# Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport

Yuncong Zhang

April 23, 2020

# Outline

Total Ordering of Events

Mutual Exclusion

Physical Clock

# Events in Distributed System

What does it mean by "*a* happens before *b*" in distributed system?

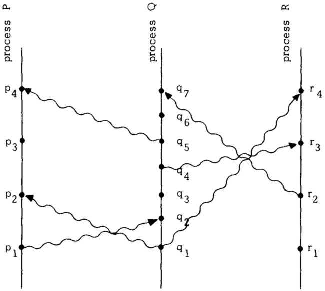



Figure: Events and Messages in Processes

# Events in Distributed System

What does it mean by "*a* happens before *b*" in distributed system?

- In one process, **earlier** events happens before **later** events

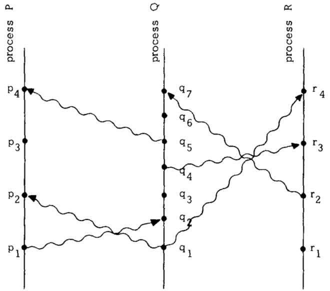



Figure: Events and Messages in Processes

# Events in Distributed System

What does it mean by "*a* happens before *b*" in distributed system?

- In one process, **earlier** events happens before **later** events
- **Sending message** happens before **receiving message**

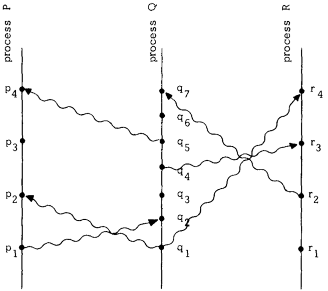



Figure: Events and Messages in Processes

# Events in Distributed System

What does it mean by "*a* happens before *b*" in distributed system?

- In one process, **earlier** events happens before **later** events
- **Sending message** happens before **receiving message**
- If $a$ happens before $b$, and $b$ happens before $c$, then $a$ happens before $c$

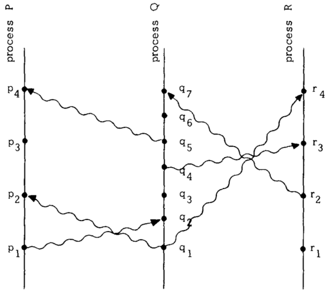



Figure: Events and Messages in Processes

# Partial Ordering of Events

Denote by $a \to b$ if $a$ happens before $b$.

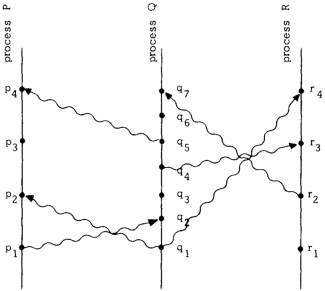



Figure: Events and Messages in Processes

# Partial Ordering of Events

Denote by $a \to b$ if $a$ happens before $b$.

- "$\to$" defines a partial order.

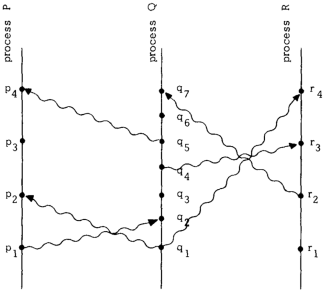



Figure: Events and Messages in Processes

# Partial Ordering of Events

Denote by $a \rightarrow b$ if $a$ happens before $b$.

- "$\rightarrow$" defines a partial order.
- If $a \nrightarrow b$ and $b \nrightarrow a$ then $a$ and $b$ are *concurrent*

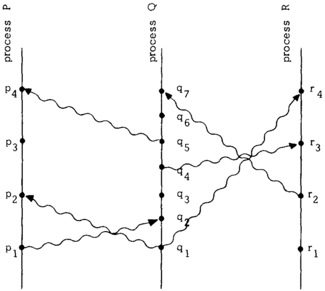



Figure: Events and Messages in Processes

# Partial Ordering of Events

Denote by $a \to b$ if $a$ happens before $b$.

- "$\to$" defines a partial order.
- If $a \nrightarrow b$ and $b \nrightarrow a$ then $a$ and $b$ are *concurrent*
- $a \to b$ is equivalent to saying one can go from $a$ to $b$ in the diagram by moving forward in time along process and message lines.

