

Stockage

INF3173 – Principes des systèmes d'exploitation
Automne 2024

Francis Giraldeau
giraldeau.francis@uqam.ca

Université du Québec à Montréal



Agenda

- Introduction
- Fonctions d'un système de fichier
- Technologies de stockage
- Gestion des blocs
- Arborescence et attributs
- Fiabilité
- Sécurité (chiffrement et confidentialité)
- Études de cas, exemples, exercices

Périphériques de stockage



Disque SSD

\$\$\$\$\$\$



Disque magnétique

\$\$\$



Disques optiques

\$\$

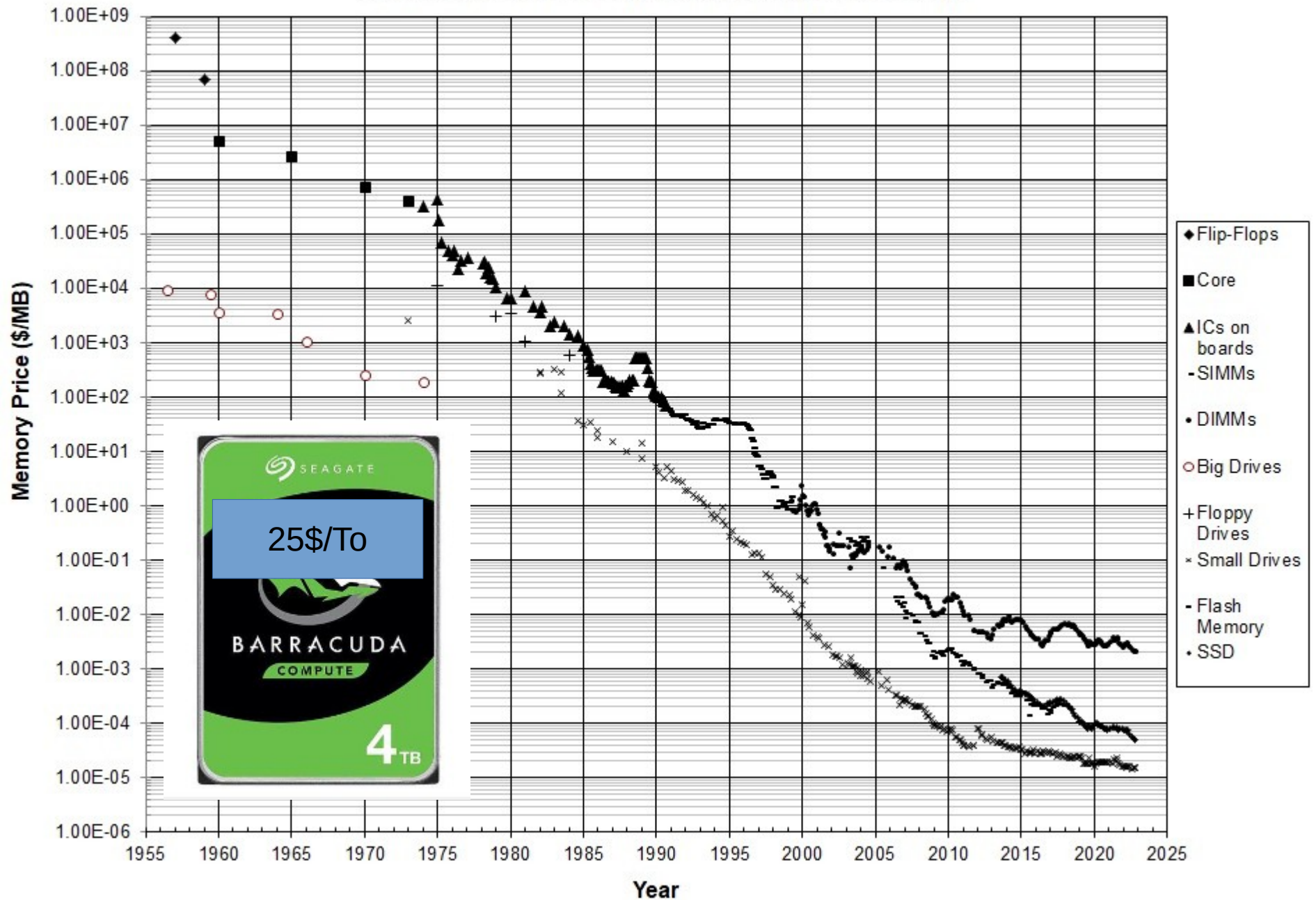


Bande magnétique

\$



Historical Cost of Computer Memory and Storage



Fonctions d'un système de fichier



Système de fichier

tmp	12 éléments	dossier	sam 11 fév 2012 20:45:01 EST
usr	11 éléments	dossier	dim 08 jan 2012 22:25:00 EST
bin	2475 éléments	dossier	ven 10 fév 2012 15:42:26 EST
etc	1 élément	dossier	dim 08 jan 2012 22:25:00 EST
games	9 éléments	dossier	jeu 02 fév 2012 14:44:12 EST
include	281 éléments	dossier	lun 30 jan 2012 14:31:53 EST
lib	1557 éléments	dossier	ven 10 fév 2012 15:42:23 EST
lib32	630 éléments	dossier	jeu 15 déc 2011 20:50:49 EST
local	9 éléments	dossier	mer 12 oct 2011 10:26:57 EDT
bin	35 éléments	dossier	dim 08 jan 2012 22:54:02 EST
AntidoteHD	8,6 Mo	Lien vers exéc	ven 16 oct 2009 22:55:00 EDT
bosstthread	20,0 ko	exécutable	dim 08 jan 2012 22:54:02 EST

- Détermine comment répartir les fichiers en blocs
- Présente une vue hiérarchique de répertoires
- Implémente les fonctions d'accès (read, write, create, delete, etc.)
- Stock les permissions d'accès et d'autres attributs
 - Permet au système d'exploitation d'imposer les permissions, quotas, etc.
- Assure l'intégrité des fichiers
- Fonctions de sécurité: encryption et confidentialité

Monter un système de fichier

Accéder au deuxième disque SCSI (sda, sdb) par le répertoire backup avec le système de fichier ext4.

```
$ mount -t ext4 /dev/sdb /media/backup
```

[...]
0000080 2000 a0fb 7c64 ff3c 0274 c288 bb52 0417
0000090 2780 7403 be06 7d88 17e8 be01 7c05 41b4
00000a0 aabb cd55 5a13 7252 813d 55fb 75aa 8337
00000b0 01e1 3274 c031 4489 4004 4488 89ff 0244
00000c0 04c7 0010 8b66 5c1e 667c 5c89 6608 1e8b
00000d0 7c60 8966 0c5c 44c7 0006 b470 cd42 7213
00000e0 bb05 7000 76eb 08b4 13cd 0d73 c2f6 0f80
00000f0 d084 be00 7d93 82e9 6600 b60f 88c6 ff64
[...]

```
├── dir1
│   ├── dir11
│   │   └── file1
│   ├── dir12
│   ├── file1
│   ├── file2
│   └── file3
├── dir2
│   └── file1
└── dir3
```

