

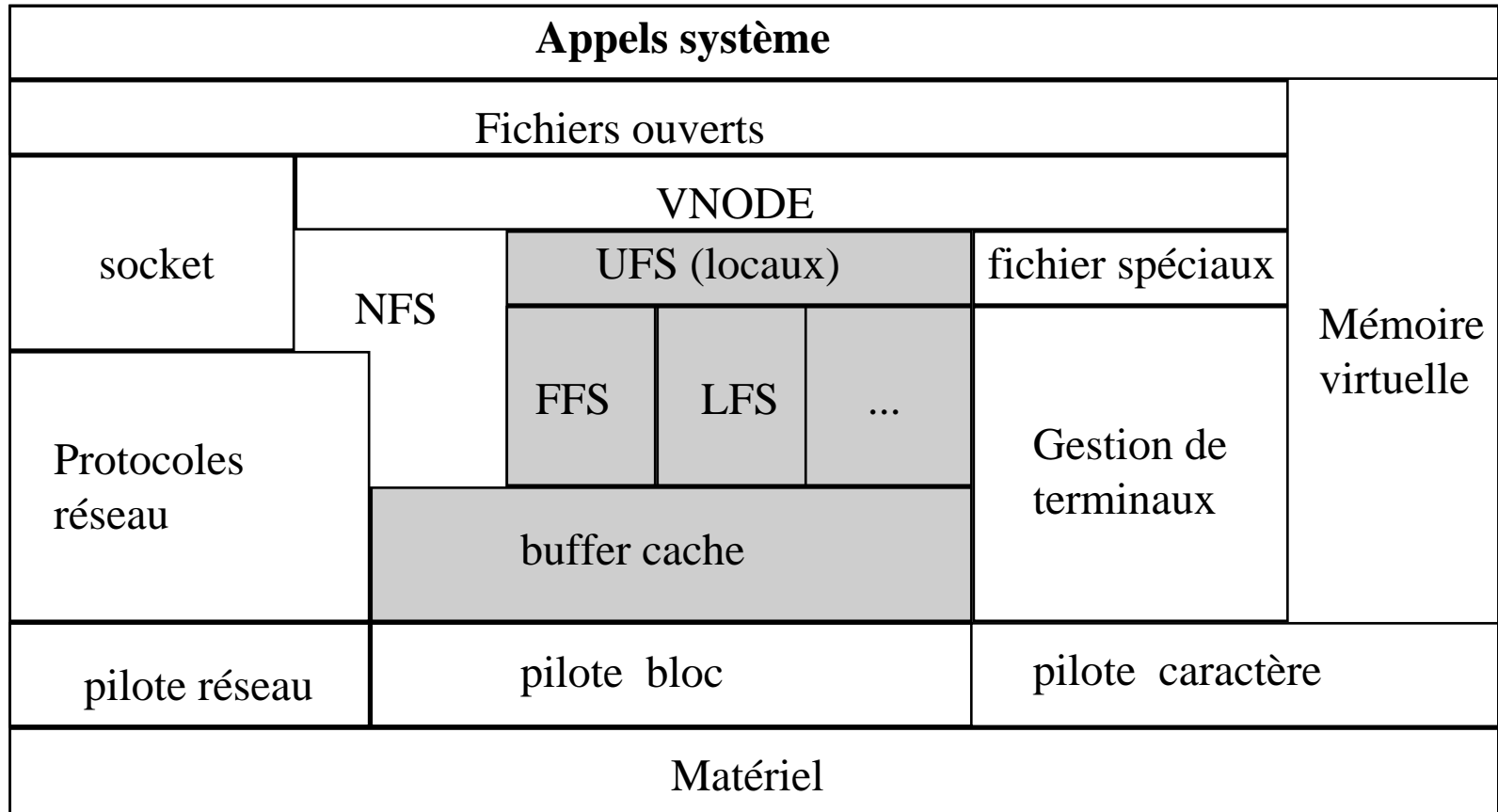
Système de gestion des Entrées/sorties

1- Le sous système d'entrées/sorties

2- Les systèmes de fichiers locaux

3- Les systèmes de fichiers réseaux

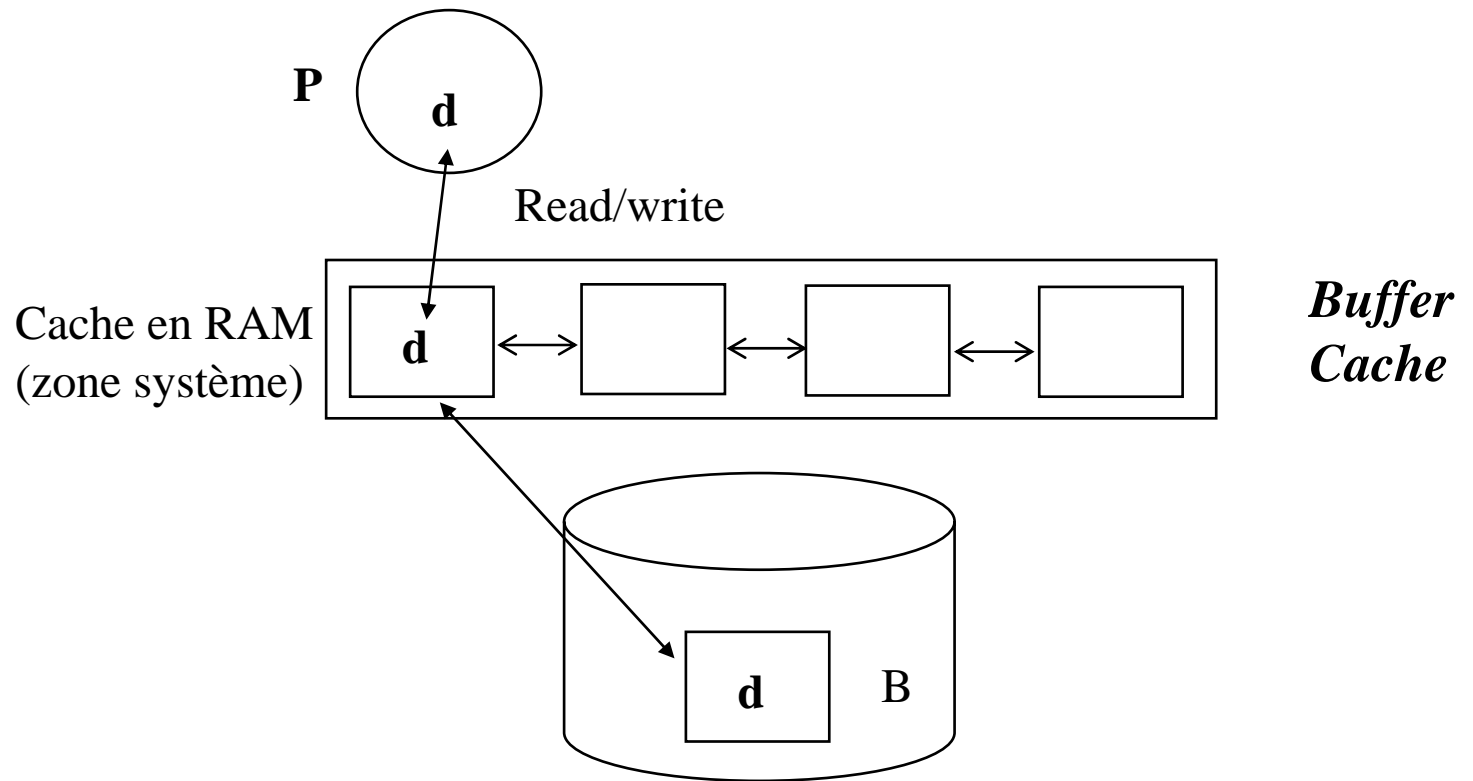
I - Système de fichiers locaux



PARTIE 1 : Cache

Principe du cache

Accès donnée **d** dans bloc B

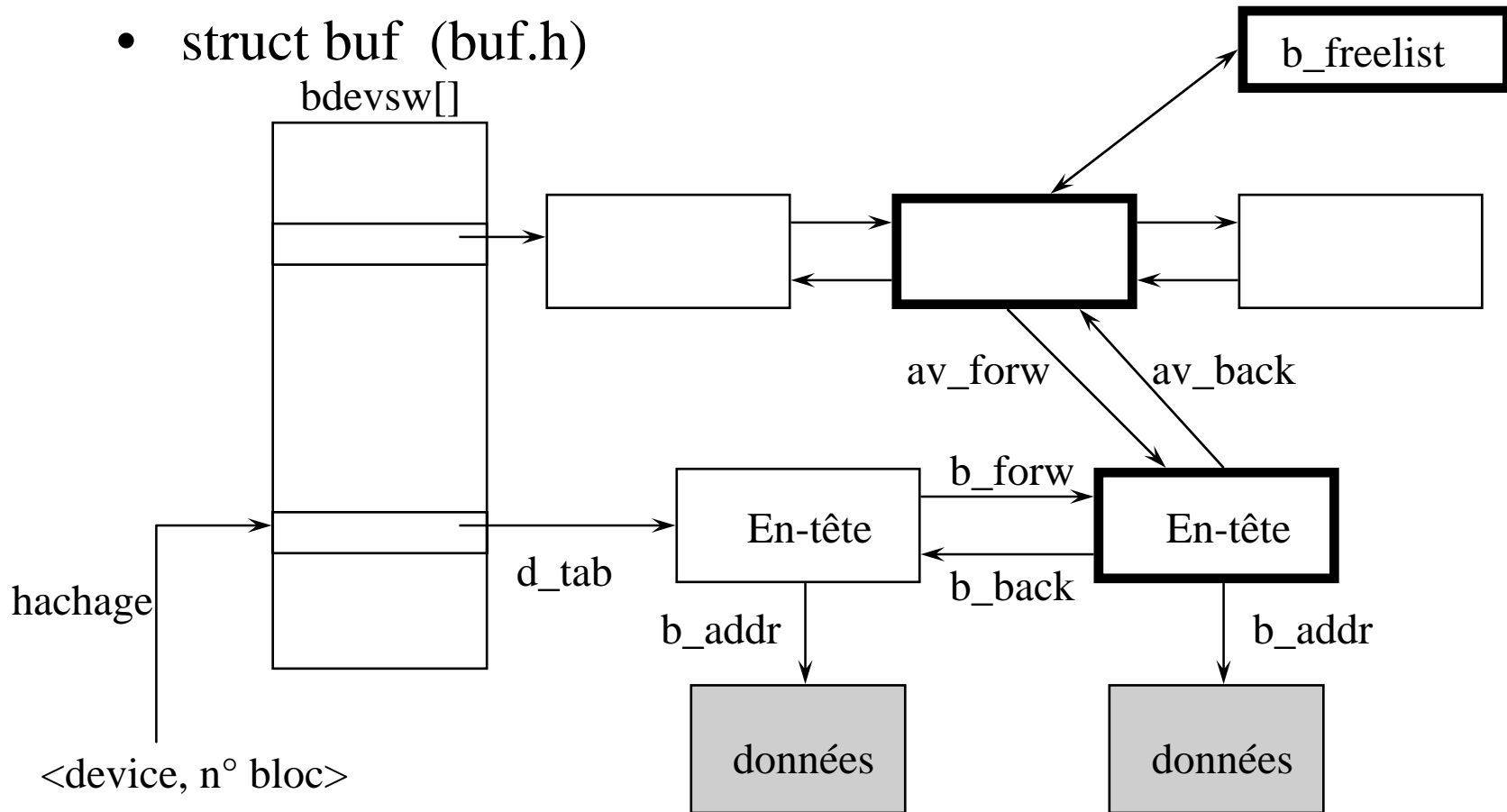


1 - Gestion du cache - le buffer cache

- Principe:
 - Les lectures/écritures par blocs
 - Les bloc sont conservés en mémoire dans une zone du système = buffer cache
- Avantages:
 - Limiter le nombres d'E/S (localité)
 - Dissocier E/S logique et E/S physique (asynchronisme)
 - Anticipation en cas d'accès séquentiel
- Inconvénient:
 - Risque d'incohérence (perte de données) en cas de défaillance

Structure générale

- struct buf (buf.h)



