Plan

- **Partie 1 : Temps dans un système distribué**
  - Temps logique
  - Chronogramme
  - Dépendance causale
  - Parallélisme logique
- **Partie 2 : Horloges de Lamport**
- **Partie 3 : Exclusion mutuelle basée sur les Horloges de Lamport**

Source : Cours Système Distribués de Badr BenMammar

Partie 1 :

Temps dans un système distribué

Source : Cours Système Distribués de Badr BenMammar

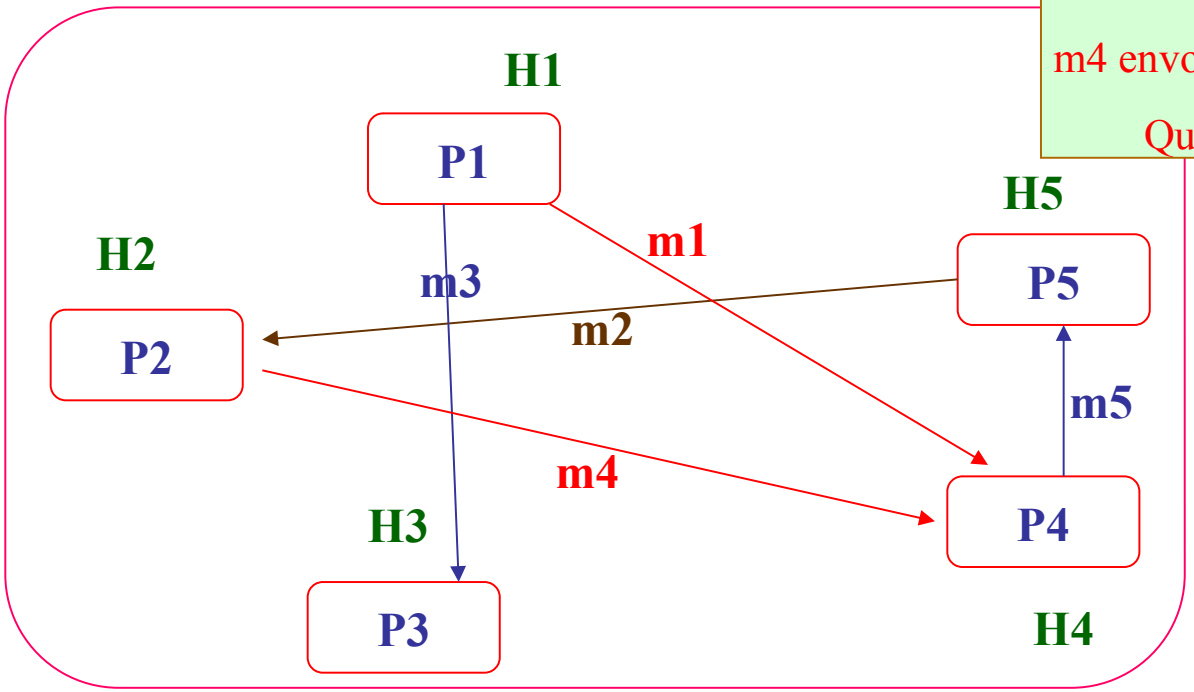
Temps dans un système distribué

- **Objectif :** définir un **temps global** cohérent et « identique » (ou presque) pour tous les processus.
  - Créer **un temps logique.**
    - Temps qui n'est pas lié à un temps physique mais à une relation de cause → effet.

m1 envoyé par P1 vers P4 à l'instant t

m4 envoyé par P2 vers P4 à l'instant t'

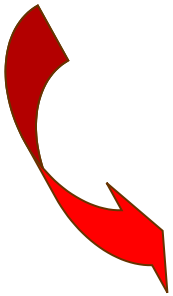
Quel est le lien entre t et t' ?



Source : Cours Système Distribués de Badr BenMammar

Temps logique

- **Exemple d'application :** 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.**

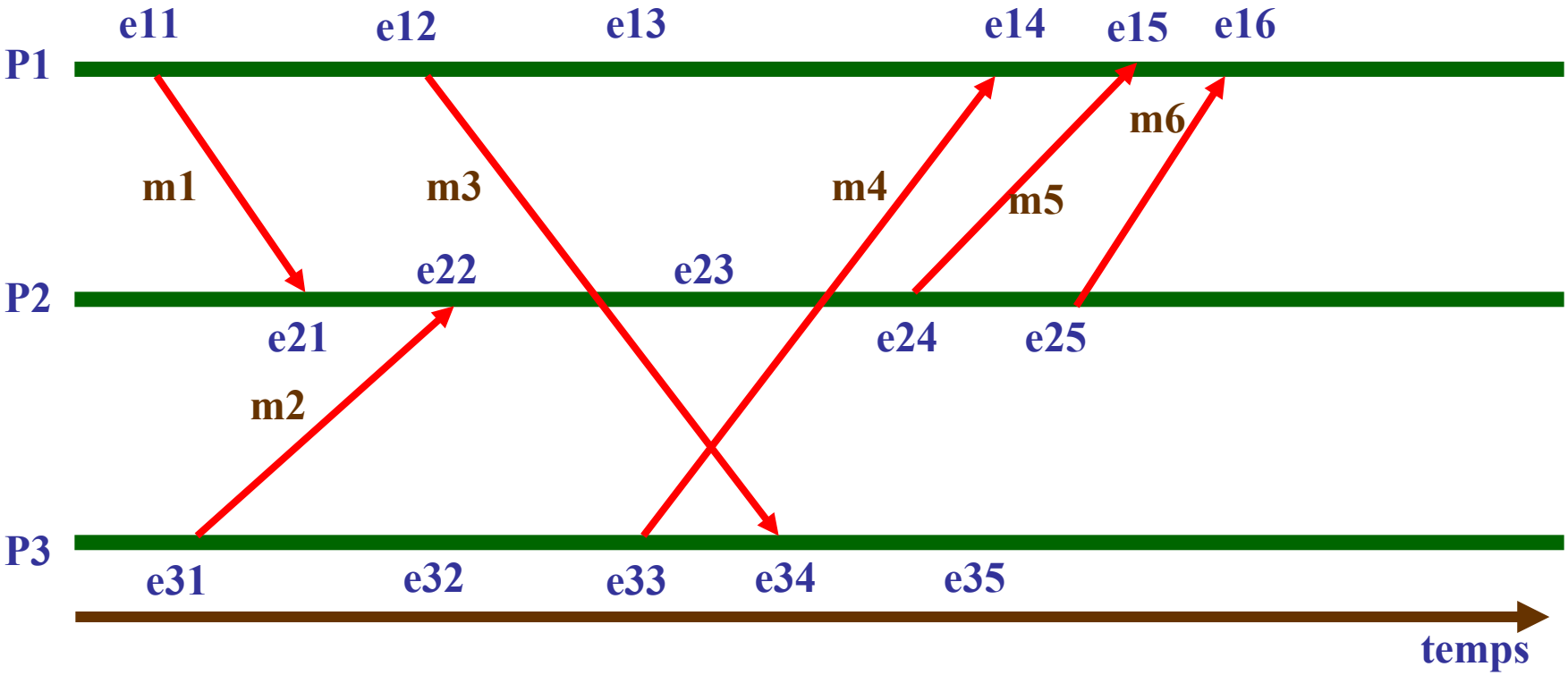


- **Utiliser une horloge logique.**

Source : Cours Système Distribués de Badr BenMammar

Chronogramme

- **Exemple :** trois processus tous reliés entre eux par des canaux, les messages ne peuvent pas se perdre.
- **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.



