L'EXCLUSION MUTUELLE DANS UN ENVIRONNEMENT DISTRIBUÉ

SAIDOUNI Djamel Eddine

Université Constantine 2 - Abdelhamid Mehri
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département d'Informatique Fondamentale et ses Applications

Laboratoire de Modélisation et d'Implémentation des Systèmes Complexes

Djamel.saidouni@univ-constantine2.dz
saidounid@hotmail.com
Tel: 0559082425

ALGORITHME DE RICART ET AGRAWALA

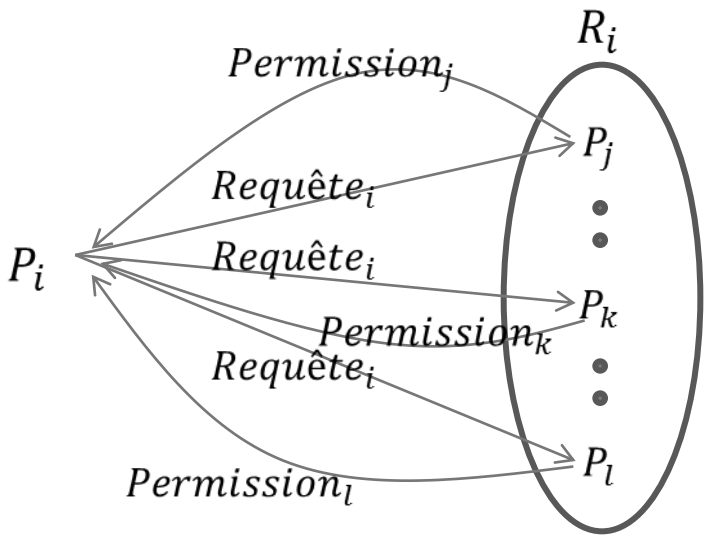
ALGORITHME DE RICART ET AGRAWALA

Principe:

- n sites. L'ensemble des sites est indexé par l'ensemble $\{\mathbf{1}, \dots, \boldsymbol{n}\}$
- Les ensembles R_i de chaque site i vérifient $\forall \boldsymbol{i} \neq \boldsymbol{j}, \ \boldsymbol{i} \in \boldsymbol{R_j}$ $\boldsymbol{etj} \in \boldsymbol{R_i}$, c'est-à-dire $\forall \boldsymbol{i}, \boldsymbol{R_i} = \{\mathbf{1}, \dots, \boldsymbol{n}\} - \{\boldsymbol{i}\}$
- Chaque site i est doté d'une horloge H_i

Une utilisation de la SC nécessite $\mathbf{2} * (\boldsymbol{n} - \mathbf{1})$ messages, $(\boldsymbol{n} - \mathbf{1})$ requêtes et $(\boldsymbol{n} - \mathbf{1})$ permissions

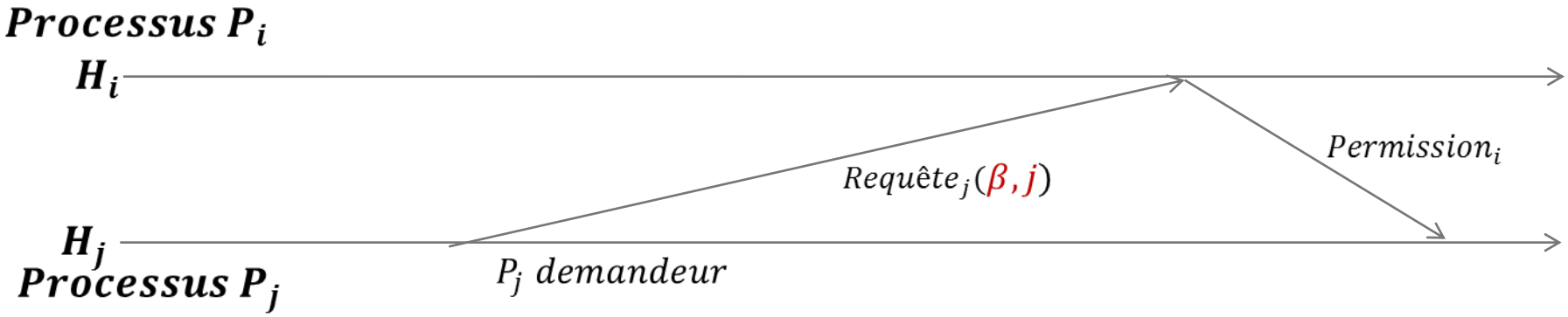
La réception de toutes les permissions demandées est une **condition nécessaire et suffisante** pour l'accès à la section critique.



