Persistence : implémentation du système de fichiers

Thibaud Martinez
thibaud.martinez@dauphine.psl.eu

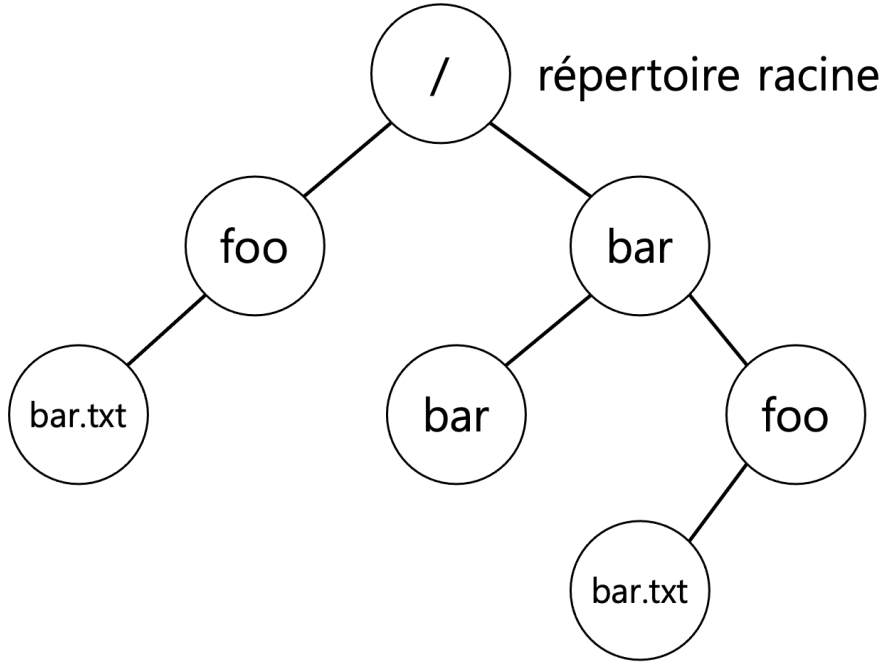
Dauphine | PSL
UNIVERSITÉ PARIS

Cette présentation couvre les chapitres 39 et 40 de Operating Systems: Three Easy Pieces.

Les diapositives sont librement adaptées de diapositives de *Youjip Won (Hanyang University)*.

L'abstraction du système de fichiers

- Le système d'exploitation masque les particularités des disques et des autres périphériques d'E/S pour présenter à l'utilisateur un modèle abstrait.
- Les données sont organisées au sein d'un **système de fichiers** correspondant à une arborescence de **fichiers** et de **répertoires**, avec une racine unique (`/`).
- Pour l'utilisateur, un **fichier** correspond à un tableau linéaire d'octets, qui peuvent être lus ou écrits.



Exemple d'arborescence de répertoires et fichiers

Montage

- Le **montage** est un processus par lequel le système d'exploitation rend les fichiers et les répertoires d'un périphérique de stockage (disque dur, clé usb, etc.) accessibles aux utilisateurs via le système de fichiers.
- L'emplacement dans l'arborescence où le nouveau support monté a été enregistré est appelé **point de montage** (ex: `/home/users`).
- La commande `mount` est utilisée à cet effet.
- Le processus inverse est le **démontage**, au cours duquel le système d'exploitation coupe l'accès des utilisateurs aux fichiers et aux répertoires du support.

Comment implémenter un système de fichiers ?

- **Structures de données ?**
  - Quels types de structures sur le disque sont utilisés par le système de fichiers pour organiser ses données et métadonnées ?
- **Méthodes d'accès ?**
  - Comment les appels effectués par un processus comme `open()`, `read()`, `write()`, etc. sont-ils pris en compte ?
  - Quelles structures sont lues lors de l'exécution d'un appel système particulier ?

vsfs (Very Simple File System)

