Table des matières

1. Horloges logiques

1.1. Problèmatique

Lors-ce qu'on travaille dans un système distribué, il est crucial de pouvoir ordonner les événements qui s'y produisent. Cependant, en l'absence d'une horloge globale, il devient difficile de déterminer l'ordre exact des événements.

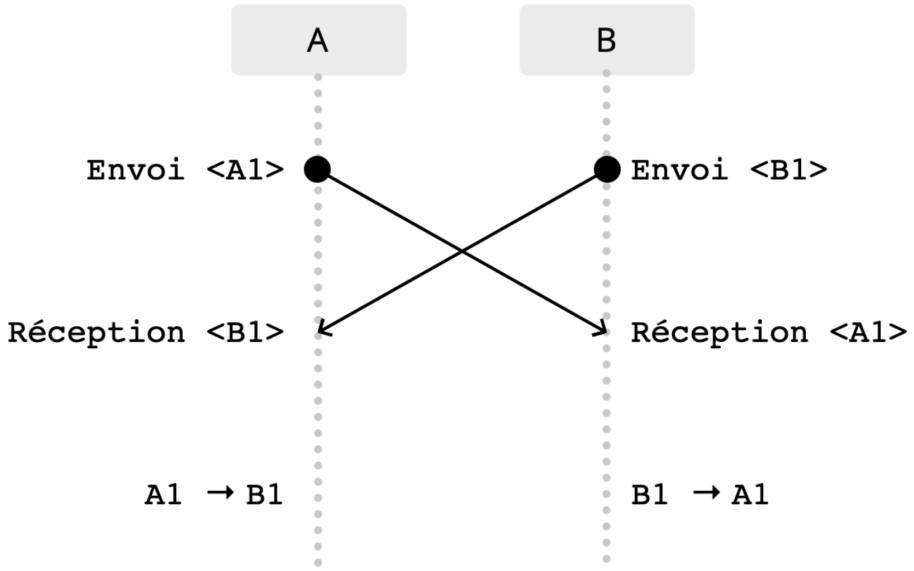

Figure 1: Capture des slides du cours - Horloges logiques

> ⚠ – **Warning**
> Sur cette image, nous voyons la problèmatique concernant l'ordonnancement des événements dans un système distribué. Chaque système peçois que son événement est le premier.

1.2. Protocole existant

- **NTP**: Network Time Protocol (NTP) `ms` *precision*
- **PTP**: Precision Time Protocol (PTP) `µs` *precision*

1.3. Résolution

On ne cherche pas l'heure excate mais plutot un ordre d'évenement.

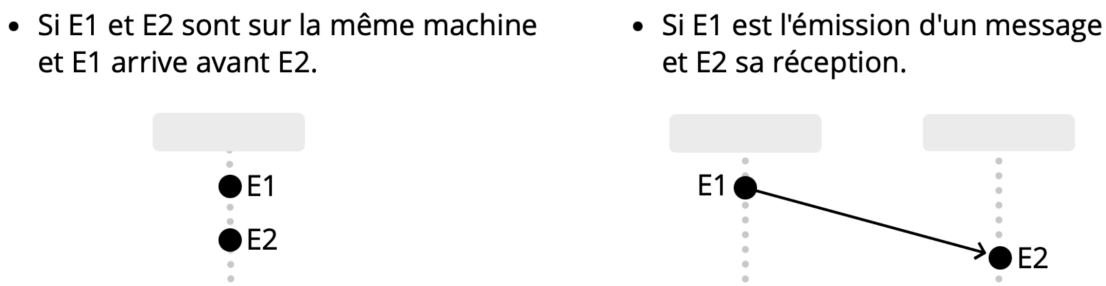

Figure 2: Capture des slides du cours - Ordre d'événements

1.4. Propiétés des ordres

Nous pouvons définir trois propriétés pour un ordre d'événements:

- **Transitivité**: Si un événement A précède un événement B, et que B précède C, alors A précède C.
- **Anti-réflexif**: Aucun événement ne peut précéder lui-même.

- **Antisymétrique**: Si un événement A précède B, alors B ne peut pas précéder A.

