Systèmes d'Exploitation
Cours 8 : Systèmes de fichiers

Thomas Lavergne

Université Paris-Saclay

Licence 3 - semestre S5

Plan

Plan

Système de fichiers

Partie visible de l'OS

- Mécanisme de stockage sur un support physique
- Accès aux données stockées
- Accès aux programmes
- Accès à l'ensemble du système informatique

Système de fichiers

Partie visible de l'OS

- Mécanisme de stockage sur un support physique
- Accès aux données stockées
- Accès aux programmes
- Accès à l'ensemble du système informatique

Définition

Un fichier est une collection nommée d'information accessibles via un périphérique.

Unité *logique*

- Indépendante du support physique (périphérique)
- Abstraction des propriétés physiques

Structure d'un fichier

Type de fichier

- Code source
- Données
- Bibliothèque
- Fichier exécutable
- Point d'accès à un périphérique
- *etc*

$\rightarrow$ À chaque type de fichier correspond une structure spécifique !

Généralement indiqué par son extension.

Structure d'un fichier : exemples I

Fichiers texte : `.txt`

Données textuelles à l'usage de l'utilisateur humain

→ Succession de caractères, organisés en lignes séparées par un caractère spécial ($\backslash n$, $\backslash r$, ... selon l'OS)

Fichiers source : `.c`, `.java`, ...

Fourni par un utilisateur humain pour être traité par la machine

→ succession de fonctions et sous-programmes, composés d'instructions séparées par des symboles spécifiques

Structure d'un fichier : exemples II

Bibliothèque : `.o`, `.dll`, ...

Construits par un compilateur à partir d'un fichier source

→ Succession d'octets organisés en blocs interprétables par l'éditeur de lien

Exécutable : `.exe`, ...

→ Succession d'instructions que l'OS peut charger en mémoire pour exécuter un programme

File Control Block

Définition

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

File Control Block

Définition

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

Contenu

Nom : `stdio` : indépendant de l'OS, lisible

Identifiant : numérique, unique, pour l'OS

Emplacement : pointeur sur un périphérique

Taille : en octets ou en blocs

Protection : lecture, écriture, exécution...

Date(s) : création, modification, accès...

Utilisateur : propriétaire du fichier

Partage de fichiers

Définition

Rendre accessible à un utilisateur B un fichier de l'utilisateur A

Partage de fichiers

Définition

Rendre accessible à un utilisateur B un fichier de l'utilisateur A

Exemple : lecture de /bin/sh, accès à /dev/mouse0, ...

Partage de fichiers

Définition

Rendre accessible à un utilisateur B un fichier de l'utilisateur A

Exemple : lecture de /bin/sh, accès à /dev/mouse0, ...

Politique de protection

Définir qui peut accéder à quel(s) fichier(s)

- Identifiant utilisateur $\rightarrow$ identifiant processus
- Contrôle d'accès dans le FCB

Protection de fichiers I

Liste de contrôle d'accès (ACL)

Utilisateur $\rightarrow$ droits

Problèmes :

- ✗ 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 $\times$ type d'accès (lecture, écriture, ...)

- ✗ Impraticable

Protection de fichiers II

Classes d'utilisateurs

Exemple : Propriétaires vs Autres

→ quelques bits par fichier

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=[dev-ram,user-disque]`
  - ➜ `robin` n'a pas accès à `toto.c`

Protection de fichiers III

Exemple : Unix

Classes + groupes

- 3 classes : utilisateur, groupe, autres
- N groupes
- 3 droits : read, write, execute

➜ 3×3 bits par fichier

Exemple :

```
$ ls -l
total 248
drwx------ 6 nico prof     4096 nov.  20 12:06 private
-rw------- 1 nico prof     2356 déc.   5 15:30 notes.ods
drwxrwx--- 8 nico prof     4096 déc.   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
```

Notion de répertoire

Définition

Le répertoire est la structure de stockage des informations des fichiers (les FCB) sur le périphérique (le disque).

Notion de répertoire

Définition

Le répertoire est la structure de stockage des informations des fichiers (les FCB) sur le périphérique (le disque).

Structure

- Contenu du répertoire = FCB des fichiers

Notion de répertoire

Définition

Le répertoire est la structure de stockage des informations des fichiers (les FCB) sur le périphérique (le disque).

Structure

- Contenu du répertoire = FCB des fichiers

Principe

L'OS récupère l'information sur les fichiers dans le répertoire

Opérations sur un fichier

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

$\rightarrow$ effectuées à partir des appels systèmes de base

Plan

Ouverture de fichier

Problème

- Nécessité d'accéder au FCB à chaque opération sur le fichier
- Le FCB est stocké dans le répertoire du périphérique

$\rightarrow$ Très coûteux en accès disque (donc en temps) !

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) !

Définition

L'appel système `open` permet de charger le FCB en mémoire

Accès à un fichier

L'OS impose que tout accès à un fichier soit précédé d'une ouverture.

Table des fichiers ouverts

Stockage des FCB en RAM

La table des fichiers ouverts de l'OS contient l'ensemble des FCB des fichiers ouverts.

- Ouverture $\rightarrow$ chargement du FCB depuis le répertoire + ajout dans la table
- Fermeture $\rightarrow$ retrait de la table
- Pas d'impact sur le fichier !

Table des fichiers ouverts

Stockage des FCB en RAM

La table des fichiers ouverts de l'OS contient l'ensemble des FCB des fichiers ouverts.

- Ouverture → chargement du FCB depuis le répertoire + ajout dans la table
- Fermeture → retrait de la table
- Pas d'impact sur le fichier !

Gestion par l'OS

- Une table de fichiers ouverts globale avec compteurs + une table par processus → fermeture à la terminaison

Plan

Structure des disques

Disque

Structure physique

Partition

Structure logique (disque « virtuel »)

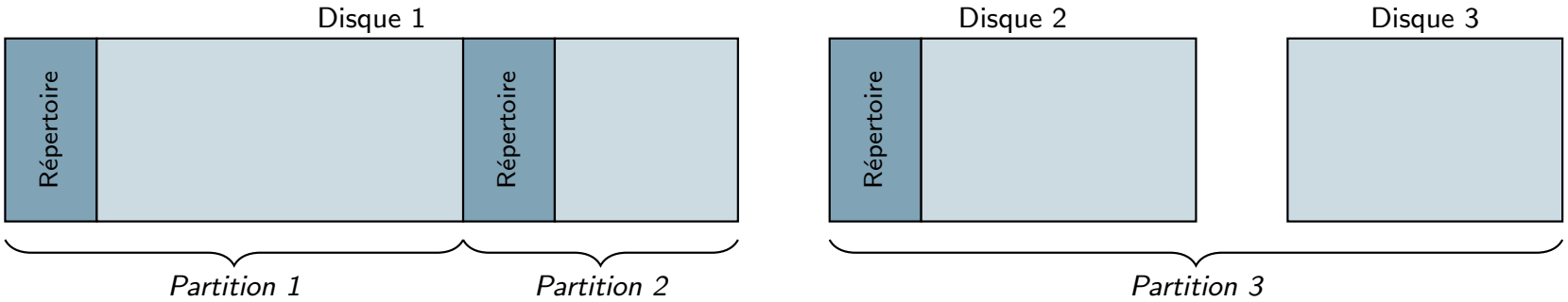
- Base : 1 disque = 1 partition
- 1 disque = N partitions
- 1 partition = 1 ou N disques (selon OS)

Répertoire

Un répertoire par partition : l'ensemble des FCB

➜ Nom/identifiant → FCB

Répertoire : structure de base



Structure à 1 niveau

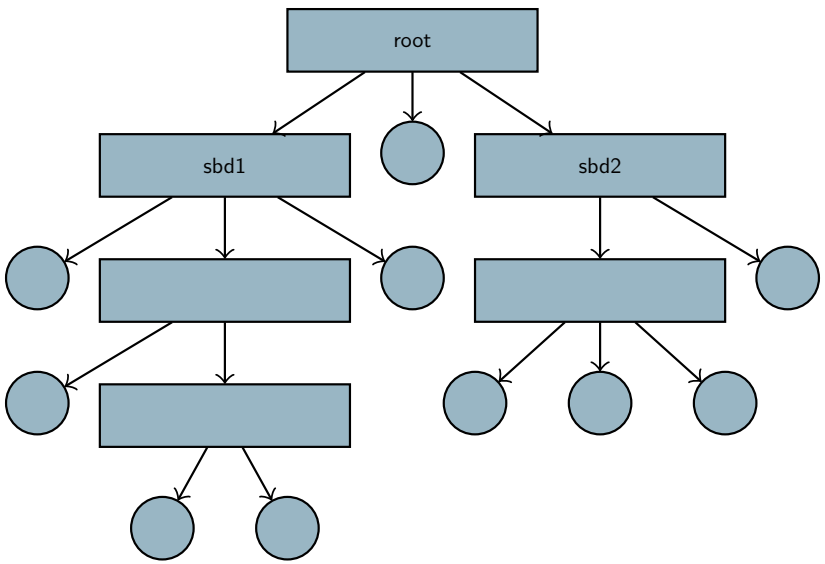
- ✓ Nom → FCB
- ✗ Taille du répertoire proportionnelle au nombre de fichiers
  → borner le nombre de fichiers...
- ✗ Utilisateur : organiser les fichiers, unicité de nom, ...

Structure arborescente

Principe

Généralisation de la structure à 2 niveaux :

- Répertoire racine (MFD)
- Sous-répertoires, pouvant à leur tour jouer le rôle de MFD



Structure arborescente : implémentation

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)

Appels systèmes

- Création/Suppression de répertoire
- Changement de répertoire courant

Structure en graphe

Principe

Généralisation de l'arbre avec des liens

➜ Graphe acyclique

Structure en graphe

Principe

Généralisation de l'arbre avec des liens

➜ Graphe acyclique

Liens

Référencer un fichier décrit dans un autre répertoire

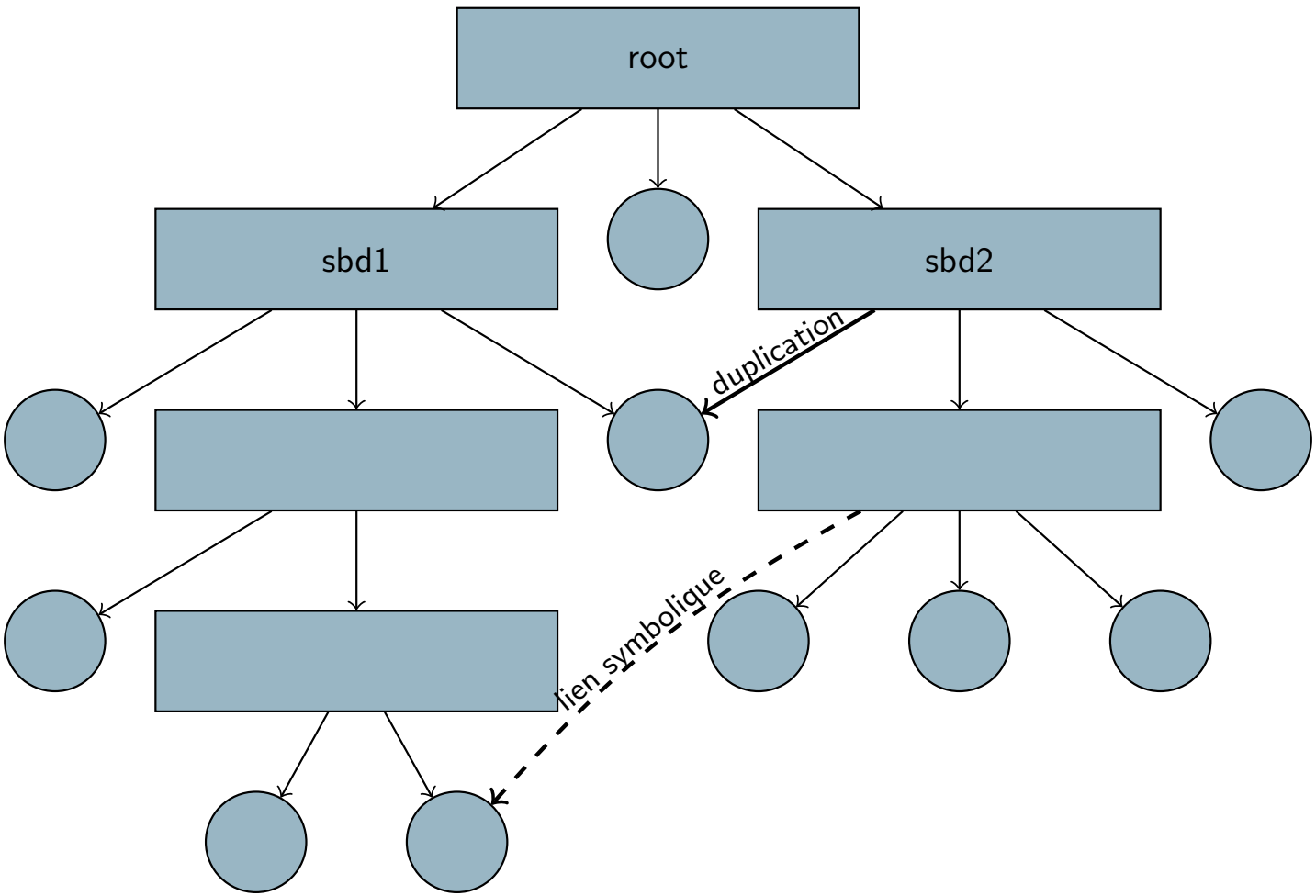
➜ bit « lien » dans le répertoire + chemin absolu

Extension : duplication

➜ FCB recopié → copie et original indiscernables

➜ Compteur de liens (pour savoir quand libérer l'espace sur le support physique)

Structure en graphe : exemple



Plan

Notion de système de fichier

Définition

Comment stocker les informations (données & code) sur le disque

- Comment les organiser
- Comment y accéder

Définit une norme de gestion (Ex : FAT, NTFS, EXT4FS, NFS...)

Notion de système de fichier

Définition

Comment stocker les informations (données & code) sur le disque

- Comment les organiser
- Comment y accéder

Définit une norme de gestion (Ex : FAT, NTFS, EXT4FS, NFS...)

Plus généralement

Organisation de l'ensemble des données et des périphériques gérés par l'OS

Exemple : Linux
→ chaque périphérique est représenté par un fichier

Notion de système de fichier

Différence avec la RAM

- Grande quantité de données
- Accès lent (rapport 10^3 à 10^6)

Notion de bloc

- Unité de base du support physique
  - ➜ Toutes les données à stocker sont découpées en bloc
- Taille fixe (de 32o à 4Mo selon support)

  Exemples : disques dur années 2000 = 512o, SSD actuel = 4Mo

Système de fichiers (*FileSystem*)

Organisation des fichiers en blocs (*cf. mémoire paginée*)

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

  cf. mémoire paginée

- Identification des blocs physiques selon support

Lien : pilote de périphérique

Appel système (ex : chargement bloc 456) → instruction matériel

Structure de contrôle

Sur le disque

- Bloc de démarrage → chargement de l'OS depuis une partition
- Bloc de contrôle de partition : *Master File Table*
- Bloc de contrôle de répertoire ou de fichier (FCB)

Structure de contrôle

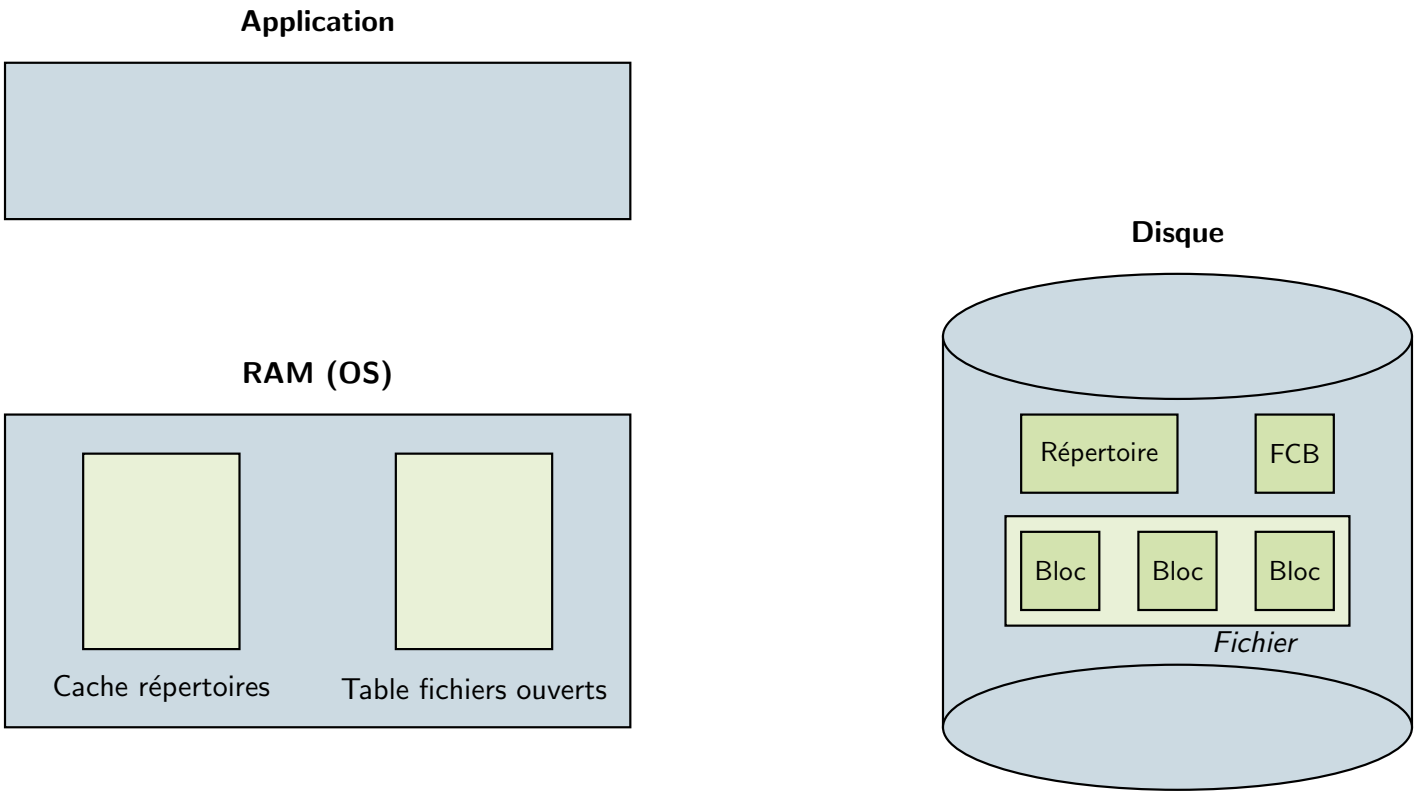
Sur le disque

- Bloc de démarrage → chargement de l'OS depuis une partition
- Bloc de contrôle de partition : *Master File Table*
- Bloc de contrôle de répertoire ou de fichier (FCB)

