Mécanismes fondamentaux d'un système d'exploitation

Processus

Thread

Techniques d'ordonnancement

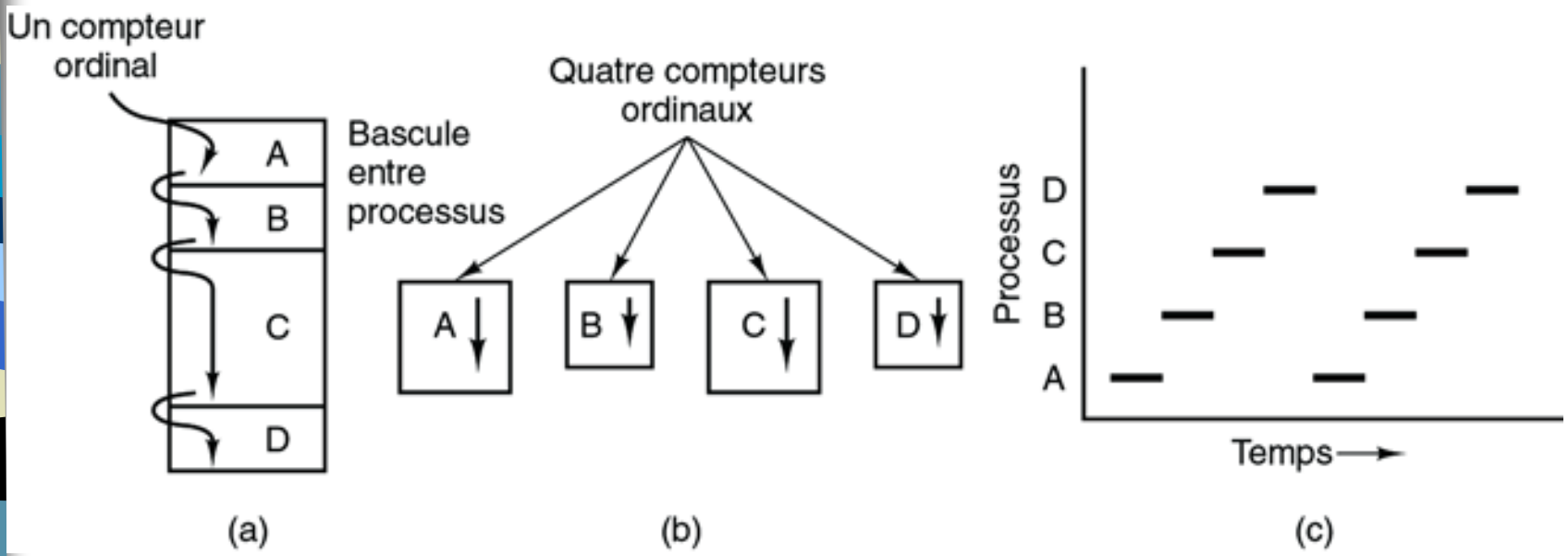
Gestion du processeur

1. Les processus
2. Les threads
3. Politiques d'ordonnancement classiques

Processus

- Modèle de processus
- Création d’un processus
- Fin d’un processus
- Hiérarchie des processus
- Etats des processus
- Implémentation des processus

Processus
Modèle de processus



a) Multiprogrammation de quatre programmes
b) 4 processus indépendants
c) À chaque instant, un seul processus est actif

Processus
Création d'un processus

- Événements causant la création d'un processus:
  - Initialisation du système
  - Exécution d'un appel système demandé par un processus
  - Un usager demande de créer un processus
  - Initiation d'une tâche sur un système de traitement par lots.

➡ La création d'un nouveau processus résulte d'un appel système de création de processus, par un processus existant

Sous UNIX: `fork`

Sous WINDOWS: `CreateProcess`

Processus
Création d'un processus

Sous UNIX

```
// Crée une copie exacte du processus
    appelant
pid_t fork(void)

// Remplace l'image du processus appelant
int execve(
    const char *fichier,
    char * const argv [],
    char * const envp[]
);
```

Processus
La fin d'un processus

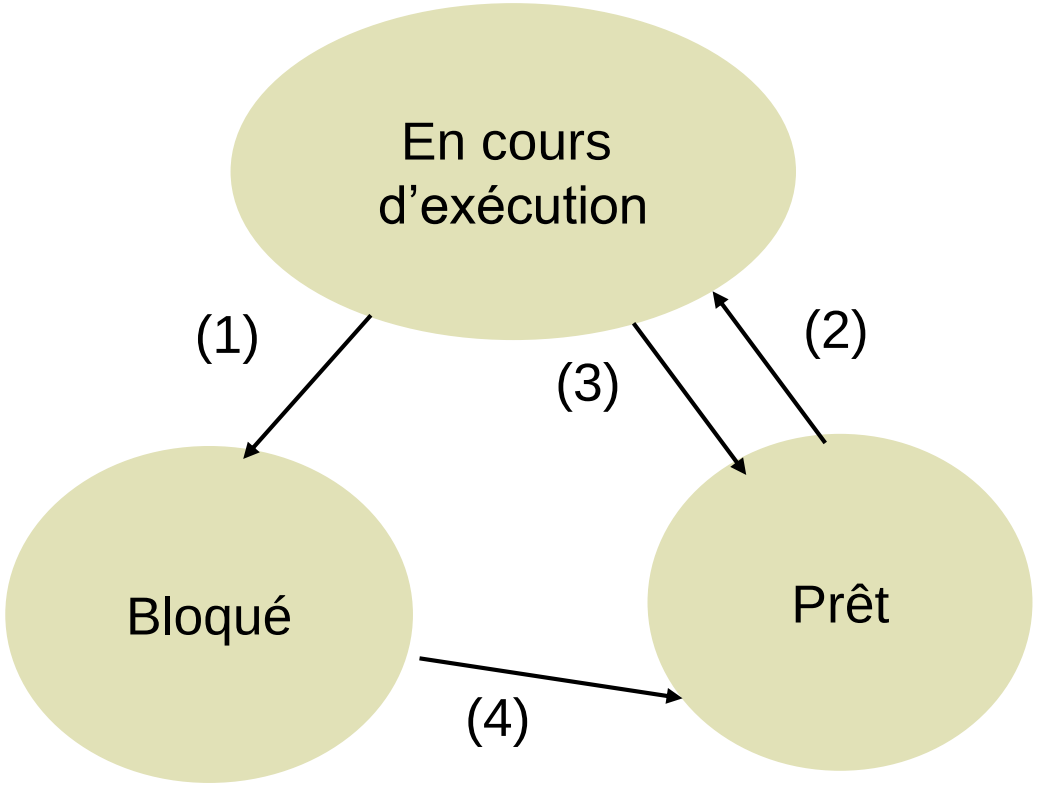
Conditions de fin d'exécution:

1. Arrêt normal (volontaire)
2. Arrêt avec erreur (volontaire)
3. Arrêt avec erreur fatale (involontaire)
4. Tué par un autre processus (involontaire)

Processus
La Hiérarchie des processus

- Notion de processus parent et enfants
- Les processus parent et enfants continuent de s’exécuter de manière associée
- Le processus enfant peut lui-même créer d’autres processus