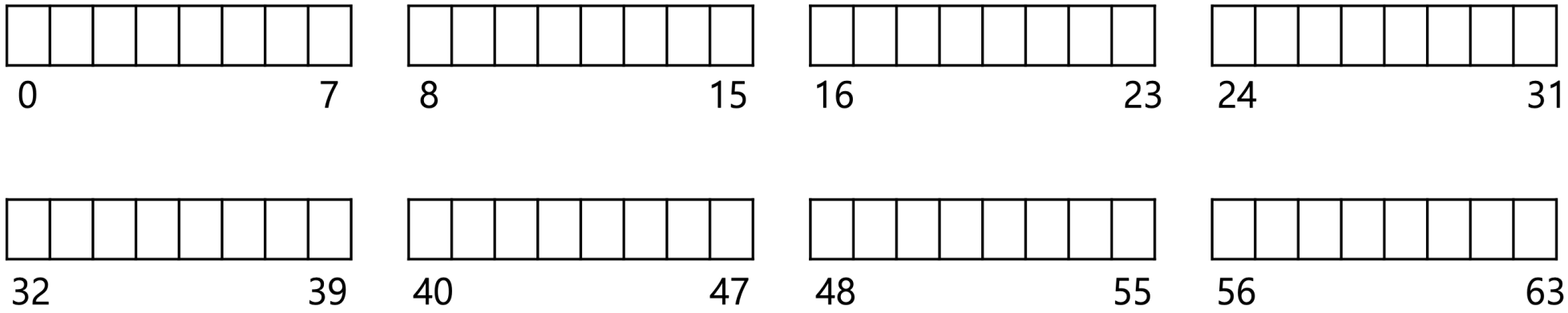
- Une **version simplifiée** d'une implémentation de système de fichiers Unix typique.
- Sert à présenter certaines des structures de base sur le disque, les méthodes d'accès et les différentes politiques que vous trouverez dans de nombreux systèmes de fichiers aujourd'hui.
- Un parmi de nombreux systèmes de fichiers : ext2, ext3, ZFS, AFS...

Organisation générale des structures de données

Le disque est divisé en **blocs**.

- La taille des blocs est 4 KB.
- Les blocs ont des adresses allant de `0` à `N-1` pour un système de fichiers de taille `N`.

Exemple : 64 blocs de 4 KB



Région des données

Région réservée pour le sockage des données utilisateurs.

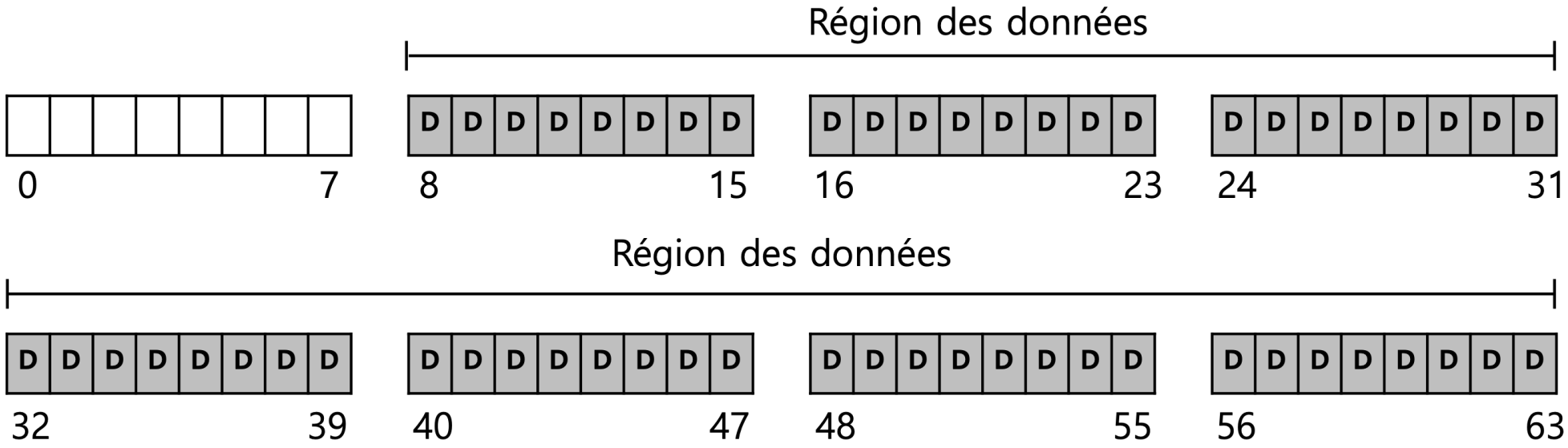


Table des inodes

- Contient un tableau d'inodes sur le disque.
- Taille d'un inode : 256 octets → Un bloc de 4 KB peut contenir 16 inodes.
- Le système de fichiers contient **80 inodes** (nombre maximal de fichiers).

