SDR

Jetons

Mutex par jeton unique (Raymond)

Olivier Lemer

Rappels

Suppositions

Pas de **duplication** de message

Pas de **perte** de message

Pas de **changement d'ordre** de message

Pas de **panne** de processus

Concepts

Horloge logique

Heure virtuelle donnée à tout évènement, tel qu'un ordre strict existe.

Mutex

Un seul processus à la fois peut être dans l'état SC.

Lamport

*"Si ma requête est **le plus ancien message** que j'aie reçu, je peux entrer en SC"*

*"Je suis responsable de maintenir une idée de l'**état des autres**."*

Ricart et Agrawala

*"**Si tout le monde est OK** que je le fasse, je peux entrer en SC"*

*"Je ne suis responsable que de dire quand **c'est bon pour moi** qu'un autre soit en SC."*

Carvalho et Roucairol

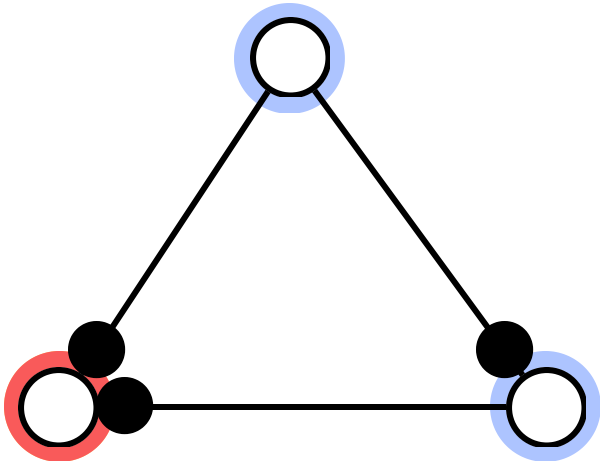
*"**Si j'ai les jeton de tout le monde**, je peux entrer en SC"*

*"Je ne suis responsable que de **donner mon jeton** si je n'en ai pas besoin"*

↓ Jetons : un ou plusieurs ?

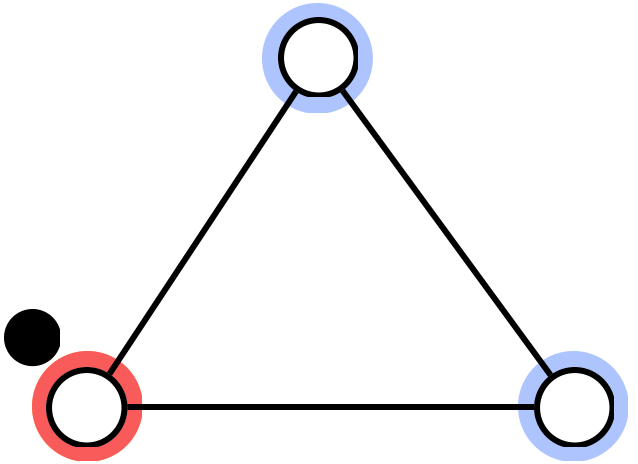
Programmation par jetons

Algorithme par jetons
Un jeton par paire de voisins.



"Il me faut le jeton de tout le monde pour avoir le droit d'être en SC."

Algorithme par jetons ✗
Un jeton, tout court.



*"Il me faut **le jeton** pour avoir le droit d'être en SC."*

Algorithme par jeton unique

Enjeux

Unicité *(Correctness)*
Assurer que le jeton ne sera pas dupliqué ni perdu

Transmission
Gérer le transit du jeton d'un demandeur à un autre

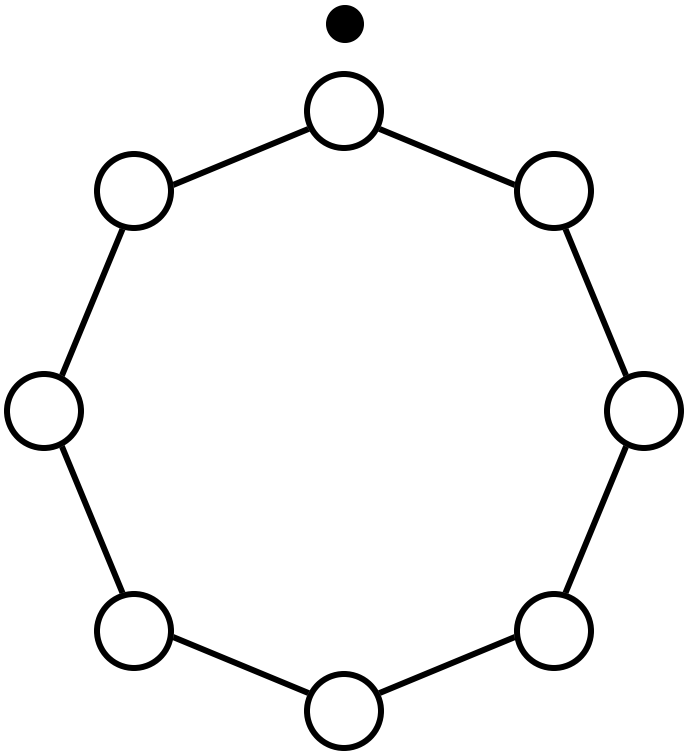
Progrès
Assurer que tout demandeur aura le jeton un jour

En général, trouver un protocole de **transmission** assurant **unicité** et **progrès**.

Mutex sur un anneau

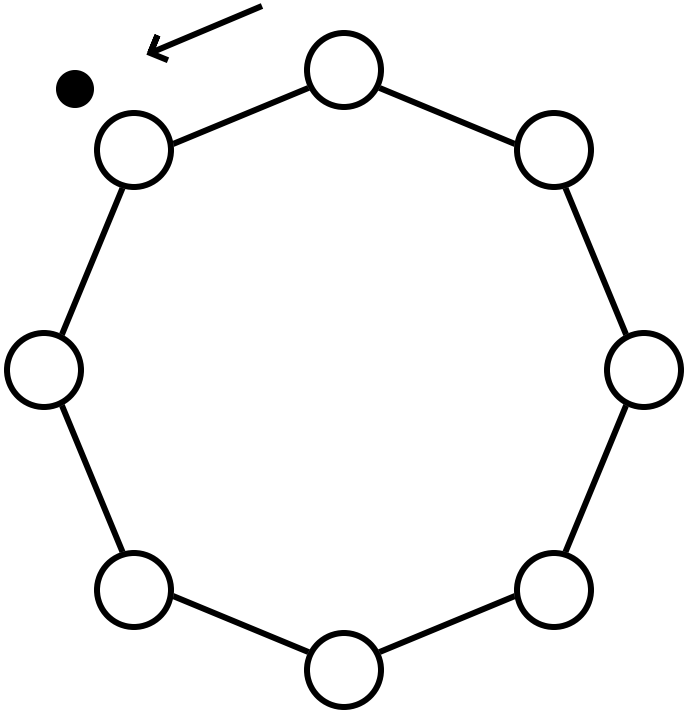
Idée

Solution simple



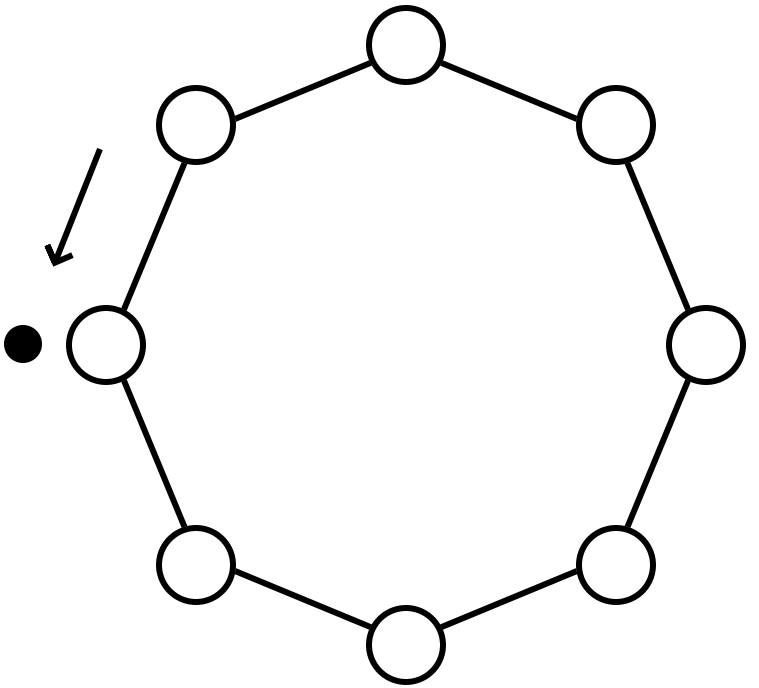
Solution simple

Jeton transite sur la boucle infiniment



Solution simple

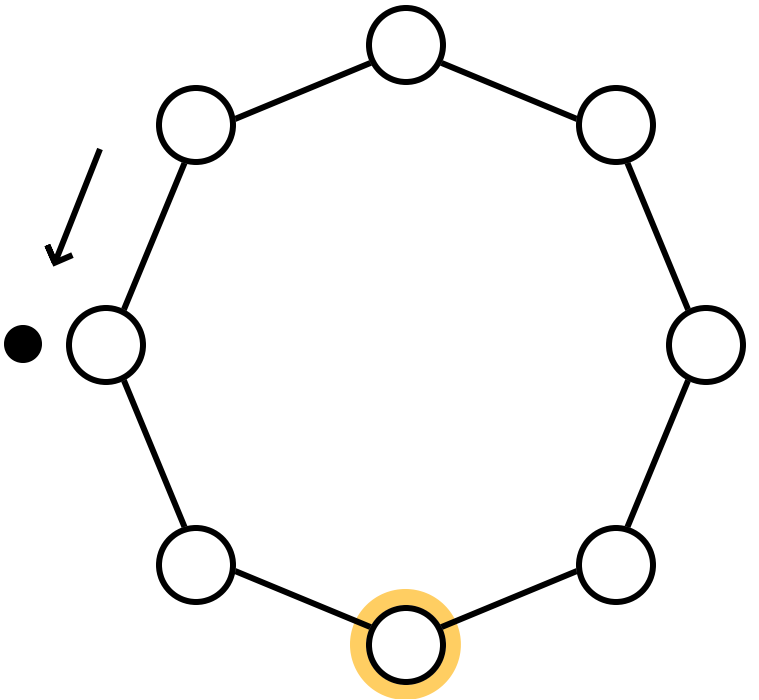
Jeton transite sur la boucle infiniment



Solution simple

Jeton transite sur la boucle infiniment

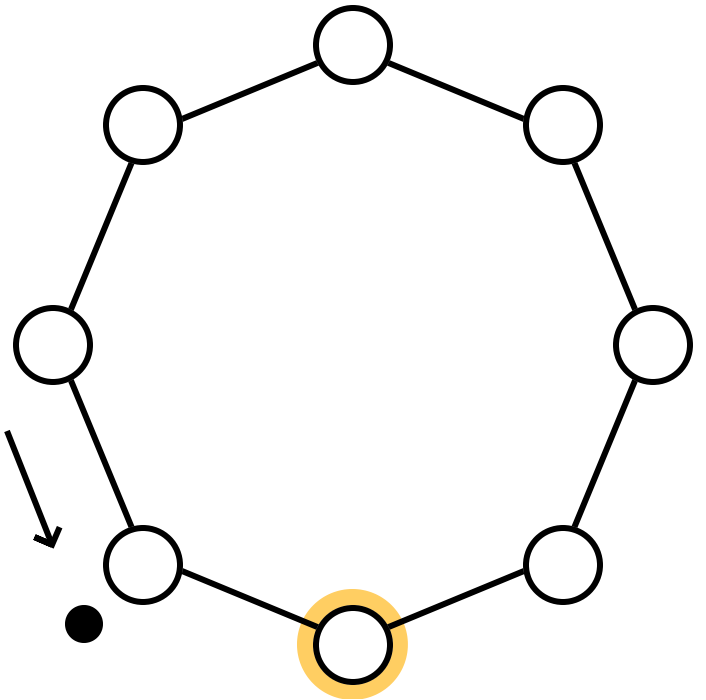
Un demandeur attend que le jeton lui arrive



Solution simple

Jeton transite sur la boucle infiniment

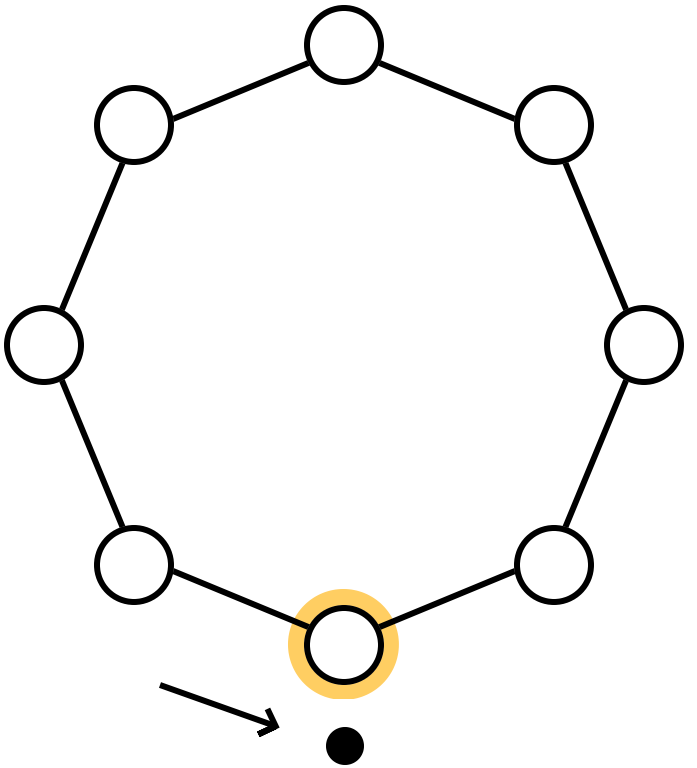
Un demandeur attend que le jeton lui arrive



