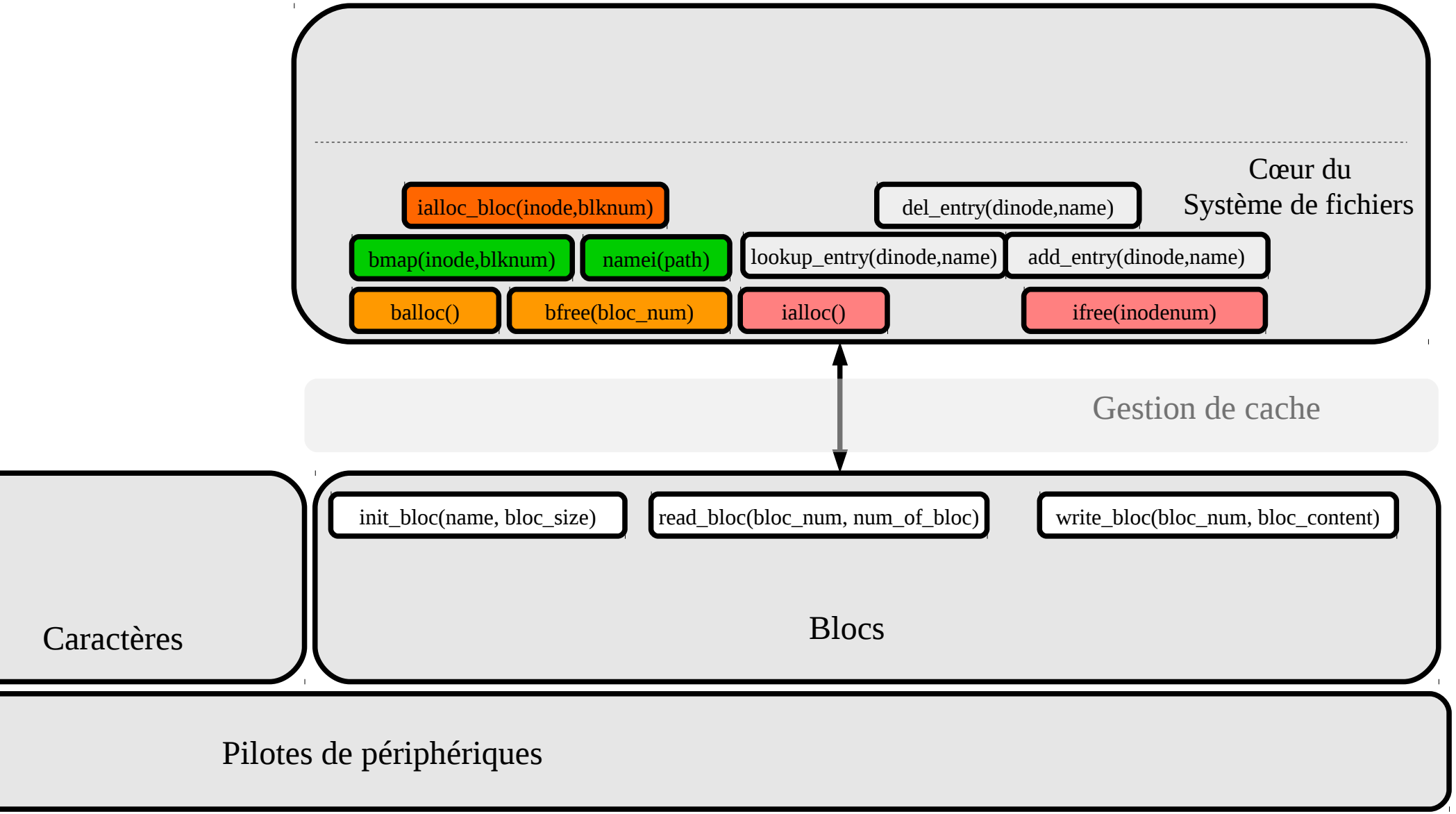
2 – Systèmes de fichiers
Partie 2 : Fonctions de gestion internes

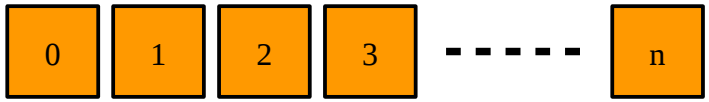
HEPIA
Année académique 2015/2016

Vue globale


Cœur du
Système de fichiers
ialloc_bloc(inode,blknum)
del_entry(dinode,name)
bmap(inode,blknum)
namei(path)
lookup_entry(dinode,name)
add_entry(dinode,name)
balloc()
bfree(bloc_num)
ialloc()
ifree(inodenum)
Gestion de cache
init_bloc(name, bloc_size)
read_bloc(bloc_num, num_of_bloc)
write_bloc(bloc_num, bloc_content)
Caractères
Blocs
Pilotes de périphériques

