Gestion du temps & état global dans un système distribué

Babahenini Med Chaouki

WEBLIOGRAPHIE

- Durant la préparation de ce cours j'ai eu recours aux supports disponibles sur le WEB et notamment au cours des personnes suivantes:
- ***Gérard FLORIN****: professeur en informatique au CNAM de Paris.*
- ***Michel RIVEILL*** *: professeur d'informatique à l'Ecole Polytechnique de l'Université de Nice*
- ***Yahya SLIMANI*** *: professeur en informatique à la Faculté des Sciences de Tunis*
- **Sacha KRAKOWIAK***: professeur émérite en informatique à l'université Joseph Fourier, Grenoble*
- **Eric CARIOU:** *Maître de conférences au département informatique de l'université de Pau*

État global et horloge

- Pour chaque processus du système
  - État local : valeur des variables du processus à un instant *t*
- État global du système
  - Valeur de toutes les variables de tous les processus du système à un instant *t*
    - Union à l'instant *t* des états locaux de tous les processus
- Problème
  - Un état est lié à un instant *t* mais
    - Chaque processus à une horloge physique locale
    - Pas d'horloge globale dans un système distribué
- La définition d'un état global est possible seulement si on est capable de définir un temps global

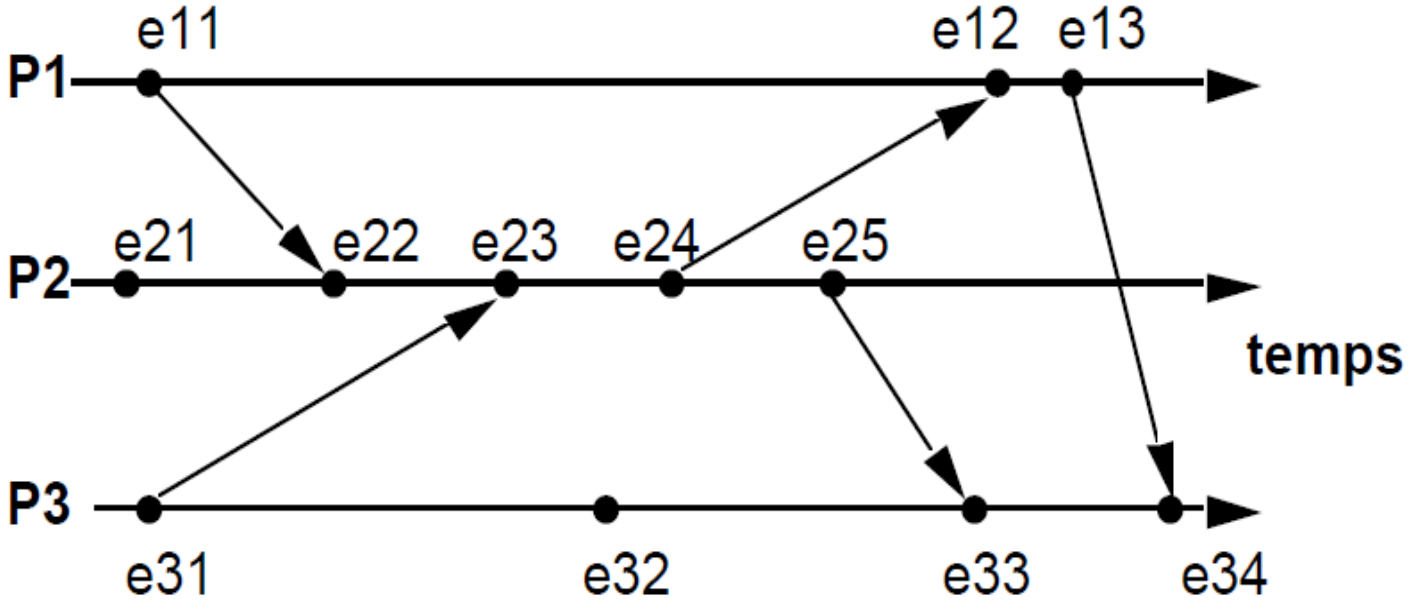
Temps

- Définir un temps global cohérent et « identique » (ou presque) pour tous les processus
  - Soit synchroniser au mieux les horloges physiques locales avec une horloge de référence ou entre elles
  - Soit créer un temps logique
- Synchronisation des horloges physiques locales
  - But est d'éviter qu'une horloge locale dérive trop par rapport à un référentiel de temps
    - La dérive est bornée en augmentation et en diminution
- Deux modes
  - Synchronisation interne
  - Synchronisation externe

Etat global et ordre des événements

- Systèmes répartis à évolution asynchrone
  - pas de mémoire commune (communication par messages)
  - pas d'horloge commune
- Propriétés du système de communication
  - temps de communication potentiellement longs par rapport aux temps de transition locaux
  - temps de transmission variables
  - (éventuellement) non-préservation de l'ordre des messages
- Conséquences
  - perception différente des mêmes événements depuis des sites distincts
  - impossibilité de définir simplement un "état global instantané"
  - conséquences sur la synchronisation, l'observation et la mise au point, la tolérance aux fautes

Synchronisation dans un système réparti



Synchronisation dans un système réparti

- Système = ensemble de processus (1 par site) + canaux de communication
- Processus = séquence d'événements locaux + mémoire locale + horloge locale
- Evénement = changement d'état du processus (événement interne), ou émission d'un message, ou réception d'un message
- Problèmes fondamentaux :
  - définir un état global
  - définir un temps global
  - déterminables localement par tout processus
- temps global = temps virtuel (ordre sur les événements) tout ordre doit être compatible avec la causalité

