Systèmes de fichiers

Thomas Lavergne
lavergne@lisn.fr

Partie visible de l'OS

- Stockage sur support physique
- Accès aux données et programmes
- Accès à l'ensemble du système

Partie visible de l'OS

- Stockage sur support physique
- Accès aux données et programmes
- Accès à l'ensemble du système

Définition

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

Partie visible de l'OS

- Stockage sur support physique
- Accès aux données et programmes
- Accès à l'ensemble du système

Définition

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

Unité *logique*

- Indépendante du support physique
- Abstraction des propriétés physiques

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

Fichiers texte : `.txt`

Données textuelles à l'usage de l'utilisateur humain

$\longrightarrow$ Succession de caractères, organisés en lignes séparées par un caractère spécial (`\n`, `^M`, `...` `selon l'OS`)

Structure d'un fichier : exemples

Fichiers texte : `.txt`

Données textuelles à l'usage de l'utilisateur humain

$\longrightarrow$ Succession de caractères, organisés en lignes séparées par un caractère spécial (`\n`, `^M`, ... selon l'OS)

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

Fourni par un utilisateur humain pour être traité par la machine

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

Structure d'un fichier : exemples

Bibliothèques : `.o`, `.dll`, ...

Construits par un compilateur à partir d'un fichier source

$\longrightarrow$ Succession d'octets organisés en blocs interprétables par l'éditeur de lien.

Structure d'un fichier : exemples

Bibliothèques : `.o`, `.dll`, ...

Construits par un compilateur à partir d'un fichier source

$\longrightarrow$ Succession d'octets organisés en blocs interprétables par l'éditeur de lien.

Exécutable : `.exe`, ...

$\longrightarrow$ 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

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

Définition

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

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

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 $\longrightarrow$ identifiant processus
- Contrôle d'accès dans le FCB

Liste de contrôle d'accès (ACL)

Utilisateur $\longrightarrow$ droits

Problèmes :

- X les utilisateurs doivent être connus a priori
- X taille du FCB ! (grossit avec le nombre d'utilisateurs)

Liste de contrôle d'accès (ACL)

Utilisateur $\longrightarrow$ droits

Problèmes :

- ✗ les utilisateurs doivent être connus 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, ...)

- ✗ Pas très pratique $\longrightarrow$ peu utilisé

Classes d'utilisateurs

Exemple : Propriétaires vs Autres

$\longrightarrow$ quelques bits par fichier

Classes d'utilisateurs

Exemple : Propriétaires vs Autres

$\longrightarrow$ 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 $\longrightarrow$ liste de groupes

Ex : `robin, g=[dev-ram,user-disque]`

$\longrightarrow$ `robin` n'a pas accès à `toto.c`

Exemple : Unix

Classes + groupes

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

$\rightarrow$ 3×3 bits par fichier

Exemple : Unix

Classes + groupes

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

$\longrightarrow$ 3×3 bits par fichier

Exemple :

```
$ ls -l
drwx------ 6 tom prof 4096 nov.  2 12:06 private
-rwx------ 1 tom prof 2356 déc.  5 15:31 notes.tex
drwxrwx--- 8 tom prof 4096 déc.  4 17:30 doc
-rwxrwxr-x 1 joe stud 9234 jan.  1 11:12 prog.c
```

Notion de répertoire

Définition

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

Notion de répertoire

Définition

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

Structure

- Entrée du répertoire = id et/ou nom du fichier
  $\rightarrow$ deux méthodes d'accès
- Contenu du répertoire = FCB des fichiers
  $\rightarrow$ plusieurs ko de données non scindables !

Principe

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

Interface

Opérations sur un fichier

Appels systèmes de base

- Création : allocation espace + entrée répertoire
- Lecture : pointeur de lecture
- Écriture : pointeur d'écriture
- Repositionnement : déplacer le pointeur
- Suppression : retrait de l'entrée dans le répertoire
- Troncature : changement de taille

Opérations sur un fichier

Appels systèmes de base

- Création : allocation espace + entrée répertoire
- Lecture : pointeur de lecture
- Écriture : pointeur d'écriture
- Repositionnement : déplacer le pointeur
- Suppression : retrait de l'entrée dans le répertoire
- Troncature : changement de taille

Opérations composées

Ex : copie, renommage

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

Problème

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

$\longrightarrow$ 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

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

Définition

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

Ouverture de fichier

Problème

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

$\longrightarrow$ 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.

Stockage des FCB en RAM

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

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

Stockage des FCB en RAM

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

- Ouverture $\rightarrow$ chargement du FCB
- Fermeture $\rightarrow$ retrait de la table
- 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 globale + une table par processus

Répertoires

Structure des disques

Support de stockage *(disque)*

Structure physique de stockage permanent.

Support de stockage *(disque)*

Structure physique de stockage permanent.

Partition

Structure logique (disque « virtuel »)

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

Support de stockage *(disque)*

Structure physique de stockage permanent.

Partition

Structure logique (disque « virtuel »)

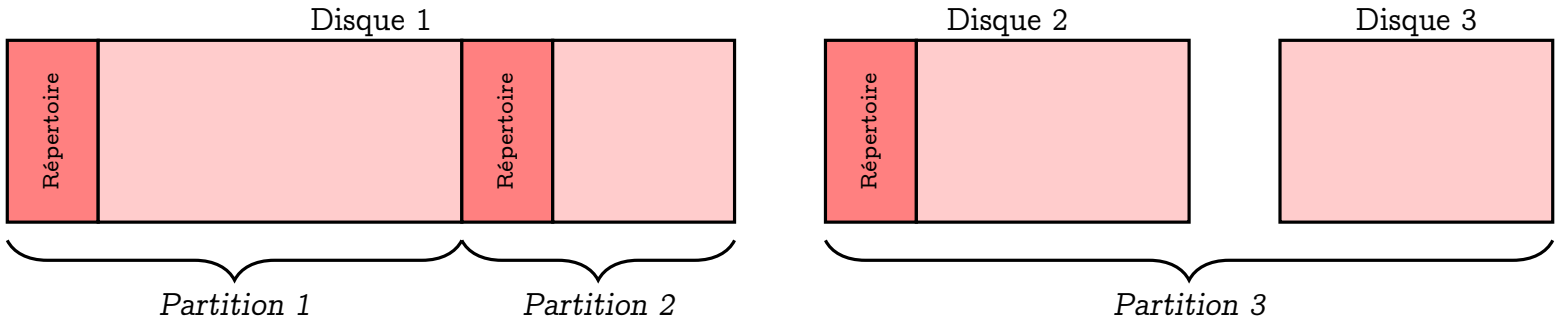
- Base : 1 disque = 1 partition
- 1 disque = N partitions
- 1 partition = 1 ou N disques (selon OS)

Répertoire

Un répertoire par partition : l'ensemble des FCB

- Nom/id→ FCB

Répertoire : structure de base



Structure à 1 niveau

✓ Nom → FCB

✗ Taille du répertoire proportionnelle au nombre de fichiers

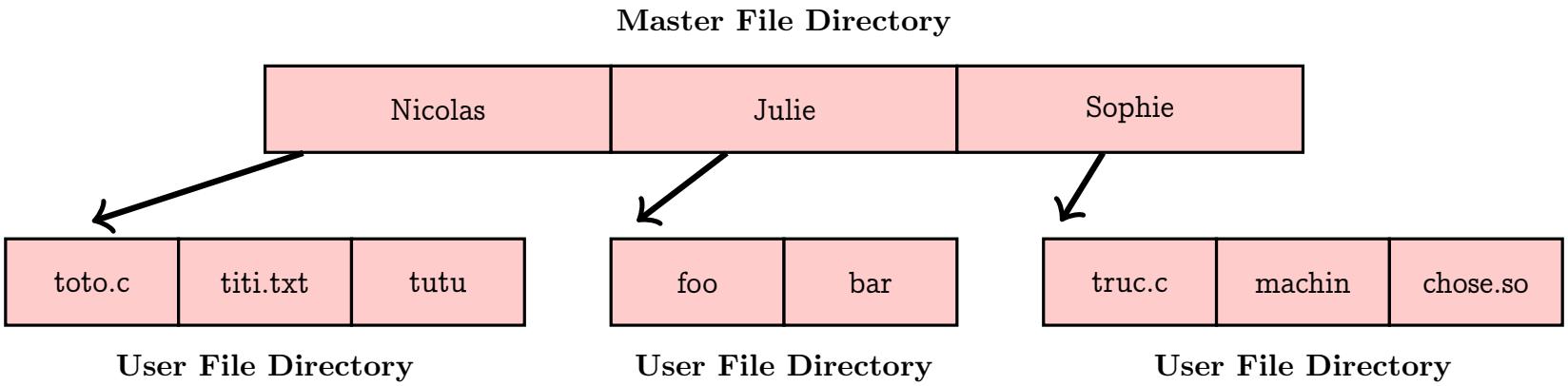
→ borner le nombre de fichiers...

✗ Utilisateur : organisation, unicité de nom, ...

Principe

Un répertoire par utilisateur

- Master File Directory (MFD)
- User File Directory (UFD)



Remarques

- Les UFD sont des fichiers
- Le nom de chaque UFD doit être unique

Mais deux fichiers de deux UFD différents peuvent avoir le même nom !

`nicolas/cours1.pdf` `julie/cours1.pdf`

Structure à deux niveaux : avantages et inconvénients

Remarques

- Les UFD sont des fichiers
- Le nom de chaque UFD doit être unique

Mais deux fichiers de deux UFD différents peuvent avoir le même nom !

`nicolas/cours1.pdf` `julie/cours1.pdf`

Accès

- Les fichiers des utilisateurs sont isolés
- Nommage : `partition + user + fichier`

Structure à deux niveaux : partage de fichiers

Fichiers de l'OS

- Exemples : bibliothèques, routines, etc

$\rightarrow$ Problème : comment ne pas les charger dans chaque UFD ?

Structure à deux niveaux : partage de fichiers

Fichiers de l'OS

- Exemples : bibliothèques, routines, etc

$\rightarrow$ Problème : comment ne pas les charger dans chaque UFD ?

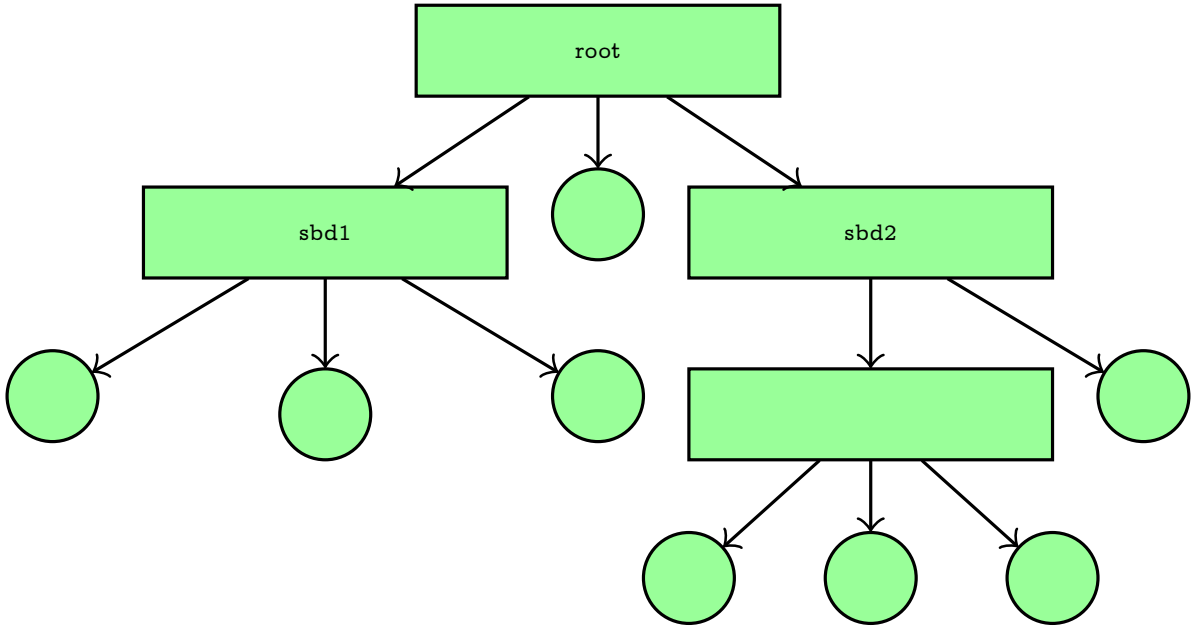
Utilisateur OS

- Répertoire d'utilisateur spécial
- Par défaut si absent du répertoire de l'utilisateur
- Ordre de recherche spécifié (variable `PATH`)

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

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)

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

Principe

Généralisation de l'arbre avec des liens

- Graphe acyclique

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
- Compteur de liens

Système de fichiers

Différence avec la RAM

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

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
  - Les données à stocker sont découpées en bloc
- Taille fixe (de 32o à 16Mo selon support)

Exemples : disques dur années 2000 = 512o, SSD actuel = 1Mo

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
  - Les données à stocker sont découpées en bloc
- Taille fixe (de 32o à 16Mo selon support)

Exemples : disques dur années 2000 = 512o, SSD actuel = 1Mo

Système de fichiers *(FileSystem)*

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

Système logique

- Structure de répertoires
- FCB + gestion de la protection

Système logique

- Structure de répertoires
- FCB + gestion de la protection

Système physique

- Fichiers $\longrightarrow$ ensemble de blocs logiques
- Bloc logique $\longrightarrow$ blocs physique *cf. mémoire paginée*
- Identification des blocs physiques selon support

Système logique

- Structure de répertoires
- FCB + gestion de la protection

Système physique

- Fichiers $\longrightarrow$ ensemble de blocs logiques
- Bloc logique $\longrightarrow$ blocs physique *cf. mémoire paginée*
- Identification des blocs physiques selon support

Lien : pilote de périphérique

ex : chargement bloc 456 $\longrightarrow$ instruction matérielle

Le système de fichiers associe...

... des noms à des blocs logiques (fichiers)

... des noms à des répertoires venus du disque !

Le système de fichiers associe...

... des noms à des blocs logiques (fichiers)

... des noms à des répertoires venus du disque !

Montage de répertoire

Le montage consiste à positionner un répertoire dans le système de fichier

Le système de fichiers associe...

... des noms à des blocs logiques (fichiers)

... des noms à des répertoires venus du disque !

Montage de répertoire

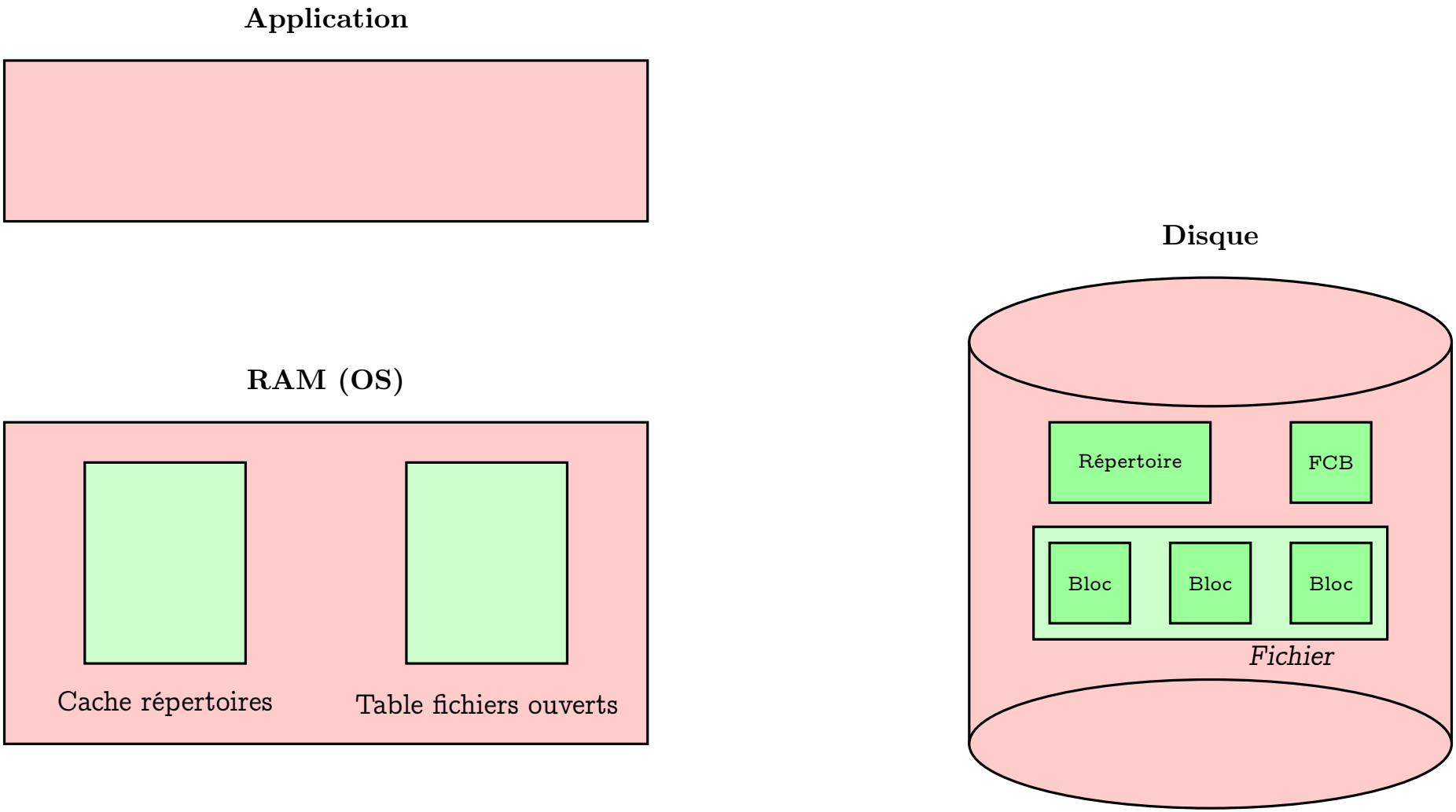
Le montage consiste à positionner un répertoire dans le système de fichier

Principe

- Chargement du FCB par l'OS (disque $\rightarrow$ RAM)
- Association dans le MFD au niveau logique