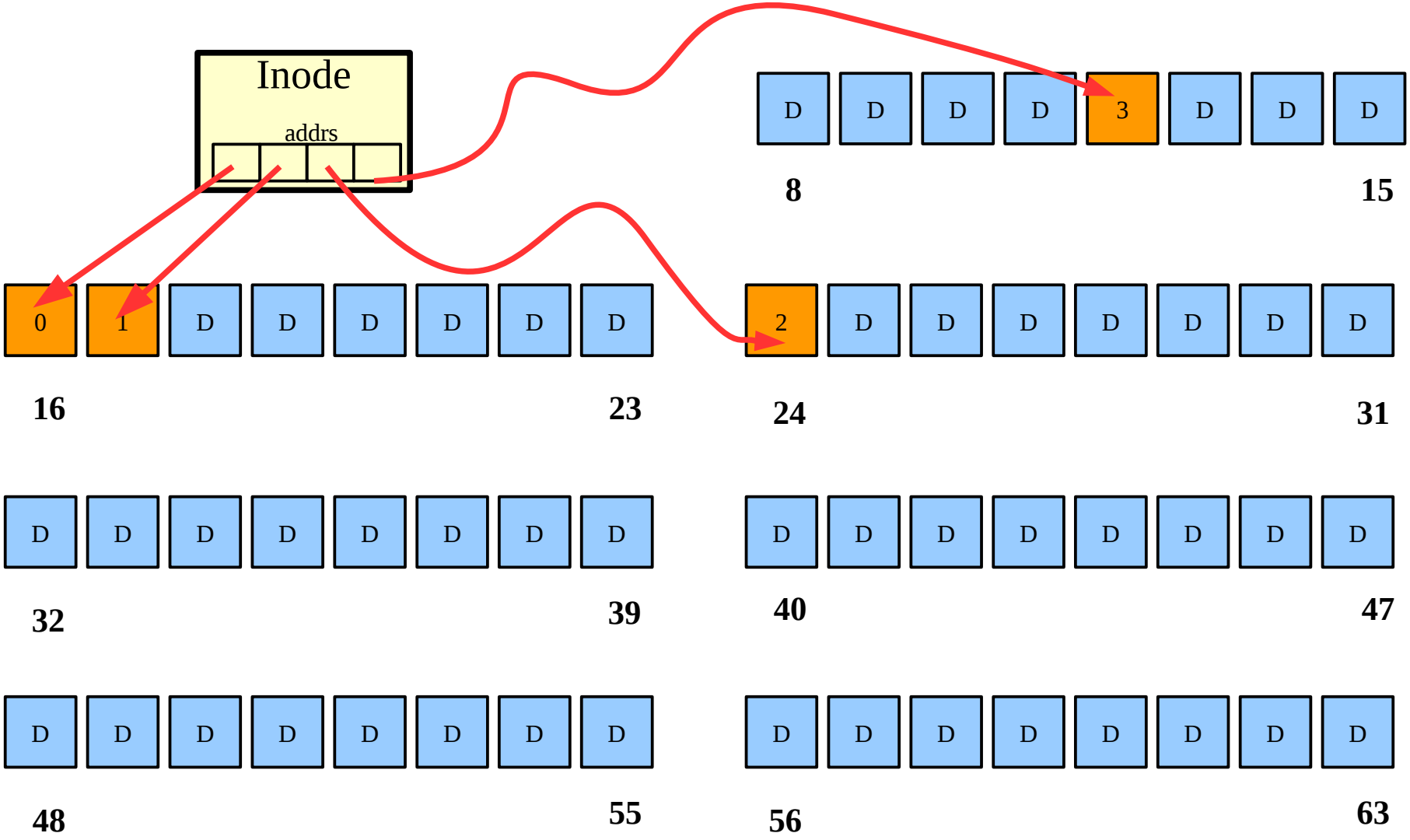
La fonction bmap

On veut offrir un service d'accès **ordonné** aux blocs qui composent un fichier/inode : début du fichier en bloc 0, suite en bloc 1,etc



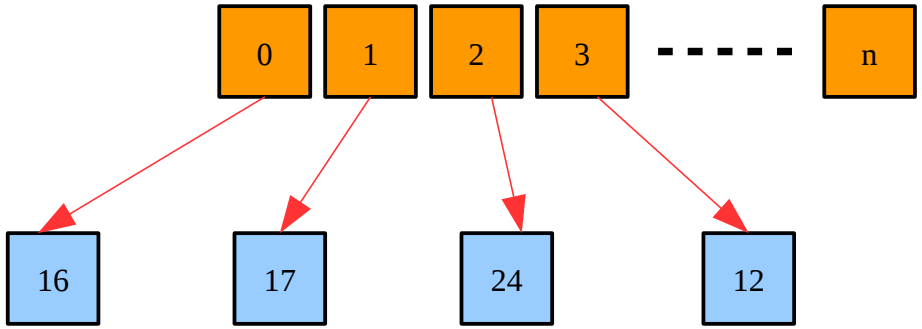
La fonction bmap

Sauf que pour une bonne raison (cf cours 1), les blocs de données d'un fichier ne sont pas toujours contigus sur le disque.



