Université de Gabés
Faculté des Sciences de Gabés

Licence en génie logiciel et systèmes d'information
Niveau: 3eme année
Semestre 1

Chapitre 7
Problèmes fondamentaux de la répartition

Dhikra KCHAOU
dhikrafsegs@gmail.com

2023-2024

Plan du chapitre

1. Introduction
2. Système synchrone / Asynchrone
3. État et horloge globales
4. Temps logique
5. Horloge de Lamport
6. Horloge de Mattern & Fidge

Introduction

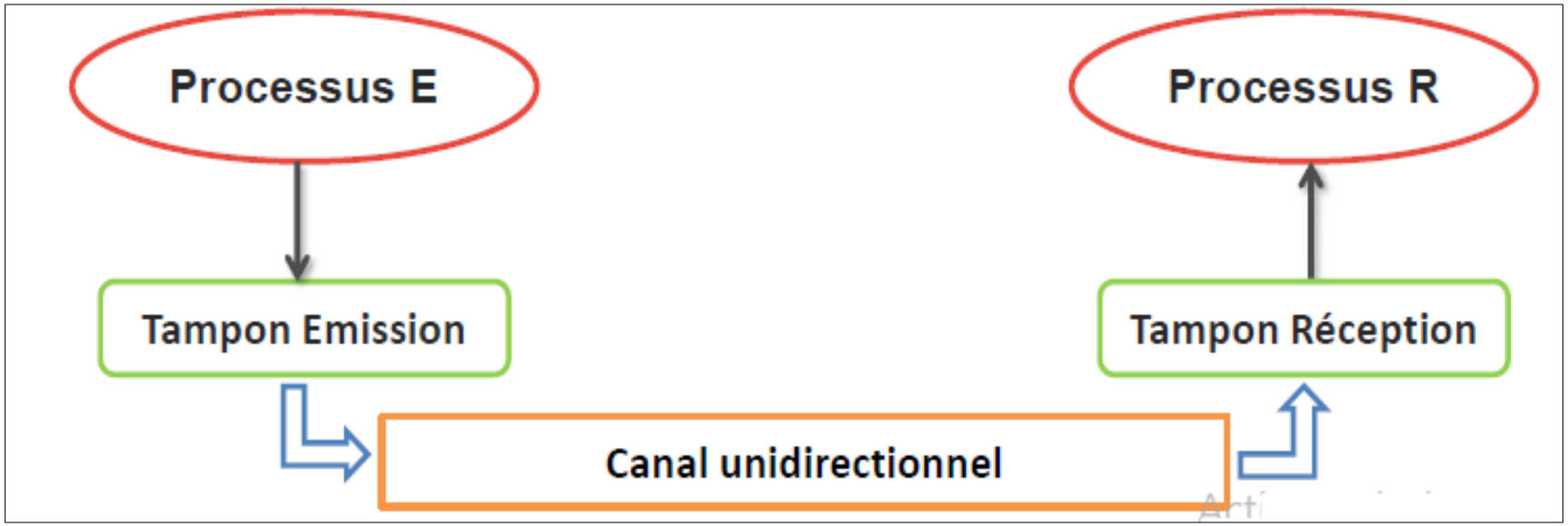
- Dans le premier chapitre, nous avons exprimé les différences entre un système centralisé et un système réparti:
  - **Architecture matérielle**: Processeur/ Horloge/ mémoire
  - **Evénements**: totalement ordonnées/ partiellement ordonnées
  - **État** : unique et global/ absence d'état global
- **Contraintes posées par les systèmes réparties**:
  - Pas de mémoire commune (support habituel de l'état)
  - Pas d'horloge commune (qui définit le séquencement des événements).
  - Asynchronisme des communications : Pas de borne supérieure sur le temps de transit d'un message.
  - Asynchronisme des traitements : Pas de borne sur le rapport relatif des vitesses d'exécution sur les deux sites.

Introduction

- Dans ce chapitre, nous allons étudier les concepts de base pour la gestion des processus en mode réparti: ordonnancement, synchronisation, interblocage, … On parlera de **programmation répartie** ou **algorithmique répartie**.
- **Algorithmique distribuée**:
  - est le développement d'algorithmes dédiés aux systèmes distribués et prenant en compte les spécificités de ces systèmes (Horloge globale, état global, diffusion causale, consensus ...)
  - s'intéresse principalement à deux grandes familles de problèmes
    - Synchronisation et coordination entre processus distants
    - Détection de la cohérence globale d'un système dans un contexte non fiable (crash de processus, perte de messages ...)

Système distribué

- **Ensemble d'éléments logiciels s'exécutant en parallèle** : On parlera de processus pour ces éléments logiciels.
- **Ces éléments communiquent via des canaux de communication:** Liaison entre un processus et un canal.



