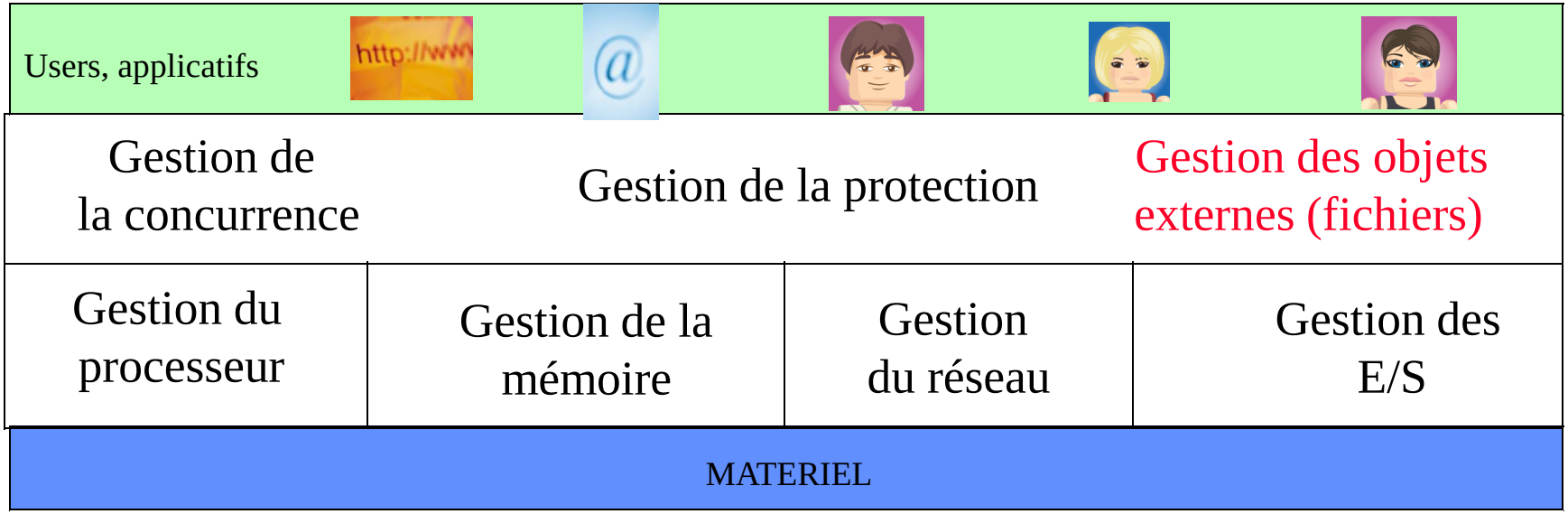
Système de gestion de fichiers



Gestion des objets externes : 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.*
  - *Le système d'exploitation gère les fichiers via le Système de gestion de fichiers (SGF)*
  - *Deux vues : une vue logique (utilisateur) ; une vue physique (système)*

A. Vue logique du SGF

- Le fichier logique est la vue de l'utilisateur de l'ensemble des données mémorisées sur le support de masse
  - Un type de donnée (programmation)
  - Un ensemble de données groupées sous forme d'enregistrements

- Le SGF est vue comme une arborescence de fichiers

Notion de fichier logique

```
/*********************************************************************/
/*            Exemple de manipulation d'un fichier:          */
/*              création, positionnement, fermeture          */
/*********************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
main()
{
struct eleve {
            char nom[10];
            int note;
           };
int fd, i, ret;
struct eleve un_eleve;
fd = open ("/home/delacroix/fichnotes",O_RDWR|O_CREAT, S_IRUSR|S_IWUSR);
i = 0;
while (i<4)
{
  printf ("Donnez le nom de l'élève \n"); scanf ("%s", un_eleve.nom);
  printf ("Donnez la note de l'élève \n"); scanf ("%d", &un_eleve.note);
  write (fd, &un_eleve, sizeof(un_eleve));
  i = i + 1;
}
ret = lseek(fd,0,SEEK_SET);
printf ("la nouvelle position est %d\n", ret);
i = 0;
while(i<4)
{
  read (fd, &un_eleve, sizeof(un_eleve));
  printf ("le nom et la note de l'élève sont %s, %d\n", un_eleve.nom,
un_eleve.note);
  i = i + 1;
}
close(fd);
```

- En programmation, un fichier logique est un type de donnée sur lequel peuvent être appliquées des opérations spécifiques. C'est un ensemble d'enregistrements, désigné par un nom, Accessible selon différentes méthodes d'accès

Liaison via le SGF avec le fichier physique
Représentation du fichier interne au programme

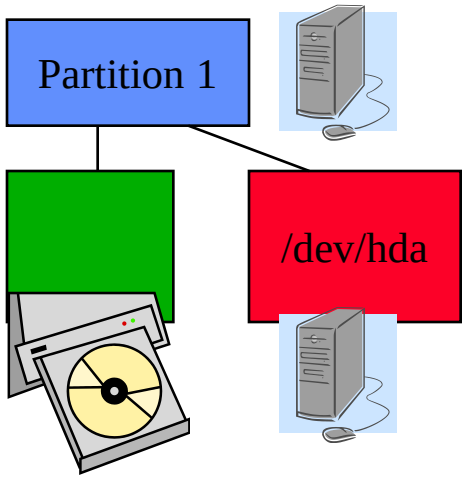
Accès séquentiel

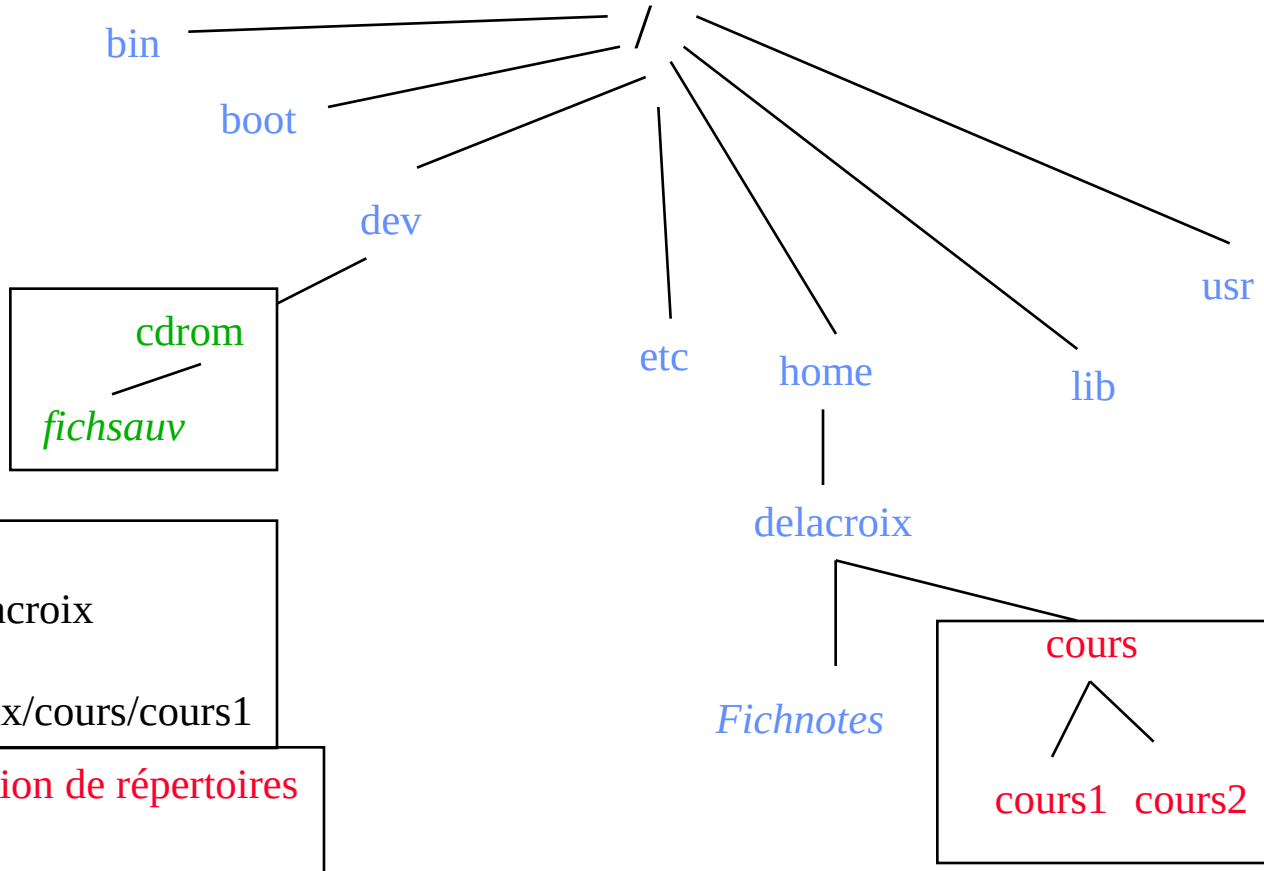
Accès direct

Rupture de la Liaison avec le fichier physique

Arborescence de fichiers

fd = open ("/home/delacroix/fichnotes",….)
Arborescence **unique** ("/") pouvant regrouper plusieurs partitions locales ou distantes montées. Le **point de montage** ("mount point") est le répertoire de la structure de fichier (n'importe lequels des répertoires de l'arborescence) à partir duquel sera monté le support.





Fichiers / répertoires
Répertoire de travail /home/delacroix
Chemin relatif : cours/cours1
Chemin absolu : /home/delacroix/cours/cours1

mkdir, rmdir : création, destruction de répertoires
cp : copie fichier
ls : lister le contenu d'un répertoire
rm : détruire un fichier

mount /dev/hda /home/delacroix

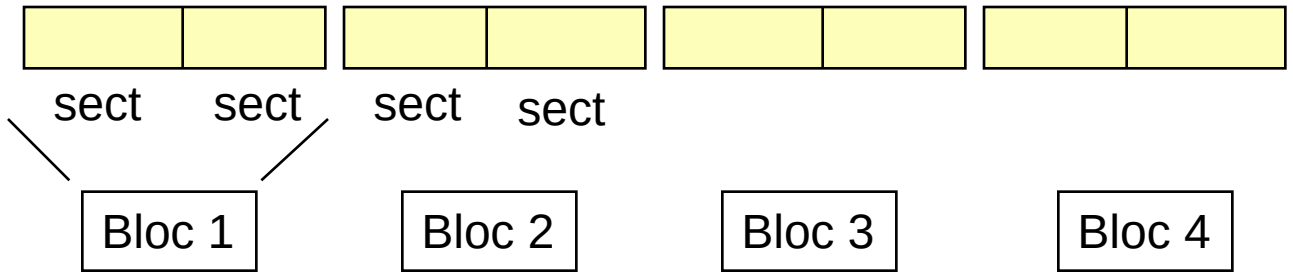
B. Vue physique

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

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

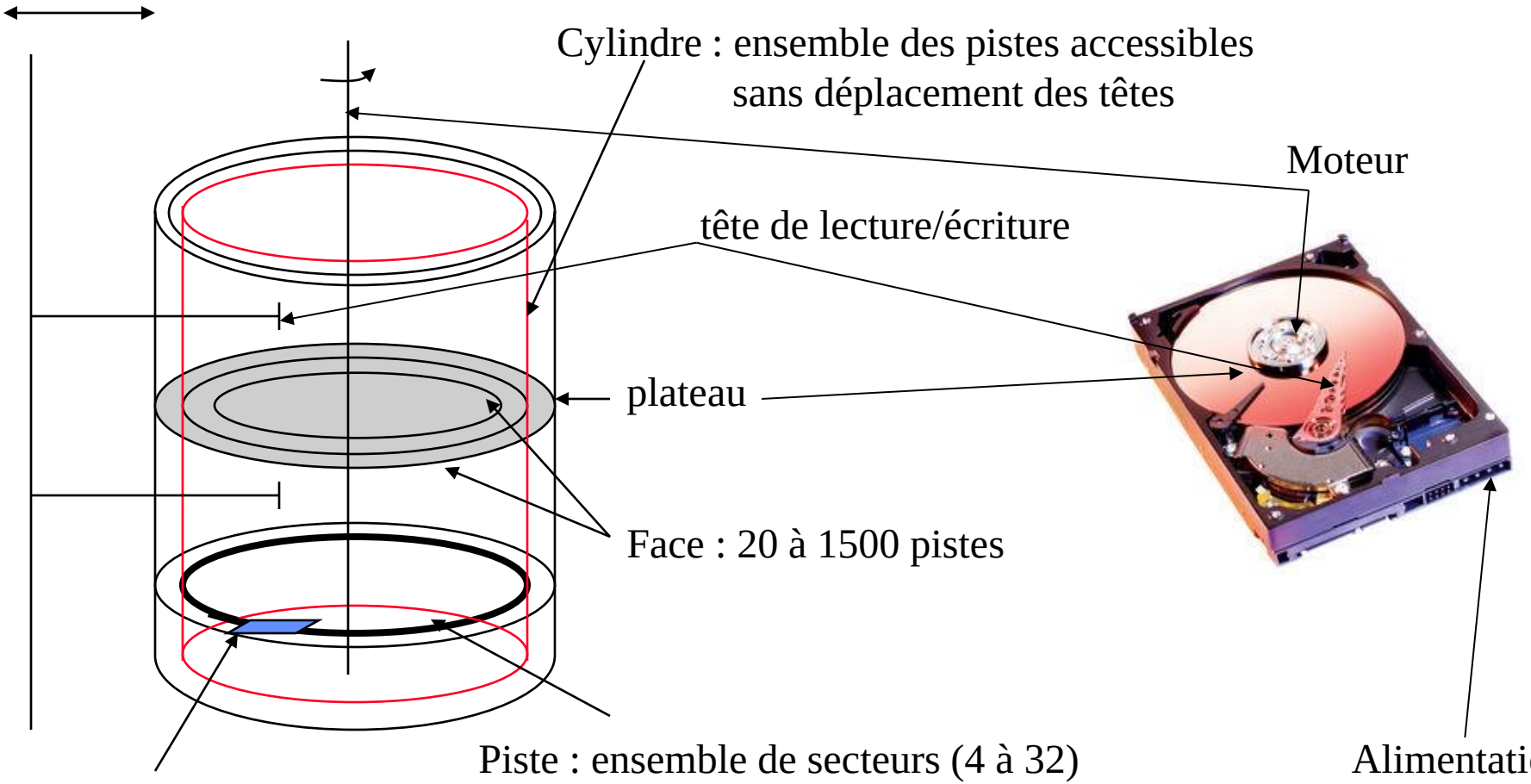
    1. allocation contiguë (séquentielle simple)
    2. allocation par blocs chainés
    3. allocation indexée

