Partie 2: Synchronisation

Objectifs

- Comprendre le problème de l'exclusion mutuelle
  - Ressource critique,
  - Section critique,
  - Solutions logicielles et matérielles pour l'exclusion mutuelle.
- Comprendre le concept de synchronisation entre processus:
  - Techniques de synchronisation,
  - Application sur des problèmes types: Allocateur de ressources, lecteurs/rédacteurs, producteurs/ consommateurs….

Exclusion mutuelle

Introduction (Exemple d'illustration)

- Considérons n processus pour réservation de sièges dans un avion (s'exécutant sur un serveur monoprocesseur).

```
int nbr_sieges= 100; (variable partagée )
Processus réservation Pi (i=1,..,n)
{
if (nbr_sieges>0) {reserver(); nbr_sieges--;}
}
```

Introduction (Exemple d'illustration 2)

- Considérons 2 processus A et B pouvant consulter et modifier une variable commune (compte)

A : versement	B : Retrait
A1: lire (compte, c1); A2: c1=c1+10000; A3: ecrire(c1, compte);	B1: lire (compte, c2); B2: c2=c2-5000; B3: ecrire(c2, compte);

- Quels sont les résultats possibles des exécutions des processus P1 et P2 (en pseudo-parallélisme)
  - On suppose que le compte contient 20000 unités.

Introduction

- Remarque:
  - La variable partagée ne doit pas être manipuler (lecture/écriture) par un processus lorsqu'elle est utilisée par un autre (accès exclusif).
- Solution:
  - Protéger l'accès à la variable tant qu'elle est utilisée par un processus.
  - À la fin de la manipulation, le processus doit libérer la donnée.

Définitions

- Resource : Une ressource désigne toute entité dont a besoin un processus pour s'exécuter.
  - La ressource peut être matérielle comme le processeur ou logicielle comme une variable.
- Ressource critique: Une ressource ne pouvant être utilisé que par un seul processus à la fois.
  - Exemples: Processeur, Imprimante, …

Définitions

- Section critique SC : c'est la partie de programme dans laquelle se font des accès à une ressource critique (partagée).
- Exclusion mutuelle : a un instant donné, au plus un seul processus est entrain d'exécuter sa section critique
  - Deux processus (dépendants) ne peuvent pas être en même temps en section critique,
  - Ce qui implique de rendre indivisible (atomique) la séquence d'instruction composant la section critique.

Structure d'un processus qui exécute une section critique

- La SC est précédée par un prélude (protocole d'entrée en section critique): verrouiller la SC.
- La SC est suivie d'un postlude (protocole de sortie de la section critique): déverrouiller la SC

```
< Variables communes >
Processus Pi (i=1,...N)
        Section non critique
        Protocole d'entrée en SC
        < Section critique >
        Protocole de sortie de la SC
        Section non critique
```

Réalisation de l'exclusion mutuelle: Hypothèses de Dijkstra

- Les vitesses relatives des processus sont quelconques et inconnues.
  - Mais, on suppose que tout processus sort de sa section critique au bout d'un temps fini.
- L'ordre d'accès à la ressource est quelconque (ordre d'exécution de la section critique).
- On suppose que les instructions d'affectation et de teste sur une variable sont atomiques (indivisibles)
- La section critique doit respecter les contraintes suivantes :

Contraintes de l'exclusion mutuelle

a. L'exclusion mutuelle: À tout instant un processus au plus exécute la SC de son programme.
b. Absence d'interblocage: Si un processus est bloqué en dehors d'une SC, ce blocage ne doit pas empêcher l'entrée d'un autre processus en SC.
c. Absence de blocage: Si un processus n'est pas intéressé par la section critique, ceci ne doit pas empêcher un autre d'y accéder.
d. Absence de famine: Un processus qui a demandé à entrer en SC ne doit pas attendre indéfiniment.

