Logical Time

Etienne Renault

2 octobre 2020

`https://www.lrde.epita.fr/~renault/teaching/algorep/`

Problem Statement 1/3

Problem

How to capture chronological and causal relationships in a distributed system ?

Problem Statement 1/3

Problem

How to capture chronological and causal relationships in a distributed system ?

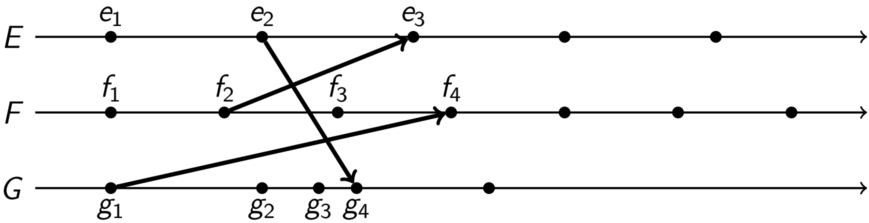
In other words. . .

Given two events e_1 and e_2 : does e_1 happens before e_2 ?

Problem Statement 2/3

Consider 3 processes E, F, and G

- With some local events : e_1, f_1, f_3, g_2, g_3
- With some *send* events : g_1, f_2 and e_2
- With some *receive* events : e_3, f_4 and g_4



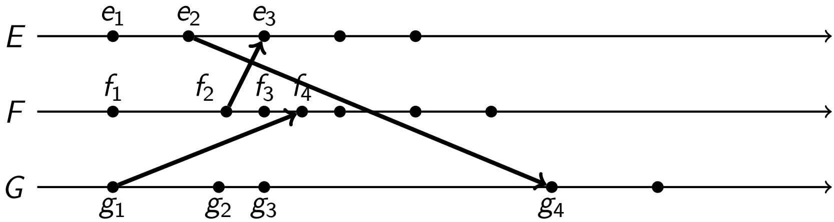
---

1. Thanks to A. Duret-Lutz the canvas used for the figures in this presentation.

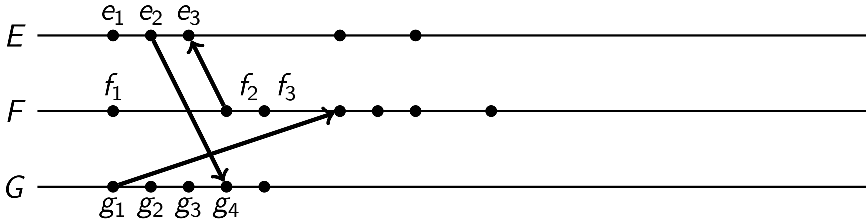
Indistinguishability

For the system the two following executions are indistinguishable.

- First execution



- Second execution



Processes Ordering

Why it is so important to detect the order between events of different processes ?

- Synchronisation between processes
- Global vision of a system state
- Commits
- Mutual Exclusion
- Tracking of dependent events
- Progress of a computation
- Concurrency Measure
- . . .

Definitions

- Consider a time domain T and a logical clock C

Definitions

- Consider a time domain T and a logical clock C
- Elements of T form a partially ordered set over a relation $<$

Definitions

- Consider a time domain T and a logical clock C
- Elements of T form a partially ordered set over a relation $<$
- Relation $\rightarrow$ is called the happened before or causal precedence

Definitions

- Consider a time domain T and a logical clock C
- Elements of T form a partially ordered set over a relation $<$
- Relation $\rightarrow$ is called the happened before or causal precedence
- The logical clock C is a function that maps an event e to an element in T, denoted as $C(e)$ and called the timestamp of e.

  For e_i, e_j two events, $e_i \rightarrow e_j \implies C(e_i) < C(e_j)$

Implementing Logical Clocks

Rule R1

How does a process update the local logical clock when it executes an event ?

Implementing Logical Clocks

Rule R1

How does a process update the local logical clock when it executes an event ?

Rule R2

How does a process update its global logical clock to update its view of the global time and global progress ?

Implementing Logical Clocks

Rule R1

How does a process update the local logical clock when it executes an event ?

Rule R2

How does a process update its global logical clock to update its view of the global time and global progress ?

Systems of logical clocks differ in their representation of logical time and also in the protocol to update the logical clocks.