Stockage réel

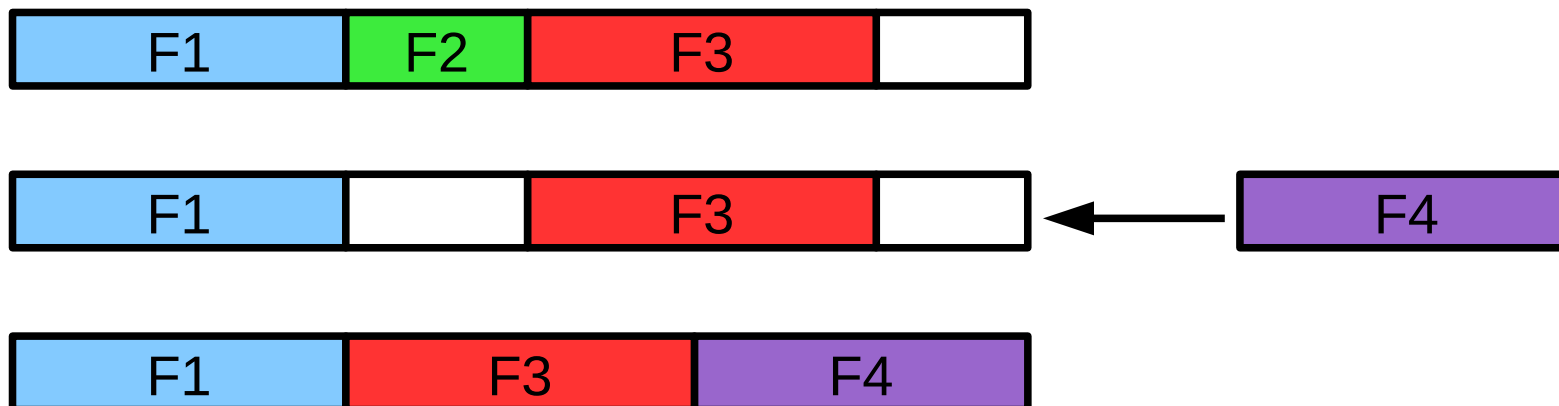
Représentation utilisateur

Allocation continue (1)

*fichier
externe de
elle est*

- Consiste à enregistrer les fichiers en séquence
- Fragmentation de l'espace libre → fragmentation externe
- Même si l'espace libre est suffisant, il devient inutilisable
- Méthode utilisée pour l'écriture unique (ex: cédérom)

Exemple: Soit trois fichiers F1, F2, F3. Lorsque F2 est effacé, il laisse un trou qui n'est pas assez grand pour un nouveau fichier F4. Il faut déplacer F3 (compacter) pour rassembler l'espace libre, une opération coûteuse.



Allocation continue (2)

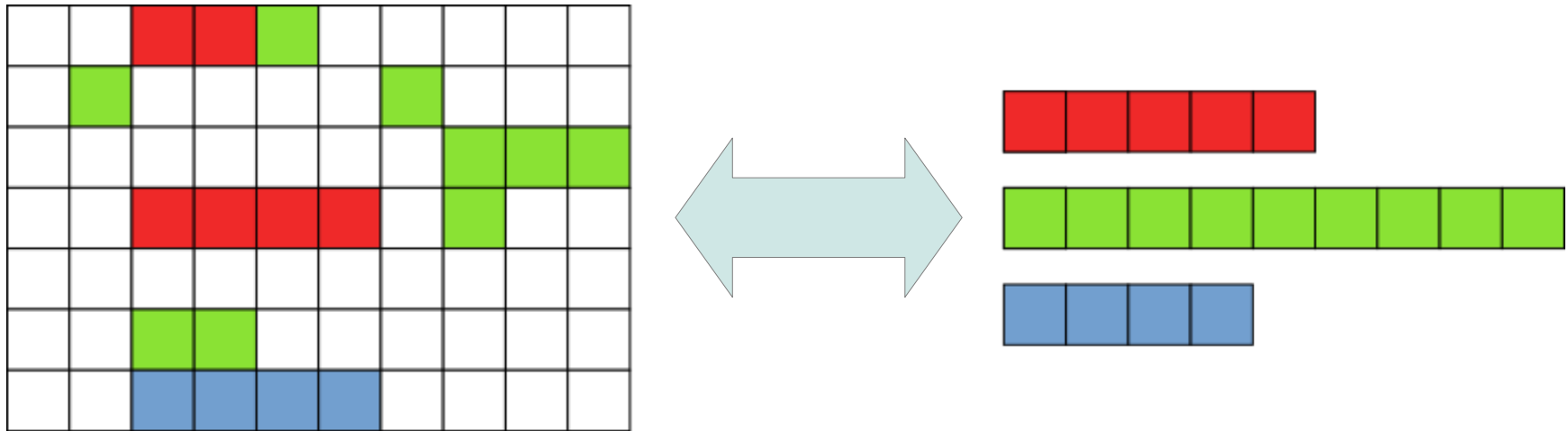


Table d'allocation des blocs

Fichier	Bloc début	Taille
A	2	2
B	5	4
C	10	5

Handwritten signature

Allocation par blocs

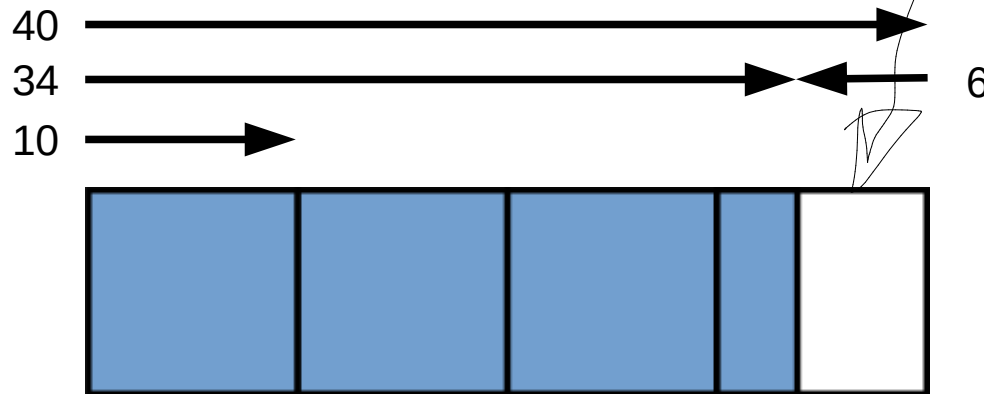


- Le système de fichier abstrait l'emplacement des blocs
- Le fichier apparaît continu
- Agrandir ou rétrécir un fichier se fait par bloc entier
- Méthode généralement utilisée pour l'allocation dynamique (ex: disque dur)

Allocation par blocs : fragmentation interne

- Le dernier bloc n'est pas nécessairement plein → fragmentation interne
- L'espace inutilisé du dernier bloc ne peut pas être utilisé par un autre fichier → perte d'espace

