Chapitre IV:

Systèmes de fichiers

&

Structure de mémoire de masse (disques)

Partie I

Systèmes de fichiers

Que c’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, docs formatés
- Source: classes, méthodes, procédures…
- 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

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
- **Troncature: remise de la taille à zéro tout en conservant les attributs**

Autres opérations

- **Ajout d'infos**
- **Rénommage**
- **Copie (peut être faite par rénommage)**
- **Ouverture d'un fichier: le fichier devient associé à un processus qui en garde les attributs, position, etc.**
- **Fermeture**
- **Ouverture et fermeture peuvent être explicites (ops open, close)**
- **ou implicites**

Types de fichiers

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

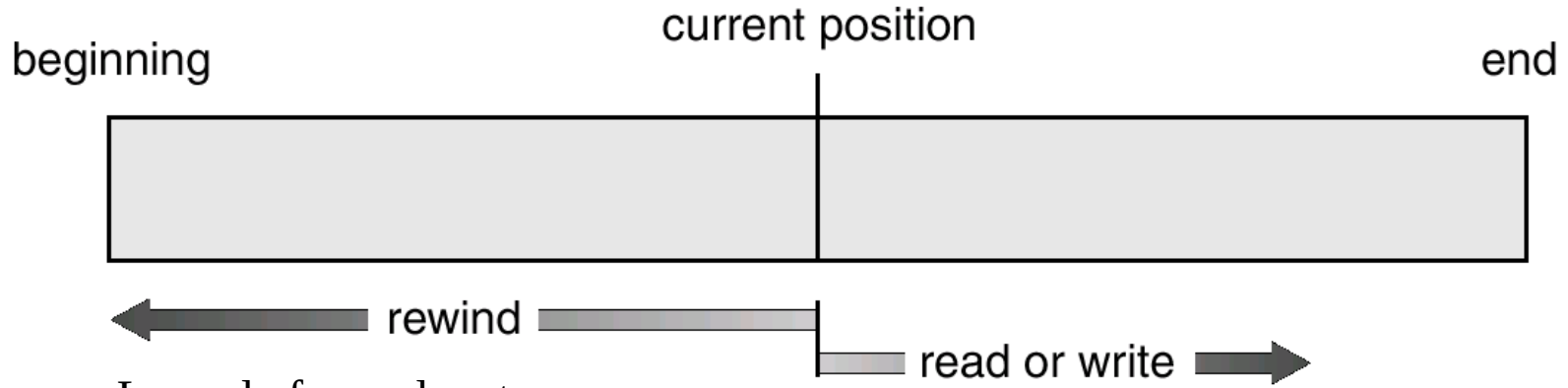
Structure logique des fichiers

- **Le type d'un fichier spécifie sa structure**
  - Le SE doit alors supporter les différentes structures correspondant aux types de fichiers
    - Cela complexifie 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 »**
  - Donc ici, 1 enregistrement = 1 octet
  - C'est l'application qui interprète le contenue et spécifie une structure
  - Plus versatile mais plus de travail pour le programmeur

Méthodes d’accès: 4 de base

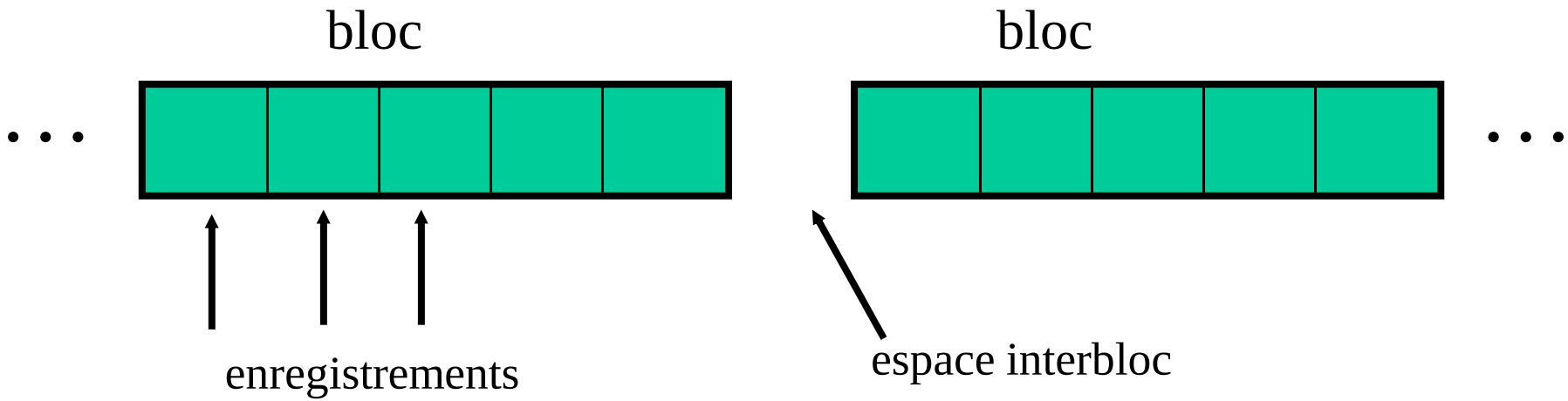
- **Séquentiel (rubans ou disques): lecture ou écriture des enregistrements dans un ordre fixe**
- **Indexé séquentiel (disques): accès séquentiel ou accès direct (aléatoire) par l’utilisation d’index**
- **Indexée: multiplicité d’index selon les besoins, accès direct par l’index**
- **Direct ou hachée: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

Fichiers à accès séquentiel (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 (no 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 proch. 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'adjonction 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'**

Accès direct ou haché ou aléatoire:

accès direct à travers tableau d'hachage

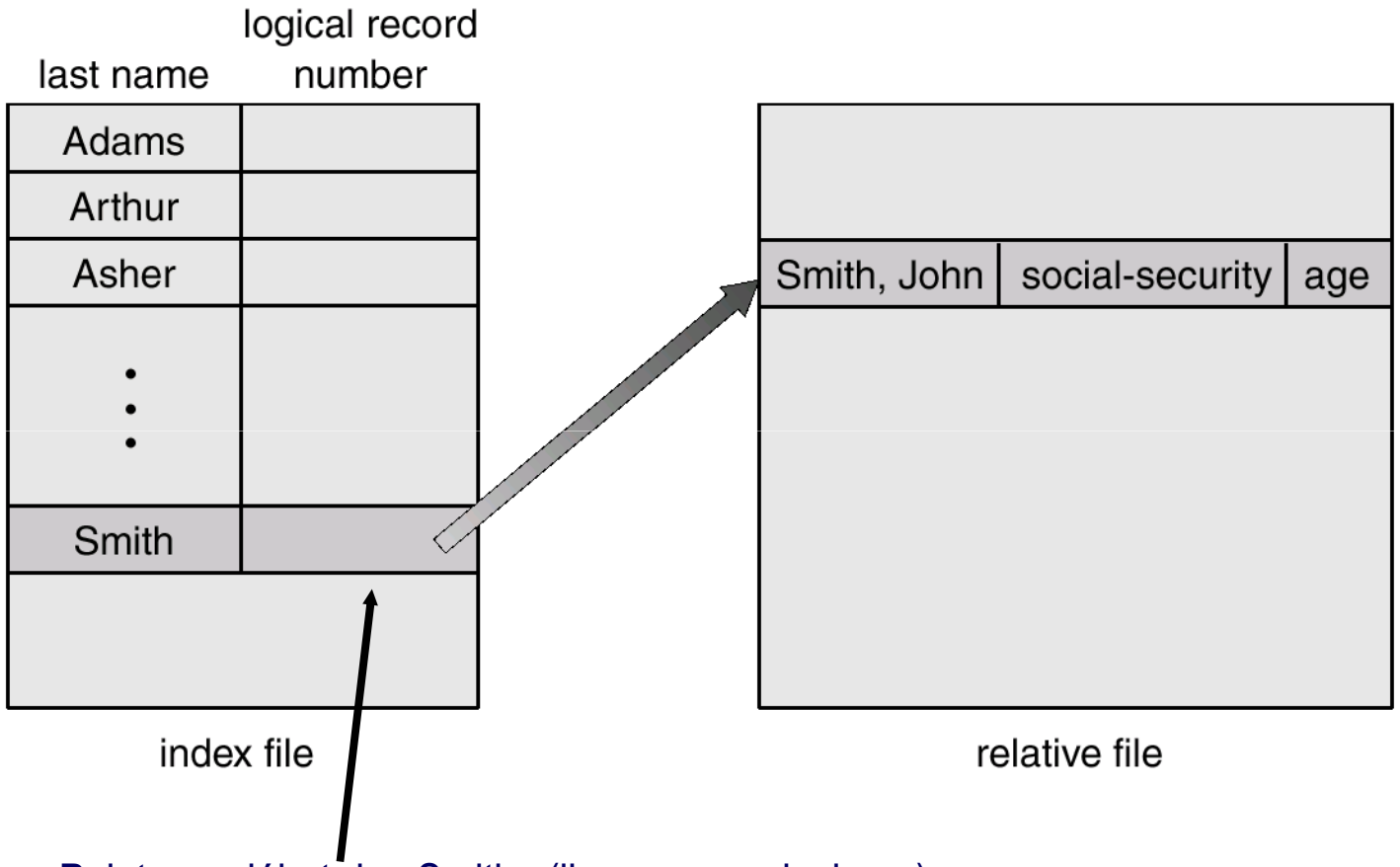
- **Une fonction d'hachage est une fonction qui traduit une clé dans adresse,**
  - P.ex. No étudiant → adresse disque

