Distribution, coordination et temporalité

Master 1 Androide – UE FoSyMa

Avril 2019

Cédric Herpson
cedric.herpson@lip6.fr

Le jeu des 16 illusions

Les 8 illusions des systèmes multi-agents

1. Les agents s'exécutent chacun leur tour
2. Les agents sont homogènes et s'exécutent à la même vitesse
3. Les agents ont accès à des ressources illimitées
4. Les agents sont fiables
5. Les agent sont sûrs
6. Le nombre d'agent ne varie pas au cour du temps
7. Les agents disposent d'une vision globale
8. Les communications respectent les 8 illusions des systèmes distribués

Les 8 illusions des systèmes distribués

[Deutsch1994, Gosling2004, Rotem-Gal-Oz2008]

1. Le réseau est fiable
2. La latence est nulle
3. La bande passante est infinie
4. Le réseau est sécurité
5. La topologie est fixe
6. Le réseau dispose d' *UN* administrateur
7. Les coûts de communications sont nuls
8. Le réseau est homogène

Coordination et systèmes répartis

- Coordonner des actions nécessite de pouvoir :
  - Etablir des prédicats
    - Nécessité d'accéder à un *état* du système considéré
  - Définir un ordre
    - Disposer d'une *relation d'ordre* (totale)

- Problèmes des systèmes répartis :
  - Pas de mémoire commune
  - Pas d'horloge commune

  - Asynchronisme *des traitements*
    - Pas de bornes sur le rapport relatif des vitesses d'exécution sur différents sites
    - Variabilité de la charge et absence de garanties pour l'allocation de ressources
  - Asynchronisme *des communications*
    - Pas de borne supérieure sur le temps de transmission d'un message (internet)

Propriétés de sureté et de vivacité

- **Sureté** (safety) :
  - Un évènement indésirable n'arrivera jamais

  Exemples : Incohérence dans les données, interblocage, famine,..

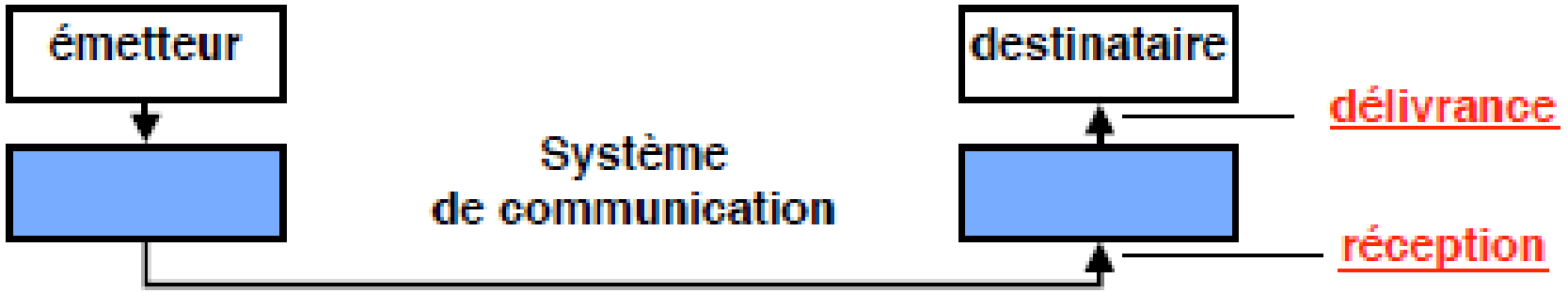
- **Vivacité** (liveness)
  - Un évènement désirable finira par arriver

  Exemples : Ressource disponible, terminaison, message délivré,…

  Les propriétés de vivacité sont (beaucoup) plus difficiles à établir

Rappel sur les messages

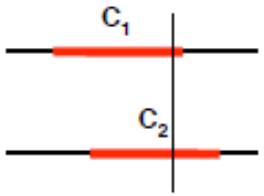
- Distinguer *réception* d'un message (Agent), et *délivrance* à son destinataire (behaviour)



- Propriétés :
  - Un message fini par arriver (mais son temps de transmission n'est pas borné
  - Un message arrive intact

Synchronisation

- L'exécution d'un processus p_1 se traduit par l'occurrence d'une succession d'évènements (e_1^i) appelée *trace* du processus.
  - Cette suite est ordonnée par l'horloge locale du processus
  - Synchroniser deux processus revient à imposer un ordre entre les évènements appartenant à ces deux processus



- Problème:
  - Il faut définir une relation de *précédence globale*
  - Celle-ci ne peut être établie qu'à partir d'*informations locales*

