SCIENCES SORBONNE UNIVERSITÉ

Exclusion Mutuelle en Réparti

Master informatique SU 2017-2018
UE AR (4I403)

Plan

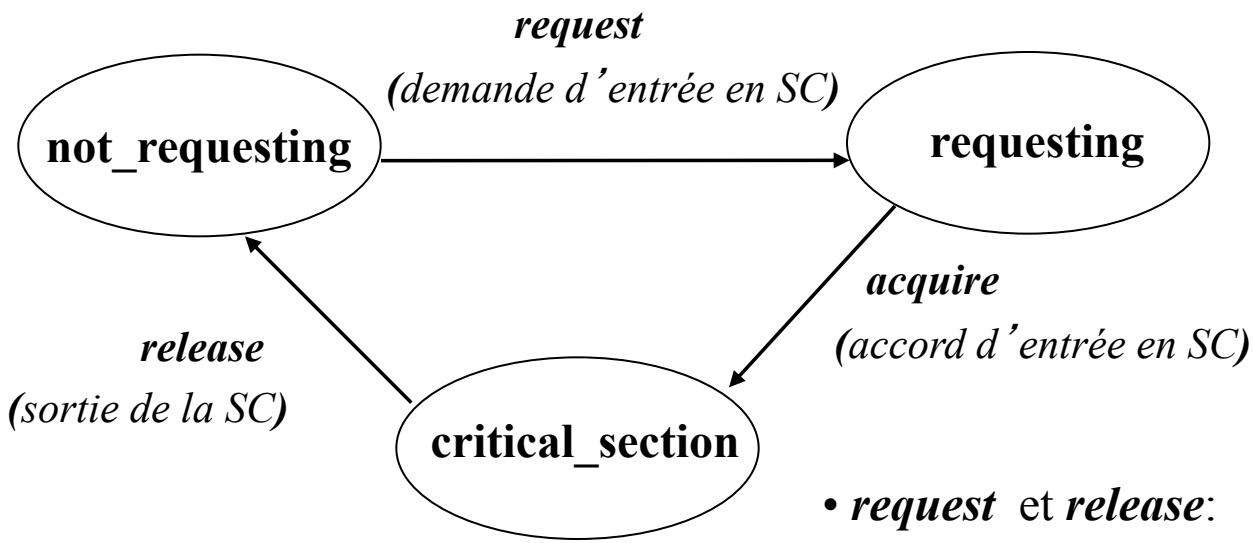
- **Exclusion mutuelle en réparti**
  - Types d'algorithmes
  - Propriétés
  - Classes d'algorithmes
    - Algorithmes à permission
      - Lamport
      - Ricart-Agrawala
      - Maekawa (quorum)
    - Algorithmes à jeton
      - Martin
      - Raymonde
      - Naimi-Trehel
      - Susuki-Kasami

Exclusion Mutuelle

- **Objectif:**
  - Coordonner des processus se partageant une ressource commune pour qu'à tout instant, au plus un processus ait accès à cette ressource.
  - L'accès se fait dans une section critique (SC), dont les processus demandent l'entrée et signalent la sortie.
- **Exclusion Mutuelle en réparti:**
  - Les processus sont répartis et ne communiquent que par passage de messages.

Transitions d'un processus

- **N processus: P_1,…, P_N**
  - état$_i$: {*requesting*, *not_requesting*, *critical_section*}



• ***request*** et ***release***:
fonctions invoquées par les processus
• ***acquire***: transition contrôlée par l'algorithme.

Algorithme d'Exclusion Mutuelle

- **Un algorithme d'exclusion mutuelle doit garantir :**
  - Au plus un processus exécute la section critique à un instant donné.
  - Pas d'**interblocage**
    - Si des processus demandent concurremment à entrer en section critique, la sélection ne peut pas être ajournée indéfiniment.
  - Pas de **famine**
    - La demande d'un processus ne peut pas être différée indéfiniment. Autrement dit, un processus qui demande à entrer en section critique doit être autorisé à le faire dans un temps fini.

