Programmation Système

Département Informatique

IUT2 de Grenoble

BUT2 - Ressource 3.05

Organisation du cours

1. Système d'exploitation
2. Processus
3. Partage des ressources
4. Système de Gestion de Fichiers
5. Entrées/Sorties

3. Partage des ressources

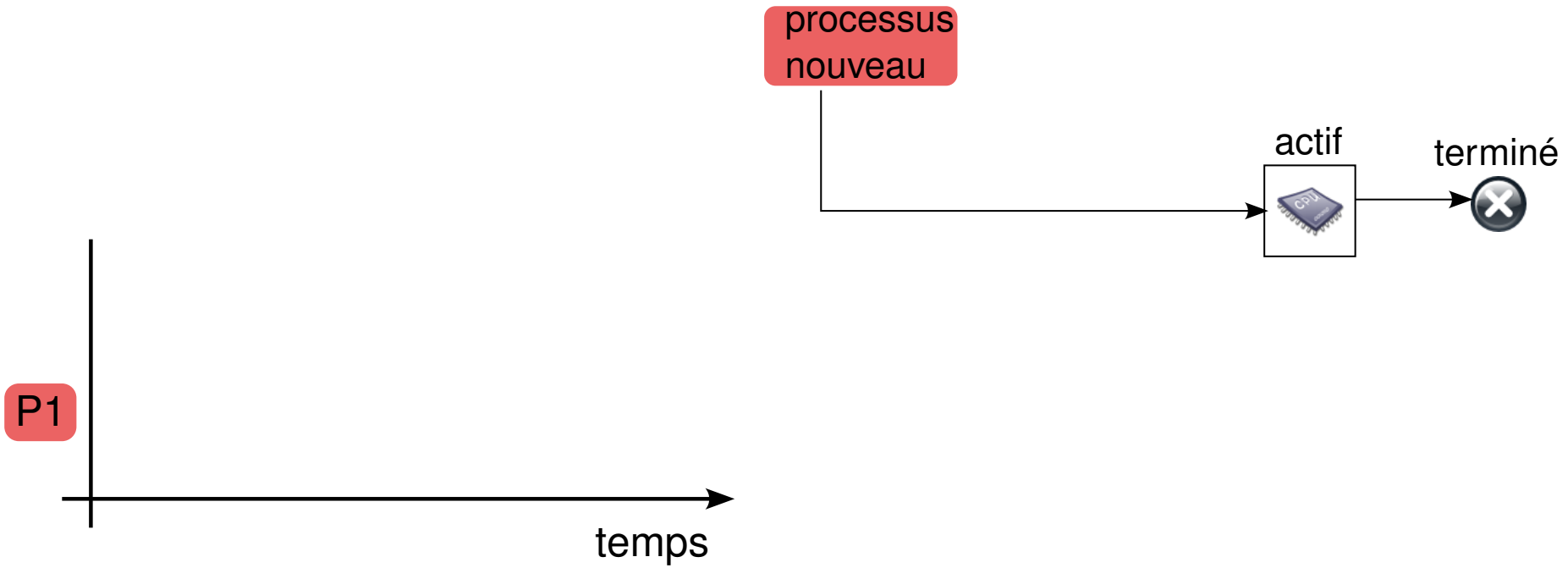
3.1. Partage du CPU

- Etats d'un processus
- Ordonnancement de processus

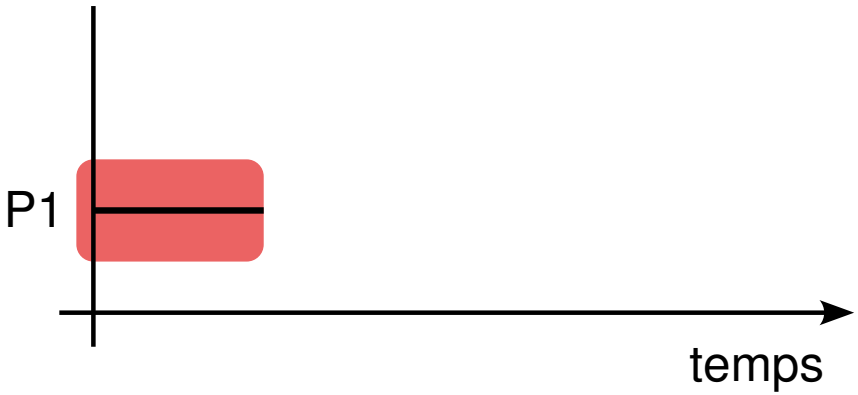
3.2. Partage de la mémoire centrale

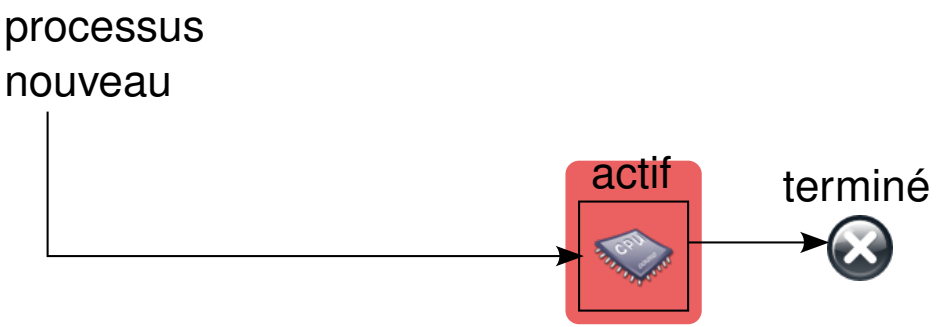
- Pagination
- Mémoire virtuelle

Vie d'un processus : exécution séquentielle

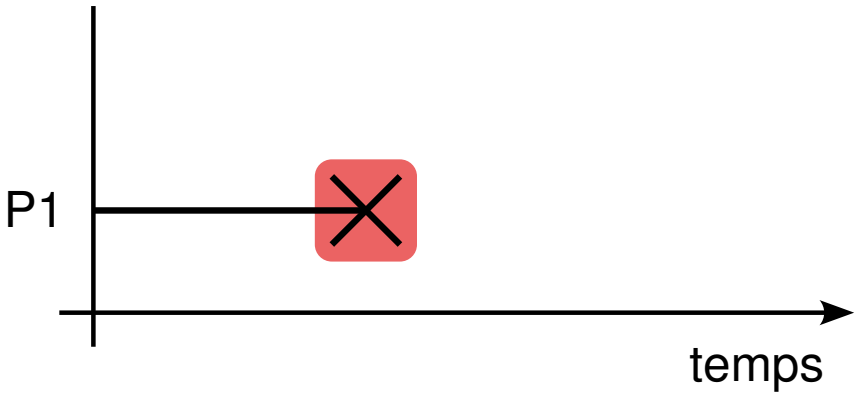


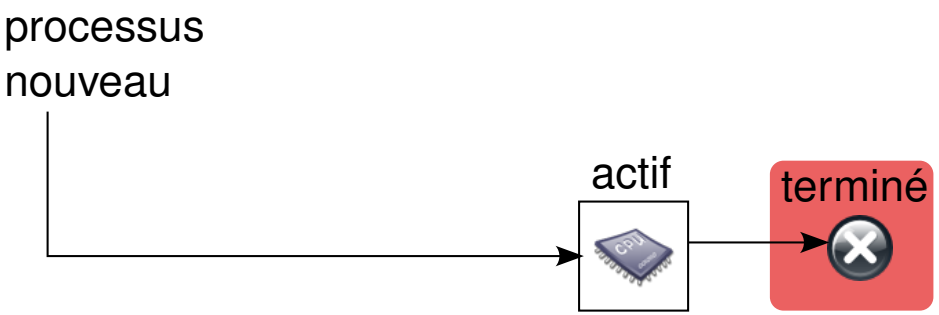
Vie d'un processus : exécution séquentielle



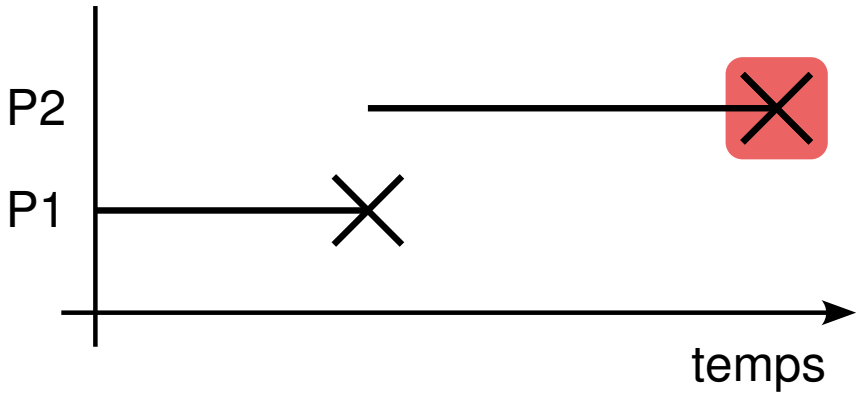


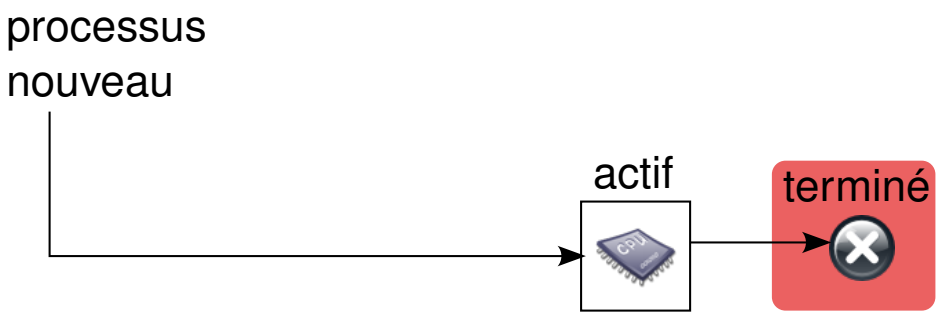
Vie d'un processus : exécution séquentielle



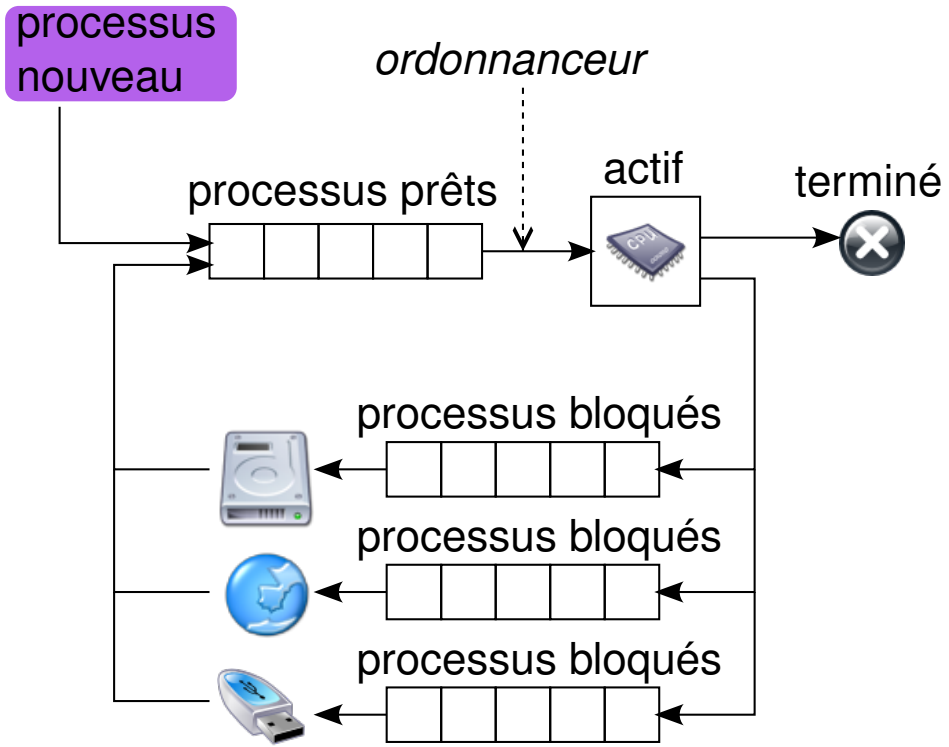


Vie d'un processus : exécution séquentielle

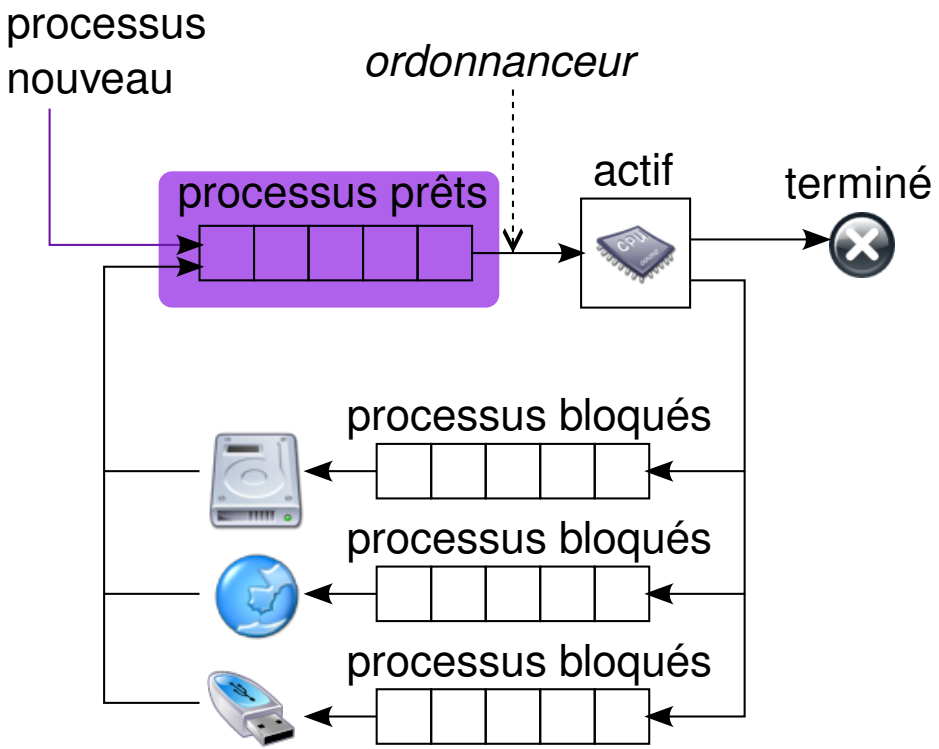




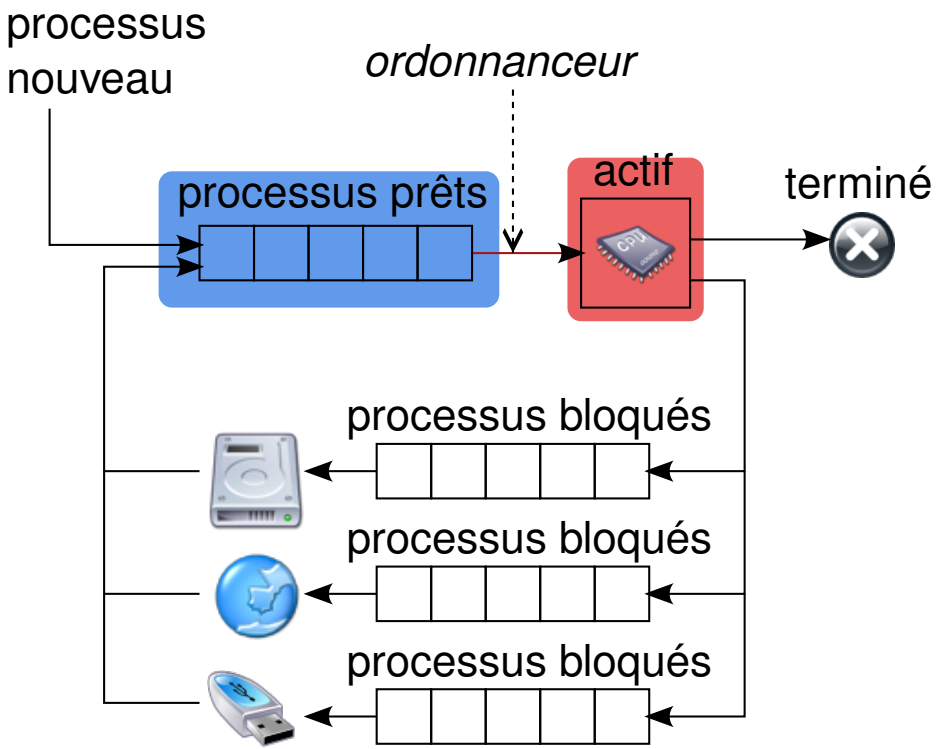
Vie d'un processus : multi-programmation



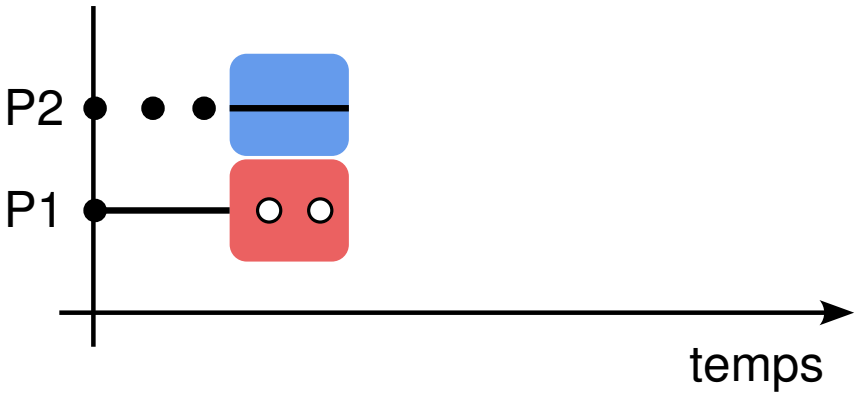
Vie d'un processus : multi-programmation

