Chapitre

Synchronisation

Java

Processus

- Unité d'exécution gérée par le système
- Interaction limitée entre processus

Unix: fork

Processus en Java

- Méthode Runtime.exec (String cmd)
  - retourne une instance de la classe Process
  - commande exécutée dans un processus séparé

Possibilité de passer un environnement en paramètre

Classe Process

Contrôle du processus

- Destruction
- Flot E/S
- Attente terminaison

Utilisations possibles

- Mécanisme "secondaire"
- Pas de gestion fine de la concurrence
- Lancement de commandes (prog. C)

Les processus légers

- Concurrence à l'intérieur d'un même processus
- Threads (processus léger, fil d'exécution, activité dans un programme)

Deux instructions d'un même programme peuvent être traitées en même temps.

Propriétés des threads

- Exécution dans le même espace mémoire
- Exécution en concurrence

Un objet créé par un thread peut être utilisé par un autre thread.

Classe Thread

- Un objet de la classe Thread contrôle un processus léger

Création d'un thread par héritage

- Redéfinir la méthode "run" dans une sous-classe de "Thread"
- Exécuter la méthode "start"

Création d'un thread par héritage

```
public class MonRunnable extends Thread {
  public void run() {
    for (int i=0 ; i<5 ; i++) {
        System.out.println(i);
        try {
                sleep(500);
        } catch (InterruptedException e) {}
    }
}
```

Création d'un thread par héritage

```
MonThread th = new MonThread();
th.start();
```

Provoque un appel de la méthode "run"

Création de thread par implémentation de Runnable

```
public class MonRunnable implements Runnable {
  public void run() {
    for (int i=0 ; i<5 ; i++) {
        System.out.println(i);
        try {
                Thread.sleep(500);
        } catch (InterruptedException e) {}
    }
}
```

Création de thread par implantation de Runnable

```
Thread th = new Thread(new MonRunnable());
th.start();
```

Avantage de Runnable

- Possibilité d'héritage

Groupes de threads

- Un thread appartient à un groupe
- Application de propriétés au groupe

```
ThreadGroup g = new ThreadGroup("nouveau");
Thread th = new Thread(g,new MonRunnable(),"nom");
g.setMaxPriority(Thread.NORM_PRIORITY);
```

Contrôle des threads

- yield()

  le thread rend le contrôle

  mais peut le reprendre tout de suite

- sleep(int t)

  temps t exprimé en ms

  prévoit déjà les temps en ns

Contrôle des threads

- join()

  attente de la fin d'un thread

- suspend() - resume()

  à éviter (deprecated)

  java propose de meilleurs moyens de synchronisation

Contrôle des threads

- stop()

  ne pas utiliser (deprecated)

  risque d'incohérences en cas de synchronisation avec d'autres threads

Contrôle des threads

Méthode stop

Un thread stoppé débloque tous ses moniteurs (envoi de l'exception `ThreadDeath`) .

L'exception `ThreadDeath` détruit les threads de manière silencieuse (il est difficile de s'en apercevoir).

Un thread devrait toujours s'arrêter en sortant de la méthode run

Contrôle des threads (mauvais arrêt d'un thread)

```
private Thread blinker; // dans applet
public void start() {
 blinker = new Thread(this);
 blinker.start();
}
public void stop() {
 blinker.stop(); // MAUVAIS!!
}
public void run() {
 Thread thisThread = Thread.currentThread();
 while (true) {
  try { thisThread.sleep(interval); }
  catch (InterruptedException e){ }
  repaint(); }
}
```

Contrôle des threads (arrêt d'un thread)

```
private volatile Thread blinker;
public void stop() {
 blinker = null;
}
public void run() {  // arrêt par test
 Thread thisThread = // d'une variable
   Thread.currentThread();
 while (blinker == thisThread) {
  try { thisThread.sleep(interval); }
  catch (InterruptedException e){ }
  repaint(); } }
```

Évite des optimisations du compilateur

Contrôle des threads

- destroy()

  ne pas utiliser (non implémenté)

Daemons

- Threads de type "user"
- Threads de type "daemon"

Daemon: tâche de fond qui s'arrête quand l'application se termine

Daemons

```
Thread th = new MonThread();
th.setDaemon(true);
th.start();
// th.setDaemon(false); // erreur
```

Etat des threads

- Accès aux ressources
- Attente de ressources
- Attente d'événement

Comment libérer le processeur?

Priorité

- Thread.MIN_PRIORITY (1)
- Thread.NORM_PRIORITY (5)
- Thread.MAX_PRIORITY (10)

```
th.setPriority(Thread.MAX_PRIORITY);
th.getPriority()
```

Préemption

- Algorithme de scheduling non défini
- Tranches de temps parfois
- Dépend du système hôte

Solaris

- Threads Solaris non utilisés (LWP)
- Green threads utilisés
- Multi-processeurs pas pris en compte
- Fonctionnent au niveau "user"

Préemption sur priorité, pas de tranches de temps, pas d'appels système bloquants

Windows (95, NT, …)

- Utilisation des threads WIN32
- Comportement des programmes Java comme les autres programmes WIN32

Scheduling en fonction des priorités dynamiques, round robin si même priorité, risque de famine si priorité faible, possibilité de fonctionnement en multi-processeurs

Remarques

- Portage de Solaris à Windows
- Les primitives de synchronisation évitent les blocages
- Utiliser yield() si nécessaire et si disponible
- Eviter les priorités ?

