Algorithmes à base de Jeton

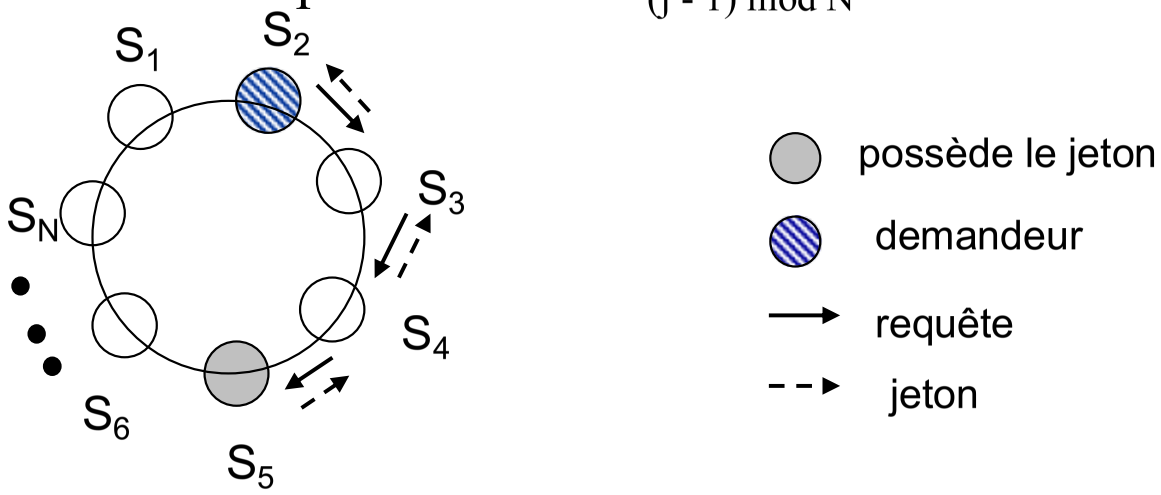
- **Anneau**
  - Martin
- **graphe complet**
  - Susuki/Kasami
- **Arbre**
  - Raymond (statique)
  - Naimi-Trehel (dynamique)

Algorithme à base de jeton

- **La permission pour rentrer en section critique est réalisée par la possession d'un jeton.**
  - L'unicité du jeton assure la sûreté.
- **Algorithmes doivent mettre en oeuvre la vivacité**
  - Déplacement du jeton
    - **Mouvement perpétuel du jeton**
      - Lorsque le jeton arrive sur un site, il passera au suivant si le site est dans l'état *not_requesting*; si le site est dans l'état r*equesting*, il passe à l'état *section_critique* et rentre en section critique.
      - Exemple: anneau de communication (garantie de la vivacité).
    - **Envoie de requêtes**
      - Anneau : **Martin**
      - Arborescence: **Naimi/Trehel**
      - Diffusion : **Suzuki/Kasami**

Algorithme de Martin (anneau)

- **Sites organisés en anneau logique statique**
  - Jeton circule dans le sens inverse des requêtes.
  - Un site demandeur entre en SC critique lorsqu'il possède le jeton
  - Quand S_i veut entrer en section critique, il envoie une requête à son successeur, $S_{(i+1) \bmod N}$, et attend le jeton. En recevant une requête de son prédécesseur, si S_j ne possède pas le jeton, il retransmet la requête à son successeur $S_{(j+1) \bmod N}$. Sinon, s'il le possède et ne l'utilise pas, il l'envoie à son prédécesseur $S_{(j-1) \bmod N}$.



Algorithme de Martin
Evaluation

- **Nombre de Messages par exécution de SC :**
  - Si K = nombre de sites entre S_i (site qui demande la SC) et le site S_p (site qui possède le jeton), alors :
    - Nb messages = 2*(K+1);
- **Avantages :**
  - Simplicité.
  - Pas de diffusion.
- **Inconvénients :**
  - Pas extensible.
  - un site qui n'est pas intéressé par la section critique est souvent sollicité à transmettre les requêtes et le jeton.

