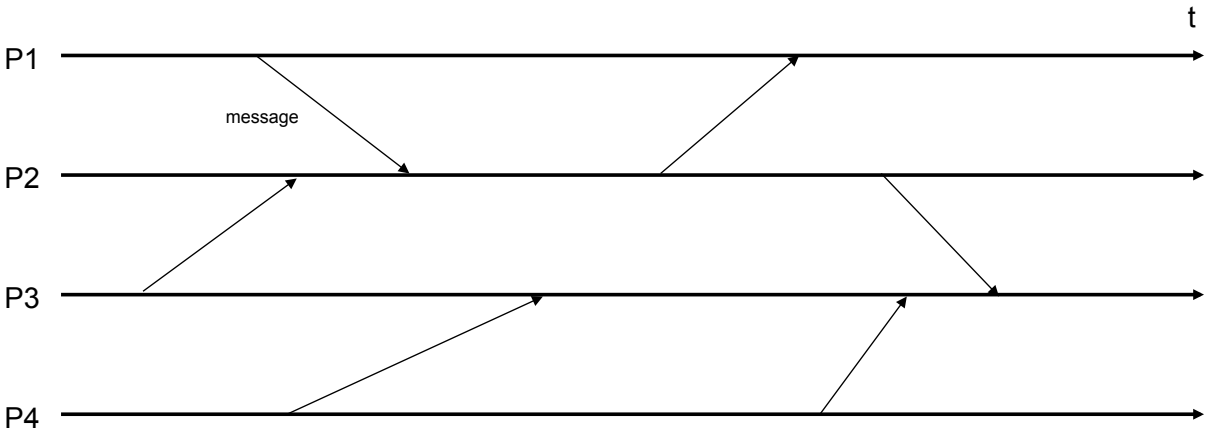
# Gestion du temps dans les applications réparties

# Plan

- **Relation de précédence**
  - Principe de la causalité
  - Consistance des horloges
  - Forte consistance (caractérisation de la causalité)
- **Modèles d'horloges**
  - Scalaires (Lamport)
  - Vectorielles (Mattern / Fidge)
- **Implémentation d'horloges vectorielles**
  - Singhal et Kshemkalyani
- **Vecteur de Dépendance**
  - Fowler et Zwaenepoel
- **Horloge matricielles**
  - Fisher et Michael
- **Causalité entre les messages**

# Contexte (1)

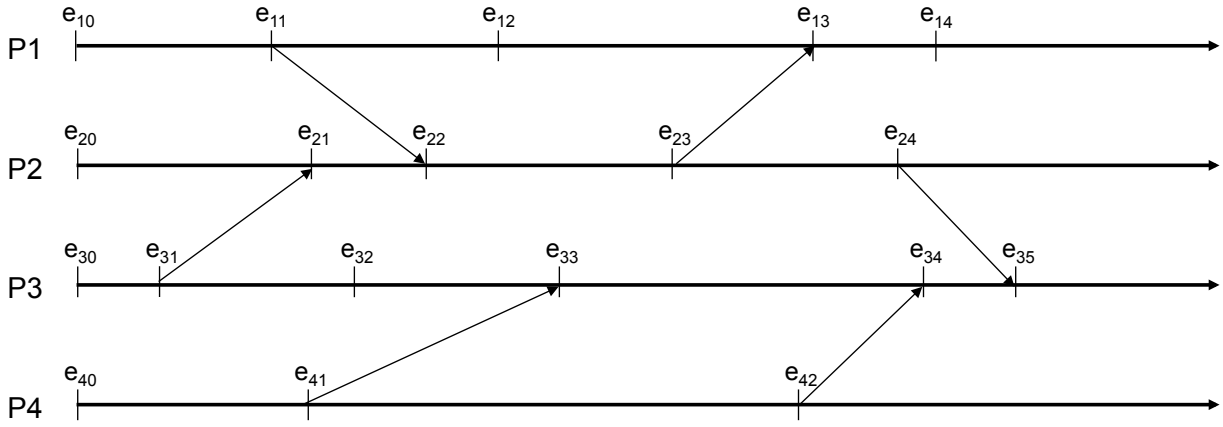
On considère un système réparti comme un ensemble de **processus asynchrones** s'exécutant sur différents sites. Pas d'horloge globale. Les processus communiquent **uniquement par message** (pas de mémoire commune). Les délais de communication sont finis, mais **non connus** et **fluctuants (modèle asynchrone).**




# Contexte (2)

Pour étudier le système réparti, on s'intéresse à la succession des **événements** se produisant sur les différents processus. On définit trois types d'événements :

- les événements locaux (changement de l'état interne d'un processus)
- les émissions de message
- les réceptions de message



# Le temps

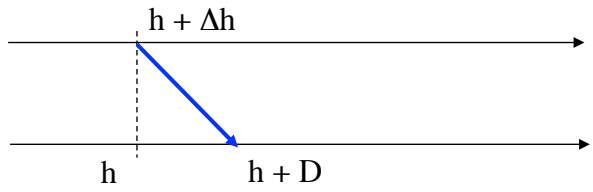
- **A-t-on besoin du temps ? On veut pouvoir**
  - Tracer une exécution
    - debug
  - ...

  Relation de causalité entre événements

- **Pourquoi ne peut-on pas utiliser le temps physique ?**

Δh : dérive

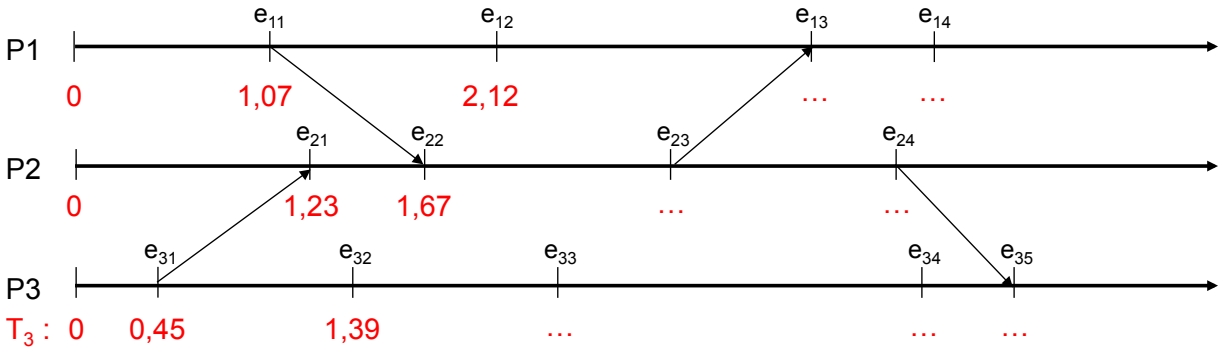
D : délai de transmission




**et si Δh > D ?**

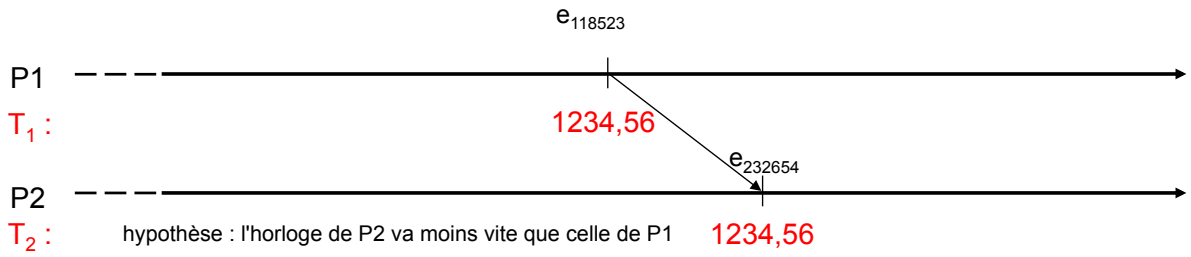
# Horloge Physique

Les événements sont datés à l'aide de l'horloge physique de chaque site $S_i$. On suppose que :
- toutes les horloges sont parfaitement synchronisées au départ,
- leur résolution est suffisante pour dater distinctement tous les événements locaux.