$\rightarrow$ Les fichiers deviennent accessibles

Accès à un fichier



Accès à un fichier

Application

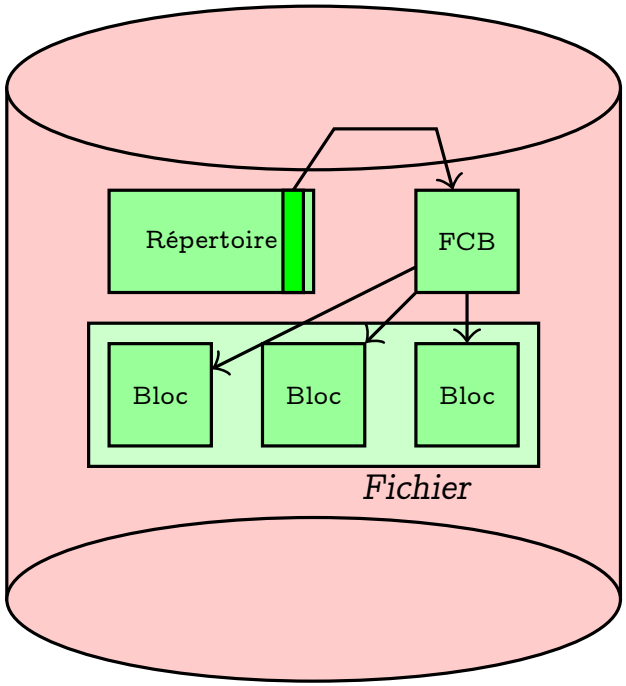
RAM (OS)

Cache répertoires

Table fichiers ouverts

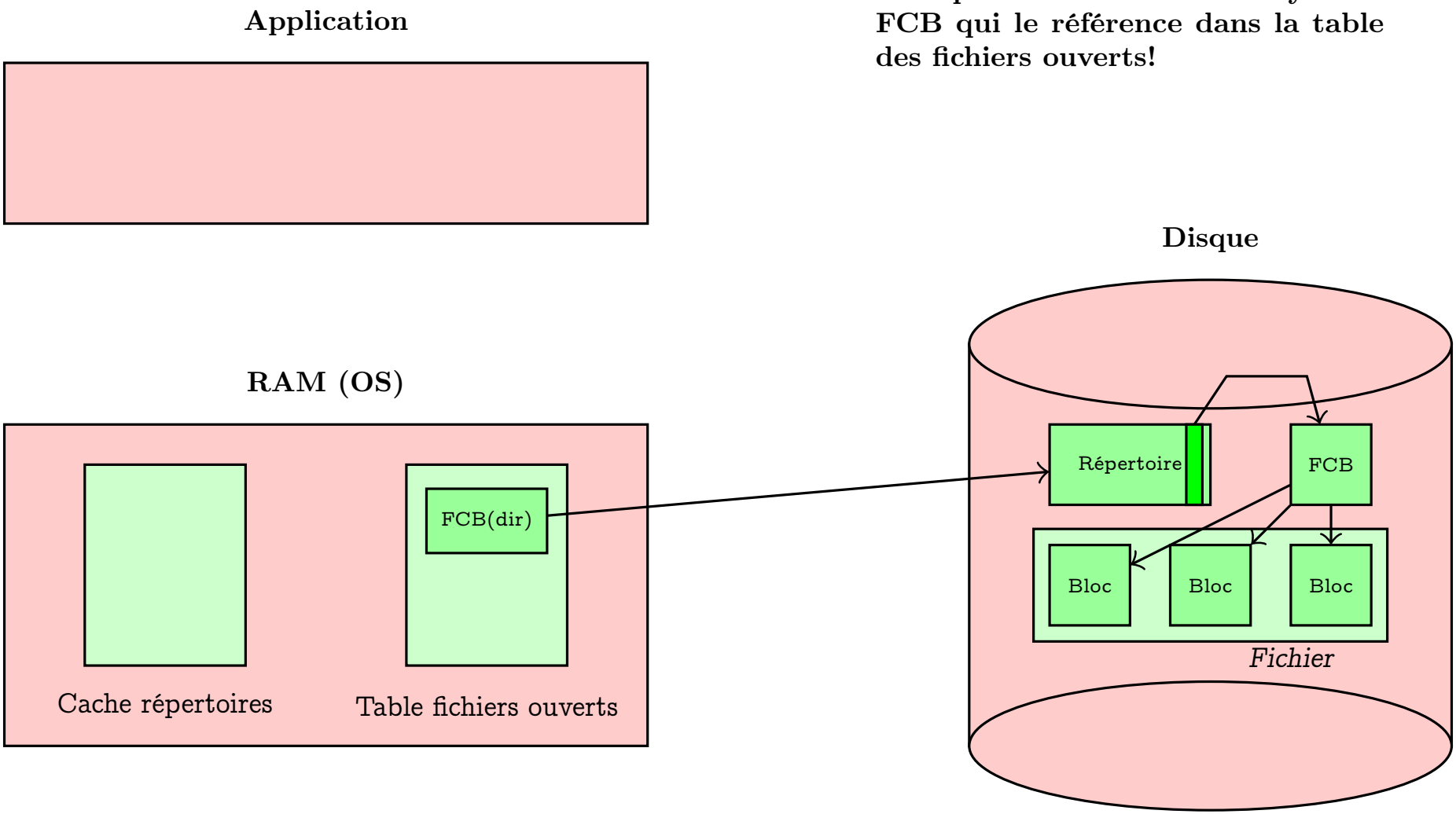
Le système de fichier (sur le disque) définit la structuration des données

Disque



Accès à un fichier

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



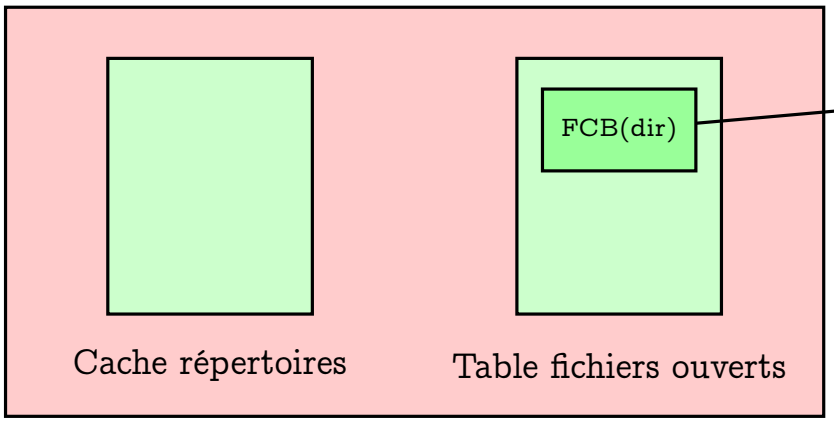
Accès à un fichier

Lors du premier accès à un fichier...

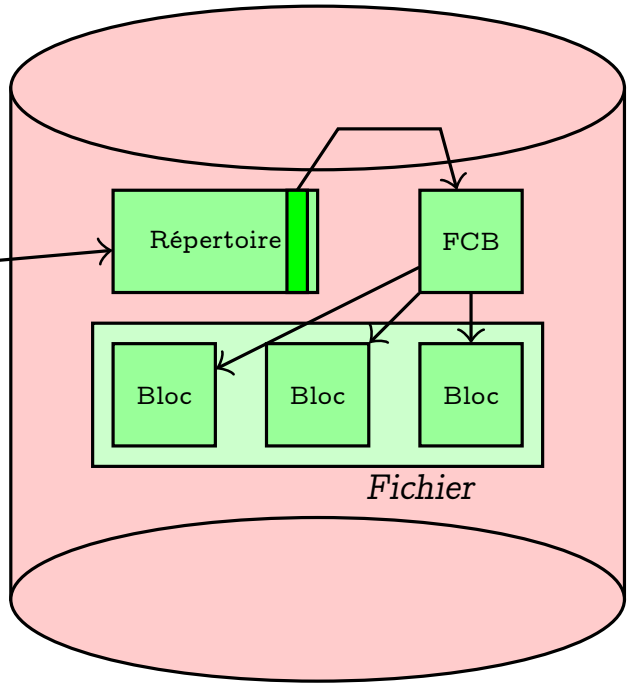
Application

```
open(dir/file.ext)
```

RAM (OS)

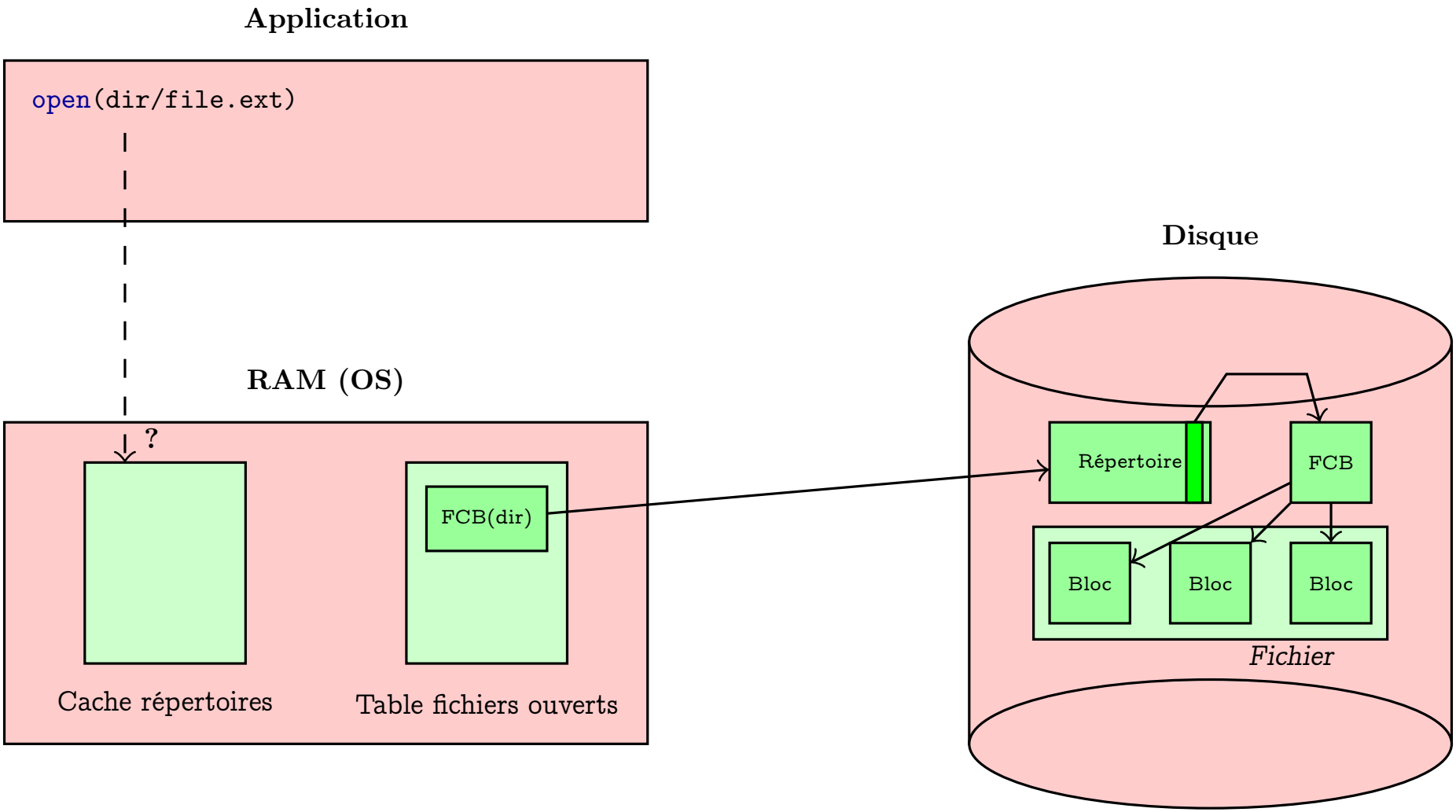


Disque



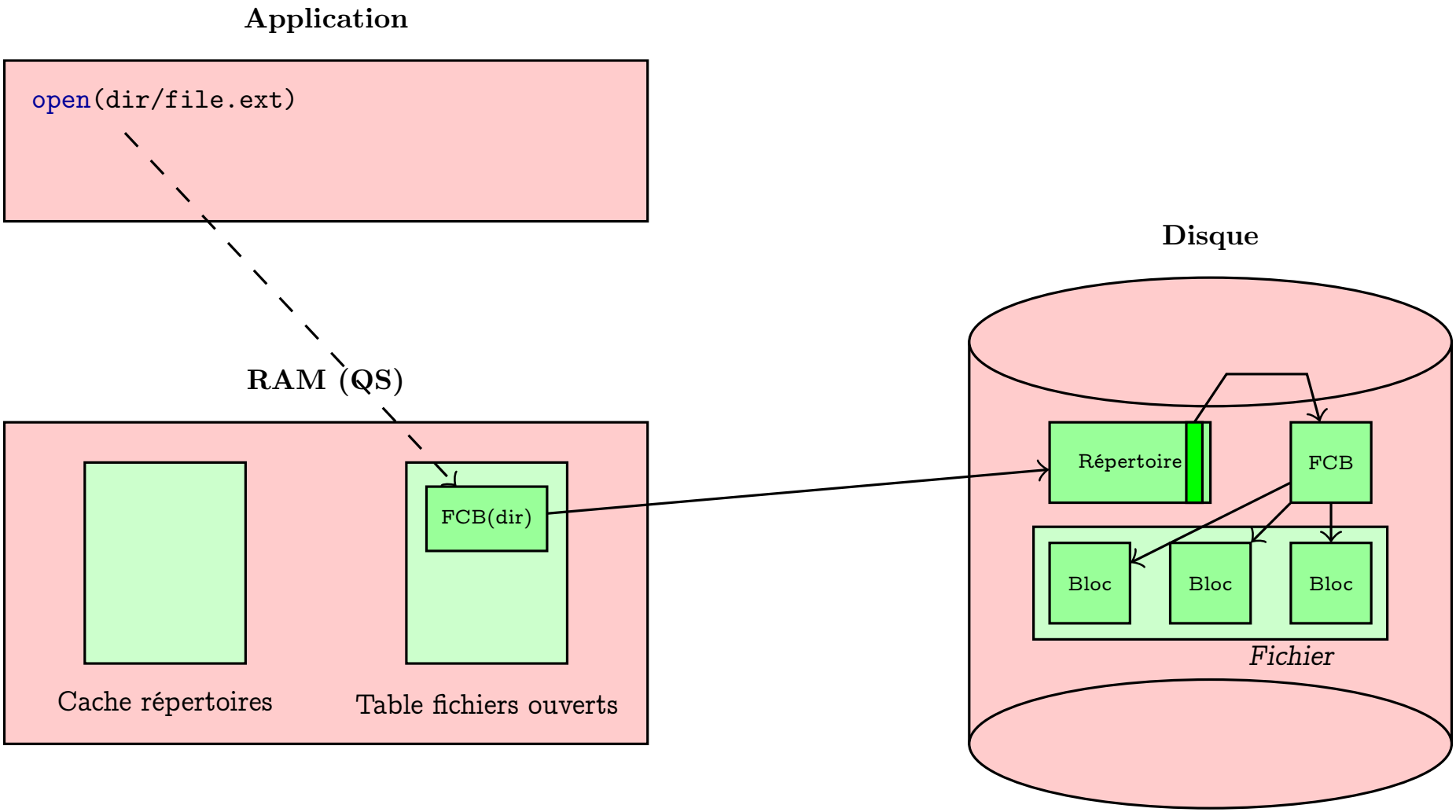
Accès à un fichier

Lors du premier accès à un fichier...



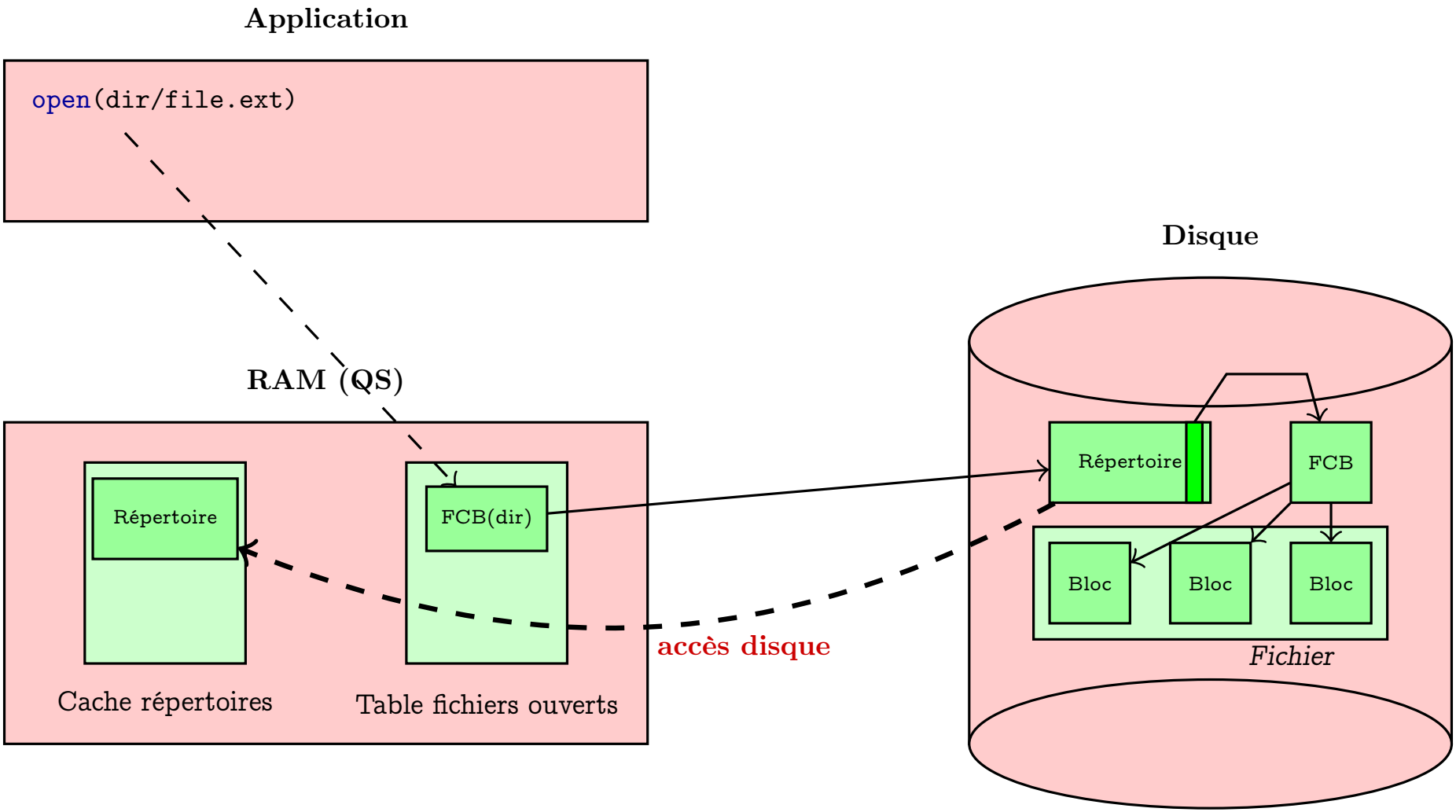
Accès à un fichier

Lors du premier accès à un fichier...



