

# chmod e fchmod

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int chmod(const char *path, mode_t mode);
```

```
int fchmod(int fildes, mode_t mode);
```

- Consentono di modificare i permessi di accesso ai file
  - chmod prende in input un pathname
  - fchmod prende in input un file descriptor (file deve essere aperto).
- Il parametro “mode” puo' essere un bitwise OR delle seguenti costanti:
  - S\_ISUID, S\_ISGID, S\_ISVTX (set User/GroupID, saved text-sticky bit)
  - S\_IRWXU, S\_IRUSR, S\_IWUSR, S\_IXUSR (read,write,exec user)
  - S\_IRWXG, S\_IRGRP, S\_IWGRP, S\_IXGRP (read,write,exec group)
  - S\_IRWXO, S\_IROTH, S\_IWOTH, S\_IXOTH (read, write,exec world)

In <sys/stat.h>

# chmod e fchmod

**setuid e setgid:** permettono agli utenti di lanciare eseguibili con i permessi del owner. Spesso usati per lanciare programmi con elevati privilegi temporanei per eseguire task specifici.

**Sticky Bit:** applicato alle directory blocca gli item contenuti permettendone la modifica (rinomina o cancellazione) solo al proprietario del file, al proprietario della directory ed all'utente root (spesso questo flag viene impostata sulla directory `/tmp` per evitare che utenti ordinari cancellino o spostino i file appartenenti agli altri utenti).

# Esempio

```
int main(void){
    struct stat statbuf;

    /* Pone a "uno" il bit set-group ID ed a "zero" il permesso di esecuzione per il gruppo */
    if (stat("foo", &statbuf) < 0)
        { printf("stat error for foo"); exit(1)}
    if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0){
        printf("chmod error for foo");

    /* Pone le protezioni a "rw-r--r--" */
    if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
        printf("chmod error for bar");
    exit(0);
}
```

Prima del lancio del programma:

```
ls -l foo bar
```

```
-rw-----
```

```
-rw-rw-rw-
```

Dopo il lancio del programma:

```
ls -l foo bar
```

```
-rw-r--r--
```

```
-rw-rwSrw-
```

# chown

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
int chown(const char *path, uid_t owner, gid_t group);
```

```
int fchown(int fd, uid_t owner, gid_t group);
```

```
int lchown(const char *path, uid_t owner, gid_t group);
```

- Modificano il campo **st\_uid** ed **st\_gid** del file indicato dal pathname (chown e lchown) o dal file descriptor (fchown)
- Se il parametro “owner” o “group” e' uguali a -1, il campo corrispondente non viene modificato
- In molti sistemi, solo un processo del superuser puo' modificare il campo **st\_uid**
- Un processo puo' modificare il gruppo se
  - (a) e' owner del file e
  - (b) il parametro group e' uguale all'effective GID del processo o ad uno dei gruppi “alternativi” (posso modificare solo tra gruppi che mi

# File size

- Il campo **st\_size** della struttura struct contiene la dimensione in byte del file
  - Ha senso solo per file regolari, directory e link
- Il campo **st\_blksize** contiene la dimensione “ottimale” del blocco per operazioni di I/O
  - Ottimizzano le performance di accesso
- Il campo **st\_blocks** indica il numero di blocchi allocati al file

Per i symbolic link, il size e' il numero di byte nel filename:  
lrwxrwxrwx 1 root 7 sep 25 07:14 lib -> usr/lib

(in questo caso size 7)