ALGORITHME DE RICART ET AGRAWALA (SUITE)

Principe (suite):

- Lorsque le site i reçoit un message $requête(\beta, j)$ alors:



Cas 1: P_i n'est pas prioritaire, il donne sa permission

$$\begin{cases} \textbf{Soit } P_i \text{ n'est pas demandeur} \\ \textbf{Soit } P_i \text{ est demandeur avec } (\beta, j) < (\alpha, i) \end{cases}$$

Cas 2: P_i est prioritaire, il diffère la donnée de sa permission

$$\begin{cases} \textbf{Soit } P_i \text{ est dedans} \\ \textbf{Soit } P_i \text{ est demandeur avec } (\alpha, i) < (\beta, j) \end{cases}$$

ALGORITHME DE RICART ET AGRAWALA (SUITE)

Evènements :

- Evènements à l'initiative du processus
  - Demander l'accès à la section critique
  - Libérer la section critique
- Evènement subit par le processus
  - Recevoir une requête en provenance d'un processus
  - Recevoir une permission d'un autre processus

A chaque évènement on associe une procédure, d'où les procédures.

L'ALGORITHME

Variables locales pour un processus P_i

- $\boldsymbol{R_i} = \{1, \dots, n\} - \{i\}$
- $\boldsymbol{Etat_i}: \{dehors, demandeur, dedans\}$
- $\boldsymbol{H_i}: entier\ croissant\ init$ à 0
- $\boldsymbol{Last_i}: entier\ croissant\ init$ à 0
- $\boldsymbol{Priorité_i}: Booléen$
- $\boldsymbol{Attendus_i}: ensemble\ de\ sites\ init$ à $\emptyset$
- $\boldsymbol{Différés_i}: ensemble\ de\ sites\ init$ à $\emptyset$

Remarque: L'indice i est rajouter aux noms des variables pour faciliter la compréhension de l'algorithme (on peut s'en passer de cet indice).

L'ALGORITHME

PROCÉDURES DU PROCESSUS P_i

Lors d'un appel à acquérir

$Etat_i = demandeur$;

H_i++ ;

$Last_i = H_i$;

$Attendus_i = R_i$;

$\forall j \in R_i : envoyer\ requête(Last_i, i)$ à j ;

$attendre(Attendus_i = \emptyset)$;

$Etat_i = dedans$;

L'ALGORITHME

PROCÉDURES DU PROCESSUS P_i (SUITE)

Lors d'un appel à libérer

$$Etat_i = dehors\ ;$$

$$\forall j \in Différés_i : \textbf{\textit{envoyer}}\ permission(i)\ \text{à}\ j\ ;$$

$$Différés_i = \emptyset\ ;$$

Lors de la réception de $permission(j)$

$$Attendus_i = Attendus_i - \{j\}\ ;$$

L'ALGORITHME

PROCÉDURES DU PROCESSUS P_i (SUITE)

Lors de la réception de $requête(K, j)$

$H_i = Max(H_i, K)_i$;

$Priorité_i = (Etat_i = dedans)\ ou\ \big((Etat_i = demandeur)\ et\ (Last_i, i) < (K, j)\big)$;

