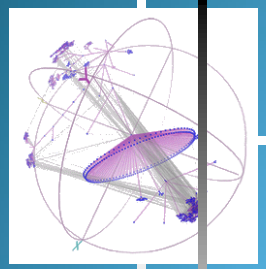
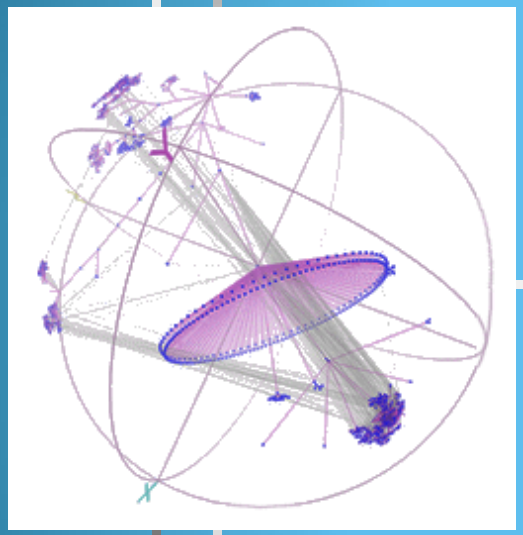
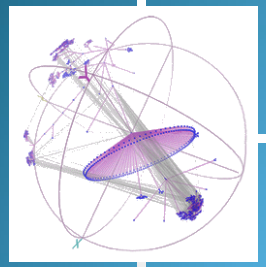




Chapitre 4
Le système de Gestion de Fichiers

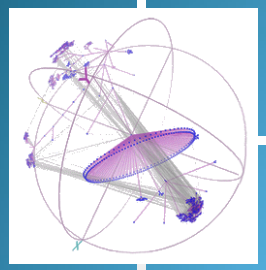
1. **Systèmes d'entrée/sortie**
2. **Systèmes de fichiers**
3. **Structure de mémoire de masse (disques)**

Systèmes d'entrée/sortie



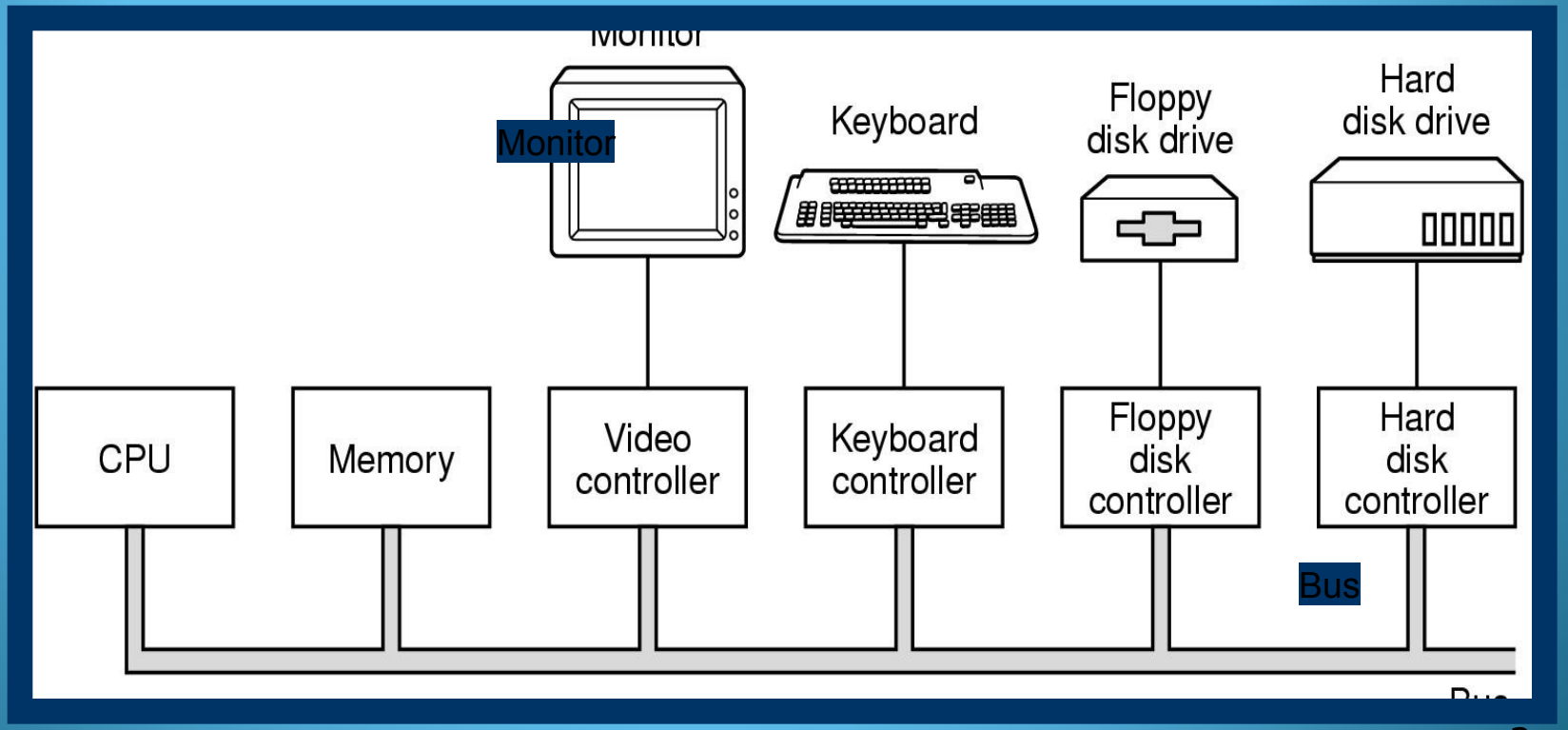
Concepts importants :

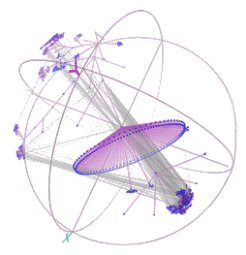
- Matériel E/S
- Communication entre UCT et contrôleurs périphériques
- DMA
- Pilotes et contrôleurs de périfs
- Sous-système du noyau pour E/S
  - Tamponnage, cache, spoule



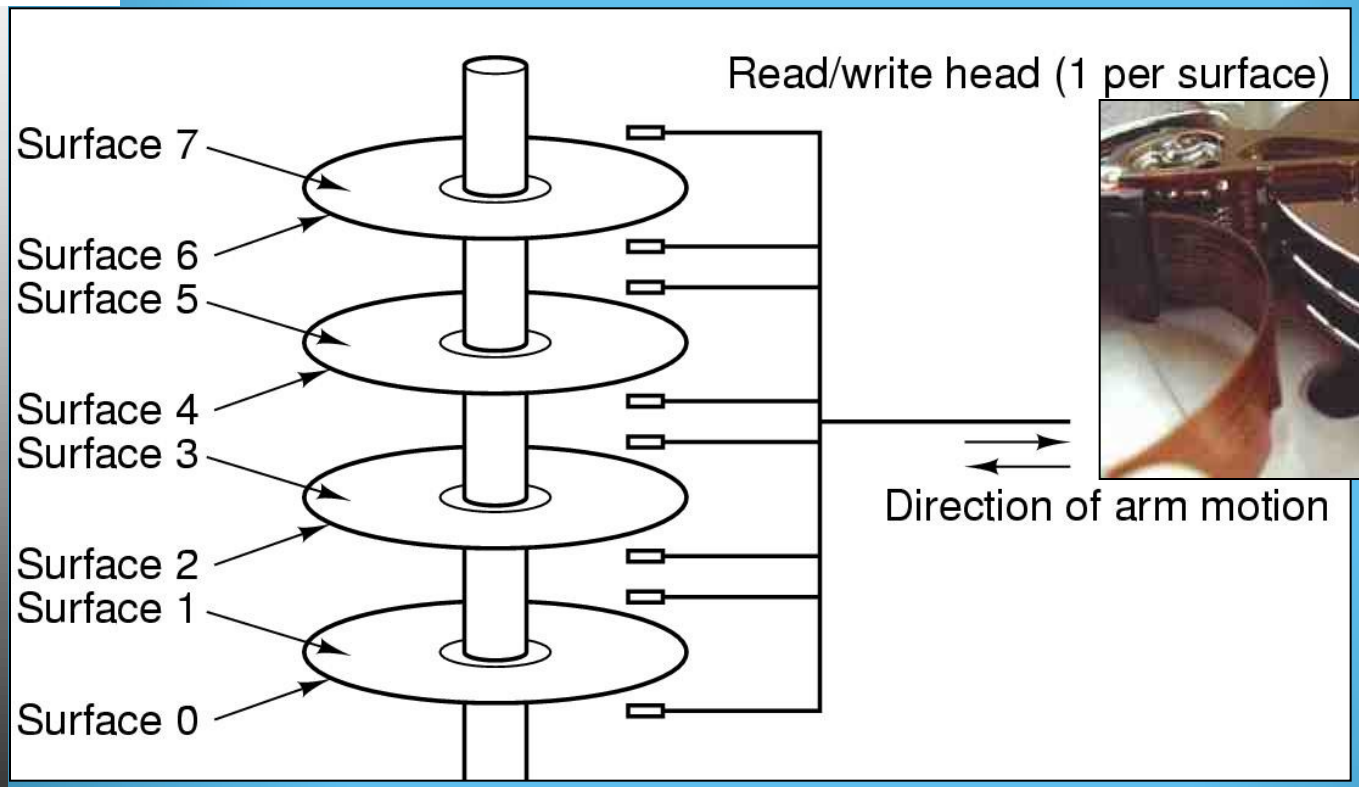
Contrôleurs de périphériques

- On se rappel: Les périphériques d'E/S ont typiquement une composante mécanique et une composante électronique
  - La partie électronique est le contrôleur



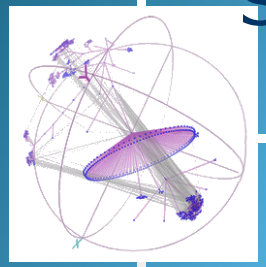


Revue des disques magnétiques



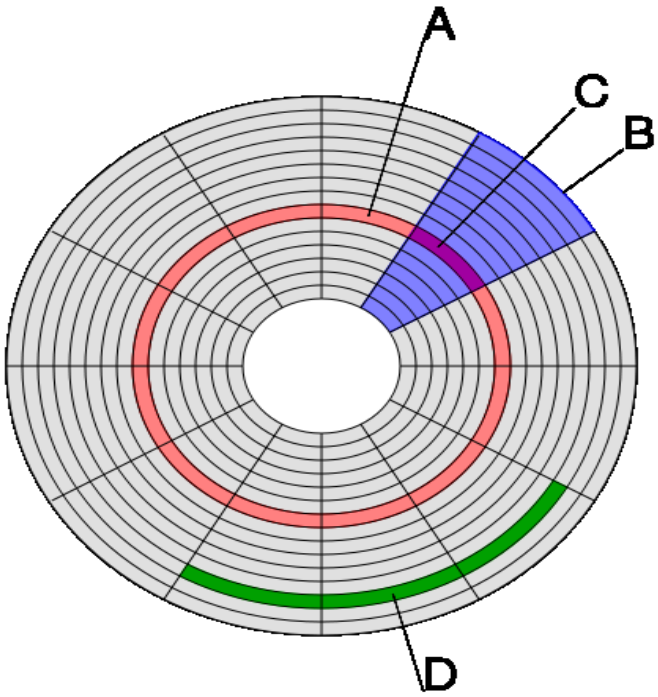
- Les disques sont organisés en cylindres, pistes et secteurs

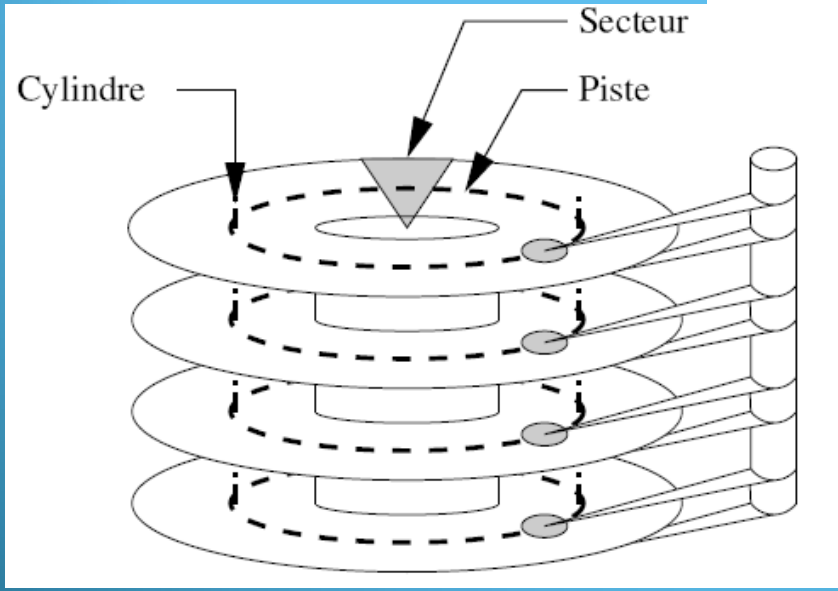
Support physique de codage de l'information

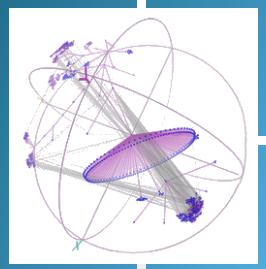


- Disque dur

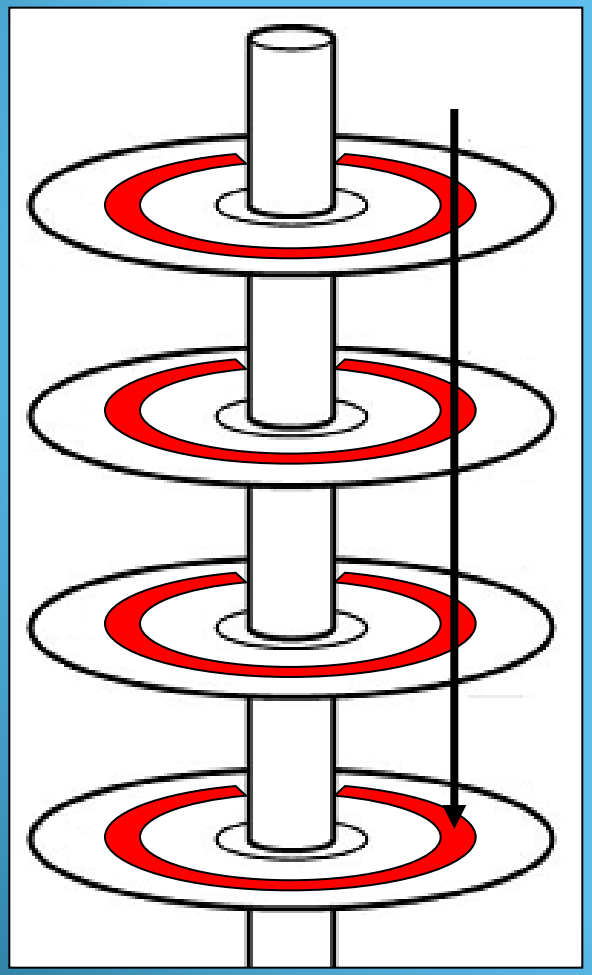
(A) Piste
(B) Secteur géométrique
(C) secteur d'une piste
(D) cluster



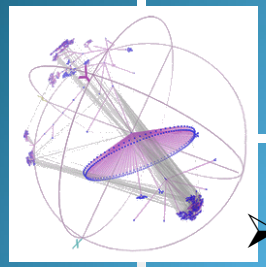




Revue des disques magnétiques

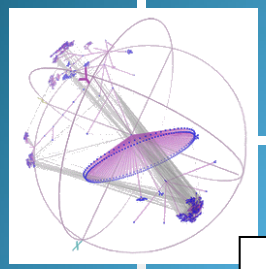


- Toutes les pistes pour une position donnée du bras forment un cylindre.
  - Donc le nombre de cylindre est égale au nombre de piste par côté de plateau
  - La location sur un disque est spécifié par (cylindre, tête, secteur)

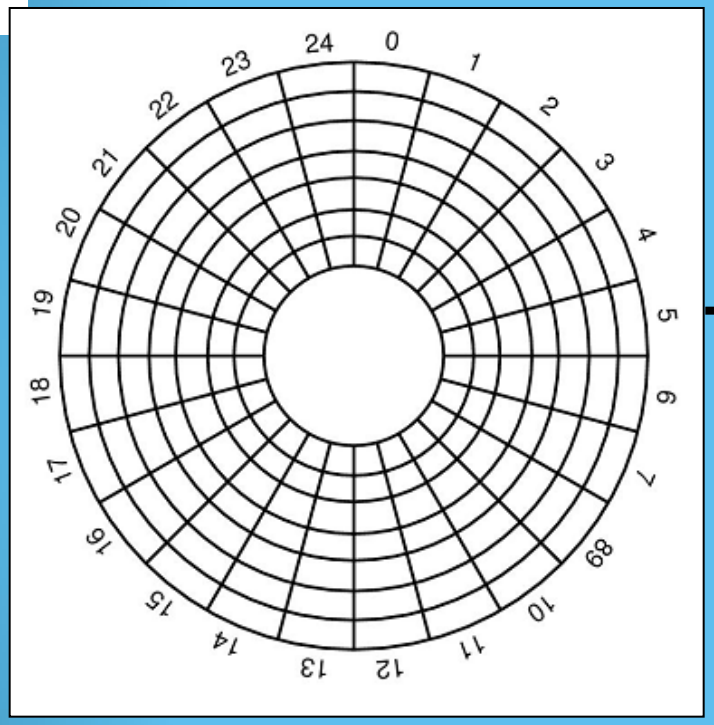


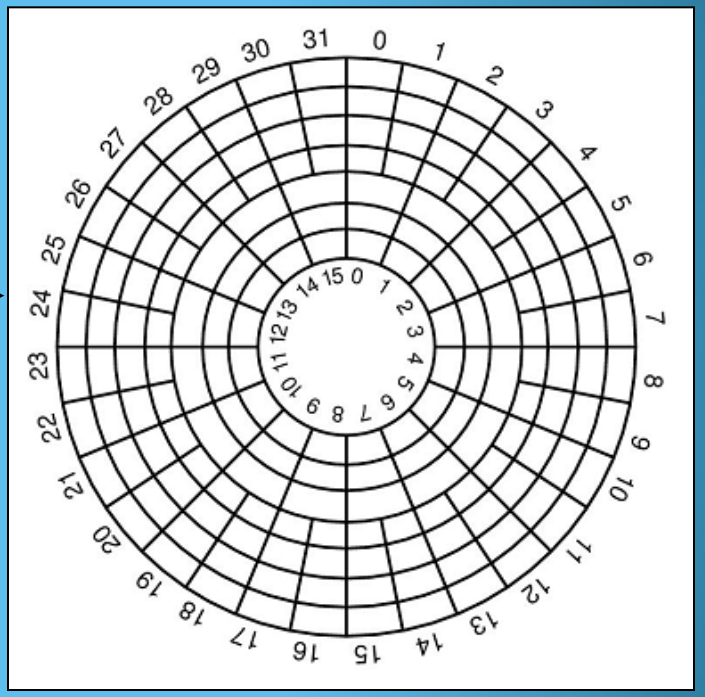
Disques magnétiques

- Sur les disques plus vieux, le nombre de secteurs par piste étaient constant pour tout les cylindres
  - Ceci gaspille de l'espace de stockage potentiel sur les cylindres du disque
  - Les disques modernes réduisent le nombre de secteurs par piste vers l'intérieur du disque. Certains par paliers (plusieurs pistes) certains de façon linéaire
  - Souvent, les contrôleurs modernes attachés aux disques présentent une géométrie virtuelle au système d'exploitation, mais ont un arrangement physique beaucoup plus compliqué...



