Master Réseaux et Systèmes Distribués (RSD)

Algorithmique des systèmes et applications réparties

Horloges logiques (Partie 1)

Badr Benmammar

badr.benmammar@gmail.com

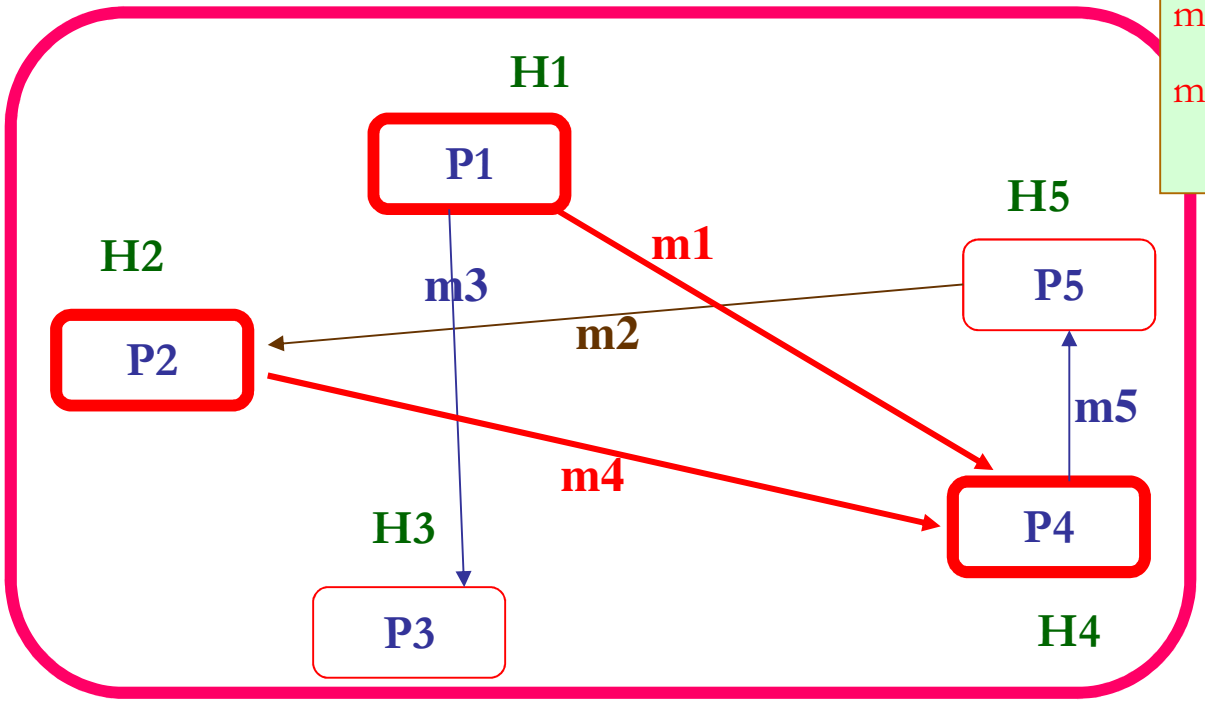
Plan

- **Temps dans un système distribué**
  - Temps logique && Horloge logique
  - Chronogramme
  - Dépendance causale
  - Parallélisme logique
- **Horloges logiques**
  - Estampille (horloge de Lamport)
  - **Vectorielle (horloge de Mattern)**
  - **Matricielle**

Le temps dans un système distribué

Temps logique && Horloge logique

- **Motivation :** définir un **temps global** pour tous les processus.
  - Créer **un temps logique (contrairement au temps physique).**
    - Temps qui n'est pas lié à un temps physique (propre à chaque processus).



Un calcul parallèle

m1 envoyé par P1 vers P4 à l'instant T1

m4 envoyé par P2 vers P4 à l'instant T2

Quel est le lien entre T1 et T2 ?

Ordonnancement → Temps

Temps → Horloge

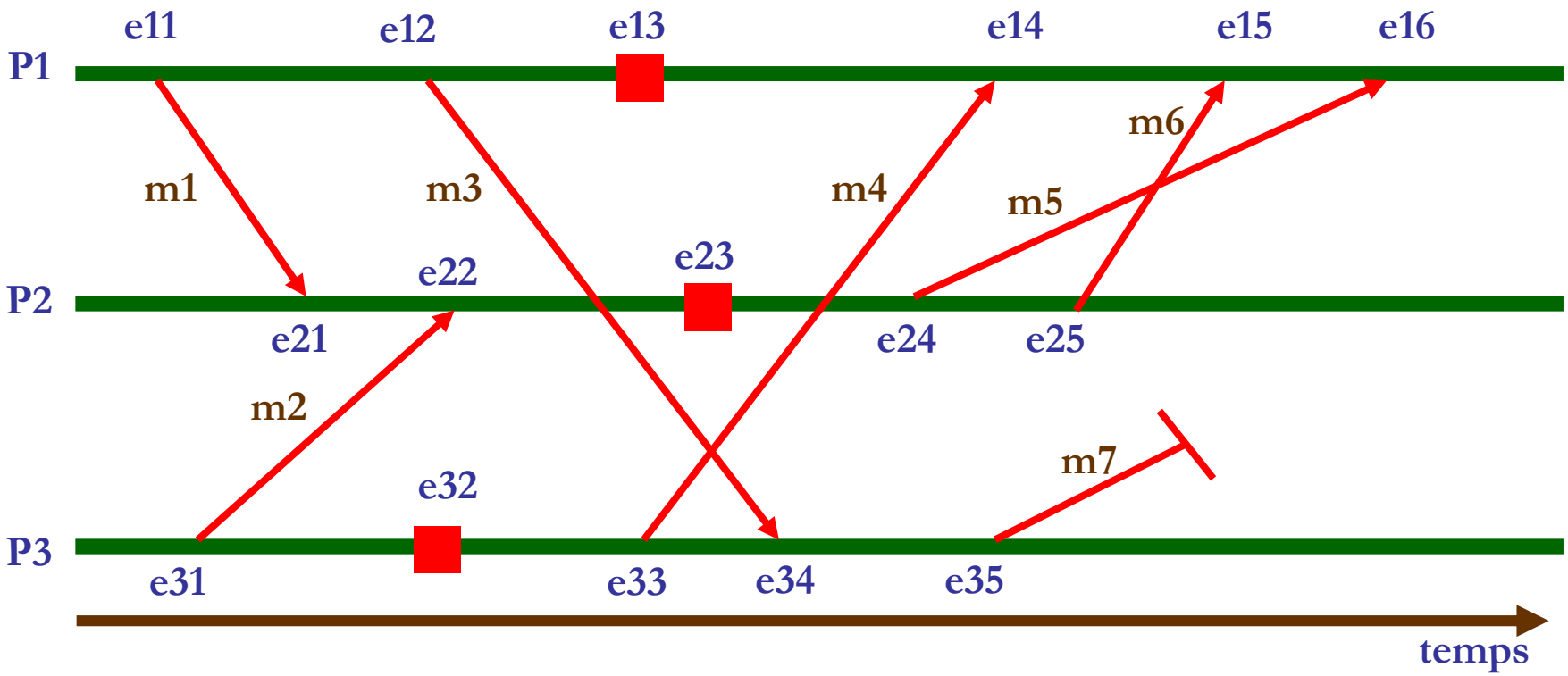
Système d'exploitation distribué

Chronogramme

- **Définition :** 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 (local)** dans l'évolution du processus.
- Les messages échangés doivent respecter la topologie de liaison des processus via les canaux.

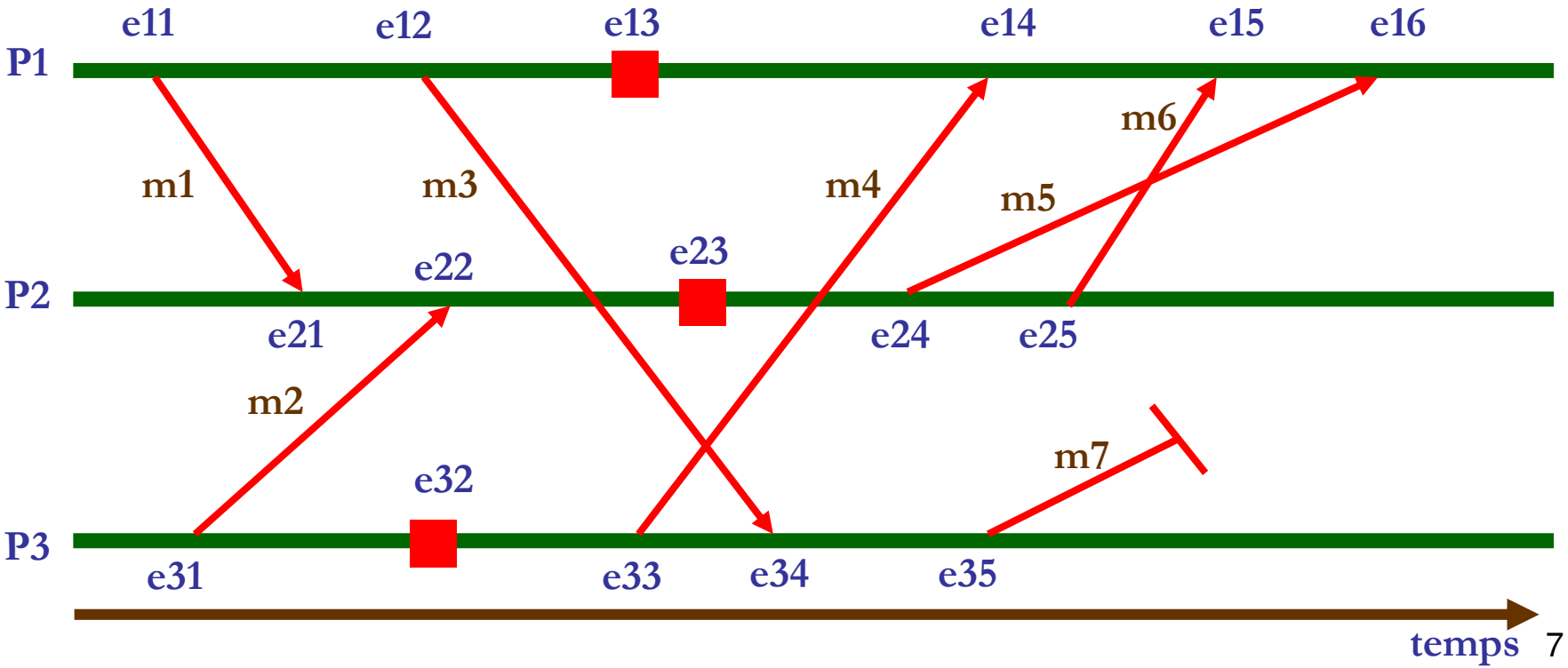
Chronogramme

- **Exemple :** trois processus tous reliés entre eux par des canaux avec la possibilité de perte de message.
- **Règle de numérotation d'un événement :** **eXY** avec **X le numéro du processus** et **Y le numéro de l'événement pour le processus**, dans l'ordre croissant.



