Systèmes et Algorithmiques Répartis

Master 1
Informatique des Organisations - MIDO

Joyce EL HADDAD
elhaddad@lamsade.dauphine.fr

Chapitre 4 : La Concurrence

- Introduction
- Algorithme de Lamport
- Algorithme de Ricart et Agrawala
- Algorithme de Carvalho et Roucairol

Introduction

- La concurrence des processus lors de l'accès à des ressources partagées peut provoquer l'*incohérence* de celles-ci
- Gérer la concurrence revient à contrôler que des accès concurrents s'exécutent de manière cohérente
- La manière la plus simple de parvenir à cette cohérence consiste à garantir que pour chaque ressource au plus une section de code qui la manipule soit en cours d'exécution. Une telle section de code est appelée section critique
- Objectif : la mise en œuvre de l'exclusion mutuelle entre sections critiques

Introduction

- Lorsqu'un processus applicatif désirera exécuter une section critique, son programme se présentera comme suit :

  Prologue( ); Section critique; Epilogue( );

- Prologue et Epilogue sont des primitives du service qui devront garantir les deux propriétés caractéristiques de l'exclusion mutuelle :
  - Propriété de sûreté : à tout instant, au plus un processus est en cours d'exécution d'une section critique
  - Propriété de vivacité : 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

Introduction

- De nombreux algorithmes ont été proposés dans la littérature. Parmi les plus anciens :
  - Lamport 1978
  - Ricart et Agrawala 1981
  - Carvalho et Roucairol 1983
  - …
- Ces algorithmes sont tous basés sur les horloges logiques
- Chaque nouvel algorithme se construit par une modification conceptuelle du précédent et diminue la complexité *(mesurée en nombre de messages échangés par exécution d'une section critique)*

Algorithme de Lamport 1978

- Algorithme qui repose sur le mécanisme des horloges logiques

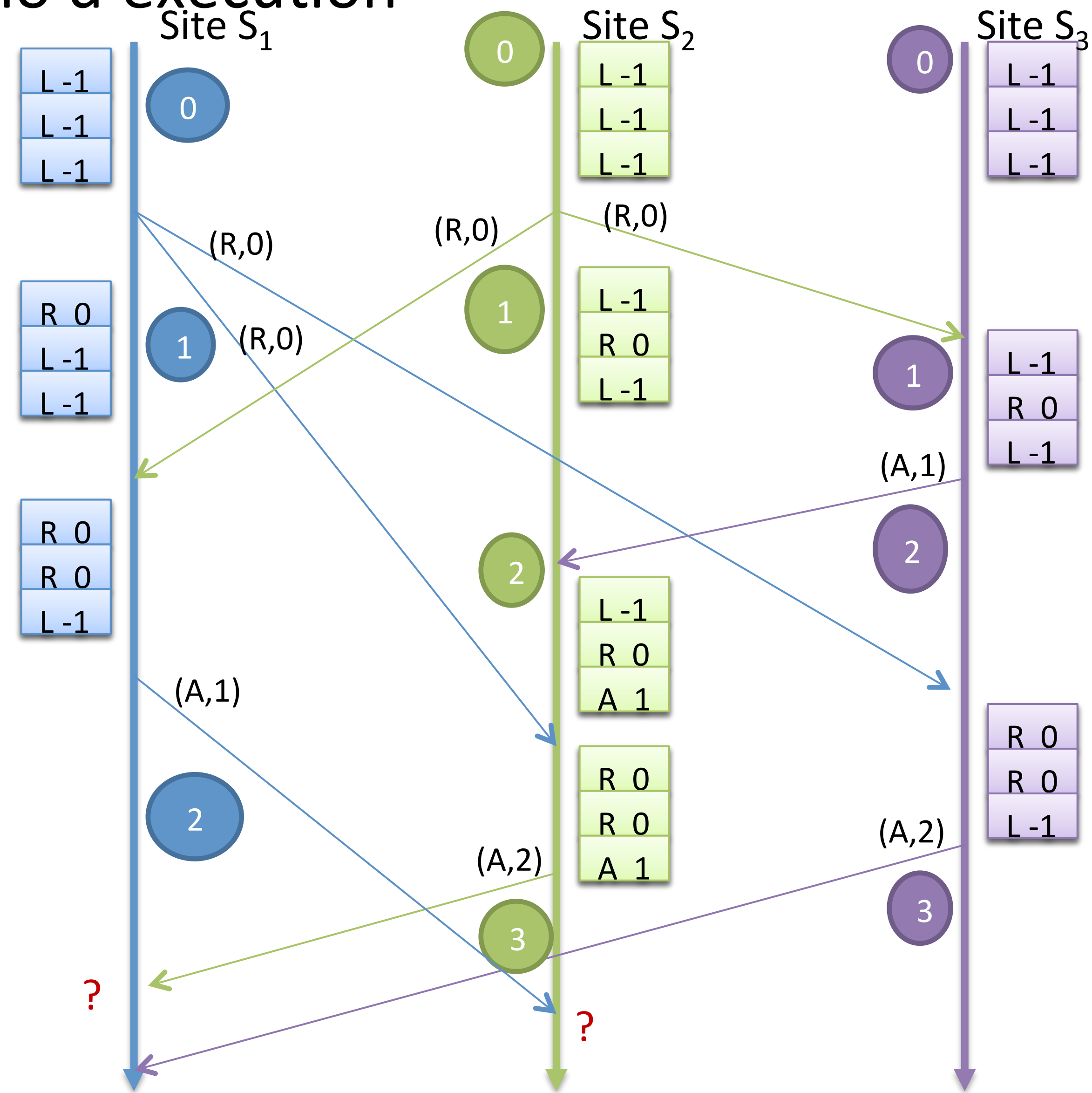
- Principe : échange de trois types de message
  - des requêtes d'exécution de section critique diffusées à tous les autres sites par le site demandeur,
  - des acquittements de requête pour indiquer que le site a "enregistré" la requête,
  - des libérations diffusées à tous les autres sites par un site qui a terminé l'exécution d'une section critique.

