

Il File System



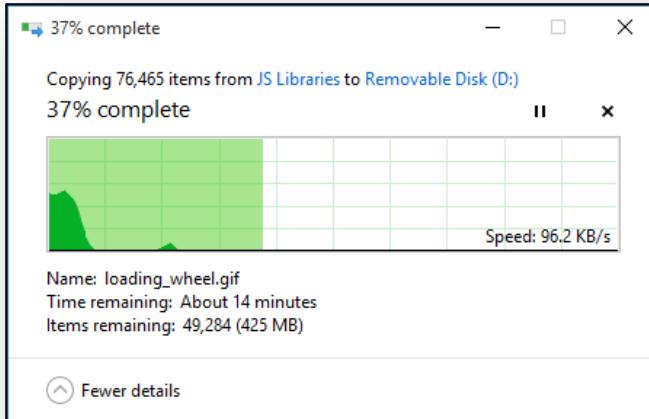
Corso di Laurea in Ingegneria Informatica
Università degli Studi di Napoli Federico II
Anno Accademico 2024/2025, Canale San Giovanni



Il File System

- Sommario
 - Struttura logica e metodi di accesso
 - Organizzazione fisica del file system
 - Cenni ai File System UNIX/Linux e Windows
 - File System per SSD
- Riferimenti
 - P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari, “Sistemi Operativi”, Mc-Graw-Hill (Cap.6)
 - www.ostep.org, Capp. 39, 40, 42
 - W. Stallings, Operating Systems, par. 12.8 – 12.10
 - Dispensa didattica su SSD

Importanza dei filesystem

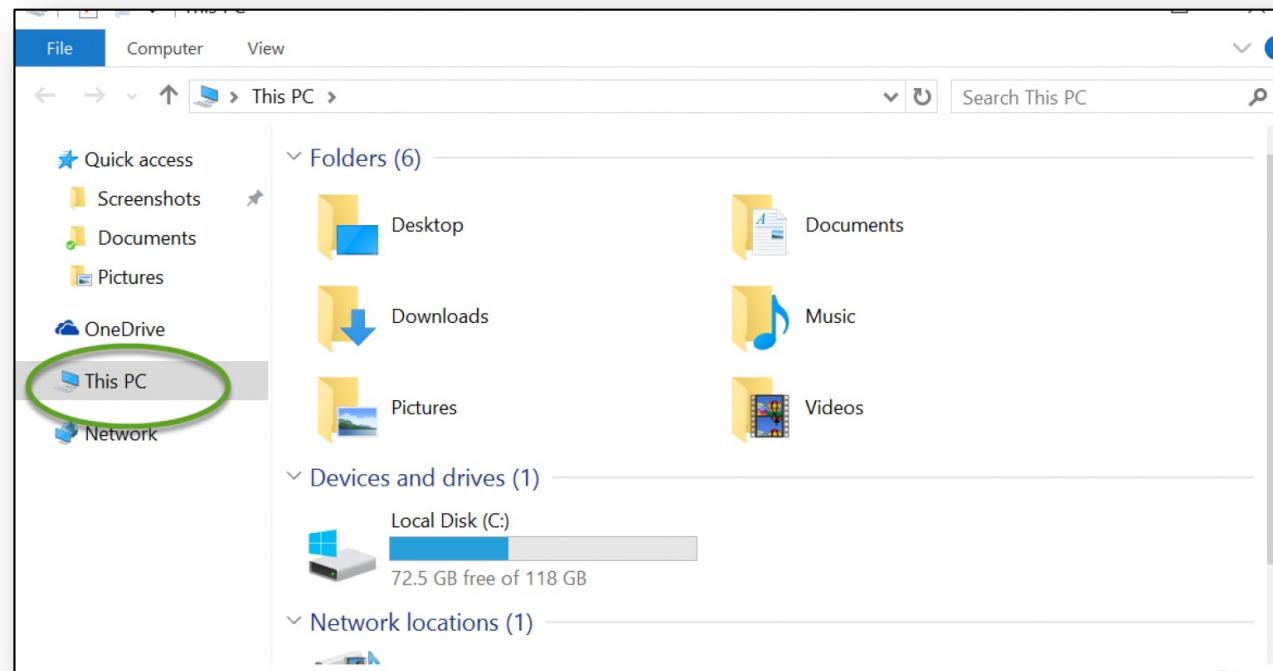


- I filesystem sono un fattore importante per le **prestazioni**
- Conoscere i filesystem è fondamentale per la sicurezza informatica (**digital forensics**)