Les opérations de lecture et d'écriture du SGF se font bloc par bloc
1 bloc = 2 secteurs de 512 octets soit 1KO



Structure du disque dur

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



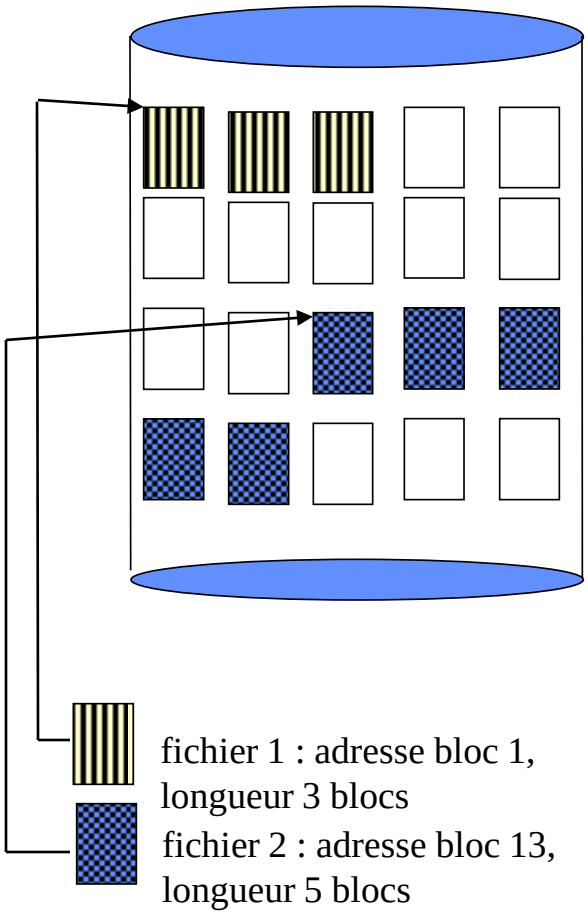
Secteur : plus petite unité d'information accessible

32 à 4096 octets

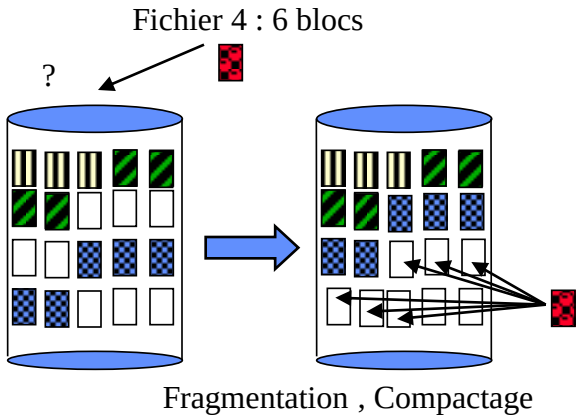
512

Allocation contiguë

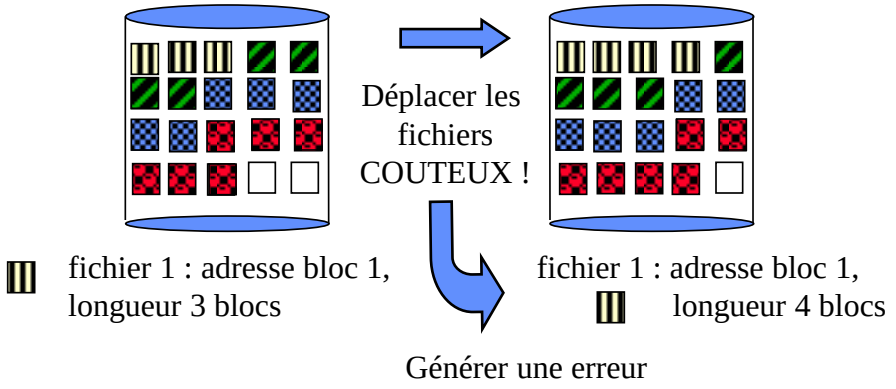
- Un fichier occupe un ensemble de blocs contigus sur le disque
- Allocation : trouver un trou suffisant (First Fit, Best Fit)
- Difficultés



- création d'un nouveau fichier

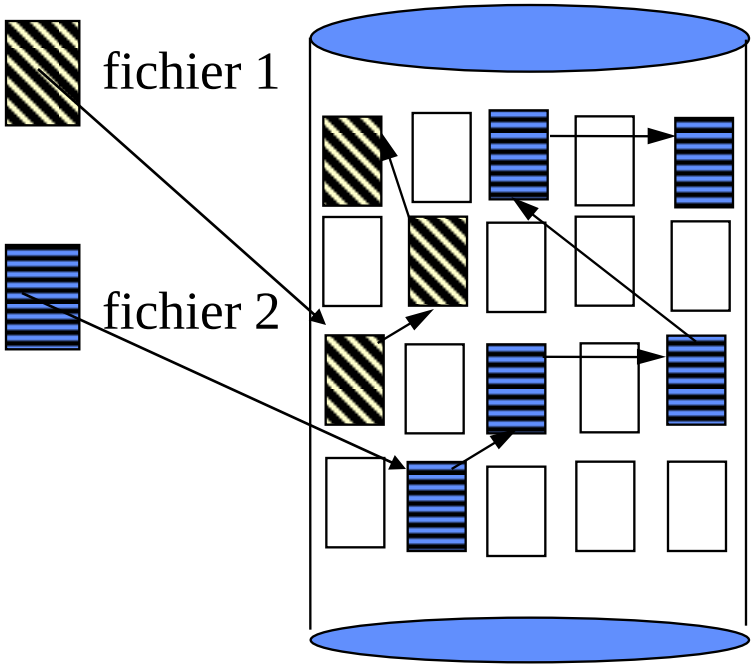


- extension du fichier



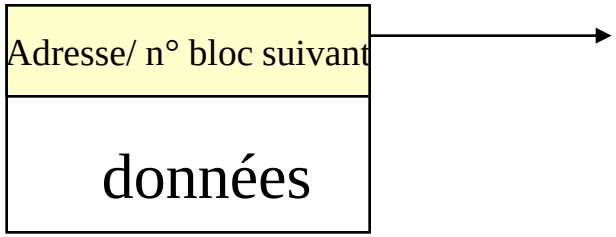
Allocation par bloc chainée

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



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

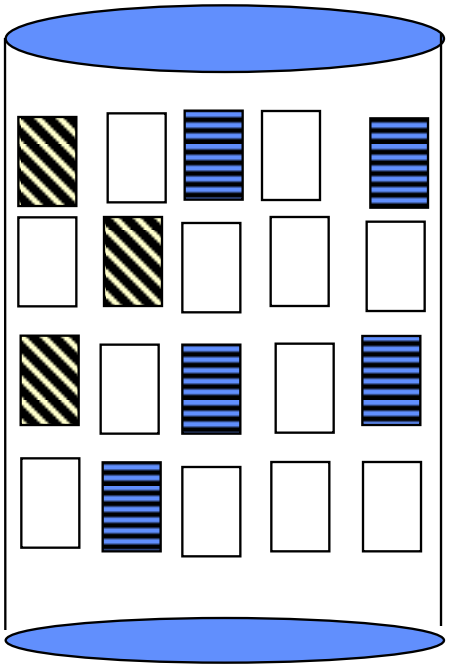
• Difficultés :
- le chaînage du bloc suivant occupe de la place dans un bloc

