

# LES SYSTÈMES D'EXPLOITATION

## LES SYSTÈMES DE FICHIERS

🎓 3A - Cours Ingénieurs - Dominante Informatique et Numérique





🏛️ CentraleSupélec - Université Paris-Saclay - 2025/2026



**Idir AIT SADOUNE**

[idir.aitsadoue@centralesupelec.fr](mailto:idir.aitsadoue@centralesupelec.fr)

# PLAN

-  La notion de fichier
-  Les répertoires
-  Les systèmes de fichiers
-  Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

# LE STOCKAGE SECONDAIRE

- Le **stockage secondaire** conserve des **programmes** et des **données**.
- L'**OS** masque la **complexité** et la **diversité** des unités de stockage (matériel, système de fichiers, ...) grâce à:
  - Une **vue logique** des données :
    - ▢ **fichiers**: unité de stockage logique
    - ▢ **répertoires**: classement arborescent
    - ▢ **volumes montés**: vue globale des systèmes de fichiers
  - Une **organisation physique** des espaces de stockage :
    - ▢ découpage en **blocs**
    - ▢ affectation et libération de **blocs**
  - Un **système d'entrées/sorties** (**non traité dans ce cours**) :
    - ▢ gestion du caches, algorithmes d'optimisation des accès,
    - ▢ pilotes gérant les communications avec les périphériques

# PLAN

- > La notion de fichier
- > Les répertoires
- > Les systèmes de fichiers
- > Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

# LA NOTION DE FICHIER

- **Fichier**
  - une **collection nommée** d'information accessibles via un périphérique.
- **Unité logique**
  - indépendante du support physique (périphérique)
  - abstraction des propriétés physiques
- **Type de fichier**
  - code source, données, bibliothèque, exécutable, ...
  - généralement indiqué par son **extension**
  - un type de fichier → une **structure spécifique**

# EXEMPLES

- **Fichiers texte:** `.txt`
  - données textuelles à l'usage de l'utilisateur humain
  - succession de caractères ...
- **Fichiers source:** `.c`, `.java`, ...
  - fourni par un humain pour être traité par la machine
  - composés d'instructions ...
- **Exécutable:** `.exe`, ...
  - construits par un compilateur à partir d'un fichier source
  - succession d'octets ...

# FILE CONTROL BLOCK - FCB

**Structure de données** de l'**OS** pour stocker les informations nécessaires à la gestion des fichiers

<b>Nom</b>	indépendant de l'OS, lisible
<b>Identifiant</b>	numérique, unique, pour l'OS
<b>Type</b>	extension ou en-tête de fichier
<b>Emplacement</b>	pointeur sur un périphérique
<b>Taille</b>	en octets ou en blocs
<b>Protection</b>	lecture, écriture, exécution ...
<b>Date(s)</b>	création, modification, accès ...
<b>Utilisateur</b>	propriétaire du fichier

...

# PROTECTION

- **Partage de fichiers**
  - rendre **accessible** à un **utilisateur B** un fichier de l'**utilisateur A**.
- **Politique de protection**
  - définir qui peut accéder à quel(s) fichier(s)
    - ▢ identifiant utilisateur → identifiant processus
    - ▢ contrôle d'accès dans le **FCB**
  - spécifier pour chaque fichier la liste des **sujets** autorisés à effectuer un **type d'accès**



# TECHNIQUES DE PROTECTION

- Liste de contrôle d'accès (ACL)
  - utilisateur → droits
  - l'ensemble des utilisateurs doit être connu a priori
  - ✗ taille du FCB grossit avec le nombre d'utilisateurs
- Mot de passe
  - 1 mot de passe par fichier × type d'accès (r, w, ...)
  - ✗ pas très pratique → peu utilisé
- Classes d'utilisateurs
  - Exemple: Propriétaires vs Autres

# TECHNIQUES DE PROTECTION

- **Notion de groupe**

- Ensemble de groupes définis a priori
  - Ex: `admin`, `dev-disque`, `user-disque`, `dev-ram`, `user-ram`
- **FCB**: 1 utilisateur + 1 groupe (propriétaires)
  - Ex: `toto.c`, `u=batman`, `g=dev-disque`
- Utilisateur → liste de groupes
  - Ex: `robin`, `g=[user-disque, dev-ram]`
    - ➡ `robin` n'a pas accès à `toto.c`

- Dans les systèmes **POSIX** on distingue :

- **3 modes** (lecture, écriture, exécution)
- **3 catégories de sujets** (le propriétaire, son groupe et le reste des sujets)

# TECHNIQUES DE PROTECTION

- Dans les systèmes **POSIX** on distingue :
  - **3 modes** (lecture, écriture, exécution)
  - **3 catégories de sujets** (le propriétaire, son groupe et le reste des sujets)
- Exemple → **Unix**
  - $3 \times 3$  bits par fichier

```
$ ls -l
total 248
drwx----- 6 nico prof      4096 nov. 20 12:06 private
-rw----- 1 nico prof      2356 dec. 5 15:30 notes.ods
drwxrwx--- 8 nico prof      4096 dec. 4 17:31 doc
-rw-rw-r-- 1 joe  student    36 jan. 21 16:49 programme.c
-rwxrwxr-x 1 joe  student   996 jan. 28 10:53 programme
...
```

# OPÉRATIONS SUR UN FICHER

- Appels systèmes de base
  - Création: allocation espace + entrée répertoire
  - Lecture: pointeur de lecture
  - Écriture: pointeur d'écriture
  - Repositionnement: déplacer un pointeur
  - Suppression: retrait de l'entrée dans le répertoire
  - Troncature: vider mais garder l'entrée
- Opérations composées
  - Ex: copie, renommage
  - ➡ effectuées à partir des appels systèmes de base

# OUVERTURE DE FICHIER

## ✗ Problème

- Nécessité d'**accéder au FCB** à **chaque opération** sur le fichier
- Le **FCB** est stocké dans le répertoire du périphérique
- Très coûteux en **accès disque** (donc en temps)!

