Algorithmique Répartie

La Concurrence

Université François Rabelais de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Master 1 Informatique

Option Systèmes d'Information et Aide à la Décision

Mohamed Taghelit
taghelit@univ-tours.fr

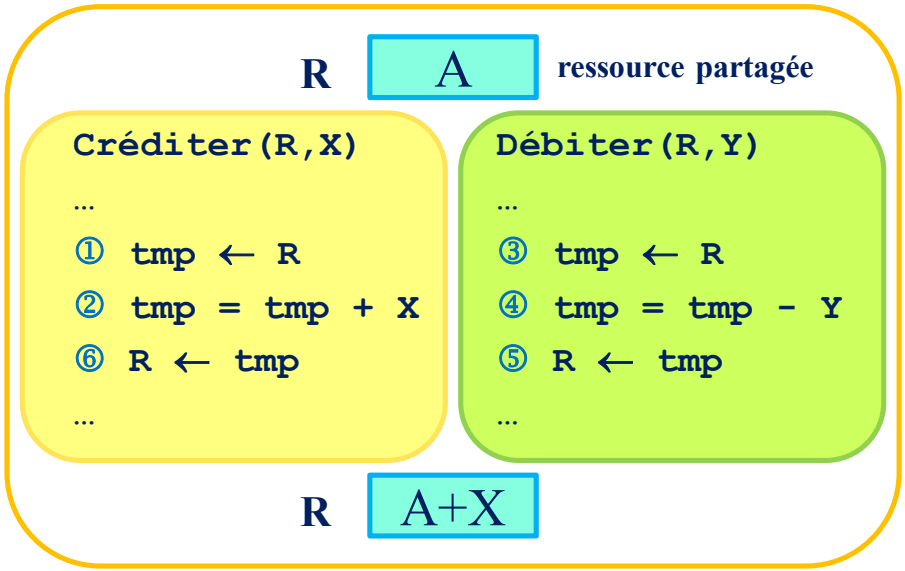
La Concurrence

Problématique

- l'accès concurrentiel à des ressources partagées peut provoquer l'incohérence de ces dernières,

- gérer la concurrence revient à contrôler que des accès concurrents s'exécutent de manière cohérente,
  - garantir que pour chaque ressource au plus une section de code qui la manipule soit en cours d'exécution (une telle section est appelée section critique),
  - séquentialisation des transactions (BD),

- mise en œuvre de l'exclusion mutuelle entre sections critiques
  - lorsqu'un processus applicatif voudra exécuter une section critique, son programme se présentera comme ci-contre :



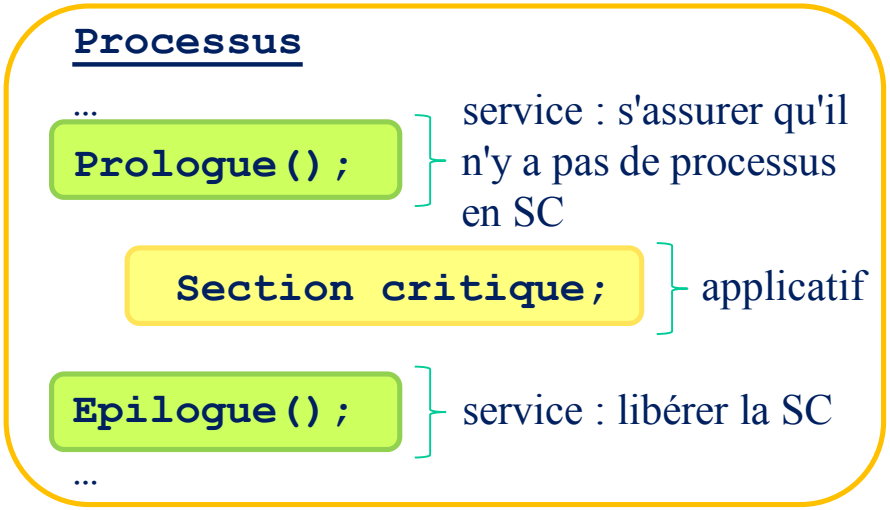

```
Processus
...
Prologue();

    Section critique;

Epilogue();
...
```

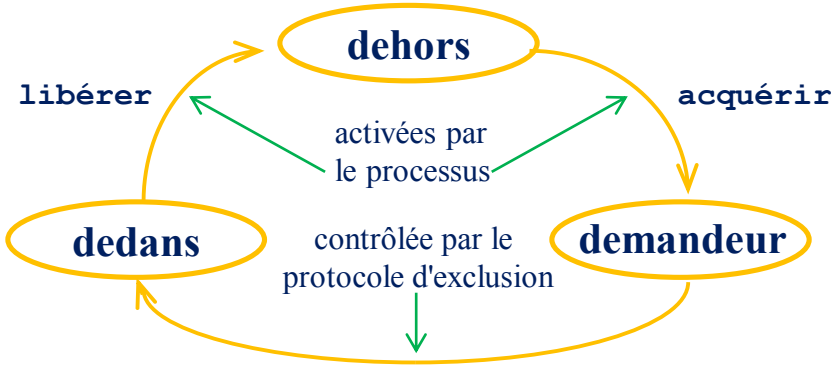
La Concurrence

`Prologue` et **`Epilogue`** sont des primitives du service qui devront garantir les deux propriétés caractéristiques de l'exclusion mutuelle :

- A tout instant, au plus un processus est en cours d'exécution d'une section critique. Ce type de propriété est appelée **propriété de sûreté** et traduit le fait que *"quelque chose de mal n'arrivera jamais"*.

- Tout processus demandant à exécuter une section critique (par appel à **`Prologue`**), pourra l'exécuter (par retour de **`Prologue`**) au bout d'un temps fini. Ce type de propriété est appelée **propriété de vivacité** et traduit que *"quelque chose de bien finira par arriver"*.





