# 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**

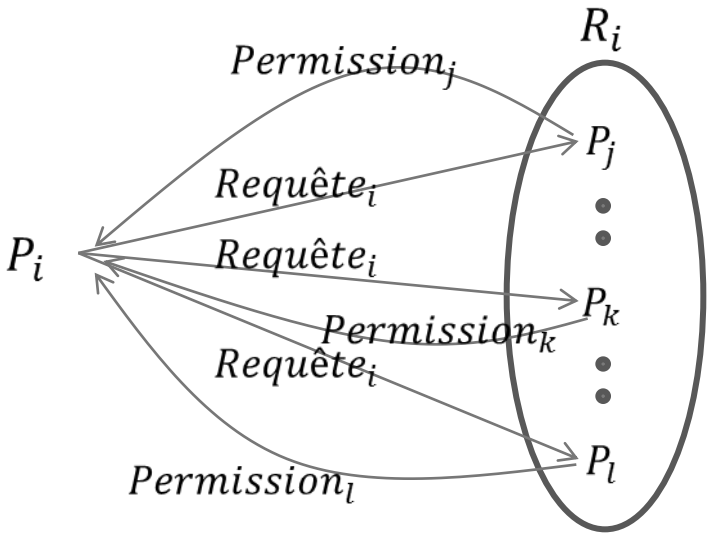
**Laboratoire de Modélisation et d'Implémentation des Systèmes Complexes**

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

# ALGORITHMES BASÉS SUR LES PERMISSIONS

# LES PERMISSIONS POUR GARANTIR LA SÛRETÉ

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



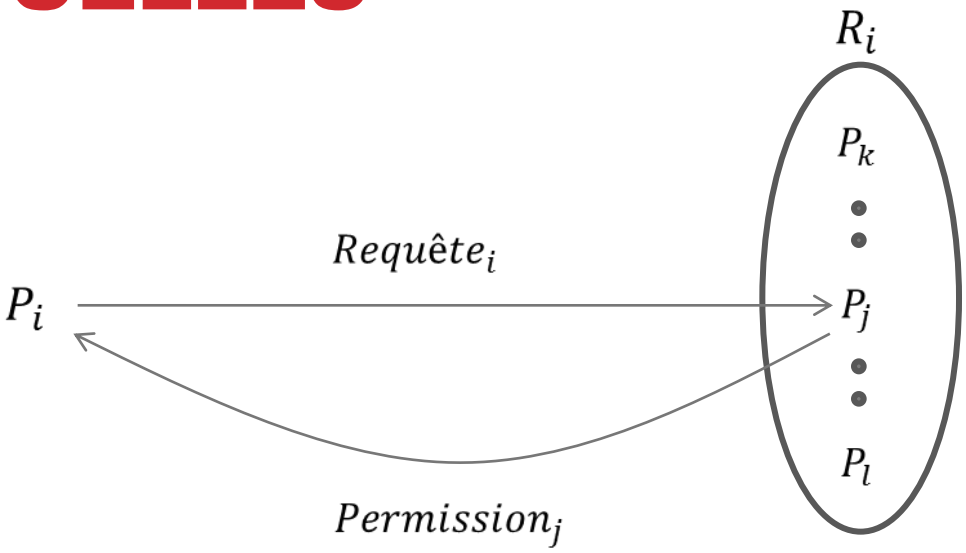

Que doit vérifier les ensembles $R_i$ pour garantir la sûreté ?

Les ensembles $R_i$ doivent vérifier certaines propriétés selon la nature des permissions.

