Université Mohamed Premier Faculté des Sciences Département d'Informatique Oujda

Année universitaire 2020/2021

Cours des systèmes d'exploitation II

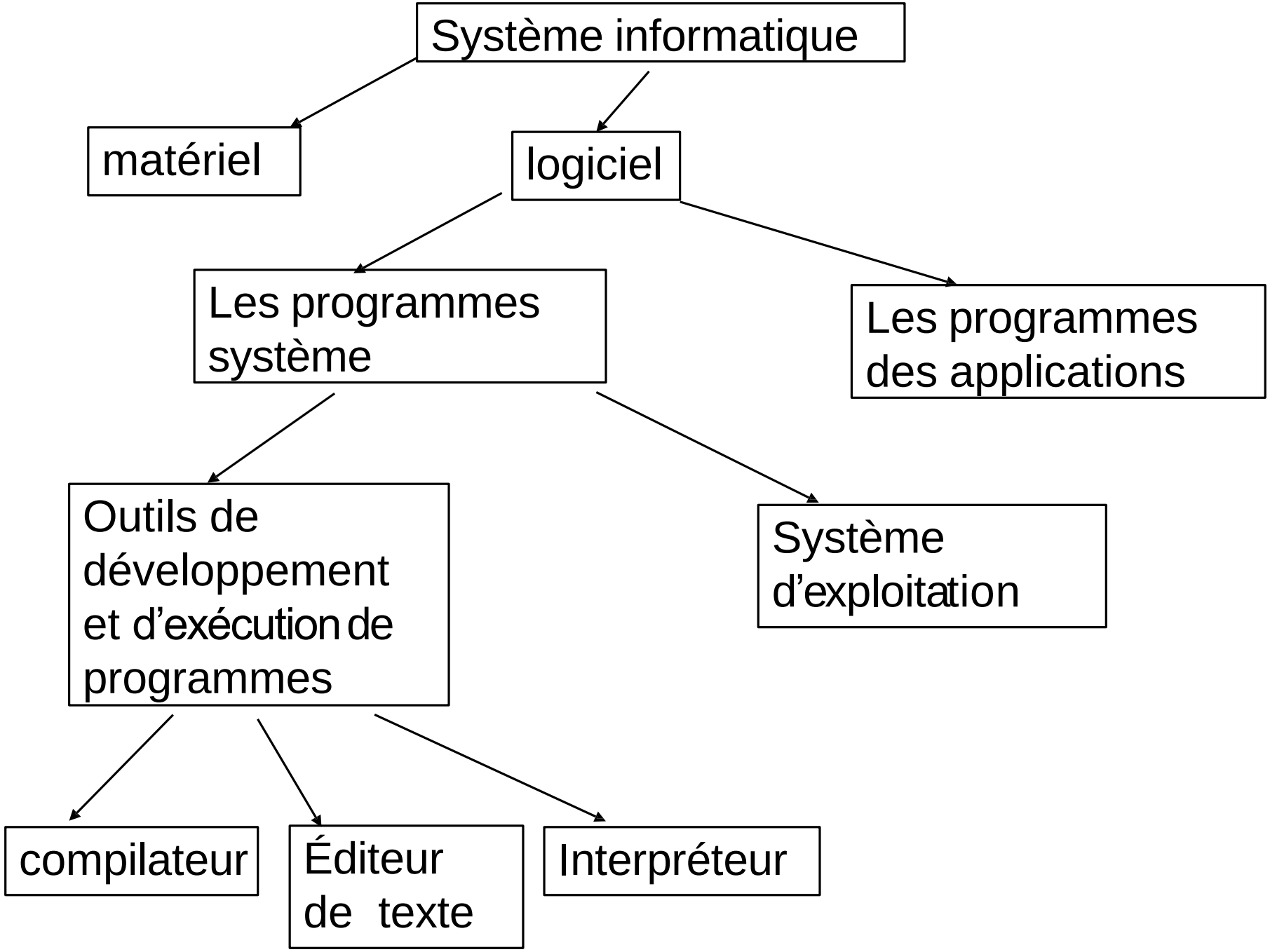
Filière SMI
Quatrième semestre

Professeur : Aicha.KERFALI

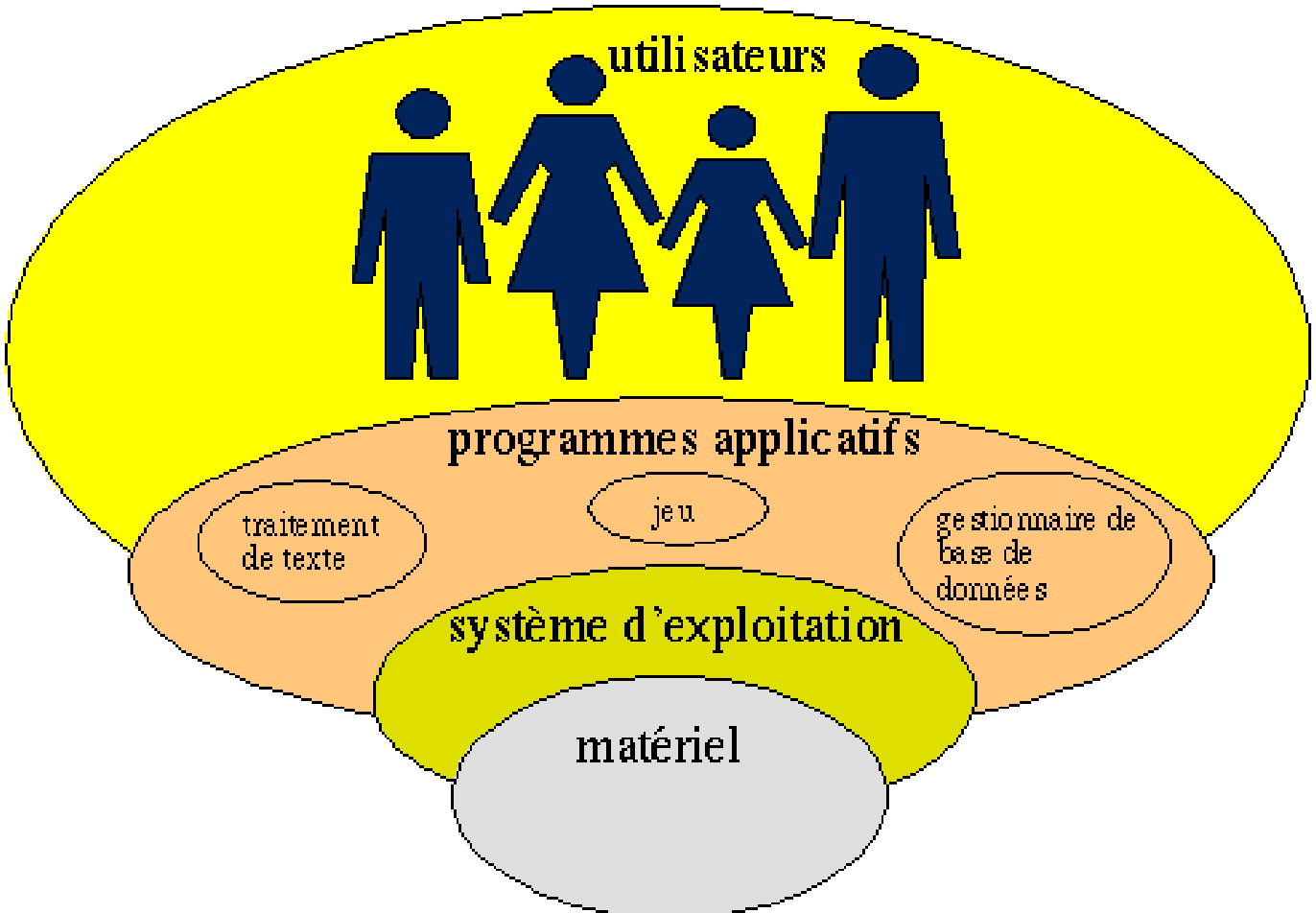
Chapitre1 généralités

I. Système informatique

- Un système informatique est constitué du matériel est du logiciel.
- Son objectif est de permettre le traitement automatique de l'information


Système informatique
matériel
logiciel
Les programmes système
Les programmes des applications
Outils de développement et d'exécution de programmes
Système d'exploitation
compilateur
Éditeur de texte
Interpréteur

Les programmes d'application accèdent au matériel à travers différentes couches logicielles. Le système d'exploitation constitue la couche intermédiaire entre le matériel et les programmes des applications.



- Les tâches d’un systèmes d’exploitation
  - La gestion du processeur,
  - La gestion de la mémoire centrale,
  - La gestion de la mémoire secondaire,
  - La gestion des entrées/sorties
  - La gestion du réseau
- Les systèmes d'exploitation se diffèrent par l'interface qu'ils proposent et les algorithmes et stratégies qu'ils appliquent

II. Notion de processus

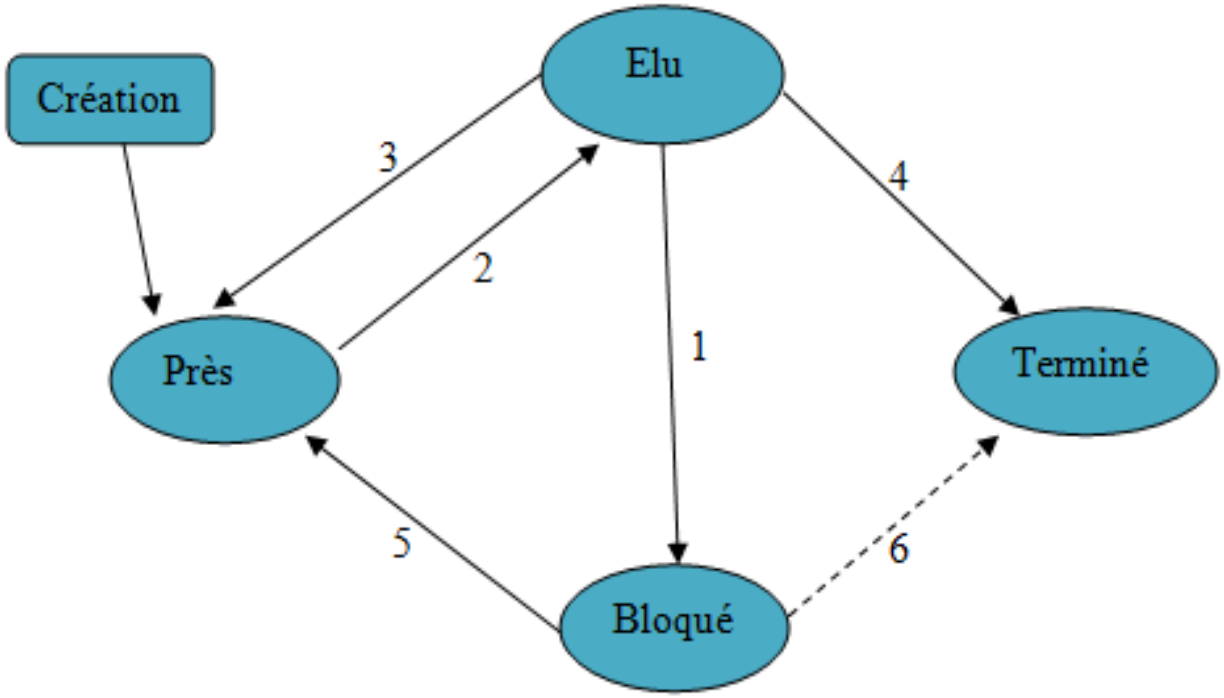
Définition Un processus est un programme en cours d'exécution. Il possède son propre compteur ordinal, ses registres et ses variables.

- Le processus est crée par le système d'exploitation ou l'utilisateur au moment où l'exécution du programme doit commencer,
- Une fois le processus terminé, il est supprimé par le système d'exploitation,
- Un programme a une existence statique, il est stocké sur le disque puis chargé en mémoire afin d’être exécuté.
- Le processus en revanche a un contact direct avec le processeur en effet c'est l'entité exécutée par le processeur

Les principales fonctionnalités du système d'exploitation en matière de gestion de processus consistent à:

- La création, suppression et interruption de processus,
- L'ordonnancement des processus afin de garantir un ordre d’exécution équitable entre les utilisateurs tout en privilégiant les processus du système,
- La synchronisation entre les processus ainsi que la communication,
- La protection des processus d'un utilisateur contre les actions d'un autre utilisateur

Etats d'un processus



La transition 1 a lieu quand le processus ne peut plus poursuivre son exécution car il a besoin d'une ressource non disponible par exemple : il passe à l'état en attente.

La transition 2 a lieu quand le processus est sélectionné par l'ordonnanceur pour être exécuté

La transition 3 signale que le système d'exploitation a sélectionné un autre processus pour l'exécuter.

La transition 4 indique que le processus a fini son exécution.

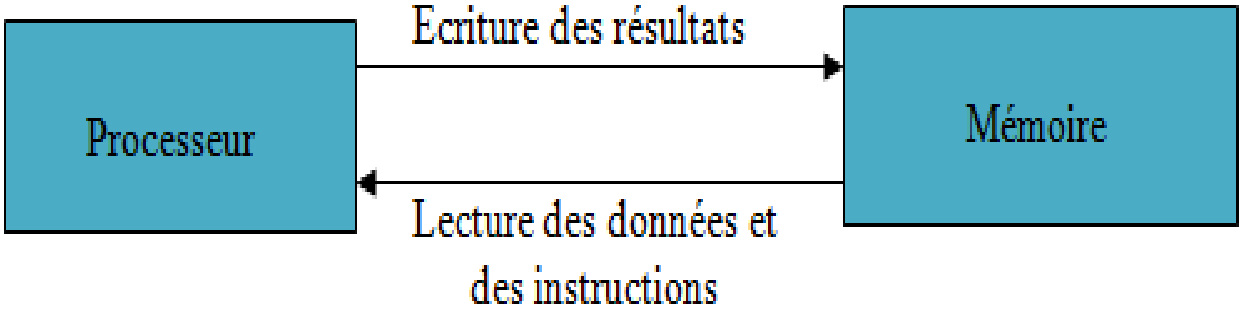
La transition 5 a lieu lorsque le processus n'a plus de raison d'être bloqué; par exemple les données deviennent disponibles.

La transition 6 a lieu quand l'événement attendu par le processus ne peut se réaliser. Il est donc inutile de faire patienter davantage ce processus

Pour exécuter les instructions d'un programme, le processeur va réaliser en boucle ce qu'on appelle le cycle *fetch-decode-execute* et qui consiste dans les grandes lignes à :

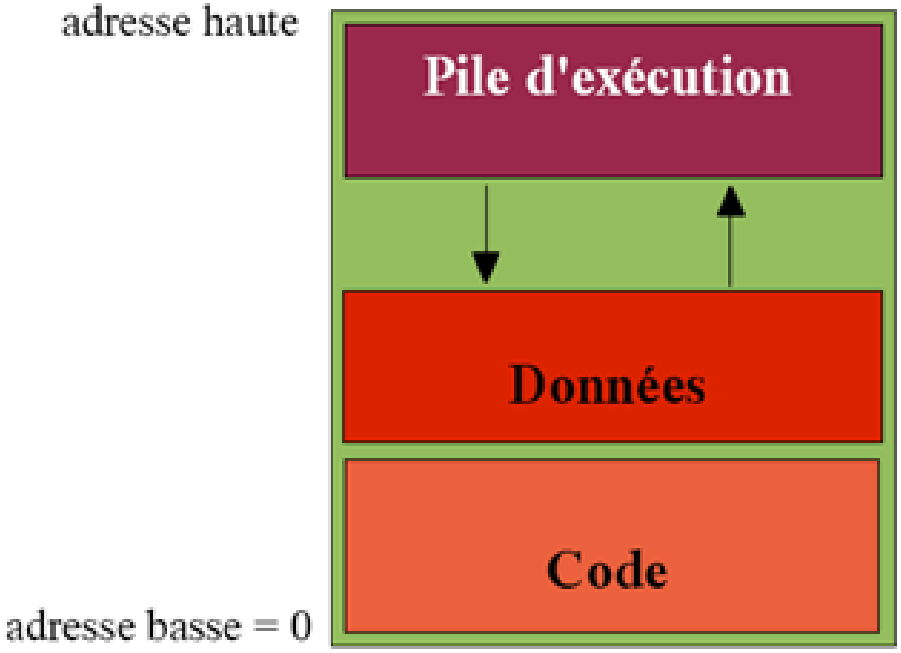
- Charger l'instruction dont l'adresse est contenue dans le compteur ordinal depuis la mémoire où est stocké le code à exécuter puis incrémenter le compteur (*fetch*) pour passer à l'instruction suivante,
- Décoder l'instruction (*decode*),

-Exécuter l'instruction (*execute*). Pendant cette phase le processeur peut interagir avec la mémoire pour lire les données et écrire les résultats. Il utilise un ensemble de registres pour stocker les données et leurs adresses



Schémas simplifié d'interaction processeur-mémoire

Structure de l'espace mémoire d'un processus



Le processus dans la RAM

Le code correspond aux instructions, en langage machine, du programme à exécuter.
La zone de données contient les variables globales ou statiques du programme ainsi que les allocations dynamiques de mémoire.
Enfin, les appels de fonctions, avec leurs paramètres et leurs variables locales, viennent s'empiler sur la pile.

Le contexte d'un processus

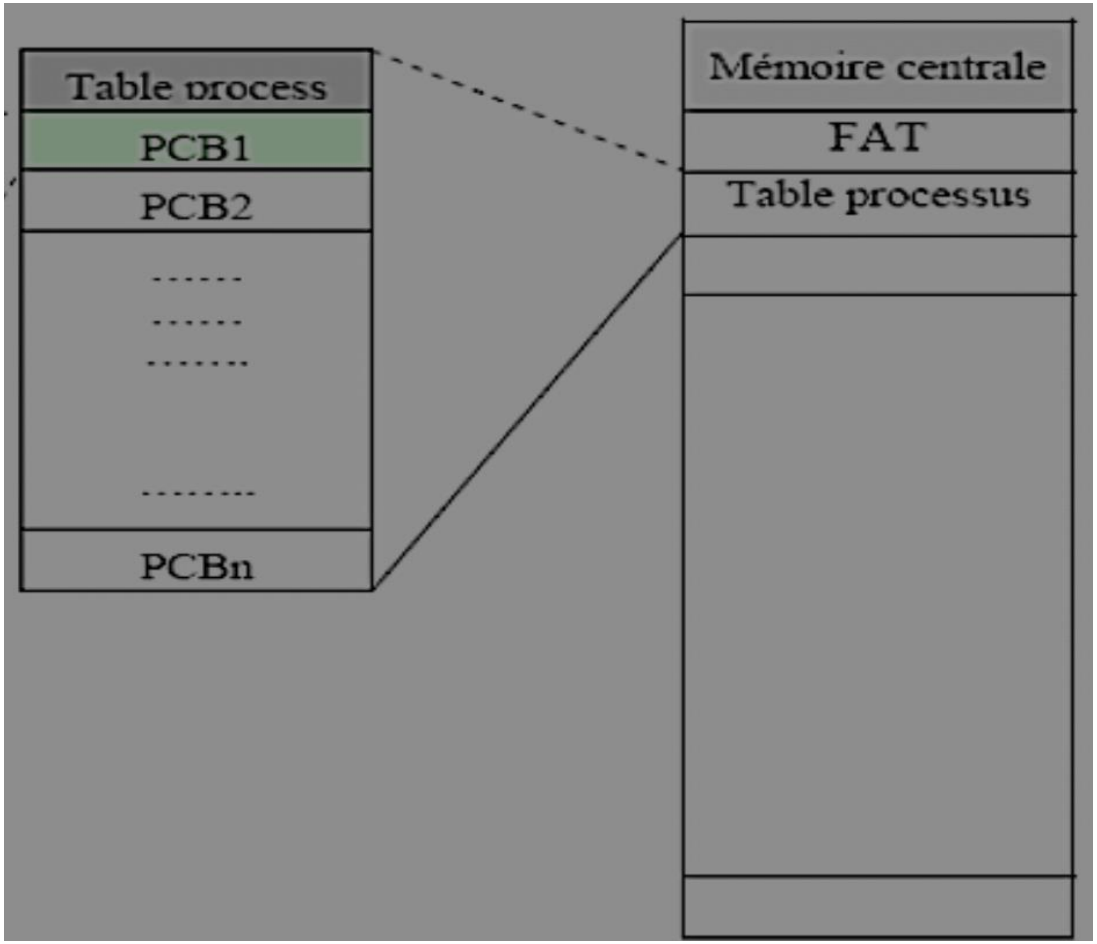
En informatique, **un contexte d'exécution** d'un Processus est l'ensemble minimal de données sauvegarder pour permettre une interruption du processus à un moment donné, et une reprise de son exécution au point où il a été interrompu. Il comporte les informations suivantes :

- Le contenu du compteur ordinal : adresse de la prochaine instruction à exécuter par le processeur
- Les contenus des registres généraux : ils contiennent les résultats calculés par le processus
- Les registres qui décrivent l'espace qu'il occupe en mémoire centrale
- Le registre variable d'état qui indique l'état du processus
- D'autres informations telle que la priorité du processus,

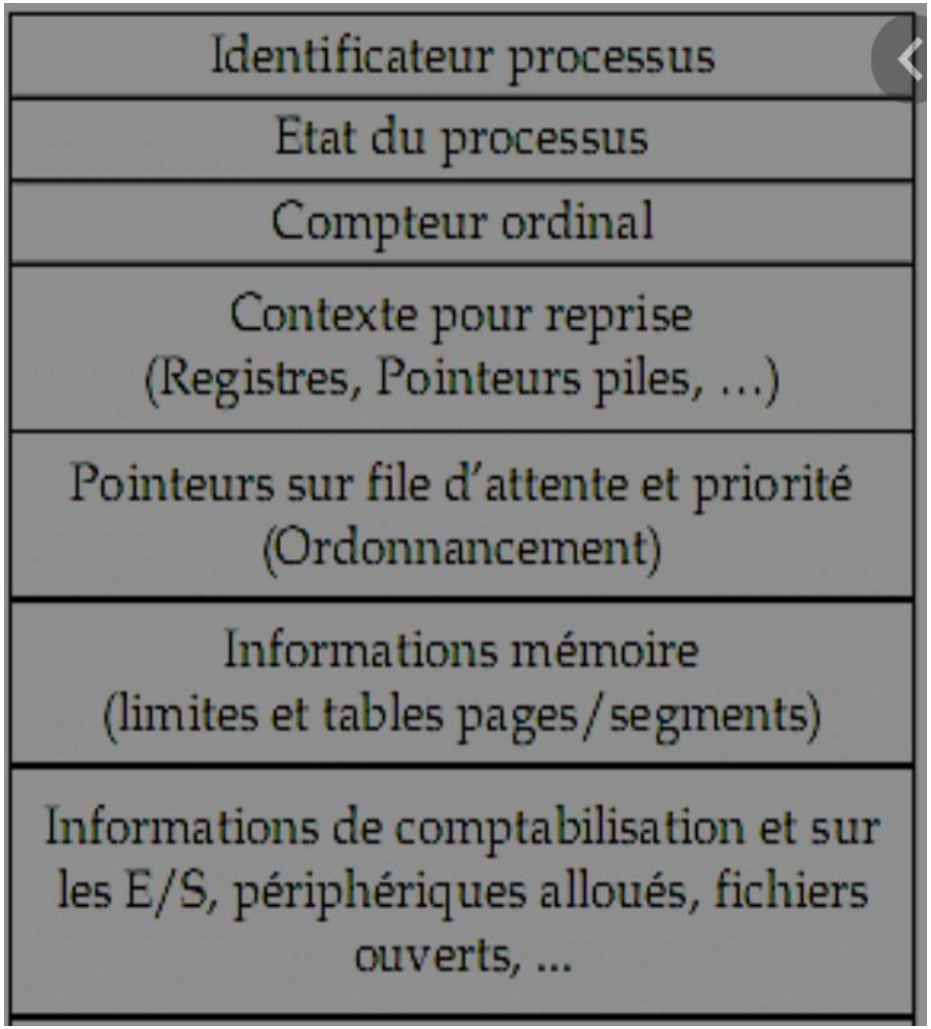
L'opération qui consiste à sauvegarder le contexte d'un processus et à copier le contexte d'un autre processus dans l'unité centrale s'appelle changement ou commutation de contexte

Table des processus

Pour la gestion des processus, le système détient en mémoire centrale une table appelée table des processus(table process). Chaque entrée dans cette table contient le bloc de contrôle d'un processus (PCB)



Structure du PCB d'un processus



III. Les modes superviseur et utilisateur

Afin de protéger l'exécution du système d'exploitation de celles des processus utilisateurs les processeurs proposent deux modes de fonctionnement :

- *Le mode utilisateur* dans lequel les processus utilisateurs sont exécutés
- *Le mode protégé* ou superviseur (également appelé mode noyau) est réservé à l'exécution des primitives du système d'exploitation.
  - Dans ce mode le processeur peut exécuter toutes les instructions.
- L'accès aux différentes ressources de la machine n'est autorisé qu'aux processus s'exécutant en mode protégé, le système d'exploitation protège ses ressources et contraint les processus utilisateurs à faire appel à lui pour accéder aux ressources de la machine.

Les appels système

Les appels système sont l'interface proposée par le système d'exploitation pour accéder aux différentes ressources de la machine.

- Pour chaque composant de l'ordinateur le système d'exploitation propose des appels système. Ils sont généralement classés en quatre catégories:
  - Gestion des processus,
  - Gestion des fichiers,
  - Communication et stockage d'informations,
  - Gestion des périphériques.

Exécution d'un appel système

- Lorsqu'un programme effectue un appel système, son exécution en mode utilisateur est interrompue et le système prend le contrôle en mode superviseur.
- Un appel système provoque une interruption logicielle et il suffit alors de programmer la machine pour que la routine correspondant à l'interruption fasse partie du système d'exploitation.
- Un appel système est exécuté en mode noyau même si le programme ayant demandé son exécution est exécuté en mode utilisateur.

IV. Les différentes classes de systèmes d'exploitation

- **Selon les services rendus**
  - ***mono/multi-taches***

    un système multi-tâches peut exécuter plusieurs processus simultanément. C'est le cas d'UNIX, Windows XP,…
  - **mono/multi-utilisateurs**

    un système Multi-utilisateurs peut gérer plusieurs utilisateurs utilisant simultanément les mêmes ressources matérielles. C'est le cas d'UNIX, ***windows XP***

- **Selon leur architecture**
  - **Systèmes centralisés**

    L'ensemble du système est entièrement présent sur la machine considérée. Le système ne gère que les ressources de la machine sur laquelle il est présent.
  - **Systèmes répartis (distributed systems)**

    Les différentes abstractions du système sont réparties sur un ensemble de machines. Le système d'exploitation réparti apparaît aux yeux de ses utilisateurs comme une machine virtuelle monoprocesseur même lorsque cela n'est pas le cas.

Chapitre 1 Ordonnancement de processus

I. Introduction

- Conceptuellement, chaque processus a son processeur propre virtuel. En réalité, le vrai processeur commute entre plusieurs processus sous la direction d'un ordonnanceur.
- L'Ordonnanceur (*scheduler*) est la partie du noyau qui s'occupe de la sélection des processus qui vont être exécutés
- L'ordonnanceur ne prend en charge que les processus insérés dans la fille d'attente des processus prêts suivants une priorité donnée.

Quand faut-il ordonnancer ?

- A la création d'un processus
- A la fin d'un processus
- Lors du blocage d'un processus
- Régulièrement, par exemple à chaque interruption d’horloge

Objectifs de l'ordonnancement

- S’assurer que chaque processus en attente d’exécution reçoit sa part de temps processeur.
- Minimiser le temps de réponse : l’utilisateur devant sa machine ne doit pas trop attendre
- Le processeur doit être utilisé à 100%
- Prendre en compte des priorités.
- Il faut exploiter au maximum le système

II. Environnements d'ordonnancement

On distingue 3 types d'environnements :

- Environnement de traitements par lots dans lesquels Il n'y a pas d'utilisateur en attente.
- Environnement interactif dans lequel un utilisateur interagit avec le système. il faut empêcher un processus de monopoliser le processeur.
- Environnement temps réel dans lequel la contrainte de temps est très importante. Ainsi les tâches doivent pouvoir s'exécuter quasi immédiatement, elles ne peuvent pas se permettre d'avoir du retard.