Source : Cours Système Distribués de Badr BenMammar

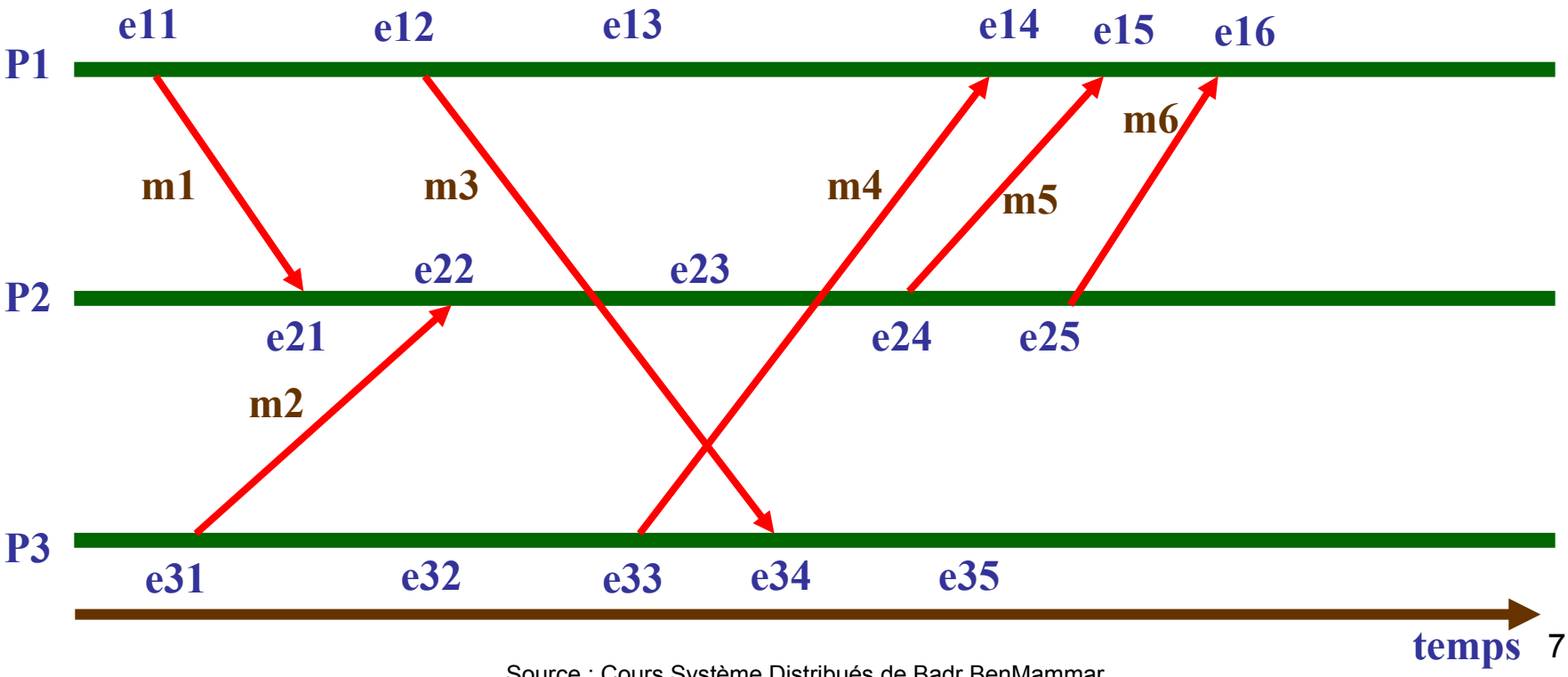
Chronogramme

- **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** dans l'évolution du processus.
- Les messages échangés se font sans perte, duplication ou ré-ordonnancement (les canaux de communication inter-processus sont des FIFO)

Source : Cours Système Distribués de Badr BenMammar

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 en e24, message m6 envoyé en e25.



Source : Cours Système Distribués de Badr BenMammar

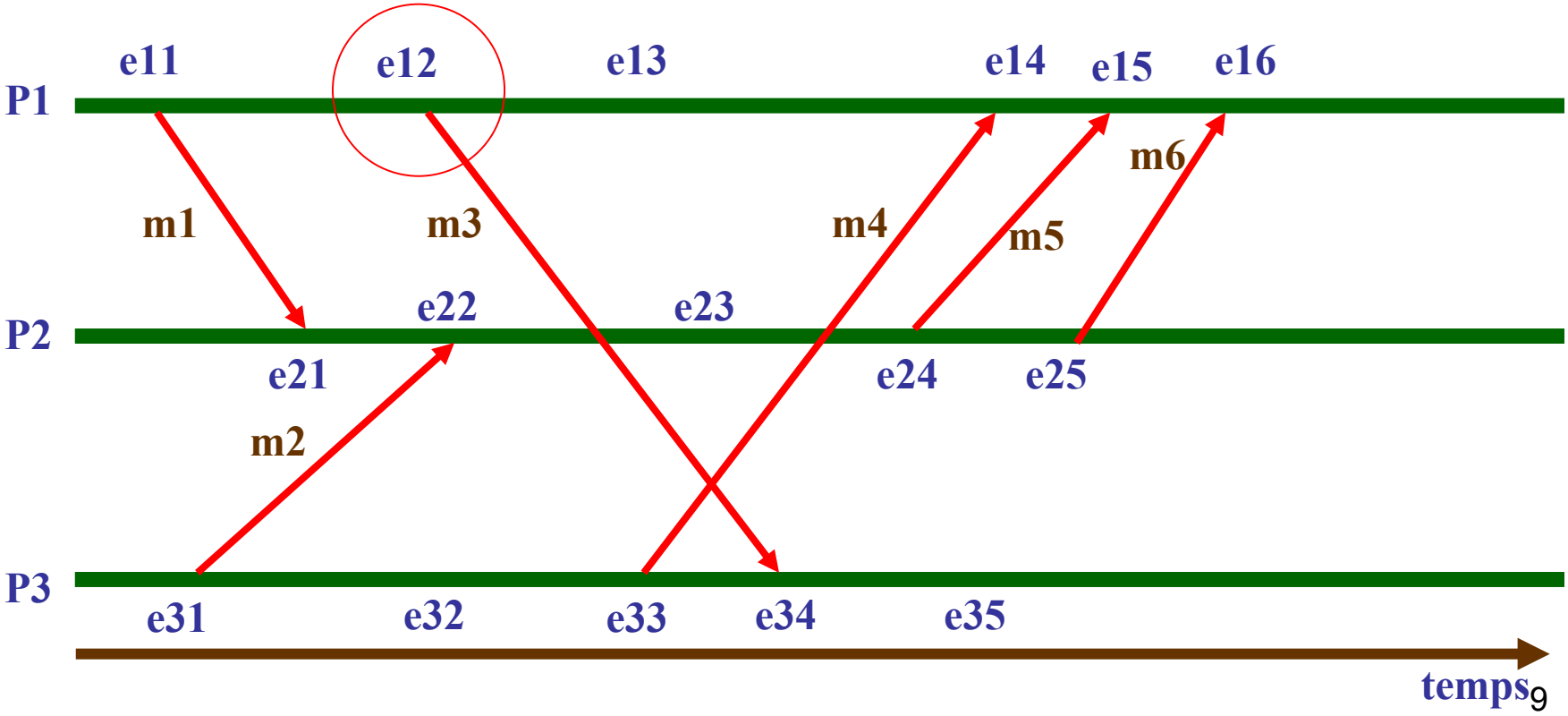
Dépendance causale

- **Relation de dépendance causale :** Il y a une dépendance causale entre 2 événements si un événement **doit avoir lieu avant l'autre.**
- **Notation :** e → e'.
  - *e* doit se dérouler avant *e'*.
- Si e → e', alors une des trois conditions suivantes doit être vérifiée pour *e* et *e'*.
  - Si *e* et *e'* sont des événements d'un même processus, *e* précède localement *e'*.
  - Si *e* est l'émission d'un message, *e'* est la réception de ce message.
  - Il existe un événement *f* tel que $e \rightarrow f$ et $f \rightarrow e'$.
- **Ordonnancement des événements :**
  - Les dépendances causales définissent des **ordres partiels** pour des ensembles d'événements du système.

Source : Cours Système Distribués de Badr BenMammar

Dépendance causale

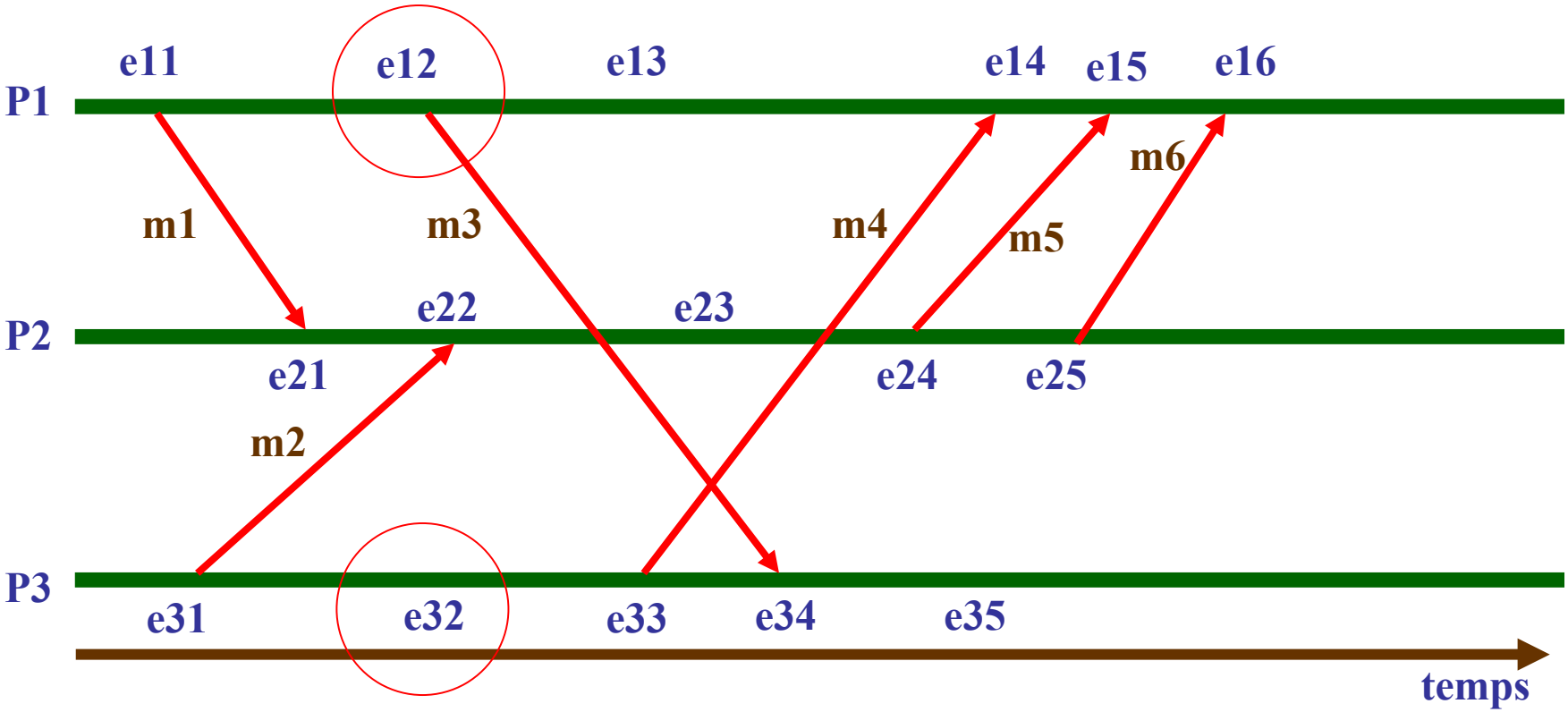
- Sur l'exemple précédent, quelques dépendances causales autour de e12.
- Localement : e11 → e12, e12 → e13.
- Sur message : e12 → e34.
- Par transitivité : e12 → e35 (car e12 → e34 et e34 → e35).
  - e11 → e13 (car e11 → e12 et e12 → e13).