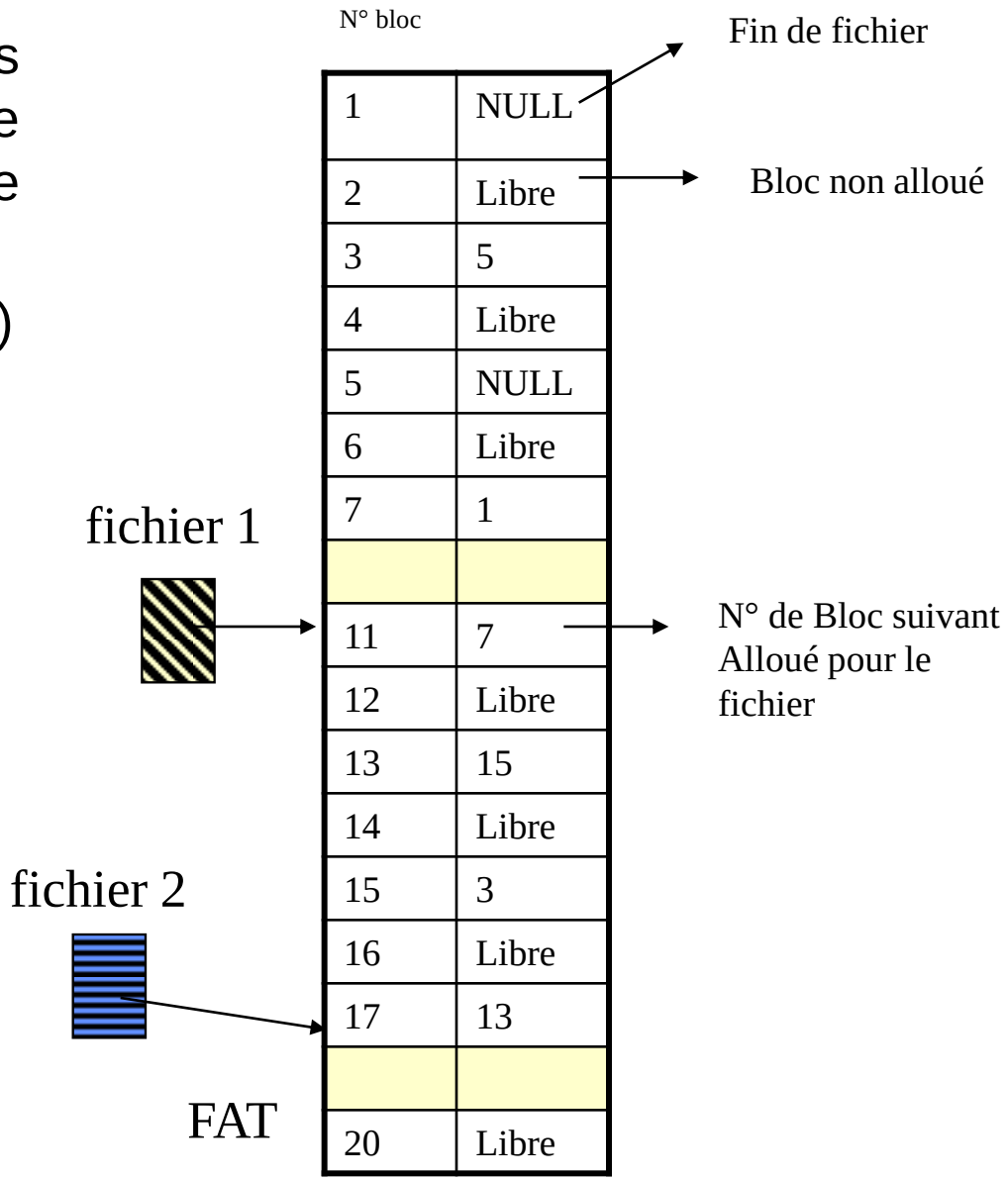


Allocation par bloc chainée : variante

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

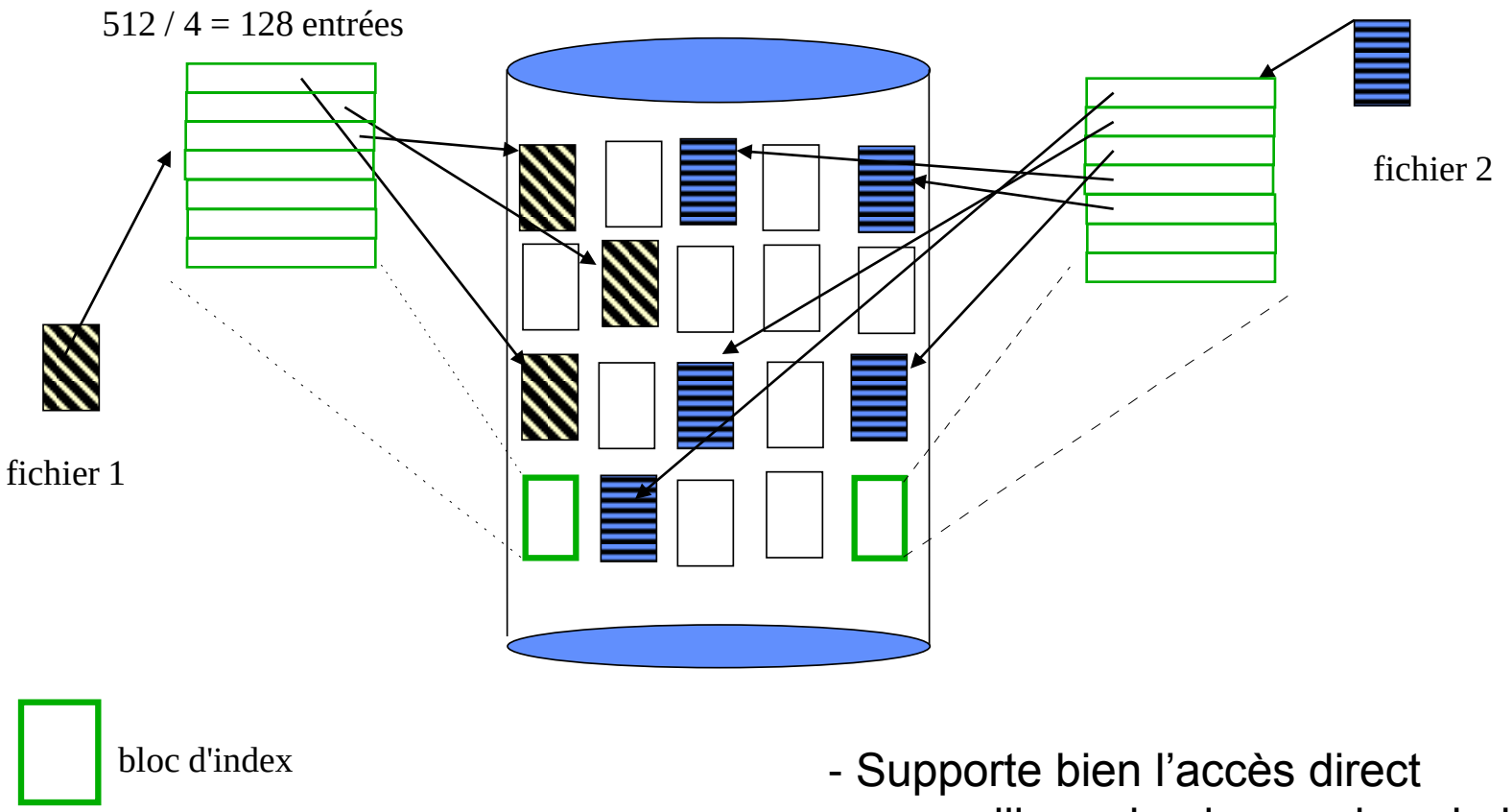
(exemple systèmes windows)





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

Gestion de l'espace libre

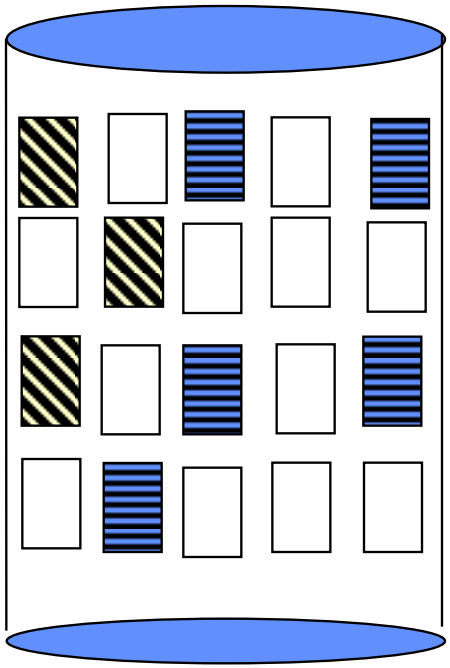
- Le système maintient une liste d'espace libre, qui mémorise tous les blocs disque libres (non alloués)
  - Création d'un fichier : recherche dans la liste d'espace libre de la quantité requise d'espace et allocation au fichier : l'espace alloué est supprimé de la liste
  - Destruction d'un fichier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

Il existe différentes représentations possibles de l'espace libre

- vecteur de bits
- liste chainée des blocs libres

Gestion de l'espace libre par un vecteur de bits

- La liste d'espace libre est représentée par un vecteur binaire, dans lequel chaque bloc est figuré par un bit.
  - Bloc libre : bit à 1
  - Bloc alloué : bit à 0



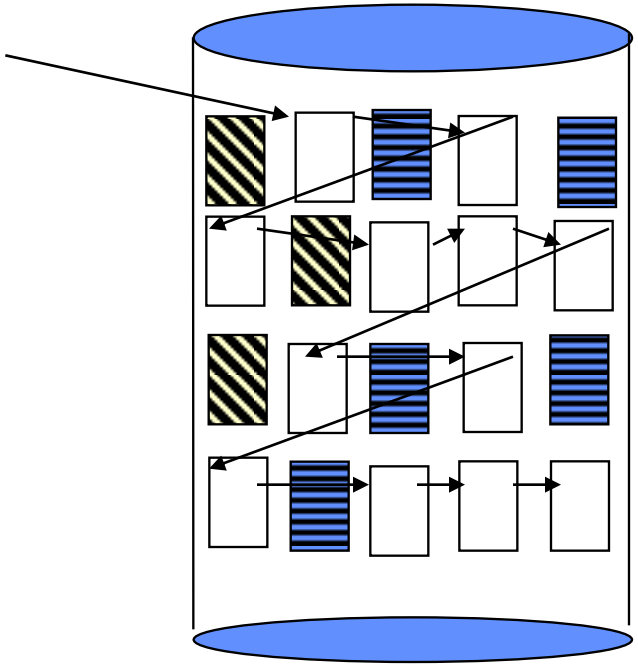
01010101110101010111

- Facilité de trouver n blocs libres consécutifs
- Système Macintosh

Gestion de l'espace libre par liste chainée

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

Liste des blocs libres



➢ Parcours de la liste couteux

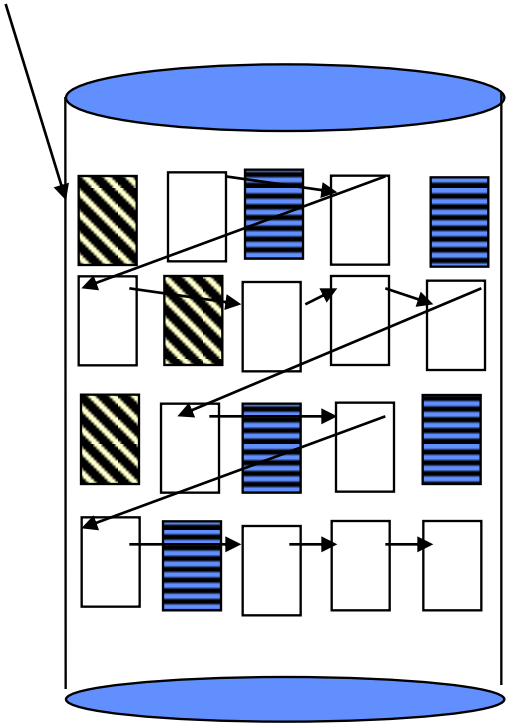
➢ Difficile de trouver un groupe de blocs libres

→ Variante par comptage

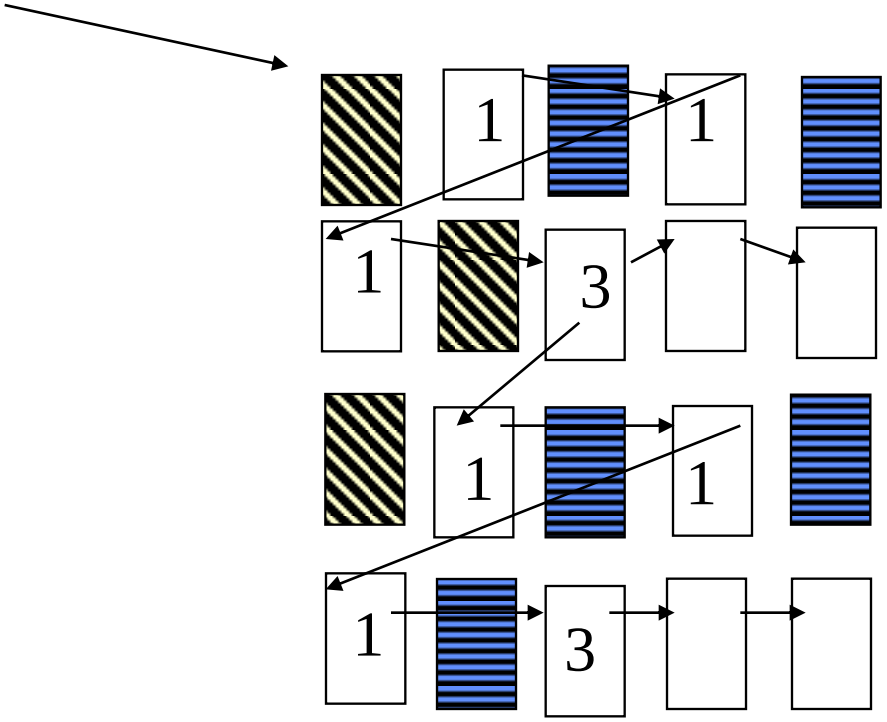
Gestion de l'espace libre par liste chainée: variante avec comptage

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

Liste des blocs libres

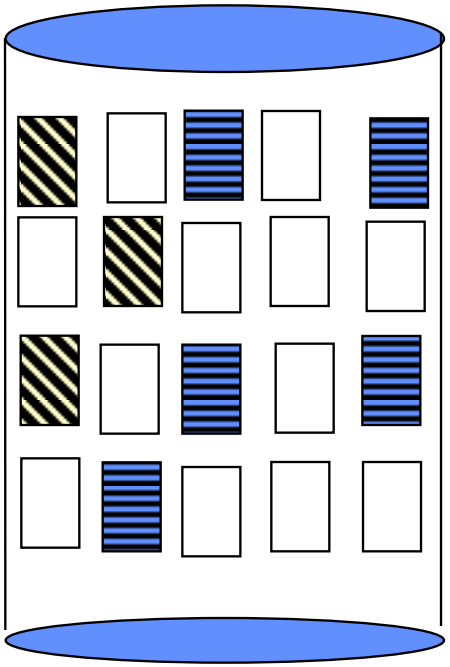


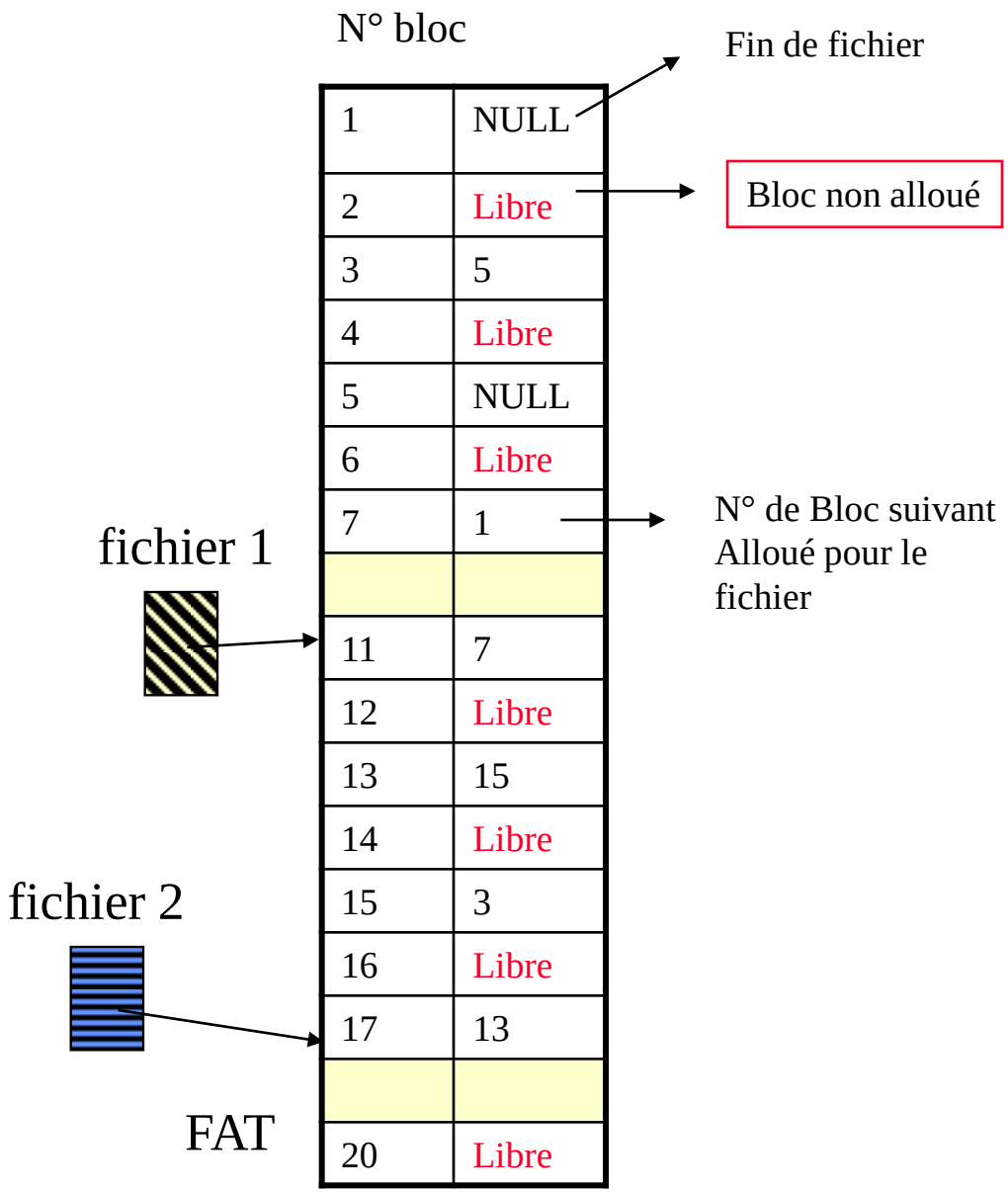
Liste des blocs libres



Gestion de l'espace libre

- La FAT intègre directement la gestion de cet espace.