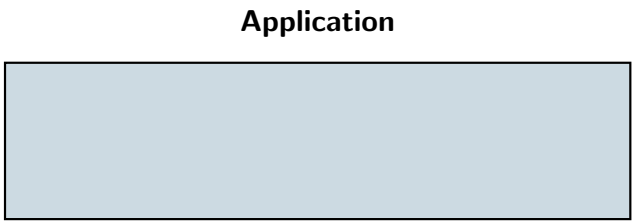
Au niveau de l'OS

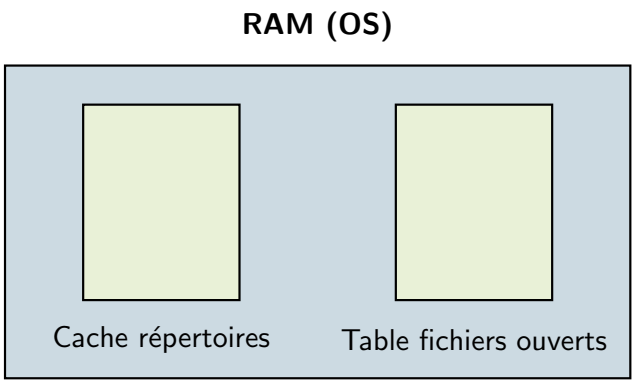
- Table des partitions/répertoires montés
- Cache des répertoires
- Table des fichiers ouverts (copie des FCB)
- Blocs logiques

Accès à un fichier

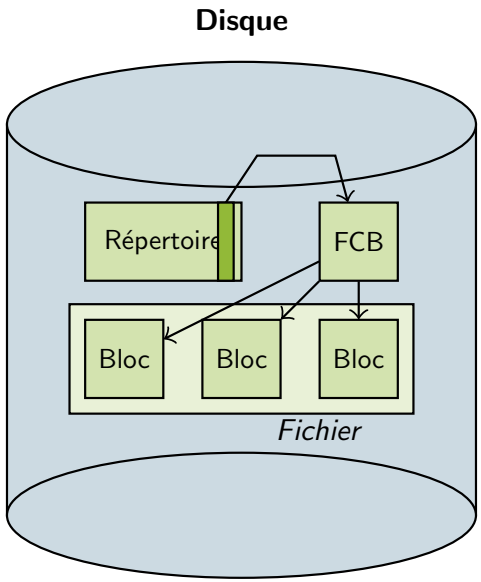


Accès à un fichier



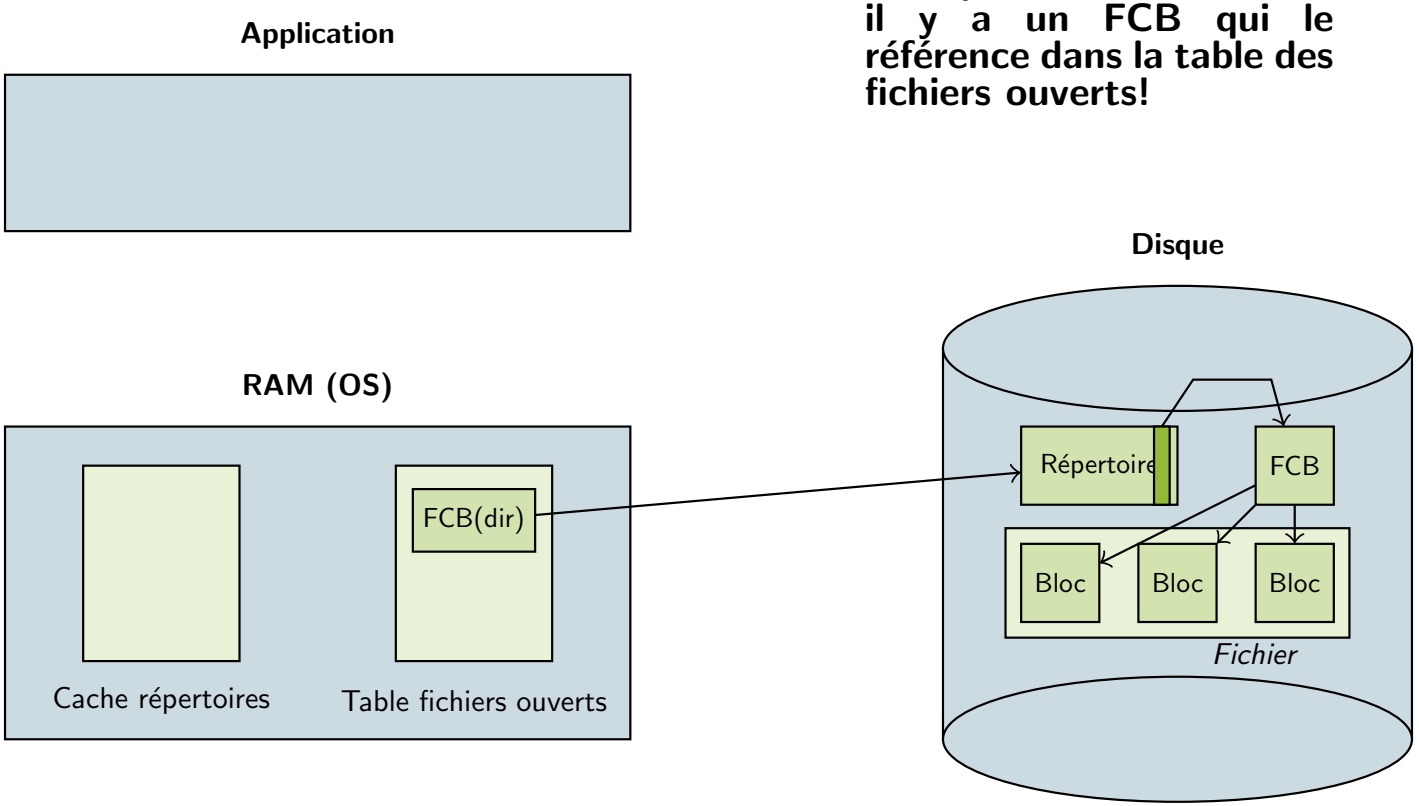


Le système de fichier (sur le disque) définit la structuration des données

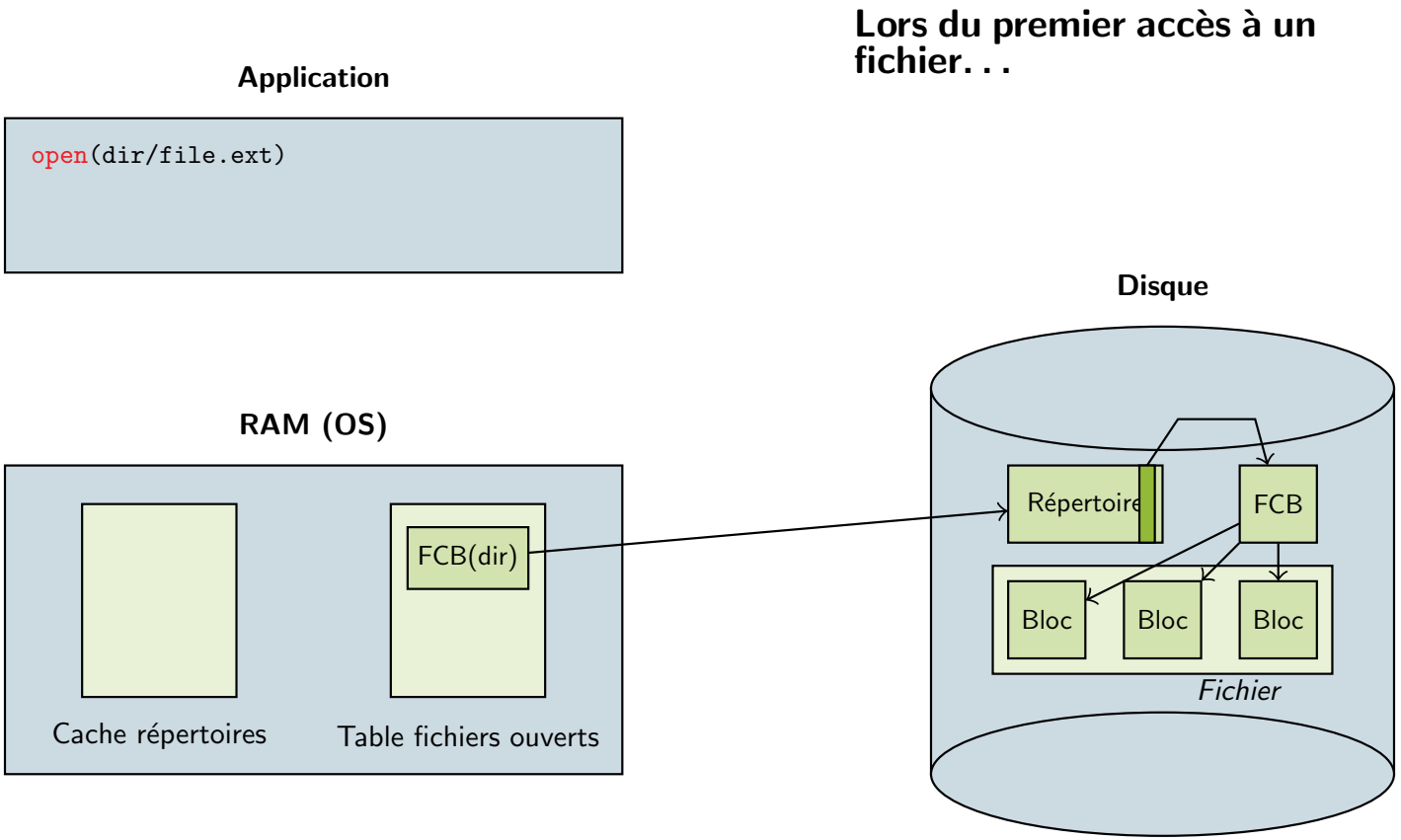


Accès à un fichier

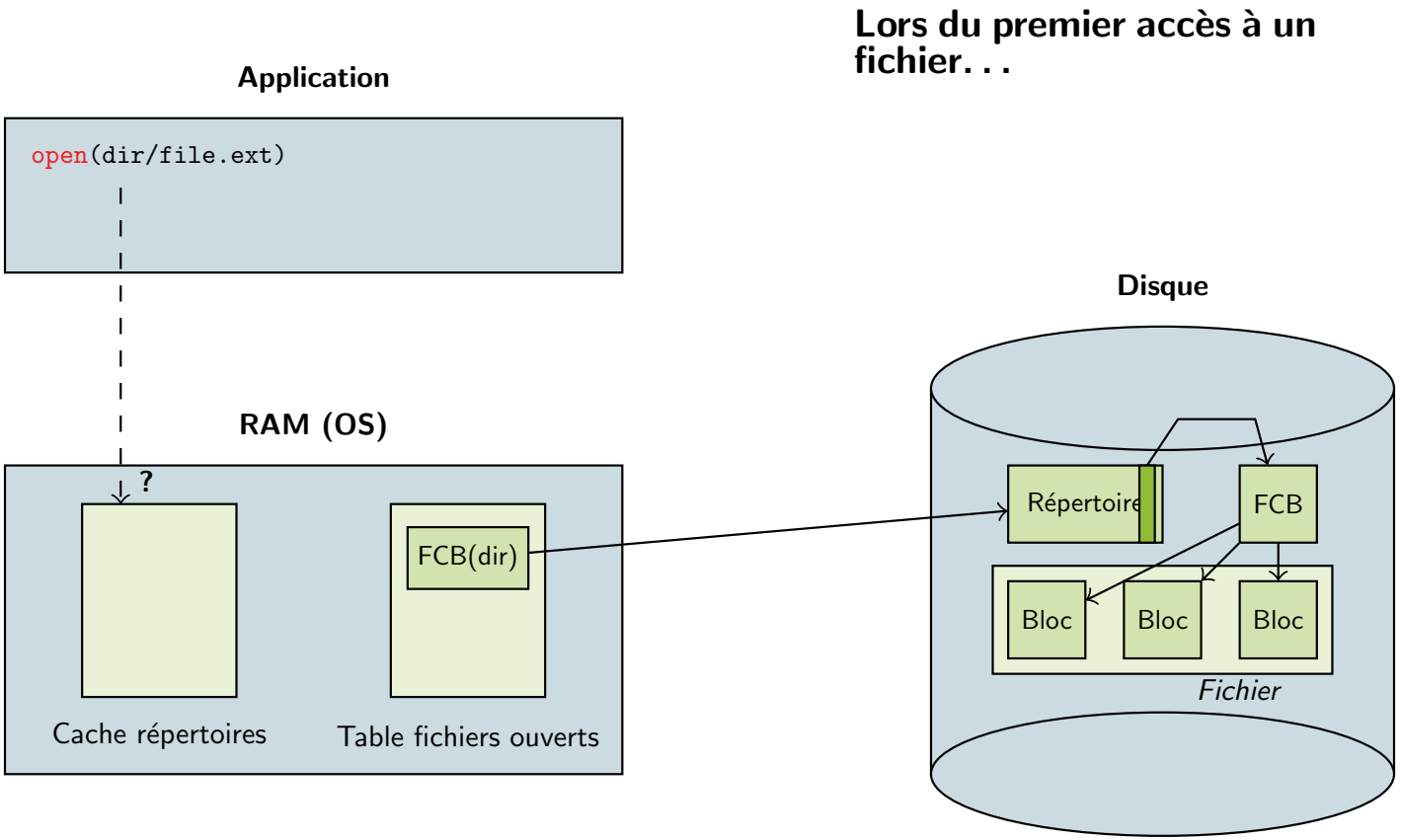
Le répertoire est monté : il y a un FCB qui le référence dans la table des fichiers ouverts!