En-tête du buffer cache

- Extraits de struct buf:
 - b_flags : états du bloc
 - *b_forw : pointeur buffer suivant dans le même pilote
 - *b_back : pointeur buffer précédent dans le même pilote
 - *av_forw : pointeur buffer libre suivant (dans la b_freelist)
 - *av_back : pointeur buffer libre précédent
 - b_addr : pointeur vers les données
 - b_blkno : numéro logique du bloc
 - b_error : code de retour après une E/S

Etats d'un buffer

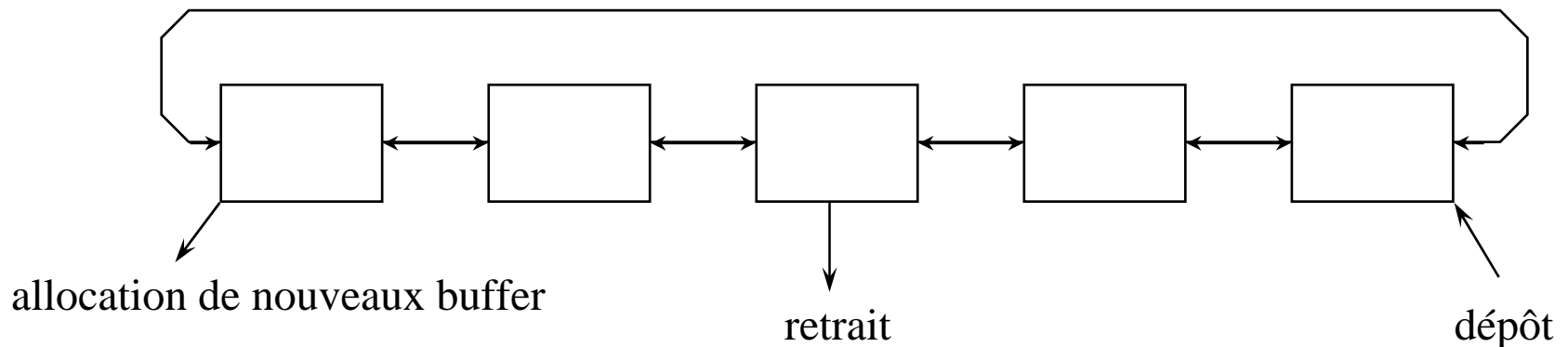
- Valeurs du champs b_flags
- Disponible : pas d'E/S en cours => dans la b_freelist
- Indisponible (B_BUSY positionné dans b_flags)
 - B_DONE : E/S terminée
 - B_ERROR : E/S incorrecte
 - B_WANTED : désiré par un processus (réveiller en fin E/S)
 - B_ASYNC : ne pas attendre fin E/S (E/S asynchrone)
 - B_DELWRI : retarder l'écriture sur disque (tampon «sale»)

Les primitives

- Lecture d'un bloc : `bread`
- Lecture par anticipation d'un bloc : `breada`
- Ecriture d'un bloc : `bdwrite` (buffer *delayed* write)
- Ecriture par anticipation : `bawrite`
- Ecriture synchrone : `bwrite`
- Libération d'un buffer : `brelease`
- Recherche ou allocation d'un buffer : `getblk`

Gestion des tampons

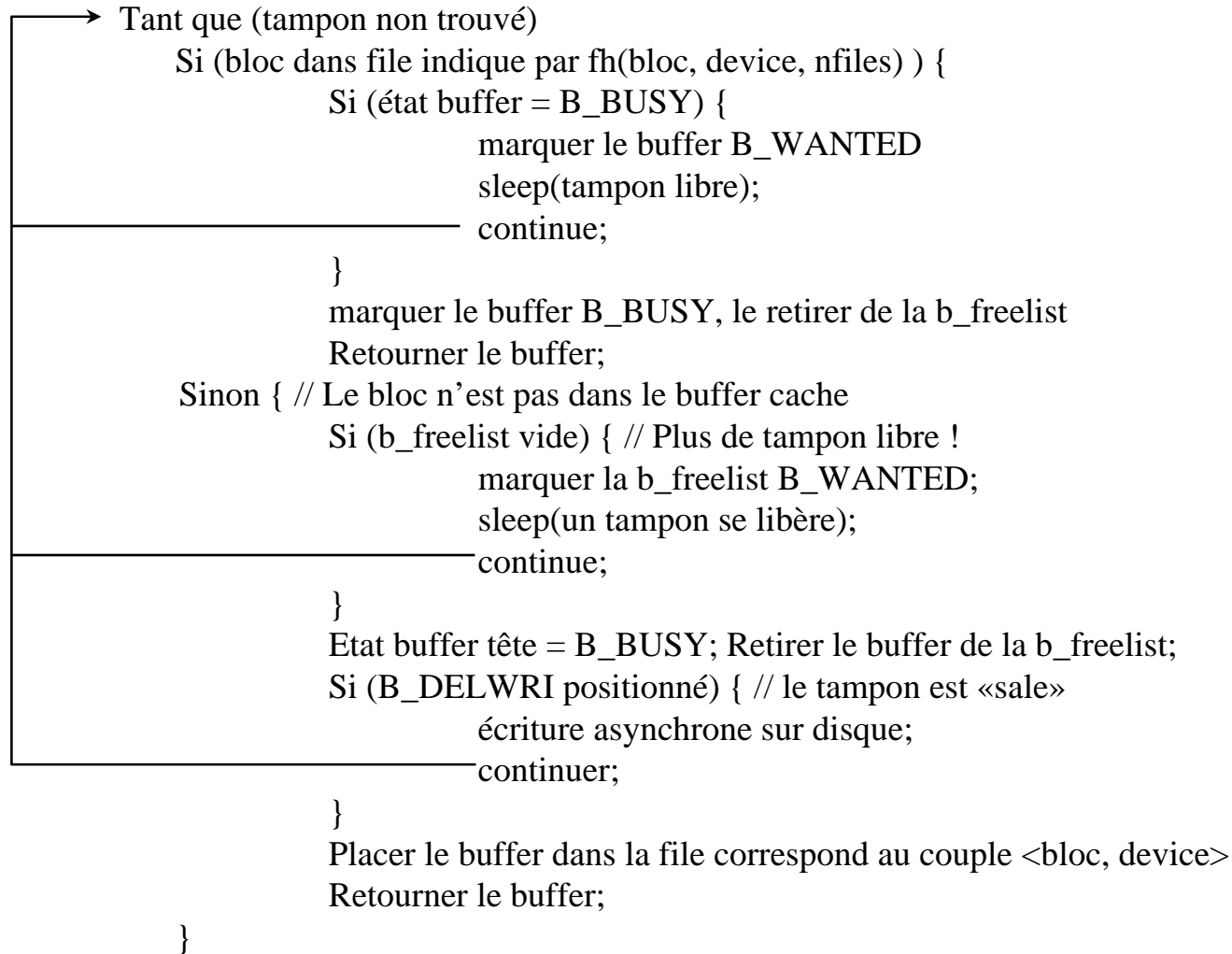
- Liste des buffer libres : listes circulaires avec gestion LRU



- Accès à un buffer par hash-coding
 - $fh(b_dev, b_blkno, \text{nombre de files})$
 - Distribution uniforme des tampon dans les files

Recherche/Allocation de buffer (getblk)

Entrées : numéro de bloc, device



Libération d'un buffer (brelse)

- Réveiller tous les processus en attente qu'un buffer devienne libre
- Réveiller tous les processus en attente que ce buffer devienne libre
- Masquer les interruptions
- Si (contenu du buffer valide)
 - mettre le tampon en queue de la b_freelist
- Démasquer les interruptions
- retirer bit B_BUSY

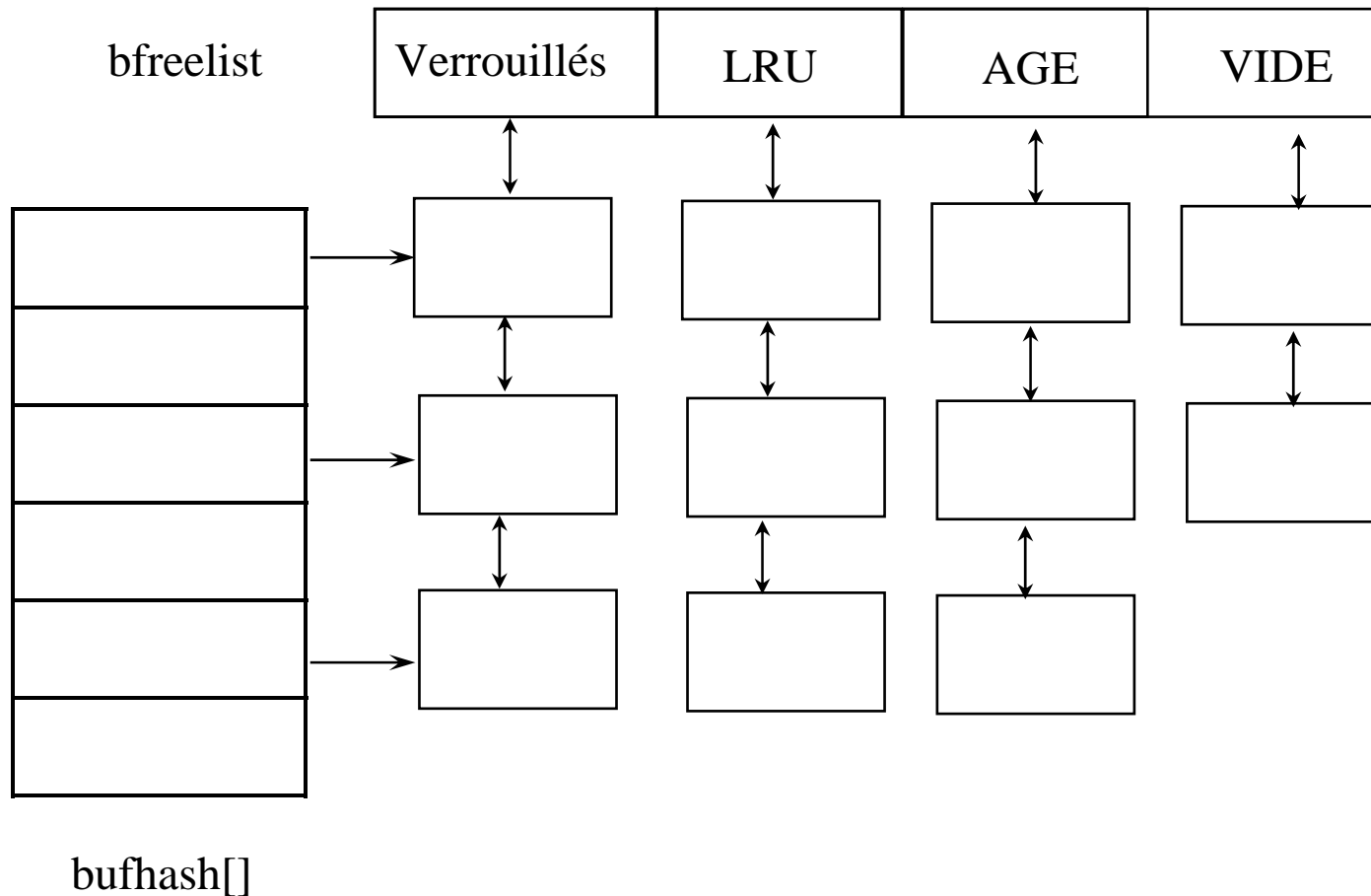
Lecture d'un buffer (bread)

- **Entrées** : device, bloc
- Rechercher ou allouer le bloc (getblk)
- Si (buffer valide et B_DONE)
 retourner le tampon
- Lancer une lecture sur disque (appel du pilote - strategy)
- sleep(attente fin E/S)
- retourner le buffer