Solution simple

Jeton transite sur la boucle infiniment

Un demandeur attend que le jeton lui arrive

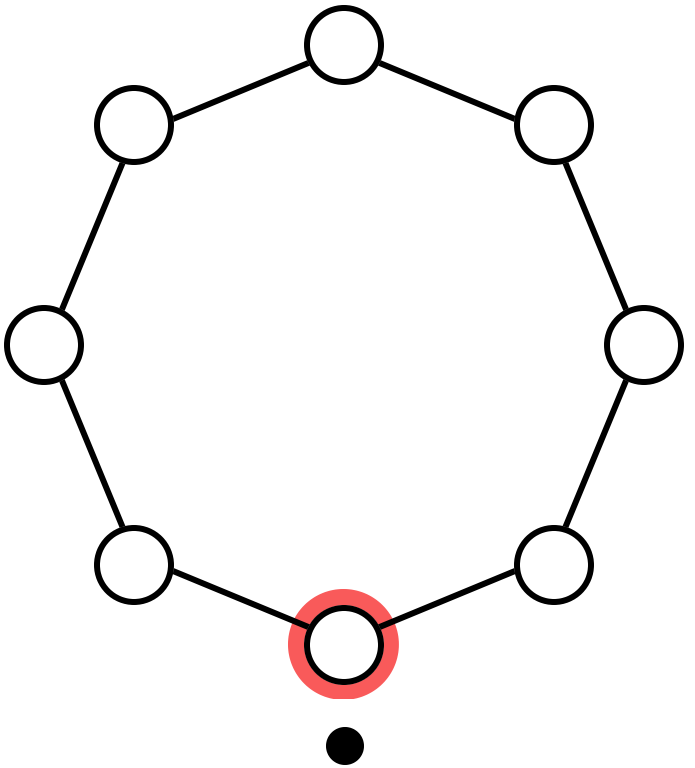


Solution simple

Jeton transite sur la boucle infiniment

Un demandeur attend que le jeton lui arrive

Il garde alors le jeton pendant sa SC



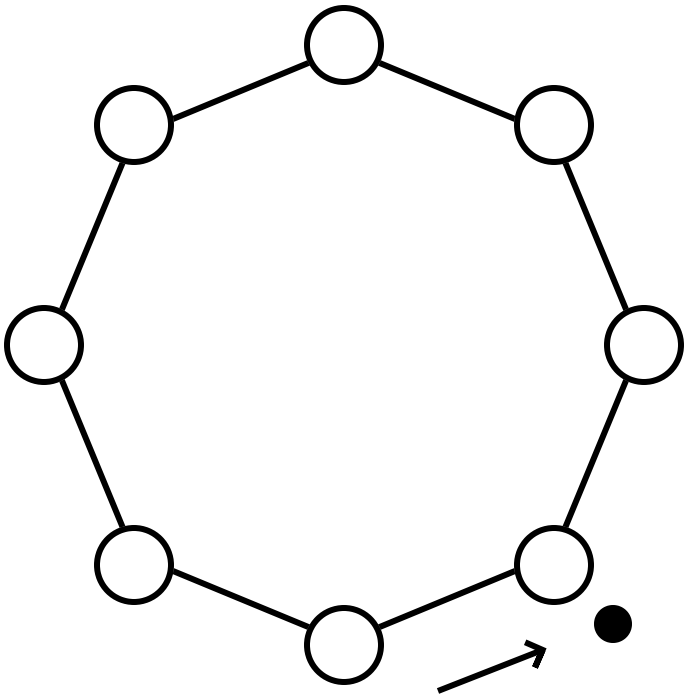
Solution simple

Jeton transite sur la boucle infiniment

Un demandeur attend que le jeton lui arrive

Il garde alors le jeton pendant sa SC

Jusqu'à avoir fini, puis le fait passer plus loin



Solution simple

Propriétés

Avantages

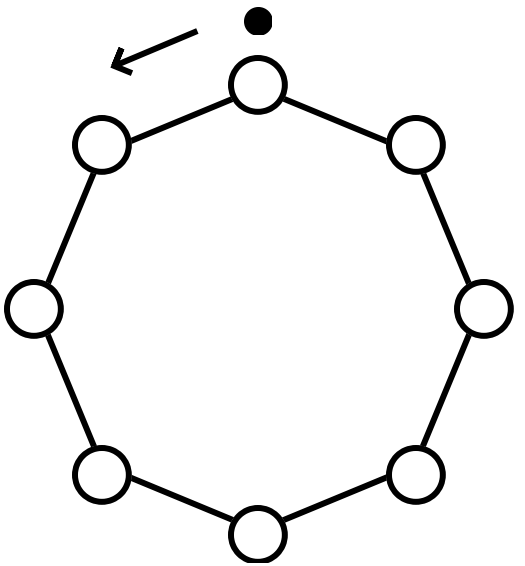
Correctness évidente

Un seul jeton, donc une seule SC

Inhérent à tout algo à jeton unique

Progrès évident

Jeton passe chez tout le monde périodiquement



Mais

Complexe si pannes autorisées

Risque de perte de jeton et de famine.

Besoin de protocole de régénération.

Inhérent à tout algo à jeton unique

Inefficace

Temps d'attente linéaire après requête.

Évitable

Mutex sur un arbre

Idée

L'arbre - une autre structure

Efficacité

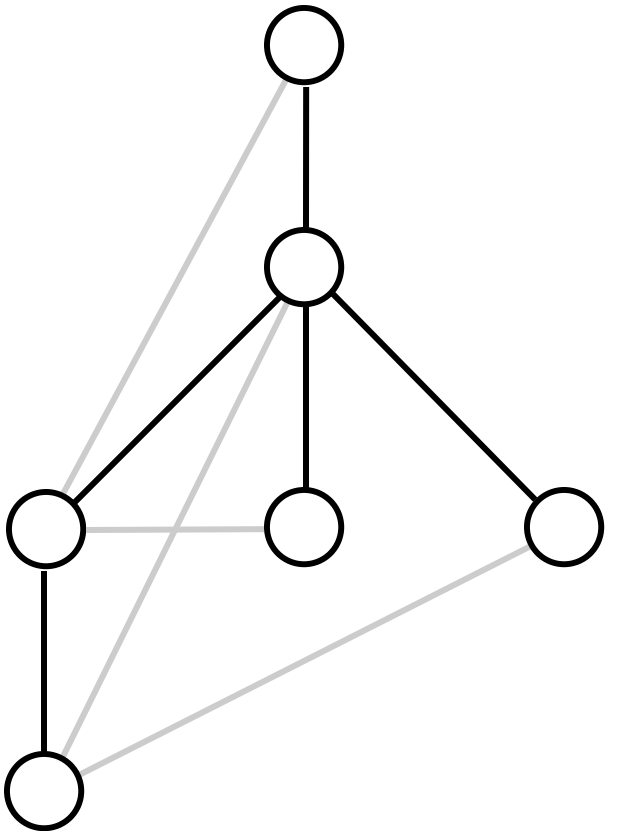
S'il est équilibré, distances logarithmiques.

Simplicité

Un seul chemin entre tous deux points.

Réalisme

Les vrais réseaux sont rarement des cliques.



(Pourra être intégré sur un réseau plus complet)

Mutex sur un arbre

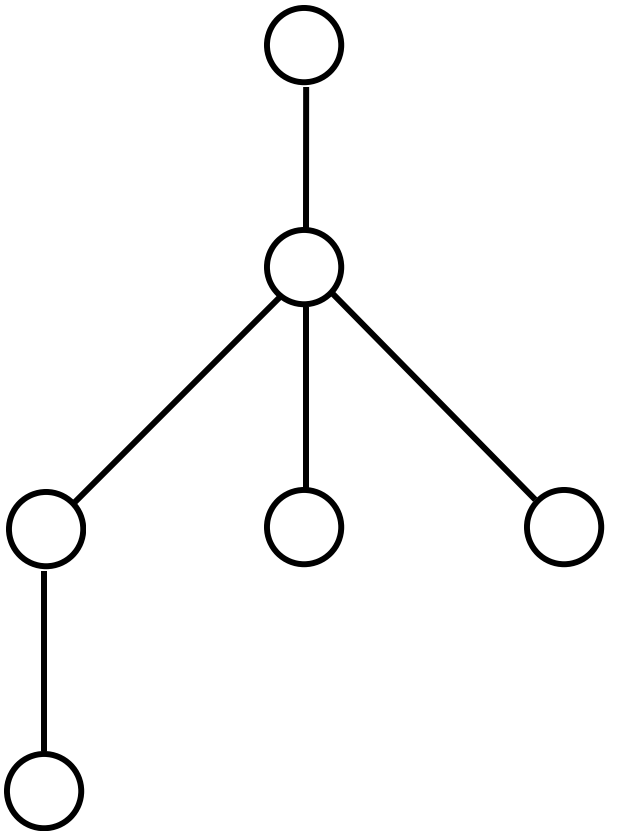
Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ` *"Donne-moi le jeton"*

`OK` *"Tiens"*

Trouver où se trouve le jeton



Mutex sur un arbre

Implications du choix de l'arbre

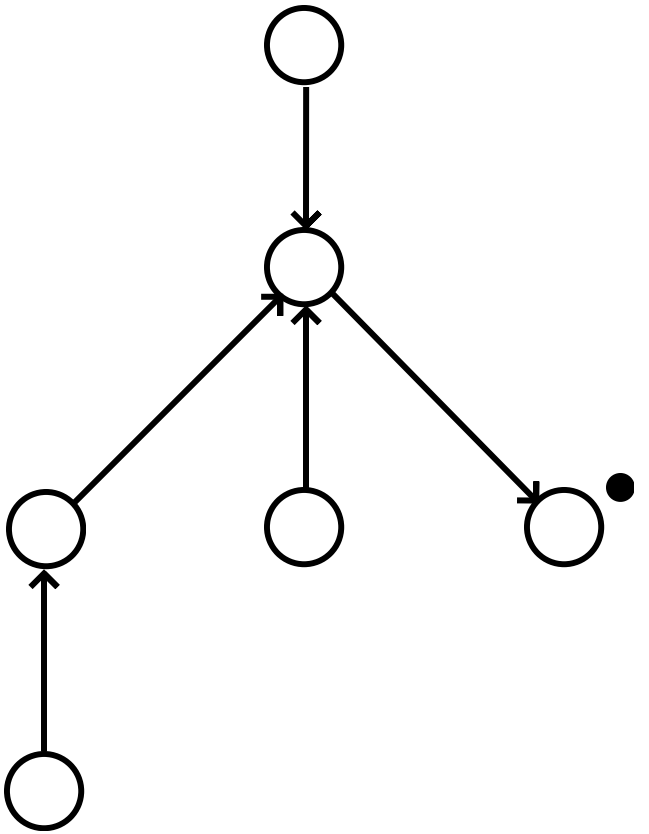
Repasser à un système de requêtes

`REQ` *"Donne-moi le jeton"*

`OK` *"Tiens"*

Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.



Mutex sur un arbre

Implications du choix de l'arbre

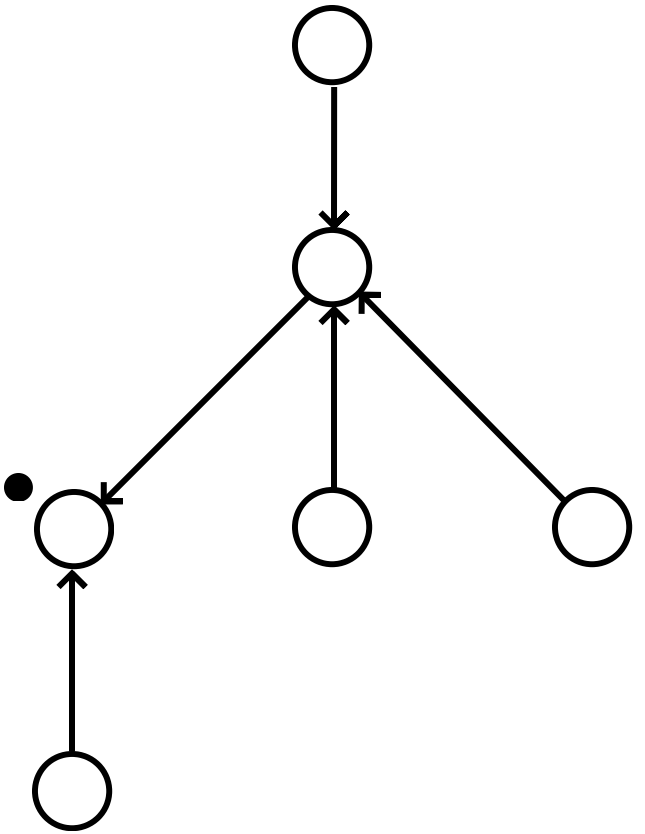
Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

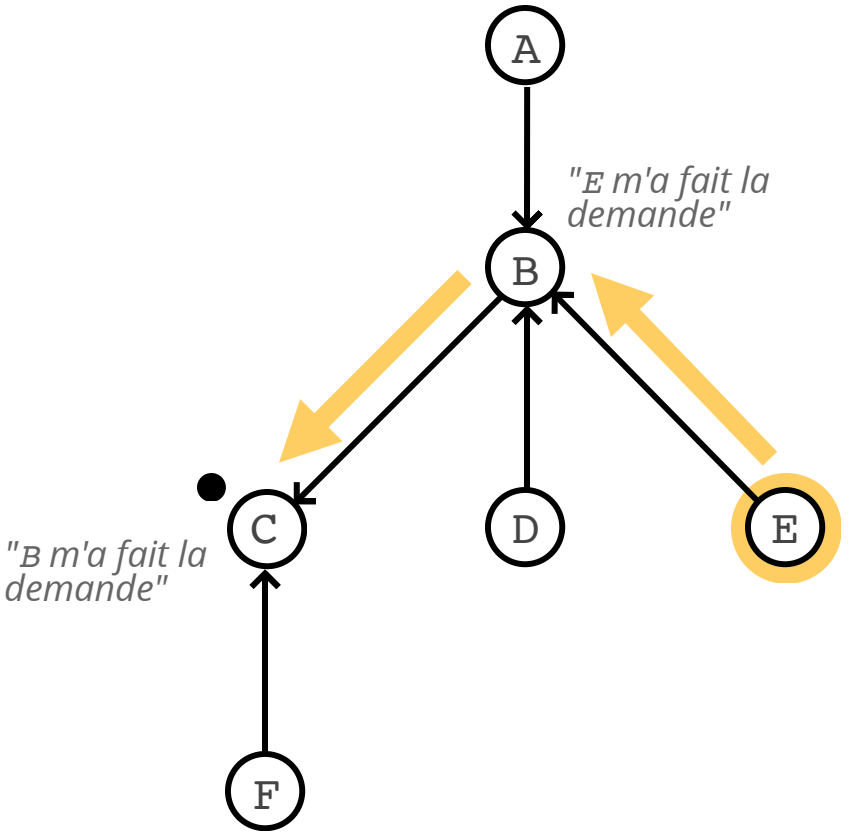
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Transmettre les requêtes via ces liens.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

