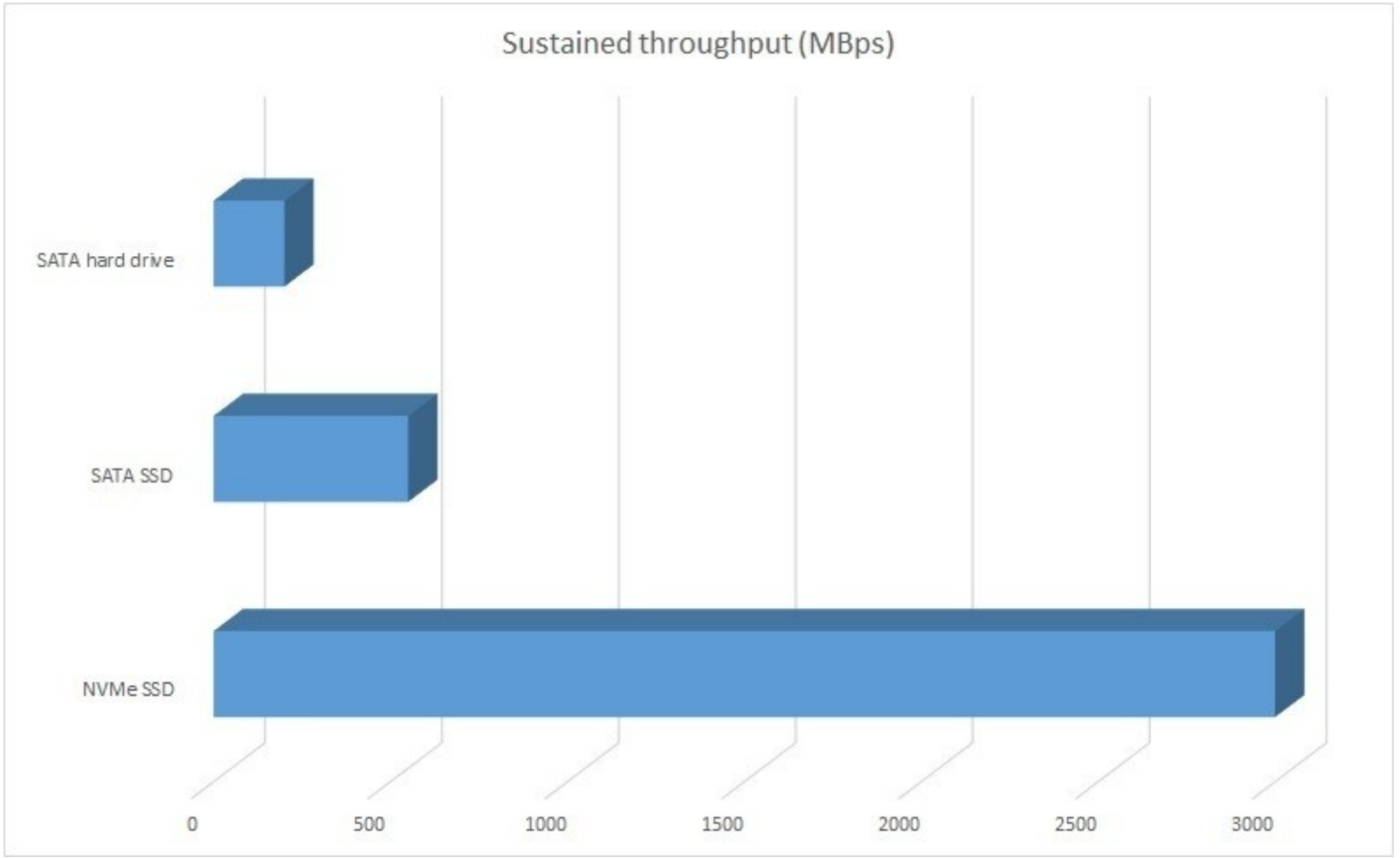
Systèmes de fichiers

HDD : Hard Disk Drive
SSD : Solid-State Drive
NVME : Non-Volatile Memory express)
...

Évolution du coût du stockage sur disque dur

Aujourd'hui # 0,01$/Go


Sustained throughput (MBps)
SATA hard drive
SATA SSD
NVMe SSD
0
500
1000
1500
2000
2500
3000

> 7000

Les données : point de vue de l'utilisateur

- **Persistance :**

  les données persistent entre les appels de programme, les coupures d'alimentation, les plantages. (Nécessite un stockage non volatile).

- **Vitesse :**

  obtenir les données rapidement.

- **Taille :**

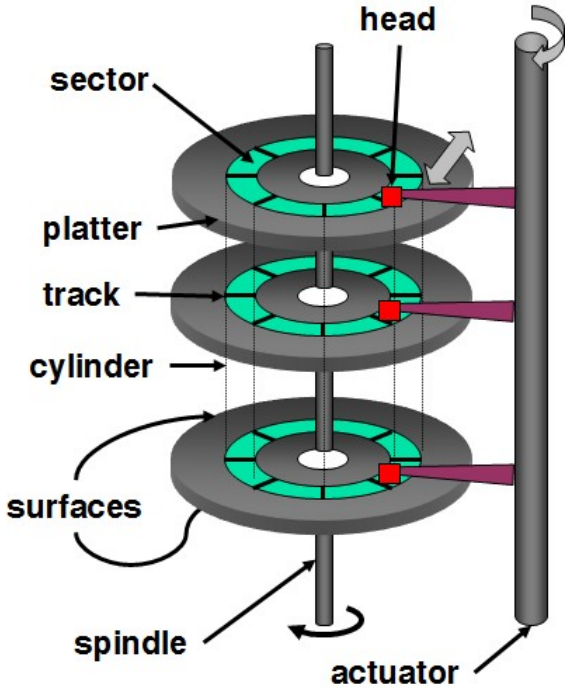
  autant que nécessaire !

- **Partage / Protection :** Partager le cas échéant ou rester privé si nécessaire (droits, verrous)

- **Facilité d'utilisation :** l'utilisateur peut facilement trouver, examiner, modifier des données, etc.

Un système de fichier ? Pour quoi faire ?

- Premier support de stockage civilisé : le disque dur !
  - Un empilement de disques (plateaux)
  - Sur chaque plateau : pistes (regroupées en cylindres)
  - Sur chaque piste : des secteurs
  - Un moteur, un bras de lecture



- 1 bloc de donnés identifié par
  - Un numéro de cylindre
  - Un numéro de tête
  - Un numéro de secteur

- Adressage simplifié et uniforme
  - 1 bloc est identifié par
    - Un numéro de bloc !!!
    - Translation physique faite par le disque

Un système de fichiers ?

- Un fichier est défini par un format
- Le format dépend de l'application (image, texte, etc.)
- Pour le système de fichiers, tous les fichiers sont équivalents → séquence d'octets d'une certaine taille
- Enregistré sur un périphérique de type bloc
- Un bloc est composé d'octet contigü
  - **L'accès par bloc est plus performant pour des disques durs**
- Un système de fichier abstrait le stockage des fichiers :

  représentation logique/physique

- Le fichier est l'**unité logique de données** sur un périphérique de stockage.

  = abstraction logique uniforme pour le stockage physique des informations
- Représenté comme un espace d'adressage logique contigü
- Types de fichiers:
  - Données
    - numérique
    - texte
    - binaire
  - Programme, documents, images
- Contenu défini par le créateur du fichier

Types de fichiers - extensions

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Les extensions ne garantissent pas le contenu !

Modèles d'accès au fichier

- Modèles d'accès aux fichiers du **point de vue du programmeu**r:
  - **Séquentiel**: données traitées dans l'ordre, un octet ou un enregistrement à la fois.
    - La plupart des programmes utilisent cette méthode.
    - **Exemple**: compilateur lisant un fichier source.
  - **Direct**: adresse un bloc en fonction d'une valeur clé.
    - **Exemple**: recherche de base de données, table de hachage, dictionnaire
- Modèles d'accès aux fichiers du **point de vue du système d'exploitation:**
  - **Séquentiel**: garde un pointeur sur l'octet suivant du fichier.
  - **Direct**: adresse tout bloc dans le fichier directement en fonction d'un décalage dans le fichier.
    - Aussi connu sous le nom d'**accès aléatoire (random access)**.

- Le système de fichiers se compose de
  - Une collection de fichiers
  - Une structure de répertoires
  - (éventuellement) Partitions
- Aspects importants
  - Protection des fichiers
  - Partage de fichiers

Métadata

- **Nom :** forme lisible par l'homme !
- **Identifiant :** numéro unique pour chaque fichier du système de fichiers
- **Type :** nécessaire pour les systèmes prenant en charge différents types
- **Emplacement :** pointeur vers l'emplacement du fichier sur l'appareil
- **Taille :** taille du fichier
- **Protection :** contrôle qui peut lire, écrire, exécuter
- **Heure, date et identification de l'utilisateur** : données pour la protection, la sécurité et la surveillance de l'utilisation, dernier accès, dernière modification, date de création.
- Les informations sur les fichiers sont conservées dans la structure de répertoires, qui est conservée sur le disque
- De nombreuses variantes, y compris des attributs de fichier étendus, somme de contrôle (checksum), ACL

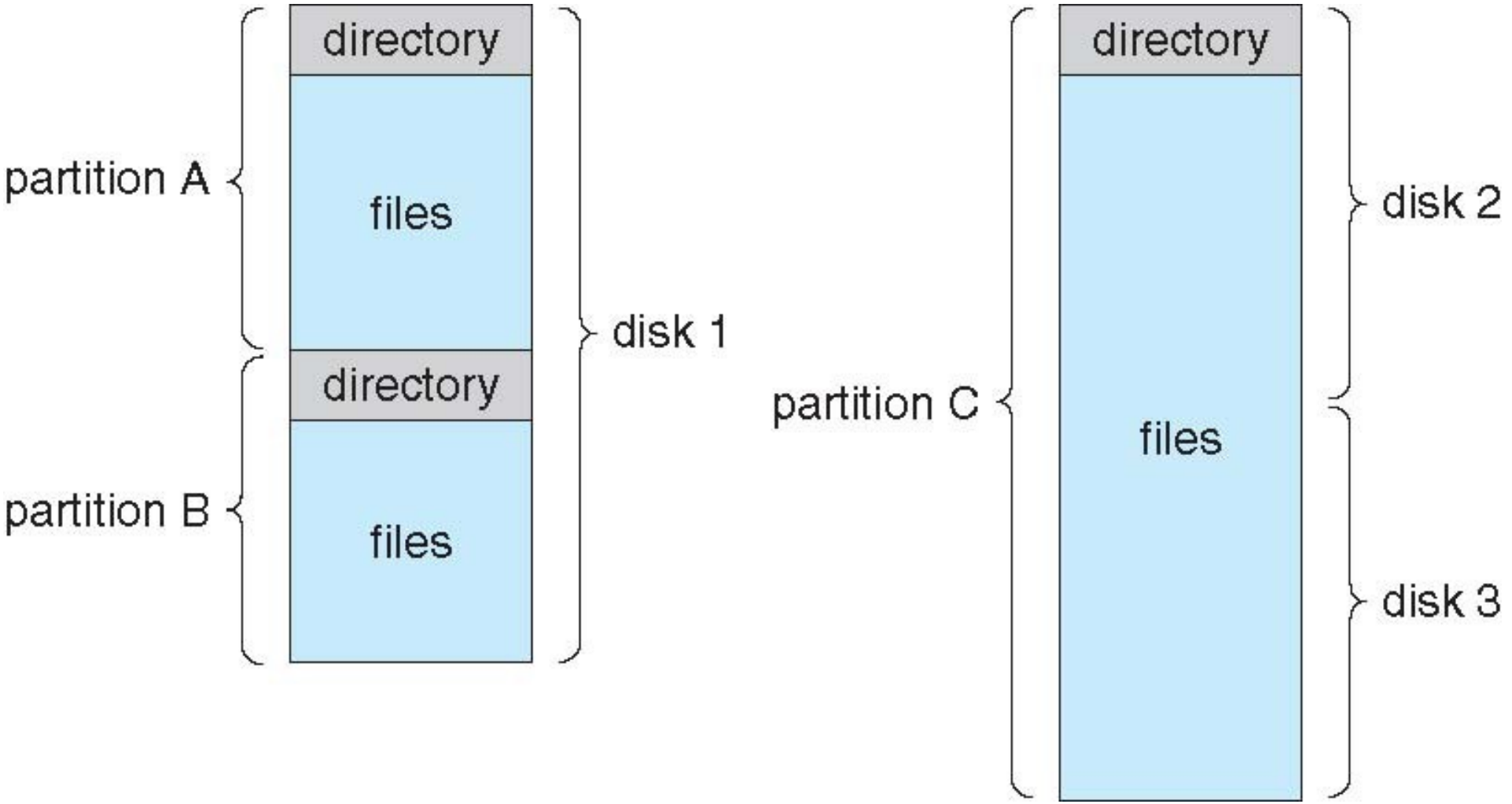
Contraintes selon le FS

- 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

Structure des disques

- Le disque peut être divisé en **partitions (divisions logiques)**
- Les disques ou partitions peuvent être protégés par **RAID** contre les pannes
  - cf plus loin
- Partitions également appelées “minidisk, slices”
- L’entité contenant un système de fichiers est appelé **volume** (peut être contenu sur une partition mais aussi sur plusieurs disque)
- Outre les systèmes de fichiers à usage général, il existe de nombreux systèmes de fichiers à usage spécial, souvent tous dans le même système d'exploitation ou ordinateur. (tmpfs, devfs...)