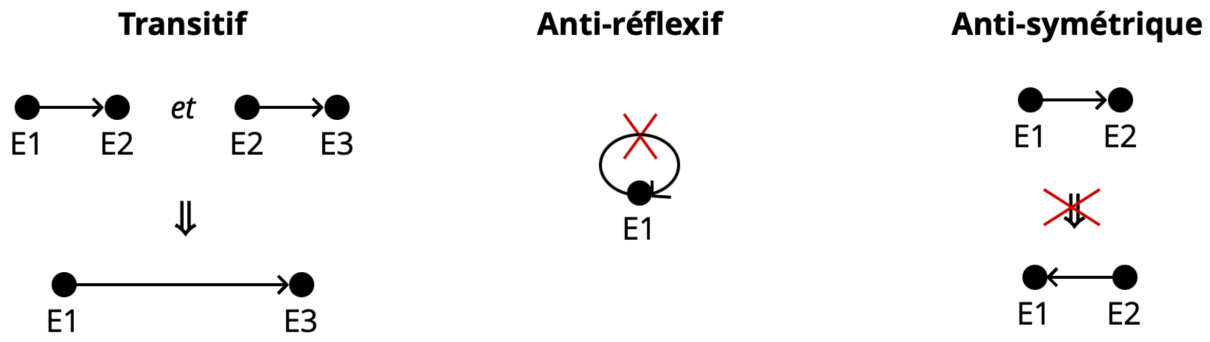

Figure 3: Capture des slides du cours - Propriétés des ordres

1.5. Horloge logique

On cherche à définir une fonction `H` qui associe un entier à chaque événement, de manière à respecter les propriétés d'ordre définies précédemment.

Une idée serait d'utiliser une horloge locale pour chaque processus, qui s'incrémente à chaque événement. Cependant, cela ne garantit pas que les événements soient ordonnés correctement à travers les différents processus. Le risque de décalage entre les horloges locales est trop grand et nous pourrons nous retrouver avec un événement 6 qui arrive en réalité avant un événement 5.

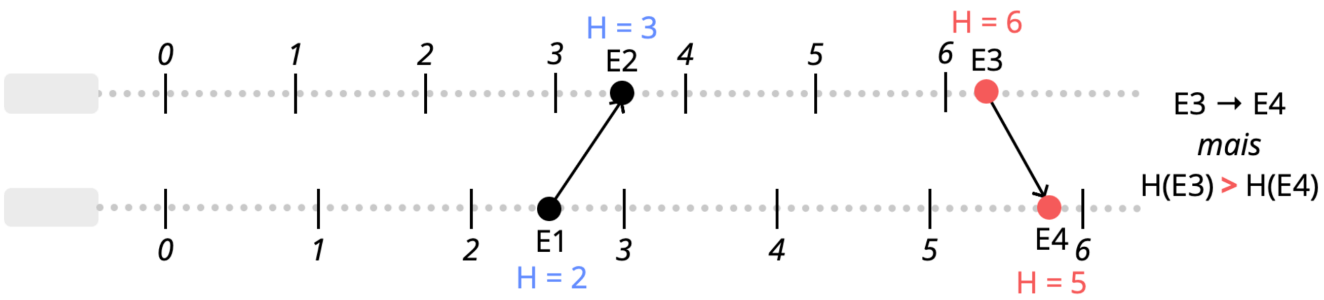

Figure 4: Capture des slides du cours - Risque d'utilisation d'horloges locales

1.6. Horloge de Lamport

Pour résoudre ce problème, Lamport propose un algorithme d'horloge logique qui utilise des horloges locales, mais ajoute des règles pour garantir l'ordre des événements à travers les processus.

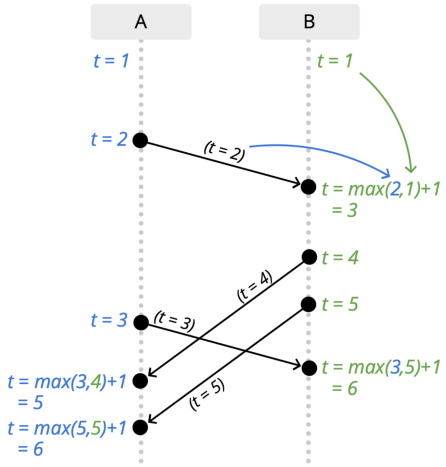

Figure 5: Capture des slides du cours - Algorithme de Lamport

- Chaque site maintient un numéro
- À chaque événement local, le site incrémente son numéro
- Le timestamp est attaché au message
- À la réception d'un message, le site met à jour son numéro avec le maximum entre son numéro et le timestamp reçu, puis incrémente ce numéro de 1

> ⚠ – **Warning**
> Cet algorithme n'est pas déterministe, si deux événements sont concurrents, l'ordre peut varier selon les exécutions. **Solution**: priorité à la machine avec l'ID le plus petit

⚠ – **Warning**

- Les timestamps de Lamport n'ordonnancent pas les messages immédiatement.
- Ils assurent seulement qu'un ordre strict sera obtenu un jour.