Processus

- Élément logiciel effectuant une tâche, un calcul
  - Exécution d'un ensemble d'instructions
  - Une instruction correspond à un événement local au processus dont les événements d'émission et de réception de messages
    - Les instructions sont généralement considérées comme atomiques
  - Il possède une mémoire locale
  - Il possède **un état local** (Ensemble de ses données et des valeurs de ses variables locales)
  - Il possède un identifiant qu'il connaît
  - Pas de connaissance sur l'état des autres processus
  - Les processus d'un système s'exécutent en parallèle

Canal de communication

- **Canal logique de communication point à point** est utilisé pour la communication entre 2 processus afin de transiter des messages.
- **Caractéristiques d'un canal :**
  - Uni ou bidirectionnel.
  - Fiable ou non : perd/modifie ou pas des messages.
  - Ordre de réception par rapport à l'émission: Exemple : FIFO = les messages sont reçus dans l'ordre où ils sont émis.
  - Synchrone ou asynchrone :
    - Synchrone : l'émetteur et le récepteur se synchronisent pour réaliser l'émission et/ou la réception
    - Asynchrone : pas de synchronisation entre émetteur et récepteur
  - Taille des tampons de message des deux cotés émetteur et récepteur : Limitée ou illimitée

Canal de communication

- Modèle généralement utilisé d'un canal: Fiable, FIFO, tampon de taille illimitée, asynchrone (en émission et réception) et bidirectionnel.
- **Exemple : modèle des sockets TCP:**
  - Fiable.
  - FIFO.
  - Bidirectionnel.
  - Synchrone pour la réception.
    - On reçoit quand l'émetteur émet.
  - Asynchrone en émission.
    - Emetteur n'est pas bloqué quand il émet

Système synchrone / Asynchrone

- **Un modèle synchrone est un modèle où les contraintes temporelles sont bornées :**
  - On sait qu'un processus évoluera dans un temps borné.
  - On sait qu'un message arrivera en un certain délai.
- **Un modèle asynchrone n'offre aucune borne temporelle :**
  - Modèle bien plus contraignant et rendant impossible ou difficile la réalisation de certains algorithmes distribués.
  - Exemple : on ne sait pas différencier en asynchrone:
    - Le fait qu'un processus est lent ou est planté.
    - Le fait qu'un message est long à transiter ou est perdu.

Ordonnancement : Ordre et temps

- **Objectif de l'ordonnancement** : définir un ordre entre les événements d'un système qui est nécessaire pour:
  - Suivre l'évolution de l'état des processus et du système.
  - Synchroniser les processus
  - Prendre des décisions: Allocation des ressources (concurrence), préemption, …
- **Ordonnancement dans le cas d'un système centralisé**:
  - L'ordonnancement est toujours possible:
    - Horloge physique unique : dates différentes pour deux évènements:

      (e1,t1) < (e2,t2)
    - Evènements totalement ordonnées: relation de précédence sur les événements = relation d'ordre total.
- **Ordonnancement dans le cas d'un système réparti**:
  - L'ordonnancement n'est pas possible
  - Difficulté de définir un temps unique

État et horloge globales

- Pour chaque processus du système, possède **un 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*
- **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

➡ Solution: Définir un temps global cohérent et « identique » (ou presque) pour tous les processus : **Créer un temps logique.**

Temps logique

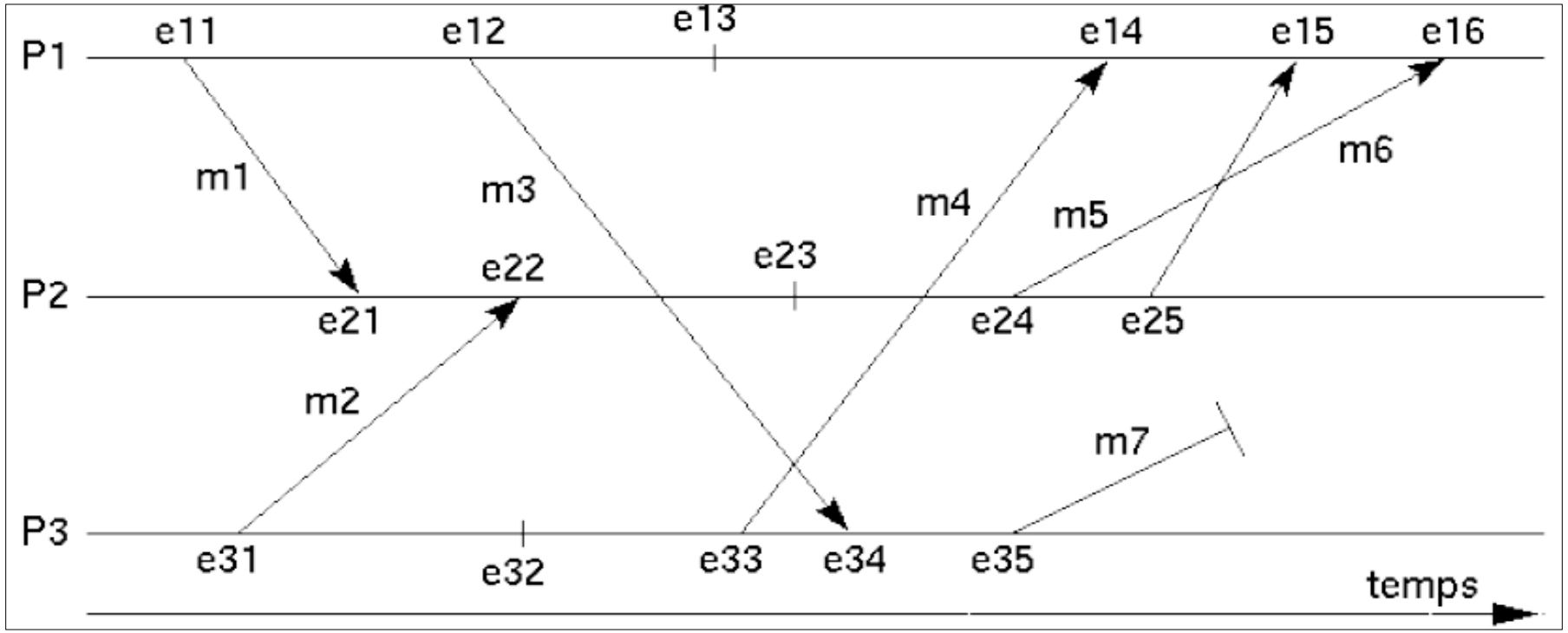
- **Temps logique** :
  - Temps qui n'est pas lié à un temps physique
  - Son but est de pouvoir **préciser l'ordonnancement de l'exécution des processus** et de leur communication
  - En fonction des événements locaux des processus, des messages envoyés et reçus, on créé **un ordonnancement logique**
- Deux approches principales
  - **Horloge de Lamport** : méthode par estampille
  - **Horloge de Mattern** : horloge vectorielle