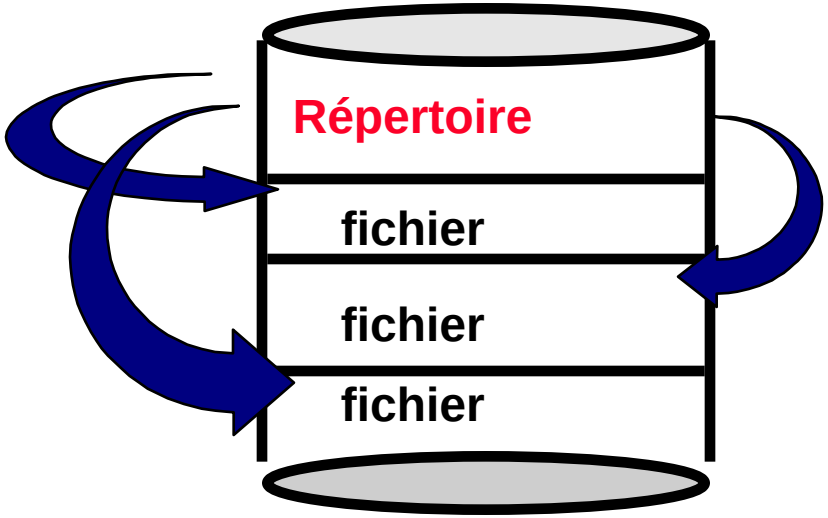


C. Correspondance fichier logique - fichier physique Désignation des fichiers : le répertoire

Le répertoire

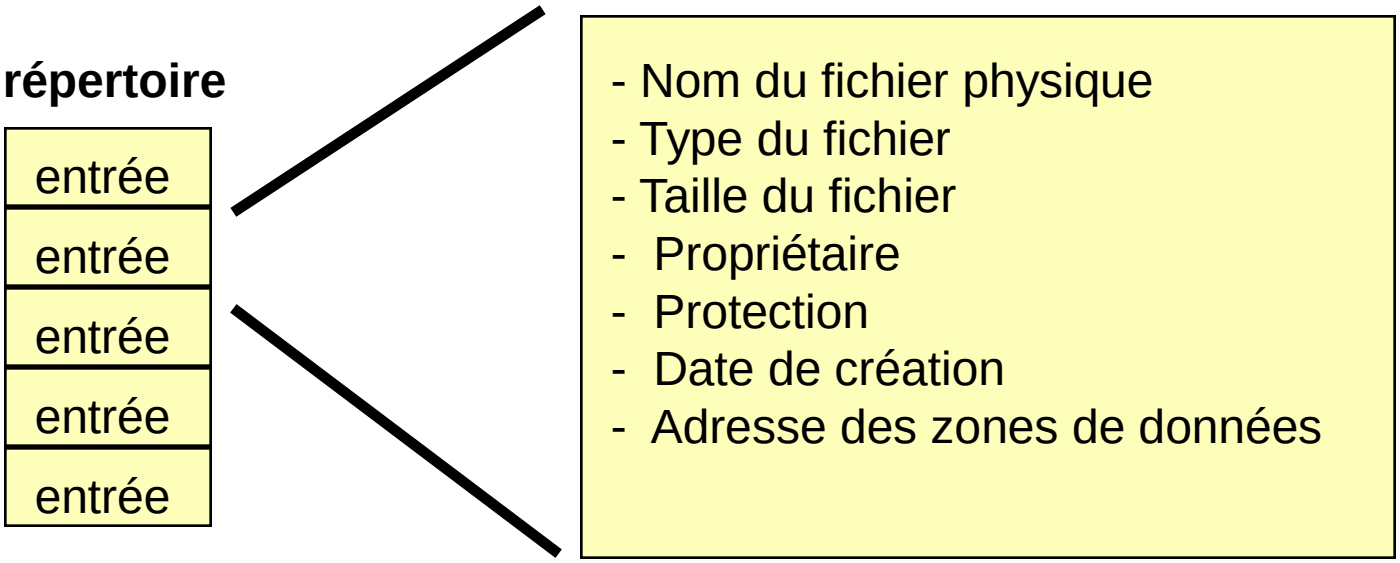
– Le répertoire est une table sur le support permettant de référencer tous les fichiers existants du SGF avec leur nom et leurs caractéristiques principales

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



Le répertoire

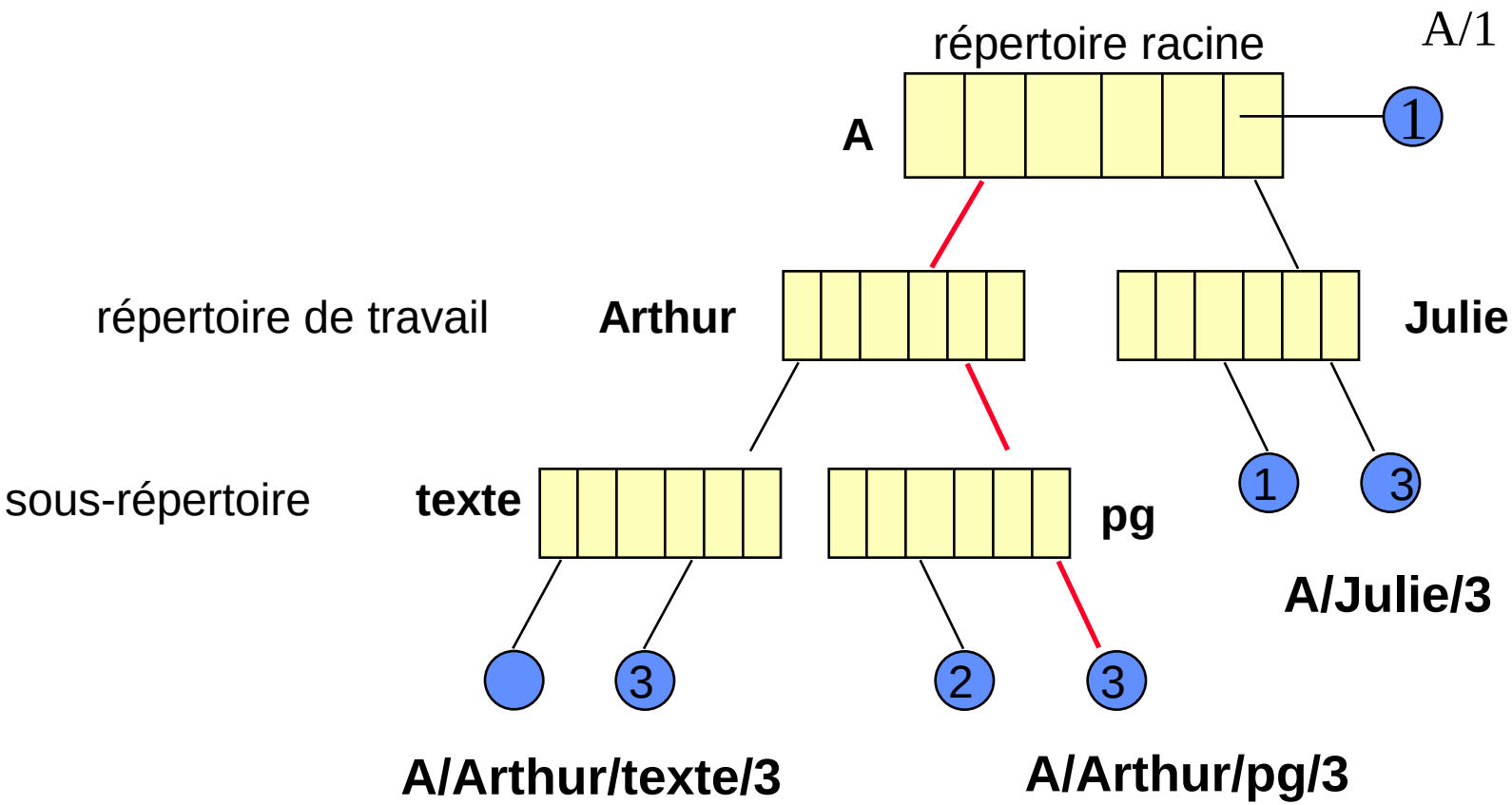
Un répertoire est une zone disque réservée par le SGF.
Le répertoire comprend un certain nombre d'entrées.
Une entrée est allouée à chaque fichier du SGF



1 entrée : attributs du fichier physique

Organisation des répertoires

- Répertoire à structure arborescente :
  - ☞ chaque utilisateur dispose d'un sous-répertoire propre (*Répertoire de travail)*
  - ☞ l'utilisateur peut créer des sous-répertoires à l'intérieur de son répertoire de travail



Système de gestion de fichiers LINUX

- Structure d'un fichier Linux
- Partition Linux

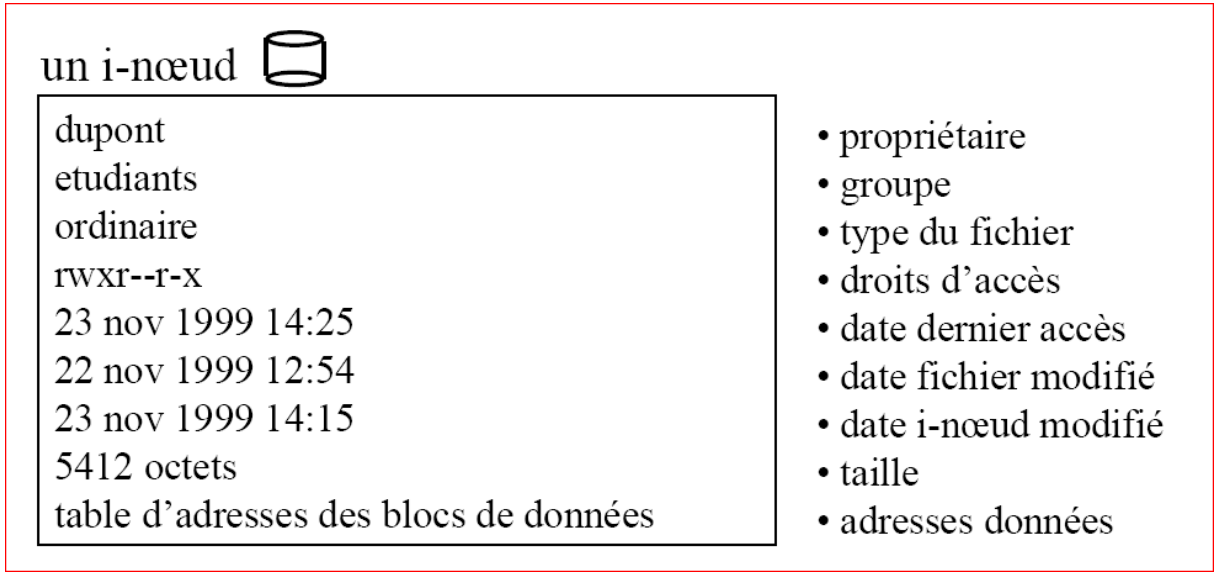
- Structure de gestion des fichiers dans un processus
- Mécanisme du buffer cache

Fichier Linux

- Identifié par un nom, sans structure logique (suite d'octets)
- La méthode d 'allocation mise en œuvre est de type allocation indexée.
- Un fichier Linux est composé d'un descripteur appelé « inode » et de blocs physiques, qui sont soit des blocs d'index, sont des blocs de données. Les blocs de données sont alloués au fur et à mesure de l'extension du fichier.
- Un bloc est identifié par un numéro codé sur 4 octets. La taille d'un bloc est un multiple de la taille d'un secteur (512 octets)

Fichier Linux : inode

- Structure stockée sur le disque, allouée à la création du fichier et répérée par un numéro
- Contient les attributs du fichier :
  - Nom
  - Type : fichiers normaux, répertoires, *périphériques, tubes nommés , sockets*
  - Droits d'accès
  - Heures diverses
  - Taille du fichier en octets
  - Table des adresses des blocs de données