Ecriture d'un buffer sur disque

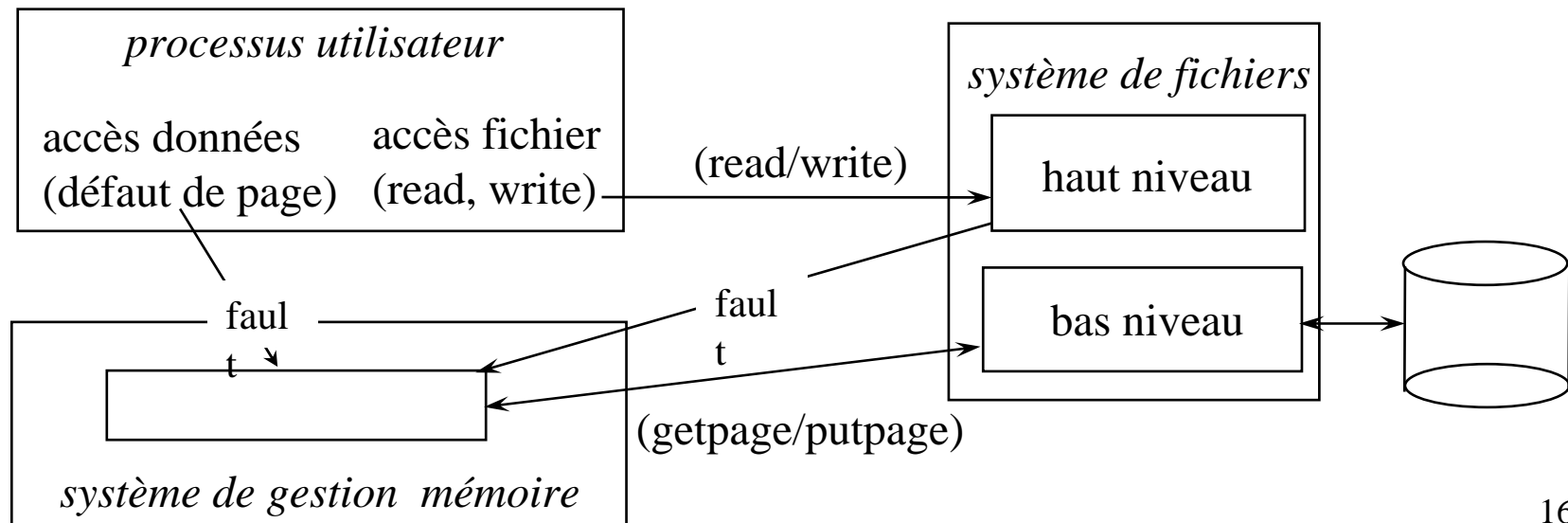
- Bdwrite
Positionnement de B_DELWRI pour E/S asynchrone
- Besoin de place
Dans getblk: Si le bloc n'est pas dans le cache
=> allouer un nouveau buffer
Si B_DELWRI => Ecriture
- Régulièrement **sync** parcourt la liste des buffers
Si B_DELWRI => Ecriture

Organisation du buffer cache (BSD)



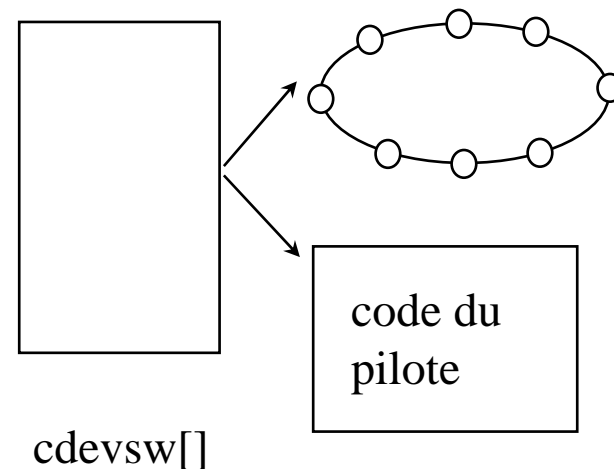
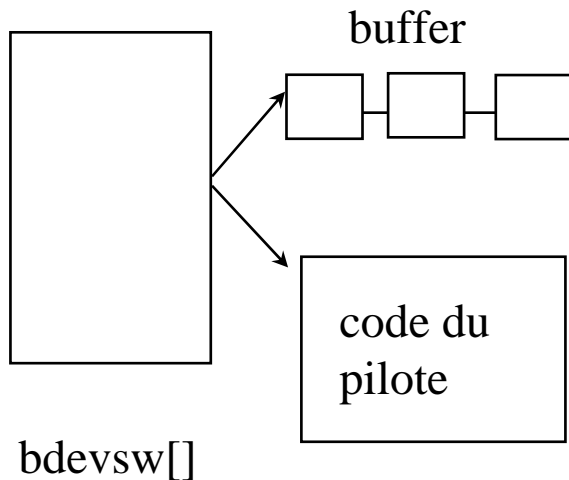
Mémoire virtuelle et buffer cache

- Mécanismes très voisins
(cases => tampons, swap => fichier)
- Buffer cache intégré dans la pagination (SunOs, SVR4)
 - Cases pour les pages **et** les tampons
 - Fichier correspond à une zone de mémoire virtuelle (seg_map)
 - lecture d'un bloc non présent => **défaul de page**



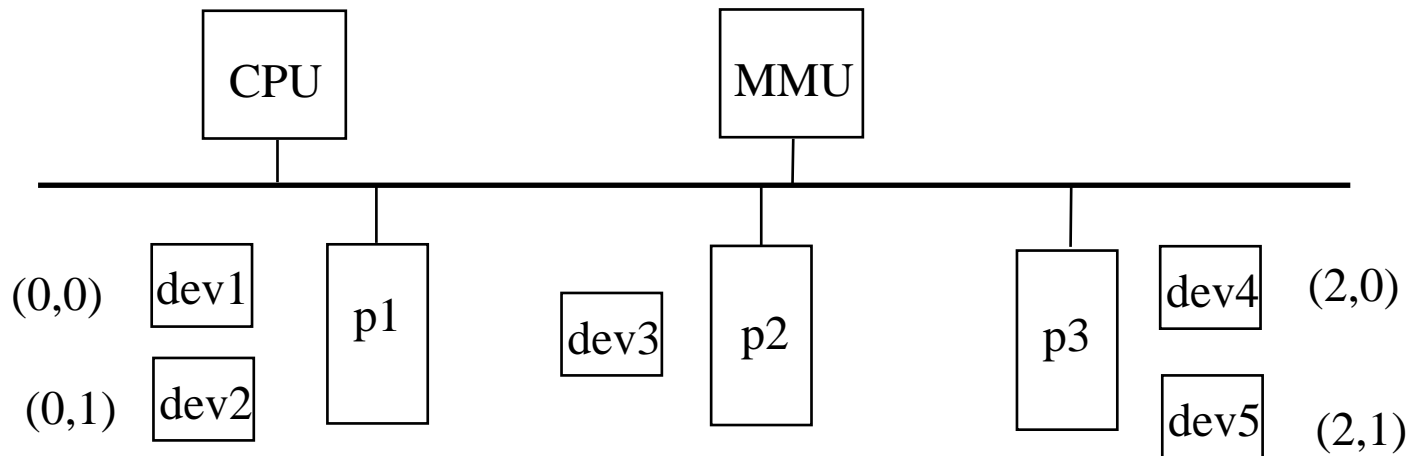
2 - Les Entrées/sorties

- Les types de périphérique
- Mode bloc : accès direct + structuration en bloc
- Mode caractère : accès séquentiel, pas de structuration des données (flux)



Les pilotes de périphérique

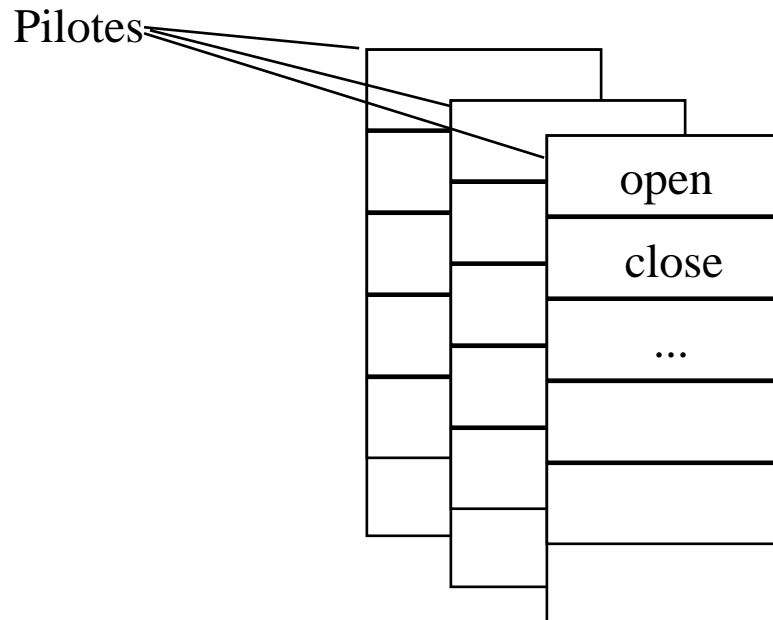
- Configuration typique



- Adresse logique :
 - majeur : numéro de pilote
 - mineur : numéro d'ordre de l'unité logique

Les tables internes

- Pour chaque pilote un ensemble de fonction (points d'entrées)
- Une table pour chaque pilote (device switch)



Interface structurée

- 2 types de tables : bdevsw (bloc), cdevsw (car.)

Pilotes en mode bloc

- Table bdevsw :


```
struct bdevsw {  
    int (*d_open)();           /* ouverture */  
    int (*d_close)();          /* fermeture */  
    int (*d_strategy)();        /* Tranfert : Lecture/Ecriture */  
    int (*d_size)();            /* Taille de la partition */  
    int (*d_dump)();            /* Ecrire toute la mémoire physique  
                                sur périphérique */  
    ( int *d_tab;               /* Pointeur vers tampon */ )  
    ...  
} bdevsw[];
```

```
(*bdevsw[major(dev)].d_open)(dev, ...);
```

Requêtes d'E/S

- Algorithme ascenseur (C_LOOK) - BSD
- => limiter les déplacements de têtes

position courante



30	34	35	50	100	150
----	----	----	----	-----	-----

liste des requêtes **après** la position courante

2	7	15	17	20	26
---	---	----	----	----	----

liste des requêtes **avant** la position courante

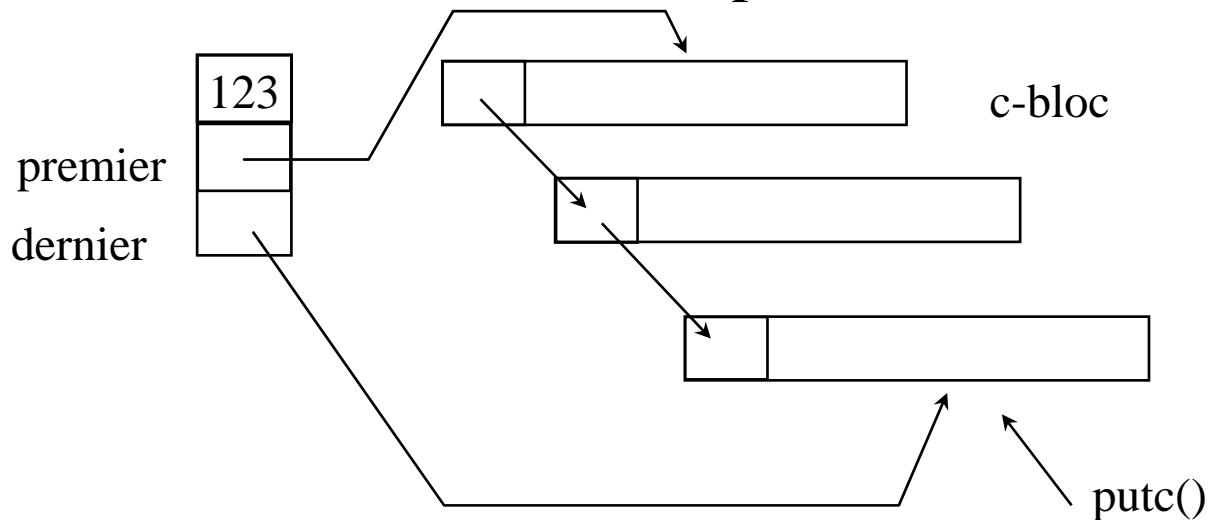
Pilotes en mode caractère

- table cdevsw

```
struct cdevsw {  
    int (*d_open)();  
    int (*d_close)();  
    int (*d_read)();           /* lecture */  
    int (*d_write)();          /* ecriture */  
    int (*d_ioctl)();          /* contrôle */  
    ...  
} cdevsw[];
```

