

SDR - Systèmes Distribués et Repartis

Mutex par jetons

13 octobre 2025

Table des matières

1 Ricart & Agrawala	1
1.1 Amélioration	2
1.2 Pseudo-code	3
1.3 Optimisation d'accès	4
1.4 Propriétés	4
1.5 Cas particulier	4
2 Carvalho & Roucairol	4
2.1 Amélioration	5
2.2 Propriétés	6
2.3 Pseudo-code	6

1 Ricart & Agrawala

On passe d'une approche à 3 messages:

- `REQ` : demande d'accès à la section critique
- `ACK` : autorisation d'accès à la section critique
- `REL` : libération de la section critique

i Info

Ce que nous pouvons tirer comme conclusion est que nous sommes assez indirect dans notre approche. Ce que nous voulons est d'entrer en **section critique**.

Jusqu'à présent, nous avions 3 messages, hors, ce que l'on souhaite c'est demander d'entrer en section critique, une fois que l'on a **toutes** les autorisations, on entre en section critique, puis on libère la section critique. On en comprend donc que le message `REQ` est nécessaire, cependant on peut simplifier `ACK` et `REL` en un seul message `OK` qui autorise l'entrée en section critique.

💡 Hint

En cherchant à limiter le nombre de messages, on peut penser à une approche par jetons.

1.1 Amélioration

App veut entrer en SC

J'envoie un `REQ` à tout le monde

Réception d'un `REQ`

Si je ne suis pas en SC

- Si je ne veux pas entrer en SC
 - Je réponds avec `OK`
- Si je veux entrer en SC
 - Si je n'ai pas la priorité, je réponds avec `OK`.
 - Si j'ai la priorité, j'attends d'avoir fini, puis je réponds avec `OK`.

Si je suis en SC

- J'attends d'avoir fini, puis je réponds avec `OK`.

App sort de SC

Je réponds par `OK` à tous les `REQ` en attente.

Réception d'un `OK`

Je le comptabilise.

- Si j'ai le `OK` de tout le monde, alors je peux entrer en SC.

Fig. 1. – Capture des slides du cours – Version améliorée

Grâce à cette nouvelle approche, nous pouvons désormais $2(n - 1)$ messages par processus pour entrer en section critique.