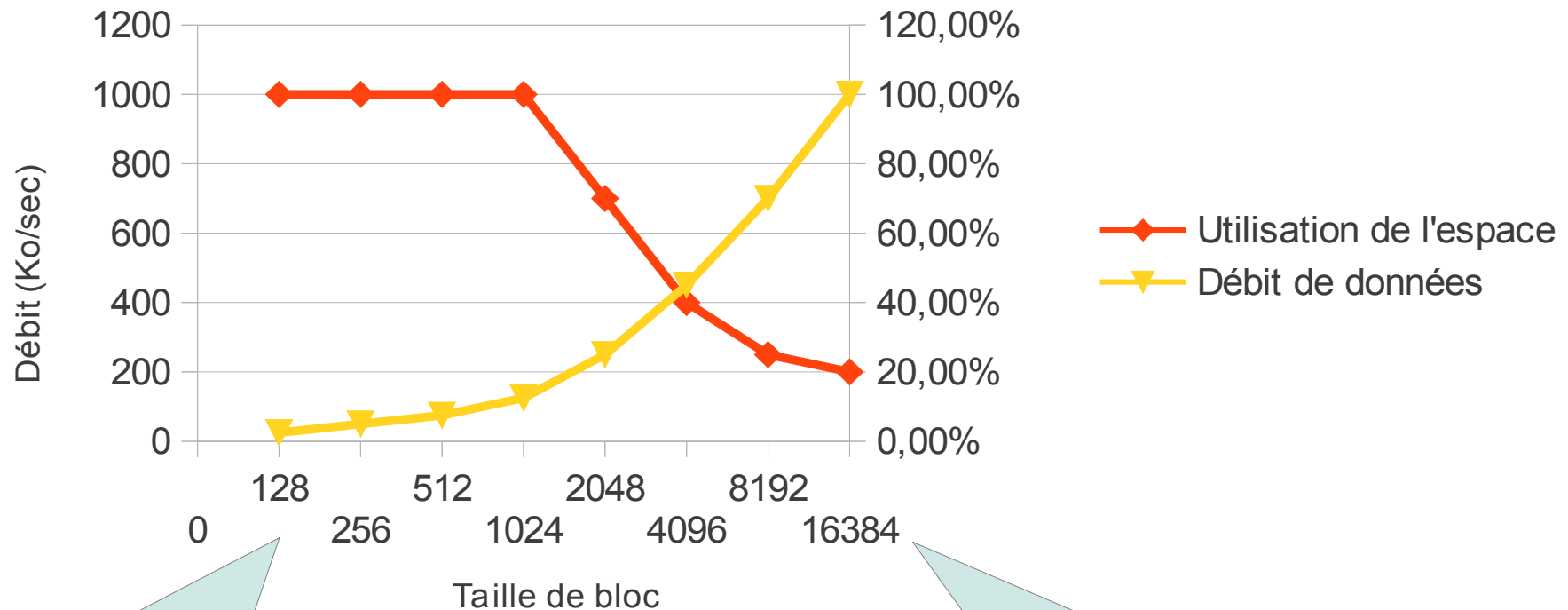
Exemple:



Taille des blocs	10
Blocs alloués	4
Taille totale allouée	40
Taille du fichier	34
Perte	6

Taille des blocs: compromis entre débit et perte

Efficacité et débit selon la taille de bloc



Traiter un grand nombre de petits blocs augmente la surcharge et réduit les performances

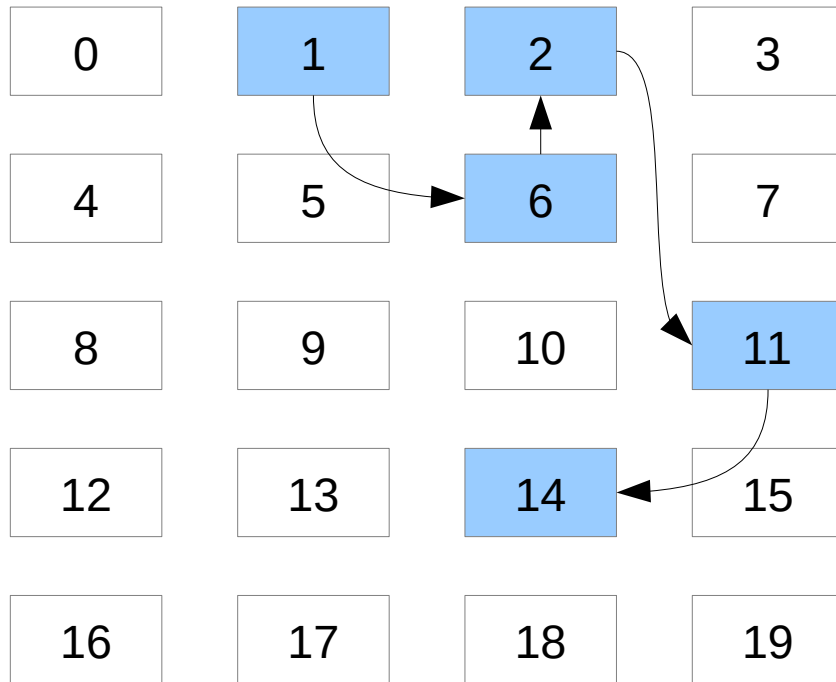
De grands blocs produisent une perte élevée pour stocker un grand nombre de petit fichiers

Source: Modern Operating System second edition, Andrew S. Tanenbaum, p. 412

Exercice 1

- Un certain système de fichier utilise des blocs de 2Kio. Si tous les fichiers ont exactement 1Kio, quelle est la taille perdue par fragmentation interne? 50 %
- Discutez de la perte selon la distribution de la taille des fichiers. Dans quelle condition la perte sera plus grande ou plus petite que celle obtenue précédemment?

Allocation par blocs chaînés

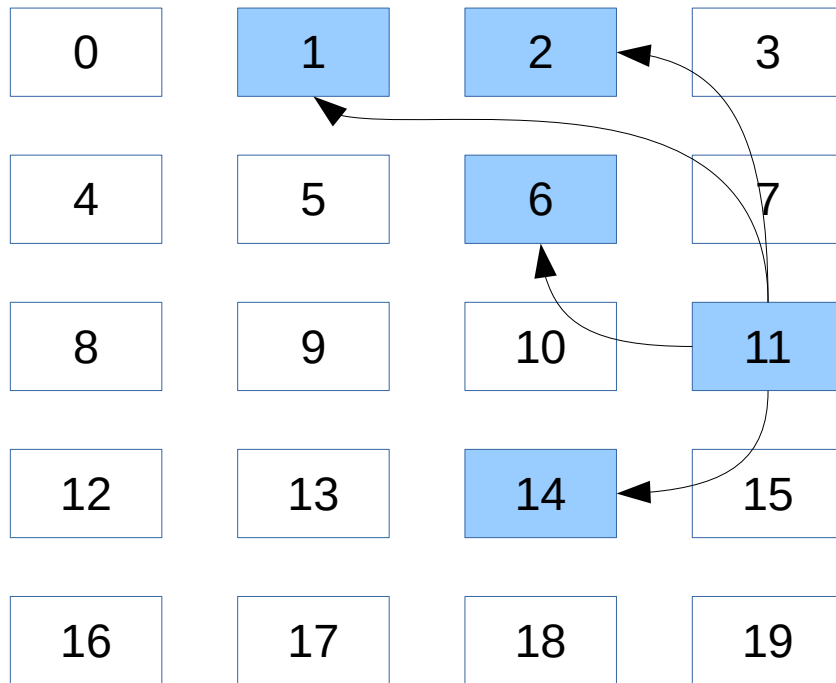


Les blocs n'ont pas besoin d'être contigus.

Le fichier débute au bloc 1 et occupe 5 blocs.

Le déplacement dans le fichier requiert de consulter les blocs depuis le début du fichier.

