Chapitre 5

Détermination d’un état global

Chap 5 Détermination d'un état global

- L'observation d'un état global permet :
  - ⇒ de détecter des propriétés caractérisant l'exécution
    (terminaison, interblocage, vérification d'assertions, etc.)
  - ⇒ d'effectuer des mesures de performances
  - ⇒ de capter des états pouvant servir de points de reprise en cas de défaillance
- Difficulté d'observation globale dans un contexte réparti :
  - ⇒ pas d'horloge commune
  - ⇒ temps de transfert des messages non borné (fini si pas de perte)
  - ⇒ impossible d'effectuer une observation simultanée
    ⇒ exemple des photographies des vols d'oiseaux
- Objectif :
  - Détermination d'un état global cohérent qui peut être observé
    ⇒ constitué des états locaux des sites et des canaux de communication

1. Etat global

1.1. Définitions [Raynal]

Chaque processus et chaque canal possède à tout moment un **état local** :

- - l'état local el_i d'un processus P_i résulte de son état initial et de la séquence d'événements dont ce processus a été le siège.
- - l'état ec_{ij} d'un canal c_{ij} est l'ensemble des messages en transit sur ce canal, c'est à dire qui ont été émis par le processus P_i et n'ont pas encore été reçus par le processus P_j.

1.1. Définitions

Chaque **événement** met en jeu un processus et éventuellement un canal. On distingue :

- *événement interne* sur P_i : provoque la transition de el_i à el_i', états avant et après l'événement
- *émission* de *m* par P_i sur c_{ij} (cet événement est noté $émission_i(m)$) qui provoque :
  - la transition de el_i à el_i' et l'affectation $ec_{ij} := ec_{ij} \cup \{m\}$
- *réception* de *m* par P_i sur c_{ji} (cet événement est noté $réception_i(m)$) qui provoque :
  - la transition de el_i à el_i' et l'affectation $ec_{ji} := ec_{ji} \setminus \{m\}$.

Chacun de ces événements est supposé *atomique*

1.1. Définitions

- l'état local d'un processus n'est immédiatement observable que par un observateur local à ce processus
- l'état local d'un canal c_{ij} n'est immédiatement observable ni par son origine P_i ni par son extrémité P_j

- **L'état global *S*** d'un système réparti est constitué de l'ensemble des états locaux des processus et des canaux qui le constituent:

$$S = \{\cup\, e_{li} \,,\, \cup\, ec_{ij}\,\}$$

- Un **état global cohérent** correspond à un état global dans lequel le système peut se trouver.

1.1. Définitions

Formellement, un **état global cohérent** est tel que :

i) el_i est un état local du processus P_i

ii) Les conditions **C***1* et **C***2* suivantes sont vérifiées:

- **C***1***)** : si l'événement *émission*$_i$ (*m*) est capté dans el_i, alors l'événement *réception*$_j$ (*m*) est soit capté dans el_j, soit le message *m* appartient à ec_{ij}
- **C2)** : si l'événement *émission*$_i$ (*m*) n'est pas capté dans el_i , l'événement *réception*$_j$ (*m*) n'est pas non plus capté dans el_j

Un état global cohérent est aussi appelé "**coupe cohérente**" **(« *consistent cut* »)**.

1.1. Définitions

Exemple

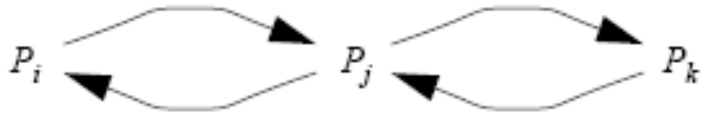


Figure 1. Un graphe de communication.

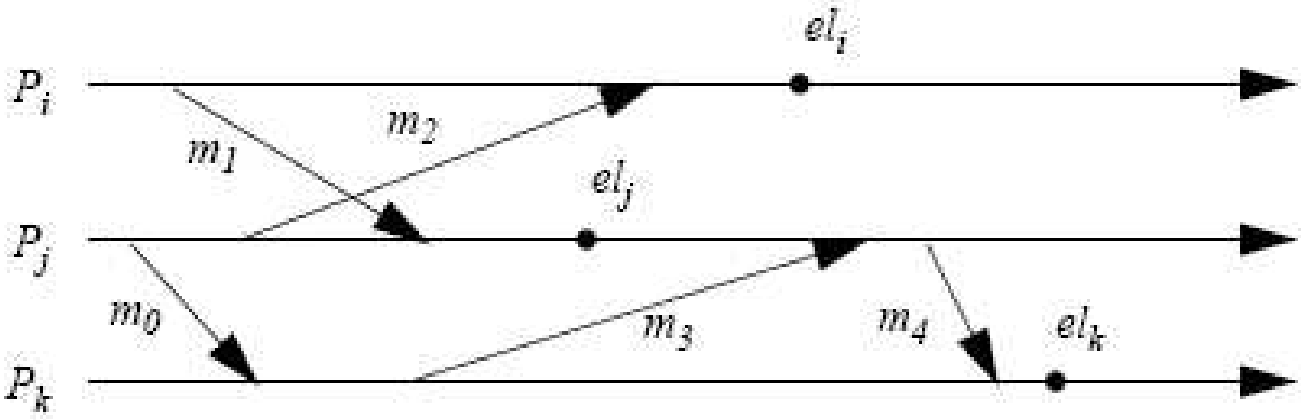


Figure 2. Une exécution répartie.