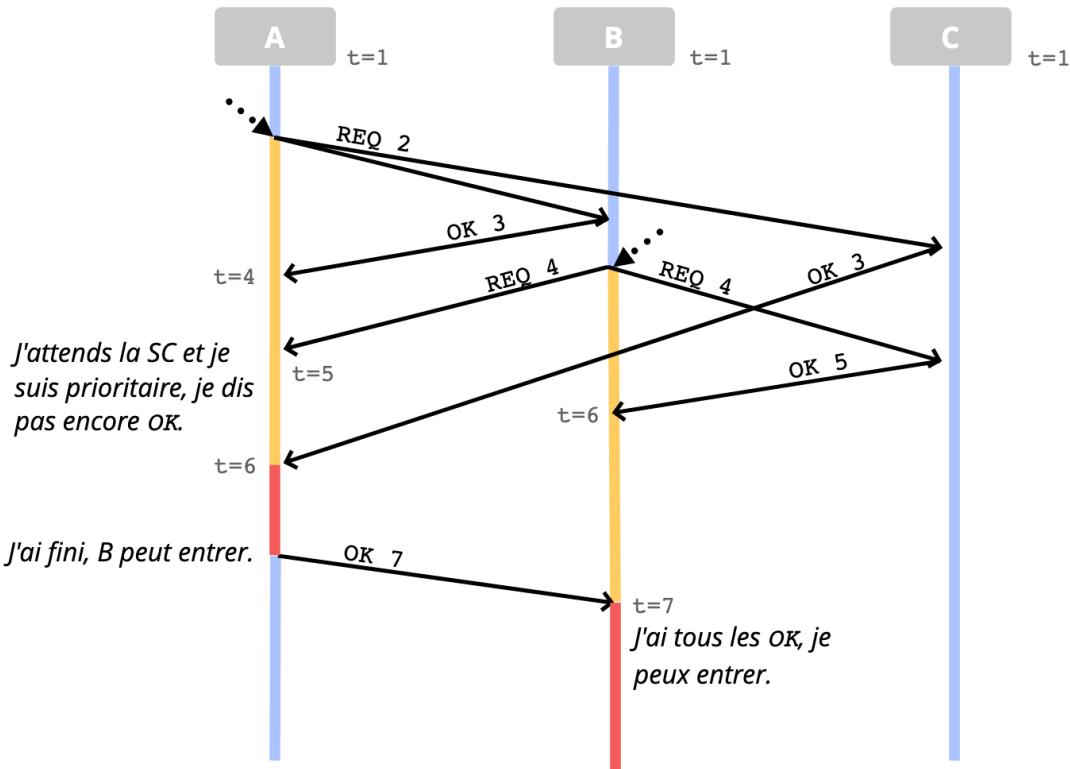


Fig. 2. – Capture des slides du cours – Echanges de messages

1.2 Pseudo-code

<code>n</code>	<i>entier, constant</i>	nombre de processus
<code>self</code>	<i>entier, entre 0 et n-1</i>	mon numéro de processus
<code>ts</code>	<i>entier, init 0</i>	mon timestamp Lamport actuel
<code>hasRequested</code>	<i>booléen, init false</i>	ssi j'ai fait une demande de SC
<code>selfRequestTs</code>	<i>entier</i>	timestamp de cette demande
<code>missingOKs</code>	<i>entier</i>	nombre de OKs que j'attends encore
<code>waitingPs</code>	<i>ensemble d'entiers</i>	numéros des processus qui attendent mon OK.

Fig. 3. – Capture des slides du cours – Variables du pseudo-code

Traitement : Demande de SC de la couche applicative.

```

ts += 1
hasRequested ← true
selfRequestTs ← ts
missingOKs ← n-1
Envoi de {REQ, ts} à tous les autres processus.

```

Traitement : Passage de missingOKs à 0.

Autorisation de la couche application d'entrer en SC.

Traitement : Sortie de SC par la couche applicative.

```

ts += 1
hasRequested ← false
Envoi de {OK, ts} à tous les processus de waitingPs
Réinitialisation de waitingPs

```

Écouter infiniment les événements suivants:

- Demande de SC de la couche applicative
- Sortie de SC dans la couche applicative
- Passage de missingOKs à 0
- Réception de {REQ, tsi} du processus i
- Réception de {OK, tsi} du processus i

Fig. 4. – Capture des slides du cours – Initialisation du pseudo-code

Fig. 5. – Capture des slides du cours – Fonctionnement du pseudo-code

Traitement : Réception {REQ, tsi} du processus i.

```

ts ← max(tsi, ts) + 1
Si hasRequested et
  (selfRequestTs < tsi ou
   (selfRequestTs == tsi et self < i))
    Ajout de i dans waitingPs
Sinon
  Envoi de {OK, ts} à i

```

Traitement : Réception {OK, tsi} du processus i.

```

ts ← max(tsi, ts) + 1
waitingOKs -= 1

```

Fig. 6. – Capture des slides du cours – Fonctionnement du pseudo-code

1.3 Optimisation d'accès

Un autre point est que dans cette approche, si on récupère l'accès pour entrer en section critique, je peux en déduire que j'y ai accès tant que personne ne redemande l'accès.

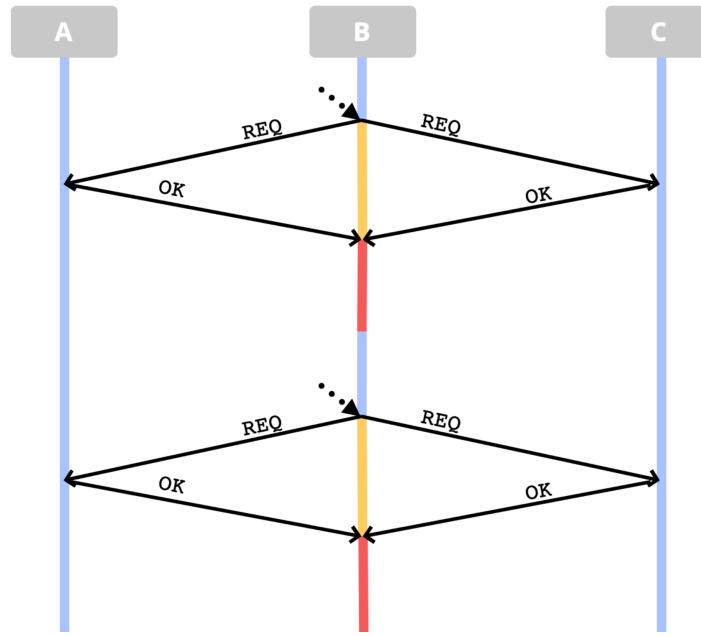


Fig. 7. – Capture des slides du cours – Optimisation d'accès

On peut voir cela un peu comme un jeton que l'on possède. Tant que l'on possède le jeton, on peut entrer en section critique.

