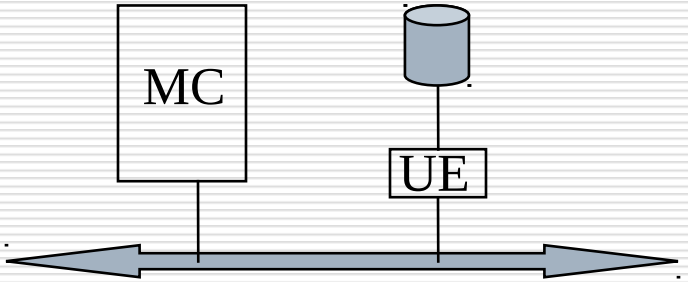
Système de gestion de fichiers

- 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

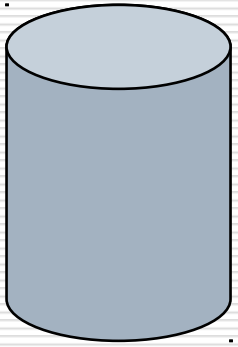
Niveau utilisateur

Interface : fonctions du SGF

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

- allocation des fichiers sur le support de masse
- répertoire

↳ Fichier Physique



Introduction : un exemple

```
Pour les élèves
 Lire (nom_élève);
 Lire (note_examen);
 Écrire (table);
fait
Sauvegarder(mon_fichier)
```

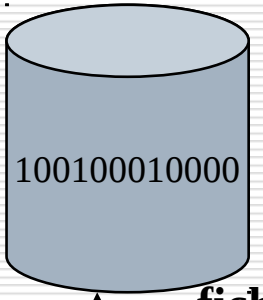
Programme instructions

Fichier logique : suite d'enregistrements
mon_fichier

Toto	Dupont	César	Cléopâtre
12	20	09	15

toto	**12**
dupont	**20**
cesar	**09**
cléopâtre	**15**

Programme données



Fichier physique
Structure selon
Allocation

fichier_notes_nfa004

Le fichier logique

Notion de fichier logique

Le fichier logique est la vue de l'utilisateur de l'ensemble des données mémorisées sur le support de masse.

Le fichier logique est :

- un type de donnée (programmation)
- un ensemble de données groupées sous forme d'enregistrements

Notion de fichier logique

En programmation, un fichier logique est un type de donnée sur lequel peuvent être appliquées des opérations spécifiques.

```
Program toto;

fichier : file of ...

créer (fichier, mode)
ouvrir (fichier, mode)


fermer (fichier)
detruire (fichier)
```

Représentation du fichier interne au programme

Liaison via le SGF avec le fichier physique
Liaison `fichier` avec fichier_notes_nfa004

Rupture de la Liaison avec le fichier physique

Notion de fichier logique

Un fichier logique est un ensemble d'enregistrements, désigné par un nom et accessible via des fonctions d'accès.

```
Type element = record
    nom-eleve : char;
    note : entier;
end;
```

enregistrement

Toto 12	Dupont 20	César 09	Cléopâtre 15

```
mon_fichier : file of element
```

Nom logique : **mon_fichier**

fonctions d'accès

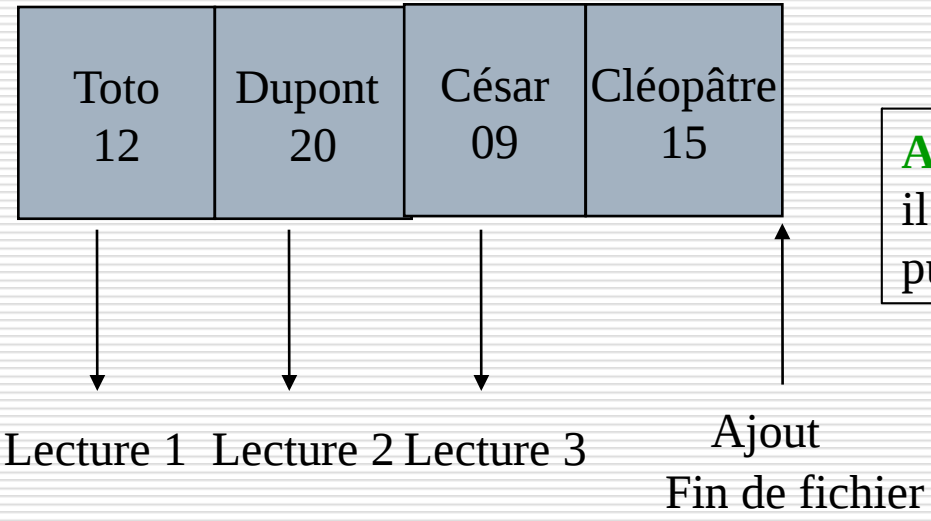
lire (enregistrement)
écrire (enregistrement)
insérer (enregistrement)
supprimer (enregistrement)

Organisation définissant sa structure logique : le mode d'accès

Fichier à mode d'accès séquentiel

Les enregistrements du fichier ne peuvent être accédés que les uns à la suite des autres :

- Ouverture du fichier : positionne sur le premier enregistrement
- Opération de lecture : délivre l'enregistrement courant et se positionne sur le suivant
- Opération d'ajout : obligatoirement en fin de fichier

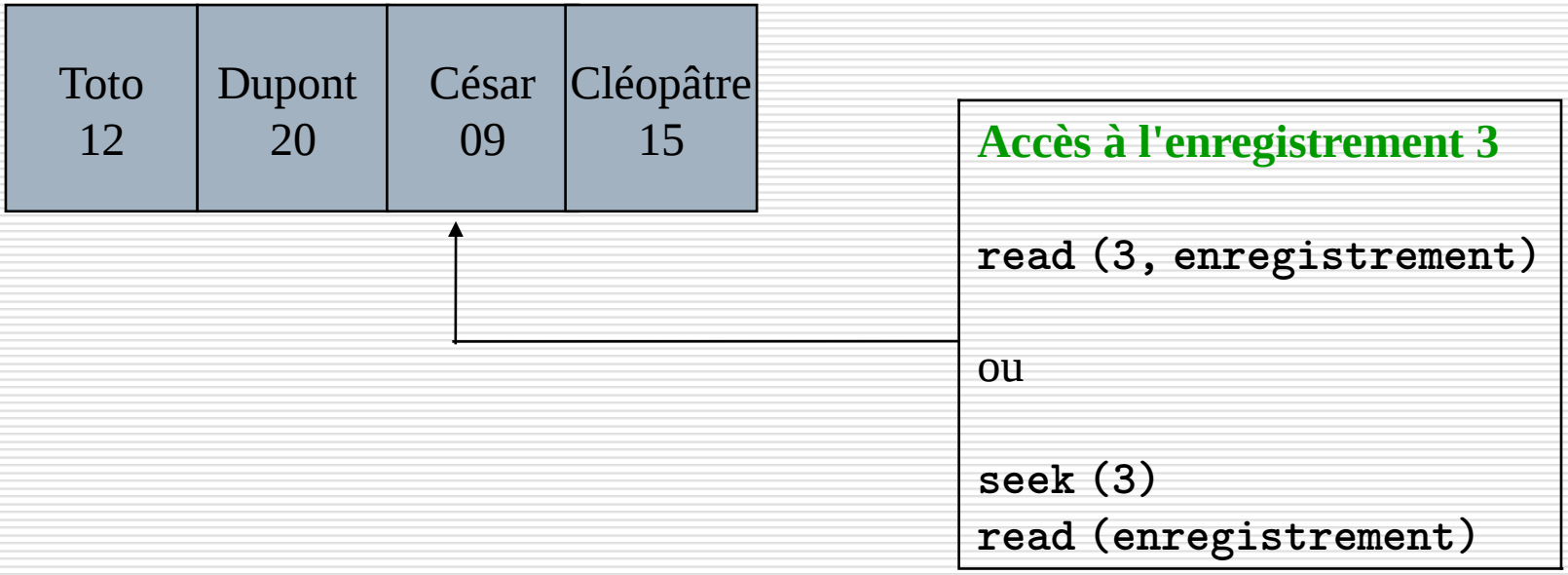


Accès à l'enregistrement 3 :
il faut lire d'abord l'enregistrement 1, puis l'enregistrement 2.

Fichier à mode d'accès direct

Un enregistrement est accédé en fonction de sa position relative dans le fichier :

- Une opération de lecture, écriture, ajout ou destruction d'enregistrement spécifie le numéro i (position relative) de l'enregistrement accédé
- Une opération spécifique de déplacement permet de pointer l'enregistrement i



Exemple : les fichiers en langage pascal

le cnam

```
program acces_fichiers
enrg_note = record nom_eleve : string;
                   note : integer;
            end;
t_fichier_notes = file of enrg_note;
var  mon_fichier : t_fichier_notes;
begin
 assign(mon_fichier,'fichier_notes_nfa004');
 reset ( mon_fichier );   (* ouvrir le fichier *)
 while not eof ( nom_fichier ) do
 (* parcourir les élts du fichier - accès séquentiel*)
 begin
  get ( mon fichier );
 end;
 (* accès direct : aller à l'élément 6 *)
 seek (mon_fichier,6) ;
 get (mon_fichier) ;
 (* fermeture automatique en fin de programme *)
end.
```

Enregistrement

Déclaration fichier logique

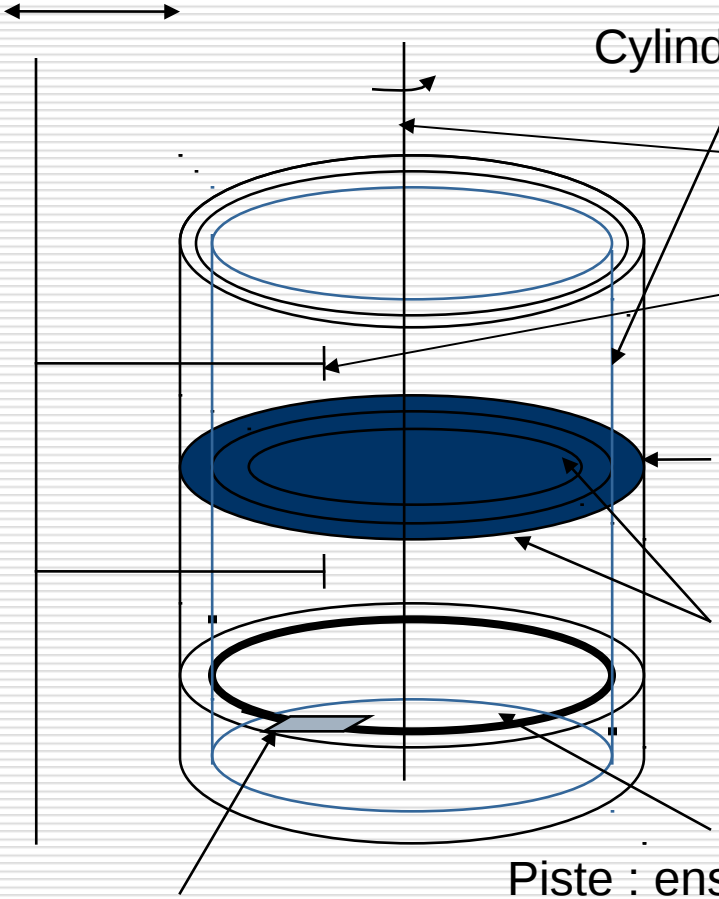
Mise en correspondance

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

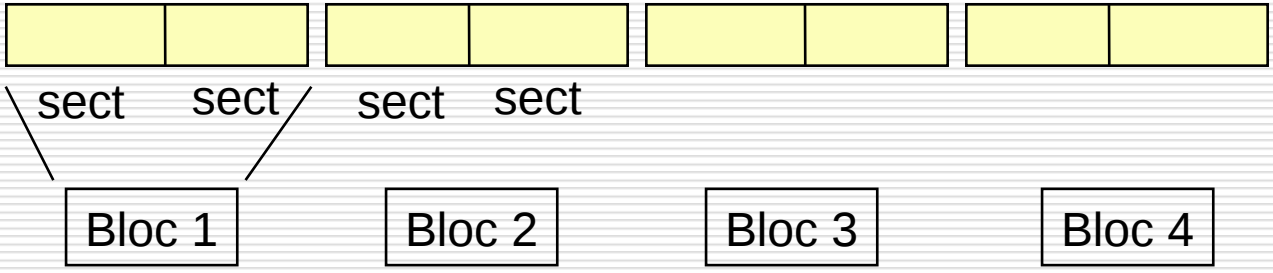
Secteur : plus petite unité d'information accessible
32 à 4096 octets

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

```
Pour les eleves
 Lire (nom_eleve);
 Lire (note_examen);
 Ecrire (table);
fait
Sauvegarder
(mon_fichier)
```

Programme instructions

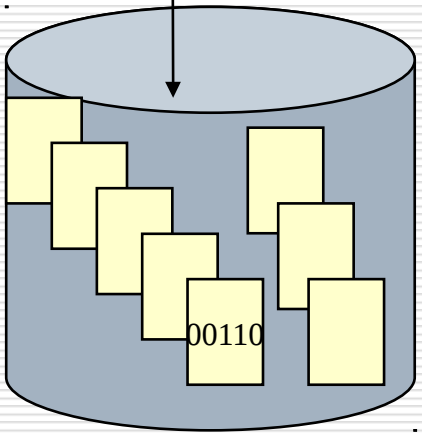
Fichier logique : suite d'enregistrements

mon_fichier

Toto 12	Dupont 20	César 09	Cléopâtre 15

toto	12
dupont	20
cesar	09
cléopâtre	15

Programme donnée



Fichier physique
Ensemble de blocs physiques
Fichier_notes_nfa004

Fichier physique

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

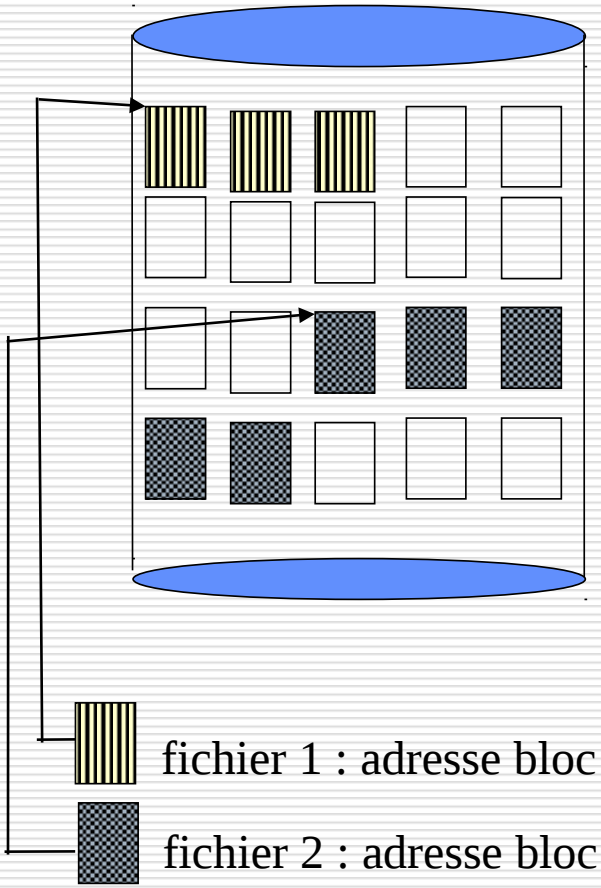
Il est constitué d'un ensemble de blocs physique. Il existe plusieurs méthodes d'allocation des blocs physiques :

- 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

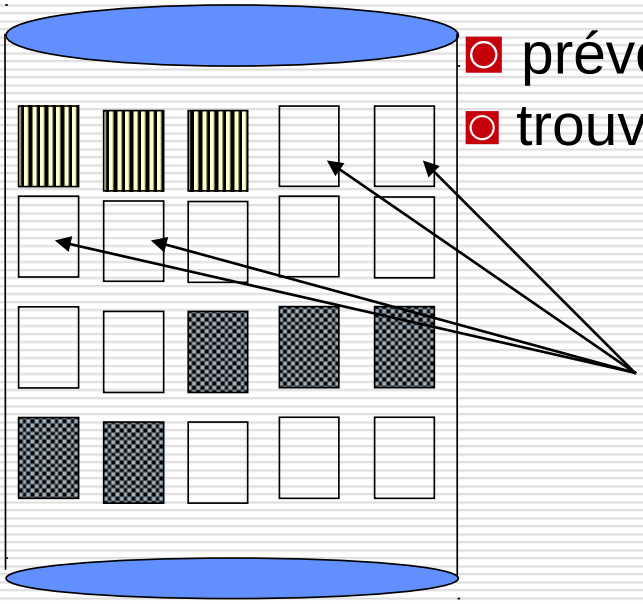
fichier 1 : adresse bloc 1, longueur 3 blocs

fichier 2 : adresse bloc 13, longueur 5 blocs

Allocation contiguë

création d'un nouveau fichier : il faut allouer un nombre de blocs suffisants dépendant de la taille du fichier :

- prévoir cette taille
- trouver un trou suffisant (First Fit, Best Fit)



Fichier 3 : 4 blocs

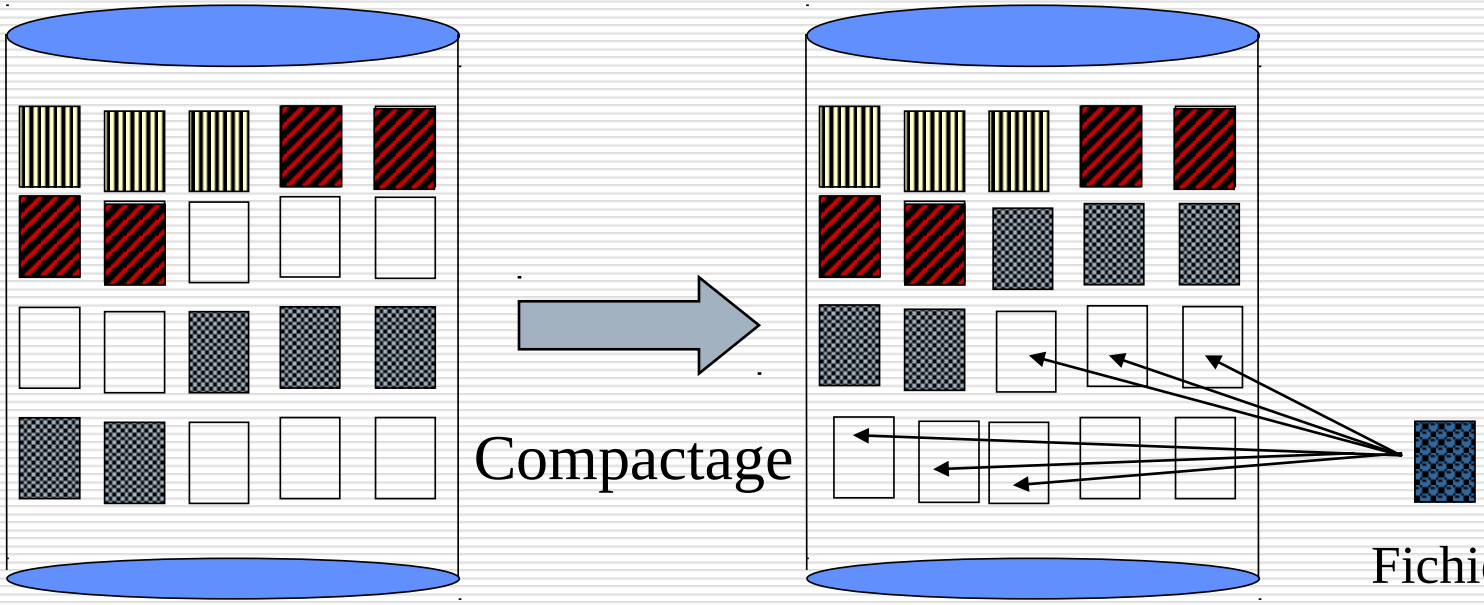
Fichier 4 : 6 blocs

fichier 1 : adresse bloc 1, longueur 3 blocs

fichier 2 : adresse bloc 13, longueur 5 blocs

Allocation contiguë

Trouver un trou de taille suffisante



fichier 1 : adresse bloc 1, longueur 3 blocs

fichier 2 : adresse bloc 13, longueur 5 blocs

fichier 3 : adresse bloc 3, longueur 4 blocs

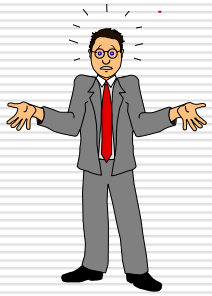
adresse bloc 1, longueur 3 blocs

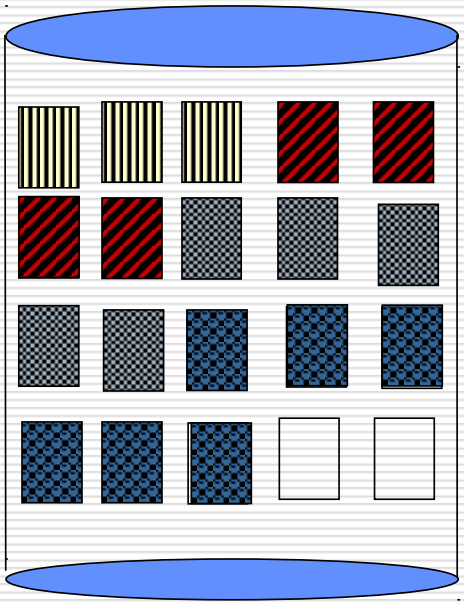
adresse bloc 8, longueur 3 blocs

adresse bloc 4, longueur 3 blocs

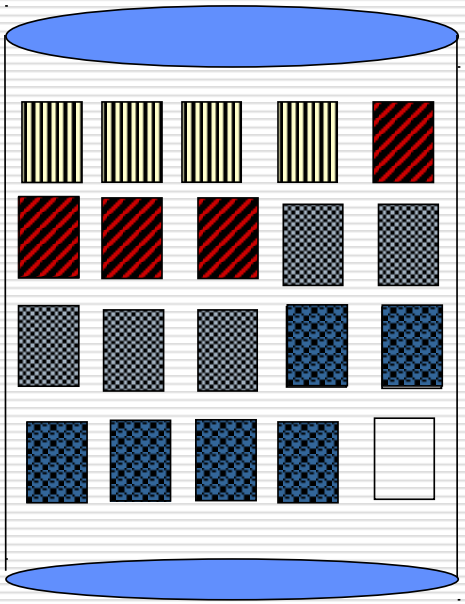
Allocation contiguë

Étendre le fichier 1 avec un bloc de données





Déplacer les fichiers COUTEUX !