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 → **UDP.**
- **Processus P3 :** le message m7 est perdu par le canal de communication → **UDP.**



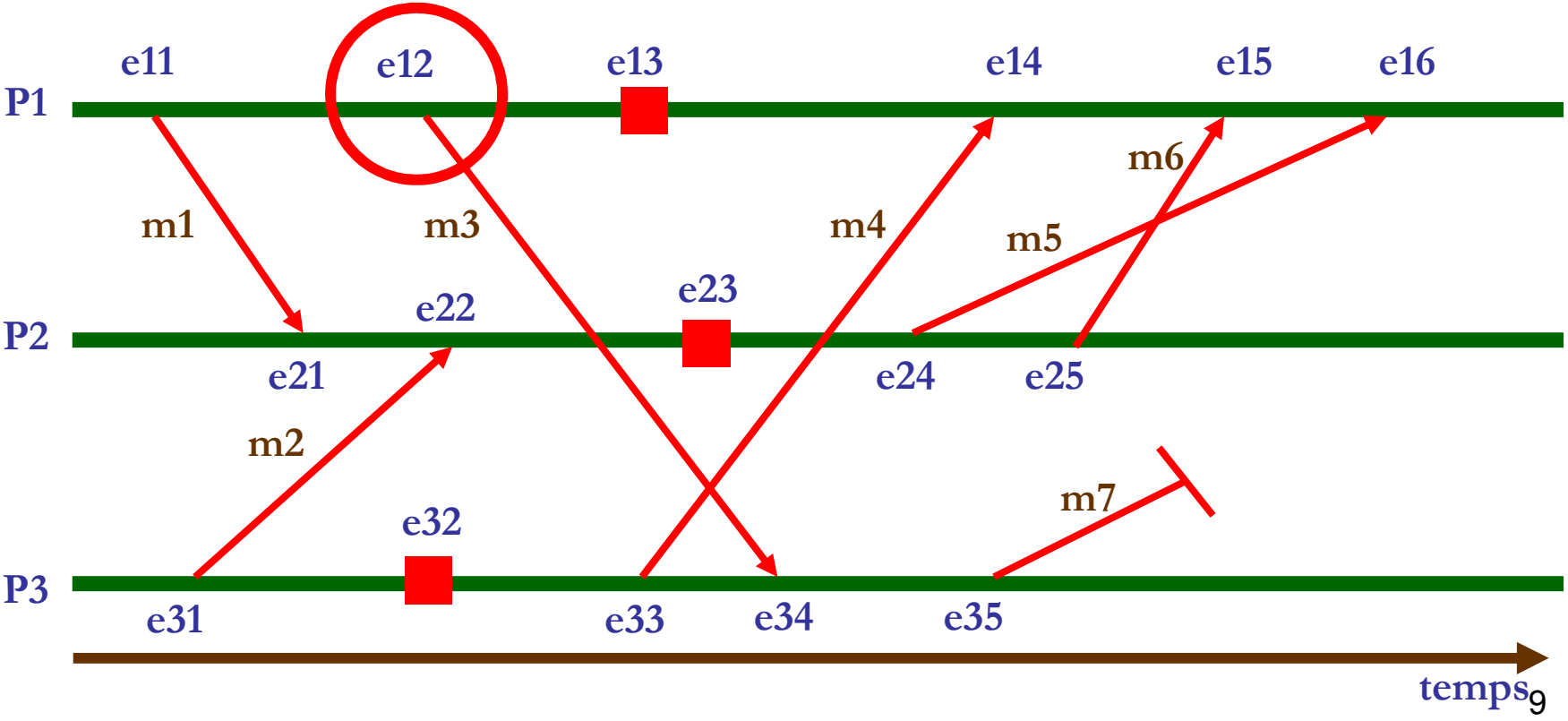
Dépendance causale

- **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' dépend causalement de 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'*.**

Dépendance causale

- Sur l'exemple précédent, 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 (car e11 → e12 et e12 → e13).**



Dépendance causale

- **La dépendance causale est une relation d'ordre partiel sur l'ensemble d'événements du système.**

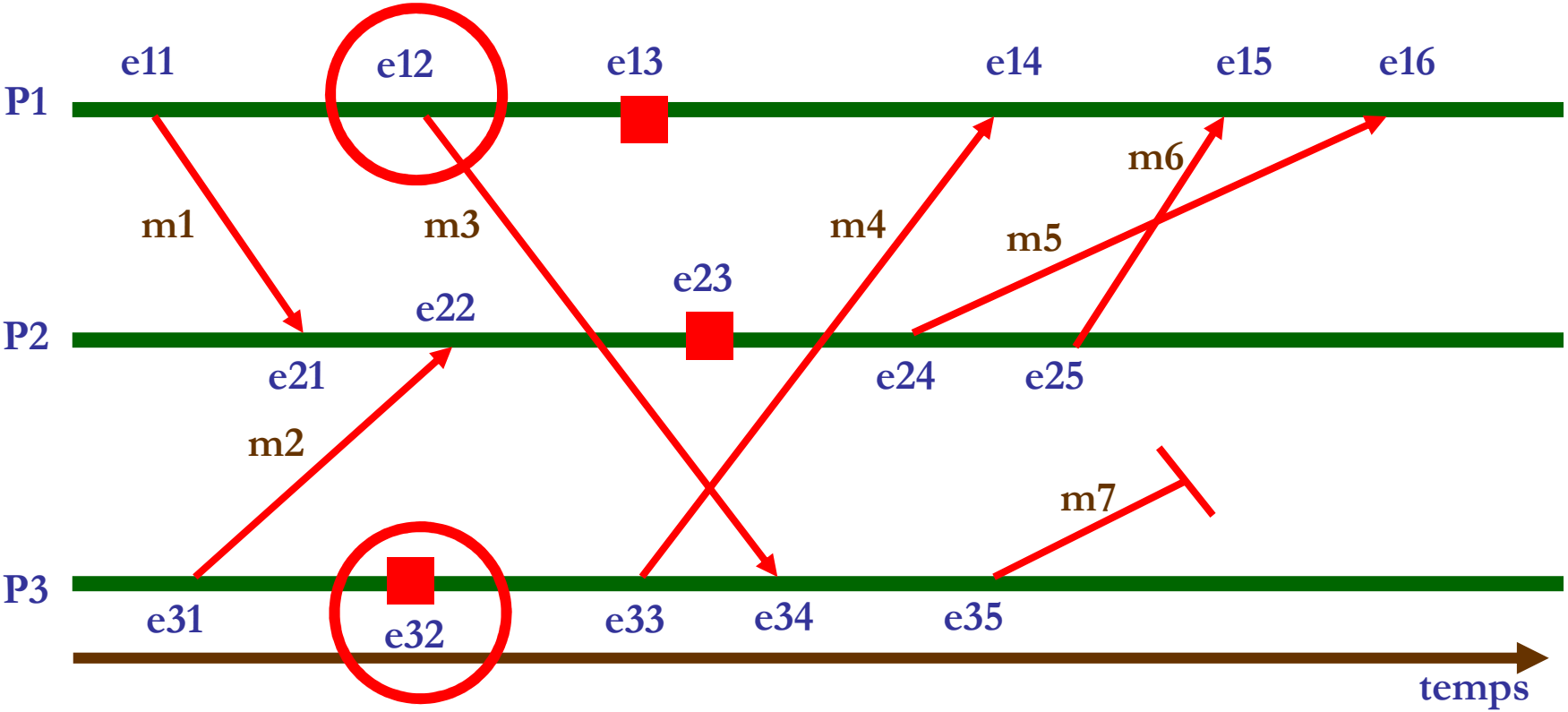
- **R est une relation d'ordre total sur E :**
  - **∀ (x et y) ∈ E : (x R y) ou (y R x).**
  - **Exemple :** $\leq$ est une relation d'ordre total sur les entiers.

- **R est relation d'ordre partiel sur E :**
  - **∃ (x et y) ∈ E, tel que ¬ (x R y) et ¬ (y R x).**
  - **Exemple :** $<$ est une relation d'ordre partiel sur les entiers ($2 < 2$).

- **R =** dépendance causale. **E** = ensemble d'événements du système.

Parallélisme logique

- **Question :** dépendance causale entre e12 et e32 ?
- A priori non : absence de dépendance causale.
- **Des événements non liés causalement se déroulent en parallèle.**



Parallélisme logique

- e et e' sont en dépendance causale ⇔ **((e → e') ou (e' → e)).**
- Relation de parallélisme : ||.

- **e || e' ⇔ $\overline{((e \rightarrow e') \text{ ou } (e' \rightarrow e))}$ ⇔ $\overline{(e \rightarrow e')}$ et $\overline{(e' \rightarrow e)}$.**

- **Parallélisme logique :** ne signifie pas que les 2 événements se déroulent simultanément mais **qu'il peuvent se dérouler dans n'importe quel ordre.**

Horloges logiques

Horloges logiques

- **Objectif :** dater les événements du système.
- **Principe :** respecter les dépendances causales entre les événements.

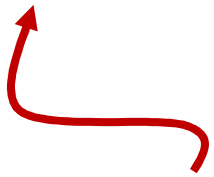
- **3 familles d'horloge :**
  - **Estampille (horloge de Lamport) : une donnée** par événement.
  - **Vectorielle (horloge de Mattern) : un vecteur** par événement.
  - **Matricielle : une matrice** par événement.

Horloge de Lamport

Horloge de Lamport

- Introduit en 1978 par **Leslie Lamport.**
- **C'est le premier type d'horloge logique introduit en informatique.**
- Une date (estampille) est associée à chaque événement.

- **estampille** représente un couple **(i, nb).**
  - **i : numéro du processus.**
  - **nb : numéro d'événement.**



Représente le temps de l'horloge logique.

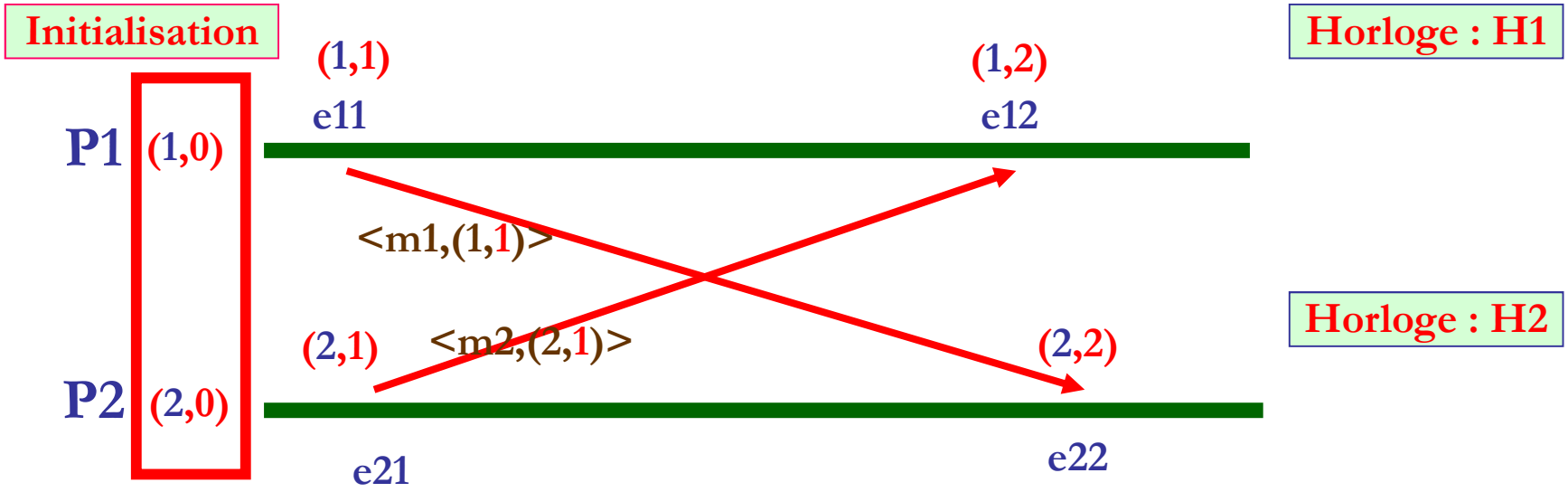
Horloge de Lamport

- **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.**
    - **Hi = Hi + 1** puis on envoie le message avec (i, Hi) comme estampille.
  - **Réception d'un message *m* avec estampille (i, nb)**
    - **Hj = max(Hj, nb) +1** et marque l'événement de réception avec Hj.

Temps de l'horloge locale dans Pj

Horloge de Lamport

- Créer un temps logique à l'aide de l'horloge de Lamport.



- Relation importante : **H(s, nb) < H(s', nb') si (nb < nb') ou (nb = nb' et s < s').**
- **Ordonnancement global :**

(1,1) (1,2)

(2,1) (2,2)

- **Résultat : e11 << e21 << e12 << e22.**
- **H(e11) < H(e21) < H(e12) < H(e22).**