La fonction bmap

Le rôle principal de la fonction bmap est de mapper un numéro de bloc du fichier vers un numéro de bloc du disque :



bmap(inode, 0) =16

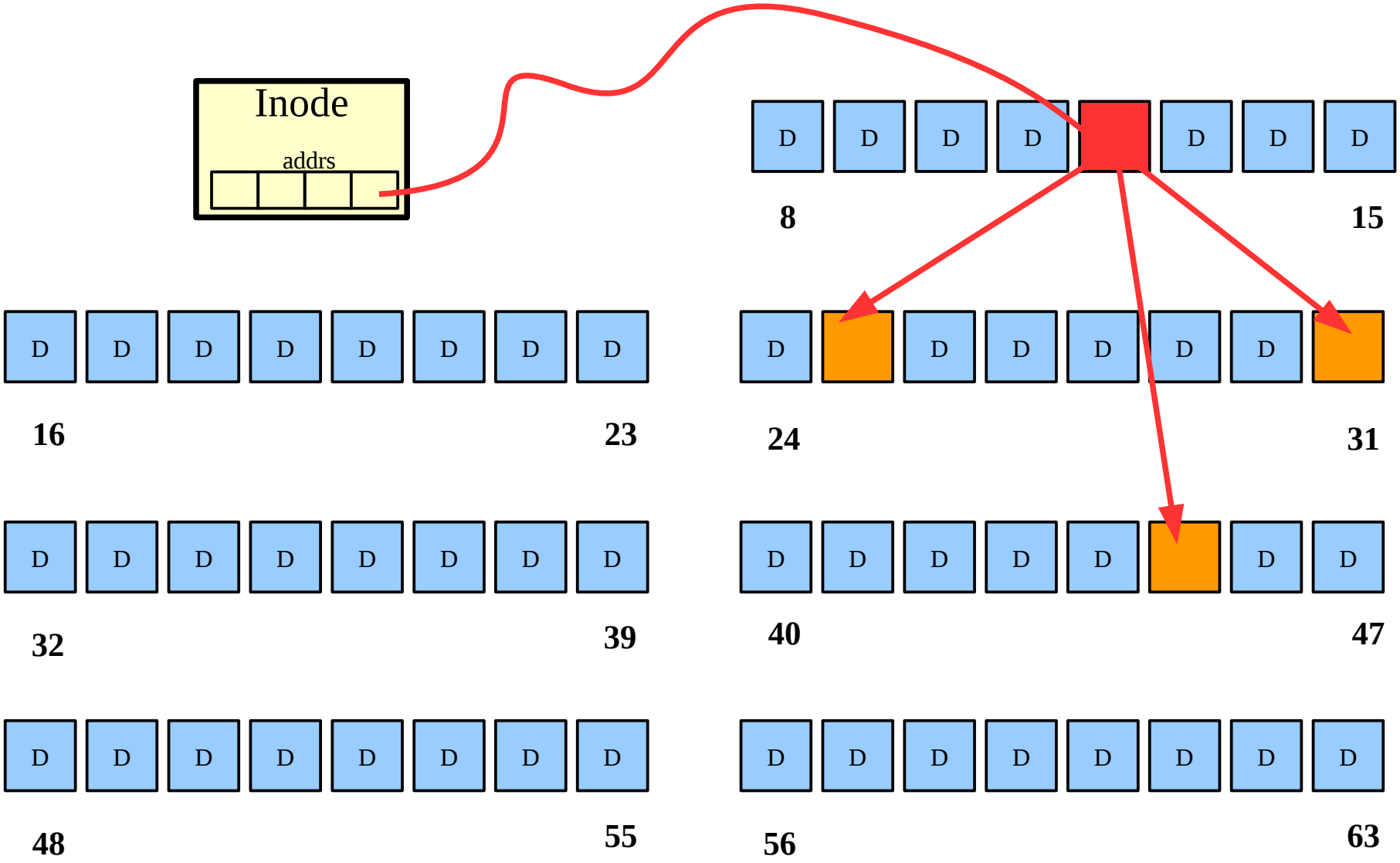
bmap(inode, 1) =17

bmap(inode, 2) =24

bmap(inode, 3) =12

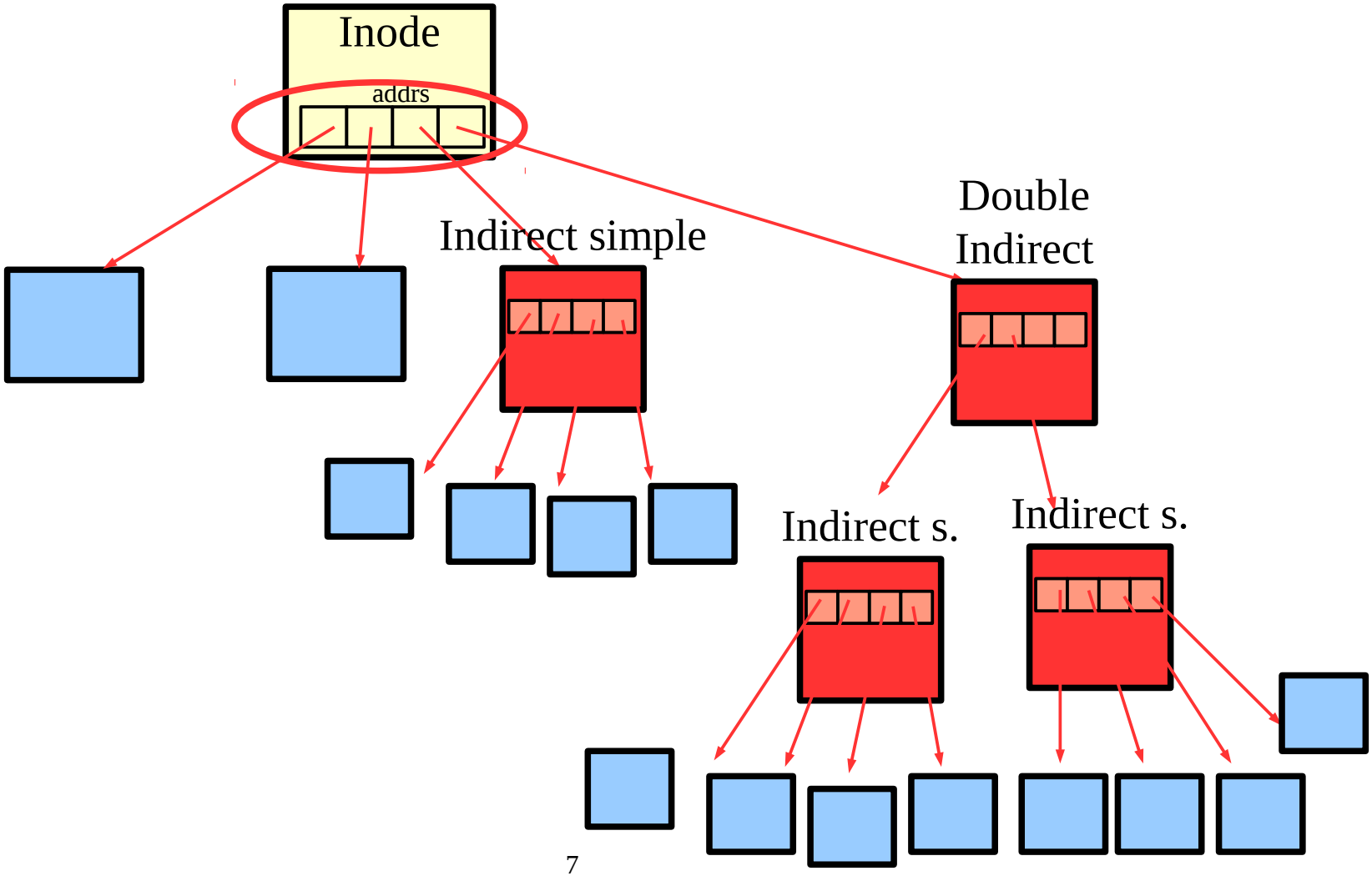
La fonction bmap

En plus, certains blocs peuvent être des blocs indirects, c'est à dire contenir les adresses d'autres blocs



La fonction bmap

En suivant le même principe, on peut ajouter des blocs doublement/triplement/... indirects. La distinction entre bloc simple, bloc indirect et bloc double indirect se fait sur sa position dans le tableau des adresses de l'inode



La fonction bmap

Ainsi la structure des inodes de Minix :

```
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 */
};
```

peut aussi s'écrire :

```
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[9];          /* Adresse logique des blocs direct/indirect ([7]/double indirect ([8]) du fichier */
};
```

La fonction bmap : algorithme

Pseudo-code avec 2 niveau d'indirections:

```
int bmap(inode, blknum) :
```

Cas 0)

```
Si blknum < nombre de blocs adressables directement
     renvoyer inode.addrs[blknum]

blknum = blknum – nombre de bloc adressables directement
```

Cas 1)

```
Si blknum < nombre de blocs adressables avec une seule indirection
     indirect_bloc = read_bloc(inode.addrs[indice du bloc des indirections simples])
     renvoyer indirect_bloc[blknum]

blknum = blknum – nombre de blocs adressables avec une seule indirection
```

Cas 2)