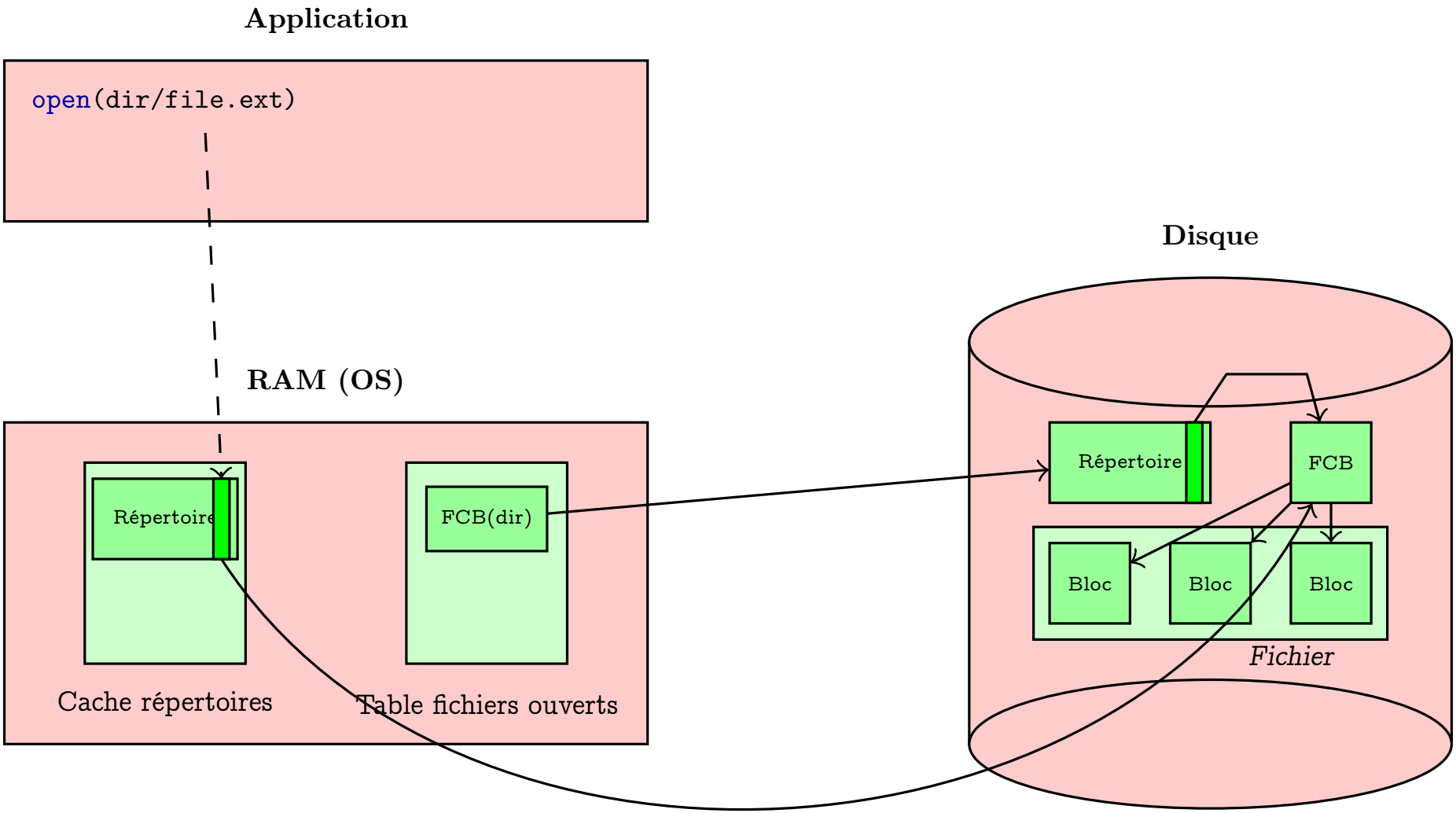
Accès à un fichier

Lors du premier accès à un fichier...



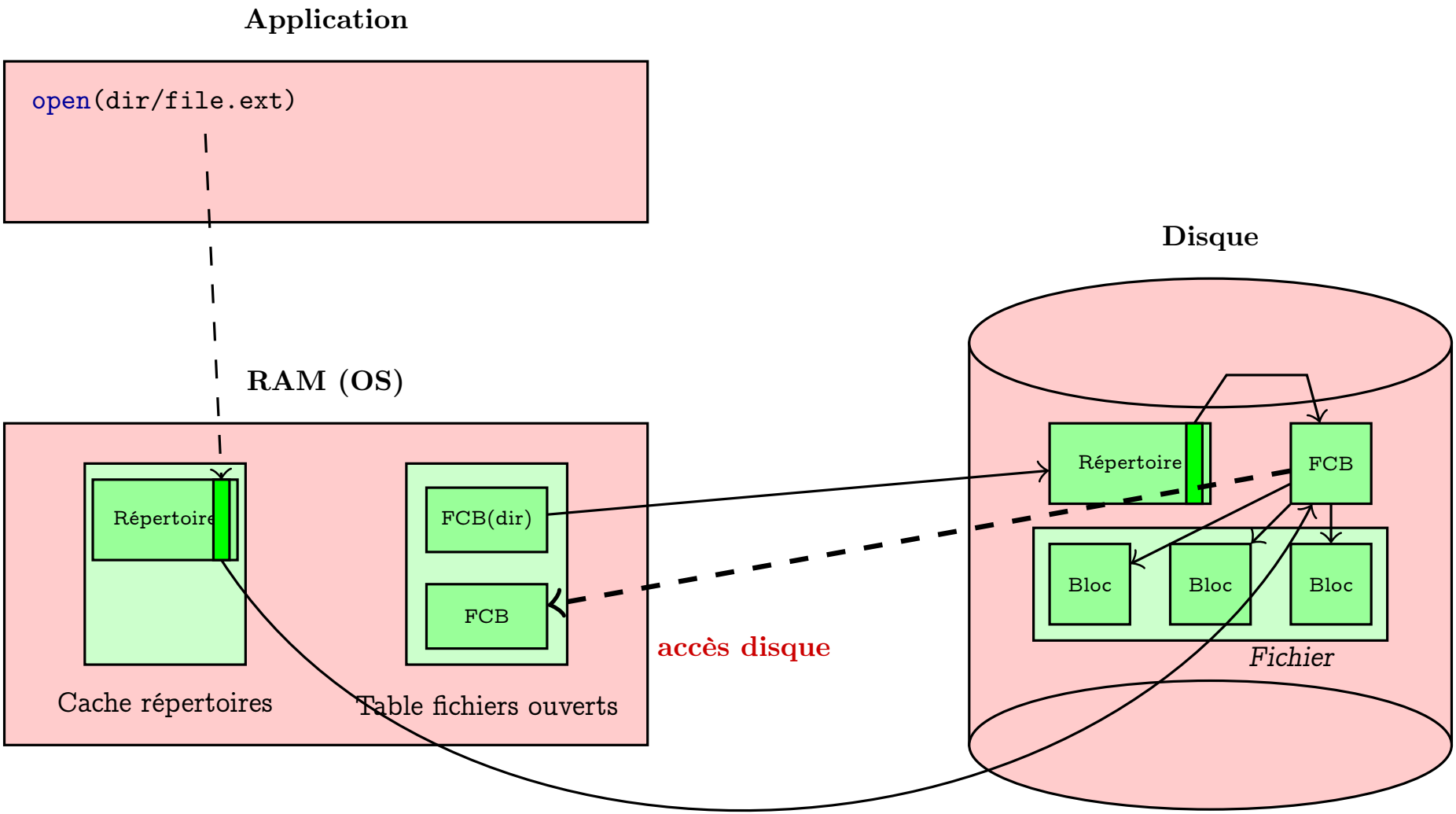
Accès à un fichier

Lors du premier accès à un fichier...



Accès à un fichier

Lors du premier accès à un fichier...

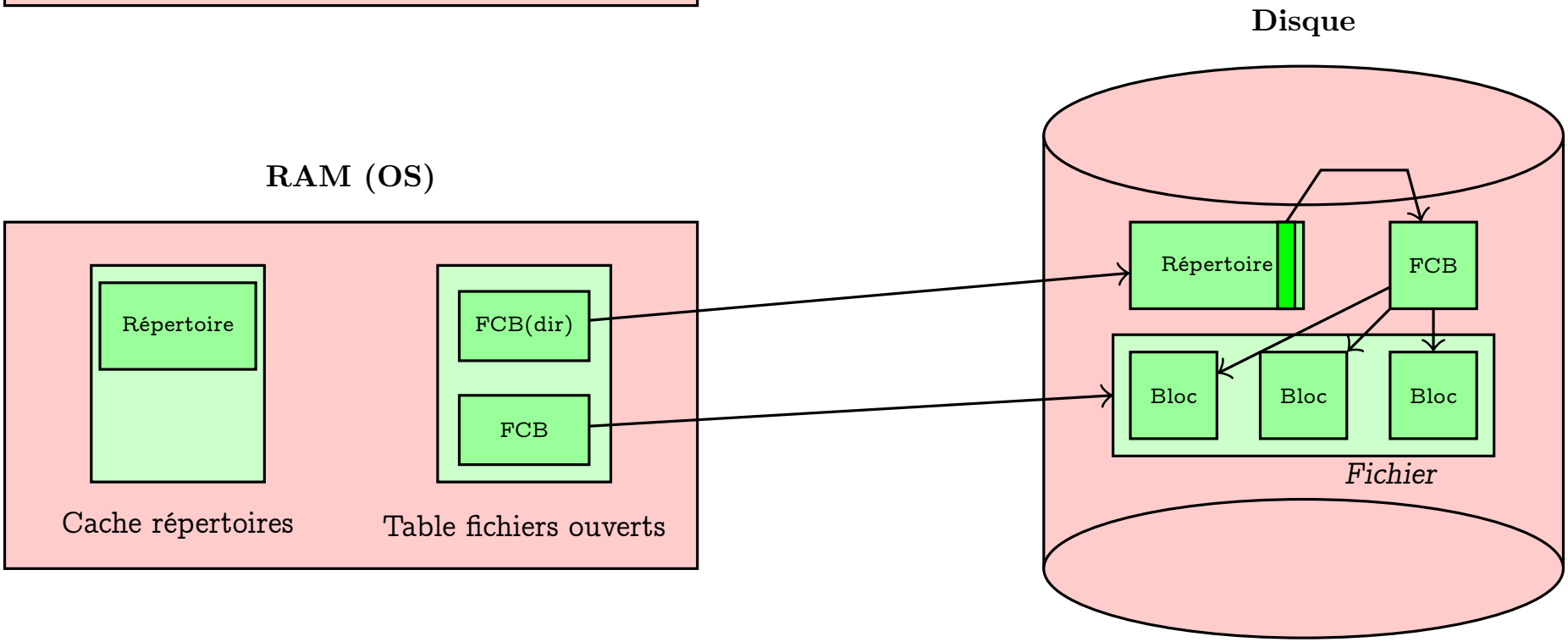


Accès à un fichier

Lors du premier accès à un fichier...

Application

```
open(dir/file.ext)
```



Accès à un fichier

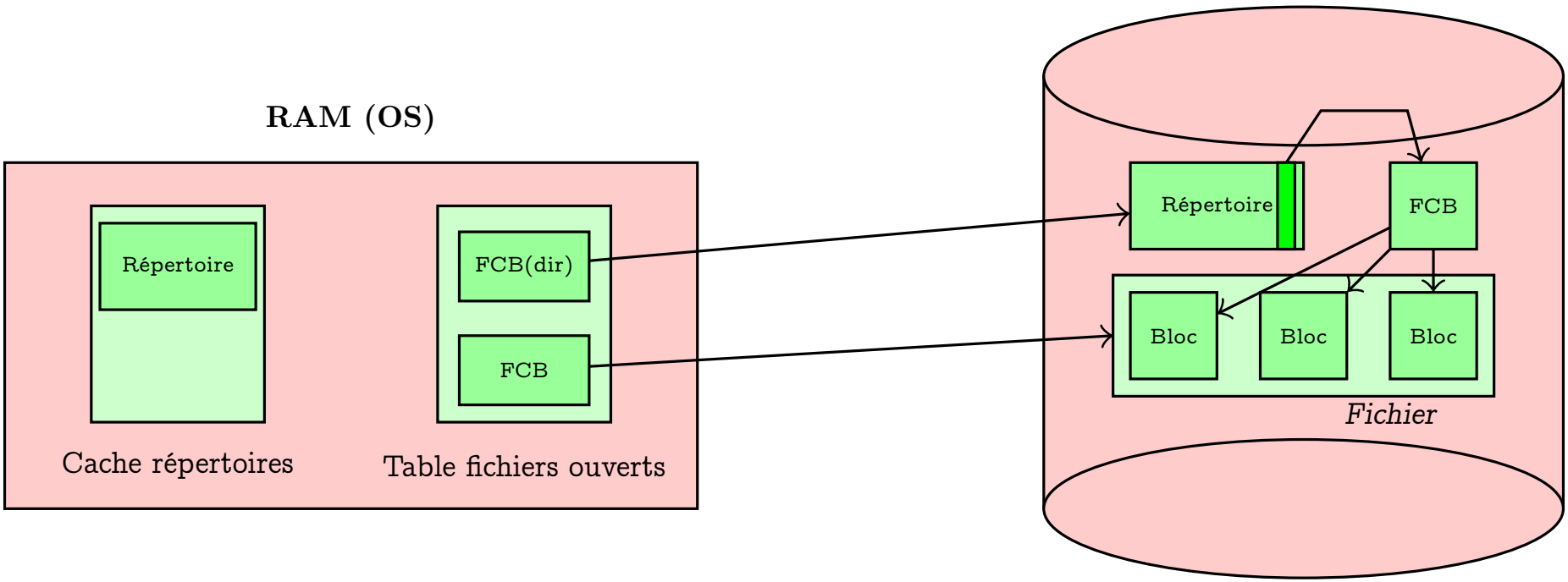
Accès au fichier ouvert...

Application

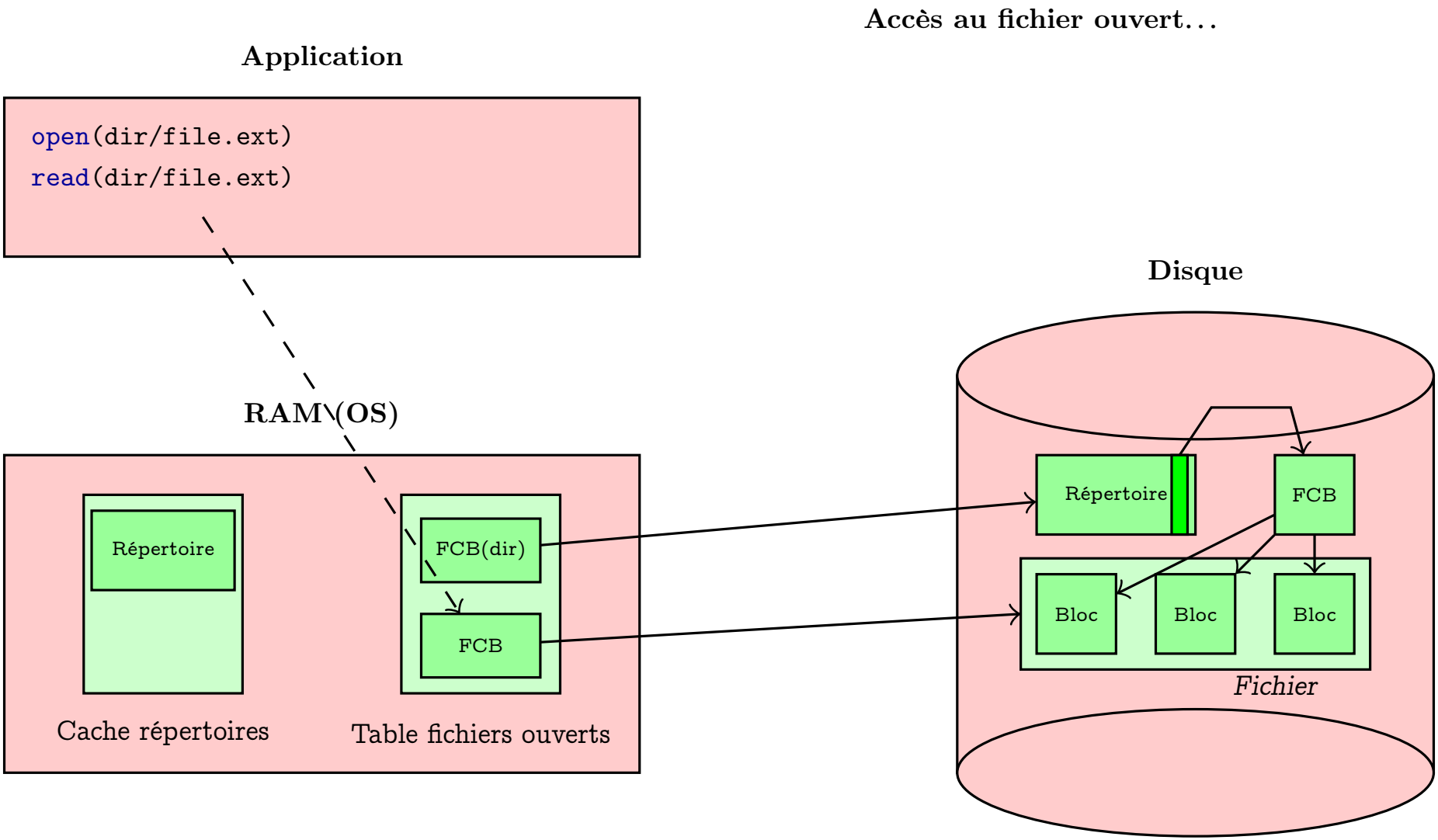
```
open(dir/file.ext)
read(dir/file.ext)
```