1.1. Définitions

Définition équivalente d'une coupure cohérente [Rifflet] :

- Une **coupure C** de l'histoire globale d'un système est un n-uplet dont chaque composante est un préfixe de l'histoire locale du site correspondant :

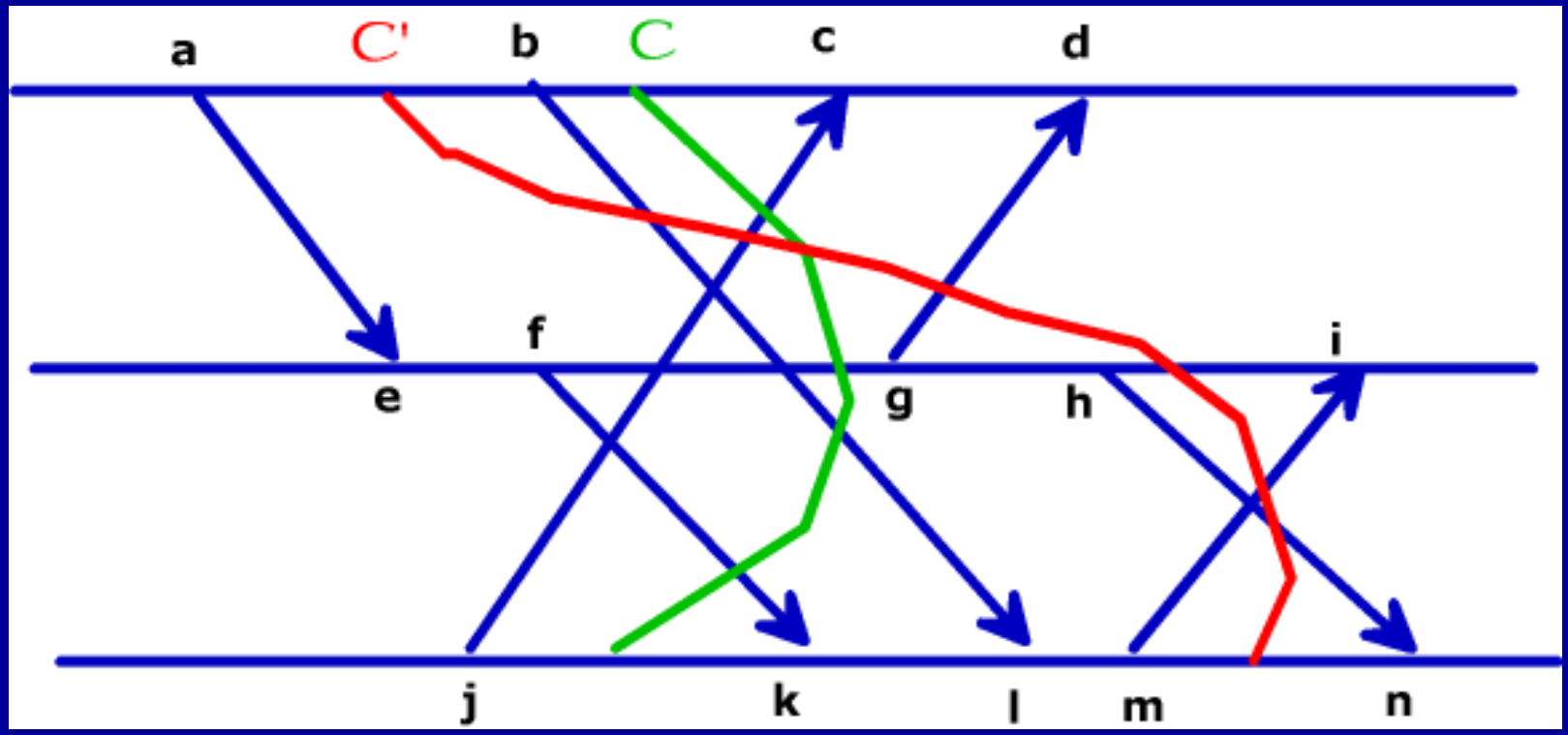
  $C = < c_1, \ldots, c_j , \ldots c_n >$ où $c_j = <e_{j1}, e_{j2}, \ldots, e_{jm} >$

- Une **coupure C** est dite **cohérente** (ou consistante) si elle est fermée vis à vis du passé de ses éléments :

  $\Rightarrow$ tout élément dans le passé d'un événement de la coupure appartient lui-même à la coupure

Exercice

Dans l’exemple suivant, les coupures C et C’ sont-elles cohérentes ?



