

Chapitre 6

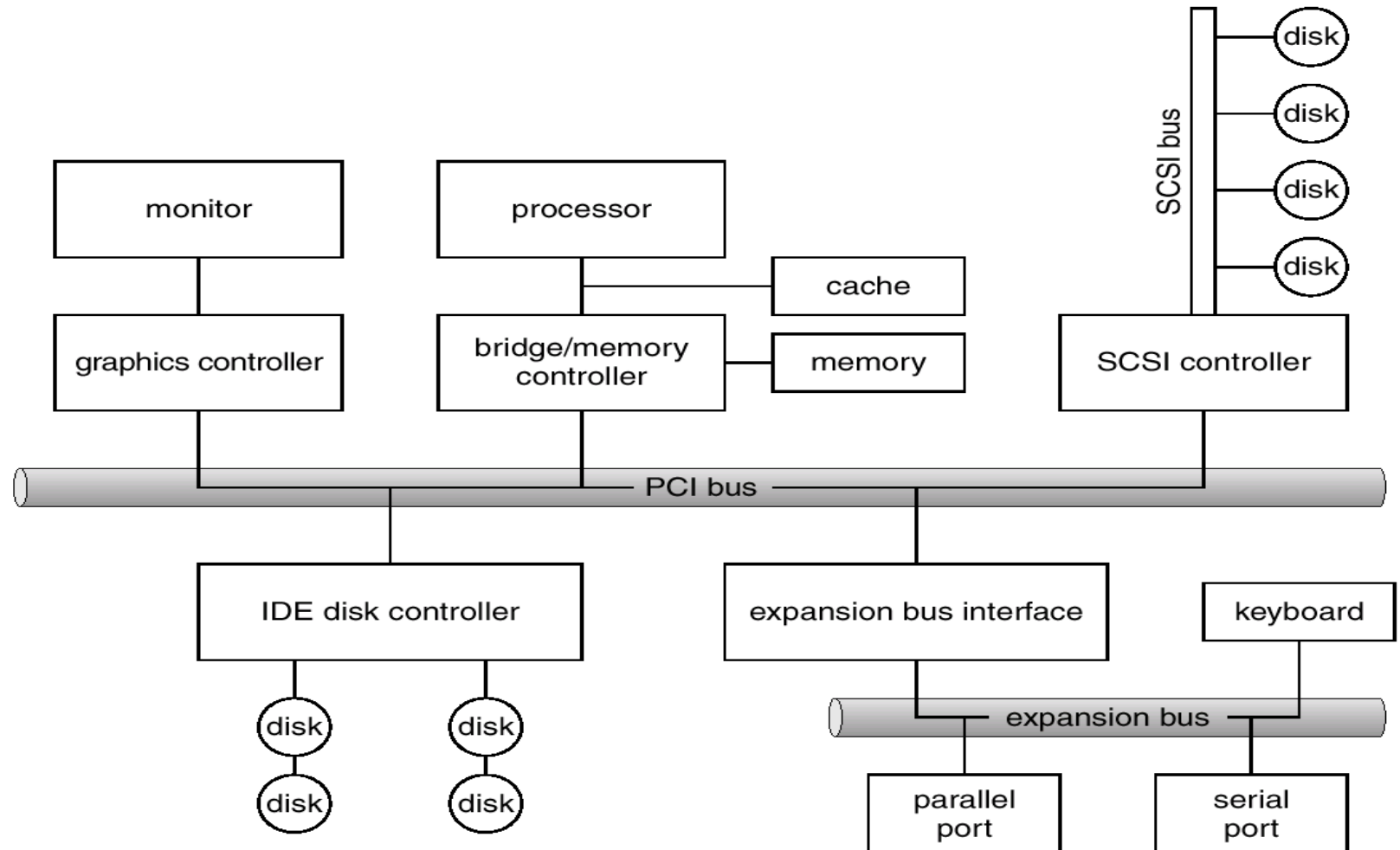
1. **Systemes d'entrée/sortie**
2. **Systemes de fichiers**
3. **Structure de mémoire de masse (disques)**

1. 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

Structure typique de bus PC

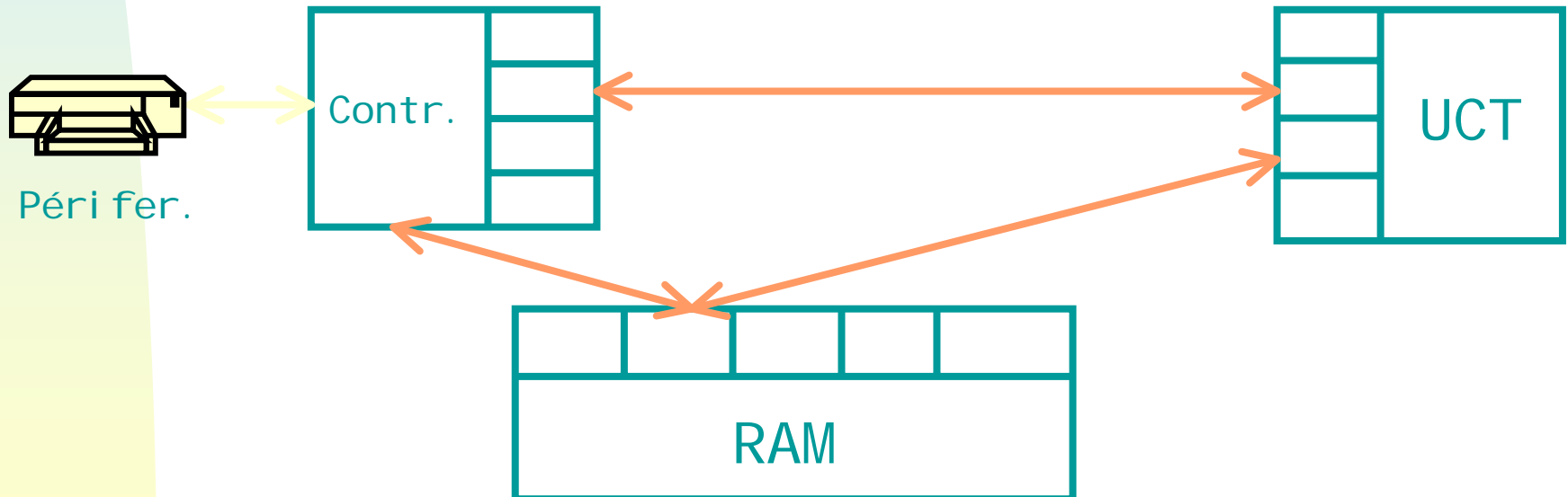


PCI: Peripheral Component Interconnect

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



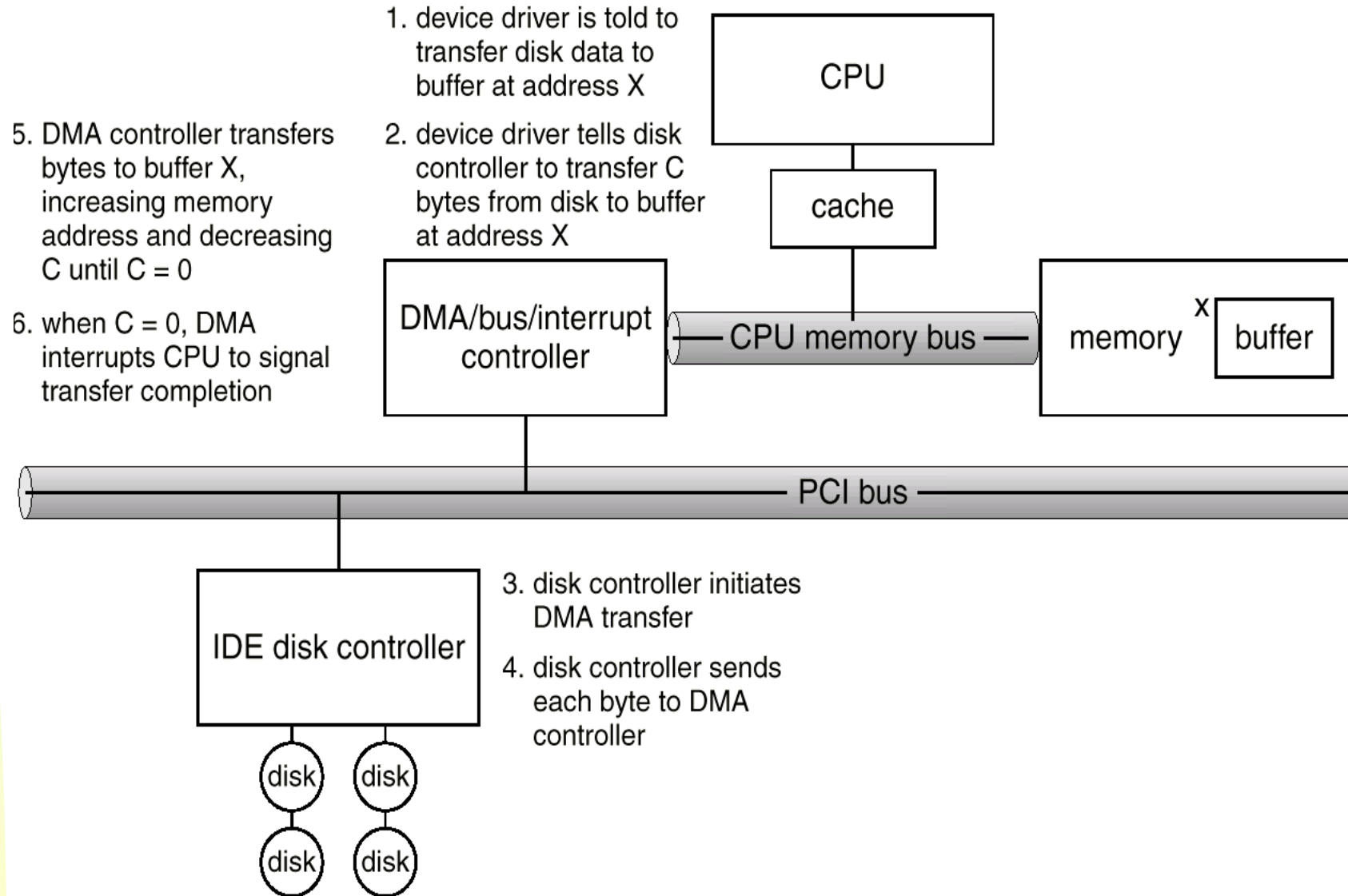
Adresses des ports d'E/S des périphériques PC

I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

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

DMA: six étapes

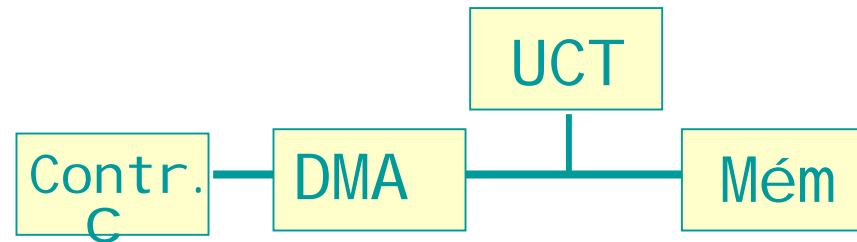


DMA: six étapes

- 1- CPU demande au pilote du périphérique (disque) (software) de transférer les données du disque au buffer à l'adresse x
- 2 - Le pilote du disque demande au contrôleur du disque (hardware) 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 transfère 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

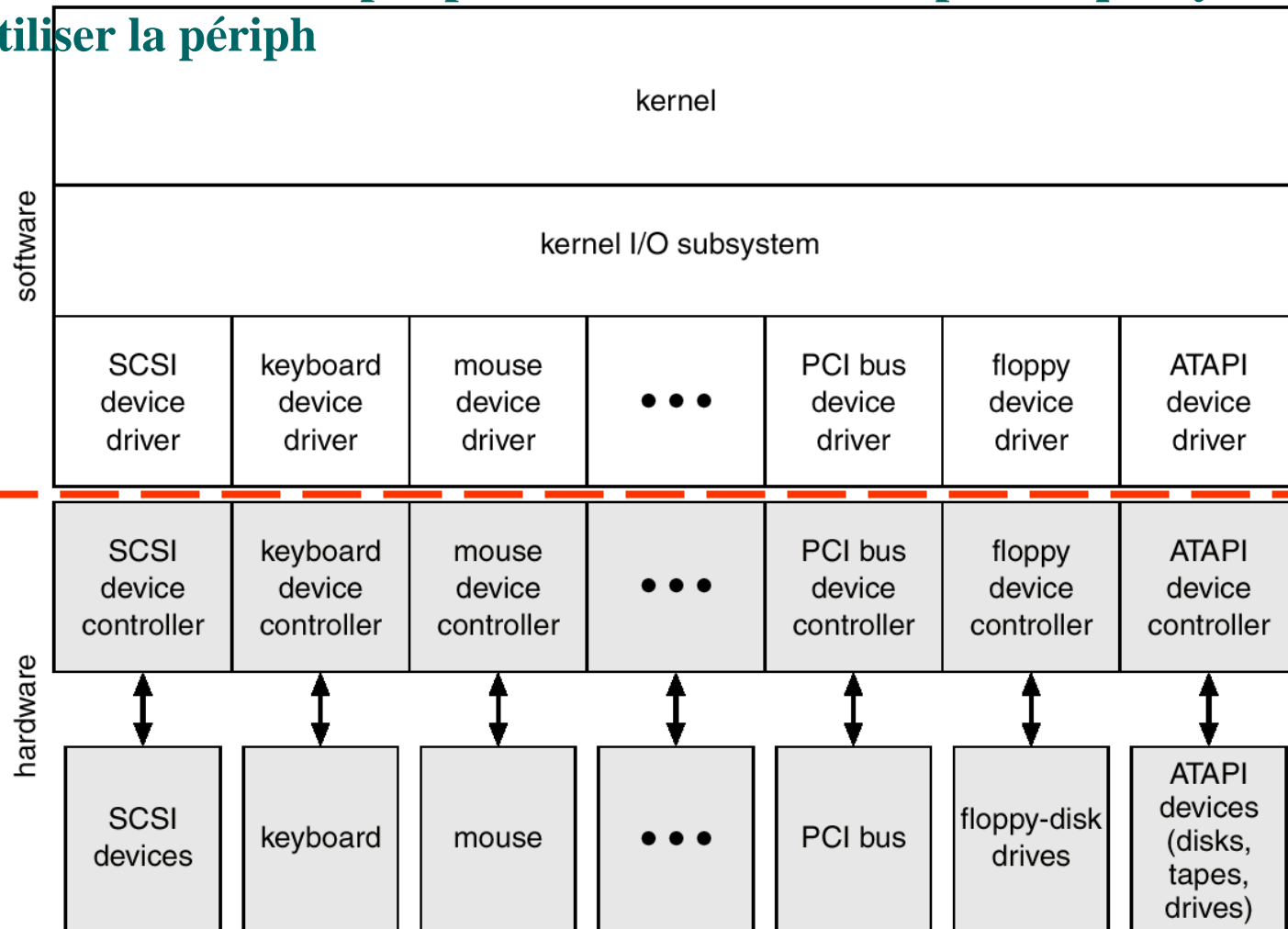


- Mais beaucoup moins que sans DMA, quand l'UCT doit s'occuper de gérer le transfert
Mémoire <-> Périphérique



Structure E/S d'un noyau (driver=pilote)

- Le système d'E/S encapsule le comportement des périphériques dans les pilotes
- La couche pilote de périfs masque les différences existantes entre périfs
- Mais le constructeur d'un périph doit créer autant de pilotes qu'il y a de SE qui peuvent utiliser la périph



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.