Lamport Clocks : Informal

- Proposed by Lamport in 1978

Lamport Clocks : Informal

- Proposed by Lamport in 1978
- Attempt to totally order events in a distributed system

Lamport Clocks : Informal

- Proposed by Lamport in 1978
- Attempt to totally order events in a distributed system
- Time domain is the set of non-negative integers

Lamport Clocks : Informal

- Proposed by Lamport in 1978
- Attempt to totally order events in a distributed system
- Time domain is the set of non-negative integers
- The logical local clock of a process p_i and its local view of the global time are squashed into one integer variable C_i

Lamport Clocks : Definition

Rule R1

Before executing an event (send, receive, or internal), process pi executes the following

$$C_i := C_i + d \qquad \text{with } d > 0\text{, typically } 1$$

Lamport Clocks : Definition

Rule R1

Before executing an event (send, receive, or internal), process pi executes the following

$$C_i := C_i + d \qquad \text{with } d > 0\text{, typically } 1$$

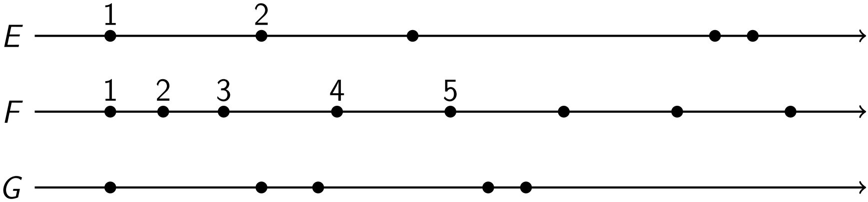
Rule R2

Each message piggybacks the clock value of its sender at sending time.

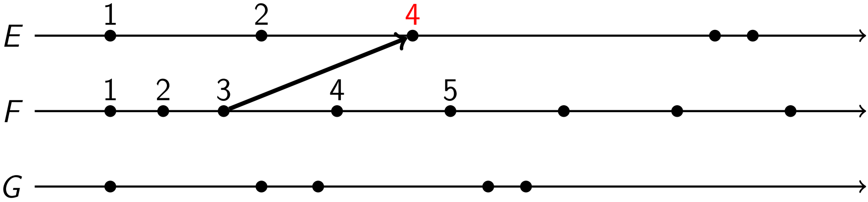
When a process p_i receives a message with timestamp C_{msg}, it executes the following actions :