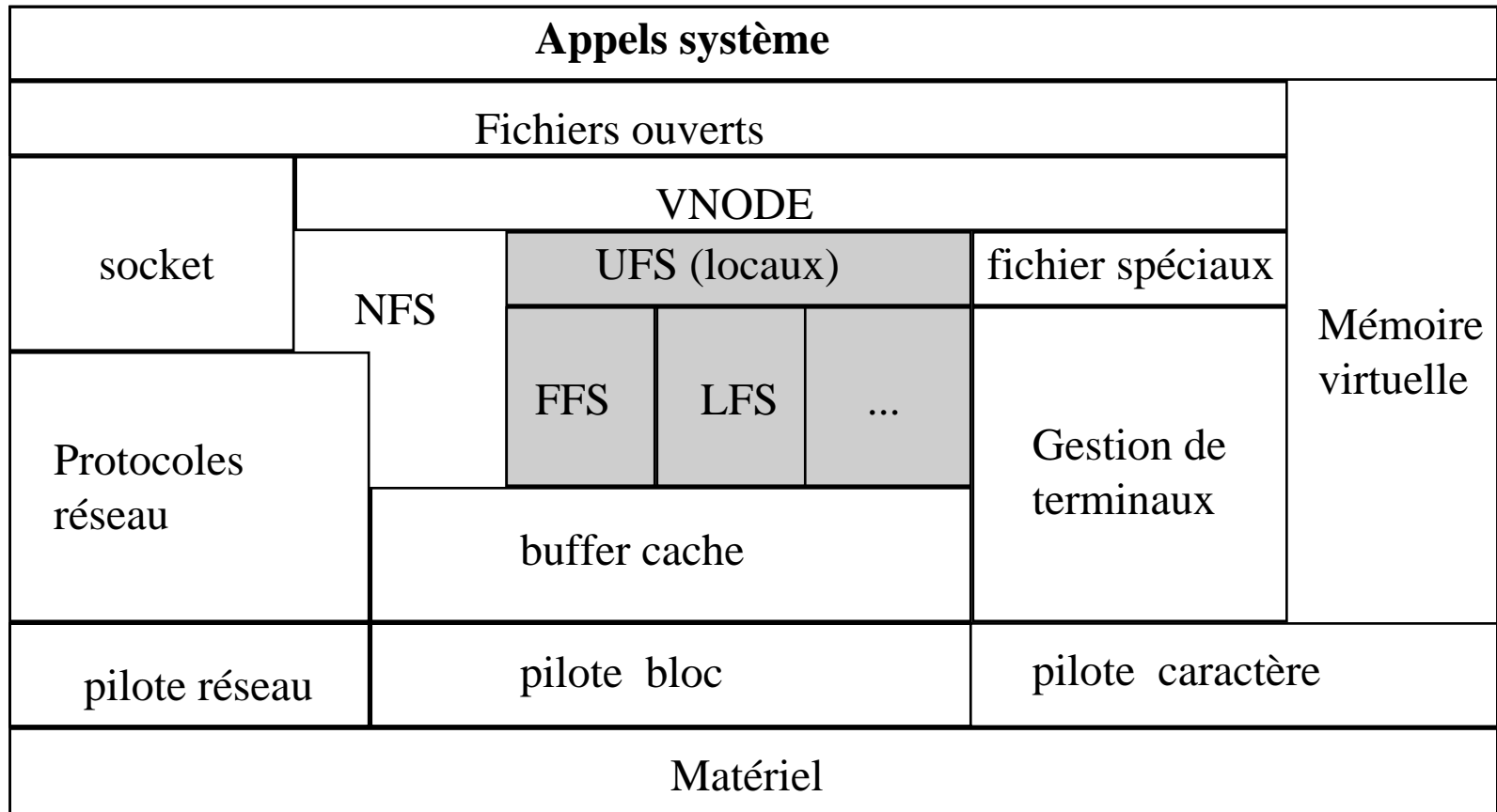
Tampon

- Terminaux : une c-list par terminal



- Les caractères sont copiés vers le contrôleur soit par le processeur soit par le contrôleur (DMA)
- Chaque type de périphérique gère ses propres tampons (possibilité de transférer directement depuis espace util.)

PARTIE 2 : Systèmes de Fichiers



Les différents systèmes de fichiers

- 2 principaux systèmes de fichiers locaux :

System V File System (s5fs)

- Système de fichier de base (78)

Fast File System (FFS)

- Introduit dans 4.2BSD

- Système de fichiers générique

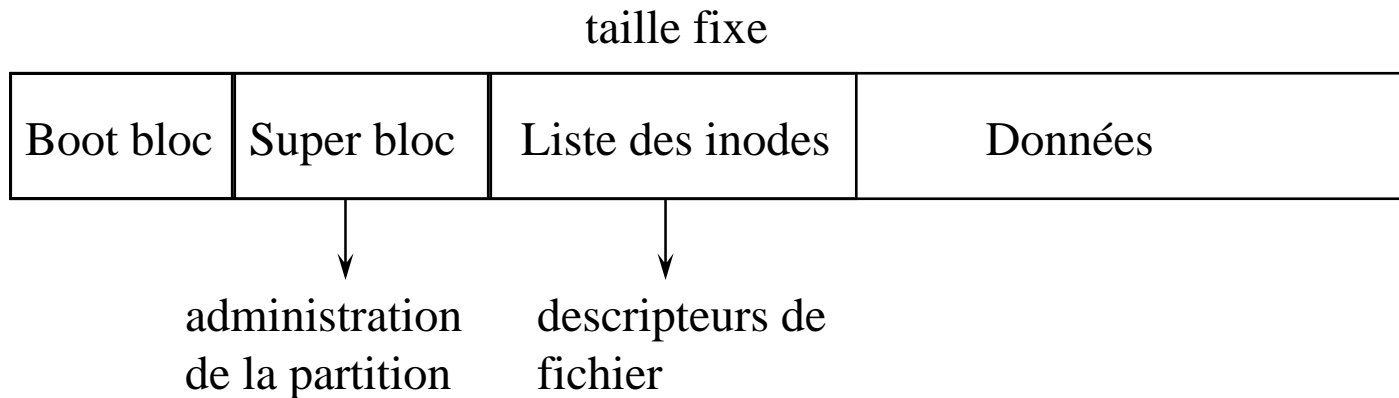
- Virtual File System (Sun 86)

- De nombreux autres systèmes de fichiers :

- Ext2fs (linux, FreeBSD) ...

Organisation générale du disque (s5fs)

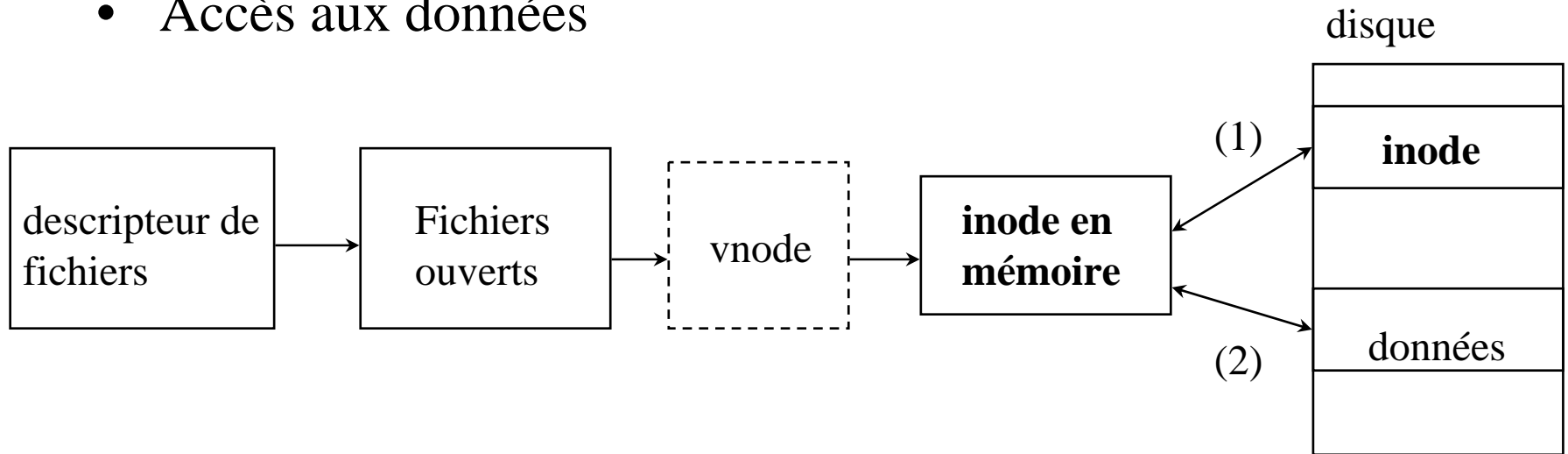
- Disque divisé en partitions
 - Chaque partition possède ses propres structures
- Organisation d'un partition :



- Numéro d'inode => accès aux blocs du fichier

Les structures

- Accès aux données



- Allocation de bloc

- Le superbloc contient la liste des blocs libres, des inodes libres

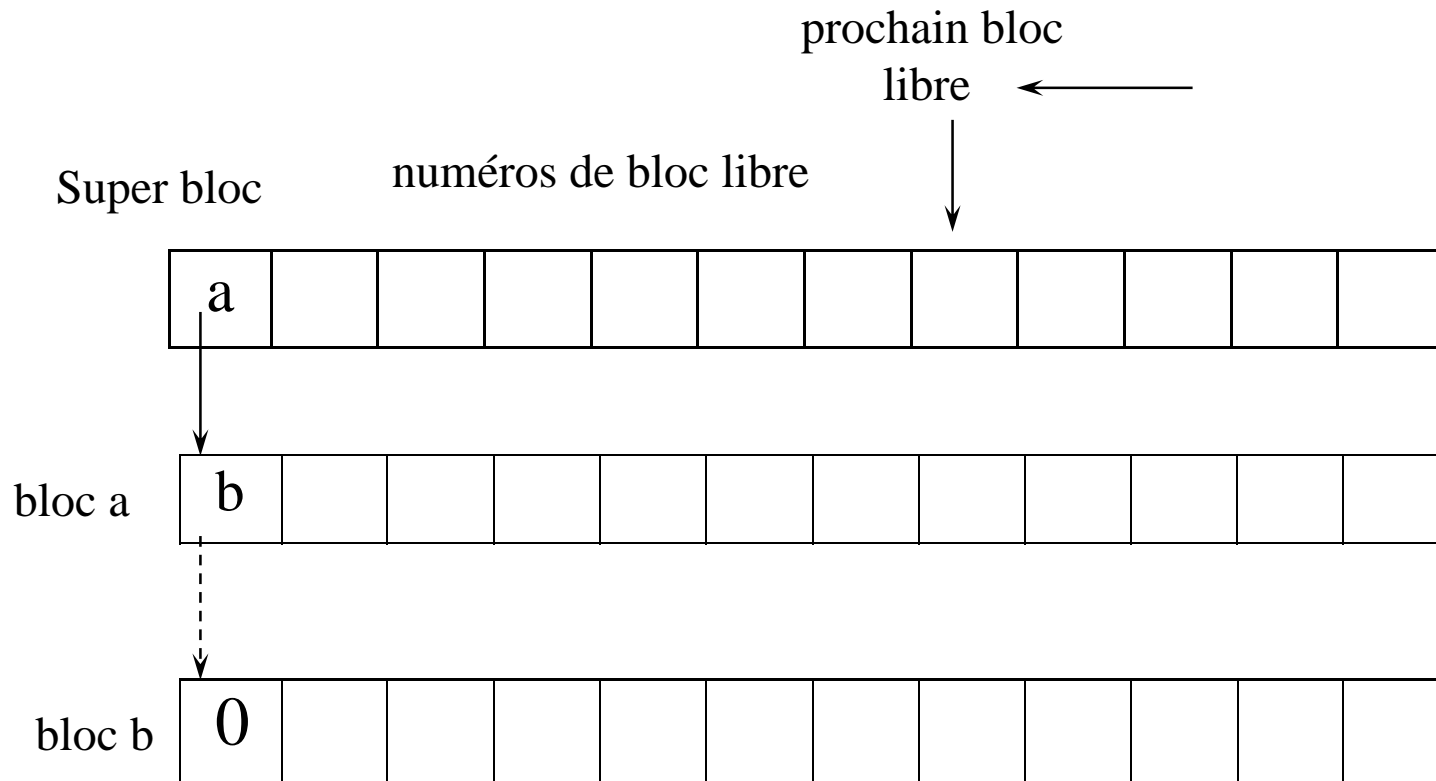
Gestion des blocs libres

- Le superbloc : struct fs (fs.h)