1.2. **Caractérisation des coupures cohérentes par l'estampillage vectoriel de Lamport**

- On associe à une coupure C la suite des événements
  $< e_1, \ldots, e_j , \ldots e_n >$ (e_j est l'événement le plus récent du site S_j appartenant à la coupure).

- Associons à la coupure C l'estampille vectorielle EV(C) définie par :
  $EV(C) = <EV(e_1)\ [1], \ldots, EV(e_j)\ [j] , \ldots\ EV(e_n)\ [n] >$

- La **coupure est cohérente** si et seulement si :

  $EV(C) = \sup (EV(e_1), \ldots EV(e_j), \ldots EV(e_n))$, avec :

  $\sup (EV(e_1), \ldots EV(e_j), \ldots EV(e_n))\ [k] = (\sup (EV(e_1)\ [k], ..., \sup (EV(e_j)\ [k], ... \sup (EV(e_n)\ [k]$

- Exercice : appliquer ce résultat aux coupures C et C' de l'exemple précédent

2. **Hypothèses sur les canaux de communication**

2.1. Définition des types de messages

Soit c un canal, m et m1 deux messages empruntant ce canal.

On dit que *m double m1* si et seulement si :
émission (m1) $\rightarrow$ émission (m) et réception (m) $\rightarrow$ réception (m1)

Quatre types possibles de messages peuvent être définis
selon les contraintes de doublement :

i) Un message m est de type *marqueur* s'il ne peut ni doubler ni être doublé par aucun message transitant sur le même canal :

$\forall$ *m1, émission (m1) $\rightarrow$ émission (m)* $\Rightarrow$ *réception (m1) $\rightarrow$ réception (m)*
et $\forall$ *m2, émission (m) $\rightarrow$ émission (m2)* $\Rightarrow$ *réception (m) $\rightarrow$ réception (m2)*

2.1. Définition des types de messages

ii) Un message m est de type *ct_passé* (contraint par son passé) s'il ne peut doubler aucun message :

$\forall$ *m1,* *émission (m1)* $\rightarrow$ *émission (m)* $\Rightarrow$ *réception (m1)* $\rightarrow$ *réception (m)*

iii) Un message m est de type *ct_futur* (contraint par son futur) s'il ne peut être doublé par aucun message :

$\forall$ *m2,* *émission (m)* $\rightarrow$ *émission (m2)* $\Rightarrow$ *réception (m)* $\rightarrow$ *réception (m2)*

iv) Un message m de type *ordinaire* n'impose pas de conditions de réception :

- il ne peut pas doubler les messages *marqueur* et *ct_futur*
- il ne peut pas être doublé par les messages *marqueur* et *ct_passé*

2.1. Définition des types de messages

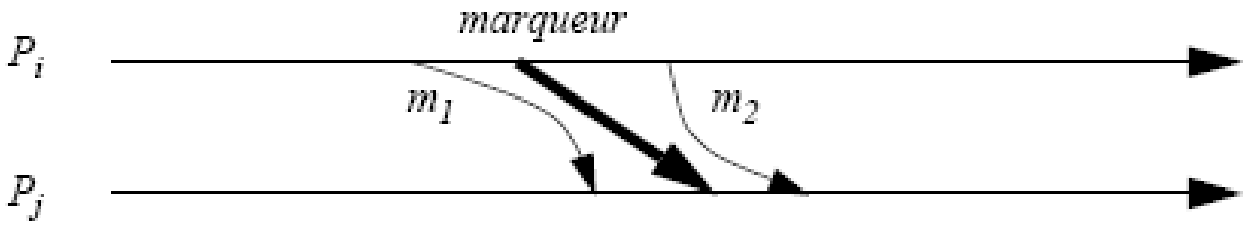


Figure 3. Livraison d'un marqueur.

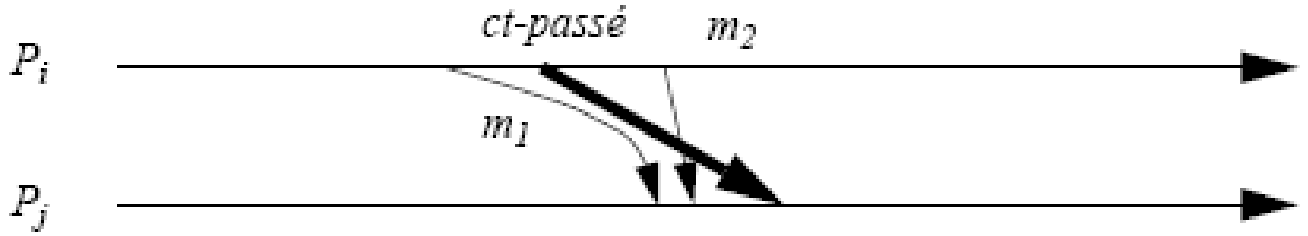


Figure 4. Livraison d'un message contraint par son passé.

