2 – Systèmes de fichiers
Partie 1 : structures sur disque

HEPIA
Année académique 2016/2017

Contenu

Introduction

Exemple d'une structure simple d'un système de fichier sur le disque

Etude de cas : Minix fs v1.0

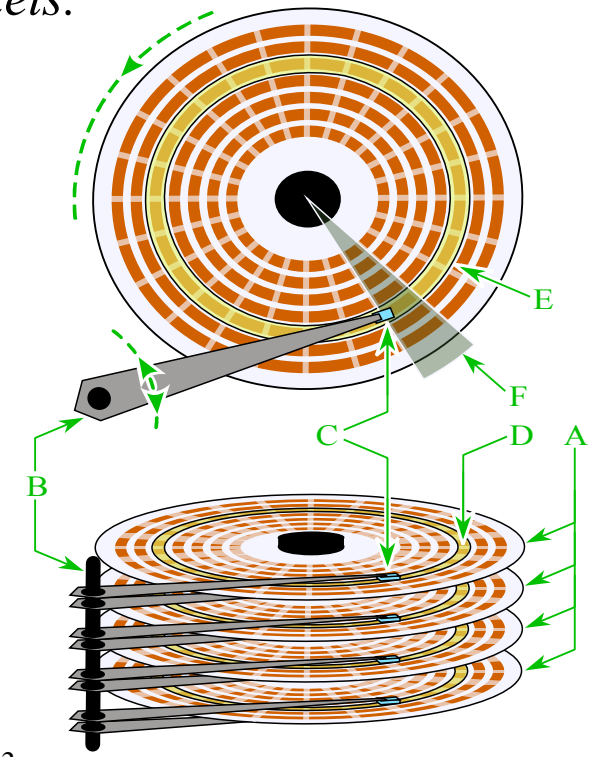
Introduction

Historiquement, les systèmes de fichiers ont été conçu pour accéder avec une interface simple à des périphériques de stockage de masse basé sur des disques magnétiques appelés "Hard Disk Drive" (disque dur HDD) dont la plus petite unité de lecture/écriture était le **secteur** adressable par le triplet Cylinder, Head, Sector (Adressage CHS)

1er disque dur en 1956 : IBM 305 Ramac : poids : 1000 Kg, taille : environ deux grand réfrigérateurs. 50 plateaux de 60cm/24", contenant chacun 100 pistes/faces, contenant chacune 5 secteurs de 100 octets.

Vitesse : - 8.8 ko/s
- 1200 tours/min,
- 2 têtes de lectures/écriture qui passent d'un plateau à l'autre en 1 seconde

Prix : 50 000$



A. Plateaux

B. Bras mobile

C. Têtes de lecture/écriture reliées par le bras mobile

D. Cylindre = coupe **transversale** des plateaux

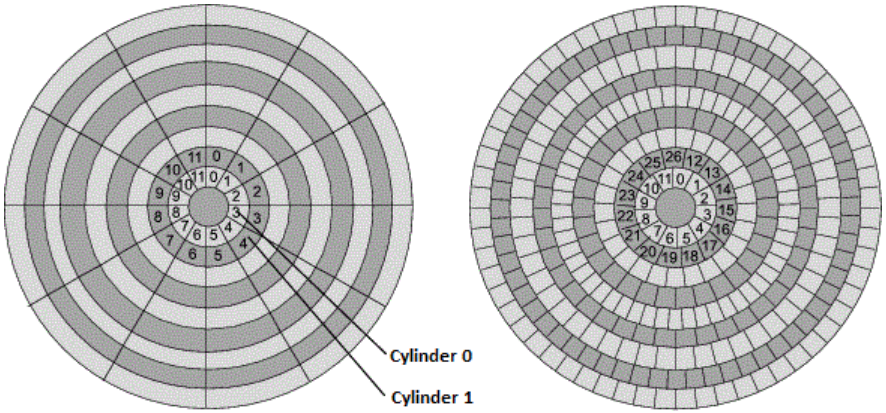
E. Pistes = plusieurs secteurs

F. Secteur angulaire

Introduction

Aujourd'hui (2014), la mécanique de base des HDD reste la même (*des cylindres, des têtes, des secteurs*), simplement :

- les performances et les capacités de stockage ont très fortement augmentée.
- L'adressage physique CHS est remplacé par un adressage logique (LBA).
- La taille standard des secteurs est de 4096 octets avec possibilité d'émulation à 512 octets (la taille standard jusqu'en 2011).
- Le même adressage logique est utilisé sur d'autres technologies (SSD, Flash drives, etc)



CHS Adressing

LBA Adressing

HDD de 3"5 datant de 1998

Introduction

Evolution du prix du stockage d'un 1Go

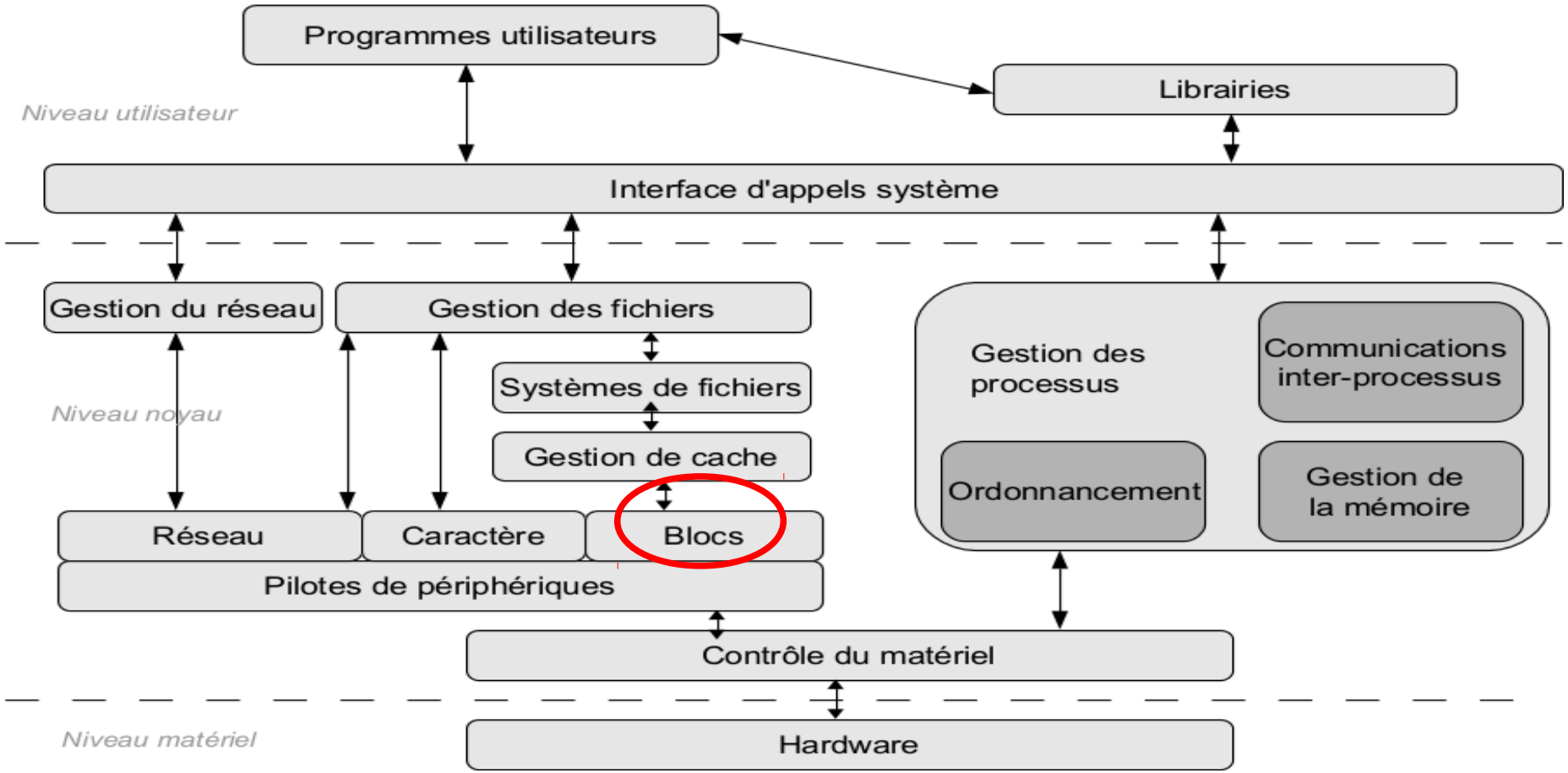
Année	Prix théorique
1956	100 000 000 000$
1981	300 000$
1987	50 000$
1990	10 000$
1994	1000$
2000	10$
2012	0.10$
2014	0.05$

- Les octets de stockage sont passés en quelques décennies d'une ressource rare et chère à une ressource sur-abondante à prix insignifiant:
- qui vérifie encore vraiment la place prise par ses données à part lorsqu'il n'en n'a plus ?

Note : Les prix sont calculés à partir du prix de vente des disque durs à l'époque. A mettre en comparaison avec les prix de l'espace de stockage du « cloud ».

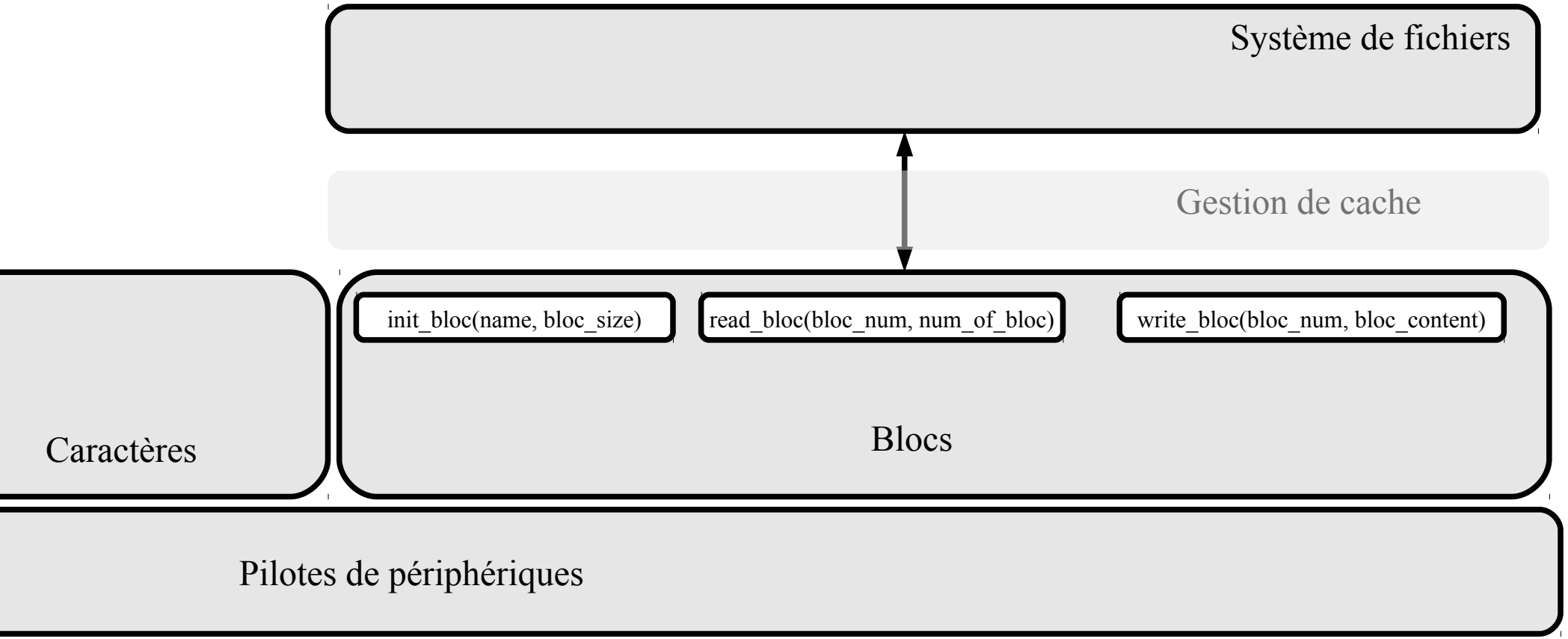
Introduction

Sous Unix on parle de **blocs** plutôt que de secteurs, pour s'abstraire du type de périphérique : tout périphérique dont on peut lire/écrire les données par unité de 512/1024/4096,... octets, est géré par le module de gestion des lectures/écriture par blocs, indépendamment du type de périphérique



Introduction

Si on fait abstraction de la gestion du cache, l'interface d'accès à un périphérique de type bloc est très simple: après initialisation sur la taille d'un bloc, chaque bloc est adressable **logiquement** via son numéro par une fonction d'écriture et une fonction de lecture.