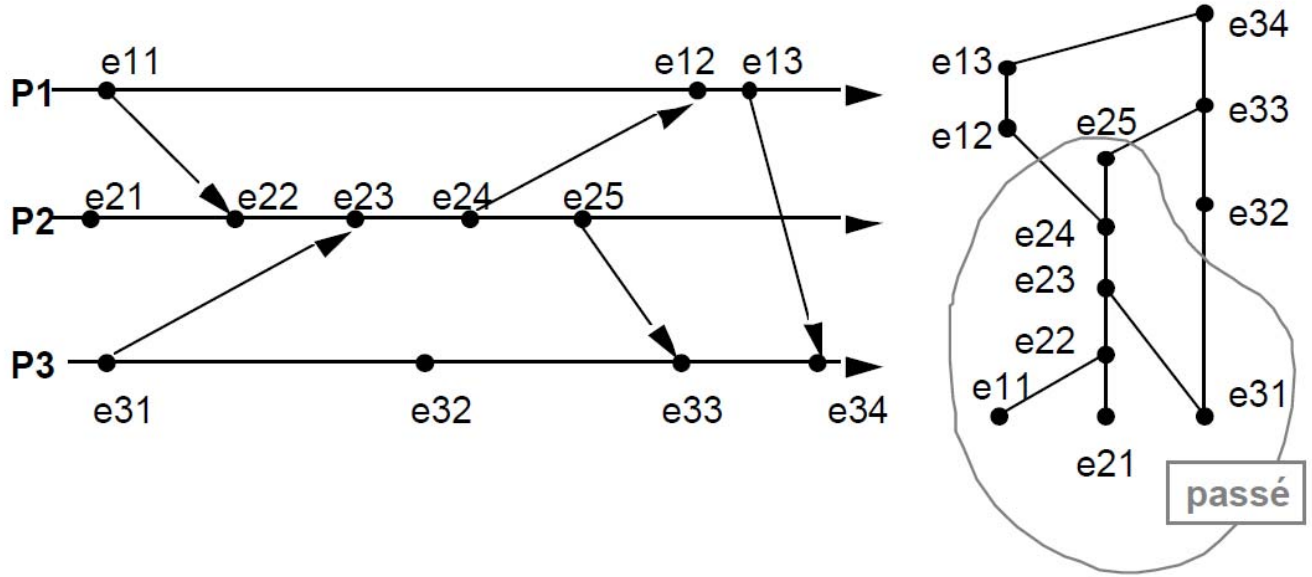
Temps logique : chronogramme

- Chronogramme: Décrit l'ordonnancement temporel des événements des processus et des échanges de messages
- Chaque processus est représenté par une ligne
- Trois types d'événements signalés sur une ligne
  - Émission d'un message à destination d'un autre processus
  - Réception d'un message venant d'un autre processus (interactions entre les sites)
  - Événement interne dans l'évolution du processus (mémoire commune)
- Les messages échangés doivent respecter la topologie de liaison des processus via les canaux

Temps logique : dépendance causale

- Relation de dépendance causale
  - Il y a une dépendance causale entre 2 événements si un événement doit avoir lieu avant l'autre
  - Notation : e → e'
    - *e* doit se dérouler avant *e'*
- Si e → e', alors une des trois conditions suivantes doit être vérifiée pour *e* et *e'*
  - Si *e* et *e'* sont des événements d'un même processus, *e* précède localement *e'*
  - Si *e* est l'émission d'un message, *e'* est la réception de ce message
  - Il existe un événement *f* tel que e → f et f → e'
- Relation de parallélisme : ||
  - e || e' ↔ ¬ ((e →e') v (e' →e))
  - Parallélisme logique : ne signifie pas que les 2 événements se déroulent simultanément mais qu'ils peuvent se dérouler dans n'importe quel ordre

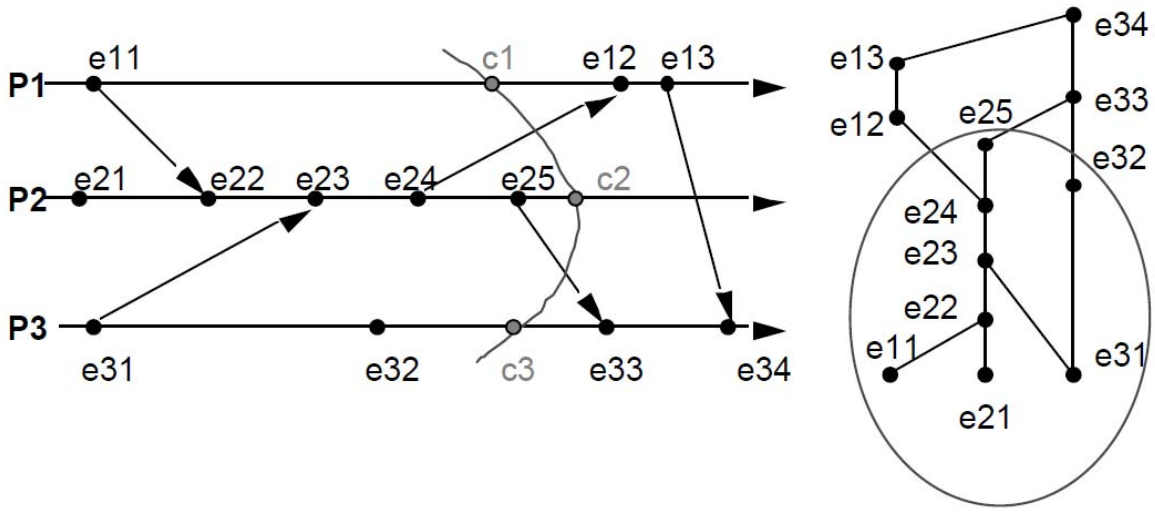
Temps logique : dépendance causale



- Passé (ou historique) d'un événement e (noté hist(e)) : e ensemble des événements e' tels que: e' → e. Seul le passé strict de e peut inflencer e.
- Chaîne causale : e0, ... en tels que ei-1 → ei, i=1,...n