Propriétés à assurer

Les propriétés d'un algorithme se classent en deux catégories:

- **sûreté (safety) :**
  - jamais rien de "mauvais" n'arrive
- **vivacité (liveness) :**
  - quelque chose de "bien" finit par arriver

- **Algorithme d'exclusion mutuelle**
  - **Sûreté** :
    - à tout instant il y a au plus un processus dans la section critique.
      - *Si ($etat_i$ = critical_section) alors ($etat_j$!= critical_section) pour tout j<> i;*
  - **Vivacité :**
    - Tout processus qui demande la section critique doit l'obtenir au bout d'un temps fini.
      - *($etat_i$ = requesting) -> ($etat_i$ = critical_section)*
      - Garantir l'absence d'interblocage et de famine.
  - Propriétés auxquelles on ajoute l'**équité.**

Exclusion mutuelle en réparti

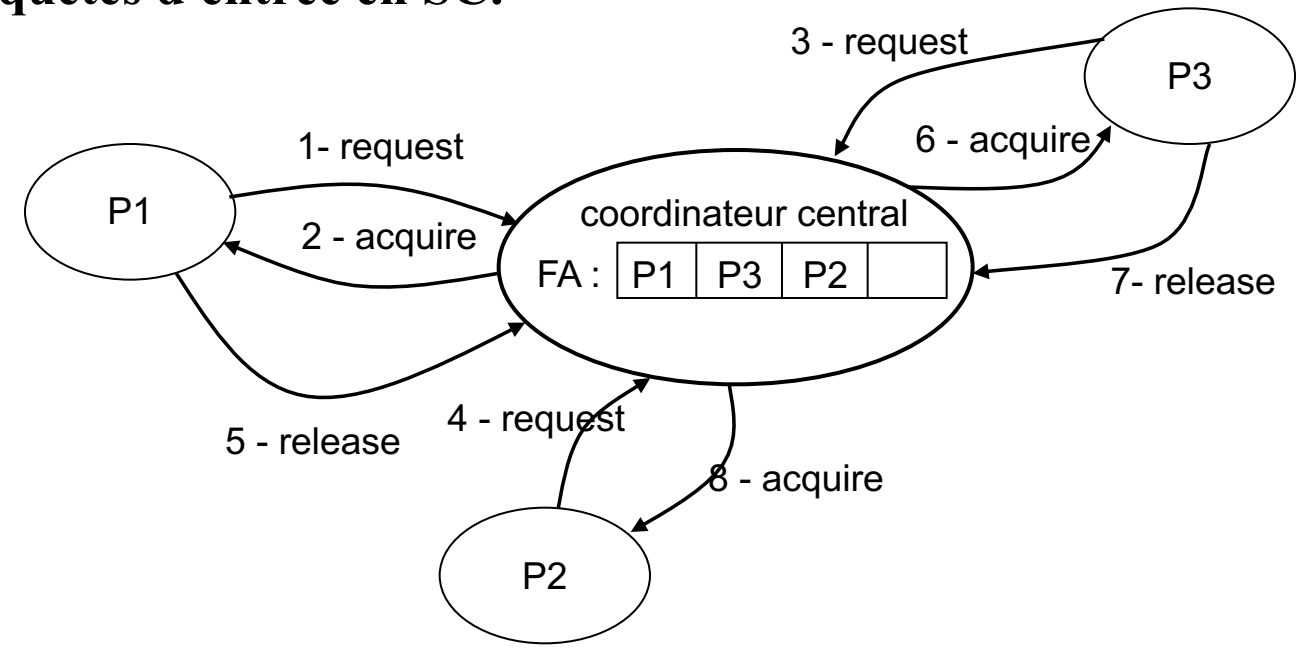
- **Le contexte réparti :**
  - *N* sites (nœuds ou processus).
  - Pas de mémoire globale partagée ni d'horloge physique globale.
  - Les sites communiquent par passage de messages, qui sont envoyés et reçus sur des canaux.
    - Aucune hypothèse temporelle n'est faite pour le délai de transmission des messages.
  - Les canaux:
    - Fiables.
    - "FIFO" ou non "FIFO", selon les hypothèses de l'algorithme.