On distingue deux familles d'algorithmes :

- Les algorithmes à permissions individuelles.
- Les algorithmes à permissions d'arbitre

# ALGORITHMES À PERMISSIONS INDIVIDUELLES



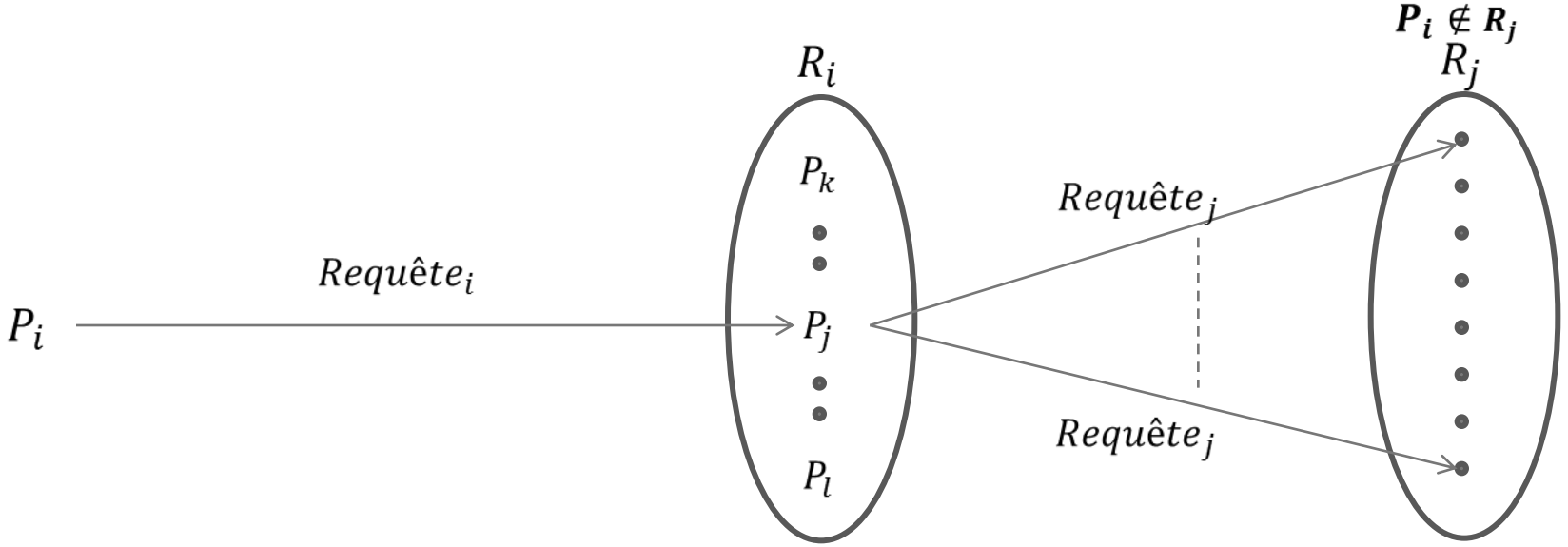

$P_j$ donne sa permission en se référant uniquement à son état, c'est-à-dire s'il n'est pas prioritaire par rapport à $P_i$:

- Soit $P_j$ est dans l'état dehors.
- Soit $P_j$ est demandeur mais sa requête est plus récente que celle de $P_i$

$P_i$ doit obtenir les permissions des autres processus pour accéder à sa SC

La sémantique d'une permission donnée par $P_j$ à $P_i$ est : En ce qui me concerne, vous pouvez rentrer dans votre SC.

# ALGORITHMES À PERMISSIONS INDIVIDUELLES (SUITE)




Montrons que la condition:

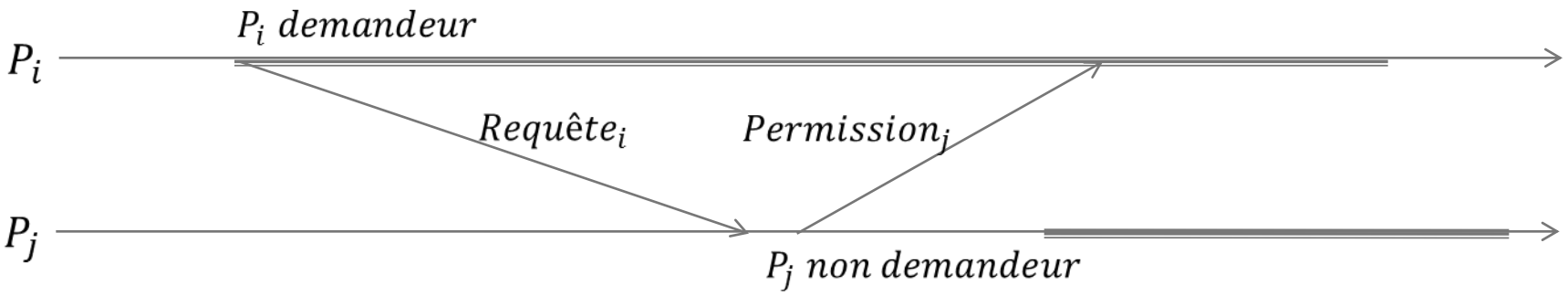
$$\forall i \neq j,\ P_i \in R_j \text{ ou } P_j \in R_i$$

est **suffisante** pour assurer l'exclusion mutuelle (la sûreté).

# ALGORITHMES À PERMISSIONS INDIVIDUELLES (SUITE)

$$\boldsymbol{Cas\ extrême\ P_i \notin R_j\ et\ P_j \in R_i}$$

**Cas 1:**

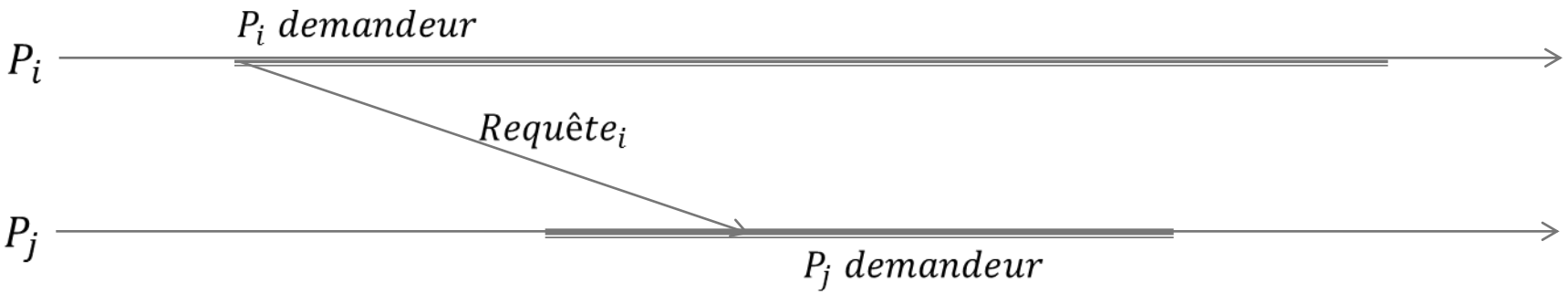



Dans ce cas $P_j$ donne sa permission à $P_i$. Même si $P_j$ deviendra demandeur par la suite et reçoit toutes les permissions requises, il se souviendra qu'il a donné sa permission au processus $P_i$, il diffèrera l'accès à sa section critique jusqu'à la réception d'un message $RetourPermission_i$ du processus $P_i$ qui est envoyé après la sortie de $P_i$ de sa SC.

# ALGORITHMES À PERMISSIONS INDIVIDUELLES (SUITE)

$$P_i \notin R_j \; et \; P_j \in R_i$$

**Cas 2:**




Dans ce cas deux sous cas se présentent:

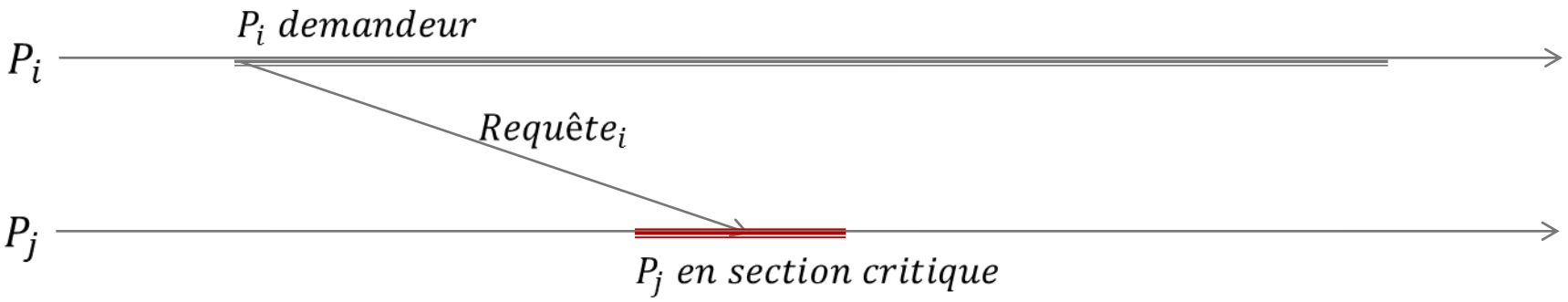
*a)* $P_j$ est prioritaire: Dans ce cas il diffère la donnée de sa permission.

*b)* $P_j$ n'est pas prioritaire: Dans ce cas il donne sa permission et conditionnera son accès à sa SC par la réception d'un message $RetourPermission_i$ du processus $P_i$.

# ALGORITHMES À PERMISSIONS INDIVIDUELLES (SUITE)

$$P_i \notin R_j \; et \; P_j \in R_i$$

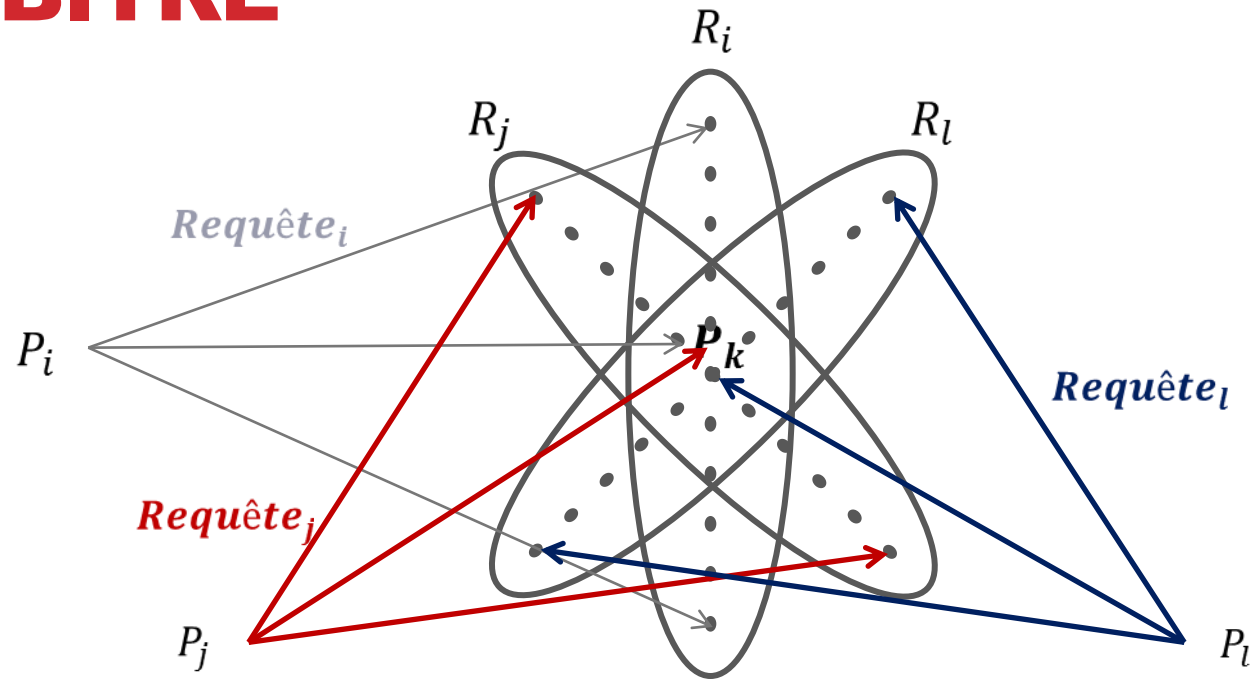
**Cas 3:**




Dans ce cas $P_j$ est prioritaire, il diffère donc la donnée de sa permission à $P_i$.