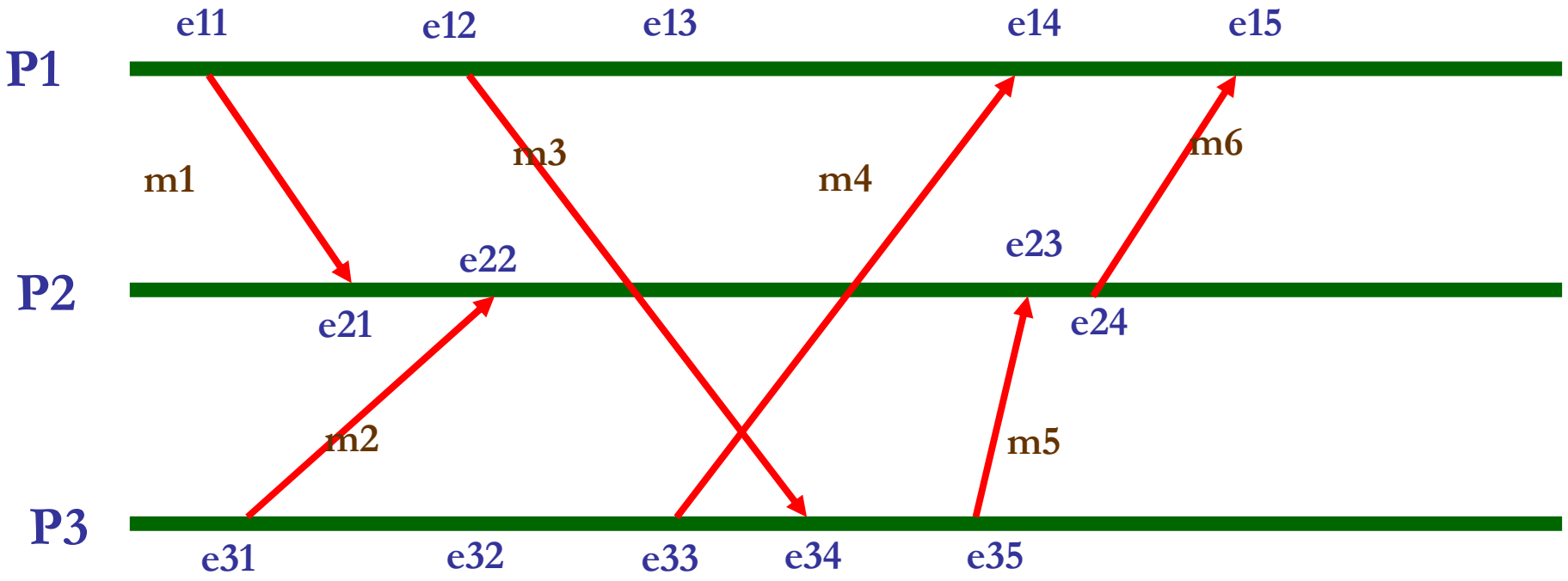
Horloge de Lamport

- **Ordonnancement global : e11 << e21 << e12 << e22.**
- **H(e11) < H(e21) < H(e12) < H(e22).**
- L'horloge de Lamport respecte la dépendance causale :
  - **(e → e') ⇒ (H(e) < H(e')).**
- Selon l'horloge : **H(e11) < H(e21).**
- **Pourtant, il y a une absence de dépendance causale entre e11 et e21 (un parallélisme).**

- **Résumé :**
- **L'horloge de Lamport respecte la dépendance causale :**
  - (e → e') ⇒ (H(e) < H(e')).
- **Mais pas la réciproque de la dépendance causale :**
  - (H(e) < H(e')) ⇏ (e → e').

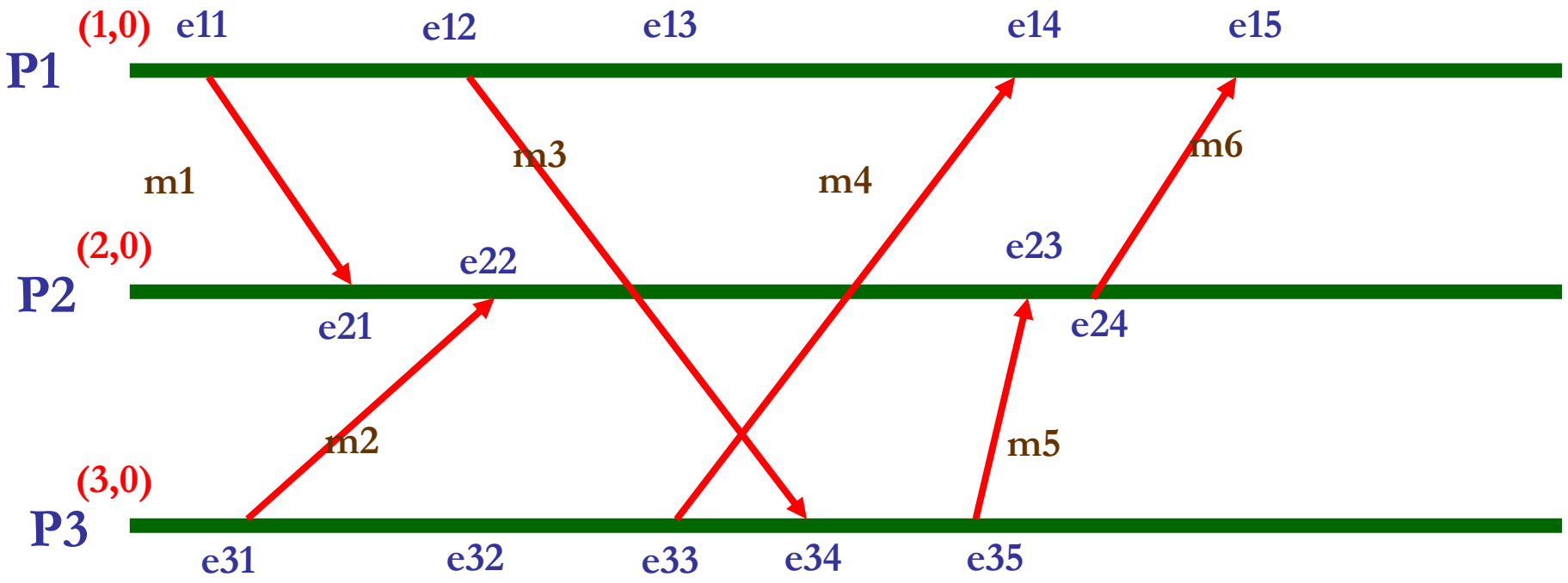
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.



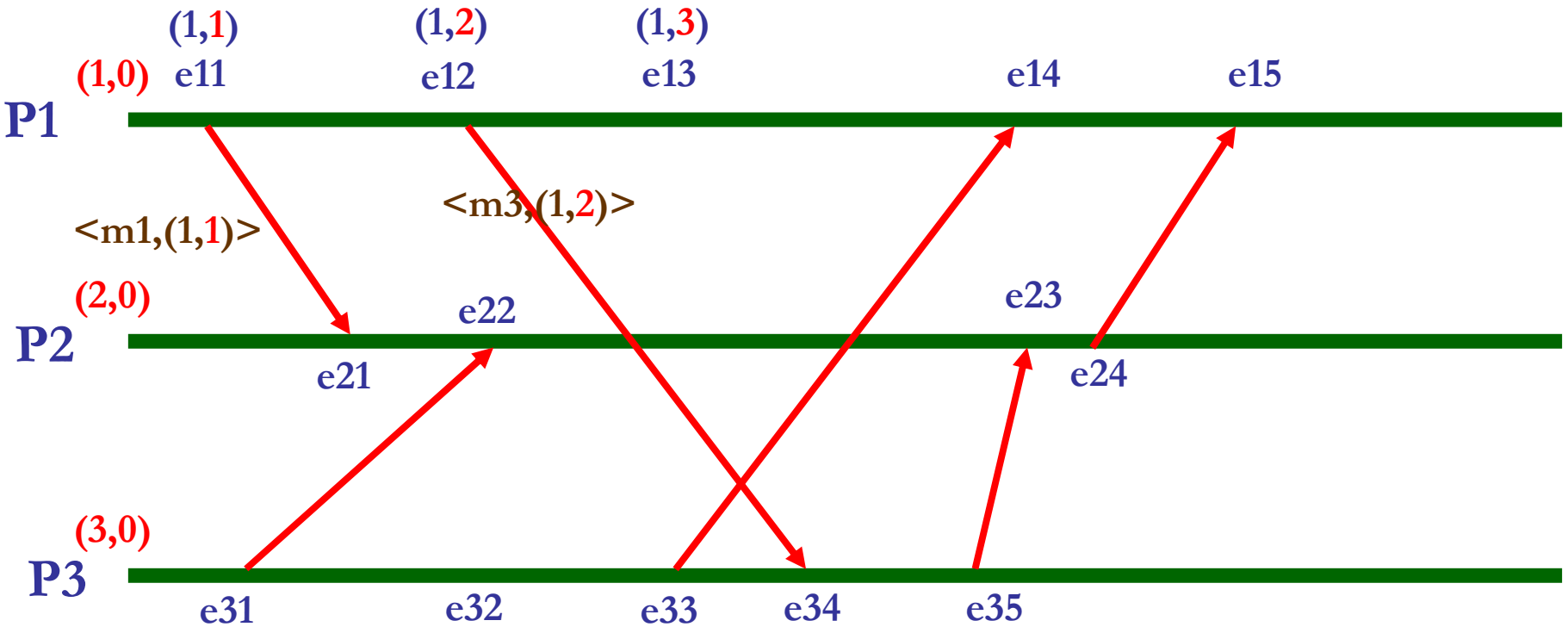
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.



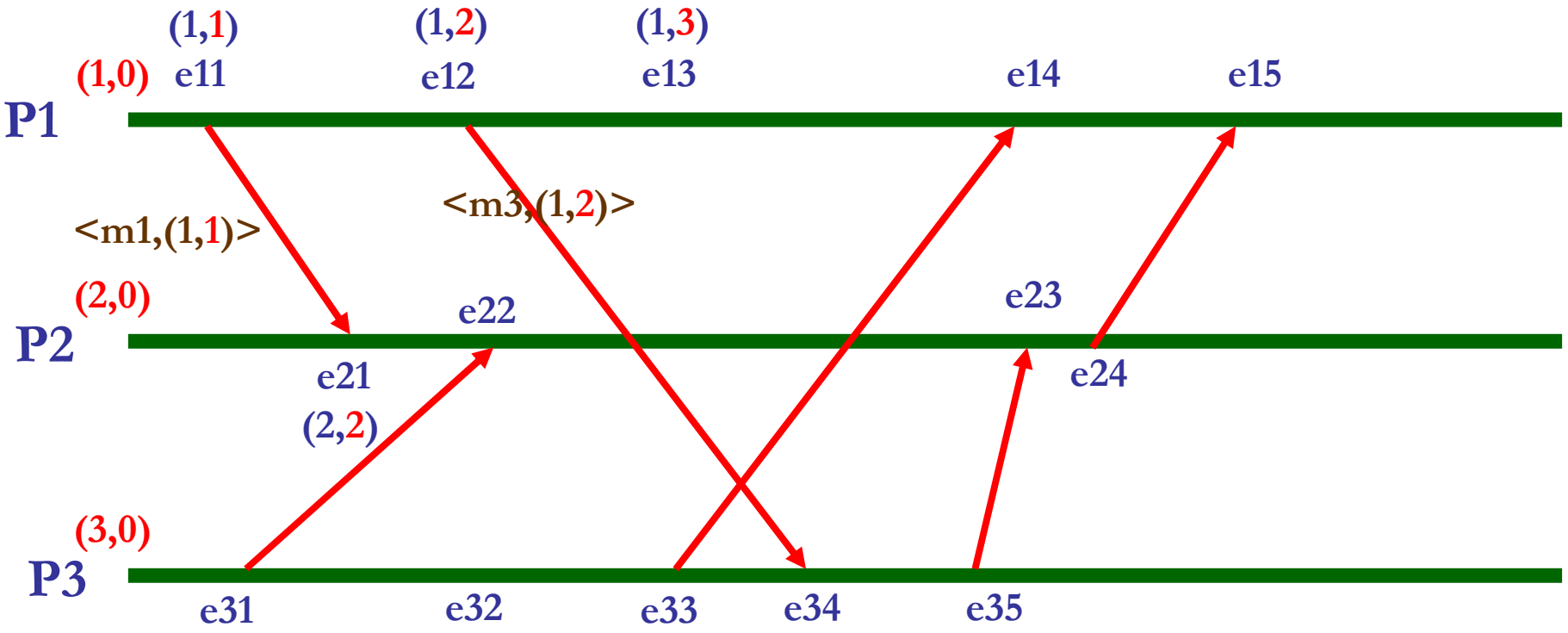
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.