Algorithme centralisé x réparti

- **Algorithme centralisé - coordinateur**
  - Le processus coordinateur est le seul à prendre une décision sur les accès à la section critique.
  - Tous les informations nécessaires pour l'algorithme sont concentrées dans le coordinateur.
- **Solutions entièrement réparties** :
  - Tous les processus peuvent participer à la décision sur l'accès à la section critique.
  - Les informations pour réaliser l'algorithme sont réparties entre les processus.

Algorithme centralisé

- **Tous les processus s'adressent à un coordinateur pour demander l'entrée et signaler la sortie de SC. Le coordinateur maintient une file d'attente (FA) dans laquelle il range par ordre d'arrivée les requêtes d'entrée en SC.**



Algorithme centralisé (Evaluation)

- **Nombre de Messages par exécution de SC :**
  - 3 messages.
- **Equitable** :
  - requêtes traitées par ordre d'arrivée sur le coordinateur.
- **Avantages** :
  - simplicité (en fonctionnement normal).
  - faible complexité en messages.
- **Inconvénients** :
  - goulot d'étranglement sur le coordinateur.
  - panne du coordinateur relativement complexe à résoudre.

Algorithmes répartis

- **Classes d'algorithmes**
  - **A base de permission** :
    - Afin d'entrer en section critique, un processus P_i doit demander la permission à d'autre processus.
    - Le droit d'entrée en SC est acquis lorsque le processus a obtenu un nombre suffisant de permissions.
  - **A base de jeton** :
    - Seul le processus possédant le jeton peut entrer en section critique.
    - L'unicité du jeton garantit la propriété de sûreté
    - Différentes façons de réaliser la vivacité:
      - informer le site qui possède le jeton des requêtes en cours.
      - assurer le routage du jeton vers les processus demandeurs.

Algorithmes à base de Permission

- **Lamport**
- **Ricart-Agrawala**
- **Maekawa (quorum)**

Algorithme à Base de Permission

- **Lamport (1978) et Ricart/Agrawala (1981)**
  - Un message de demande d'entrée en SC est envoyé à tous les autres sites R_i
    - $R_i = \{1,2,3,\ldots N\} - \{i\}$
  - Ordre total des requêtes d'entrée en section critique :
    - Les requêtes sont totalement ordonnées et satisfaites selon cet ordre.
    - La date d'une requête est la valeur de l'**horloge logique scalaire** du processus émetteur P_i, complétée par son identifiant : (H_i, i).
      - $(H_i, i) < (H_j, j) \Leftrightarrow (H_i < H_j \text{ ou } (H_i = H_j \text{ et } i < j))$
    - Garantie de la **sûrete** et de la **vivacité.**
  - Chaque processus P_i gère une horloge logique, une file d'attente FA_i de requêtes **classées par date** et les attentes de permission At_i.

Algorithme de Lamport

- **Hypothèses :**
  - Le nombre **N** de processus est connu de tous.
  - Les **canaux** de communication sont fiables et **FIFO.**

- **Messages :**
  - **Types :**
    - *REQUEST* **:** demande d'entrer en SC.
    - *REPLY* : réponse à la réception d'un message *REQUEST*.
    - *RELEASE***:** libération de la SC.
  - **Contenu:**
    - (type, (H_i, S_i));

- **Variables Locales du processus P_i :**
  - H_i : Horloge logique scalaire
  - FA_i *:* File d'attente de requêtes
    - Dans l'ordre induit par la valeur de leurs estampilles (y compris celle de P_i*)*
  - At_i : Attente de permission.