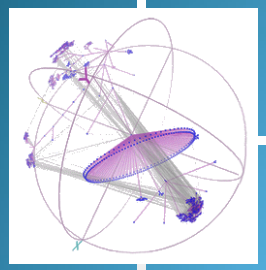
Disques magnétiques



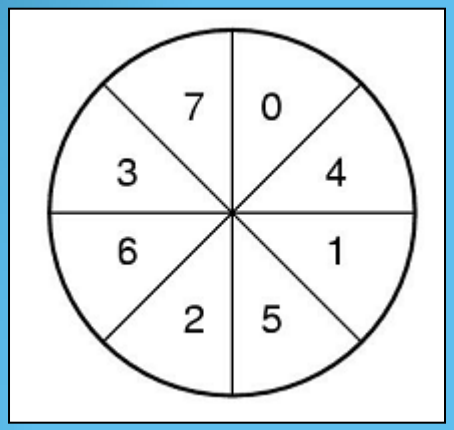


Organisation virtuelle

Géométrie Physique

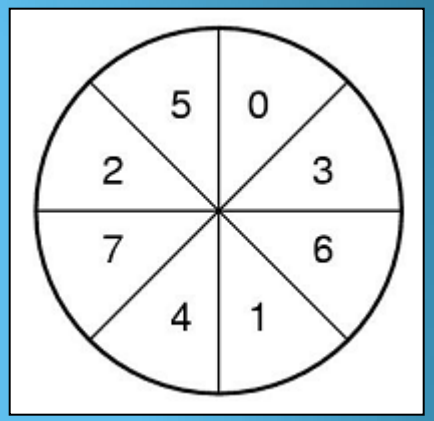


Entrelacement

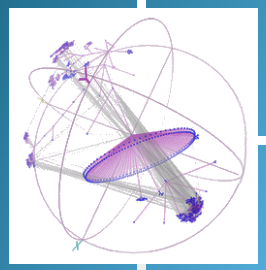


Entrelacement simple

Entrelacement double

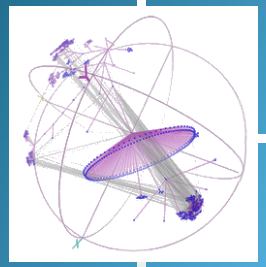


- L’entrelacement permet au disque de tourner et de passer n secteurs et de prendre le prochain secteur virtuellement contiguë pendant que les données sont transférés du contrôleur vers la mémoire



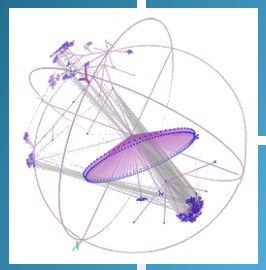
Les Périphériques

- Périphériques blocs ou caractères
  - Périphériques blocs: disques, rubans…
    - Commandes: read, write, seek
    - Accès brut (raw) ou à travers système fichiers
    - Accès représenté en mémoire (memory-mapped)
      - Semblable au concept de mémoire virtuelle ou cache:
        - » une certaine partie du contenu du périphérique est stocké en mémoire principale(cache), donc quand un programme fait une lecture de disque, ceci pourrait être une lecture de mémoire principale
  - Périphériques par caractère (écran)
  - Get, put traitent des caractères
  - Librairies au dessus peuvent permettre édition de lignes, etc.

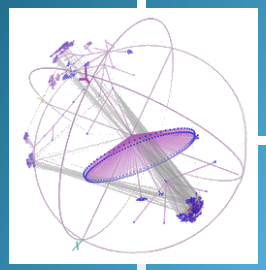


Pilotes de périphériques

- Pilotes de périphériques
  - Chaque périphérique d'E/S attaché à l'ordinateur requiert du code spécifique pour faire l'interface entre le matériel et le SE. Ce code s'appel pilote de périphérique
    - Ceci est parce qu'au niveau du matériel, les périphériques sont radicalement différents les uns des autres
    - Parfois un pilote va prendre soins d'une classe de périphériques qui sont proche ex.: un nombre de souris
  - Les pilotes de périphériques sont normalement produit par le manufacturier du périphérique pour les SEs populaires

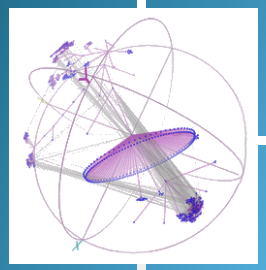


- Pilotes de périphériques
  - Typiquement les pilotes sont dans le noyau pour qu'ils puissent avoir accès au registres de contrôle du périphérique
    - Ce n'est pas un requis. Vous pourriez avoir un pilote dans l'espace utilisateur et faire des appels de systèmes pour communiquer avec les registres. Par contre la pratique courante est d'avoir les pilotes dans le noyau.
  - Étant donné que c'est la méthode habituel d'implémenter les pilotes, l'architecture normale est de mettre les pilotes 'en bas' du SE



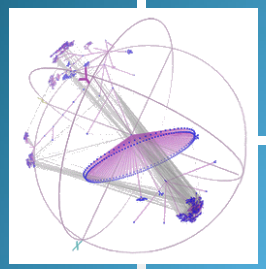
Pilotes de périphériques

- Que font les pilotes de périphériques?
  - Ils acceptent les commandes abstraites de lecture/écriture de la couche supérieure
  - Fonctions assorties:
    - Initialise le périphérique
    - Gère la puissance – Arrête un disque de tourner, ferme un écran, ferme une caméra, etc.



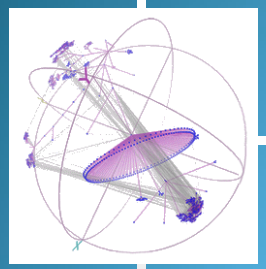
Pilotes de périphériques

- Qu'est-ce que un pilote fait sur une lecture/écriture?
  - Vérifie les paramètres d'entrée & retourne les erreurs
  - Converti les commandes ***abstraites*** (lit le secteur) en commandes ***physiques*** (tête, traque, secteur, et cylindre)
  - Met les demandes dans une queue si le périphérique est occupé
  - Amène le périphérique en état de fonctionnement si requis
    - Monter la vitesse du moteur, température, etc.
  - Contrôle le périphérique en envoyant des commandes par les registres de contrôle



Contrôleurs de périphériques

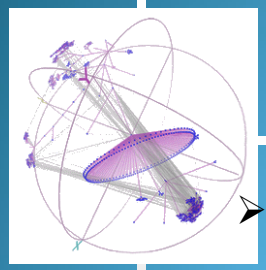
- Sur un PC, le contrôleur de périphérique est habituellement sur un circuit imprimé
  - Il peut être intégré sur la carte mère
- Le job du contrôleur est de convertir un flot de série de bits en octets ou en blocs d'octets et de faire les conversions et corrections
  - En fin de compte touts les périphériques traitent des bits. C'est le contrôleur qui groupe ou dégroupe ces bits



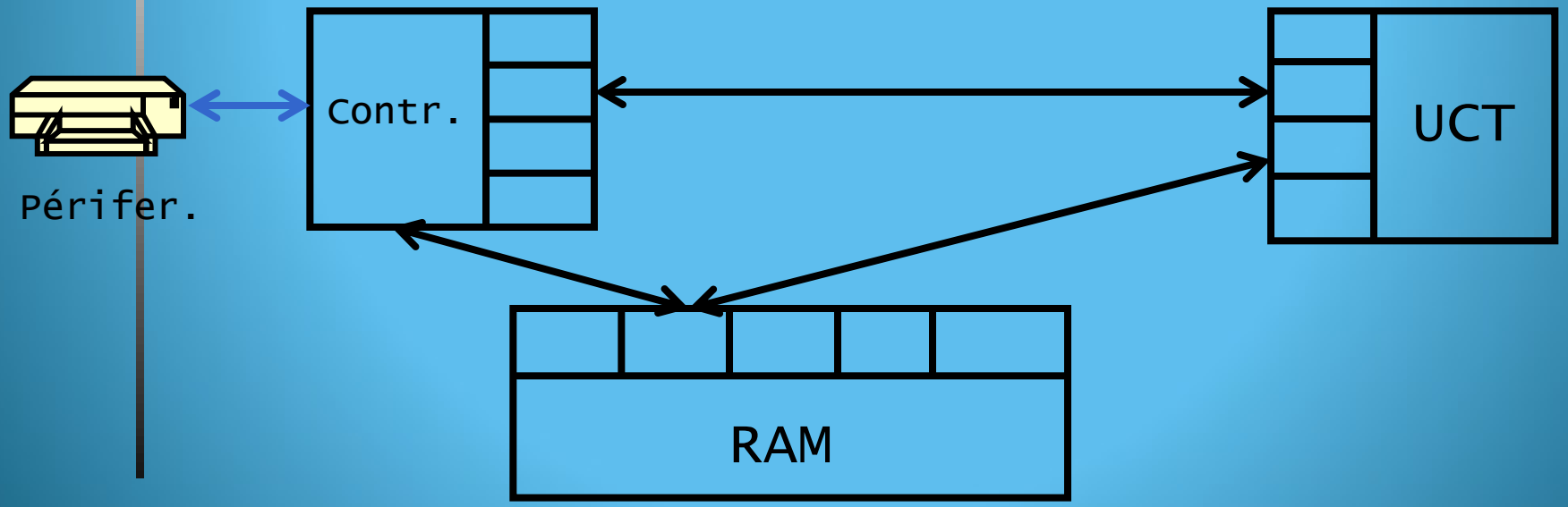
Registres de contrôle

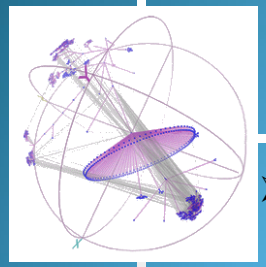
- Chaque contrôleur possède quelques registres qui servent à la communication avec le processeur
  - Le système d'exploitation peut commander les périphériques de fournir ou accepter des données, de s'éteindre, etc.
  - Le SE peut aussi lire certains registres pour déterminer l'état du périphérique
- Les contrôleurs ont *typiquement* des tampons de données que le SE peut lire et écrire
  - Important pour ne pas perdre de données
  - Parfois utilisé comme 'partie' du périphérique, ie: RAM vidéo dans laquelle les programmes ou le SE peuvent écrire

Communication entre UCT et contrôleurs périphériques



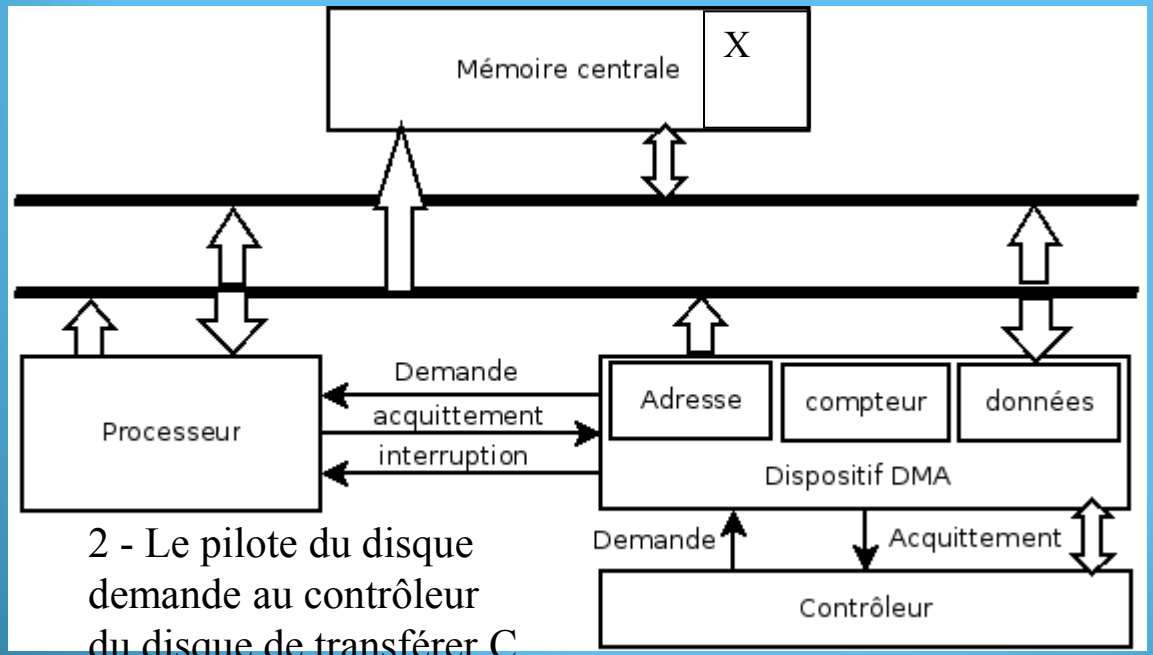
- Deux techniques de base:
  - UCT et contrôleurs communiquent directement par des registres
  - UCT et contrôleurs communiquent par des zones de mémoire centrale
  - Combinaisons de ces deux techniques





Accès direct en mémoire (DMA)

- Dans les systèmes sans DMA, l'UCT est impliquée dans le transfert de chaque octet
- DMA est utile pour exclure l'implication de l'UCT surtout pour des E/S volumineuses
- Demande un contrôleur spécial a accès direct à la mémoire centrale MMU (Memory Management Unit)



1- CPU demande au pilote du périphérique (HD)

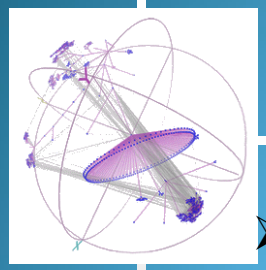
2 - Le pilote du disque demande au contrôleur du disque de transférer C octets du disque vers le buffer à l'adresse x

3 - Le contrôleur du disque initie le transfert DMA

4 - Le contrôleur du disque envoie chaque octet au contrôleur du DMA

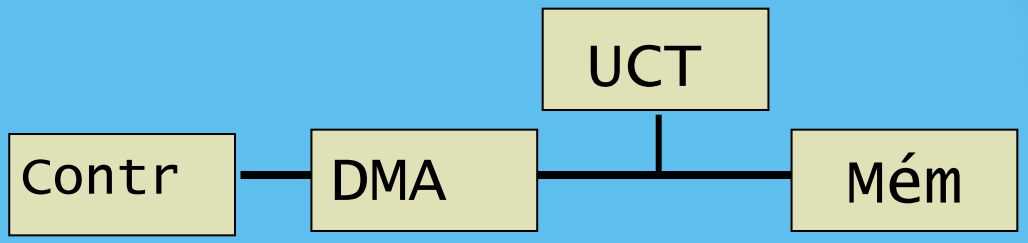
5 - Le contrôleur DMA transfert les octets au buffer x en augmentant l'adresse x et décrémentant le compteur c

6 - Lorsque C=0 DMA envoie une interruption pour signaler la fin du transfert



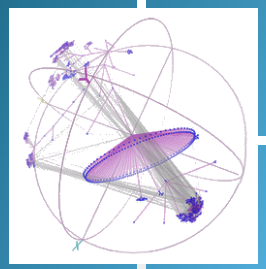
Vol de cycles

- Le DMA ralentit encore le traitement d'UCT car quand le DMA utilise le bus mémoire, l'UCT ne peut pas s'en servir



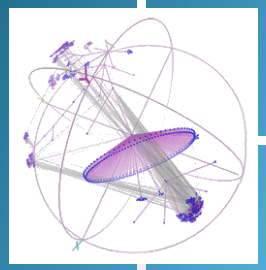
- Mais beaucoup moins que sans DMA, quand l'UCT doit s'occuper de gérer le transfert

Mémoire <-> Périphérique



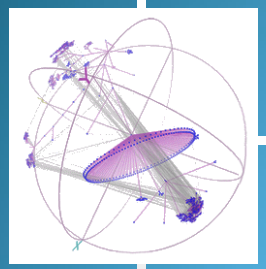
Sous-système E/S du noyau

- Fonctionnalités:
  - Mise en tampon
  - Mise en cache
  - Mise en attente et réservation de périphérique spoule
  - Gestion des erreurs
  - Ordonnancement E/S
    - Optimiser l'ordre dans lequel les E/S sont exécutées



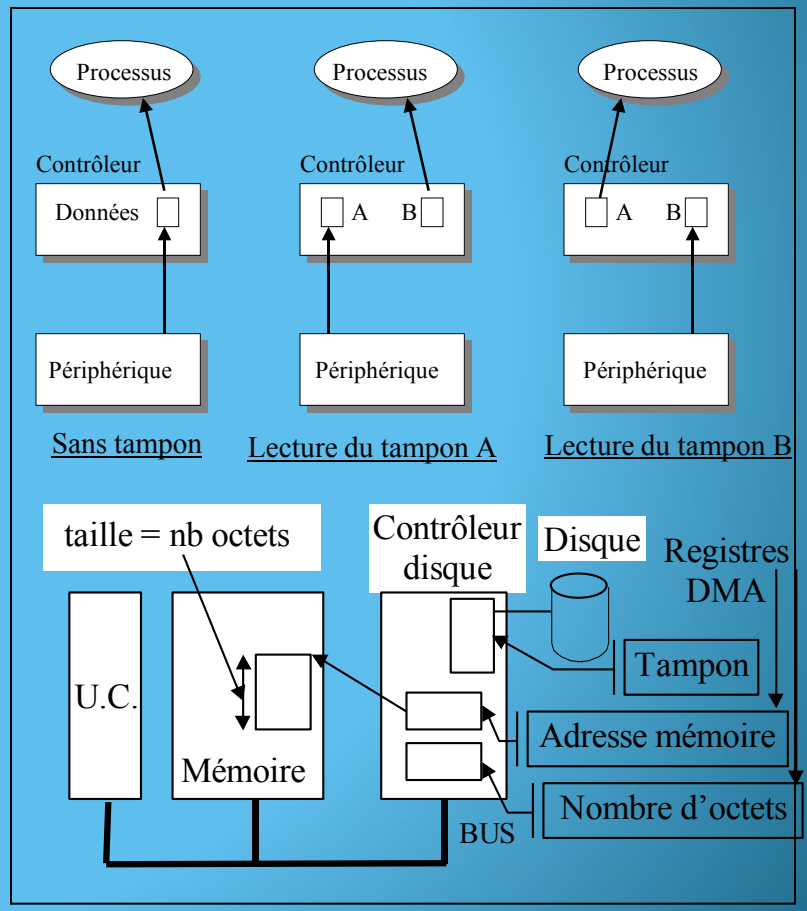
Mise en tampon

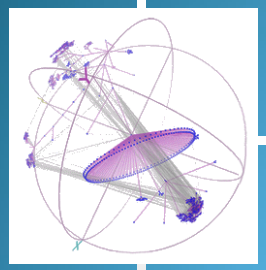
- Double tamponnage:
  - P.ex. en sortie: un processus écrit le prochain enregistrement sur un tampon en mémoire tant que l'enregistrement précédent est en train d'être écrit
  - Permet superposition traitement E/S



Principes de gestion de périphériques
Tamponnage des données

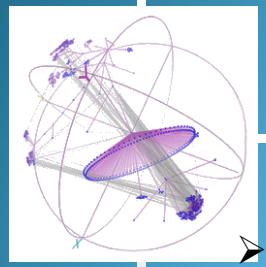
- Principes.
  - Simultanéité des opérations d'entrées et de sorties avec les opérations de calcul.
  - Le contrôleur de périphérique inclue plusieurs registres de données.
  - Pendant que l'UCT accède à un registre, le contôleur peut accéder à un autre registre.