Fichier Linux : structure

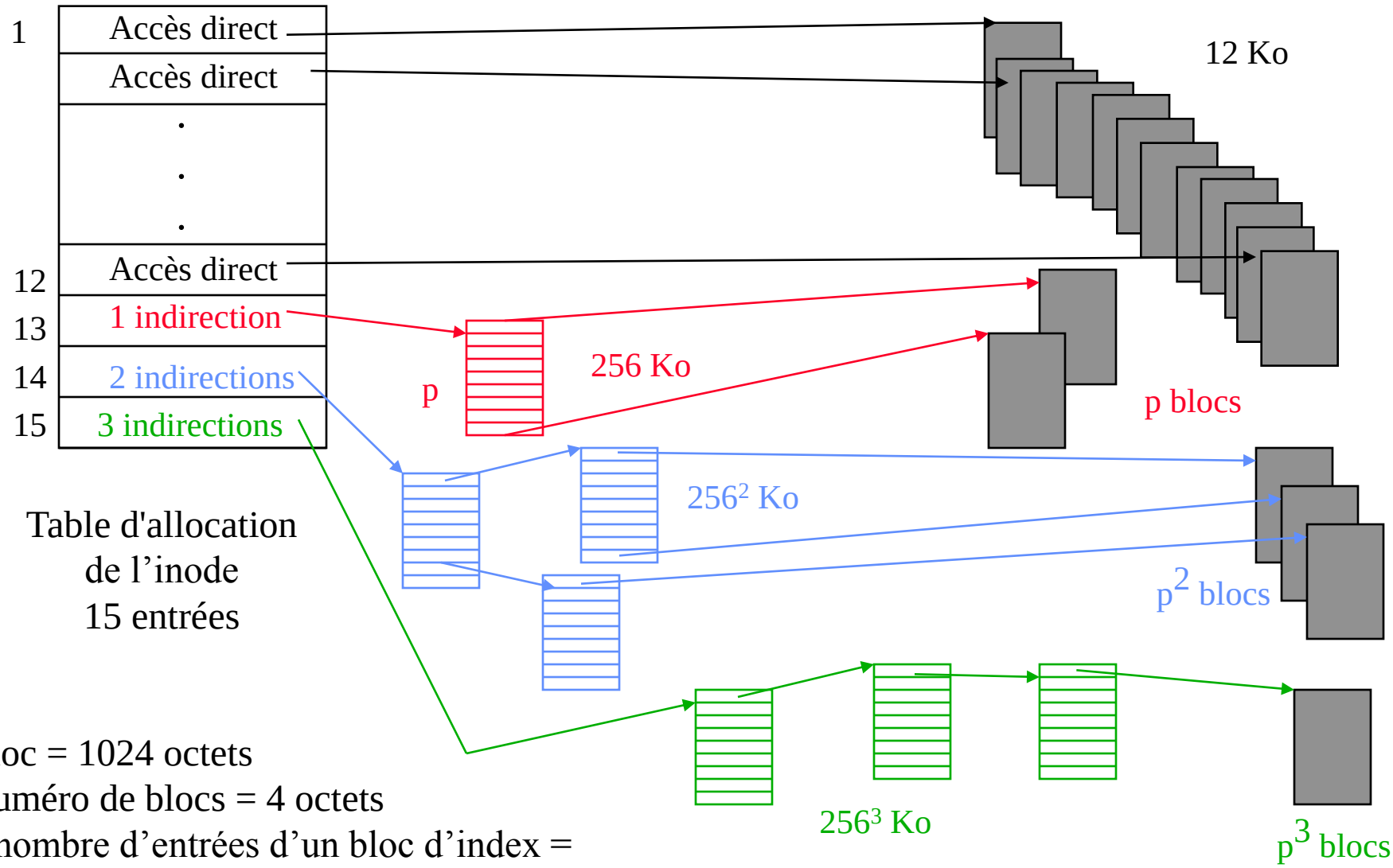
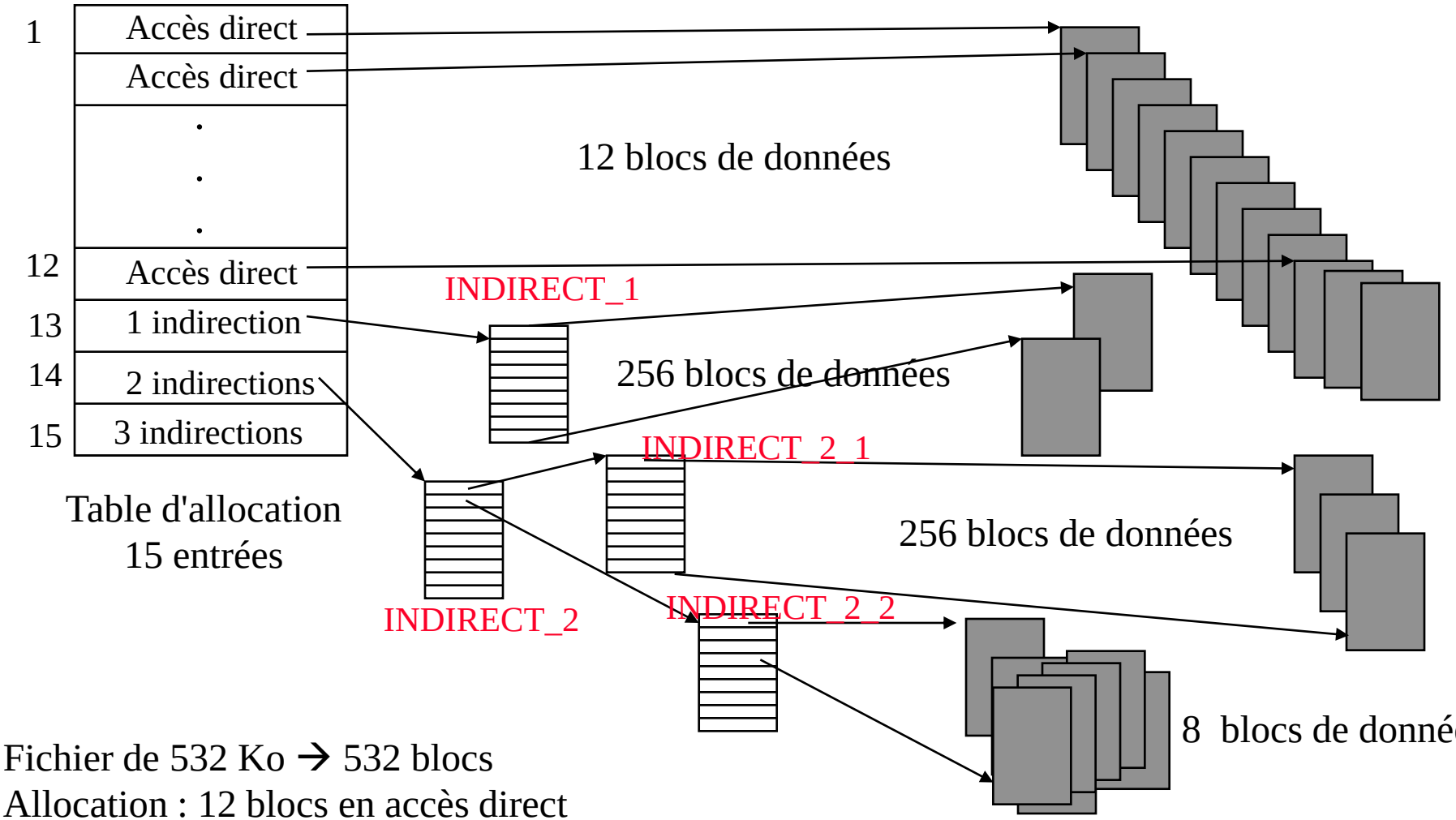


Table d'allocation
de l'inode
15 entrées

Bloc = 1024 octets
Numéro de blocs = 4 octets
p nombre d'entrées d'un bloc d'index =
taille bloc / taille numéro de bloc

EXEMPLE

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



Fichier de 532 Ko → 532 blocs

Allocation : 12 blocs en accès direct

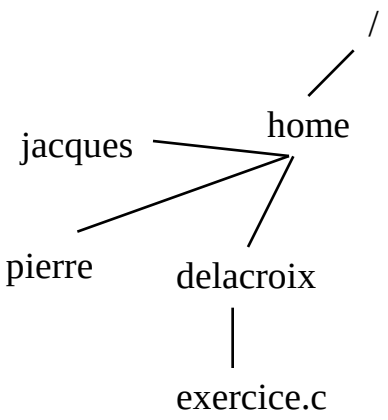
: 256 blocs de données pointés par le bloc index INDIRECT 1

: restent 532 – 12 – 256 = 264 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

Répertoire Linux : structure

- Fichier spécial avec des blocs de données contenant des couples (nom de fichiers, n°inode)

Inode	Longueur de l'entrée	Longueur du nom de fichier	Type de fichier	Nom
23	12	1	2 (rep)	.\0\0\0
24	12	2	2 (rep)	..\0\0
68	20	10	1 (fichier régulier)	exercice.c\0 \0



Parcours des blocs de données du fichier –répertoire home

12	pierre	15	jacques	17	delacroix

Parcours des blocs de données du fichier –répertoire d'inode 17 (delacroix)

17	.	24	..	68	exercice .c

Fichiers et Répertoire Linux : droits d'accès

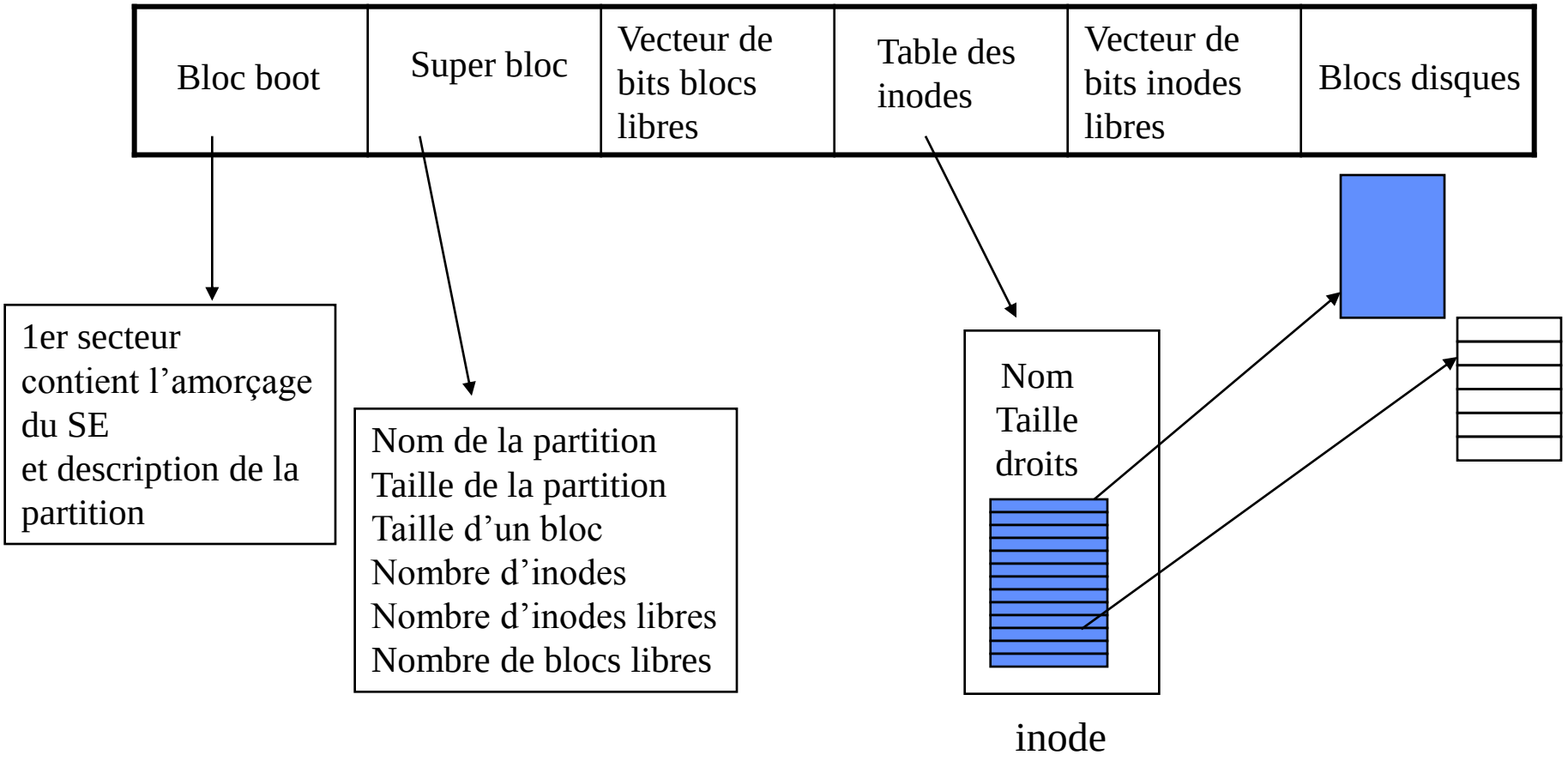
- Définition de trois entités :
  - le propriétaire : celui qui a crée le fichier
  - le groupe auquel appartient le propriétaire : le groupe de travail
  - les autres : tous les autres

- Les droits :
  - Permission en lecture (r) ; (listage pour répertoire)
  - Permission en écriture (w) ; (création et suppression pour répertoire)
  - Permission en exécution (x); (traversée du répertoire)

```
> ls -la
drwxr-xr-x   2        delacroi      ensinf       4096      Oct 22  1998        repertoire
-rw-r--r--   1        delacroi      ensinf       6401      Jan  8  1997        eleve.c
-rwxr-xr-x   1        delacroi      ensinf      24576      Dec 15  1998        essai
-rw-r--r--   1        delacroi      ensinf       67        Dec 15  1998        essai.c

> chmod a+w essai.c
> ls -l essai.c
-rw-rw-rw-   1        delacroi      ensinf       67        Dec 15  1998        essai.c
```