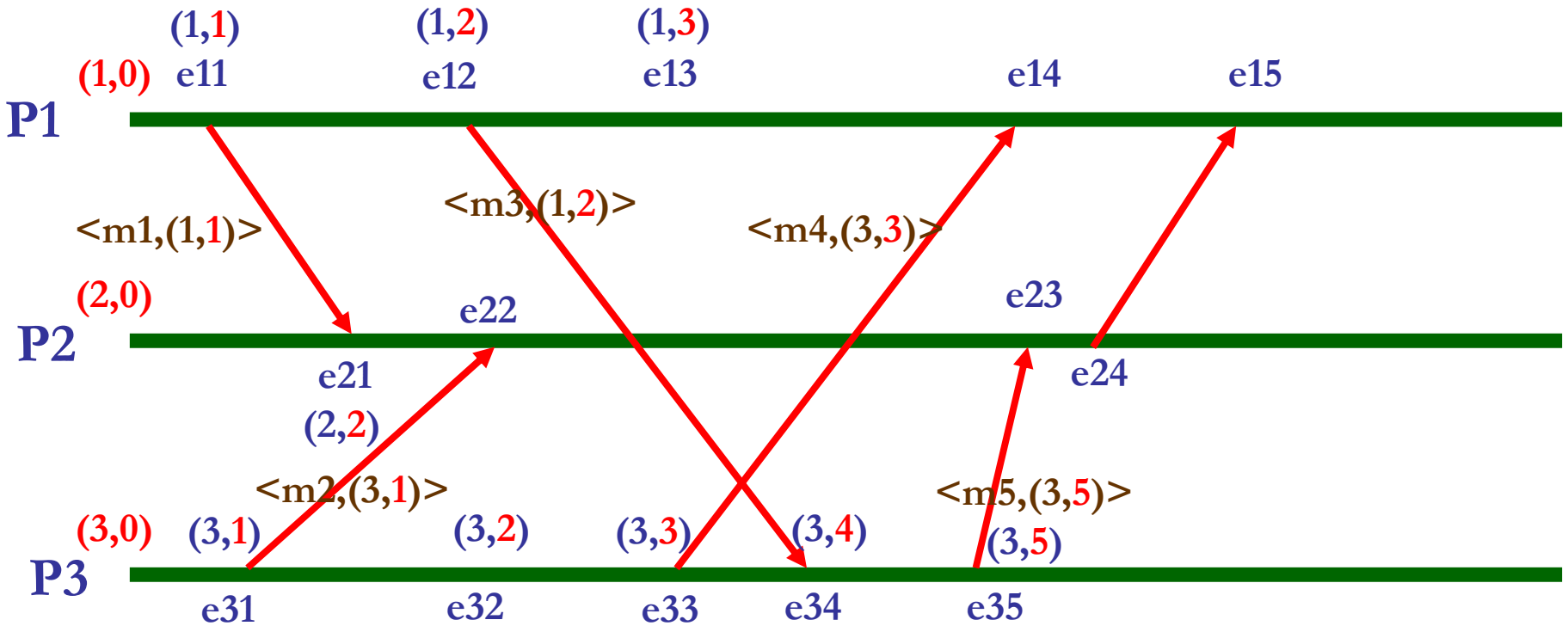
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.



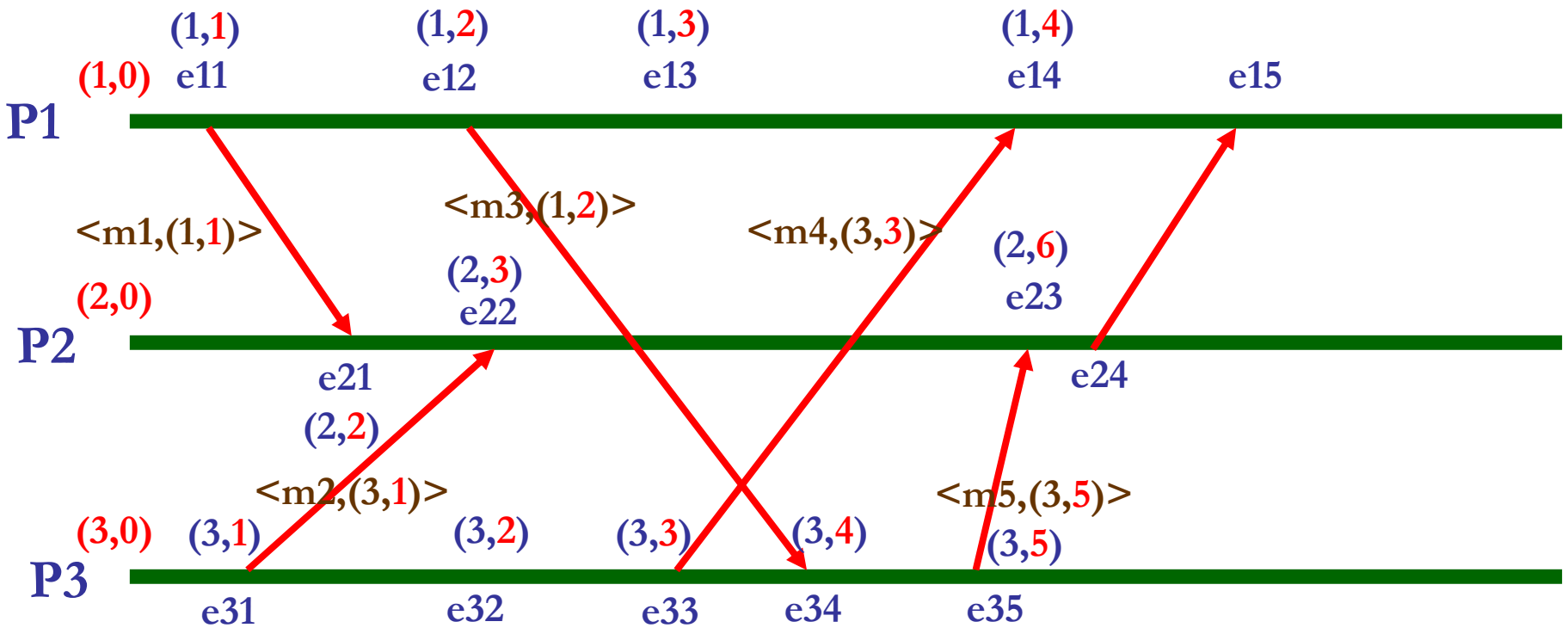
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.



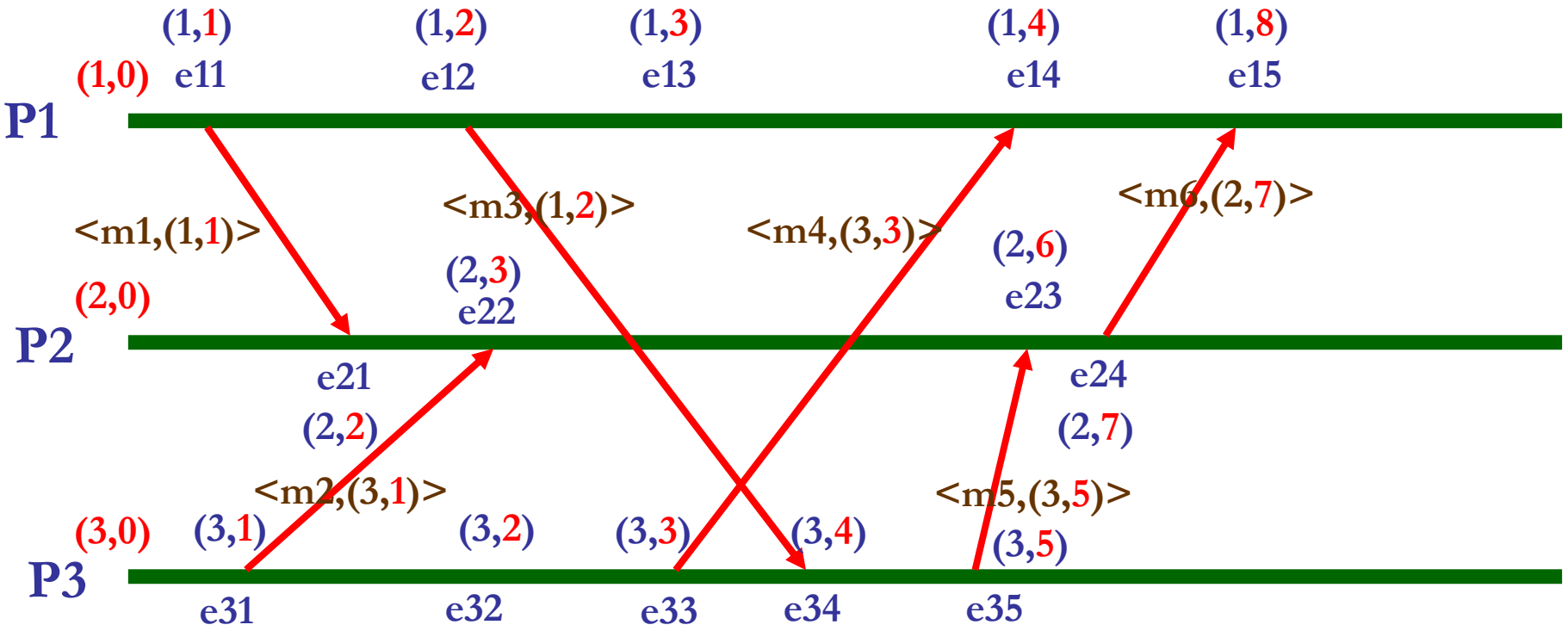
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.



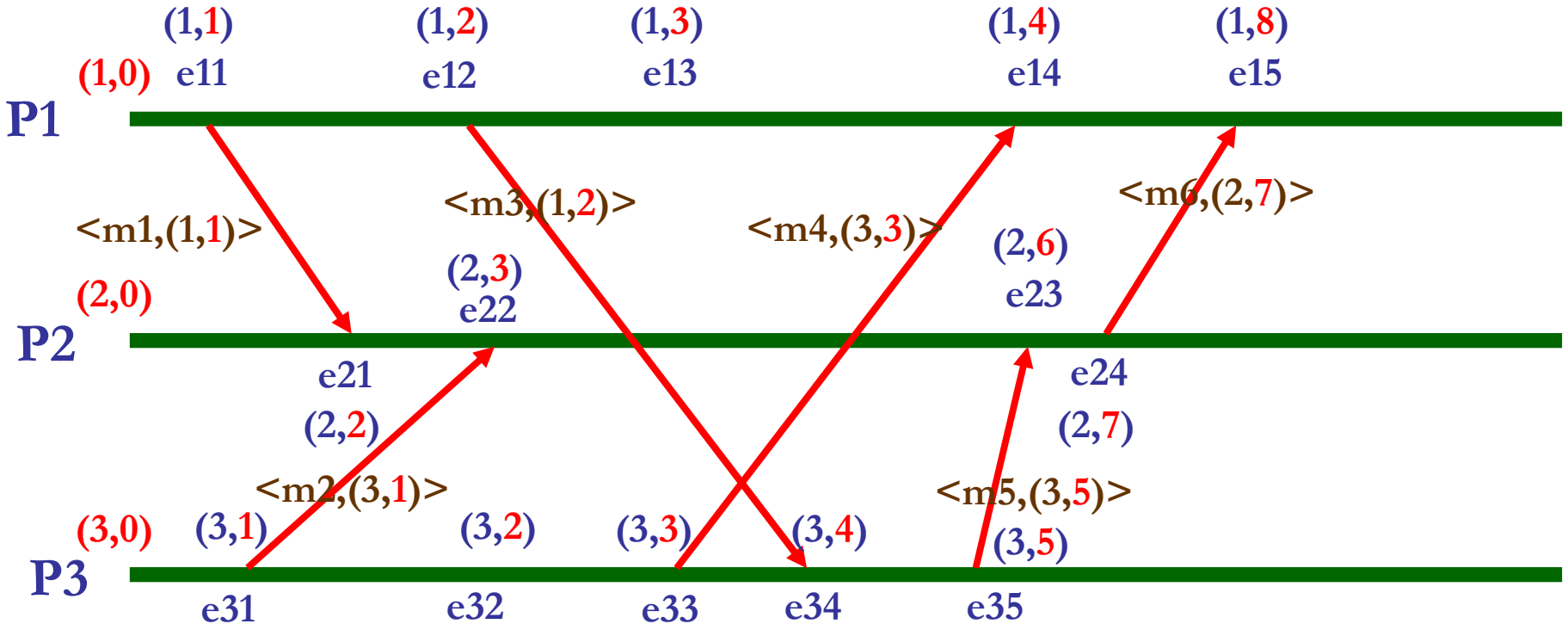
Exemple

- **Exemple 2 :** chronogramme avec ajouts des estampilles.

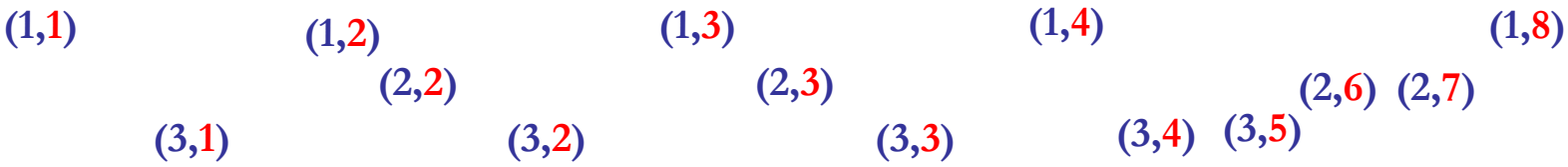


- **Pour e11, e12, e13 ... :** incrémentation de +1 de l'horloge locale.
- **Date de e23 : 6** car le message m5 reçu avait une valeur de 5 et l'horloge locale est seulement à 3 (max (3,5)+1).
- **Date de e34 : 4** car on incrémente l'horloge locale vu que sa valeur est supérieure à celle du message m3 (max (3,2)+1).

Ordonnancement global



❑ **Ordre total global obtenu pour l'exemple :**



e11 << e31 << e12<< e21 << e32 << e13 << e22 << e33 << e14 << e34 << e35 << e23 << e24 << e15.

