Plan du cours

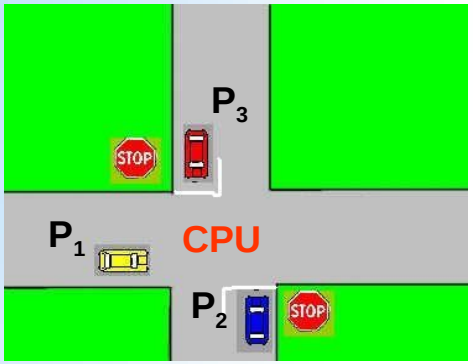
- Fonctions d'un système d'exploitation
- **Partage des ressources et virtualisation**
- IHM et ligne de commande
- Langages de commande

Le problème des ressources partagées

- Plusieurs processus → **accès concurrents** aux ressources

- Une **ressource** désigne toute entité dont a besoin un processus pour s'exécuter :
  - ressource **matérielle** (processeur, mémoire, périphérique...)
  - ressource **logicielle** (variable)

Partage du processeur



Allocation du processeur aux processus

- Comment contrôler l'ordre de passage des processus sur le processeur ?

- Comment contrôler la répartition du temps d'exécution entre les processus ?

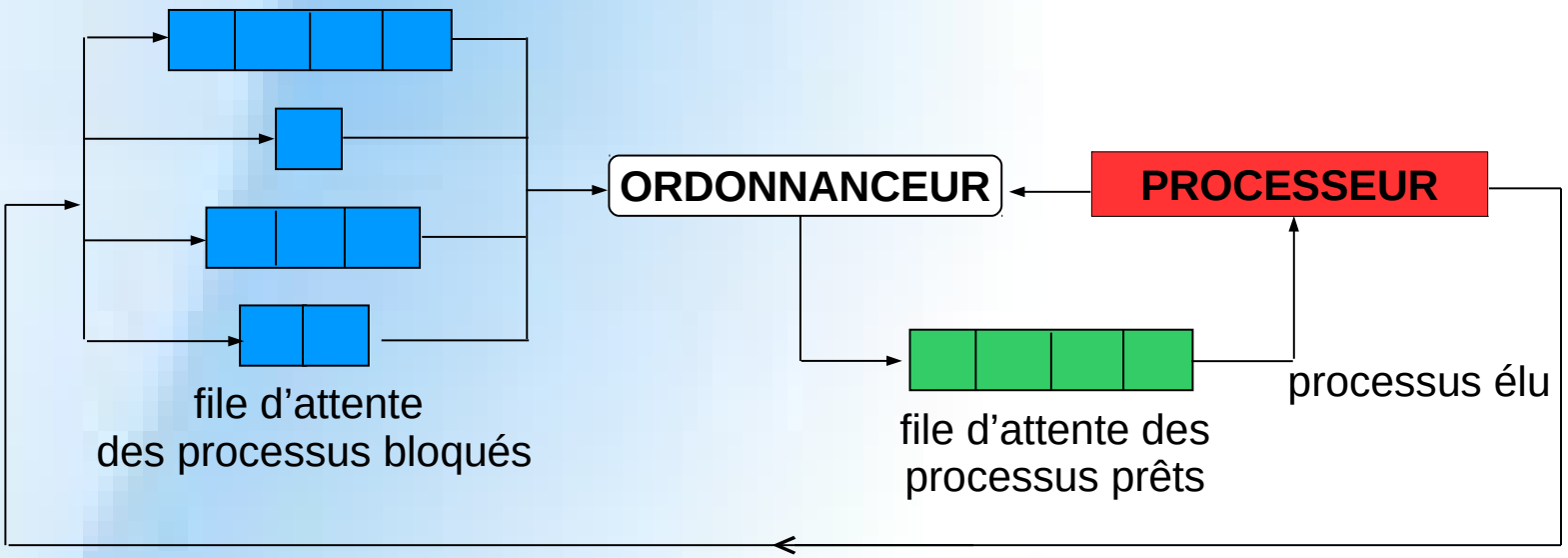
⇒ **L'ordonnanceur**

- L'ordonnanceur est un composant (procédure) du système d'exploitation

Ordonnancement : quels objectifs ?

- L'ordonnancement consiste à :
  - **choisir** le processus à exécuter à un instant *t*
  - **déterminer** le temps durant lequel le processeur lui sera alloué
- **Objectifs d'ordonnancement :**
  - Maximiser le nombre de processus exécutés par unité de temps
  - Minimiser le temps d'attente d'exécution de chaque processus
  - Maximiser le taux d'utilisation des processeurs et autres ressources
  - Favoriser les processus les plus prioritaires
  - Minimiser le nombre et la durée des changements de contexte