Allocation par blocs indexés

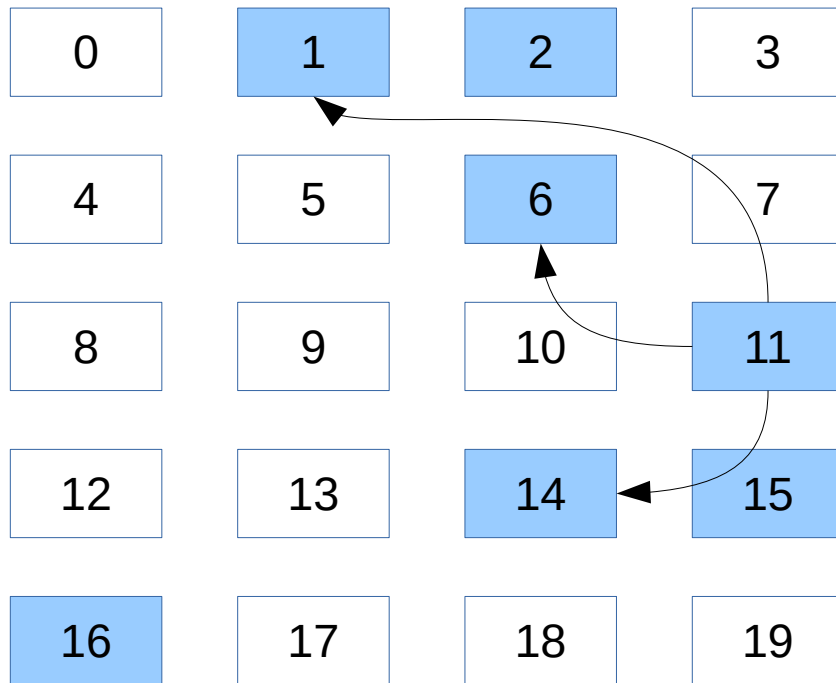


Un bloc d'index conserve les références à tous les blocs de données.

Index
1
2
6
14

Le déplacement dans un fichier nécessite de consulter le bloc d'index uniquement.

Allocation par blocs indexés avec taille variable



Un bloc d'index conserve les références à tous les blocs de données, avec le nombre de blocs contigus. Diminue le nombre de pointeurs requis.

Bloc début	Taille
2	2
6	1
14	3

Inventaire des blocs libres

- Tableau de bit: un bit par bloc indique s'il est occupé ou libre.
- Espace libre chaîné: la fin d'un espace libre pointe vers le prochain emplacement disponible.
- Index avec taille: un bloc possède le début d'un espace libre et sa taille.
- Optimisation: conserver un sommaire des zones libres en mémoire pour des accès rapide.

Contraintes

- Longueur maximale des noms de fichiers
- Caractères autorisés dans les noms de fichiers
- Sensibilité à la case
- Nombre maximal de fichiers total
- Nombre maximal de fichiers par répertoire
- Taille maximale d'un fichier
- Taille maximale du volume

Format du système de fichier FAT

Périphérique

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

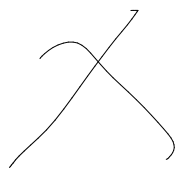
Table des fichiers

Fichier	Index
A	18
B	9
C	2

Table d'allocation
des blocs

0	
1	nul
2	11
3	nul
4	
5	
6	1
7	
8	nul
9	8
10	
11	14
12	
13	
14	6
15	
16	
17	
18	19
19	nul

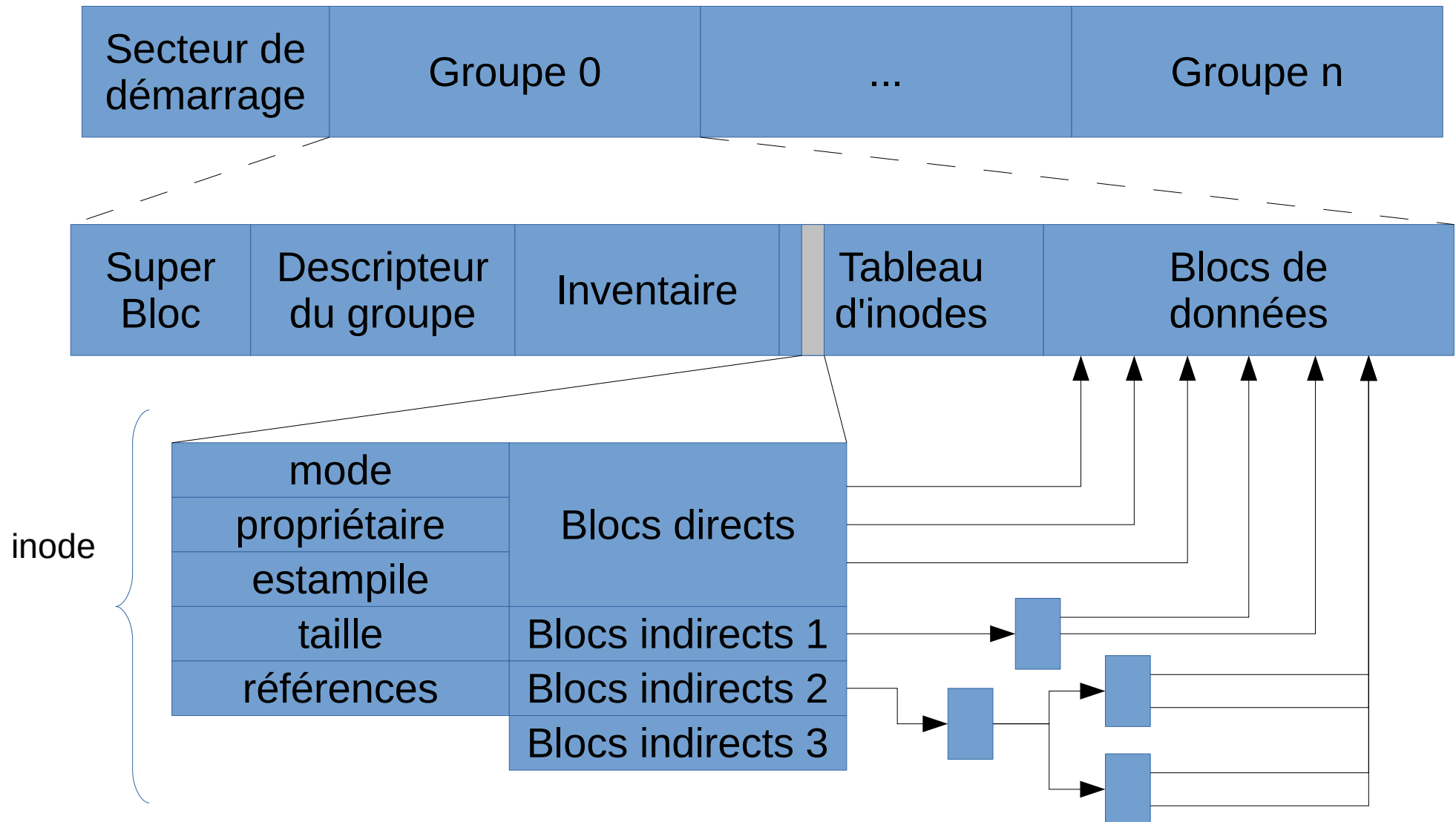
- La table est allouée à l'avance, peu importe le nombre de blocs utilisés.
- Temps d'accès moyen de $O(n)$ comme une liste chaînée.
- Taille de la table proportionnelle au nombre de blocs.
- Exemple: déterminer les blocs pour un fichier:
 - A: 18, 19
 - B: 9, 8
 - C: 2, 11, 14, 6, 1



