Systèmes de fichiers

2017 – 2018

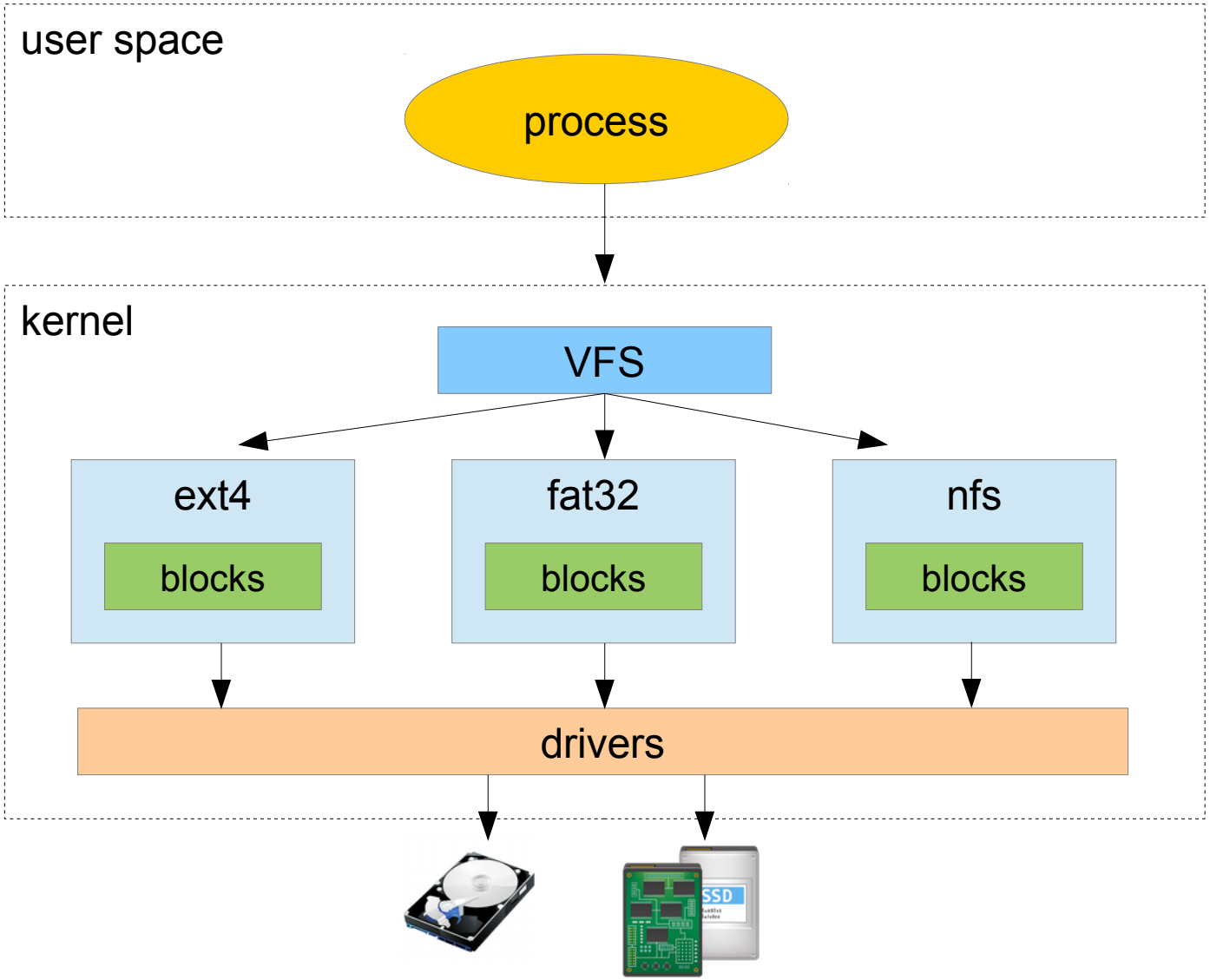
Florent Gluck – Florent.Gluck@hesge.ch

Version 0.3

h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

Hes·so GENÈVE
Haute Ecole Spécialisée de Suisse occidentale