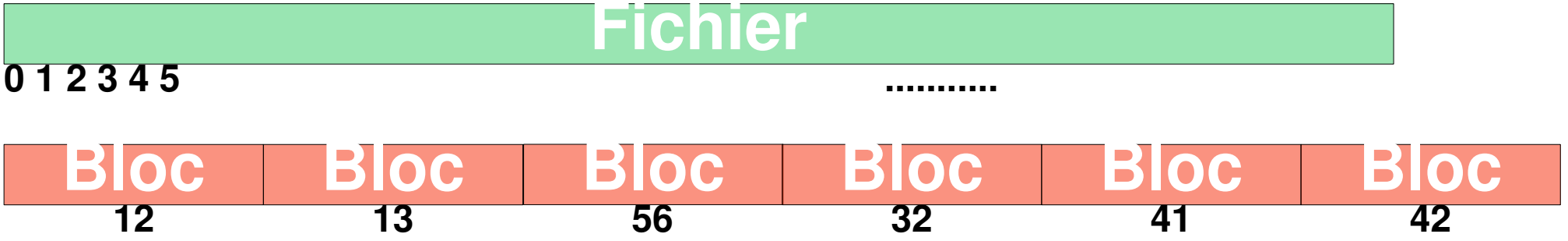
Organisation typique



Un Système de fichier : pour quoi faire ???

- Ce que veut stocker l'utilisateur :
  - Un **ensemble d'octets de taille quelconque**
- Ce que propose les disques
  - Un ensemble de petits **blocs de taille fixe**
- **Abstraction de stockage** proposée : **le fichier**
  - Une séquence d'octets de **taille** quelconque
  - Identifié par un **nom**

- **Problème : faire correspondre l'abstraction à la réalité matérielle**



Rôle du système de fichiers

- Offrir l'abstraction simple de « fichier »
- Masquer les supports de stockage
  - Complexité d'organisation et d'accès au disque
  - Diversité des tailles, organisation physique, etc...
- Gérer au mieux l'utilisation du support de stockage
  - Éviter le gaspillage d'espace disque
  - Eviter les conflits (un octet appartient à un seul fichier!)
  - Optimiser les accès au disque (limiter les déplacements du bras de lecture dans le cas des HDD)

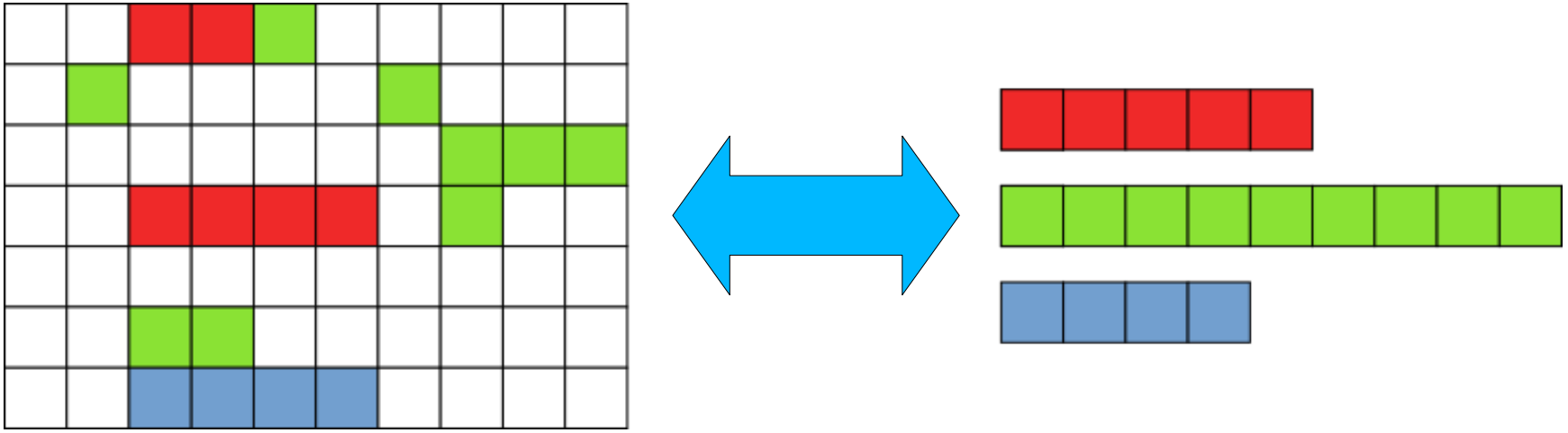
Fonctions d'un système de fichier

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

Méthodes d’allocations

- Une méthode d’allocation permet de définir comment les blocs sont alloués sur le disque pour constituer des fichiers
- Passer de la vision « logique » du fichier au physique (HDD...)

L’allocation est faite par blocs

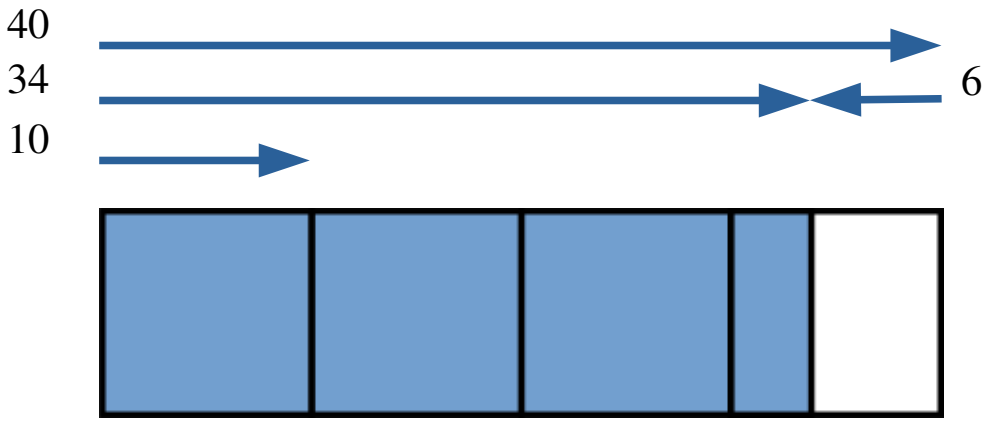


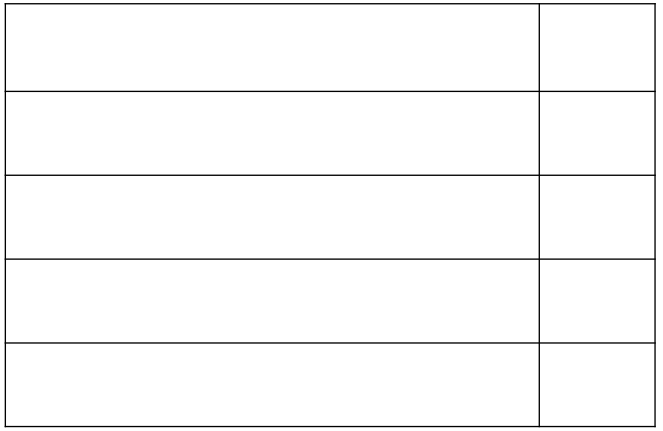
- Le système de fichier abstrait l'emplacement des blocs
- Le fichier apparait continu
- Agrandir ou rétrécir un fichier **se fait par bloc entier**
- Méthode généralement utilisée pour l'allocation dynamique (ex: disque dur)

Allocation par blocs : fragmentation interne

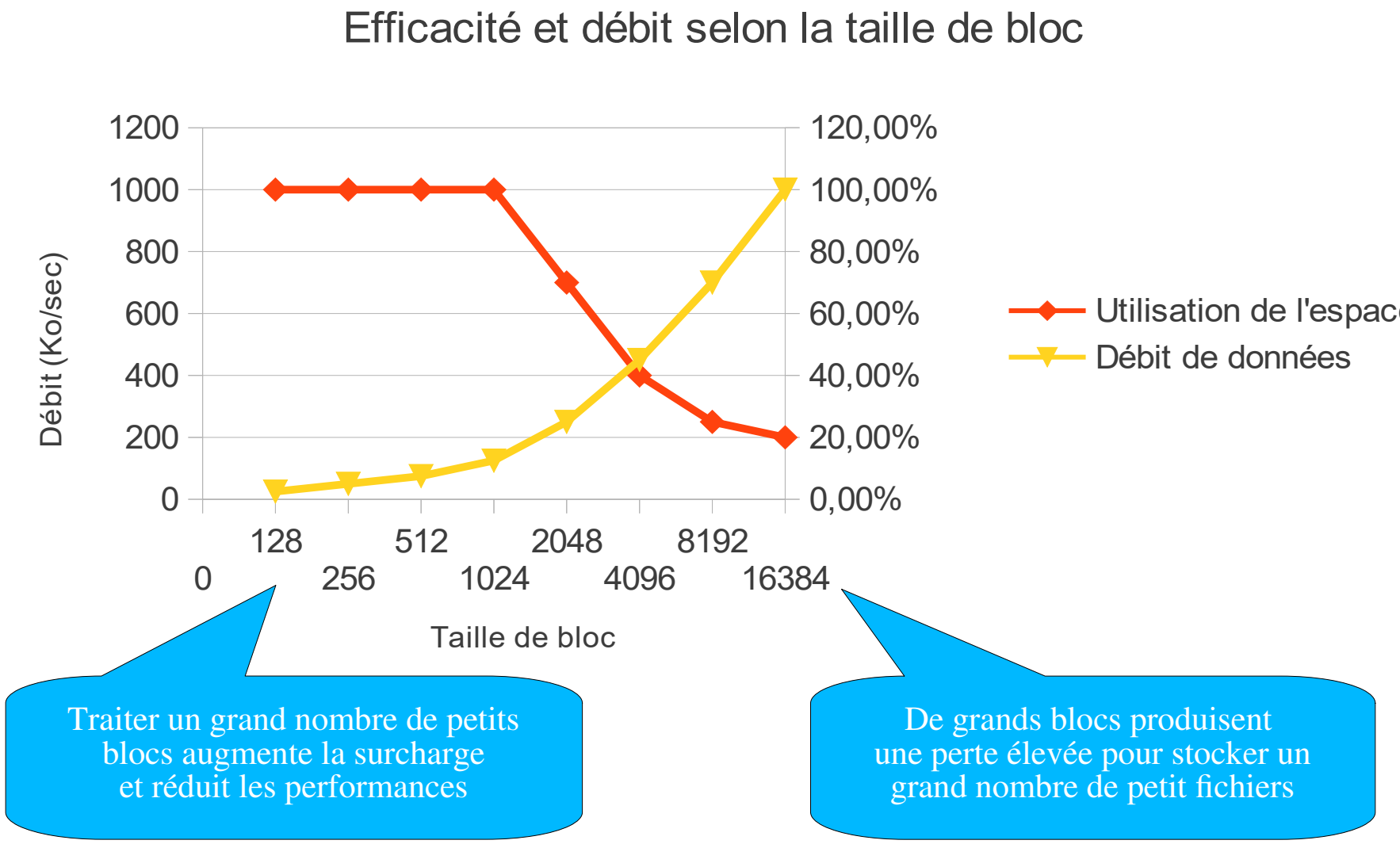
- Le dernier bloc n'est pas nécessairement plein → **fragmentation interne**
- L'espace inutilisé du dernier bloc ne peut pas être utilisé par un autre fichier → perte d'espace