Source : Cours Système Distribués de Badr BenMammar

Dépendance causale

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



Source : Cours Système Distribués de Badr BenMammar

Parallélisme logique

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

Négation de la dépendance causale.

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

Source : Cours Système Distribués de Badr BenMammar

Partie 2 : Horloge de Lamport

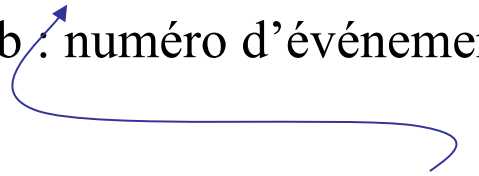
Principe : datation de chacun des événements du système avec respect des dépendances causales entre événements.

But : Ordonner totalement les évènements d'un système distribué, sans utiliser d'horloge physique

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

- Introduit en 1978 par **Leslie Lamport.**
  - **https://lamport.azurewebsites.net/pubs/time-clocks.pdf**
- 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.

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

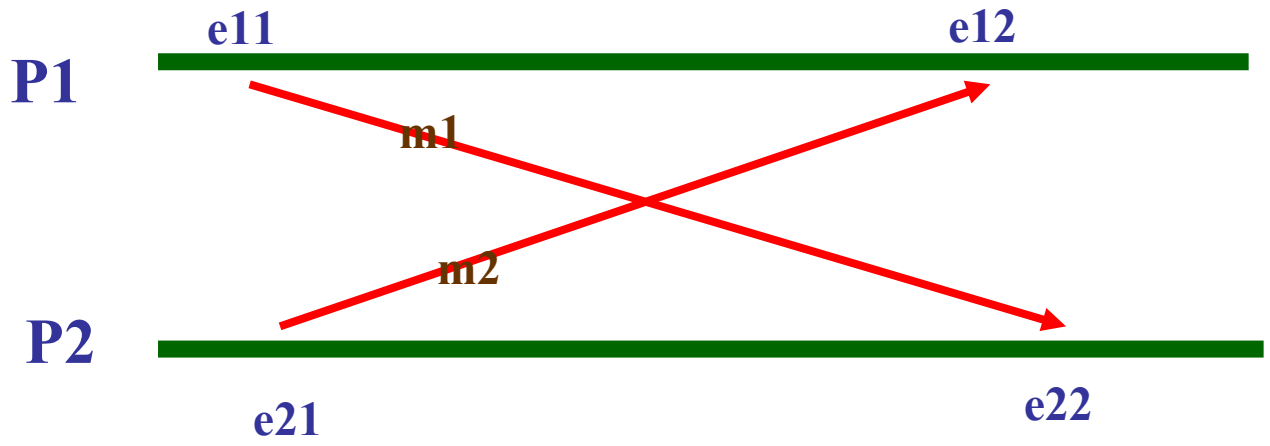
- **Création du temps logique :**
  - Localement, chaque processus Pi possède une horloge locale logique **nb**, initialisée à 0.
    - Sert à dater les événements.
  - **Pour chaque événement local de Pi.**
    - **nb = nb + 1** : on incrémente l'horloge locale.
    - L'événement est daté localement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

Temps de l'horloge locale dans Pj

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport et dépendance causale

- **Exemple 1 :** Echange de messages entre 2 processus.

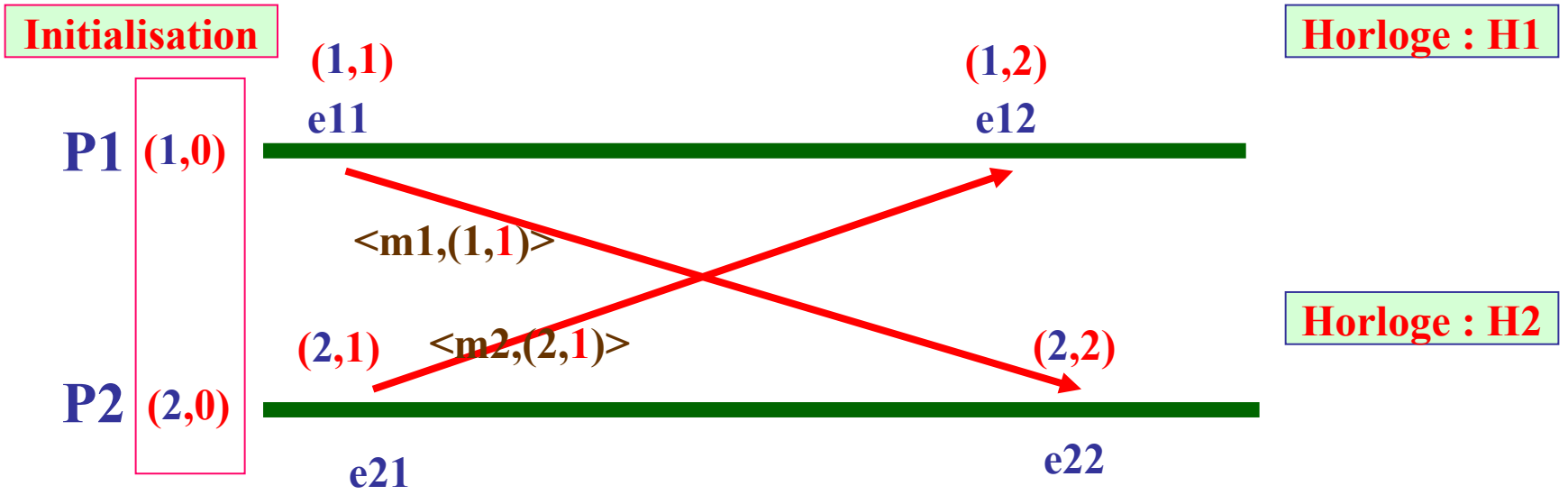


- Les dépendances causales : e11 → e12, e11 → e22, e21 → e22, e21 → e12.
- Absence de dépendance causale entre e11 et e21.
  - **Parallélisme logique entre e11 et e21.**
- **L'horloge de Lamport H(x) respecte la dépendance causale :**
  - **(e → e') ⇒ (H(e) < H(e')).**
  - **Exemple : (e11 → e12) ⇒ (H(e11) < H(e12)).**

