Cours 2

2. Le temps et l'état global dans un système réparti

- **2.1 Le temps physique et la synchronisation des horloges.**
- **2.2 Le temps logique et l'ordre.**
- 2.3. Applications de la datation logique
  - 2.3.1 problèmes de synchronisation
    - Exclusion Mutuel:
      - Algorithme de Leslie Lamport
      - Algorithme de Glenn & Agrawala
      - D'autres algorithmes
  - 2.3. 2 État global d'un système réparti
    - Passé et coupures cohérentes
    - Détermination d'un état global cohérent (Chandy - Lamport)
  - 2.3.3 Applications Distribués

Introduction

- Time is often required to :
  - **measure delays** between distributed components,
  - synchronize streams (e.g., sound and vision),
  - detect event ordering for casual analysis,
  - and many utilities and applications use **timestamps** (e.g., in SSL).

Introduction

- We say that **event *a*** *occurred before* **event *b*** *if the physical time at which the event* ***a*** *happened,* ***ta****, is less than* ***tb****, the* time at which ***b*** *happened.*
- *The use of this ordering requires access to some clocking* mechanism which can record ***ta*** *and* ***tb****.*
- ***In the absence of precisely synchronized*** clocks in distributed systems, this is a hard task.
- This leads us to consider ordering relationships which are **useful and implementable** in distributed systems.

UNIX make programm example

the UNIX *make* program, as a single example. Normally, in UNIX, large programs are split up into multiple source files, so that a change to one source file only requires one file to be recompiled, not all the files. If a program consists of 100 files, not having to recompile everything because one file has been changed greatly increases the speed at which programmers can work.

Comment ?

The way *make* norm the programmer has finished changing all the so ich examines the times at which all the source and object files were last modified. If the source file *input.c* has time 2151 and the corresponding object file *input.o* has time 2150, *make* knows that *input.c* has been changed since *input.o* was created, and thus *input.c* must be recompiled. On the other hand, if *output.c* has time 2144 and *output.o* has time 2145, no compilation is needed here. Thus *make* goes through all the source files to find out which ones need to be recompiled and calls the compiler to recompile them.

UNIX make program example

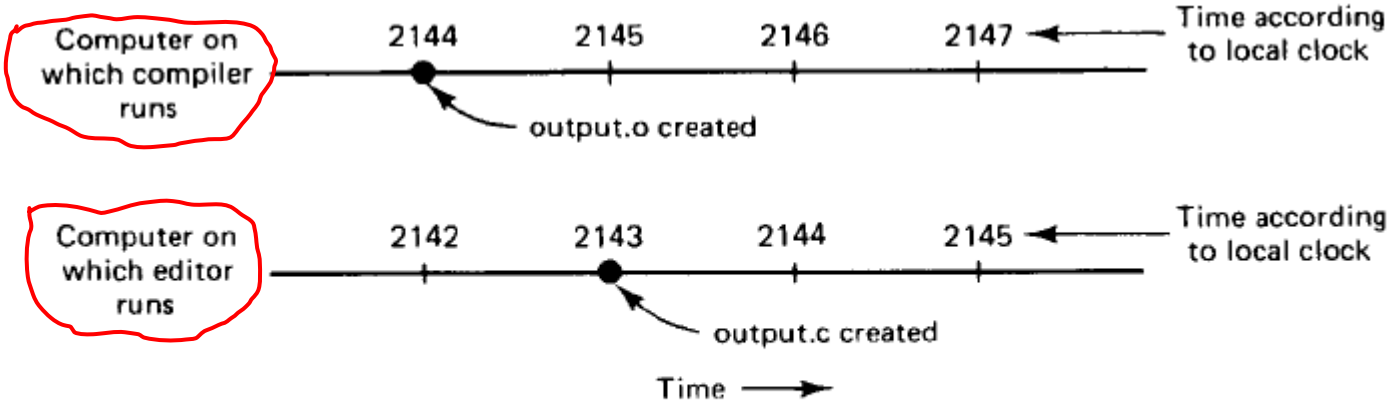


Fig. 3-1. When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Now imagine what could happen in a distributed system in which there is no global agreement on time. Suppose that *output.o* has time 2144 as above, and shortly thereafter *output.c* is modified but is assigned time 2143 because the clock on its machine is slightly slow, as shown in Fig. 3-1. *Make* will not call the compiler. The resulting executable binary program will then contain a mixture of object files from the old sources and the new sources. It will probably not work, and the programmer will go crazy trying to understand what is wrong with the code.

