Etat global dans un système distribué

Master 2 Informatique - UFR S.A.T

Pr. Ousmane THIARE

ousmane.thiare@ugb.edu.sn
http://www.ousmanethiare.com/

10 mai 2024

Chapitre 7 : Déterminer un état global dans un système distribué

- de détecter des propriétés caractérisant l'exécution des programmes (vérification d'assertions, terminaison, inter-blocage, ...) ;

Or

$\Longrightarrow$ impossibilité d'effectuer une observation "simultanée" de l'état des différents processus et canaux de communication. L'exemple de Chandy et Lamport est parlant :

- de détecter des propriétés caractérisant l'exécution des programmes (vérification d'assertions, terminaison, inter-blocage, ...) ;
- de calculer de performances ;

Or

$\Longrightarrow$ impossibilité d'effectuer une observation "simultanée" de l'état des différents processus et canaux de communication. L'exemple de Chandy et Lamport est parlant :

Introduction
Problématique

- de détecter des propriétés caractérisant l'exécution des programmes (vérification d'assertions, terminaison, inter-blocage, ...) ;
- de calculer de performances ;
- de capturer un état de reprise en cas de panne ;

Or

$\Longrightarrow$ impossibilité d'effectuer une observation "simultanée" de l'état des différents processus et canaux de communication. L'exemple de Chandy et Lamport est parlant :

Introduction
Problématique

- de détecter des propriétés caractérisant l'exécution des programmes (vérification d'assertions, terminaison, inter-blocage, ...) ;
- de calculer de performances ;
- de capturer un état de reprise en cas de panne ;
- ...

Or

$\Longrightarrow$ impossibilité d'effectuer une observation "simultanée" de l'état des différents processus et canaux de communication. L'exemple de Chandy et Lamport est parlant :

Introduction
Problématique

- de détecter des propriétés caractérisant l'exécution des programmes (vérification d'assertions, terminaison, inter-blocage, ...) ;
- de calculer de performances ;
- de capturer un état de reprise en cas de panne ;
- ...

Or

- pas d'horloge globale

$\Longrightarrow$ impossibilité d'effectuer une observation "simultanée" de l'état des différents processus et canaux de communication. L'exemple de Chandy et Lamport est parlant :

Introduction
Problématique

- de détecter des propriétés caractérisant l'exécution des programmes (vérification d'assertions, terminaison, inter-blocage, ...) ;
- de calculer de performances ;
- de capturer un état de reprise en cas de panne ;
- ...

Or

- pas d'horloge globale
- temps de transfert des messages (dans le cas général) non borne

$\Longrightarrow$ impossibilité d'effectuer une observation "simultanée" de l'état des différents processus et canaux de communication. L'exemple de Chandy et Lamport est parlant :

Introduction
Problématique

Un groupe de photographe observe une scène dynamique, par exemple un vol d'oiseaux migrateurs. La scène est si vaste qu'elle ne peut être prise par un seul photographe. Les différents photographes doivent donc se charger de prendre chacun une partie de la scène tout en sachant :

- que les photographies ne peuvent être prise au même moment

Il faut aussi que la scène reconstituée ait "un sens" : reste à définir ce qu'est "avoir un sens" et de déterminer la manière de faire ces prises de vues.

Introduction
Problématique

Un groupe de photographe observe une scène dynamique, par exemple un vol d'oiseaux migrateurs. La scène est si vaste qu'elle ne peut être prise par un seul photographe. Les différents photographes doivent donc se charger de prendre chacun une partie de la scène tout en sachant :

- que les photographies ne peuvent être prise au même moment
- que ces opérations ne doivent pas perturber le phénomène à observer (par exemple, il n'est pas raisonnable de demander aux oiseaux d'arrêter leur vol pendant les prises de vue)

Il faut aussi que la scène reconstituée ait "un sens" : reste à définir ce qu'est "avoir un sens" et de déterminer la manière de faire ces prises de vues.

Chaque processus P_i et chaque canal Ci, j (entre P_i et P_j) possède à tout instant un état local.
Soit $el_i(k)$ l'état local d'un processus P_i et $ec_{i,j}(k')$ celui d'un canal $C_{i,j}$ (ensemble des messages en transit sur $C_{i,j}$) à l'instant t.
Trois sortes d'événements peuvent survenir :

- un événement interne à un processus : provoque la transition de P_i de $el_i(k)$ à $el_i(k+1)$

Chaque processus P_i et chaque canal Ci, j (entre P_i et P_j) possède à tout instant un état local.
Soit $el_i(k)$ l'état local d'un processus P_i et $ec_{i,j}(k')$ celui d'un canal $C_{i,j}$ (ensemble des messages en transit sur $C_{i,j}$) à l'instant t.
Trois sortes d'événements peuvent survenir :

- un événement interne à un processus : provoque la transition de P_i de $el_i(k)$ à $el_i(k+1)$
- correspondant à l'émission de m par P_i sur $C_{i,j}$ et noté $emission_{i_r}(m)$: provoque la transition de P_i de $el_i(k)$ à $el_i(k+1)$ et l'affectation $ec_{ij}(k'+1) = ec_{ij}(k') \cup \{m\}$

Chaque processus P_i et chaque canal Ci, j (entre P_i et P_j) possède à tout instant un état local.
Soit $el_i(k)$ l'état local d'un processus P_i et $ec_{i,j}(k')$ celui d'un canal $C_{i,j}$ (ensemble des messages en transit sur $C_{i,j}$) à l'instant t.
Trois sortes d'événements peuvent survenir :

- un événement interne à un processus : provoque la transition de P_i de $el_i(k)$ à $el_i(k+1)$
- correspondant à l'émission de m par P_i sur $C_{i,j}$ et noté $emission_{i_r}(m)$: provoque la transition de P_i de $el_i(k)$ à $el_i(k+1)$ et l'affectation $ec_{ij}(k'+1) = ec_{ij}(k') \cup \{m\}$
- correspondant à la réception de m par P_i sur $C_{i,j}$ et noté $reception_{i_r}(m)$: provoque la transition de P_i de $el_i(k)$ à $el_i(k+1)$ et l'affectation $ec_{ij}(k'+1) = ec_{ij}(k') \backslash \{m\}$

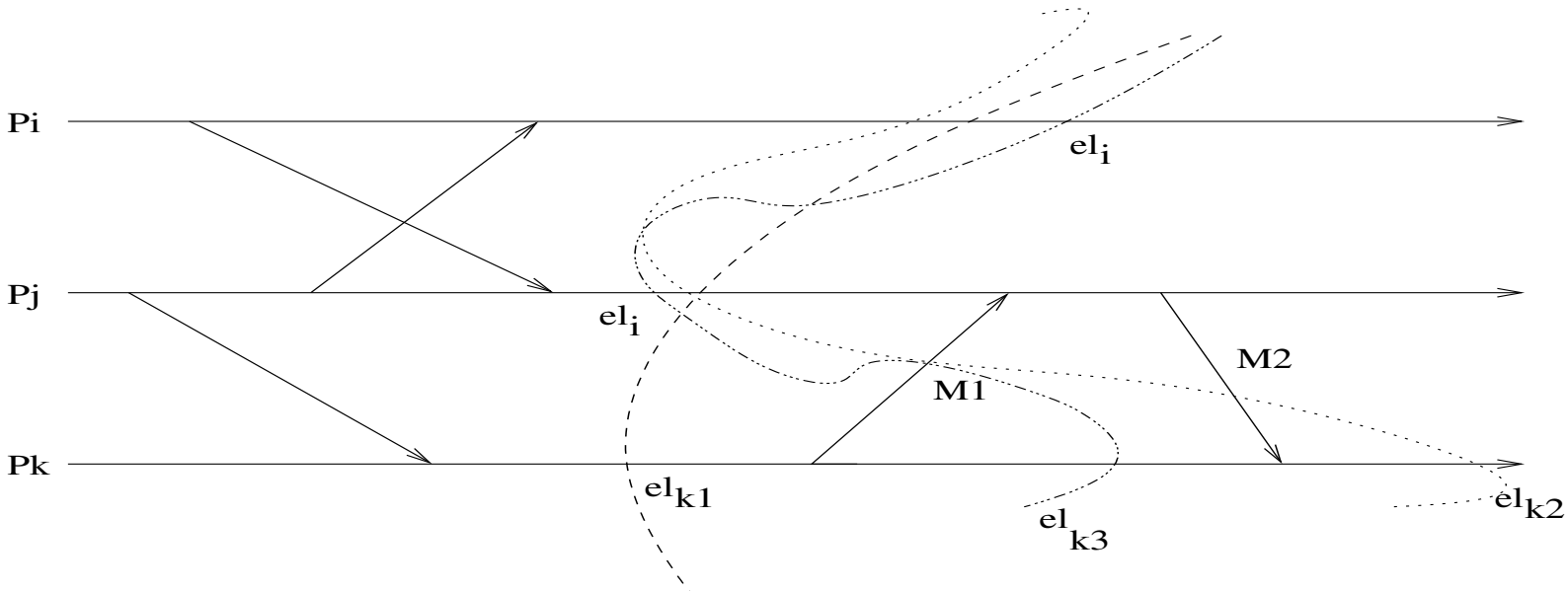
Notons eli l'état courant du processus P_i et ec_{ij} celui du canal $C_{i,j}$.
L'état global d'un système est alors : $S = \{\bigcup_i el_i, \bigcup_{i,j} ecij\}$
Définition Un état global d'un système est **cohérent** si :

- $\forall\, i,\ el_i$ est un état local de P_i

Notons eli l'état courant du processus P_i et ec_{ij} celui du canal $C_{i,j}$.
L'état global d'un système est alors : $S = \{\bigcup_i el_i, \bigcup_{i,j} ecij\}$
Définition Un état global d'un système est **cohérent** si :

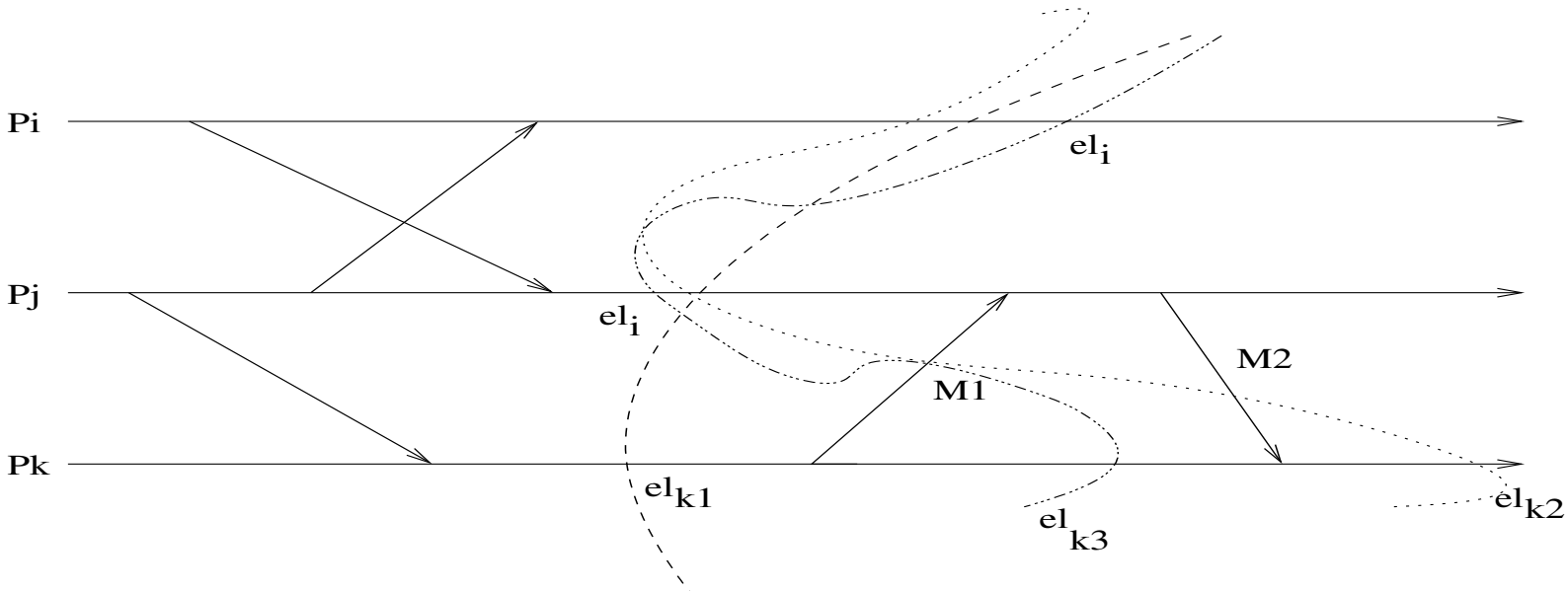
- $\forall\, i,\ el_i$ est un état local de P_i
- les deux conditions suivantes sont respectées :
  - **C1 :** si l'événement $emission_{i_k}(m)$ est capté dans el_i alors l'événement $reception_{j_r}(m)$ est soit capté dans el_j, soit $m \in ec_{ij}$
  - **C2 :** si l'événement $emission_{i_k}(m)$ n'est pas capté dans el_i alors l'événement $reception_{j_r}(m)$ n'est pas capté dans el_j

Ainsi :



- $S_1 = \{e_i, e_j, e_{k_1}\}$ forme un ensemble d'états locaux cohérents (correspond à une "coupe cohérente")

Ainsi :



- $S_1 = \{e_i, e_j, e_{k_1}\}$ forme un ensemble d'états locaux cohérents (correspond à une "coupe cohérente")
- $S_2 = \{e_i, e_j, e_{k_2}\}$ ne forme pas un état global cohérent (C1 : violée pour M1, C2 violée pour M2)

- $S_3 = \{\{e_i, e_j, e_{k_3}\}, \{c_{k,j} = \{M1\}\}\}$ forme un état global cohérent.