Exemple File Allocation Table

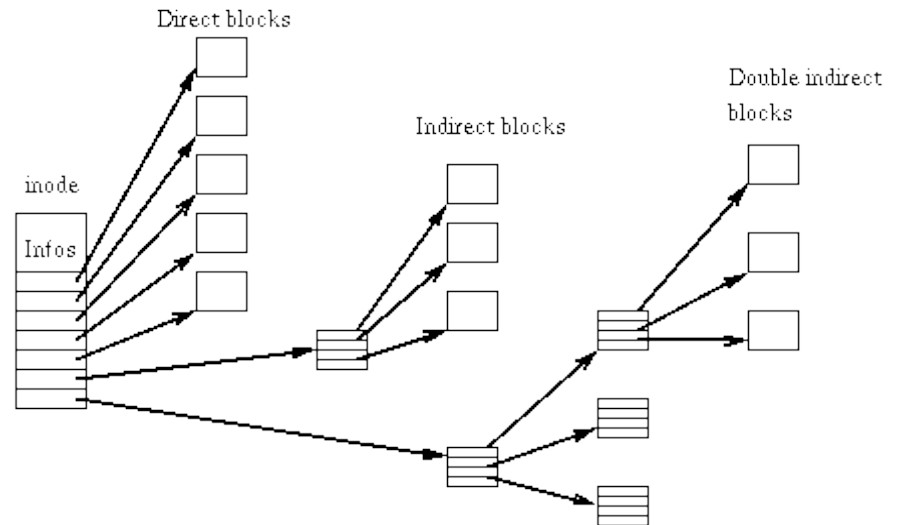
- FAT12
 - 12 bits pour l'index de bloc = $2^{12} = 4096$ blocs maximum
 - Taille maximale du volume avec blocs de 4Kio = 16Mio
 - De grands blocs signifie plus de perte de fragmentation interne
 - Taille de la table d'allocation : $(2^{12} * 12)/8 = 6144$ octets
 - Table d'allocation au début du disque
 - Noms de fichiers limités à 11 caractères
 - La table sert aussi de répertoire des blocs libres
- FAT16 augmentent ces limites en utilisant des index de 16 bits.
- FAT32 encore courant sur les clés USB et les petits systèmes de fichiers.

Format d'un système de fichier UNIX



Blocs indirects

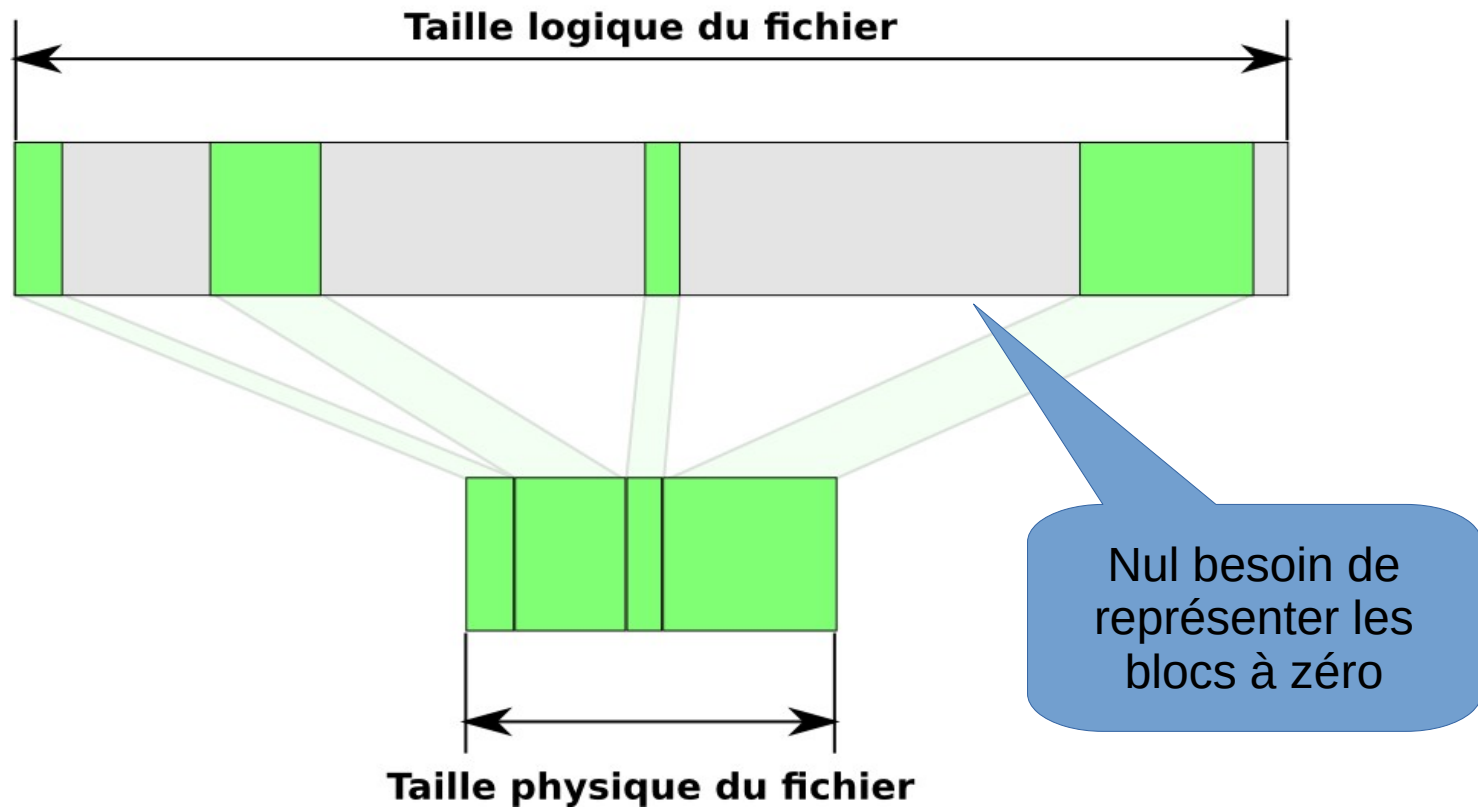
- Pointeurs directs utilisés pour les petits fichiers.
- Niveaux d'indirections supplémentaires utilisés lorsque la taille l'exige.
- Le nombre de pointeurs utilisés augmente selon la taille allouée.
- Déplacement $O(1)$ pour les petits fichiers, $O(\log(n))$ pour les grands fichiers.



Taille bloc	512
Taille pointeur	4

Type	Nombre de blocs
Direct	12
Indirect 1	128
Indirect 2	16384
Indirect 3	2097152
Total	2113676

Fichier éparsé



Exemple de fichiers éparses

```
$ dd if=/dev/null of=sparse-file bs=1 seek=1M
0+0 enregistrements lus
0+0 enregistrements écrits
0 octet (0 B) copié, 1,4644e-05 s, 0,0 kB/s
```

```
$ echo "foo" >> sparse-file
```

```
$ hexdump -C sparse-file
```

```
00000000  00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00  |.....|
*
00100000  66 6f 6f 0a                                |foo.|
00100004
```

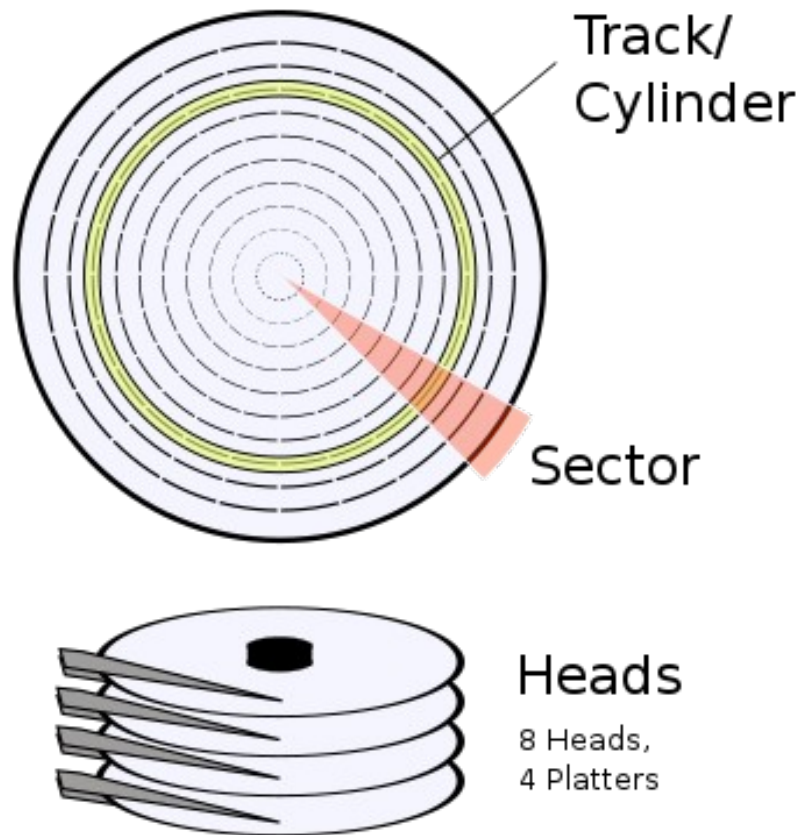
```
$ ls -lskh sparse-file
```

```
4,0K -rw-rw-r-- 1 francis francis 1,1M 2012-02-17 13:24 sparse-file
```