# File truncation

```
#include <unistd.h>
#include <sys/types.h>
int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```

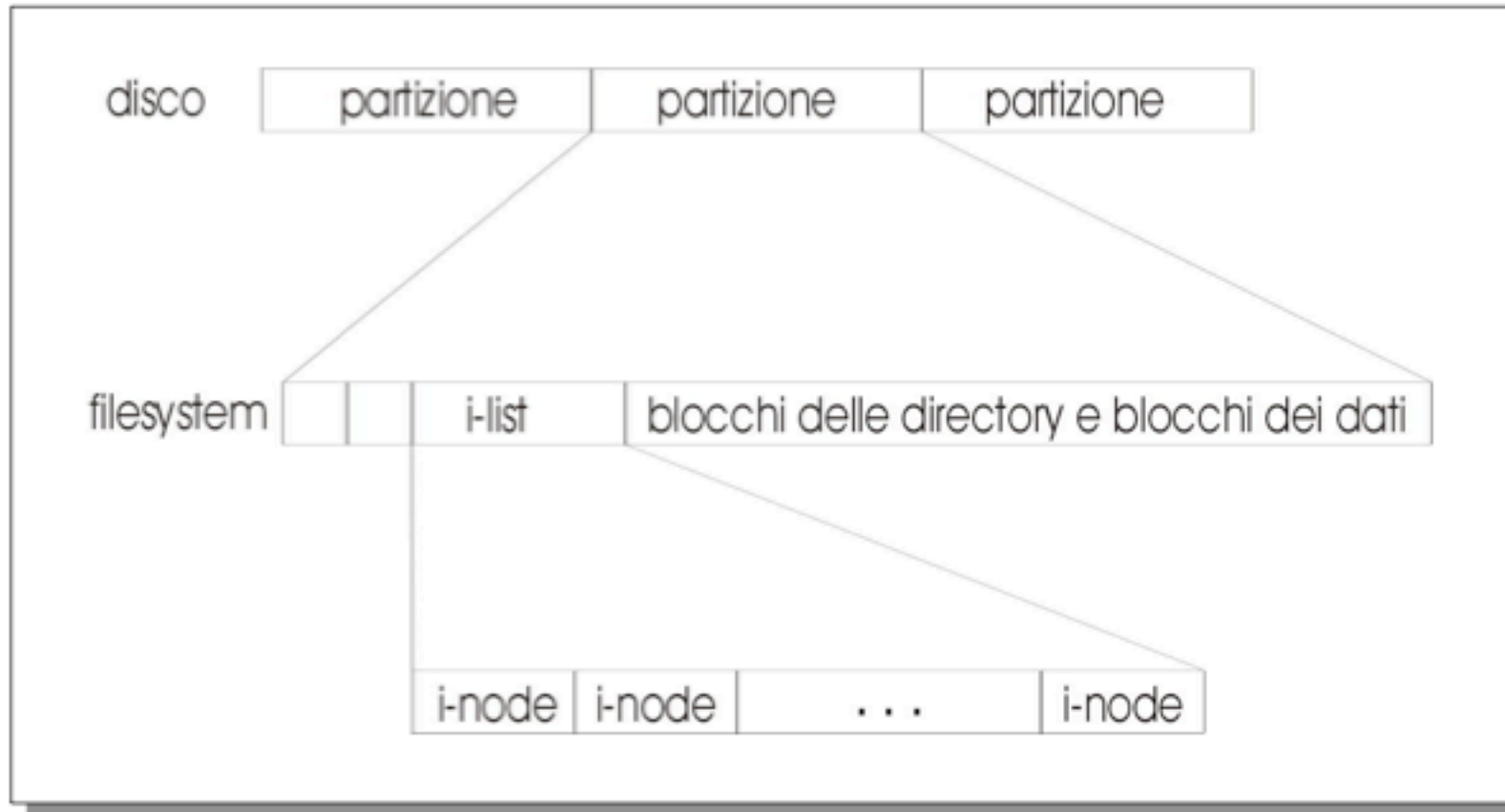
Ritornano: 0 se Ok, -1 su errore

Queste funzioni “troncano” il file dopo “length” bytes.

- Se la dimensione e' maggiore del parametro length, i dati dal byte length+1 non sono piu' accessibili
- Se la dimensione e' minore di length, il comportamento e' system dependent