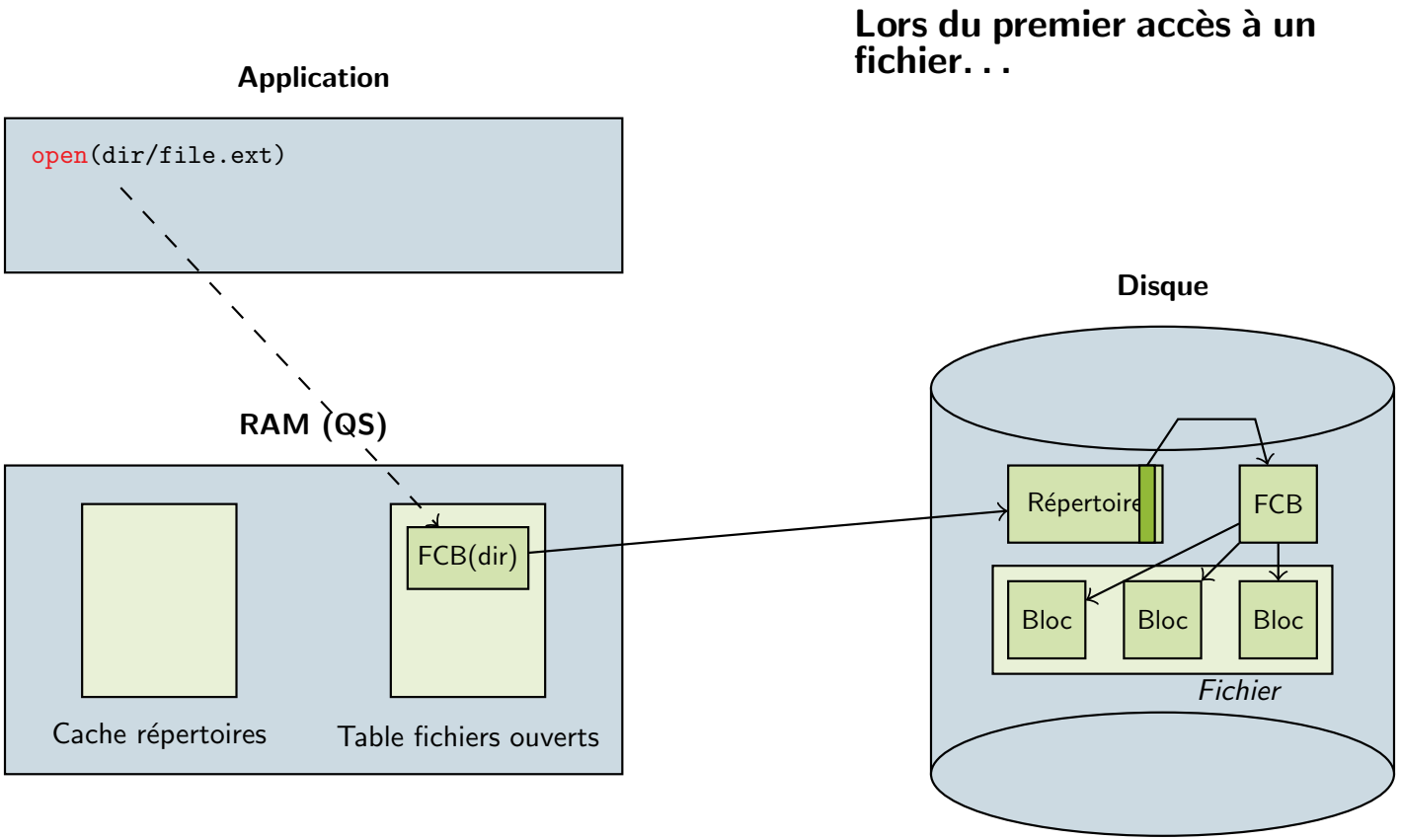
Accès à un fichier



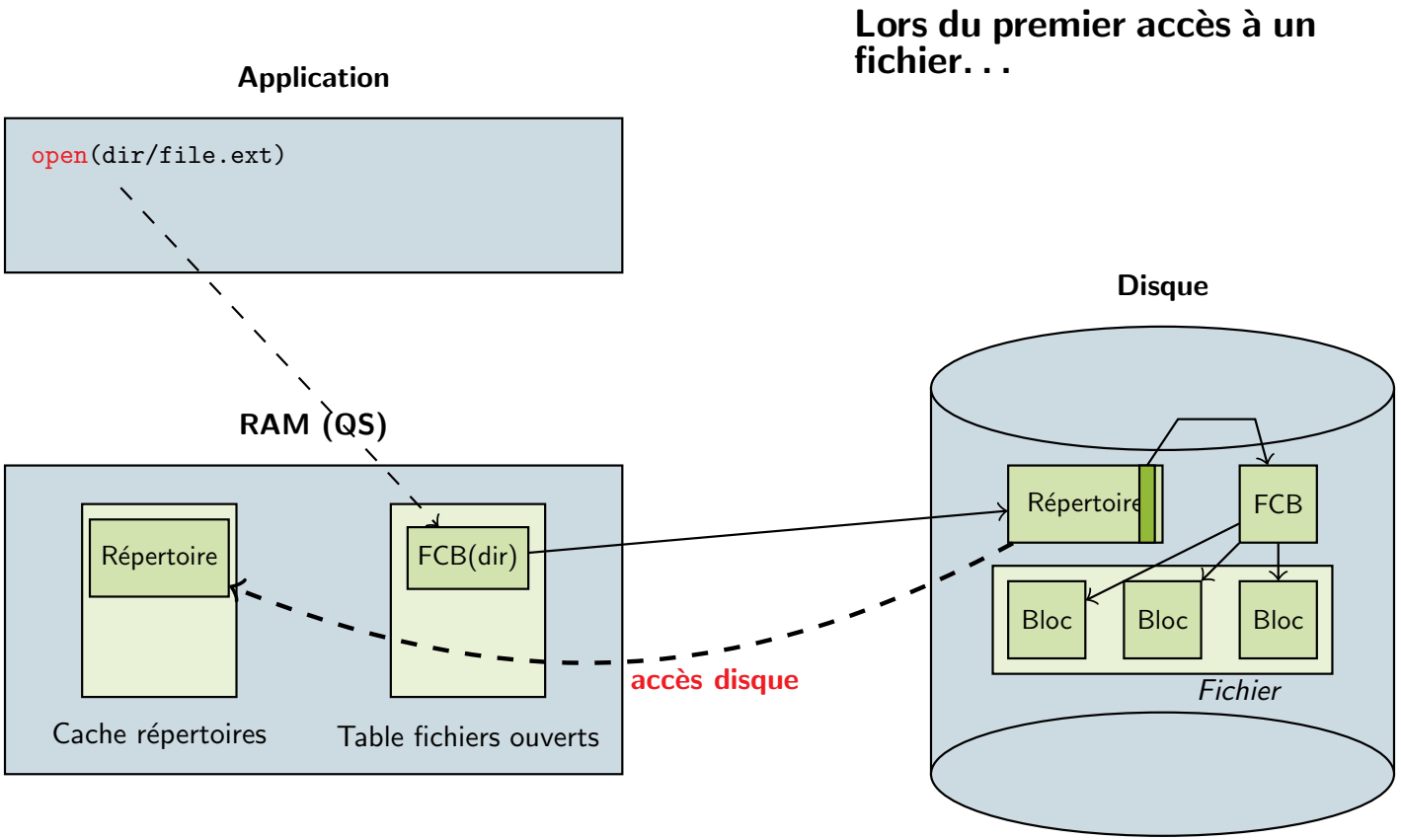
Accès à un fichier



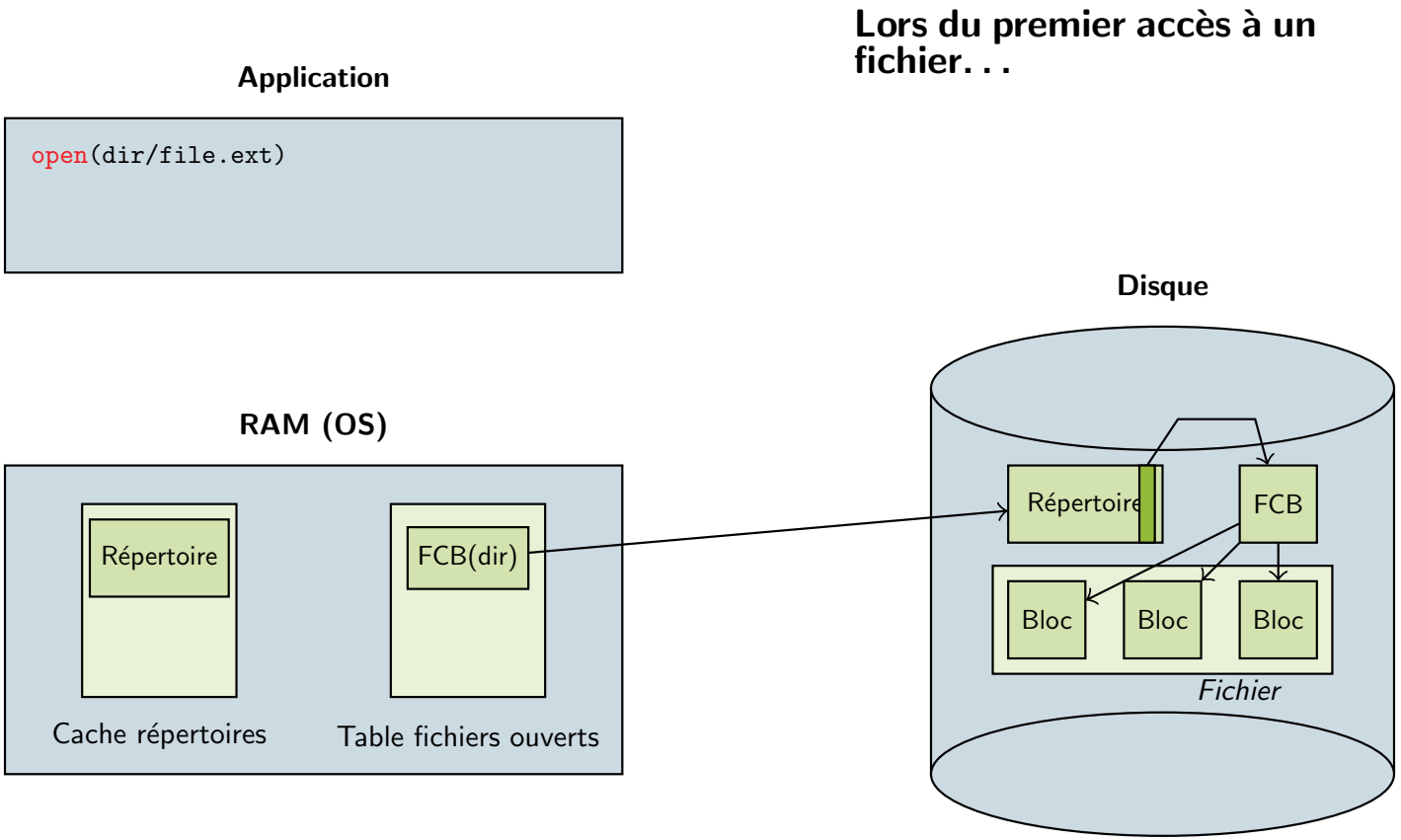
Accès à un fichier



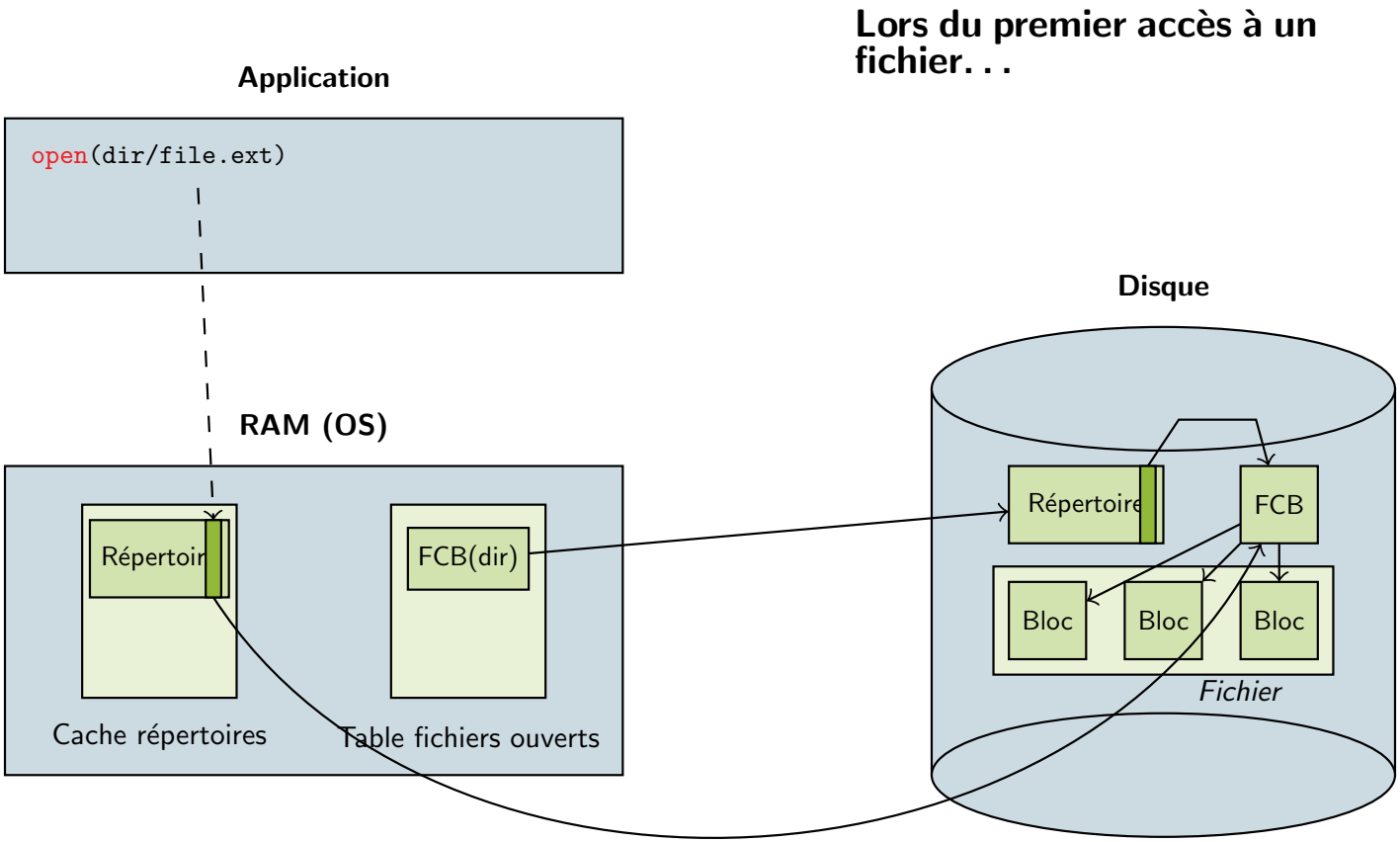
Accès à un fichier



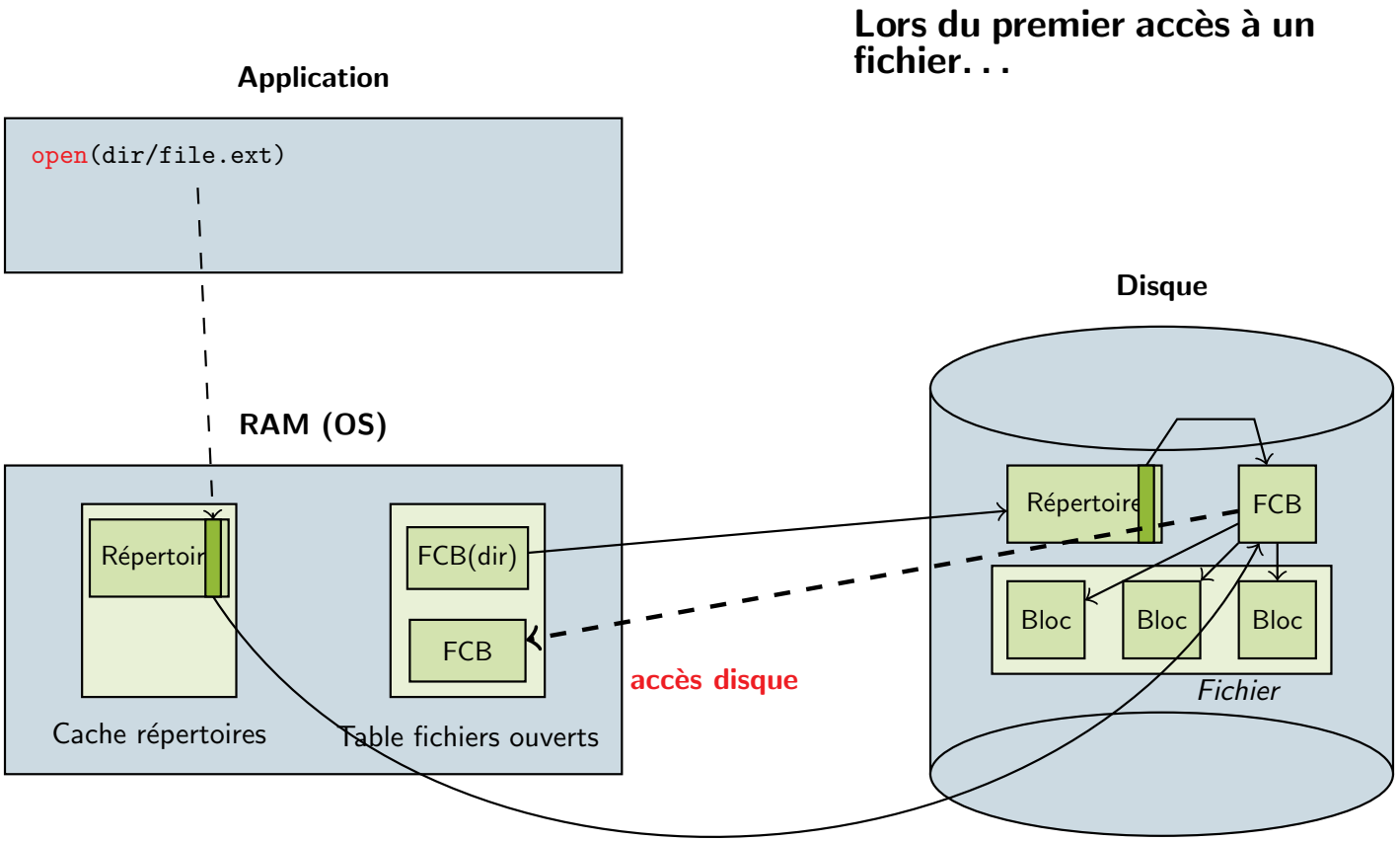
Accès à un fichier



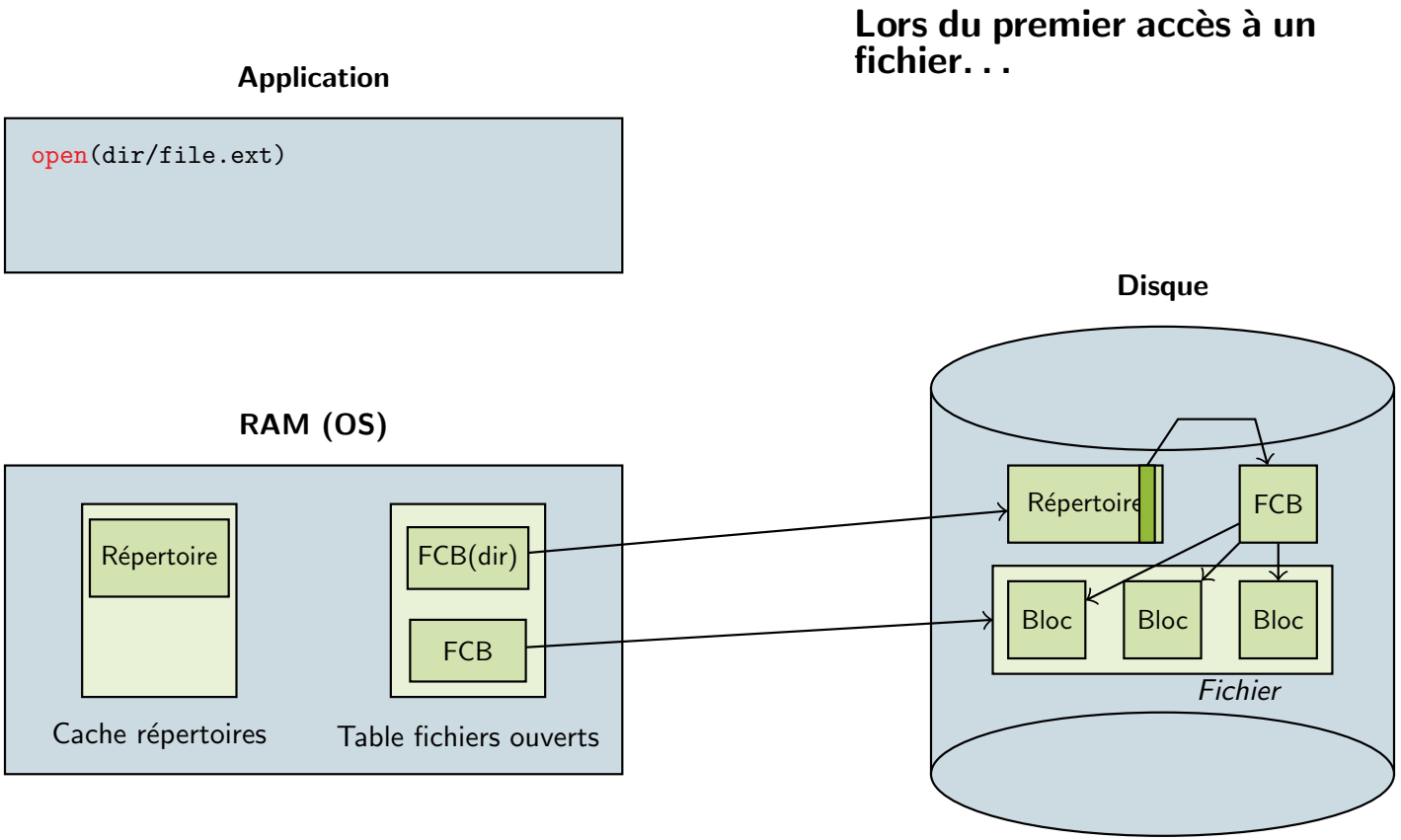
Accès à un fichier



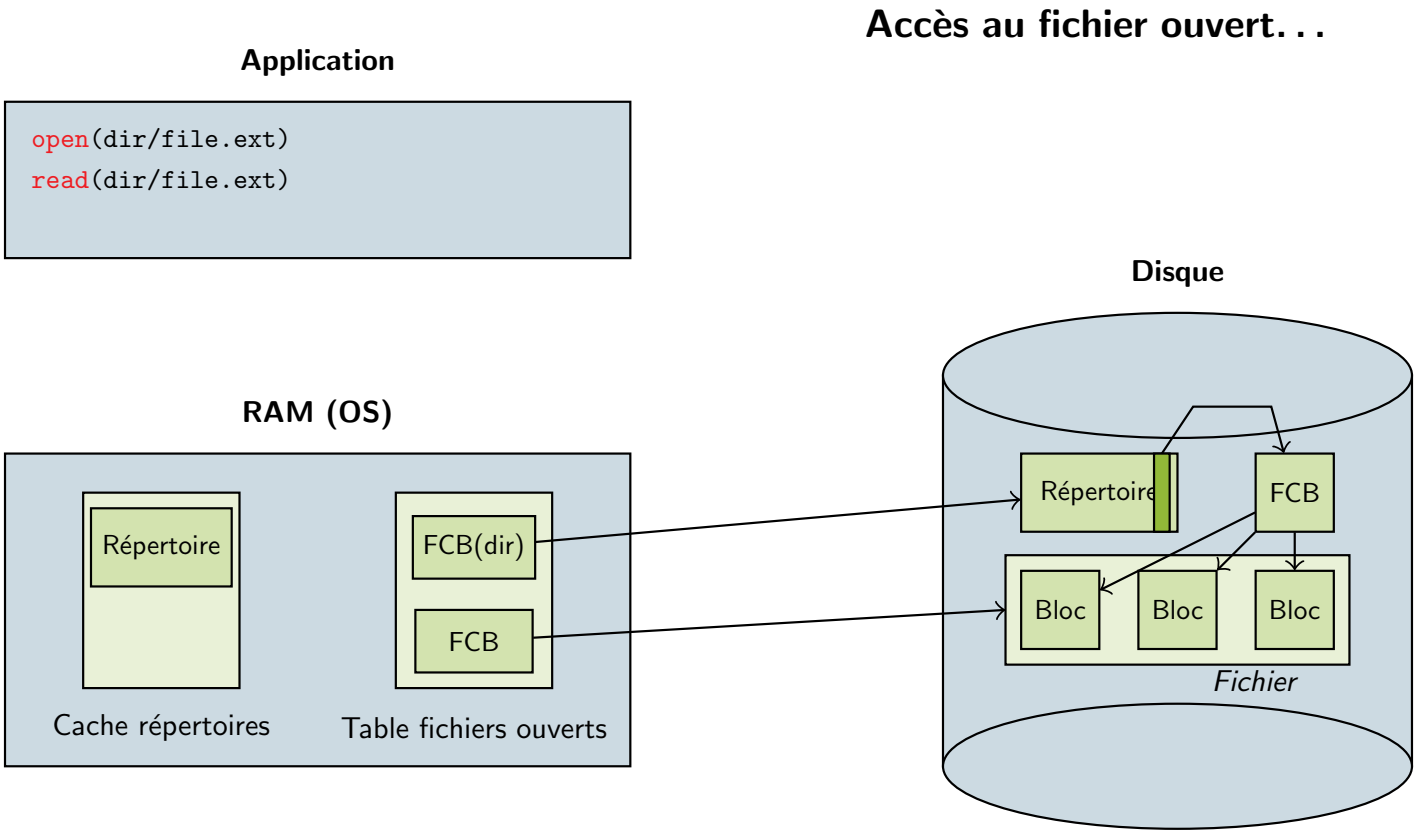
Accès à un fichier



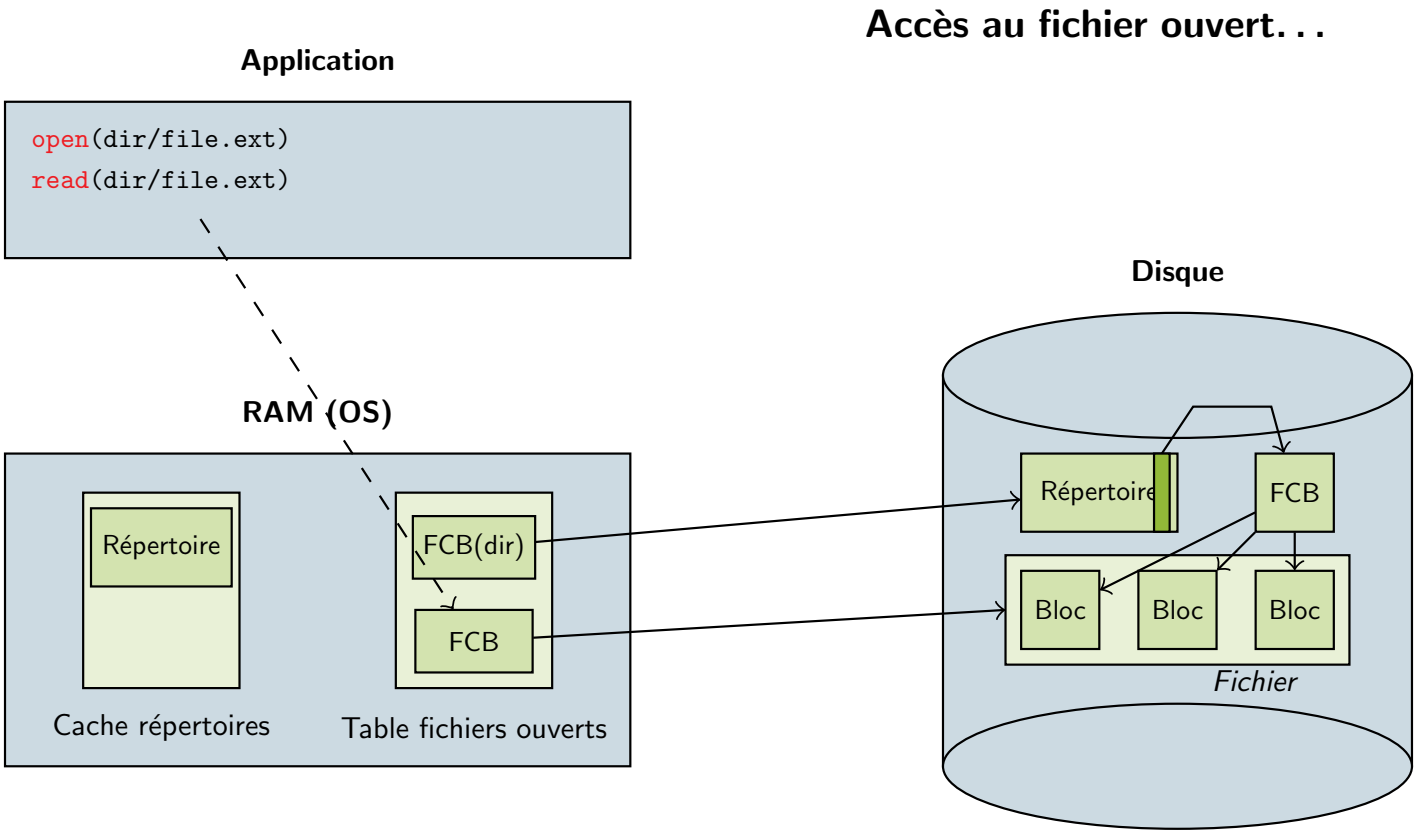
Accès à un fichier



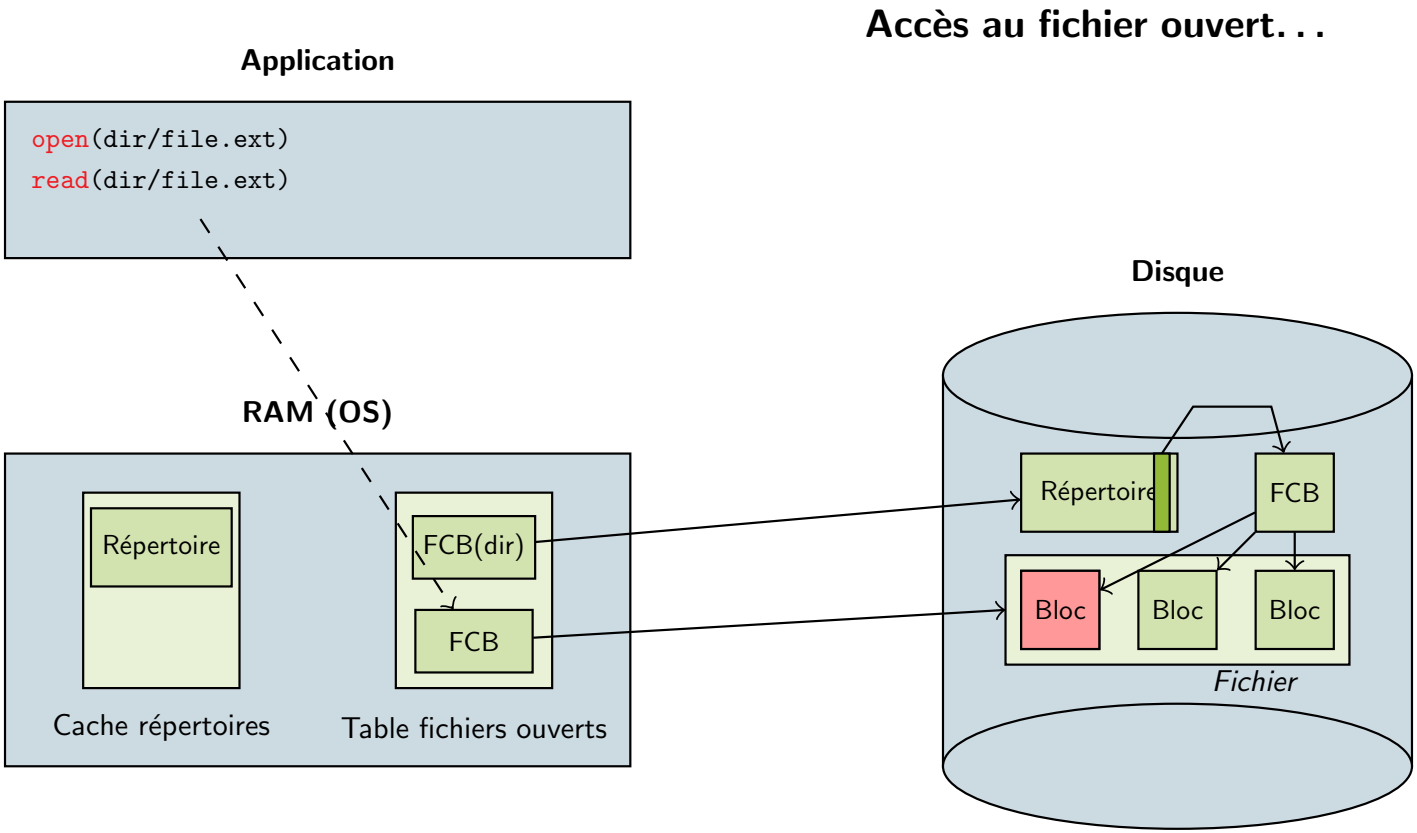
Accès à un fichier



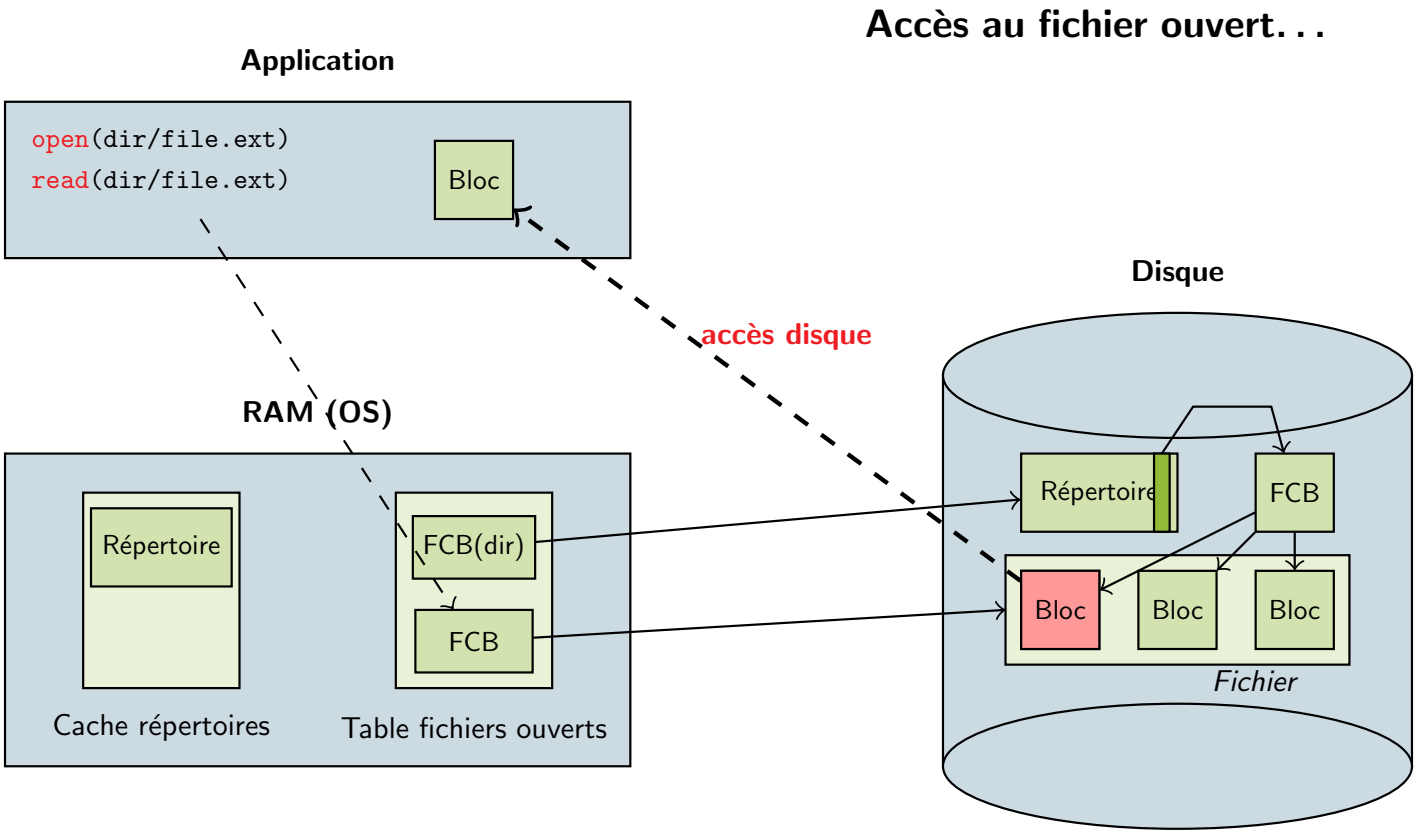
Accès à un fichier



Accès à un fichier



Accès à un fichier