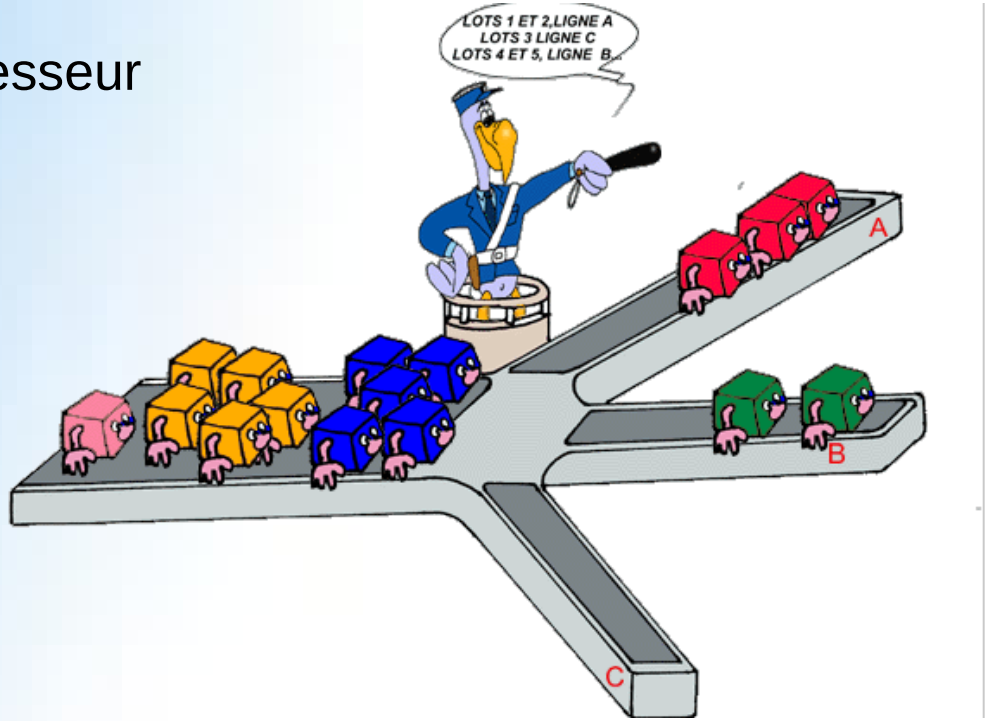
Ordonnancement des processus



- **ORDONNANCEUR** : alloue le processeur aux différents processus selon un algorithme d'ordonnancement donné

Typologie des algorithmes d'ordonnancement

- Monoprocesseur / multiprocesseur
- En-ligne / Hors-ligne
- Préemptif / Non préemptif
- Oisif / Non oisif



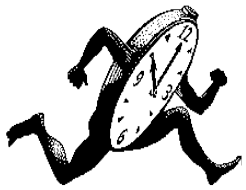
Ordonnancement non préemptif

- Ordonnancement **selon l'ordre d'arrivée** :
  - premier arrivé, premier servi

  First Come First Served (FCFS)

- Ordonnancement **selon la durée de calcul** :
  - travail le plus court d'abord

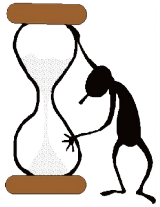
  Shortest Job First (SJF)



Ordonnancement préemptif

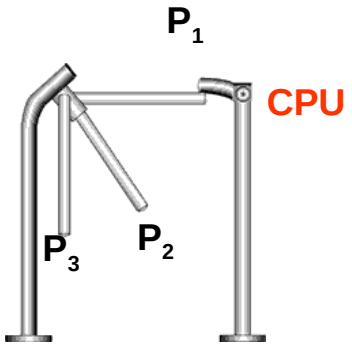
- Ordonnancement **selon la durée de calcul restante** :
  - temps restant le plus court d'abord

  Shortest Remaining Time (SRT)

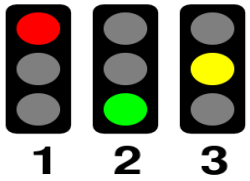


- Ordonnancement **sans notion de priorité** :
  - temps-partagé avec politique du tourniquet

  Round-Robin (RR)

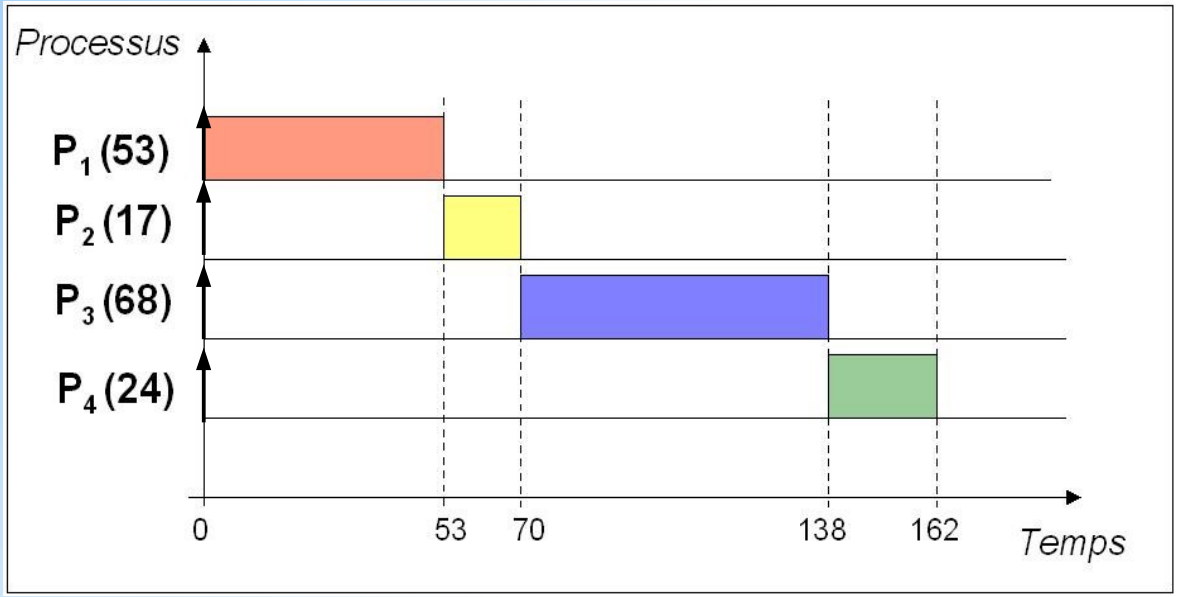


- Ordonnancement **à priorités** (statiques ou dynamiques) :
  - la tâche la plus prioritaire obtient le processeur



Ordonnancement First Come First Served (FCFS)

