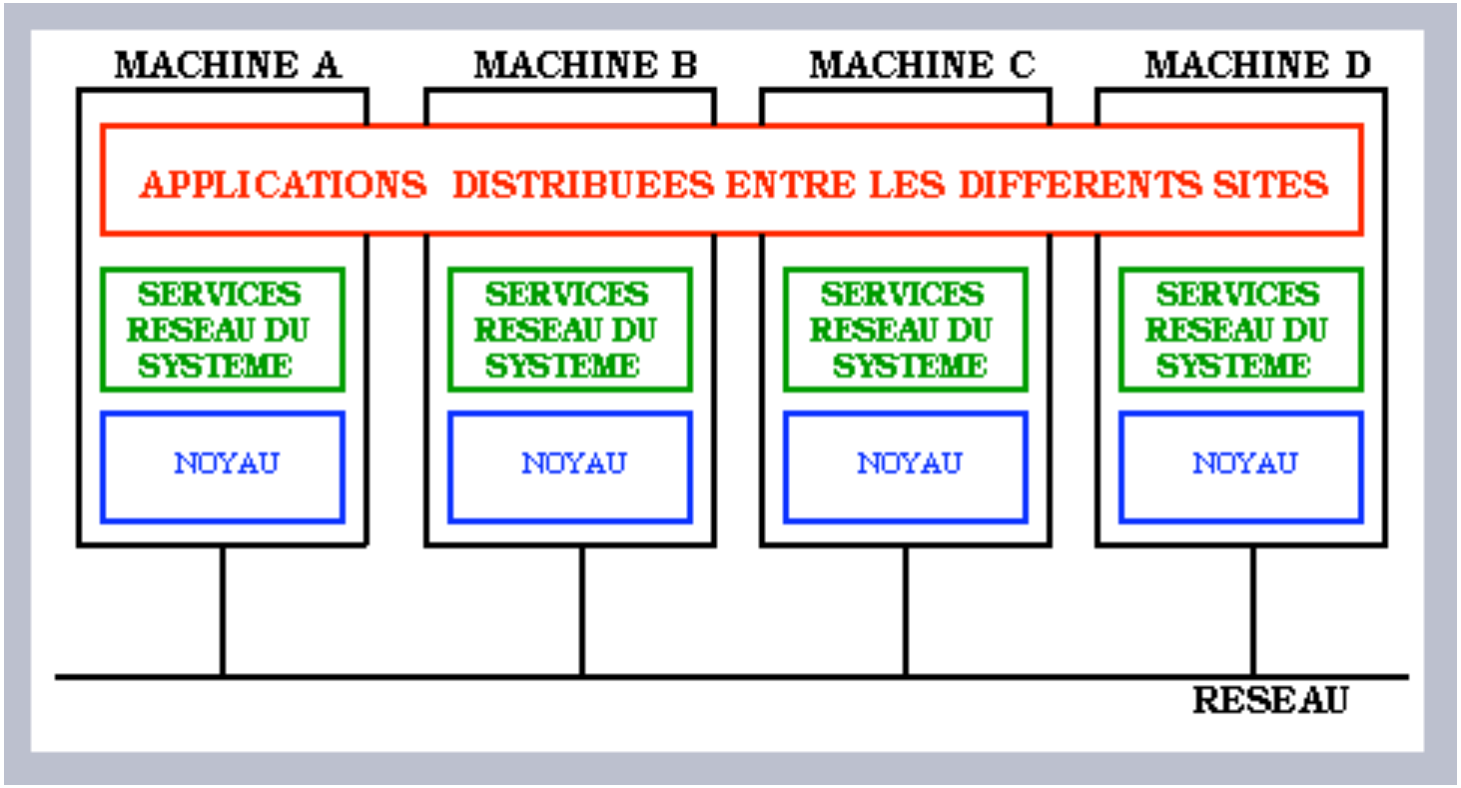
Horloges logiques et datation des évènements.

Système réparti

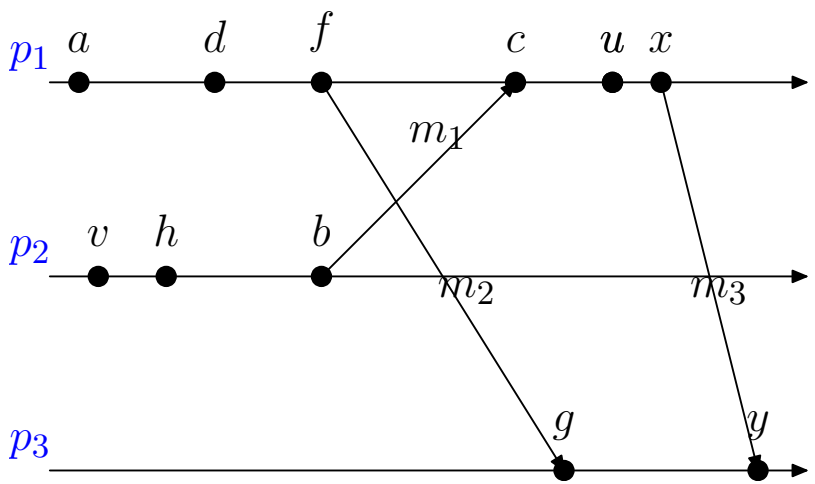


Être capable de raisonner sur un système réparti

- définir des prédicats (état)
- coordonner des instructions (ordre)

Comment dater les évènements

- Sur un même site : ordre des événements = ordre exécution des instructions
- sur plusieurs sites : ordre des événements ? ? ?



Difficile car....

- pas de mémoire commune, pas d'horloge commune
- Asynchronisme des communications et des traitements.

Relation d'ordre sur les évènements : précédence.

- Sur plusieurs sites : il faut définir un ordre des événements tel que
  - il soit global (c'est-à-dire total)
  - il doit être calculer entre deux évènements en fonction de l'information local.

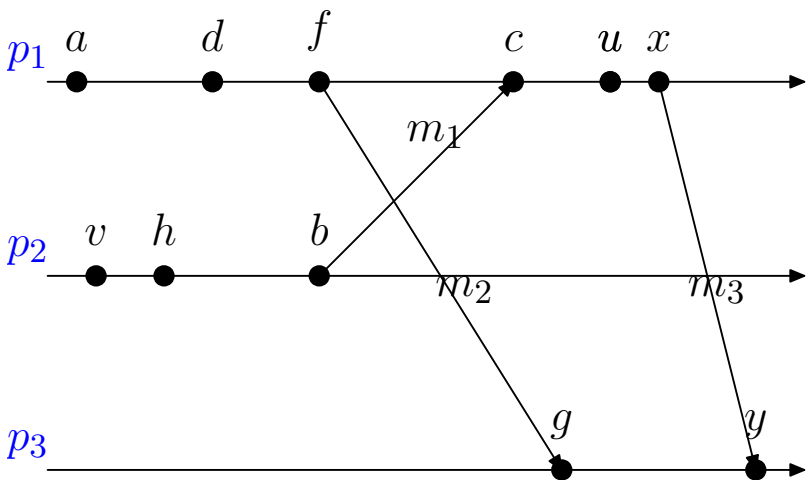
Solution : Définition de la causalité [Lamport78].

Ordre causal : l'ordre partiel sur les événement

Si a et b sont deux événements, a précède b ($a \rightsquigarrow b$)

si et seulement si l'une des trois conditions est vraie :

- a et b ont lieu sur le même site avec a avant b.
- $a = \mathit{envoyer}(M)$ et $b = \mathit{recevoir}(M)$ du même message M.
- Il existe un évènement c tel que $a \rightsquigarrow c$ et $c \rightsquigarrow b$.



- a, d, f, c précèdent u
- b précède c
- $v \rightsquigarrow u$ car $v \rightsquigarrow u$ et $c \rightsquigarrow u$

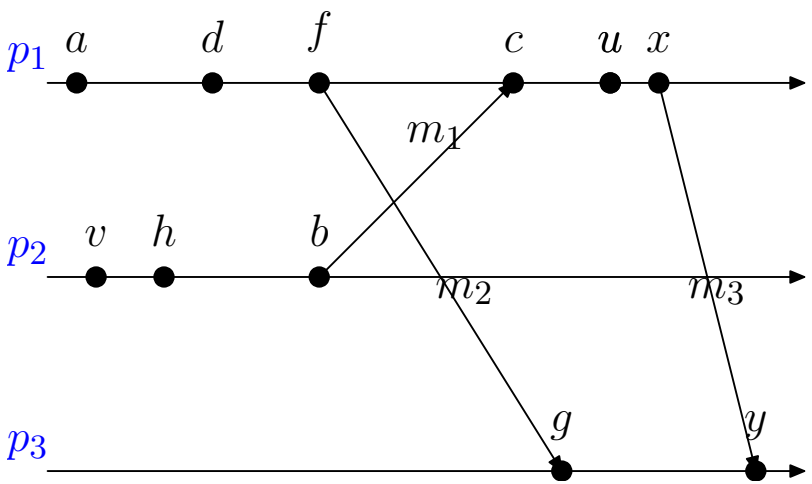
Ordre causal

A un événement **e** trois ensembles d'événements sont associés :

