
Le système de fichiers

Louis Mandel

`louis.mandel@lri.fr`

Laboratoire de Recherche en Informatique

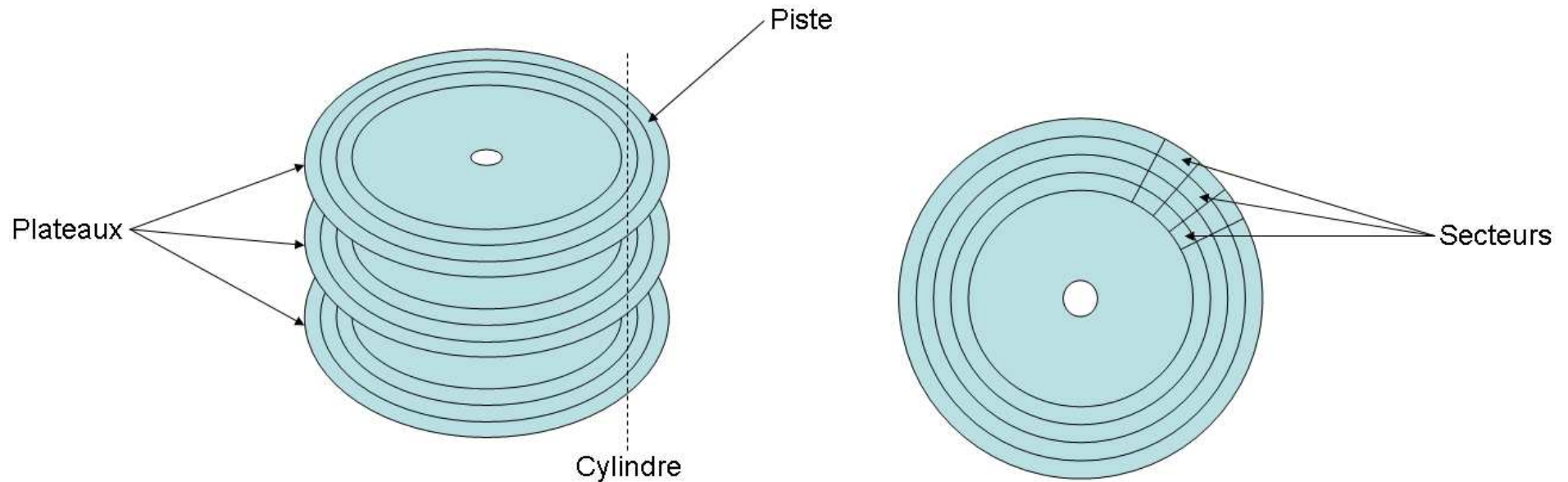
Université Paris-Sud 11

IFIPS

Cycle ingénieur de la filière étudiant

année 2007/2008

Structure physique d'un disque



- Secteur : plus petite unité qui peut être lue par un disque (typiquement 512 octets)

Remarques : des figures de ce cours ont été prises sur Wikipedia

Organisation et codage des secteurs

- ▶ Adressage
 - ▶ CHS (*Cylinder, Head, Sector*)
 - {num. cylindre, num. face, num secteur dans la piste}
 - ▷ num. cylindre, num. face = num. piste
 - ▶ LBA (*Logical Block Address*)
 - ▷ Adresses logiques par numéros consécutifs
 - ▷ Le contrôleur de disque dispose d'une table de correspondance entre les adresses LBA et CHS

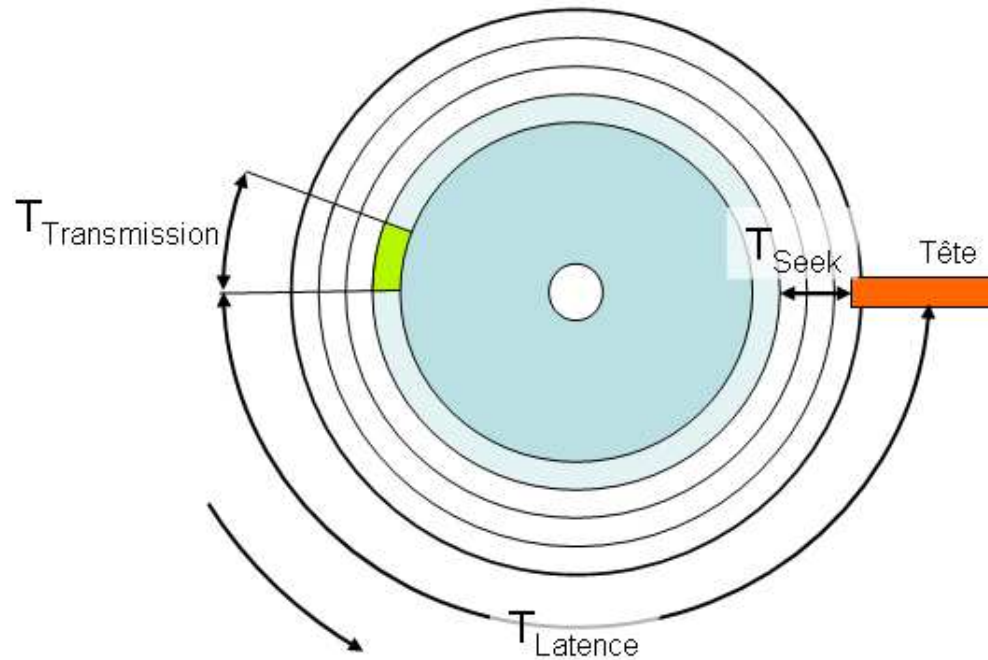
Formatage de bas niveau

- ▶ Exemple de structure d'un secteur

gap	synchro	id	données	CRCC	gap	synchro
-----	---------	----	---------	------	-----	---------