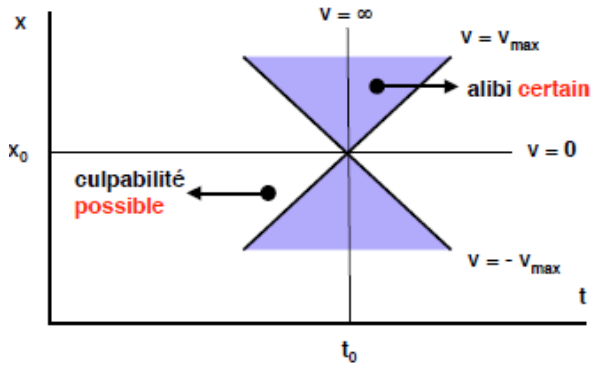
- Solution : Utiliser le **principe de causalité**

Notion de causalité
[Lamport1978]

- e précède causalement e' (e→e') si :

$$e \rightarrow e' \begin{cases} e \text{ précède localement } e' \text{ (sur 1 site, dans 1 processus), ou} \\ \exists \text{ message m tel que } e = \text{émission(m)}, e' = \text{réception(m)}, \text{ ou} \\ \exists e'' \text{ tel que (e précède e'') et (e'' précède e')} \end{cases}$$

- Attention : Cette relation défini une causalité *potentielle*
  - Si e→ e', alors e peut influencer e'
  - Si !(e→e') alors il est certain que e ne peut influencer e '
  - Si !(e→e') et !(e'→e) on dit que e et e' sont causalement indépendants

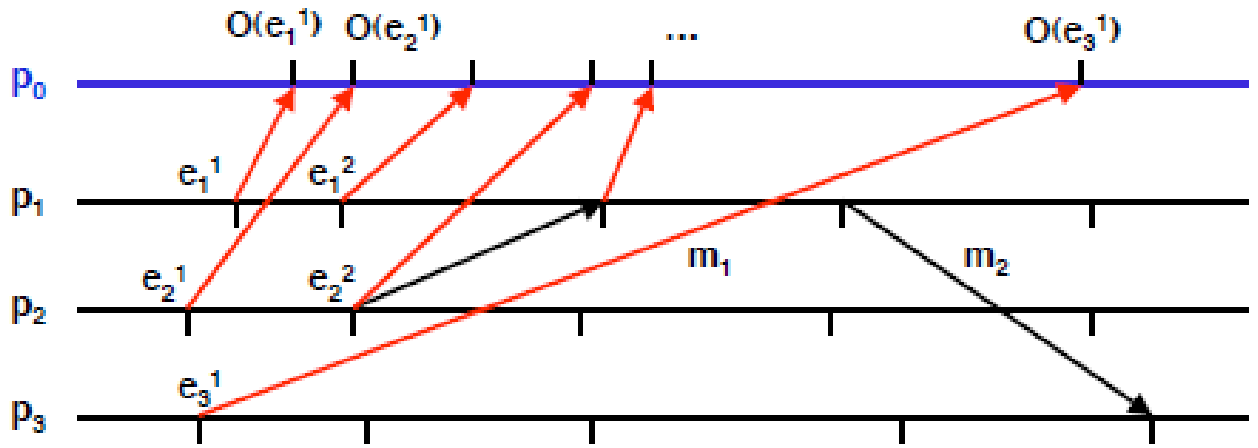


Notion de causalité
[Lamport1978]

- Comment construire un système de datation des évènements qui soit compatible avec la notion de causalité ?

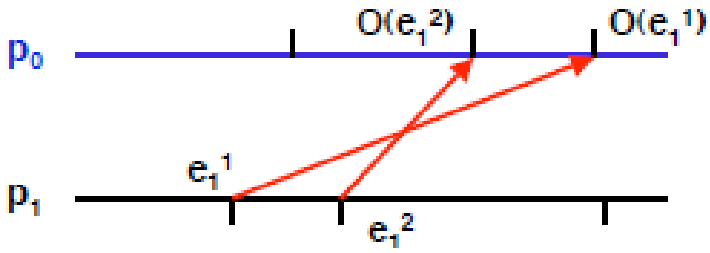
Solution 1 :

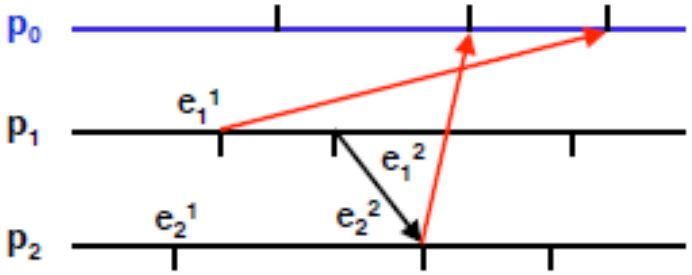
Utiliser un processus observateur centralisant tous les évènements du système considéré sous la forme d'observables



Problèmes liés à l'asynchronisme

- L'ordre des observations ne respecte pas nécessairement l'ordre d'occurrence des évènements.





