Systèmes Distribués

(Exclusion Mutuelle &) Inter-blocages

Pascal Mérindol

merindol@unistra.fr
http://www-r2.u-strasbg.fr/~merindol/

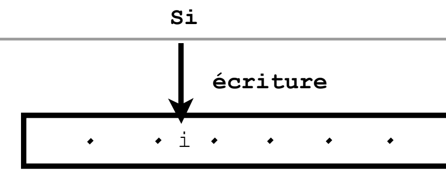
Plan du cours

Exclusion mutuelle

- Propriétés de base
  - ✓ un seul processus à la fois en Section Critique (SC)
  - ✓ SC atteignable en un temps fini par les procs en attente
  - ✓ le protocole d'exclusion mutuelle est inaltérable par des procs hors SC/attente
  - ✓ pas de processus chef
- Contexte mono-processeur
  - ✓ attente active (variable de condition, test&set)
  - ✓ attente passive : sémaphores et moniteurs
  - ➡ valable seulement dans un environnement fortement couplé (ressource commune)
- Contexte répartie => solution logicielle !
  - ✓ centralisé : un seul proc. responsable (goulet d'étranglement ?)
  - ✓ distribué : «boulangerie» (mémoire partagée), jeton, estampille

A la boulangerie !

- On s’attribue tout seul un numéro...
  - ✓ tableau T «partagé» accessible en lecture par tous les sites
  - ✓ -1 : pas dans la file, > -1 dans la file ou en SC




```
//Entree dans la file
choix[i] = 1 /* variable précisant aux autres qu’on fait notre choix de ticket */
T[i] = 1 + max(T[0],..., T[N-1])
choix[i] = 0

for(j=i ;j!=i ; j=(j+1)%N) { /* «balade» circulaire sur T */
{
attendre que (!choix[j]) /* protège le choix de ticket des autres sites */
if ((T[i],i) > (T[j],j) && (T[j] != -1)) /* ordre strict si égalité */
attendre que T[j] redevienne égale à -1
}

  => SC

//Sortie de la file
T[i] = -1
```

A la boulangerie !

- Pas d'inter-blocage ?
  - ✓ Ordre strict implique un comportement FIFO
    - ➡ progression continue vers la SC
- Exclusion mutuelle ?
  - ✓ S_i en SC et S_k en attente, prouvons que : `p :=(T[k],k) > (T[i],i) && T[i] != -1`
    - ✓ Soit S_i est entrée en SC avant que S_k ne tente d'y entrer => OK
    - ✓ Soit S_k finit son «T[k]=...» avant que S_i n'entre en SC :
      - ➡ T[k] et T[i] n'ont pas pu changés depuis cet instant
        - ➡ Au moment où S_k fait son test la propriété **p** est tjs valable => OK
- Inconvénient ?
  - ✓ Mise en oeuvre d'une mémoire commune et T[i]'s vite très grands
  - ✓ Attente active («scrutage» continu de **T** et de **choix**)

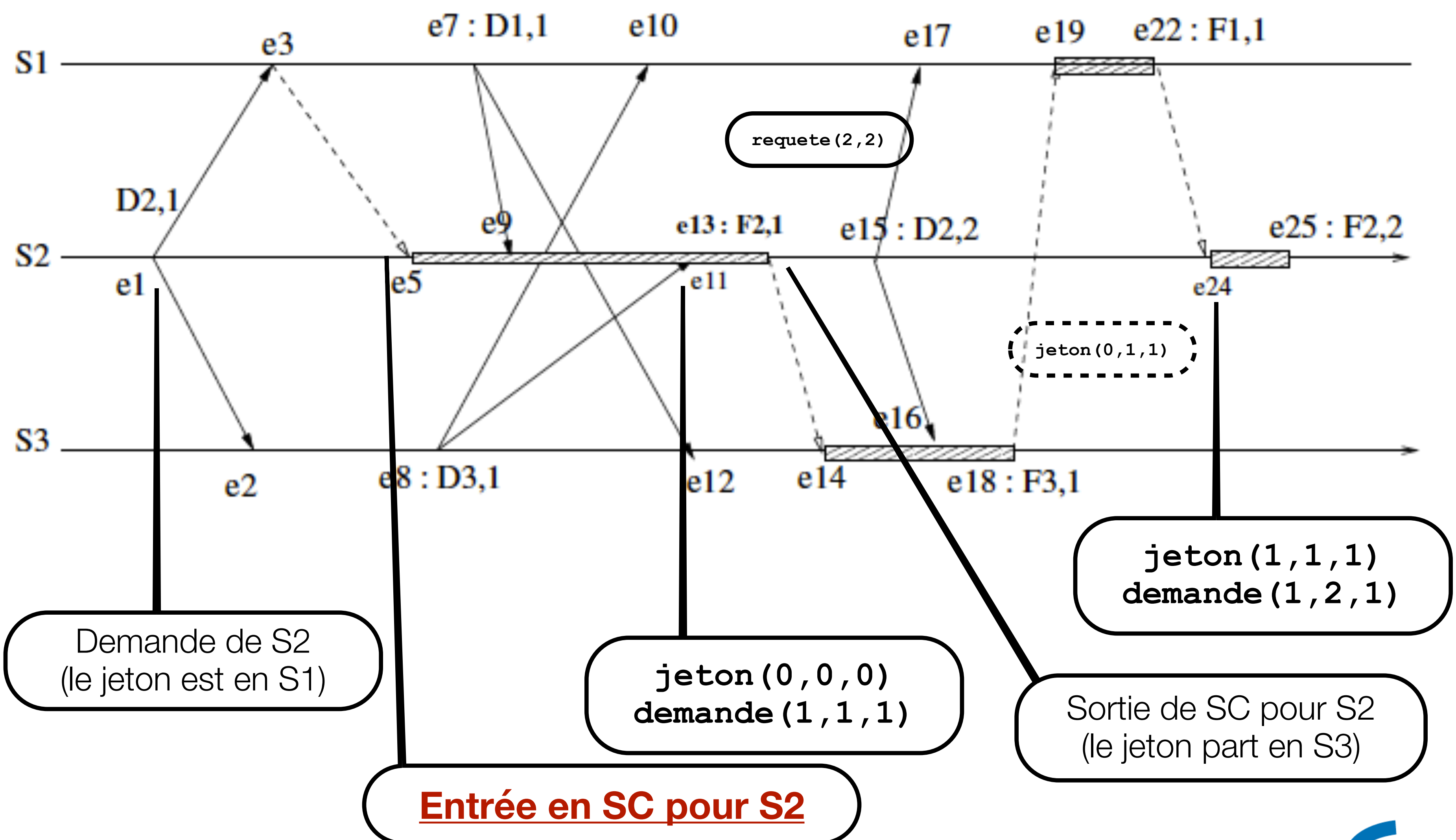
Avec un jeton (unique)

- ✓ Seul le possesseur du jeton entre en SC
- ✓ Tableau comportant N dates : **jeton**
  - ➡ estampille de la dernière entrée en SC pour chq site
- ✓ Demande du jeton = **requete** : date + id_du_site
  - ➡ tableau des demandes : **demande**