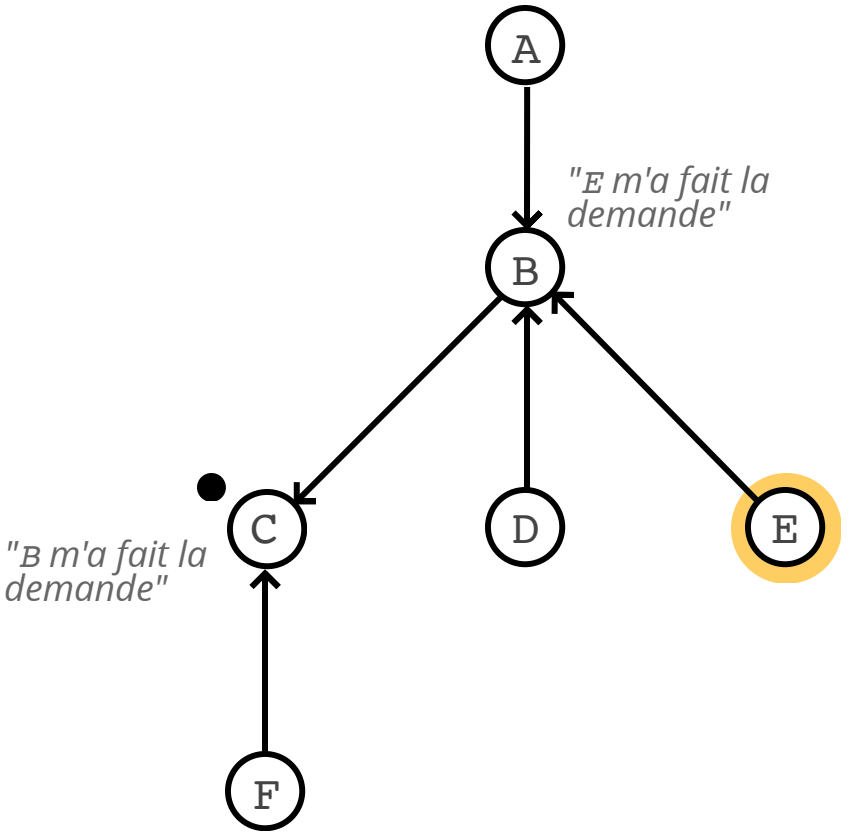
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Transmettre les requêtes via ces liens.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

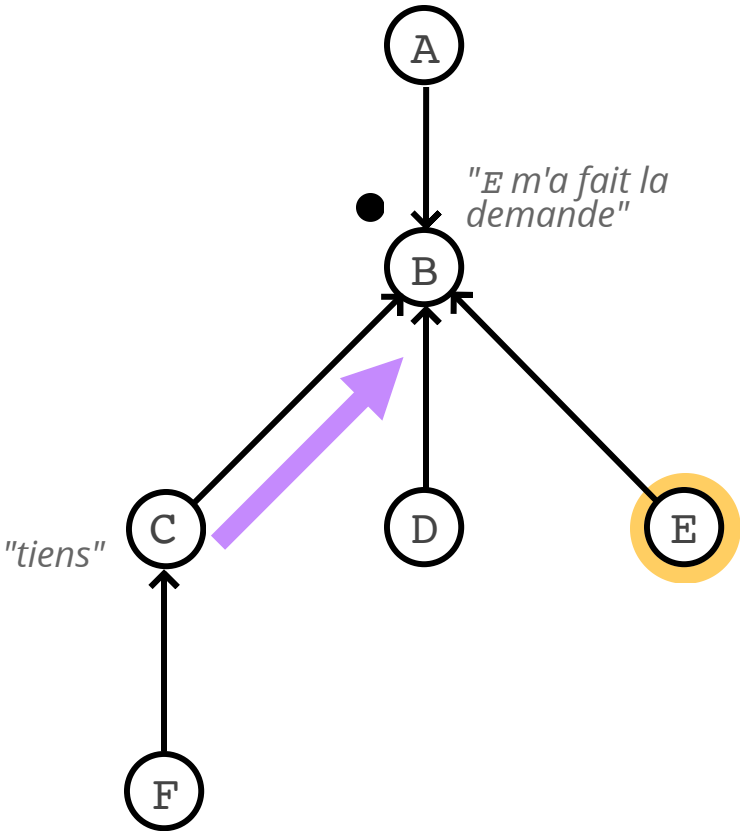
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

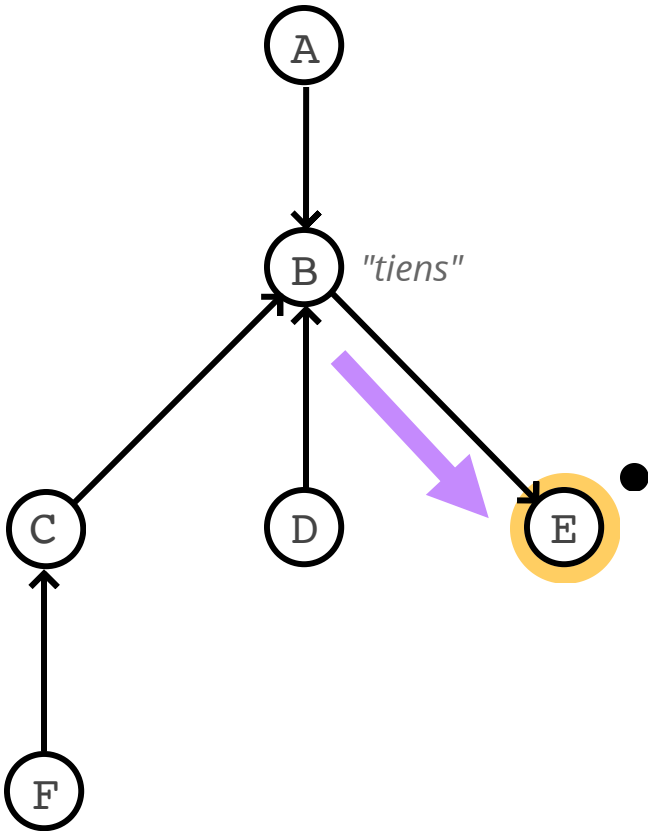
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

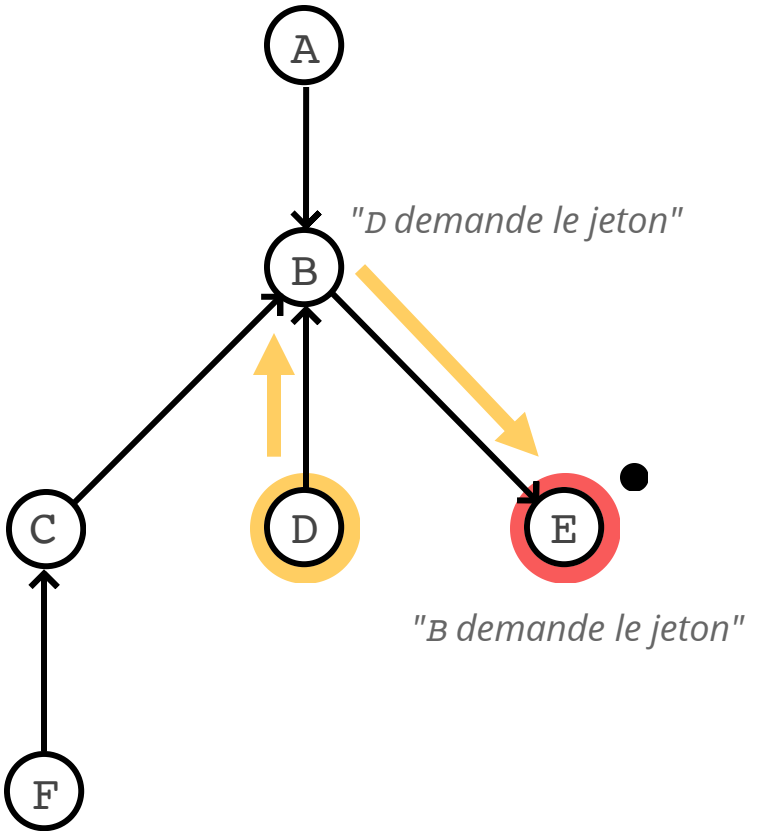
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

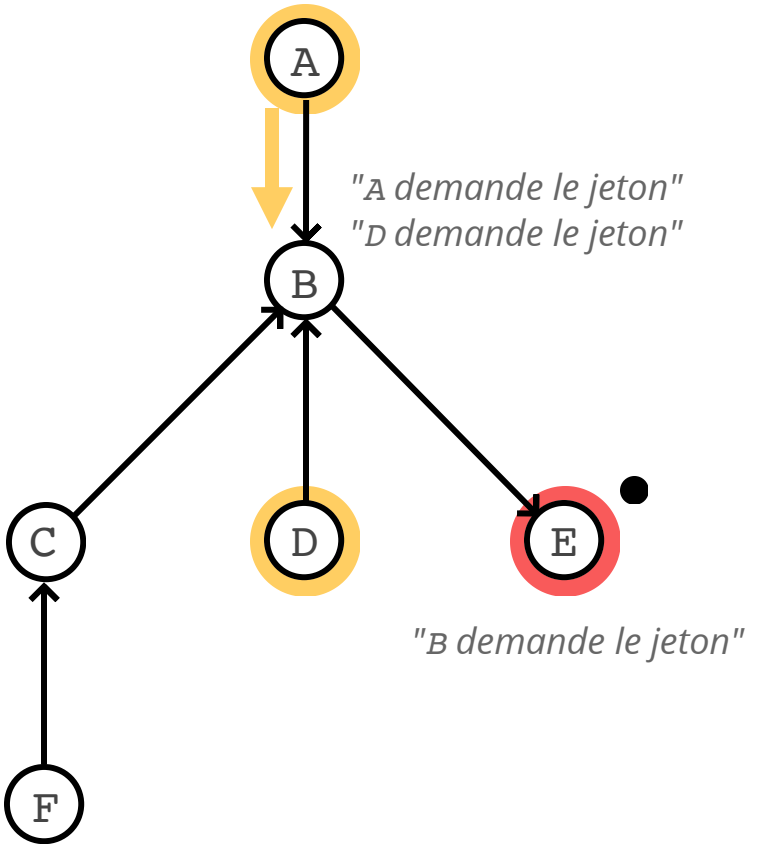
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"
sous forme de FIFO



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

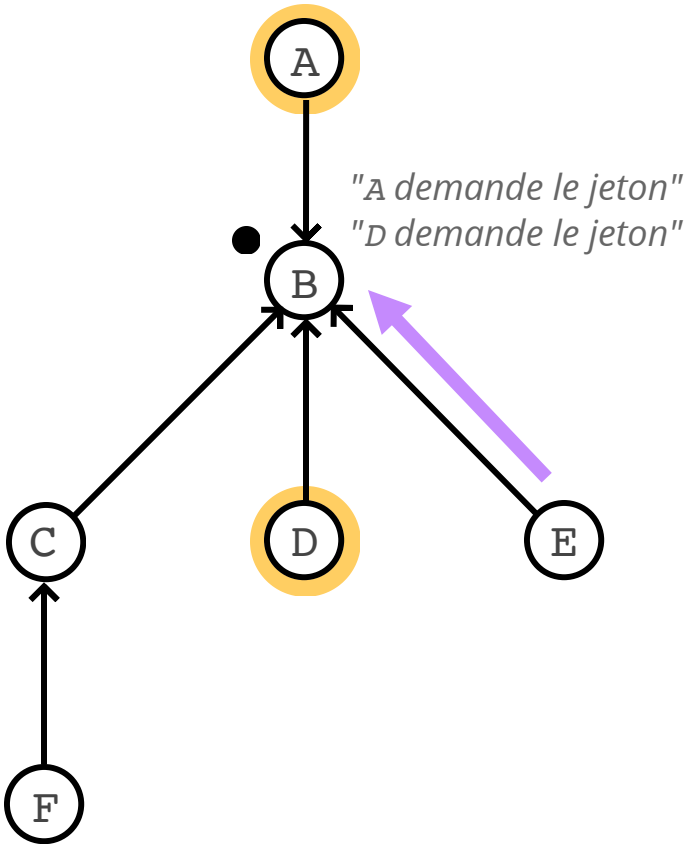
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"
sous forme de FIFO