Catégories d'algorithmes d'ordonnancement

Dans chacun des environnements, on peut distinguer deux catégories d'algorithmes d'ordonnancement

- **Algorithme d'Ordonnancement sans réquisition (non préemptif)**: sélectionne un processus qui continue à s'exécuter jusqu'à la fin (soit il termine ou il se bloque (passe à l'état bloqué) sur une E/S ou en attente d'un autre processus). Cet algorithme est inefficace: exemple un processus qui exécute une boucle infinie.
- **Algorithme d'Ordonnancement avec réquisition (préemptif) :**
  - A chaque signal d'horloge, le système décide si le processus courant doit poursuivre son exécution ou il doit être interrompu pour laisser la place à un autre processus.
  - S'il décide de suspendre l'exécution d'un processus au profil d'un autre, il doit sauvegarder l'état des registres du processeur avant de charger dans les registres les données du processus à lancé. Cette sauvegarde est nécessaire pour pouvoir poursuivre ultérieurement l'exécution du processus suspendu.

2.1 L'ordonnancement sur les systèmes de traitement par lots

- **L'algorithme du premier arrivé, premier servi (FIFO, ou FCFS)**
  - La fille des processus prêts est formée selon leur ordre d'arrivée
  - L'ordonnanceur sélectionne le premier processus de la fille des processus prêts
  - Le choix d'un nouveau processus ne se fait que sur blocage ou terminaison du processus courant.
  - C'est un algorithme simple à comprendre et à programmer
  - **Inconvénients**: par exemple, certaines tâches rapides devront attendre longuement la fin des tâches précédentes.
- **L'algorithme le plus court temps d'exécution d'abord(SJF)**
  - La fille des processus prêts est formée selon leur temps d'exécution
  - C'est un algorithme qui peut être *non préemptif ou préemptif*. Il oblige à connaître à l'avance les temps d'exécution des jobs (tâches).
  - Le principe est donc de donner la priorité aux tâches les plus courtes.

on appel TR_P le temps de réponse d'un processus P.

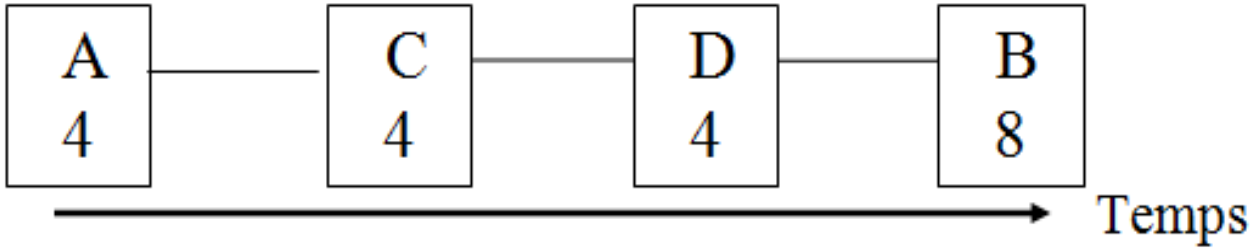
TR_P=instant de fin d'exécution du processus P – instant d'arrivé

On appel TRM le temps moyen de réponse de tous les processus

TRM=Somme des temps de réponse/Nombre de processeurs

Exemples avec SJF

1. Tous les processus sont disponible à l'instant t=0



TR_A=4-0, TR_C=8-0, TR_D=12-0 et TR_B=20-0

TRM=(TR_A+TR_C+TR_D+TR_B)/4

2. Les processus arrive à des instants différents

Processus	Temps d'exécution	Temps d'arrivage
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

Processus	Temps de séjour
A	3-0 = 3
B	9-1 = 8
E	10-7 = 3
D	12-6 = 6
C	16-4 = 12

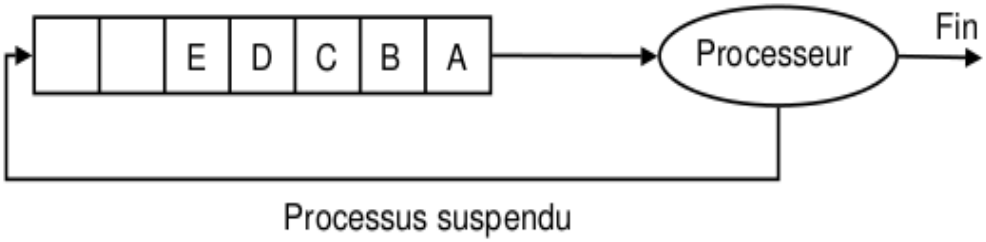
TR=Temps de séjour

TRM= (3+8+3+6+12)/5=6,4.

- **L'algorithme temps d’exécution restant le plus court d’abord(SRTF)**
  - C'est un algorithme préemptif dont l’objectif est de choisir la tâche dont le temps d'exécution restant est le plus court parmi les autres. Les processus prêts sont insérés dans la la fille selon leur temps d’exécution restant
  - Le temps d'exécution restant doit être connu.
  - Si une nouvelle tâche est créée et que son temps d'exécution total est plus court que le temps restant de la tâche en cours, la tâche en cours est suspendue pour laisser la place au nouveau job (préemption).
  - L'avantage principal de ce principe est qu'il favorise les tâches courtes, ce qui est important sur des systèmes de traitements par lots.

2.2 L'ordonnancement sur les systèmes interactifs

- **L'algorithme du tourniquet (round robin, circulaire)**
  - Cet algorithme est ancien et très simple. C'est également le plus équitable et un des plus utilisés.
  - Après leur insertion dans la fille d'attente des processus prêts, tous les processus ont la même priorité
  - Son principe est d'assigner un intervalle de temps d'exécution unique (un quantum d'exécution) à chaque processus.



  - Cet algorithme alloue le processeur au processus en tête de la file pendant un quantum de temps.
  - Si le processus se bloque ou se termine avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus (celui en tête de file).

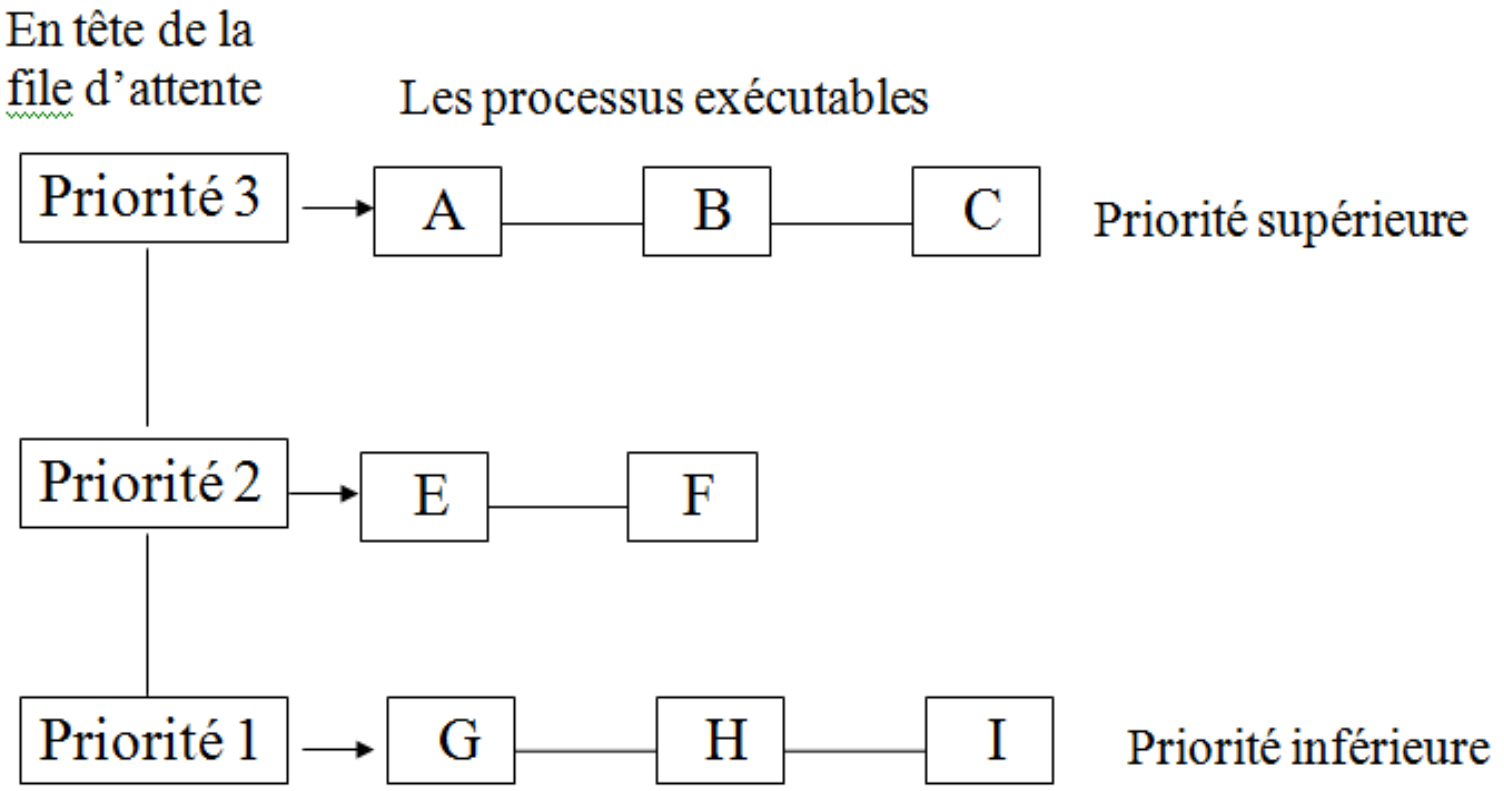
- Si le processus ne se termine pas au bout de son quantum, son exécution est suspendue. Le processeur est alloué à un autre processus (celui en tête de file) et ainsi de suite. Le processus suspendu est inséré en queue de file.
- Les processus qui arrivent ou qui passent de l'état bloqué à l'état prêt sont insérés en queue de file.
- La problématique centrale de cet algorithme est donc la durée de l'intervalle de temps d'exécution (quantum).
- Chaque changement de processus (changement du contexte d'exécution) nécessite un certain nombre de tâches administratives coûteuses en terme de temps (enregistrement et chargement des registres, vidage et chargement du cache mémoire …).

Exemple:

- Si la commutation nécessite 1ms et le quantum 4ms
  - 20% du temps processeurs perdu dans la commutation.
- Pour améliorer l'efficacité, on peu augmenter la durée du quantum. Par exemple, si le quantum dure 99 ms le temps perdu n'est d'un 1%. Dans ce cas se pose le problème de temps de réponse.
- Par exemple, si 10 processus (10 utilisateurs) en attente, le dernier utilisateur doit attendre 1s pour pouvoir exécuter une commande.
- Problème=réglage du quantum
  - quantum trop petit: le processeur passe son temps à commuter
  - quantum trop grand: augmentation du temps de réponse des processus.
- Il faut donc trouver la durée de quantum idéale en fonction de ce temps nécessaire au changement de processus et du type de tâches que souhaite réaliser l'utilisateur. Un compromis raisonnable semble être un quantum d'environ 50 ms.

2.3 Ordonnancement avec niveau de priorité

- L'algorithme *round robin* permet une répartition équitable du processeur. Cependant il n'est pas intéressant si certains processus sont plus importants ou urgents que d'autres.
- L'ordonnancement avec priorité donne à chaque processus un niveau de priorité. Le processus à exécuter est celui qui détienne la priorité la plus élevé (le plus prioritaire).
- En pratique, les processus sont regroupés en catégories de priorités(files de processus). Les processus ayant la même priorité sont gérer par un algorithme d'ordonnancement tel que tourniquet, FIFO,….
- Les processus d'une catégorie de niveau de priorité *k* ne peuvent être exécutés que s'il n'existe aucun processus de niveau *j* avec $j>k$ qui est prêt.



Le problème posé dans ce ca est que les processus prioritaires peuvent s'exécuter indéfiniment. Pour éviter ce phénomène, on peut baisser régulièrement le niveau de priorité du processus en cours d'exécution. Ainsi, lorsque sa priorité n'est plus la plus forte, il se repositionne dans la queue de la file ayant la même priorité que la sienne.

Le tableau ci-dessous résume les caractéristique de chacun des algorithmes

Nom	Définition	Non préemptif	Préemptif
First Come First Serve (**FCFS**)	Selon l'ordre d'arrivée	X	
Shortest Job First (**SJF**)	Temps de traitement le plus court d'abord	X	X
Shortest Remaining Time First (**SRTF**)	Temps de traitement restant le plus court d'abord		X
Round-Robin (**RR**)	Accès au processeur pour une durée limitée. Equité de service		X (partiel)
Ordonnancement à priorités	Selon les priorités affectées aux processus	X	X

Chapitre2 Synchronisation des processus

I. Introduction

1.1. Exposé du problème

- Les processus en cours d'exécution sont généralement :
  - Indépendants et Asynchrones
    - Leur fonctionnement ne dépend pas a priori du travail réalisé par les autres processus
    - Ils peuvent a priori progresser à leur rythme sans se soucier les uns des autres : Ils pourraient s'exécuter en parallèle
- Pourtant, ces processus peuvent être en concurrence pour l'utilisation de ressources
- & Avoir besoin de se synchroniser et communiquer : dans ce cas, ils seront au contraire dépendants les uns des autres

- Des ressources peuvent être accessibles par plusieurs processus à la fois
- Des ressources ne peuvent être accessibles que par un seul processus à la fois: on parle de ressource critique
- Lorsque deux ou plusieurs processus sont en compétition pour le partage d'une ressource critique, ont dit qu'ils sont des processus concurrents.
- Les instructions du processus qui permettent d'accéder à la ressource critique en lecture ou en écriture forme ce qu'on appelle la section critique.

1.2. Problème de partage des ressources

Exemple 1

Soient p1 et p2 deux processus concurrents. Chaque processus veut décrémenter une variable V dans l'espace d'adressage qui lui est propre. Supposons la valeur initiale de chaque variable est v0. Chaque processus exécute les instructions suivantes (programmes A1 et A2):

p1

A1

/* x dans l'espace de p1*/

```
1. x=V-1;
2. V=x;
```

p2

A2

/* x dans l'espace de p2*/

```
1. x=V-1;
2. V=x;
```

alors la valeur finale de la variable V pour chaque processus est égale à v0-1.

Exemple2

Soient p1 et p2 deux processus concurrents qui veulent décrémenter une variable partagée V. Supposons que la valeur initiale de V est v0 et on veut après la décrémentation obtenir v0-2 comme valeur de V. Supposons que chaque processus exécute les instructions suivantes:

p1	p2
A1	**A2**
/* x dans l'espace de p1 */	**/* x dans l'espace de p2 */**
1. x=V-1;	**1. x=V-1;**
2. V=x;	**2. V=x;**

En multiprogrammation plusieurs scénarios d'exécution de P1 et P2 sont possibles.

Premier scénario:

Supposons que c'est le p1 qui commence l'exécution et que l'ordonnanceur ne commute les tâches est alloue le processeur à p2 qu'après la fin d'exécution de l'instruction (2. V=x).

- Dans ce cas la valeur de V après la fin du p1 est égale à v0-1.
- Après la fin d'exécution de p2, la valeur de V devienne v0-2.
- Dans ce cas on a obtenu la bonne valeur

Deuxième scénario:

Supposons que c'est le p1 qui commence l'exécution.

- p1 lit la valeur initiale v0 puis effectue l'opération x=V-1.
- Avant que p1 écrit la nouvelle valeur dans la variable V (instruction V=x), l'ordonnanceur commute les tâches et alloue le processeur à p2
- p2 lit la valeur initiale de V (soit v0) et effectue les opérations x=V-1; et V=x. La nouvelle valeur de V devienne v0-1.
- L'ordonnanceur réactive le premier processus p1 qui continue son exécution au point où il était arrêté, c'est-à-dire effectue l'opération V=x, avec la valeur de x qui est v0-1.

Les opérations des processus sont effectuées dans l'ordre suivant:

p1.1; p2.1; p2.2; p1.2.

Donc après l'exécution des instruction dans cet ordre, la valeur finale de V est égale à v0-1. (au lieu de v0-2 ce qui était attendu).

Problème

Accès concurrent à une variable partagée. *Pour l'éviter,* on doit synchroniser les tâches: *s'assurer* que l'ensemble des opérations sur cette variable (accès + mise à jour) est exécuté de manière indivisible (*atomique*). Par exemple on impose que p2 ne doit commencer son travail qu'après la fin d'exécution de p1.

- Si A1 et A2 sont atomiques, le résultat de l'exécution de A1 et A2 ne peut être que celui de A1 suivie de A2 ou de A2 suivi de A1, à l'exclusion de tout autre
- On dit aussi que les séquences d'actions <1; 2 > (dans p1 et p2) est une *section critique*
- Une section critique est exécutée en exclusion mutuelle (un seul processus au plus peut être dans sa section critique à un instant donné).

II. Solutions de la section critique

- Considérons n processus qui partage la même ressources. Le programme sous jacent à chaque processus a la structure suivante:

Répéter

Section restante

Section critique

Jusqu' à faux

- Pendant l'exécution parallèle de p1,p2,… et pn, il est possible que plusieurs processus soient à un instant donné entrain d'exécuter une instruction de leur section critique.

- 2 grandes classes de solutions envisageables :
  - Solution avec attente active : dans ce cas on intègre aux codes une boucle qui fait rien et empêche le processus d'avancer tant qu'une condition n'est pas vérifiée
  - Solutions avec blocage: dans ce cas le processus fait un appel implicite pour accéder à une ressource partagée. Si cette ressource est disponible il le prend sinon il se bloque en attendant sa libération.
- Les 2 classes exigent l'ajout de code de protection autour de la section critique: *Section d'entrée* et *Section de sortie* ce qui donne:

Répéter

Section restante

Section d'entrée

Section critique

Section de sortie

jusqu'à faux

- Le processeur est un cas particulier de ressource partagée, car la demande de son allocation n'est pas faite par les processus. Mais il est allouer par l'ordonnanceur suivant un algorithme d'ordonnancement sans leur intervention.

Propriétés attendues d'une solution

- Exclusion mutuelle : A tout instant, un processus au plus exécute des instructions de sa section critique
- Absence de blocage (permanent): Si plusieurs processus attendent pour entrer en SC, et si aucun processus n'est déjà en SC, alors un des processus qui attendent doit pouvoir entrer en SC au bout d'un temps fini
- Condition de progression : Un processus qui se trouve hors de sa SC et hors du section d'entrée ne doit pas empêcher un autre processus d'entrer dans sa SC: un processus ne doit pas ralentir un autre
- Equité (Absence de famine): Un processus qui est bloqué à l'entrée de la section critique n'attendra pas indéfiniment son tour. Pour un processus qui veut entrer en SC, il existe une borne supérieure au nombre de fois où d'autres processus exécuteront leur SC avant lui (La valeur de la borne permet de mesurer à quel point une solution est équitable)

2.1 Solutions avec attente active

Principe :

Un processus désirant entrer en SC attend d'une façon active qu'une condition soit vérifiée. Pendant l'attente active le processus occupe le processeur sans avancer dans son exécution.

Répéter

Section restante

Tant que (condition indique SC non libre) Faire rien

Fin tant que

<section de code critique>

Modifier condition pour refléter SC libre

Jusqu'à faux

Problème : Consommation inutile de temps CPU

2.1.1 Solutions logicielles: Algorithme1

tour une variable partagée initialisée à 1

p1	p2
Répéter	**Répéter**
Section restante1	**Section restante2**
Tant que tour=2 faire rien;	***Tant que tour=1 faire rien***
Fin tant que	***Fin tant que***
Section critique1	**Section critique2**
Tour=2	***Tour=1***
Jusqu'à faux	**Jusqu'à faux**

- Les processus ne peuvent pas entrer tous les deux en section critique
- Absence de blocage
- Mais Supposons que p1 est dans <section restante> et tour=1, p2 ne peut pas entrer dans sa section critique. Il doit attendre que p1 exécute sa section critique et met tour à 2 ce qui met en cause la condition de progression

Algorithme1: Application

tour une variable partagée initialisée à 1

x une variable locale à P1	***x une variable locale à P2***
P1	**P2**
Tant que tour=2 faire rien	***Tant que tour=1 faire rien;***
Fin tant que	***Fin tant que***
x=V-1	**x=V-1**
V=x	**V=x**
Tour=2	***Tour=1***

Algorithme2 : D1=faux; D2=faux

p1	p2
Répéter	Répéter
Section restante1	Section restante2
D1=vrai	D2=vrai
Tant que D2 faire rien	*Tant que D1 faire rien*
Fin tant que	*Fin tant que*
Section critique1	Section critique2
D1=faux	*D2=faux*
Jusqu'à faux	Jusqu'à faux

Inconvénient

Blocage si D1=D2=vrai

Algorithme3 [Peterson 1981]

On envisage alors de signaler d'abord que le processus demande l'accès à la S.C. en affectant un booléen, puis y entre effectivement s'il n'y a pas de conflit d'accès, c-à-d si l'autre processus n'a pas signalé aussi son intention d'entrer en S.C.

p1	p2
Répéter	Répéter
Section restante1	Section restante2
D1=vrai	*D2=vrai*
Tour=2	*Tour=1*
Tant que (D2 et tour=2) faire rien	*Tant que (D1 et tour=1) faire rien*
Fin tant que	*Fin tant que*
Section critique1	Section critique2
D1=faux	*D2=faux*
Jusqu'à faux	Jusqu'à faux

Assure les quatre propriétés attendues d'une solution à savoir *l'exclusion mutuelle, l'absence de blocage, progression d'exécution et l'équité*

2.1.2. Solution matériel

- De nombreux ordinateurs dispose d'une instruction

  Nommée **test and set** qui permet de Lire et écrire le contenu d'un mot mémoire d'une manière indivisible.

  Cette instruction a deux opérande:

  – Un registre a
  – Un mot mémoire b

```
Procedure TS(var a,b:entier)
        debut
                a←b
                b←1
        fin
```

Algorithme4

verrou une variable partagé initialisé à 0

p1	p2
Répéter	**Répéter**
Section restante1	**Section restante2**
TS(test1,verrou)	***TS(test2,verrou)***
Tant que test1=1 faire	***Tant que test2=1 faire***
TS(test1,verrou)	***TS(test2,verrou)***
Fin tant que	***Fin tant que***
Section critique1	**Section critique2**
Verrou=0	***Verrou=0***
Jusqu'à faux	**Jusqu'à faux**

Un processus qui trouve verrou=1 lorsqu'il veut entrer dans sa section critique effectue une attente active

2.2. Solutions avec blocage (attente passive)

Pour empêcher le processus qui ne peut pas entrer dans sa section critique d'effectuer une attente active, il faut le faire passer à l'état bloqué à l'aide des sémaphores et des moniteurs.

2.2.1. Les sémaphores [Dijkstra, 1965]

- Description

Sémaphore = structure de données (compteur + file d'attente de processus) + interface (opérations sur la structure de données)

Type semaphore=enregistrement

valeur:entier

liste_d'attente des processus

fin

P(s:semaphore)
debut
si s.valeur<=0 alors
        <ajouter le processus à s.liste_d'attente>
            <mettre le processus en état bloqué>
sinon
        s.valeur=s.valeur-1
Finsi
fin
V(s:semaphore)
début
si s.liste_d'attente non vide alors
<choisir et enlever un processus de s.liste_d'attente>
<faire passer à l'état prêt le processus choisi>
sinon
        s.valeur=s.valeur+1
finsi
fin

- Les opérations P et V sur un sémaphore sont supposées être exécutées de manière indivisible. Ceci signifie que, pendant qu'un processus exécute une opération P ou V sur un sémaphore S, aucun autre processus ne peut exécuter P ou V sur ce même sémaphore S.
- La solution au problème de la section critique

```
Init(s_mut,1)
Répeter
  Section restante
  P(s_mut)
  Section critique
  V(s_mut)
Jusqu'à faux
```

- un processus qui exécute l'opération P(S) et trouve S.valeur négative ou nulle effectue une attente passive puisqu'il est placé en état bloqué. Ce blocage se poursuit jusqu'à ce qu'un processus exécute V sur le même sémaphore et qu'il soit choisi.

Exemple3

Soit p1 et p2 deux processus qui partagent un tableau d'entier. Leur programmes sous-jacent sont les suivants:

p1	p2
Section restante1	**Section restante2**
Remplir le tableau T	**Afficher le tableau T**
Suite1	**Suite2**

Si on lance l'exécution parallèle de p1 et p2, on ne sait pas lequel des processus va commencer. Par conséquent, il se peut que p2 affiche le tableau avant qu'il soit rempli par p1.

Si on veut que la partie *remplir le tableau T* du processus p1 soit exécutée avant la partie *afficher le tableau T* de p2, il faut synchroniser ces deux processus. Une solution parmi d'autres : utilisation des sémaphores

Modification des programmes sous-jacent à p1 et p2.

- soit *mut* un sémaphore ***initialisé à 0***

p1	p2
Section restante1	**Section restante2**
Remplir le tableau T	***P(mut)***
V(mut)	**Afficher la tableau T**
Suite1	**Suite2**

- P(mut) permet de bloquer p2 tant que p1 n'a pas encore rempli le tableau
- Après le remplissage du tableau, p1 exécute V(mut) et débloque le processus p2 afin d'afficher le tableau T
- Les sémaphores peuvent servir à réguler d'autres interactions de nature « Synchronisation » entre entités.
  - permettre à un nombre borné (éventuellement plus grand que 1) de processus d'entrer en section critique
  - Attendre qu'un nombre minimal de processus soient bloquées avant que l'un d'eux puisse continuer (rendez-vous)

2.2.2. Les moniteurs

- Module comprenant
  - Des données
  - Des procédures d'accès (P1,..,Pn)
  - Une procédure d'initialisation
  - Des conditions
- Les procédures sont exécutées en exclusion mutuelle
- Une condition est une structure qui permet de bloquer un processus « à l'intérieur » du moniteur
- Une condition ressemble aux sémaphores : manipulée au travers d'une interface:
  - Opération WAIT() indivisible
  - Opération SIGNAL() indivisible

Soit x de type condition

- x.WAIT()
  - mettre le processus appelant dans x.File
  - mettre ce processus dans l'état bloqué
- x.SIGNAL()

  Si (x.File non vide) Alors
  - retirer de x.File un processus
  - mettre ce processus dans l'état prêt
- Contrairement à V(), SIGNAL() ne laisse pas de trace. On peut en faire plus que nécessaire, mais attention aux signaux de réveil « perdus », car pas mémorisés …
- Wait() bloque toujours le processus appelant par contre P() n'est bloquante que si la valeur du sémaphore est négative ou nulle

Schéma de moniteur

type <monitor-name> =moniteur
début
<variables partagées + déclarations des conditions >
procedure P1 (…) début
. . .
Fin
procedure P2 (…) début
. . .
Fin
. . .
procedure Pn (…) début
. . .
fin
/*initialisation*/
Début
. . .
Fin
Fin /*du moniteur*/

Fonctionnement d'un moniteur

- Au maximum un seul processus actif dans le moniteur
- Si un processus exécute wait() et se bloque sur une variable de condition à l'intérieur du moniteur, il doit laisser libre l'accès au moniteur
- Si un processus à l'intérieur du moniteur exécute signal() et réveille un autre bloqué sur une variable de condition, il doit :
  - Pour Hoare : être bloqué jusqu'à ce que le processus réveillé quitte le moniteur
  - Pour Brinch-Hansen : quitter immédiatement le moniteur

La synchronisation du processus qui remplit un tableau avec celui qui l'affiche en utilisant les moniteurs se fait de la façon suivante:

```
Type Synchro = moniteur
début
Var fait:booléen, tab_rempli: condition
Procedure fin_ecriture()
début
Fait=vrai
Rempli.signal()
Fin
```

```
procedure début_lecture()
debut
Si non (fait) alors
rempli.wait();
Fin si
Fin
/*initialisation*/
début
Fait = faux
Fin
Fin //fin moniteur
```

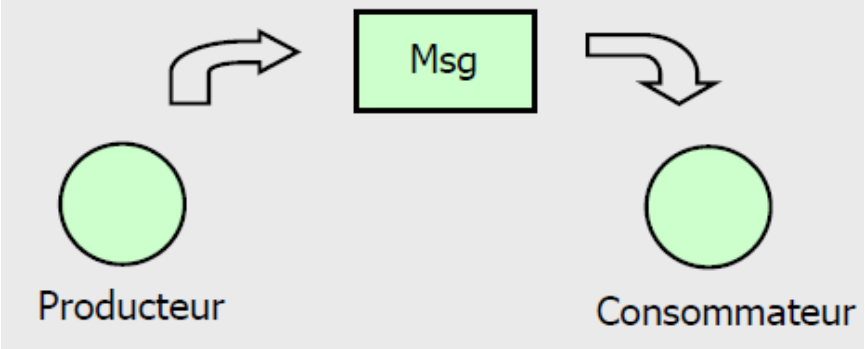
```
S:variable partagé de type Synchro
Processus P1
Section restante1
Remplir_tableau T
S.fin_ecriture()
Suite1
```

```
Processus P2
Section restante2
S.debut_lecture()
Afficher_tableau T
Suite2
```

2.2.3 Problème classique de synchronisation

2.2.3.1 Producteur / Consommateur avec les sémaphores

Il s'agit de synchroniser deux processus : un processus producteur dépose un message dans un tampon et un autre appelé consommateur le retire.