Algorithme de Lamport 1978

- Point clef : la condition d'entrée en section critique
  - Chaque site maintient un tableau de messages indicé par les identités des sites
  - Chaque cellule de ce tableau contient le type du message et son heure. Toutes les cellules sont initialisées avec un message de libération daté de l'heure –1
  - Lorsqu'un site désire exécuter sa section critique, il met à jour sa propre cellule avec sa requête
  - La condition d'entrée en section critique est alors d'avoir dans sa cellule le message le plus âgé du tableau
  - La règle de mise à jour des autres cellules du tableau?

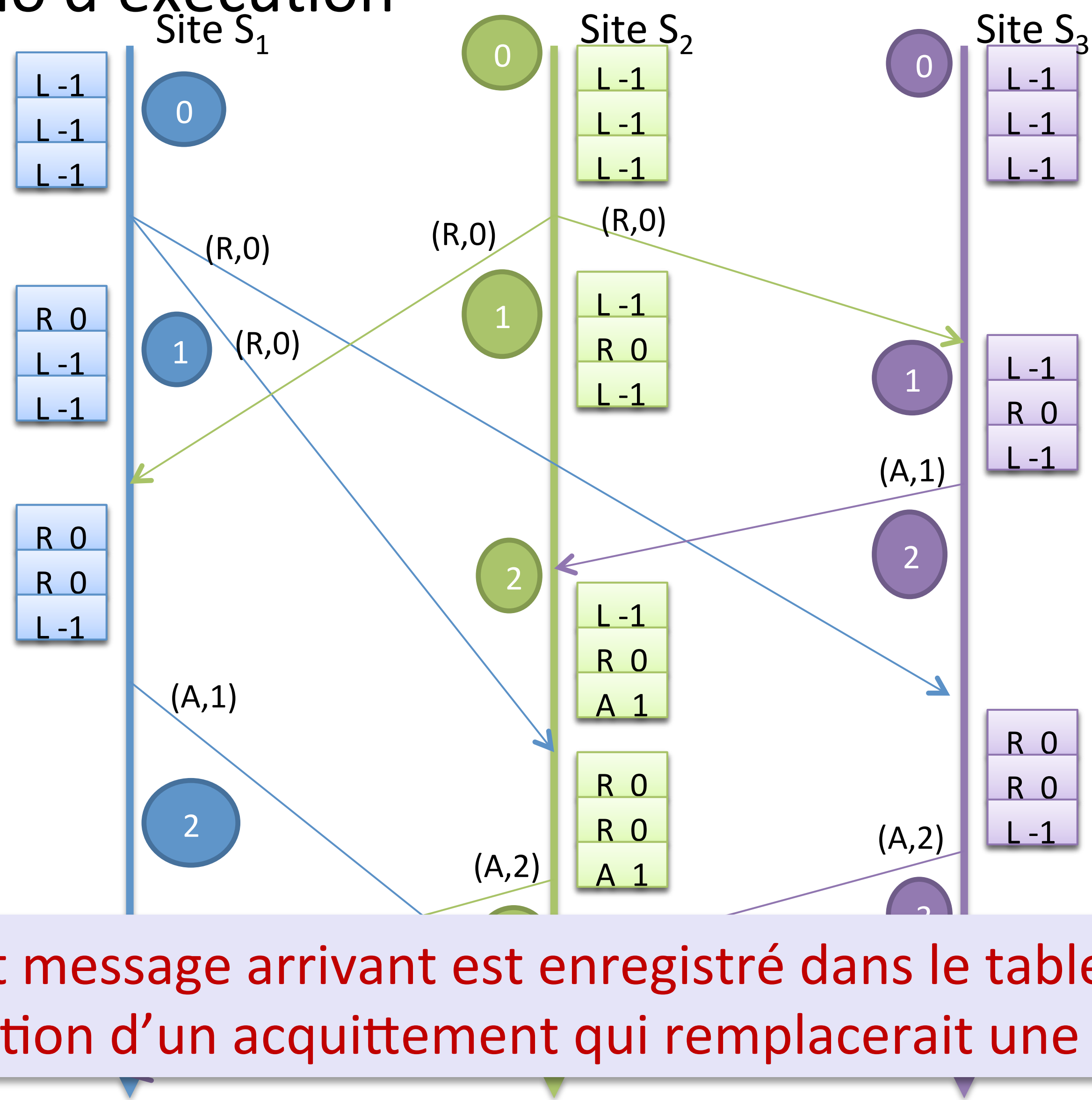
Algorithme de Lamport 1978

- Scénario d'exécution



Algorithme de Lamport 1978

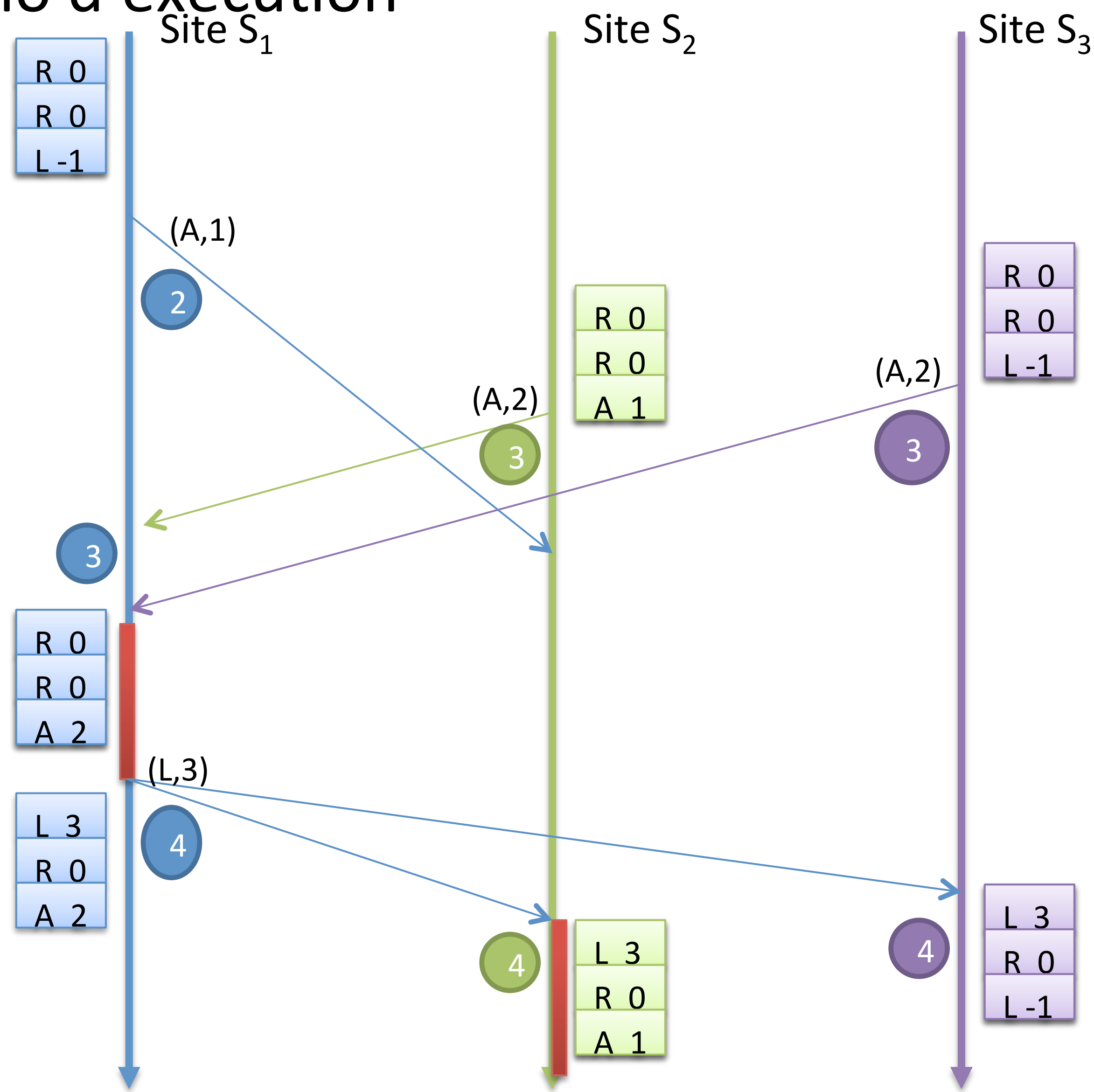
- Scénario d'exécution



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

Algorithme de Lamport 1978

- Scénario d'exécution



Algorithme de Lamport 1978

- Les variables d'un site S_i
  - h_i : variable contenant la valeur de l'horloge locale du site S_i. Elle est initialisée à 0.