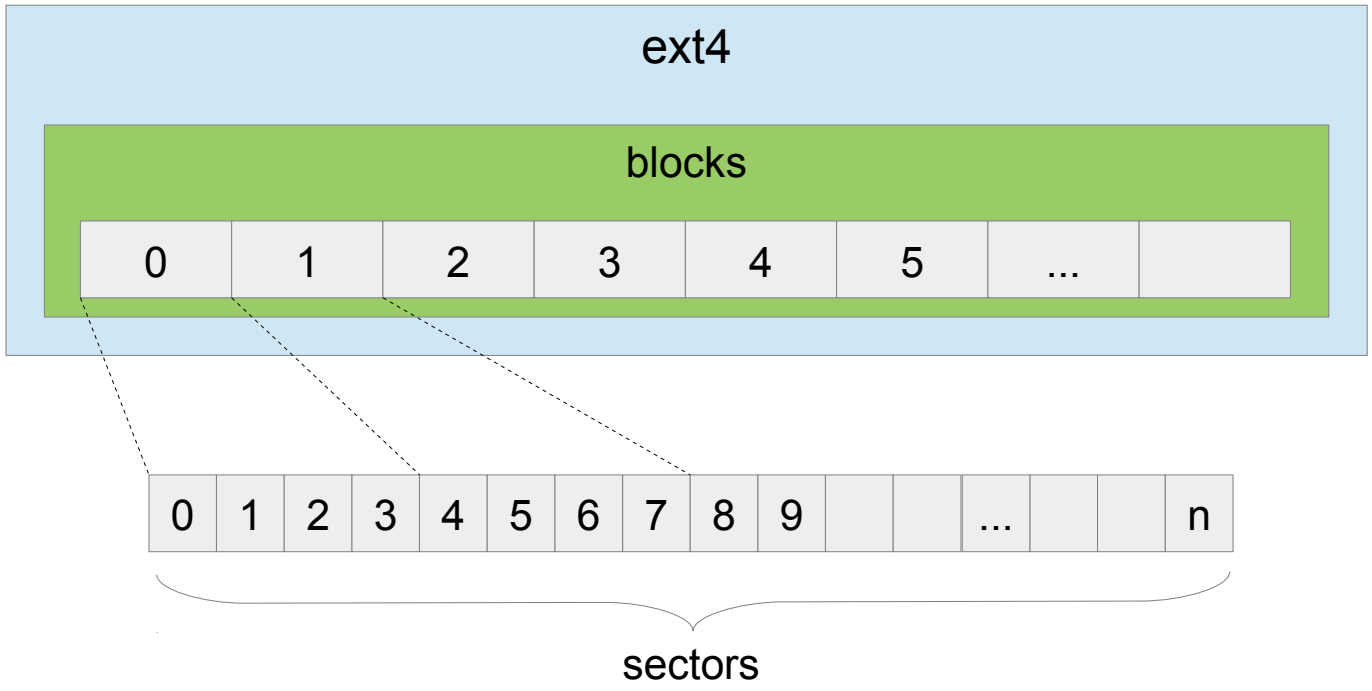
Sys. d'exploitation et sys. de fichiers



Blocs

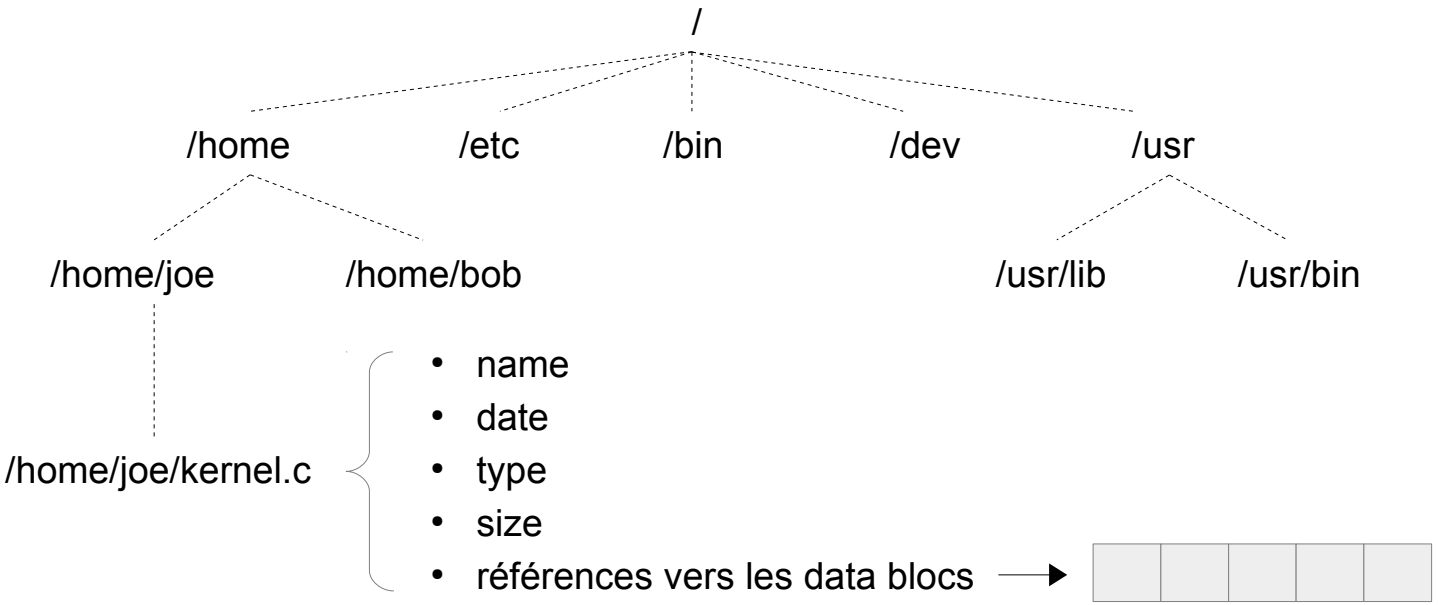
- Le système de fichiers (FS) possède sa propre notion de bloc.
- A un bloc au niveau FS, correspond un ou plusieurs secteurs.