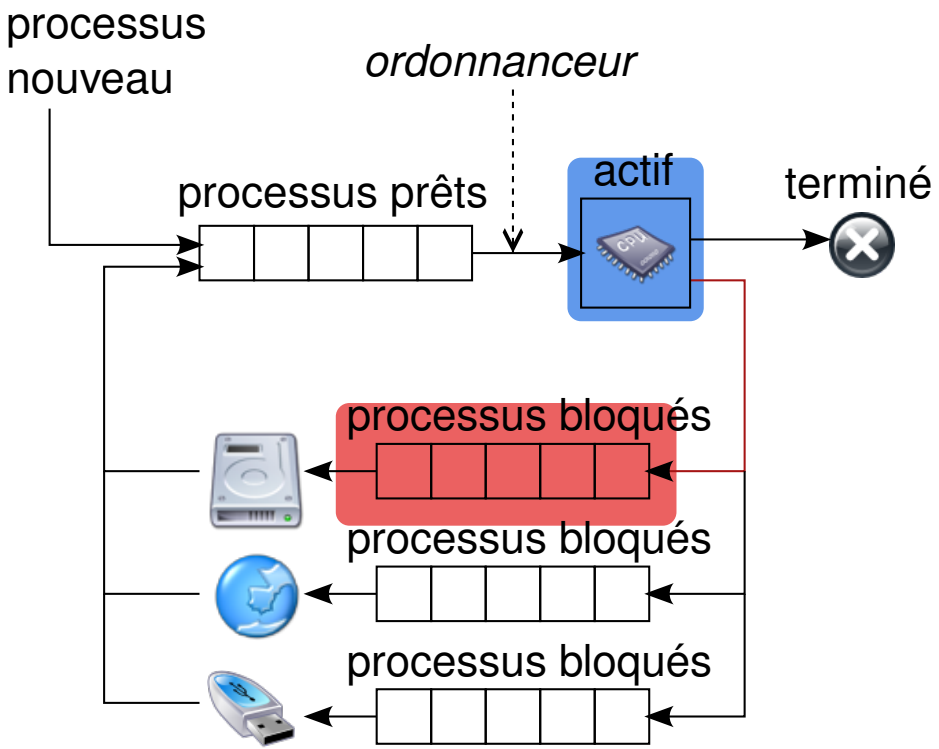


Vie d'un processus : multi-programmation

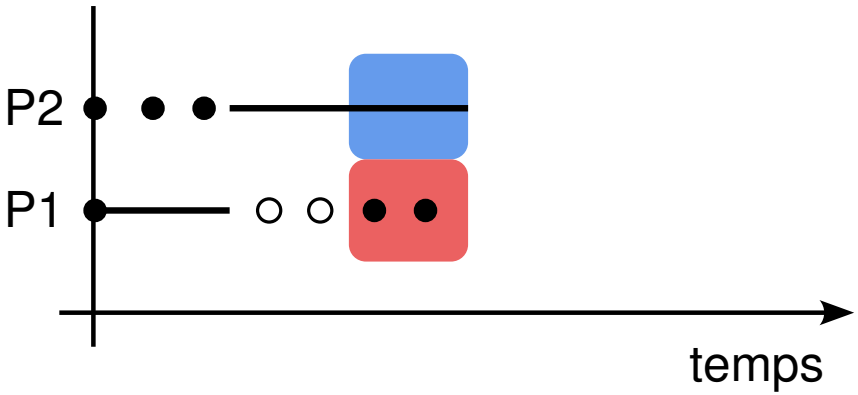


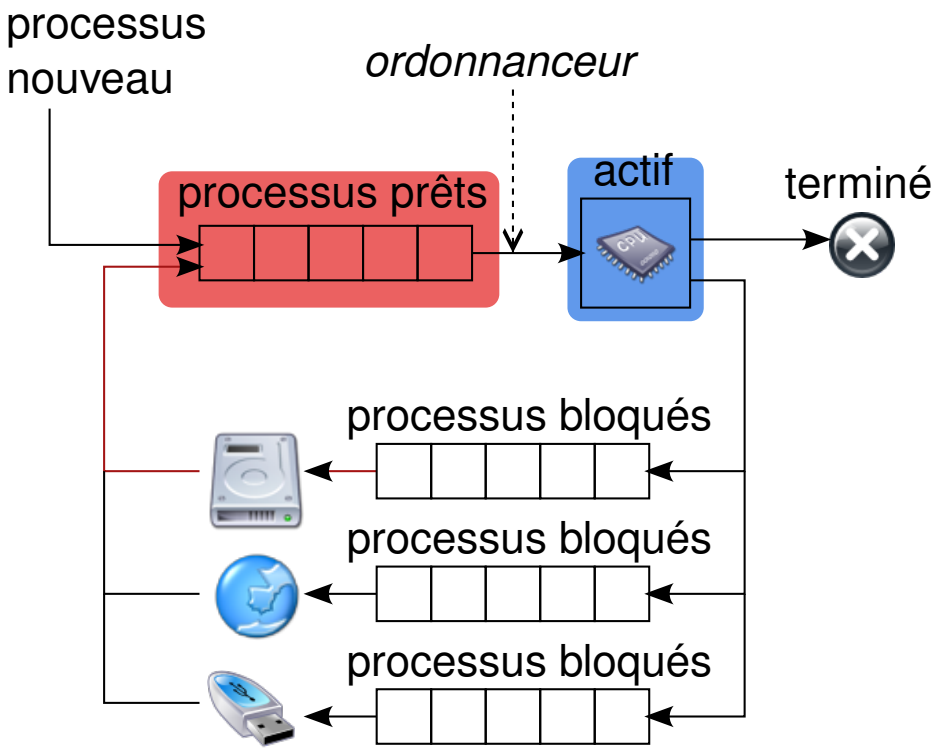
Vie d'un processus : multi-programmation



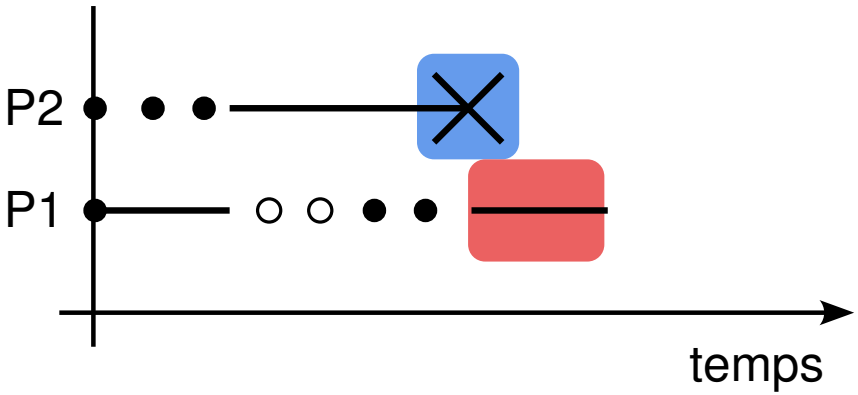


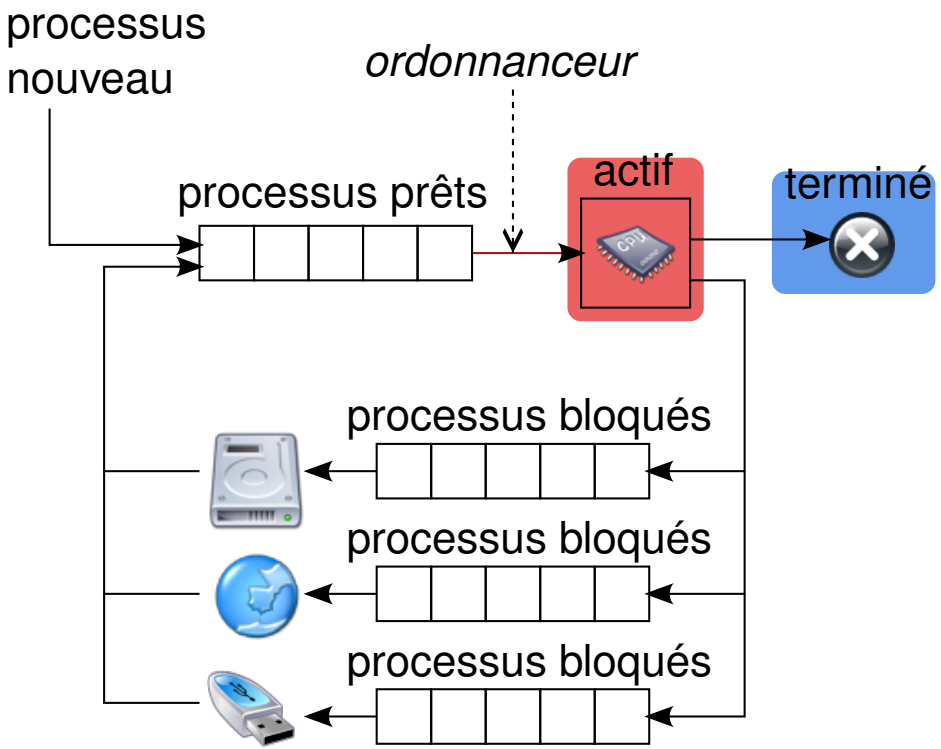
Vie d'un processus : multi-programmation



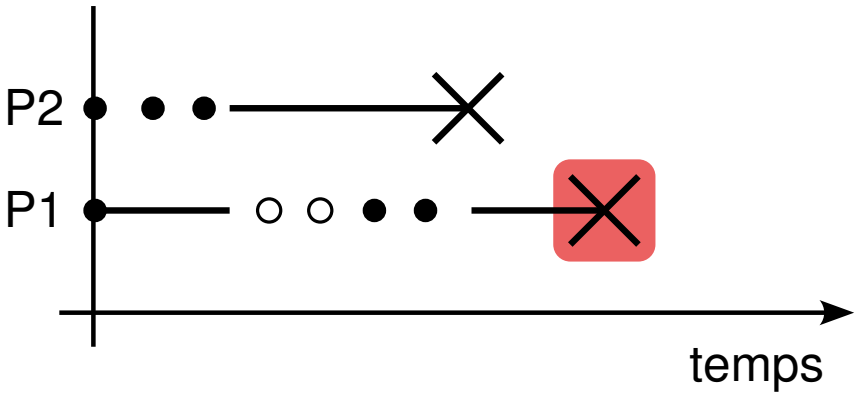


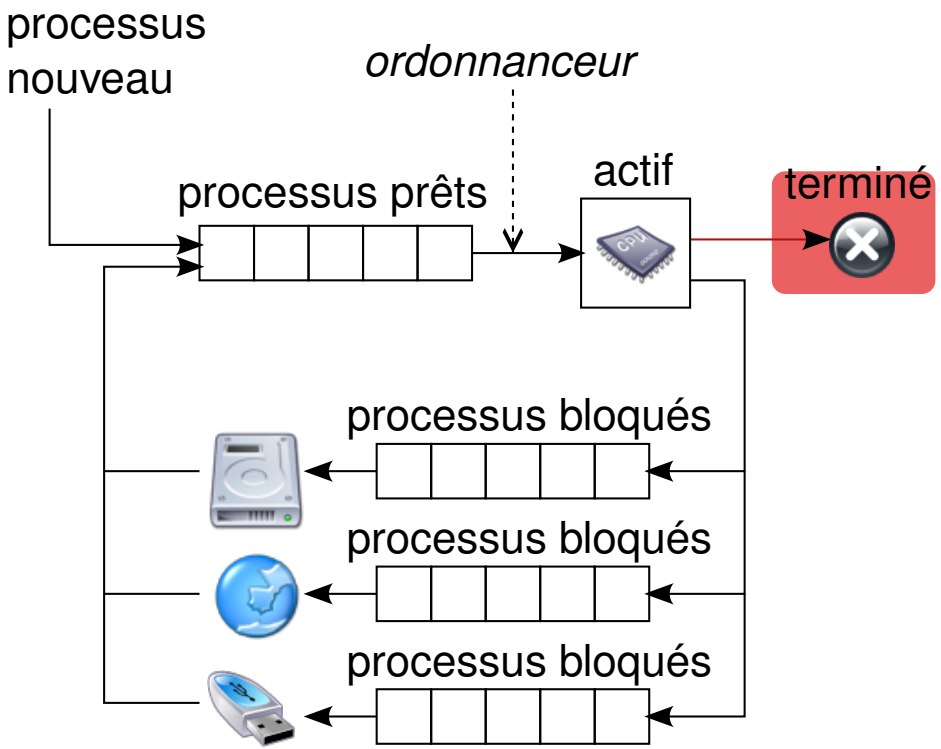
Vie d'un processus : multi-programmation



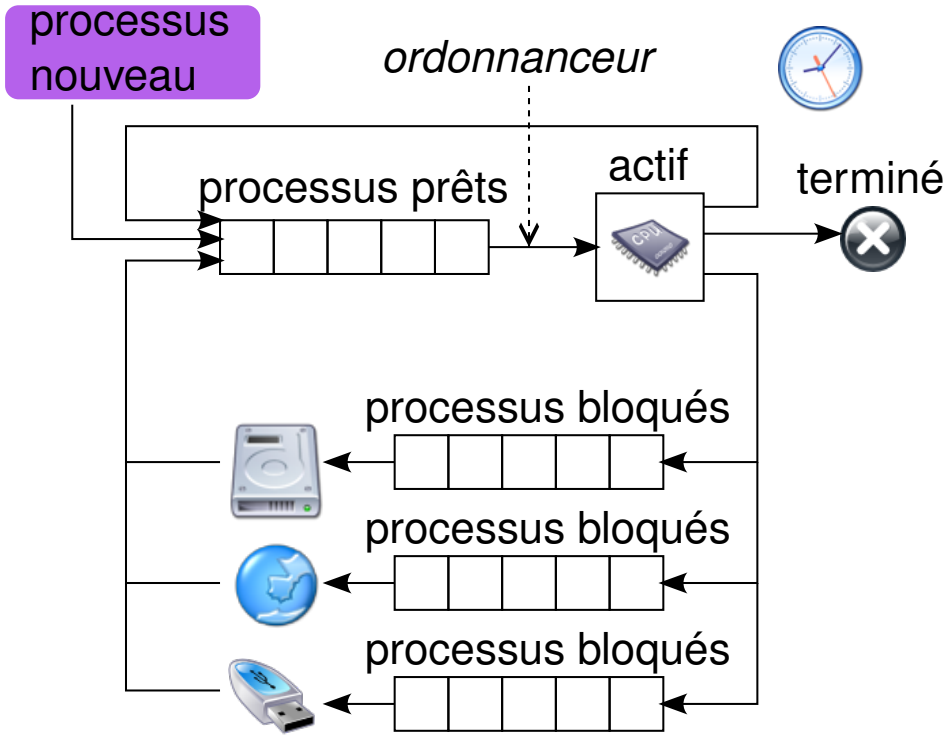


Vie d'un processus : multi-programmation

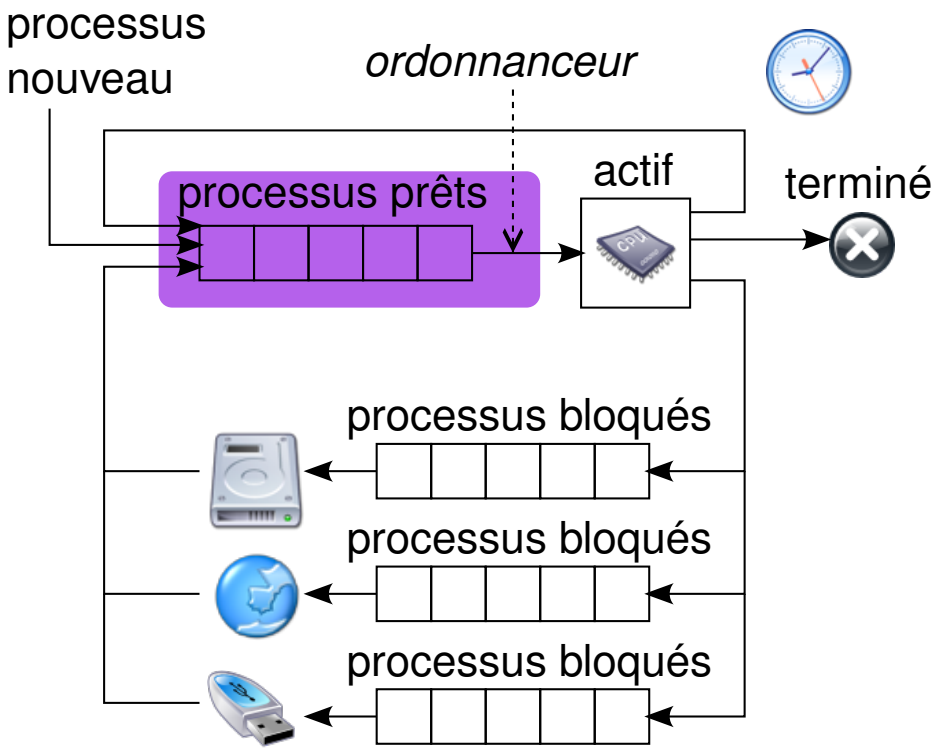




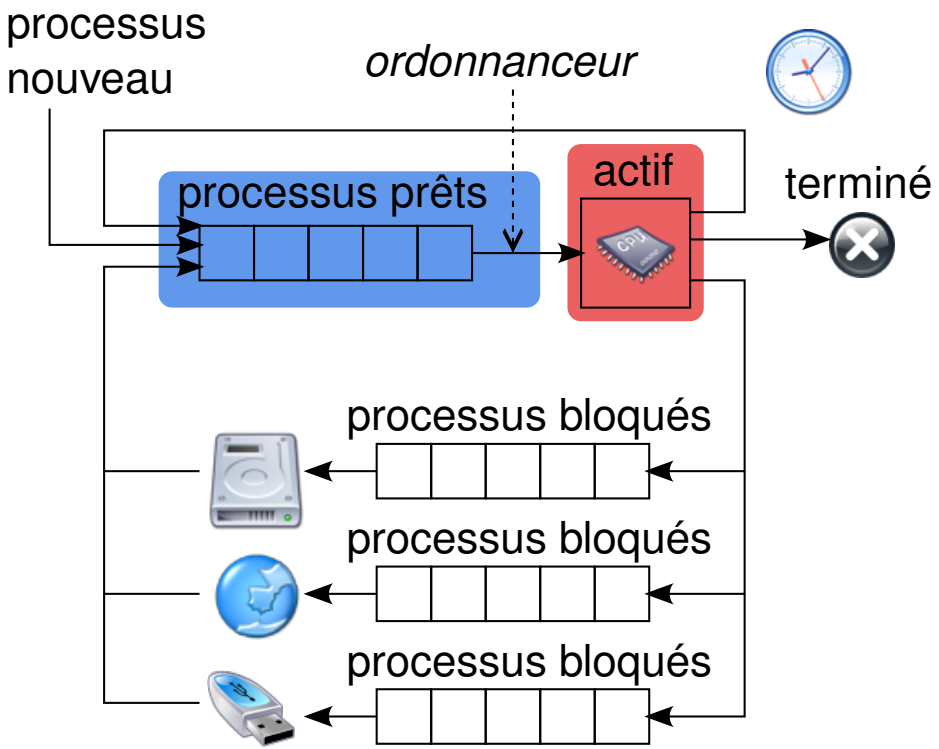
Vie d'un processus : préemption et partage du temps



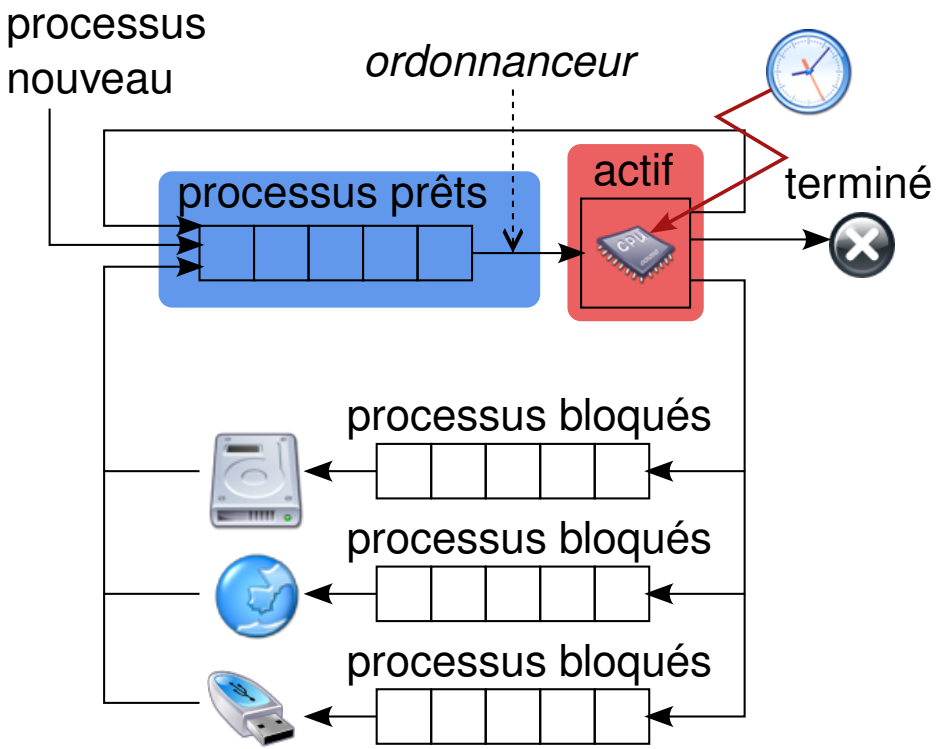
Vie d'un processus : préemption et partage du temps