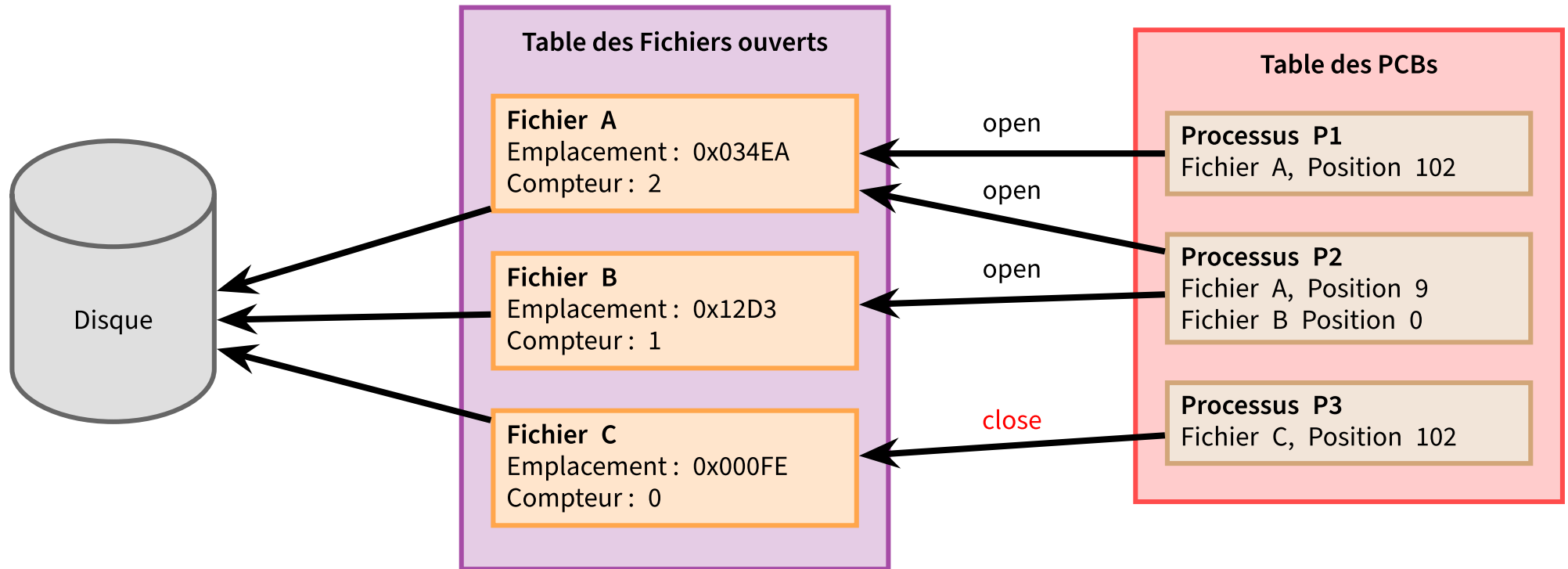
## ✓ Solution

- L'appel système **open** permet de **charger le FCB en mémoire**.
- L'**OS** impose que tout accès à un fichier soit **précédé d'une ouverture**.

# TABLE DES FICHIERS OUVERTS

- La **table des fichiers ouverts** de l'**OS** contient l'ensemble des **FCB** des **fichiers ouverts**.
  - **ouverture** → chargement du **FCB** + ajout dans la table
  - **fermeture** → retrait de la table
  - ➡ les **FCB** sont chargés en **RAM**
  - ➡ pas d'impact sur le fichier!
- Gestion par l'**OS**
  - **implicite** → **open** implicite au premier accès
  - **explicite** → **exception** si le fichier n'a pas été ouvert avant
  - **une table de fichiers ouverts globale** avec compteurs
  - **une table par processus** → fermeture à la terminaison
  - ➡ possibilité d'**interdire l'accès aux autres processus**

# TABLE DES FICHIERS OUVERTS



# PLAN

- La notion de fichier
- Les répertoires
- Les systèmes de fichiers
- Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)



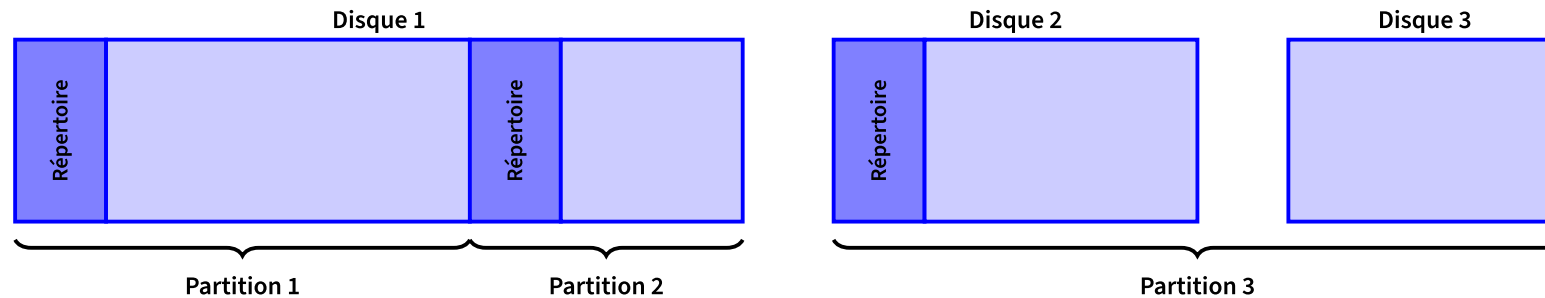
# NOTION DE RÉPERTOIRE

- Le **répertoire** est la structure de **stockage des informations** des fichiers (les **FCB**) dans les **supports de stockage**.
  - ➡ **entrée du répertoire** = identifiant du fichier et/ou nom du fichier
  - ➡ **contenu du répertoire** = **FCB** des fichiers
- L'**OS** récupère les informations sur les fichiers dans **le répertoire**

# STRUCTURE DES DISQUES

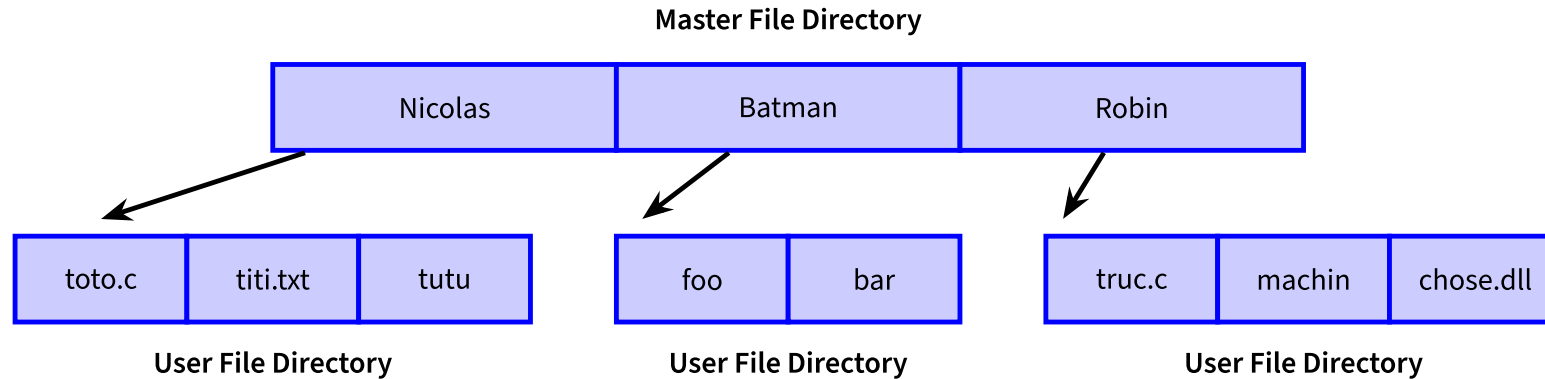
- **Disque** → Structure physique
- **Partition** → Structure logique (disque "virtuel")
  - Base → 1 disque = 1 partition
  - 1 disque = N partitions
  - 1 partition = 1 ou N disques (selon **OS**)