- *Passe*(**e**) : ens. des événements tel que $\mathbf{e}' \in \mathit{Passe}(\mathbf{e})$ si $\mathbf{e}' \rightsquigarrow \mathbf{e}$
- *Futur*(**e**) : ens. des événements tel que $\mathbf{e}' \in \mathit{Futur}(\mathbf{e})$ si $\mathbf{e} \rightsquigarrow \mathbf{e}'$
- *Concurrent*(**e**) : ens. des événements tel que $\mathbf{e}' \in \mathit{Concurrent}(\mathbf{e})$ si

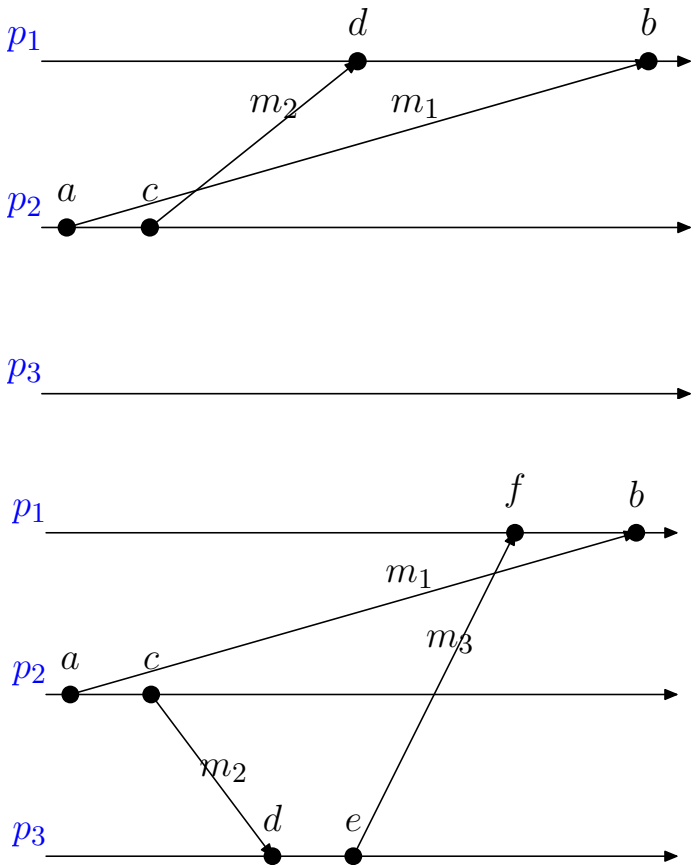
$$\mathbf{e}' \notin \mathit{Futur}(\mathbf{e}) \cup \mathit{Passe}(\mathbf{e}).$$

Notation $\mathbf{e} \,||\, \mathbf{e}'$: deux événements **e** et $\mathbf{e}'$ sont concurrents.



- $\mathit{Passe}(u) = \{a, b, c, f, v, h\}$,
- $\mathit{Futur}(u) = \{x, y\}$,
- $\mathit{Concurrent}(u) = \{g\}$
- $u \,||\, g$

Remarque sur certains points incohérents



Canal non FIFO :

- $c \rightsquigarrow d$
- $d \rightsquigarrow b$
- intérêt de recevoir m_1 ?

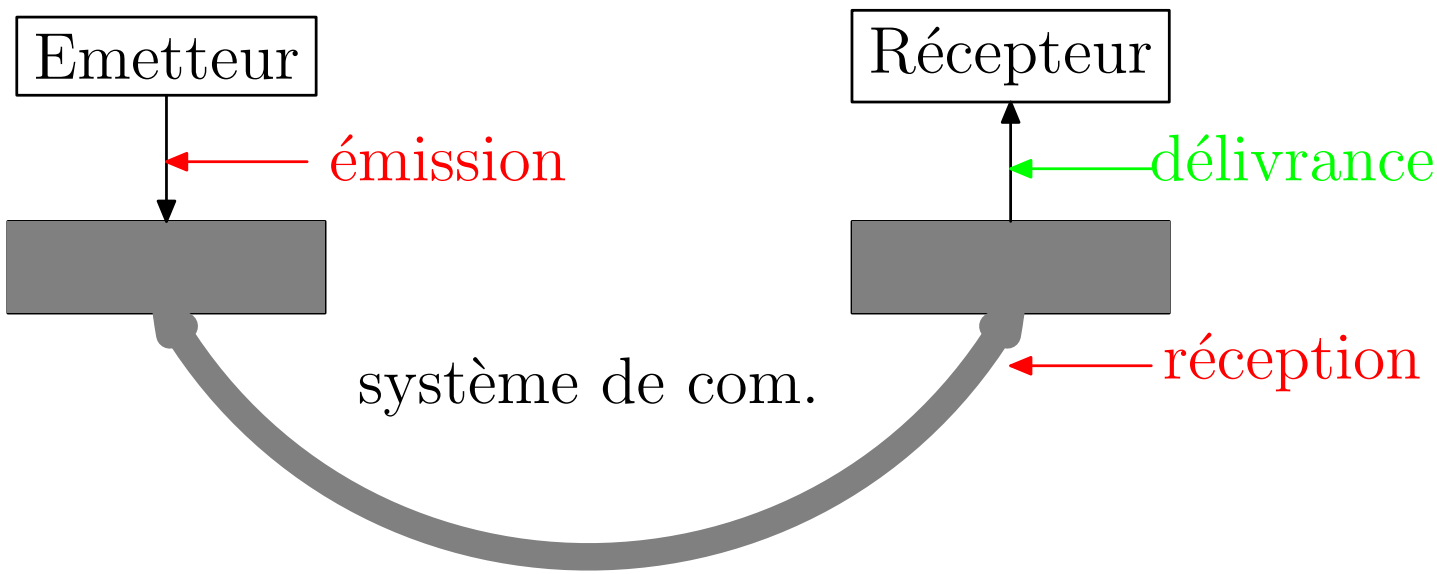
Autre scénario :

- $a \rightsquigarrow f$
- $f \rightsquigarrow b$
- intérêt de recevoir m_1 ?

Délivrer à l'application cliente de manière correcte

Délivrance d'un message

La communication entre les noeuds est en général assurée par une couche de communication spécifique utilisée par des entités de la couche supérieure pour émettre/recevoir des messages.

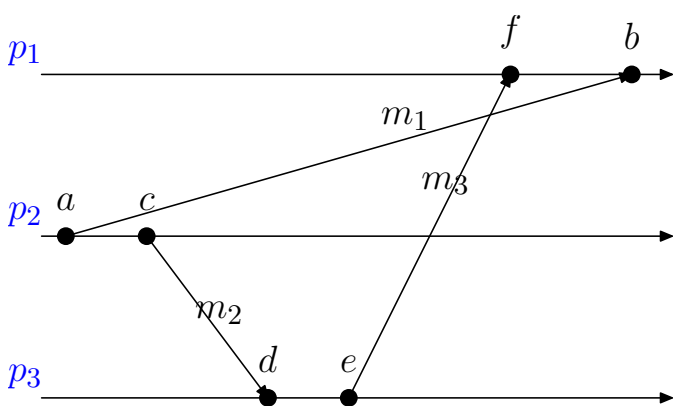


La délivrance d'un message : l'opération consistant à le rendre accessible aux applications clientes.
Par exemple :le protocole TCP dans la couche transport.

Notation

Nous noterons par

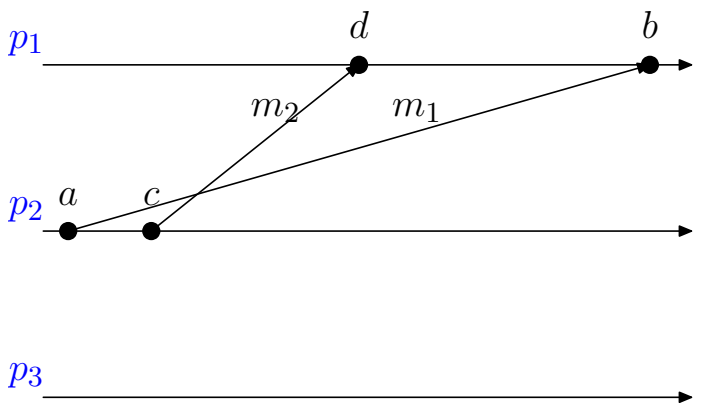
- $envoyer_j(M)$: l'envoi du message M par le site j.
- $recevoir_j(M)$: la réception du message M par le site j.
- $del_j(M)$: la délivrance du message M par le site j.
- $Passe_j(e)$: le passé de e en se restreignant aux évènements du site j.