# Limites Horloges Physiques

Les horloges physiques dérivent au cours du temps, et le sens et l'amplitude de la dérive est propre à chaque machine. Les événements locaux restent correctement ordonnés par leur date, mais pas nécessairement ceux se produisant sur des machines distinctes. Par conséquent, la datation par horloge physique ne tient pas compte de la causalité des événements non locaux.




Conséquence : la précédence causale n'est pas respectée.
$e_{118523} \rightarrow e_{232654}$ et pourtant $T(e_{118523}) = T(e_{232654})$

# Relation de Précédence : Causalité

Principe de la causalité : la cause précède l'effet.

Pour étudier la causalité entre les événements, on définit la relation de **précédence causale**, notée →, traduisant une causalité **potentielle** entre les événements.

**Définition**

**a** → **b** si et seulement si :
- **a** et **b** sont des événements du **même processus** et **a** a lieu **avant b**
- **a** est l'**envoi** d'un message *m* et **b** est la **réception** de *m*
- il existe **c** tel que **a** → **c** et **c** → **b** (relation transitive)

a → b

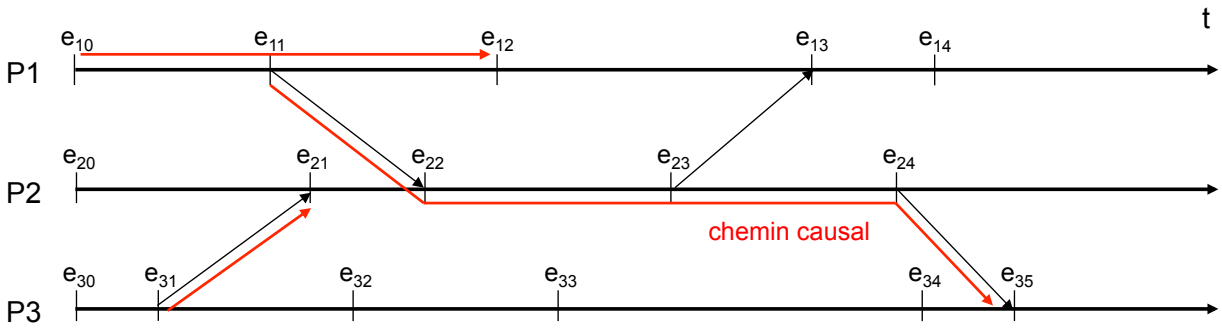
**a** *a potentiellement causé* **b**

**a** *précède causalement* **b**

**a** *happened before* **b**

# Relation de Précédence: causalité

Deux événements qui ne sont pas causalement dépendants l'un de l'autre sont dits **concurrents**, noté || : e || e' ⇔ e ↛ e'et e' ↛ e




Relations :

$e_{10} \rightarrow e_{12}$
$e_{31} \rightarrow e_{21}$
$e_{11} \rightarrow e_{35}$

$e_{12} \parallel e_{22}$
$e_{32} \parallel e_{23}$
$e_{33} \parallel e_{14}$

# Précédence causale : e → f

- **$e_i^n$ :** $n^{ème}$ événement sur le site i

**Definition (Lamport 78)**

**$e_i^n \rightarrow e_j^m$ ( $e_i^n$ précède causalement $e_j^m$ ) ssi :**

- i = j et n < m : précédence locale

ou

- **∃ mes, $e_i^n$ = send(mes) et $e_j^m$ = receive(mes)** : précédence par message

ou

- ∃ $e_k^p$ , $e_i^n \rightarrow e_k^p$ et $e_k^p \rightarrow e_j^m$ : relation transitive

• l'ordre défini est **partiel** (existence d'événements ||)

# Datation

A chaque événement on associe sa date d'occurrence :

$a \xmapsto{D} D(a)$

**Objectif** : trouver un système de datation D qui **respecte** et **représente** au mieux la relation de précédente causale.

- respecte : $(a \rightarrow b) \Rightarrow D(a) < D(b)$ exigence minimale, naturelle
- représente : $(a \rightarrow b) \Leftarrow D(a) < D(b)$ plus difficile à obtenir

# Définition d'une horloge logique

- **H :** ensemble des événements de l'application, muni de l'ordre partiel $\rightarrow$

  si ( e $\not\rightarrow$ e' et e' $\not\rightarrow$ e ) alors **e || e'** (**concurrence**)

- **T :** domaine de temps, muni de l'ordre partiel $<$

- **Horloge logique :**

  **C** : $H \rightarrow T$

  $e \rightarrow C(e)$

  tel que $e \rightarrow e' \Rightarrow C(e) < C(e')$ **consistance (simple)***

* L'horloge capture la causalité

# Causalité et horloges logiques

- **On utilise souvent les horloges pour reconstruire (en partie) la relation de causalité**

- **C(e) < C(e') ⇒ ?**
  - Soit e → e'
  - Soit e || e'
- **Si**

  **C(e) < C(e') ⇒ e → e' ( consistance forte )**

  les horloges *caractérisent* la causalité

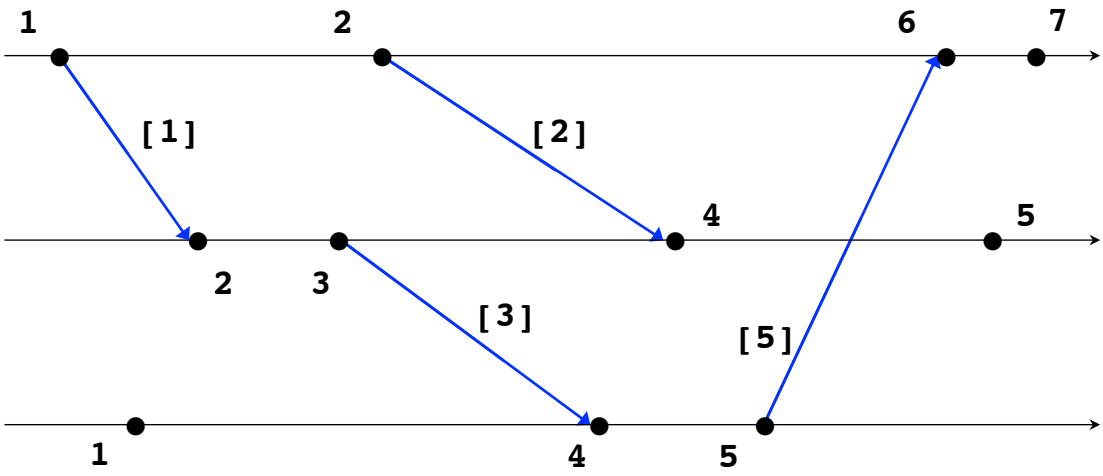
Choisir le type d'horloge en fonction des besoins de l'application