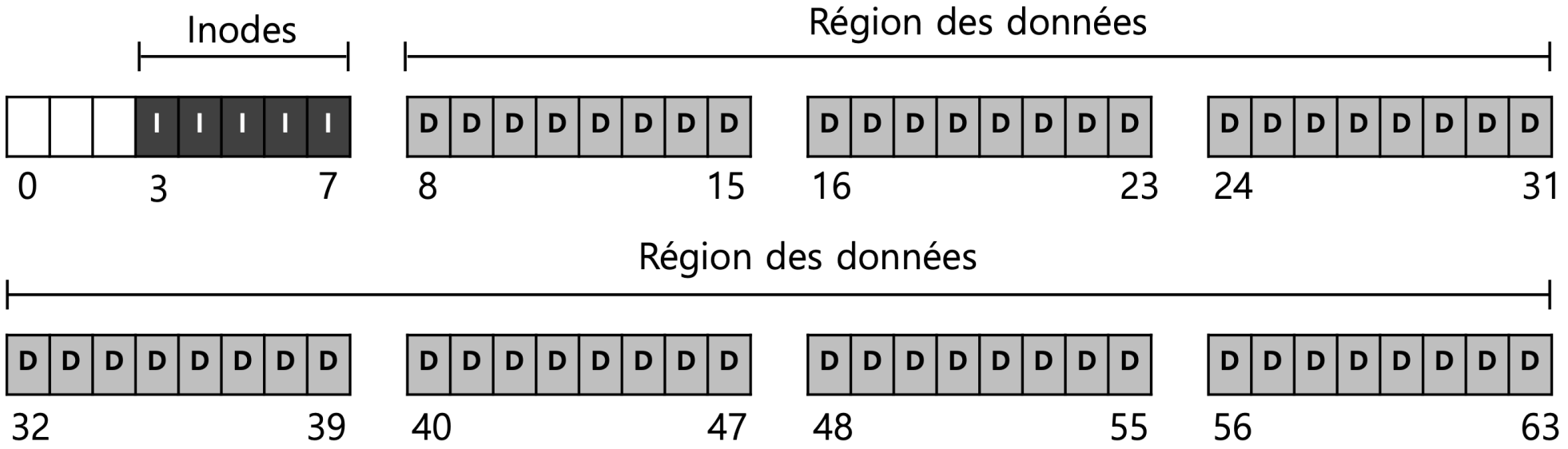
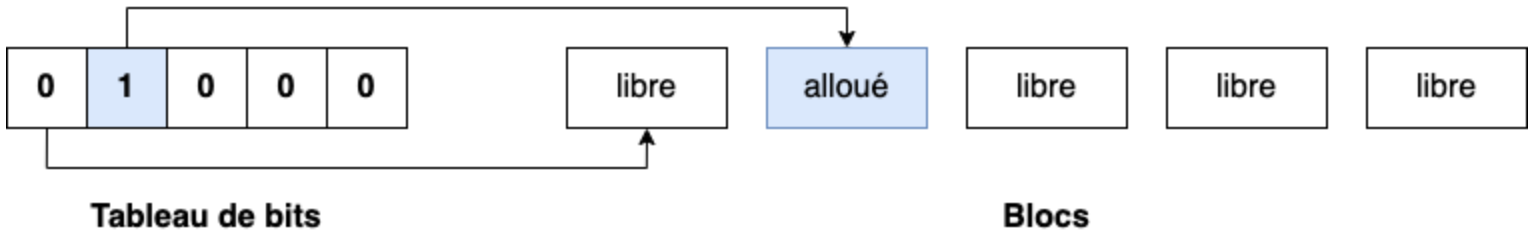


Tableau de bits (*bitmap*)