Délivrance FIFO

La délivrance FIFO assure que si deux messages sont envoyés successivement depuis un même site S_i vers un même destinataire S_j, alors le premier envoyé sera délivré sur le site S_j avant le second. Formellement, on a

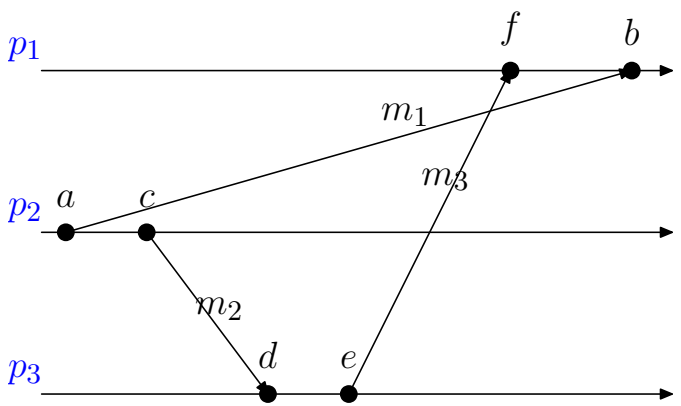
$$envoyer_i(m_1, j) \rightsquigarrow envoyer_i(m_2, j) \text{ implique } del_j(m_1) \rightsquigarrow del_j(m_2)$$



Délivrance causale

La délivrance causale assure que si l'envoi du message m_1 par le site S_i à destination du site S_k précède (causalement) l'envoi du message m_2 par le site S_j à destination du site S_k, le message m_1 sera délivré avant le message m_2 sur le site S_k .
Formellement, on a

$$envoyer_i(m_1, k) \rightsquigarrow envoyer_j(m_2, k) \text{ implique } del_k(m_1) \rightsquigarrow del_k(m_2)$$



Plus précisément

Horloges logiques et datation des évènements

- Horloges et estampilles scalaires
- Horloges et estampilles vectorielles

Horloges scalaires

Chaque site i gère un compteur (une horloge HL_i) dont la valeur est un entier.

Utilisation :

- Chaque évènement local est daté (HL_i).
- Chaque message m envoyé est estampillé (daté) par la date de son évènement (noté par EL_m).

Réactualisation :

- Si un événement est local, alors $HL_i \leftarrow HL_i + 1$
- Si l'événement est l' envoi d'un message m alors

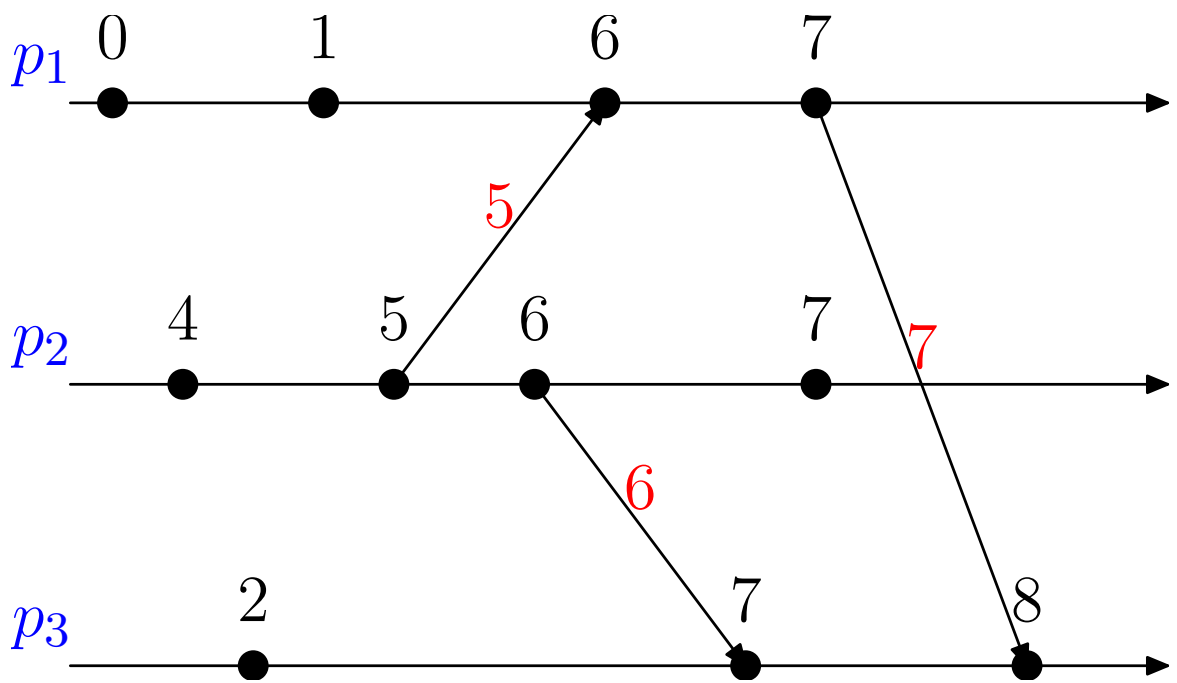
$$\begin{cases} HL_i \leftarrow HL_i + 1 \\ EL_m \leftarrow HL_i \end{cases}$$

- Si l'événement est la réception du message m alors

$$HL_i \leftarrow \max(HL_i, EL_m) + 1$$

Illustration

L'ordre des événements n'est pas un ordre strict : plusieurs événements peuvent porter la même valeur.



Ordre strict si modification suivante

- La date logique (estampille) $HL(e)$ d'évènement e sur le site i est

  le couple (HL_i, i).

- L'ordre strict sur les estampilles est le suivant :

$$(HL_i, i) \prec (HL_j, j) \text{ si et seulement si } \begin{cases} HL_i < HL_j \\ ou \quad HL_i = HL_j \wedge i < j \end{cases}$$

Propriété : respect de l'ordre causal ?

Propriété

Si $\mathbf{e} \rightsquigarrow \mathbf{e}'$, alors $HL(\mathbf{e}) \prec HL(\mathbf{e}')$

Démonstration.

Raisonnement par récurrence :

sur la longueur n de la suite d'évènements reliant $\mathbf{e}$ à $\mathbf{e}'$

- si $n = 1$: alors $\mathbf{e}$ précède immédiatement $\mathbf{e}'$

  et on a $HL(\mathbf{e}) \prec HL(\mathbf{e}')$
- si $n > 1$: soit $\mathbf{e} = \mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_n = \mathbf{e}'$ une suite d'évènements tel que $\forall i \in [0, \ldots, n-1], \mathbf{e}_i \rightarrow \mathbf{e}_{i+1}$
  - Par récurrence, on a $\mathbf{e} \rightsquigarrow \mathbf{e}_{n-1}$ et $HL(\mathbf{e}_i) \prec HL(\mathbf{e}_{n-1})$
  - Comme $\mathbf{e}_{n-1} \rightarrow \mathbf{e}_n$, deux cas se présentent
    - Si $\mathbf{e}_{n-1}$, et $\mathbf{e}_n$ appartiennent au même site,

      alors on a $HL(\mathbf{e}_{n-1}) \prec HL(\mathbf{e}_n)$
    - Si il existe un message m tel que $envoyer(m) = \mathbf{e}_{n-1}$ et $recevoir(m) = \mathbf{e}_n$:

      alors on a $HL(\mathbf{e}_{n-1}) \prec HL(\mathbf{e}_n)$

Exemple d'application : exclusion mutuelle

Problème

Un seul site à la fois peut accèder à la ressource critique

Algorithme distribué

Gestion d'une file d'attente distribuée.

1. Chaque site i voulant accèder à la ressource date sa demande
2. Il envoie à tout le monde
   un message de demande d'accès + sa date
3. Tous les sites ne voulant pas y accèder ou voulant accéder mais ayant une date de demande postérieure à la sienne
   lui donnnent leur accord.
4. Pour les autres, ils attendent qu'ils ont accèdé et libéré la ressource pour lui donner leur accord.
5. Dès qu'il a reçu l'accord de tout le monde,
   il rentre dans la section critique.

Hypothèse : Tous les sites connaissent le nombre total de sites.

Plus formellement (1/2)

Commande

$$\texttt{règle gardée} \rightarrow \texttt{instructions}$$

- Une `règle gardée` est un prédicat sur les états des voisins.
- Les `instructions` sont exécutées si la `règle gardée` est vraie.

Plus formellement : sur le site i (2/2)

Procédure d'acquisition

1. $h_i \leftarrow h_i + 1$;
2. $D_i \leftarrow vrai$;$h\_d_i \leftarrow h_i$;
3. $rep\_attendues_i \leftarrow n - 1$;
4. pour tout ($j \in V \backslash \{i\}$) faire
   4.1 envoyer <Accès,h_d_i> à j

Procédure Libération

1. $D_i \leftarrow faux$; $h_i \leftarrow h_i + 1$;
2. pour tout $j \in V \backslash \{i\}$ faire
   Si $differe_i[j] = vrai$
   2.1 envoyer <ok> à j ;
   2.2 $differe_i[j] \leftarrow faux$;

Réception de <Accès,h_ℓ> venant de $\ell \rightarrow$

1. $h_i = max(h_i, h) + 1$;
2. si (non D_i) ou $(h\_d_i, i) > (h, \ell)$ alors envoyer <ok> à ℓ
3. sinon $differe_i[j] \leftarrow vrai$;

Réception de <ok> venant de $\ell \rightarrow$

1. $h_i = max(h_i, h) + 1$; $rep\_attendues_i \leftarrow rep\_attendues_i - 1$;
2. si ($rep\_attendues_i = 0$) alors ACCEDER A LA RESSOURCE

Exemple : 3 sites.

