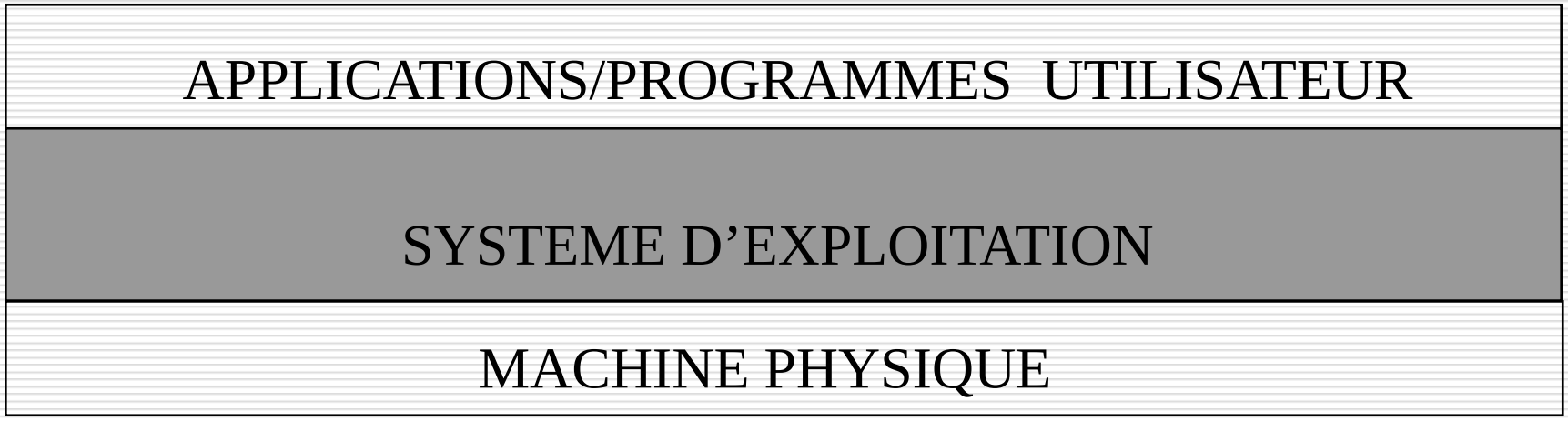
le cnam

Cours de Révision
Les systèmes d'exploitation

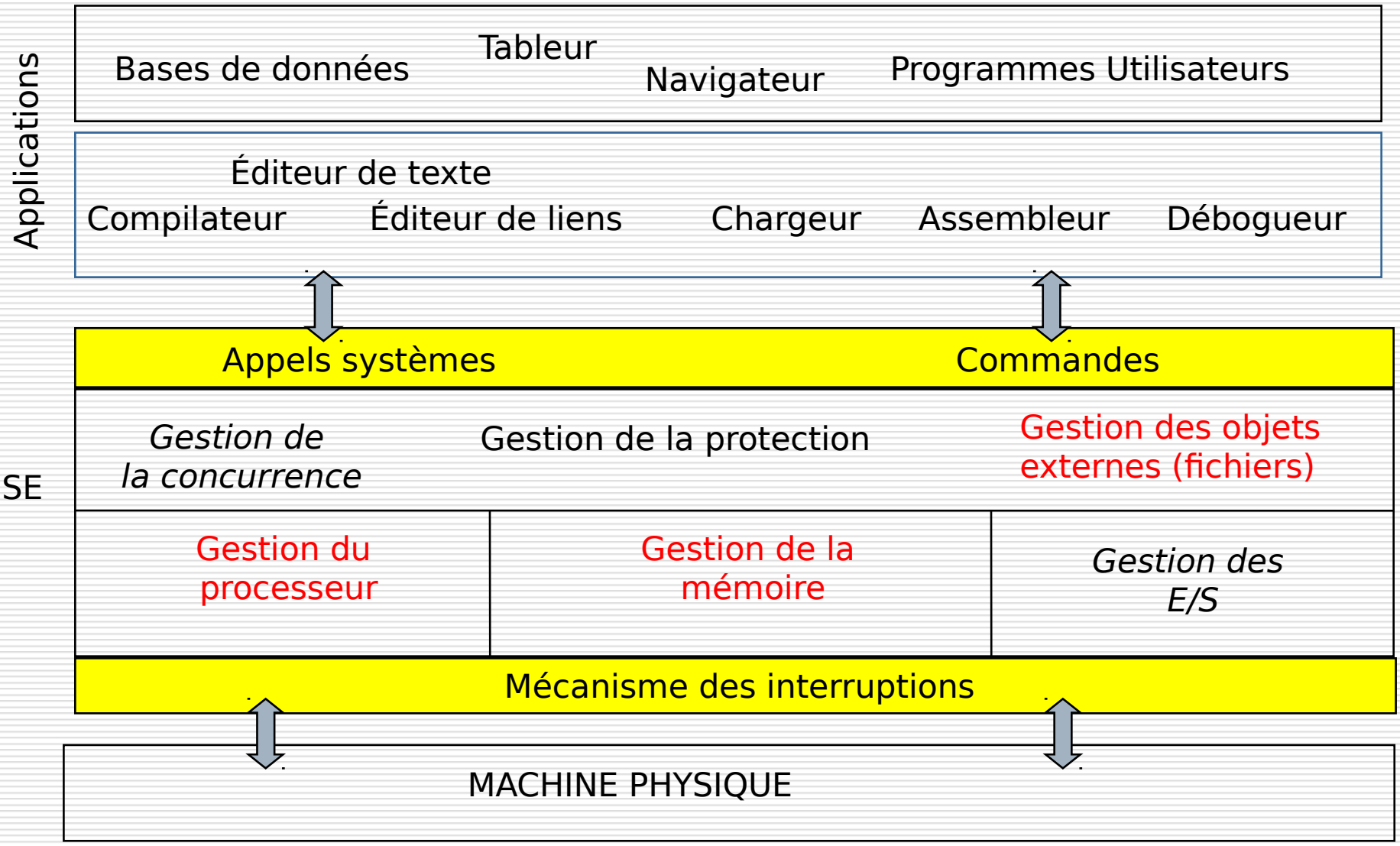
Définition d'un système d'exploitation



Ensemble de programmes qui réalisent l'interface entre le matériel de l'ordinateur et les utilisateurs.

→ Il a en charge l'exploitation de la machine pour en faciliter l'accès, le partage et pour l'optimiser.

Fonctions d'un système d'exploitation

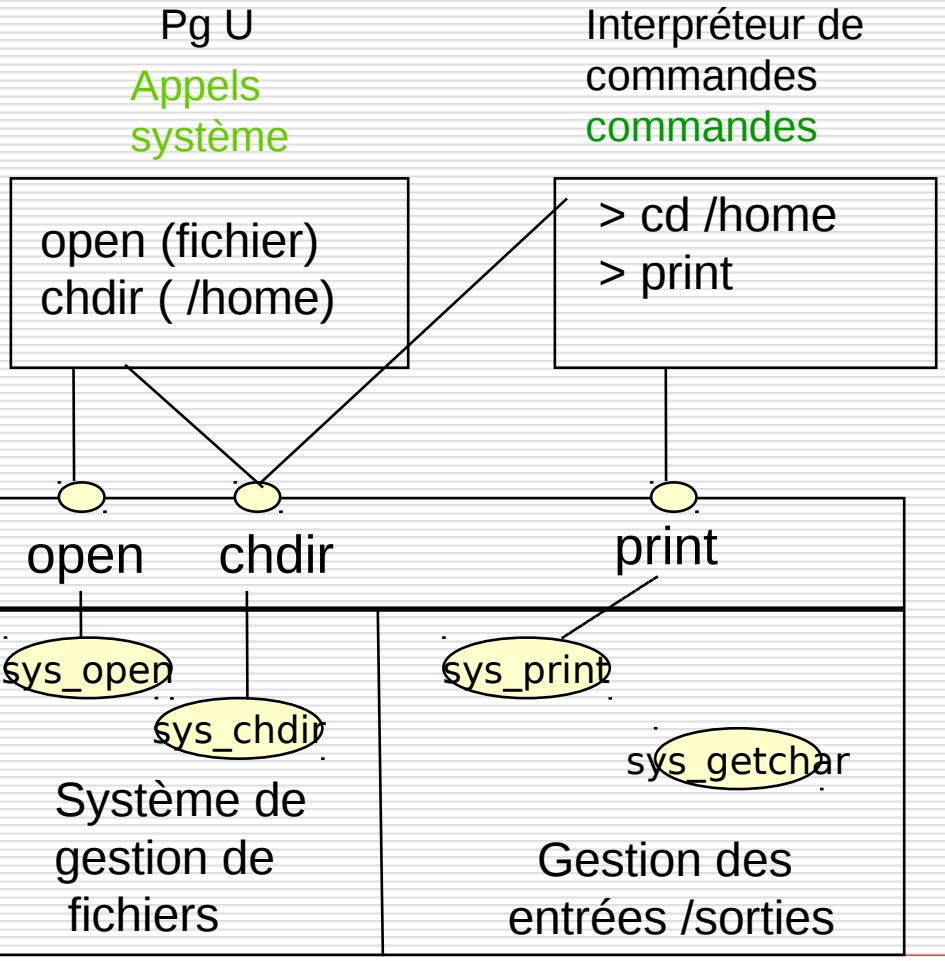


Notions de base

- Modes d'exécution
- Interruptions logicielles et matérielles
- Chargement du système

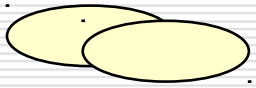
Appels système et commandes

Les fonctionnalités du système d'exploitation sont accessibles par le biais des **commandes** ou des **appels système**



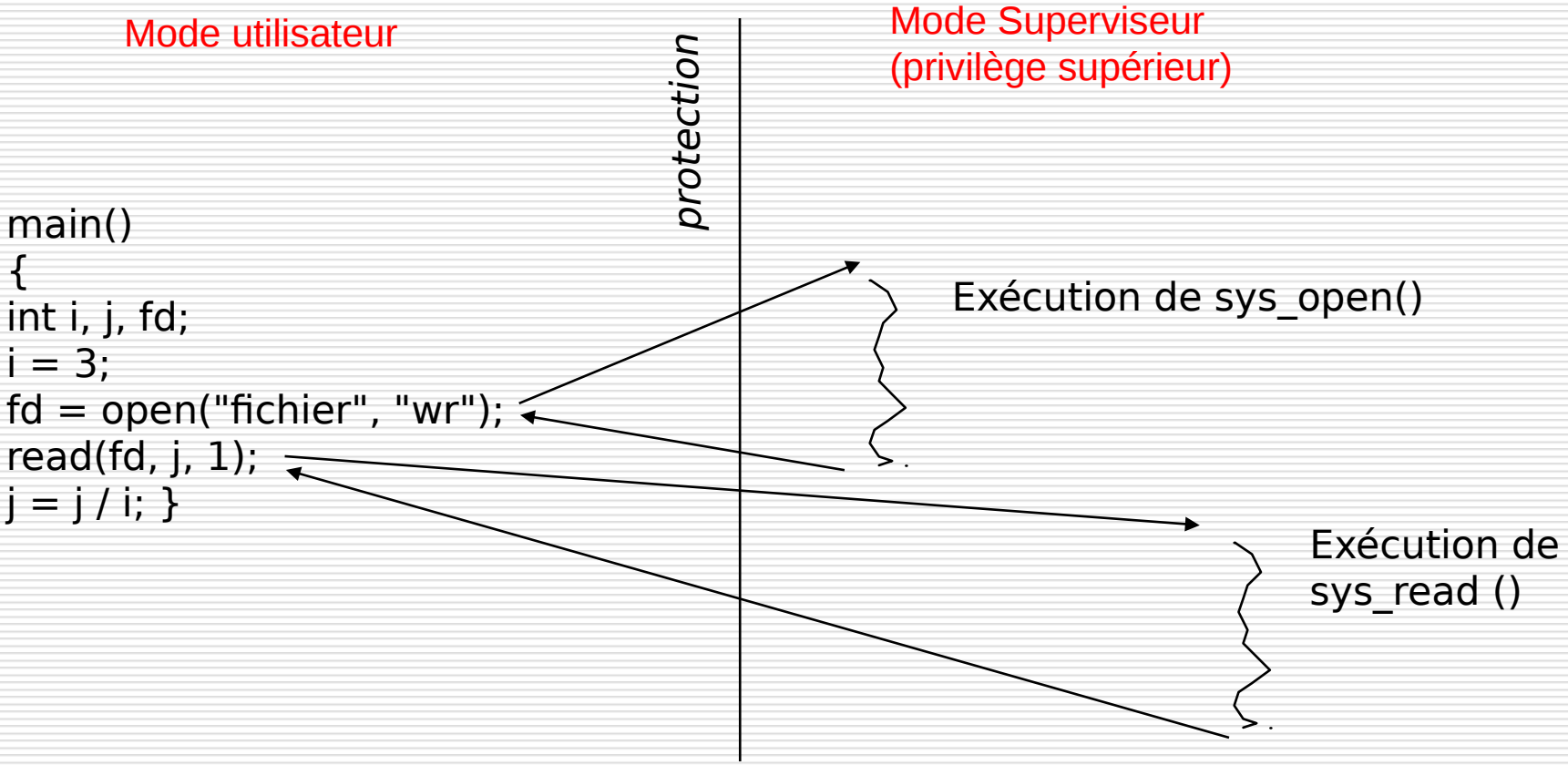
Interface d'appel

SE : ensemble de fonctions



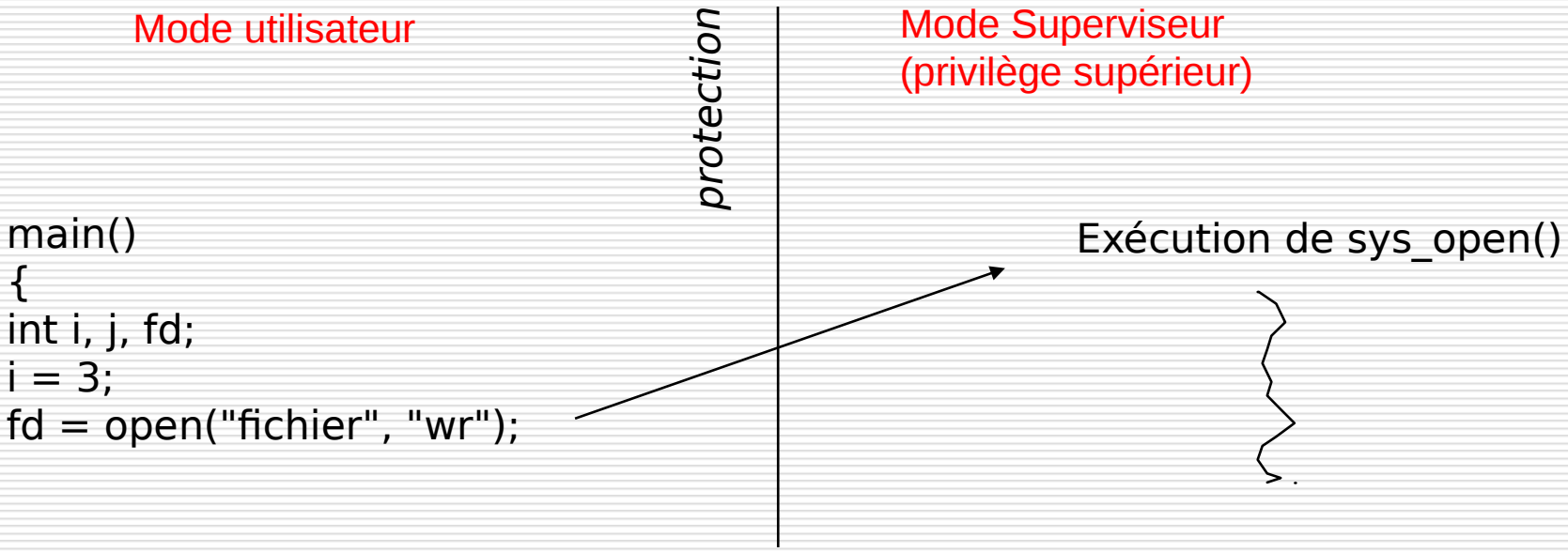
Modes d'exécutions

Lors de l'appel a une fonction du système, le programme utilisateur passe d'un **mode d'exécution dit utilisateur** à un **mode d'exécution dit superviseur**.



Modes d'exécutions

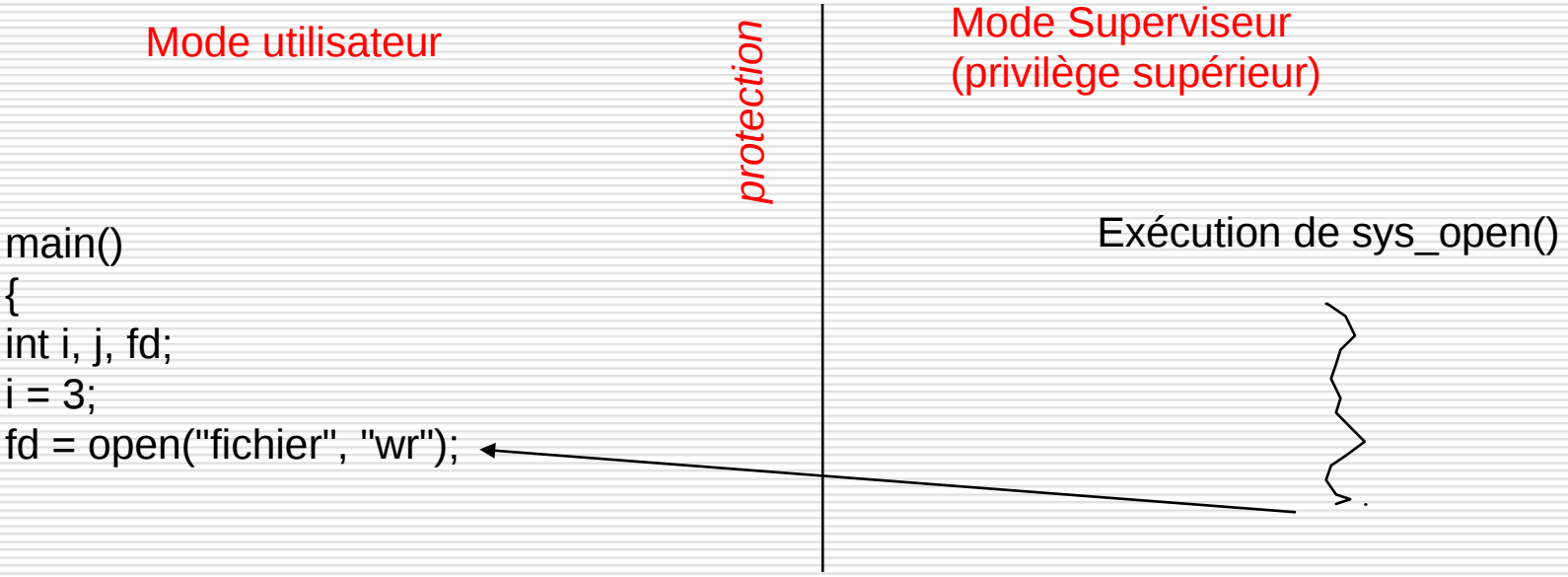
Le passage du mode utilisateur au mode superviseur s'accompagne d'opérations de **commutation de contexte : sauvegarde de contexte utilisateur**



sauvegarde CO, PSW utilisateur
chargement CO ← adresse de la fonction open
chargement PSW ← mode superviseur

Modes d'exécutions

Le passage du mode superviseur au mode utilisateur s'accompagne d'opérations de **commutation de contexte : restitution de contexte utilisateur**



restitution du contexte utilisateur
chargement CO ← CO sauvegardé
chargement PSW ←PSW sauvegardé

Quand passer en mode superviseur depuis un programme utilisateur ?

le cnam

Mode Superviseur

Mode utilisateur

protection

```
main()
{
int i, j, fd;
i = 0;
fd = open("fichier", "wr");
read(fd, j, 1);
j = j / i; }
```

Exécution de sys_open()
APPELS SYSTEME

TRAPPE
erreur irrécouvrable
arrêt du programme

Interruptions logicielles

IRQ
Exécution du
traitant d'irq Horloge

Interruptions matérielles

IT HORLOGE

MATERIEL

La chaîne de production de programme

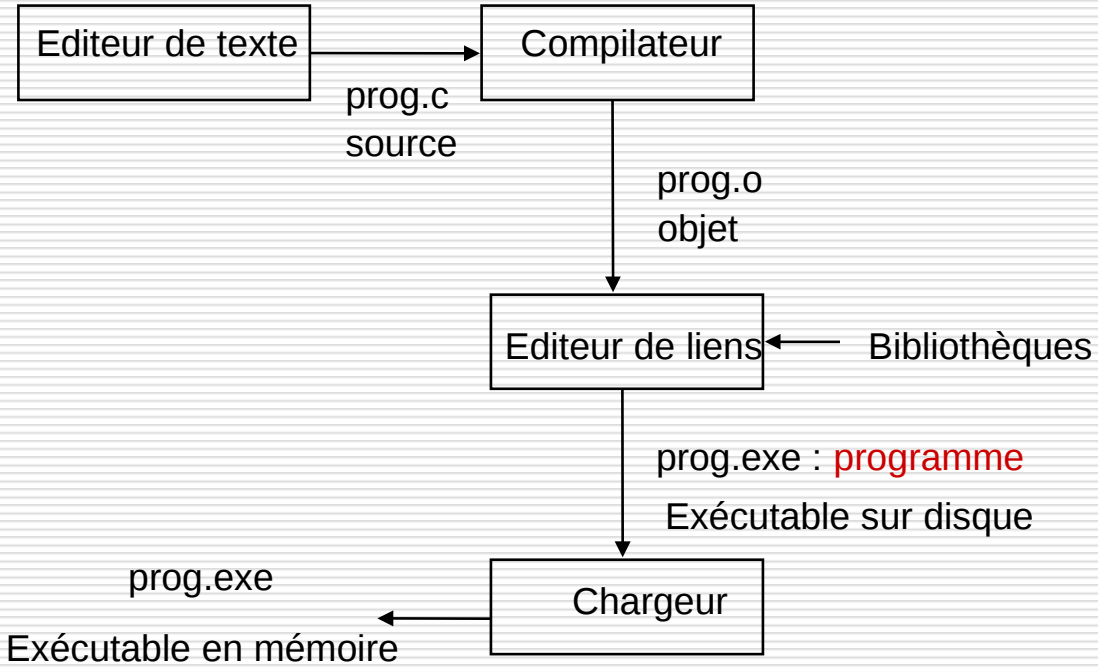
Du programme source au processus :

- Compilation
- Éditions des liens et chargement
- L'utilitaire Make

La chaîne de production de programmes

Cette chaîne est l'ensemble des étapes nécessaires à la construction d'un programme exécutable à partir d'un fichier source :

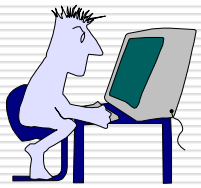
- La compilation
- L'édition des liens
- Le chargement



Les niveaux de langage de programmation

le cnam

Programme en langage haut niveau
(indépendant machine physique)



```
While (x > 0)
do
      y := y + 1;
      x := x - 1;
done;
```

COMPILATEUR

Programme en langage d'assemblage
(très proche du langage machine)

```
loop : add  R1, 1
       sub R2, 1
       jmpP loop
```

ASSEMBLEUR

Programme en binaire
(langage machine)

```
1000 : 0001 0000 0001 000000000001
       0010 0000 0010 000000000001
       0110 0000000000000001000
```

Rôle du compilateur

- Un compilateur traduit un **programme source** écrit en langage de haut niveau en un **programme objet** en langage de bas niveau.