Sous-système E/S du noyau

■ **Fonctionnalités:**

◆ Ordonnancement E/S

- 👉 Optimiser l'ordre dans lequel les E/S sont exécutées

◆ Mise en tampon

◆ Mise en cache

◆ Mise en attente et réservation de périphérique spoule

◆ Gestion des erreurs

Sous-système E/S du noyau

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

Sous-système E/S du noyau

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.

Sous-système E/S du noyau

Mise en attente et réservation de périphérique: spoule

- **Spoule 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

Sous-système E/S du noyau

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

2- 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**

Qu'est qu'un fichier

- **Collection nommée d'informations apparentées, enregistrée sur un stockage secondaire**
 - ◆ Nature permanente
- **Les données qui se trouvent sur un stockage secondaires doivent être dans un fichier**
- **Différents types:**
 - ◆ Données (binaire, numérique, caractères....)
 - ◆ Programmes

Structures de fichiers

- **Aucune – séquences d'octets...**
- **Texte: Lignes, pages, documents formatés**
- **Source: programmes...**
- **Etc.**

Attributs d'un fichier

- **Constituent les propriétés du fichiers et sont stockés dans un fichier spécial appelé répertoire (directory). Exemples d'attributs:**
 - ◆ **Nom:**
 - ☞ pour permet aux personnes d'accéder au fichier
 - ◆ **Identificateur:**
 - ☞ Un nombre permettant au SE d'identifier le fichier
 - ◆ **Type:**
 - ☞ Ex: binaire, ou texte; lorsque le SE supporte cela
 - ◆ **Position:**
 - ☞ Indique le disque et l'adresse du fichier sur disque
 - ◆ **Taille:**
 - ☞ En bytes ou en blocs
 - ◆ **Protection:**
 - ☞ Détermine qui peut écrire, lire, exécuter...
 - ◆ **Date:**
 - ☞ pour la dernière modification, ou dernière utilisation
 - ◆ **Autres...**

Opérations sur les fichiers: de base

- **Création**
- **Écriture**
 - ◆ Pointeur d'écriture qui donne la position d'écriture
- **Lecture**
 - ◆ Pointeur de lecture
- **Positionnement dans un fichier (temps de recherche)**
- **Suppression d'un fichier**
 - ◆ Libération d'espace

Autres opérations

- **Ajout d'infos**
- **Rénommage**
- **Copie**
 - ◆ peut être faite par renommage: deux noms pour un seul fichier
- **Ouverture d'un fichier: le fichier devient associé à un processus qui en garde les attributs, position, etc.**
 - ◆ Pointeurs de fichier
 - ☞ Pour accès séquentiel
 - ☞ P.ex. pour read, write
 - ◆ Compteur d'ouvertures
 - ◆ Emplacement
- **Fermeture**

Types de fichiers

- **Certains SE utilisent l'extension du nom du fichier pour identifier le type.**
 - ◆ Microsoft: Un fichier exécutable doit avoir l'extension .EXE, .COM, ou .BAT (sinon, le SE refusera de l'exécuter)
- **Le type n'est pas défini pour certains SE**
 - ◆ Unix: l'extension est utilisée (et reconnue) seulement par les applications
- **Pour certains SE le type est un attribut**
 - ◆ MAC-OS: le fichier a un attribut qui contient le nom du programme qui l'a généré (ex: document Word Perfect)

Types de fichiers

file type	usual extension	function
executable	exe, com, bin or none	read 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, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	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	binary file containing audio or A/V information

Structure logique des fichiers

- **Le type d'un fichier spécifie sa structure**
 - ◆ Le SE peut alors supporter les différentes structures correspondant aux types de fichiers
 - ☞ Cela rend plus complexe le SE mais simplifie les applications
- **Généralement, un fichier est un ensemble d'enregistrements (records)**
 - ◆ Chaque enregistrement est constitué d'un ensemble de champs (fields)
 - ☞ Un champ peut être numérique ou chaîne de chars.
 - ◆ Les enregistrements sont de longueur fixe ou variable (tout dépendant du type du fichier)
- **Mais pour Unix, MS-DOS et autres, un fichier est simplement une suite d'octets « byte stream »**
- **C'est l'application qui interprète le contenu et spécifie une structure**

Méthodes d'accès

Séquentielle
Indexée
Directe

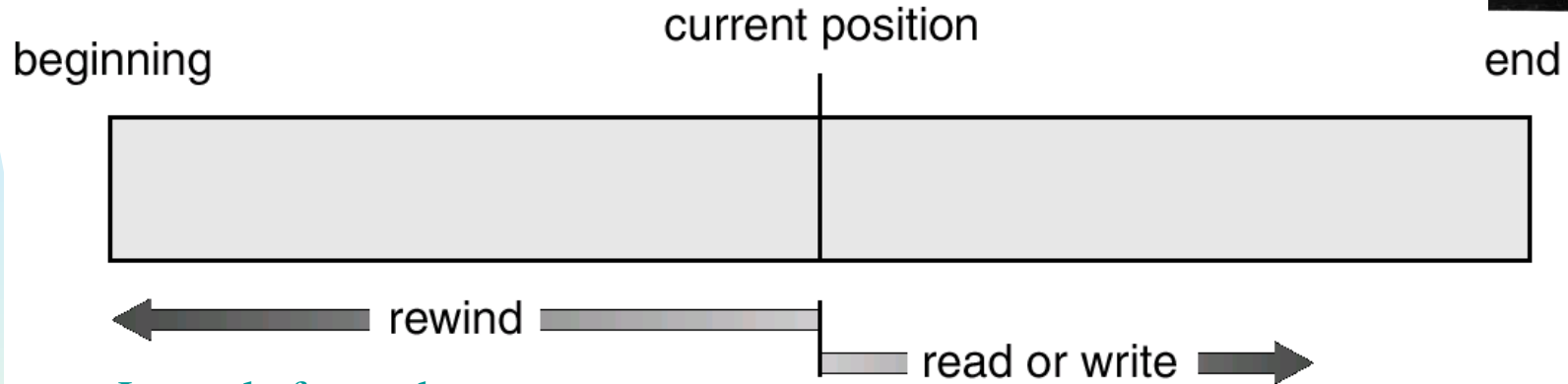
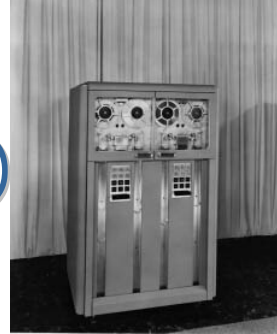
Méthodes d'accès: 4 de base

- **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
- **Pas tous les SE supportent les méthodes d'accès**
 - ◆ Quand le SE ne les supporte pas, c'est à l'application de les supporter

Méthodes d'accès aux fichiers

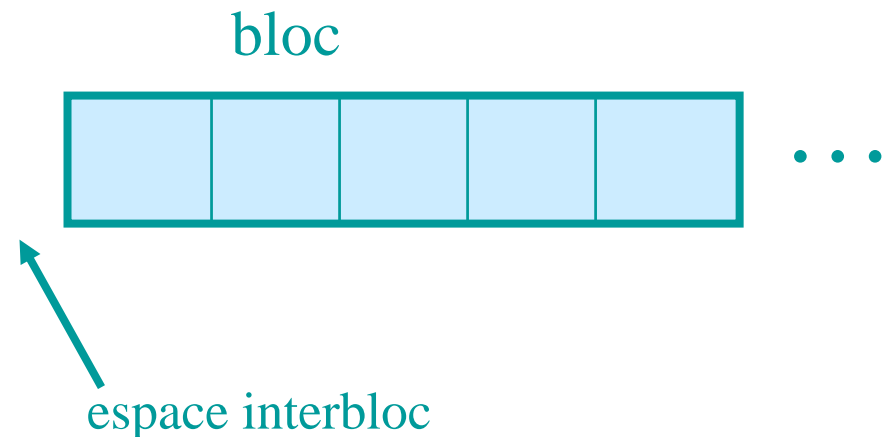
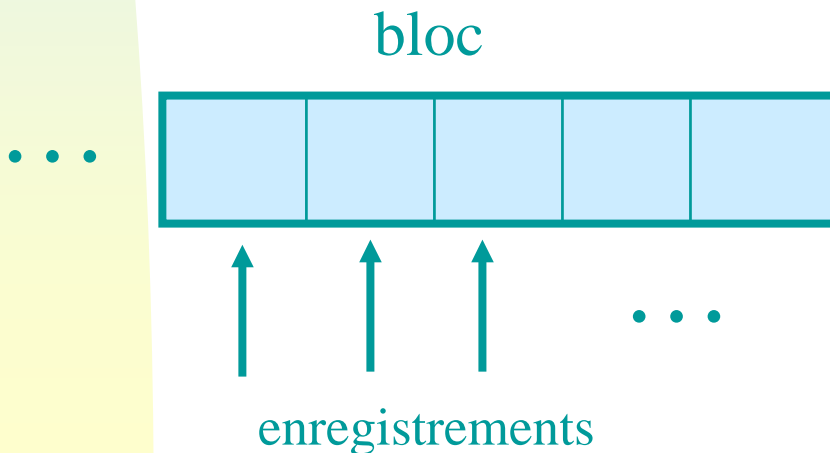
- **La structure logique d'un fichier détermine sa méthode d'accès**
- **Plusieurs SE modernes (Unix, Linux, MS-DOS...) fournissent une seule méthode d'accès (séquentielle) car les fichiers sont tous du même type (ex: séquence d'octets)**
 - ◆ Mais leur méthode d'allocation de fichiers permet habituellement aux applications d'accéder aux fichiers de différentes manières
- **Ex: les systèmes de gestions de bases de données (DBMS) requièrent des méthodes d'accès plus efficaces que juste séquentielle**

Fichiers à accès séquentiel (archétype: rubans)



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

En avant seulement, 1 seul enreg. à la fois



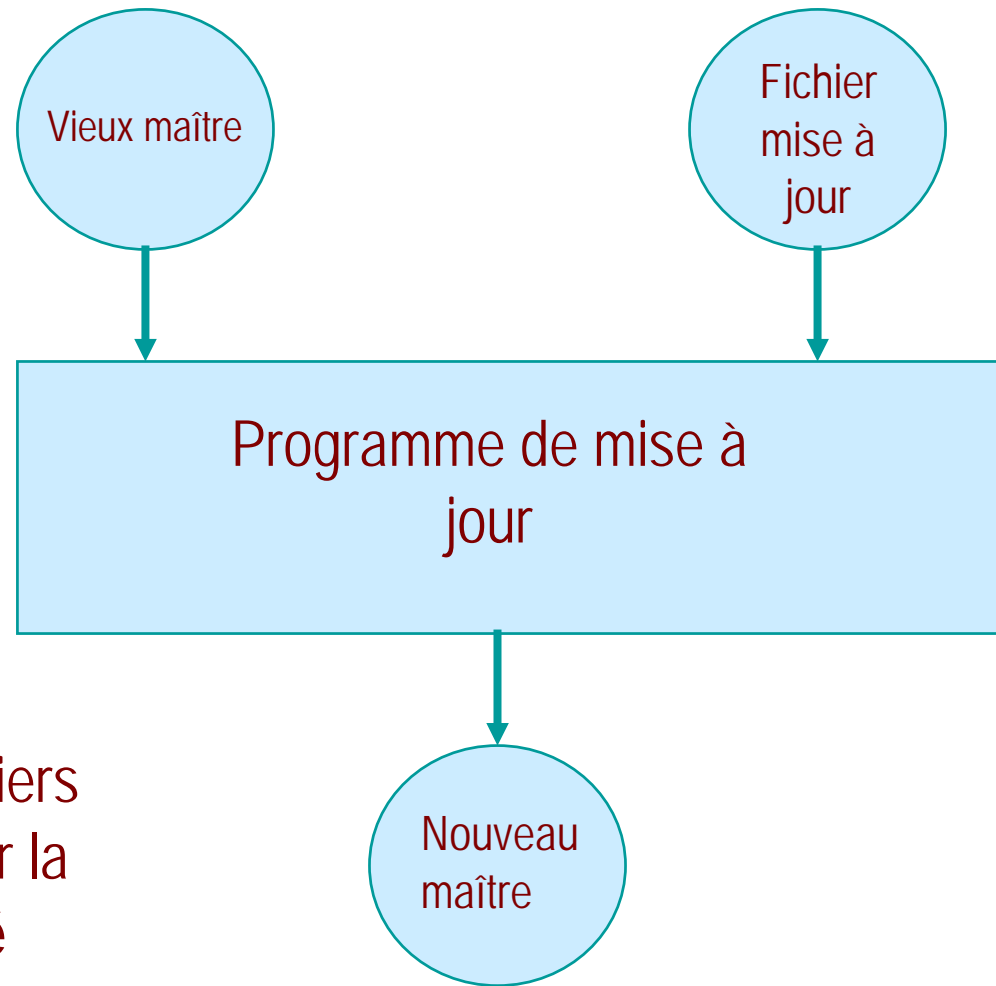
Lecture physique et lecture logique dans un fichier séquentiel

