# Module 10 – Structure de mémoire de masse (disque)

## Chapitre 12

# Concepts importants du Module 10

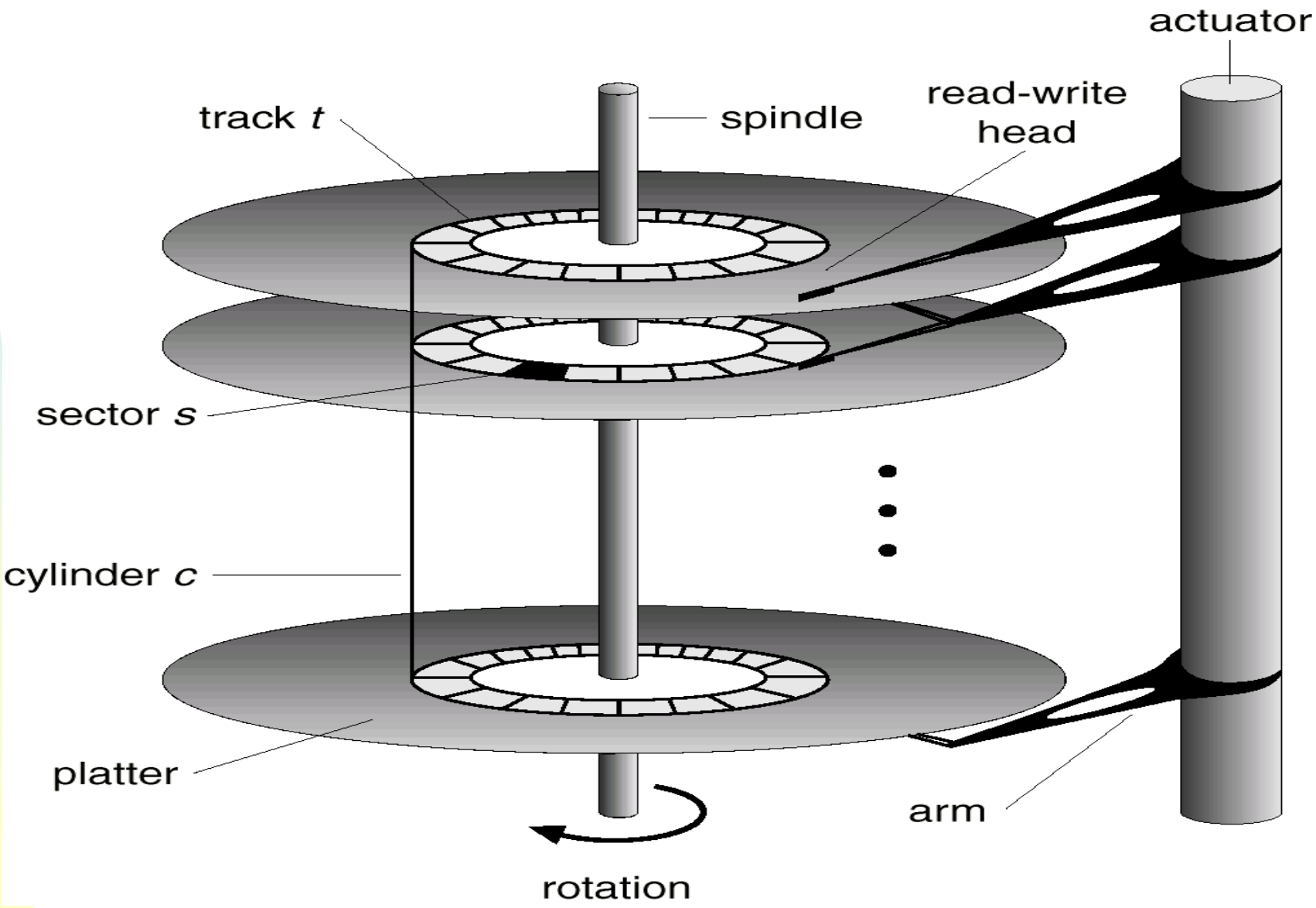
- **Fonctionnement et structure des unités disque**
- **Calcul du temps d'exécution d'une séquence d'opérations**
- **Différents algorithmes d'ordonnancement**
  - Fonctionnement, rendement
- **Gestion de l'espace de permutation**
  - Unix
- **RAID – disques résistants aux erreurs**

# Disques magnétiques

- **Plats, rigides, couverts de matériaux d'enregistrement magnétique**
  - La surface du disque est divisée en pistes/sillons (tracks) subdivisées en secteurs
  - le contrôleur du 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

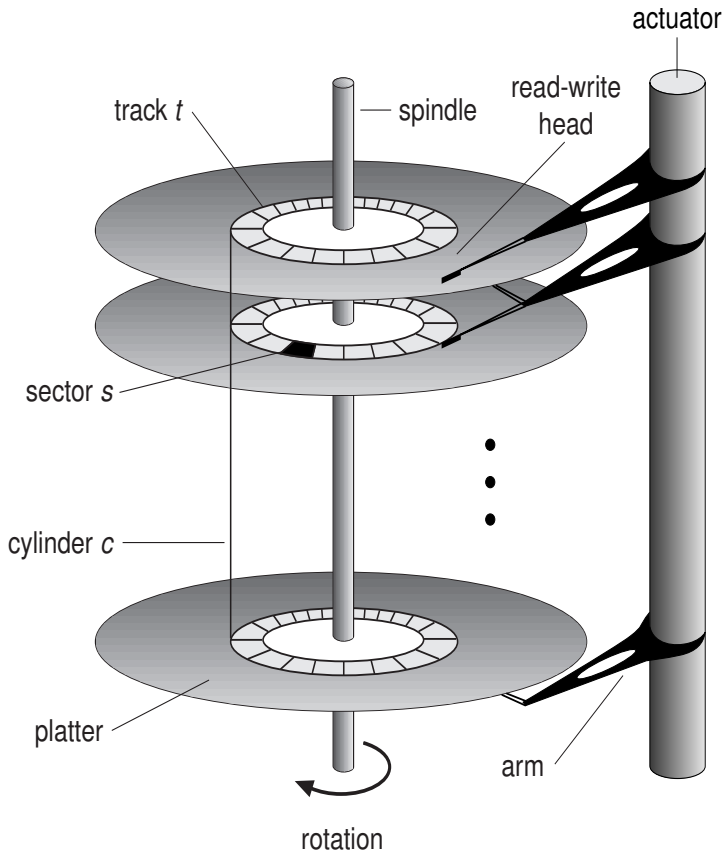
- **Aujourd'hui nous trouvons de plus en plus des types de mémoires qui sont adressées comme s'ils étaient des disques, mais sont complètement électroniques**
  - Ex. flash memory – clefs USBs
  - Il n'y a pas les temps de positionnement, latence, etc. discutés plus tard

# Ordonnancement disques

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

# Paramètres à prendre en considération

- **Temps de positionnement (seek time):**
  - le temps pris par l`unité du 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**