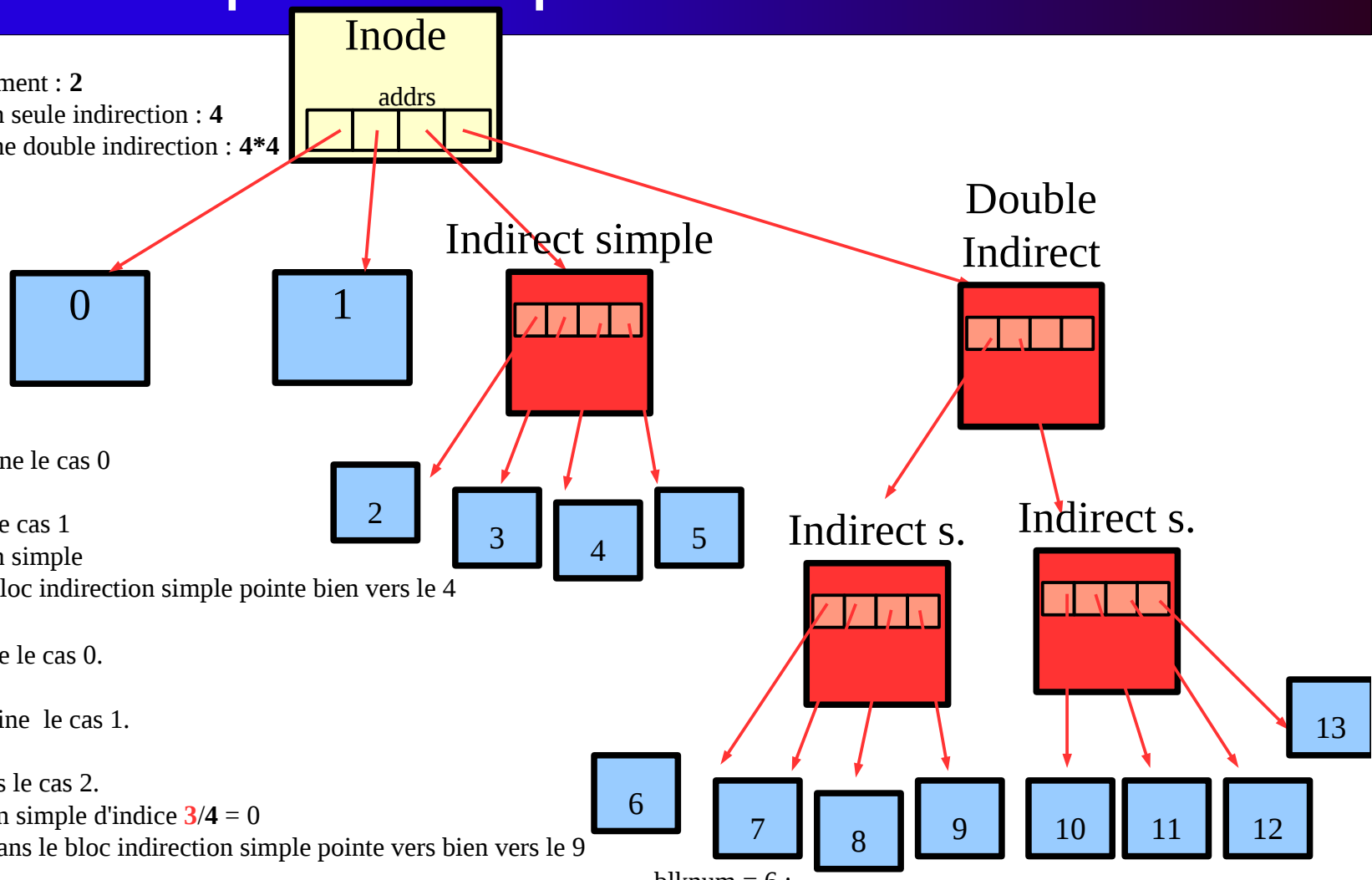
```
Si blknum < nombre de bloc adressables avec une double indirection
     indirect_bloc = read_bloc(inode.addrs[indice du bloc des indirections doubles])
     indirect2_bloc= read_bloc(indirect_bloc[blknum/nombre de blocs adressables avec une seule indirection])
     Renvoyer indirect2_bloc[blknum modulo nombre de blocs adressables avec une seule indirection]
```

Pourquoi la soustraction entre chaque cas ? Permet de réajuster l'indice dans le bloc lors d'une indirection : on passe d'un indice absolu à un indice relatif au bloc indirect.

Par exemple, si 7 blocs sont adressables directement, mais qu'on veut le 8ème, on veut le premier indice du bloc indirect, pas l'indice 8.