  → Détermine la taille d'un bloc au niveau FS.



Concepts

- 3 concepts différents pour manipuler répertoires et fichiers
  - Entrée de répertoire (*directory entry*)
  - Méta-données (*meta data*)
  - Blocs du fichier (*data blocks*)



Méthodes d'allocation (1)

- La création d'un fichier ou répertoire nécessite l'allocation de blocs.
- La méthode d'allocation doit s'assurer :
  - d'utiliser l'espace de manière efficace
  - Que les fichiers peuvent être accédés rapidement
- Méthodes d'allocation principales :
  - Contigüe
  - Basée sur les *extent*
  - Liste chaînée
  - Indexée

Méthodes d'allocation (2)

- Métriques principales
  - Fragmentation ?
  - Fichiers peuvent grandir ?
  - Performance des accès séquentiels ?
  - Performance des accès aléatoires ?
  - Facilité d'implémentation ?
  - Efficacité du stockage (*storage overhead*) ?

- Le choix de la méthode dépend de :
  - la technologie de stockage (HDD, SDD, etc.)
  - La politique d'accès et les types d'utilisation

Types de fragmentations

Fragmentation externe

L'espace mémoire total libre est suffisant pour satisfaire une requête d'allocation, mais celui-ci n'est pas contigü → allocation impossible.

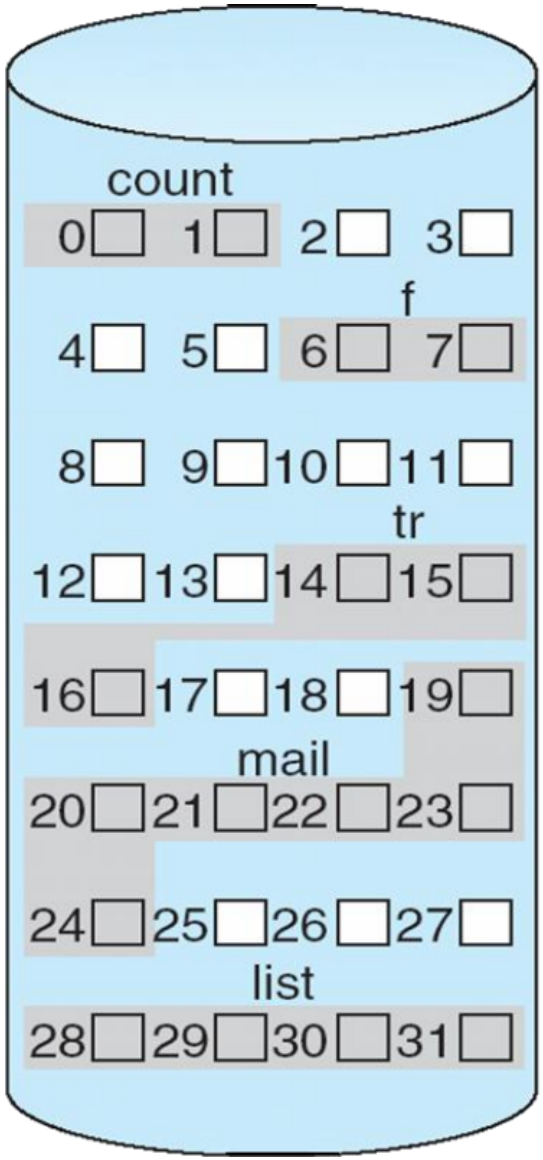
Fragmentation interne

L'espace mémoire alloué peut être un peu plus grand que la mémoire demandée ; cette différence de taille est un espace interne au bloc alloué mais non utilisé → gaspillage.

Allocation contigüe (1)

- Allocation des fichiers similaire à de l'allocation de mémoire contigüe (base et limite) :
  - Utilisateur spécifie la longueur et le FS alloue l'espace en un seul bloc contigü.
  - Métadonnées simple : bloc de début et longueur du fichier.