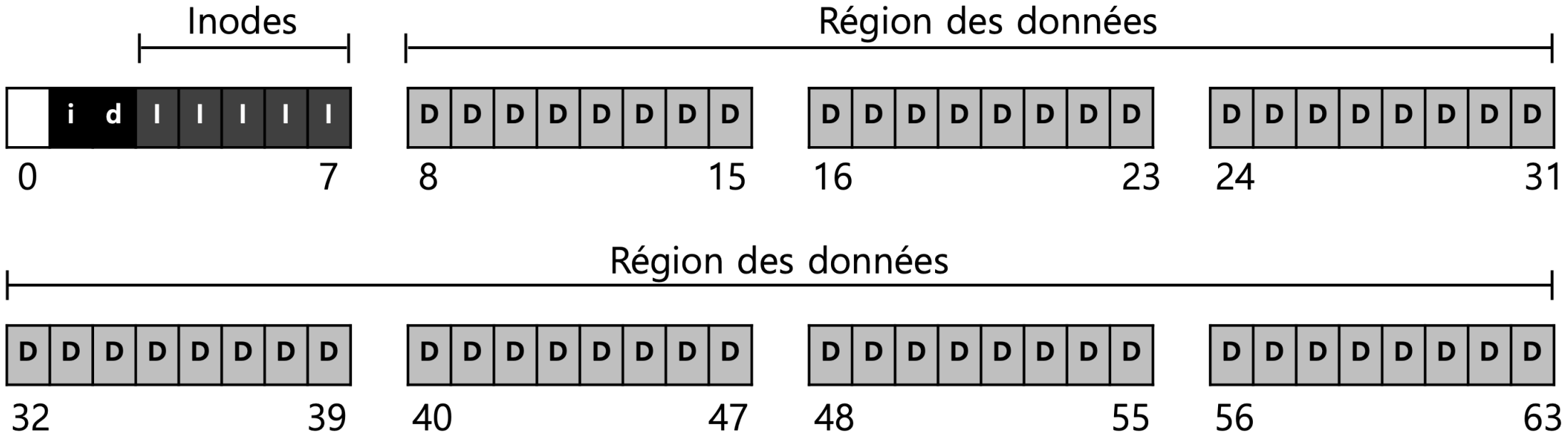
Il est nécessaire de suivre si les inodes ou les blocs de données sont libres ou alloués.

Pour cela, on utilise un **tableau de bits (*bitmap*)** indiquant libre (`0`) ou en cours d'utilisation (`1`).



Structures d'allocation

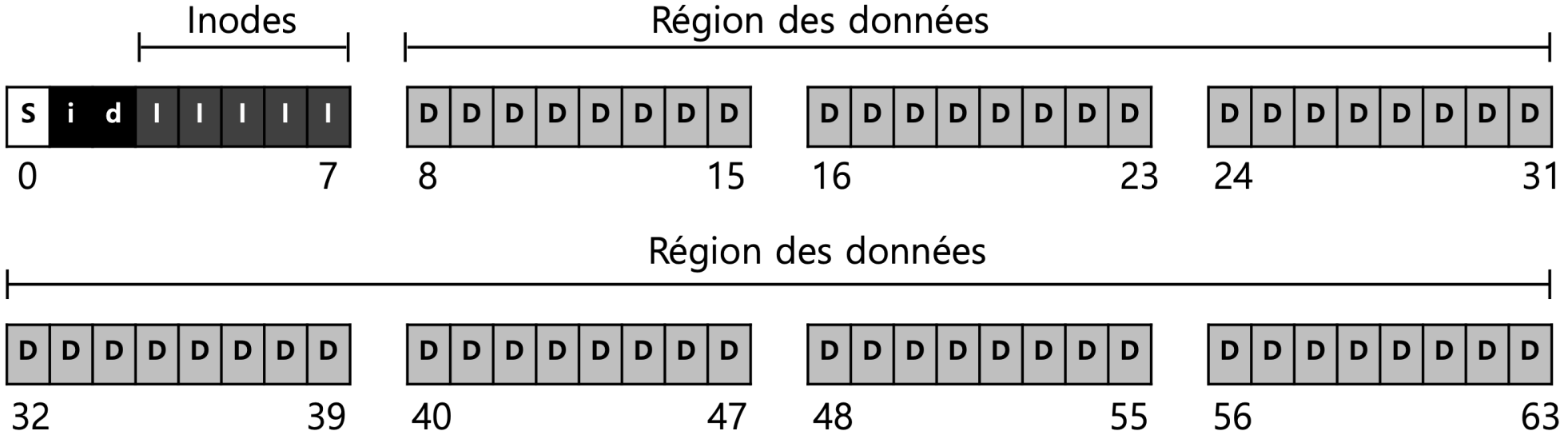
- **Tableau de bits des données** (*inode bitmap*) : pour la région de données.
- **Tableau de bits des inodes** (*data bitmap*) : pour la table des inodes.