Processus
Les états des processus



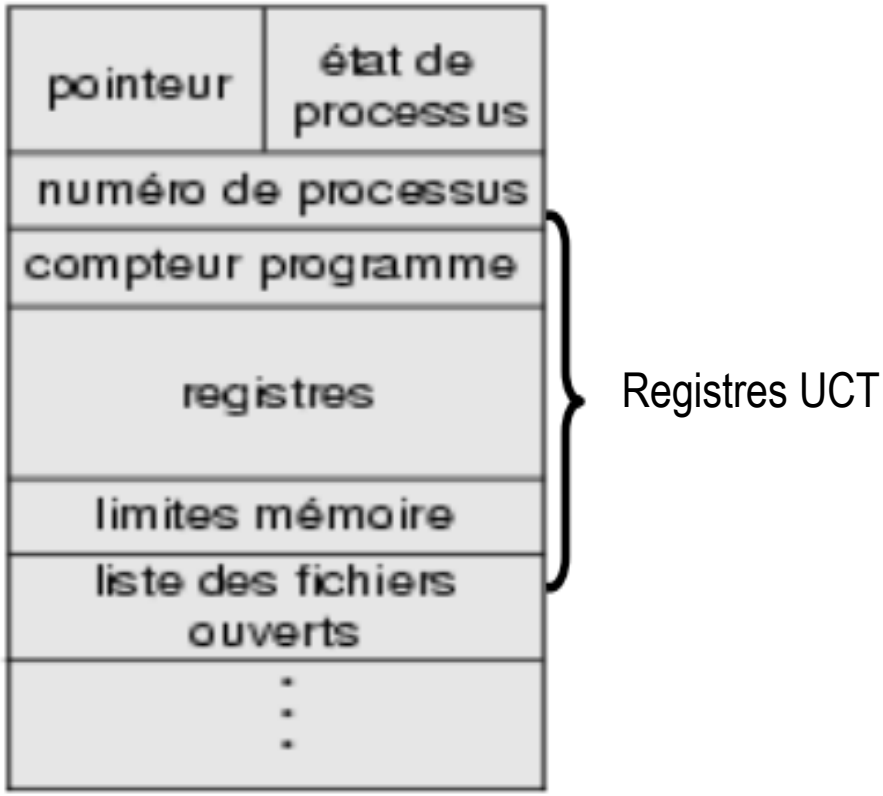
Processus
L'implémentation des processus

Table de processus contenant une entrée par processus:

Gestion du processus	Gestion de la mémoire	Gestion de fichier
Registres	Ptr vers segment de texte	Répertoire racine
Compteur ordinal	Ptr vers segment de données	Répertoire de travail
Etat du processus	Ptr vers segment de la pile	Descripteur de fichiers
Priorité		ID user
Paramètres d'ordonnancement		ID groupe
ID du processus		
Processus parent		
Groupe du processus		
Signaux		
Hre de début du processus		
Temps de traitement utilisé		
Temps de traitement du fils		
Hre de a prochaine alerte		

PCB = Process Control Block

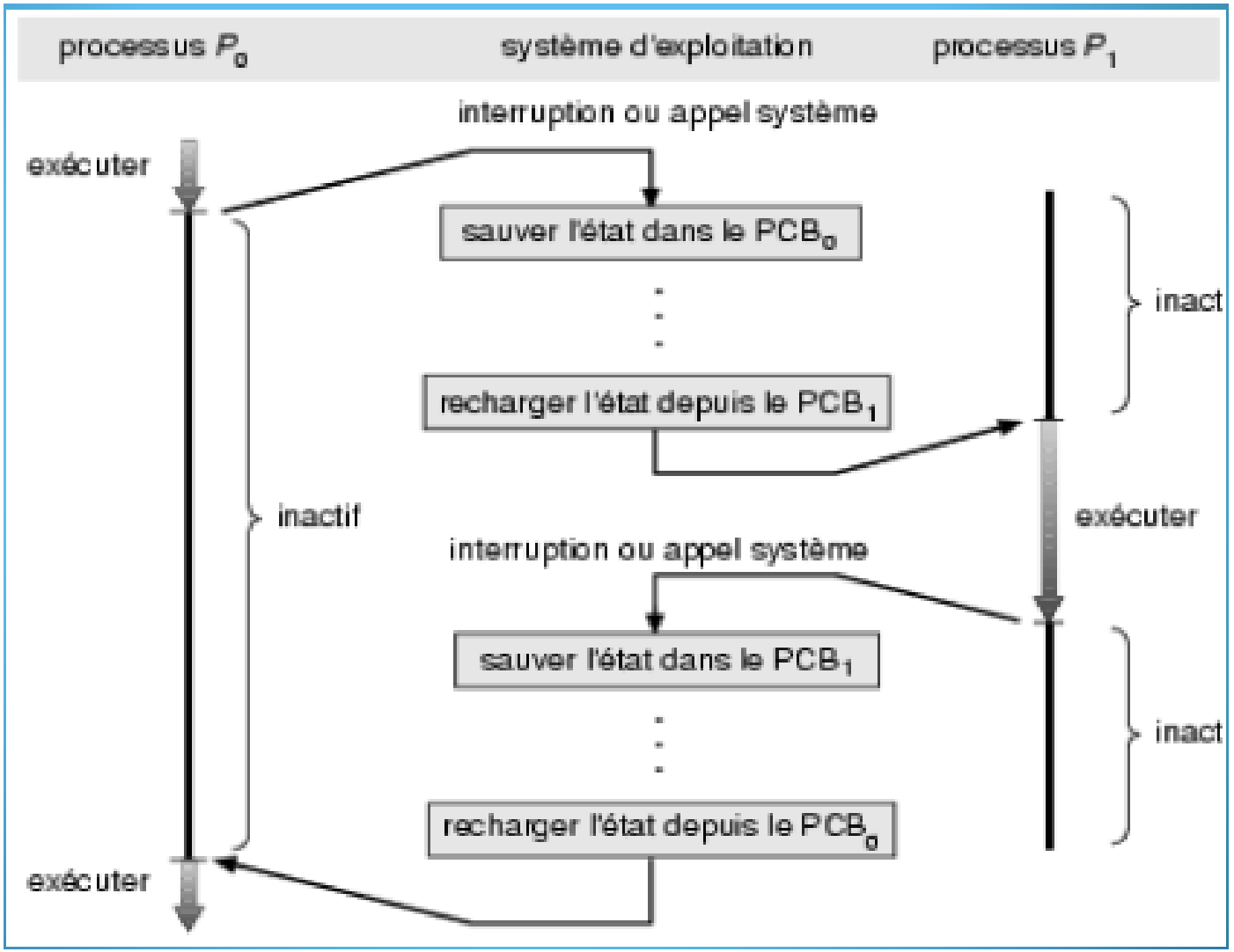
Représente la situation actuelle d'un processus, pour le reprendre plus tard



Process Control Block

- Le PCB contient entre autres:
  - pointeur: les PCBs sont rangés dans des listes enchaînées (à voir)
  - état de processus: ready, running, waiting…
  - compteur programme: le processus doit reprendre à l ‟instruction suivante
  - autres registres UCT
  - bornes de mémoire
  - fichiers qu‟il a ouvert
  - etc.,

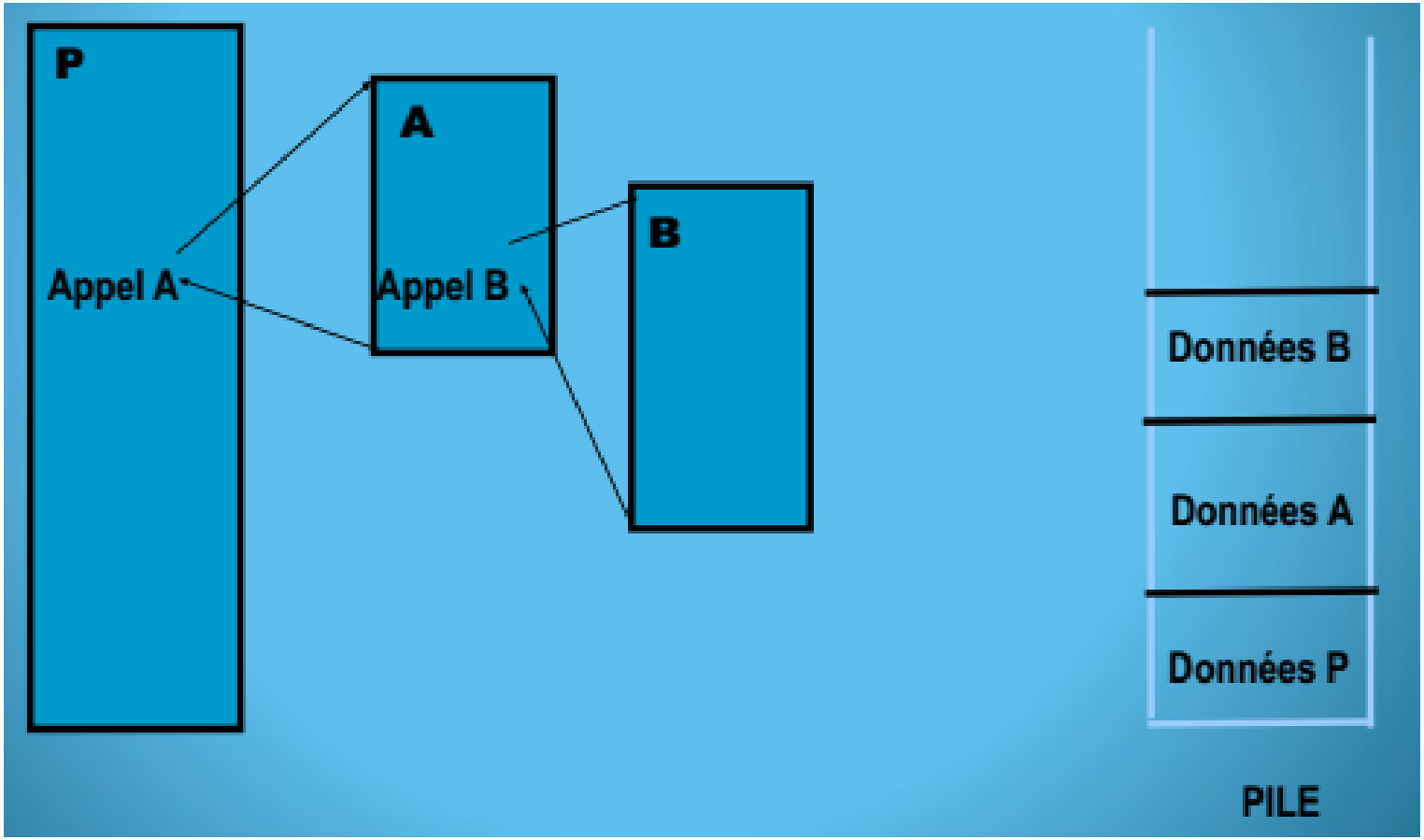
Commutation de processeur (context switching)