Ordonnancement global

- **La relation << (l'ordonnancement global) est une relation d'ordre total sur l'ensemble d'événements du système.**
- Via Hi, on ordonne tous les événements du système entre eux.
- **Soit *e* événement de *Pi* et *e'* événement de *Pj* :**
  - **e << e' ⇔ (Hi(e) < Hj(e')) ou (Hi(e) = Hj(e') avec i < j).**
- 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.

Utilité de l'horloge de Lamport

Faire l'ordonnancement global des événements dans un système distribué.

Master Réseaux et Systèmes Distribués (RSD)

Algorithmique des systèmes et applications réparties

Horloges logiques (Partie 2)

Badr Benmammar

badr.benmammar@gmail.com

Plan

- **Temps dans un système distribué**
  - Temps logique && Horloge logique
  - Chronogramme
  - Dépendance causale
  - Parallélisme logique
  - Délivrance

- **Horloges logiques**
  - Estampille (horloge de Lamport)
  - **Vectorielle (horloge de Mattern)**
  - **Matricielle**

Horloge de Mattern

Horloge de Mattern

- Introduit indépendamment en 1988 par **Colin Fidge et Friedemann Mattern.**
- **Horloge qui assure la réciproque de la dépendance causale :**
  - **H(e) < H(e') ⇒ (e → e').**
- **Permet également de savoir si 2 événements sont parallèles (non dépendants causalement).**
- Ne définit par contre pas un ordre total global.

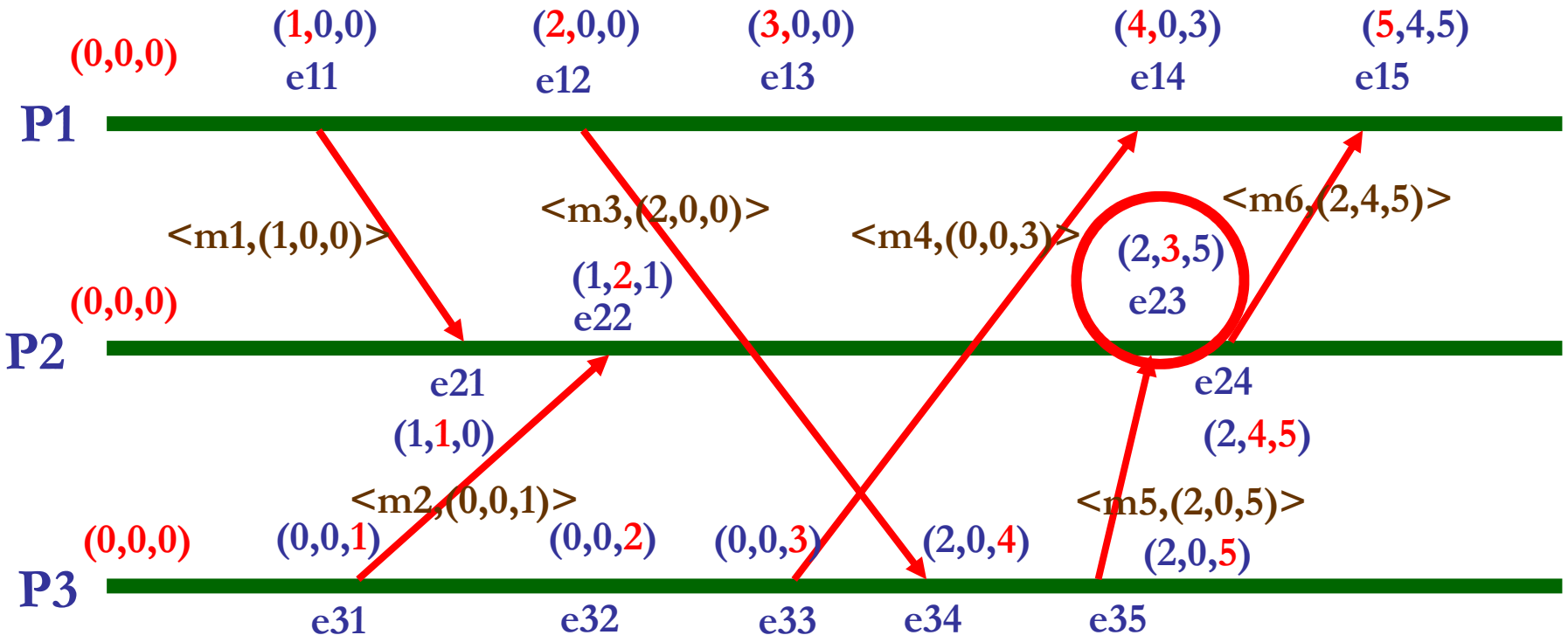
- **Principe :**
- Utilisation de vecteur V de taille égale au nombre de processus.
  - Localement, chaque processus *Pi* a un vecteur *Vi*.
  - Un message est envoyé avec un vecteur de date.

Horloge de Mattern

- **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 **(garder les autres compteurs si événement local ou émission).**
    - 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* provenant du processus *Pj*, on met à jour les cases k ≠ i de son vecteur local *Vi*.
    - **∀ k ≠ i : Vi [k] = max (Vm [k], Vi [k]).**

Horloge de Mattern

- Même exemple que pour horloge de Lamport :



- **V(e23) = (2,3,5) : (2 relatif à P1, 3 à P2, 5 à P3).**
  - **3 = 2+1 (Incrémentation du compteur local).** 3ème évent dans P2.
  - **2 =max (2,0).** 2 évents dans P1 (e11 et e12) qui sont en dépendance causale par rapport à l'évent e23.
  - **5 =max (5,3).** 5 évents dans P3 (e31, e32, e33, e34 et e35) qui sont en dépendance causale par rapport à l'évent e23.

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 $\overline{((V < V') \text{ ou } (V' < V))} \Leftrightarrow \overline{(V < V')}$ et $\overline{(V' < V)}$.**
- **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 **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 (les 2 événements sont en **parallèle**).

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 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 ainsi que la réciproque.**
  - **Horloge de Lamport respecte uniquement les dépendances causales.**

- 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 (le parallélisme).**

État Global

- **État global :** état du système à un instant donné → **Défini à partir de coupures.**
  - 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.**

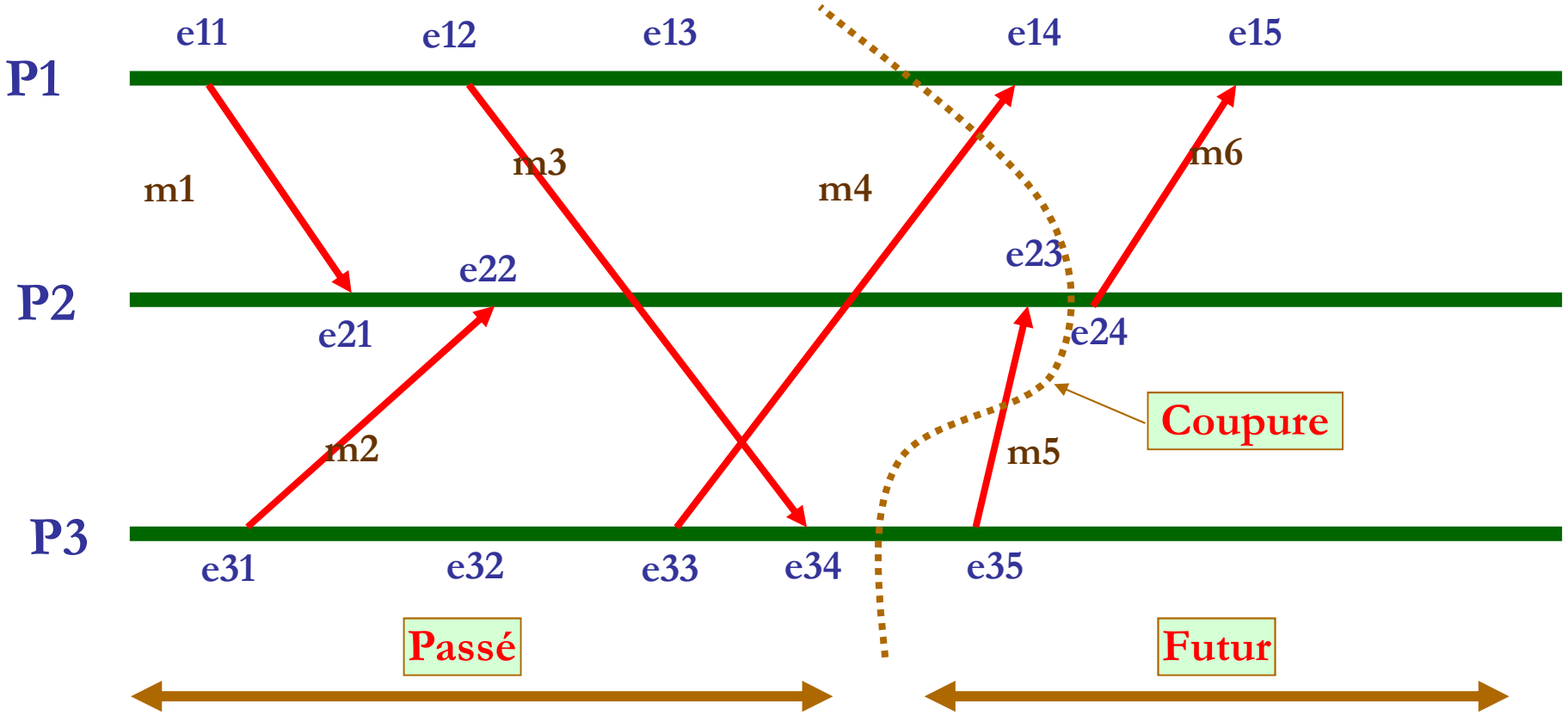
- **Coupure :** photographie à un instant donné de l'état du système → un état est associé à une coupure.
  - Définit les événements appartenant **au passé** et **au futur** par rapport à l'instant de la coupure.
  - **Coupure =** les événements appartenant **au passé** par rapport à l'instant de la coupure.

Coupure

- **Définition :**
  - **Calcul distribué = ensemble *E* d'événements.**
  - Coupure *C* est un sous-ensemble fini de *E* tel que :
    - ∀ (a et b) **deux événements du même processus :**

      ((a ∈ C) et (b → a)) ⇒ (b ∈ C).
    - Si un événement d'un processus appartient à la coupure, alors tous les événements locaux le précédant y appartiennent également.

Exemple de coupure



- Coupure = les événements du passé (avant la coupure).
- Coupure = {e11, e12, e13, e21, e22, e23, e31, e32, e33, e34}.

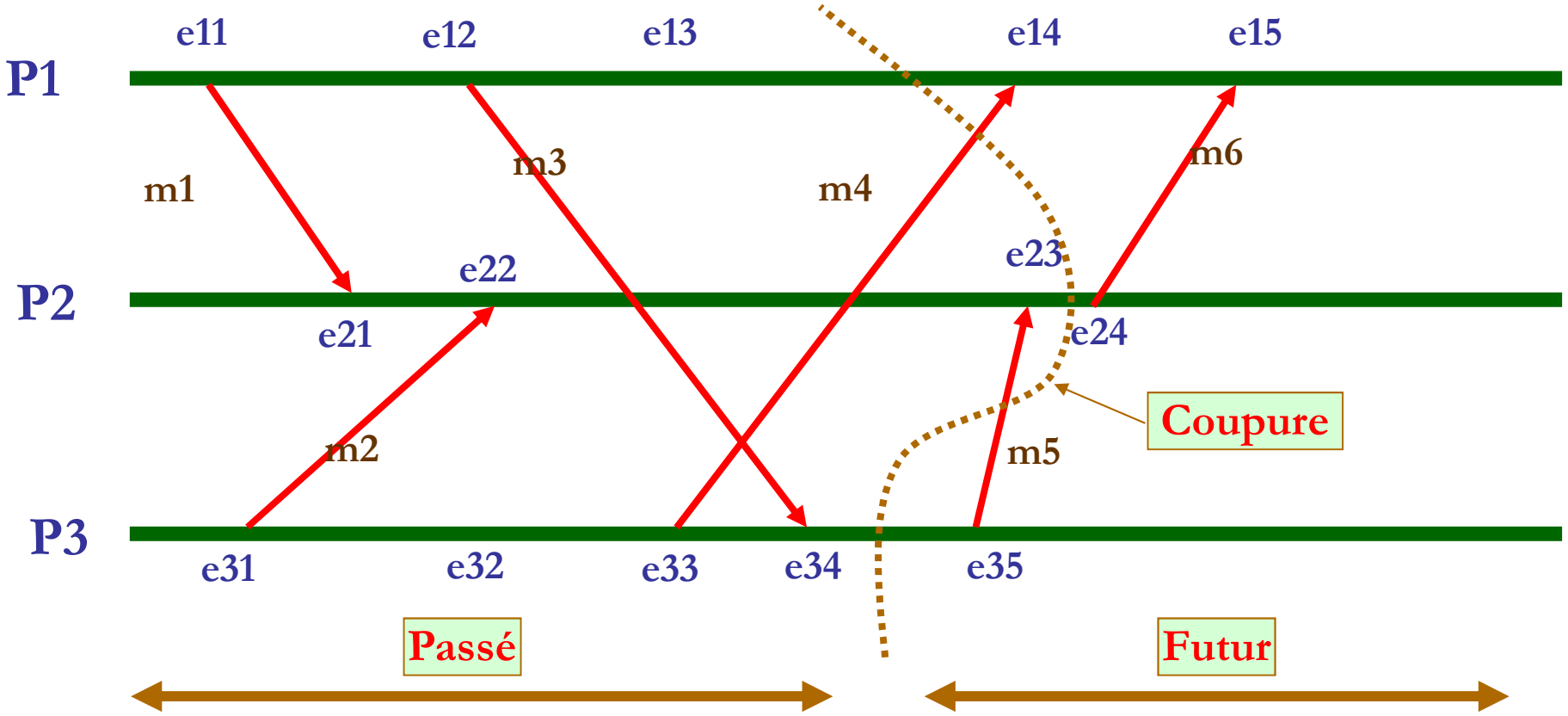
Etat associé à une coupure

- 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 :