Mise en cache

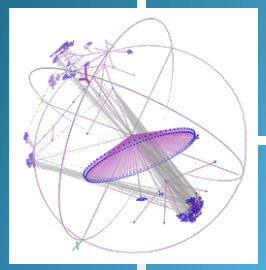
- Quelques éléments couramment utilisés d'une mémoire secondaire sont gardés en mémoire centrale
- Donc quand un processus exécute une E/S, celle-ci pourrait ne pas être une E/S réelle:
  - Elle pourrait être un transfert en mémoire, une simple mise à jour d'un pointeur, etc.



Mise en attente et réservation de périphérique: spool

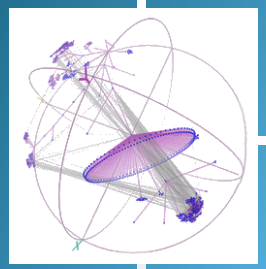
- Spool ou Spooling est un mécanisme par lequel des travaux à faire sont stockés dans un fichier, pour être ordonnancés plus tard

- Pour optimiser l'utilisation des périphériques lents, le SE pourrait diriger à un stockage temporaire les données destinés au périphérique (ou provenant d'elle)
  - P.ex. chaque fois qu'un programmeur fait une impression, les données pourraient au lieu être envoyées à un disque, pour être imprimées dans leur ordre de priorité
  - Aussi les données en provenance d'un lecteur optique pourraient être stockées pour traitement plus tard



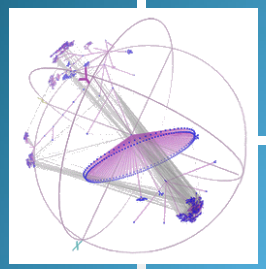
Gestion des erreurs

- Exemples d'erreurs à être traités par le SE:
  - Erreurs de lecture/écriture, protection, périph non-disponible
- Les erreurs retournent un code 'raison'
- Traitement différent dans les différents cas…



Gestion de requêtes E/S

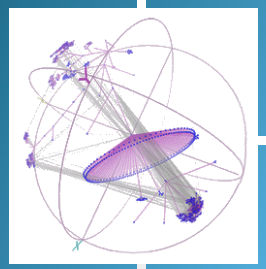
- P. ex. lecture d'un fichier de disque
  - Déterminer où se trouve le fichier
  - Traduire le nom du fichier en nom de périphérique et location dans périphérique
  - Lire physiquement le fichier dans le tampon
  - Rendre les données disponibles au processus
  - Retourner au processus



Systèmes de fichiers

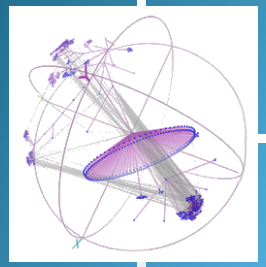
Concepts importants :

- Systèmes fichiers
- Méthodes d'accès
- Structures Répertoires
- Structures de systèmes fichiers
- Méthodes d'allocation
- Gestion de l'espace libre
- Implémentation de répertoires



Le concept de fichier

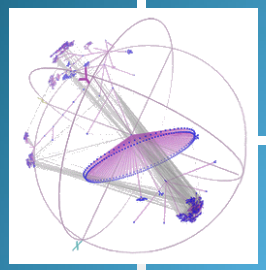
- Un fichier est un ensemble nommé d'informations reliées qui est stocké sur tout type de système de stockage
  - Programmes, données, etc.
- Le système gère la correspondance entre fichiers et périphériques de stockage matériel (disques, bandes magnétiques, disquettes, CDs, DVDs, etc.)



Attributs d'un fichier

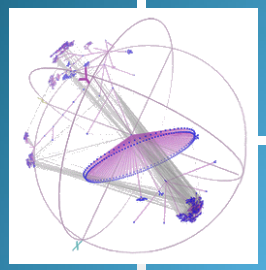
- Nom : sous forme lisible.
- Identifiant : généralement un entier.
- Type : exécutable, image, etc.
- Emplacement : pointeur vers le périphérique et la position sur celui-ci.
- Taille : réelle et éventuellement maximum possible.
- Protections : lecture, écriture et exécution.
- Heure, date et utilisateur : pour la création, la dernière modification et dernière utilisation.

- Ces informations sont stockées dans la structure en répertoires du disque.



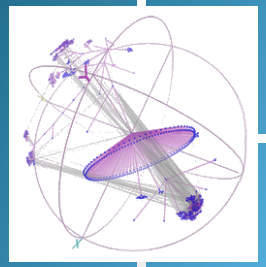
Opérations sur les fichiers

- Création
  - Trouver l'espace dans le système de fichier
  - Ajouter les entrées dans les répertoires
- Écriture : à la position courante
- Lecture : à la position courante
- Repositionnement dans le fichier : modifier la position courante
- Suppression : supprimer l'entrée et libérer l'espace disque
- Vider : mettre la taille à 0 (libération de l'espace) dans modifier les autres attributs



Implémentation

- Les fichiers doivent être ouvert avant d'effectuer lectures et écritures et fermés ensuite.
  - Une seule recherche du fichier dans le répertoire
- Open(filename) – recherche le fichier dans le répertoire et copie l'entrée en mémoire dans une table des fichiers ouverts. Retourne un pointeur vers cette entrée.
- Unix: write(fd, data, bytes), read(fd, data, bytes)
  - fd – index dans la table des fichiers ouverts
- Close (F) – déplace le contenu de l'entrée F dans la structure du répertoire et supprime F de la table des fichiers ouverts

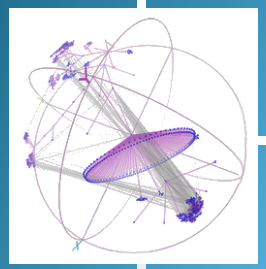


Implémentation

- Dans un système multi-utilisateurs, plusieurs processus peuvent ouvrir simultanément le même fichier.

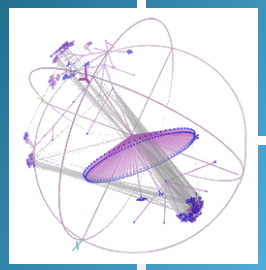
Deux tables pour les fichiers ouverts :

- Table globale : informations sur les fichiers (nom, emplacement, dates d'accès, taille, etc.)
  - Compteur du nombre d'ouverture pour chaque fichier : chaque fermeture décrémente ce compteur et l'entrée est supprimée quand il atteint 0.
- Table locale au processus
  - Chaque entrée contient la position dans le fichier, les droits d'accès et un pointeur vers une entrée dans la table globale.

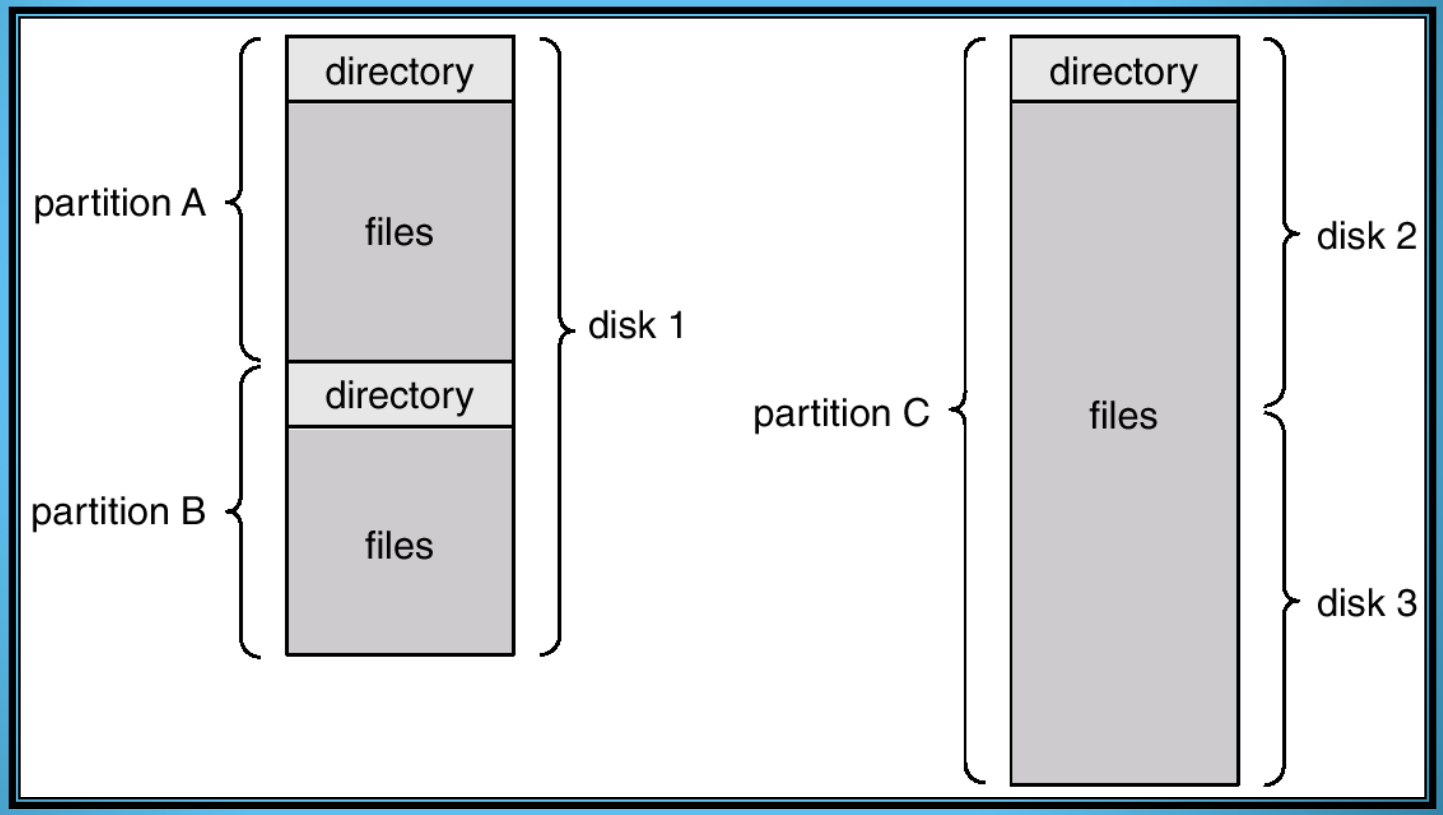


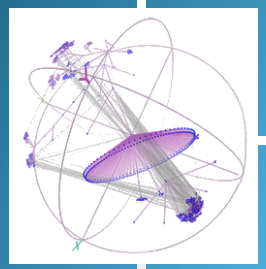
Structure en répertoires

- Les disques peuvent être partagés en partitions.
- Une partition peut s'étendre sur plusieurs disques
- Chaque partition à un système de fichiers propre :
  - Répertoires
  - Fichiers



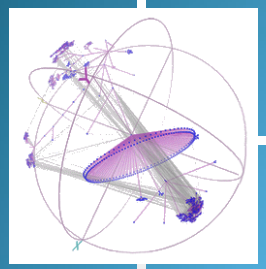
Organisation typique





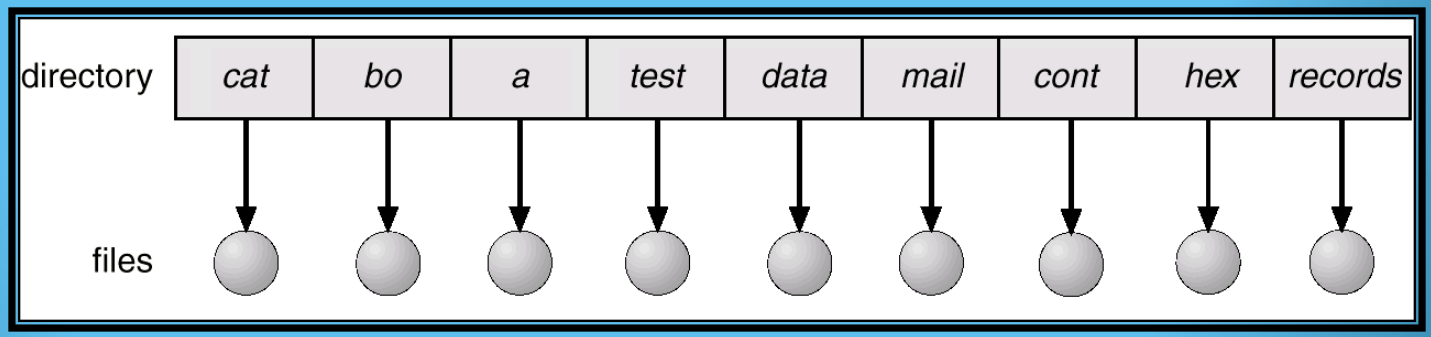
Agir sur les répertoires

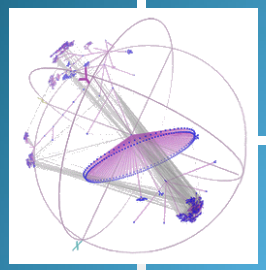
- Chercher un fichier
- Créer un fichier
- Supprimer un fichier
- Lister un répertoire
- Renommer un répertoire.
- Traverser le système de fichiers (visiter l'ensemble de la structure)



Répertoires à un niveau

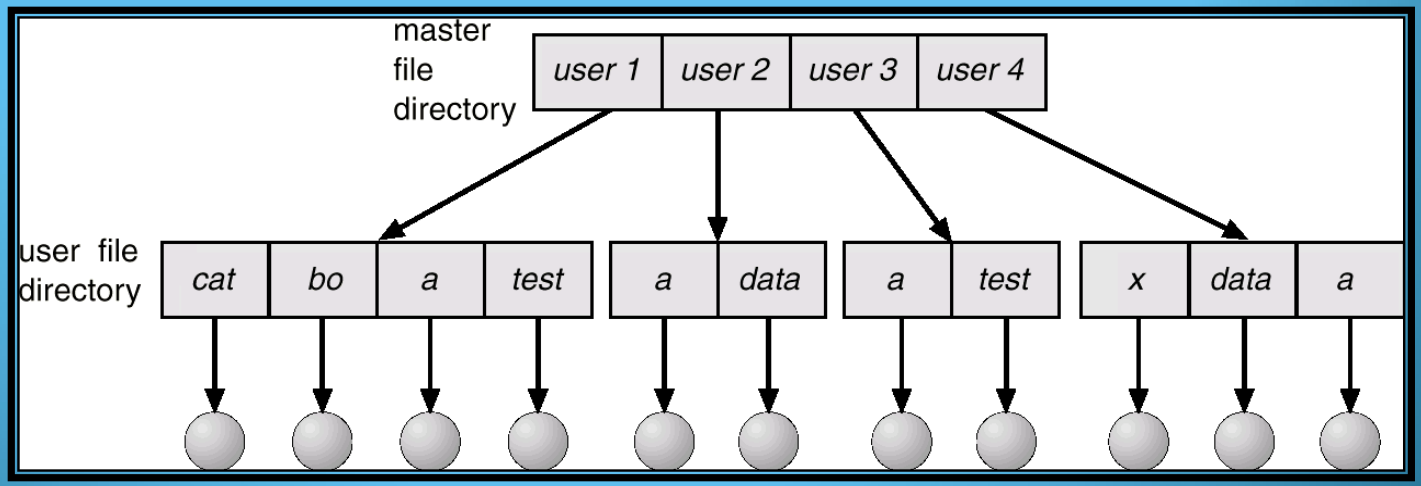
- Un seul répertoire pour tous les utilisateurs.
- Tous les fichiers doivent avoir un nom unique (dur à gérer s'il y a beaucoup d'utilisateurs ou de fichiers)

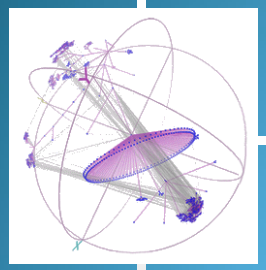




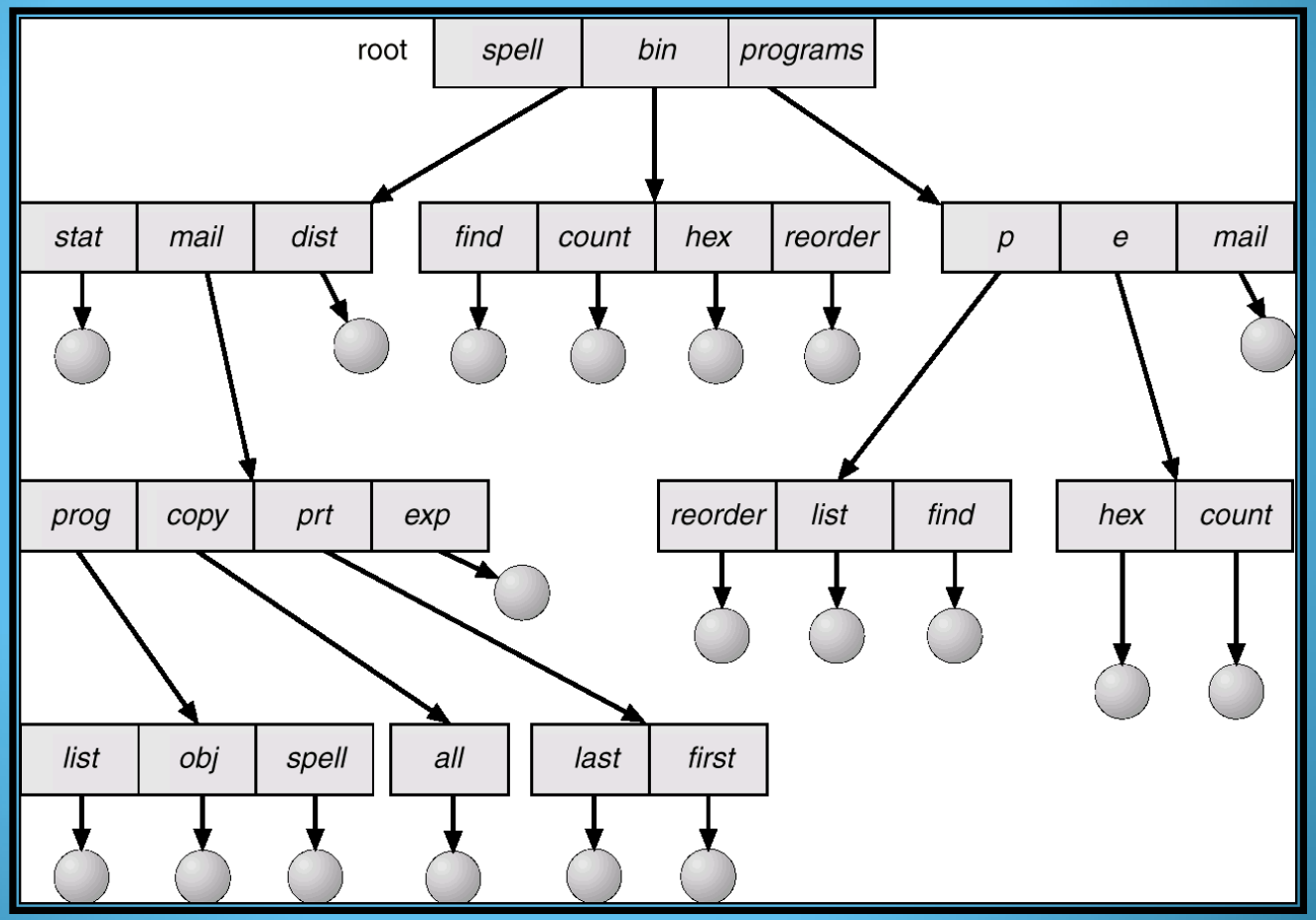
Répertoires à deux niveaux

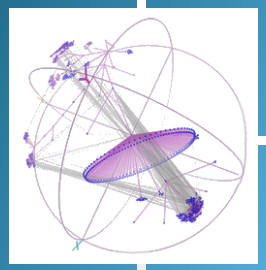
- Un répertoire par utilisateur
  - Noms de fichiers identiques pour des utilisateurs différents
  - Recherche efficace
- Chaque fichier est identifié par nom de l'utilisateur + nom du fichier