La fonction bmap : exemples

Nombre de blocs adressables directement : **2**
Nombre de blocs adressables avec un seule indirection : **4**
Nombre de blocs adressables avec une double indirection : **4*4**



blknum = 4 :
- pas à inférieur à **2**, on élimine le cas 0
- 4-**2** = **2**
- **2** inférieur à **4**, on va dans le cas 1
- lecture du bloc d'indirection simple
- résultat = indice 2 dans le bloc indirection simple pointe bien vers le 4

blknum = 9 :
- pas inférieur à **2**, on élimine le cas 0.
= 9-**2** = **7**
- **7** pas inférieur à **4**, on élimine le cas 1.
- **7**-**4** = **3**
- **3** inférieur à **16**, on est dans le cas 2.
- lecture du bloc d'indirection simple d'indice **3**/**4** = 0
-indice 3 modulo **4** dans le bloc indirection simple pointe vers bien vers le 9

blknum = 10 :
- pas inférieur à **2**, on élimine le cas 0.
= 10-**2** = **8**
- **8** pas inférieur à **4**, on élimine le cas 1.
- **8**-**4** = **4**
- **4** inférieur à **16**, on est dans le cas 2.
- lecture du bloc d'indirection simple d'indice **4**/**4** = 1
-indice **4** modulo **4** dans le bloc indirection simple pointe vers bien vers le 10

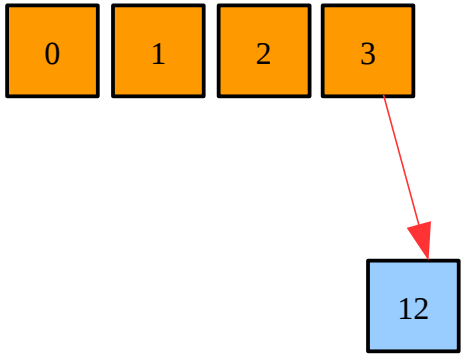
blknum = 6 :
- pas inférieur à **2**, on élimine le cas 0.
= 6-**2** = **4**
- **4** pas inférieur à **4**, on élimine le cas 1.
- **4**-**4** = **0**
- **0** inférieur à **16**, on est dans le cas 2.
- lecture du bloc d'indirection simple d'indice **0**/**4** = 0
-indice **0** modulo **4** dans le bloc indirection simple pointe vers bien vers le 6

La fonction ialloc_bloc

Rajoute un bloc vide à la carte des blocs géré par l'inode, alloué dans la liste des blocs disponibles du volume et renvoit le numéro de bloc du disque alloué correspondant.

La fonction ialloc_bloc

Rajoute un bloc vide à la carte des blocs géré par l'inode, alloué dans la liste des blocs disponibles du volume, renvoit le numéro de bloc du disque alloué correspondant.



Si le bloc existe déjà dans l'inode, le comportement de la fonction est équivalent à bmap

sinon on mappe sur un bloc ajouté dont le numéro est passé en paramètre

La fonction ialloc_bloc

Pseudo-code avec 1 niveau d'indirection:

```
int ialloc_bloc(inode, blknum) :
```

Cas 0)

```
Si blknum < nombre de blocs adressables directement
     Si inode.addrs[blknum] == 0
          inode.addrs[blknum] = balloc()
     Renvoyer inode.addrs[blknum]

blknum = blknum – nombre de bloc adressables directement
```

Cas 1)

```
Si blknum < nombre de blocs adressables avec une seule indirection
     Si inode.addrs[indice du bloc des indirection simples] == 0
          inode.addrs[indice du bloc des indirection simples] = balloc()

     indirect_bloc = read_bloc(inode.addrs[indice du bloc des indirection simples])
     si indirect_bloc[blknum] == 0
          indirect_bloc[blknum] = balloc()
          write_bloc(inode.addrs[indice du bloc des indirection simples], indirect_bloc)
     renvoyer indirect_bloc[blknum]
```

La fonction namei

Les inodes n'ont pas de noms, juste un numéro. Le rôle de namei est de retrouver le numéro d'un inode à partir de son chemin d'accès dans la chaîne des répertoires/annuaires du système de fichier.

