


# Systèmes d'Exploitation

## Implémentation des systèmes de fichiers

Didier Verna

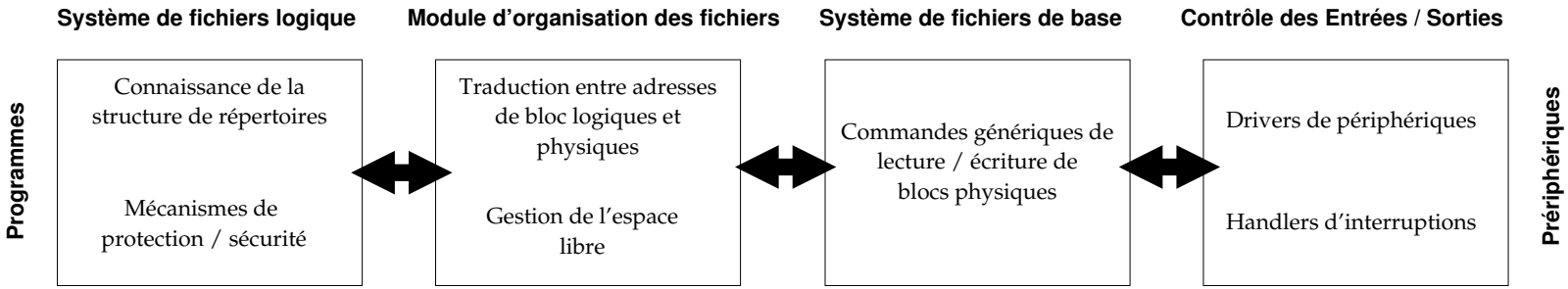
didier@lrde.epita.fr
http://www.lrde.epita.fr/~didier


# Table des matières


# Fonctionnalités requises



- **Définir l'interface utilisateur :** caractéristiques et attributs des fichiers, opérations sur les fichiers, structures des répertoires *etc.*
- **Définir l'interface matérielle :** structure de données, algorithmes, liaison entre système logique et dispositif de stockage en mémoire auxiliaire




# Allocation et efficacité

## Utiliser la mémoire auxiliaire le plus efficacement possible



- **Efficacité des entrées / sorties :**
  transferts de données par « blocs » (Cf. aussi le DMA).
  $\Longrightarrow$ Allocation d'espace disque également par bloc plutôt que par octet. Compromis espace / performance : blocs de 1K.
- **Accès direct :**
  par opposition aux périphériques à accès séquentiel (ex. bandes magnétiques).
  $\Longrightarrow$ Implémentation des méthodes d'accès aux fichiers facile.

# Allocation contiguë
## Exemple : IBM VM/CMS



- **Principe**
  - Fichiers stockés par blocs contigus sur le disque
  - Temps de positionnement des têtes minimal
  - Entrée de répertoire : adresse du premier bloc et longueur (en nombre de blocs)
  - Accès direct et séquentiel faciles à implémenter : il suffit de mémoriser l'adresse du premier bloc
  - Gestion de l'espace libre : Cf. $*$-fit, fragmentation externe, compactage *etc.*
- **Problème majeur :** fichiers de taille variable
  - **Trop d'espace :** fragmentation interne
  - **Pas assez d'espace :** déplacement (coûteux) du fichier. Pas toujours possible.
- **Utilisation actuelle :** CD / DVD-ROM


# Allocation chaînée



- **Principe**
  - Fichier = chaîne non contiguë de blocs disque
  - Chaque bloc se termine par un pointeur sur le bloc suivant
  - Une entrée de répertoire contient un pointeur sur le premier bloc
- **Avantages**
  - Pas de fragmentation externe
  - Pas de limite de taille
- **Inconvénients**
  - Accès direct inefficace
  - Fiabilité : perte de pointeur critique. Solutions : listes doublement chaînées, reproduction du nom de fichier et numéro de bloc dans chaque bloc *etc.* Coûteux dans tous les cas.