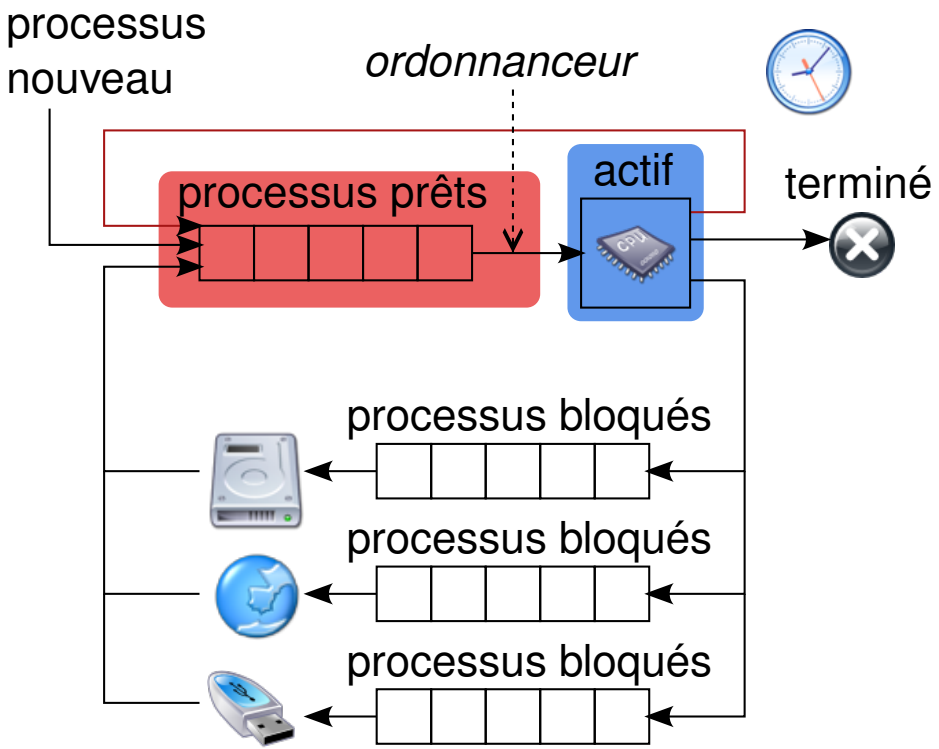
Vie d'un processus : préemption et partage du temps



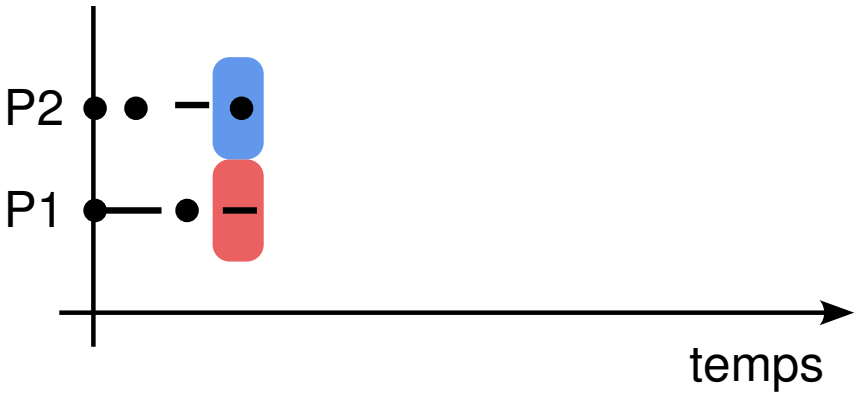
Vie d'un processus : préemption et partage du temps

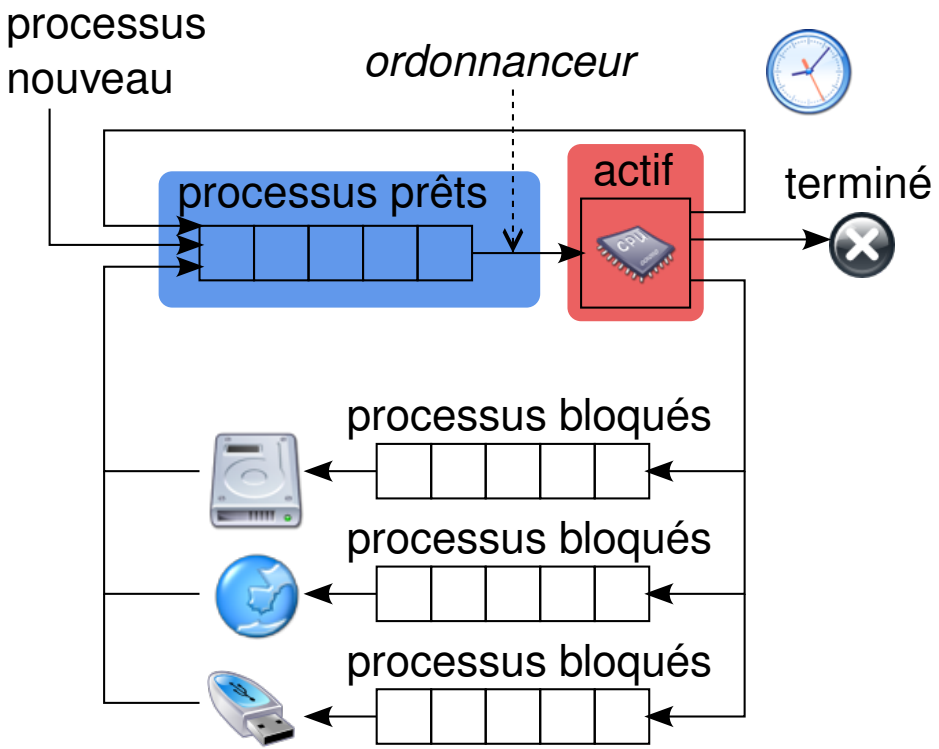


Vie d'un processus : préemption et partage du temps

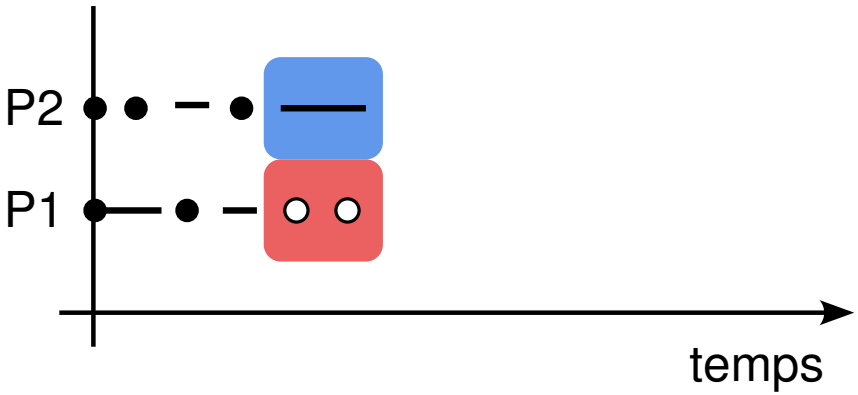


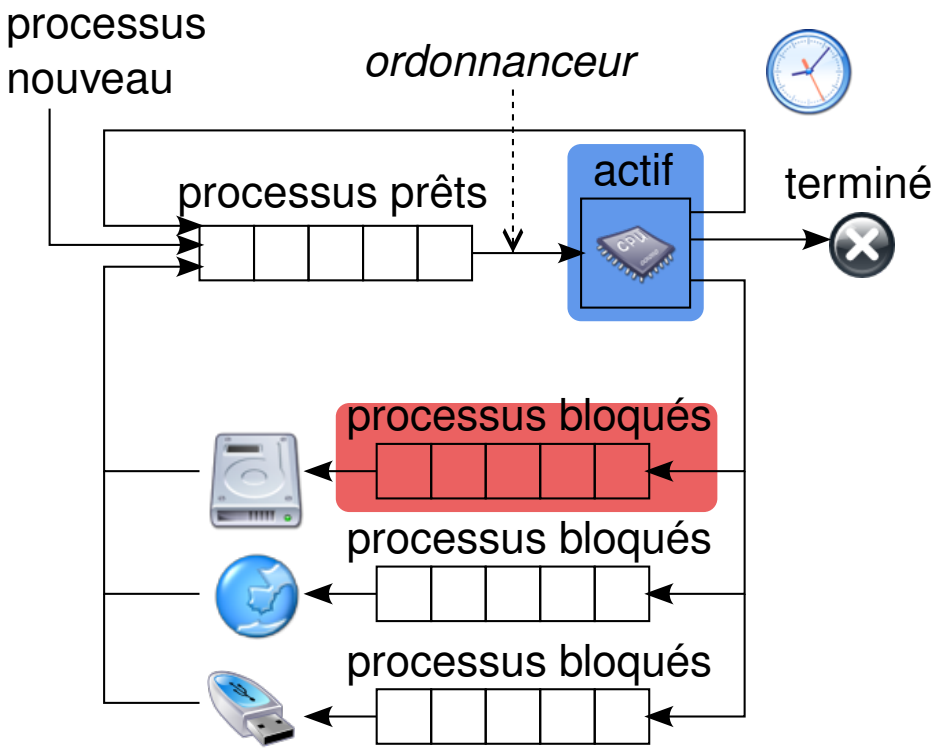
Vie d'un processus : préemption et partage du temps



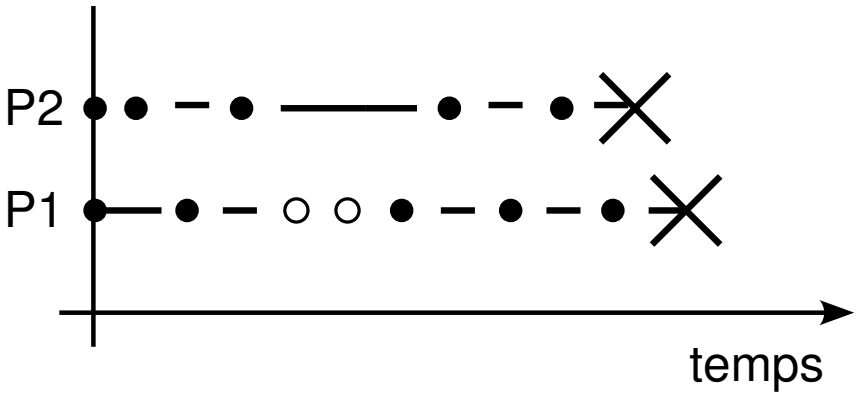


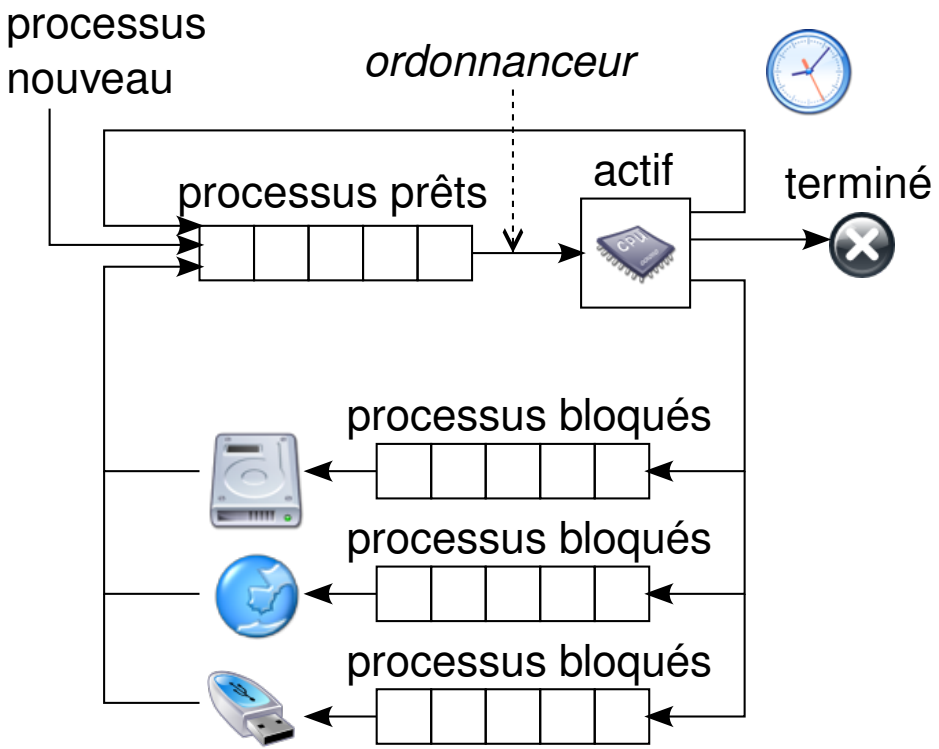
Vie d'un processus : préemption et partage du temps





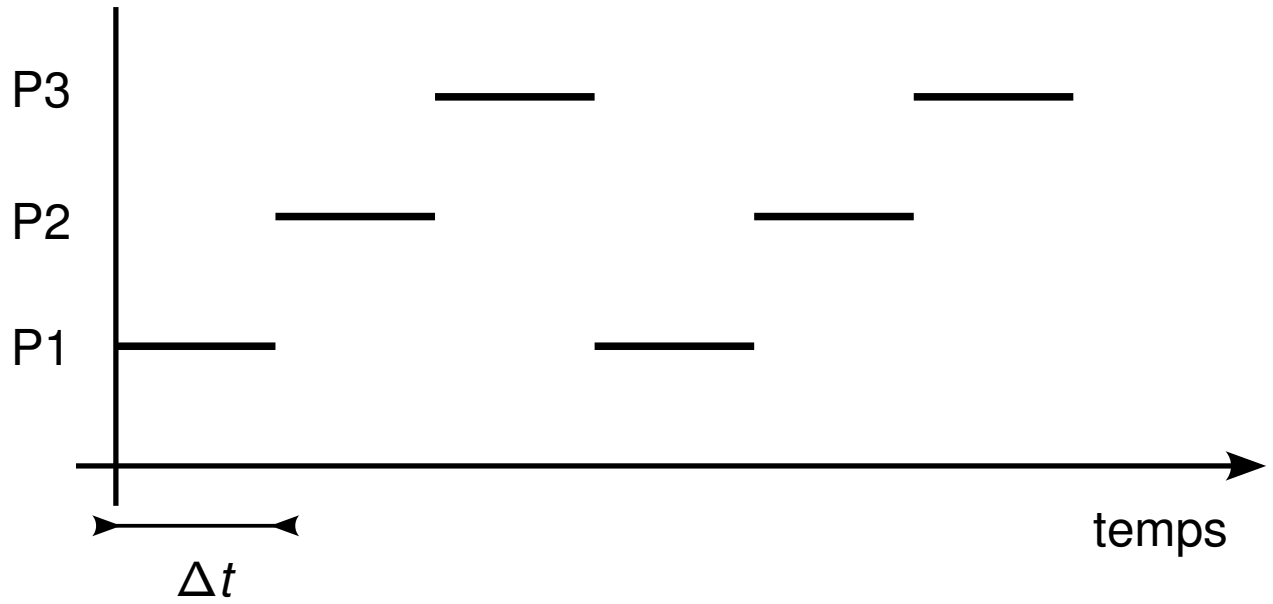
Vie d'un processus : préemption et partage du temps





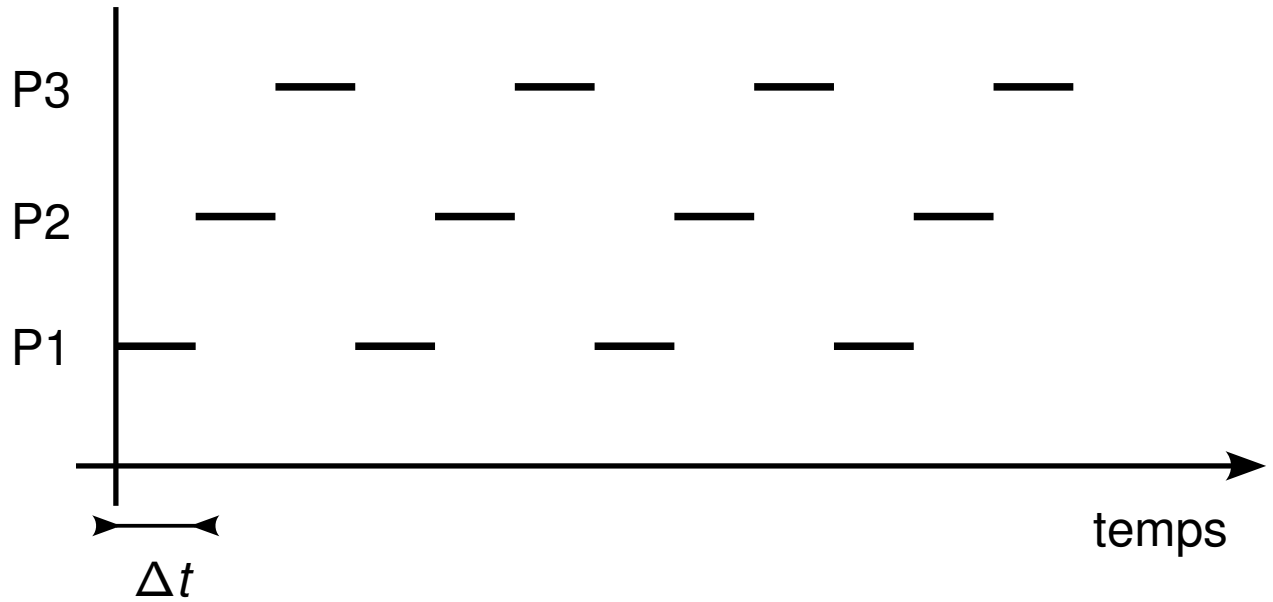
Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction ≫ quantum ≫ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