Algorithme de Raymonde arborescence statique

- **Les processus sont organisés en arbre ayant pour racine le site qui possède le jeton.**
  - Les arrêts sont orientés vers la racine
- **Les demandes du jeton**
  - sont propagées vers la racine
  - sont enregistrées dans une file locale sur chaque site du trajet

Algorithme de Raymonde arborescence statique

- **Un nœud ne communique qu'avait ses voisins**
- **Chaque nœud possède:**
  - variable *holder* qui pointe en direction du nœud racine
- ***file FIFO* pour sauvegarder les requêtes pendantes de ses voisins**
- **Arbre modifié (inversion du pointeur) à chaque transmission du jeton**

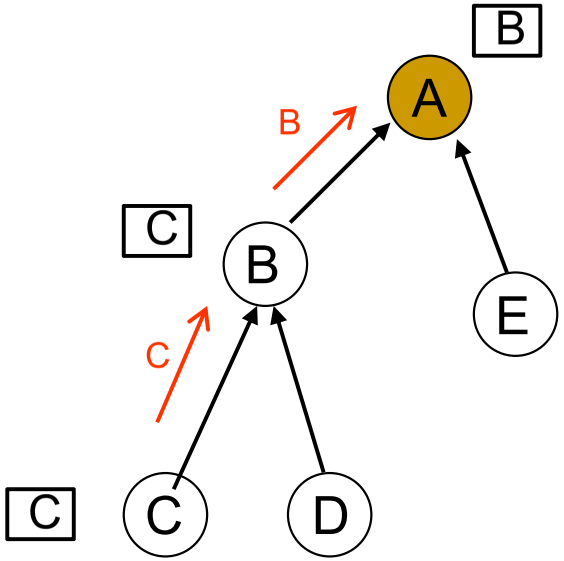
Algorithme de Raymonde arborescence statique

- **Algorithme**
  - Lorsqu'un nœud demande lui-même le jeton ou reçoit une requête pour le jeton de ses voisins, le noeud ajoute la requête dans sa file locale.
    - Si la file était vide il renvoie un requête à son *holder*
  - En recevant une requête, le nœud qui possède le jeton le libère lorsqu'il ne l'utilise plus.
    - A chaque libération du jeton un nœud inverse la direction de *holder*

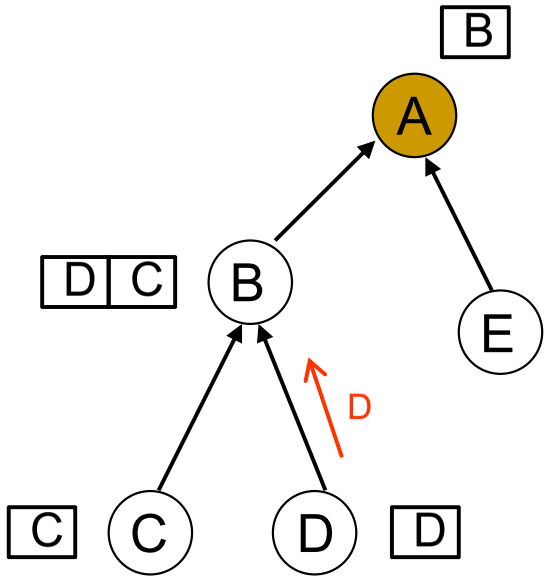
Algorithme de Raymonde arborescence statique

- **Algorithme (cont.)**
  - Lorsqu'un nœud reçoit le jeton, il enlève le premier élément *first* de sa file.
    - Si *first* est le propre nœud, il rentre en section critique
    - Sinon le jeton est renvoyé à *first*
  - Si la file n'est pas vide, une requête pour le jeton est renvoyé au voisin.