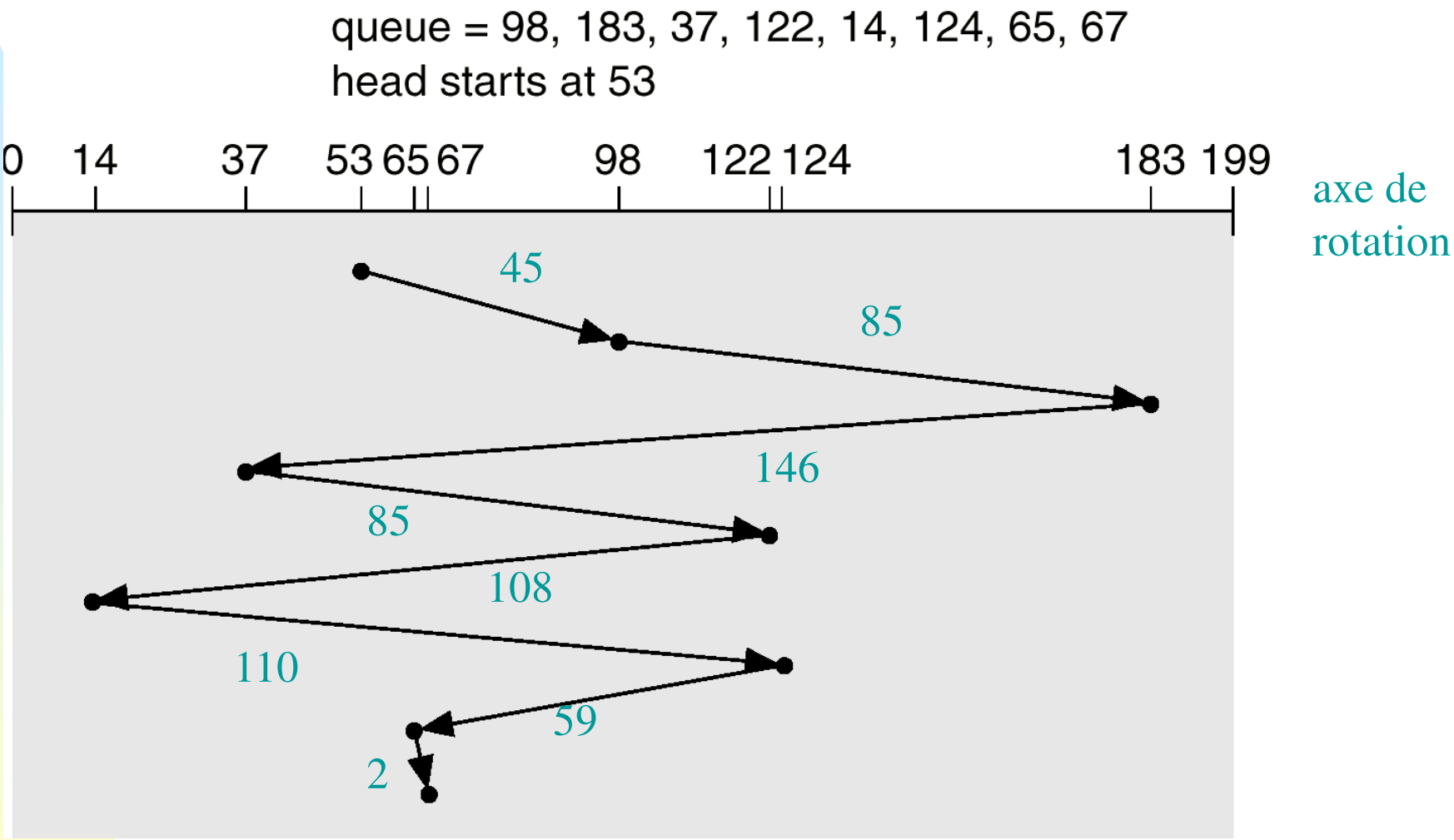


# File d'attente disque

- Dans un système multiprogrammé avec une mémoire virtuelle, il y a normalement une file d'attente pour l'unité disque
- Dans quel ordre choisir les requêtes d'opérations de disques de façon à minimiser les temps totaux de recherche
- Nous étudierons différentes méthodes en se réfèrent 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 du cylindre
- Hypothèse simpliste: un déplacement d'1 cylindre coûte 1 unité de temps

# Premier entré, premier sorti: FIFO or FCFS



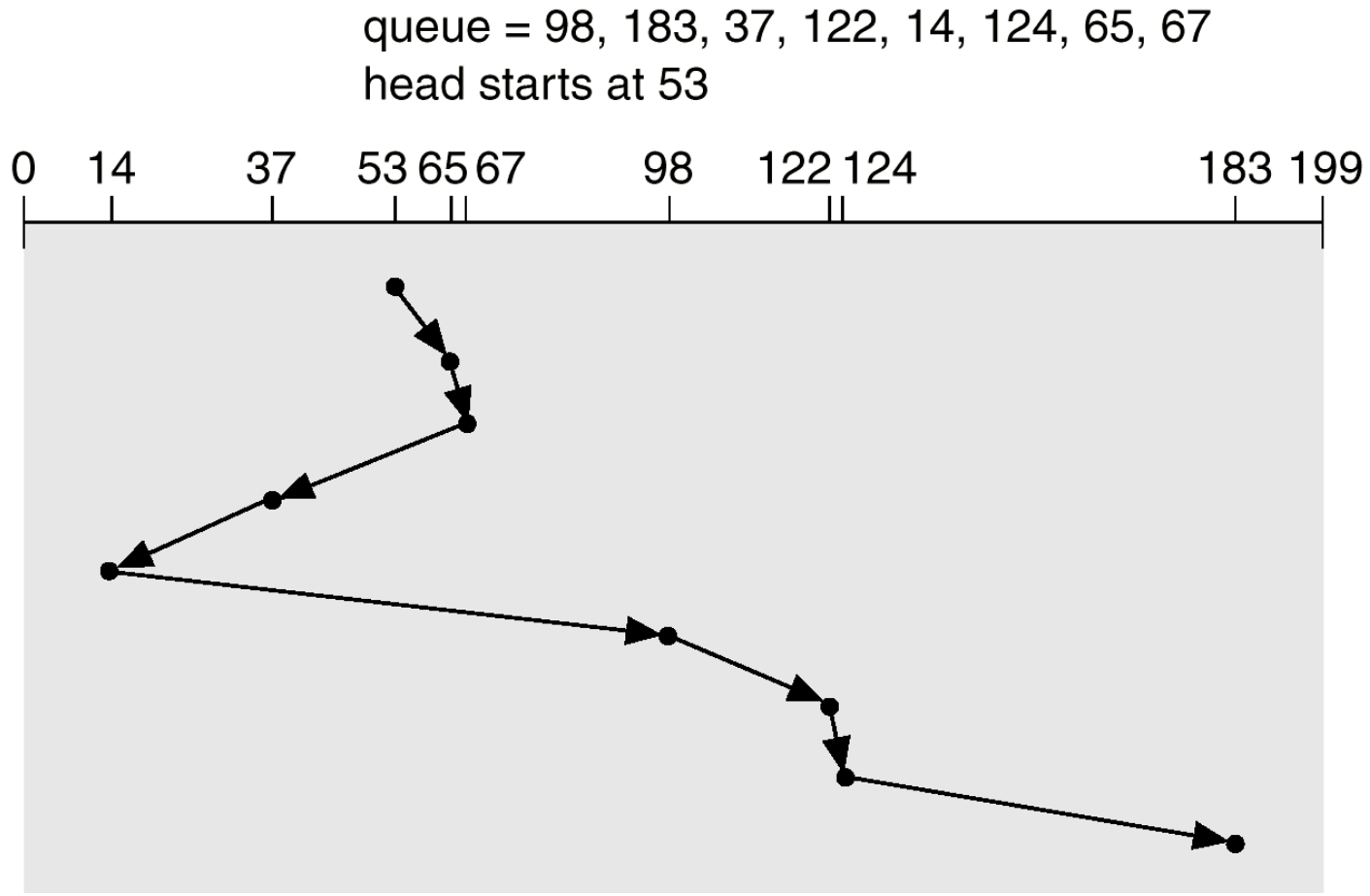

Mouvement total: 640 cylindres = (98-53) + (183-98)+...