(tous les messages émis avant lui sont reçus avant lui)

2.1. Définition des types de messages

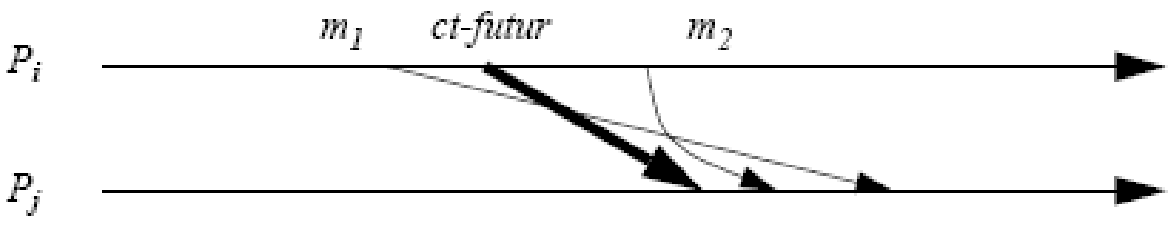


Figure 5. Livraison d'un message contraint par son futur.
(tous les messages émis après lui seront reçus après lui)

Les types de messsages définissent une gamme de comportement du canal :

- *le moins contraint (tous les messages sont ordinaires)*
- *le plus contraint (tous les messages sont de type marqueur) ⇒ canal FIFO*

Pour un canal donné, les trois propriétés A1, A2 et A3 suivantes sont équivalentes :

- *A1 : tous les messages sont de type ct_passé*
- *A2 : tous les messages sont de type ct_futur*
- *A3 : tous les messages sont de type marqueur*

2.2. Hypothèses globales sur les canaux de communication

Rappel : la propriété d'ordre causal porte sur l'ensemble des canaux.

$\forall P_i, P_j, P_k$, $\forall m$ émis sur c_{ij} , $\forall m_1$ émis sur c_{kj} :

$émission_i(m) \rightarrow émission_k(m_1) \Rightarrow réception_j(m) \rightarrow réception_j(m_1)$

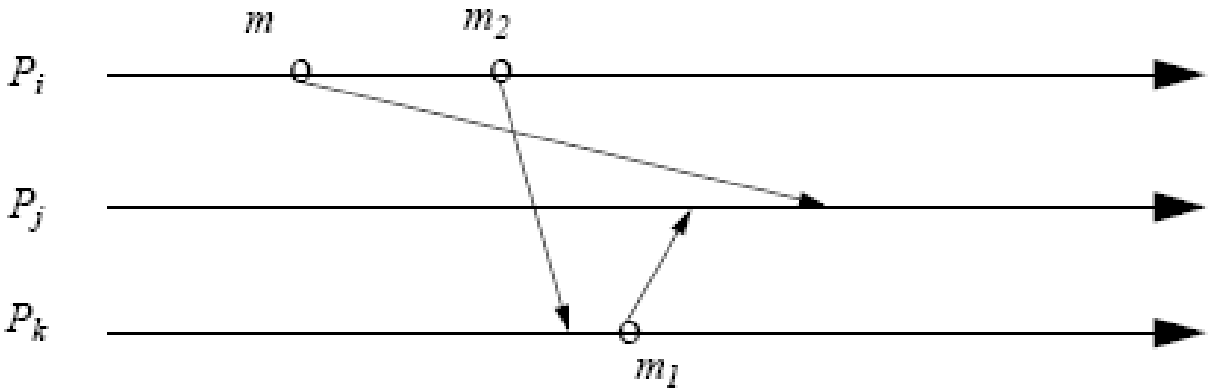


Figure 6. Réception non causale.

Remarque : si la propriété d'ordre causal est garantie alors chaque canal a un comportement FIFO (réciproque fausse)

2.3. Superposition

- A chaque processus Pi est associé un processus observateur CTLi : ce processus peut lire l'état de Pi.
  - La réception d'un message par Pi est réalisée par CTLi, qui le délivre ensuite à Pi ;
  - Pi confie à CTLi les messages qu'il veut transmettre vers tout autre processus Pj :
    - CTLi se charge alors de son émission vers CTLj
- Les processus CTLi sont chargés de capter un état global cohérent de l'application.
- Ils coopèrent entre eux à l'aide de messages de contrôle.
- Un tels schéma d'observation des processus Pi par des contrôleurs CTLi est appelé *superposition*.

