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

Leslie Lamport

October 18, 2022

Presented by: Ricky Takkar
Instructor: Robbert van Renesse
Cornell CS

Fun facts:

1. With >13k citations, this is Lamport's most often cited paper
2. LaTeXoriginates from a set of macros created by Lamport for Donald Knuth's TeXtypesetting system

Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Communications of the ACM* 21.7 (July 1978), pp. 558–565. issn: 0001-0782. doi: 10.1145/359545.359563. url: https://doi.org/10.1145/359545.359563 (visited on 10/02/2022).

Abstract

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Abstract

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

How Humans View Time

We say that something happened at 3:15 if it occurred:

How Humans View Time

We say that something happened at 3:15 if it occurred:

- *after* our clock read 3:15 and

How Humans View Time

We say that something happened at 3:15 if it occurred:

- *after* our clock read 3:15 and
- *before* it read 3:16

How Humans View Time

We say that something happened at 3:15 if it occurred:

- *after* our clock read 3:15 and
- *before* it read 3:16

For example, in an airline reservation system we specify that a request for a reservation should be granted if it is made *before* the flight is filled.

How Humans View Time

We say that something happened at 3:15 if it occurred:

- *after* our clock read 3:15 and
- *before* it read 3:16

For example, in an airline reservation system we specify that a request for a reservation should be granted if it is made *before* the flight is filled.

The concept of the temporal ordering of events pervades our thinking about systems.

Machines Think Different™

Distributed Systems 101:

- they consist of a collection of **distinct processes which are spatially separated**, and which communicate with one another by exchanging messages

Machines Think Different™

Distributed Systems 101:

- they consist of a collection of **distinct processes which are spatially separated**, and which communicate with one another by exchanging messages
- e.g., network of interconnected computers, even a single computer (central control unit, memory units, I/O channels are separate processes)

Machines Think Different™

Distributed Systems 101:

- they consist of a collection of **distinct processes which are spatially separated**, and which communicate with one another by exchanging messages
- e.g., network of interconnected computers, even a single computer (central control unit, memory units, I/O channels are separate processes)

A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

Machines Think Different™

Distributed Systems 101:

- they consist of a collection of **distinct processes which are spatially separated**, and which communicate with one another by exchanging messages
- e.g., network of interconnected computers, even a single computer (central control unit, memory units, I/O channels are separate processes)

A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

So what's different about them?

- sometimes impossible to say one of two events occurred first in a distributed system

Machines Think Different™

Distributed Systems 101:

- they consist of a collection of **distinct processes which are spatially separated**, and which communicate with one another by exchanging messages
- e.g., network of interconnected computers, even a single computer (central control unit, memory units, I/O channels are separate processes)

A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

So what's different about them?

- sometimes impossible to say one of two events occurred first in a distributed system
- relation "happened before" is therefore only a partial ordering of the events in the system

Machines Think Different™

Distributed Systems 101:

- they consist of a collection of **distinct processes which are spatially separated**, and which communicate with one another by exchanging messages
- e.g., network of interconnected computers, even a single computer (central control unit, memory units, I/O channels are separate processes)

A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

So what's different about them?

- sometimes impossible to say one of two events occurred first in a distributed system
- relation "happened before" is therefore only a partial ordering of the events in the system

Problems often arise because people are not fully aware of this fact and its implications.

Intro to Partial Ordering

Recap: Most **people** would probably say that an event a happened before an event b if a happened at an earlier time than b. However, if a **system** is to meet a specification correctly, then that specification must be given in terms of events observable within the system.

Intro to Partial Ordering

Recap: Most **people** would probably say that an event a happened before an event b if a happened at an earlier time than b. However, if a **system** is to meet a specification correctly, then that specification must be given in terms of events observable within the system.

Let's say the spec is in terms of physical time and the system contains real clocks. It's impossible to guarantee clock accuracy. **Uh-oh!**

Intro to Partial Ordering

Recap: Most **people** would probably say that an event a happened before an event b if a happened at an earlier time than b. However, if a **system** is to meet a specification correctly, then that specification must be given in terms of events observable within the system.

Let's say the spec is in terms of physical time and the system contains real clocks. It's impossible to guarantee clock accuracy. **Uh-oh!**

No worries! Lamport defined the "happened before" relation without using physical clocks.