Solution pour des canaux FIFO
Rappel

Dans un canal fifo C_{ij} :
$m_1 < m_2 \Leftrightarrow emission(m_1) < emission(m_2) \Leftrightarrow reception(m_1) < reception(m_2)$
Soit el_i l'état de P_i à un instant I donné : seuls les messages émis avant cet instant I sont captés par el_i.
Pour assurer la cohérence entre el_i et un el_j de P_j seules les réceptions des messages émis avant I doivent être captées par el_j (évident).
Pour mettre en oeuvre cette contrainte, P_i émet, en I un message de contrôle noté Mk sur le canal C_{ij}. Le processus P_j doit alors enregistrer son état à la réception de ce message.
La "démonstration" qu'alors C1 et C2 sont vérifiées est évidente.

L'algorithme de Chandy et Lamport, 1985, est alors : (Il peut exister un processus Pc chargé de récupérer l'état global du système : **facultatif**) Chaque processus P_i

- dispose d'une variable el_i qui lui permet de mémoriser son état local

Solution pour des canaux FIFO
Rappel

L'algorithme de Chandy et Lamport, 1985, est alors : (Il peut exister un processus Pc chargé de récupérer l'état global du système : **facultatif**) Chaque processus P_i

- dispose d'une variable el_i qui lui permet de mémoriser son état local
- dispose de deux tableaux :

L'algorithme de Chandy et Lamport, 1985, est alors : (Il peut exister un processus Pc chargé de récupérer l'état global du système : **facultatif**) Chaque processus P_i

- dispose d'une variable el_i qui lui permet de mémoriser son état local
- dispose de deux tableaux :
  - $R_i[1..N]$ qui représente C_{ji} (ensemble des messages reçus après la mémorisation de l'état local et avant la réception de Mk de P_j), initialisé à $\emptyset$;

Solution pour des canaux FIFO
Rappel

L'algorithme de Chandy et Lamport, 1985, est alors : (Il peut exister un processus Pc chargé de récupérer l'état global du système : **facultatif**) Chaque processus P_i

- dispose d'une variable el_i qui lui permet de mémoriser son état local
- dispose de deux tableaux :
  - $R_i[1..N]$ qui représente C_{ji} (ensemble des messages reçus après la mémorisation de l'état local et avant la réception de Mk de P_j), initialisé à $\emptyset$;
  - $EL_i[1..N]$: qui permet du dire si P_i a reçu Mk de P_j , initialisé à Faux

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état
    - $EtatMemorise_i$ =Vrai ;

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état
    - $EtatMemorise_i$ =Vrai ;
    - Pour tout $j \neq i\{R_i[j] = \emptyset$; *mettre Mk vers* $P_j\}$

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état
    - $EtatMemorise_i$ =Vrai ;
    - Pour tout $j \neq i\{R_i[j] = \emptyset$; *mettre Mk vers* $P_j\}$
  - Un processus P_i qui reçoit un Mk de P_j :

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état
    - $EtatMemorise_i$ =Vrai ;
    - Pour tout $j \neq i\{R_i[j] = \emptyset$; *mettre Mk vers* $P_j\}$
  - Un processus P_i qui reçoit un Mk de P_j :
    - Si $EtatMemorise_i$ = Faux, il exécute Prise_en_compte()

L'algorithme de Chandy et Lamport, 1985, est alors :

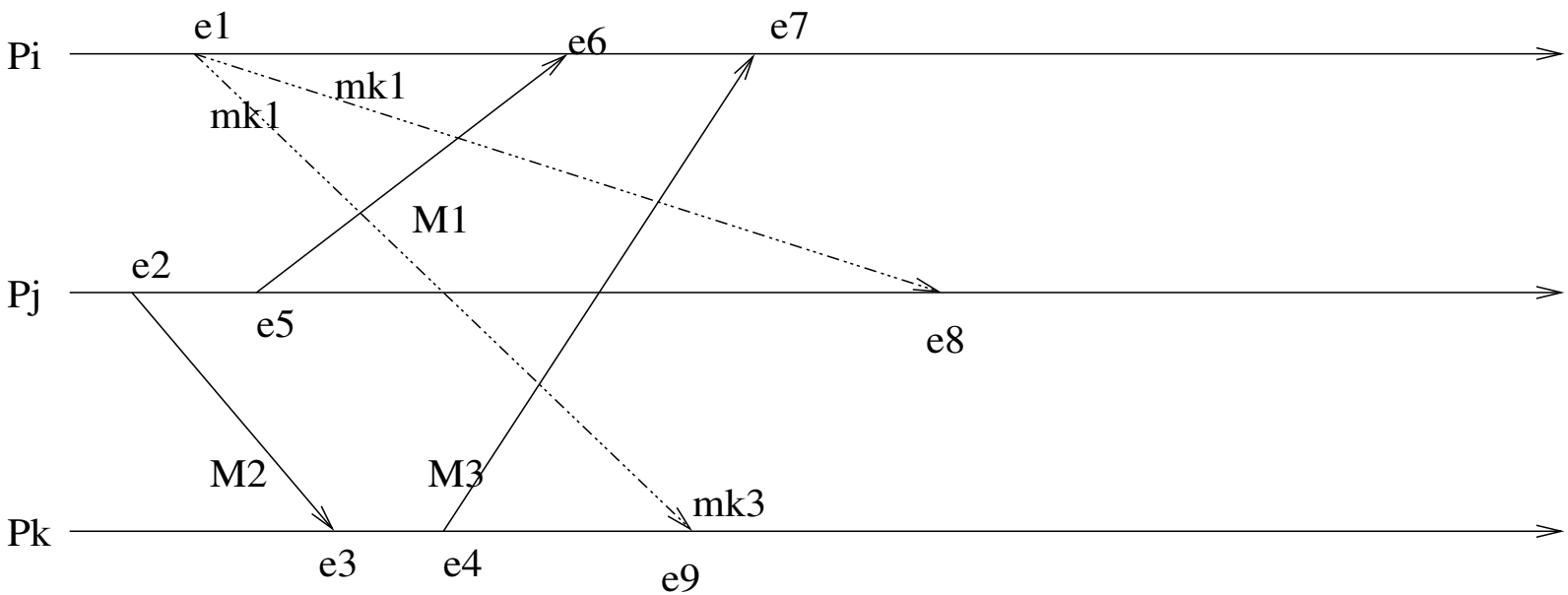
- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état
    - $EtatMemorise_i$ =Vrai ;
    - Pour tout $j \neq i\{R_i[j] = \emptyset$; *mettre Mk vers* $P_j\}$
  - Un processus P_i qui reçoit un Mk de P_j :
    - Si $EtatMemorise_i$ = Faux, il exécute Prise_en_compte()
    - Dans tous les cas, il marque $EL_i[j] = Vrai$.

L'algorithme de Chandy et Lamport, 1985, est alors :

- Un processus P_i qui reçoit un message m (autre que Mk) de P_j le stocke dans $R_i[j]$ si $EtatMemorise_i$ = Vrai et $EL_i[j] = k$
- Un processus P_i qui décide de mémoriser son état local exécute :
  - Prise_en_compte() ;
    - el_i = état local du processus P_i ; //P_i sauvegarde son état
    - $EtatMemorise_i$ =Vrai ;
    - Pour tout $j \neq i\{R_i[j] = \emptyset$; *mettre Mk vers* $P_j\}$
  - Un processus P_i qui reçoit un Mk de P_j :
    - Si $EtatMemorise_i$ = Faux, il exécute Prise_en_compte()
    - Dans tous les cas, il marque $EL_i[j] = Vrai$.
    - Si $(\forall j \neq i, EL_i[j] = Vrai)$, il envoie l'état $\{el_i, R_i\}$ au processus Pc.

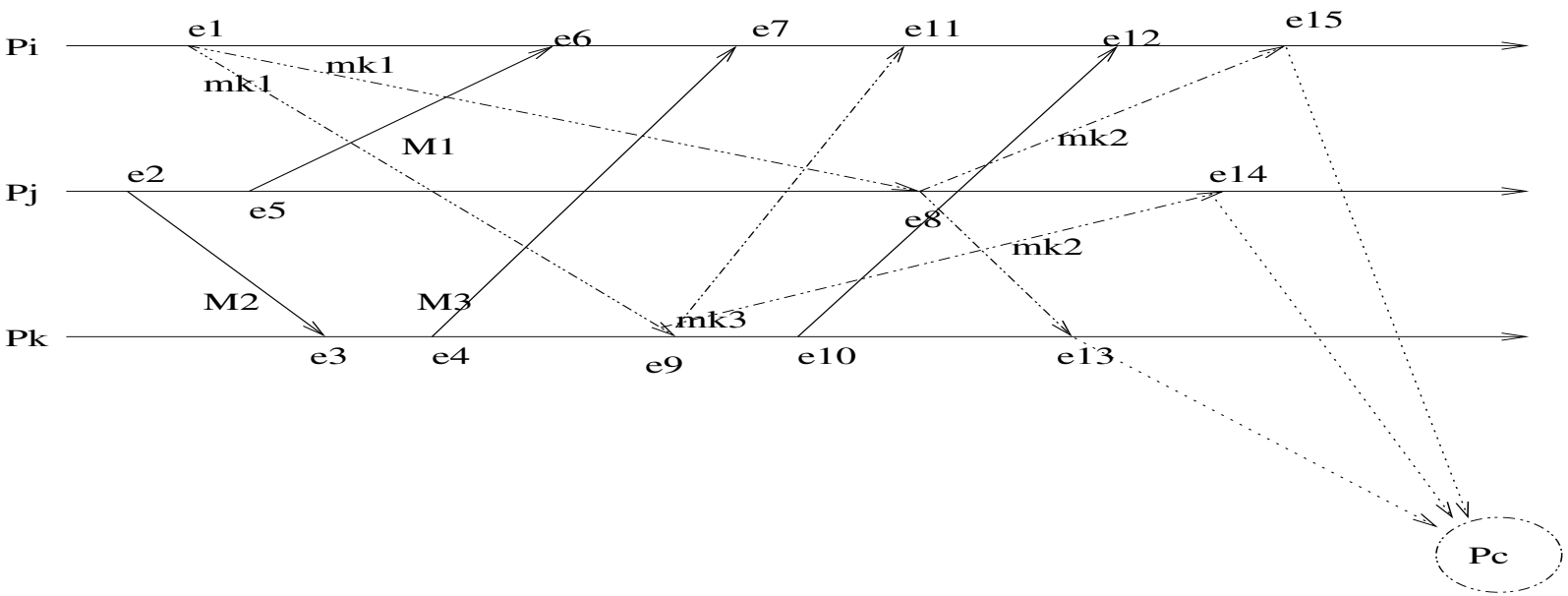
Exemple

On suppose qu'il n'y a pas d'événements locaux
Initialement, P_1 lance la captation d'un état global (cela pourrait être fait aussi par Pc : il suffit qu'il diffuse Mk)



Exemple

Ce qui nous donne



Exemple

E	1	2	3	4	5	6	7	8	9
Etat	el_1								
R[2]/EL[2]						M1, Faux	M1, Faux		
R[3]/EL[3]							M3, Faux		
Act	D(mk1)					R(M1)	R(M3)		
Etat	∅							el_2	
R[1]/EL[1]								∅, vrai	
R[3]/EL[3]								∅, Faux	
Act		E(M2)			E(M1)			D(mk2)	
Etat	∅								el_3
R[1]/EL[1]									∅, vrai
R[2]/EL[2]									∅, Faux
Act			R(M2)	E(M3)					D(mk3)

Exemple

E	10	11		13	14	15
Etat						
R[2]/EL[2]		M1, Faux	M1, Faux			M1, Vrai
R[3]/EL[3]		M3, Vrai	M3, Vrai			M3, Vrai
Act			R(M4)			E(el1,{ ,{M1},{M3}})–>Pc
Etat						
R[1]/EL[1]						
R[3]/EL[3]						
Act					E(el2,{ }) –>Pc	
Etat						
R[1]/EL[1]				, Vrai		
R[2]/EL[2]				, Vrai		
Act	E(M4)			E(el3,{ }) –>Pc		

Exemple

- Pourquoi $E(el_1\{\emptyset, \{M1\}, \{M3\}\}) \rightarrow Pc$?