- **Rapide mais:**
  - Si elles sont trop éparpillés, gaspillage d'espace
  - Si les adresses disques 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

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)

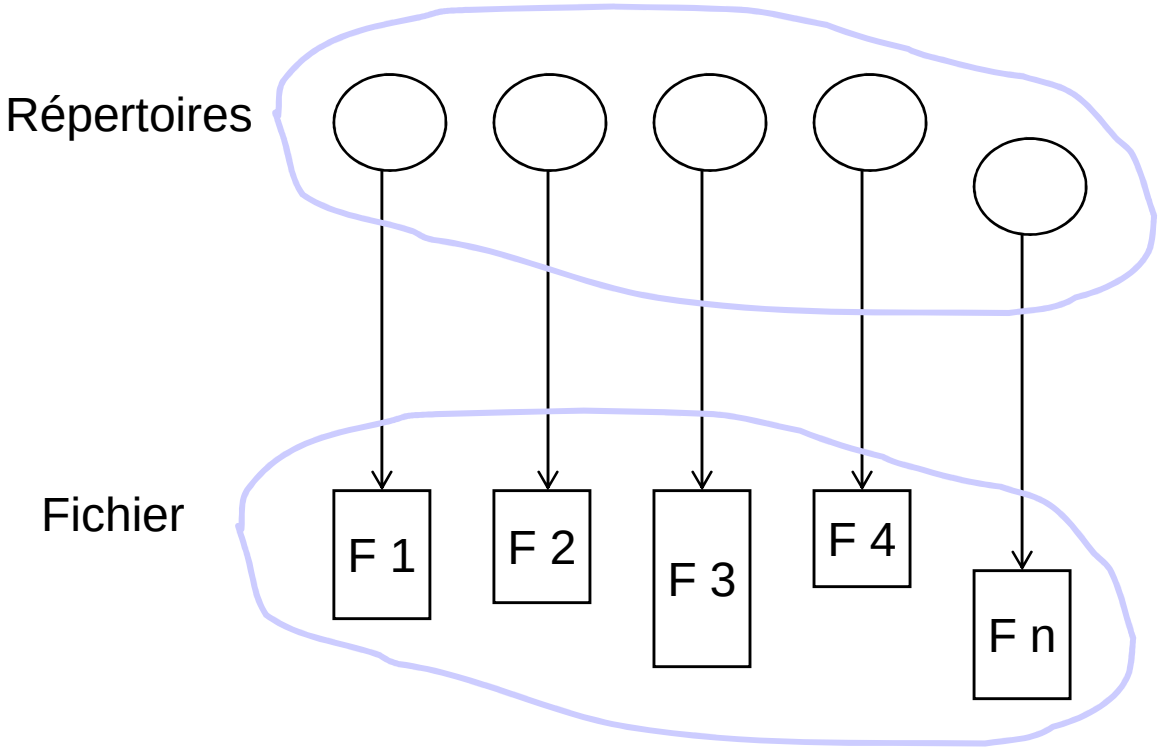
Le fichier index est à accès direct, le fichier relatif est à accès séquentiel

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 information des étudiants par No. de matricule)**

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 rép. racine en mém. centrale

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**
- **Renommer un fichier**