- Une observation est un ordonnancement particulier des évènements du système considéré

→ Une observation est dite *valide* si :

Pour tout couple (e,e') tel que e→e', on a O(e)→O(e')

Problèmes liés à l'asynchronisme

- Chaque évènement transmis à l'observateur est estampillé par l'horloge de l'observateur.
  - Chaque observation est donc un couple (e,H(e))

Dans un système où les observations sont ordonnées par des estampilles H, une condition suffisante de validité des observation est :

$$e \rightarrow e' \Rightarrow H(e) < H(e')$$

On parle de *condition de validité faible* car l'implication n'est valable que dans un sens.

Comment construire une horloge qui garantisse cette propriété ?

Horloge logique de Lamport

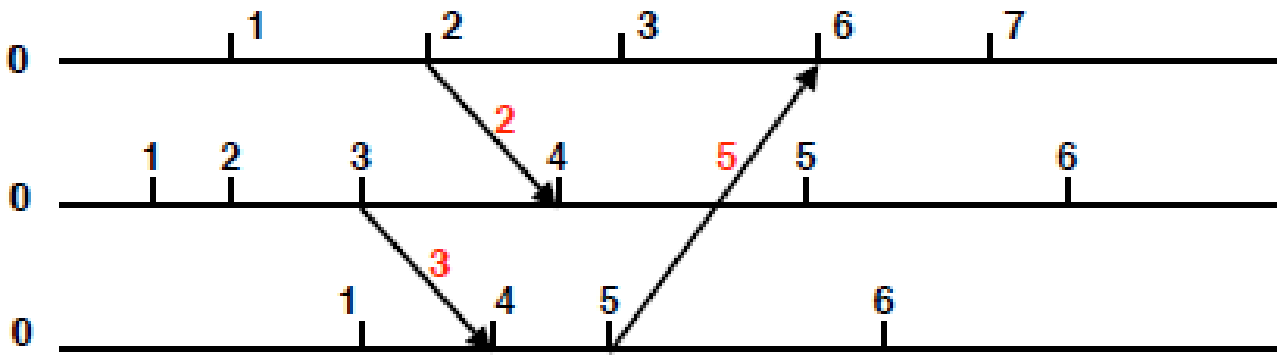
- On va associer à chaque site/machine i un compteur d'évènements H_i
- Un évènement e sur i est daté par la valeur courante de Hi

Règle de fonctionnement d'une horloge sur i

- Init : Hi =0
- Evènement local : Hi= Hi +1
- Envoi d'un message m: on envoie (m,Em)
  - Avec Em=Hi, et Hi a été incrémenté pour compter le message comme un évènement
- Réception d'un message (m,Em) : Hi= max(Hi,Em)+1

Horloge logique de Lamport

- Exemple :

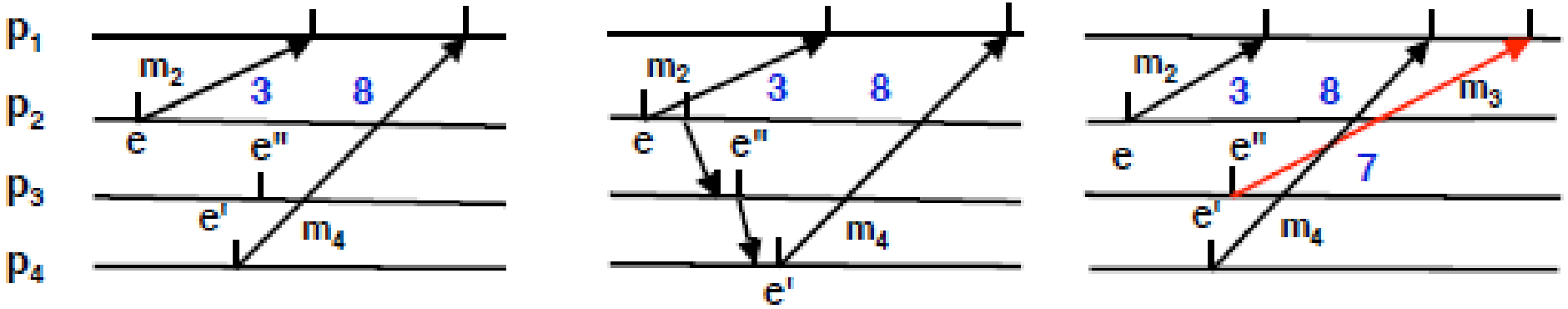


- H satisfait la condition de validité faible : e→e' ➔ H(e)<H(e')
- L'ordre n'est pas strict, deux évènements peuvent avoir la même estampille (ils sont alors causalement indépendants)
- Pour disposer d'un ordre strict on ajoute un paramètre (n° proc)

Horloge logique de Lamport

L'asynchronisme et les évènements manquants

- P1 reçoit les messages des autres processus
- On souhaite qu'ils lui soient délivrés dans l'ordre de leurs estampilles.
- On a délivré m2(3) et on reçoit m4(8), peut-on le délivrer ?