- Le compilateur est lui-même un programme important et volumineux.

- Le travail du compilateur se divise en plusieurs phases :

  (1) analyse lexicale (recherche des mots-clés)

  (2) analyse syntaxique (vérification de la syntaxe)

  (3) analyse sémantique (vérification de la sémantique)

  (4) génération du code objet

Formalisation de la syntaxe du langage

- Il faut exprimer la syntaxe du langage. On utilise pour cela la **notation de BACKUS-NAUR (BNF)**

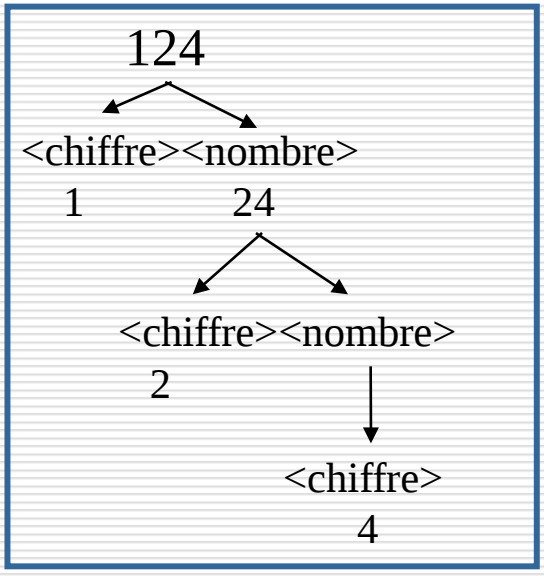
- < objet du langage > **::=** <objet du langage> | symbole

  - | représente une alternative
  - < > entoure les objets du langage

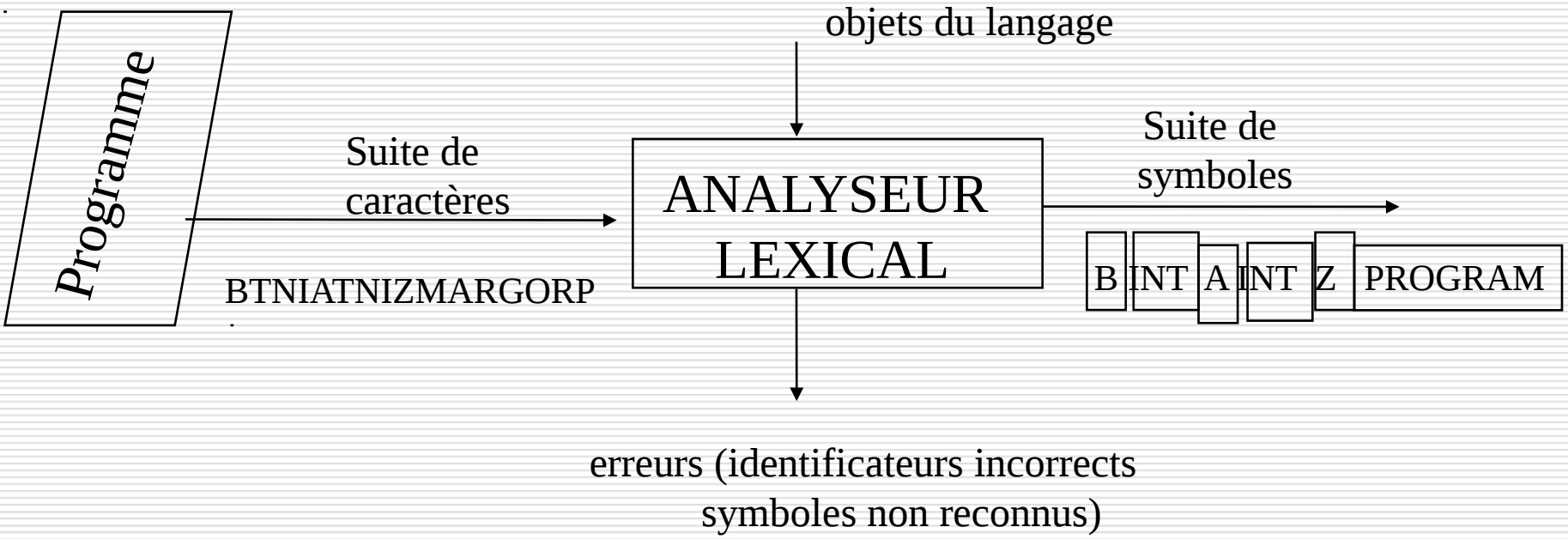
Exemple :

<nombre> ::= <chiffre> | <chiffre><nombre>

<chiffre> ::= 0 | 1| 2 | 3 | 4| 5 | 6 | 7 | 8 | 9



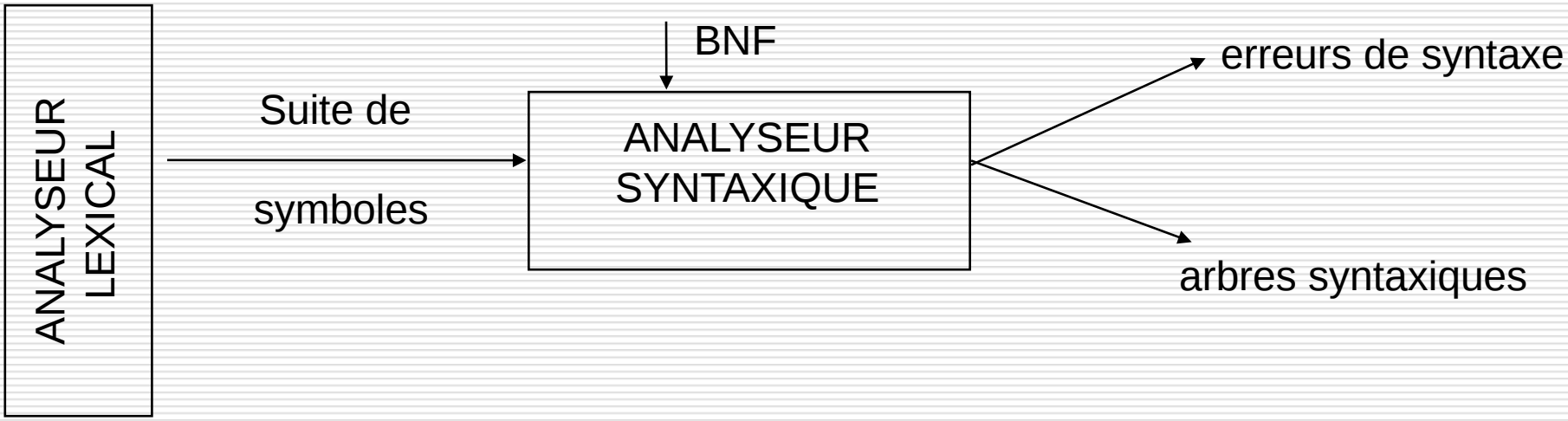
Analyse lexicale



Rôle de l'analyse lexicale

- reconnaître dans la suite de caractères que constitue un programme les objets du langage
- éliminer le "superflu" (espaces, commentaires)

Analyse syntaxique

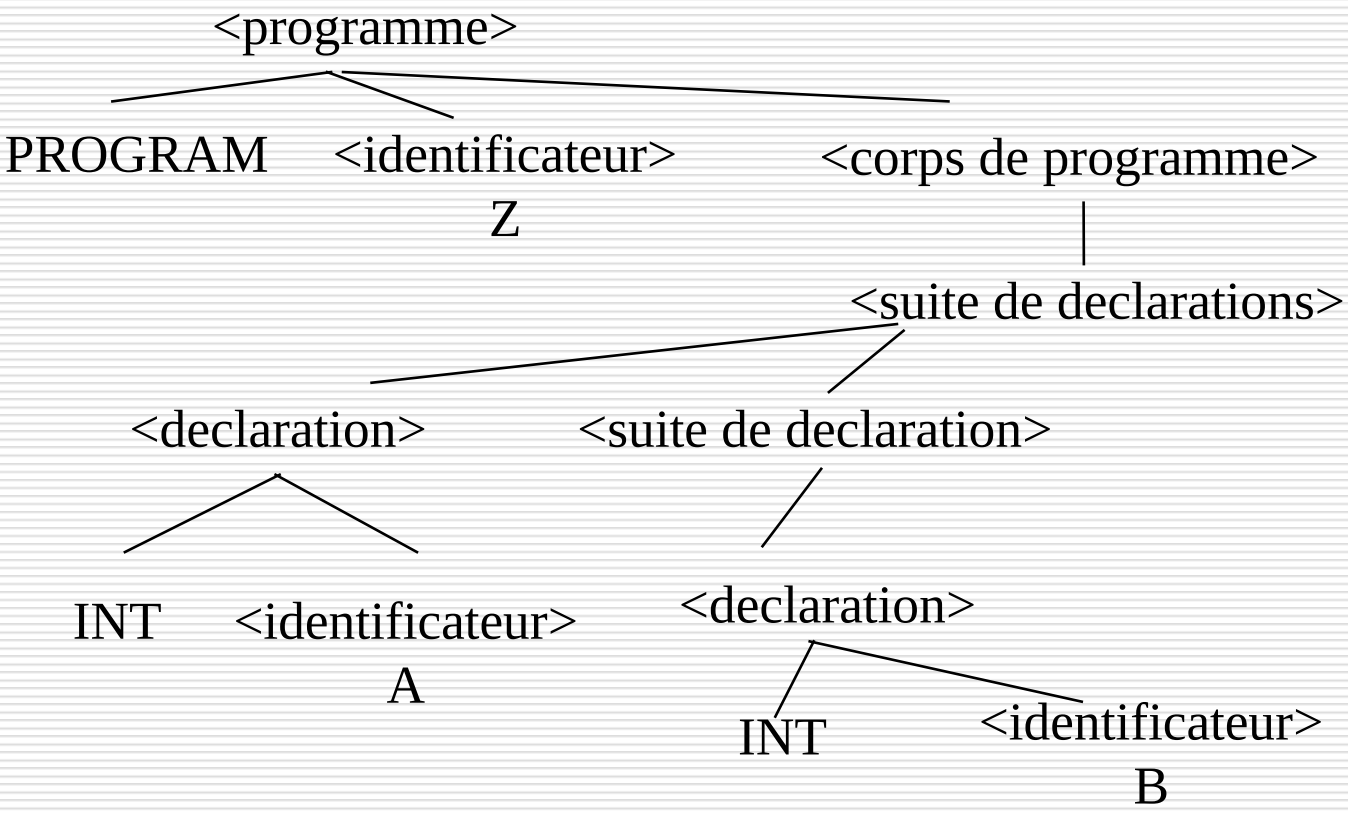


Rôle de l'analyse syntaxique :

reconnaître si la suite de symboles issue de l'analyse lexicale respecte la syntaxe du langage

➔ construction de **l'arbre syntaxique** correspondant au programme analysé

Arbre Syntaxique : Exemple




```
PROGRAM Z
  INT A
  INT B
DEBUT
  A := 5
  B := A * 2
FIN
```

<programme> ::= PROGRAM <identicateur><corps de programme>
<corps de programme> ::= <suite de declarations> DEBUT <suite d'affectations> FIN
<suite de declarations> ::= <declaration> | <declaration><suite de declarations>
<declaration> ::= INT <identificateur>

Analyse sémantique

Rôle de l'analyse sémantique :

Contrôler la signification des différentes phrases du langage

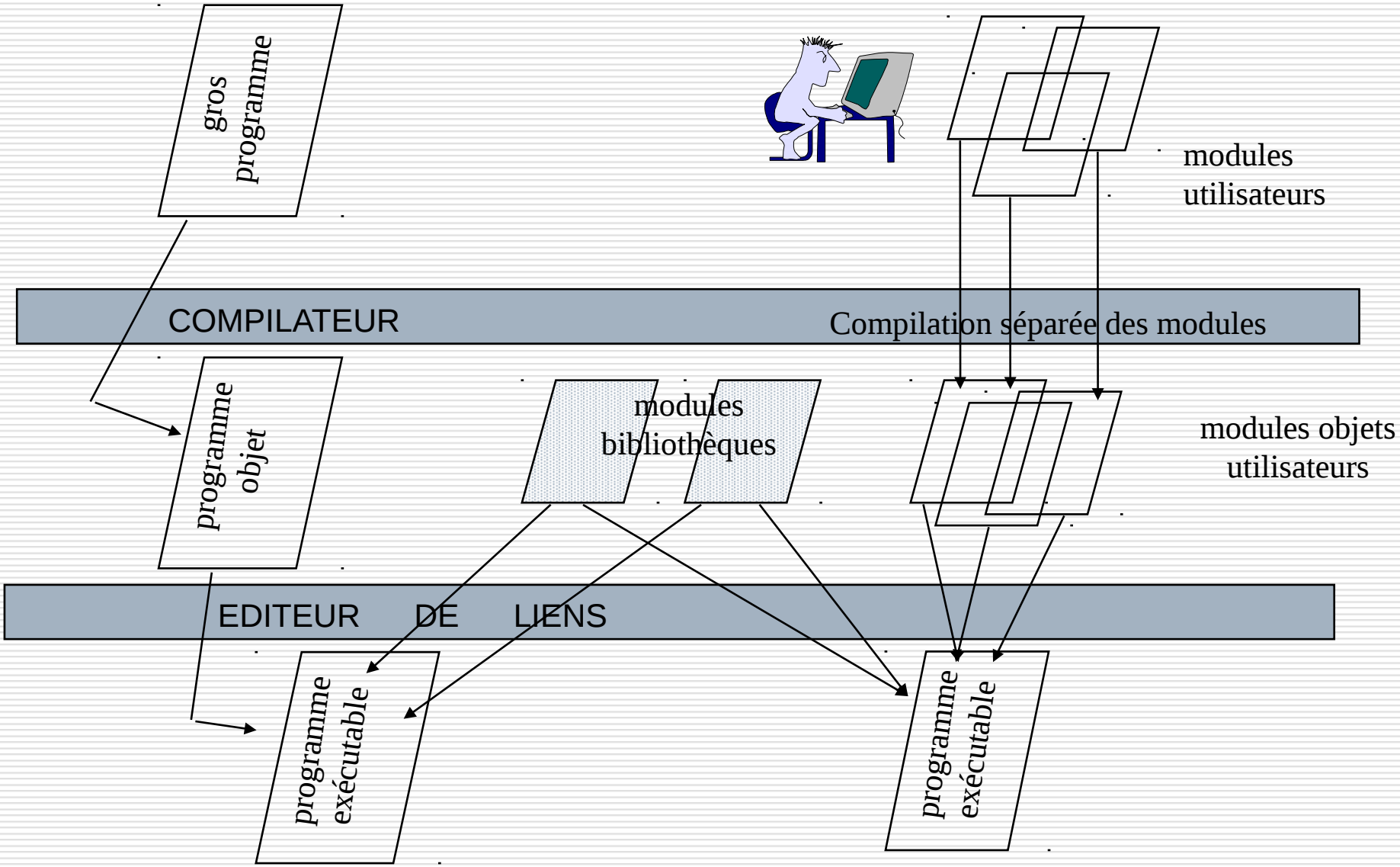
A partir de la liste des objets manipulés, le compilateur connaît leurs propriétés :

- type, durée de vie, taille, adresse

Contrôler la cohérence dans l'utilisation des objets :

- Erreur de type, absence de déclarations, déclarations multiples, déclarations inutiles, expressions incohérentes

Le développement d'un "gros programme"



Rôle de l'éditeur de liens

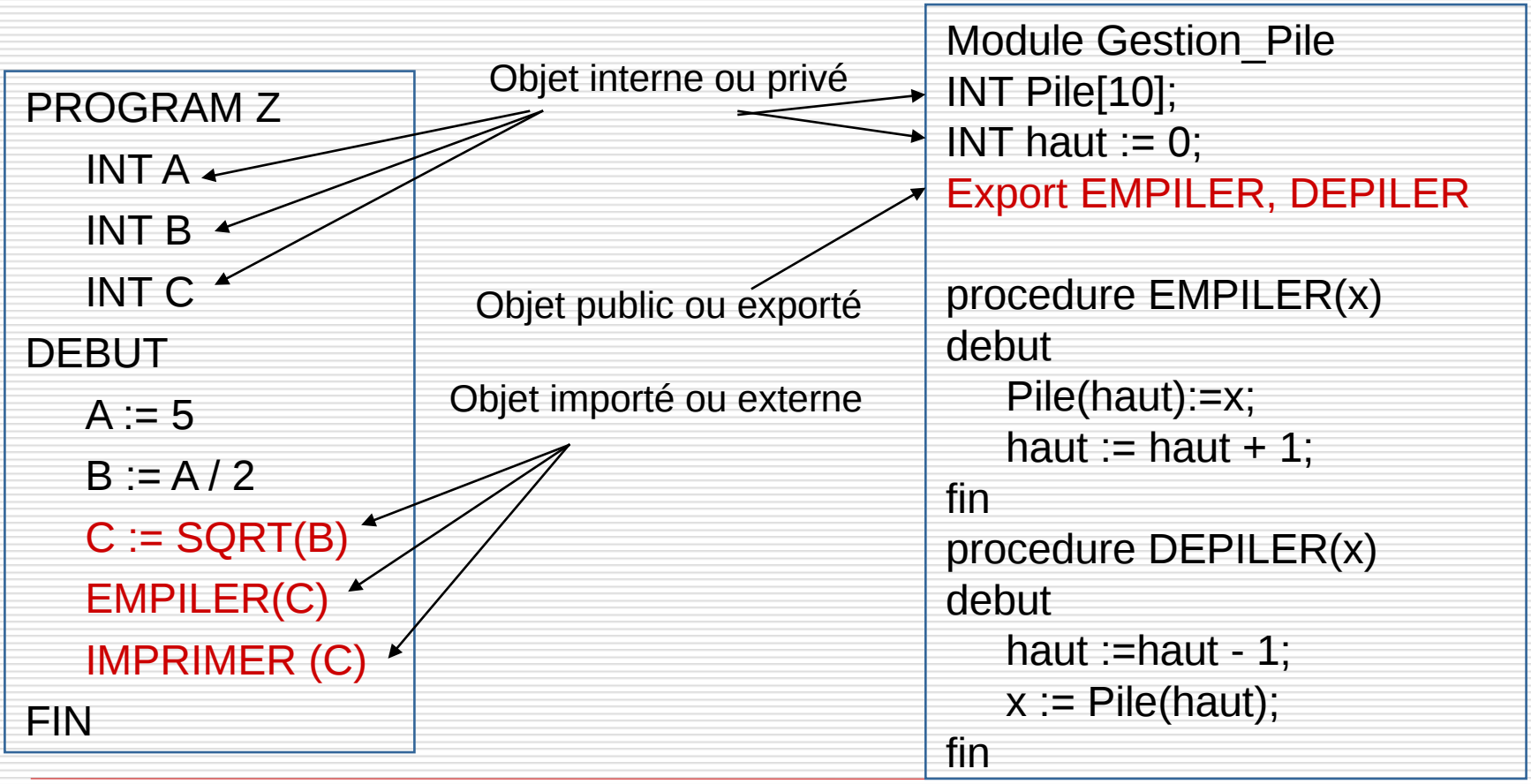
Un **éditeur de liens** est un logiciel qui permet de combiner plusieurs modules objets obtenus par compilation séparée pour construire un seul programme exécutable. Il combine deux sortes de modules objets :

- Les modules objets utilisateur
- Les modules objets prédéfinis dans des bibliothèques
  - fonctions interfaces des appels systèmes
  - fonctions mathématiques
  - fonctions graphiques
  - etc...

Les types d'objets des modules

Un module comprend trois catégories d'objets :

- objet interne au module, inaccessible de l'extérieur
- objet interne au module mais accessible de l'extérieur (**objet exporté ou public**)
- objet n'appartenant pas au module, mais utilisé par le module (**objet importé ou externe**)



Rôle de l'éditeur de liens

L'éditeur de liens doit construire le programme exécutable final à partir des modules objet entrant dans sa composition.