- ▶ Informations auxiliaires
 - ▶ Synchronisation
 - ▶ Identification du secteur
 - ▶ Codes de détection et correction d'erreurs (CRCC = Cyclic Redundancy Check Code)
 - ▶ Séparations des données par des « *gaps* »
- ▶ Le formatage de bas niveau :
 - ▶ partage le disque en secteurs
 - ▶ marquage des informations auxiliaires

Performances des disques



- ▶ Temps de recherche (*seek time*) : positionnement de la tête en face du bon cylindre
- ▶ Temps de latence : positionnement du secteur en face de la tête de lecture (dépend de la vitesse de rotation)
- ▶ Temps de transfert : transfert des données vers l'ordinateur

Stratégies d'ordonnement de requêtes

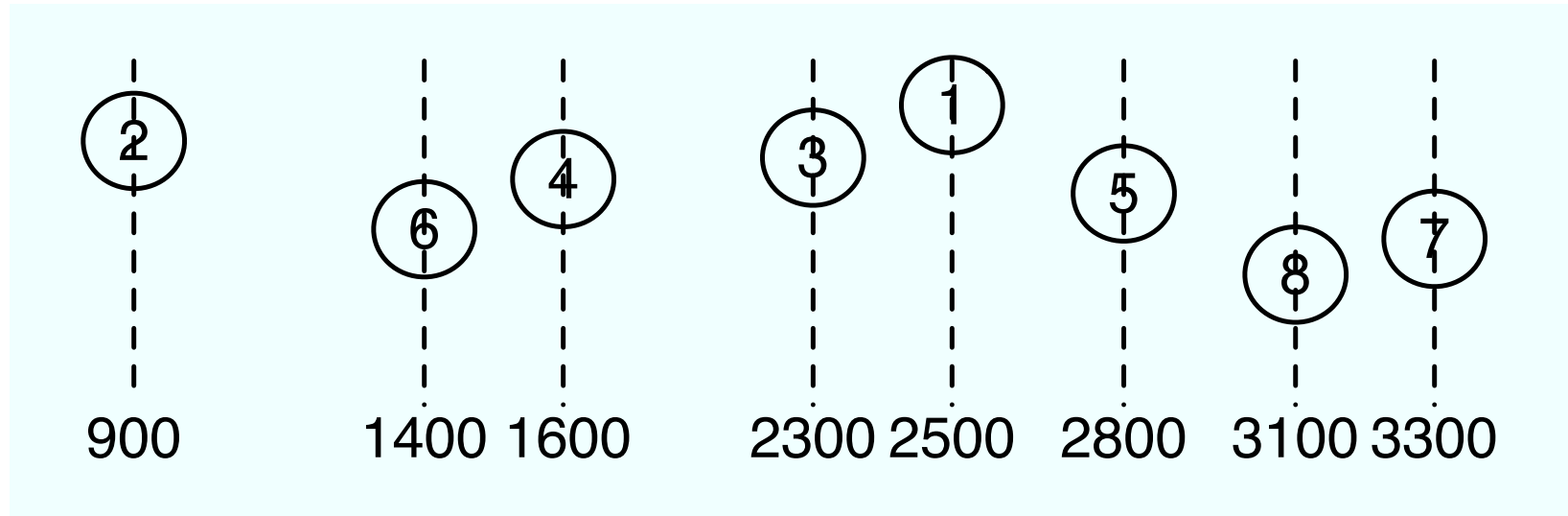
- ▶ Changer l'ordre des requêtes pour :
 - ▶ Réduire les délais de recherche de cylindres
 - ▶ Réduire les délais de latence pour un même cylindre
 - ▶ Optimisation globale des délais de positionnement et rotation
- ▶ **Attention** : Pour garantir que l'on accède toujours à la dernière version d'une donnée, les optimisations doivent conserver l'ordre des requêtes d'écriture et de lecture concernant un même secteur.

Réduction des positionnements de cylindres

- ▶ Les stratégies doivent tenir compte des nouvelles requêtes qui sont dynamiquement déposées pendant les échanges
- ▶ Stratégie Premier Arrivé Premier Servi (FCFS = First Come First Served)
 - ▶ Avantage : équité de service
 - ▶ Inconvénient : Nombreux déplacements d'un bout à l'autre du disque
- ▶ Stratégie Plus Proche Piste (SSF = Shortest Seek Time)
 - ▶ Inconvénient : problème d'équité. L'optimisation doit éviter de ne servir qu'un sous ensemble de cylindres proches
- ▶ Stratégie de Balayage (Scan)
 - ▶ Parcourir tous les cylindres dans un sens en servant toutes les requêtes puis dans l'autre sens
 - ▶ Variantes :
 - ▷ Ne servir les requêtes que dans un sens (Circular Scan)
 - ▷ Limiter le nombre de requêtes servies pour chaque cylindre