- **Conditions de dépôt et de retrait**
  - **Le producteur ne peut pas déposer le message si le tampon est plein, il doit attendre que le consommateur le vide.**
  - **Le consommateur ne peut pas prendre le message si le tampon est vide, il doit attendre son remplissage par le producteur**

- 2 Sémaphores pour la synchronisation conditionnelle:
  - Si le tampon n’est pas vide, le processus type producteur attend
  - Si le tampon est vide , processus type consommateur attend

Données partagées :

- // condition de production
- Semaphore vide Init(vide,1)
- // condition de consommation
- Semaphore plein Init(plein,0)

Producteur

Répéter

Produire_message()

P(vide)

tampon=message

V(plein)

Juqu’à faux

Consommateur

Répéter

P(plein)

message=tampon

V(vide)

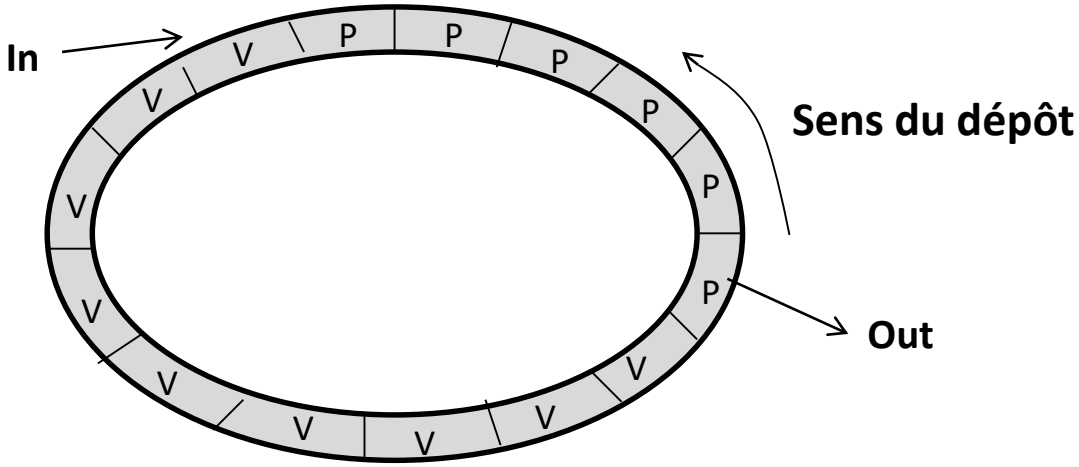
utiliser_message()

Juqu’à faux

2.2.3.2 Producteurs / Consommateurs avec les sémaphores

- **Gestion d'un buffer de N cases (N>1)**

P:plein
V:vide



- tampon (circulaire) avec nombre borné de places
- 2 Sémaphores pour la synchronisation conditionnelle :
  - Si pas de place disponible, processus type producteur attend
  - Si pas de place remplie, processus type consommateur attend
- Deux sémaphores pour gérer l'exclusion mutuelle entre producteurs ou entre consommateurs pour ne pas accéder à la même case

Données partagées

Entier N : la taille du tampon*; In :*indice utilisé par les producteurs*,* Out:indice utilisé par les consommateurs*. In* et *Out* sont initialisés à *0*

Sémaphore plein initialisé à 0*,* compte le nombre d'emplacements occupés.

Sémaphore vide initialisé à N, compte le nombre d'emplacements libres

Sémaphore Mutex_In initialisé à 1 assure l'accès exclusif au tampon entre les producteurs

Sémaphore Mutex_Out initialisé à 1 assure l'accès exclusif au tampon entre les consommateurs

Producteur
Répéter
Produire_objet()
P(vide)
P(mutex_In)
tampon[In]=objet; In=(In+1) mod N
V(mutex_In)
V(plein)
Juqu'à faux

Consommateur
Répéter
P(plein)
P(mutex_Out)
Objet=tampon[out];
Out=(Out+1) mod N
V(mutex_Out)
V(vide)
Juqu'à faux

2.2.3.3 Producteurs / Consommateurs avec les moniteurs

```
Type prodcon = moniteur
debut
Var  nonvide, nonplein : condition
C,in,out:entier
procedure produire(m:message)
debut
Si C=N alors
nonplein.wait();
Fin si
tampon[In] = m
In = In + 1 mod N;
C++;
nonvide.signal();
Fin
```

```
Procedure consommer(m:message)
debut
Si C=0 alors
nonvide.wait();
Fin si
m=tampon[Out]
Out=Out+1 mod N
C--;
nonplein.signal();
Fin
/*initialisation*/
début
C=0;In=0;Out=0
Fin
Fin //fin moniteur
```

P: variable partagé de type Prodcon

Producteur
Répeter
Constuire_message m
P.produire(m)
Jusqu'à faux

consommateur
Répeter
P.consommer(m)
Utiliser_message m
Jusqu'à faux

Chapire 3

Interblocage

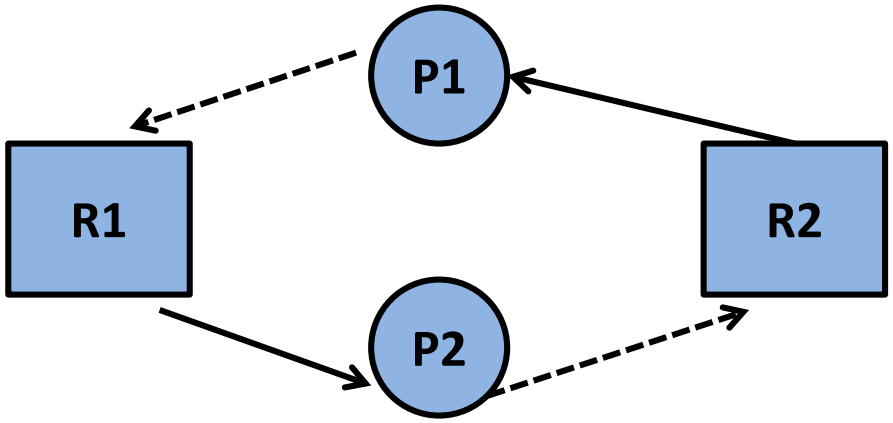
- Un système informatique possède un nombre fini de ressources qui doivent être distribuées parmi un certain nombre de processus concurrents.
- Les ressources sont groupées en plusieurs types,
- Chaque type peut exister en plusieurs instances identiques. L'espace mémoire, le processeur, les périphériques sont des exemples de types de ressources.
- Par exemple, si un système a 2 processeurs, on dira que le type de ressource processeur possède 2 instances, et si le système est doté de 5 imprimantes, on dira que le type de ressource imprimante possède 5 instances.

- Dans des conditions normales de fonctionnement, un processus ne peut utiliser une ressource qu'en suivant la séquence suivante :

  Requête – Utilisation - Libération

- *La requête :* le processus fait une demande pour utiliser la ressource. Si cette demande ne peut pas être satisfaite immédiatement, parce que la ressource n'est pas disponible, le processus demandeur se met en état attente(bloqué) jusqu'à ce que la ressource devienne libre.
- *Utilisation :* Le processus peut exploiter la ressource.
- *Libération :* Le processus libère la ressource qui devient disponible pour les autres processus éventuellement en attente.

- Un ensemble de processus est dans une situation d'interblocage si chaque processus de l'ensemble attend un événement qui ne peut être produit que par un autre processus de l'ensemble.
- *Exemple :* Un système possède une instance unique de chacun des deux types de ressources R1 et R2 à accès exclusif. Un processus P1 détient l'instance de la ressource R1 et un autre processus P2 détient l'instance de la ressource R2. Pour suivre son exécution, P1 a besoin de l'instance de la ressource R2, et inversement P2 a besoin de l'instance de la ressource R1. Une telle situation est une situation d'interblocage.



L'accès exclusif aux deux ressources R1 et R2 à l'aide des sémaphore de Dijkstra peut être fait la façon suivante:

S1,S2:sémaphore

Init(S1,1);init(S2,1)

Processus P1	Processus P2
. . .	. . .
P(S1)	P(S2)
détenir R1	détenir R2
P(S2)	P(S1)
détenir R2	détenir R1
V(S1)	V(S2)
V(S2)	V(S1)
. . .	. . .

Supposons qu'après l'exécution de P(S1) par le processus P1, le système alloue le processeur à P2. Ce dernier commence son exécution, mais après l'exécution de P(S2), l'ordonnanceur l'interrompe et alloue le processeur à P1. quand P1 exécute P(S2) il se bloque en attendant que P2 exécute V(S2). Quand l'ordonnanceur alloue de nouveau le processeur à P2 il se bloque à son tour sur P(S1)en attendant que P1 exécute V(S1) ce qui conduit un interblocage

conditions d’interblocage

Une situation d’interblocage peut survenir si les quatre conditions suivantes se produisent simultanément (Habermann) :

1. ***Accès exclusif*** : Les ressources ne peuvent être exploitées que par un seul processus à la fois.
2. ***Attente et occupation*** *:* Les processus qui demandent de nouvelles ressources gardent celles qu'ils ont déjà acquises et attendent la satisfaction de leur demande
3. ***Pas de réquisition*** *:* Les ressources déjà allouées ne peuvent pas être réquisitionnées.
4. ***Attente circulaire*** *:* Les processus en attente des ressources déjà allouées forment une chaîne circulaire d'attente.

Graphe d'allocation des ressources

- On peut décrire l'état d'allocation des ressources d'un système en utilisant un graphe. Ce graphe est composé de N nœuds et de A arcs.
- L'ensemble des nœuds est partitionné en deux types :
  - P={P1, P2, …, Pm} : l'ensemble de tous les processus
  - R={R1, R2, …, Rn} l'ensemble de tous les types de ressources du système
- Un arc allant du processus Pi vers un type de ressource Rj est noté Pi→Rj ; il signifie que le processus Pi a demandé une instance du type de ressource Rj. Un arc du type de ressourcé Rj vers un processus Pi est noté Rj→Pi ; il signifie qu'une instance du type de ressource Rj a été alloué au processus Pi.

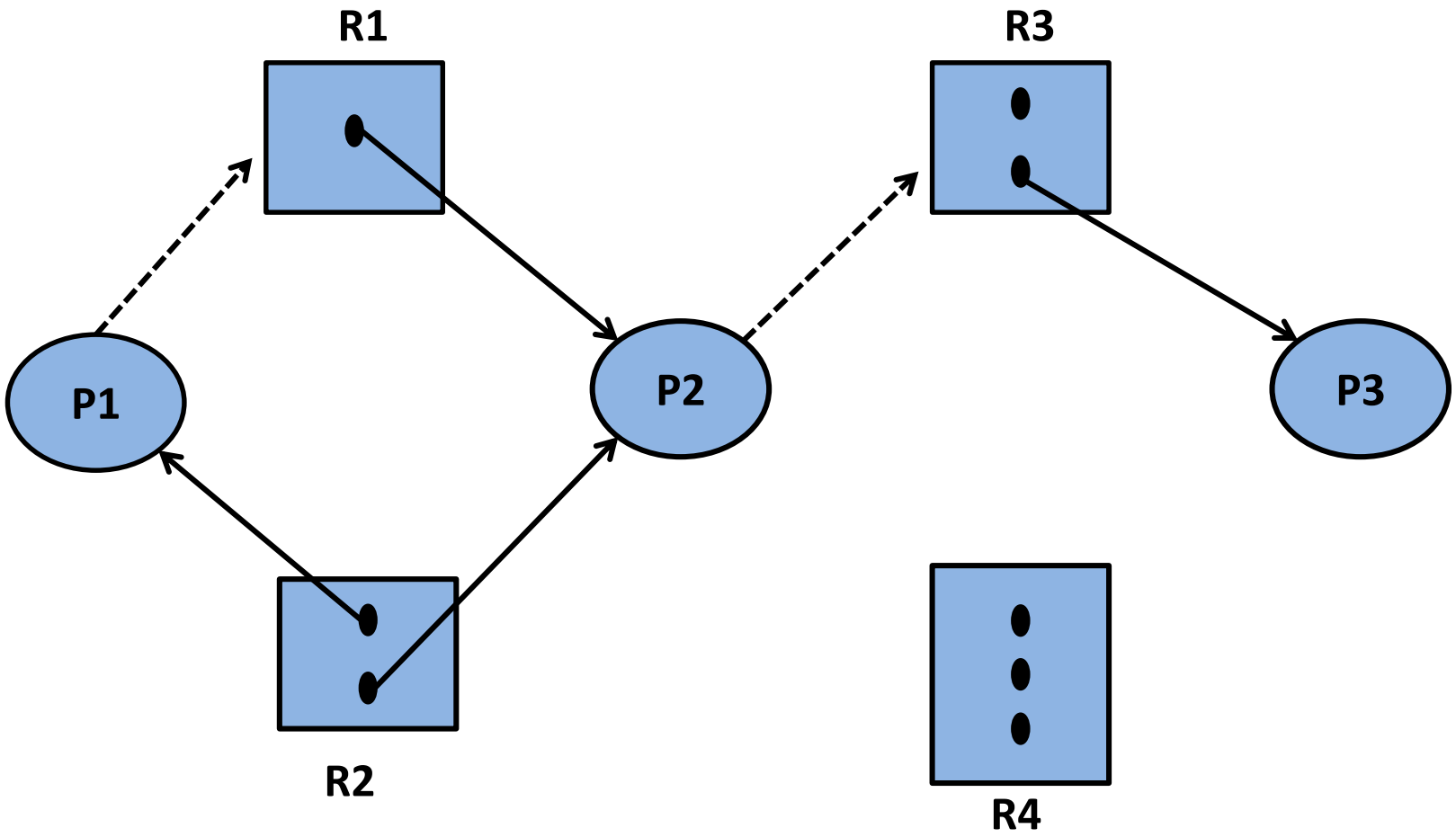
- Graphiquement, on représente chaque processus Pi par un cercle et chaque type de ressource Rj comme un rectangle.
- Puisque chaque type de ressource Rj peut posséder plus d'une instance, on représente chaque instance comme un point dans le rectangle.
- Un arc de requête désigne seulement le rectangle Rj, tandis que l'arc d'affectation doit aussi désigner un des points dans le rectangle.
- Quand un processus Pi demande une instance du type de ressource Rj, un arc de requête est inséré dans le graphe d'allocation des ressources.
- Quand cette requête peut être satisfaite, l'arc de requête est instantanément transformé en un arc d'affectation.
- Quand plus tard, le processus libère la ressource l'arc d'affection est supprimé.

Exemple : L'état d'allocation d'un système est décrit par les ensembles suivants :

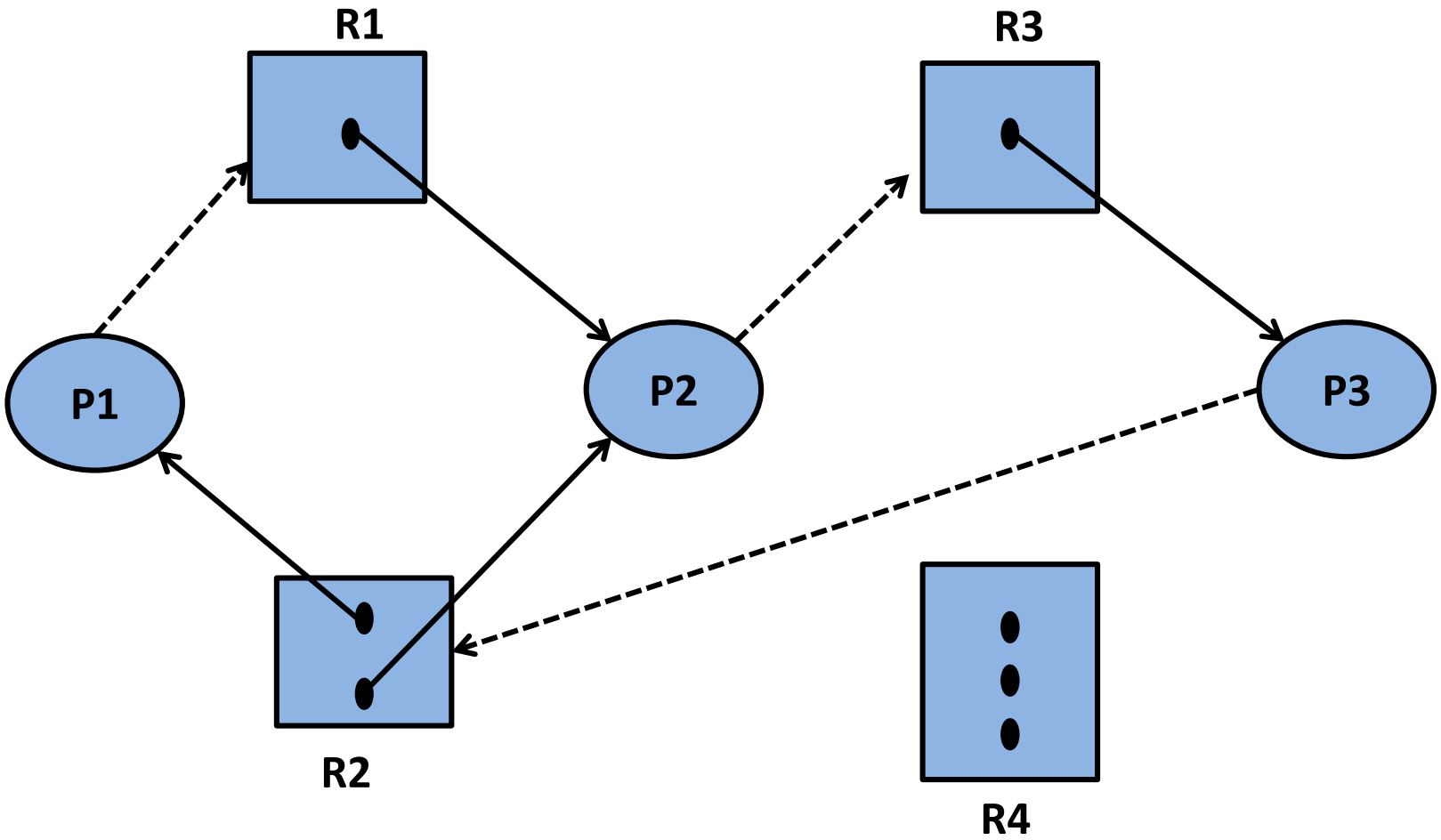
- Ensemble des processus P={P1, P2, P3}
- Ensemble des ressources R={R1, R2, R3, R4}
- Ensemble des arcs A={P1→R1, P2→R3, R1→P2, R2→P2, R2→P1, R3→P3}
- Le nombre d'instances par ressources est donné par ce tableau :

Type de ressources	Nombre d'instances
R1	**1**
R2	**2**
R3	**2**
R4	**3**

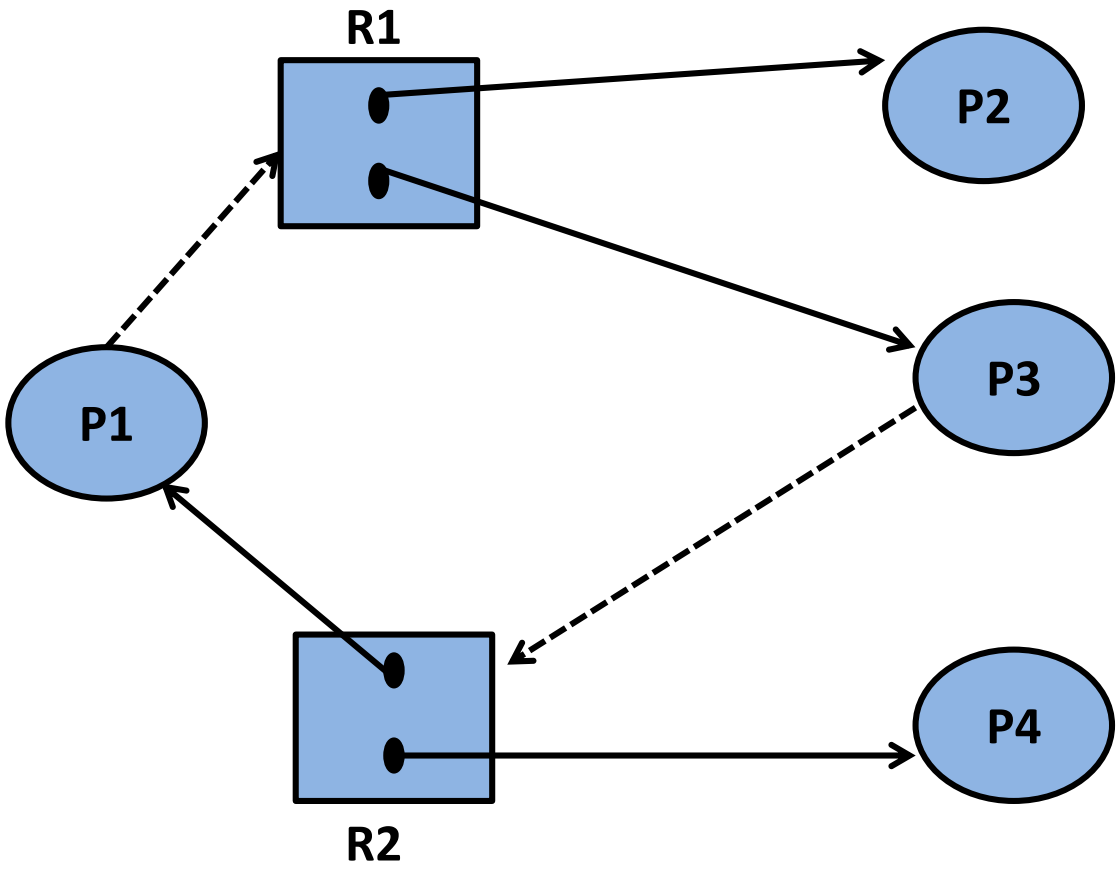
- Voici le graphe d'allocation des ressources associé à ce système :



Si le graphe d'allocation contient un *circuit, alors il peut exister une situation d'interblocage. C'est le cas du Graphe d'allocation des ressources suivant:*



- La condition d’existence de circuit est nécessaire mais pas suffisante. Par exemple ce graphe contient un circuit mais sans interblocage



Méthodes de traitement des interblocages

- Ignorer les interblocages (politique de l'autruche)

  Exemple : le système Unix
- Détection des interblocages
  - Laisser se produire les interblocages , ensuite tenter de les détecter et de les supprimer.
  - Si chaque ressource existe en un seul exemplaire, alors un interblocage existe si le graphe d'allocation des ressources contient un cycle
  - L'existence d'un cycle dans le graphe d'allocation n'est pas une CNS pour détecter les interblocages si une ressource peut exister en plusieurs exemplaires
- Eviter dynamiquement les interblocages en allouant les ressources avec précaution
- Les prévenir en empêchant l'apparition des 4 conditions de leur existence

Chapitre 4: Gestion de la mémoire

I-Introduction

- L'exécution d'un processus demandant que le code du programme et les données utilisées soient présents en mémoire, cette dernière est une ressource essentielle du système d'exploitation.
- La gestion de la mémoire est confiée à un allocateur qui l'attribuera au(x) processus demandeur(s). L'objet de ce chapitre est donc l'étude de l'allocation de la ressource mémoire au sein d'un système d'exploitation
- Le terme "mémoire" fait surtout référence à la mémoire principale, c'est à dire à la RAM, mais la gestion de celle-ci demande la contribution de la mémoire auxiliaire (mémoire de masse, spacieuse mais lente) et à la mémoire cache (rapide mais de taille restreinte).

Rôle du gestionnaire mémoire:

- Connaître les parties libres et les parties en cours d'utilisation de la mémoire physique.
- Allouer de la mémoire au processus qui en ont besoin en essayant d'éviter le gaspillage
- Récupérer la mémoire libérée après terminaison d'un processus
- Gérer le va et vient (ou *swapping*) entre la mémoire principale et le disque lorsque la mémoire principale disponible est trop petite pour contenir tous les processus

On distingue deux modes de programmation

La monoprogrammation (cas simple)

- Un programme peut se trouver en mémoire
- Pour exécuter un second programme, on doit d'abord décharger le 1er programme de la mémoire puis charger le second
- Ce mode n'est plus utilisé aujourd'hui

La Multiprogrammation

- Plusieurs programmes peuvent cohabiter en même temps en mémoire
- Mécanisme de protection qui empêche deux programmes d'interférer entre eux.
- Comment organiser la mémoire le plus efficacement possible ?
- La réponse à cette question fait l'objet de la suite de ce chapitre

Les stratégies d'allocation de la mémoire

- En multiprogrammation, on trouve essentiellement deux modes d'allocation de la mémoire centrale : le mode ***contigu*** et celui ***non contigu***.
- Selon le mode d'allocation qui est appliqué, lorsqu'un programme est chargé en mémoire centrale à partir du disque, le programme sera placé dans une seule zone (allocation contiguë) ou réparti entre plusieurs zones (allocation non contiguë)..
- Le mode contigu alloue à un programme une zone de mémoire de taille limitée, or tout programme est amené à augmenter de taille lors de son exécution. En effet, des résultats sont calculés et des variables peuvent être créées dynamiquement.
- On distingue alors des systèmes qui utilisent des zones de taille fixe et d'autres qui permettent au programme de s'étendre sur l'espace avoisinant si celui-ci est libre.
- Le mode d'allocation contiguë n'est plus appliqué de nos jours, nous n'en donnerons qu'un rapide aperçu en guise d'historique.

II-Allocation contiguë en mémoire centrale

2.1 Les partitions de taille fixe

- A l'initialisation du système, la mémoire est divisée en *n* partitions de taille fixe, pas nécessairement égales (méthode MFT [*Multiprogramming with a Fixed number of Tasks*] apparue avec les IBM 360). Il existe deux méthodes d'affectation:

1. On crée une file d'attente par partition . Chaque nouveau processus est placé dans la file d'attente de la plus petite partition pouvant le contenir.

Inconvénients :

- on perd en général de la place au sein de chaque partition.
- il peut y avoir des partitions inutilisées (leur file d'attente est vide).

2. On crée une seule file d'attente globale. Il existe deux stratégies:
   - dès qu'une partition se libère, on lui affecte la première tâche de la file qui peut y tenir.