Site 1		Site 2		Site 3	
envoi/reception	$(D_1, h_1, \#ok)$,Diff	envoi/reception	$(D_2, h_2, \#ok)$	envoi/reception	$(D_3, h_3, \#ok)$
	$(D, 14, \#0)$				
env(Acces,14)$\rightarrow$					
		R(Accès,14,1)		R(Accès,14,1)	
		$\leftarrow (ok, 1)$		$\leftarrow (ok, 1)$	
R(ok,1),R(ok,1)					
	(D, 14, 2 ok)				
Ressource			(D,21,0ok)		
Ressource		env(Accès,21)			
R.+ R(Accès,21,2)					(D,20,0ok)
	(D,14,2ok),{2}			env(Accès,20)	
R.+ R(Accès,20,3)		R(Accès,20,3)			
	(D,14,2ok),{2, 3}	$(ok, 3) \rightarrow$			
Ressource				R(Accès,21,2)	(D,20,0ok),{2}
Ressource				$R(ok, 3)$	(D,20,1ok),{2}
Liberation	$(F, *, 0)$				
$(ok, 2), (ok, 3) \rightarrow$					
		$R(ok, 2)$		$R(ok, 3)$	
			(D,1,1ok)		(D,20,2ok),{2}
				Ressource	
				Liberation	(D,20,2ok),{2}
				$\leftarrow (ok, 2)$	(F,*,0ok),
		$R(ok, 2)$			
			(D,20, 2ok)		
			Ressource		

Plus précisément

Horloges logiques et datation des évènements

- Horloges et estampilles scalaires
- Horloges et estampilles vectorielles

Horloges vectorielles

Chaque site i gère un vecteur d'entiers (une horloge HV_i) de n éléments (n est le nombre de sites).

Utilisation :

- Chaque évènement local est daté (HV_i).
- Chaque message m envoyé est estampillé (noté par EV_m).

Réactualisation :

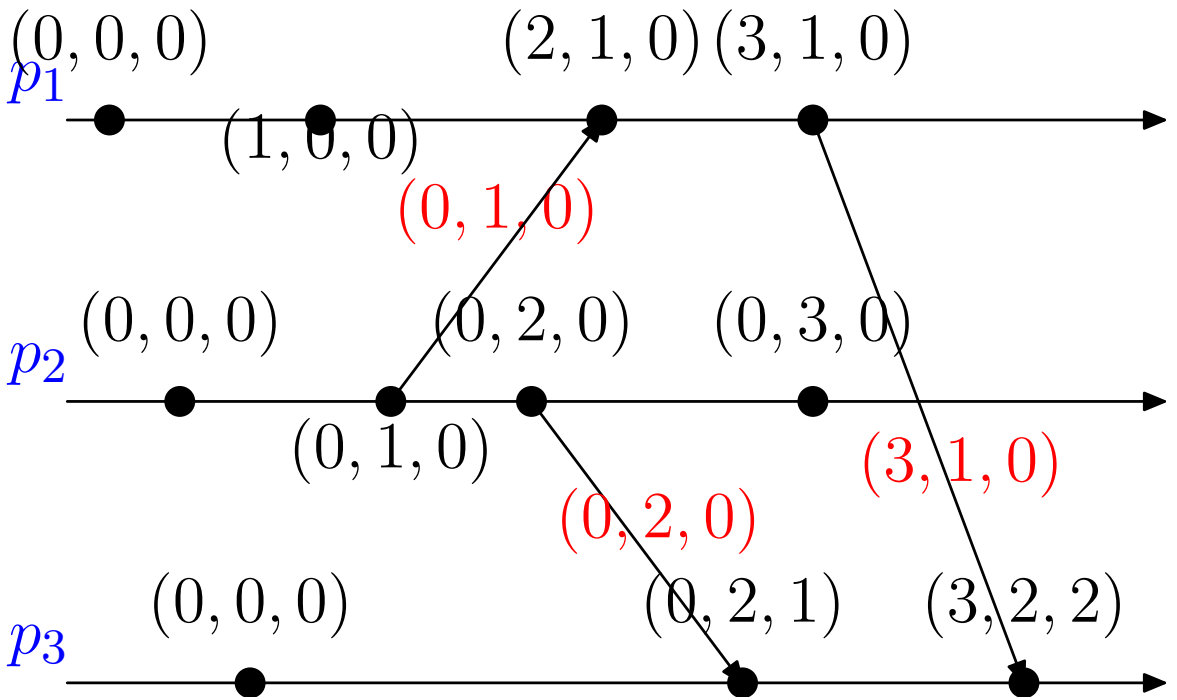
- Si un événement est local, alors $HV_i[i] \leftarrow HV_i[i] + 1$
- Si l'événement est l'envoi d'un message m alors

$$\begin{cases} HV_i[i] \leftarrow HV_i[i] + 1 \\ EV_m \leftarrow HV_i \end{cases}$$

- Si l'événement est la réception du message m alors

$$\begin{cases} HV_i[i] \leftarrow HV_i[i] + 1 \\ HV_i[j] \leftarrow \max(HV_i[j], EV_m[j]) \quad \text{pour tout } j \neq i \end{cases}$$

illustration



Propriété

Propriété

Soit $\mathbf{e}$ un évènement ayant $EV_{\mathbf{e}}$ comme date. La valeur de la $i$ème composante de $EV_{\mathbf{e}}$ correspond au nombre d'évènements du site S_i appartenant au passé de $\mathbf{e}$.

$$\forall i, EV_{\mathbf{e}}[i] = |\{\mathbf{e}' : \mathbf{e}' \in S_i \wedge \mathbf{e}' \rightsquigarrow \mathbf{e}\}|$$

Démonstration.

Soit $\mathbf{e}'$ un évènement de S_i tel que $EV_{\mathbf{e}'}[i] = EV_{\mathbf{e}}[i]$.

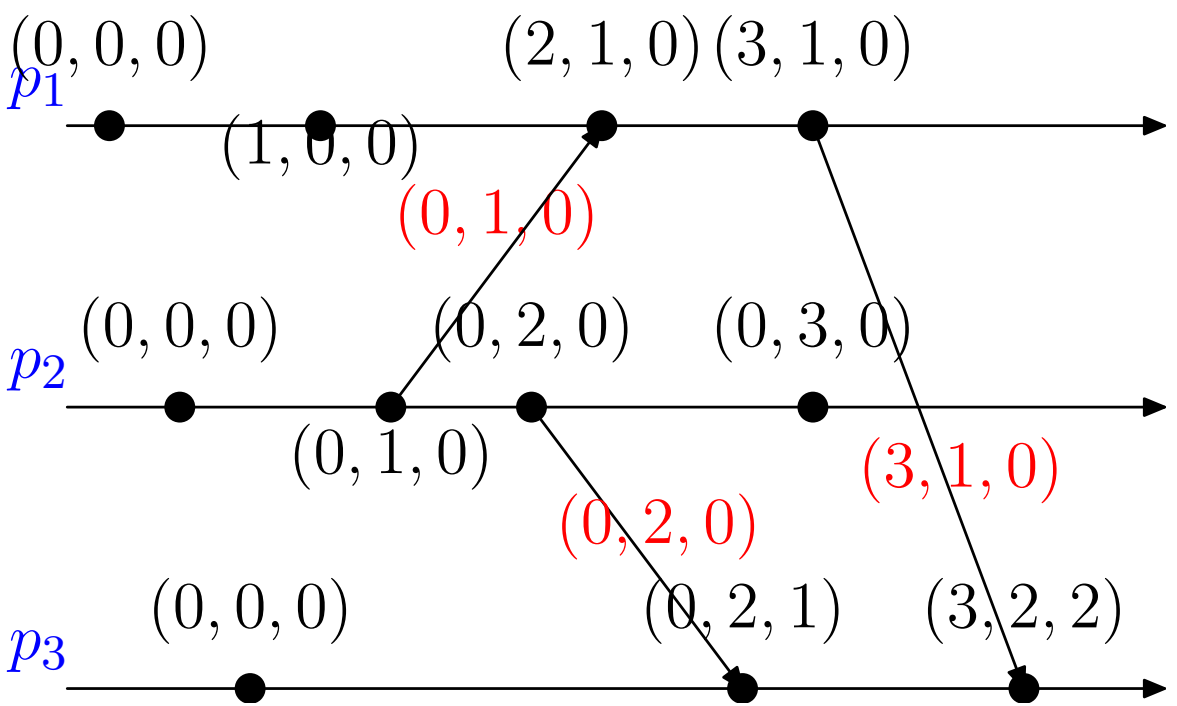
- Supposons que $\mathbf{e}$ est un évènement de S_i. Donc $\mathbf{e} = \mathbf{e}'$.
- Supposons que $\mathbf{e}$ n'est pas un évènement de S_i.
  - $\mathbf{e}' \in PASSE(\mathbf{e})$
  - il n'existe pas un évènement $\mathbf{e}''$ de S_i tel que $\mathbf{e}'' \in Futur(\mathbf{e}') \cap Passe(\mathbf{e})$
- on a $|PASSE(\mathbf{e}') \cap \{\mathbf{e}''$ appartenant à $S_i\}| = EV_{\mathbf{e}'}[i]$

□

La relation d'ordre.

La relation d'ordre suivante peut par ailleurs être définie sur les estampilles vectorielles :

$$EV_e \preceq EV_{e'} \text{ si et seulement si } \forall i,\ EV_e[i] \preceq EV_{e'}[i]$$



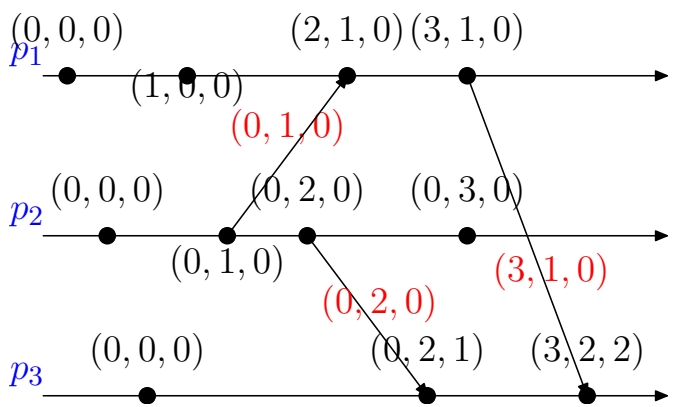
Remarques (1/2)

Propriété

$e \rightsquigarrow e'$ si et seulement si $EV_e \preceq EV_{e'}$

Démonstration.