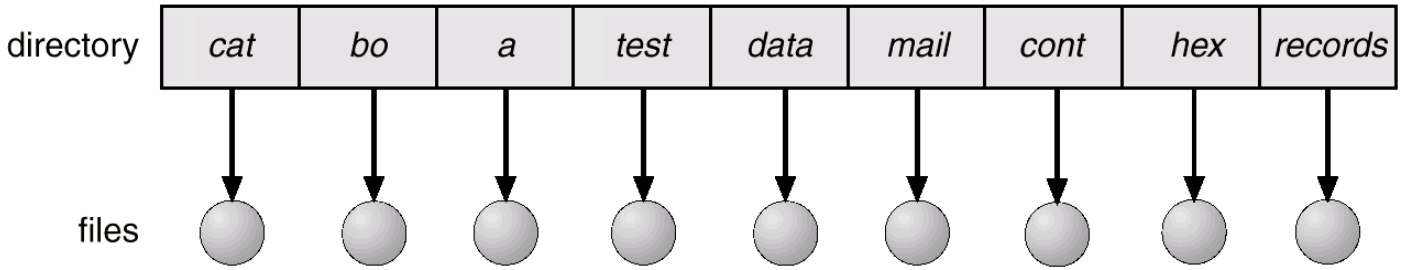
…

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

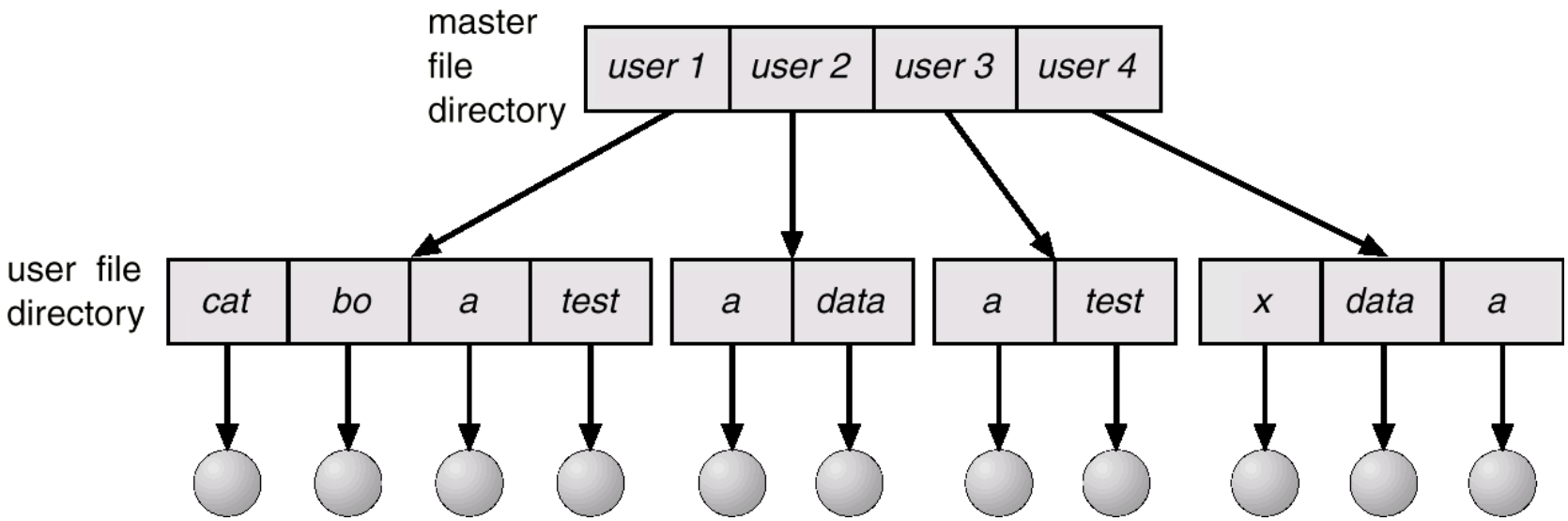
Structure à un niveau

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

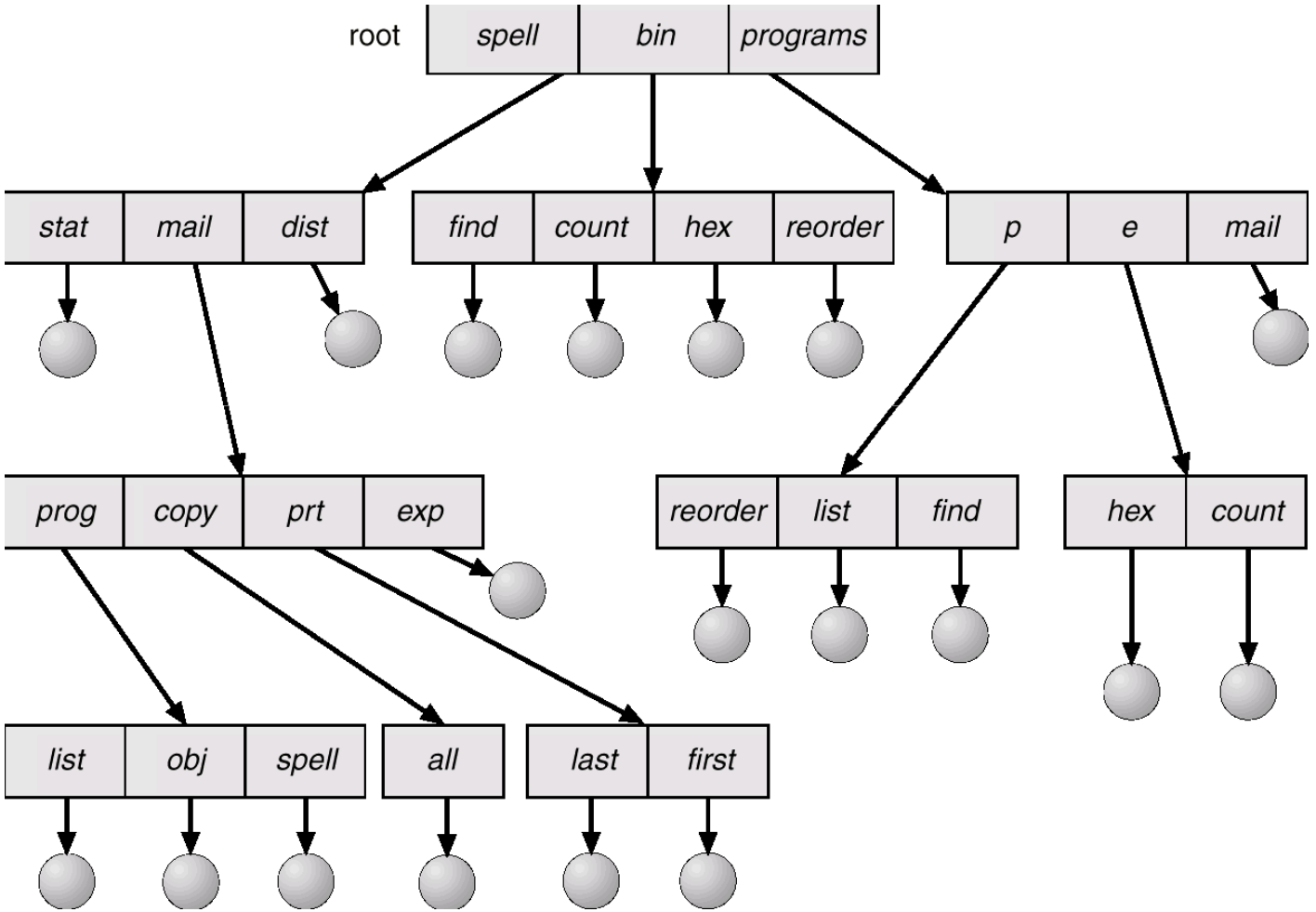


Répertoires à deux niveaux

- **Rép. séparé pour chaque usager**
- **`path name`, nom de chemin**
- **même nom de fichier pour usagers différents est permis**
- **recherche efficace**



Répertoires à arbres (normal aujourd'hui)



Partage de fichiers

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

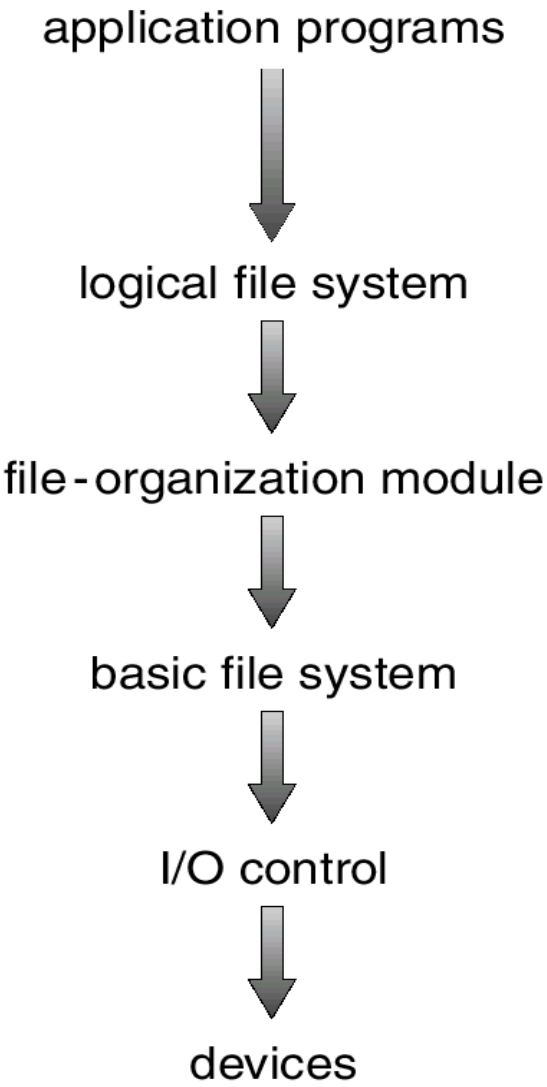
Protection

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

Structures de systèmes de fichiers

- **Structure de fichiers: deux façons de voir un fichier:**
  - unité d’allocation espace
  - collection d ’informations reliées
- **Le système de fichiers réside dans la mémoire secondaire: disques, rubans...**
- **File control block: structure de données contenant de l ’info sur un fichier**

Systèmes de fichiers à couches



Structure physique des fichiers

- **La mémoire secondaire est subdivisée en blocs de taille fixe et chaque opération d'E /S s'effectue en unités de blocs**
  - Les blocs sont numérotés: le 1er bloc du médium se trouve à l'adresse 0, et le nième bloc est à l'adresse n-1
  - 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 (ex: « main-frame IBM »)
- **Les fichiers sont alloués en unité de blocs. Le dernier bloc est donc rarement rempli de données**
  - Fragmentation interne

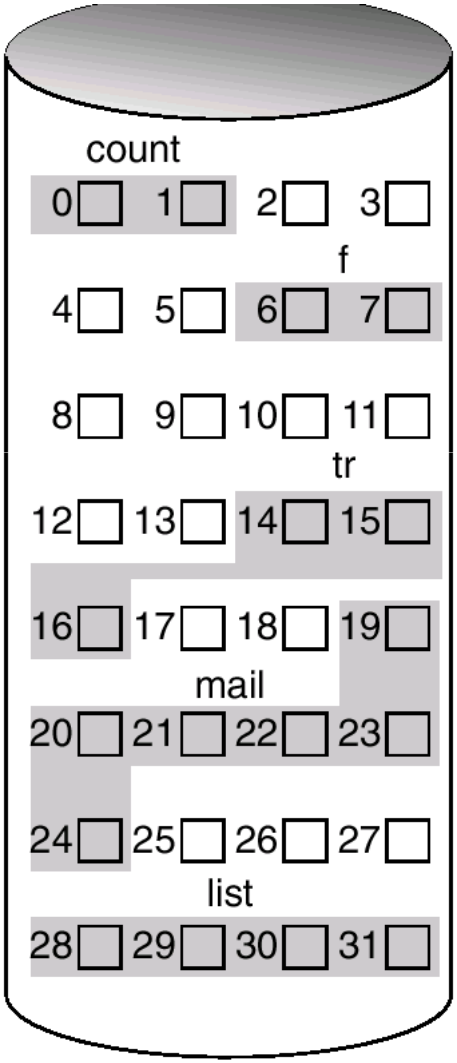
Trois méthodes d’allocation de fichiers

- Allocation contiguë
- Allocation enchaînée
- Allocation indexée

Allocation contiguë

- **Chaque fichier occupe un ensemble de blocs contigu sur disque**
- **Simple: nous n'avons besoin que d'adresses de début et longueur**
- **Supporte tant l 'accès séquentiel, que l 'accès direct**
- **Gaspilleur: pbs d 'allocation dynamique**
  - v. discussion sur allocation de mémoire contiguë: problèmes de fragmentation
  - les fichiers ne peuvent pas grandir
  - difficile d 'ajouter des nouveaux enregistrements ' au milieu '
- **Application des méthodes de compression, first fit, best fit...**
- **Compliquée à gérer si on désire gérer efficacement l'espace en même temps donnant la flexibilité à l 'usager**

Allocation contiguë sur disque



directory répertoire

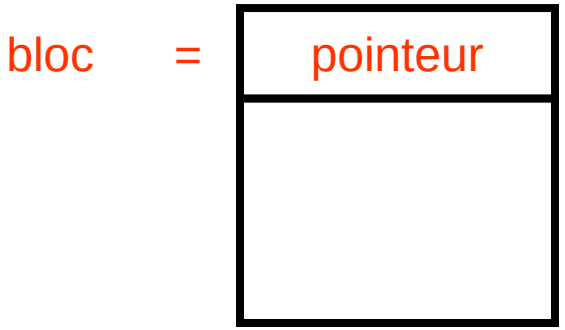
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation contiguë

- **Bon support pour fichiers séquentiels et fichiers à accès direct**
- **Moins pratique pour les autres méthodes**
- **Fragmentation externe: il peut être difficile de trouver k blocs contigus pour contenir un fichier**
- **Exécution périodique d'une compression (compaction) pour récupérer l'espace libre**

Allocation enchaînée

- **Le répertoire contient l ’adresse du premier et dernier bloc, possibl. le nombre de blocs**
- **Chaque bloc contient un pointeur à l’adresse du prochain bloc:**



