

Communication entre processus : communication par fichiers partagés

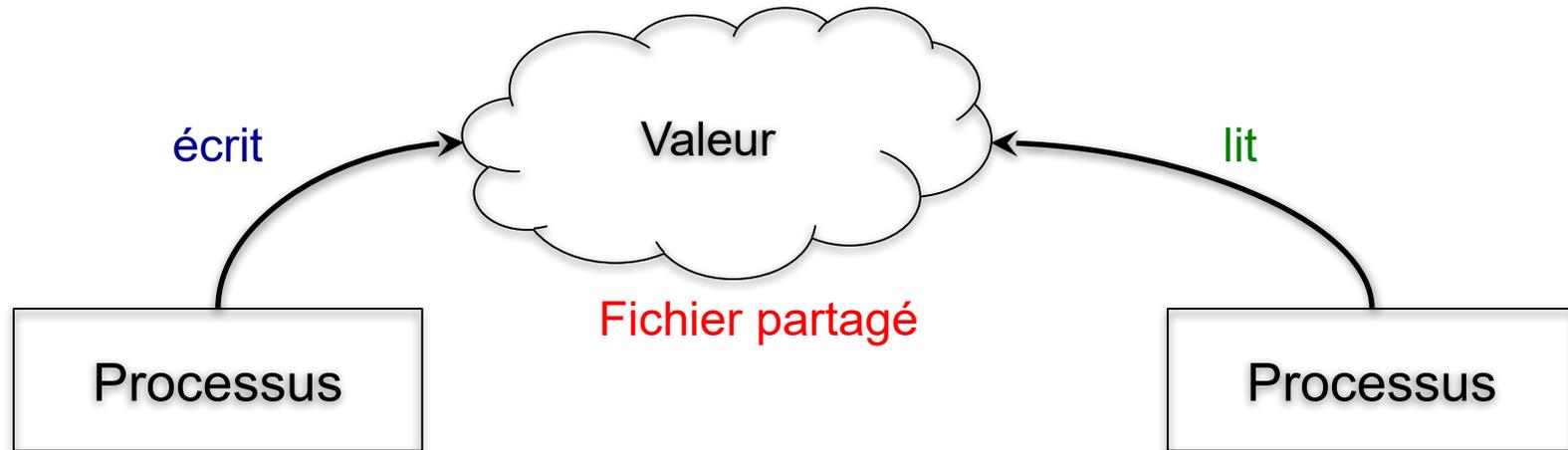
CSC 3102

Introduction aux systèmes d'exploitation

Gaël Thomas

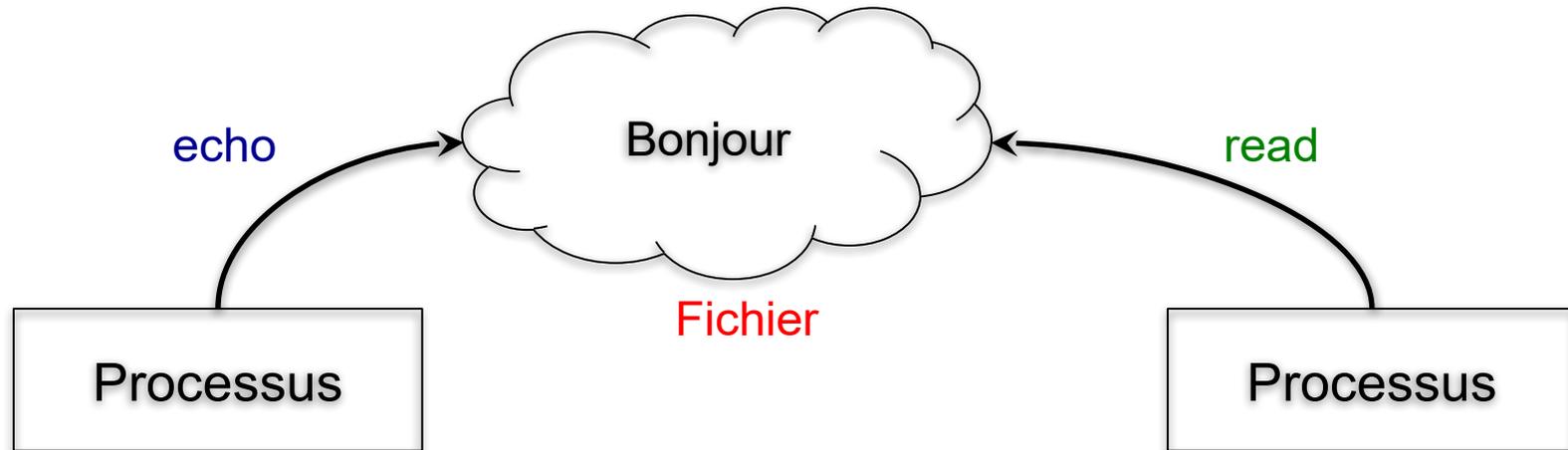


Communication par fichiers partagés



- Des processus écrivent dans et lisent un fichier partagé

Communication par fichiers partagés



■ Exemple

- P1 exécute : `echo "Bonjour" > f1`
- P2 exécute : `read a < f1`

■ Différence entre tube nommé et fichier

- Tube nommé : messages supprimés après la lecture