Réduction des positionnements de cylindres : Exemple

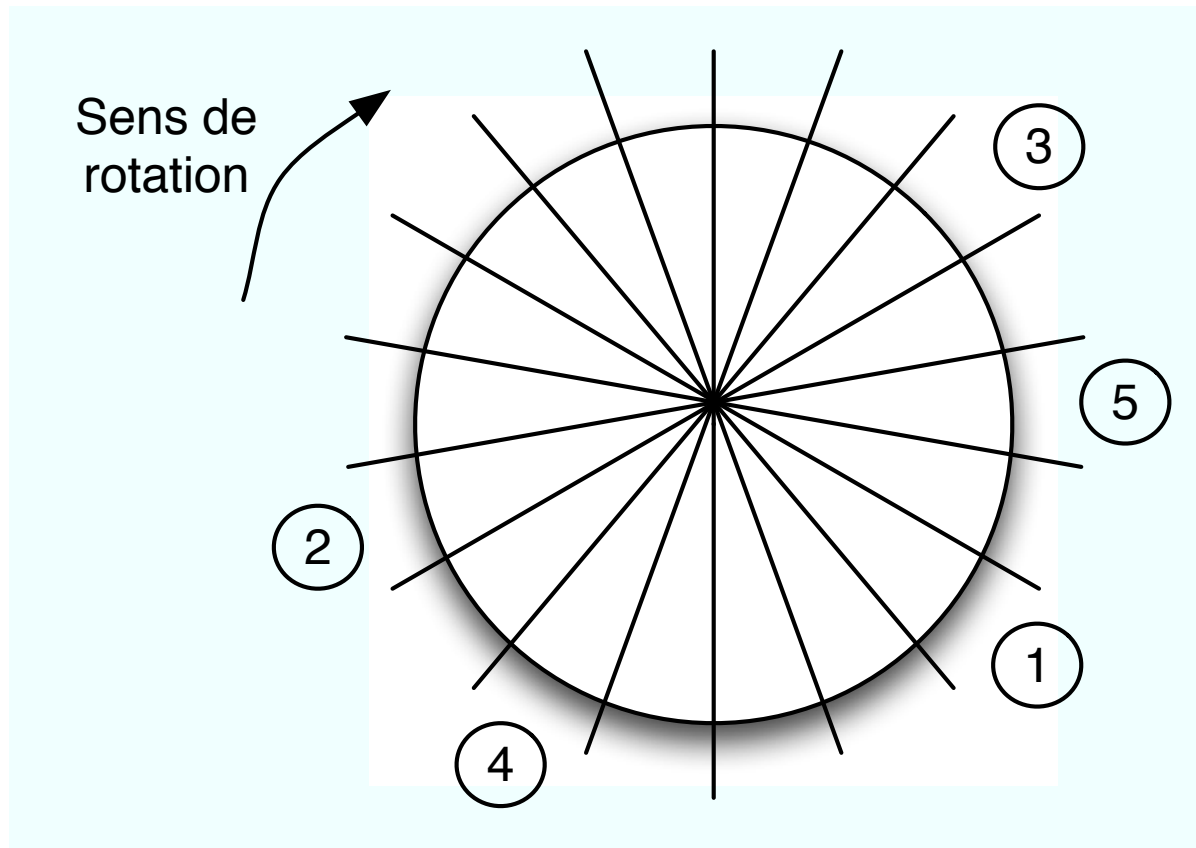
- Supposons que le disque doit traiter les demandes requêtes suivante :



- Représenter les stratégies FCFS, SSF, Scan et Circular Scan

Réduction des attentes de rotation

- ▶ Stratégie Premier Arrivé Premier Servi (FCFS)
- ▶ Stratégie Plus Proche Secteur (SATF = Shortest Acces Time First)



Optimisations globales

- ▶ Pour des cylindres proches, les délais de positionnement et de rotation sont comparables
- ▶ Optimisation de la somme des temps
- ▶ STF = Shortest Time First
 - ▶ On sert d'abord la requêtes qui minimise la somme des deux délais
 - ▶ Problème d'équité
- ▶ GSTF = Grouped STF
 - ▶ On divise le disque en groupes de cylindres
 - ▶ On sert circulairement les groupes de cylindres
 - ▶ On applique STF dans chaque groupe de cylindres
- ▶ GSTF(x) où x est le nombre de cylindres par groupe

Optimisation du temps de transfert

- ▶ Utilisation de caches

- ▶ Mémoire cache du processeur d'entrées/sorties

- ▶ Mémoire principale

- ▷ Tampons système (*buffer cache*)

- ▷ Projection de fichiers en mémoire (*memory mapped files*)

```
#include <sys/mman.h>
```

```
void *mmap(void *start, size_t length, int prot,  
           int flags, int fd, off_t offset);
```

```
int munmap(void *start, size_t length);
```

Projection de fichiers en mémoire : Exemple (proj.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>