3. Algorithme de Chandy et Lamport

3.1. Principe

- Hypothèse : **canaux FIFO**

*m*1 < *m*2 ⇔ *émission (m*1*)* → *émission (m2)*
⇔ *réception (m*1*)* → *réception (m2)*

- Tout message m définit deux sous-ensembles disjoints :
  - messages émis avant m, noté ***<m***
  - messages émis après m, noté ***>m***

- **Considérons l'enregistrement de l'état local el_i du processus Pi**

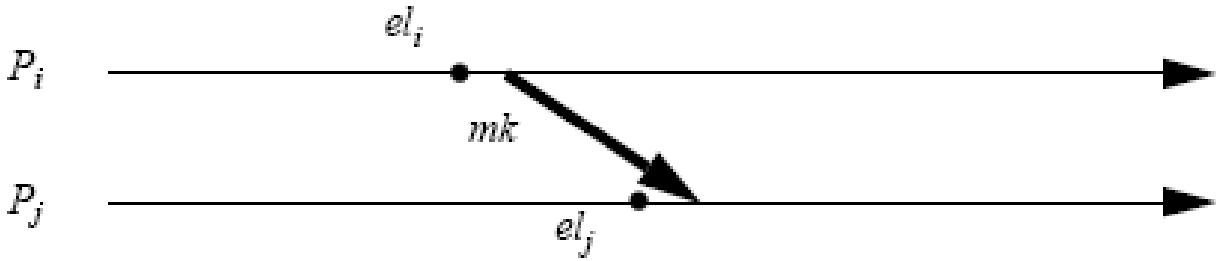


Figure 7. Echange d'un message de contrôle mk.

3.1. Principe de l'algorithme de Chandy et Lamport

- Pour assurer la cohérence d'un couple d'états locaux (el_i, el_j) (conditions C_1 et C_2), seules les réceptions de messages émis par P_i avant l'enregistrement de el_i doivent être captées dans el_j.

$\Rightarrow$ Le processus P_i émet, lorsqu'il enregistre son état el_i, un **message de contrôle (appelé *mk*)** sur le canal c_{ij}.

$\Rightarrow$ le processus P_j doit alors enregistrer son état local el_j au plus tard à la réception de ce message *mk*.

$\Rightarrow$ en effet, par définition :

$<mk = \{m \mid émission_i(m)$ captée dans $el_i\}$

$>mk = \{m \mid émission_i(m)$ non captée dans $el_i\}$

- *De plus, el_j étant enregistré au plus tard à la réception du message mk :*

  $<mk \supseteq \{m \mid réception_j(m)$ captée dans $el_j\}$

- *D'où :*

$\{m \mid réception_j(m)$ captée dans $el_j\} \subseteq \{m \mid émission_i(m)$ captée dans $el_i\}$

$\Rightarrow$ *ce qui assure le respect des conditions C_1 et C_2.*

3.1. Principe de l'algorithme de Chandy et Lamport

Principe du fonctionnement :

Chaque processus P_j enregistre son état local el_j et l'état de ses canaux entrants en observant les deux règles suivantes :

R_1 : un processus P_j qui capte son état local envoie sur tous ses canaux de sortie un message de contrôle (appelé mk) pour signifier à ses voisins qu'il a enregistré son état local.