Exclusion mutuelle

- Garantir des opérations atomiques

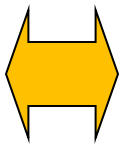
Exemple:

```
int x;

x  + + ;
```

Exclusion mutuelle

Exemple:

```
int x;

x ++ ;
```



```
MOV   X,REG0
INCR  REG0
MOV   REG0,X
```

Exclusion mutuelle (t1 et t2 exécutent x++)

Thread t1

```
; X vaut 1
; t1 activé
MOV X,REG0
; t1 suspendu



; t1 activé
INCR REG0
MOV  REG0,X
```

Thread t2

```
; t2 activé
MOV X,REG0
INCR REG0
MOV  REG0,X
; t2 suspendu
```

Mécanismes d'exclusion mutuelle

- Sémaphores
- Moniteurs

Sémaphores

- Sémaphores binaires
- Sémaphores de comptage

- Opération **P** (*Proberen*, tester)
- Opération **V** (*Verhogen*, incrémenter)

Puis-je ?
Vas-y !

Moniteurs

```
Moniteur nom_de_moniteur
{
      Déclarations de variables

      Procedure P1 (...) { ... }
      Procedure P2 (...) { ... }
      Procedure P3 (...) { ... }

      {
      // code de l'initialisation
      }
}
```

L'exclusion mutuelle est garantie.

Un seul processus peut être actif dans un moniteur

Moniteurs (exemple)

Au même moment:

Processus proc1 demande à exécuter la séquence P1 du moniteur

Processus proc2 demande à exécuter la séquence P3 du moniteur

Moniteurs (exemple)

Dans ce cas:

Un des processus sera suspendu, l'autre processus pourra exécuter la séquence demandée. Par exemple, Proc2 pourra exécuter P3.

Quand l'exécution de P3 sera terminée, Proc1 pourra exécuter P1.

Conditions dans les moniteurs

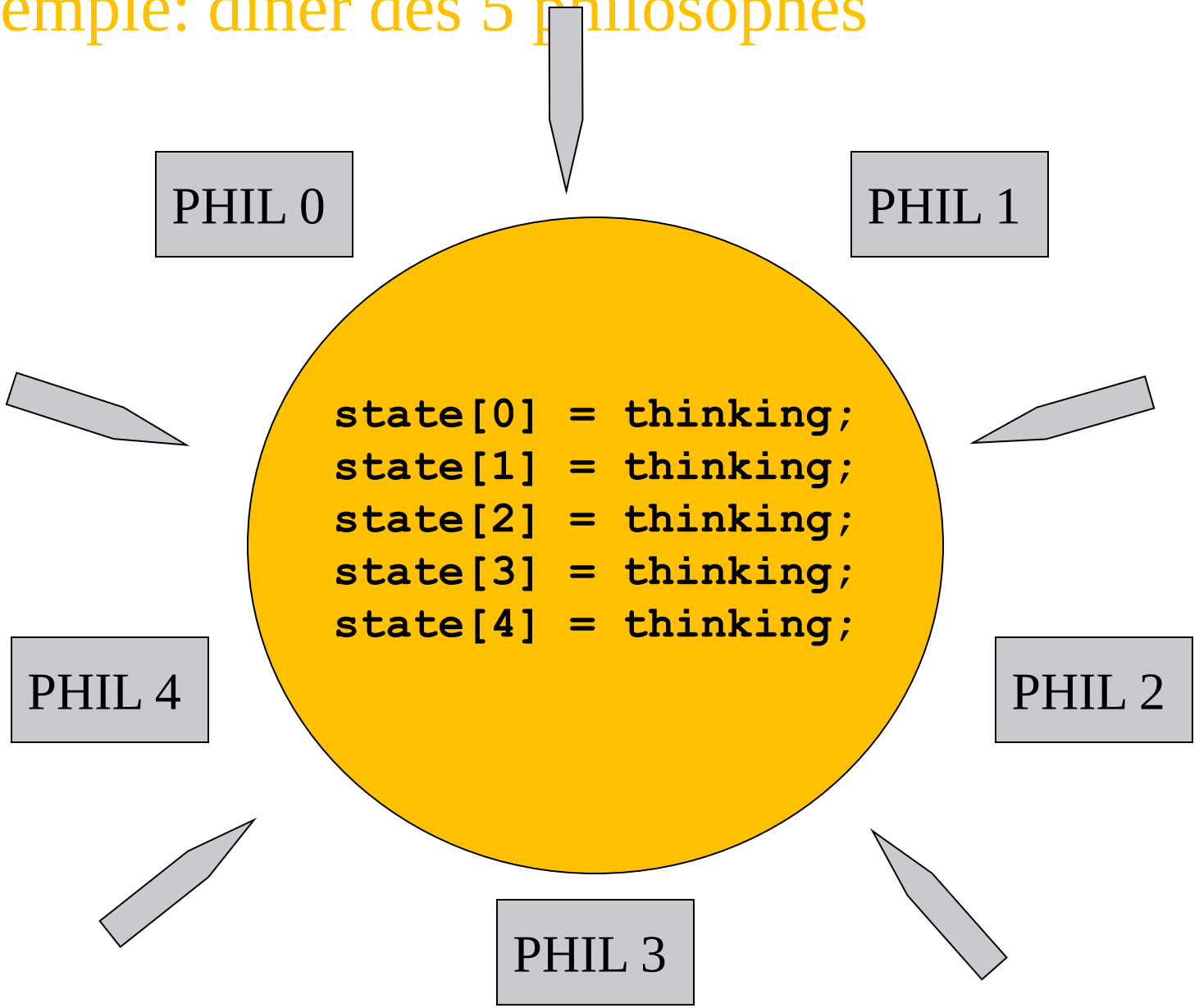
Les conditions sont des variables définies dans le moniteur.