Répertoires arborescents

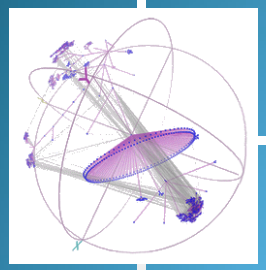




Répertoires arborescents

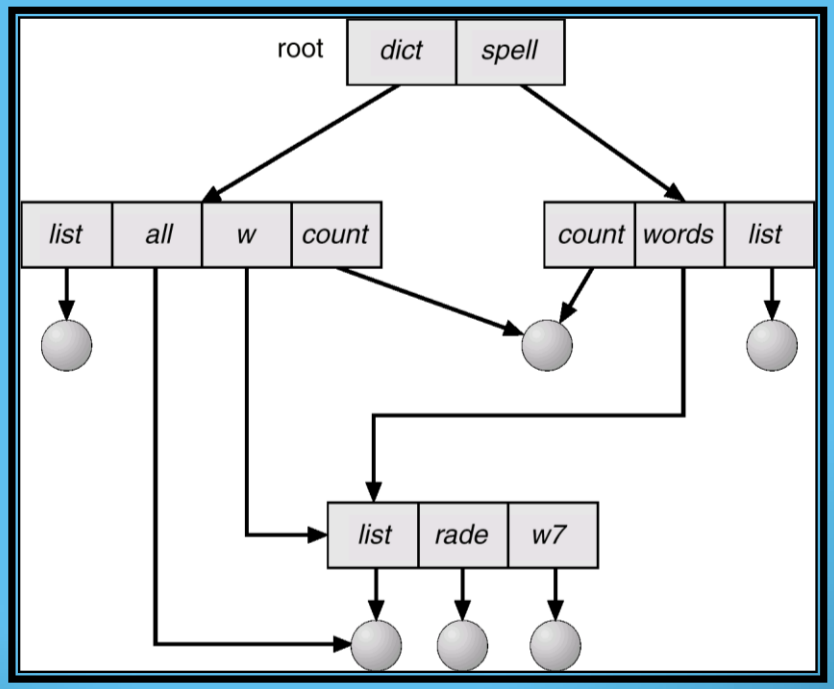
- Possibilité de créer des sous-repertoires
- Un répertoire contient des fichiers et des sous-repertoires

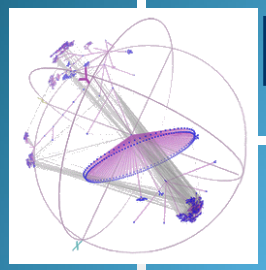
- Chaque fichier à un unique chemin (path name)
- Références absolues ou relatives :
  - À partir de la racine
  - Relatives au répertoire courant



Structure en graphe sans cycle

- Permet le partage de fichiers et de répertoires
- Un seul fichier existe réellement en cas de partage => les modifications faites par un utilisateur sont visibles par d'autres

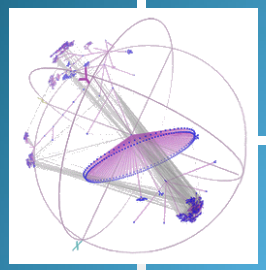




Montage du système de fichiers

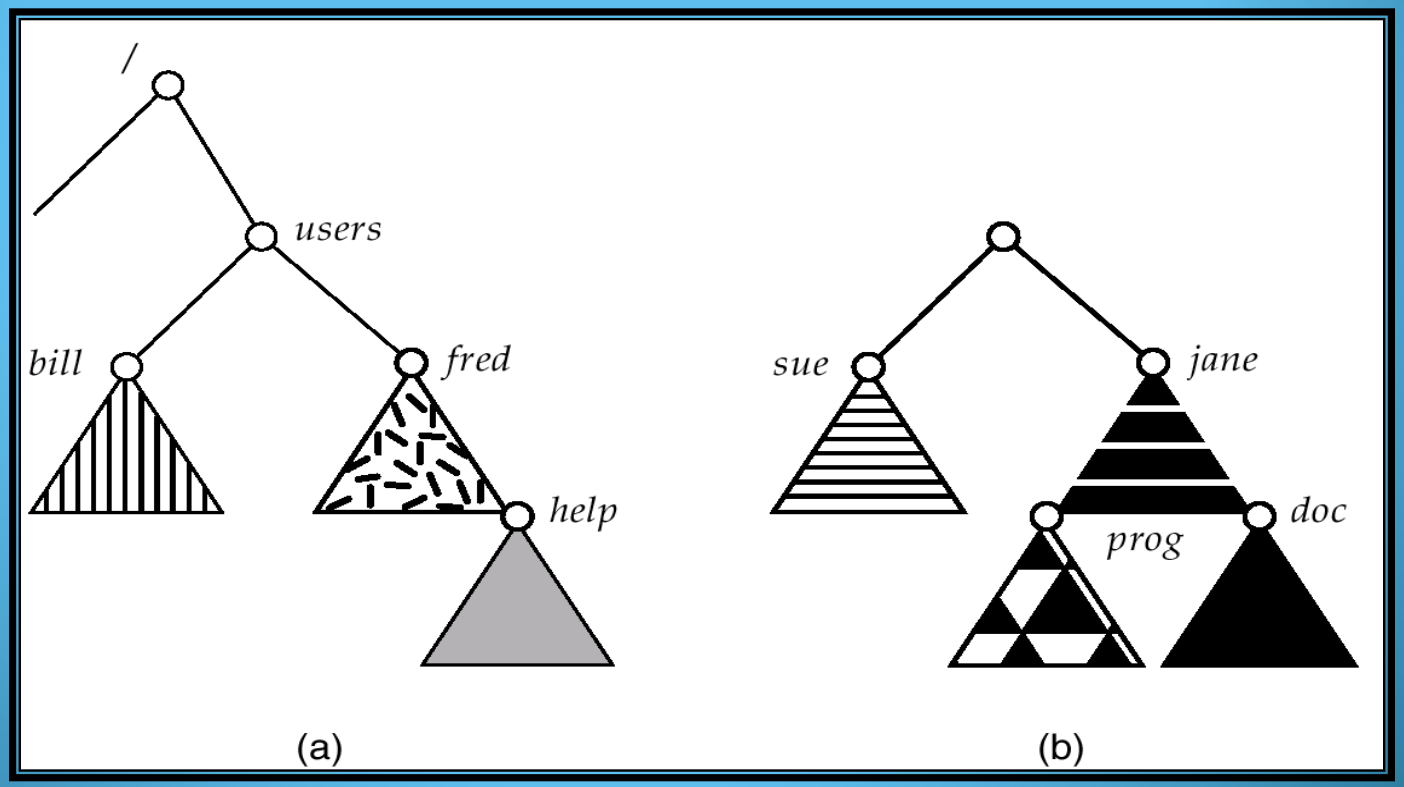
- Un système de fichiers doit être monté avant d'être accessible.
- Un système non monté est monté dans la structure à un point de montage.

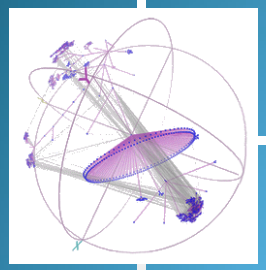
- UNIX: la commande ‘mount’ permet d'attacher le système de fichiers d'un périphérique dans l'arborescence.
- Windows: les périphériques et partitions ont une lettre pour les identifier.



(a) Partition existante.

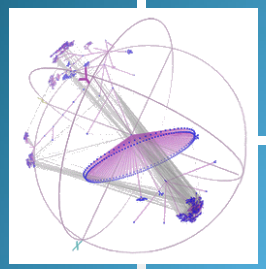
(b) Partition non montée.





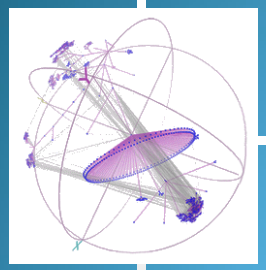
Partage à distance

- Dans un système distribué, les fichiers peuvent être partagé via un réseau :
  - Transfert manuel (FTP)
  - Système de fichiers distribué (DFS)
  - World Wide Web (WWW)
- Le modèle client-serveur
  - Le serveur contient les fichiers
  - Le client veut y accéder
  - Ils communiquent en utilisant un protocole d'échange



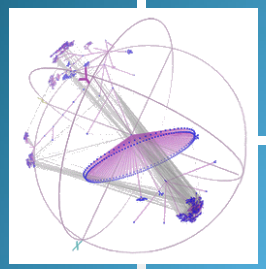
Protection

- Protection des fichiers contre les accès non désirés
- Le possesseur doit pouvoir contrôler :
  - Ce qui peut être fait
  - Par qui
- Types d'accès pouvant être contrôlés
  - Lecture
  - Écriture
  - Exécution
  - Concaténation (Append)
  - Suppression
  - List (voir les attributs)



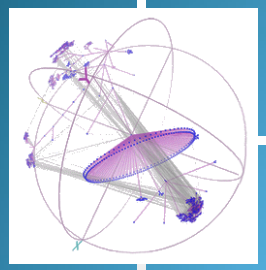
Liste de contrôles d'accès

- La plupart des approches font dépendre les accès de l'identité de l'utilisateur
- Plus généralement, chaque fichier/répertoire contient une liste de contrôle d'accès :
  - contenue dans l'entrée du répertoire
  - Spécifie le nom de l'utilisateur et les droits



Protection sous UNIX

- Trois classifications par fichier :
- Possesseur (Owner) : l'utilisateur qui a créé le fichier
  - Peut être modifié avec ‘chown’ (par root)
- Groupe (Group) : un ensemble d'utilisateurs avec les même droits d'accès
  - Créé par root
  - Peut être modifié par root avec ‘chgrp’
- Autres (Universe) : tous les autres

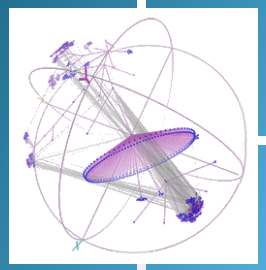


Protection sous UNIX (2)

- On peut spécifier les droits en lecture, écriture et exécution pour le possesseur, le groupe et les autres :

	RWX
possesseur	1 1 0 = 6
groupe	1 1 0 = 6
autres	0 0 0 = 0

- La commande ‘chmod’ permet de changer les droits
  - Ex : chmod 660 cours.ppt

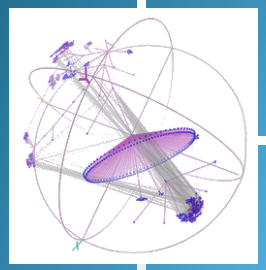


Protection sous UNIX (3)

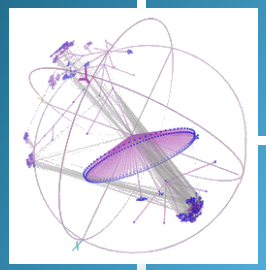
- Un exemple de répertoire :

```
$ ls -l
total 10547
drwxr-xr-x  1 ujgo7402 mkgroup-l-d       0 Feb  7 14:47 P2P_IPTPS
drwxr-xr-x  1 ujgo7402 mkgroup-l-d       0 Feb  6 19:27 PAPER
-rw-r--r--  1 ujgo7402 mkgroup-l-d 5117801 Feb  7 15:05 PAPER[1].tar.gz
-rw-r--r--  1 ujgo7402 mkgroup-l-d 5681444 Feb  7 15:04 iptps.tar.gz
```

- Répertoires :
  - R = lecture : lire le contenu du répertoire
  - W = écriture : créer ou supprimer des fichiers dans le répertoire
  - X = exécution : possibilité de se déplacer dans le répertoire

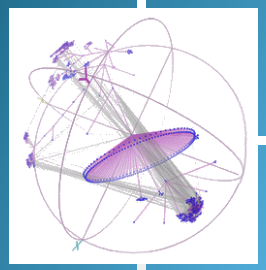


Implémentation des systèmes de fichiers



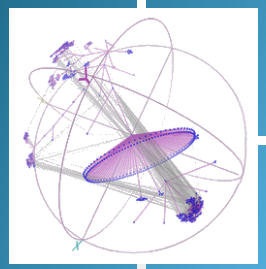
Introduction

- L'implémentation a pour but d'étudier :
  - Comment les fichiers sont stockés.
  - Comment les répertoires sont stockés.
  - Comment l'espace disque est géré.
  - Comment rendre le système de fichiers efficace.



Généralités

- Les systèmes de fichiers sont stockés sur disque.
- Les partitions contiennent des systèmes de fichiers indépendants.
- Plusieurs structures sont utilisées pour l'implémentation effective d'un système de fichiers. Ceci peut varier d'un système à l'autre.
- Le point crucial est de savoir en permanence quel fichier est associé à quel bloc sur le disque.

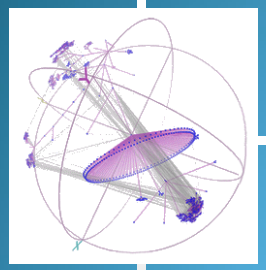


Blocs de contrôle

Sur le disque on trouve :

- Un bloc de contrôle de démarrage (boot control block)
  - contient les informations nécessaire pour démarrer le SE depuis cette partition.
  - Typiquement le premier bloc de la partition.
  - Sous UNIX (UFS) : boot block.
  - Avec NTFS : partition boot block.

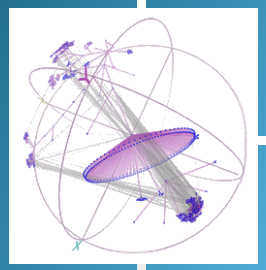
- Un bloc de contrôle de la partition
  - Contient des informations sur la partition (nombre de blocs, taille des blocs, nombre de blocs libres, pointeurs sur les blocs libres).
  - UFS : superblock.
  - NTFS : Master file table.



Bloc de contrôle d'un fichier

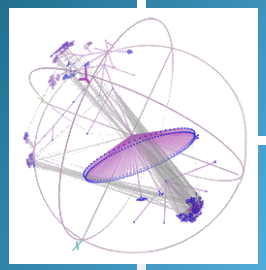
- Le bloc de contrôle d'un fichier (FCB) contient les informations sur le fichier (permissions, taille, etc.)

- UFS : inode.
- NTFS : stocké dans la Master File Table.



Méthodes d'accès

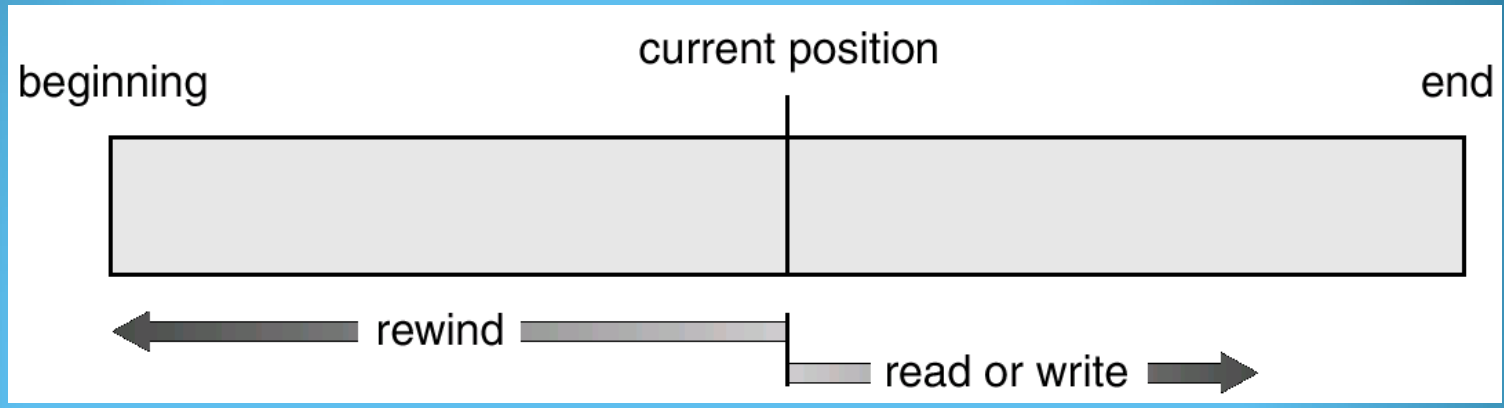
Séquentielle
Indexée
Directe



Méthodes d'accès: 4 de base

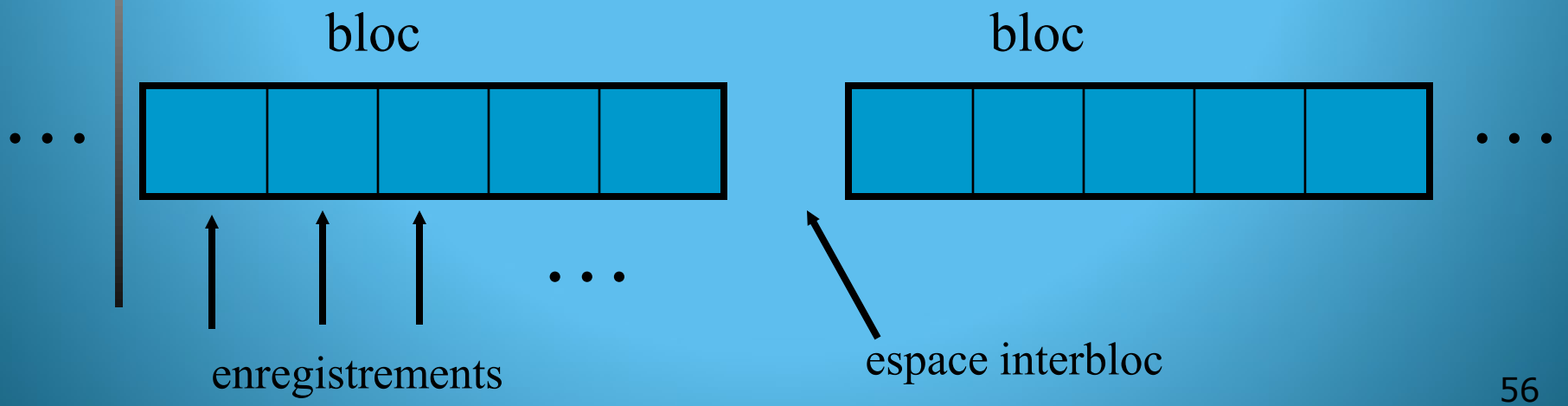
- Séquentiel (rubans ou disques): lecture ou écriture des enregistrements dans un ordre fixe
- Indexé séquentiel (disques): accès séquentiel ou accès direct (aléatoire) par l'utilisation d'index
- Indexée: multiplicité d'index selon les besoins, accès direct par l'index
- Direct ou hachée: accès direct à travers tableau d'hachage

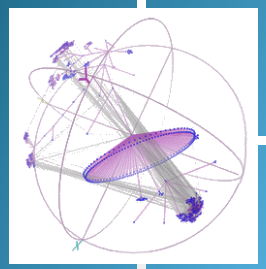
Fichiers à accès séquentiel (rubans)



La seule façon de retourner en arrière est de retourner au début (rébobiner, rewind)

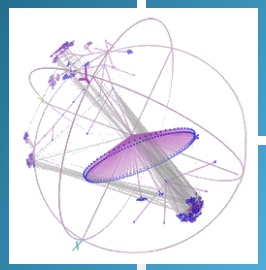
En avant seulement, 1 seul enreg. à la fois