int main(int argc, char **argv) {
    int desc1, desc2, desc3;
    char c;
    char *adr1, *adr2;
    if ( (desc1 = open(argv[1], O_RDWR)) == -1 ) { perror(argv[1]); exit(2); }
    if ( (desc2 = open(argv[1], O_RDWR)) == -1 ) { perror("open2"); exit(2); }
    if ( (desc3 = open(argv[1], O_RDWR)) == -1 ) { perror("open3"); exit(2); }
    adr1 = (char *)mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, desc1, 0);
    if (adr1==MAP_FAILED) { perror("mmap1"); exit(2); }
    adr2 = (char *)mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_PRIVATE, desc2, 0);
    if (adr2==MAP_FAILED) { perror("mmap2"); exit(2); }
    printf("valeur de adr1: %p\nvaleur de adr2: %p\n", adr1, adr2);
    printf("10-eme caractere de adr1: %c\n", adr1[9]);
    printf("10-eme caractere de adr2: %c\n", adr2[9]);
}
```

Projection de fichiers en mémoire : Exemple (suite)

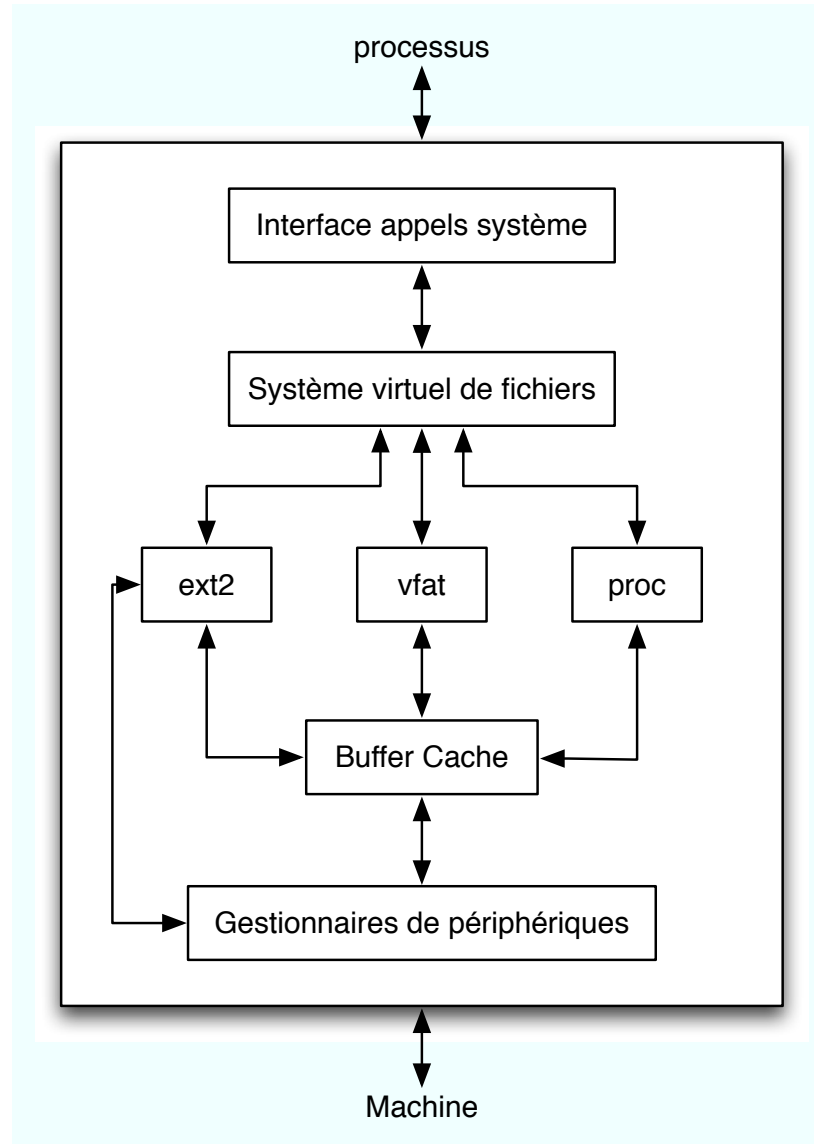
```
    adr1[9] = adr1[9] + 1;
    printf("10-eme caractere de adr1: %c\n", adr1[9]);
    printf("10-eme caractere de adr2: %c\n", adr2[9]);
    lseek(desc1, (off_t)9, SEEK_SET);
    read(desc1, &c, 1);
    printf("10-eme caractere lu par read: %c\n", c);
    adr2[9] = adr2[9] + 1;
    printf("10-eme caractere de adr1: %c\n", adr1[9]);
    printf("10-eme caractere de adr2: %c\n", adr2[9]);
    lseek(desc3, (off_t)9, SEEK_SET);
    read(desc3, &c, 1);
    printf("10-eme caractere lu par read: %c\n", c);
    lseek(desc3, (off_t)9, SEEK_SET);
    write(desc3, "X", 1);
    printf("10-eme caractere de adr1: %c\n", adr1[9]);
    printf("10-eme caractere de adr2: %c\n", adr2[9]);
    sleep(10);
    return 0;
}
```

Allocation du disque

- ▶ Problèmes d'allocation du disque
 - ▶ Représentation de l'espace libre
 - ▷ limiter la taille des tables
 - ▶ Recherche d'un secteur libre ou d'un ensemble de secteurs libres
 - ▷ Fragmentation des fichiers : dispersion sur les cylindres éloignés
 - ▷ Fragmentation de l'espace libre : rend l'allocation difficile
- ▶ Les blocs = N secteurs successifs sur la même piste
 - ▶ Blocs petits :
 - ▷ décomposition peu efficace des accès
 - ▷ allocation trop fine
 - ▷ tables trop importantes
 - ▶ Blocs gros :
 - ▷ Fragmentation interne
- ▶ Un disque trop rempli a des performances dégradées

Implantation du système de fichiers de Linux

Le système virtuel de fichiers



Opération assurées par le VFS

- ▶ Les fonctionnalités assurées par le VFS sont indépendantes des systèmes de fichiers physiques.
- ▶ Le VFS implante
 - ▶ le cache des noms : association noms de fichiers / identifiant (numéro de périphérique, numéro d'i-nœud)
 - ▶ le buffer cache : cache pour les entrées/sorties en mode bloque
 - ▶ les parties commune des appels systèmes
 - ▷ vérification des arguments
 - ▷ conversion des noms de fichiers
 - ▷ vérification des permissions
 - ▷ appel de la fonction du système de fichiers sous-jacent correspondant à l'appel système

Système de fichiers virtuel

- ▶ Le VFS utilise une approche « orientée objet »
 - ▶ chaque système de fichiers monté doit implanter des opérations de bases
- ▶ Types d'opérations que doivent fournir un système de fichiers
 - ▶ Opérations sur des systèmes de fichiers