fichier 1 : adresse bloc 1, longueur 4 blocs

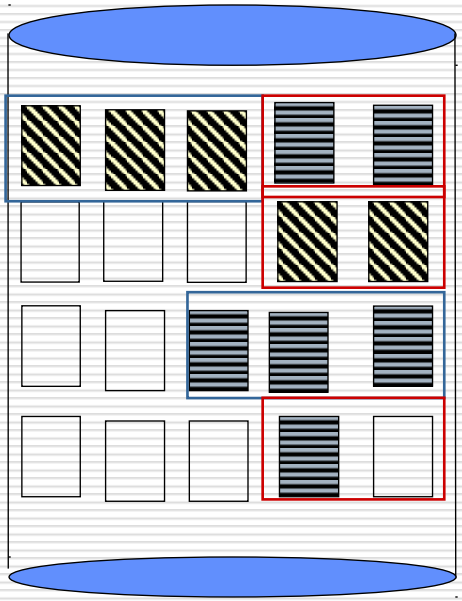
fichier 1 : adresse bloc 1, longueur 3 blocs

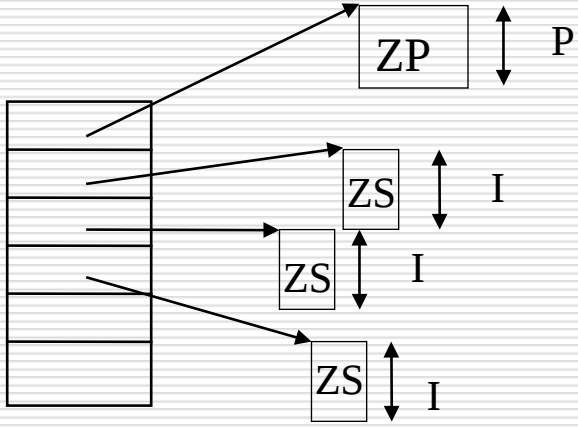
Générer une erreur

Allocation par zones

Un fichier est constitué de plusieurs zones physiques disjointes (exemple système MVS IBM) :

- une zone primaire allouée à la création
- des zones secondaires (extensions) allouées au fur et à mesure des besoins
- Chaque zone est allouée de façon indépendante

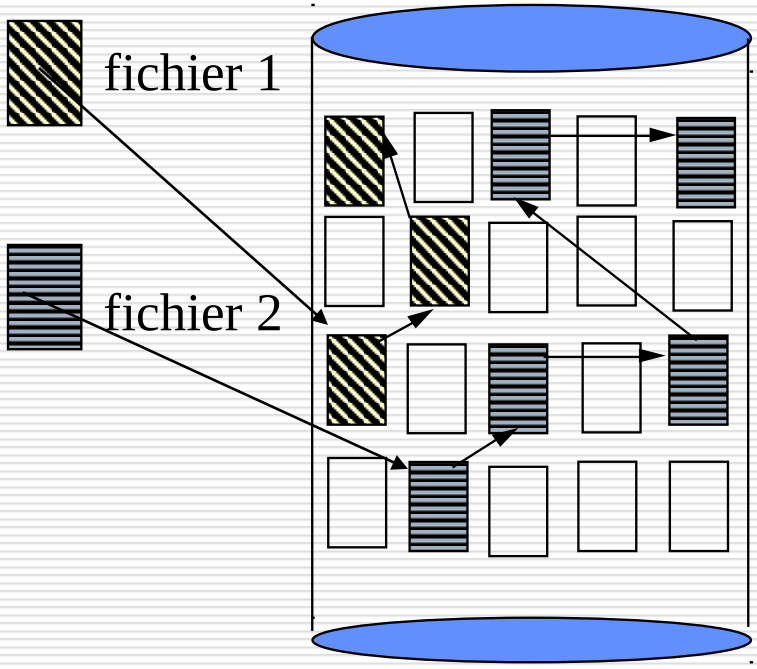




↳ Problèmes précédents atténués mais toujours existants

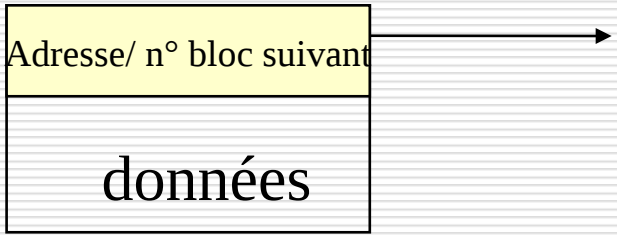
Allocation par bloc chainée

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

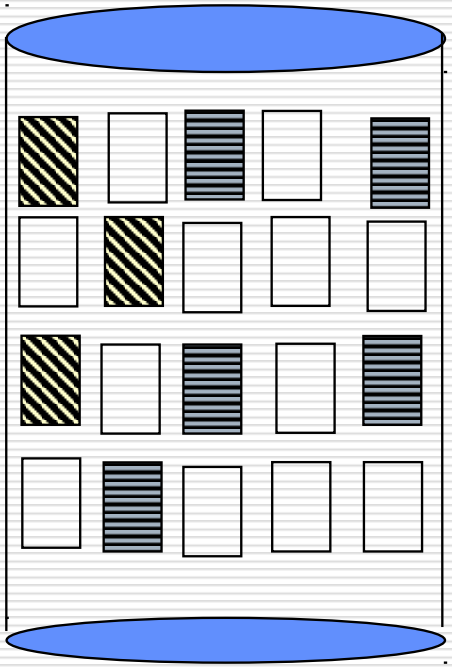
- Difficultés :
  Mode séquentiel seul
  Le chaînage du bloc suivant occupe de la place dans un bloc

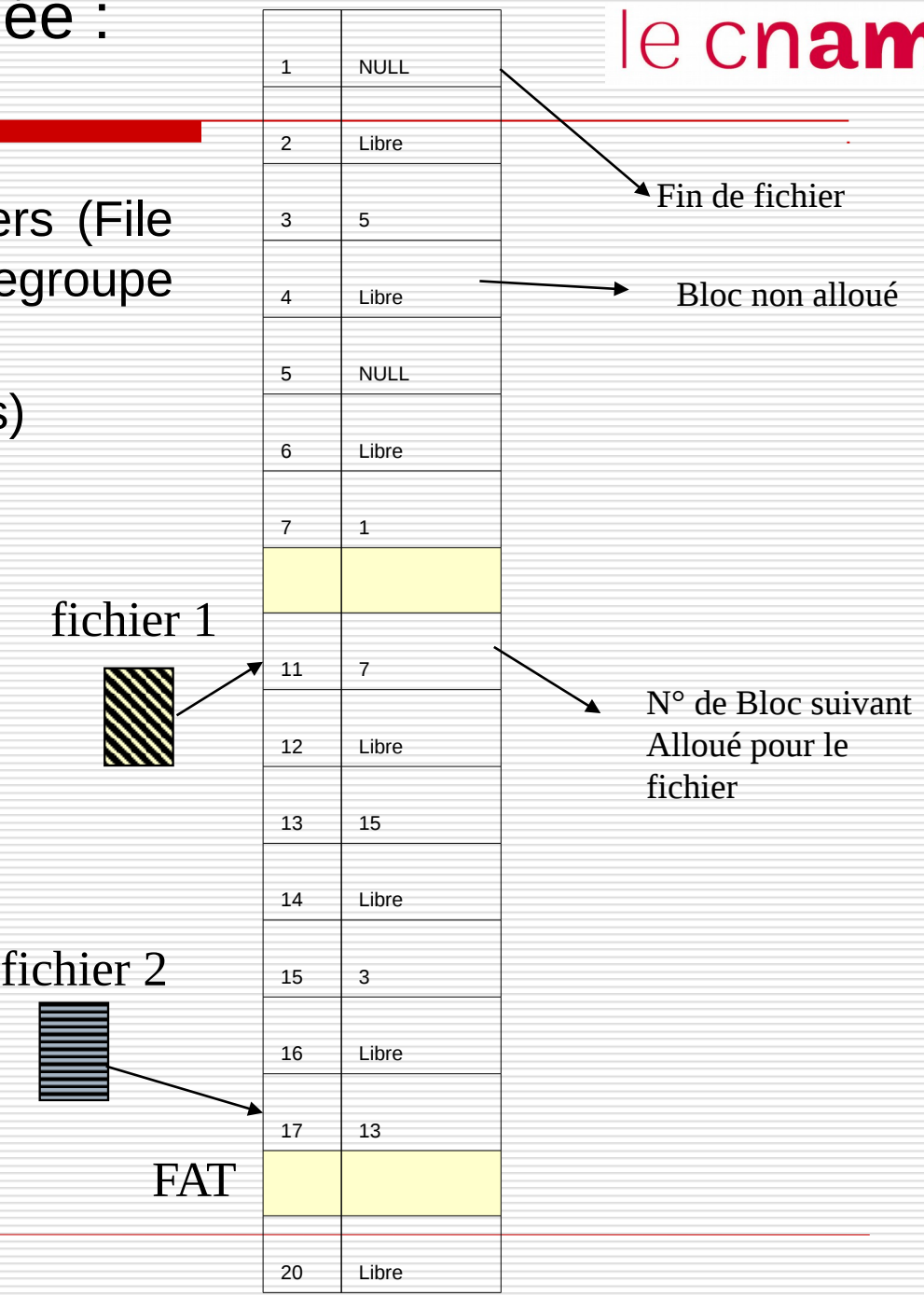


Allocation par bloc chainée : variante

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

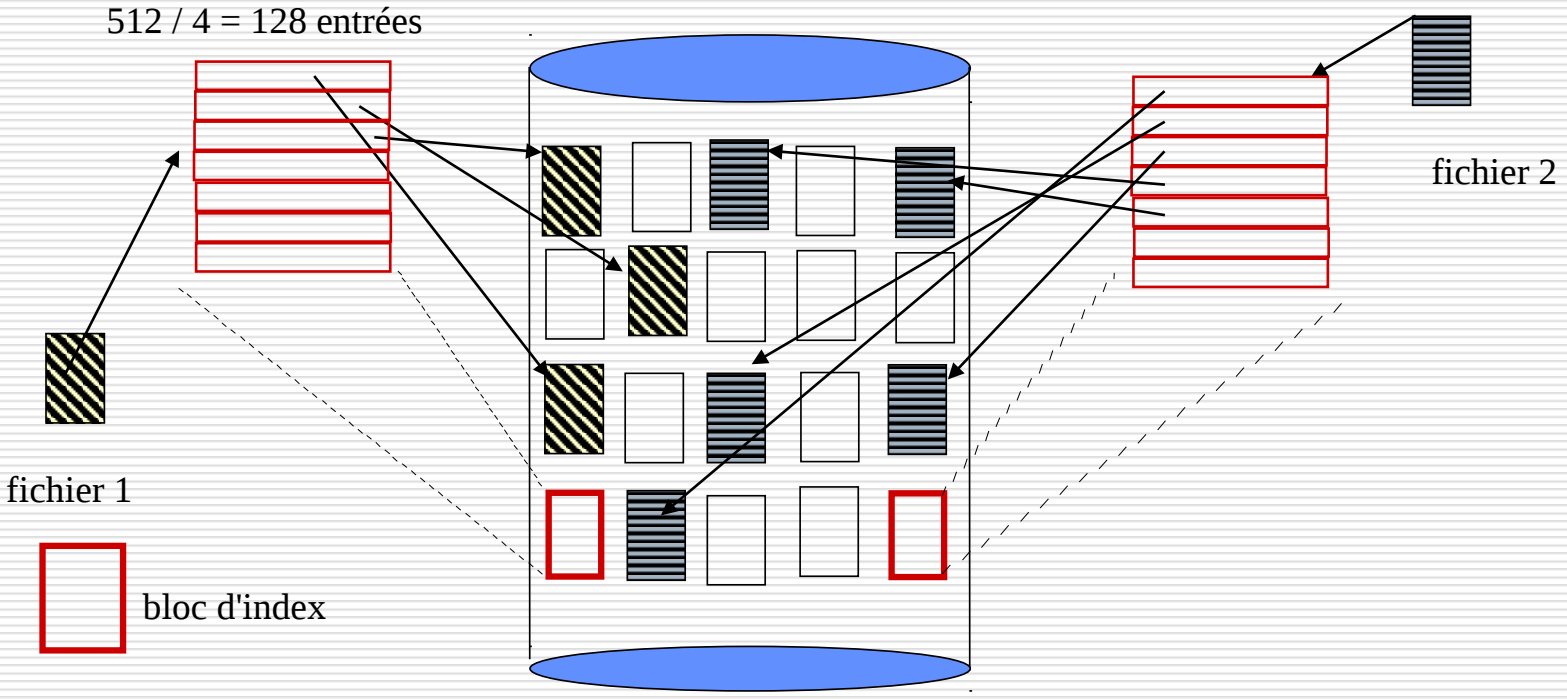
(exemple systèmes Windows)