2. Exclusion mutuelle

2.1. Problèmatique

Dans un système distribué, il est crucial de gérer l'accès concurrent aux ressources partagées. L'exclusion mutuelle garantit qu'une ressource ne peut être utilisée que par un seul processus à la fois, évitant ainsi les conflits et les incohérences.

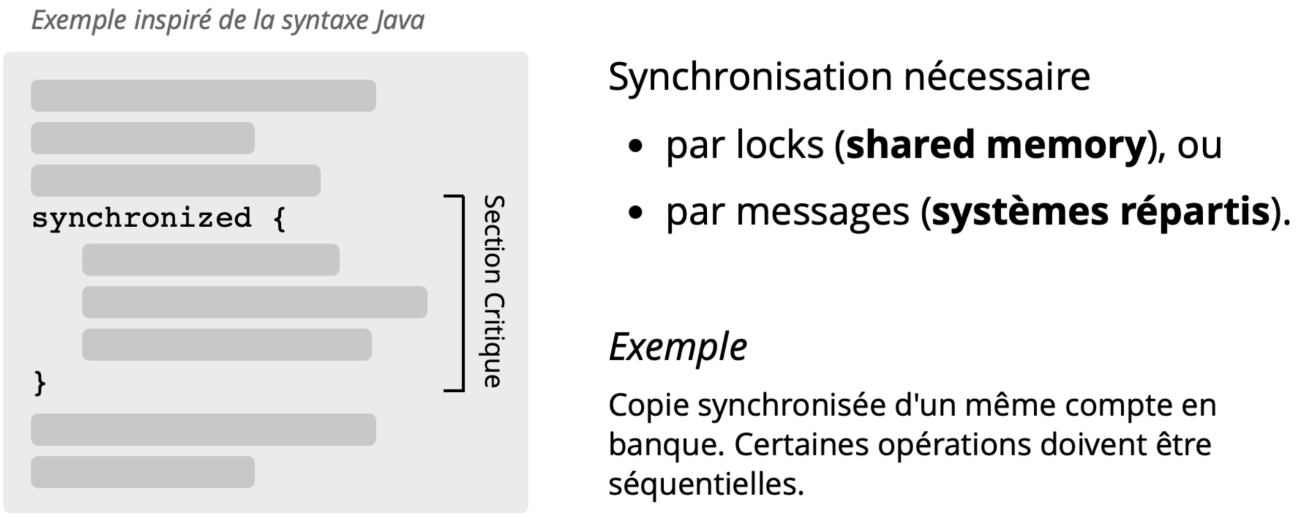

Figure 6: Capture des slides du cours - Exclusion mutuelle

2.2. Systpème centralisé

Un serveur centralisé gère les demandes d'accès aux ressources. Chaque processus doit demander la permission au serveur avant d'accéder à la ressource.