- Si $e \rightsquigarrow e'$ alors, $EV_e \preceq EV_{e'}$
  - $Passe(e) \subseteq Passe(e')$,
  - $\forall i,\ EV_e[i] \leq EV_{e'}[i]$
- Si $EV_e \preceq EV_{e'}$ alors $e \rightsquigarrow e'$
  - Seul un état local de S_i peut modifier la $i$ème composante de $EV[i]$.



Remarques (2/2)

Propriété

$e \parallel e'$ si et seulement si $EV_e \parallel EV_{e'}$

Démonstration.

Soit S_i et S_j les sites appartenant à e et e'

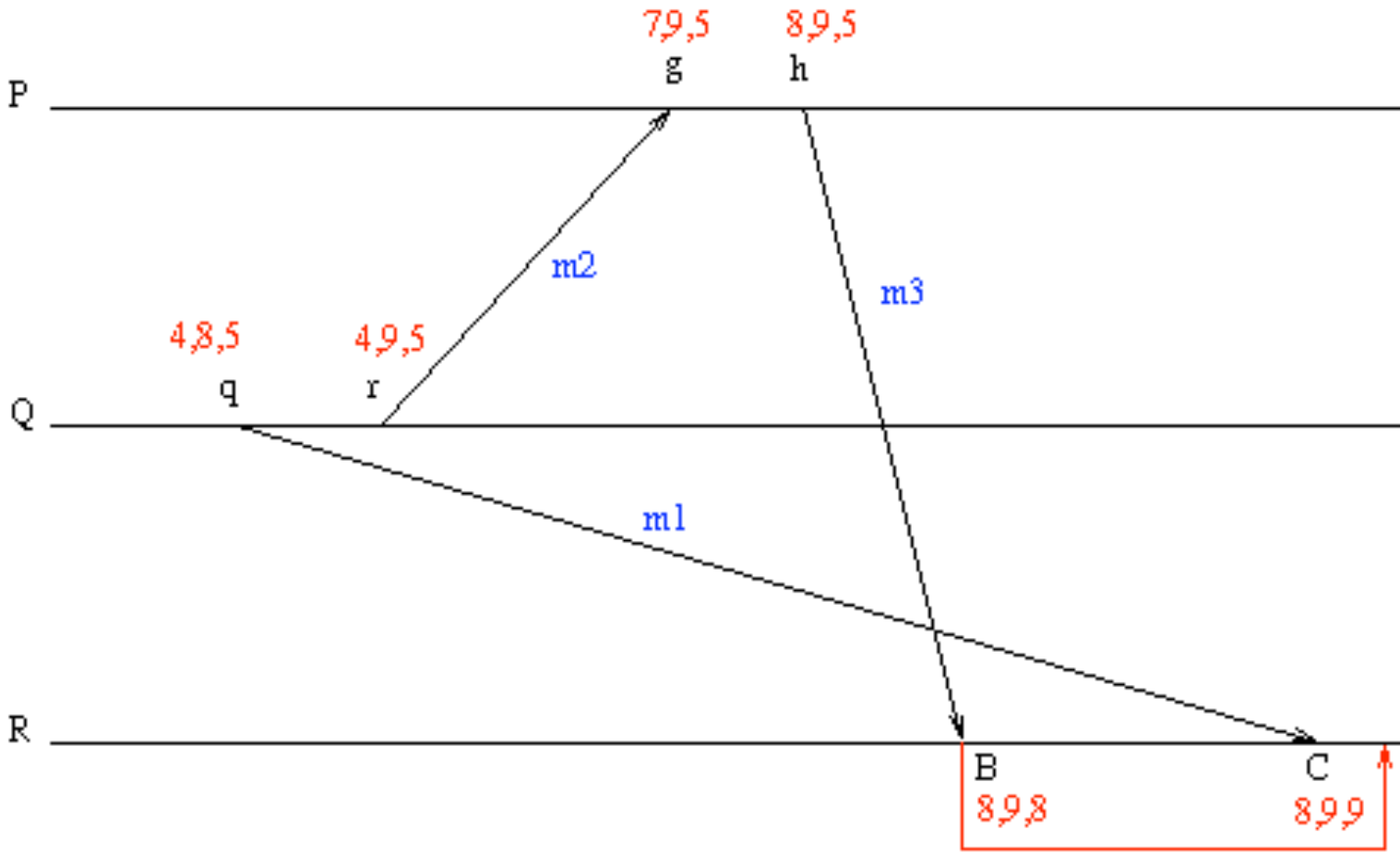
- si $j = i$ alors, e et e' ne peuvent pas être concurrents.

  donc $j \neq i$
- Soit $x(e, j)$ le plus grand élément du $Passe_j(e)$
- on a $x(e, j) \rightarrow^+ e'$ (car si $e' \rightsquigarrow x(e, j)$, alors $e' \in Passe(e)$)

  donc : $Passe_j(e) = Passe_j(x(e, j)) \subset Passe_j(e')$
- Par conséquent : $EV_e[j] < EV_{e'}[j]$
- Utilisant le même raisonnement : on a $EV_{e'}[i] < EV_e[i]$

□

Cet ordre permet-il la délivrance causale ? NON



En résumé

- Horloge scalaire :

  HL_i : ce que p_i connaît du système (nbre d'évènements)

- Horloge vectorielle :

  $HV_i[j]$:ce que p_i connaît du site p_j

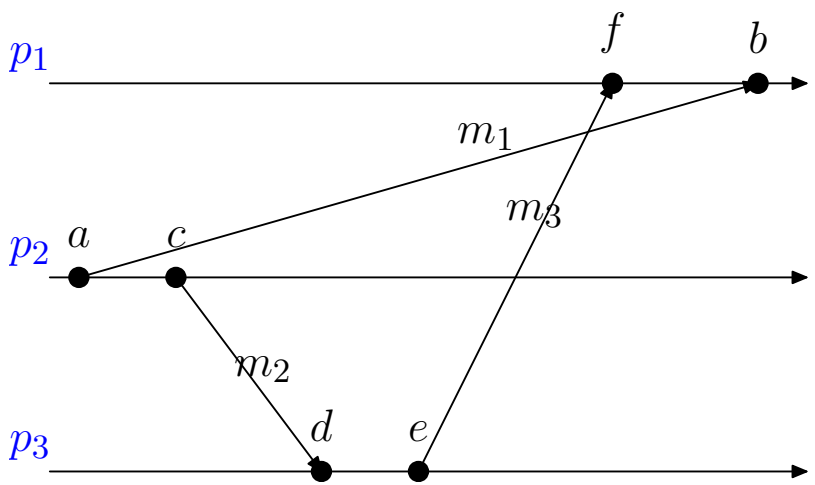
Plus précisément

État d'un système réparti

- Notion non-triviale : observateur universel ayant un accès instantané à l'ensemble du système.

Irréalisable

- Notion de coupure : capture l'état du dernier évènement avant "la photo" sur chaque site.