Allocation contigüe (2)



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation contigüe (3)

Avantages

- Facile à implémenter
- Faible *overhead* de sockage (métadonnées très simples)
- Accès séquentiel rapide
- Accès aléatoire rapide

Désavantages

- Fragmentation externe importante
- Fichiers ne peuvent pas (ou difficilement) grandir

Allocation par *extent* (1)

- Plusieurs régions contigües par fichier
  - Chaque région est un *extent*
  - Métadonnées : contiennent un tableau d'entrées désignant les *extents*
    - Chaque entrée le début et la longueur de l'*extent*

Allocation par *extent* (2)

Avantages

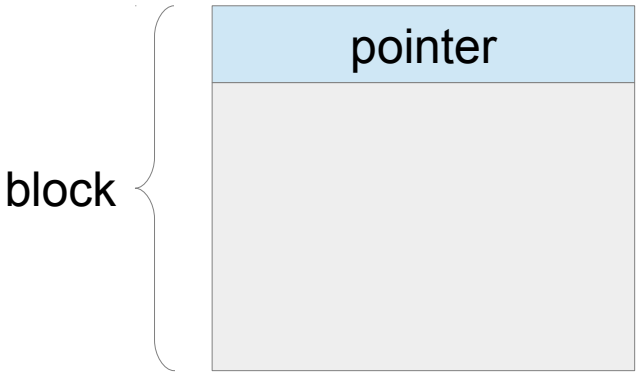
- Facile à implémenter
- Faible *overhead* de sockage (métadonnées très simples)
- Fichier peut grandir (tant qu'il y a des extents disponibles)
- Accès séquentiel rapide
- Accès aléatoire rapide

Désavantage

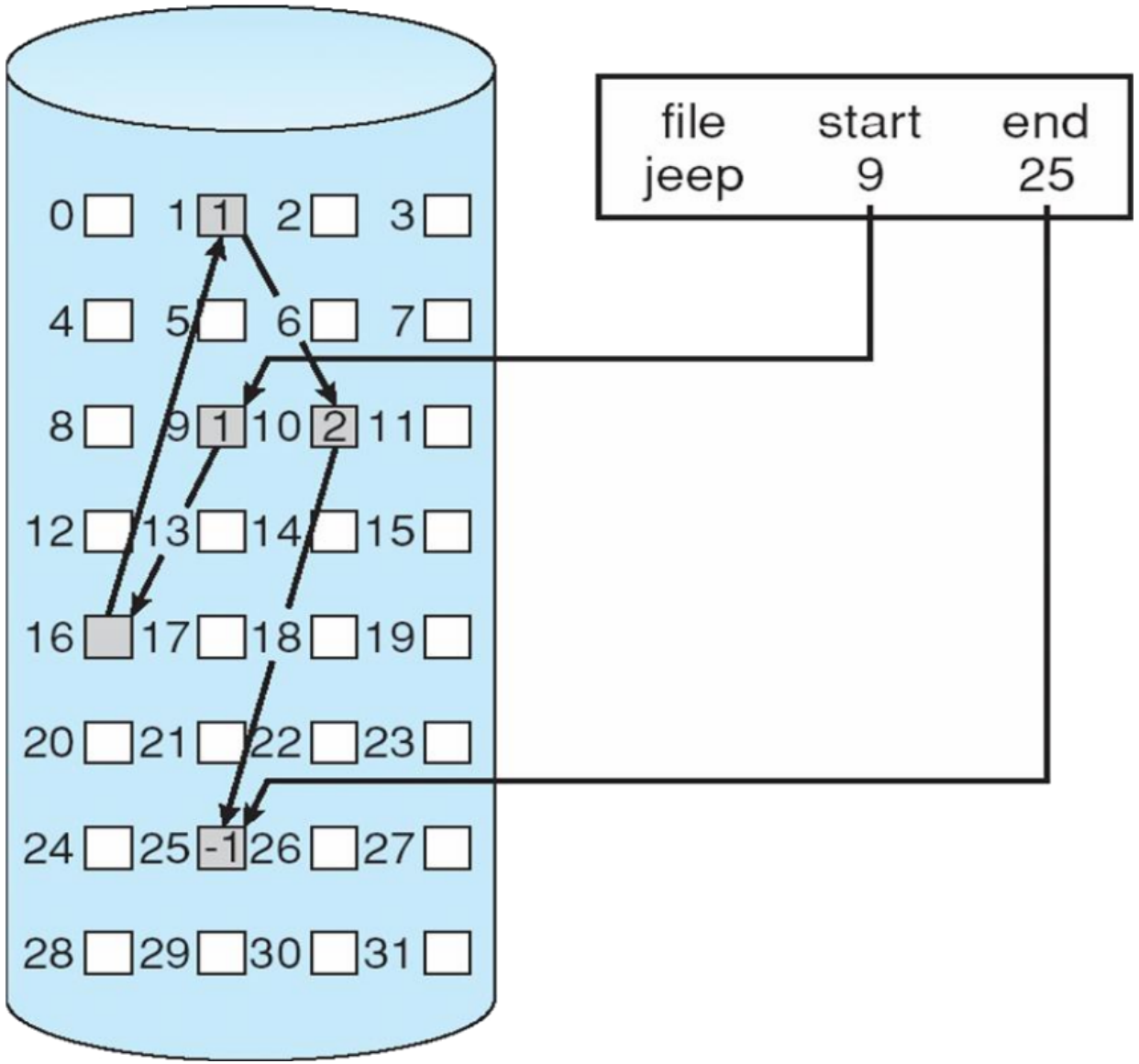
- Fragmentation externe améliorée p/r à l'allocation contigüe, mais encore présente