  - $Mess_i[1..N]$: tableau des messages. Chaque cellule de ce tableau est composée des deux champs <type,heure>. Le champs type prend ses valeurs parmi {Req, Acq, Lib} et est initialisé à Lib. Le champs heure est initialisé à -1.

Algorithme de Lamport 1978

- Algorithme d'un site S_i

Prologue( )

Début

$Mess_i[i].heure = h_i;$

$Mess_i[i].type = req;$

$Diffuser(req,h_i);$

$h_i = h_i + 1;$

$Attendre\ (\forall j \neq i,\ (Mess_i[i].heure,i) < (Mess_i[j].heure,j));$

Fin

Epilogue( )

Début

$Mess_i[i].heure = h_i;$

$Mess_i[i].type = lib;$

$Diffuser(lib,h_i);$

$h_i = h_i + 1;$

Fin

Algorithme de Lamport 1978

- Algorithme d'un site S_i

```
Sur_réception_de ( j, (tp, h) )
      Début
            hᵢ=max(hᵢ,h+1) ;
            Si (tp == Req) Alors
                        envoyer_à(j, (acq, hᵢ) );
                        hᵢ++;
            Finsi
            Si (tp != Acq) || (Messᵢ[j].type != Req) Alors
                        Messᵢ[j].heure = h;
                        Messᵢ[j].type = tp;
            Finsi
      Fin
```