  - Nei sistemi linux, la dimensione del file viene aumentata e la parte in eccesso contiene “zeri”.

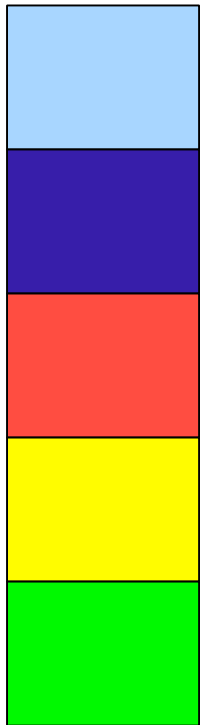
# Unix File System

Il disco e' diviso in una o più partizioni, ogni partizione contiene un file system



Gli i-node sono entry che contengono la maggior parte delle informazioni sui file.

# Unix File system



“Header”: Boot block, superblock, Cylinder group info

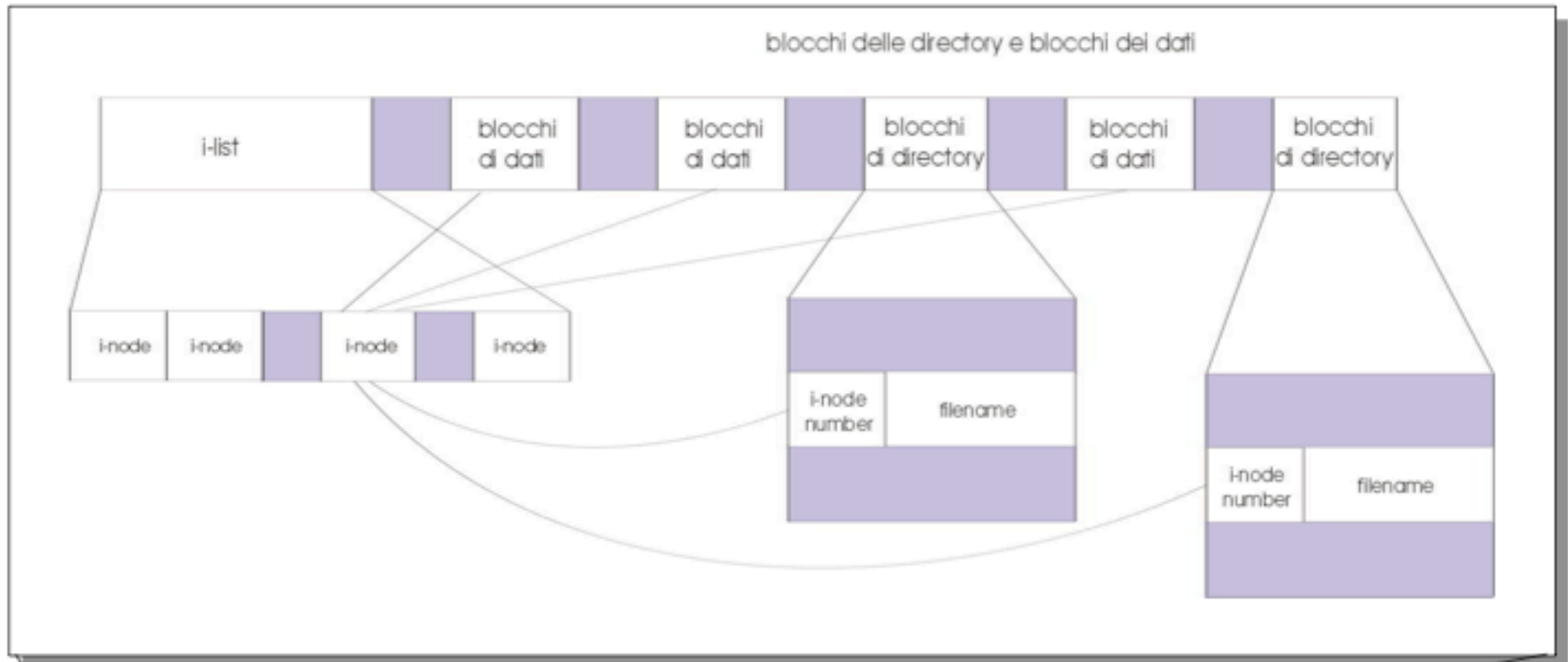