La pile d'un processus

- Il faut aussi à sauvegarder entre autre: la pile d'un processus
  - Quand un processus fait appel à une procédure, à une méthode, etc., il est nécessaire de mettre dans une pile l'adresse à laquelle le processus doit retourner après avoir terminé cette procédure, méthode, etc.
  - Aussi on met dans cette pile les variables locales de la procédure qu'on quitte, les paramètres, etc., pour les retrouver au retour
  - Donc il y a normalement une pile d'adresses de retour après interruption et une pile d'adresses de retour après appel de procédure
    - Ces deux piles fonctionnent de façon semblable, mais sont normalement séparées
  - Les informations relatives à ces piles (base, pointeur…) doivent aussi être sauvegardées au moment de la commutation de contexte

La Pile d'un processus



Commutation de processus

- Comme l'ordinateur n'a, la plupart du temps, qu'un processeur, il résout ce problème grâce à un pseudo-parallélisme

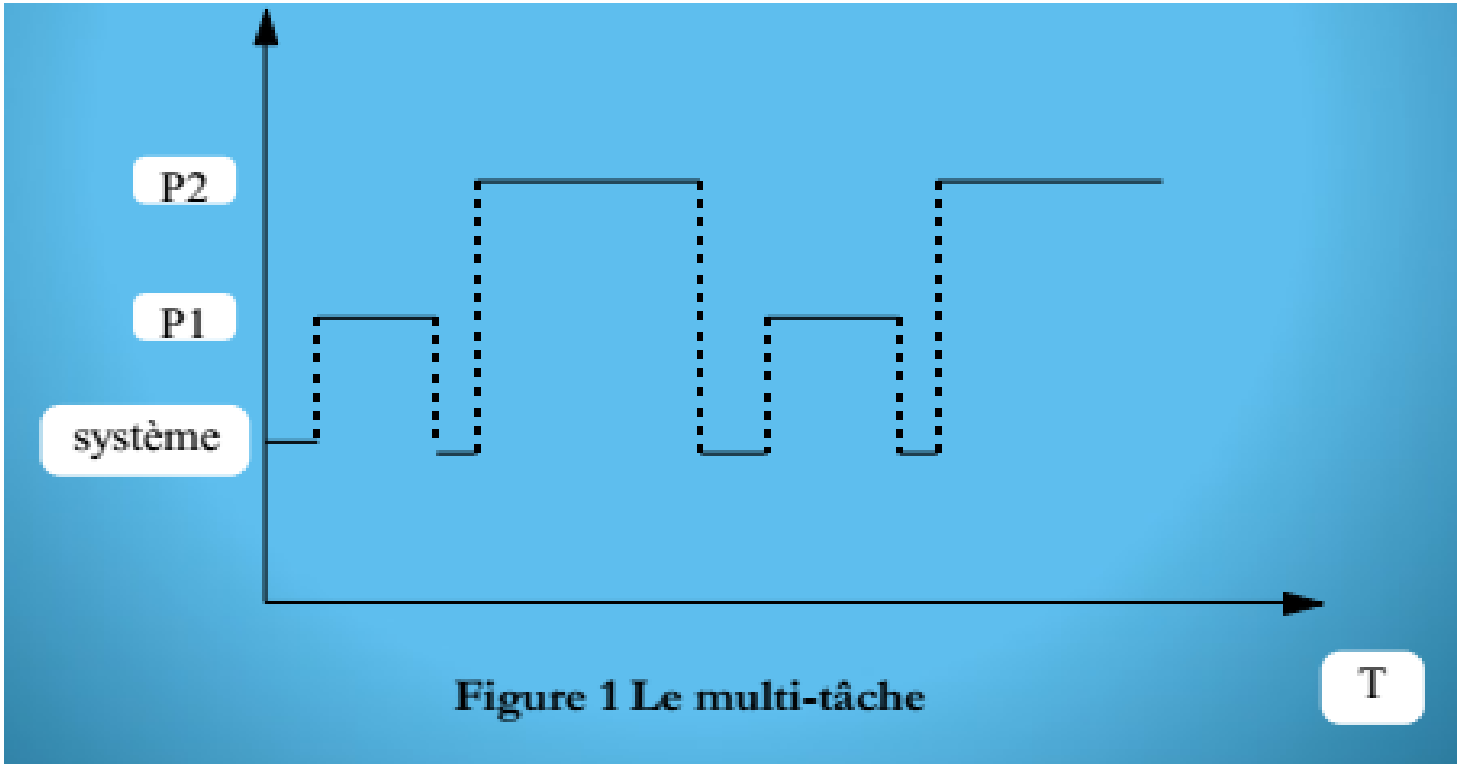


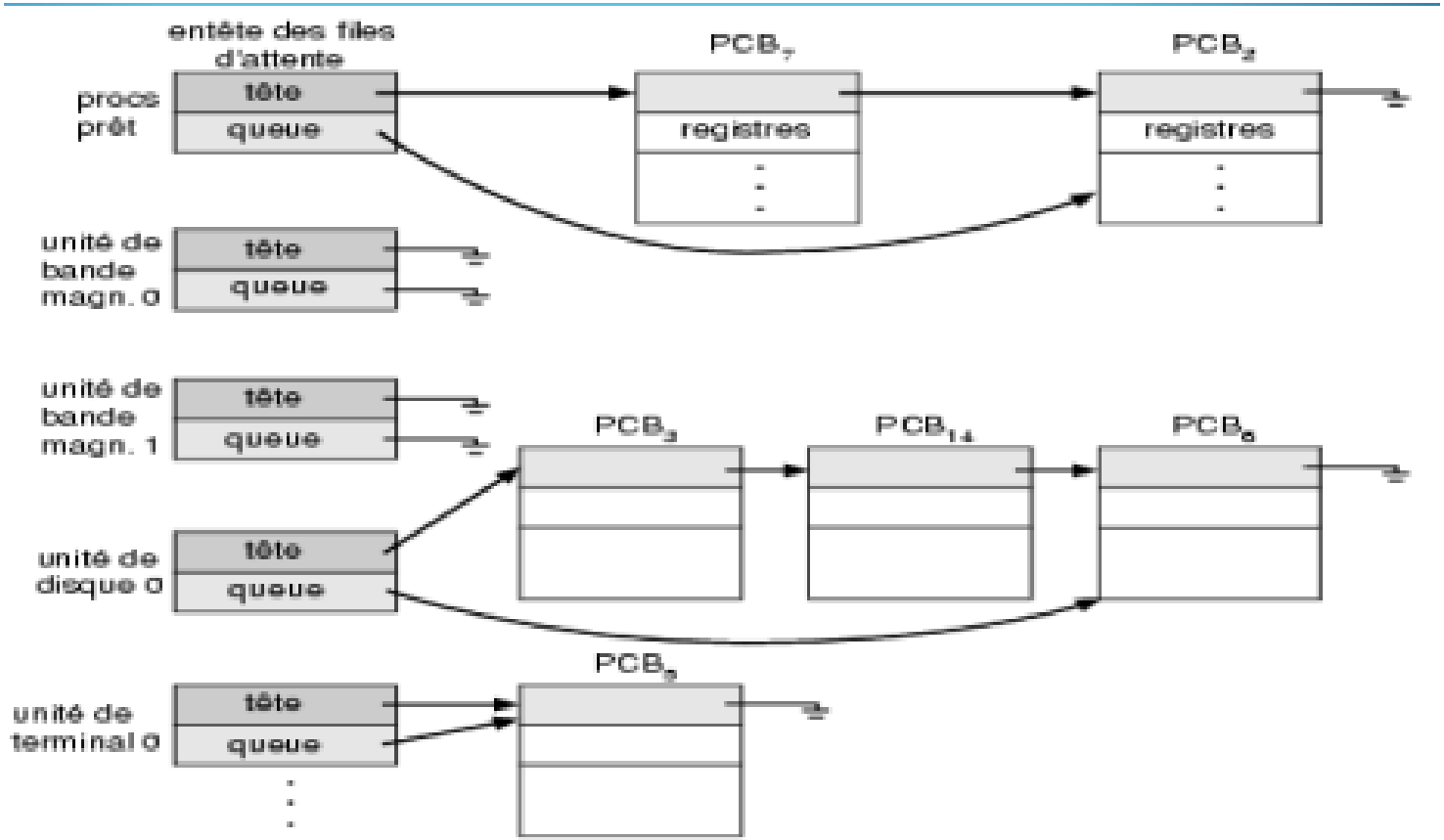
Figure 1 Le multi-tâche

Files d'attente

- Les ressources d'ordinateur sont souvent limitées par rapport aux processus qui en demandent
- Chaque ressource a sa propre file de processus en attente