# File Allocation Table (FAT)

## Variante de l'allocation chaînée (MS-DOS, OS/2)



- **Principe**
  - Une FAT au début de chaque partition
  - Table indexée par numéros de bloc
  - Chaque entrée pointe sur le numéro de bloc suivant
  - Une entrée de répertoire contient un pointeur sur le premier bloc
- **Avantages** (à condition de mettre la FAT en mémoire)
  - Moins de risque de corruption
  - Accès séquentiel aussi rapide
  - Accès direct (presque) aussi rapide que l'accès séquentiel
- **Inconvénients**
  - Taille de la FAT
    Disque de 20G, blocs de 1Kb $\Longrightarrow$ FAT de 80M

# Allocation indexée (« i-nodes »)

## Schéma analogue à la pagination



- **Principe**
  - Chaque fichier possède un bloc d'index ($1^{er}$ bloc)
  - Une entrée de répertoire pointe sur le bloc d'index
  - La $i^e$ entrée du bloc d'index pointe sur le $i^e$ bloc de données du fichier
- **Avantages**
  - Implémentation efficace de l'accès direct
  - Table des i-nodes de taille proportionnelle au nombre de fichiers (Cf. FAT : taille du disque)
- **Inconvénients**
  - Fragmentation interne plus grande qu'avec l'allocation chaînée
  - Problème de la taille des index


# Schémas d'indexation

## Comment stocker des index sur plusieurs blocs ?

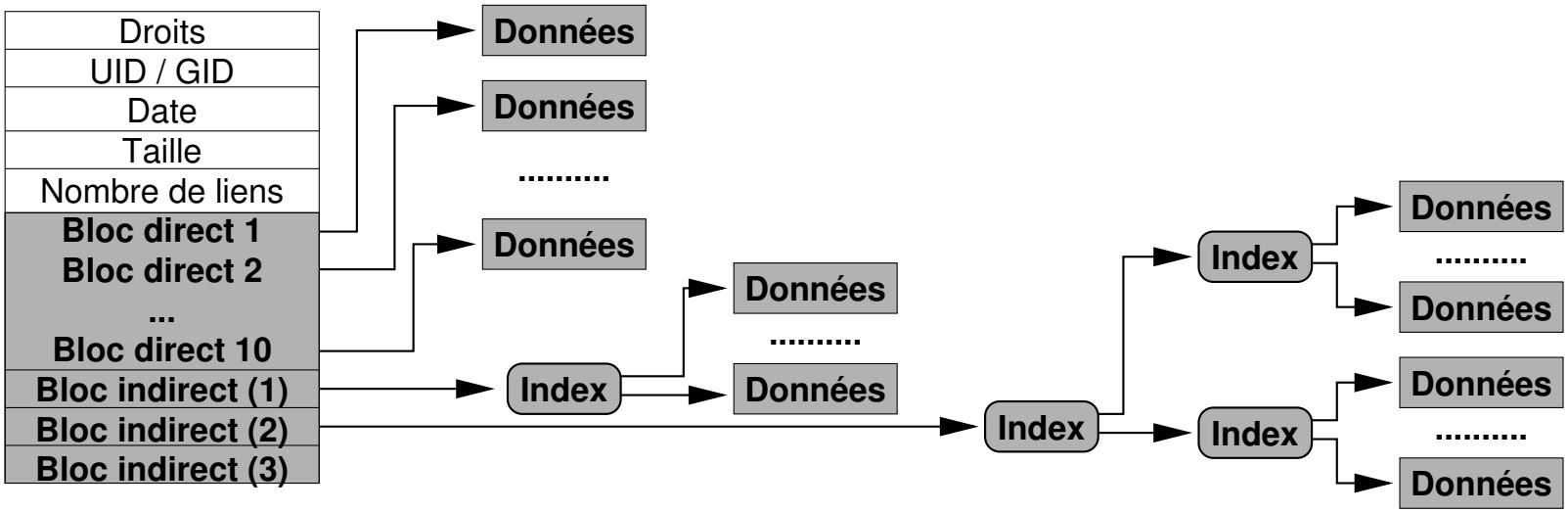


- **Schéma chaîné :** réserver le dernier mot du bloc d'index pour un pointeur sur le bloc d'index suivant.
- **Index à multiniveaux :** analogue à la pagination à plusieurs niveaux. Index pointant sur des index pointant . . . sur des blocs de données. Indexation à 2 niveaux $\Longrightarrow$ Fichiers de l'ordre de 10 Go.
- **Schéma combiné :** les premières entrées pointent sur des blocs de données. Les suivantes pointent sur des blocs d'index de différents niveaux.