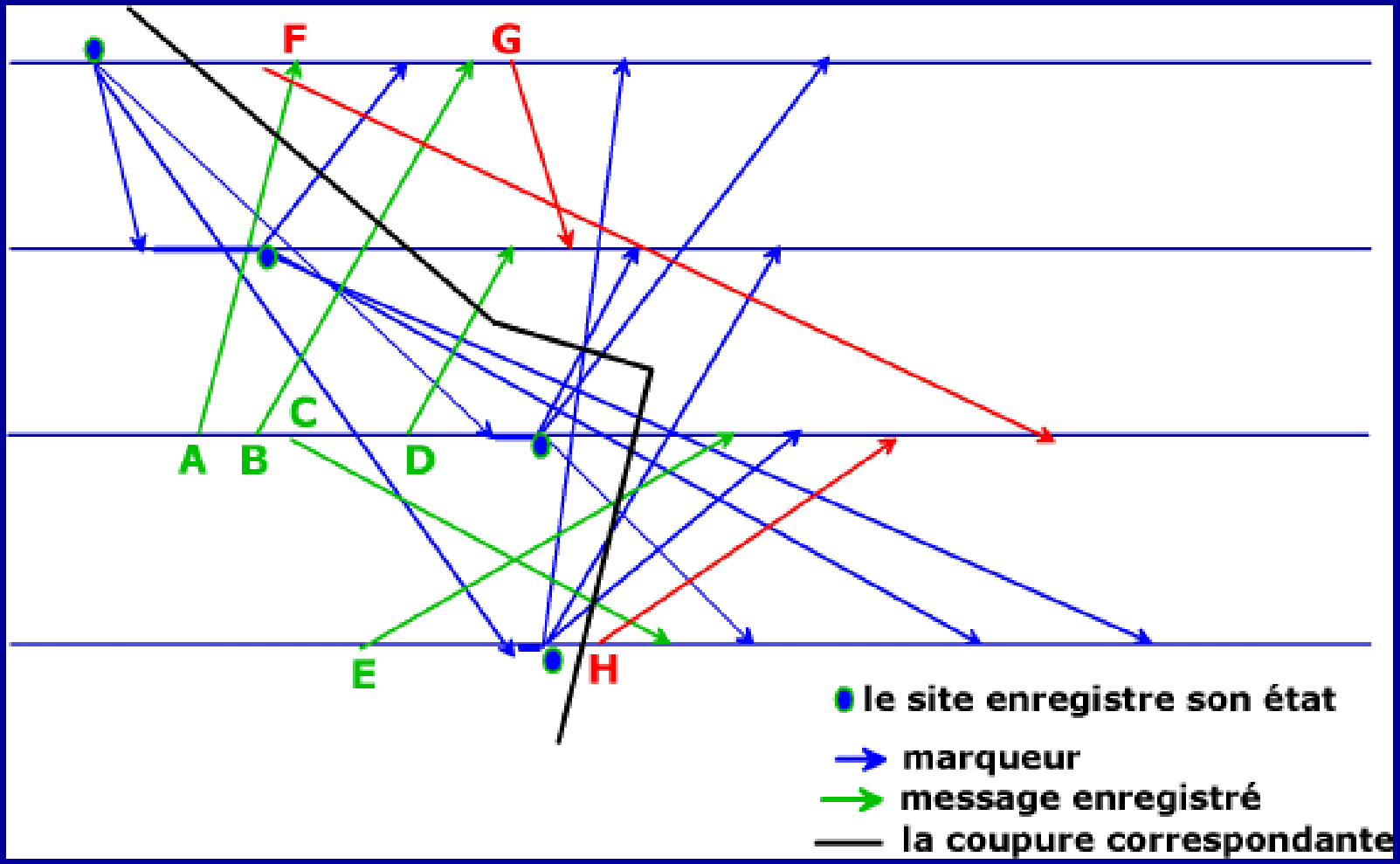
R_2 : à la réception d'un message mk sur un de ses canaux d'entrée, par exemple c_{ij} , un processus P_j peut se trouver dans l'une des deux situations suivantes :

- P_j n'a pas encore capté son état local : il doit alors enregistrer cet état (R_1) $\Rightarrow$ l'état ec_{ij} du canal c_{ij} est vide car P_j a reçu tous les messages émis avant le message mk.

- P_j a déjà enregistré son état local (soit de sa propre initiative, soit suite à une réception antérieure d'un message mk sur *un autre canal d'entrée*) $\Rightarrow$ l'*état du canal* c_{ij} est constitué des messages reçus sur ce canal après que Pj ait enregistré son état local et avant qu'il ne reçoive mk.

3.2. Exemple de l'algorithme de Chandy et Lamport

Exemple de fonctionnement :



3.3. Algorithme de Chandy et Lamport

Chaque processus P_j du système est doté de k_j canaux d'entrée
et de l_j canaux de sortie :

- **c_in_j** : [1..k_j] canaux d'entrée
- **c_out_j** : [1..l_j] canaux de sortie

L'observateur CTL_j associé au processus P_j, a accès à ces canaux et est doté des variables suivantes :

- el_j : état local
- ec_j : [1..k_j] files de messages
- $reçu_j$: [1..k_j] booléens initialisés à faux
- $enreg\_état_j$: booléen initialisé à faux

- el_j et $ec_j[x]$ mémorisent respectivement l'état local et l'état du canal $c\_in_j[x]$;
- $reçu_j$ [x] passe à vrai lorsqu'un message mk est reçu sur le canal $c\_in_j[x]$;
- $enreg\_état_j$ sert à indiquer que le processus P_j a enregistré son état local.

3.3. Algorithme de Chandy et Lamport

Procédure exécutée par CTL_j lorsque l'état local de P_j est capté :

Procédure **capter_état_local**
début
$el_j \leftarrow$ état local du processus P_j ;
$enreg\_état_j \leftarrow$ vrai ;
$\forall\ x : 1 \leq x \leq k_j : ec_j[x] \leftarrow \varnothing$;
$\forall\ y : 1 \leq y \leq l_j$: envoyer mk sur $c\_out_j[y]$
fin

Procédure exécutée par CTL_j lorsque la participation de P_j à l'état global est calculée :

Procédure **fini**
début
si $\forall\ x : 1 \leq x \leq k_j : reçu_j[x]$ alors
stocker (el_j , $ec_j\ [1..k_j]$) chez le processus prévu à cet effet
fsi
fin

3.3. Algorithme de Chandy et Lamport

Le comportement de CTL_j est exprimé par les deux énoncés suivants :