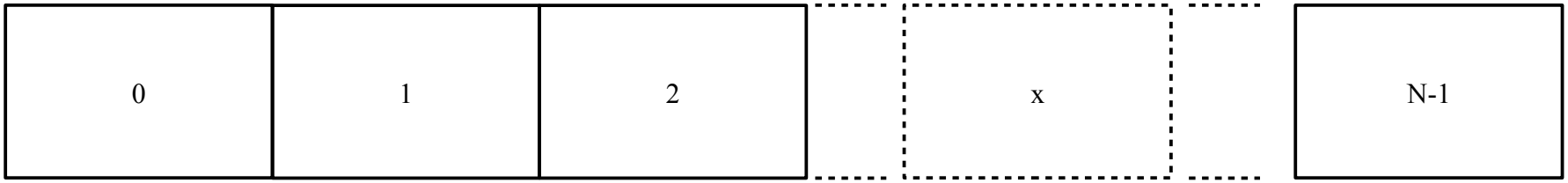
Introduction

- les blocs d'un périphérique de taille de `N` blocs de `k` octets sont adressables de `0` à `N-1` (similaire à un tableau en C).

- La lecture/écriture se fait uniquement par unité de `k` octets

- La lecture/écriture du bloc `0` traite les octets de l'offset `0` à l'offset `k-1` du périphérique.

- La lecture/écriture du bloc `x` traite les octets de l'offset `x*k` à l'offset `x*k+k-1`



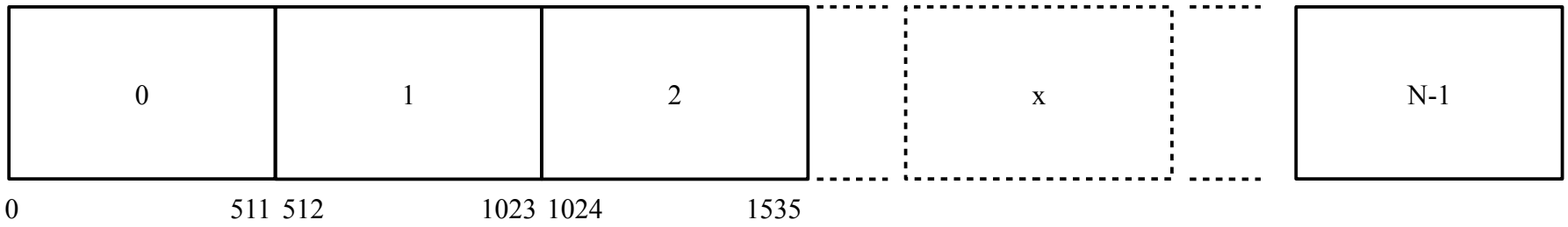
Introduction

Exemple : périphérique de 10 Mio, taille de bloc de 512 octets

- Quelle est l'intervalle d'adressage logique des blocs ?

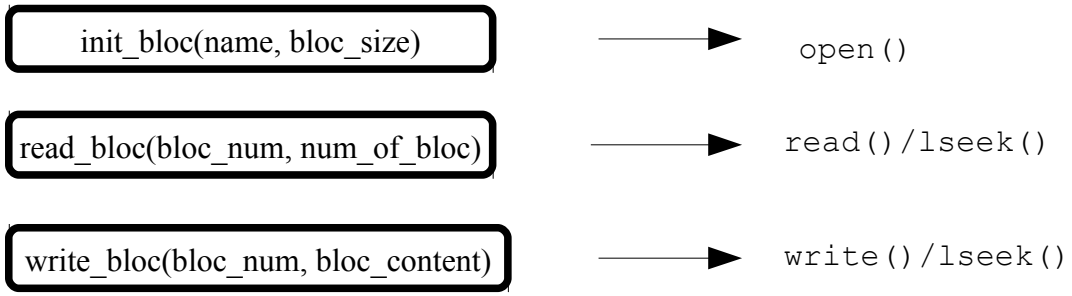
- Quelle est l'intervalle des offsets des octets du bloc numéro 17 ?

- Dans quel bloc se trouve l'octet du périphérique d'offset 10000 ?

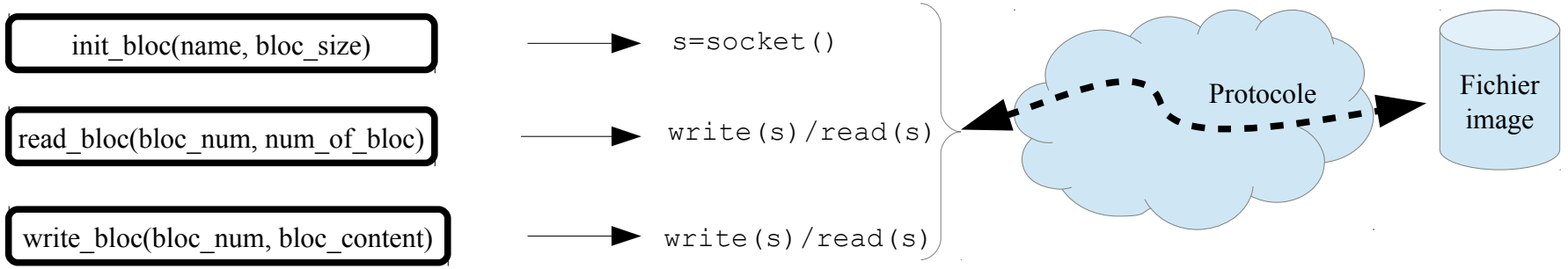


Introduction

Avec cet adressage logique, on peut facilement émuler un périphérique bloc à partir d'un fichier image local, accédé par les appels systèmes classique `read/write/lseek`.



Le fichier image peut-être distant, si on implémente l'abstraction au dessus des sockets SOCK_STREAM et d'un protocole qui reste très simple (cf question 7 de TP2_socket.txt)



Introduction

On peut lire/écrire des blocs de manière adressés logiquement, mais on est encore loin d'un système de fichier.

Quelles autres fonctions manquent encore ?

- Indice : commencer par les fonctions de plus bas niveau puis remonter peu à peu vers les appels système connus.

Contenu

Introduction

Exemple d'une structure simple d'un système de fichier sur le disque

Etude de cas : Minix fs 1.0

Concepts clefs d'un système de fichier stocké

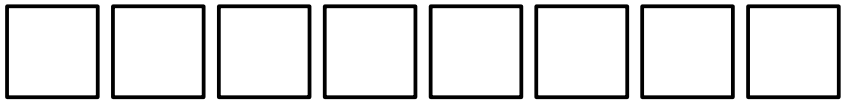
Trois points clefs à comprendre, dont tout le reste dépends :

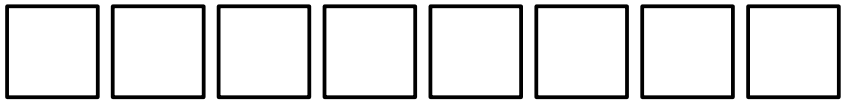
1) Quelles sont les structures de données stockées sur le disque qui organisent les données des fichiers ainsi que leur **méta-données** ?

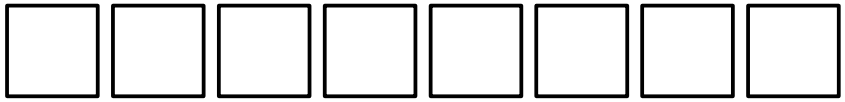
2) Comment est-ce que ces structures de données sont-elles reliées les unes aux autres ?

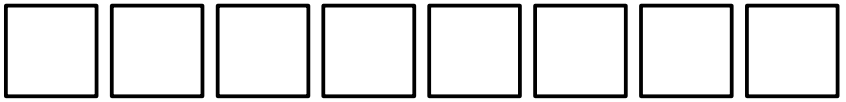
3) Quelles sont les interfaces d'accès au système de fichier ? Autrement dit, comment est-ce que le système associe les appels à par exemple read(), write() ou open(), etc aux points 1) et 2) ?