Allocation enchaînée

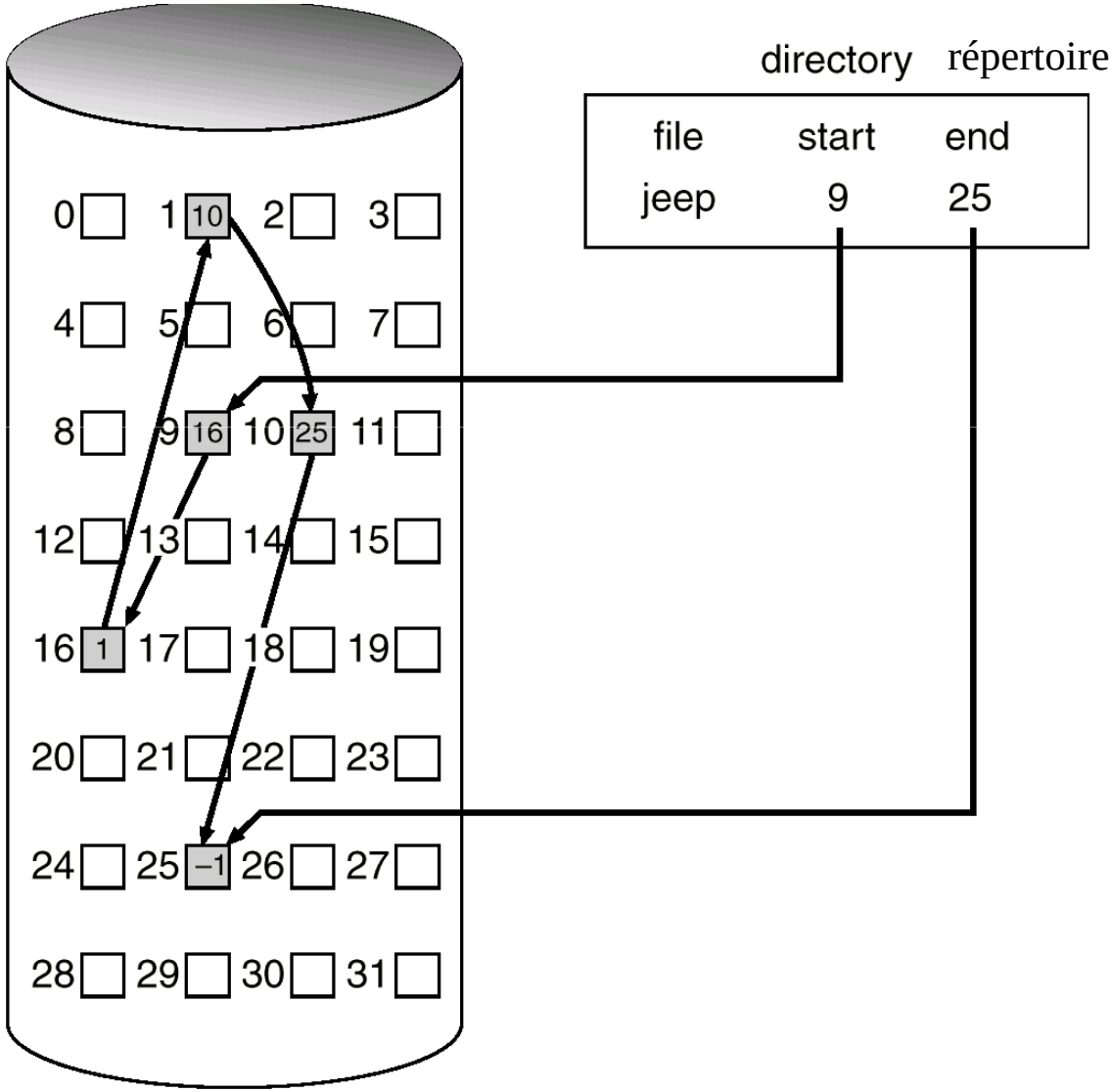
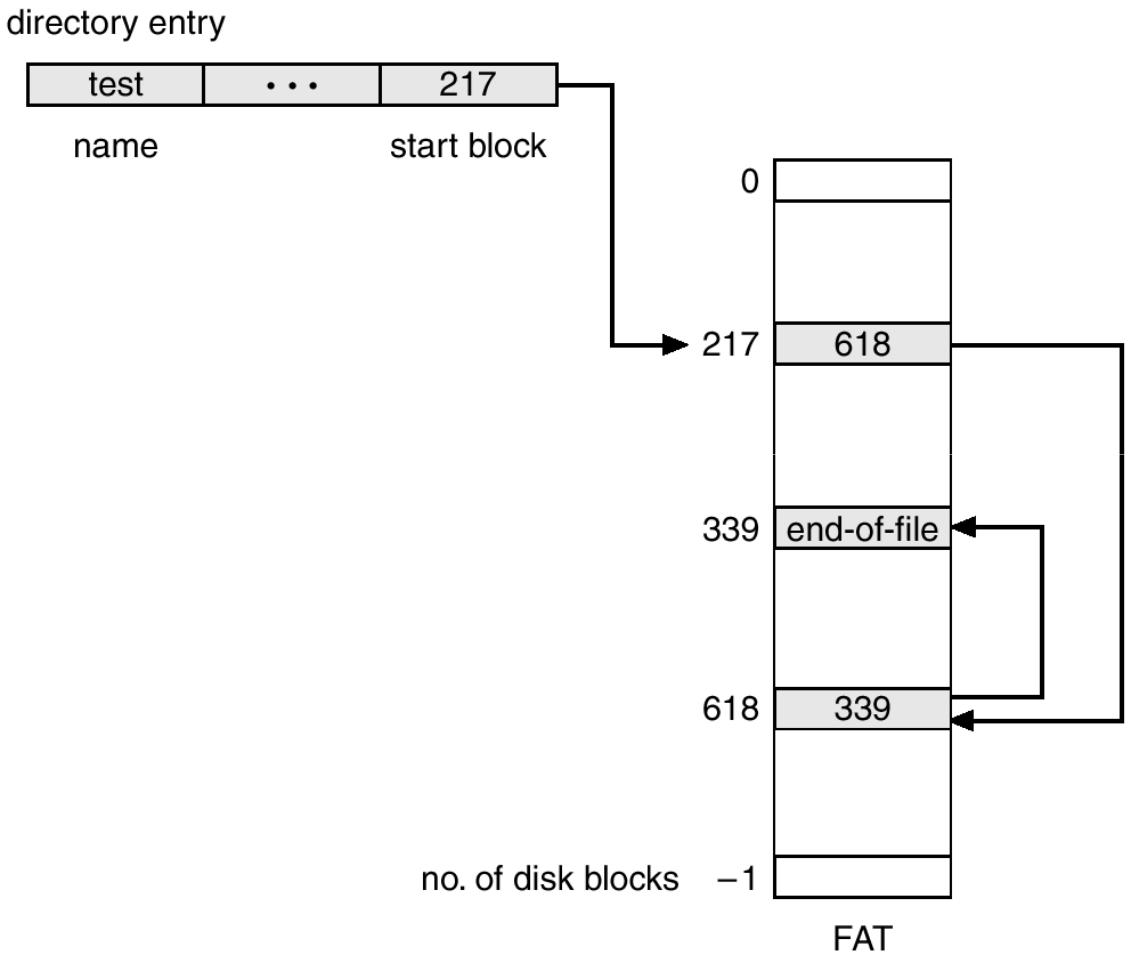


Tableau d’allocation de fichiers: **FAT**

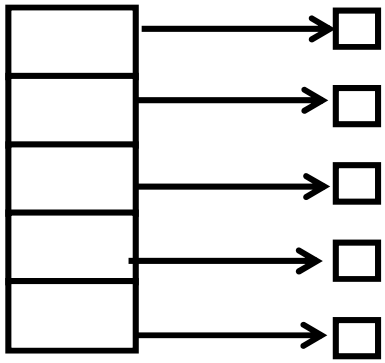


Avantages - désavantages

- **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 façon de trouver directement le 4ème enregistrement...
  - N’utilise pas la localité car les enregistrements seront éparpillés
- **L ’intégrité des pointeurs est essentielle**
- **Les pointeurs gaspillent un peu d ’espace**