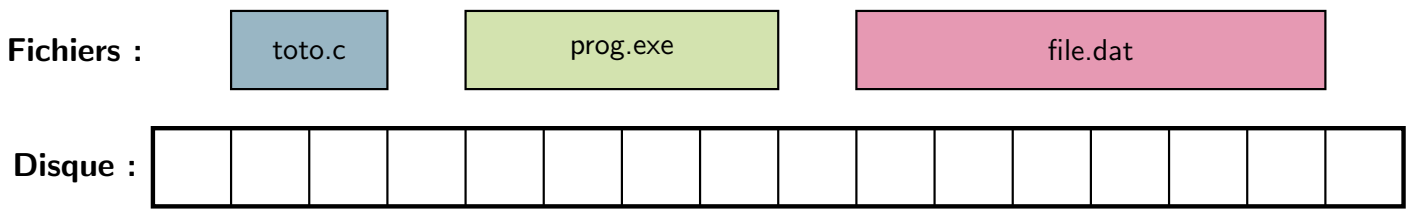


Allocation

Problème

Fichiers → blocs logiques → blocs physiques

➜ Choix des blocs physiques pour 1 fichier donné

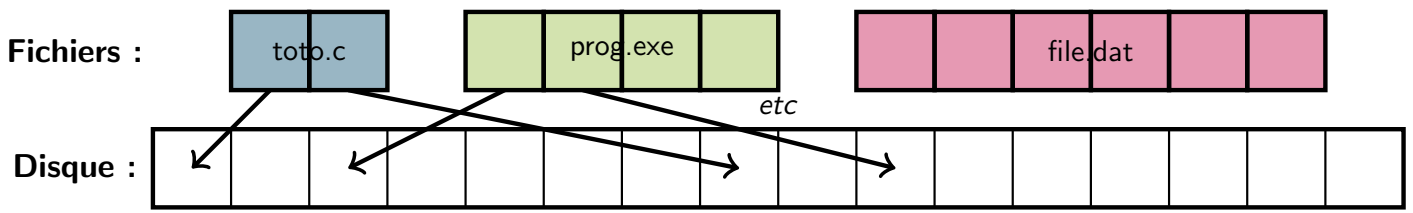


Allocation

Problème

Fichiers → blocs logiques → blocs physiques

➜ Choix des blocs physiques pour 1 fichier donné

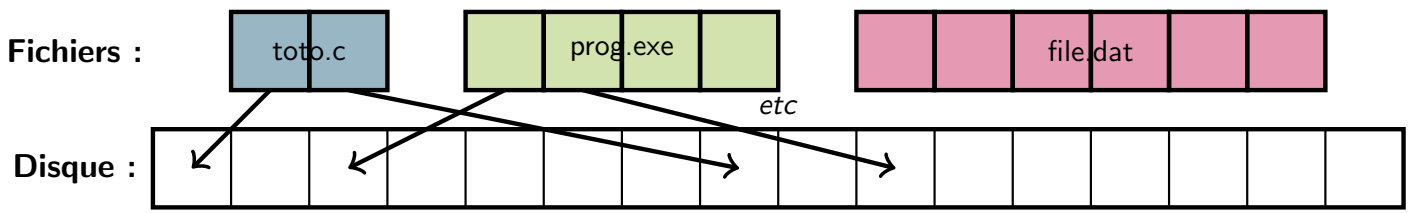


Allocation

Problème

Fichiers → blocs logiques → blocs physiques

➜ Choix des blocs physiques pour 1 fichier donné



3 méthodes possibles

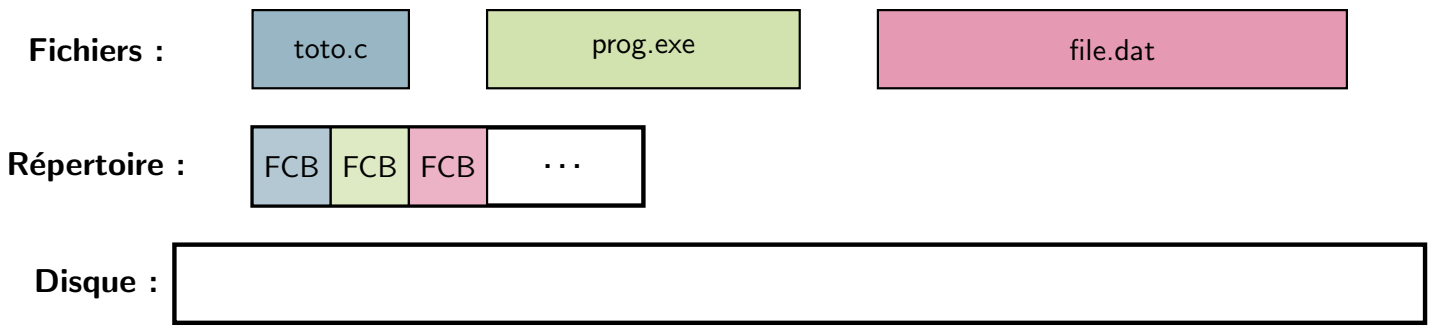
- Allocation contiguë
- Allocation chaînée
- Allocation indexée

Chaque méthode a ses avantages et inconvénients !