- Fichier partagé : données non supprimées après la lecture

Le problème des fichiers partagés

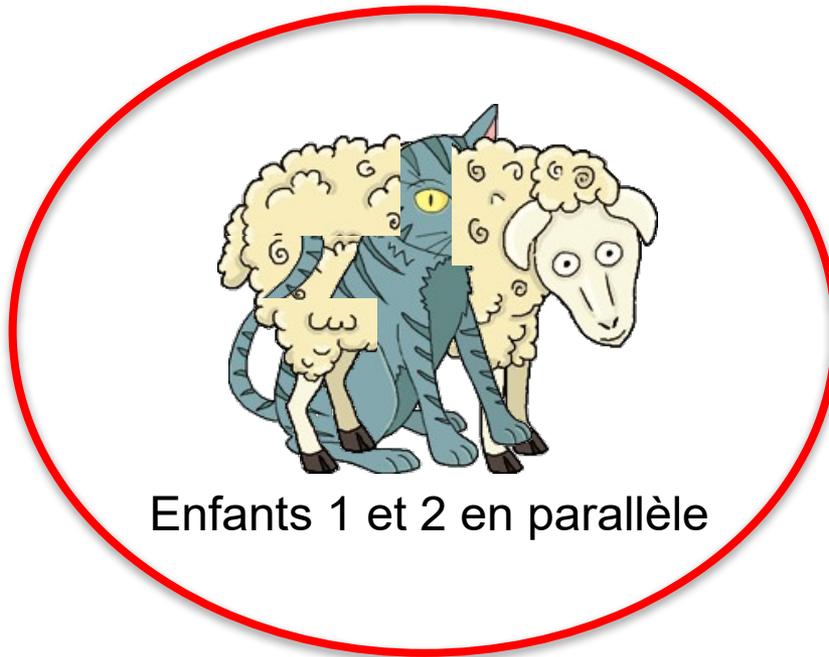
- Les **fichiers** peuvent être mis à jour concurremment
- Les accès concurrents aux fichiers partagés peuvent mener à des incohérences

Problème de la mise à jour concurrente

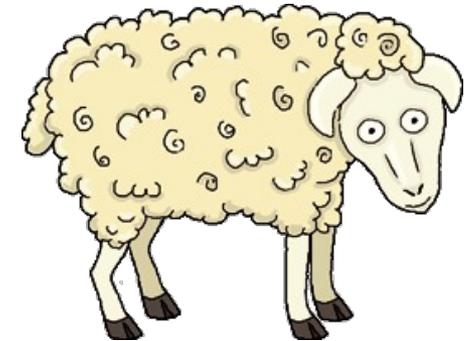
Deux enfants dessinent sur un tableau



Enfants 1 puis 2



Enfants 1 et 2 en parallèle



Enfants 2 puis 1

Donnée incohérente!