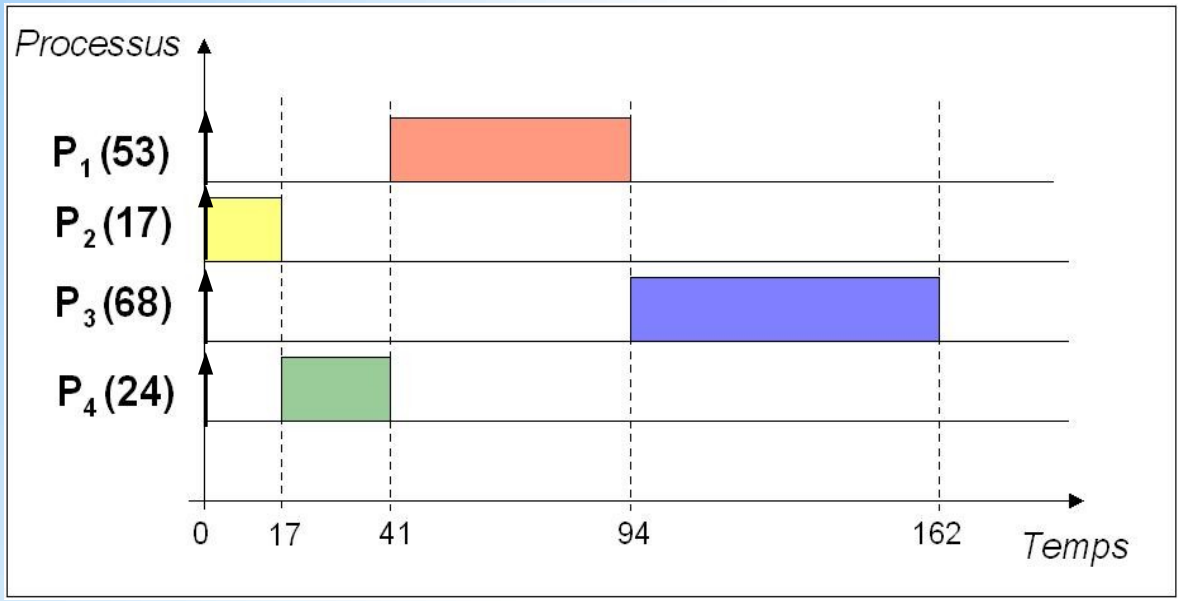
- **Principe** : Les processus sont ordonnancés selon leur ordre d'arrivée
- **Exemple** (P_1, P_2, P_3 et P_4 arrivent dans cet ordre à t = 0) :



Ordonnancement Shortest Job First (SJF)

- **Principe** : Le processus dont le temps d'exécution est le plus court est ordonnancé en priorité

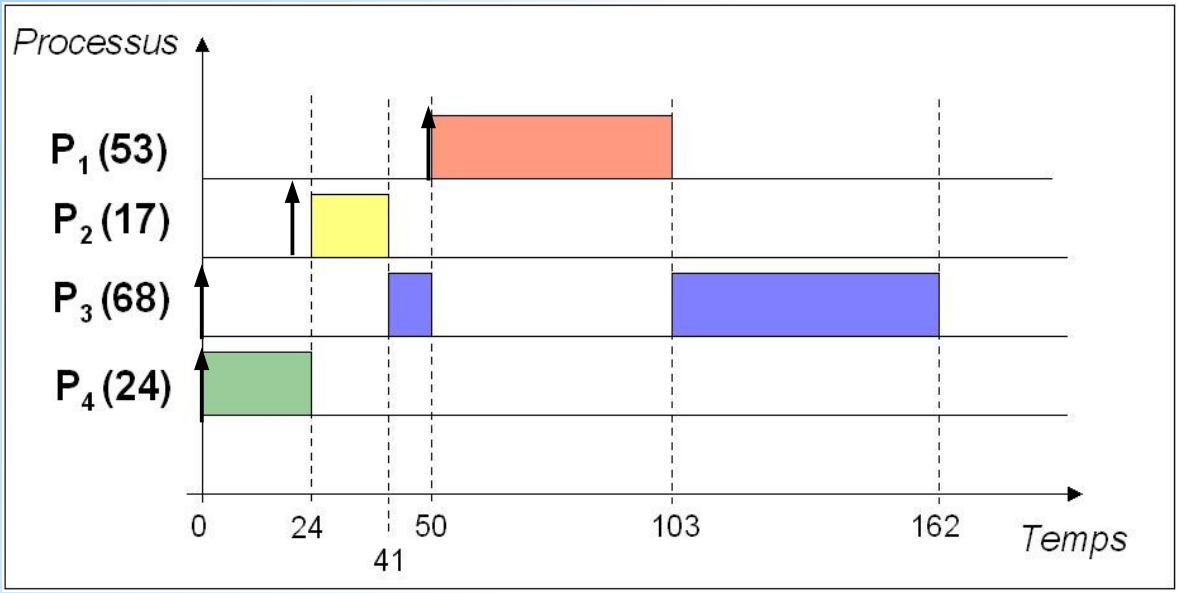
- **Exemple** (P_1, P_2, P_3 et P_4 arrivent à t = 0) :



Ordonnancement Shortest Remaining Time (SRT)

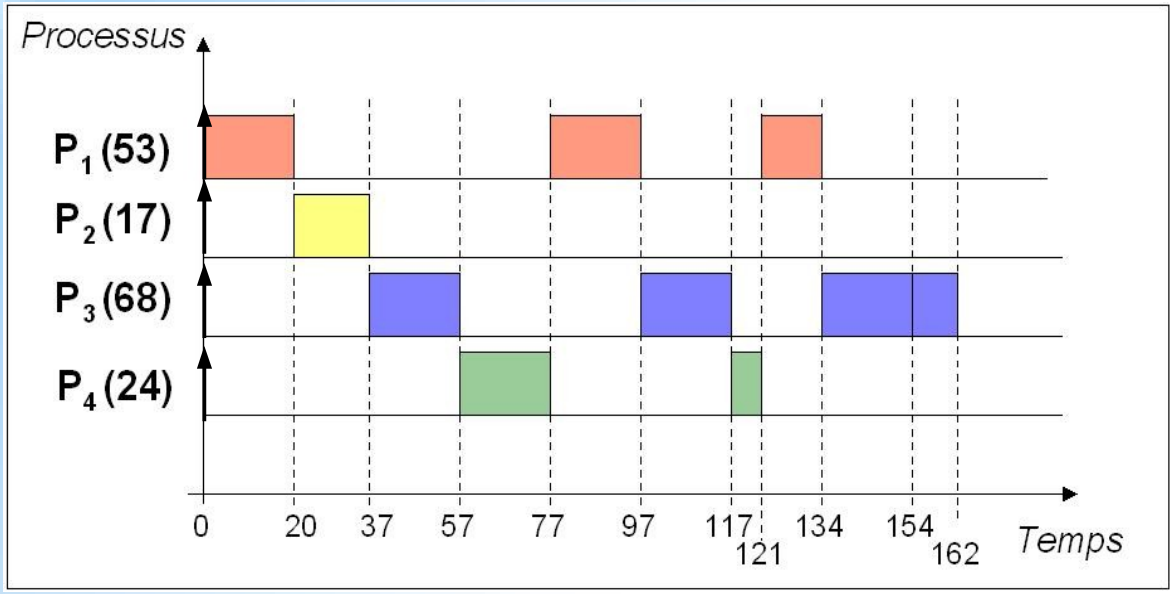
- **Principe** : Le processus dont le temps d'exécution restant est le plus court parmi ceux qui restent à exécuter, est ordonnancé en premier