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

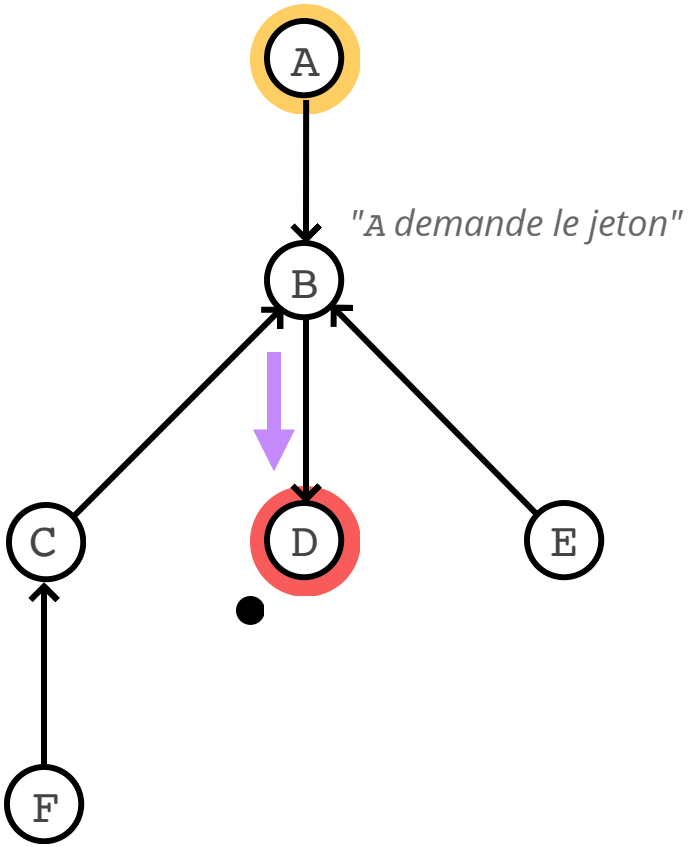
Trouver où se trouve le jeton

Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"
sous forme de FIFO



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

Trouver où se trouve le jeton

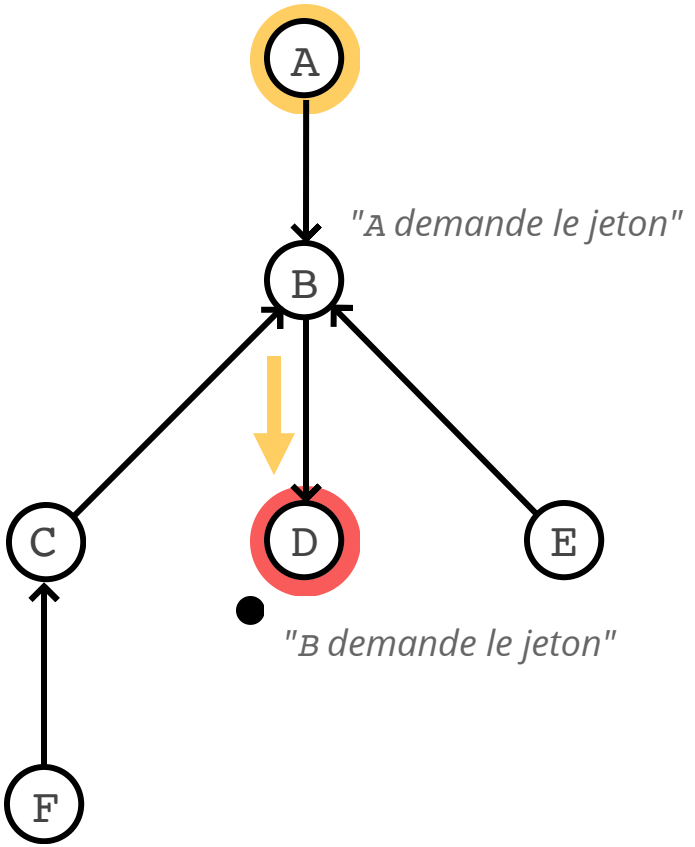
Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"
sous forme de FIFO

Et envoyer une requête après le jeton s'il y en a d'autres dans la queue



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

Trouver où se trouve le jeton

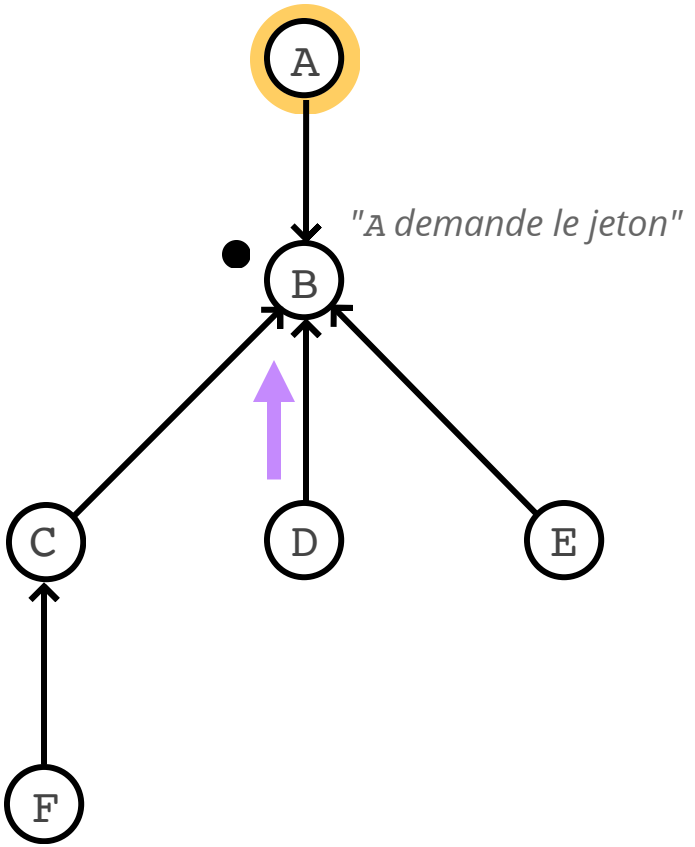
Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"
sous forme de FIFO

Et envoyer une requête après le jeton s'il y en a d'autres dans la queue



Mutex sur un arbre

Implications du choix de l'arbre

Repasser à un système de requêtes

`REQ`
"Donne-moi le jeton"

`OK`
"Tiens"

Trouver où se trouve le jeton

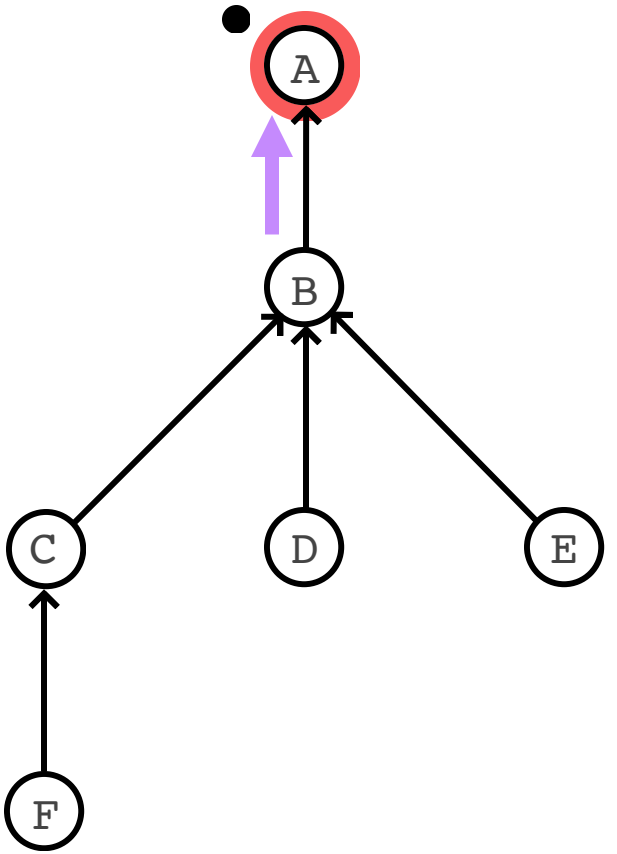
Rendre les liens **dirigés**
vers le voisin le plus proche du jeton.

Les maintenir ainsi.

Se souvenir d'où la requête vient

Se souvenir de "qui m'a demandé le jeton"
sous forme de FIFO

Et envoyer une requête après le jeton s'il y en a d'autres dans la queue





Mutex sur un arbre

Résumé

Algorithme par jeton unique

Résumé

Deux messages

`REQ`	`OK`
"Donne-moi le jeton"	*"Tiens"*

Des liens toujours dirigés vers le jeton

Maintenu à jour quand le jeton bouge.

Permet de savoir où envoyer une requête.

Une file d'attente par processus

Maintient la liste des demandes de jeton.

Permet de savoir à qui renvoyer le jeton.

Renvoi d'une requête après le jeton si besoin

Permet d'informer le nouveau possesseur
qu'il s'appelle "reviens".

Algorithme par jeton unique

App veut entrer en SC

Envoie d'un `REQ` à moi-même

Réception d'un `REQ`

Push l'envoyeur dans ma file

Si je n'ai pas le jeton

- Si je n'ai pas de `REQ` en attente
  - J'envoie un `REQ` à mon parent

Si j'ai le jeton

- Décider quoi faire du jeton

On push le voisin qui nous a fait le requête, pas le processus qui en est à l'origine

App sort de SC

Décider quoi faire du jeton

Réception d'un `OK`

Décider quoi faire du jeton

Décider quoi faire du jeton

Si ma file est vide, ne rien faire

Sinon, pop la tête de ma file

Si c'est moi

- J'entre en SC

Si c'est un autre processus `p`

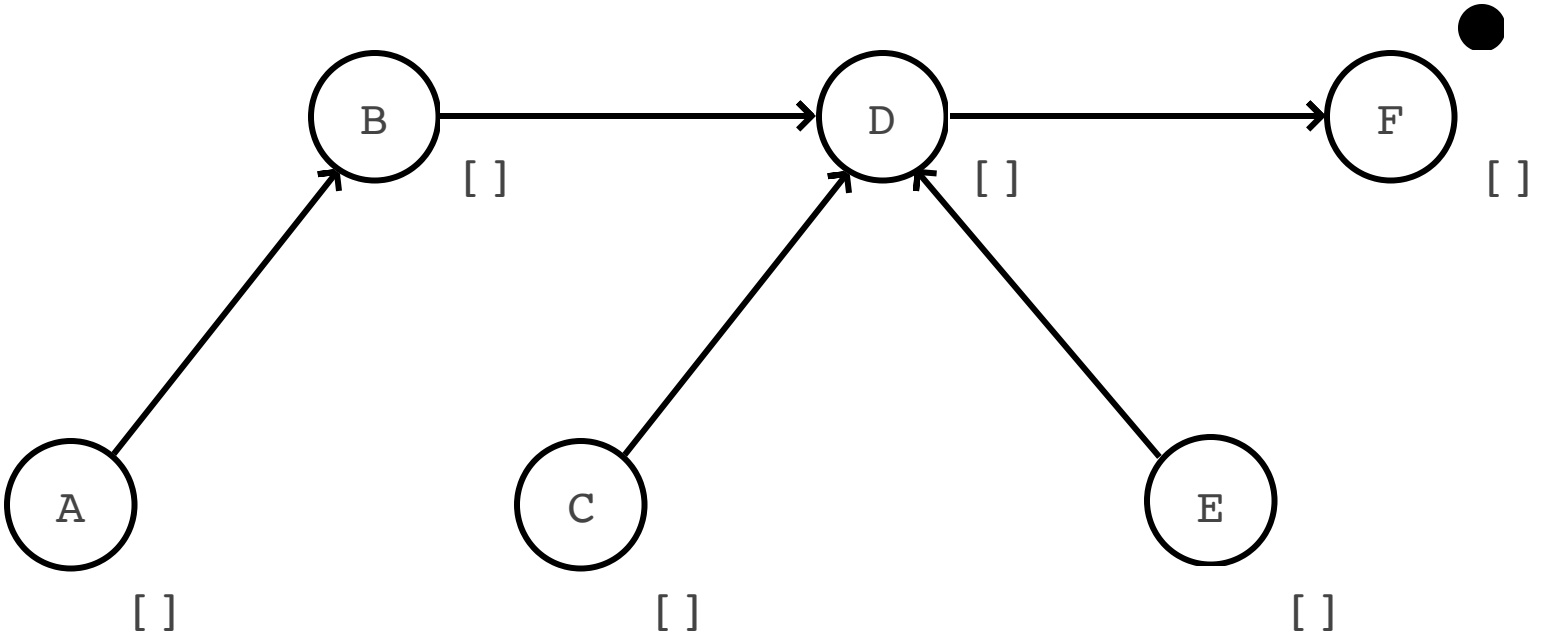
- Je lui passe le jeton
- Mon parent devient `p`
- Si ma file d'attente n'est pas vide
  - J'envoie un `REQ` à `p`