Allocation par liste chaînée (1)

- Les blocs sont stockés dans une liste chaînée
  - Chaque bloc possède un pointeur sur le suivant
  - Métadonnées : pointeur sur le premier bloc



Allocation par liste chaînée (2)



Allocation par liste chaînée (3)

Avantages

- Pas de fragmentation externe
- Fichier peut facilement grandir, sans limite
- Facile à implémenter, mais stockage du pointeur ennuyeux

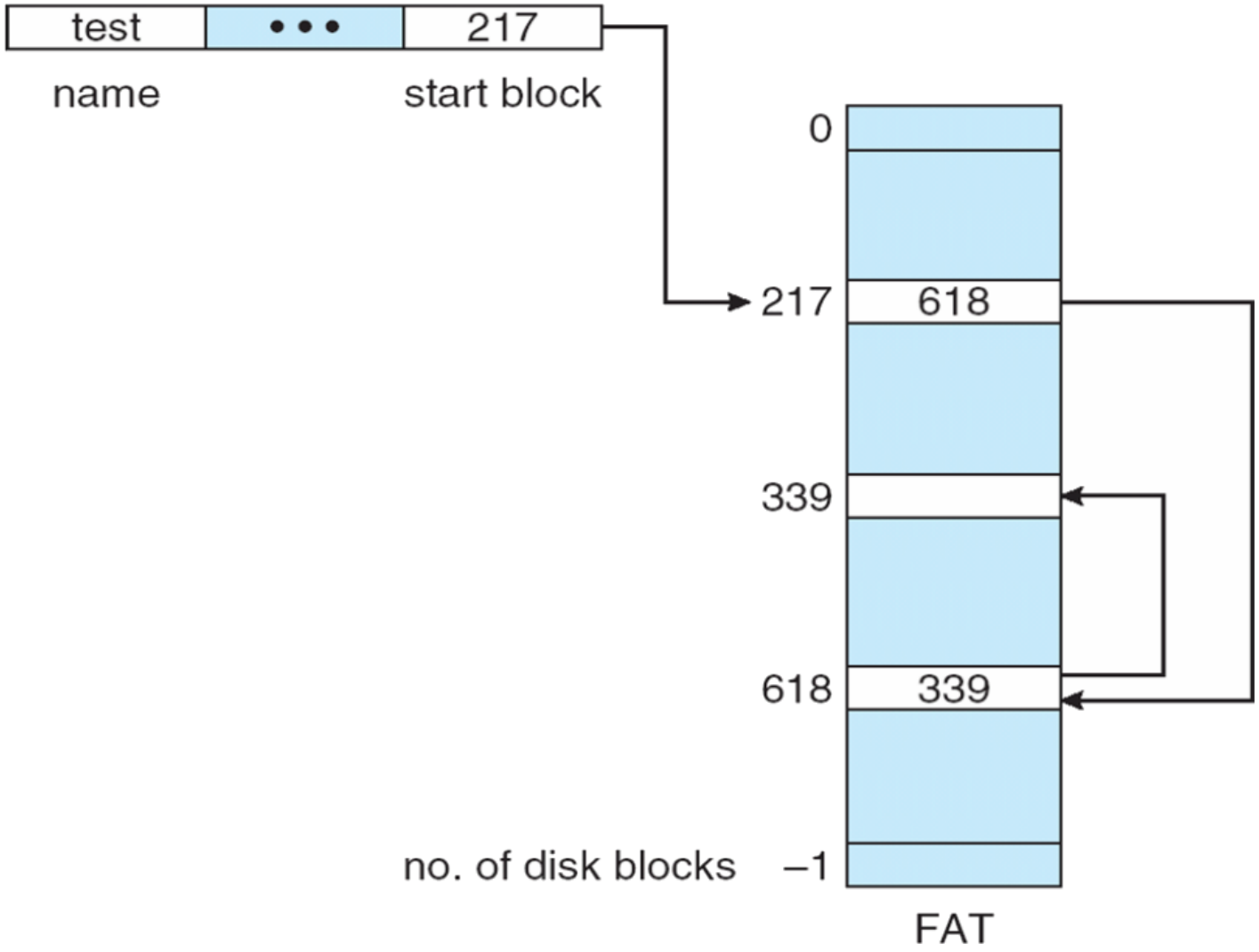
Désavantages

- *Overhead* de stockage (pointeur par bloc)
- Accès séquentiel potentiellement lent
- Accès aléatoire : adresse difficile à calculer