**Conclusion:** qu'avec la condition $\forall i \neq j, \; P_i \in R_j \; ou \; P_j \in R_i$ est donc **suffisante** pour assurer l'exclusion mutuelle (la sûreté) dans le cas d'un algorithme à permissions individuelles.

# ALGORITHMES À PERMISSIONS D'ARBITRE




$P_k$ joue le rôle de l'arbitre pour les processus $P_i$, $P_j$ et $P_l$.

- Il donne sa permission à un et un seul processus qui le sollicite.
- Lorsque le processus ayant eu sa permission sort de sa SC il envoie un message $RetourPermission$ à $P_k$.
- Ce dernier donnera donc sa permission au processus le plus prioritaire parmis ceux qui sont en attente.

# ALGORITHMES À PERMISSIONS D'ARBITRE (SUITE)

La sémantique d'une permission donnée par $P_k$ (l'arbitre) est:

***En ce qui concerne les processus qui m'ont sollicité, vous pouvez rentrer dans votre SC.***

Pour assurer l'exclusion mutuelle il faut régler le conflit entre chaque deux processus. Cela est possible par l'existence d'un arbitre entre chaque deux processus. Cela se traduit par:

$$\forall \boldsymbol{i \neq j}, \boldsymbol{R_i \cap R_j \neq \emptyset}$$

# GARANTIR LA VIVACITÉ

**Constat:**

- Absence d'une horloge globale. Donc impossibilité d'ordonner les requêtes selon leur date d'occurrence.
- On a besoin d'ordonner uniquement les évènements qui sont corrélés (une émission d'un message précède sa réception).
- Etant donné qu'un processus ne connait pas l'état exact des autres processus, il suffit d'imposer un ordre entre les requêtes même si cela ne correspond pas à l'ordre dans le temps physique.

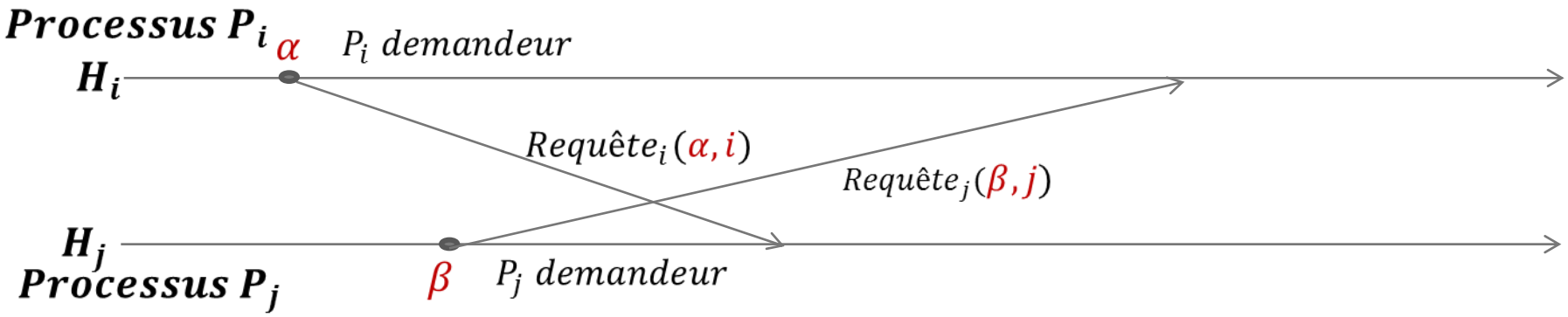
**Solution:**

- Utilisation des horloges logiques (Leslie Lamport) et utilisation d'un protocole pour leur gestion.