⇒ lors de la décision locale de calculer un état global

début
si ¬ enreg_état$_j$ alors capter_état_local fsi
fin

⇒ lors de la réception de m sur c_in$_j$[x]

début
si m=mk alors
    si ¬ enreg_état$_j$ alors capter_état_local fsi ;
    reçu$_j$[x] ← vrai ;
    fini ;
sinon
    si enreg_état$_j$ et ¬ reçu$_j$[x] alors
      ec$_j$[x] ← ec$_j$[x] ⊕ m ; /* ajout de m en queue de ec$_j$[x] */
    fsi ;
    délivrer m au processus P_j du calcul observé
fsi
fin

4. Solutions pour canaux non FIFO

- Rappel : *contrainte de non-perturbation*

- Si les canaux sont FIFO, l'utilisation des messages de contrôle *mk* dans l'algorithme de Chandy et Lamport permet de séparer l'ensemble des messages émis avant l'enregistrement de l'état local du processus émetteur de l'ensemble des messages émis après

- Si les canaux ne sont plus FIFO la délimitation ne peut pas se faire sans perturber l'ordre des messages.

  $\Rightarrow$ techniques n'utilisant pas de messages de contrôle explicites (Lai Yang, Mattern).

  $\Rightarrow$ solutions utilisant des messages de contrôle (Ahuja)

4.1. Solution sans messages de contrôle explicites

Méthode cumulative (mais rapide) de Lai et Yang

Cet algorithme est basé sur l'utilisation de deux couleurs, symbolisant
- pour les processus : *l'avant* (vert) et *l'après* (rouge) enregistrement de l'état local
- pour les messages : la couleur de leur émetteur.

Chaque processus obéit aux règles suivantes :

R1 : un processus qui n'a pas encore enregistré son état est coloré en vert, et tous les messages qu'il émet sont colorés en vert.

R2 : lorsqu'on processus enregistre son état local, il devient rouge et tous les messages qu'il émet ensuite sont colorés en rouge.

R3 : un processus vert peut à tout moment enregistrer son état local, et au plus tard lorsqu'il reçoit un message rouge.

R4 : les messages verts émis et reçus sur chaque canal sont stockés (aspect cumulatif de l'algorithme) .

R5 : chaque processus rouge transmet à un même collecteur sont état local et l'ensemble des messages (verts) qu'il a stockés*****

*dans msg_émisi (Cf. ci-dessous) : $\forall j \neq i$, P_i envoie à Pc les messages stockés dans $msg\_émis_i[j]$ (qui est vidé)

4.1.1. Méthode cumulative (mais rapide) de Lai et Yang

Un état global cohérent est calculé par le processus collecteur lorsque ce dernier a reçu les informations de tous les processus participants :

L'état ec_{ij} d'un canal cij est la différence entre l'ensemble des messages verts émis par P_i vers P_j , noté $msg\_émis_i[j]$, et l'ensemble des messages verts reçus par P_j depuis P_i, noté $msg\_reçus_j[i]$.

Preuve : montrer que les conditions C1 et C2 sont bien vérifiées

Remarques :

- Par rapport à l'algorithme de Chandy et Lamport, aucun message de contrôle supplémentaire n'est nécessaire.
- Par contre le processus observateur CTL_i doit observer les envois et réceptions de messages effectués par P_i de manière continue (aspect cumulatif) $\Rightarrow$ nombre de messages stockés important

Exercice : Cf. feuille d'exercices du chapitre 5.

4.1.2. Méthode non cumulative (mais lente) de Mattern

Principe :

- L'inconvénient de l'algorithme de Lai et Yang (aspect cumulatif) disparaît dans l'algorithme de Mattern, qui utilise des compteurs et des messages supplémentaires pour éviter le stockage des ensembles de messages
  $\Rightarrow$ plus grand délai pour l'obtention de l'état global

- Cet 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 un processus P_j sait que les messages verts qu'il a reçu après son enregistrement sont des messages en transit sur le canal, mais il ne sait pas s'il les a tous reçus (car un message rouge peut très bien avoir doublé des messages verts : canaux non FIFO).

- On va donc utiliser des compteurs de messages : 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 messages en transit sur l'ensemble des canaux est alors : $mt = \sum i\ d_i$

4.1.2. Méthode non cumulative (mais lente) de Mattern

Algorithme 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 sont état local et son d_i.

R'5 : un processus rouge transmet à Pc tous les messages verts qu'il reçoit.