Bloc d'administration de la partition qui contient :

- Taille en blocs du système de fichier
- Taille en blocs de la table des inodes
- Nombre de blocs libres et d'inodes libres
- Liste des blocs libres
- Liste des inodes libres sur disque

Allocation des blocs libres



Allocation/libération de bloc : exemple

Configuration initiale

Superbloc



bloc 10



Libération du bloc 180

Superbloc



bloc 10



Allocation de 3 blocs (180, 234, 10)

Superbloc

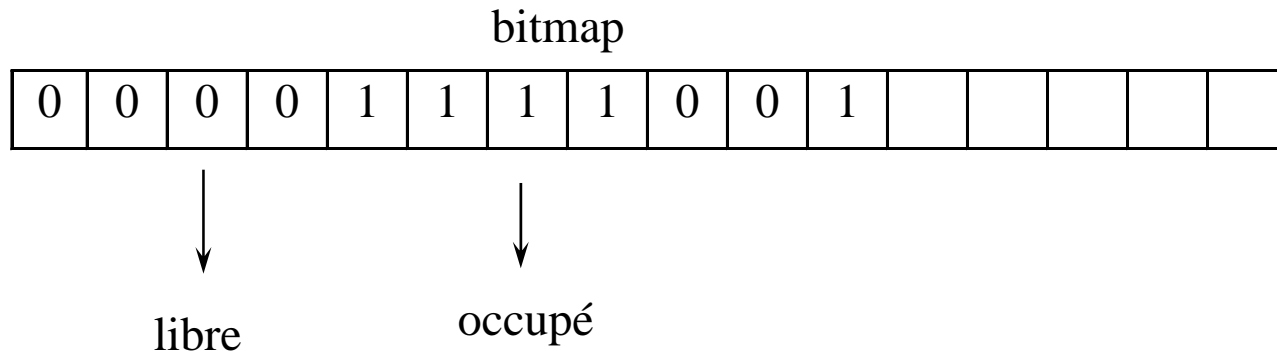


bloc 50



Allocation dans les nouveaux systèmes de fichiers

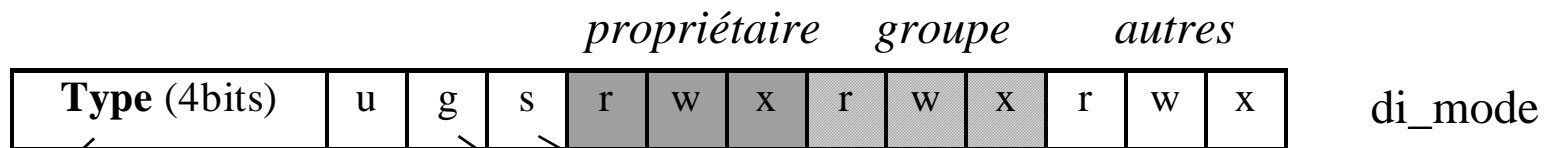
- Pb de la stratégie "classique" : pas de prise en compte de la contiguïté des blocs libres
- => vecteur binaire



- Exemple Ext2fs

Accès au données

- Structure des **inodes** = caractéristiques du fichier
- Sur disque : struct dinode
 - di_mode : type + droits
 - di_nlink nombre de liens physique
 - di_uid, di_gid
 - di_addr : table de blocs de données
 - di_atime, di_mtime, di_ctime : dates consultation, modification, modification inodes



IFREG fichiers normal
IFDIR répertoire
IFBLOCK périphérique bloc
IFCHR périphérique caractère

suid sgid sticky

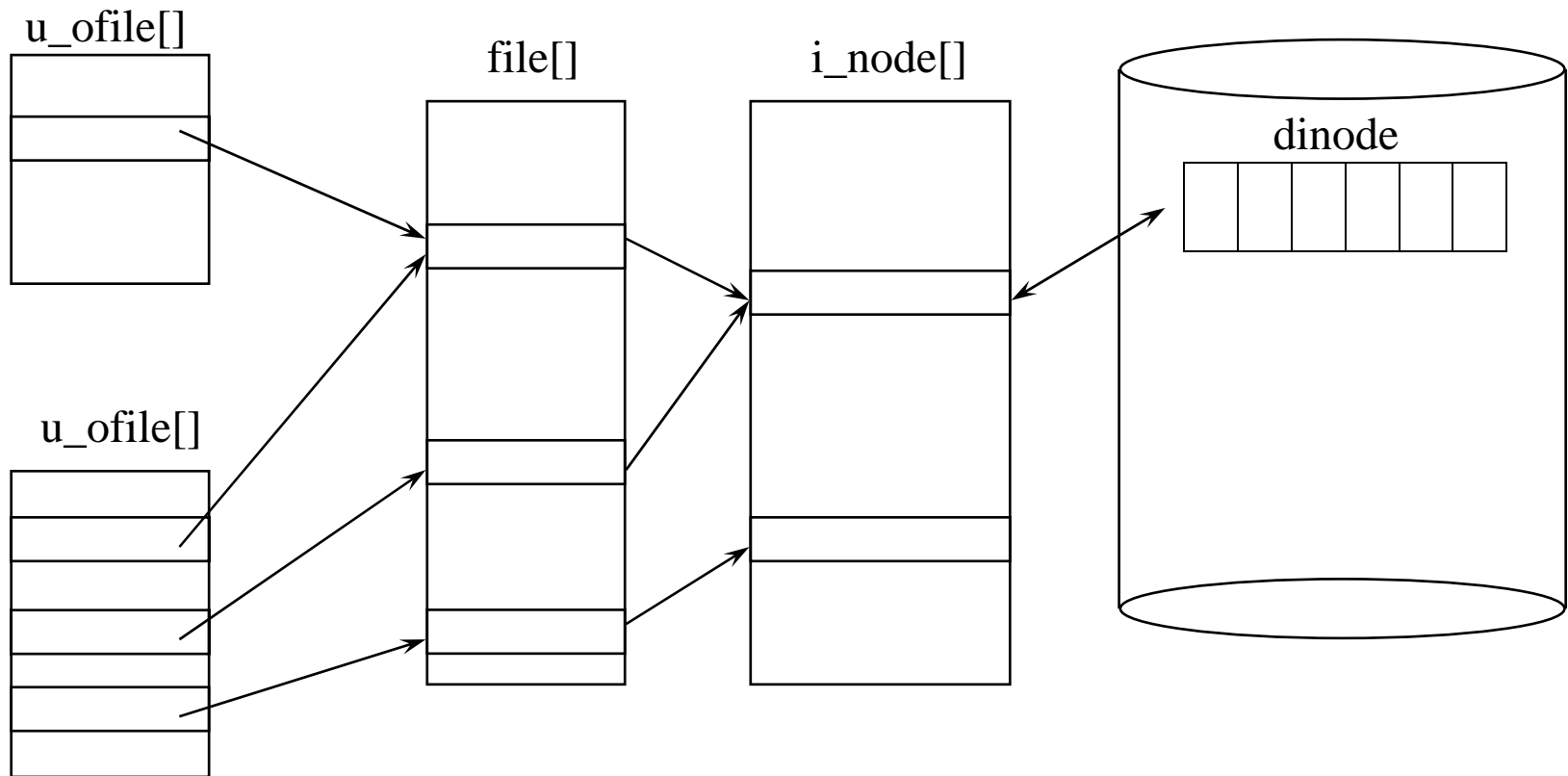
Structure inode

- En mémoire : struct **inode**
 - dinode avec en plus :
 - i_dev : device (partition)
 - i_number : numéro d'inode
 - i_flags : Flags (synchronisation, cache)
 - pointeurs sur la freelist (liste des inodes libres)
- } accès à l'inode sur disque (mises à jour)

Les autres structures

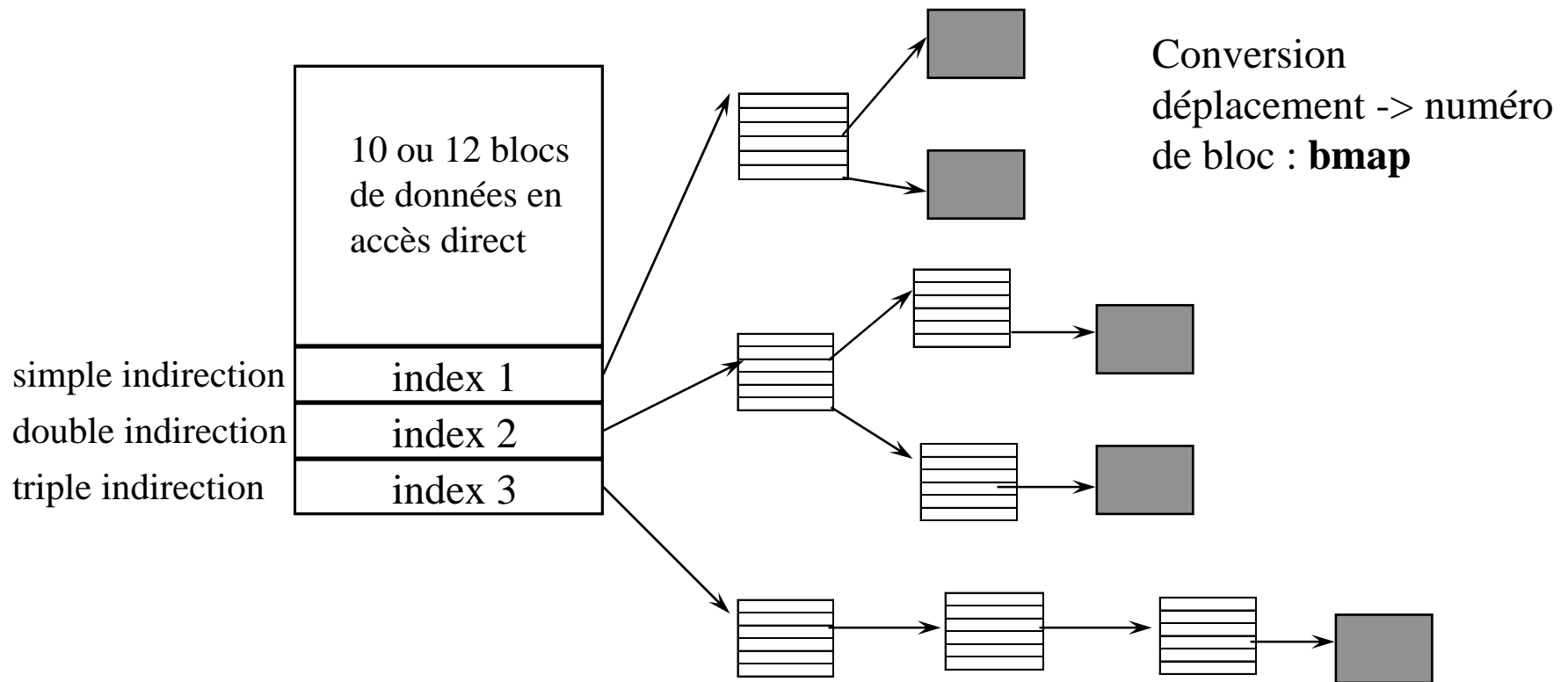
- `file[]` : table globale des fichiers ouverts
 - `f_flag` : mode d'ouverture (Lecture, Ecriture, Lecture/Ecriture)
 - `f_offset` : déplacement dans le fichier
 - `f_inode` : numéro d'inode
 - `f_count` : nombre de références
- `u_ofile[]` : Table locale des ouverts ouverts par un processus
 - entrée dans `file`

Résumé des structures



Où trouver les blocs ?