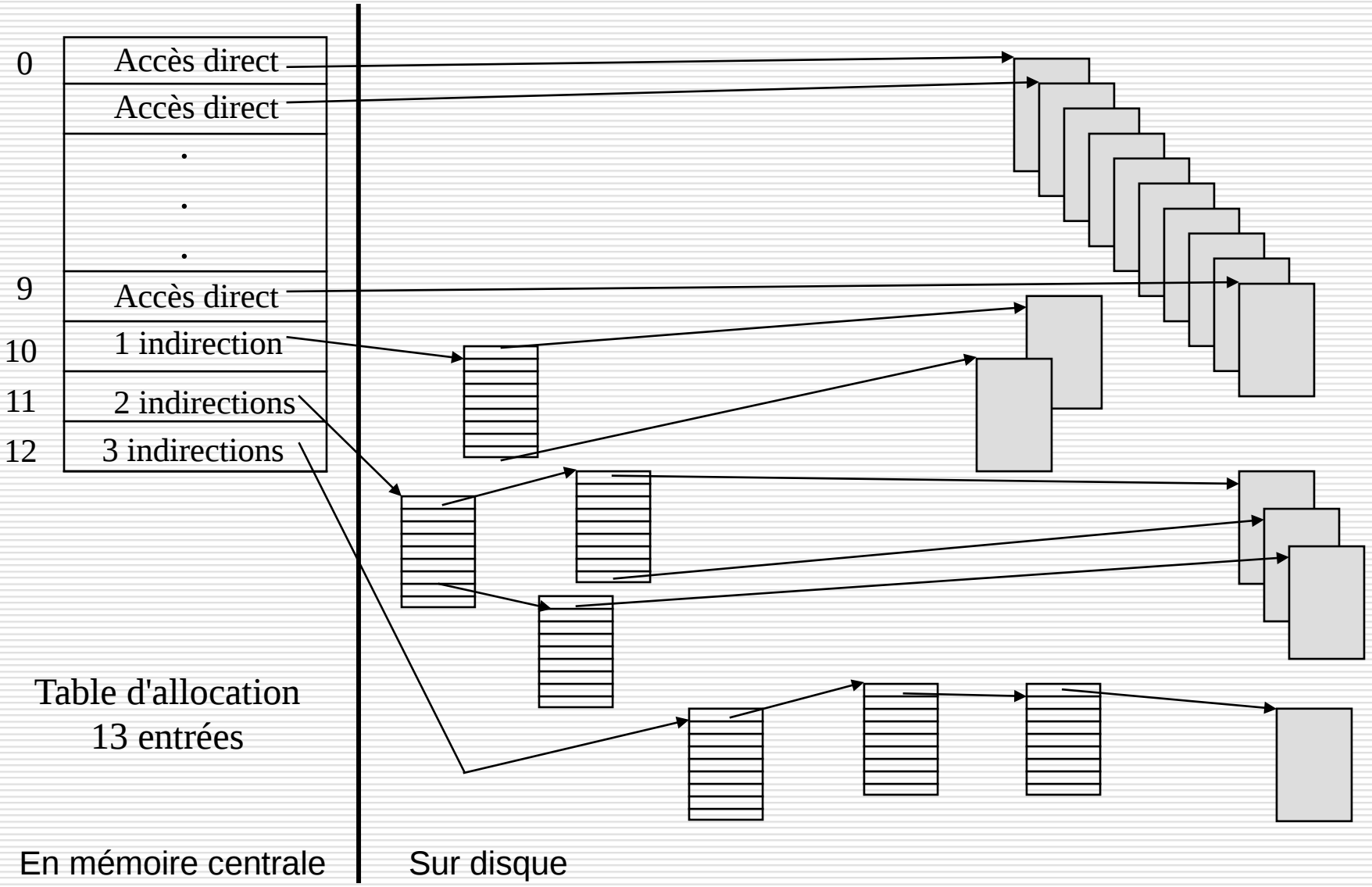
Allocation indexée

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/Linux



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

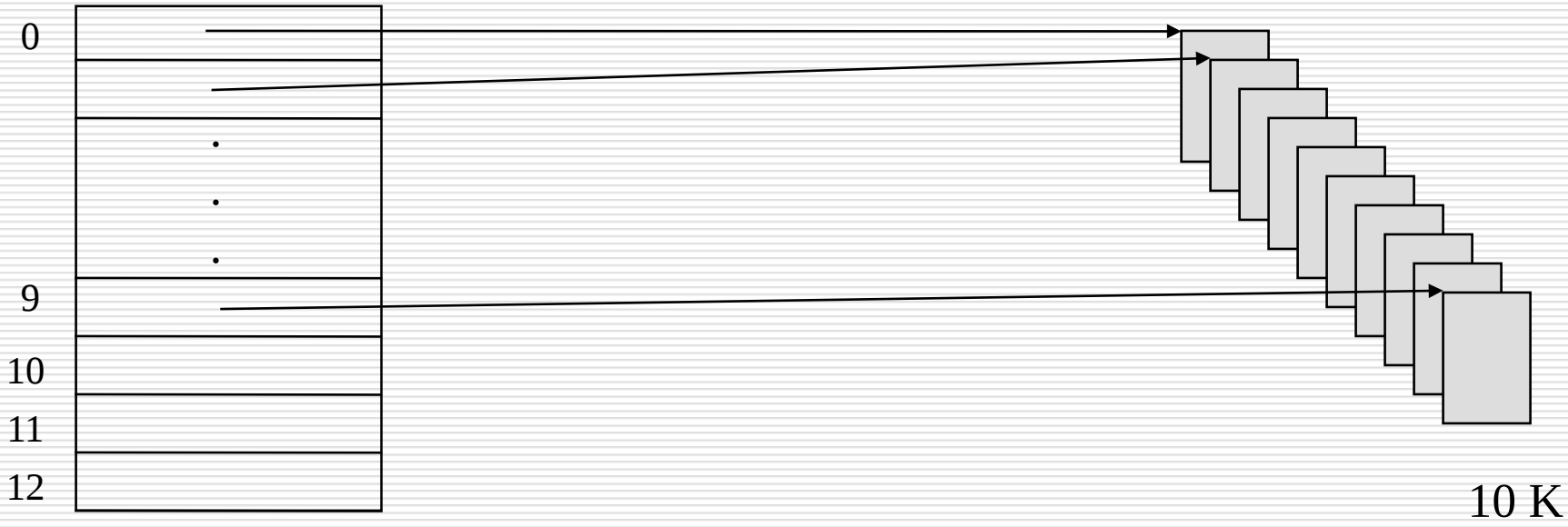


Table d'allocation
13 entrées

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

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

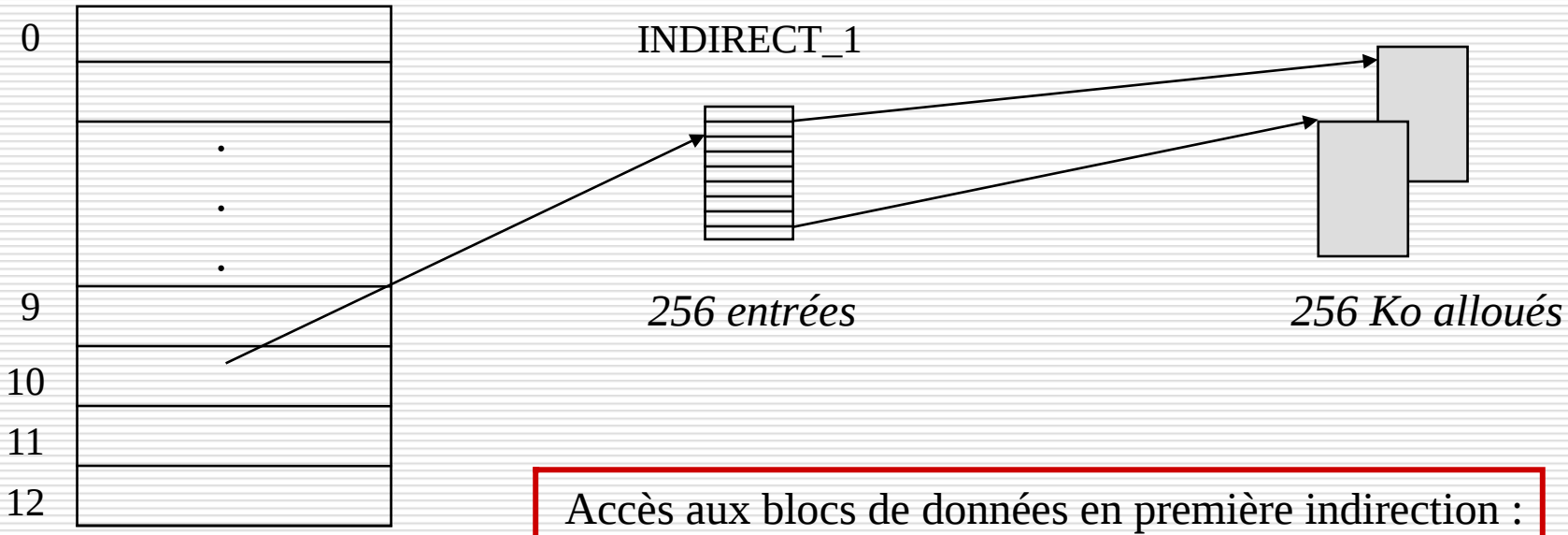


Table d'allocation
13 entrées

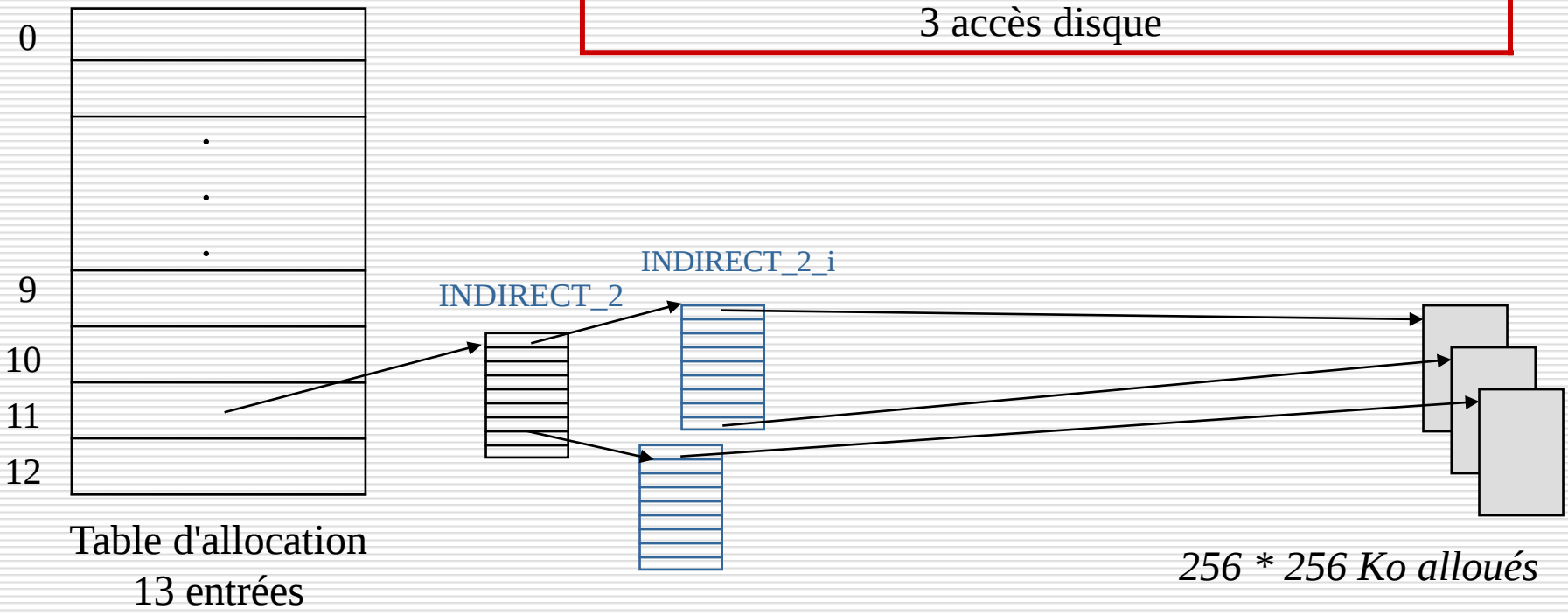
Accès aux blocs de données en première indirection :
2 accès disque

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

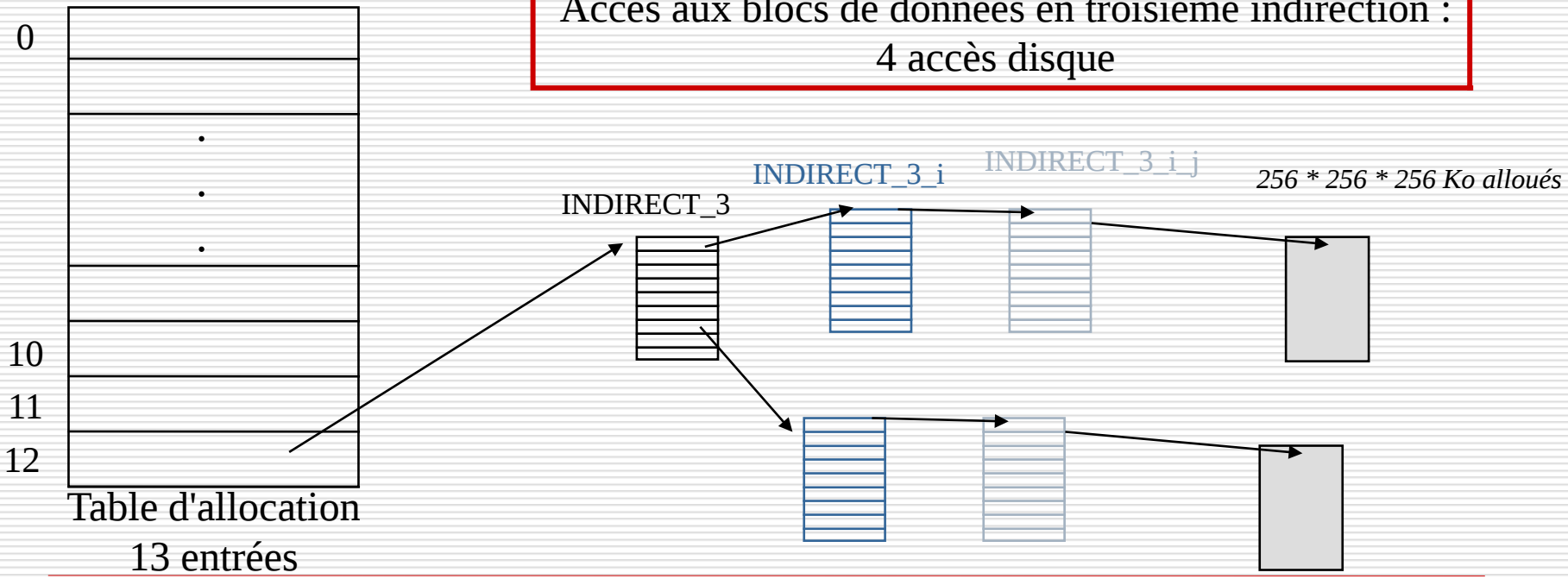


Allocation indexée : la solution Unix

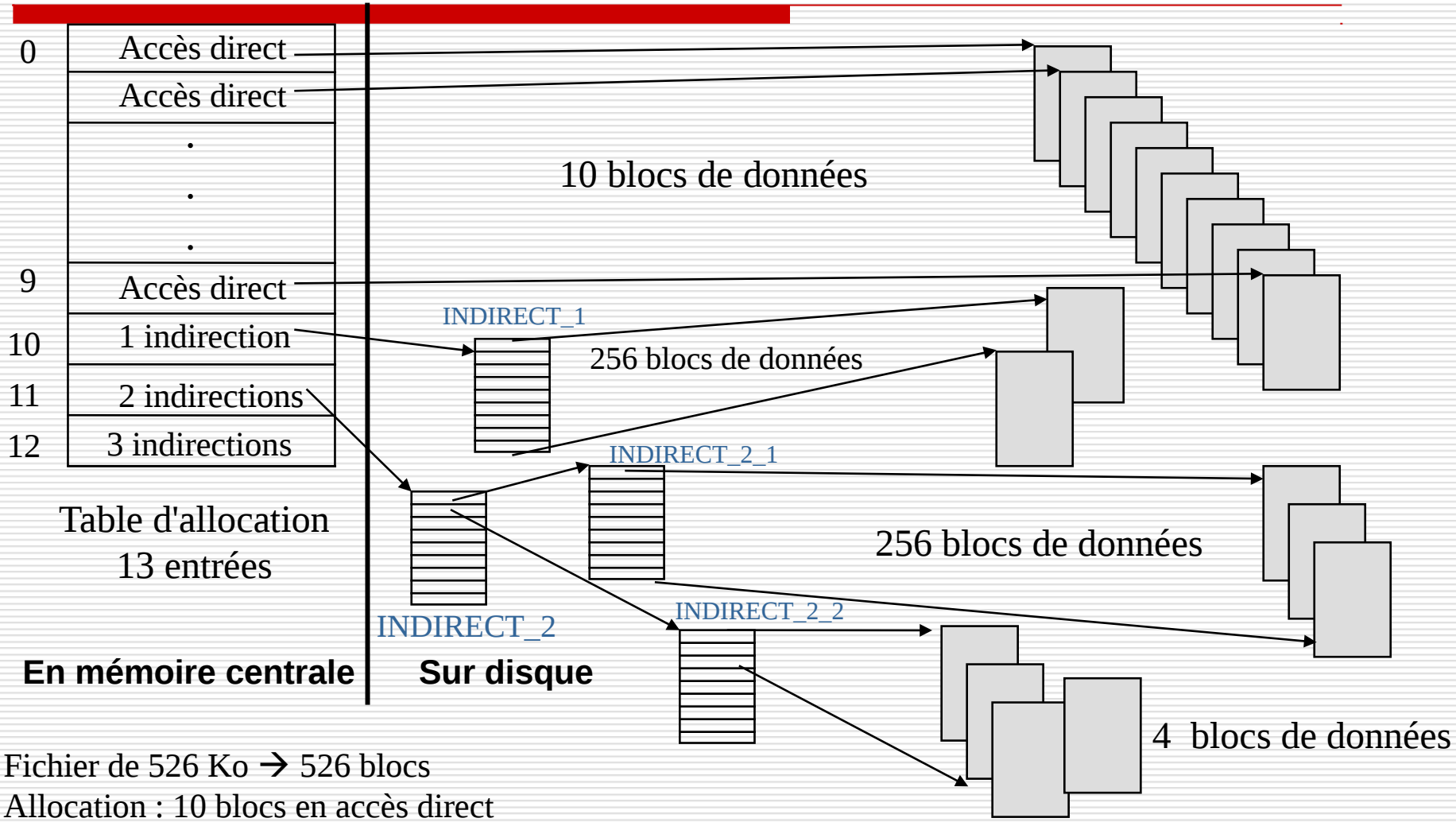
La 13ème entrée de la table contient l'adresse d'un bloc d'index INDIRECT_3. Ce bloc d'index contient des adresses de blocs d'index INDIRECT_3_i. Chaque bloc d'index INDIRECT_3_i contient des adresses de blocs d'index INDIRECT_3_i_j. Chaque bloc d'index INDIRECT_3_i_j contient des adresses de blocs de 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 : 4 accès disque



Allocation indexée : un exemple



Fichier de 526 Ko → 526 blocs

Allocation : 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. 2 blocs d 'index INDIRECT_2_1 et INDIRECT_2_2 sont nécessaires à ce niveau

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

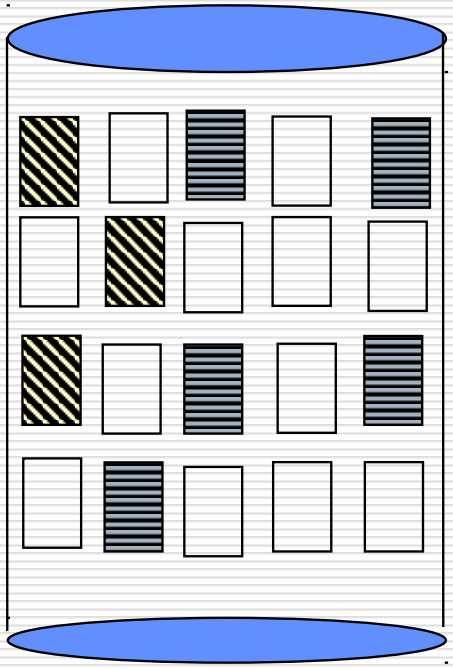
Il existe différentes représentations possibles de l'espace libre

- vecteur de bits
- liste chaînée 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



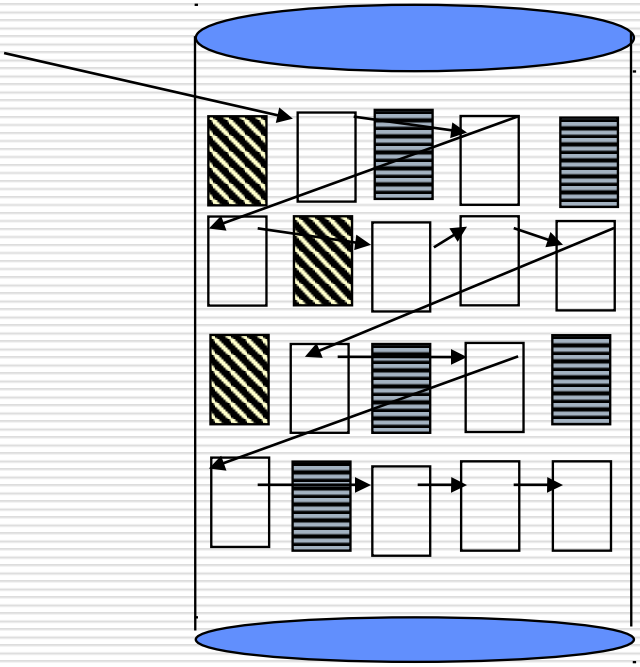
01010101110101010111

- Facilité de trouver n blocs libres consécutifs
- Système Macintosh

Gestion de l'espace libre par liste chaînée

La liste d'espace libre est représentée par une liste chaînée des blocs libres

Liste des blocs libres

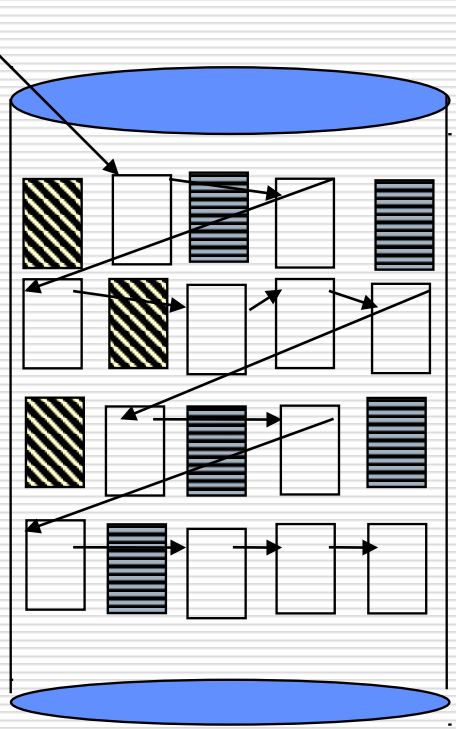


- Parcours de la liste coûteux
- Difficile de trouver un groupe de blocs libres
- Variante par comptage

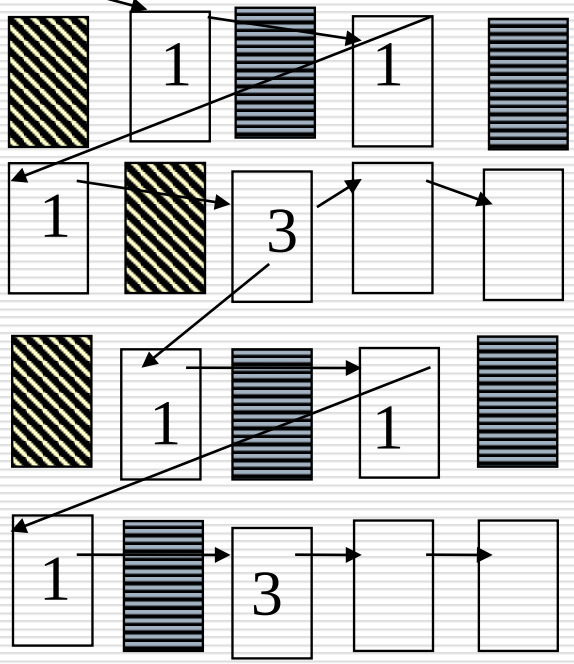
Gestion de l'espace libre par liste chaînée: variante avec comptage

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.

Liste des blocs libres

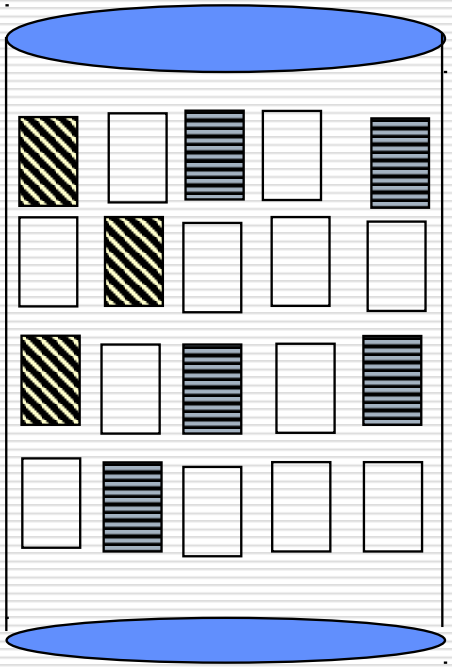


Liste des blocs libres



Gestion de l'espace libre

La FAT intègre directement la gestion de cet espace.



fichier 1

fichier 2

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
17	13
20	Libre

FAT

Fin de fichier

Bloc non alloué

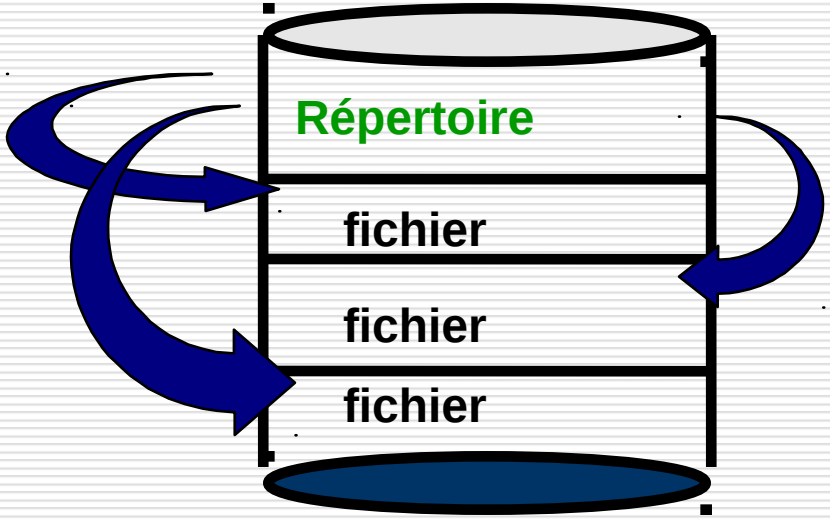
N° de Bloc suivant alloué pour le fichier

Correspondance
fichier logique - fichier physique
Désignation des fichiers : le répertoire

Le répertoire

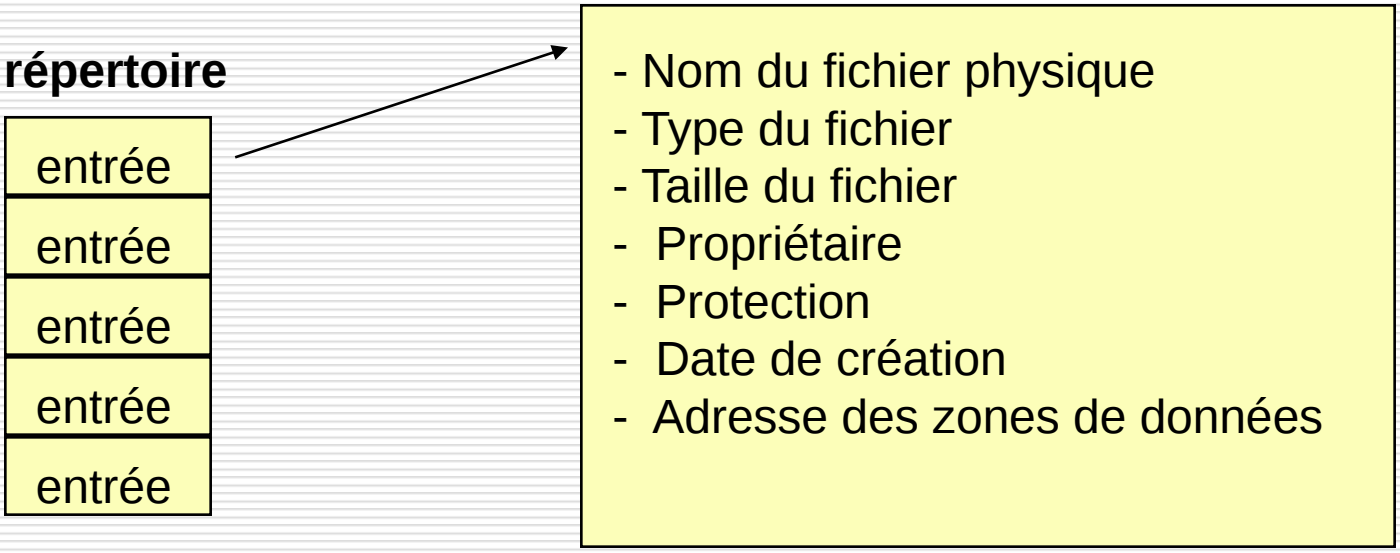
Le répertoire est une table sur le support permettant de référencer tous les fichiers existants du SGF avec leur nom et leurs caractéristiques principales.