# STRUCTURE À 1 NIVEAU



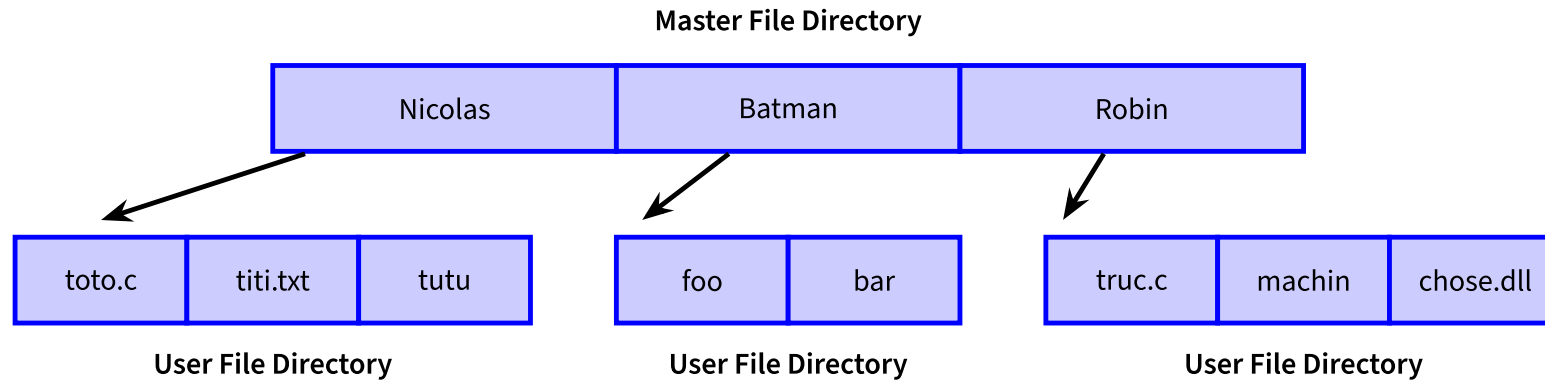
- **Répertoire** → Un répertoire par partition → l'ensemble des **FCB**
  - Nom/identifiant → **FCB**
    - Exemple: **MSDOS** et **Windows** → 11 octets (8 nom + 3 extension)
    - Exemple: **Unix** et **Mac** → 255 octets
- ✗ Taille du répertoire proportionnelle au nombre de fichiers
  - ➡ borner le nombre de fichiers . . .
- ✗ Utilisateur: organiser les fichiers, unicité de nom, . . .

# STRUCTURE À 2 NIVEAUX



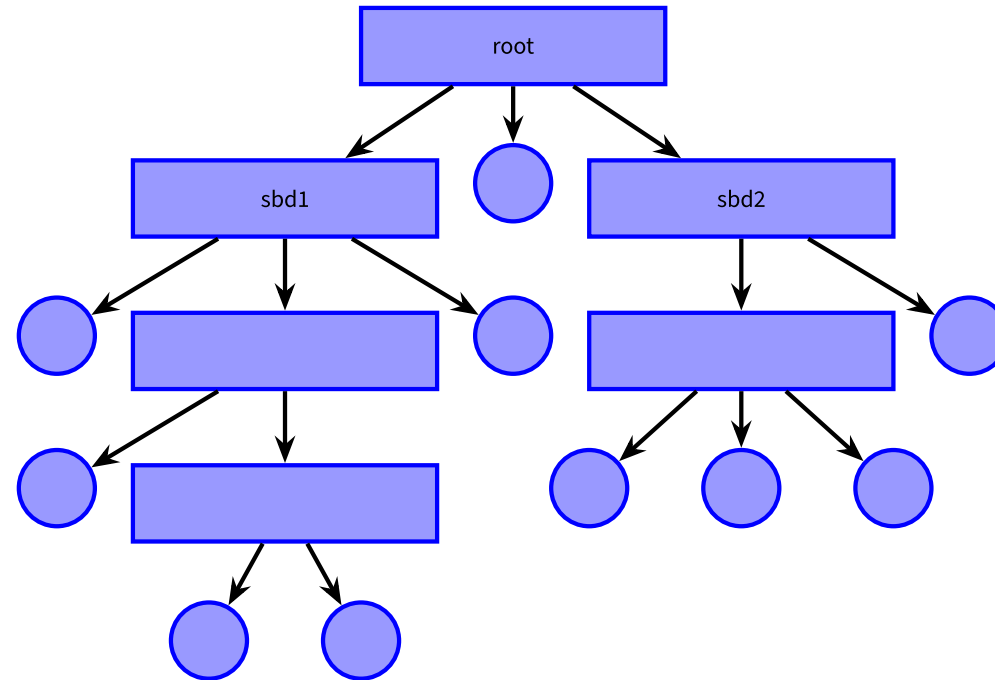
- Un répertoire par utilisateur
  - Identifiant + Nom → **FCB**
  - Répertoire des utilisateurs: **Master File Directory (MFD)**
    - ▢ Identifiant → **User File Directory**
  - Répertoire par utilisateur: **User File Directory (UFD)**
    - ▢ Nom → **FCB**

# STRUCTURE À 2 NIVEAUX



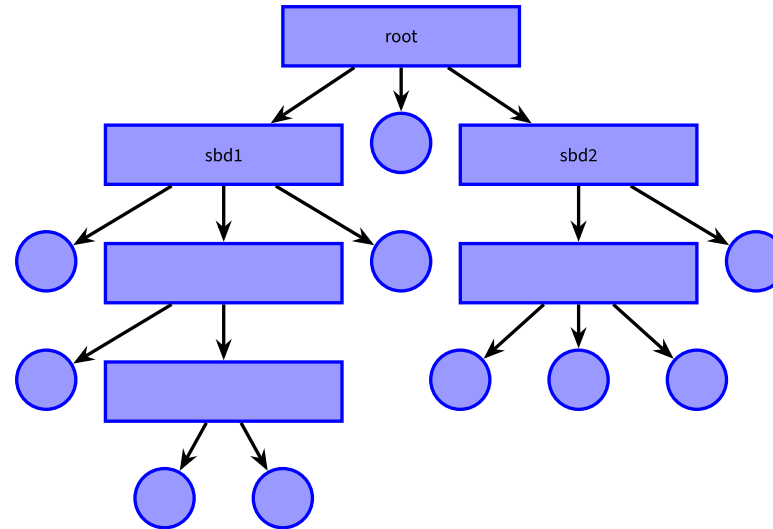
- ✓ Pas beaucoup plus coûteux en taille
- ✓ Utilisateur: organiser les fichiers, unicité de nom, . . .
- ✗ Taille des répertoires proportionnelle au nombre de fichiers
- ✗ Partage de fichiers