Exemple

- Pourquoi $E(el_1\{\emptyset, \{M1\}, \{M3\}\}) \rightarrow Pc$?
  - à cause de C1 : les émissions M1 et M3 sont captées par el_2 et el_3, comme les réceptions ne sont pas captées par el_1, elles doivent apparaître dans les canaux

Exemple

- Pourquoi $E(el_1\{\emptyset, \{M1\}, \{M3\}\}) \rightarrow Pc$?
  - à cause de C1 : les émissions M1 et M3 sont captées par el_2 et el_3, comme les réceptions ne sont pas captées par el_1, elles doivent apparaître dans les canaux
  - à cause de C2 : l'émission de M4 n'est pas captée dans el_3, la réception ne doit pas être captée dans el_1, et M4 ne doit pas apparaître dans le canal.

Exemple

- Pourquoi $E(el_1\{\emptyset, \{M1\}, \{M3\}\}) \rightarrow Pc$?
  - à cause de C1 : les émissions M1 et M3 sont captées par el_2 et el_3, comme les réceptions ne sont pas captées par el_1, elles doivent apparaître dans les canaux
  - à cause de C2 : l'émission de M4 n'est pas captée dans el_3, la réception ne doit pas être captée dans el_1, et M4 ne doit pas apparaître dans le canal.
- Pc peut maintenant mémoriser l'état global par : $S = \{\{el_1, el_2, el_3\},$

C_{ij}	j=1	2	3
i=1	$\emptyset$	$\emptyset$	$\emptyset$
2	M1	$\emptyset$	$\emptyset$
3	M3	$\emptyset$	$\emptyset$

$\}$

Exemple

Revient donc à dire que avec l'état el_1 , P_1 n'a pas pris en compte M1 et M3.

- La remise à Faux de toutes les variables $EtatMemorise_i$ pourra alors être faite par Pc

Méthode cumulative (mais rapide) de Lai et Yang

Basé sur la notion d'état des processus : état avant enregistrement de l'état local / état après l'enregistrement

On associe une "couleur" à chacun de ces états : **vert** pour l'état AVANT , **rouge** pour l'état APRÈS.

D'où l'algorithme de **Lai et Yang**

Chaque processus obéit aux règles suivantes :

- **R1** : Un processus qui n'a pas encore enregistré son état est vert et tous les messages qu'il émet sont verts.

Méthode cumulative (mais rapide) de Lai et Yang

Basé sur la notion d'état des processus : état avant enregistrement de l'état local / état après l'enregistrement

On associe une "couleur" à chacun de ces états : **vert** pour l'état AVANT , **rouge** pour l'état APRÈS.

D'où l'algorithme de **Lai et Yang**

Chaque processus obéit aux règles suivantes :

- **R1** : Un processus qui n'a pas encore enregistré son état est vert et tous les messages qu'il émet sont verts.
- **R2** : Lorsqu'un processus enregistre son état local, il devient rouge et tous les messages qu'il émet ensuite sont rouges.

Méthode cumulative (mais rapide) de Lai et Yang

Basé sur la notion d'état des processus : état avant enregistrement de l'état local / état après l'enregistrement

On associe une "couleur" à chacun de ces états : **vert** pour l'état AVANT , **rouge** pour l'état APRÈS.

D'où l'algorithme de **Lai et Yang**

Chaque processus obéit aux règles suivantes :

- **R1** : Un processus qui n'a pas encore enregistré son état est vert et tous les messages qu'il émet sont verts.
- **R2** : Lorsqu'un processus enregistre son état local, il devient rouge et tous les messages qu'il émet ensuite sont rouges.
- **R3** : Un processus peut à tout moment enregistrer son état local et doit l'enregistrer (si ce n'est pas déjà fait) lorsqu'il reçoit un message rouge, sans le prendre en compte.

Méthode cumulative (mais rapide) de Lai et Yang
D'où l'algorithme de **Lai et Yang**

- **R4** : Les messages verts émis et reçus sont stockés par le processus (aspect cumulatif).

L'état global est calculé par Pc lorsqu'il a reçu les informations de tous les processus.
Remarque : l'état ec_{ij} d'un canal est la différence entre l'ensemble des messages verts émis par P_i vers P_j (noté msg_$emis_i[j]$) et l'ensemble des messages verts reçus par P_j depuis P_i (noté msg_$recu_j[i]$).

Méthode cumulative (mais rapide) de Lai et Yang

D'où l'algorithme de **Lai et Yang**

- **R4** : Les messages verts émis et reçus sont stockés par le processus (aspect cumulatif).
- **R5** : Chaque processus qui devient rouge transmet au processus collecteur Pc, son état local et l'ensemble des messages verts qu'il a stockés

L'état global est calculé par Pc lorsqu'il a reçu les informations de tous les processus.

Remarque : l'état ec_{ij} d'un canal est la différence entre l'ensemble des messages verts émis par P_i vers P_j (noté msg_$emis_i[j]$) et l'ensemble des messages verts reçus par P_j depuis P_i (noté msg_$recu_j[i]$).

Méthode cumulative (mais rapide) de Lai et Yang

Montrons que C1 et C2 sont bien vérifiées :

- si l'émission de m est captée dans el_i , alors m est coloré en vert (R1) et $m \in msg\_emis_i[j]$ (R4) :

Ceci assure C1

Ce qui assure C2.

Méthode cumulative (mais rapide) de Lai et Yang

Montrons que C1 et C2 sont bien vérifiées :

- si l'émission de m est captée dans el_i , alors m est coloré en vert (R1) et $m \in msg\_emis_i[j]$ (R4) :
  - si sa réception est captée dans el_j (c-à-d qu'il était présent au moment de l'enregistrement) alors $m \in msg\_recu_j[i]$ (R4) (au msg_recu_j qui est transmis à Pc) et donc $m \notin ec_{ij}$

Ceci assure C1

Ce qui assure C2.

Méthode cumulative (mais rapide) de Lai et Yang

Montrons que C1 et C2 sont bien vérifiées :

- si l'émission de m est captée dans el_i , alors m est coloré en vert (R1) et $m \in msg\_emis_i[j]$ (R4) :
  - si sa réception est captée dans el_j (c-à-d qu'il était présent au moment de l'enregistrement) alors $m \in msg\_recu_j[i]$ (R4) (au msg_recu_j qui est transmis à Pc) et donc $m \notin ec_{ij}$
  - si sa réception n'est pas captée dans el_j alors $m \notin msg_e mis_i[j]$ (R4) et donc $m \in ec_{ij}$.

  Ceci assure C1

Ce qui assure C2.

Méthode cumulative (mais rapide) de Lai et Yang

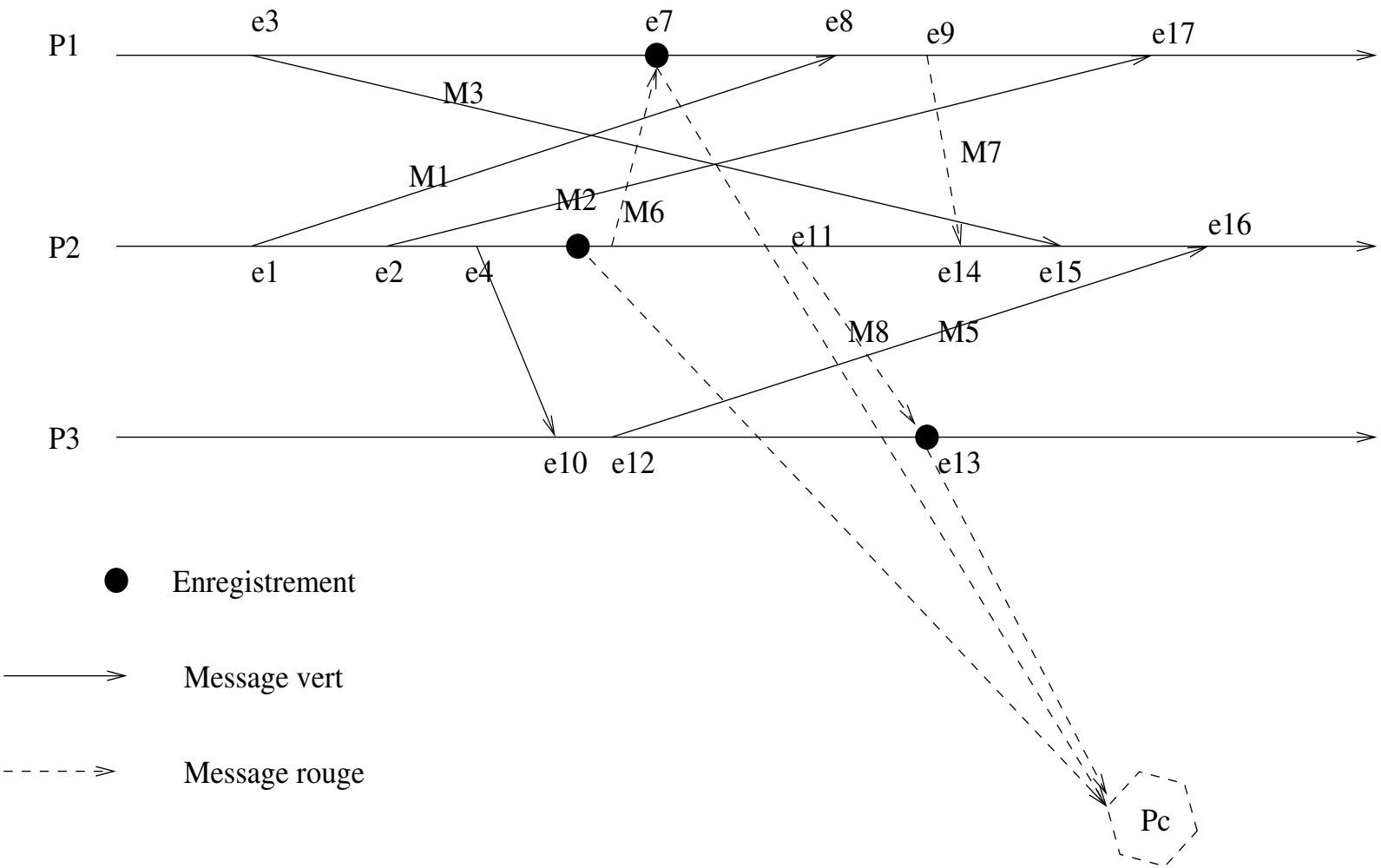
Montrons que C1 et C2 sont bien vérifiées :

- si l'émission de m est captée dans el_i , alors m est coloré en vert (R1) et $m \in msg\_emis_i[j]$ (R4) :
  - si sa réception est captée dans el_j (c-à-d qu'il était présent au moment de l'enregistrement) alors $m \in msg\_recu_j[i]$ (R4) (au msg_recu_j qui est transmis à Pc) et donc $m \notin ec_{ij}$
  - si sa réception n'est pas captée dans el_j alors $m \notin msg_e mis_i[j]$ (R4) et donc $m \in ec_{ij}$.

  Ceci assure C1
- si l'émission de m n'est pas captée dans el_i, alors m est coloré en rouge (R2). Lorsque m est reçu par P_j, ce dernier processus a déjà enregistré son état local, ou doit le faire sans prendre en compte m (R3). Donc la réception de m n'est pas captée dans el_js.