 - ▷ lecture/écriture d'un i-nœud, modification du super-bloc, etc.
 - ▶ Opérations sur les i-nœuds en cours d'utilisation
 - ▷ création/suppression de liens, répertoires, etc.
 - ▶ Opérations sur des fichiers ouverts
 - ▷ primitives d'entrées/sorties sur les fichiers, etc.
 - ▶ Opérations sur les quotas

Buffer cache

- ▶ Liste des tampons mémoire en cours d'utilisation
- ▶ Lecture d'un bloc :
 - ▶ le contenu du bloc est placé dans le tampon
- ▶ Écriture d'un bloc :
 - ▶ le changement est effectué dans le tampon
 - ▶ le tampon est marqué comme modifié
- ▶ À intervalles réguliers, le processus `update` appelle la primitive `sync` pour forcer la réécriture de tous les tampons modifiés sur les disques

Implantation de systèmes de fichiers à i-nœuds

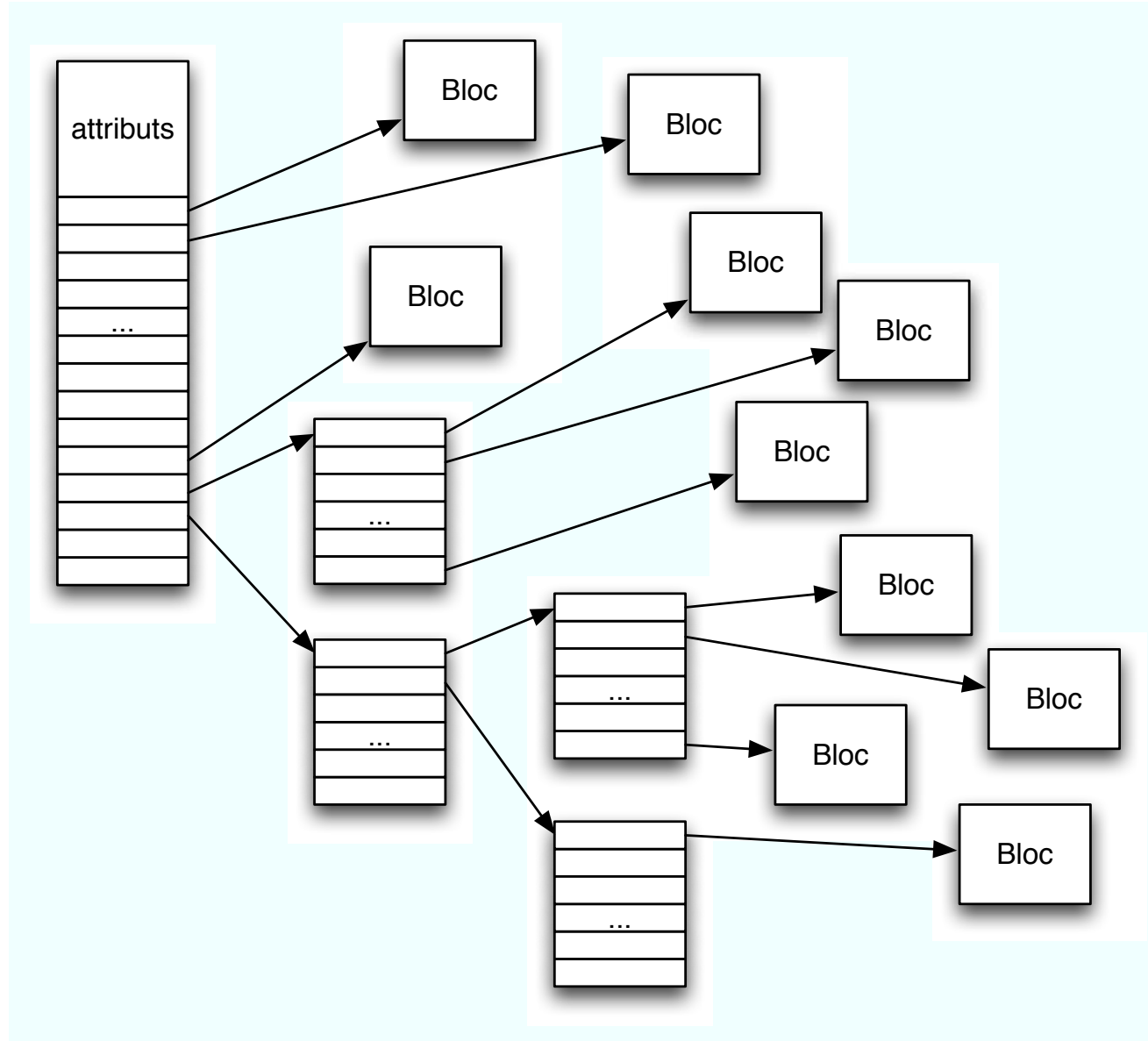
Rappels

- ▶ Structure logique du disque



- ▶ Le superbloc contient en particulier
 - ▶ la table d'allocation des blocs
 - ▷ Utilisation d'un bit par bloc
 - ▶ la table d'allocation des i-nœuds
- ▶ Le nombre d'i-nœud fixé au moment du formatage (de haut niveau)

Rappels : Structure d'un i-nœud



Avantages et inconvénients

▶ Avantages :

- ▶ Structure arborescente des i-nœuds permet un accès quasi-immédiat à chaque bloc du fichier
- ▶ Structure des répertoires est simple : toutes les attributs des fichiers sont directement dans les i-nœuds
- ▶ Possibilité de créer des liens physiques
- ▶ Table d'allocation petite
 - ▷ Permet de faire des petits blocs et donc de limiter la fragmentation interne

▶ Inconvénients :

- ▶ Beaucoup de méta-données
- ▶ Pointeurs inutiles pour les petits fichiers (compensé par la faible fragmentation interne)

Exemple : EXT2

- ▶ Ext était le premier système de fichier à utiliser le VFS
- ▶ Ext est un système de fichiers à base d'i-nœuds
- ▶ Ext2 est une extension de Ext
- ▶ Ext2 supporte tous les types de fichiers Unix
- ▶ Limitations :

	Minix FS	Ext	Ext2
Taille maximale	64 Mo	2 Go	2 To
Taille maximale d'un fichier	64 Mo	2 Go	2 Go
Taille maximale d'un nom de fichier	14 c	255 c	255 c

Ext2 : Structure physique

- ▶ Structure à deux niveaux
 - ▶ Les groupes de blocs

init	Groupe de blocs 1	Groupe de blocs 2	...	Groupe de blocs N
------	-------------------	-------------------	-----	-------------------

- ▶ Un groupe de blocs

super bloc	descripteurs	bitmap blocs	bitmap i-nœuds	table des i-nœuds	blocs de données
---------------	--------------	-----------------	-------------------	----------------------	------------------

Ext2 : Structure physique

- ▶ Chaque groupe de bloc contient
 - ▶ Une copie du superbloc
 - ▷ augment la fiabilité du système de fichier
 - ▷ réduit les temps d'accès au superbloc
 - ▶ Table des descripteurs
 - ▷ Contient les adresses des blocs de bitmap et de la table des i-nœuds
 - ▶ Bitmap pour les blocs
 - ▷ Table de bits
 - ▷ Indique pour chaque bloc du groupe s'il est libre ou occupé
 - ▶ Bitmap pour les i-nœuds
 - ▷ Indique pour chaque i-nœuds du groupe s'il est alloué ou disponible
 - ▶ Table des i-nœuds
 - ▷ une partie de la table des i-nœuds du système de fichiers
 - ▶ Blocs de données

Ext2 : Caractéristiques