Coupure

Définition

Une coupure C est l'ensemble des d'évènements tels que

Si $e \in C$ et si e' précède localement e, alors $e' \in C$

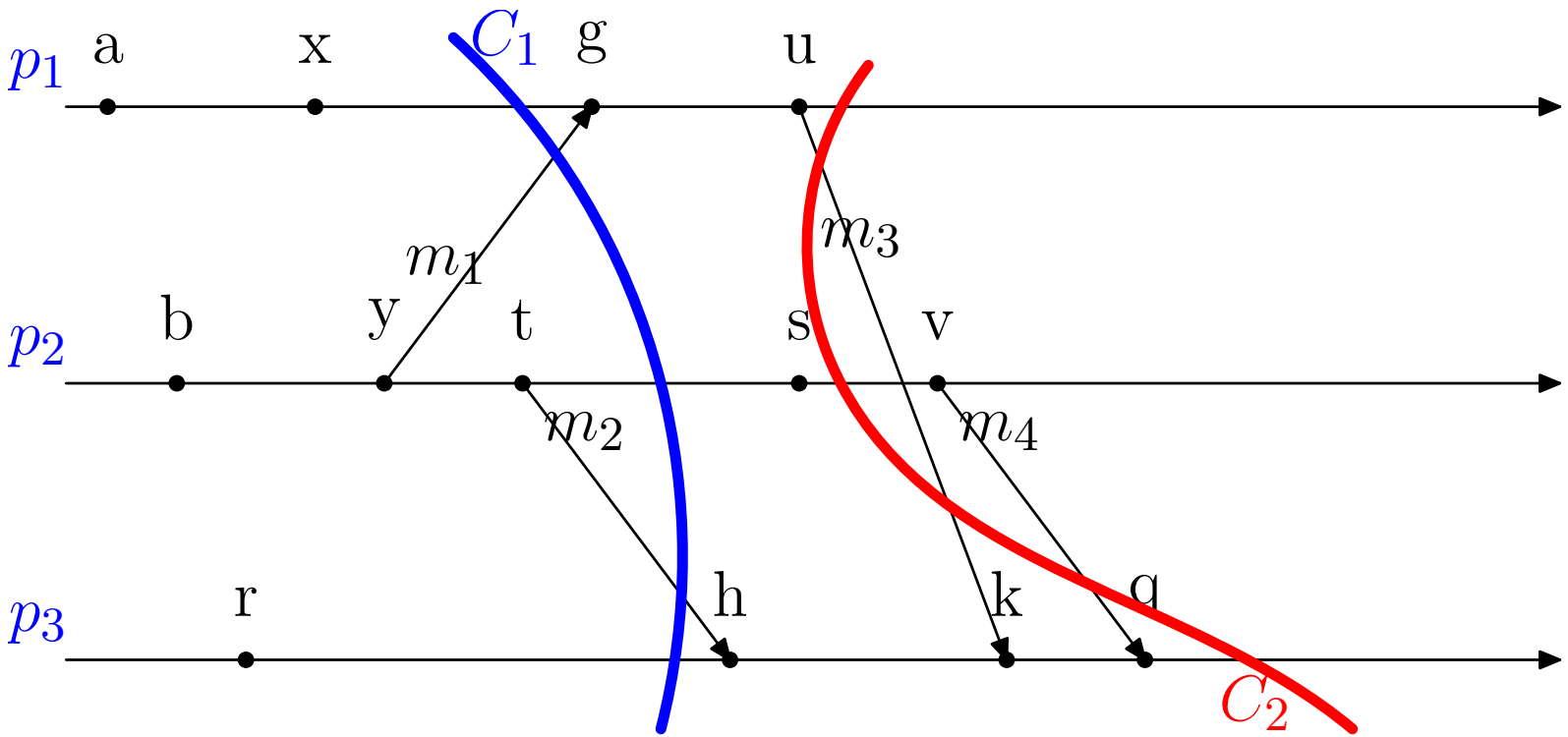


Figure: la coupure C_1 contient les évènements $\{a, x, b, y, t, r\}$

Coupure cohérente

Définition

Une coupure C est une coupure cohérente si :

$$(e \in C) \text{ et } (e' \rightsquigarrow e) \Longrightarrow e' \in C$$

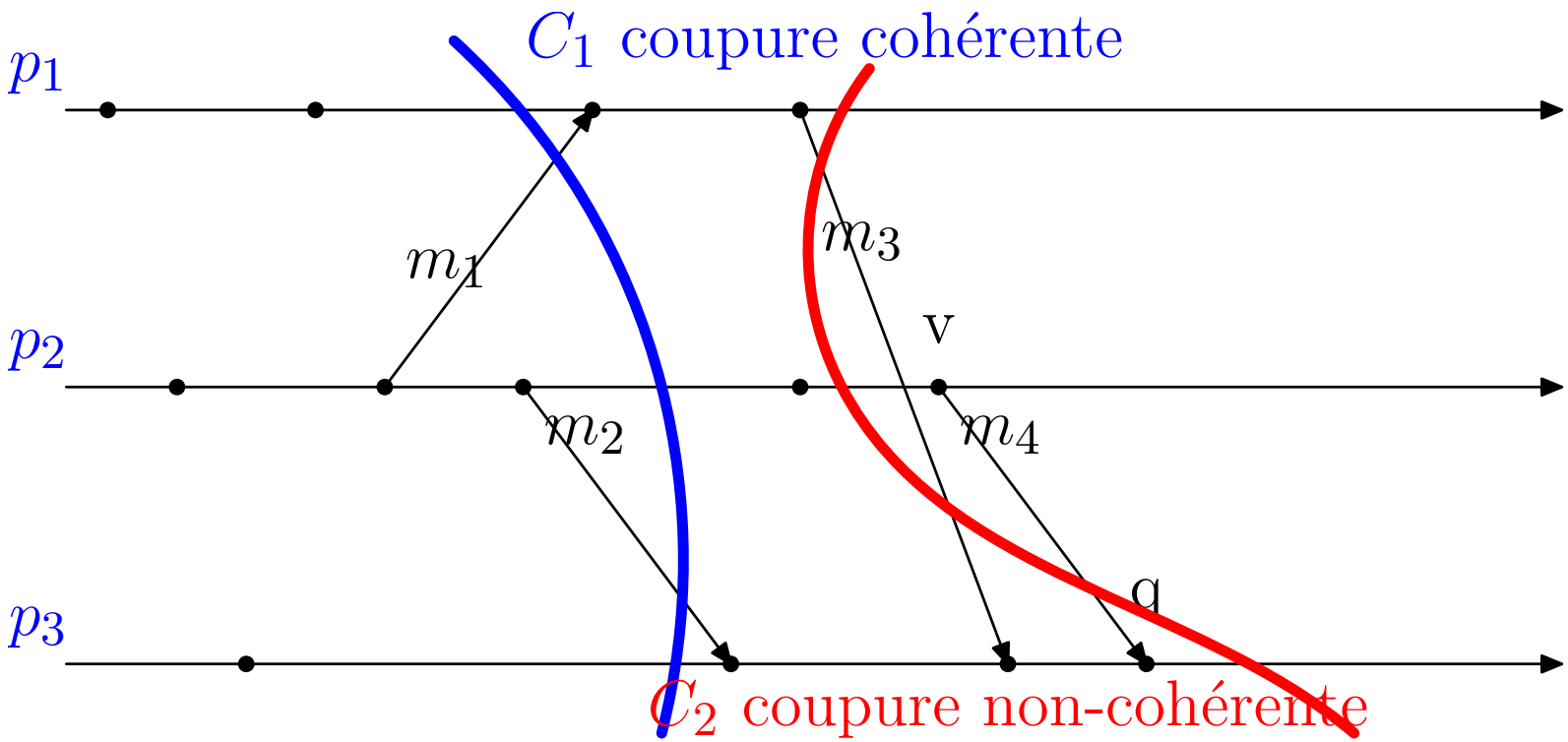
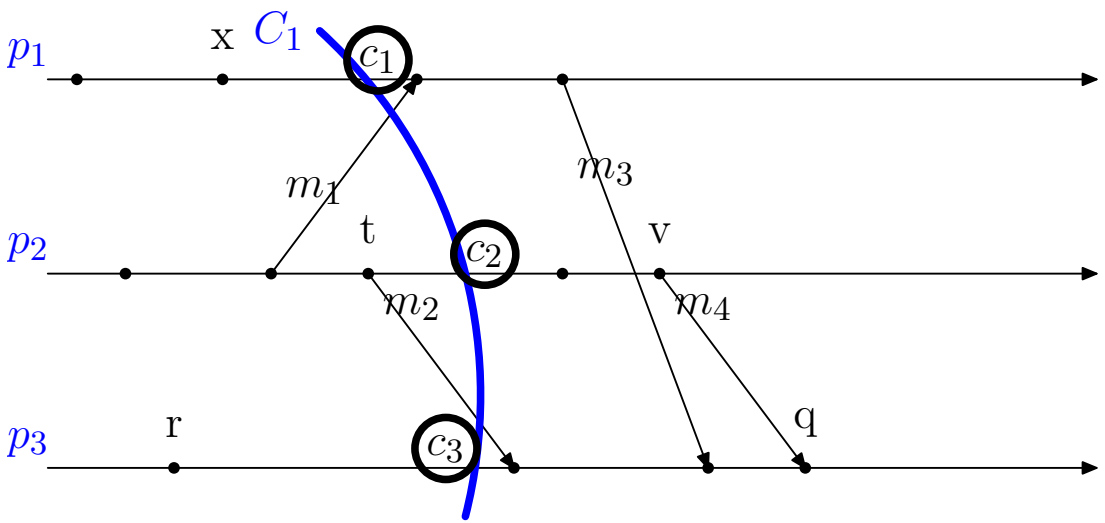


Figure: C_2 non-cohérente car $v \notin C_2$ et $q \in C_2$

Caractérisation des coupures cohérentes (1/2)

La coupure C peut être définie par tous les derniers évènements c_i de chaque site i.

Notation : $C = (c_1, \ldots, c_n)$.



$$C_1 = (c_1, c_2, c_3)$$

Caractérisation des coupures cohérentes (2/2)

Propriété

Soit c_i un évènement de S_i n'étant pas un envoi ou une émission de message.
$C = (c_1, \ldots, c_n)$ est une coupure cohérente $\Leftrightarrow (\forall i, j,\ c_j \parallel c_i)$

Démonstration.

Soit C est une coupure cohérente.

- Par contradiction, supposons que $\exists i \neq j,\ c_i \rightsquigarrow c_j$.
  - $c_i \rightsquigarrow a_i \rightsquigarrow a_j \rightsquigarrow c_j$ (message entre les deux sites)
  - Comme $a_i \rightsquigarrow c_j$, on a $a_i \in C$.
  - Ceci est en contradiction avec le fait que $a_i \rightsquigarrow c_i$

Soit $C = (c_1, \ldots, c_n)$ avec $\forall i, j,\ c_j \parallel c_i$.