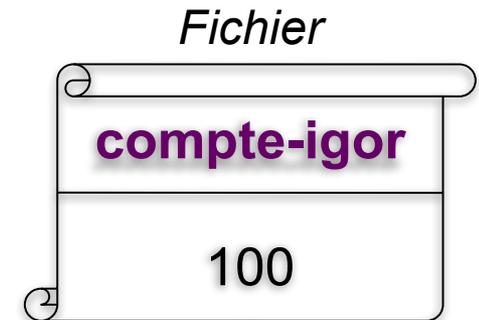
Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
read a < compte-igor  
a=$(expr $a + 2)  
echo $a > compte-igor
```

P2 : débite le compte d'Igor de 100 euros

```
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor
```



Fichier partagé par P1 et P2

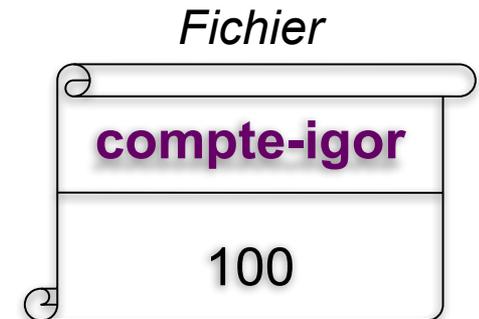
Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
➤ read a < compte-igor  
a=$(expr $a + 2)           a : 100  
echo $a > compte-igor
```

P2 : débite le compte d'Igor de 100 euros

```
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor
```



Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
read a < compte-igor
```

```
➤ a=$(expr $a + 2) a : 102
```

```
echo $a > compte-igor
```

Fichier



P2 : débite le compte d'Igor de 100 euros

```
read b < compte-igor
```

```
b=$(expr $b - 100)
```

```
echo $b > compte-igor
```

Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
read a < compte-igor
```

```
a=$(expr $a + 2)           a : 102
```

```
➤ echo $a > compte-igor
```

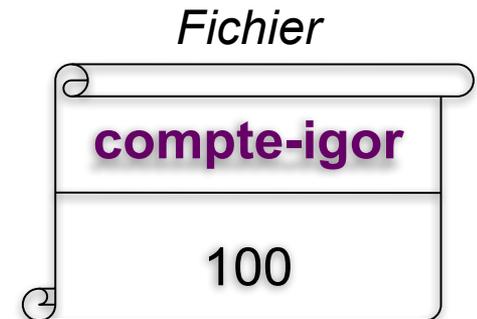
Commutation de P1 vers P2 avant le echo

P2 : débite le compte d'Igor de 100 euros

```
➤ read b < compte-igor
```

```
b=$(expr $b - 100)         b : 100
```

```
echo $b > compte-igor
```



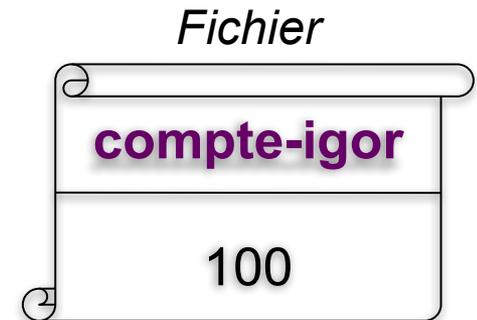
Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
read a < compte-igor
```

```
a=$(expr $a + 2)           a : 102
```

```
➤ echo $a > compte-igor
```



P2 : débite le compte d'Igor de 100 euros

```
read b < compte-igor
```

```
➤ b=$(expr $b - 100)           b : 0
```

```
echo $b > compte-igor
```

Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
read a < compte-igor
```

```
a=$(expr $a + 2)
```

a : 102

```
➤ echo $a > compte-igor
```

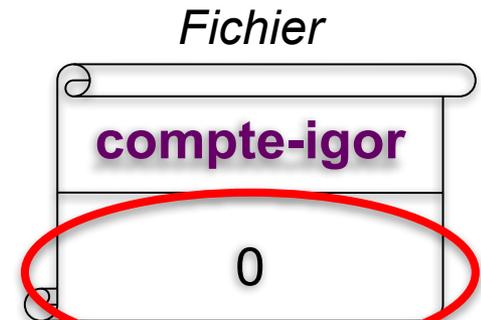
P2 : débite le compte d'Igor de 100 euros

```
read b < compte-igor
```

```
b=$(expr $b - 100)
```