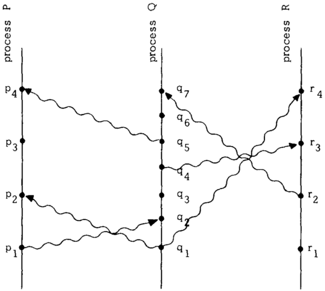



Figure: Events and Messages in Processes

# Partial Ordering of Events

Example: $p_1 \rightarrow r_4$

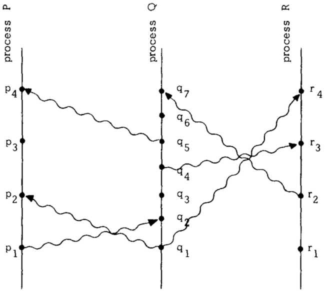



Figure: Events and Messages in Processes

# Partial Ordering of Events

Example: $p_1 \rightarrow r_4$

- $p_1 \rightarrow q_2$

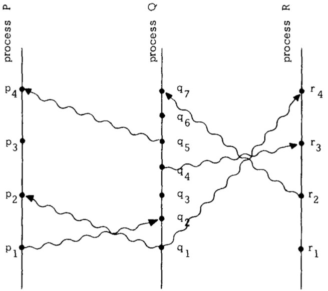



Figure: Events and Messages in Processes

# Partial Ordering of Events

Example: $p_1 \rightarrow r_4$

- $p_1 \rightarrow q_2$
- $q_2 \rightarrow q_4$

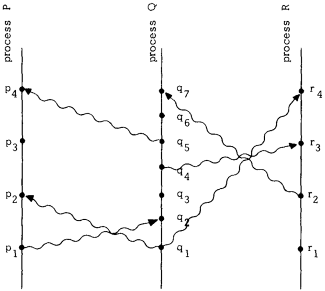



Figure: Events and Messages in Processes

# Partial Ordering of Events

Example: $p_1 \rightarrow r_4$

- $p_1 \rightarrow q_2$
- $q_2 \rightarrow q_4$
- $q_4 \rightarrow r_3$

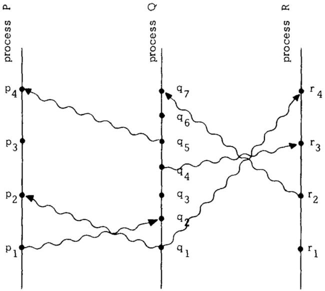



Figure: Events and Messages in Processes

# Partial Ordering of Events

Example: $p_1 \rightarrow r_4$

- $p_1 \rightarrow q_2$
- $q_2 \rightarrow q_4$
- $q_4 \rightarrow r_3$
- $r_3 \rightarrow r_4$

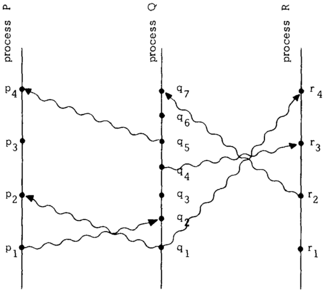



Figure: Events and Messages in Processes

# Logical Clocks

Logical clock is an assignment of numbers on events

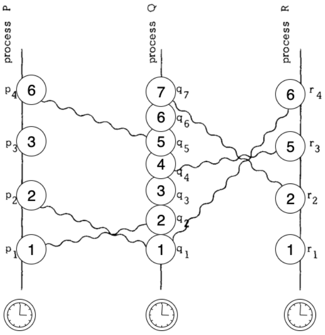



Figure: Logical Clock

# Logical Clocks

Logical clock is an assignment of numbers on events

- Clock $C_i$ assigns number $C_i\langle a\rangle$ to event $a$ in process $P_i$

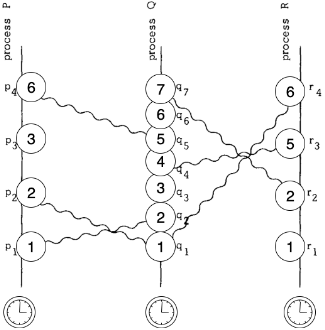



Figure: Logical Clock

# Logical Clocks

Logical clock is an assignment of numbers on events

- Clock $C_i$ assigns number $C_i\langle a\rangle$ to event $a$ in process $P_i$
- Clock $C$ for the entire system defined by $C\langle a\rangle = C_i\langle a\rangle$ if $a$ is in process $P_i$