- Un fichier séquentiel consiste en **blocs** d'octets enregistrés sur un support tel que ruban, disque...
- La dimension de ces blocs est dictée par les caractéristiques du support
- Ces blocs sont lus (lecture physique) dans un tampon en mémoire
- Un bloc contient un certain nombre **d'enregistrements (records)** qui sont des unités d'information logiques pour l'application (un étudiant, un client, un produit...)
 - ◆ Souvent de longueur et contenu uniformes
 - ◆ Triés par une *clé*, normalement un code (code d'étudiant, numéro produit...)
- Une lecture dans un programme lit le prochain **enregistrement**
- Cette lecture peut être réalisée par
 - ◆ La simple mise à jour d'un pointeur si la lecture logique précédente ne s'était pas rendue à la fin du tampon
 - ◆ La lecture du prochain bloc (dans un tampon d'E/S en mémoire) si la lecture logique précédente s'était rendue à la fin du tampon
 - ☞ Dans ce cas le pointeur est remis à 0

Autres propriétés des fichiers séquentiels

- Pour l'écriture, la même idée: une instruction d'écriture dans un programme cause l'ajout d'un enregistrement à un tampon, quand le tampon est plein il y a une écriture de bloc
- Un fichier séquentiel qui a été ouvert en lecture ne peut pas être écrit et vice-versa (impossible de mélanger lectures et écritures)
- Les fichiers séquentiels ne peuvent être lus ou écrits qu'un enregistrement à la fois et seulement dans la direction 'en avant'

Mise à jour de fichiers séquentiels



Tous les fichiers
sont triés par la
même **clé**

Mise à jour de fichiers séquentiels triés: exemple



02
05
12
17
21
26

Retirer 5
Modif 12
Ajout 20
Ajout 27

02
12
17
20
21
26
27

(12 a été modifié)

Vieux maître + Mises à jour = Nouveau maître

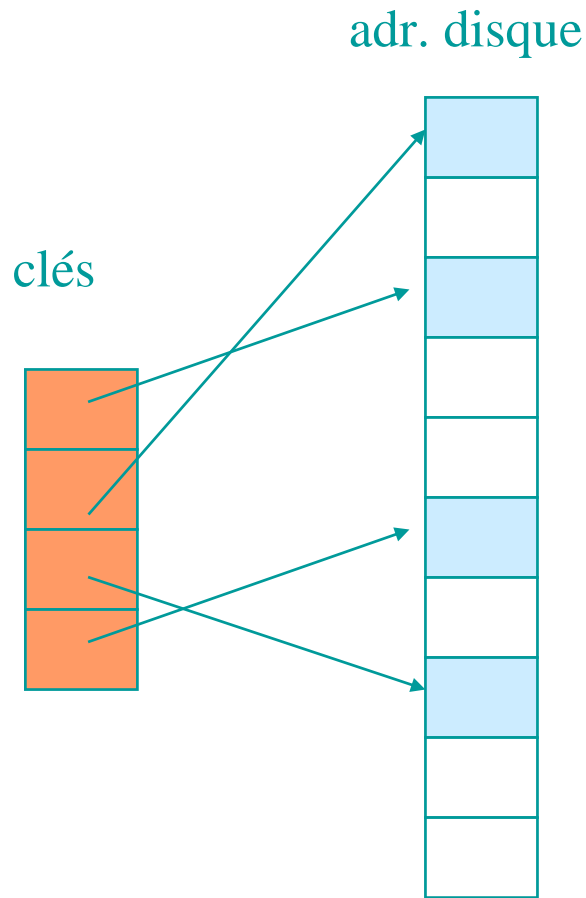
L'algorithme fonctionne lisant un enregistrement à la fois, en séquence,
du vieux maître et du fichier des mises à jour

Accès direct ou haché ou aléatoire:

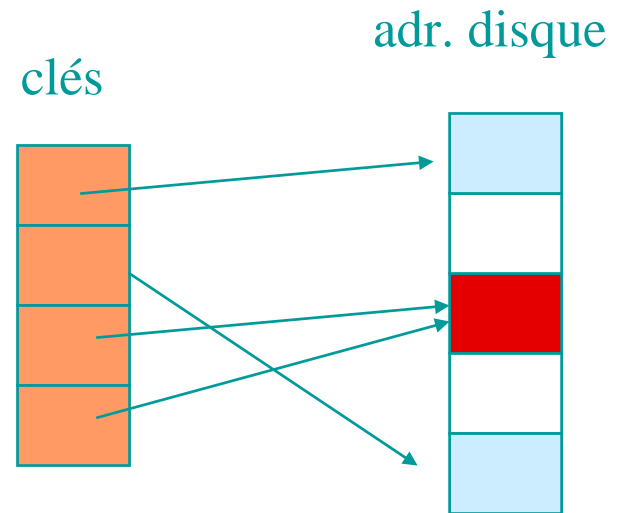
accès direct à travers tableau d'hachage

- **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



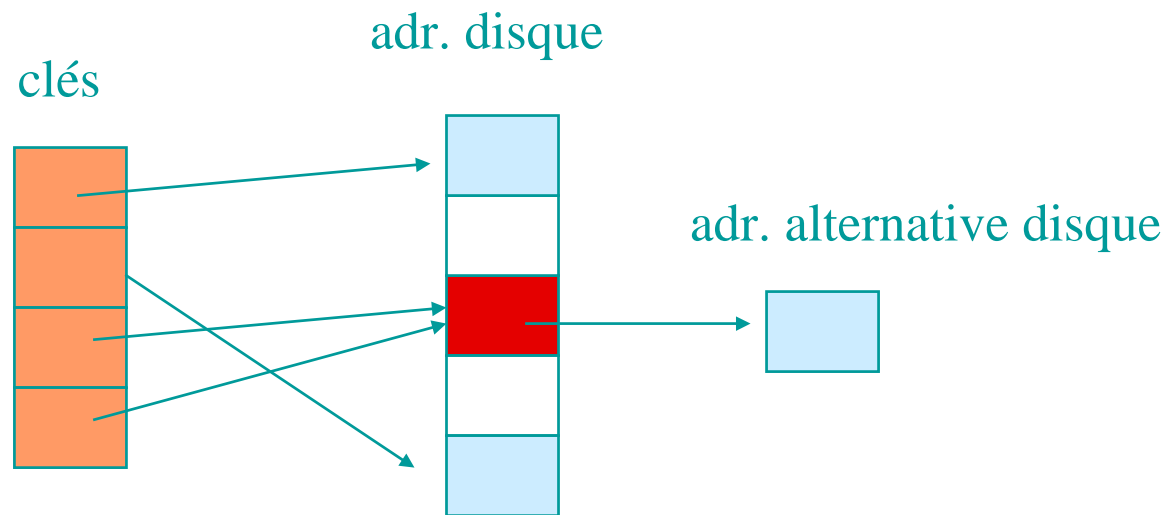
Fonction d'hachage dispersée
qui n'utilise pas bien l'espace
disponible



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

Hachage: Traitement des doubles affectations

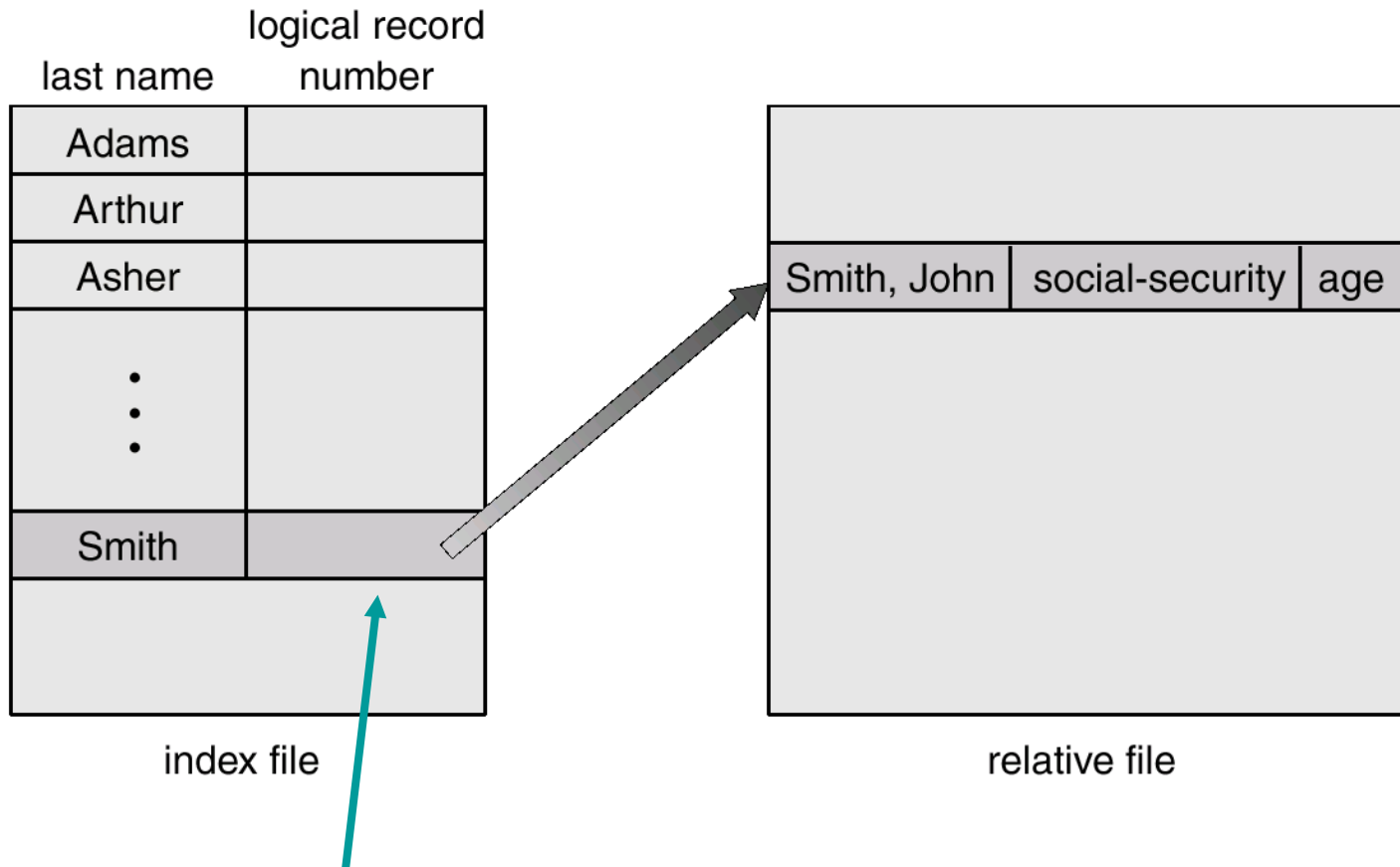
- **On doit détecter les doubles affectations, et s'il y en a, un des deux enregistrements doit être mis ailleurs**
 - ◆ ce qui complique l'algorithme



Adressage Indexé séquentiel (index sequential)

- **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)

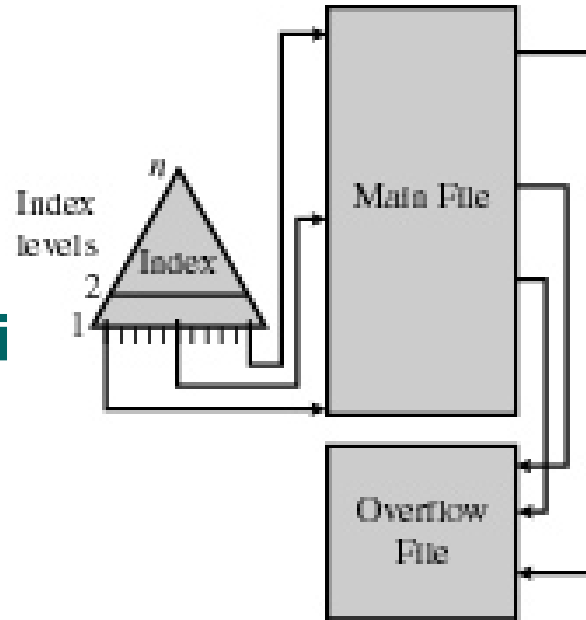
Le fichier index est à accès direct, le fichier relatif est à accès séquentiel

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

Mais... besoin de fichier débordement

- Les nouveaux enregistrements seront ajoutés à un fichier débordement
- Les enregistrements du fichier principal qui le précèdent dans l'ordre de tri seront mis à jour pour contenir un pointeur au nouveau enregistrement
 - Donc accès additionnels au fichiers débordement
- Périodiquement, le fichier principal sera fusionné avec le fichier débordement