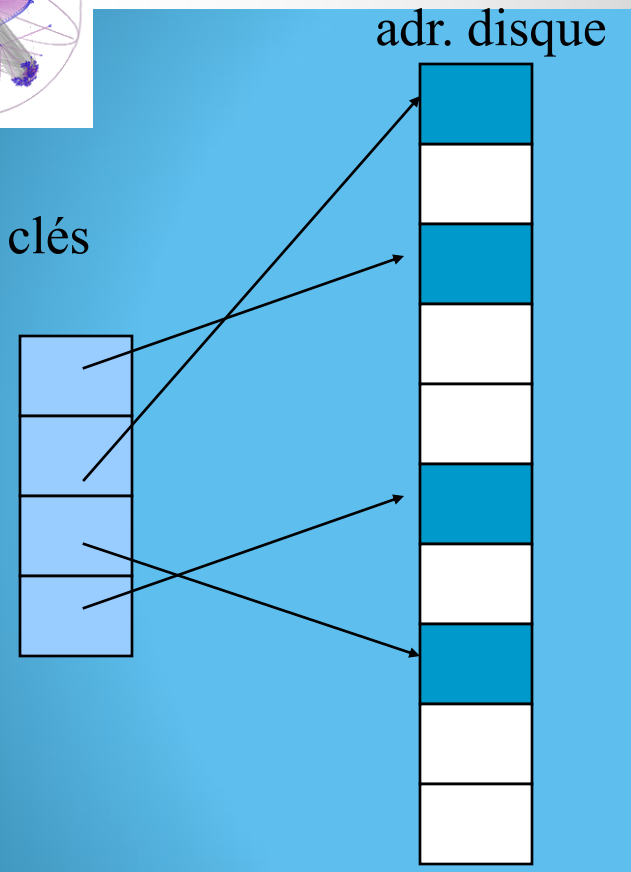


Accès direct ou haché ou aléatoire: accès direct à travers tableau d'hachage

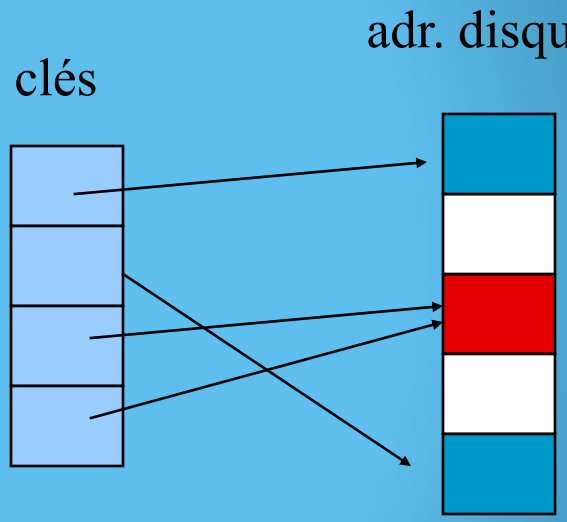
- Une fonction d'hachage est une fonction qui traduit une clé dans adresse,
  - P.ex. Matricule étudiant → adresse disque
- Rapide mais:
  - Si les adresses générées sont trop éparpillées, gaspillage d'espace
  - Si les adresses ne sont pas assez éparpillées, risque que deux clés soient renvoyées à la même adresse
    - Dans ce cas, il faut de quelques façon renvoyer une des clés à une autre adresse



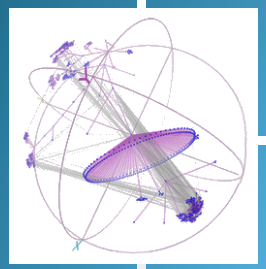
Problème avec les fonctions d'hachage



Fonction d'hachage dispersée qui n'utilise pas bien l'espace disponible

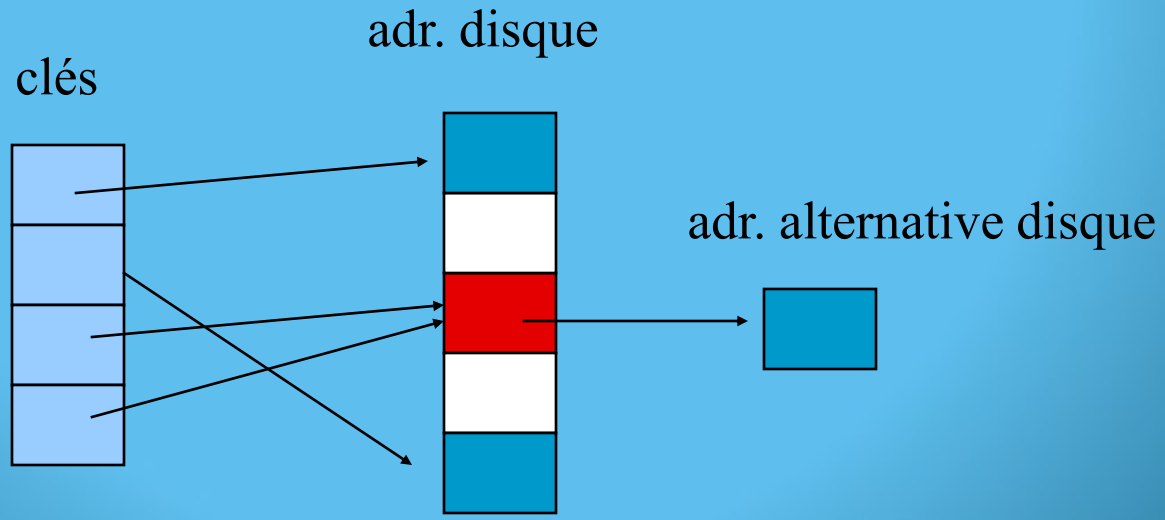


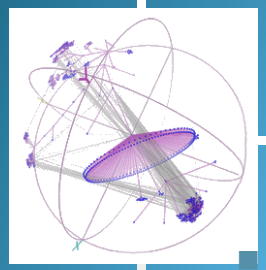
Fonction d'hachage concentrée qui utilise mieux l'espace mais introduit des doubles affectations



Hachage: Traitement des doubles affectations

- On doit détecter les doubles affectations, et s'il y en a, un des deux enregistrements doit être mis ailleurs
  - ce qui complique l'algorithme

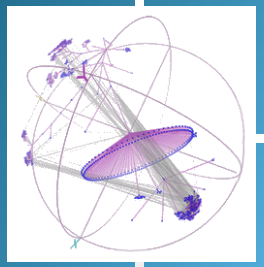




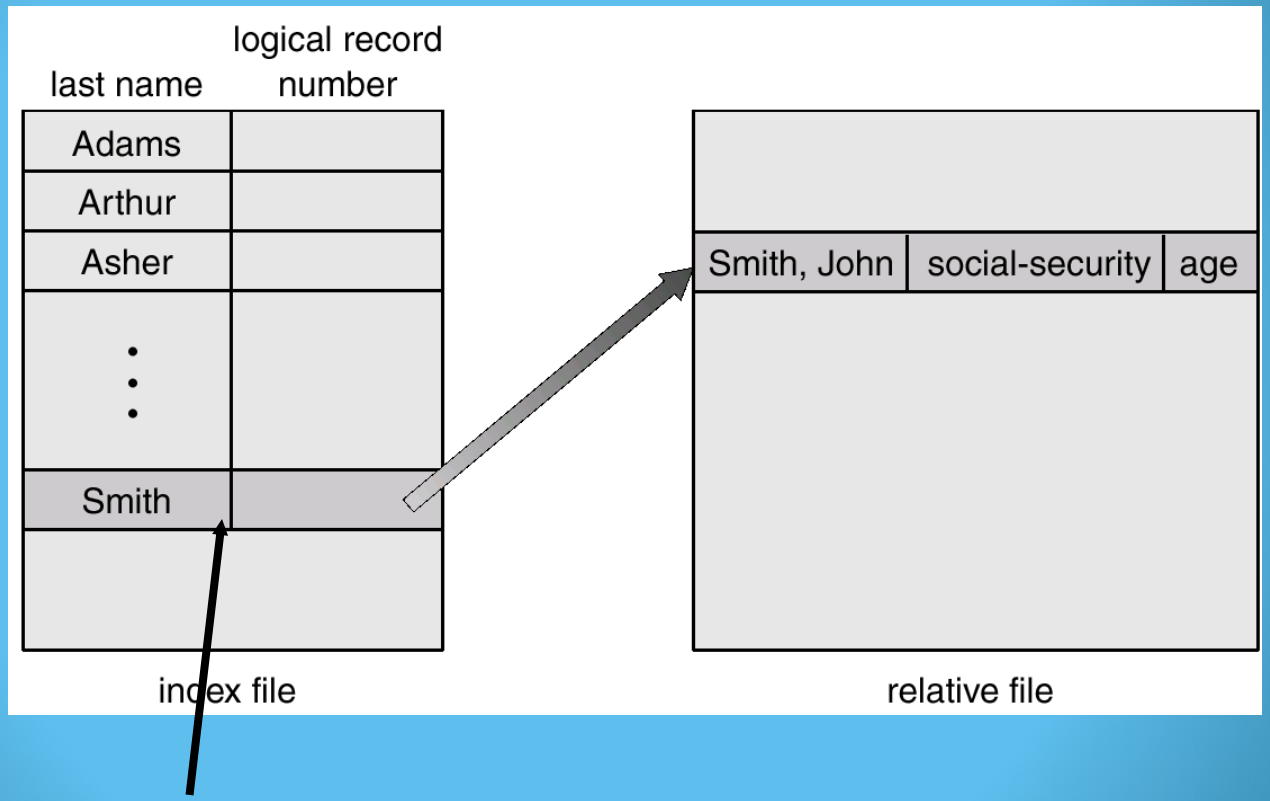
Adressage Indexé séquentiel

Un index permet d'arriver directement à l'enregistrement désiré, ou en sa proximité

- Chaque enregistrement contient un champ clé
- Un fichier index contient des repères (pointeurs) à certain points importants dans le fichier principal (p.ex. début de la lettre S, début des Lalande, début des matricules qui commencent par 8)
- Le fichier index pourra être organisé en niveaux (p.ex. dans l'index de S on trouve l'index de tous ceux qui commencent par S)
- Le fichier index permet d'arriver au point de repère dans le fichier principal, puis la recherche est séquentielle

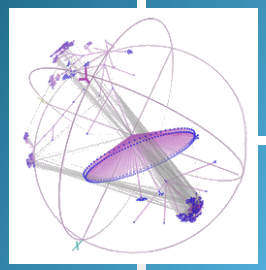


Exemples d'index et fichiers relatifs



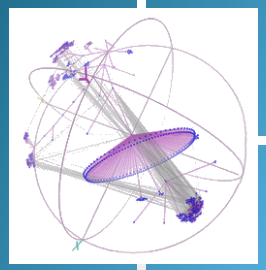
Pointe au début des Smiths (il y en aura plusieurs)

Le fichier index est à accès direct, le fichier relatif est à accès séquentiel

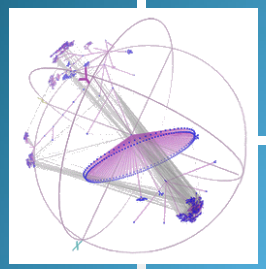


Comparaison : Séquentiel et index séquentiel

- P.ex. Un fichier contient 1 million d'enregistrements
- En moyenne, 500.000 accès sont nécessaires pour trouver un enregistrement si l'accès est séquentiel!
- Mais dans un séquentiel indexé, s'il y a un seul niveau d'indices avec 1000 entrées (et chaque entrée pointe donc à 1000 autres),
- *En moyenne*, ça prend 1 accès pour trouver le repère approprié dans le fichier index
- Puis 500 accès pour trouver séquentiellement le bon enregistrement dans le fichier relatif

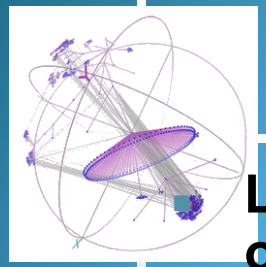


Méthodes d’allocation



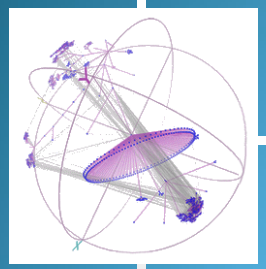
Structures de systèmes de fichiers

- Le système de fichiers réside dans la mémoire secondaire: disques, rubans...
- File control block: structure de données contenant de l'info sur un fichier (RAM)



Structure physique des fichiers

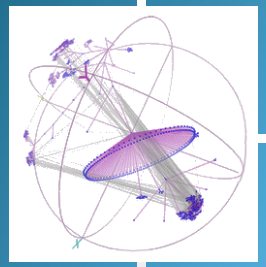
- **La mémoire secondaire est subdivisée en blocs et chaque opération d'E/S s'effectue en unités de blocs**
  - Les blocs ruban sont de longueur variable, mais les blocs disque sont de longueur fixe
  - Sur disque, un bloc est constitué d'un multiple de secteurs contiguës (ex: 1, 2, ou 4)
    - la taille d'un secteur est habituellement 512 octets
- **Il faut donc insérer les enregistrements dans les blocs et les extraire par la suite**
  - Simple lorsque chaque octet est un enregistrement par lui-même
  - Plus complexe lorsque les enregistrements possèdent une structure
- **Les fichiers sont alloués en unité de blocs. Le dernier bloc est donc rarement rempli de données**
  - Fragmentation interne



Méthodes d’allocation

Comment les blocs sur le disque sont alloués pour les fichiers. Il existe plusieurs méthodes, dont :

- Allocation contiguë
- Allocation liée
- Allocation indexée



Allocation contiguë

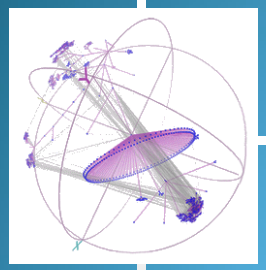
- Chaque fichier occupe des blocs contigus sur le disque.

Avantages

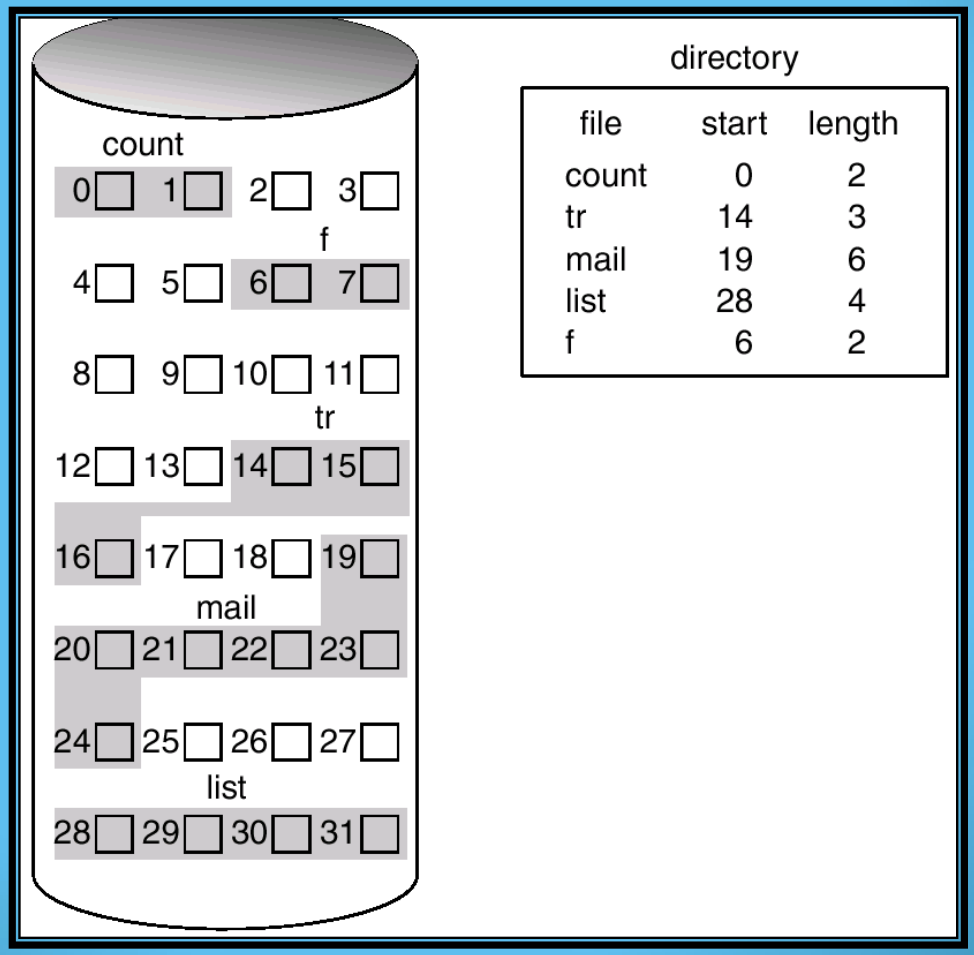
- Simplicité : il suffit de connaître la position (numéro du bloc) et la longueur du fichier (nombre de blocs).
- Accès aléatoire à l'information.

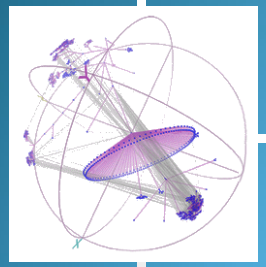
Inconvénients

- Perte d'espace disque (problèmes de fragmentation).
- Les fichiers ne peuvent pas grandir.