Solution?

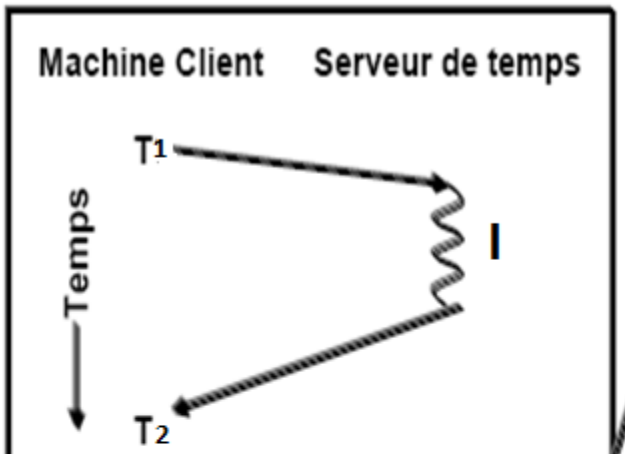
Solution

- Il faut trouvez des mécanismes correctes pour dater les événements dans un système distribué.
  1. Mécanismes basés sur le temps physiques des horloges.
  2. Mécanismes basés sur le temps logique.

2.1 Le temps physique et la synchronisation des horloges.

- **Besoin pour les applications « temps réel »**
- **Solutions:**
  - **Synchronisation avec l'horloge universelle**
  - **Synchronisation entre les horloges**

Algorithmes de Synchronisation d'horloges Physiques
1. Algorithme de Cristian (1989)



Chaque machine interroge périodiquement le serveur de temps pour avoir l 'heure courante afin de pouvoir corriger d 'une manière adéquate son horloge

La correction d 'une horloge doit se faire graduellement. On retarde (ou accélère) une horloge rapide (ou lente) en diminuant (ou ajoutant) un certain nombre de ns de la (au) constante de temps utilisée pour avancer l 'horloge logique dans la routine d 'interruption. Cette correction a pour une durée nécessaire correspondant au décalage de l 'horloge locale par rapport le C_{TUC}

La machine Client doit tenir compte le temps de propagation des messages entre lui et le serveur de temps. La valeur de correction est

$$C_{cor} = (T2-T1)/2 + C_{TUC}$$

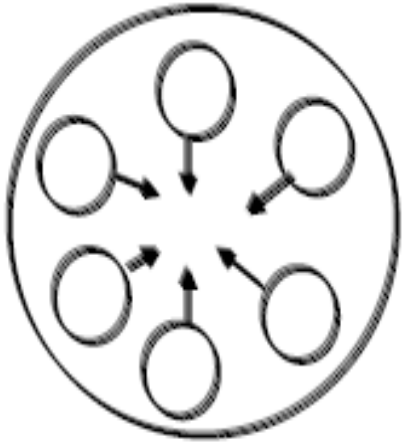
Si le temps de gestion l 'interruption I sur serveur est connu, on peut le tenir compte dans la formule :

$$C_{cor} = (T2-T1-I)/2 + C_{TUC}$$

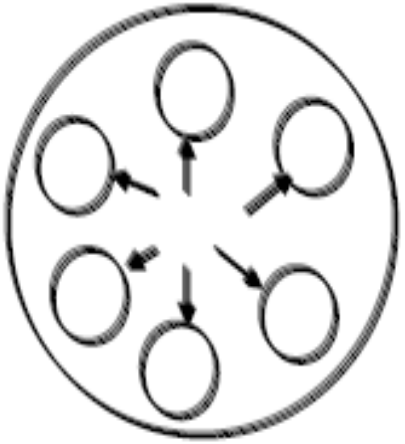
Algorithmes de Synchronisation d'horloges Physiques
1. Algorithme de Cristian (1989)

- Application : **Network Time Protocol (NTP).**
- **NTP** est un protocole standard de synchronisation d'horloges utilisé dans les réseaux informatiques depuis les années 1980 (RFC 958, puis RFC 5905).
  Il permet à des ordinateurs distribués de **synchroniser leur heure locale avec une référence de temps universelle (UTC)**.