États d'un processus

Mise en Œuvre de l'Exclusion Mutuelle

Nombreux algorithmes proposés dans la littérature

- présentation de trois algorithmes,
- algorithmes basés sur les horloges logiques,
- chaque nouvel algorithme se construit par la modification conceptuelle du précédent,
- chaque nouvel algorithme diminue la complexité (en nombre de messages échangés par exécution d'une section critique)

Solutions basées sur la circulation de jeton

- solutions non satisfaisantes,
- la mesure de la complexité se base sur le nombre de messages échangés pour une section critique,
- même si les sites ne sont pas désireux d'exécuter une section critique, le jeton doit circuler,

  $\Rightarrow$ le nombre de messages échangés peut s'accroître indéfiniment sans une seule entrée en section critique.

Solutions Logicielles avec Partage de Variables

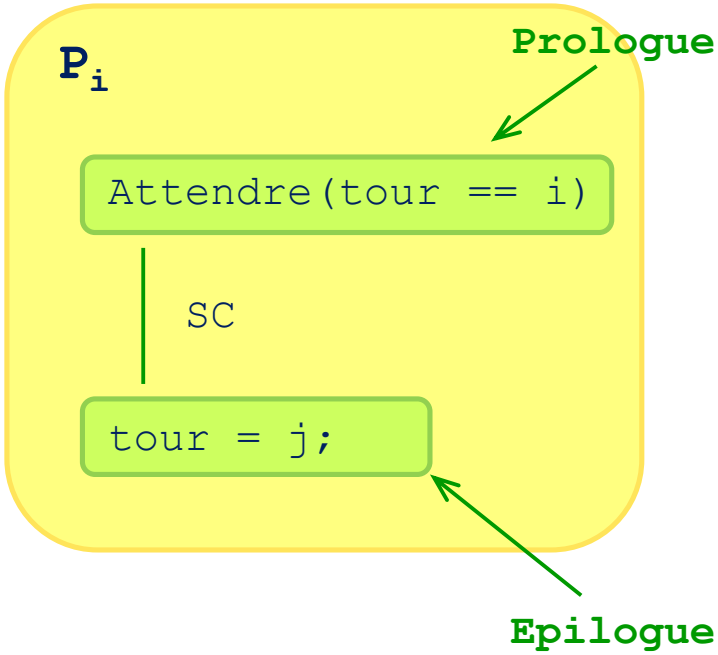
Cas de deux processus P_0 et P_1 :

Algorithme 1 :

P_0 et P_1 partagent une variable commune entière **tour** initialisée à **0** (ou **1**).

- La propriété de sûreté est vérifiée (un seul processus à la fois se trouve en SC).

- Problème :
  si **tour = 0** et P_1 veut entrer en SC, il ne le pourra pas même si P_0 n'en veut pas (d'entrer en SC) ou est hors SC.
  L'algorithme ne se rappelle pas l'état de chacun des processus mais se rappelle uniquement lequel est autorisé à entrer en SC.



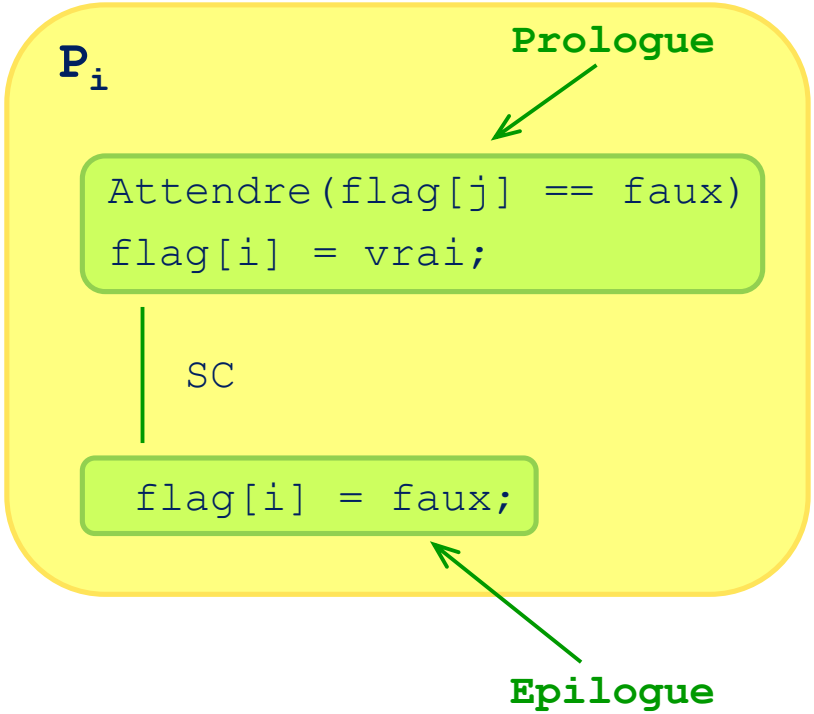
Solutions Logicielles avec Partage de Variables

Cas de deux processus **P_0** et **P_1** :

Algorithme 2 :

P_0 et **P_1** partagent le vecteur commun de booléens **`flag[0..1]`** initialisé à **`faux`**.
`flag[i] = vrai` lorsque **P_i** est en SC.

- La propriété de sûreté n'est pas vérifiée (les deux processus peuvent être simultanément en SC).
- Problème :
  P_i prend une décision concernant l'état de **P_j** avant que ce dernier n'ait eu l'opportunité de changer l'état de la variable **`flag[j]`**.