Algorithme de Lamport 1978

- Complexité en nombre de messages dans le pire des cas
  - Soit n le nombre de sites dans le système
  - Une section critique s'accompagne de 3(n-1) messages
    - (n-1) messages de requête,
    - (n-1) messages d’acquittement, et
    - (n-1) messages de libération

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de vivacité
    - Nous raisonnons par l'absurde
    - Supposons qu'il existe une exécution au cours de laquelle un ensemble de sites E_0 reste indéfiniment bloqué au cours d'un appel à Prologue()
    - Les autres sites peuvent être partitionnés en deux catégories : E_1 l'ensemble des sites qui n'accèdent qu'un nombre fini de fois à une section critique et E_2 l'ensemble des sites qui accèdent un nombre infini de fois à une section critique

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de vivacité
    - Plaçons nous en un instant t_0 où : tous les messages des requêtes associées aux appels à Prologue() non terminés des sites de E_0 ont été reçus par les autres sites et donc enregistrés dans leurs tableaux. Ces messages restent indéfiniment dans les tableaux. Tous les acquittements associés à ces requêtes ont été reçus. Tous les messages de libération associés aux derniers appels à Epilogue() des sites de E_1 ont été reçus par les autres sites et donc enregistrés dans leurs tableaux.
    - Montrons que E_2 est vide

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de vivacité
    - Supposons qu'il existe un site $S_i \in E_2$
    - Par définition de E_2, l'application de S_i appelle Prologue() après l'instant t_0
    - D'après le mécanisme des horloges logiques, cette requête a nécessairement une heure plus grande que celle d'une requête non traitée d'un quelconque site de E_0
    - Puisque tous les messages des requêtes non terminés ont été reçus par les autres sites et donc enregistrés dans leurs tableaux, S_i reste bloqué indéfiniment
    - Ce qui est contradictoire avec la définition de E_2

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de vivacité
    - Soit $S_i \in E_0$ et $S_j \in E_1$
    - Examinons le contenu de la cellule $Mess_i[j]$ après l'instant t_0. L'acquittement par S_j de la requête non traitée de S_i a été reçu :
      - S'il a été enregistré, il ne peut bloquer S_i puisque son heure est plus grande que celle de la requête. Les messages, qui éventuellement lui succèderont dans cette cellule, ayant des heures plus grandes ne peuvent non plus bloquer S_i
      - S'il n'a pas été enregistré, cela signifie qu'une requête était présente dans $Mess_i[j]$. Dans ce cas, par définition de E_1, S_i recevra au pire à l'instant t_0 le message de libération correspondant .

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de vivacité
    - Soit $S_i \in E_0$ et $S_j \in E_1$
    - Examinons le contenu de la cellule Mess$_i$[j] après l'instant t_0. L'acquittement par S_j de la requête non traitée de S_i a été reçu :
      - Ce message de libération ne pourra bloquer S_i puisque son heure est plus grande que celle de l'acquittement et donc de la requête non traitée de S_i. Les messages, qui éventuellement lui succèderont dans cette cellule, ayant des heures plus grandes ne peuvent non plus bloquer S_i
    - Un site de E_1 ne peut bloquer un site de E_0 à partir de l'instant t_0

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de vivacité
    - Soit $S_i \in E_0$ dont la requête non traitée est la plus âgée
    - A l'instant t_0, il ne peut être bloqué par un message des autres sites de E_0 ni par les messages des sites de E_1
    - En conséquence son appel à Prologue() devrait se terminer. D'où la contradiction.

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de sûreté
    - Raisonnement par l’absurde
    - Supposons que deux sites soient à un même instant en cours d’exécution d’une section critique
    - Soit S_1 le site dont la requête correspondante est la plus jeune et S_2 l’autre site.
    - S_1 diffuse la requête correspondante r_1 à l’heure logique h_1 et le site S_2 diffuse la sienne (r_2) à l’heure logique h_2.
    - Soit m un message émis par S_2 à destination de S_1, présent dans le tableau au moment où S_1 pénètre en section critique

Algorithme de Lamport 1978

- Vérification de l'algorithme
  - Propriété de sûreté
    - *r_2 précède m* : quand S_1 pénètre en section critique, S_2 n'a pas terminé la section critique correspondant à r_2. Il n'a donc pas envoyé de message de libération et les messages d'acquittement ne peuvent remplacer r_2 dans le tableau de S_1
    - *r_2 et m confondus* : la requête du site S_2 étant plus âgée, elle empêcherait S_1 de pénétrer en section critique
    - *m précède r_2* : soit h l'heure logique de m. Puisque S_1 pénètre en section critique $(h_1,1) < (h,2)$. Puisque m précède r_2, $(h,2) < (h_2,2)$. Par transitivité, $(h_1,i_1) < (h_2,i_2)$. Ce qui est contradictoire avec nos hypothèses.