Algorithmes de Synchronisation d'horloges Physiques
2. Algorithme basé sur la moyenne



Chaque machine diffuse son heure au début d 'une période de temps [T0+iR, T0+(i+1)R]



Chaque machine démarre un temporisateur et collecte tous les messages arrivés durant l 'intervalle S

Chaque machine exécute un même algorithme à partir des données reçu pour l 'obtention d 'une heure moyenne

Résultats

- Lack of global time, that is, it is difficult for two machines to agree what the current time is.
- At any instance in a distributed system however, indepen nes will be differen tially). This differen d a typical rate is 1 in 10^{-6}, or 1 second in 11.6 days
- Une solution c'est le temps logique défini par Leslie Lamport "L. Lamport. Time, clocks, and the ordering of events in a distributed system. ***Communications of the ACM**, 21 (7):558-565, July 1978".*

https://cacm.acm.org/

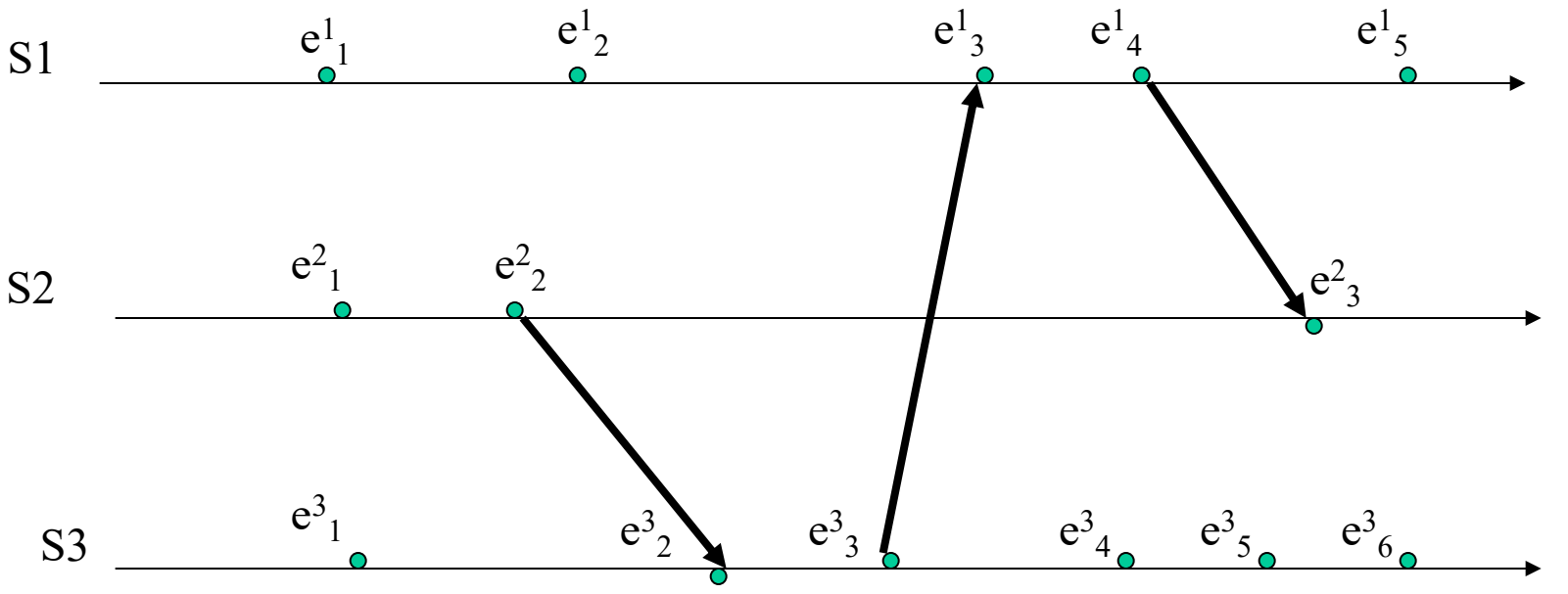
Temps logique (Adapté)

Prof. Nadjib Badache

USTHB, Alger.

Notions de base

1. Site.
2. Evénement: interne, émission ou réception d'un message
3. Message.
4. Canal : liaison point à point entre un émetteur et un récepteur.