- **Exemple** (P_3 et P_4 arrivent à t = 0; P_2 à t = 20; P_1 à t = 50) :



Ordonnancement temps-partagé (Round-Robin)

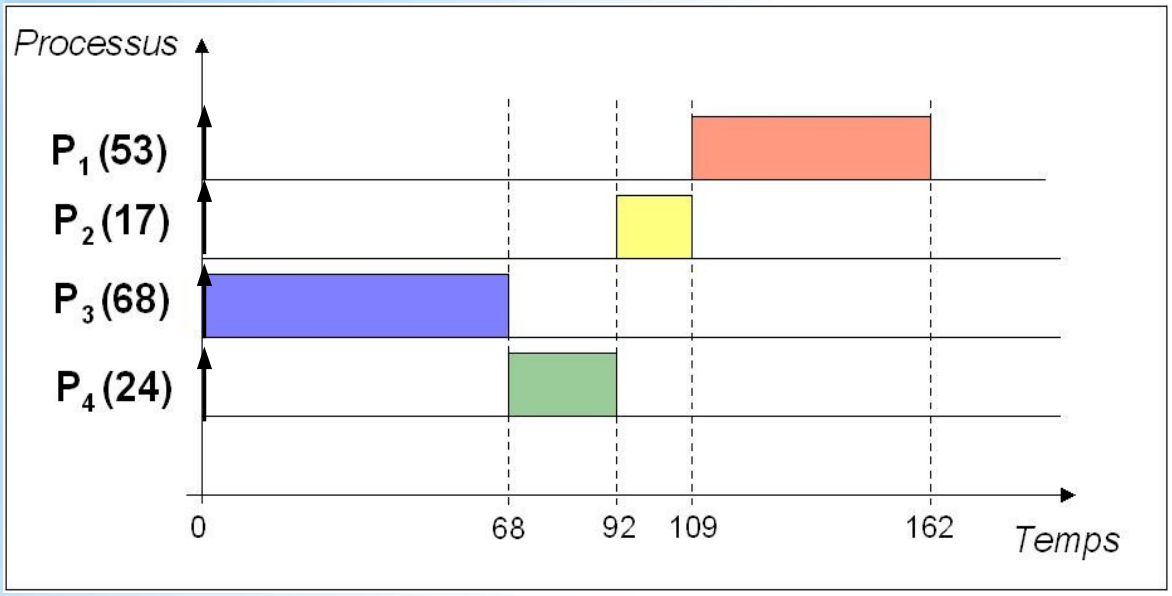
- **Principe** : allocation du processeur par tranche (quantum) de temps
- **Exemple** (q=20, n=4) :



- Chaque tâche obtient le processeur au bout de **(n-1)*q** unités de temps au plus

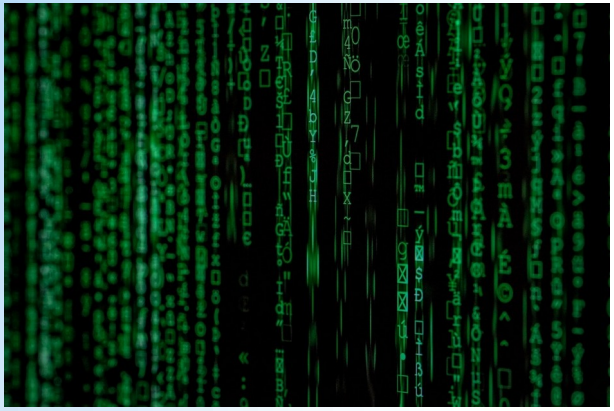
Ordonnancement à priorités statiques

- **Principe** : allocation du processeur selon des priorités statiques (numéros affectés aux processus pour toute la vie de l'application)
- **Exemple** (*priorités*(P_1, P_2, P_3, P_4)=(3,2,0,1)) :



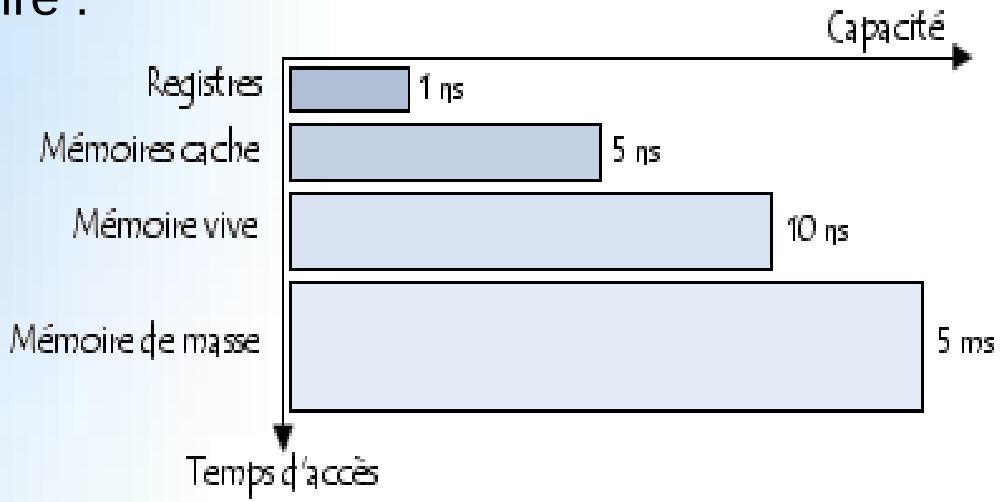
- Dans certains systèmes, l'échelle des priorités est inversée (0 est alors la priorité la plus faible)

Partage de la mémoire



Quelle mémoire ?