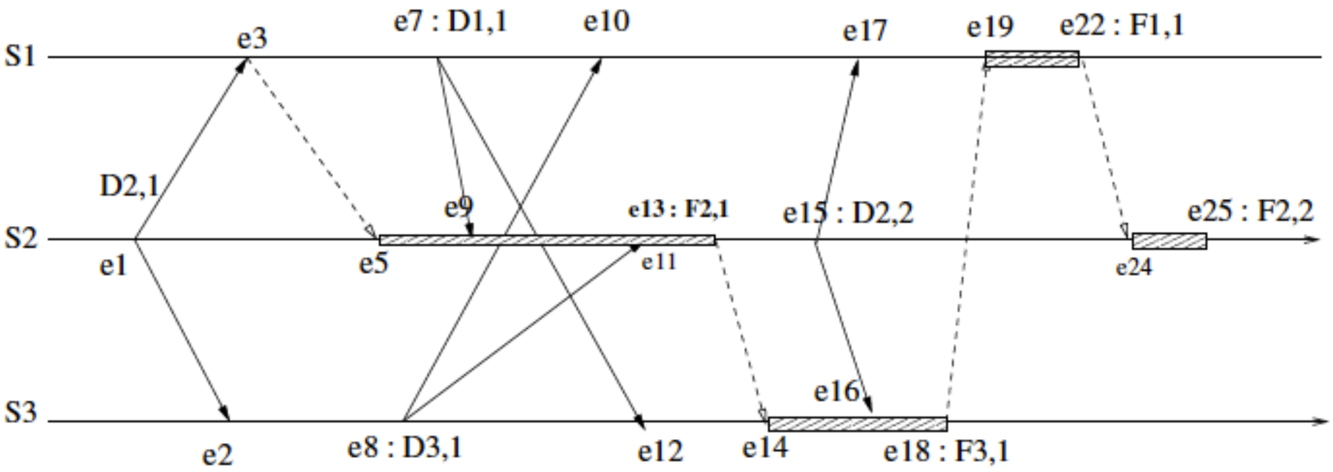
```
//Demande d'Entree en SC sur un site i
estampille ++;
requete.emetteur = i; requete.date = estampille;//dernière tentative d'accès maj
demande[i] = estampille;
diffuser(requete);
Si (!jeton_present) attendre(jeton);
dedansSC = 1;
jeton_present = 1;
     => SC
//Sortie de la SC
jeton[i] = estampille; //dernière possession maj
dedansSC = 0;
for (j=i+1;j=i;j++%N) // chacun son tour (pas de priorité)
Si (demande[j] > jeton[j]) && (jeton_present) // la demande de j est + récente que son dernier accès en SC
{jeton_present = 0; envoyer(jeton, j);}
}
```

```
//Traitement d'une Requete, requete
int k = requete.emetteur;
demande[k] = max(demande[k], requete.date);
Si (jeton_present) && (!dedansSC)
=> code ~ sortie de la SC
=> sinon rien (pas de jeton)
```

Un exemple avec jeton



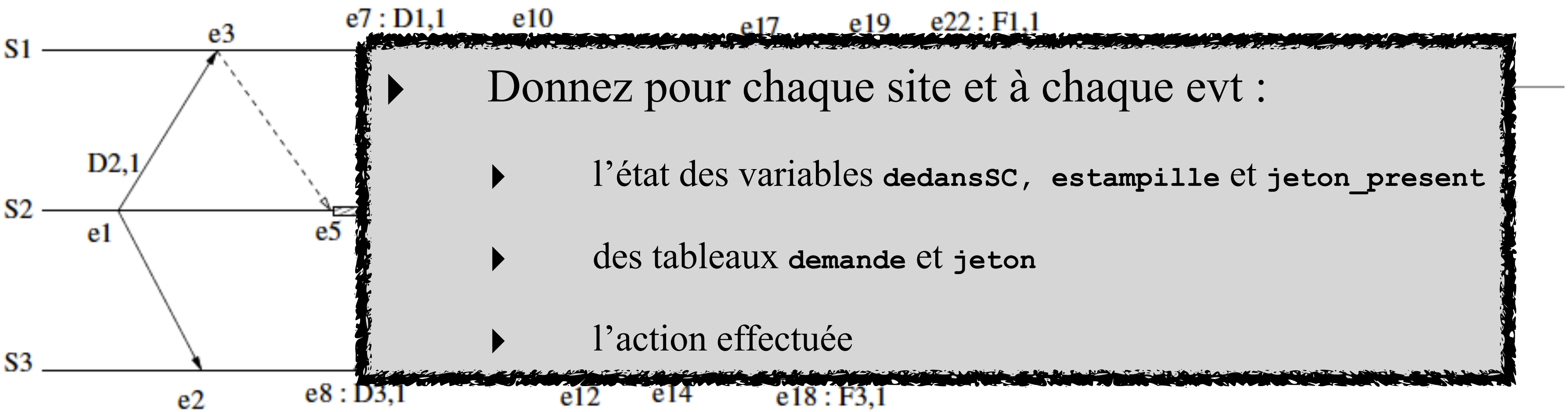
A vous de jouer




```
//Demande d'Entree en SC sur un site i
estampille ++;
requete.emetteur = i; requete.date = estampille;//dernière tentative d'accès maj
demande[i] = estampille;
diffuser(requete);
Si (!jeton_present) attendre(jeton);
dedansSC = 1;
jeton_present = 1;
     => SC
//Sortie de la SC
jeton[i] = estampille; //dernière possession maj
dedansSC = 0;
for (j=i+1;j=i;j++%N)  // chacun son tour (pas de priorité)
Si (demande[j] > jeton[j]) && (jeton_present) // la demande de j est + récente que son dernier accès en SC
{jeton_present = 0; envoyer(jeton, j);}
```

```
//Traitement d'une Requete, requete
int k = requete.emetteur;
demande[k] = max(demande[k], requete.date);
Si (jeton_present) && (!dedansSC)
=> code ~ sortie de la SC
=> sinon rien (pas de jeton)
```

A vous de jouer

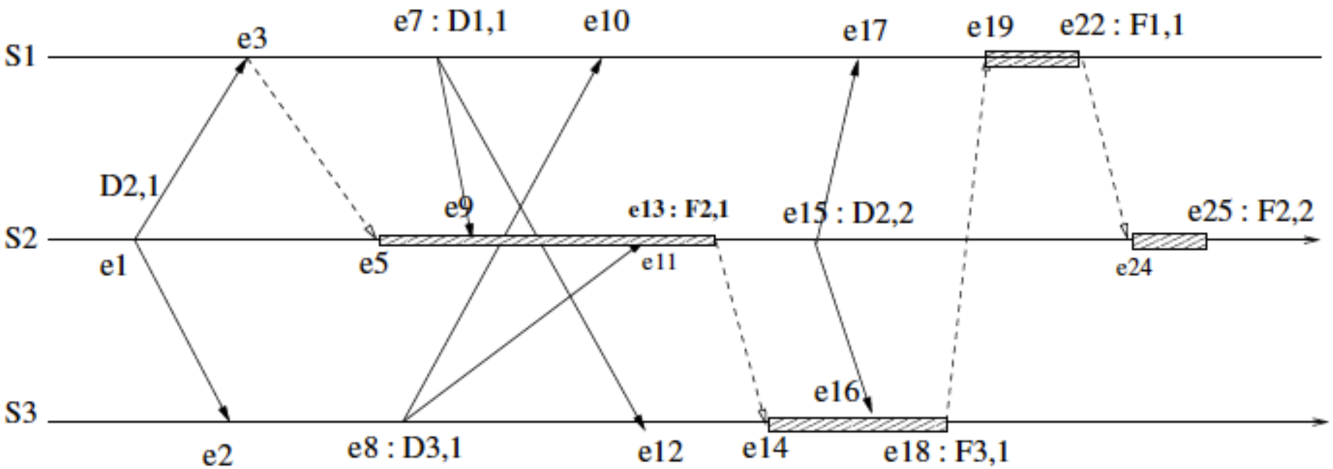


- Donnez pour chaque site et à chaque evt :
  - l'état des variables **`dedansSC`**, **`estampille`** et **`jeton_present`**
  - des tableaux **`demande`** et **`jeton`**
  - l'action effectuée

```
//Demande d'Entree en SC sur un site i
estampille ++;
requete.emetteur = i; requete.date = estampille;//dernière tentative d'accès maj
demande[i] = estampille;
diffuser(requete);
Si (!jeton_present) attendre(jeton);
dedansSC = 1;
jeton_present = 1;
     => SC
//Sortie de la SC
jeton[i] = estampille; //dernière possession maj
dedansSC = 0;
for (j=i+1;j=i;j++%N) // chacun son tour (pas de priorité)
Si (demande[j] > jeton[j]) && (jeton_present) // la demande de j est + récente que son dernier accès en SC
{jeton_present = 0; envoyer(jeton, j);}
```

```
//Traitement d'une Requete, requete
int k = requete.emetteur;
demande[k] = max(demande[k], requete.date);
Si (jeton_present) && (!dedansSC)
=> code ~ sortie de la SC
=> sinon rien (pas de jeton)
```