b : 0

```
➤ echo $b > compte-igor
```



Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

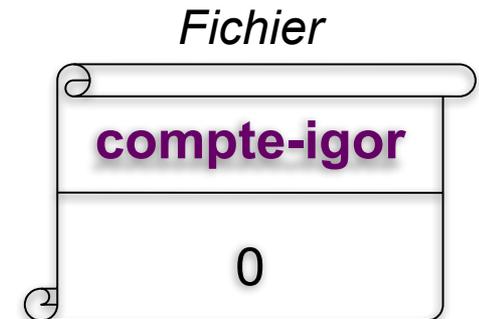
```
read a < compte-igor
```

```
a=$((expr $a + 2))
```

a : 102

```
➤ echo $a > compte-igor
```

P2 se termine



Problème de la mise à jour concurrente

P1 : crédite le compte d'Igor de 2 euros

```
read a < compte-igor
```

```
a=$((expr $a + 2))
```

```
➤ echo $a > compte-igor
```

P2 se termine

⇒ **commutation de P2 vers P1**

⇒ **P1 exécute le echo**

a : 102



Le retrait de 100 euros a été perdu!

Principe de solution

Éviter que deux sections de code accédant au même fichier partagé puissent s'exécuter en même temps

⇒ on parle de sections de code en **exclusion mutuelle**

Sections critiques : sections de code en exclusion mutuelle

⇒ les sections critiques s'exécutent entièrement l'une après l'autre

Remarque : une section critique est souvent en exclusion mutuelle avec elle-même

Mise en œuvre de l'exclusion : le verrou

- Mutex : verrou (lock) en exclusion mutuelle
- Principe :
 - Verrouille le verrou avant d'entrer en section critique
 - Déverrouille le verrou à la sortie d'une section critique
- Deux opérations atomiques

Atomique : semble s'exécuter instantanément

 - P.sh : attend que le verrou soit déverrouillé et le verrouille
P comme « puis-je? » (*Proberen = tester en Néerlandais*)
 - V.sh : déverrouille le verrou
V comme « vas-y » (*Verhogen = incrémenter en Néerlandais*)

Mise en œuvre de l'exclusion : le verrou

- Le verrou a été introduit par **Edsger W. Dijkstra** en 1965

E. W. Dijkstra. 1965. Solution of a problem in concurrent programming control. Commun. ACM 8, 9, pp. 569-569.

- Puis généralisé par **Edsger W. Dijkstra** avec les sémaphores

E. W. Dijkstra. 1968. The structure of the "THE"-multiprogramming system. Commun. ACM 11, 5, pp. 341-346.



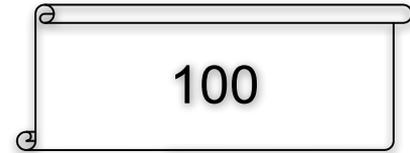
Exemple

P1 : crédite le compte d'Igor de 2 euros

```
P.sh compte-igor.lock  
read a < compte-igor  
a=$(expr $a + 2)  
echo $a > compte-igor  
V.sh compte-igor.lock
```

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock  
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor  
V.sh compte-igor.lock
```



compte-igor



compte-igor.lock

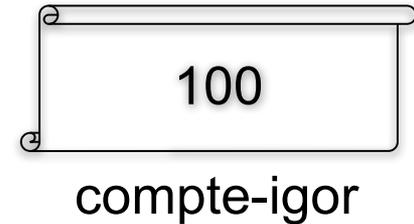
Exemple

P1 : crédite le compte d'Igor de 2 euros

```
➤ P.sh compte-igor.lock  
read a < compte-igor  
a=$(expr $a + 2)  
echo $a > compte-igor  
V.sh compte-igor.lock
```

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock  
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor  
V.sh compte-igor.lock
```



compte-igor.lock

Exemple

P1 : crédite le compte d'Igor de 2 euros

```
P.sh compte-igor.lock
```

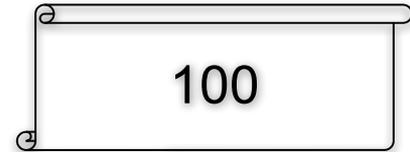
```
➤ read a < compte-igor
```

```
a=$(expr $a + 2)
```

a : 100

```
echo $a > compte-igor
```

```
V.sh compte-igor.lock
```



compte-igor



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock
```

```
read b < compte-igor
```

```
b=$(expr $b - 100)
```

```
echo $b > compte-igor
```

```
V.sh compte-igor.lock
```

Exemple

P1 : crédite le compte d'Igor de 2 euros

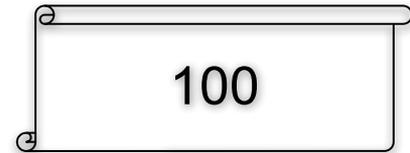
```
P.sh compte-igor.lock
```