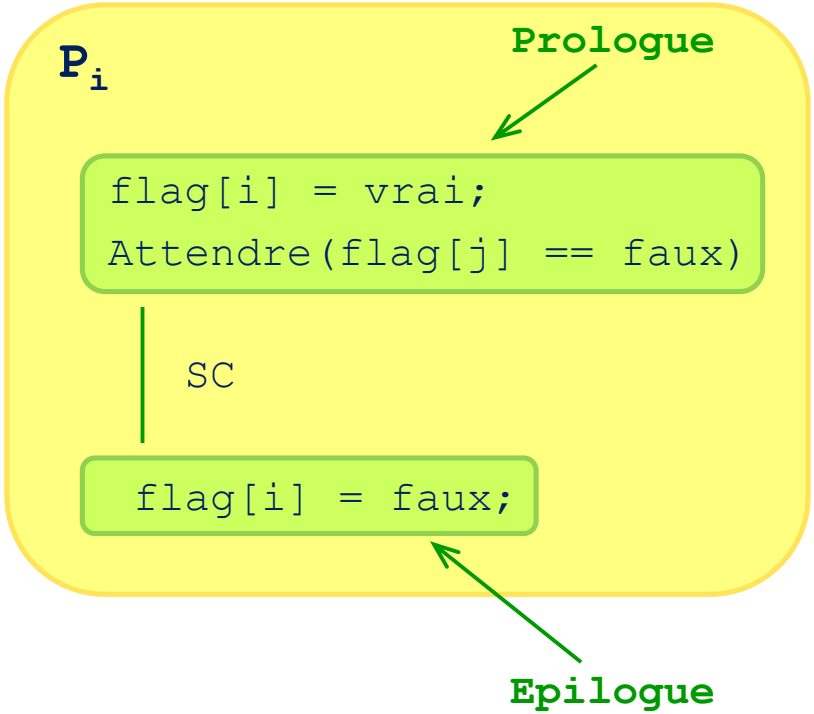
Solutions Logicielles avec Partage de Variables

Cas de deux processus P_0 et P_1 :

Algorithme 3 :

P_0 et P_1 partagent le vecteur commun de booléens `flag[0..1]` initialisé à **`faux`**.
`flag[i] = vrai` indiquera seulement que P_i veut entrer en SC.

- La propriété de sûreté est vérifiée (un seul processus à la fois se trouve en SC).
- La propriété de vivacité n'est pas vérifiée (P_0 et P_1 peuvent attendre indéfiniment).

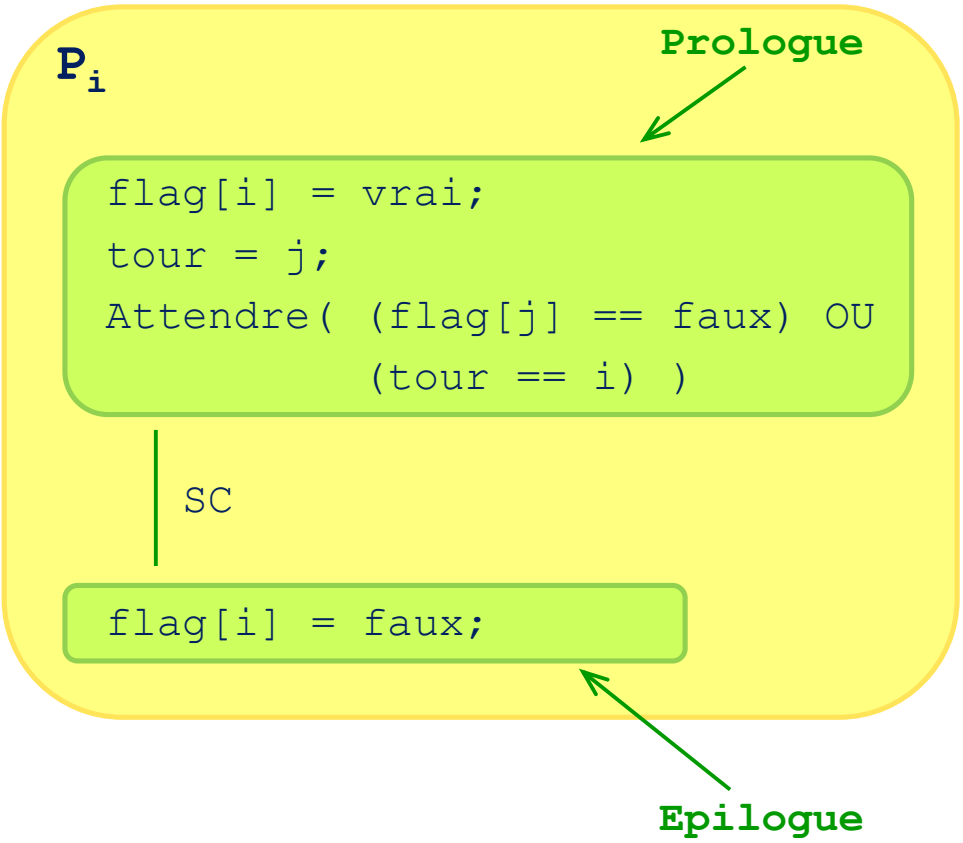


Solutions Logicielles avec Partage de Variables

Cas de deux processus P_0 et P_1 :

Algorithme 4 : Peterson (1981)
(Combinaison des algorithmes 1 et 3)

P_0 et P_1 partagent deux variables :
- **flag[0..1]** : vecteur de booléens, initialisé à **faux**.
- **tour : 0..1**, variable entière initialisée à **0** (ou **1**).



- La propriété de sûreté est vérifiée (un seul processus à la fois se trouve en SC).
- La propriété de vivacité est vérifiée.
- Généralisation à **n** processus ?