Introduction

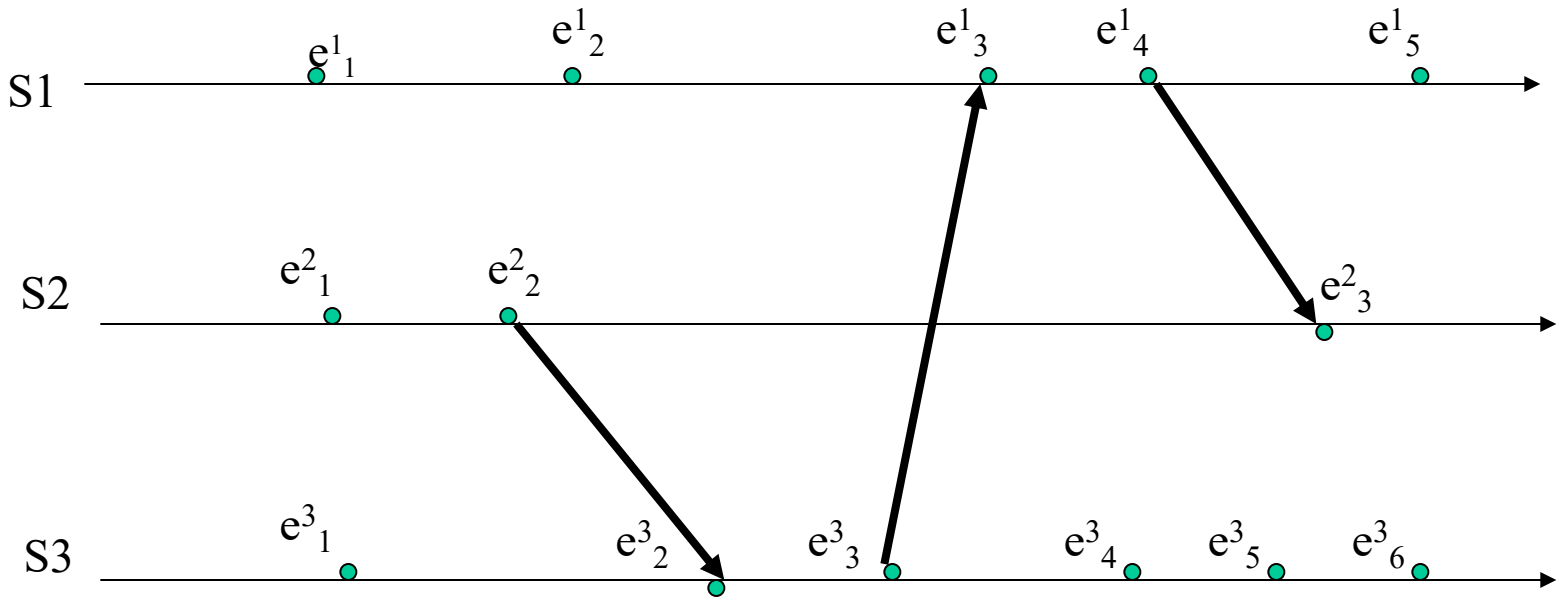
Dans un système réparti composé de n sites connectés par des canaux :

- Les événements sur un même site sont totalement ordonnés (émission, réception, événements interne).
  - -A a précédé B si A et B sont des événements qui ont été générés dans cet ordre sur le même site S (ordre d'exécution local) A→B
- Pour chaque message, l'événement d'émission précède toujours sa réception.
  - -A a précédé B si A est l'événement d'émission d'un message M par le site P et que B est l'événement de réception du message M sur Q : A → B (ordre causal pour chaque message)
- La relation de précédence dans un système réparti est la fermeture transitive des deux relations précédentes.

si A → B et B → C alors A → C

- La précédence est un ordre partiel entre les événements du système réparti.
- La causalité entre événements implique la précédence entre eux :
- A **cause** B ==> A → B
- Une précédence entre événements exprime une causalité potentielle :
- (si A → B, A peut avoir influencé B)

Diagramme d'une exécution répartie.