En moyenne: 640/8 = 80

# SSTF: Shortest Seek Time First
## Plus court recherche 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! (v. manuel)
- Peut causer famine

# SSTF: Plus court servi



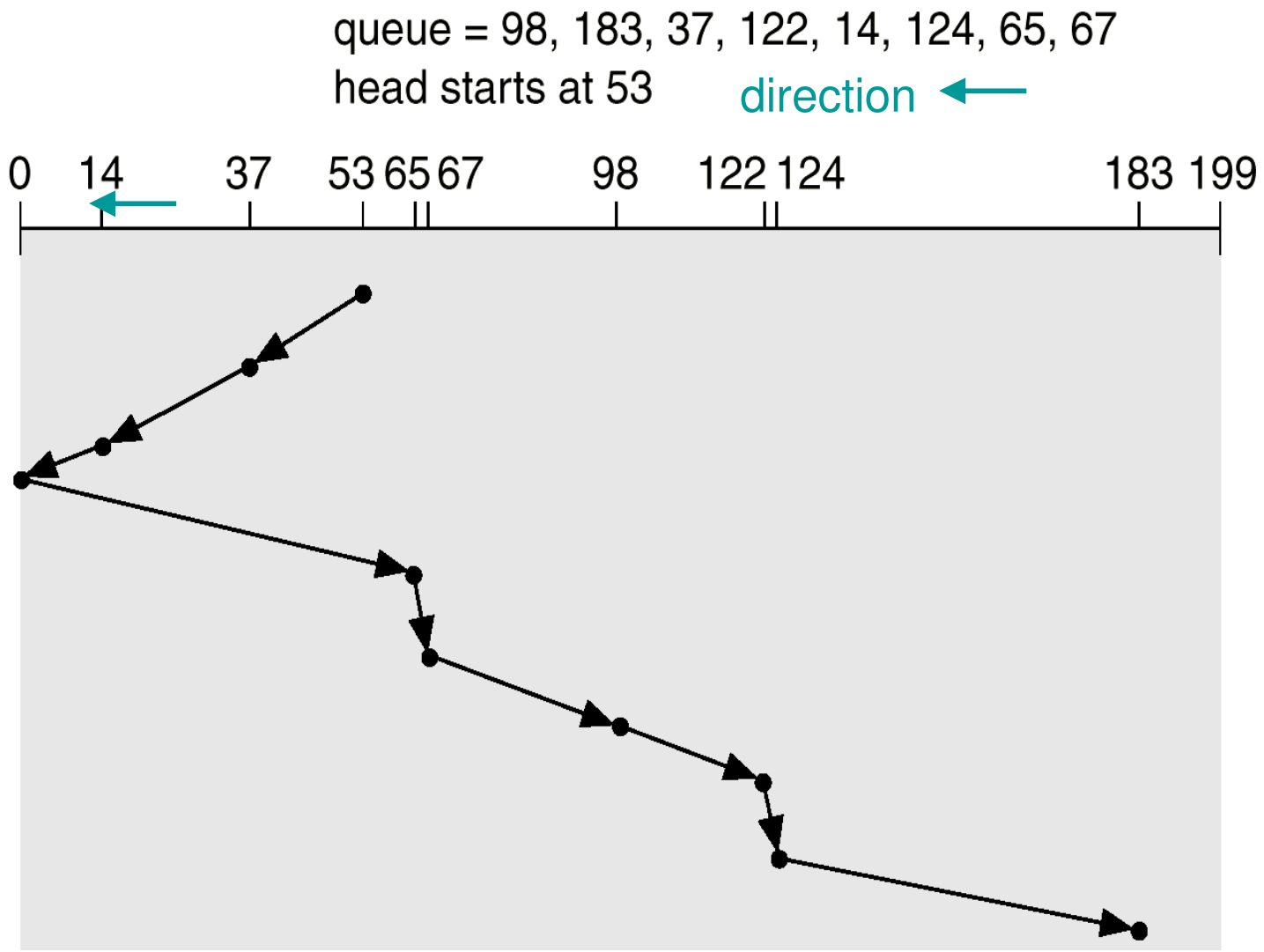

Mouvement total: 236 cylindres (680 pour le précédent)

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**
  - Pas de famine

# SCAN: l 'ascenseur



# 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 SCAN Circulaire

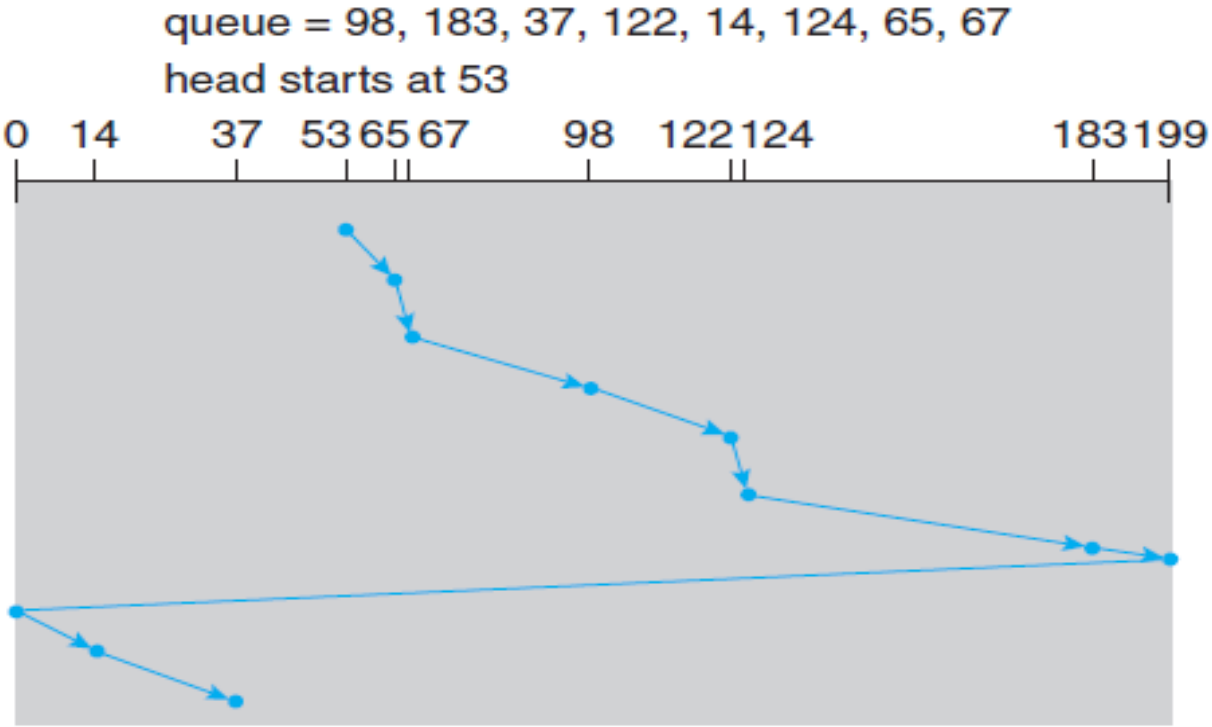
- Va d'une extrémité à une autre. Il ne fournira de service que dans une direction, lorsqu'il atteint l'autre extrémité il retourne à celle où il a débuté sans fournir de service puis refait le même chemin.
- Hypothèse: le mécanisme de retour est beaucoup plus rapide que le temps de visiter les cylindres