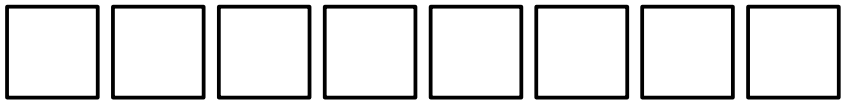
Exemple de structure simple d'un FS

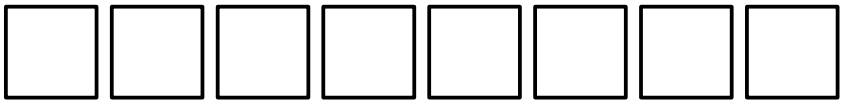


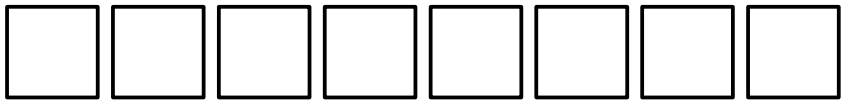


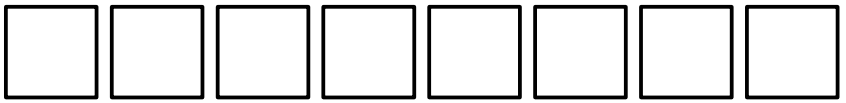






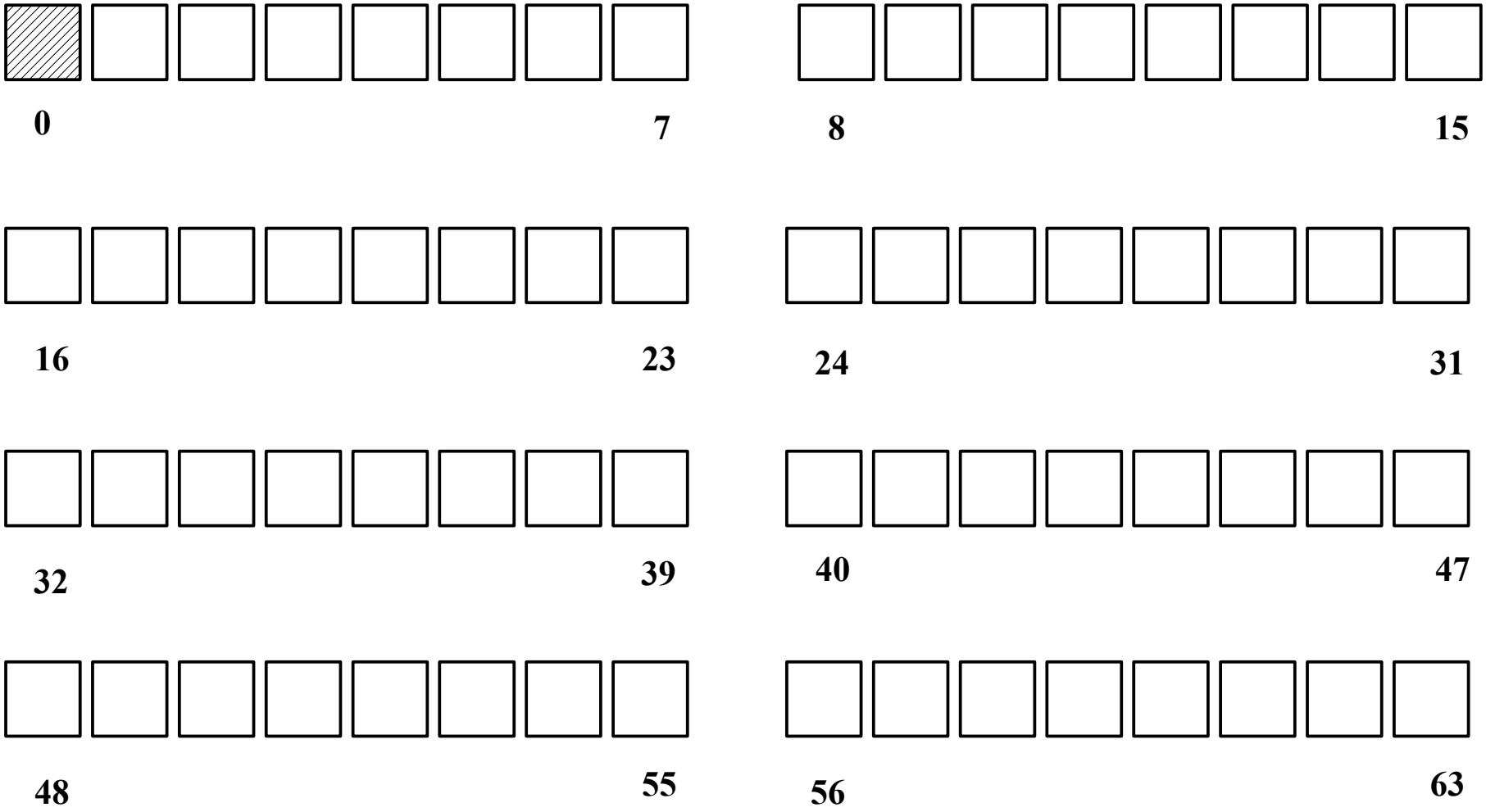






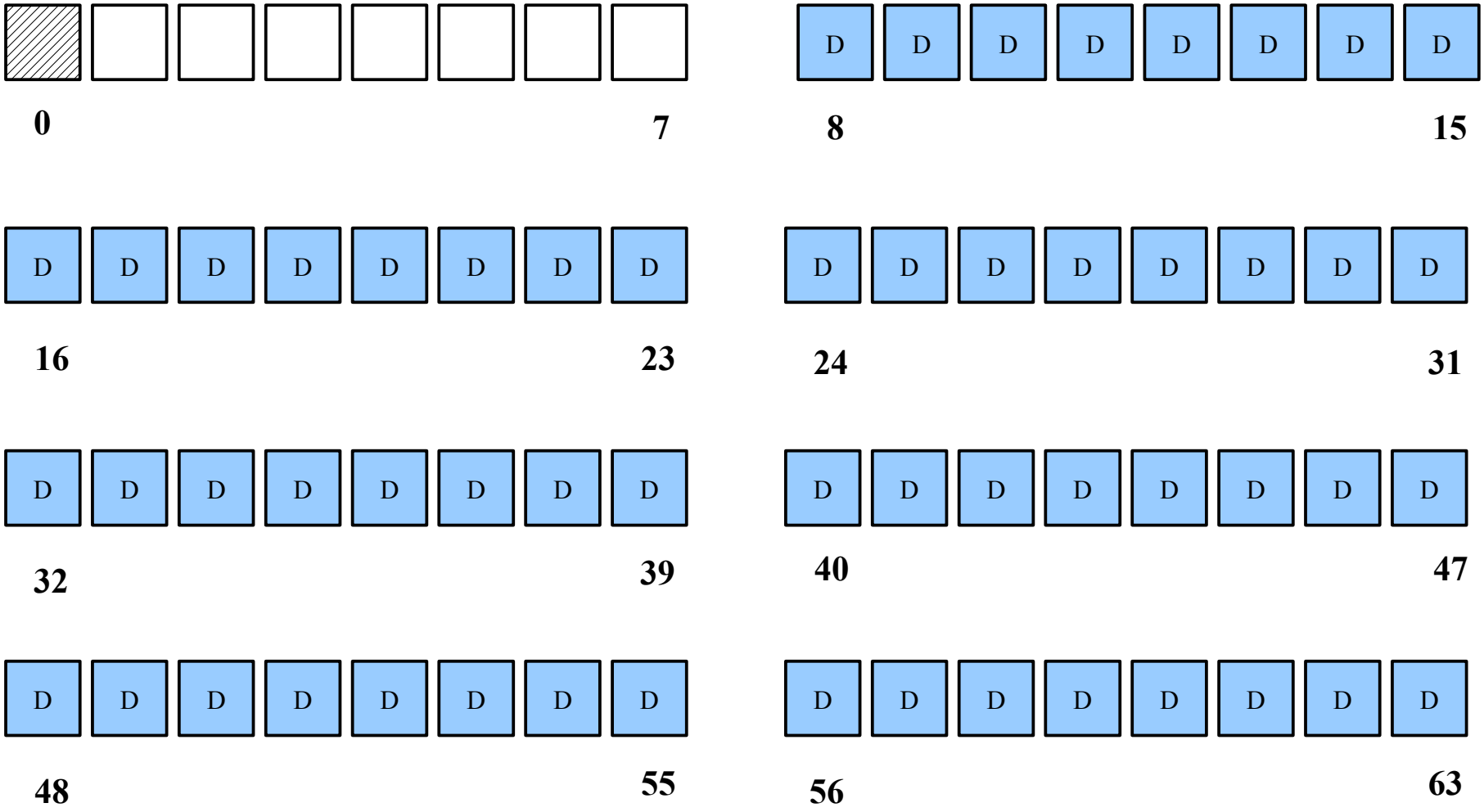
Exemple de structure simple d'un FS

Le bloc 0 est réservé : table de partition, secteur de boot du disque, index de bloc libre dans les inodes...

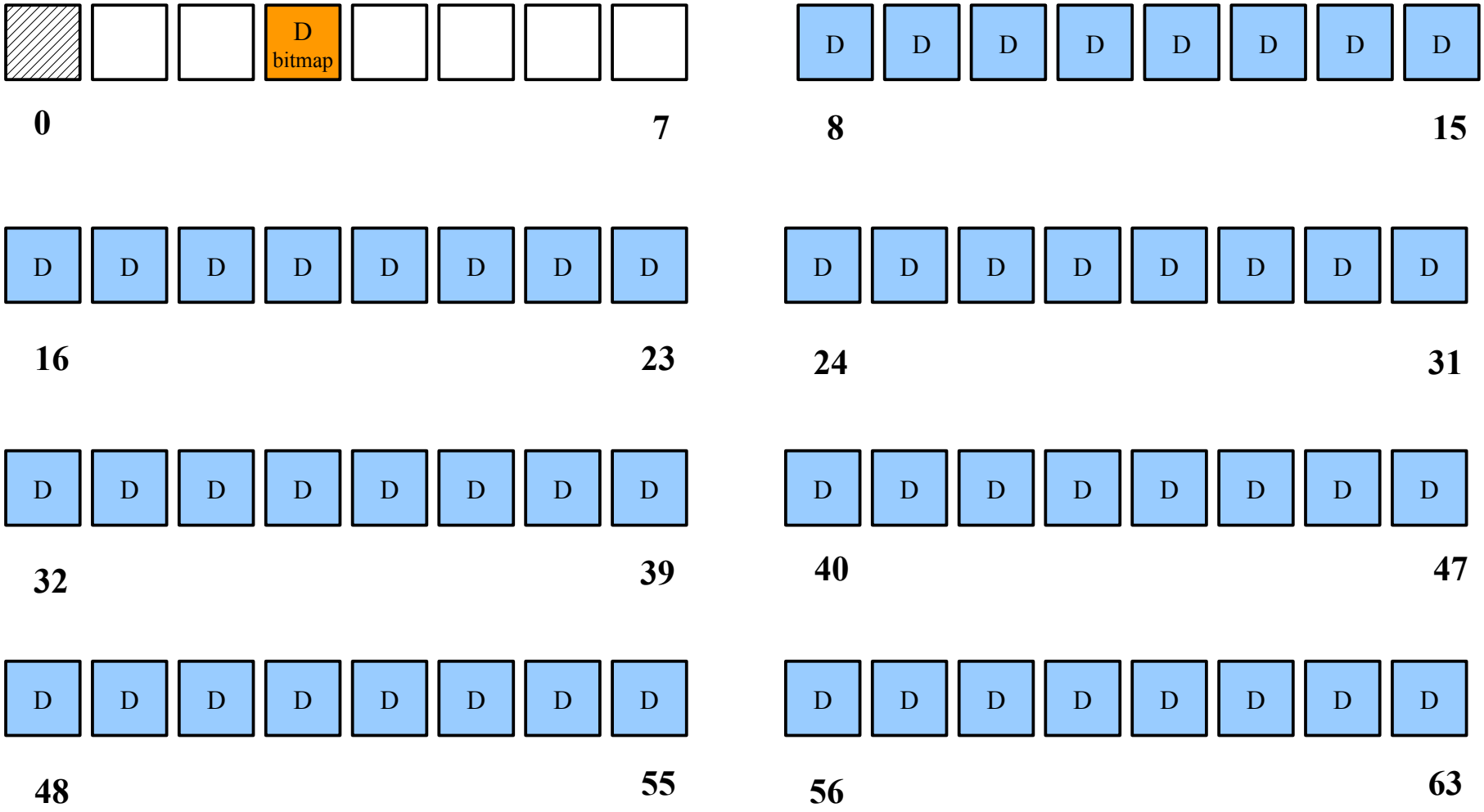


Blocs de données

Si on doit stocker des données, c'est préférable que la majorité des blocs soient utilisé pour cela...