Allocation contiguë

Principe

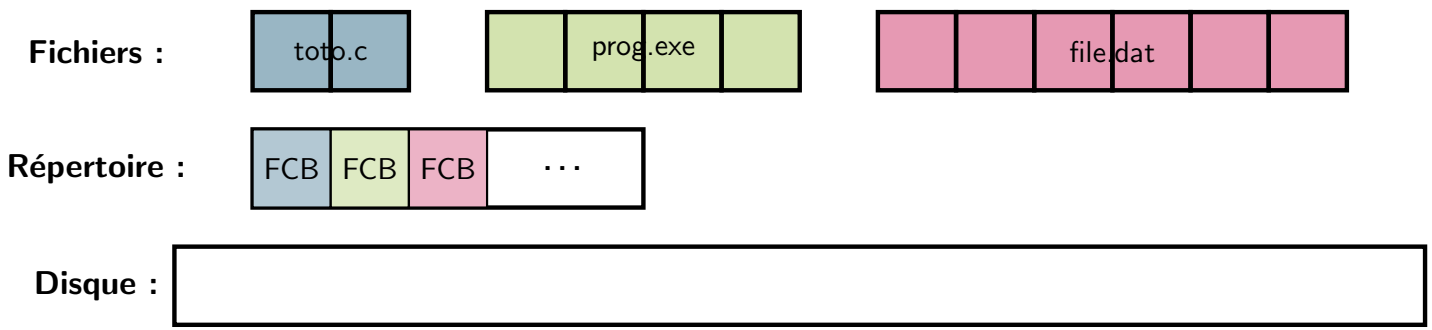
Ranger les blocs les uns derrière les autres



Allocation contiguë

Principe

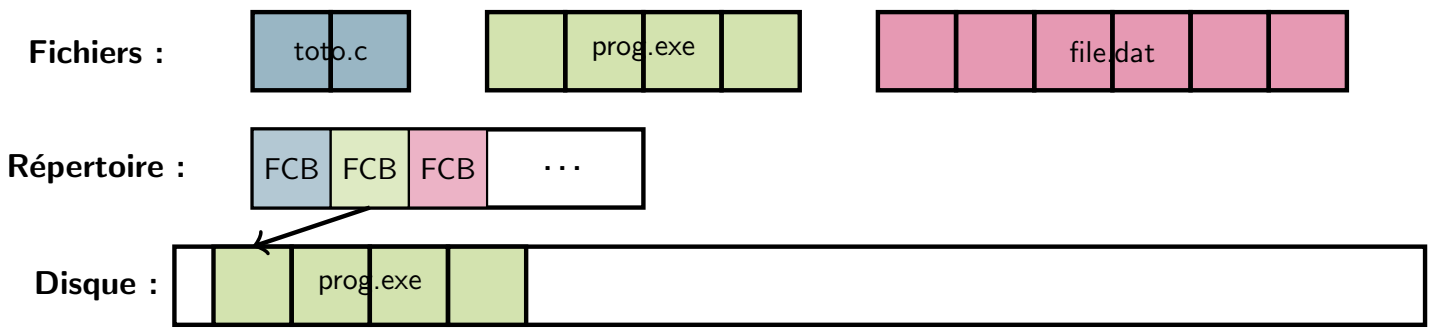
Ranger les blocs les uns derrière les autres



Allocation contiguë

Principe

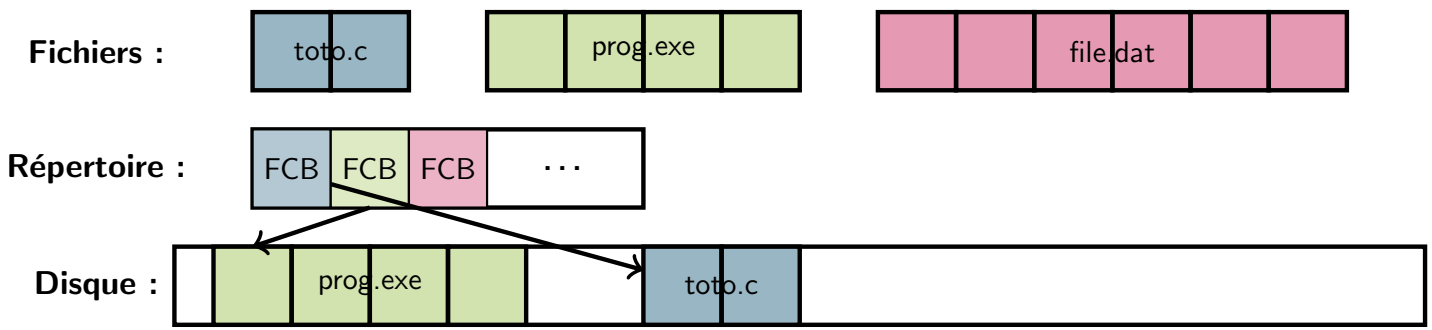
Ranger les blocs les uns derrière les autres



Allocation contiguë

Principe

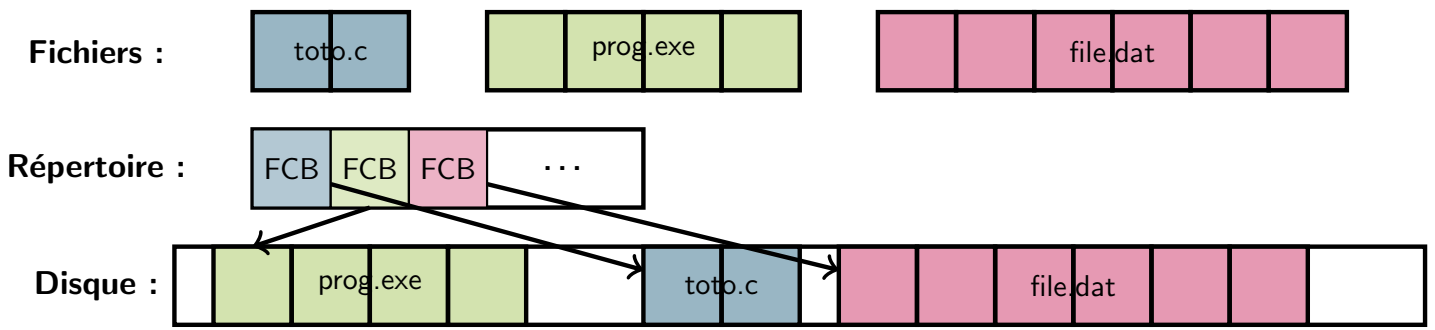
Ranger les blocs les uns derrière les autres



Allocation contiguë

Principe

Ranger les blocs les uns derrière les autres



Allocation contiguë

Principe

Ranger les blocs les uns derrière les autres

Avantages

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

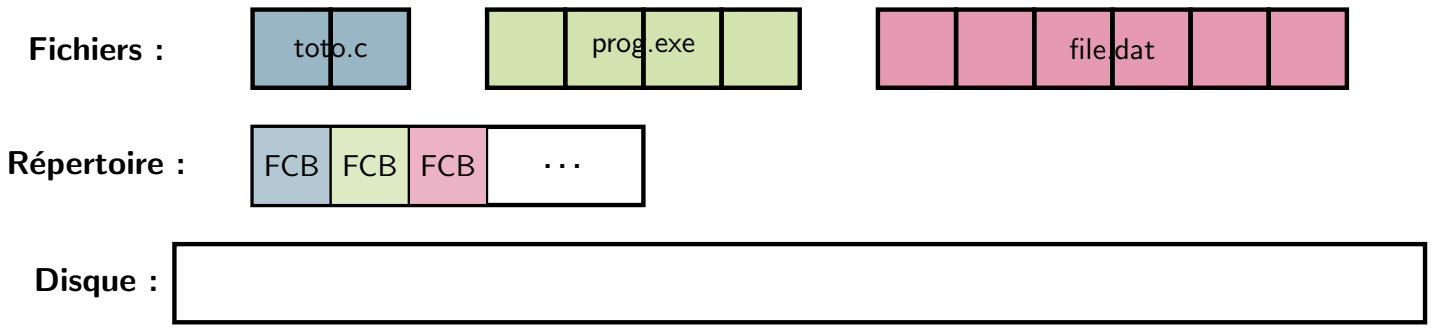
Inconvénients

- ✗ 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

Principe

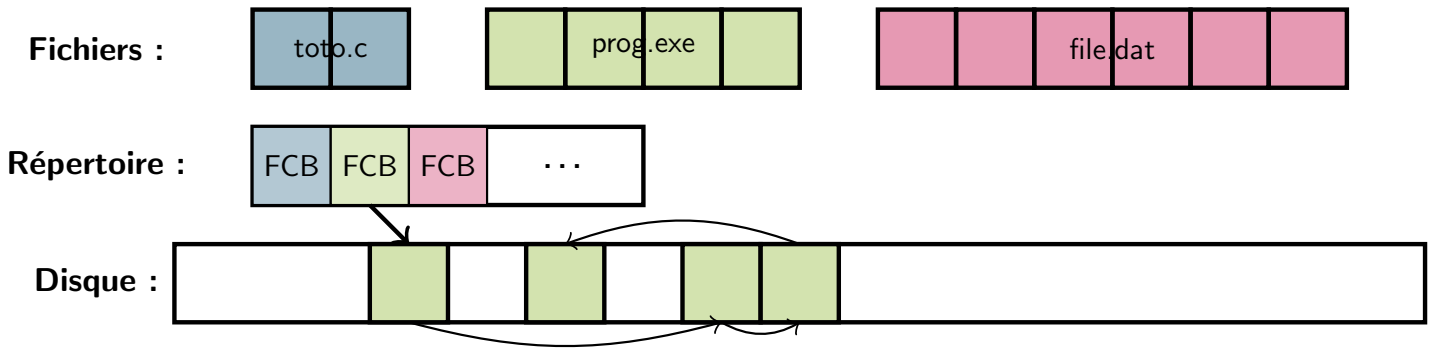
Fichier = liste chaînée de blocs



Allocation chaînée

Principe

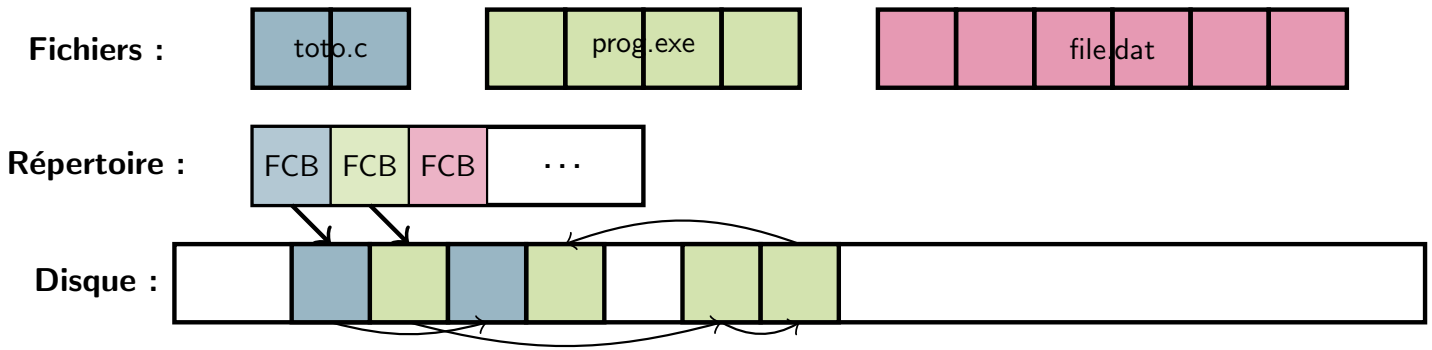
Fichier = liste chaînée de blocs



Allocation chaînée

Principe

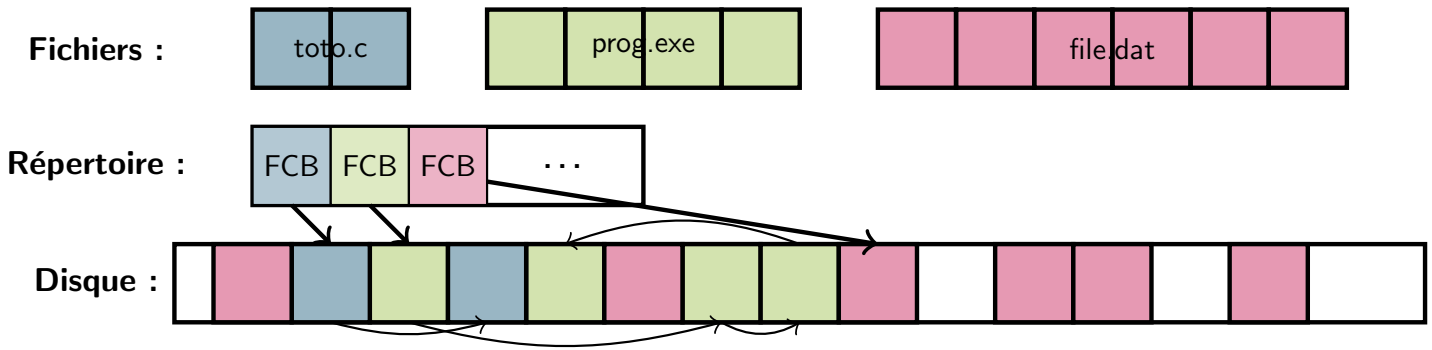
Fichier = liste chaînée de blocs



Allocation chaînée

Principe

Fichier = liste chaînée de blocs



Allocation chaînée

Principe

Fichier = liste chaînée de blocs

Avantages

- ✓ FCB : adresse premier et dernier blocs
- ✓ Pas de fragmentation
- ✓ Fichiers taille quelconque

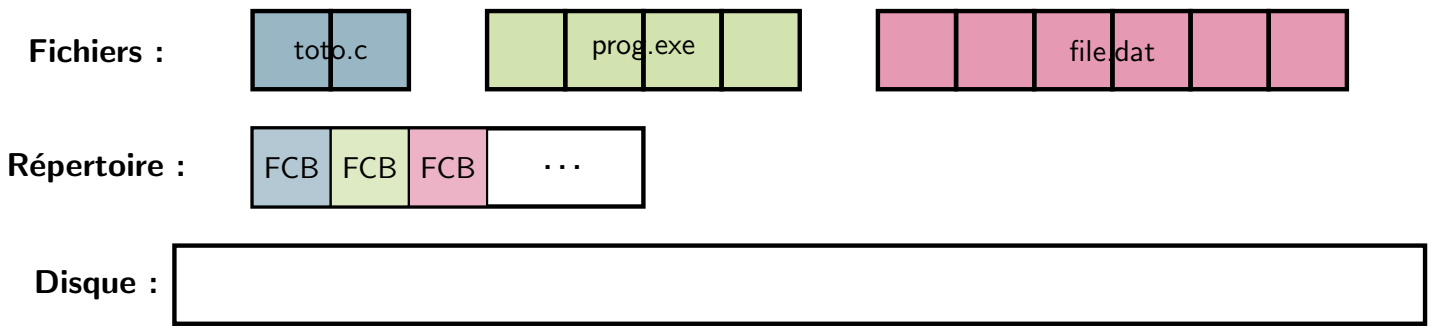
Inconvénients

- ✗ Accès séquentiel : N^e bloc $\rightarrow$ N accès disques !
- ✗ Fiabilité : 1 bloc endommagé $\rightarrow$ tout le fichier est perdu
- ✗ Espace utilisé par les pointeurs

Allocation indexée

Principe

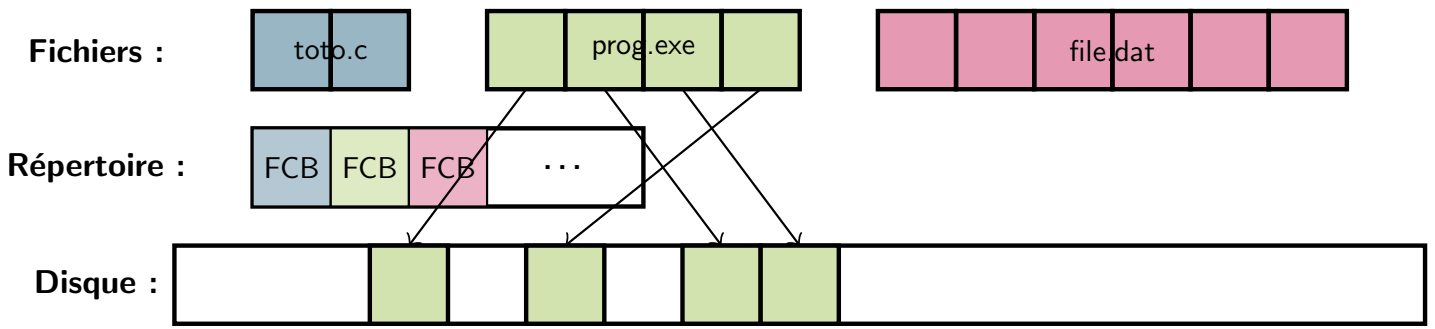
Rassembler tous les pointeurs dans un bloc d'index
(1 bloc par fichier)



Allocation indexée

Principe

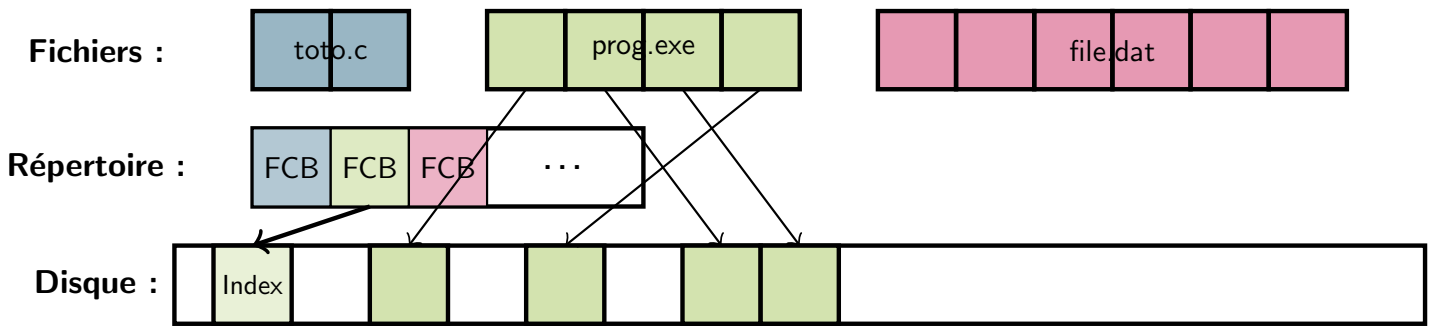
Rassembler tous les pointeurs dans un bloc d'index
(1 bloc par fichier)



Allocation indexée

Principe

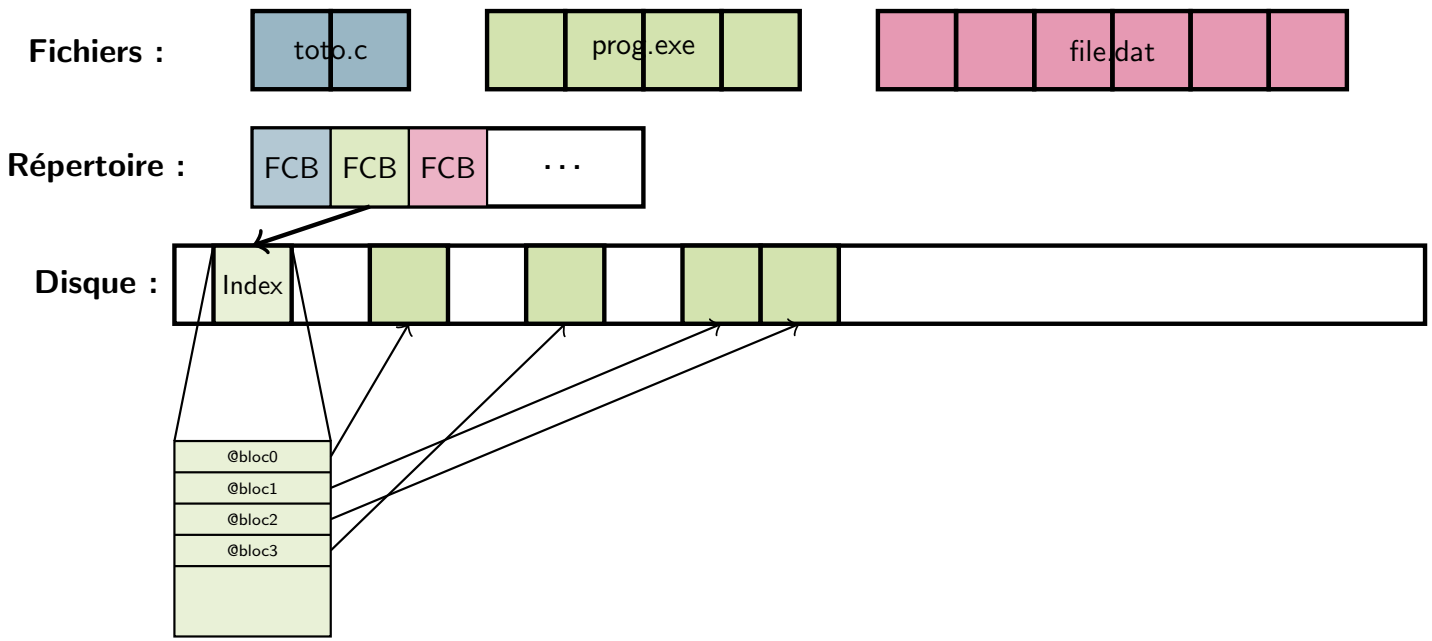
Rassembler tous les pointeurs dans un bloc d'index
(1 bloc par fichier)