# Horloges Scalaire

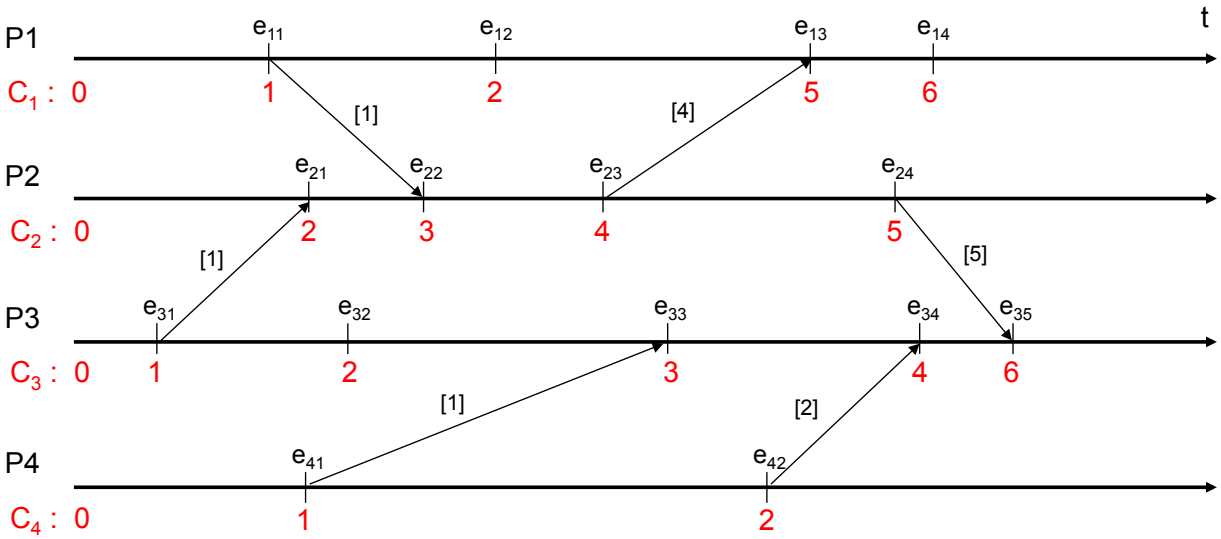
# Horloges scalaires (Lamport [78])

- **T =** entiers naturels N
- Tous les messages portent l'horloge de leur émetteur à l'instant d'émission **(estampillage)**
- **2 règles de mise à jour :**
  - **R1** : avant tout événement (interne, émission, réception), le processus i exécute
    - $C_i = C_i + d$ ($d > 0$, généralement $d = 1$)
  - **R2** : lorsque le site i reçoit un message portant une estampille $C_{msg}$, il exécute
    - $C_i = \max(C_i, C_{msg})$
    - Appliquer R1
    - Délivrer le message ➤ événement « réception »

# Exemple



# Horloges scalaires : exemple




# Horloges scalaires : propriétés

• C respecte la causalité : $(a \rightarrow b) \Rightarrow C(a) < C(b)$
  - vrai pour deux événements locaux
  - vrai pour une émission et une réception
  - vrai dans tous les cas par induction le long du chemin causal

• C capture la causalité mais ne la caractérise pas : $C(a) < C(b) \not\Rightarrow a \rightarrow b$
  . exemple : $C(e_{12}) = 2 < C(e_{23}) = 4$ et pourtant $e_{12} \parallel e_{23}$

• **C(e) = n** indique que (**n - 1**) événements se sont déroulés séquentiellement avant **e**.

# Ordre total des événements

• Deux événements peuvent avoir la même date logique (ils sont alors concurrents).

• Horloges scalaires peuvent être utilisées pour définir un **ordre total** **<<** sur les événements

• Pour ordonner totalement les événements, on complète la date logique d'un événement par le numéro du processus où il s'est produit : D(e) = ($C_i$, i), et on utilise la relation d'ordre :

$$e_i << e_j \Leftrightarrow C(e_i) < C(e_j) \text{ ou } [ C(e_i) = C(e_j) \text{ et } i < j ]$$

Exemples utilisation :

- Exclusion mutuelle répartie ( la datation logique permet d'ordonner totalement les requêtes pour garantir la vivacité de l'algorithme).
- Diffusion fiable et totalement ordonnée

# Horloges Vectorielles

# Horloges vectorielles : définition

**Fidge, Mattern [88]**

- Supposent un nombre fixe **N** de processus
- $\mathbf{T} = \mathrm{IN}^{N}$
- Chaque processus **i** gère un vecteur $\mathbf{VC_i}$ de taille **N**
  - $\mathbf{VC_i[i]}$ : nombre d'événements du processus **i**
  - $\mathbf{VC_i[j]}$ : connaissance qu'a **i** de l'avancement de l'horloge de **j**
- A tout instant, l'état réel d'avancement du système est donné par

  $\mathbf{W} = (\mathbf{VC_1[1]}, \ldots, \mathbf{VC_i[i]}, \ldots, \mathbf{VC_N[N]})$

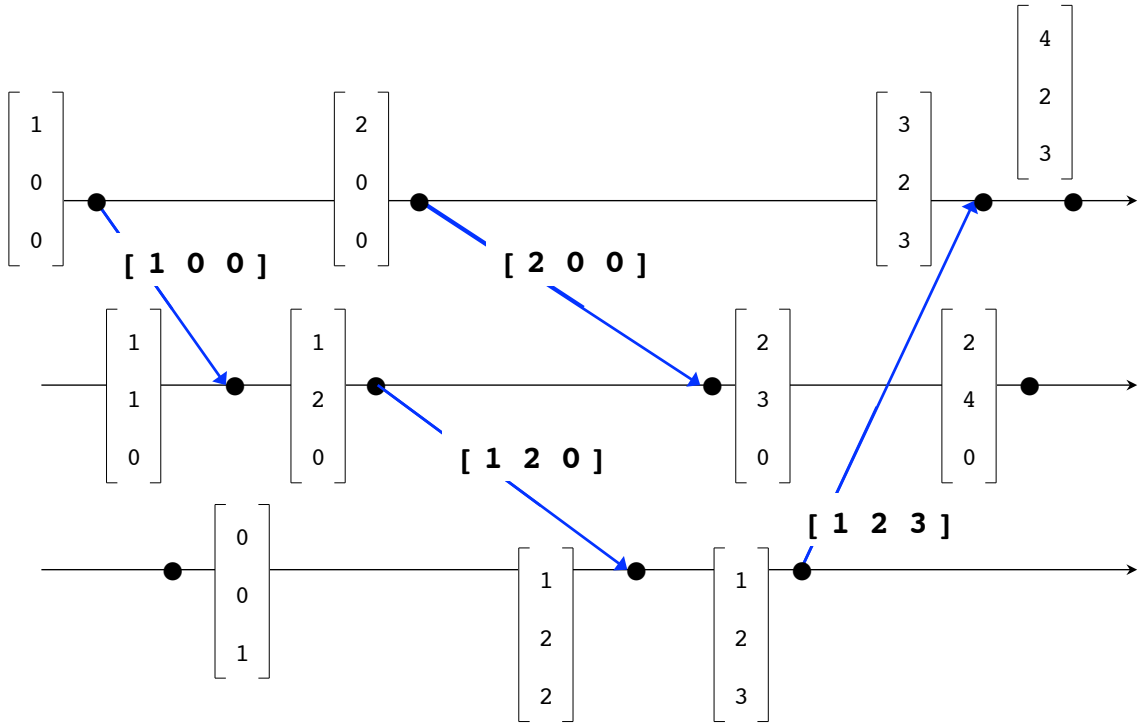
  mais aucun processus ne peut l'obtenir...