- 2 niveaux de gestion de la mémoire :



  - **Niveau matériel**

    les registres du processeur

    la mémoire cache

  - **Niveau système d'exploitation**

    la mémoire principale (appelée également *mémoire centrale ou interne*)

    la mémoire secondaire (appelée également *mémoire de masse ou physique*)

Rôle du gestionnaire de mémoire du SE

Le gestionnaire de mémoire du SE

- **Objectifs du gestionnaire de mémoire** du système d'exploitation :
  - **Partager la mémoire** (système multi-tâche)
  - **Allouer des blocs de mémoire** aux différents processus
  - **Protéger les espaces mémoire** utilisés
  - **Optimiser la quantité de mémoire disponible**

La mémoire virtuelle (1)

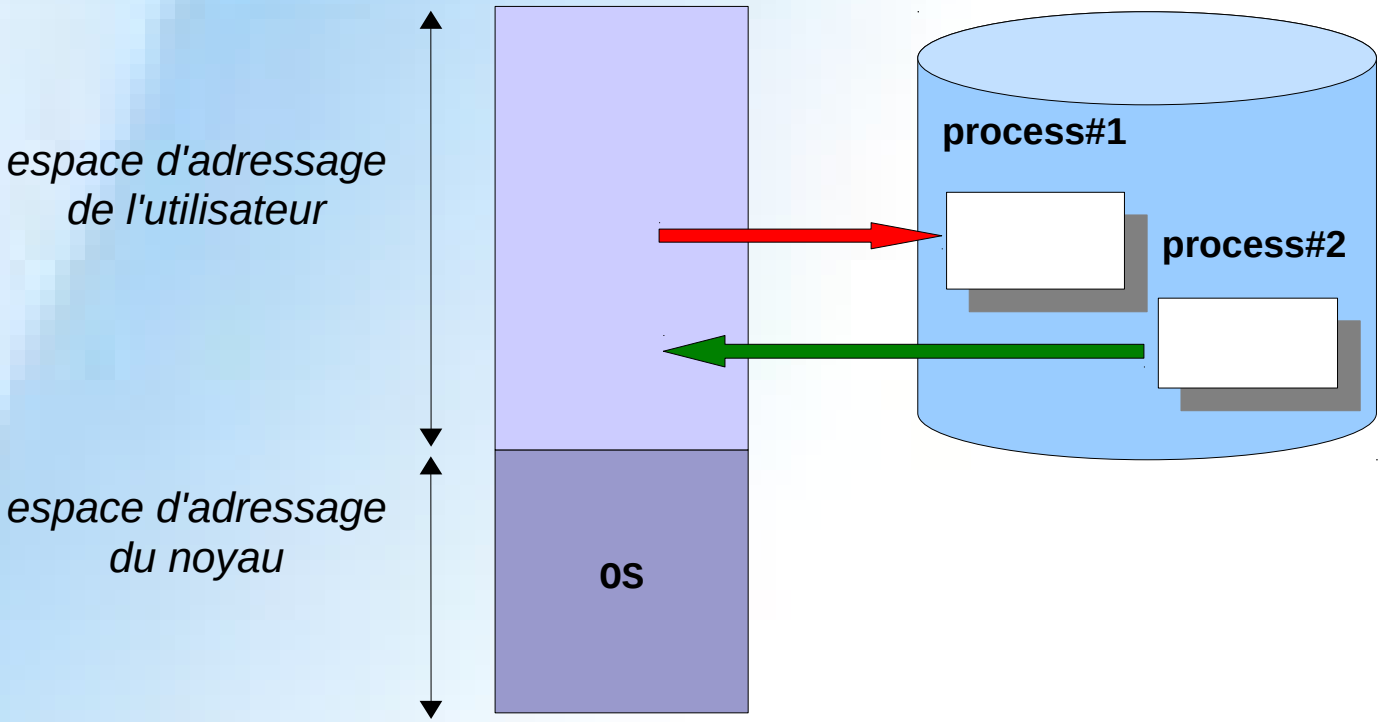
- La **mémoire virtuelle** est une technique permettant d'exécuter des programmes dont la taille excède la taille de la mémoire réelle

- La **mémoire virtuelle** permet :
  - d'augmenter le nombre de processus présents simultanément en mémoire centrale
  - de mettre en place des mécanismes de protection mémoire
  - de partager la mémoire entre processus

- Mécanisme mis au point dans les années 60

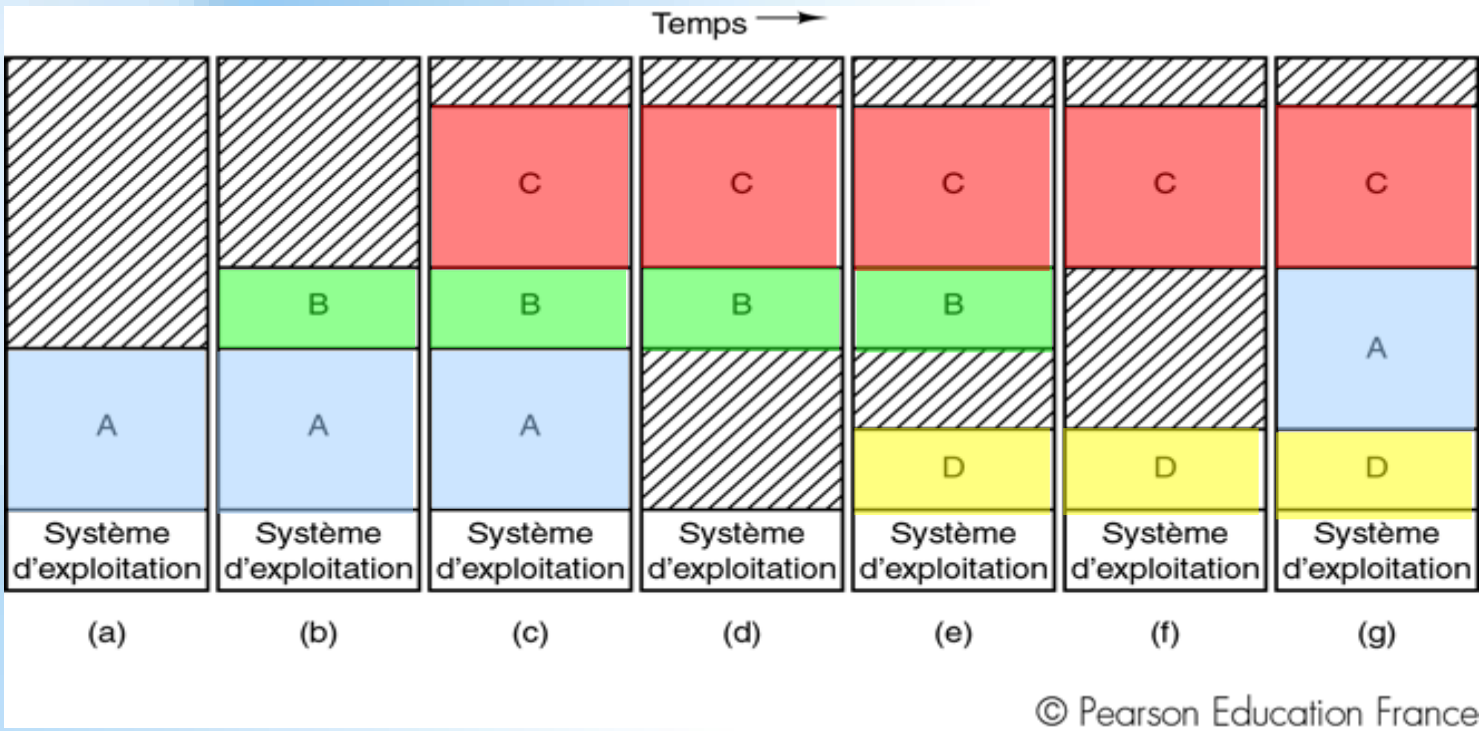
La mémoire virtuelle (2)