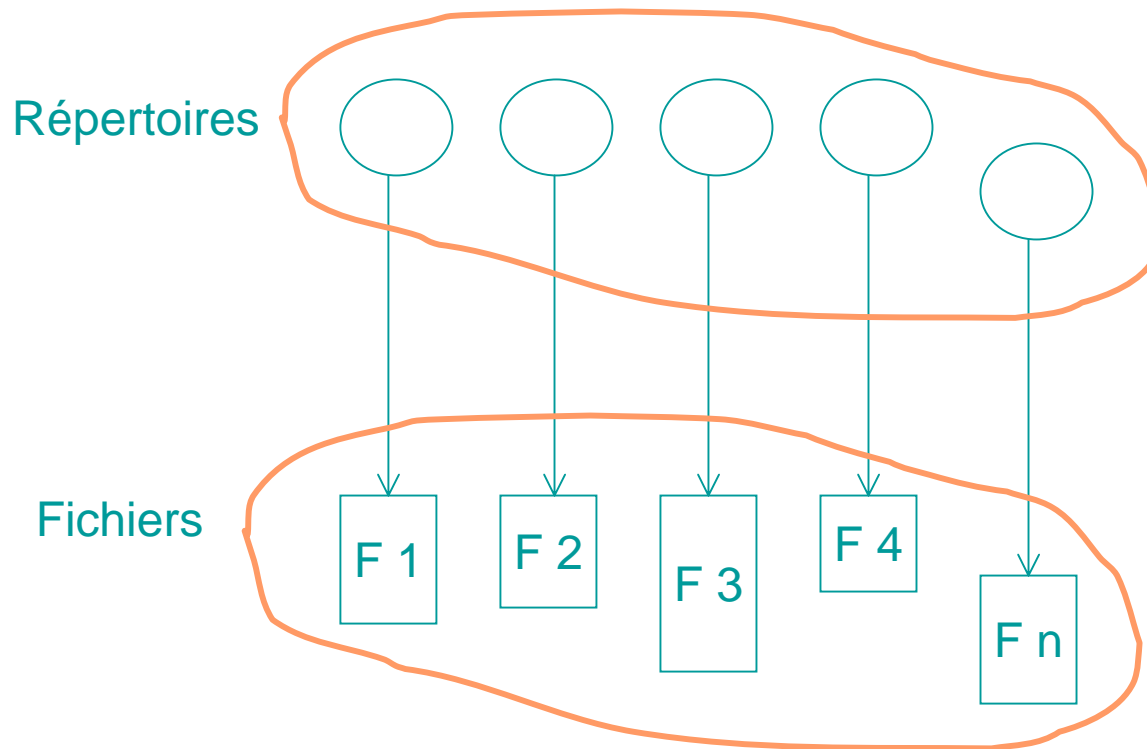
Utilisation des 4 méthodes

- **Séquentiel (rubans ou disques): lecture ou écriture des enregistrements dans un ordre fixe**
 - Pour travaux 'par lots': salaires, comptabilité périodique...
- **Indexé séquentiel (disques): accès séquentiel ou accès direct par l'utilisation d'index**
 - Pour fichiers qui doivent être consultés parfois de façon séquentielle, parfois de façon directe (p.ex. par nom d'étudiant)
- **Indexée: multiplicité d'index selon les besoins, accès direct par l'index**
 - Pour fichiers qui doivent être consultés de façon directe selon des critères différents (p.ex. pouvoir accéder aux infos concernant les étudiants par la côte du cours auquel ils sont inscrits)
- **Direct ou hachée: accès direct à travers tableau d'hachage**
 - Pour fichiers qui doivent être consultés de façon directe par une clé uniforme (p.ex. accès aux informations des étudiants par No. de matricule)

Répertoires

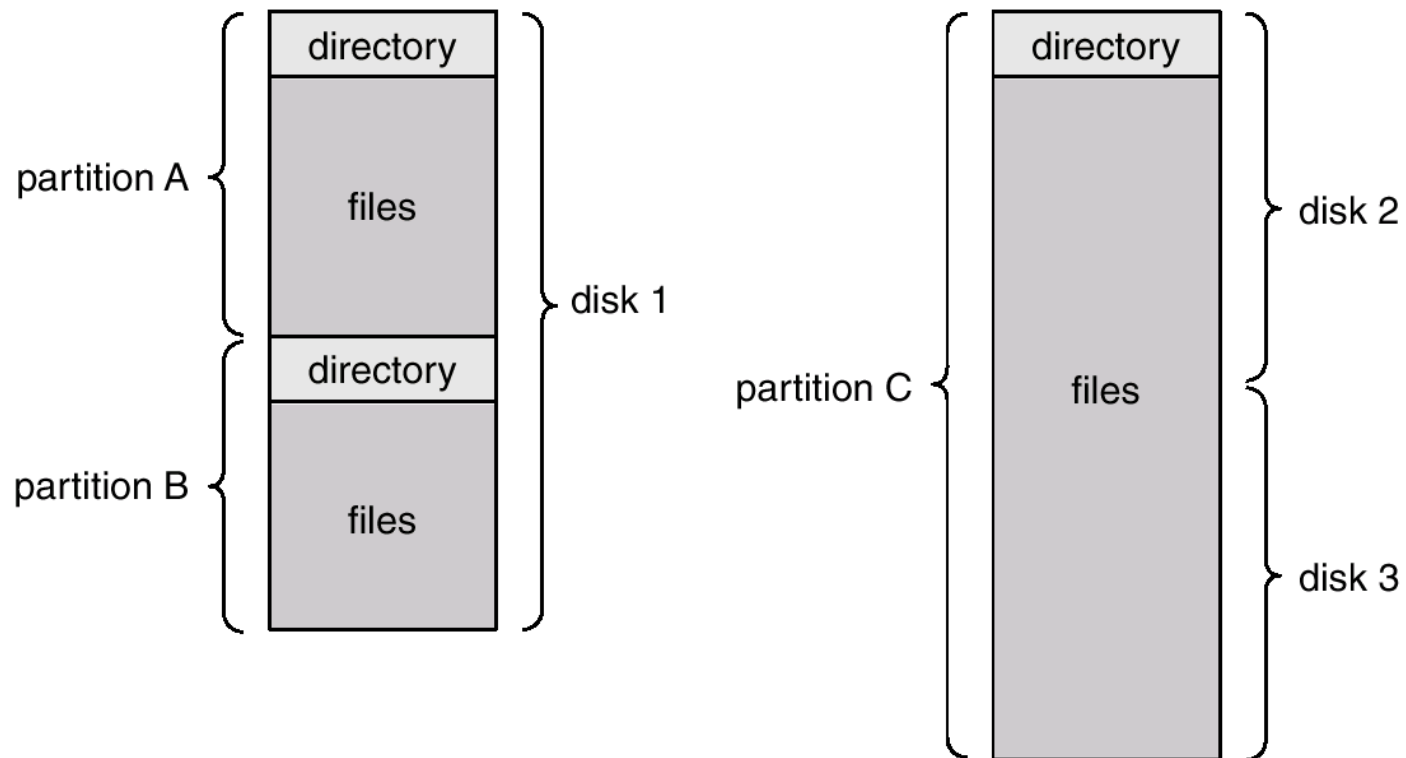
Structures de répertoires (directories)

- Une collection de structures de données contenant infos sur les fichiers.



- Tant les répertoires, que les fichiers, sont sur disques
- À l'exception d'un répertoire racine en mémoire centrale

Organisation typique de système de fichiers



Information dans un répertoire

- **Nom du fichier**
- **Type**
- **Adresse sur disque, sur ruban...**
- **Longueur courante**
- **Longueur maximale**
- **Date de dernier accès**
- **Date de dernière mise à jour**
- **Propriétaire**
- **Protection**

Opérations sur répertoires

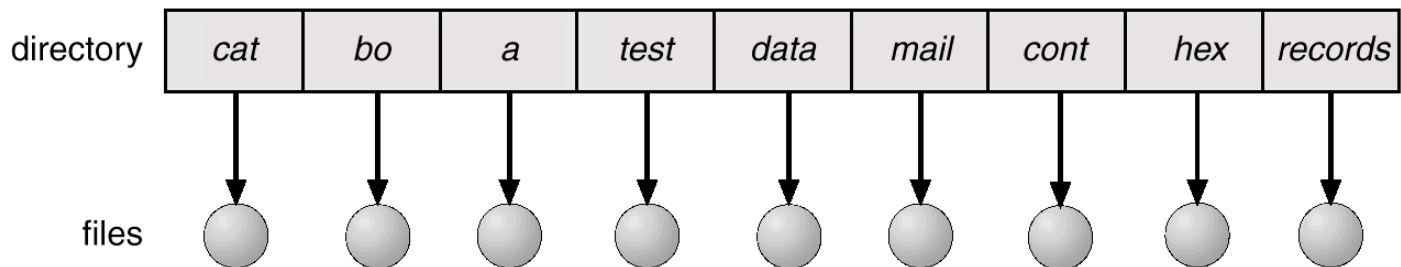
- **Recherche de fichier**
- **Création de fichier**
- **Suppression de fichier**
- **Lister un répertoire**
- **Rénommer un fichier**
- **Traverser un système de fichier**

Organisation de répertoires

- **Efficacité: arriver rapidement à un enregistrement**
- **Structure de noms: convenable pour usager**
 - ◆ deux usagers peuvent avoir le même noms pour fichiers différents
 - ◆ Le même fichier peut avoir différents noms
- **Groupement de fichiers par type:**
 - ◆ tous les programmes Java
 - ◆ tous les programmes objet

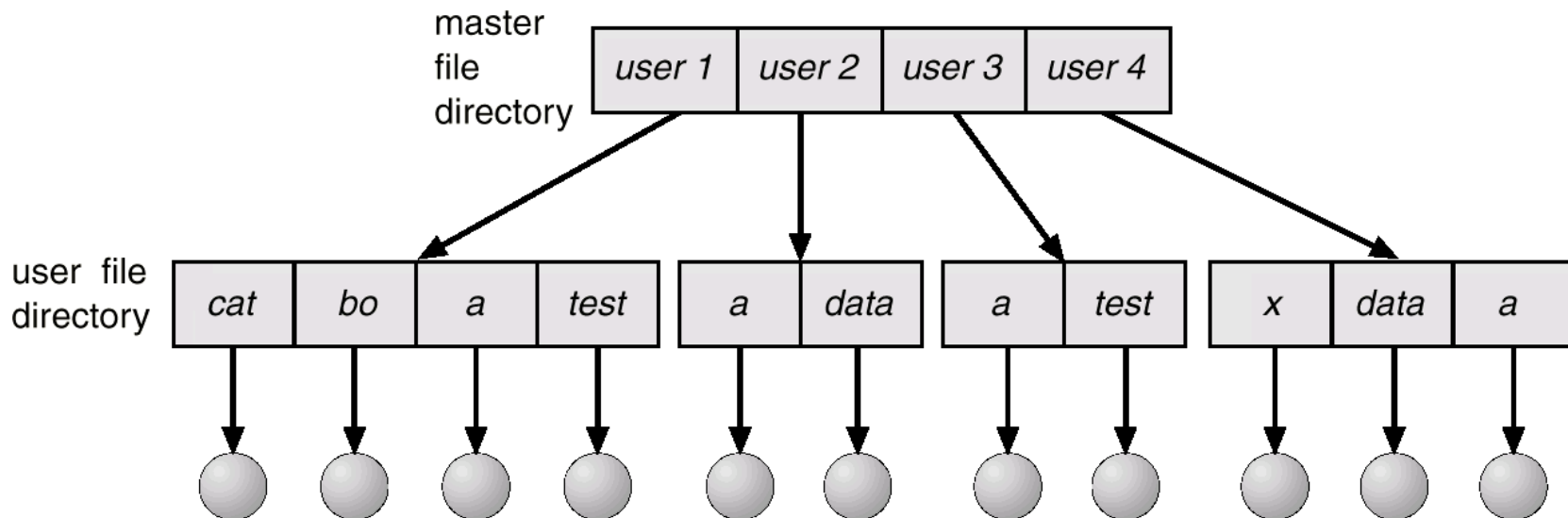
Structure à un niveau

- **Un seul répertoire pour tous les usagers**
- **Ambiguïté de noms**
- **Problèmes de groupement**
- **Primitif, pas pratique**

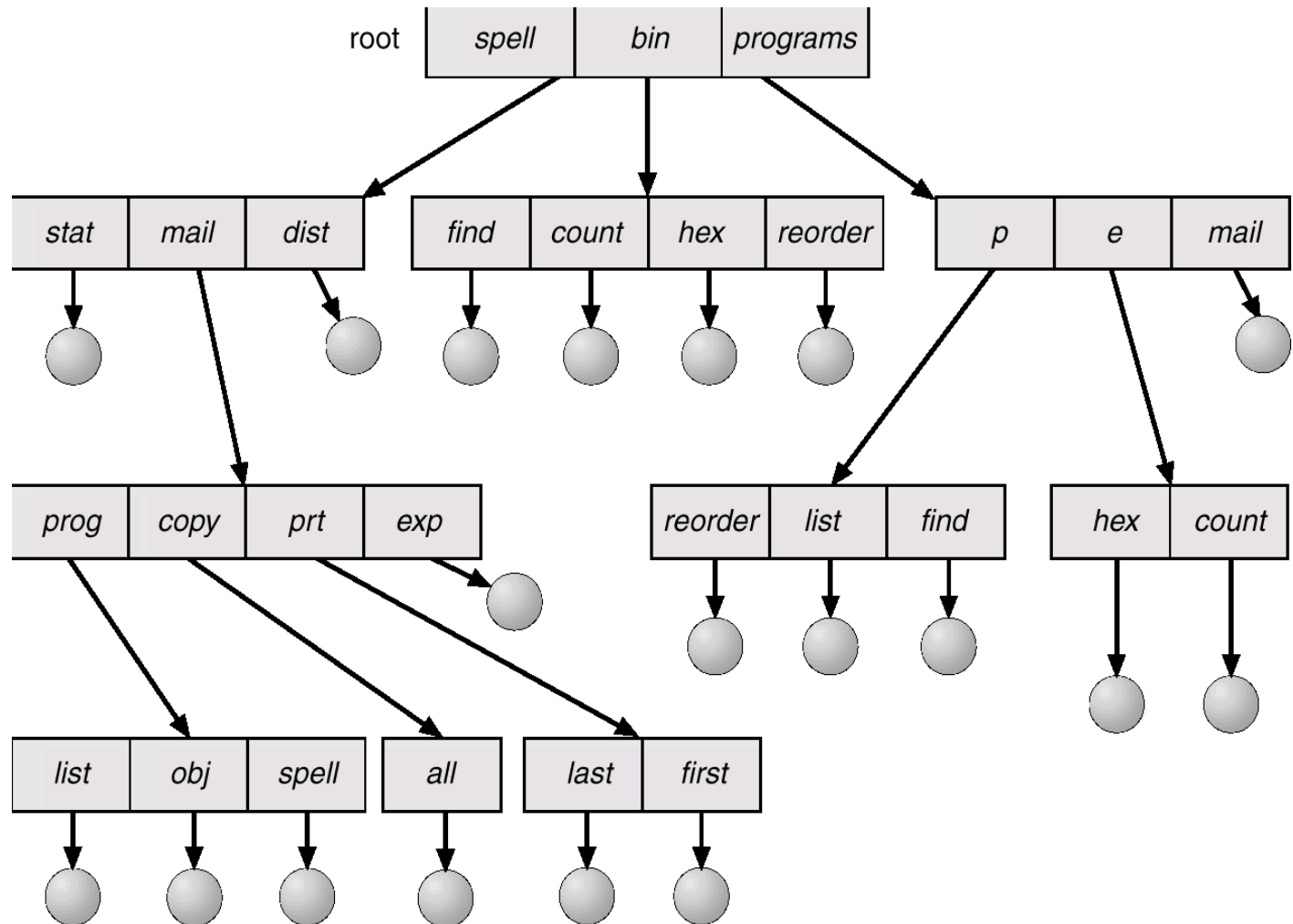


Répertoires à deux niveaux

- Répertoire séparé pour chaque usager
- 'path name', nom de chemin
- même nom de fichier pour usagers différents est permis
- recherche efficace
- Pas de groupements