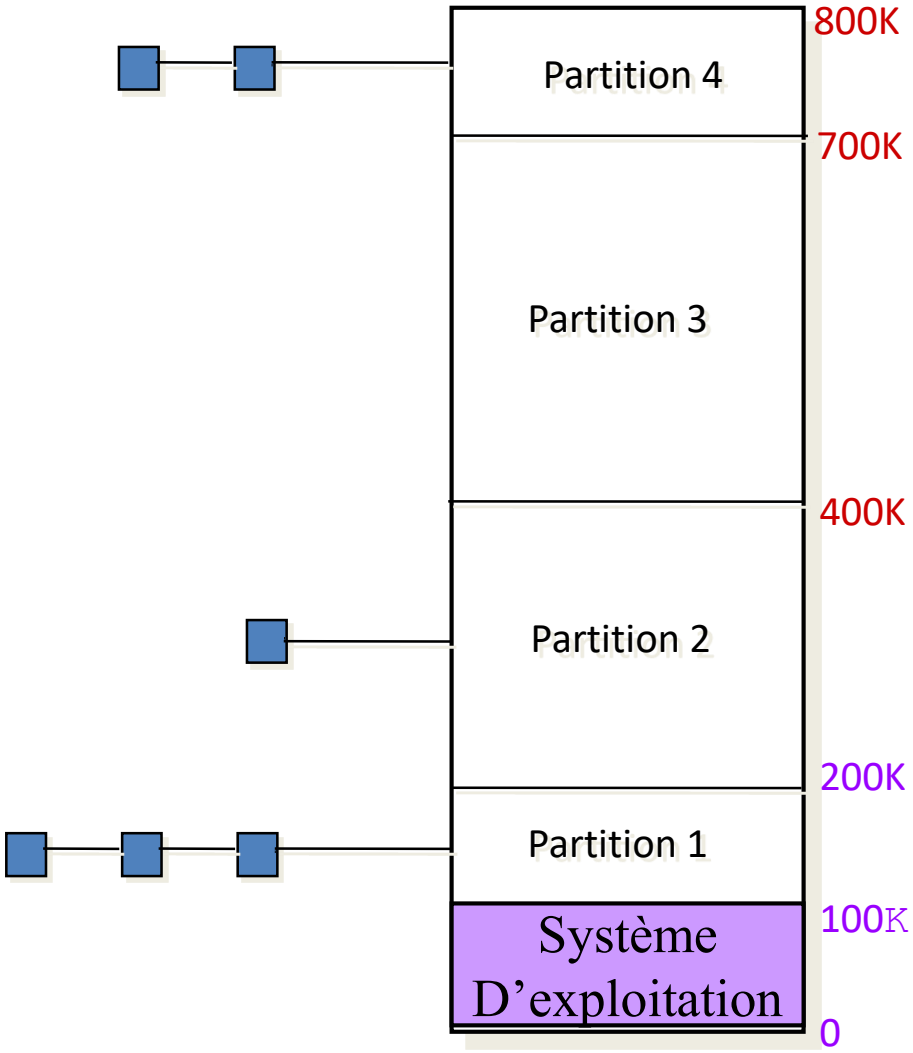
     Inconvénient : on peut ainsi affecter une partition de grande taille à une petite tâche et perdre beaucoup de place

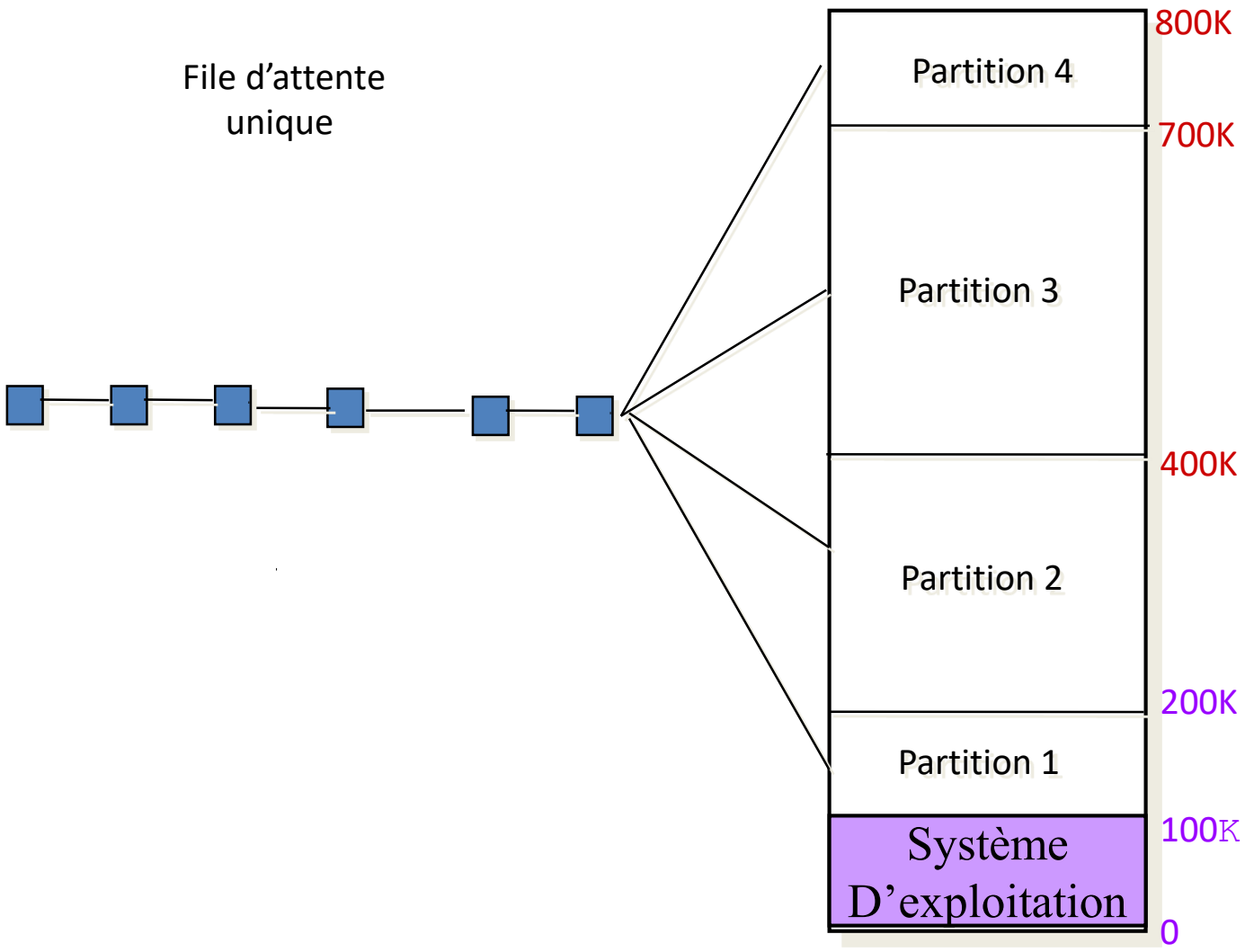
   - dès qu'une partition se libère, on lui affecte la plus grande tâche de la file qui peut y tenir.

     Inconvénient : on pénalise les processus de petite taille.

- Un autre inconvénient de l'allocation par partition de taille fixe est la saturation, un processus peut rapidement occuper l'ensemble de la partition qui lui est allouée et l'exécution du processus serait ainsi terminée pour faute d'espace supplémentaire. La seule issue dans une telle situation est le déplacement du processus sur une autre partition.

File d’attente multiple

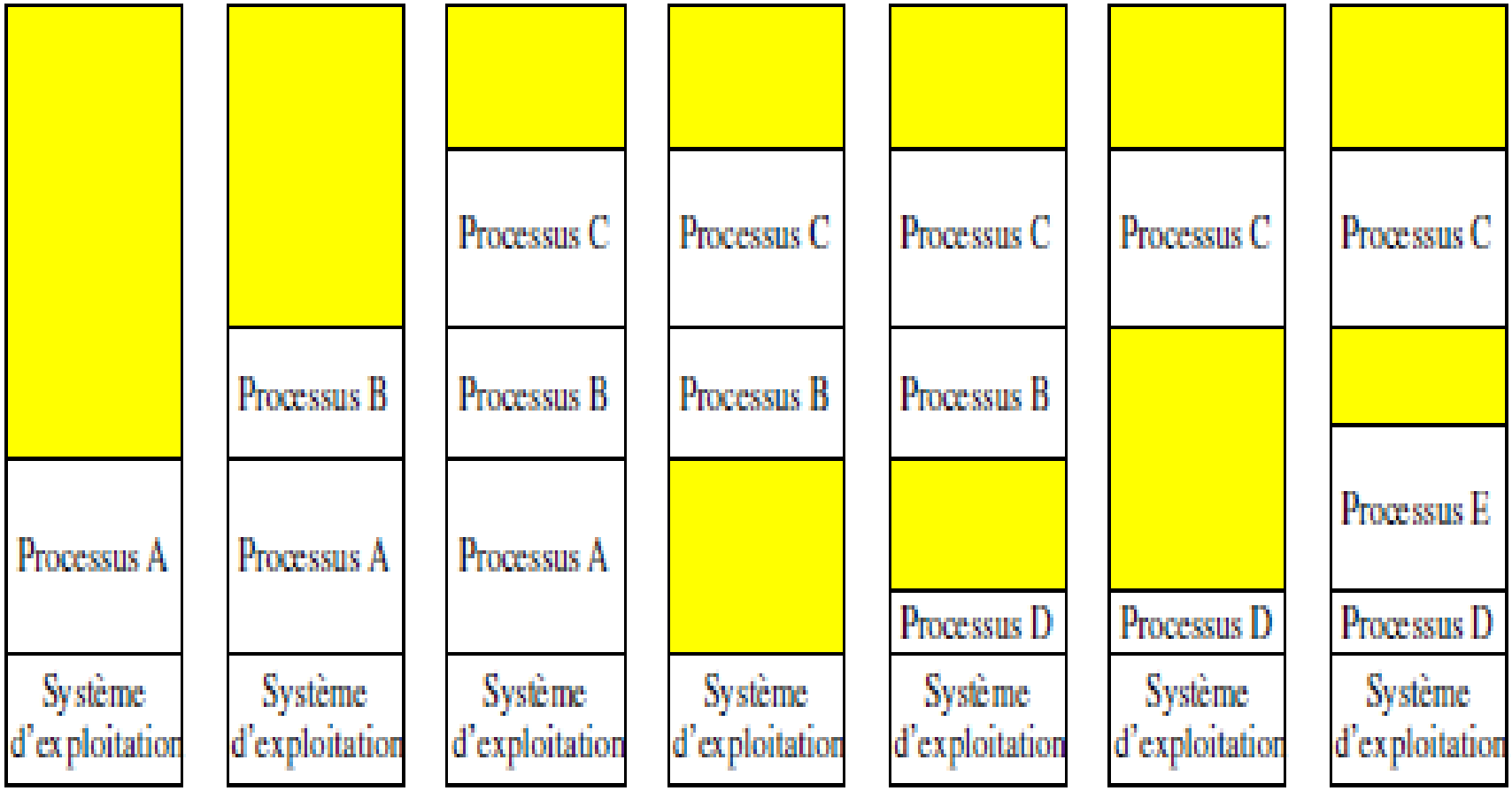



File d'attente
unique
Partition 4
Partition 3
Partition 2
Partition 1
Système
D'exploitation
800K
700K
400K
200K
100K
0

2.2 Les partitions de taille variable avec va et vient

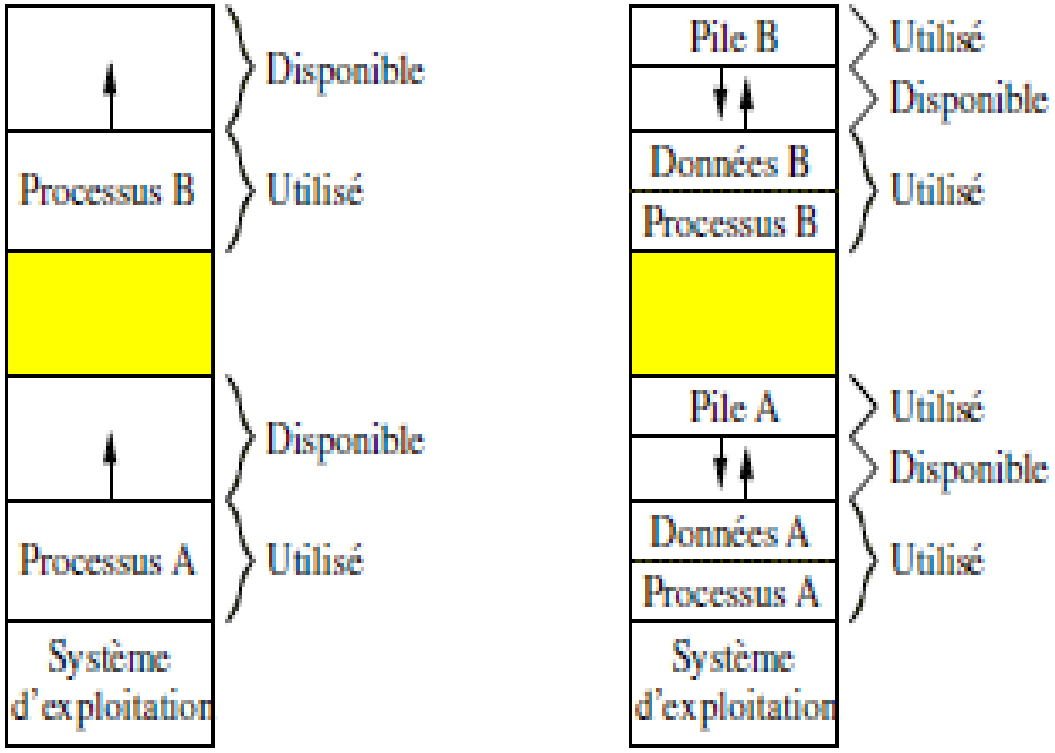
- Dans les systèmes multiprogrammés, dès que le nombre de processus devient supérieur au nombre de partitions, il faut stocker temporairement sur le disque des images de processus afin de libérer de la mémoire centrale pour d'autres processus. Il faut ramener régulièrement les processus stocker sur le disque en mémoire. Le mouvement des processus entre la mémoire et le disque est appelé va et vient (recouvrement ou swapping).

- En pratique on utilise des partitions de taille variable, car le nombre, la taille et la position des processus peuvent varier dynamiquement au cours du temps. On n'est plus limité par des partitions trop grandes ou trop petites comme avec les partitions fixes.



On améliore ainsi grandement l'utilisation de la mémoire en rendant cependant les politiques d'allocation et de libération plus compliquées

- Un problème important concerne la taille des partitions attribuée à chaque processus, car celle-ci tend à augmenter avec le temps.
- Une solution consiste à allouer à chaque processus un espace légèrement plus grand que sa taille actuelle comme le montre la figure suivante:



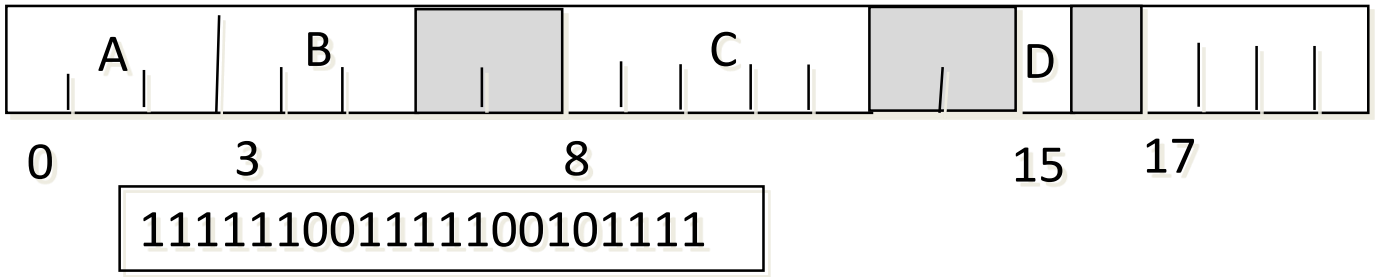
2.3 Gestion de la mémoire

Dans tous les cas, il faut disposer d'un mécanisme pour mémoriser les zones libre s et occupées, minimiser l'espace perdu lors d'une allocation et réduire autant que possible la fragmentation. Il existe trois manière de mémoriser l'occupation de la mémoire: les tables de bit, les listes et les subdivisions

1. **Gestion de la mémoire par table de bits**

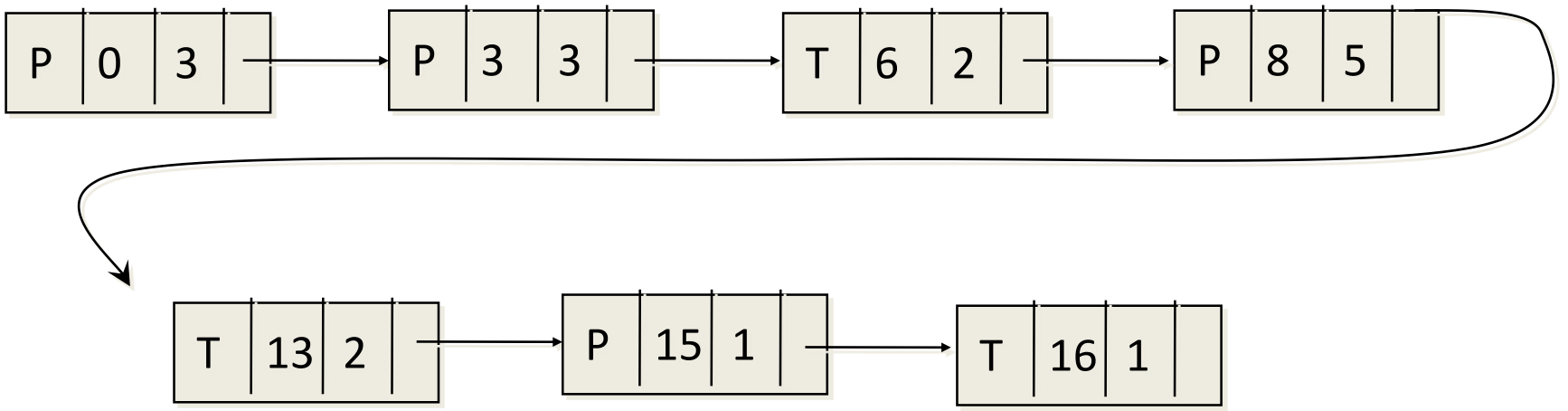
   On divise la MC en unités d'allocations de quelques octets à quelques Ko. A chaque unité, correspond un bit de la table de bits : valeur 0 si l'unité est libre, 1 sinon. Cette table est stockée en mémoire centrale. Plus la taille moyenne des unités est faible, plus la table occupe de place.

- A chaque swap (charger un processus en mémoire), le gestionnaire doit chercher une zone libre suffisamment grande pour contenir le processus (rechercher suffisamment de 0 consécutifs dans la table)
- L'exemple suivant montre une partie de la mémoire avec 4 processus et 3 trous (zones libres). Le tableau au dessous montre le table des bits correspondant.



2. Gestion de la mémoire par liste chaînée

On utilise une liste chaînée des segments de mémoire occupés et libres ; dans cette liste, un segment est soit un processus (P), soit un trou (zone Libre) entre deux processus. La mémoire de la figure précédente peut être représentée par une liste chaînée où chaque entrée de cette liste indique un trou (T) ou un processus (P), l'adresse à laquelle il débute et éventuellement sa taille, et elle spécifie un pointeur sur la prochaine entrée.



A l'achèvement d'un processus ou à son transfert sur disque, il faut du temps (mise à jour des liste chaînées) pour examiner si un regroupement avec ses voisins est possible pour éviter une fragmentation excessive de la mémoire.

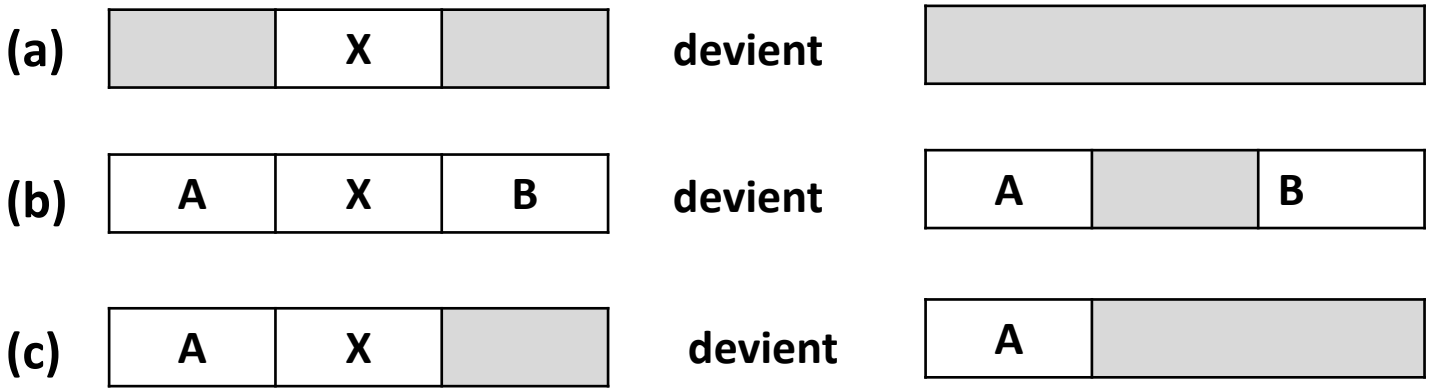
A la libération d'une zone mémoire, trois cas peuvent se présenter

a. La zone occupée est située entre deux zones libres

b. La zone occupée est situées ente deux zones occupées

c. La zone occupée est située ente une zone occupée et une zone libre.

Dans le cas a, il faut fusionner les 2 zones libres avec la zone libérée. Dans le cas c, il faut fusionner la zone libre adjacente avec la zone libérée.



Pour le chargement d’un processus, si on ne trouve pas de zone de taille suffisante mais qu’en réalité la somme des espaces libres est supérieur à la taille demandée (phénomène de fragmentation), la solution est d’appliquer un algorithme de compactage (ou ramasse miette, garbage collector en anglais) qui regroupe les espaces libres en un seul bloc en faisant des déplacements en mémoire des zones mémoires occupées.

- En résumé, les listes chaînées sont une solution plus rapide que la précédente pour l'allocation, mais plus lente pour la libération.

Allocation de la mémoire

Plusieurs algorithmes peuvent servir à allouer de la mémoire à un processus nouvellement crée ou un processus existant chargé depuis le disque.

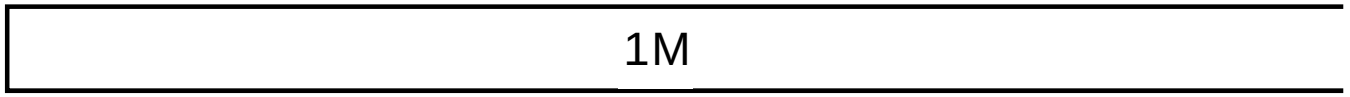
algorithme de la première zone libre (*first fit*): La liste est parcourue jusqu'à trouver une zone libre qui soit assez grand. Le bloc libre trouvé est ensuite divisé en deux partie, l'une destinée au processus et l'autre à la mémoire non utilisée. L'algorithme de la première zone libre est rapide parce qu'il limite ses recherches autant que possible.

algorithme du meilleur ajustement (*best fit*): La liste est entièrement parcourue, et on prend le plus petit bloc dont la taille est supérieure à celle de la mémoire demandée.

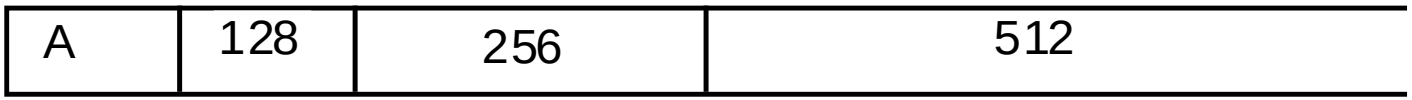
Algorithme du plus grand résidu (*worst fit):* similaire au précédent mais on prend la plus grande zone libre. Risque de perdre des zones qui ne seront plus utilisables.

3. Gestion de la mémoire par subdivisions

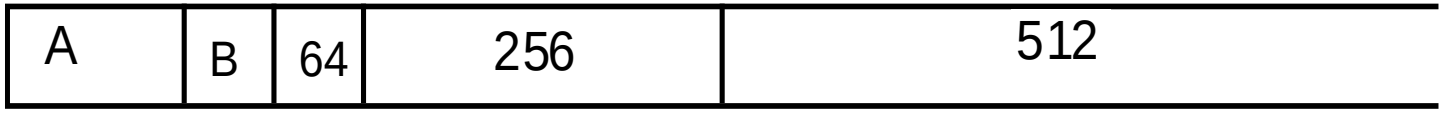
- Le gestionnaire mémorise une liste de blocs libres dont la taille est une puissance de 2 (1, 2, 4, 8 octets, ..., jusqu'à la taille maximale de la mémoire).
- Par exemple, avec une mémoire de 1 Mo, initialement, la mémoire est vide.



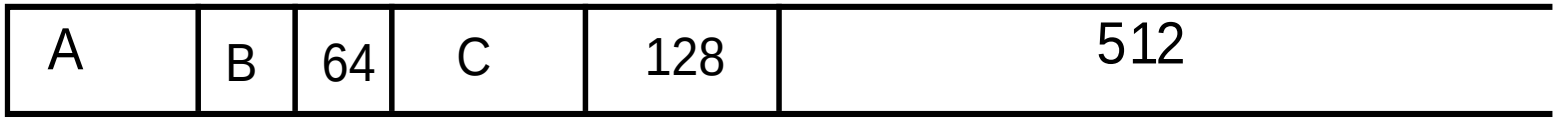
- Un processus A demande 70 Ko : la mémoire est fragmentée en deux blocs de 512 Ko; l'un d'eux est fragmenté en deux blocs de 256 Ko; l'un d'eux est fragmenté en deux blocs de 128 Ko et on loge A dans l'un d'eux, puisque $64 < 70 < 128$:



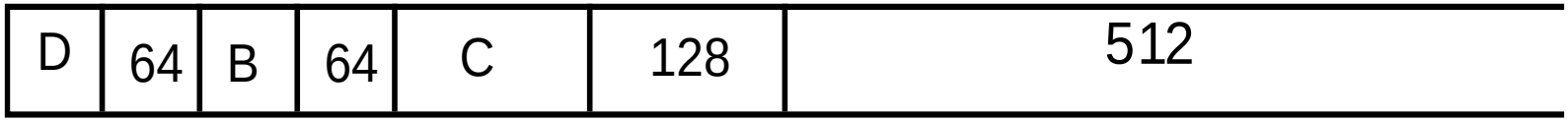
- Un processus B demande 35 Ko : l'un des deux blocs de 128 Ko est fragmenté en deux blocs de 64 Ko et on loge B dans l'un d'eux puisque $32 < 35 < 64$:



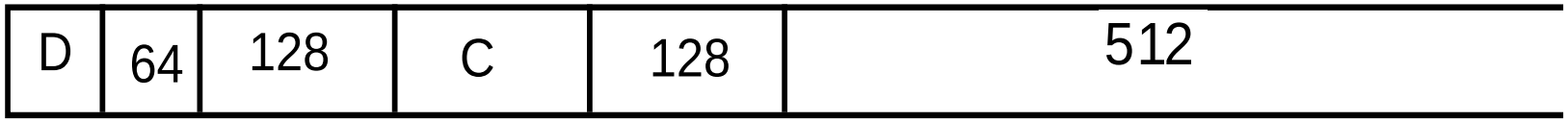
- Un processus C demande 80 Ko : le bloc de 256 Ko est fragmenté en deux blocs de 128 Ko et on loge C dans l'un d'eux puisque $64 < 80 < 128$:



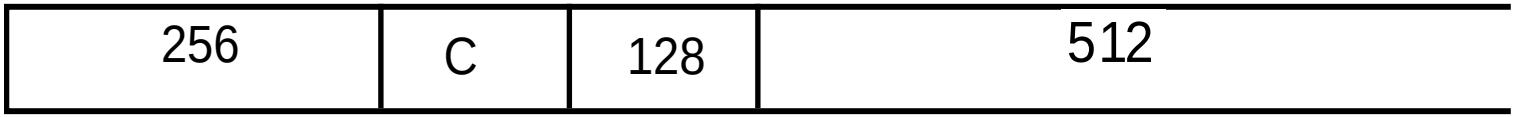
- A s'achève et libère son bloc de 128 Ko. Puis un processus D demande 60 Ko : le bloc libéré par A est fragmenté en deux de 64 Ko, dont l'un logera D :



- B s'achève, permettant la reconstitution d'un bloc de 128 Ko :



D s'achève, permettant la reconstitution d'un bloc de 256 Ko , etc...