# HORLOGES LOGIQUES

## PRINCIPE

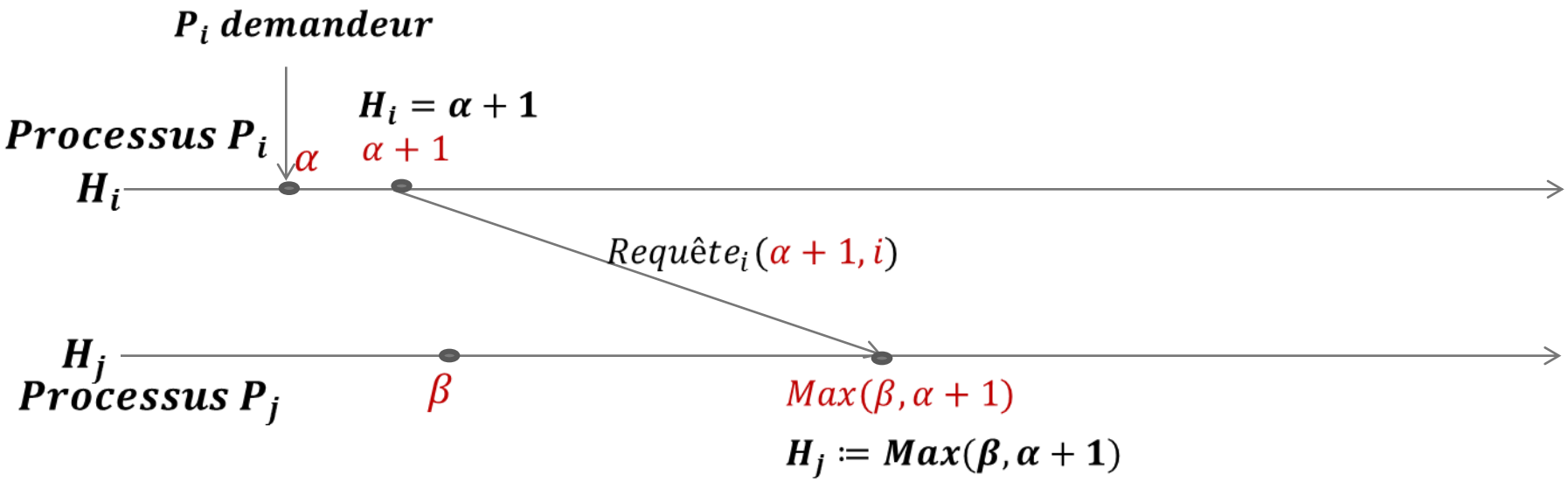
- $Site\ Si \rightsquigarrow H_i$ tel que $H_i$: $Variable\ de\ type\ Naturel$ initialisée à $0$.
- Estampiller chaque requête d'un processus par $P_i$ par la valeur courante de l'horloge $H_i$.
- Les horloges étant locales, deux requêtes de deux processus distincts peuvent avoir les mêmes estampilles.
  - Utilisation de l'ordre lexicographique.




$$(\alpha, i) < (\beta, j) \Leftrightarrow \begin{cases} \textbf{Soit } \alpha < \beta \\ \textbf{Soit } \alpha = \beta \ \ and\ i < j \end{cases}$$

# HORLOGES LOGIQUES
# PROTOCOLE

1) Lorsqu'un site $S_i$, par le biais du processus $P_i$, invoque l'opération **acquérir**, il incrémente l'horloge $H_i$ de 1 et utilise la nouvelle valeur de $H_i$ comme date d'occurrence de l'opération. Tout message de requête associé à cette opération est estampillé $(H_i, i)$.




2) Lorsqu'un site $S_j$, par le biais du processus $P_j$, reçoit un message $Requête_i(K, i)$ il recale son horloge $H_j$ par $H_j := Max(H_j, K)$.