Allocation indexée: semblable à la pagination

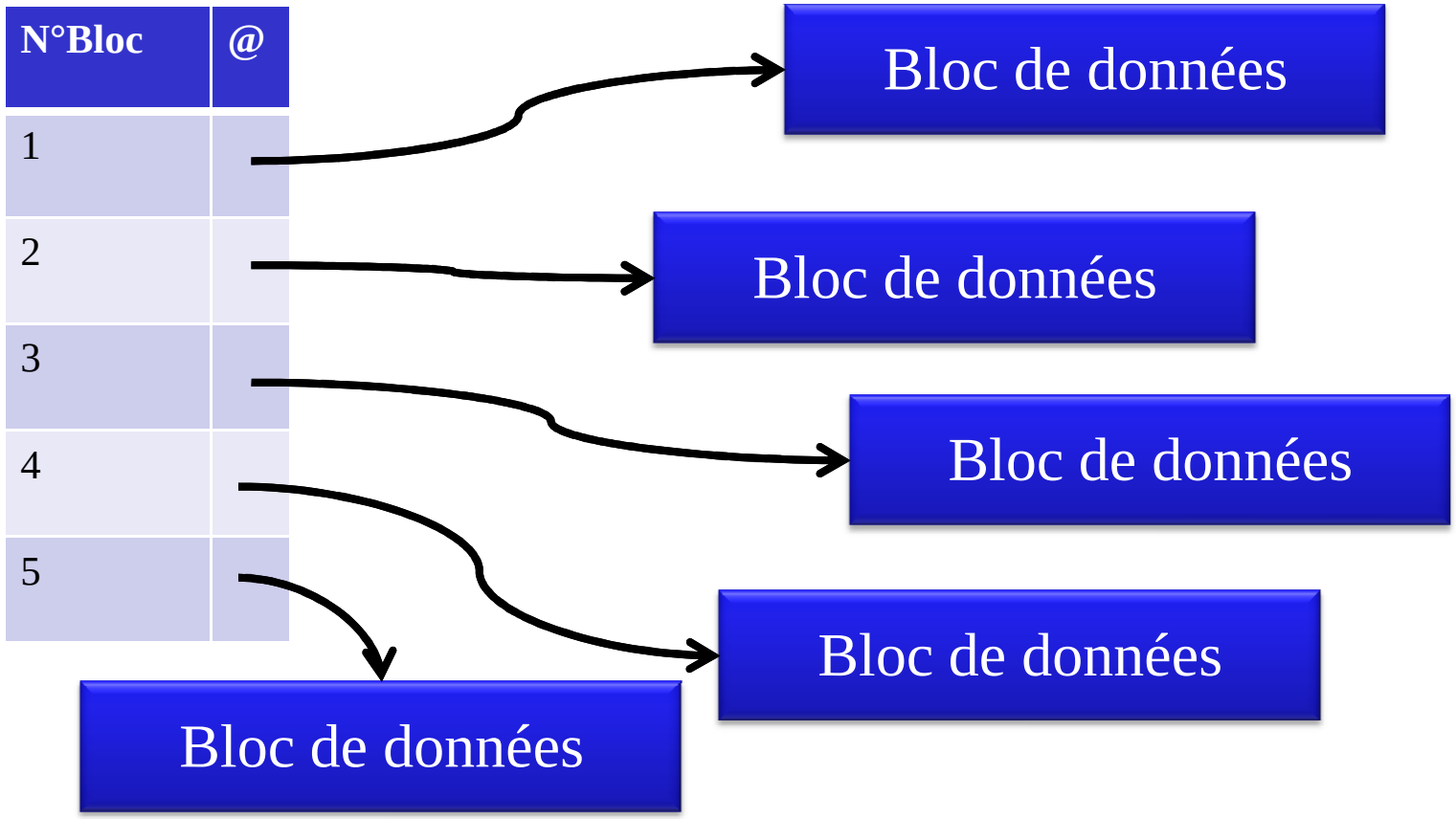
- **Tous les pointeurs sont regroupés dans un tableau (index block)**



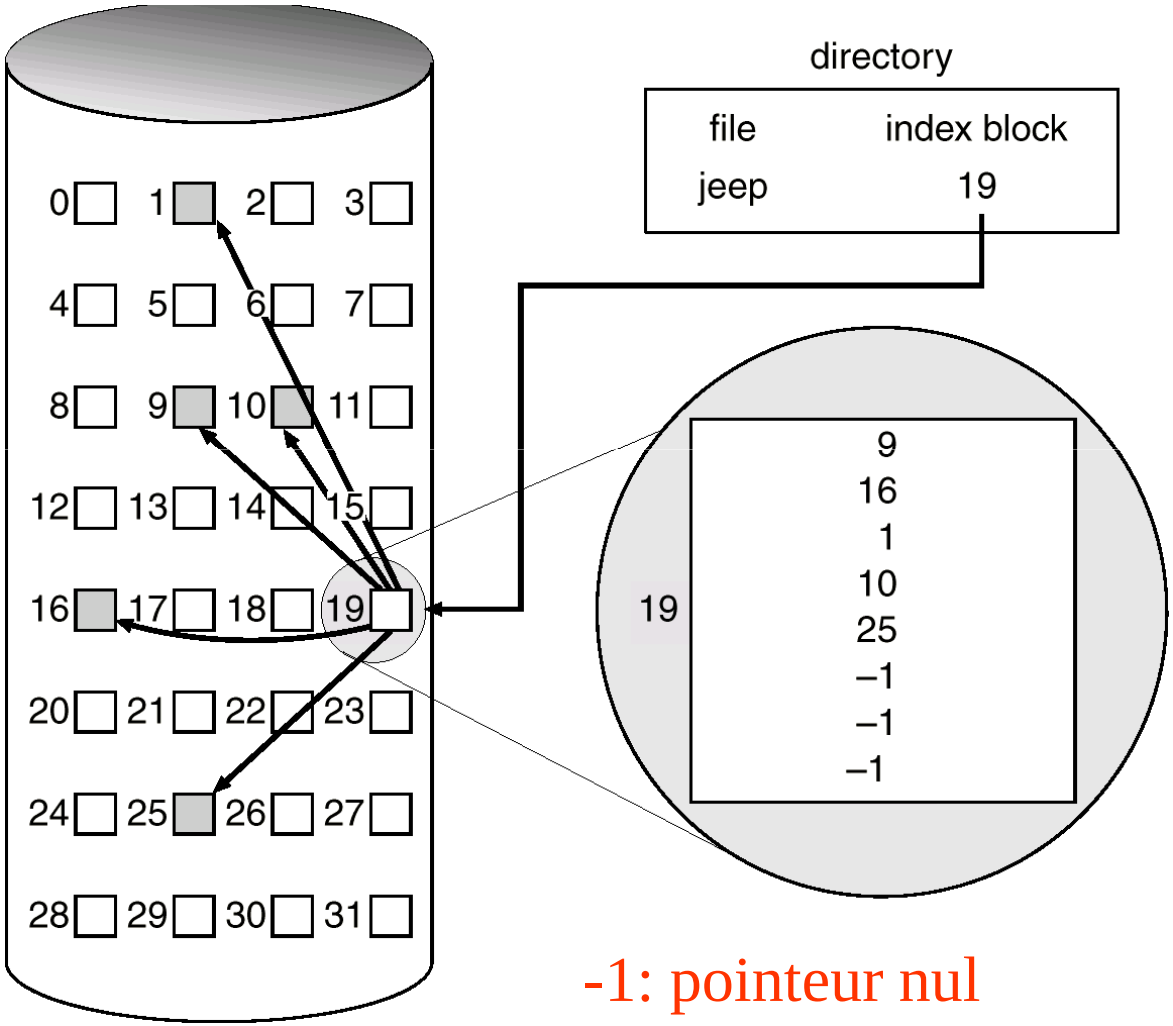
index table

Fichier Ordinaire : La structure physique

allocation non contiguë indexée :



Allocation indexée



-1: pointeur nul

Allocation indexée

- **À la création d ’un fichier, tous les pointeurs dans le tableau sont nul (-1)**
- **Chaque fois qu’un nouveau bloc doit être alloué, on trouve de l ’espace disponible et on ajoute un pointeur avec son adresse**