# Horloges vectorielles : algorithme de mise à jour

- **Pour le processus $P_i$ :**
  - A chaque événement (émission, réception ou événement interne) de $P_i$, incrémentation de $VC_i[i]$ :
    - $VC_i[i] = VC_i[i] + 1$
  - A l'émission d'un message m :
    - Le message porte la valeur courante de $VC_i$
  - A la réception d'un message m portant une horloge m.VC :
    - Mise à jour de l'horloge courante :

      $\forall x,\quad VC_i[x] = \max(VC_i[x], m.VC[x])$ et $VC_i[i] = VC_i[i] + 1$

# Exemple




# Horloges vectorielles et causalité

- **Les relations suivantes sont définies :**

  $VC_i \leq VC_j \Leftrightarrow \forall k\ VC_i[k] \leq VC_j[k]$

  $VC_i < VC_j \Leftrightarrow VC_i \leq VC_j$ et $VC_i \neq VC_j$

  $VC_i \parallel VC_j \Leftrightarrow !(VC_i \leq VC_j)$ et $!(VC_j \leq VC_i)$

- **Les horloges vectorielles définissent un ordre partiel sur les événements**
  - tous les vecteurs ne sont pas comparables
  - V ‖ V' dénote 2 vecteurs V et V' non comparables

- **Les horloges vectorielles caractérisent la causalité (consistance forte) :**
  - Soit **VC(e)** l'horloge vectorielle d'un événement *e*

    $e \rightarrow e' \quad \Leftrightarrow \quad VC(e) < VC(e')$

    $e \parallel e' \quad \Leftrightarrow \quad VC(e) \parallel VC(e')$

# Simplification du test de causalité

- Si **e** a lieu sur le site **i**, **e'** sur le site **j** :

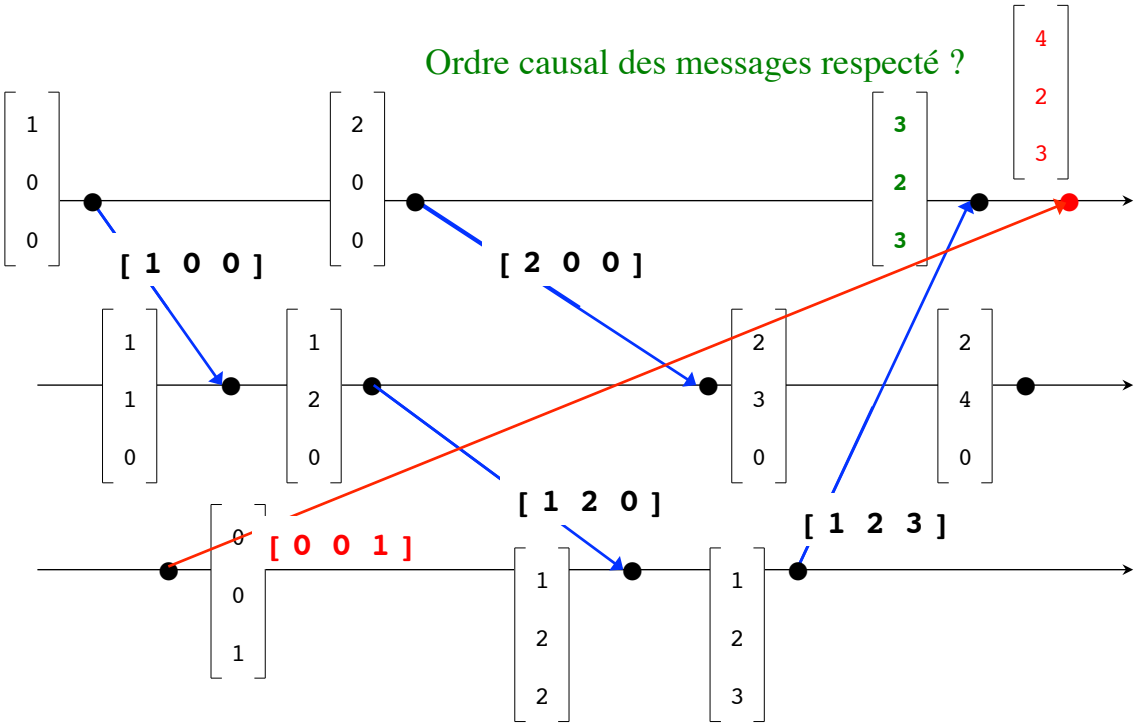
$e \rightarrow e' \quad \Leftrightarrow \quad VC(e)[j] < VC(e')[j]$

$e \parallel e' \quad \Leftrightarrow \quad VC(e)[i] > VC(e')[i]$ et
$VC(e')[j] > VC(e)[j]$

# Horloges vectorielles : LA solution ?

- **Caractérisent la causalité mais…**
  - Augmentent la quantité d'information gérée localement
  - Augmentent l'information circulant sur le réseau
    - Fonction du nombre de processus
  - Taille minimum : Charron-Bost a prouvé que la causalité des événements d'un système réparti avec N processus ne peut être caractérisée qu'avec des horloges vectorielles ayant au moins N entrées.
  - Détection à la volée du respect de l'ordre causale impossible

# Exemple




# Horloges vectorielles : LA solution ?

- **Caractérisent la causalité mais…**
  - Augmentent la quantité d'information gérée localement
  - Augmentent l'information circulant sur le réseau
    - Fonction du nombre de processus
  - Taille minimum : Charron-Bost a prouvé que la causalité des événements d'un système réparti avec N processus ne peut être caractérisée qu'avec des horloges vectorielles ayant au moins N entrées.
  - Détection online du respect de l'ordre causale impossible
- **Peut-on arriver au même résultat en manipulant moins d'information ?**
  - On ne peut pas diminuer l'information gérée localement
  - On peut optimiser l'information envoyée sur le réseau