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



Le répertoire

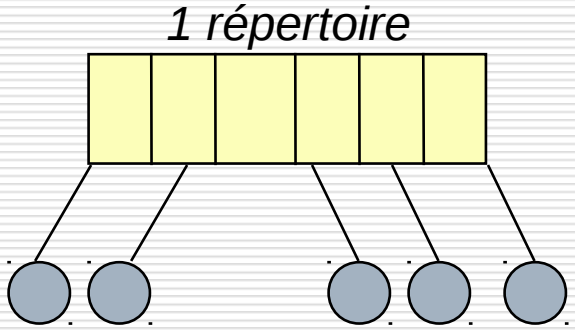
Un répertoire est une zone disque réservée par le SGF.
Le répertoire comprend un certain nombre d'entrées.
Une entrée est allouée à chaque fichier du SGF



1 entrée : attributs du fichier physique

Organisation des répertoires

Répertoire à un niveau : tous les fichiers du SGF sont répertoriées dans un unique catalogue.



↳ Tous les noms de fichiers doivent être différents.

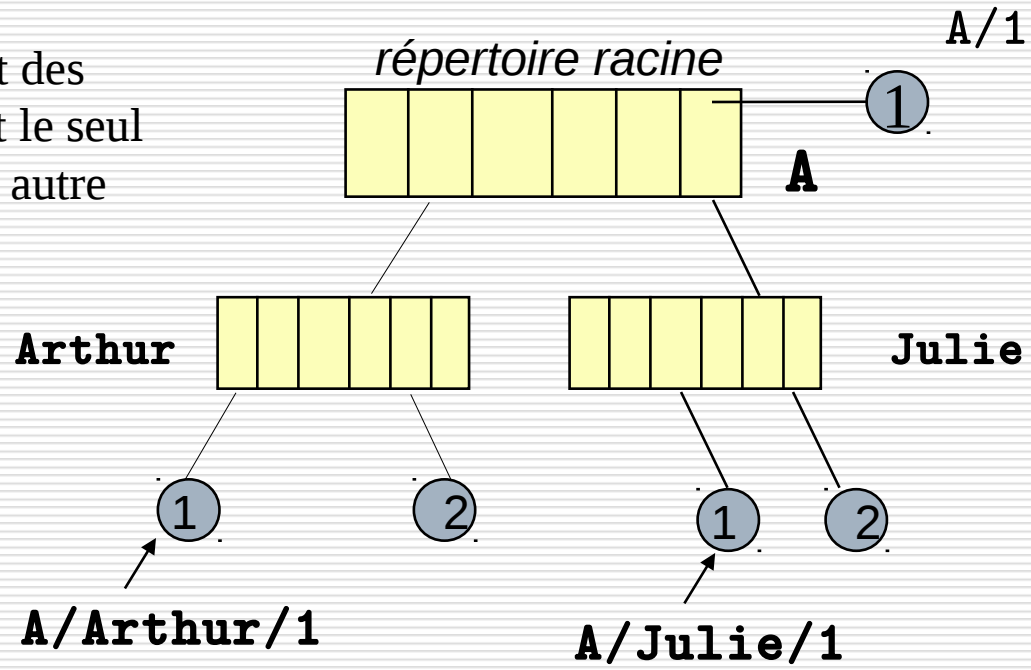
Difficile si plusieurs utilisateurs

Organisation des répertoires

Répertoire à deux niveaux : chaque utilisateur dispose d'un sous-répertoire (*User File Directory (UFD)*) dans lequel sont référencés tous les fichiers lui appartenant.

Le répertoire racine contient des répertoires et des fichiers. Il est le seul répertoire non inclus dans un autre

sous-répertoire
utilisateur **Arthur**
utilisateur **Julie**

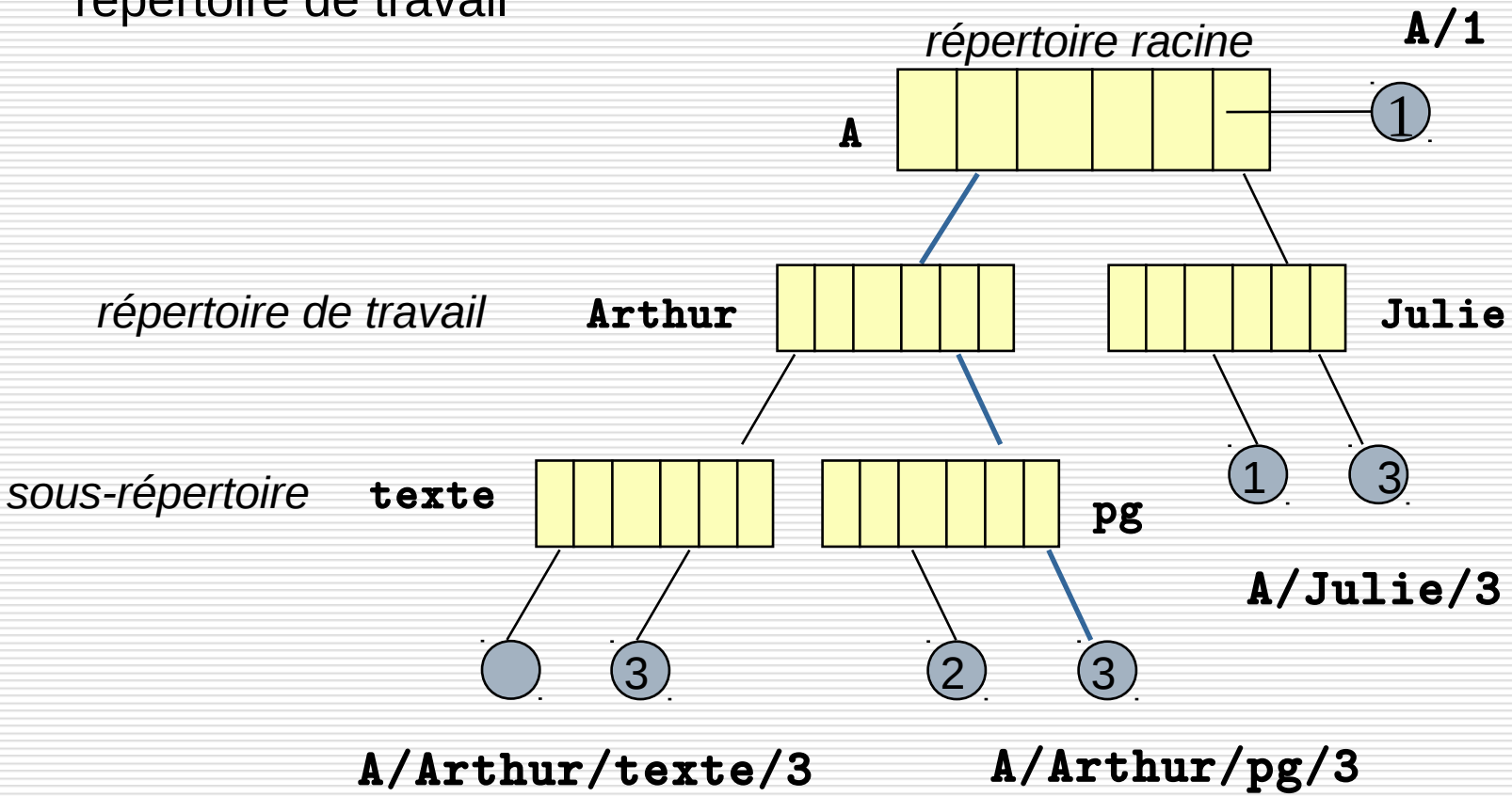


Nom de fichier `A/utilisateur /nomdufichier`

Organisation des répertoires

Répertoire à structure arborescente :

- chaque utilisateur dispose d'un sous-répertoire propre (r*épertoire de travail)*
- l'utilisateur peut créer des *sous-répertoires* à l'intérieur de son répertoire de travail

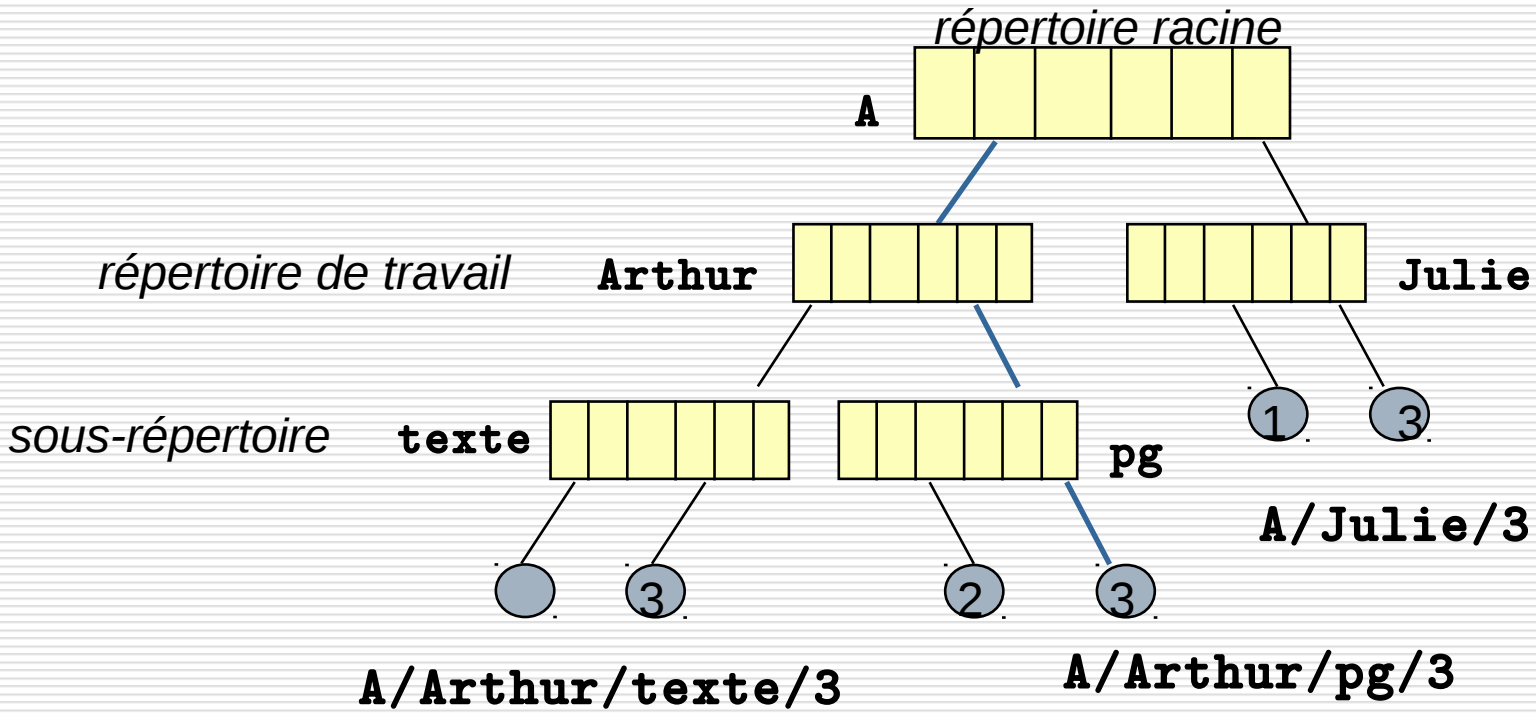


Organisation des répertoires

On appelle chemin (path) la succession des répertoires en partant de la racine pour atteindre un fichier. Un chemin est de la forme :

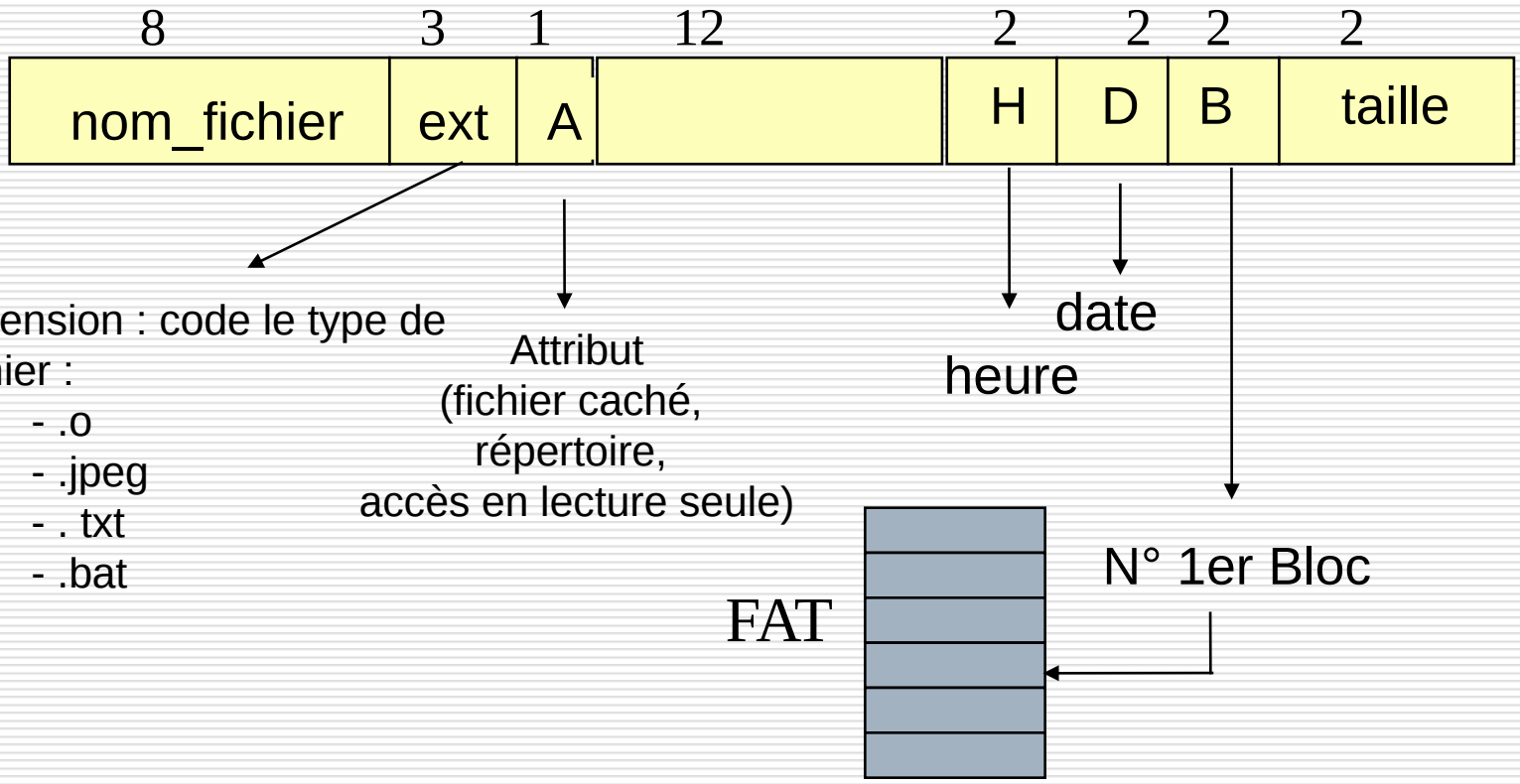
RepRacineséparateur**sousRep**séparateur**nomFichier**

sous unix le séparateur est / ; sous Windows le séparateur est \



Répertoire : exemple MS-DOS

Chaque fichier occupe 32 octets dans le répertoire



Un sous-répertoire occupe une entrée comme un fichier
le répertoire racine a 112 entrées.
les autres sous-répertoires n'ont pas d'entrées limitées

Opérations du Système de gestion de fichiers

- Structuration du disque
- Traitement des requêtes disque
- Opérations de haut niveau
- Protection des fichiers

Structuration du disque dur

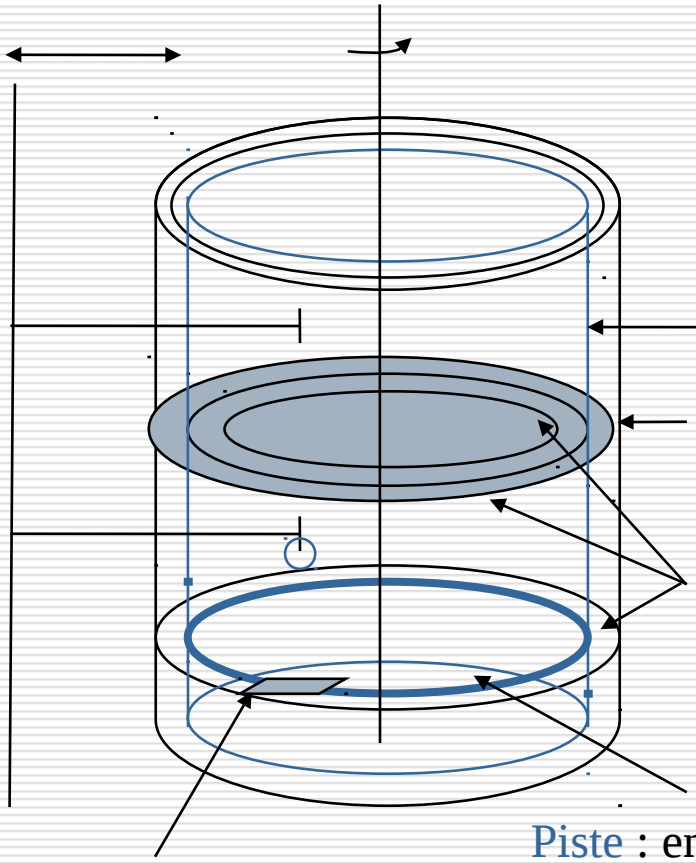
Structuration du disque dur

Trois opérations pour structurer le disque dur :

- Le formatage physique
- Le formatage logique
- Le partitionnement

Formatage physique

Le formatage physique ou de bas niveau permet de diviser la surface du disque en éléments basiques.



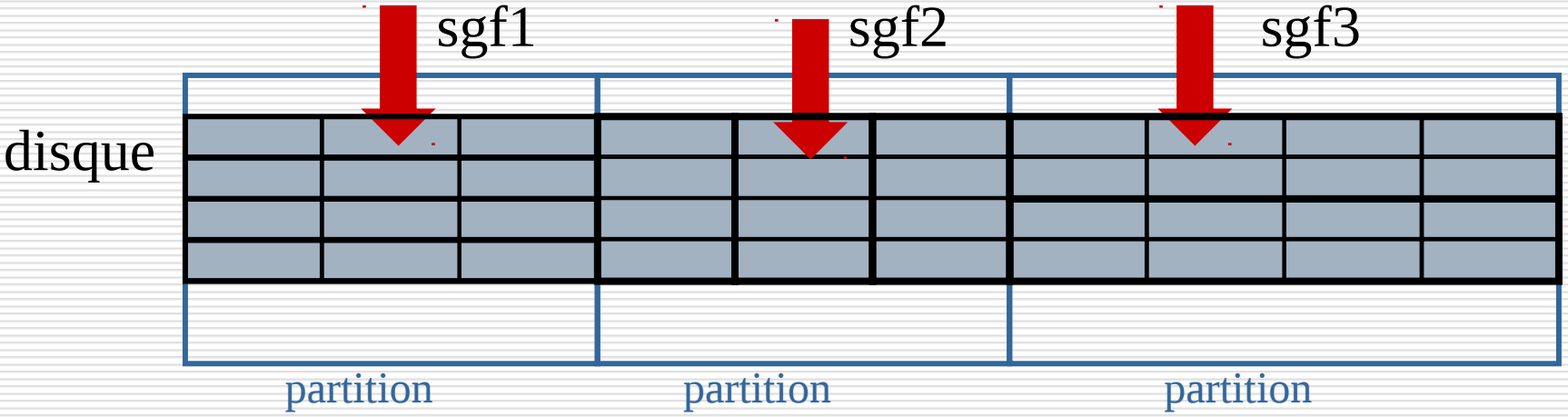
Il prépare le disque à accueillir des données

Adresse (HCS, head cylinder sector)

Formatage logique/ Partitionnement

Le formatage logique ou de haut niveau crée un système de gestion de fichiers sur le disque.

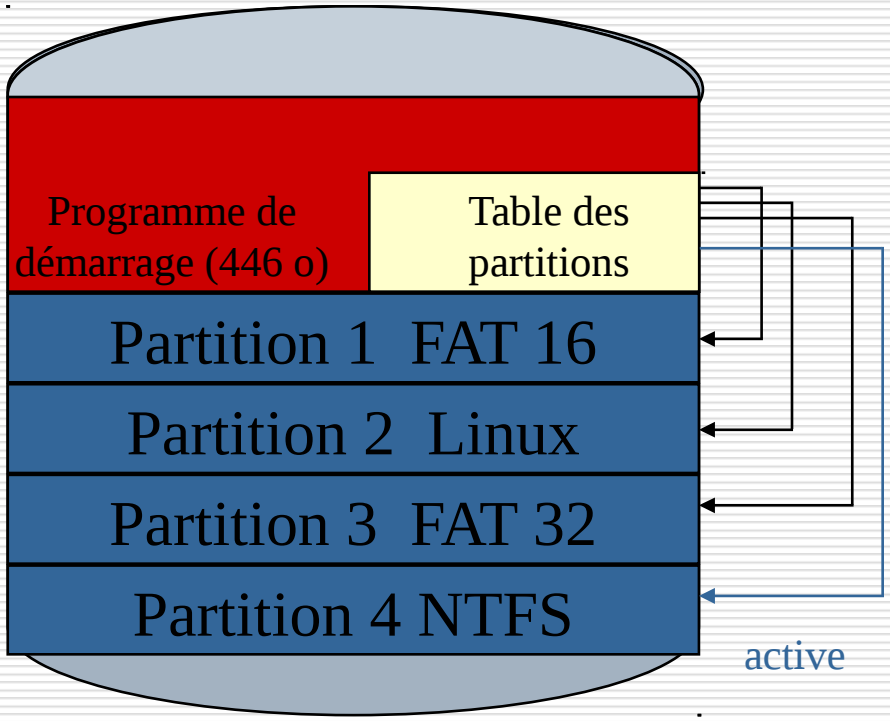
- Le type de SGF installé dépend du système d'exploitation. Il forme les clusters ou blocs
- Il est possible d'installer plusieurs types de SGF sur un disque grâce au partitionnement du disque..