# STRUCTURE ARBORESCENTE



- Généralisation de la structure à 2 niveaux:
  - Répertoire racine → **Master File Directory (MFD)**
  - **Sous-répertoires**, pouvant à leur tour jouer le rôle de **MFD**

# STRUCTURE ARBORESCENTE



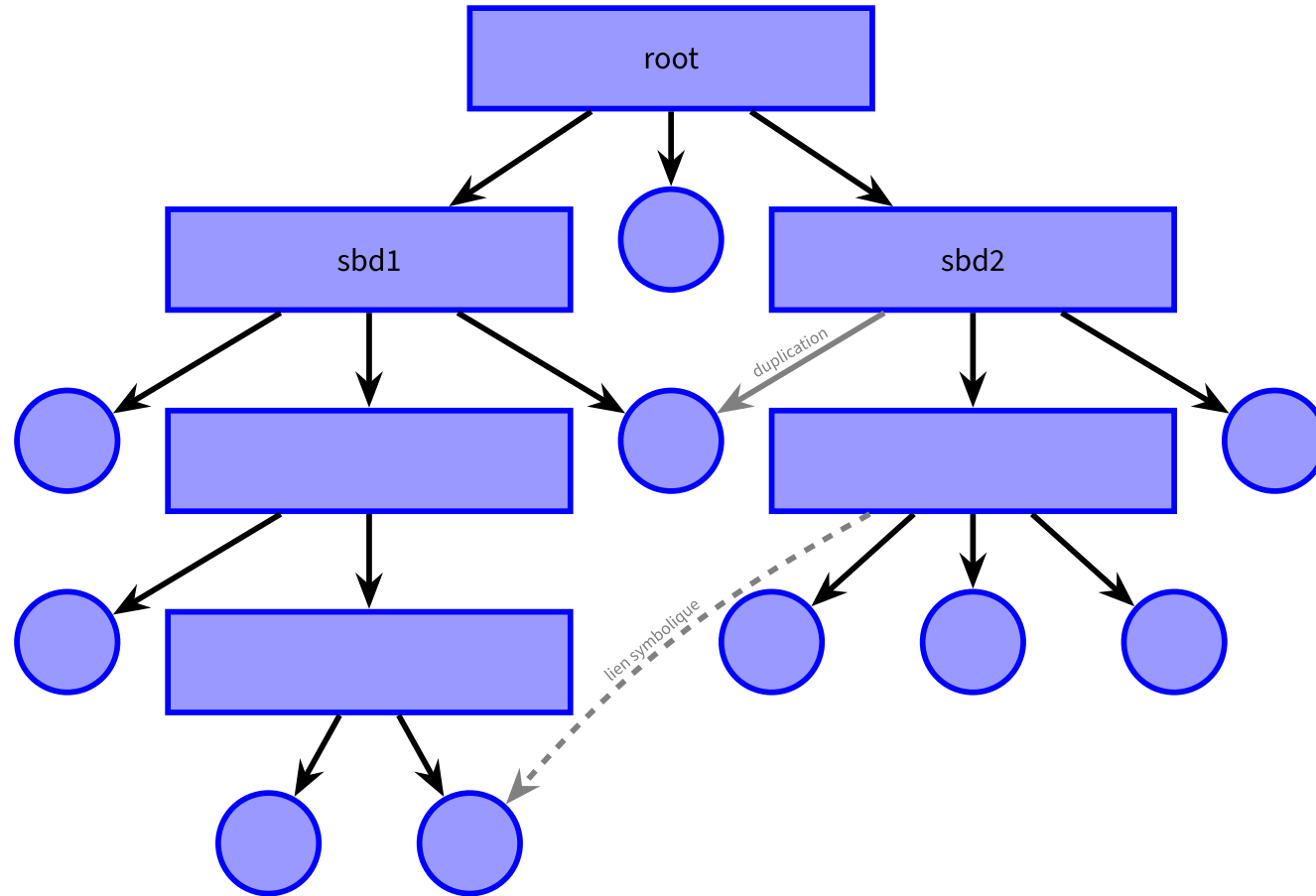
- **Fichiers**
  - Bit "répertoire" dans le FCB
  - Nom unique = chemin depuis la racine (chemin absolu)
- **OS** → Répertoire courant (par processus)
  - ➡ Recherche à partir du répertoire courant (chemin relatif)
  - ➡ Recherche par défaut (PATH)

# STRUCTURE EN GRAPHE

- Généralisation de l'arbre avec des **liens**
  - ➡ Graphe **acyclique**
- **Liens**: **référencer** un fichier décrit dans un autre répertoire
  - ➡ bit **"lien"** dans le répertoire + chemin absolu
- **Extension**: **duplication**
  - ➡ **FCB** recopié → copie et original indiscernables
  - ➡ Compteur de liens (pour libérer l'espace sur le support physique)



# STRUCTURE EN GRAPHE



# TECHNIQUES DE PROTECTION

- Dans les systèmes **POSIX** on distingue :
  - **3 modes** (lecture, écriture, exécution)
  - **3 catégories de sujets** (le propriétaire, son groupe et le reste des sujets)

```
$ ls -l
total 248
drwx----- 6 nico prof      4096 nov.  20 12:06 private
-rw----- 1 nico prof      2356 dec.   5 15:30 notes.ods
drwxrwx--- 8 nico prof      4096 dec.   4 17:31 doc
-rw-rw-r-- 1 joe  student    36 jan.  21 16:49 programme.c
-rwxrwxr-x 1 joe  student   996 jan.  28 10:53 programme
...
```

Mode	Lecture (r)	Écriture (w)	Exécution (x)
<b>Fichier</b>	mode lecture	mode écriture	exécution du fichier
<b>Répertoire</b>	lister le contenu	créer, renommer et supprimer un fichier	accéder au répertoire et à son contenu

# PLAN

- La notion de fichier
- Les répertoires
- Les systèmes de fichiers
- Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

# NOTION DE SYSTÈME DE FICHER

- Comment stocker les informations (données et code) sur le disque
  - Comment les **organiser** ?
  - Comment y **accéder** ?
- Différence avec la **RAM**
  - Grande quantité de données
  - Accès lent (rapport  $10^3$  à  $10^6$ )
- ➡ Définir une **norme** de gestion
  - Organisation de l'ensemble des **données** et des **périphériques**
  - **Exemple: Linux** → chaque périphérique est représenté par un fichier

# LA VUE LOGIQUE

## Système de fichiers (File System - FS)

- Un **système de fichiers** est un ensemble de **structures de données** et de **fonctions** qui permettent à un **OS** de **manipuler des fichiers**.