Il procède en trois étapes :

(1) *Construction de la carte d'implantation du programme*
(2) Construction de la table des liens utilisables
(3) Construction du programme exécutable final

Exercice

On dispose d'un ensemble de modules définis comme suit:

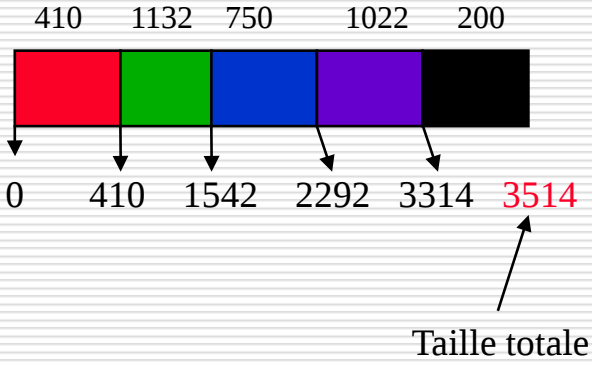
module GERER_ABONNEMENT taille 410
liens à satisfaire CREER_ABONNE
RECHERCHER_ABONNE
PAYER_ABONNEMENT
TROUVER_LIVRE
TROUVER_DVD
liens utilisables ENREGISTRER 212
RENDRE 321
adresse de lancement 100

module GERER_ABONNE taille 1132
liens utilisables CREER_ABONNE 212
RECHERCHER_ABONNE 620
DETRUIRE_ABONNE 1010
liens à satisfaire ENVOYER_COURRIER
IMPRIMER_CARTE

module ENTREES_SORTIES taille 750
liens utilisables ENVOYER_COURRIER 120
IMPRIMER_CARTE 324
EDITER_FACTURE 612

module GERER_FOND taille 1022
liens utilisables TROUVER_LIVRE 340
TROUVER_DVD 514
CREER_LIVRE 1011

module CAISSE taille 200
liens utilisables PAYER_ABONNEMENT 124
liens à satisfaire EDITER_FACTURE



CREER_ABONNE las 212 + 410
RECHERCHER_ABONNE las
PAYER_ABONNEMENT las
TROUVER_LIVRE las
TROUVER_DVD las
ENREGISTRER 212
RENDRE 321

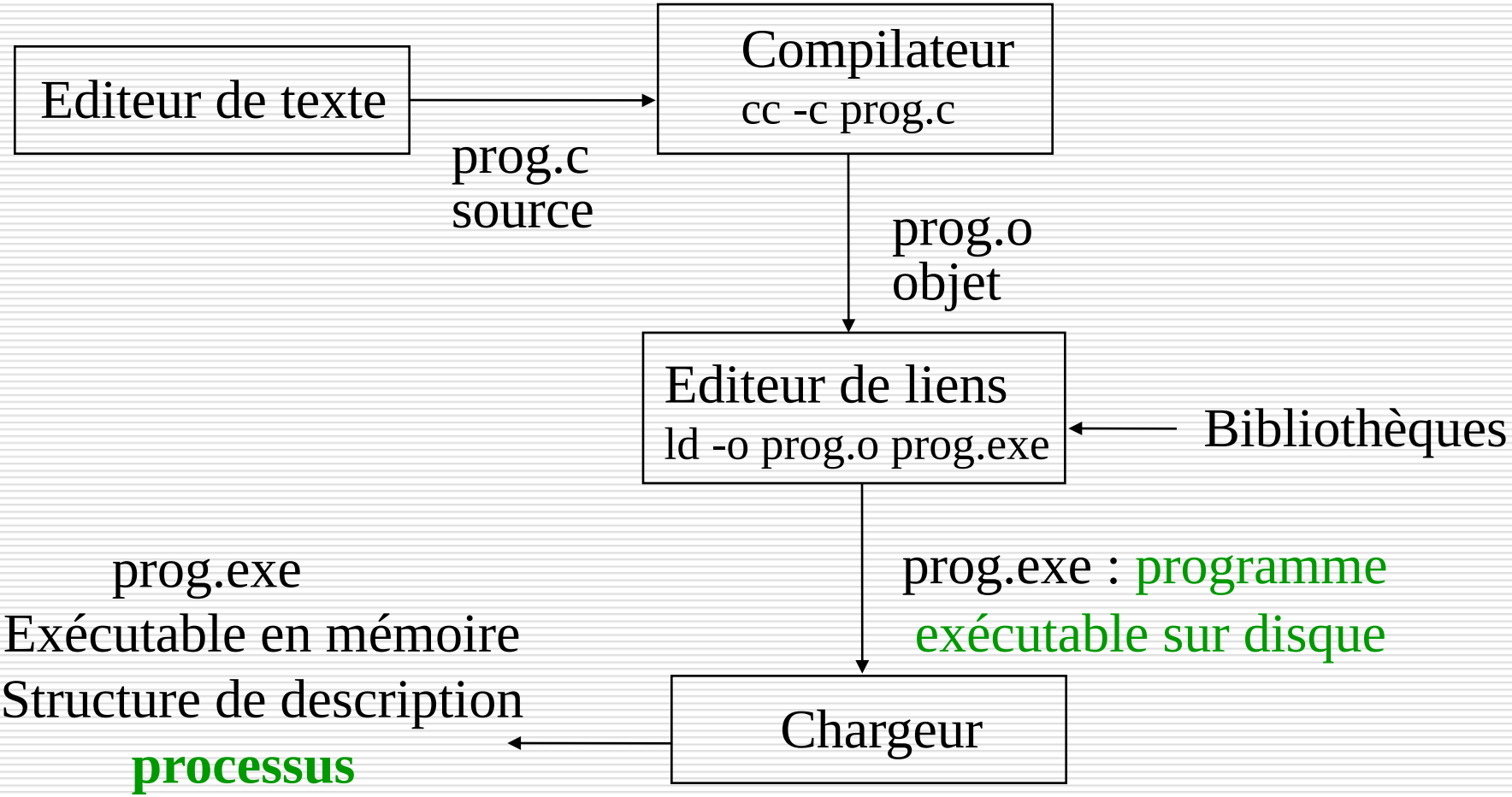
Exercice

Objet	LU	adresse
CREER_ABONNE	LAS, LU	212 + 410 = 622
RECHERCHER_ABONNE	LAS, LU	620 + 410 = 1030
PAYER_ABONNEMENT	LAS, LU	124 + 3314 = 3438
TROUVER_LIVRE	LAS, LU	340 + 2292 = 2632
TROUVER_DVD	LAS, LU	514 + 2292 = 2806
ENREGISTRER	LU	212 + 0 = 212
RENDRE	LU	321 + 0 = 321
DETRUIRE_ABONNE	LAS, LU	1010 + 410 = 1420
ENVOYER_COURRIER	LAS, LU	120 + 1542 = 1662
IMPRIMER_CARTE	LAS, LU	324 + 1542 = 1866
EDITER_FACTURE	LU	612 + 1542 = 2154
CREER_LIVRE	LU	1011 + 2292 = 3303

La gestion des exécutions des programmes

- Notion de processus
- Algorithmes d'ordonnancement
- L'exemple de Linux

Du programme au processus



prog.exe

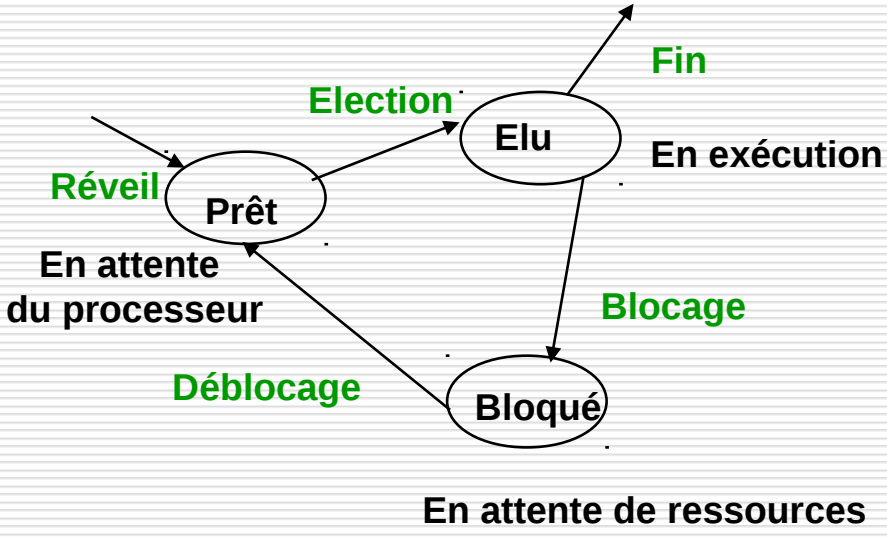
- Exécutable en mémoire
- Structure de description

processus

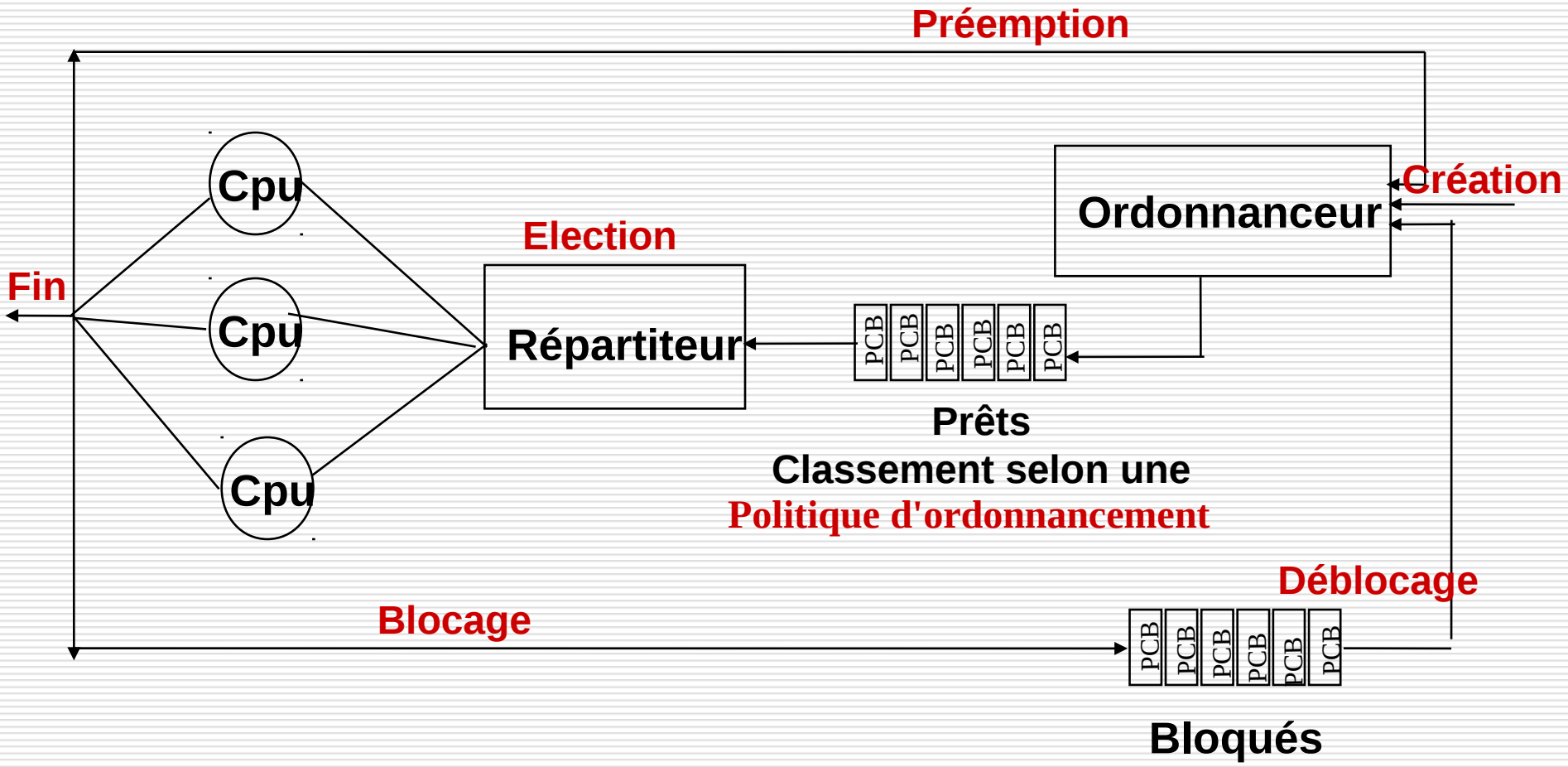
Bloc de Contrôle de Processus PCB

identificateur processus
état du processus
compteur instructions
contexte pour reprise (registres et pointeurs, piles,..)
pointeurs sur file d' attente et priorité(ordonnancement)
informations mémoire (limites et tables pages/segments
informations de comptabilisation et sur les E/S, périphériques alloués, fichiers ouverts,..

Bloc de contrôle de processus ou PCB



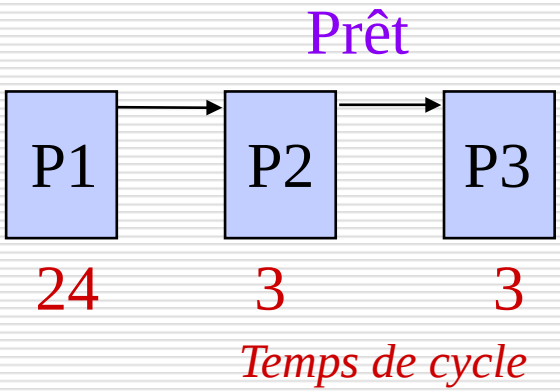
Ordonnanceur et répartiteur



Algorithme : Premier Arrivé Premier Servi

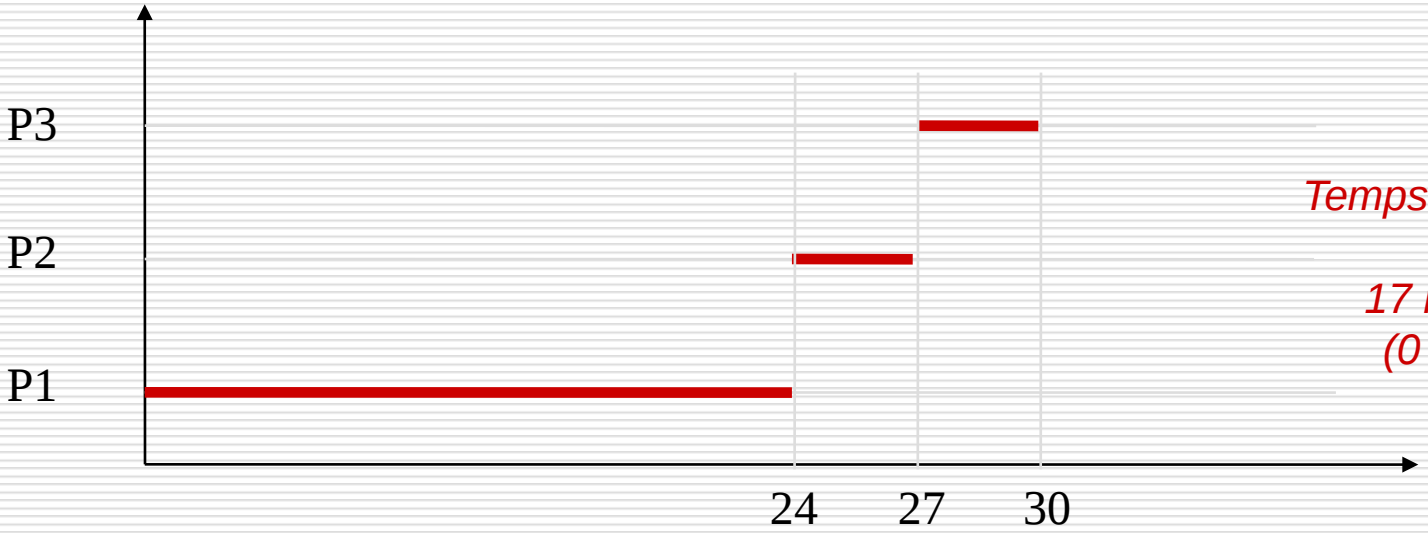
le cnam

FIFO, sans réquisition



Tps d'attente = date début exécution – date soumission

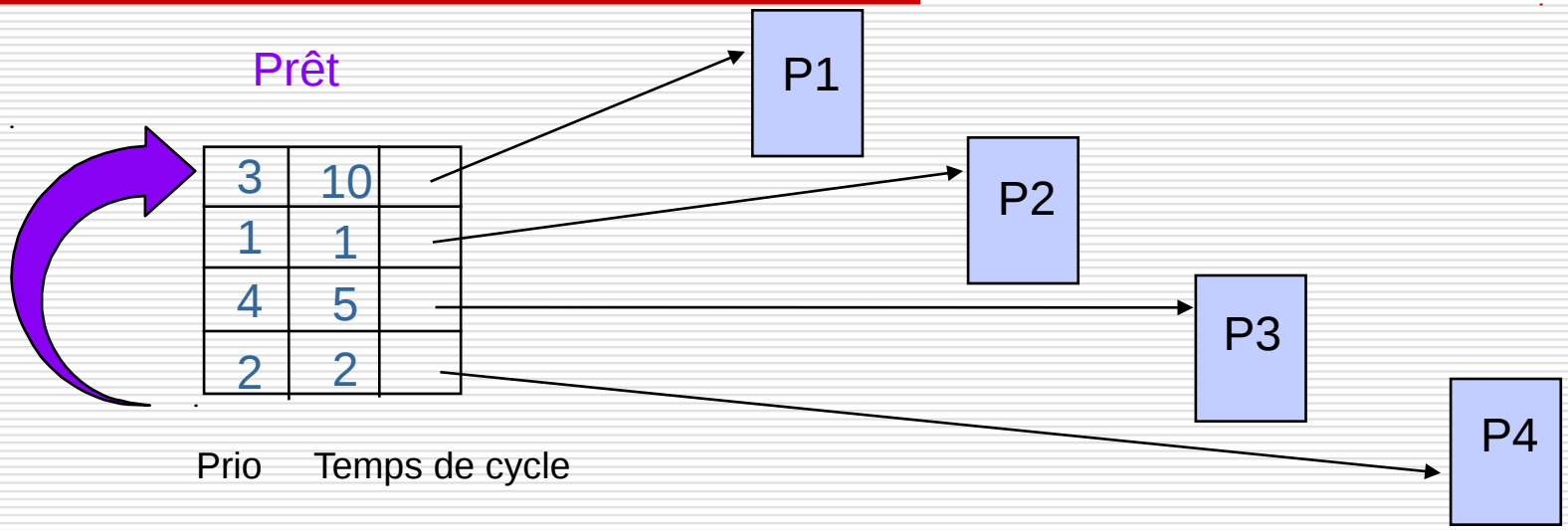
Tps de réponse = date fin exécution – date soumission



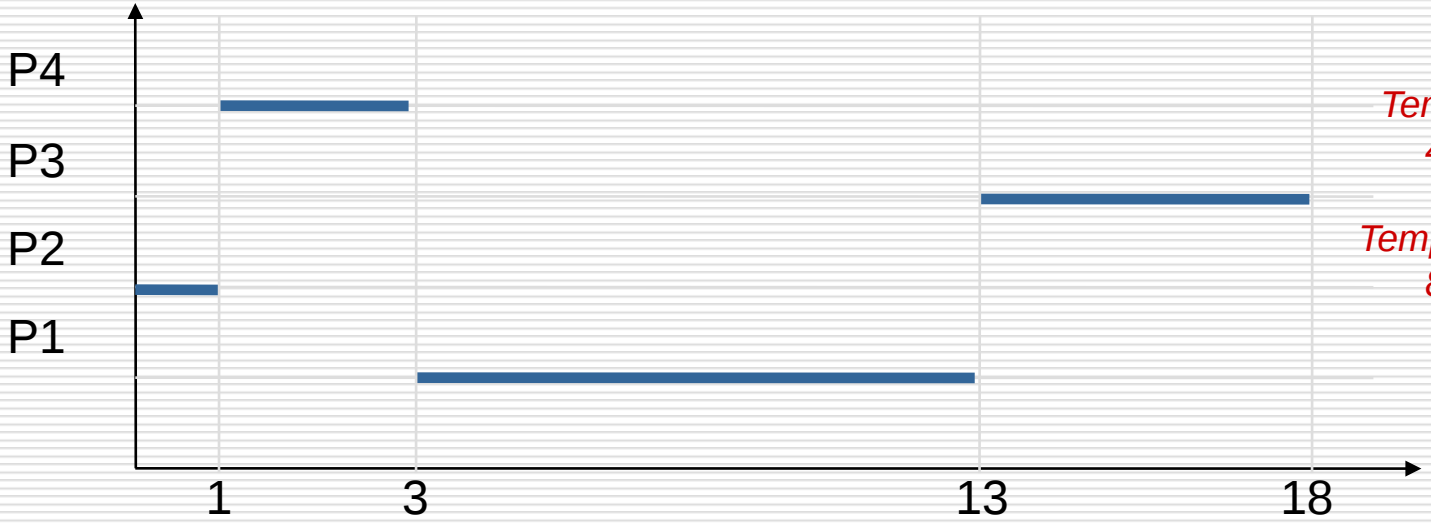
Temps moyen d'attente

17 millisecondes
(0 + 24 + 27)/3

Algorithme : avec priorités



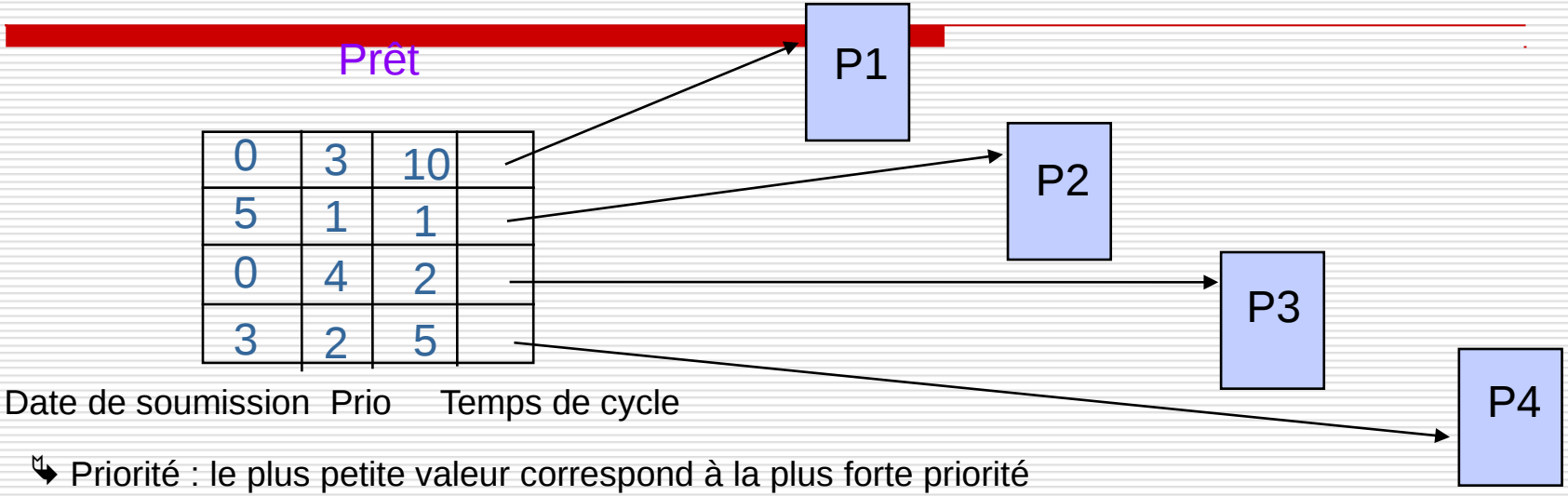
- Priorité : le plus petite valeur correspond à la plus forte priorité