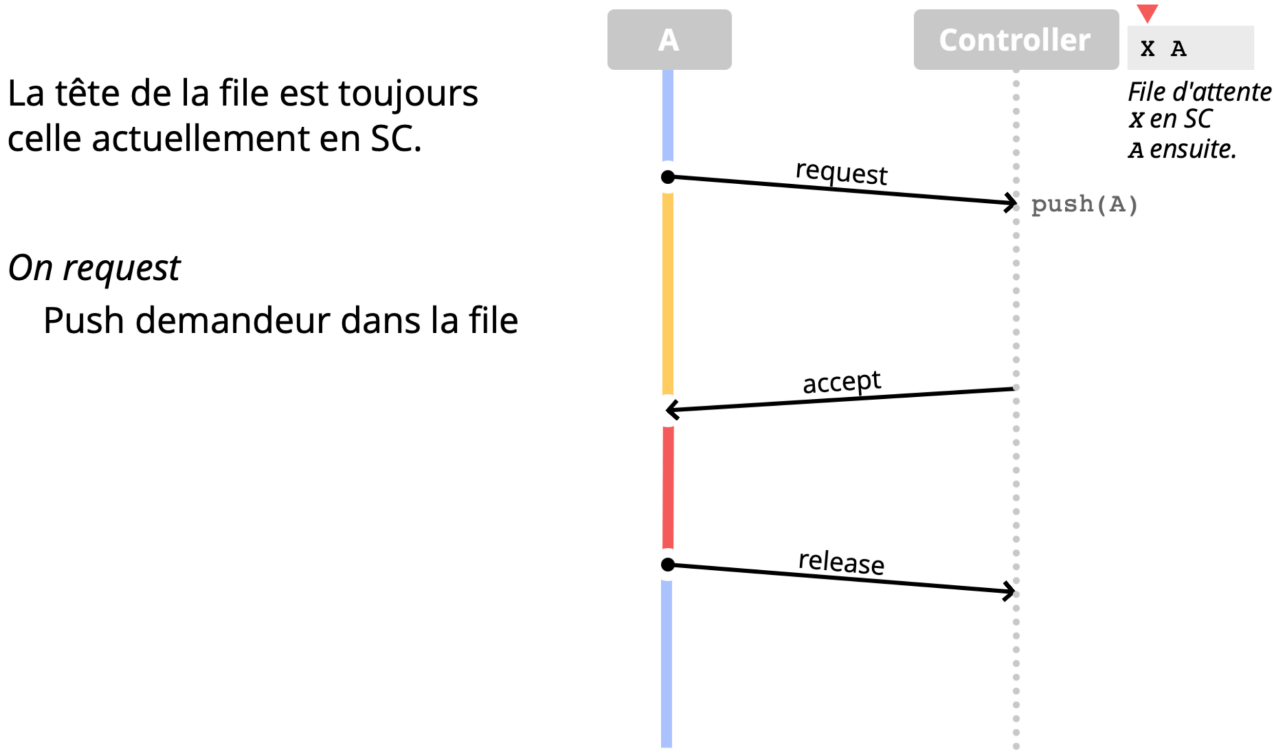

Figure 7: Capture des slides du cours - Système centralisé

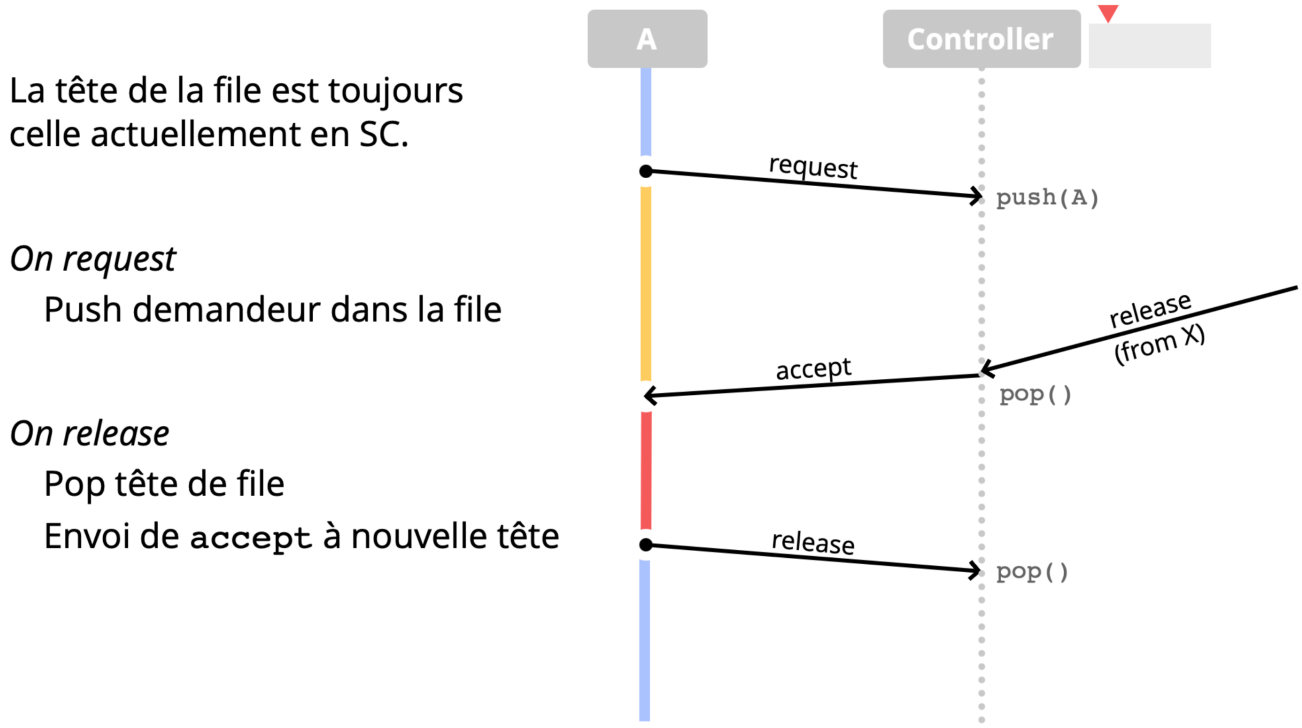

Figure 8: Capture des slides du cours - Système centralisé 2

2.3. Solution répartie

Chaque processus maintient une file d'attente des demandes d'accès aux ressources, ordonnée par les horloges logiques. Lorsqu'un processus souhaite