Algorithme de Lamport

Tous les processus S_i :

Variables Locales:

FA_i = ø;
H_i = 0;
At_i = ø;

Request_CS(S_i) :

- H_i = ++;
- Placer sa requête req_i dans la file d'attente;
- Envoyer un message *REQUEST* à tous les autres sites ($At_i = R_i$-{Si});
- Attendre l'accord de tous les autres sites (msg *REPLY*) et que sa propre requête soit la plus ancienne de toutes (At_i = ø; et req_i = head (FA_i));

Release_SC(S_i) :

- H_i ++;
- Diffuser un message *RELEASE* à tous les autres sites (***Ri-{Si}***);

Enlever sa requête req_i de la file d'attente FA_i;.

Reception (msg de S_j) :

- Mettre à jour H_i: (H_i = max (H_i,H_j) +1;)
- Switch (type msg) :{

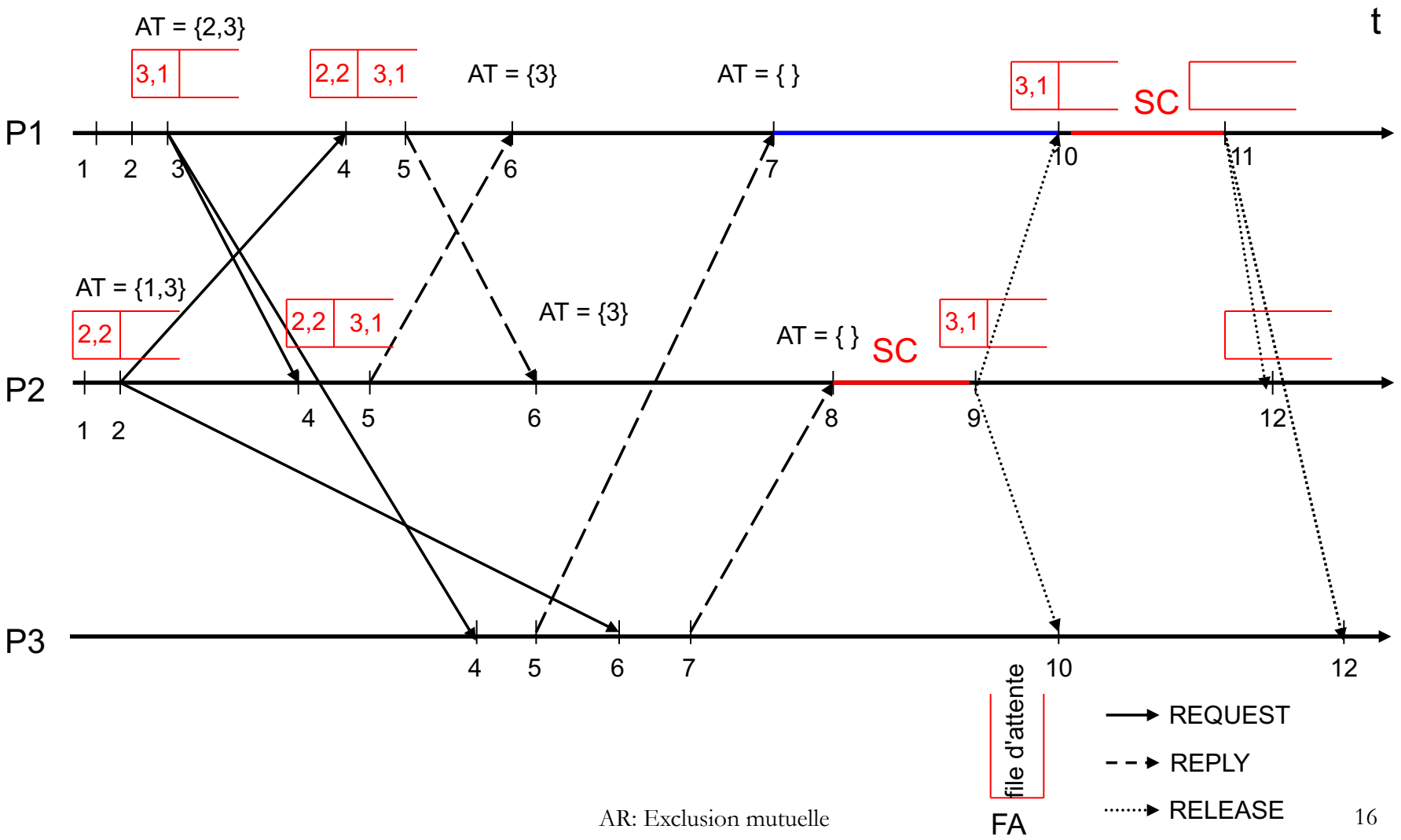
REQUEST : - placer la requête reçue dans la file d'attente FA_i dans l'ordre des estampilles : (FA_i U {msg S_j});
- envoyer un message *REPLY* à S_j.