Réalisation d'une section critique

- Il existe deux façons pour réaliser une section critique, ce sont les outils de synchronisation :
  - Logiciels ou Matériels.

a. Solutions logicielles :

- Lorsqu'un processus désire entrer dans une SC, il doit être mis en attente si la SC est non libre.
- Lorsqu'un processus quitte la SC, il doit le signaler aux autres processus.

a. Solutions logicielles

L'attente peut être :

- Active : le protocole d'entrée en SC est une boucle
  - La condition est un teste qui porte sur des variables indiquant la présence ou non d'un processus en SC.
- Non active : le processus passe dans l'état endormi et ne sera réveillé que lorsqu'il sera autorisé à entrer en SC.

a. Solutions logicielles

- Les solutions logicielles cités dans les diapos suivants reposent sur le principe de l'attente active:

> Tant que <condition indique SC non libre> faire <rien>
> < Section critique>
> Modifier condition pour refléter que la SC est libre

- Problème : Consommation inutile de temps CPU
  - On peut remplacer <rien> par: sleep(court délai)

Solution 1 (Naïve, incorrecte) : Avec une variable booléenne.

```
var occupe : booléen ;
occupe :=faux ; /* la section critique est vide */

processus Pi (i=1,…, n)
    …
    tant que (occupe) faire <rien>
    occupe :=vrai ;
    < Section critique >
    occupe :=faux ; /*protocole de sortie */
    …
```

Solution 1 (Naïve, incorrecte) : Avec une variable booléenne.

- La 1ère contrainte de l'exclusion mutuelle n'est pas vérifiée.
- La variable « occupe » elle-même est une ressource critique!
  - Elle doit être utilisée en exclusion mutuelle.

Solution 2: Alternance (entre 2 processus)

```
var tour : 0..1; /* tour est initialisée à 0 ou 1*/

Processus i (i=0,1)
    …
    tant que tour ≠ i faire <rien>
    <section critique>
    tour=  i + 1 mod 2; /*protocole de sortie */
    …
```

Solution 2: Alternance (entre 2 processus)

- Cette solution garantit qu'un seul processus à la fois peut se trouver en section critique.
- Alternance obligatoire
  - La solution ne respecte pas la propriété « Absence de blocage » : Les processus ne peuvent entrer en section critique qu'à tour de rôle
- Deux processus (généralisable à plus)

Solution 2: Alternance (entre N processus)

```
var tour : 0..N; /* tour est initialisée à 0 */

Processus i (i=0,..N)
   …
   tant que tour ≠ i faire <rien>
   <section critique>
   tour   i + 1 mod N; /*protocole de sortie */
   …
```

Solution 3 : Avec deux variables booléennes (pour deux processus)

```
var c : tableau[0..1] :booléen ;
c[0]=faux ;
c[1]=faux ;

processus Pi (i=0,1)
    …
    c[i]=vrai ; /* c[i]=vrai → le processus i est dans sa section critique ou demande à y entrer */
    tant que c[1- i ] faire <rien>
    < Section critique >
    c[i]=faux ;
    …
```

Programmes des deux processus

c[0] := faux; c[1] := faux;	
Processus 0	**Processus 1**
… c[0]=vrai ; tant que c[1] faire <rien> < Section critique > c[0]=faux ; …	… c[1]=vrai ; tant que c[0] faire <rien> < Section critique > c[1]=faux ; …

Analyse de la solution 3

- Cette solution garantit l'exclusion mutuelle
  - Preuve par l'absurde: P0 ET P1 en SC
    - P0 en SC $\rightarrow$ c[0]=vrai et c[1]= faux
    - P1en SC $\rightarrow$ c[1]=vrai et c[0]= faux
    - P0 ET P1 en SC $\rightarrow$ (c[0]=vrai et c[0]= faux) et (c[1]=vrai et c[1]= faux): Impossible
- Mais, si les deux processus s'engagent simultanément P0 : c[0]:= vrai; P1 : c[1]:= vrai;
  - Aucun processus ne pourra entrer dans la section critique
  - Cette solution ne respecte pas l'Absence de l'interblocage.