Allocation indexée

Principe

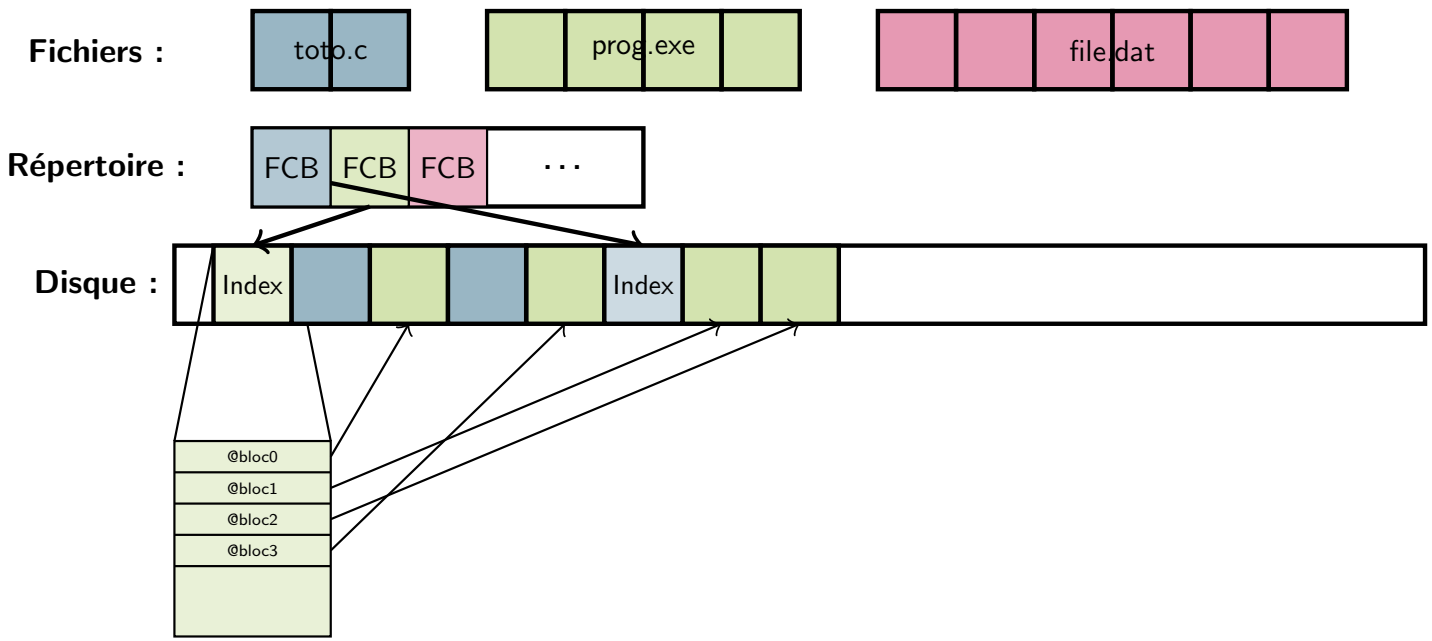
Rassembler tous les pointeurs dans un bloc d'index
(1 bloc par fichier)



Allocation indexée

Principe

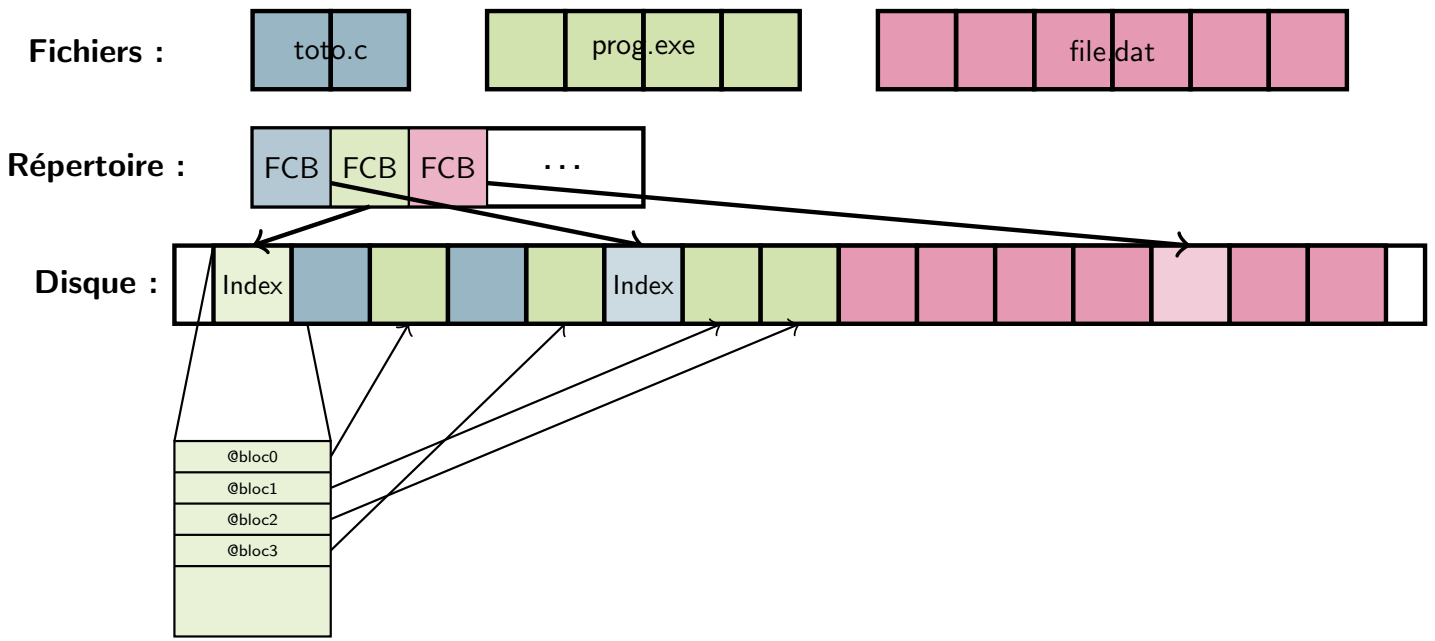
Rassembler tous les pointeurs dans un bloc d'index
(1 bloc par fichier)



Allocation indexée

Principe

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



Allocation indexée

Principe

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

$$64\ bits \times 64\ blocs = 512o \qquad \Rightarrow max = 64 \times 512o = 32\ Ko$$

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

➜ Taille de l'adresse = ?

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

✓ Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits $=$ 2 octets

➜ Nombre max d'index/bloc $=$?

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

✓ Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits = 2 octets

✓ Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs

➜ Taille max d'un fichier = ?

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

✓ Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits = 2 octets

✓ Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs

✓ Taille max d'un fichier = 4 blocs = 32o

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

✓ Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits = 2 octets

✓ Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs

✓ Taille max d'un fichier = 4 blocs = 32o

Fichier :

Disque :

Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

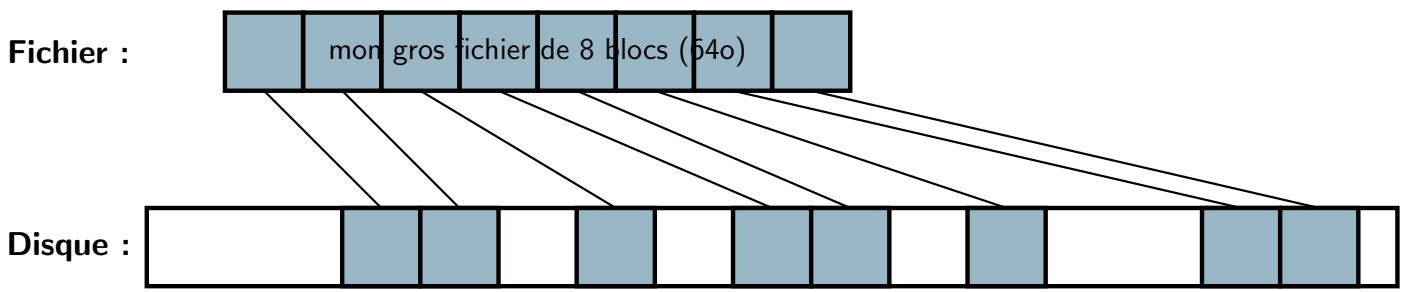
Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

✓ Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits = 2 octets

✓ Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs

✓ Taille max d'un fichier = 4 blocs = 32o



Allocation indexée

Problème

Fichiers nécessitant plus d'un bloc d'index

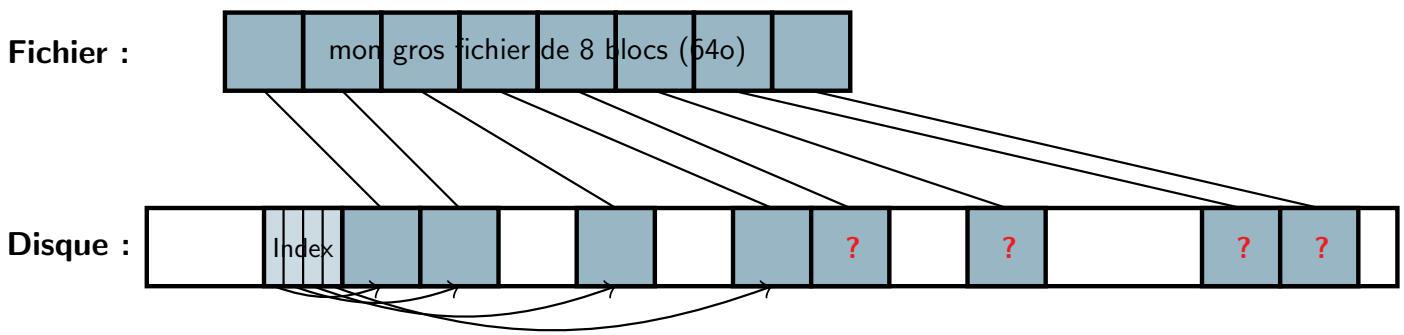
Exemple :

- 512ko de disque, blocs de 8o (exemple non réaliste !)

✓ Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits = 2 octets

✓ Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs

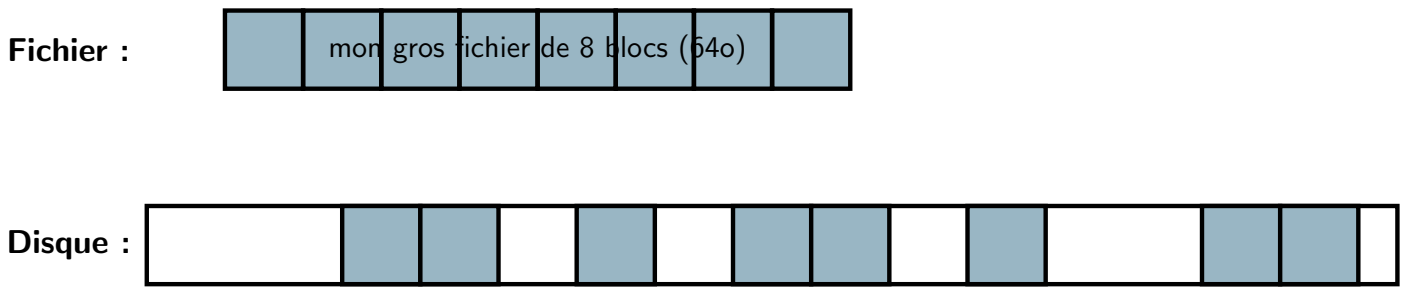
✓ Taille max d'un fichier = 4 blocs = 32o



Solution 1 : liste chaînée de blocs d'index

Principe

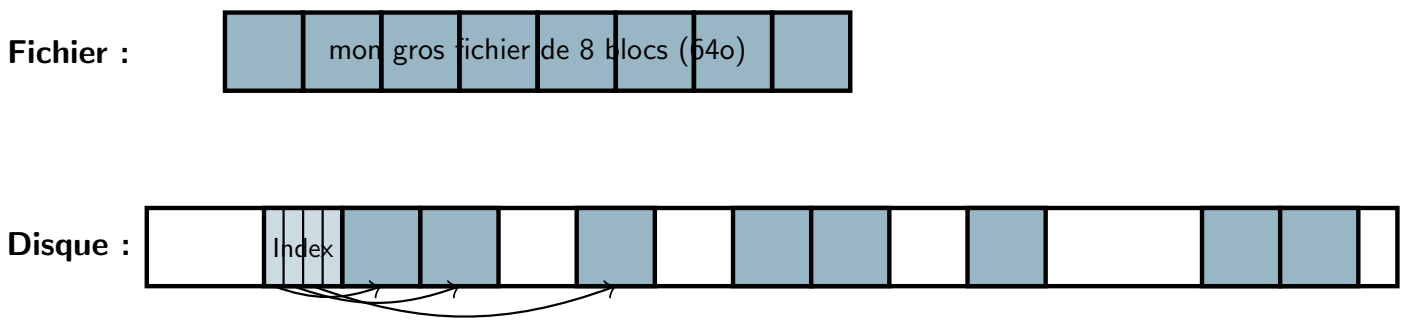
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

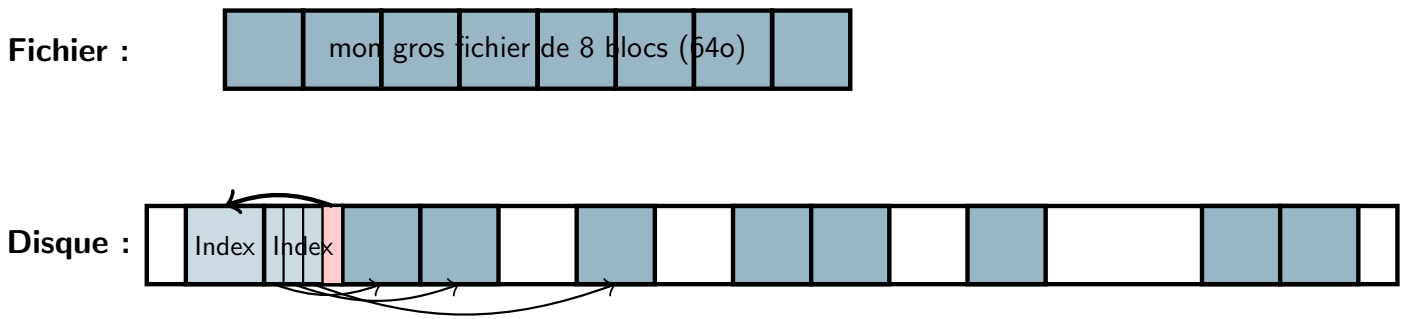
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

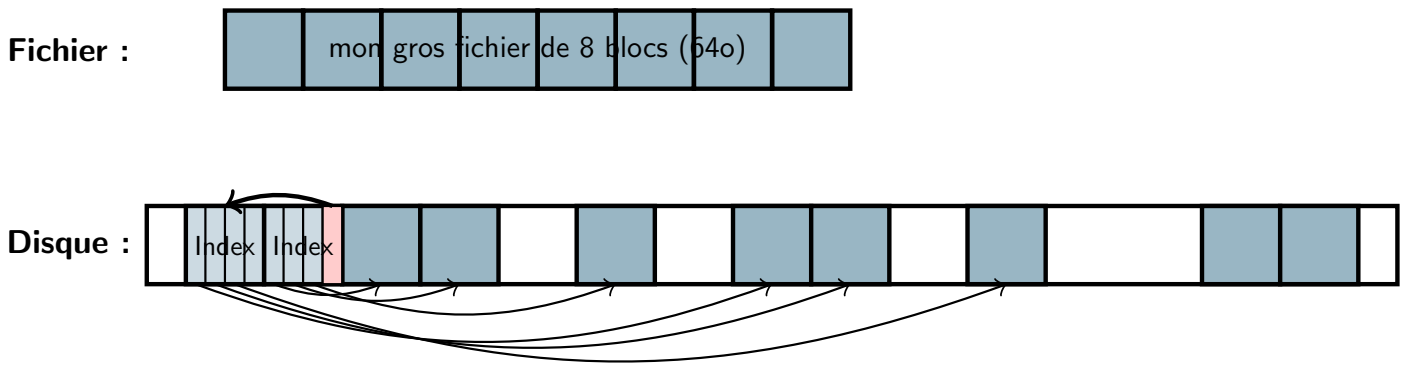
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

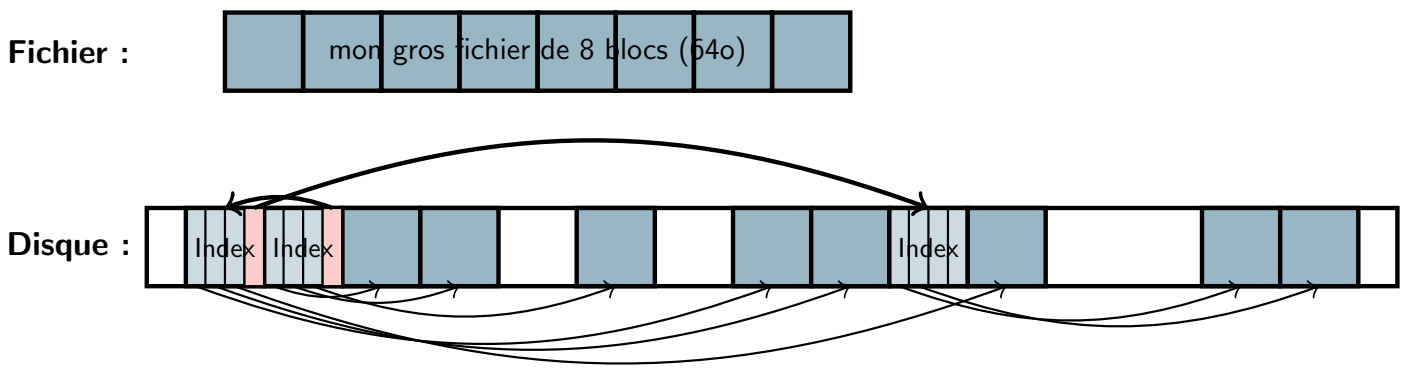
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

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



Solution 1 : liste chaînée de blocs d'index

Principe

- Le répertoire pointe vers un bloc d'index
- Chaque 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

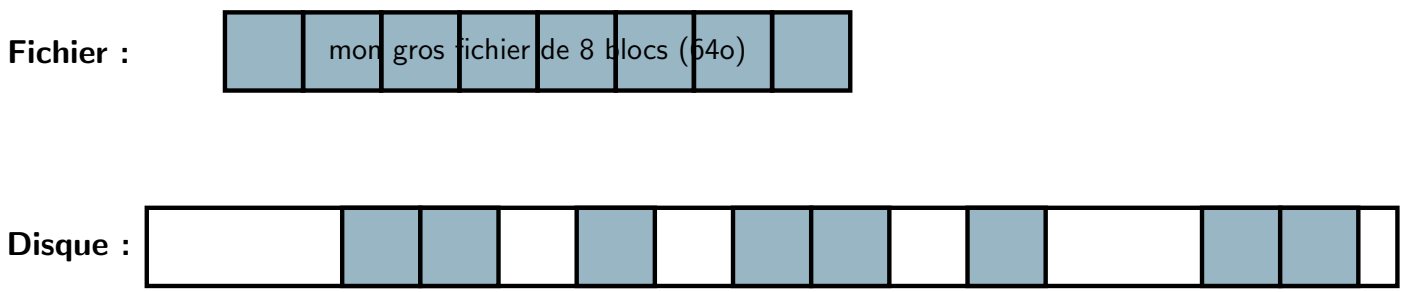
✗ Accès indirect ($\geq$2 accès disque) mais plus rapide que la liste chaînée de blocs !

Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index « maître »
- Le bloc maître pointe vers des blocs d'index
  $\rightarrow n^2$ blocs indexables au lieu de n

  Ex : 16 blocs au lieu de 4
- Éventuellement, indexation sur 3 niveaux (n^3)

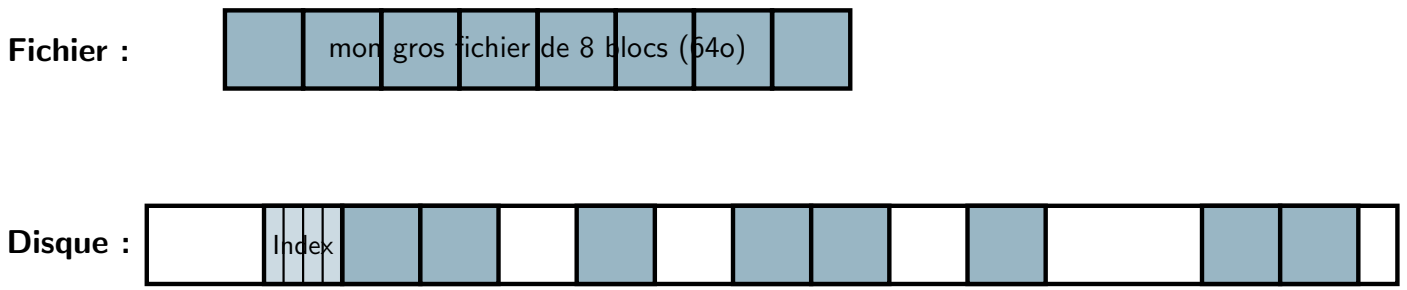


Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index « maître »
- Le bloc maître pointe vers des blocs d'index
  $\rightarrow n^2$ blocs indexables au lieu de n

  Ex : 16 blocs au lieu de 4
- Éventuellement, indexation sur 3 niveaux (n^3)

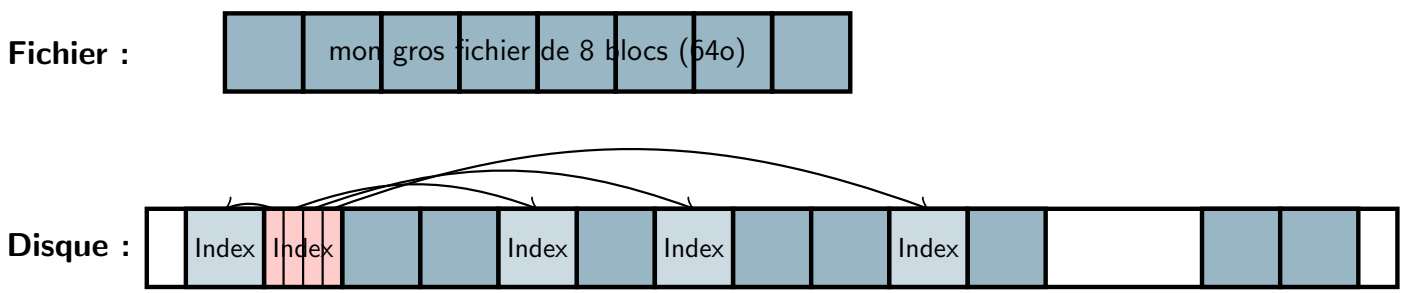


Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index « maître »
- Le bloc maître pointe vers des blocs d'index
  $\rightarrow n^2$ blocs indexables au lieu de n

  Ex : 16 blocs au lieu de 4
- Éventuellement, indexation sur 3 niveaux (n^3)

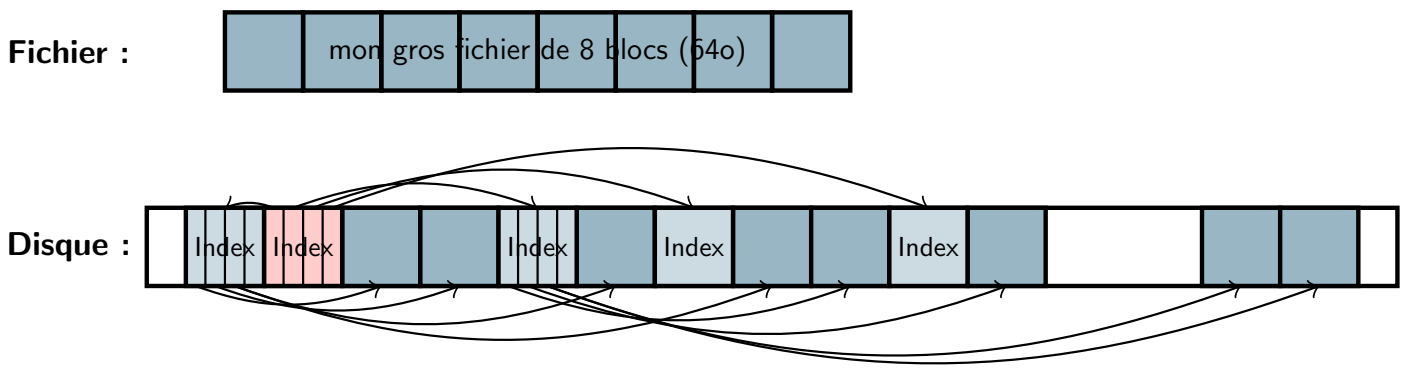


Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index « maître »
- Le bloc maître pointe vers des blocs d'index
  $\rightarrow n^2$ blocs indexables au lieu de n

  Ex : 16 blocs au lieu de 4
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

- Le répertoire pointe vers un bloc d'index « maître »
- Le bloc maître pointe vers des blocs d'index

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)

Principe

➜ Combiner allocation chaînée et allocation indexée

Index = k premiers blocs du fichier + $n - k$ blocs d'indirection (index 2 à $n - k + 1$)

✓ Moins de perte pour les petits fichiers

✓ Accès rapide

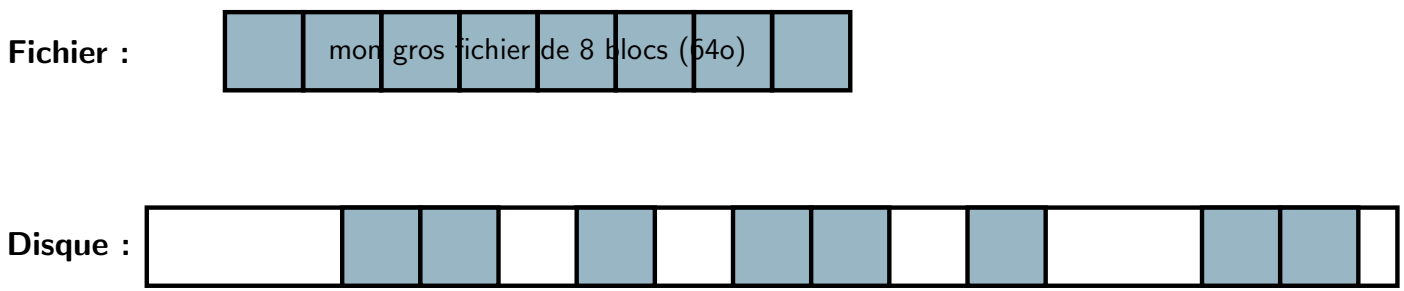


Schéma combiné (ex : Linux extfs)

Principe

➜ Combiner allocation chaînée et allocation indexée

Index = k premiers blocs du fichier + $n - k$ blocs d'indirection (index 2 à $n - k + 1$)

✓ Moins de perte pour les petits fichiers

✓ Accès rapide

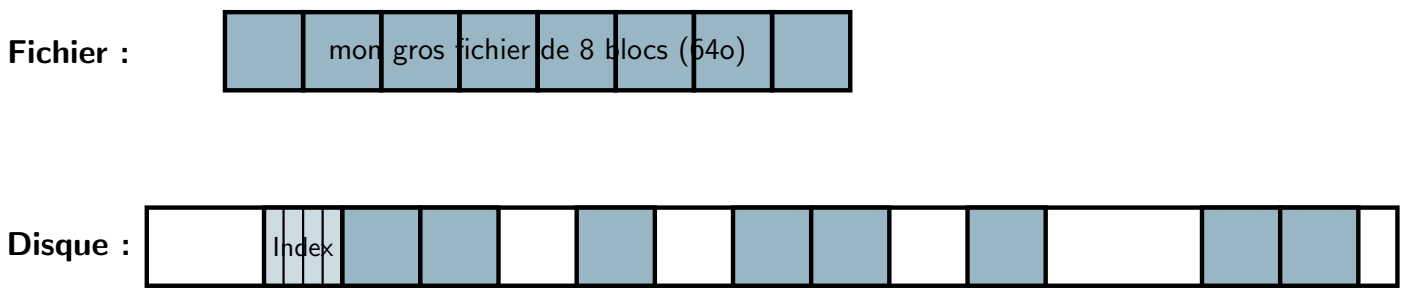


Schéma combiné (ex : Linux extfs)

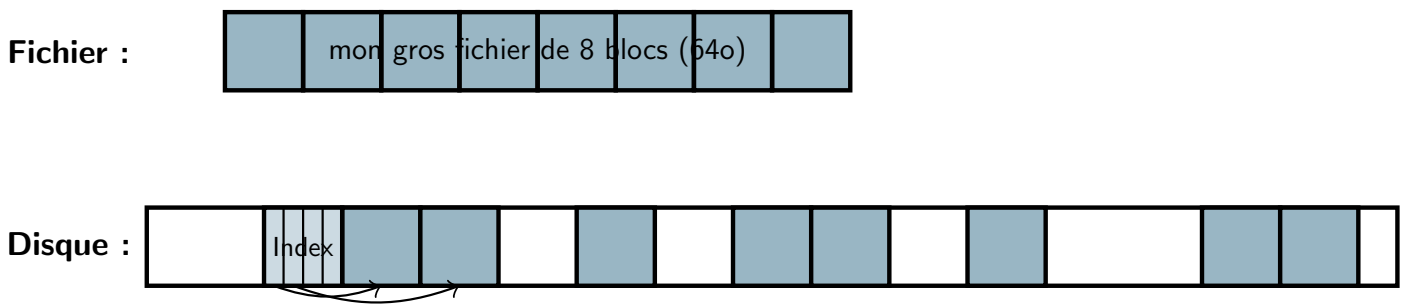
Principe

➜ Combiner allocation chaînée et allocation indexée

Index = k premiers blocs du fichier + $n - k$ blocs d'indirection (index 2 à $n - k + 1$)

✓ Moins de perte pour les petits fichiers

✓ Accès rapide



Ici, k=2 (et n=4)

Schéma combiné (ex : Linux extfs)

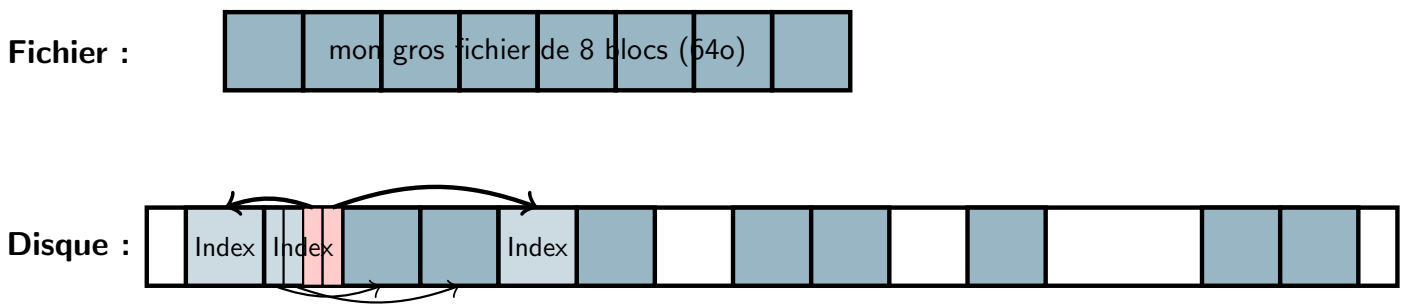
Principe

➜ Combiner allocation chaînée et allocation indexée

Index = k premiers blocs du fichier + $n - k$ blocs d'indirection (index 2 à $n - k + 1$)

✓ Moins de perte pour les petits fichiers

✓ Accès rapide



Ici, k=2 (et n=4)

Schéma combiné (ex : Linux extfs)

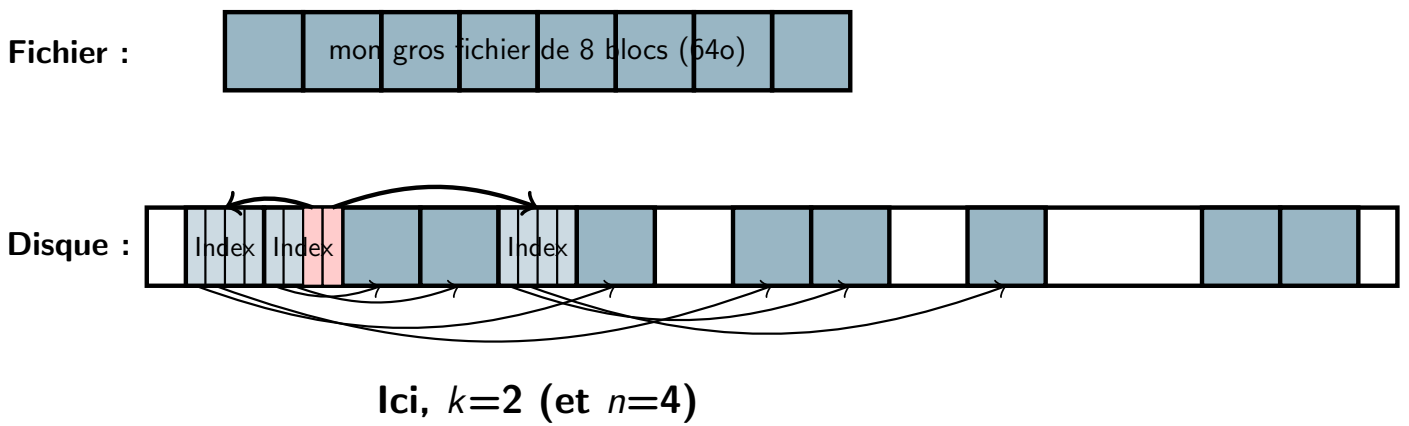
Principe

➜ Combiner allocation chaînée et allocation indexée

Index = k premiers blocs du fichier + $n - k$ blocs d'indirection (index 2 à $n - k + 1$)

✓ Moins de perte pour les petits fichiers

✓ Accès rapide



Ici, k=2 (et n=4)

Schéma combiné (ex : Linux extfs)

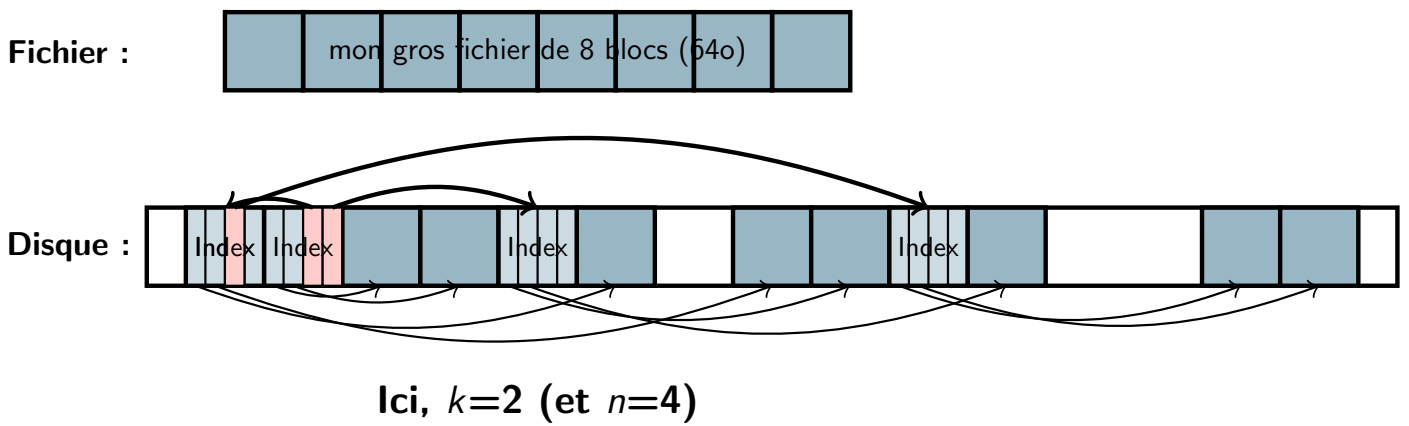
Principe

➜ Combiner allocation chaînée et allocation indexée

Index = k premiers blocs du fichier + $n - k$ blocs d'indirection (index 2 à $n - k + 1$)

✓ Moins de perte pour les petits fichiers

✓ Accès rapide

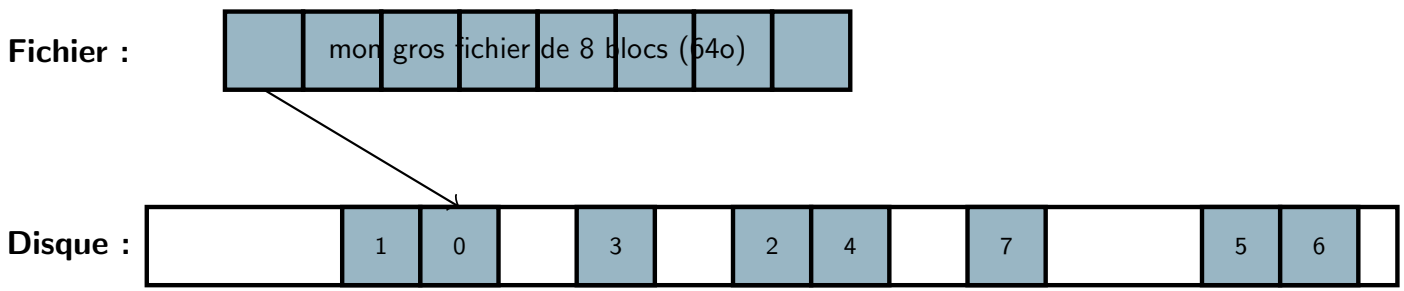


Ici, k=2 (et n=4)

File Allocation Table (FAT)