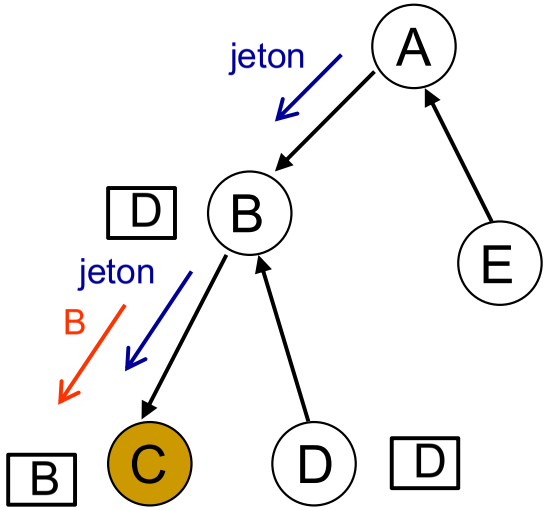
Algorithme de Raymonde arborescence statique



C demande le jeton



D demande le jeton



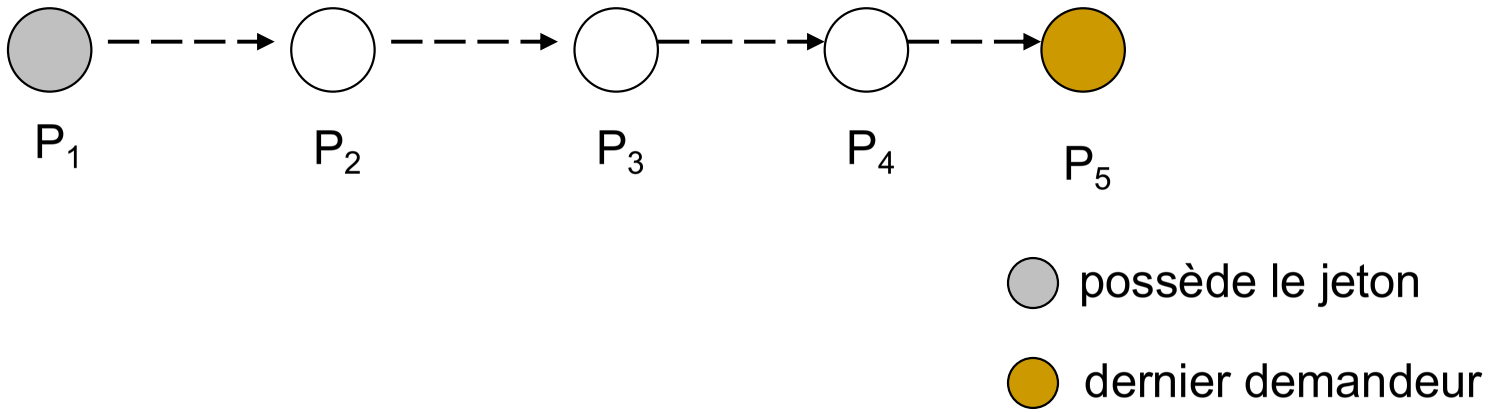
A libère le jeton

Algorithme de Naimi/Trehel arborescence dynamique

- **Deux structures de données:**
  - **File de requêtes : "*next*"**
  - **Arbre de chemins vers le dernier demandeur : "*father*"**

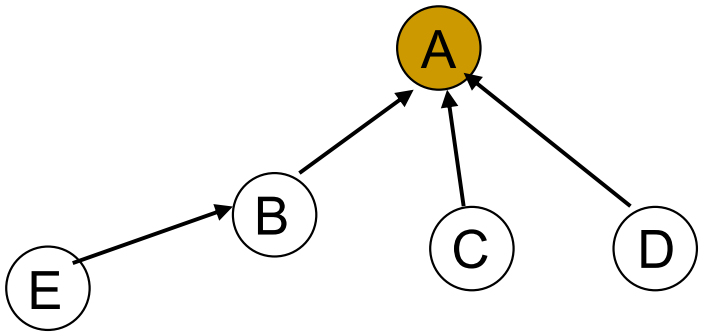
Algorithme de Naimi/Trehel

- File de requêtes : "*next*"
  - Processus en tête de la file possède le jeton.
  - Le processus à la fin de la file est le dernier processus qui a fait une requête pour entrer en section critique.
  - Une nouvelle requête est toujours placée en fin de la file.