Si $Priorité_i$ **Alors**

$$Différés_i = Différés_i \cup \{j\}$$

Sinon

envoyer $permission(i)$ à j ;

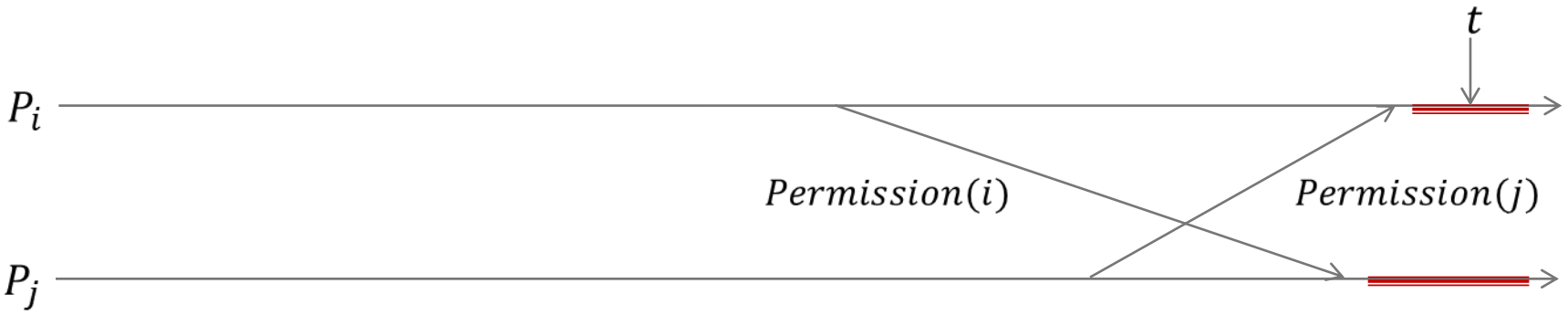
Fsi

PREUVE DE L'ALGORITHME
SÛRETÉ

$\forall t, \exists$ *au plus un site en SC*. On procède par l'absurde.

On suppose, pour $i \neq j$, à *un instant* t, P_i *en* SC_i *et* P_j *en* SC_j.

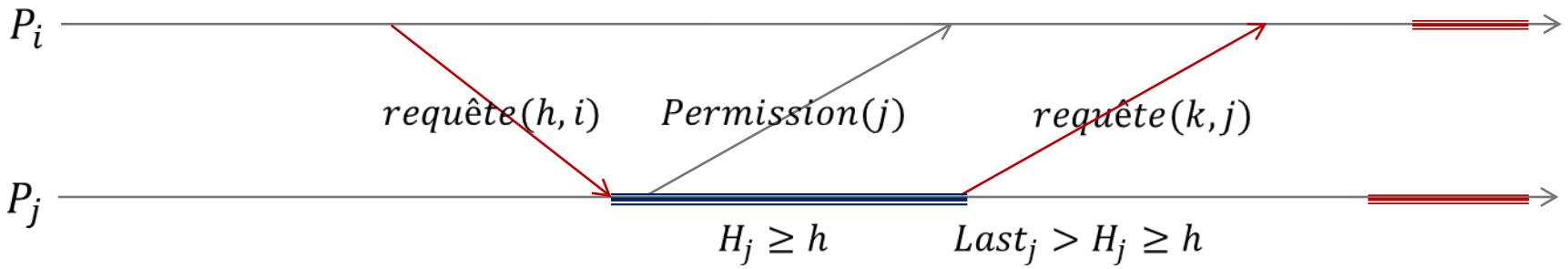
Donc $\begin{cases} P_i \text{ a envoyé } requête(h,i) \text{ à } P_j \text{ et a obtenu sa permission et} \\ P_j \text{ a envoyé } requête(k,j) \text{ à } P_i \text{ et a obtenu sa permission} \end{cases}$



Trois cas se distinguent selon l'état de chacun des processus lors de la réception du message requête.