Algorithme de L. Lamport

Principe de l'algorithme de L. Lamport (1978)

- repose sur le mécanisme des horloges logiques (comme les deux suivants),
- la mise à jour de l'horloge logique sera implicite (pour éviter l'écriture d'un code répétitif),
  - la mise à jour sera faite au niveau de la couche réseau à l'émission et à la réception,
  - la couche service fournira une valeur d'horloge dans tous les messages de l'algorithme.
- le protocole échange trois types de messages :
  - des requêtes (**`req`**) d'exécution de section critique diffusées à tous les autres sites par le site demandeur,
  - des acquittements (**`acq`**) de requête pour indiquer que le site a "enregistré" la requête,
  - des libérations (**`lib`**) diffusées à tous les autres site par le site qui a terminé l'exécution d'une section critique.
- une section critique s'accompagne de **`3.(n-1)`** messages où **`n`** est le nombre de sites :
  - **`n-1`** requêtes,
  - **`n-1`** acquittements,
  - **`n-1`** libérations.

Algorithme de L. Lamport

Principe de l'algorithme

- Chaque site maintient un tableau $\mathbf{Mess_i[n]}$ de messages indicé par les identités des sites,
- Chaque cellule du tableau contient le type du message et son heure,
  - chaque message est estampillé par l'heure logique,
  - date de naissance d'un message → (horloge, identité du site)

$$\mathbf{(h_i,\ i) < (h_j,\ j) \Leftrightarrow (h_i < h_j)\ OU\ (\ (h_i = h_j)\ ET\ (i < j)\ )}$$

évite le cas d'égalité des âges de deux messages

- Condition d'entrée en section critique :

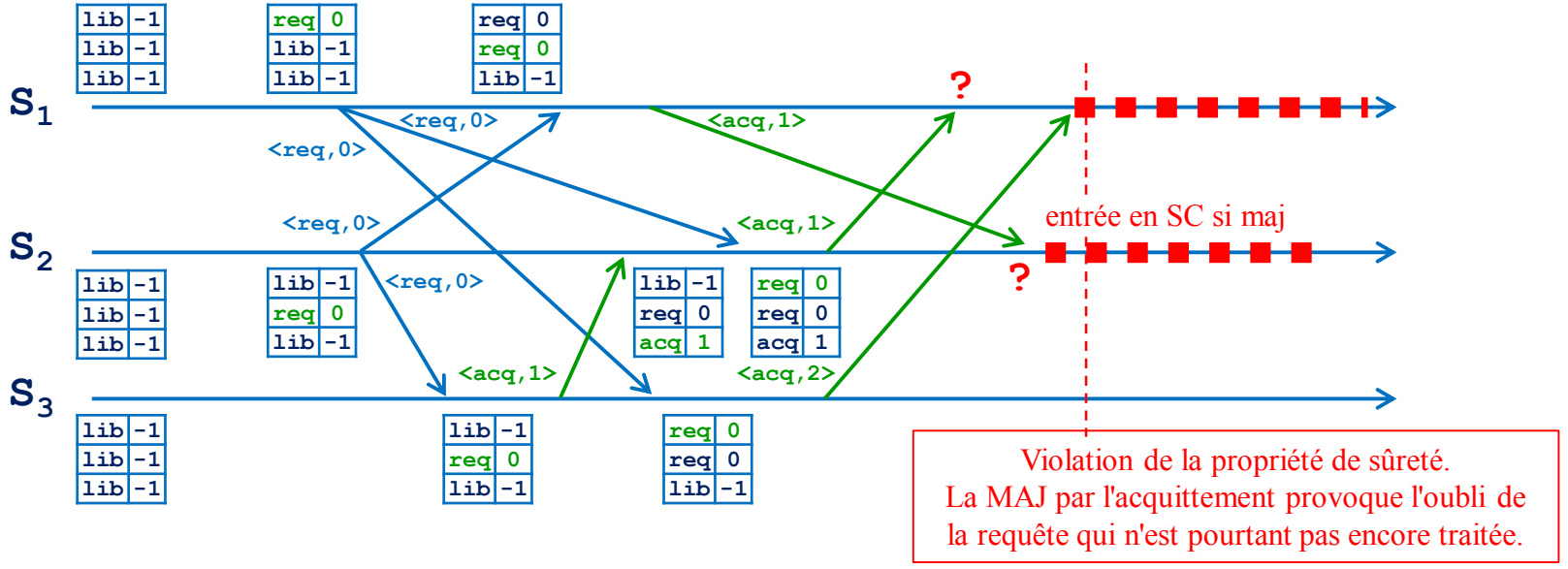
$$\mathbf{\forall\ j \neq i,\ (Mess_i[i].heure,\ i) < (Mess_i[j].heure,\ j)}$$

avoir dans sa cellule le message (requête) le plus âgé du tableau

Algorithme de L. Lamport

Principe de l'algorithme

- Lorsqu'un site désire exécuter sa section critique, il met à jour sa propre cellule **`Mess`$_i$`[i]`** avec sa requête,
- Toutes les cellules du tableau **`Mess`$_i$`[n]`** sont initialisées avec un message de libération daté de l'heure -1
  - une requête initiale doit donner lieu à des acquittements pour exécuter la section critique.
- Règles de mise à jour des autres cellules du tableau
  - tout message reçu de **`j`** doit-il provoquer la mise à jour de la cellule **`j`** du tableau (**`Mess`$_i$`[j]`**) ?