$e^1_1 \rightarrow e^1_2$ $\quad$ $e^2_2 \rightarrow e^3_2$

$e^1_2 \rightarrow e^1_3$ $\quad$ $e^1_1 \,||\, e^2_1$

Deux événements x et y tels que ni $x \rightarrow y$ ni $y \rightarrow x$ sont dits Indépendants ou concurrents. On notera $x \,||\, y$.

Problème :

Trouver des mécanismes qui permettent d'associer des dates aux événements concernés tels que si :

$a \rightarrow b$ alors la date associée à *b* doit être relativement à un temps logique global, après la date associée à *a*.
(propriété de monotonie)

Mécanismes d'estampillage

❑ **Deux mécanismes d'estampillage essentiels :**

- Temps linéaire (Lamport-78)
  - Temps logique représenté par un entier
- Temps vectoriel (Mattern-89, Fidge-89)
  - Temps représenté par un vecteur de dimension n

❑ **Modèle**

Ces deux mécanismes obéissent au même modèle :

- **Structure de données pour représenter le temps**

A chaque site sont associées des variables qui lui permettent :

- **Mesurer sa propre progression, ce qui est assuré par son horloge logique locale (mise à jour par la règle1)**

- **Avoir une bonne représentation du temps logique global; c'est une vue locale du temps global (mise à jour par la règle2).**

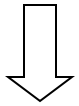
➢ Protocole

Assure que l'horloge logique locale et la vue locale du temps Global de chaque site sont gérées de manière cohérente relativement à la relation $\rightarrow$.

- Règle 1: **Avant de produire un événement (émission, réception, événement interne), un site doit incrémenter son horloge locale (parce qu'il progresse).**

• Règle 2:

Pour que la date (estampille) correspondant à la réception d'un message soit après la date correspondante à l'événement émission du même message, ⇒ chaque message m transporte la valeur du temps logique global vu par l'émetteur Au moment de l'émission.



Ceci permet au récepteur de mettre à jour sa vue du temps global. Puis il exécute la règle 1.

Temps linéaire

❑ **Protocole**

A chaque site est associée une variable entière h_i ayant des valeurs croissantes.

Règle 1: **Avant de produire un événement (émission, réception, interne)**
Faire :

$$h_i := h_i + d \ (d > 0)$$

*% Chaque fois que cette règle est exécutée, d **peut** avoir une valeur*
% différente.

Règle 2: **Lorsqu'un site S_i reçoit un message *(m,h)***
Faire :

$$h_i := max(h_i, h)$$

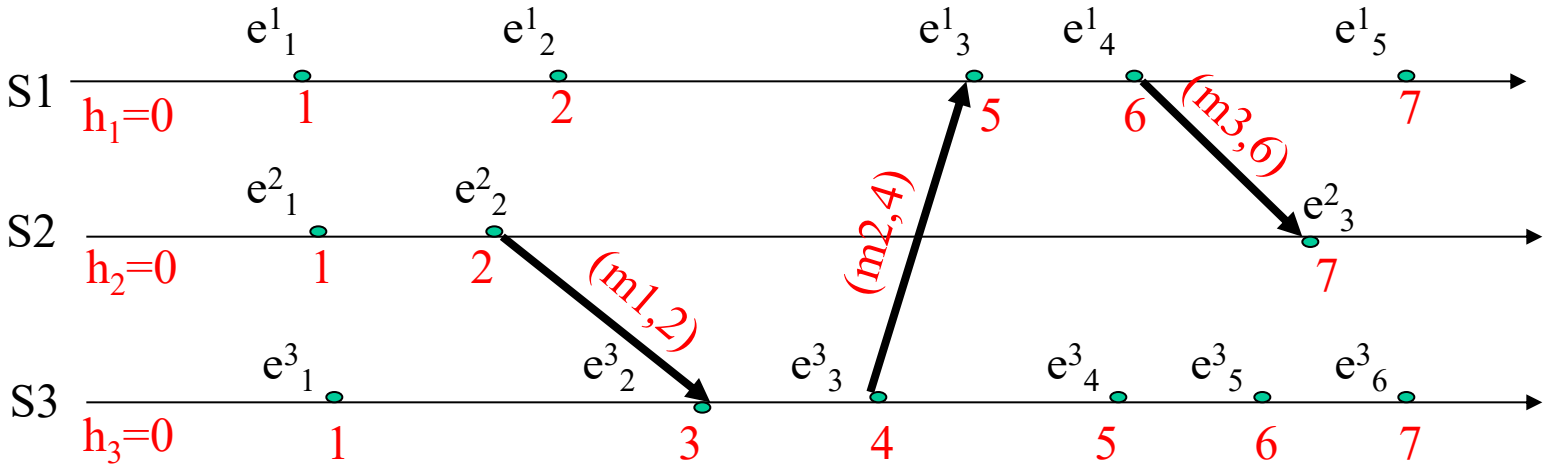
Puis exécuter la règle 1.