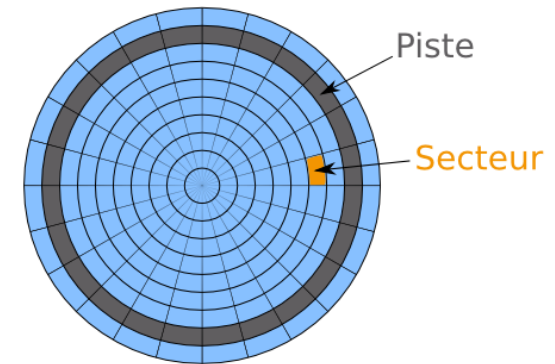
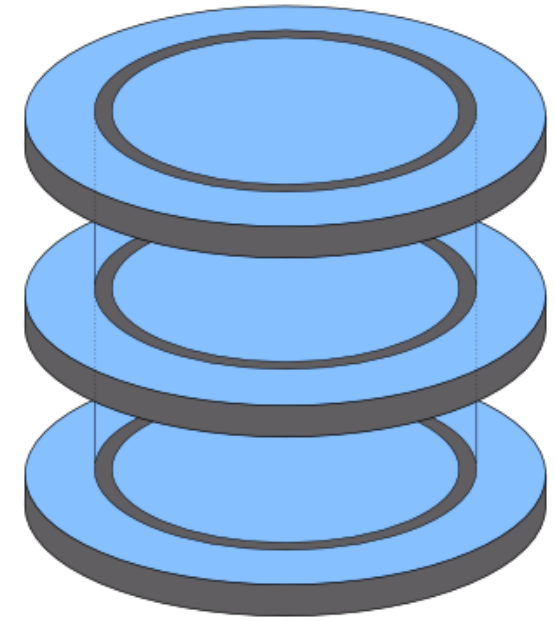
## Unité logique

- Du point de vue de l'**OS**, le **FS** doit rendre des services qui ne **dépendent pas de son implémentation** (indépendant du support physique).
- Il doit répondre à des problèmes comme :
  - la diversité des supports de stockage ;
  - la sécurisation des données.

# LA VUE PHYSIQUE

- Les supports de stockage sont **décomposés en blocs** (les éléments atomiques du **FS**).
- Selon les technologies, la **création des blocs** émule toujours celle des **disques durs**.
- **Le formatage de bas niveau :**
  1. décomposer en **secteurs** (**cylindre**, **piste** et **secteur** de la piste)
  2. regrouper les **secteurs** en **blocs**.

Le découpage en blocs génère de **la fragmentation interne** (l'espace inutilisé du dernier bloc).



# STRUCTURE D'UN SYSTÈME DE FICHIERS

- **Système logique**
  - Structure de répertoires
  - **FCB** + gestion de la protection
- **Système physique**
  - Fichiers → ensemble de blocs logiques
  - Bloc logique → blocs physique
  - Identification des blocs physiques selon support
- **Lien → pilote de périphérique**
  - **Appel système** (ex: chargement bloc 456) → instruction matériel

# MONTAGE DE RÉPERTOIRE

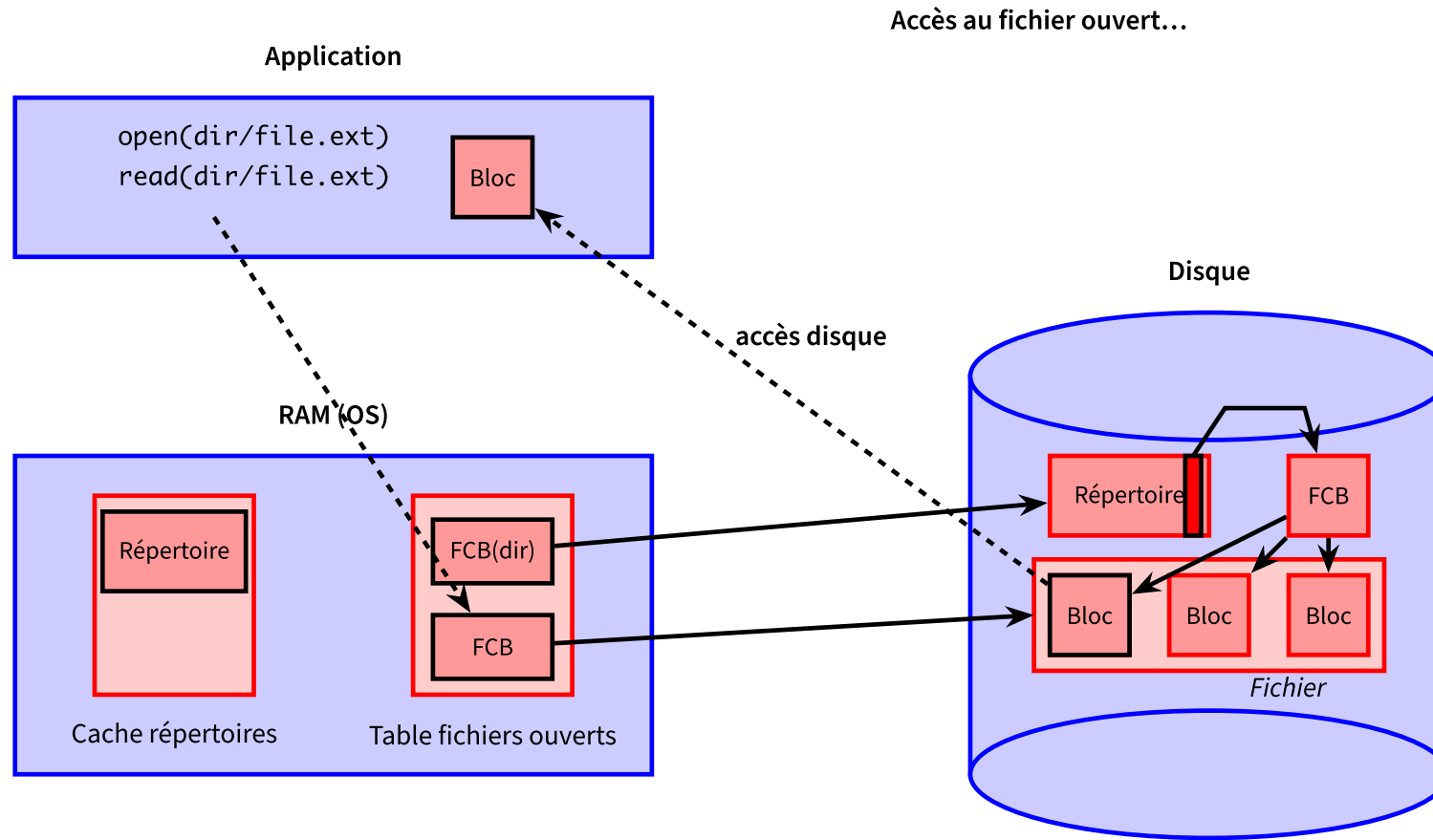
- **Le système de fichiers . . .**
  - . . . associe des noms à des blocs logiques (**fichiers**)
  - . . . associe des noms à des **répertoires** venus du disque!
- **Montage de répertoire**
  - Le montage consiste à positionner un répertoire dans le **FS**
    - Chargement du **FCB** par l'**OS** (disque → **RAM**)
    - Attribué à l'utilisateur du processus
    - Association dans le **MFD** au niveau logique
    - Les fichiers deviennent accessibles