- L'allocation et la libération des blocs sont très simples. La fusion se fait très rapidement, mais il peut y avoir beaucoup de perte mémoire. Un processus de taille $2^n + 1$ octets utilisera un bloc de 2^{n+1} octets ! Il y a beaucoup de perte de place en mémoire.

III - Allocation non contiguë en mémoire centrale

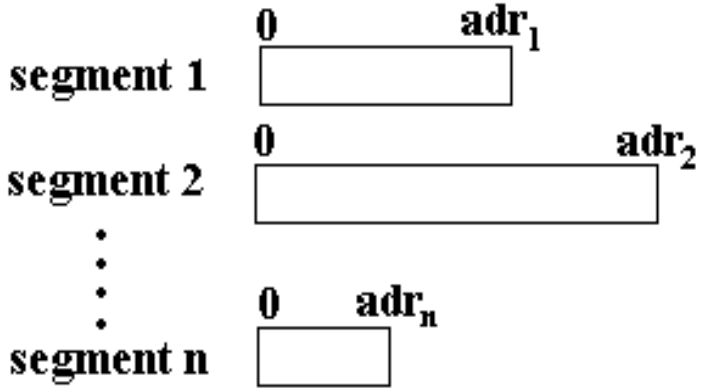
- C’est le mode d’allocation qui est appliqué par les systèmes actuels, ainsi un fichier peut être chargé à des adresses dispersées en mémoire. La correspondance entre les adresses virtuelles et les adresses physique est réalisée au cours de l’exécution.
- La mémoire peut être allouée par zones de taille fixe ou variable.
- Quand toutes les zones ont la même taille, on parle de **page** et de **systèmes paginés.**
- Quand leur taille peut varier, on parle **de segments et de systèmes segmentés.**
- On peut combiner les deux modes: des segments composés de pages.

3.1 Mémoire virtuelle et segmentation

- On désigne par mémoire virtuelle, une méthode de gestion de la mémoire physique permettant de faire exécuter une tâche dans un espace mémoire plus grand que celui de la mémoire centrale MC. Par exemple dans Windows et dans Linux, un processus fixé se voit alloué un espace mémoire de 4 Go. Si la mémoire centrale physique possède une taille de 512 Mo, le mécanisme de **mémoire virtuelle** permet de ne mettre à un instant donné dans les 512 Mo de la MC, que les éléments strictement nécessaires à l'exécution du processus, les autres éléments restant stockés sur le disque dur, prêts à être ramenés en MC à la demande.
- Un moyen employé pour gérer la topographie de cette mémoire virtuelle se nomme la segmentation, nous figurons ci-après une carte mémoire segmentée d'un processus.

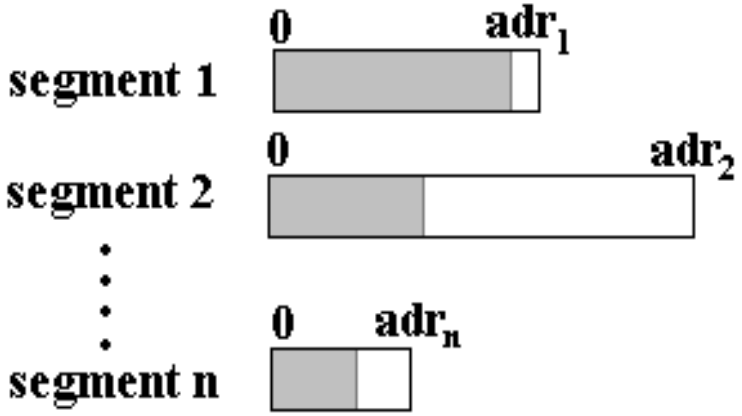
Segment de mémoire

- Un segment de mémoire est un ensemble de cellules mémoires contiguës.
- Le nombre de cellules d'un segment est appelé la taille du segment, ce nombre n'est pas nécessairement le même pour chaque segment, toutefois tout segment ne doit pas dépasser une taille maximale fixée.
- La première cellule d'un segment a pour adresse 0, la dernière cellule d'un segment adr_k est bornée par la taille maximale autorisée pour un segment.



Un segment contient généralement des informations de même type (du code, une pile, une liste, une table, ...) sa taille peut varier au cours de l'exécution (dans la limite de la taille maximale), par exemple une liste de données contenues dans un segment peut augmenter ou diminuer au cours de l'exécution.

– Les cellules d'un segment ne sont pas toutes nécessairement entièrement utilisées.

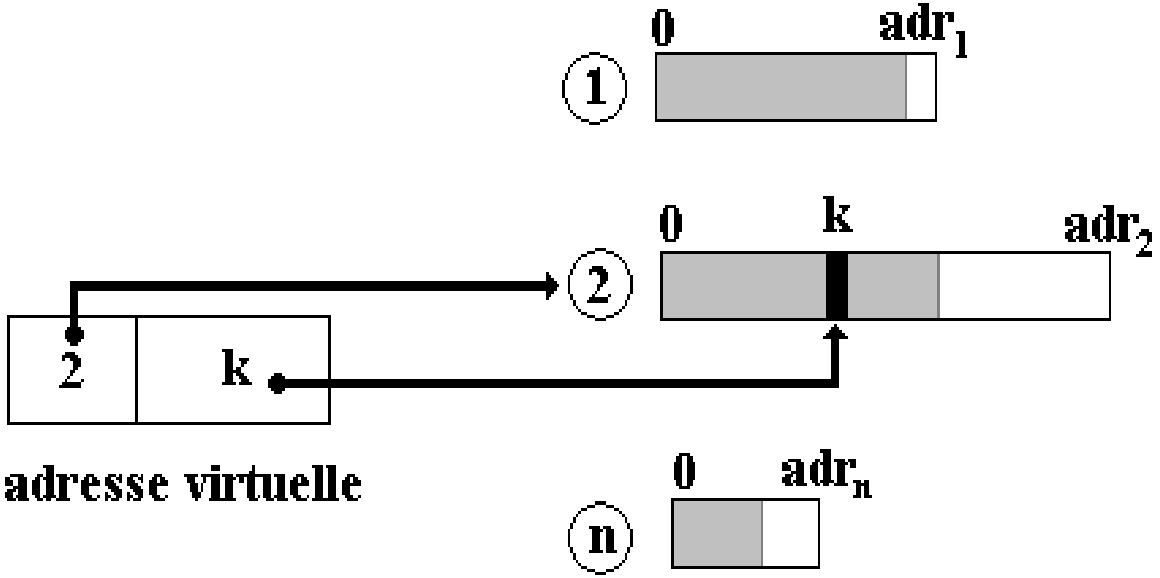


L'adresse d'une cellule à l'intérieur d'un segment s'appelle l'**adresse relative** au segment ou **déplacement**. On utilise plus habituellement la notion d'**adresse logique** permettant d'accéder à une donnée dans un segment, par opposition à l'**adresse physique** qui représente une adresse effective en mémoire centrale.

- C'est un ensemble de plusieurs segments que le système de gestion de la mémoire utilise pour allouer de la place mémoire aux divers processus qu'il gère.
- Chaque processus est **segmenté** en un nombre de segments qui dépend du processus lui-même.

Adresse logique ou virtuelle

- Une **adresse logique** aussi nommée **adresse virtuelle** comporte deux parties : le numéro du segment auquel elle se réfère et l'adresse relative de la cellule mémoire à l'intérieur du segment lui-même.



Remarques :

- Le nombre de segments présents en MC n'est pas fixe.
- La taille effective d'un segment peut varier pendant l'exécution
- Pendant l'exécution de plusieurs processus, la MC est divisée en deux catégories de blocs : les blocs de **mémoire libre** (libéré par la suppression d'un segment devenu inutile) et les blocs **de mémoire occupée** (par les segments actifs).

Fragmentation mémoire

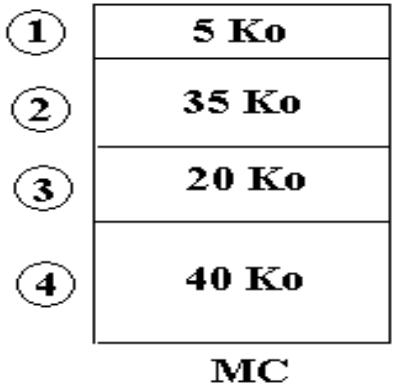
- Le partitionnement de la MC entre blocs libres et blocs alloués se dénomme la fragmentation mémoire, au bout d'un certain temps, la mémoire contient une multitude de blocs libres qui deviendront statistiquement de plus en plus petits jusqu'à ce que le système ne puisse plus allouer assez de mémoire contiguë à un processus.

Exemple

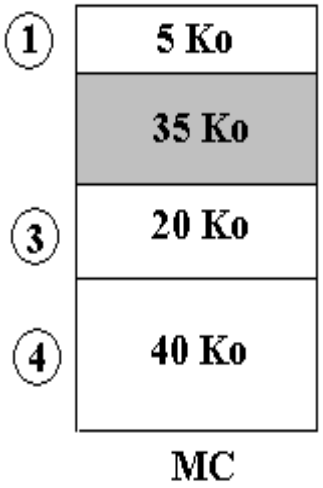
- Soit une MC fictive de 100 Ko segmentable en segments de taille maximale 40 Ko, soit un processus P segmenté par le système en 6 segments dont nous donnons la taille dans le tableau suivant :

Numéro du segment	Taille du segment
1	5 Ko
2	35 Ko
3	20 Ko
4	40 Ko
5	15 Ko
6	23 Ko

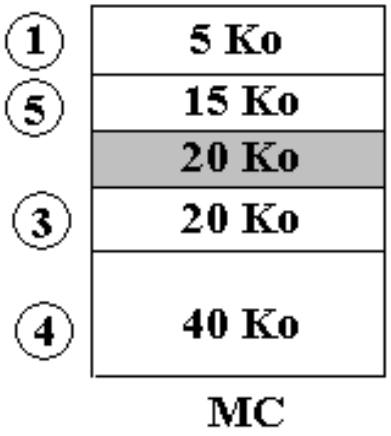
Supposons qu'au départ, les segments 1 à 4 sont chargés dans la MC :



Supposons que le segment n°2 devenu inutile soit désalloué :

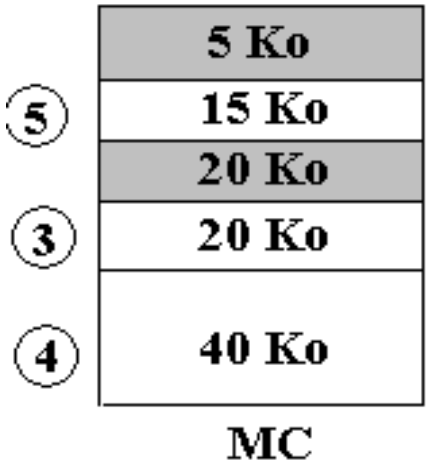


Puis chargeons en MC le segment n°5 de taille 15 Ko dans l'espace libre qui passe de 35 Ko à 20 Ko :

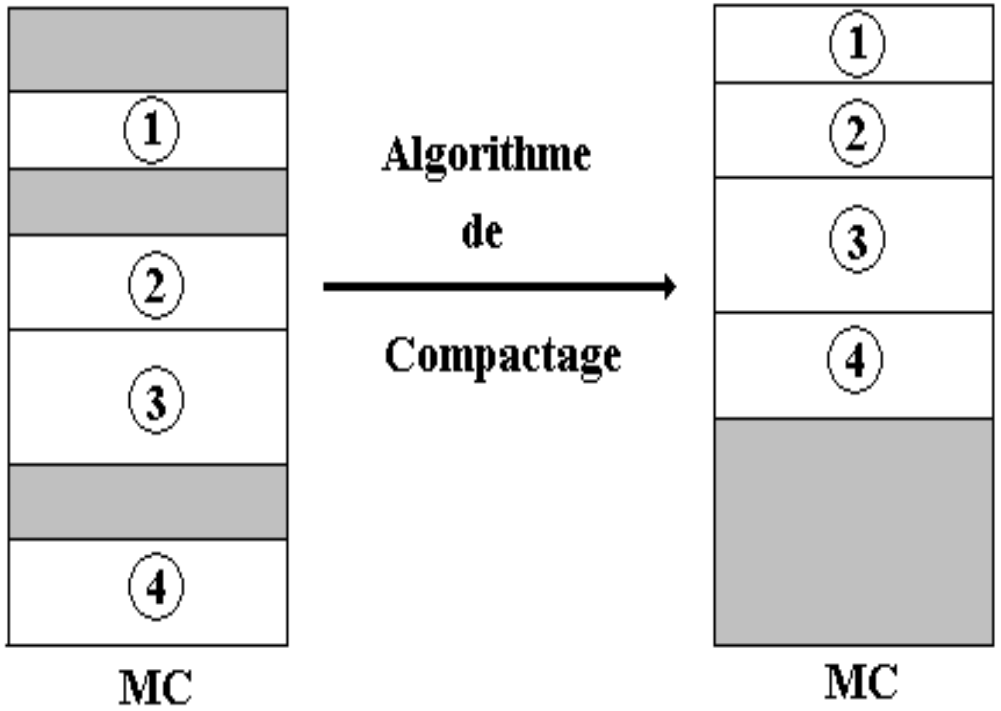


La taille du bloc d'espace libre diminue.

Continuons l'exécution du processus P en supposant que ce soit maintenant le segment n°1 qui devienne inutile :



- Il y a maintenant séparation de l'espace libre (fragmentation) en deux blocs, l'un de 5 Ko de mémoire contiguë, l'autre de 20 Ko de mémoire contiguë, soit un total de 25 Ko de mémoire libre. Il est toutefois impossible au système de charger le segment n°6 qui occupe 23 Ko de mémoire, car il lui faut 23 Ko de mémoire contiguë. Les système doit alors procéder à une réorganisation de la mémoire libre afin d'utiliser "au mieux" ces 25 Ko de mémoire libre.



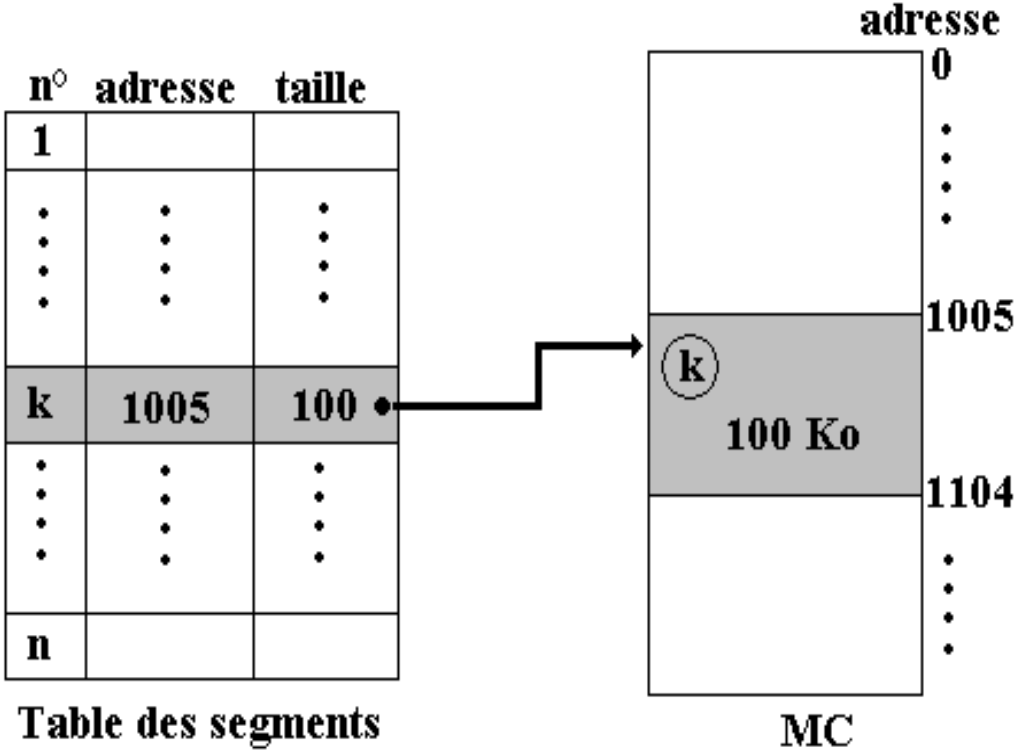
La figure précédente montre à gauche, une mémoire fragmentée, et à droite la même mémoire une fois compactée.

Adresse virtuelle - adresse physique

- Nous avons parlé d'adresse logique d'une donnée. Comment le système de gestion d'une mémoire segmentée retrouve-t-il l'adresse physique associée ?
- l'OS dispose pour cela d'une table décrivant la "carte" mémoire de la MC.
- Cette table est dénommée table des segments, elle contient une entrée par segment actif et présent dans la MC.
- Une entrée de la table des segments comporte le numéro du segment, l'adresse physique du segment dans la MC et la taille du segment.

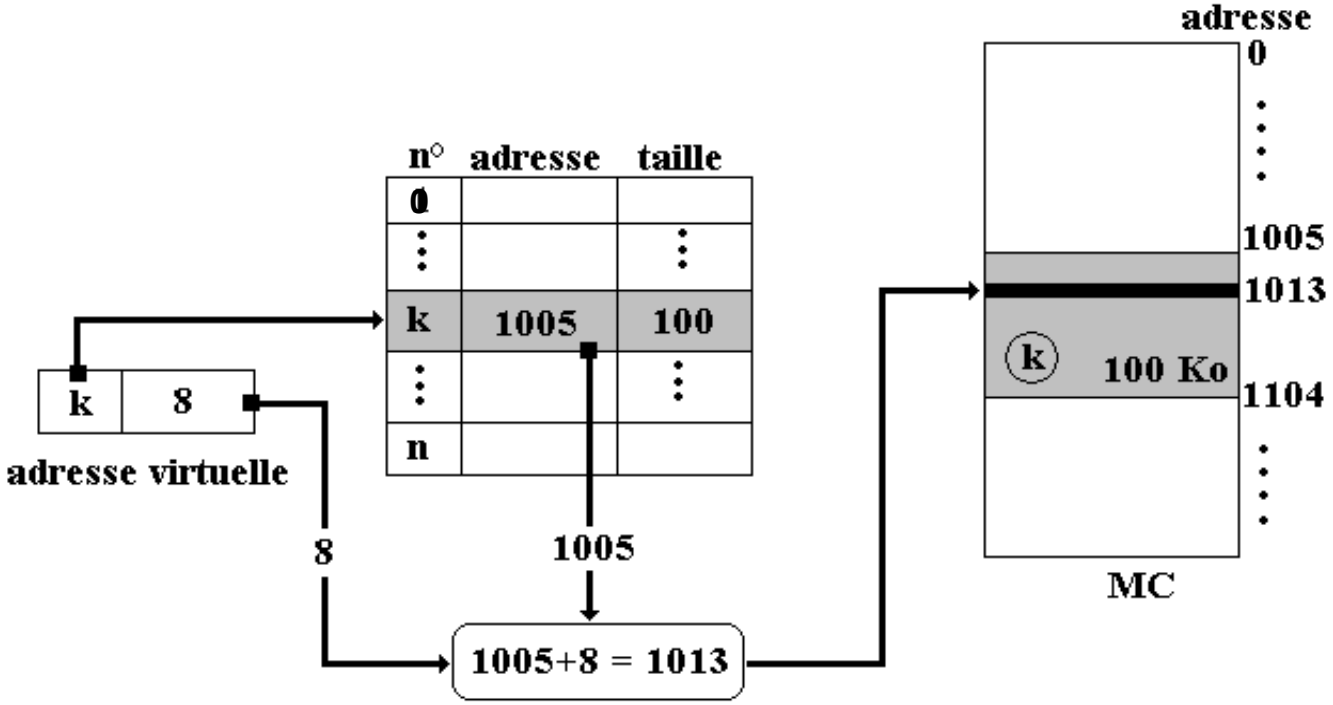
n° segment	adresse segment	taille segment
n° segment	adresse segment	taille segment
n° segment	adresse segment	taille segment
⋮		
n° segment	adresse segment	taille segment

Liaison entre Table des segments et le segment lui-même en MC :



Lorsque le système de gestion mémoire rencontre une adresse virtuelle de cellule (n° segment, Déplacement), il va chercher dans la table l'entrée associée au numéro de segment, récupère dans cette entrée l'adresse de départ en MC du segment et y ajoute le déplacement de l'adresse virtuelle et obtient ainsi l'adresse physique de la cellule.

En reprenant l'exemple de la figure précédente, supposons que nous présentons l'adresse virtuelle (**k** , 8). Il s'agit de référencer la cellule d'adresse 8 à l'intérieur du segment numéro **k**. Comme le segment n°k est physiquement implanté en MC à partir de l'adresse 1005, la cellule cherchée dans le segment se trouve donc à l'adresse physique 1005+8 = 1013.



La segmentation mémoire n'est pas la seule méthode utilisée pour gérer de la mémoire virtuelle, nous proposons une autre technique de gestion de la mémoire virtuelle très employée : la pagination mémoire. Les OS actuels employant un mélange de ces deux techniques,

3.2 Mémoire virtuelle et pagination

- Comme dans la segmentation mémoire, la pagination est une technique visant à partitionner la mémoire centrale en blocs (nommés ici **cadres de pages**) de taille fixée contrairement aux segments de taille variable.
- Lors de l'exécution de plusieurs processus découpés chacun en plusieurs pages nommées **pages virtuelles**, on parle alors de **mémoire virtuelle paginée**. Le nombre total de mémoire utilisée par les pages virtuelles de tous les processus, excède généralement le nombre de cadres de pages disponibles dans la MC.

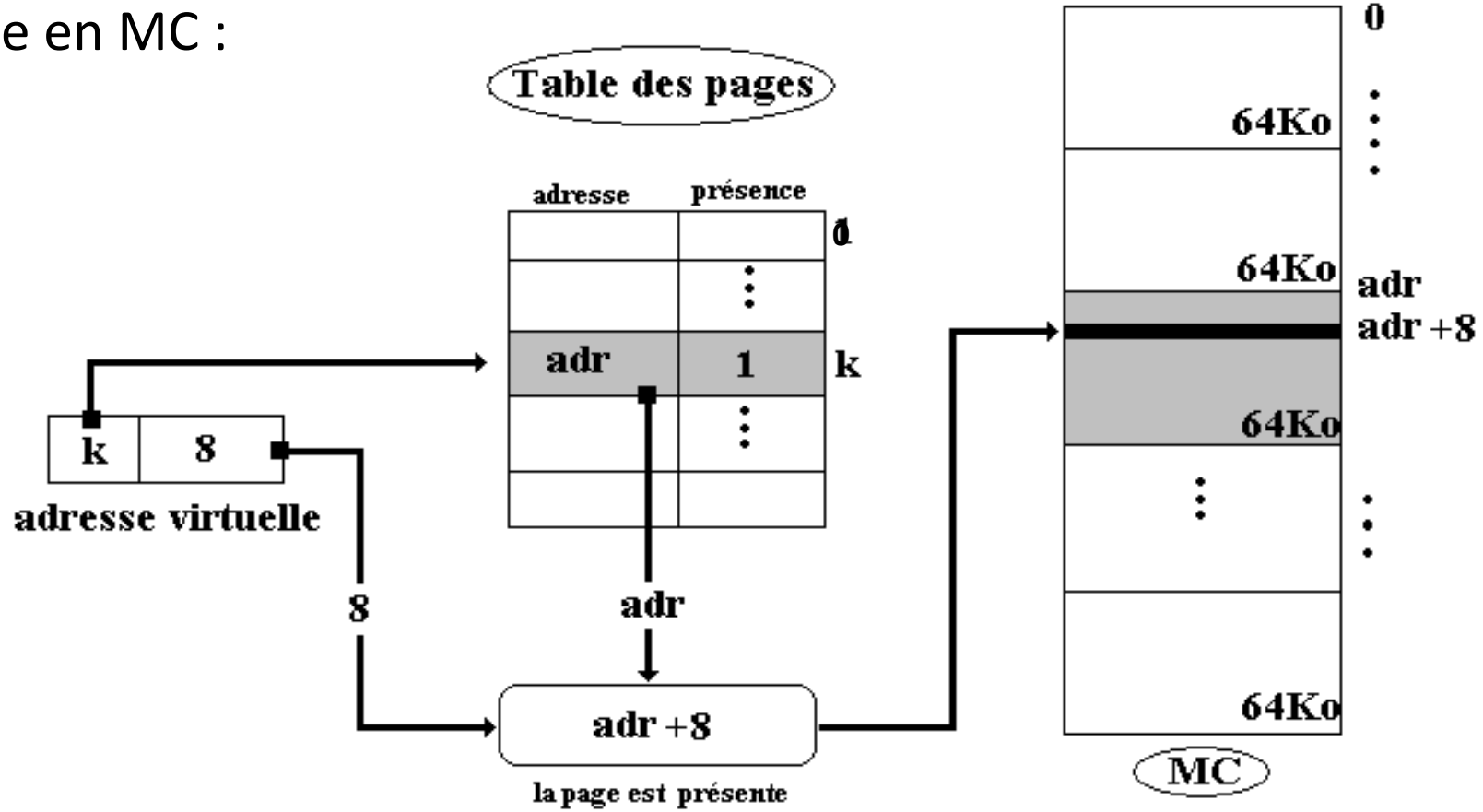
- Le système de gestion de la mémoire virtuelle paginée est chargé de gérer l'allocation et la désallocation des pages dans les cadres de pages.
- La MC est divisée en un nombre de cadres de pages fixé par le système (généralement la taille d'un cadre de page est une puissance de 2 inférieure ou égale à 64 Ko).
- La taille d'une page virtuelle est exactement la même que celle d'un cadre de page.
- Comme le nombre de pages virtuelles est plus grand que le nombre de cadres de pages on dit aussi que l'espace d'adressage virtuel est plus grand que l'espace d'adressage physique. Seul un certain nombre de pages virtuelles sont présentes en MC à un instant fixé.

- Comme le cas de la segmentation, l'adresse virtuelle (logique) d'une donnée dans une page virtuelle, est composée par le numéro d'une page virtuelle et le déplacement dans cette page. L'adresse virtuelle est transformée en une adresse physique réelle en MC, par une entité se nommant la MMU (Memory Management Unit) assistée d'une table des pages semblable à la table des segments.

La table des pages virtuelles

- Nous avons vu dans le cas de la segmentation que la table des segments était plutôt une liste (ou table dynamique) ne contenant que les segments présent en MC, le numéro du segment étant contenu dans l'entrée. La table des pages virtuelles quant à elle, est un vrai tableau indicé sur les numéros de pages. Le numéro d'une page est l'indice dans la table des pages, d'une entrée contenant les informations permettant d'effectuer la conversion d'une adresse virtuelle en une adresse physique.
- Comme la table des pages doit référencer toutes les pages virtuelles et que seulement quelques unes d'entre elles sont physiquement présentes en MC, chaque page virtuelle se voit attribuer un drapeau de présence (représenté par un bit, la valeur 0 indique que la page est actuellement absente, la valeur 1 de ce bit indique qu'elle est actuellement présente en MC).

- Le schéma simplifié d'une gestion de MC paginée (page d'une taille de 64Ko) suivant illustre le même exemple que pour la segmentation. Soit un accès à une donnée d'adresse 8 dans la page de rang k, le cadre de page en MC ayant pour adresse 1005, la page étant présente en MC :



Lorsque la même demande d'accès à une donnée d'une page a lieu sur une page qui n'est pas présente en MC, la MMU se doit de la charger en MC pour poursuivre les opérations.