- Les 3 situations sont *indistinguables* avec les seules HL.
- Si HL(e)<HL(e'), existe-t'il e'' tel que e→e''-->e' ?
- Cette question est *insoluble* (Il s'agit d'une propriété de vivacité).

Horloge logique de Lamport

L'asynchronisme et les évènements manquants

- La question de la délivrance des messages est insoluble, comment faire dans la pratique ?

- Un message reçu est dit **stable** si aucun message portant une estampille inférieure ne parviendra au destinataire.
  - Pour s'en assurer, on attend d'avoir reçu un message de tous les émetteurs potentiels.

  Problèmes :
  - Il faut connaitre tous les émetteurs potentiels
    - Certaines situations ne s'y prêtent pas.
  - Que faire si on n'a rien reçu d'un processus ?
    - Lui envoyer un message request, faire un ping
  - On ne garanti pas la terminaison en temps fini.
    - On utilise généralement un délai de garde, qui ne couvre pas tous les cas.

Horloge logique de Lamport

- File d'attente répartie
  - Exclusion mutuelle (trésor..)
  - Mise à jours de copies multiples (cartes..)
  - Diffusion cohérente
- Détermination de l'accès le plus « récent »
  - Gestion du cache
  - Gestion de la mémoire virtuelle répartie
- Génération de noms uniques
  - <Adresse de site, estampille locale>

Horloge logique de Lamport
L'exemple de la diffusion causale (1/2)

- (Rappel) Diffusion :
  - Envoi d'un message m à un ensemble de destinataires
    - Diffusion générale (broadcast)
    - Diffusion sélective (multicast)

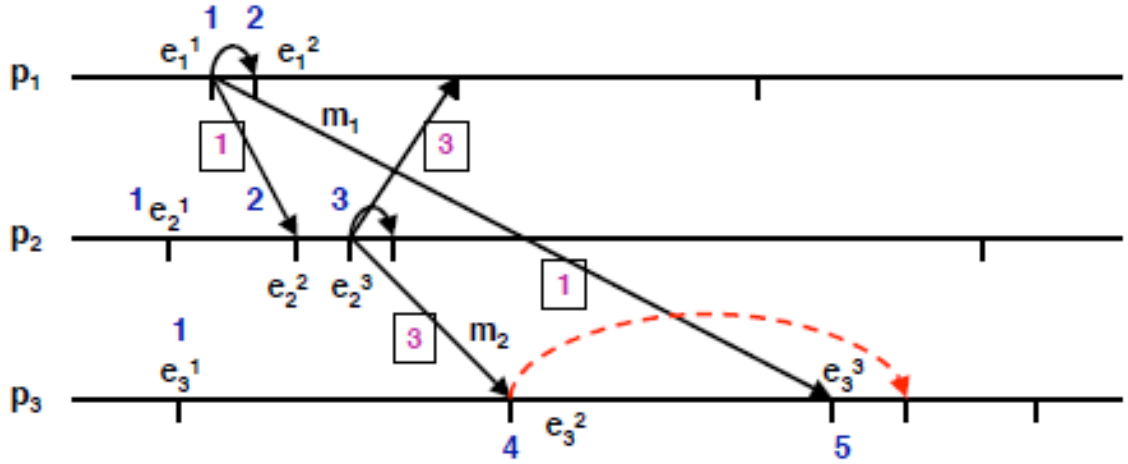
- Mécanisme fondamental pour partager un état et/ou maintenir la cohérence de données réparties
  - Spécifier les invariants à maintenir (fonction de la forme de diffusion considérée)
  - Garantir les propriétés spécifiées en cas de défaillance est un problème très difficile

- Dans le cas de la diffusion causale :
  - On suppose les communications fiables
  - La diffusion causale doit garantir :

$\forall m, m'$; ∀i : send(m) → send(m') ➔ deliver$_i$(m) → deliver$_i$(m')

Horloge logique de Lamport L'exemple de la diffusion causale (2/2)

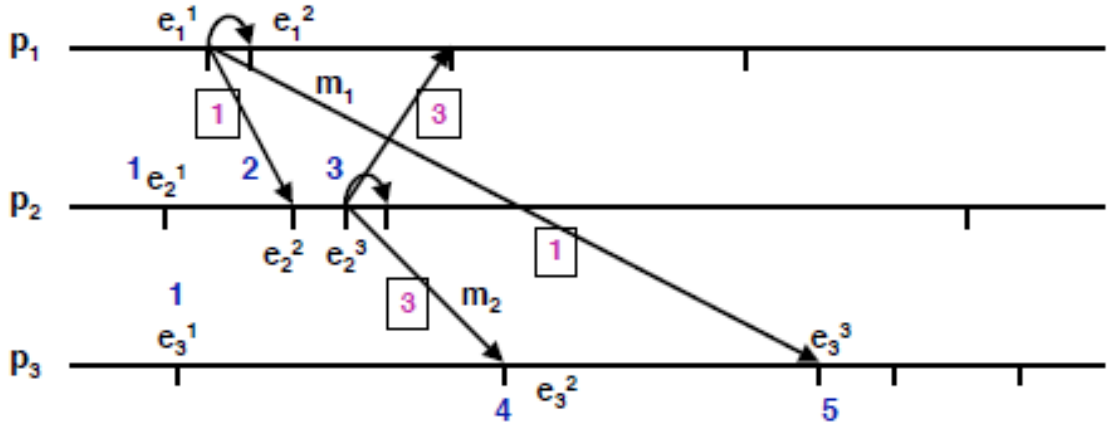
- Cette exécution pose-t-elle un problème ?