A vous de jouer




```
//Demande d'Entree en SC sur un site i
estampille ++;
requete.emetteur = i; requete.date = estampille;//dernière tentative d'accès maj
demande[i] = estampille;
diffuser(requete);
Si (!jeton_present) attendre(jeton);
dedansSC = 1;
jeton_present = 1;
     => SC
//Sortie de la SC
jeton[i] = estampille; //dernière possession maj
dedansSC = 0;
for (j=i+1;j=i;j++%N)  // chacun son tour (pas de priorité)
Si (demande[j] > jeton[j]) && (jeton_present) // la demande de j est + récente que son dernier accès en SC
{jeton_present = 0; envoyer(jeton, j);}
```

```
//Traitement d'une Requete, requete
int k = requete.emetteur;
demande[k] = max(demande[k], requete.date);
Si (jeton_present) && (!dedansSC)
=> code ~ sortie de la SC
=> sinon rien (pas de jeton)
```


UNIVERSITÉ DE STRASBOURG

Avec un jeton, ça marche !

- Démonstration
  - ✓ Exclusion ?
    - ➡ La variable `jeton_présent` est *vraie* en un seul site à la fois
  - ✓ Progression ? (pas de rétention du jeton)
    - ➡ Dès qu'un site a fini, soit il transmet sur une demande mémorisée soit sur une requête
  - ✓ Attente bornée ?
    - ➡ grâce au parcours circulaire du jeton, pas de famine

Avec liste d'attente répartie

- Basé sur une horloge logique
  - ✓ Chaque site S_i gère un tableau Fi[N]
    - ➡ Fi[j] : file de messages reçus de S_j
    - ➡ message : `(type={hors_SC,requete_SC,Ack},date,id_site)`

```
//Entree_en_SC par un site Si
Diffusion(requete_SC,hi,i);
Fi[i]←(requete_SC,hi,i);
hi++;
Attendre que ∀j != i | Fi[i].date < Fi[j].date // .date => estampille (date, id_site)

    => SC

//Sortie_de_SC
Diffusion(hors_SC,hi,i);
Fi[i]←(hors_SC, hi, i);
hi++;
```

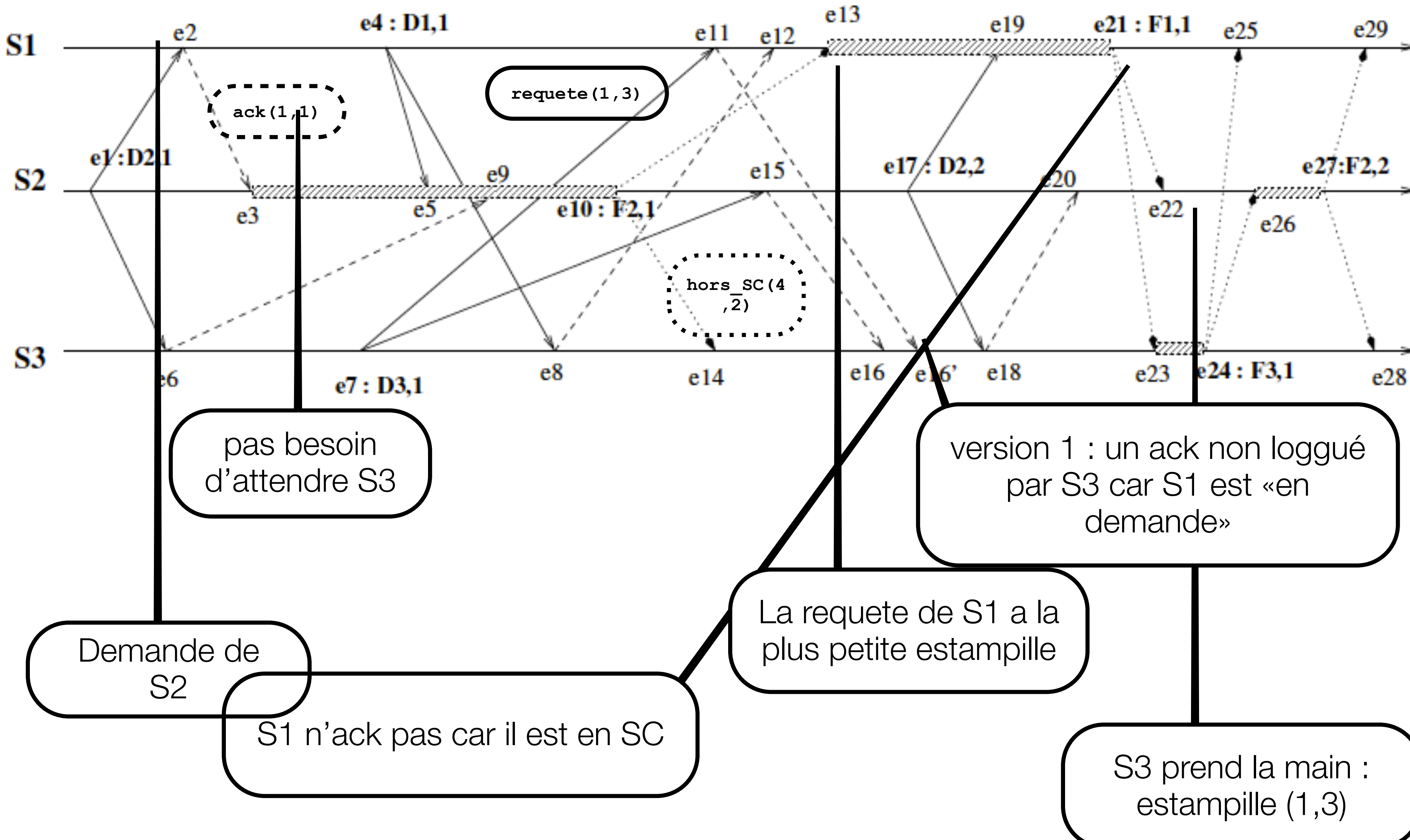
```
//Réception d'un msg(type,h,j)

hi = max(h,hi)+1;
switch (type) in {
case Hors_SC : Fi[j]←(Hors_SC, h, j);
// puis 2 versions possibles ...}
```

Avec liste d'attente répartie

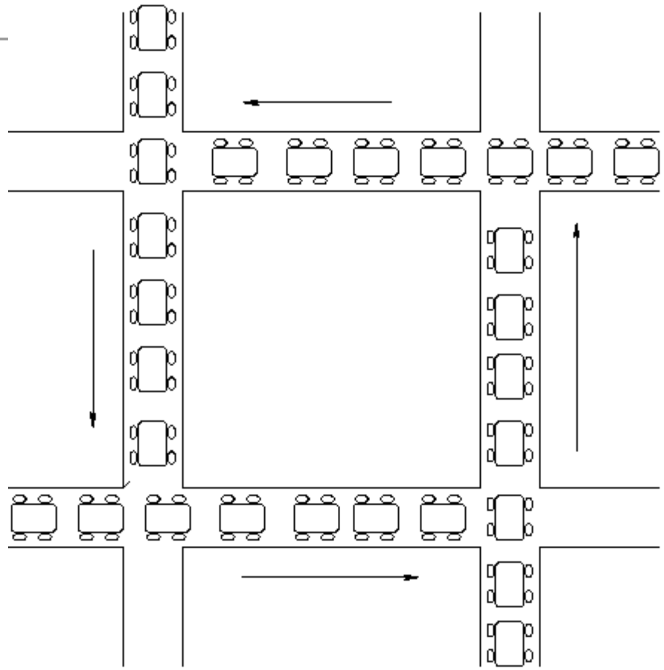
- 2 versions possibles dans le traitement des msgs
  - ✓ On loggue/envoie l'**`Ack`** vers j ssi son dernier msg n'est pas une **`requete_SC`**
    - ➡ envoi automatique mais refus de logguer l'**Ack**
    - ➡ refus d'envoie de l'**Ack** mais logguage automatique

```
//Première version
Fi[j] ← (requete_SC, h, j)
envoi(Ack, hi, i) à Sj }
case Ack : If (Fi[j].type != Requete_SC) : Fi[j] ← (Ack, h, j)


//Seconde version
Fi[j] ← (requete_SC, h, j)
If (Fi[i].type != Requete_SC) : envoi(Ack, hi, i) à Sj }
case Ack : Fi[j] ← (Ack, h, j)
```