Solution de Dekker (1965) pour deux processus avec 3 variables

- L'algorithme de Dekker combine les 2 solutions précédentes
- Initialisation :
  - tour:= 0 ou 1;
  - c[0]:= faux; c[1]:= faux;
- Demande d'entrée en section critique d'un processus Pi (i=0..1)
  1. c[i] := vrai;
  2. En cas de conflit : c[i]=vrai et c[1-i]=vrai ;

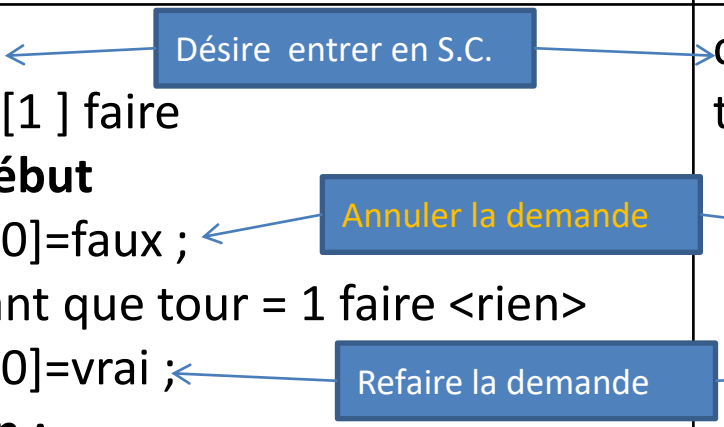
     tester tour :

     si tour=1-i alors Pi annule sa demande d'entrée en section critique: c[i]:= faux; et attend que tour soit égal à i;
- Sortie de la section critique
  - tour:= 1-i; tour est modifié seulement à la sortie de la section critique.
  - c[i]:= faux;

Solution de Dekker

```
var c : tableau [0..1] : booléen ;
tour : 0..1 ;
tour :=0 ; /* peut être initialise a 1 */
c[0]=faux ;
c[1]=faux ;
```

Processus 0	**Processus 1**
c[0]=vrai ;	c[1]=vrai ;
tant que c[1] faire	tant que c[0] faire
début	**début**
c[0]=faux ;	c[1]=faux ;
tant que tour = 1 faire <rien>	tant que tour = 0 faire <rien>
c[0]=vrai ;	c[1]=vrai ;
fin ;	**fin ;**
< Section critique >	< Section critique >
tour =1 ;	tour =0 ;
c[0]=faux ;	c[1]=faux ;



Analyse de la solution de Dekker (raisonnement par l'absurde)

1. L'exclusion mutuelle: Supposons que P0 et P1 soient dans leurs sections critiques.

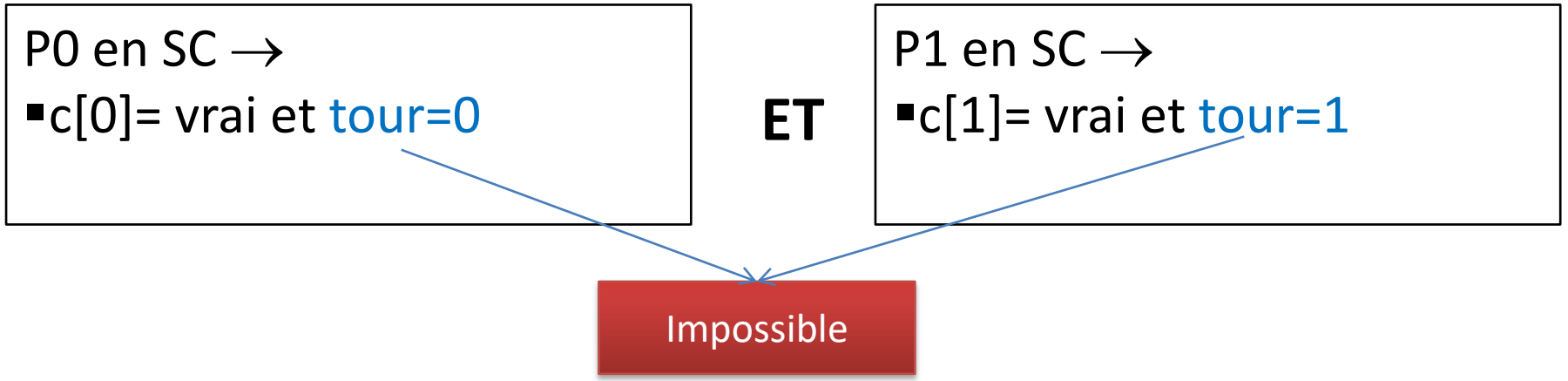
P0 en SC $\rightarrow$
- c[0]=vrai et c[1]=faux (P1 n'est pas intéresser par la SC)