Temps logique : chronogramme

- Un 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
  - **Événement interne** dans l'évolution du processus

Temps logique : chronogramme

- Trois processus tous reliés entre-eux par des canaux
- Temps de propagation des messages quelconques et possibilité de perte de message.



Temps logique : chronogramme

- **Exemples d'événements**:
  - **Processus P1**
    - e11 : événement d'émission du message m1 à destination du processus P2
    - e13 : événement interne au processus
    - e14 : réception du message m4 venant du processus P3
  - **Processus P2** : message m5 envoyé avant m6 mais m6 reçu avant m5
  - **Processus P3** : le message m7 est perdu par le canal de communication
- **Règle de numérotation d'un événement:**
  - e_{xy} avec **x** le numéro du processus et **y** le numéro de l'événement pour le processus, dans l'ordre croissant

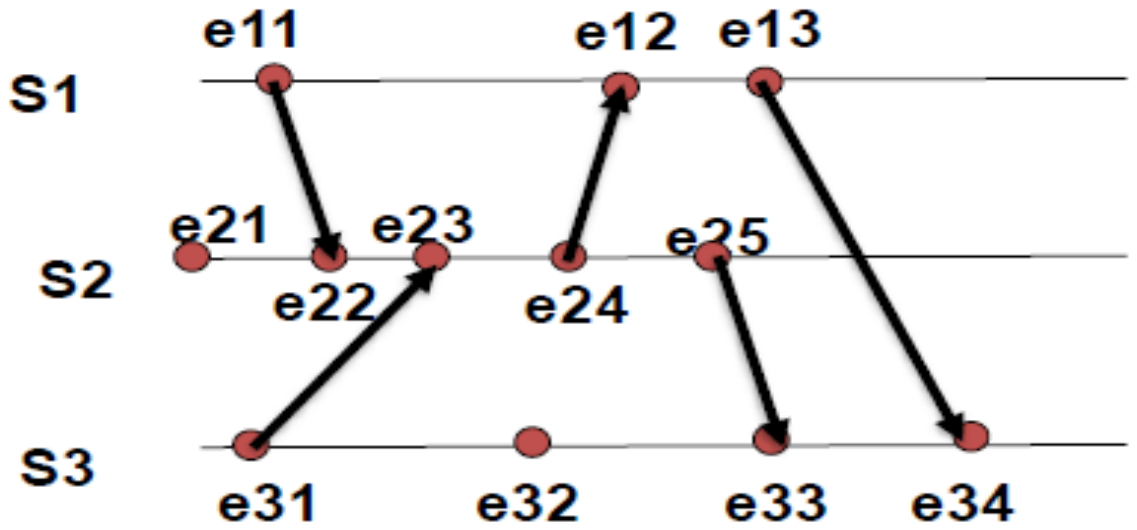
Temps logique : dépendance causale

- **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 précède e’)**
  - e doit se dérouler avant e'
  - Si **e → e**', alors une des trois conditions suivantes doit être vérifiée pour e et e' :
    1. Si e et e' sont des événements d'un même processus, e précède localement e'
    2. Si e est l'émission d'un message, e' est la réception de ce message
    3. Il existe un événement f tel que e → f et f → e‘