Partition Linux : structure

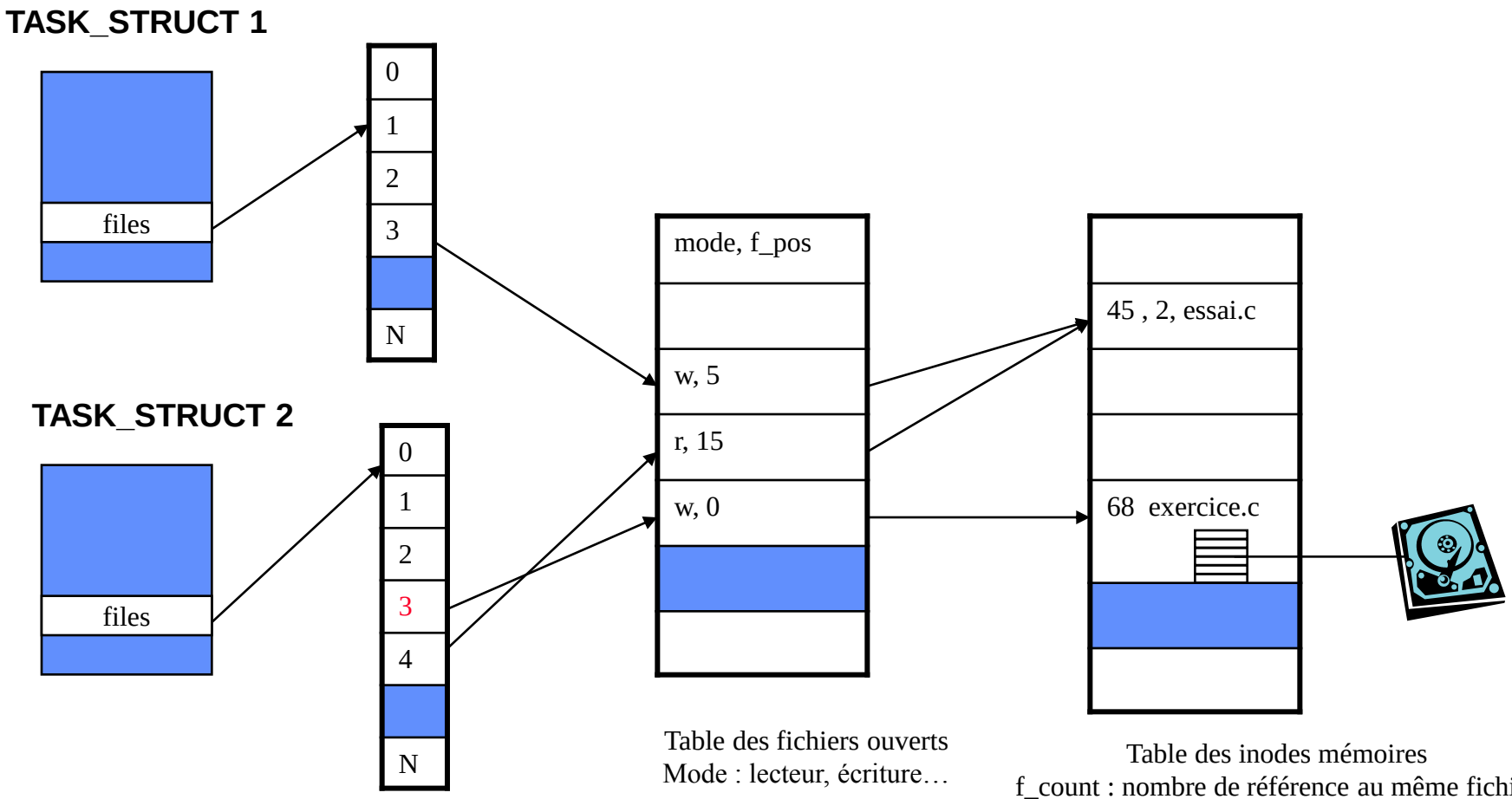


Structure de gestion des fichiers dans un processus

TASK_STRUCT

volatile long state; - - ***état du processus***
long counter; - - ***quantum***
long priority; -- ***priorité SCHED_OTHER***
struct task_struct *next_task, *prev_task; -- ***chainage PCB***
struct task_struct *next_run, *prev_run; -- ***chainage PCB Prêt***
int pid; -- ***pid du processus***
struct task_struct *p_opptr, *p_pptr, *p_cptr,; -- ***pointeurs PCB père originel, père actuel, fils***
long need_resched; -- ***ordonnancement requis ou pas***
long utime, stime, cutime, cstime;
-- ***temps en mode user, noyau, temps des fils en mode user, noyau***
unsigned long policy; -- ***politique ordonnancement SCHED_RR, SCHED-FIFO, SCHED_OTHER***
unsigned rt_priority; -- ***priorité SCHED_RR et SCHED_FIFO***
struct thread_struct tss; -- ***valeurs des registres du processeur***
struct mm_struct *mm; -- ***contexte mémoire***
struct files_struct *files ; -- table fichiers ouverts
struct signal_struct *sig ; -- table de gestion des signaux

Structure de gestion des fichiers dans un processus



fp = open (« /home/delacroix/exercices.c » , « w »)
fp = 3

17	.	34	..	68	exercice .c

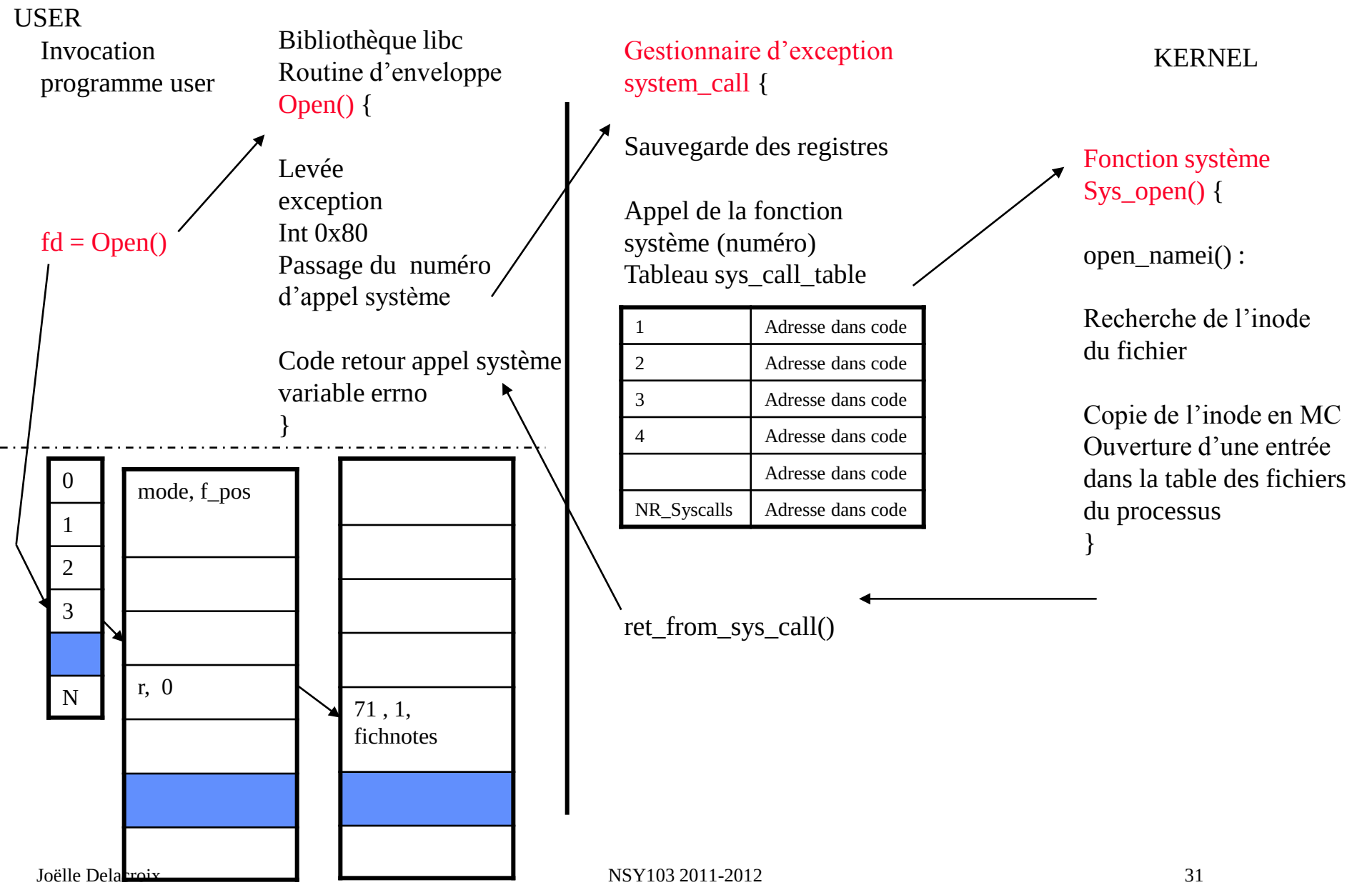
Ouverture, lecture et écriture d'un fichier : le buffer cache

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
main()
{
struct eleve {
            char nom[10];
            int note;
           };
int fd;
struct eleve un_eleve;
fd = open ("/home/delacroix/fichnotes",O_RDONLY);
while(read (fd, &un_eleve, sizeof(un_eleve)) > 0);
  printf ("le nom et la note de l'élève sont %s, %d\n", un_eleve.nom, un_eleve.note);

  printf ("Donnez le nom de l'élève \n"); scanf ("%s", un_eleve.nom);
  printf ("Donnez la note de l'élève \n"); scanf ("%d", &un_eleve.note);
  write (fd, &un_eleve, sizeof(un_eleve));