REPLY: - traiter la réception de l'acquittement (At_i –{S_j})

RELEASE: - Enlever la requête de S_j de la file d'attente (FA_i –{ msg S_j}) ;

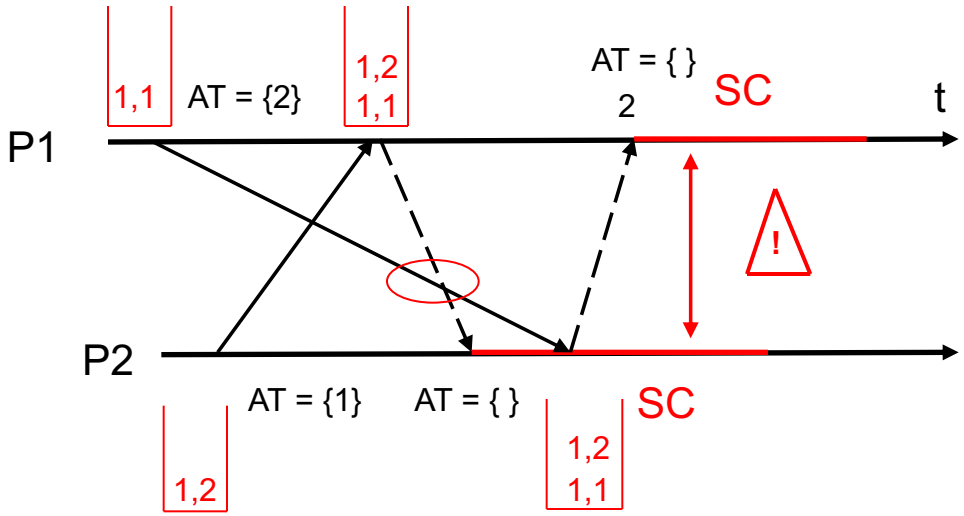
}

Algorithme de Lamport (exemple)



Algorithme de Lamport

- Si les canaux ne sont pas FIFO, l'exclusion mutuelle n'est pas garantie.



  - Propriété **FIFO** de canaux garantie:
    - Si un site S_i a reçu un message d'accord (REPLY) de S_j, toute requête antérieure de S_j lui est forcément arrivée. Toute demande lui arrivant de S_j sera postérieure à la sienne.

Algorithme de Lamport

- **L'ordre total sur les demandes garantit :**
  - **La sûreté**:
    - Seul le site en tête de la file d'attente *FA* pourra rentrer en SC; les autres attendent que cette demande soit retirée (réception du message RELEASE).
  - **La vivacité:**
    - Toute demande finira par avoir la plus petite estampille et donc se trouvera en tête de la file d'attente.

Algorithme de Lamport (Evaluation)

- **Nombre de Messages par exécution de SC:**
  - 3* (N-1) messages.
- **Equitable :**
  - requêtes traitées par l'ordre total.
- **Avantages :**
  - simplicité (en fonctionnement normal).
- **Inconvénients :**
  - Hypothèse de canaux FIFO.
  - Pas extensible.