$0x100000 = 1048576 = 2^{20}$

Taille physique = 4Kio (1 bloc)
Taille logique = 1,1 Mio

Disque magnétique rotatif



- Vitesse de rotation
 - 7 à 15k tours par minutes
- Vitesse de positionnement de la tête de lecture
- But: minimiser le déplacement de la tête de lecture

Algorithme FIFO

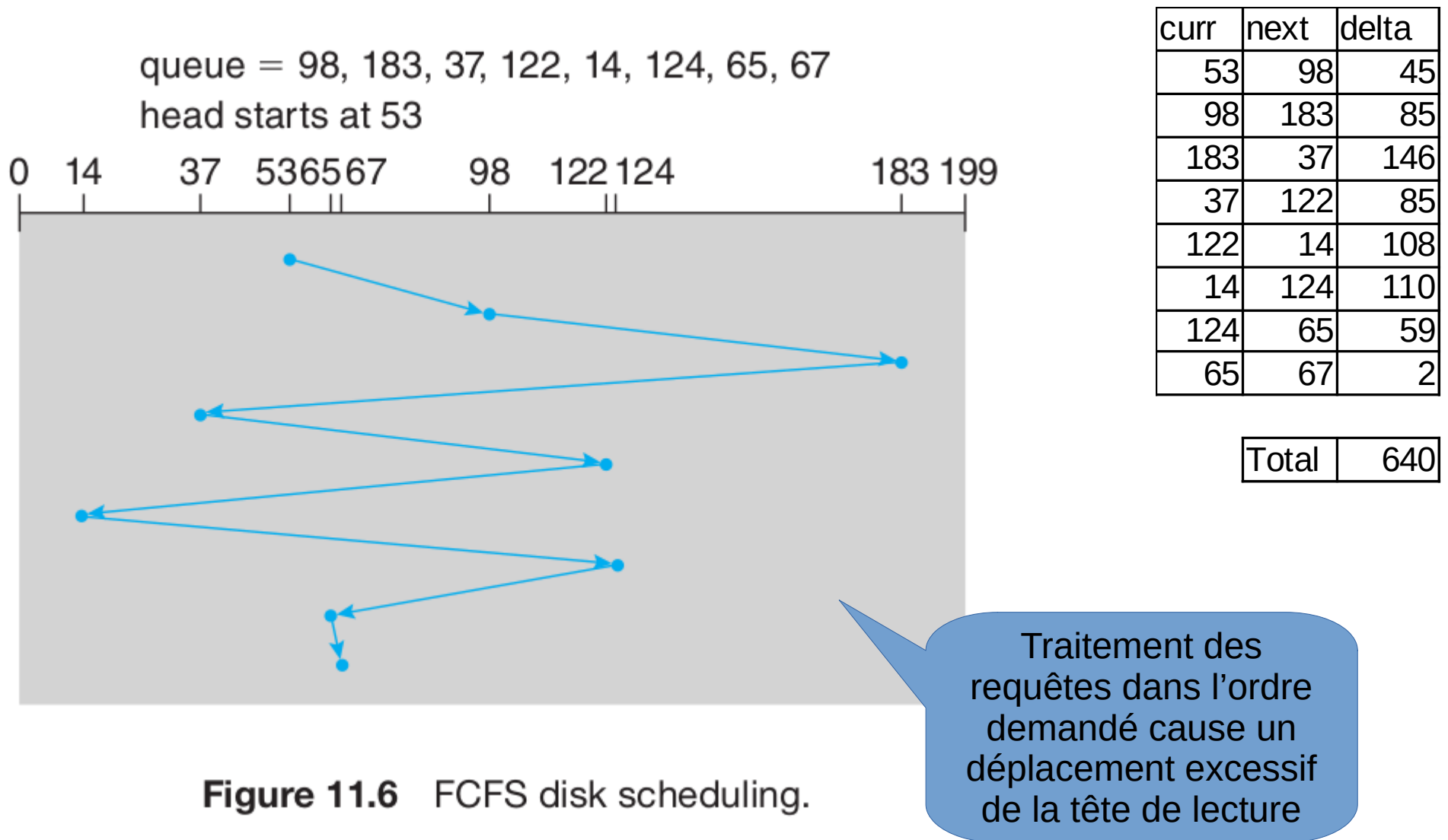


Figure 11.6 FCFS disk scheduling.