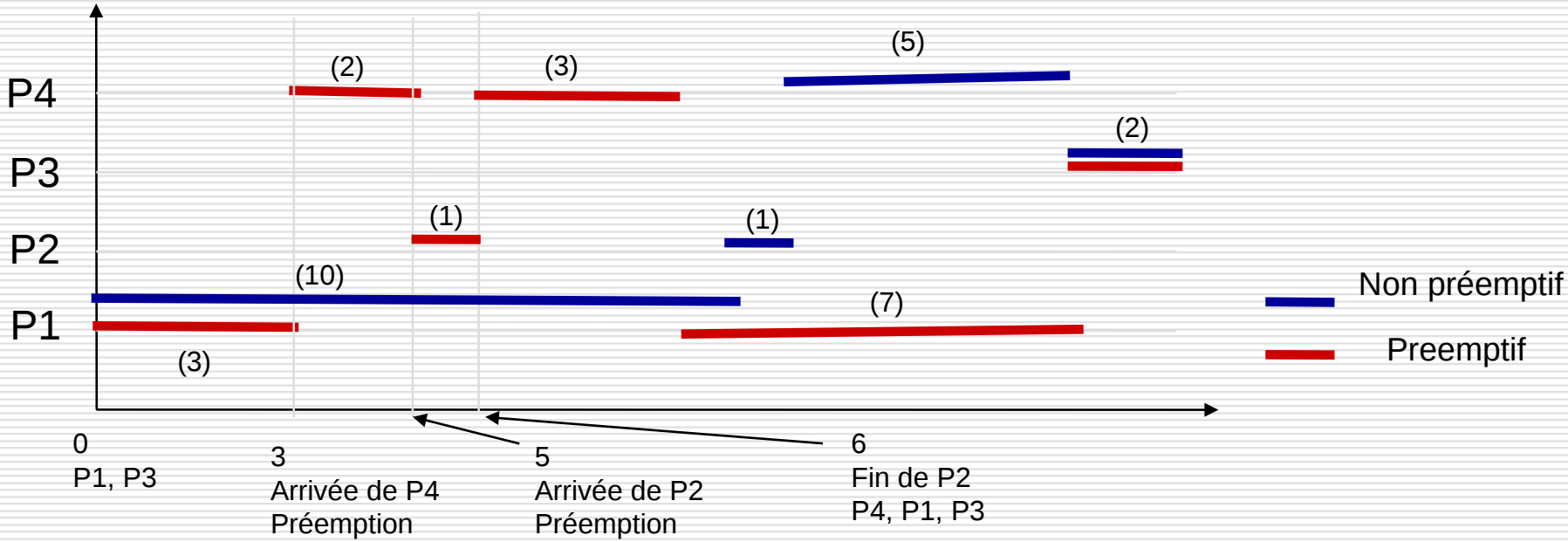
Temps moyen d'attente
4,25 millisecondes

Temps moyen de réponse
8,75 millisecondes

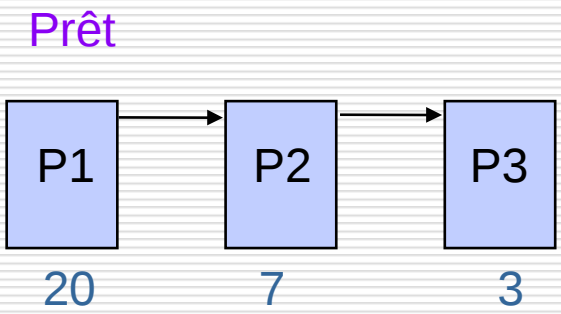
Algorithme : avec priorités

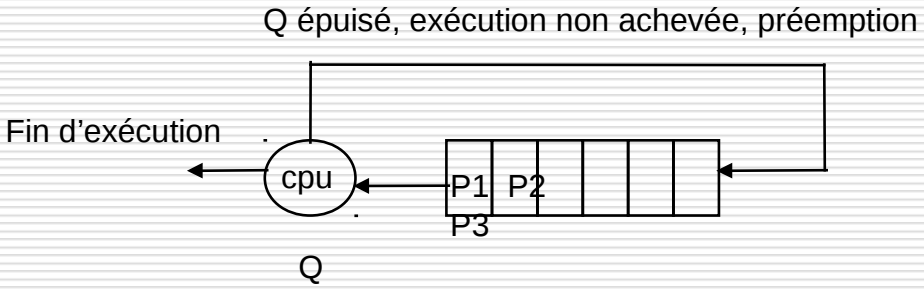


➥ Priorité : le plus petite valeur correspond à la plus forte priorité

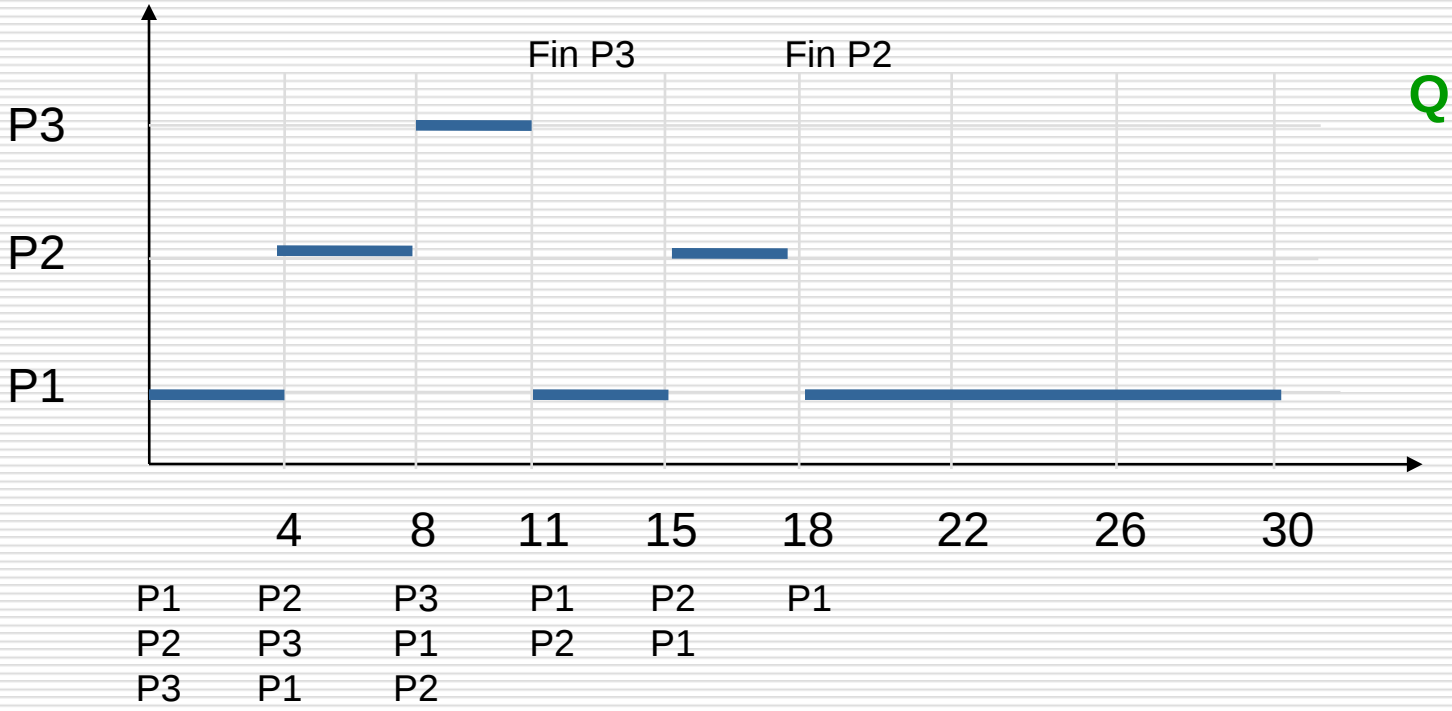


Algorithme : tourniquet





Temps de cycle



Quantum = 4

Exercice

On considère les deux processus suivants, P1 et P2 qui effectuent du calcul et des entrées-sorties avec un disque. Les entrées-sorties consistent à lire ou écrire un bloc du disque (B1 ou B2) ce qui demande 20 ms. Lors de la fin d'une entrée-sortie pour un processus, celui-ci est mis en bout de la file des processus prêts.

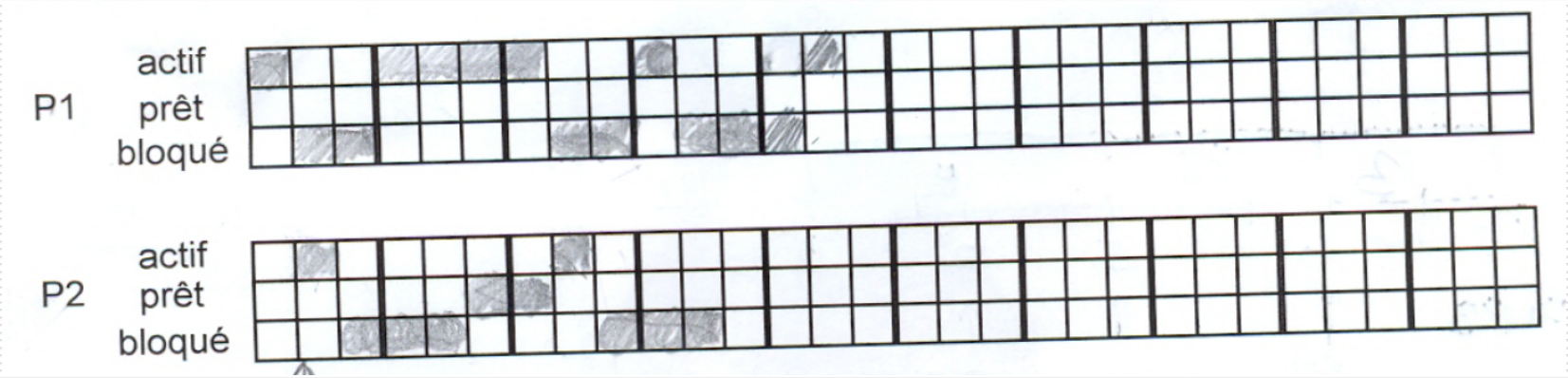
Processus P1

```
calcul 10 ms
   lecture B1
   calcul 40 ms
   écriture B1
   calcul 10 ms
   lecture B2
   calcul 10 ms
```

Processus P2

```
calcul 10 ms
   lecture B1
   calcul 10 ms
   écriture B1
```

Le processus P1 est lancé au temps 0, et **le processus P2 est lancé 10 ms après**. Donner le chronogramme correspondant en colorant les cases du diagramme suivant.



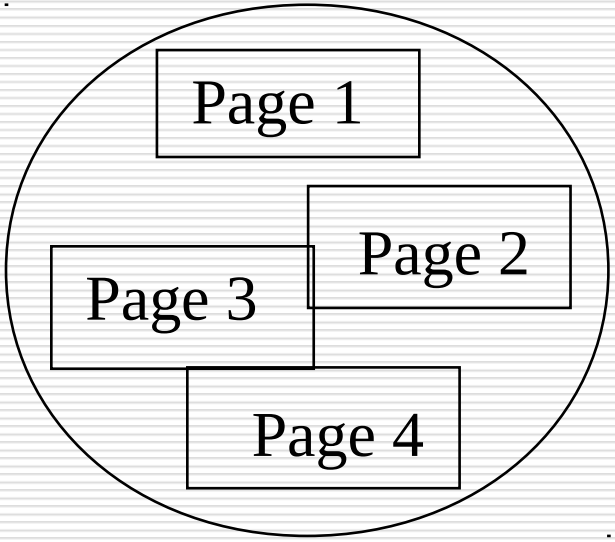
Gestion de la mémoire centrale

- Allocation de la mémoire physique : la pagination
- Gestion de la mémoire virtuelle
- Un exemple : linux

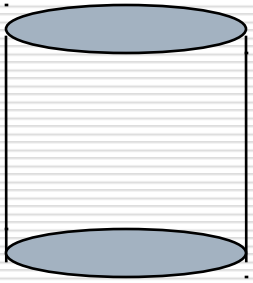
La pagination

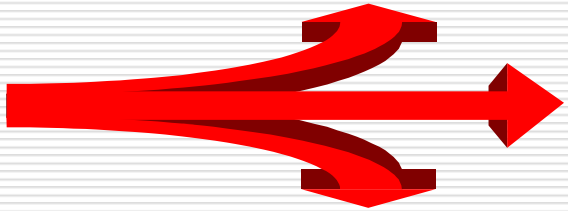
L'espace d'adressage du processus est découpé en **pages**
L'espace de la mémoire physique est découpé en **cases**
La taille d'une case est égale à la taille d'une page

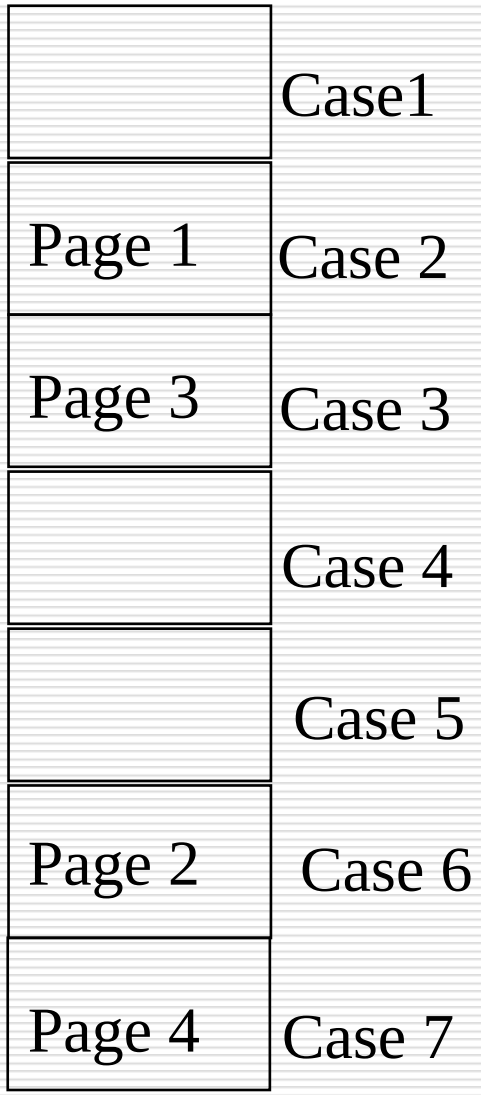
→ **Les pages sont placées dans n'importe quelle case libre de la mémoire centrale**



Espace d'adressage du programme

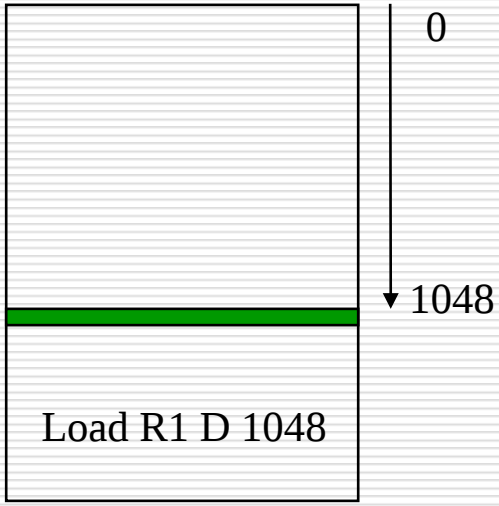




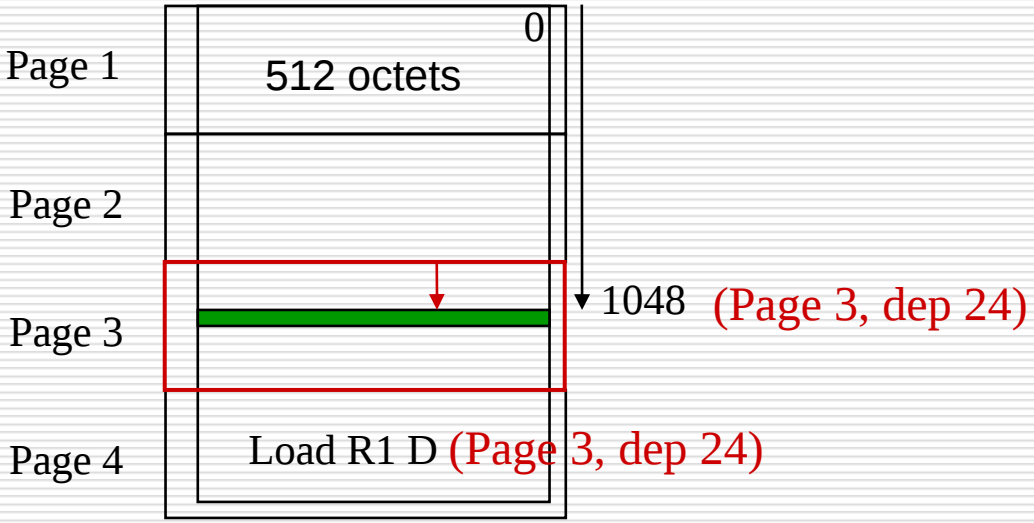


La pagination

L'espace d'adressage linéaire du programme est découpé en **pages**. Chaque adresse devient une **adresse paginée** formée d'un couple (numéro de page, déplacement dans la page)

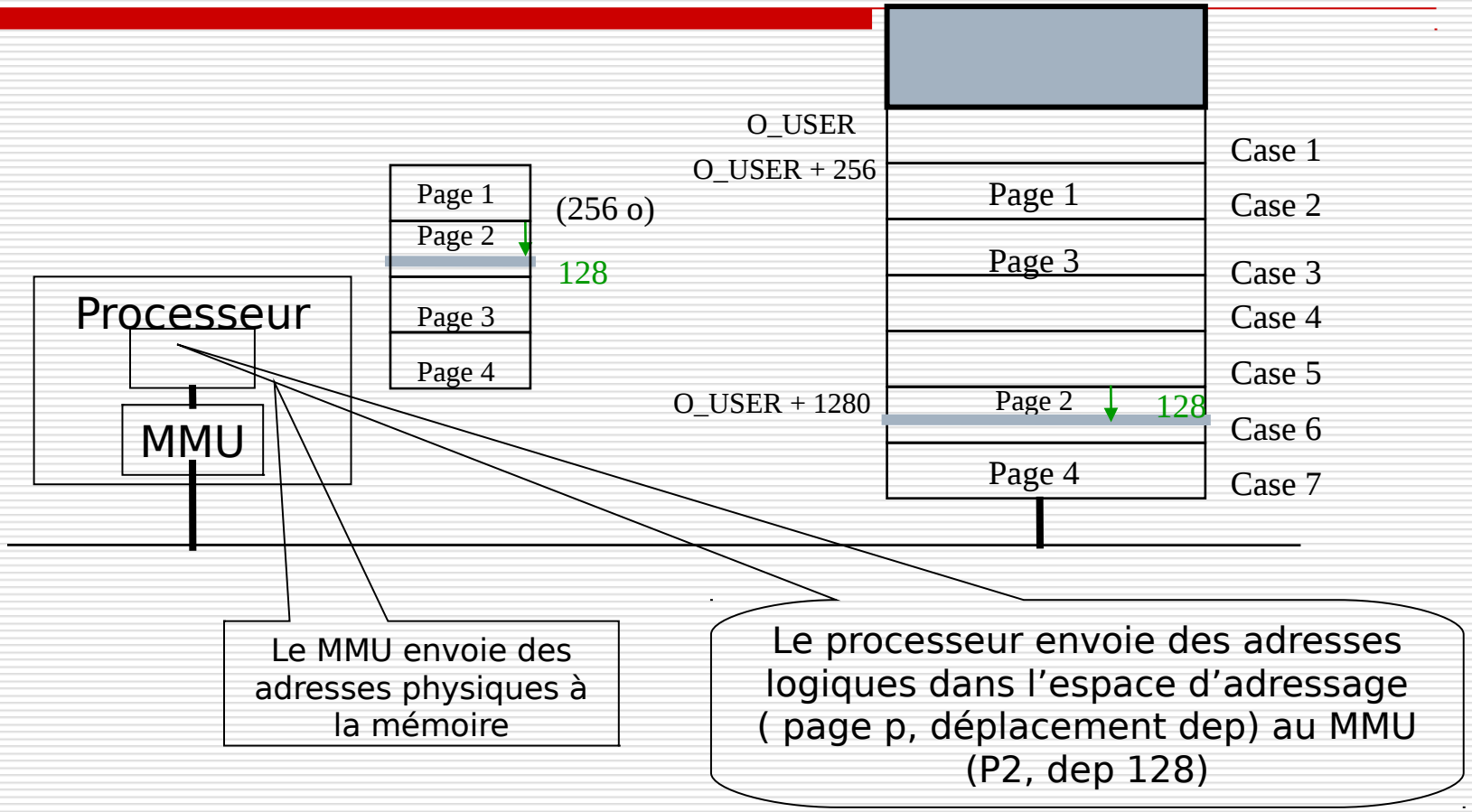


Espace d'adressage
du programme
LINEAIRE
Adresse linéaire
(déplacement depuis 0)