- Une partie de l'espace d'adressage d'un processus peut être enlevé temporairement de la mémoire centrale au profit d'un autre (**swapping**)



La mémoire virtuelle (3)

- Illustration du mécanisme de **va-et-vient** (**swapping**)



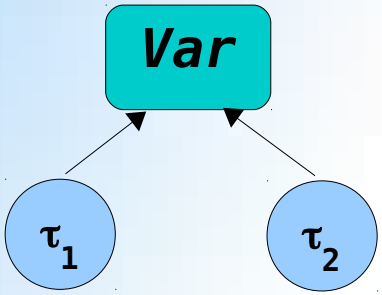
Les mécanismes de découpage de la mémoire

- La mémoire principale peut être découpée de 3 façons :
  - **par pagination**
    elle consiste à diviser la mémoire en blocs, et les programmes en pages de **longueur fixe**.
  - **par segmentation**
    les programmes sont découpés en parcelles ayant des **longueurs variables** appelées *segments*.
  - **par segmentation paginée**
    certaines parties de la mémoire sont segmentées, d'autres paginées

Partage de variables

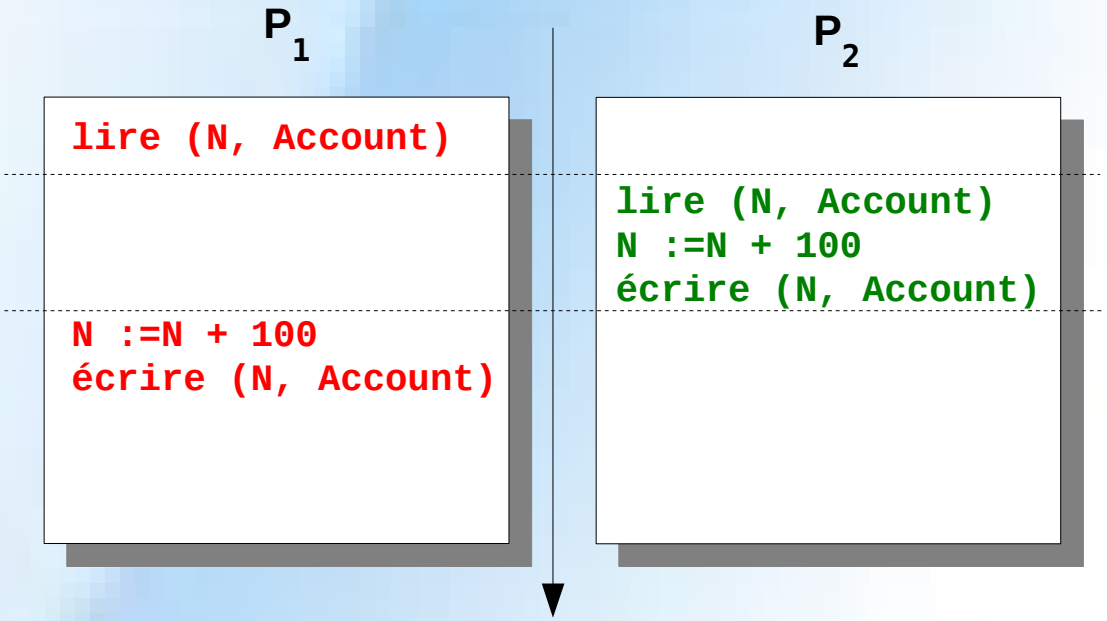
Pourquoi synchroniser les exécutions ?

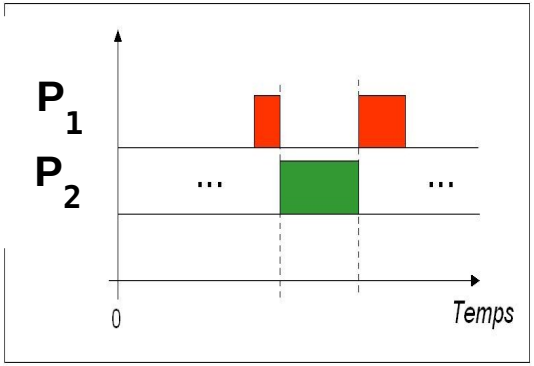
- Garantir la cohérence des variables partagées



Garantir la cohérence des variables partagées

- Problème de la synchronisation relative de l'exécution des processus = cas d'**incohérence de données** :