- $C_i := max(C_i, C_{msg})$
- Execute **R1** and deliver message

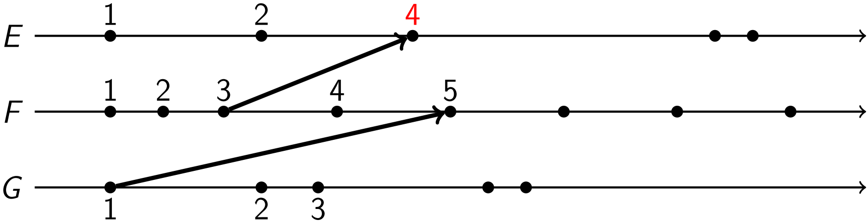
Example



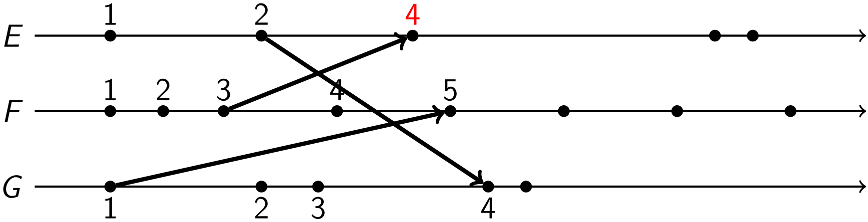
Example



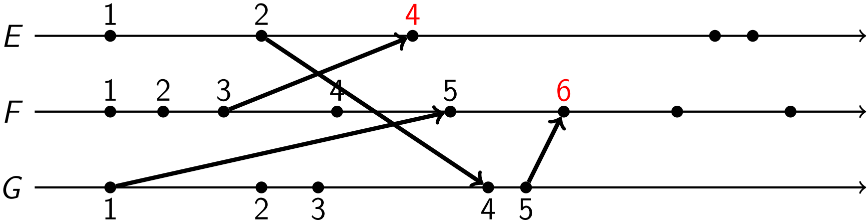
Example



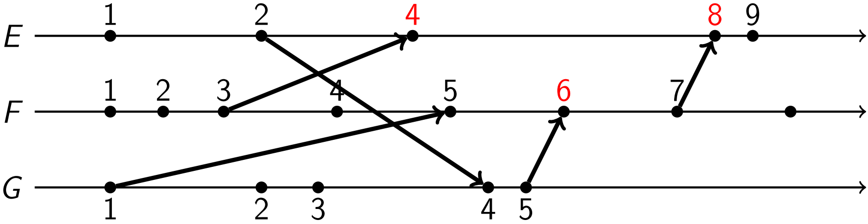
Example



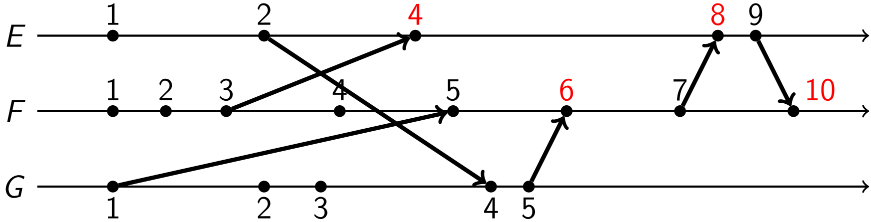
Example



Example



Example



Remarks

- Scalar clocks can be used to totally order events in a distributed system
- If the increment value d is always 1, there is an interesting property : if event e has a timestamp h, then h-1 represents the minimum logical duration, counted in units of events, required before producing the event e
- No Strong Consistency : For e_i, e_j two events,
  $C(e_i) < C(e_j) \not\Longrightarrow e_i \rightarrow e_j$
  The reason that scalar clocks are not strongly consistent is that the logical local clock and logical global clock of a process are squashed into one

Problem

The main problem in totally ordering events is that two or more events at different processes may have identical timestamp !

Mattern Clocks : Informal

- Vector clocks were developed independently by Fidge, Mattern and Schmuck in 1988

Mattern Clocks : Informal

- Vector clocks were developed independently by Fidge, Mattern and Schmuck in 1988

- The time domain is represented by a set of n-dimensional non-negative integer vectors

Mattern Clocks : Informal

- Vector clocks were developed independently by Fidge, Mattern and Schmuck in 1988
- The time domain is represented by a set of n-dimensional non-negative integer vectors
- Each process p_i maintains a vector $vt_i[1..n]$, where $vt_i[i]$ is the local logical clock of p_i and describes the logical time progress at process p_i.

Mattern Clocks : Informal

- Vector clocks were developed independently by Fidge, Mattern and Schmuck in 1988
- The time domain is represented by a set of n-dimensional non-negative integer vectors
- Each process p_i maintains a vector $vt_i[1..n]$, where $vt_i[i]$ is the local logical clock of p_i and describes the logical time progress at process p_i.
- The entire vector vt_i constitutes p_i's view of the global logical time and is used to timestamp events

Mattern Clocks : Definition

Rule R1

Before executing an event (send, receive, or internal), process p_i executes the following

$$vt_i[i] := vt_i[i] + d \qquad \text{with } d > 0\text{, typically } 1$$

Mattern Clocks : Definition

Rule R1

Before executing an event (send, receive, or internal), process p_i executes the following

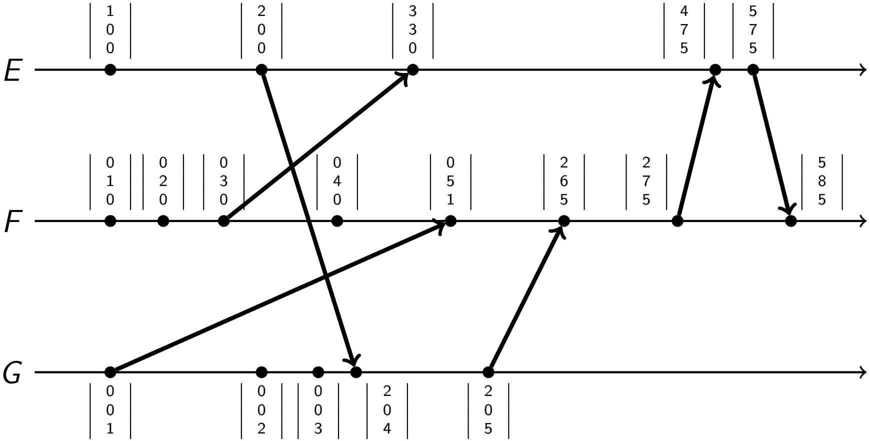
$$vt_i[i] := vt_i[i] + d \qquad \text{with } d > 0\text{, typically } 1$$