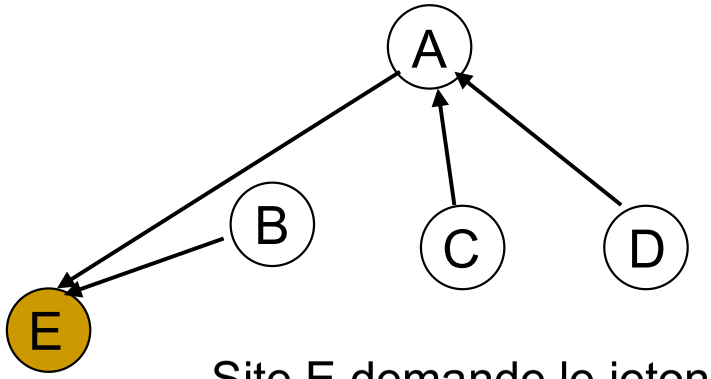


Algorithme de Naimi/Trehel

- **Arbre de chemins vers le dernier demandeur : "*father*"**
  - Racine de l'arbre : dernier demandeur (dernier élément de la file des "*next*").
  - Une nouvelle requête est transmise à travers un chemin de pointeurs "*father*" jusqu'à la racine de l'arbre (*father = nil*).
    - Reconfiguration dynamique de l'arbre. Le nouveau demandeur devient la nouvelle racine de l'arbre.
    - Les sites dans le chemin compris entre la nouvelle et l'ancienne racine changent leur pointeur "*father*" vers la nouvelle racine.



Site A dernier demandeur



Site E demande le jeton

Algorithme de Naimi/Trehel

Local Variables:
Token : boolean;
requesting; boolean
next, father: 1,.. N U {nil}

Initialisation de S_i:
```
father = S1; next = nil;
requesting = false;
Token = (father == Si);
if (father == Si)
   father = nil;
```

Request_CS (S_i):
```
requesting = true;
if (father <> nil) {
  send (Request, Si) to father;
  father = nil;
}
attendre ( Token == true);
```

Release_CS (S_i):
```
requesting = false;
if (next <> nil) {
   send (Token) to next;
   Token = false;
    next = nil;
}
```

Algorithme de Naimi/Trehel (cont)

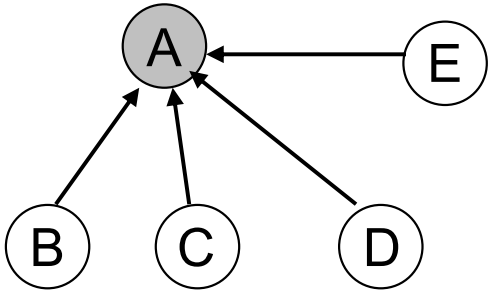
Receive_Request_CS(S_j):

```
if (father == nil) {
    if (requesting)
        next = Sj;
   else { token = false;
            send (Token) to Sj;
}
else
     send (Request, Sj) to father;
father = Sj;
```

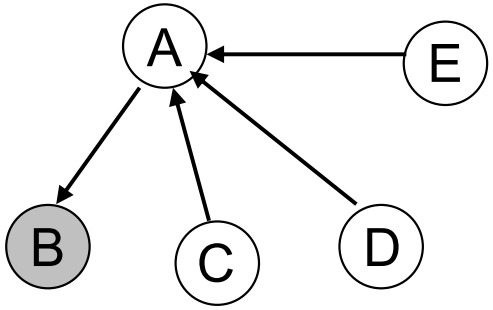
Receive_Token (S_j):

```
Token = true;
```