# Structure des i-node

## Schéma combiné d'Unix (BSD, System V)






# Implémentation des répertoires



- **Liste linéaire :** noms de fichiers / attributs / pointeurs sur des blocs de données, ou noms de fichiers / pointeurs sur des i-nodes.
  - **Avantage :** implémentation simple
  - **Inconvénient :** recherche linéaire dans la liste
    - Caching des répertoires fréquemment utilisés. Triage de la liste (arbre binaire chaîné, recherche dichotomique *etc.*)
- **Table de hachage :** transformation des noms de fichiers en index de tableau (fonction de hachage).
  - **Avantage :** temps de recherche plus court
  - **Inconvénient :** tables de taille fixe, fonction de hachage dépendante de la taille de la table


# Implémentation du swap



- **Utilisation**
  - Stockage des images mémoires des processus
  - Stockage des pages supprimées de la mémoire
- **Emplacement**
  - **Système de fichiers** (Windows). Utilisation des primitives standard de création, destruction, allocation d'espace. Implémentation facile. Inconvénient : lenteur.
  - **Partition distincte** Pas de système de fichiers. Plus de fragmentation interne, mais accès plus efficace. Inconvénient : réservation de l'espace au moment du formattage disque.
- **Remarque :** certains systèmes (ex. Solaris 2) permettent l'existence de plusieurs espaces de swap, en partition ou en fichiers.


# Représentation de l'espace libre



- **Vecteur binaire :** (Mac OS) représentation de chaque bloc dans l'ordre par un bit d'occupation. Instructions matérielles facilitant la manipulation (i386, 68030). Efficace à condition que le vecteur en entier soit maintenu en mémoire vive.
- **Liste chaînée :** un pointeur sur le premier bloc libre. Chaque bloc libre pointe sur le suivant. Inefficace en cas de besoin de plus d'un bloc.
- **Groupement :** stockage des adresses de blocs libres dans autant de blocs libres que nécessaire. Efficace grâce au principe de lecture par bloc.
- **Compactage :** en général, plusieurs blocs libres se suivent. Stockage d'une adresse de bloc et du nombre de blocs libres contigus. Liste plus courte que dans les cas précédents.


# Corruption des systèmes de fichiers

## Profiter de la redondance . . .



- **Vérification de blocs :** parcours de tous les i-nodes et calcul de deux compteurs par blocs : présence dans des fichiers et dans l'espace libre. Chaque bloc ne devrait être présent qu'une fois : en libre ou en fichier.
- **Vérification de fichiers :** parcours de tous les répertoires et calcul d'un compteur de références par fichier. Comparaison avec le compteur de référence de l'i-node.