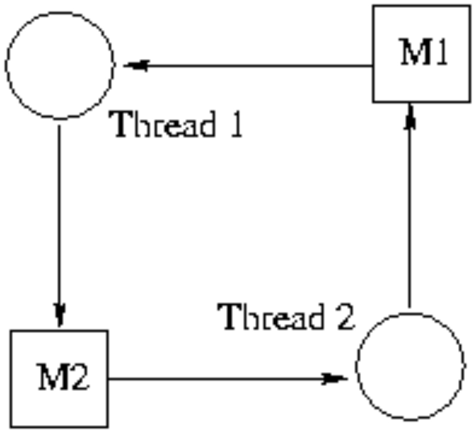
Avec liste d'attente répartie



Partage & Blocage



- Blocage simple
  - ✓ Attente finie d'une ressource
- Inter-Blocage (*deadlock*)



- Trois conditions :
  - ✓ exclusion mutuelle
  - ✓ relation cyclique (un proc disposant d'une ressource peut en demander une autre)
  - ✓ non réquisition (on ne peut libérer une ressource acquise)

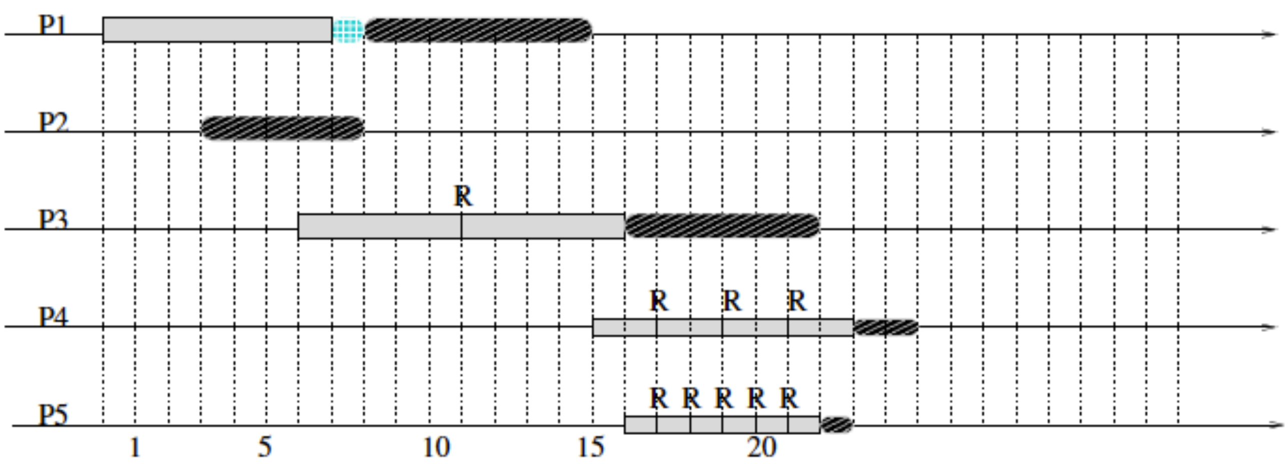
Partage & Blocage

- Trois (quatre) solutions pour venir à bout du problème
  - ✓ Prévention (pro-actif) ? supprimer les cycles de dépendances !
  - ✓ Evitement (ré-actif) ? voir dans le futur...
  - ✓ Détection puis reprise ? on détecte une *non terminaison* puis on «casse» le cycle
  - ✓ Technique de l'autruche ?

Prévention

- A quel niveau ?
  - ✓ Ressources : on leur affecte des niveaux
    - ➡ un process détenant un niveau *n* ne peut plus demander des niveaux <= *n*
    - ➡ à tout instant il existe un process non bloqué (au pire, celui détenant le + grand numéro)
    - ➡ impossible de définir ces niveaux de manière efficace ...
  - ✓ Processus : définir un ordre de priorité stricte inter-proc.
    - ➡ ex : estampille datant la création du proc.
    - ➡ un proc. P_i peut attendre les ressources détenus par un autre P_j ssi $P_i < P_j$ (ou le contraire)
    - ➡ deux options :
      - ➡ «attendre ou crever»
      - ➡ «blesser ou attendre»

Prévention



➡ «attendre ou crever»

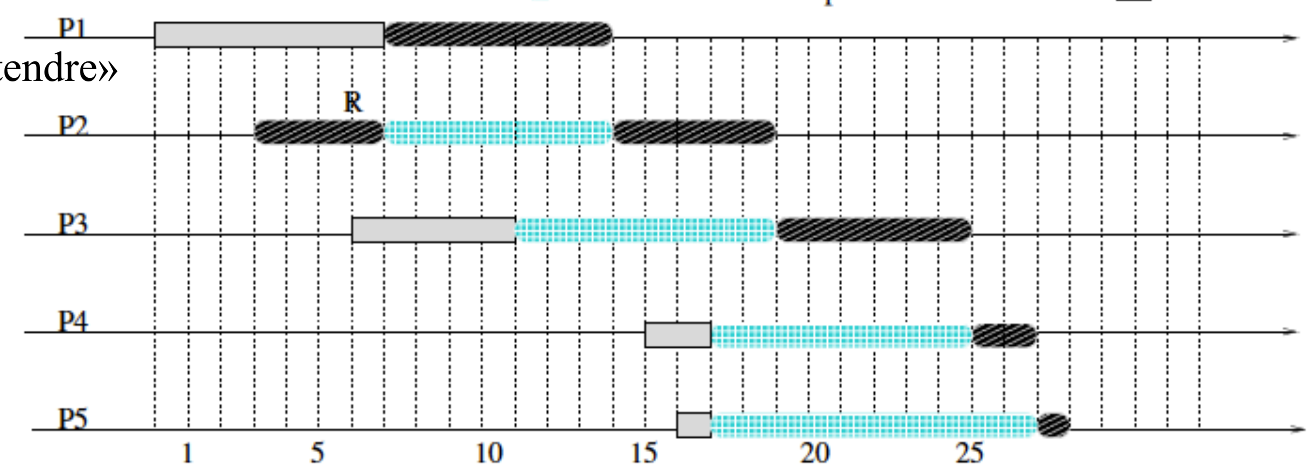
Proc	D_c	D_d	D_u
1	0	7	7
2	3	0	5
3	6	5	6
4	15	2	2
5	16	1	1

Dc : estampille création
Dd : date demande / Dc
Du : temps d'utilisation

➡ 25 sec, 1 sec d'attente, 9 reprises, tps CPU 52 sec

Attente   Dispose de la ressource   Travaille sans ressource

➡ «blesser ou attendre»



➡ 28 sec, 33 sec d'attente, 1 reprise, tps CPU 40 sec

Evitement : quelques notations & définitions

- ✓ Soit un système comportant *m* ressources $R_1,..,R_j,...,R_m$ avec N_j unités offertes pour R_j
  - ➡ capacité du système : $(N_1,...,N_j,...,N_m)$
  - ➡ Soit $S = P_1,\ldots,P_i,...,P_n$ un ensemble de *n* proc. indépendants
  - ➡ chaque proc. P_i peut être subdivisé en k_i sous tâches séquentiels $T_i(1),...,T_i(k_i)$
- ✓ A chaque tâche $T_i(t)$ on associe un début $d_i(t)$ et une fin $f_i(t)$
  - ➡ On *demande* des ressources R_j en début de tâche, on les *libère* à la fin !

processus : P_i	tâche : $T_i(t)$	$demande_i(t)$	$libere_i(t)$
P_1	$T_1(1)$	(1,2)	(0,1)
	$T_1(2)$	(2,0)	(3,1)
P_2	$T_2(1)$	(1,1)	(0,0)
	$T_2(2)$	(2,2)	(3,3)
P_3	$T_3(1)$	(1,0)	(1,0)

$T_1(2)$ demande deux unités de + de R1 et aucune unité en + de R2

$T_1(2)$ libère tout !