Definition

The "happened before" relation, denoted by "$\rightarrow$", on the set of events of a system is the smallest relation satisfying the following three conditions:

Definition

The "happened before" relation, denoted by "$\rightarrow$", on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.

Definition

The "happened before" relation, denoted by "$\rightarrow$", on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.

Definition

The "happened before" relation, denoted by "$\rightarrow$", on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$. Two distinct events a and b are said to be concurrent if $a \not\rightarrow b$ and $b \not\rightarrow a$.

Definition

The "happened before" relation, denoted by "$\rightarrow$", on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$. Two distinct events a and b are said to be concurrent if $a \not\rightarrow b$ and $b \not\rightarrow a$.

Another way to think about concurrency: $a \rightarrow b$ means it's possible for a to causally affect b. Concurrent events don't causally affect each other.

Figure 1

Space-time diagram

- horizontal direction represents space and the vertical direction represents time—later times are higher
- dots denote events
- vertical lines denote processes
- wavy lines denote messages

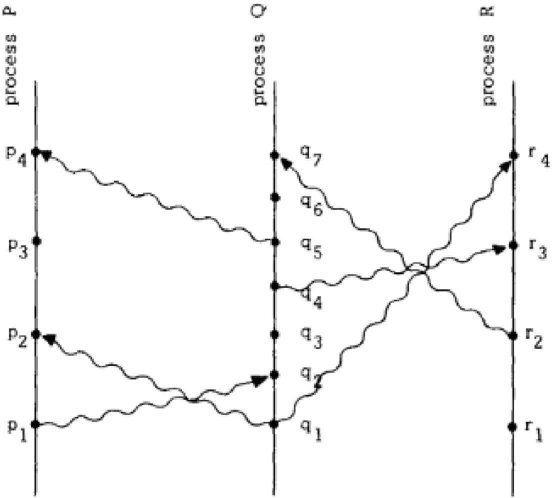


Figure 1

Intro to Logical Clocks

Time's up for your perception of clocks! Lamport defines it differently. He:

Intro to Logical Clocks

Time's up for your perception of clocks! Lamport defines it differently. He:

- *defines* a clock C_i for each process P_i to be a function which assigns a number $C_i\langle a\rangle$ to any event a in that process

Intro to Logical Clocks

Time's up for your perception of clocks! Lamport defines it differently. He:

- *defines* a clock C_i for each process P_i to be a function which assigns a number $C_i\langle a\rangle$ to any event a in that process
- *represents* the entire system of clocks by the function C which assigns to any event b the number $C\langle b\rangle$, where $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process P_j

Intro to Logical Clocks

Time's up for your perception of clocks! Lamport defines it differently. He:

- *defines* a clock C_i for each process P_i to be a function which assigns a number $C_i\langle a\rangle$ to any event a in that process
- *represents* the entire system of clocks by the function C which assigns to any event b the number $C\langle b\rangle$, where $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process P_j

What makes C_i "logical" rather than "physical" clocks is that we make no assumption about the relation of the numbers $C_i\langle a\rangle$ to physical time.

Intro to Logical Clocks

Time's up for your perception of clocks! Lamport defines it differently. He:

- *defines* a clock C_i for each process P_i to be a function which assigns a number $C_i\langle a\rangle$ to any event a in that process
- *represents* the entire system of clocks by the function C which assigns to any event b the number $C\langle b\rangle$, where $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process P_j

What makes C_i "logical" rather than "physical" clocks is that we make no assumption about the relation of the numbers $C_i\langle a\rangle$ to physical time.

What about correctness? Remember: no physical time! The strongest reasonable condition is that if an event a occurs before another event b, then a should happen at an earlier time than b.

Clock Condition

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

Clock Condition

Recap For any events a, b: if $a \rightarrow b$ then $C\langle a\rangle < C\langle b\rangle$. Oh, this is the clock condition. But there's more...

Note that we can't expect the converse condition, *i.e.*, if $C\langle a\rangle < C\langle b\rangle$ then $a \rightarrow b$, to hold as well because that would imply that any two ____ events must occur at the ____ time.