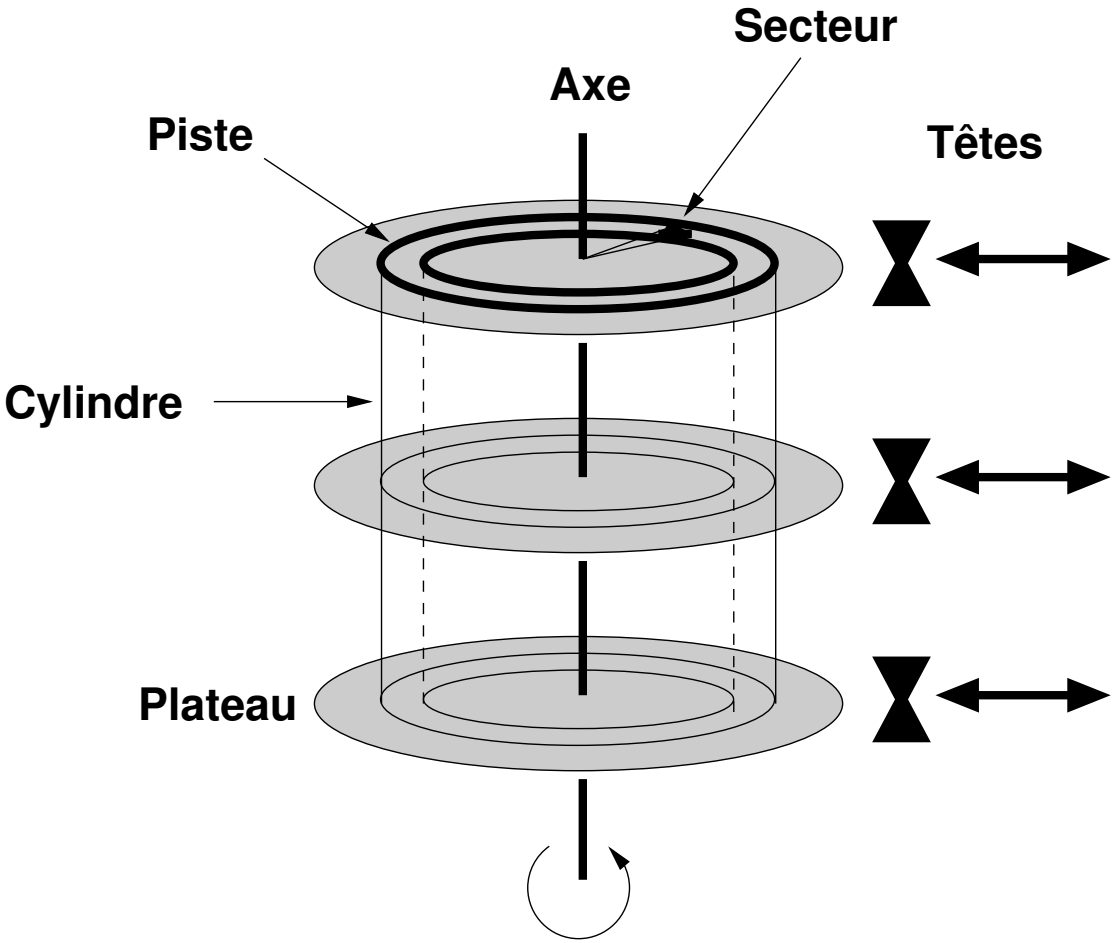

# RAM $= O(ns)$, disque = $O(ms)$



- **Caching :** Cf. algorithmes de pagination (LRU *etc.*). Mais attention aux blocs critiques pour la cohérence du système de fichiers : sauvegarder immédiatement tous les blocs structurels, ou simplement tous les blocs (« write-through cache »).
- **Read-ahead :** lire à l'avance plus d'un bloc. Analogue à la prépagination. Intéressant pour l'accès séquentiel.
- **Organisation physique :** profiter de blocs libres contigüs, maintenir une proximité entre les i-nodes et les blocs de données correspondants.

# Structure des disques






# Délai de réception d'une E/S



- **Temps de positionnement :** déplacement des têtes sur la piste (le cylindre) considéré(e).
- **Temps de latence :** passage du bloc sous la tête par rotation des plateaux.
- **Temps de transfert :** temps effectif de transfert des données vers (ou depuis) la mémoire.


# Algorithmes d'ordonnancement des disques



- **FCFS :** servir la requête arrivée en premier. Implémentation simple. Intrinsèquement juste. Risque de déplacement « sauvage » des têtes.
- **SSTF :** servir la requête la plus proche de la position actuelle. Basé sur l'écart entre pistes. Analogue au SJF. Risque de famine. Non optimal (contrairement au SJF).
- **SCAN :** balayage avant / arrière perpétuel des pistes. Service des requêtes en fonction de la position des têtes. Aussi appelé « ascenseur » ou « chasse-neige ».
- **C-SCAN :** SCAN circulaire. En fin de disque, retour au début sans traiter aucune requête.
- **[C-]LOOK :** [C-]SCAN avec retour quand plus aucune requête n'existe dans la direction courante. Le plus utilisé.