Temps logique : dépendance causale

- Exemple de dépendances causales
- Quelques dépendances causales autour de e12
  - Localement : e11 → e12, e12 → e13
  - Sur message : e12 → e34
  - Par transitivité : e12 → e35 (car e12 → e34 et e34 →e35) (e11 → e13)
- Dépendance causale entre e12 et e32 ?
  - A priori non : absence de dépendance causale
- Des événements non liés causalement se déroulent en parallèle
  - Relation de parallélisme : ||
  - e || e' $\Longleftrightarrow$ ((e → e') V (e' → e))
  - **Parallélisme logique** : ne signifie pas que les 2 événements se déroulent simultanément mais qu'il peuvent se dérouler dans n'importe quel ordre

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,e32,e23,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.
- Le but d'une horloge logique (selon le type d'horloge)
  - Créer un **ordre total global** sur tous les événements de tous les processus
  - Déterminer si un événement a eu lieu avant un autre ou s'il n'y a pas de dépendances causales entre eux
  - Vérifier que des propriétés d'ordre sur l'arrivée de messages sont respectées
- **Horloge logique**
  - Fonction H(e) : associe une date à chaque événement
  - Respect des dépendances causales :
    - e → e' ⟹ H(e) < H(e')

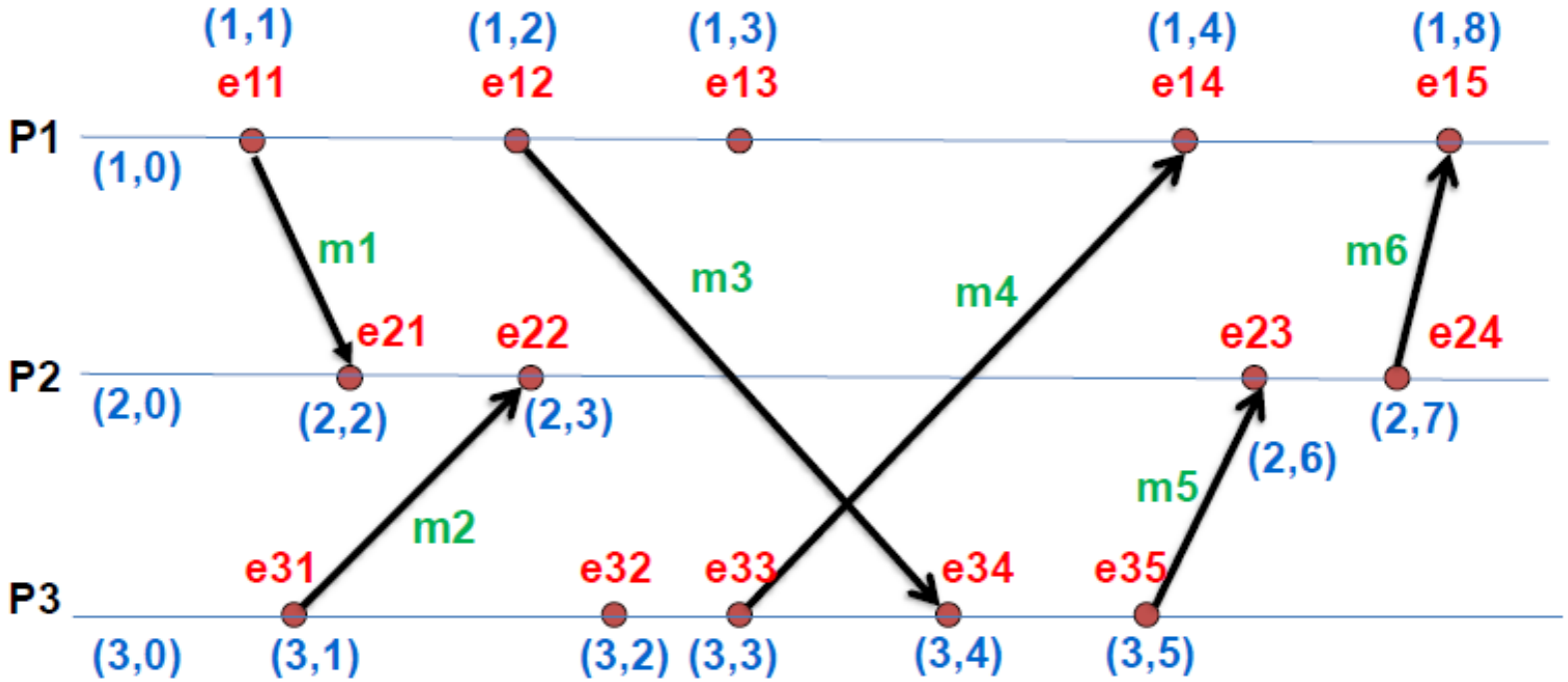
Horloge de Lamport 1978

- A chaque événement *e*, une date H(e) (**estampille**) renvoie un couple (s, nb)
  - s : numéro du processus
  - nb : numéro d'événement
  - Unicité des dates : pas le même couple (s,nb) pour deux événements différents
- Implique que sur un même processus, deux événements différents n'ont pas la même estampille
- Deux dates sont toujours ordonnées
  - H(e) < H(e') => (e.nb < e'.nb) ou ((e.nb=e'.nb) et (e.s < e'.s))
- Horloges de Lamport respectent les dépendances causales
  - e → e' => H(e) < H(e')
  - La réciproque n'est pas vraie
    - H(e) < H(e') => ¬ (e' → e)
    - C'est-à-dire : soit e → e', soit e || e'