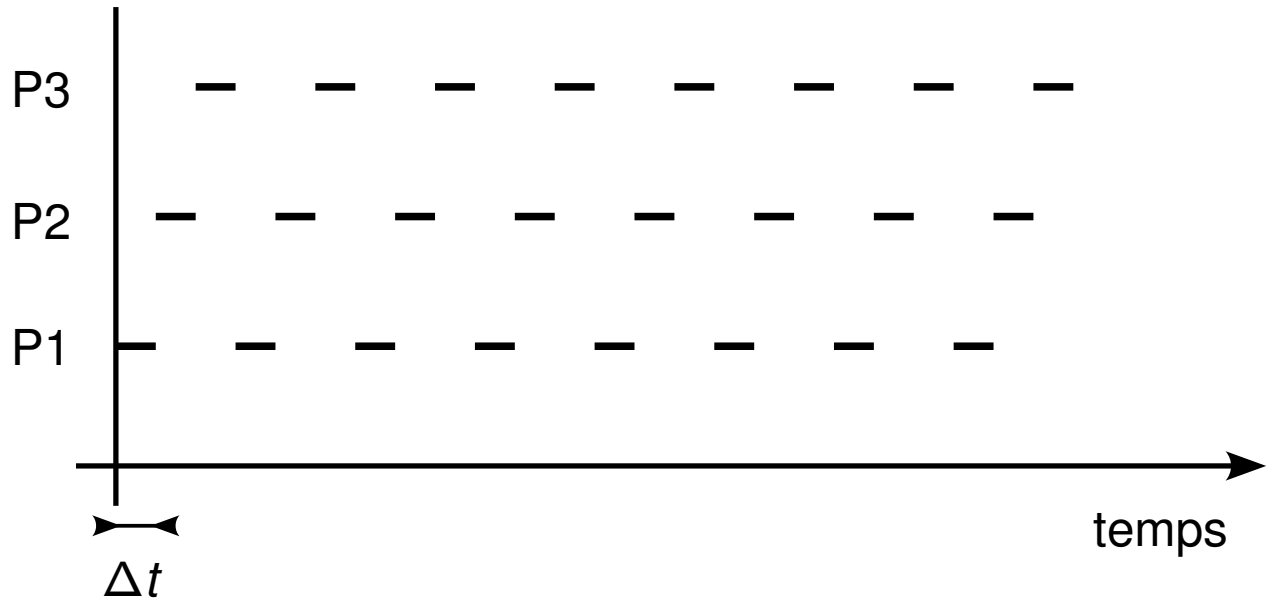
Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction ≫ quantum ≫ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



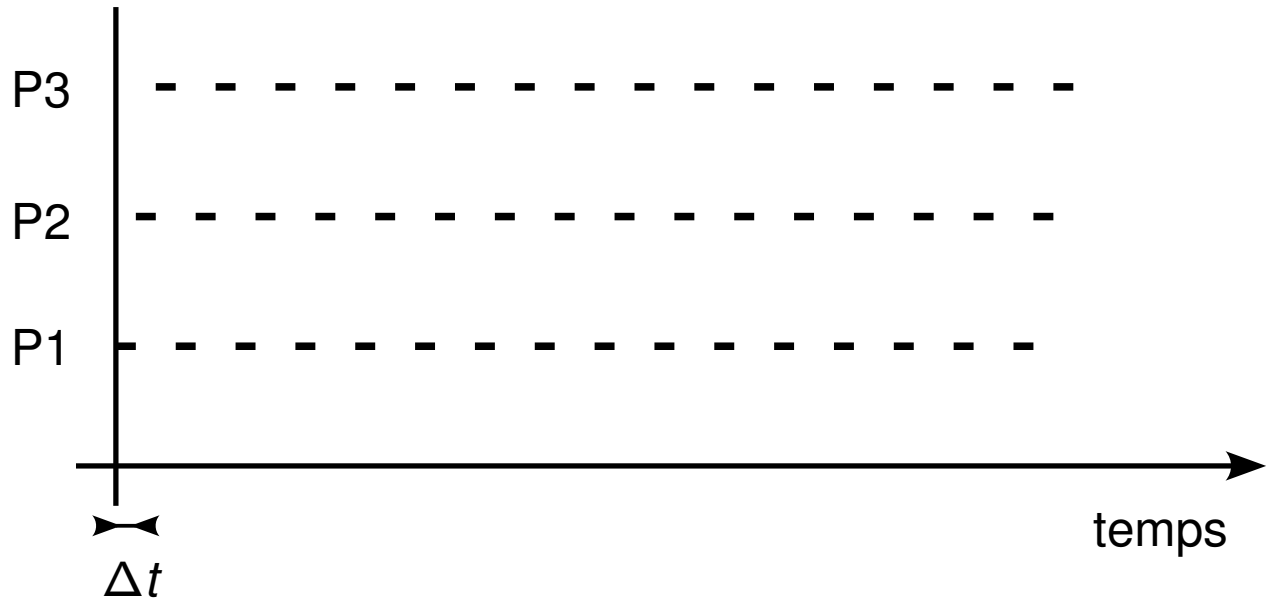
Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction $\gg$ quantum $\gg$ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



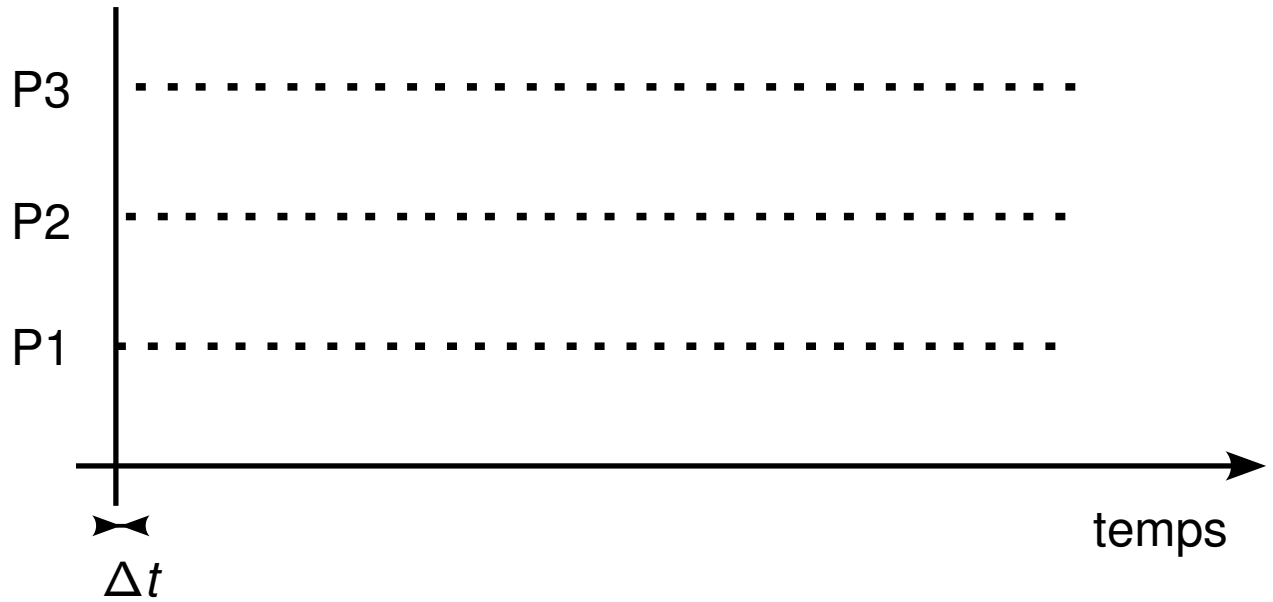
Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction $\gg$ quantum $\gg$ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



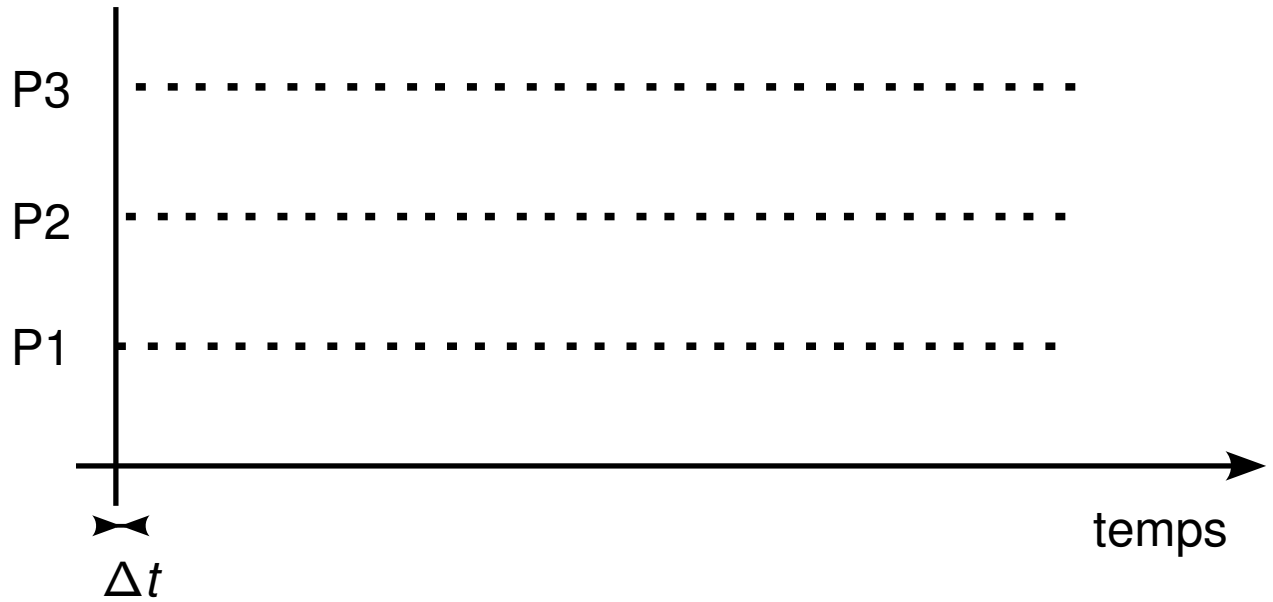
Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction ≫ quantum ≫ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



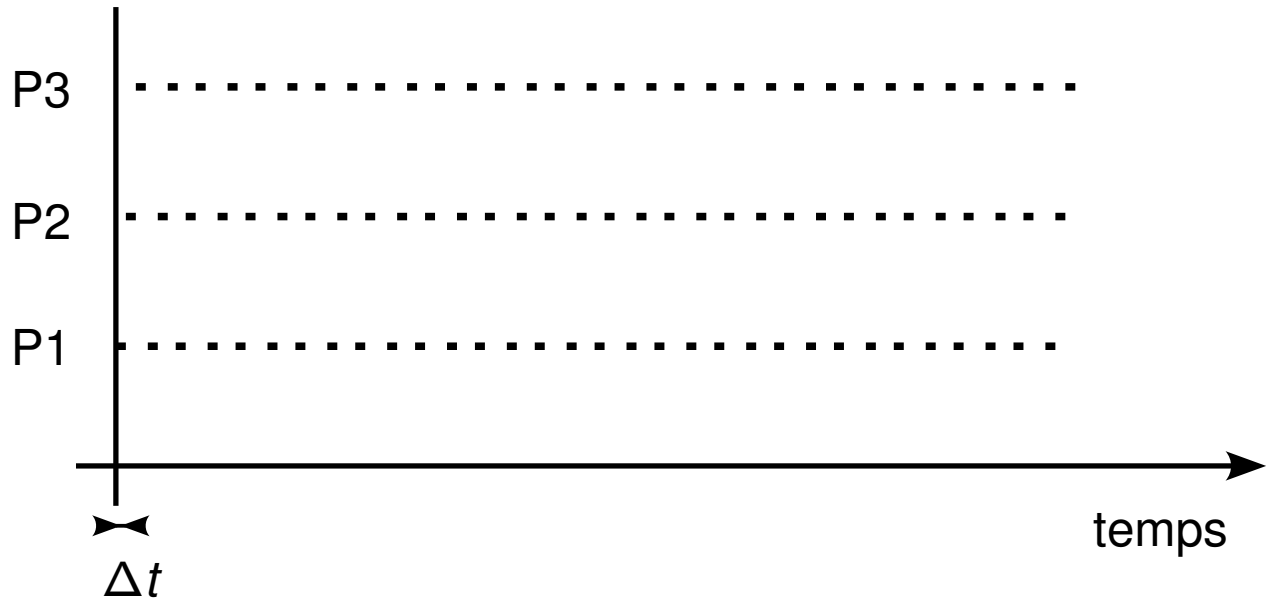
Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction $\gg$ quantum $\gg$ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



Préemption et partage du temps

- une horloge définit un quantum de temps + interruption
- quantum très petit => illusion tâches simultanées
- l'interactivité possible
- mais commuter des processus est coûteux :
  tps réaction $\gg$ quantum $\gg$ temps de commutation
  en pratique : Q = 10 à 100 ms , Comm < 0,01 ms



Etats d'un processus

- **nouveau** : créé mais pas encore éligible à l'exécution
- **prêt** (éligible) : peut être exécuté si CPU disponible
- **actif** (élu) : exécuté par le CPU **(unique)**
- **bloqué** : attend un événement extérieur
- **terminé** : a achevé son exécution