PREUVE DE L'ALGORITHME
SÛRETÉ

Cas 1: P_i a envoyé sa requête à P_j et celui-ci l'a reçue avant de faire la sienne.

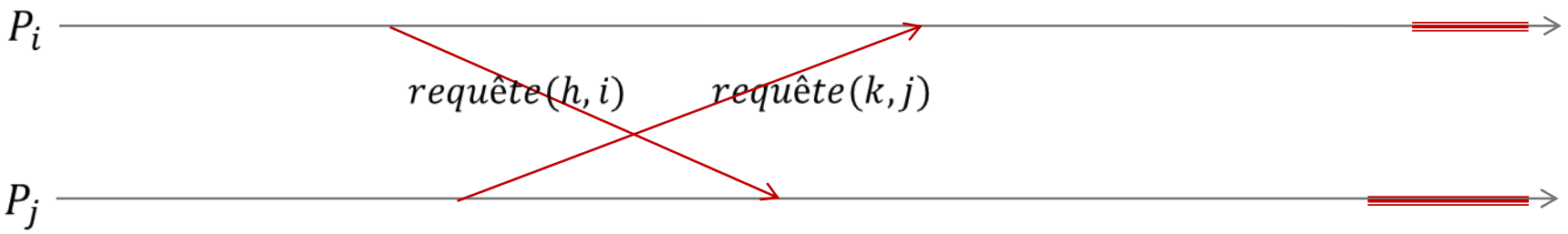


- Dans ce cas, à la réception du message $requête(h,i)$ par P_j, ce dernier est dans l'état $dehors$. Il met à jour son horloge et envoie sa permission à P_i et $H_j \geq h$.
- Lorsque P_j exécute $acquérir$ et envoie sa requête on aura: $k = Last_j = H_j + 1$. Donc $k > h$.
- A la réception du message $requête(k,j)$ par P_i, ce dernier se trouvera prioritaire et n'enverra pas sa permission. Ce qui est **absurde** avec l'hypothèse de travail.

Cas 2: Cas inverse, idem au premier par l'inversion des rôles des deux processus.

PREUVE DE L'ALGORITHME
SÛRETÉ (SUITE)

Cas 3: P_i et P_j envoient leur requête respective en même temps.



- Dans ce cas, soit $(h,i) < (\mathrm{k,j})$ ou $(\mathrm{k,j}) < (\mathrm{h,i})$.
- Etant donné que les deux processus sont dans l'état demandeur à la réception de la requête de l'autre processus, l'un des deux se trouvera prioritaire et n'enverra pas sa permission à l'autre processus. Ce qui est **absurde** avec l'hypothèse de travail.

On conclue que la propriété de sûreté est vérifiée

PREUVE DE L'ALGORITHME

VIVACITÉ

Les estampilles des requêtes sont totalement ordonnées. Il existe donc une requête de plus petite estampille, soit (h, i). Lorsque celle-ci arrive aux sites, chacun répond favorablement et envoie sa permission au site i qui finira par accéder à sa SC. La résidence en SC étant de durée fini, à la sortie de sa SC, la requête du site i sortira du système. Le site dont la requête et de plus petite valeur obtiendra ainsi les permissions requises et accèdera à son tour à sa SC.