i-node bitmap

data block bitmap

i-node array

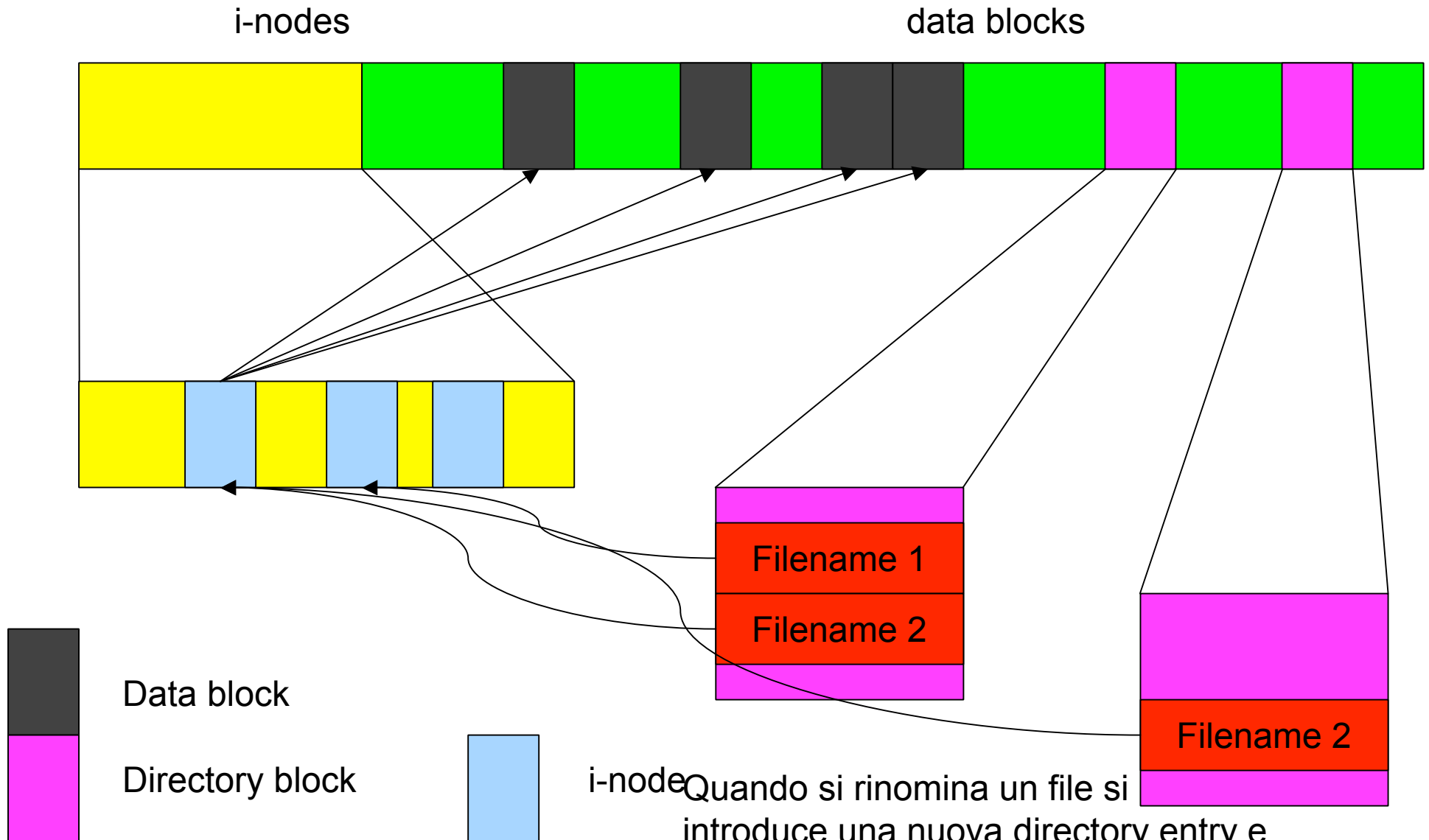
data block array

# Unix File System



Due directory puntano allo stesso i-node; ogni i-node ha un contatore di link che conta quante dir lo puntano. Solo quando il link counter va a zero il file puo' essere

# Unix File system

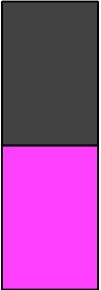
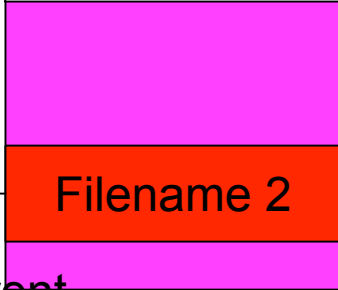
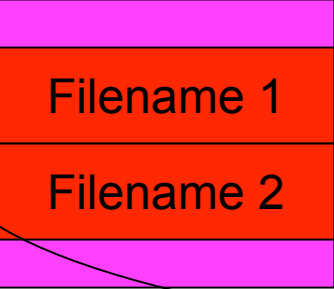