- En changeant d'état, les processus se déplacent d'une file à l'autre
  - File prêt: les processus en état prêt=ready
  - Files associés à chaque unité E/S
  - etc.

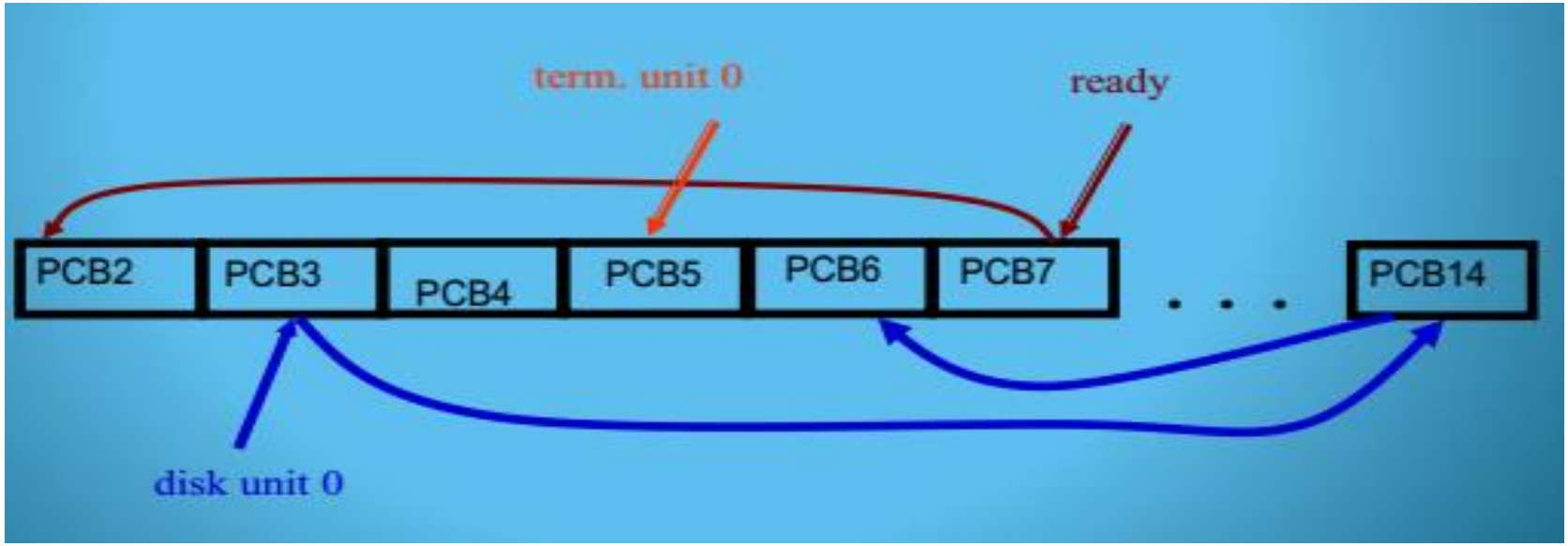
Files d‟attente

- Ce sont les PCBs qui sont dans les files d’attente



Les PCBs

- **Les PCBs ne sont pas déplacés en mémoire pour être mis dans les différentes files: ce sont les pointeurs qui changent.**



Threads

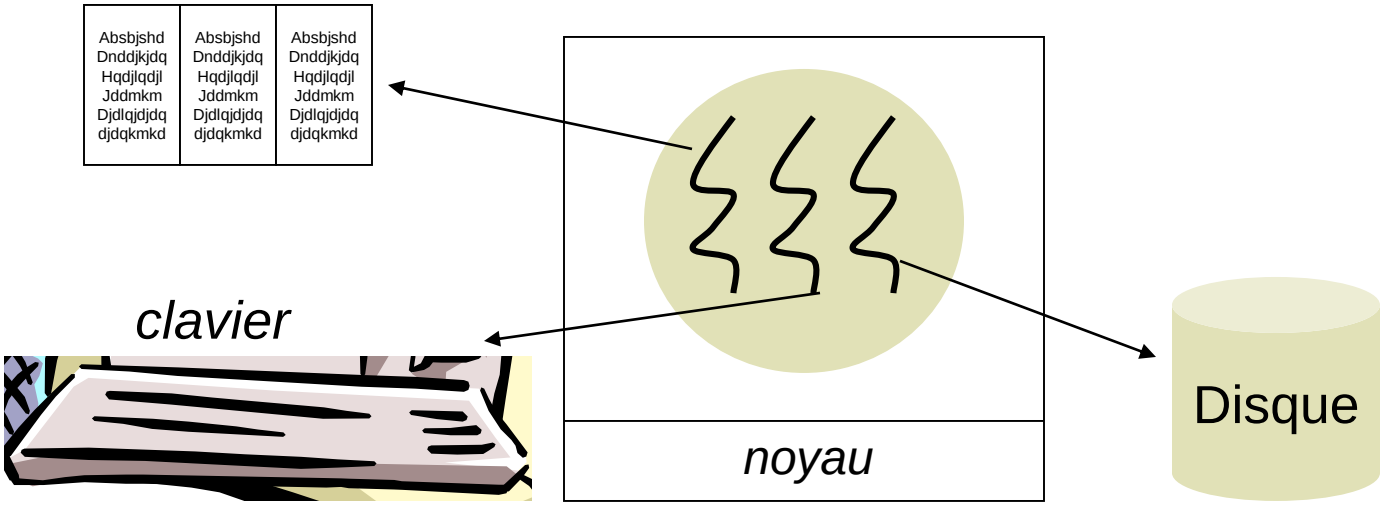
- Notion de threads
- Modèle de thread classique
- Implémentation des threads
- Les threads de POSIX

Threads
Notion de threads

- Un processus possède un espace d'adressage et un thread de contrôle unique.
- Un thread –processus léger, miniprocessus ou *leightweight process* inclut:
  - un **compteur ordinal**, qui effectue le suivi des instructions à exécuter
  - des **registres**, qui détiennent les variables en cours.
  - une **pile**, qui contient l'historique de l'exécution.
- Etats d'un thread: en cours d'exécution, bloqué, prêt ou arrêté.
- Procédures de bibliothèque: *thread_create, thread_exit, thread_wait, thread_yield* (abandon volontaire de la CPU).
- Temps de création d'un processus >> Temps de création d'un thread (100 ×)