OU
- c[0]=vrai et tour=0

P1 en SC $\rightarrow$
- c[1]=vrai et c[0]=faux (P0 n'est pas intéresser par la SC)

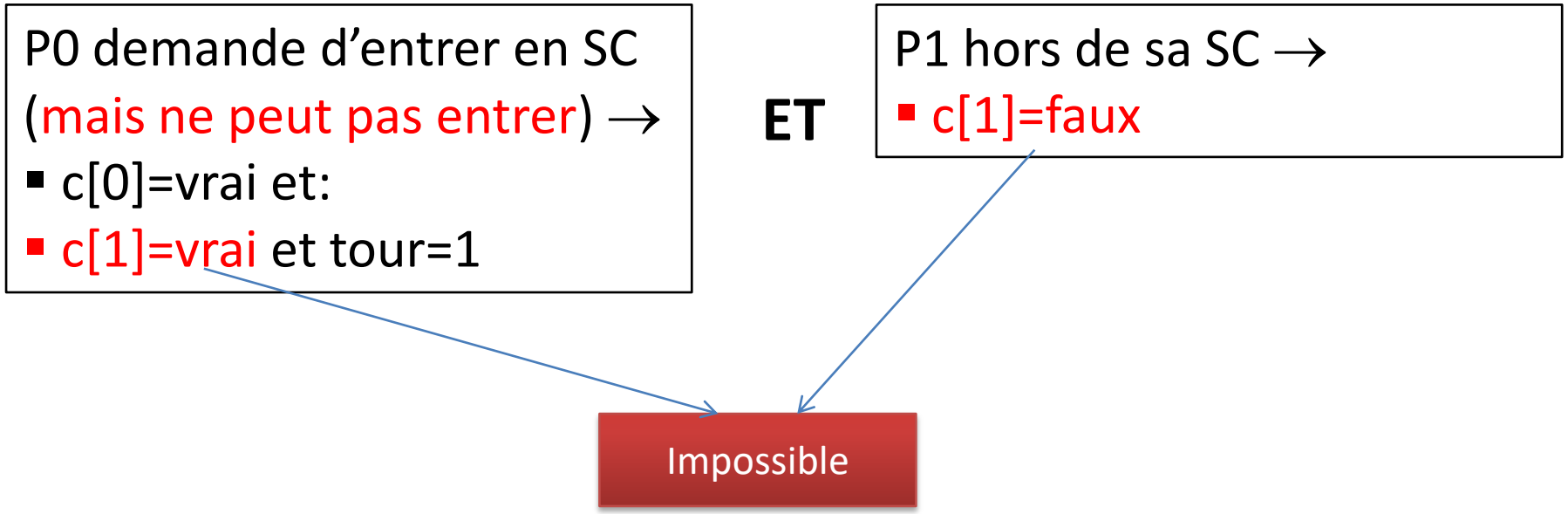
OU
- c[1]=vrai et tour=1

1. L’exclusion mutuelle



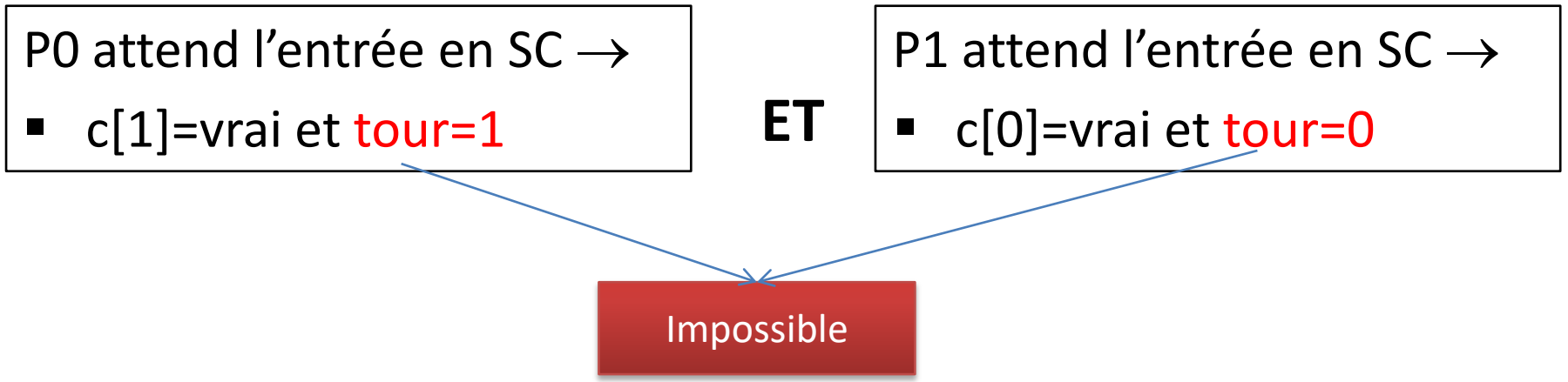
2. Absence de blocage

- Supposons que P1 n'est pas intéressé par la SC (en dehors de sa critique (actif ou bloqué) ou n'existe pas encore) .



3. Absence de l'interblocage

- Supposons que la section critique est libre et P0 et P1 demandent à y entrer; mais aucun d'eux ne peut entrer dans sa SC.



4. Absence de famine

- Supposons que:
  - P0 est en SC.
  - P1 demande la SC mais doit se mettre en attente.
  - P0 termine mais demande une 2ème fois la SC.
- Si P1 est bloqué au niveau de la 2ème boucle alors, c[1]= faux.
  - P0 a la chance d'accéder une 2ème fois en SC avant P1.
  - donc, il y a un problème de famine

Solution de Peterson (1981) pour deux processus avec 3 variables

<table>
<tr><td colspan="2">var c : tableau [0..1] : booléen ;
tour : 0..1 ;
tour :=0 ; /* peut être initialise a 1 */
c[0]=faux ;
c[1]=faux ;</td></tr>
<tr><td>Processus 0</td><td>Processus 1</td></tr>
<tr><td>c[0] = vrai ;
tour =1 ;
répéter <Rien> jusqu’à (c[1] = faux) ou (tour= 0) ;
< Section critique >
c[0]=faux ;</td><td>c[1] = vrai ;
tour =0 ;
répéter <Rien> jusqu’à (c[0] = faux) ou (tour= 1) ;
< Section critique >
c[1]=faux ;</td></tr>
</table>

Analyse de la solution de Peterson

- C'est une solution bien plus simple que la solution de Dekker.