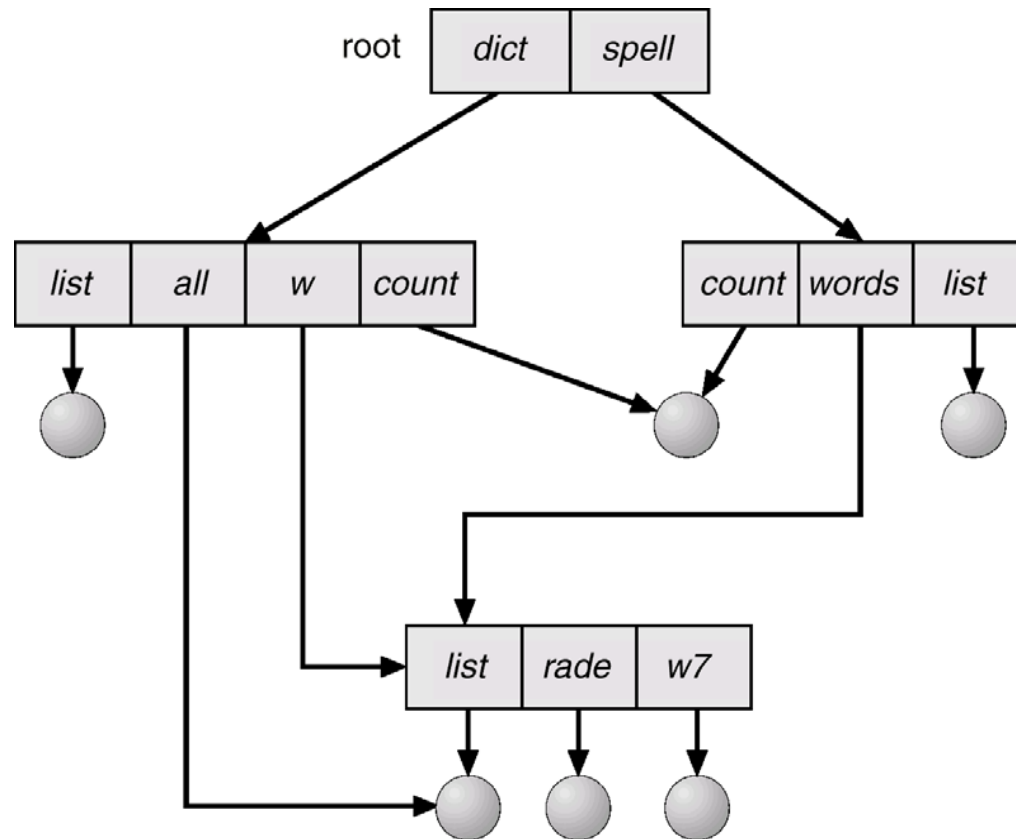
Répertoires à arbres (normal aujourd'hui)



Caractéristiques des répertoires à arbres

- **Recherche efficace**
- **Possibilité de grouper**
- **Repertoire courant (working directory)**
 - ◆ **cd /spell/mail/prog**

Graphes sans cycles: permettent de partager fichiers



Unix: *symbolic link* donne un chemin à un fichier ou sous-répertoire

Partage de fichiers

- **Désirable sur les réseaux**
- **Nécessité de protection**

Protection

- **Types d 'accès permis**
 - ◆ **lecture**
 - ◆ **écriture**
 - ◆ **exécution**
 - ◆ **append (annexation)**
 - ◆ **effacement**
 - ◆ **listage: lister les noms et les attributs d 'un fichier**

Listes et groupes d'accès - UNIX

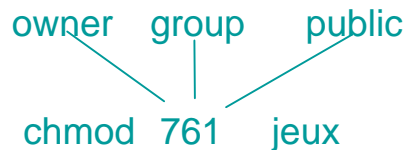
- **Modes d'accès: R W E**
- **Trois classes d'usager:**
 - ◆ propriétaire
 - ◆ groupe
 - ◆ public
- **Demander à l'administrateur de créer un nouveau groupe avec un certain usager et un certain propriétaire**

Listes et groupes d'accès

- **Mode d'accès: read, write, execute**
- **Trois catégories d'utilisateurs:**

			RWX
a) owner access	7	⇒	1 1 1
			RWX
b) group access	6	⇒	1 1 0
			RWX
c) others access	1	⇒	0 0 1

- **Demander au gestionnaire de créer un groupe, disons G, et ajouter des utilisateurs au groupe**
- **Pour un fichier particulier, disons jeux, définir un accès approprié**



Changer le groupe d'un fichier

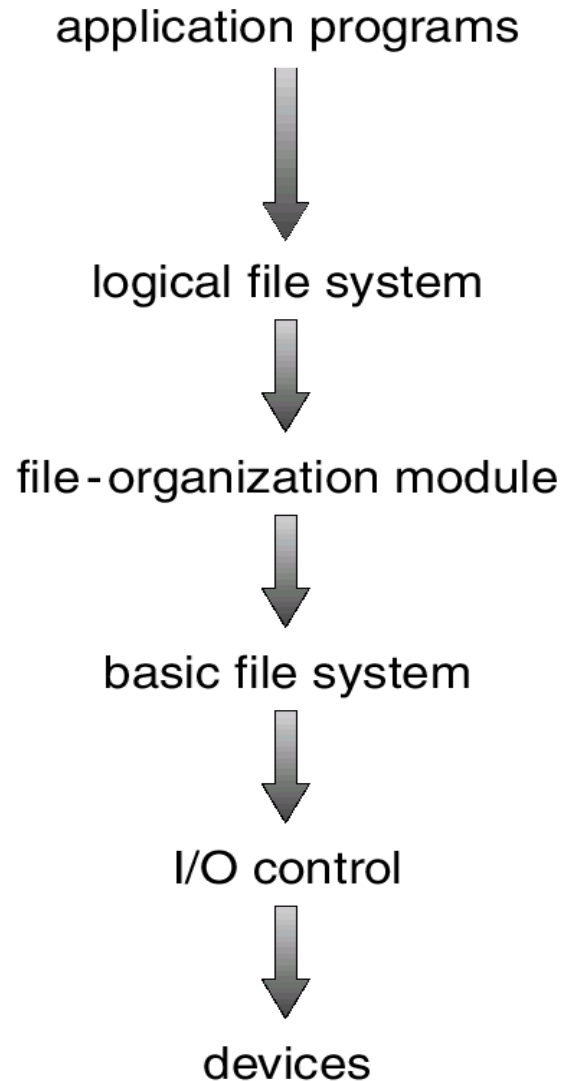
chgrp G jeux

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)**

Systemes de fichiers à couches



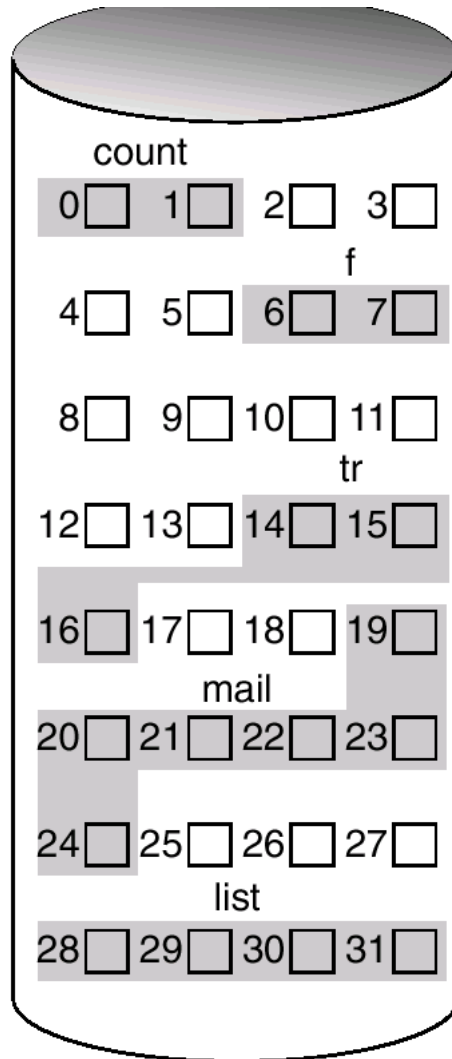
Structure physique des fichiers

- **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 bytes
- **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

Trois méthodes d'allocation de fichiers

- ◆ Allocation contiguë
- ◆ Allocation enchaînée
- ◆ Allocation indexée

Allocation contiguë sur disque



directory répertoire

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	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

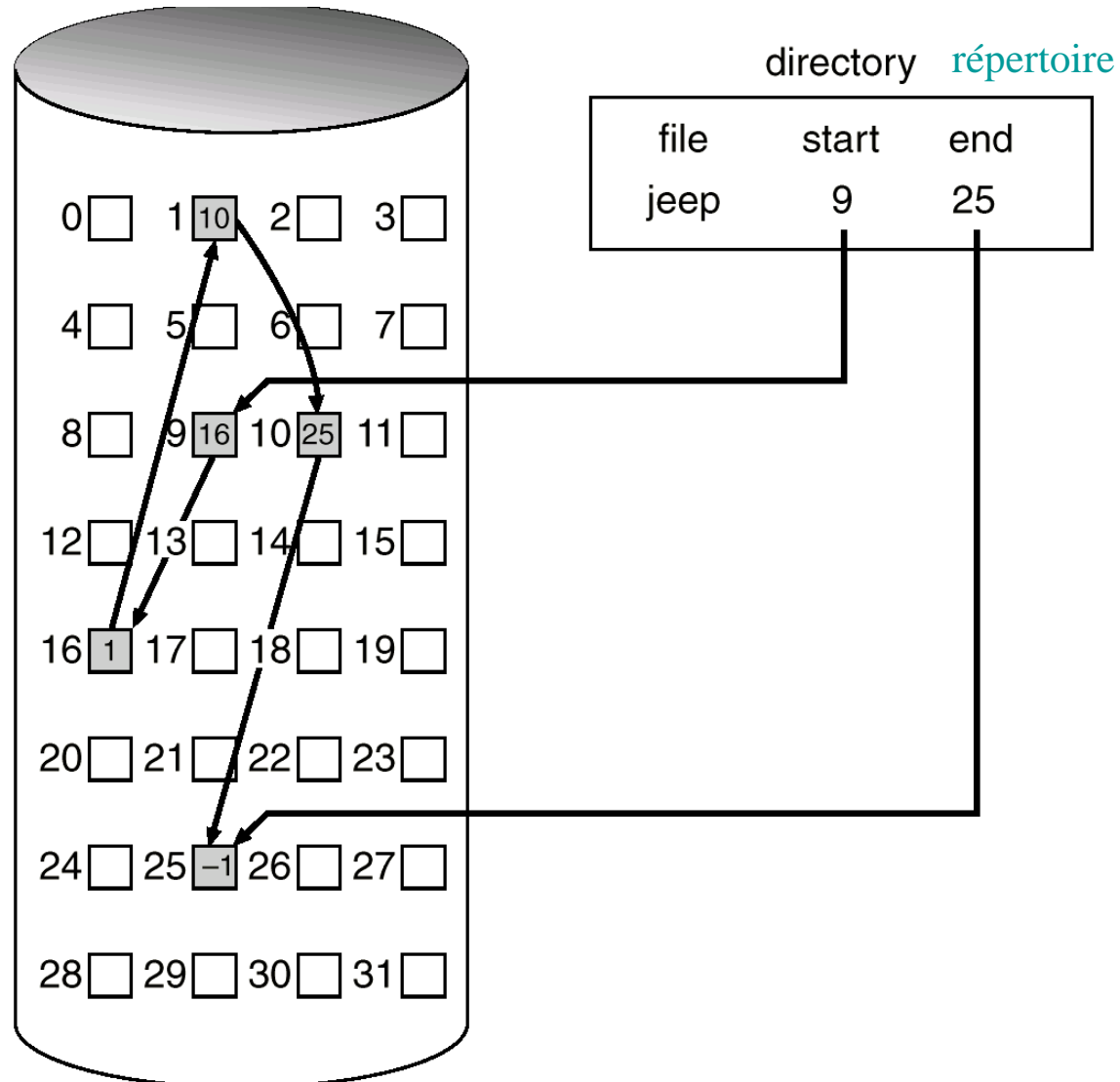
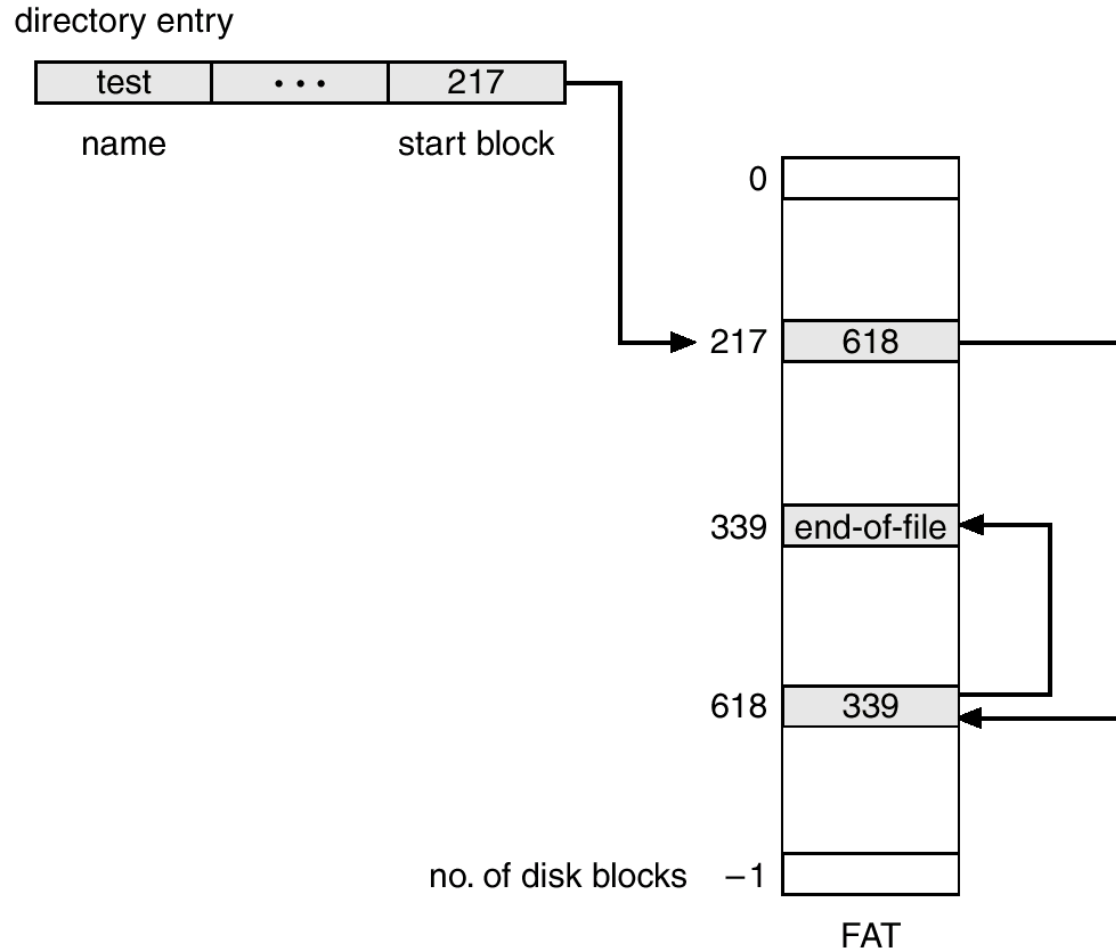