# IMPLÉMENTATION

- **Sur le disque**
  - Bloc de démarrage (**Boot Control Block**) → chargement de l'**OS**
  - Bloc de contrôle de partition (**Master File Table**)
    - nombre de blocs, leur taille,
    - liste les blocs libres, liste les structures de descriptions de fichiers libres
  - Bloc de contrôle de répertoire ou de fichier (**FCB**)
- **Au niveau de l'OS**
  - Table des partitions/répertoires montés
  - Cache des répertoires
  - Table des fichiers ouverts (copie des **FCB**)
  - Blocs logiques

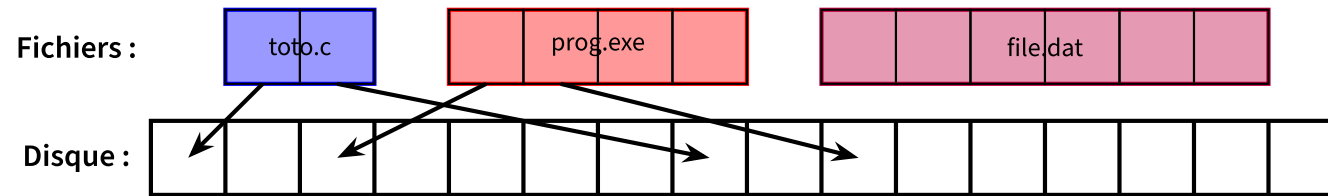
# ACCÈS À UN FICHER



# ALLOCATION

Fichiers → blocs logiques → blocs physiques

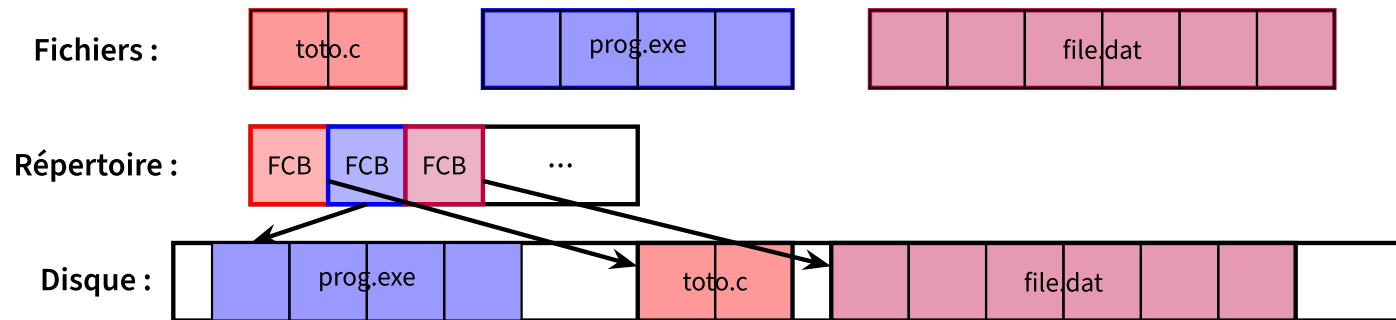
➡ Choix des blocs physiques pour 1 fichier donné



- **3 méthodes possibles**
  1. Allocation contiguë
  2. Allocation chaînée
  3. Allocation indexée

# ALLOCATION CONTIGUË

Ranger les blocs les uns derrière les autres



# ALLOCATION CONTIGUË

Ranger les blocs les uns derrière les autres

- **Avantages**

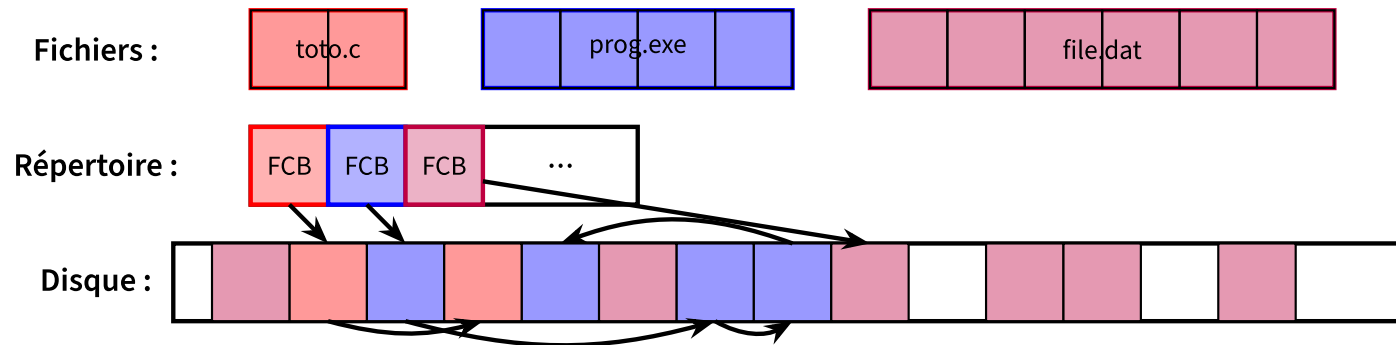
- ✓ Accès au bloc suivant: aucun coût
- ✓ **FCB**: adresse bloc départ + taille

- **Inconvénients**

- ✗ Fragmentation (compactage coûteux)
- ✗ Connaître à l'avance la taille des fichiers
- ✗ Recherche d'espace libre coûteux
- ✗ Stratégies d'allocation (**BestFit**, **FirstFit**, **WorstFit**) à définir