Algorithme de Ricart et Agrawala 1981

- Dans l'algorithme de Lamport, un acquittement s'interprète comme le fait d'enregistrer la requête

- Le principal apport de l'algorithme de Ricart et Agrawala consiste à modifier la sémantique de l'acquittement

- Un acquittement de S_j à une requête de S_i signifie que S_j autorise S_i à pénétrer en section critique

- La condition d'entrée en section critique devient la réception d'un acquittement venant de chacun des autres sites

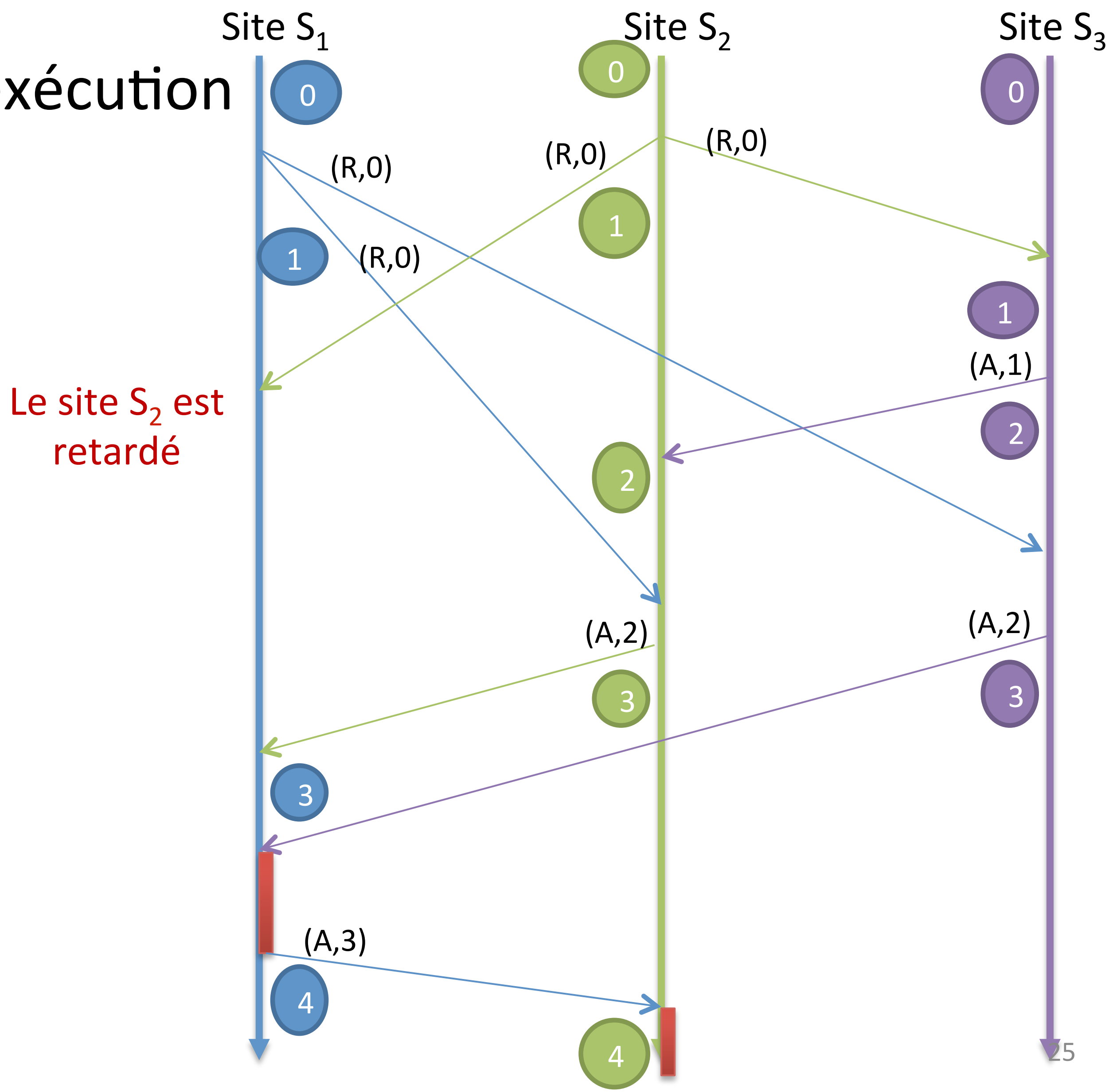
- Comme chaque site n'envoie qu'un seul acquittement relatif à une requête, il suffit de compter le nombre d'acquittements reçus

Algorithme de Ricart et Agrawala 1981

- La politique du site S_j lorsqu'il reçoit cette requête :
  - Si son application n'est pas en attente ou en cours d'exécution d'une section critique, alors il délivre immédiatement l'acquittement.
  - Sinon, S_j compare l'âge de sa requête avec celui de la requête de S_i. Si cette dernière est plus âgée, l'acquittement est délivré immédiatement. Dans le cas contraire, S_j diffère l'acquittement jusqu'à la fin de sa section critique. Les messages de libération de l'algorithme de Lamport s'interprètent alors comme des acquittements différés. Bien entendu, il s'agit pour un site de mémoriser, les sites qu'il a retardés pour réaliser cet envoi différé.

Algorithme de Ricart et Agrawala 1981

- Scénario d'exécution



Algorithme de Ricart et Agrawala 1981

- Les variables d'un site S_i
  - h_i : valeur de l'horloge logique. Elle est initialisée à 0.
  - $état_i$: état du site. Cette variable peut prendre l'une des deux valeurs {repos, en_cours}. Initialement, tous les sites sont dans l'état repos.
  - $hreq_i$: heure de la requête courante.
  - $nbacq_i$: compteur des acquittements reçus.
  - $retarde_i[1..N]$: tableau de booléens. $retarde_i[j]$ est vrai lorsque S_i retarde l'acquittement d'une requête de S_j.