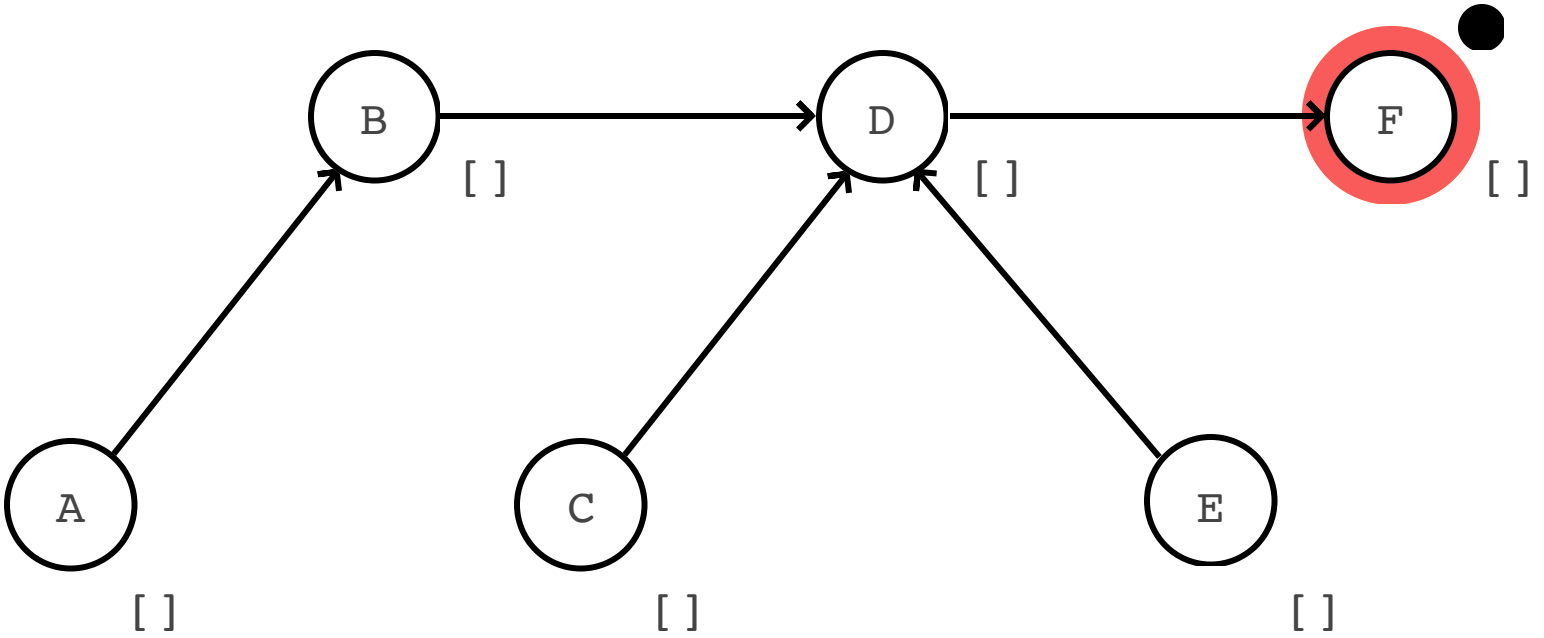
Mutex sur un arbre

Exemple

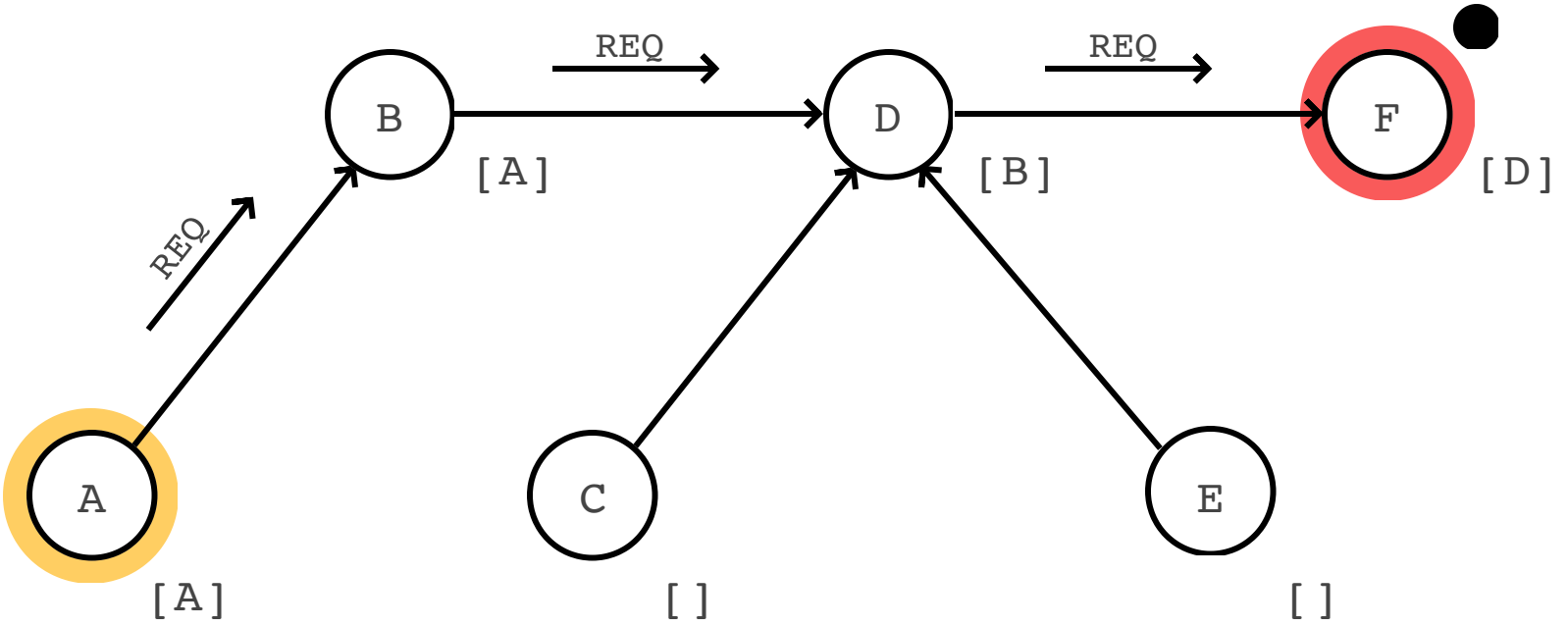
Exemple



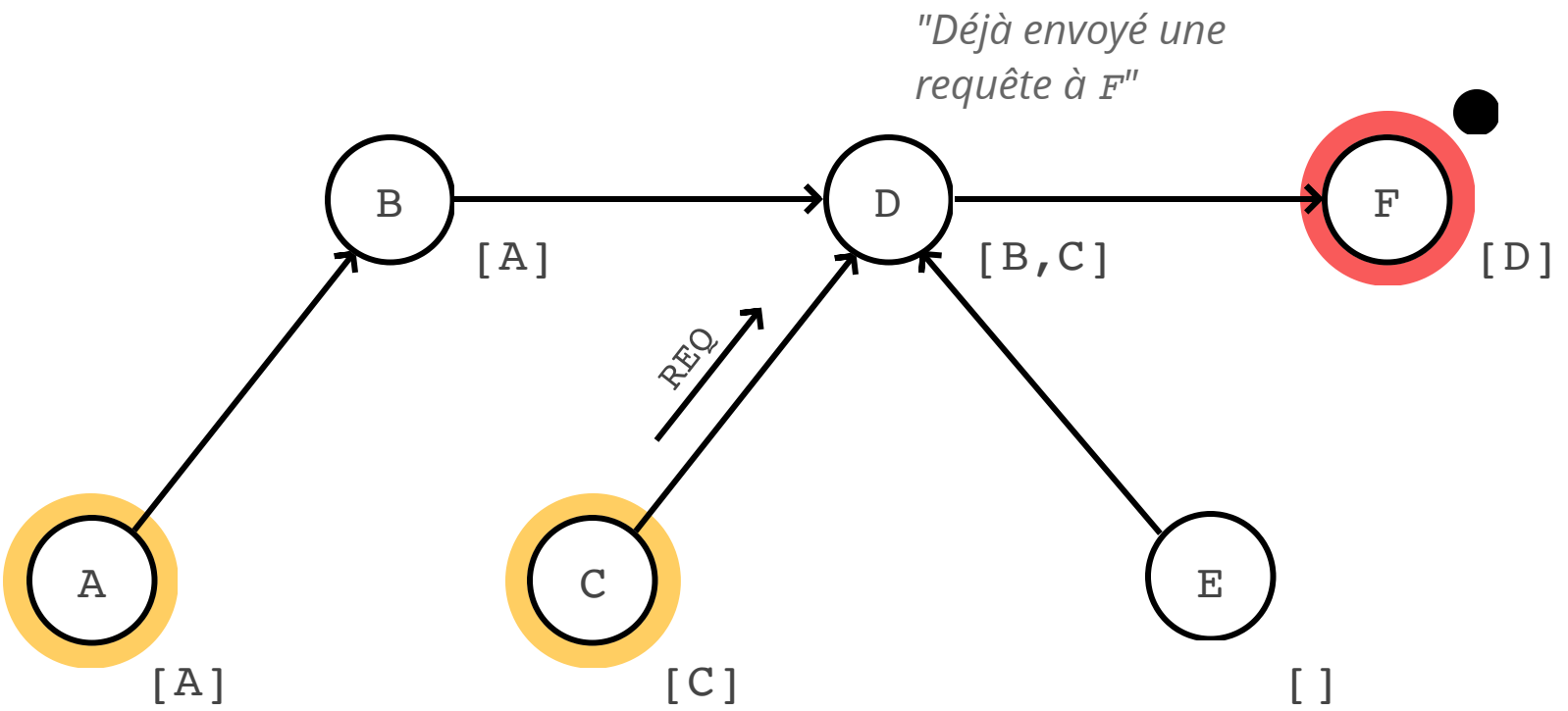
Exemple



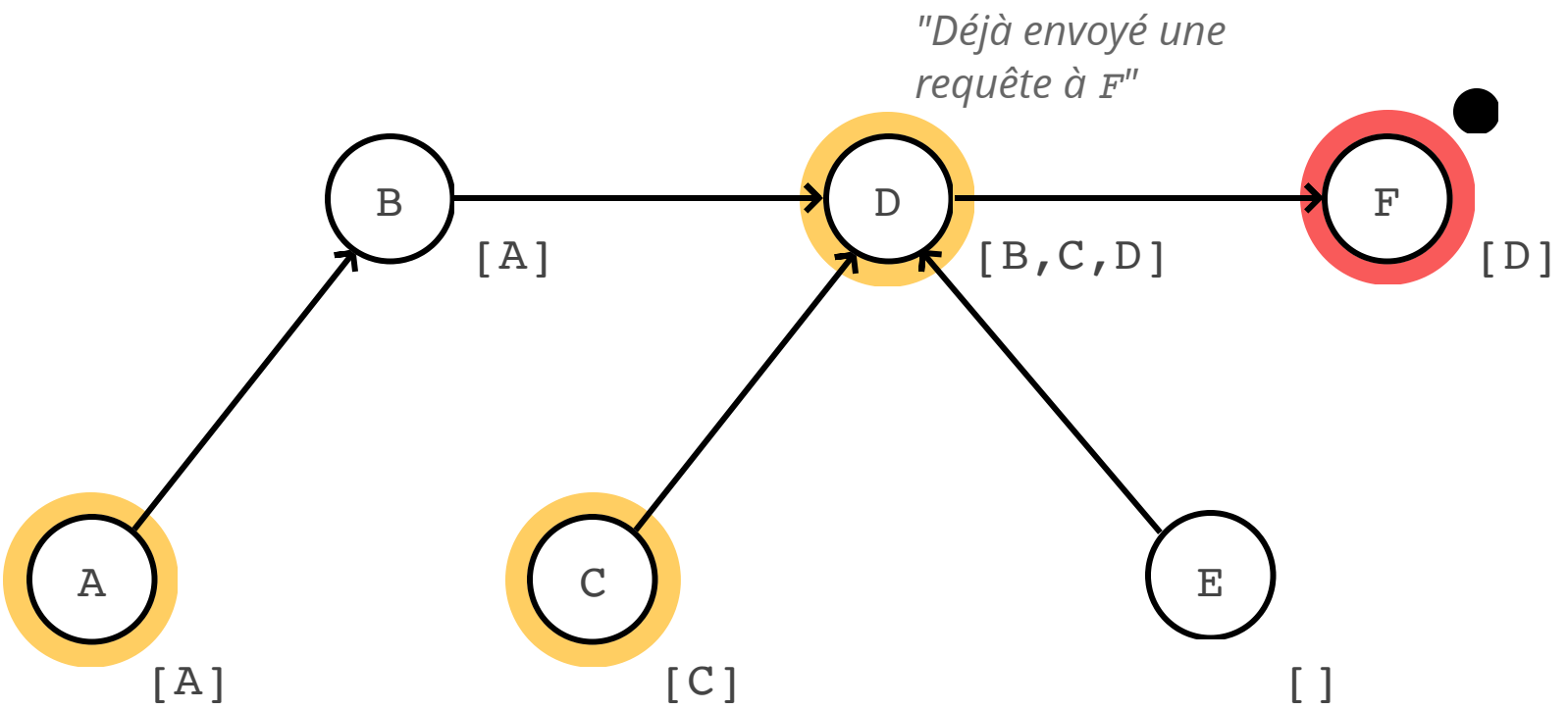
Exemple



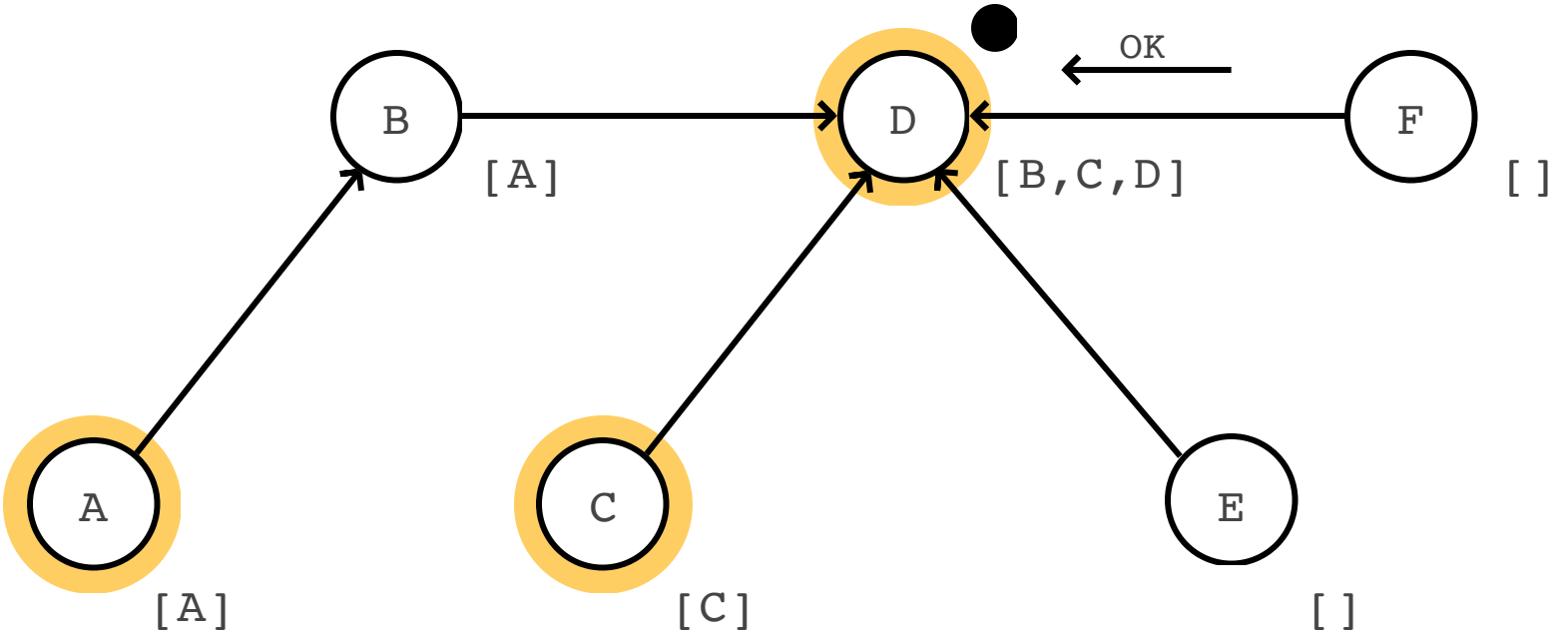
Exemple



Exemple

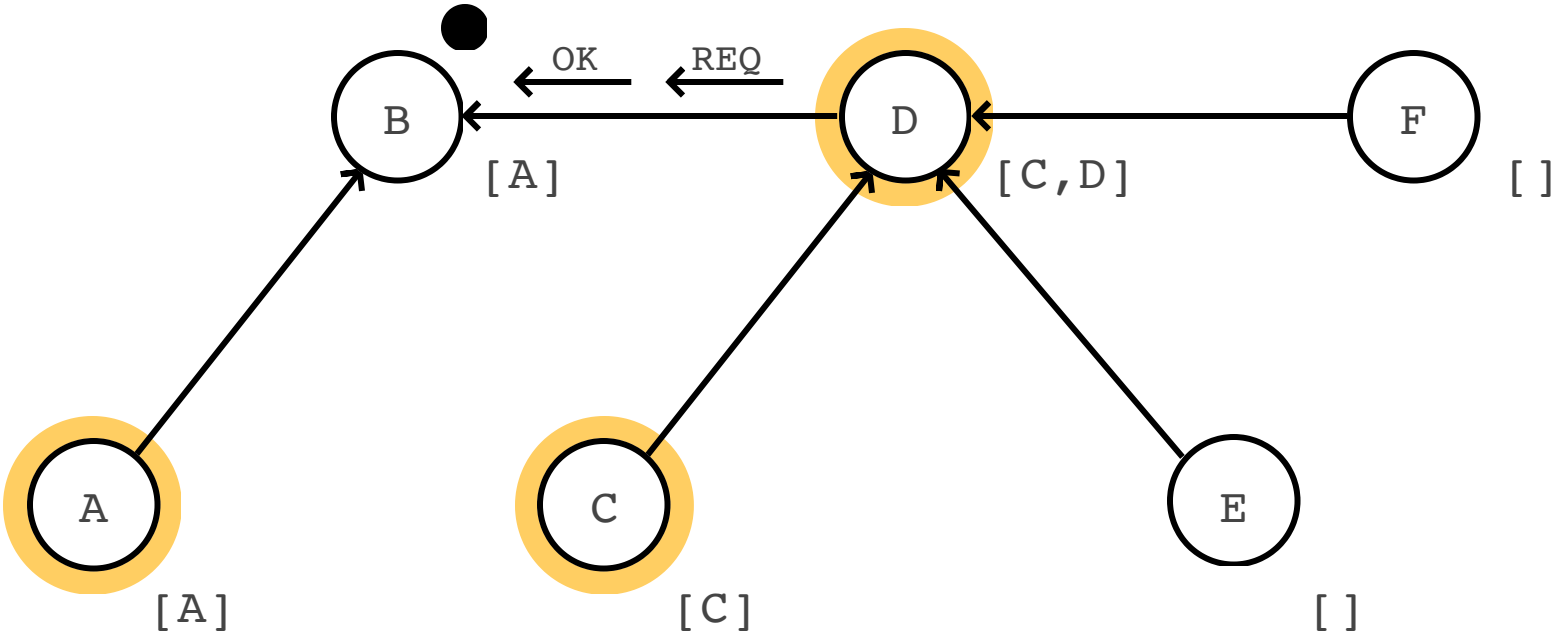