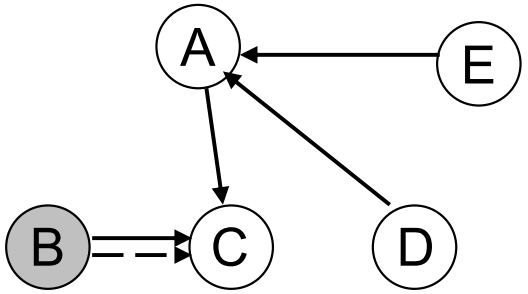
Algorithme de Naimi/Trehel (Exemple)



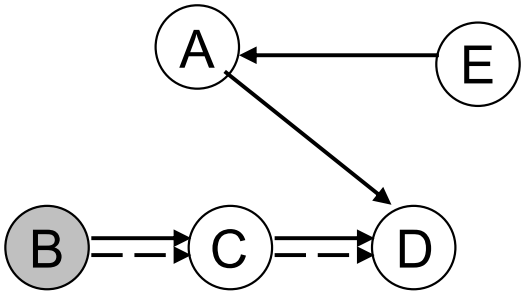
Site A possède le jeton



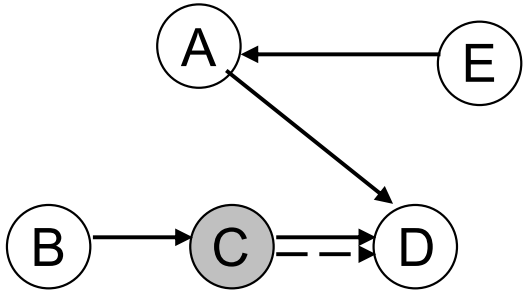
Site B fait une requête
B entre en SC



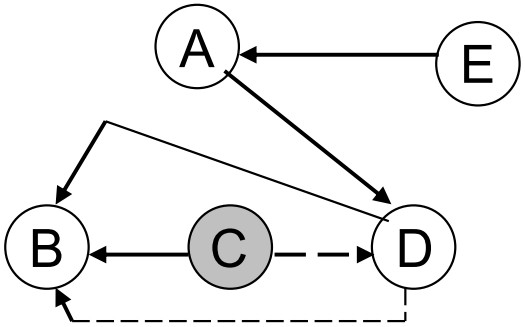
Site C fait une requête



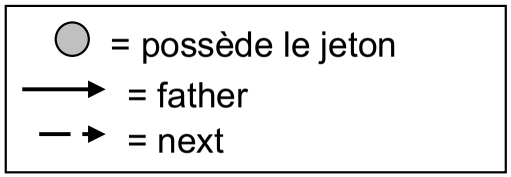
Site D fait une requête



Site B sort de la SC
C entre en SC



Site B fait une requête



Algorithmes de Raymonde et Naimi/Trehel (Evaluation)

- **Nombre de Messages par exécution de SC :**
  - Entre 0 et N par demande
  - Moyenne : O(log N).

- **Avantages :**
  - Extensibilité : O(log N).
  - Naimi-Tréhel
    - un site qui n'est pas intéressé par la section critique ne sera plus sollicité après quelques transferts de requêtes "adaptativité".

Algorithme de Susuki/Kasami (diffusion)

- Pour entrer en SC, un processus **diffuse une demande de jeton à tous les autres processus**.
  - Si le processus qui possède le jeton n'est pas en SC, il renvoie immédiatement le jeton au processus demandeur. Sinon, il attend la sortie de la SC et envoie le jeton au premier processus dont la requête n'a pas été satisfaite**.**
    - Les requêtes pendantes sont transmises dans le message du jeton en respectant l'ordre FIFO.
- Chaque processus gère un compteur des requêtes qu'il a effectuées et une table des requêtes effectuées par les autres processus.
- Le jeton est un message particulier, unique, contenant la table des requêtes satisfaites et un file d'attente de requêtes pendantes.