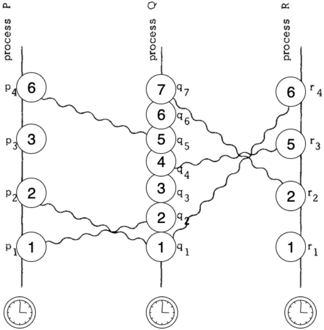



Figure: Logical Clock

# Logical Clocks

### Clock Condition

For any events $a, b$:
if $a \rightarrow b$ then $C\langle a\rangle < C\langle b\rangle$

### Remark

The converse is not true:
$C\langle a\rangle < C\langle b\rangle$ does not imply $a \rightarrow b$

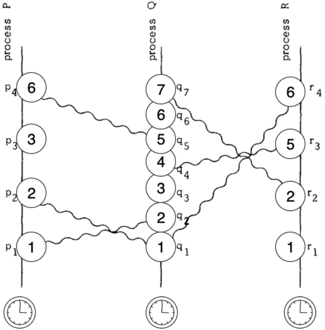



Figure: Logical Clock

# Logical Clocks

Implement the logical clock:

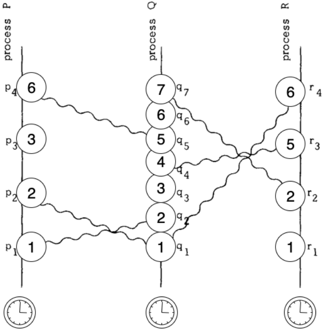



Figure: Logical Clock

# Logical Clocks

Implement the logical clock:

- Each process $P_i$ increments $C_i$ between successive events

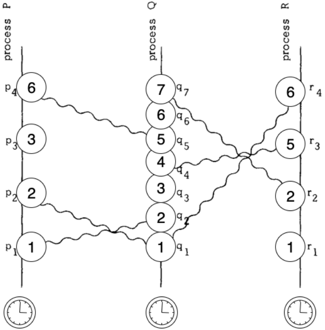



Figure: Logical Clock

# Logical Clocks

Implement the logical clock:

- Each process $P_i$ increments $C_i$ between successive events
- If event $a$ is sending message $m$ from $P_i$, then $m$ contains timestamp $T_m = C_i\langle a\rangle$

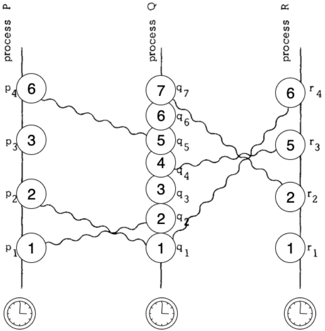



Figure: Logical Clock

# Logical Clocks

Implement the logical clock:

- Each process $P_i$ increments $C_i$ between successive events
- If event $a$ is sending message $m$ from $P_i$, then $m$ contains timestamp $T_m = C_i\langle a \rangle$
- On receiving message $m$, $P_j$ sets $C_j$ to be greater than both $T_m$ and previous event