Transitions d'états

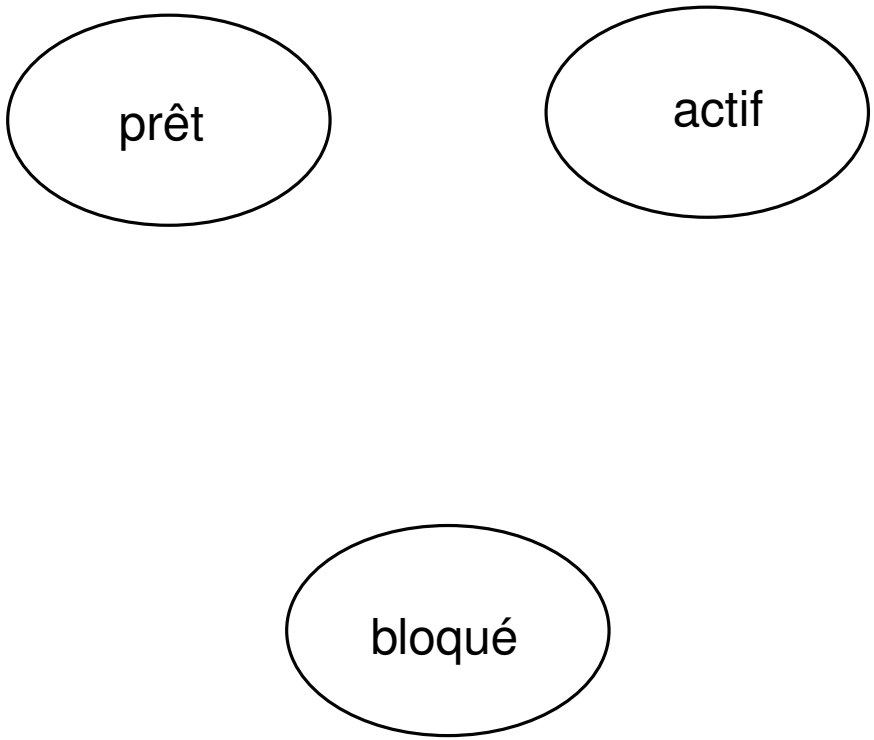


Diagramme d'états d'un processus

Transitions d'états

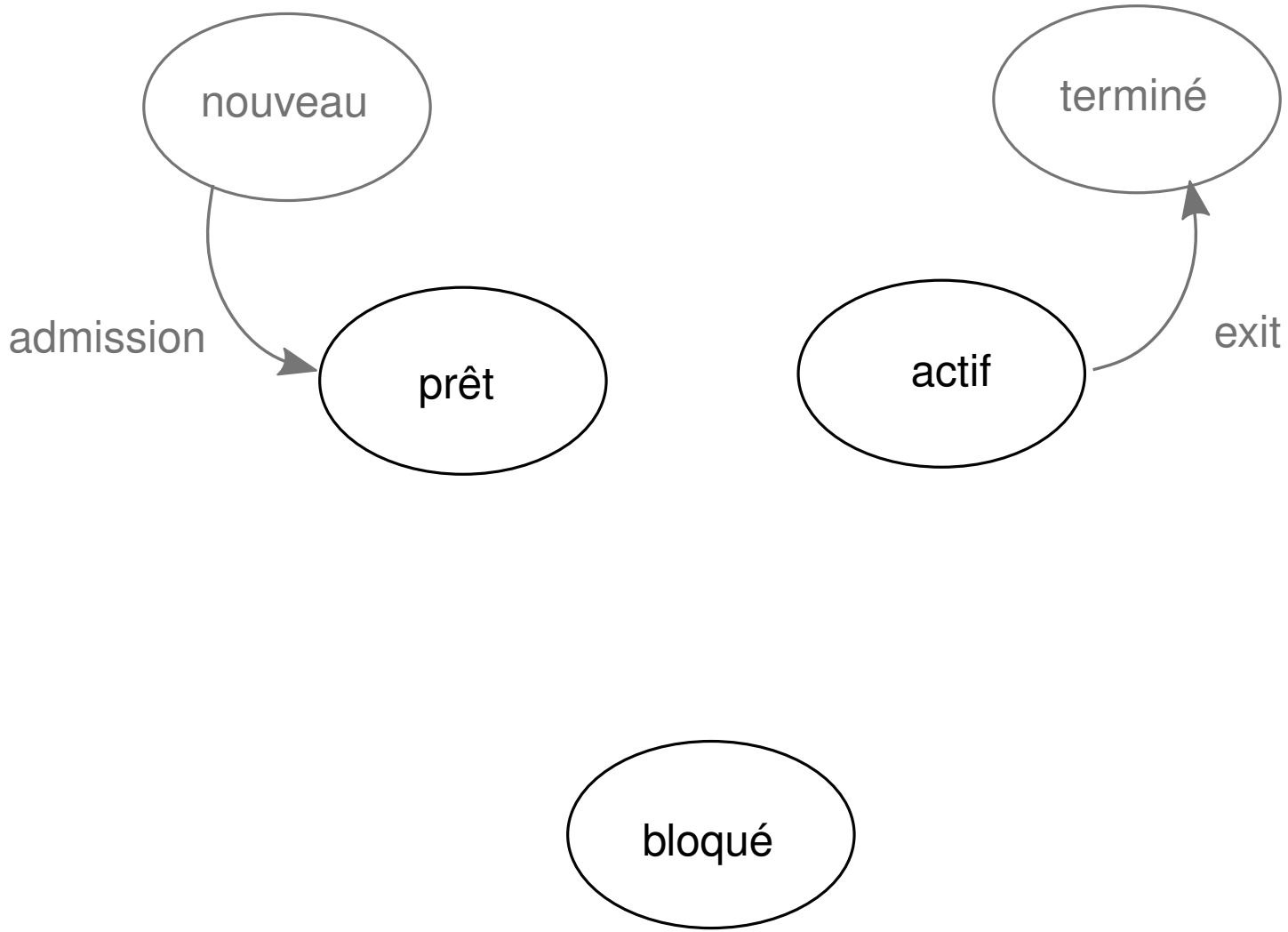


Diagramme d'états d'un processus

Transitions d'états

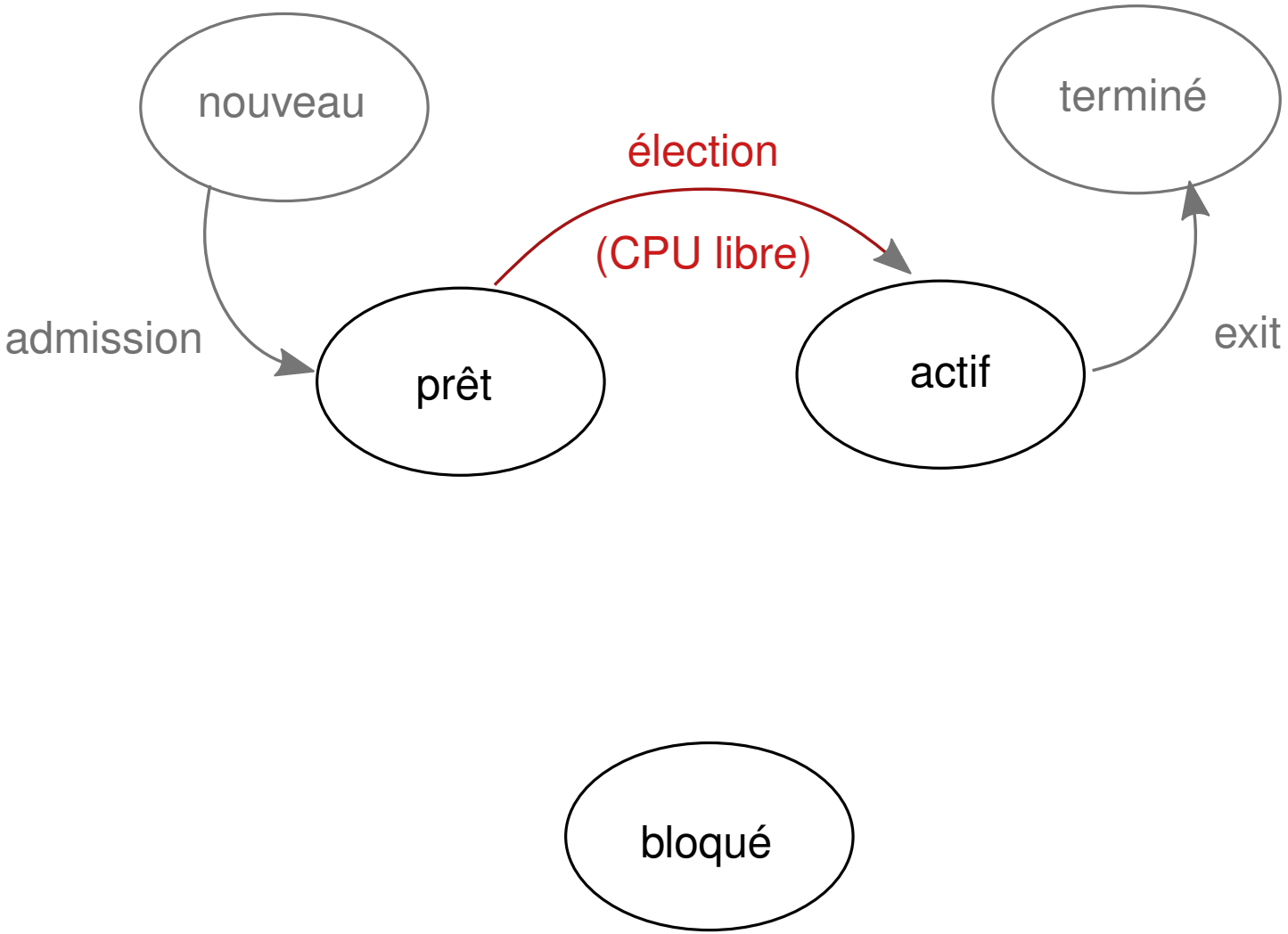


Diagramme d'états d'un processus

Transitions d'états

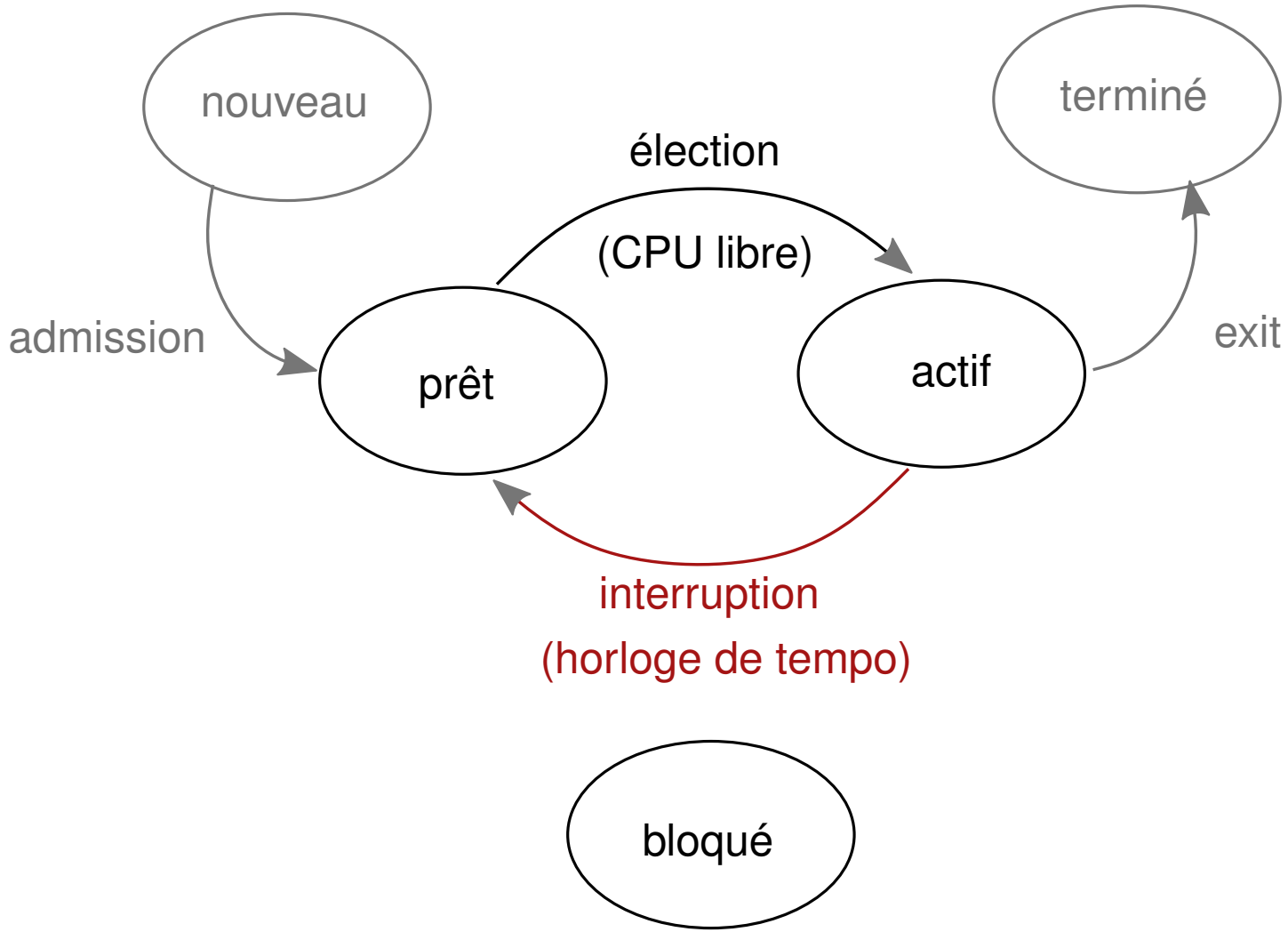


Diagramme d'états d'un processus

Transitions d'états

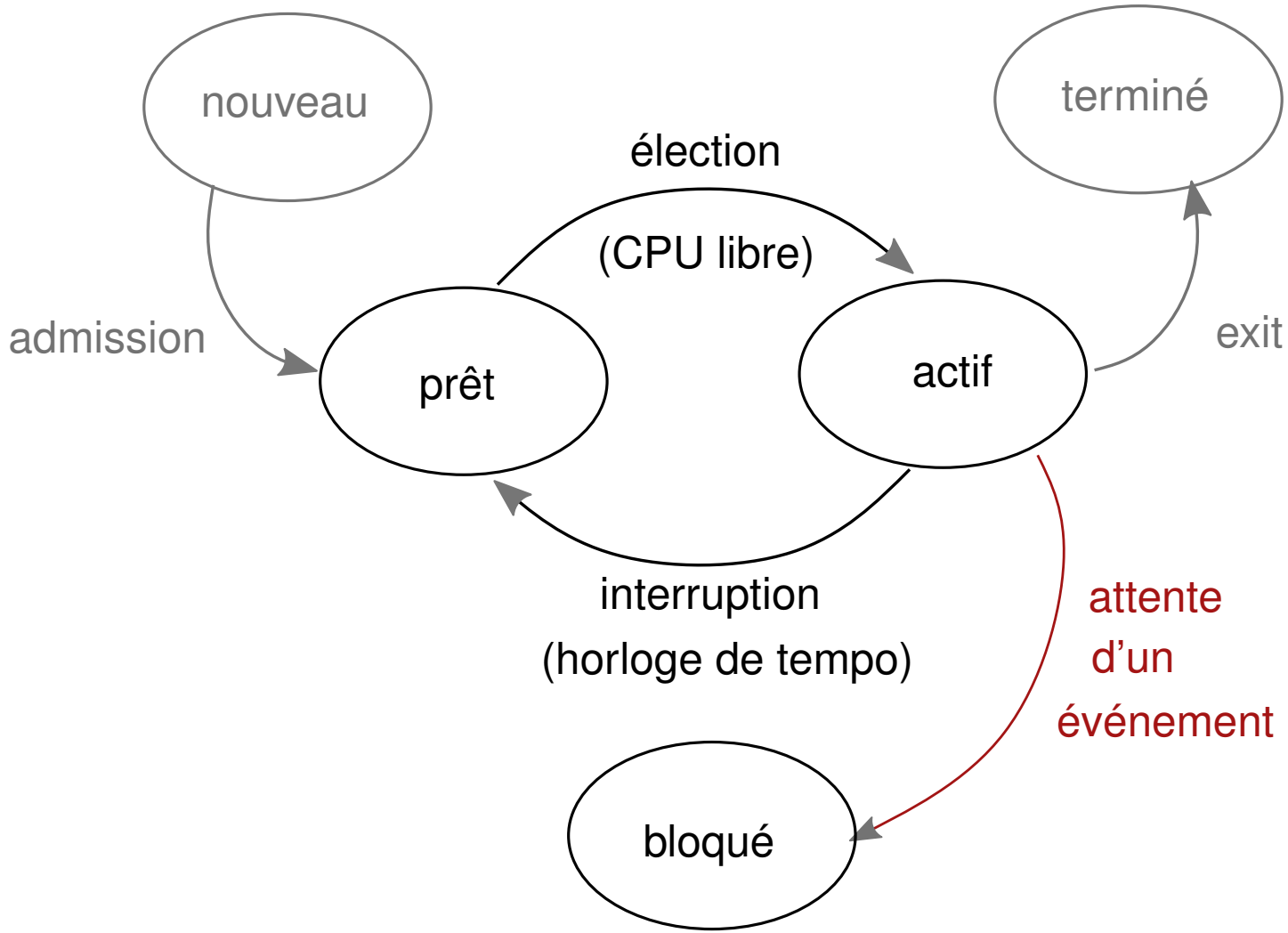


Diagramme d'états d'un processus

Transitions d'états

TD

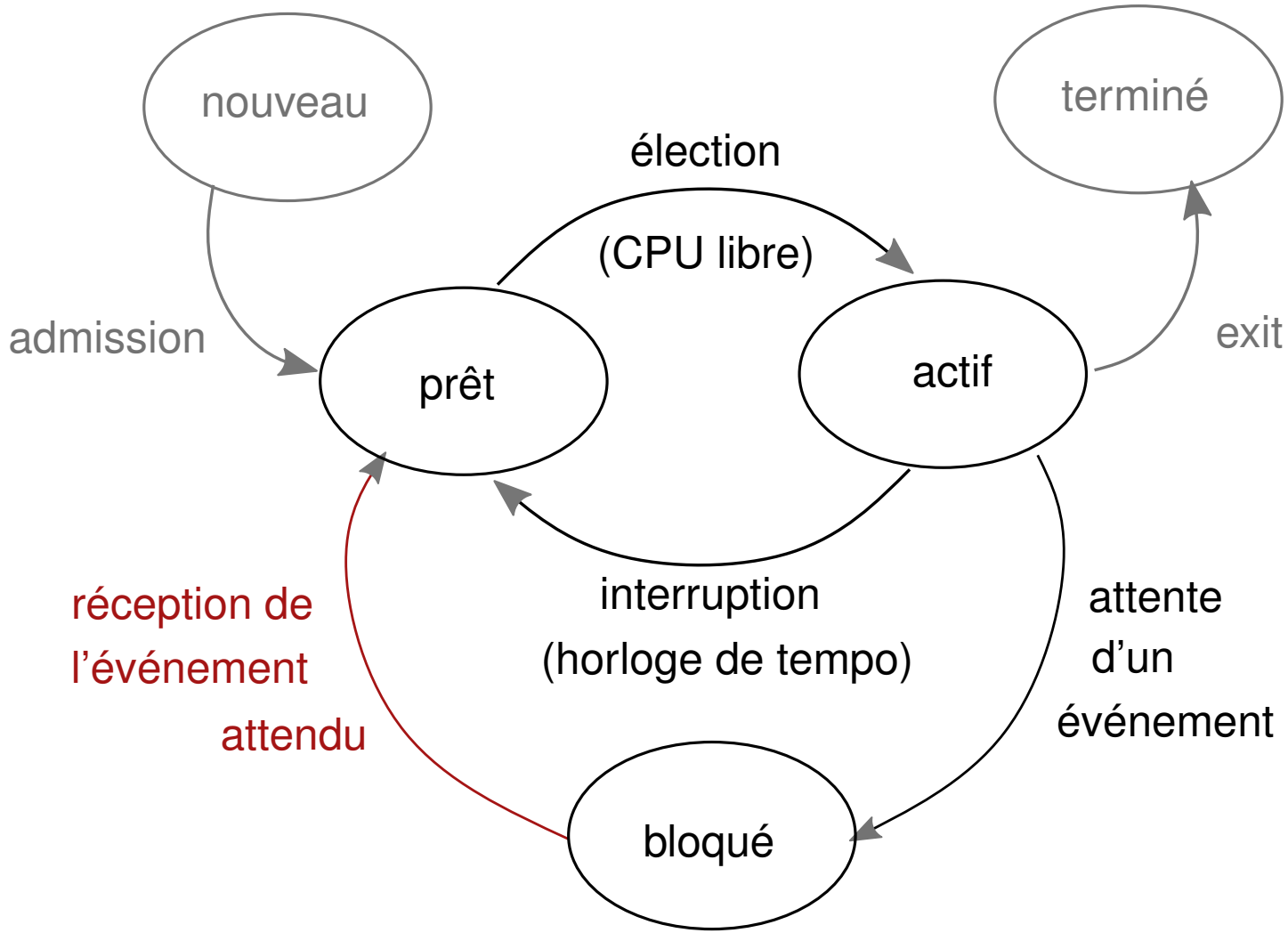


Diagramme d'états d'un processus

3. Partage des ressources

3.1. Partage du CPU

- Etats d'un processus
- Ordonnancement de processus

3.2. Partage de la mémoire centrale

- Pagination
- Mémoire virtuelle

Elire, parmi les prêts, le futur processus actif

Acteur

ordonnanceur : classe les processus prêts

Décider

- quel processus prêt choisir lorsque le CPU devient libre ?
- faut-il retirer le CPU au processus actif pour l'accorder à un prêt (si oui, lequel ?)
  - à l'arrivée d'un nouveau processus prêt ;
  - à des intervalles de temps réguliers.

Elire, parmi les prêts, le futur processus actif

Acteur

ordonnanceur : classe les processus prêts

Décider

- quel processus prêt choisir lorsque le CPU devient libre ?
- faut-il retirer le CPU au processus actif pour l'accorder à un prêt (si oui, lequel ?)
  - à l'arrivée d'un nouveau processus prêt ;
  - à des intervalles de temps réguliers.

Quel objectif ? Un compromis

Quelques critères pour l'ordonnancement à court terme

+ équité : chaque processus a un accès correct au CPU
+ rendement : occuper le **CPU** le + possible (40 à 90%)
+ débit : nombre de travaux terminés par unité de temps (1 à 10/s)
- temps de réalisation d'un **job** : temps entre sa soumission et sa terminaison
- temps d'attente : uniquement temps passé dans l'état prêt
- temps de **réponse** : temps entre requête et réponse (interactif)
- prise en compte de la nature différente des processus (noyau/user)

Quel objectif ? Un compromis

Quelques critères pour l'ordonnancement à court terme

+ équité : chaque processus a un accès correct au CPU
+ rendement : occuper le **CPU** le + possible (40 à 90%)
+ débit : nombre de travaux terminés par unité de temps (1 à 10/s)
- temps de réalisation d'un **job** : temps entre sa soumission et sa terminaison
- temps d'attente : uniquement temps passé dans l'état prêt
- temps de **réponse** : temps entre requête et réponse (interactif)
- prise en compte de la nature différente des processus (noyau/user)

Quel objectif ? Un compromis

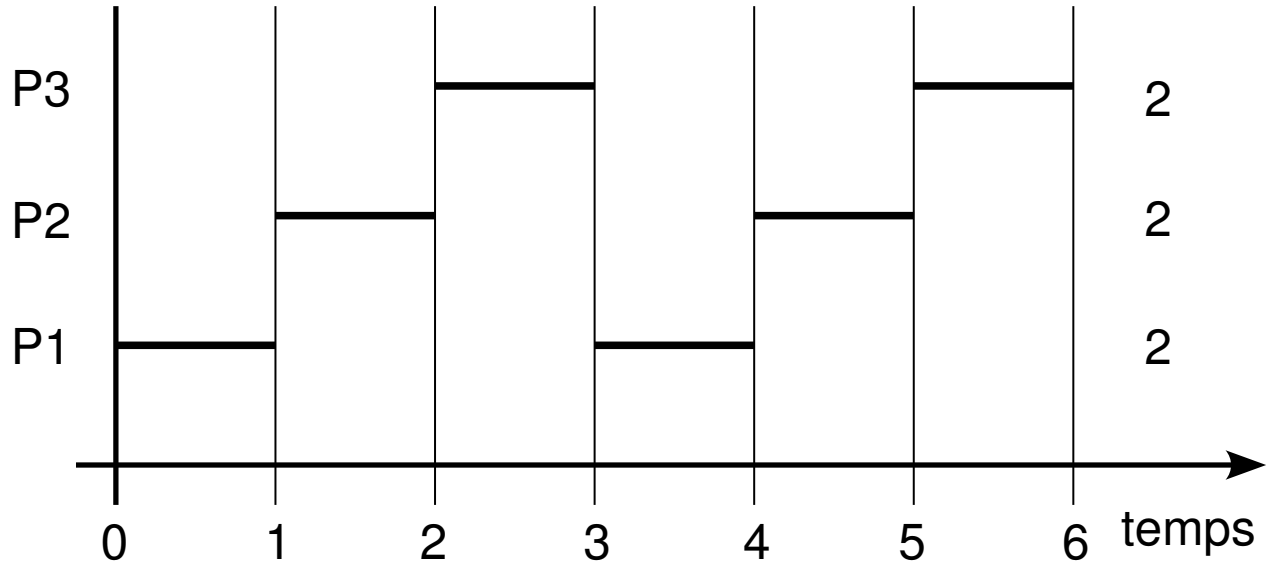
Quelques critères pour l'ordonnancement à court terme

+ équité : chaque processus a un accès correct au CPU
+ rendement : occuper le **CPU** le + possible (40 à 90%)
+ débit : nombre de travaux terminés par unité de temps (1 à 10/s)
- temps de réalisation d'un **job** : temps entre sa soumission et sa terminaison
- temps d'attente : uniquement temps passé dans l'état prêt
- temps de **réponse** : temps entre requête et réponse (interactif)
- prise en compte de la nature différente des processus (noyau/user)

Tourniquet (RR pour Round-Robin)