Variante : FAT (1)

- Les pointeurs de la liste chaînée sont stockés à l'extérieur des blocs, dans une table d'allocation de fichiers (File Allocation Table – FAT)
  - Une entrée par bloc
  - Liste chaînée d'entrées pour chaque fichier
- Uitlisé historiquement par MS-DOS, Window et aujourd'hui sur cartes SD (photo) et clés USB

Variante : FAT (2)



Variante : FAT (3)

Avantages

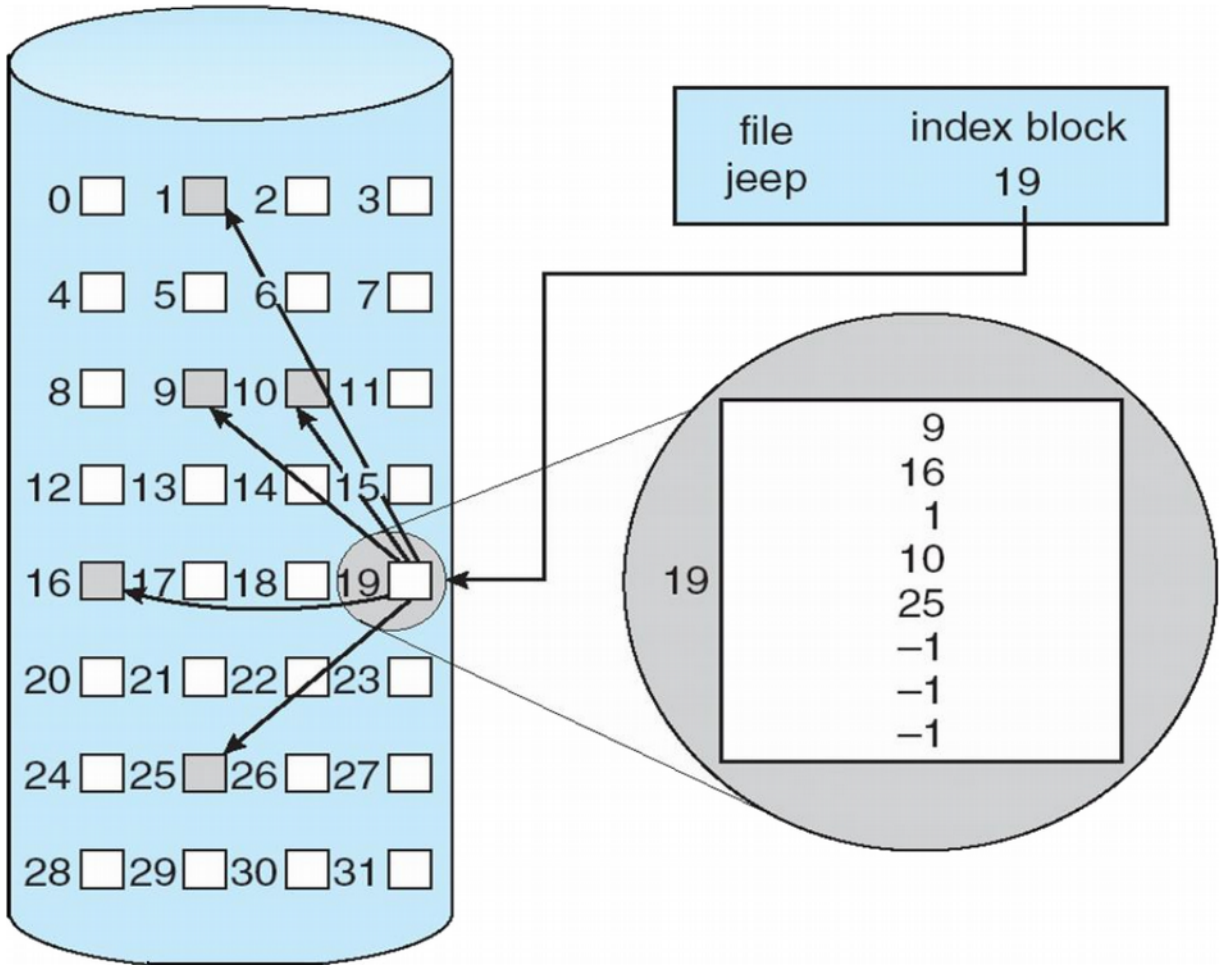
- Accès aléatoire rapide (FAT chargée en RAM)

Désavantages

- *Overhead* important pour le stockage de la FAT (disque et RAM)
- Accès séquentiel potentiellement lent