Disque

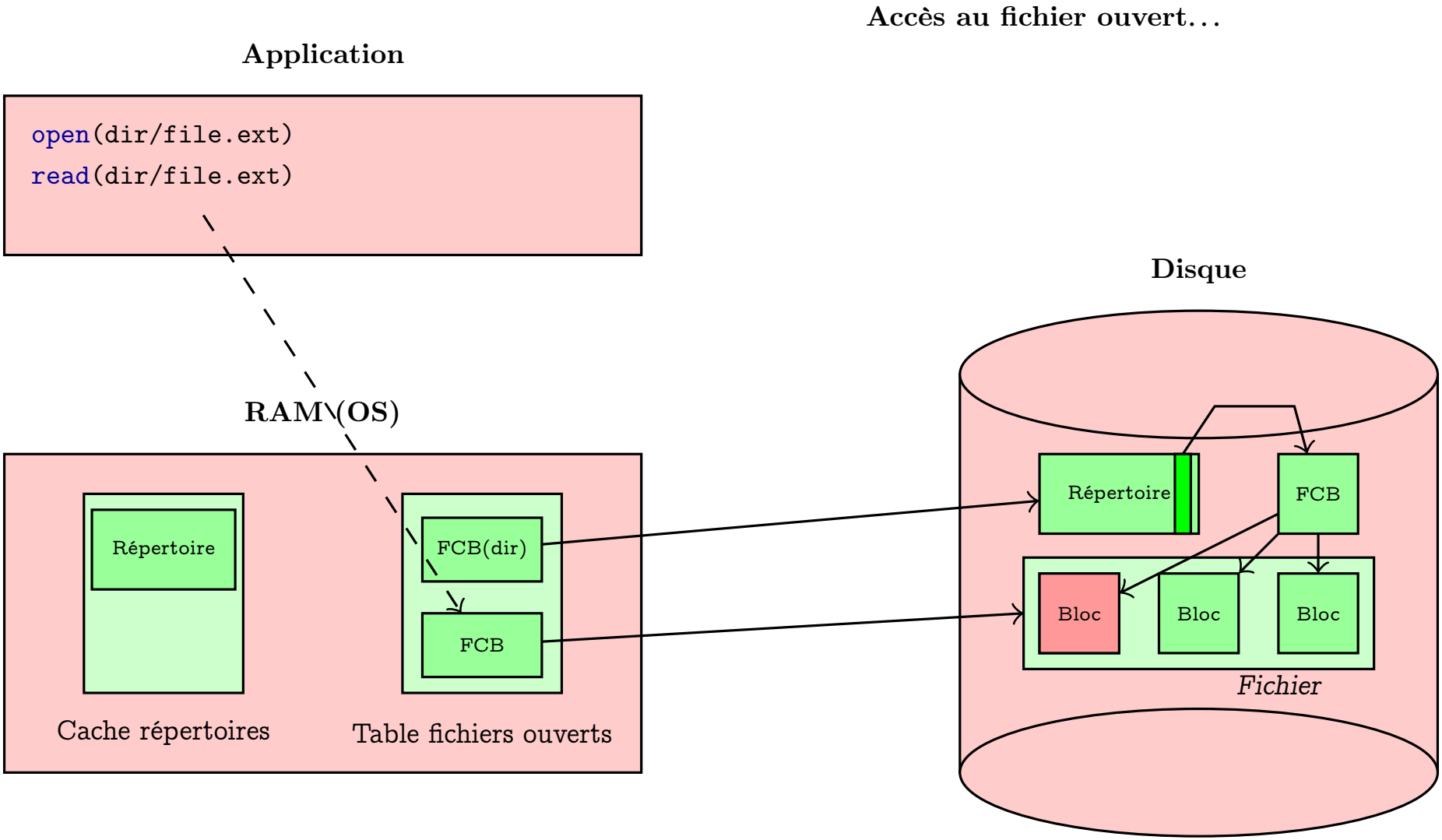
RAM (OS)



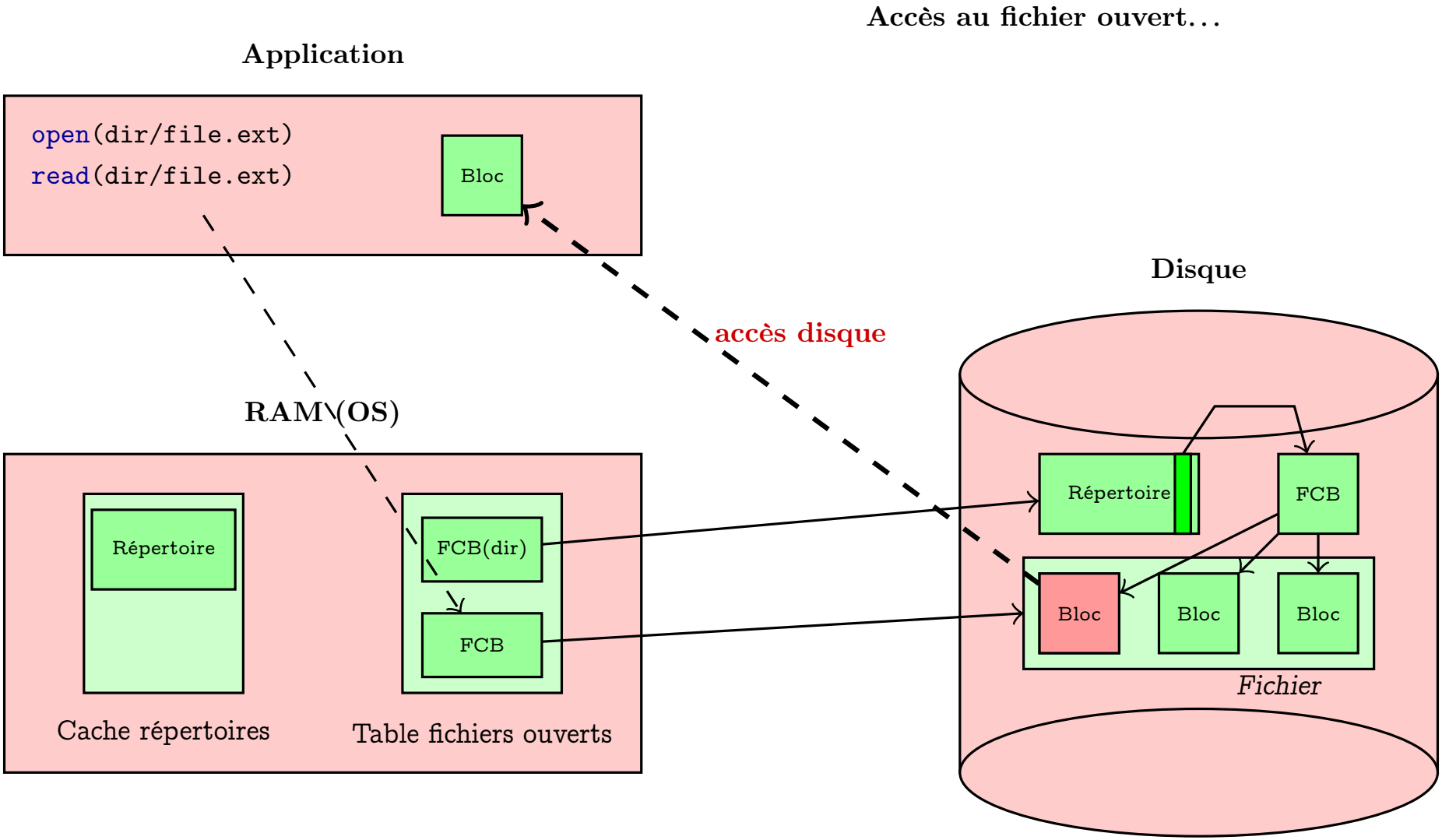
Accès à un fichier



Accès à un fichier



Accès à un fichier

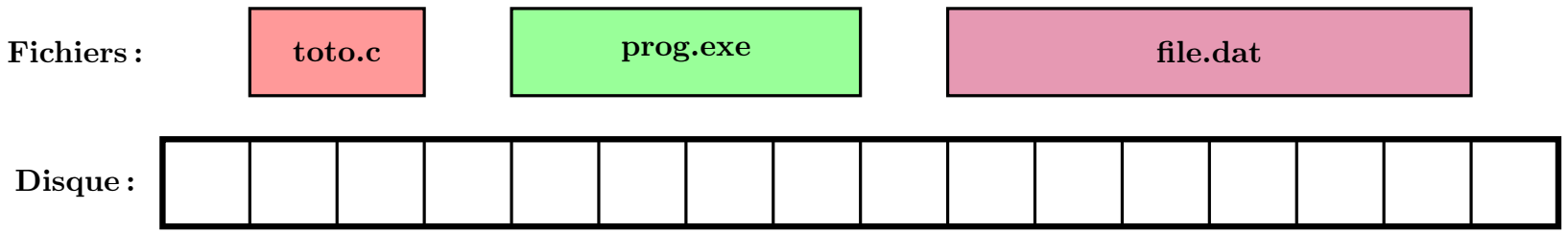


Allocation

Problème

Fichiers $\longrightarrow$ blocs logiques $\longrightarrow$ blocs physiques

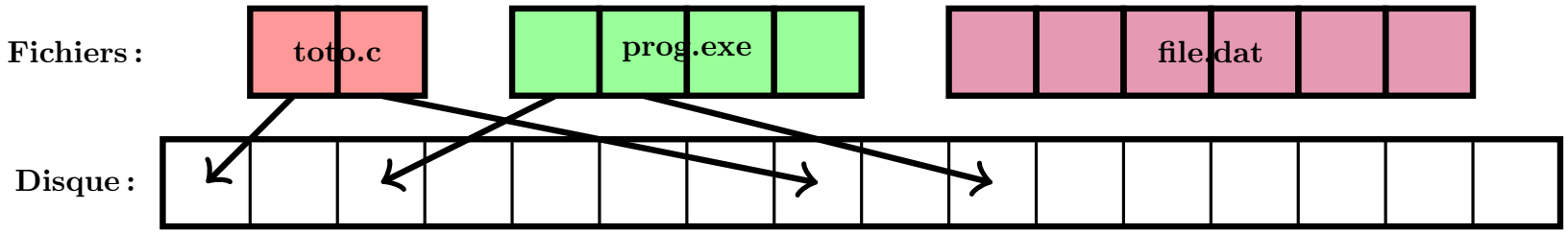
- Choix des blocs physiques pour 1 fichier donné



Problème

Fichiers $\longrightarrow$ blocs logiques $\longrightarrow$ blocs physiques

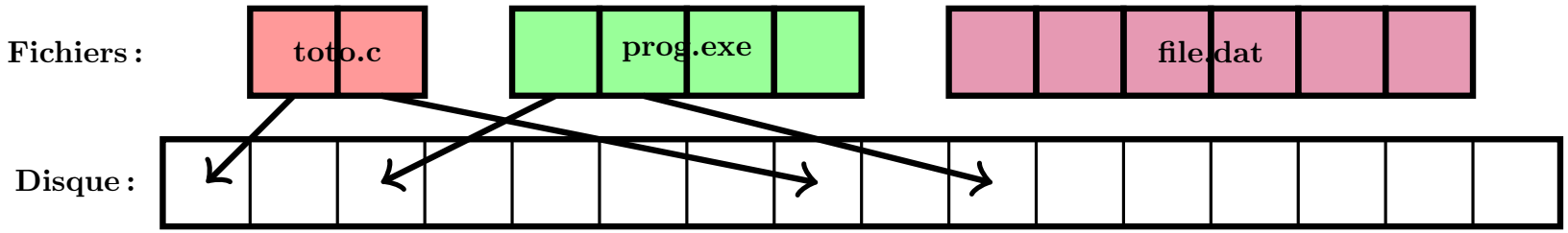
- Choix des blocs physiques pour 1 fichier donné



Problème

Fichiers $\longrightarrow$ blocs logiques $\longrightarrow$ blocs physiques

- Choix des blocs physiques pour 1 fichier donné



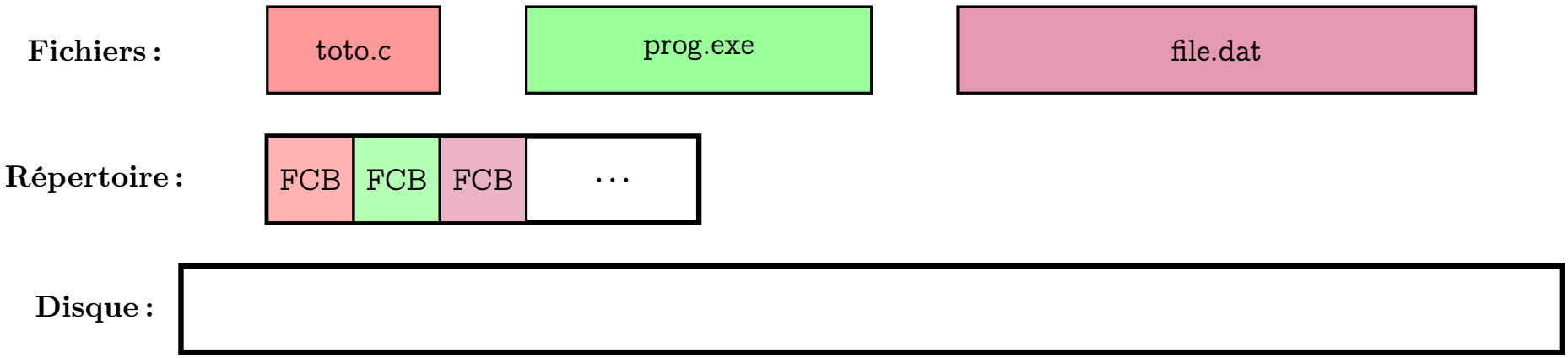
3 principales méthodes possibles

- Allocation contiguë
- Allocation chaînée
- Allocation indexée

Chaque méthode a ses avantages et inconvénients !