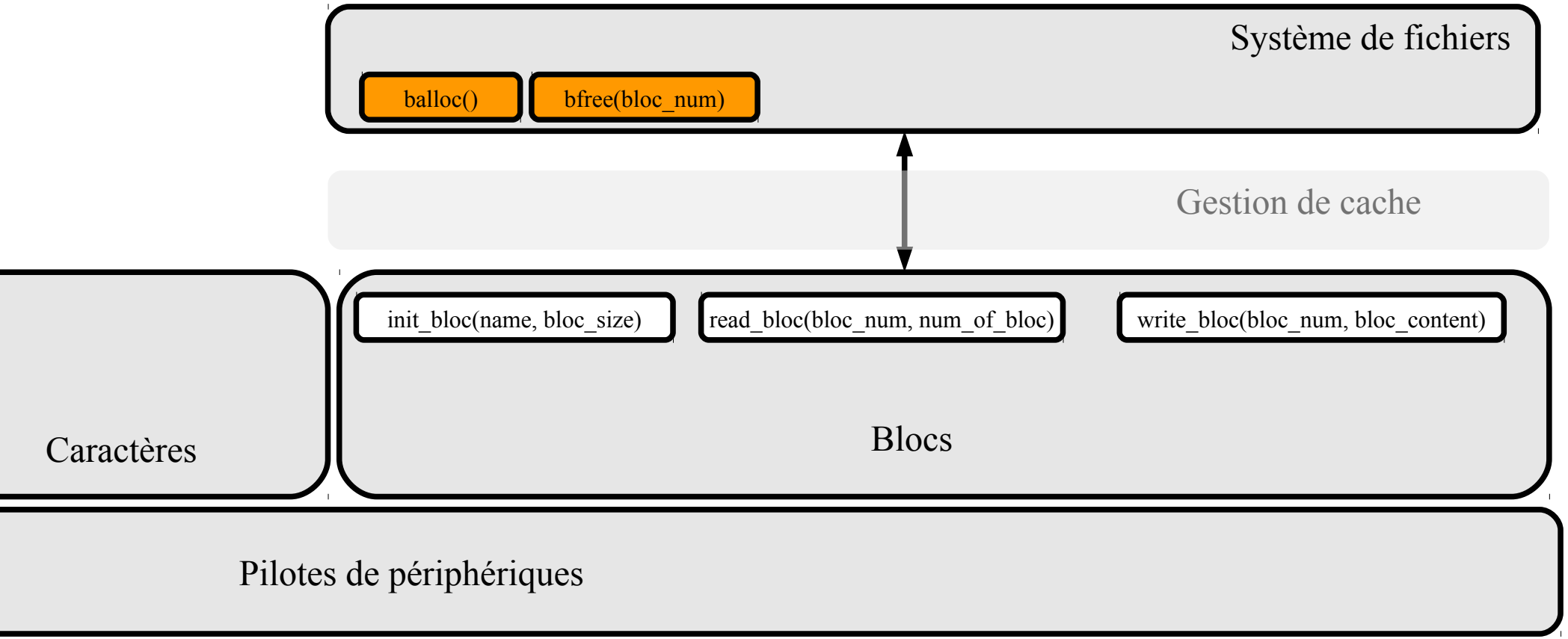


Allocation des blocs de données



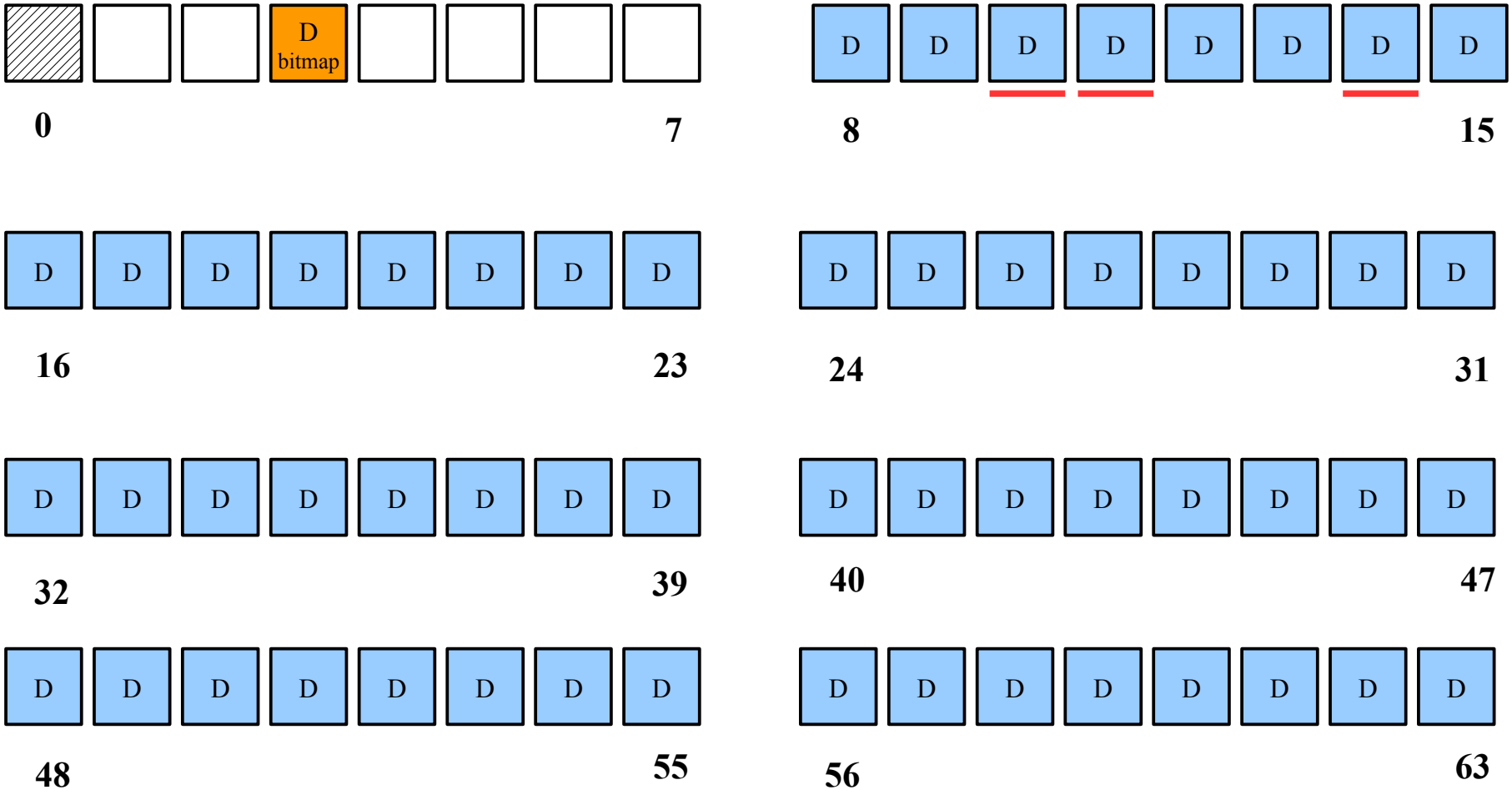
Allocation des blocs de données

Deux nouvelles fonctions :



Exemple de bitmap

Taille du bitmap : `56` bits, le `3ème`, `4ème` et `7ème` blocs sont occupés tout le reste est libre : `00110010 0.....0`



Taille maximum du système de fichier en octets si la taille des blocs = 1024 octets, et 3 blocs consacrés à une allocation par bitmap ?

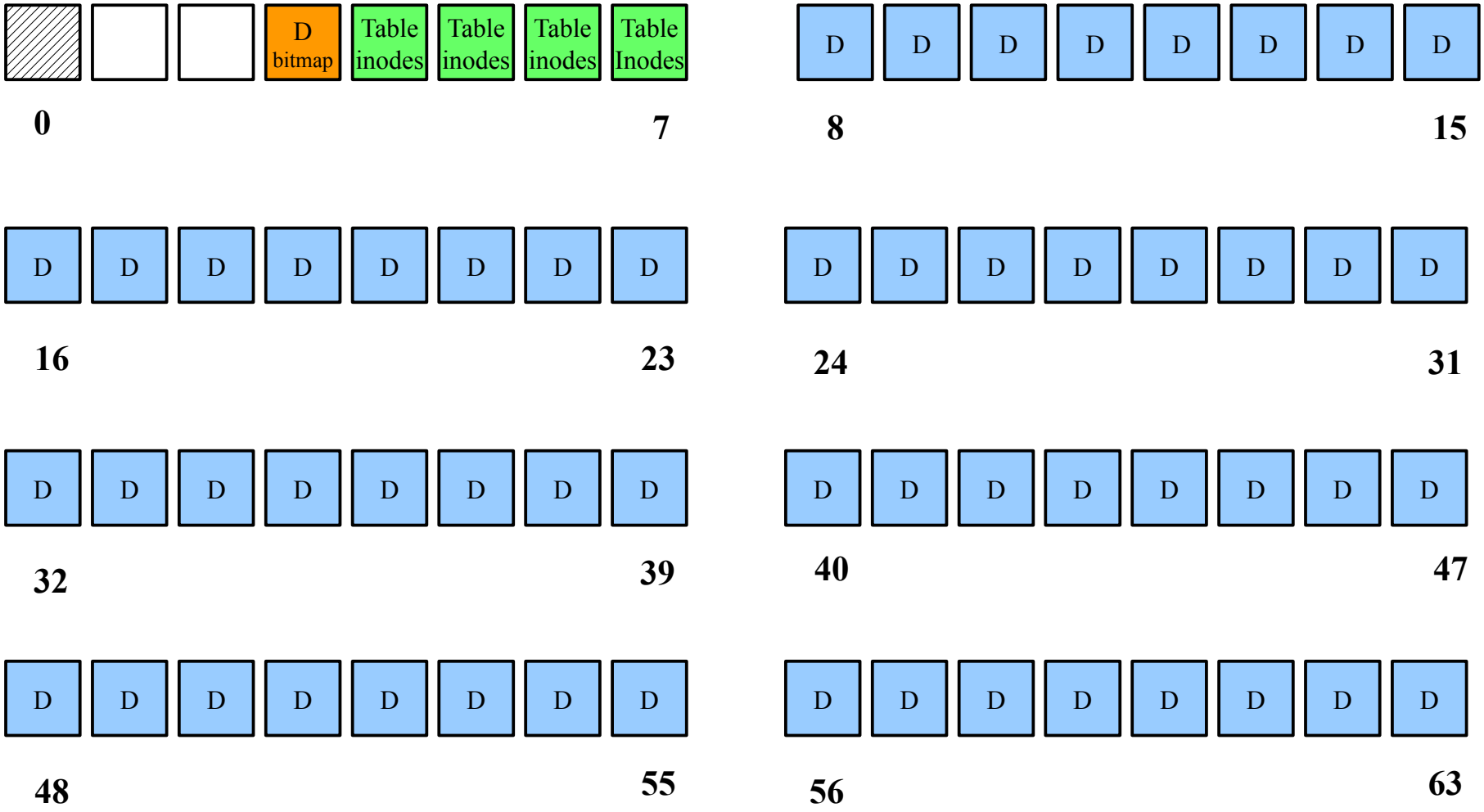
Allocation des blocs de données

Une bitmap pour allouer les blocs n'est qu'un exemple, d'autres structures d'allocation de blocs existent : listes, arbres, etc

On peut maintenant allouer/libérer des blocs du périphérique, mais ce qu'on veut c'est aussi allouer un ensemble précis de bloc à un/des fichiers

