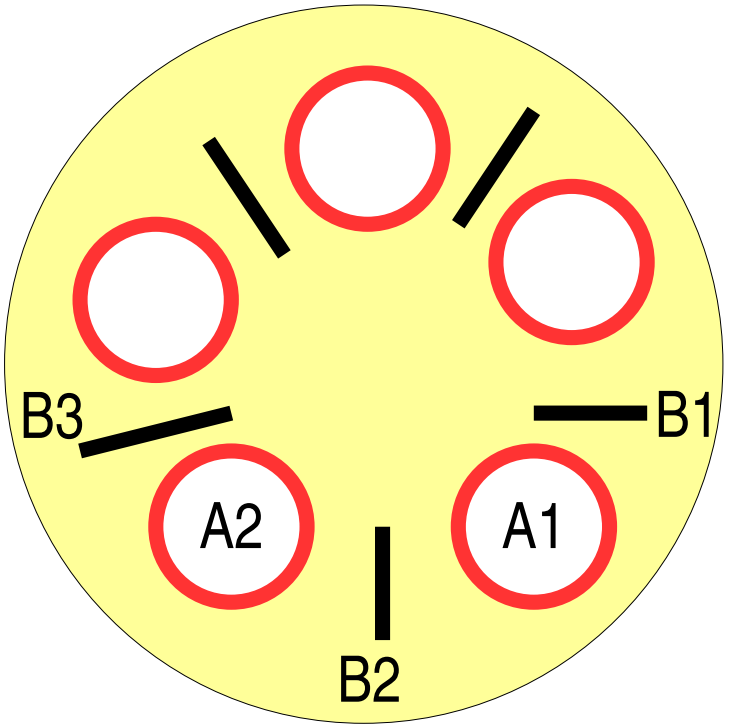
Programmation système – TD6

Le dîner des philosophes Chinois

Le problème à résoudre



• Chaque philosophe nécessite deux baguettes

• Les baguettes sont des **ressources critiques** !

→ *Une baguette ne peut pas être utilisée par plusieurs mangeurs en même temps.*

• Or, il n'y a que cinq baguettes !

→ *Il en faudrait dix.*

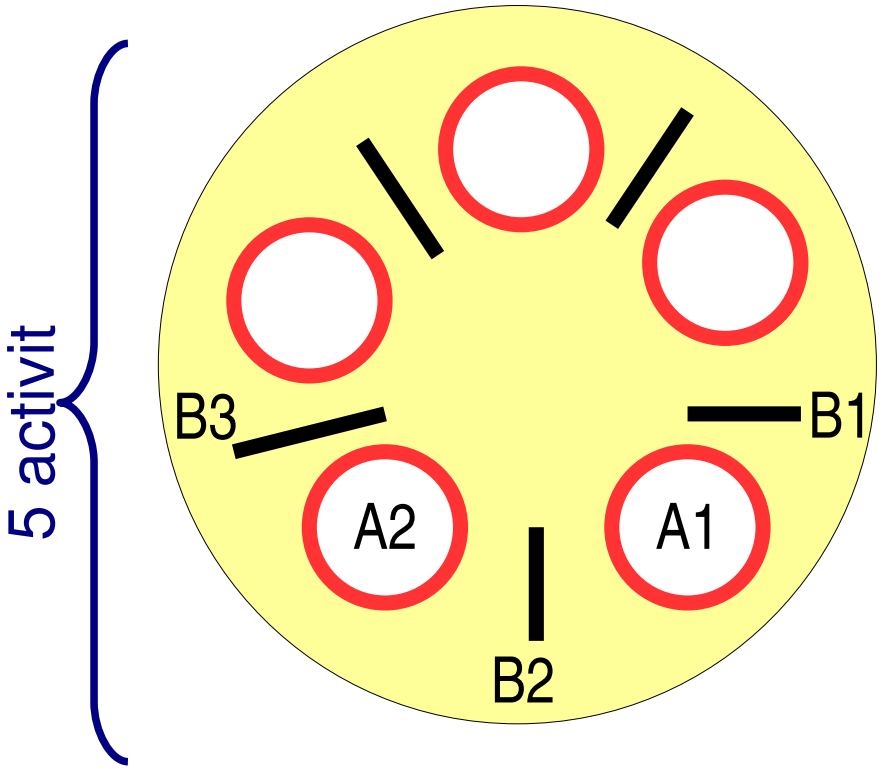
► **Des philosophes pourront être en attente de disponibilité de baguettes.**