- Soit $a_i \rightsquigarrow c_i$, et $a_j \rightsquigarrow a_i$ (avec a_x un évènement du site S_x)
- Alors $a_j \rightsquigarrow c_i$, et $a_j \in C$
- Supposons que $c_j \rightsquigarrow a_j$: on aurait $c_j \rightsquigarrow a_j \rightsquigarrow a_i \rightsquigarrow c_i$ en contradiction avec le fait que $c_j \parallel c_i$.
- Donc C est une coupure cohérente.

Plus précisément

Enregistrement l'état d'un système

- Motivation :
  - Observation, détection de propriétés.
  - reprise en cas de panne
- Contraintes :
  - L' état enregistré doit être cohérent
  - Le coût de l'enregistrement doit être raisonnable
- Difficulté :
  - opérations locales /propriété globale

Protocole de Chandy-Lamport [1985]

Hypothèses :

- les canaux de communication entre processus sont FIFO
- Un seul processus décide de lancer la procédure d'enregistrement

Objectif :

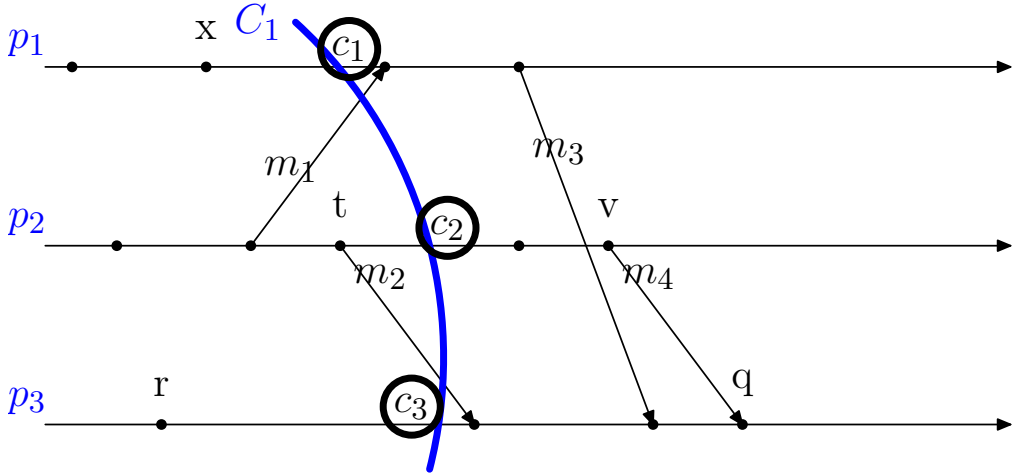
enregistrement d'un état cohérent en un temps fini

sous quelle forme ?

Définition d'un état.

L'état d'un système de réparti est

- l'ensemble des états de chacun de ses sites
- et l'ensemble des états de chacun de ses canaux de communications

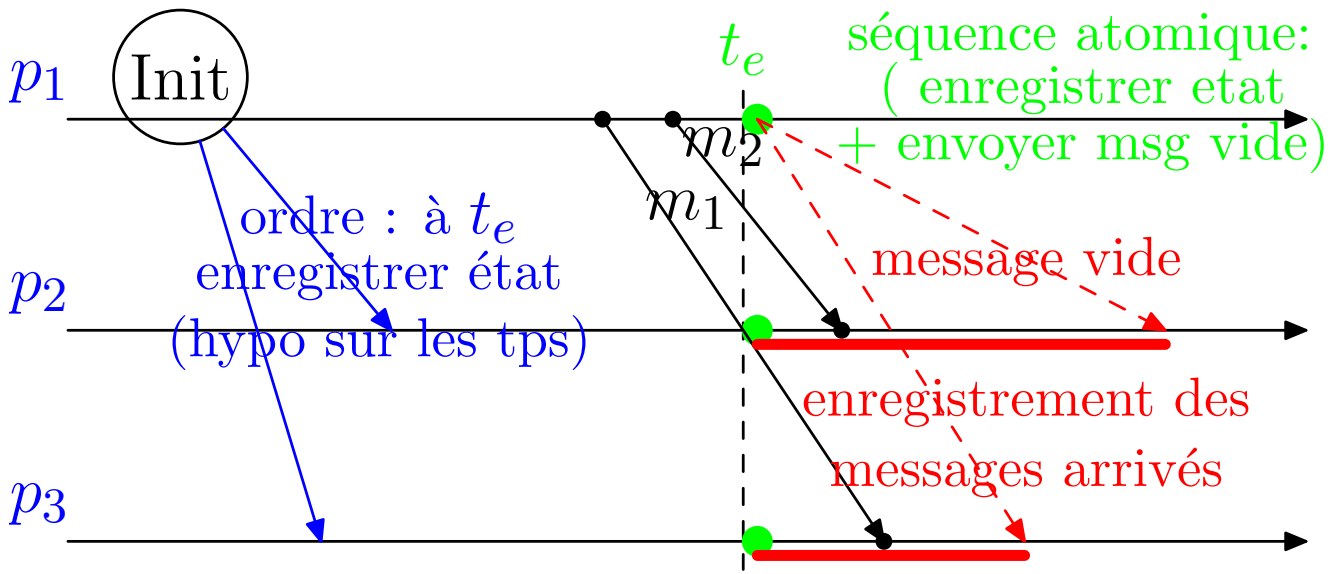


- état de p_1 :x
- état de p_2 : t
- état de p_3 : r
- état de $p_2 \rightarrow p_3$: m_2
- état de $p_2 \rightarrow p_1$: m_1

Première version : (1/2)

Hypothèses :

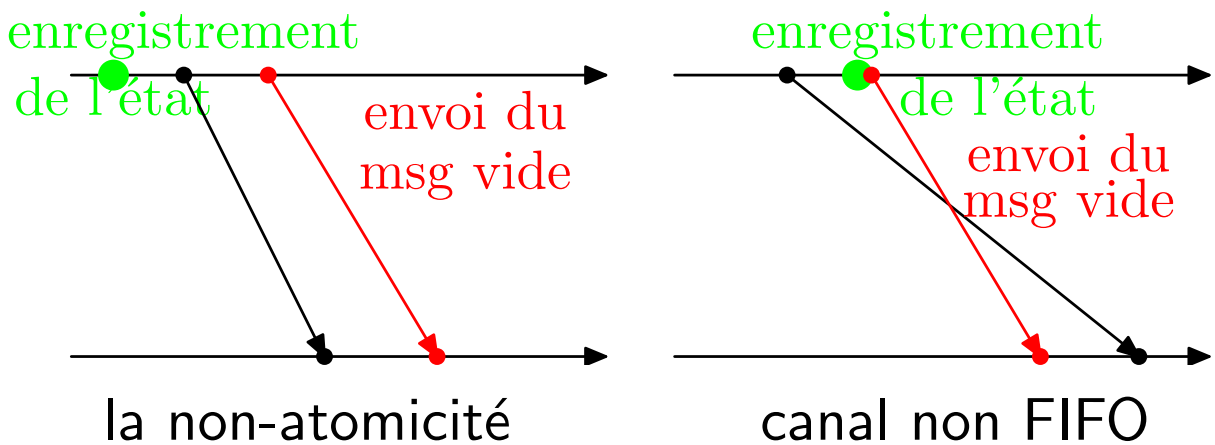
- horloge globale *HR* (temps réel).
- délai de transmission et rapport des vitesses des sites bornés.



Première version : (2/2)

Propriétés : la coupure C définit au tps t_e est une coupure cohérente

- si $e \in C$ et si $e' \rightsquigarrow e$ (localement)
  alors $HR(e') < HR(e)$ et $e' \in C$
- L'état du canal est correct du
  - au fait de l'atomicité des instructions `enregistrement état ; envoi message vide`
  - et du fait que le canal de communication est FIFO



la non-atomicité canal non FIFO

Deuxième version :(1/3)

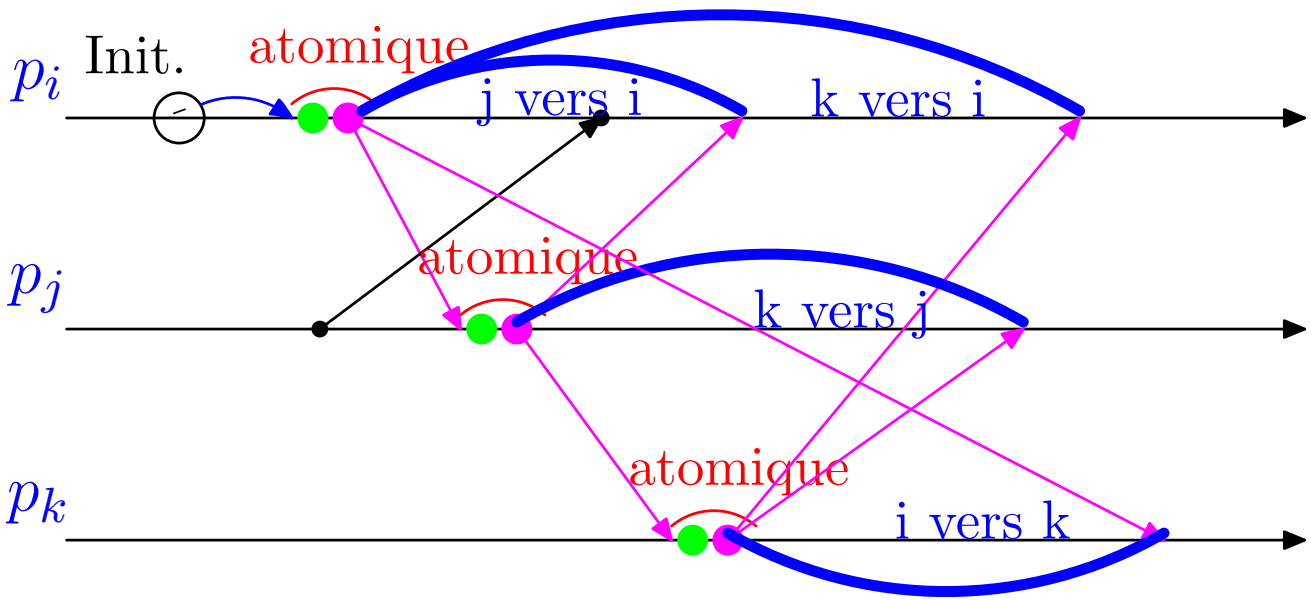
Algorithme sur le site p_i :