Rule R2

Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m, vt), process p_i executes the following sequence of actions :

- $\forall k \in [1, n], vt_i[k] := max(vt_i[i], vt[i])$
- Execute **R1** and deliver message

Example


E
F
G
(1,0,0)
(2,0,0)
(3,3,0)
(4,7,5)
(5,7,5)
(0,1,0)
(0,2,0)
(0,3,0)
(0,4,0)
(0,5,1)
(2,6,5)
(2,7,5)
(5,8,5)
(0,0,1)
(0,0,2)
(0,0,3)
(2,0,4)
(2,0,5)

Remarks

- The following relations are defined to compare two vector timestamps, *vh* and *vk* :

$$\begin{aligned} vh = vt \Leftrightarrow & \quad \forall x, vh[x] = vt[x] \\ vh \leq vt \Leftrightarrow & \quad \forall x, vh[x] \leq vt[x] \\ vh < vt \Leftrightarrow & \quad vh[x] \leq vt[x] \wedge \exists x, vh[x] < vt[x] \\ vh \mid\mid vt \Leftrightarrow & \quad \neg(vh[x] < vt[x]) \wedge \neg(vt[x] < vh[x]) \end{aligned}$$

Remarks

- The following relations are defined to compare two vector timestamps, vh and vk :

$$\begin{aligned} vh = vt \Leftrightarrow & \quad \forall x, vh[x] = vt[x] \\ vh \leq vt \Leftrightarrow & \quad \forall x, vh[x] \leq vt[x] \\ vh < vt \Leftrightarrow & \quad vh[x] \leq vt[x] \wedge \exists x, vh[x] < vt[x] \\ vh \mid\mid vt \Leftrightarrow & \quad \neg(vh[x] < vt[x]) \wedge \neg(vt[x] < vh[x]) \end{aligned}$$

- Strong consistency by examining the vector timestamp of two events, we can determine if the events are causally related.

Remarks

- The following relations are defined to compare two vector timestamps, *vh* and *vk* :

$$\begin{aligned}
vh = vt \Leftrightarrow & \quad \forall x, vh[x] = vt[x] \\
vh \leq vt \Leftrightarrow & \quad \forall x, vh[x] \leq vt[x] \\
vh < vt \Leftrightarrow & \quad vh[x] \leq vt[x] \land \exists x, vh[x] < vt[x] \\
vh \mid\mid vt \Leftrightarrow & \quad \neg(vh[x] < vt[x]) \land \neg(vt[x] < vh[x])
\end{aligned}$$

- Strong consistency by examining the vector timestamp of two events, we can determine if the events are causally related.

- If the number of processes in a distributed computation is large, then vector clocks will require piggybacking of huge amount of information in messages.

Efficient Implementation of Vector Clocks

- Singhal-Kshemkalyani's Differential Technique 1996
- Based on the observation that between successive message sends to the same process, only a few entries of the vector clock at the sender process are likely to change.
- When a process p_i sends a message to a process p_j , it piggybacks only those entries of its vector clock that differ since the last message sent to p_j.
- Implementation of this technique requires each process to remember the vector timestamp in the message last sent to every other process.

Problem

How to capture chronological and causal relationships in a distributed system ?

In other words

What if channels are reordering channels ?

1. Scalar Time : Lamport Clocks

2. Vector Time : Mattern Clocks

3. **Matrix Time**

4. Virtual Time System

5. Conclusion

Matrix Clocks : Informal

- Time is represented by a set of n x n matrices of non-negative integers

Matrix Clocks : Informal

- Time is represented by a set of n x n matrices of non-negative integers
- A process p_i maintains a matrix $mt_i[1..n][1..n]$

Matrix Clocks : Informal

- Time is represented by a set of n x n matrices of non-negative integers

- A process p_i maintains a matrix $mt_i[1..n][1..n]$
    - $mt_i[i][i]$ denotes the local logical clock of p_i and tracks the progress of the computation at process pi

Matrix Clocks : Informal

- Time is represented by a set of n x n matrices of non-negative integers
- A process p_i maintains a matrix $mt_i[1..n][1..n]$
  - $mt_i[i][i]$ denotes the local logical clock of p_i and tracks the progress of the computation at process pi
  - $mt_i[i][j]$ denotes the latest knowledge that process p_i has about the local logical clock, $mt_j[j, j]$, of process p_j