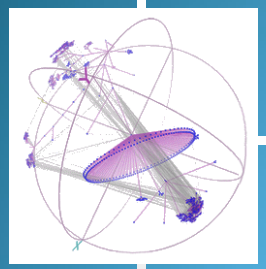
Allocation contiguë (2)





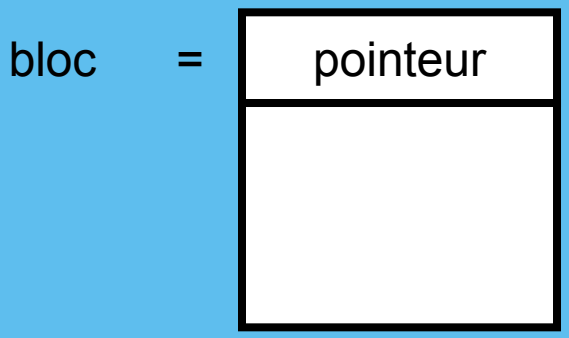
Allocation contiguë

- Chaque fichier occupe un ensemble de blocs contiguë sur disque
- Simple: nous n'avons besoin que d'adresses de début et longueur
- Supporte tant l'accès séquentiel, que l'accès direct
- Application des problèmes et méthodes vus dans le chapitre de l'allocation de mémoire contiguë
- Les fichiers ne peuvent pas grandir
- Impossible d'ajouter au milieu
- Exécution périodique d'une compression (compaction) pour récupérer l'espace libre

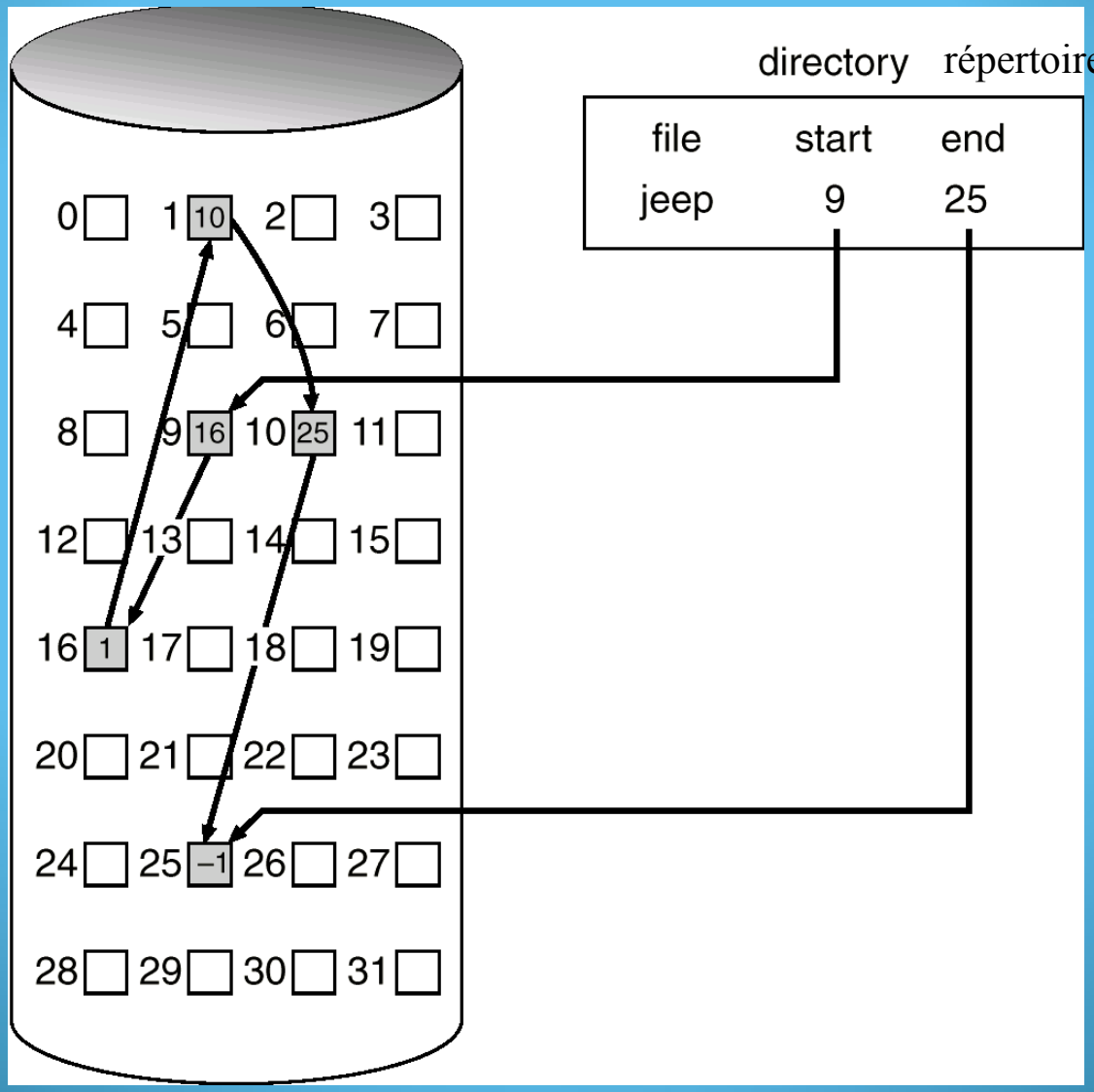


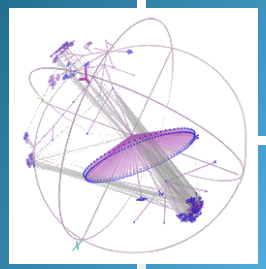
Allocation enchaînée

- Le répertoire contient l'adresse du premier et dernier bloc, possible le nombre de blocs
- Chaque bloc contient un pointeur à l'adresse du prochain bloc:



Allocation enchaînée





Avantages - désavantages

- Pas de fragmentation externe - allocation de mémoire simple, pas besoin de compression
- L'accès à l'intérieur d'un fichier ne peut être que séquentiel
  - Pas de façon de trouver directement le 4ème enregistrement...
- L'intégrité des pointeurs est essentielle
- Les pointeurs gaspillent un peu d'espace

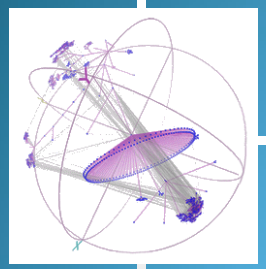


Table d'allocation des fichiers

- Une variation de l'allocation liée est d'utiliser une table d'allocation des fichiers (FAT).
  - Utilisé par MS-DOS et OS/2.
  - Une partie du disque est réservée pour stocker la table qui contient les pointeurs vers tous les fichiers de la partition.
  - Chaque entrée dans la FAT correspond à un bloc sur le disque. Chaque entrée contient le pointeur vers le bloc suivant du fichier.
  - Une valeur spéciale indique la fin du fichier.
  - Une entrée nulle signifie un bloc inutilisé.

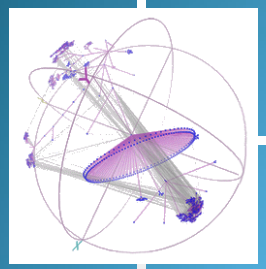
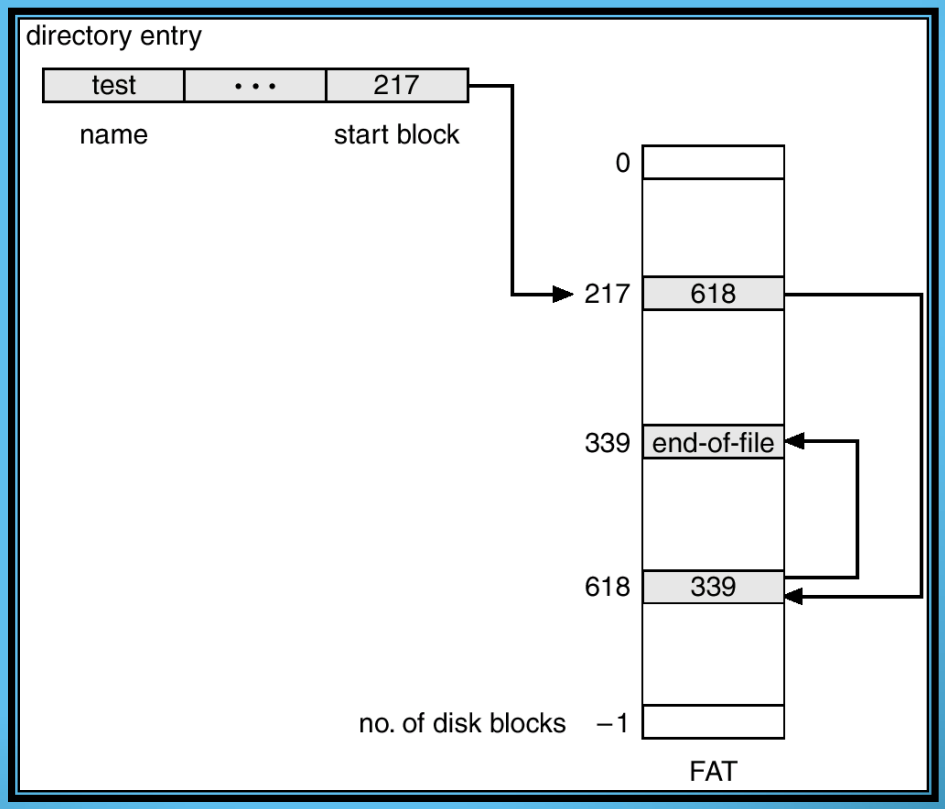
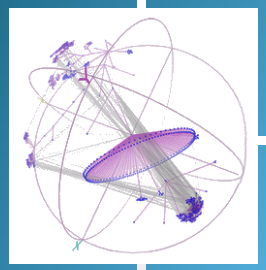


Table d'allocation

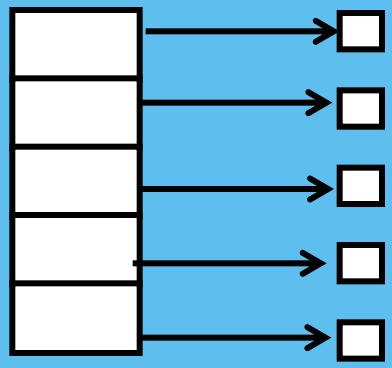
- Les FATs sont stockées en mémoire tant que le SE est actif

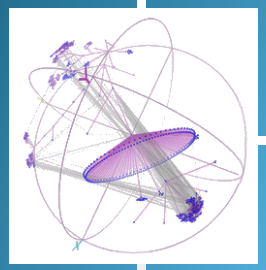




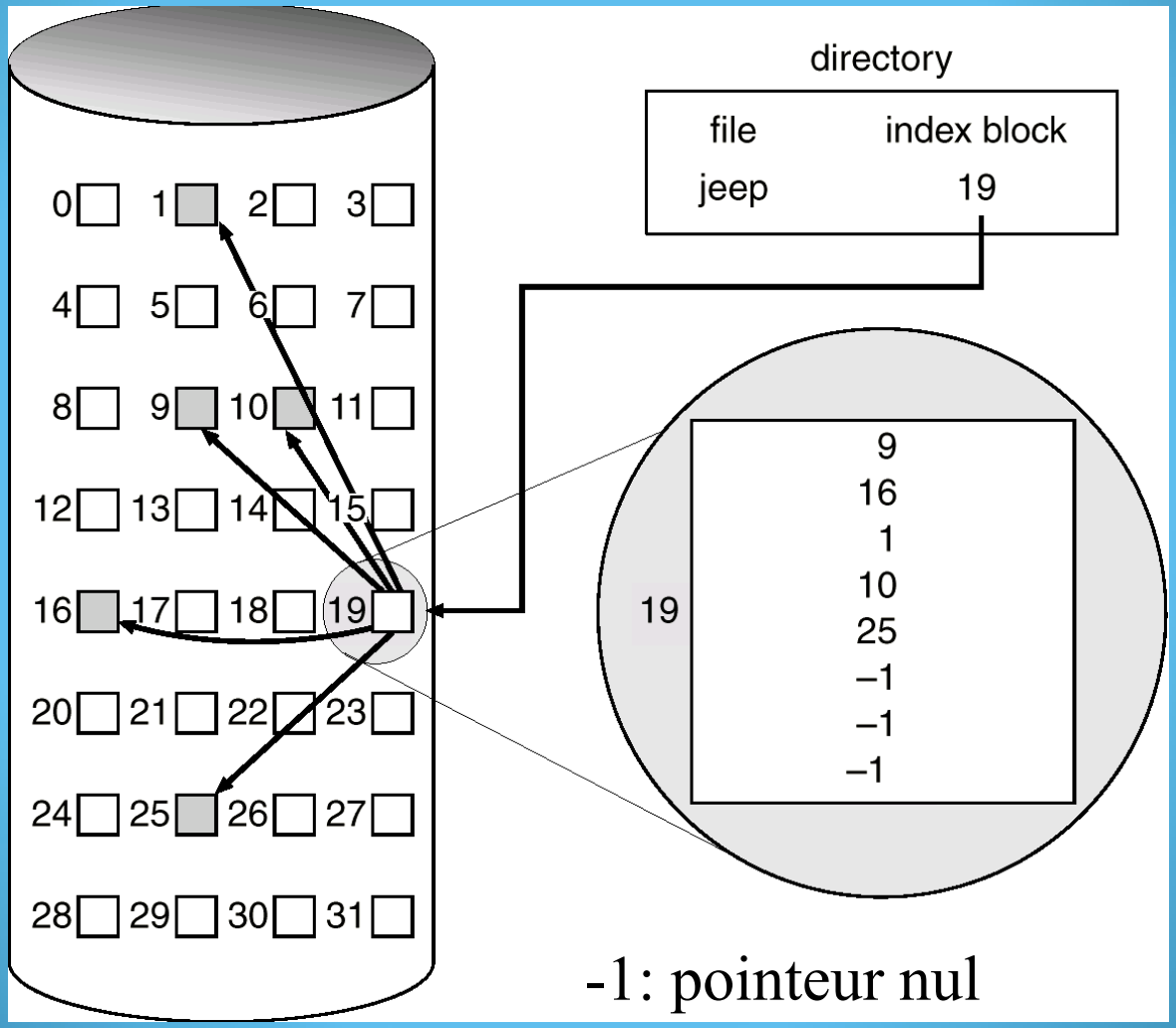
Allocation indexée: semblable à la pagination

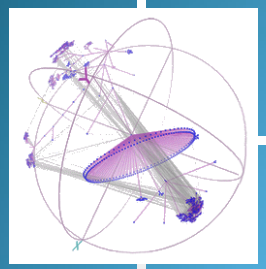
- Tous les pointeurs sont regroupés dans un tableau (index block)





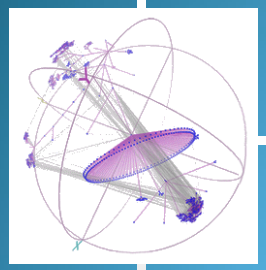
Allocation indexée



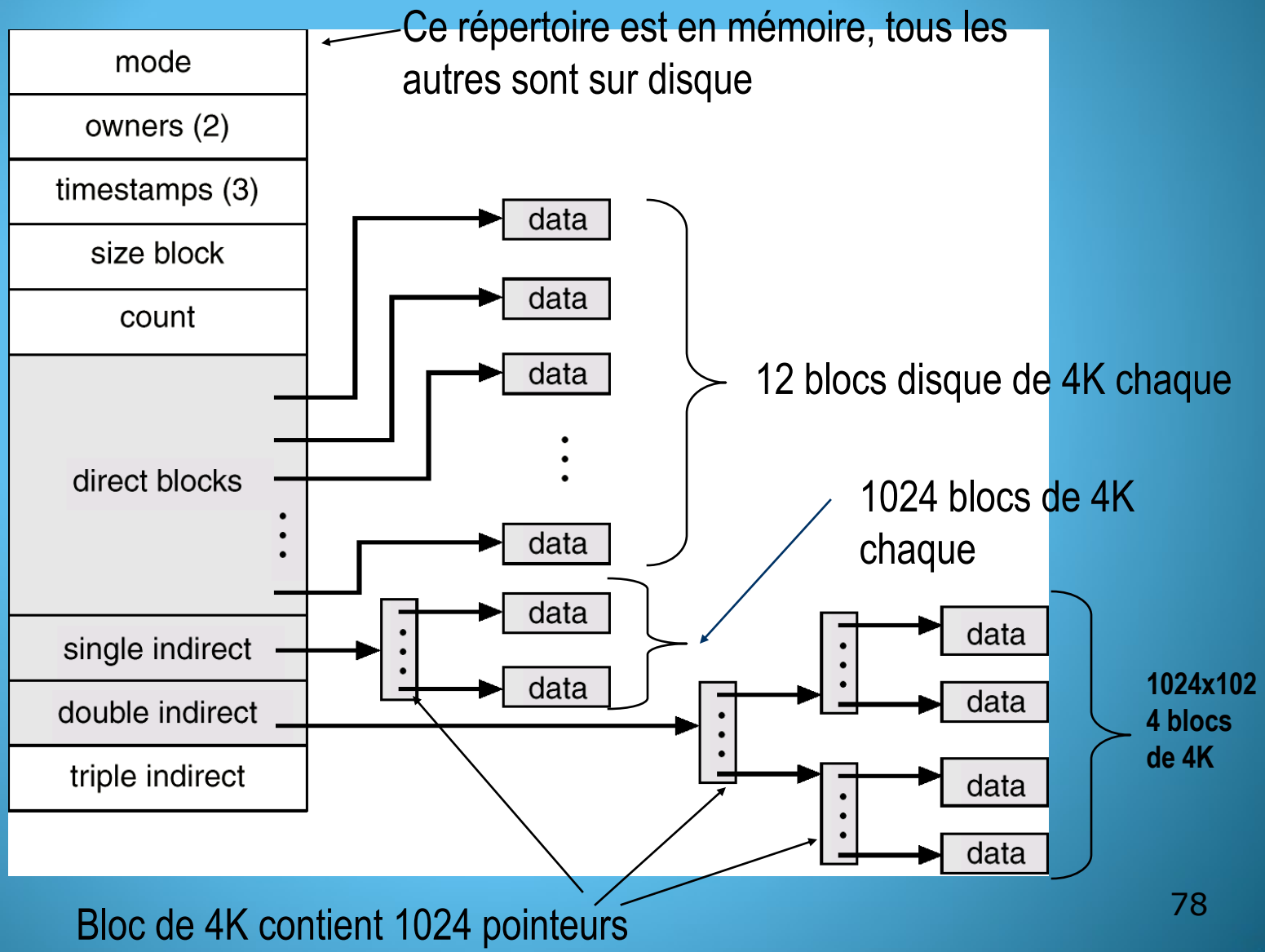


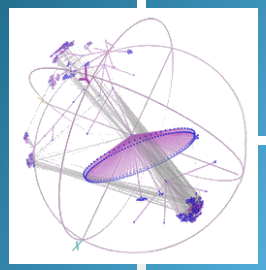
Allocation indexée

- À la création d'un fichier, tous les pointeurs dans le tableau sont *nil (-1)*
- Chaque fois qu'un nouveau bloc doit être alloué, on trouve de l'espace disponible et on ajoute un pointeur avec son adresse
- Pas de fragmentation externe, mais les index prennent de l'espace
- Permet accès direct (aléatoire)
- Taille de fichiers limitée par la taille de l'index block
  - Mais nous pouvons avoir plusieurs niveaux d'index: Unix
- Index block peut utiliser beaucoup de mémoire



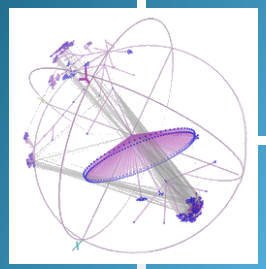
UNIX BSD: indexé à niveaux (config. possible)





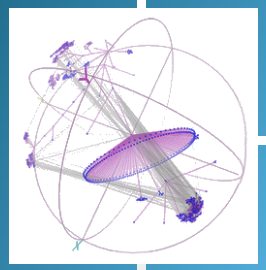
Concepts communs

- Les fichiers sont représentés par des inodes
- Les répertoires sont des fichiers
- Les périphériques sont accédés par des entrées sorties sur des fichiers spéciaux (/dev/hda0)
- Les liens sont autorisés