Algorithme de Ricart et Agrawala 1981

- Algorithme d'un site S_i

```
Prologue()
    Début
        hreq_i = h_i;
        nbacq_i = 0;
        Pour temp_i de 1 à n faire /*n est le nombre de sites*/
            retarde_i[temp_i] = Faux; /* temp_i  est une variable temporaire*/
        Finpour
        état_i = en_cours;
        Diffuser(req,h_i); hi++
        Attendre(nbacq_i == (n-1));
    Fin
```

Algorithme de Ricart et Agrawala 1981

- Algorithme d'un site S_i

```
Sur_réception_de (j, (req, h) )
    Début
        hi=max(hi, h+1)
        Si (état_i == en_cours) && ( (hreq_i,i) < (h,j) ) Alors
                retarde_i[j] = Vrai;
        Sinon
                envoyer_à(j,(acq,h_i)); hi++
        Finsi
    Fin
```

```
Sur_réception_de (j, (acq, h) )
    Début
    hi=max(hi, h+1)
        nbacq_i ++;
    Fin
```

Algorithme de Ricart et Agrawala 1981

- Algorithme d'un site S_i

```
Epilogue()
    Début
        Pour tout temp_i de 1 à n faire
            Si retarde_i[temp_i] Alors
                envoyer_à(temp_i, (acq, h_i) );
            Finsi
        Finpour
        hi++;
        état_i = repos;
    Fin
```

Algorithme de Ricart et Agrawala 1981

- Complexité en nombre de messages dans le pire des cas
  - Soit n le nombre de sites dans le système

  - Une section critique s'accompagne de 2(n-1) messages
    - (n-1) messages de requête, et
    - (n-1) messages d'acquittement

  - La complexité de cet algorithme est donc meilleure que celle de l'algorithme de Lamport.

Algorithme de Carvalho et Roucairol 1983

- Dans l'algorithme de Ricart et Agrawala, un acquittement s'interprète comme le fait d'autoriser l'entrée en section critique

- L'apport de l'algorithme de Carvalho et Roucairol consiste à modifier la sémantique de l'acquittement

- Un acquittement de S_j à une requête du site S_i signifie que S_j autorise S_i à entrer en section critique pour la section critique courante mais aussi pour les sections critiques suivantes

- L'acquittement s'interprète alors comme l'envoi d'une permission partagée entre S_i et S_j

Algorithme de Carvalho et Roucairol 1983

- Si S_j veut par la suite entrer en section critique, il doit réclamer la permission qu'il a concédée à S_i.

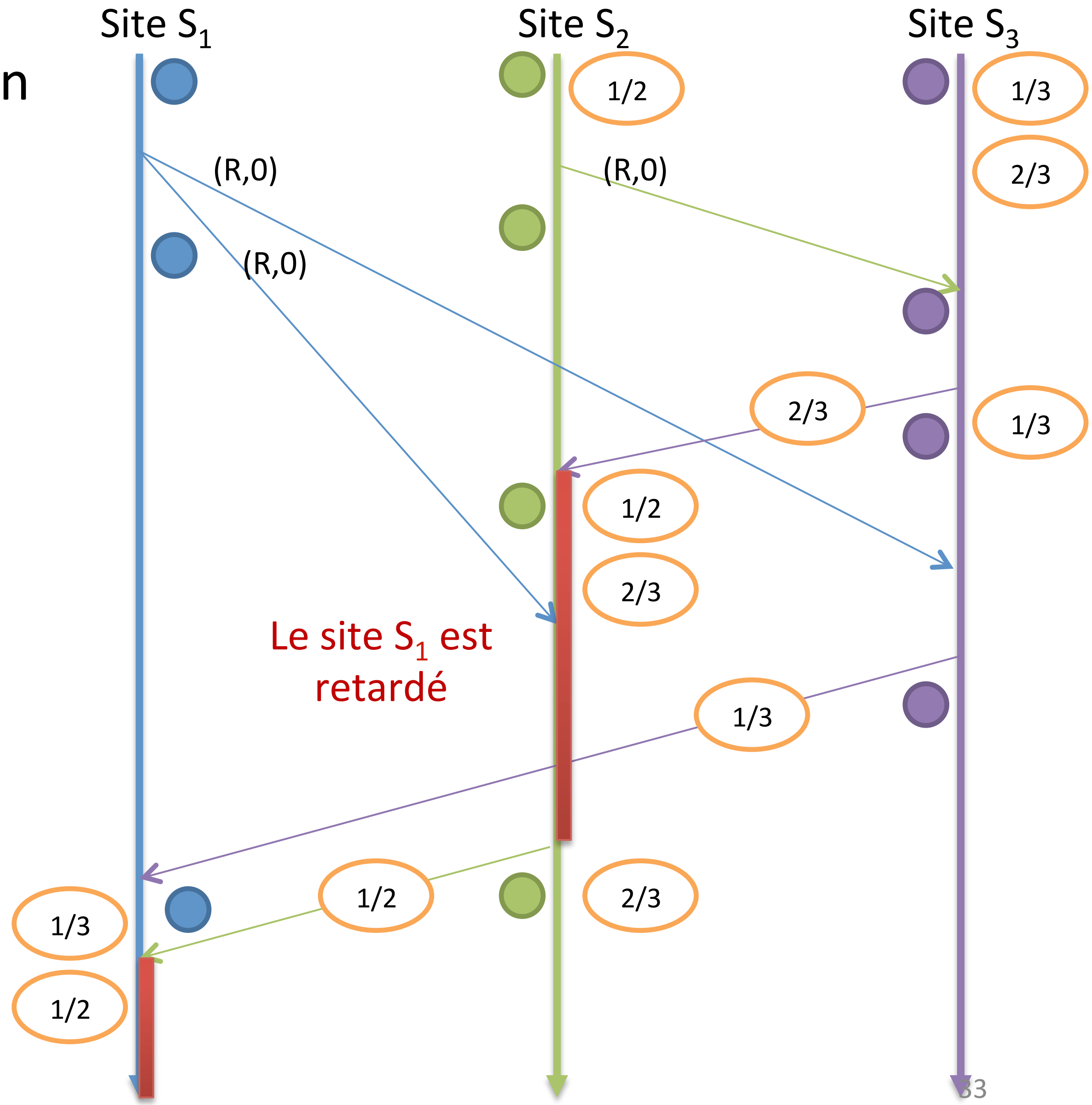
- La condition d'entrée en section critique devient la possession des permissions partagées avec chacun des autres sites