- ➡ Deux ressources (j=2, avec capacité (3,3)) et trois processus (i=3)

UNIVERSITÉ DE STRASBOURG

Evitement : formalisation *(procs : i->n, ressources : j->m, evt : l->p, sous-tâche : t->k$_i$)*

- ✓ Comportement d'un système :
  - ➡ a_l : date d'un evt. de début ou fin d'un tache
  - ➡ $p = 2(k_1 + ... + k_n)$: le nb total d'evt
  - ➡ w = $a_1,..., a_l,..., a_p$: un scénario comportemental possible
  - ➡ $Dispo(l) = (dispo_1(l),...,dispo_m(l))$: vecteur des ressources dispo après a_l
- ✓ Etat d'un système : à chq. evt. a_l est associé un état s_l
  - ➡ une matrice $n \times m$ $besoin_{i,j}(l)$: P_i demande $besoin_{i,j}(l)$ unités de R_j de + après a_l
    - ➡ P_i peut lancer sa tâche ssi $besoin_{i,j}(l) \leq Dispo(l)\ \forall j$
  - ➡ une matrice $n \times m$ $detenu_{i,j}(l)$: P_i détient $detenu_{i,j}(l)$ unités de R_j après a_l
    - ➡ $dispo_j(l) = N_j - \sum_i detenu_{i,j}(l)$
- ✓ Calcul pour les transitions $s_{l-1} \rightarrow s_l$:
  - ➡ si $a_l = d_i(t)$: $besoin_{i,j}(l) = 0$ *(satisfaction !)* & $detenu_{i,j}(l) = detenu_{i,j}(l-1) + demande_i(t)$
  - ➡ si $a_l = f_i(t)$: $besoin_{i,j}(l) = demande_i(t+1)$ & $detenu_{i,j}(l) = detenu_{i,j}(l-1) - libere_i(t)$

Evitement : formalisation *(procs : i->n, ressources : j->m, evt : l->p, sous-tâche : t->k_i)*

✓ Comportement d'un système :

➡ a_l : date d'un evt. de début ou fin d'un tache

▸ Donnez l'évolution des matrices *besoin* et *détenu* pour le scénario suivant :

$w=d_3(1)d_1(1)f_1(1)f_3(1)d_1(2)f_1(2)d_2(1)f_2(1)d_2(2)f_2(2)$

processus : P_i	tâche : $T_i(j)$	$demande_i(j)$	$libere_i(j)$
P_1	$T_1(1)$ $T_1(2)$	(1,2) (2,0)	(0,1) (3,1)
P_2	$T_2(1)$ $T_2(2)$	(1,1) (2,2)	(0,0) (3,3)
P_3	$T_3(1)$	(1,0)	(1,0)

✓ Eta

rès a_l

➡ $dispo_j(l) = N_j - \sum_i detenu_{i,j}(l)$

✓ Calcul pour les transitions $s_{l-1} \rightarrow s_l$:

➡ si $a_l = d_i(t)$: $besoin_{i,j}(l) = 0$ *(satisfaction !)* & $detenu_{i,j}(l) = detenu_{i,j}(l-1) + demande_i(t)$

➡ si $a_l = f_i(t)$: $besoin_{i,j}(l) = demande_i(t+1)$ & $detenu_{i,j}(l) = detenu_{i,j}(l-1) - libere_i(t)$

Evitement : formalisation *(procs : i->n, ressources : j->m, evt : l->p, sous-tâche : t->k_i)*

- ✓ Comportement d'un système :
  - ➡ a_l : date d'un evt. de début ou fin d'un tache
  - ➡ $p = 2(k_1 + ... + k_n)$: le nb total d'evt
  - ➡ $w = a_1, ..., a_l, ..., a_p$: un scénario comportemental possible
  - ➡ $Dispo(l) = (dispo_1(l), ..., dispo_m(l))$: vecteur des ressources dispo après a_l
- ✓ Etat d'un système : à chq. evt. a_l est associé un état s_l
  - ➡ une matrice $n \times m$ $besoin_{i,j}(l)$: P_i demande $besoin_{i,j}(l)$ unités de R_j de + après a_l
    - ➡ P_i peut lancer sa tâche ssi $besoin_{i,j}(l) \leq Dispo(l)\ \forall j$
  - ➡ une matrice $n \times m$ $detenu_{i,j}(l)$: P_i détient $detenu_{i,j}(l)$ unités de R_j après a_l
    - ➡ $dispo_j(l) = N_j - \sum_i detenu_{i,j}(l)$
- ✓ Calcul pour les transitions $s_{l-1} \rightarrow s_l$:
  - ➡ si $a_l = d_i(t)$: $besoin_{i,j}(l) = 0$ *(satisfaction !)* & $detenu_{i,j}(l) = detenu_{i,j}(l-1) + demande_i(t)$
  - ➡ si $a_l = f_i(t)$: $besoin_{i,j}(l) = demande_i(t+1)$ & $detenu_{i,j}(l) = detenu_{i,j}(l-1) - libere_i(t)$

Evitement - exemple

Une suite d'evenements

$$w = d_3(1)\, d_1(1)\, f_1(1)\, f_3(1)\, d_1(2)\, f_1(2)\, d_2(1)\, f_2(1)\, d_2(2)\, f_2(2)$$

$$s_0 = \left[\begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right];\ s_1 = \left[\begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \right];\ s_2 = \left[\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \right];$$

$Dispo(1)=(2,3)$ $Dispo(2)=(1,1)$

$$s_3 = \left[\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \right];\ s_4 = \left[\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right];\ s_5 = \left[\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 3 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right];$$

$Dispo(3)=(1,2)$ $Dispo(4)=(2,2)$ $Dispo(5)=(0,2)$

$$s_6 = \left[\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right];\ s_7 = \left[\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \right];\ s_8 = \left[\begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \right];$$

$Dispo(6)=(3,3)$ $Dispo(7)=(2,2)$ $Dispo(8)=(2,2)$

$$s_9 = \left[\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 3 & 3 \\ 0 & 0 \end{bmatrix} \right];\ s_{10} = \left[\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right]$$

$Dispo(9)=(0,0)$ $Dispo(10)=(3,3)$

besoin *detenu*

P_1	$T_1(1)$	(1,2)	(0,1)
	$T_1(2)$	(2,0)	(3,1)
P_2	$T_2(1)$	(1,1)	(0,0)
	$T_2(2)$	(2,2)	(3,3)
P_3	$T_3(1)$	(1,0)	(1,0)

$f_2(1) = s8$:
T_2 (2) demande encore (2,2) mais ne libère rien

Evitement & Blocage : définition d’un inter-blocage

- *Un processus P_i* est dit bloqué ssi ∃j i.e : *$besoin_{i,j}(l)$ > $dispo_j$ (l)*
- *L’état s_l est un inter-blocage si :*
  - ∃ un sous ensemble *D* de processus bloqués (en attente d’au moins une ressource)
  - *pour chq P_i, $i \in D$*, ∃ au moins une ressource R_j i.e *$besoin_{i,j}(l) > dispo_j(l) + \sum_{i \notin D} detenu_{i,j}(l)$*