❖ Propriétés

Il est possible d'utiliser ce mécanisme d'estampillage pour construire un ordre total noté $\rightarrow^{T}$.

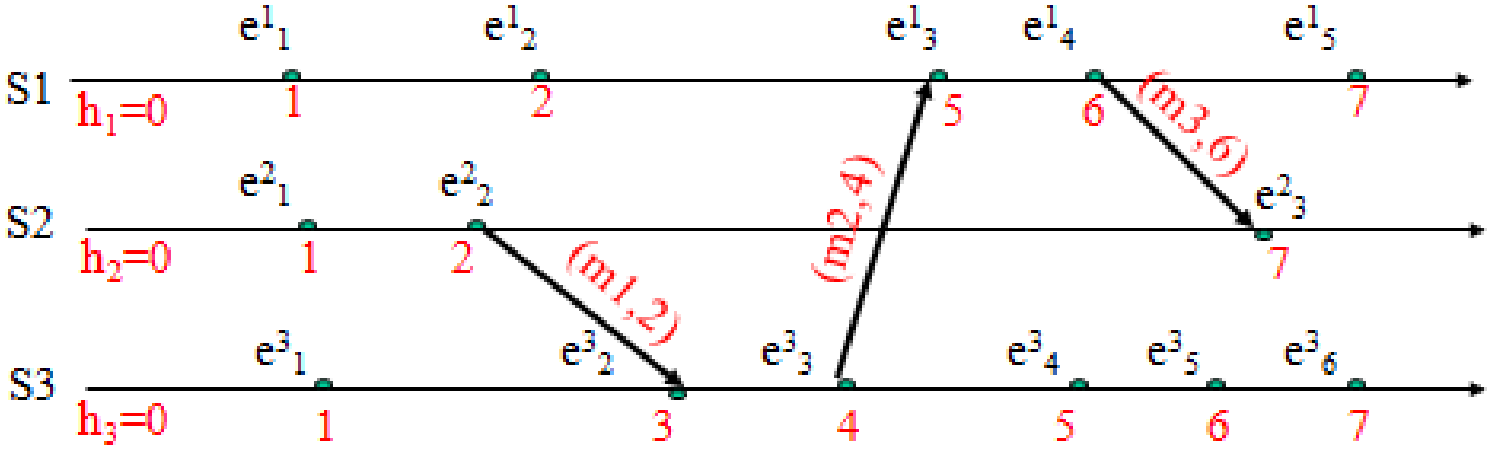
- **Estampille (e) ≡ date d'occurrence + identité du site qui a produit e.**
- **Si on a deux événements x et y estampillés respectivement par *(h,i)* et *(k,j)*, on définit :**

$$x \rightarrow^{T} y \Leftrightarrow (h < k \text{ ou } (h = k \text{ et } i < j))$$



Limite de l'horloge de Lamport

- **Lamport *timestamps*** garantissent seulement : si a → b (causalité), alors H(a) < H(b).
- Mais la réciproque n'est pas vraie : H(a) < H(b) **ne veut pas dire** que a → b.
- Exemple : H(e12)=2<H(e32)=3 mais e12 ne précède pas e32.



Horloges vectorielles

- Le temps est ici représenté par un vecteur de dimension : n.

❑ **Protocole**

Chaque site S_i est doté d'un vecteur $vt_i[1..n]$ où :

- $vt_i[i]$; **décrit l'évolution du temps logique du site S_i : l'horloge logique locale de S_i. Elle prend des valeurs croissantes générées localement**.

- $vt_i[j]$: **Représente la connaissance qu'a le site S_i sur l'évolution du temps sur le site S_j. C'est une image locale de la valeur $Vt_j[j]$.**

- **Le vecteur vt_i constitue une vue locale du temps logique global utilisé pour estampiller les événements**.