Espace d'adressage
du programme
PAGINE
Adresse paginée (logique)
(n° de page, déplacement dans la page depuis 0)

Conversion d'adresse

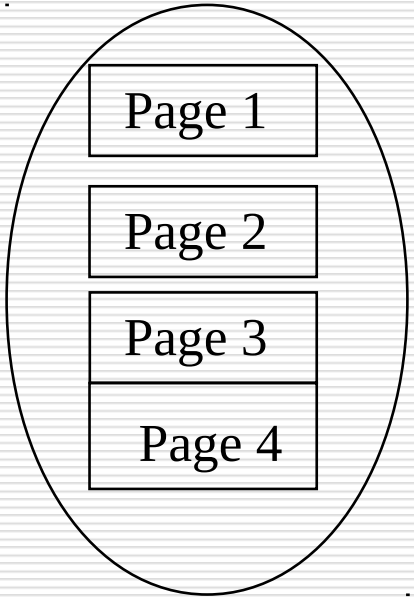


Il faut convertir l'adresse paginée en son équivalent adresse physique

Ad. physique = ad. impl. case contenant la page + deplacement dep

→ Nécessité d'une table des pages

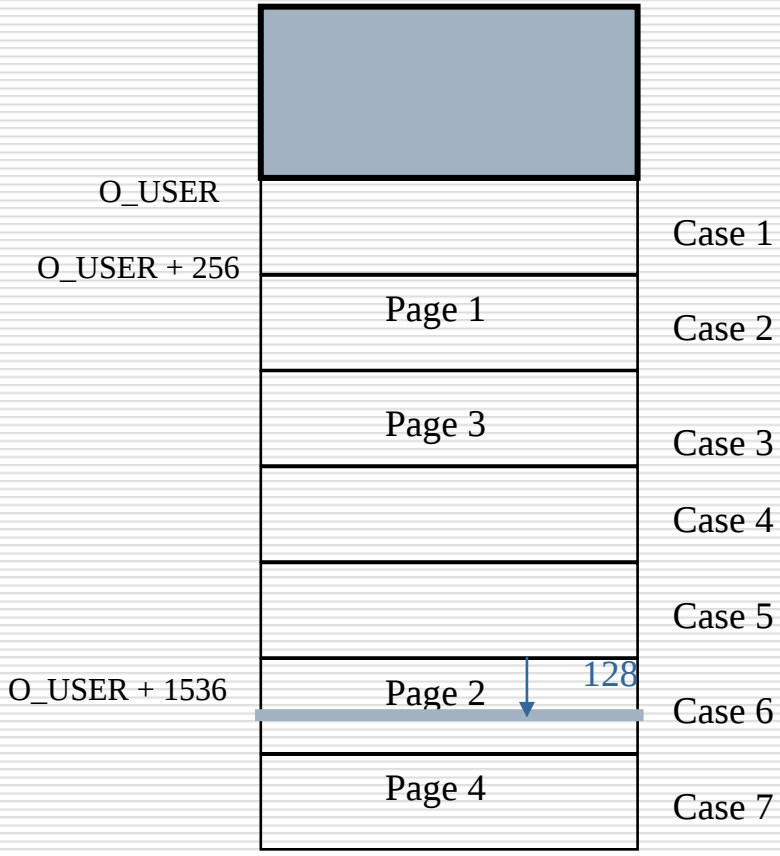
La mémoire paginée : table des pages



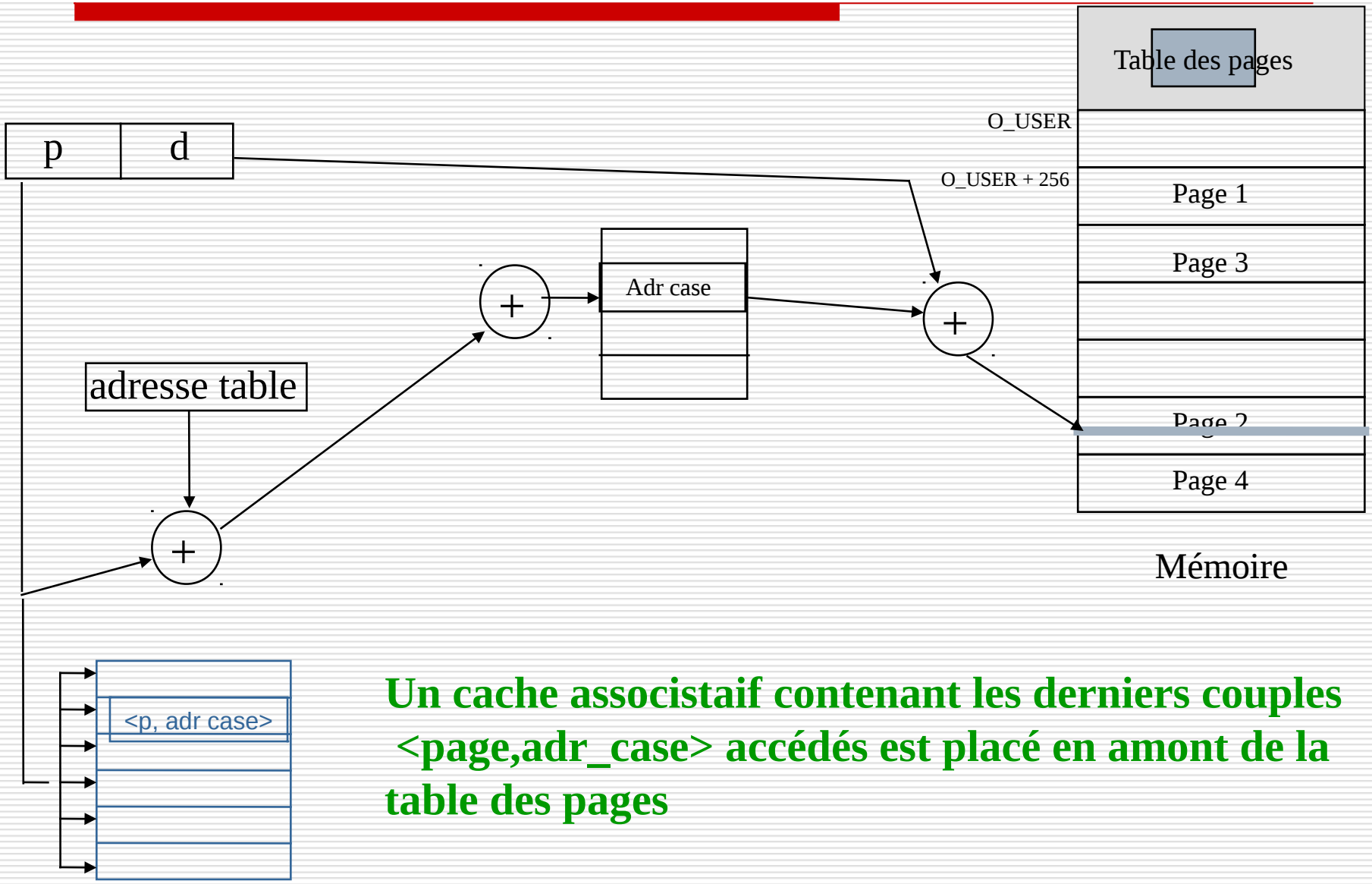
Espace d'adressage

Table des pages de l'espace d'adressage

Numéro page	adresse case
1	Adr C2 (O_User + 256)
2	Adr C6 (O_User + 1280)
3	Adr C3 (O_User + 512)
4	Adr C7 (O_User + 1536)



Accès à un mot physique



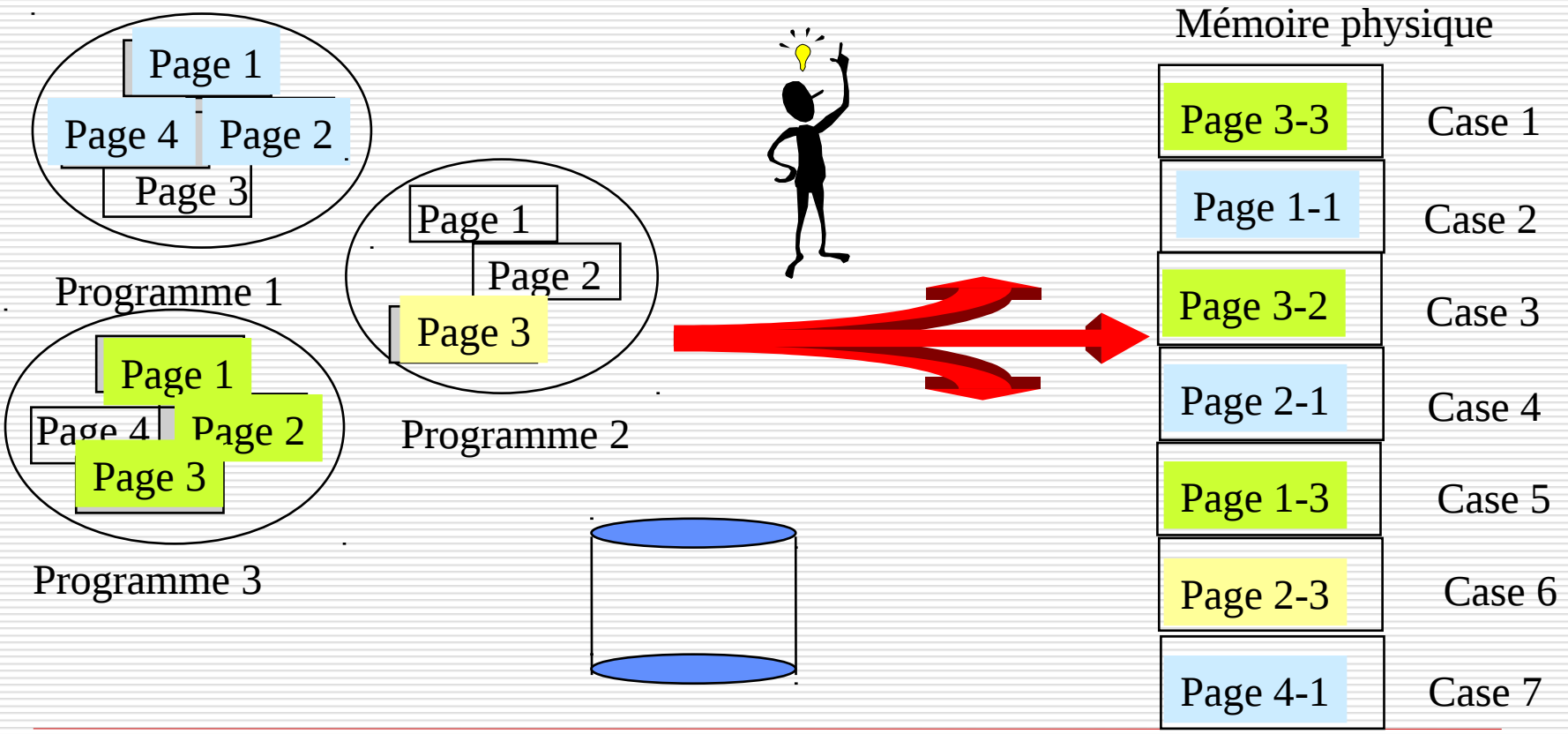
Un cache associstaif contenant les derniers couples <page,adr_case> accédés est placé en amont de la table des pages

La mémoire virtuelle

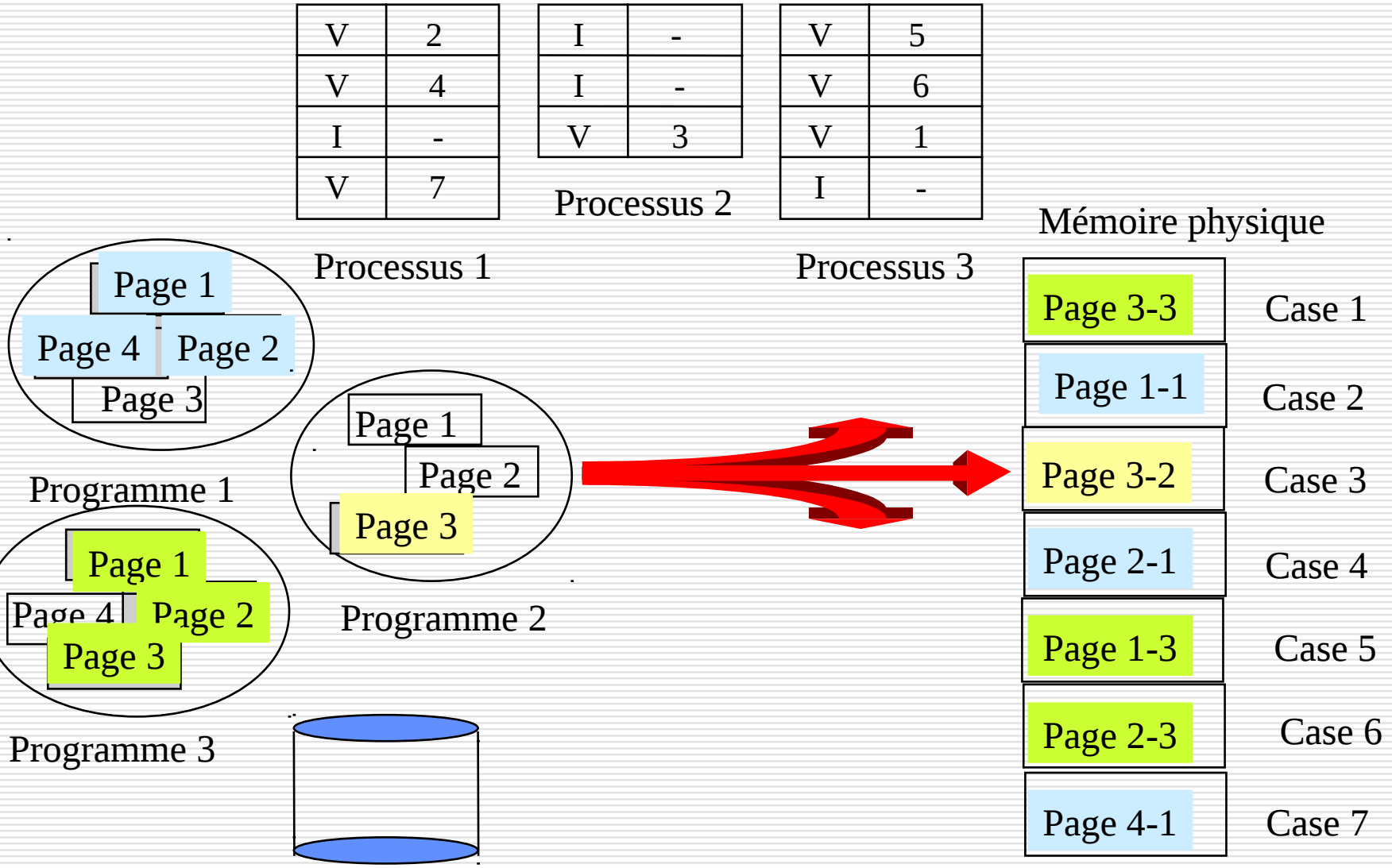
Mémoire virtuelle

La capacité de la mémoire centrale est trop petite pour charger l'ensemble des pages des programmes utilisateurs.

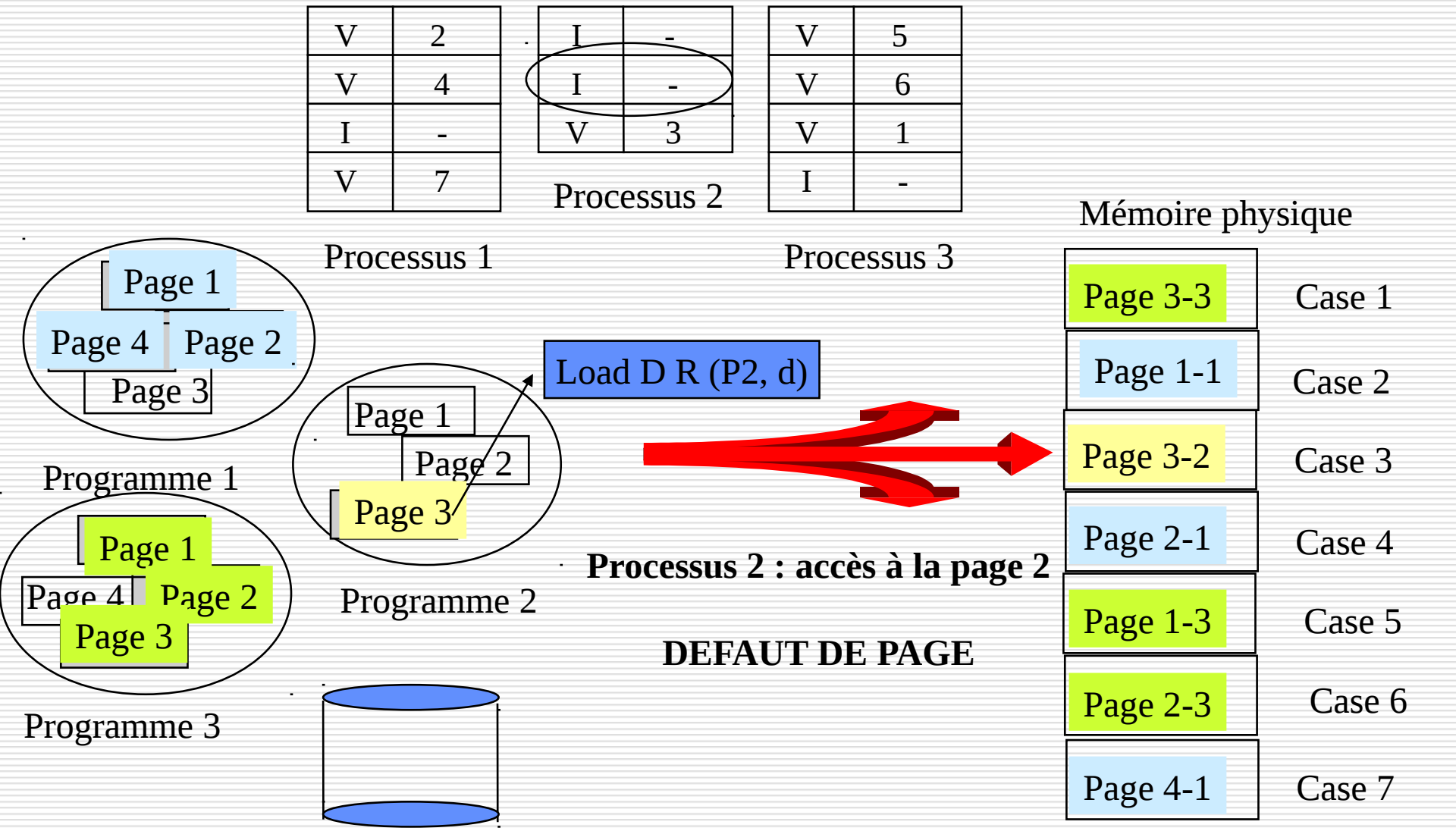
- Ne charger que les pages utiles à un instant (principes de localité).