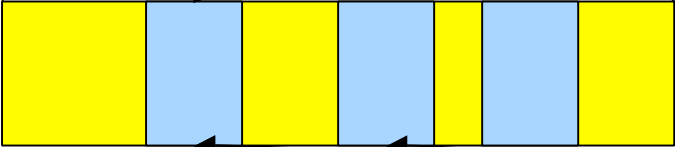


# Unix File System

Quando si crea una nuova directory: `mkdir testdir 2 entry ( . , .. )`

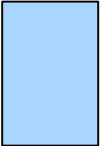
i-nodes

data blocks



Data block

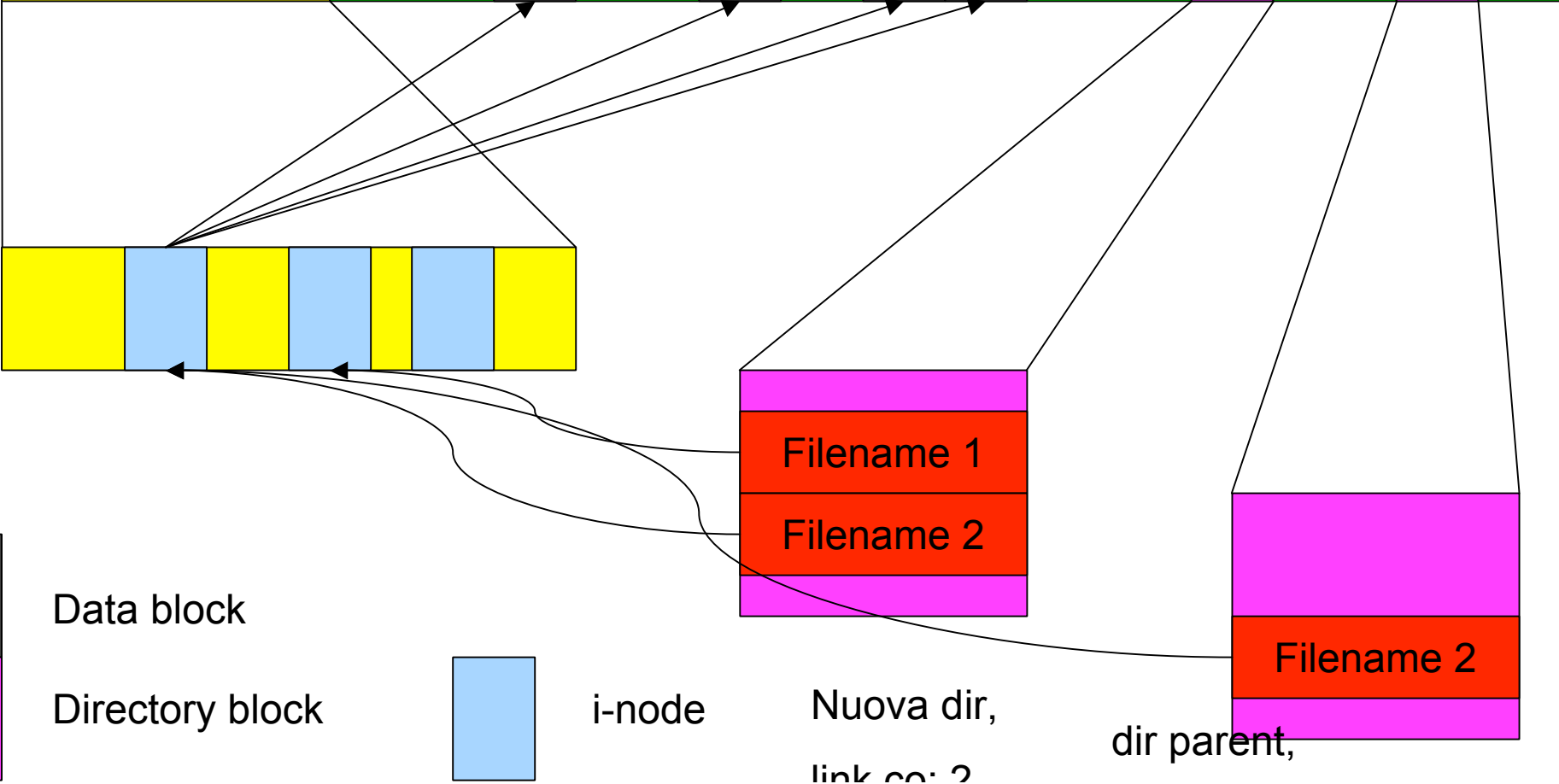
Directory block



i-node

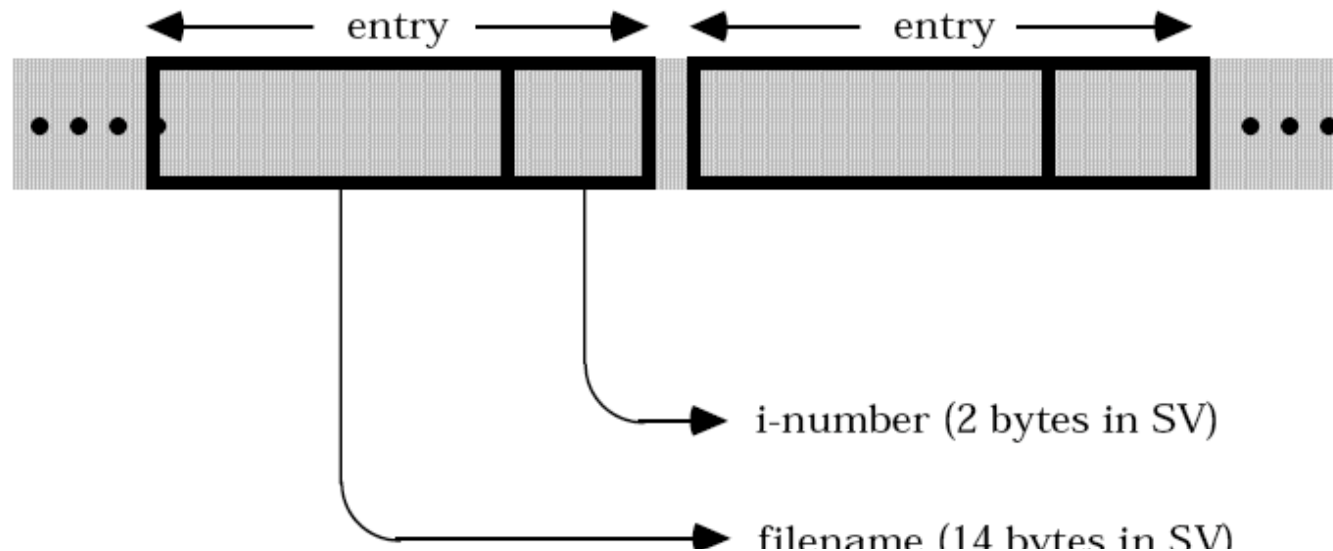
Nuova dir,  
link co: 2

dir parent,



# Directories in Unix

- Sono sequenze di bytes come i file ordinari;