# Implémentation d'horloges vectorielles

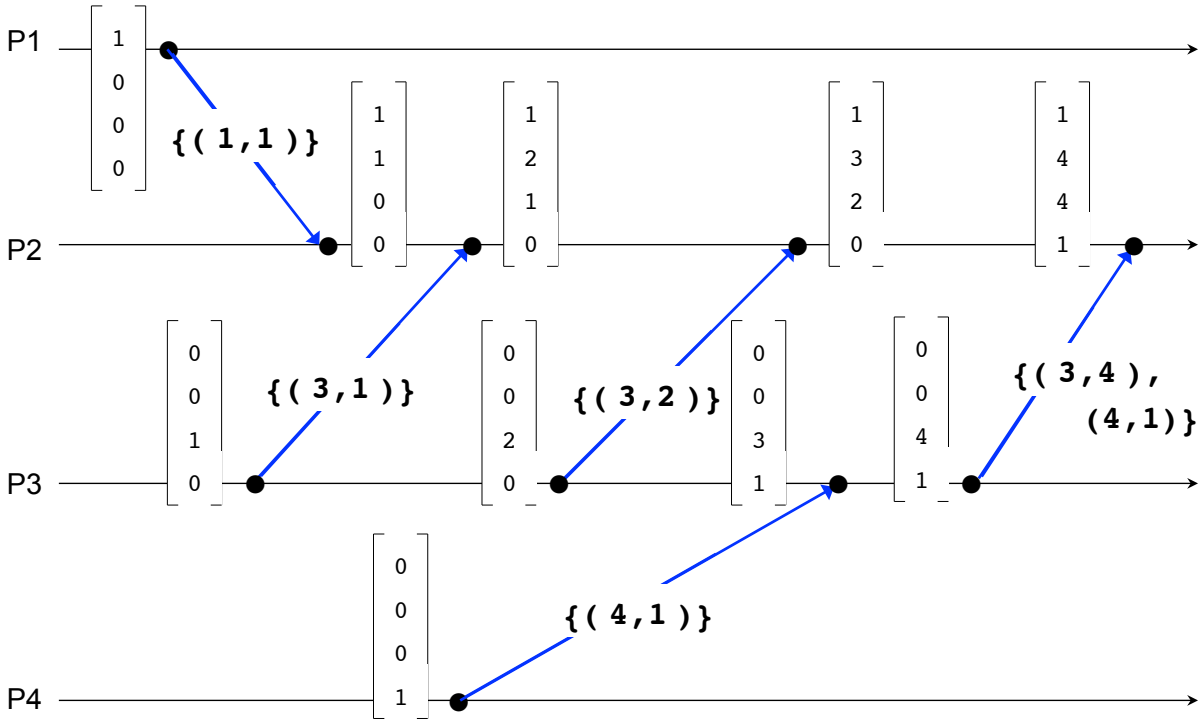
# Singhal / Kshemkalyani [92]

- **Principe : Gestion incrémentale des horloges vectorielles**
  - Dans un message envoyé par **j** à **i**, **j** n'inclut que les composantes de son horloge qui ont été modifiées depuis le dernier message qu'il a envoyé à **i**.
  - L'estampille du message est une liste de couples **(id, val)** tels que :
    - **$VC_j$[id]** a été modifié depuis le dernier message envoyé par j à i
    - La valeur courante de $VC_j$[id] est **val**.
  - Pour tout couple (id, val) contenu dans le message de j :

$$VC_i[id] = \max(VC_i[id], val)$$

- **Conditions d'application :**
  - Canaux FIFO

# Exemple




# Implémentation de la méthode S/K

- **Quelle information maintenir sur chaque site ?**
  - Celle qui permet à **j** de savoir quelles composantes de son vecteur d'horloge envoyer à **i** :
    - Horloge vectorielle de dernière émission vers i
- **Sous quelle forme ?**
  - Matrice ?
    - La colonne **i** est le vecteur d'horloge du dernier envoi à **i**

    PB : stockage local en $N^2$
  - Vecteurs ?
    - On peut stocker les dates d'émission sous forme scalaire : **LS**
    - On stocke aussi les dates (scalaires) de modification de chacune des composantes du vecteur d'horloge : **LU**

➔ 2 vecteurs sur chaque site en plus de l'horloge vectorielle

# Implémentation (suite)

- **Pourquoi des dates scalaires ?**
  - Tous les événements à considérer ont lieu sur le même site **j**

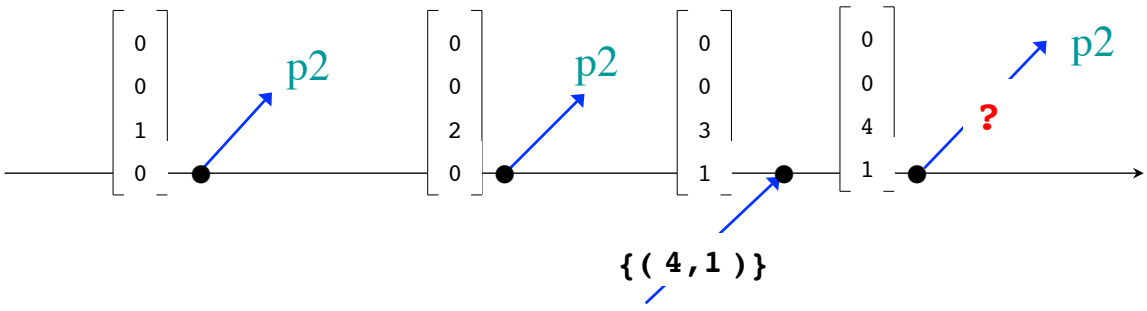
  ⇒ ils sont identifiés de manière unique par $\mathbf{VC_j[j]}$
- **Gestion des dates d'émission : vecteur LS**
  - LS : Last Sent
  - $LS_j[i]$ = valeur de $VC_j[j]$ lors du dernier envoi de j à i
- **Gestion des dates de modification : vecteur LU**
  - LU : Last Update
  - $LU_j[k]$ = valeur de $VC_j[j]$ lors de la dernière modification de $VC_j[k]$

# Identification des valeurs émises

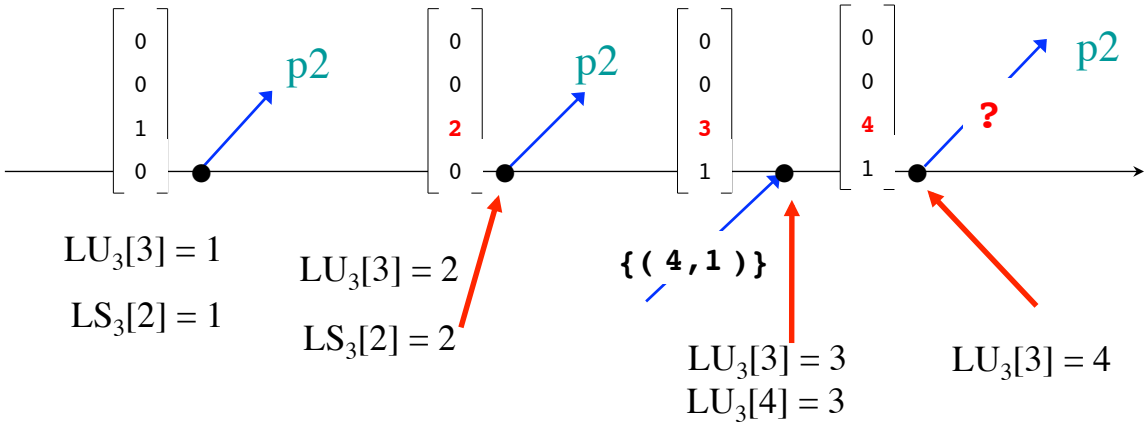
- **j** envoie à **i** toutes les composantes **k** de $\mathbf{VC_j}$ telles que :

$$\mathbf{LU_j[k] > LS_j[i]}$$

  sous la forme (k, $VC_j[k]$)
- **Exemple : calcul des valeurs envoyées par p3**



# Exemple




Valeur des vecteurs pour l'événement $\mathbf{e_3^4}$ :