accéder à une ressource, il envoie une demande à tous les autres processus et attend les réponses.

2.3.1. Version priority queue

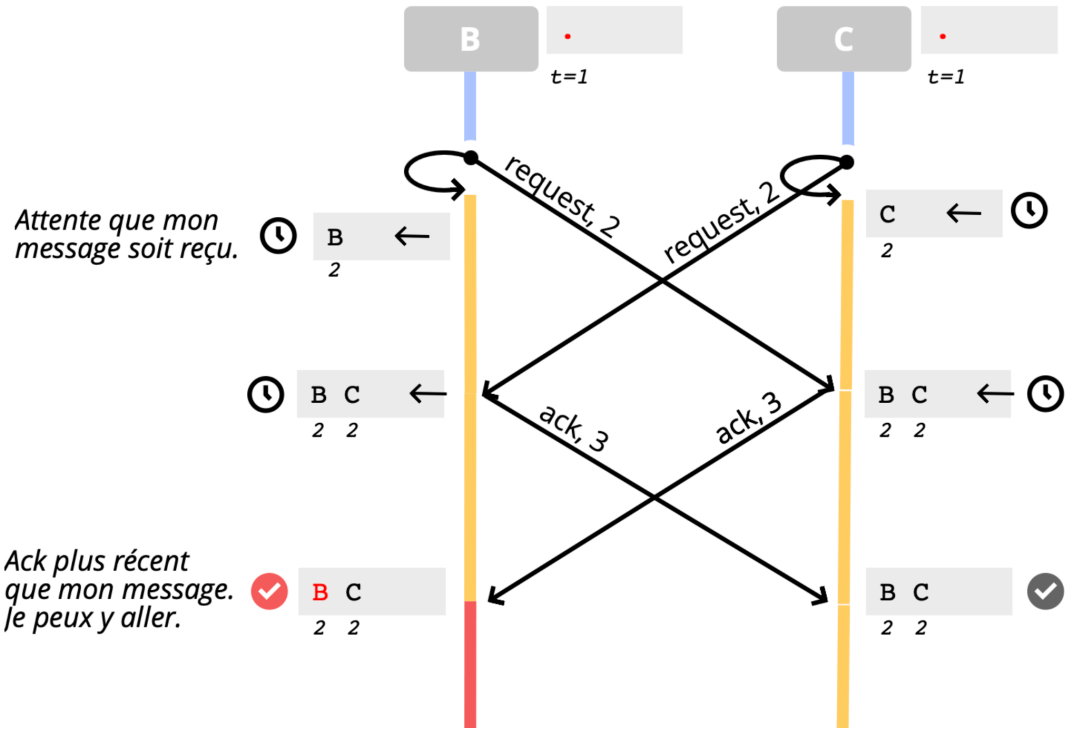

Figure 9: Capture des slides du cours - Solution répartie

2.3.1.1. On request

- Push demandeur dans la file d'attente
- Réodonner la file d'attente par heure de Lamport
- Envoie un `ack` avec nouveau timestamp