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

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



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

(1,1) **(1,2)**

(2,1) **(2,2)**

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

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

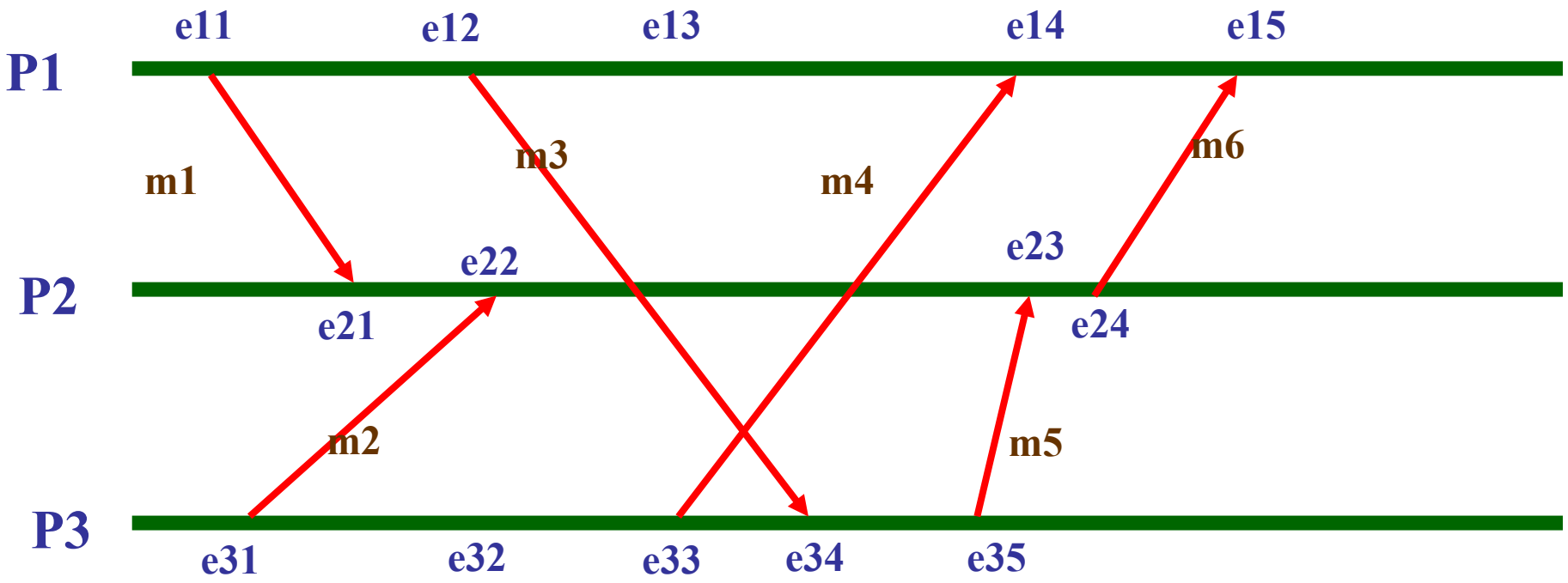
- **Ordonnancement global : e11, e21, e12, e22.**
- **H(e11) < H(e21) < H(e12) < H(e22).**
- L'horloge de Lamport respecte la dépendance causale :
  - **(e → e') ⇒ (H(e) < H(e')).**
- Selon l'horloge : **H(e11) < H(e21).**
- Pourtant, il y a une absence de dépendance causale entre e11 et e21.

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

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

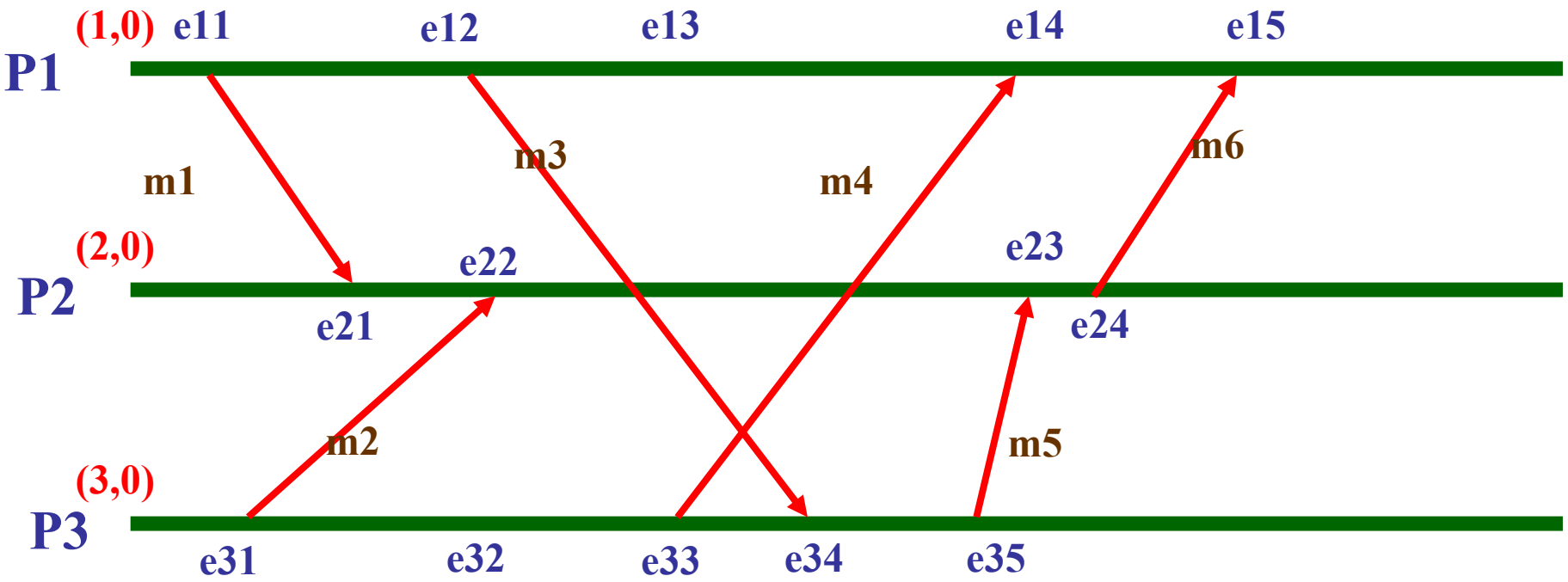
- **Exemple 2 :** chronogramme avec ajouts des estampilles.



Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

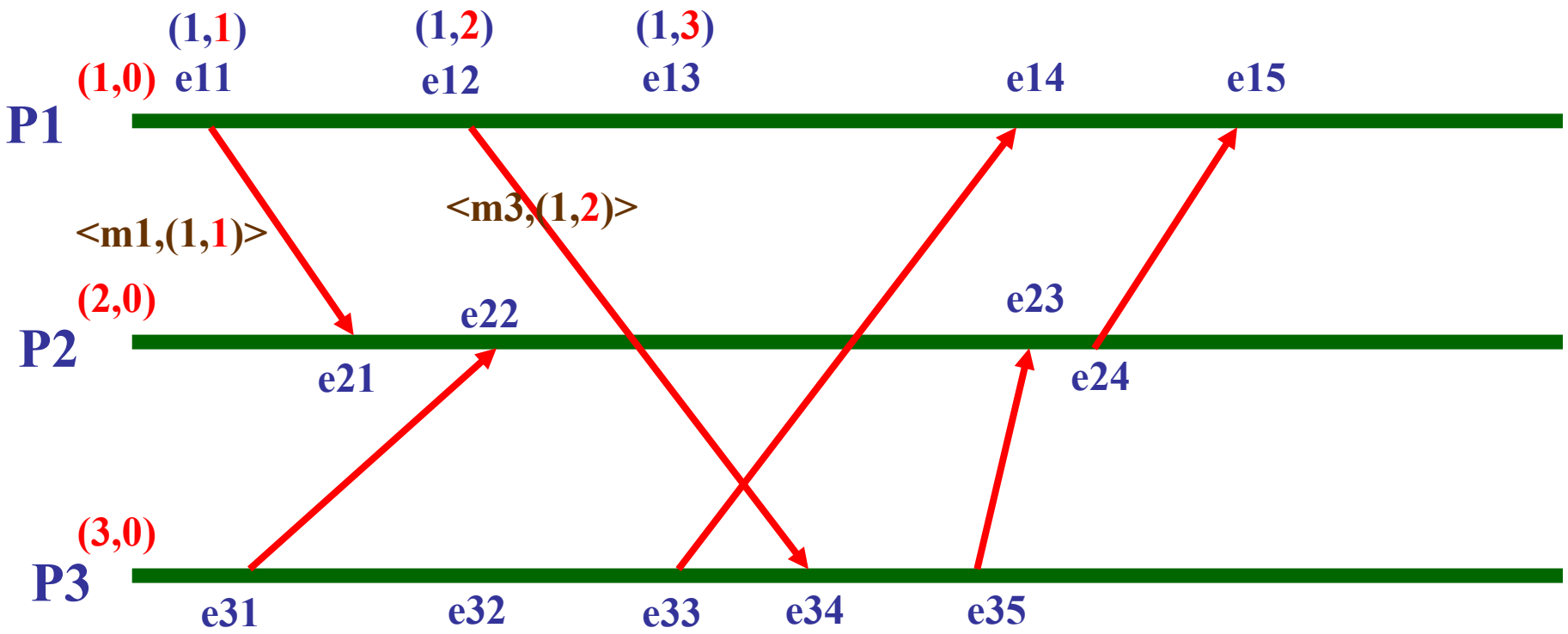
- **Exemple 2 :** chronogramme avec ajouts des estampilles.



Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

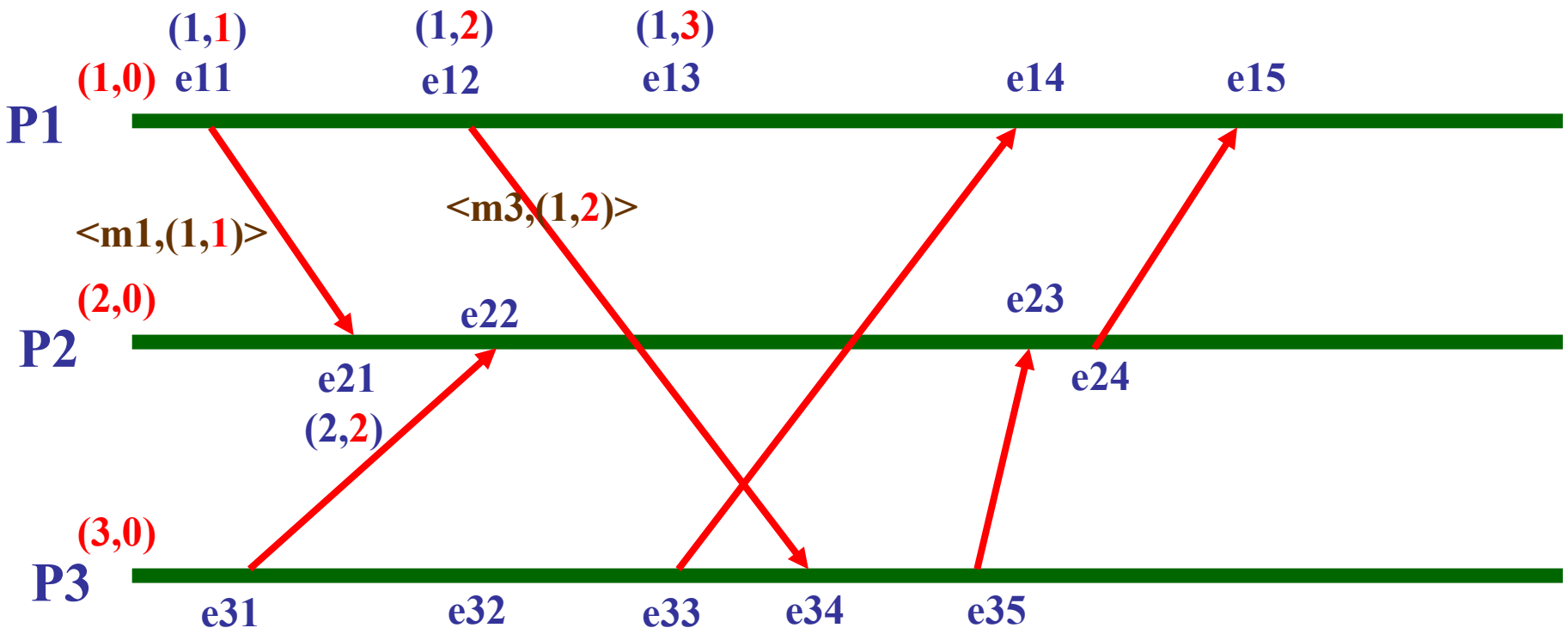
- **Exemple 2 :** chronogramme avec ajouts des estampilles.



Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

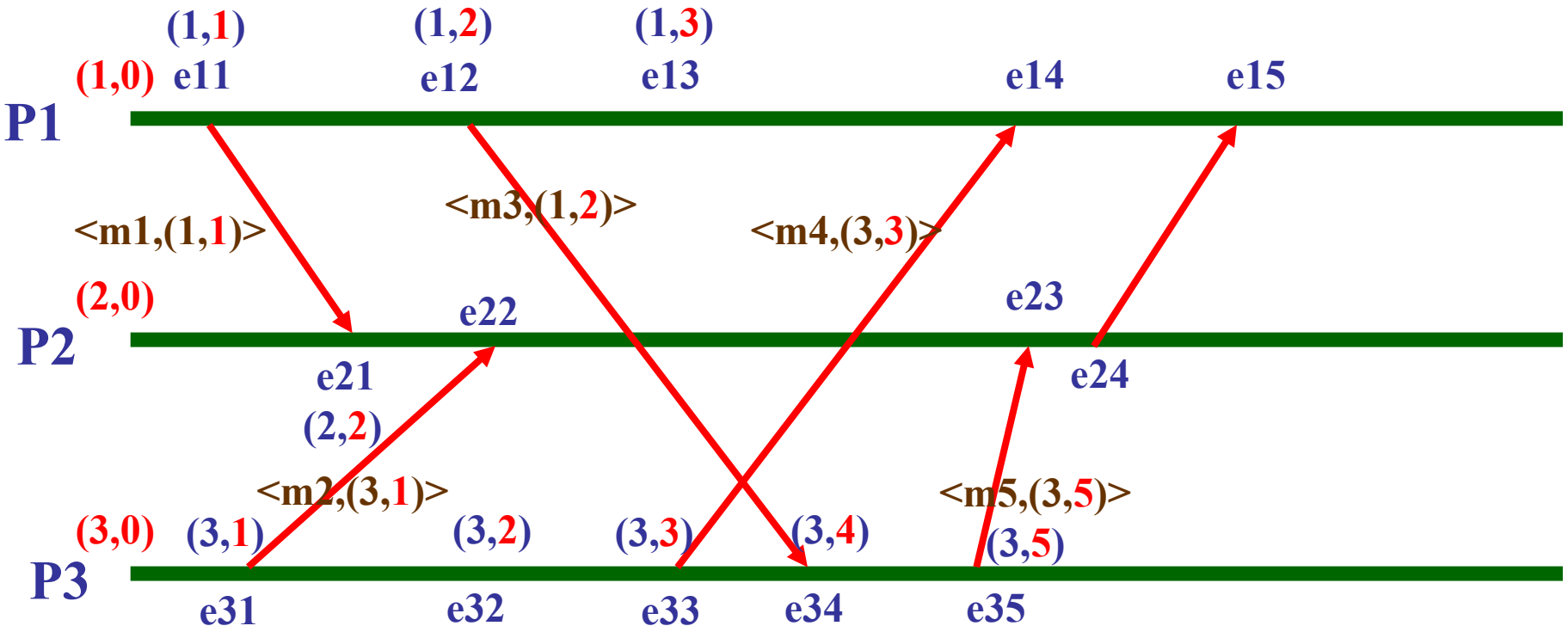
- **Exemple 2 :** chronogramme avec ajouts des estampilles.



Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

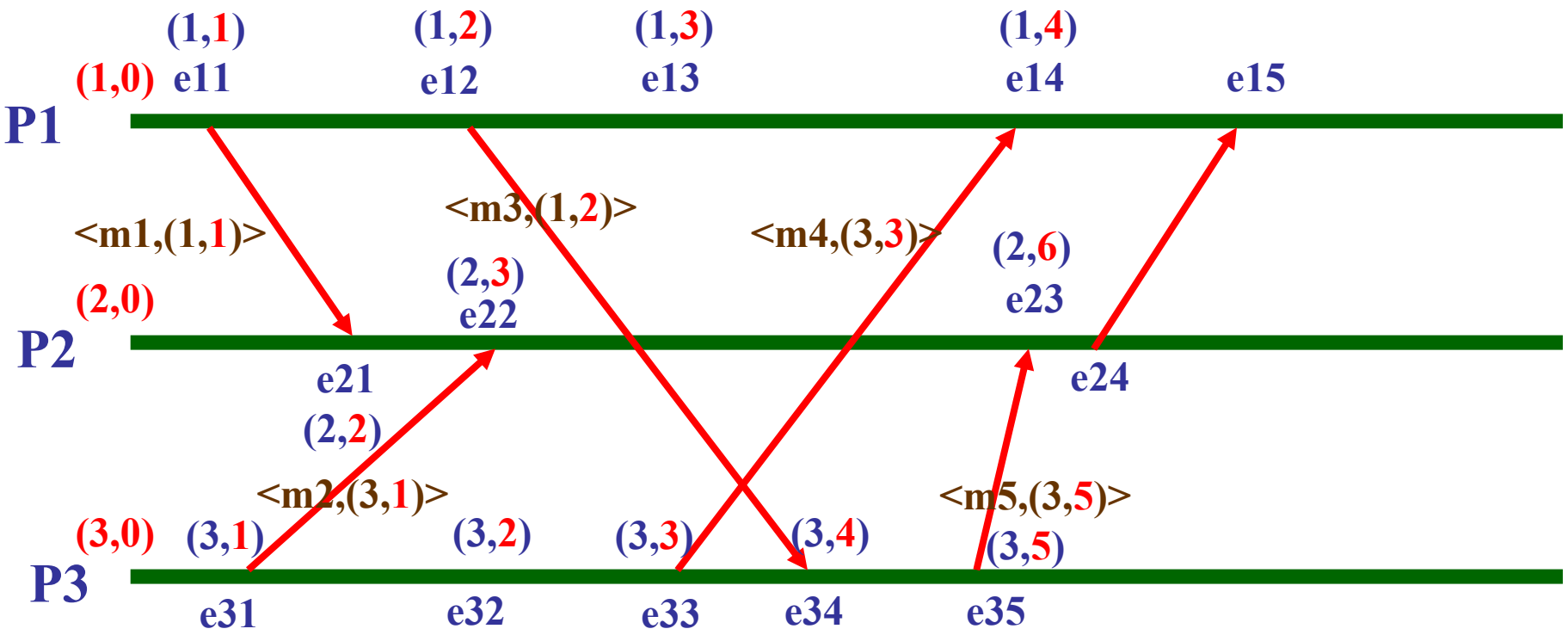
- **Exemple 2 :** chronogramme avec ajouts des estampilles.



Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

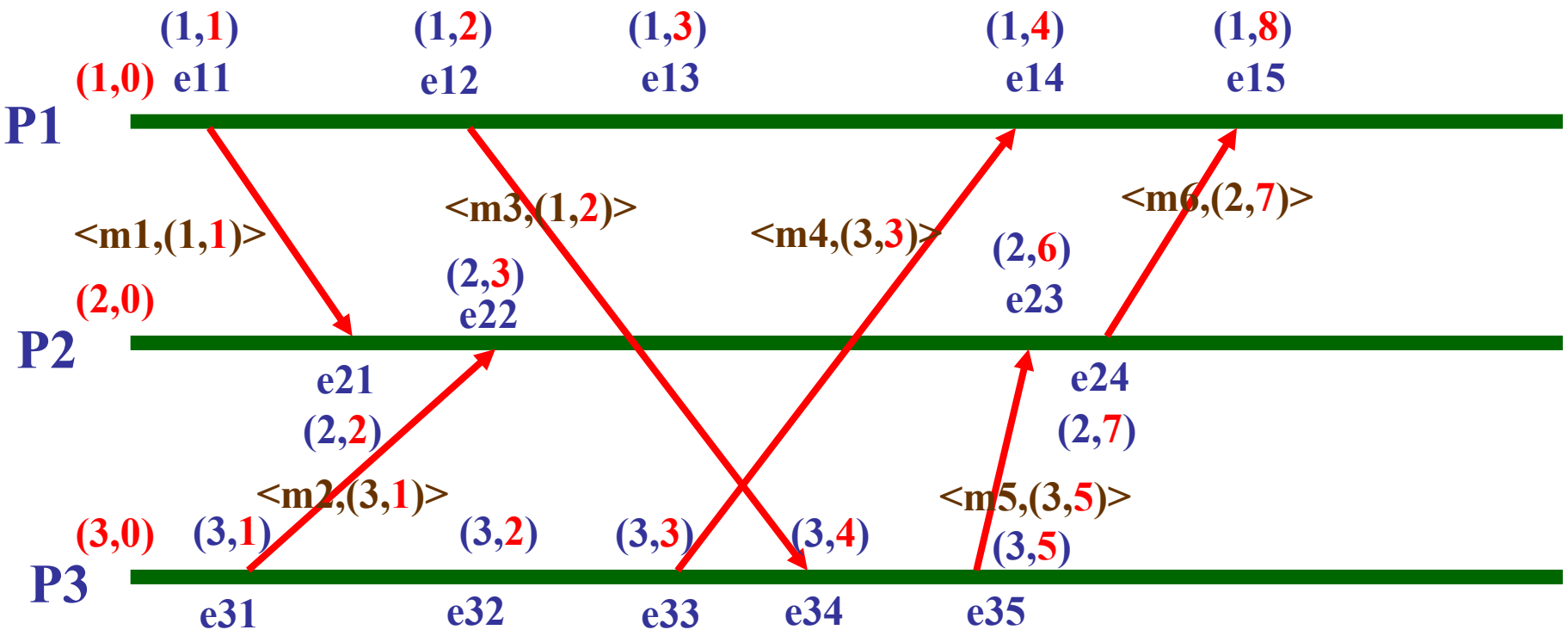
- **Exemple 2 :** chronogramme avec ajouts des estampilles.



Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

- **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).

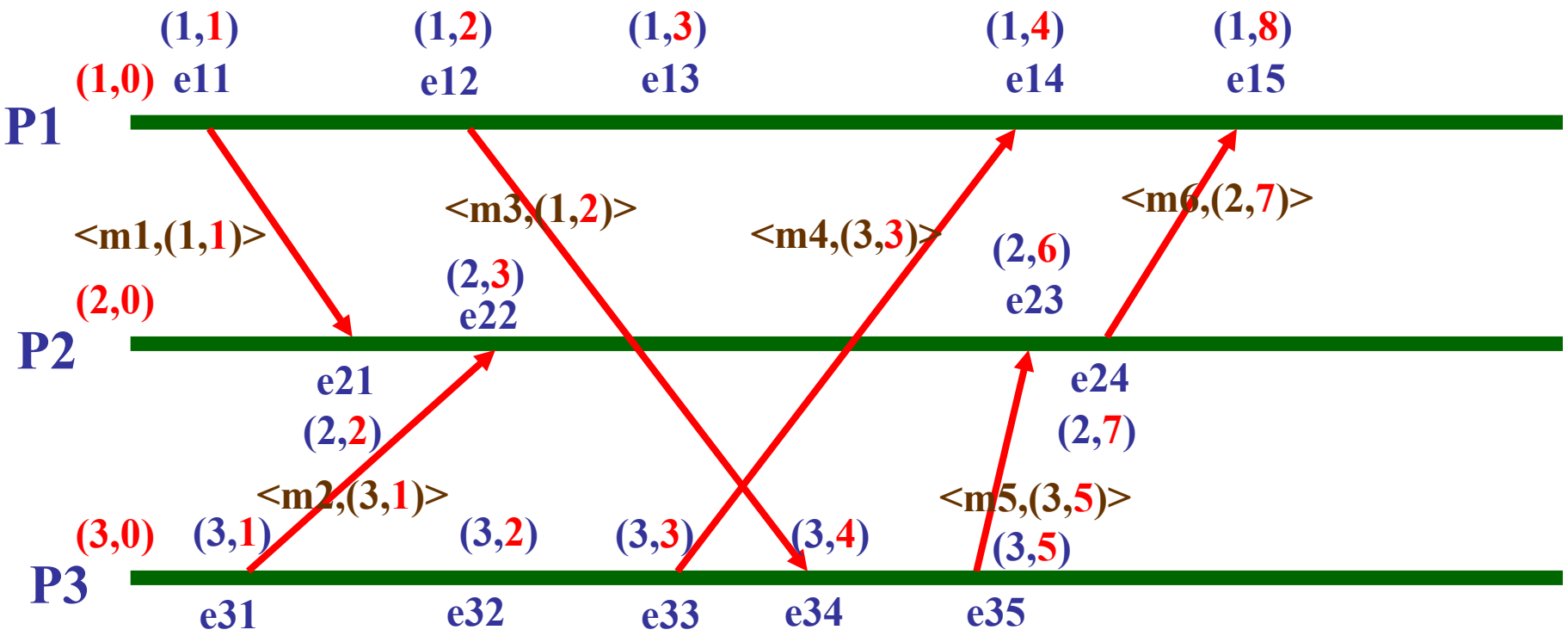
Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport

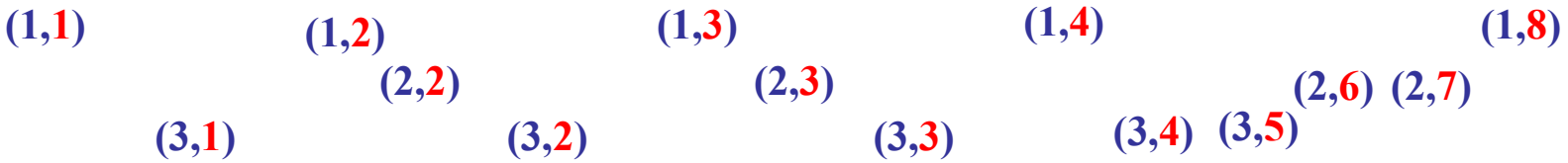
- **Ordonnancement global :**
  - Via Hi, on ordonne tous les événements du système entre eux.
- **Ordre total, noté e << e' : *e* s'est déroulé avant *e'*.**
- Soit *e* événement de *Pi* et *e'* événement de *Pj* :
  - **e << e' ⇔ (Hi(e) < Hj(e')) ou (Hi(e) = Hj(e') avec i < j).**
- Localement (si i = 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.