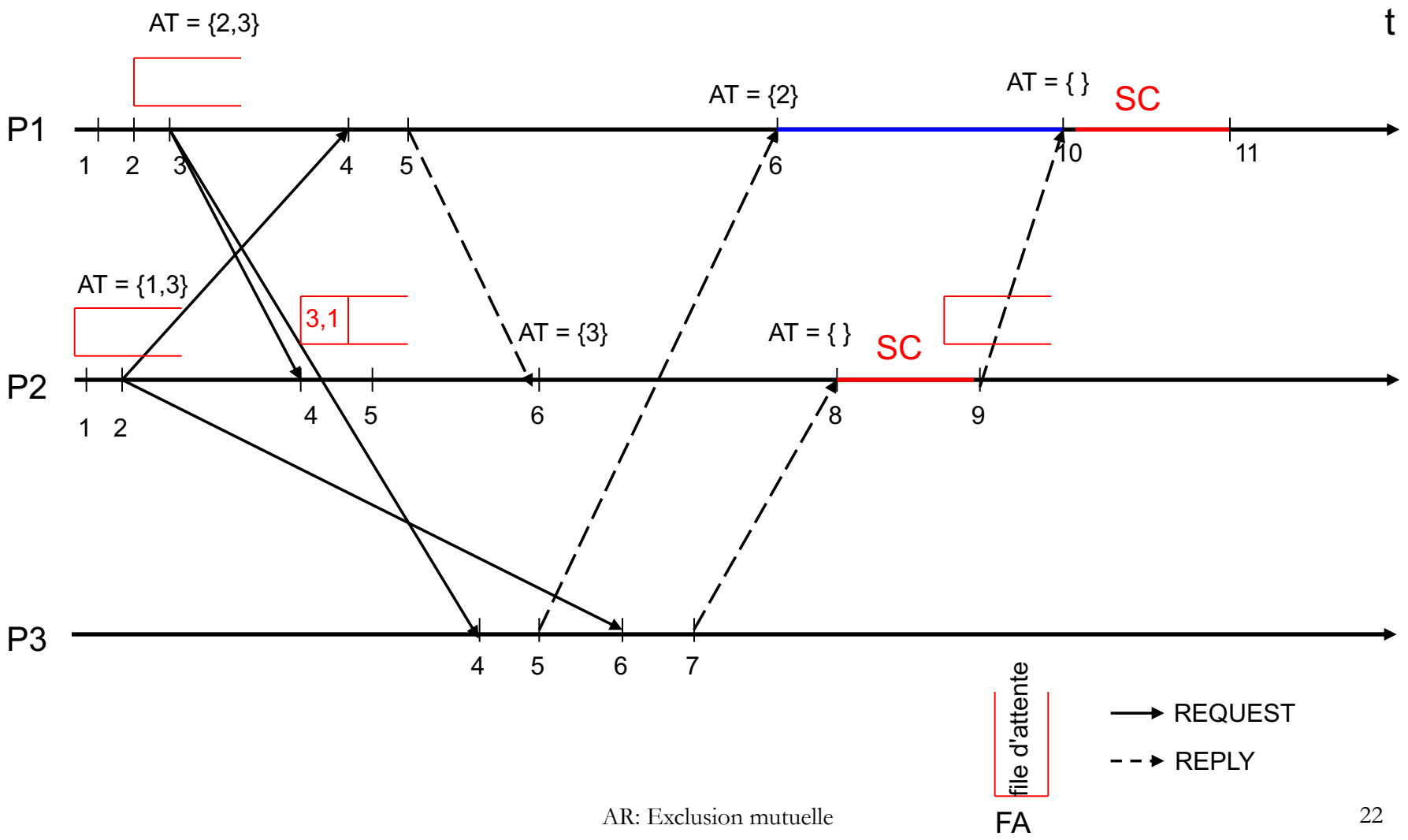
Algorithme Ricart/Agrawala

- **Amélioration de l'algorithme de Lamport:.**
  - **Message REPLY :** possède le sens d'une autorisation d'accès, délivrée de façon conditionnelle. Un processus P_i n'acquitte une requête que s'il n'est pas en SC et sa requête en cours n'est pas plus prioritaire.
  - **Message RELEASE** : n'est envoyé qu'aux processus dont la requête a été différée. Remplacé par le message **REPLY**.
  - **File d'attente** : chaque processus P_i ne conserve dans sa file d'attente FA_i *q*ue les requêtes dont l'acquittement a été différé.