- Elle viole la causalité. En effet m2 dépend causalement de la réception de m1, mais m2 est reçu avant m1, le message m2 reçu par p3 n'est dont pas stable.
- Les horloges logiques ne permettent pas de détecter cette incohérence.

Horloge logique de Lamport
Exemple : Mise à jour de copies multiples

- Une carte est maintenue en 3 exemplaires, (une par agent).
- Chaque agent peut consulter la carte ou la mettre à jours.

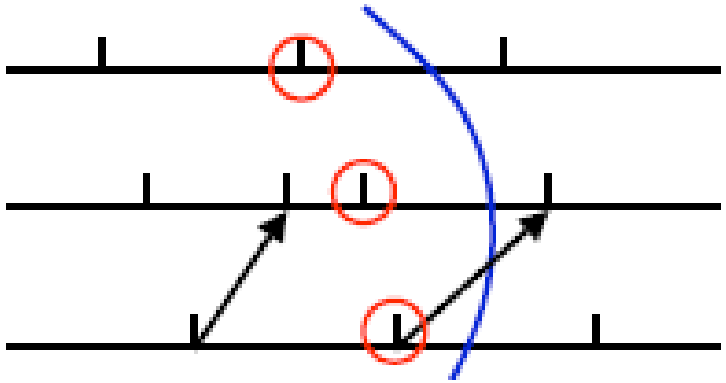


- Si m1 et m2 sont des mises à jours, l'exécution ci-dessus conduit à une incohérence si les mises à jours ne commutent pas
- Principe de mise à jours cohérente : n'appliquer la mise à jours que si elle est stable.
- Pour la consultation, on lit la valeur locale si on peut accepter une valeur périmée, sinon on assure d'abord la mise à jours de la copie.

Etat d'un système réparti

- L'idée même d'état est un problème
  - Elle suppose l'existence d'un observateur universel ayant accès de manière instantanée à l'ensemble du système.
  - Dans la pratique, un tel observateur n'existe pas

- Notion de **coupure** :



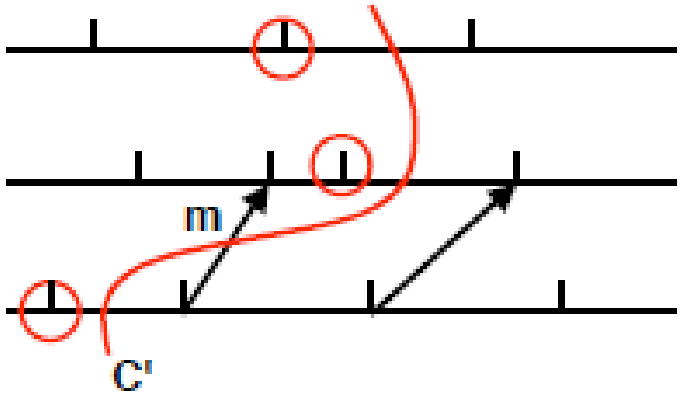
  - Intuitivement : capture d'écran
  - On capture l'état résultant du « dernier » évènement « avant » la coupure de chaque processus
  - Une coupure est un ensemble d'évènements (ceux « à gauche » de la coupure)

Etat d'un système réparti

- Une coupure est dite cohérente si :

$$(\mathrm{e}' \epsilon\ C\ et\ e \rightarrow e') \Rightarrow e\ \epsilon\ C$$

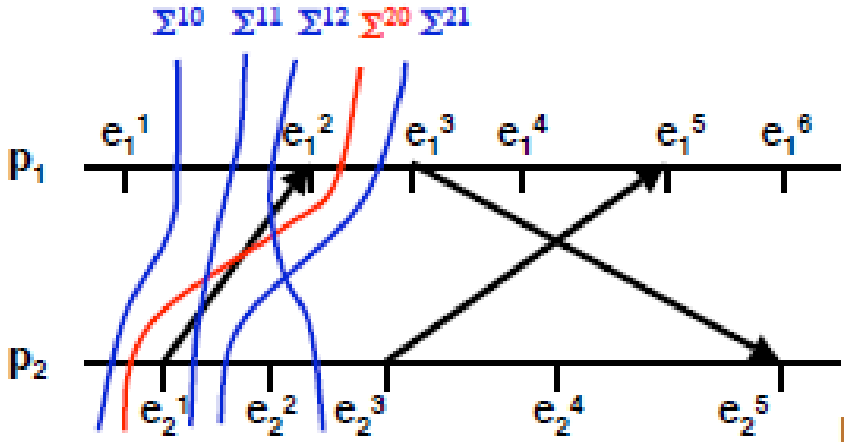
- Une coupure cohérente est fermée par la relation de précédence causale
- Tout message reçu dans la coupure a donc été envoyé dans celle-ci (mais l'inverse n'est pas
- nécessairement vrai)