Allocation indexée

- **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 mém.**

Fichier Ordinaire : La structure physique

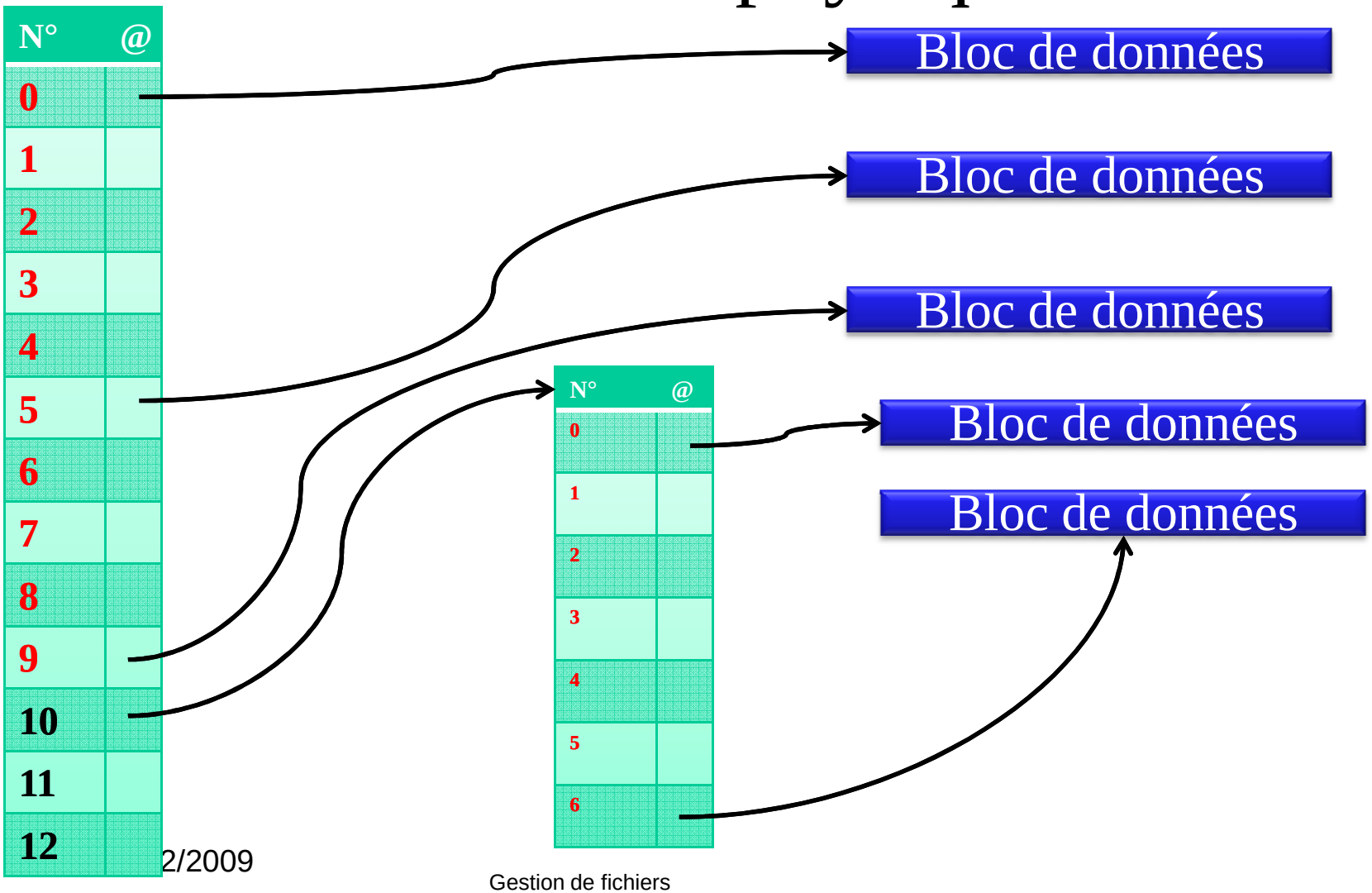
allocation non contiguë mixte : (EXT)

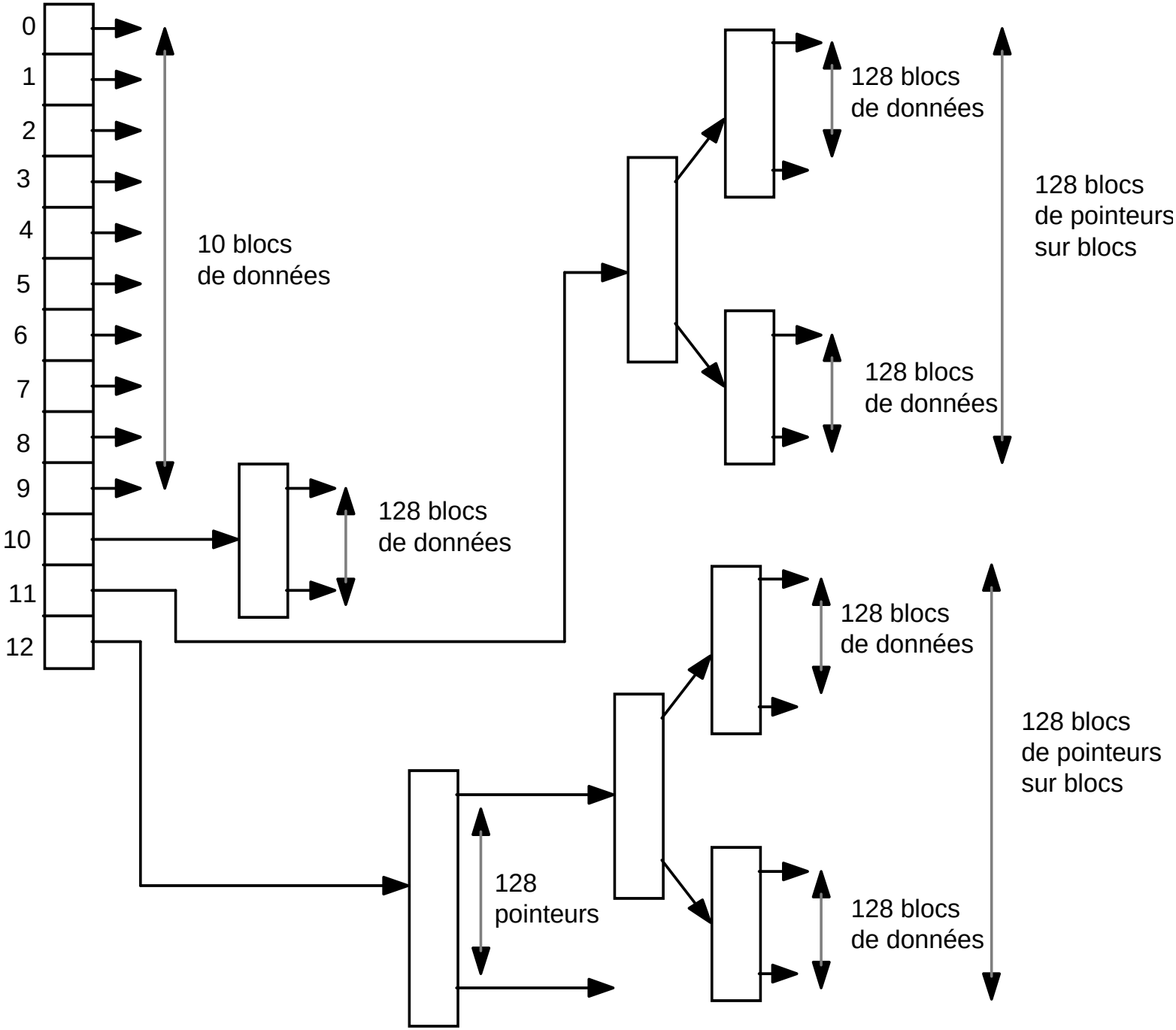
on peut améliorer la solution précédente en utilisant plusieurs niveaux de tables d'index. Citons par

exemple la méthode utilisée par UNIX :

- chaque fichier se voit affecter une table **à 13 entrées**
- les entrées 0 à 9 pointent sur un **bloc de données**
- l'entrée 10 pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc de données (simple indirection)
- l'entrée 11 pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (double indirection)
- l'entrée 12 pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (triple indirection)

Fichier Ordinaire : La structure physique

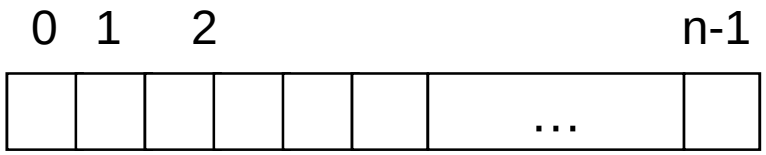



0
1
2
3
4
5
6
7
8
9
10
11
12
10 blocs
de données
128 blocs
de données
128 blocs
de données
128 blocs
de pointeurs
sur blocs
128 blocs
de données
128 blocs
de données
128 blocs
de pointeurs
sur blocs
128
pointeurs

Gestion d'espace libre

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

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



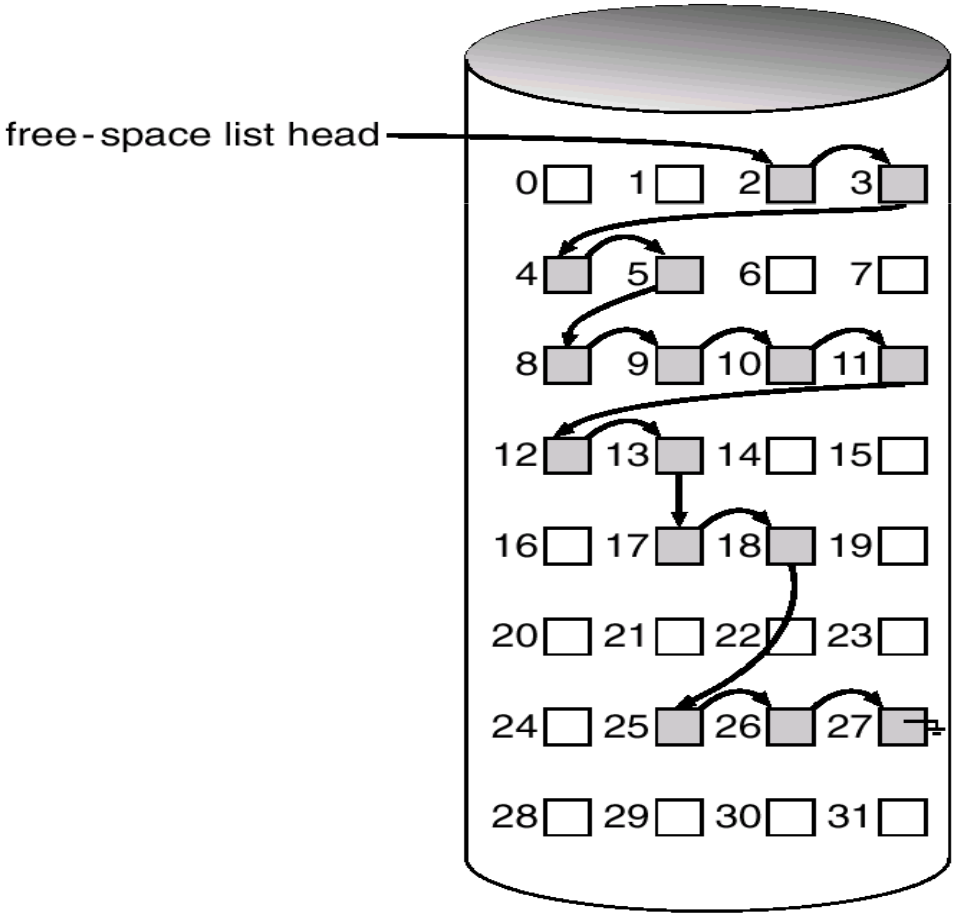
bit[*i*] = { 0 ⇒ block[*i*] libre ; 1 ⇒ block[*i*] occupé