1.4 Propriétés

- **Corectness:** Jamais plus d'un processus dans la section critique.
- **Progress:** Toute demande d'entrée en section critique finit par être satisfaite.
- **Complexité de message:** $2(n - 1)$ messages par entrée en section critique.

1.5 Cas particulier

Dans le cas où un processus vient de sortir de SC et qu'il souhaite y rentrer à nouveau, il doit à nouveau envoyer des demandes à tous les autres processus, alors qu'en optimisant il pourrait directement s'y rendre à nouveau.

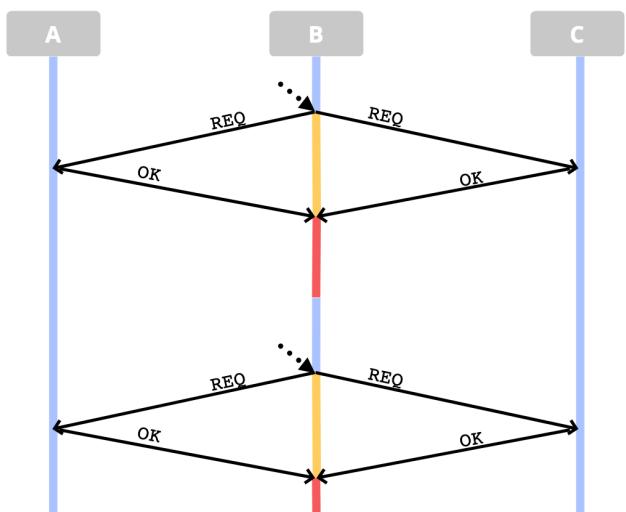


Fig. 8. – Capture des slides du cours – Cas particulier

2 Carvalho & Roucairol

2.1 Amélioration

On pourrait penser à modifier l'algorithme de Ricart & Agrawala en se disant: **Tant qu'on m'a pas redemandé l'accès, je peux y retourner directement sans redemander l'accès.**

Dès ce moment, on peut imaginer que chaque processus possède un jeton. Lorsqu'un processus souhaite entrer en section critique, il doit posséder le jeton. S'il ne le possède pas, il doit le demander au processus qui le possède.

On peut visualiser cet algorithme comme un graph où chaque processus est un noeud, et entre chaque noeuds, il y a une arête qui représente le jeton.

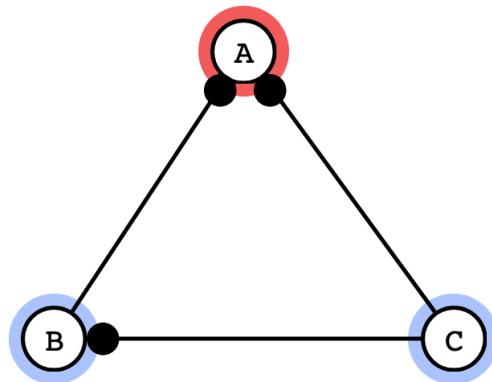
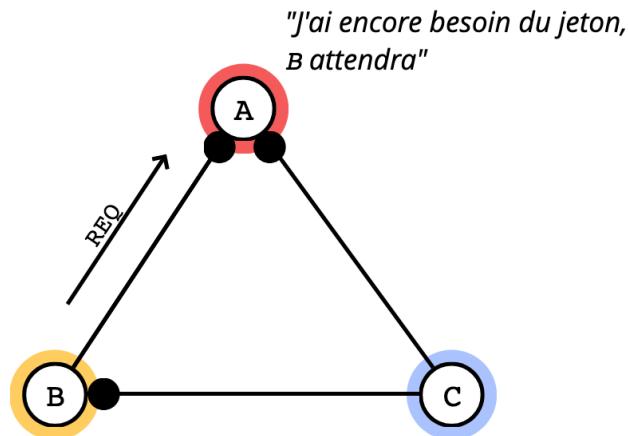


Fig. 9. – Capture des slides du cours – Echanges de messages

Lorsque qu'un processus souhaite entrer en section critique, il envoie une demande aux noeuds voisins dont il ne possède pas déjà le jeton.