Association blocs de données <-> fichiers

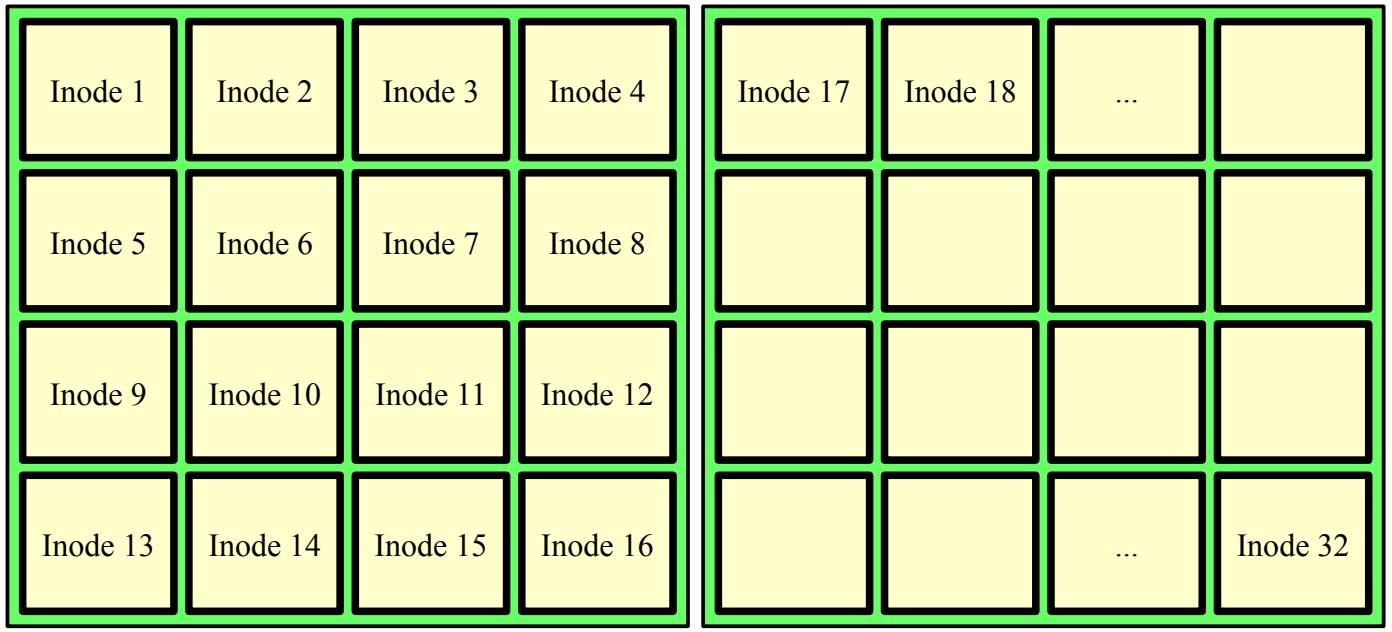
C'est la fonction principale des **inodes**



Les inodes

Leur taille pour un système de fichier donné est fixe, mais selon le système de fichier elle peut varier entre128 et 256 octets, soit entre 16 et 32 Inodes par blocs pour des blocs de 4096 octets

Les inodes sont Indexés à 1



Les inodes : relations par rapport aux fichiers

- Chaque fichier a un seul numéro d'inode

- Chaque inode est unique (à un instant donné) dans un système de fichier.

- Des systèmes de fichiers différents peuvent utiliser les mêmes numéros d'inodes. Ce numéro peut-être recyclé après un « effacement » du fichier

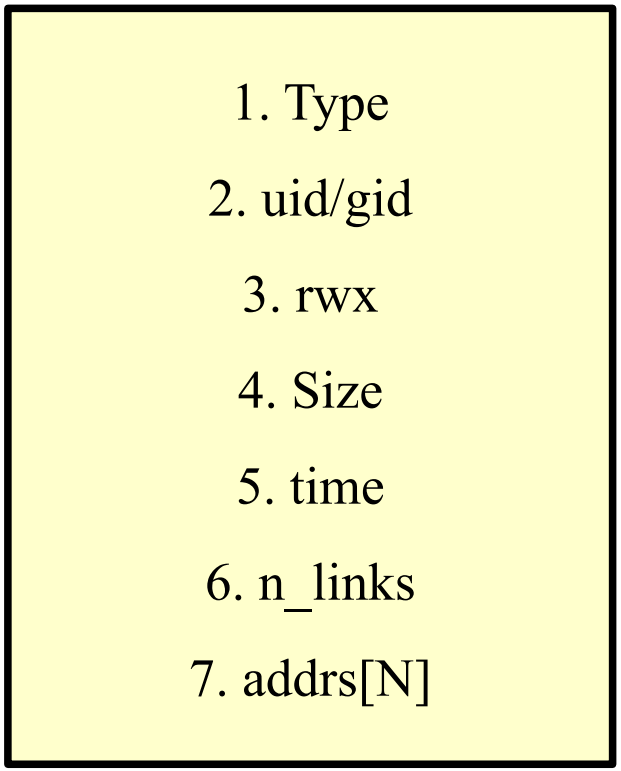
Les inodes : origine de leur nom

« In truth, I don't know either. It was just a term that we started to use. ‘Index’ is my best guess, because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk... »

Dennis Ritchie

Les inodes : leur structure

Structure typique, mais qui peut varier en fonction du système de fichier :



1. Fichier, répertoire ou autre ?

2. Numéro du propriétaire

3. Droits d'accès

4. Taille en octet

5. Date de dernier accès

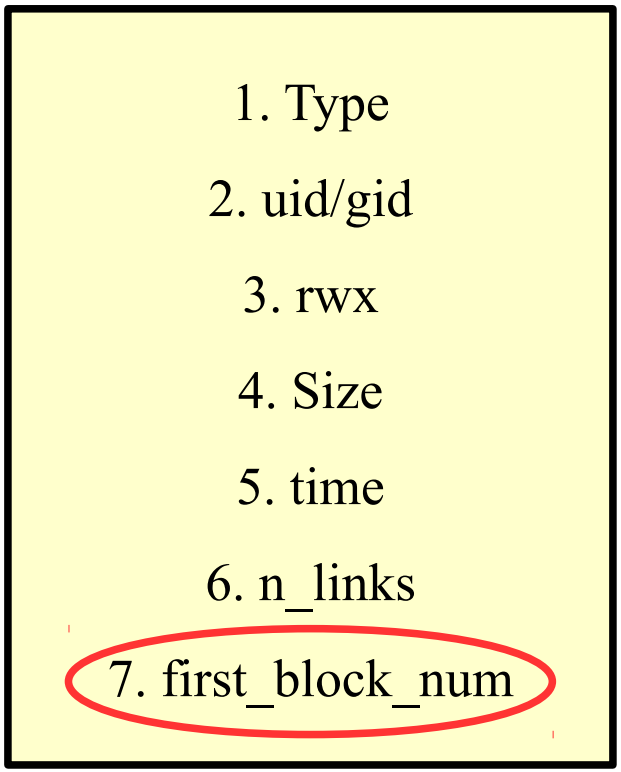
6. Nombre de liens/chemins d'accès vers l'inode.

7. Numéro des blocs du fichiers, dans l'ordre, N < 20

1. Quelle information du fichier manque ?
2. On aurait pas un problème avec le champs 7 ?

Les inodes : Allocation des blocs de données

Pourquoi la structure alternative suivante est une mauvaise idée ?



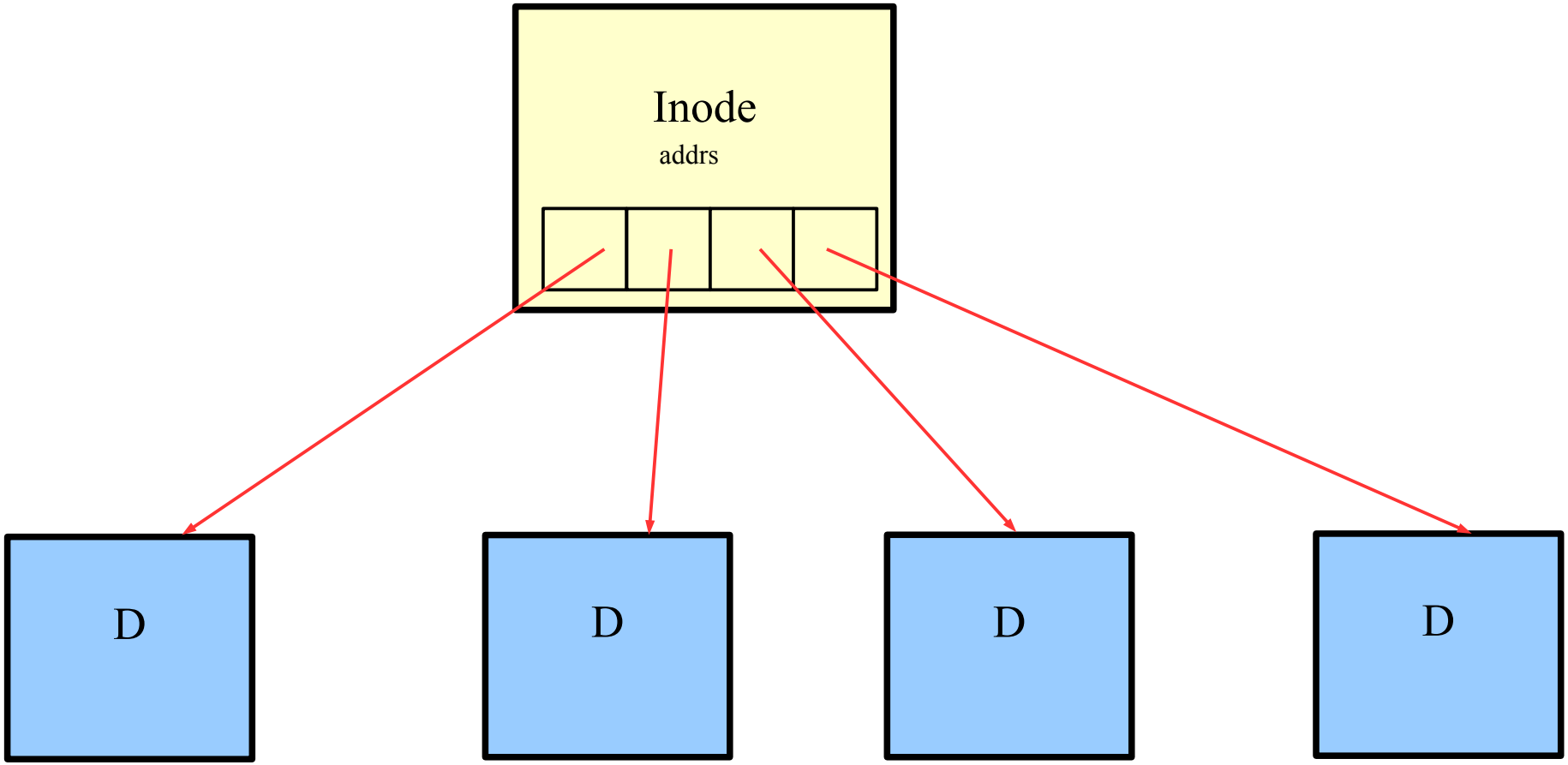
7. Numéro du premier bloc

Les inodes : Allocation des blocs de données

Considérons le cas suivant et supposons que B doit augmenter de taille :

Conséquences ?