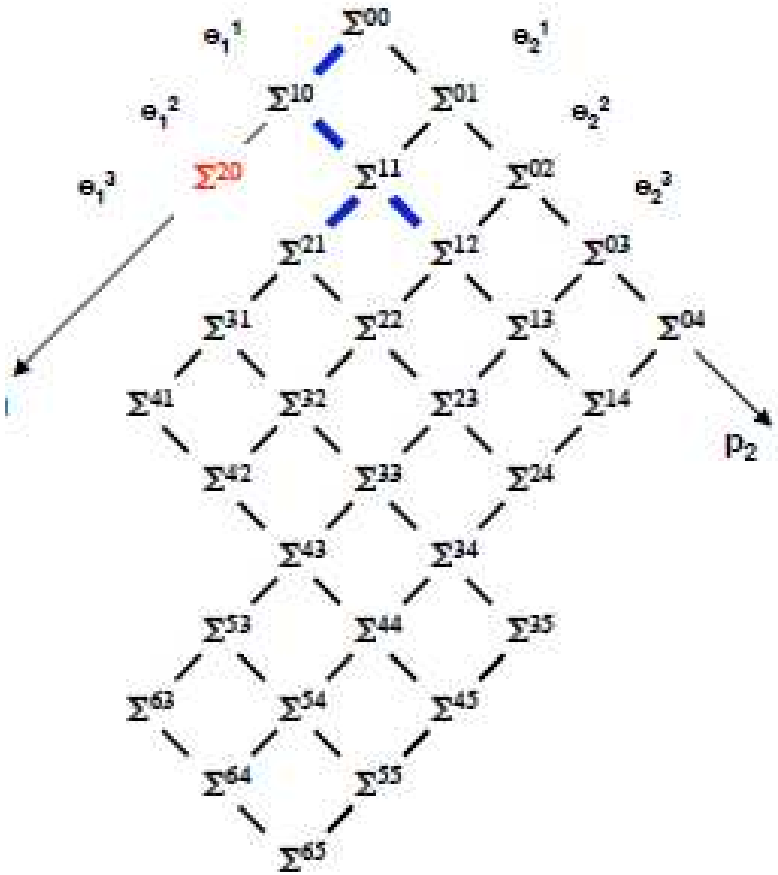


- Une coupure incohérence donne une vision du système qui viole la causalité.
- Le sous ensemble des coupures cohérentes d'un système forme un treillis.

Etat d'un système réparti

- De là, une observation valide correspond à un chemin dans le treillis des coupures cohérentes.





- Remarque : Une même exécution réelle peut conduire à plusieurs observations valides différentes.

Limite des horloges logiques

- Horloge logiques :

$$e \rightarrow e' \Rightarrow H(e) < H(e')$$

Condition *de validité faible* car l'implication n'est valable que dans un sens.

- On cherche à construire un système de datation qui ait la propriété de *validé forte* :

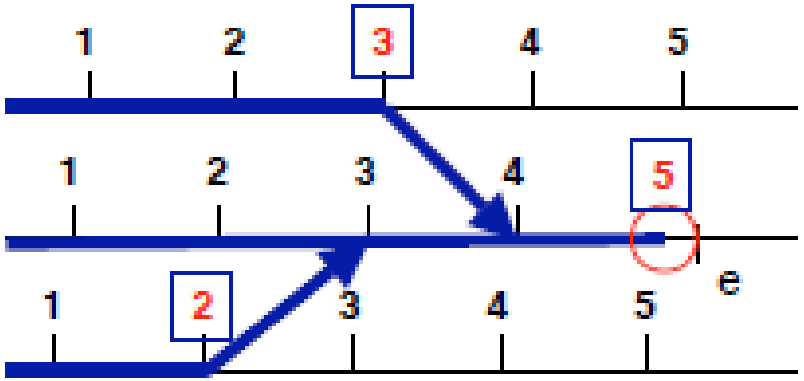
$$e \rightarrow e' \leftrightarrow H(e) < H(e')$$

  - Observation
  - communication causale,
  - maintient de la cohérence du système

Construire un système de datation qui ait la propriété de validité forte

Une idée ?