Horloge de Lamport : Création du temps logique:

- Localement, chaque processus Pi possède une horloge locale logique (**estampille**) Hi, initialisée à 0 qui sert à dater les événements
- Pour chaque événement local de Pi
  - Hi = Hi + 1 : on incrémente l'horloge locale
  - L'événement a pour date (i,Hi)
- Lors de l'émission d'un message par Pi
  - On incrémente Hi de 1 puis on envoie le message en envoyant la date (i, Hi) de l'événement d'émission avec le message
- Réception d'un message *m* avec la date (s, nb)
  - Hi = max(Hi, nb) +1 et marque l'événement de réception avec (i,Hi)

Exemple : chronogramme avec ajouts des estampilles



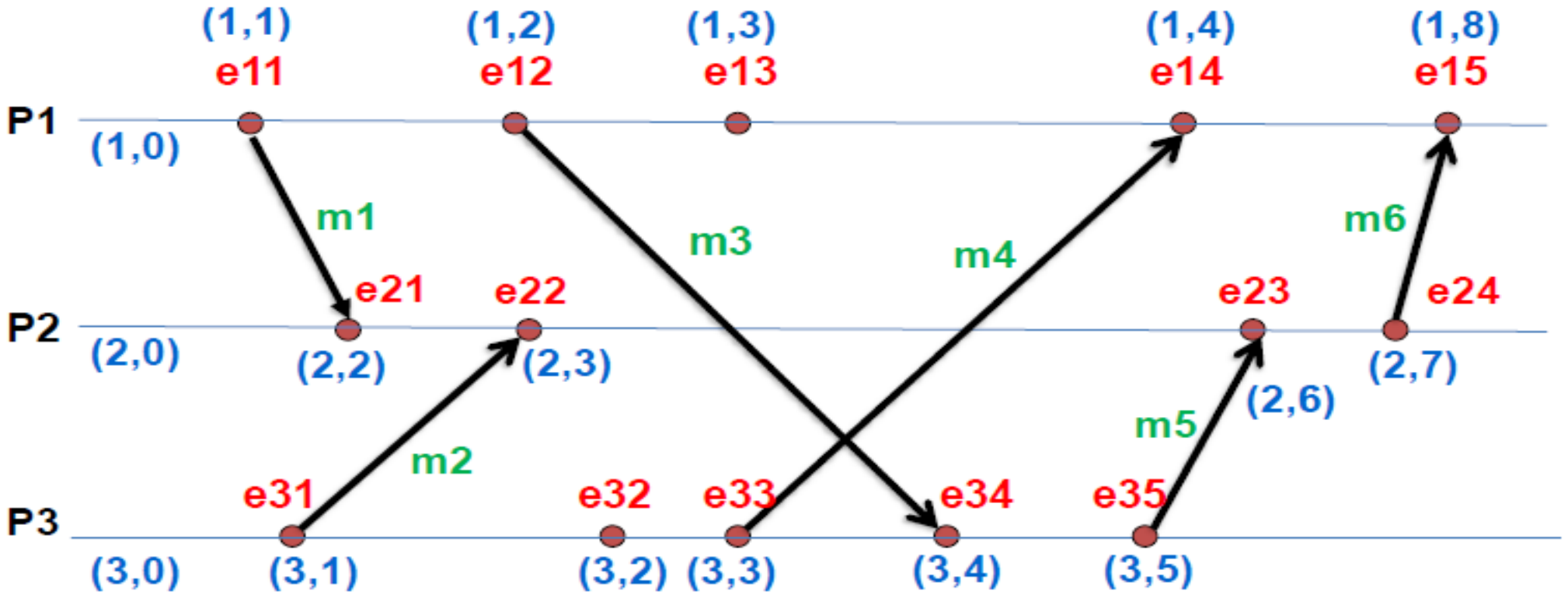
- Date de e23 : 6 car le message m5 reçu avait une valeur de 5 et l'horloge locale est seulement à 3
- Date de e34 : 4 car on incrémente l'horloge locale vu que sa valeur est supérieure à celle du message m3
- Pour e11, e12, e13 ... : incrémentation de +1 de l'horloge locale

Horloge de Lamport

- L'intérêt des horloges de Lamport est de créer un ordre global total entre tous les événements du système
  - Ordre qui respecte les dépendances causales entre événements
  - Pour deux événements indépendants causalement, un choix arbitraire est fait entre les deux
    - Ne pose pas de problème car pas de contraintes de dépendances à respecter
- **Ordre total**, noté **e << e'** : *e* s'est déroulé avant *e'*
  - Soit *e* un événement de *Pi* et *e' un* événement de *Pj* :
    - e << e' => Hi(e) < Hj(e')
  - 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'estampille, **le numéro du processus est utilisé pour les ordonner**)

Horloge de Lamport

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 ...