close(fd);
}
```

Ouverture d'un fichier :
fd = open ("/home/delacroix/fichnotes",O_RDONLY);



Lecture d'un fichier : le buffer cache

while(read (fd, &un_eleve, sizeof(un_eleve)) > 0);

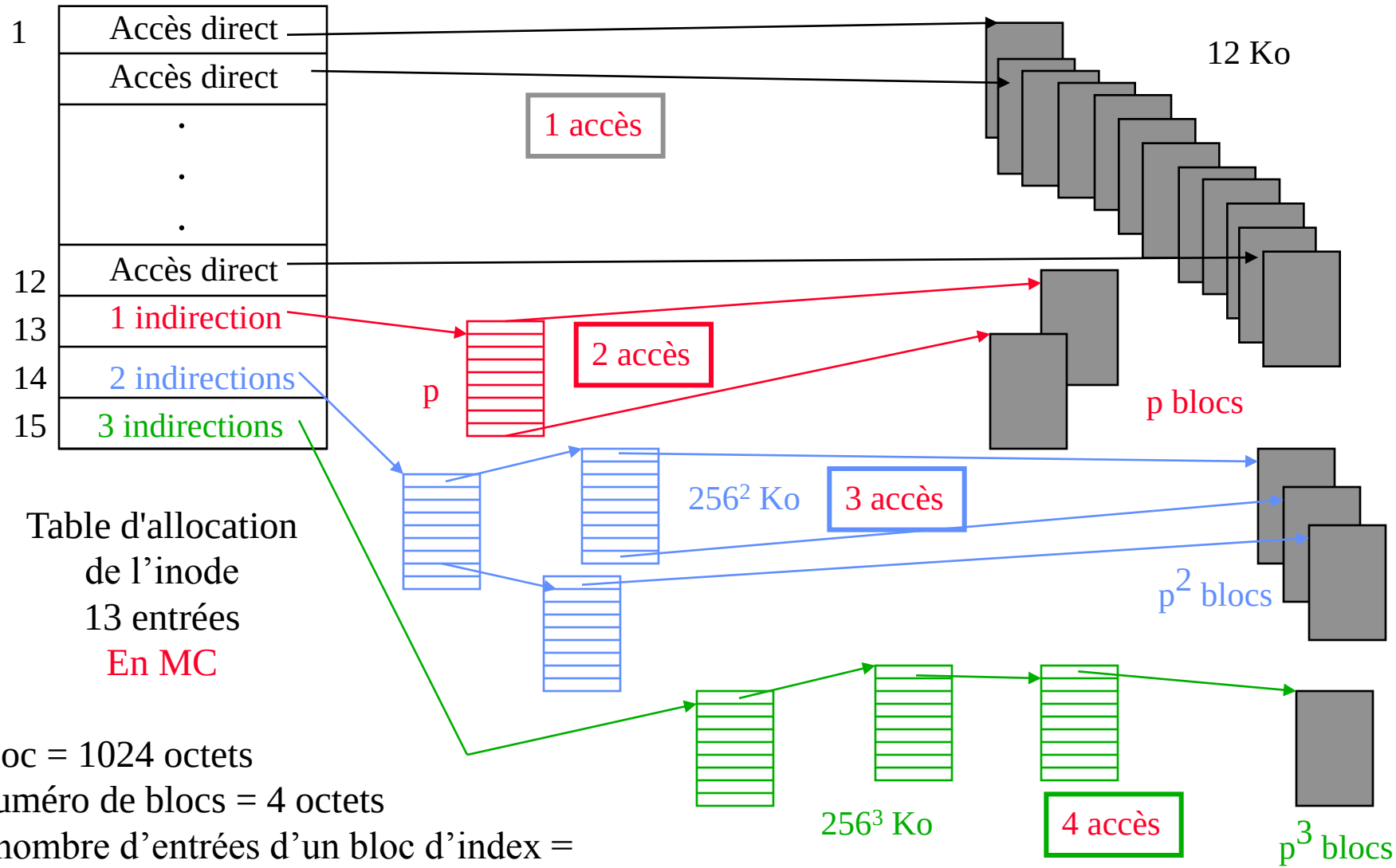
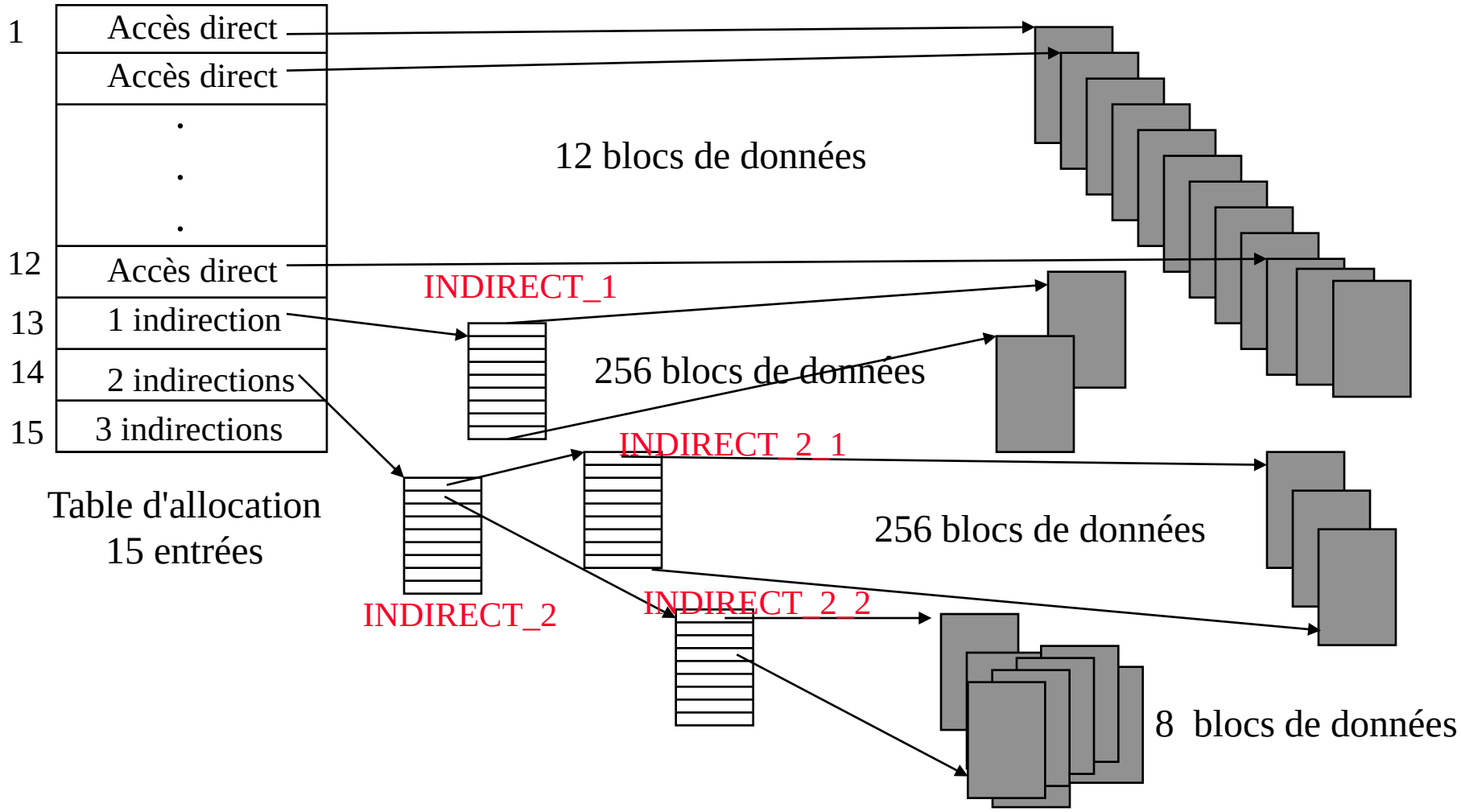


Table d'allocation
de l'inode
13 entrées
En MC

Bloc = 1024 octets
Numéro de blocs = 4 octets
p nombre d'entrées d'un bloc d'index =
taille bloc / taille numéro de bloc

EXEMPLE

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



Nombre d'accès disque pour lire le fichier :

12 + (2 * 256) + (3 *256) + (3 * 8) = 1 316 AD

Temps de lecture = 1316 * 10 ms = 13160 ms = 13 s

Lecture d'un fichier : le buffer cache

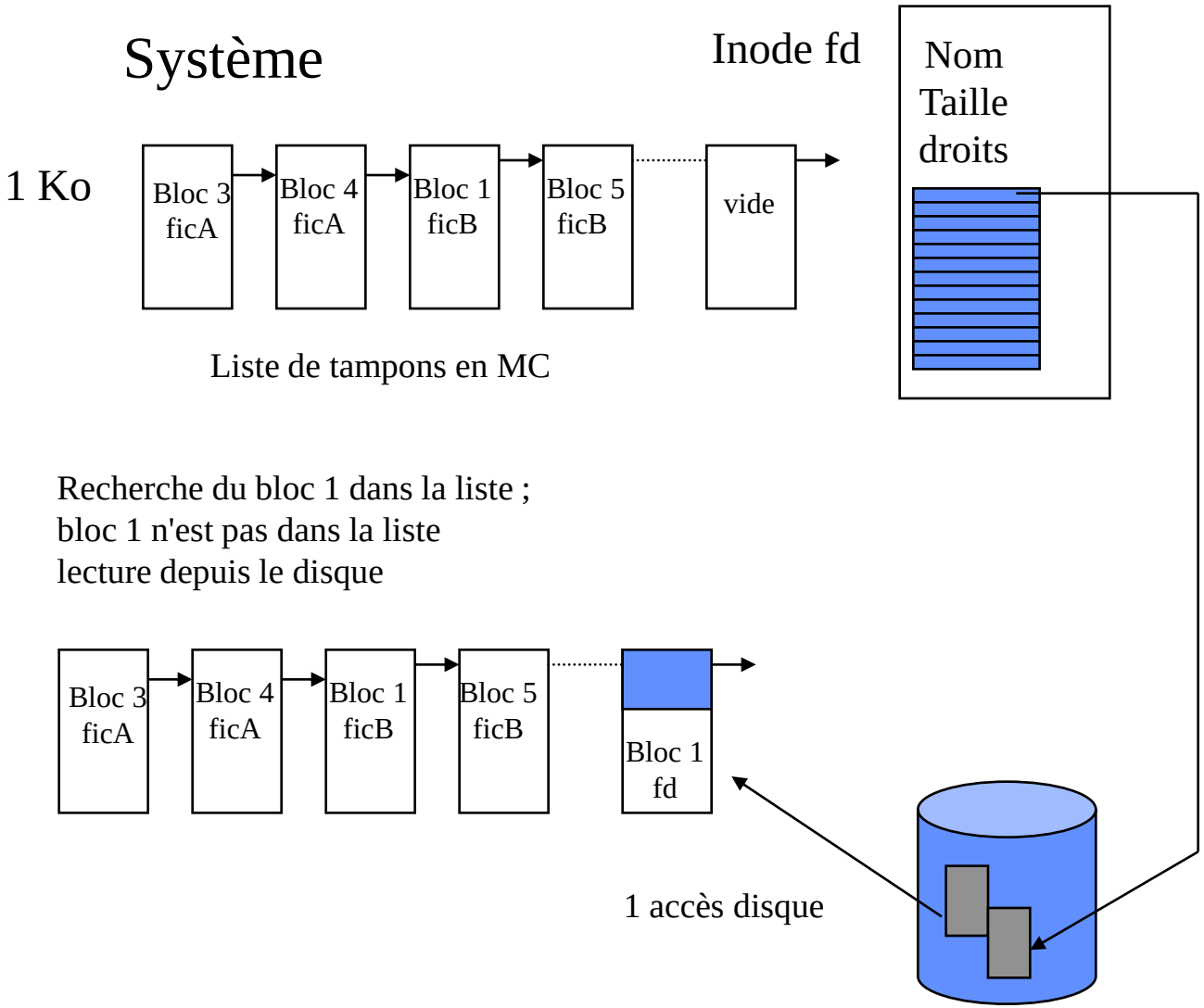
while(read (fd, &un_eleve, sizeof(un_eleve)) > 0);

- 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. Il est identifié par un numéro de bloc physique et numéro de périphérique.
- 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.

Lecture fichier
while(read (fd, &un_eleve, sizeof(un_eleve)) > 0);

1ère lecture : accès au bloc 1 de fd

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

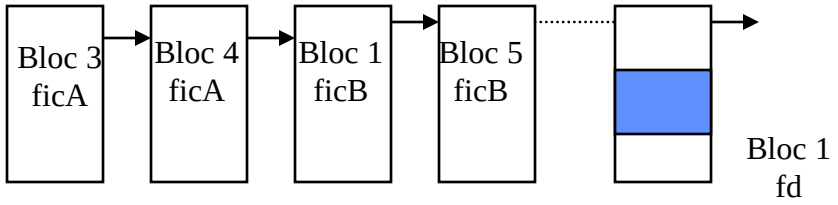


Lecture fichier

2ème lecture : accès au bloc 1 de fd

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

Système

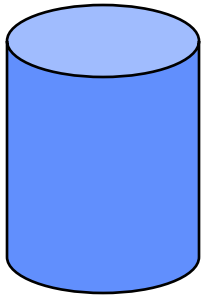


Liste de tampons

Recherche du bloc 1 de fd dans la liste
bloc 1 de fd est dans la liste

---

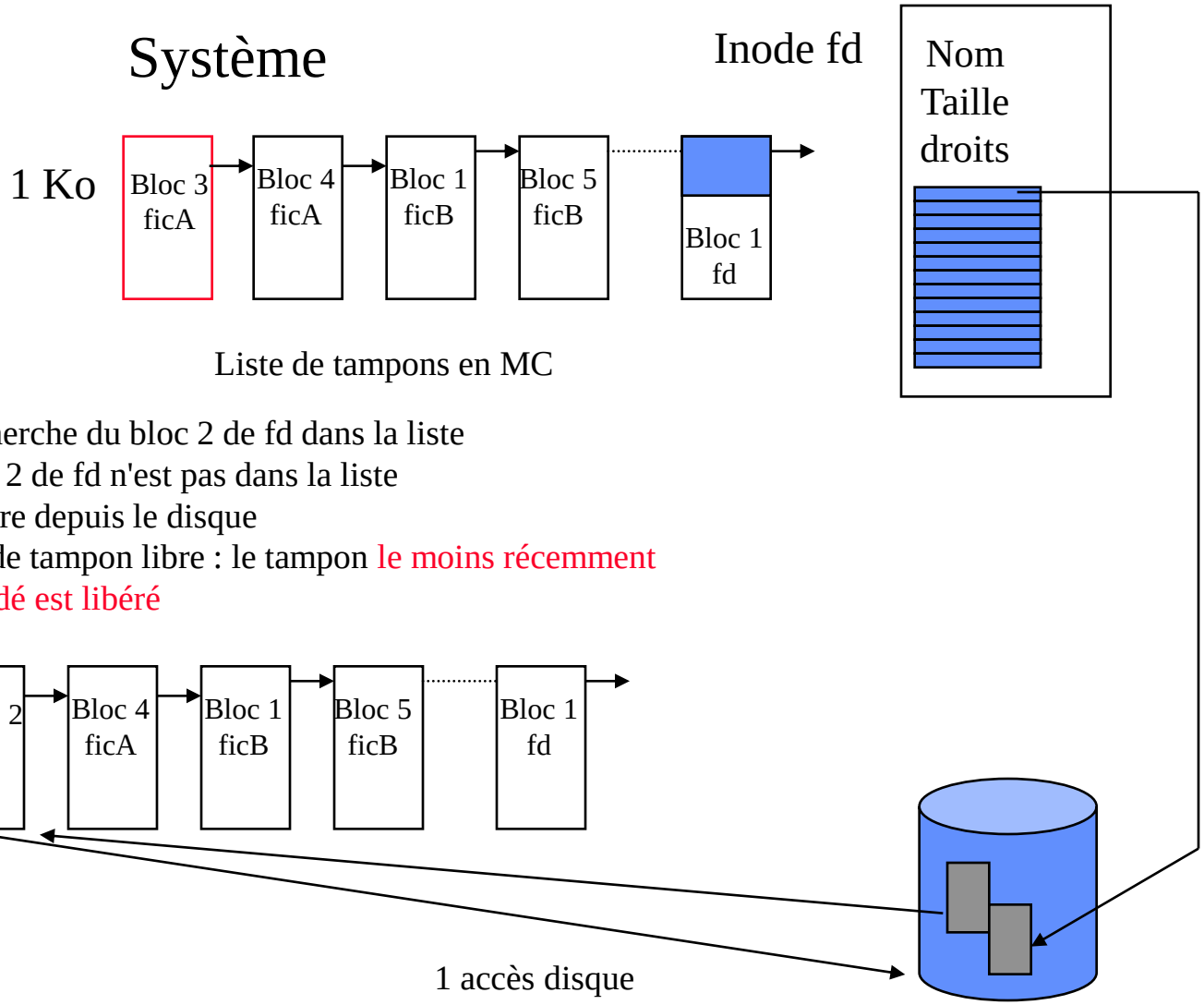
0 accès disque



Lecture fichier

33 ème lecture : accès au bloc 2 de fd

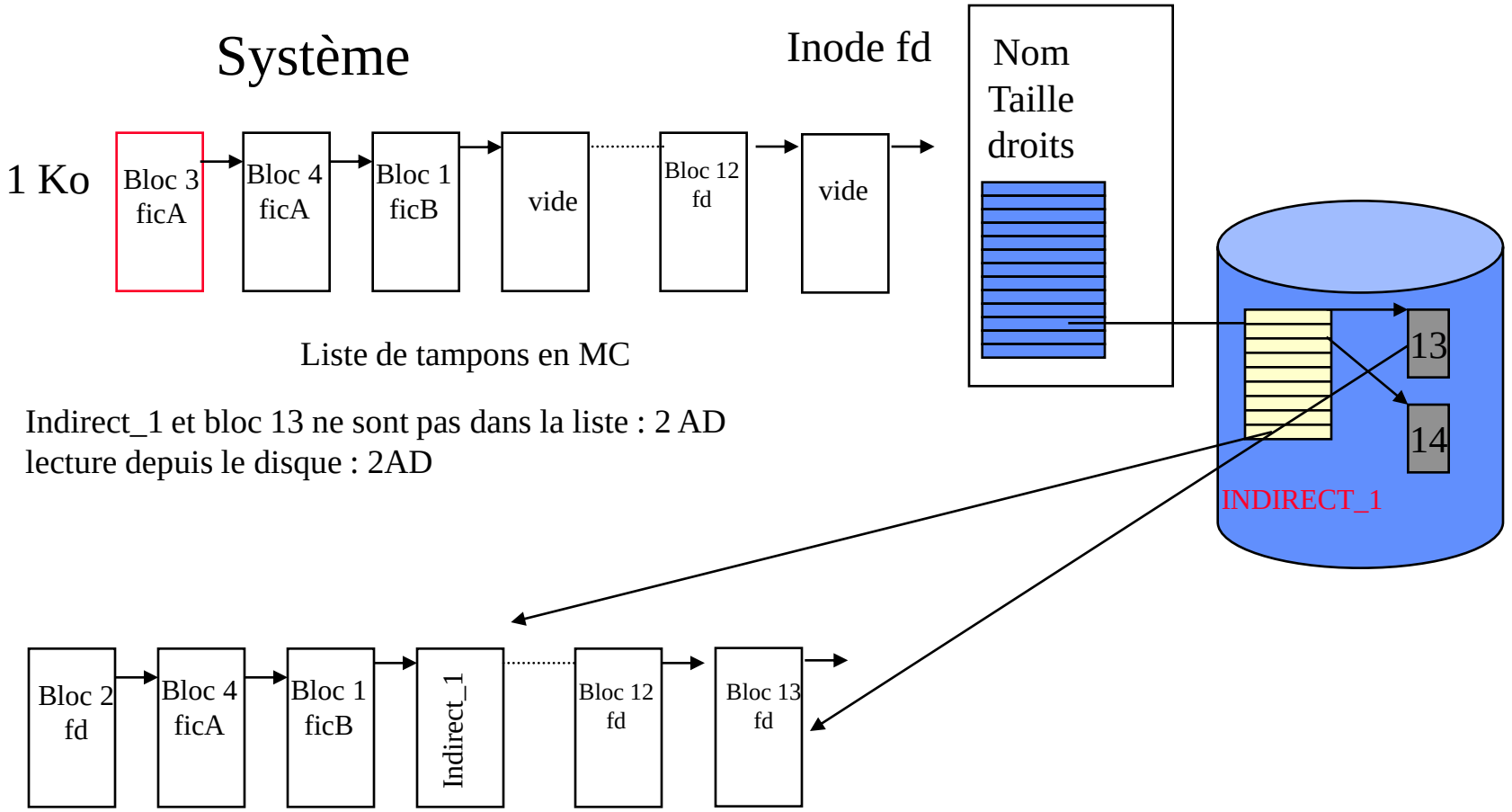
Bloc de 1 Ko, enrg de 32 octets → 32 enregistrements /bloc



Lecture fichier

385 ème lecture : accès au premier niveau d'indirection

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



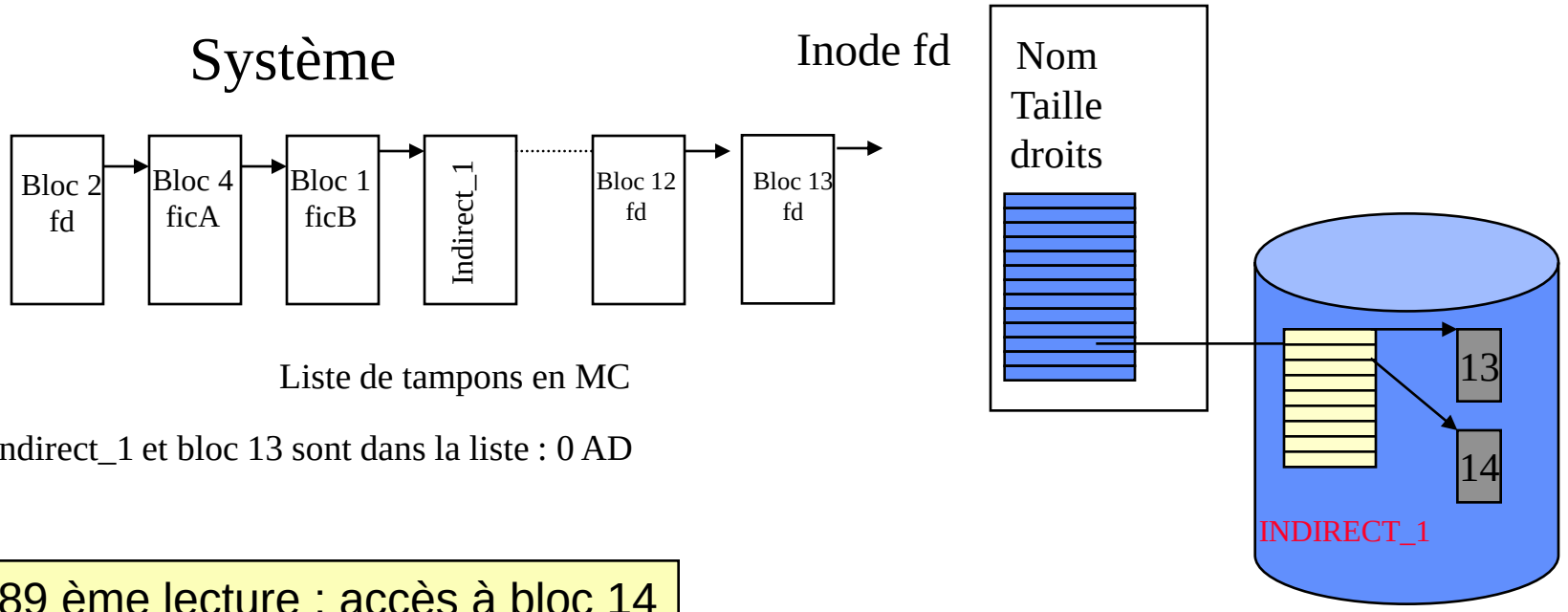
Indirect_1 et bloc 13 ne sont pas dans la liste : 2 AD
lecture depuis le disque : 2AD

1 accès disque

Lecture fichier

386- 388 ème lecture : accès au bloc 13

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



Indirect_1 et bloc 13 sont dans la liste : 0 AD

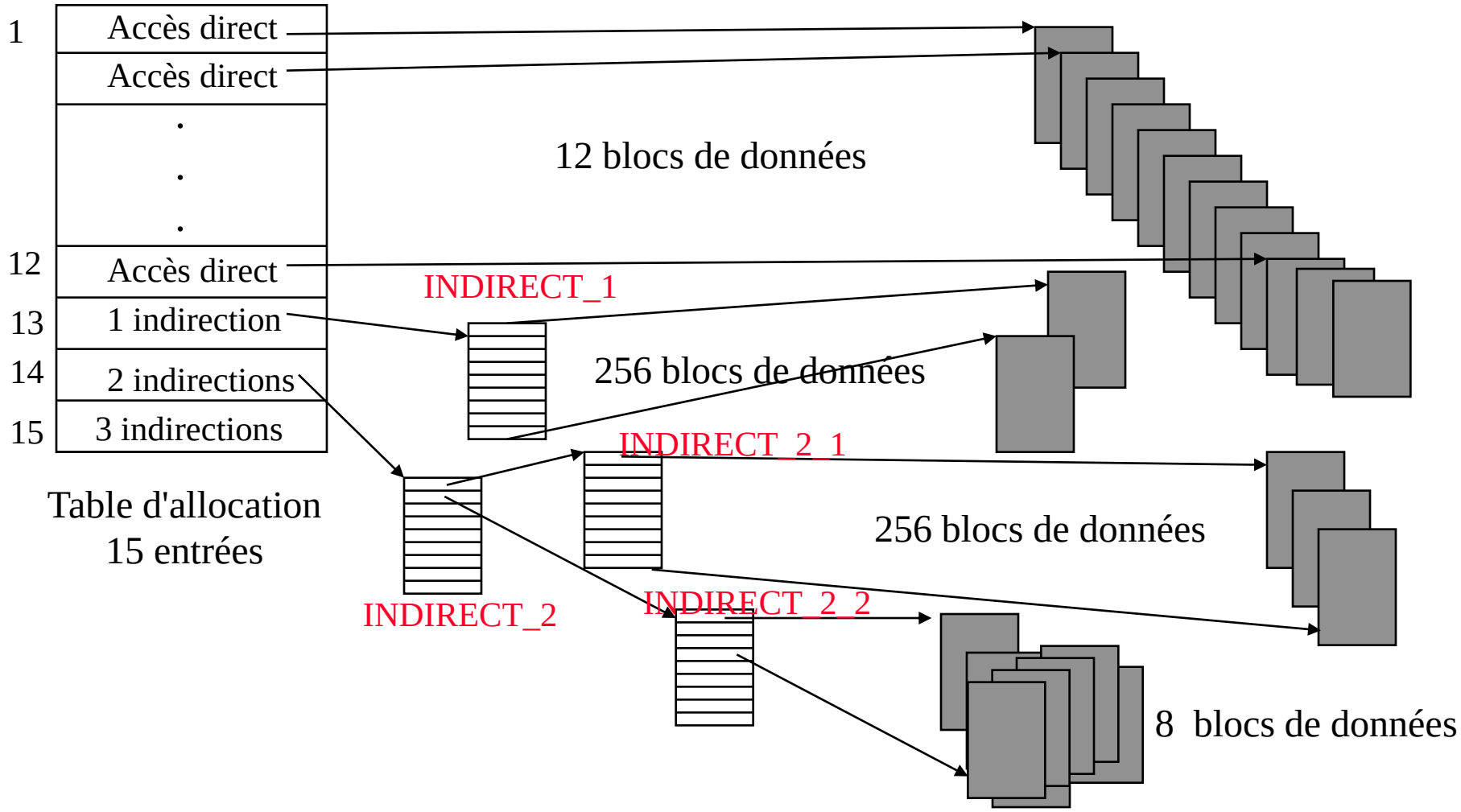
389 ème lecture : accès à bloc 14