Exemple:





Taille des blocs: compromis entre débit et perte



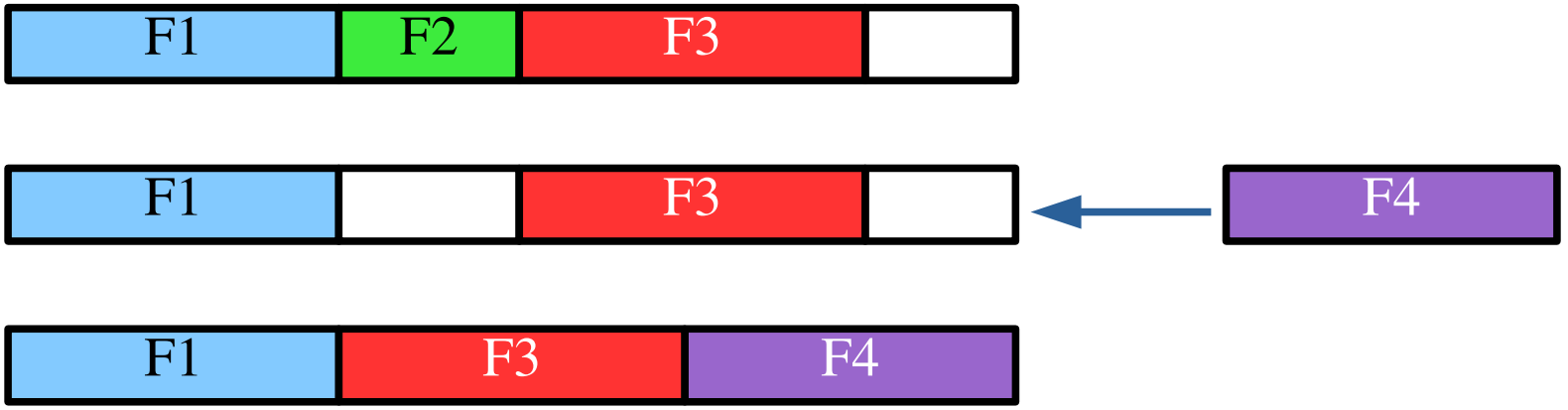
Source: Modern Operating System second edition, Andrew S. Tanenbaum, p. 412

Allocation continue

- Consiste à enregistrer les fichiers sur des **blocs contigüs**
- Simple : seulement besoin du **numéro de bloc de départ** et **du nombre de blocs**
- **Rapide** : le plus efficace, accès aux blocs immédiat
- Mais…

Allocation continue

Exemple: 3 fichiers F1, F2, F3.
- F2 est effacé : il laisse un trou qui n'est pas assez grand
- pour un nouveau fichier F4.
- Il faut déplacer F3 (compacter = défragmenter) pour rassembler l'espace libre, coût important !



- Fragmentation de l'espace libre : **fragmentation externe**
- Même si l'espace libre est suffisant, il devient inutilisable
- Méthode utilisée pour l'écriture unique (ex: CDROM)

Problème :
- trouver de l'**espace libre adapté** pour un nouveau fichier
- d'autant plus que la taille d'un fichier n'est pas connue lors de sa création…
- Difficultés à **faire grandir le fichier**

Il faut **défragmenter** pour éviter la perte de place :

- hors ligne (décidé par l'utilisateur ou le système)
- en ligne (système de fichier)

Allocation continue

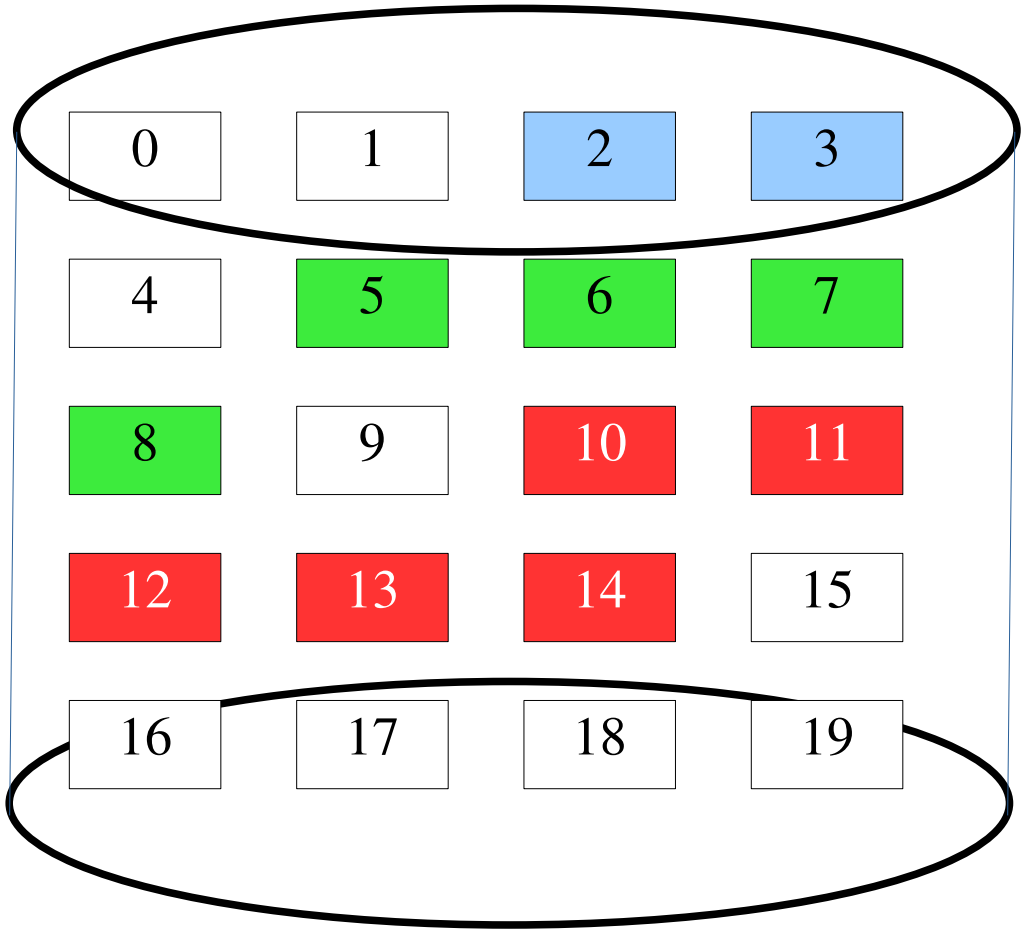


Table d'allocation des blocs

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

Système par “**extents**” (*domaines*)

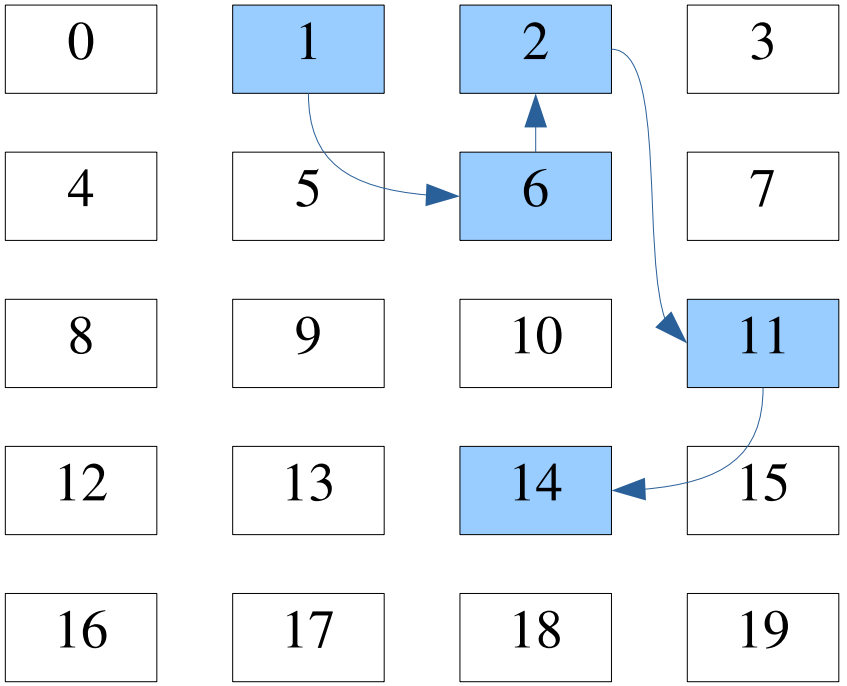
- Comme l’accès le plus rapide se fait avec des blocs contigüs on essaie d’utiliser ce type de blocs en évitant les problèmes !
- Les blocs sont alloués par paquets de blocs contigüs : les “**extents**”
- Un fichier est constitué d’un ou plusieurs extent

- **Principe** :
  - Initialement un extent est alloué au fichier
  - Si ce n'est pas suffisant un autre extent est ajouté (chainé à l'extent précédent dans un bloc) = indirect extent
  - Efficace pour les gros fichiers
  - Préserve certains disques
- **Problèmes** :
  - Fragmentation interne si les extents sont trop grands
  - Fragmentation externe par désallocation des extents.

Allocation par blocs chainés

- Chaque fichier est une **liste chainée de blocs**
- Le fichier est terminé par un pointeur null
- **Pas de fragmentation externe**
- Chaque bloc contient un **pointeur vers le bloc suivant**
- **Gestionnaire de l'espace des blocs libres** appelé quand on ajoute un bloc.
- On peut grouper les blocs pour augmenter la vitesse mais fragmentation interne
- Potentiellement des **problèmes de fiabilité**
- Rechercher un bloc peut être long

Allocation par blocs chainés

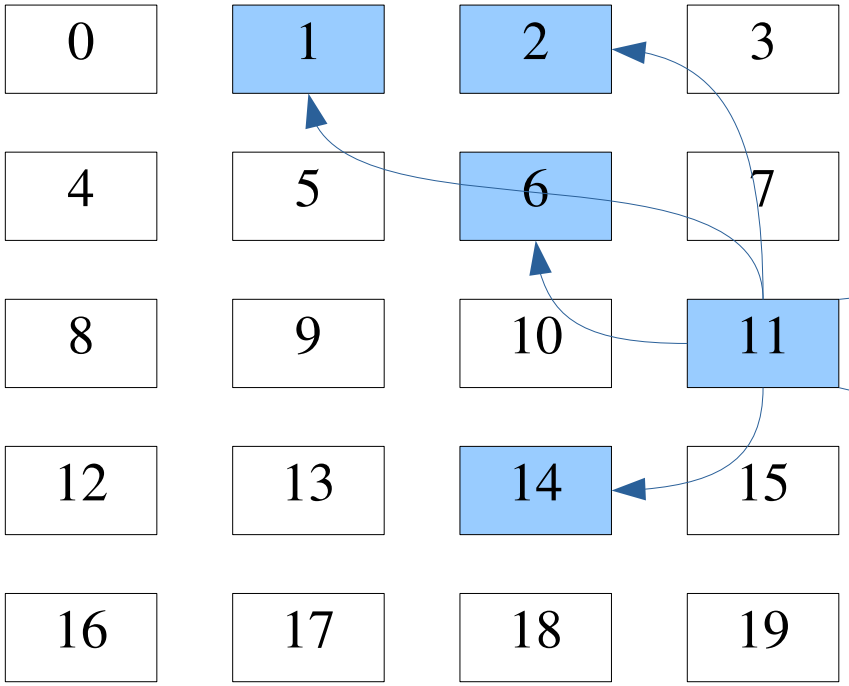


Les blocs n'ont pas besoin d'être contigüs.

Ici 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 référence à tous les blocs de données.