Clock Condition

Recap For any events a, b: if $a \rightarrow b$ then $C\langle a\rangle < C\langle b\rangle$. Oh, this is the clock condition. But there's more...

Note that we can't expect the converse condition, *i.e.*, if $C\langle a\rangle < C\langle b\rangle$ then $a \rightarrow b$, to hold as well because that would imply that any two ____ events must occur at the ____ time.

The following two conditions must hold to satisfy the Clock Condition:

C1 If a and b are events in process P_i, and a comes before b, then $C_i\langle a\rangle < C_i\langle b\rangle$

Clock Condition

Recapp For any events a, b: if $a \to b$ then $C\langle a\rangle < C\langle b\rangle$. Oh, this is the clock condition. But there's more...

Note that we can't expect the converse condition, *i.e.*, if $C\langle a\rangle < C\langle b\rangle$ then $a \to b$, to hold as well because that would imply that any two ____ events must occur at the ____ time.

The following two conditions must hold to satisfy the Clock Condition:

- **C1** If a and b are events in process P_i, and a comes before b, then $C_i\langle a\rangle < C_i\langle b\rangle$
- **C2** If a is the sending of a message by process P_i and b is the receipt of that message by process P_j, then $C_i\langle a\rangle < C_j\langle b\rangle$

Figure 2

Space-time diagram

- dashed "tick line" through all the like-numbered ticks of the different processes.
- consider the tick lines to be the time coordinate lines of some Cartesian coordinate system on space-time

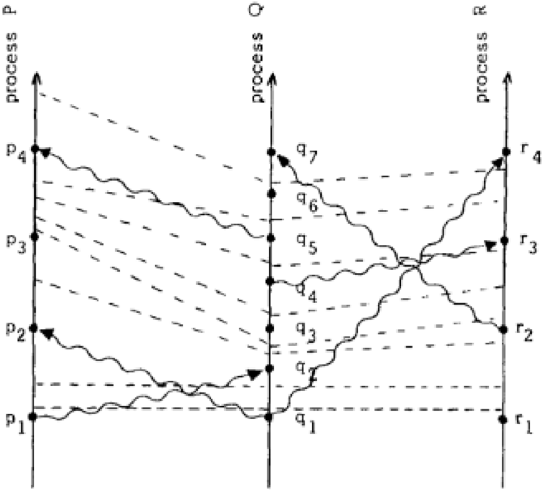


Figure 2

Figure 3

Space-time diagram

- Same as Figure 2 except we straightened the coordinate lines
- Which figure is a better representation? No right answer due to lack of physical time concept in system.

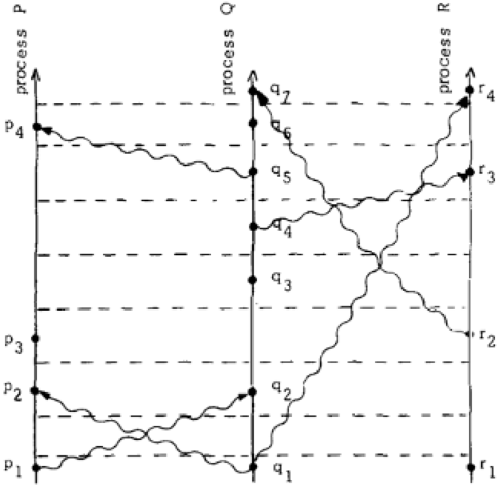


Figure 3

Implementation Rule

Let's make things less abstract! Assume now that processes are algorithms, and the events represent certain actions during their execution. How do we introduce these clocks we've been talking about into processes?

Implementation Rule

Let's make things less abstract! Assume now that processes are algorithms, and the events represent certain actions during their execution. How do we introduce these clocks we've been talking about into processes?

Note: Process P_i's clock is represented by a register C_i, so that $C_i\langle a\rangle$ is the value contained by C_i during the event a. The value of C_i will change between events, so changing C_i does not itself constitute an event.

Implementation Rule

Let's make things less abstract! Assume now that processes are algorithms, and the events represent certain actions during their execution. How do we introduce these clocks we've been talking about into processes?

Note: Process P_i's clock is represented by a register C_i, so that $C_i\langle a \rangle$ is the value contained by C_i during the event a. The value of C_i will change between events, so changing C_i does not itself constitute an event.