Principe

- quantum accordé successivement à chaque processus prêt
- quantum considéré infinitésimal



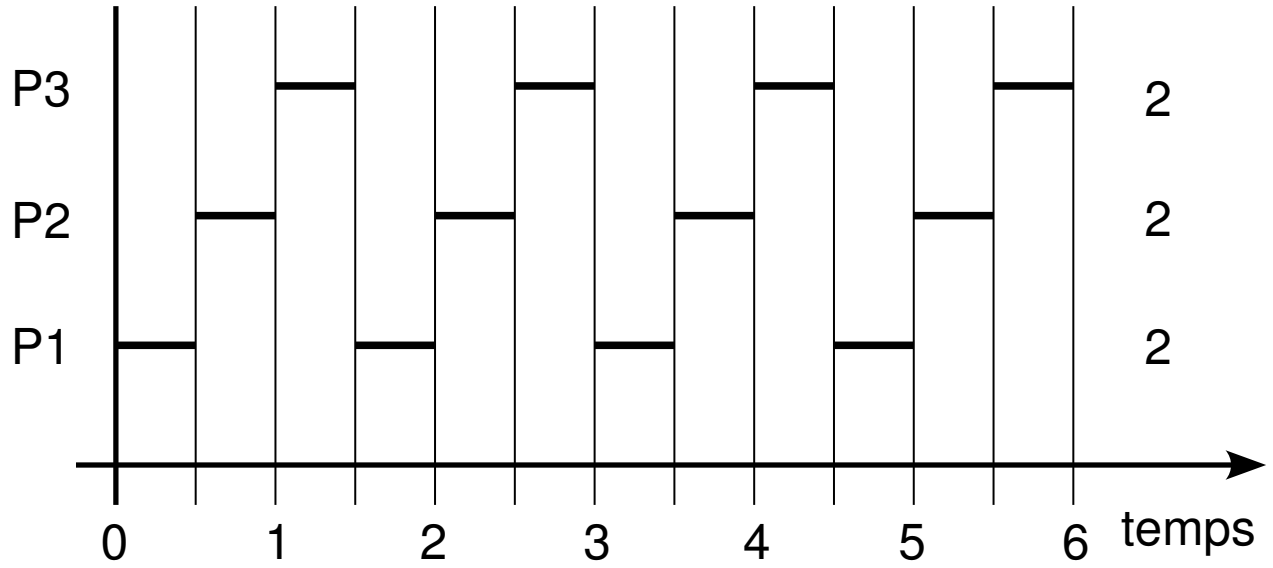
Propriétés

- stratégie optimale pour l'interactivité
- mais ne tient pas compte des natures différentes des processus

Tourniquet (RR pour Round-Robin)

Principe

- quantum accordé successivement à chaque processus prêt
- quantum considéré infinitésimal



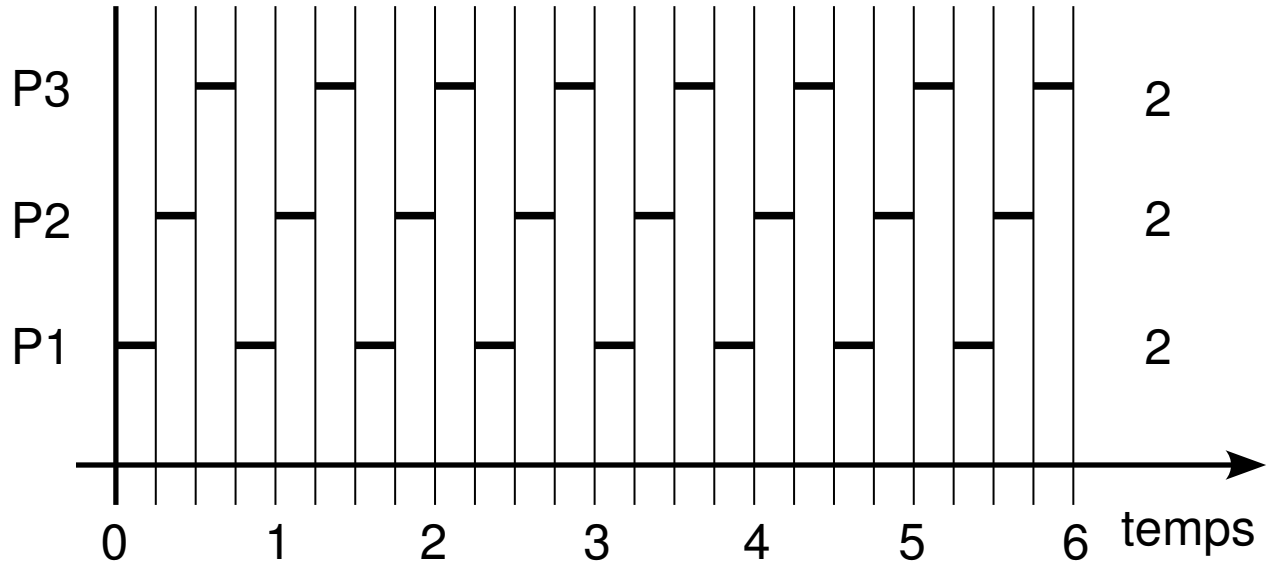
Propriétés

- stratégie optimale pour l'interactivité
- mais ne tient pas compte des natures différentes des processus

Tourniquet (RR pour Round-Robin)

Principe

- quantum accordé successivement à chaque processus prêt
- quantum considéré infinitésimal



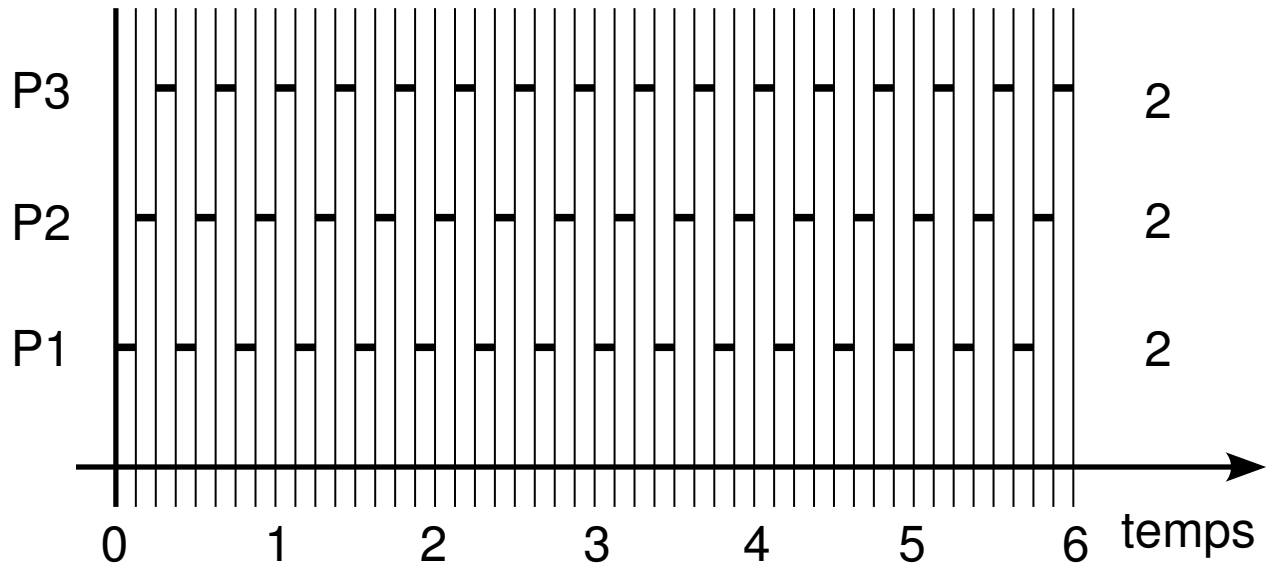
Propriétés

- stratégie optimale pour l'interactivité
- mais ne tient pas compte des natures différentes des processus

Tourniquet (RR pour Round-Robin)

Principe

- quantum accordé successivement à chaque processus prêt
- quantum considéré infinitésimal



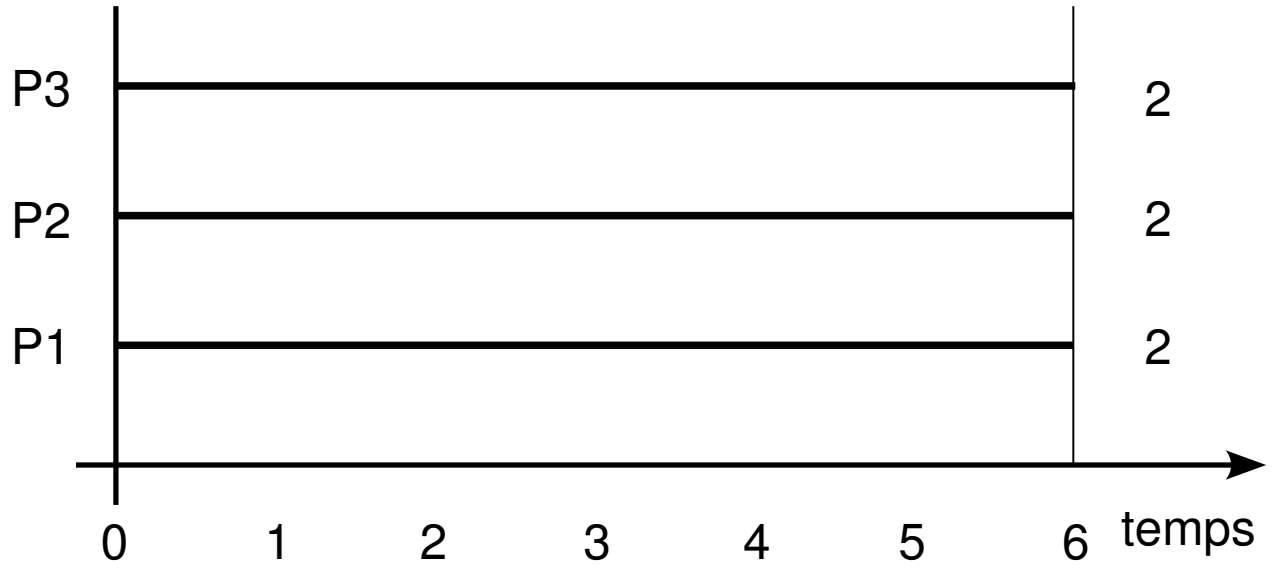
Propriétés

- stratégie optimale pour l'interactivité
- mais ne tient pas compte des natures différentes des processus

Tourniquet (RR pour Round-Robin)

Principe

- quantum accordé successivement à chaque processus prêt
- quantum considéré infinitésimal



Propriétés

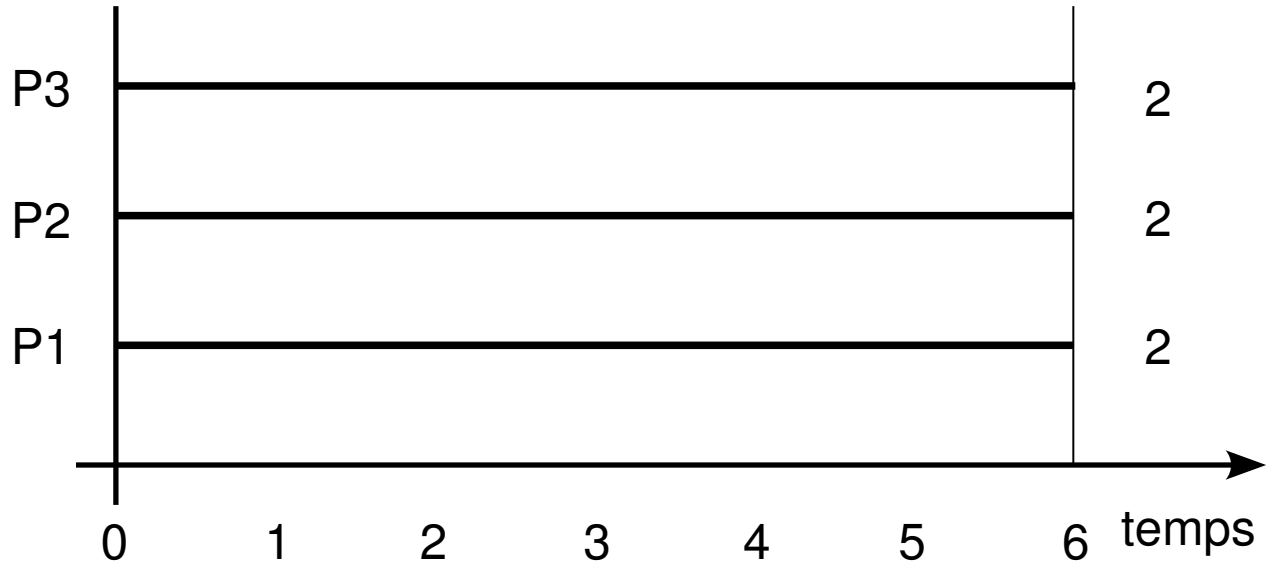
- stratégie optimale pour l'interactivité
- mais ne tient pas compte des natures différentes des processus

Tourniquet (RR pour Round-Robin)

TD

Principe

- quantum accordé successivement à chaque processus prêt
- quantum considéré infinitésimal



Propriétés

- stratégie optimale pour l'interactivité
- mais ne tient pas compte des natures différentes des processus

Priorités (Prio)

Principe : le processus prioritaire doit s'exécuter

- une priorité est affectée à chaque processus
  - de façon **externe** (utilisateur)
  - de façon **interne** (SE) : en fonction du passé du processus
- processus classé parmi les prêts selon sa priorité
- nouveau prêt prioritaire sur l'actif remplace celui-ci

Processus prioritaires

- disponibilité du système : processus noyau
- QoS : interactif (lecture audio) vs. batch (calcul)

Priorités (Prio)

Principe : le processus prioritaire doit s'exécuter

- une priorité est affectée à chaque processus
  - de façon **externe** (utilisateur)
  - de façon **interne** (SE) : en fonction du passé du processus
- processus classé parmi les prêts selon sa priorité
- nouveau prêt prioritaire sur l'actif remplace celui-ci

Processus prioritaires

- disponibilité du système : processus noyau
- QoS : interactif (lecture audio) vs. batch (calcul)

Priorités (Prio) - déclinaisons

Inconvénients

- risque de famine pour les processus de priorité faible
  $\Rightarrow$ *aging* : augmenter au cours du temps la priorité des processus prêts
- perte de l'illusion de multiples processus simultanés

Prio + RR

- (POSIX `SCHED_RR`)
  RR entre processus de plus haute priorité
- (Linux ou POSIX `SCHED_OTHER`)
  RR avec quantum dépendant de la priorité

Priorités (Prio)

Linux

- priorité : valeur par défaut (20) + valeur de courtoisie (-20 à +19)
- courtoisie forte : ≥ 0 , courtoisie faible : < 0
- utilisateur non root ne peut qu'augmenter la courtoisie de ses proc.
- **`nice -n 19 comm`** : exécute `comm` avec courtoisie = + 19
- **`renice 19 pid`** : affecte courtoisie = + 19 au processus pid

3. Partage des ressources

3.1. Partage du CPU

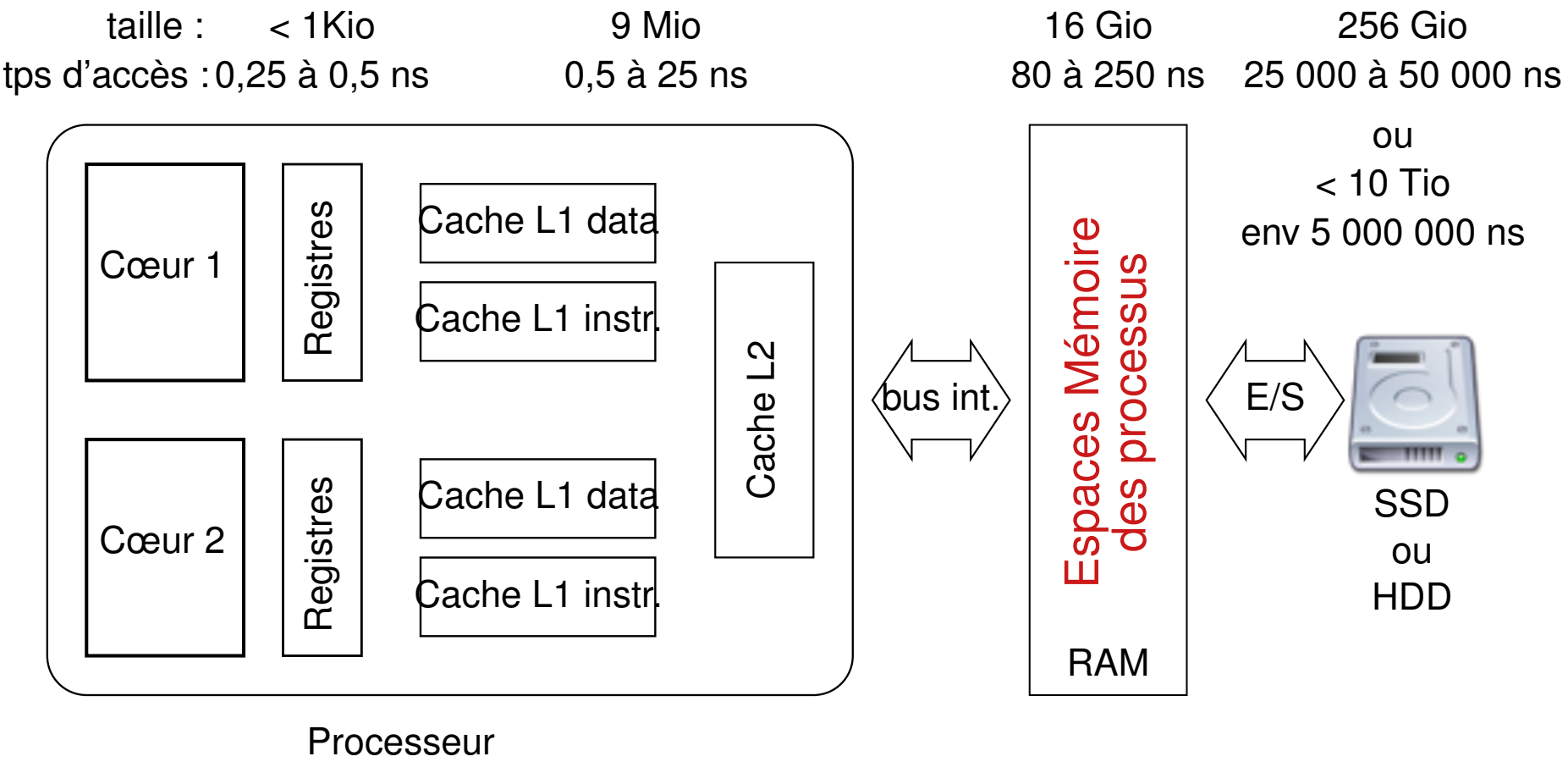
- Etats d'un processus
- Ordonnancement de processus

3.2. Partage de la mémoire centrale

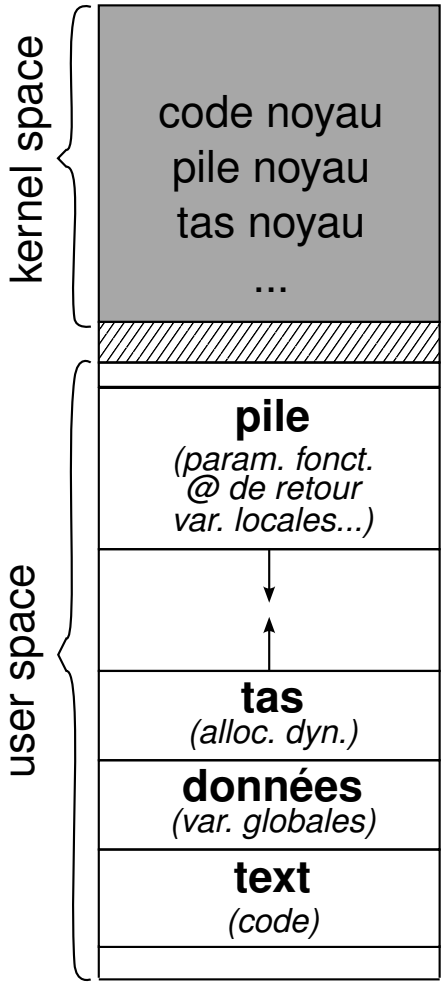
- Pagination
- Mémoire virtuelle

Composants - Mémoires [R2.04a]

Exemple de hiérarchie



Espace mémoire d'un processus [R3.16]



Espace mémoire d'un processus
(adresses logiques)

RAM partagée entre processus

Deux objectifs

- chaque processus définit **ses adresses de façon indépendante** *(chacun peut utiliser la même adresse mémoire pour pointer une donnée différente)*
- taille des espaces mémoires (4 Gio ou 4 Exio par proc.)
  $\gg$
  taille de l'espace physique (16 Gio)

Deux solutions complémentaires

- pagination
- mémoire virtuelle

RAM partagée entre processus

Deux objectifs

- chaque processus définit **ses adresses de façon indépendante** *(chacun peut utiliser la même adresse mémoire pour pointer une donnée différente)*
- taille des espaces mémoires (4 Gio ou 4 Exio par proc.)
  $\gg$
  taille de l'espace physique (16 Gio)

Deux solutions complémentaires

- pagination
- mémoire virtuelle

Pagination : chaque processus définit ses adresses de façon indépendante

Des adresses logiques et des adresses physiques

- **adresse logique = adresse vue par le processus**
- **adresse physique = adresse matérielle en RAM**
- traduction réalisée par MMU *(Memory Management Unit)*

Double question

- comment partager la mémoire de la RAM entre les processus ?
- comment traduire les adresses logiques en adresses physiques (MMU) ?