Limites de l'horloge de Lamport

- Elle 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′* ?
  - Ou aucune dépendance causale ?
- Exemple : H(e32) = 2 et H(e13) = 3 mais on a pas e32→ e13

Horloge de Mattern & Fidge, 1989-91 :

- Horloge qui assure la réciproque de la dépendance causale

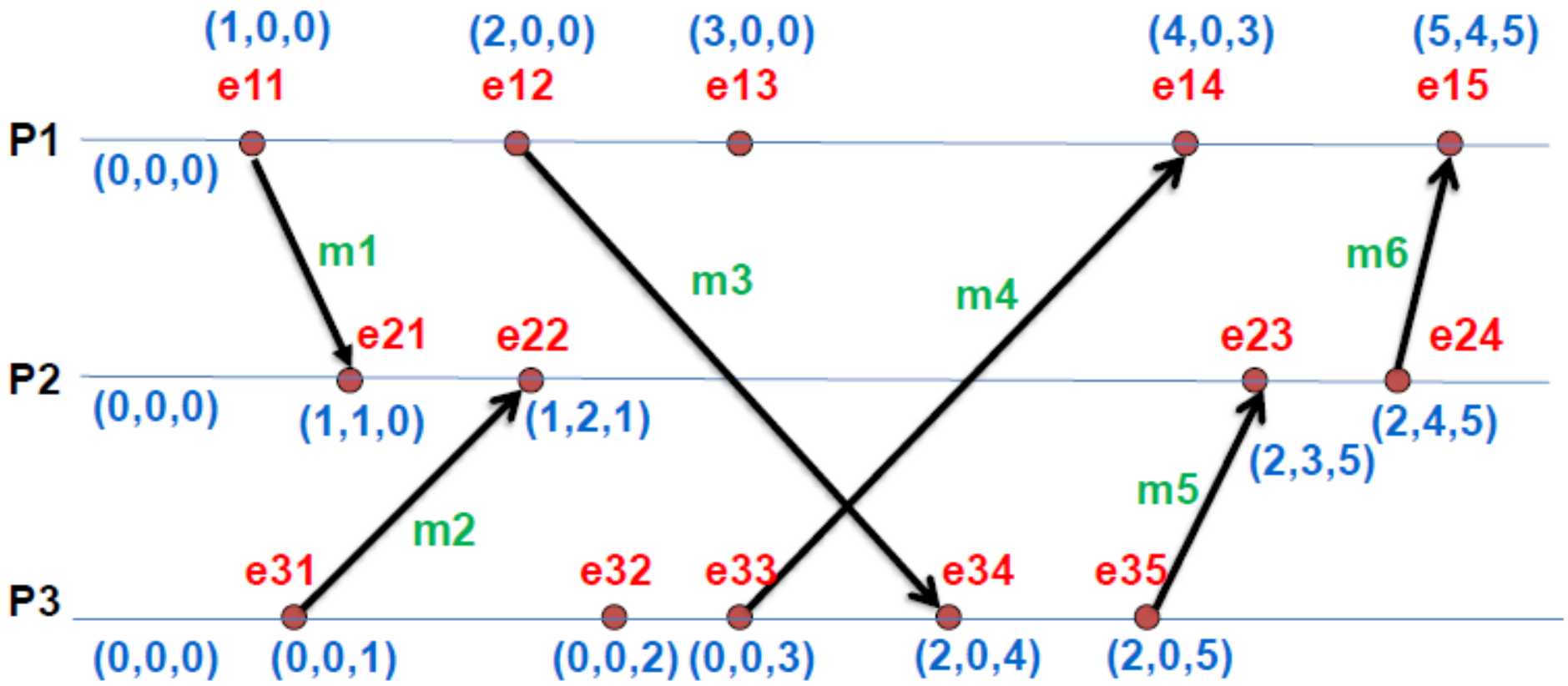
$$H(e) < H(e') \Rightarrow e \rightarrow e'$$

- Permet également de savoir si 2 événements sont parallèles (non dépendants causalement)
- Ne définit par contre pas un ordre total global
- **Principe** :
  - Chaque événement est là aussi daté par une horloge
  - Date (estampille) = 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*

Horloge de Mattern & Fidge

- **Fonctionnement de l'horloge**:
  - Initialisation : pour chaque processus *Pi*, Vi =(0, ... , 0)
  - Pour un processus *Pi*, à chacun de ses événements (local, émission, réception) :
    - Vi [i] = Vi [i] + 1
    - **Incrémentation du compteur local d'événement**
    - Si émission d'un message, alors *Vi* est envoyé avec le message
- Pour un processus *Pi*, à la réception d'un message *m* contenant un vecteur *Vm*, on met à jour les cases j ≠ i de son vecteur local *Vi*
  - ∀ j : Vi [j] = max (Vm [j], Vi [j])
    - 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