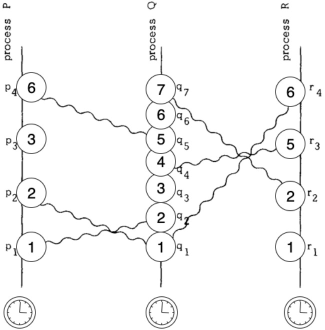



Figure: Logical Clock

# Logical Clocks

Process $Q$ receives message $p_1$, updates clock to 2

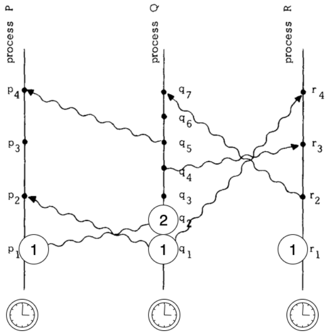



Figure: Logical Clock

# Logical Clocks

Process $P$ receives message $q_1$, updates clock to 2

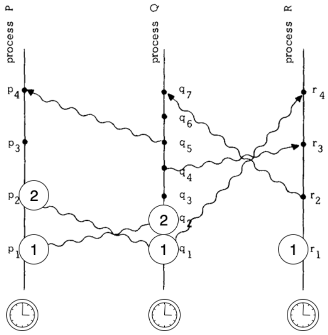



Figure: Logical Clock

# Logical Clocks

Proceeds until $Q$ sends a message to $R$ at event $q_4$ with timestamp 4

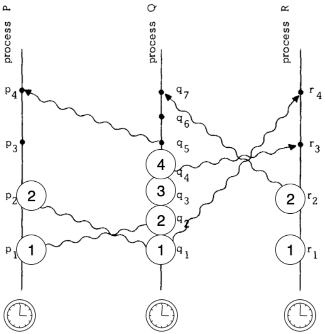



Figure: Logical Clock

# Logical Clocks

Process $R$ receives the message with timestamp 4, and updates clock to 5

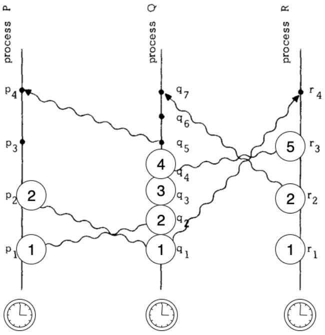



Figure: Logical Clock

# Logical Clocks

Process $Q$ sends message to $P$ with timestamp 5

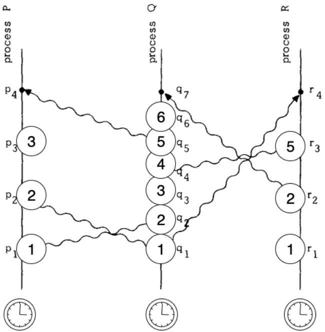



Figure: Logical Clock

# Logical Clocks

Process $P$ updates clock on receiving message with timestamp 5. Clocks of processes $Q$ and $R$ are not affected by messages.

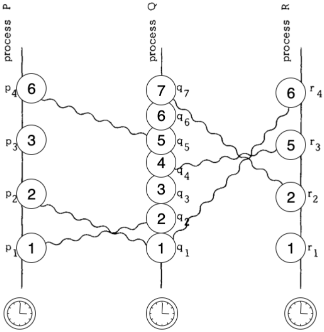



Figure: Logical Clock

# Total Ordering

With the logical clock, we can now define a total order "⇒" for all events.

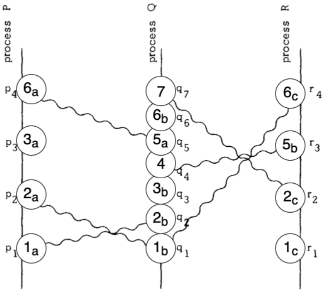



Figure: Logical Clock

# Total Ordering

With the logical clock, we can now define a total order "$\Rightarrow$" for all events.

- Define a total order $\prec$ over the processes

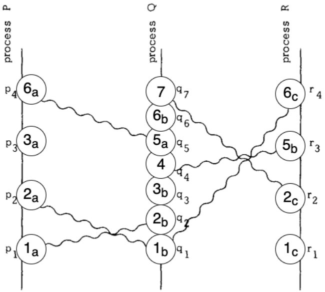



Figure: Logical Clock

# Total Ordering

With the logical clock, we can now define a total order "$\Rightarrow$" for all events.

- Define a total order $\prec$ over the processes
- For events $a$ in $P_i$ and $b$ in $P_j$, $a \Rightarrow b$ if and only if either
  - $C_i\langle a\rangle < C_j\langle b\rangle$ or;
  - $C_i\langle a\rangle = C_j\langle b\rangle$ and $P_i \prec P_j$

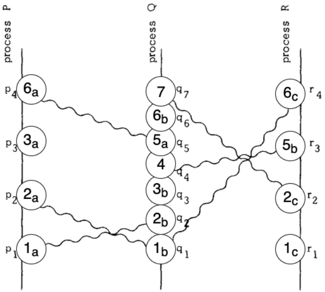



Figure: Logical Clock

# Mutual Exclusion

Consider a system composed of a fixed collection of processes which share a single resource. Only one process can use the resource at a time.

- (I) A process which has been granted the resource must release it before it can be granted to another process.
- (II) Different requests for the resource must be granted in the order in which they are made.
- (III) If every process which is granted the resource eventually releases it, then every request is eventually granted.

# Mutual Exclusion

To simplify the implementation, some assumptions are needed to avoid extra details.

# Mutual Exclusion

To simplify the implementation, some assumptions are needed to avoid extra details.

- For any $P_i$ and $P_j$, the messages sent from $P_i$ to $P_j$ are received in order

# Mutual Exclusion

To simplify the implementation, some assumptions are needed to avoid extra details.