Exemple

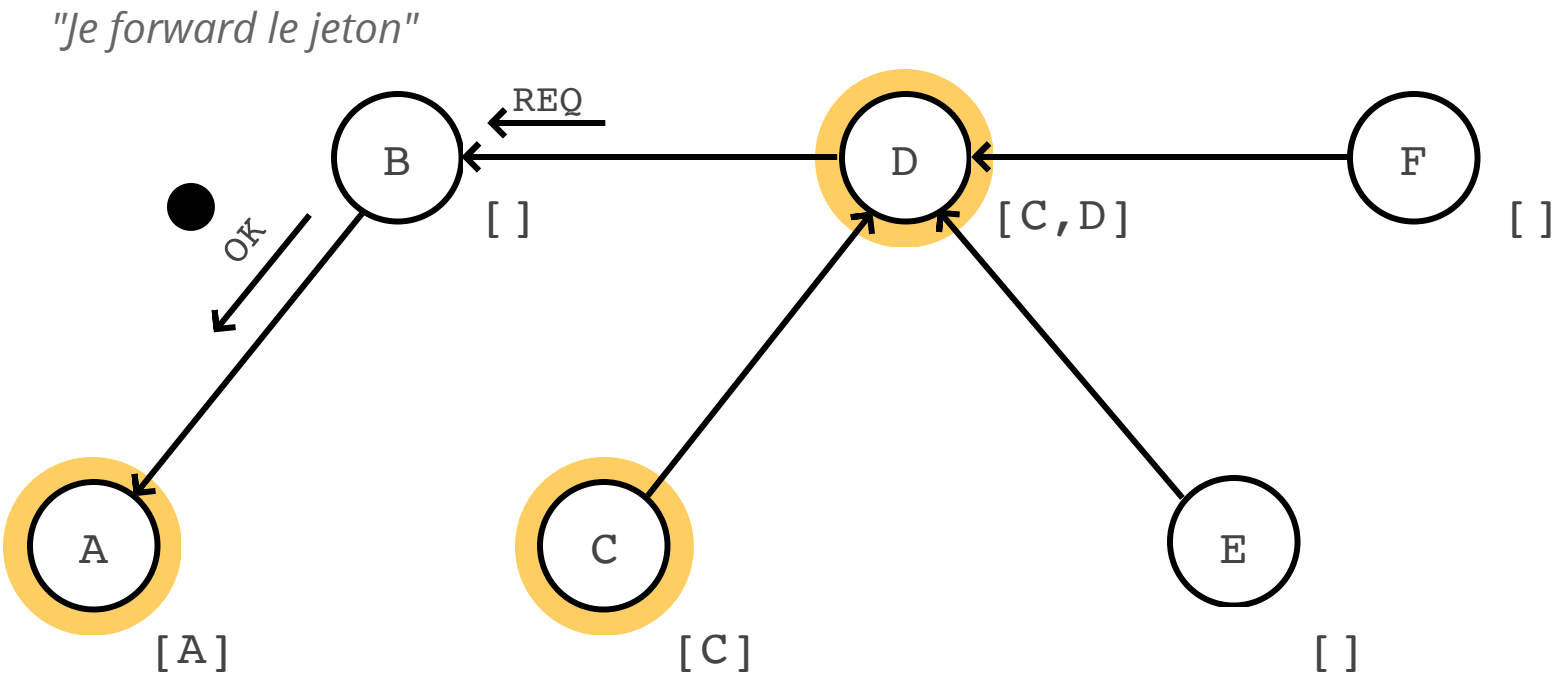


Exemple

"B est pop, je lui passe,
mais il doit me le rendre"

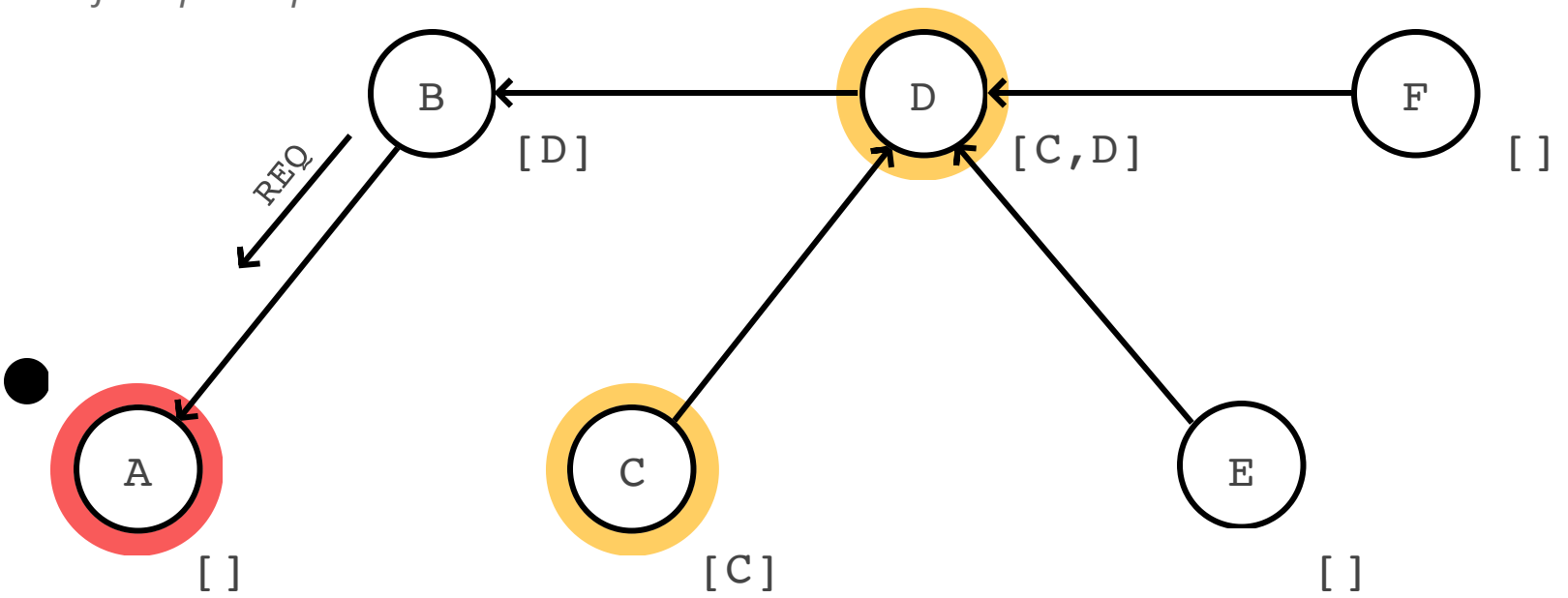


Exemple

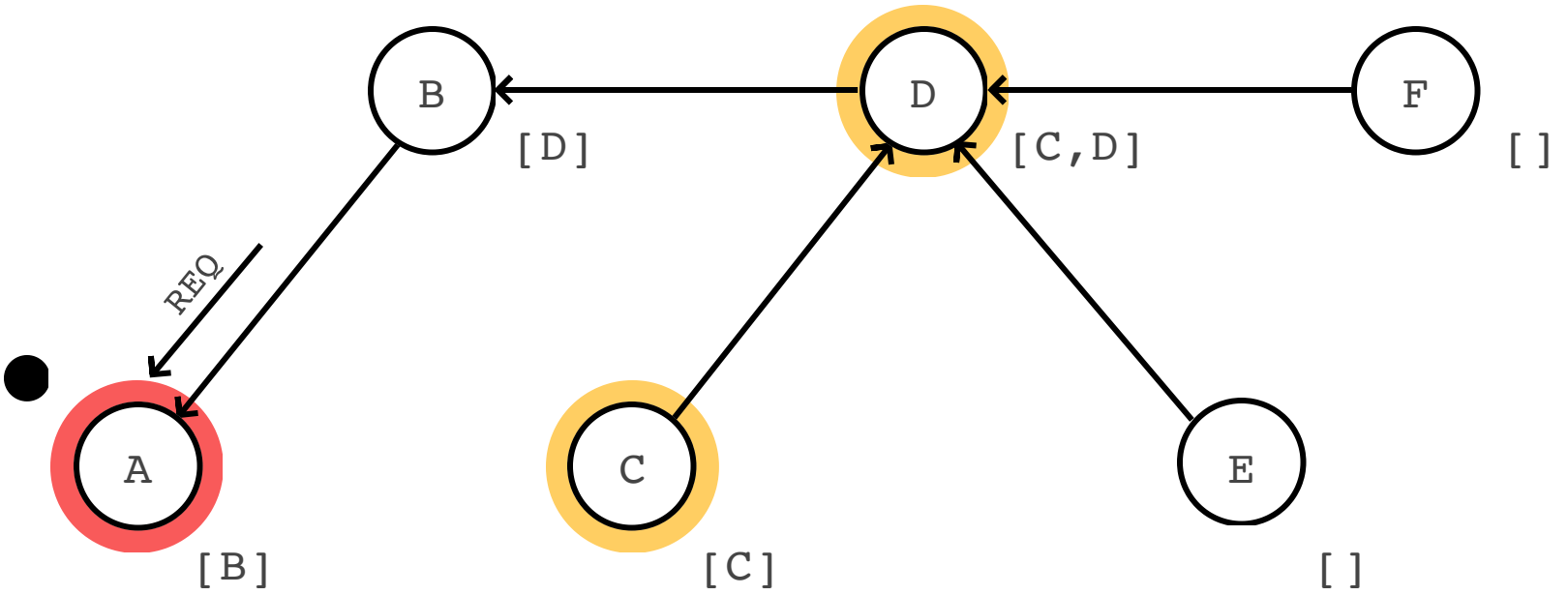


Exemple

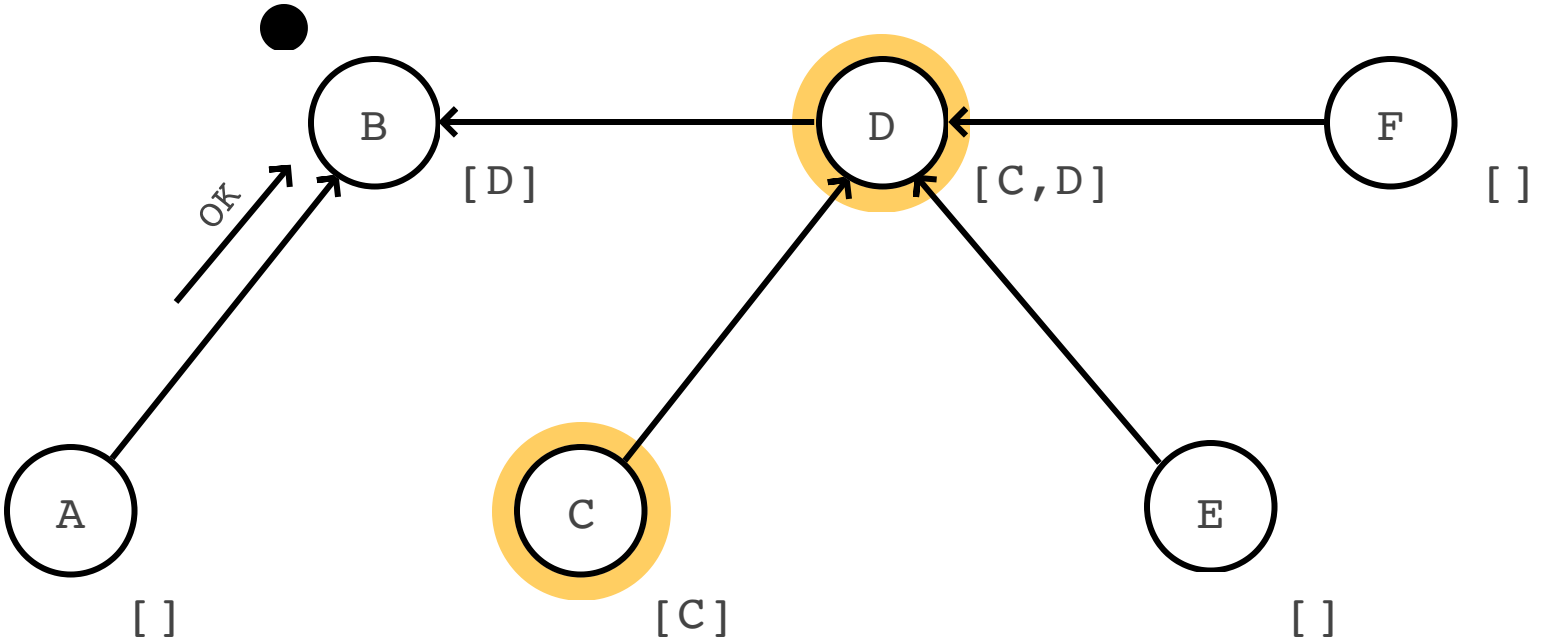
"Je reçois une requête,
je la passe plus loin"