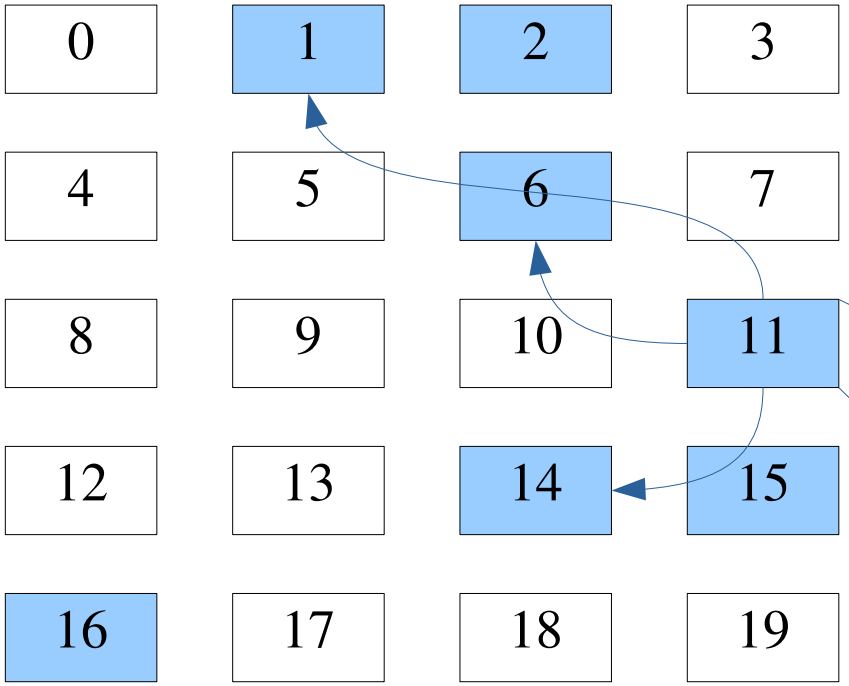
Index
1
2
6
14

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

Chaque fichier possède sa table d'index.

- Besoin de tables d’index : surcoût
- Accès “aléatoire” (random access) facile

Allocation par blocs indexés avec taille variable



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

Bloc début	Taille
1	2
6	1
14	3

Des exemples pratiques de FS...

- File Allocation Table (FAT)
- Ext2,3,4 (Linux)

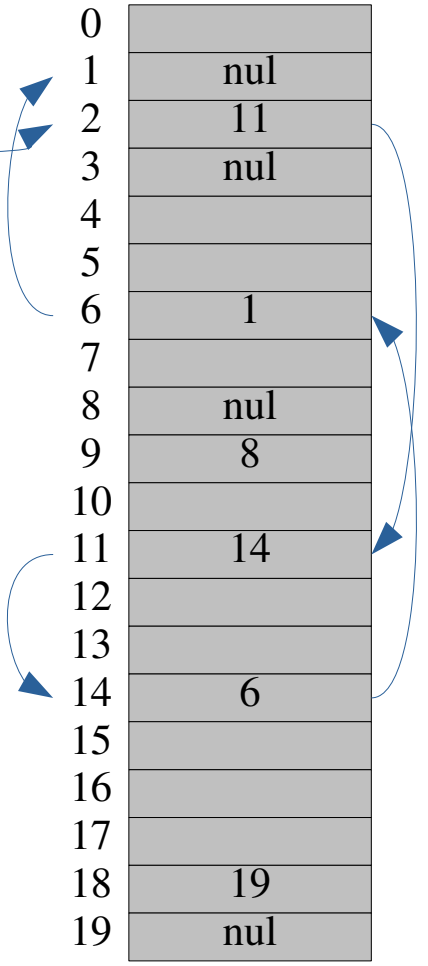
Format du système de fichier FAT (blocs chainés)

Périphérique

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

Fichier	Index
A	18
B	9
C	2

Table d'allocation des blocs



- La table est allouée à l'avance, peu importe le nombre de blocs utilisés.
- Temps d'accès moyen de O(n) comme une liste chainée.
- Taille de la table proportionnelle au nombre de blocs.

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.

- FAT 32 :
  - 4 bits réservés donc 2^{28}x512 = 128 Gio
- ExFAT :
  - 128 Pio

1024^4 Tio tebibyte

1024^5 Pio pebibyte

1024^6 Eio exbibyte

- Remarque :
  - 1 Ko = 1000 octets et 1 Kio = 1024 octets

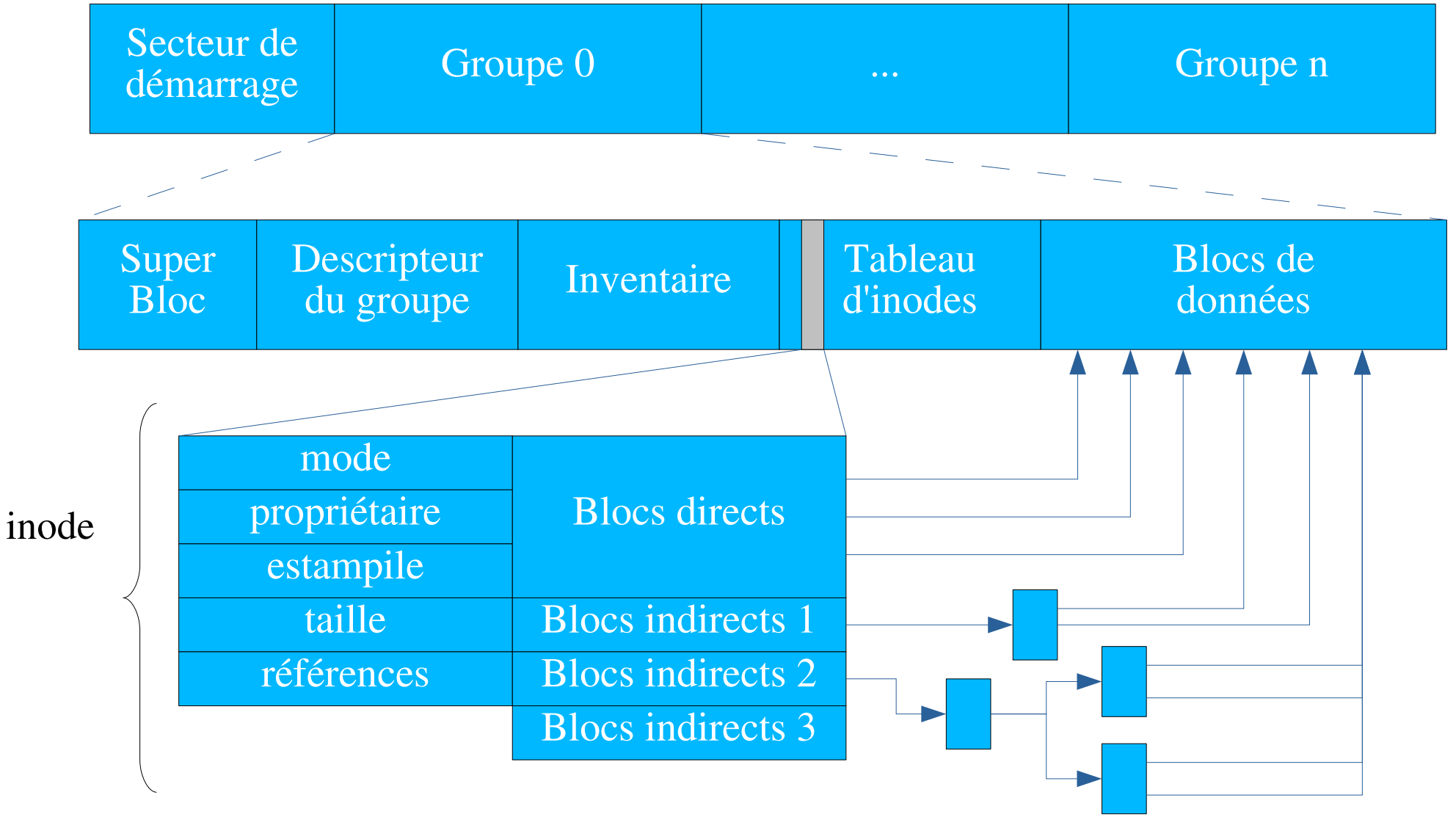
Système de fichier standard linux

- Ext2 puis ext3, ext4 (extent, defragmentation automatique...standard actuel)

Représentation d'un fichier sur disque

- Un fichier est défini par
  - Des données
  - Des méta-données (taille, propriétaire, date de création, ...)
- Sur disque
  - Des blocs de données
  - Des blocs de méta-données
    - Information sur le fichier (taille, propriétaire, ...) : les « **i-noeuds** » (**i-nodes**)
    - Localisation des blocs de données sur disque : **table d'implantation**

Format d'un système de fichier UNIX



Le FS est divisé en groupes de blocs (reduction de la fragmentation, le système essaie de garder les blocs d'un fichier dans le même groupe)

Le FS contient différentes tables système toutes stockées sur un ou plusieurs blocs consécutifs :

- Le super-bloc qui contient les données générales (magic number, taille, montage, ...)

- le descripteur de blocs contient :

  - l'@ du bloc de départ du bloc

  - le nombre de blocs, inode libres, le nombre d'entrées dans le répertoire

- l'inventaire permet de connaître les blocs et les inodes libres :

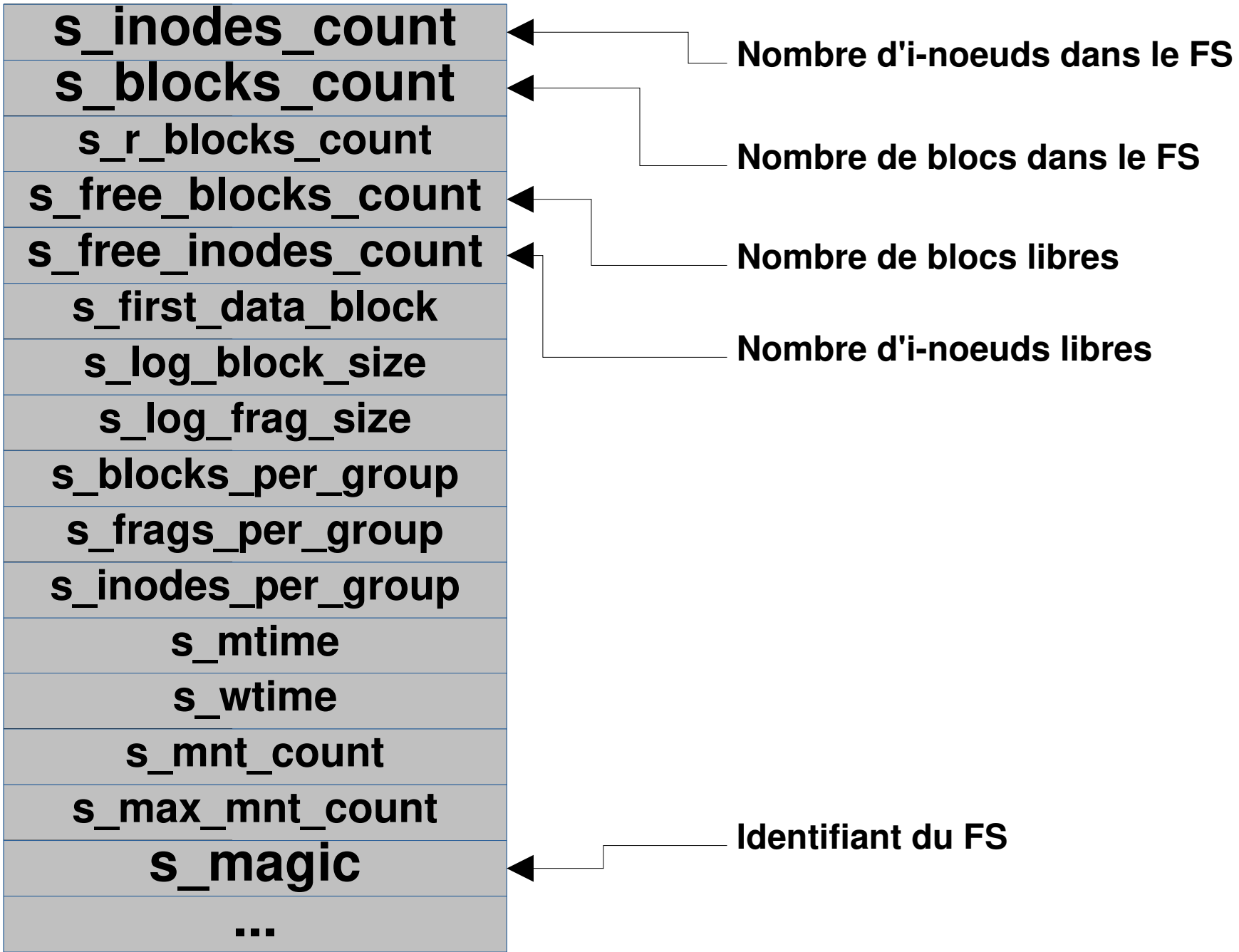
  Deux bitmaps (blocks bitmap, inode bitmaps)

  - La table des inodes qui contient la table de description et d'allocation des fichiers, chaque fichier étant représenté par un numéro d'inoeuds (inode).