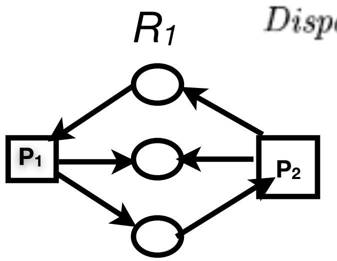
$$w = d_1(1)\, f_1(1)\, d_2(1)\, f_2(1)\, d_1(2)\, f_1(2)\, d_2(2)\, f_2(2)\, d_3(1)\, f_3(1)$$

$$s_0 = \left[\begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} , \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right] ;\ s_1' = \left[\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} , \begin{bmatrix} 1 & 2 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right] ;\ s_2' = \left[\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} , \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right] ;$$

$Dispo(1) = (2,1)$ $\qquad$ $Dispo(2) = (2,2)$

$$s_3' = \left[\begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} , \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \right] ;\ s_4' = \left[\begin{bmatrix} 2 & 0 \\ 2 & 2 \\ 1 & 0 \end{bmatrix} , \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \right] ;$$

$Dispo(3) = (1,1)$ $\qquad$ $Dispo(4) = (1,1)$



Blocage *D* = {1,2} ($d_1(2)$) :

$T_1(2)$: $besoin_{1,1}(4) = 2 > dispo_1(4) = 1$
$T_2(2)$: $besoin_{2,1}(4) = 2 > dispo_1(4) = 1$
$3 \notin D$: $detenu_3(4) = (0,0)$

P_1	$T_1(1)$	(1,2)	(0,1)
	$T_1(2)$	(2,0)	(3,1)
P_2	$T_2(1)$	(1,1)	(0,0)
	$T_2(2)$	(2,2)	(3,3)
P_3	$T_3(1)$	(1,0)	(1,0)

Evitement - Etat sûr

- ✓ Soit un comportement $w=a_1a_2...a_l$ partiel du système
  - ➡ L'état s_l est dit <u>sûr</u> s'il existe un comportement valide du système commençant par w

$$w = d_1(1)\, f_1(1)\, d_2(1)\, f_2(1)\, d_1(2)\, f_1(2)\, d_2(2)\, f_2(2)\, d_3(1)\, f_3(1)$$

$$w = d_1(1)\, f_1(1)\, d_3(1)\, f_3(1)\, d_1(2)\, f_1(2)\, d_2(1)\, f_2(1)\, d_2(2)\, f_2(2)$$

$$s_0 = \left[\begin{bmatrix}1 & 2\\ 1 & 1\\ 1 & 0\end{bmatrix}, \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\right];\ s_1' = \left[\begin{bmatrix}0 & 0\\ 1 & 1\\ 1 & 0\end{bmatrix}, \begin{bmatrix}1 & 2\\ 0 & 0\\ 0 & 0\end{bmatrix}\right];\ s_2' = \left[\begin{bmatrix}2 & 0\\ 1 & 1\\ 1 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 0 & 0\\ 0 & 0\end{bmatrix}\right];$$

Dispo(1)=(2,1) *Dispo(2)=(2,2)*

$$s_3' = \left[\begin{bmatrix}2 & 0\\ 0 & 0\\ 1 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 1 & 1\\ 0 & 0\end{bmatrix}\right];\ s_4' = \left[\begin{bmatrix}2 & 0\\ 2 & 2\\ 1 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 1 & 1\\ 0 & 0\end{bmatrix}\right];$$

Dispo(3)=(1,1) *Dispo(4)=(1,1)*

$f_2(1)$

$d_3(1)$

$$s_4'' = \left[\begin{bmatrix}2 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 1 & 1\\ 1 & 0\end{bmatrix}\right] \xrightarrow{f_2(1)} s_5'' = \left[\begin{bmatrix}2 & 0\\ 2 & 2\\ 0 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 1 & 1\\ 1 & 0\end{bmatrix}\right]$$

$f_3(1)$

$$s_5'' = \left[\begin{bmatrix}2 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 1 & 1\\ 0 & 0\end{bmatrix}\right] \xrightarrow{f_2(1)} \left[\begin{bmatrix}2 & 0\\ 2 & 2\\ 0 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\ 1 & 1\\ 0 & 0\end{bmatrix}\right]$$

A partir de là, toutes les voies possibles menent à un inter-blocage...

L'algo du banquier

- ✓ Pour passer de s_{l-1} à s_l, on vérifie que s_l est sûr :
  - ➡ si s_l est une libération => OK
  - ➡ si s_l est une demande => test : on cherche une séquence de complétion valide
- ✓ Problème : on ne connait pas le détail des demandes/libérations des sous tâches

  => on imagine le pire cas, i.e., les demandes MAX_i d'un proc P_i

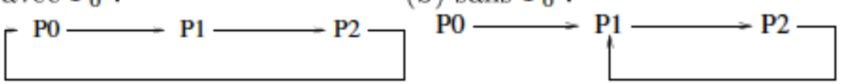
L'algo :

- ✓ Si $s_{l-1} \rightarrow s_l$ est une libération, la transition est validée
- ✓ Sinon ($s_{l-1} \rightarrow s_l$ est une demande) :
  - ➡ on calcule l'état s_l et on construit l'état virtuel au pire associé t_l
  - ➡ $t_l = (besoin^t(l), detenu(l))$ avec $besoin^t_i(l) = \begin{cases} MAX_i - detenu_i(l) \text{ si } besoin_i(l)+detenu_i(l) > 0 \\ 0 \text{ sinon} \end{cases}$
    - ➡ si t_l n'est pas un inter-blocage => on valide la transition
    - ➡ sinon on rejette la demande :((mais on ne peut pas conclure...)

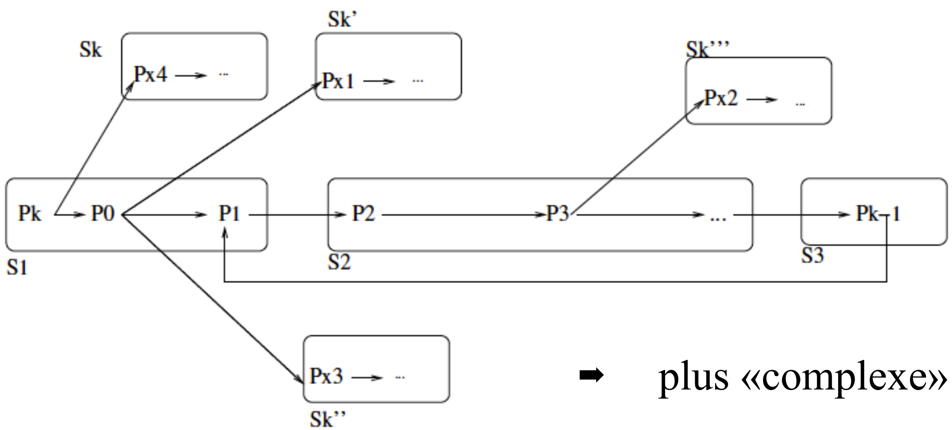
Détection

- ✓ Un processus bloqué initie une détection
- ✓ Chaque site dispose d’un contrôleur de ressource
    - ➡ connait les ressources sollicitées localement (sur des proc. locaux ou externes)
    - ➡ mémorisation des dépendances de «ses» processus
- ✓ $P_i \rightarrow P_j$: P_i dépend de P_j, P_i attend les ressources détenus par P_j

(a) avec P_0 : (b) sans P_0 :



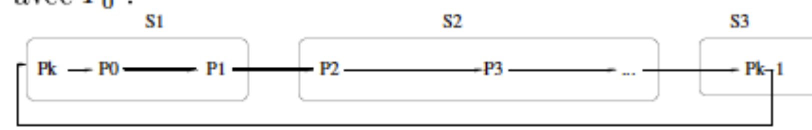
➡ En local



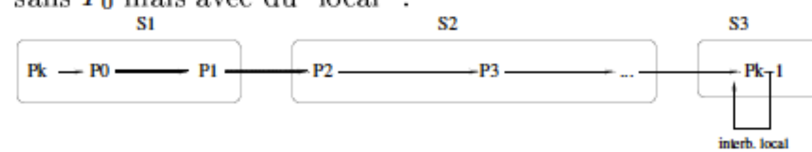
➡ plus «complexe»