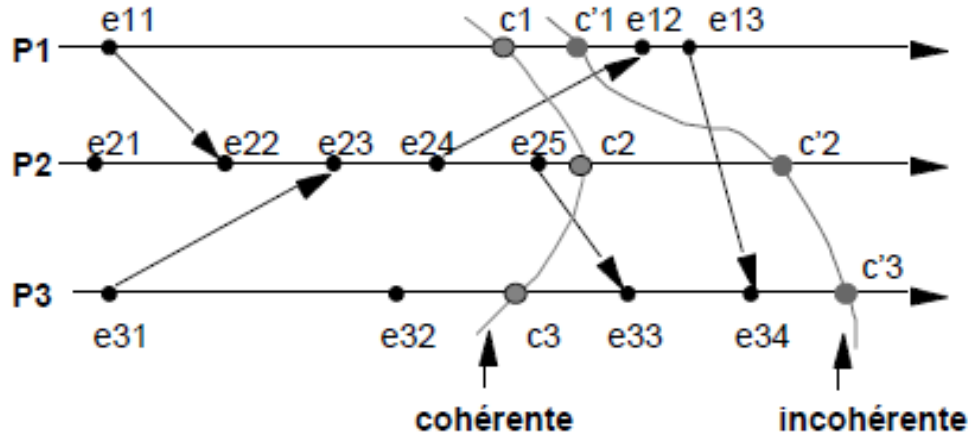
Etat global et coupures

- Calcul distribué = ensemble d'événements E
- Coupure = sous-ensemble fini C de E, tel que
  - a appartient à C et (b précède localement a) ➔ (b appartient à C)
- Coupure = "photographie" instantanée de l'état d'un système : ensemble d'événements (virtuels), un par processus ; permet de définir un passé et un futur (par rapport à la coupure)
- Relation d'ordre (partiel) sur les coupures, par inclusion des passés
- Etat associé à une coupure C = ensemble d'états locaux si (un par processus) tel que l'effet local de tout événement de C soit pris en compte dans s_i



Coupures cohérentes

- Cohérence = respect de la causalité (un message ne peut pas venir du futur)



- Coupure cohérente = coupure fermée par la relation de dépendance causale
  - a appartient à C et (b → a) alors (b appartient à C)
- Etat global cohérent = état associé à une coupure cohérente
- Exemple de coupure cohérente : le passé d'un événement
- Les coupures cohérentes permettent de définir un ordre de précédence entre états globaux cohérents

Calculs et observations distribués

- Observation d'un calcul réparti E ="linéarisation" de E (définition d'un ordre total,

  soit <<, sur les événements de E)
- Exemple : vue de la séquence des événements de E par un observateur interne ou externe au système
- Observation valide = compatible avec la relation de précédence causale (pourrait être observée par un observateur externe réel)
- e, e' appartient à E : e → e' => e << e'
- Exemple :
- e11 e21 e31 e22 e23 e32 e24 e25 e12 e33 e13 e34 valide
- e11 e21 e31 e22 e32 e23 e24 e25 e12 e33 e13 e34 valide
- e11 e21 e31 e22 e23 e32 e24 e25 e12 e33 e34 e13 invalide

Temps logique : dépendance causale

- Ordonnancement des événements
- Les dépendances causales définissent des ordres partiels pour des ensembles d'événements
- But d'une horloge logique
  - Créer un ordre total global sur tous les événements de tous les processus
- Horloge logique
  - Fonction H(e) : associe une date à chaque événement
  - Respect des dépendances causales
    - e → e' => H(e) < H(e')
    - H(e) < H(e') => ¬ (e' → e)
      - C'est-à-dire : soit e → e', soit e || e'

Horloge de Lamport

- Lamport, 1978
- Horloge de Lamport : horloge logique respectant les dépendances causales
- Une date (estampille) est associée à chaque évènement : couple (s, nb)
  - s : numéro du processus
  - nb : numéro d'événement
- Invariant sur les dates
  - Pour deux dates d'un même processus, les numéros de ces dates sont différentes
- $\forall d,d' : d.s = d'.s \Rightarrow d.nb \neq d'.nb$
- Il n'y pas deux événements locaux ayant le même numéro

Horloge de Lamport

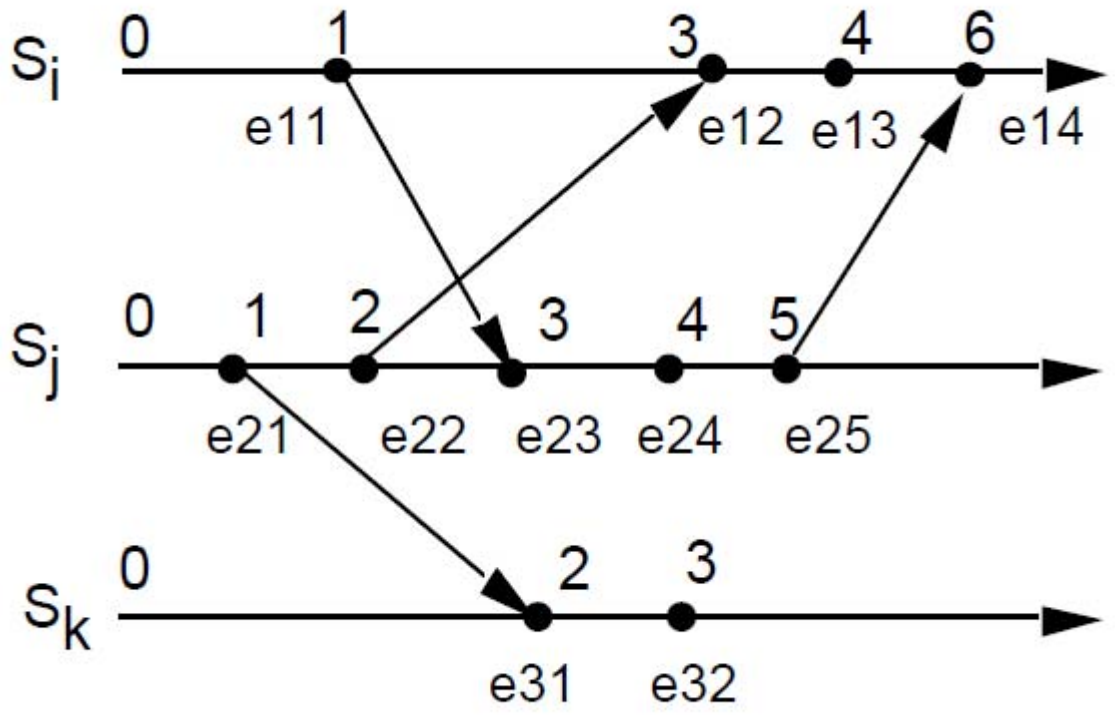
- Création du temps logique
- Localement, chaque processus Pi possède une horloge locale logique Hi, initialisée à 0
- Sert à dater les événements
- Pour chaque événement local de Pi
  - Hi = Hi + 1 : on incrémente l'horloge locale
  - L'événement est daté localement par Hi
- Émission d'un message par Pi
  - On incrémente Hi de 1 puis on envoie le message avec (i, Hi) comme estampille
- Réception d'un message *m* avec estampille (s, nb)
  - Hi = max(Hi, nb) +1 et marque l'événement de réception avec Hi
    - Hi est éventuellement recalée sur l'horloge de l'autre processus avant d'être incrémentée