## C-LOOK

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

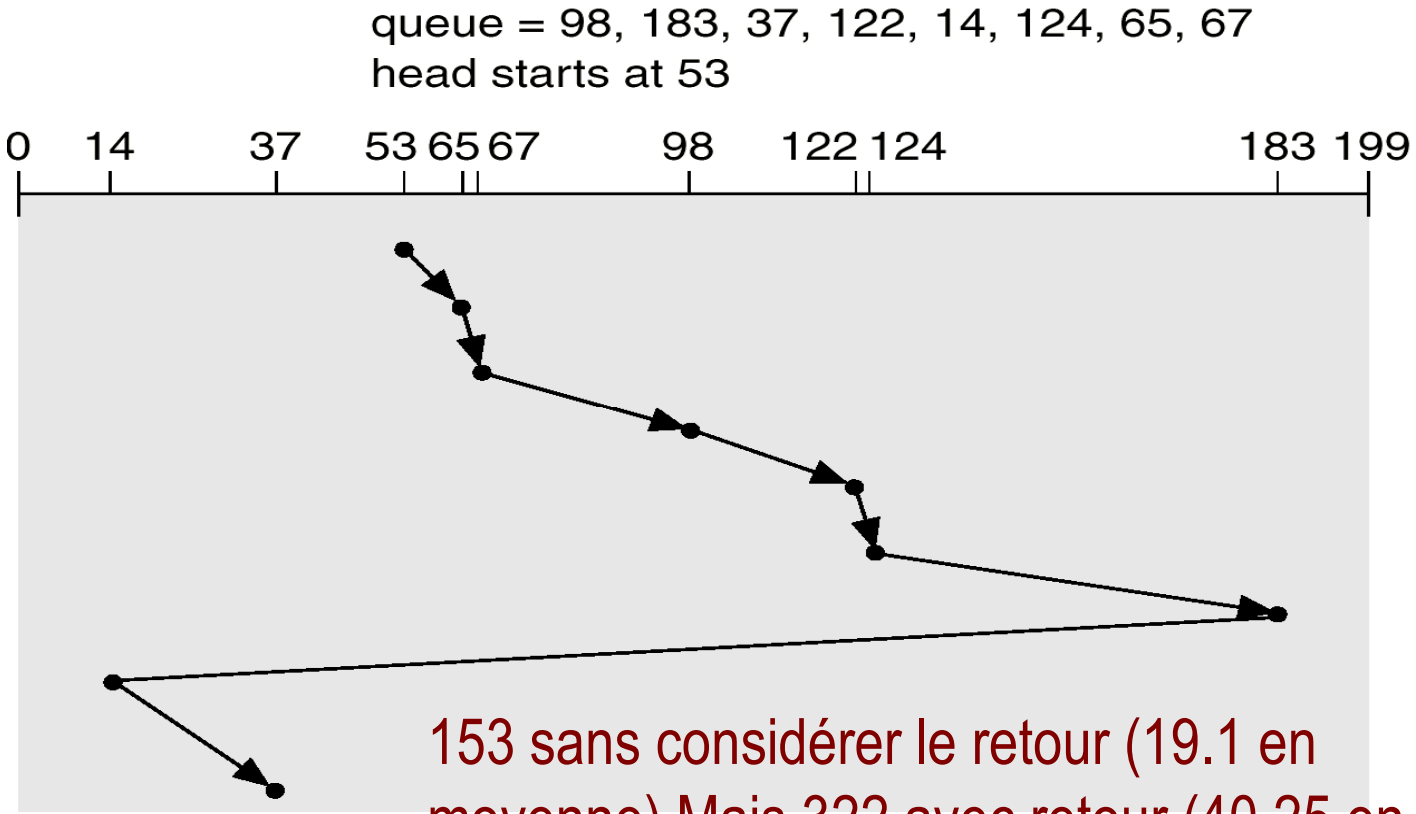
# C-SCAN: SCAN Circulaire

- **Va d'une extrémité à une autre. Il ne fournira de service que dans une direction, lorsqu'il atteint l'autre extrémité il retourne à celle où il a débuté sans fournir de service puis refait le même chemin.**



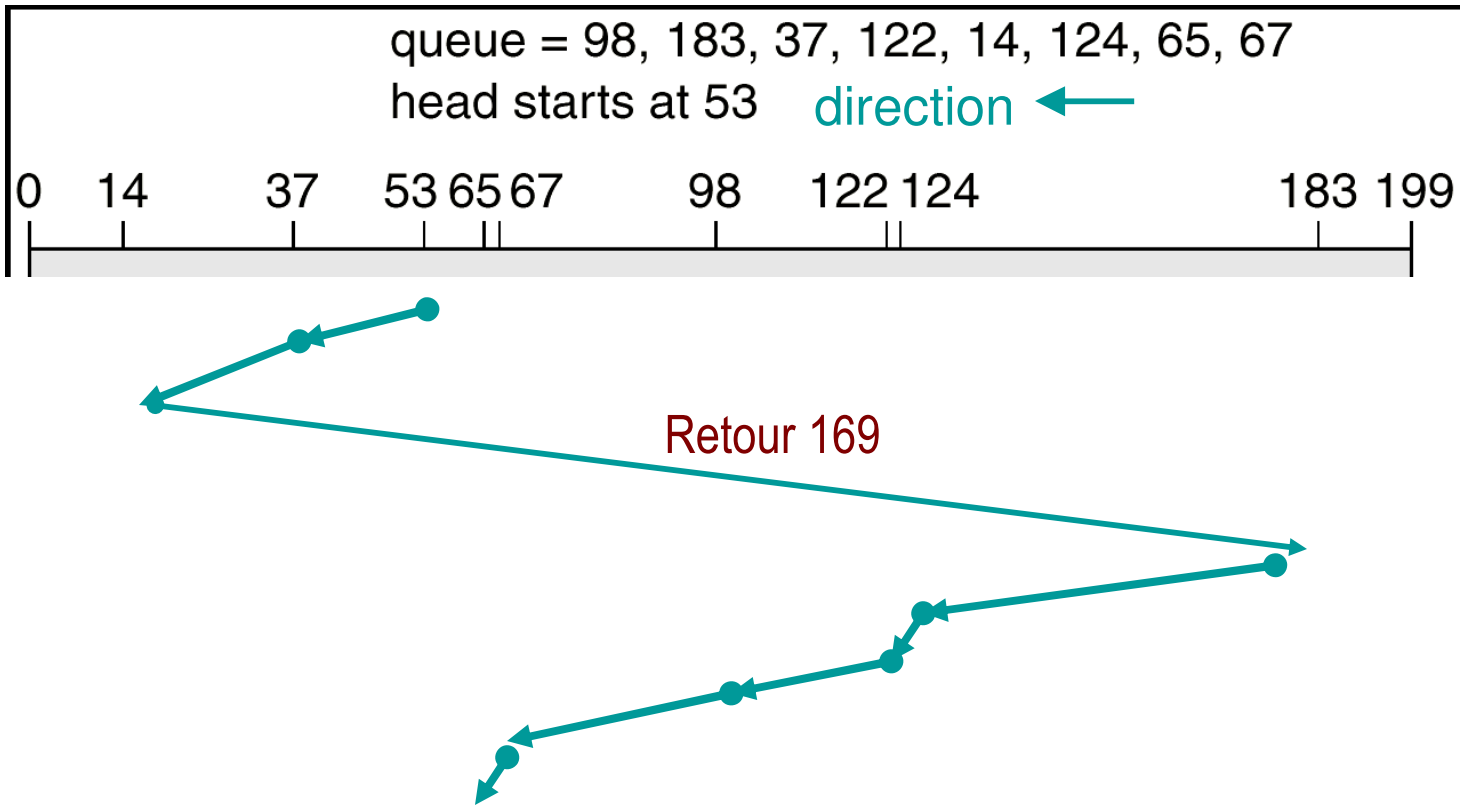
# C-LOOK

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




153 sans considérer le retour (19.1 en moyenne) Mais 322 avec retour (40.25 en moyenne)