Super-bloc

Le **super-bloc** contient des informations propres au système de fichiers. Par exemple, le nombre d'inodes, l'emplacement de la table des inodes, etc.

Lors du montage d'un système de fichiers, le système d'exploitation lit d'abord le super-bloc, afin d'initialiser diverses informations.

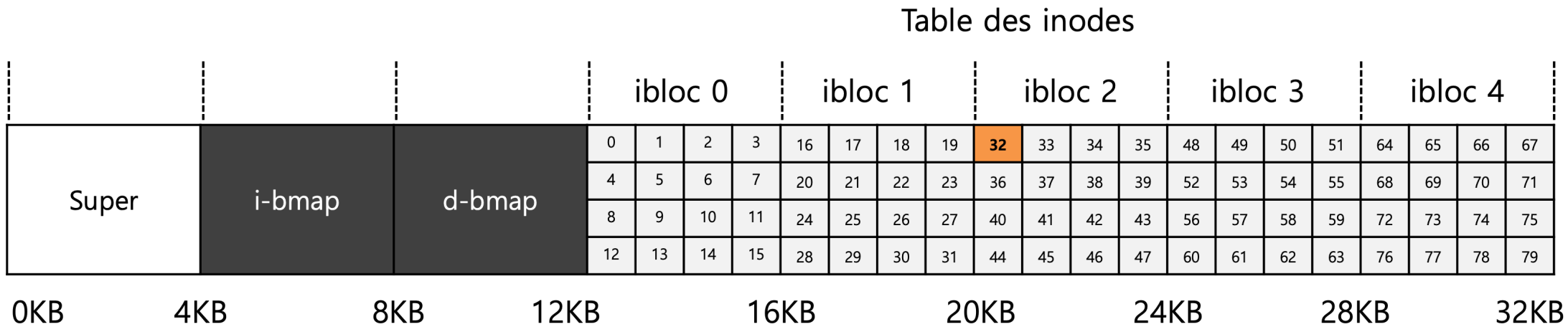


Organisation des fichiers : l'inode

Chaque inode est désigné par un **numéro d'inode** → permet de calculer l'emplacement de l'inode sur le disque.

Exemple : numéro d'inode `32`

- Calcule du décalage (*offset*) dans la table des inodes : `32 x sizeof(inode) (256 bytes) = 8KB`
- Ajout de l'adresse de départ de la table des inodes (`12 KB`) + l'*offset* (`8 KB`) = `20 KB`



⚠️ Les disques ne sont pas adressables par octet, mais par secteur de 512 octets.

Exemple : récupérer le secteur de l'inode de numéro 32

Adresse du secteur de l'inode : $\frac{20KB}{512} = \frac{20 \times 1024}{512} = 40$

inode (*index node*)

Un inode contient toutes les informations relatives à un fichier :

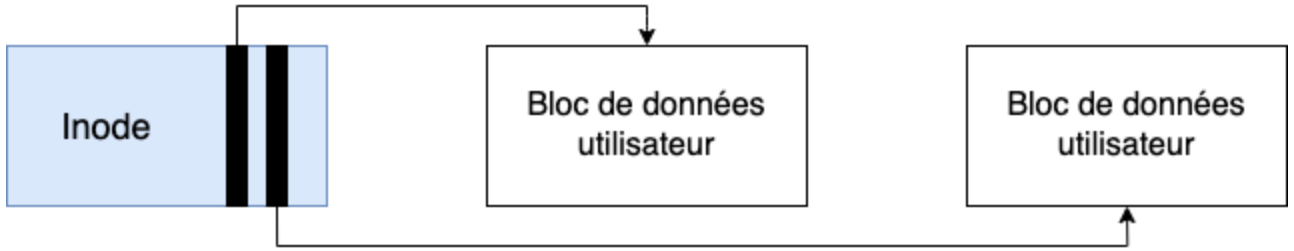
- le type de fichier (fichier normal, répertoire, etc.);
- la taille;
- le nombre de blocs qui lui sont alloués;
- les informations de protection (qui possède le fichier, qui peut y accéder, etc.);
- des données temporelles (date de création, date de modification);
- etc.