Les opérations "**wait**" et "**signal**" sont les seules qui peuvent être appliquées à une variable conditionnelle. L'opération "wait" suspend le processus jusqu'à ce qu'un autre processus appelle "signal". L'opération "signal" reprend exactement le processus suspendu. S'il n'existe aucun processus suspendu, l'opération "signal" n'a aucun effet (noter la différence avec l'opération V sur les sémaphores).

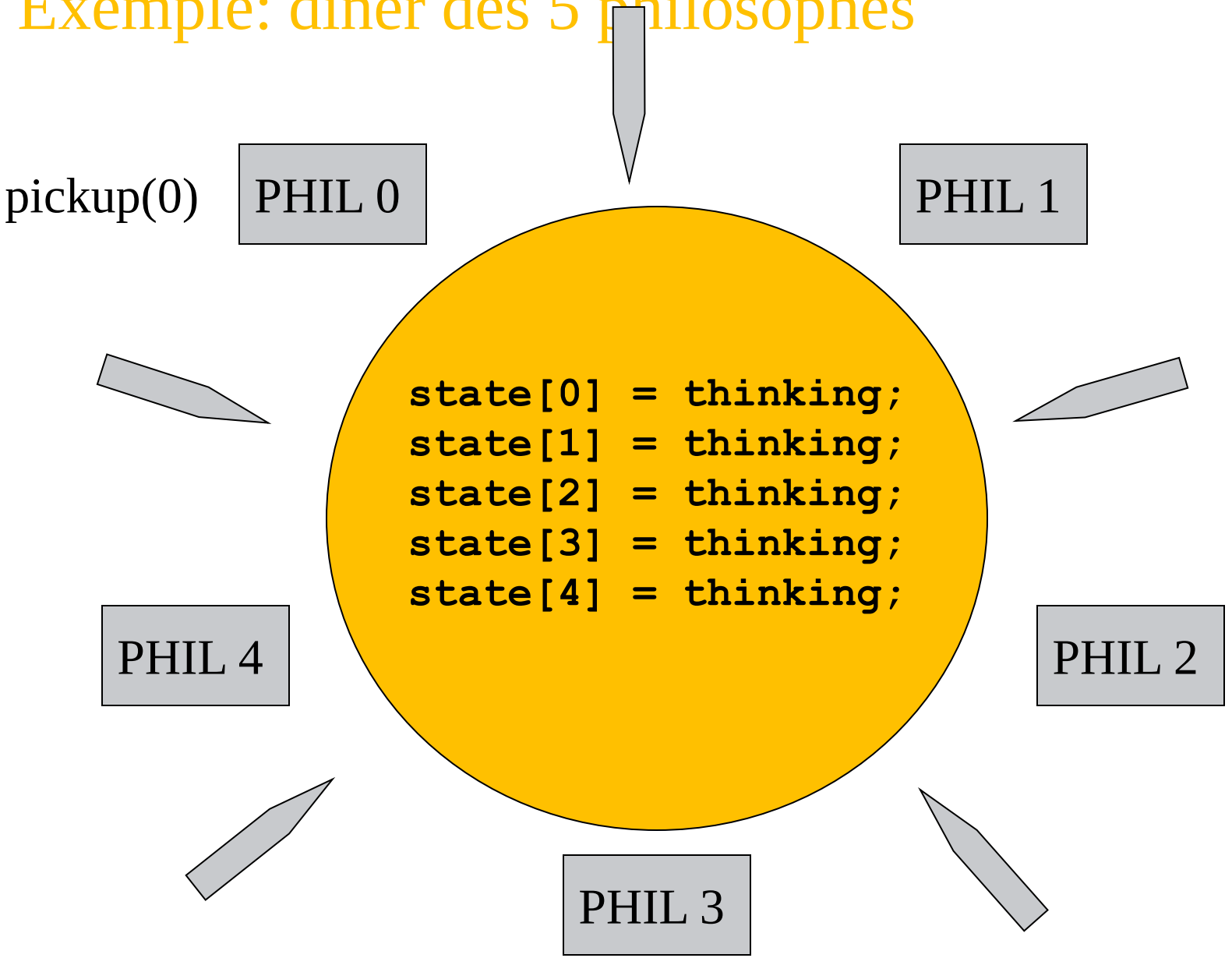
Exemple: dîner des 5 philosophes

```
Moniteur diner_des_5_philosophes {
        int state[5];     // thinking, hungry, eating
        condition self[5];

        procedure pickup(int i) {
                state[i] = hungry;
                test(i);
                if (state[i] != eating) self[i].wait;
        }
        procedure putdown (int i) {
                state[i] = thinking;
                test(i+4 mod 5);
                test(i+1 mod 5);
        }
        procedure test (int k) {
                if ((state[k+4 mod 5] != eating) &&
                                (state[k] == hungry) &&
                                (state[k+1 mod 5] != eating)) {
                        state[k] = eating;
                        self[k].signal;
                }
        }

        {for (int i=0 ; i<5 ; i++) state[i] = thinking;}
}
```