- ▶ Caractéristiques de performances
 - ▶ Prélecture : lecture de plusieurs blocs contiguës à la requête de lecture
 - ▶ Allocation optimisées : allocation le plus possible dans le même groupe de blocs
 - ▶ Préallocation : allocation jusqu'à huit blocs adjacents lors de l'allocation d'un bloc
 - ▶ Lutte contre la fragmentation

Ext2 : Caractéristiques

- ▶ Autres caractéristiques
 - ▶ Choix de la taille du bloc
 - ▶ Liens symboliques rapides
 - ▶ Mémorisation de l'état
 - ▷ Propre : il n'y a eu que des lecture
 - ▷ Sale : il y a eu au moins une écriture
 - ▶ Mémorisation du nombre de fois que le système a été monté et date de la dernière vérification d'intégrité
 - ▶ Blocs réservés au super-utilisateur
 - ▶ Effacement sécurisé : le contenu des i-nœuds sont physiquement effacés
 - ▶ Fichiers auxquels on peut seulement ajouter des données

Persistence des Fichiers

Persistence des Fichiers

Incohérences du système de fichiers

Détection et Réparation des incohérences

- ▶ But : savoir remettre le système de fichier dans un état cohérent même en cas de problème (exemple : après une coupure de courant)
- ▶ Exemple de problème : suppression d'un fichier se fait en au moins deux étapes :
 - ▶ supprimer son entrée dans le répertoire courant
 - ▶ marquer l'i-nœud du fichier comme libre

Que se passe-t-il si la machine s'arrête entre ces deux opérations ?

Ext2 : File System Check (fsck)

- ▶ Comment retourner dans un système de fichiers « propre » à partir après une erreur ?
- ▶ File System Check : Opération très couteuse !
- ▶ Vérification au niveau du fichier
 - ▶ Parcourir toute l'arborescence du système de fichier pour corriger
 - ▷ bitmap des i-nœuds
 - ▷ compte de liens dans les i-nœuds
- ▶ Vérification au niveau du bloc

fsck : Vérification au niveau du bloc

- ▶ Pour chaque bloc b on construit :
 $TF[b]$ = nombre de référence de b dans les fichiers
- ▶ On appelle TA la table d'allocation des blocs
- ▶ le système est cohérent si :
 $(TF[b] = 0 \text{ et } TA[b] = 0)$ ou $(TF[b] = 1 \text{ et } TA[b] = 1)$
 - ▶ si $TF[b] = 0$ et $TA[b] = 1$ alors b est orphelin
 - ▶ si $TF[b] \geq 1$ et $TA[b] = 0$ alors il faut modifier TA
 - ▶ si $TF[b] \geq 2$ et $TA[b] = 1$ alors il faut allouer et recopier le bloc pour avoir dans chaque fichier des blocs distincts mais de contenu identiques
- ▶ Remarque : l'information contenu dans les fichiers est considérée comme plus pertinente

Persistance des Fichiers

Les systèmes à journal

Les systèmes à journal

- ▶ Idée : marquer les changements à faire dans un journal avant les effectuer. Ainsi, les changements sont
 - ▶ soit réalisés normalement
 - ▶ soit réalisés au redémarrage
 - ▶ soit pas réalisés si l'arrêt arrive avant l'écriture dans le journal
- ▶ Formes physiques possibles d'un journal :
 - ▶ Fichier standard
 - ▶ Zone réservée du disque
- ▶ Types de journaux :
 - ▶ Journal complet : contient les modifications faites aux données et aux méta-données
 - ▶ Méta-journal : contient seulement les modifications faites aux méta-données

Les systèmes à journal : Exemple

- ▶ Augmenter la taille d'un fichier peut se faire en trois étapes
 - ▶ changer l'information de taille dans l'i-nœud