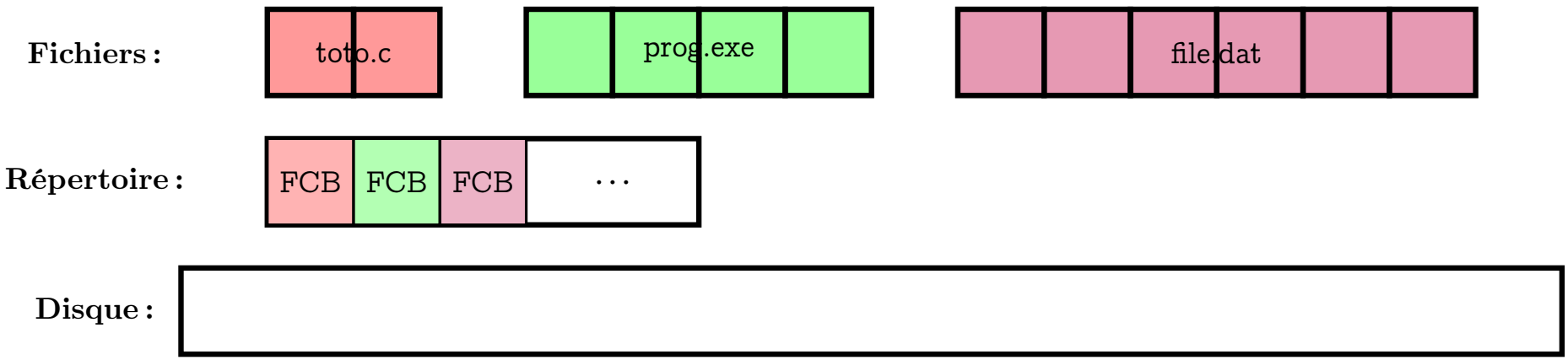
Principe

Ranger les blocs les uns derrière les autres



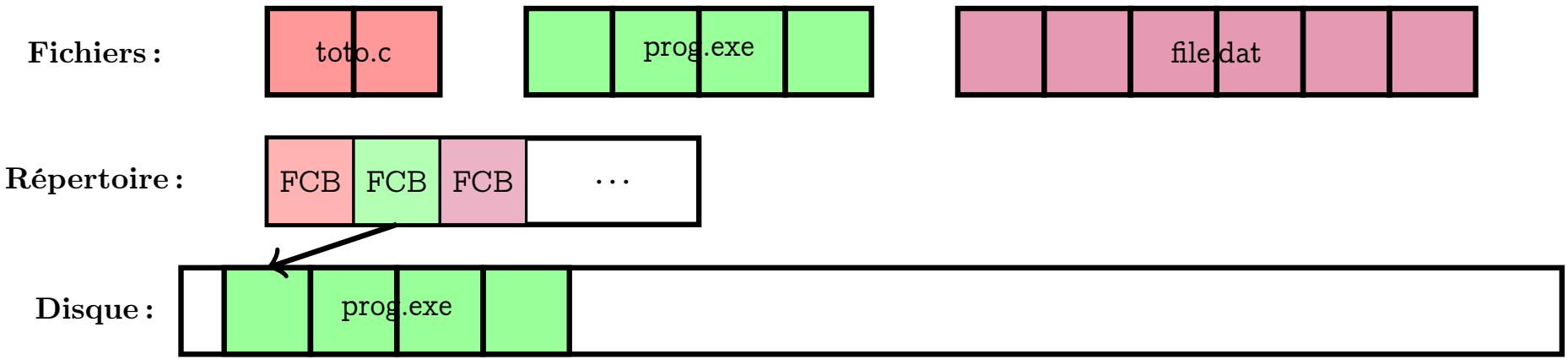
Principe

Ranger les blocs les uns derrière les autres



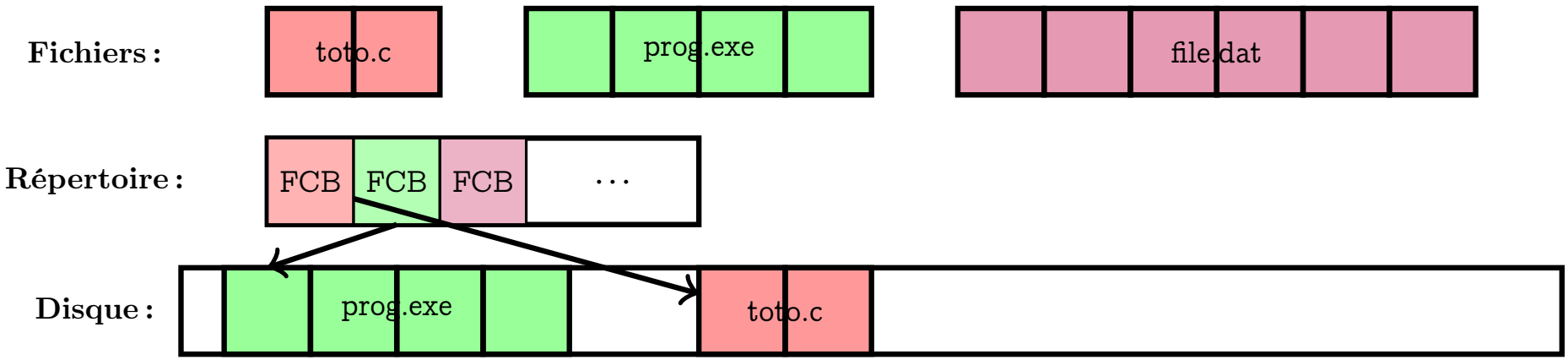
Principe

Ranger les blocs les uns derrière les autres



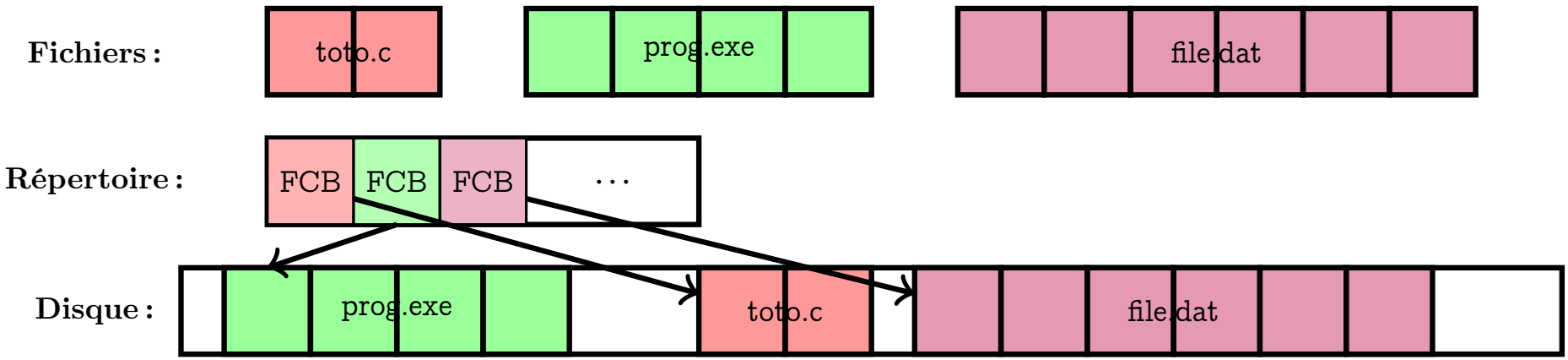
Principe

Ranger les blocs les uns derrière les autres



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

Allocation contiguë

Principe

Ranger les blocs les uns derrière les autres

Avantages

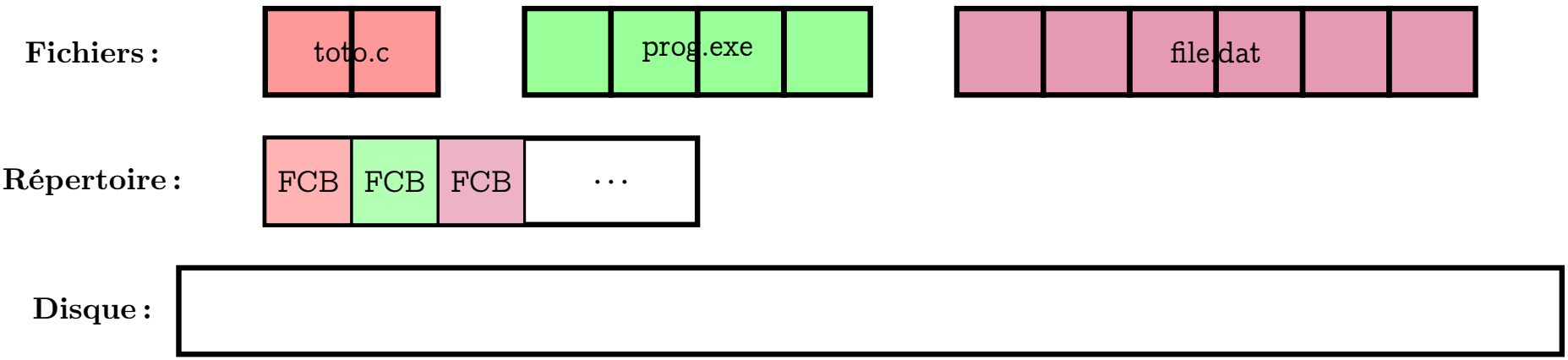
- ✓ Accès au bloc suivant : aucun coût
- ✓ FCB : adresse bloc départ + taille

Inconvénients

- X Fragmentation (compactage coûteux)
- X Connaître à l'avance la taille des fichiers
- X Recherche d'espace libre coûteux
- X Stratégies d'allocation à définir

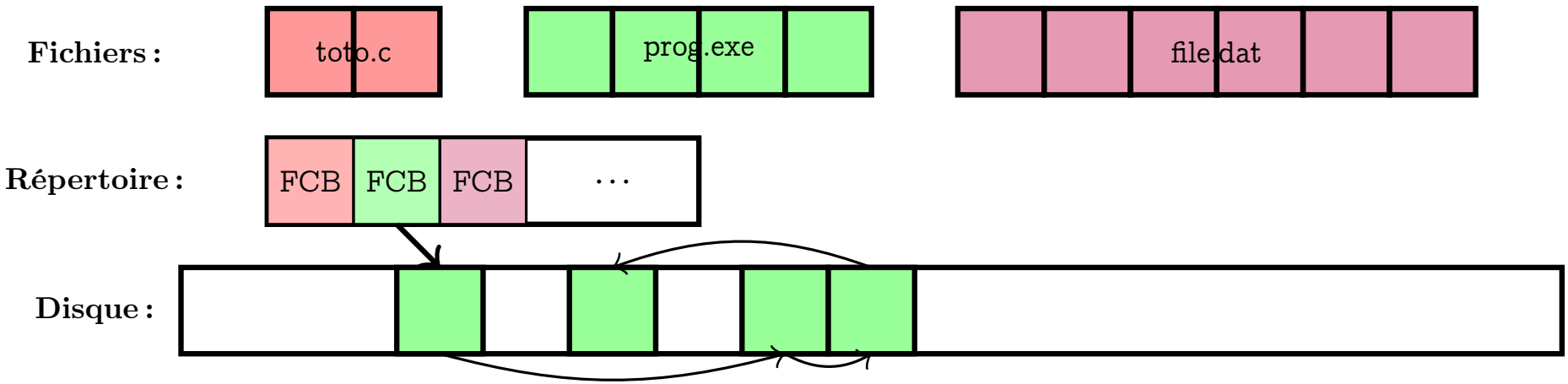
Principe

Fichier = liste chaînée de blocs



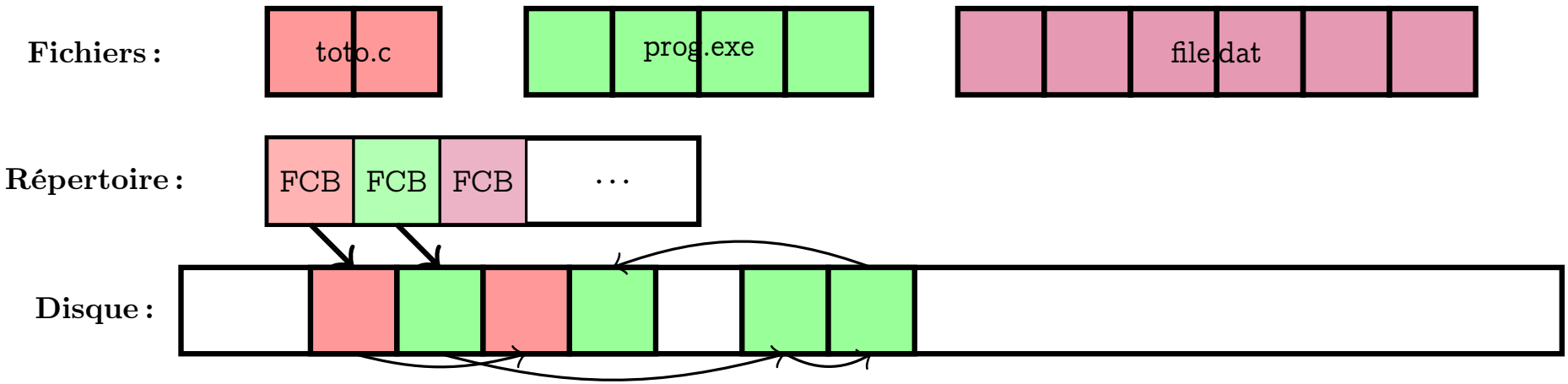
Principe

Fichier = liste chaînée de blocs



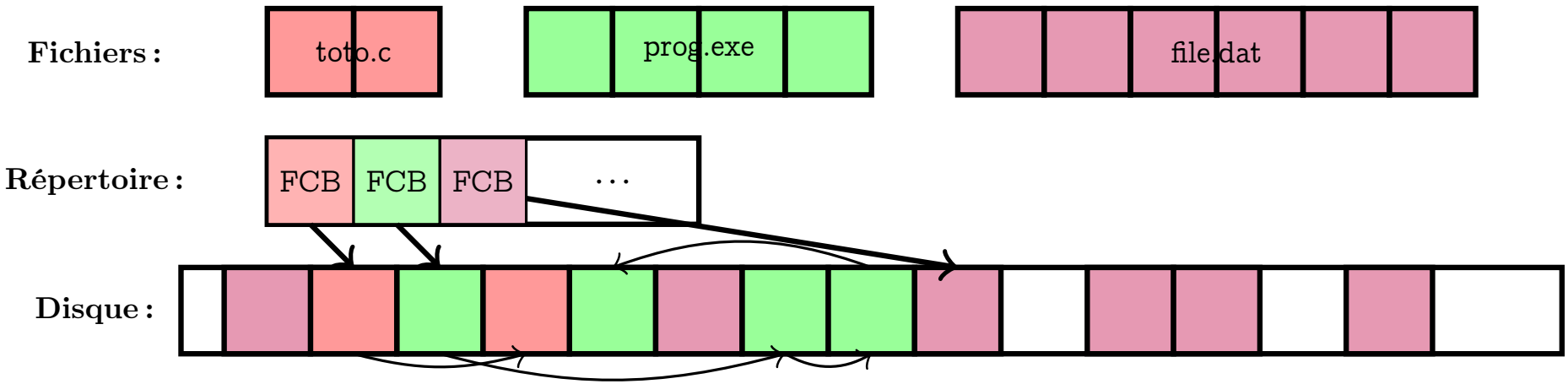
Principe

Fichier = liste chaînée de blocs



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

Allocation chaînée

Principe

Fichier = liste chaînée de blocs

Avantages

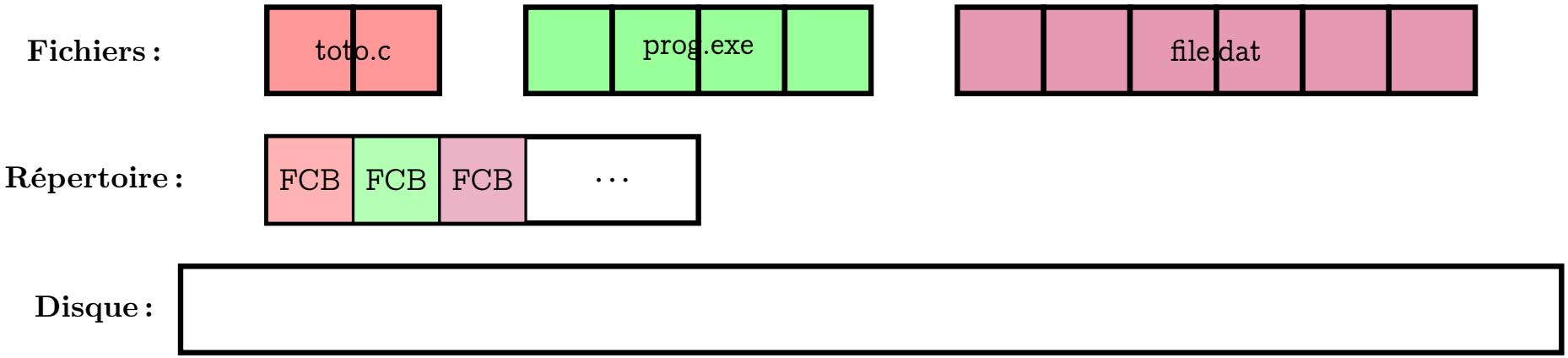
- ✓ FCB : adresse premier et dernier blocs
- ✓ Pas de fragmentation
- ✓ Fichiers taille quelconque