Exemple: dîner des 5 philosophes



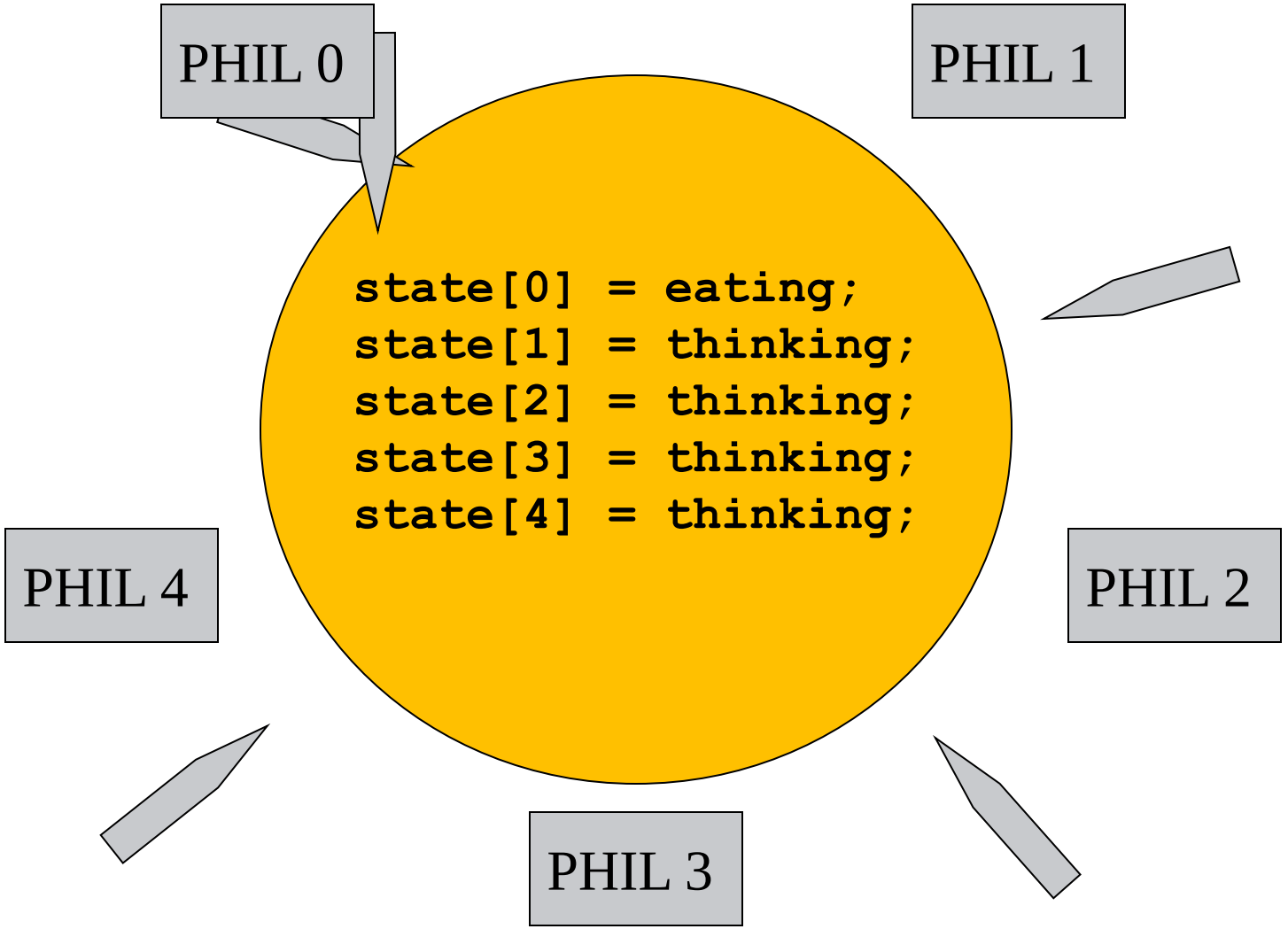
Exemple: dîner des 5 philosophes



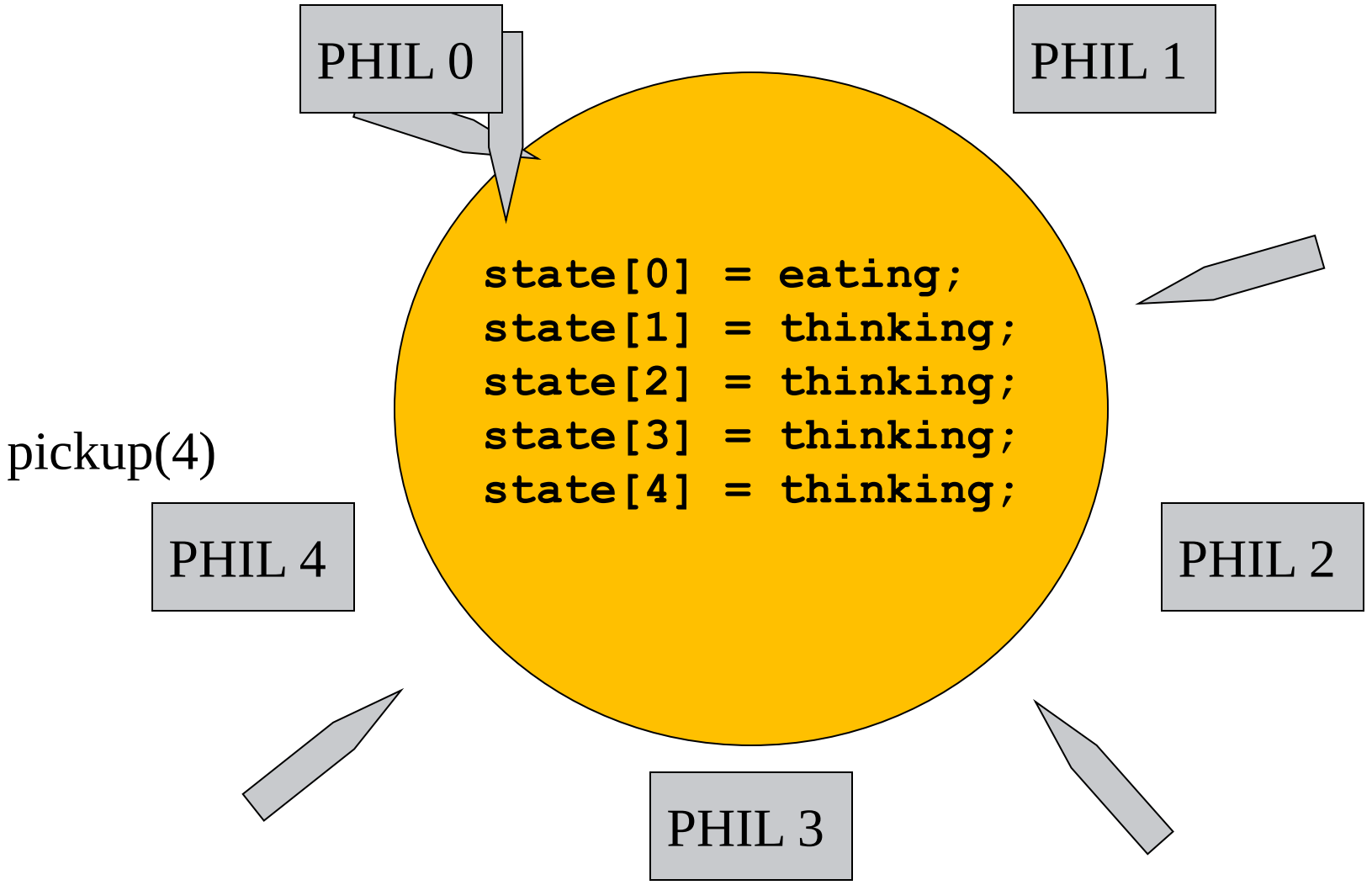
Exemple: dîner des 5 philosophes

```
Moniteur diner_des_5_philosophes {
        int state[5];     // thinking, hungry, eating
        condition self[5];

        procedure pickup(int i) {
                state[i] = hungry;
                test(i);
                if (state[i] != eating) self[i].wait;
        }
        procedure putdown (int i) {
                state[i] = thinking;
                test(i+4 mod 5);
                test(i+1 mod 5);
        }
        procedure test (int k) {
                if ((state[k+4 mod 5] != eating) &&
                                (state[k] == hungry) &&
                                (state[k+1 mod 5] != eating)) {
                        state[k] = eating;
                        self[k].signal;
                }
        }

        {for (int i=0 ; i<5 ; i++) state[i] = thinking;}
}
```