 - ▶ allouer l'espace nécessaire dans la bitmap des blocs
 - ▶ ajouter les données à la fin du fichier
- ▶ Supposons un arrêt entre les étapes 2 et 3
 - ▶ Un journal complet sera détecter et corriger le problème
 - ▶ Un méta-journal laissera un fichier avec un contenu incorrect

Ext3 ~ Ext2 + journal

- ▶ Ext3 propose trois niveaux de journal :
 - ▶ Journal complet
 - ▶ Méta-journal
 - ▶ Journal ordonné
 - ▷ identique au méta-journal
 - ▷ mais les données sont écrites avant les méta-données
 - ▷ On peut perdre des données mais un fichier ne peut pas contenir des informations fausses
- ▶ Avantages
 - ▶ Compatibilité avec Ext2
 - ▶ Simplicité
- ▶ Inconvénients
 - ▶ Basé sur une architecture ancienne
 - ▶ Exemple : le nombre d'i-nœuds est fixé au moment du formatage

NTFS

- ▶ Système inspiré des systèmes à base d'i-nœuds
- ▶ Système de fichiers journalisé
- ▶ Structure



- ▶ Master Boot Record
- ▶ Master File Table ~ table des i-nœuds
- ▶ Structure d'une entrée de la MFT



- ▶ Traitement spécial des petits fichiers

Persistance des Fichiers

Erreurs physiques

Erreurs physiques

- ▶ Fautes possibles
 - ▶ mauvais état du disque \Rightarrow secteurs endommagés
 - ▶ faute de disque
 - ▶ destruction accidentelle de fichiers par des utilisateurs

- ▶ Gestion des blocs endommagés
 - ▶ Détection lors du formatage
 - ▷ Marquage des blocs défectueux
 - ▶ Détection en écriture
 - ▷ réécriture dans un nouveau bloc (perte de performance)
 - ▶ Détection en lecture
 - ▷ récupération éventuelle s'il existe une copie de sauvegarde assez récente

Sauvegardes et Redondance

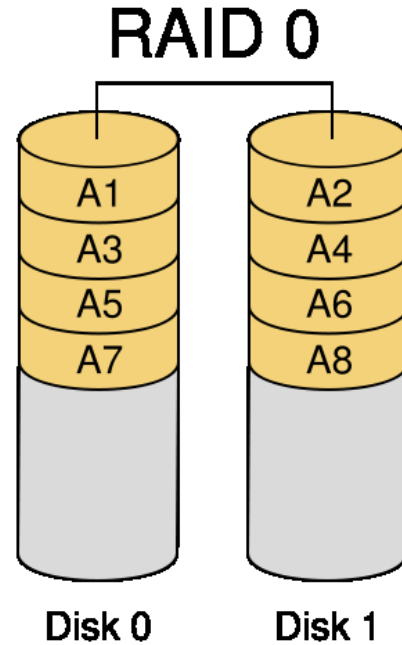
- ▶ Copies de sauvegarde (*Backup*)
 - ▶ Tous les fichiers sont copiés régulièrement sur un autre support

- ▶ Copies incrémentales
 - ▶ Le système gère des dates de sauvegarde
 - ▶ Sauvegarde des modifications

- ▶ Tableau de disques à contrôle de parité
 - ▶ RAID = Redundant Array of Inexpensive Disks
 - ▶ RAID = Redundant Array of Independent Disks

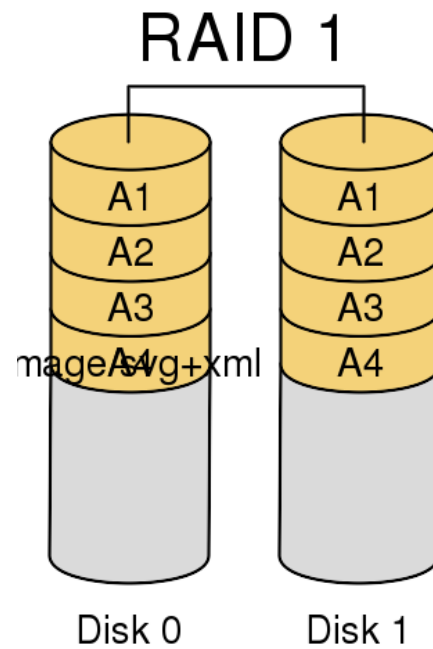
RAID

- ▶ RAID 0 : Entrelacement (stripping)
- ▶ Les blocs d'un fichier sont répartis sur plusieurs disques
- ▶ Améliore les performances
- ▶ N'augmente pas la fiabilité



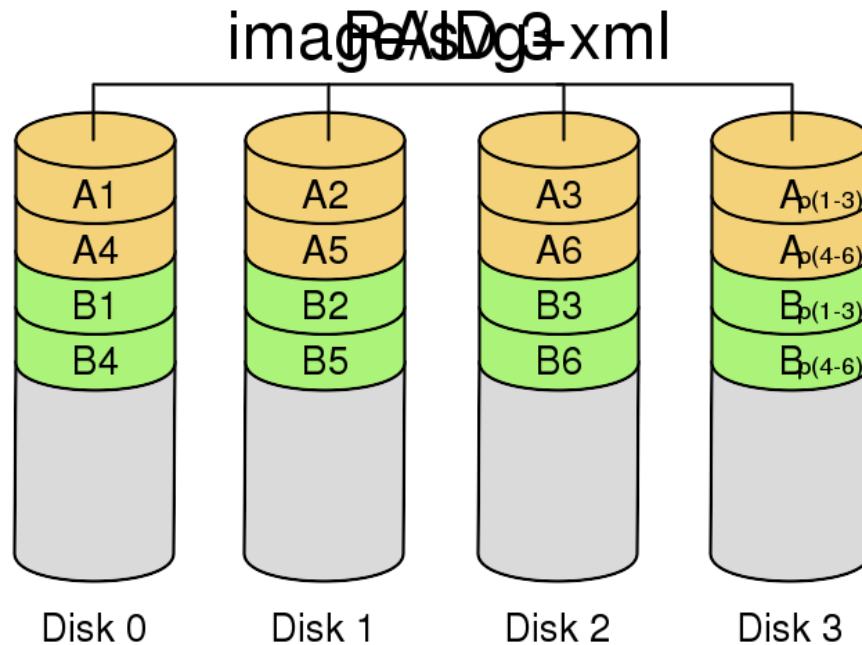
RAID

- ▶ RAID 1 : miroir
 - ▶ Augmente la fiabilité
 - ▶ 2 fois plus de disques



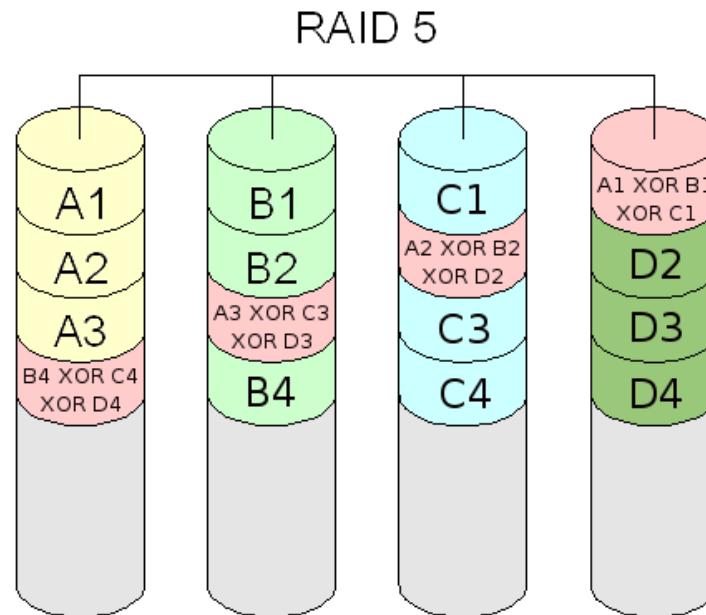
RAID

- ▶ RAID 3 : RAID 0 + un disque de code correcteur d'erreur
 - ▶ Augmente la fiabilité à un coût inférieur (tolère une panne)
 - ▶ Surcharge du disque de parité
- ▶ RAID 4 : RAID 3 par blocs



RAID

- ▶ RAID 5 : RAID 4 mais avec code correcteur réparti sur tous les disques
 - ▶ améliore les performances
 - ▶ meilleure disponibilité en cas de panne
- ▶ RAID 6 : RAID 5 avec plus de redondance (tolère 2 pannes)



Autres types de systèmes de fichiers

Le système FAT (File Allocation Table) : MS-DOS

- ▶ Système de fichier qui **n'est pas** à base d'i-nœud
- ▶ Structure logique du disque

init	MBR	FAT	FAT Miroir	/	Autres fichiers
------	-----	-----	------------	---	-----------------

- ▶ MBR (*Master Boot Record*)