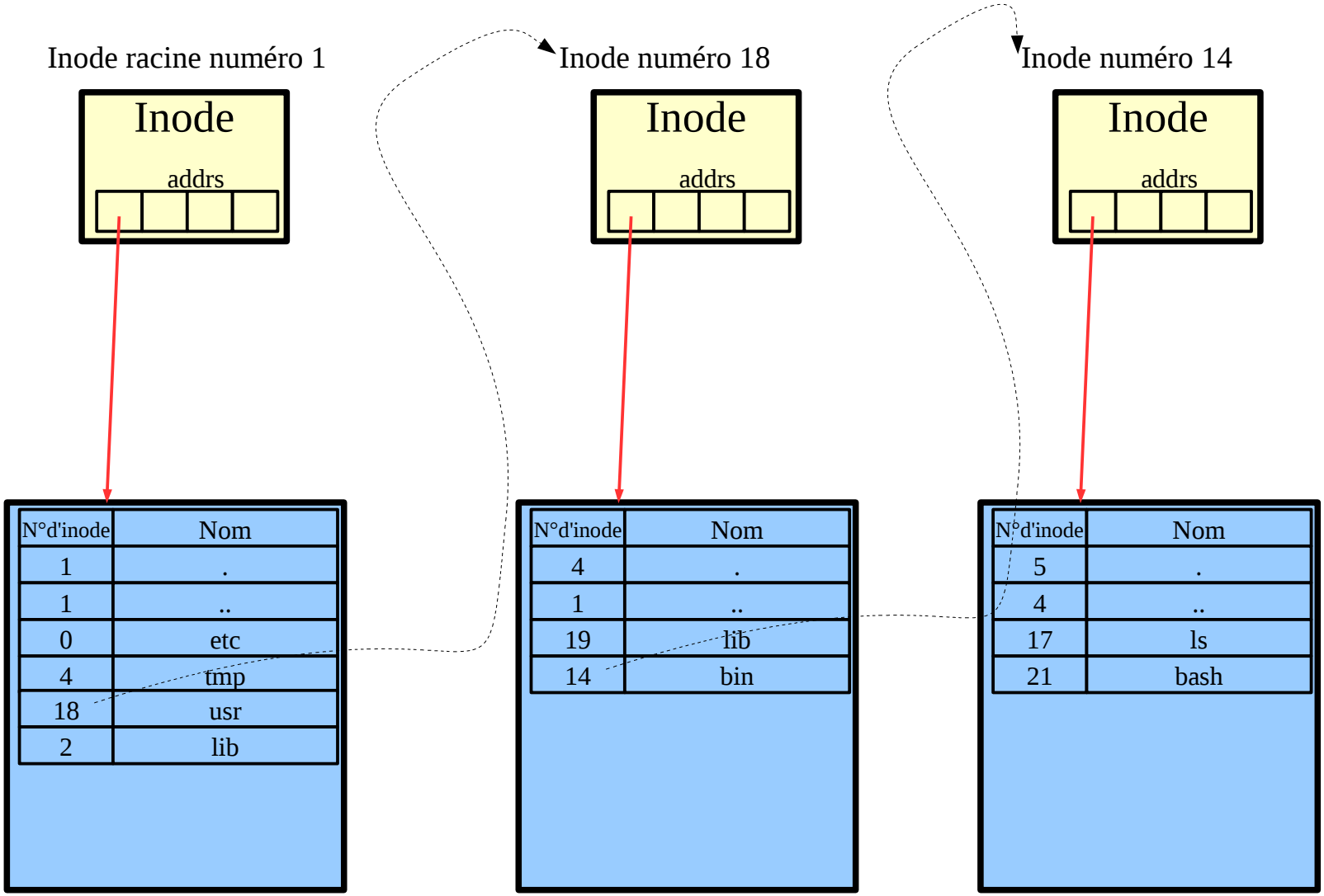
Ex : namei("/usr/bin/bash") =16

Cette fonction sera utilisée par tous les appels systèmes qui demandent un chemin d'accès en paramètre.

La recherche débute depuis l'inode de la racine si le chemin d'accès est absolu, le path working directory du processus si le chemin d'accès est relatif.

La fonction namei : exemple

namei sur le chemin d'accès « /usr/bin/bash » va parcourir le contenu des inodes suivants et renverra 21



La fonction namei : algorithme

Pseudo-code pour 2 niveau d'indirections:

```
int namei(path) :

     inode=ROOT_INODE
     pour chaque composant du path p:
          inode=lookup_entry(inode,p)
     renvoyer inode
```

La fonction lookup_entry

C'est une simple fonction de recherche linéaire d'une chaîne de caractère dans les entrées d'un répertoire dont l'inode est donné, elle renvoie le numéro d'inode d'un nom de fichier donné en paramètre à part que :

- Le répertoire peut-être réparti sur plusieurs blocs : il faut utiliser bmap

- La taille du répertoire n'est pas forcément un multiple de la taille des blocs : il faut utiliser le champs taille de l'inode pour limiter la recherche.