Ne charger que les pages utiles à un instant (principes de localité)



Bit de validation et défaut de page



Bit de validation et défaut de page

le cnam

Ne charger que les pages utiles à un instant

il faut pouvoir tester la présence d'une page en mémoire centrale :

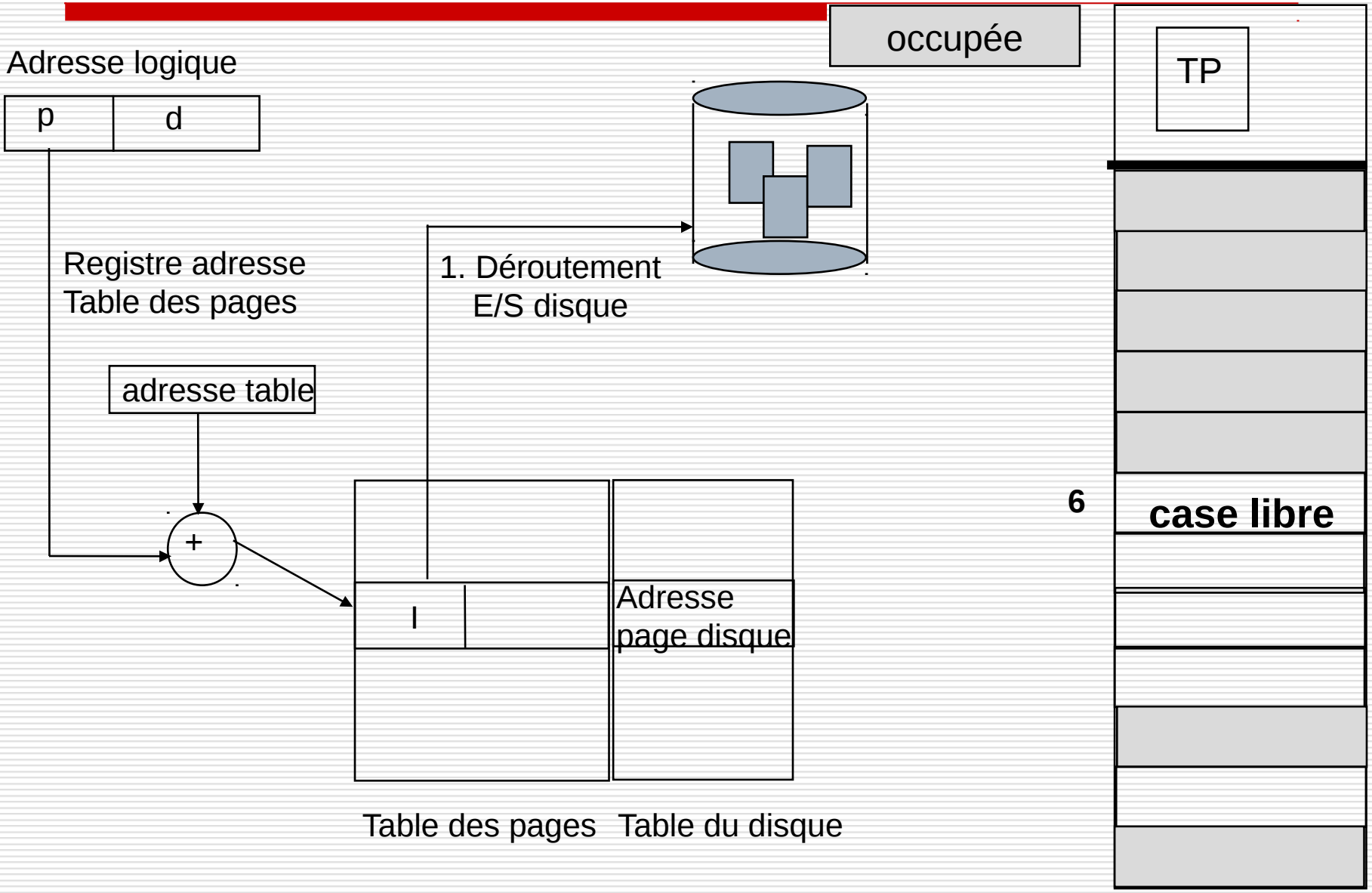
→ rôle du bit de validation

Si un processus cherche à accéder à une page non présente en mémoire centrale, il se produit un **déroutement de défaut de page :**

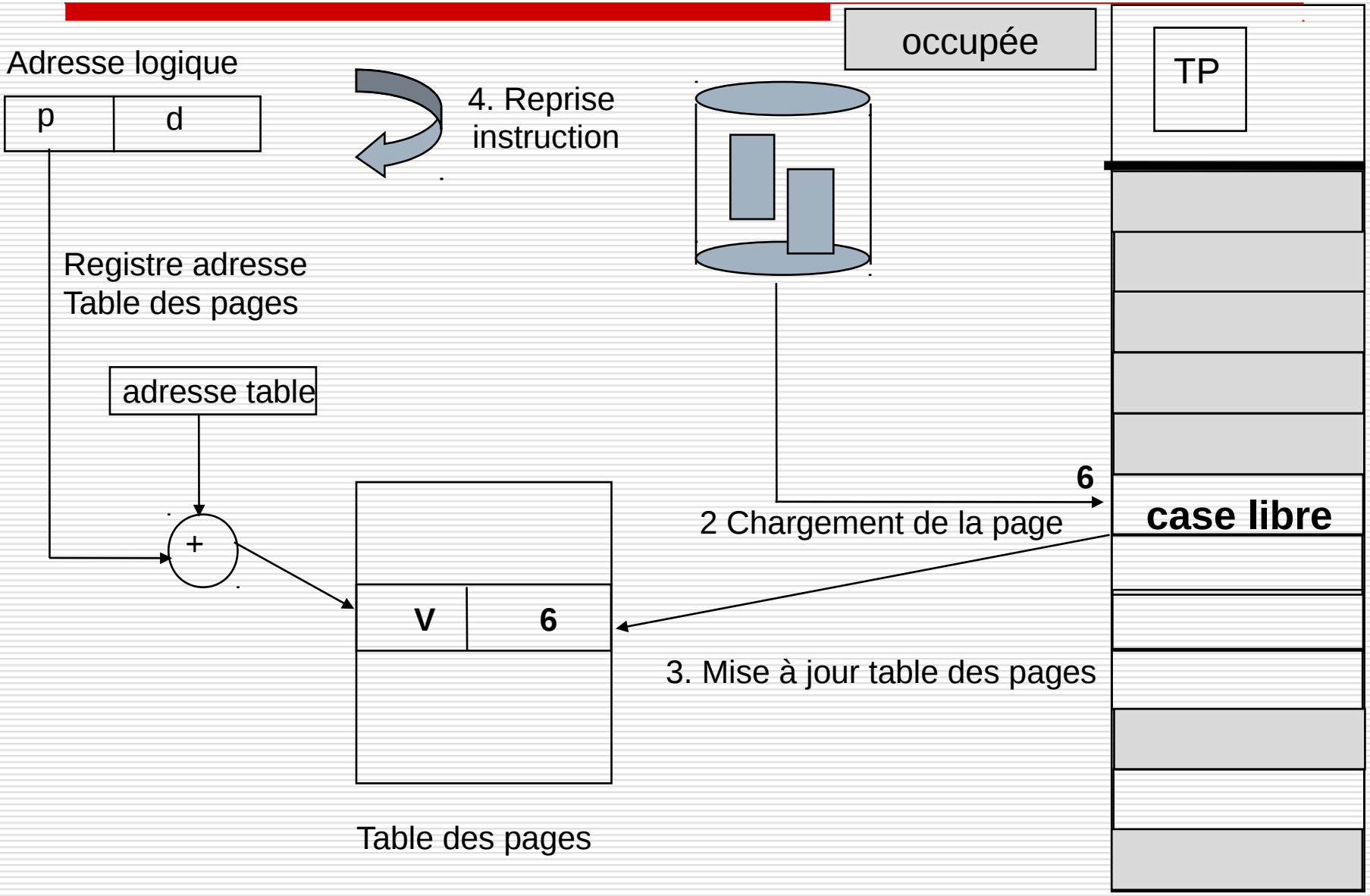
(1) Le système d'exploitation lance une entrée/sortie disque pour charger la page en mémoire dans une case libre.

(2) L'adresse de la page sur disque est stockée dans la table des pages.

Défaut de page



Défaut de page



Chargement de page

Lors d'un défaut de page, la page manquante est chargée dans une case libre

Mais la totalité des cases de la mémoire centrale peuvent être occupées

• le système d'exploitation utilise un algorithme pour choisir une case à libérer

L'optimal est de retirer une page devenue inutile

- Au hasard
- FIFO (First In, First out)
- LRU (Least Recently Used) : moins récemment utilisée

Exercice

On considère trois processus PA, PB et PC qui disposent d'un espace d'adressage paginé, respectivement composé de 4, 2 et 5 pages.
La mémoire centrale est composée de 15 cases numérotées de 1 à 15. Chaque case a une capacité de 512 octets. Lors d'un défaut de pages, la page manquante est chargée **dans la case libre de plus petit numéro.**
A l'instant t, l'allocation des espaces d'adressage est la suivante :
Pour le processus PA, seules les pages P1, P2 et P3 sont chargées en mémoire centrale respectivement dans les cases 5, 2 et 1 ;
Pour le processus PB, seule la page P1 est chargée en mémoire centrale dans la case 10 ;
Pour le processus PC, seules les pages P1, P2 et P5 sont chargées en mémoire centrale respectivement dans les cases 4, 8 et 11.

Question 1
Représentez sur un schéma les structures de données (tables des pages et mémoire centrale) correspondant à l'allocation décrite.

Question 2
Le processus PA accède à l'adresse linéaire 804 dans son espace d'adressage Donnez l'adresse paginée puis l'adresse physique correspondante.
Le processus PB accède à l'adresse linéaire 804 dans son espace d'adressage Donnez l'adresse paginée puis l'adresse physique correspondante.
Le processus PC accède à l'adresse linéaire 2544 dans son espace d'adressage Donnez l'adresse paginée puis l'adresse physique correspondante.

Num de case	MC	Adresse de la case
1	PA,3	0
2	PA,2	512
3		1024
4	PC,1	1536
5	PA,1	2048
6		2560
7		3072
8	PC,2	3584
9		4096
10	PB,1	4608
11	PC,5	5120
12		5636
13		6144
14		6656
15		7168

PA		
1	V	5 : o_user + 2048
2	V	2 : o_user + 512
3	V	1 : o_user +0
4	I	

PB		
1	V	10 : o_user + 4608
2	I	

PC		
1	V	4 : o_user + 1536
2	V	8 : o_user + 3584
3	V	
4	I	
5	V	11 : o_user +5120

PA accède à l'adresse linéaire 804 qui correspond à l'adresse paginée (2,292).
L'adresse physique correspondante est
o_user + 512 + 292

Numéro de case	MC	Adresse de début de case
1	PA,3	0
2	PA,2	512
3	PB,2	1024
4	PC,1	1536
5	PA,1	2048
6		2560
7		3072
8	PC,2	3584
9		4096
10	PB,1	4608
11	PC,5	5120
12		5636
13		6144
14		6656
15		7168

PB		
1	V	10 : o_user + 4608
2	V	3 : o_user + 1024

PB accède à l'adresse linéaire 804 qui correspond à l'adresse paginée (2,292).

La page 2 de PB n'est pas chargée, il y a défaut de page. La page 2 de PB est chargée en MC dans la case libre de plus petit numéro : ici la case 3.

La table des pages de PB est mise à jour.

L'adresse physique correspondant est : o_user + 1024 + 292

Numéro de case	MC	Adresse de début de case
1	PA,3	0
2	PA,2	512
3	PB,2	1024
4	PC,1	1536
5	PA,1	2048
6		2560
7		3072
8	PC,2	3584
9		4096
10	PB,1	4608
11	PC,5	5120
12		5636
13		6144
14		6656
15		7168

PC		
1	V	4 : o_user + 1536
2	V	8 : o_user + 3584
3	V	
4	I	
5	V	11 : o_user +5120

PC accède à l'adresse linéaire 2544 qui correspond à l'adresse paginée (5,496).

L'adresse physique correspondant est : o_user + 5120 + 496

FIFO : la plus anciennement chargée

Référence	0	1	4	2	0	1	3	0	4
défauts	*	*	*	*	*	*	*		*
	0	0	0	**2**	2	2	**3**	3	3
		1	1	1	**0**	0	0	0	**4**
			4	4	4	**1**	1	1	1

LRU : la moins récemment accédée

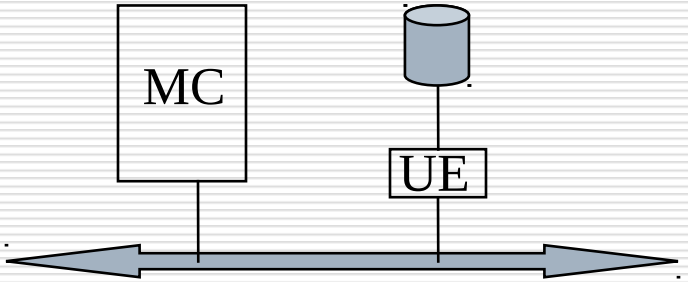
référence	0	1	4	2	0	1	3	0	4
défauts	*	*	*	*	*	*	*		*
	0	0	0	**2**	2	2	**3**	3	3
		1	1	1	**0**	0	0	0	0
			4	4	4	**1**	1	1	**4**

Système de gestion de fichiers

- La notion de fichier logique
- La notion de fichier physique : allocation du support de masse
- Notion de répertoires

Introduction

La mémoire centrale est une mémoire volatile



Il faut stocker les données devant être conservées au delà de l'arrêt de la machine sur un support de masse permanent

↳ l'unité de conservation sur le support de masse est **le *fichier***

Introduction

Les données dans le fichier sont organisées selon les besoins de l'utilisateur.

↳ Fichier logique

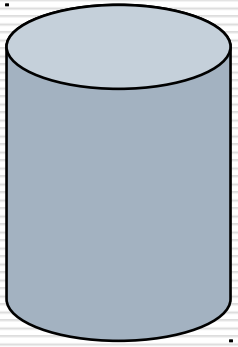
Niveau utilisateur

Interface : fonctions du SGF

Niveau Système ou physique (Système de Gestion de Fichiers) :

- allocation des fichiers sur le support de masse
- répertoire

↳ Fichier Physique

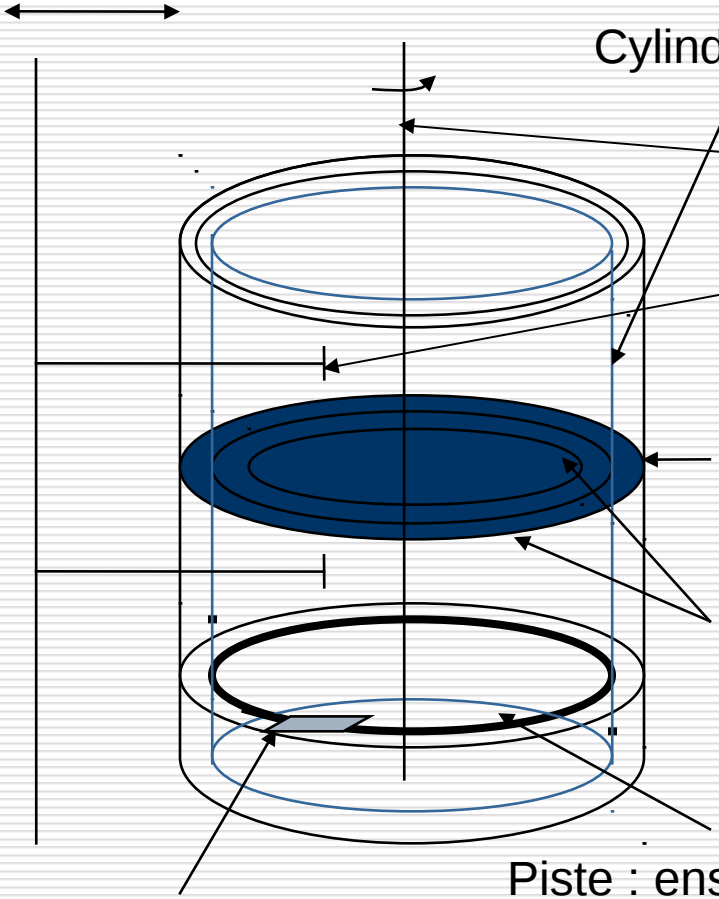


Le fichier physique

Structure du disque dur

le cnam

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



Cylindre : ensemble des pistes accessibles sans déplacement des têtes

Moteur

tête de lecture/écriture

plateau

Face : 20 à 1500 pistes

Piste : ensemble de secteurs (4 à 32)

Alimentation

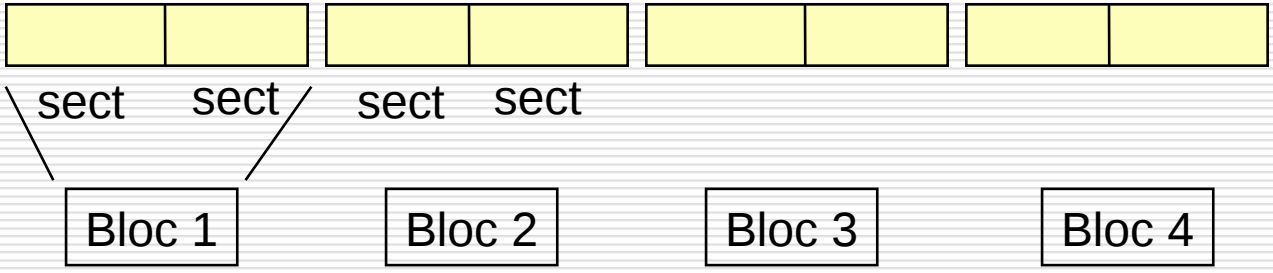
Secteur : plus petite unité d'information accessible
32 à 4096 octets

Allocation du disque : le bloc physique

L'unité d'allocation sur le disque dur est le **bloc physique**.
Il est composé de 1 à n secteurs

ex: **1 bloc = 2 secteurs de 512 octets soit 1KO**

Les opérations de lecture et d'écriture du SGF se font bloc par bloc



Fichier physique

Le fichier physique correspond à l'implémentation sur le support de masse de l'unité de conservation : *le fichier*

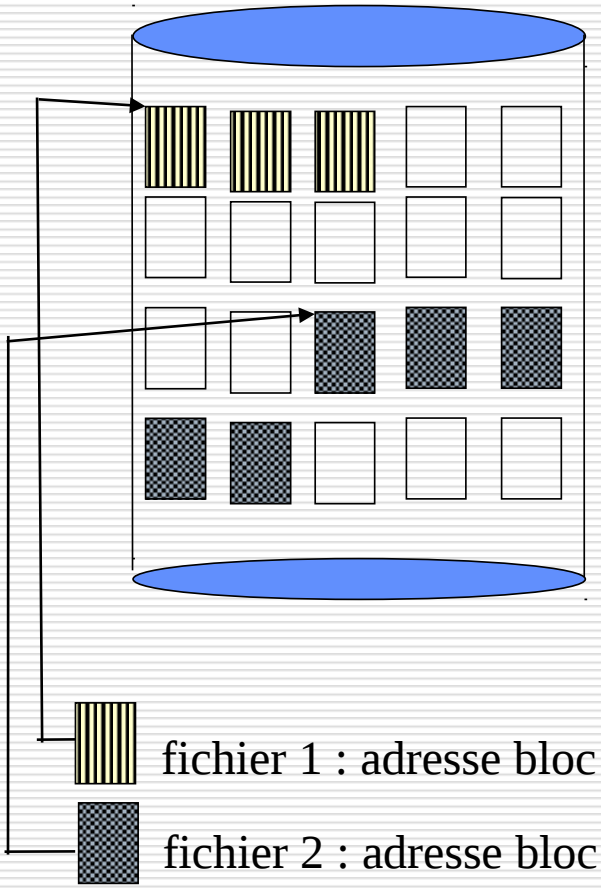
Il est constitué d'un ensemble de blocs physique. Il existe plusieurs méthodes d'allocation des blocs physiques :

- allocation contiguë (séquentielle simple)
- allocation par zones
- allocation par blocs chainés
- allocation indexée

→ il faut gérer et représenter l'espace libre

Allocation contiguë

Un fichier occupe un ensemble de blocs contigus sur le disque



Avantage : Bien adapté au méthodes d'accès séquentielles et directes.

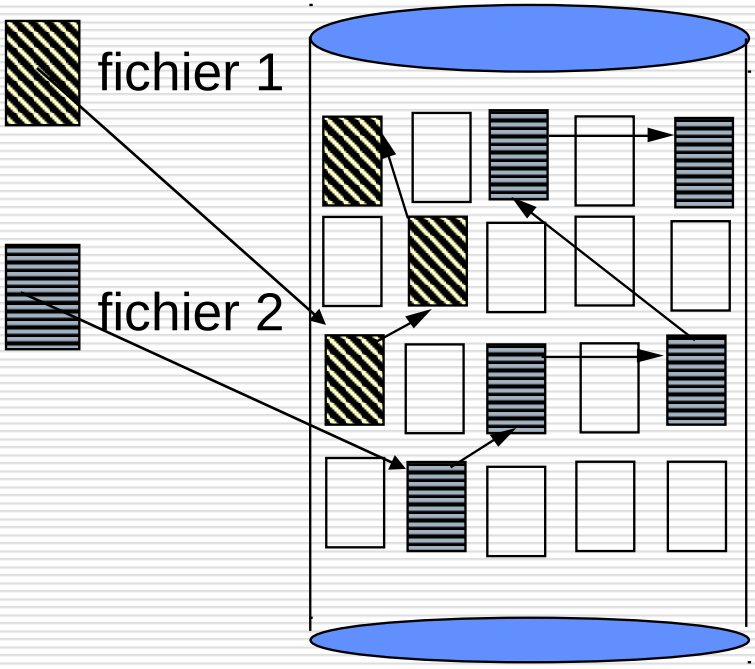
Difficultés :
- création d'un nouveau fichier
- extension du fichier

fichier 1 : adresse bloc 1, longueur 3 blocs

fichier 2 : adresse bloc 13, longueur 5 blocs

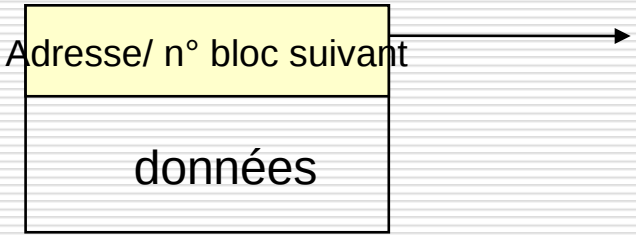
Allocation par bloc chainée

Un fichier est constitué comme une liste chainée de blocs physiques, qui peuvent être dispersés n'importe où.



- Avantages :
  Extension simple du fichier : allouer un nouveau bloc et le chainer au dernier
  Pas de fragmentation

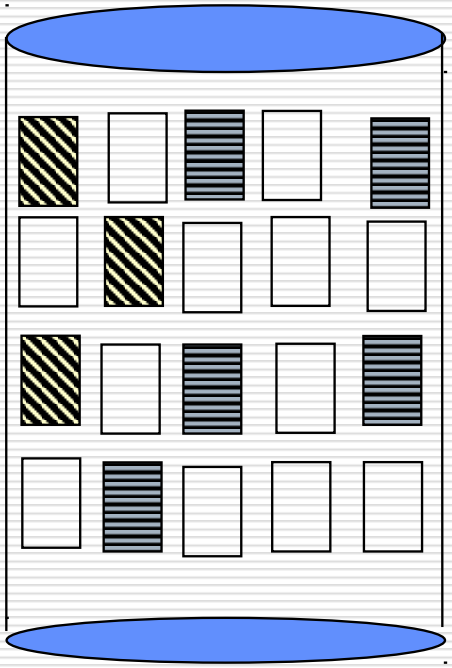
- Difficultés :
  Mode séquentiel seul
  Le chaînage du bloc suivant occupe de la place dans un bloc



Allocation par bloc chainée : variante

Une table d'allocation des fichiers (File Allocation Table - FAT) regroupe l'ensemble des chainages.