Horloge de Lamport

- Création du temps logique
- Localement, chaque processus Pi possède une horloge locale logique Hi, initialisée à 0
  - Sert à dater les événements
- Pour chaque événement local de Pi
  - Hi = Hi + 1 : on incrémente l'horloge locale
  - L'événement est daté localement par Hi
- Émission d'un message par Pi
  - On incrémente Hi de 1 puis on envoie le message avec (i, Hi) comme estampille
- Réception d'un message *m* avec estampille (s, nb)
  - Hi = max(Hi, nb) +1 et marque l'événement de réception avec Hi
    - Hi est éventuellement recalée sur l'horloge de l'autre processus avant d'être incrémentée

Horloge de Lamport

- Exemple : chronogramme avec ajouts des estampilles



Horloge de Lamport

- Réalisation d'un ordre total (<<)
- Soit a sur Si, b sur Sj

  a << b ⇔ (H(a) < H(b)) ou (H(a) = H(b) et i<j)

  L'ordre sur les sites est arbitraire
  - Localement (si i = j), Hi donne l'ordre des événements du processus
  - Les 2 horloges de 2 processus différents permettent de déterminer l'ordonnancement des événements des 2 processus
    - Si égalité de la valeur de l'horloge, le numéro du processus est utilisé pour les ordonner
- On dispose donc d'un ordre total sur les événements, détectable à partir d'informations locales uniquement

  observation : e11 e21 e22 e31 e12 e23 e32 e13 e24 e25 e14
- Il ne doit pas y avoir de communications "cachées" (via d'autres canaux que ceux qui estampillent les messages)

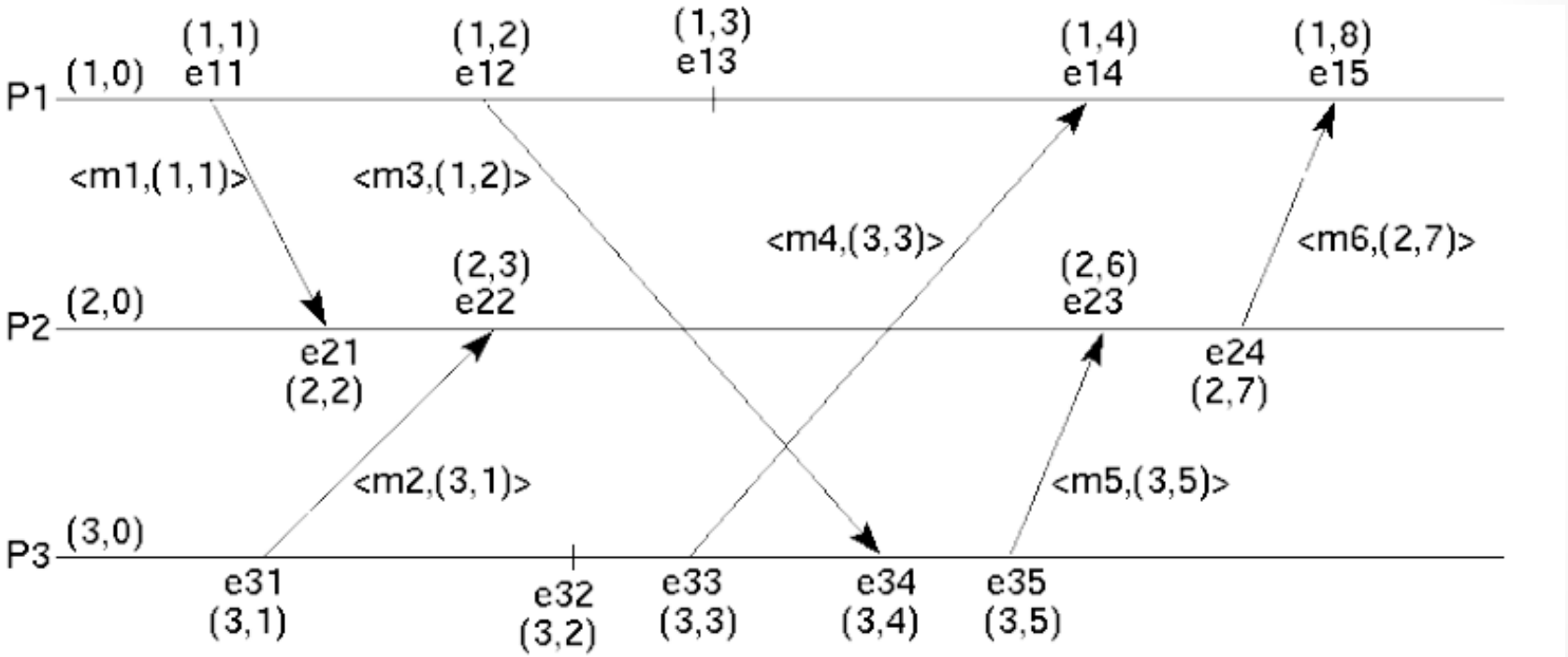
Utilisation des estampilles

- **Algorithmes utilisant une “file d’attente virtuelle” répartie**
  - exclusion mutuelle répartie
  - mise à jour de copies multiples
  - diffusion cohérente (ordre de réception uniforme)
- **Détermination de l’accès “le plus récent”**
  - gestion cohérente de caches multiples
  - fichiers répartis
  - mémoire virtuelle répartie
- **Génération de noms uniques**
  - désignation interne par noms universels
  - mécanismes d’authentification
- **Synchronisation des horloges physiques**
  - borne supérieure sur la dérive entre sites