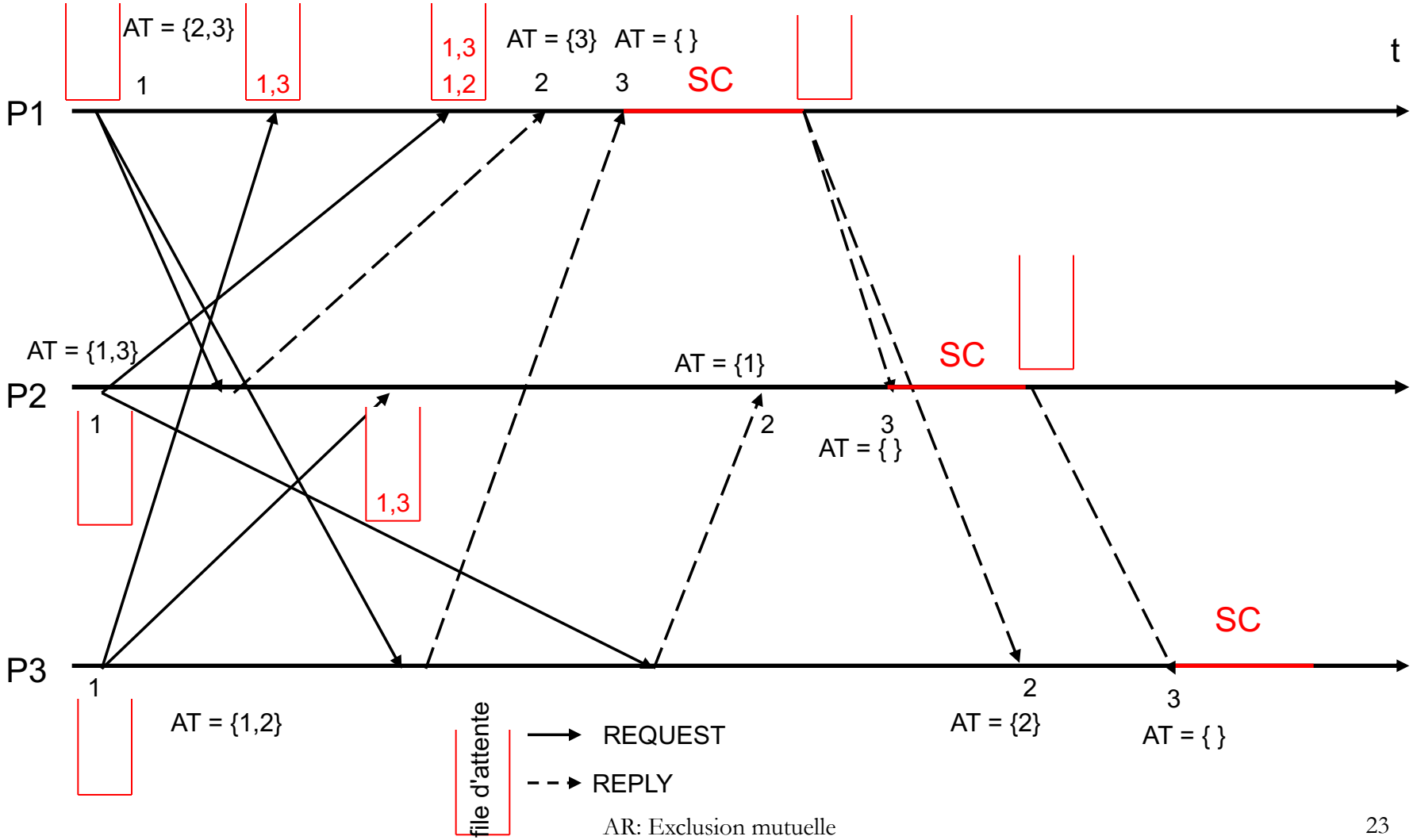
Algorithme de Ricart/Agrawala

- **Hypothèses :**
  - Le nombre **N** de processus est connu de tous.
  - Les canaux de communication sont fiables, mais pas **FIFO.**
- **Messages :**
  - Types :
    - *REQUEST* **:** demande d'entrer en SC.
    - *REPLY* : réponse à la réception d'un message *REQUEST***.**
  - Contenu :
    - (type, (H_i, S_i));
- **Variables Locales du processus P_i :**
  - H_i : Horloge logique scalaire
  - FA_i *:* File d'attente
  - At_i Attente de permission.
  - $Etat_i$: *requesting*, *not_requesting*, *critical_section.*
- **Algorithme**
  - Sera vu en TD et TME.

Algorithme de Ricart/Agrawala (exemple 1)