Exemple: dîner des 5 philosophes



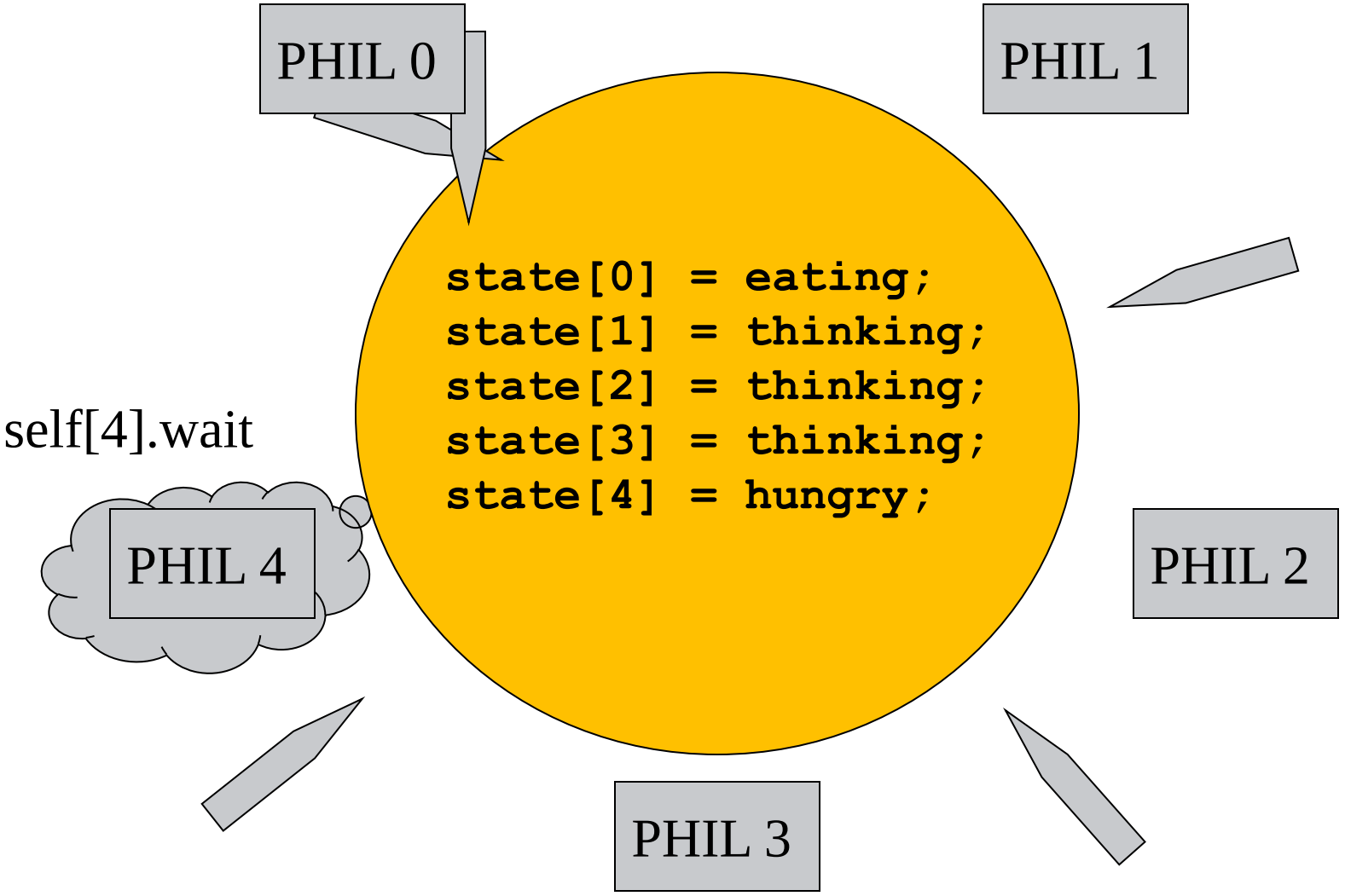
Exemple: dîner des 5 philosophes



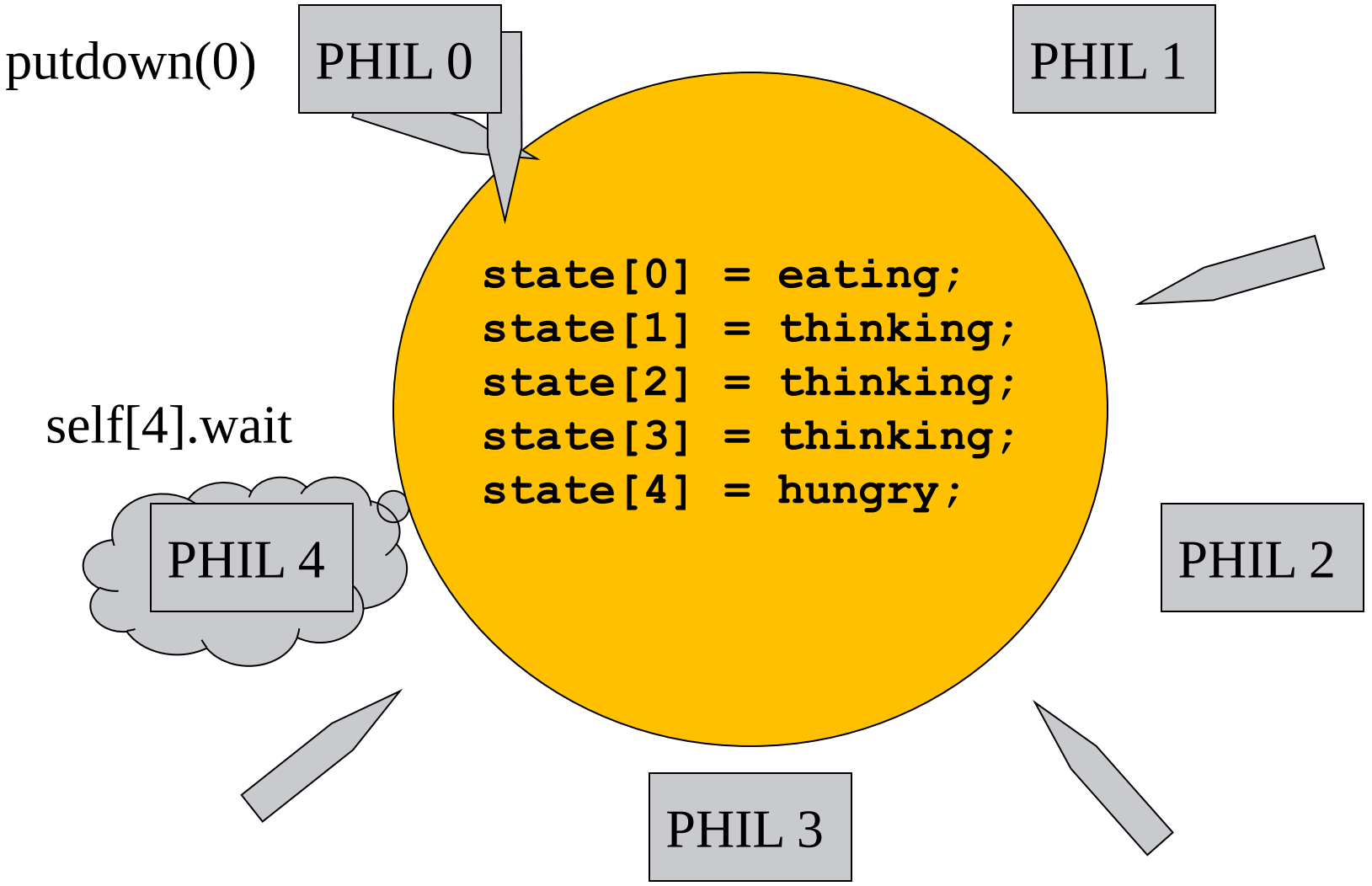
Exemple: dîner des 5 philosophes

```
Moniteur diner_des_5_philosophes {
        int state[5];     // thinking, hungry, eating
        condition self[5];

        procedure pickup(int i) {
                state[i] = hungry;
                test(i);
                if (state[i] != eating) self[i].wait;
        }
        procedure putdown (int i) {
                state[i] = thinking;
                test(i+4 mod 5);
                test(i+1 mod 5);
        }
        procedure test (int k) {
                if ((state[k+4 mod 5] != eating) &&
                                (state[k] == hungry) &&
                                (state[k+1 mod 5] != eating)) {
                        state[k] = eating;
                        self[k].signal;
                }
        }

        {for (int i=0 ; i<5 ; i++) state[i] = thinking;}
}
```