- **Exemple d'un vecteur de bits où les blocs 3, 4, 5, 9, 10, 15, 16 sont occupés:**
  - **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 arriver au premier bloc de mémoire libre, plus. accès de disque pourraient être demandés
  - Pour éviter ceci, nous pouvons garder en mémoire centrale l 'adresse de ce bloc

Partie II

Structure de mémoire de masse (disques)

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

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

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

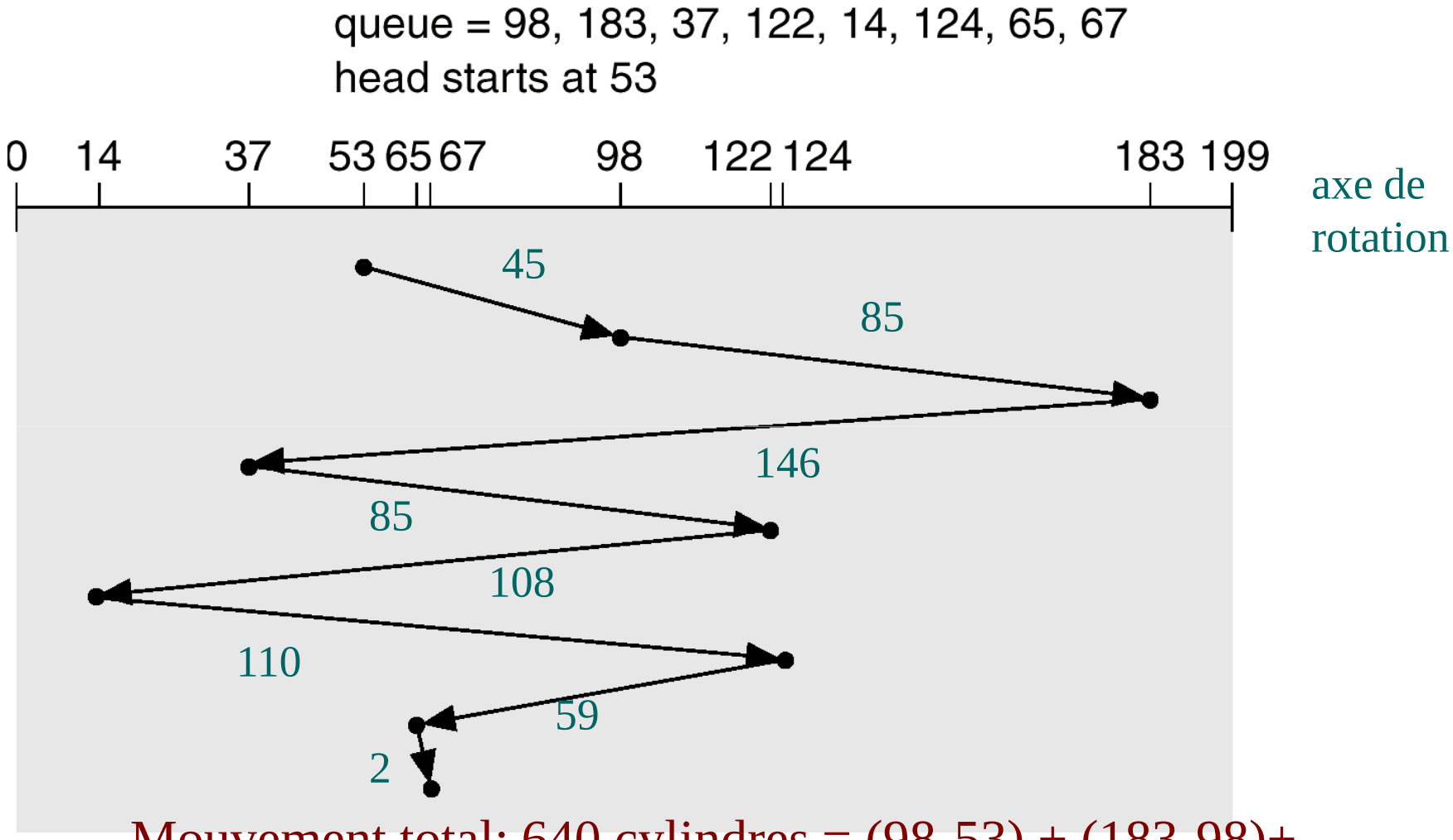
- **Temps de positionnement (seek time):**
  - le temps pris par l`unité disque pour se positionner sur le cylindre désiré
- **Temps de latence de rotation**
  - le temps pris par l ’unité de disque qui est sur le bon cylindre pour se positionner sur le secteur désirée
- **Temps de lecture**
  - temps 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)+...

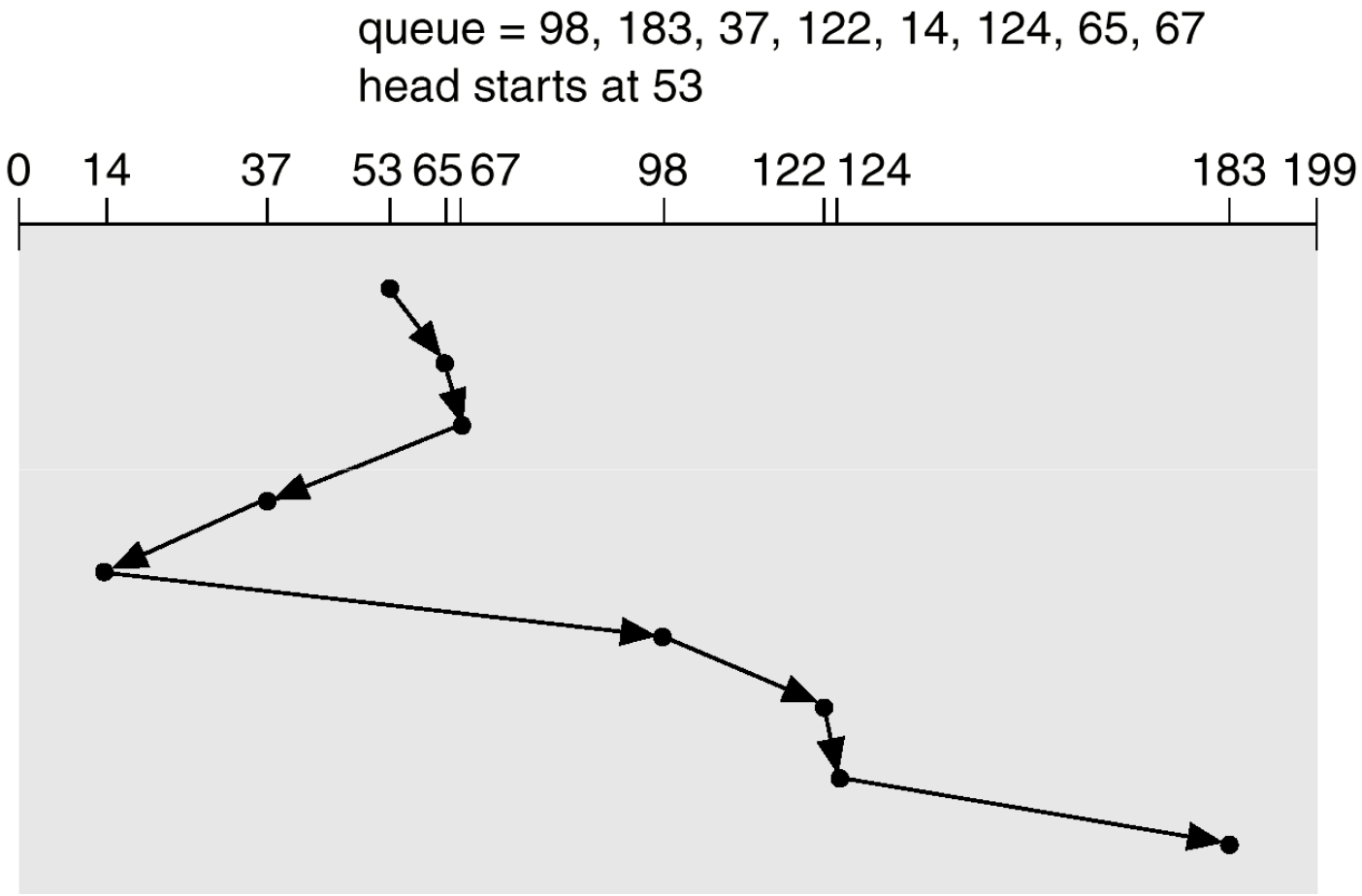
En moyenne: 640/8 = 80

SSTF: Shortest Seek Time First
Plus court d'abord

- **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



Mouvement total: 236 cylindres (680 pour le précédent)