Threads
Notion de threads

Exemple:

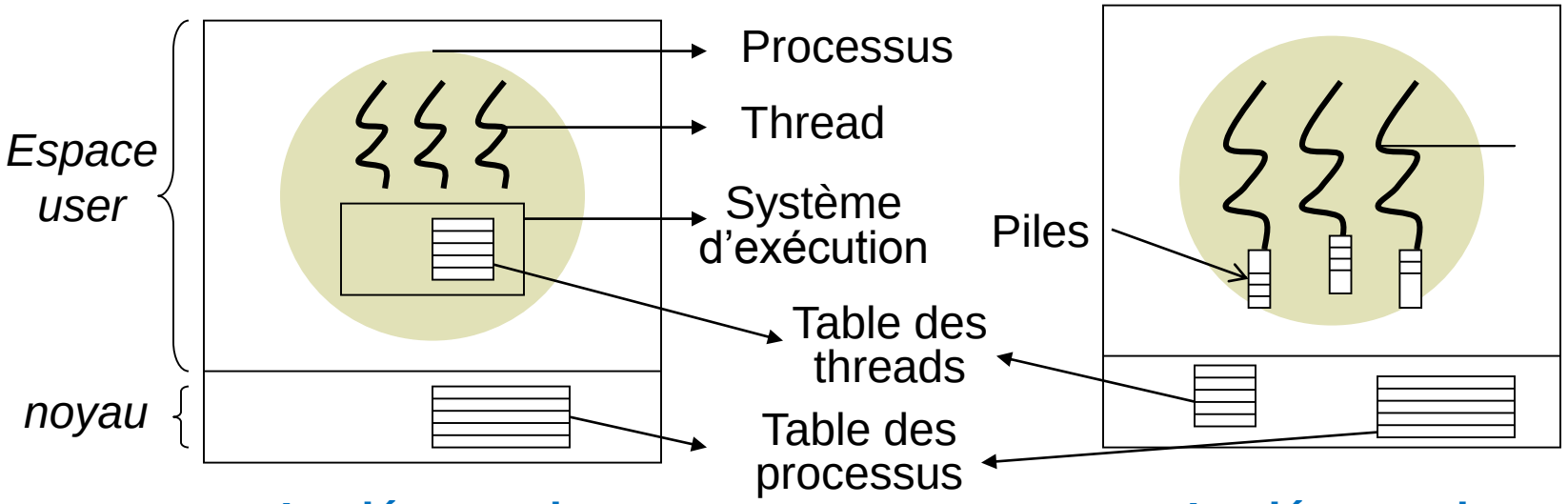


Thread 1**:** remet en forme le document

Thread 2**:** interaction avec l'utilisateur

Thread 3**:** écrit périodiquement le contenu de la RAM sur le disque

Threads
Implémentation des threads

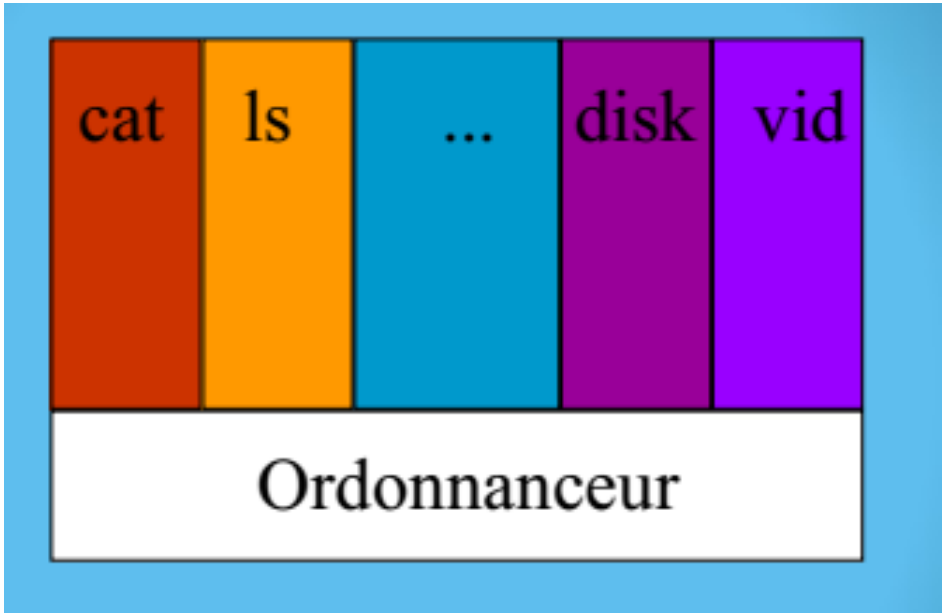


- Système d'exécution: gère la table de threads,
- Le basculement entre threads en cas d'implémentation en espace user est + rapide que celui en mode noyau.
- Implémentation en espace user: le processus a son propre algo d'ordonnancement entre les threads

Threads
les threads de POSIX

Appel	Description
Pthread_create	Crée un nouveau thread
Pthread_exit	Termine le thread appelant
Pthread_join	Attend la fin d'un thread
Pthread_yield	Libère l'UC pour laisser un autre thread s'exécuter
Pthread_attr_init	Crée et initialise une structure attribut de thread
Pthread_attr_destroy	Supprime une structure attribut de thread

Ordonnanceur



Tous les services sont des processus

Ordonnanceurs (schedulers)

Programmes qui gèrent l'utilisation de ressources de l'ordinateur

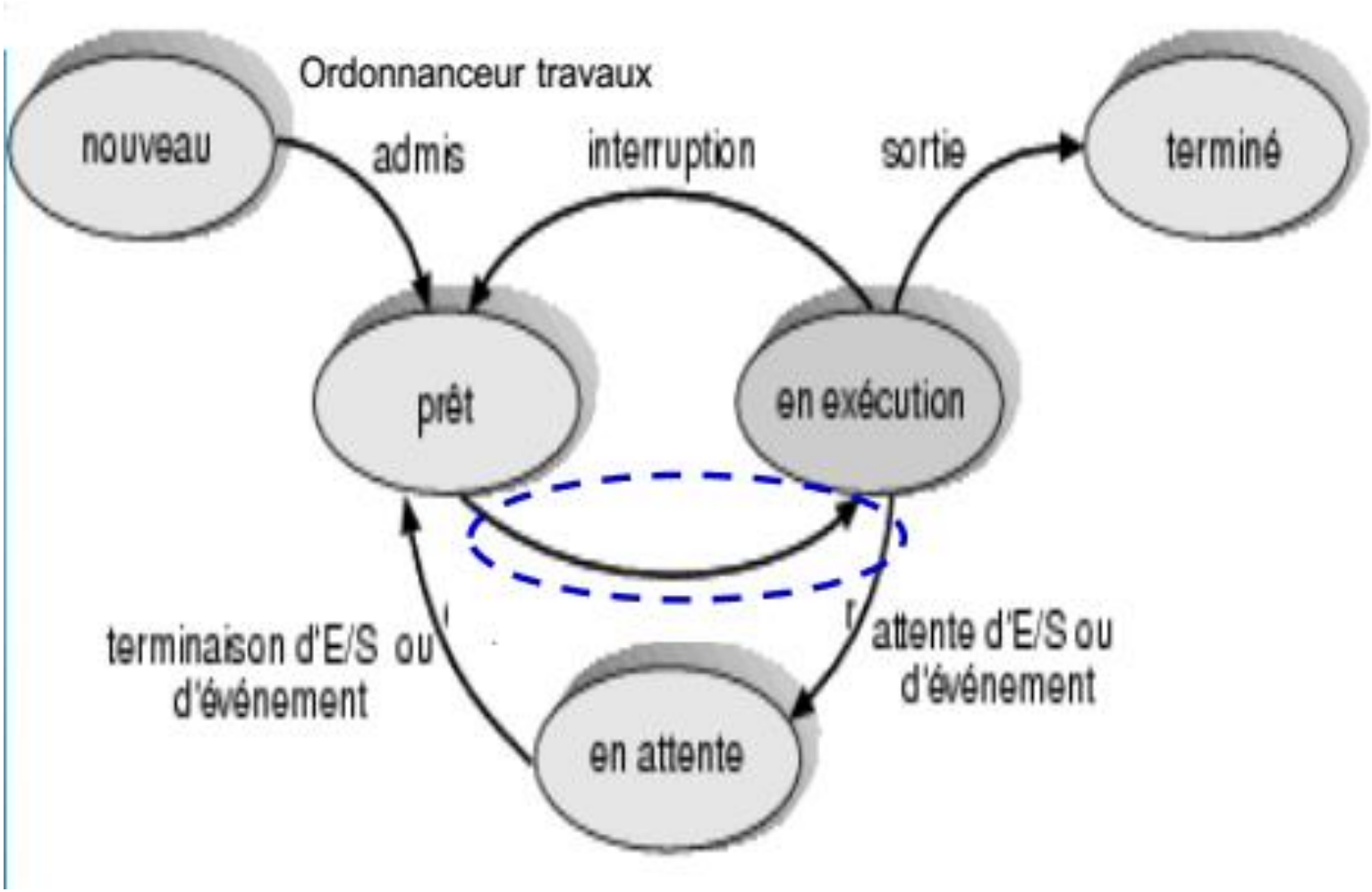
- Trois types d'ordonnanceurs :
  - À court terme = ordonnanceur processus: sélectionne quel processus doit exécuter la transition prêt ⇨ exécution
  - À long terme = ordonnanceur travaux: sélectionne quels processus peuvent exécuter la transition nouveau ⇨ prêt (événement *admitted*) (de spoule travaux à file prêt)
  - À moyen terme: répond au manque de mémoire