La solution des mutexes

Une solution : associer à chaque baguette un **mutex** utilisé suivant **l'algorithme de Dijkstra.**

SB1 : **1**
SB2 : **1**
SB3 : **1**
SB4 : **1**
SB5 : **1**

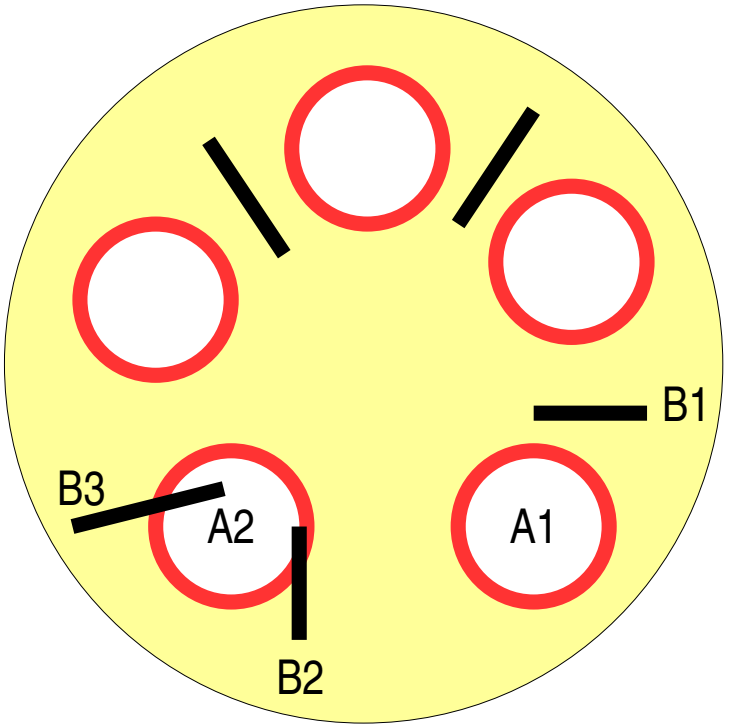
Philosophe P1
TantQue VRAI **faire**
    Pensage (lol)
    P(**SB1**)
    P(**SB2**)
    Mangeage (lol)
    V(**SB1**)
    V(**SB2**)
FinTantQue



La solution des mutexes

Hypothèse : le philosophe **P2** mange et **P1** veut manger**.**

SB1 :	**0**
SB2 **:**	**0**
SB3 **:**	**0**
SB4 **:**	**1**
SB5 **:**	**1**



Philosophe P1

TantQue VRAI **faire**
- Pensage (lol)
- P1(**SB1**) Acquisition de **B1**
- P1(**SB2**) Sommeil !
- Mangeage (lol)
- V1(**SB1**)
- V1(**SB2**)

FinTantQue

Problème : **P1** est endormi en détenant inutilement **B1.**

La solution des mutexes

Hypothèse : tous les philosophes veulent manger**.**

SB1 : **1** SB2 : **1** SB3 : **1** SB4 : **1** SB5 : **1**

Philosophe P1
TantQue VRAI **faire**
Pensage (lol)
P(**SB1**)

P(**SB2**) *sommeil !!!*

Philosophe P2
TantQue VRAI **faire**
Pensage (lol)

P(**SB2**)

P(**SB3**) *sommeil !!!*

Philosophe P3
TantQue VRAI **faire**
Pensage (lol)

P(**SB3**)

P(**SB4**)

etc...

Problème : tous les philosophes ont acquis leur baguette droite et sont **tous bloqués** en attente de leur baguette gauche !!!
→ **Interblocage**

Comment éviter le problème ?

- ✗ Demander au philosophe en attente d'une baguette de **relâcher** l'autre baguette au bout d'un certain temps
- ✗ Cela fonctionne-t-il à tous les coups ?
- ✗ **Non** : si tous les philosophes relâchent leur baguette en même temps, tous peuvent être à nouveau bloqués !
- ✗ La solution ? Acquérir les deux baguettes ou rien.