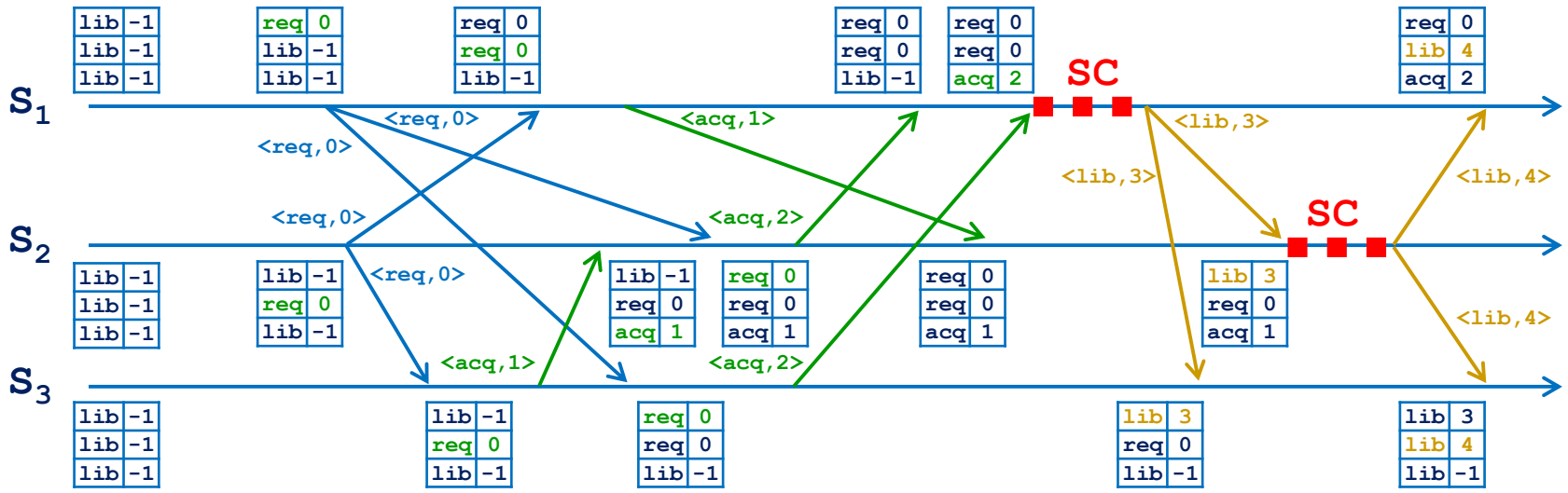


Algorithme de L. Lamport

Principe de l'algorithme

- Règle de mise à jour du tableau lors d'une réception

  Tout message arrivant est enregistré dans le tableau à l'exception d'un acquittement qui remplacerait une requête.



Algorithme de L. Lamport

Variables du site **i** :

- **h_i** : variable contenant la valeur de l'horloge locale du site **i**. Elle est initialisée à **0**.
- **$Mess_i$[1..n]** : tableau des messages. Chaque cellule est composée des deux champs **<type, heure>**.
  - **type** prend ses valeurs parmi **{req, acq, lib}** et est initialisé à **lib**.
  - **heure** est initialisé à **-1**.

Algorithme du site **i** :

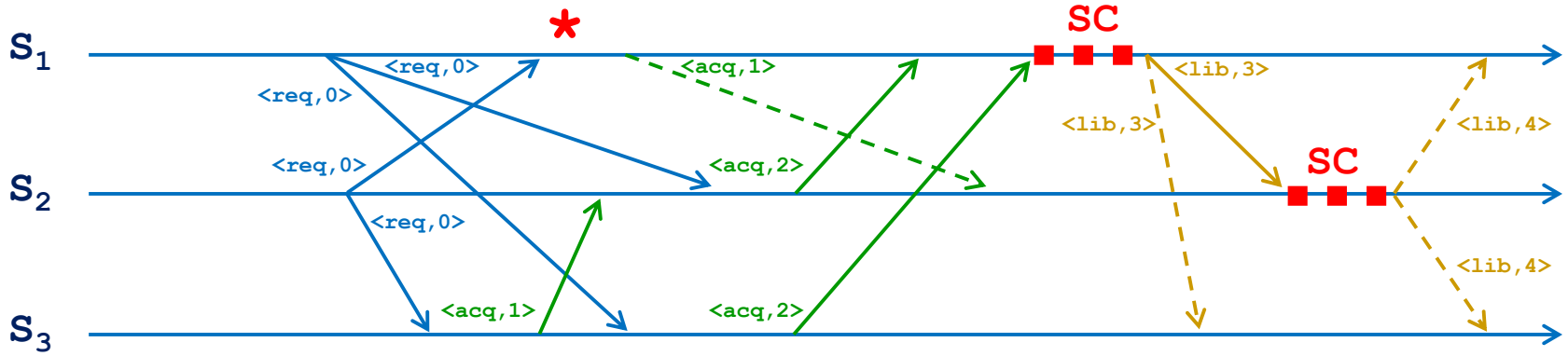
```
Epilogue(){
   Diffuser(lib, h_i);
}
```

```
Prologue(){
   Mess_i[i].heure = h_i;
   Mess_i[i].type = req;
   Diffuser(req, h_i);
   Attendre( ∀ j ≠ i, (Mess_i[i].heure, i) < (Mess_i[j].heure, j) )
}
```

```
sur_réception_de(j,(tp,h)){
   Si ( tp == req ) Alors
   |      envoyer_à(j,(acq, h_i));
   Fsi
   Si ( (tp != acq) OU (Mess_i[j].type != req) ) Alors
   |      Mess_i[j].heure = h;
   |      Mess_i[j].type = tp;
   Fsi
}
```

Algorithme de Ricart et Agrawala

Principe de l'algorithme de Ricart et Agrawala (1981)



- Fusionner messages "acquittement" et "libération" en un seul message "acquittement"
- Sémantique du message d'acquittement du site **`j`** à une requête du site **`i`** :
  - Lamport → **`j`** informe **`i`** qu'il a enregistré sa requête
  - Ricart et Agrawala → **`j`** autorise **`i`** à entrer en section critique

- Condition d'entrée en section critique

  Réception d'une autorisation de chacun des autres sites (nombre acquittements reçus = **`n - 1`**)

- une section critique s'accompagne de **`2.(n-1)`** messages où **`n`** est le nombre de sites :
  - **`n-1`** requêtes,
  - **`n-1`** acquittements,