Algorithme de Susuki/Kasami

- **Type de Message:**
  - **REQUEST (S_j,k):**
    - k = (1,2,… N). Indique que site S_j est en train de faire sa *kème* demande d'entrée en section critique.
  - **TOKEN (Q, LN)**
    - **Q** : une file d'attente de demandes pour entrer en section critique des différents sites.
    - **LN** : où LN[j] est le numéro de la dernière demande d'entrée en section critique du site S_j qui a été satisfaite.

Algorithme de Susuki/Kasami

- **Variables:**
  - **$Etat_i$** : *requesting*, *not_requesting*, *critical_section.*
  - **$Token_i$** : indique la présence du jeton sur le site S_i.
  - **RN_i**: vecteur de N positions :
    - RN_i [j] est le numéro de la dernière requête reçue de la part du site S_j.
    - RN_i [i] correspond au nombre de requêtes faites par le site S_i.
  - **LN_i**: vecteur de N positions des requêtes satisfaites

Algorithme de Susuki/Kasami

Initialisation variables locales (S_i):

```
Token = (Si ==S1);
Etat = not_requesting;
RN [j] = 0, j = 1,2, …N;
LN [j] = 0, j=1,2,…N;
Q=Ø;
```

Request_CS (i):

```
Etat=requesting;
if (Token == false) {
   RN[i] = RN[i] +1;
   diffuser REQUEST(Si,RN[I]);
   attendre (Token == true)
}
Etat = critical_section;
```

Release_CS (i):

```
LN[i] = RN[i];
for (site =1; site <=n;site ++) {
   if ((site!=i) && (site not in Q) &&
       (RN[site] > LN[site] ))
          ajouter site à la fin de Q;
 }

 if (Q != Ø) {
    Token = false;
    site = extraire (Q); /*premier de la file
    send TOKEN (Q,LN) to site ;
}

Etat = not_requesting;
```

Algorithme de Susuki/Kasami (cont)

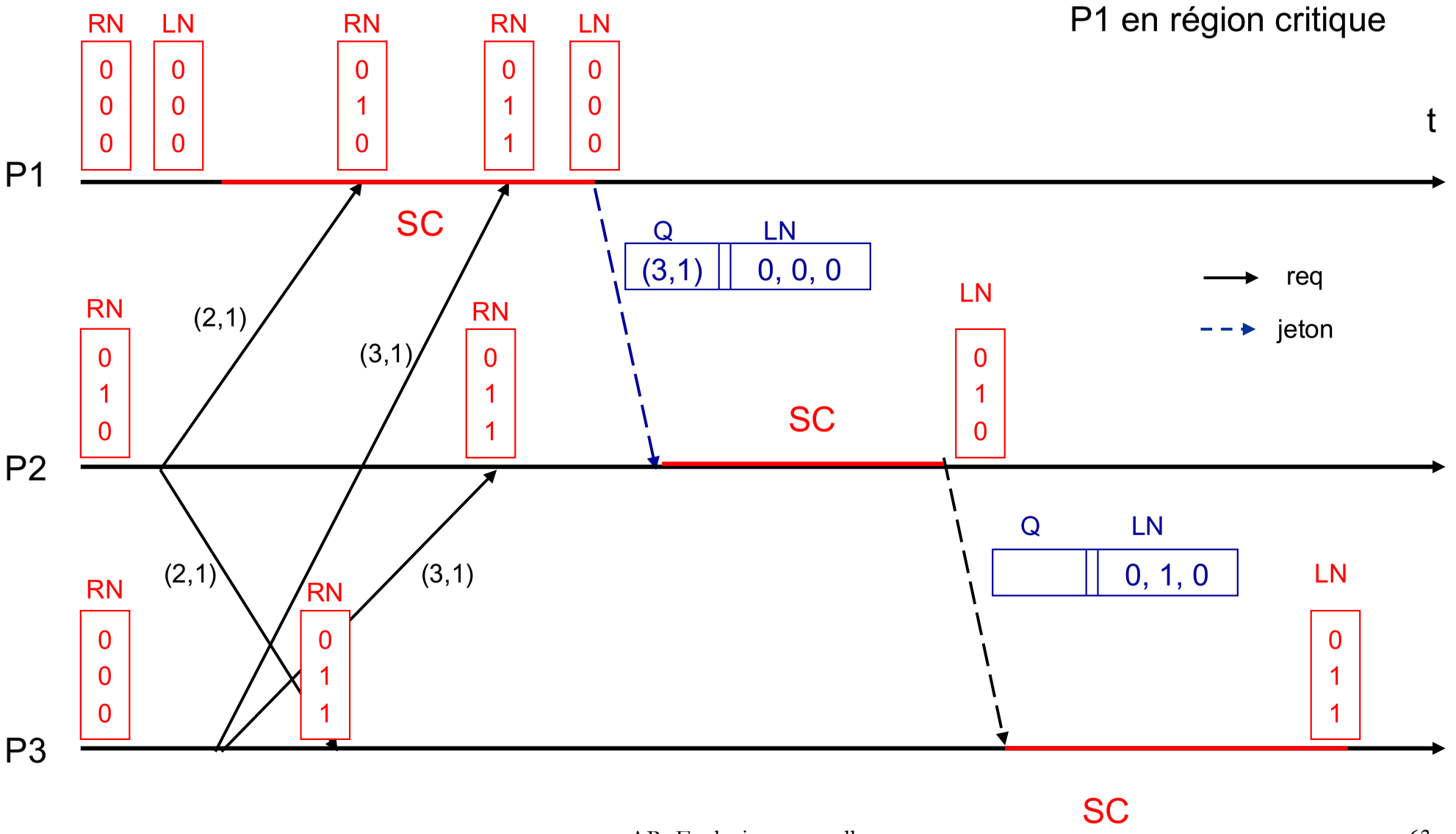
Receive_Request_CS(S_j, REQUEST (j,k)):

```
RN[j] = max (RN[j],k);
if ((Token = true) && (Etat == not_requesting) && (RN[j] > LN [j] )) {
      Token = false;
      send TOKEN (Q,LN) to Sj;
 }
```