Ordonnanceurs

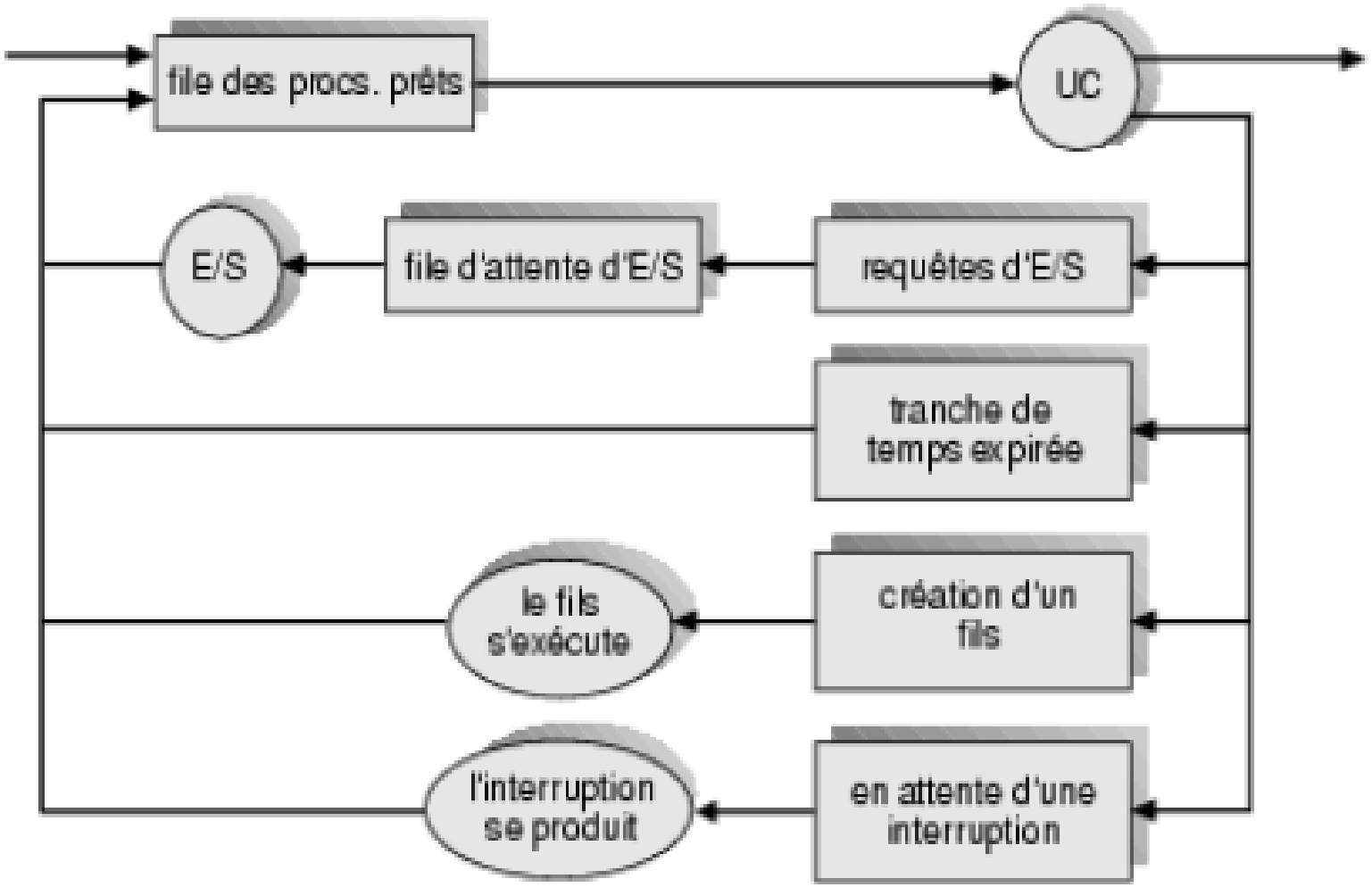
- L'ordonnanceur à court terme est exécuté très souvent (millisecondes), Il faut donc que ça aille très vite
  - typiquement de 1 à 1000 microsecondes
- L'ordonnanceur à long terme doit être exécuté beaucoup plus rarement: il contrôle le niveau de multiprogrammation
  - doit établir une équilibre entre les travaux liés à l'UCT et ceux liés à l'E/S de sorte à ce que les ressources de l'ordinateur soient bien utilisées

Ordonnanceurs

Ordonnanceur travaux = long terme
Ordonnanceur processus = court terme



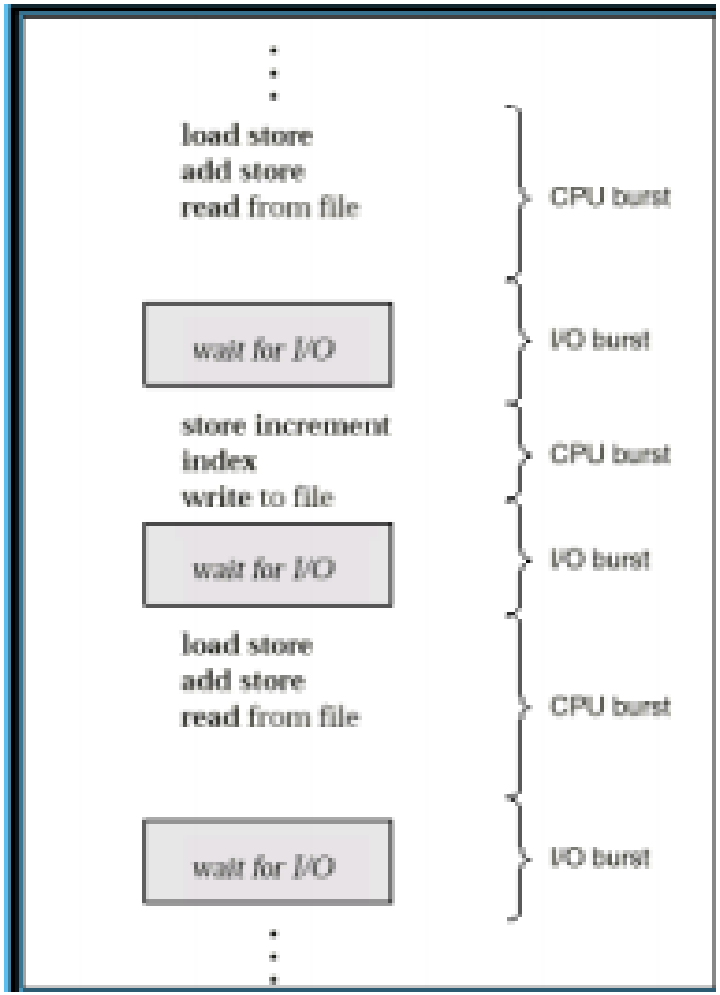
Ordonnancement de processus (court terme)



Ordonnanceur à moyen terme

- Le manque de ressources peut parfois forcer le SE à *suspendre* des processus
  - ils seront plus en concurrence avec les autres pour des ressources
  - ils seront repris plus tard quand les ressources deviendront disponibles
- Ces processus sont enlevés de mémoire centrale et mis en mémoire secondaire, pour être repris plus tard
- «swap out», «swap in» , va-et-vient

Alternance CPU E/S



Cycles (bursts) d'UCT et E/S: l'exécution d'un processus consiste de séquences d'exécution sur UCT et d'attentes E/S

Quand invoquer l‟ordonnanceur

- Choisir un processus parmi ceux qui sont prêts et lui donner les ressources CPU.
- L'ordonnancement a lieu quand un processus :
  1. Se termine.
  2. Passe de l'état "actif" à "attente".
  3. Passe de l'état "actif" à "prêt".
  4. Passe de l'état "attente" à "prêt".