# C-LOOK avec direction initiale opposée




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

# Exemple pratique…

- **Si on doit ramasser des gens de Gatineau Est à Aylmer, il serrait plus rapide de faire le tour à Gatineau, puis prendre l'autoroute jusqu'à Aylmer et ramasser le reste, au lieu d'arriver à Aylmer par rues régulières tout en ramassant des gens…**

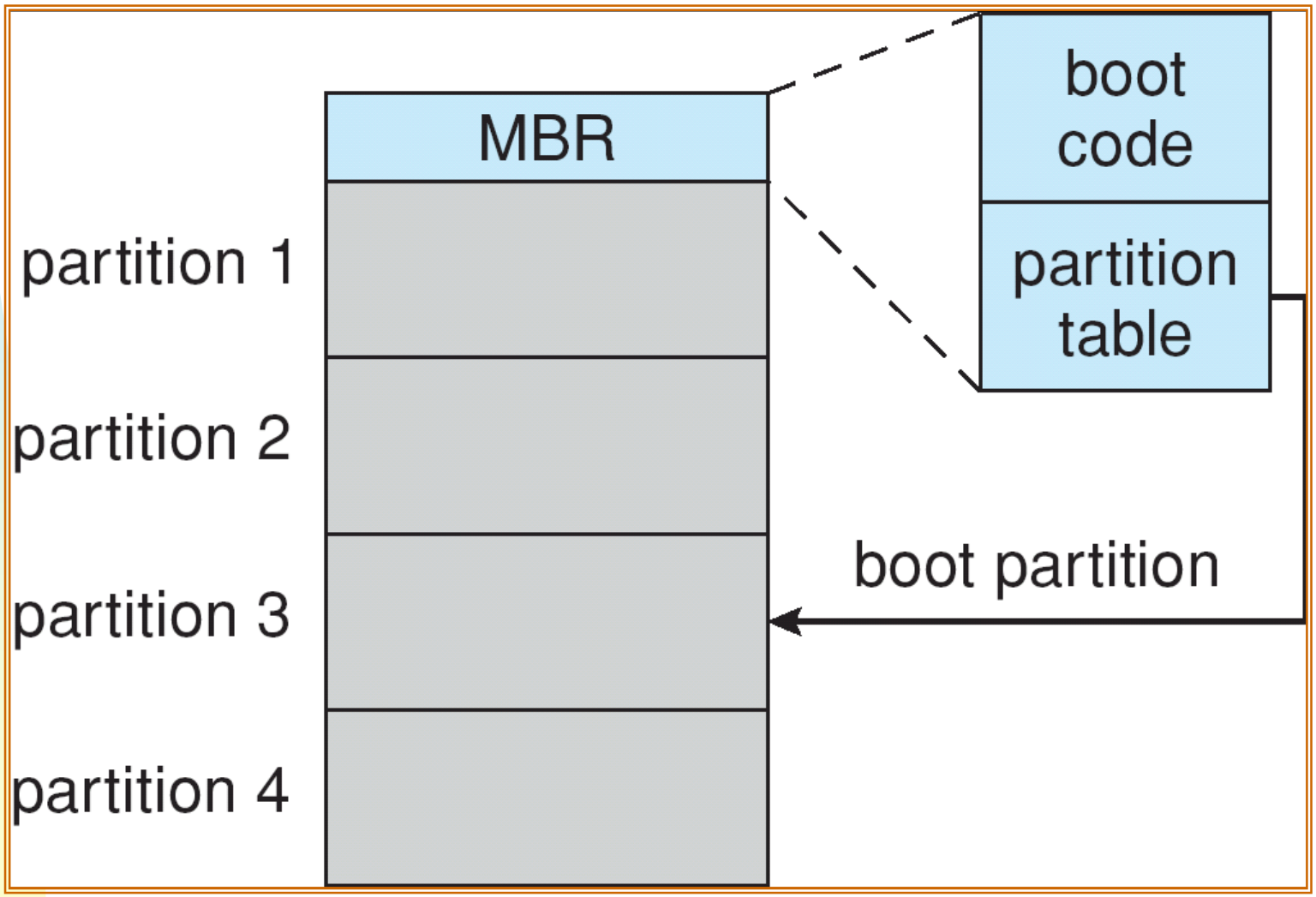
# Comparaison

- **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 du retour au début
  - L'organisation des fichiers et des répertoires
    - Les répertoires sont sur le disque aussi…
  - La longueur moyenne de la file
  - Le débit d'arrivée des requêtes

# Gestion des disques

- ***Formatage de bas niveau, i.e. formatage physique.***
  - Division du disque en secteurs manipuler par le contrôleur
  - Initialisation du secteur (fanion de début et de fin)
  - Se fait chez le fournisseur.
- ***Les partitions***
  - Division du disque en partition en groupes d'un ou plusieurs cylindres (pour stocker un système de fichier).
  - Définir la partition boot.
- ***Le format logique ou « création du système de fichier »***
  - *Écrire le SF sur le disque*
  - *i.e. tableau FAT, SB, tableau d'inodes, répertoire racine.*

# Démarrer d'un disque en Windows 2000



# Gestions de mauvais blocs

**Il y a des dizaines et centaines de millions de blocs dans un disque**

**Des mauvais blocs y sont présents, qu'en faire?**

- Il y a des blocs en surplus pour les remplacer
  - i.e. Disque contiens 100 blocs, mais a été fabriqué avec 110, donc 0..99 utilisé, 100..109 sont en surplus
  - Si le bloc 50 est endommagé, le contrôleur (après avoir été informé par le noyau) le remplace avec le bloc 100 (il devient le bloc 50 logiquement)
- Ceci affect l'efficacité des algorithmes d'ordonnancement des disques
- Pas de panique!  Les blocs en surplus se retrouvent dans chaque cylindre: **sector sparing** or **sector slipping**

# Gestion de l'espace de permutation en mémoire virtuelle (swap space) (12.6)

- **Nous avons vu comment les systèmes de mémoire virtuelle utilisent la mémoire secondaire**
- **Grande variété d'implémentations de systèmes d'espace de permutation dans différents SE**
- **L'espace de permutation**
  - peut être composé de fichiers normaux dans l'espace de disque utilisé par les autres fichiers,
  - ou peut avoir sa propre partition de disque
- **Peut être mis dans des disques plus efficaces**

# Gestion d'espace de permutation (disque) en Unix 4.3BSD

- **Pour chaque processus, il y a**
  - Un segment de texte = le programme
    - Ne change pas pendant exécution
  - Et il y a aussi un segment de données
    - Sa taille peut changer pendant l'exécution

# Gestion d'espace de permutation

- **Swap-space — mémoire virtuel qui utilise l'espace disque pour étendre la mémoire principale**
- **Où se trouve-t-il sur le disque?**
  - Dans un fichier normal du système
    - avantage: Simple, flexible
    - désavantage: Lent
  - Dans une partition séparée
    - Structures et algorithmes optimaux et spécifiques pour le swap.
    - Ex: fragmentation pas un vrai problème
    - avantage : plus rapide
    - désavantage : peut gaspiller de l'espace.