File System



Il **file system (FS)** è responsabile della gestione e l'organizzazione dei **file** su storage





Obiettivi di un file system

Gli **obiettivi** di un FS sono quelli di garantirne

- 1) la persistenza
- 2) il recupero
- 3) uno schema di naming
- 4) la condivisione e la protezione

Un FS fornisce alle applicazioni una **interfaccia unica** per la gestione dei file (chiamate di sistema)

Struttura di un *file system*



Processi



chiamate di sistema

Struttura logica

Associa dei nomi di file ai dati

Gestisce dati e metadati di un file

Verifica i permessi di accesso

Legge e scrive i dati e metadati su disco

Allocata ed accede ai blocchi su disco

Buffering, disk I/O

Accesso

**Organizzazione
fisica**



device driver

Disco



Struttura logica



Concetto di file

- Un file è la prima **astrazione** realizzata dal FS
 - un insieme di informazioni
 - raggruppate e salvate in memoria secondaria
 - a cui è assegnato un identificativo simbolico (nome)



Concetto di file



```
$ cat ciao.txt
```

```
ciao
```

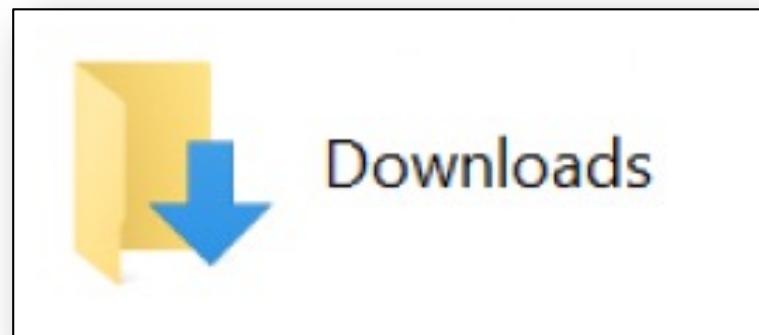
```
$ hexdump -C ciao.txt
```

```
00000000  63 69 61 6f 0a          |ciao.|
```



Concetto di directory

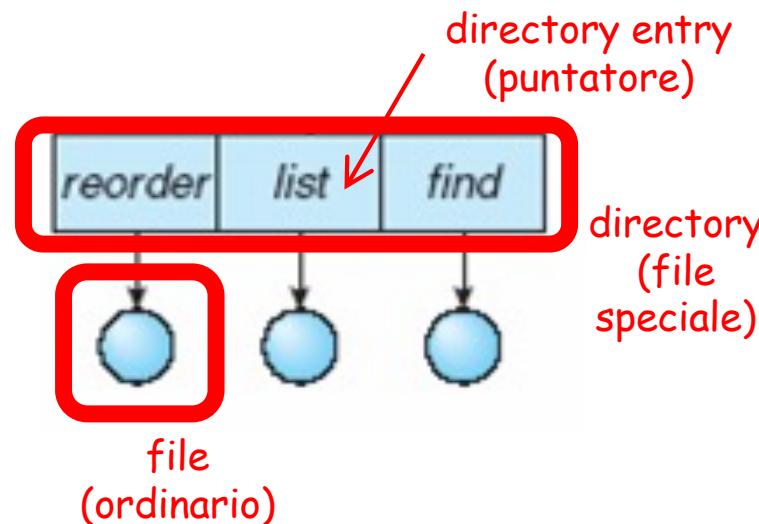
- La **directory (cartella)** è la seconda principale astrazione dei FS, per "raggruppare" più file