1. Exclusion mutuelle: Supposons que P0 et P1 soient dans leurs sections critiques.

P0 en SC →
- c[1]=faux (P1 n'est pas intéresser par la SC)
- c[0]=vrai et tour=0

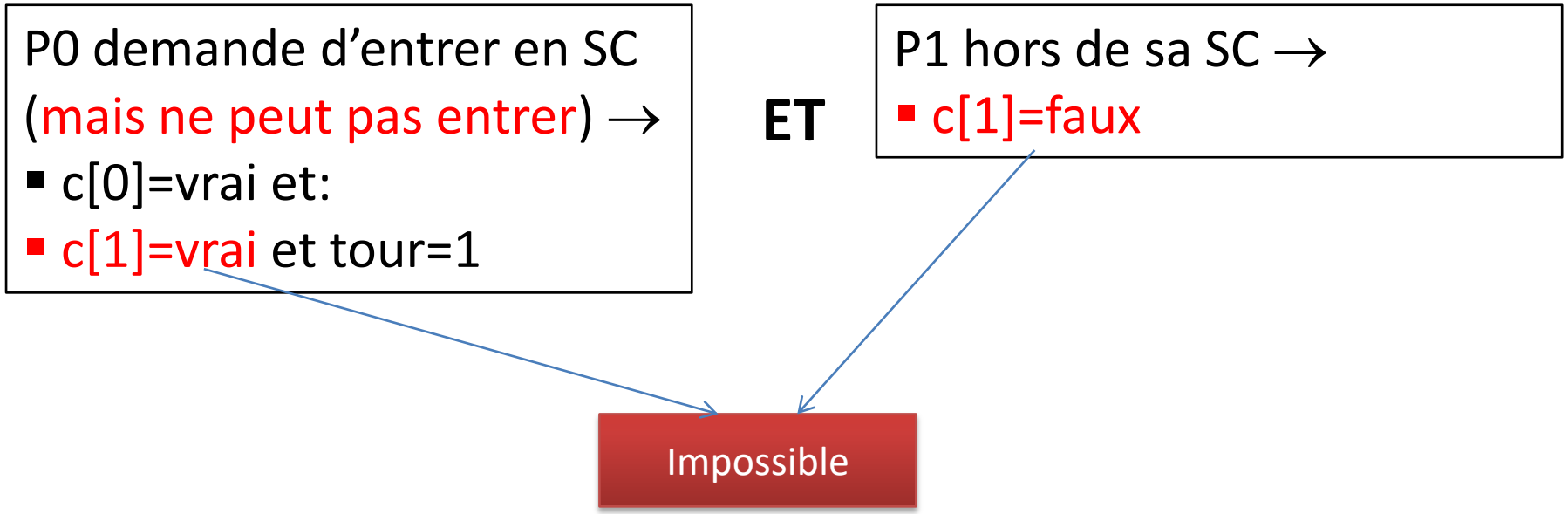
P1 en SC →
- c[0]=faux (P0 n'est pas intéresser par la SC)
- c[1]=vrai et tour =1

ET

Impossible

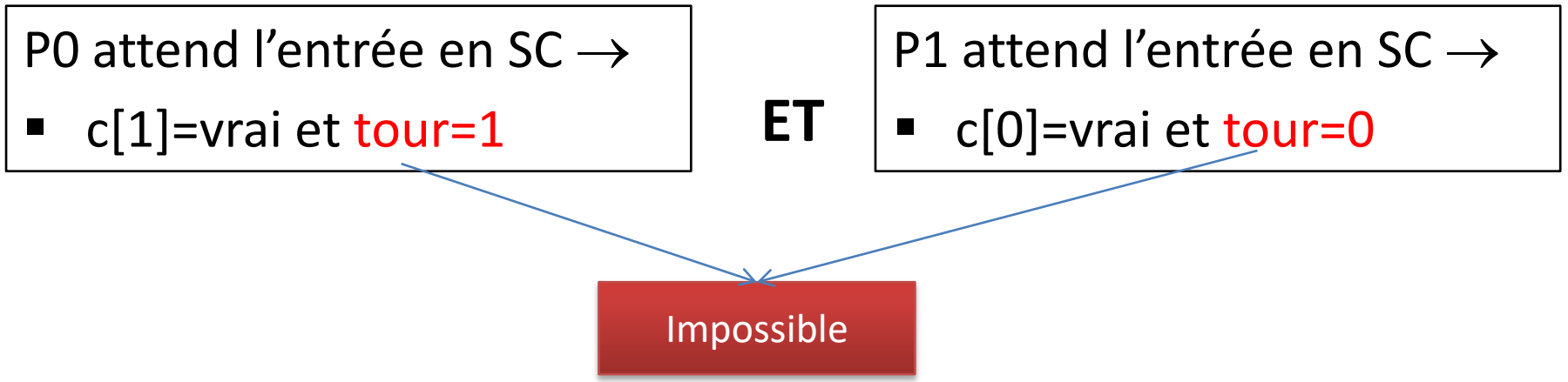
2. Absence de blocage

- Supposons que P1 n'est pas intéressé par la SC (en dehors de sa critique (actif ou bloqué) ou n'existe pas encore) .