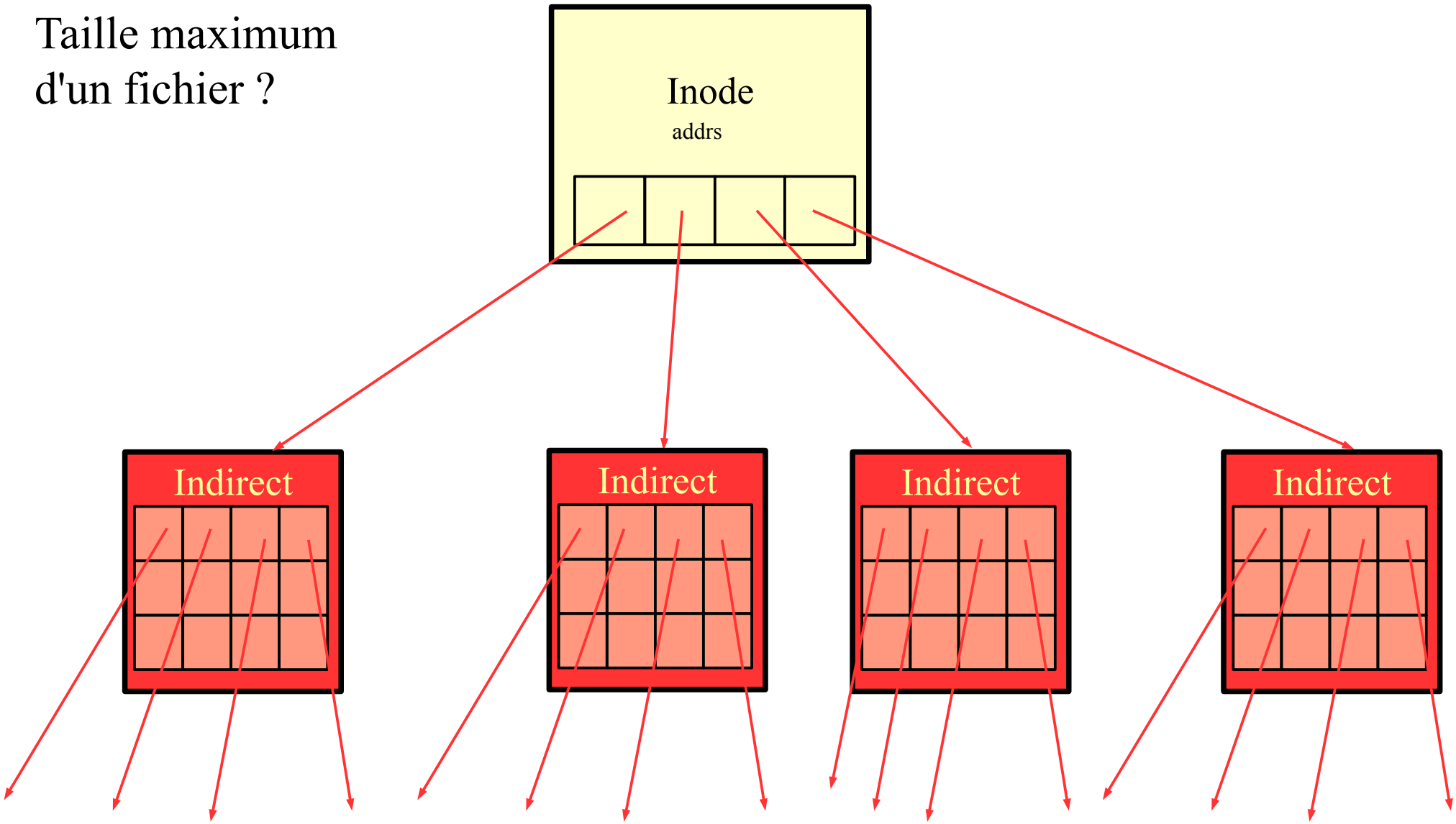
Les inodes : Adressage de blocs direct



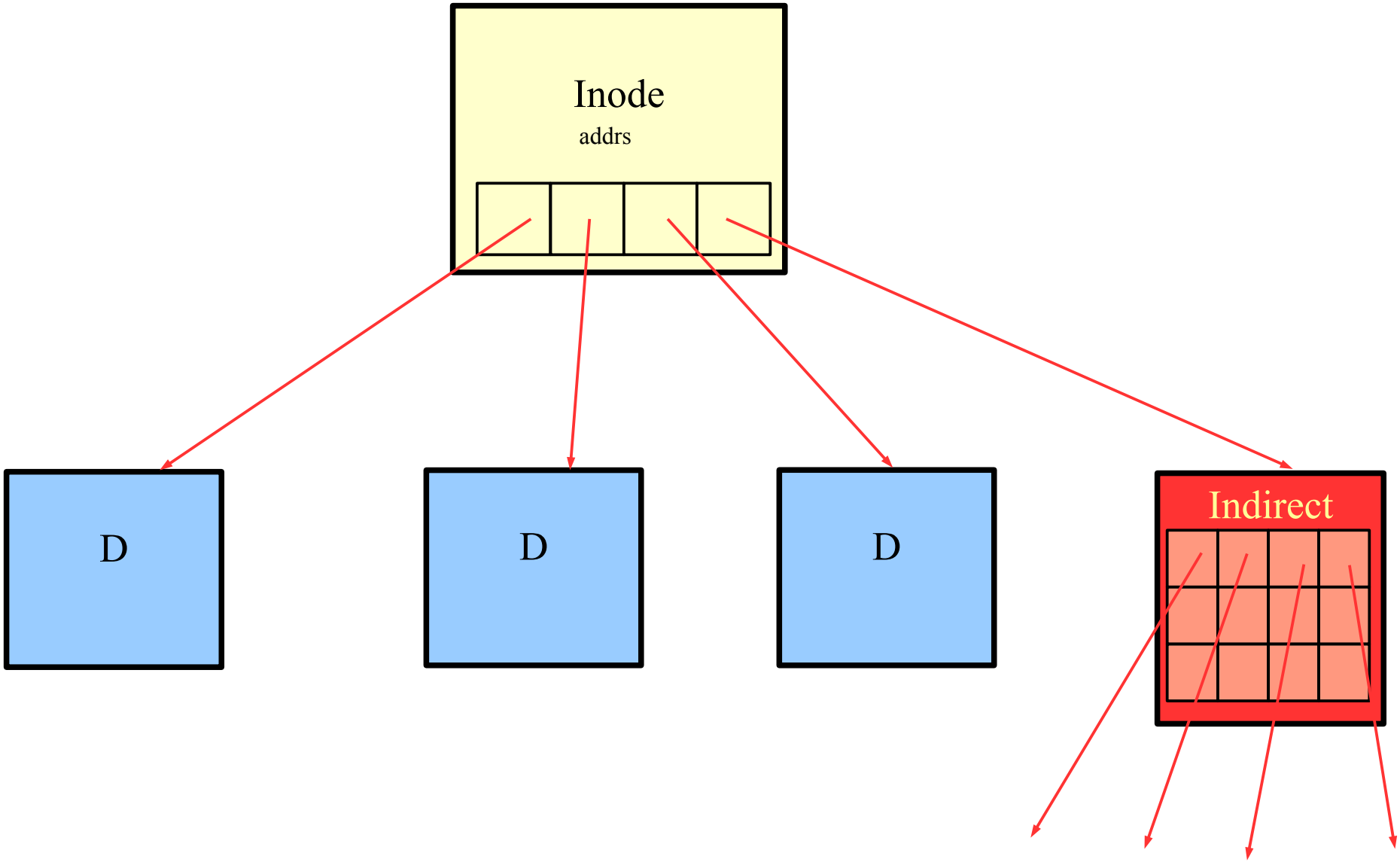
Un numéro de bloc à zéro indique que le bloc n'est pas utilisé
Taille maximum d'un fichier ?

Les inodes : Adressage de blocs indirect

Taille maximum
d'un fichier ?