Ce qui assure C2.

Méthode cumulative (mais rapide) de Lai et Yang



Méthode cumulative (mais rapide) de Lai et Yang

Evénement	1	2	3	4	5	6	7	8	9	10
Couleur	V						R			
E1[2]			M3				∅			
R1[2]								M1		
E1[3]										
R1[3]										
Action			E(M3,v)				R(M6,R), el1	R(M1,v)	E(M7,R)	
Couleur	V				R					
E2[1]	M1	M1, M2			∅					
R2[1]										
E2[3]				M4	∅					
R2[3]										
Action	E(M1,v)	E(M2,v)		E(M4,v)	el2	E(M6,R)				
Couleur	V									
E3[1]										
R3[1]										
E3[2]										
R3[2]										M4
Action										R(M4,v)

Méthode cumulative (mais rapide) de Lai et Yang

Evénement	11	12	13	14	15	16	17	18	19
Couleur									
E1[2]									
R1[2]							{M1,M2}		
E1[3]									
R1[3]									
Action							R(M2,v)		
Couleur									
E2[1]									
R2[1]					M3	M3			
E2[3]									
R2[3]						M5			
Action	E(M8,R)			R(M7,R)	R(M3,v)	R(M5,v)			
Couleur			R						
E3[1]									
R3[1]									
E3[2]		M5	∅						
R3[2]		M4	∅						
Action		E(M5,v)	R(M8,R),el3						

Méthode cumulative (mais rapide) de Lai et Yang

Pc a reçu, en plus des états locaux : $E_1[2] = \{M3\}$, $E_2[1] = \{M1, M2\}$, $E_2[3] = \{M4\}$, $E_3[2] = \{M5\}$ et $R_3[2] = \{M4\}$

Comme, $\forall (i, j), i \neq j, ec_{ij} = E_i[j] \backslash R_j[i]$, on obtient

P_i	canal	1 $C_{y,1}$	2 $C_{y,2}$	3 $C_{y,3}$
1	$C_{1,x}$	x	{M3}	$\varnothing$
2	$C_{2,x}$	{M1, M2}	x	$\varnothing$
3	$C_{3,x}$	$\varnothing$	{M5}	x

Revient à dire que P_1 n'a pas pris en compte $\{M1, M2\}$, P_2 n'a pas pris en compte $\{M3, M5\}$, P_3 a tout pris en compte.

Problème : le volume des messages à stocker peut devenir important. D'où une méthode plus lente mais

Méthode non cumulative (mais lente) de Mattern

L'algorithme est basé sur la propriété suivante : Les messages en transit sur c_{ij} sont exactement les messages verts émis par le processus P_i (captés dans el_i) et reçus par le processus rouge P_j (non captés dans el_j).
Or si un processus P_j sait que les messages verts qu'il a reçu après son enregistrement sont les messages en transit sur le canal, il ne sait pas si il les a tous reçus car un message rouge peut très bien avoir doubler des messages verts. On va utiliser des compteurs de messages :

Méthode non cumulative (mais lente) de Mattern

Chaque fois qu'un processus P_i enregistre son état local, il rajoute, aux informations qu'il envoie au processus centralisateur, la différence d_i entre le nombre de messages émis (donc verts) et le nombre de messages verts reçus AVANT l'enregistrement. Le nombre exact mt de message en transit sur l'ensemble des canaux est alors : $mt = \sum_i d_i$

D'où l'algorithme de **Mattern :**

Méthode non cumulative (mais lente) de Mattern

Chaque processus obéit aux règles suivantes :

- **R1** : Un processus qui n'a pas encore enregistré son état est vert et tous les messages qu'il émet sont verts.

Méthode non cumulative (mais lente) de Mattern

Chaque processus obéit aux règles suivantes :

- **R1** : Un processus qui n'a pas encore enregistré son état est vert et tous les messages qu'il émet sont verts.
- **R2** : Lorsqu'un processus enregistre son état local, il devient rouge et tous les messages qu'il émet ensuite sont rouges.

- **R'4** : Chaque processus qui devient rouge transmet au processus collecteur Pc, son état local et son d_i.

Méthode non cumulative (mais lente) de Mattern

Chaque processus obéit aux règles suivantes :

- **R1** : Un processus qui n'a pas encore enregistré son état est vert et tous les messages qu'il émet sont verts.
- **R2** : Lorsqu'un processus enregistre son état local, il devient rouge et tous les messages qu'il émet ensuite sont rouges.
- **R3** : Un processus peut à tout moment enregistrer son état local et doit l'enregistrer (si ce n'est pas déjà fait) lorsqu'il reçoit un message rouge, sans le prendre en compte.
- **R'4** : Chaque processus qui devient rouge transmet au processus collecteur Pc, son état local et son d_i.
- **R'5** : Un processus rouge transmet à Pc tous les messages verts qu'il reçoit

Méthode non cumulative (mais lente) de Mattern

- Le processus Pc, lorsqu'il reçoit :

Méthode non cumulative (mais lente) de Mattern

- Le processus Pc, lorsqu'il reçoit :
  - un enregistrement (el_i, d_i) : il exécute N = N ? 1 (où N est initialisé au nombre de processus) et $mt = mt + d_i$

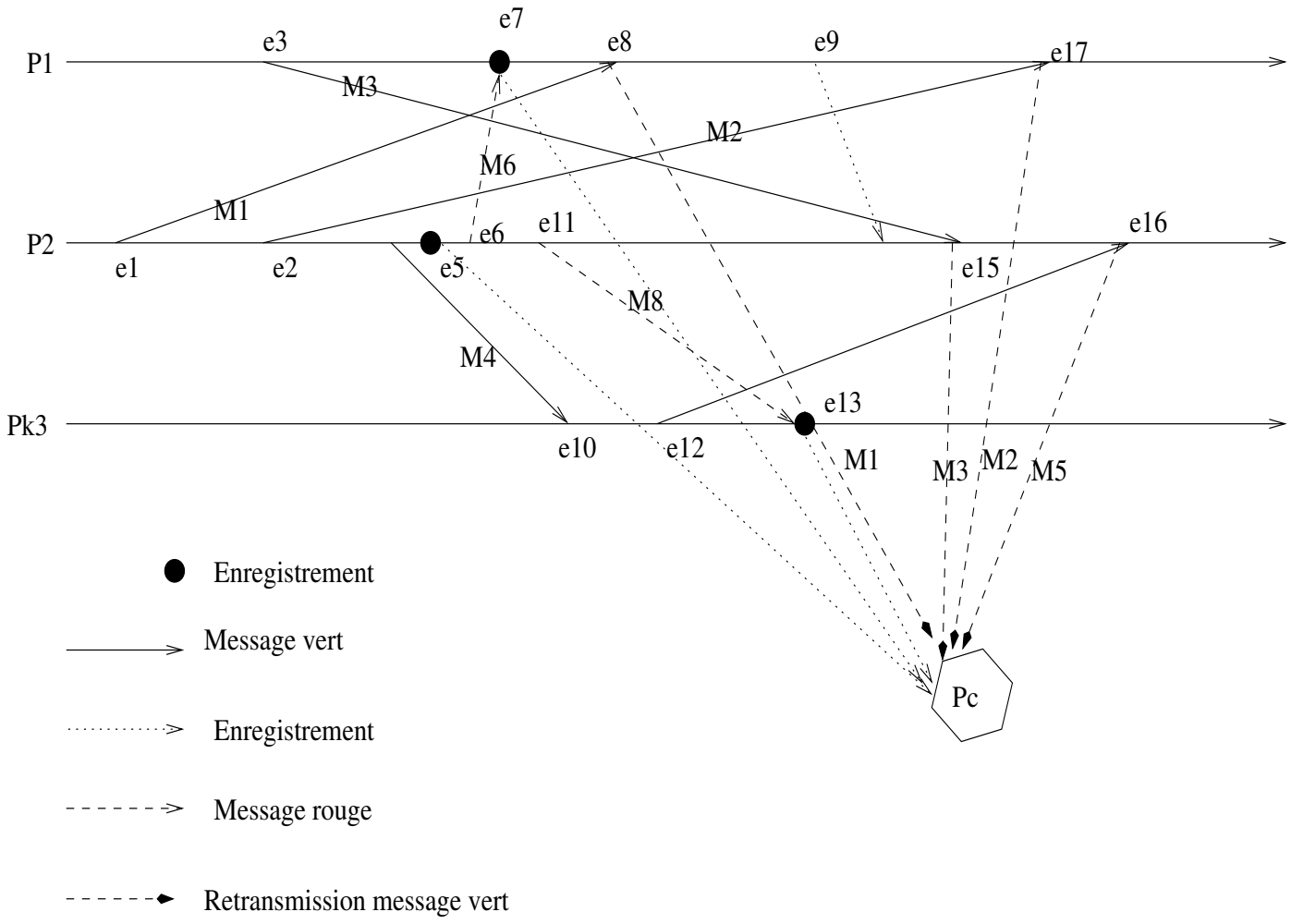
Méthode non cumulative (mais lente) de Mattern

- Le processus Pc, lorsqu'il reçoit :
  - un enregistrement (el_i, d_i) : il exécute N = N ? 1 (où N est initialisé au nombre de processus) et $mt = mt + d_i$
  - un message vert (M, (site emetteur P_i, site recepteur P_j)) (en provenance donc de P_j) : il rajoute M à c_{ij} et décrémente mt

Méthode non cumulative (mais lente) de Mattern

- Lors que N = 0 et mt = 0 , Pc dispose de tous les état.s locaux et de tous les messages en transit : il a donc l'état global.

Méthode non cumulative (mais lente) de Mattern



Solution pour des canaux non FIFO
Solutions sans messages de contrôle

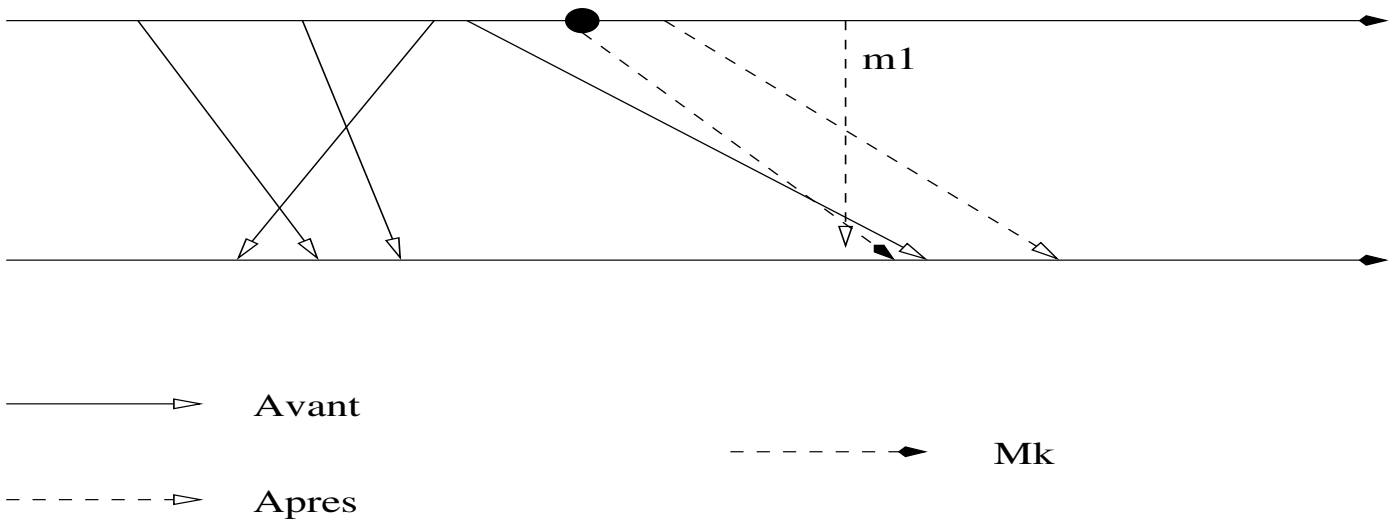
Remarque importante

Dans ces deux algorithmes, contrairement à l'algorithme de Chandry, le processus Pc est indispensable : il est le seul capable de calculer l'état global (dans l'algorithme de Chandry, chaque processus connaissait son état local et l'état de chacun de ses canaux : Pc ne servait qu'a avoir un vision "unique")

Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

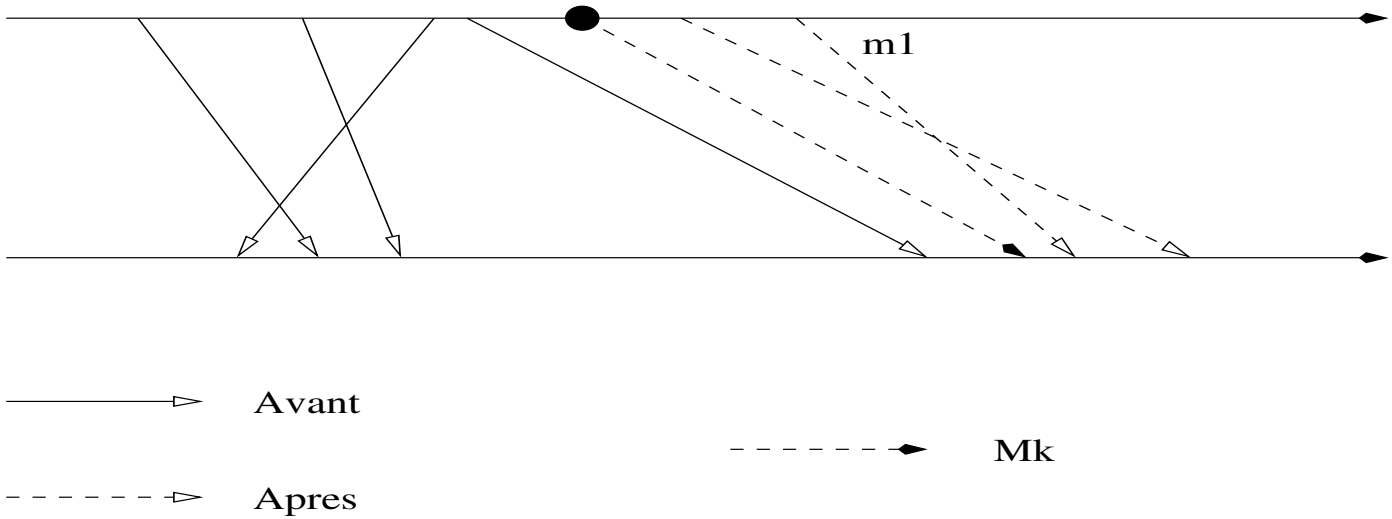
On peut adapter l'algorithme de Chandy et Lamport si on dispose de marqueur particulier. En effet une première adaptation de cet algorithme consiste à introduire un type de messages chargés de séparer l'ensemble des messages émis AVANT enregistrement de el_i et de l'ensemble des messages émis APRÈS. En effet, sans contrainte sur les messages on peut obtenir avec Chandy et Lamport si le réseau n'est plus FIFO.



Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

Si on introduit des messages Marqueur qui assure que tout message émis AVANT arrive AVANT et tout message émis APRÈS arrivera APRÈS, on obtient :

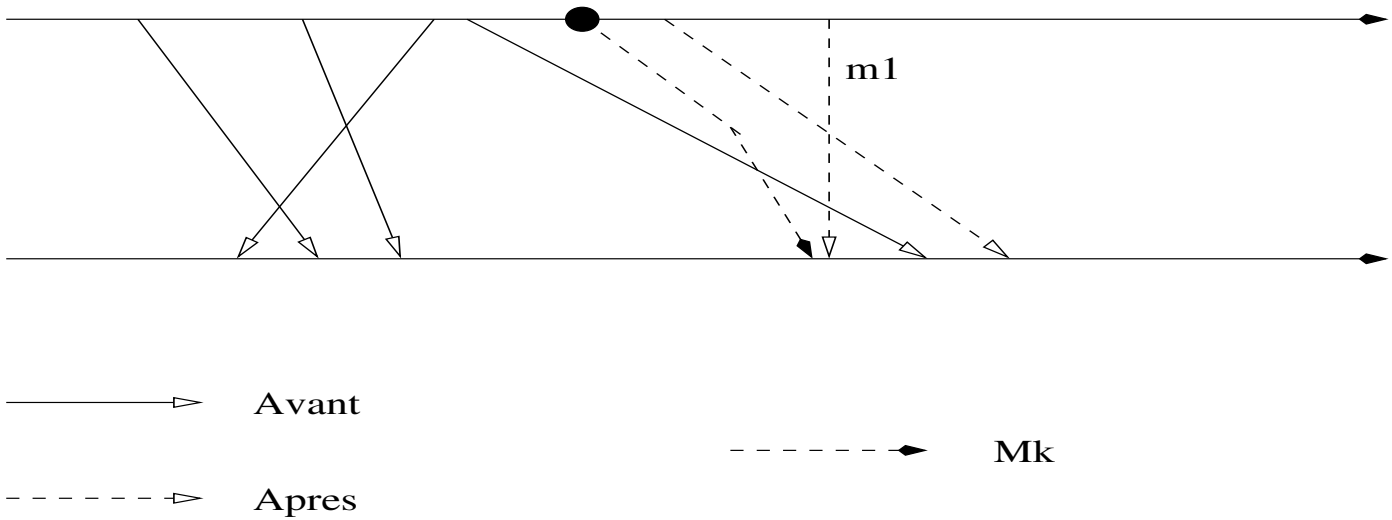


Or on voit que l'ordre de réception, sur cet exemple, est différent, ce qui interdit d'utiliser cette méthode car une des caractéristiques (contrainte) d'un algo de détermination d'un état global est qu'il ne doit pas perturber le phénomène observé (cf vol des oiseaux).

Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

On peut par contre introduire des marqueurs du type :
ct_futur : (dit contraint par le futur) qui garantissent simplement que tout message émis APRÈS arrivera APRÈS

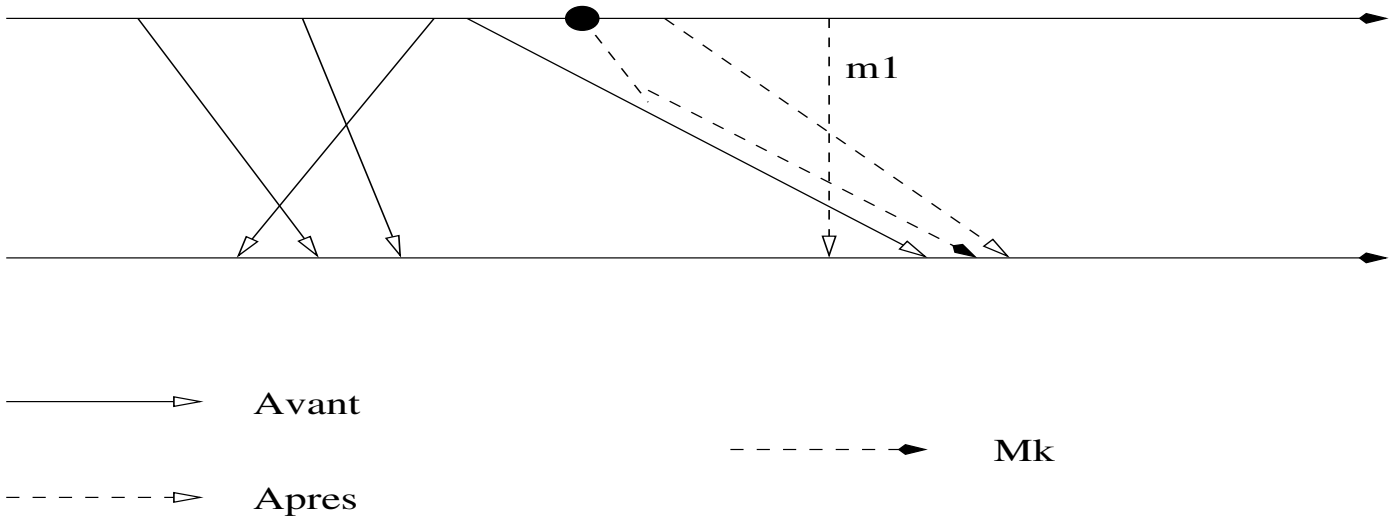


la réception de m1 peut éventuellement être retardée : mais l'ordre des messages reçus APRÈS le marqueur est conservé.

Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

ct_passe : (dit contraint par le passé) qui garantissent simplement que tout message émis AVANT arrivera AVANT



et alors on peut utiliser l'**algorithme d'Ahuja :**

Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

Algorithme d'Ahuja :

Chaque P_i enregistre son état local el_i et l'état de ses canaux en respectant les règles suivantes :

- **R1** : P_i lorsqu'il enregistre son état local, envoie sur tous ses canaux de sortie, un message Ek du type **ct_futur** avant d'envoyer tout autre message

Algorithme d'Ahuja :

Chaque P_i enregistre son état local el_i et l'état de ses canaux en respectant les règles suivantes :

- **R1** : P_i lorsqu'il enregistre son état local, envoie sur tous ses canaux de sortie, un message Ek du type **ct_futur** avant d'envoyer tout autre message
- **R2** : à la réception d'un message Ek, le processus P_i enregistre son état selon la règle **R1**, s'il ne la pas déjà fait.

Algorithme d'Ahuja :

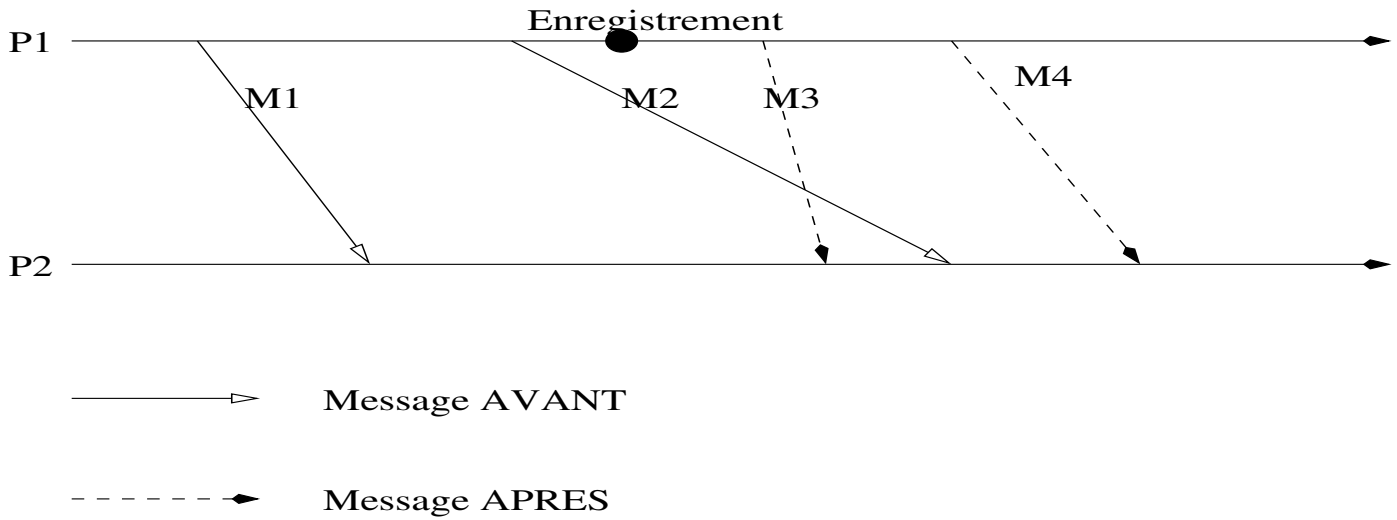
Chaque P_i enregistre son état local el_i et l'état de ses canaux en respectant les règles suivantes :

- **R1** : P_i lorsqu'il enregistre son état local, envoie sur tous ses canaux de sortie, un message Ek du type **ct_futur** avant d'envoyer tout autre message
- **R2** : à la réception d'un message Ek, le processus P_i enregistre son état selon la règle **R1**, s'il ne la pas déjà fait.
- **R3** : P_i lorsqu'il enregistre son état local envoie sur CHACUN de ses canaux de sortie, après le message Ek (du type **ct_futur**) et avant d'envoyer tout autre message, un message Ck du type **ct_passé** avec le même numéro que le dernier message émis sur ce canal : les messages sont numérotés par canal (le message estampillé t est le t ? ieme message envoyé

Algorithme d'Ahuja :

- **R4** : à la réception d'un message Ck numéroté t , sur un canal C_{ij} , P_j enregistre ec_{ij} comme étant l'ensemble des messages de numéro inférieur ou égal à t reçus après l'enregistrement de l'état local el_j.