2.3.1.2. On realese

- Pop tête de file
- Entrer en section critique
  - Si je suis la nouvelle tête et que j'ai reçus tous les `acks`

2.3.2. Version tableau

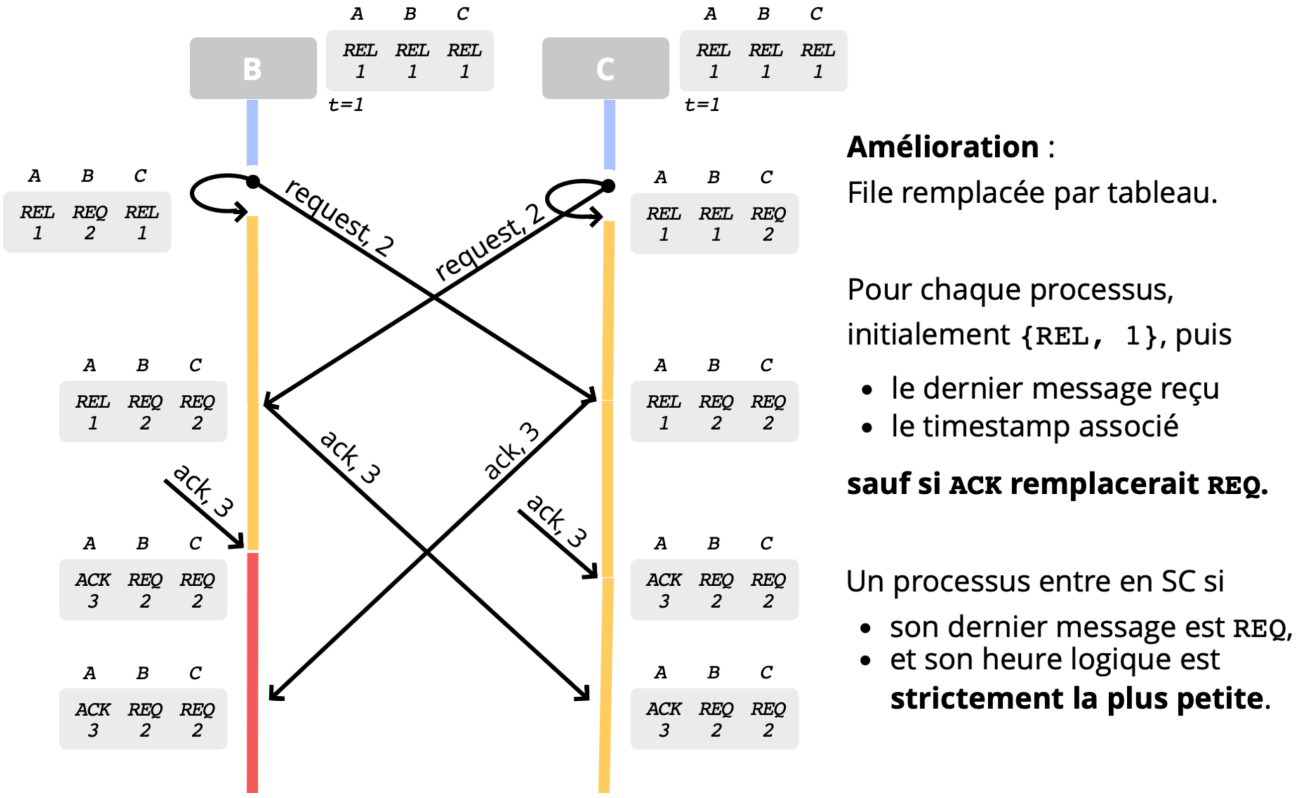

Figure 10: Capture des slides du cours - Solution répartie version tableau

Chaque processus maintient un tableau des derniers messages reçus ainsi qu'un timestamp pour chaque message, par exemple `(REQ, id)`.

2.4. Propriétés de l'algorithme Lamport

- **Corectness**: Un seul processus peut être en section critique à la fois.
- **Progrès**: Toute demande d'entrée en SC sera autorisée un jour.
- **Complexité**
  - Communication par SC par processus : $3n(n-1)$
  - Calcul par événement :
    - $O(n)$ par réception de message
    - $O(1)$ pour le reste.