- Liste de blocs dans l'inode

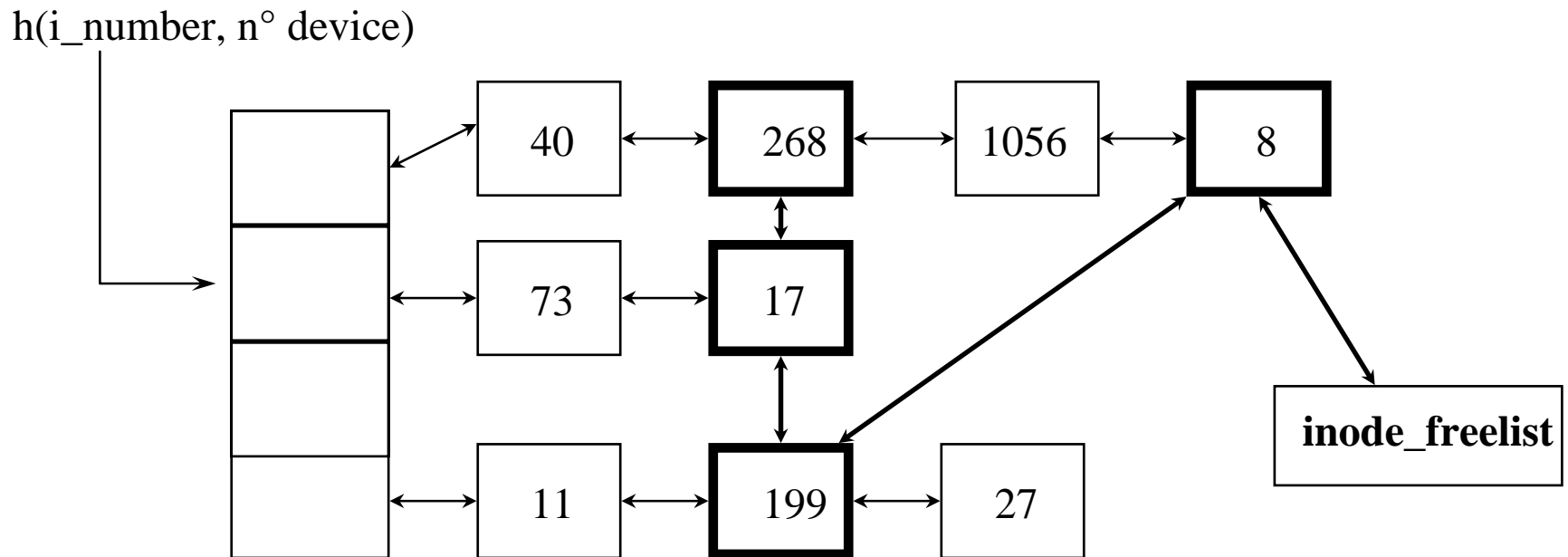


- Vision de l'utilisateur :



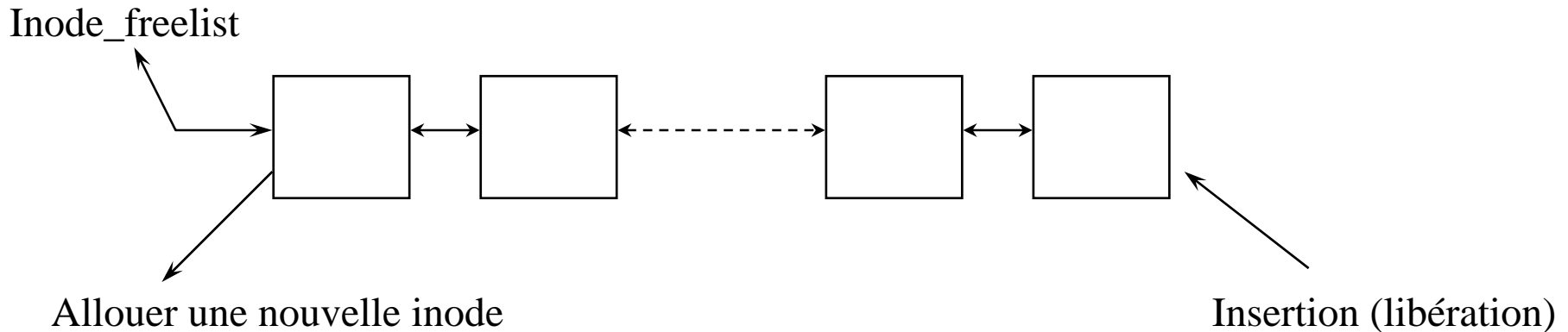
Les inodes en mémoire

- Les entrées de la table des inodes sont chaînées
- Pour trouver rapidement une inode à partir de son numéro utilisation d'une fonction de hachage



Gestion des inodes libres en mémoire

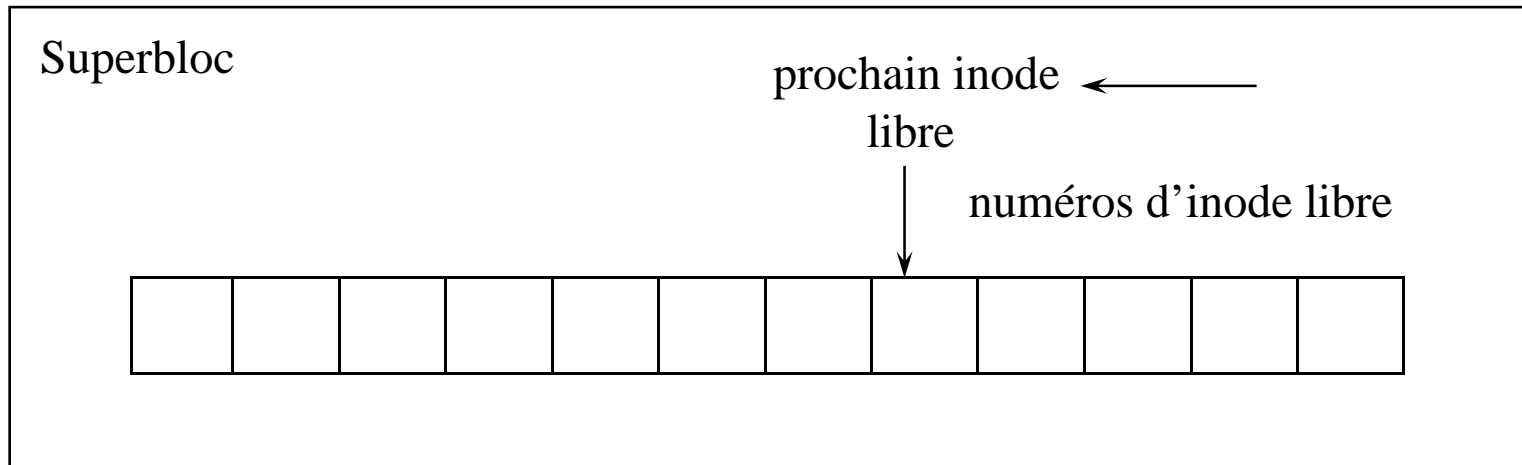
- Si une inode n'est plus utilisée par aucun processus => insertion dans inode_freelist.
- Inode_freelist = cache des anciennes inodes



- Gestion LRU (Least Recently Used) SVR3
(autre critère dans SVR4)

Gestion des inodes libres sur disque

- Le superbloc contient une liste partielle des inodes libres



- Si liste vide, réinitialiser la liste en «scannant» la table des inodes sur disque

Fonction de manipulation des inodes

- namei : retrouve une inode à partir d'un nom de fichier (open)
- ialloc : allouer une nouvelle inode disque à un fichier (creat)
- ifree : détruire une inode sur disque (unlink)
- iget : allouer/initialiser une nouvelle inode en mémoire
- iput : libérer l'accès à une inode en mémoire

Principe de ialloc

- Vérifier si aucun autre processus n'a verrouillé le superbloc (sinon sleep)
- Verrouiller le superbloc
- Si liste des inodes libres sur disque non vide
 - Prendre l'inode libre suivante dans superbloc
 - attribuer une inode en mémoire (iget)
 - mise à jour sur disque (inode marquée prise)
- Si liste vide
 - Verrouiller le superbloc
 - parcourir la liste des inodes sur disque pour remplir le superbloc
- Tester à nouveau si l'inode est vraiment libre sinon la libérer et recommencer (conflit d'accès à un même inode !)

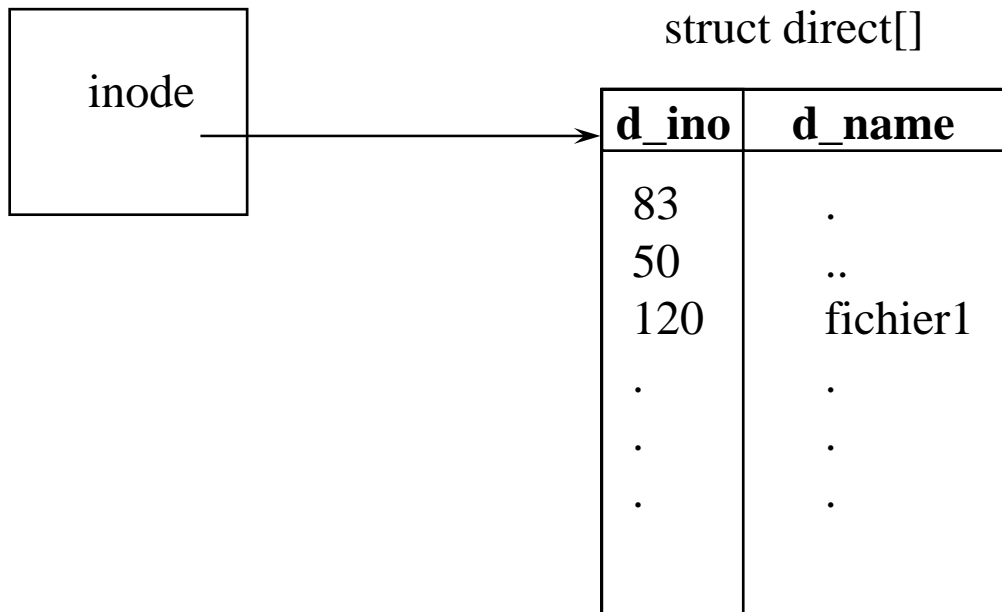
Conflit d'accès à une inode

Principe de iget

- Trouver l'inode en mémoire à partir du couple `<i_number, device>`
- Si inode non présente allouer un inode libre en mémoire (à partir de la `inode_freelist`)
- Remplir l'inode à partir de l'inode sur disque

Les répertoires

- Répertoire = un **fichier** de type répertoire
=> référencé par une inode



d_name (14 caractères) SVR4
(255 caractères) BSD

Algorithme de namei

Entrées: nom du chemin

Sortie: inode

Si (premier caractère du chemin == '/')

 dp = inode de la racine (rootdir) (iget)

sinon

 dp = inode du répertoire courant (u.u_cdir) (iget)

Tant qu'il reste des constituants dans le chemin {

 lire le nom suivant dans le chemin

 vérifier les droits et que dp désigne un répertoire

 si dp désigne la racine et nom = ".."

 continuer

 lire le contenu du répertoire (bmap pour trouver le bloc puis bread)

 si nom suivant appartient au répertoire

 dp = inode correspond au nom

sinon

 // Pas d'inode

}

retourner dp

Exemple

Les liens

- Fichiers spéciaux
 - Liens symboliques : contient le nom d'un fichier
 - Liens physiques : désigne la même inode

```
ln -s /users/paul/f1 /users/pierre/lfs1
```

```
ln /users/paul/f2 /users/pierre/lhf2
```

```
rm /users/pierre/lfs1
```