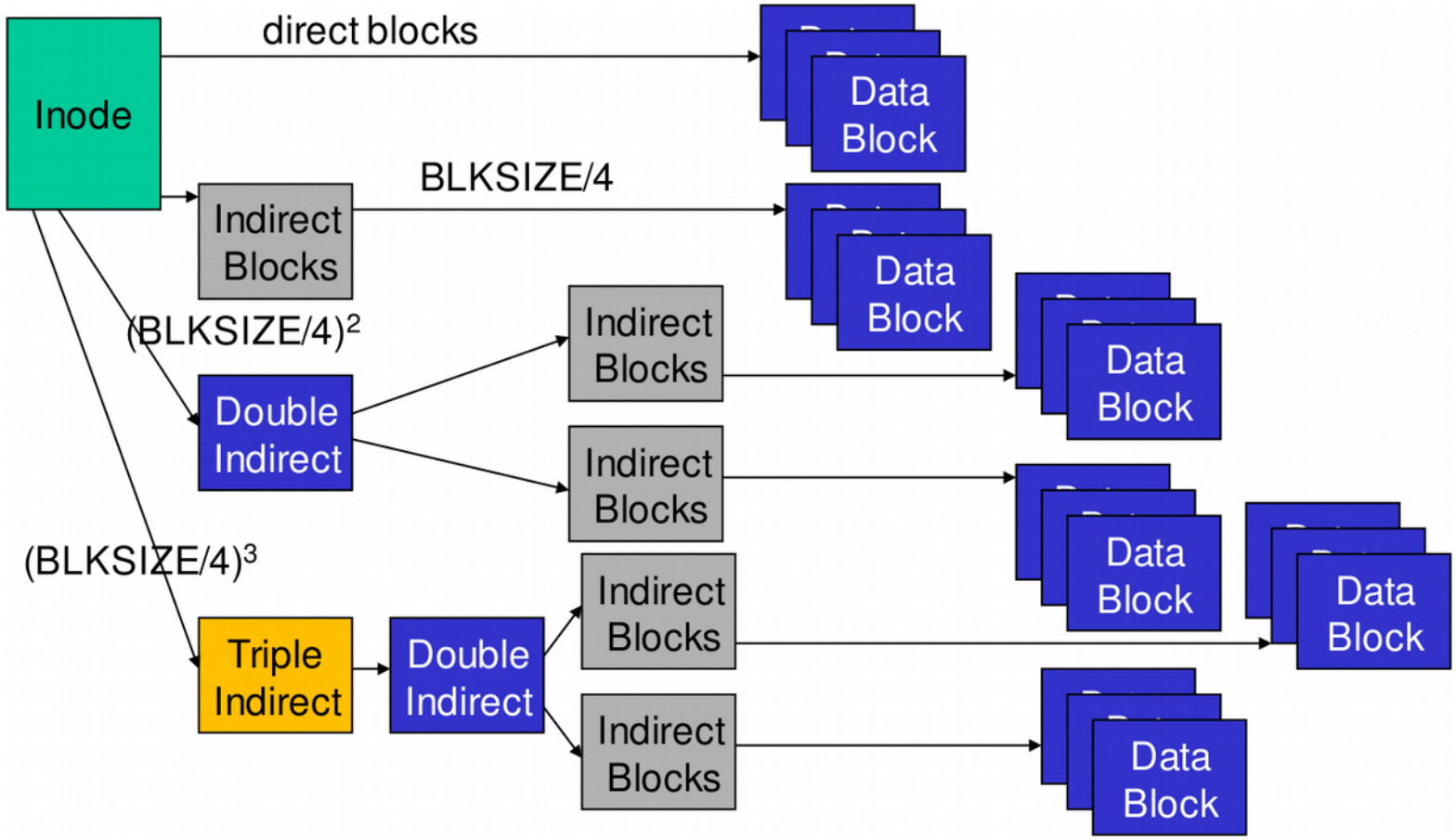
Allocation indexée (1)

- Le fichier possède un tableau de pointeurs (indices) vers les blocs



Allocation indexée (2)

- Plusieurs niveaux d'indirections des indices des blocs (ex. : minix fs, ext2, ext3, UNIX FFS, etc.)