  - **∀ e ∈ C (pour tous les événements de la coupure) :**
    - **∀ i (pour chaque processus) : (e ∈ Pi) ⇒ (e → ei).**

- **L'état est défini à la frontière de la coupure : l'événement le plus récent pour chaque processus.**

Etat de la coupure

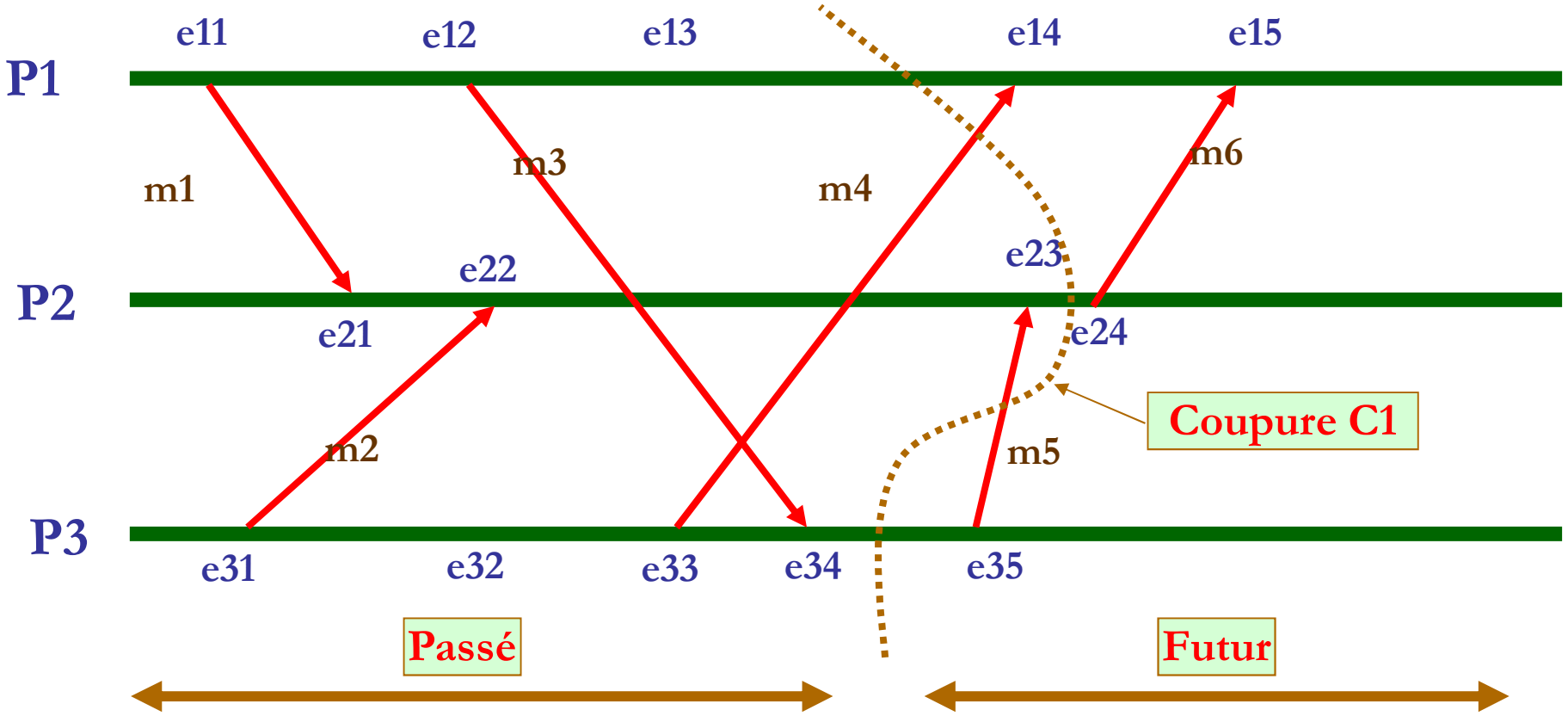


- Coupure = {e11, e12, e13, e21, e22, e23, e31, e32, e33, e34}.
- État de la coupure = (e13, e23, e34).
- Le dernier événement dans chaque processus avant la coupure.

Type de coupure (cohérente ou non)

- **Coupure cohérente :** coupure qui respecte les dépendances causales des événements du système et pas seulement les dépendances causales locales à chaque processus.
  - ∀ (a et b) **deux événements du système :**
    - **((a ∈ C) et (b → a)) ⇒ (b ∈ C).**
  - **Coupure cohérente :** aucun message ne vient du futur.
- **État cohérent : État associé à une coupure cohérente.**
  - **Permet une reprise après panne.**
- **Coupure non cohérente :**
  - ∃ (a et b) **deux événements du système :**
    - **((a ∈ C) et (b → a)) et (b ∉ C).**
  - **Coupure non cohérente :** il existe au moins un message qui vient du futur.
  - **Ne permet pas la reprise après panne.**

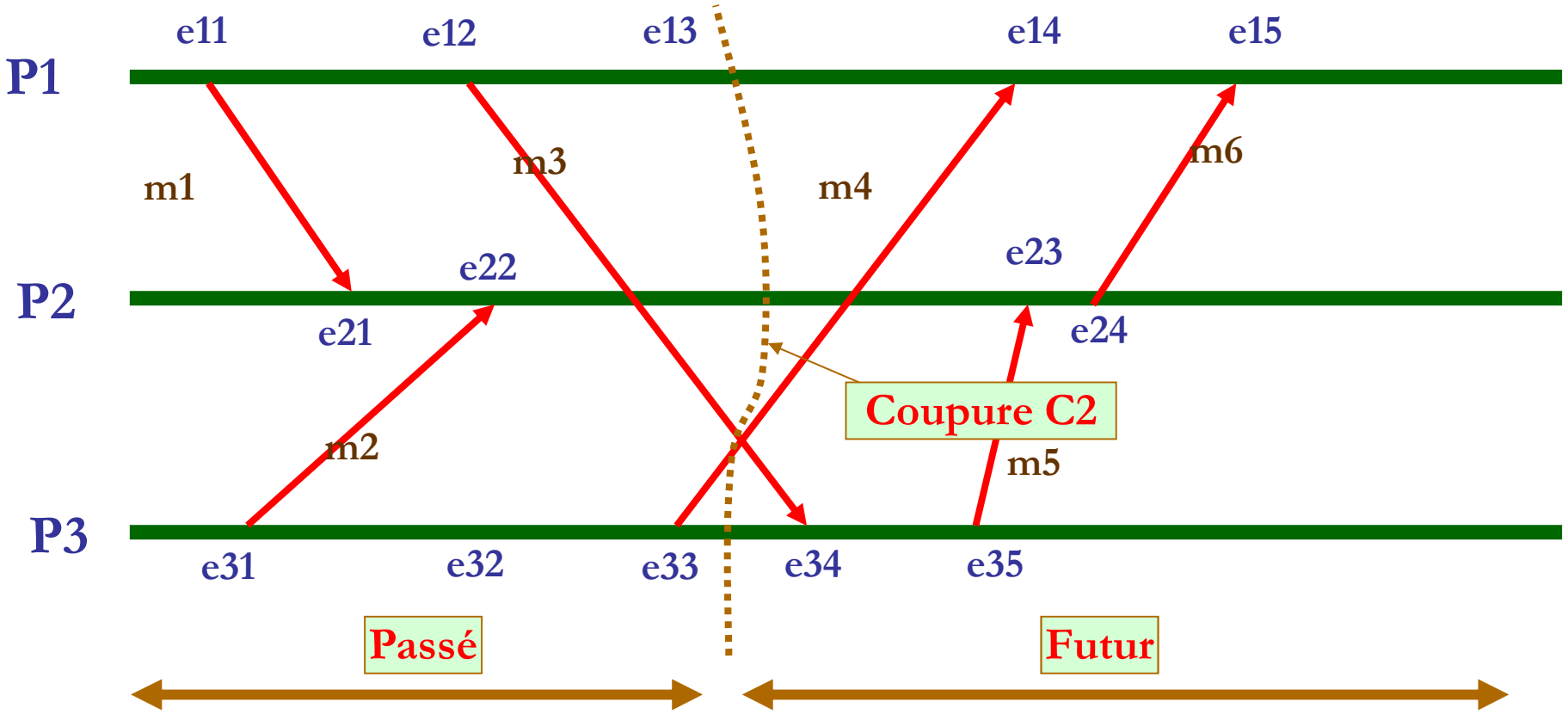
Exemple de coupure non cohérente



- **Coupure C1 : non cohérente** car : e23 ∈ C1 et e35 → e23 mais e35 ∉ C1.
- La réception de *m5* est dans la coupure mais pas son émission.
- *m5* vient du futur par rapport à la coupure → **ne permet pas le reprise après panne.**

Exemple de coupure cohérente

- Coupure C2 : cohérente → permet la reprise après panne.



Datation coupure

- **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)) :
    - **∀ 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 ou non.**

si (V(C) = (V(e1)[1], ... , V(ei)[i], ... , V(eN) [N])) alors
    Cohérente
Sinon
    Non cohérente
Fin si

Datation coupure

- Datation des coupures de l'exemple :
- **Coupure C2 : 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 C1 : 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] ≠ 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).**

Utilité de l'horloge de Mattern

Validation de la cohérence d'un état global

(faire la reprise après panne ou non)

&&

Savoir si 2 événements sont en dépendance causale

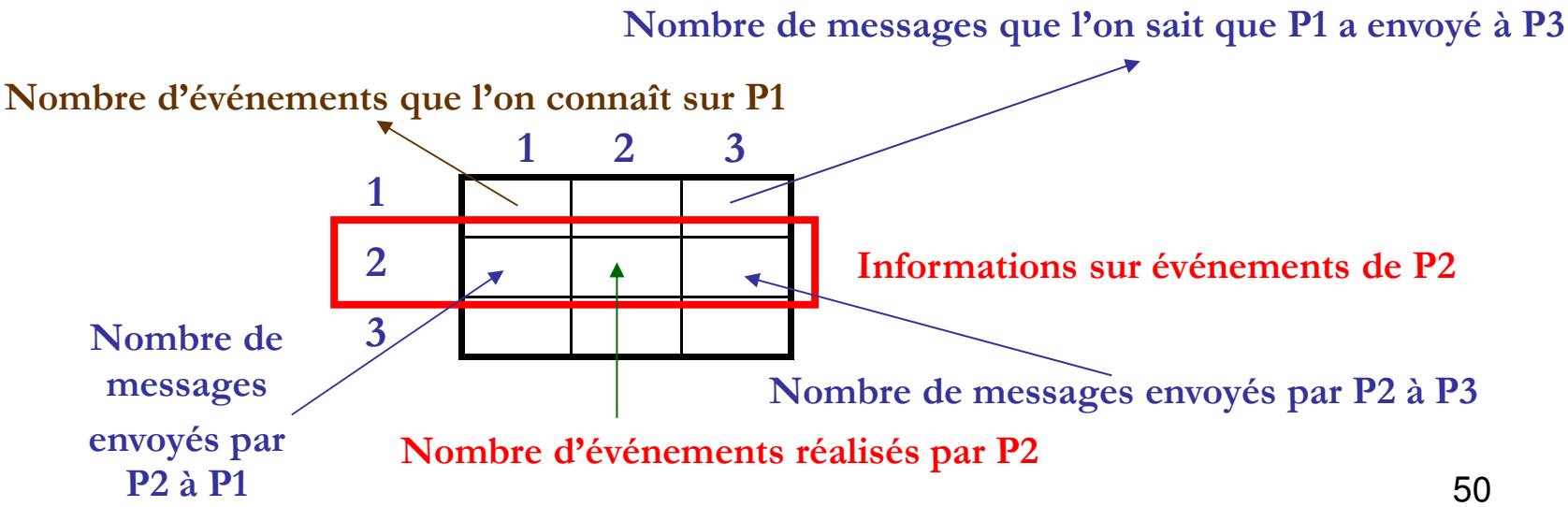
&&

Savoir si 2 événements sont en parallèle

Horloge matricielle

Horloge matricielle

- *n* processus : matrice ***M de (n x n)*** pour dater chaque événement.
- **Sur processus Pi :**
  - Ligne *i* : informations sur événements de Pi :
    - Mi [i , i] : nombre d'événements réalisés par Pi.
    - Mi [i , j] : nombre de messages envoyés par Pi à Pj (avec j ≠ i).
  - Ligne *j* (avec j ≠ i) :
    - Mi [j , j] : nombre d'événements que l'on connaît sur Pj.
    - Mi [j , k] : nombre de messages que l'on sait que Pj a envoyé à Pk (avec j ≠ k).
  - **Avec 3 processus (sur le processus P2):**