Inconvénients

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

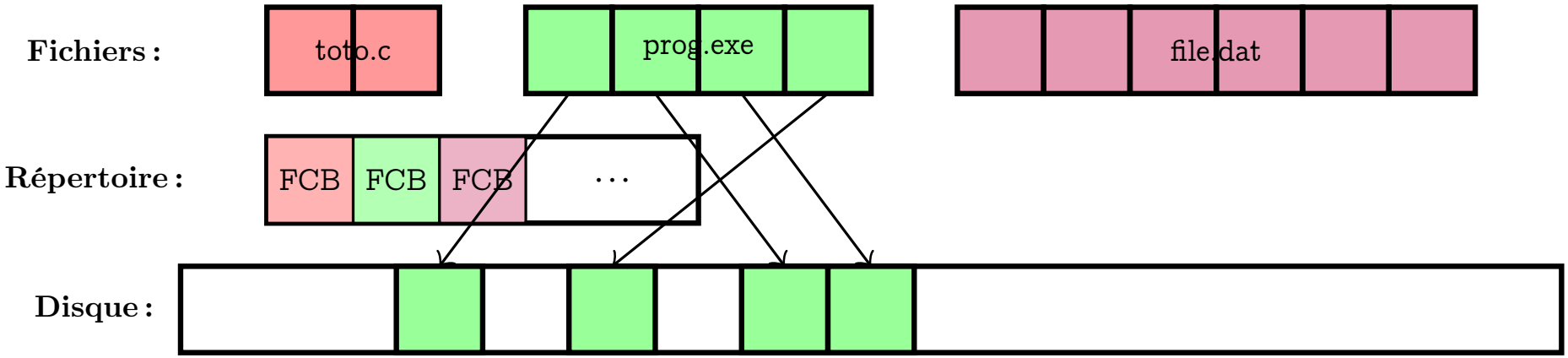
Principe

Rassembler tous les pointeurs dans un bloc d'index



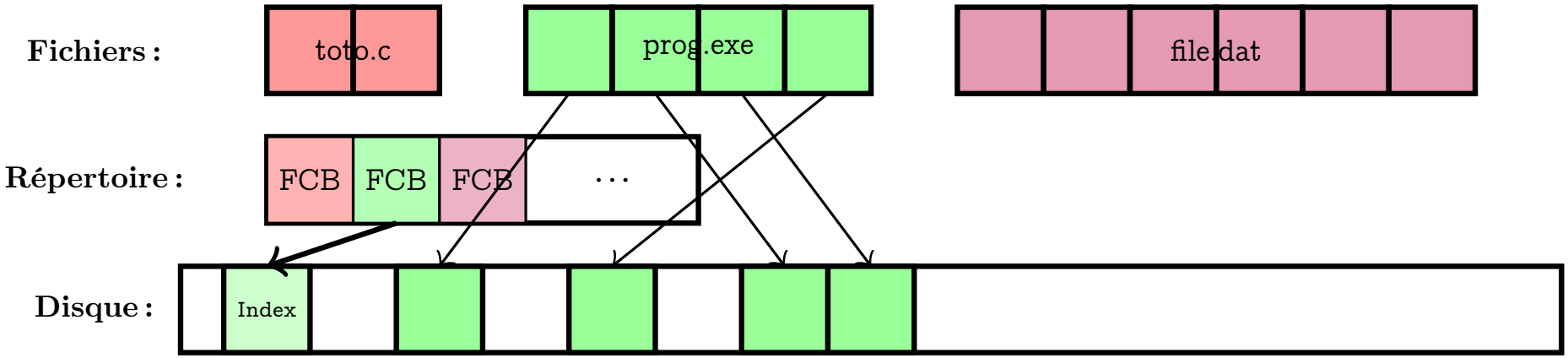
Principe

Rassembler tous les pointeurs dans un bloc d'index



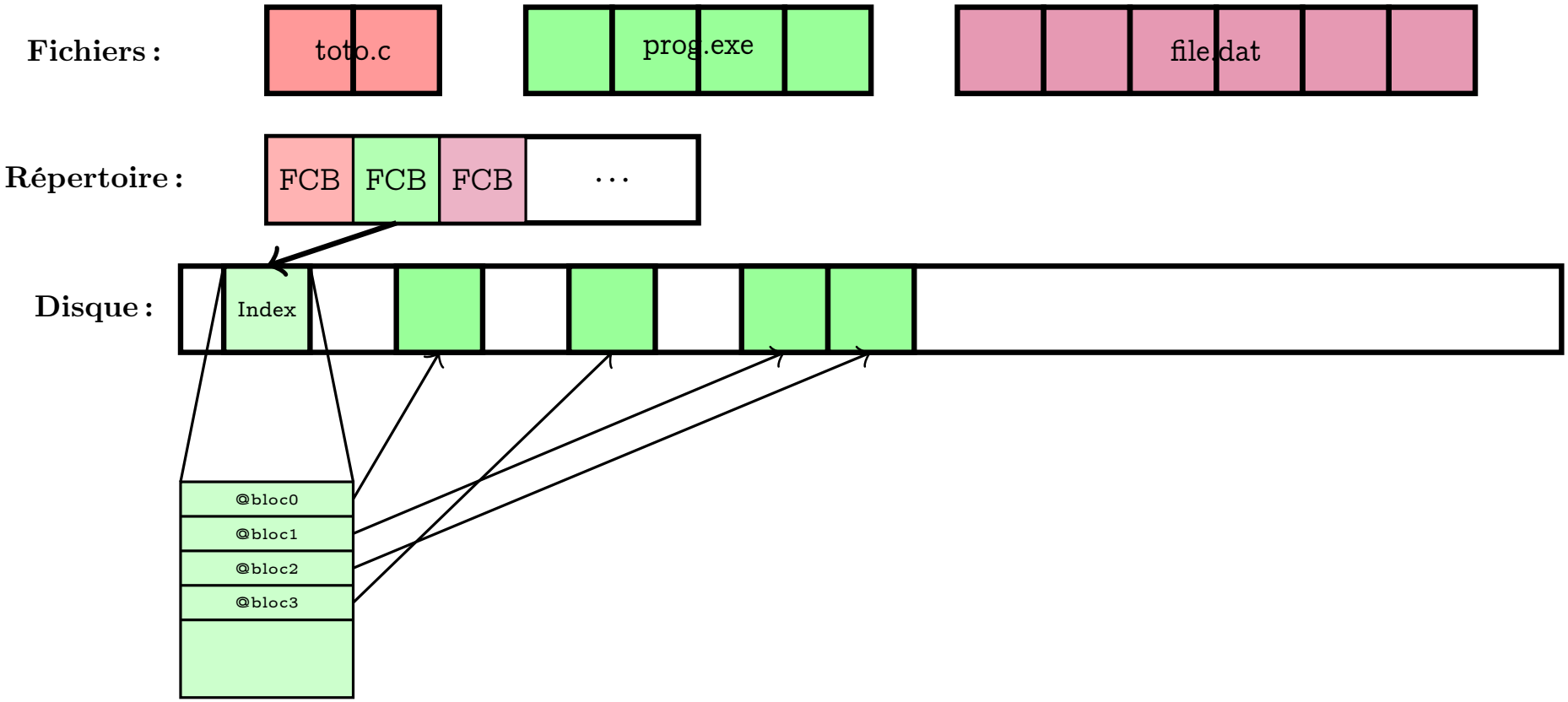
Principe

Rassembler tous les pointeurs dans un bloc d'index



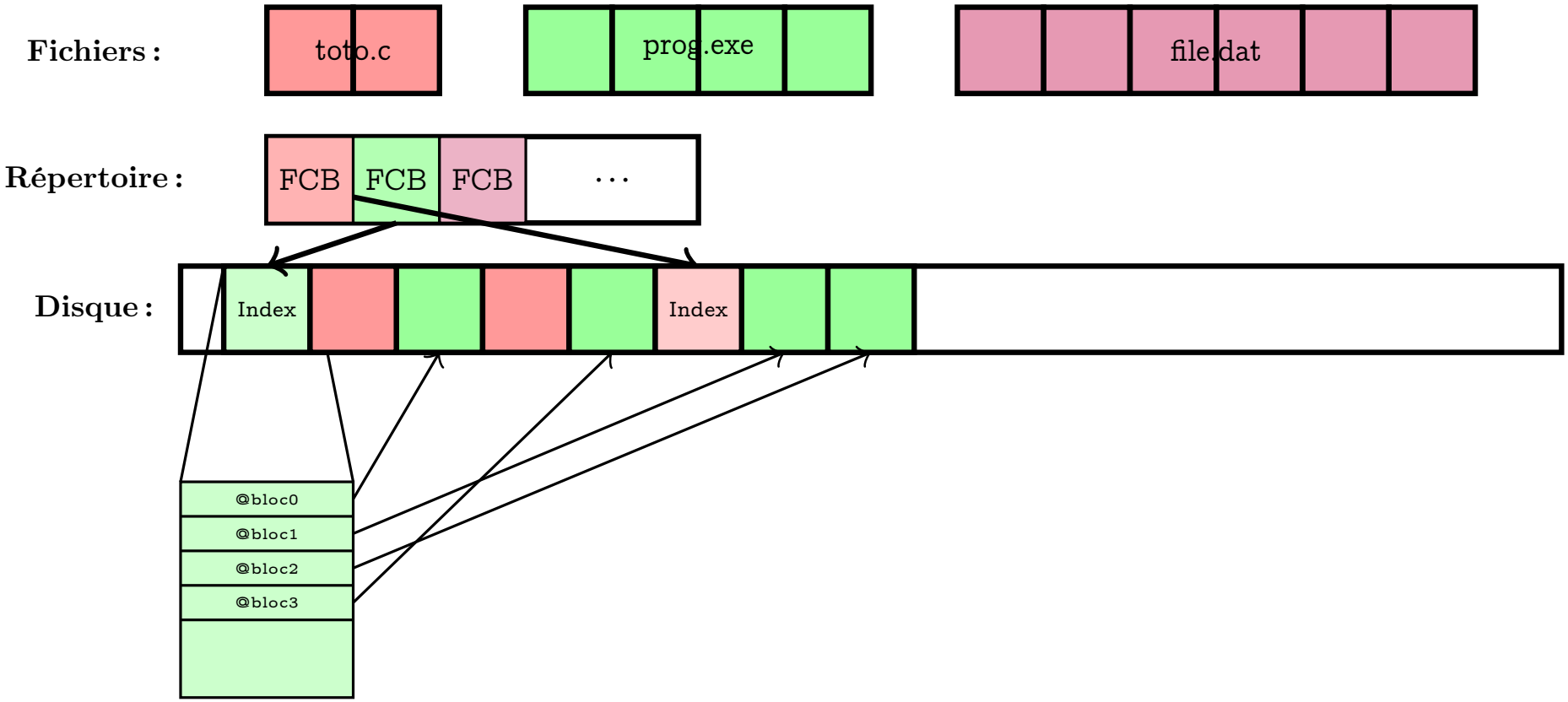
Principe

Rassembler tous les pointeurs dans un bloc d'index



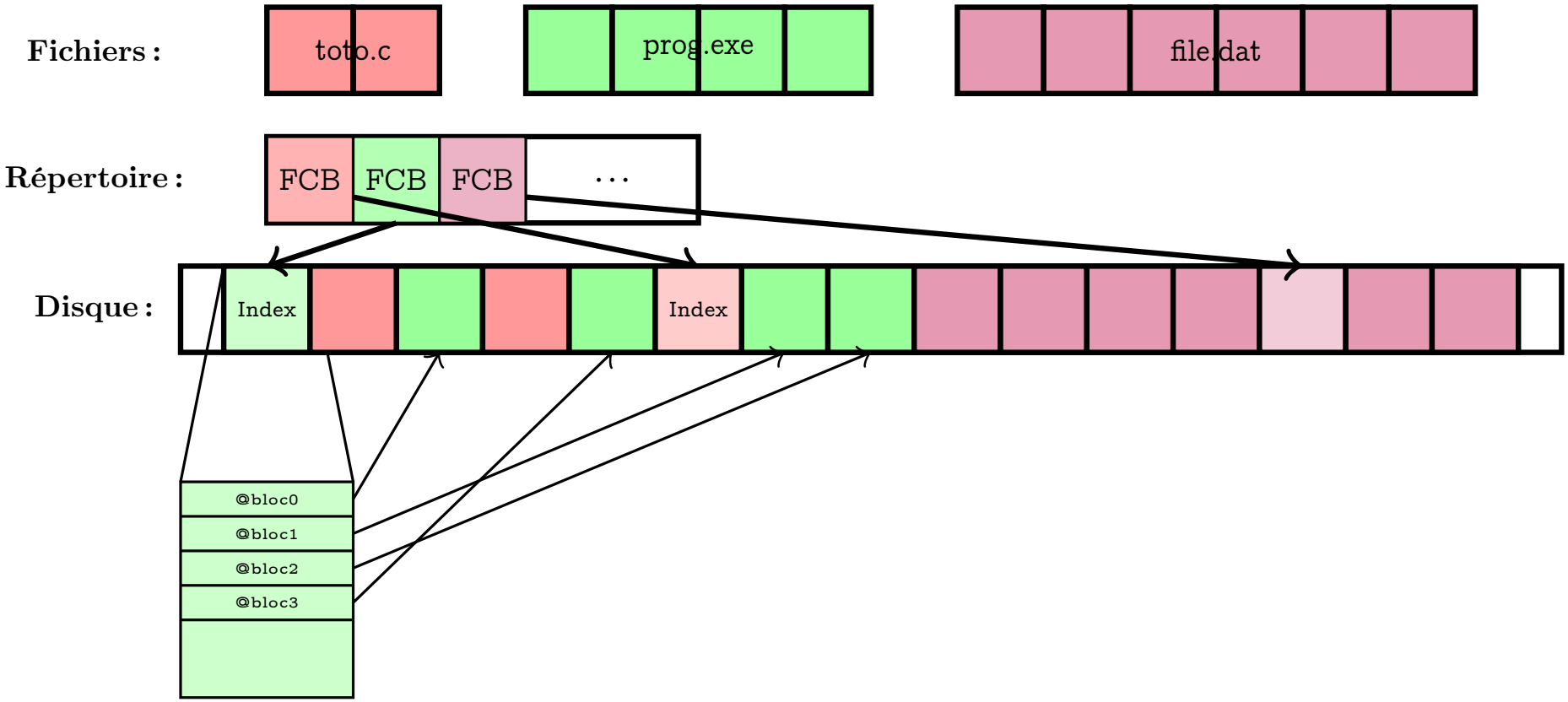
Principe

Rassembler tous les pointeurs dans un bloc d'index



Principe

Rassembler tous les pointeurs dans un bloc d'index



Allocation indexée

Principe

Rassembler tous les pointeurs dans un bloc d'index

Avantages

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

Allocation indexée

Principe

Rassembler tous les pointeurs dans un bloc d'index

Avantages

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

Inconvénients

- X Taille fichier limitée par taille bloc

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

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512Kio de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse = ?

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512Kio de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc = ?

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512Kio de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs
- Taille max d'un fichier $=$?

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

- 512Kio de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs
- Taille max d'un fichier $=$ 4 blocs $=$ 32o

Problème

Fichiers nécessitant plus d'un bloc d'index

Exemple :

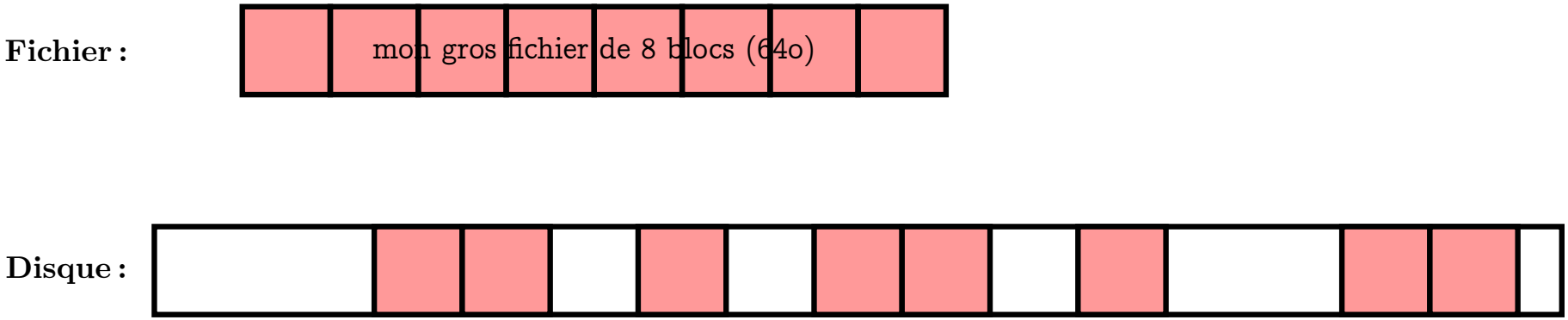
- 512Kio de disque, blocs de 8o (exemple non réaliste !)
- Taille de l'adresse $= 2^{19} \div 2^3 = 2^{16} =$ 16 bits
- Nombre max d'index/bloc $= 8 \div 2 =$ 4 blocs
- Taille max d'un fichier $=$ 4 blocs $=$ 32o

Un peu petit...

Solution 1 : liste chaînée de blocs d'index

Principe

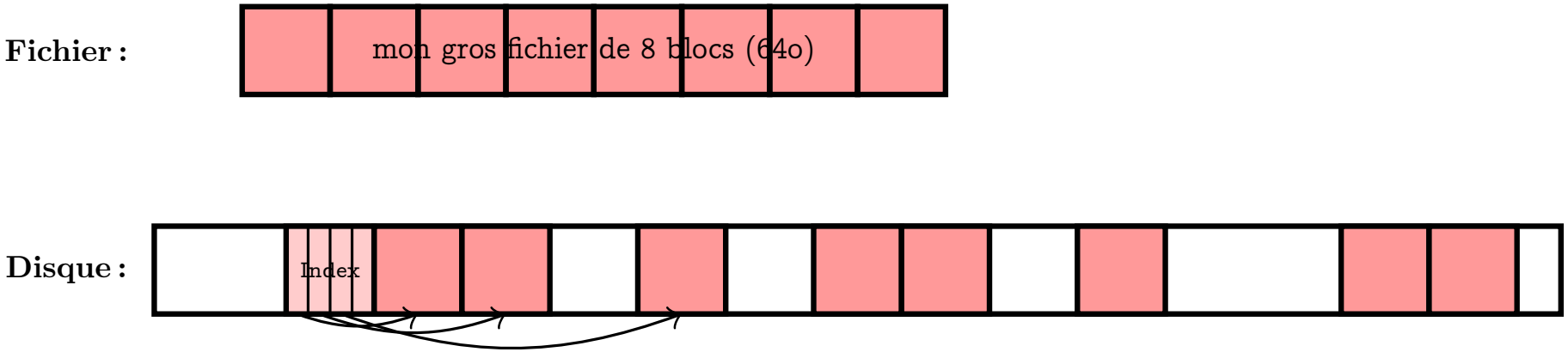
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

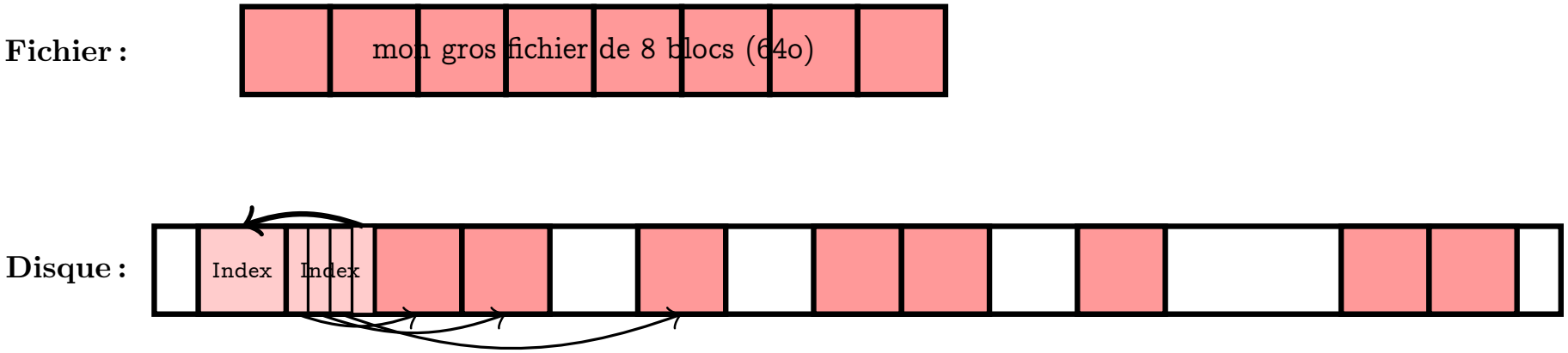
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

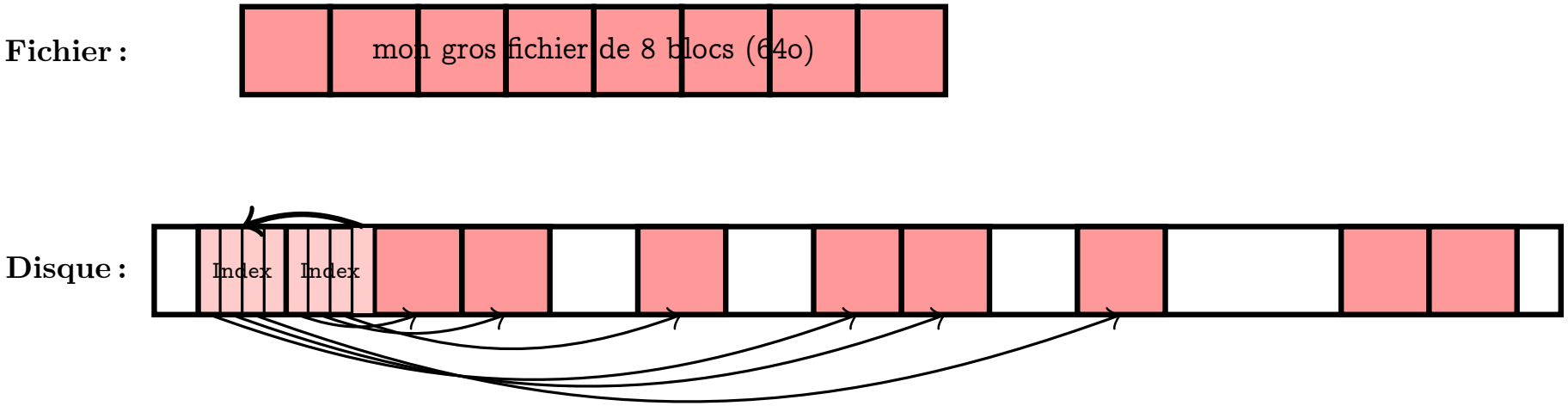
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