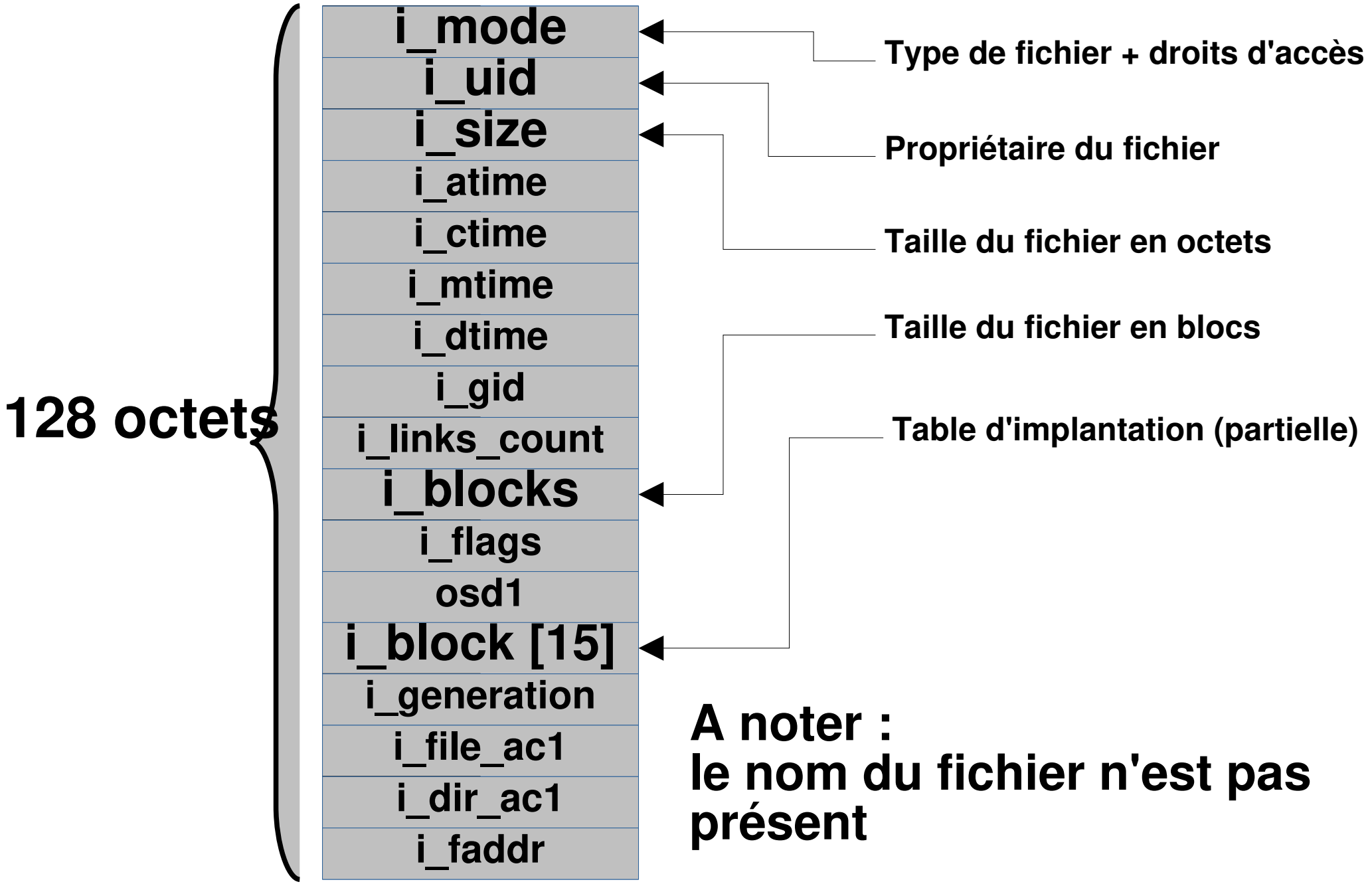
  - Un répertoire est une table de correspondance de fichiers/numéro

  d'inoeud.

Super Bloc : l'exemple d'Ext2

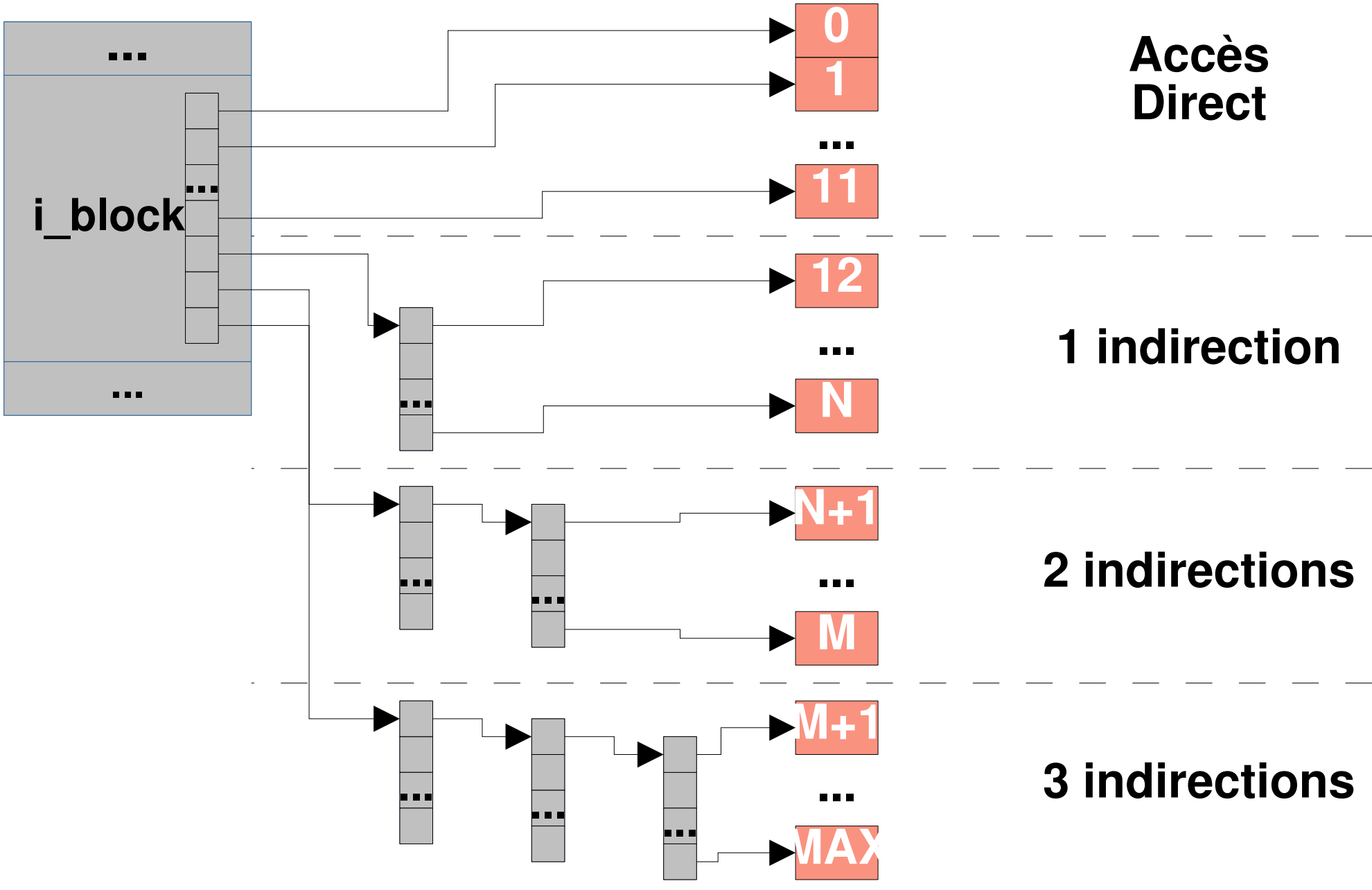


Les i-noeuds : l'exemple d'Ext2



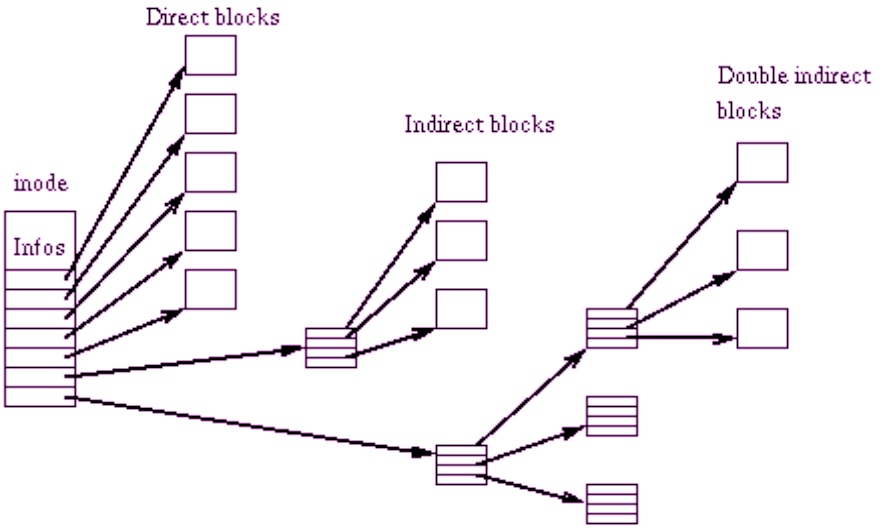
A noter :
le nom du fichier n'est pas présent

Table d'implantation : l'exemple d'Ext2



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.

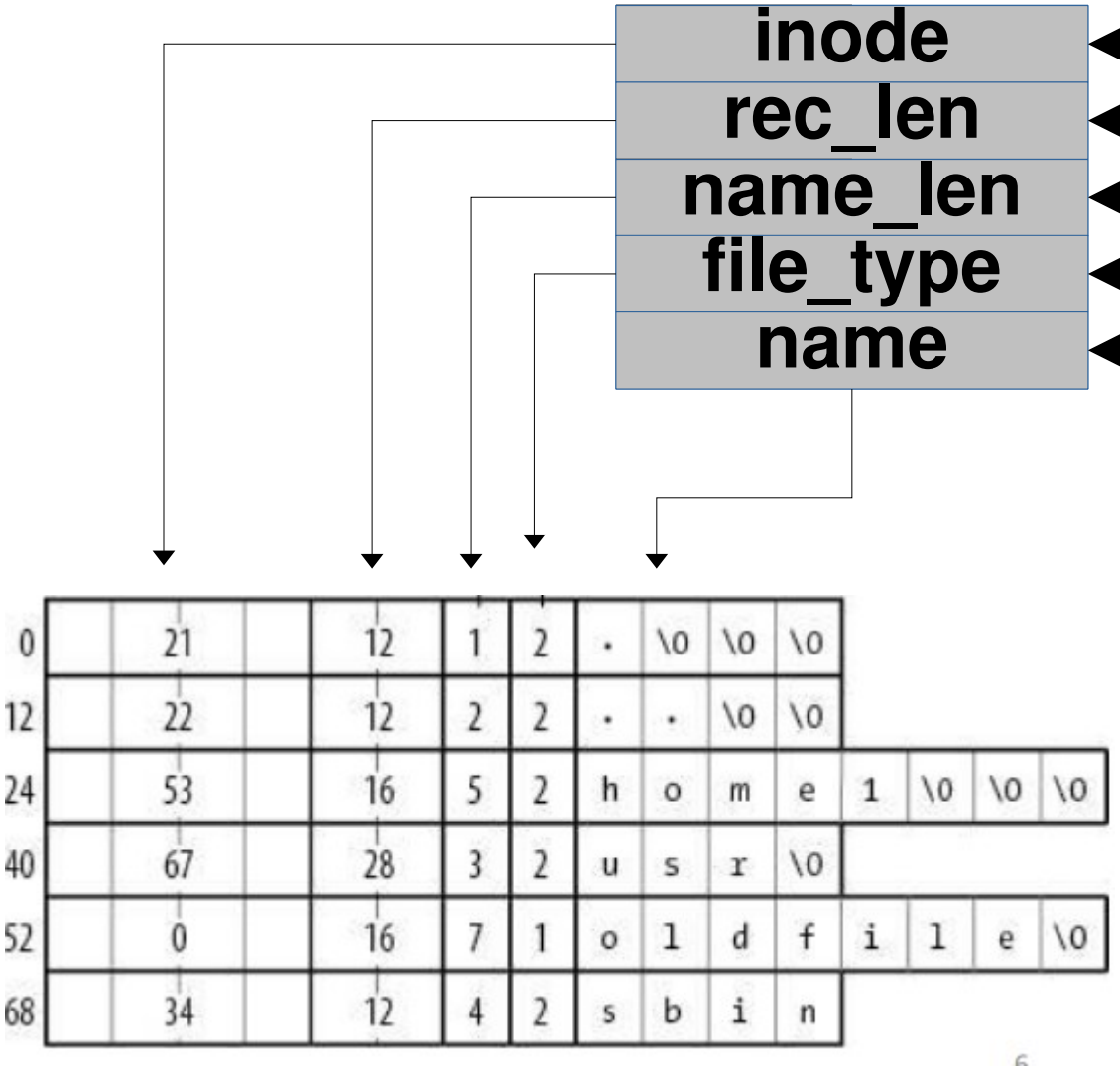


Taille bloc	512
Taille pointeur	4

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

- Un fichier a un nom...
  - toto
- ... et un chemin d'accès
  - /tmp/toto
- Le nom est stocké dans un répertoire
  - Liaison entre « nom de fichier » et « numéro d'i-noeud »
  - Lien matériel : plusieurs noms sur le même i-noeud
- Un répertoire est stocké dans un fichier
  - Stockage et gestion identique aux fichiers réguliers
  - Fichier « marqué » comme *répertoire*
- Le chemin d'accès n'est stocké nulle part
  - Implicitement stocké dans la succession de répertoires à parcourir pour atteindre le fichier
  - Le chemin n'est pas unique