4.1.2. Méthode non cumulative (mais lente) de Mattern

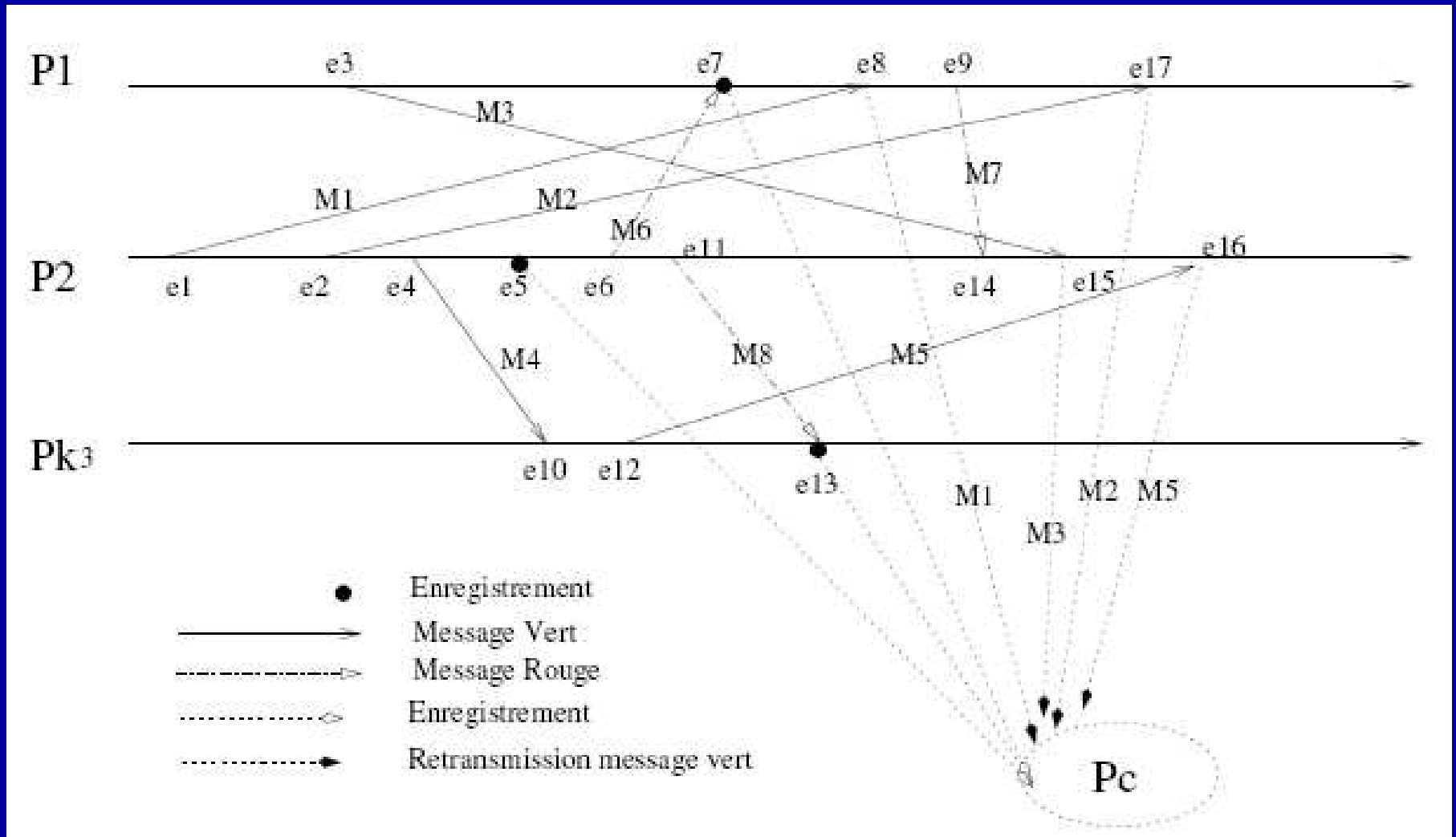
Algorithme de Mattern :

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 émetteur P_i, site récepteur P_j)) : (en provenance donc de P_j) : il rajoute M à c_{ij} et décrémente mt
- lorsque N = 0 et mt = 0, Pc dispose de tous les états locaux et de tous les messages en transit : il a donc l'état global.

Conclusion : dans ces deux algorithmes, le processus collecteur Pc est indispensable pour calculer l'état global $\Rightarrow$ aspect **centralisé**

Exemple de scénario :



4.2. Autres solutions

4.2.1. Solutions avec des messages de contrôle explicites

- solutions **décentralisées** qui adaptent l'algorithme de Chandy et Lamport
- basées sur l'utilisation de messages de contrôle - marqueurs - de type CT_PASSE ou CT_FUTUR
- ces messages vont permettre de séparer l'ensemble des messages émis AVANT l'enregistrement et l'ensemble des messages émis APRES ⇒ *algorithme d'Ahuja*.

4.2.2. Solutions fondées sur l'ordre causal

Rappel : l'ordre causal sur un ensemble de canaux est une propriété plus forte que les canaux FIFO

⇒ on peut définir des algorithmes plus simples que Chandy et Lamport
⇒ *algorithmse de Acharya et Badrinah* et d'Alagar et Venkatesan