(exemple systèmes Windows)



fichier 1

fichier 2

N° bloc	
1	NULL
2	Libre
3	5
4	Libre
5	NULL
6	Libre
7	1
11	7
12	Libre
13	15
14	Libre
15	3
16	Libre

FAT

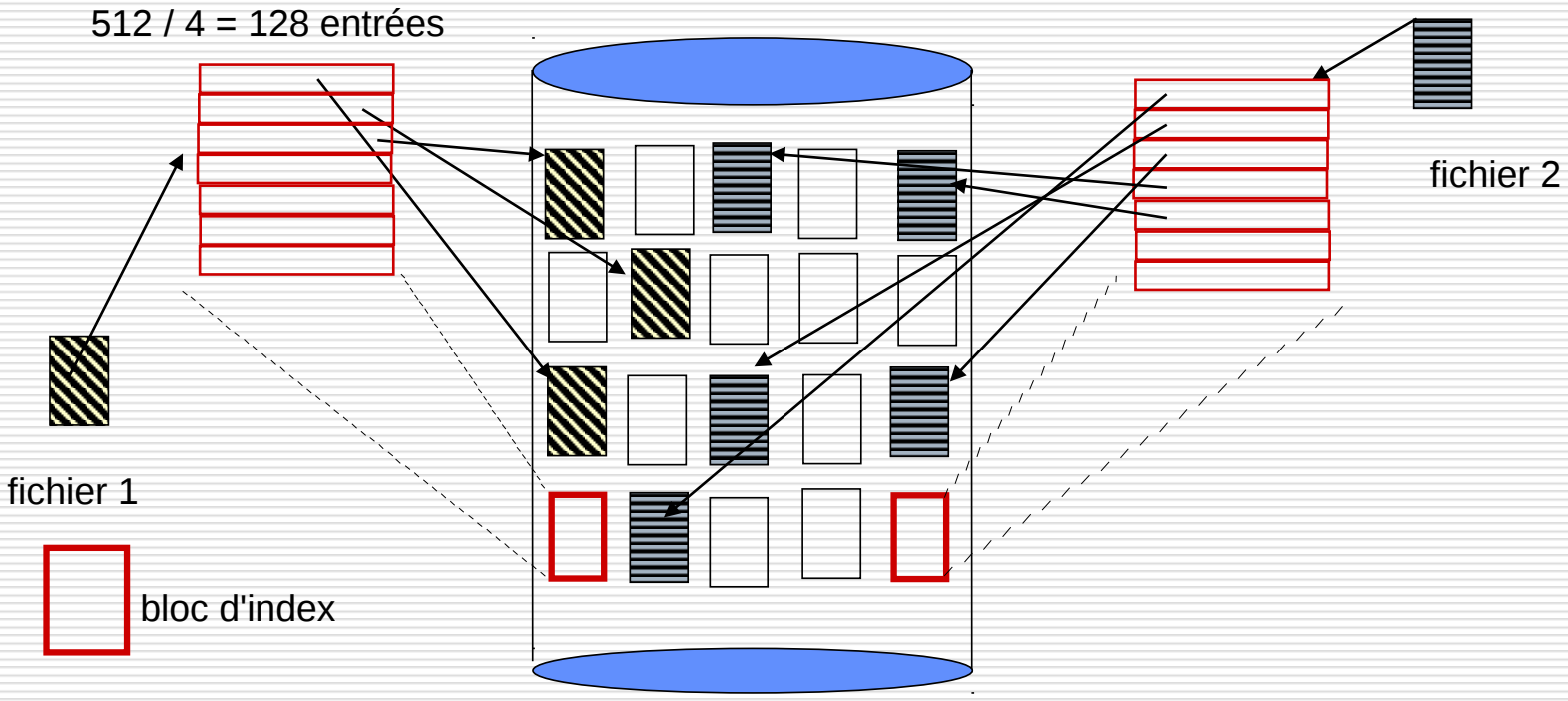
Fin de fichier

Bloc non alloué

N° de Bloc suivant Alloué pour le fichier

Allocation indexée

Les adresses des blocs physiques constituant un fichier sont rangées dans une table appelée index, elle-même contenue dans un ou plusieurs blocs disque



- Supporte bien l'accès direct
- « gaspillage de place » dans le bloc d'index

Allocation indexée : la solution Unix

Les 10 premières entrées de la table contiennent l'adresse d'un bloc de données du fichier

Bloc = 1024 octets → 10 Ko alloués

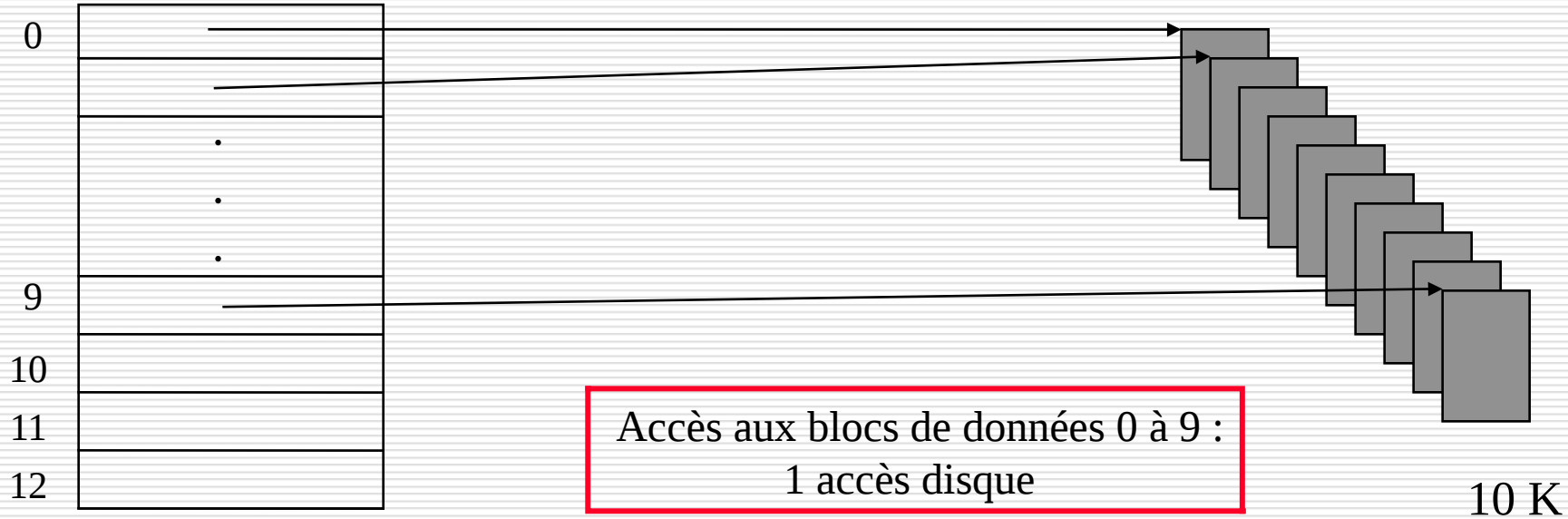


Table d'allocation
13 entrées

Allocation indexée : la solution Unix

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 = 1024 octets ; adresse de bloc = 4 octets → 256 entrées dans le bloc d'index

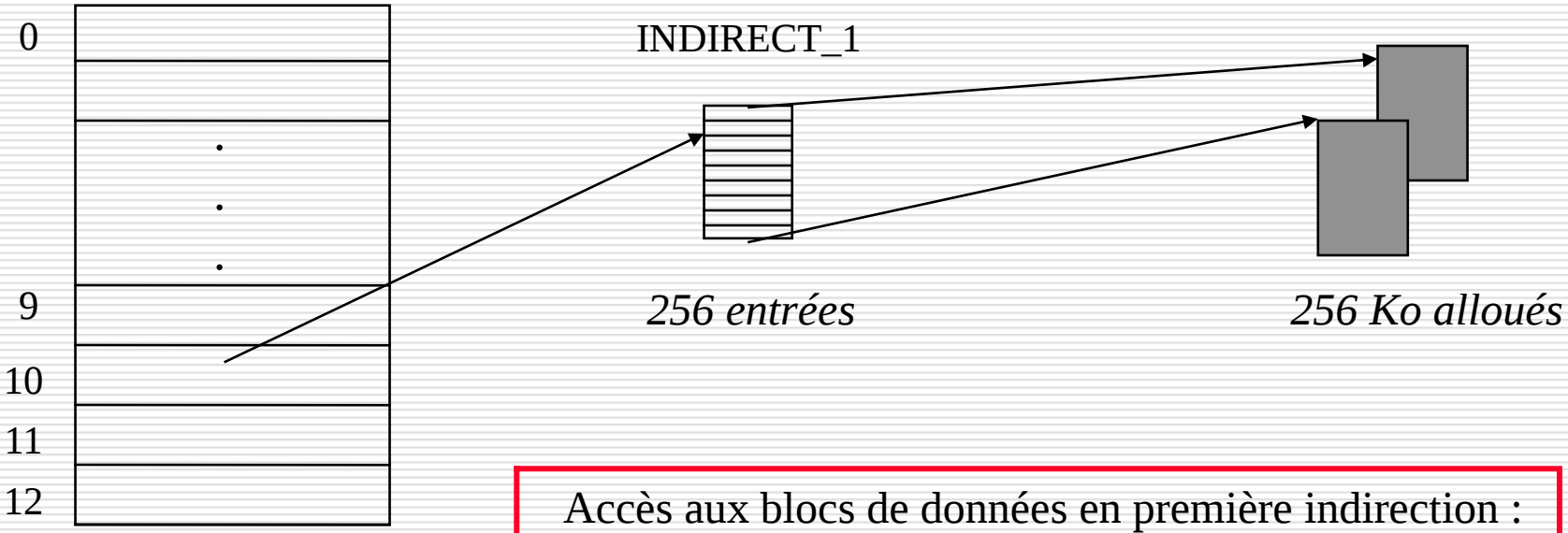


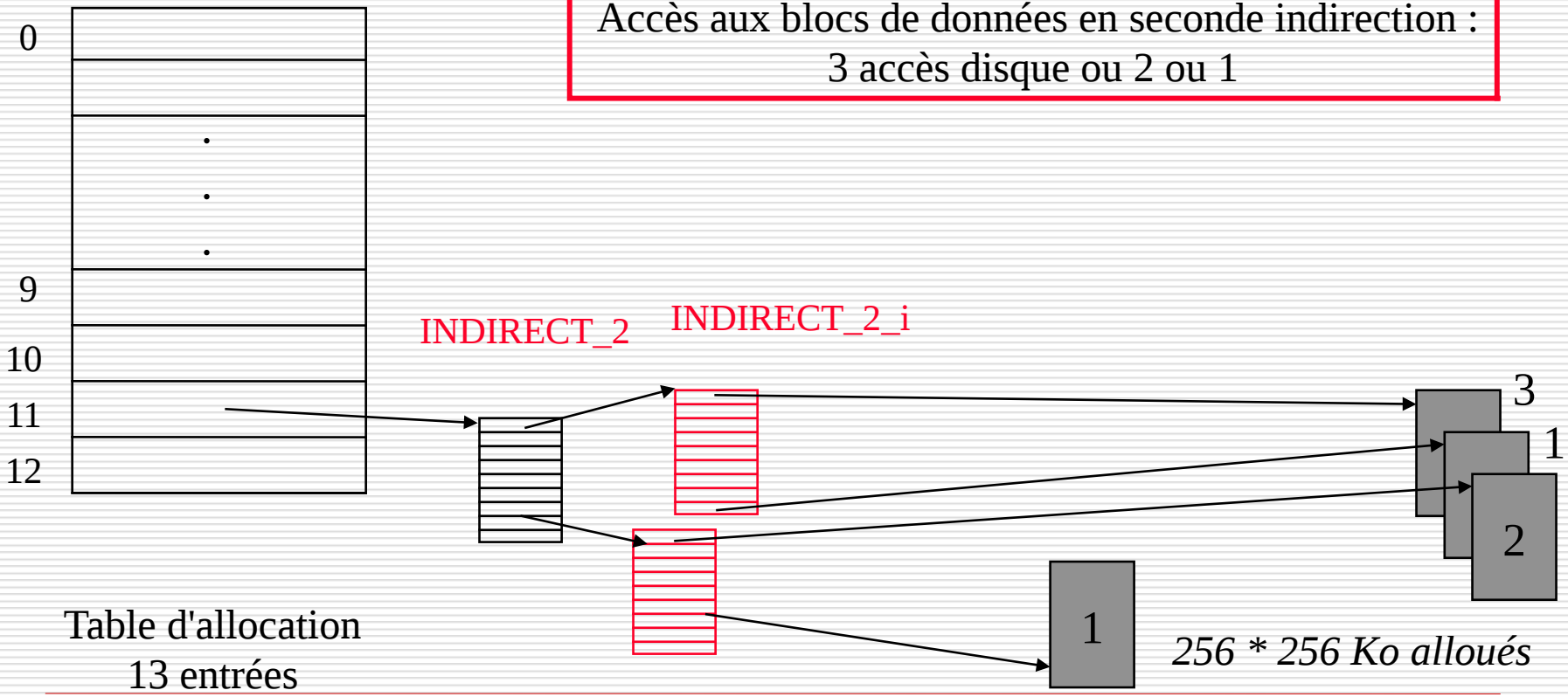
Table d'allocation
13 entrées

Accès aux blocs de données en première indirection :
2 accès disque pour le premier bloc, puis 1 accès disque

Allocation indexée : la solution Unix

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
Bloc = 1024 octets ; adresse de bloc = 4 octets → 256 entrées dans le bloc d'index

Accès aux blocs de données en seconde indirection :
3 accès disque ou 2 ou 1



Allocation indexée : la solution Unix

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 :

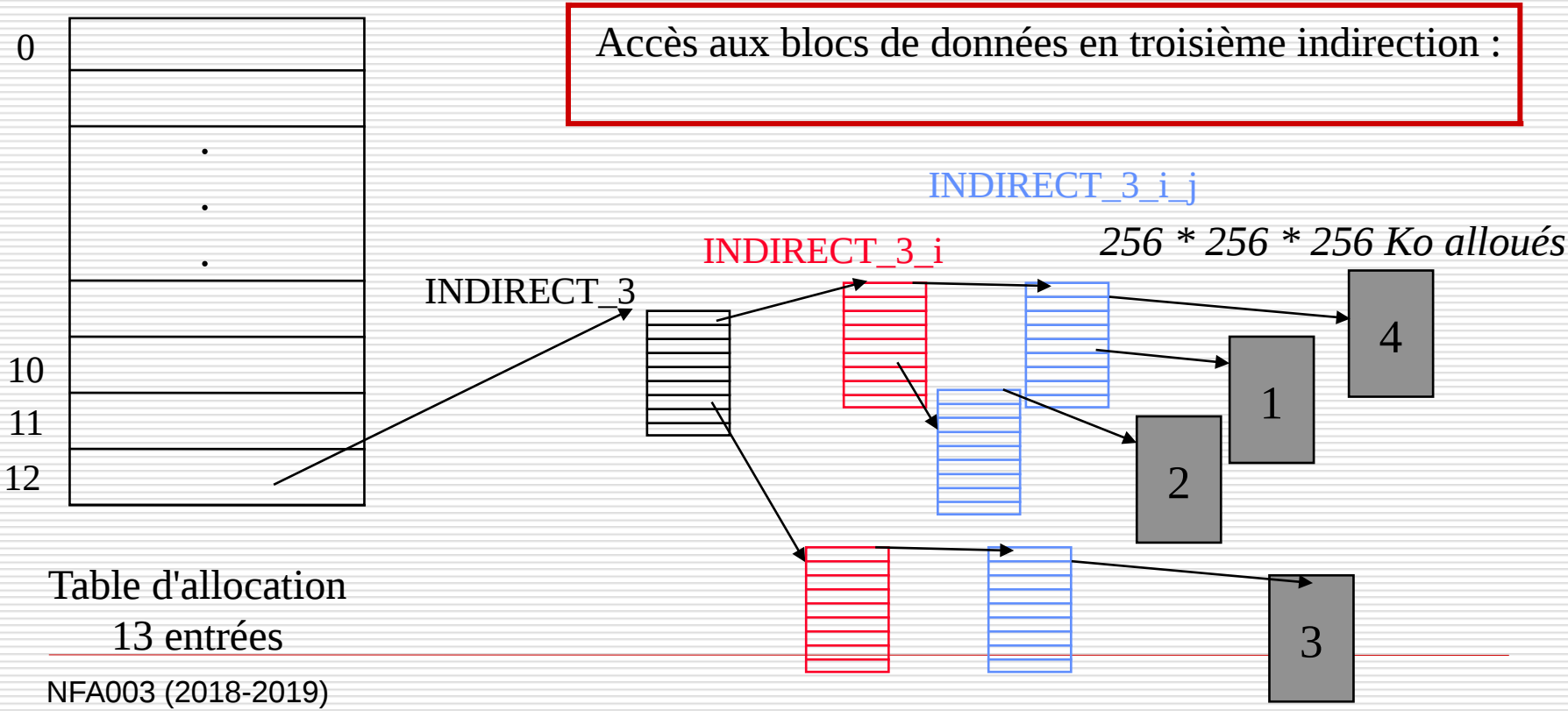
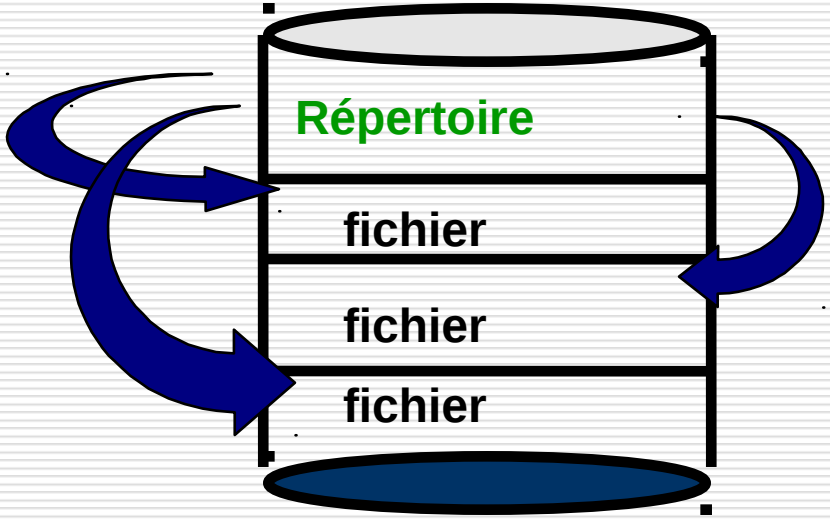


Table d'allocation
13 entrées

Le répertoire

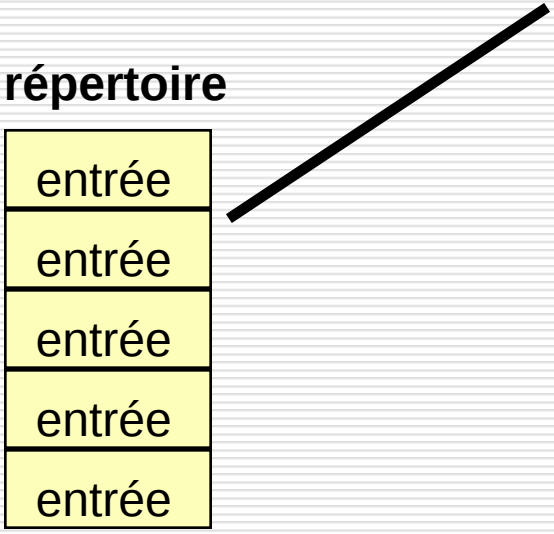
Le répertoire est une table sur le support permettant de référencer tous les fichiers existants du SGF avec leur nom et leurs caractéristiques principales.