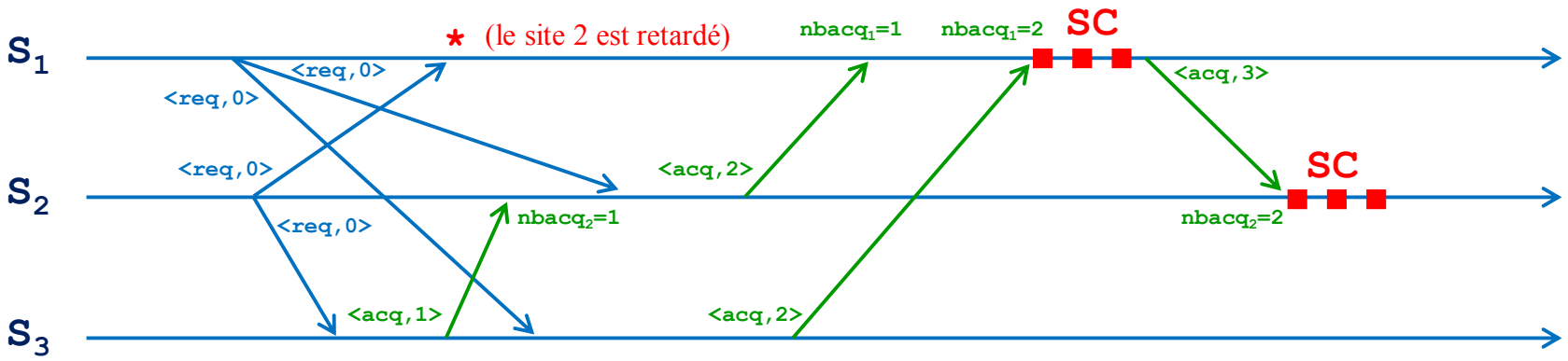
Algorithme de Ricart et Agrawala

Principe de l'algorithme

- Politique du site **i** lorsqu'il reçoit une requête du site **j** :
  - si l'application de **i** n'est pas en attente ou en cours d'exécution d'une SC → envoi immédiat de l'acquittement à **j**,
  - sinon, **i** compare l'âge de sa requête avec celui de la requête de **j** :
    - si la requête de **j** est plus âgée → envoi immédiat de l'acquittement à **j**
    - sinon, **i** diffère l'envoi de l'acquittement à **j** jusqu'à sa sortie de SC

      ⇒ nécessité de mémoriser les sites pour lesquels l'envoi de l'acquittement a été différé.

- Scénario d'exécution



Algorithme de Ricart et Agrawala

Variables du site **i** :

- h_i : variable contenant la valeur de l'horloge locale du site **i**. Elle est initialisée à **0**.
- $état_i$: état du site. Cette variable peut prendre l'une des deux valeurs **{repos, en_cours}**. Initialisée à **repos**.
- $hreq_i$: heure de la requête courante de **i**.
- $nbacq_i$: compteur des acquittements reçus.
- $Retardé_i[1..n]$: tableau de booléens. $Retardé_i[j]$ = **Vrai** lorsque **i** retarde l'acquittement d'une requête de **j**.
- $temp_i$: variable temporaire.

Algorithme du site **i** :

La réception d'un acquittement incrémente le compteur des acquittements reçus.

```
sur_réception_de(j,(acq, h)){
   nbacq_i++;
}
```

Lorsqu'un site désire entrer en SC, il diffuse une requête puis se met en attente de tous les acquittements.

```
Prologue(){
   hreq_i = h_i;
   nbacq_i = 0;
   Pour tout temp_i de 1 à n Faire
   |      Retardé_i[temp_i] = Faux;
   Finpour
   état_i = en_cours;
   Diffuser(req, h_i);
   Attendre( nbacq_i == (n-1) );
}
```

Algorithme de Ricart et Agrawala

La réception d'une requête suit la politique décrite précédemment (diapo 14).

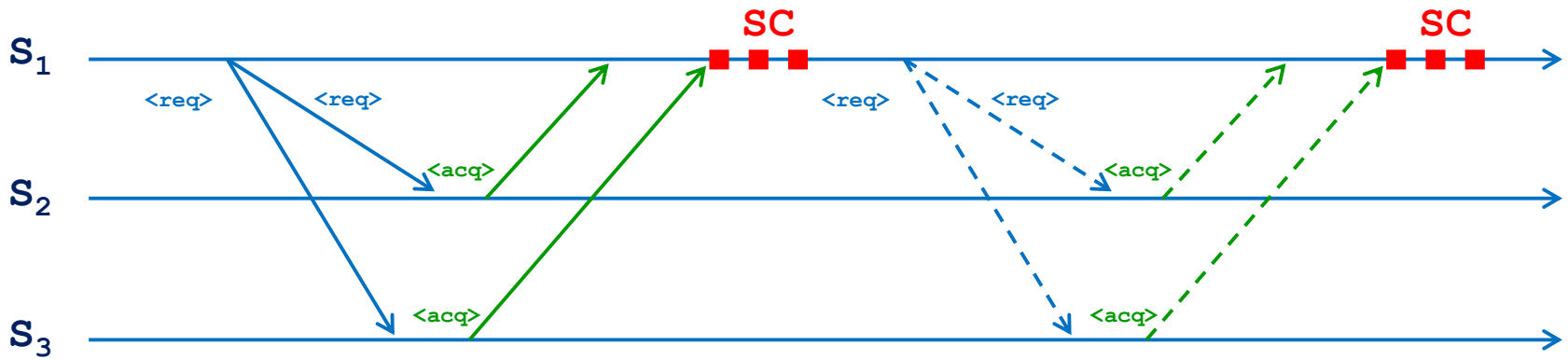
```
sur_réception_de(j,(req, h)){
  Si ( (état_i == en_cours) ET ((hreq_i,i) < (h,j)) ) Alors
  |     Retardé_i[j] = Vrai;
  Sinon
  |     envoyer_à(j, (acq, h_i));
  Fsi
}
```

A la sortie de la SC, le site envoie des acquittements aux sites qu'il a retardés.

```
Epilogue(){
  Pour tout temp_i de 1 à n Faire
  | Si ( Retardé_i[temp_i] ) Alors
  | |    envoyer_à(temp_i, (acq, h_i));
  | Fsi
  Finpour
  état_i = repos;
}
```

Algorithme de Carvalho et Roucairol

Principe de l'algorithme de Carvalho et Roucairol (1983)



- Sémantique du message d'acquittement du site **j** à une requête du site **i** :
  - Ricart et Agrawala → **j** autorise **i** à entrer en section critique pour la SC courante
  - Carvalho et Roucairol → **j** autorise **i** à entrer en section critique pour la SC courante mais aussi pour les SC suivantes, jusqu'à révocation

    L'acquittement s'interprète comme l'envoi d'une permission partagée entre **i** et **j**.