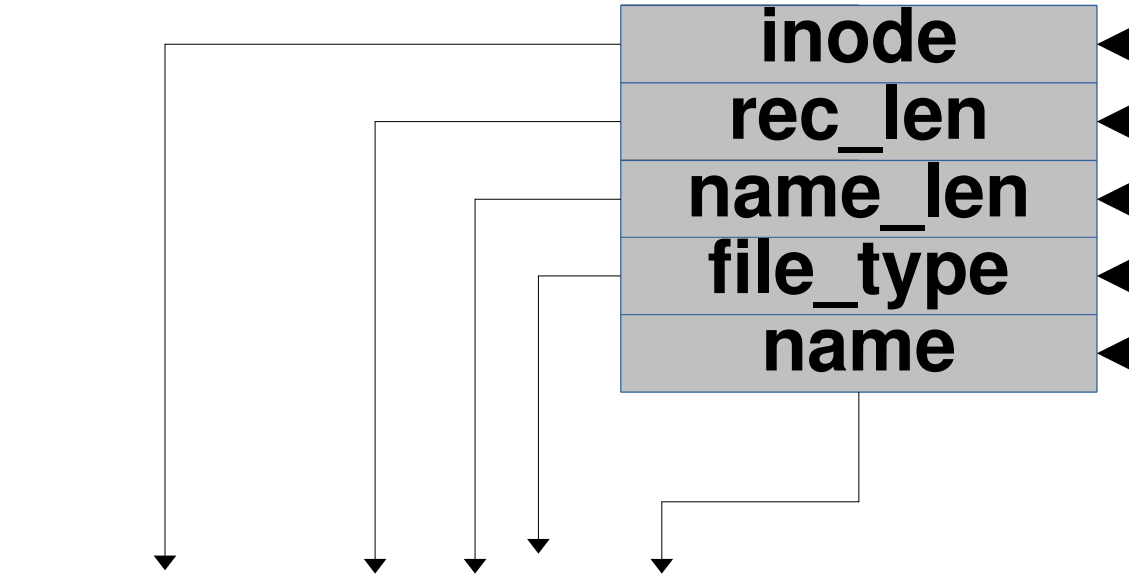
Répertoires dans Ext2



- Les noms sont « paddés » par des \0 : multiples de 4 octets
- . et .. sont représentés explicitement

rec_len # déplacement vers la prochaine entrée correcte

Répertoires dans Ext2



Numéro d'i-noeud du fichier

Taille de l'entrée dans le répertoire (multiple de 4)

Taille du nom de fichier sur 1 octet

Type du fichier

Nom du fichier

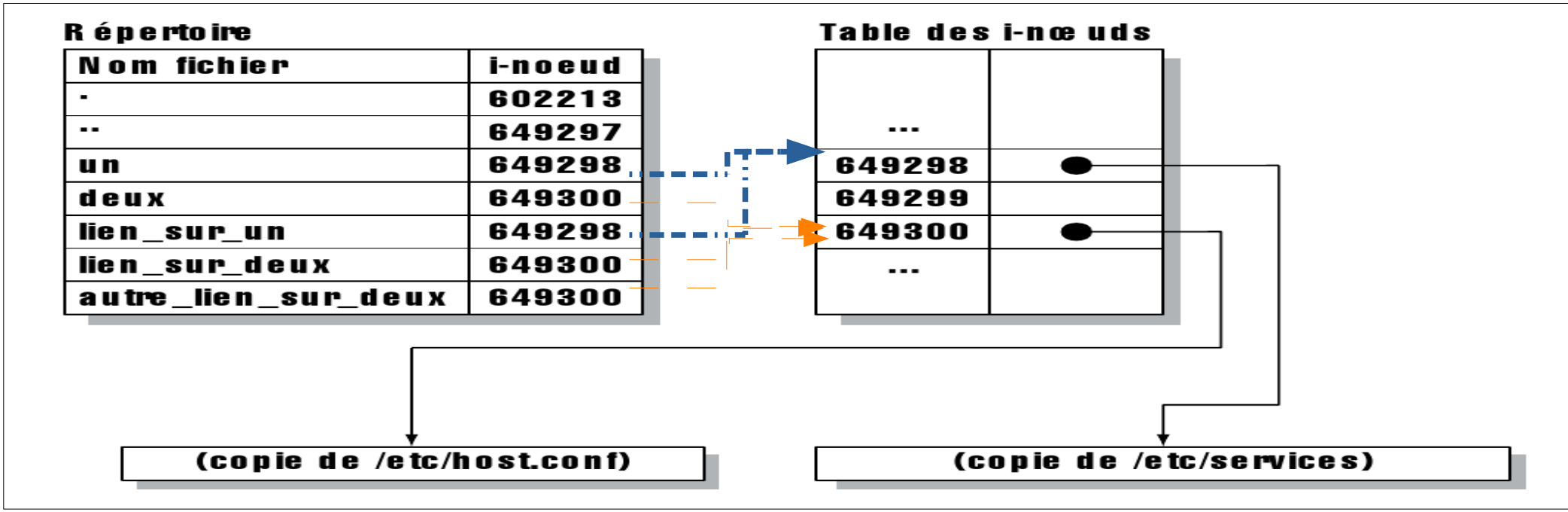
0	21	12	1	2	.	\0	\0	\0				
12	22	12	2	2	.	.	\0	\0				
24	53	16	5	2	h	o	m	e	1	\0	\0	\0
40	67	28	3	2	u	s	r	\0				
52	0	16	7	1	o	l	d	f	i	l	e	\0
68	34	12	4	2	s	b	i	n				

rec_len # déplacement vers la prochaine entrée correcte

Pour **effacer** un fichier :
- inode =0
- décalage de rec_len vers la prochaine entrée valide

Exemple : oldfile

Rappel du cours 2




```
$ cp /etc/services ./un
$ cp /etc/host.conf ./deux
$ ln un lien_sur_un
$ ln deux lien_sur_deux
$ ln deux autre_lien_sur_deux
```

d'après C. Blaess

Extent dans Ext4

- Un extent dans ext4 peut représenter jusqu'à 128 MiO d'espace contigü avec une taille de bloc de 4 KiO.
- Il peut y avoir quatre extent stockés directement dans l'inode.
- Lorsqu'il y a plus de quatre extent dans un fichier, le reste des extent est indexé dans un arbre.

Max. volume size : 1 EiB (for 4 KiB block size)

Max. file size : 16 TiB (for 4 KiB block size)

Max. number of files : 4 billion (specified at filesystem creation time)

Max. filename length : 255

https://www.backblaze.com/blog/what-is-an-exabyte/

1024^4 TiB tebibyte

1024^5 PiB pebibyte

1024^6 EiB exbibyte

Gestion des blocs libres : Inventaire des blocs libres

- **Vecteur de bits**: un bit par bloc indique s'il est occupé ou libre.
- **Espace libre chainé**: 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.

Gestion d'espace libre

Solution 1:
vecteur de bits (Macintosh, Windows2000)

bit[*i*] = 0 ⇒ block[*i*] libre
1 ⇒ block[*i*] occupé

- Exemple d'un vecteur de bits où les blocs 3, 4, 5, 9, 10, 15, 16 sont occupés:

0001110001100011...

- L'adresse du premier bloc libre peut être trouvée par un simple calcul (cpu)

- Trouver des blocs contigüs est facile

Gestion d’espace libre

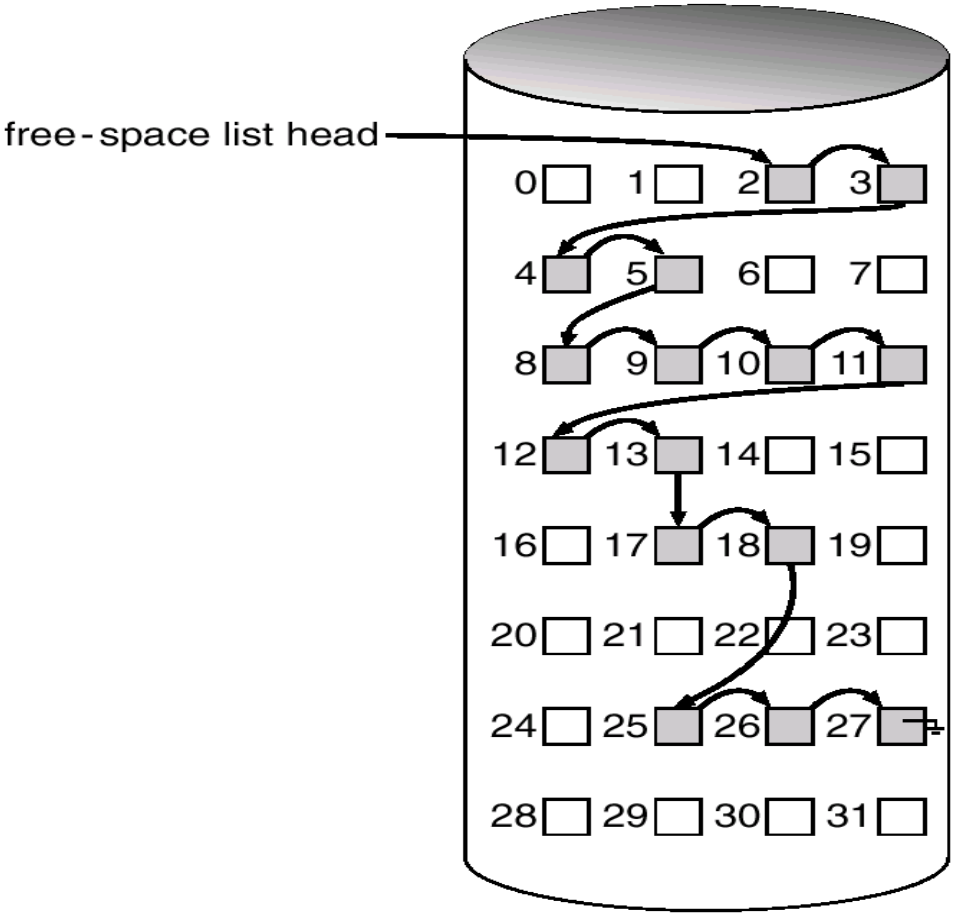
Solution 2 : Tous les blocs de mémoire libre sont liés ensemble par des pointeurs

Liste chainée :

- difficulté de trouver des blocs contigüs

- pas de perte de place

- pas besoin de traverser



Des variantes pour optimiser :

- grouping :

stockage d'un nombre fixé de blocs libres dans le premier bloc plus un pointeur vers le bloc qui contient les pointeurs vers les blocs libres.

- Counting :

Comme l'espace utilisé par les fichiers est souvent contigü (extents…)

- On conserve l'adresse du premier bloc libre et le nombre de blocs libres à sa suite.