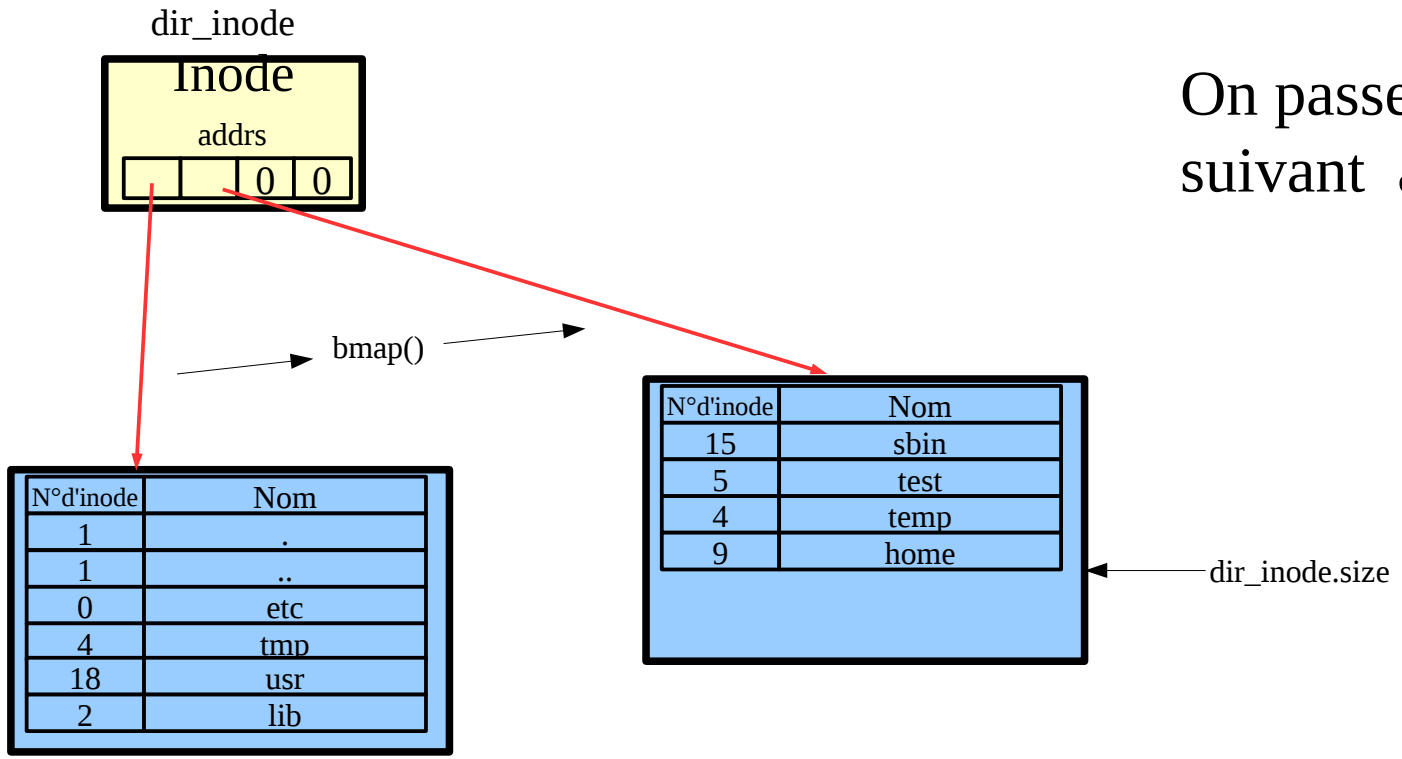
La fonction lookup_entry : algorithme

Pseudo-code :

```
int lookup_entry(dir_inode, name) :

     Pour chaque bloc de dir_node et dans la limite de dir_inode.size

          Comparer le nom de chaque entrée du bloc avec name
          Si l'un des entrées correspond à name, renvoyer le numéro d'inode correspondant
```



On passe d'un bloc au bloc suivant avec bmap()

Les fonctions add_entry et del_entry

Fonctions similaires à lookup_entry : parcours des entrées du répertoires pouvant être réparties sur plusieurs blocs.

- Pour add_entry, on cherche la première entrée libre, c'est à dire dont le numéro d'inode est à zéro, pour y insérer le nouveau nom et le nouveau numéro d'inode, puis **on écrit le bloc modifié sur le disque**.

```
add_entry(dir_inode, name, new_node_num)
```

- Il faut prévoir le cas où on doit rajouter étendre le nombre de blocs du répertoire !

```
del_entry(dir_inode, name)
```

- Pour del_entry, on cherche l'entrée correspondante, puis on y pose le numéro d'inode à zéro, puis **on écrit le bloc modifié sur le disque**.

A propos d'encodage de noms de fichiers

Les fonctions `lookup_entry`, `add_entry`, `del_entry` et `namei` prennent toutes une chaîne de caractère en paramètre.

Le format exact de la chaîne de caractère est en fait **libre**, puisque dans les répertoires, ces chaînes de caractères sont stockées sous forme de tableaux d'octets. La seule limite est la taille de ce tableau d'octets, par exemple `14` sous Minix fs version 1.0

Ainsi, même si les appels systèmes `open()`, `create()`, `mkdir()`, `...` prennent un tableau de char en paramètre, en réalité il s'agit d'un tableau d'octets et on peut manipuler bien plus que de l'ascii avec ces fonctions, sans les réécrire. Essayer `strace mkdir é` pour s'en convaincre