To satisfy the Clock Condition, we introduce implementation rules IR1 and IR2, where condition C1 is satisfied by the process obeying IR1 and condition C2 is satisfied by the process obeying IR2:

Implementation Rule

Let's make things less abstract! Assume now that processes are algorithms, and the events represent certain actions during their execution. How do we introduce these clocks we've been talking about into processes?

Note: Process P_i's clock is represented by a register C_i, so that $C_i\langle a\rangle$ is the value contained by C_i during the event a. The value of C_i will change between events, so changing C_i does not itself constitute an event.

To satisfy the Clock Condition, we introduce implementation rules IR1 and IR2, where condition C1 is satisfied by the process obeying IR1 and condition C2 is satisfied by the process obeying IR2:

IR1 Each process P_i increments C_i between any two successive events.

Implementation Rule

Let's make things less abstract! Assume now that processes are algorithms, and the events represent certain actions during their execution. How do we introduce these clocks we've been talking about into processes?

Note: Process P_i's clock is represented by a register C_i, so that $C_i\langle a\rangle$ is the value contained by C_i during the event a. The value of C_i will change between events, so changing C_i does not itself constitute an event.

To satisfy the Clock Condition, we introduce implementation rules IR1 and IR2, where condition C1 is satisfied by the process obeying IR1 and condition C2 is satisfied by the process obeying IR2:

IR1 Each process P_i increments C_i between any two successive events.

IR2 (a) If event a is the sending of a message m by process P_i, then the message m contains a timestamp $T_m = C_i\langle a\rangle$. (b) Upon receiving a message m, process P_j sets C_j greater than or equal to its present value and greater than T_m.

But How? Informally, Like So

- The system of clocks must satisfy the Clock Condition

But How? Informally, Like So

- The system of clocks must satisfy the Clock Condition
- Order the events by the times at which they occur

But How? Informally, Like So

- The system of clocks must satisfy the Clock Condition
- Order the events by the times at which they occur
- Tiebreaker: use any arbitrary total ordering $\prec$ of the processes

But How? Now, Lamport-Style

We define a relation $\Rightarrow$ as follows: if a is an event in process P_i and b is an event in process P_j, then $a \Rightarrow b$ if and only if either:

(i) $C_i\langle a\rangle < C_j\langle b\rangle$, or

(ii) $C_i\langle a\rangle = C_j\langle b\rangle$ and $P_i \prec P_j$

♩ In other words, please be true ♩, in other words, the relation $\Rightarrow$ is a way of completing the "happened before" partial ordering to a total ordering.

But How? Now, Lamport-Style

We define a relation $\Rightarrow$ as follows: if a is an event in process P_i and b is an event in process P_j, then $a \Rightarrow b$ if and only if either:

(i) $C_i\langle a\rangle < C_j\langle b\rangle$, or

(ii) $C_i\langle a\rangle = C_j\langle b\rangle$ and $P_i \prec P_j$

♩ In other words, please be true ♩, in other words, the relation $\Rightarrow$ is a way of completing the "happened before" partial ordering to a total ordering.

- Given any total ordering relation $\Rightarrow$ which extends $\rightarrow$, there is a system of clocks satisfying the Clock Condition which yields that relation. It is only the partial ordering $\rightarrow$ which is uniquely determined by the system of events

Example: Mutual Exclusion Problem

Why bother totally ordering events in a distributed system? Why do anything ever at all?

Example: Mutual Exclusion Problem

Why bother totally ordering events in a distributed system? Why do anything ever at all?

Mutual exclusion problem: in a system consisting of many processes and one resource, we wish to find an algorithm for granting the resource to a process which satisfies the following three conditions:

Example: Mutual Exclusion Problem

Why bother totally ordering events in a distributed system? Why do anything ever at all?

Mutual exclusion problem: in a system consisting of many processes and one resource, we wish to find an algorithm for granting the resource to a process which satisfies the following three conditions:

1. A process which has been granted the resource must release it before it can be granted to another process.
2. Different requests for the resource must be granted in the order in which they are made.
3. If every process which is granted the resource eventually releases it, then every request is eventually granted.