 - ▷ nombre d'octets par secteurs, nombre de secteurs par cluster, nombre de secteurs sur le disque, etc.
- ▶ Table d'allocation des clusters (FAT)
 - ▷ chaque case représente un cluster (clusters = bloc mais généralement plus gros : jusqu'à 128 Ko)
- ▶ Le répertoire racine
- ▶ Les clusters de données

Table d'allocation des clusters

- ▶ Chaque case peut contenir une valeur codée sur 16 ou 32 bits
- ▶ La signification de ces valeurs est :
 - ▶ 0 : cluster vide
 - ▶ 1 : cluster réservé au système
 - ▶ 0xFFF7 : cluster défectueux
 - ▶ Une autre valeur n :
 - ▷ le cluster est occupé par un fichier
 - ▷ la suite du fichier est contenu dans le cluster n
 - ▷ si n est compris entre 0xFFF8 et 0xFFFF, c'est le dernier cluster du fichier
- ▶ Un fichier est une liste chaînée de clusters

Table d'allocation des clusters : exemple

The diagram shows a 4x16 grid representing a cluster allocation table. Arrows indicate pointers between specific cells:

- Row 1: Arrows from 0001 to 0002, 0002 to 0003, 0003 to 0004, 0004 to FFF8, 000A to 000B, 000B to 000C, and 000C to FFF8.
- Row 2: An arrow from the 0 in column 3 to the 0 in column 4.
- Row 3: An arrow from the 0 in column 3 to the 0 in column 4.
- Row 4: An arrow from the 0 in column 3 to the 0 in column 4.

0001	0002	0003	0004	FFF8	0	0	0	0	000A	000B	000C	FFF8	0	0	0
FFFF	FFF8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0023	0028	0039	0	0	0024	0027	0	0	0	FFF7	FFF7	FFF7	FFF7
FFF7	FFF7	0	0	0	0	0	0	0	FFF8	0	0	0	0	0	0

Structure d'un répertoire

- ▶ Un répertoire est une liste d'entrées contenant :
 - ▶ Le nom du fichier (8 + 3 caractères)
 - ▶ Les attributs (lecture seule, fichier caché, sous-répertoire, etc.)
 - ▶ Les dates (création, accès, modification)
 - ▶ Numéro du premier cluster
 - ▶ Taille du fichier
- ▶ Contrairement aux systèmes de fichiers avec i-nœuds
 - ▶ Les meta-données sur les fichiers sont stockées au niveau des répertoires
 - ▶ Il ne peut pas y avoir de liens physiques

FAT : Avantages et inconvénients

- ▶ Avantages :
 - ▶ Simple et léger
 - ▶ Bien adapté à des petits disques (≤ 1 Go)
- ▶ Inconvénients :
 - ▶ Limitations fonctionnelles : nom de fichier, propriétaires, etc.
 - ▶ Mauvaises adaptations aux opérations courantes sur les fichiers : accès aléatoire, écriture en fin de fichier (problème accentué sur les gros fichiers)
 - ▶ Limitation sur la taille des disques (table d'allocation)
 - ▷ Remarque : dans les systèmes à i-nœuds il suffit d'un bit par bloc
 - ▶ Fragmentation interne (dernier cluster sous-utilisé)

procfs

- ▶ Informations gérées par le noyau
- ▶ Fichiers virtuels (pas des données stockées sur disque mais en mémoire centrale)
- ▶ Exemples de fichiers de /proc
 - ▶ `cpuinfo` : description du ou des processeurs
 - ▶ `loadavg` : charge du système
 - ▶ `meminfo` : état d'occupation de la mémoire
 - ▶ `uptime` : temps écoulé depuis le démarrage du système
 - ▶ etc.
- ▶ En plus de ces fichiers, /proc contient des répertoires comme :
 - ▶ `net` : informations sur le réseau
 - ▶ `scsi` : informations sur les gestionnaires de périphériques SCSI
 - ▶ un répertoire par processus existant dans le système
 - ▷ `cmdline`, `cwd`, `environ`, `fd`, etc.