# Gestion d'espace de permutation

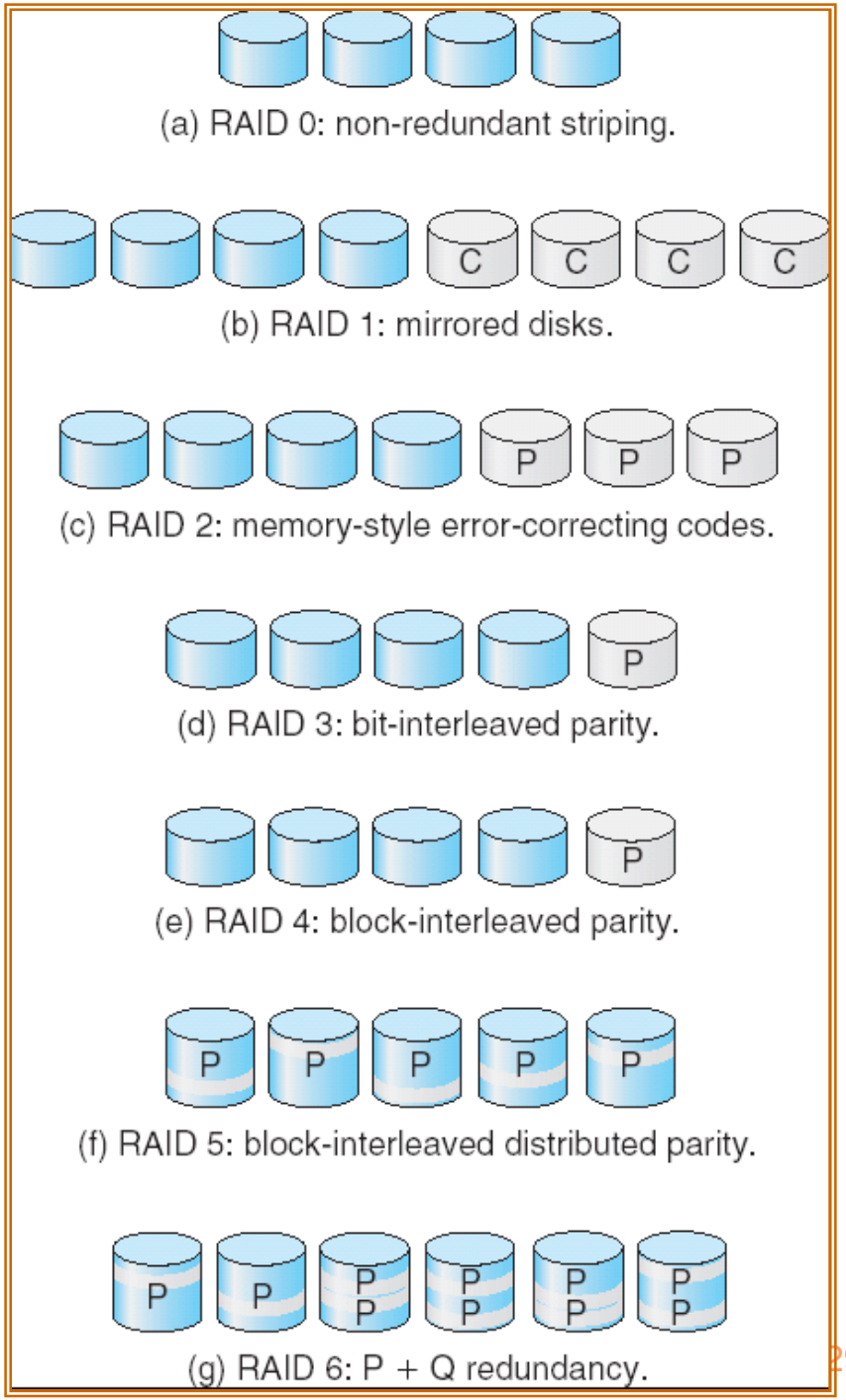
- **Quand alloué l'espace dans le swap?**
  - Quand le processus démarre – la page virtuelle est créée.
  - Quand la page est remplacée.
- **Qu'est qui doit-être géré?**
  - L'utilisation des « swap maps » pour permettre la gestion des pages durant le swap.

# Autres mécanismes en Unix

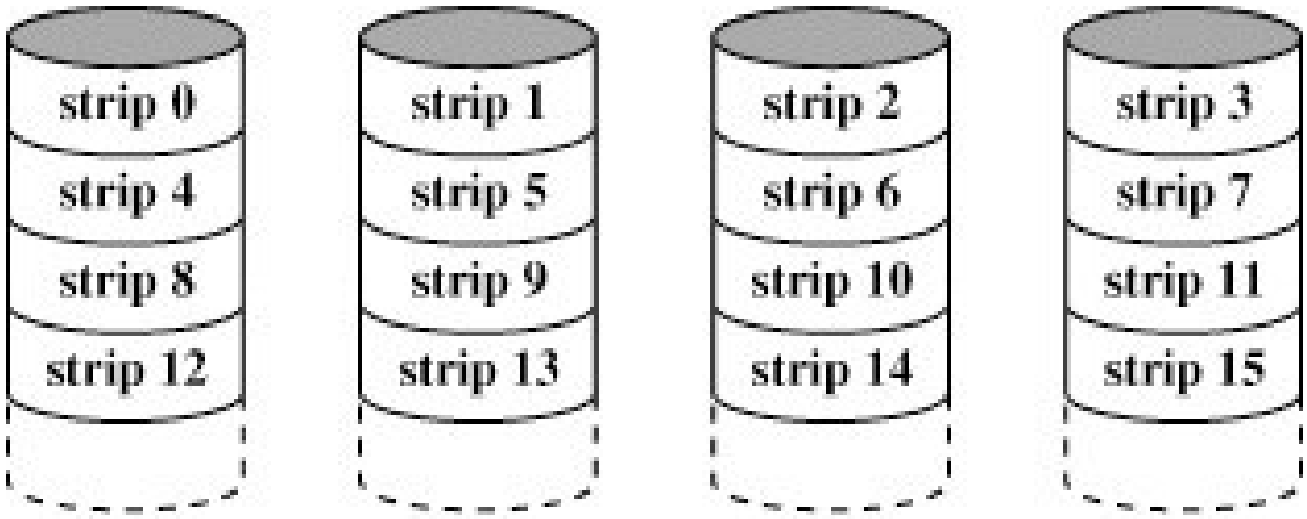
- Les mécanismes sont différents dans différentes versions de Unix
- Certaines versions fonctionnent avec certains mécanismes, comme la pagination, les systèmes compagnons (buddy) etc.