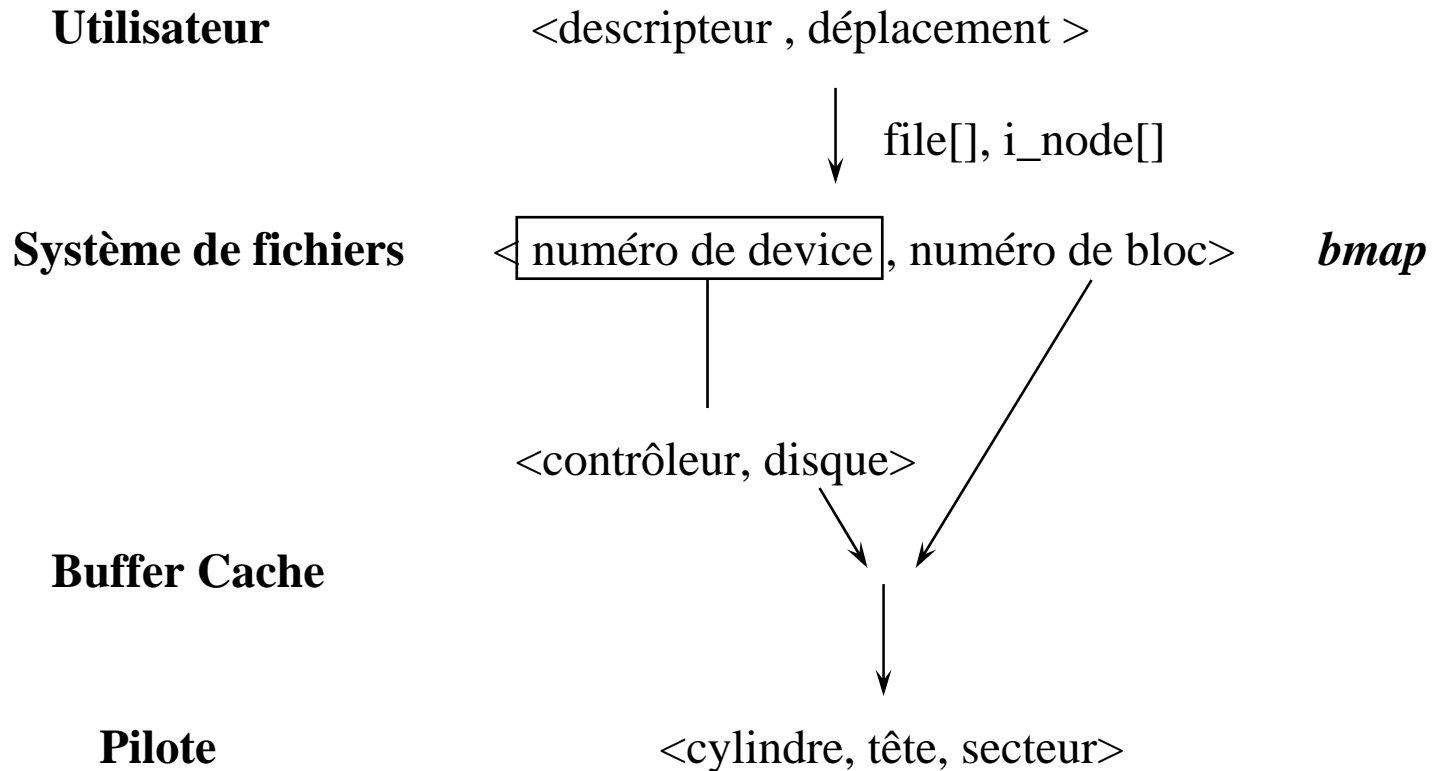
```
rm /users/pierre/lhf2
```

Droits : 1) droits sur le lien
2) droits sur le fichiers

Droits : droits sur le fichier

Implémentation des appels système

- Systèmes d'adressage :



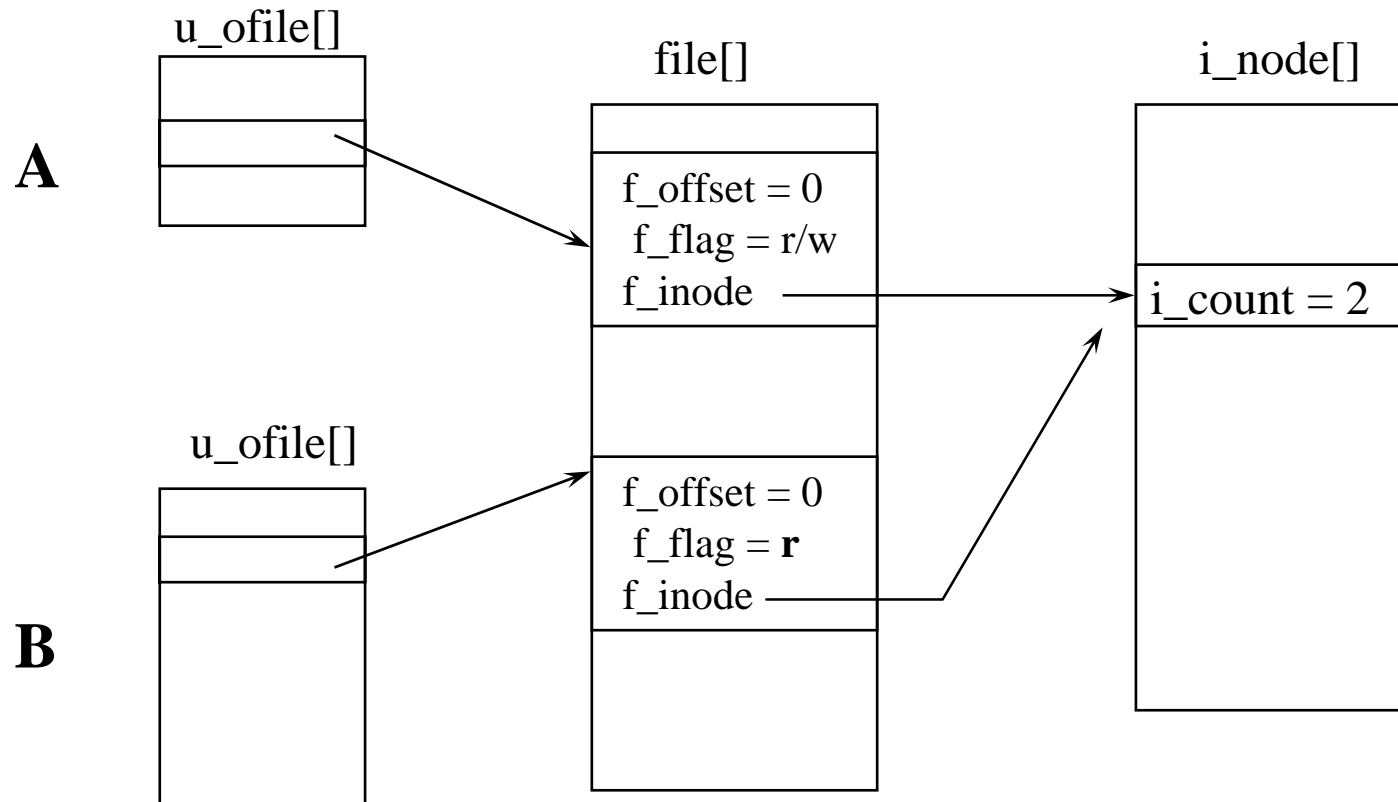
Algorithme de open

- Retrouver l'inode à partir du nom de fichier (namei)
- Si (fichier inexistant ou accès non permis) retourner erreur
- allouer et initialiser un élément dans la table file[]
- allouer et initialiser une entrée dans u_ofile du processus
- Si (mode indique une troncature) libérer les blocs (free)
- déverrouiller inode
- retourner le descripteur

Exemple

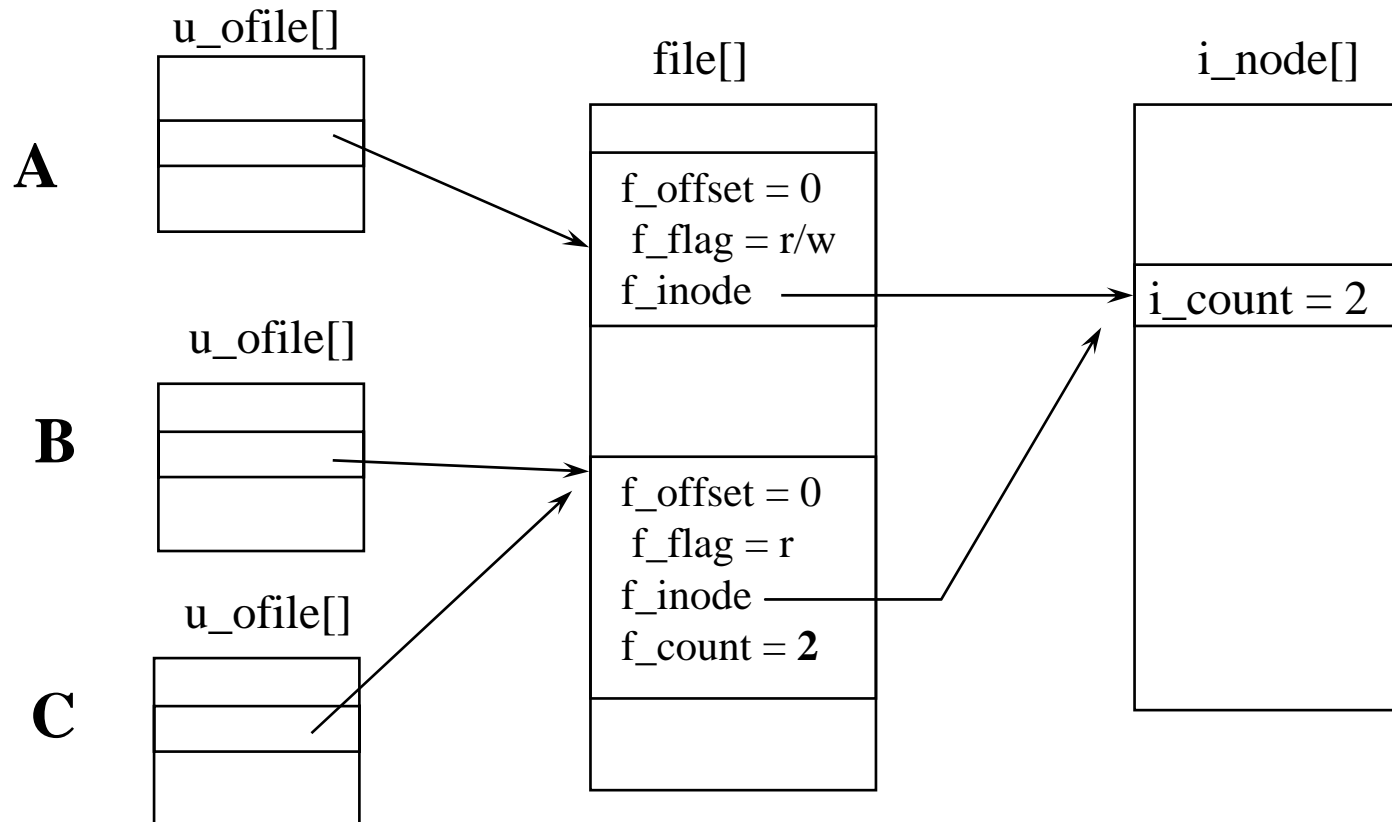
Processus A : `fd = open ("/home/sens/monfichier", O_RDWR|O_CREAT, 0666);`

Processus B : `fd = open("/home/sens/monfichier", O_RDONLY);`



Exemple (2)

Processus B : fork()