Avantages

- Trois principaux avantages :
  - Première datation répartie intro duite
  - Économiques : la datation est réalisée par un seul nombre et non par un vecteur
  - Causalité des messages est respectée par remise à l'heure du récepteur

Exemple


P1
P2
P3
(1,0)
(1,1)
e11
(1,2)
e12
(1,3)
e13
(1,4)
e14
(1,8)
e15
<m1,(1,1)>
<m3,(1,2)>
<m4,(3,3)>
<m6,(2,7)>
(2,0)
(2,3)
e22
(2,6)
e23
e21
(2,2)
e24
(2,7)
<m2,(3,1)>
<m5,(3,5)>
(3,0)
e31
(3,1)
e32
(3,2)
e33
(3,3)
e34
(3,4)
e35
(3,5)

Limitations de l'ordonnancement par estampilles

- L'ordonnancement global obtenu est arbitraire et n'est pas forcément celui obtenu en pratique. => Les estampilles "effacent" artificiellement la notion de dépendance causale.
  - H(e32) = 2 et H(e22) = 3 pourtant e22 est exécuté en pratique avant e32
- E1-> E2 implique H(E1) < H(E2) est une condition faible, car on ne peut rien affirmer si H(E1) < H(E2)
  - soit les évènements E1 et E2 sont concurrents
  - soit les évènements E1 et E2 sont ordonnés
- Ordre total global obtenu pour l'exemple
  - e11 << e31 << e12<< e21 << e32 << e13 << e22 << e33 << e14 << e34 << e35 << e23 << e24 << e15
  - D'autres sont valides ...

Limitations de l'ordonnancement par estampilles

- Les estampilles ne sont pas "denses" => Horloge de Lamport respecte les dépendances causales mais avec *e* et *e'* tel que H(e) < H(e') on ne peut rien dire sur la dépendance entre *e* et *e'*
  - Dépendance causale directe ou transitive entre *e* et *e'*?
  - Exemple : H(e32) = 2 et H(e13) = 3 mais on a pas e32→ e13

soient E1 et E2 tels que H(E1) < H(E2), on ne peut pas savoir, s'il existe E3 tel que E1-> E3 et/ou E3-> E2

Horloges vectorielles (de Mattern)

- Les estampilles ne conservent pas la trace des chemins de causalité : réciproque de la dépendence causale
- Non constantes = pas de lien avec la précédence
- Nécessité de conserver l'état de chaque estampille pour déterminer localement la précédence entre deux événements
- Vecteur d'horloges
- Horloge de Mattern & Fidge, 1989-91
  - Horloge qui assure la réciproque de la dépendance causale
    - $H(e) < H(e') \Rightarrow e \rightarrow e'$
  - Permet également de savoir si 2 événements sont parallèles (non dépendants causalement)
  - Ne définit par contre pas un ordre total global