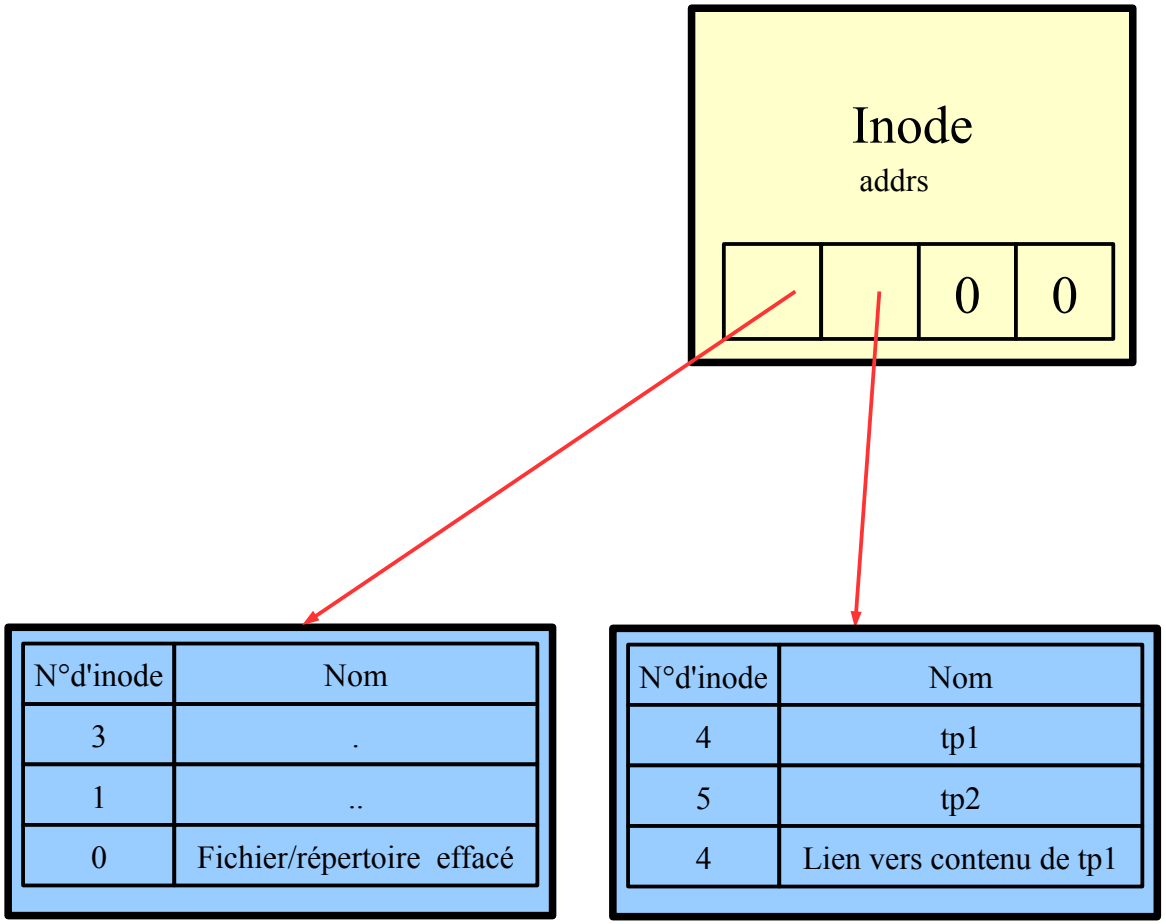


Les inodes : Optimisation pour les petits fichiers



Les inodes : Répertoires

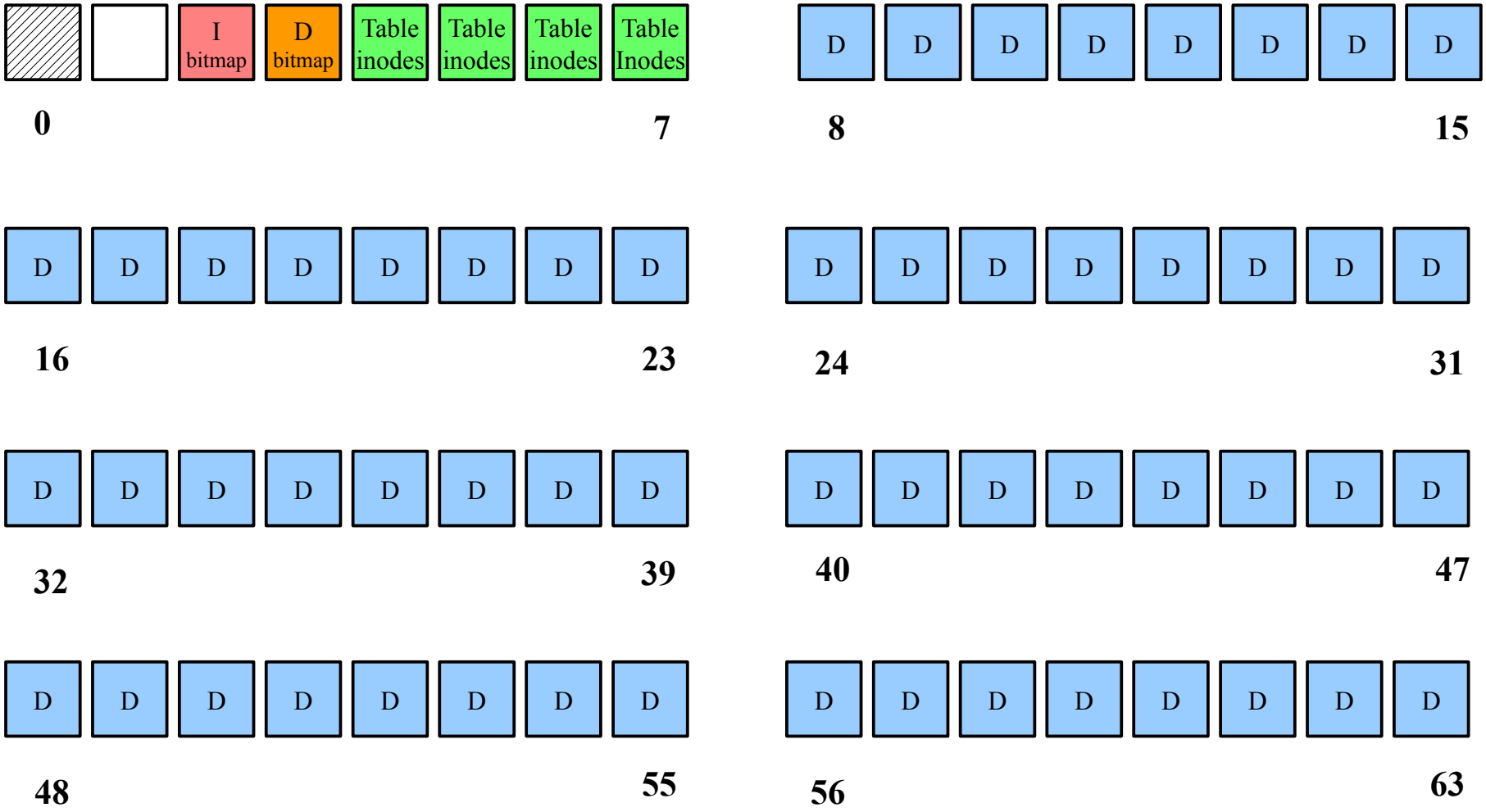
Type de fichier particulier qui associe un nom à un numéro d'inode, "répertoire" vient en fait de l'anglais directory, qui veut aussi dire annuaire...



Par convention, l'inode numéro 1 pointe souvent sur la racine du système de fichier : « / »

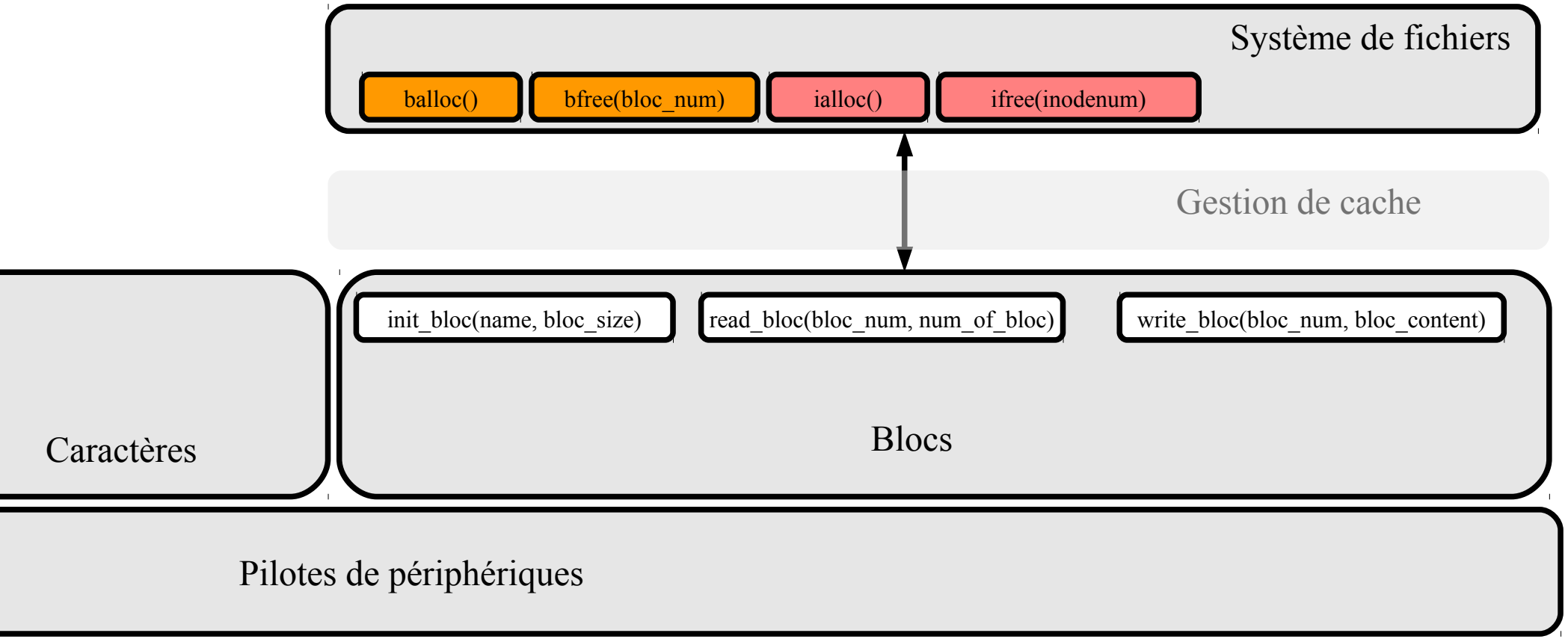
Allocation des inodes

On peut procéder comme pour les blocs : une bitmap, encore une fois ce n'est qu'un exemple



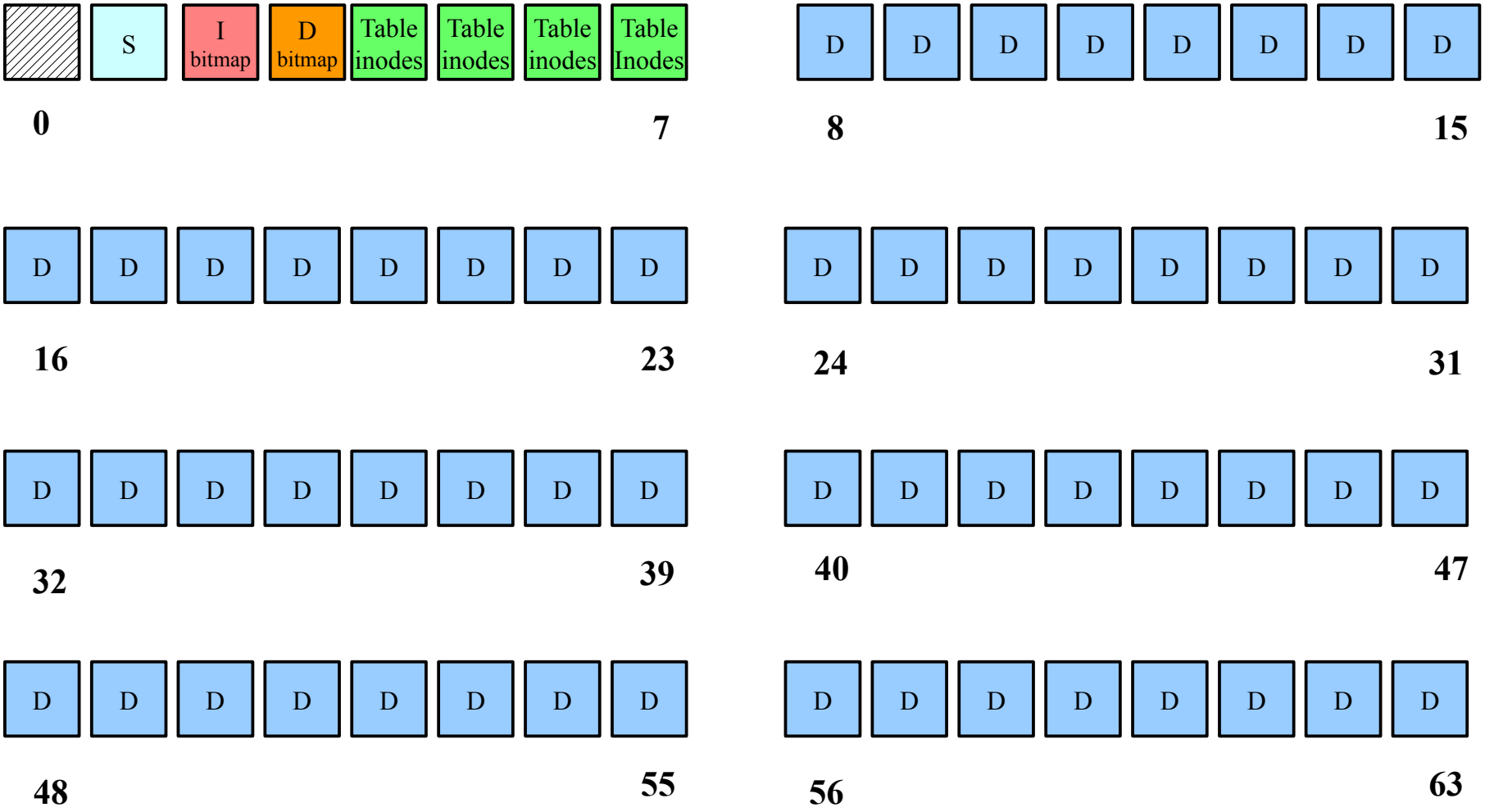
Allocation des inodes

Deux nouvelles fonctions :



Le superbloc

Il stocke des informations globales sur le système de fichier: nombre d'inodes total, nombre de blocs, taille de la zone des bitmaps, signature pour la version, début de la zone de données, etc...