- Les permissions sont similaires aux jetons multiples. Initialement, le jeton du couple de sites {i,j} est possédé par le site d'identité max(i,j)

- Lors du Prologue() seules les permissions manquantes seront réclamées. Ce qui signifie qu'un site peut entrer immédiatement en section critique sans échanger un seul message !

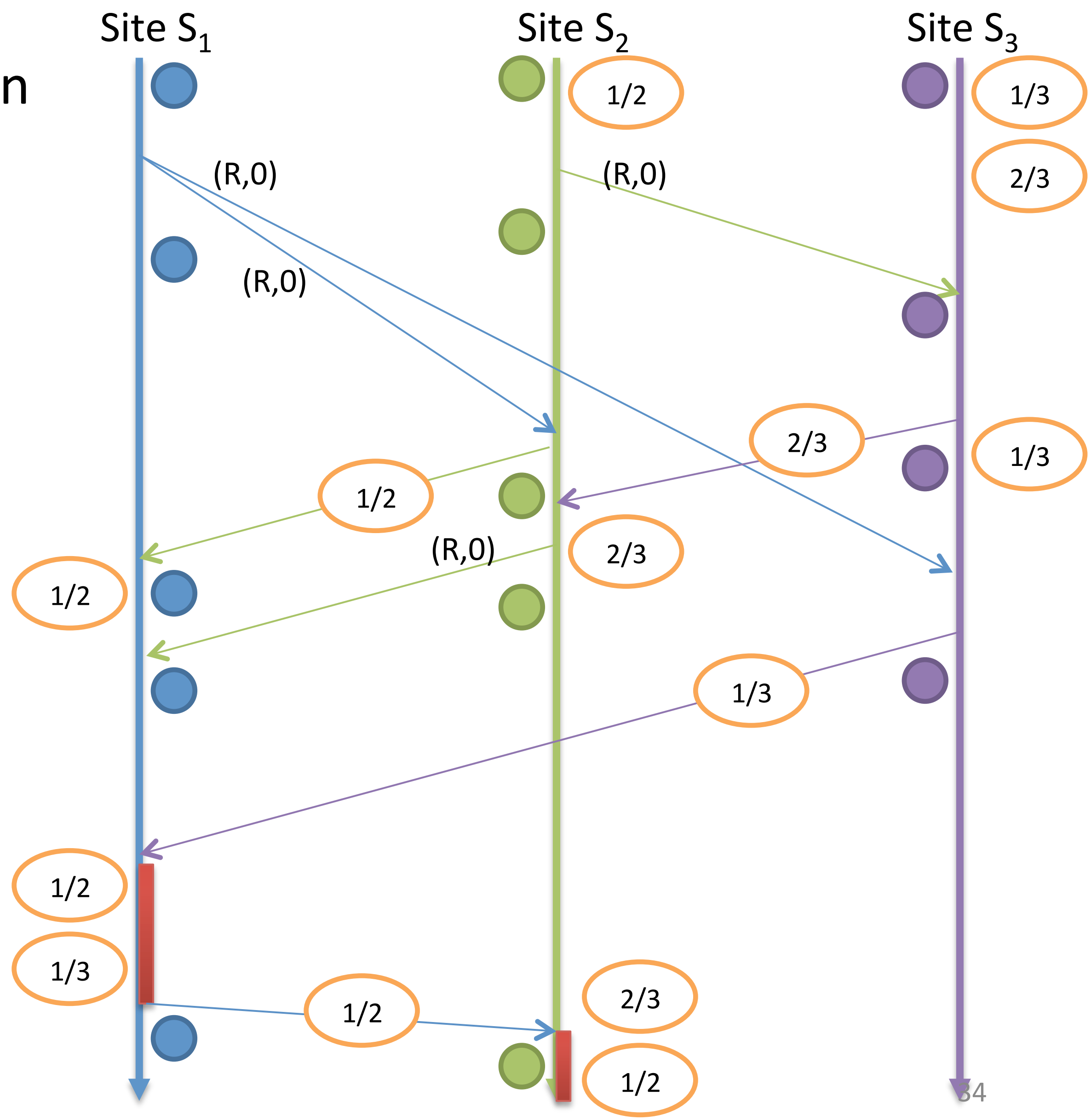
Algorithme de Carvalho et Roucairol 1983