Exemple : inode ext2

Taille	Nom	À quoi sert ce champ inode ?
2	mode	Ce fichier peut-il être lu/écrit/exécuté ?
2	uid	À qui appartient ce fichier ?
4	size	Combien d'octets ce fichier contient-il ?
4	time	À quelle moment ce fichier a-t-il été consulté pour la dernière fois ?
4	ctime	À quelle moment ce fichier a-t-il été créé ?
4	mtime	À quelle moment ce fichier a-t-il été modifié pour la dernière fois ?
4	dtime	À quelle moment cet inode a-t-il été supprimé ?
2	gid	À quel groupe ce fichier appartient-il ?
2	links_count	Combien de liens physiques pointent vers ce fichier ?
4	blocks	Combien de blocs ont été alloués à ce fichier ?
4	flags	Comment ext2 doit-il utiliser cet inode ?
4	osd1	Un champ spécifique à l'OS
60	block	Un ensemble de pointeurs de disque (15 au total)
4	generation	Version du fichier (utilisée par NFS)
4	file_acl	Un nouveau modèle de permissions au-delà des bits de mode
4	dir_acl	Listes de contrôle d'accès appelées

Comment l'inode fait référence aux blocs de données ?

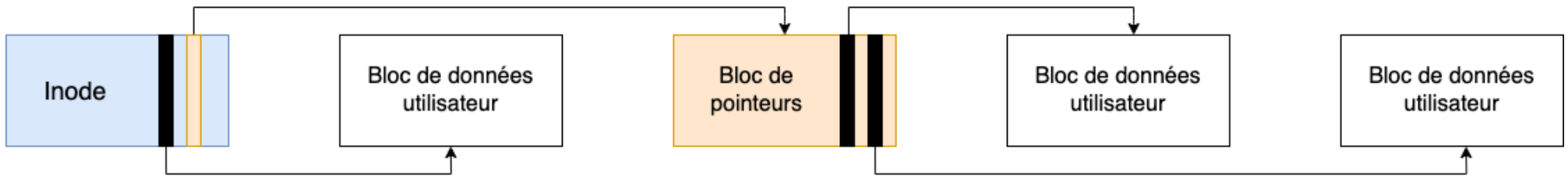
- Une approche simple consisterait à avoir **un ou plusieurs pointeurs directs (adresses de disque) à l'intérieur de l'inode**.
- Chaque pointeur **renvoie à un bloc de disque** appartenant au fichier.



Cette approche est **limitée** : si on souhaite avoir un fichier très volumineux (c'est-à-dire plus grand que la taille du bloc multipliée par le nombre de pointeurs directs dans l'inode), cela est impossible.

Index multi-niveaux

- Pour prendre en charge des fichiers plus volumineux, une idée courante est d'avoir un pointeur spécial : le **pointeur indirect**.
- Au lieu de pointer vers un bloc de données utilisateur, il pointe vers un bloc qui contient plusieurs pointeurs, chacun d'entre eux pointant vers des données utilisateur.
- Ainsi, un inode peut avoir un certain nombre fixe de **pointeurs directs** (par exemple, 12) et un seul **pointeur indirect**.



- Si un fichier devient suffisamment volumineux, un bloc indirect est alloué et le pointeur indirect est défini de manière à pointer sur ce bloc.
- Avec un pointeur direct, en supposant des blocs de 4 KB et des adresses de disque de 4 octets, cela ajoute 1024 pointeurs supplémentaires; le fichier peut atteindre une taille de `(12 + 1024) * 4 KB = 4144 KB`.
- **Niveaux d'indirection supplémentaires** :
    - Un **pointeur indirect double** pointe sur un bloc qui contient des pointeurs indirects.
    - Un **pointeur indirect triple** pointe vers un bloc qui contient des pointeurs indirects doubles.

De nombreux systèmes de fichiers utilisent un index à plusieurs niveaux, tels que Linux ext2 et ext3.

Organisation des répertoires

- Un répertoire est un **fichier** spécial avec un inode dans la table des inodes.
- Les blocs de données du répertoire contiennent une liste de paires `(nom d'entrée, numéro d'inode)`.
- Chaque répertoire a deux fichiers supplémentaires `.` pour le répertoire courant et `..` pour le répertoire parent.

Exemple de répertoire

Contient trois fichiers : `foo`, `bar` et `foobar`.

inum	reclen	strlen	name
5	4	2	.
2	4	3	..
12	4	4	foo
13	4	4	bar
24	8	7	foobar

- `inum` : numéro d'inode du fichier.
- `reclen` (*record length*) : , c'est-à-dire le nombre total d'octets pour le nom, plus l'espace restant.
- `strlen` (*string length*): la longueur réelle du nom.
- `name` : le nom du fichier.