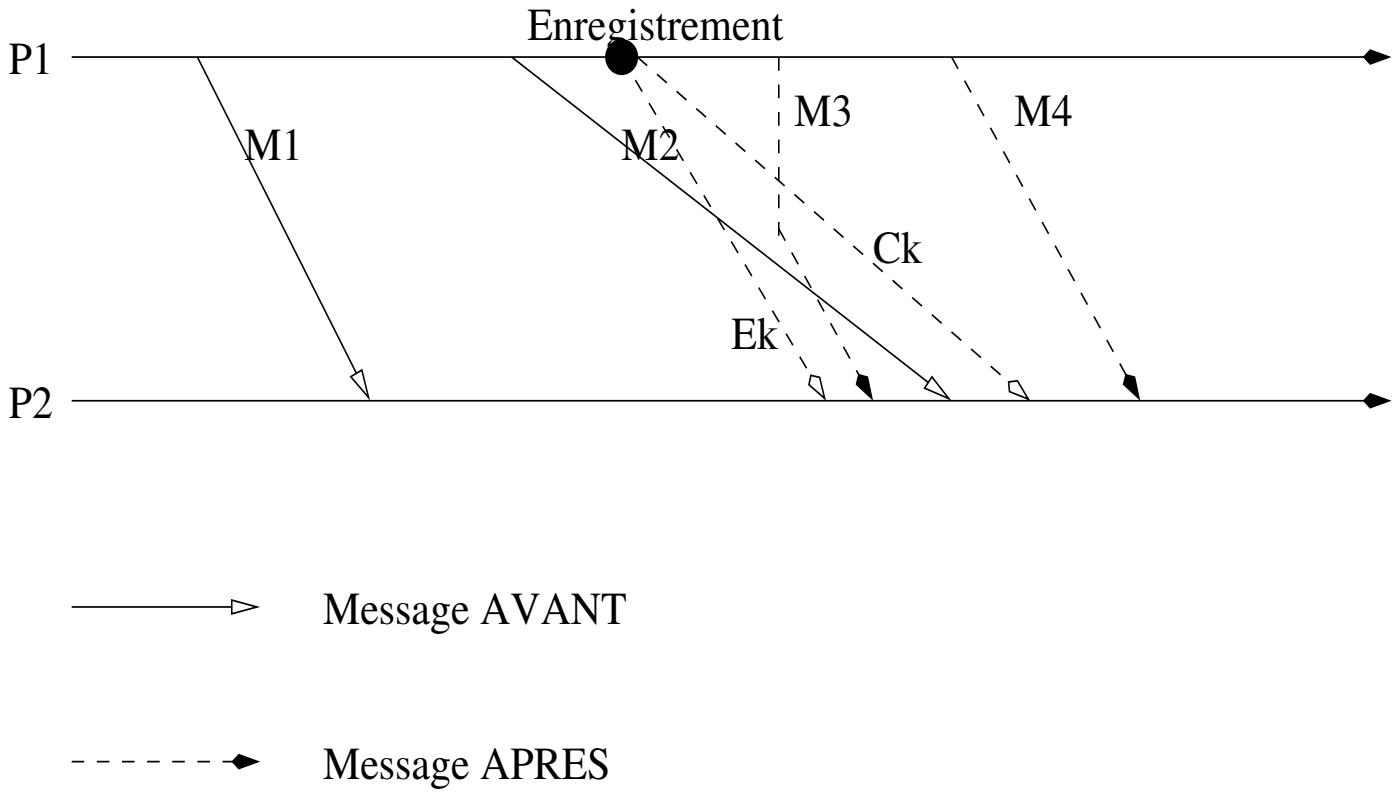
Ainsi sur l'exemple suivant :



Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

On obtient :



Montrons que cela marche.

Solution pour des canaux non FIFO
Solutions utilisant des messages de contrôle

- **R1** et **R2** assure C2 car elles permettent de capter des états locaux cohérents :
  En effet, si un message m, traversant C_{ij}, est tel que $emission_i$(m) n'est pas captée dans el_i (exemple M3 et M4) , alors m est émis APRÈS le message Ek sur C_{ij} ; m est donc reçu APRÈS ce message (car celui-ci est **ct_futur**) et donc P_j aura déjà capté son état (dès la réception de Ek) et donc $reception_j$(m) ne sera pas captée dans el_j.

Solution pour des canaux non FIFO
Solutions utilisant des messages de contrôle

- **R1** et **R2** assure C2 car elles permettent de capter des états locaux cohérents :
  En effet, si un message m, traversant C_{ij}, est tel que $emission_i$(m) n'est pas captée dans el_i (exemple M3 et M4) , alors m est émis APRÈS le message Ek sur C_{ij} ; m est donc reçu APRÈS ce message (car celui-ci est **ct_futur**) et donc P_j aura déjà capté son état (dès la réception de Ek) et donc $reception_j$(m) ne sera pas captée dans el_j.
- **R3** et **R4** assure C1 car elles permettent de bien saisir l'état des canaux : En effet, soit un message m, traversant C_{ij}, émis AVANT l'enregistrement de el_i ($emission_i$(m) est donc captée dans celui-ci) (exemple M1 et M2) : son estampille t_m est nécessairement $\leq$ t de Ck.

Solution pour des canaux non FIFO

Solutions utilisant des messages de contrôle

Deux cas :

- soit m arrive avant Ek : pas de problème, $reception_j$(m) sera captée dans el_j (exemple M1)

donc C2 est vérifiée. L'avantage de cet algorithme par rapport à Lai ou à Mattern, c'est qu'il n'y a pas besoin de site centralisateur.

Solution pour des canaux non FIFO
Solutions utilisant des messages de contrôle

Deux cas :

- soit m arrive avant Ek : pas de problème, $reception_j$(m) sera captée dans el_j (exemple M1)
- soit m arrive après Ek : alors il arrive nécessairement AVANT Ck car celui-ci ne peut être dépassé (ct_passe) (exemple M2) Donc, lorsque P_j enregistrera l'état du canal, il aura nécessairement reçu m comme $t_m \leq t$ il sera mis dans l'état du canal. Réciproque, si un message mm (émis APRÈS Ck arrive AVANT Ck (exemple M3) son estampille *tt* sera nécessairement supérieure à t et *mm* ne sera pas captée dans ec_{ij}.

donc C2 est vérifiée. L'avantage de cet algorithme par rapport à Lai ou à Mattern, c'est qu'il n'y a pas besoin de site centralisateur.

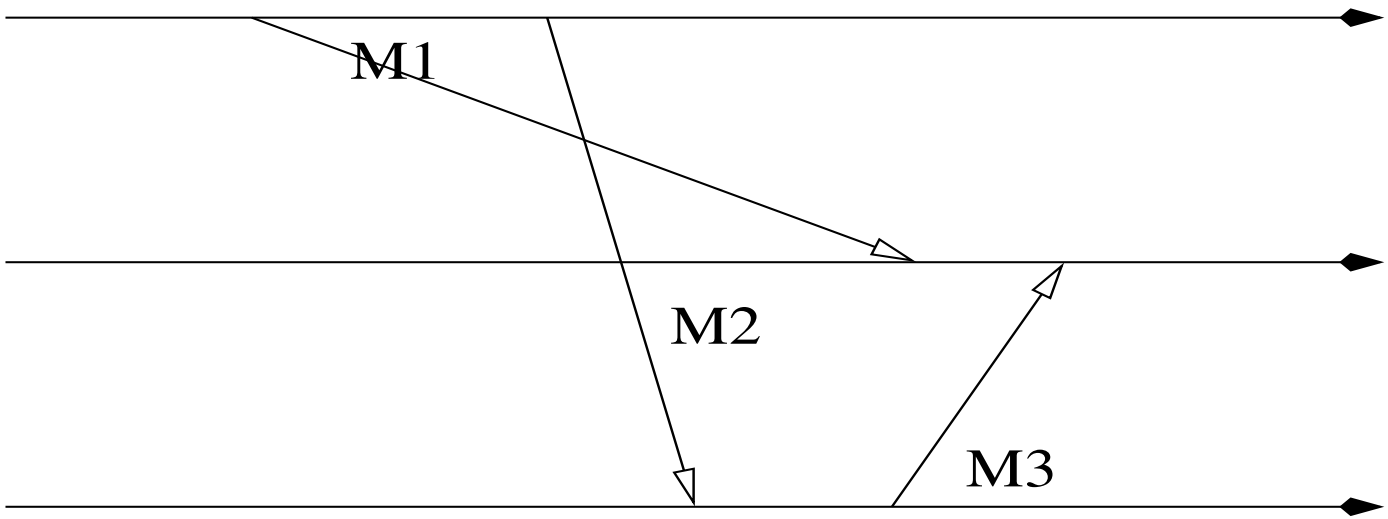
Solutions fondées sur l'ordre causal

Rappel : dans un système, on dira que l'ensemble des messages respecte l'ordre causal si pour la relation "précède" noté $\rightarrow$:

$\forall\ P_i, P_j, P_k, \forall m_1$ emis sur $C_{ij}, \forall m_2$ emis sur C_{kj} :

$emission_i(m_1) \rightarrow emission_k(m_2) \Rightarrow reception_j(m_1) \rightarrow reception_j(m_2)$

Exemple :



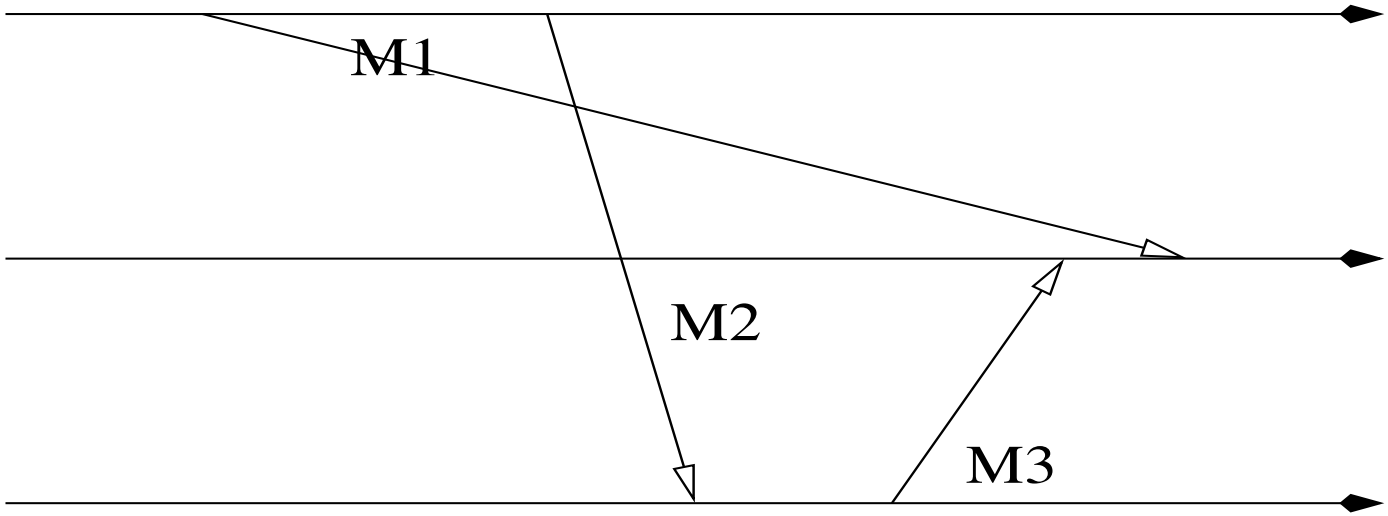
Solutions fondées sur l'ordre causal

On voit que, par exemple que $emission_1(m_1) \rightarrow emission_1(m_2)$ et $emission_1(m_2) \rightarrow emission_3(m_3)$ d'où $emission_1(m_1) \rightarrow emission_3(m_3)$ d'où il faut $reception_2(m_1) \rightarrow reception_2(m_3)$ ce qui est bien le cas.

Attention : l'ordre causal définit un ordre sur la réception des messages strictement plus fort que celui imposé par des canaux FIFO. De fait, si la propriété de causalité est respectée alors chaque canal a un comportement FIFO. Alors que le fait que tous les canaux soit FIFO ne garantit pas l'ordre causal.

Solutions fondées sur l'ordre causal

Exemple :



De ce fait, on peut d'attendre à des algorithmes plus simples.