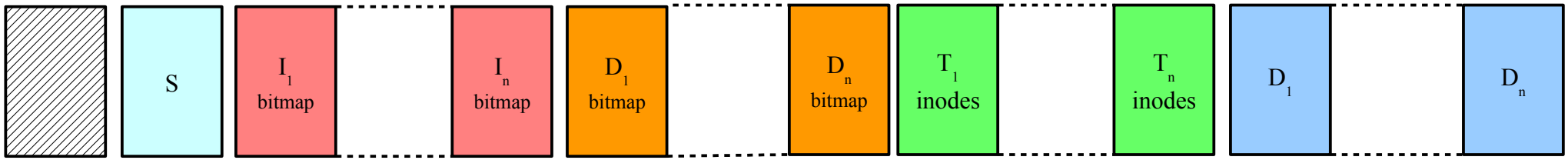
Contenu

Introduction

Exemple d'une structure simple d'un système de fichier sur le disque

Etude de cas : Minix fs 1.0

Minix-fs v1.0 : structure générale



Minix-fs v1.0 : structure des données

Inodes, taille : 32 octets

Attention : Minix a été conçu sur une architecture little endian ! Les structures de données du disque sont donc normalement ordonnées dans ce sens.

```
struct minix_inode {
        u16 i_mode;             /* File type and permissions for file */
        u16 i_uid;              /* user id */
        u32 i_size;             /* File size in bytes */
        u32 i_time;             /* inode time */
        u8  i_gid;              /* group id */
        u8  i_nlinks;           /* number of path pointing to this inode */
        u16 i_zone[7];          /* Adresse logique des blocs direct du fichier */
        u16 i_indir_zone;       /* adresse logique d'un bloc contenant des n° de blocs du fichier */
        u16 i_dbl_indr_zone;    /* adresse logique d'un bloc contenant des n° de blocs qui contiennent des n° de blocs */
};
```

Si taille bloc=1kio, 1024/2= 512 numéros de blocs dans un bloc indirect

Entrée d'un répertoire : 16 octets

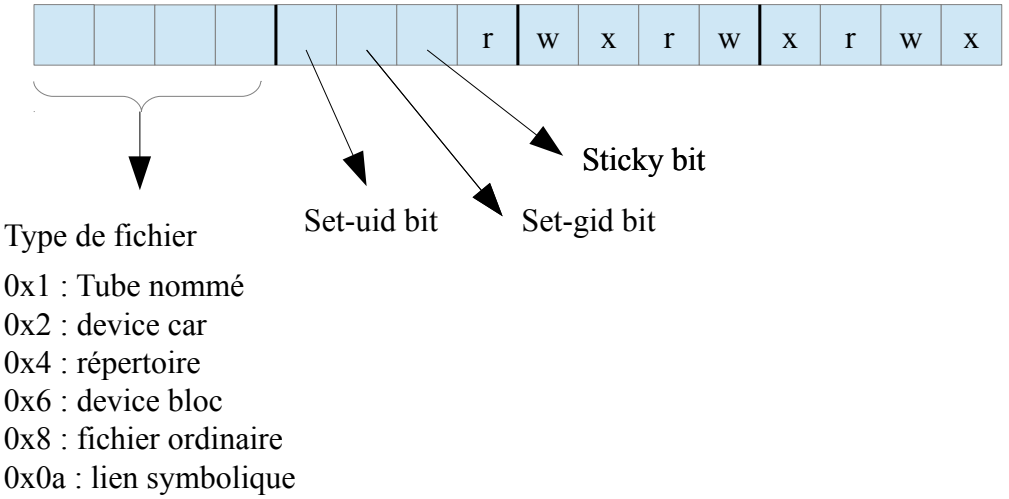
```
#define MINIX_NAME_LEN 14

struct minix_dir_entry {
    u16 inode;                          /* numéro d'inode */
    char name[MINIX_NAME_LEN];          /* nom du fichier, 14 octets maximum, encodage des caractères libre (ascii, utf-8, ...)*/
};
```

Si taille bloc=1kio, 1024/16= 64 entrées de répertoire par bloc ensuite on est en adressage indirect

Minixfs 1.0 : mode du fichier

Structure du champs i_mode :



Le type de fichier (et autres informations) peuvent-être consultés avec des macros spécialisées : `S_ISREG`, `S_ISDIR`, ...
Pour plus de détails : `man 2 stat`

Un fichier de type lien symbolique est un bloc qui contient une chaîne de caractère égale à la valeur du lien. Lors de l'analyse d'un chemin d'accès, Si le système rencontre un lien symbolique, son contenu est concaténé avec le chemin déjà parcouru, puis l'analyse recommence au début de ce nouveau chemin

Minix-fs v1.0 : structure du super-bloc

Structure du superbloc : 20 octets

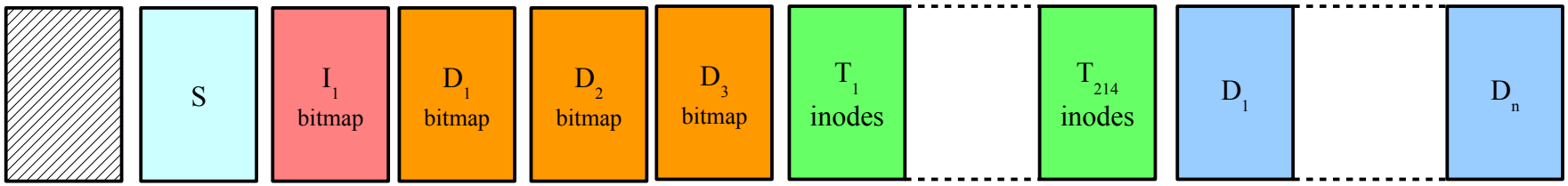
```
struct minix_super_block {
   u16 s_ninodes;          /* Total Number of i-nodes */
   u16 s_nzones;           /* Nombre de blocs total, tout compris, y compris bloc 0 */
   u16 s_imap_blocks;      /* Taille de la bitmap des inodes en blocs (I bitmap) */
   u16 s_zmap_blocks;      /* Taille de la bitmap des zones libres/occupée en blocs (D bitmap)*/
   u16 s_firstdatazone;    /* Index du premier bloc avec des données (D) */
   u16 s_log_zone_size;    /* taille d'un bloc données en octets = 1024*2^s_log_zone_size */
   u32 long s_max_size;    /* Taille maximum d'un fichier en octets */
   u16 s_magic;            /* 0x137f Minix filesystem version 1, si on lit 7f13, on est sur architecture big endian*/
   u16 s_state;            /* as-t-il été proprement démonté ? */
};
```

Taille maximum du système de fichier en Mio, avec une taille de bloc de 1kio ?

Minix-fs v1.0 : Exemple

Partition de 20 Mio, taille de bloc de 1024 octets

```
struct minix_super_block {
   u16 s_ninodes = 6848;          /* Au maximum 6848 fichiers pourront être crées */
   u16 s_nzones  = 20480;         /* 20480 blocs de 1024 octets, soit 20 Mio */
   u16 s_imap_blocks = 1;         /* 1 bloc bitmap d'inode libres, peuvent représenter 8192 inodes */
   u16 s_zmap_blocks = 3;         /* 3 blocs de bitmap de blocs libres peuvent représenter 24576 blocs */
   u16 s_firstdatazone = 220;     /* Note : bloc 0 + 6848/32 = 214 blocs d'inodes + 1 bloc de superbloc
                                     + 4 blocs de bitmap (1+3)  = 220 blocs numérotés de 0 à 219 */
   u16 s_log_zone_size = 0;       /* taille d'un bloc données en octets = 1024*2^s_log_zone_size */
   u32 long s_max_size = 268966912; /* Taille maximum d'un fichier en octets si taille bloc = 1024 bytes :
                                     1024*7+512*1024+512*512*1024 */
                                  *  car un bloc indirect/double indirect peut contenir 512 index de blocs
   u16 s_magic = 0x137F;          /* 0x137f Minix filesystem version 1 */
   u16 s_state    = 1;            /* as-t-il été proprement démonté ? */
};
```



Minix fs 1.0 : Exemple

Contenu de l'inode 15

```
struct minix_inode {
        u16 i_mode =  0x81c9 (08711 en octal);        /* Fichier ordinaire  rwx --x --x*/
        u16 i_uid    = 0;                    /* user id = 0, c'est root */
        u32 i_size   = 283652;               /* File size in bytes, soit 278 blocs de 1024 octets */
        u32 i_time  = 1425332938 ;           /* nombre de secondes écoulée depuis le 1/1/1970 */
        u8  i_gid     = 0 ;                  /* group id */
        u8  i_nlinks = 2;                    /* number of paths pointing to this inode */
        u16 i_zone[7] = {234, 235, 236, 237, 238, 239, 240 }        /* Adresse logique des blocs direct du fichier */
        u16 i_indir_zone = 241;              /* le bloc 241 contient des adresses indirectes, il contient 271 adresses */
        u16 i_dbl_indr_zone = 0 ;            /* pas de bloc double indirects */
};
```

Contenu de l'inode 1 : la racine du système de fichier

```
struct minix_inode {
        u16 i_mode =  0x41ed (04755 en octal);        /* Répertoire   rwx r-x r-x*/
        u16 i_uid    = 0;                    /* user id = 0, c'est root */
        u32 i_size   = 256;                  /* 256/16 : il y a 16 entrées de répertoire potentielles*/
        u32 i_time  = 1425332938 ;           /* nombre de secondes écoulée depuis le 1/1/1970 */
        u8  i_gid     = 0 ;                  /* group id */
        u8  i_nlinks = 8;                    /* number of paths pointing to this inode */
        u16 i_zone[7] = {220, 0, 0, 0, 0, 0,0 }        /* Adresse logique des blocs direct du fichier */
        u16 i_indir_zone = 0;                /* le bloc 241 contient des adresses indirectes, il contient 271 adresses */
        u16 i_dbl_indr_zone = 0 ;            /* pas de bloc double indirects */
};
```

Minixfs 1.0 : Exemple

Contenu du bloc 220, offset = 220*1024 = 0x37000

```
00037000  00 01 2e 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00037010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00037020  00 01 2e 2e 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00037030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00037040  00 02 72 6f 6f 74 00 00  00 00 00 00 00 00 00 00  |..root..........|
00037050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00037060  00 03 62 69 6e 00 00 00  00 00 00 00 00 00 00 00  |..bin...........|
00037070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00037080  00 04 75 73 72 00 00 00  00 00 00 00 00 00 00 00  |..usr...........|
00037090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000370a0  00 09 65 74 63 00 00 00  00 00 00 00 00 00 00 00  |..etc...........|
000370b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000370c0  09 0e 74 6d 70 00 00 00  00 00 00 00 00 00 00 00  |..tmp...........|
000370d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000370e0  00 31 64 65 76 00 00 00  00 00 00 00 00 00 00 00  |.1dev...........|
000370f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Les répertoires `root`, `bin` , `usr`, `etc`, `tmp` et `dev` sont décrits par les inodes respectifs numéros `1,2,3,4,9,14 et 49`

Les noms `.` et `..` pointent sur la racine de numéro d'inode `1`

Le bloc ci-dessus montre des entrées vides (numéro d'inode à 0) intercalées avec des entrées utilisées : une entrée fait 16 octets