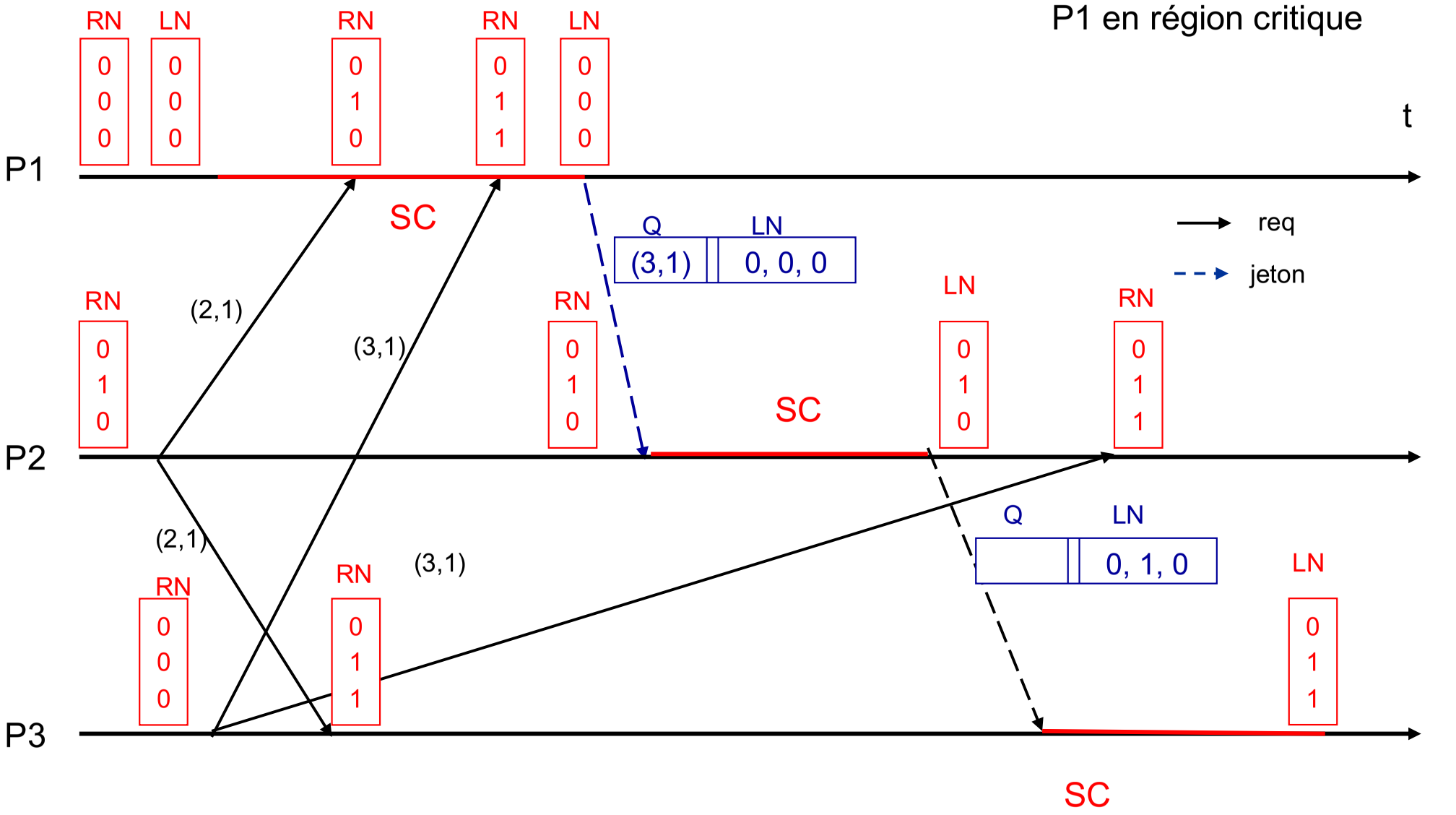
Recieve_Token (TOKEN (Q,LN)):

```
Token = true;
LN = TOKEN.LN ;
Q = TOKEN.Q;
```