Tableau d'allocation de fichiers

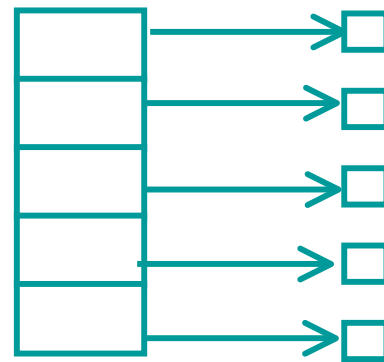


Avantages - désavantages

- **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**

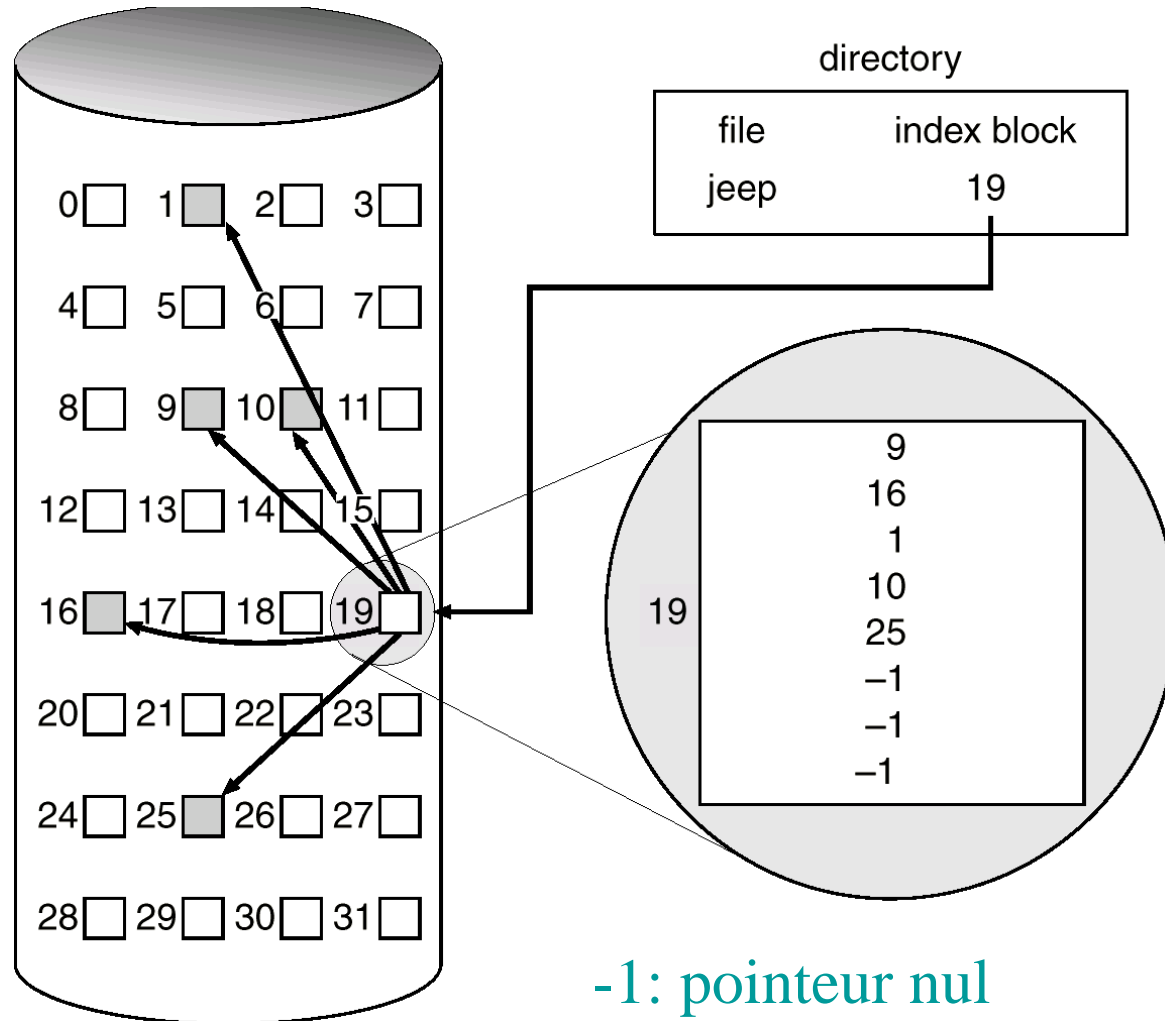
Allocation indexée: semblable à la pagination

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



index table

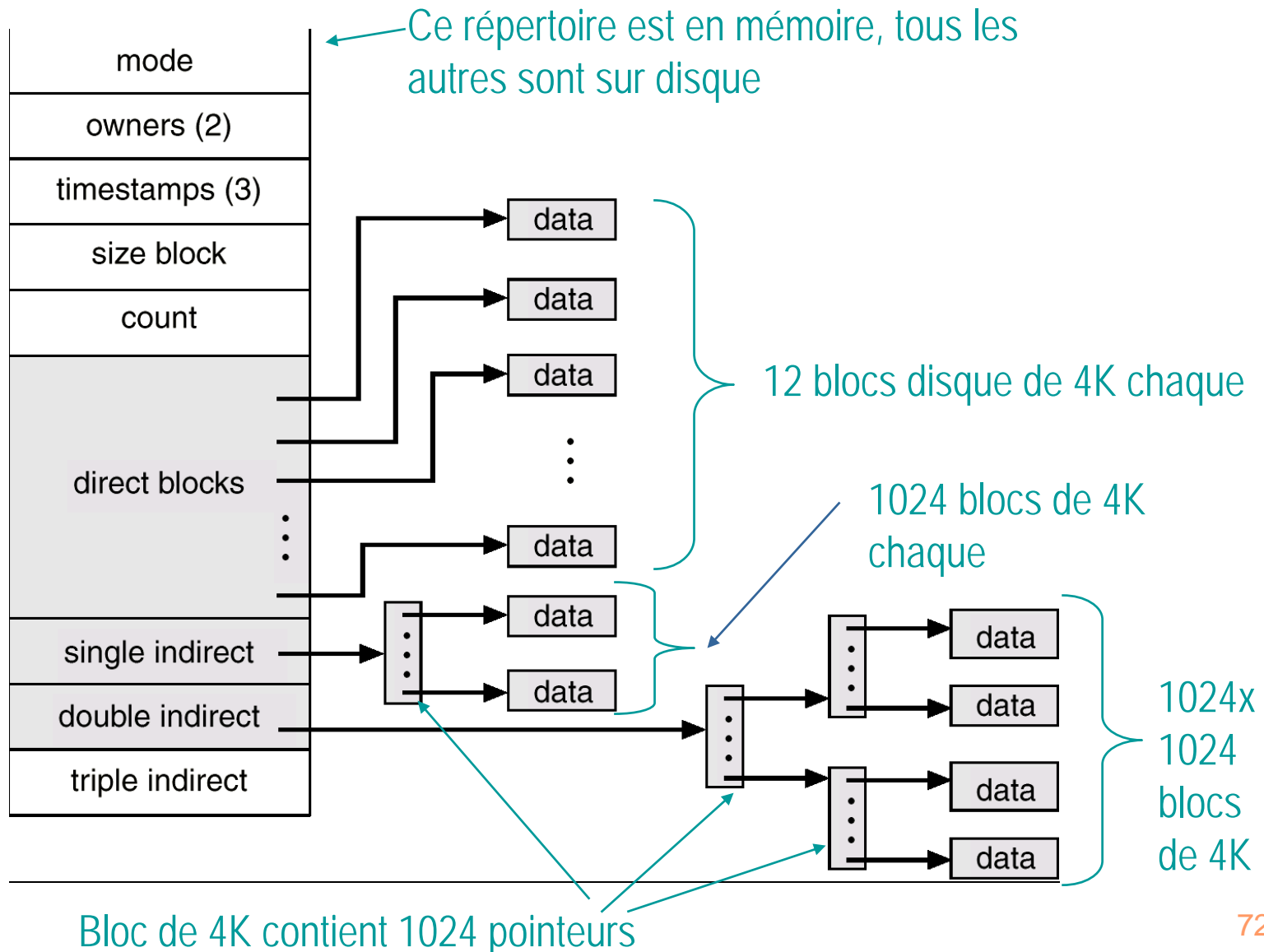
Allocation indexée



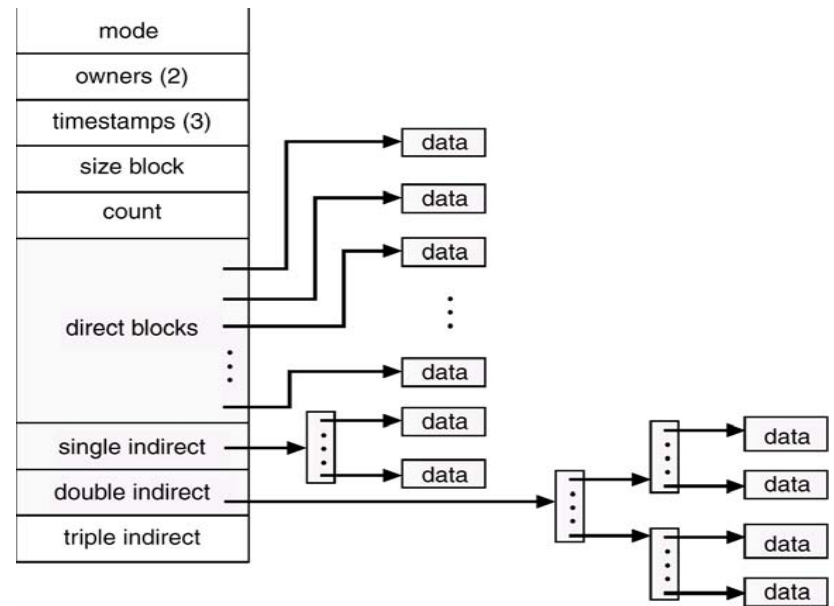
Allocation indexée

- 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)



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 fichiers avec un petit répertoire en mémoire

Gestion d'espace libre

Solution 1: vecteur de bits (solution Macintosh, Windows 2000)