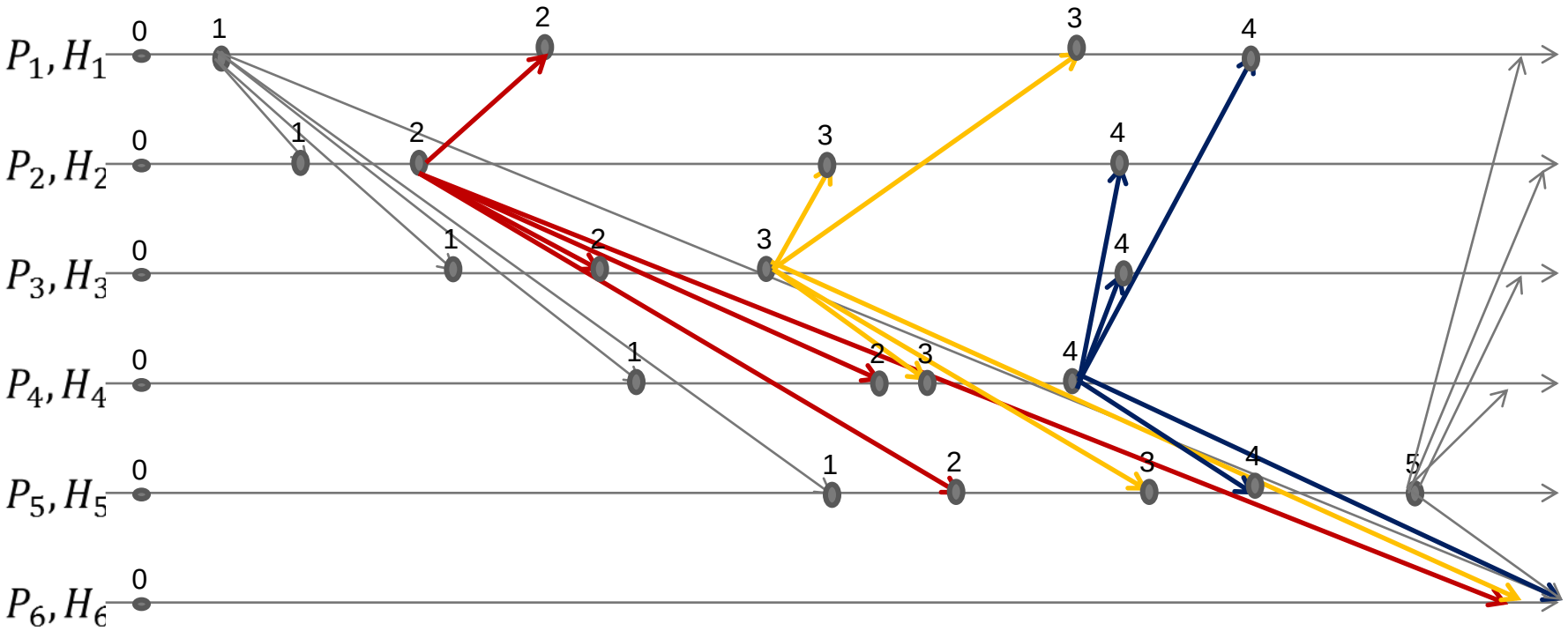
Par ailleurs si le site i deviendra demandeur, l'estampille de sa nouvelle requête est supérieur aux estampilles des requêtes aux quelles il a donné sa permission. D'où la dynamicité du site de plus petite estampille.

Conclusion: La vivacité est donc vérifiée.

PREUVE DE L'ALGORITHME
BORNÉTUDE DES VARIABLES

Propriété: Dans un système à n sites, **l'écart maximal** entre deux horloges quelconques H_i et H_j est égal à $n-1$.

On considère le cas qui creuse l'écart au maximum entre les horloges.