# ALLOCATION CHAÎNÉE

Fichier = liste chaînée de blocs



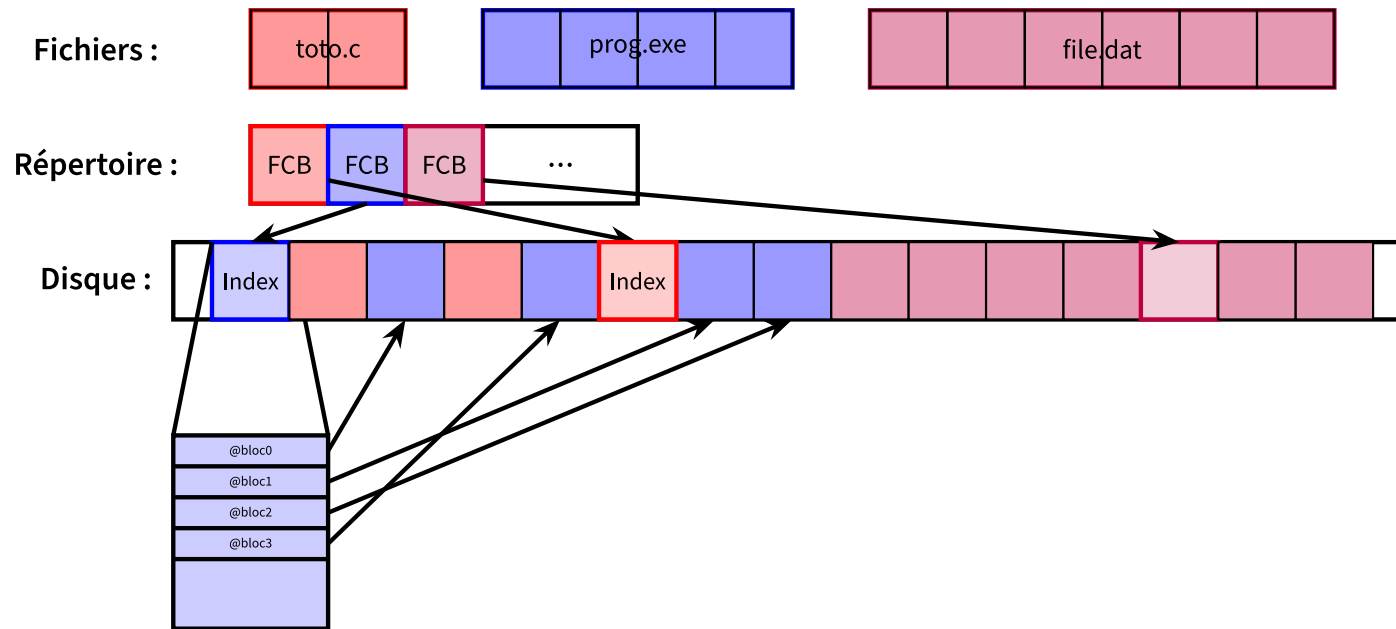
# ALLOCATION CHAÎNÉE

Fichier = liste chaînée de blocs

- **Avantages**
  - ✓ **FCB**: adresse premier et dernier blocs
  - ✓ Pas de fragmentation
  - ✓ Fichiers taille quelconque
- **Inconvénients**
  - ✗ Accès séquentiel:  $N^e$  bloc  $\rightarrow$  N accès disques!
  - ✗ Fiabilité: 1 bloc endommagé  $\rightarrow$  tout le fichier est perdu
  - ✗ Espace utilisé par les pointeurs (relativement négligeable)

# ALLOCATION INDEXÉE

Rassembler tous les pointeurs dans un bloc d'index  
(1 bloc par fichier)





# ALLOCATION INDEXÉE

Rassembler tous les pointeurs dans un bloc d'index  
(1 bloc par fichier)

- **Avantages**

- ✓ Pas de fragmentation
- ✓ Accès direct (2 accès disque)

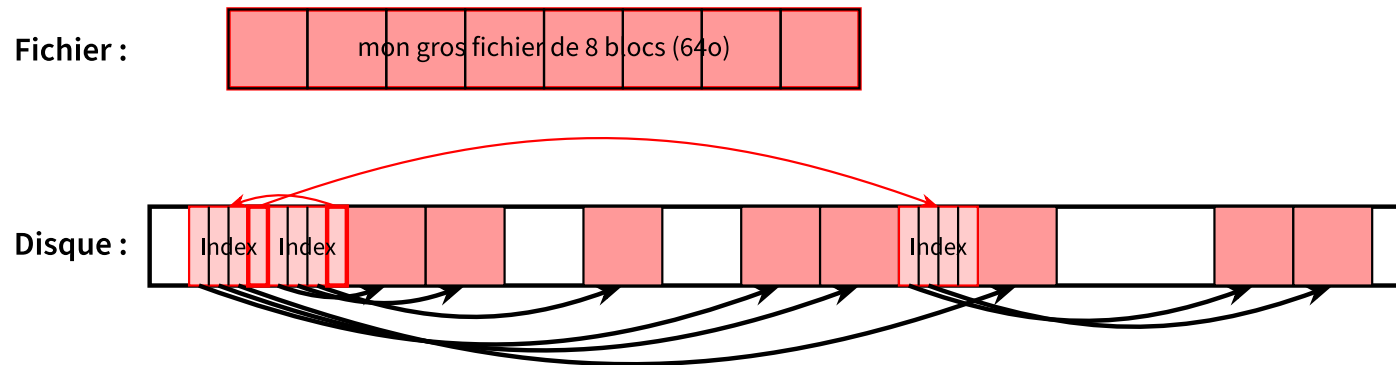
- **Inconvénients**

- ✗ 1 bloc perdu par fichier
- ✗ Taille fichier limitée par taille bloc

$64 \text{ bits} \times 64 \text{ blocs} = 5120 \Rightarrow \text{max} = 64 \times 5120 = 32 \text{ Ko}$

# LISTE CHAÎNÉE DE BLOCS D'INDEX

- Le répertoire pointe vers un bloc d'index
- Le bloc d'index se termine par un pointeur vers un autre bloc d'index

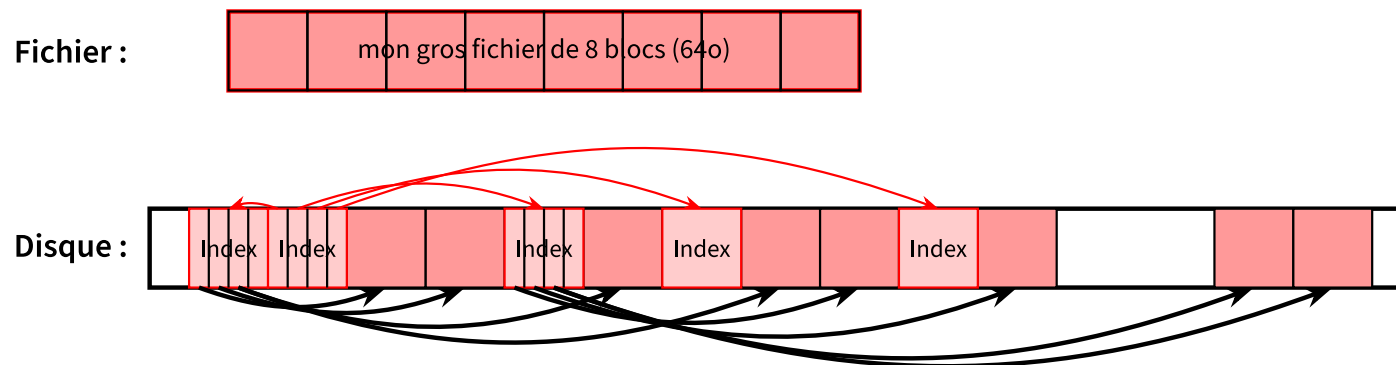


# LISTE CHAÎNÉE DE BLOCS D'INDEX

- Le répertoire pointe vers un bloc d'index
- Le bloc d'index se termine par un pointeur vers un autre bloc d'index
- **Avantages**
  - ✓ Pas de fragmentation
  - ✓ Taille de fichier quelconque
- **Inconvénients**
  - ✗  $N$  blocs perdus par fichier
  - ✗ Accès indirect ( $\geq 2$  accès disque) mais plus rapide que la liste chaînée

# INDEXATION À PLUSIEURS NIVEAUX

- Le répertoire pointe vers un bloc d'index "maître"
- Le bloc maître pointe vers des blocs d'index
  - ➡  $n^2$  blocs indexables au lieu de  $n$
- Éventuellement, indexation sur 3 niveaux ( $n^3$ )