Solution 1 : liste chaînée de blocs d'index

Principe

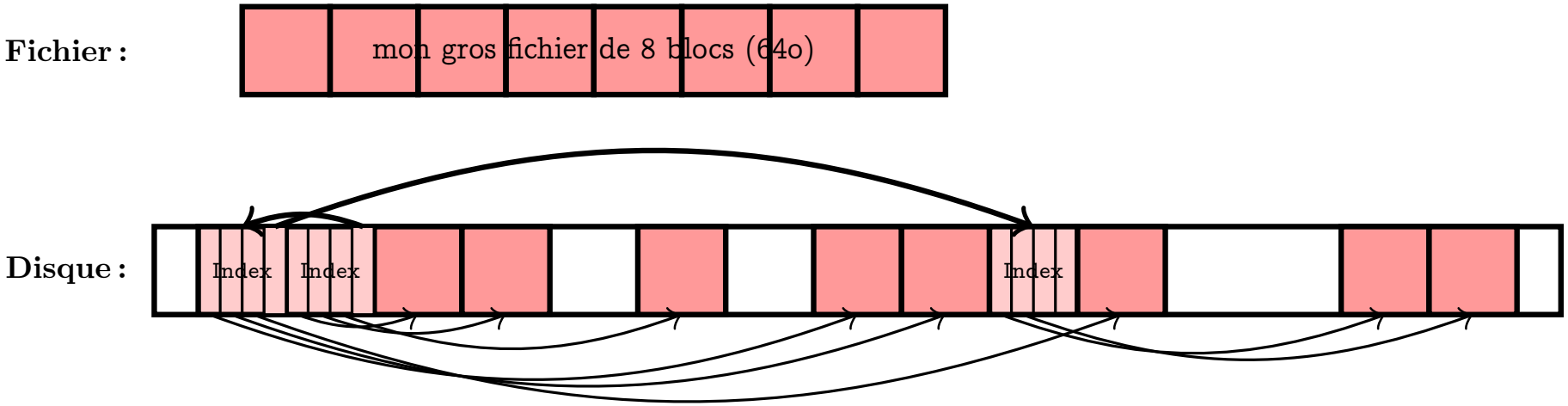
- Le répertoire pointe vers un bloc d'index
- Chaque bloc d'index se termine par un pointeur vers un autre bloc d'index



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

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

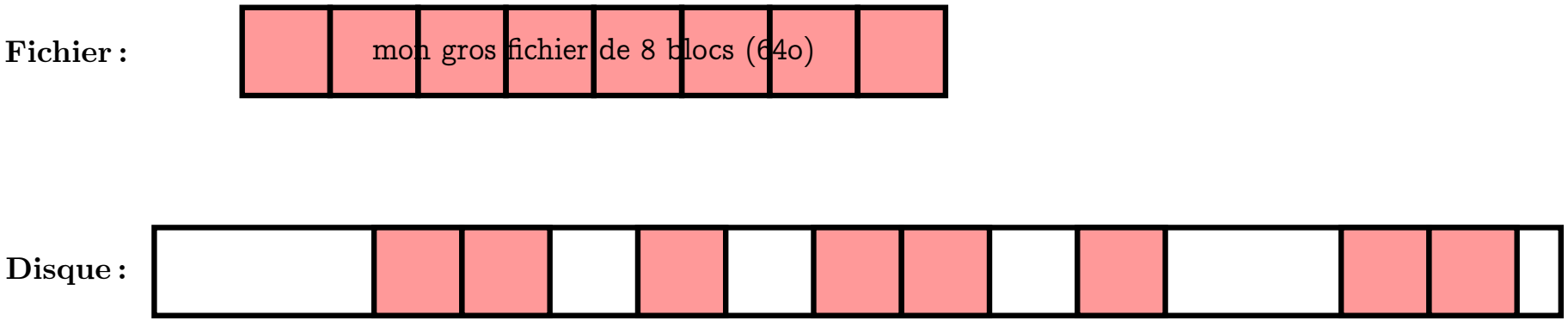
X N blocs perdus par fichier

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

Solution 2 : indexation à plusieurs niveaux

Principe

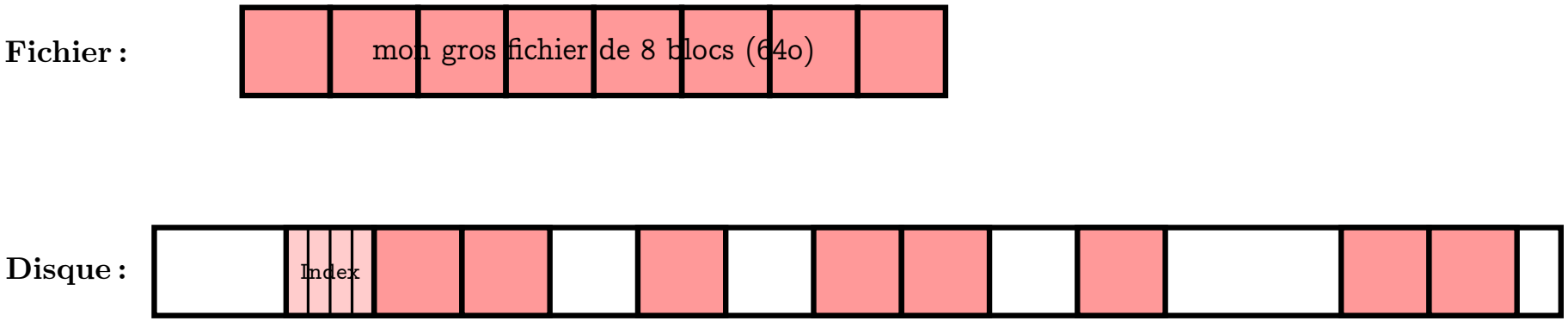
- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

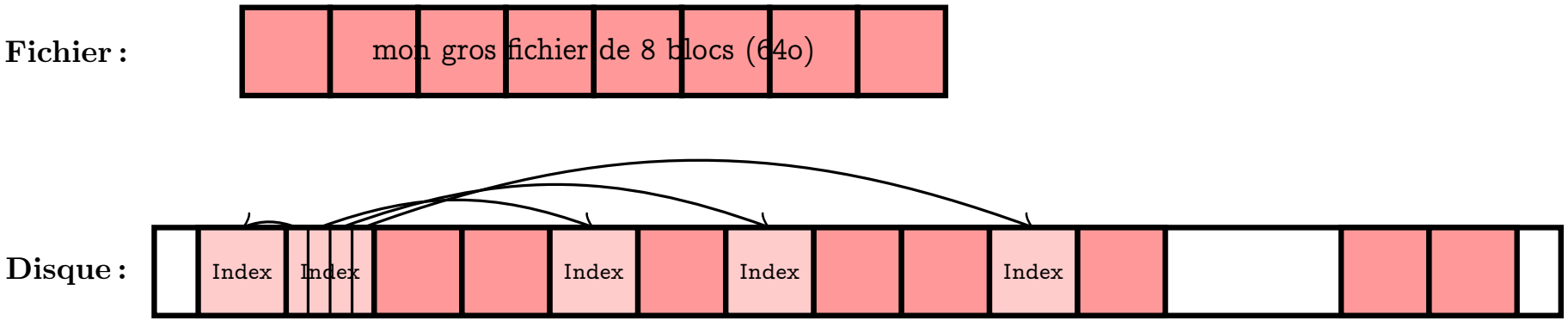
- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



Solution 2 : indexation à plusieurs niveaux

Principe

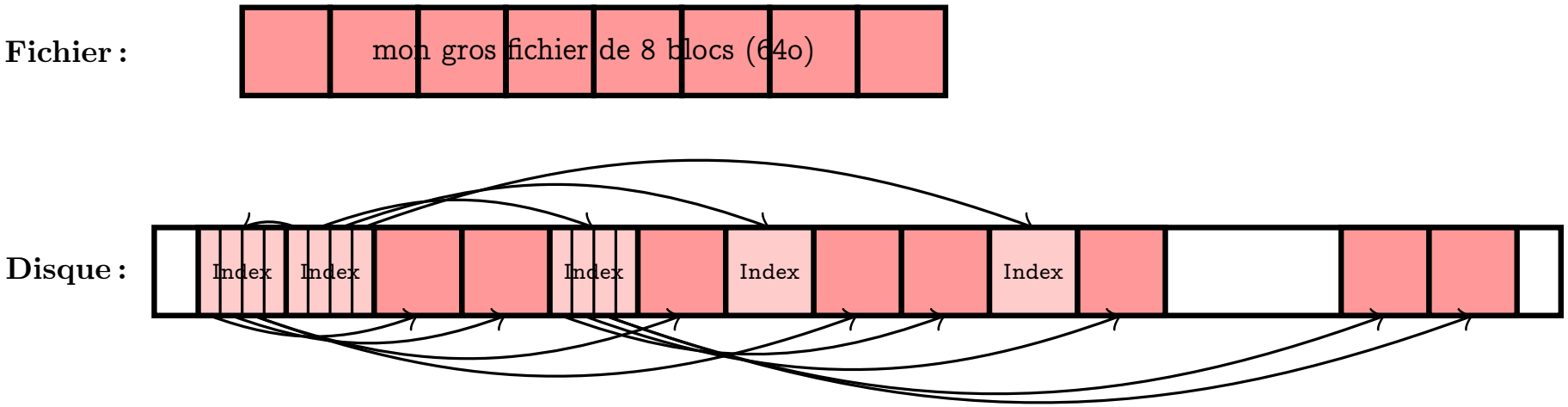
- Le répertoire pointe vers un bloc d'index *maître*
- Le bloc maître pointe vers des blocs d'index
- Éventuellement, indexation sur 3 niveaux (n^3)



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
- 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)

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

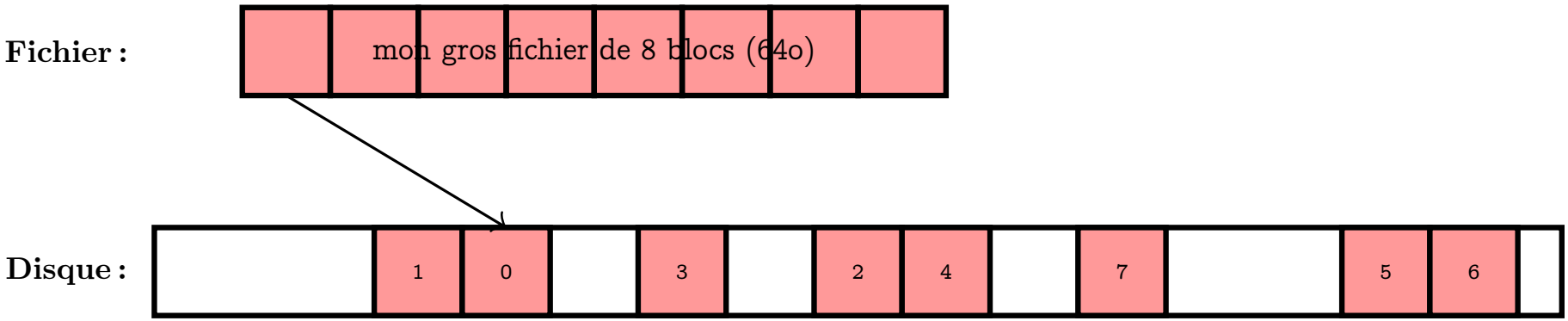
X $\geqslant$ 2 blocs perdus par fichier

(même si on ne crée pas les index en trop)

File Allocation Table (FAT)

Principe

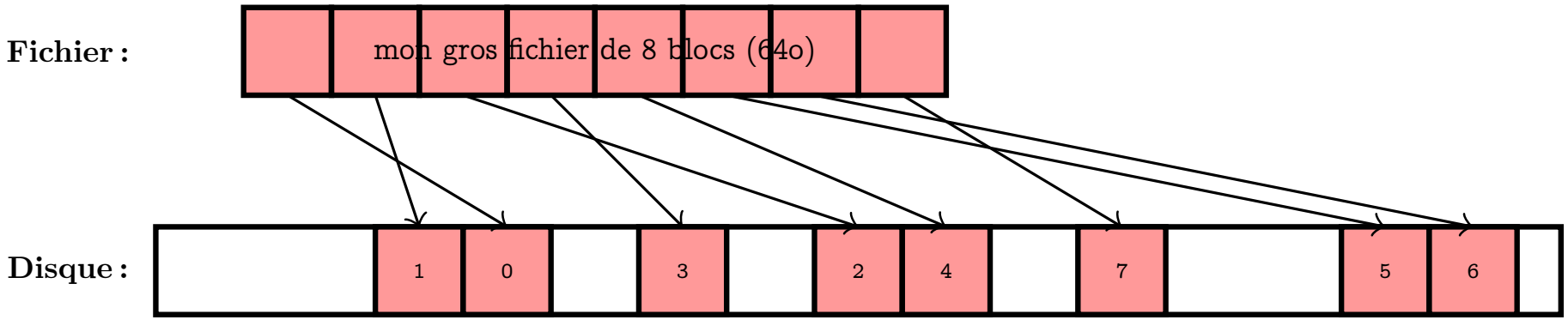
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

Principe

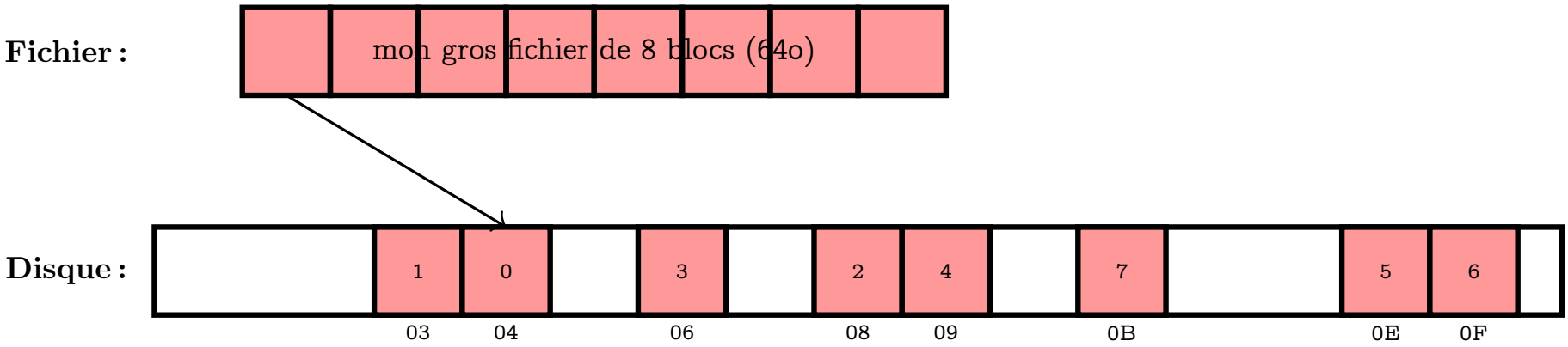
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

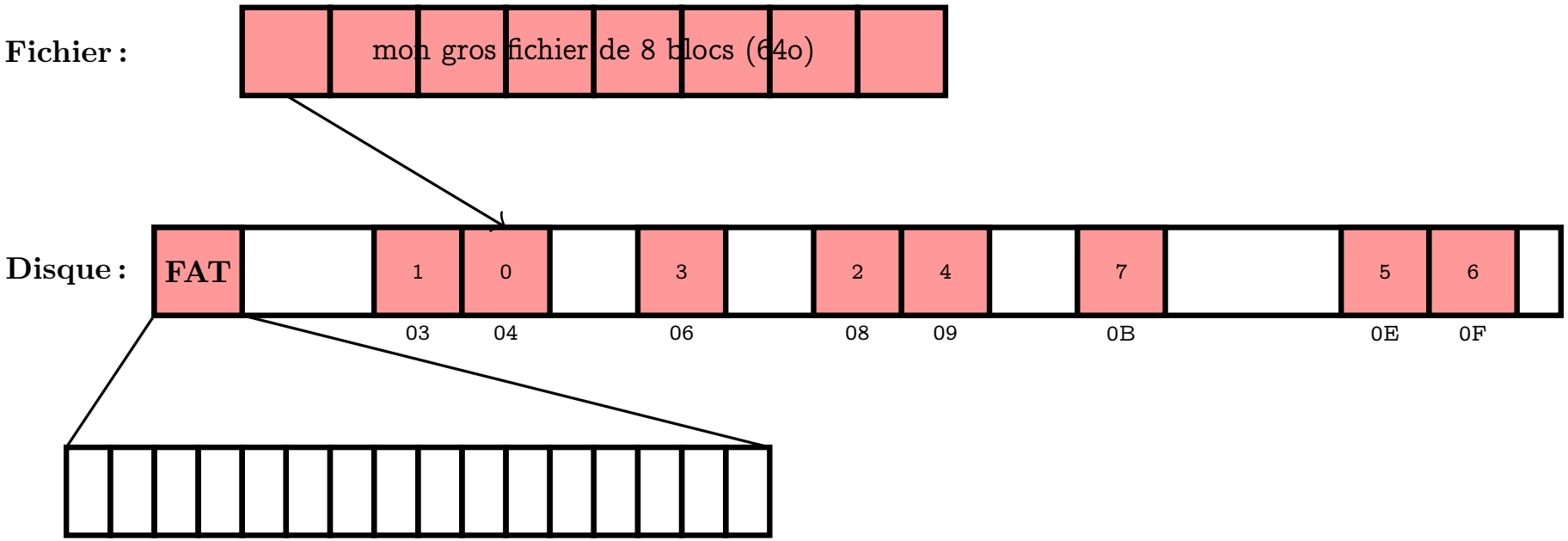
Principe

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



Principe

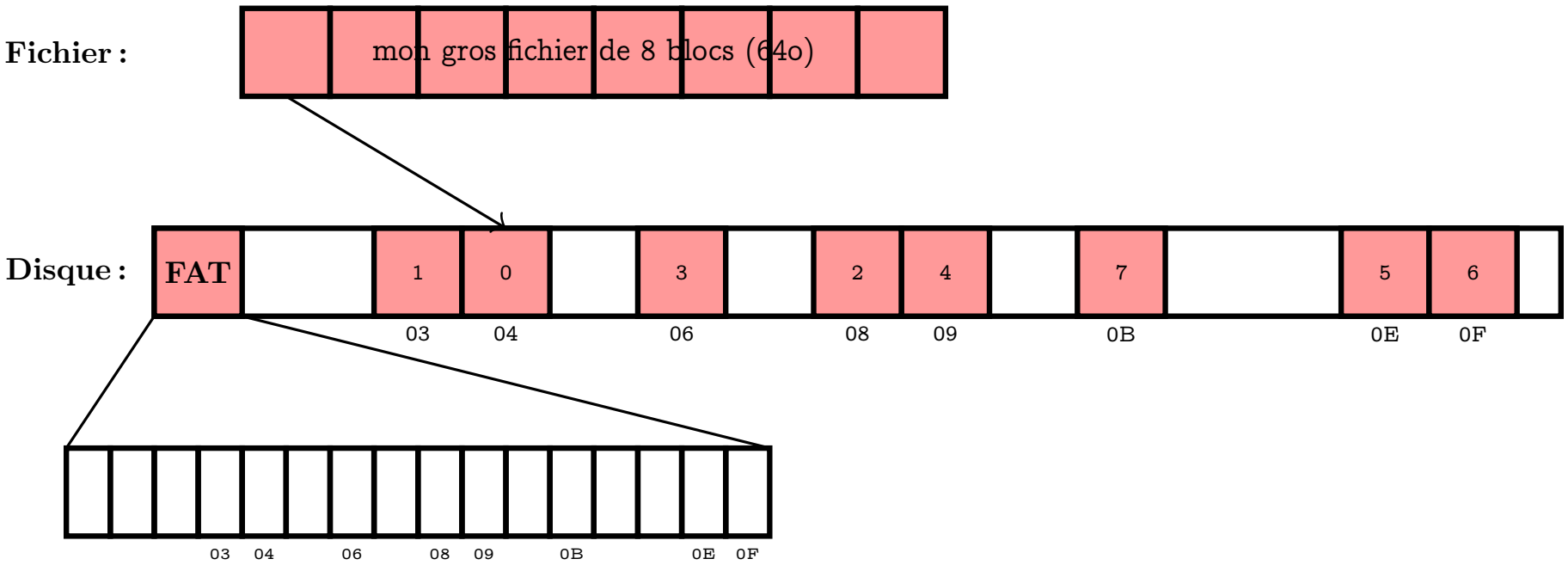
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