Algorithme de Susuki/Kasami
Exemple 1



Algorithme de Susuki/Kasami
Exemple 2



Algorithme de Susuki/Kasami (Evaluation)

- **Nombre de Messages par exécution de SC:**
  - N, si le processus n'a pas le jeton.
  - 0, si le processus a le jeton
- **Vivacité**
  - Garantie par l'ordre FIFO de la file Q
- **Inconvénient :**
  - Pas extensible

Bibliographie

- Lamport, L. *Time, clocks and the ordering of events in a desitributed system,* Communications of the ACM, vol. 21, no. 7, july 1978, pages 558-565.
- Ricart, G. and Agrawala, A. *An optimal algorithm for mutual exclusion en computer networks*, Communications of the ACM, vol. 24, no. 1, jan 1981, pages 9-17.
- Suzuki, I. and Kasami, T. *A distributed mutual exclusion algorithm,* ACM Transactions on Computer Systems, vol. 3, no. 4, nov. 1985, pages 344-349.
- Naimi, M. and Trehel, M. *A Log (N) distributed mutual algorithm based on the Path Reversal*, Journal of Parallel and Distributed Computing vol. 34 no. 1, avril 1996, pages 1-13.
- Raynal, M. *Synchronisation et Etat Global dans les Systèmes Répartis,* 1992.
- K. Raymond 1989. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems 7, 61–77.*
- MARTIN, A.J. Distributed Mutual Exclusion on a Ring of Processes, *Science of Computer Programming*, vol5. pp-265-276, Feb. 1985.
- MAEKAWA, *M. A sqrt(n) algorithm for mutual exclusion in descentralized systems*, ACM Transaction on Computer Systems, vol. 3, no.2, mai 1985, pages 145-159
- ALBERT,A. and SANDLER, R, *An Introduction to Finite Projective Planes*, Holt, Rinehart and Winston, NY, 1968.