Exemple : chronogramme d'application horloge de Mattern



Horloge de Mattern

- **Relation d'ordre partiel sur les dates**
  - V ≤ V' défini par ∀ i : V[i] ≤ V'[i]
  - V < V' défini par V ≤ V' et ∃ j tel que V [j] < V' [j]
  - V || V' défini par non (V < V') et non(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') => e → e‘ : Si deux dates sont ordonnées, on a forcément une dépendance causale entre les événements datés
  - V(e) || V(e') => e || e‘ : Si il n'y a aucun ordre entre les 2 dates, les 2 événements sont indépendants causalement

Horloge de Mattern

- Retour sur l'exemple
  - 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 e14 → 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 détecte **les indépendances causales**

Horloge de Mattern

- 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

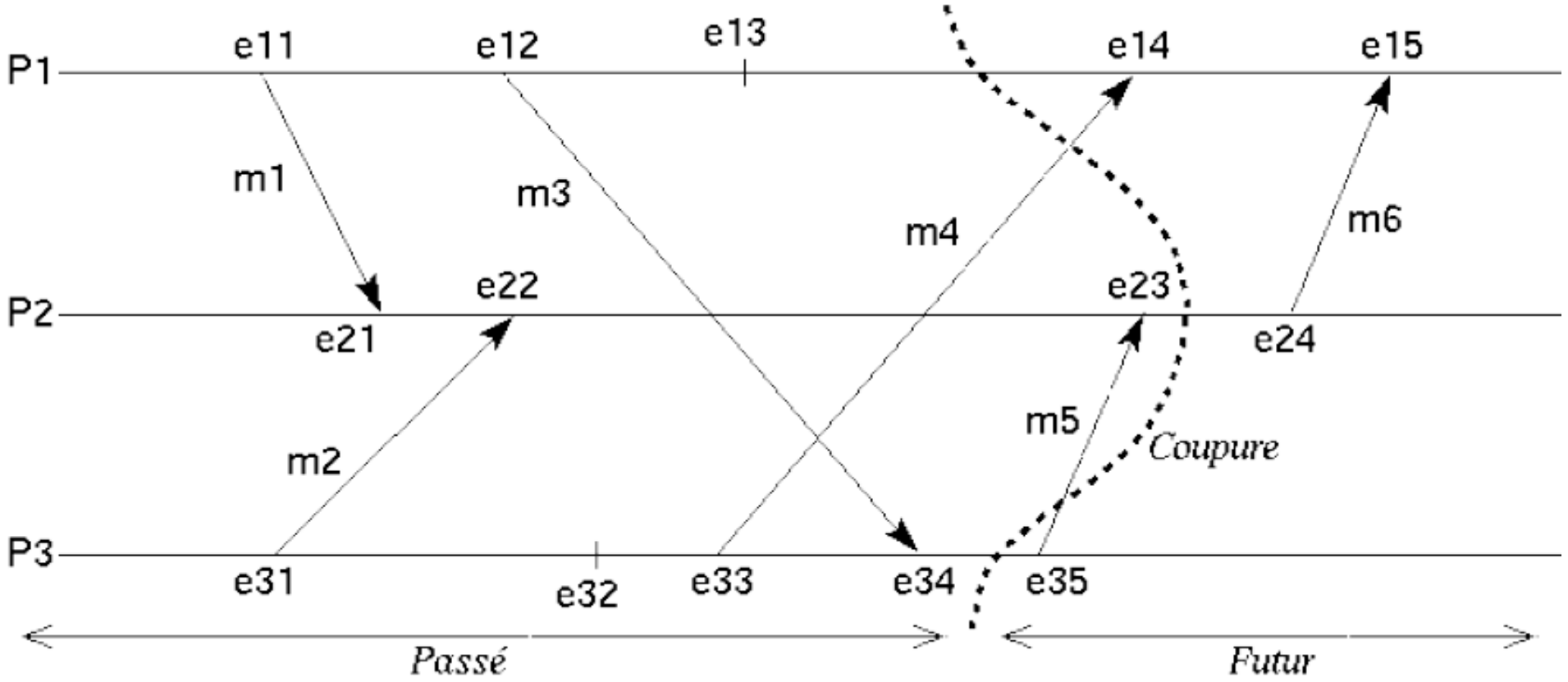
État Global

- Un état global est l'état du système à un instant donné
- Buts de la recherche d'états globaux:
  - Trouver des états cohérents à partir desquels on peut reprendre un calcul distribué en cas de plantage du système
  - Détecter les propriétés stables, du respect d'invariants
  - Faciliter le debugging et la mise au point d'applications distribuées
- Un état global est défini à partir de coupures.
- **Une coupure** est une photographie à un instant donné de l'état du système
- Une coupure définit les événements appartenant au passé et au futur par rapport à l'instant présent de la coupure.