$LS_3 = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}$ $LU_3 = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 3 \end{bmatrix}$ $VC_3 = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix}$

$LU_3[3] > LS_3[2]$

$LU_3[4] > LS_3[2]$

➥ envoi de $\{ (3, VC_3[3]), (4, VC_3[4]) \}$

# Efficacité de la méthode

- **Efficace si les interactions entre processus sont localisées :**
  - Chaque processus ne communique qu'avec un petit nombre d'autres processus

    ➥ le nombre d'entrées modifiées est faible
- **Mais perte d'information au niveau de la causalité :**
  - Un processus qui reçoit deux messages en provenance d'émetteurs différents ne peut pas établir la relation de causalité entre leurs émissions

    ce n'est pas forcément un problème…

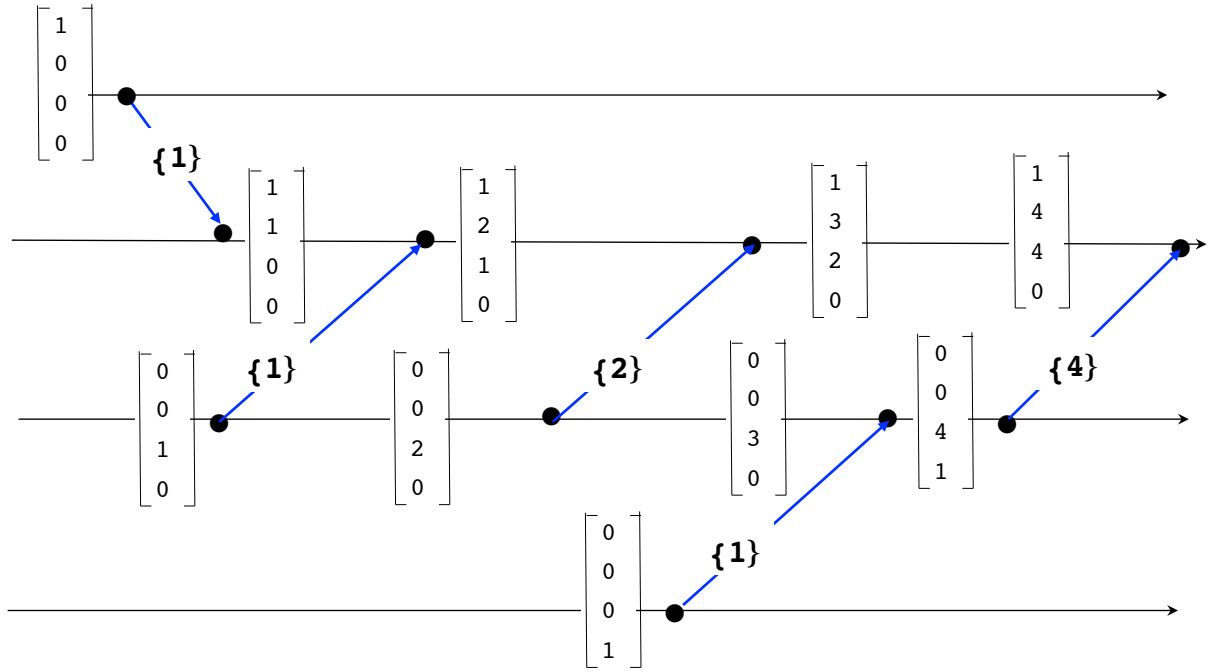
# Fowler/Zwaenepoel vecteurs de dépendence

# Fowler/Zwaenepoel

- **Gestion de vecteurs de dépendance**
  - Un vecteur de dépendance **n'est pas** une horloge vectorielle
  - Dépendance directe
  - Mais il permet de les reconstruire
- **Principe :**
  - Chaque processus gère un vecteur D de taille N
  - Lors d'une émission de message, un processus y inclut la dernière valeur de sa composante locale $D_i[i]$
  - Lors de la réception par i d' un message en provenance de j contenant la valeur d :

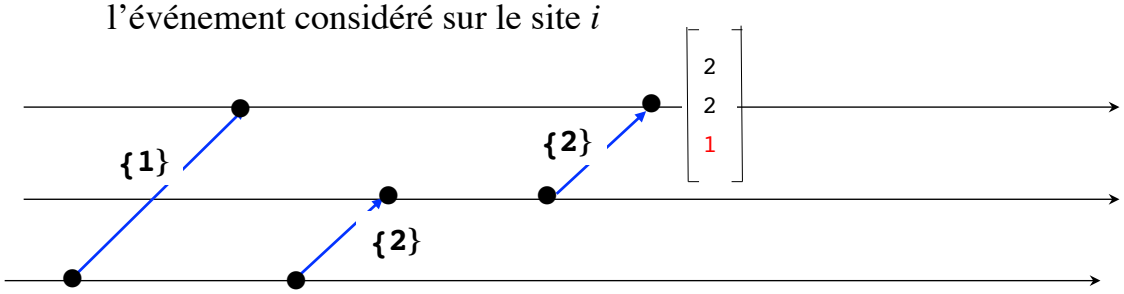
$$D_i[j] = \max (D_i[j], d)$$

# Exemple




# Analyse Information

- **Interprétation de $D_i[k]$**
  - Indice du dernier événement sur le site *k* dont dépend directement l'événement considéré sur le site *i*




- **Perte de l'information liée à la transitivité**
  - $D_i[k]$ stocke l'information en provenance directe de *k*
  - *i* ne peut pas apprendre par l'intermédiaire de *j* l'état d'avancement de *k*