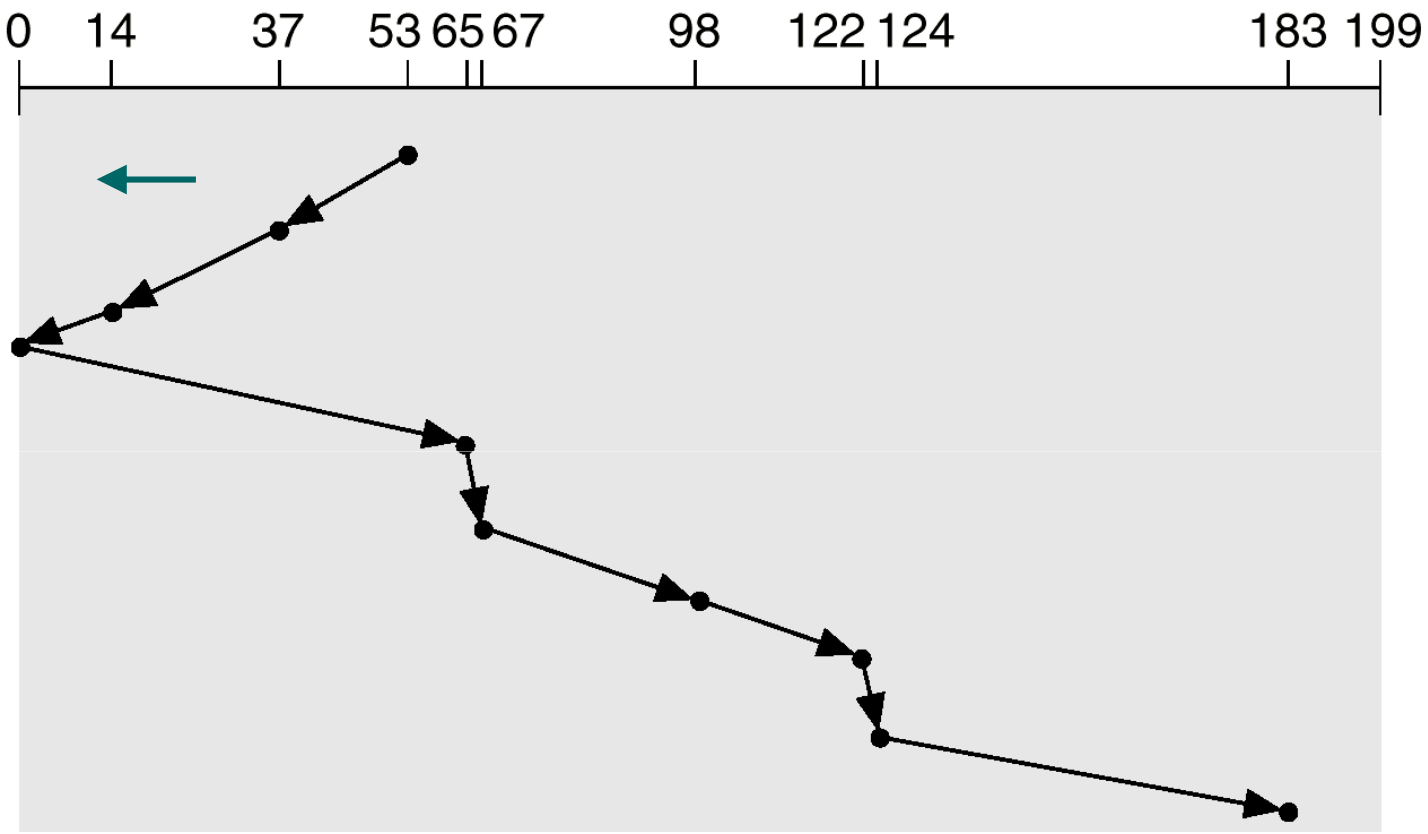
En moyenne: 236/8 = 29.5 (80 pour le précédent)

SCAN: l’algorithme de l’ascenseur

- **La tête balaye le disque dans une direction, puis dans la direction opposée, 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 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

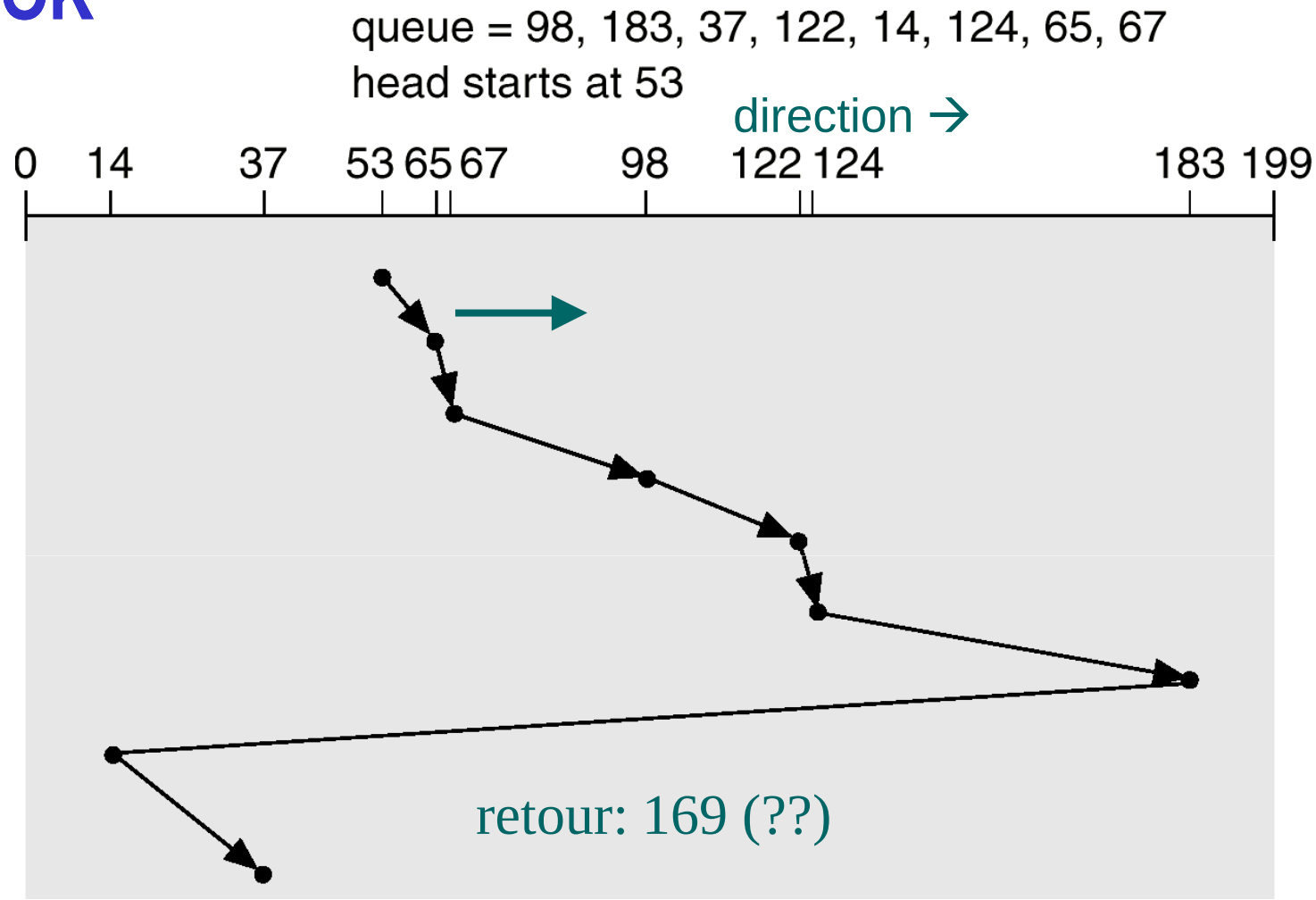
C-SCAN

- **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**
  - Disques 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

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

Conclusion

- **Fonctionnement et structure des unités disque**
- **Calcul du temps d'exécution d'une séquence d'opérations**
- **Différents algorithmes d'ordonnancement**
  - Fonctionnement, rendement