Définition d'une coupure

- Un calcul distribué peut être considéré comme un ensemble *E* d'événements.
- Une coupure *C* est un sous-ensemble fini de *E* tel que
  - Soit *a* et *b* deux événements du même processus : a ∈ C et b → a => b ∈ C
- Si un événement d'un processus appartient au passé, alors tous les événements locaux le précédant y appartiennent également
- **État associé à une coupure** : Si le système est composé de N processus, l'état associé à une coupure est défini au niveau d'un ensemble de N événements (e1, e2, ... ei, ... eN), avec ei événement du processus Pi tel que:
  - ∀ i : ∀ e ∈ C et *e* un événement du processus Pi => e → ei
  - L'état ei est défini à **la frontière de la coupure : l'événement le plus récent pour chaque processus**

Exemple de coupure

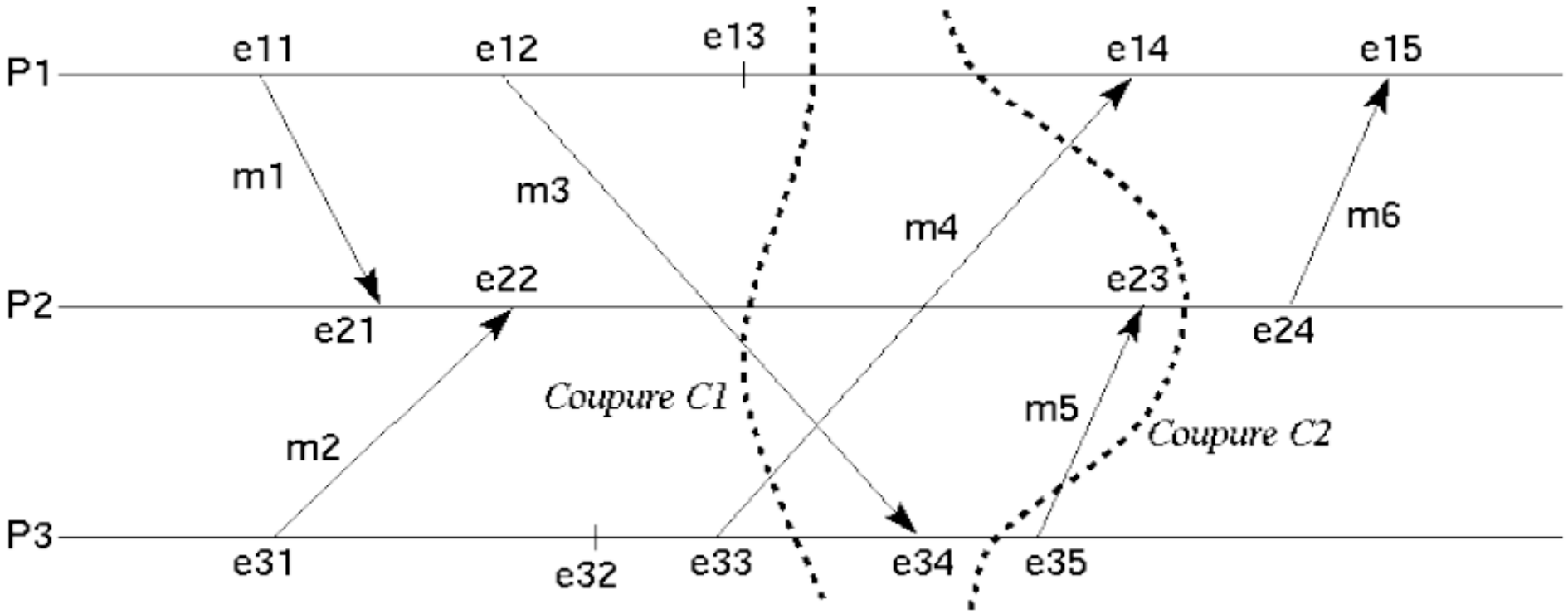


- Coupure = ensemble { e11, e12, e13, e21, e22, e23, e31, e32, e33, e34 }
- **État** défini par la coupure = (e13, e23, e34)

Coupure/état cohérent

- **Coupure cohérente**:
  - Coupure qui respecte les dépendances causales des événements du système et pas seulement les dépendances causales locales à chaque processus.
  - Coupure cohérente → **aucun message ne vient du futur**
- État cohérent:
  - État associé à une coupure cohérente
  - Permet par exemple une reprise sur faute

Coupure cohérente



- Coupure C1 : cohérente
- Coupure C2 : non cohérente car e35 → e23 mais e35 ∉ C2
  - **La réception de *m5* est dans la coupure mais pas son émission**
  - ***m5* vient du futur par rapport à la coupure (viole la causalité)**

Reprise d'un système distribué

- A partir d'un état global, on peut relancer un système à partir (l'état est enregistré à l'aide d'un algorithme spécifique):
  - On **recharge** chaque processus avec son état local
  - On **réémet** les messages qui se trouvaient dans les états des canaux : Ils ont été émis mais pas encore reçus par leurs destinataires
  - Chaque processus **reprend son exécution là où son état avait été enregistré**
- **Mais, on a besoin d'une coupure cohérente** associé à un état:
  - Exemple: la coupure C2 est non cohérente
  - P2 a déjà reçu et traité le message m5
  - P3 redémarre juste après e34 et la première chose que fera P3 est d'envoyer en e35 le message m5 qui sera donc reçu et traité deux fois par P2