- Un nouveau processus doit être choisi pour 1 et 4
- Pour 2 et 3 : mode préemptif ou non préemptif.

Politiques d'ordonnancement classiques

- Ordonnancement classique de l'UC
- Ordonnancement des taches dépendantes
- Ordonnancement des systèmes temps réel

Politiques d'ordonnancement classiques
Ordonnancement

Ordonnanceur (*scheduler*): partie du SE qui sélectionne les processus.

Algo d'ordonnancement (*scheduling algorithm*)

- ~ non-préemptif: sélectionne un processus, puis le laisse s'exécuter jusqu'à ce qu'il se bloque (E/S, wait) ou se termine.

- ~ préemptif: sélectionne un processus et le laisse s'exécuter pendant un quantum, préemption par l'interruption horloge

Comportement de Processus:
- Processus de traitement
- Processus d'E/S

Politiques d'ordonnancement classiques
Objectifs de l'ordonnancement

Tous les systèmes

Équité: attribuer à chaque processus un temps CPU équitable

Équilibre: toutes les parties du système sont occupées

Systèmes de traitement par lots

Capacité de Trait: optimiser le #jobs/ heure

Délai de rotation: réduire le délai entre la soumission et l'achèvement

Taux d'utilisation du processeur

Systèmes interactifs

Temps de réponse: réponse rapide aux requêtes

Systèmes temps réel

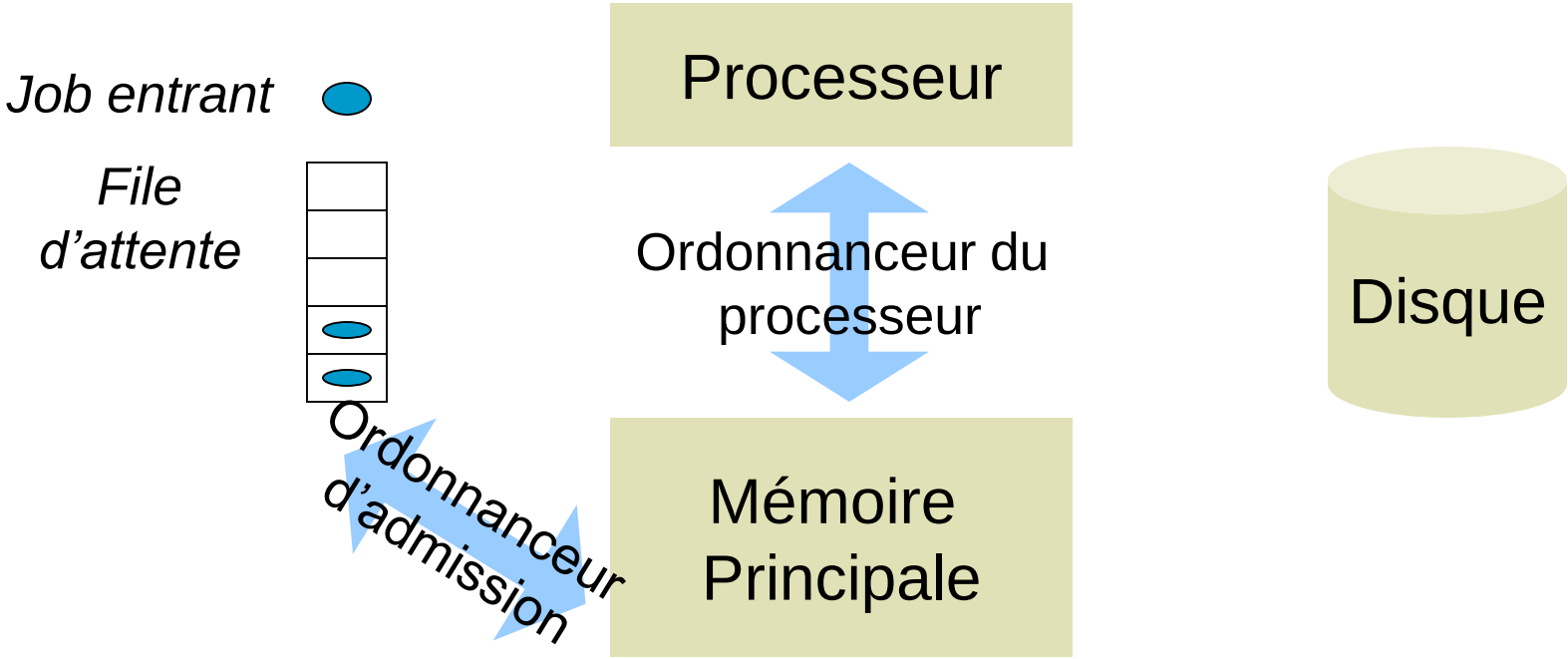
Respect des délais: éviter de perdre les données

Prévisibilité: éviter la dégradation de la qualité

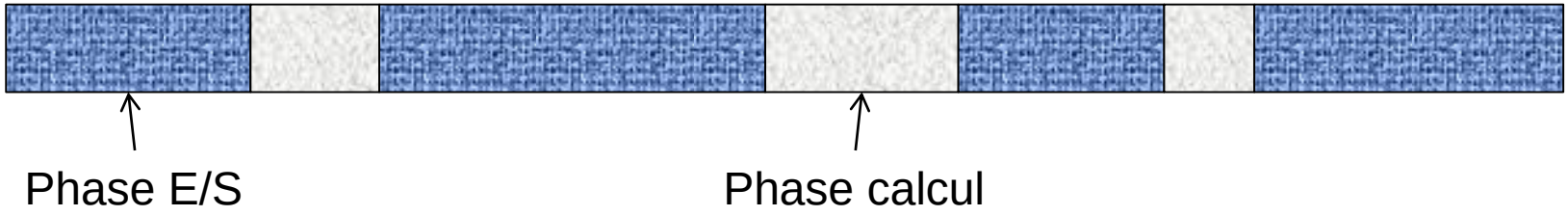
Politiques d'ordonnancement classiques
Rôle de l'ordonnanceur

- Optimiser
  - **Taux d'utilisation de l'UC:** rapport entre la durée où l'UC est active et la durée totale
  - **Débit:** Nbre de programmes users traités en moyenne par unité de temps,
  - **Temps de traitement moyen:** moyenne des intervalles de temps séparant la soumission d'une tâche de sa fin d'exécution
  - **Temps de traitement total:** d'un ensemble de processus donné
  - **Temps de réponse maximum:** durée max séparant la soumission d'une requête par un processus et son accomplissement

Politiques d'ordonnancement classiques
Ordonnancement à 3 niveaux



Politiques d'ordonnancement classiques
Modèle de système de tâche d'un processus



- Processus = tâche unique ou une suite de tâches
- Bloc de contrôle (BCT) décrit chaque tâche
- à chaque tâche sont assignés 2 réels τ_i (indicatif de sa durée d'exécution) et t_i (sa date d'arrivée dans la file d'attente)
- l'algo d'ordonnancement construit une **assignation:** description de l'exécution des tâches sur le ou les processeurs

Politiques d'ordonnancement classiques
Modèle de système de tâche d'un processus

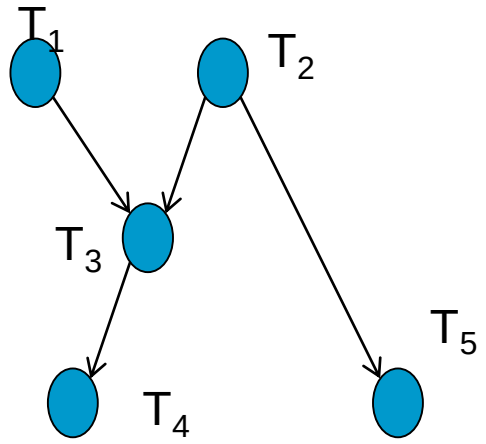
- l'assignation est représentée par un diagramme de **Gantt**