➡ sur plusieurs sites

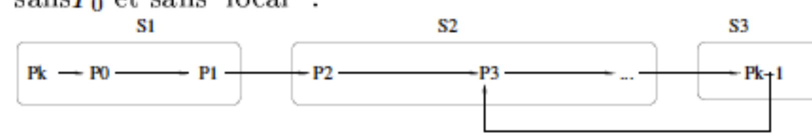
(a) avec P_0 :



(b) sans P_0 mais avec du “local” :



(c) sansP_0 et sans “local” :



Algorithme de détection

- ✓ Soit P_o le proc. initiant la demande, un proc. réagit comme suit :

- ✓ Blocage local :
  - ➡ si «je suis l'initiateur» => l'application est inter-bloquée localement => FIN
  - ➡ sinon : inter-blocage «multi-sites» => proclamation du blocage
- ✓ Pas blocage local :
  - ➡ Sans dépendance avec des sites distants
    - ➡ si «je suis l'initiateur» => l'application n'est pas bloquée => FIN
    - ➡ sinon => l'application n'est pas bloquée *a priori* => répondre «non bloqué» au demandeur
  - ➡ Sinon : émission vers les sites distants d'une demande de détection
- ✓ A la réception d'une demande de détection, sur un site quelconque :
  - ➡ si initiée par lui-même : l'application est bloquée => FIN
  - ➡ sinon s'il a détecté un inter-blocage => proclamation du blocage
  - ➡ sinon alors qu'il n'a pas détecté un inter-blocage => (re)lance l'algo de détection
- ✓ A la réception d'une proclamation : blocage => FIN (si je suis le demandeur) ou propagation au demandeur
- ✓ A la réception d'un msg «non bloqué» pour ttes les demandes : non bloqué => FIN
  - ✓ (si je suis le demandeur) ou propagation au demandeur

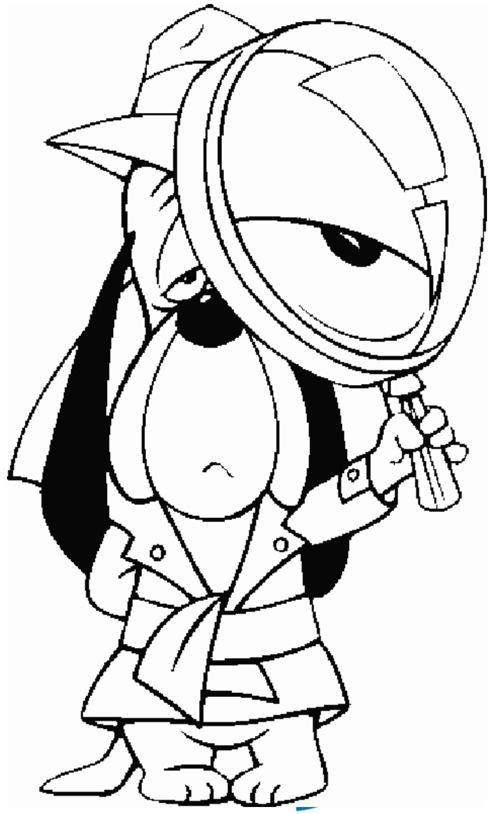
Pseudo-code de détection

- Soit P_i les procs, S_r les sites et C_r les contrôleurs de ces sites
- Deux fonctions de base :
  - `TEST_LOCAL(Pi,Cr)` → `BOOLEAN` /* return TRUE if a cycle occur, FALSE otherwise */
  - `DEPEND(Pi,Cr)` → `{Pj}` /* list of process that depends directly or not from P_i (at the border too!) */

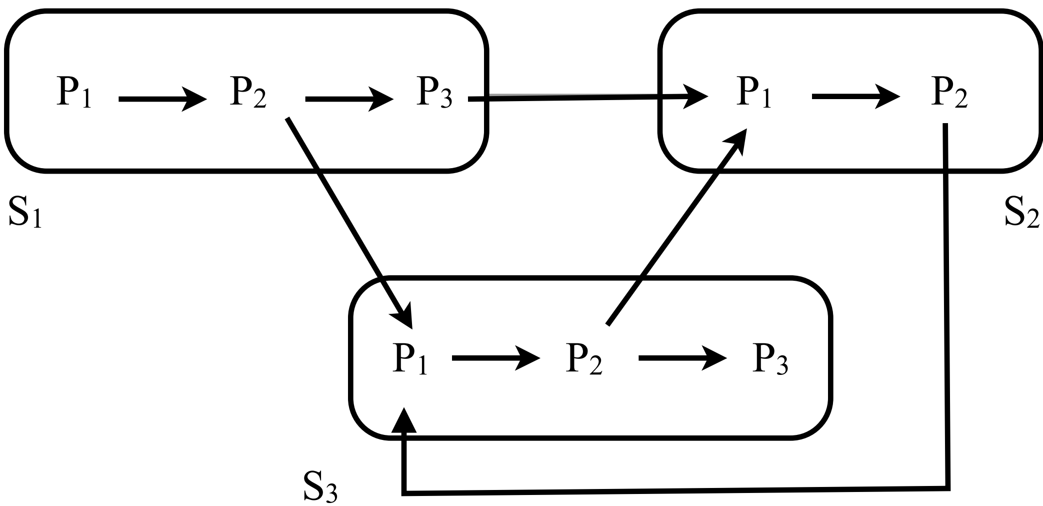
```
//Le proc. Pi initie une détection sur le site Sr
Detect_interblocage(i){
if(TEST_LOCAL(Pi,Cr))
return INTERBLOCAGE;
for all (j,k)∈ DEPEND(Pi,Cr)× DEPEND(Pi,Cr) | Pi,Pj ∈ Sr && Pk ∉ Sr
send(MSG_GLOBAL(i,j,{n1,n2,...})) towards Pk; /* n1,n2,... : dépendances entre Pi et Pj */
while(not_all_responses_received)
{wait(response); if(response == TRUE) return INTERBLOCAGE;} return ATTENTE;}


//Réception d'un msg TEST_GLOBAL(i,j,{n1,n2,...}) par (Pk,Sp)
Receive(TEST_GLOBAL(i,j,{n1,n2,...})){
if(!ATTENTE) return FALSE to Pi;
if(k ∈ {n1,n2,...}) return TRUE to Pi;
if(TEST_LOCAL(Pk,Cp)) return TRUE to Pi;
for all (a,b)∈ DEPEND(Pk,Cr)× DEPEND(Pk,Cr) | Pa,Pk ∈ Sp && Pb ∉ Sp
send(MSG_GLOBAL(k,a,{n1,n2,...}U{n'1,n'2,...})) towards Pb;
while(not_all_responses_received)
{wait(response); if(response == TRUE) return TRUE to Pi;} return FALSE to Pi;}
```

Détection, qqs questions...

- ✓ **Qui lance la détection ?** problème équivalent terminaison, élection, ...
  - ✓ => *un classique*
    - ➡ cas 1 : un proc particulier surveille le bon déroulement => comment sait-il ?
    - ➡ cas 2 : soit on laisse la main aux procs en attente => +sieurs détection simultanée
    - ➡ cas 3 : les controleurs (/ site)
- ✓ **Pour quel coût ?**
  - ✓ au pire, un tour complet des sites...
    - ➡ cas 1 : une seule occurence (mais coût de surveillance)
    - ➡ cas 2 : au pire, une occurence par proc.
    - ➡ cas 3 : au pire, une ocurence par site.
- ✓ **Qu'est-ce qu'on fait après ?**
  - ➡ pas de solution générique...
  - ➡ qui tuer ? pour libérer quoi ?