Partitionnement

Une partition est une partie d'un disque dur destinée à accueillir un SGF. Elle est identifiée par un nom appelé **`nom de volume`**. Elle est constituée d'un ensemble de cylindres contigus

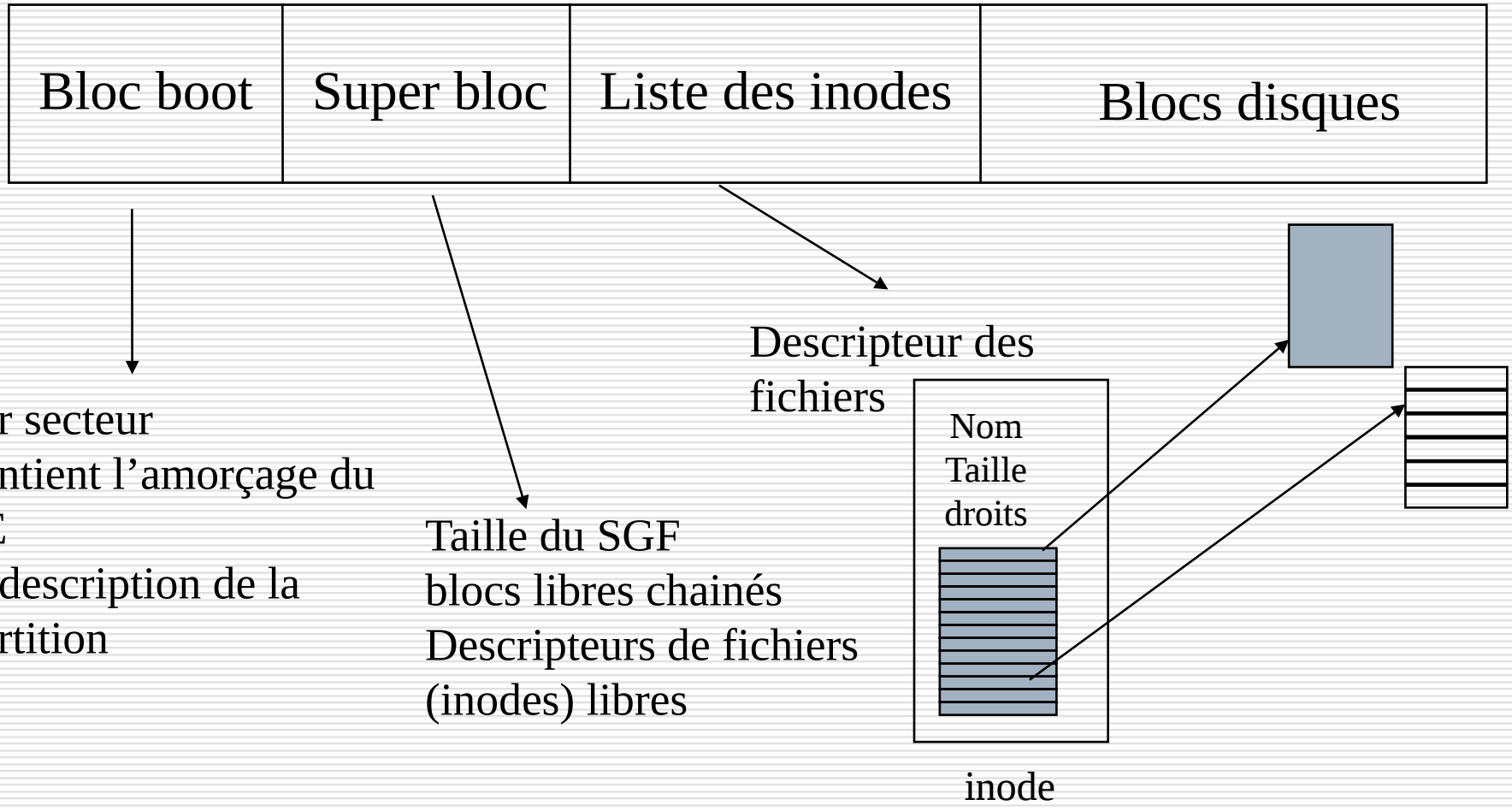
Un disque peut accueillir 4 partitions différentes. Une seule est active à la fois.



Premier secteur (cylindre 0, secteur 1, face 0)
Master Boot Record

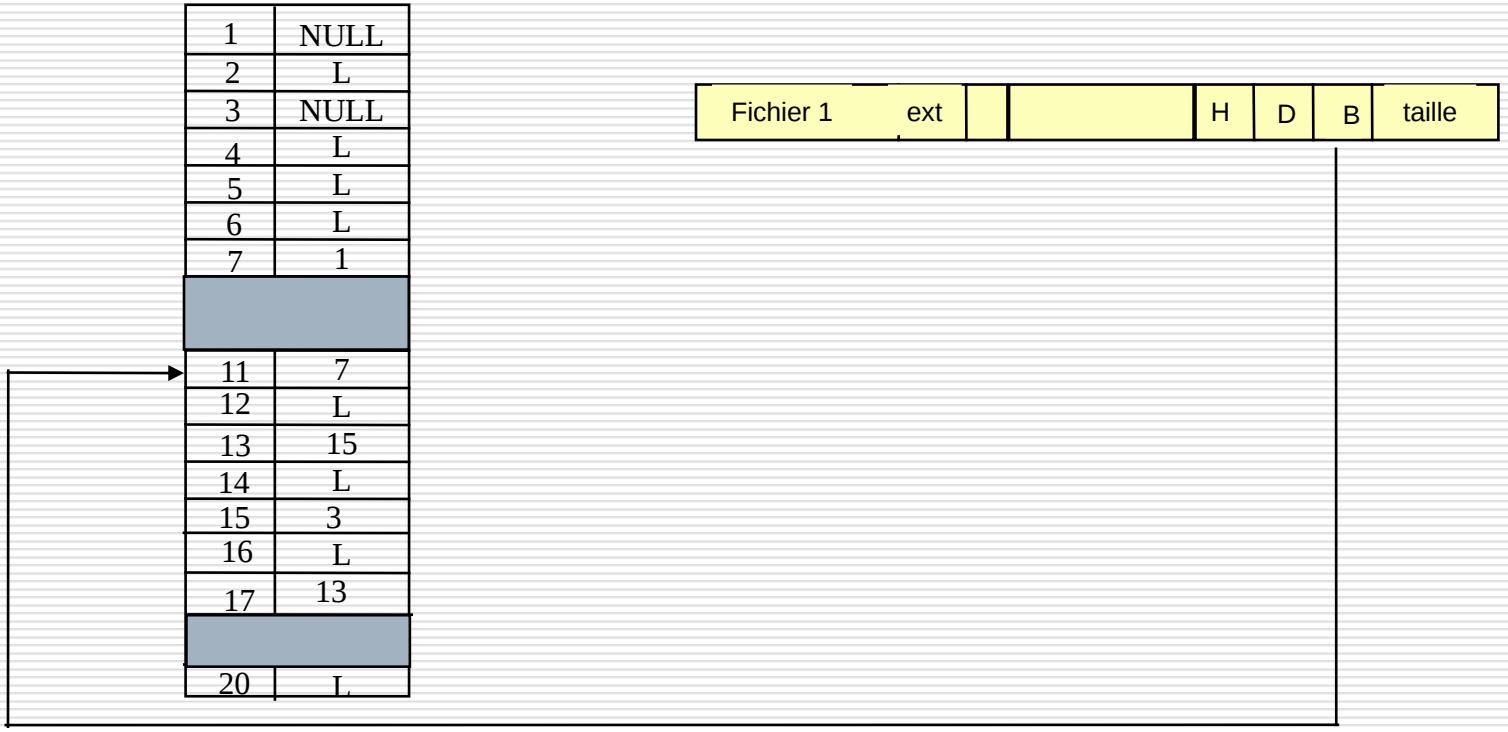
4 descripteurs de partition
4 * 16 octets
Adresse HCS premier secteur
Adresse HCS dernier secteur
Nombre de secteurs
type

Organisation de partition : LINUX



Organisation de partition : DOS

Secteur d'amorçage	FAT	Copie de la FAT	Répertoire racine	Blocs de données



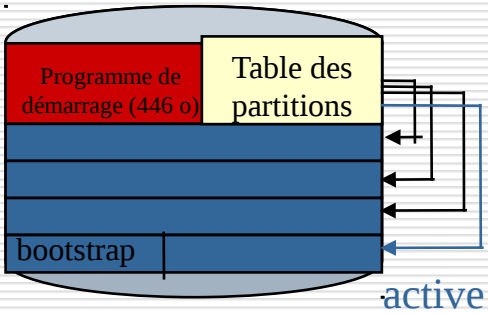
Démarrage de l'ordinateur

1. L'utilisateur appuie sur le bouton d'alimentation de l'unité centrale

2. Une fois le courant stabilisé, le processeur démarre et exécute le code du BIOS stocké dans la ROM à une adresse prédéfinie

3. Le BIOS exécute une séquence de vérification des composants (mémoire, périphériques de base) (POST : Power-OnSelf Test)

4. Le BIOS accède au CMOS pour lire la configuration matérielle de la machine (date, heure, périphérique de masse contenant le système d'exploitation).



5. Le BIOS accède au MBR du disque ; il charge en mémoire centrale le programme de démarrage

6. Le programme de démarrage détermine la partition active et transfert le contrôle au bootstrap de la partition pour charger le système d'exploitation

Traitement des requêtes disque

Traitement des requêtes disque

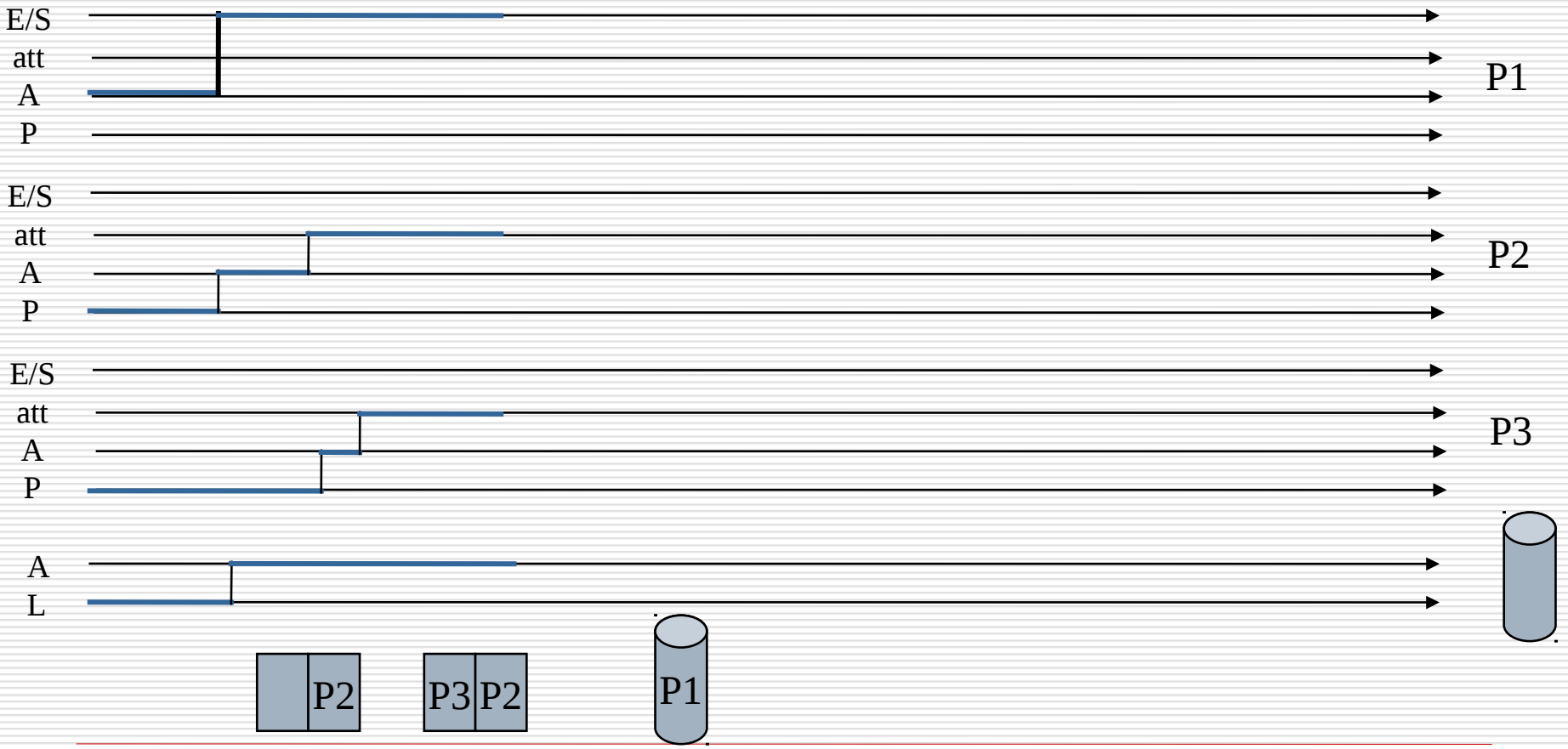
Le temps de traitement d'une entrée-sortie disque est lente par rapport à l'exécution d'un programme :

→plusieurs requêtes disque peuvent être adressées au disque alors qu'il est encore en train de traiter la première

Pour optimiser le traitement des requêtes disque accumulées dans la file d'attente du contrôleur, on utilise des algorithmes d'ordonnancement du bras

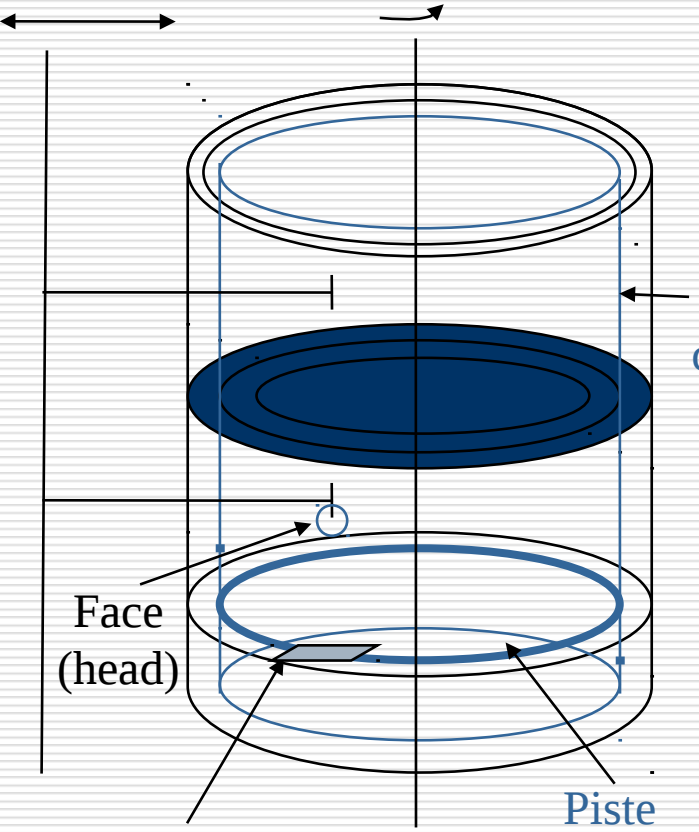
Traitement des requêtes disque

Le temps de traitement d'une entrée-sortie disque est lente par rapport à l'exécution d'un programme →plusieurs requêtes disque peuvent être adressées au disque alors qu'il est encore en train de traiter la première



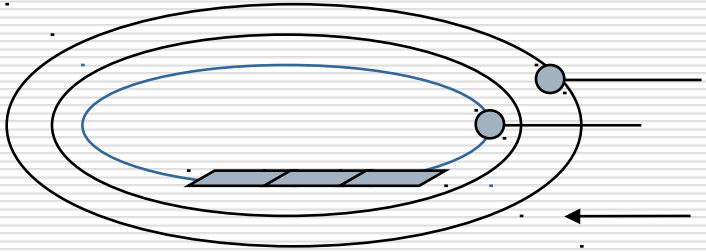
Temps d'accès à un secteur

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

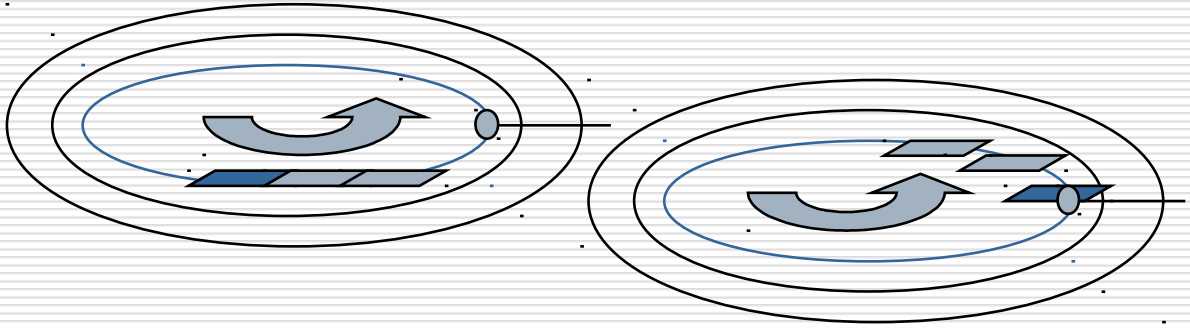


Secteur : 512 octets

Adresse HCS (3, 1, 30)



1. positionnement du bras sur la piste

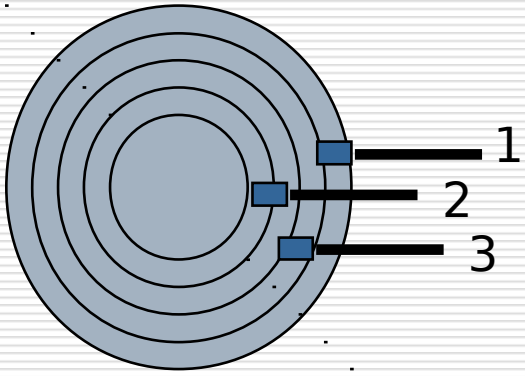


2. temps d'attente pour le passage du secteur sous la tête de lecture

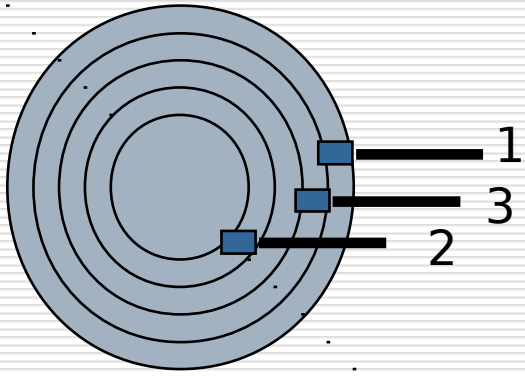
- le temps de positionnement du bras est le plus pénalisant : on utilise des algorithmes de services des requêtes disques pour réduire au mieux les mouvements du bras

L'ordonnancement du bras

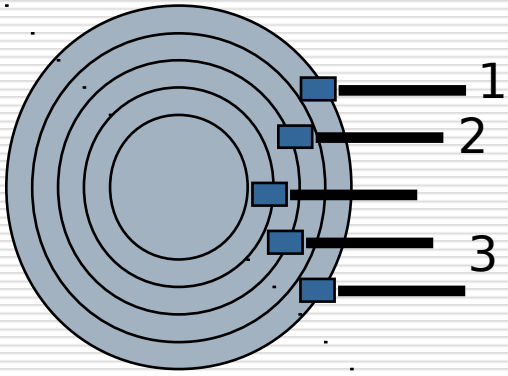
Les algorithmes d'ordonnancement du bras qui améliorent le temps d'accès :
- FCFS (First Come, First Serverd): requêtes servies séquentiellement
- SSTF (Shortest Seek Time First) : requête la plus proche d'abord
- SCAN (ou C-SCAN) : Algorithme de l'ascenseur



FCFS



SSTF



SCAN

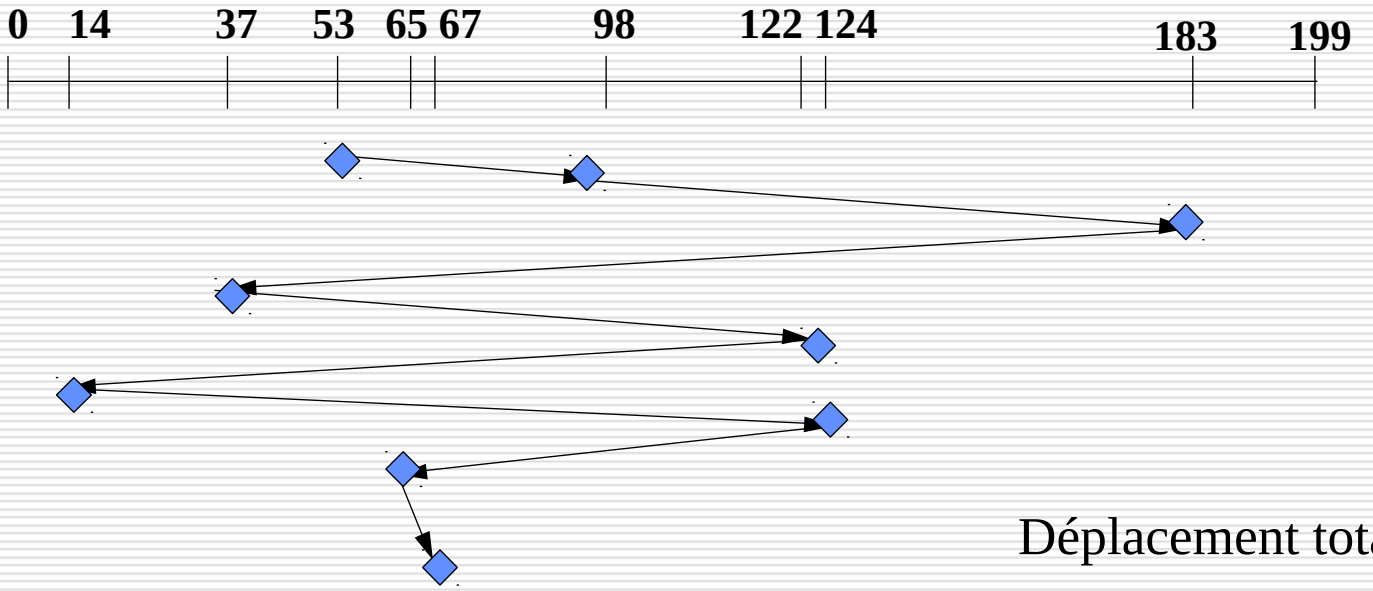
Ordonnancement du disque : FCFS

Les requêtes disque sont servies selon leur ordre d'arrivée.