Matrix Clocks : Informal

- Time is represented by a set of n x n matrices of non-negative integers

- A process p_i maintains a matrix $mt_i[1..n][1..n]$
  - $mt_i[i][i]$ denotes the local logical clock of p_i and tracks the progress of the computation at process pi
  - $mt_i[i][j]$ denotes the latest knowledge that process p_i has about the local logical clock, $mt_j[j, j]$, of process p_j
  - $mt_i[j][k]$ represents the knowledge that process pi has about the latest knowledge that p_j has about the local logical clock, $mt_k[k][k]$, of p_k

Matrix Clocks : Informal

- Time is represented by a set of n x n matrices of non-negative integers
- A process p_i maintains a matrix $mt_i[1..n][1..n]$
  - $mt_i[i][i]$ denotes the local logical clock of p_i and tracks the progress of the computation at process pi
  - $mt_i[i][j]$ denotes the latest knowledge that process p_i has about the local logical clock, $mt_j[j,j]$, of process p_j
  - $mt_i[j][k]$ represents the knowledge that process pi has about the latest knowledge that p_j has about the local logical clock, $mt_k[k][k]$, of p_k
- The entire matrix mt_i denotes p_i's local view of the global logical time

Matrix Clocks : Definition

Rule R1

Before executing an event (send, receive, or internal), process p_i executes the following

$$mt_i[i][i] := mt_i[i][i] + d \qquad \text{with } d > 0\text{, typically } 1$$

Matrix Clocks : Definition

Rule R1

Before executing an event (send, receive, or internal), process p_i executes the following

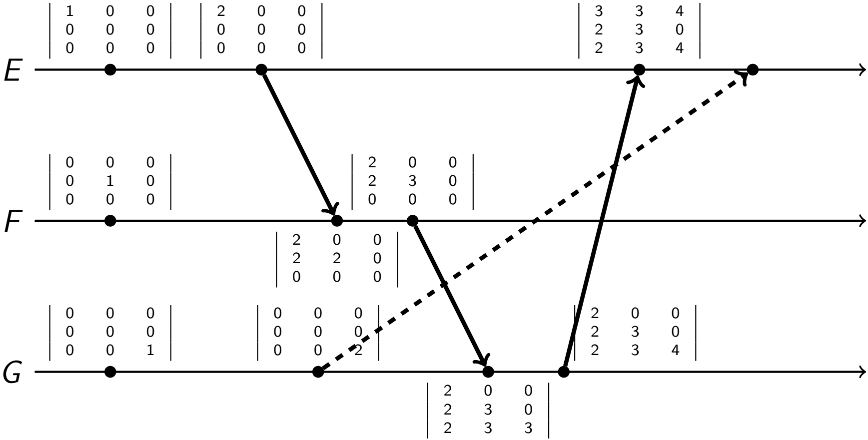
$$mt_i[i][i] := mt_i[i][i] + d \qquad \text{with } d > 0 \text{, typically } 1$$

Rule R2

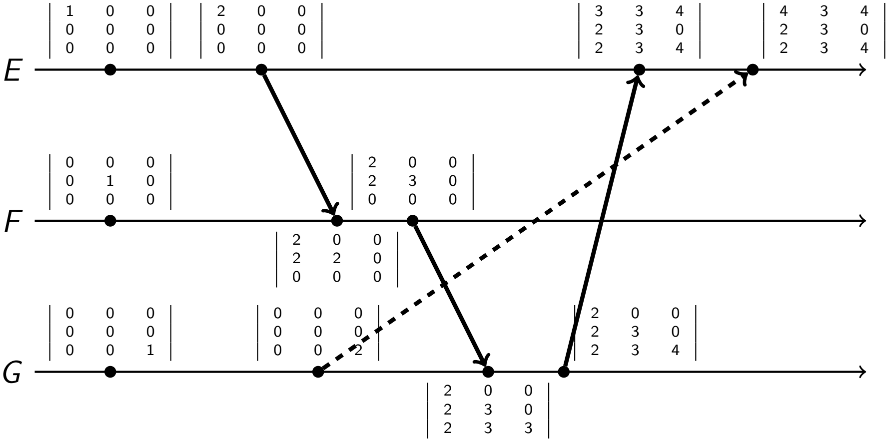
Each message m is piggybacked with the matrix clock mt. When p_i receive a message (m, mt) from a process p_j , p_i executes the following sequence of actions :