*"Je veux entrer en SC mais
il me manque un jeton"*

Fig. 10. – Capture des slides du cours – Echanges de messages

Une fois que le processus A a terminé sa section critique, il envoie le jeton au processus B qui lui a fait la demande. Une fois que B a terminé, si personne n'a demandé les jetons, il les garde.

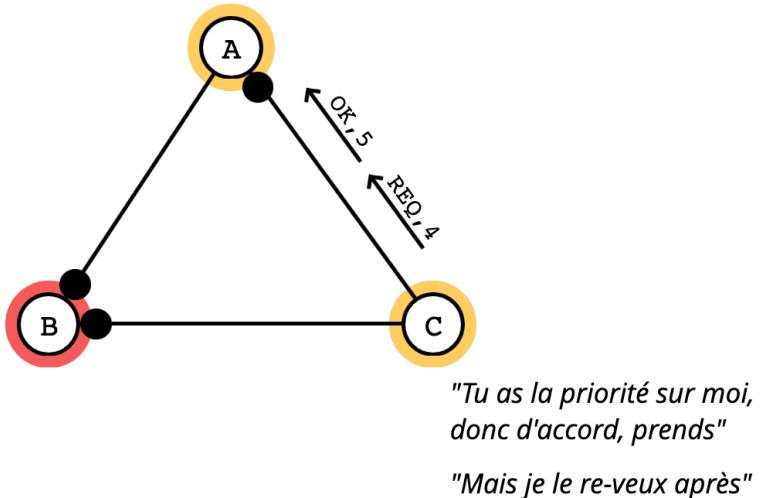


Fig. 11. – Capture des slides du cours – Echanges de messages

Dans le cas où le processus **A** demande le jeton à **C** mais que **C** demande immédiatement après avoir donné son jeton à **A** pour qu'il lui soit retourné à la fin de la section critique de **A**.

2.2 Propriétés

- Correctness:** Jamais plus d'un processus dans la section critique.
- Progress:** Toute demande d'entrée en section critique finit par être satisfaite.
- Complexité de communication:** Entre 0 et $2(n - 1)$ messages par entrée en section critique.

2.3 Pseudo-code

n	entier, constant	nombre de processus
self	entier, entre 0 et $n - 1$	mon numéro de processus
ts	entier, init 0	mon timestamp Lamport actuel
hasRequested	booléen, init false	ssi j'ai fait une demande de SC
selfRequestTs	entier	timestamp de cette demande
requesters	ensemble d'entiers	numéros des processus qui attendent mon jeton.
jetons	ensemble d'entiers	numéros des processus dont j'ai le jeton. Initialement arbitraire.

Fig. 12. – Capture des slides du cours – Variables du pseudo-code

Écouter infiniment les événements suivants:
Demande de SC de la couche applicative
Sortie de SC dans la couche applicative
Réception de {REQ, tsi} du processus i
Réception de {OK, tsi} du processus i

Fig. 13. – Capture des slides du cours – Initialisation du pseudo-code

Traitement : Demande de SC de la couche applicative.
 $ts += 1$
 $hasRequested \leftarrow true$
 $selfRequestTs = ts$
Pour chaque processus i qui n'est pas dans jetons :
Envoi de {REQ, ts}
Si jetons a une taille $n - 1$, entrer en SC

Traitement : Sortie de SC par la couche applicative.
 $ts += 1$
 $hasRequested \leftarrow false$
Pour tout i dans requesters :
Envoi de {OK, ts}
jetons.remove(i)
Réinitialisation de requesters

Fig. 14. – Capture des slides du cours – Fonctionnement du pseudo-code

Traitement : Réception {REQ, tsi} du processus i.
 $ts \leftarrow \max(tsi, ts) + 1$
Si hasRequested est
($selfRequestTs < tsi$ ou
 $(selfRequestTs == tsi \text{ et } self < i)$)
Ajout de i dans requesters
Sinon
Envoi de {OK, ts} à i
jetons.remove(i)
Si hasRequested
Envoi de {REQ, selfRequestTs} à i

Fig. 15. – Capture des slides du cours – Fonctionnement du pseudo-code

Traitement : Réception {OK, tsi} du processus i.
 $ts \leftarrow \max(tsi, ts) + 1$
jetons.add(i)
Si jetons a une taille $n - 1$ et hasRequested :
Entrer en SC

Fig. 16. – Capture des slides du cours – Fonctionnement du pseudo-code