Exemple d'application

- Considérons un système contenant 3 processus. Tous les processus possèdent des horloges logiques matricielles.
- Supposons que l'horloge matricielle HM3 du processus 3 est HM3 =

6	2	2
1	6	1
1	2	7

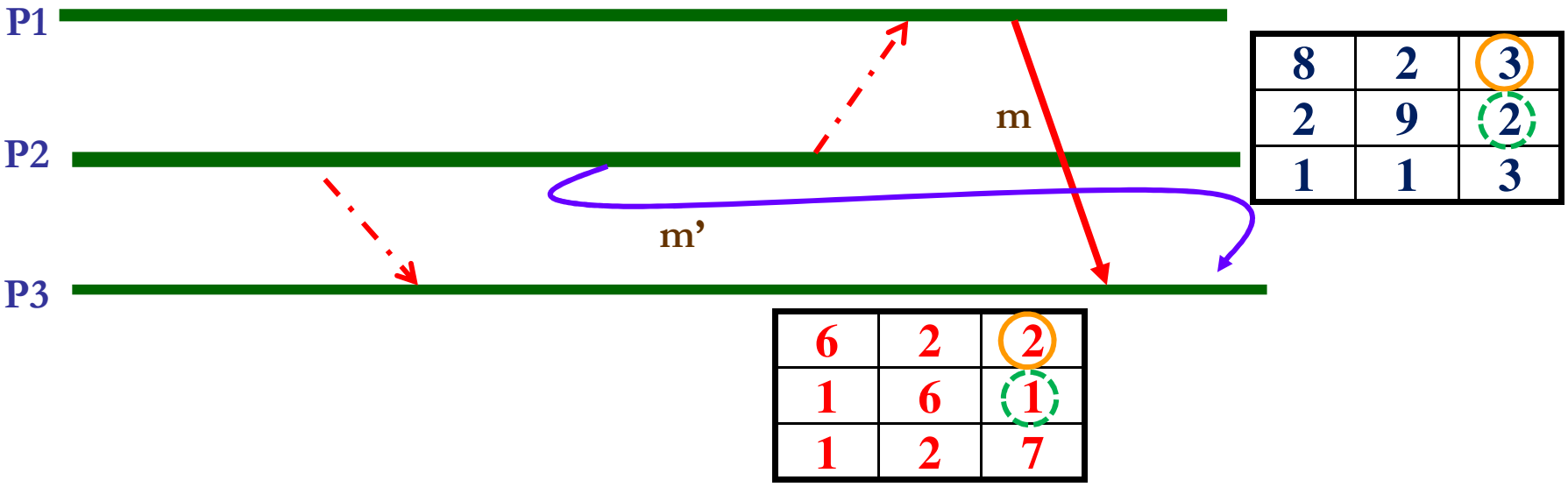
- A quoi correspond l'élément de l'horloge matricielle HM3 [3, 1], pour le processus 3 ?
- Même question pour les éléments HM3 [1, 3], HM3 [2, 3] pour le processus 3.
- Le processus 3 reçoit le message m en provenance du processus 1. L'estampille du message m est EMm =

8	2	3
2	9	2
1	1	3

- Que peut déduire le processus 3 par rapport aux éléments EMm [1, 3], EMm [2, 3] ?
- Le processus 3 peut-il délivré le message m (délivrance causale) ? Justifier votre réponse.

Correction

- Le message m correspond au **3ème message envoyé par P1 à P3.**
  - Comme P3 a déjà reçu **2 messages** venant de P1 donc **il n'existe pas de message encore non reçu venant de P1 dans le passé de la réception de m.**
- Dans le passé de l'émission de m, il existe l'évènement suivant : P2 a émis un message m' vers P3 mais P3 ne l'a pas encore reçu.

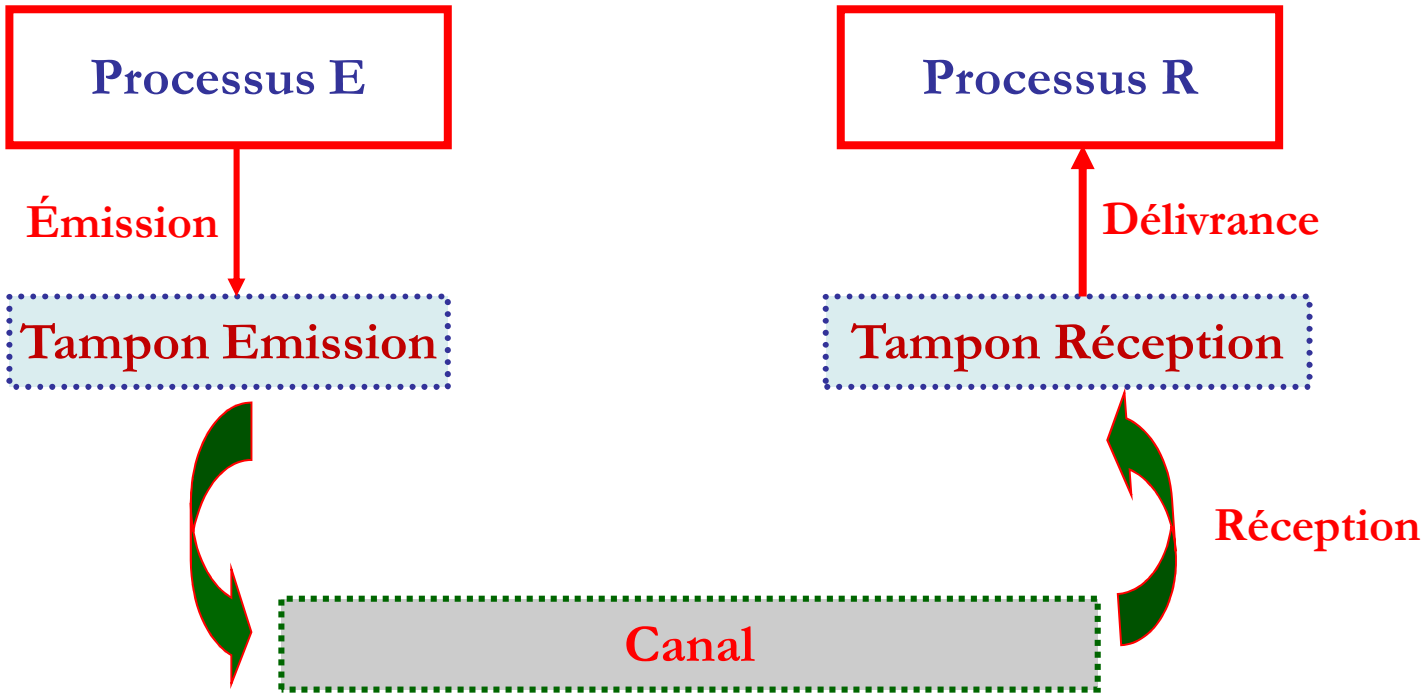


- **Pour respecter la délivrance causale, P3 doit attendre la réception de m', il le délivre par la suite il délivre le message m.**

Utilité de l'horloge matricielle

Assurer la délivrance causale de messages entre plusieurs processus.

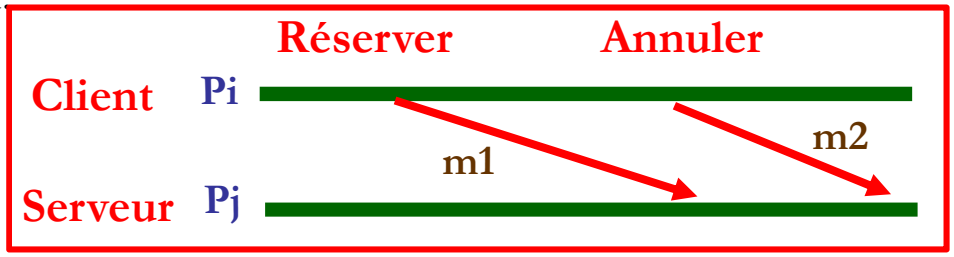
Réception vs Délivrance



- **La délivrance d'un message :** l'opération consistant à le rendre accessible aux applications clientes.

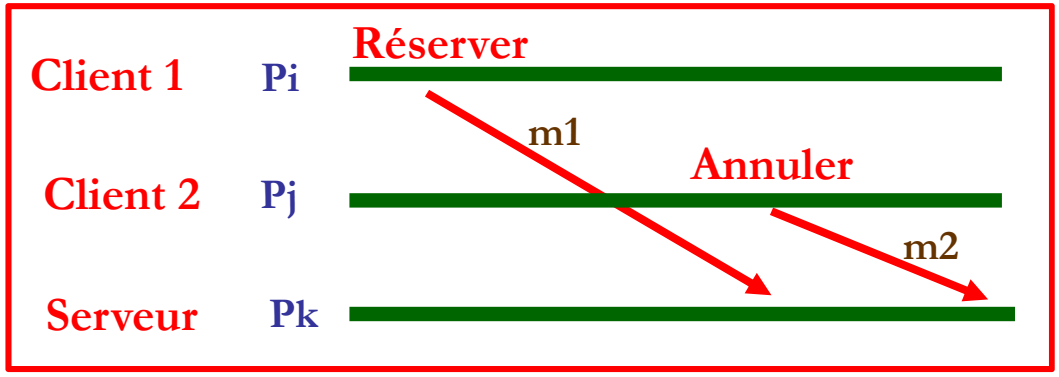
Délivrance FIFO vs Délivrance Causale

- **Délivrance FIFO :** cette propriété assure que si deux messages sont envoyés successivement de Pi vers un même destinataire Pj, le premier sera délivré à Pj avant le second.



Exemple : communication avec les sockets TCP

- **Délivrance Causale :** cette propriété étend la précédente à des communications à destination d'un même processus en provenance de plusieurs autres.
- Elle assure que si l'envoi du message m1 par Pi à destination de Pk précède (causalement) l'envoi du message m2 par Pj à destination de Pk, le message m1 sera délivré avant le message m2 à Pk.



M1 Réseaux et Systèmes Distribués
Algorithmique des systèmes et applications réparties
TD n°1 2019/2020

Soit 4 processus interconnectés entre eux via des canaux et qui exécutent les séquences de pseudo-code suivantes :

Processus P1	Processus P2	Processus P3	Processus P4
1. x = 1 2. send(x, P2) 3. x = x * 2 4. x = receive(P2)	1. y = receive(P1) 2. y = y / 2 3. send(y, P4) 4. y = receive(P3) 5. send(y, P1)	1. x = 4 2. x = receive(P4) 3. x = 2 + x 4. send(x, P2)	1. z = 3 2. z = receive(P2) 3. send(z, P3)

send (nb, Px) envoie la valeur de l'entier nb au processus Px.
nb = receive(Px) attend un message contenant un entier de la part du processus Px.
L'entier reçu est placé dans nb.

Partie 1 : l'horloge de Lamport

1.1 Dessinez le chronogramme correspondant à l'exécution en parallèle des 4 processus.

1.2 Datez chacun des événements en utilisant la méthode de l'horloge de Lamport.

1.3 Donnez l'ordre total global défini par la datation via la méthode de l'horloge de Lamport.

Partie 2 : l'horloge de Mattern

Dessinez dans le chronogramme précédent, une coupure qui se produira après le **3 ème événement** dans P1, le **3 ème événement** dans P2, le **2 ème événement** dans P3 et le **2 ème événement** dans P4 respectivement.

2.1 Serait-il possible de savoir si la coupure est cohérente ou pas sans passer par l'horloge de Mattern ? Voir la définition formelle d'une coupure cohérente.