Pagination

Principe

- espace physique découpé en cases de taille fixe (Linux : 4 Kio)
- espace logique découpé en pages de même taille (4 Kio)

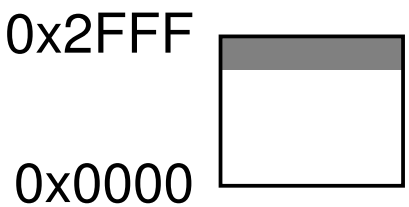
Adressage

table des pages : correspondance entre num_page et num_case

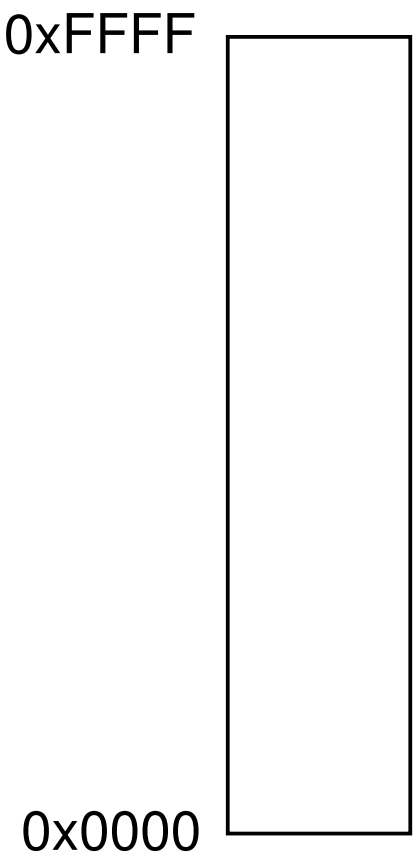
Discussion

- Possibilité d'avoir une zone physique mappée dans plusieurs espaces logiques (kernel space)
- Doublement des accès à la RAM
- Pourquoi des pages et pas un simple décalage ? (fragmentation)

Pagination : Principe

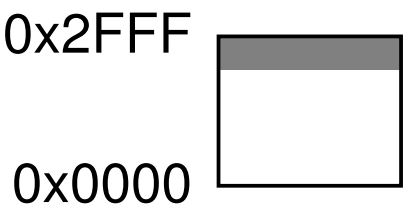


espace logique
d'un processus
de 12 Kio

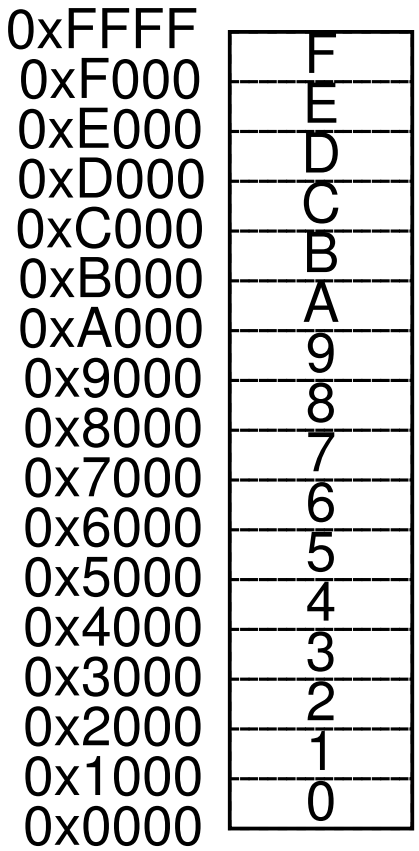


espace physique
d'une RAM partagée
de 64 Kio

Pagination : Principe



espace logique
d'un processus
de 12 Kio

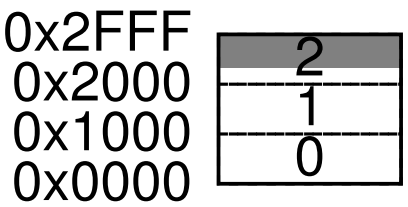


espace physique
d'une RAM partagée
de 64 Kio = 16 pages

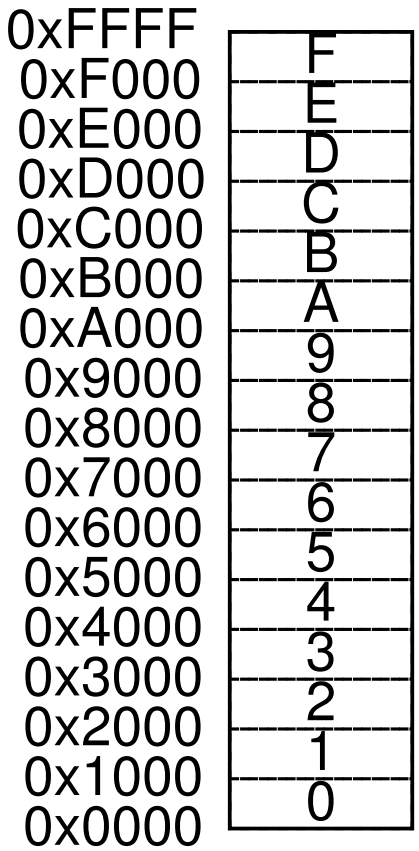
taille d'une case = 4 Kio = 0x1000 octets

Pagination : Principe

TD



espace logique
d'un processus
de 12 Kio = 3 pages



espace physique
d'une RAM partagée
de 64 Kio = 16 pages

taille d'une page = taille d'une case = 4 Kio = 0x1000 octets

Pagination : Adressage

adresse logique =	numéro de page		+	déplacement
0x2EE0 =	2	* 0x1000	+	0x0EE0

Pagination : Adressage

adresse logique =	numéro de page		+	déplacement
0x2EE0 =	2	* 0x1000	+	0x0EE0

adresse physique =	numéro de case		+	déplacement
? =	?		+	?

Pagination : Adressage

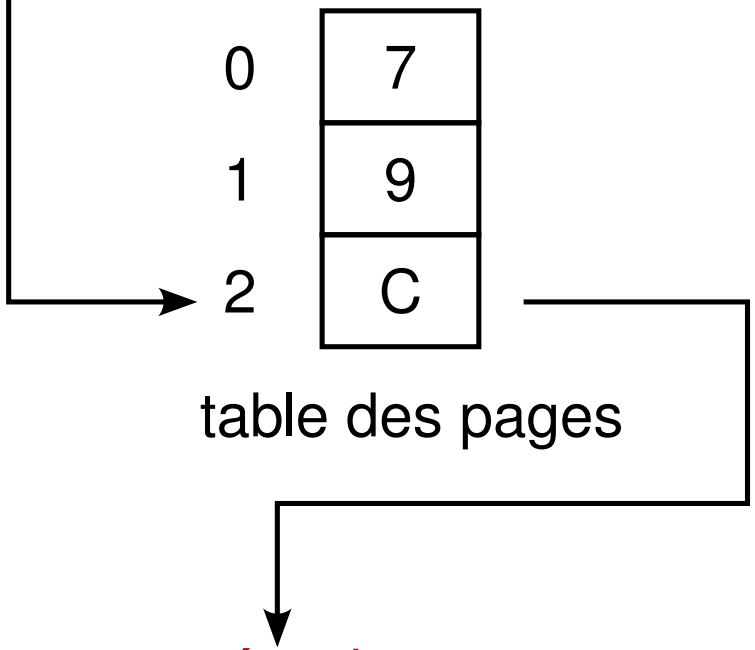
adresse logique =	numéro de page		+	déplacement
0x2EE0 =	2	* 0x1000	+	0x0EE0

adresse physique =	numéro de case	+	déplacement
? =	?	+	0x0EE0

Pagination : Adressage

adresse logique = numéro de page + déplacement

0x2EE0 = 2 * 0x1000 + 0x0EE0



adresse physique = numéro de case + déplacement

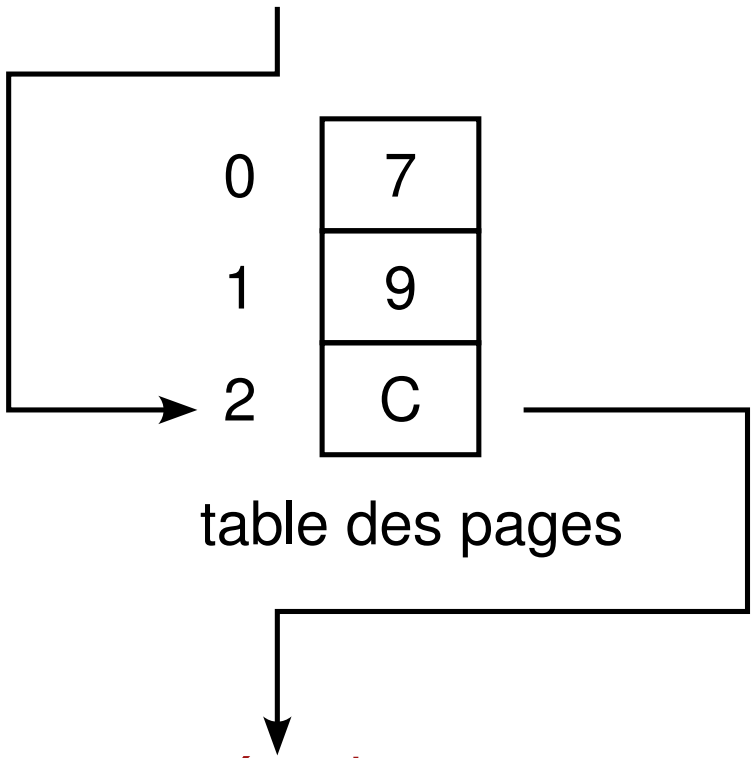
? = C * 0x1000 + 0x0EE0

Pagination : Adressage

TD

adresse logique = numéro de page + déplacement

0x2EE0 = 2 * 0x1000 + 0x0EE0



adresse physique = numéro de case + déplacement

0xCEE0 = C * 0x1000 + 0x0EE0

Pagination

Principe

- espace physique découpé en cases de taille fixe (Linux : 4 Kio)
- espace logique découpé en pages de même taille (4 Kio)

Adressage

table des pages : correspondance entre num_page et num_case

Discussion

- Possibilité d'avoir une zone physique mappée dans plusieurs espaces logiques (kernel space)
- Doublement des accès à la RAM
- Pourquoi des pages et pas un simple décalage ? (fragmentation)

Pagination

Principe

- espace physique découpé en cases de taille fixe (Linux : 4 Kio)
- espace logique découpé en pages de même taille (4 Kio)

Adressage

table des pages : correspondance entre num_page et num_case

Discussion

- Possibilité d'avoir une zone physique mappée dans plusieurs espaces logiques (kernel space)
- Doublement des accès à la RAM
- Pourquoi des pages et pas un simple décalage ? (fragmentation)

Pagination

Principe

- espace physique découpé en cases de taille fixe (Linux : 4 Kio)
- espace logique découpé en pages de même taille (4 Kio)

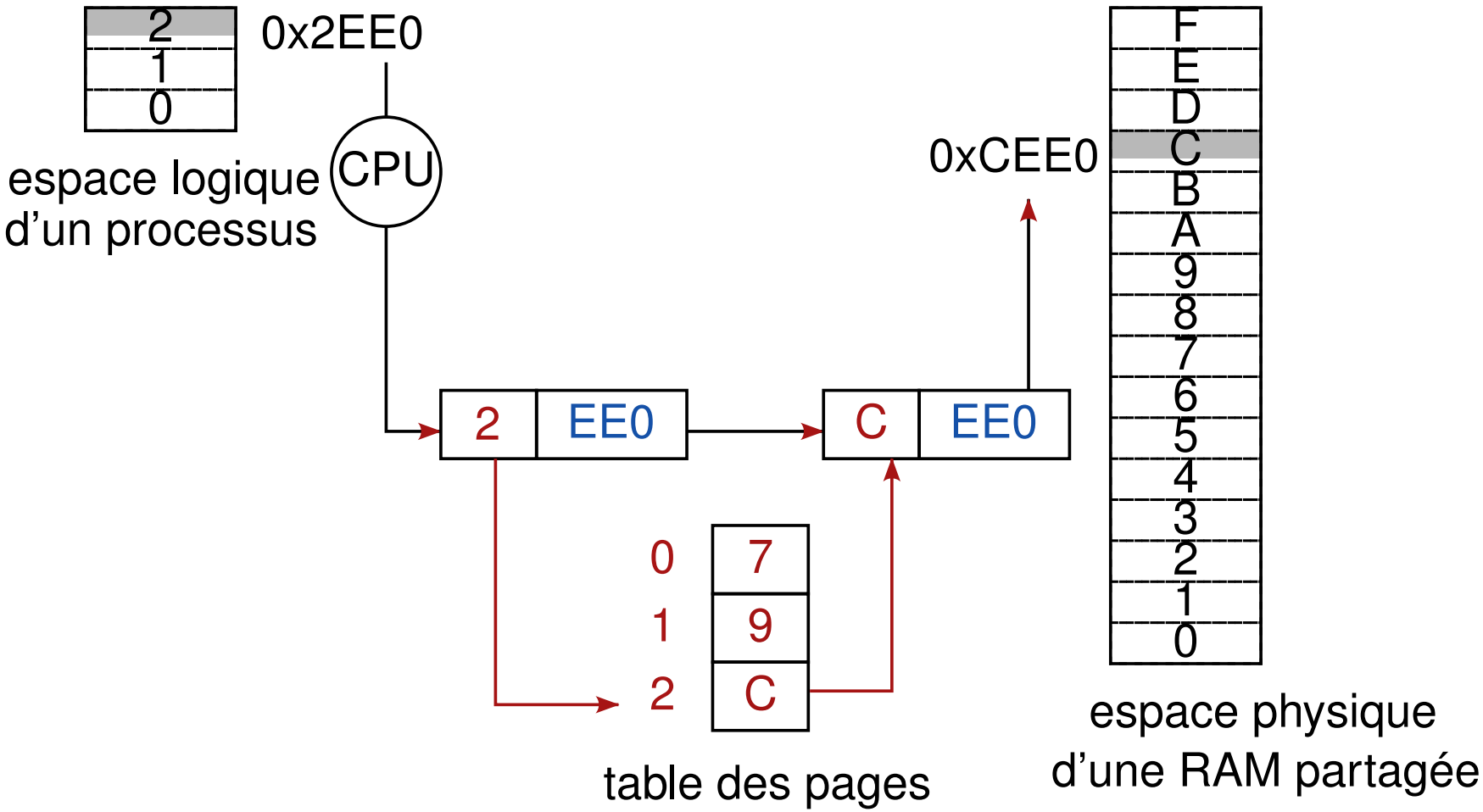
Adressage

table des pages : correspondance entre num_page et num_case

Discussion

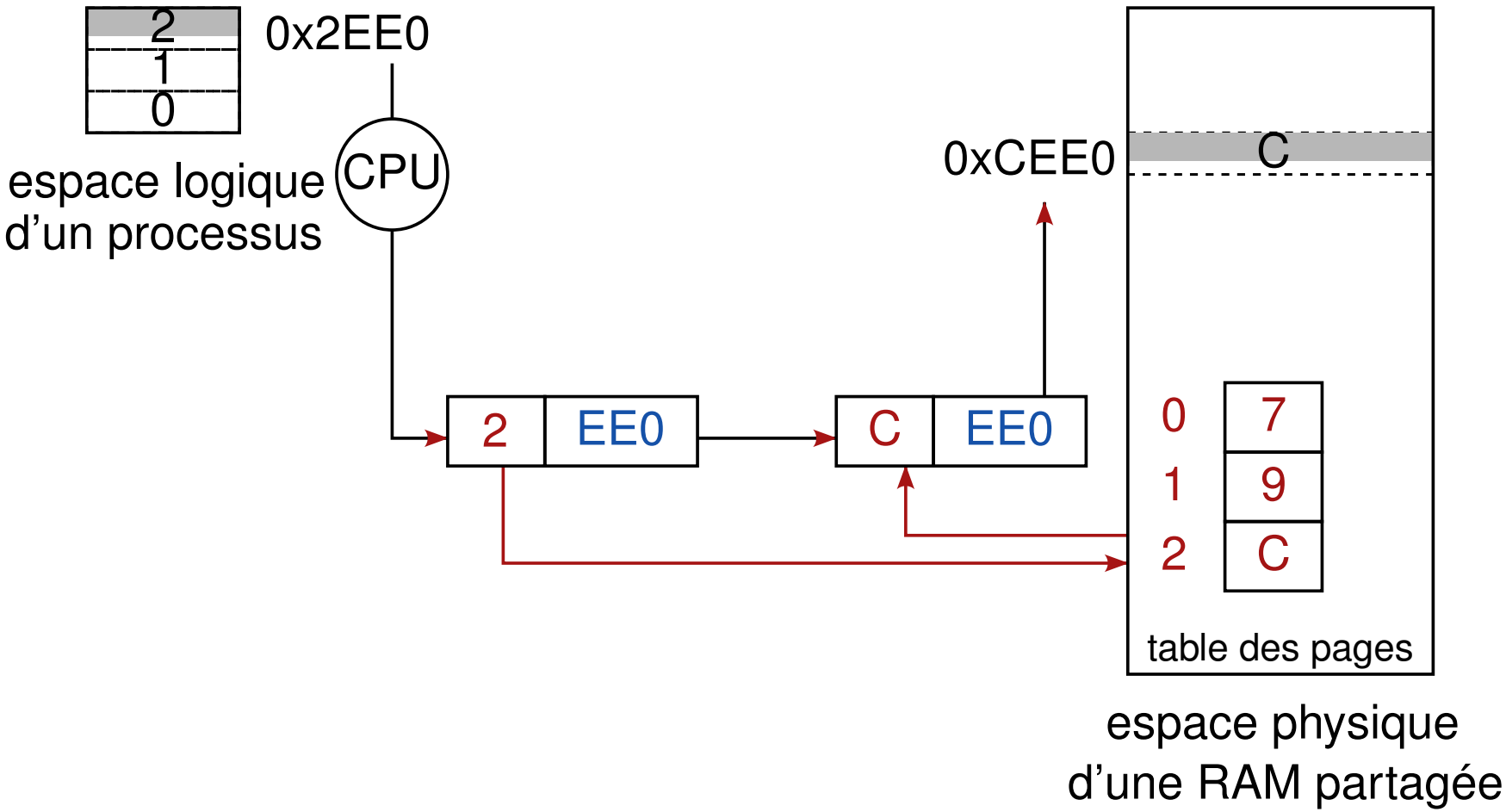
- Possibilité d'avoir une zone physique mappée dans plusieurs espaces logiques (kernel space)
- Doublement des accès à la RAM
- Pourquoi des pages et pas un simple décalage ? (fragmentation)