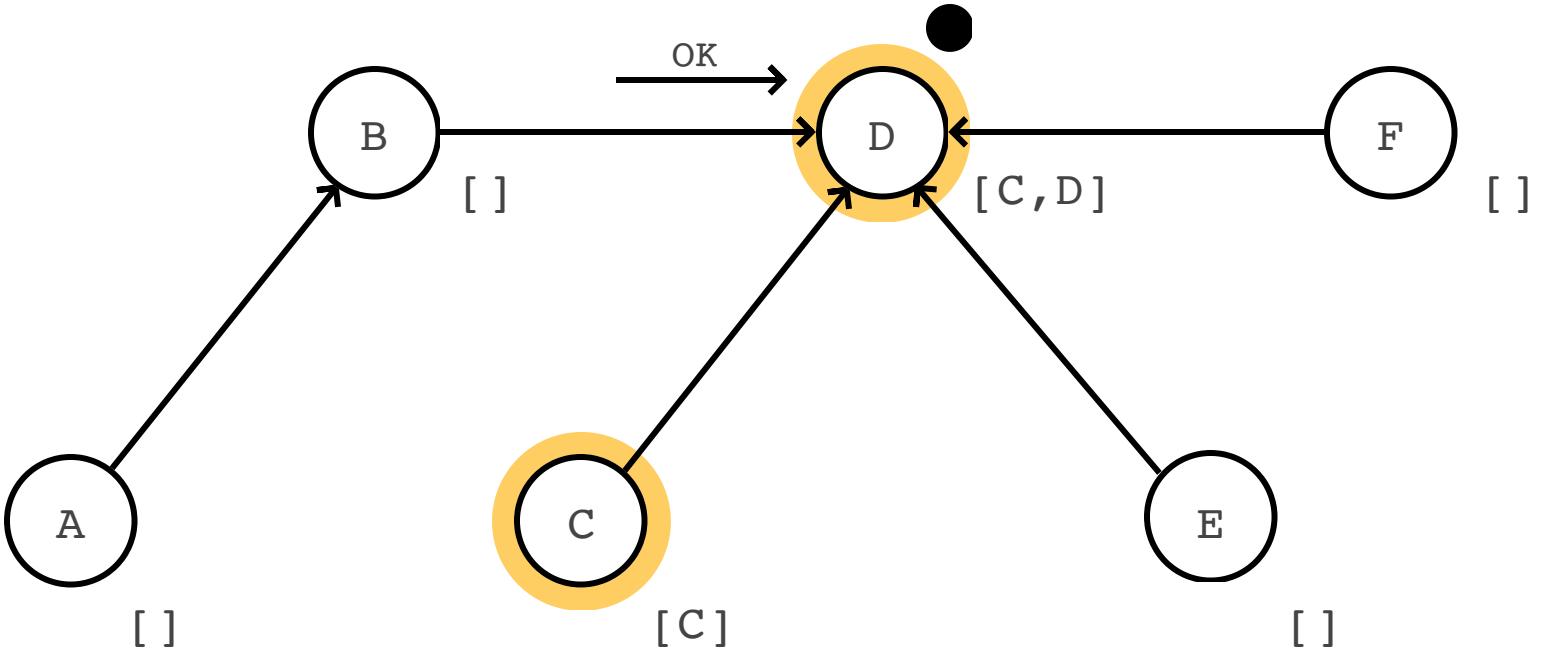
Exemple



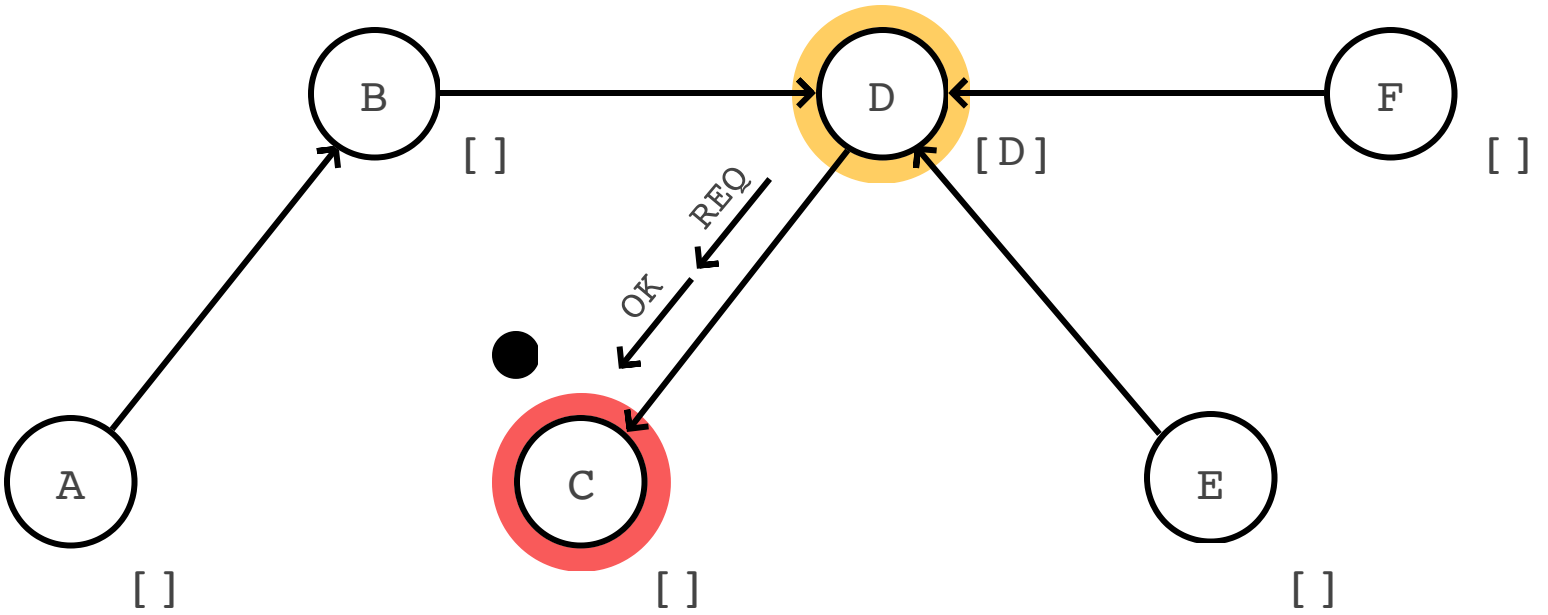
Exemple



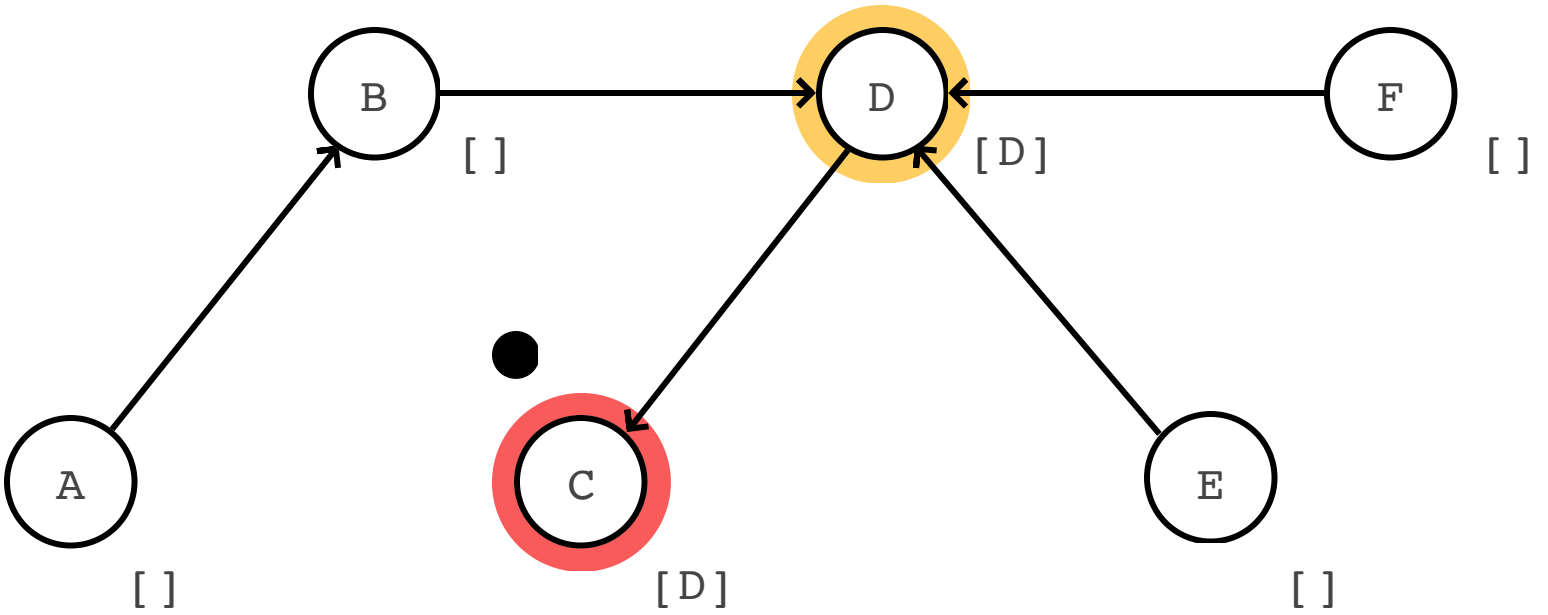
Exemple



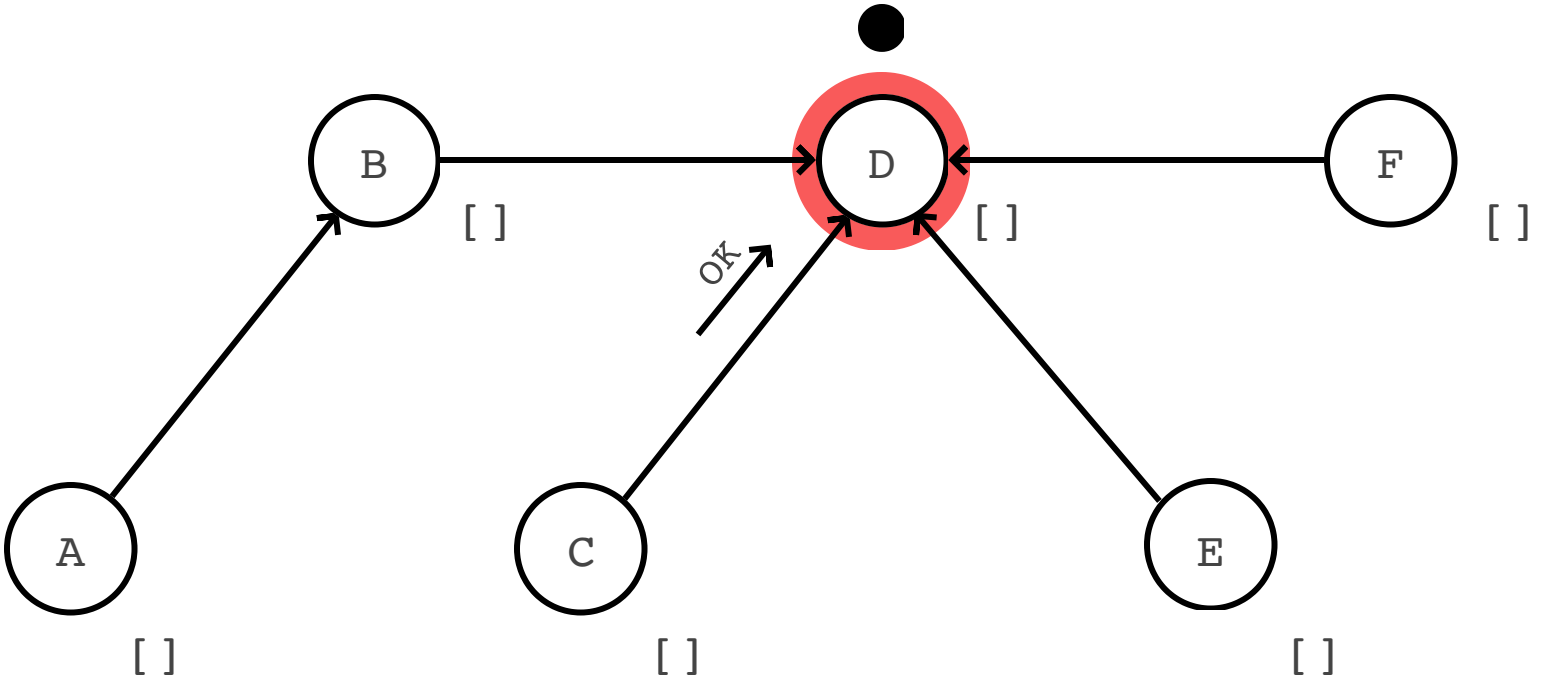
Exemple



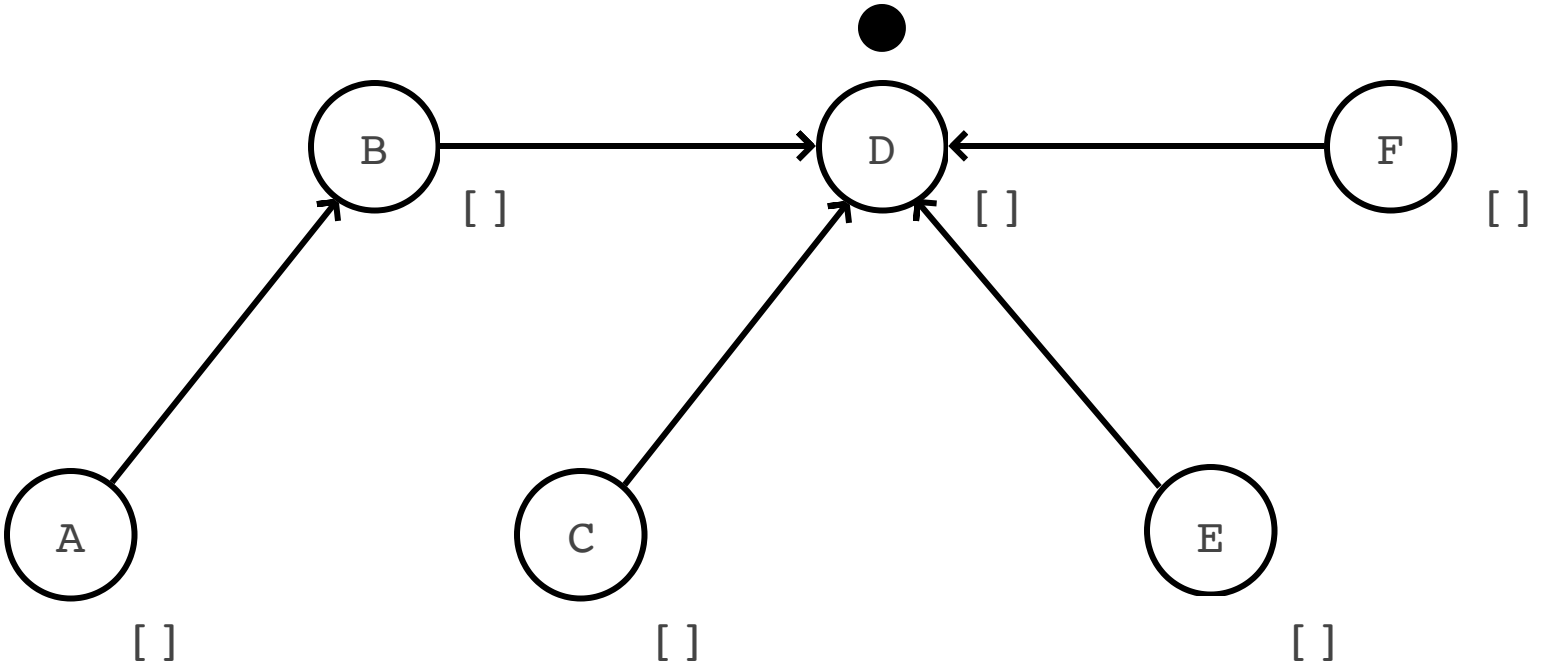
Exemple



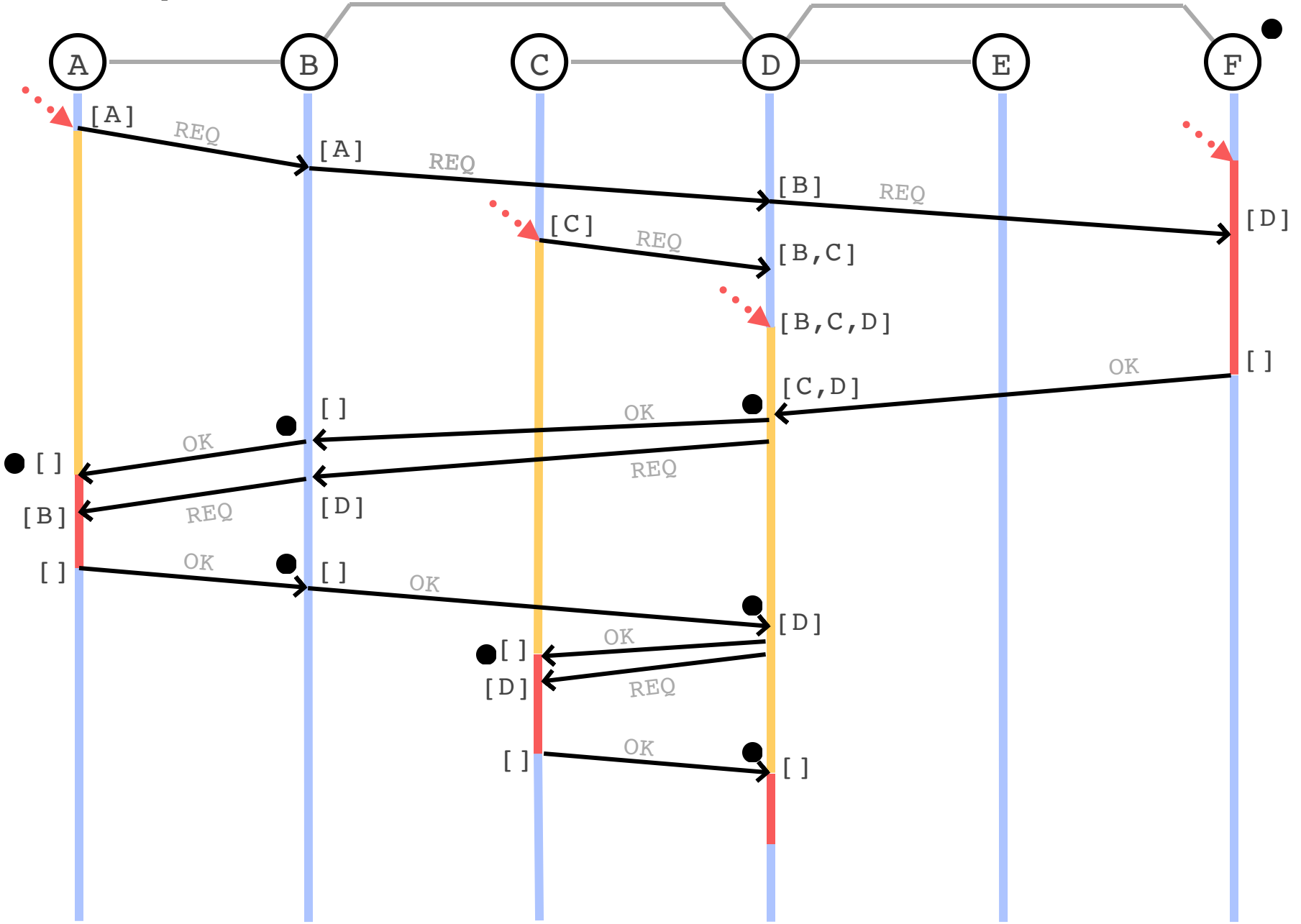
Exemple



Exemple



Exemple


A
B
C
D
E
F
[A]
REQ
[A]
REQ
[B]
REQ
[D]
[C]
REQ
[B,C]
[B,C,D]
[]
OK
[C,D]
[]
OK
OK
[]
REQ
[B]
REQ
[D]
[]
OK
[]
OK
[D]
[]
OK
[D]
REQ
[]
OK
[]

Algorithme de Raymond

Propriétés & Pseudocode

Algorithme par jeton unique

aka : Raymond, 1989

Propriétés

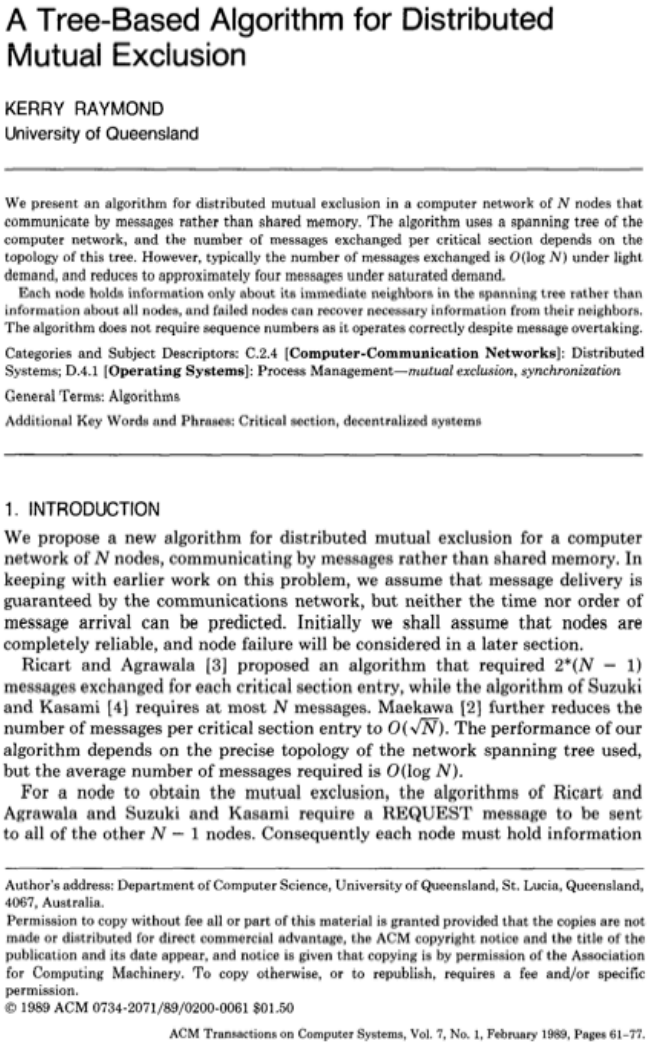
Correctness Jamais plus d'un processus ne sera en SC à un instant T.

Progrès Toute demande d'entrée en SC sera autorisée un jour.

Complexité de communication

~`2log(n)` messages par processus souhaitant entrer en SC, si l'arbre est équilibré *(en charge faible)*.

En charge forte, elle tend vers 4 messages.



A Tree-Based Algorithm for Distributed Mutual Exclusion

KERRY RAYMOND
University of Queensland

We present an algorithm for distributed mutual exclusion in a computer network of N nodes that communicate by messages rather than shared memory. The algorithm uses a spanning tree of the computer network, and the number of messages exchanged per critical section depends on the topology of this tree. However, typically the number of messages exchanged is $O(\log N)$ under light demand, and reduces to approximately four messages under saturated demand.

Each node holds information only about its immediate neighbors in the spanning tree rather than information about all nodes, and failed nodes can recover necessary information from their neighbors. The algorithm does not require sequence numbers as it operates correctly despite message overtaking.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.1 [**Operating Systems**]: Process Management—*mutual exclusion, synchronization*

General Terms: Algorithms

Additional Key Words and Phrases: Critical section, decentralized systems

1. INTRODUCTION

We propose a new algorithm for distributed mutual exclusion for a computer network of N nodes, communicating by messages rather than shared memory. In keeping with earlier work on this problem, we assume that message delivery is guaranteed by the communications network, but neither the time nor order of message arrival can be predicted. Initially we shall assume that nodes are completely reliable, and node failure will be considered in a later section.

Ricart and Agrawala [3] proposed an algorithm that required $2^*(N - 1)$ messages exchanged for each critical section entry, while the algorithm of Suzuki and Kasami [4] requires at most N messages. Maekawa [2] further reduces the number of messages per critical section entry to $O(\sqrt{N})$. The performance of our algorithm depends on the precise topology of the network spanning tree used, but the average number of messages required is $O(\log N)$.