- Le système d'exploitation utilise une structure de données pour décrire les pages d'un processus. Une entrée de la table contient les informations suivantes :

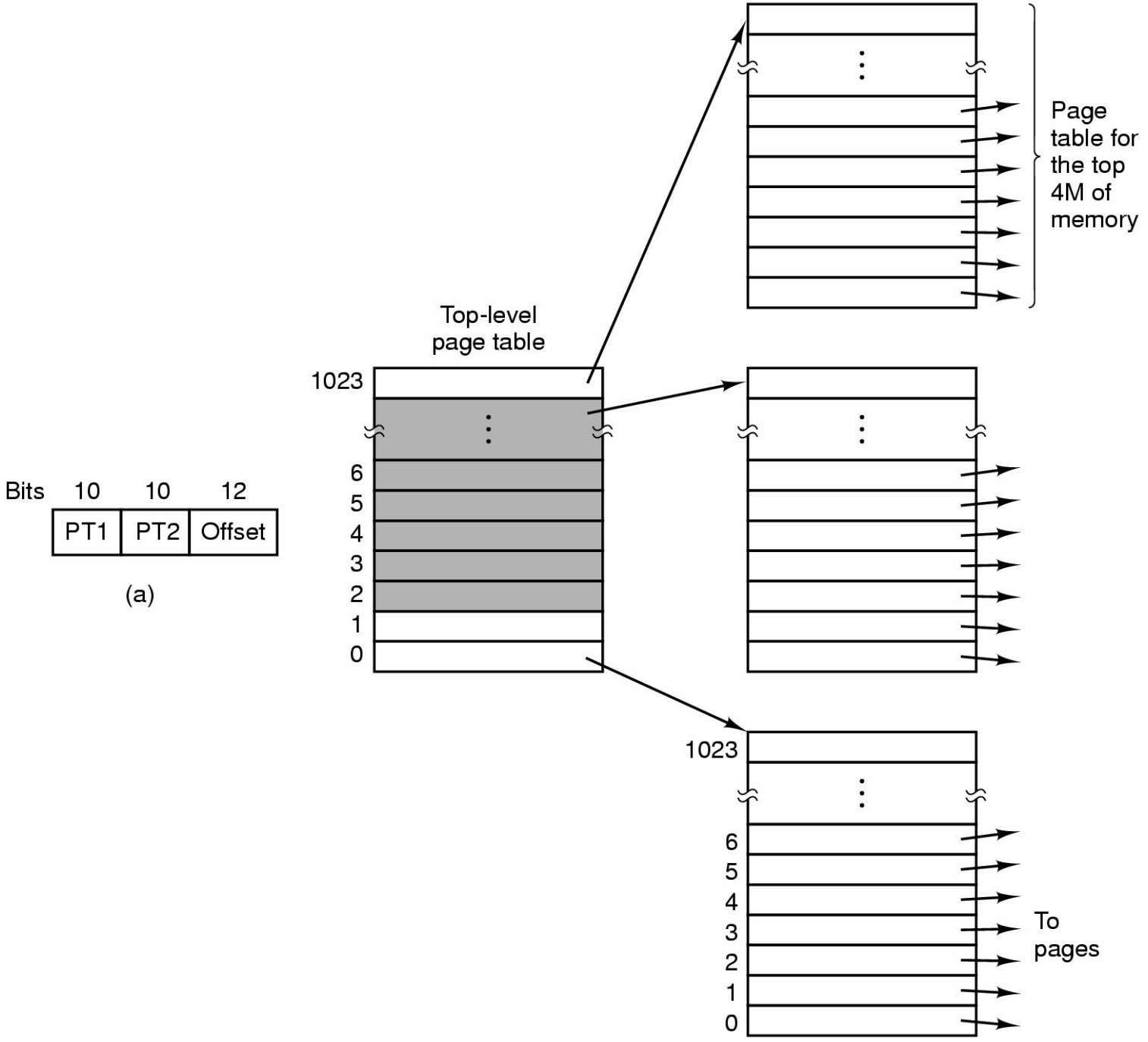
Bit de présence	Bit de modification	Bits de protection	Bit de référence	Adresse sur le DD	Numéro de la case mémoire

- Le bit de présence indique si la page est chargée en mémoire ou non
- Les bits de protection définissent le mode d'accès à la page en lecture ou en écriture
- Le bit de modification permet d’économiser une recopie sur le disque si la case va être allouée à une autre page.
- Le bit de référence indique que la page placée dans cette case a été référencée c'est-à-dire accédée, cette information sert aux algorithmes de remplacement de pages.

La table des pages virtuelles multiniveaux

- chaque processus a sa propre table qui doit être chargée en mémoire,
- La table des pages contient une entrée pour chaque page de l'espace d'adressage.
- Si le système d'exploitation applique la mémoire virtuelle avec un registre d'adresse de 32 bits, l'espace adressable contient 2^{32} adresses.
- Si la taille de chaque page est 4 KO, une page peut contenir 4096 adresses. Le nombre des pages est égal à la taille de la mémoire virtuelle divisé par la taille d'une page. Soit 4 G0/4 KO = 2^{20} environ 1 million de pages.
- Ainsi le principal inconvénient des systèmes paginés est la taille gigantesque de la table des pages.

- La solution a été proposée avec la pagination multi-niveaux où la table des pages est décomposée en petites tables de taille raisonnable qui sont chargées au fur et à mesure que le système en a besoin.
- On peut souligner deux intérêts à cette organisation : partant du fait qu'une adresse n'appartient à l'espace d'adressage d'un processus que si elle est utilisée par le processus, les tables sont allouées au fur et à mesure que les adresses qu'elles comportent sont utilisées.
- Par ailleurs on ne charge à partir du disque que les tables dont on se sert, ainsi on charge à la demande ce qui évite d'avoir toutes les tables en mémoire centrale. On dit alors que la table des pages est elle-même paginée.

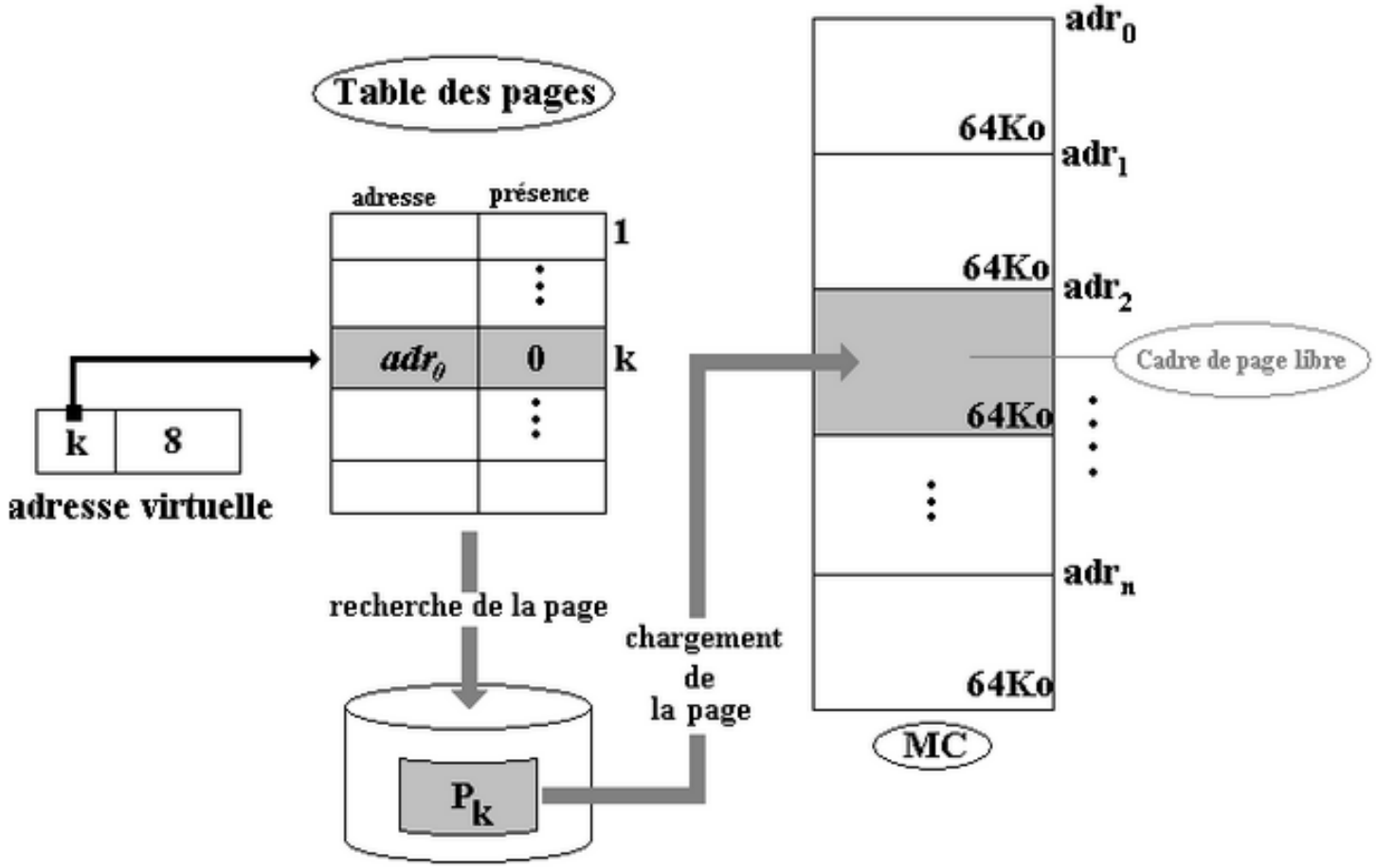

Bits 10 10 12
PT1
PT2
Offset
(a)
Top-level page table
1023
6
5
4
3
2
1
0
Page table for the top 4M of memory
1023
6
5
4
3
2
1
0
To pages

- La table du premier niveau comporte 1024 entrées et permet d'adresser autant de tables de pages (1024 = 2^{10}). Ainsi les 10 premiers bits de l'adresse logique sont considérés comme index dans cette table.
- Une fois qu'on a accédé à l'entrée adéquate, on trouve l'adresse de la case mémoire où se trouve la table des pages (de second niveau). Les 10 autres bits sont alors utilisés comme index dans cette table et l'entrée nous délivre la case mémoire associée à la page virtuelle.
- Les pages ayant une taille de 4 KO soit 2^{12}, on retrouve les 32 bits de l'adresse logique (10 + 10 + 12).

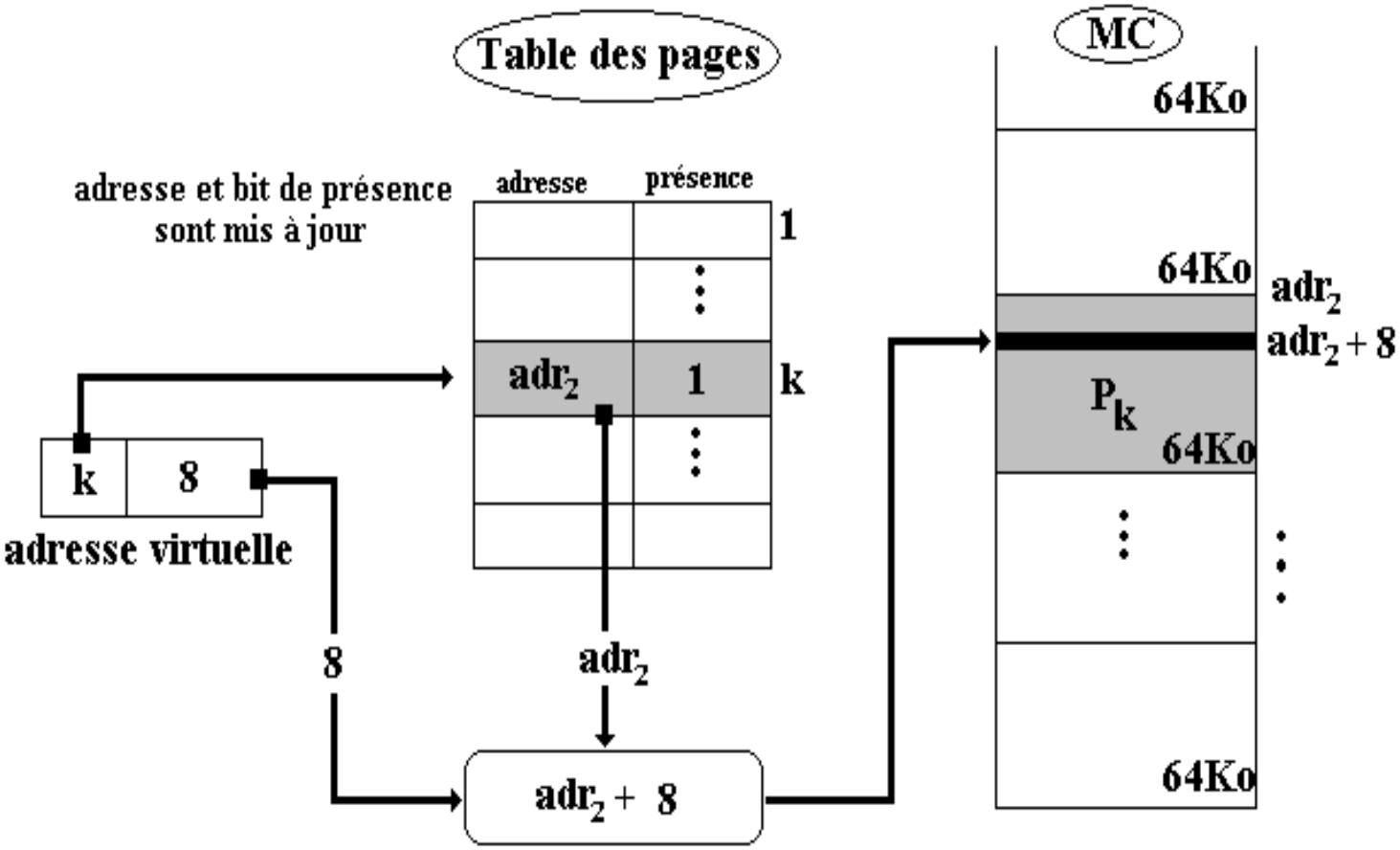
- Prenons l'exemple de l'adresse virtuelle 5269875 que l'on écrit 506973 en hexadécimal. Les 12 bits de poids faible donnent un déplacement de 973. Le nombre 506 en hexadécimal s'écrit sur 12 bits en binaire $(010100000110)_2$, si nous retenons les 10 bits correspondants à l'index dans la seconde table, nous obtenons un numéro de page = 100000110 soit l'entrée 262 dans la table de deuxième niveau et les 2 bits restants $(01)_2$ l'entrée n° 1 dans la table de premier niveau.
- La pagination à 2 niveaux ne résout pas définitivement le problème de la taille de la table des Pages. C'est pour cette raison que les systèmes d'exploitation utilisent des tables à plusieurs niveaux généralement 3 ou 4. Il va de soi que le nombre de niveaux est décidé par le matériel, donc lié au MMU utilisé pour gérer la pagination

Défaut de page

- Nous dirons qu'il y a défaut de page lorsque le processeur envoie une adresse virtuelle localisée dans une page virtuelle dont le bit de présence indique que cette page est absente de la mémoire centrale. Dans cette éventualité, le système doit interrompre le processus en cours d'exécution, il doit ensuite lancer une opération d'entrée-sortie dont l'objectif est de rechercher et trouver un cadre de page libre disponible dans la MC dans lequel il pourra mettre la page virtuelle qui était absente, enfin il mettra à jour dans la table des pages le bit de présence de cette page et l'adresse de son cadre de page.
- Le défaut de page peut entraîner un remplacement si le système d'exploitation ne trouve aucune case libre.



La figure précédente illustre un défaut de page d'une page P_k qui avait été anciennement chargée dans le cadre d'adresse adr_0, mais qui est actuellement absente. La MMU recherche cette page par exemple sur le disque, recherche un cadre de page libre (ici le bloc d'adresse adr_2 est libre) puis charge la page dans le cadre de page et l'on se retrouve ramené au cas d'une page présente en MC :



En fait, lorsqu'un défaut de page se produit tous les cadres de pages contiennent des pages qui sont marquées présentes en MC, il faut donc en sacrifier une pour pouvoir caser la nouvelle page demandée. Il est tout à fait possible de choisir aléatoirement un cadre de page, de le sauvegarder sur disque et de l'écraser en MC par le contenu de la nouvelle page.

- Cette attitude qui consiste à faire systématiquement avant tout chargement d'une nouvelle page une sauvegarde de la page que l'on va écraser, n'est pas optimisée car si la page que l'on sauvegarde est souvent utilisée elle pénalisera plus les performances de l'OS (car il faudra que le système recharge souvent) qu'une page qui est très peu utilisée (qu'on ne rechargera pas souvent).
- Cette recherche d'un "bon" bloc à libérer en MC lors d'un défaut de page est effectuée selon plusieurs algorithmes appelés algorithmes de remplacement(voir TD)

Chapitre 5 :gestion des fichiers

I - Introduction

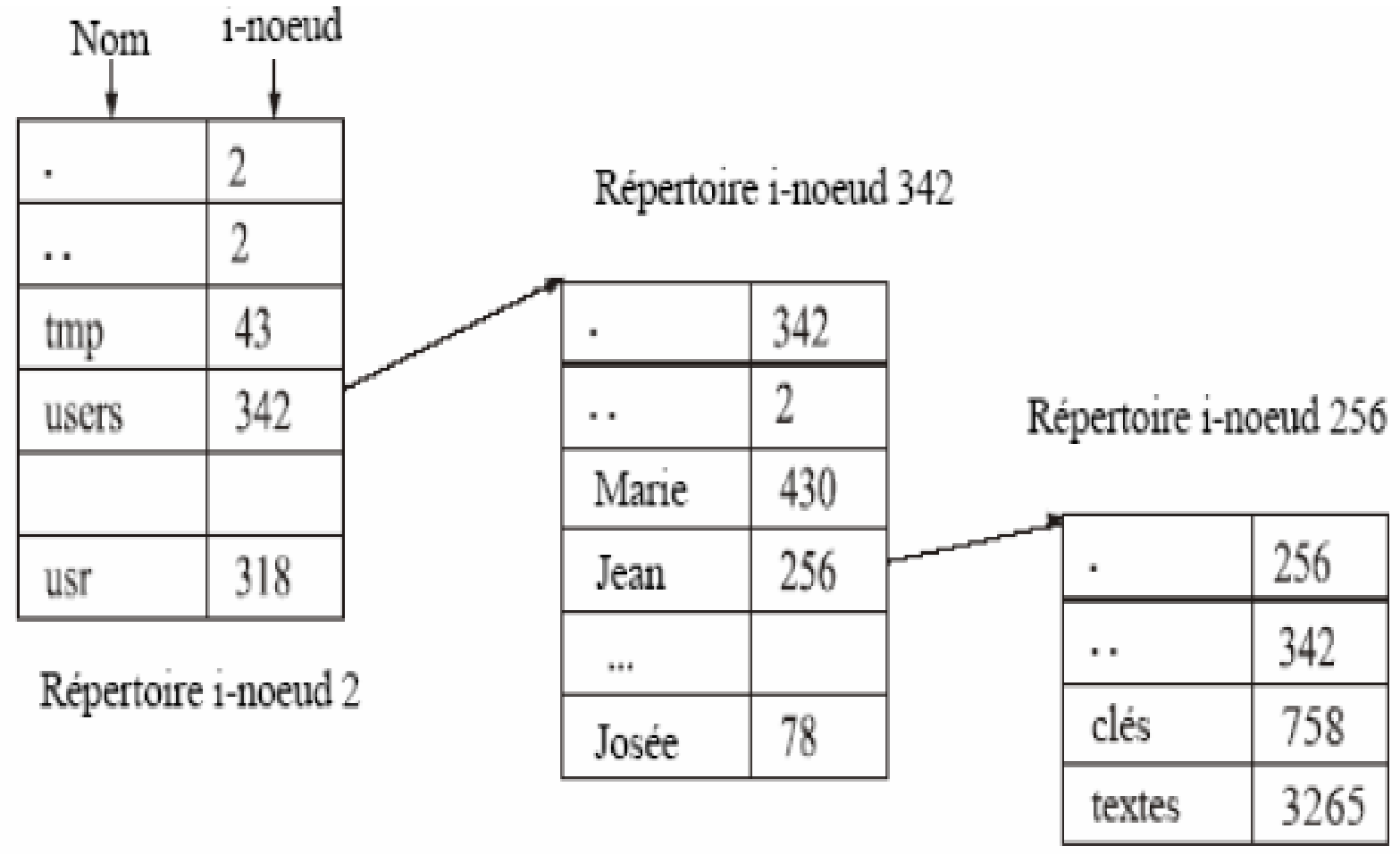
- Le système de gestion de fichiers (SGF) est la partie la plus visible d'un système d'exploitation qui se charge de gérer le stockage et la manipulation de fichiers (sur une unité de stockage : partition, disque, CD, disquette.
- Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers.

Le concept de fichier

- Un fichier est l'unité de stockage logique mise à la disposition des utilisateurs pour l'enregistrement de leurs données. Le SE établi la correspondance entre le fichier et le système binaire utilisé lors du stockage de manière transparente pour les utilisateurs.
- Dans un fichier on peut écrire du texte, des images, des calculs, des programmes…
- Les fichiers sont généralement créés par les utilisateurs. Toutefois certains fichiers sont générés par les systèmes ou certains outils tels que les compilateurs.
- Afin de différencier les fichiers entre eux, chaque fichier a un ensemble d'attributs qui le décrivent. Parmi ceux-ci on retrouve : le nom, l'extension, la date et l'heure de sa création ou de sa dernière modification, la taille, la protection.

La notion de répertoire

- Un répertoire est une entité crée pour l'organisation des fichiers. En effet on peut enregistrer des milliers, voir des millions de fichiers sur un disque dur et il devient alors impossible de s'y retrouver.
- Avec la multitude de fichiers créés, le système d'exploitation a besoin d'une organisation afin de structurer ces fichiers et de pouvoir y accéder rapidement. Cette organisation est réalisée au moyen de **répertoires** également appelés **catalogues ou directory.**
- Un répertoire est lui-même un fichier puisqu'il est stocké sur le disque et est destiné à contenir des fichiers.
- Du point de vue SGF, un répertoire est un fichier qui dispose d'une structure logique : il est considéré comme un tableau qui contient une entrée par fichier. L'entrée du répertoire permet d'associer au nom du fichier (nom externe au SGF) les informations stockées en interne par le SGF.
- Chaque entrée peut contenir des informations sur le fichier (attributs du fichier) ou faire référence à (pointer sur) des structures qui contiennent ces informations.


Nom
i-noeud
.
2
..
2
tmp
43
users
342
usr
318
Répertoire i-noeud 2
Répertoire i-noeud 342
.
342
..
2
Marie
430
Jean
256
...
Josée
78
Répertoire i-noeud 256
.
256
..
342
clés
758
textes
3265

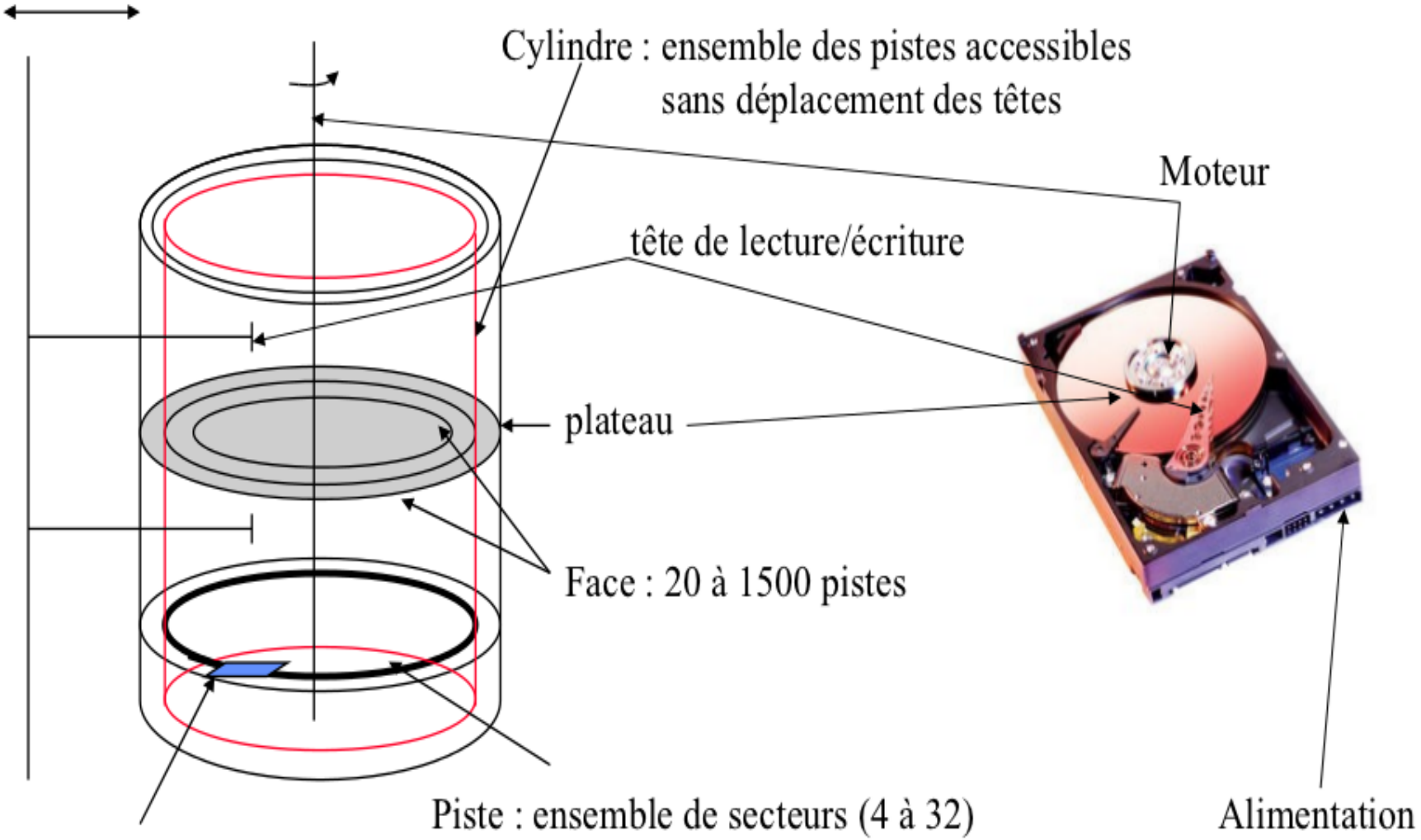
Rôles d'un système de gestion de fichiers

Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers. Il effectue généralement les tâches suivantes :

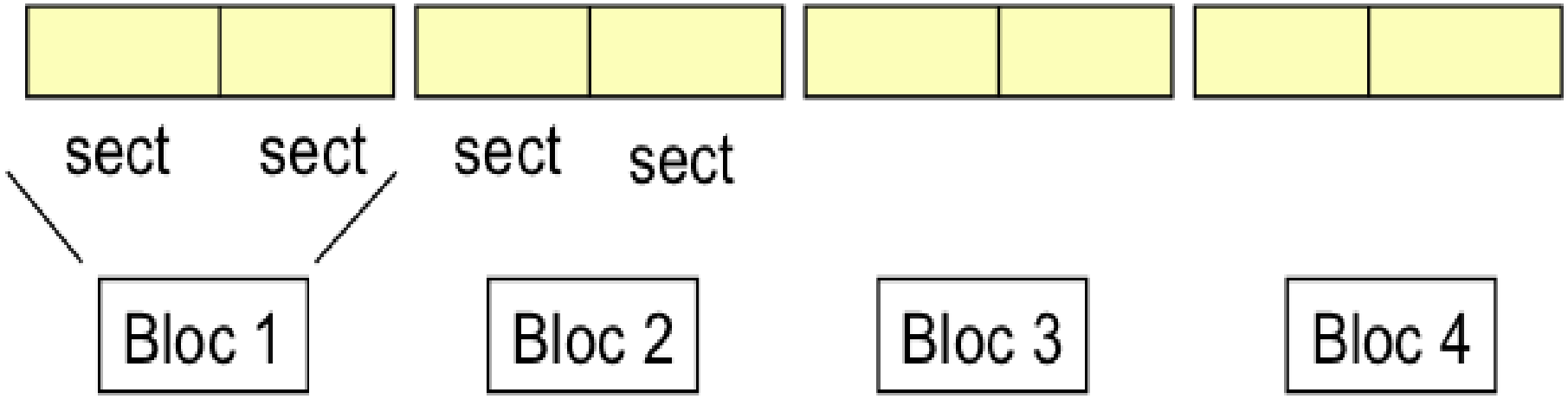
- Fournit une interface conviviale pour manipuler les fichiers (vue fournie à l'utilisateur).
  - Il s'agit de simplifier la gestion des fichiers pour l'utilisateur (généralement, l'utilisateur fournis seulement les attributs nom et extension du fichier, les autres attributs sont gérés implicitement par le SGF)
  - Cette interface fournit la possibilité d'effectuer plusieurs opérations sur les fichiers. Ces opérations permettent généralement d'ouvrir, de fermer, de copier, de renommer des fichiers et des répertoires.
- La gestion de l'organisation des fichiers sur le disque (allocation de l'espace disque aux fichiers)
- La gestion de l'espace libre sur le disque dur
- La gestion des fichiers dans un environnement Multi-Utilisateurs, la donnée d'utilitaires pour le diagnostic, la récupération en cas d'erreurs, l'organisation des fichiers.