- **Vecteur de bits (n blocs)**



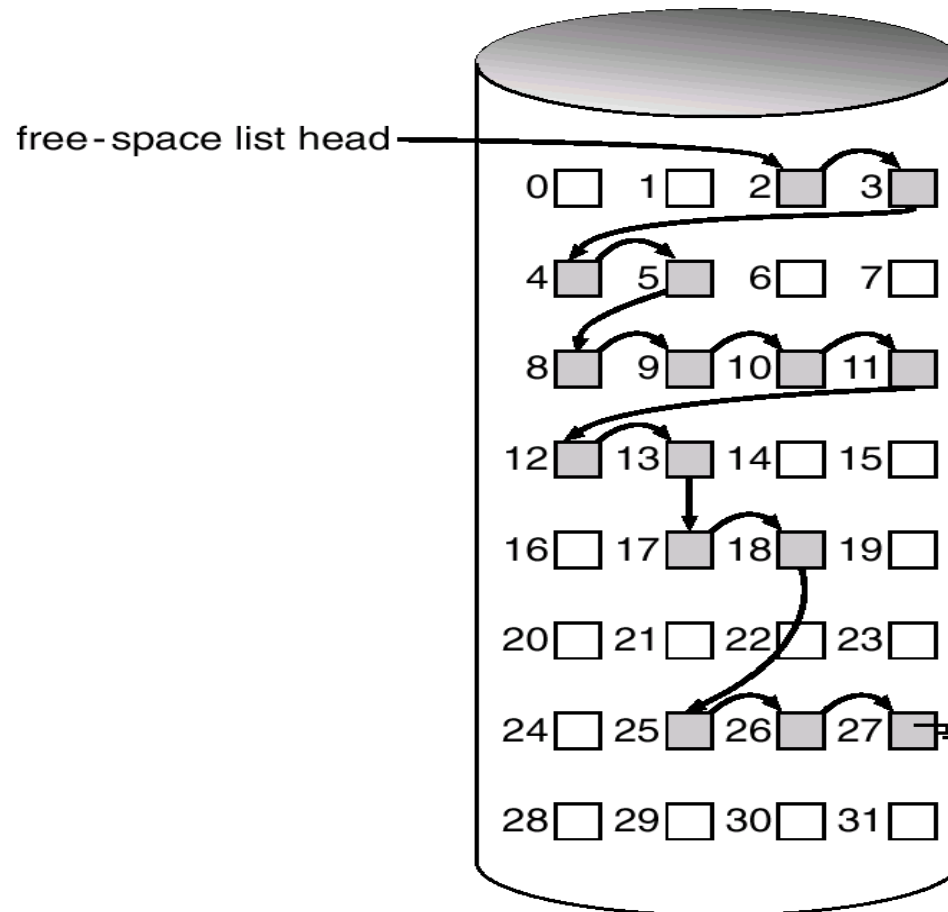
$\text{bit}[i] = \begin{matrix} 0 \\ 1 \end{matrix} \Rightarrow \begin{matrix} \text{block}[i] \text{ libre} \\ \text{block}[i] \text{ occupé} \end{matrix}$

- **Exemple d'un vecteur de bits où les blocs 3, 4, 5, 9, 10, 15, 16 sont occupés:**
 - ◆ 00011100011000011...
- **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

3. Structure de mémoire de masse (disques)

Concepts importants :

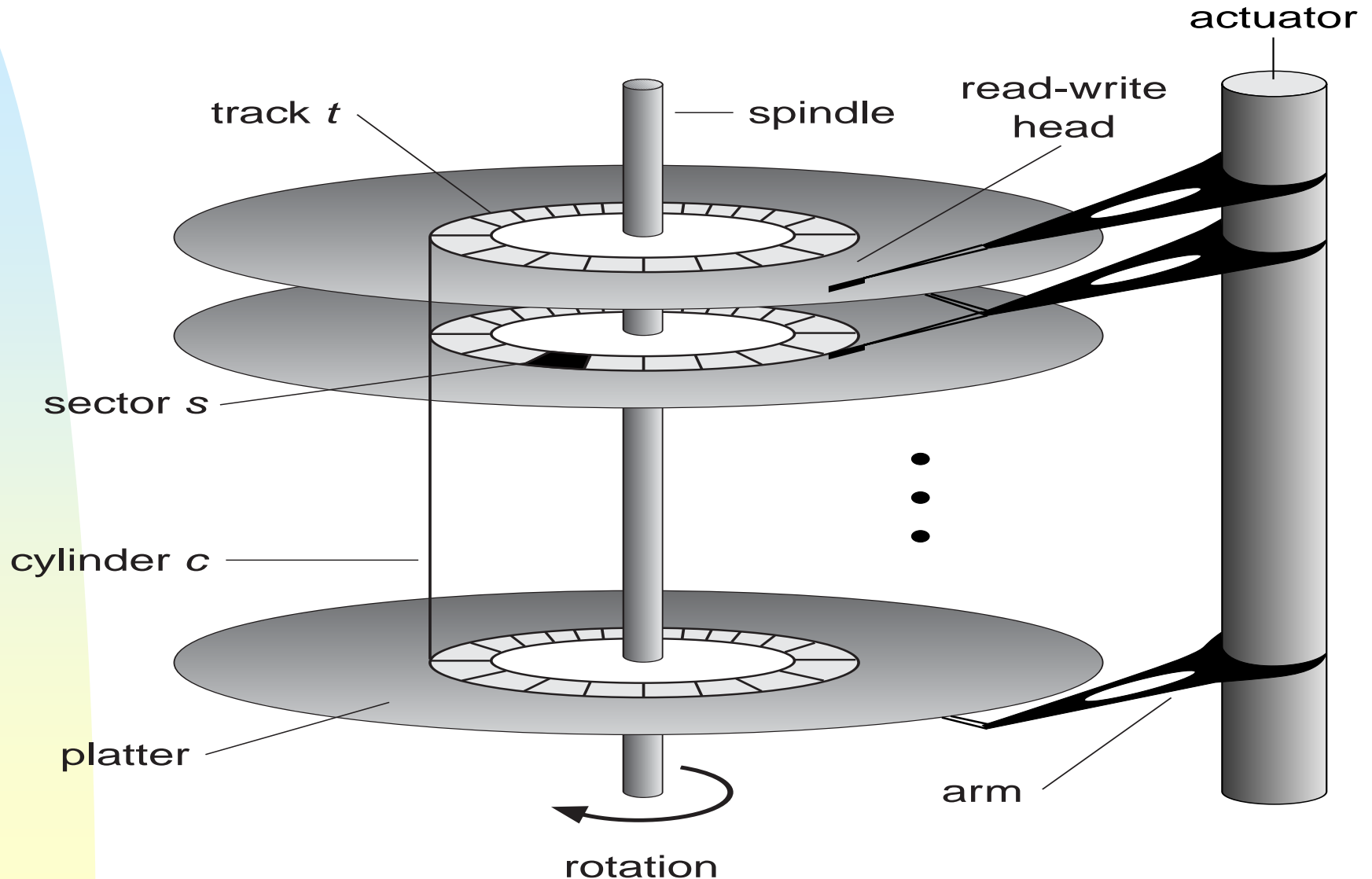
- **Fonctionnement et structure des unités disque**
- **Calcul du temps d'exécution d'une séquence d'opérations**
- **Différents algorithmes d'ordonnancement**
 - ◆ Fonctionnement, rendement
- **Gestion de l'espace de permutation**
 - ◆ Unix

Disques magnétiques

- **Plats rigides couverts de matériaux d'enregistrement magnétique**
 - ◆ surface du disque divisée en **pistes** (tracks) qui sont divisées en **secteurs**
 - ◆ le contrôleur disque détermine l'interaction logique entre l'unité et l'ordinateur

Nomenclature -

cylindre: l'ensemble de pistes qui se trouvent dans la même position du bras de lecture/écriture



Disques électroniques

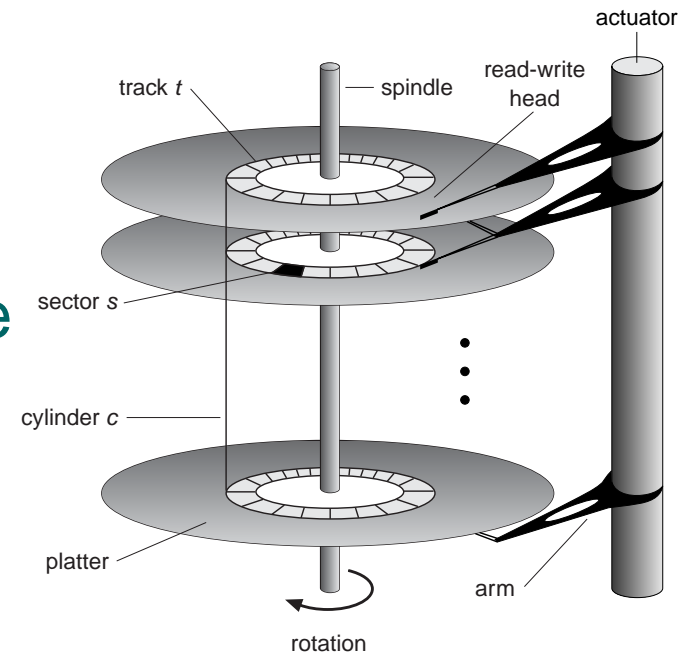
- **Aujourd'hui nous trouvons de plus en plus des types de mémoires qui sont adressées comme si elle étaient des disques, mais sont complètement électroniques**
 - ◆ P. ex. flash memory
 - ◆ Il n'y aura pas les temps de positionnement, latence, etc.

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 désiré
- **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

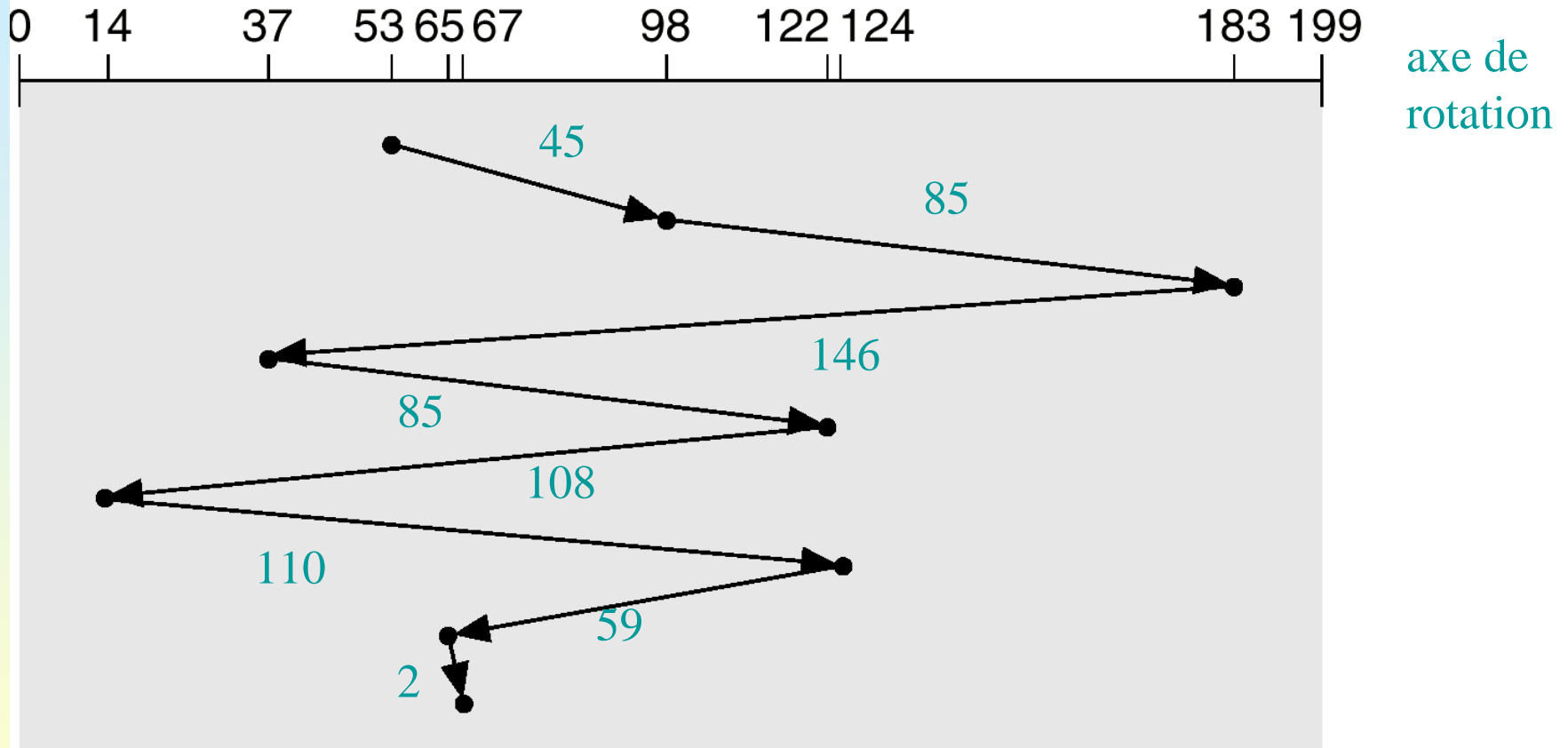
- Dans un système multiprogrammé avec mémoire virtuelle, il y aura normalement une file d'attente pour l'unité disque
- Dans quel ordre choisir les requêtes d'opérations disques de façon à minimiser les temps de recherche totaux
- Nous étudierons différentes méthodes par rapport à une file d'attente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

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Mouvement total: 640 cylindres = $(98-53) + (183-98) + \dots$
En moyenne: $640/8 = 80$