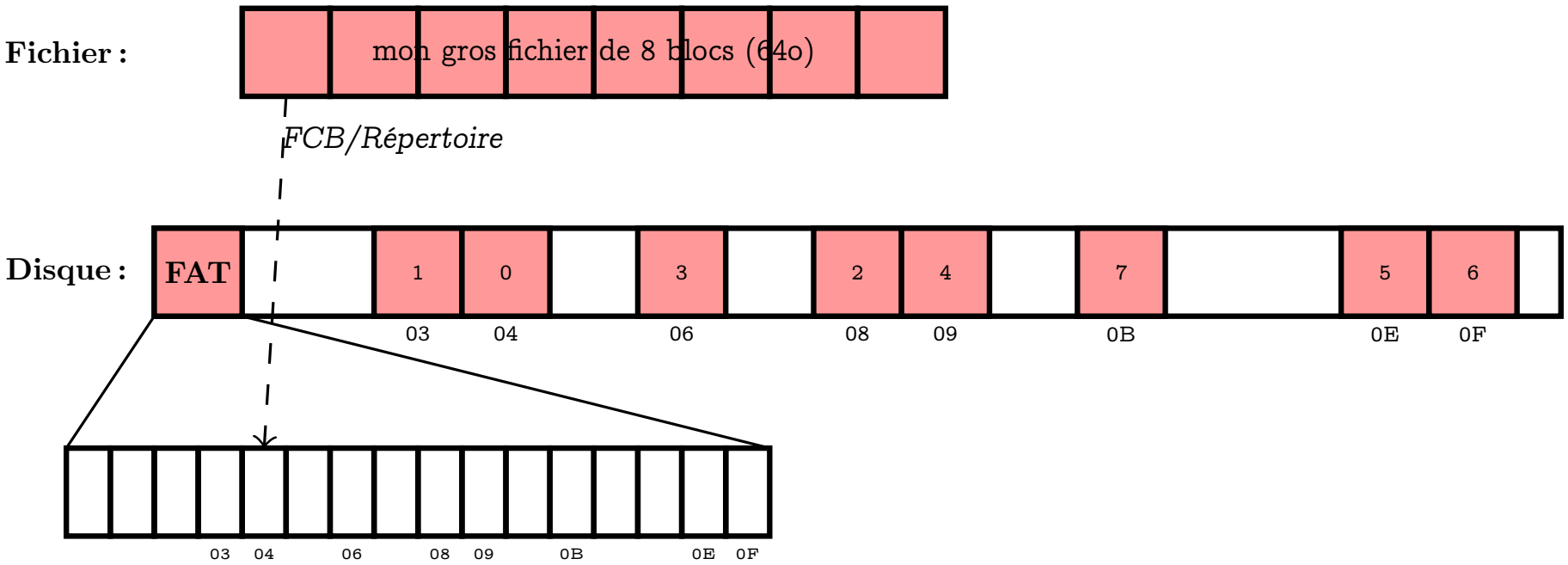
Principe

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



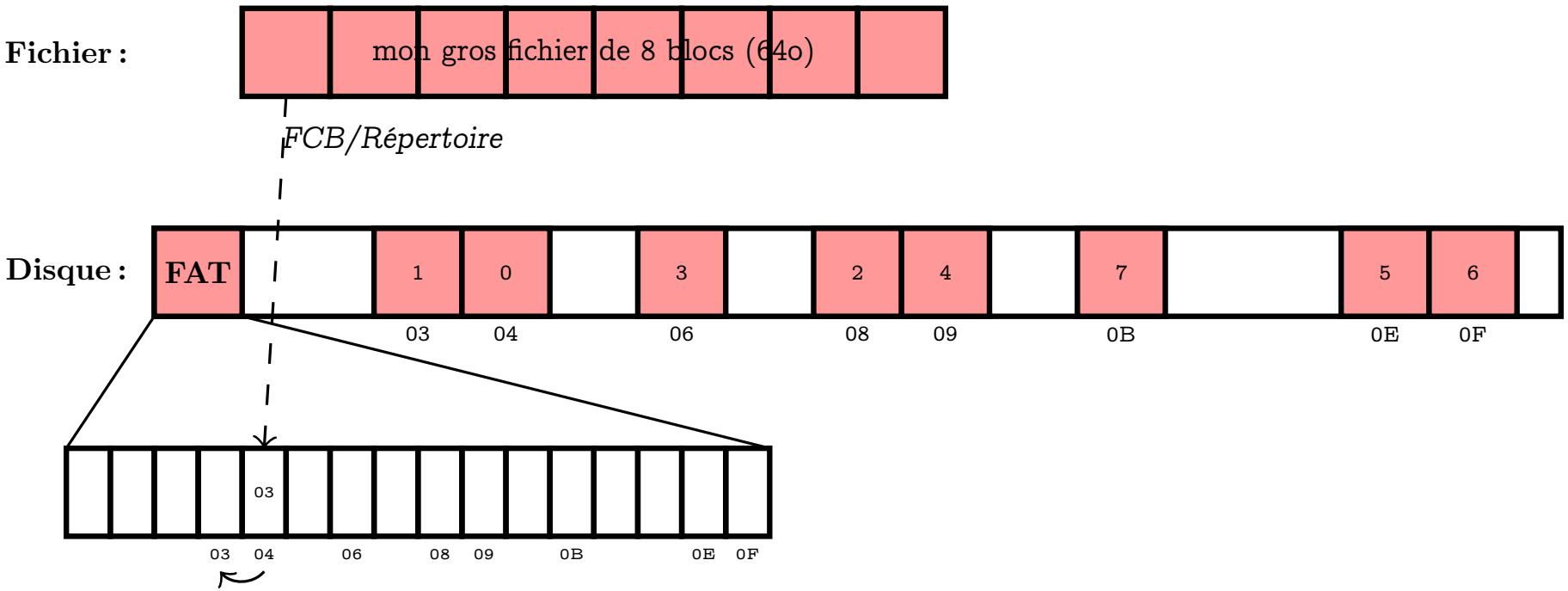
Principe

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



Principe

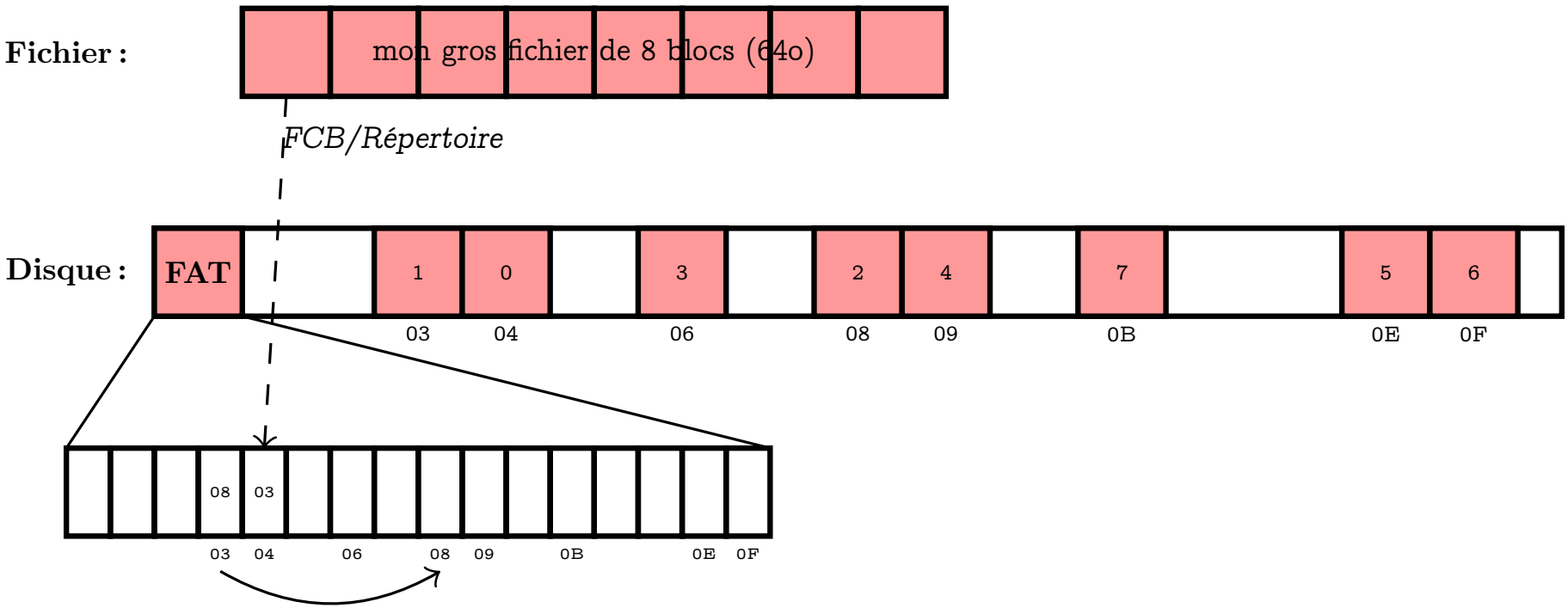
- Utilisé sous MSDOS (Intel) et OS/2 (IBM)
- Allocation indexée
- Liste chaînée des index en début de partition



File Allocation Table (FAT)

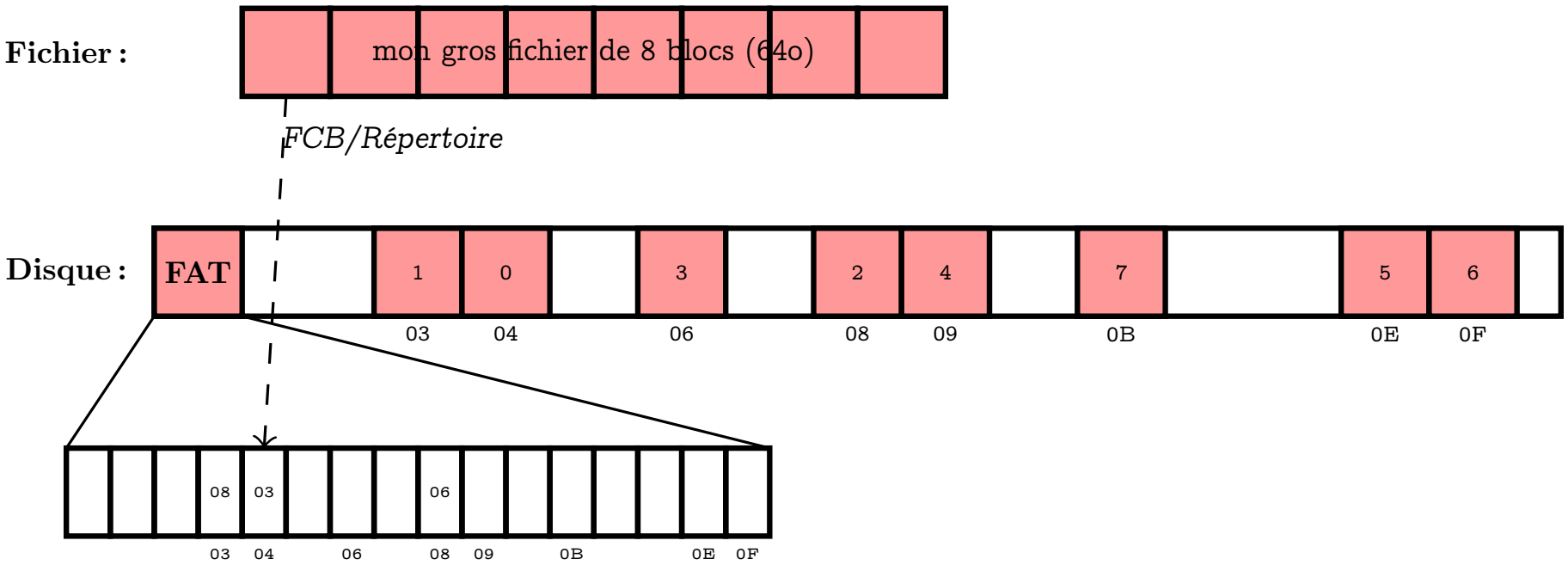
Principe

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



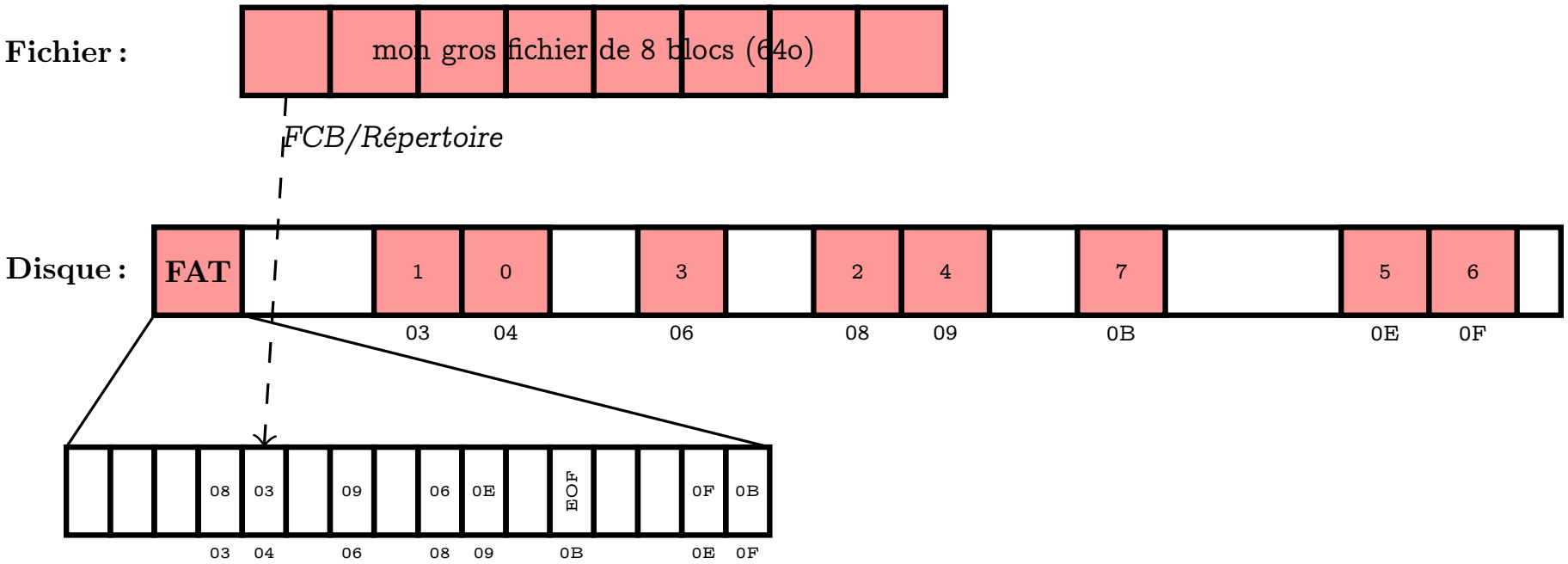
Principe

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



Principe

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



File Allocation Table (FAT)

Principe

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

Avantages

✓ FCB : adresse premier bloc = premier index
✓ Pas de fragmentation (allocation indexée)
✓ Allocation bloc simple
✓ Accès rapide (FAT chargée en cache)

File Allocation Table (FAT)

Principe

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

Avantages

✓ FCB : adresse premier bloc = premier index
✓ Pas de fragmentation (allocation indexée)
✓ Allocation bloc simple
✓ Accès rapide (FAT chargée en cache)

Inconvénients

- Fiabilité : FAT perdue $\rightarrow$ disque foutu !

File Allocation Table (FAT)

Principe

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

Avantages

✓ FCB : adresse premier bloc = premier index
✓ Pas de fragmentation (allocation indexée)
✓ Allocation bloc simple
✓ Accès rapide (FAT chargée en cache)

Inconvénients

- Fiabilité : FAT perdue $\rightarrow$ disque foutu !

doubler la FAT (sur 2 blocs distincts)

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