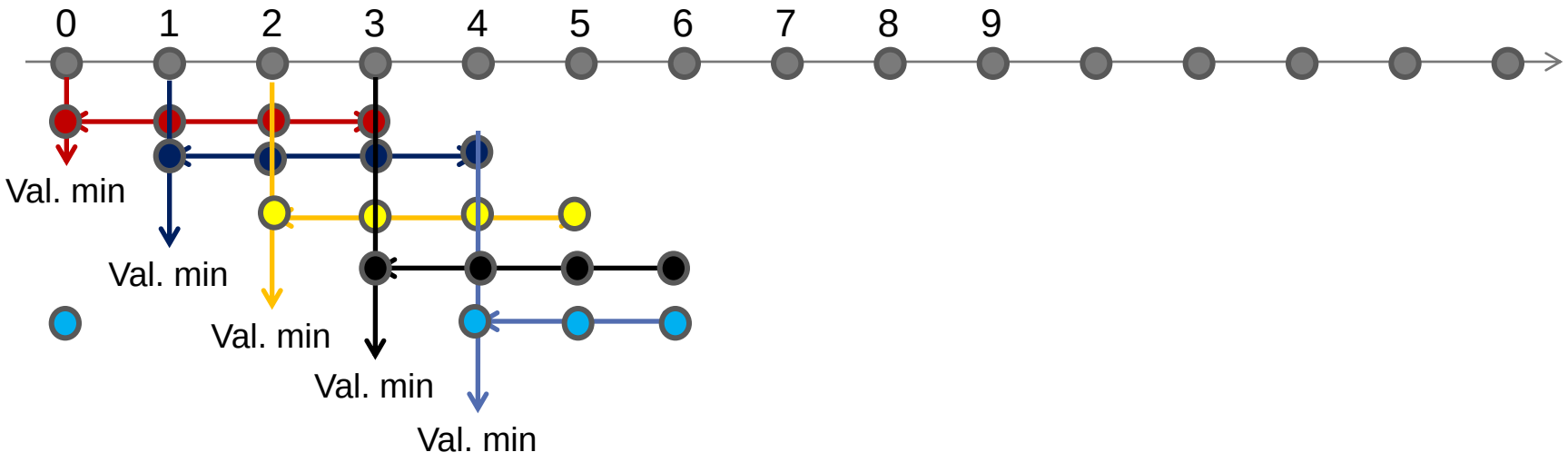


L'écart au maximum entre les horloges est égal à 5.

PREUVE DE L'ALGORITHME
BORNÉTUDE DES VARIABLES (SUITE)

Propriété: Dans l'algorithme de Ricart et Agrawla on peut incrémenter les horloges des sites modulo $2 * n - 1$.

Exemple: $n = 4$, on travaille modulo 7.



$$écart > (n-1) \Rightarrow la\ plus\ petite\ valeur\ est\ la\ plus\ récente\ date$$

L'ALGORITHME AVEC VARIABLES BORNÉES

Lors de la réception de $requête(K, j)$

$H_i = Max(H_i, K)$;

Si $(Etat_i = dedans)$ **Alors** $Priorité_i = True$;

Sinon Si $(Etat_i = demandeur)$ **Alors**

Si $|Last_i - K| < n \ et \ (Last_i, i) < (K, j)$ **Alors**

$Priorité_i = True$

Sinon $Priorité_i = (Last_i > K)$ **Finsi**

Finsi;

Si $Priorité_i$ **Alors** $Différés_i = Différés_i \cup \{j\}$

Sinon ***envoyer*** $permission(i)$ à j **Finsi**

PREUVE DE L'ALGORITHME
CALCUL DE LA PRIORITÉ

Propriété: Dans l'algorithme de Ricart et Agrawala, le calcul de la priorité peut être remplacé par:

$$Priorité_i = (Etat_i \neq dehors)\ et\ \big(\ (Last_i, i) < (K, j)\big)$$

Preuve: On montre que

$$(Etat_i = dedans)\ ou\ \big((Etat_i = demandeur)\ et\ (Last_i, i) < (K, j)\big)$$
$$\Leftrightarrow$$
$$(Etat_i \neq dehors)\ et\ \big(\ (Last_i, i) < (K, j)\big)$$

Propriété: L'algorithme de Ricart et Agrawala a les propriétés de performance suivantes:

- Nombre de message de contrôle = 2*(n-1)
- Si T est le temps moyen d'acheminement d'un message entre deux sites, le temps moyen que la SC reste libre alors qu'il y a des processus demandeurs est égale à : 2*T
- Les variables horloges peuvent être bornées.
- L'algorithme n'est pas adaptatif dans le sens qu'un site qui n'est pas intéressé par l'accès à la SC est toujours sollicité par les sites demandeurs d'accès à la SC.