Allocation indexée (3)

Avantages

- Facile à implémenter
- Pas de fragmentation externe
- Fichier peut facilement grandir (tant qu'il y a des pointeurs)
- Accès aléatoire rapide

Désavantages

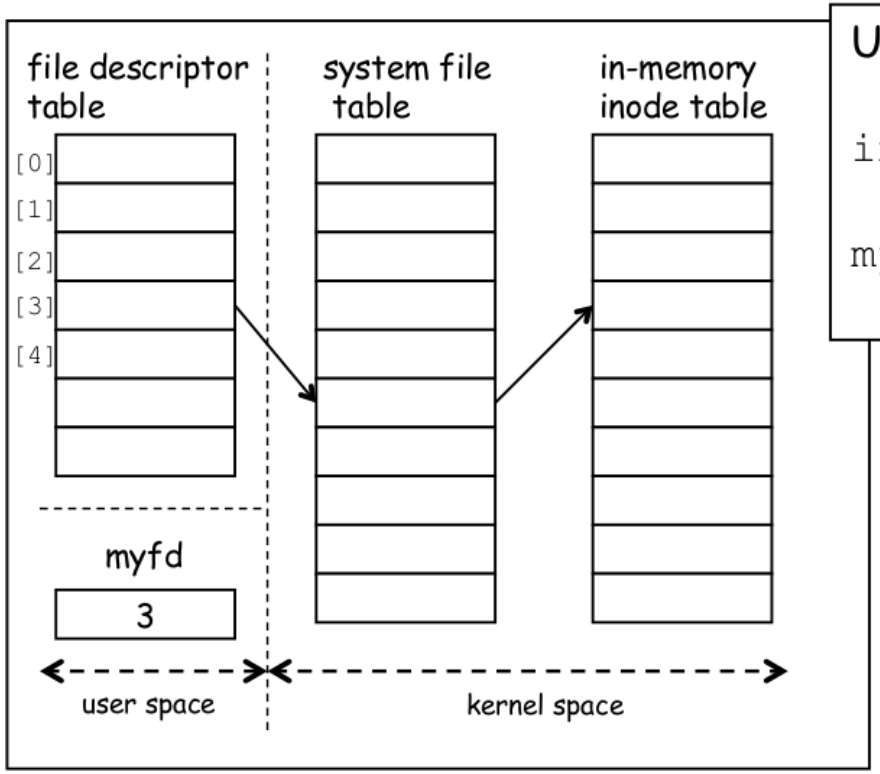
- *Overhead* de stockage pour les indices
- Accès séquentiel potentiellement lent
  - Allocation contigüe des blocs est nécessaire pour un accès rapide

Ressources

- **Operating systems concepts** (9 ème édition), Silberschatz A., Galvin P., Gagne G.

- **Operating Systems: Three Easy Pieces**, Remzi H. et Andrea C. Arpaci-Dusseau. Arpaci-Dusseau Books.

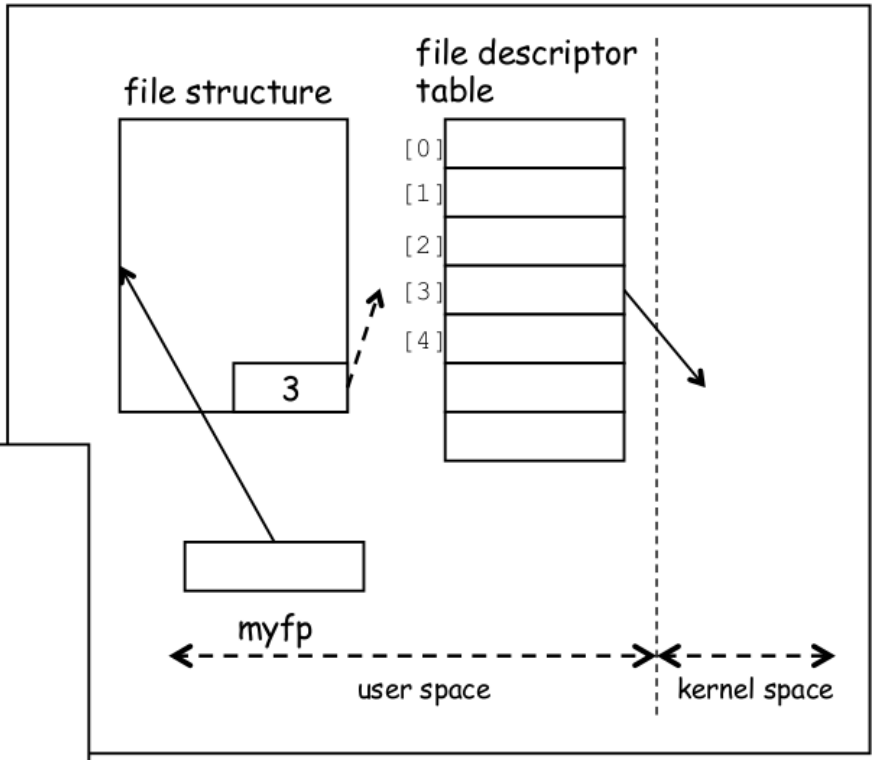
Descripteurs de fichiers



UNIX File Descriptors:

```
int myfd;

myfd = open("myfile.txt", O_RDONLY);
```



ISO C File Pointers:

```
FILE *myfp;

myfp = fopen("myfile.txt", "w");
```