Source : Cours Système Distribués de Badr BenMammar

Horloge de Lamport



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



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

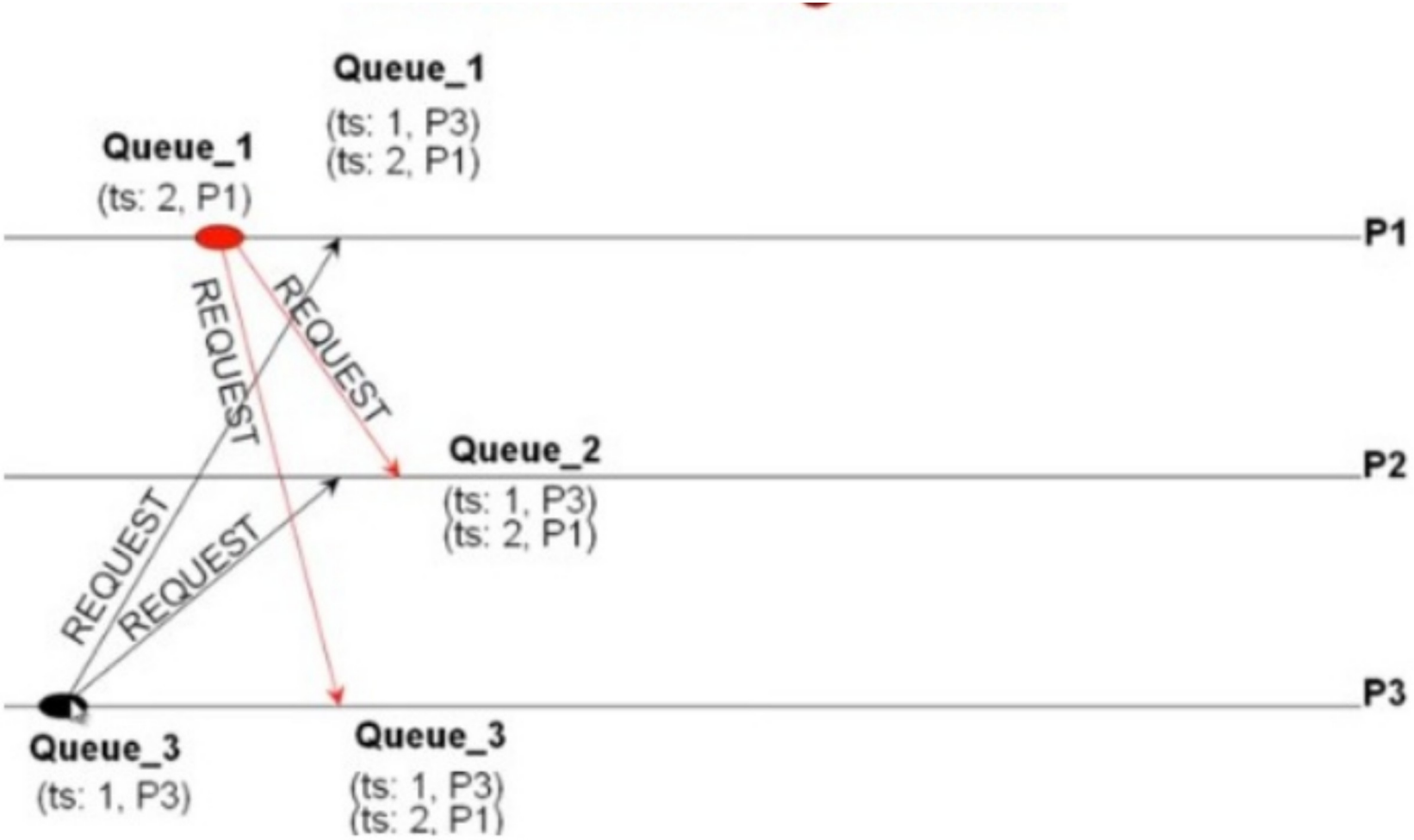
Source : Cours Système Distribués de Badr BenMammar

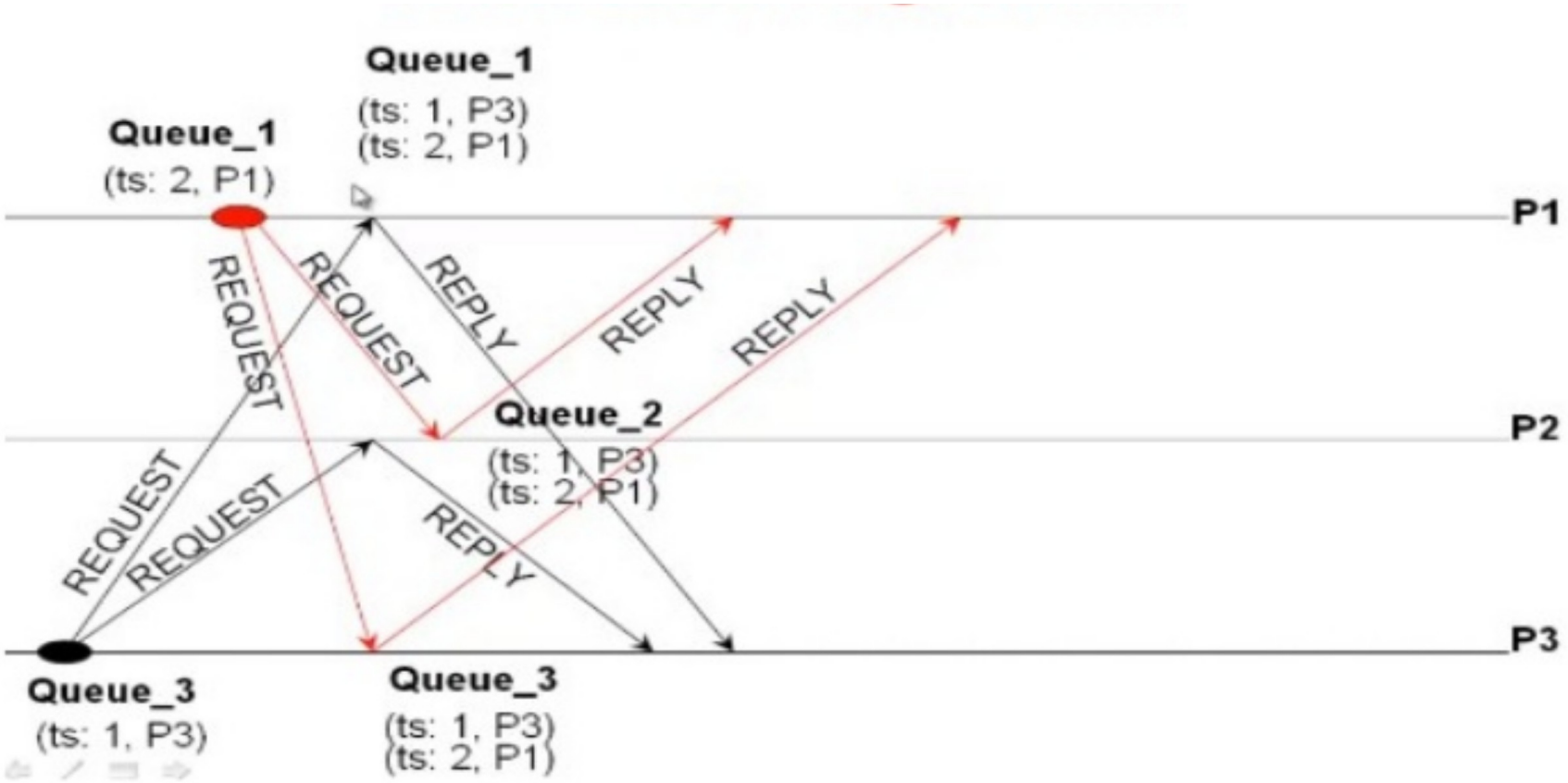
Partie 3 : Exclusion mutuelle basée sur les Horloge de Lamport