Calcul des états locaux :

- **R1 :** L'initiateur P_{init} diffuse (y compris a lui même) un message de contrôle enreg_etat_local

Démonstration de C2

Considérons un message m envoyé sur le canal c_{ij} tel que $emission_i$(m) n'est pas capté dans el_i mais $reception_j$(m) captée dans el_j, ce qui serait une violation de C2.

Par exemple :

Calcul des états locaux :

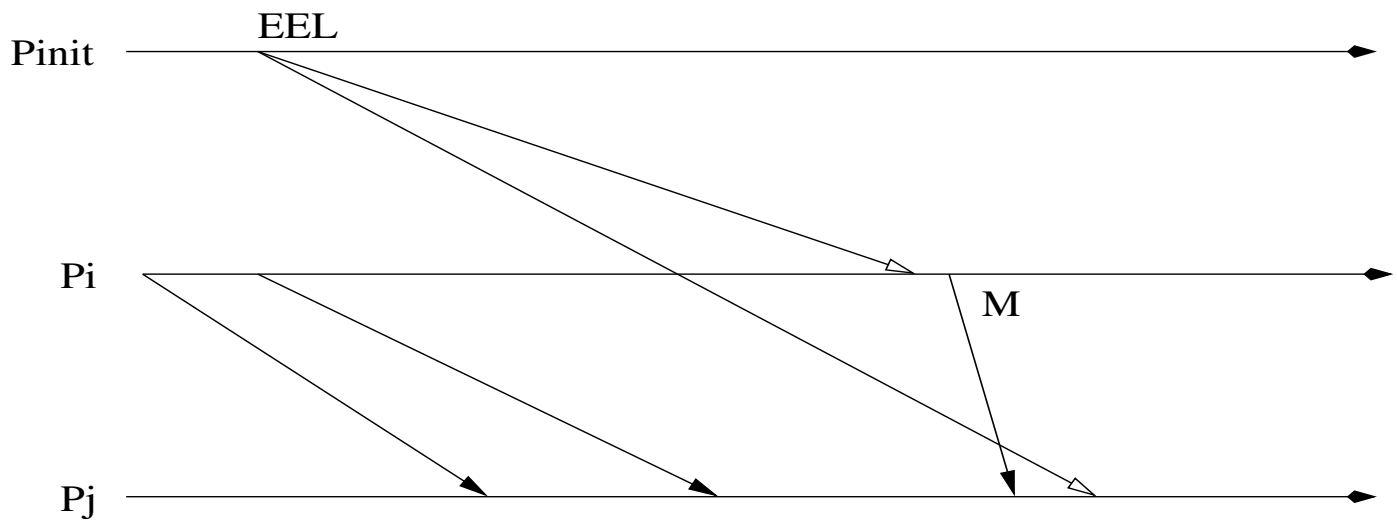
- **R1 :** L'initiateur P_{init} diffuse (y compris a lui même) un message de contrôle enreg_etat_local
- **R2 :** Lorsqu'un processus P_i reçoit ce message de contrôle, il enregistre son état local el_i

Démonstration de C2

Considérons un message m envoyé sur le canal c_{ij} tel que $emission_i$(m) n'est pas capté dans el_i mais $reception_j$(m) captée dans el_j, ce qui serait une violation de C2.

Par exemple :

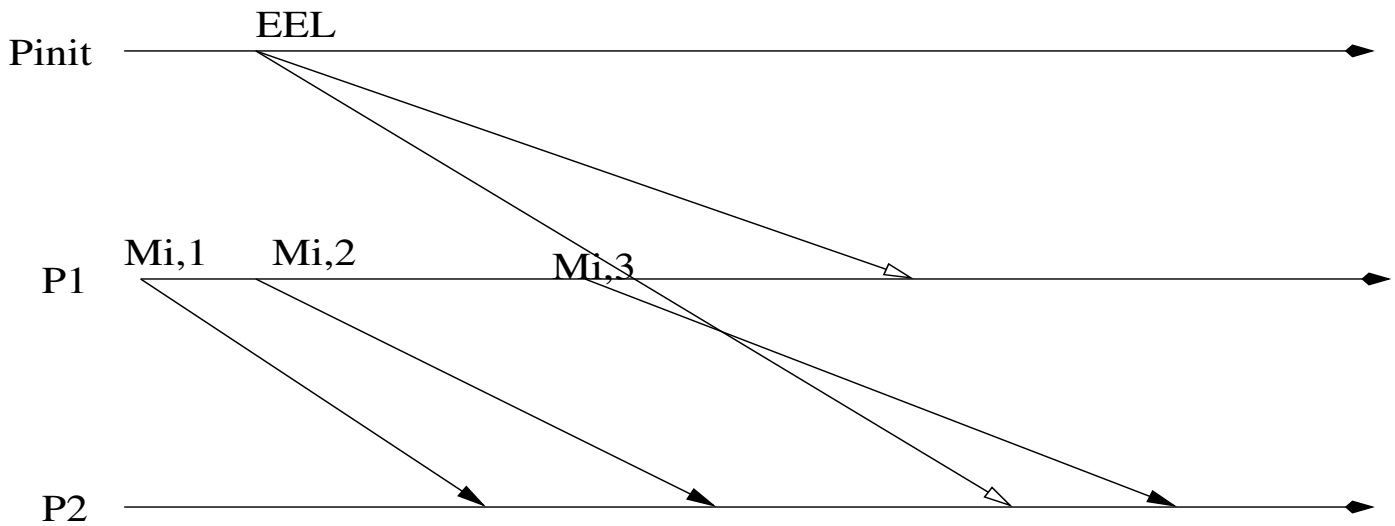
Calcul des états locaux :



Le message m a donc été émis par P_i après que celui-ci ait reçu le dernier message de contrôle *enreg_etat_local*, donc : $reception_i(enreg\_etat\_local) \rightarrow emission_i(m)$ d'où $emission_{init}(enreg\_etat\_local) \rightarrow reception_i(enreg\_etat\_local) \rightarrow emission_i(m)$ d'où $emission_{init}(enreg\_etat\_local) \rightarrow emission_i(m)$.

Calcul des états locaux : Si sa réception était captée dans el_j on aurait $reception_j(m) \rightarrow reception_j(enreg\_etat\_local)$ et l'hypothèse d'ordre causal serait violée.

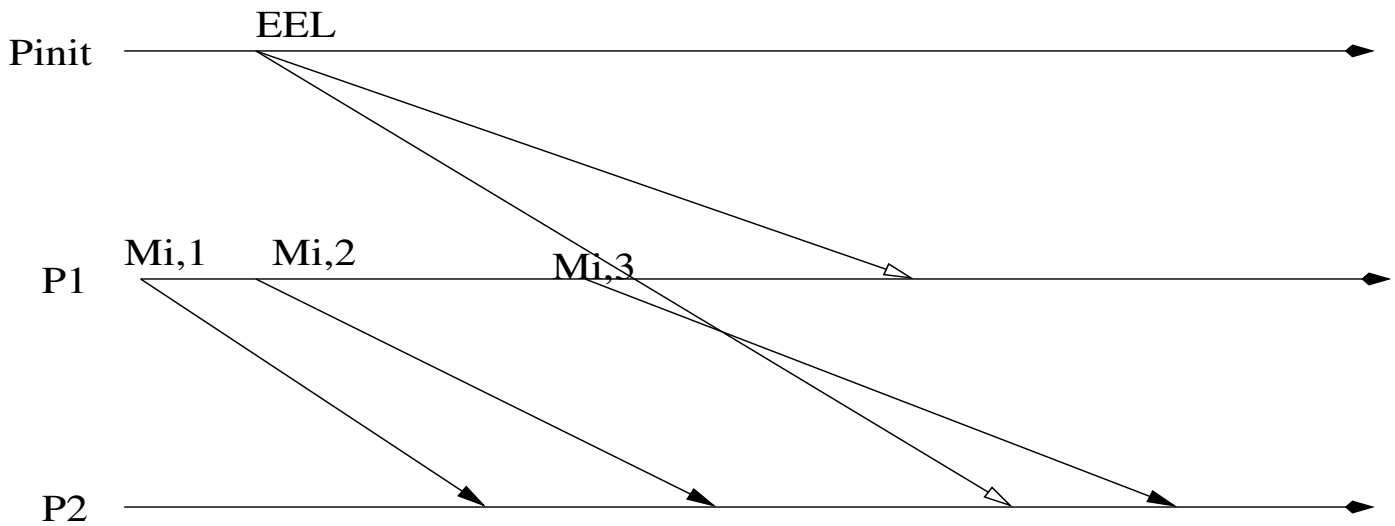
Calcul de l'état des canaux : Le cas suivant est toujours possible :



Pour assurer la captation de l'état des canaux, chaque P_i gère deux tableaux :

- $S_i[1..N]$ tel que $S_i[j] = n_{ij} \Leftrightarrow P_i$ a envoyé n_{ij} messages à P_j

Calcul de l'état des canaux : Le cas suivant est toujours possible :



Pour assurer la captation de l'état des canaux, chaque P_i gère deux tableaux :

- $S_i[1..N]$ tel que $S_i[j] = n_{ij} \Leftrightarrow P_i$ a envoyé n_{ij} messages à P_j
- $R_i[1..N]$ tel que $R_i[j] = n_{ji} \Leftrightarrow P_i$ a reçu n_{ji} messages de P_j

Calcul de l'état des canaux :

Dans l'exemple $(S_1 = \{3\}, R_1 = \{\})$ et $(S_2 = \{\}, R_2 = \{2\})$

R2' : Lorsqu'un P_i reçoit le message enreg_etat_local, il enregistre son état local el_i et les deux tableaux, puis il envoie à P_{init} le message $etat\_local(el_i, S_i, R_i)$

Soit m le t^{ieme} message émis par P_i vers P_j , c'est donc aussi let^{ieme} messages qui sera reçu par P_j depuis P_i (ordre causal) :

Supposons que $R_j[i] < t \leq S_i[j]$

$\Leftrightarrow$ $reception_j(m)$ non captée dans el_j (puisque m n'avait pas encore été reçu par P_j) mais par contre $emission_i(m)$ est captée dans el_i

$\Leftrightarrow$ m en transit relativement aux états locaux el_i et el_j d'où $m \in c_{ij} \Leftrightarrow$ m doit donc être mis dans l'état du canal.

Calcul de l'état des canaux :
D'où, pour garantir C1 :
R3 : lorsque l'initiateur P_{init} a reçu tous les messages $etat\_local(el_i, S_i, R_i)$, il possède tous les états locaux et calcule l'état des canaux par :

- $\forall j,\ c_{init,j} = \emptyset$

(On suppose que P_{init} a accès a tous les messages : fréquent dans des algorithmes chargés de garantir la causalité)

Calcul de l'état des canaux :
D'où, pour garantir C1 :
R3 : lorsque l'initiateur P_{init} a reçu tous les messages $etat\_local(el_i, S_i, R_i)$, il possède tous les états locaux et calcule l'état des canaux par :

- $\forall\, j,\ c_{init,j} = \emptyset$
- $\forall\, i \neq init, \forall\, j, c_{ij} = \{M_{i,R_j[i]+1} \cdots M_i, S_i[j]\}$ ce sont tous messages que P_i a envoyés à P_j et que P_j n'a pas encore reçus

(On suppose que P_{init} a accès a tous les messages : fréquent dans des algorithmes chargés de garantir la causalité)

Solutions fondées sur l'ordre causal
Solution répartie de Alagar et Venkatesan

Chaque processus va devoir pouvoir calculer son état local mais aussi l'état de ses canaux entrants.
On utilise un technique de coloration non pas à l'émission, mais à la réception des messages : lorsque P_j reçoit un message m, celui-ci est coloré en **rouge** si et seulement si : $emision_{init}(enreg\_etat\_local) \rightarrow emission_i(m)$ (les structures de données maintenues par les protocoles de maintient de l'ordre causal permettent à P_j de faire ce test)

- **R1 :** L'initiateur P_{init} diffuse (y compris a lui même) un message de contrôle enreg_etat_local