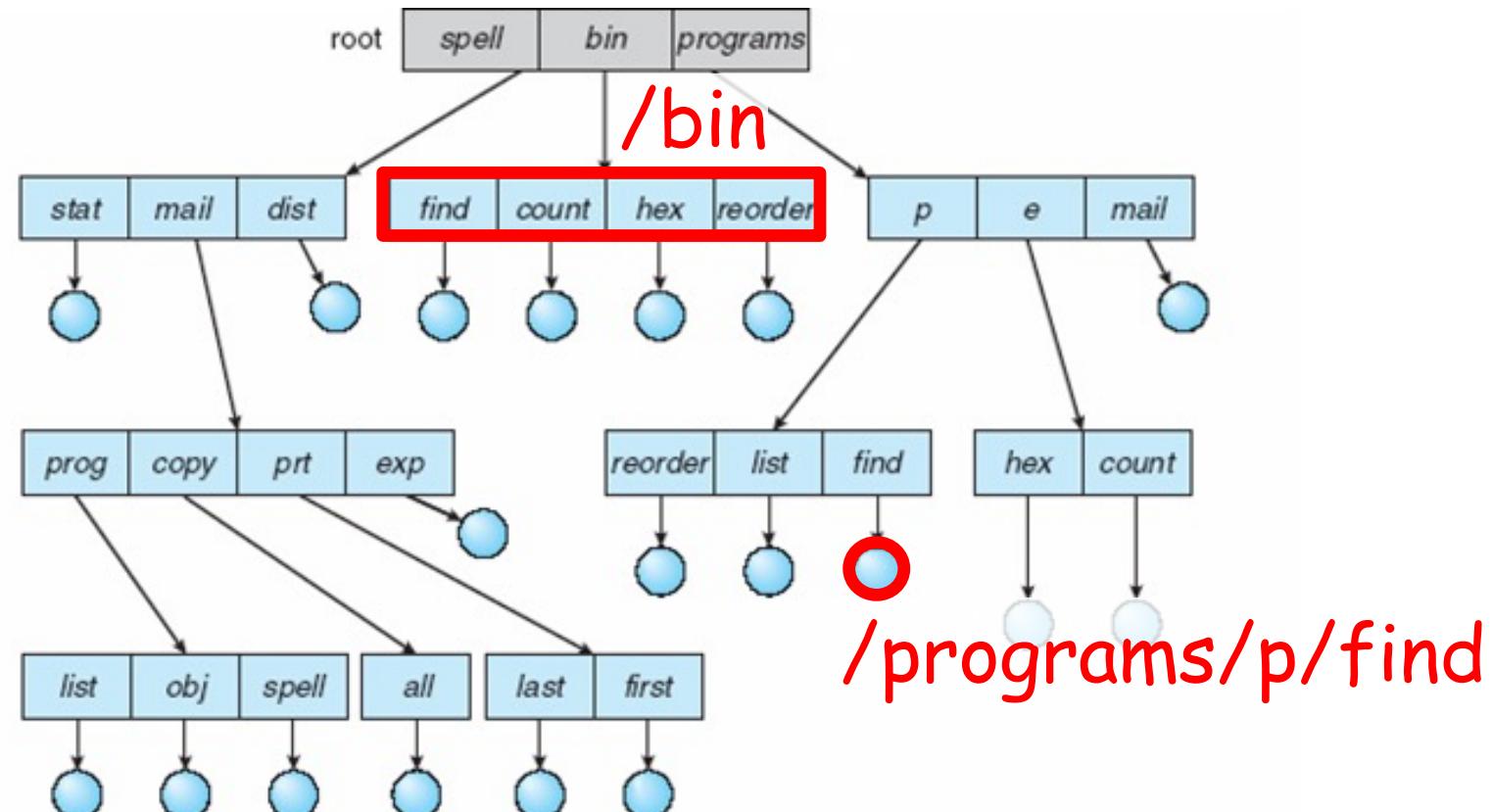
Concetto di directory

- Le directory sono trattate dal SO come file "speciali"
- Non contengono dati, ma puntatori ai file
- Directory Entry = nome del file + riferimento al file su disco