```
read a < compte-igor
```

```
➤ a=$(expr $a + 2) a : 102
```

```
echo $a > compte-igor
```

```
V.sh compte-igor.lock
```



compte-igor



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock
```

```
read b < compte-igor
```

```
b=$(expr $b - 100)
```

```
echo $b > compte-igor
```

```
V.sh compte-igor.lock
```

Exemple

P1 : crédite le compte d'Igor de 2 euros

```
P.sh compte-igor.lock
```

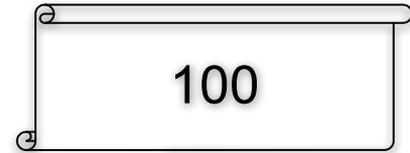
```
read a < compte-igor
```

```
a=$(expr $a + 2)
```

a : 102

```
➤ echo $a > compte-igor
```

```
V.sh compte-igor.lock
```



compte-igor



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
➤ P.sh compte-igor.lock
```

```
read b < compte-igor
```

```
b=$(expr $b - 100)
```

```
echo $b > compte-igor
```

```
V.sh compte-igor.lock
```

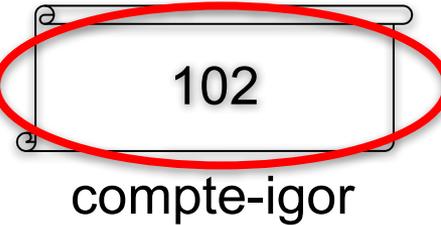
**Commutation de P1 vers P2
P2 bloque car verrou pris**

Exemple

P1 : crédite le compte d'Igor de 2 euros

```
P.sh compte-igor.lock  
read a < compte-igor  
a=$(expr $a + 2)  
➤ echo $a > compte-igor  
V.sh compte-igor.lock
```

a : 102



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
➤ P.sh compte-igor.lock  
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor  
V.sh compte-igor.lock
```

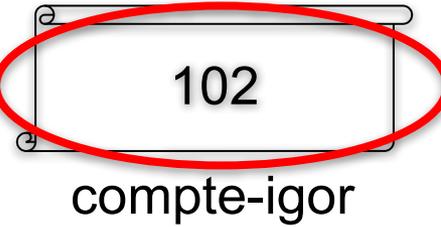
Commutation de P1 vers P2
P2 bloque car verrou pris
⇒ réélection de P1

Exemple

P1 : crédite le compte d'Igor de 2 euros

```
P.sh compte-igor.lock  
read a < compte-igor  
a=$(expr $a + 2)  
echo $a > compte-igor  
➤ V.sh compte-igor.lock
```

a : 102



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
➤ P.sh compte-igor.lock  
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor  
V.sh compte-igor.lock
```

**P1 a terminé sa
section critique
⇒ déverrouille le verrou**

Exemple

102
compte-igor



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
➤ P.sh compte-igor.lock  
read b < compte-igor  
b=$(expr $b - 100)  
echo $b > compte-igor  
V.sh compte-igor.lock
```

Fin P1
⇒ commutation vers P2
Et P2 prend le verrou

Exemple

102
compte-igor



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock
```

```
➤ read b < compte-igor
```

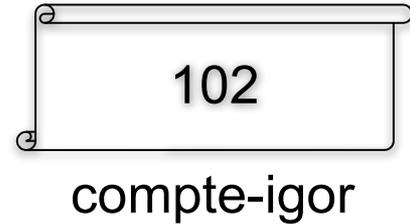
```
b=$(expr $b - 100)
```