# Reconstruction des Horloges Vectorielles

- **Evénement *e* sur le site *i***
- **Algorithme itératif :**

  $VC_i(e) = D_i(e)$

  Repeat

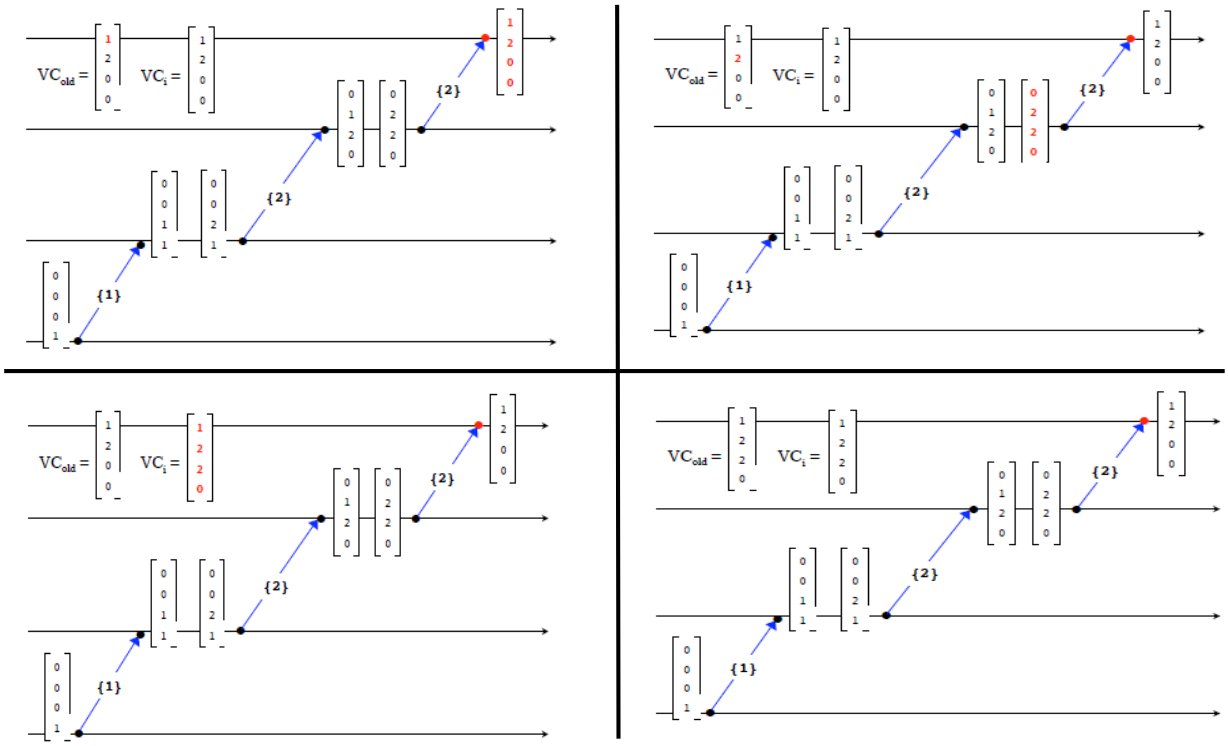
  - $VC_{old}(e) = VC_i(e)$
  - Pour chaque composante non nulle $VC_i(e)[k]$ :

    $VC_i(e) = \sup (VC_i(e) , D_k(e_k^{VCold[k]}) )$

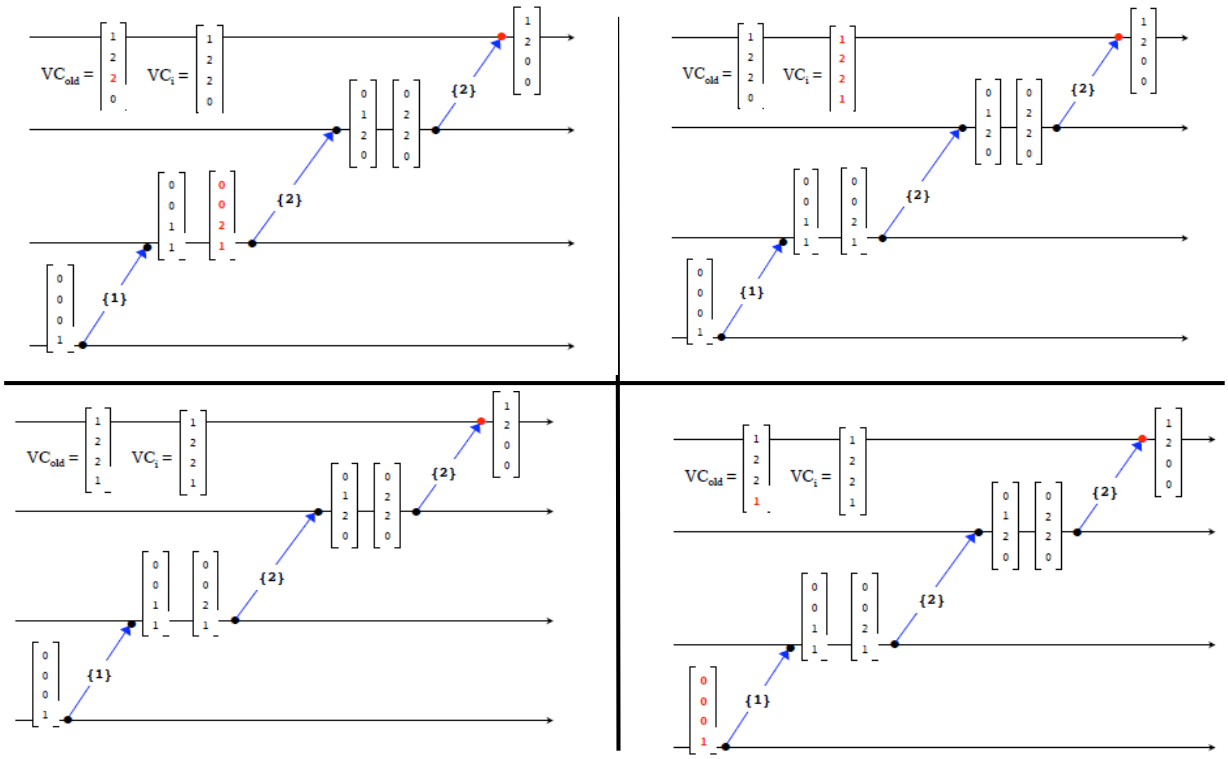
  Until $VC_i(e) = VC_{old}(e)$

- **Algorithme exécutable uniquement «off-line»**

# Exemple



# Exemple




# Considérations sur l'approche

- **Minimisation de la somme d'information gérée :**
  - Un vecteur par processus
  - Un scalaire par message
- **Inapplicable si l'application doit systématiquement reconstruire les horloges vectorielles**

# Horloges Matricielles

# Horloges Matricielles

- **M.J. Fisher, A. Michael [82]**
- **T = IN$^{N2}$**
- **Principe :**
  - Chaque processus gère une matrice $MT_i$ de taille $N^2$
  - $Mt_i[i,i]$ : Nombre d'événements du processus `i`
  - $Mt_i[i,*]$ : horloge vectorielle du processus `i`
  - $Mt_i[k,l]$ : Connaissance qu'a `i` de ce que le processus `k` connaît de l'horloge du processus `l` (c'est-à-dire $Mt_l[l,l]$)

# Algorithme de mise à jour

- **Pour le processus $P_i$**
  - Chaque événement :

$$Mt_i[i,i] = Mt_i[i,i]+1$$

  - Chaque émission d'un message $m$ :

le message porte $Mt_i$

  - Chaque réception d'un message m émis par j et portant l'horloge $m.MT$
    - Pour tout $x \leq N$, $Mt_i[i,x] = \max(Mt_i[i,x], m.MT[j,x])$, c'est-à-dire que l'horloge vectorielle de $i$ est mise à jour par rapport à celle de $j$
    - Pour tout $x \leq N$, pour tout $y \leq N$, $Mt_i[x,y] = \max(Mt_i[x,y], m.MT[x,y])$