- For any $P_i$ and $P_j$, the messages sent from $P_i$ to $P_j$ are received in order
- Every message is eventually received

# Mutual Exclusion

To simplify the implementation, some assumptions are needed to avoid extra details.

- For any $P_i$ and $P_j$, the messages sent from $P_i$ to $P_j$ are received in order
- Every message is eventually received
- A process can send messages directly to every other process

# Mutual Exclusion

Each process maintains its own *request queue*.

# Mutual Exclusion

Each process maintains its own *request queue*.

- If process $P_i$ wants the resource, it sends a message $\langle T_m : P_i \textit{ requests resource}\rangle$ to every other process, and puts the message on its request queue, where $T_m$ is the message timestamp

# Mutual Exclusion

Each process maintains its own *request queue*.

- If process $P_i$ wants the resource, it sends a message $\langle T_m : P_i \textit{ requests resource} \rangle$ to every other process, and puts the message on its request queue, where $T_m$ is the message timestamp
- When process $P_j$ receives $\langle T_m : P_i \textit{ requests resource} \rangle$, it puts the message on its request queue, and sends an acknowledgement message to $P_i$

# Mutual Exclusion

Releasing the resource works in similar way.

# Mutual Exclusion

Releasing the resource works in similar way.

- If process $P_i$ wants to release the resource, it sends a message $\langle T_m : P_i$ *releases resource*$\rangle$ to every other process, and removes any $\langle T_m : P_i$ *requests resource*$\rangle$ from its request queue

# Mutual Exclusion

Releasing the resource works in similar way.

- If process $P_i$ wants to release the resource, it sends a message $\langle T_m : P_i \textit{ releases resource} \rangle$ to every other process, and removes any $\langle T_m : P_i \textit{ requests resource} \rangle$ from its request queue
- When process $P_j$ receives $\langle T_m : P_i \textit{ releases resource} \rangle$, it removes any $\langle T_m : P_i \textit{ requests resource} \rangle$ from its request queue

# Mutual Exclusion

$P_i$ is granted the resource if the following conditions hold.

# Mutual Exclusion

$P_i$ is granted the resource if the following conditions hold.

- There is a $\langle T_m : P_i \textit{ requests resource}\rangle$ in its request queue ordered before any other messages inside the queue by relation "$\Rightarrow$"

# Mutual Exclusion

$P_i$ is granted the resource if the following conditions hold.

- There is a $\langle T_m : P_i$ *requests resource*$\rangle$ in its request queue ordered before any other messages inside the queue by relation "$\Rightarrow$"
- $P_i$ received the acknowledgements from every other process

# Mutual Exclusion

The above protocol satisfies conditions (I)(II) and (III).

# Mutual Exclusion

The above protocol satisfies conditions (I)(II) and (III).

- **Conditions I and II:** When $\langle T_m : P_i \textit{ requests resource}\rangle$ is sent, no request ordered after this request will be granted before this request is released, because they either:
  - Are not acknowledged by $P_i$
  - Received $\langle T_m : P_i \textit{ requests resource}\rangle$ which is ordered before them in their queues

# Mutual Exclusion

The above protocol satisfies conditions (I)(II) and (III).

- **Conditions I and II:** When $\langle T_m : P_i \textit{ requests resource}\rangle$ is sent, no request ordered after this request will be granted before this request is released, because they either:
    - Are not acknowledged by $P_i$
    - Received $\langle T_m : P_i \textit{ requests resource}\rangle$ which is ordered before them in their queues
- **Condition III:** If all the requests before $\langle T_m : P_i \textit{ requests resource}\rangle$ are released, this request will be ordered before any other requests in $P_i$'s queue

# Physical Clock

Let $t$ denote the physical time, and $C_i(t)$ be the value of clock $C_i$.

# Physical Clock

Let $t$ denote the physical time, and $C_i(t)$ be the value of clock $C_i$.

- $C_i(t)$ is continuous and differentiable function of $t$ except for jump discontinuities

# Physical Clock

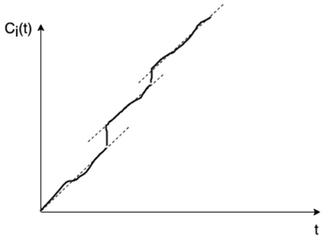
Let $t$ denote the physical time, and $C_i(t)$ be the value of clock $C_i$.