Pagination : Translation Look-aside Buffer



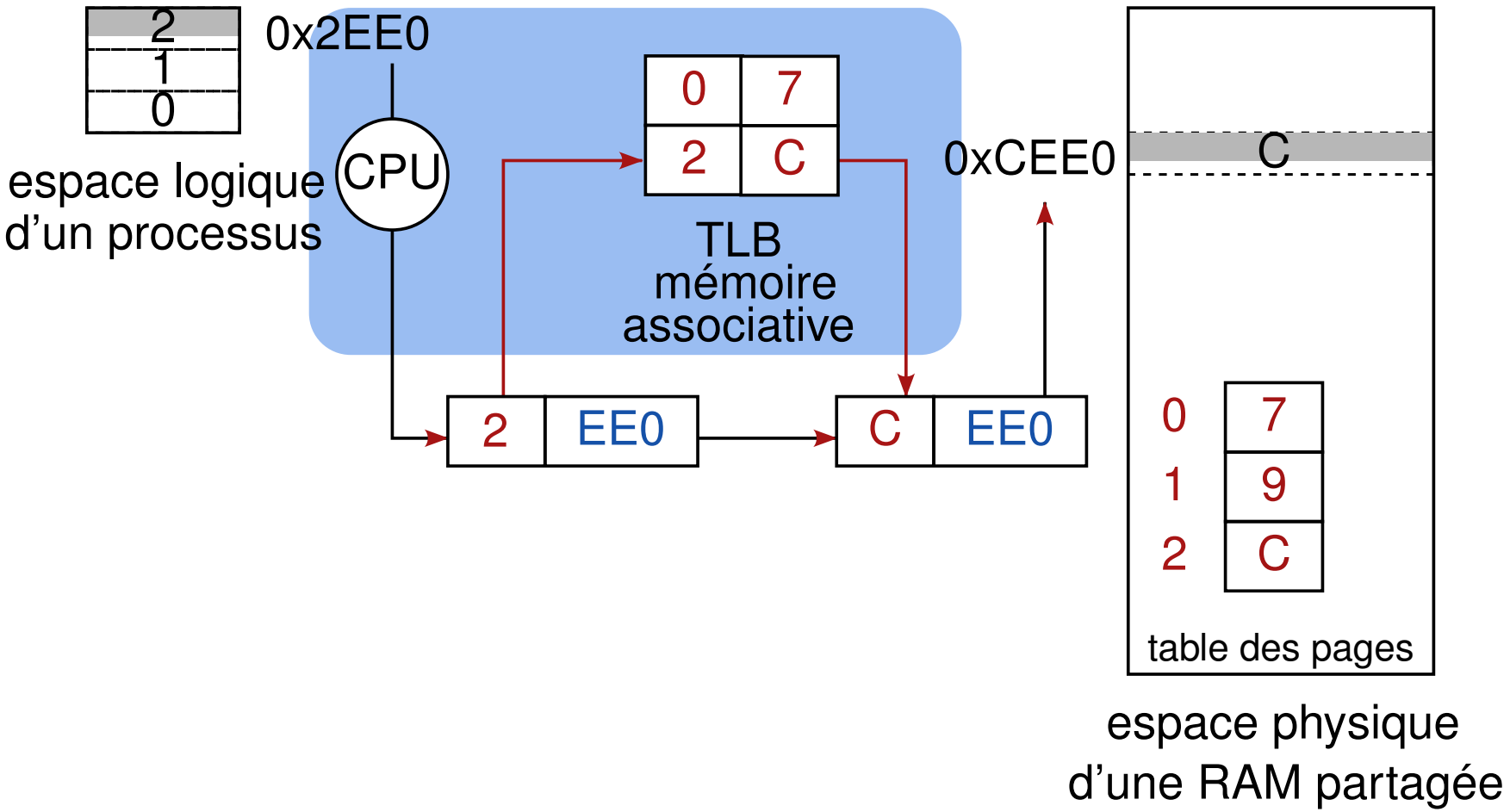
taille d'une page = taille d'une case = 4 Kio = 0x1000 octets

Pagination : Translation Look-aside Buffer



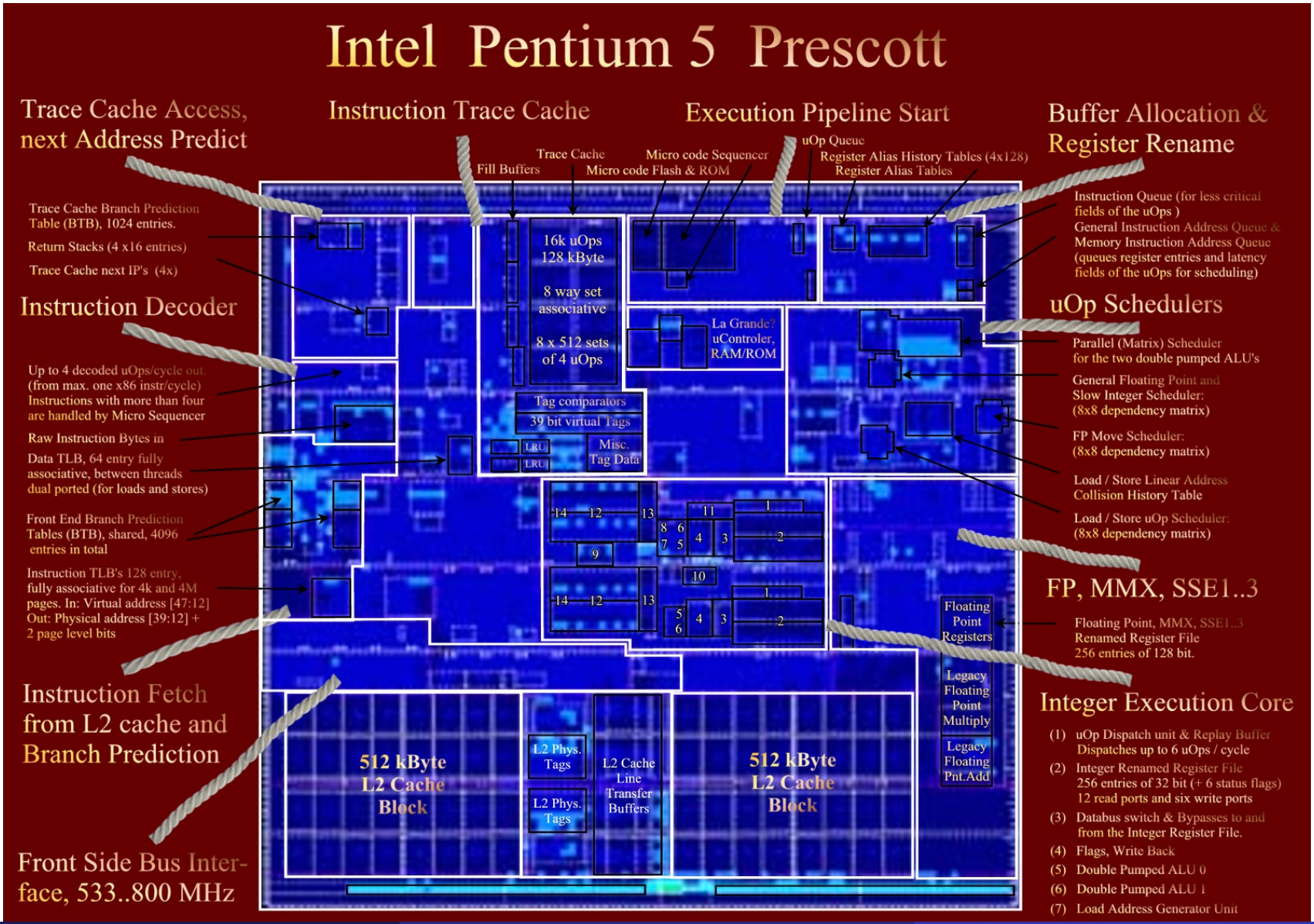
taille d'une page = taille d'une case = 4 Kio = 0x1000 octets

Pagination : Translation Look-aside Buffer



taille d'une page = taille d'une case = 4 Kio = 0x1000 octets

TLB : un composant physique



Pagination

Principe

- espace physique découpé en cases de taille fixe (Linux : 4 Kio)
- espace logique découpé en pages de même taille (4 Kio)

Adressage

table des pages : correspondance entre num_page et num_case

Discussion

- Possibilité d'avoir une zone physique mappée dans plusieurs espaces logiques (kernel space)
- Doublement des accès à la RAM limité par Translation Look-aside Buffer (TLB) : mémoire associative, cache de la table des pages *(localité temporelle)*
- Pourquoi des pages et pas un simple décalage ? (fragmentation)

Pagination

Principe

- espace physique découpé en cases de taille fixe (Linux : 4 Kio)
- espace logique découpé en pages de même taille (4 Kio)

Adressage

table des pages : correspondance entre num_page et num_case

Discussion

- Possibilité d'avoir une zone physique mappée dans plusieurs espaces logiques (kernel space)
- Doublement des accès à la RAM limité par Translation Look-aside Buffer (TLB) : mémoire associative, cache de la table des pages *(localité temporelle)*
- Pourquoi des pages et pas un simple décalage ? (fragmentation)

Fragmentation (externe)

Définition

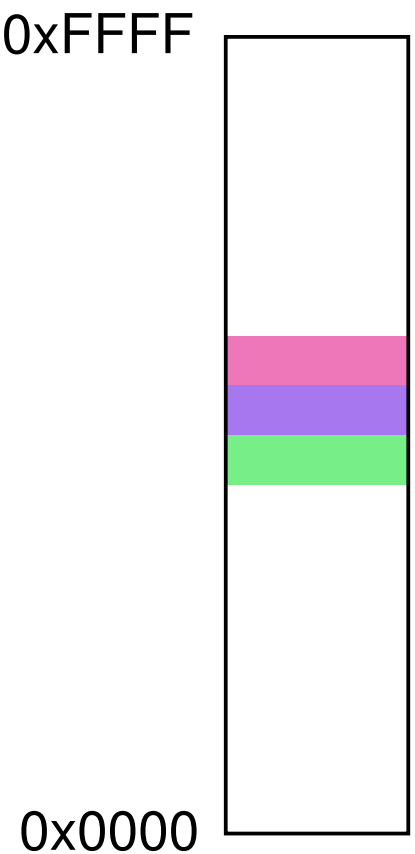
allocation refusée alors que l'union des zones libres serait suffisante

Défragmentation

- déplacer les zones mémoires pour réunir les zones libres
- "relogement dynamique" de processus en cours d'exécution pas toujours possible

Fragmentation (externe)

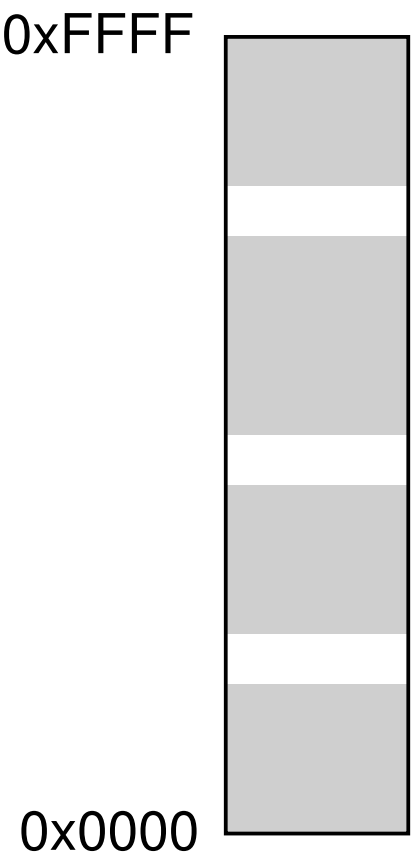
espace logique
d'un processus
de 12 Kio



espace physique
d'une RAM partagée
de 64 Kio

Fragmentation (externe)

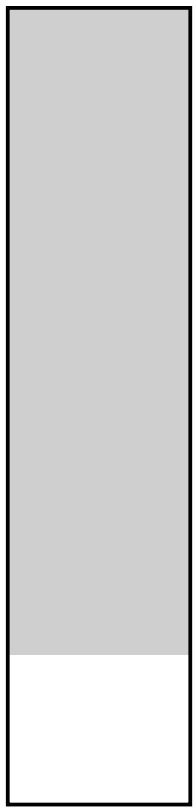
espace logique
d'un processus
de 12 Kio



espace physique
d'une RAM partagée
de 64 Kio

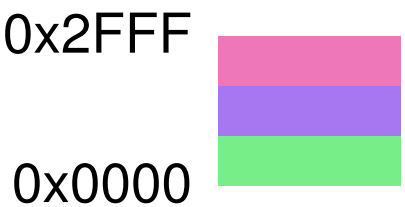
Fragmentation (externe)

espace logique
d'un processus
de 12 Kio

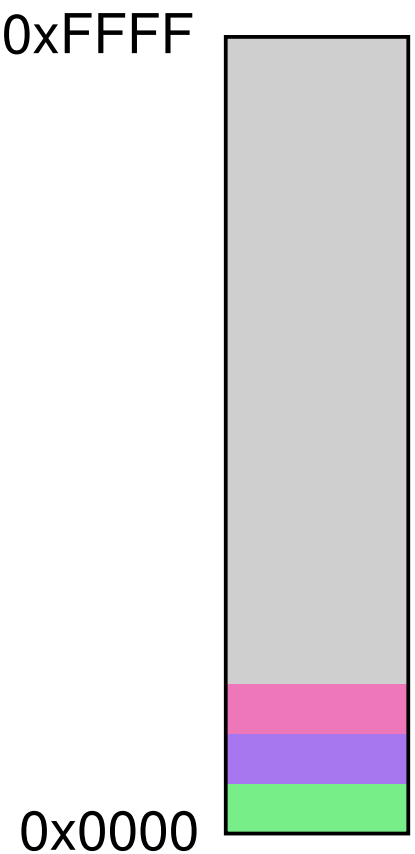


espace physique
d'une RAM partagée
de 64 Kio

Fragmentation (externe)



espace logique
d'un processus
de 12 Kio



espace physique
d'une RAM partagée
de 64 Kio

Fragmentation (externe)

Définition

allocation refusée alors que l'union des zones libres serait suffisante

Défragmentation

- déplacer les zones mémoires pour réunir les zones libres
- "relogement dynamique" de processus en cours d'exécution pas toujours possible

Pagination et fragmentations

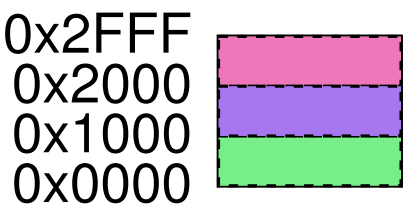
Risque de fragmentation externe

- lors d'allocation contiguë de zones de tailles variables
- résolue par pagination

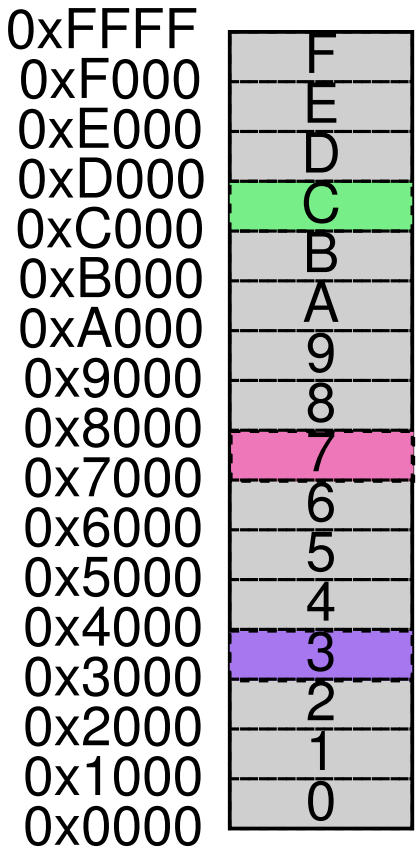
Fragmentation interne

- l'espace physique alloué à un processus est plus important que celui qu'il utilise effectivement
- présente avec la pagination
- $\Rightarrow$ bien choisir la taille des cases pour minimiser cette perte (mais dépend du hardware qui propose un choix limité)

Pagination : pas de fragmentation externe



espace logique
d'un processus
de 12 Kio = 3 pages



espace physique
d'une RAM partagée
de 64 Kio = 16 pages

taille d'une page = taille d'une case = 4 Kio = 0x1000 octets

Pagination et fragmentations

Risque de fragmentation externe

- lors d'allocation contiguë de zones de tailles variables
- résolue par pagination
- ce risque existe car toutes les pages d'un espace de processus ne sont pas nécessairement chargées en mémoire !

Fragmentation interne

- l'espace physique alloué à un processus est plus important que celui qu'il utilise effectivement
- présente avec la pagination
- $\Rightarrow$ bien choisir la taille des cases pour minimiser cette perte (mais dépend du hardware qui propose un choix limité)

Pagination et fragmentations

Risque de fragmentation externe

- lors d'allocation contiguë de zones de tailles variables
- résolue par pagination
- **ce risque existe car toutes les pages d'un espace de processus ne sont pas nécessairement chargées en mémoire !**

Fragmentation interne

- l'espace physique alloué à un processus est plus important que celui qu'il utilise effectivement
- présente avec la pagination
- $\Rightarrow$ bien choisir la taille des cases pour minimiser cette perte (mais dépend du hardware qui propose un choix limité)

RAM partagée entre processus

Deux objectifs

- chaque processus définit **ses adresses de façon indépendante** *(chacun peut utiliser la même adresse mémoire pour pointer une donnée différente)*
- taille des espaces mémoires (4 Gio ou 4 Exio par proc.)
  $\gg$
  taille de l'espace physique (16 Gio)

Deux solutions complémentaires

- pagination
- mémoire virtuelle