- $\forall k \in [1, n], mt_i[i][k] := max(mt_i[i][k], mt[j][k])$
- $\forall k, \ell \in [1, n]^2, mt_i[k][\ell] := max(mt_i[k][\ell], mt[k][\ell])$
- Execute **R1** and deliver message

Example


E
F
G
1 0 0
0 0 0
0 0 0
2 0 0
0 0 0
0 0 0
3 3 4
2 3 0
2 3 4
0 0 0
0 1 0
0 0 0
2 0 0
2 3 0
0 0 0
2 0 0
2 2 0
0 0 0
0 0 0
0 0 0
0 0 1
0 0 0
0 0 0
0 0 2
2 0 0
2 3 0
2 3 4
2 0 0
2 3 0
2 3 3

Example


1 0 0
0 0 0
0 0 0
2 0 0
0 0 0
0 0 0
3 3 4
2 3 0
2 3 4
4 3 4
2 3 0
2 3 4
E
0 0 0
0 1 0
0 0 0
2 0 0
2 3 0
0 0 0
F
2 0 0
2 2 0
0 0 0
0 0 0
0 0 0
0 0 1
0 0 0
0 0 0
0 0 2
2 0 0
2 3 0
2 3 4
G
2 0 0
2 3 0
2 3 3

Remarks

Important properties

$min_k(mt_i[k][\ell]) \geq t \implies$ process pi knows that every other process p_k knows that p_ℓ's local time has progressed till t.

Virtual Time System

Virtual time system is an (**optimistic**) paradigm for organizing and synchronizing distributed systems

- Relies on Time Warp mechanism, i.e. lookahead-rollback mechanism
- When a conflict is discovered, the offending processes are rolled back to the timejust before the conflict
- Processes are then executed forward along the **revised path**

Description

Virtual time is a global, one dimensional, temporal coordinate system on a distributed computation to measure the computational progress and to defines ynchronization.

- Virtual time is implemented a collection of several loosely synchronized local virtual clocks
- These local virtual clocks move forward to higher virtual times ; however, **occasionaly they move backwards**
- Virtual time systems are subject to two semantic rules similar to Lamport'sclock conditions :
  - Virtual send time of each message $<$ virtual receive time of that message.
  - Virtual time of each event in a process $<$ Virtual time of next event in that process.

Comparison with Lamport's Logical Clocks

In Lamport's logical clock, an artificial clock is created one for each process with unique labels from a totally ordered set in a manner consistent with partial order.

In virtual time, the reverse of the above is done by assuming that every event is labeled with a clock value from a totally ordered virtual time scale satisfying Lamport's clock conditions

Time Warp mechanism is an inverse of Lamport's scheme
A process advances its clock as soon as it learns of new causal dependency

Time Warp Mechanism

The Time warp mechanism assumes that message communication is reliable, and messages may not be delivered in FIFO order

Two major parts :

- **The local control mechanism** insures that events are executed and messages are processed in the correct order
- **The global control mechanism** takes care of global issues such as global progress, termination detection, I/O error handling, flow control, etc.
  not discussed here

The Local Control Mechanism

Each process maintain :

- A **local virtual clock** : Virtual spaces coordinate
- A **state queue** : contains saved copies of process's recent states
- An **input queue** : contains all recently arrived messages in order of virtual receive time
- An **output queue** : contains negative copies of messages the process hasrecently sent in virtual send time order (set of **antimessages**)

The Rollback Mechanism 1/2

1. When a process sends a message, a copy is retained in its own output queue
2. When a message is received (with timestamp T) :
    - If timestamp greater than virtual clock time of receiver, the message is enqueued
    - Otherwise, the process must do a rollback

Rollback procedure :

- Search in the State Queue for the last saved state with timestamp that is lessthan T
- Set current timestamp to T
- Unsent all messages, by transmitting antimessages

The Rollback Mechanism 2/2

When a process receive an antimessage :

- If message has not yet been processed ⇒ no rollback, just remove positive message
- Otherwise the negative message causes the receiver to roll back to a virtual time when the positive message was received

Domino Effect ?

In the worst case, all processes in system roll back to same virtual time asoriginal one did and then proceed forward again

Conclusion

- Different kind of logical time
    - Scalar : Global Order, not causal
    - Vector : Strong Consistency, Causal Dependency
    - Matrix : Causality and Chronology

- Virtual time system (Jefferson) is a paradigm for organizing and synchronizing distributed systems
    - If a conflict is discovered, the offending processes are rolled back to the time just before the conflict and executed forward along the revised path.