Structure du disque dur

Adresse d'un secteur : n°face, n°cylindre,n°secteur



- L’unité d’allocation sur le disque dur est le bloc physique. Il est composé de 1 à n secteurs.
- Un bloc composé de deux secteurs de 512 octets a une taille égale à 1KO
- Les opérations de lecture et d'écriture du SGF se font bloc par bloc.

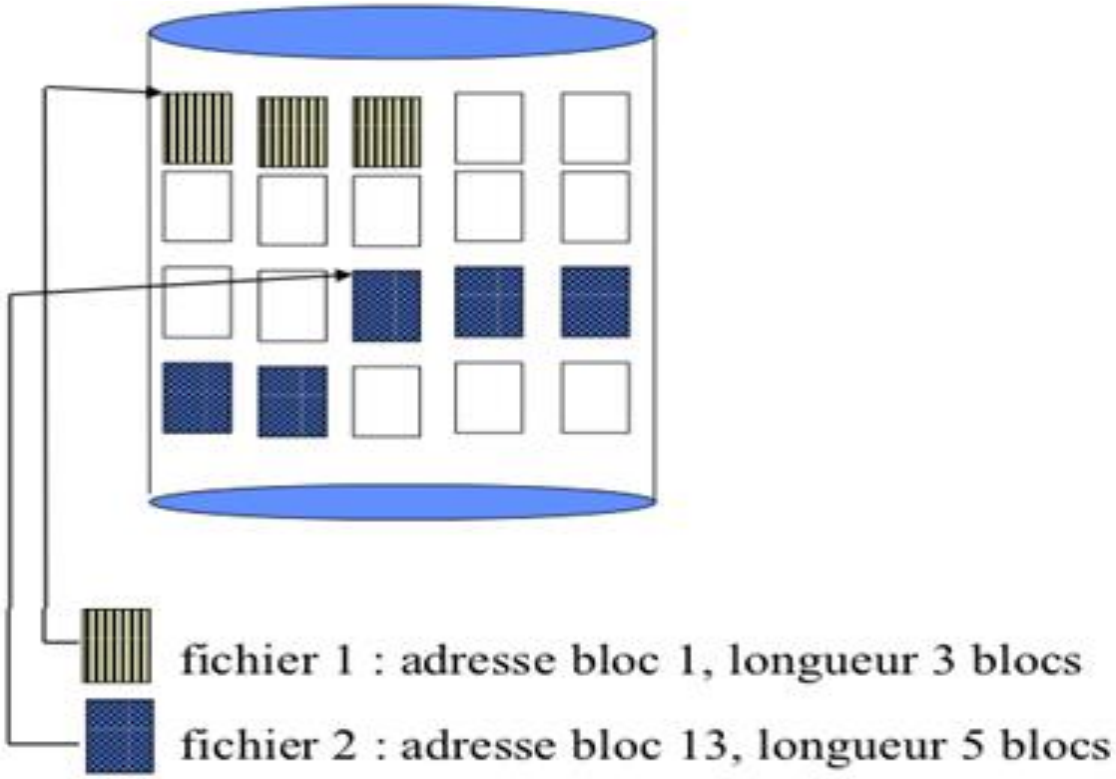


II - Allocation des fichiers dans les blocs

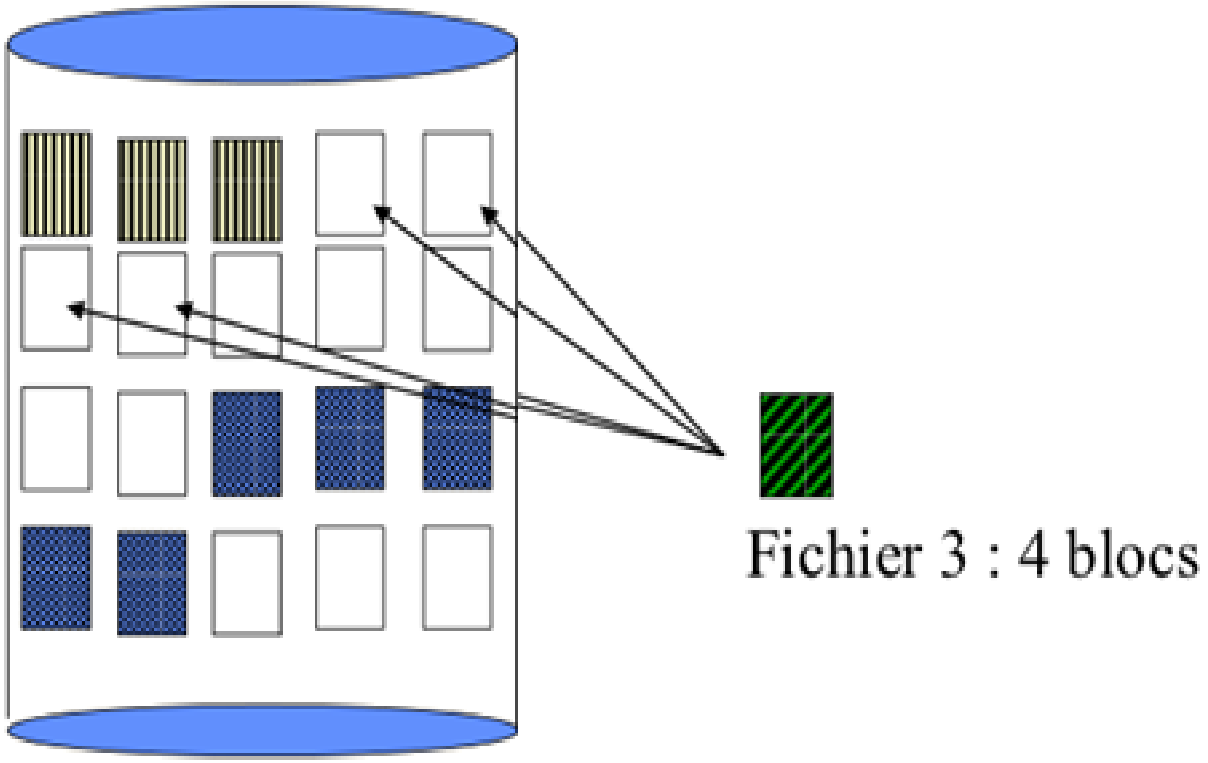
- Un fichier physique est constitué d'un ensemble de blocs physique.
- A la création d'un fichier, le SGF doit :
  - Attribuer de l'espace sur disque (c'est l'allocation) ;
  - Mémoriser son implantation et son organisation sur le disque.
  - Maintenir ces informations en cas de modifications de fichiers.
- Il existe plusieurs méthodes d'allocation des blocs physiques :
  - allocation contiguë (séquentielle simple)
  - allocation par blocs chainés
  - allocation indexée

2.1 Allocation contiguë

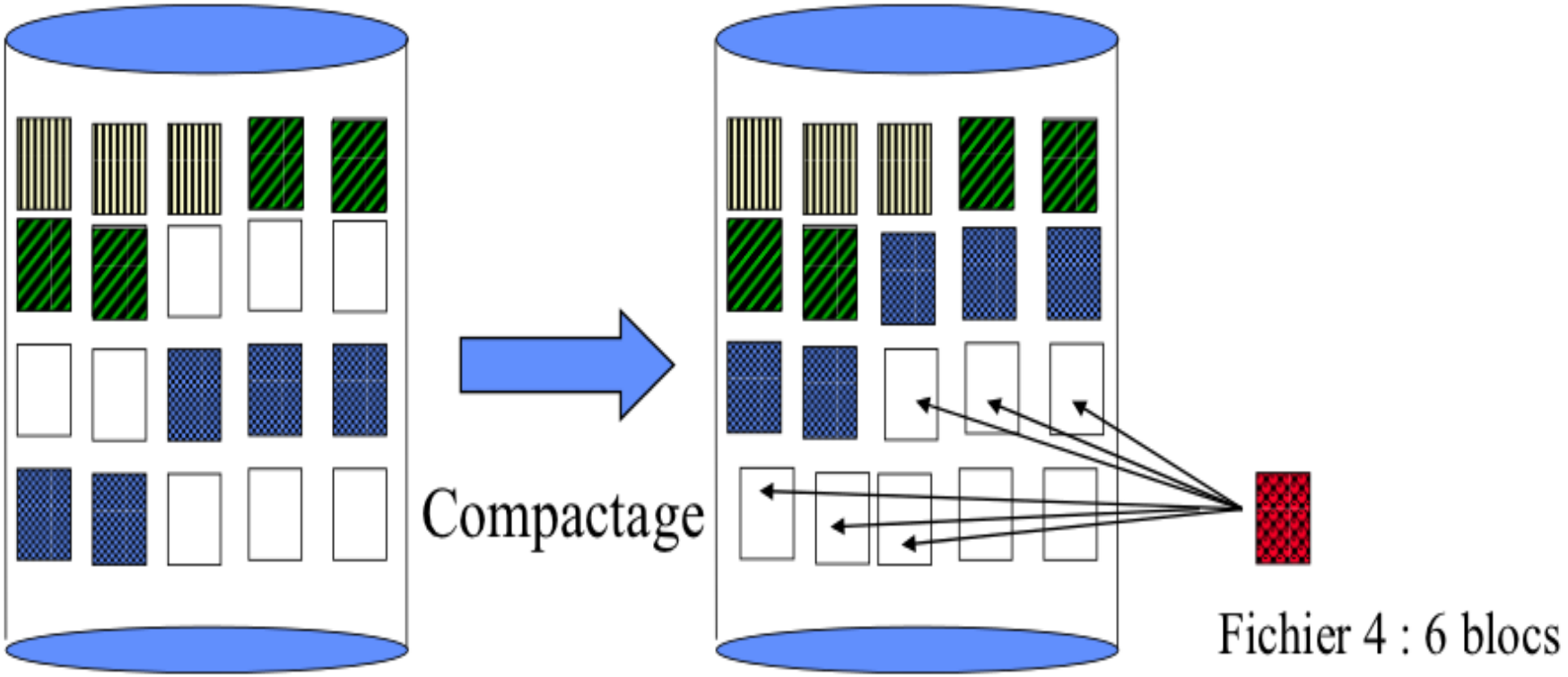
- Un fichier occupe un ensemble de blocs contigus sur le disque. Elle est bien adapté au méthodes d'accès séquentielles et directes
- Difficultés :
  - création d'un nouveau fichier
  - extension du fichier



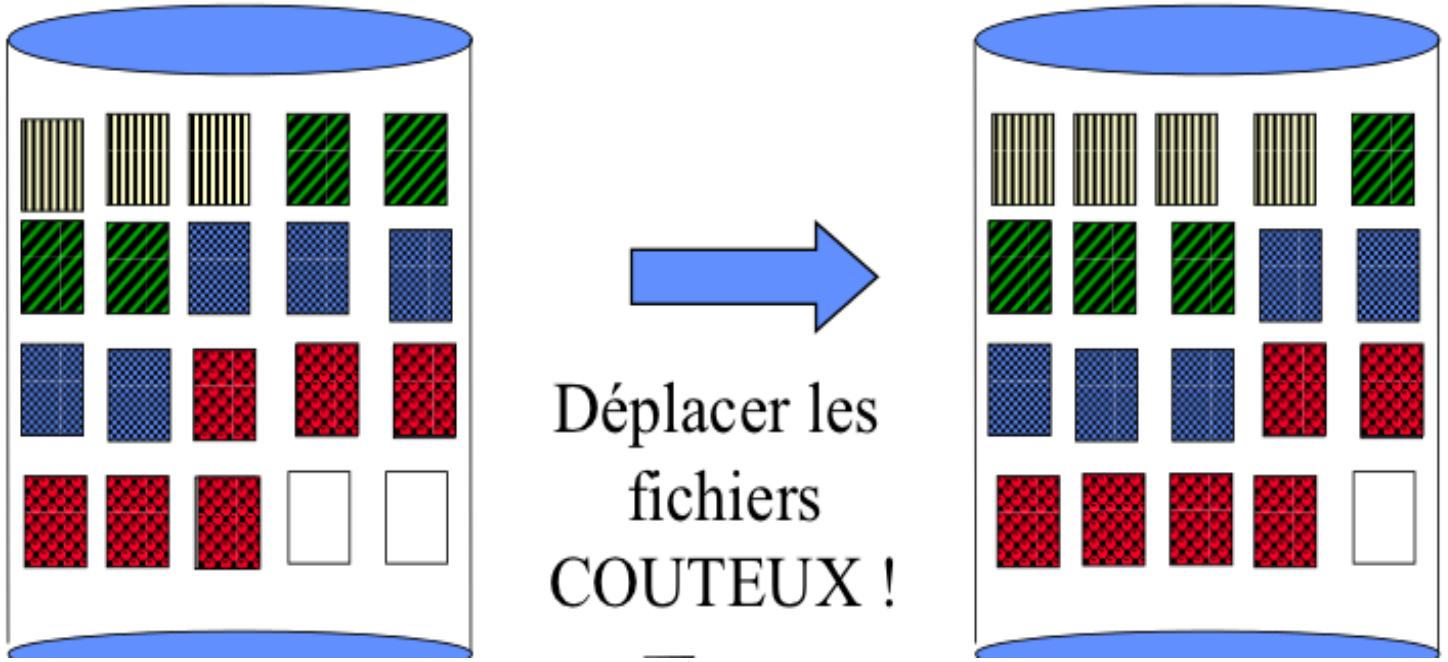
- Pour créer un nouveau fichier, il faut allouer un nombre de blocs suffisants dépendant de la taille du fichier. Par exemple si on veut stocker deux autres fichiers nommés fichier3 (4 blocs) et fichier4 (6 blocs), le système doit chercher pour chacun des fichier les blocs nécessaires pour leur stockage. Pour le fichier3, on peut lui trouver une place comme il le montre la figure.



- On ne possède pas 6 blocs contigus, donc, Comment stocker le fichier4 ?
- Pour trouver un trou suffisant, il faut utiliser la technique de compactage



- maintenant, si on veut étendre le fichier1 d'un bloc ???

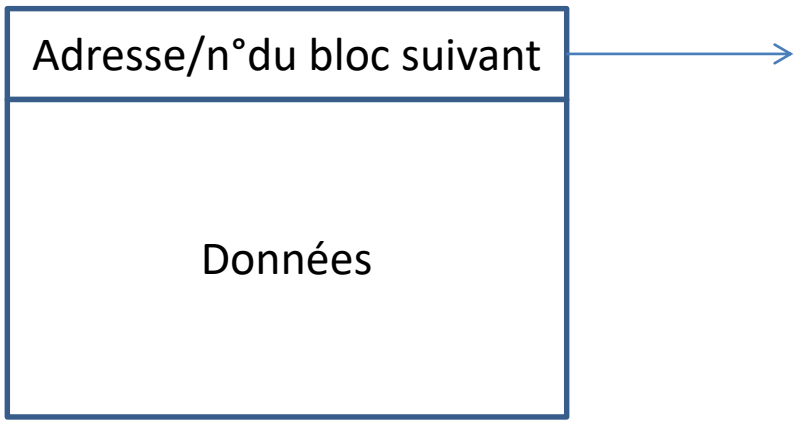


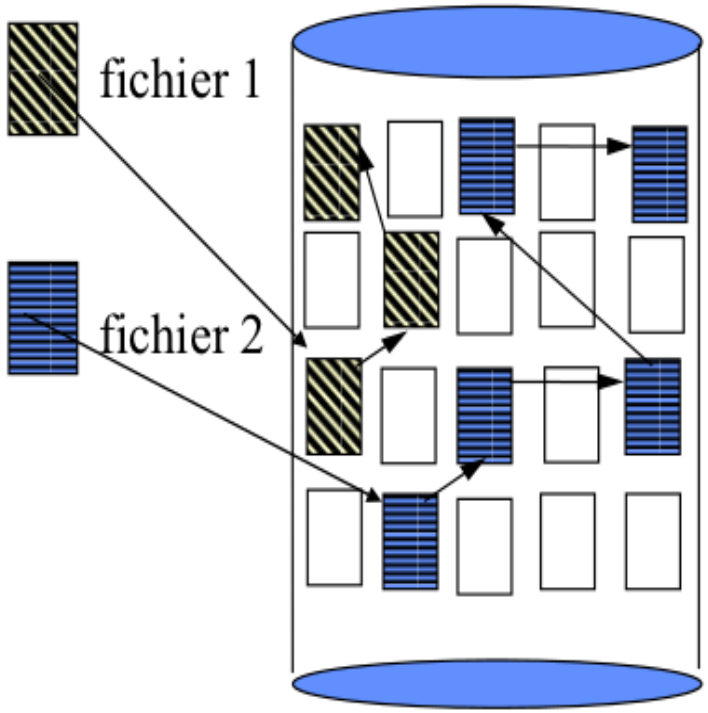
2.2 Allocation par blocs chainés

- Un fichier est constitué comme une liste chainée de blocs physiques, qui peuvent être dispersés n'importe où.
  - Extension simple du fichier : allouer un nouveau bloc et le chainer au dernier
  - Pas de fragmentation

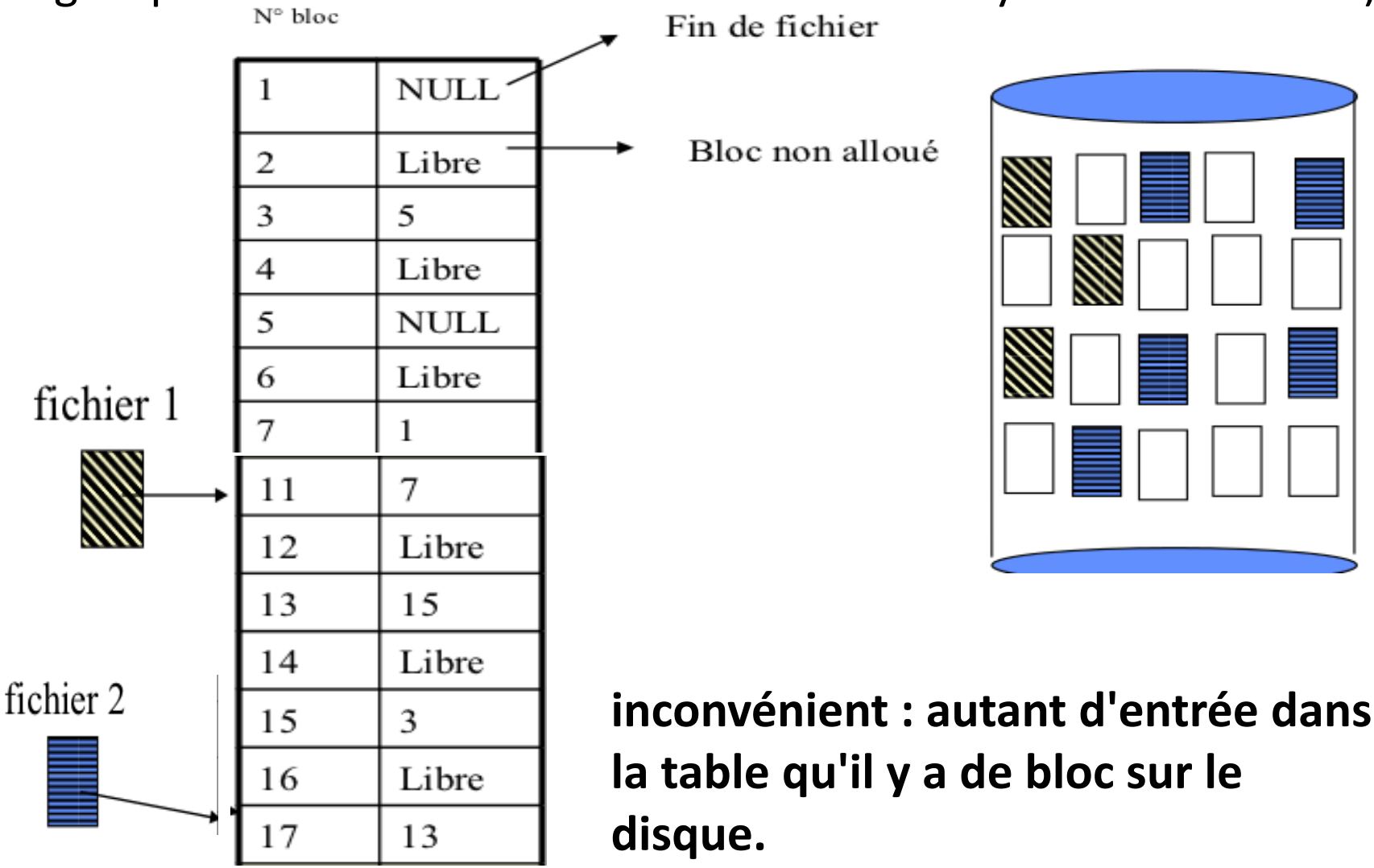
Difficultés :

- mode séquentiel seul
- le chaînage du bloc suivant occupe de la place dans un bloc





- Allocation par bloc chainée : variante
- Une table d'allocation des fichiers (File allocation table - FAT) regroupe l'ensemble des chainages. (exemple systèmes windows)



inconvénient : autant d'entrée dans la table qu'il y a de bloc sur le disque.

2.3 Allocation indexée : la solution Unix /Linux

- Les 10 premières entrées de la table contiennent l'adresse d'un bloc de données du fichier
- Bloc = 1024 octets donc 10 Ko alloués

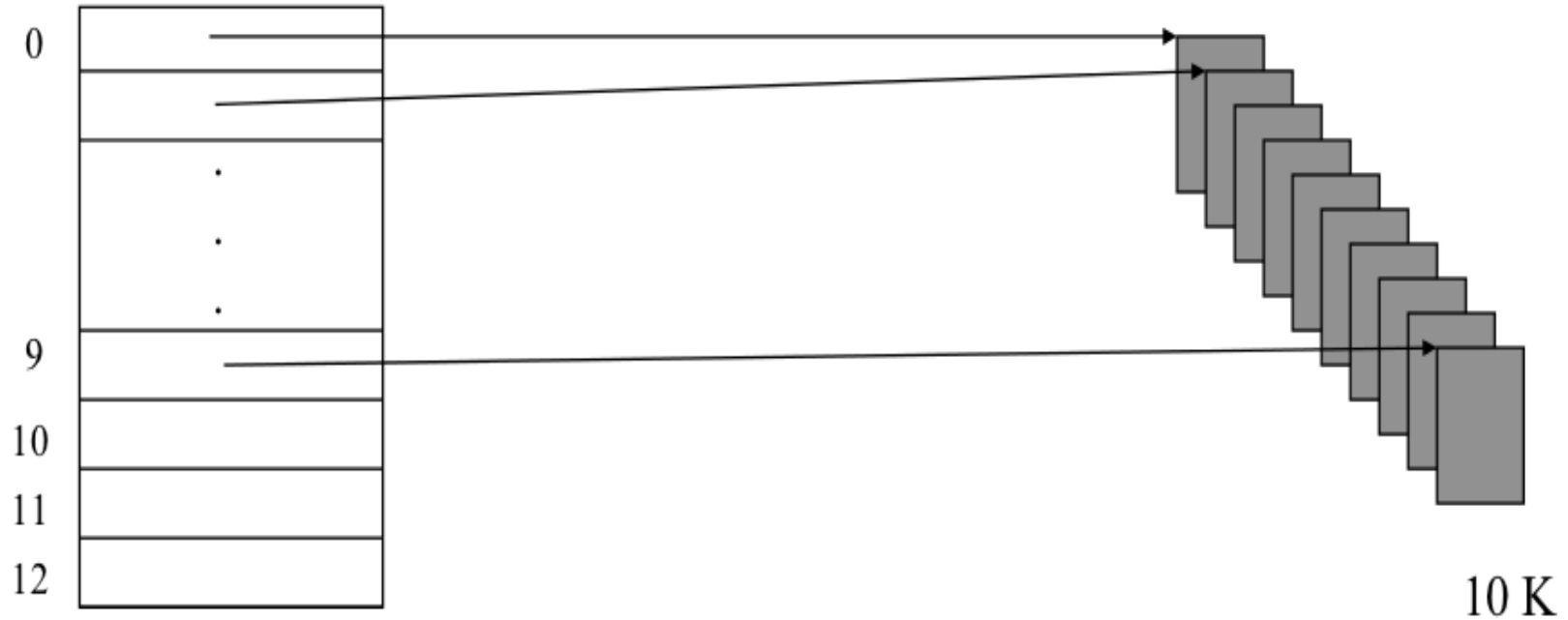
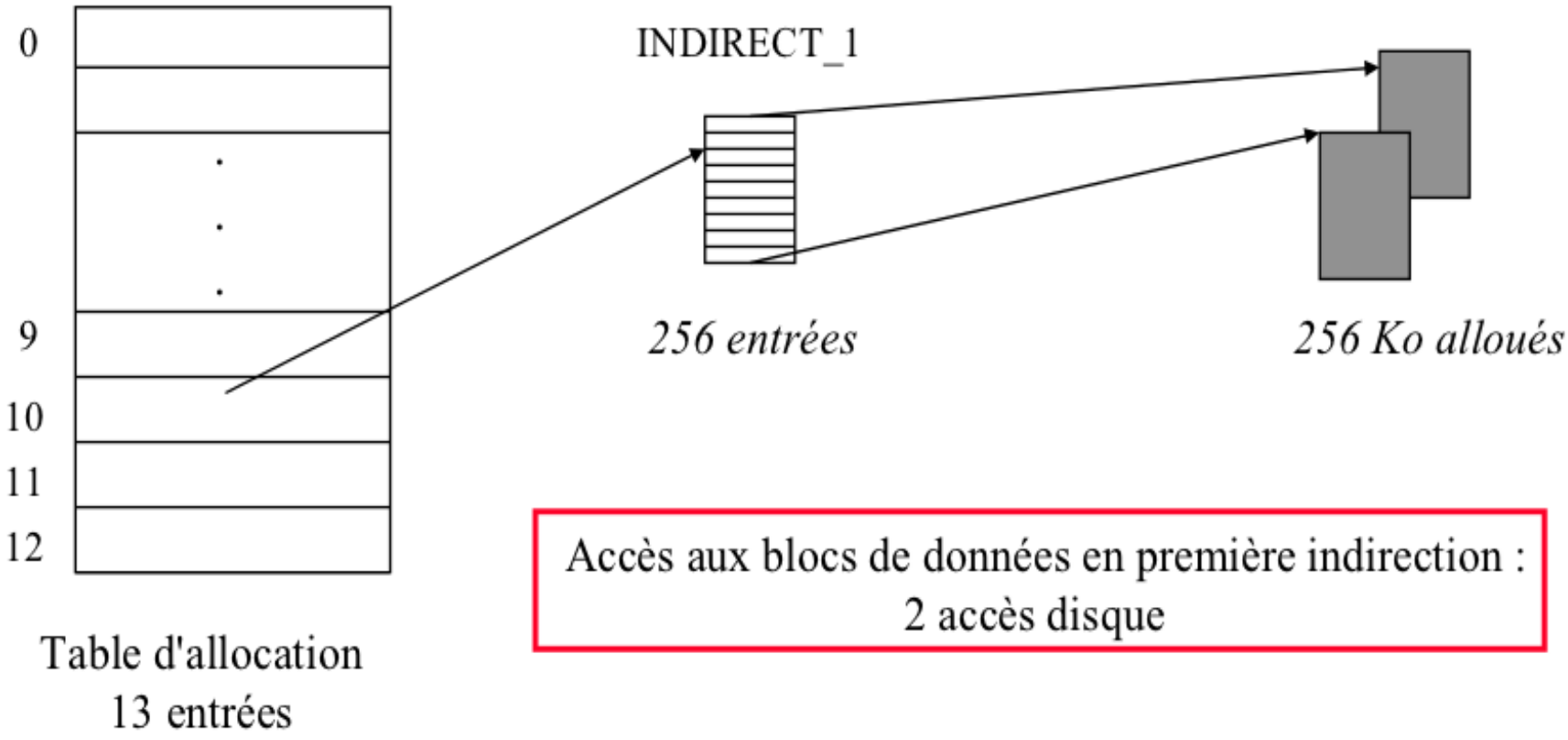