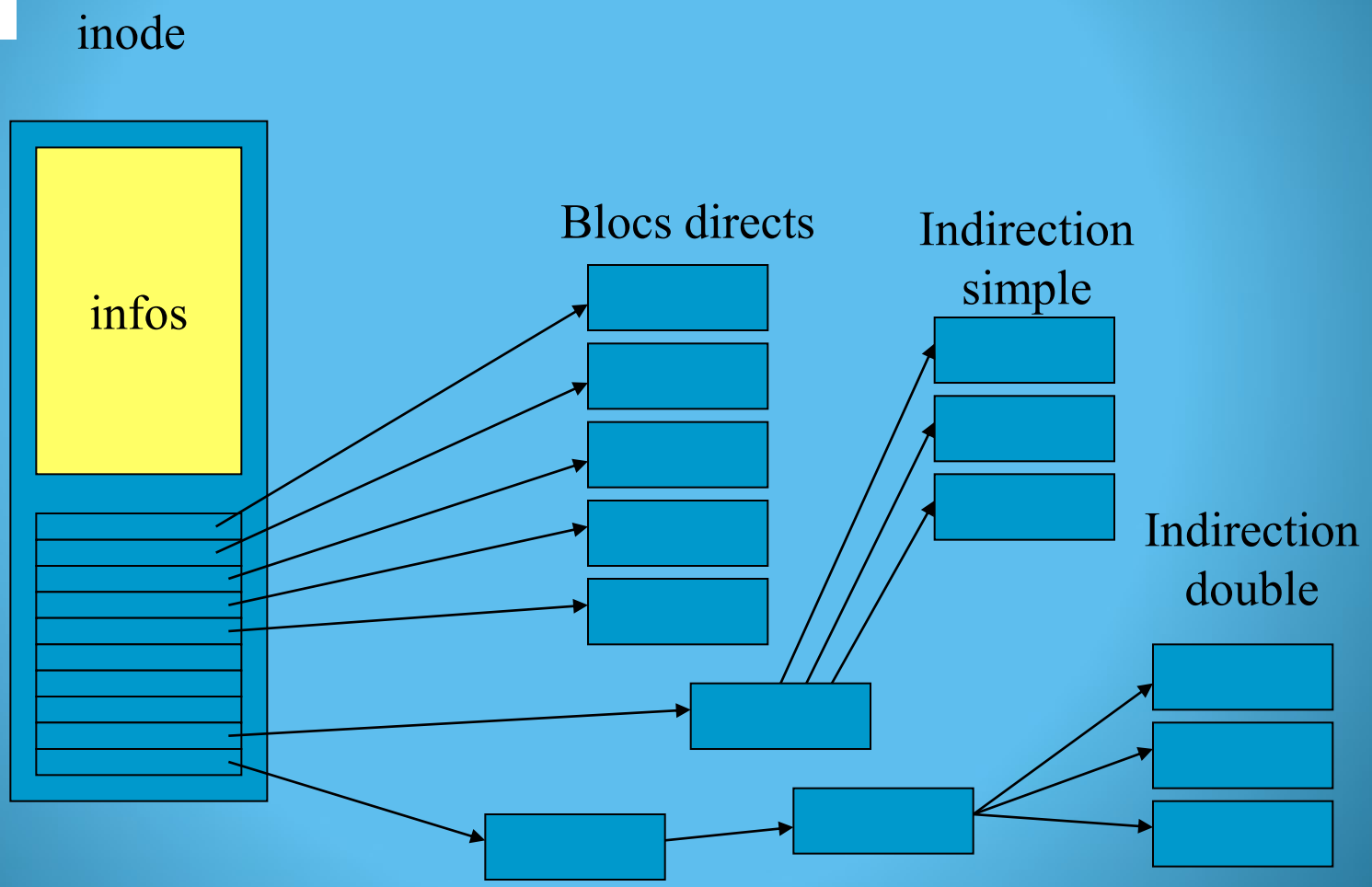


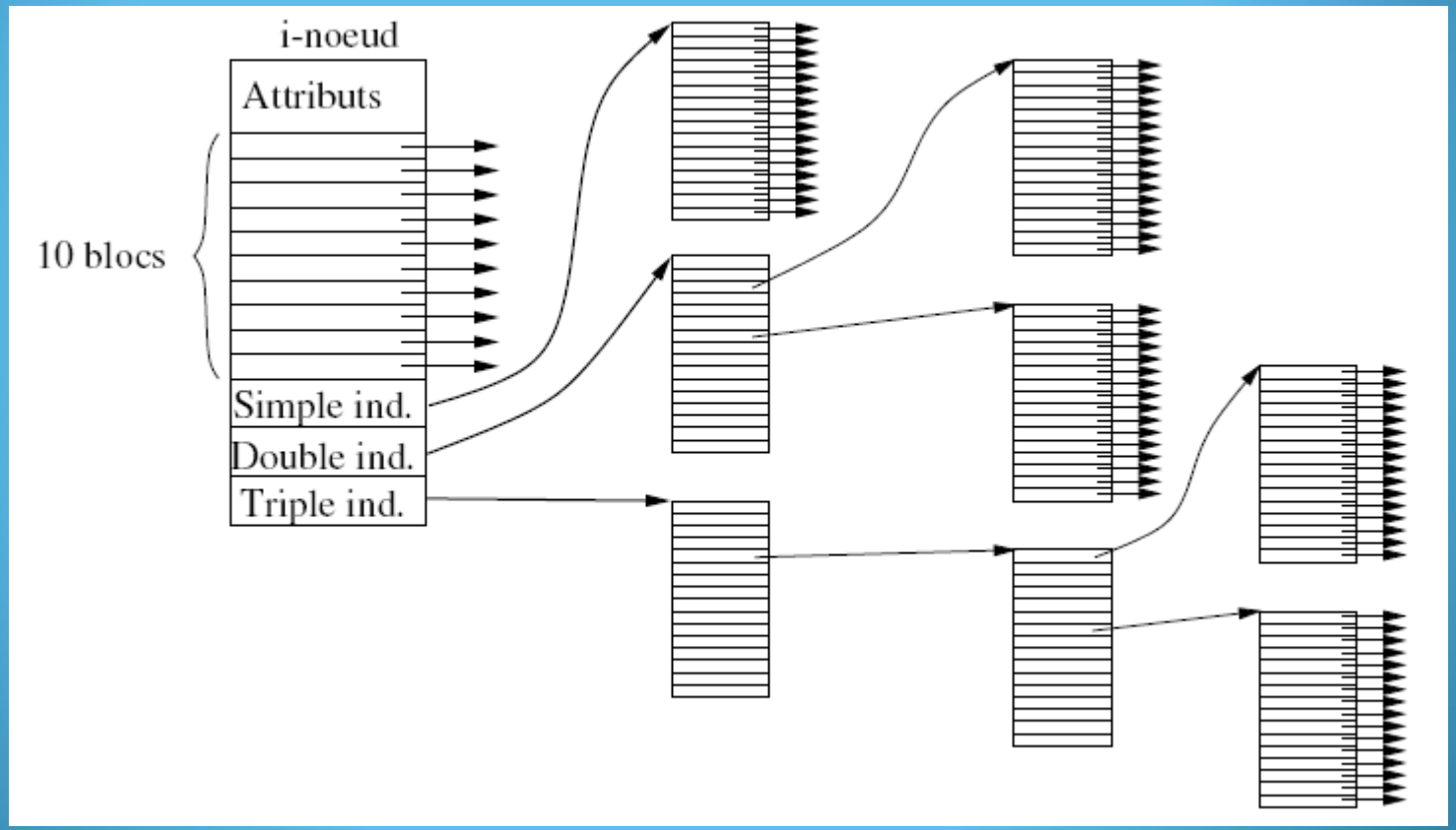
Inodes

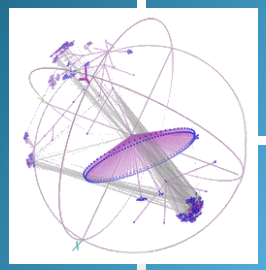
- Une structure qui contient la description du fichier :
  - Type
  - Droits d'accès
  - Possesseurs
  - Dates de modifications
  - Taille
  - Pointeurs vers les blocs de données
- Les inodes sont stockés en mémoire tant que le fichier est ouvert



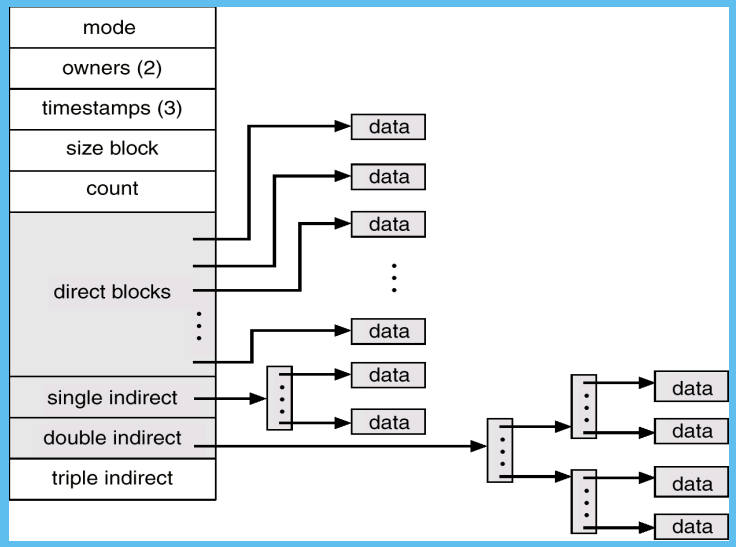
Inodes (2)


inode
infos
Blocs directs
Indirection simple
Indirection double


i-noeud
Attributs
10 blocs
Simple ind.
Double ind.
Triple ind.

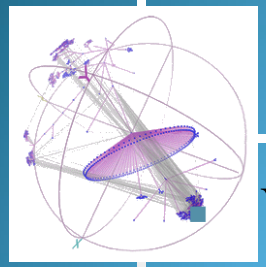


UNIX BSD

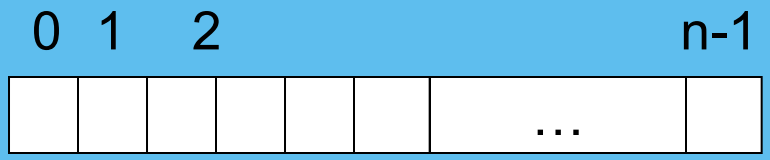


- Les premiers blocs d'un fichier sont accessibles directement
- Si le fichier contient des blocs additionnels, les premiers sont accessibles à travers un niveau d'indices
- Les suivants sont accessibles à travers 2 niveaux d'indices, etc.
- Donc le plus loin du début un enregistrement se trouve, le plus indirect est son accès
- Permet accès rapide à petits fichiers, et au début de tous les fichiers.
- Permet l'accès à des grands fichier avec un petit répertoire en mémoire

Gestion d'espace libre



Vecteur de bits (*n* blocs)



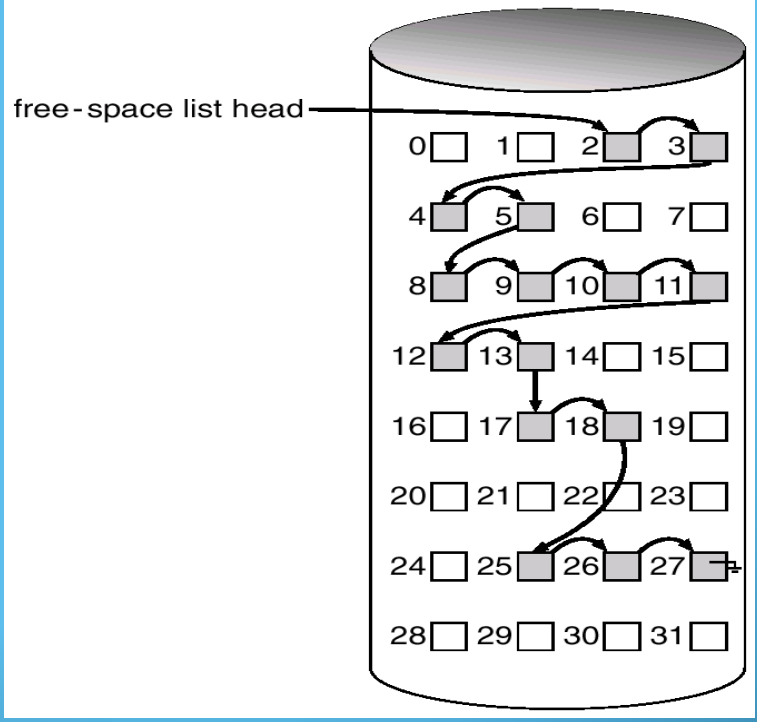
$$bit[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ libre} \\ 1 \Rightarrow \text{block}[i] \text{ occupé} \end{cases}$$

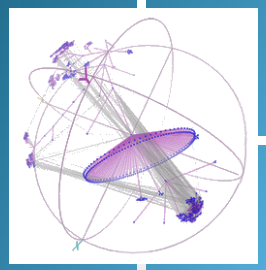
- **Exemple d'un vecteur de bits où les blocs 3, 4, 5, 9, 10, 15, 16 sont occupés:**
  - 0001110001100001 1…
- **L'adresse du premier bloc libre peut être trouvée par un simple calcul**

Gestion d'espace libre

Solution 2: Liste liée de mémoire libre (MS-DOS, Windows 9x)

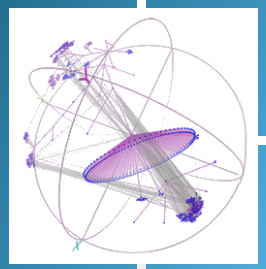
Tous les blocs de mémoire libre sont liés ensemble par des pointeurs





Comparaison

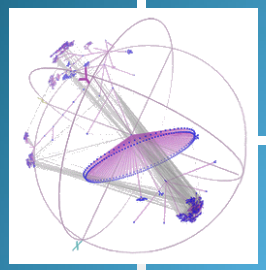
- Bitmap:
  - si la bitmap de toute la mémoire secondaire est gardée en mémoire principale, la méthode est rapide mais demande de l'espace de mémoire principale
  - si les bitmaps sont gardées en mémoire secondaire, temps de lecture de mémoire secondaire...
    - Elles pourraient être paginées, p.ex.
- Liste liée
  - Pour trouver plusieurs blocs de mémoire libre, plus d'accès disques pourraient être demandés
  - Pour augmenter l'efficacité, nous pouvons garder en mémoire centrale l'adresse du 1er bloc libre



Structure de mémoire de masse (disques)

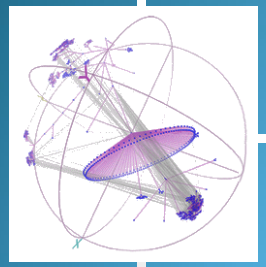
Concepts importants :

- **Fonctionnement et structure des unités disque**
- **Calcul du temps d'exécution d'une séquence d'opérations**
- **Différents algorithmes d'ordonnancement**
  - Fonctionnement, rendement
- **Gestion de l'espace de permutation**
  - Unix



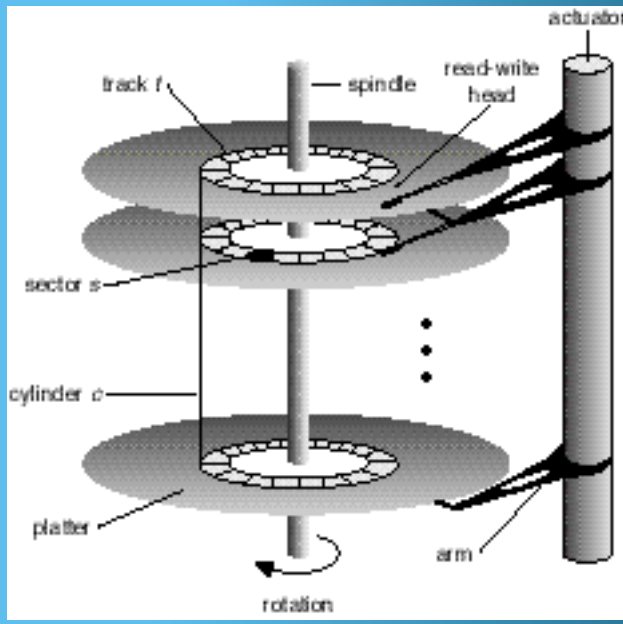
Ordonnancement disques

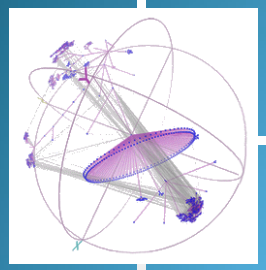
- Problème: utilisation optimale du matériel
- Réduction du temps total de lecture disque
  - étant donné une file de requêtes de lecture disque, dans quel ordre les exécuter?



Paramètres à prendre en considération

- Temps de positionnement (seek time):
  - le temps pris par l`unité disque pour se positionner sur le cylindre désiré
- Temps de latence de rotation (latency time)
  - le temps pris par l ’unité de disque qui est sur le bon cylindre pour se positionner sur le secteur désirée
- Temps de lecture
  - temps nécessaire pour lire la piste
- Le temps de positionnement est normalement le plus important, donc il est celui que nous chercherons à minimiser



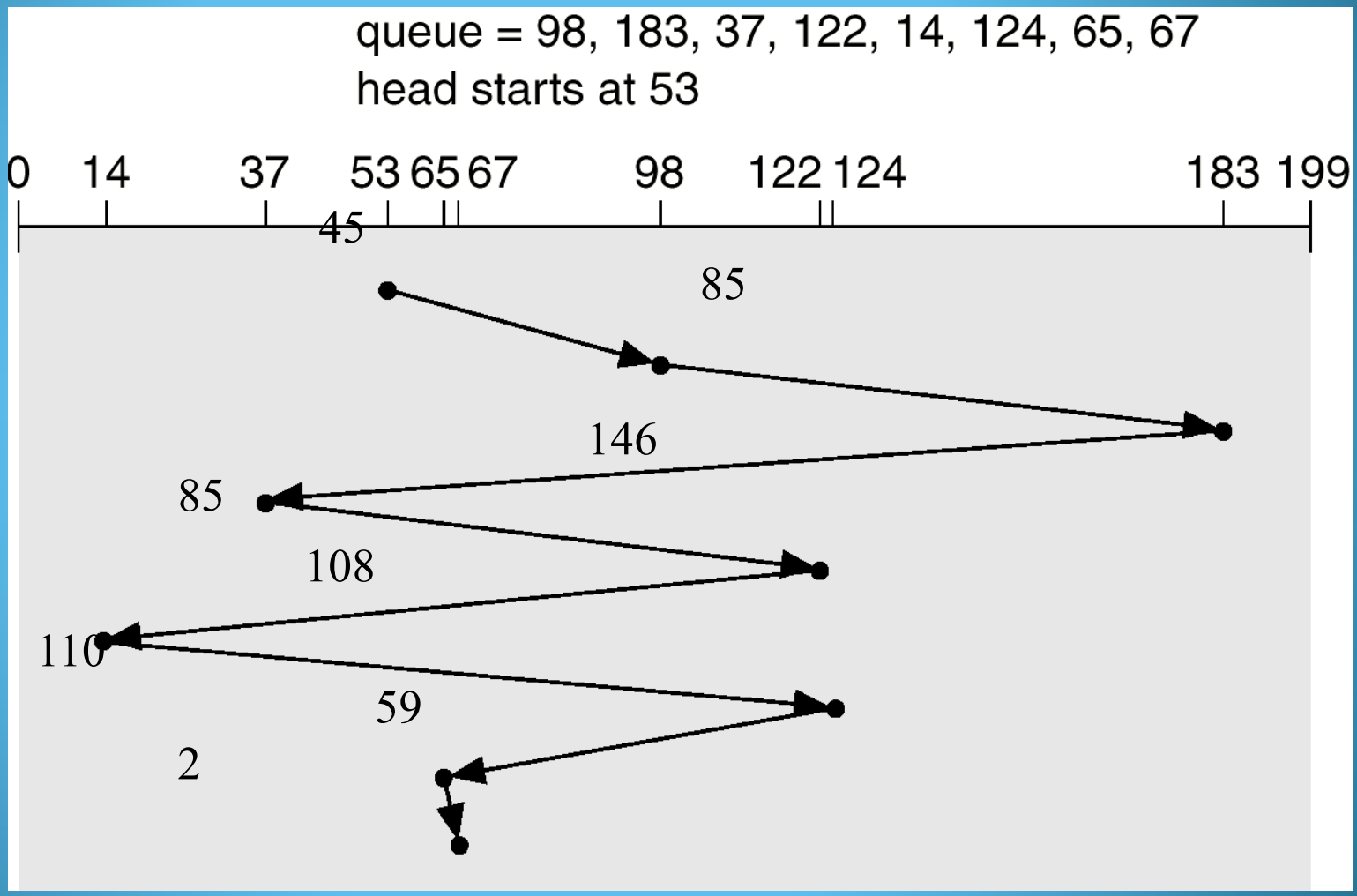


File d'attente disque

- Dans un système multiprogrammé avec mémoire virtuelle, il y aura normalement une file d'attente pour l'unité disque
- Dans quel ordre choisir les requêtes d'opérations disques de façon à minimiser les temps de recherche totaux
- Nous étudierons différents méthodes par rapport à une file d'attente arbitraire:

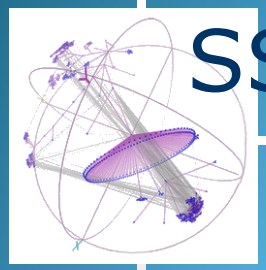
  98, 183, 37, 122, 14, 124, 65, 67
- Chaque chiffre est un numéro séquentiel de cylindre
- Il faut aussi prendre en considération le cylindre de départ: 53
- Dans quel ordre exécuter les requêtes de lecture de façon à minimiser les temps totaux de positionnement cylindre
- Hypothèse simpliste: un déplacement d`1 cylindre coûte 1 unité de temps

Premier entré, premier sorti: FIFO



Mouvement total: 640 cylindres = (98-53) + (183-98)+...