Source: Operating System Concepts 10th edition, p. 459

Algorithme de l'ascenseur

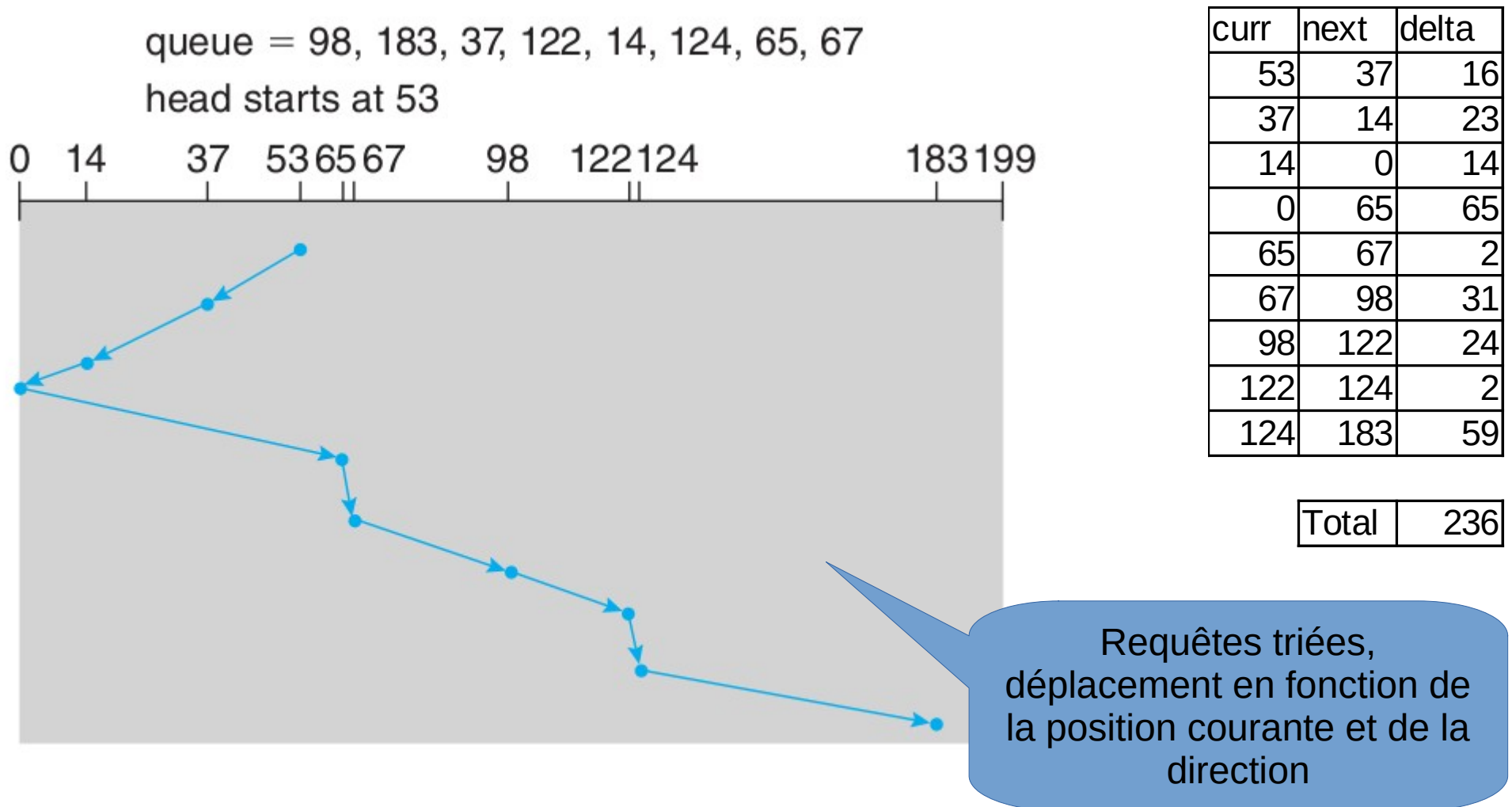


Figure 11.7 SCAN disk scheduling.

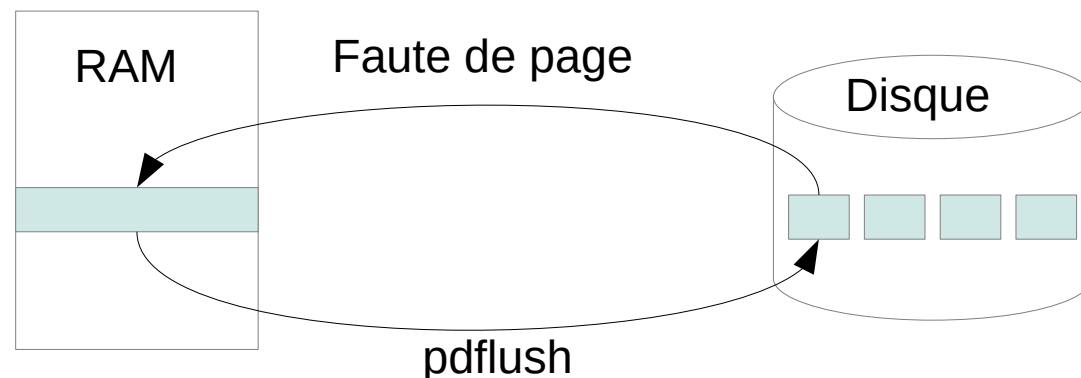
Source: Operating System Concepts 10th edition, p. 459

Ordonnancement des E/S du disque

- Principe de localité → probabilité élevée d'accéder à des blocs successifs
- Il vaut la peine de retarder un peu les requêtes au disque et de les accumuler, il peut être possible de les grouper pour maximiser le débit et minimiser la latence globale.
- Terme utilisé: plug (bouchon)
- L'ordonnanceur des E/S du disque (elevator) permet de faire cette optimisation.
 - Noop : base comparative, simple queue FIFO
 - Completely Fair Scheduler (cfq) : meilleur dans la plupart des cas (trois files d'attentes: temps-réel, dans la mesure du possible, si rien d'autre ne doit être fait)
 - Deadline : garanti une attente maximale
 - Anticipatory : délai élevé pour grouper un maximum de requête sur des disques très lents
- Référence:
- <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/block/switching-sched.rst>

Miroir en mémoire (mmap)

- Un miroir en mémoire permet d'accéder un fichier comme s'il était chargé dans l'espace mémoire du processus.
- Lors de l'accès, une faute de page cause le chargement des blocs en mémoire.
- Si la page est écrite, elle est marquée comme modifiée et un démon en arrière plan synchronisera cette page sur le disque à interval régulier (pdflush: page daemon flush).
- Appel système sync(): force à ex.cuter toutes les écritures en attentes.



Écritures en arrière-plan (asynchrone)

- write() copie en mémoire et retourne tout de suite
- L'écriture sur disque se fait en arrière-plan, éventuellement
- Avantage: réduire le délai d'écriture perçu par l'utilisateur
- Désavantage: données perdues si une panne survient pendant que les données sont seulement en mémoire

```
$ dd if=/dev/zero of=temp.dat oflag=sync bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes (1,0 GB, 1000 MiB) copied, 2,47472 s, 424 MB/s
```

```
$ dd if=/dev/zero of=temp.dat bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes (1,0 GB, 1000 MiB) copied, 0,163178 s, 6,4 GB/s
```

Différence entre écriture synchrone et asynchrone

Exemple linux/fs/ext4/file.c

```
const struct file_operations ext4_file_operations = {  
    .llseek      = ext4_llseek,  
    .read        = do_sync_read,  
    .write       = do_sync_write,  
    .aio_read    = generic_file_aio_read,  
    .aio_write   = ext4_file_write,  
    .unlocked_ioctl = ext4_ioctl,  
    .mmap        = ext4_file_mmap,  
    .open        = ext4_file_open,  
    .release     = ext4_release_file,  
    .fsync       = ext4_sync_file,  
    .splice_read = generic_file_splice_read,  
    .splice_write = generic_file_splice_write,  
    .fallocate   = ext4_fallocate,  
};
```

aio : E/S asynchrone

mmap: miroir en mémoire
(memory map)

splice: transfert entre
tube et fichier sans copie

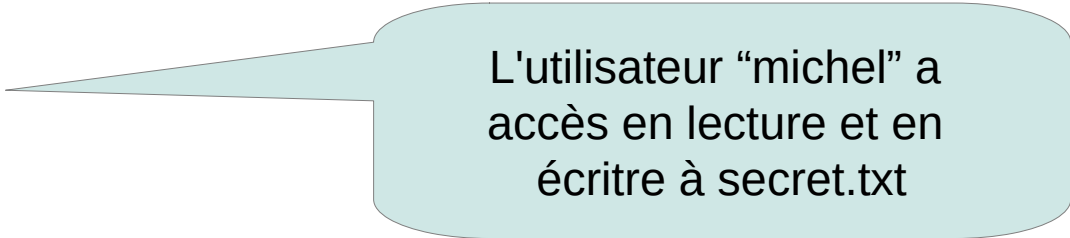
```
const struct inode_operations ext4_file_inode_operations = {  
    .truncate      = ext4_truncate,  
    .setattr       = ext4_setattr,  
    .getattr       = ext4_getattr,  
    .setxattr      = generic_setxattr,  
    .getxattr      = generic_getxattr,  
    .listxattr     = ext4_listxattr,  
    .removexattr   = generic_removexattr,  
    .check_acl     = ext4_check_acl,  
    .fiemap        = ext4_fiemap,  
};
```

Liste de contrôle d'accès

- Permissions UNIX de base limitées à un utilisateur et un groupe
- La liste de contrôle d'accès → gestion fine des droits d'accès par utilisateur et par groupe
- Les permissions suivent la même sémantique:

rwX

```
$ setfacl -m user:michel:rw secret.txt
$ getfacl secret.txt
# file: secret.txt
# owner: francis
# group: francis
user::rw-
user:michel:rw-
group::rw-
mask::rw-
other::r--
```



L'utilisateur "michel" a accès en lecture et en écriture à secret.txt

Fiabilité: journalisation

- En cas de panne de courant, il se peut qu'une opération sur le disque soit terminée partiellement et cause une corruption.
- Solution: écrire les modifications dans un journal, puis exécuter sur la copie maitresse.
- Les modifications apportées avec succès sont retirée du journal. Il est plus rapide de ne réviser que les dernières opérations plutôt que toute la partition!