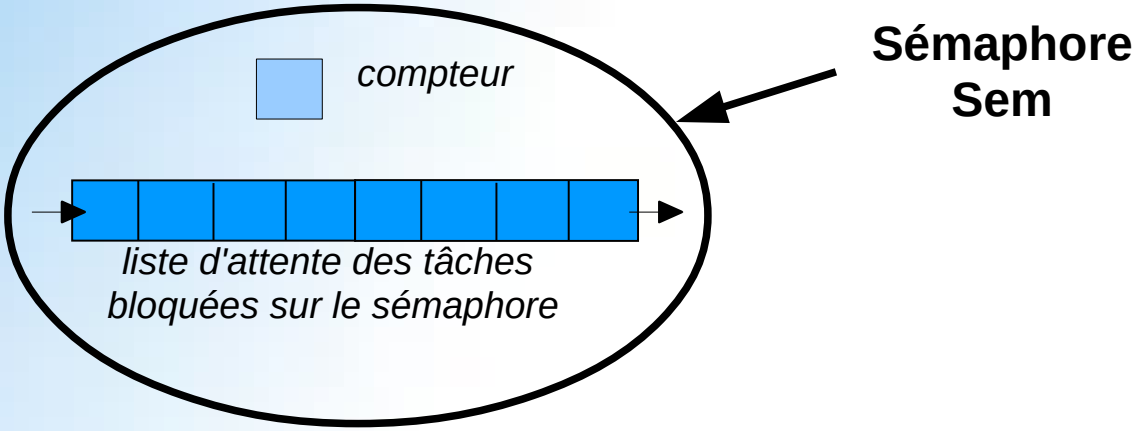




- P_2 ne doit pas accéder à N tant que P_1 l'utilise !

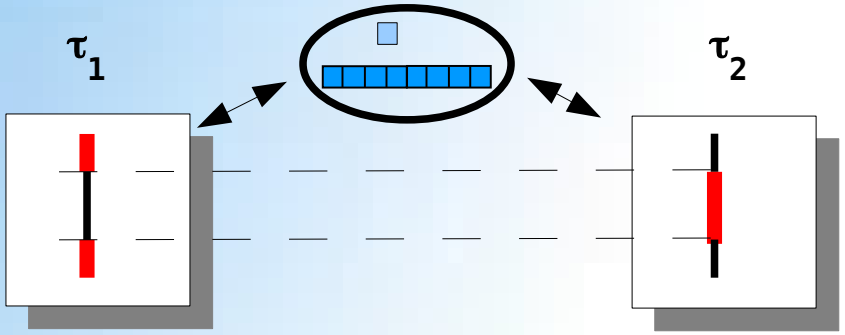
Solution : les sémaphores (1)

- Un sémaphore est une **structure de données**
  - contenant un compteur (valeur entière non négative)
  - gérant une file d'attente de tâches attendant qu'advienne une condition particulière propre au sémaphore



Solution : les sémaphores (2)

- Mise en œuvre de l'**exclusion mutuelle**

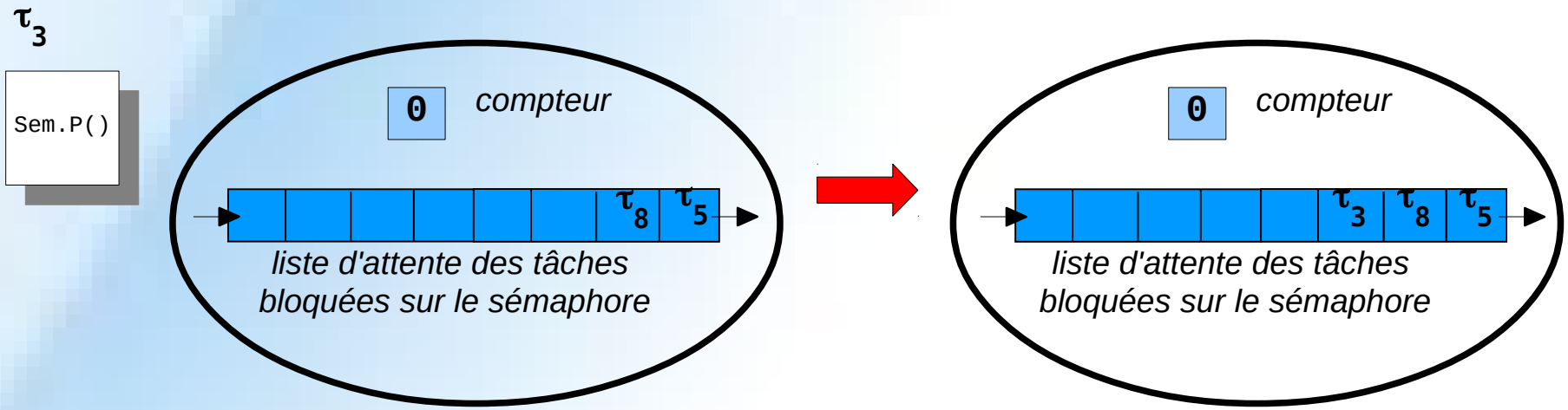


Section critique

Principe des sémaphores (1)

- 1ère opération : **test de prise** du sémaphore (« Puis-je ?)

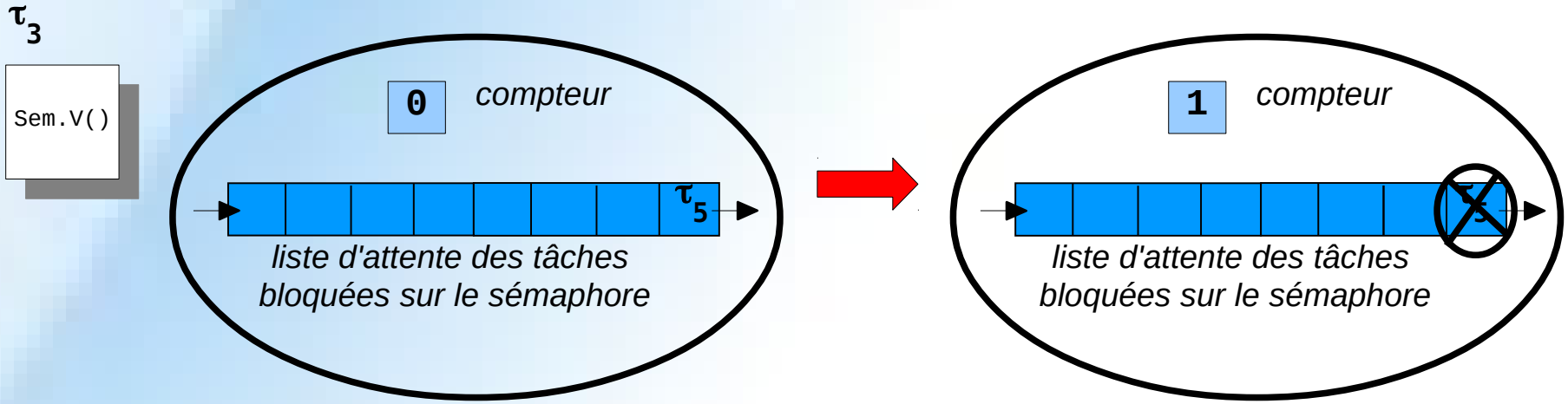
```
Sem.P() :     si (Sem.compteur>0)
                    alors Sem.compteur = Sem.compteur-1
                    sinon insère_ce_processus(Sem.file)
              finsi
```