- Differiscono dai file ordinari solo perché non possono essere scritte da programmi ordinari
- Il loro contenuto è una serie di **directory entries**:  
associazione fra gli i-number (usati dal sistema) e i **filename** mnemonici (usati dall'utente):



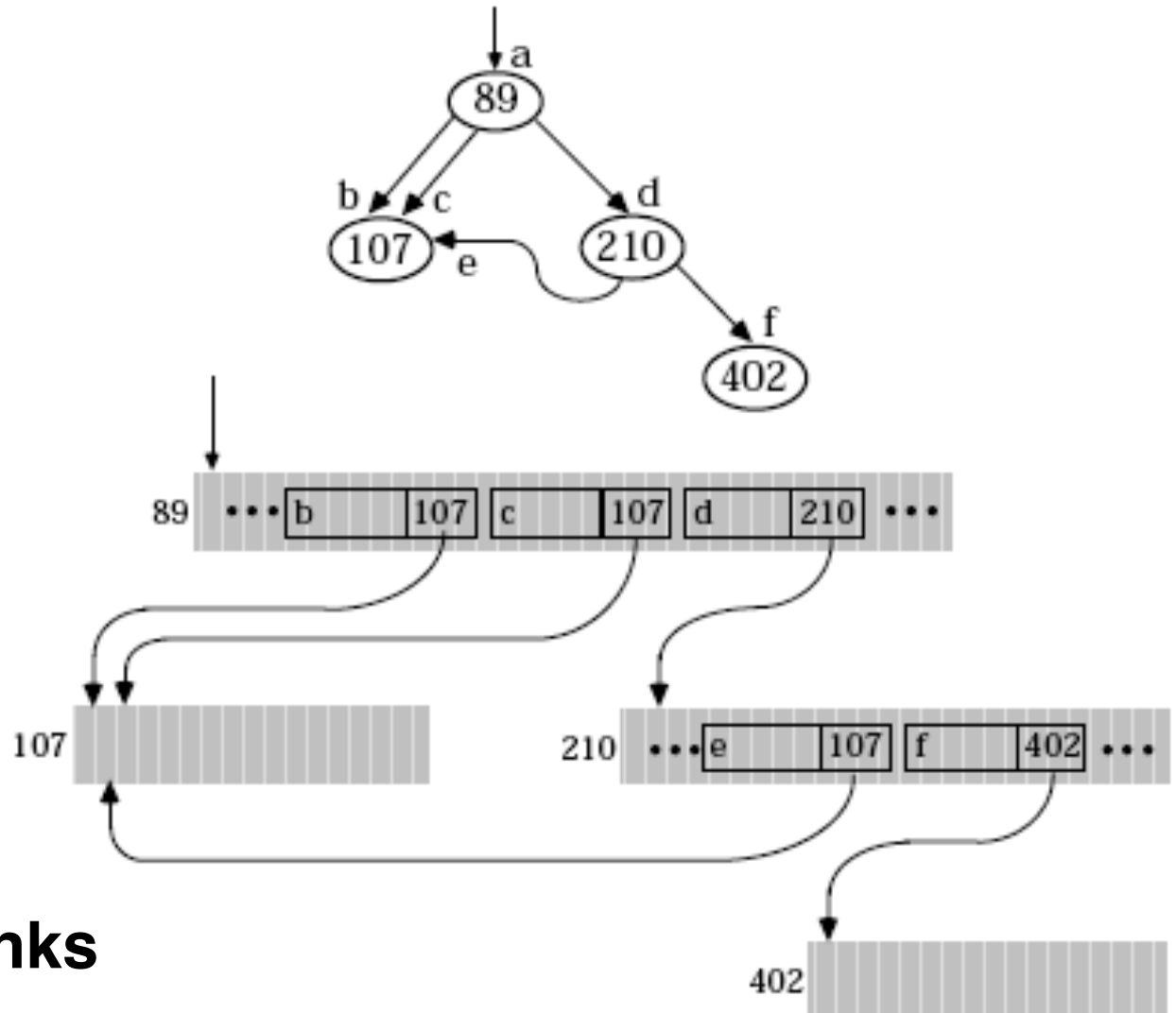
# Unix File system

- La maggior parte delle informazioni della struttura stat sono contenute nell'i-node
- Il nome del file e l'i-node number (**st\_ino**) sono contenute nel directory block che contiene il file (i-node,nome)
- Il campo **st\_nlink** contiene il numero di link al file
  - Il file e' cancellato solo se questo campo ha valore 0.
- Per le directory il link count e' almeno 2
  - Un puntatore dalla directory "padre"
  - Un puntatore dalla directory "."

# Files Sinonimi: links

Un file può avere più filename (ma sempre un solo i-number)

Esempio:



**Il file 107 ha 3 links**

# link/unlink