1. Lors de la 1ière réception du marqueur depuis p_j
   1.1 enregistrer état de p_i
   1.2 envoyer le marqueur à tous ses voisins
   1.3 état du canal $j \rightarrow i := \emptyset$
   1.4 enregistrer les messages sur les canaux entrants

   atomicité

2. Lors de la réception suivante du marqueur depuis p_k :
   état du canal $k \rightarrow i :=$ msg reçus depuis l'enregistrement.
   ne plus enregistrer les messages provenant de p_k

Deuxième version :(2/3)



Deuxième version :(3/3)

Propriété

La coupure C ainsi calculée est une coupure cohérente.

Démonstration.

En exercice ?

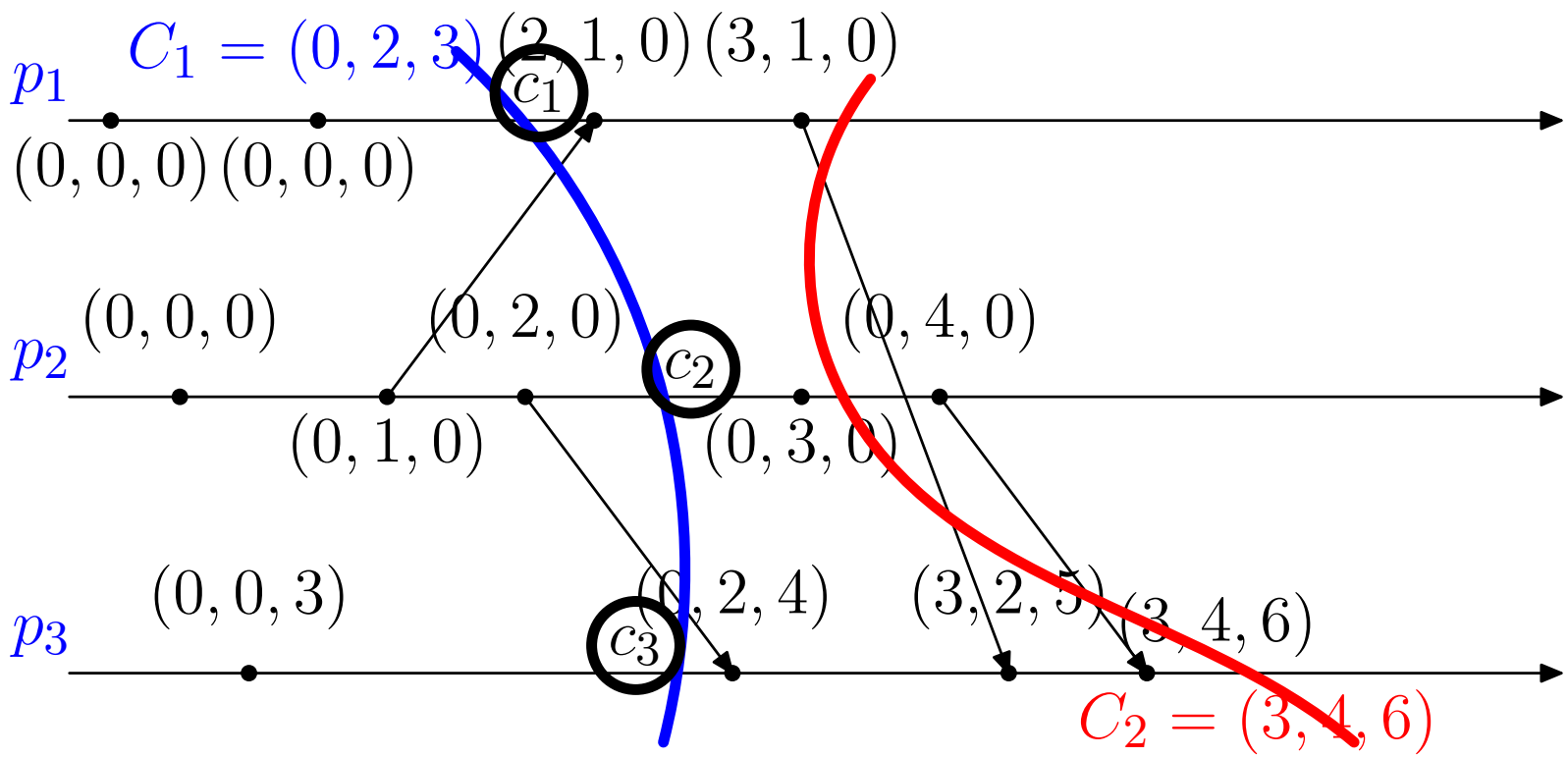
□

Coupures cohérentes et horloges vectorielles

- La date de la coupure $C = (c_1, \ldots, c_n)$ est définie par

$$VH(C) = sup(VH(c_1), \ldots, VH(c_n))$$

- Illustration

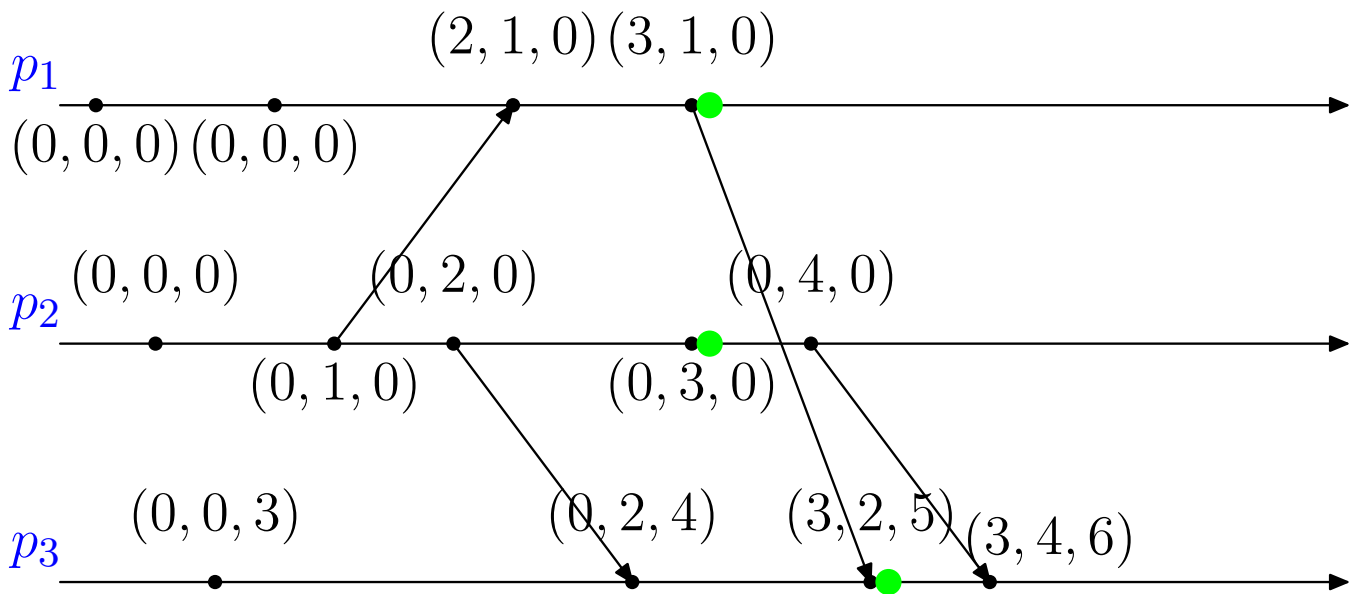


Condition pour les coupures soient cohérentes.

Propriété : C cohérente $\Leftrightarrow VH(C) = (VH_{c_1}[1], \ldots, VH_{c_n}[n])$

Démonstration.

- Soit C cohérente. $\forall i \neq j,\ VH_{c_i}[i] \geq VH_{c_j}[i]$.
  - $VH_{c_i}[i]$ est modifié par un évènement local ou d'un msg passé



Suite de la démonstration

- Soit $C = (c_1, \ldots, c_n)$ non-cohérente.
  - $\exists i, j$ tels qu'un message m de p_i émis après C a été reçu par p_j avant C. Soit $EV(m)$ son estampille
  - Alors $VH(c_i)[i] < EV(m)[i] \leq VH(c_j)[i]$
    donc on a $(VH(c_1)[1], \ldots, VH(c_n)[n]) < VH(C)$