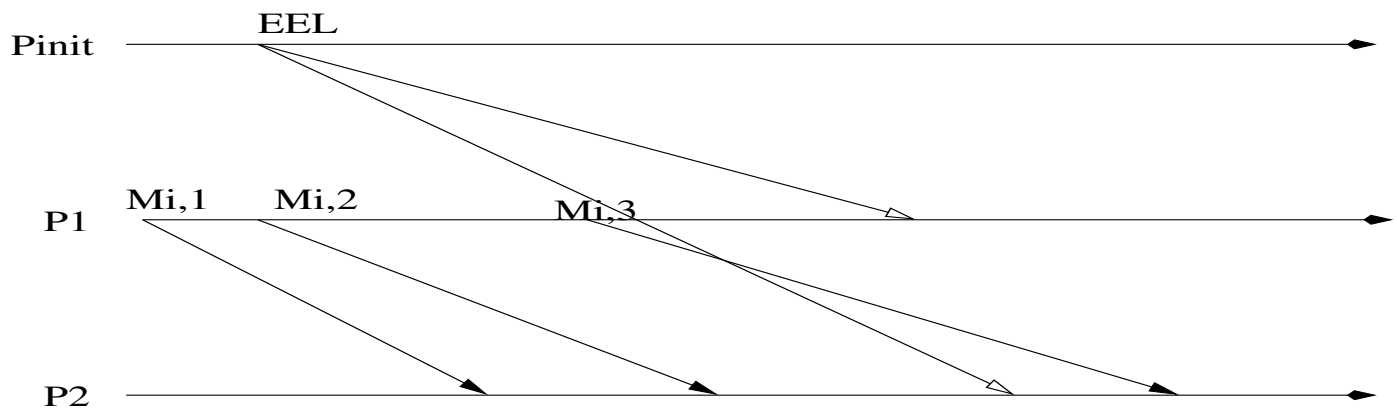
Solutions fondées sur l'ordre causal
Solution répartie de Alagar et Venkatesan

Chaque processus va devoir pouvoir calculer son état local mais aussi l'état de ses canaux entrants.
On utilise un technique de coloration non pas à l'émission, mais à la réception des messages : lorsque P_j reçoit un message m, celui-ci est coloré en **rouge** si et seulement si : $emision_{init}(enreg\_etat\_local) \rightarrow emission_i(m)$ (les structures de données maintenues par les protocoles de maintient de l'ordre causal permettent à P_j de faire ce test)

- **R1 :** L'initiateur P_{init} diffuse (y compris a lui même) un message de contrôle enreg_etat_local
- **R2" :** Lorsque P_i reçoit le message enreg_etat_local, il enregistre son état local el_i, initialise l'état de ses canaux entrants à $\emptyset$ et répond à P_i un message FAIT.

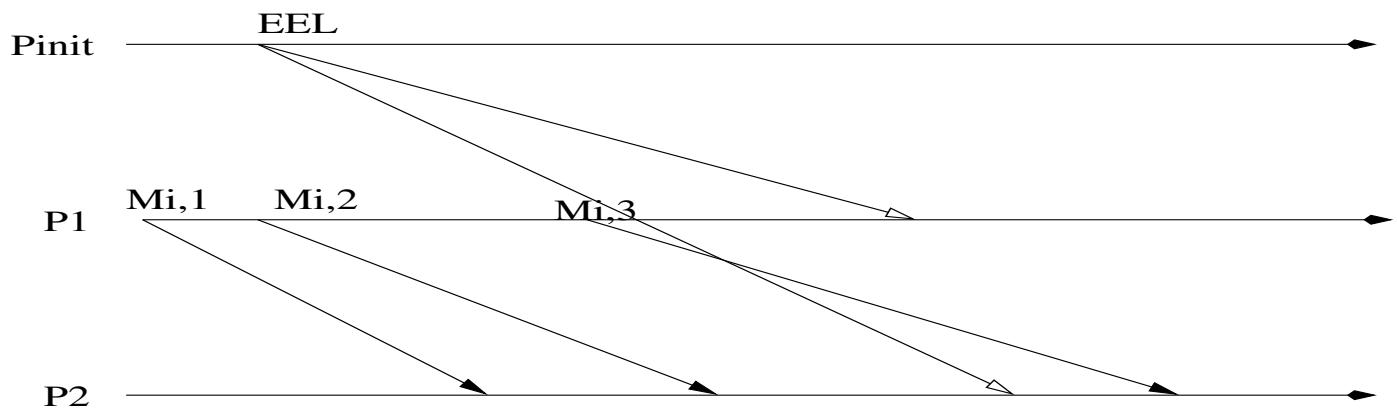
- **R3 :** Lorsque, après avoir enregistré son état local, P_j reçoit un message sur c_{ij} il teste si ce message est à colorer en **rouge**. S'il ne l'est pas, il le met dans l'ensemble ec_{ij}

Exemple :



- **R3 :** Lorsque, après avoir enregistré son état local, P_j reçoit un message sur c_{ij} il teste si ce message est à colorer en **rouge**. S'il ne l'est pas, il le met dans l'ensemble ec_{ij}
- **R4 :** Lorsque P_{init} a reçu un FAIT de tous les processus, il diffuse TERMINE

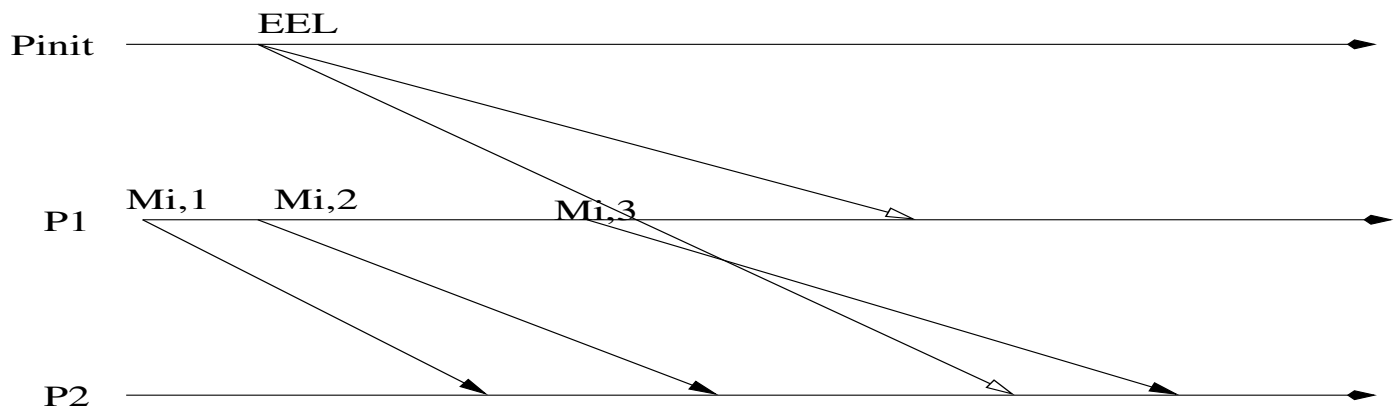
Exemple :



Solutions fondées sur l'ordre causal
Solution répartie de Alagar et Venkatesan

- **R3 :** Lorsque, après avoir enregistré son état local, P_j reçoit un message sur c_{ij} il teste si ce message est à colorer en **rouge**. S'il ne l'est pas, il le met dans l'ensemble ec_{ij}
- **R4 :** Lorsque P_{init} a reçu un FAIT de tous les processus, il diffuse TERMINE
- **R5 :** Lorsque P_i reçoit TERMINE il enregistre l'état de ses canaux.

Exemple :



Démonstration

Montrons que lorsque que P_j reçoit TERMINE, il a nécessairement reçu de chaque $P_{i\neq j}$ tous les messages M enregistrés dans les états locaux des $P_{i\neq j}$
Soit m un tel message sur c_{ij} (donc en transit entre P_i et P_j avec $emission_i(m)$ captée dans el_i). On a :

- $emission_i(m) \rightarrow reception_i(enreg\_etat\_local)$ car m est capté dans el_i

d'où $emission_i(m) \rightarrow emission_{init}(TERMINE)$ (transitivité de $\rightarrow$)

Démonstration

Montrons que lorsque que P_j reçoit TERMINE, il a nécessairement reçu de chaque $P_{i\neq j}$ tous les messages M enregistrés dans les états locaux des $P_{i\neq j}$
Soit m un tel message sur c_{ij} (donc en transit entre P_i et P_j avec $emission_i(m)$ captée dans el_i). On a :

- $emission_i(m) \rightarrow reception_i(enreg\_etat\_local)$ car m est capté dans el_i
- $reception_i(enreg\_etat\_local) \rightarrow emission_i(FAIT)$ (règle **R"**)

d'où $emission_i(m) \rightarrow emission_{init}(TERMINE)$ (transitivité de $\rightarrow$)

Démonstration

Montrons que lorsque que P_j reçoit TERMINE, il a nécessairement reçu de chaque $P_{i\neq j}$ tous les messages M enregistrés dans les états locaux des $P_{i\neq j}$
Soit m un tel message sur c_{ij} (donc en transit entre P_i et P_j avec $emission_i(m)$ captée dans el_i). On a :

- $emission_i(m) \rightarrow reception_i(enreg\_etat\_local)$ car m est capté dans el_i
- $reception_i(enreg\_etat\_local) \rightarrow emission_i(FAIT)$ (règle **R"**)
- $emission_i(FAIT) \rightarrow reception_{init}(FAIT)$ (par définition)

d'où $emission_i(m) \rightarrow emission_{init}(TERMINE)$ (transitivité de $\rightarrow$)

Démonstration

Montrons que lorsque que P_j reçoit TERMINE, il a nécessairement reçu de chaque $P_{i\neq j}$ tous les messages M enregistrés dans les états locaux des $P_{i\neq j}$
Soit m un tel message sur c_{ij} (donc en transit entre P_i et P_j avec $emission_i(m)$ captée dans el_i). On a :

- $emission_i(m) \rightarrow reception_i(enreg\_etat\_local)$ car m est capté dans el_i
- $reception_i(enreg\_etat\_local) \rightarrow emission_i(FAIT)$ (règle **R"**)
- $emission_i(FAIT) \rightarrow reception_{init}(FAIT)$ (par définition)
- $reception_{init}(FAIT) \rightarrow emission_{init}(TERMINE)$ (**R4**)

d'où $emission_i(m) \rightarrow emission_{init}(TERMINE)$ (transitivité de $\rightarrow$)

Démonstration

d'où, d'après l'ordre causal :
$reception_j(m) \rightarrow reception_j(TERMINE)$: il a bien reçu tous les messages en transit lorsqu'il enregistre les états de ses canaux.
Pour démontrer C1 il faut encore montrer que les messages reçus après $reception_j(enreg\_etat\_local)$ qui sont donc tel que $reception_j(m)$ est non captée dans el_j seront bien mis dans l'état du canal c_{ij} . Soit m tel que $reception_j(m)$ est non captée dans
$el_j \Leftrightarrow reception_j(enreg\_etat\_local) \rightarrow receptionj(m)$
Or m n'est pas à colorer en **rouge**. En effet, $emission_i(m)$ est captée dans el_i, donc
$emission_{init}(enreg\_etat\_local)$ not $\rightarrow emission_i(m)$
$\Rightarrow$ m n'est donc pas coloré en **rouge**, d'où P_j enregistre le message.

Démonstration

$\Rightarrow$ le message m est bien enregistré dans ec_{ij} (**R3** et **R5**)
Donc, lorsque P_i reçoit TERMINE, il a enregistré tous les messages reçus soit dans son état eli soit dans les canaux de communication. La condition C1 est donc bien respectée.
Réciproquement, il est évident que si m est enregistré dans ec_{ij} on a nécessairement :
$reception_j(enreg\_etat\_local) \rightarrow reception_j(m) \rightarrow reception_j(TERMINE)$ et m n'est pas coloré en rouge.
D'où si m est enregistré dans ec_{ij} alors nécessairement :

- $reception_j(m)$ n'est pas captée dans el_j

Ce qui permet d'assurer C2.

Démonstration

$\Rightarrow$ le message m est bien enregistré dans ec_{ij} (**R3** et **R5**)
Donc, lorsque P_i reçoit TERMINE, il a enregistré tous les messages reçus soit dans son état eli soit dans les canaux de communication. La condition C1 est donc bien respectée.
Réciproquement, il est évident que si m est enregistré dans ec_{ij} on a nécessairement :
$reception_j(enreg\_etat\_local) \rightarrow reception_j(m) \rightarrow reception_j(TERMINE)$ et m n'est pas coloré en rouge.
D'où si m est enregistré dans ec_{ij} alors nécessairement :

- $reception_j(m)$ n'est pas captée dans el_j
- $emission_i(m)$ est captée dans el_i

Ce qui permet d'assurer C2.