```
int link(const char *oldpath, const char *newpath);
```

```
int unlink(const char *pathname);
```

Valori di ritorno: 0 se OK; -1 su errore

- link: Crea un (hard) link tra oldpath (file esistente) e newpath.
  - Molti sistemi consentono la creazione di hard link alle directory solo ai processi con UID 0 (superuser per evitare loop!)
- unlink: Rimuove dalla directory entry il riferimento al file ed elimina il file se il campo **st\_nlink** diventa 0.
  - necessari permessi di scrittura ed exec (“ricerca” sulla directory) o, se lo sticky bit e' settato, essere proprietari della directory o del file o essere superuser.
  - Nota: Il kernel elimina il file solo se non vi sono altri processi che lo usano

# rename

```
int rename(const char *oldpath, const char *newpath);
```

Valori di ritorno: 0 se OK; -1 su errore

Rinomina il file indicato da oldpath come newpath.

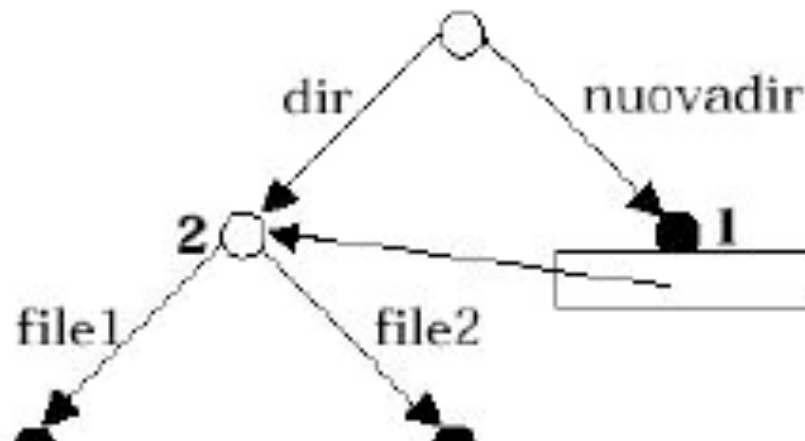
- Se oldpath NON e' una directory, newpath NON puo' essere una directory. Se newpath esiste, viene cancellato.
- Se oldpath e' una directory
  - newpath deve essere una directory vuota
  - newpath non puo' contenere oldpath come prefisso
    - Non e' possibile rinominare /usr come /usr/bin
- Se oldpath o newpath e' un link simbolico, viene processato il link (non il file puntato dal link)

# Link simbolici

- I link simbolici consentono di:
  - Creare link tra entita' su filesystem diversi
  - La creazione di link a directory da parte di utenti.
- Directory entry:
  - Per un hard link contiene il numero di i-node del file “puntato”
  - Per un symbolic link contiene il puntatore ad un “data block” che contiene il nome del file puntato
- Quando la funzione contiene un “pathname” e' necessario controllare sempre se la funzione “segue” il link
  - e.g., “rename” di un symbolic link NON segue il link

# Link Simbolici

- In `-s name1 name2`
- Permette di creare link a directory;
- Permette di creare link fra file o directory che stanno su file system diversi;
- Viene creato un file `name2` che contiene il link simbolico (i.e. il path di `name1`)



# Loop con link simbolici

- `mkdir foo`
- `touch foo/a`
- `ln -s ../foo foo/testdir`
- `ls -l foo`
- ```
-rw-r---- 1 sar 0 jan 00:16 a  
lrwxrwxrwx 1 sar 6 jan 00:16 testdir -> ../foo
```
- Creato una directory foo con un link simbolico che punta a foo
- Si puo' rimuovere con `unlink` (che non insegue il link simbolico), se `hard-link` molto piu' difficile da gestire

# Link simbolici

```
#include <unistd.h>
```

```
int symlink(const char *oldpath, const char *newpath);
```

```
int readlink(const char *path, char *buf, size_t bufsiz);
```

- `symlink` crea un link simbolico tra `oldpath` e `newpath`.
  - Non e' richiesto che `oldpath` esista.
- `readlink`: legge il "nome del file" a cui il link punta (apre il file, legge il link, chiude, mette il nome nel buffer non terminato da null).