# Considérations sur l'approche

- A toutes les propriétés des vecteurs d'horloges
- Très couteux en espace, sur les messages et en calcul
- Processus $i$ sait que tous les autres processus $p_j$ savent que le temps local de chaque $p_k$ a progressé jusqu'au moins $t$, c'est-à-dire :

$$\min(Mt_i[k,l]) \geq t$$

- Permet d'implémenter un mécanisme tampon (file de messages) pour respecter l'ordre causal des messages à la volée

# Dépendence Causale entre messages

# Dépendance causale entre messages

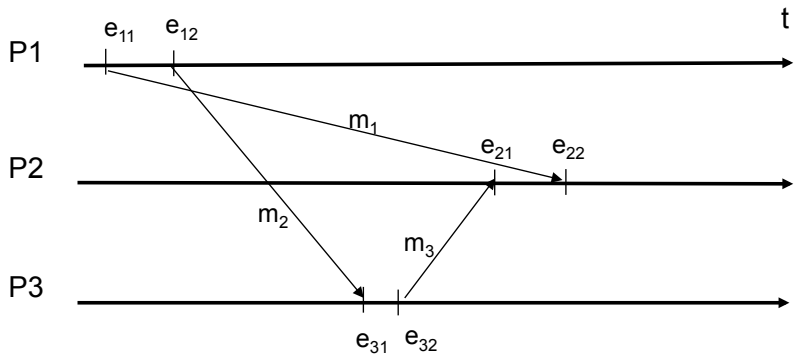
- On considère que ***send(m)*** **et** ***receive(m)*** correspondent à l'émission et la réception d'un message *m* et que les messages doivent respecter la dépendance causale.

- **Définition :**

  Les communications sont ordonnées causalement ("Causal Ordered" - CO), si, pour tout message *m* et *m'*, on a :

  $send(m) \rightarrow send(m') \Rightarrow receive(m) \rightarrow receive(m')$

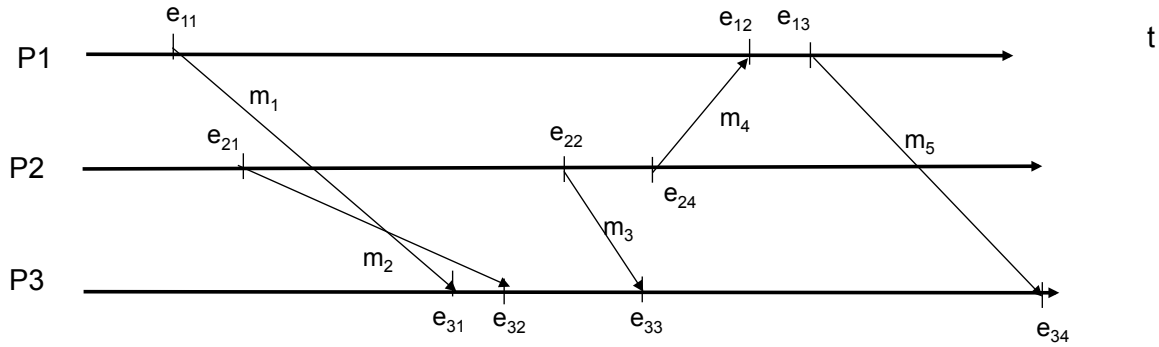
# Dépendance causale entre messages : Exemple (1)




La réception sur P2 ne respecte pas la dépendance causale des messages :

send($m_1$) → send($m_3$) mais receive($m_1$) ↛ receive($m_3$)

# Dépendance causale entre messages : Exemple (2)



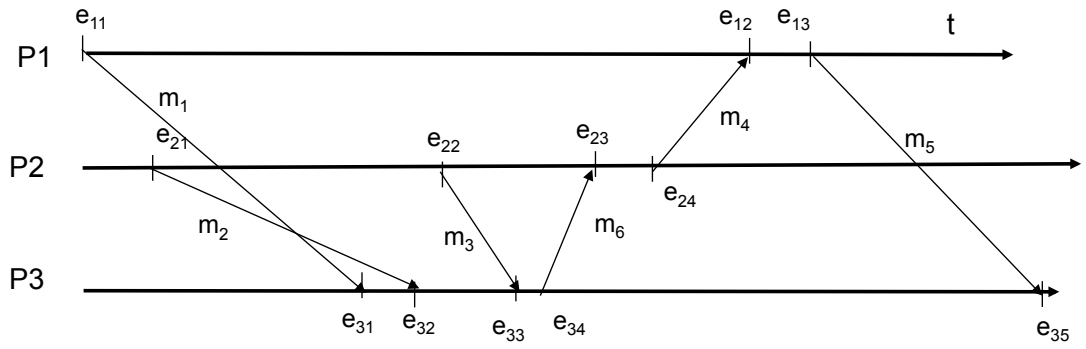

Toutes les communications ne sont pas ordonnées causalement :

send($m_3$) → send($m_4$) mais receive($m_3$) ↛ receive($m_4$)

send($m_2$) → send($m_4$) mais receive($m_2$) ↛ receive($m_4$)

Observation : les émissions de $m_1$ et $m_2$ ne sont pas en relation de précédence

# Dépendance causale entre messages : Exemple (3)




Les communications sont ordonnées causalement

Observation : les émissions de $m_1$ et $m_2$ ne sont pas en relation de précédence

# Conclusions

- **La causalité est la relation qui permet d'analyser une application répartie**
  - Pas d'exécutions identiques
  - Nécessité de tracer le lien entre les événements
- **Le type d'horloge dépend de l'application**
  - Consistance forte ? Horloges vectorielles
  - Consistance simple ? Horloges scalaires
  - Ordre causal des messages ? Horloges matricielles
- **Approche Fowler / Zwaenepoel**
  - Dépendance directe

# Bibliographie


- Lamport, L. *Time, clocks, and the ordering of events in a distributed system*. Communications of the ACM, 21(7), 1978.
- Mattern, F. *Virtual Time and Global States of Distributed Systems*. In Proceedings. Workshop on Parallel and Distributed Algorithms,1989.
- C. Fidge. *Logical time in distributed computing systems*. IEEE Computer, pages 28-33, July 1991.
- M.Singhal, A. Khsemkalyani. *An efficient implementation of vector clocks*, Information Processing Letters, 992, vol. 43, no1, pp. 47-52.
- M. Raynal and M. Singhal. *Logical time: a way to Capture Causality in Distributed Systems*. Technical Report n. 2472, INRIA - Rennes,1995
- Bernadette Charron-Bost: *Concerning the Size of Logical Clocks in Distributed Systems*. Information. Processing Lett. 39(1): 11-16 (1991)
- Fowler J. and Zwaenepoel W. *Causal Distributed Breakpoints*. Proc of the Int. Conf. On Districuted Computing Systems, pp. 134-141, 1990.