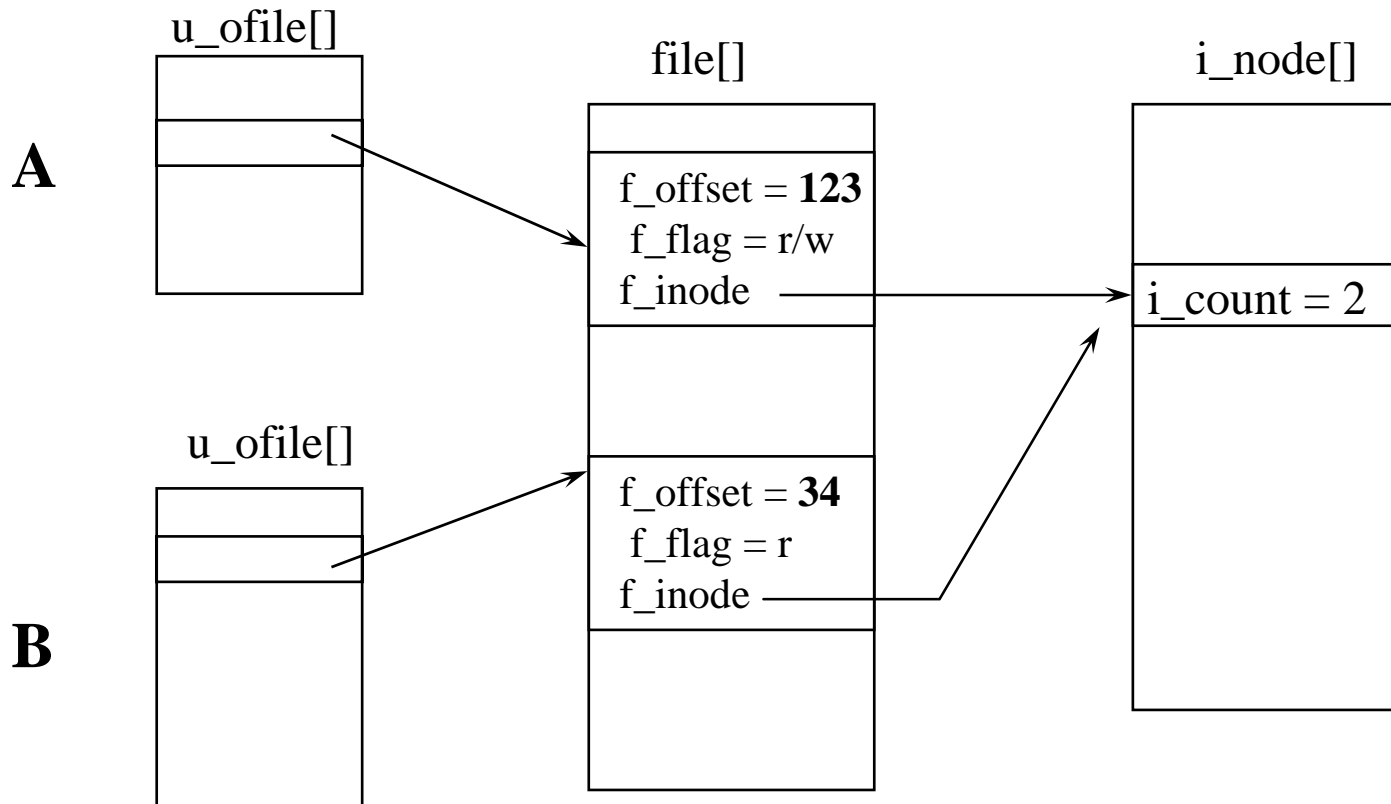
Algorithme de read

- Accéder à l'entrée de file[] à partir de u_ofile[fd]
- Vérifier le mode d'ouverture (champs f_flag)
- Copier dans la zone u les informations pour le transfert
- Verrouiller l'inode (f_inode)
- Tant que (nombre octets lus < nombre à lire)
 - Conversion déplacement numéro bloc (bmap)
 - Calculer le déplacement dans le bloc
 - Si (nb octets restants == 0) break; // Fin de fichier
 - Lecture du bloc dans le cache (bread)
 - Transférer tampon dans zone u
 - libérer le tampon (verrouiller par bread)
- Déverrouiller l'inode; Mettre à jour file[]
- Retourner nombre octets lus

Exemple

Processus A : nb = write(fd, buf, 123);

Processus B : nb = read(fd, buf, 34);



Les optimisations : fast file system (ffs)

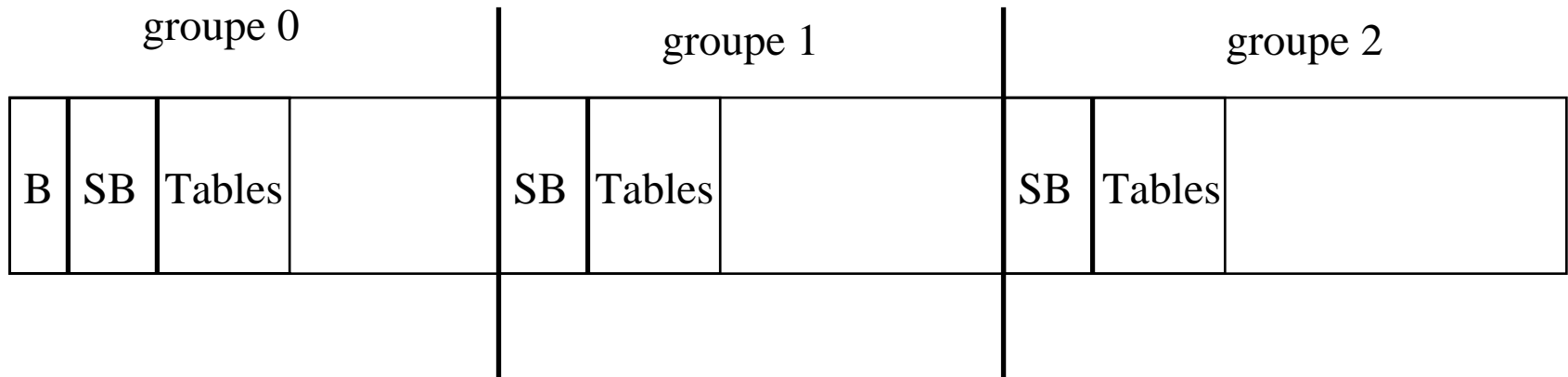
- Intégrer dans tous les unix (connu comme ufs)
- De nombreuses améliorations

=> Augmenter la fiabilité

=> Augmenter les performances

Organisation en groupe

- Disque divisé en groupe de cylindre



- Réplication du superbloc => augmenter la fiabilité
- Dissémination des tables => réduction des temps d'accès

Blocs et fragments

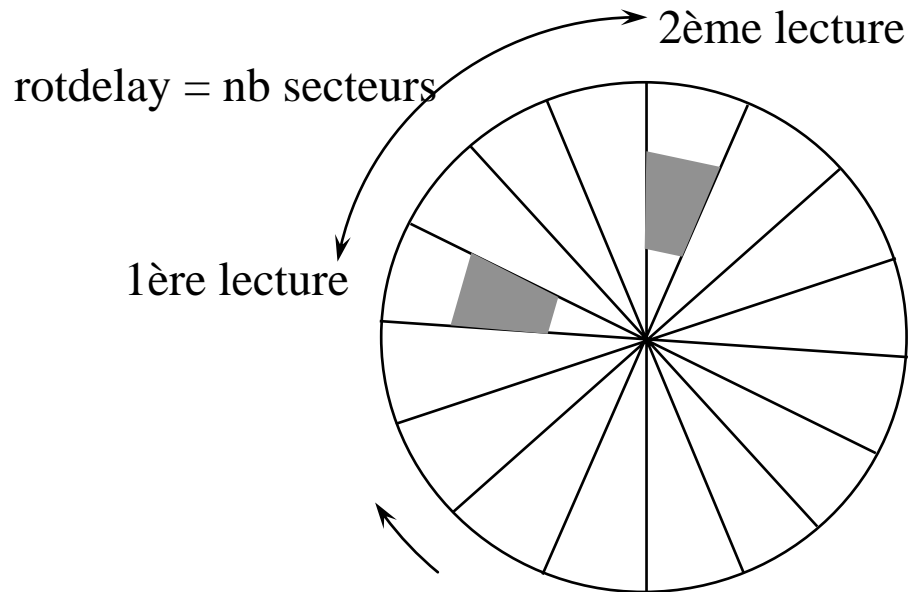
- Problème sur la taille des blocs
 - Taille de blocs importante => plus de données transférées en une E/S
plus d'espace perdu (1/2 bloc en moyenne)
- Idée : partager les blocs entre plusieurs fichiers
- => Blocs divisés en fragment
 - 2,4,8 fragments par bloc
 - Taille "classique" : blocs 8Ko, fragment 512 octets
- Unité d'allocation = segment
 - => perte réduite
 - => plus de structures de données

Optimisations

- Optimisations :
 - 1) Regrouper toutes les inodes d'un même répertoire dans un même groupe
 - 2) Inode d'un nouveau répertoire sur un autre groupe
=> distribution des inodes
 - 3) Essayer de placer les blocs de données d'un fichier dans un même groupe que l'inode
- => limiter les déplacements de tête

Politique d'allocation de bloc

- Constatations : la plupart des lectures sont séquentielles
- => placement des blocs d'un même fichier
 - En fonction de la vitesse de rotation pour optimiser lecture séquentielle
 - Objectif : faire en sorte que lors de la lecture suivant le bloc soit sous la tête

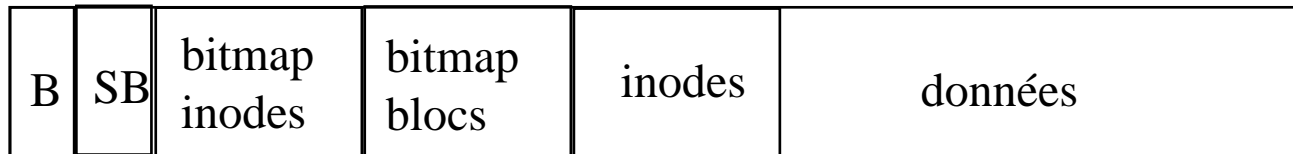


Performances

- Stratégie d'allocation efficace si disque pas trop plein (< 90%)
- Sur VAX/750
- Accès lecture débit = 29 Ko/s s5fs
débit = 221 Ko/s ffs
- Accès écriture débit = 48Ko/s s5fs
débit = 142 Ko/s ffs

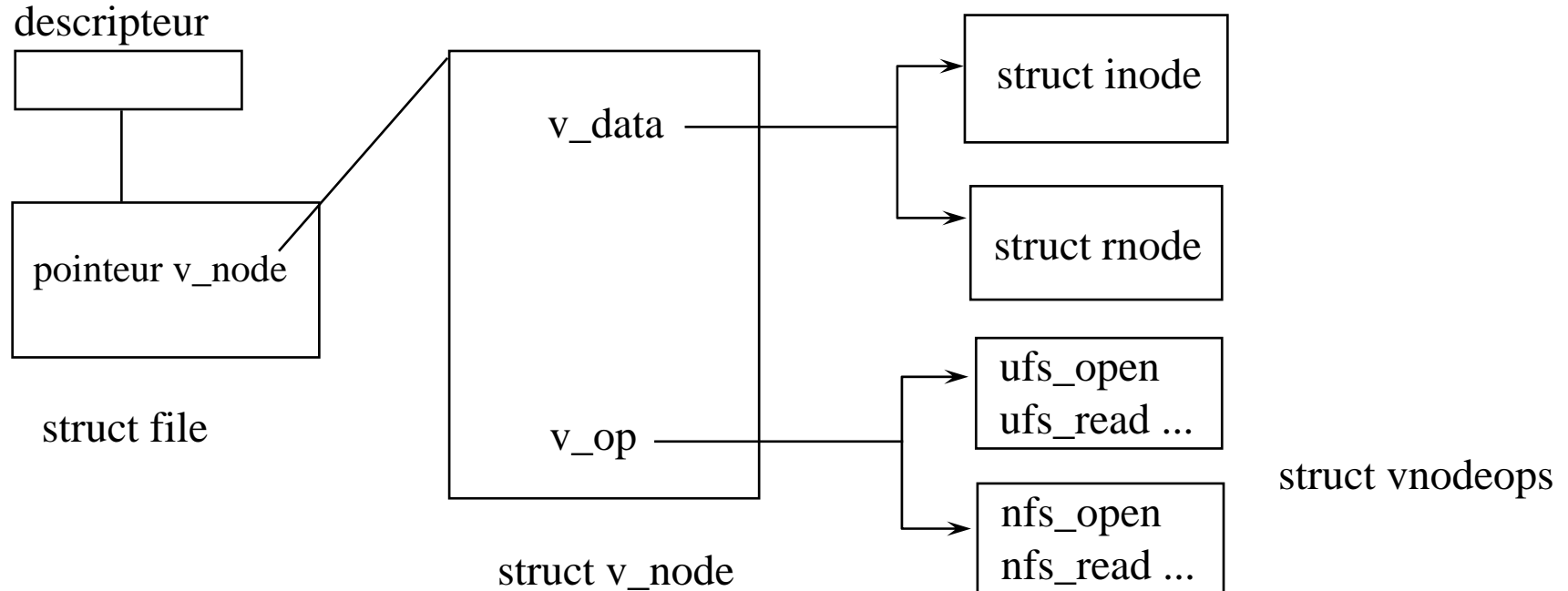
D'autres organisations

Exemple Ext2fs



Systemes génériques : VFS

- Objectifs : gérer différents systèmes de fichiers locaux et distants => Virtual File System
- Ajout d'une couche supplémentaire responsable de l'aiguillage : couche vnode (virtual node)



Architecture VFS