Gestion de l'espace libre

- Le système de fichiers doit déterminer si un inode et un bloc de données sont **libres** ou non.
- La gestion de l'espace libre se fait grâce au **tableau de bits des inodes** et au **tableau de bits des données**.
- Lorsqu'un fichier est **créé**, le système **alloue un inode** en recherchant dans le tableau de bits des inodes puis en le mettant à jour. Il en est de même pour les blocs de données.
- Une politique de pré-allocation est généralement utilisée pour allouer des blocs de données contigus.

Chemins d'accès : lecture d'un fichier sur le disque

Exécuter `open("/foo/bar", O_RDONLY)`

- Traverser le nom de chemin et ainsi localiser l'inode désiré, en commençant à la **racine** du système de fichiers (`/`).
  - Dans la plupart des systèmes de fichiers Unix, le numéro de l'inode racine est 2.
- Lecture du bloc qui contient l'inode numéro 2.
- Regarder à l'intérieur du bloc pour trouver des pointeurs vers les blocs de données (contenu de `/`).
- En lisant un ou plusieurs blocs de données du répertoire, il trouvera le répertoire `foo`.
- Traverser récursivement le nom du chemin jusqu'à l'inode désiré (`bar`).
- Vérifier les permissions finales, allouer un descripteur de fichier pour ce processus et renvoyer le descripteur de fichier à l'utilisateur.

Exécuter `read()` pour lire le fichier

- Lire le premier bloc du fichier, en consultant l'inode pour trouver l'emplacement de ce bloc.
    - Mise à jour de l'inode avec une nouvelle date de dernier accès.
    - Mise à jour de la table des fichiers ouverts en mémoire pour le descripteur de fichier avec l'*offset* du fichier.

Lorsque le fichier est fermé avec `close()`

- Le descripteur de fichier doit être désalloué, mais pour l'instant, c'est tout ce que le système de fichiers doit faire. Aucune entrée/sortie n'a lieu.

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
open(bar)			read			read				
				read						
							read			
					read					
read()					read					
					write			read		
read()					read					
					write				read	
read()					read					
					write					read

Chronologie de la lecture d'un fichier (temps croissant vers le bas)

Chemins d'accès : écriture d'un fichier sur le disque

Exécuter write() pour mettre à jour un fichier avec son nouveau contenu

→ Le fichier peut allouer un bloc (à moins que le bloc ne soit écrasé).

- Nécessite de mettre à jour le bloc de données et le bitmap de données.
- Cela génère 5 entrées/sorties :
  - une pour lire le bitmap de données;
  - une pour écrire le bitmap (pour refléter son nouvel état sur le disque);
  - deux autres pour lire puis écrire l'inode;
  - une pour écrire le bloc lui-même.

Pour créer un fichier, on doit également allouer de l'espace dans le répertoire, ce qui entraîne un nombre d'E/S élevé.

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
create (/foo/bar)			read			read				
				read						
							read			
		read write								
							write			
					read write					
				write						
write()					read					
	read write									
								write		
					write					
write()					read					
	read write									
									write	
					write					
write()					read					
	read write									
										write
					write					

Chronologie de la création d'un fichier (temps croissant vers le bas)

Mise en cache

- La lecture et l'écriture de fichiers sont **coûteuses** et nécessitent de nombreuses entrées/sorties.
  Exemple, long chemin `/1/2/3/..../100/fichier.txt` :
  - Une E/S pour lire l'inode du répertoire et au moins une pour ses données.
  - Il faut effectuer des centaines de lectures rien que pour ouvrir le fichier.
- Pour réduire les E/S, les systèmes de fichiers utilisent la **mémoire vive pour mettre en cache des blocs populaires**. Un cache fixe est utilisé dans les premiers systèmes, les plus modernes, utilisent une approche de dynamique.

Mise en mémoire tampon

Le système de fichiers utilise la mise en **mémoire tampon** pour améliorer les performances d'écriture.

En retardant les écritures, le système de fichiers regroupe certaines mises à jour en un **ensemble plus restreint d'E/S**. Le système de fichiers peut alors aussi **ordonnancer les E/S**.

Certaines applications forcent le transfert des données sur le disque en appelant `fsync()` ou en effectuant des entrées/sorties directes.

fsync(2) — Linux manual page