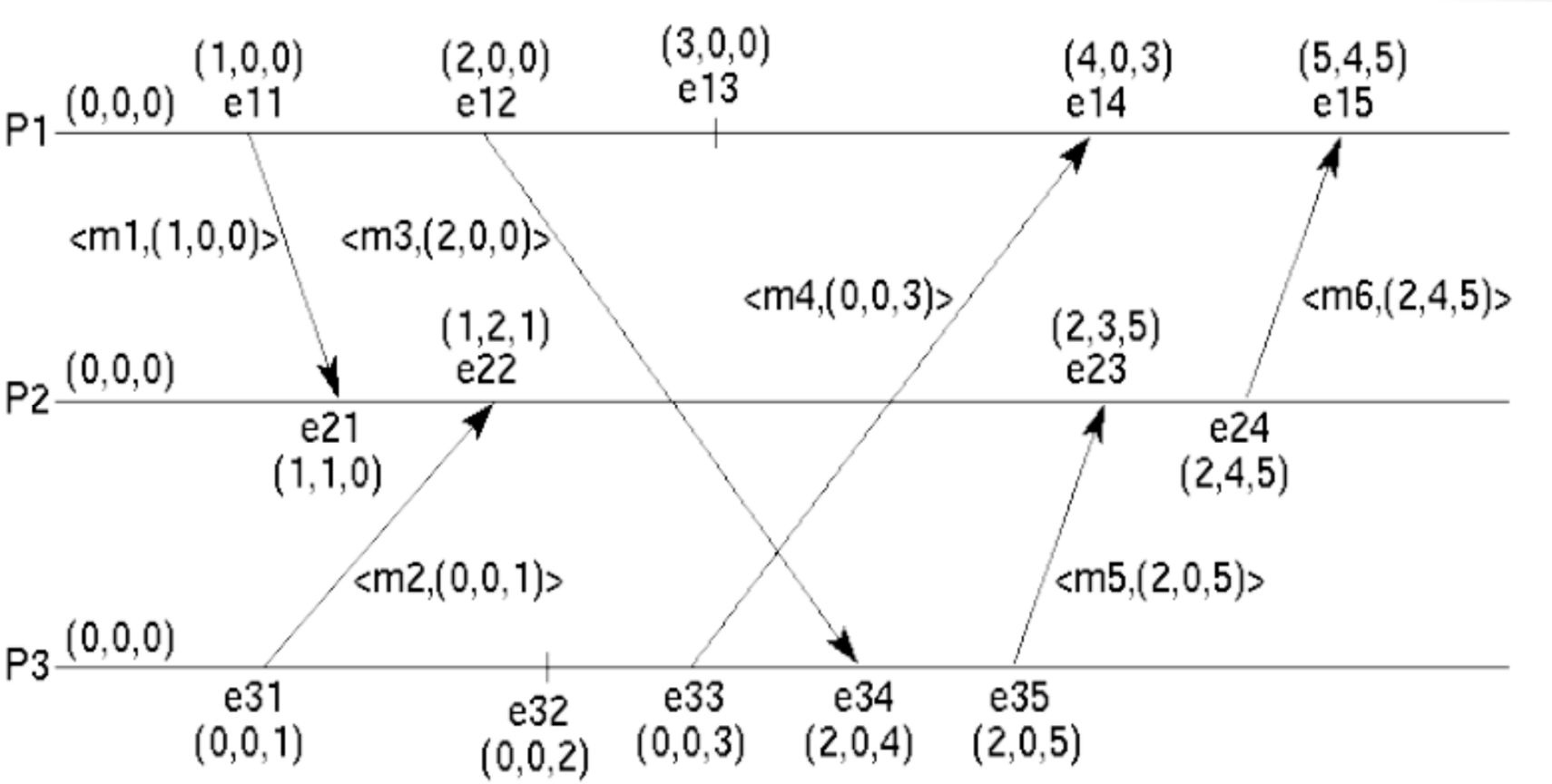
Principe des horloges vectorielles

- Principe
  - Utilisation d'estampilles sur les messages
    - Vecteur V de taille égale au nombre de processus
  - Localement, chaque processus *Pi* a un vecteur *Vi*
  - Pour chaque processus *Pi*, chaque case *Vi[j]* du vecteur contiendra des valeurs de l'horloge du processus *Pj*

Horloges vectorielles (de Mattern)

- On associe un vecteur Vi à chaque processus Pi (ou site Si), i = 1, ... , n
- Initialement, Vi = (0, ... , 0)
- A chaque événement local à Pi,
  - Vi[i] := Vi[i] + 1
- Chaque message m porte une estampille Vm
  - (Vm = Vi de l'émetteur)
- A la réception de (m, Vm) par un processus Pi :
- Vi[j] := max(Vi[j], Vm[j]) pour j = 1, ... , n, j <> i
  - Mémorise le nombre d'événements sur Pj qui sont sur Pj dépendants causalement par rapport à l'émission du message
  - La réception du message est donc aussi dépendante causalement de ces événements sur Pj
- Vi[i] := Vi[i] + 1

Exemple : chronogramme d'application horloge de Mattern



Caractéristiques des HV

- Relation d'ordre partiel sur les dates
  - $V \leq V'$ défini par $\forall i : V[i] \leq V'[i]$
  - $V < V'$ défini par $V \leq V'$ et $\exists j$ tel que $V [j] < V' [j]$
  - $V \,||\, V'$ défini par $\neg(V < V') \wedge \neg(V' < V)$
- Dépendance et indépendance causales
  - Horloge de Mattern assure les propriétés suivantes, avec *e* et *e'* deux événements et *V(e)* et *V(e')* leurs datations
    - $V(e) < V(e') \Rightarrow e \rightarrow e'$
      - Si deux dates sont ordonnées, on a forcément dépendance causale entre les événements datés
    - $V(e) \,||\, V(e') \Rightarrow e \,||\, e'$
      - Si il n'y a aucun ordre entre les 2 dates, les 2 événements sont indépendants causalement

Caractéristiques des HV

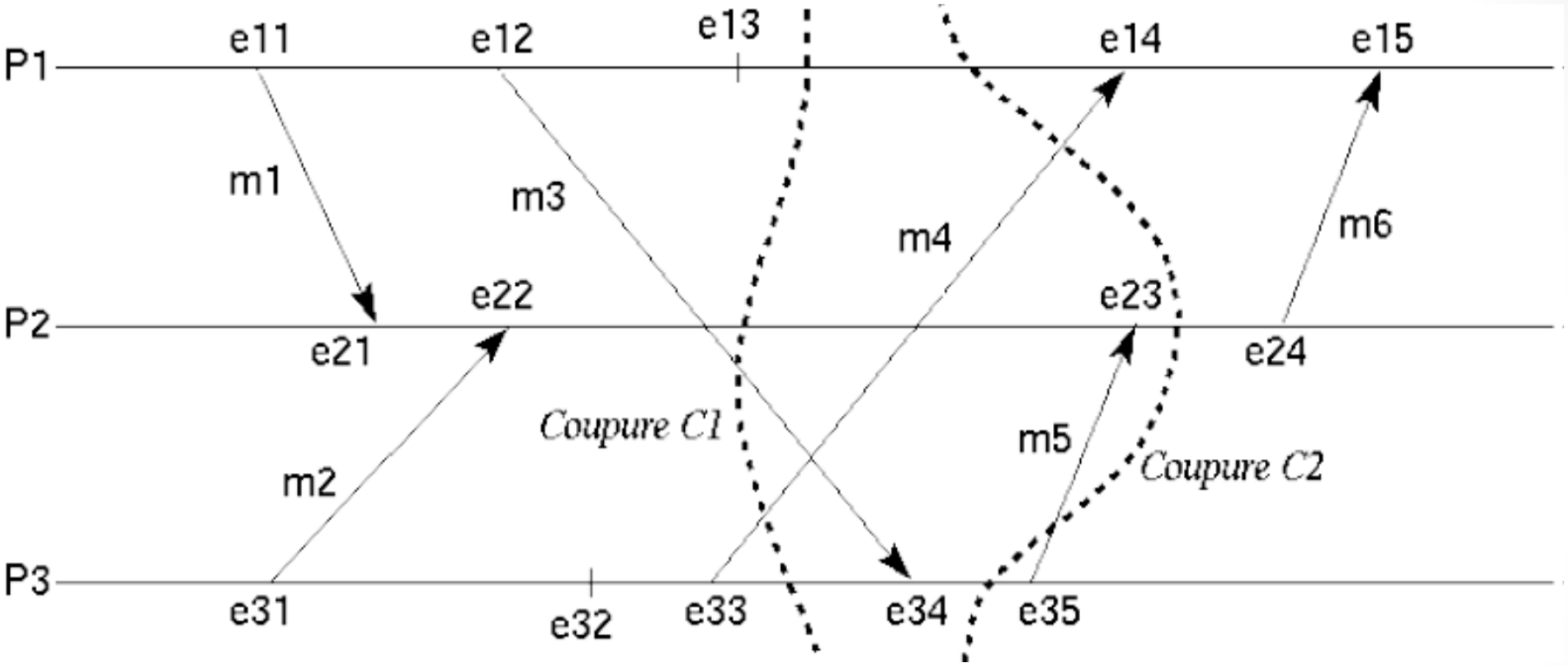
- V(e13) = (3,0,0), V(e14) = (4,0,3), V(e15) = (5,4,5)
  - V(e13) < V(e14) donc e13 → e14
  - V(e14) < V(e15) donc e13 → e15
- V(e35) = (2,0,5) et V(e23) = (2,3,5)
  - V(e35) < v(e23) donc e35 → e23
- L'horloge de Mattern respecte les dépendances causales des événements
  - Horloge de Lamport respecte cela également
- V(e32) = (0,0,2) et V(e13) = (3, 0, 0)
  - On a ni V(e32) < V(e13) ni V(e13) < V(e32) donc e32 || e13
  - L'horloge de Mattern respecte les indépendances causales
    - L'horloge de Lamport impose un ordre arbitraire entre les événements indépendants causalement.
- Limite de l'horloge de Mattern
- Ne permet pas de définir un ordre global total
- En cas de nombreux processus, la taille du vecteur peut-être importante et donc des données à transmettre relativement importante