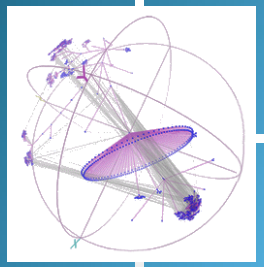
En moyenne: 640/8 = 80



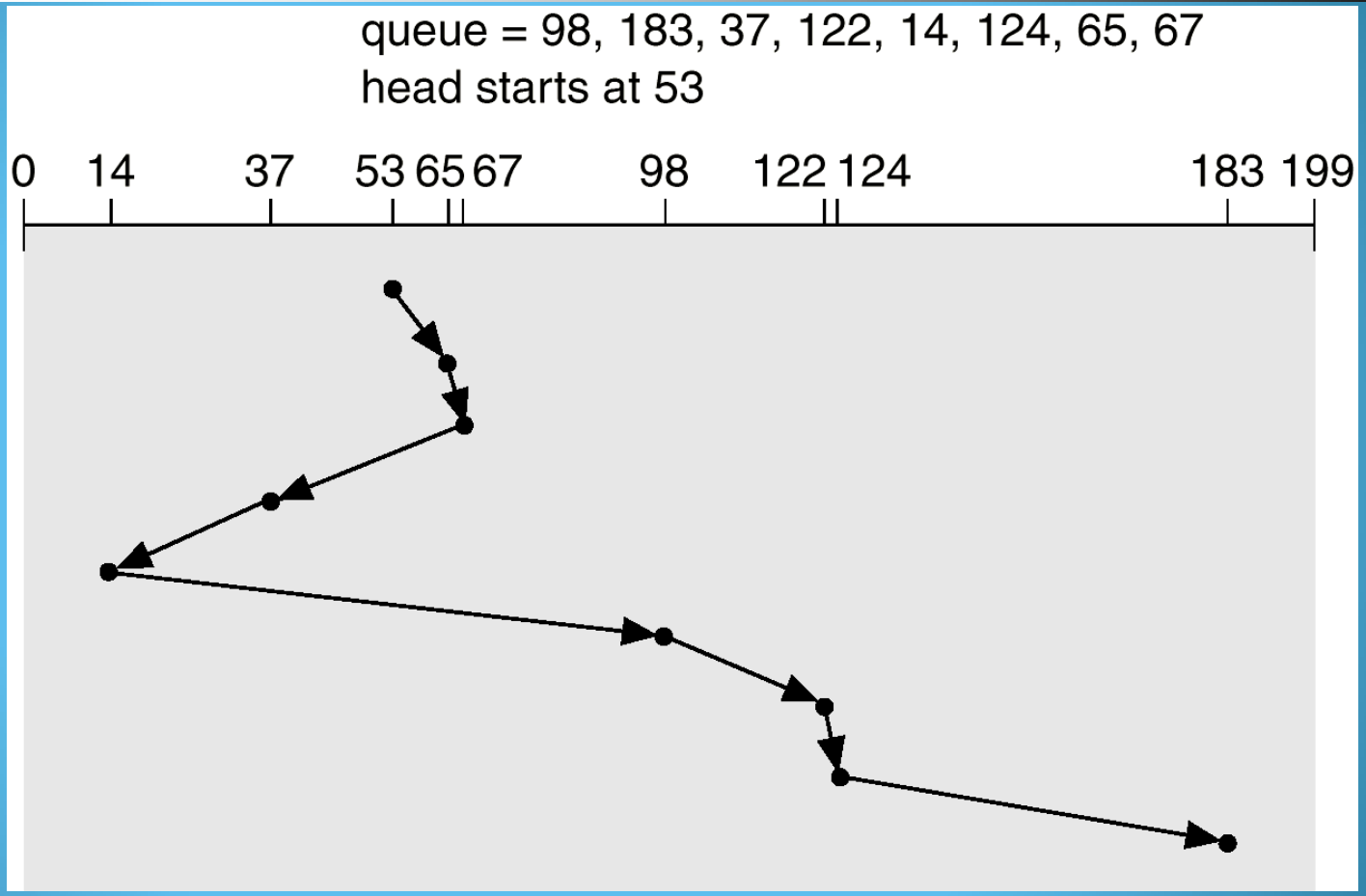
SSTF: Shortest Seek Time First

Plus Court Temps de Recherche (positionnement) d'abord (PCTR)

- À chaque moment, choisir la requête avec le temps de recherche le plus court à partir du cylindre courant
- Clairement meilleur que le précédent
- Mais pas nécessairement optimal!
- Peut causer famine

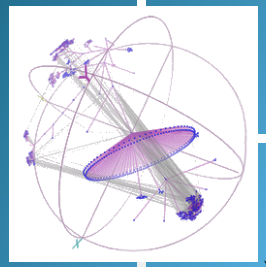


SSTF: Plus court servi (PCTR)



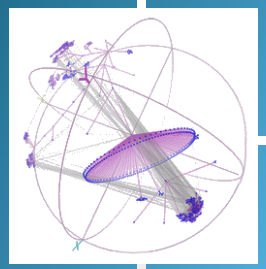
Mouvement total: 236 cylindres (680 pour le précédent)

En moyenne: 236/8 = 29.5 (80 pour le précédent)



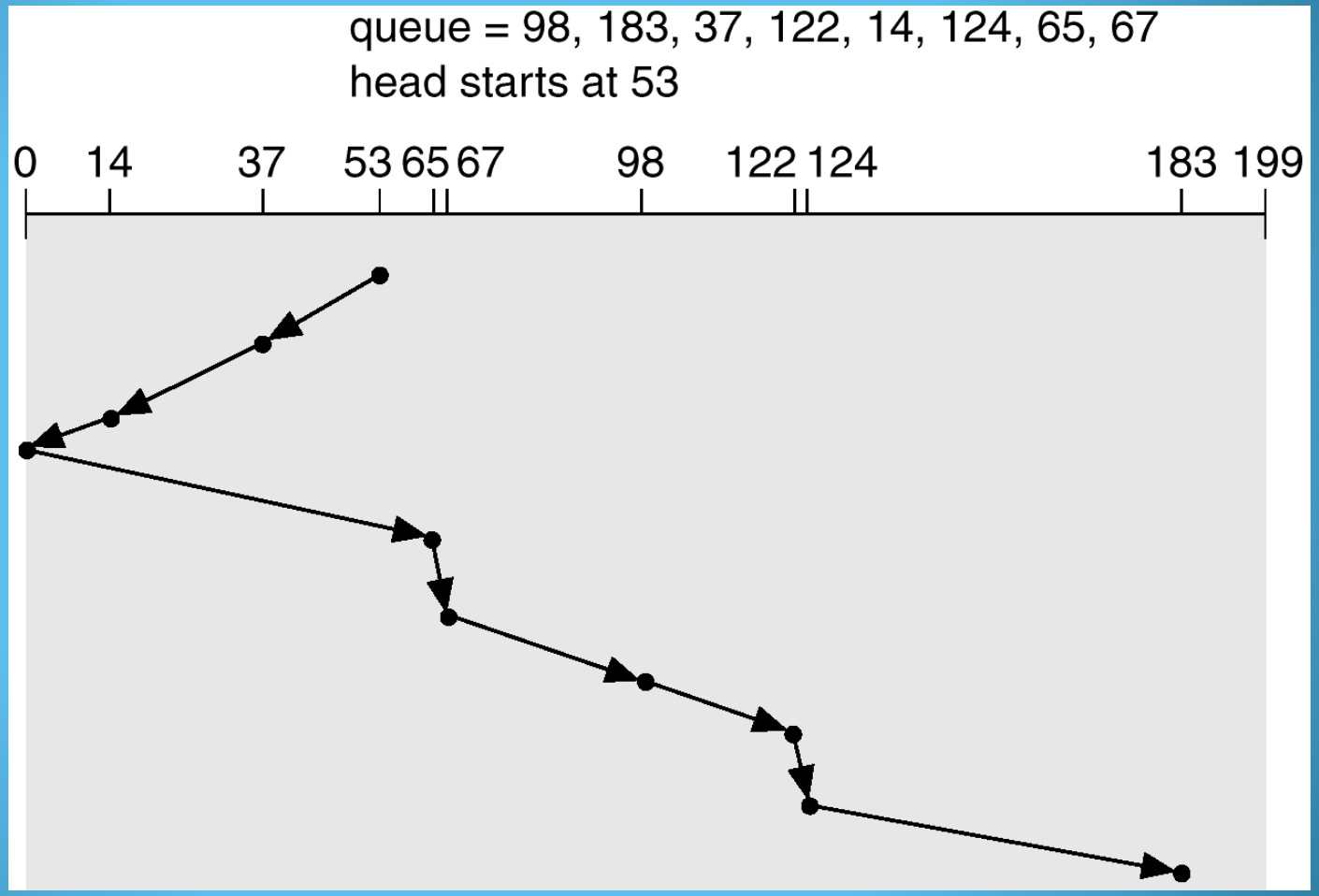
LOOK: l'algorithme de l'ascenseur
SCAN: l'algorithme du bus

- Scan : La tête balaye le disque dans une direction, puis dans la direction opposée, jusqu'au bout. etc., en desservant les requêtes quand il passe sur le cylindre désiré
  - Pas de famine
- Look : La tête balaye le disque dans une direction jusqu'il n'y aie plus de requête dans cette direction, puis dans la direction opposée de même, etc., en desservant les requêtes quandil passe sur le cylindre désiré



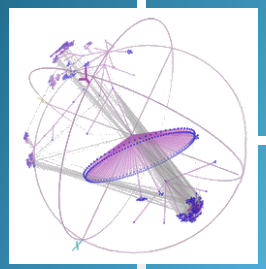
SCAN: l'ascenseur

direction ←



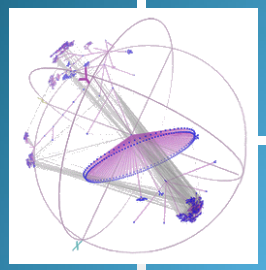
Mouvement total: 208 cylindres

En moyenne: 208/8 = 26 (29.5 pour SSTF)



Problèmes du SCAN

- Peu de travail à faire après le renversement de direction
- Les requêtes seront plus denses à l'autre extrémité
- Arrive inutilement jusqu'à 0



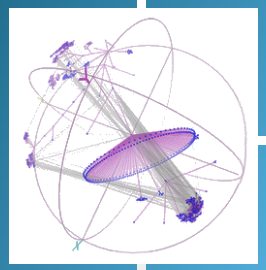
C-SCAN
C-Look

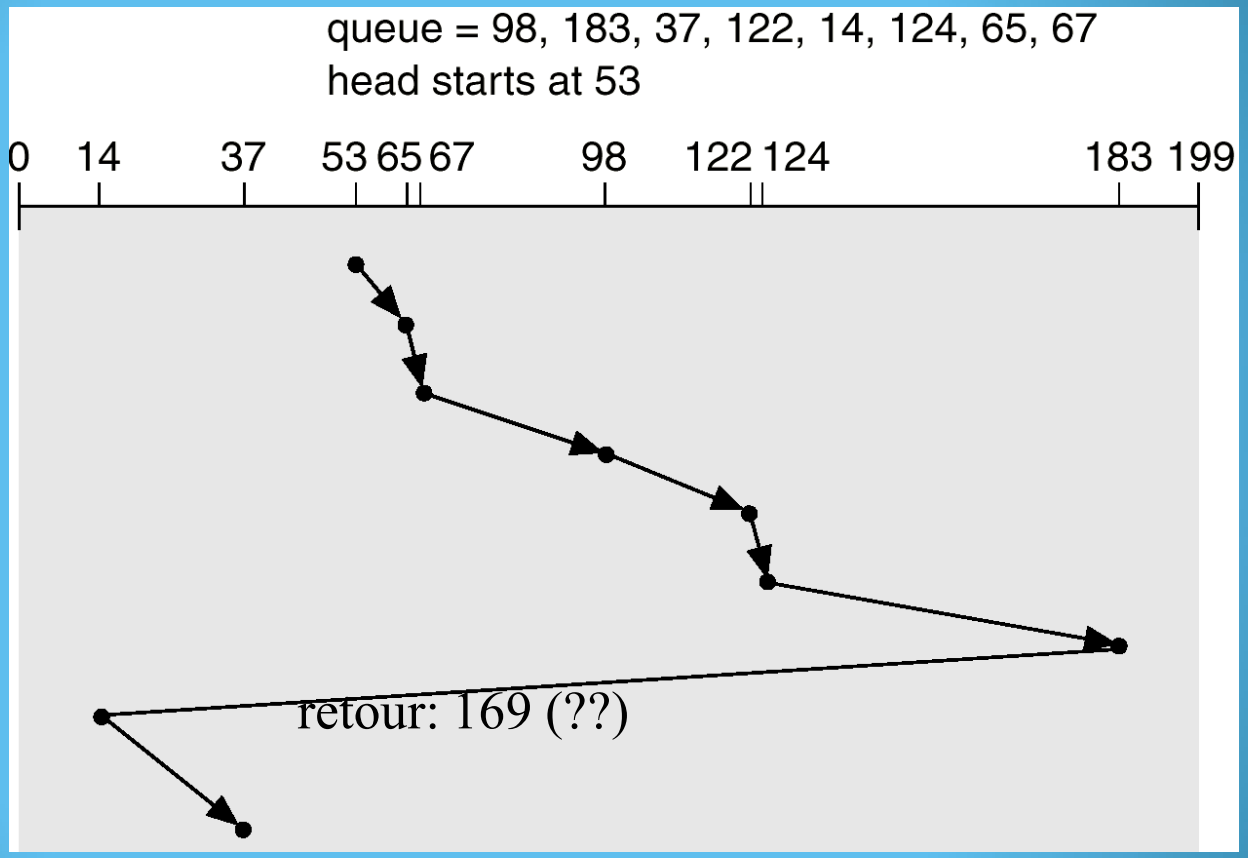
- Retour rapide au début (cylindre 0) du disque au lieu de renverser la direction
- Hypothèse: le mécanisme de retour est beaucoup plus rapide que le temps de visiter les cylindres
  - Comme si les disques étaient en forme de beignes!

C-LOOK

- La même idée, mais au lieu de retourner au cylindre 0, retourner au premier cylindre qui a une requête

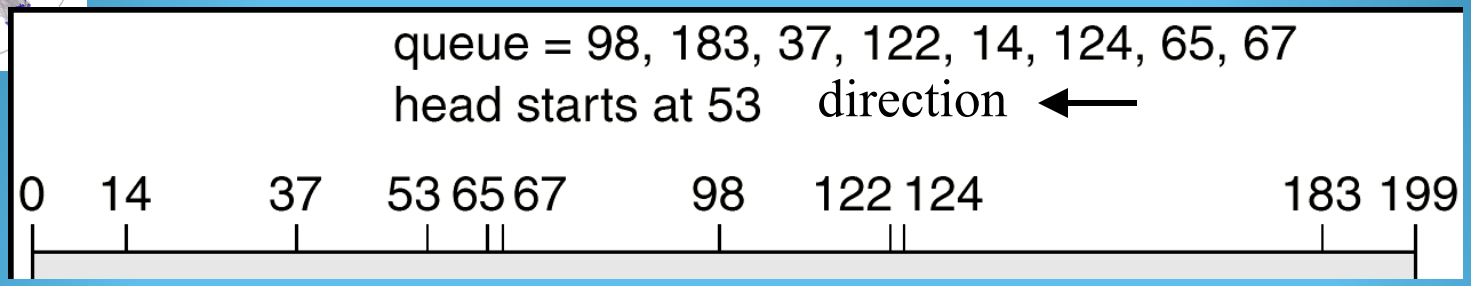
C-LOOK

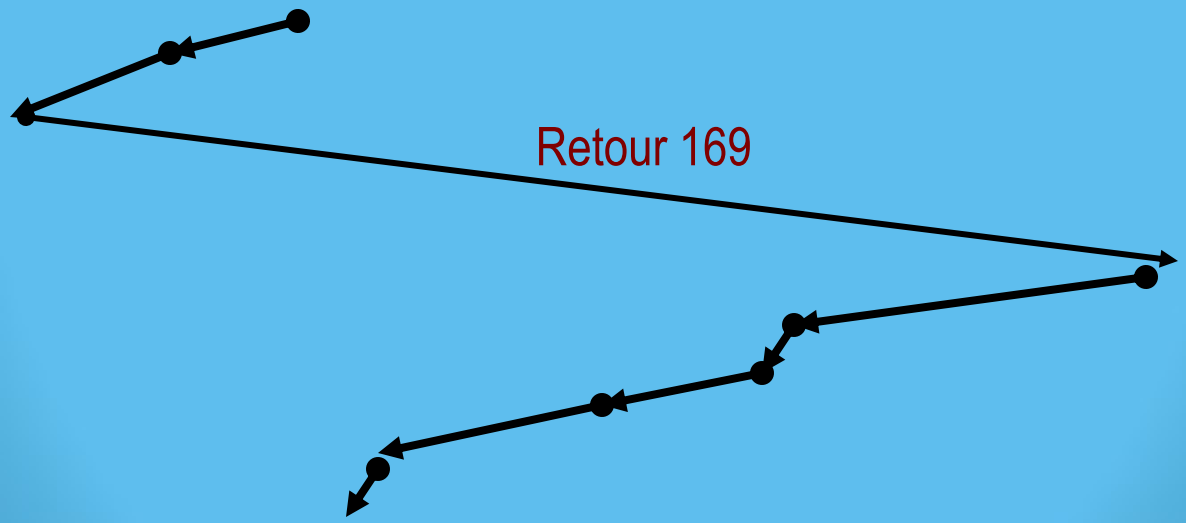




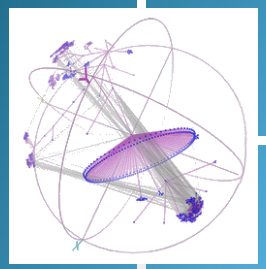
153 sans considérer le retour (19.1 en moyenne) (26 pour SCAN)
MAIS 322 avec retour (40.25 en moyenne)
Normalement le retour sera rapide donc le coût réel sera entre les deux

C-LOOK avec direction initiale opposée





Résultats très semblables:
157 sans considérer le retour, 326 avec le retour



Comparaison

- Si la file souvent ne contient que très peu d'éléments, l'algorithme du `premier servi ' devrait être préféré (simplicité)
- Sinon, SSTF ou SCAN ou C-SCAN?
- En pratique, il faut prendre en considération:
  - Les temps réels de déplacement et retour au début
  - L`organisation des fichiers et des répertoires
    - Les répertoires sont sur disque aussi…
  - La longueur moyenne de la file
  - Le débit d 'arrivée des requêtes