- On stocke les @ + tailles des blocs libres

Comparaison

- Bitmap:
  - si la bitmap de tous les disques est gardée en mémoire principale, la méthode est rapide mais demande de l'espace de mémoire!
  - si les bitmaps sont gardées en mémoire secondaire -disque) , temps de lecture de mémoire secondaire..
- Liste chainée
  - Pour trouver plusieurs blocs de mémoire libre, plus accès disque
  - Pour augmenter l'efficacité, différentes stratégies (counting, grouping...)

Ordonnancement des E/S du disque

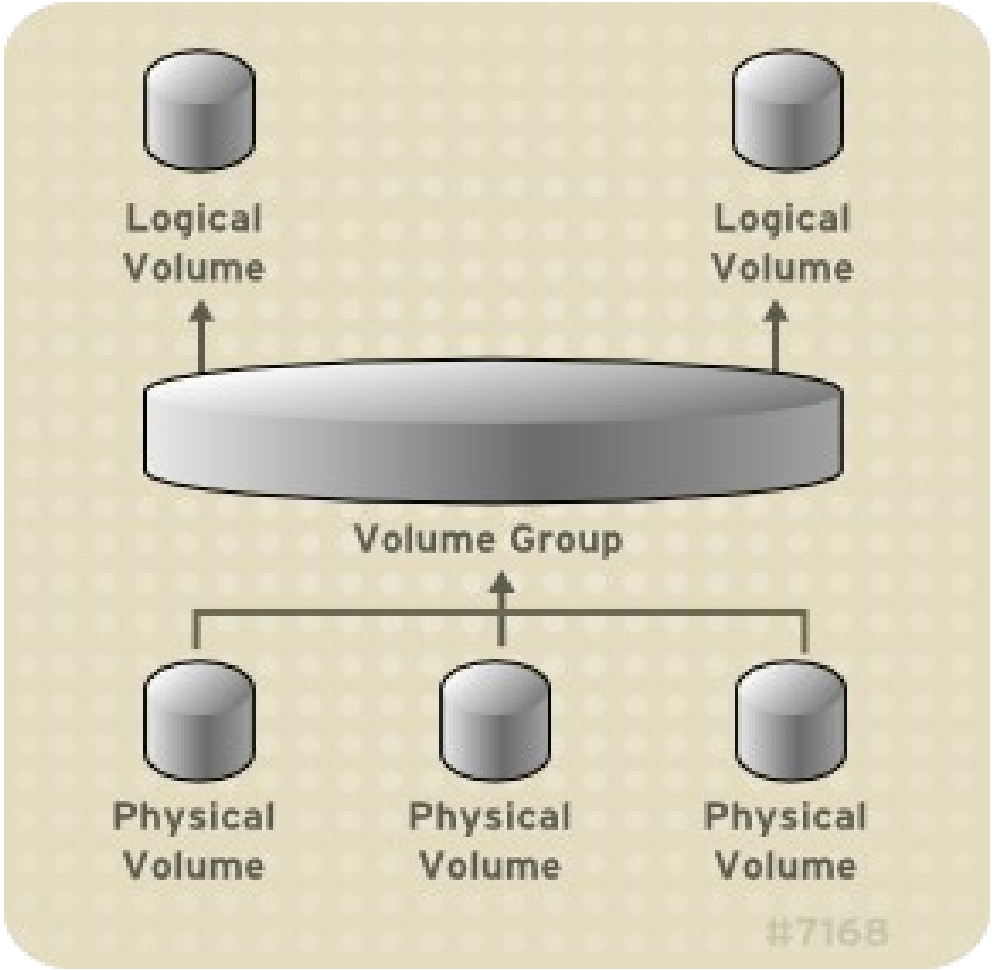
- Principe de localité → probabilité élevée d'accéder à des blocs successifs
- Il vaut la peine de retarder un peu les requêtes au disque et de les accumuler, il peut être possible de les grouper pour maximiser le débit et minimiser la latence globale.
- L'ordonnanceur des E/S du disque (elevator) permet de faire cette optimisation.
  - Completely Fair Scheduler (cfq) : meilleur dans la plupart des cas
  - Deadline : garanti une attente maximale
  - Noop : base comparative, simple queue FIFO
  - Anticipatory : délai élevé pour grouper un maximum de requête sur des disques très lents

Source: http://www.redhat.com/magazine/008jun05/features/schedulers/

Virtualisation du stockage

- Que 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: **Systèmes de fichiers avec consignation (log structured or journaling file system)**

Journalisation

- En cas de panne de courant, il se peut qu'une opération sur le disque soit terminée partiellement et cause une corruption.
- **Solution 1:** relire toutes les structures pour détecter les erreurs possibles. Très lent! Pas toujours possible…
- **Solution 2:** écrire les modifications dans un journal, puis exécuter sur la copie maitresse. Les modifications apportées avec succès sont retirées du journal. Il est plus rapide de ne réviser que les dernières opérations plutôt que toute la partition!

Systèmes de fichiers avec consignation (log structured or journaling file system)

- Les systèmes de fichiers avec consignation (**journalisés**) enregistrent chaque mise à jour du système de fichier comme une transaction (cf. BD)
- Toutes les transactions sont écrites dans un journal
  - **Une transaction commence par l'écriture dans le journal**
  - **Mais, le système de fichier (sur disque) n'est pas encore mis à jour**
- Les transactions dans le journal sont appliquées aux système de fichier de façon **asynchrone**
- Si tout se passe bien :
    - **Quand le système de fichier est effectivement modifié, la transaction est enlevée du journal**

Systèmes de fichiers avec consignation (log structured or journaling file system)

Après la reprise (redémarrage) :

- Si problème pendant l'écriture sur le disque :
    - **La transaction est effectuée , le système fichier modifié puis la transaction est enlevée du journal**
- Si problème pendant l'écriture dans le journal :
    - **La transaction n’est pas effectuée le système de fichier n’est pas modifié**
- **Plus lent, plus consommateur de ressources mais plus sûr !!!**

Données et sauvegardes

RAID
Redundancy Array of Inexpensive Disks

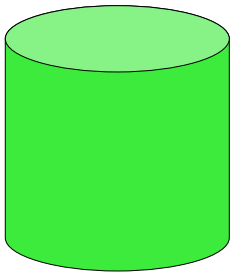
- on n'utilise plus de disques durs indépendants mais un ensemble de disques durs (en général rackable) gérés par des contrôleurs disques spécialisés.
- L'ensemble est censé supporter la panne d'au moins un disque dur. En cas de panne d'un disque dur, le système continue de travailler. Un disque en remplacement sera rempli par des données reconstruites à partir des données sur le reste des disques de l'ensemble.

https://buzut.net/tirez-le-meilleur-de-vos-disques-durs-avec-raid/

l'utilisateur voit un seul disque virtuel

plusieurs disques physiques (redondance)

raid-1 : mirroring



100G

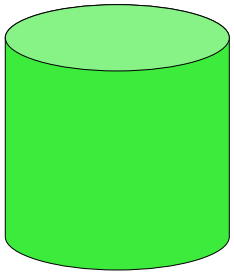
=

100G

+

100G

raid-5 : parité



200G

=

100G

+

100G

+

100G

raid : parité

disque raid (virtuel)	disque-1		disque-2		disque-3 parité
01	0	⊕	1	=	1
11	1		1		0
00	0		0		0
10	1	→	⊕	=	0
01	0		1		1
.	.		.		.
.	.		.		.
.	.		.		.

nombre impair de 1

on peut toujours reconstruire une valeur manquante avec un XOR

en pratique:
parité distribué par blocs

Exemple avec 3 disques + 1 de parité :

| 101 | 010 | 011 | ??? |

XOR (101, 010, 011) = 100

Résultat :

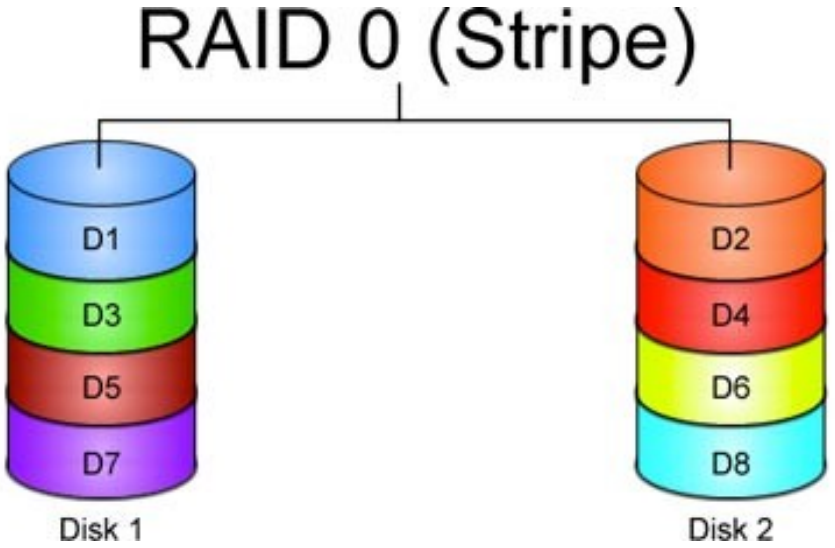
| 101 | 010 | 011 | **100** |

Si le second disque est en erreur :
| 101 | ~~010~~ | 011 | 100 |

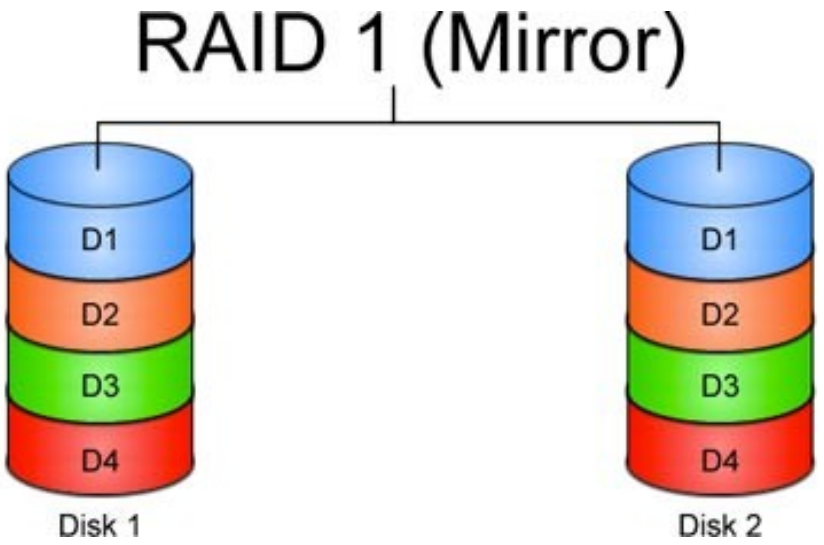
En utilisant les 3 disques restants :

XOR (101, 011, 100) = **010**

Table de vérité de XOR		
A	**B**	**R = A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

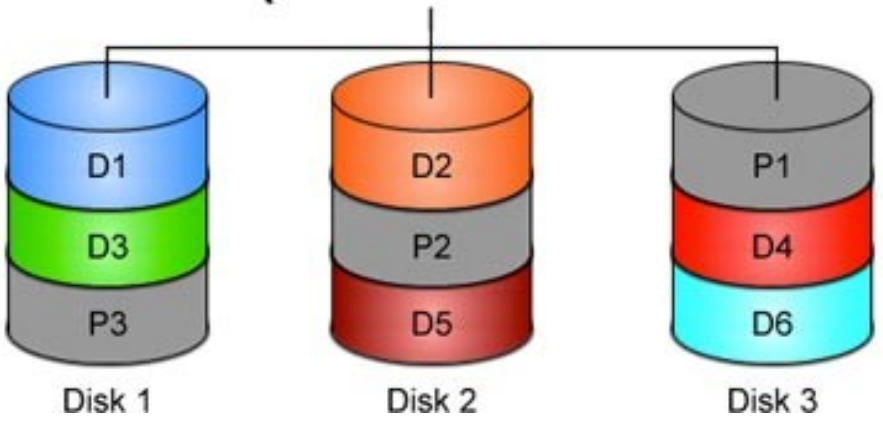


- Principe du RAID 0 : **striping**
  - minimum de 2 disques
  - les données sont découpées en blocs (A, B, C, etc.) écrits sur des disques distincts
- Avantages :
  - I/O en paralléle
  - simple à réaliser
  - maximum de performance
- Inconvénients :
  - non tolérant aux pannes (un disque en panne $\Rightarrow$ tout est perdu)