```
echo $b > compte-igor
```

```
V.sh compte-igor.lock
```

b : 102

Exemple



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock
```

```
read b < compte-igor
```

```
➤ b=$(expr $b - 100)
```

```
echo $b > compte-igor
```

```
V.sh compte-igor.lock
```

b : 2

Exemple

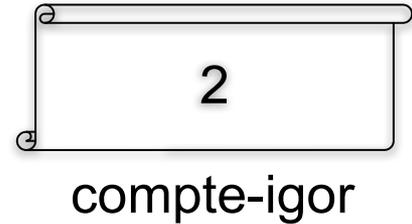
P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock  
read b < compte-igor  
b=$(expr $b - 100)  
➤ echo $b > compte-igor  
V.sh compte-igor.lock
```

b : 2



Exemple



compte-igor.lock

P2 : débite le compte d'Igor de 100 euros

```
P.sh compte-igor.lock
```

```
read b < compte-igor
```

```
b=$(expr b - 100)
```

```
echo b > compte-igor
```

b : 2

```
➤ V.sh compte-igor.lock
```

Remarques

Un mutex ne sert à rien si une section critique ne le prend pas!

Si on oublie de déverrouiller le mutex en fin de section critique, l'application reste bloquée

Attention à l'inter-blocage

- Inter-blocage : des processus concurrents s'attendent mutuellement
- Blocage définitif de l'avancée du programme
- Exemples :
 - Un processus s'attend lui-même (`wait` sur son PID...)
 - Un processus P1 verrouille V1 puis attend V2, qui est pris par P2 qui attend V1 (*voir diapositives suivantes*)

Attention à l'inter-blocage

■ Processus P1

```
P.sh v1.lock  
P.sh v2.lock  
...  
V.sh v2.lock  
V.sh v1.lock
```



v1.lock



v2.lock

■ Processus P2

```
P.sh v2.lock  
P.sh v1.lock  
...  
V.sh v1.lock  
V.sh v2.lock
```

Attention à l'inter-blocage

■ Processus P1

➤ P.sh v1.lock
P.sh v2.lock
...
V.sh v2.lock
V.sh v1.lock



v1.lock



v2.lock

■ Processus P2

P.sh v2.lock
P.sh v1.lock
...
V.sh v1.lock
V.sh v2.lock

Attention à l'inter-blocage

■ Processus P1

```
P.sh v1.lock  
➤ P.sh v2.lock  
...  
V.sh v2.lock  
V.sh v1.lock
```

■ Processus P2

```
➤ P.sh v2.lock  
P.sh v1.lock  
...  
V.sh v1.lock  
V.sh v2.lock
```

Commutation de P1 vers P2



v1.lock



v2.lock

Attention à l'inter-blocage

■ Processus P1

P.sh v1.lock

➤ P.sh v2.lock

...

V.sh v2.lock

V.sh v1.lock



v1.lock

■ Processus P2

P.sh v2.lock

➤ P.sh v1.lock

...

V.sh v1.lock

V.sh v2.lock

**P2 bloqué car
v1.lock est pris
par P1**



v2.lock

Attention à l'inter-blocage

■ Processus P1

```
P.sh v1.lock  
➤ P.sh v2.lock  
...  
V.sh v2.lock  
V.sh v1.lock
```

Commutation de P2 vers P1

**P1 bloqué car
v2.lock est pris
par P2**



v1.lock



v2.lock

■ Processus P2

```
P.sh v2.lock  
➤ P.sh v1.lock  
...  
V.sh v1.lock  
V.sh v2.lock
```

**P2 bloqué car
v1.lock est pris
par P1**

**⇒ ni P1, ni P2 ne peuvent
progresser...**

Règle pour éviter l'inter-blocage

Il faut toujours prendre les verrous dans le même ordre dans tous les processus

Notions clés

- **Section critique** : section de code en **exclusion mutuelle**
 - Deux sections critiques ne peuvent pas s'exécuter en parallèle
- **Mise en œuvre des sections critiques avec des **mutex**** :
 - $P.sh$: entrée en section critique
 - Bloque tant qu'il existe un processus en section critique
 - $V.sh$: sortie de section critique
- **Attention aux **inter-blocages**** : toujours prendre les mutex dans le même ordre dans tous les processus

À vous de jouer!