Règle 1:

Avant de produire un événement
Faire :

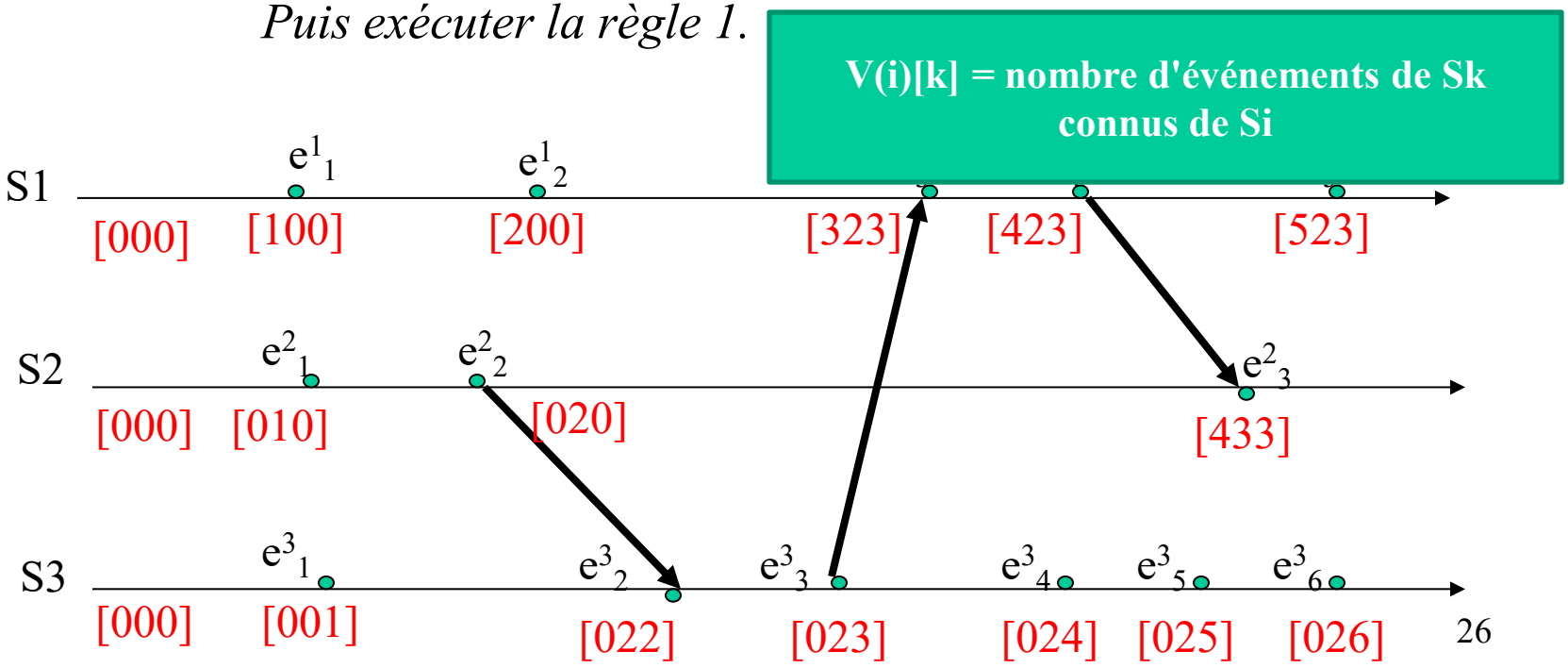
$$Vt_i[i] := vt_i[i] + d \quad (d > 0)$$

fait

Règle 2:

A la réception de *(m,vh)*
Faire :

Pour k de 1 à n faire vti[k] := max(vti[k],vh[k])
Puis exécuter la règle 1.



❑ Propriétés

- $vh \leq vk \Leftrightarrow \forall x : vh[x] \leq vk[x]$
- $vh < vk \Leftrightarrow vh \leq vk$ et $\exists x : vh[x] < vk[x]$
- $vh \mid\mid vk \Leftrightarrow \neg(vh < vk)$ et $\neg(vk < vh)$

- Si on a deux événements x et y estampillés respectivement vh et vk alors :

$x \rightarrow y \Leftrightarrow vh < vk$
$x \mid\mid y \Leftrightarrow vh \mid\mid vk$