Exemple: soit un système à 2 processeurs dans lesquelles évoluent 5 tâches

	τ_i	t_i
T_1	1	0
T_2	2	0
T_3	1	0
T_4	1	0
T_5	2	0

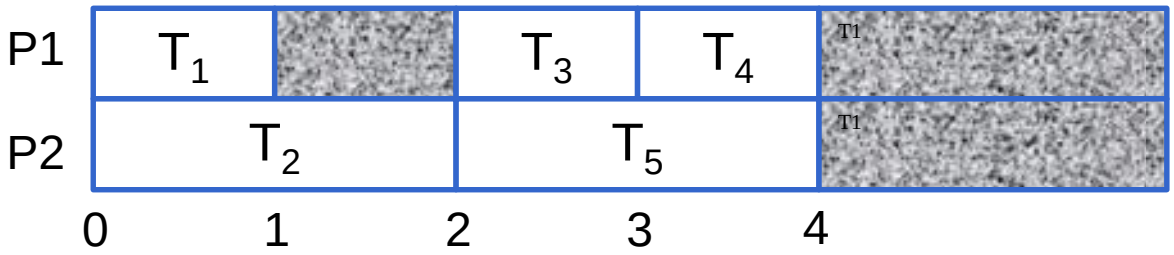
Tableau des tâches

	Processeur	instants
T_1	P1	0,1
T_2	P2	0,2
T_3	P1	2,3
T_4	P1	3,4
T_5	P2	2,4



Graphe de précédence

Diagramme de Gantt



Politiques d'ordonnancement classiques

- Tâches dépendantes
- Tâches indépendantes

Politiques d'ordonnancement classiques
Algorithmes sans réquisition

1) FIFO
2) Ordre inverse des temps d'exécution (PCTE)

Politiques d'ordonnancement classiques
Algorithmes sans réquisition

1) FIFO

Exemple

	τ_i	t_i
T_1	30	0
T_2	5	ε
T_3	2	2ε

ε: intervalle très petit par rapport à l'intervalle choisi

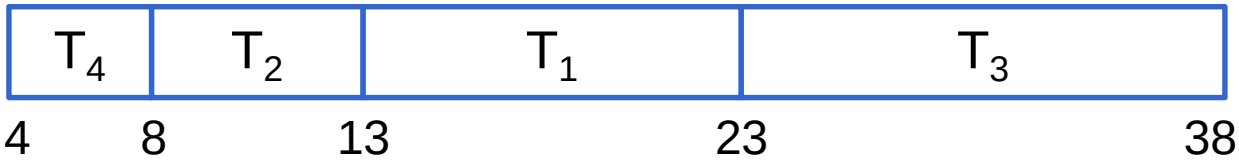
$$t_{moyen} = \frac{30 + (35 - \varepsilon) + (37 - \varepsilon)}{3} = \frac{102 - 3\varepsilon}{3} \approx 34$$

Politiques d'ordonnancement classiques
Algorithmes sans réquisition

2) **PCTE:** choisir la tâche de plus court temps d'exécution

Exemple

	τ_i	t_i
T_1	10	0
T_2	5	2
T_3	15	3
T_4	4	4



$$t_{moyen} = \frac{(8-4)+(13-2)+(23-0)+(38-3)}{4} = 18,25$$

Algo intéressant si toutes les tâches sont présentes dans la file d'attente,

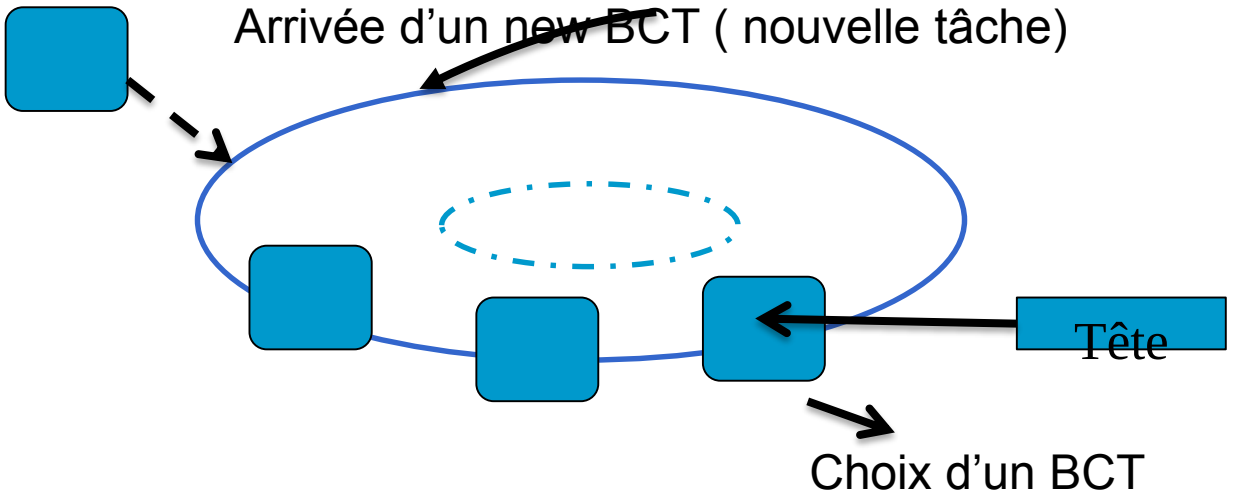
Politiques d'ordonnancement classiques
Algorithmes avec réquisition

1) **Tourniquet**
2) **Plus court temps d'exécution restant (PCTER)**

Politiques d'ordonnancement classiques
Algorithmes avec réquisition

1) Tourniquet (Round Robin)

1) Aussi appelé "balayage cyclique".
2) Les processus accèdent au processeur, chacun à leur tour, pour un temps déterminé à l'avance (le quantum).
3) Un processus en attente d'une entrée-sortie sera placée dans une file des bloqués.



Organisation d'un tourniquet

Politiques d'ordonnancement classiques
Algorithmes avec réquisition

1) Tourniquet

Exemple

	τ_i	t_i
T_1	30	0
T_2	5	ε
T_3	2	2ε

1) dessinez l'assignation produite avec un quantum de 1 et de 10

2) Calculer le temps moyen

$$t_{moyen} = \frac{37+12+6-3\varepsilon}{3} \approx 18,33$$

$$t_{moyen} = \frac{37+15+17-3\varepsilon}{3} \approx 23$$

Politiques d'ordonnancement classiques
Algorithmes avec réquisition

1) Tourniquet

- Un quantum trop grand augmente les temps de réponse
- Un quantum trop petit multiplie les commutations de contexte

Politiques d'ordonnancement classiques
Algorithmes avec réquisition

2) PCTER

- généralisation avec réquisition du PCTE
- Nouvelle entrée dans la table de tâches: durée d'exécution restante τ_i'
  - Initialisée à τ_i
  - À la fin du quantum, l'ordonnanceur soustrait à la valeur τ_i', de la tâche choisie, la valeur du quantum
- fournit la valeur optimale de la durée moyenne de traitement

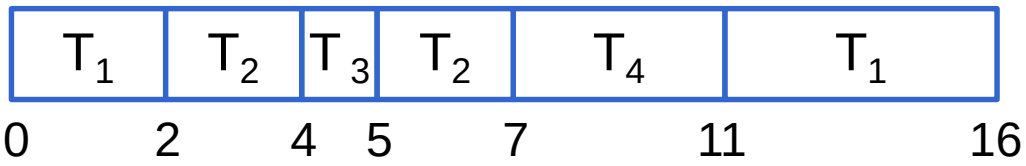
Politiques d'ordonnancement classiques
Algorithmes avec réquisition

2) PCTER

	τ_i	t_i
T_1	7	0
T_2	4	2
T_3	1	4
T_4	4	5

1) dessinez l'assignation produite

2) Calculer le temps moyen



$$t_{moyen} = \frac{16 + (7-2) + (5-4) + (11-5)}{4} = 7$$