# Niveaux RAID

- **Striping: lecture de plusieurs disques en parallèle**
  - Entrelacement de bits
  - Entrelacement de blocs
- **Redondance: données redondantes pour recouvrir données,**
  - données dupliquées (disques miroirs)
  - Parité ou codes de correction
- **Exemples: Les six niveaux RAID stockent 4 disques de données**



# RAID: Redundant Array of Independent Disks

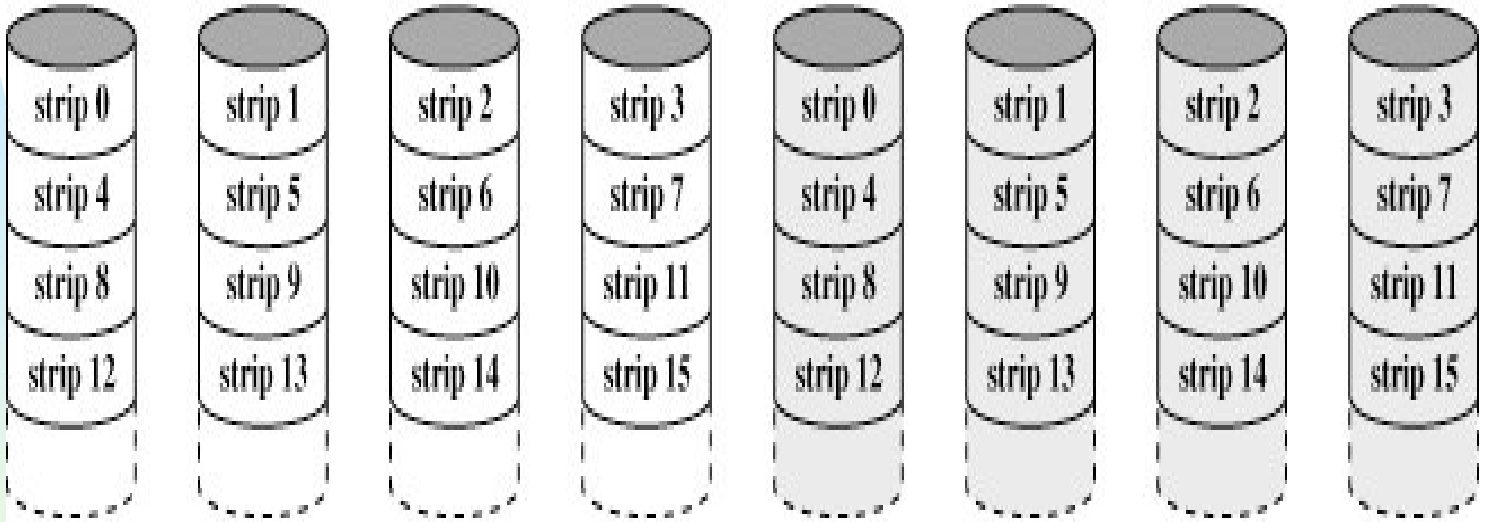



(a) RAID 0 (non-redundant)

Stallings

**En distribuant les données sur différents disques, il est probable qu'une grosse lecture puisse être faite en parallèle (au lieu de lire strip0 et strip1 en séquence, cette organisation permet de les lire en même temps)**

# Redondance dans RAID

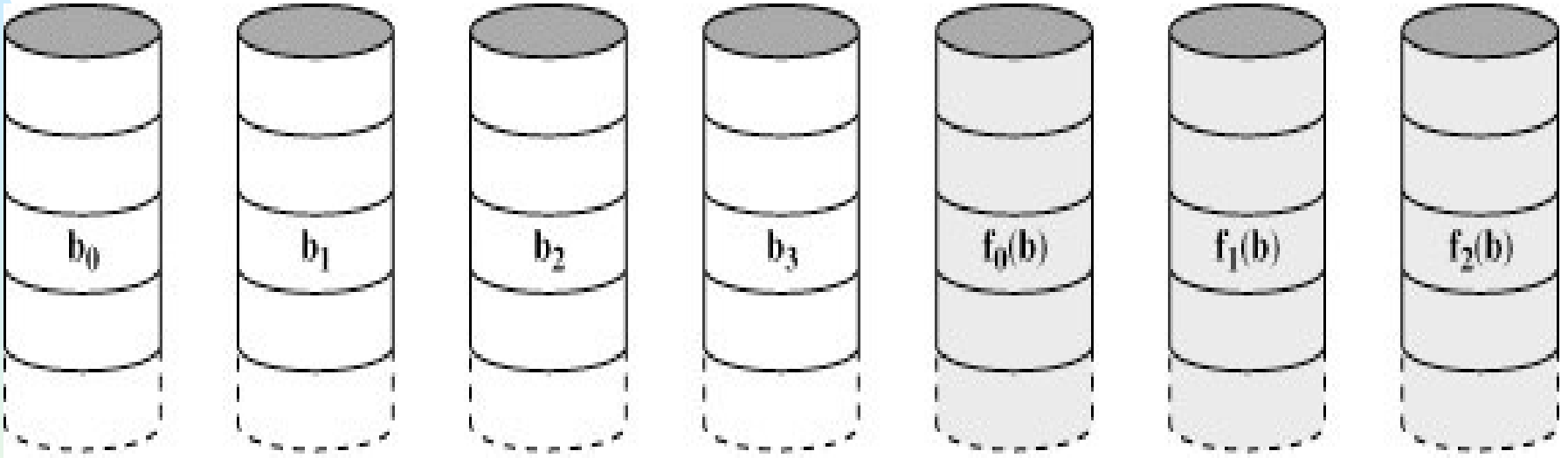



(b) RAID 1 (mirrored)

Stallings

**Dupliquer les données pour incrémenter le parallélisme et remédier aux pertes de données (coûteux mais utilisé en pratique)**

# RAID: Correction d'erreurs par codes de correction

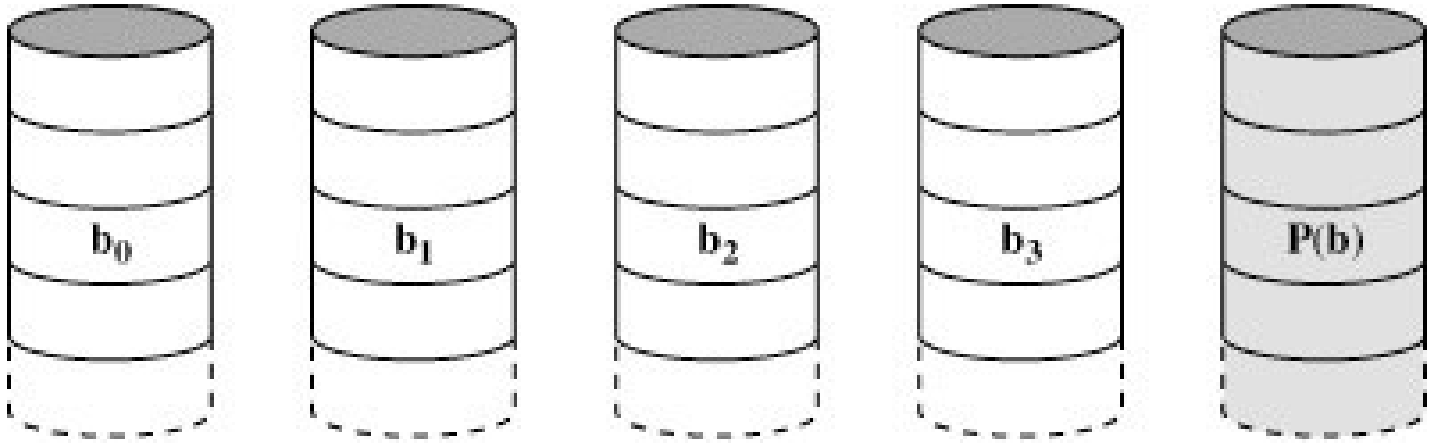



(c) RAID 2 (redundancy through Hamming code)