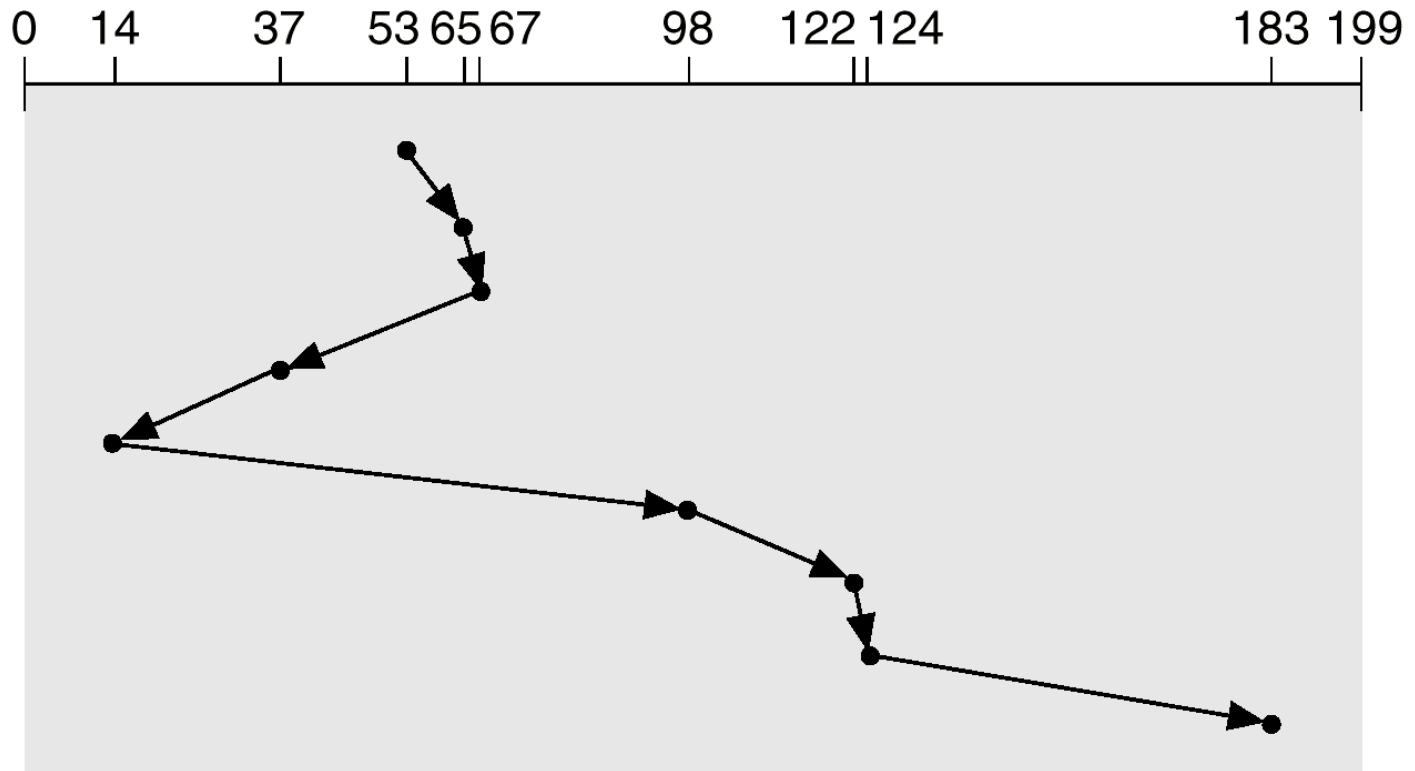
SSTF: Shortest Seek Time First

**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

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Mouvement total: 236 cylindres (680 pour le précédent)

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

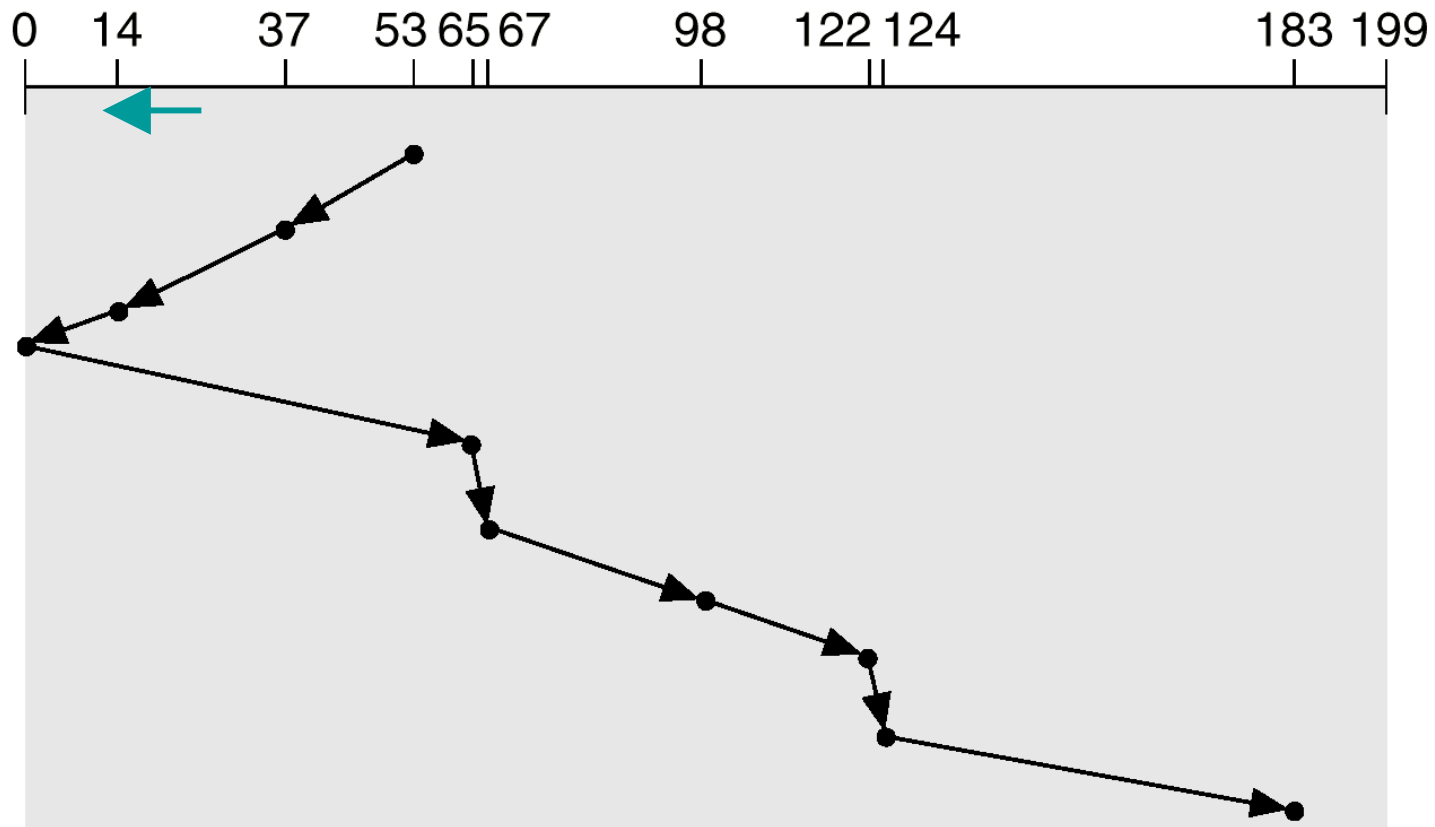
SCAN: l'algorithme de l'ascenseur

LOOK:

- **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 quand il passe sur le cylindre désiré

SCAN: l'ascenseur

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



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

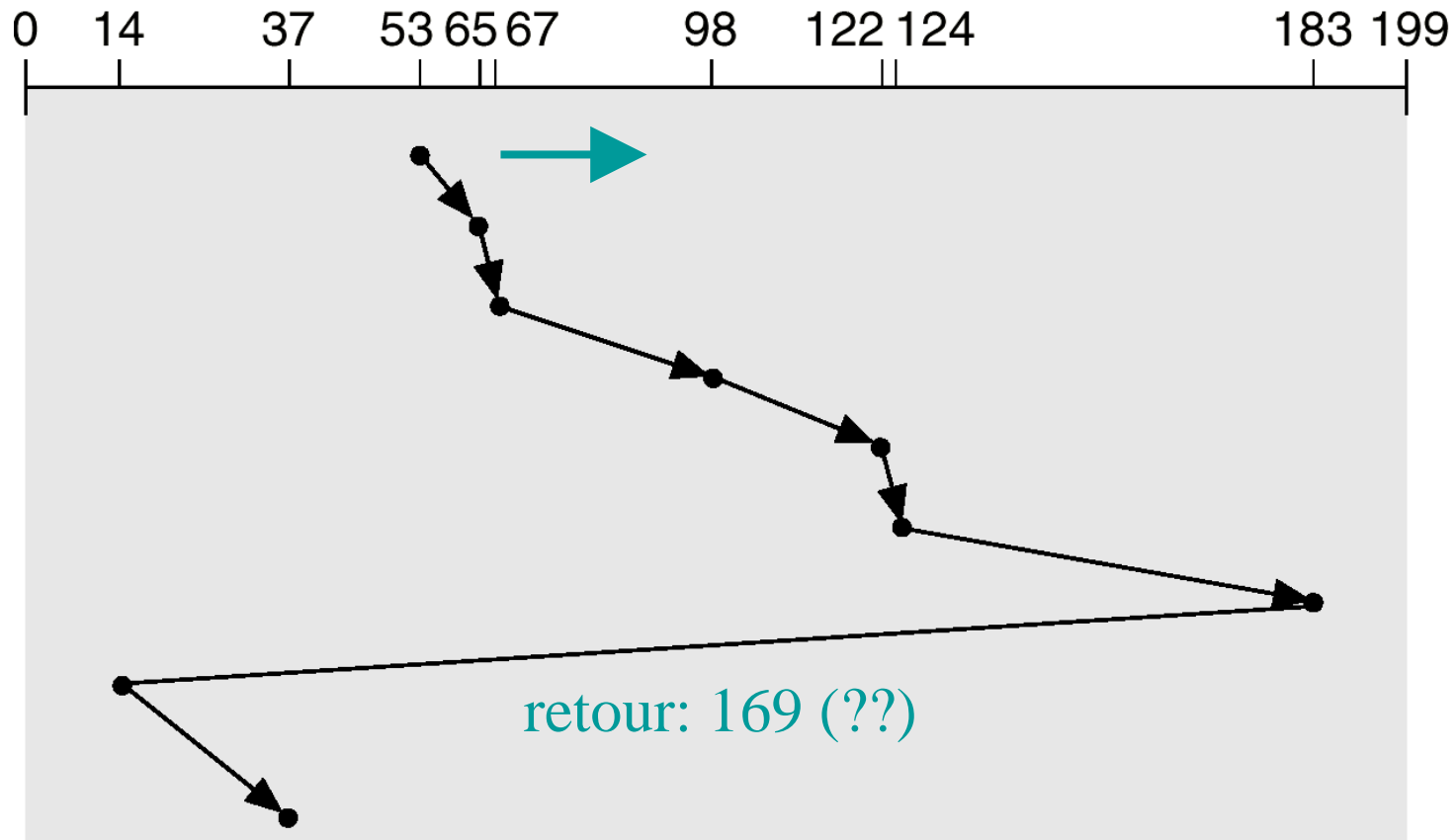
- **Retour rapide au début (cylindre 0) du disque au lieu de renverser la direction**
- **Hypothèse: le mécanisme de retour est beaucoup plus rapide que le temps de visiter les cylindres**
 - ◆ Comme si les disques étaient en forme de beignes!

C-LOOK

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

C-LOOK

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53 direction →

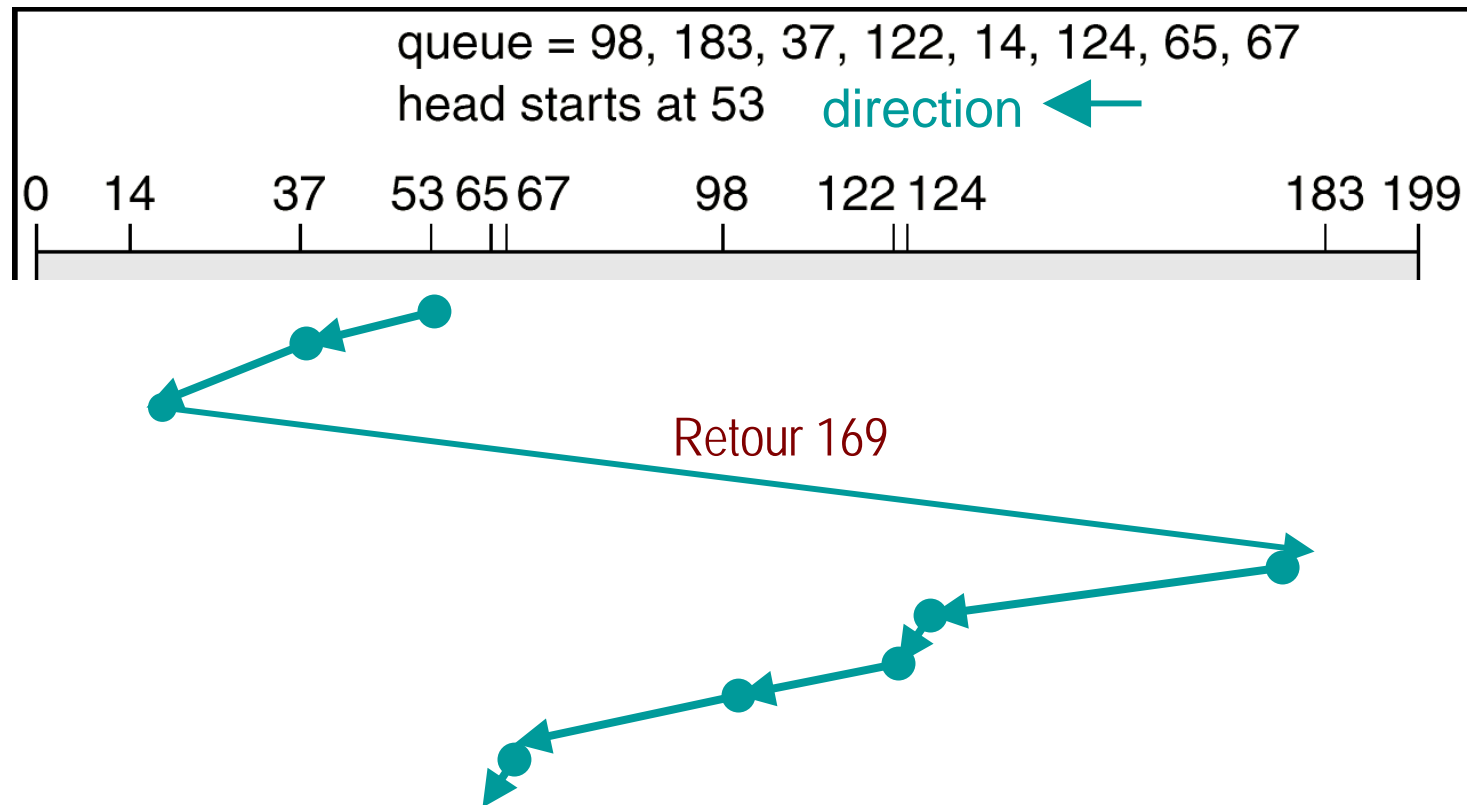


153 sans considérer le retour (19.1 en moyenne) (26 pour SCAN)

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