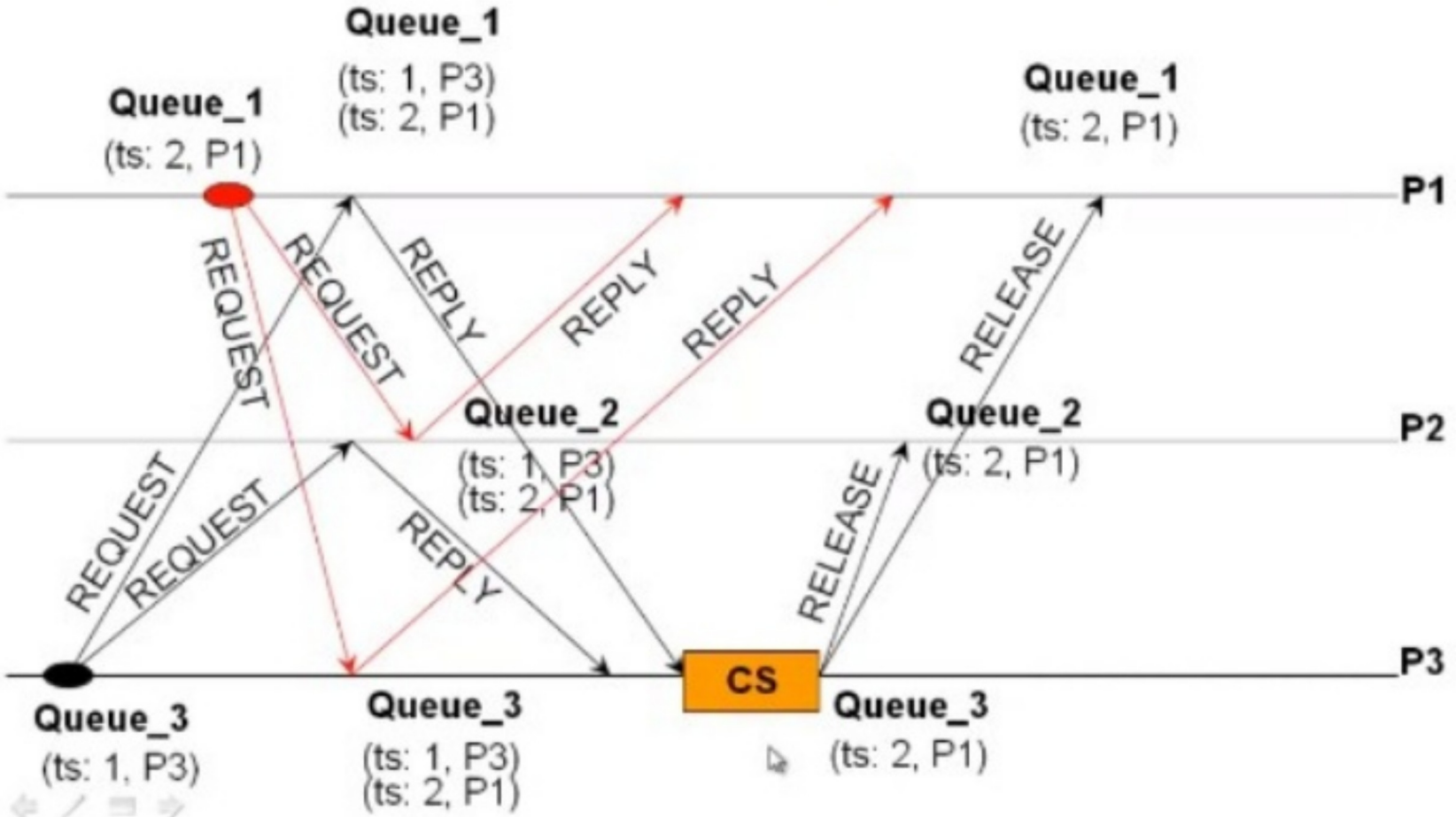
Algorithme

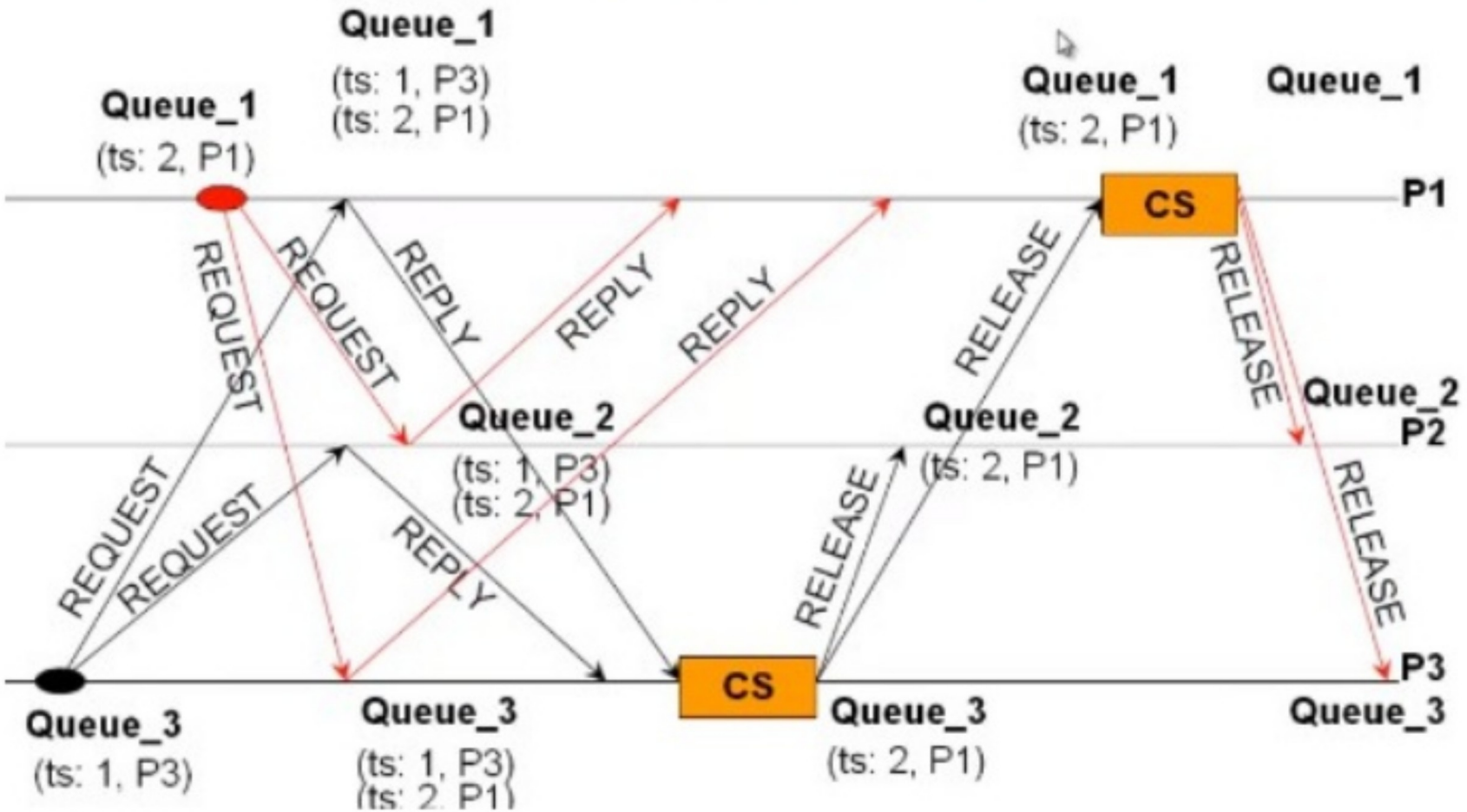
Chacune des 5 actions suivantes est considérée comme un évènement, On suppose que les files d'attente de requêtes contiennent le message (*T_0:P_0, Request*) indiquant que le processus P_0 dispose de la ressource à T_0, plus petit que tous les autres

1. Pour demander une ressource exclusive Pi envoie le message (*Tm:Pi , Request*) a tous les autres processus et mets ce messages dans sa liste d'attente de requêtes. Tm est égal à la valeur de l'horloge de Lamport du message
2. Lorsqu'un processus Pj reçoit le message (*Tm:Pi, Request*) il le met dans sa file d'attente de requêtes et acquitte le message à Pi
3. Pour libérer une ressource, le processus Pi retire tous les messages (*Tm:Pi Requests*) de sa file d'attente de requête et envoie un message (*Tm:Pi, Release*) à tous les processus.
4. Lorsqu'un processus Pj reçoit un message de *Pi Release*, il enlève tous message (*Tm:Pi Request)* de sa file d'attente
5. Le processus Pi reçoit un accès exclusif à la ressource lorsque les deux conditions suivantes sont satisfaites :
   - Il y a un message *(Tm:Pi Request)* dans la file d'attente des requêtes de Pi qui est ordonné avant tous les autres requêtes selon relation de causalité , c'est à dire si Tm est le plus petit de tous les messages de requête.
   - Pi a reçu un message de tous les autres processus qui a un temps logique supérieur à *Tm.*


Queue_1
(ts: 1, P3)
(ts: 2, P1)
Queue_1
(ts: 2, P1)
P1
REQUEST
REQUEST
REQUEST
REQUEST
Queue_2
(ts: 1, P3)
(ts: 2, P1)
P2
P3
Queue_3
(ts: 1, P3)
Queue_3
(ts: 1, P3)
(ts: 2, P1)


Queue_1
(ts: 1, P3)
(ts: 2, P1)
Queue_1
(ts: 2, P1)
P1
REQUEST
REQUEST
REPLY
REPLY
REPLY
Queue_2
(ts: 1, P3)
(ts: 2, P1)
P2
REQUEST
REQUEST
REPLY
P3
Queue_3
(ts: 1, P3)
Queue_3
(ts: 1, P3)
(ts: 2, P1)


Queue_1
(ts: 1, P3)
(ts: 2, P1)
Queue_1
(ts: 2, P1)
Queue_1
(ts: 2, P1)
P1
REQUEST
REQUEST
REPLY
REPLY
REPLY
RELEASE
Queue_2
(ts: 1, P3)
(ts: 2, P1)
Queue_2
(ts: 2, P1)
P2
REQUEST
REQUEST
REPLY
RELEASE
RELEASE
CS
P3
Queue_3
(ts: 1, P3)
Queue_3
(ts: 1, P3)
(ts: 2, P1)
Queue_3
(ts: 2, P1)


Queue_1
(ts: 1, P3)
(ts: 2, P1)
Queue_1
(ts: 2, P1)
Queue_1
(ts: 2, P1)
Queue_1
P1
CS
REQUEST
REQUEST
REPLY
REPLY
REPLY
RELEASE
RELEASE
Queue_2
P2
Queue_2
(ts: 1, P3)
(ts: 2, P1)
Queue_2
(ts: 2, P1)
REQUEST
REQUEST
REPLY
RELEASE
RELEASE
RELEASE
P3
CS
Queue_3
(ts: 1, P3)
Queue_3
(ts: 1, P3)
(ts: 2, P1)
Queue_3
(ts: 2, P1)
Queue_3