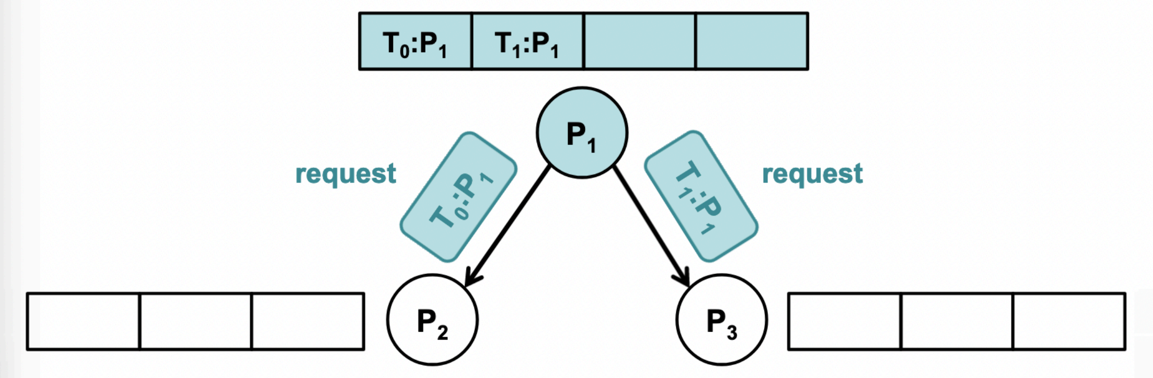
Assume that the resource is initially granted to exactly one process.

Algorithm: Rule #1 (out of 5)

1. To request the resource, process P_i sends the message $T_m : P_i$ *requests resource* to every other process, and puts that message on its request queue, where T_m is the timestamp of the message.

Algorithm: Rule #1 (out of 5)

1. To request the resource, process P_i sends the message $T_m : P_i$ *requests resource* to every other process, and puts that message on its request queue, where T_m is the timestamp of the message.



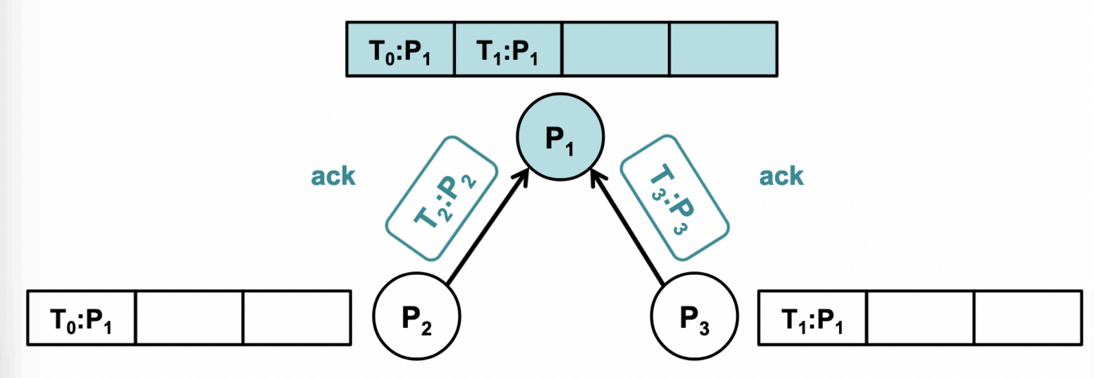
Source: Nicole Caruso, Cornell CS

Algorithm: Rule #2 (out of 5)

2. When process P_j receives the message $T_m : P_i$ *requests resource*, it places it on its request queue and sends a (timestamped) acknowledgment message to P_i.

Algorithm: Rule #2 (out of 5)

2. When process P_j receives the message $T_m : P_i$ *requests resource*, it places it on its request queue and sends a (timestamped) acknowledgment message to P_i.