Confidentialité: chiffrement

- Algorithmes de chiffrement par blocs symétriques
 - Ex: AES, DES, Blowfish
 - Blocs chiffrés individuellement avec une clé aléatoire
 - Clé symétrique chiffrée avec une clé publique, protégée par mot de passe
 - Seul la clé privée permet de déchiffrer la clé symétrique d'un bloc
- Chiffrement à bas niveaux du disque
 - Si la clé est trouvée, tout le disque peut être déchiffré
- Chiffrement d'un répertoire
 - Chiffrement des blocs de données et des noms de fichiers
 - Chaque répertoire peut-être chiffré récursivement avec sa propre clé
 - Permet un chiffrement par utilisateur

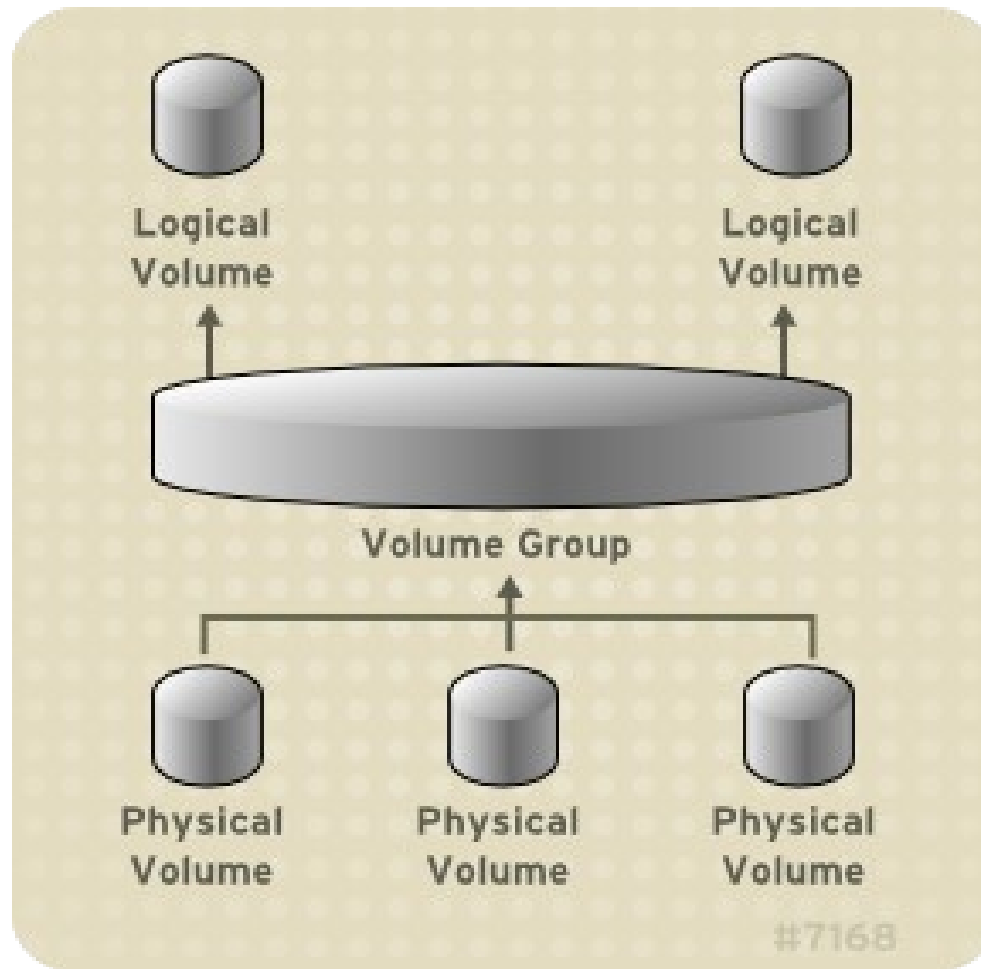
Confidentialité: effacement sécuritaire

- L'effacement d'un fichier déréférence les blocs utilisés, mais les blocs de données peuvent encore être lisibles!
- Pour effacer des données confidentielles, il faut initialiser les blocs libérés.
- Utilitaire “shred” ou “wipe” écrase plusieurs fois le contenu d'un fichier avec des octets aléatoires. Le contenu devient irrécupérable.

Virtualisation du stockage

- Quoi faire lorsqu'un disque est plein?
- Solution 1: ajouter un disque, répartir les fichiers sur les deux disques. Cette solution divise l'espace disponible et complexifie la gestion de grand volume de données.
- Solution 2: fusionner les blocs des disques comme s'ils n'étaient qu'un seul disque. Ceci évite la partition de l'espace libre et facilite la gestion.
- Logical Volume Manager (LVM) permet de fusionner plusieurs disques physiques et de rediviser ensuite l'espace en volumes logiques.

Logical Volume Manager



Source:

http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Logical_Volume_Manager_Administration/index.html

Fiabilité par la redondance

- Copie des fichiers (aka backup)
 - Facile: première ligne de défense contre la perte de données!
 - Copie logique: archive sur le même disque
 - Copie physique: périphérique distinct
- Redundant Array of Independent Disks (RAID)
 - Un ou plusieurs disques servent de contrôle de parité. Si un disque fait défaut, son contenu peut-être reconstruit à partir des disques restants.

Exemple RAID 5: ratio de redondance de $1/5$, $n \geq 3$ disques.

Il faut une panne de 2 disques pour une panne du système.

Mean Time Between Failure (MTBF) d'un disque: 1,4Mh [1]

Taux de panne par heure $r = 1 / \text{MTBF} = 7,14\text{E-}7$

Taux de panne du RAID: $n \cdot (n - 1) \cdot r^2 = 3,1\text{E-}12$

Soit un taux de panne par heure environ 200 000 fois plus faible.

[1] WD VelociRaptor SATA Hard Drives data sheet

Quotas

- Partage équitable de l'espace de stockage
- Tenir à jour la taille des fichiers de chaque utilisateur
- Limite souple: l'utilisateur obtient un avertissement
- Limite dure: toute allocation d'espace est refusée

Notifications

- Comment détecter qu'un fichier a été accédé ou modifié?
- Solution 1: faire une base de données avec les temps d'accès de tous les fichiers et balayer régulièrement les fichiers pour détecter un changement. Hautement inefficace!
- Solution 2: enregistrer des gestionnaires d'événements lors d'actions sur des fichiers. Cette solution permet de surveiller un grand nombre de fichier et de répertoire avec une faible surcharge.
- Réalisé sous Linux avec **inotify**

Système de fichier distribué

- Accès aux fichiers sur un disque distant comme s'il était local.
- Les opérations sur les fichiers se font par réseau.
- Exemples: NFS (Unix), CIFS (Windows)