For a node to obtain the mutual exclusion, the algorithms of Ricart and Agrawala and Suzuki and Kasami require a REQUEST message to be sent to all of the other $N - 1$ nodes. Consequently each node must hold information

Author's address: Department of Computer Science, University of Queensland, St. Lucia, Queensland, 4067, Australia.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1989 ACM 0734-2071/89/0200-0061 $01.50

Pseudocode

Variables

`self`	*entier, constant*	mon numéro de processus
`inSC`	*booléen, init false*	ssi je suis actuellement en SC
`hasRequested`	*booléen, init false*	ssi j'ai fait une demande de jeton
`parent`	*entier*	id du processus parent dans l'arbre
`queue`	*FIFO*	processus ayant fait une requête

Note : on appelle parent le processus dans la direction duquel aller pour se diriger vers le jeton.

Ainsi, si `parent` est `nil`, alors on peut dire qu'on a le jeton.

Pseudocode

Initialisation

Écouter infiniment les événements suivants :

- Demande d'entrée en SC
- Sortie de SC
- Réception de `REQ`
- Réception de `OK`

Pseudocode

Traitement : Demande de SC de la couche applicative

```
Envoi de REQ à self
```

Traitement : Réception de `REQ` de la part de `i`

```
Push i dans queue
passerJeton()
demanderJeton()
```

Pseudocode

Traitement : Réception de OK de la part de `i`

```
parent ← nil
passerJeton()
demanderJeton()
```

Traitement : Sortie de SC de la couche applicative

```
inSC ← false
passerJeton()
demanderJeton()
```

Pseudocode

Fonction `passerJeton`

Si `parent` n'est pas `nil`, sortir de cette fonction.

Si `inSC`, sortir de cette fonction.

Si `queue` est vide, sortir de cette fonction.

```
p ← queue.pop()
```

Si `p` vaut `self`

```
    inSC ← true
```

    Entrer en SC

Sinon

    Envoyer `OK` à `p`

```
    parent ← p
    hasRequested ← false
```

Pseudocode

Fonction `demanderJeton`

Si `parent` est `nil`, sortir de cette fonction.

Si `queue` est vide, sortir de cette fonction.

Si `hasRequested`, sortir de cette fonction.

Envoyer `REQ` à `parent`

`hasRequested ← true`

Algorithme de Raymond

Réflexion pour la prochaine fois

Un processus qui a fait une demande de SC peut être forcé de faire passer le jeton à sa tête de file, puis envoyer une requête pour qu'on le lui rende ensuite.

Ceci rend l'algorithme équitable, mais est sous-optimal en terme de communication.

Proposez une solution qui réduise le nombre de déplacements du jeton dans ce genre de situation, sans risquer la famine (qu'un demandeur ne puisse jamais entrer en SC).