3. Absence de l'interblocage

- Supposons que la section critique est libre et P0 et P1 demandent à y entrer; mais aucun d'eux ne peut entrer dans sa section critique.



4. Absence de famine

- Supposons que:
  - P0 est en SC.
  - P1 demande la SC mais doit se mettre en attente.
  - P0 termine mais demande une 2ème fois la SC.
- Lorsque P0 sort de la SC, il va mettre c[0] à faux, ensuite tour à 1 (lorsqu'il demande d'entrer en SC), ce qui permet à P1 d'entrer en SC (s'il était bloqué en attente de la SC)

b. Solutions matérielles

1. Masquage/démasquage des interruptions

- L'exclusion mutuelle peut être assurée en masquant les interruptions:
  - Dès qu'un processus accède à la SC , aucun événement susceptible d'activer un autre processus ne peut être pris en compte.
  - Evite au processus actif de perdre le processeur, pendant l'exécution de sa section critique.

1. Masquage/démasquage des interruptions

- Exemples d'interruptions:
  - Interruption horloge en fin de quantum
  - Arrivée ou réveil d'un processus plus prioritaire.

```
Processus Pi
Masquer les interruptions
<Section critique>
Démasquer les interruptions
```

1. Masquage/démasquage des interruptions

- Cette solution doit être utilisée avec précaution car elle peut affecter le temps de réponse du système.
- La section critique doit être la plus courte possible
  - Risque de perte ou de retard des interruptions prioritaires.

2. Les instructions spéciales

- Certains processeurs offrent des instructions atomiques (indivisibles) spéciales de consultation et de modification d'un mot mémoire,
  - Le blocage de l'accès au mot mémoire pendant l'exécution de l'instruction est assuré par le matériel.
- Ce type d'instructions permet d'implémenter correctement la première solution logicielle

2.1. Test And Set ou TAS(m)

- La consultation et la modification de m sont réalisées par une seule instruction indivisible.
- Cette instruction existe sur certaines machines (Ex: Motorola 68000).

```
fonction TAS(var m :booléen) :booléen ;
    début
    TAS = m ;
    m = vrai ;
    fin.
```

Utilisation de TAS

- Soit occupe la variable booléenne commune protégeant la SC;
  - occupe=faux → Ressource libre; occupe=vrai → Ressource occupée.

```
var occupe :booléen ;
occupe = faux ;

Processus Pi (i=1..n)
    tant que TAS (occupe) faire <rien>
    <Section critique>
    occupe = faux ;
```

2.2. Instruction LOCK XCHG des processeurs Intel x86

- Le préfixe LOCK permet de rendre l'instruction XCHG qui suit indivisible.
  - LOCK XCHG Registre, mémoire

Initialisation : MOV v, 0 ; v est une variable commune
Processus i
Demande d'entrée en section critique : boucle: MOV AL,1 ; AL =1 **LOCK XCHG AL, v** ; échange du contenu de AL avec celui de v ; à la fin de l'exécution de l'instruction : ;v=1 et AL = ancienne valeur de v**.** CMP AL, 0 ; tester si section critique libre : ; si v égale à 0; JNE boucle ; si section critique occupée aller à boucle.
< Section Critique >
Sortie de la section critique : MOV v, 0 ; libérer la section critique

Inconvénients des instructions spéciales

- Les instructions spéciales utilisent l'attente active:
  - Mauvaise utilisation du temps processeur (même problème qu'avec les solutions logicielles).
- Il y a risque famine: Si plusieurs processus demandent à entrer en section critique, le choix du processus à accéder en section critique est arbitraire
  - Un processus peut passer plusieurs fois alors qu'il y a des processus qui attendent l'entrée en section critique.