    Si **j** veut par la suite pénétrer en SC, il doit réclamer la permission qu'il a concédé à **i**.

- Condition d'entrée en section critique

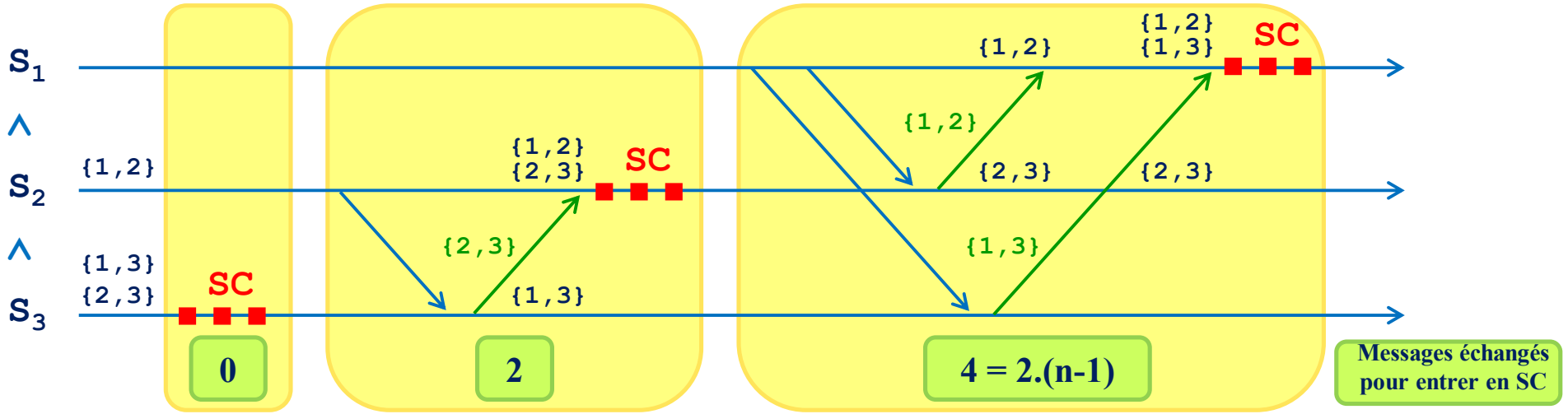
  La possession des permissions partagées avec chacun des autres sites

Les permissions sont similaires aux jetons multiples associés à l'algorithme du rendez-vous.

Algorithme de Carvalho et Roucairol

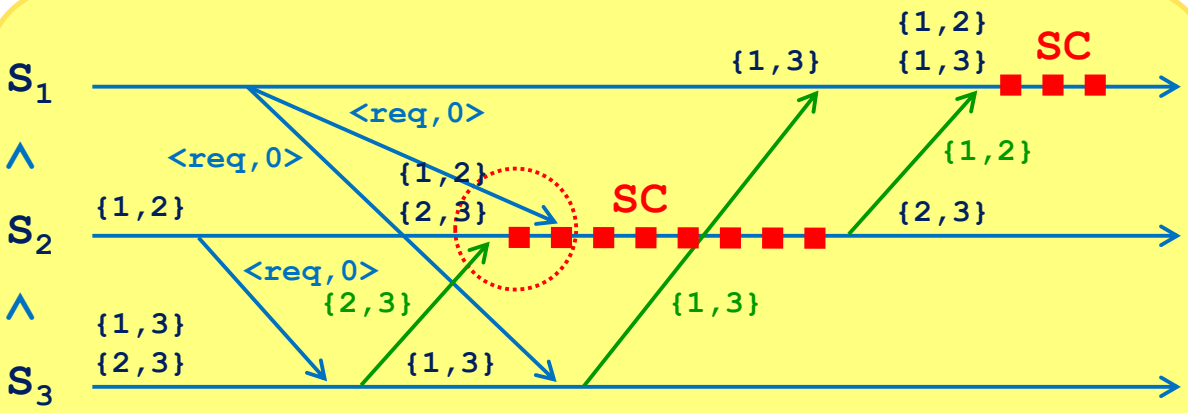
Principe de l'algorithme

- Répartition initiale des jetons entre les sites
  - pour **n** sites, il y aura **n.(n-1)/2** jetons,
  - idée simple de répartition : le jeton du couple **{i, j}** est initialement possédé par le site d'identité **max(i, j)**.
- Modification importante du prologue
  - seules les permissions manquantes sont réclamées,
  - un site peut entrer immédiatement en section critique sans échanger un seul message.
- Une section critique s'accompagne de 0 à 2.(n-1) messages où n est le nombre de sites :