Principe

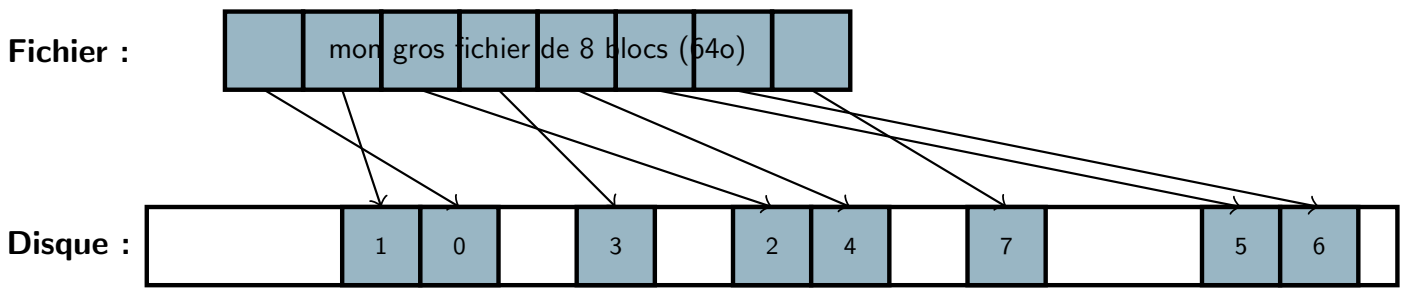
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

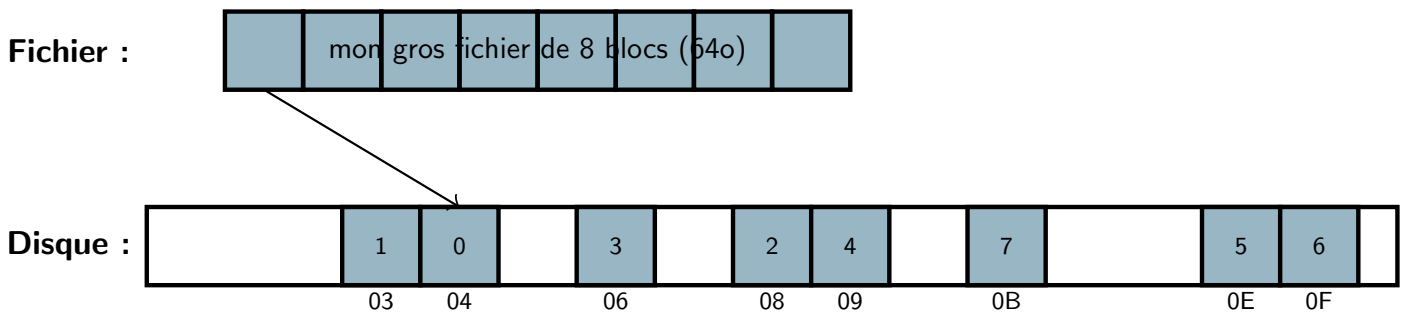
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

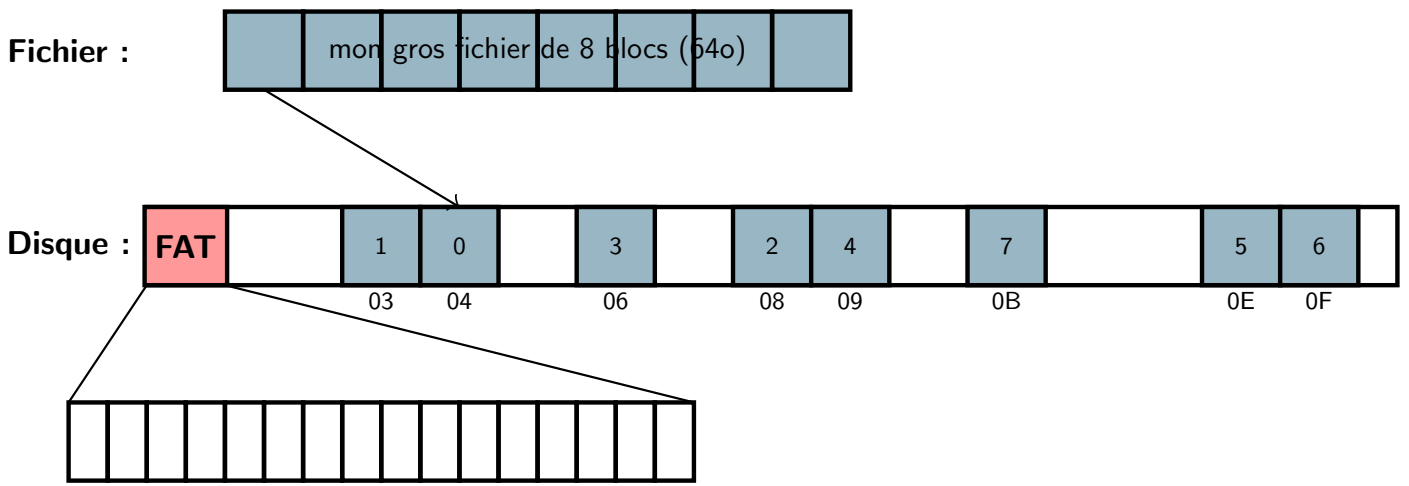
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

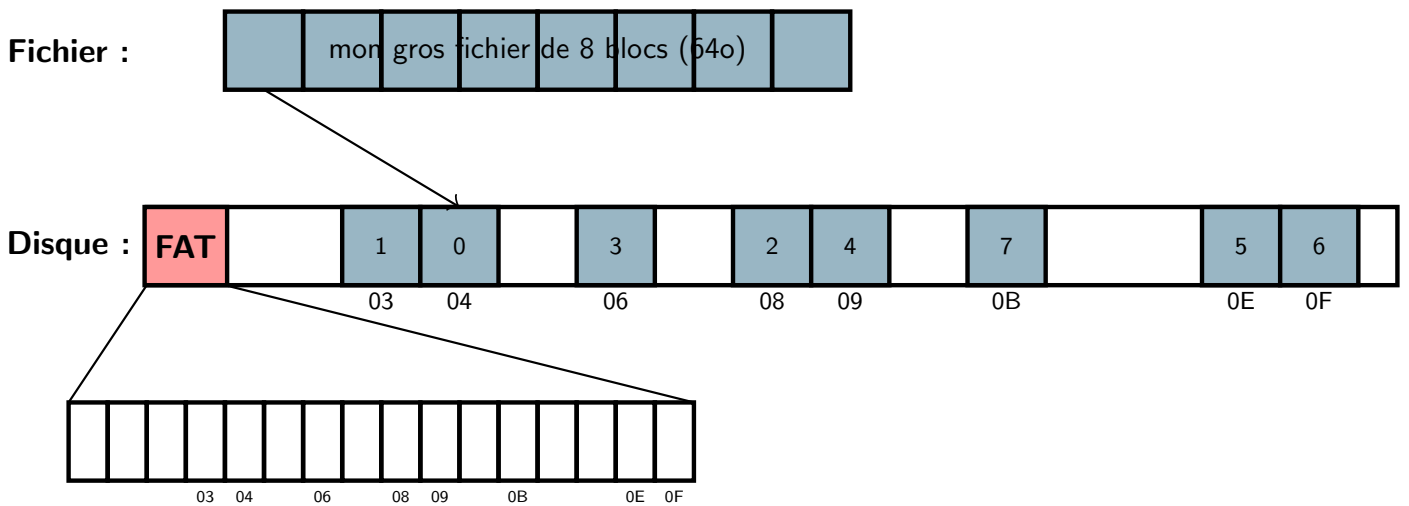
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

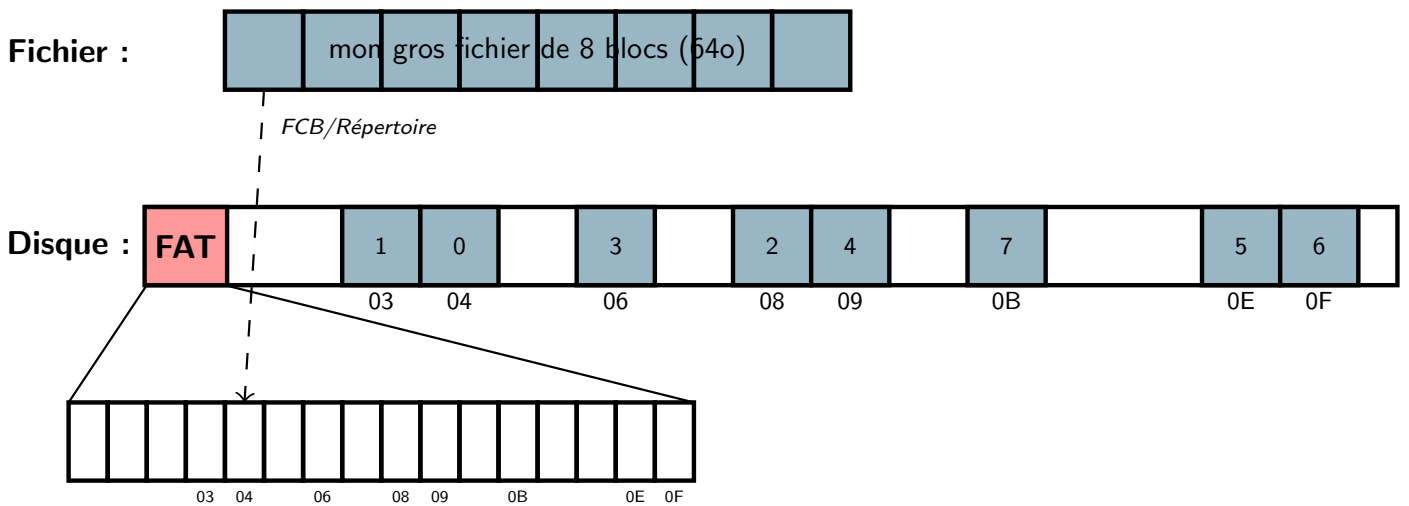
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

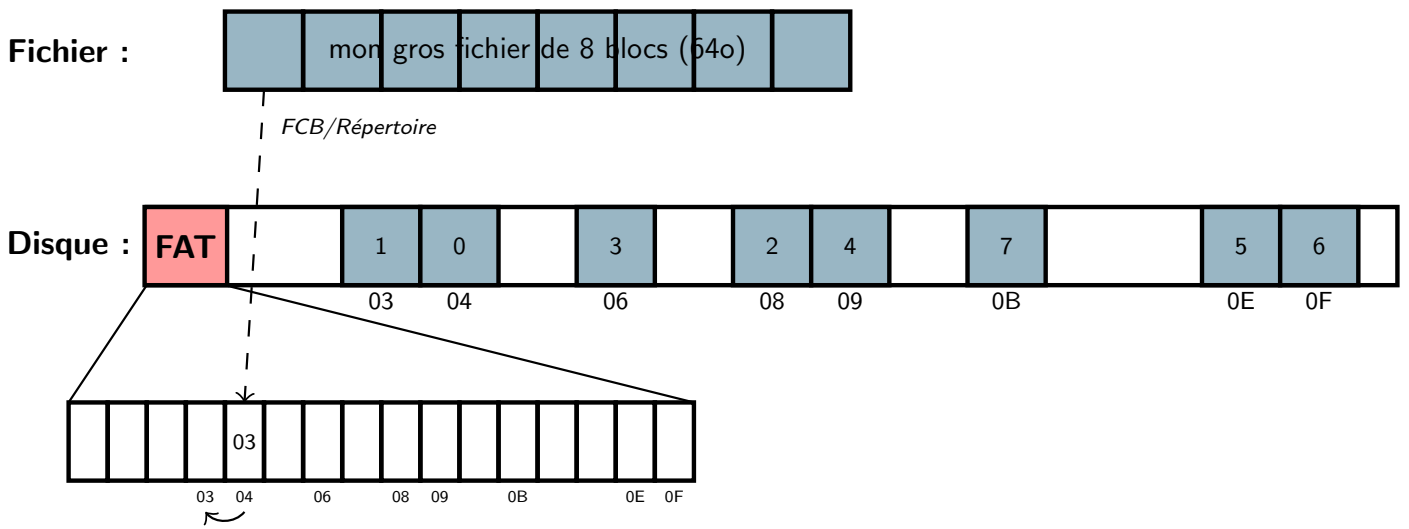
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

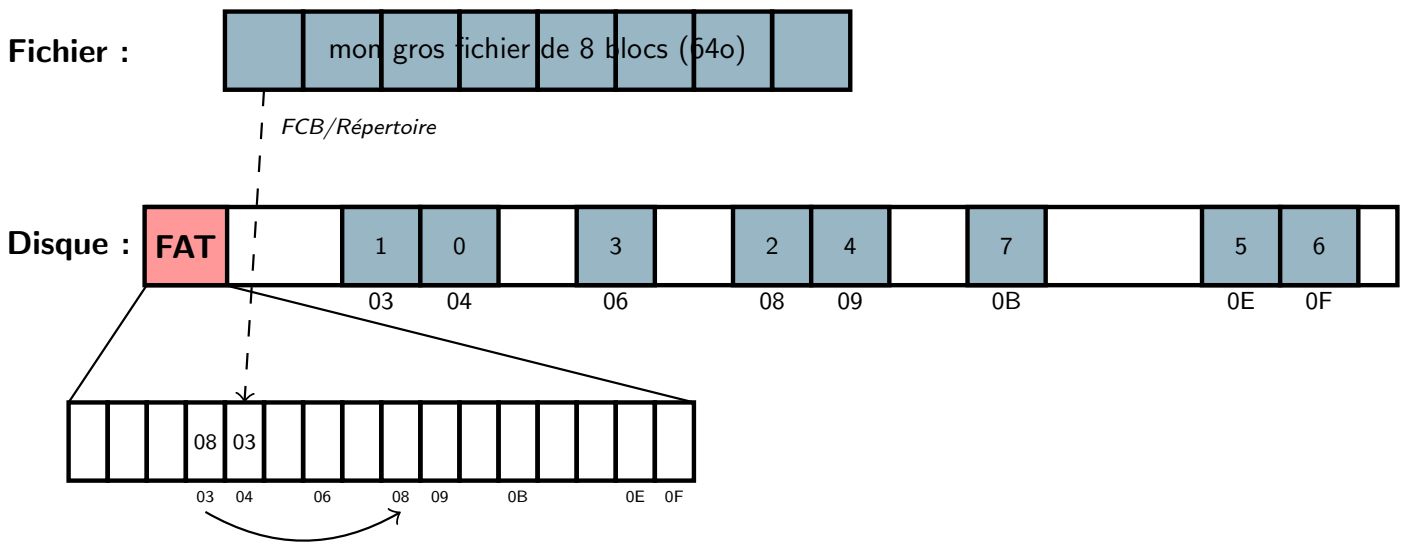
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

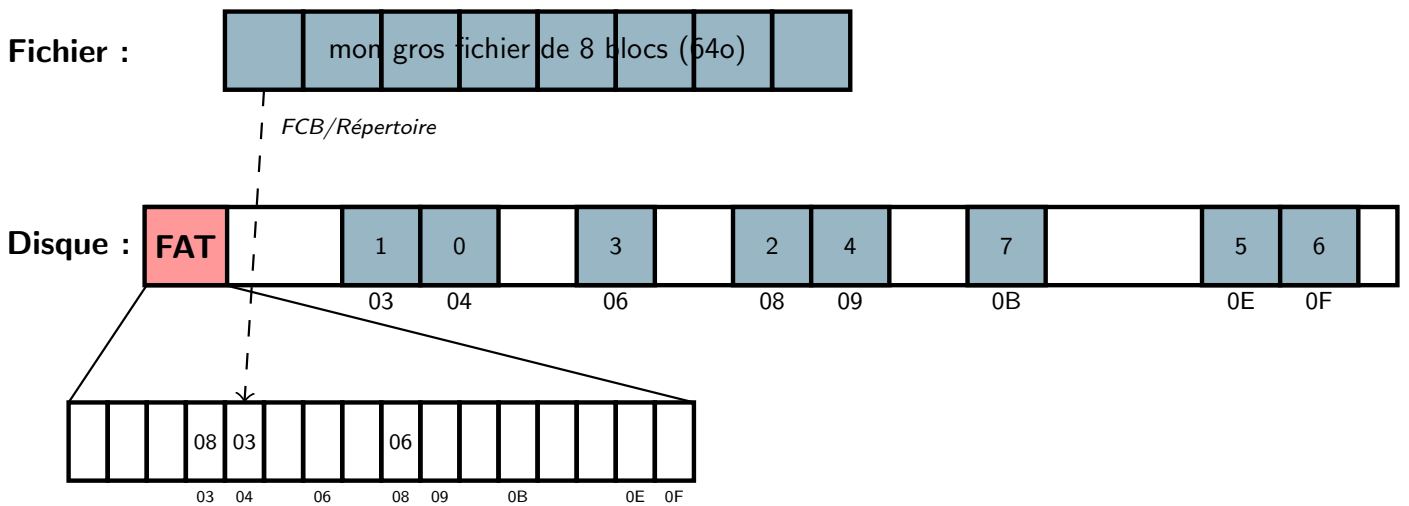
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

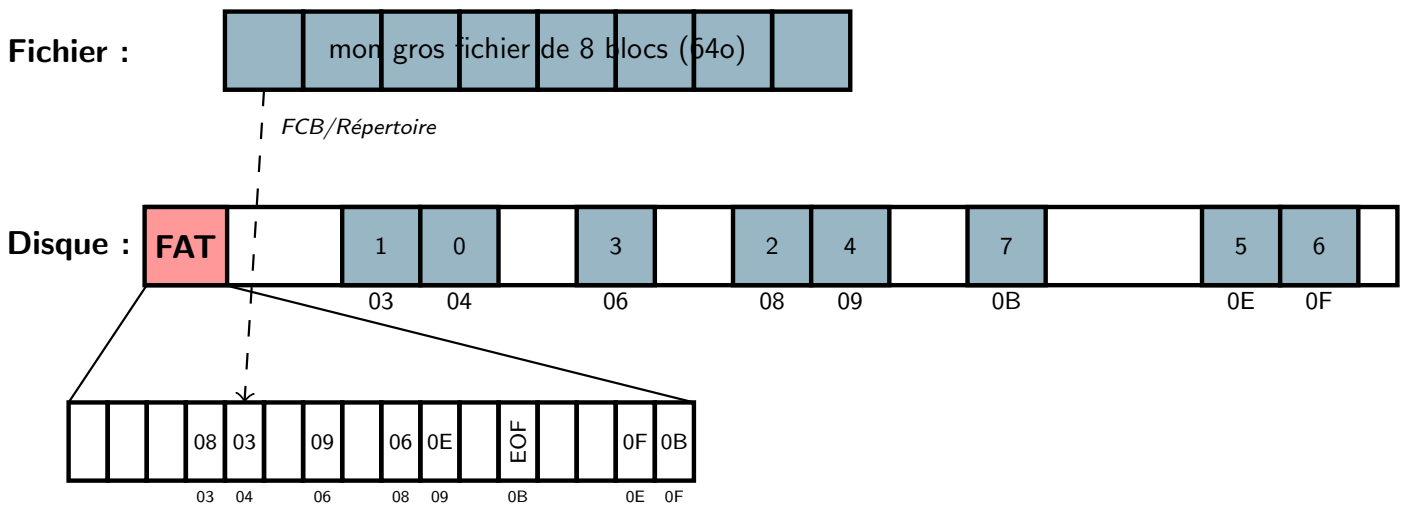
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition



File Allocation Table (FAT)

Principe

- 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)

Principe

- Allocation indexée
- Liste chaînée des index des blocs en début de chaque partition

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

Ce qu'il faut retenir

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