- $C_i(t)$ is continuous and differentiable function of $t$ except for jump discontinuities
- $dC_i(t)/dt$ is the rate of the clock at $t$, assumed that $|dC_i(t)/dt - 1| < \kappa \ll 1$

# Physical Clock

Let $t$ denote the physical time, and $C_i(t)$ be the value of clock $C_i$.

- $C_i(t)$ is continuous and differentiable function of $t$ except for jump discontinuities
- $dC_i(t)/dt$ is the rate of the clock at $t$, assumed that $|dC_i(t)/dt - 1| < \kappa \ll 1$



# Physical Clock

We need a protocol to make sure that all the $C_i(t)$'s are synced, i.e. for all $i, j$:

$$|C_i(t) - C_j(t)| < \varepsilon$$

# Physical Clock

We need a protocol to make sure that all the $C_i(t)$'s are synced, i.e. for all $i, j$:

$$|C_i(t) - C_j(t)| < \varepsilon$$

We need some assumptions and notations before describing the protocol.

# Physical Clock

We need a protocol to make sure that all the $C_i(t)$'s are synced, i.e. for all $i, j$:

$$|C_i(t) - C_j(t)| < \varepsilon$$

We need some assumptions and notations before describing the protocol.

- If a message $m$ is sent at time $t$, received at time $t'$, the total delay is $\nu_m = t' - t$, which is not known to the receiver

# Physical Clock

We need a protocol to make sure that all the $C_i(t)$'s are synced, i.e. for all $i, j$:

$$|C_i(t) - C_j(t)| < \varepsilon$$

We need some assumptions and notations before describing the protocol.

- If a message $m$ is sent at time $t$, received at time $t'$, the total delay is $\nu_m = t' - t$, which is not known to the receiver
- We assume that the receiving process knows some minimum delay $\mu_m \leq \nu_m$

# Physical Clock

We need a protocol to make sure that all the $C_i(t)$'s are synced, i.e. for all $i, j$:

$$|C_i(t) - C_j(t)| < \varepsilon$$

We need some assumptions and notations before describing the protocol.

- If a message $m$ is sent at time $t$, received at time $t'$, the total delay is $\nu_m = t' - t$, which is not known to the receiver
- We assume that the receiving process knows some minimum delay $\mu_m \leq \nu_m$
- Denote by $\xi_m = \nu_m - \mu_m$ the *unpredictable delay*

# Physical Clock

The protocol executes as follows.

# Physical Clock

The protocol executes as follows.

- If $P_i$ does not receive message at time $t$, then $P_i$ does not modify $C_i$, and $dC_i(t)/dt \approx 1$

# Physical Clock

The protocol executes as follows.

- If $P_i$ does not receive message at time $t$, then $P_i$ does not modify $C_i$, and $dC_i(t)/dt \approx 1$
- When $P_i$ sends a message $m$, it appends timestamp $T_m = C_i(t)$ in the message

# Physical Clock

The protocol executes as follows.

- If $P_i$ does not receive message at time $t$, then $P_i$ does not modify $C_i$, and $dC_i(t)/dt \approx 1$
- When $P_i$ sends a message $m$, it appends timestamp $T_m = C_i(t)$ in the message
- If at time $t'$, $P_i$ receives a message $m$ with timestamp $T_m$, $P_i$ resets $C_i(t')$ to $\max(C_i(t'), T_m + \mu_m)$

# Physical Clock

## Theorem

Assume a strongly connected graph of processes with diameter $d$ follows the above protocol. Assume that for any message $m$, $\mu_m \leq \mu$ for some constant $\mu$, and that for all $t \geq t_0$:

1. $|dC_i(t)/dt - 1| \leq \kappa \ll 1$
2. every $\tau$ seconds a message $m$ with $\xi_m < \xi$ is sent over every arc, where $\tau$ and $\xi$ are constants

Then for all $i, j$: $|C_i(t) - C_j(t)| < \varepsilon$ where $\varepsilon \approx d(2\kappa\tau + \xi)$ for all $t \gtrsim t_0 + \tau d$ and $\mu + \xi \ll \tau$