2.2 Datez chacun des événements en utilisant la méthode de l'horloge de Mattern.

2.3 Quelle est la signification des valeurs liées à l'événement e32 ?

2.4 Selon l'horloge de Mattern, quelle est la relation entre e33 et e25 ? Même question pour e13 et e42.

2.5 Donnez la coupure et l'état de la coupure.

2.6 Datez la coupure à l'aide de l'horloge de Mattern.

2.7 Selon Mattern, la coupure est elle cohérente ? Expliquez. La reprise après panne est-elle possible pour ce système distribué ?

Algorithmique des systèmes et applications réparties
M1 RSD 2019/2020 - TP N° 2 (Sockets)

Sommaire :

Exercice 1 : Multicast et Broadcast avec UDP

Le premier octet d'une adresse IP multicast commence toujours par les 4 bits **'1110'** suivi par 28 bits (adresse du groupe multicast). **1110**0000 => **224 (valeur minimale).** **1110**1111 => **239 (valeur maximale).**
Le protocole IP utilise les adresses (virtuelles) de : 224.0.0.0 à 239.255.255.255 pour le multicast.

Examinez dans ce qui suit le lien suivant :
https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html

- Quel est le rôle de cette classe ? Examinez en particulier les deux méthodes joinGroup et leaveGroup.
- Soit la classe java suivante. Lancez une exécution. Que fait ce code ?
- Mettre la ligne socket.joinGroup(group); en commentaire. Pourquoi ça ne marche pas ?

```
import java.io.*;
import java.net.*;
public class Multicast {
 public static void main(String argv [ ]) throws IOException{
        String msg = "JE SUIS ETUDIANT EN INFORMATIQUE";
        InetAddress group = InetAddress.getByName("230.0.0.0");
        MulticastSocket socket = new MulticastSocket(1000) ;
        socket.joinGroup(group);
        DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(),group, 1000);
        socket.send(hi);
        byte[] buf = new byte[1024];
        DatagramPacket recv = new DatagramPacket(buf, buf.length);
        socket.receive(recv);
        String ch= new String (recv.getData());
        System.out.println(ch);
        socket.leaveGroup(group);
 }
}
```

- Reprendre la classe Entreprise vue en cours, créer un objet de cette classe et envoyer le par MulticastSocket. Bien évidemment il faut aussi le recevoir.

- Examinez le code suivant :

```
import java.net.*;
import java.io.*;
public class Multicast {
      public static void main(String[] args) throws IOException {
       DatagramSocket socket = new DatagramSocket(5000);
       socket.setBroadcast(true);
       InetAddress address = InetAddress.getByName("255.255.255.255");
       byte[] buffer = "RSDGL".getBytes();
       DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, 5000);
       socket.send(packet);
       byte[] buf = new byte[1024];
       DatagramPacket recv = new DatagramPacket(buf, buf.length);
       socket.receive(recv);
       String ch= new String (recv.getData());
       System.out.println(ch);
       socket.close();
   }
}
```

- Quel est le rôle de la méthode setBroadcast ? Testez votre code avec le paramètre false.

- Quelle est la différence principale entre le Multicast et le Broadcast (vu en cours) ?

Exercice 2 : Manipulation de plusieurs types avec UDP

Soit les deux classes java suivantes :

```
import java.io.*;
import java.net.*;
public class ClientUDPint{
public static void main(String[] args) throws Exception {
    DatagramSocket socket = new DatagramSocket();                    // ligne 5
    InetAddress serveur = InetAddress.getByName("localhost");
        ByteArrayOutputStream a = new ByteArrayOutputStream();
        DataOutputStream b = new DataOutputStream(a);
        // Ecrire : 10 true bonjour 1.2 dans le outputstream
        b.writeInt(10);
        b.writeBoolean(true);
        b.writeUTF("bonjour");
        b.writeDouble(1.2);
       byte[] buffer = a.toByteArray();
       DatagramPacket packet = new DatagramPacket(buffer,buffer.length,serveur ,2000);
       socket.send(packet);
       System.out.println("Client a envoyé 10 true bonjour 1.2 au serveur");
}
}
```

```
import java.io.*;
import java.net.*;
public class ServeurUDPint {
   public static void main(String[] args) throws Exception {
       DatagramSocket socket = new DatagramSocket(2000);
      DatagramPacket packet = new DatagramPacket(new byte[1024] , 1024);
      socket.receive(packet);
      byte[] data = packet.getData();
      ByteArrayInputStream a = new ByteArrayInputStream(data);
       DataInputStream b = new DataInputStream(a);
       System.out.println("Entier recu : "+b.readInt());
       System.out.println("Boolean recu : "+b.readBoolean());
       System.out.println("String recu : "+b.readUTF());
       System.out.println("Double recu : "+b.readDouble());

}
}
```

- Lancez l'exécution des deux classes précédentes.
- Quel est l'intérêt de DataOutputStream et DataInputStream pour UDP ?
- Examinez la documentation de DatagramSocket dans le lien suivant https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html et affichez la taille de la zone tampon (buffer) utilisée pour recevoir et pour envoyer les données.
- Dans le DatagramPacket du client, modifiez le numéro de port de 2000 à 3000 et lancez l'exécution des deux classes précédentes. Il n'y a pas d'erreurs. Pourquoi ?
- Reprendre le DatagramPacket du client avec la valeur 2000 pour le numéro de port et ajoutez dans la classe ClientUDPint.java, les deux instructions suivantes après la ligne 5 :

  ```
  System.out.println("PORT LOCAL "+socket.getLocalPort());
  System.out.println("PORT DISTANT "+socket.getPort());
  ```
- Faites plusieurs exécutions de cette classe. Comment se fait le choix du port local par le SE ?
  Le port distant affiche toujours -1.
- En gardant les deux affichages précédents. Ajoutez l'instruction suivante après la ligne 5 (devant les deux affichages). socket.connect(new InetSocketAddress ("localhost",2000)); et lancez l'exécution des deux classes précédentes. L'exécution se déroule normalement.
- Modifiez l'instruction précédente par : socket.connect(new InetSocketAddress ("localhost",1000)); Faites une nouvelle exécution. **Le code précise le numéro de port 1000 pour connect et 2000 pour DatagramPacket.**
- Quelle est finalement la différence entre connect de TCP et connect de UDP ? Voir si c'est bloquante ou pas. UDP est un protocole déconnecté et malgré cela des exceptions sont levées dans le cas de connect (des erreurs).
- Déclarez maintenant un tableau d'entier **(ex. int [ ] tab = {1, 6, 8, 9, 13, 10};)** coté client. Ce dernier doit l'envoyer par UDP au serveur. Le serveur affichera par la suite son contenu (les entiers avec des sauts de lignes) → pensez à utiliser ObjectOutputStream : **tab est un objet qu'on peut le convertir en octets pour l'envoyer avec UDP selon le modèle vu en cours.**
- Modifiez par la suite votre code afin que le serveur puisse répondre par le sous tableau contenant uniquement les entiers pairs **({6, 8, 10}).**

Exercice 3 : Application client/serveur avec UDP

Dans ce qui suit, nous allons créer les objets suivants coté serveur (qui seront insérés dans un ArrayList) :

```
Joueur a= new  Joueur(10, "MESSI", "FCB");
Joueur b= new  Joueur(9, "BENZEMA", "REAL");
Joueur c= new  Joueur(11, "DEMBELE", "FCB");
Joueur d= new  Joueur(26, "Mahrez","City");
Joueur e= new  Joueur(10, "MODRIC", "REAL");
```

Donc, vous allez commencer par la création de la classe Joueur qui comporte les champs suivants : **numero (int), nom (String) et equipe (String).**

Vous devrez par la suite réaliser les traitements suivants avec UDP :

- Le client envoie un numéro au serveur (ex. 10), le serveur doit répondre par un objet de type ArrayList contenant les joueurs qui portent le numéro 10.
  Le résultat doit être comme ceci : **[numero = 10 nom = MESSI equipe = FCB, numero = 10 nom = MODRIC equipe = REAL].**
- Le client envoie le nom d'une équipe au serveur (ex. FCB), le serveur doit répondre par un objet de type ArrayList contenant les joueurs de cette équipe. Si vous utilisez un buffer de taille supérieur à la taille du mot reçu (coté serveur), utilisez la méthode **trim()** après la conversion du buffer d'octets en **String** pour supprimer les espaces liés aux cases non remplies dans le buffer de réception.
  Le résultat attendu est comme suit : **[numero = 10 nom = MESSI equipe = FCB, numero = 11 nom = DEMBELE equipe = FCB].**
- Affichez coté client les joueurs dont le nom commence par 'M'. Suite à la requête du client le résultat attendu sera comme suit : **[numero = 10 nom = MESSI equipe = FCB, numero = 26 nom = Mahrez equipe = City, numero = 10 nom = MODRIC equipe = REAL].**

Exercice 4 : Application client/serveur avec TCP

Soit la classe java suivante (**Etudiant.java**), elle représente les caractéristiques d'un étudiant **(son nom, sa spécialité et sa moyenne générale).**

```
import java.io.Serializable;
public class Etudiant implements Serializable{
  String nom;
  String specialite;
  int moy;
  Etudiant (String nom, String specialite, int moy) {
    this.nom = nom;
    this.specialite = specialite;
    this.moy = moy;
  }
    String getNom() {
    return nom;
  }
  public String toString() {
    return "Etudiant : "+nom+"   "+specialite+" : "+moy;
  }
}
```

Le serveur **(ServerEtudiant.java)** contient 3 étudiants, il récupère le nom d'un étudiant (envoyé par le client), cherche cet étudiant dans son tableau et envoie l'objet étudiant correspondant au client.
Pour lire le nom, le serveur utilise **readLine** de **BufferedReader** mais pour envoyer l'objet, il doit passer par **writeObject** de **ObjectOutputStream.**

```
import java.net.*;
import java.io.*;
class ServerEtudiant {
  public static void main(String args[]) {
    Etudiant[ ] tabEtudiant = {new Etudiant ("A", "GL", 13),new Etudiant ("B", "RSD", 12),new Etudiant ("C", "SIC", 14)};
    ServerSocket server = null;
    try {
      server = new ServerSocket(7777);
      while (true) {
        Socket sock = server.accept();
        System.out.println("connecte");
        ObjectOutputStream sockOut = new  ObjectOutputStream(sock.getOutputStream());
        BufferedReader sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
        String recu;  while ((recu = sockIn.readLine()) != null) {
          System.out.println("recu :"+recu);
          String nom = recu.trim();
          for (int i=0; i<tabEtudiant.length; i++)
            if (tabEtudiant[i].getNom().equals(nom)) { sockOut.writeObject(tabEtudiant[i]);break; }
        }
        sockOut.close();
        sock.close();
      } } catch (IOException e) {
      try {server.close();} catch (IOException e2) {}
    }
  }// fin main
  }// fin classe
```

Le client **(ClientEtudiant.java)** utilise **println** de **PrintWriter** pour envoyer le nom de l'étudiant au serveur et récupère l'objet étudiant (envoyé par le serveur) avec **readObject()** de **ObjectInputStream.**

```
import java.io.*; import java.net.*;
public class ClientEtudiant {
  public static void main(String[] args) throws IOException {
 String hostName = "localhost";
String NomEtudiant = "A";
    Socket sock = null;
    PrintWriter sockOut = null;
    ObjectInputStream sockIn = null;
    try {
      sock = new Socket(hostName, 7777);
      sockOut = new PrintWriter(sock.getOutputStream(), true);
      sockIn = new ObjectInputStream(sock.getInputStream());
    } catch (UnknownHostException e) {System.err.println("host non atteignable : "+hostName);  System.exit(1); }
       catch (IOException e) {System.err.println("connection impossible avec : "+hostName); System.exit(1);}
        sockOut.println(NomEtudiant); // envoyer le nom au serveur
    try {
      Object recu = sockIn.readObject(); // récupérer l'objet Etudiant envoyé par le serveur
      if (recu == null) System.out.println("erreur de connection");
      else { Etudiant etudiant = (Etudiant)recu;
        System.out.println("serveur -> client : " + etudiant);
      }
    } catch (ClassNotFoundException e) {System.err.println("Classe inconnue : "+hostName); System.exit(1);}
     sockOut.close();
    sockIn.close();
    sock.close();
  }
}
```

Modifier le tableau **tabEtudiant** coté serveur afin d'y avoir 9 étudiants. 3 de chaque spécialité (RSD, SIC et GL). Réalisez par la suite les opérations suivantes :

- Le client se connectera au serveur en envoyant une spécialité (exemple, RSD). Le serveur répondra par un tableau contenant uniquement les étudiants de cette spécialité.

- Le client se connectera au serveur en envoyant un entier (entre 1 et 19). Le serveur répondra par un tableau contenant tous les étudiants ayant une moyenne supérieure à cet entier.

- Le client se connectera au serveur en envoyant le mot 'trier'. Le serveur répondra par le tableau trié de tous les étudiants selon l'ordre décroisant de leurs moyennes.