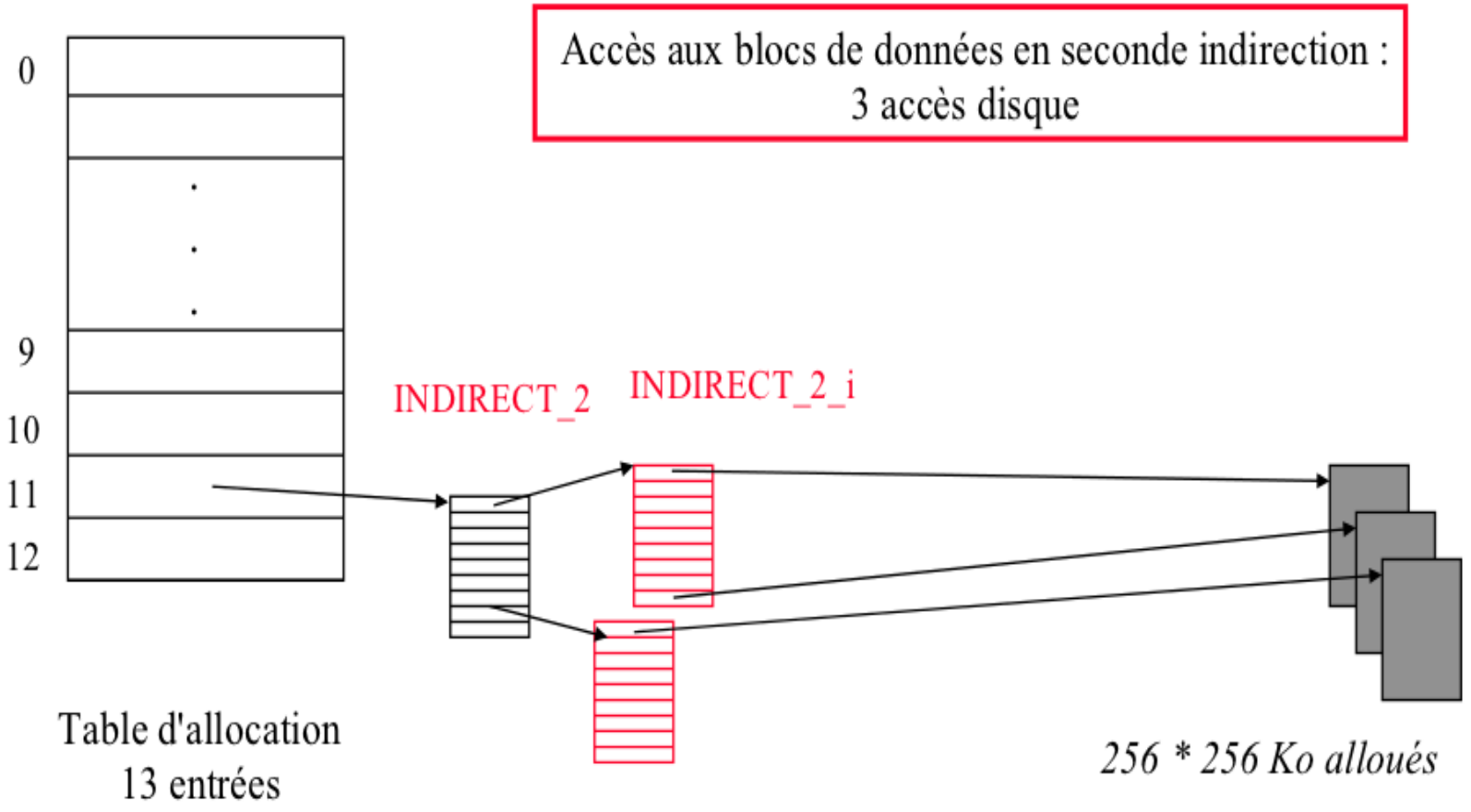
Table d'allocation
13 entrées

Accès aux blocs de données 0 à 9 :
1 accès disque

- La 11ème entrée de la table contient l'adresse d'un bloc d'index INDIRECT_1. Ce bloc d'index contient des adresses de blocs de données
- Bloc_index = 1024 octets ; adresse de bloc_données = 4 octets
  - 256 entrées dans le bloc d'index

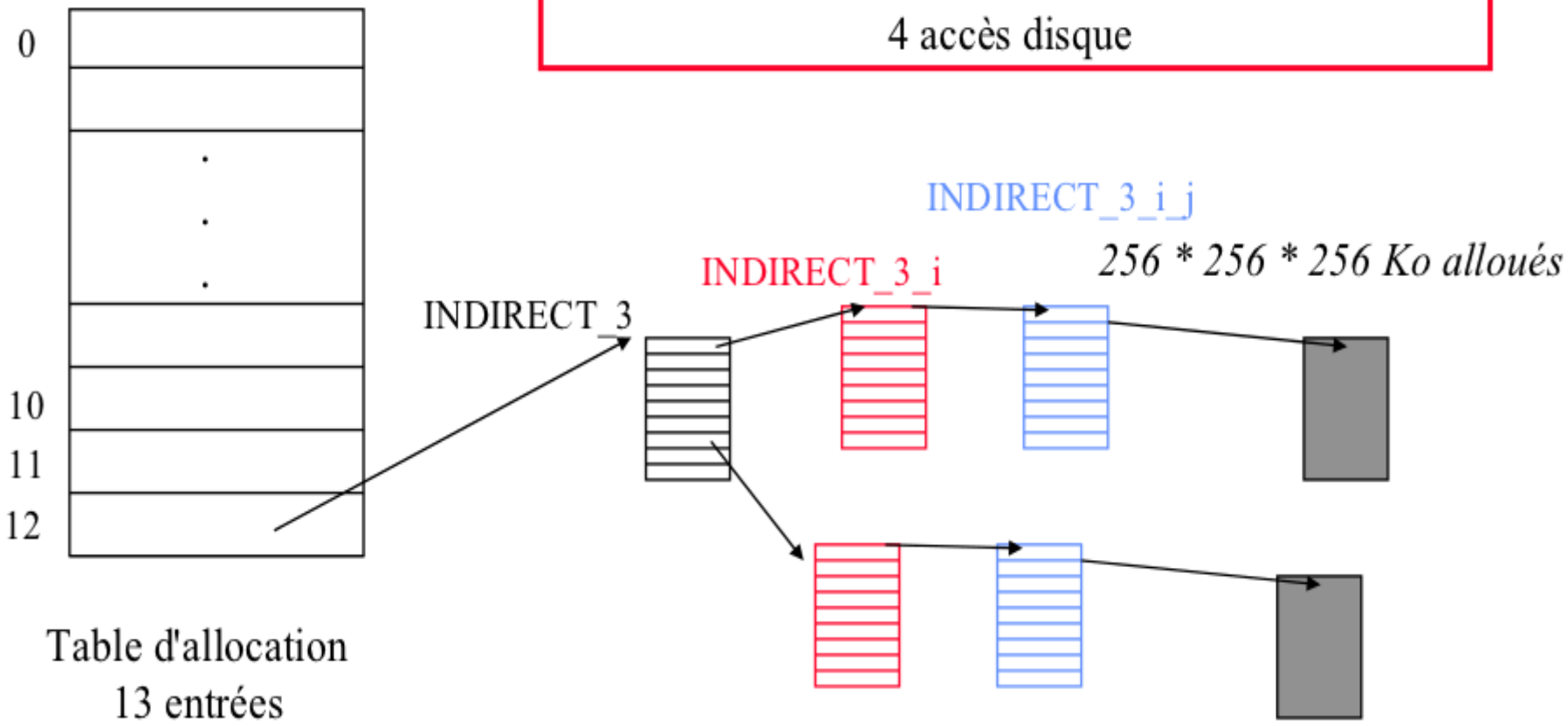


- La 12ème entrée de la table contient l'adresse d'un bloc d'index INDIRECT_2. Ce bloc d'index contient des adresses de blocs d'index INDIRECT_2_i (i de 1 à 256).
- Chaque bloc d'index INDIRECT_2_i contient des adresses de blocs de données

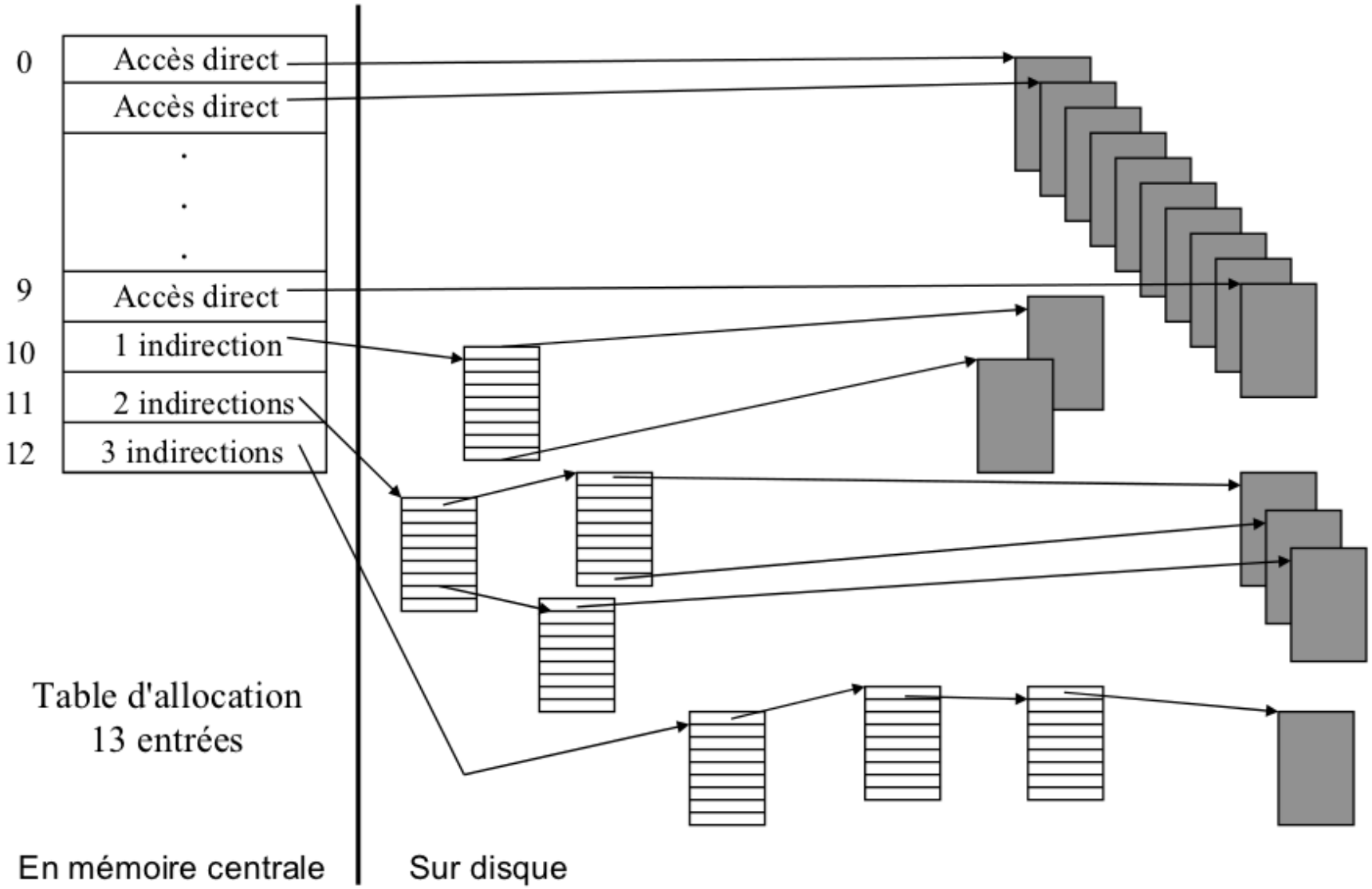


- La 13ème entrée de la table contient l'adresse d'un bloc d'index INDIRECT_3. Ce bloc d'index contient des adresses de blocs d'index INDIRECT_3_i.
- Chaque bloc d'index INDIRECT_3_i contient des adresses de blocs d'index INDIRECT_3_i_j.
- Chaque bloc d'index INDIRECT_3_i_j contient des adresses de blocs de données
- Bloc = 1024 octets ; adresse de bloc = 4 octets 256 entrées dans le bloc d'index
- (i et j évolue de 1 à 256)

Accès aux blocs de données en troisième indirection :
4 accès disque



- On peut représenter les différents niveaux d'indirections de la façon suivante :

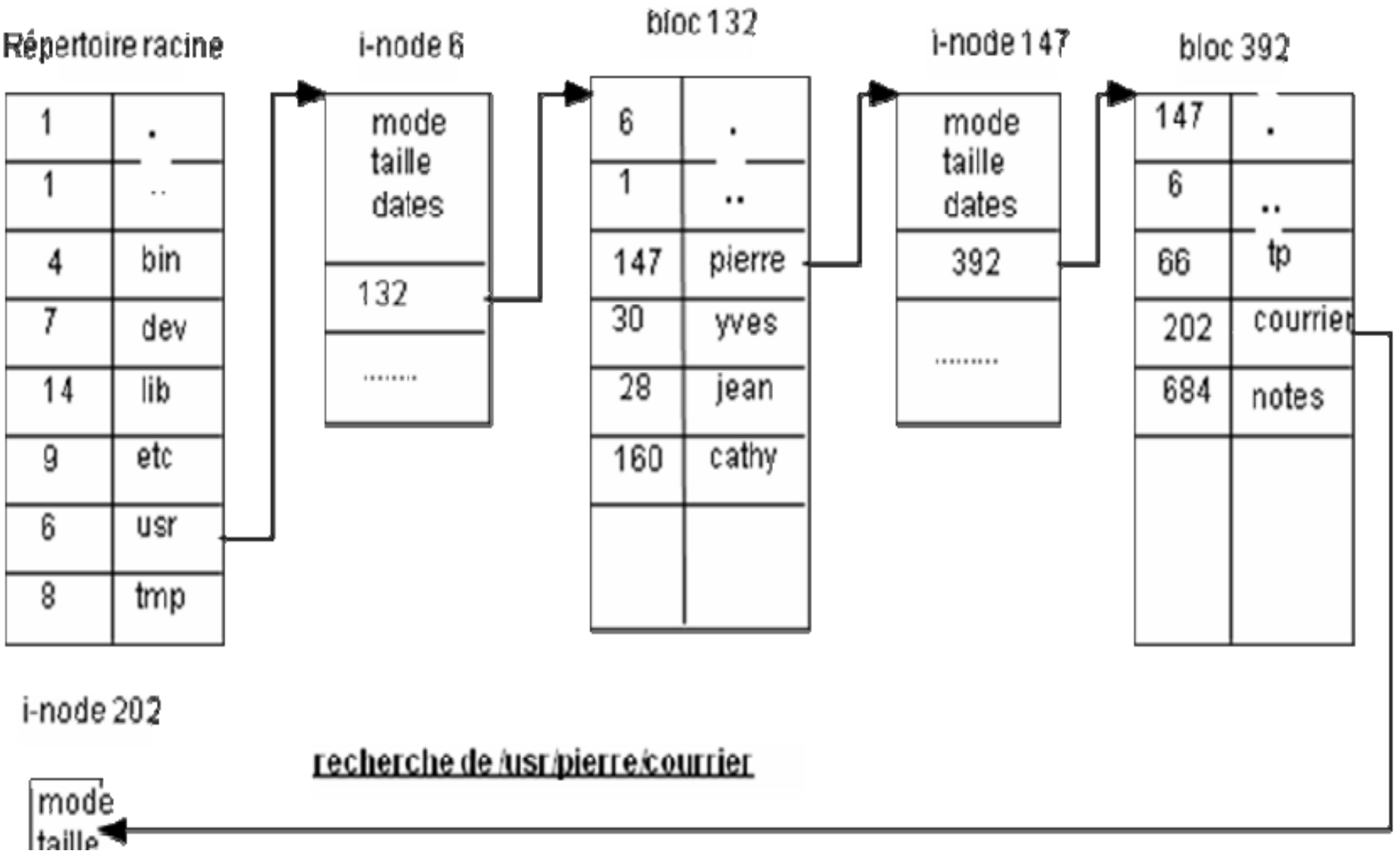


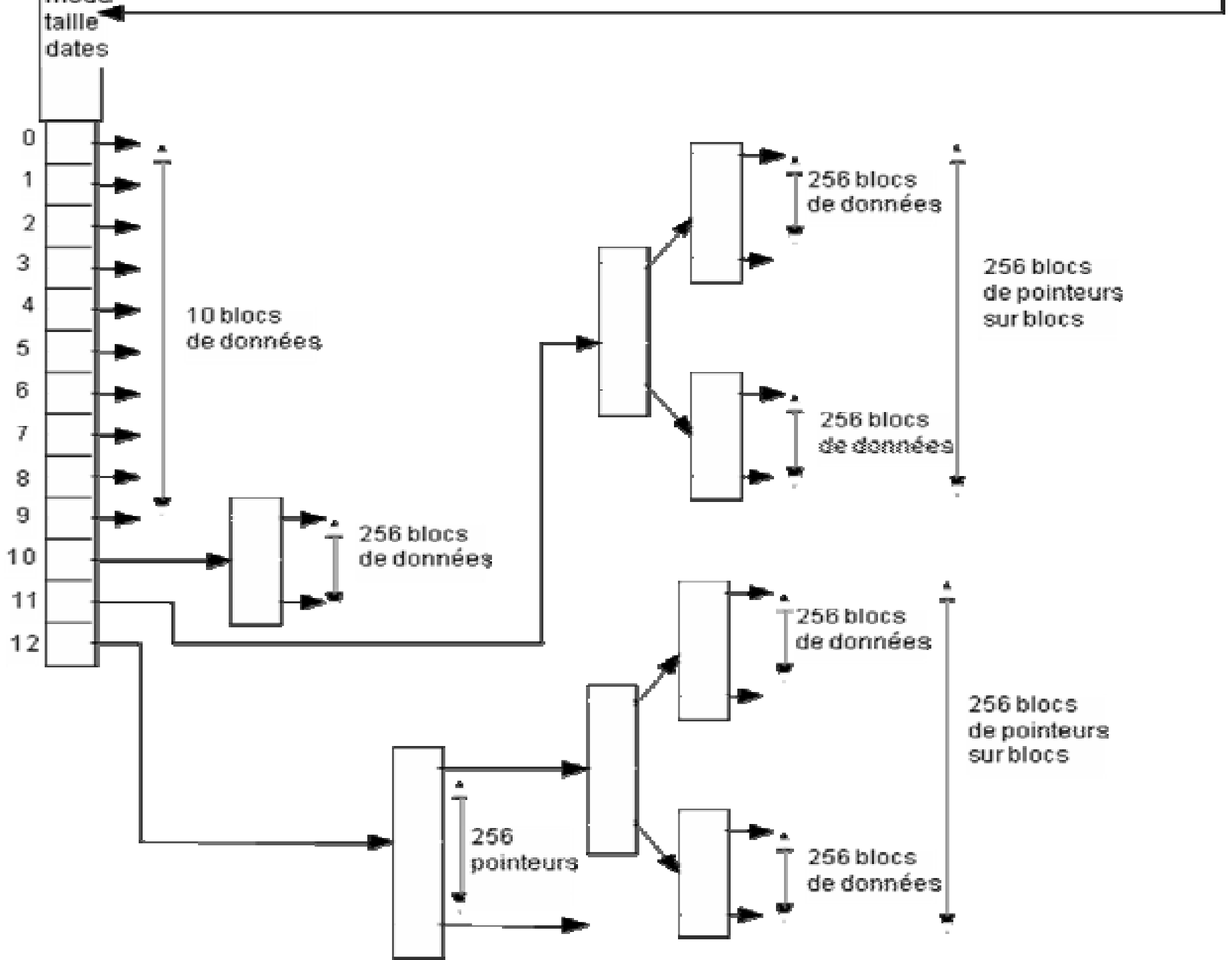
- Allocation de l'espace disque pour un fichier de 526 Ko donc 526 blocs
  - 10 blocs en accès direct
  - 256 blocs de données pointés par le bloc index INDIRECT 1
  - restent 526 – 10 – 256 = 260 blocs . Tous ces blocs sont pointés à partir du bloc d'index INDIRECT_2. deux blocs d'index INDIRECT_2_1 et INDIRECT_2_2 sont nécessaires à ce niveau .
- Si on suppose que la taille d'un bloc est de 1Ko, un fichier peut avoir la taille maximale suivante : 10 x 1Ko + 256 x 1Ko + 256 x 256 x 1Ko + 256 x 256 x 256 x 1Ko, ce qui donne en théorie plus de 16Go

Structure d'un I-Node

- Cette structure possède plusieurs entrées, elle permet au système de disposer d'un certain nombre de données sur le fichier :
- la taille,
- l'identité du propriétaire et du groupe : un fichier en Unix est crée par un propriétaire, qui appartient à un groupe,
- Les droits d'accès : pour chaque fichier, Unix définit trois droits d'accès (lecture (r), écriture (w) et exécution (x)) pour chaque classe d'utilisateurs (trois types d'utilisateur {propriétaire, membre du même groupe que le propriétaire, autres}). Donc à chaque fichier, Unix associe neuf droits,
- les dates de création, de dernière consultation et de dernière modification,
- le nombre de références existant pour ce fichier dans le système,
- les dix premiers blocs de données,

- d’autres entrées contiennent l’adresse d’autres blocs (on parle alors de bloc d’indirection) :
  - une entrée pointe sur un bloc d'index qui contient 256 pointeurs sur bloc de données (simple indirection)
  - Une entrée pointe sur un bloc d'index qui contient 256 pointeurs sur bloc d'index dont chacun contient 256 pointeurs sur bloc de données (double indirection)
  - Une entrée pointe sur un bloc d'index qui contient 256 pointeurs sur bloc d'index dont chacun contient 256 pointeurs sur bloc d'index dont chacun contient 256 pointeurs sur bloc de données (triple indirection)
- La structure d’I-Node est conçue afin d’alléger le répertoire et d’en éliminer les attributs du fichier ainsi que les informations sur l’emplacement des données.


Répertoire racine
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
i-node 6
mode
taille
dates
132
.....
bloc 132
6 .
1 ..
147 pierre
30 yves
28 jean
160 cathy
i-node 147
mode
taille
dates
392
.....
bloc 392
147 .
6 ..
66 tp
202 courrier
684 notes
i-node 202
mode
taille
recherche de /usr/pierre/courrier


taille
dates
0
1
2
3
4
5
6
7
8
9
10
11
12
10 blocs
de données
256 blocs
de données
256 blocs
de données
256 blocs
de pointeurs
sur blocs
256 blocs
de données
256 blocs
de données
256 blocs
de pointeurs
sur blocs
256
pointeurs
256 blocs
de données

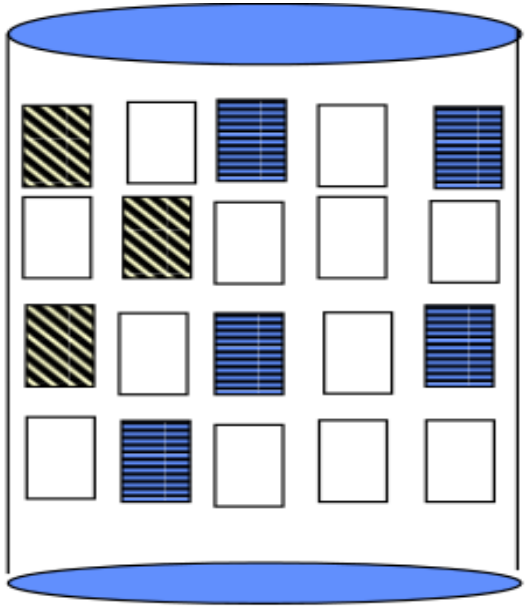
II- Gestion de l'espace libre

- Le système maintient une liste d'espace libre, qui mémorise tous les blocs disque libres (non alloués)
- Création d'un fichier : recherche dans la liste d'espace libre de la quantité requise d'espace et allocation au fichier : l'espace alloué est supprimé de la liste
- Destruction d'un fichier : l'espace libéré est intégré à la liste d'espace libre
- Il existe différentes représentations possibles de l'espace libre
  - vecteur de bits
  - liste chainée des blocs libres

Gestion de l'espace libre par un vecteur de bits

La liste d'espace libre est représentée par un vecteur binaire, dans lequel chaque bloc est figuré par un bit.

- **Bloc libre : bit à 1**
- **Bloc alloué : bit à 0**

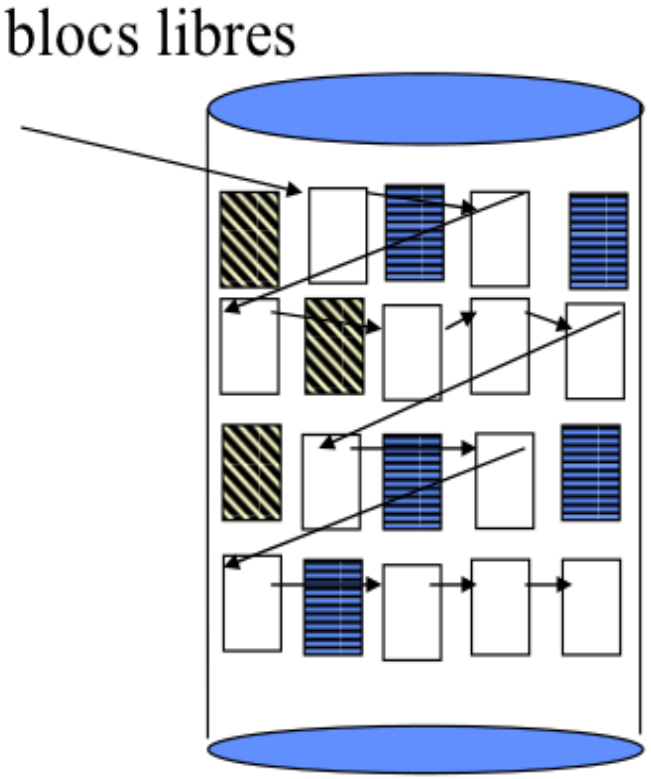


01010101110101010111

- pour cette technique, il est facile de trouver n blocs libres consécutifs . C'est la technique utilisé dans les Système Macintosh

Gestion de l'espace libre par liste chainée

- **La liste d'espace libre est représentée par une liste chainée des blocs libres**



- **Parcours de la liste**
  - **couteux**
- **Difficile de trouver un groupe de blocs libres**
- **Variante par comptage**

Gestion de l'espace libre par liste chainée: variante avec comptage

- **Le premier bloc libre d'une zone libre contient l'adresse du premier bloc libre dans la zone suivante et le nombre de blocs libres dans la zone courante.**