# Physical Clock

## Proof Sketch

# Physical Clock

## Proof Sketch

1. Prove that for any $i, j$ and any $t, t_1$ with $t_1 \geq t_0$ and $t \geq t_1 + d(\tau + \nu)$:

$$C_i(t) \geq C_j(t_1) + (1 - \kappa)(t - t_1) - d\xi$$

# Physical Clock

## Proof Sketch

1. Prove that for any $i, j$ and any $t, t_1$ with $t_1 \geq t_0$ and $t \geq t_1 + d(\tau + \nu)$:

$$C_i(t) \geq C_j(t_1) + (1 - \kappa)(t - t_1) - d\xi$$

2. Prove that for any $t, t_x$ with $t \geq t_x \geq t_0 + \mu/(1 - \kappa)$ there is a process $P_q$ and a time $t_1$ with $t_x - \mu/(1 - \kappa) \leq t_1 \leq t_x$ such that for all $i$:

$$C_i(t) \leq C_q(t_1) + (1 + \kappa)(t - t_1)$$

# Physical Clock

## Proof Sketch

1. Prove that for any $i, j$ and any $t, t_1$ with $t_1 \geq t_0$ and $t \geq t_1 + d(\tau + \nu)$:

$$C_i(t) \geq C_j(t_1) + (1 - \kappa)(t - t_1) - d\xi$$

2. Prove that for any $t, t_x$ with $t \geq t_x \geq t_0 + \mu/(1 - \kappa)$ there is a process $P_q$ and a time $t_1$ with $t_x - \mu/(1 - \kappa) \leq t_1 \leq t_x$ such that for all $i$:

$$C_i(t) \leq C_q(t_1) + (1 + \kappa)(t - t_1)$$

3. Prove that for all $i, j$:

$$|C_i(t) - C_j(t)| \lesssim d(2\kappa\tau + \xi)$$

for all $t \gtrsim t_0 + d\tau$

# Physical Clock

## Step 1

1. Define $C_i^t$ to be a clock set equal to $C_i$ at time $t$ and runs at the same rate as $C_i$, but is never reset. Then we have for any $t' \geq t$, $C_i^t(t') \leq C_i(t')$

# Physical Clock

## Step 1

1. Define $C_i^t$ to be a clock set equal to $C_i$ at time $t$ and runs at the same rate as $C_i$, but is never reset. Then we have for any $t' \geq t$, $C_i^t(t') \leq C_i(t')$
2. Suppose $P_1$ at time $t_1$ sends a message to $P_2$ which is received at $t_2$, then for any $t \geq t_2$

$$C_2^{t_2}(t) \geq C_1(t_1) + (1 - \kappa)(t - t_1) - \xi$$

# Physical Clock

## Step 1

1. Define $C_i^t$ to be a clock set equal to $C_i$ at time $t$ and runs at the same rate as $C_i$, but is never reset. Then we have for any $t' \geq t$, $C_i^t(t') \leq C_i(t')$
2. Suppose $P_1$ at time $t_1$ sends a message to $P_2$ which is received at $t_2$, then for any $t \geq t_2$
$$C_2^{t_2}(t) \geq C_1(t_1) + (1-\kappa)(t - t_1) - \xi$$
3. For any $P$ and $P'$, there is a sequence $P = P_0, P_1, \cdots, P_{n+1} = P'$, $n \leq d$, for each pair of $P_i, P_{i+1}$, assume $P_i$ receives a message at $t_i$, sends a message to $P_{i+1}$ at $t_i'$, and $P_{i+1}$ receives message at $t_{i+1}$, we can find $t_i' - t_i \leq \tau$, $t_{i+1} - t_i' \leq \nu$. Then we have
$$C_{n+1}(t) \geq C_{n+1}^{t_{n+1}}(t) \geq C_1(t_1) + (1-\kappa)(t - t_1) - n\xi$$

# Physical Clock

## Step 2

1. Assign a clock $C_m$ to each message $m$ sent at $t$ and received at $t'$, with $C_m(t) = t$ and constant rate $dC_m/dt = \mu_m/(t' - t)$

# Physical Clock

## Step 2

1. Assign a clock $C_m$ to each message $m$ sent at $t$ and received at $t'$, with $C_m(t) = t$ and constant rate $dC_m/dt = \mu_m/(t' - t)$
2. For any time $t_x \geq t_0 + \mu/(1 - \kappa)$, let $C_x$ be the clock with largest value at $t_x$. Consider two cases:

# Physical Clock

## Step 2

1. Assign a clock $C_m$ to each message $m$ sent at $t$ and received at $t'$, with $C_m(t) = t$ and constant rate $dC_m/dt = \mu_m/(t' - t)$
2. For any time $t_x \geq t_0 + \mu/(1 - \kappa)$, let $C_x$ be the clock with largest value at $t_x$. Consider two cases:
   2.1 $C_x$ is $C_q$ for some process $P_q$, then for any $i$ and $t \geq t_x$

$$C_i(t) \leq C_q(t_x) + (1 + \kappa)(t - t_x)$$

# Physical Clock

## Step 2

1. Assign a clock $C_m$ to each message $m$ sent at $t$ and received at $t'$, with $C_m(t) = t$ and constant rate $dC_m/dt = \mu_m/(t' - t)$
2. For any time $t_x \geq t_0 + \mu/(1 - \kappa)$, let $C_x$ be the clock with largest value at $t_x$. Consider two cases:
   2.1 $C_x$ is $C_q$ for some process $P_q$, then for any $i$ and $t \geq t_x$
   $$C_i(t) \leq C_q(t_x) + (1 + \kappa)(t - t_x)$$
   2.2 $C_x$ is $C_m$ for some message $m$ sent at $t_1$ by process $P_q$, then $t_x \leq t_1 + \mu/(1 - \kappa)$, and for any $i$ and $t \geq t_1$
   $$C_i(t) \leq C_q(t_1) + (1 + \kappa)(t - t_1)$$

# Physical Clock

## Step 2

1. Assign a clock $C_m$ to each message $m$ sent at $t$ and received at $t'$, with $C_m(t) = t$ and constant rate $dC_m/dt = \mu_m/(t' - t)$
2. For any time $t_x \geq t_0 + \mu/(1 - \kappa)$, let $C_x$ be the clock with largest value at $t_x$. Consider two cases:
   2.1 $C_x$ is $C_q$ for some process $P_q$, then for any $i$ and $t \geq t_x$

   $$C_i(t) \leq C_q(t_x) + (1 + \kappa)(t - t_x)$$

   2.2 $C_x$ is $C_m$ for some message $m$ sent at $t_1$ by process $P_q$, then $t_x \leq t_1 + \mu/(1 - \kappa)$, and for any $i$ and $t \geq t_1$

   $$C_i(t) \leq C_q(t_1) + (1 + \kappa)(t - t_1)$$

3. Let $t_1 = t_x$ in first case, then we have $t_1$ such that $t_x - \mu/(1 - \kappa) \leq t_1 \leq t_x$, and there exists process $P_q$ such that for all $t \geq t_x$ and for all $i$ the above equation holds.

# Physical Clock

## Step 3

1. Now we conclude that there always exists process $P_q$ and time $t_1$ such that

$$C_q(t_1) + (1-\kappa)(t-t_1) - d\xi \leq C_i(t) \leq C_q(t_1) + (1+\kappa)(t-t_1)$$

# Physical Clock

## Step 3

1. Now we conclude that there always exists process $P_q$ and time $t_1$ such that

$$C_q(t_1)+(1-\kappa)(t-t_1)-d\xi \leq C_i(t) \leq C_q(t_1)+(1+\kappa)(t-t_1)$$

2. Let $t = t_x + d(\tau + \nu)$, update the above bounds

$$\begin{aligned} C_q(t_1) + (t - t_1) - \kappa d(\tau + \nu) - d\xi &\leq C_i(t) \\ \leq C_q(t_1) + (t - t_1) + \kappa[d(\tau + \nu) + \mu/(1 - \kappa)] \end{aligned}$$

# Physical Clock

## Step 3

1. Now we conclude that there always exists process $P_q$ and time $t_1$ such that

$$C_q(t_1)+(1-\kappa)(t-t_1)-d\xi \leq C_i(t) \leq C_q(t_1)+(1+\kappa)(t-t_1)$$

2. Let $t = t_x + d(\tau + \nu)$, update the above bounds

$$C_q(t_1) + (t - t_1) - \kappa d(\tau + \nu) - d\xi \leq C_i(t)$$
$$\leq C_q(t_1) + (t - t_1) + \kappa[d(\tau + \nu) + \mu/(1 - \kappa)]$$

3. By $\mu \leq \nu \ll \tau$ and $\kappa \ll 1$, and the fact that the above holds for all $i$

$$|C_i(t) - C_j(t)| \lesssim d(2\kappa\tau + \xi)$$

Q&A