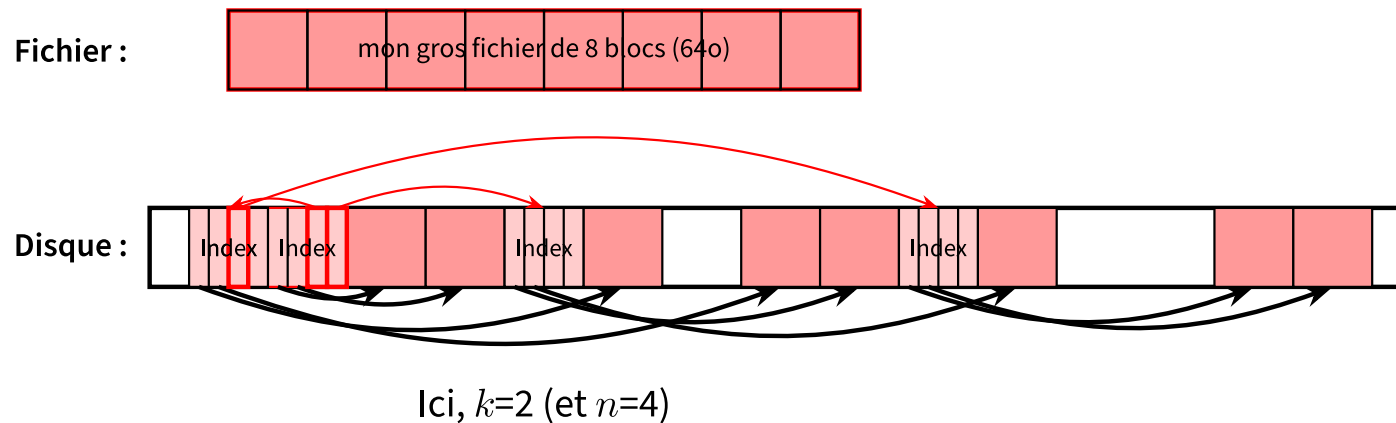
# INDEXATION À PLUSIEURS NIVEAUX

- Le répertoire pointe vers un bloc d'index "maître"
- Le bloc maître pointe vers des blocs d'index
  - ➡  $n^2$  blocs indexables au lieu de  $n$
- Éventuellement, indexation sur 3 niveaux ( $n^3$ )
- **Avantages**
  - ✓ Pas de fragmentation
  - ✓ Taille de fichier quelconque
  - ✓ Accès direct (3 accès disque max + possibilité index en cache)
- **Inconvénients**
  - ✗  $\geq 2$  blocs perdus par fichier  
(même si on ne crée pas les index en trop)

# SCHÉMA COMBINÉ

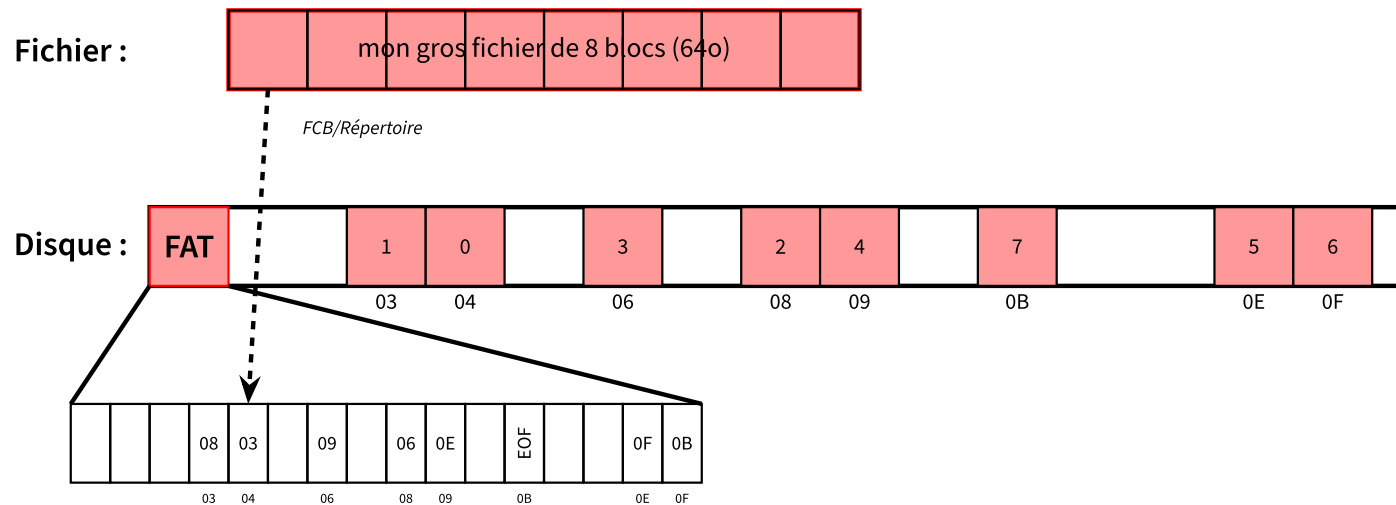
## EX: LINUX EXTFS

- Combiner allocation chaînée et allocation indexée
- Index =  $k$  premiers blocs du fichier +  $(n - k)$  blocs d'indirection
- ✓ Moins de perte pour les petits fichiers
- ✓ Accès rapide



# FILE ALLOCATION TABLE (FAT)

- Utilisé sous **MSDOS** (**Intel**) et **OS/2** (**IBM**)
- Allocation indexée
- Liste chaînée des **index** des blocs en **début de chaque partition**



# FILE ALLOCATION TABLE (FAT)

- **Avantages**

- ✓ **FCB**: adresse premier bloc = premier index
- ✓ Pas de fragmentation (allocation indexée)
- ✓ Allocation bloc simple
- ✓ Accès rapide (**FAT** chargée en cache puis accès direct disque)

- **Inconvénients**

- ✗ Fiabilité: **FAT** perdue → disque foutu!  
→ doubler la **FAT** (sur 2 blocs distincts)



# PLAN

- > La notion de fichier
- > Les répertoires
- > Les systèmes de fichiers
- > Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

# CE QU'IL FAUT RETENIR

- Les supports physiques sont partitionnés (découpés) en sous ensembles logiques sur lesquels un File System - FS est installé
- L'OS rend disponible les divers FS déclarés dans la table de montage (on parle aussi de Volumes)
- L'OS présente une vue unifiée des différents Volumes (FS) disponibles (Virtual File System - VFS)

# CE QU'IL FAUT RETENIR

- Fichier = point d'accès au système
- File Control Block
- Répertoire
- Ouverture de fichier
- Structure d'un système de fichiers
- Blocs logiques/physiques
- Allocations contiguë, chaînée, indexée, FAT

# MERCI

[Version PDF des slides](#)

[Retour à l'accueil](#) - [Retour au plan](#)