Exemple : liste de requêtes (n° de piste)

98 183 37 122 14 124 65 67

position initiale de la tête de lecture/écriture : piste 53



Déplacement total : 640 pistes

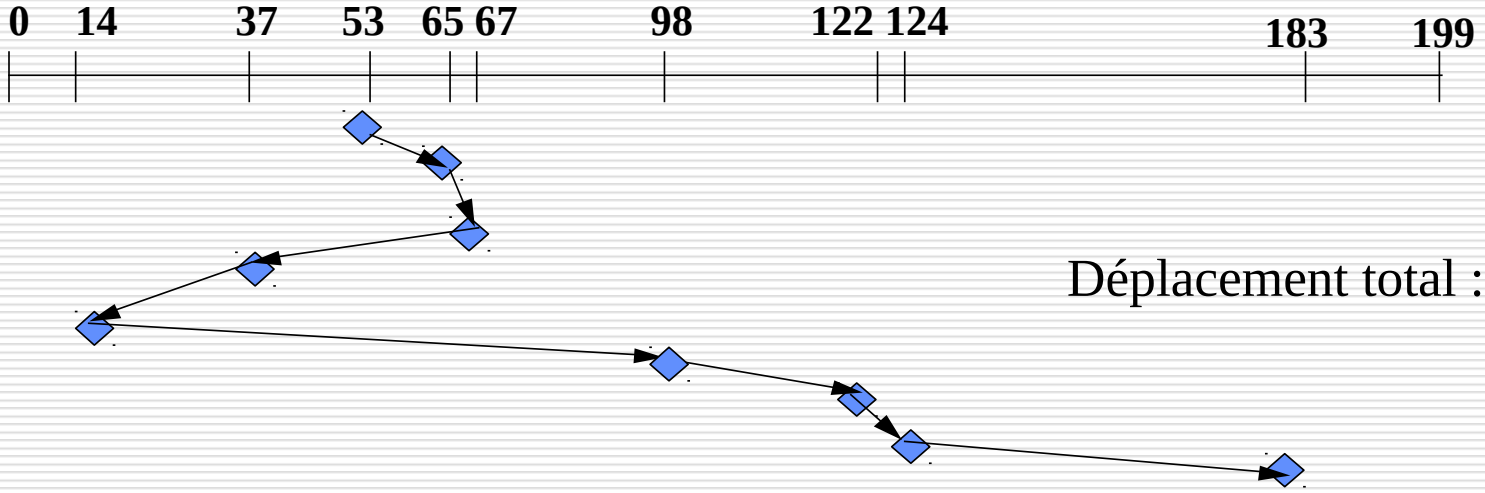
Ordonnancement du disque : SSTF

La requête servie est celle dont la position est la plus proche de la position courante

Exemple : liste de requêtes (n° de piste)

98 183 37 122 14 124 65 67

position initiale de la tête de lecture/écriture : piste 53



Déplacement total : 236 pistes

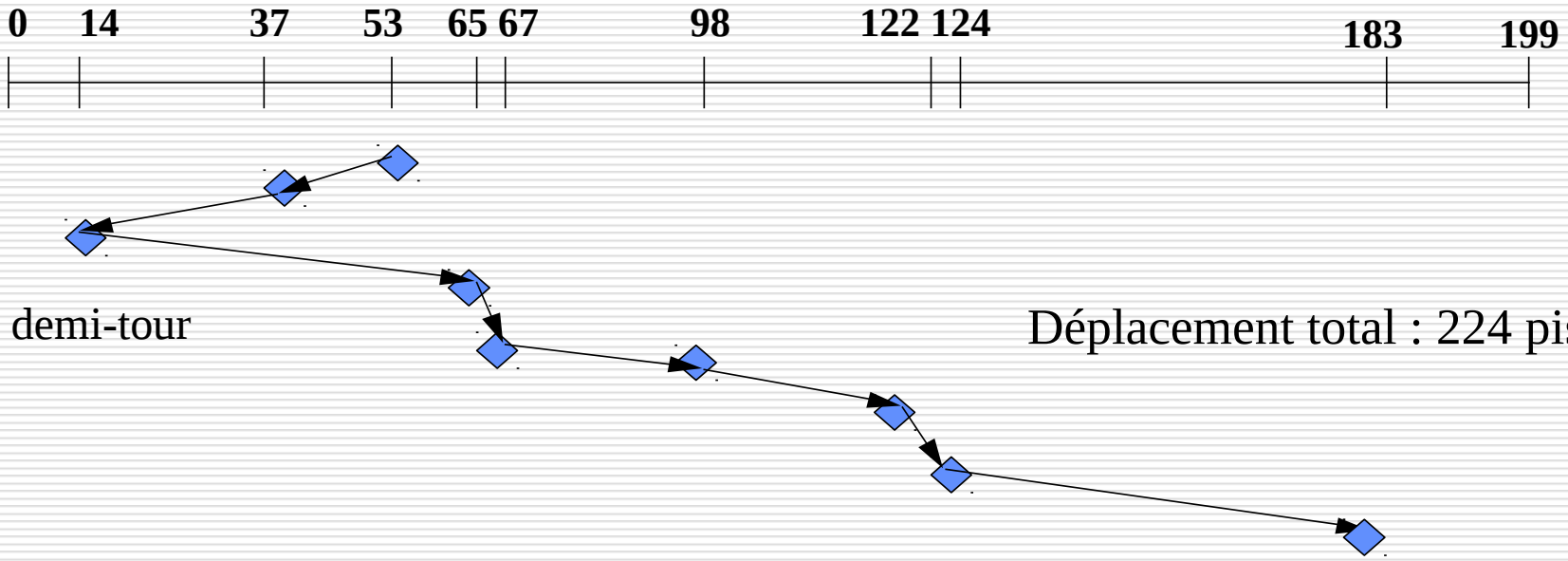
Ordonnancement du disque : SCAN

Le bras balaye l'ensemble des pistes et sert les requêtes au fur et à mesure des pistes parcourues

Exemple : liste de requêtes (n° de piste)

98 183 37 122 14 124 65 67

position initiale de la tête de lecture/écriture : piste 53



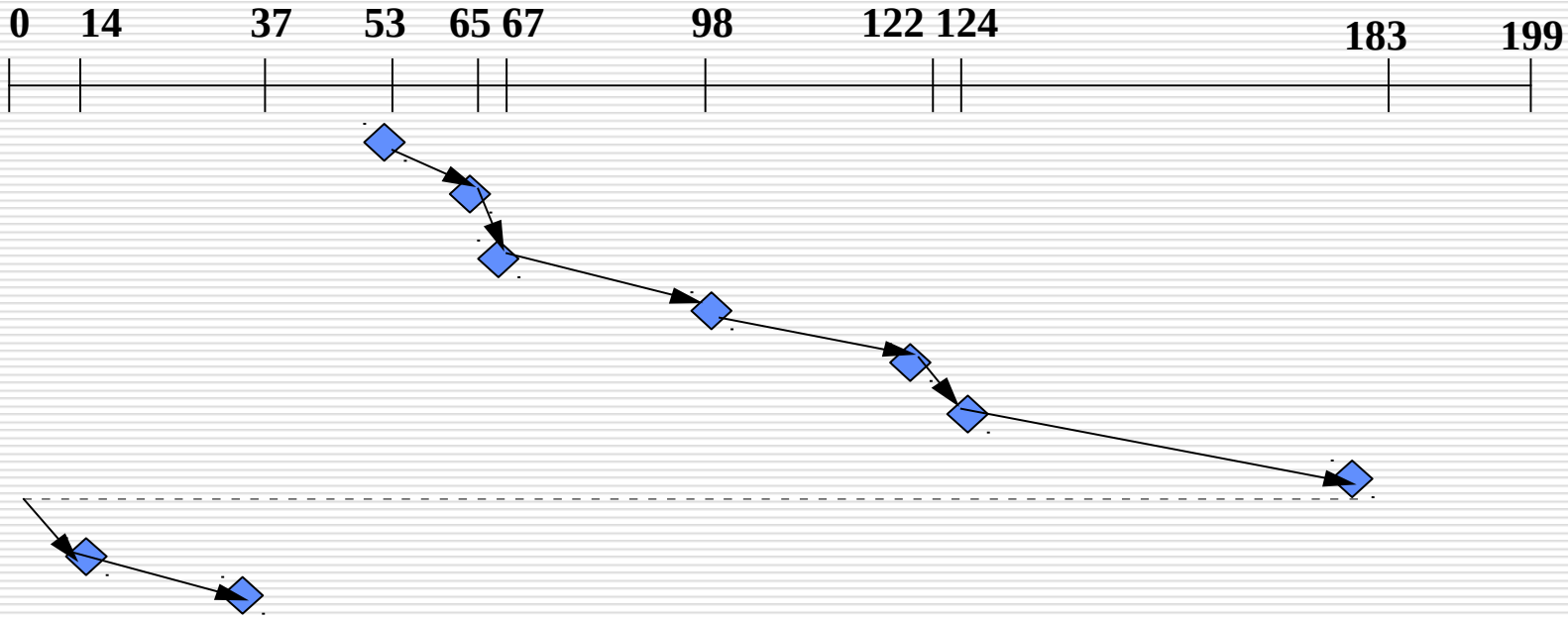
Ordonnancement du disque : C-SCAN

Le bras balaye l'ensemble des pistes toujours dans le même sens

Exemple : liste de requêtes (n° de piste)

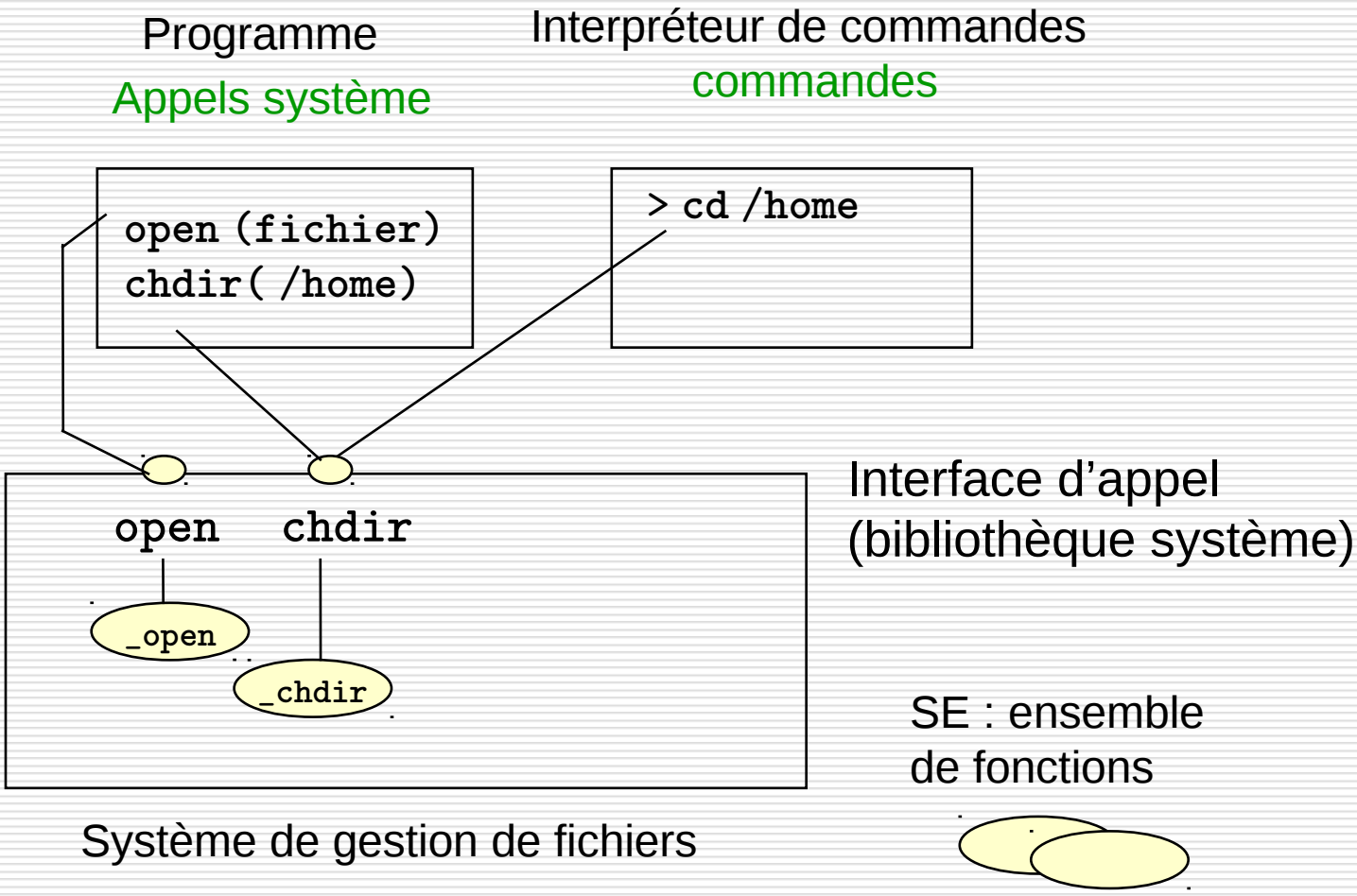
98 183 37 122 14 124 65 67

position initiale de la tête de lecture/écriture : piste 53



Réalisation des opérations de haut niveau

Interfaces d'appel du SGF



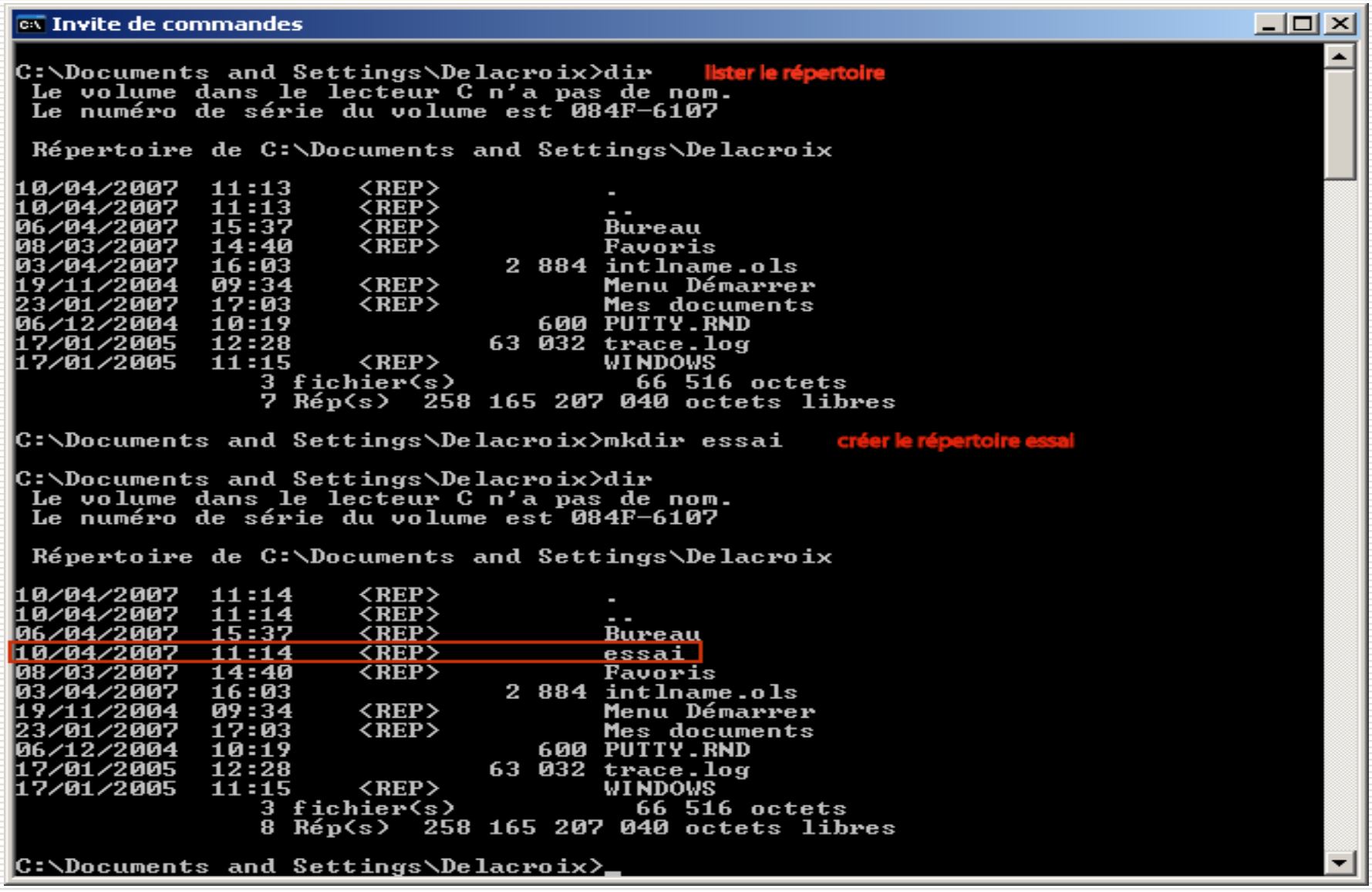
Les commandes

Quelques commandes du SGF:

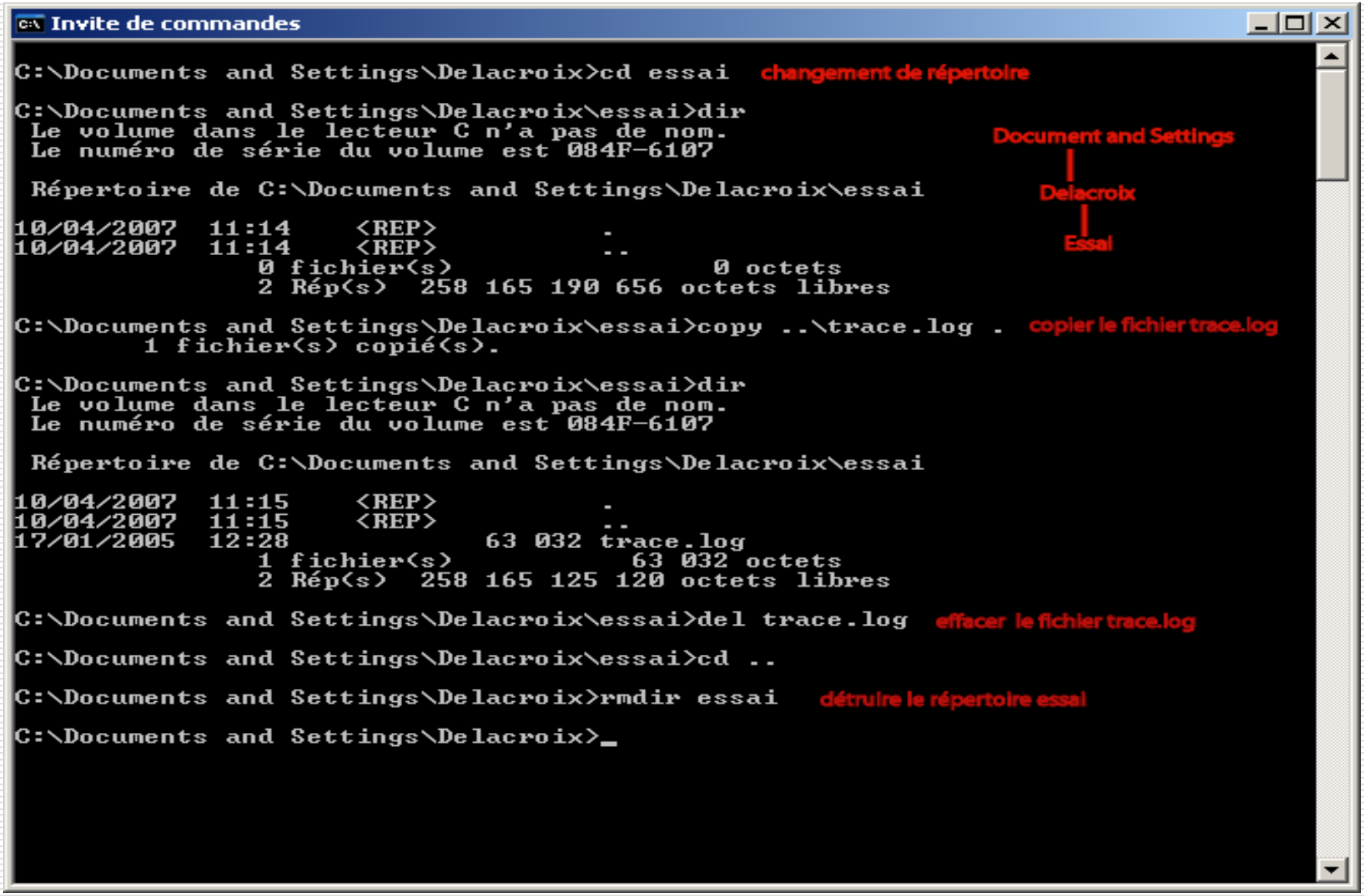
- liste du répertoire (`ls, dir`)
- changement de répertoire (`cd`)
- création répertoire (`mkdir`)
- suppression répertoire (`rmdir`)
- suppression fichier (`rm, del`)
- modification d'attributs d'un fichier (`chmod`)
- changement de nom de fichier (`mv, ren`)