- Inclure le passé de l'évènement émis dans l'estampille du message.

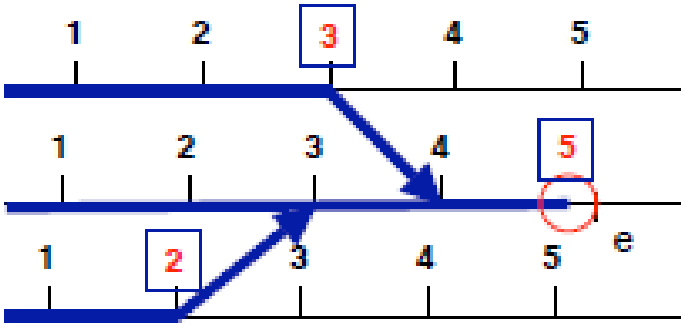


- L'histoire d'un évènement est ainsi définie par un vecteur de n éléments comprenant les numéros des évènements les plus récents de son passé sur *chaque* processus.

- hist(e) = { e' | e'--> e} U {e}
- $\text{hist}_i(e) = \{e' | e' \rightarrow e \text{ et } e' \epsilon\ pi\}$
- $\text{hist}(e) = U\ \text{histi}(e), \{e\}$

Horloges vectorielles [Fidge]

- À tout évènement e est associé un vecteur HV(e) comportant un élément par site j:
- HV(e)[i] = nb d'éléments de $hist_j(e)$ [passé de e dans p_j]
- Si e ∈ p_i, alors :
  - HV(e)[j]= nb d'élements exécutés dans pj dont pi à connaissance au moment de l'occurrence de e (connaissance partagée)
  - $\sum HV(e)\,[\mathrm{j}] - 1 = \text{nb d'évenements strictements antérieurs à e}$
  - HV(e)[i]= nb d'évènements exécutés par p_i avant e (e inclus)

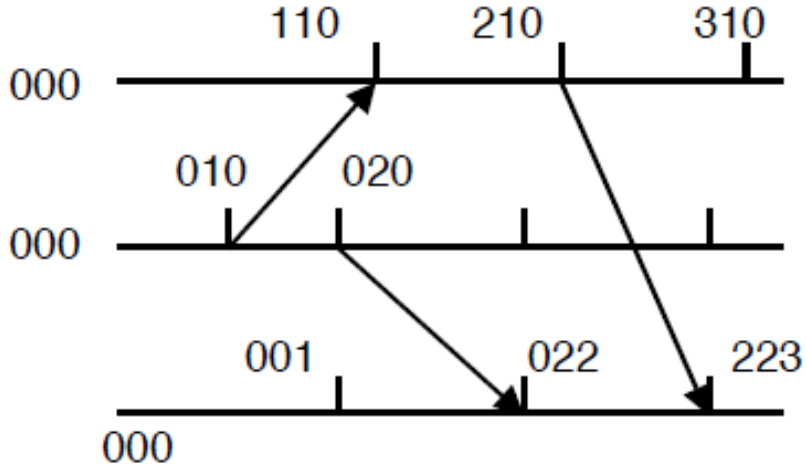


HV(e) = (3, 5, 2)

Horloges vectorielles [Fidge]

- On dispose d'une horloge vectorielle locale sur chaque machine Hvi. Chaque évènement est daté par HVi(e)

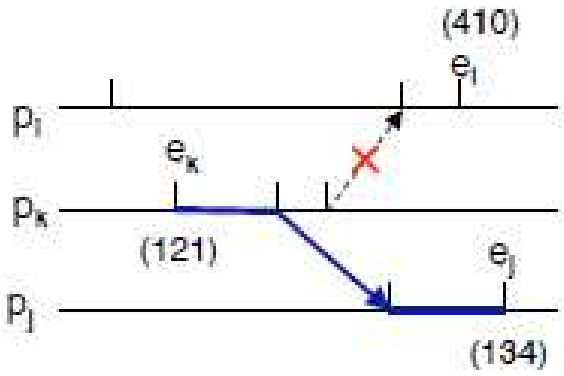
- Mise à jours des horloges sur p_i :
  - Valeur initiale : 000
  - Evènement local sur pi : $Hv_i[i] = Hv_i[i]+1$
  - Emission d'un message m sur pi : $Hv_i[i] = Hv_i[i]+1$ et le message est estampillé par EM= HV_i
  - Réception d'un message (m,Em) :
    - $Hv_i[i] = Hv_i[i]+1$
    - $Hv_i[\mathbf{j}] = \max(Hv_i[\mathbf{j}],Em[\mathbf{j}])$, avec i!=j



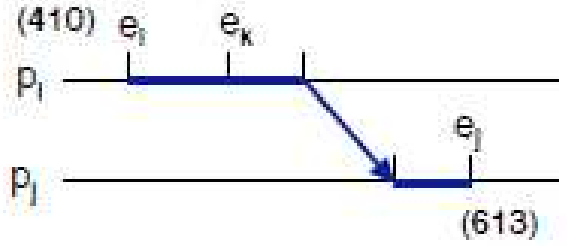
Horloges vectorielles [Fidge]