Le répertoire stocke pour chaque fichier l'adresse des zones de données allouées au fichier



Le répertoire

Un répertoire est une zone disque réservée par le SGF.
Le répertoire comprend un certain nombre d'entrées.
Une entrée est allouée à chaque fichier du SGF

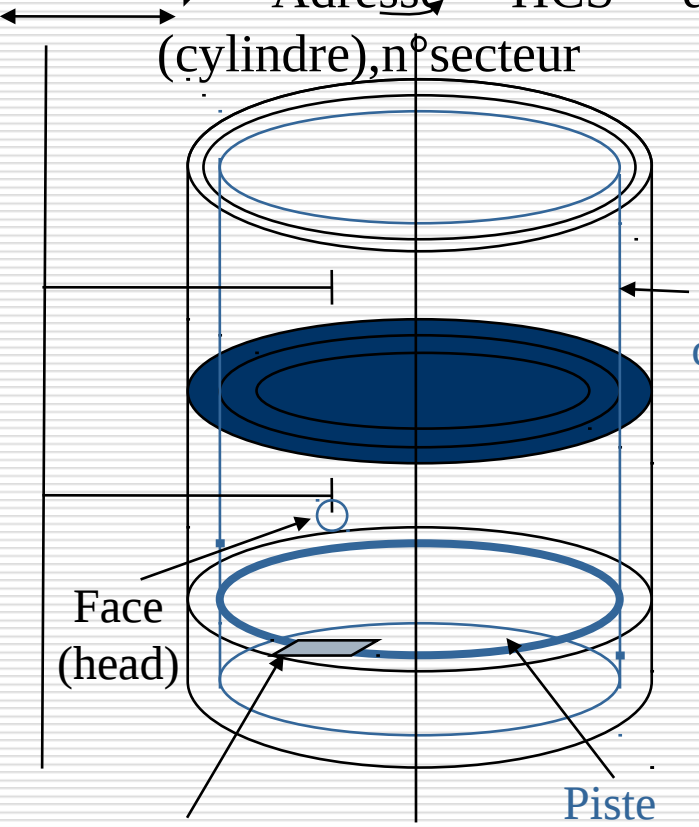


- Nom du fichier physique
- Type du fichier
- Taille du fichier
- Propriétaire
- Protection
- Date de création
- Adresse des zones de données

1 entrée : attributs du fichier physique

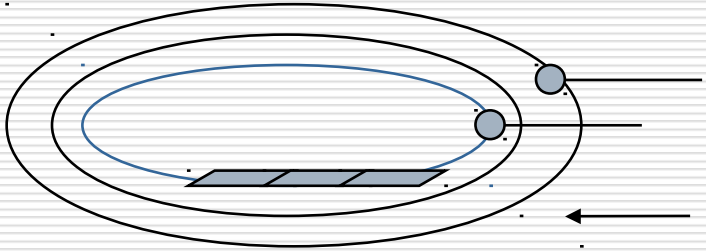
Temps d'accès à un secteur

↳ Adresse HCS d'un secteur : n°face (head), n°piste (cylindre),n°secteur

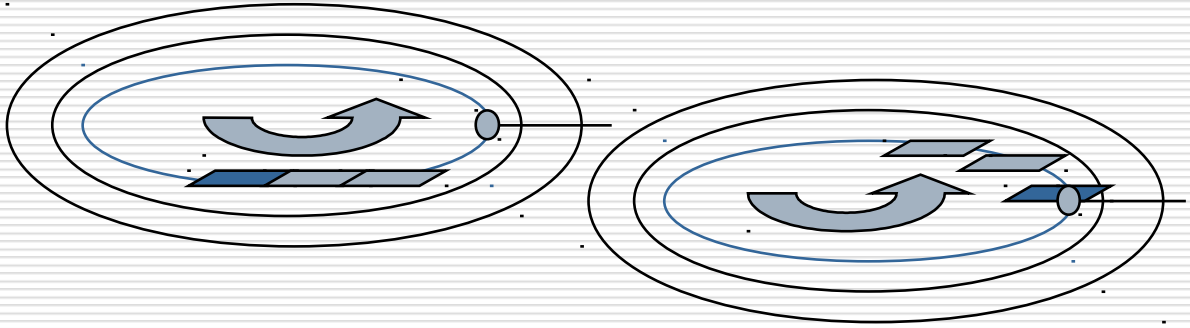


Secteur : 512 octets

Adresse HCS (3, 1, 30)



1. positionnement du bras sur la piste

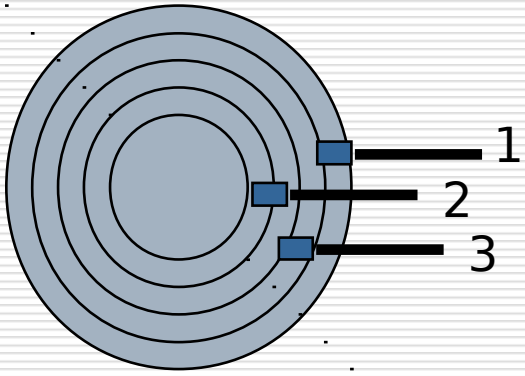


2. temps d'attente pour le passage du secteur sous la tête de lecture

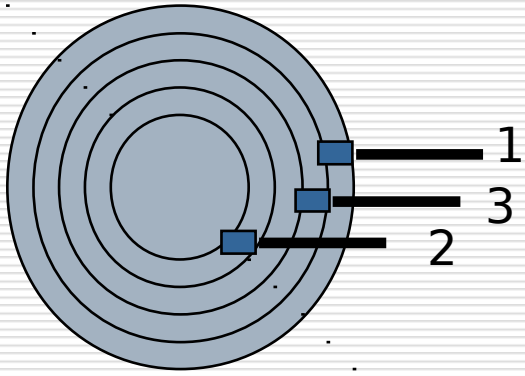
↳ le temps de positionnement du bras est le plus pénalisant : on utilise des algorithmes de services des requêtes disques pour réduire au mieux les mouvements du bras

L'ordonnancement du bras

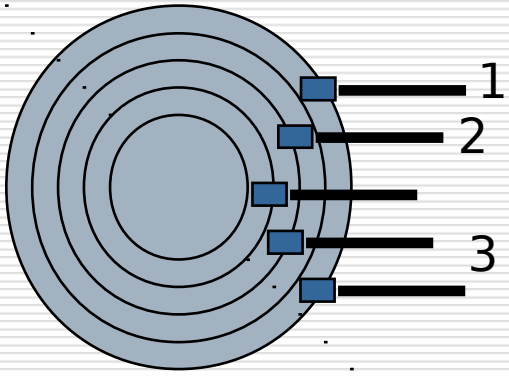
Les algorithmes d'ordonnancement du bras qui améliorent le temps d'accès :
- FCFS (First Come, First Serverd): requêtes servies séquentiellement
- SSTF (Shortest Seek Time First) : requête la plus proche d'abord
- SCAN (ou C-SCAN) : Algorithme de l'ascenseur



FCFS



SSTF



SCAN

Exercice

On considère un système Linux/Unix. Chaque utilisateur dispose d'un compte et d'un répertoire de travail qui constitue le répertoire dans lequel il peut stocker ses fichiers.
L'utilisateur **delacroi** qui appartient au groupe des enseignants en informatique **ensinf** exécute la commande **ls -l** qui permet d'afficher l'ensemble des fichiers de son répertoire.

```
lmi20: # ls -l
-rwxrw-r--  1 delacroi  ensinf  71680 Mar 25 19:28 exemple3
-rw-rw-r--  1 delacroi  ensinf 591 Mar 25 19:24 exemple3.txt
-rw-r--r--  1 delacroi  ensinf  590 Mar 25 19:24 exemple3.txt~
drwxr-xr-x  2 delacroi  ensinf  4096 Apr  5 19:27 exercices
```

A- Expliciter quels sont les droits associés au fichier **exemple3** pour les différents utilisateurs de la machine.

B- L'utilisateur **delacroi** exécute la commande suivante : **lmi20: # chmod a+w exemple3**
Que se passe-t-il ? Quels sont les droits associés au fichier **exemple3**.

C- Le fichier **exemple3** a une taille de 71680 octets. Les blocs de données du système de gestion de fichiers ont une taille de 512 octets. Un numéro de bloc occupe 4 octets.

- Combien de blocs de données comporte le fichier ?
- Combien de blocs d'index sont nécessaires ?
- Combien d'accès disque sont nécessaires pour lire de façon séquentielle ce fichier, sachant que le système de gestion de fichiers maintient un cache des blocs disque les plus récemment lus ?

Exercice

```
lmi20: # ls -l
-rwxrw-r--  1 delacroi  ensinf  71680 Mar 25 19:28 exemple3
-rw-rw-r--  1 delacroi  ensinf 591 Mar 25 19:24 exemple3.txt
-rw-r--r--  1 delacroi  ensinf  590 Mar 25 19:24 exemple3.txt~
drwxr-xr-x  2 delacroi  ensinf  4096 Apr  5 19:27 exercices
```

A- Les droits associés au fichier **exemple3** sont :

- **user (delacroi) : rwx**
- **group (ensif) : rw**
- **other : r**

B- **lmi20: # chmod a+w exemple3** met les droits en écriture (**w**) pour tous (**all : -a**)

```
-rwxrw-rw-  1 delacroi  ensinf  71680 Mar 25 19:28 exemple3
```

C- Le fichier **exemple3** a une taille de 71680 octets. Les blocs de données du système de gestion de fichiers ont une taille de 512 octets. Un numéro de bloc occupe 4 octets.

- Combien de blocs de données comporte le fichier ? 71680 / 512 = 140
- Combien de blocs d'index sont nécessaires ? On a 512/4 =128 numéro de blocs par bloc On a donc 3 blocs d'index + inode
- Combien d'accès disque sont nécessaires pour lire de façon séquentielle ce fichier, sachant que le système de gestion de fichiers maintient un cache des blocs disque les plus récemment lus ? 10 + 1 + 128 + 2 + 2 = 143 ou bien 140 (données) + 3(index) = 143

Synchronisation et communication entre processus

Principes : exclusion mutuelle et interblocage

Notion de ressources

Une ressource désigne toute entité dont a besoin un processus pour s'exécuter. Il existe deux types de ressources :

- Ressource matérielle (processeur, périphérique)
- Ressource logicielle (fichier, variable).

Une ressource est caractérisée

- par un état : libre / occupée
- par son nombre de points d'accès (nombre de processus pouvant l'utiliser en même temps)

Notion de ressources

le cnam

Utilisation d'une ressource par un processus

Trois étapes : Allocation

Utilisation

Restitution

Les phases d'allocation et de restitution doivent assurer que le ressource est utilisée conformément à son nombre de points d'accès

Une ressource critique a un seul point d'accès.

Section critique et exclusion mutuelle

le cnam

Processus
Début

Entrée Section Critique

Ressource Critique
`nb_pl_rest`

Sortie Section Critique

Fin

SECTION CRITIQUE (code d'utilisation de la ressource critique)

L'entrée et la sortie de Section Critique doivent assurer qu'à tout moment, un seul processus s'exécute en Section Critique.

→ principe de l'exclusion mutuelle.

Notion d'exclusion mutuelle entre processus
Solution logicielle : le verrou

le cnam

Un mécanisme proposé pour permettre de résoudre l'exclusion mutuelle d'accès à une ressource est le mécanisme de *verrou.* Un verrou est un objet système à deux états (libre/occupé) sur lequel deux opérations sont définies..

- `verrouiller (v)` permet au processus d'acquérir le verrou *v* s'il est libre. S'il n'est pas disponible, le processus est bloqué en attente de la ressource.
- `déverrouiller (v)` permet au processus de libérer le verrou *v* qu'il possédait. Si un ou plusieurs processus étaient en attente de ce verrou, un seul de ces processus est réactivé et reçoit le verrou.

En tant qu'opérations systèmes, ces opérations sont indivisibles, c'est-à-dire que le système qu'elles s'exécutent interruptions maquées.

Notion d'exclusion mutuelle entre processus
Solution logicielle : le verrou

le cnam

`reserver` du client 1

`V_nb_place : verrou;`

Demande de réservation client 1

`Protection de nb_pl_rest`

`nb_pl_rest > 0 = 1`

`verrouiller (V_nb_place)`
Si `V_nb_place` libre alors autoriser l'accès et mettre `V_nb_place` à l'état occupé sinon bloquer le processus

`nb_pl_rest = nb_pl_rest - 1`

`Fin protection nb_pl_rest`

`deverrouiller (V_nb_place)`
Si un processus bloqué en attente pour accéder à `nb_pl_rest`, le débloquer, Sinon `V_nb_place` libre

Notion d'exclusion mutuelle entre processus
Solution logicielle

le cnam

`V_nb_place : verrou;` -- verrou libre

`reserver` du client 1

Demande de réservation du client 1

`Verrouiller (V_nb_place)`

`V_nb_place` libre

`nb_pl_rest > 0 = 1`

`reserver` du client 1

Demande de réservation du client 2

`Verrouiller (V_Nb_Place)`

`V_Nb_Place` **occupé par le client 1**

`reserver` **du client 2 bloqué**

`nb_pl_rest = nb_pl_rest - 1`

`Deverrouiller (V_nb_place)`

Réveil de `reserver` **du client 2**

`nb_pl_rest = 0`

`deverrouiller (v_nb_Place)`

`V_nb_place : état libre`

Exercice

On considère deux processus, se partageant un verrou sem1, et une imprimante. Ils exécutent le code suivant :

```
                initialisation
                sem1.etat = libre ;
processus P1                    processus P2
repeter                         repeter
 acquerir (sem1);          acquerir(sem1);
 imprimer ("A");              imprimer ("B");
 imprimer ("1");              imprimer ("2");
 liberer (sem1);            liberer (sem1);
fin repeter;                    fin repeter;
```

Le verrou **sem1** est caractérisé par un état ; libre ou occupé. Deux opérations sont applicables au verrou :

- **acquerir(sem1)** : le processus effectuant l'opération obtient le verrou **sem1** si celui-ci est libre. Le verrou **sem1** passe alors à l'état occupé. Par contre si le verrou **sem1** est occupé, le processus appelant est bloqué.
- **liberer (sem1)** : le verrou **sem1** est relâché par le processus effectuant l'opération. Si un processus est en attente sur le verrou **sem1** celui-ci lui est attribué, sinon le verrou **sem1** repasse dans l'état libre.

A– Donner des exemples de ce qui est imprimé, et décrire le cas général en justifiant votre réponse.

Exercice

On considère deux processus, se partageant un verrou sem1, et une imprimante. Ils exécutent le code suivant :

```
                initialisation
                sem1.etat = libre ;
processus P1                    processus P2
repeter                         repeter
 acquerir (sem1);          acquerir(sem1);
 imprimer ("A");                 imprimer ("B");
 imprimer ("1");                 imprimer ("2");
 liberer (sem1);               liberer (sem1);
fin repeter;                    fin repeter;
```

A– Donner des exemples de ce qui est imprimé, et décrire le cas général en justifiant votre réponse.

```
A1A1B2B2A1B2A1 ......
```

Exercice

B– On ajoute un verrou, et on modifie le code comme suit :

```
                initialisation
                sem1.etat := libre;
                sem2.etat := libre;
processus P1                    processus P2
repeter                       repeter
 acquerir (sem1);              acquerir (sem2);
 acquerir (sem2);               acquerir (sem1);
 imprimer ("A");                imprimer ("B");
 imprimer ("1");                imprimer ("2");
 liberer (sem1);                liberer (sem2);
 liberer (sem2);                liberer (sem1);
fin repeter;                  fin repeter;
```

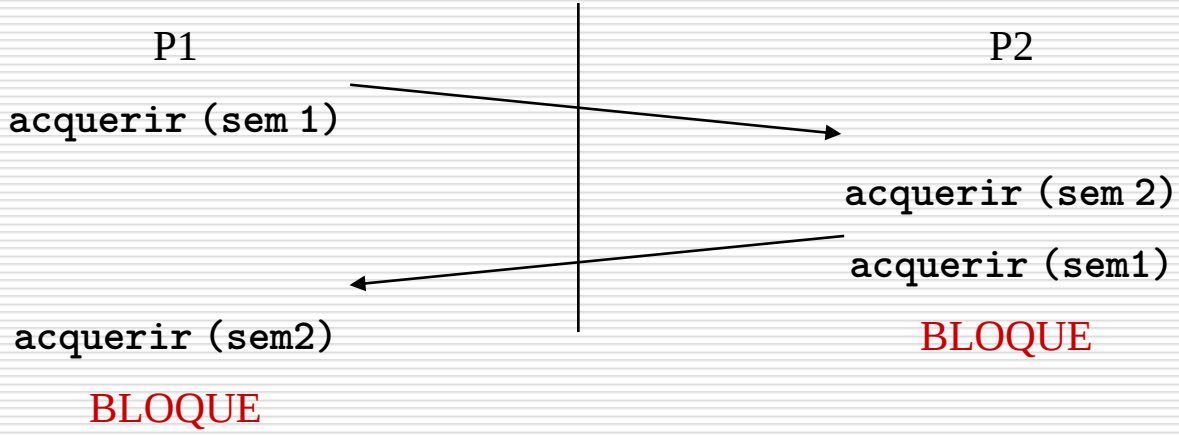
Expliquer pourquoi il est possible que les deux processus se retrouvent en interblocage.

Exercice

B– On ajoute un verrou, et on modifie le code comme suit :

```
                    initialisation
                    sem1.etat := libre;
                    sem2.etat := libre;
processus P1                          processus P2
repeter                               repeter
 acquerir (sem1);                      acquerir (sem2);
 acquerir (sem2);                      acquerir (sem1);
 imprimer ("A");                       imprimer ("B");
 imprimer ("1");                       imprimer ("2");
 liberer (sem1);                       liberer (sem2);
 liberer (sem2);                       liberer (sem1);
fin repeter;                          fin repeter;
```

Expliquer pourquoi il est possible que les deux processus se retrouvent en interblocage.



Exercice

C– On modifie le code de la façon suivante :

```
                initialisation
                sem1.etat := libre;
                sem2.etat := libre;
processus P1                    processus P2
repeter                         repeter
 acquerir (sem1);                acquerir (sem1);
 acquerir (sem2);                acquerir (sem2);
 imprimer ("A");                 imprimer ("B");
 imprimer ("1");                 imprimer ("2");
 liberer (sem1);                 liberer (sem2);
 liberer (sem2);                 liberer (sem1);
fin repeter;                    fin repeter;
```

Peut-il y avoir dans ce cas interblocage?

Exercice

C– On modifie le code de la façon suivante :

```
                initialisation
                sem1.etat := libre;
                sem2.etat := libre;
processus P1                    processus P2
repeter                       repeter
 acquerir (sem1);               acquerir (sem1);
 acquerir (sem2);               acquerir (sem2);
 imprimer ("A");                imprimer ("B");
 imprimer ("1");                imprimer ("2");
 liberer (sem1);                liberer (sem2);
 liberer (sem2);                liberer (sem1);
fin repeter;                   fin repeter;
```

Peut-il y avoir dans ce cas interblocage?

Non, car l'ordre de réquisition des ressources est le même.

Interblocage

Interblocage : Ensemble de n processus attendant chacun une ressource déjà possédée que par un autre processus de l'ensemble

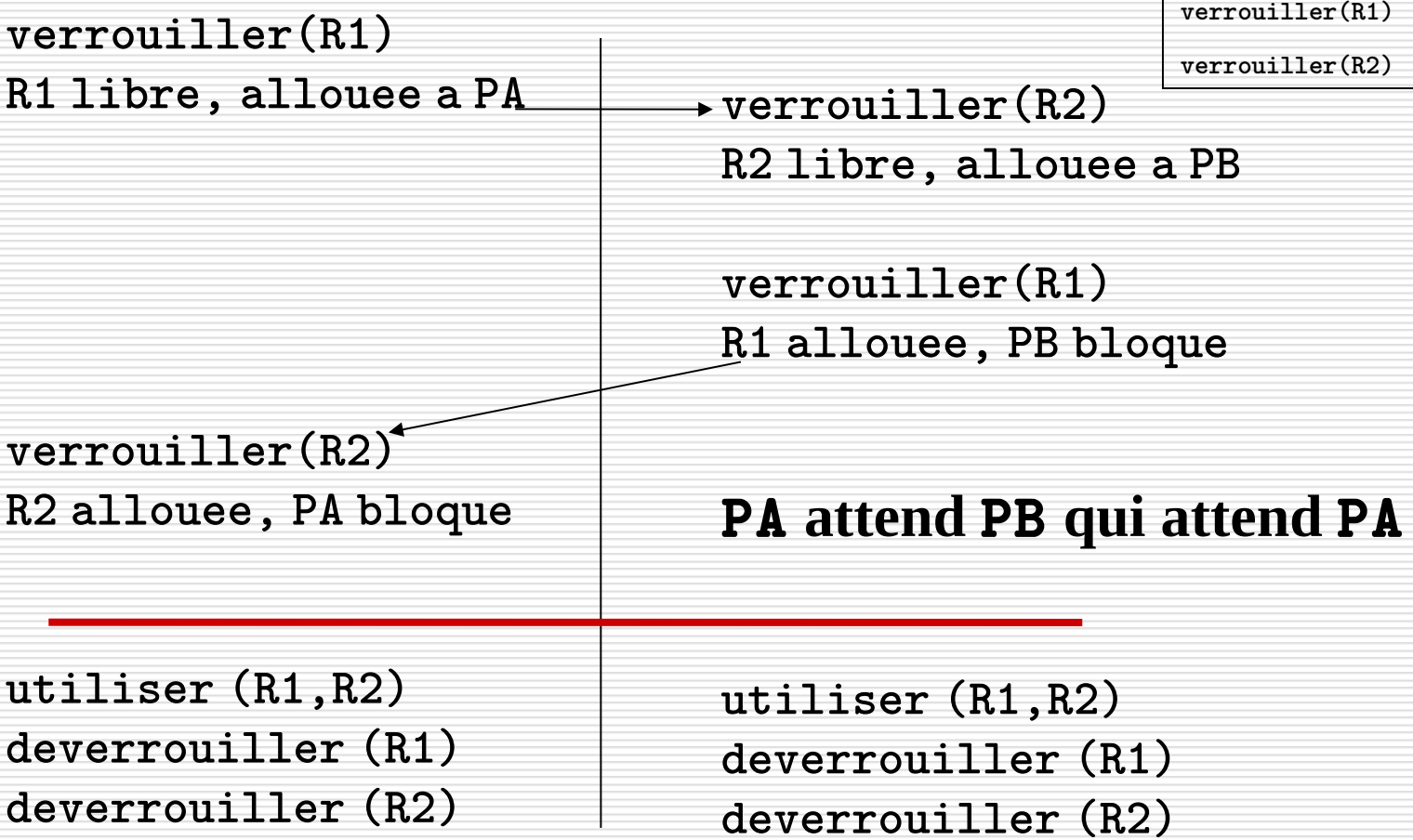
Soient deux ressources, R1 et R2 à un seul point d'accès.

On suppose R1 et R2 libres

Processus PA	Processus PB
verrouiller(R1)	verrouiller(R2)
verrouiller(R2)	verrouiller(R1)
utiliser (R1,R2)	utiliser (R1,R2)
deverrouiller (R1)	deverrouiller (R1)
deverrouiller (R2)	deverrouiller (R2)

Interblocage

Processus PA	Processus PB
verrouiller(R1)	verrouiller(R2)
verrouiller(R2)	verrouiller(R1)



Aucun processus ne peut poursuivre son exécution

Politiques de prévention

Imposer un ordre total sur l'allocation des ressources :

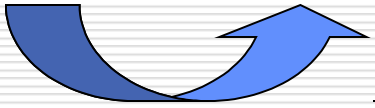
→ tout processus doit demander l'accès aux ressources selon un ordre préétabli :

Exemple : verrouiller R1 avant R2

```
verrouiller(R1)          verrouiller(R2)
verrouiller(R2)          verrouiller(R1)

utiliser (R1,R2)         utiliser (R1,R2)

deverrouiller (R1)       deverrouiller (R1)
deverrouiller (R2)       deverrouiller (R2)
```



```
verrouiller(R1)
verrouiller(R2)

utiliser (R1,R2)

deverrouiller (R1)
deverrouiller (R2)
```