Indirect_1 est dans la liste
Bloc 14 n'est pas dans la liste : 1 AD
Pas de tampon libre : le tampon libéré n'est jamais le bloc d'indirection dont on a besoin car il est toujours parmi les plus récemment accédé.

1 accès disque

EXEMPLE

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



Nombre d'accès disque pour lire le fichier :

12 + (1 + 256) + (2 + 256) + (1 + 8) = 539 AD

Temps de lecture = 539 * 10 ms = 5390 ms = 5 s (divisé selon facteur 2)

Ecriture d'un fichier : le buffer cache

write (fd, &un_eleve, sizeof(un_eleve));

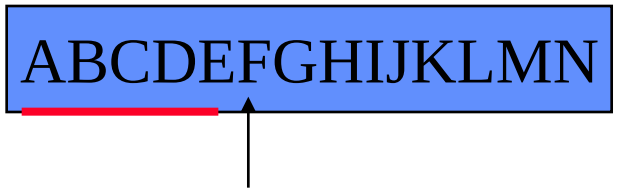
- Les écritures s'effectuent dans les blocs copiés depuis le disque.
- Un tampon modifié n'est recopié que si il est remplacé.
  → Possibilité de perte de données

- Les tampons sont sauvegardés en outre (threads noyau bdflush) :
  - Le nombre de tampons modifiés est trop important;
  - Le tampon est resté modifié en MC trop longtemps;
  - Un processus force la sauvegarde des tampons le concernant : appels systèmes sync(), fsync()…

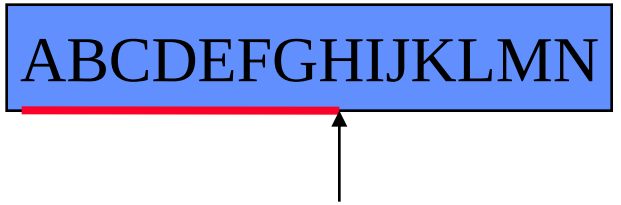
Accès concurrents à un fichier : les verrous d'accès

- Trois processus accèdent à un même fichier fich1

- Proc1 accès en lecture **read (fd, chaine, 5))**

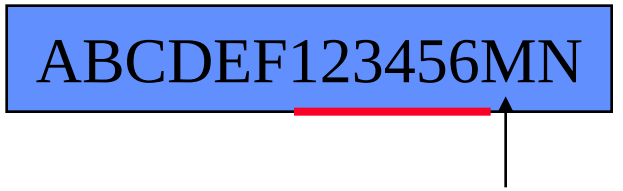


- Proc2 accès en lecture **read (fd, chaine, 8))**

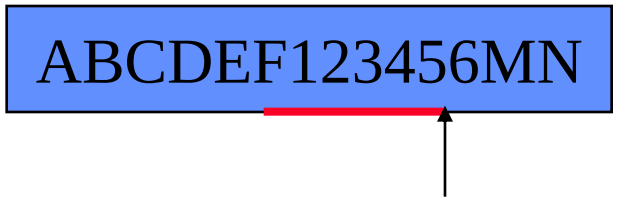


- Proc 3 accès en écriture

```
ret = lseek(fd,6,SEEK_SET);
write (fd, "123456",6);
```



- Proc1 accès en lecture **read (fd, chaine, 5))**



Accès concurrents à un fichier : les verrous d'accès

- Deux types d'opérations / processus
  - Lecture (lecteurs)
  - Ecriture (écrivains/rédacteurs)

- Les règles de synchronisation sont :
  - Une opération d'écriture à la fois OU 1 à n lectures

- Le système Linux met en place des verrous en lecture ou écriture:

Verrou posé	Verrou demandé	
	lecture	écriture
Aucun	oui	oui
lecture	oui	non
écriture	non	non

Outil de synchronisation : le verrou

- Un mécanisme proposé pour permettre de résoudre les concurrences 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.

Outil de synchronisation : le verrou

V_fichier : verrou; -- verrou libre