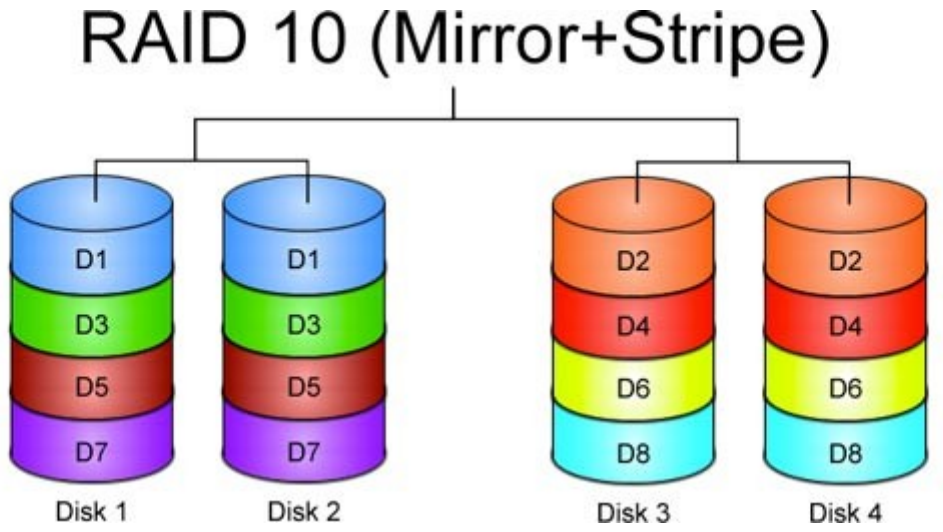
- Principe du RAID 1 : **mirroring et duplexing**
  - minimum de 2 disques
  - les données sont découpées en blocs (A, B, C, etc.) écrits sur des disques jumeaux
- Avantages :
  - lecture sur n'importe lequel des disques jumeaux $\Rightarrow$ lecture deux fois plus performante
  - simple à réaliser
- Inconvénients :
  - pas très efficace (100% de disques en plus)
  - charge le CPU en général (car réalisation en logiciel et pas en hardware)

RAID 5 (Drives with Parity)



- Principe du RAID 5 :
  - minimum de 3 disques
  - chaque bloc d'octets est écrit sur un disque
  - la parité est dispersée sur tous les disques évitant ainsi un goulet d'étranglement du RAID 4.
- Avantages :
  - Fort taux de transfert en lecture, taux moyen en écriture
  - Solution populaire
- Inconvénients :
  - Contrôleur complexe
  - Reconstruction délicate d'un disque

RAID combiné



RAID 10 (1+0)

- mirroring puis stripping.
- 4 disques au minimum.
- Fiable: on doit avoir un défaut sur tous les éléments d'une grappe pour mettre en défaut le RAID dans son ensemble.
- Reconstruction est assez rapide.
- Mais perte d'au minimum 50% de l'espace de stockage.

raid : utilisation

le raid n'est pas une sauvegarde

il n'y a pas d'historique

panne disque

interruption de service

performance

complexité = danger

important!

3 à 5 ans...

disques pas fiables !

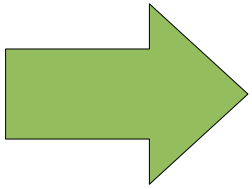
erreurs humaines

incendie, vol

...

beaucoup d'heures de travail

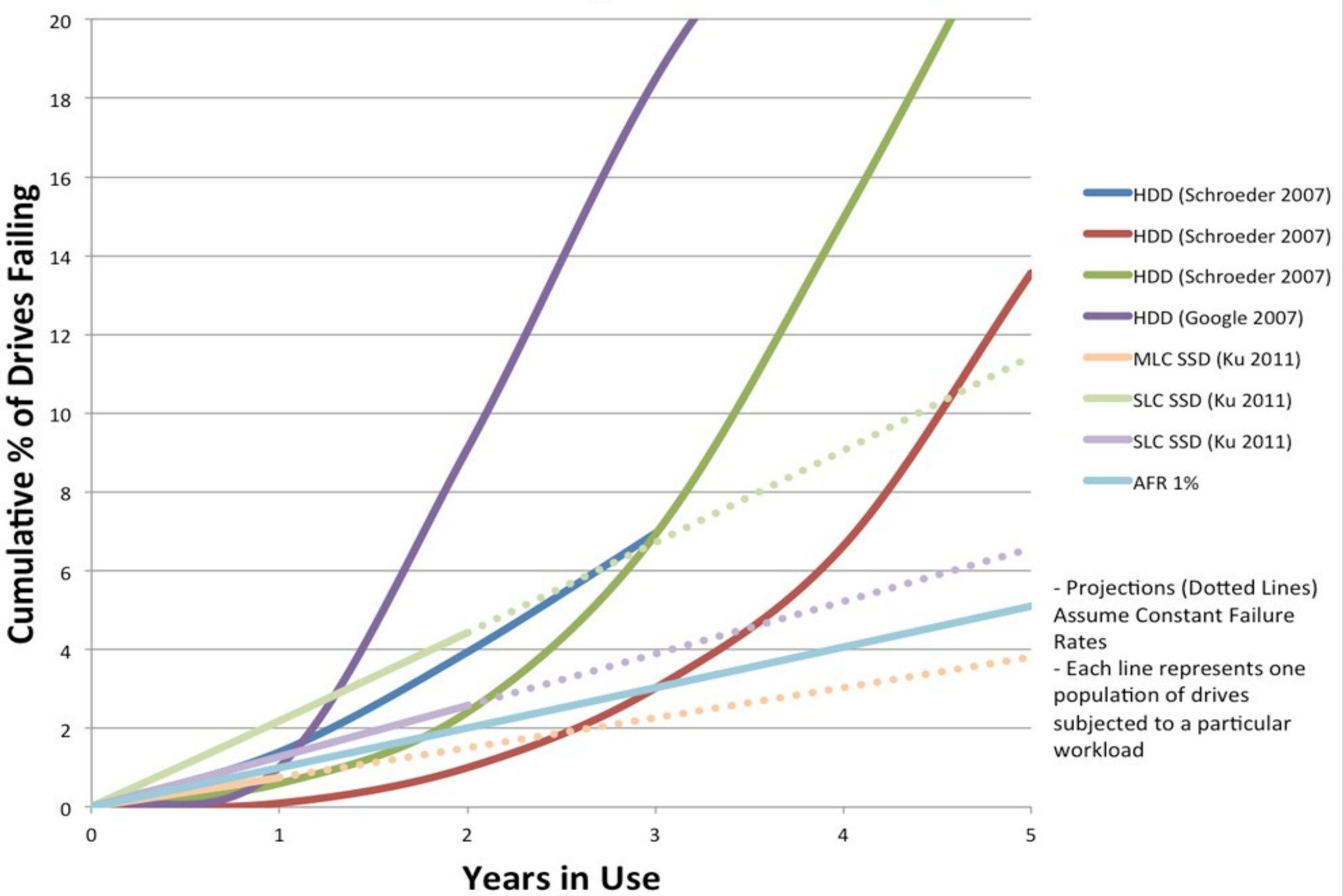
données critiques



Les données sont précieuses!

Fiabilité

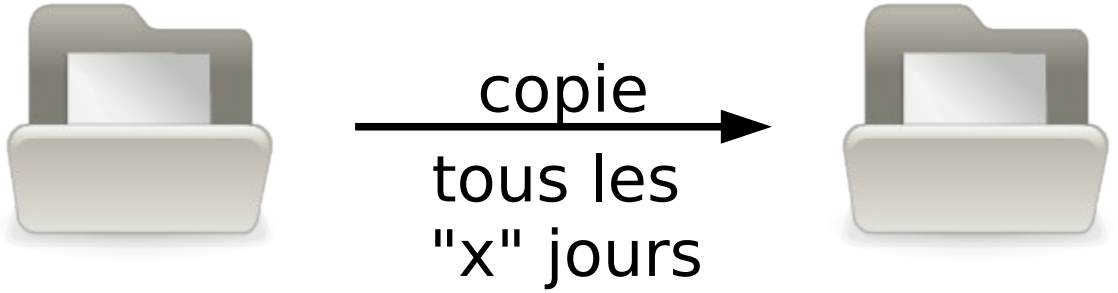
tom's hardware



sauvegardes

- facile à faire
- semi-automatique
- régulières
- lieu séparé
- formats standards
- vérifier régulièrement!

sauvegarde par duplication



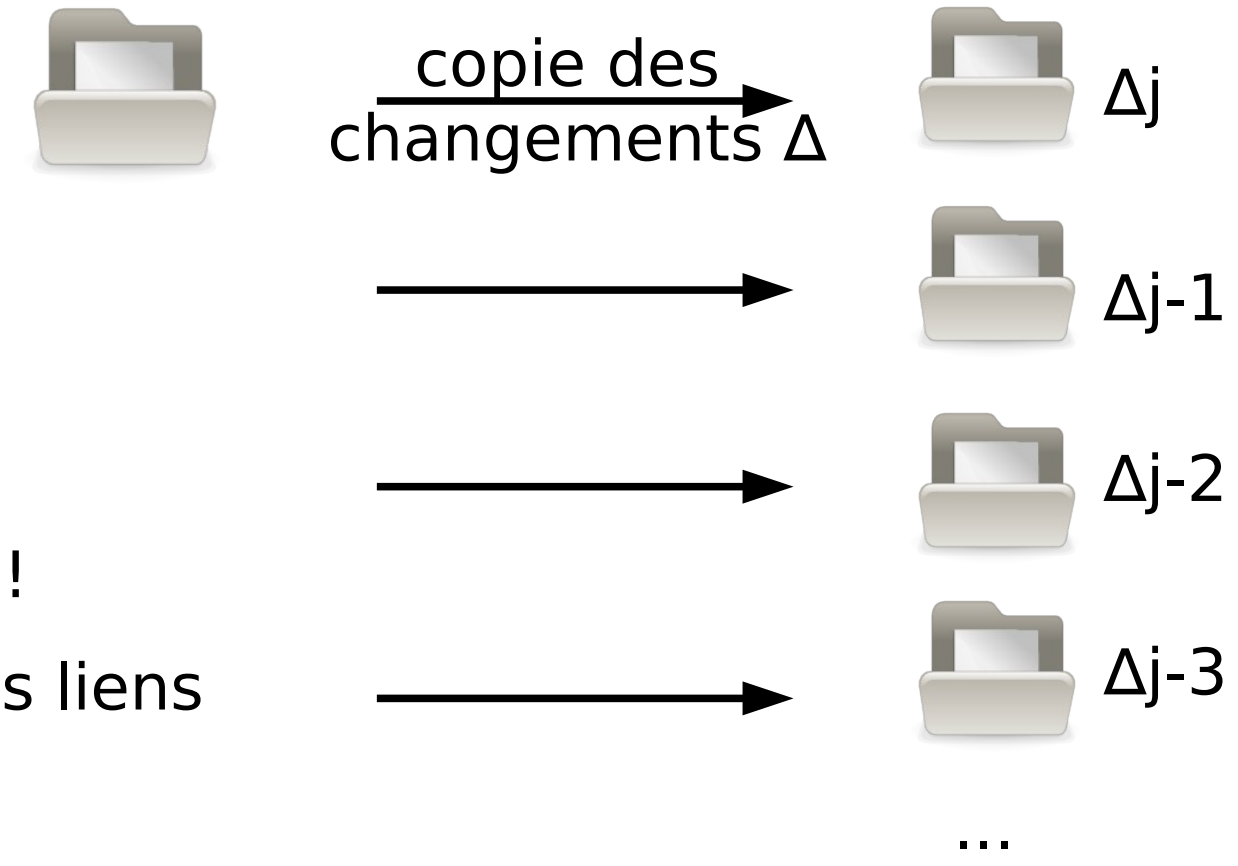
pas d'historique

pertes de donnée pas remarquée => définitive !!

procédure très simple

équipement simple

sauvegarde incrémentale



beaucoup de place!

Sauf si on utilise des liens

historique complet

très grande sécurité

en pratique:
schémas intermédiaires

Sources

http://www-info.iutv.univ-paris13.fr/~bosc

- web, Le Monde Informatique...