Au lancement d'une commande -> appel à la fonction du SGF

Les commandes



Les commandes



Les appels systèmes

Quelques appels systèmes du SGF:

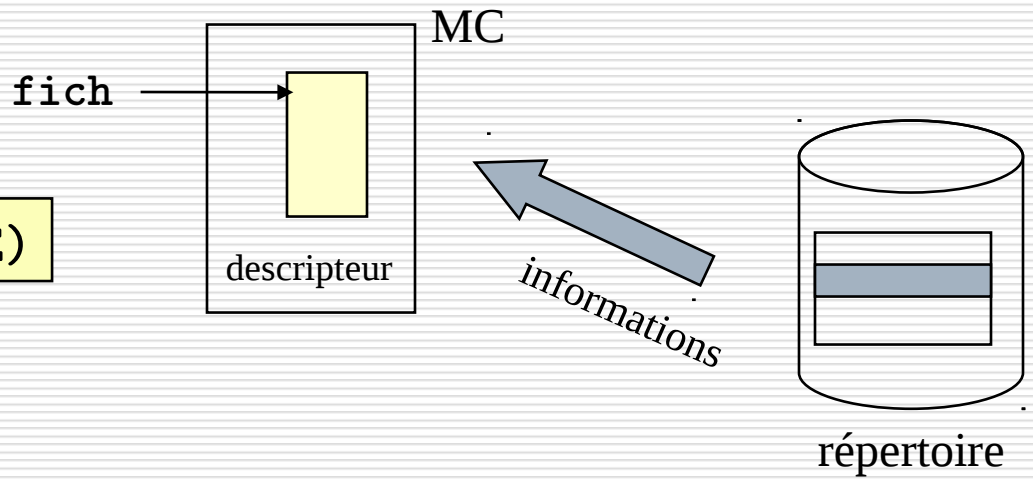
- ouverture fichier (`open`)
- création de fichier (`create`)
- fermeture fichier (`close`)
- lecture d'enregistrement (`read`)
- écriture d'enregistrement (`write`)

Exemple :

```
struct element {
    char nom[12];
    int note : integer; }
element enrg;
```

```
main () {
  int fich;
  fich = open (ficG, r);
  while (not END_OF_FILE) {
    read(fich, enrg, taille_enrg_octets) }
  close(fich)
}
```

Ouverture d'un fichier




```
fich = open (nom_fichier, L/E)
```

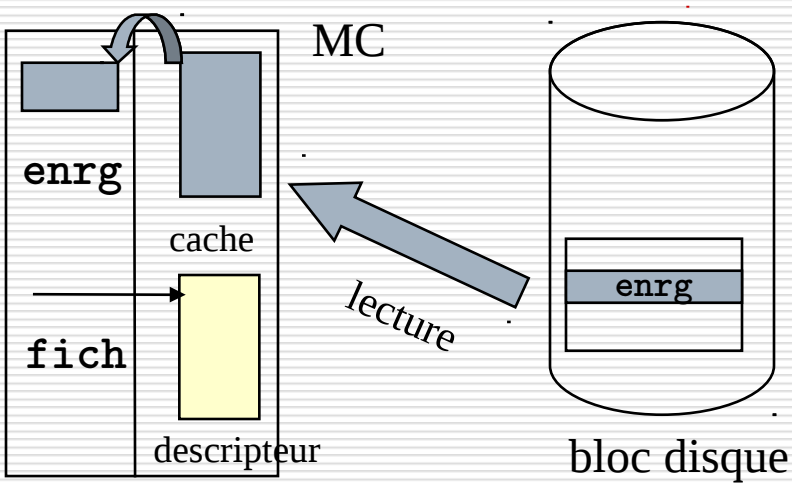
L'opération d'ouverture :

- vérifie l'existence du fichier `nom_fichier` dans le répertoire du disque
- vérifie la compatibilité des types d'accès au fichier (`L/E`)
- transfert les éléments de l'entrée de répertoire dans le descripteur correspondant en mémoire
- renvoie au programme un index `fich` qui désigne le descripteur de fichier

Le descripteur est conservé et mis à jour en mémoire jusqu'à la fermeture

Lecture d'un fichier

```
read(fich, enrg, numEnrg)
```



La lecture d'un fichier nécessite:
- le pointeur du descripteur **`fich`**
- l'adresse mémoire de la zone réceptrice de l'enregistrement **`enrg`**
- le numéro de l'enregistrement **`numEnrg`** qui peut:
  - ne pas exister si la lecture est séquentielle
  - être le rang si l'accès est direct

Il faut déterminer l'adresse du bloc physique contenant l'enregistrement à lire.
Les données lues sont transférées dans un tampon mémoire.

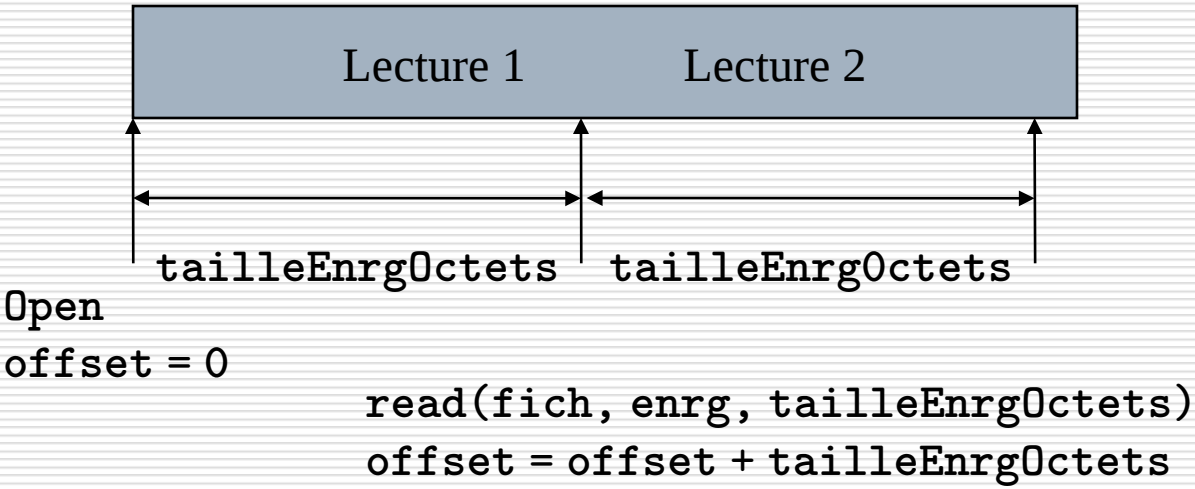
Exemple : lecture d'un fichier sous Unix

le cnam

```
read(fich, enrg, tailleEnrgOctets)
```

Un fichier Unix est une suite d'octets sans structure, accessibles de manière séquentielle. Le SGF maintient pour chaque fichier un `offset` qui pointe sur l'octet courant dans le fichier.

Une opération de lecture spécifie combien d'octets doivent être lus (`tailleEnrgOctets`).
La lecture s'effectue depuis la position courante de l'`offset` et délivre les `tailleEnrgOctets` octets suivants.

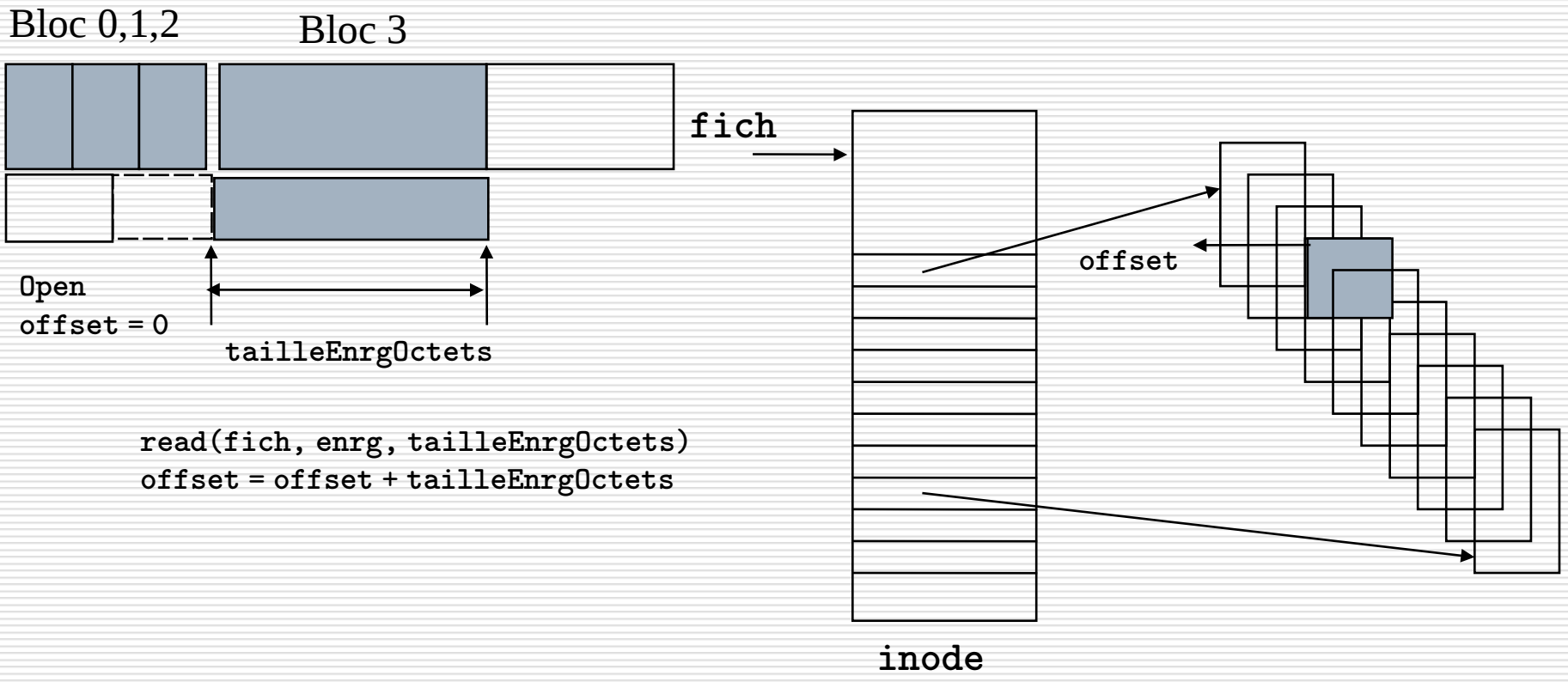


Exemple : lecture d'un fichier sous Unix

le cnam

```
read(fich, enrg, tailleEnrgOctets)
```

Il faut déterminer à quel bloc les octets à lire appartiennent, et lire ce bloc



Exemple : lecture d'un fichier sous Unix

le cnam

```
read(fich, enrg, tailleEnrgOctets)
```

- Le système maintient une liste de tampons mémoire qui joue le rôle de cache pour les blocs du disque et permet de réduire les entrées/sorties. La taille d'un tampon est égale à la taille d'un bloc disque.

- Lorsque le système doit lire un bloc depuis le disque :
  - Il cherche d'abord si le bloc est déjà présent dans la liste des tampons mémoire
  - Si non, il prend un tampon libre et copie le bloc disque dans le tampon
  - Si tous les tampons sont occupés, il libère un tampon en choisissant le moins récemment accédé.

Exemple : lecture d'un fichier sous Unix

le cnam

1ère lecture : accès au bloc 0 de ficG

Bloc de 1 Ko, **enrg** de 32 octets → 32 enregistrements /bloc

Utilisateur

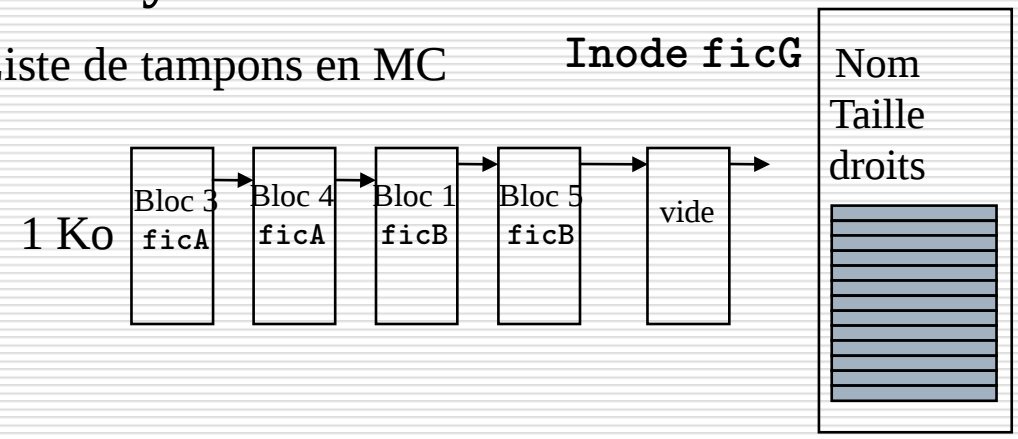
```
struct element {
      char nom [];
      int note : integer; }
element enrg;
```

32

enrg

```
fich = open(ficG, r);
while (not END_OF_FILE) {
  read(fich, enrg, tailleEnrrOctets) }
close(fich)
```

Système



Recherche du bloc 0 de **ficG**, il n'y est pas :

→ lecture depuis le disque

Exemple : lecture d'un fichier sous Unix

le cnam

1ère lecture : accès au bloc 0 de ficG

Bloc de 1 Ko, **enrg** de 32 octets → 32 enregistrements /bloc

Utilisateur

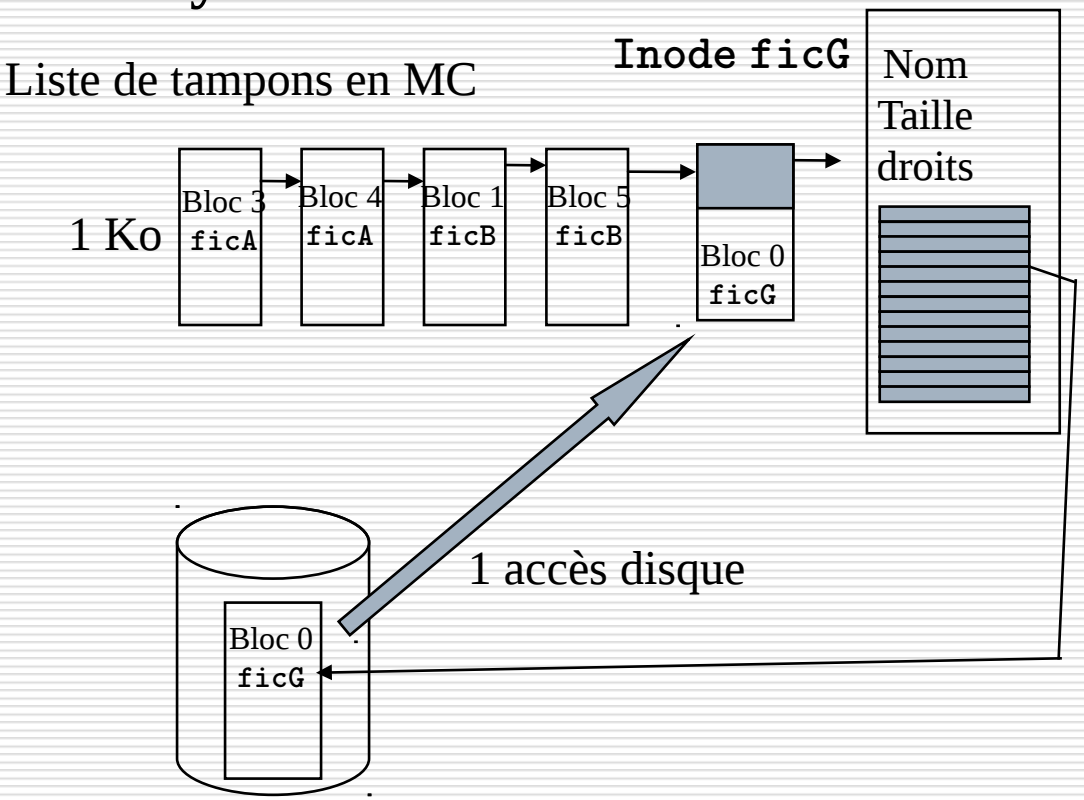
```
struct element {
      char nom [];
      int note : integer; }
element enrg;
```

32

enrg

```
fich = open(ficG, r);
while (not END_OF_FILE) {
  read(fich, enrg, tailleEnrrOctets) }
close(fich)
```

Système



Exemple : lecture d'un fichier sous Unix

le cnam

2ème lecture : accès au bloc 0 de ficG

Bloc de 1 Ko, **enrg** de 32 octets → 32 enregistrements /bloc

Utilisateur

```
struct element {
      char nom [];
      int note : integer; }
element enrg;
```

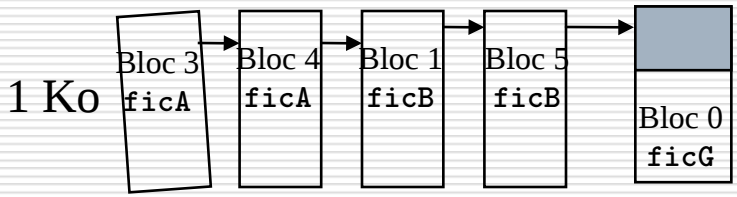
32

enrg

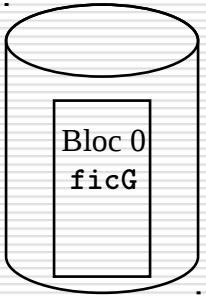
```
fich = open(ficG, r);
while (not END_OF_FILE) {
  read(fich, enrg, tailleEnrrOctets) }
close(fich)
```

Système

Liste de tampons en MC



Recherche du bloc 0 de **ficG** dans la liste
bloc 0 de **ficG** est dans la liste



0 accès disque

Exemple : lecture d'un fichier sous Unix

le cnam

33ème lecture : accès au bloc 1 de ficG

Bloc de 1 Ko, **enrg** de 32 octets → 32 enregistrements /bloc

Utilisateur

```
struct element {
      char nom [];
      int note : integer; }
element enrg;
```

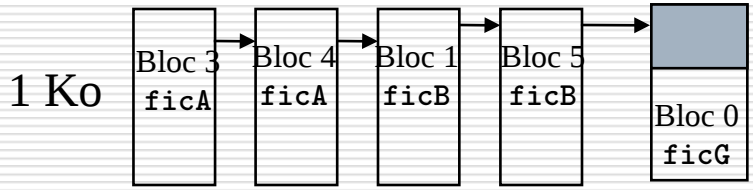
32

enrg

```
fich = open(ficG, r);
while (not END_OF_FILE) {
  read(fich, enrg, tailleEnrrOctets) }
close(fich)
```

Système

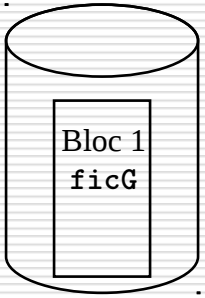
Liste de tampons en MC



Recherche du bloc 1 de **ficG** dans la liste

bloc 1 de ficG n'est pas dans la liste

→ lecture depuis le disque



Exemple : lecture d'un fichier sous Unix

33ème lecture : accès au bloc 1 de ficG

Bloc de 1 Ko, **enrg** de 32 octets → 32 enregistrements /bloc

Utilisateur

```
struct element {
      char nom [];
      int note : integer; }
element enrg;
```

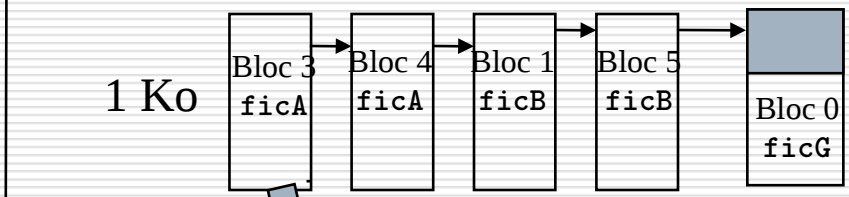
32

enrg

```
fich = open(ficG, r);
while (not END_OF_FILE) {
  read(fich, enrg, tailleEnrrOctets) }
close(fich)
```

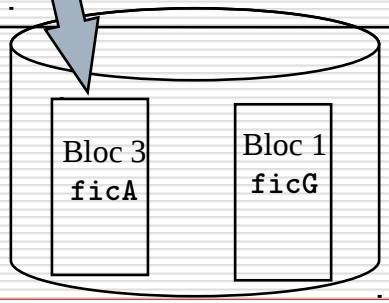
Système

Liste de tampons en MC



Écriture si le bloc a été modifié dans le cache

Aucun tampon libre
Le moins récemment accédé est libéré (Bloc3, **FicA**)