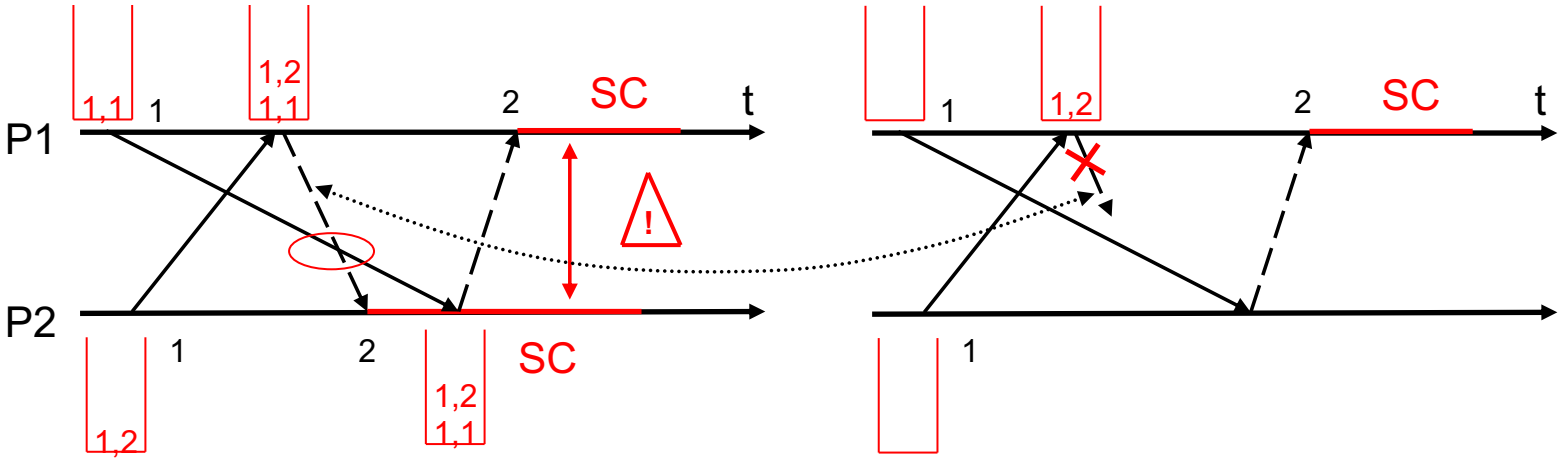


Algorithme de Ricart/Agrawala (exemple 2)



Algorithme de Ricart/Agrawala

- **Hypothèse FIFO n'est plus nécessaire**
  - les messages *REPLY* valent autorisation d'entrée en SC. Quand un processus a reçu tous les msg. *REPLY*,il n'y a plus de requête plus récente en cours.



Lamport 78

Ricart et Agrawala 81

Algorithme de Ricart/Agrawala (Evaluation)

- **Nombre de Messages par exécution de SC :**
  - 2*(N-1) messages.
- **Equitable :**
  - requêtes traitées par l'ordre total.
- **Avantages par rapport à Lamport:**
  - moins de messages envoyés.
  - taille de la file d'attente FA plus petite.
  - hypothèse FIFO non nécessaire.