Exemple: dîner des 5 philosophes



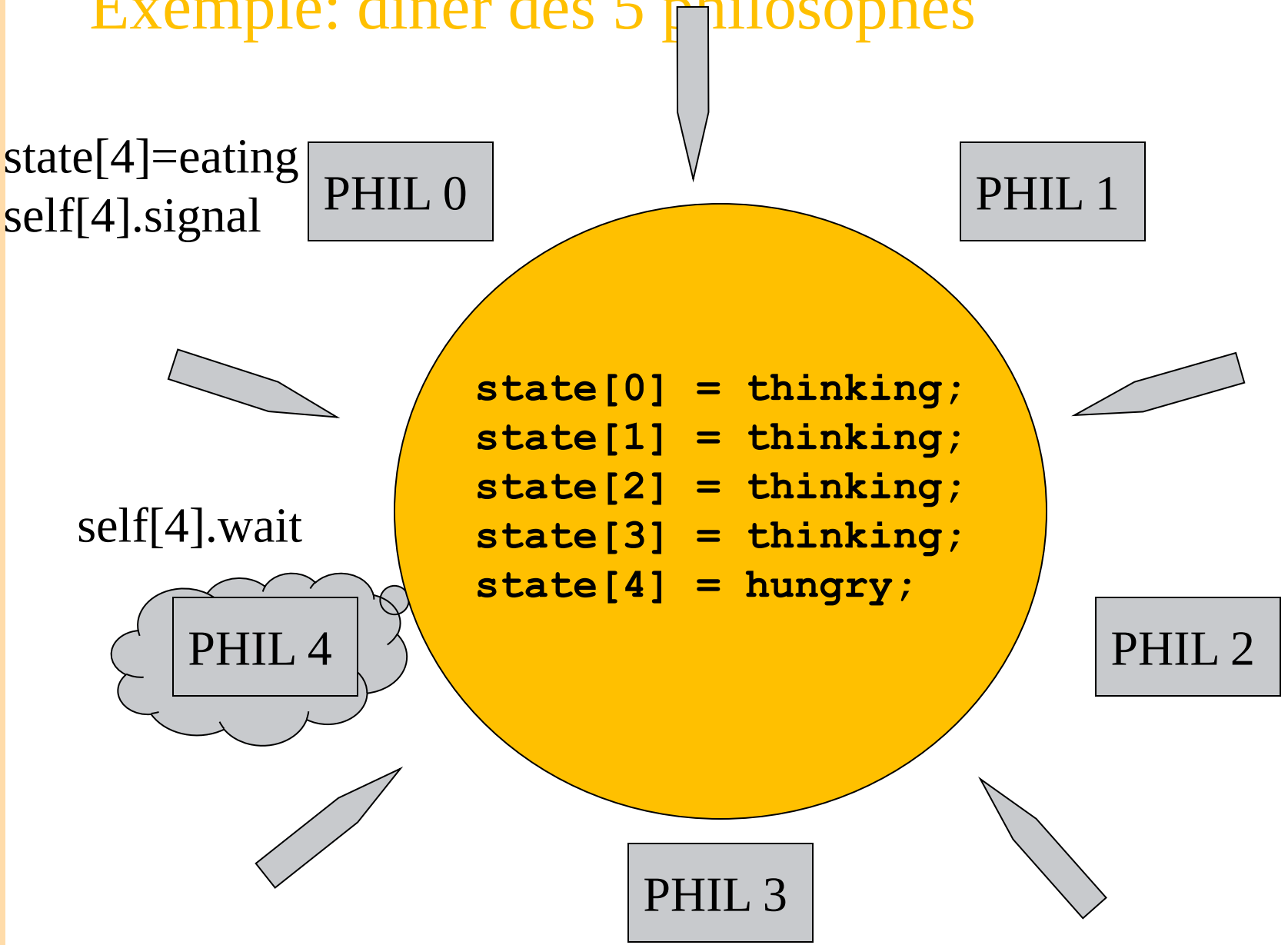
Exemple: dîner des 5 philosophes



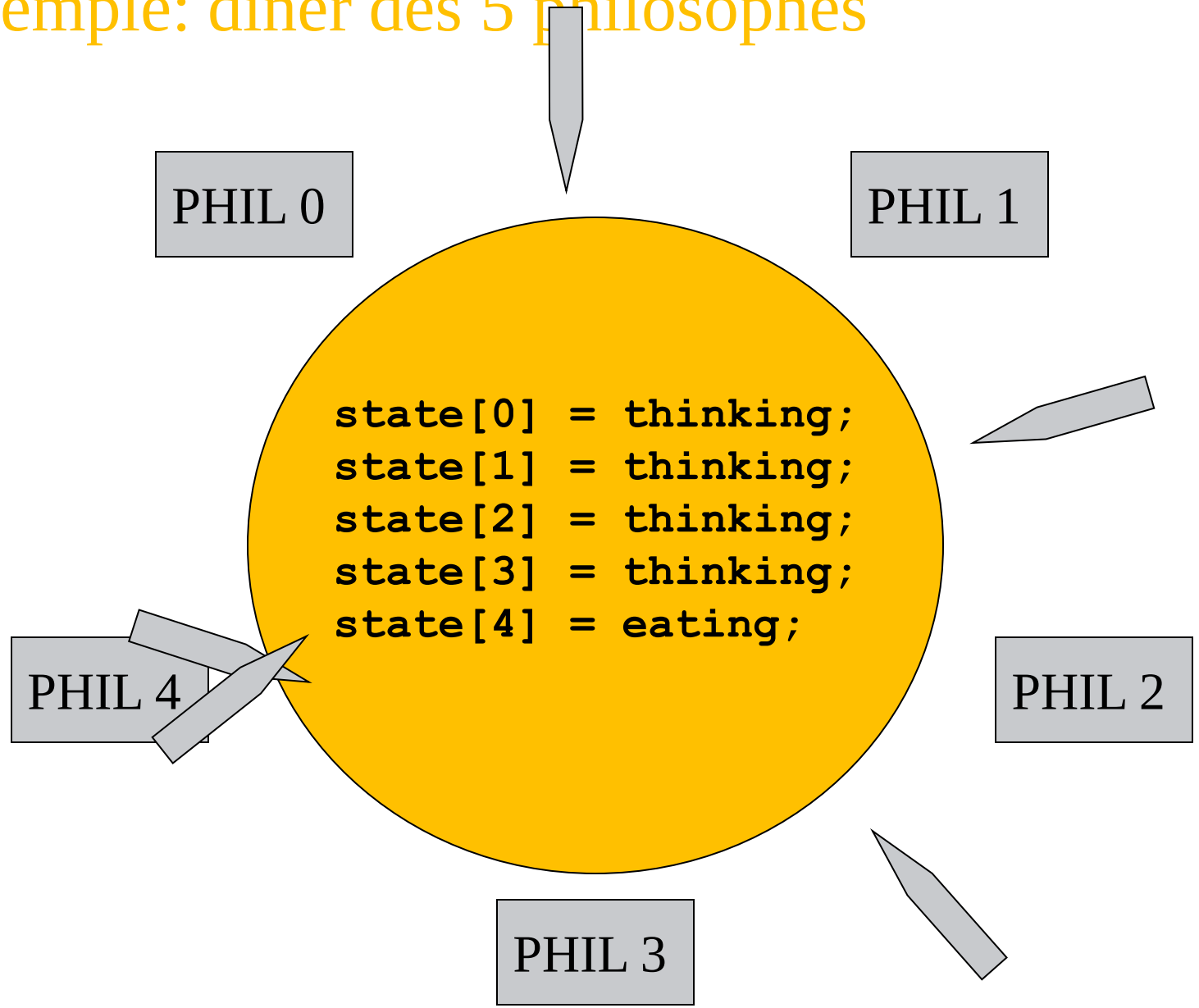
Exemple: dîner des 5 philosophes

```
Moniteur diner_des_5_philosophes {
        int state[5];     // thinking, hungry, eating
        condition self[5];

        procedure pickup(int i) {
                state[i] = hungry;
                test(i);
                if (state[i] != eating) self[i].wait;
        }
        procedure putdown (int i) {
                state[i] = thinking;
                test(i+4 mod 5);
                test(i+1 mod 5);
        }
        procedure test (int k) {
                if ((state[k+4 mod 5] != eating) &&
                                (state[k] == hungry) &&
                                (state[k+1 mod 5] != eating)) {
                        state[k] = eating;
                        self[k].signal;
                }
        }

        {for (int i=0 ; i<5 ; i++) state[i] = thinking;}
}
```

Exemple: dîner des 5 philosophes



Exemple: dîner des 5 philosophes



Exclusion mutuelle en Java

- Un moniteur est associé à chaque objet
- Mot-clé "synchronized"
- Primitive "wait"
- Primitive "notify"
- Primitive "notifyAll"

Synchronized

- Protection d'une zone de code

```