- Si H(e)<H(e'), existe-t-il e'' tel que e→ e''→ e' ?
  - Horloges logiques : pas de réponse
  - Horloges vectorielles : réponse partielle (on parle de détection faible)

- $Si\ ei \in pi, e_j \in pj\ et\ si \exists\ k \neq j\ tq\ HV(e_i)[k] < HV(e_j)[k],$ $alors \exists\ ek\ tq\ !\ (e_k \rightarrow ei)\ et\ (ek \rightarrow ej)$



Si k = i, alors on a une réponse précise :
si HV(e_i)[i] < HV(e_j)[i], alors ∃ e_k : $e_i \rightarrow e_k \rightarrow e_j$



Horloges matricielles

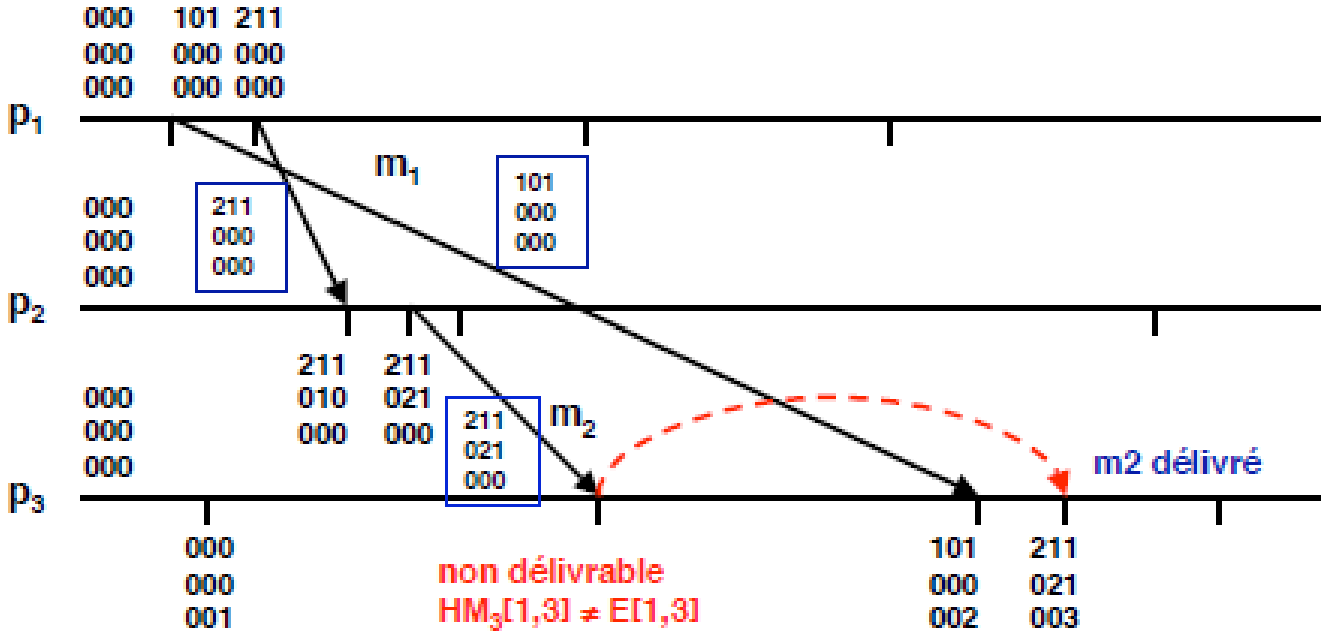
- Intuition :
  - Horloges logiques HLi : ce que pi connait du système
  - Horloge vectorielle : Hvi[j] : ce que pi connait de pj
  - Horloge matricielle: Hmi[j,k] : ce que pi connait de ce que pj connait de pk

- On maintient une horloge matricielle sur chaque machine/site
  - Hmi[j,k] = nb de messages issus de pj vers pk dont pi a connaissance (donc dont l'envoi est causalement antérieur à l'instant présent)
    - Cas particulier Hmi[i,i] : évèenemnts locaux à pi

- On peut maintenant contrôler la délivrance causale
  - Le message ne peut être délivré qu si tous les messages qui lui sont causalement antérieurs ont été délivrés.
  - Les HM sont couteuses $O(n^2)$

Horloges matricielles

- Délivrance causale des messages point à point



Retour sur le consensus

- Systèmes synchrones et asynchrones avec pannes franches
- Problème fondamental :
  - Prise de décision concertée
  - Solutions simples en absence de défaillances
  - Solutions très complexes sinon
  - De nombreux cas d'impossibilité
- Le cas du Paxos [Lamport]