HV et Coupures cohérentes

- Horloge de Mattern permet de dater la coupure
  - Soit N processus, C la coupure, ei l'événement le plus récent pour le processus Pi, V(ei) la datation de ei et V(C) la datation de la coupure
    - V(C) = max (V(e1), ... , V(eN)) :

      $\forall i : V(C)[i] = \max (V(e1)[i] , ... , V(eN)[i])$
    - Pour chaque valeur du vecteur, on prend le maximum des valeurs de tous les vecteurs des N événements pour le même indice
- Permet également de déterminer si la coupure est cohérente
  - Cohérent si V(C) = (V(e1)[1], ... , V(ei)[i], ... , V(eN) [N])
  - Pour un processus Pi, si l'événement ei est le plus récent c'est lui qui a la date la plus récente pour C : sinon un événement ej d'un processus Pj ($i \neq j$) s'est déroulé après un événement ei' de Pi avec ei' plus récent que ei
    - ei $\rightarrow$ ei' et ei' $\rightarrow$ ej avec ei $\in$ C, ej $\in$ C et ei' $\notin$ C

Coupure cohérente: Exemple



- Coupure C1 : cohérente
- Coupure C2 : non cohérente car e35 $\rightarrow$ e23 mais e35 $\notin$ C2
  - La réception de *m5* est dans la coupure mais pas son émission
  - *m5* vient du futur par rapport à la coupure

Datation des coupures de l'exemple

- Coupure C1 : état = (e13, e22, e33)
  - V(e13) = (3,0,0), V(e22) = (1,2,1), V(e33) = (0,0,3)
  - V(C) = (max(**3**,1,0), max(0,**2**,0), max(0,1,**3**)) = (3,2,3)
  - Coupure cohérente car V(C)[1] = V(e13)[1], V(C)[2] = V(e22)[2], V(C)[3] = V(e33)[3]
- Coupure C2 : état = (e13, e23, e34)
  - V(e13) = (3,0,0), V(e23) = (2,3,5), V(e34) = (2,0,4)
  - V(C) = (max(**3**,2,2), max(0,**3**,0), max (0,**5**,4))
  - Non cohérent car V(C)[3] $\neq$ V(e34)[3]
    - D'après la date de e23, e23 doit se dérouler après 5 événements de P3 or e34 n'est que le quatrième événement de P3
    - Un événement de P3 dont e23 dépend causalement n'est donc pas dans la coupure (il s'agit de e35 se déroulant dans le futur)

Utilisation des horloges vectorielles

- Datation
- Diffusion fiable avec ordonnancement causal
- Simulation répartie
- Calcul d'état global
- Mise au point

Séquencement des messages selon l ’ordre causal

- Définition

  Soient les deux événements E1 = émission (m1)

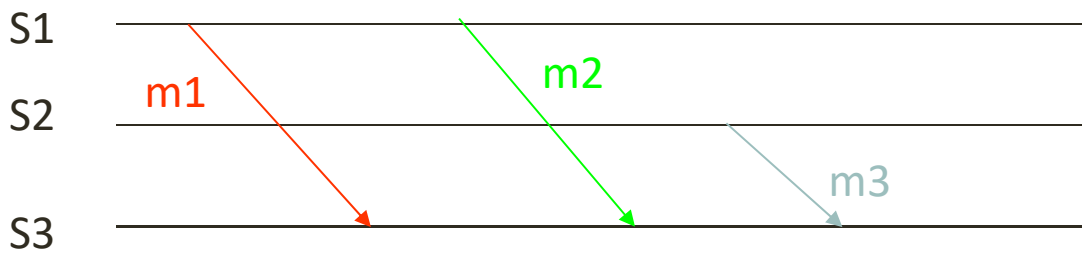
  et E2 = émission (m2),

  L ’ordre causal est respecté si la propriété suivante est respectée :