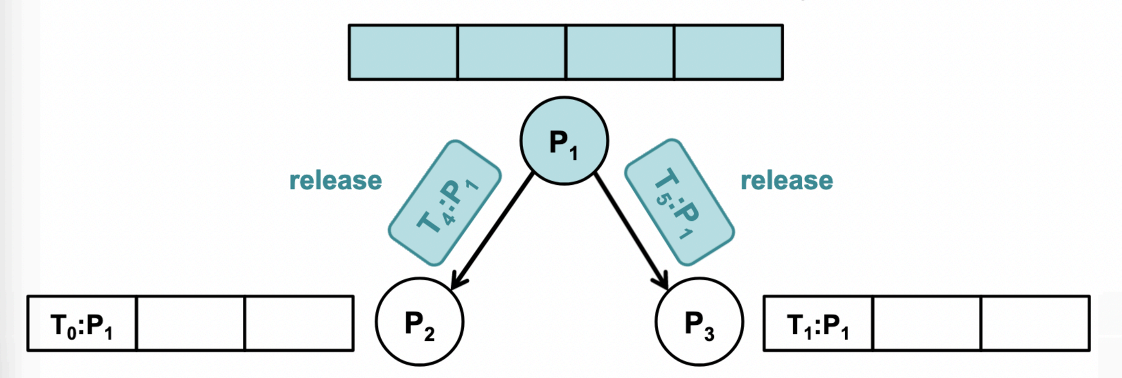
Source: Nicole Caruso, Cornell CS

Algorithm: Rule #3 (out of 5)

3. To release the resource, process P_i removes any $T_m : P_i$ *requests resource* message from its request queue and sends a (timestamped) P_i *releases resource* message to every other process.

Algorithm: Rule #3 (out of 5)

3. To release the resource, process P_i removes any $T_m : P_i$ *requests resource* message from its request queue and sends a (timestamped) P_i *releases resource* message to every other process.



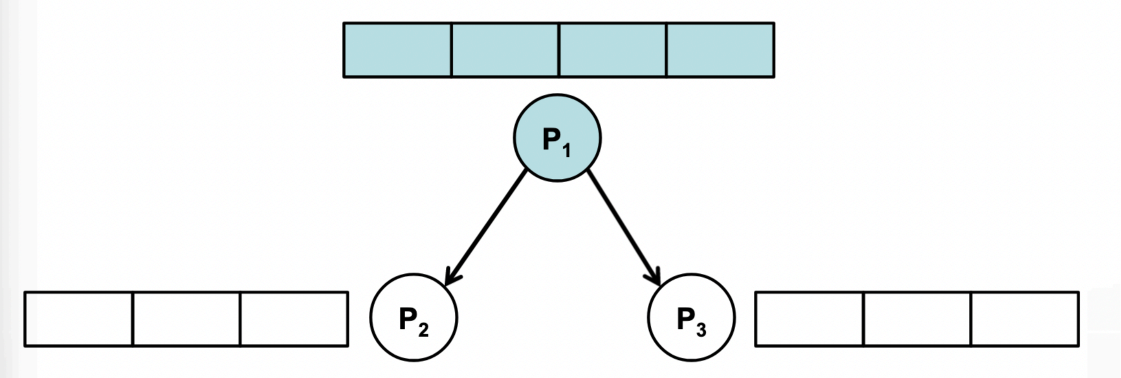
Source: Nicole Caruso, Cornell CS

Algorithm: Rule #4 (out of 5)

4. When process P_j receives a P_i *releases resource* message, it removes any $T_m : P_i$ *requests resource* message from its request queue.

Algorithm: Rule #4 (out of 5)

4. When process P_j receives a P_i *releases resource* message, it removes any $T_m : P_i$ *requests resource* message from its request queue.



Source: Nicole Caruso, Cornell CS

Algorithm: Rule #5 (out of 5)

5. Process P_i is granted the resource when the following two conditions are satisfied:
   i. There is a $T_m : P_i$ *requests resource* message in its request queue which is ordered before any other request in its queue by the relation $\Rightarrow$. (To define the relation "$\Rightarrow$" for messages, we identify a message with the event of sending it.)
   ii. P_i has received a message from every other process timestamped later than T_m.

Scenario

Recap: resource scheduling algorithm orders request in accordance with total ordering $\Rightarrow$.

Scenario

Recap: resource scheduling algorithm orders request in accordance with total ordering $\Rightarrow$.

Total ordering still permits the following type of anomalous behavior:

Scenario

Recap: resource scheduling algorithm orders request in accordance with total ordering $\Rightarrow$.

Total ordering still permits the following type of anomalous behavior:

- 2 computers in a network can try to obtain a shared resource at the *same time* causing a conflict. This can happen despite the fact that a request a may have been made on computer A before a request b may have been made on computer B because b comes before a on computer B.