1 accès disque

Exemple : lecture d'un fichier sous Unix

33ème lecture : accès au bloc 1 de ficG

Bloc de 1 Ko, **enrg** de 32 octets → 32 enregistrements /bloc

Utilisateur

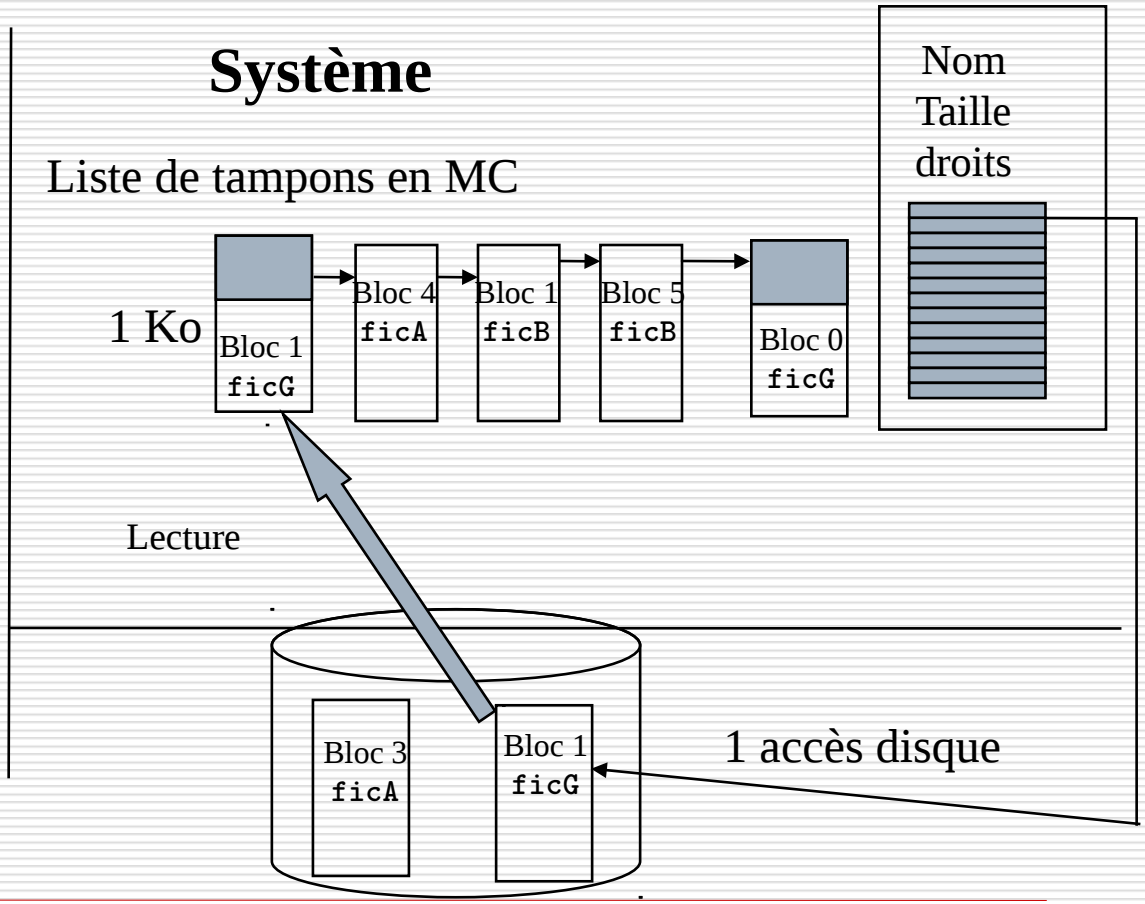
```
struct element {
      char nom [] ;
      int note : integer; }
element enrg;
```

32

enrg

```
fich = open(ficG, r);
while (not END_OF_FILE) {
  read(fich, enrg, tailleEnrrOctets) }
close(fich)
```

Système



Écriture dans un fichier

le cnam

```
write(fich, tampon, numEnrg)
```

L'écriture dans un fichier nécessite:

- un pointeur du descripteur **`fich`**
- l'adresse mémoire de la zone émettrice de l'enregistrement **`tampon`**
- le numéro de l'enregistrement **`numEnrg`** qui peut:
  - ne pas exister si l'écriture est séquentielle
  - être le rang si l'accès est direct.

Cette fonction met à jour certains éléments du descripteur **`fich`** ; elle peut entraîner l'allocation d'un nouveau bloc au fichier.

Les données sont transférées du tampon mémoire **`tampon`** vers le fichier.

Fermeture d'un fichier

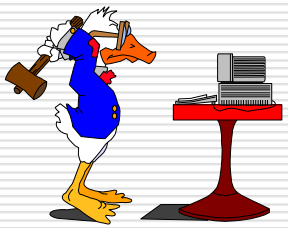
```
close(fich)
```

La fermeture transfert les éléments du descripteur `fich` vers le répertoire sur le disque.

Protection des fichiers

Protection des fichiers

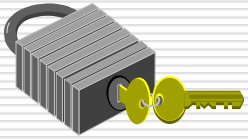
Protection contre les dégâts physiques



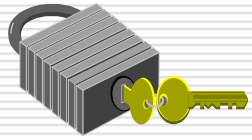
- Fiabilité du support
- Utilisation de différents types de redondance

Protection contre les accès inappropriés

- droits d'accès



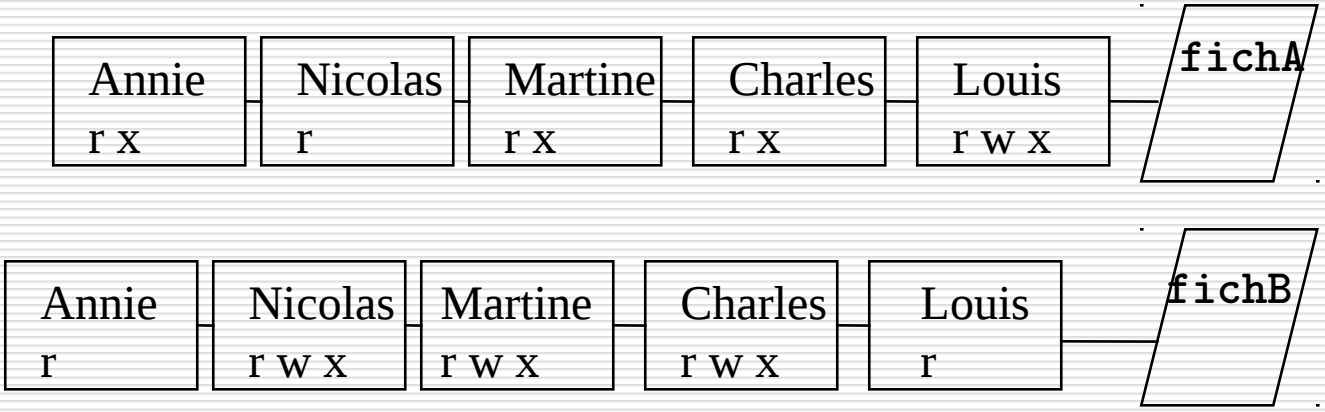
Protection contre les accès inappropriés



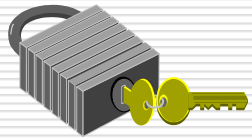
Définition de droits d'accès

- lecture (r), écriture (w), exécution (x), destruction ...

A chaque fichier est associé une liste d'accès, spécifiant pour chaque utilisateur, les types d'accès qui lui sont autorisés

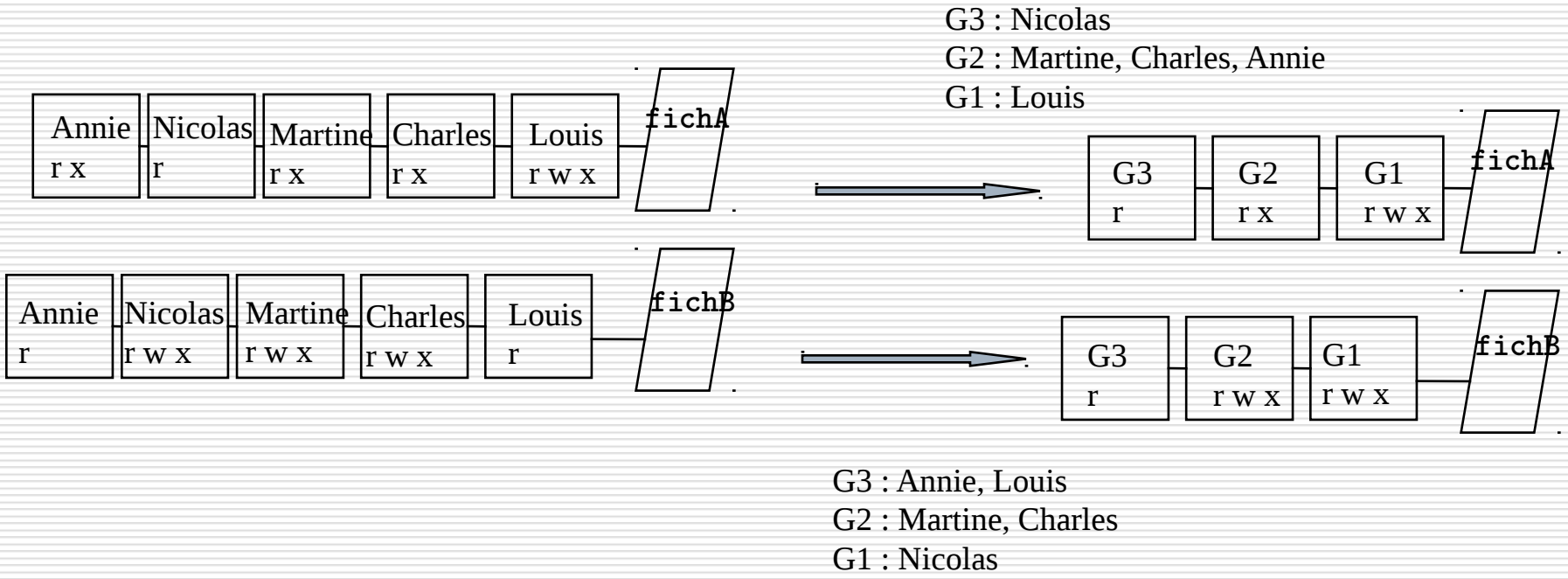


Protection contre les accès inappropriés



La liste d'accès peut être longue et difficile à gérer

- définition de groupes auxquels sont associés des droits
- un utilisateur hérite des droits du groupe auxquels il appartient

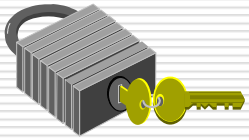


Protection : exemple Unix

le cnam

Définition de trois groupes :

- le propriétaire : celui qui a crée le fichier
- le groupe : le groupe de travail
- les autres : tous les autres



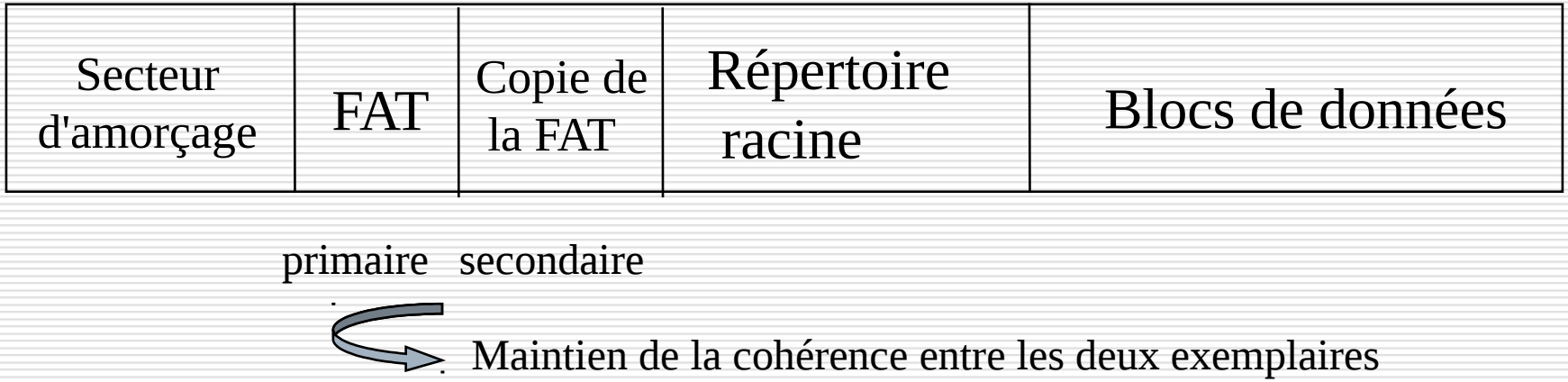
```
> ls -l
drwxr-xr-x  2 delacroi 4096 Oct 22  1998 repertoire
-rw-r--r--  1 delacroi 6401 Jan  8  1997 eleve.c
-rwxr-xr-x  1 delacroi 24576 Dec 15  1998 essai

-rw-r--r--  1 delacroi 67 Dec 15  1998 essai.c

> chmod a+w essai.c

> ls -l essai.c
-rw-rw-rw-  1 delacroi      67 Dec 15  1998 essai.c
```

Protection contre les dégâts physiques

Utilisation de la redondance interne :

- l'information existe en double exemplaire : une version primaire, une version secondaire
- le système maintient la cohérence entre les deux versions

➥ exemple : Windows dispose de deux exemplaires de la FAT

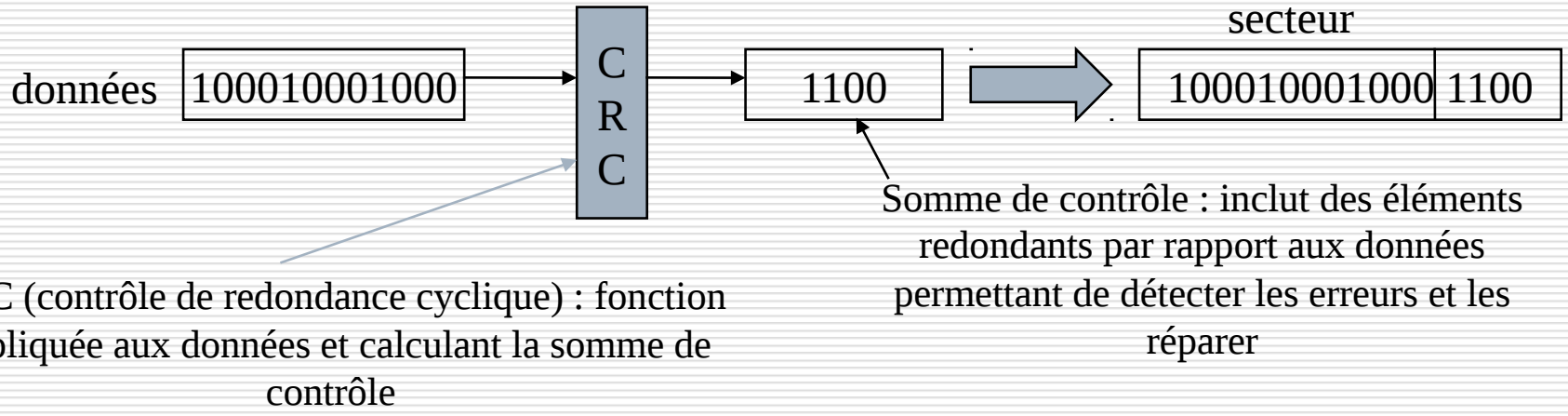


Protection contre les dégâts physiques

Utilisation de la redondance interne :

Des données supplémentaires appelées somme de contrôle sont ajoutées à l'information qui permettent de vérifier la validité des informations.

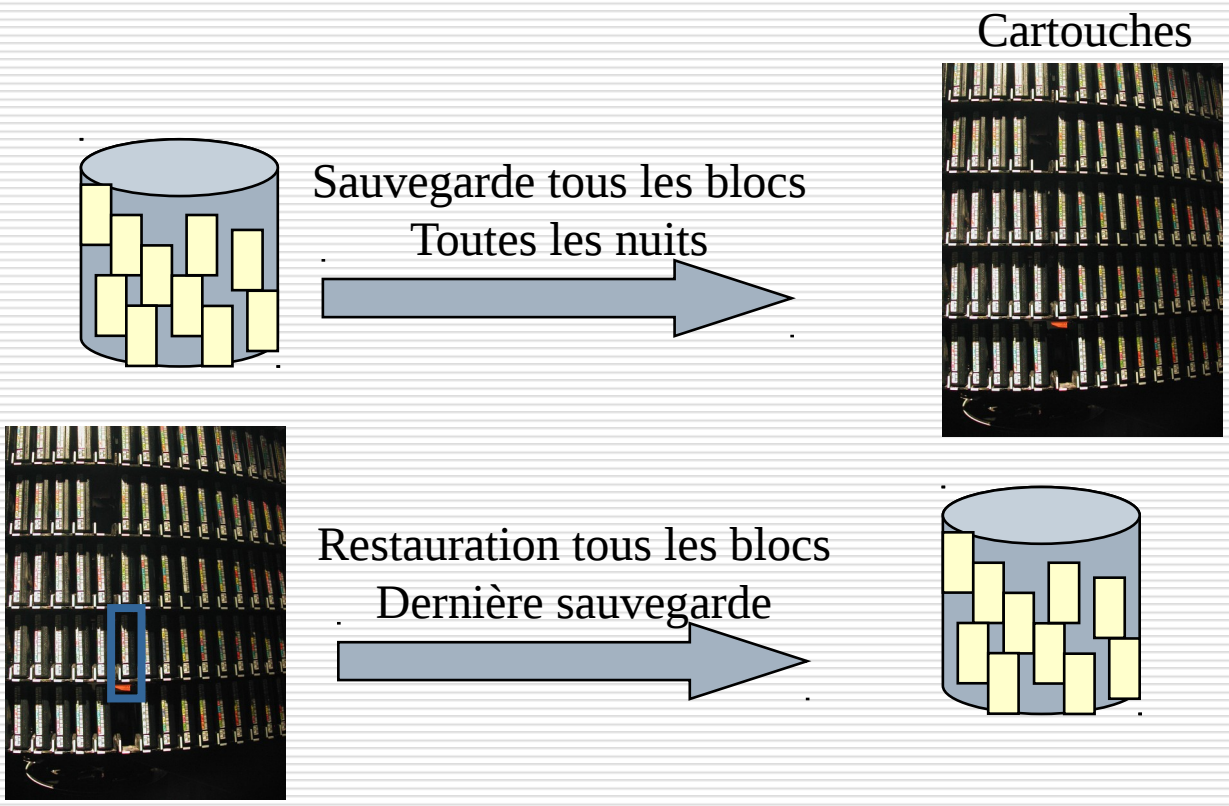
- exemple : chaque secteur du disque comporte les données + une somme de contrôle. Cette somme de contrôle permet de détecter les secteurs défectueux.



Protection contre les dégâts physiques

Redondance par sauvegarde périodique :

→ sauvegarde complète : la totalité du SGF est dupliqué même si les fichiers n'ont pas été modifiés

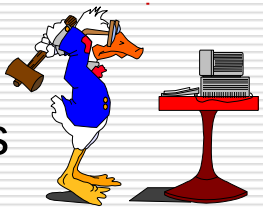


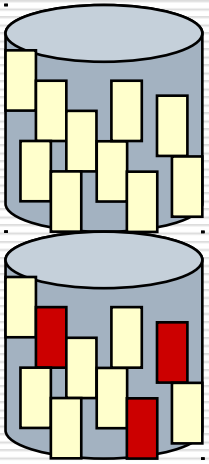
Protection contre les dégâts physiques

le cnam

Redondance par sauvegarde périodique :

→ sauvegarde incrémentale : seuls les objets modifiés depuis la dernière sauvegarde sont dupliqués.





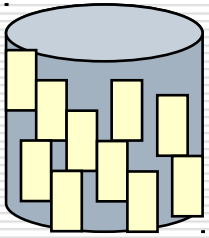
Sauvegarde tous les blocs
Dimanche nuit

Sauvegarde les blocs modifiés
L, ma, me, je, ve, sa nuit

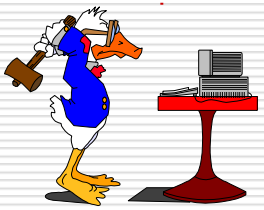
Cartouches

Restauration
dernière sauvegarde complète
Complétée par les sauvegardes incrémentales
à J-1

Trash jour J



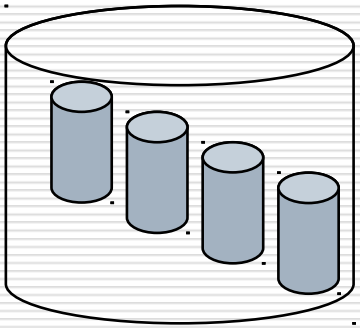
Protection contre les dégâts physiques



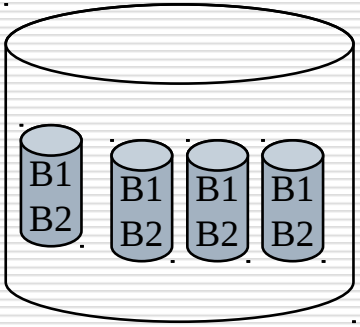
Principes des disques RAID :

→ L'unité de stockage est constituée de plusieurs disques durs constituant une grappe.

Lecture/écriture

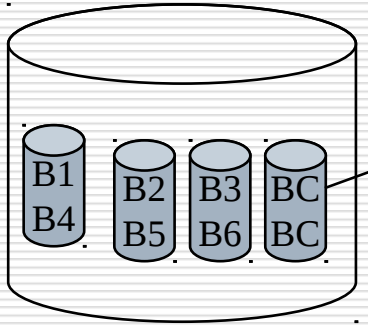


↳ RAID Niveau 1



Les données (blocs) sont dupliquées sur chaque disque

↳ RAID Niveau 4



Bloc de contrôle = f (B1,B2,B3)
Bloc de contrôle = f (B4,B5, B6)

Les données (blocs) sont répartis sur n-1 disques de la grappe
Le bloc n stocke une information redondante
des données permettant de les reconstituer en cas de perte