Principe des sémaphores (2)

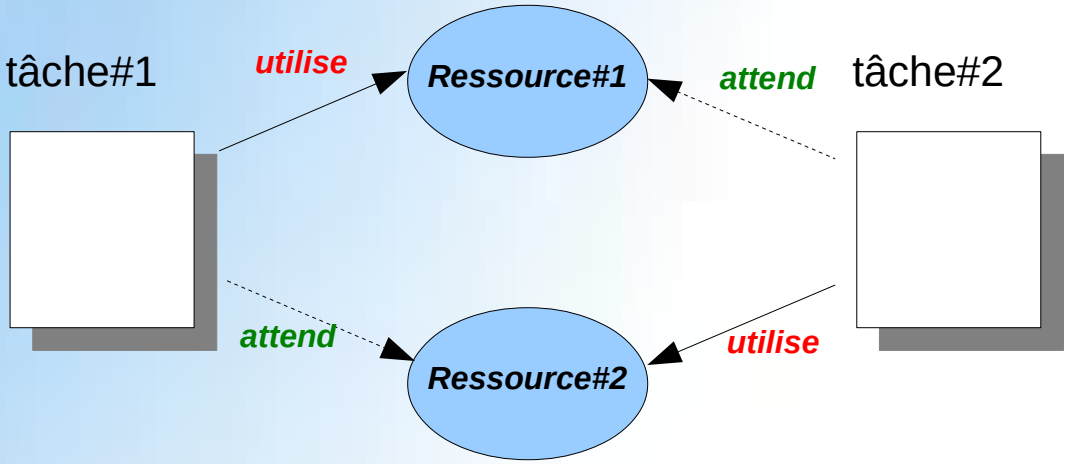
- 2ème opération : **libération** du sémaphore (« Vas-y ! »)

```
Sem.V() : Sem.compteur = Sem.compteur+1
          si (Sem.compteur > 0)
             alors extrait_un_processus(Sem.file)
          finsi
```



Problèmes liés à l'utilisation de sémaphores (1)

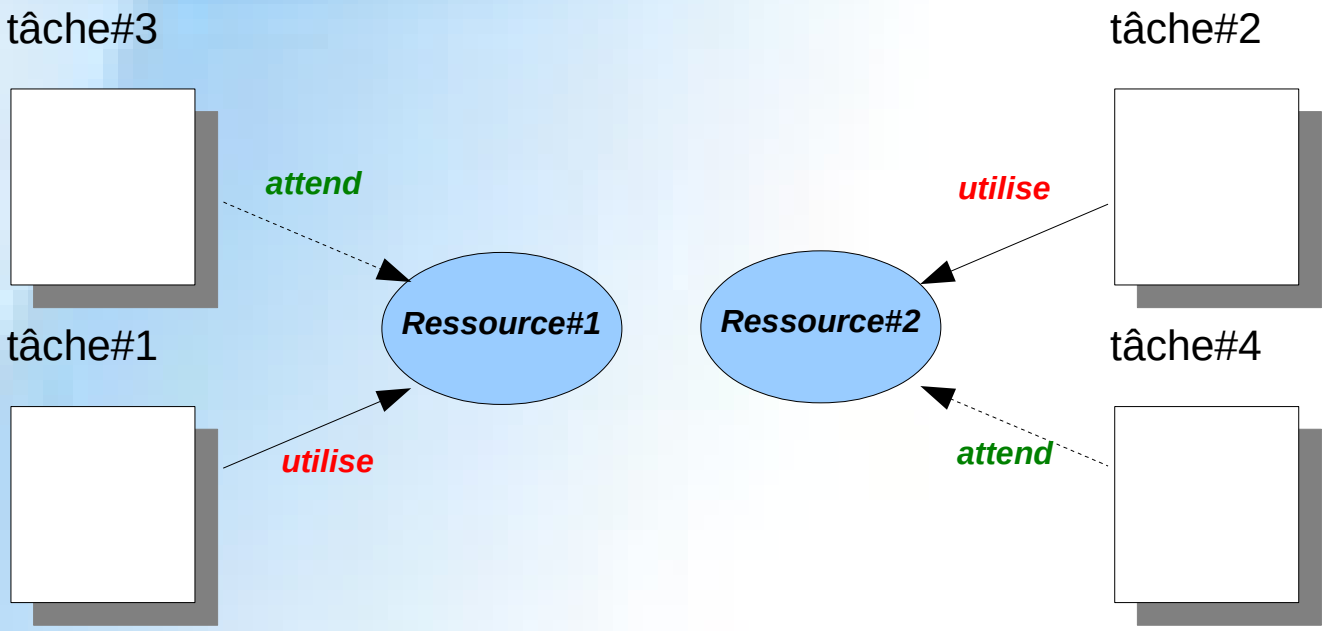
- Cas d'**interblocage (*deadlock*)** : ensemble de tâches attendant chacune une ressource déjà possédée par une tâche de l'ensemble



- L'attente est **infinie**

Problèmes liés à l'utilisation de sémaphores (2)

- Cas de **coalition et famine (*livelock*)** : ensemble de tâche monopolisant des ressources au détriment d'autres tâches.



- L'attente est **indéfinie**