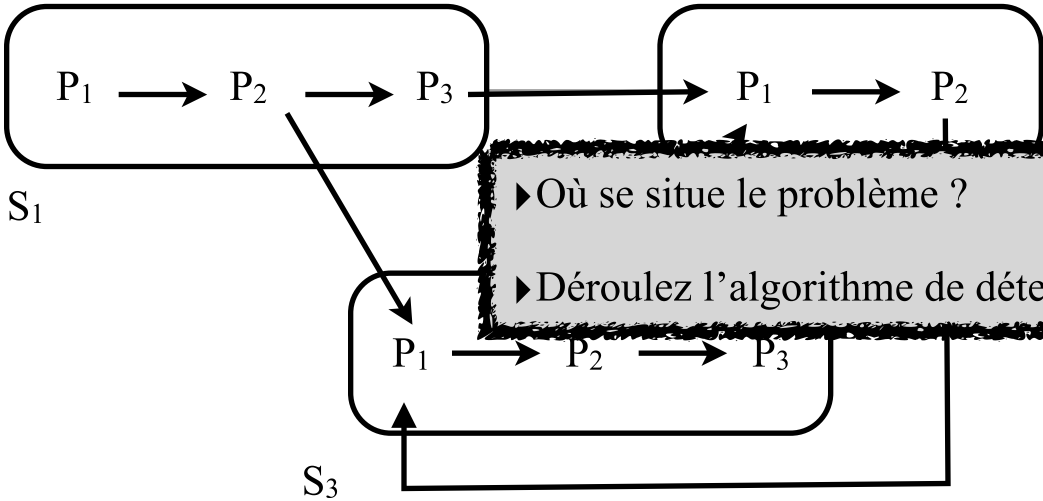


A vous de jouer




```
Detect_interblocage(i){//Le proc. P_i initie une détection sur le site S_r
if(TEST_LOCAL(P_i,C_r))
return INTERBLOCAGE;
for all (j,k)∈ DEPEND(P_i,C_r)×DEPEND(P_i,C_r) | P_i,P_j ∈ S_r && P_k ∉ S_r
send(MSG_GLOBAL(i,j,{n_1,n_2,...})) towards P_k; /* n_1,n_2,... : dépendances entre P_i et P_j */
while(not_all_responses_received)
{wait(response); if(response == TRUE) return INTERBLOCAGE;} return ATTENTE;}
//Réception d'un msg TEST_GLOBAL(i,j,{n_1,n_2,...}) par (P_k,S_p)
Receive(TEST_GLOBAL(i,j,{n_1,n_2,...})){
if(!ATTENTE) return FALSE to P_i;
if(k ∈ {n_1,n_2,...}) return TRUE to P_i;
if(TEST_LOCAL(P_k,C_p)) return TRUE to P_i;
for all (a,b)∈ DEPEND(P_k,C_r)×DEPEND(P_k,C_r) | P_a,P_k ∈ S_p && P_b ∉ S_p
send(MSG_GLOBAL(k,a,{n_1,n_2,...}∪{n'_1,n'_2,...})) towards P_b;
while(not_all_responses_received)
{wait(response); if(response == TRUE) return TRUE to P_i;} return FALSE to P_i;}
```

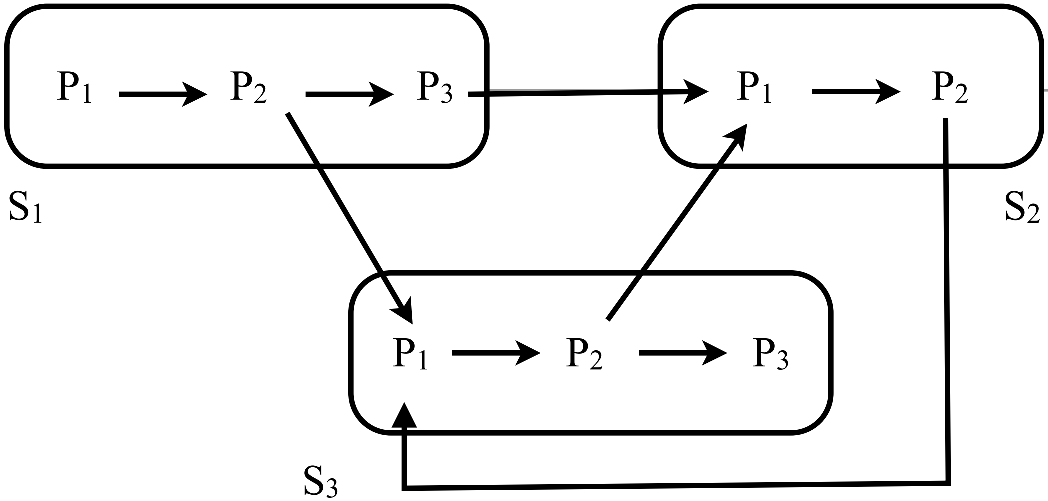
A vous de jouer



- Où se situe le problème ?
- Déroulez l'algorithme de détection si P_2 en S_1 est le site initiateur

```
Detect_interblocage(i){//Le proc. P_i initie une détection sur le site S_r
if(TEST_LOCAL(P_i,C_r))
return INTERBLOCAGE;
for all (j,k)∈ DEPEND(P_i,C_r)×DEPEND(P_i,C_r) | P_i,P_j ∈ S_r && P_k ∉ S_r
send(MSG_GLOBAL(i,j,{n_1,n_2,...})) towards P_k; /* n_1,n_2,... : dépendances entre P_i et P_j */
while(not_all_responses_received)
{wait(response); if(response == TRUE) return INTERBLOCAGE;} return ATTENTE;}
//Réception d'un msg TEST_GLOBAL(i,j,{n_1,n_2,...}) par (P_k,S_p)
Receive(TEST_GLOBAL(i,j,{n_1,n_2,...})){
if(!ATTENTE) return FALSE to P_i;
if(k ∈ {n_1,n_2,...}) return TRUE to P_i;
if(TEST_LOCAL(P_k,C_p)) return TRUE to P_i;
for all (a,b)∈ DEPEND(P_k,C_r)×DEPEND(P_k,C_r) | P_a,P_k ∈ S_p && P_b ∉ S_p
send(MSG_GLOBAL(k,a,{n_1,n_2,...}∪{n'_1,n'_2,...})) towards P_b;
while(not_all_responses_received)
{wait(response); if(response == TRUE) return TRUE to P_i;} return FALSE to P_i;}
```

A vous de jouer




```
Detect_interblocage(i){//Le proc. Pi initie une détection sur le site Sr
if(TEST_LOCAL(Pi,Cr))
return INTERBLOCAGE;
for all (j,k)∈ DEPEND(Pi,Cr)×DEPEND(Pi,Cr) | Pi,Pj ∈ Sr && Pk ∉ Sr
send(MSG_GLOBAL(i,j,{n1,n2,...})) towards Pk; /* n1,n2,... : dépendances entre Pi et Pj */
while(not_all_responses_received)
{wait(response); if(response == TRUE) return INTERBLOCAGE;} return ATTENTE;}
//Réception d'un msg TEST_GLOBAL(i,j,{n1,n2,...}) par (Pk,Sp)
Receive(TEST_GLOBAL(i,j,{n1,n2,...})){
if(!ATTENTE) return FALSE to Pi;
if(k ∈ {n1,n2,...}) return TRUE to Pi;
if(TEST_LOCAL(Pk,Cp)) return TRUE to Pi;
for all (a,b)∈ DEPEND(Pk,Cr)×DEPEND(Pk,Cr) | Pa,Pk ∈ Sp && Pb ∉ Sp
send(MSG_GLOBAL(k,a,{n1,n2,...}∪{n'1,n'2,...})) towards Pb;
while(not_all_responses_received)
{wait(response); if(response == TRUE) return TRUE to Pi;} return FALSE to Pi;}
```