Choice 1: Make Users Responsible

Guesses?

Choice 1: Make Users Responsible

Guesses?

Alex making request a receives timestamp T_a and broadcasts it to his friend Bob before he makes request b so that they ensure $T_b < T_a$

Choice 1: Make Users Responsible

Guesses?

Alex making request a receives timestamp T_a and broadcasts it to his friend Bob before he makes request b so that they ensure $T_b < T_a$

Thoughts?

Choice 2: Strong Clock Condition

Construct a system of clocks which satisfies the following condition:

Choice 2: Strong Clock Condition

Construct a system of clocks which satisfies the following condition:
Strong Clock Condition For any events a, b in S: if $a \hookrightarrow b$ then $C\langle a\rangle < C\langle b\rangle$.
Note: S refers to the set of all system events

Choice 2: Strong Clock Condition

Construct a system of clocks which satisfies the following condition:
Strong Clock Condition For any events a, b in S: if $a \hookrightarrow b$ then $C\langle a\rangle < C\langle b\rangle$.
Note: S refers to the set of all system events

One of the mysteries of the universe is that it is possible to construct a system of physical clocks which, running quite independently of one another, will satisfy the Strong Clock Condition.

Physical Clock Conditions

Let's introduce a physical time coordinate t! Let $C_i(t)$ denote the reading of the clock C_i at physical time t and $\frac{dC_i(t)}{dt}$ represent the rate at which the clock runs at t. In order for C_i to be a true physical clock, it must run at the correct rate, *i.e.*, $\frac{dC_i(t)}{dt} \approx 1$.

Physical Clock Conditions

Let's introduce a physical time coordinate t! Let $C_i(t)$ denote the reading of the clock C_i at physical time t and $\frac{dC_i(t)}{dt}$ represent the rate at which the clock runs at t. In order for C_i to be a true physical clock, it must run at the correct rate, *i.e.*, $\frac{dC_i(t)}{dt} \approx 1$.

More precisely,

PC1 There exists a constant $\kappa \ll 1$ such that for all $i : |\frac{dC_i(t)}{dt} - 1| < \kappa$, where $\kappa \leq 10^{-6}$ for quartz clocks. (Clocks individually run at approximately the correct rate) "drift"
But this is not enough...

Physical Clock Conditions

Let's introduce a physical time coordinate t! Let $C_i(t)$ denote the reading of the clock C_i at physical time t and $\frac{dC_i(t)}{dt}$ represent the rate at which the clock runs at t. In order for C_i to be a true physical clock, it must run at the correct rate, *i.e.*, $\frac{dC_i(t)}{dt} \approx 1$.

More precisely,

PC1 There exists a constant $\kappa \ll 1$ such that for all $i : |\frac{dC_i(t)}{dt} - 1| < \kappa$, where $\kappa \leq 10^{-6}$ for quartz clocks. (Clocks individually run at approximately the correct rate) "drift"
But this is not enough...

PC2 For all $i, j : |C_i(t) - C_j(t)| < \epsilon$. (Clocks must be synchronized so that $C_i(t) \approx C_j(t)$ for all i, j, and t) "skew"

Important Physical Clock Concepts

Keep in mind the following

- Clocks are never perfectly accurate, a term that refers to **“truth”**
- Any clock will also **drift** over time, causing **skew** between two clocks
- **Accuracy** relates to skew relative to a perfectly truthful clock
- **Precision** relates to skew between pairs of correct clocks in the system.

Ken Birman. *(Lecture Notes) CS5412 / Time-Related Content (Enrichment/Review).* https://www.cs.cornell.edu/courses/cs5412/2022fa/videos/lecture-9-enrichment.mp4. [Online; accessed 09-October-2022]. 2022

Specialized Rules IR1′ and IR2′

I won't cover IR1′ and IR2′ in the same level of detail as the paper because doing so requires a decent bit of math, which I think is beyond the scope of this presentation...