- Scénario d'exécution



Algorithme de Carvalho et Roucairol 1983

- Scénario d'exécution



Algorithme de Carvalho et Roucairol 1983

- Les variables d'un site S_i
  - h_i : valeur de l'horloge logique. Elle est initialisée à 0.
  - $état_i$: état du site. Cette variable peut prendre l'une des deux valeurs {repos, attente,en_SC}. Initialement, tous les sites sont dans l'état repos.
  - $hreq_i$: heure de la requête courante.
  - $Jeton_i[1..N]$: tableau de booléens indiquant la présence des jetons. $Jeton_i[j]$ est initialisé à Vrai pour $i \geq j$.
  - $retarde_i[1..N]$: tableau de booléens. $retarde_i[j]$ est vrai lorsque i retarde l'acquittement d'une requête j.

Algorithme de Carvalho et Roucairol 1983

- Algorithme d'un site S_i

```
Prologue()
    Début
        hreq_i = h_i;
        Pour temp_i de 1 à n faire /*n est le nombre de sites*/
            retarde_i[temp_i] = Faux; /* temp_i  est une variable temporaire*/
            Si Jeton_i[temp_i] == Faux Alors
                envoyer_à (temp_i, (req, hreq_i) ) ;
            Finsi
        Finpour
        état_i = attente;
        Attendre(∀j,  Jeton_i[j] == Vrai);
        état_i = en_SC;
    Fin
```

Algorithme de Carvalho et Roucairol 1983

- Algorithme d'un site S_i

```
Sur_réception_de (j, (req, h) )
    Début
        Si (état_i == repos) Alors
                envoyer_à (j, (acq, h_i) );
                Jeton_i[j] = Faux;
        Sinon   si  (état_i==en_SC)|| ( (hreq,i) < (h,j) ) Alors
                        retarde_i[j] = Vrai;
                sinon
                        envoyer_à (j, (acq, h_i) );
                        Jeton_i[j] = Faux;
                        envoyer_à (j, (req, hreq_i) );
                Finsi
    Fin
```

Algorithme de Carvalho et Roucairol 1983

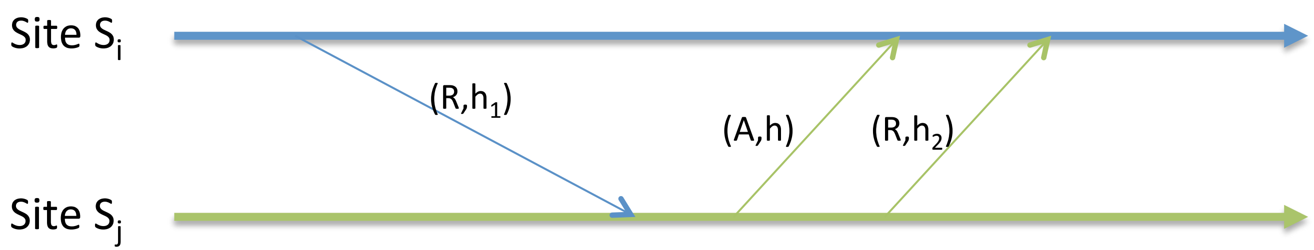
- Algorithme d'un site S_i

```
Sur_réception_de (j, (acq, h) )
    Début
        Jeton_i[j] = Vrai;
    Fin
```

```
Epilogue()
    Début
        Pour temp_i de 1 à n faire
            Si retarde_i[temp_i] Alors
                envoyer_à(temp_i, (acq, h_i) );
                Jeton_i[temp_i] = Faux;
            Finsi
        Finpour
        état_i = repos;
    Fin
```

Algorithme de Carvalho et Roucairol 1983

- Complexité en nombre de messages dans le pire des cas
  - Si un site possède tous ses jetons, il entre immédiatement en section critique
  - Dans le meilleur des cas, il n'y a pas de messages échangés pour entrer en section critique
  - Démontrons maintenant qu'un site ne réclame un jeton particulier qu'au plus une fois
  - Supposons le scénario suivant : Le site S_i récupère le jeton (i,j) et le site S_j le lui réclame avant que S_i ne soit entré en section critique



Algorithme de Carvalho et Roucairol 1983

- Complexité en nombre de messages dans le pire des cas
  - Deux cas se présentent :
    - Soit l'émission de la requête correspond à l'appel à Prologue() et donc $h_2 > h > h_1$ auquel cas le site S_j sera retardé.
    - Soit l'émission de la requête correspond à la perte du jeton lors de la réception de la requête de S_i et donc $(h_2,j) > (h_1,i)$ auquel cas le site S_j sera ici aussi retardé.
  - Par conséquent, le nombre de messages nécessaire à une section critique varie entre 0 et 2(n-1) puisqu'il y a au plus (n-1) requêtes et donc (n-1) acquittements.

Références

- O.S.F. Carvalho, G. Roucairol, "On mutual exclusion in computer networks" , Communication of ACM, volume 26, numero 2, pp. 146-147, 1983

- G. Ricart, A.K. Agrwala, "An optimal algorithm for mutual exclusion in computer networks" , Communication of ACM, volume 24, pp. 9-17, 1981

- L. Lamport, "Time, clocks, and the ordering of events in a distributed system" , Communications of the ACM, volume 21, pp. 558-565, 1978