  si (E1→E2) et m1 et m2 ont même destinataire alors

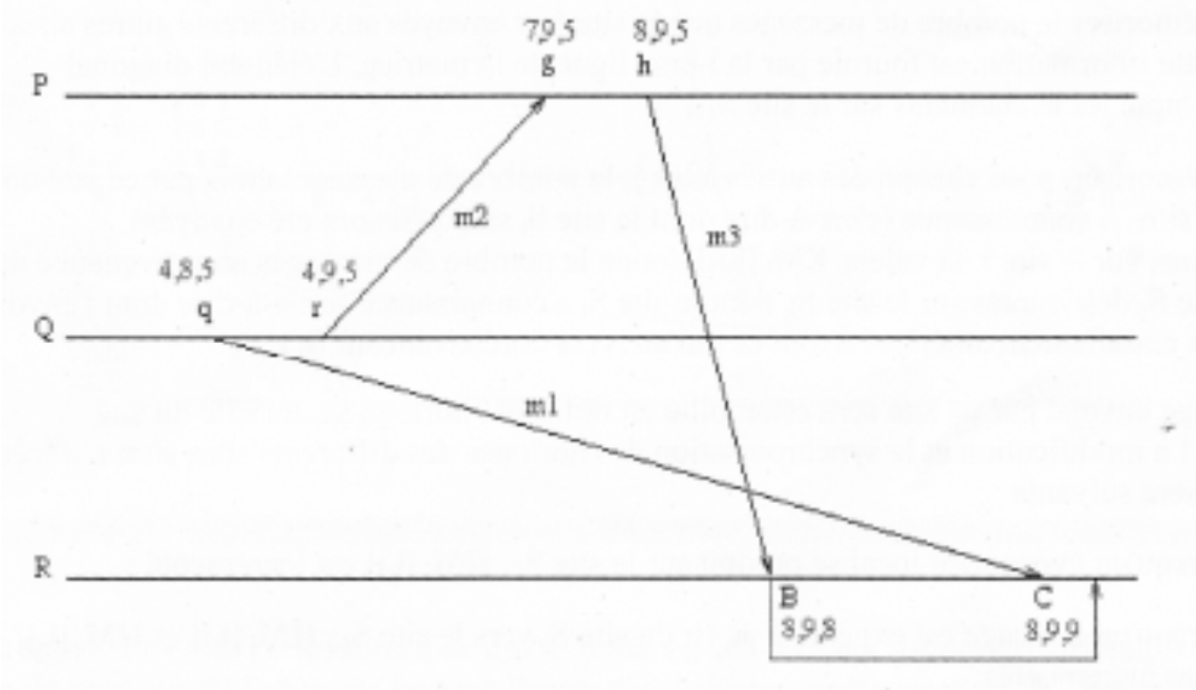
  réception(m1) →réception(m2)

- Exemple



Inconvénient des estampilles vectorielles:

- ne permettent pas de garantir une délivrance causale des messages.



- Dans l'exemple ci-dessus le message m3 est reçu trop tôt: pour respecter l'ordre causal de la délivrance il ne doit être délivré qu'après le message m1.
- Les estampilles permettent de détecter (a posteriori !)la violation de l'ordre causal dans la réception des messages

Horloges et estampilles matricielles

Définition et principe des estampilles matricielles

- Inconvénients des estampilles vectorielles :

  ⇒ Ne permettent pas de corriger la défaillance vis-à vis de la délivrance causale des messages

- Principe des estampilles matricielles :
  - Dans un système de n sites, les horloges d'un site i et les estampilles des événements (et des messages) sont des matrices carrées d'ordre n.
  - HMi désigne l'horloge matricielle du site Si,
  - EMm désigne l'estampille matricielle du message m,

Fonctionnement des horloges matricielles

Sur le site Si, la matrice HMi va :

- mémoriser le nombre de messages que le site Si a envoyé aux différents autres sites : cette information est fournie par la $i^{ème}$ ligne de la matrice.
- L'élément diagonal représente la connaissance qu'a Si du nombre d'événements locaux qui se sont déjà produits sur les différents sites Sj : correspond à l'estampille vectorielle.
- mémoriser, pour chacun des autres sites j, le nombre de messages émis par ce site dont le site i a connaissance (i.e dont le site Si sait qu'ils ont été envoyés).
- Ainsi sur le site i, la valeur EMi [j,k] donne le nombre de messages en provenance du site Sj délivrables sur le site Sk dont le site Si a connaissance (i.e dont l'envoi est causalement antérieur à tout ce qui arrivera dorénavant sur Si).

La modification synchronisation des horloges des différents sites

- lorsqu'un événement local se produit sur le site Si : HMi [i,i] est incrémenté;
- lorsqu'un message est expédié à partir du site Si vers le site Sj : HMi [i,i] et HMi [i,j] sont incrémentés ;
- lorsqu'un message m en provenance du site Sj est reçu sur le site Si , il faut s'assurer que tous les messages envoyés antérieurement au site Si y sont effectivement arrivés. Cela suppose donc que Si ait reçu :
  - d'une part tous les messages précédents de Sj
  - d'autre part tous ceux envoyés plus tôt causalement depuis d'autres sites.
- Cela correspond aux conditions suivantes (à vérifier dans l'ordre):
  1. HMm [j,i] = HMi [j,i] + 1 (ordre FIFO sur le canal (j,i))
  2. pour tout k ≠ i et j, HMm [k,i] ≤ HMi [k,i] (tous les messages en provenance des sites différents de Sj ont été reçus).

La modification synchronisation des horloges des différents sites

- Si toutes ces conditions sont vérifiées, le message est délivrable et l'horloge du site Si est mise à jour :
  1. HMi [i,i] ++ (incrémentation);
  2. HMi [j,i] ++ (incrémentation);
  3. pour le reste de la matrice : HMi [k,l] = max (HMi [k,l], EMm [k,l])
- *Si les conditions ne sont pas toutes vérifiées,* la délivrance du message est différée jusqu'à ce qu'elles le deviennent et l'horloge n'est pas mise à jour.
- La délivrance d'un message pourra ainsi provoquer celle des messages arrivés prématurément.

Exemple d'horloge matricielle