Recall PC2: For all $i, j : |C_i(t) - C_j(t)| < \epsilon$. (Clocks must be synchronized so that $C_i(t) \approx C_j(t)$ for all i, j, and t) "skew"

- Purpose of IR1′ and IR2′: to guarantee PC2 is satisfied by the system of physical clocks

Specialized Rules IR1′ and IR2′

I won't cover IR1′ and IR2′ in the same level of detail as the paper because doing so requires a decent bit of math, which I think is beyond the scope of this presentation...

Recall PC2: For all $i, j : |C_i(t) - C_j(t)| < \epsilon$. (Clocks must be synchronized so that $C_i(t) \approx C_j(t)$ for all i, j, and t) "skew"

- Purpose of IR1′ and IR2′: to guarantee PC2 is satisfied by the system of physical clocks
- IR1′ states clock readings change with physical time
- IR2′ states how clocks synchronize with each other. P_j's clock is set to max(current time, time at which message is received + expected minimum transmission delay)

Theorem

What does it do?

- States IR1$'$ and IR2$'$ establish PC2

Theorem

What does it do?

- States $IR1'$ and $IR2'$ establish PC2
- Bounds the time it takes for clocks to sync up at system startup time

Theorem

What does it do?

- States IR1′ and IR2′ establish PC2
- Bounds the time it takes for clocks to sync up at system startup time

Skipping detail due to time constraints. Also, very math intensive, so good luck! PS: Even Lamport thinks the proof of this theorem is difficult.

Conclusion

- Concept of "happening before" defines an invariant partial ordering of the events in a distributed multiprocess system

Conclusion

- Concept of “happening before” defines an invariant partial ordering of the events in a distributed multiprocess system
- We discussed an algorithm for extending that partial ordering to a somewhat arbitrary total ordering

Conclusion

- Concept of "happening before" defines an invariant partial ordering of the events in a distributed multiprocess system
- We discussed an algorithm for extending that partial ordering to a somewhat arbitrary total ordering
- Anomalous behavior arises when total ordering defined by algorithm disagrees with ordering perceived by system's users
  - Using properly synchronized clocks can prevent this

Conclusion

- Concept of "happening before" defines an invariant partial ordering of the events in a distributed multiprocess system
- We discussed an algorithm for extending that partial ordering to a somewhat arbitrary total ordering
- Anomalous behavior arises when total ordering defined by algorithm disagrees with ordering perceived by system's users
  - Using properly synchronized clocks can prevent this
- In a distributed system, it is important to realize that the order in which events occur is only a partial ordering

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2.

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial
- True or false: Any clock will skew over time, causing drift between two clocks

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial
- True or false: Any clock will skew over time, causing drift between two clocks
  False! Any clock will **drift** over time, causing **skew** between two clocks

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial
- True or false: Any clock will skew over time, causing drift between two clocks
  False! Any clock will **drift** over time, causing **skew** between two clocks
- Because we know the timestamp of event a to be less than the timestamp of event b, we can safely say that event a ____ event b.

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial
- True or false: Any clock will skew over time, causing drift between two clocks
  False! Any clock will **drift** over time, causing **skew** between two clocks
- Because we know the timestamp of event a to be less than the timestamp of event b, we can safely say that event a ____ event b.
  Trick question! We can't say anything based on just the timestamps of these events.

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial
- True or false: Any clock will skew over time, causing drift between two clocks
  False! Any clock will **drift** over time, causing **skew** between two clocks
- Because we know the timestamp of event a to be less than the timestamp of event b, we can safely say that event a ____ event b.
  Trick question! We can't say anything based on just the timestamps of these events.
- Discuss: What is the main limitation of logical time in relation to processes within a system?

Discussion Points

- True or false: a network of computers that communicate about events in a shared process without transmission delay constitute a distributed system
  False! A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process
- Fill in the blank: There is a ____ order in which an event e1 precedes an event e2 iff e1 can causally affect e2. partial
- True or false: Any clock will skew over time, causing drift between two clocks
  False! Any clock will **drift** over time, causing **skew** between two clocks
- Because we know the timestamp of event a to be less than the timestamp of event b, we can safely say that event a ____ event b.
  Trick question! We can't say anything based on just the timestamps of these events.
- Discuss: What is the main limitation of logical time in relation to processes within a system?
- Discuss: Why not just use a centralized scheduler to deal with the mutex problem?

Questions?

Thank you for attending