Autre solution

mut :	**1**
philo **:**	**0 0 0 0 0**
etat **:**	**P P P P P**

Philosophe Pi
TantQue VRAI **faire**
    /* penser */
    prendre_couverts(i)
    /* manger */
    poser_couverts(i)
FinTantQue

1. Rôle de **mut** : gérer l'accès à la ressource critique **etat**

```
procedure prendre_couverts(i : entier)
debut
    P(mut)
    etat[i] ← affame
    si(etat[i-1]!=mange) et (etat[i+1]!=mange) alors
        etat[i] ← mange
        V(philo[i])
    finsi
    V(mut)
    P(philo[i])
fin
```

```
procedure poser_couverts(i : entier)
debut
    P(mut)
    etat[i] ← pense
    si (etat[i-1]=affame) et etat[i-2]!=mange) alors
        etat[i-1] ← mange
        V(philo[i-1])
    finsi
    si (etat[i+1]=affame) et (etat[i+2]!=mange) alors
        etat[i+1] ← mange
        V(philo[i+1])
    finsi
    V(mut)
fin
```

Autre solution

mut :	**1**
philo **:**	**0 0 0 0 0**
etat **:**	**P P P P P**

Philosophe Pi
TantQue VRAI **faire**
    /* penser */
    prendre_couverts(i)
    /* manger */
    poser_couverts(i)
FinTantQue

2. Rôle de **philo** : bloquer passivement les philosophes en attente de baguettes

```
procedure prendre_couverts(i : entier)
debut
    P(mut)
    etat[i] ← affame
    si(etat[i-1]!=mange) et (etat[i+1]!=mange) alors
        etat[i] ← mange
        V(philo[i])
    finsi
    V(mut)
    P(philo[i])
fin
```

```
procedure poser_couverts(i : entier)
debut
    P(mut)
    etat[i] ← pense
    si (etat[i-1]=affame) et etat[i-2]!=mange) alors
        etat[i-1] ← mange
        V(philo[i-1])
    finsi
    si (etat[i+1]=affame) et (etat[i+2]!=mange) alors
        etat[i+1] ← mange
        V(philo[i+1])
    finsi
    V(mut)
fin
```

mut : **1**
philo **:** **0 0 0 0 0**
etat **:** **P P P P P**

```
procedure prendre_couverts(i : entier)
debut
    P(mut)                                  on s'assure l'usage exclusif de etat
    etat[i] ← affame                        on se déclare « affamé »
    si(etat[i-1]!=mange) et (etat[i+1]!=mange) alors
                                            si les voisins ne mangent pas..
        etat[i] ← mange                     … on peut manger !
        V(philo[i])                         philo[i] est levé pr ne pas être bloqué ensuite
    finsi
    V(mut)                                  l'accès à etat est libéré
    P(philo[i])                             si les voisins ne mangent pas, l'opération P est
                                            non bloquante (philo[i] vaut 1), sinon : mise en
                                            sommeil.
fin
```

mut : **1**
philo **:** **0 0 0 0 0**
etat **:** **P P P P P**

```
procedure poser_couverts(i : entier)
debut
    P(mut)                                  on s'assure l'usage exclusif de etat
    etat[i] ← pense                         on déclare qu'on ne mange plus
    si (etat[i-1]=affame) et etat[i-2]!=mange) alors
                                            Si le voisin de droite veut manger et si sa
                                            baguette droite est libre...
        etat[i-1] ← mange                   … il peut manger ...
        V(philo[i-1])                       … on le réveille (il est endormi sur P(philo))
    finsi
    si (etat[i+1]=affame) et (etat[i+2]!=mange) alors
                                            Même processus pour le voisin de gauche
        etat[i+1] ← mange
        V(philo[i+1])
    finsi
    V(mut)                                  on libère l'accès à etat
fin
```

La solution du moniteur

```
MONITEUR     Philo

VARIABLES    etat[5]                          // dénote l'état des philosophes
CONDITIONS   C[5] : tableau de conditions     // une condition par philosophe

procedure prendre_couverts(i : entier)
debut
    etat[i] ← affame                  on se déclare « affamé »
    si(etat[i-1]!=mange) et (etat[i+1]!=mange) alors
                                      si les voisins ne mangent pas...
        etat[i] ← mange               … on peut manger !
    sinon
        wait(C[i])                    sinon on s'endort
    finsi
fin
```

La solution du moniteur (suite)

```
procedure poser_couverts(i : entier)
Début
    etat[i] ← pense                          on déclare qu'on ne mange plus
    si (etat[i-1]=affame) et etat[i-2]!=mange) alors
                                             Si le voisin de droite veut manger et si sa
                                             baguette droite est libre...
        etat[i-1] ← mange                    … il peut manger ...
        signal(C[i-1])                       … on le réveille (il est endormi sur wait)
    finsi
    si (etat[i+1]=affame) et (etat[i+2]!=mange) alors
                                             Même processus pour le voisin de gauche
        etat[i+1] ← mange
        signal(C[i+1])
    finsi
Fin

Début                                        Initialisation
    Initialiser le tableau etat à « pense »
fin
```