public synchronized void changer()
{
        val++;
}
```

Le verrou est celui de l'instance de la classe

Synchronized

- Protection d'un objet

```
public void changer()
{
        synchronized (this) {
                val++;
        }
}
```

Blocage possible à ce niveau seulement

Le verrou est celui de l'instance de la classe

Méthode wait()

- Suspend le thread
- Libère le moniteur associé
- Attend une notification sur le moniteur associé

Quand le thread reçoit la notification il demande à nouveau au moniteur l'autorisation d'entrer dans le bloc synchronisé.

Méthode wait(long timeout)

- Attente bornée
- Délai en milisecondes

  (éventuellement en nanosecondes)

Après le délai, le thread n'attend plus de notification

Méthode notify()

- Réveil d'un thread (un seul) bloqué sur un wait()

Méthode notifyAll()

- Réveil de tous les threads

  en attente wait() sur le moniteur associé

Les threads "réveillés" entrent alors en concurrence pour l'acquisition du moniteur

Remarques sur notify()

Quand notify() est appelée, aucun processus léger ne peut redémarrer tout de suite.

Il faut d'abord que le thread appelant "notify" sorte du moniteur.

Remarque sur notify()

Une notification peut être perdue si le thread devant être notifié ne s'est pas encore mis en attente.

Fin