Stallings

**Les codes de correction d'erreurs des données enregistrées sur un disque sont sauvegardés sur un autre disque (plus de résistance aux erreurs)**

# RAID: Correction d'erreurs par parité



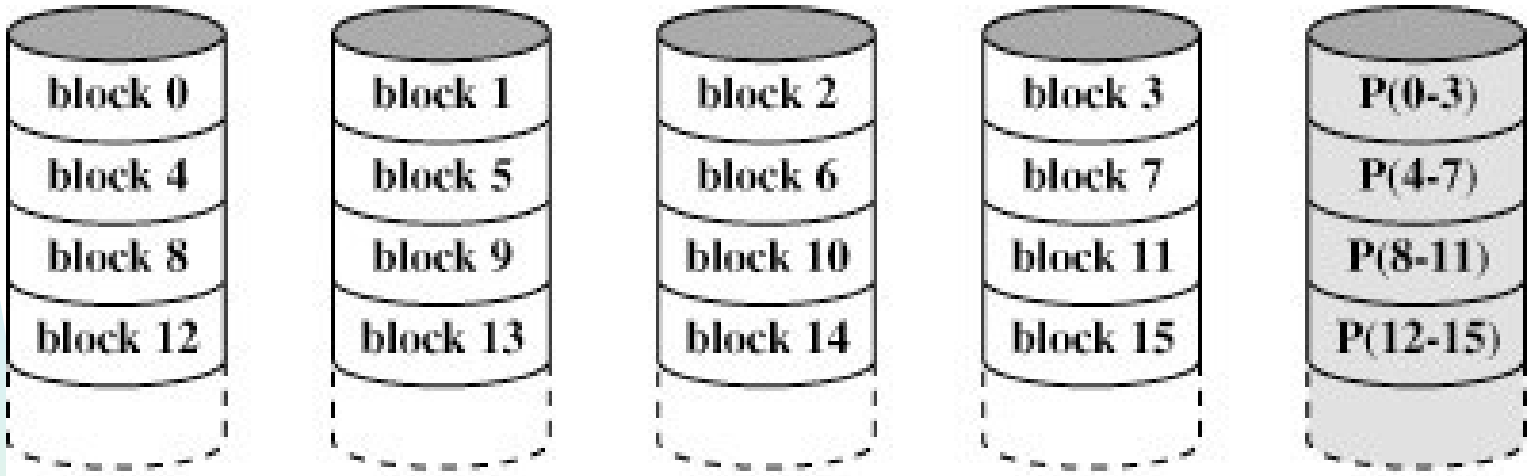

(d) RAID 3 (bit-interleaved parity)

**Figure 11.9 RAID Levels** (page 2 of 2)

Stallings

**Les bits de parité pour des données enregistrées sur un disque sont sauvegardés sur un autre disque**

# RAID: Correction d'erreurs par bits de parité avec entrelacement de blocs



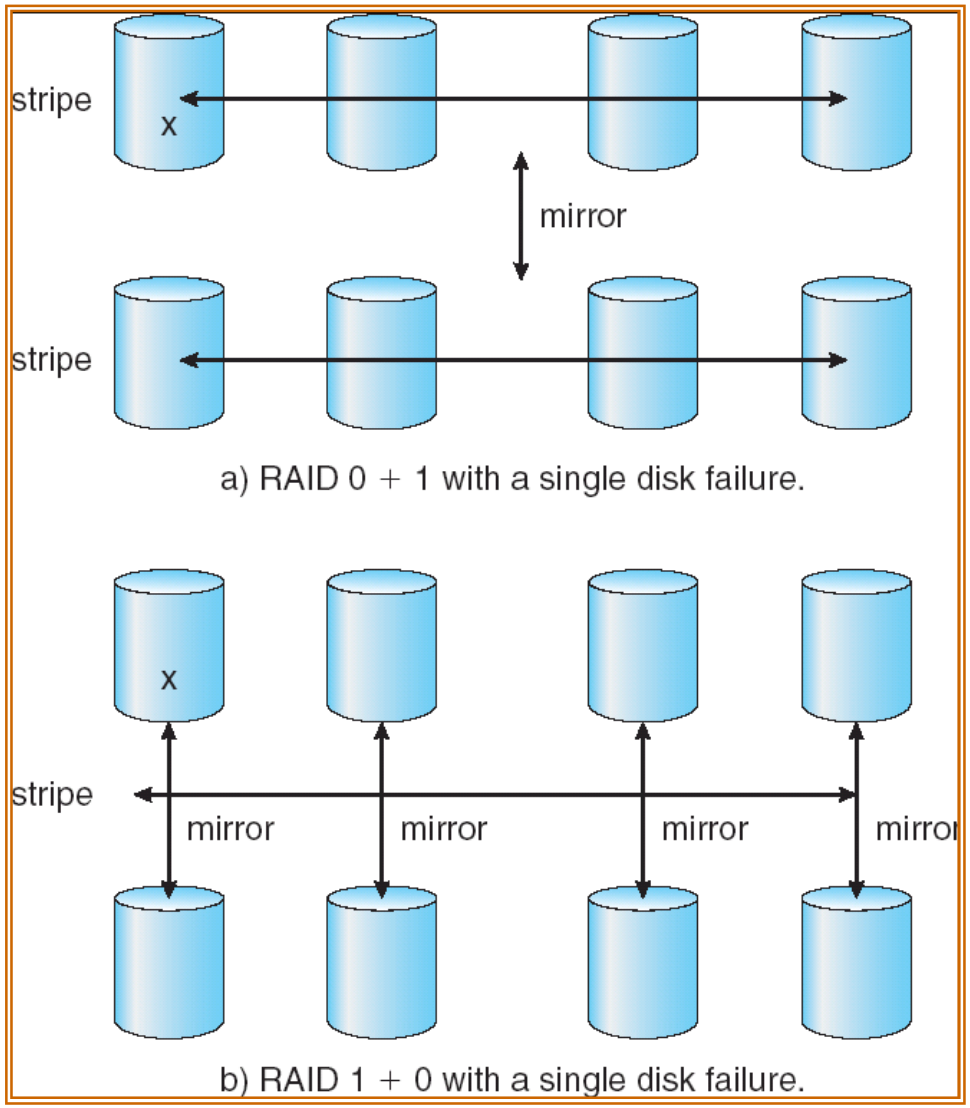

(e) RAID 4 (block-level parity)

**Figure 11.9 RAID Levels** (page 2 of 2)

Stallings

**Les codes de correction d'erreur pour des données enregistrées sur un disque sont sauvegardés sur un autre disque (plus de résistance aux erreurs)**

# RAID (0 + 1) et (1 + 0)


stripe
x
mirror
stripe
a) RAID 0 + 1 with a single disk failure.
x
stripe
mirror
mirror
mirror
mirror
b) RAID 1 + 0 with a single disk failure.

# Concepts importants du Module 10

- **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**
- **RAID: réorganisation des fichiers pour performance et résistance aux erreurs**