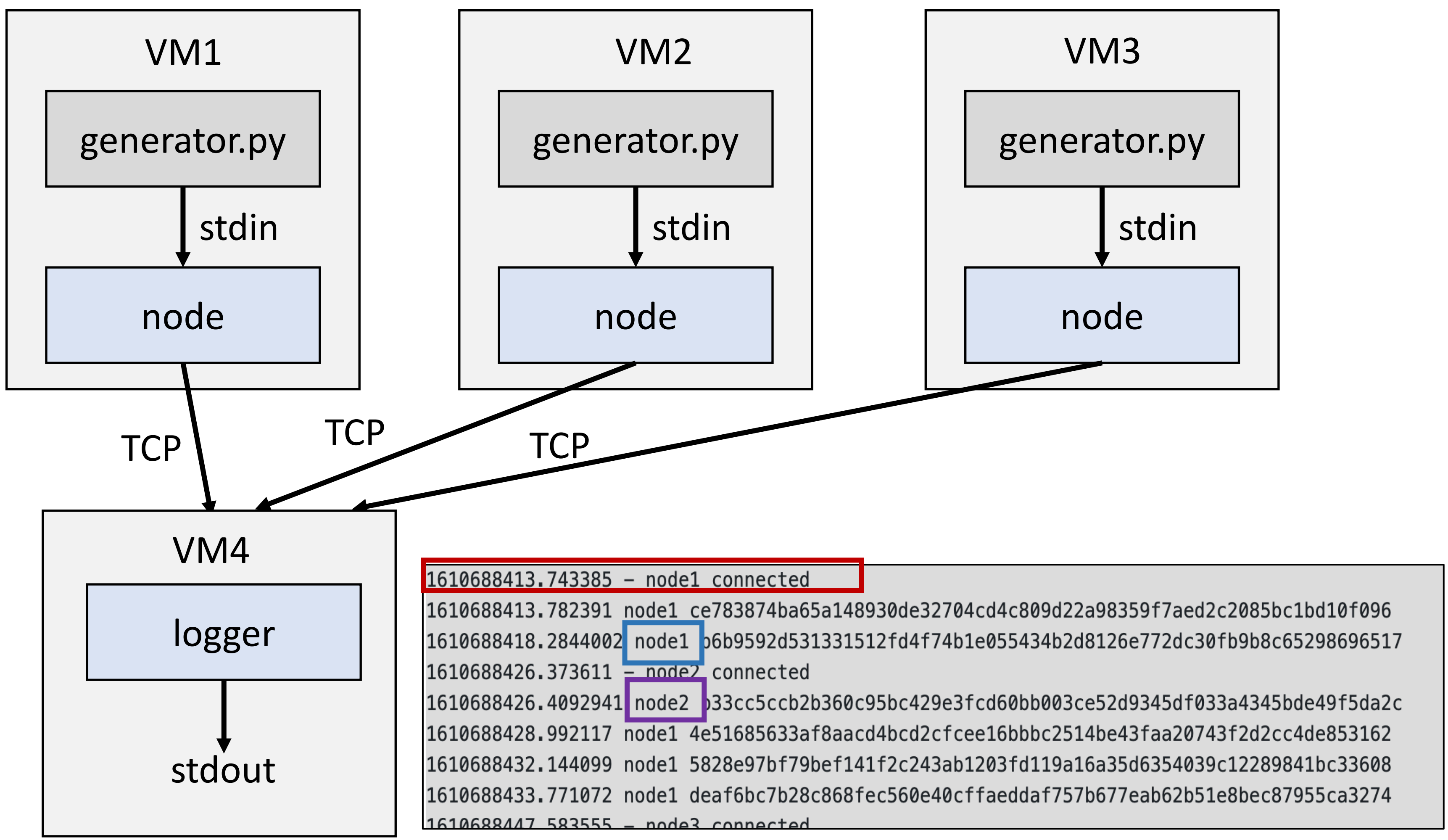
Event name (random)

```
% 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 3, 11:59pm
  - Late policy: Can submit up to 50hrs late with 2% penalty per hour.
- 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.