Algorithme de Carvalho et Roucairol

Principe de l'algorithme : Scénarios d'exécution

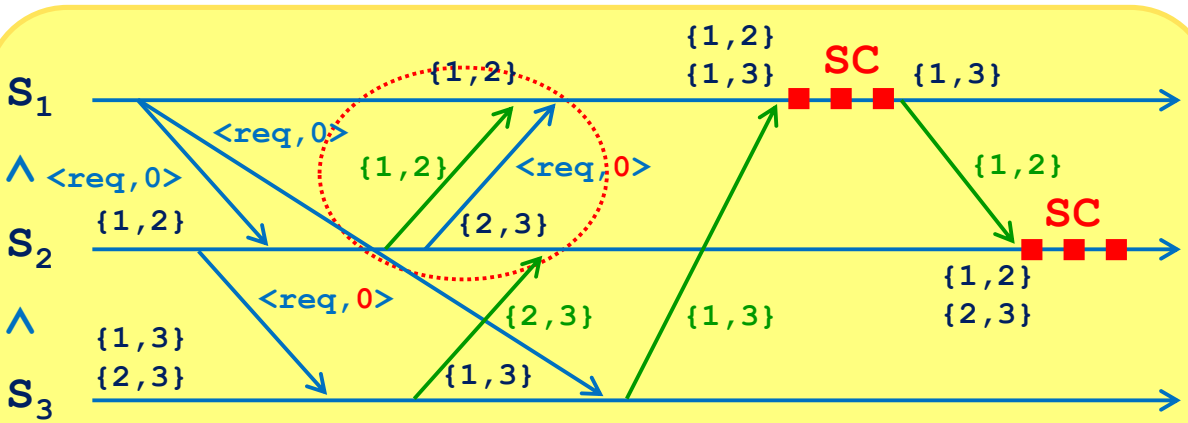


Une requête retardée par une requête plus jeune

Le site **2** entre en SC dès réception du jeton (**{2,3}**) qui lui manquait.

En cours de SC, le site **2** reçoit la requête du site **1**. Bien que celle-ci soit plus âgée que sa propre requête, le site **2** ne peut donner le jeton car l'exclusion mutuelle ne serait plus garantie.

L'algorithme doit distinguer trois états **{repos, attente, en_SC}**.



Un site qui donne un jeton et le réclame aussitôt

La requête du site **1** arrive au site **2** avant que ce dernier ne rentre en SC. Dans ce cas, la priorité donnée à la requête la plus âgée s'applique et le site **2** donne son jeton au site **1**. Il doit cependant le lui réclamer aussitôt car il ne pourrait entrer en SC. Toutes les requêtes sont datées de la même heure qu'elles soient émises immédiatement ou sur une perte ultérieure d'un jeton.

Algorithme de Carvalho et Roucairol

Variables du site **i** :

- h_i : variable contenant la valeur de l'horloge locale du site **i**. Elle est initialisée à **0**.
- $état_i$: état du site. Cette variable peut prendre l'une des trois valeurs **{repos, attente, en_SC}**. Initialisée à **repos**.
- $hreq_i$: heure de la requête courante de **i**.
- $Jeton_i[1..n]$: tableau de booléens indiquant la présence des jetons. $Jeton_i[j]$ est initialisé à la valeur **(i ≥ j)**.
- $Retardé_i[1..n]$: tableau de booléens. $Retardé_i[j]$ = **Vrai** lorsque **i** retarde l'acquittement d'une requête de **j**.
- $temp_i$: variable temporaire.

Algorithme du site **i** :

Le site réclame les jetons qui lui manquent

et

attend de posséder tous les jetons pour entrer en SC

```
Prologue(){
  hreq_i = h_i;
  Pour tout temp_i de 1 à n Faire
    Retardé_i[temp_i] = Faux;
    Si ( Jeton_i[temp_i] == Faux ) Alors
        envoyer_à(temp_i, (req, hreq_i));
    Fsi
  Finpour
  état_i = attente;
  Attendre( ∀ j, Jeton_i[j] == Vrai );
  état_i = en_SC;
}
```

Algorithme de Carvalho et Roucairol

Lors de la réception par le site **i** d'une requête émise par le site **j**,
la décision prise par **i** dépend de son état.

```
sur_réception_de(j,(req, h)){
  Si (état_i == repos) Alors
        envoyer_à(j,(acq,h_i));
        Jeton_i[j] = Faux;
  Sinon Si ( (état_i == en_SC) OU (hreq_i,i)<(h,j) ) Alors
        Retardé_i[j] = Vrai;
  Sinon
        envoyer_à(j, (acq, h_i));
        Jeton_i[j] = Faux;
        envoyer_à(j,(req, hreq_i));
  Fsi
}
```

Si le site **i** n'est pas intéressé par une SC, il envoie le jeton au site **j**.

Si le site **i** exécute sa SC ou si sa requête est plus âgée que la requête du site **j**, il retarde l'envoie du jeton au site **j** jusqu'à la fin de sa SC.

Si la requête du site **i** est plus jeune que celle du site **j**, il envoie le jeton au site **j** et le lui réclame aussitôt.

Algorithme de Carvalho et Roucairol

```
sur_réception_de(j,(acq, h)){
        Jeton_i[j] = Vrai;
}
```

A la sortie de la section critique, le site **i** envoie les jetons aux sites qu'il a retardés.

```
Epilogue(){
  Pour tout temp_i de 1 à n Faire
    Si ( Retardé_i[temp_i] ) Alors
      envoyer_à(temp_i, (acq, h_i));
      Jeton_i[temp_i] = Faux;
    Fsi
  Finpour
  état_i = repos;
}
```

Références

Cours d'Algorithmique Répartie, Joyce El Haddad et Serge Haddad, Chapitres I à VIII, Université Paris-Dauphine.

[Lam78] L. Lamport "Time, clocks and the ordering of events in a distributed system", Communications of the ACM vol 21, 7 juillet 1978, pp 558-565

[Ric81] G. Ricart, A.K. Agrawala "An optimal algorithm for mutual exclusion in computer networks", Communications of ACM vol 24, janvier 1981, pp 9-17

[Car83] O.S.F. Carvalho, G. Roucaurol "On mutual exclusion in computer networks", Communications of ACM vol 26, février 1983, pp 146-147