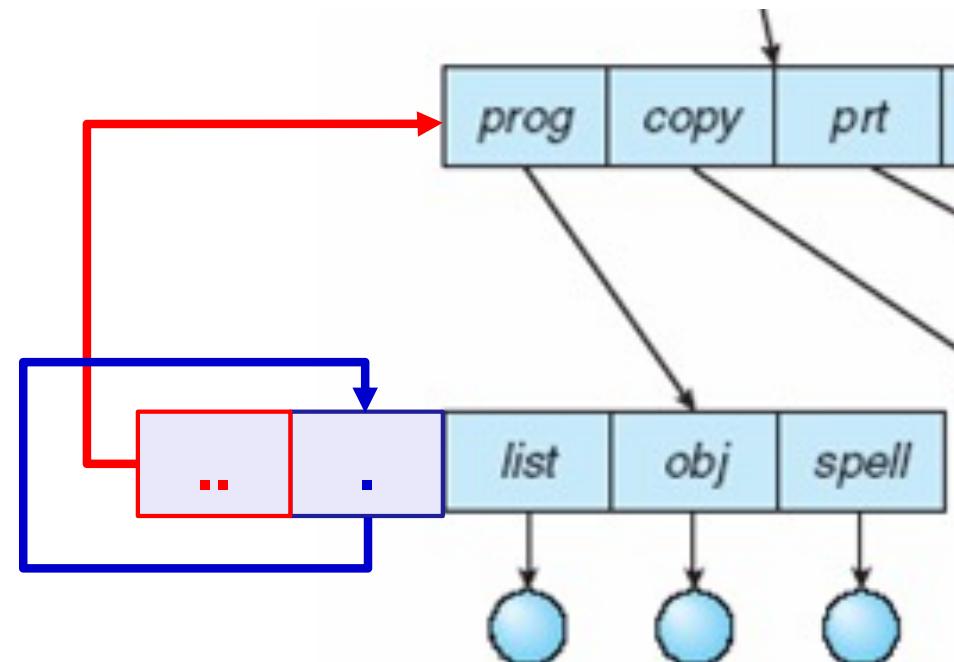
Struttura di directory ad albero





File e directory in Linux e UNIX

- In Linux/UNIX, ogni cartella include automaticamente due entries
- "`.`" è un link alla cartella stessa
- "`..`" è un link alla cartella-padre





File e directory in Linux e UNIX

- Per informazioni sui metadati: `ls -l`

Numero di riferimenti
(directory entry) al
file o directory.

```
so@so-vbox:~/so_esempi$ ls -l
totale 68
drwxrwxr-x 2 so so 4096 nov 16 14:33 bank
drwxrwxr-x 2 so so 4096 ott 25 12:34 code
drwxrwxr-x 2 so so 4096 ott 21 18:12 come
drwxrwxr-x 2 so so 4096 ott 22 20:01 gestione
drwxrwxr-x 4 so so 4096 ott 25 12:34 lettere
drwxrwxr-x 2 so so 4096 ott 16 14:20 librerie
drwxrwxr-x 2 so so 4096 nov 30 17:01 linux_mem
drwxrwxr-x 8 so so 4096 ott 25 12:34 moduli-kernel
drwxrwxr-x 7 so so 4096 ott 25 12:34 monitor
drwxrwxr-x 2 so so 4096 ott 25 12:34 produttore_consumatore
drwxrwxr-x 6 so so 4096 ott 25 12:34 PThreads
-rw-rw-r-- 1 so so 0 nov 16 14:33 test
drwxrwxr-x 2 so so 4096 ott 25 12:34 vedi
drwxrwxr-x 2 so so 4096 ott 25 12:34 vedi2
drwxrwxr-x 7 so so 4096 ott 25 12:34 vedi3
drwxrwxr-x 8 so so 4096 ott 25 12:34 vedi4
so@so-vbox:~/
```

Dimensione del file in byte.

Le directory sono un file di 4KB (pre-alloca righe nella tabella dei nomi), espandibile.

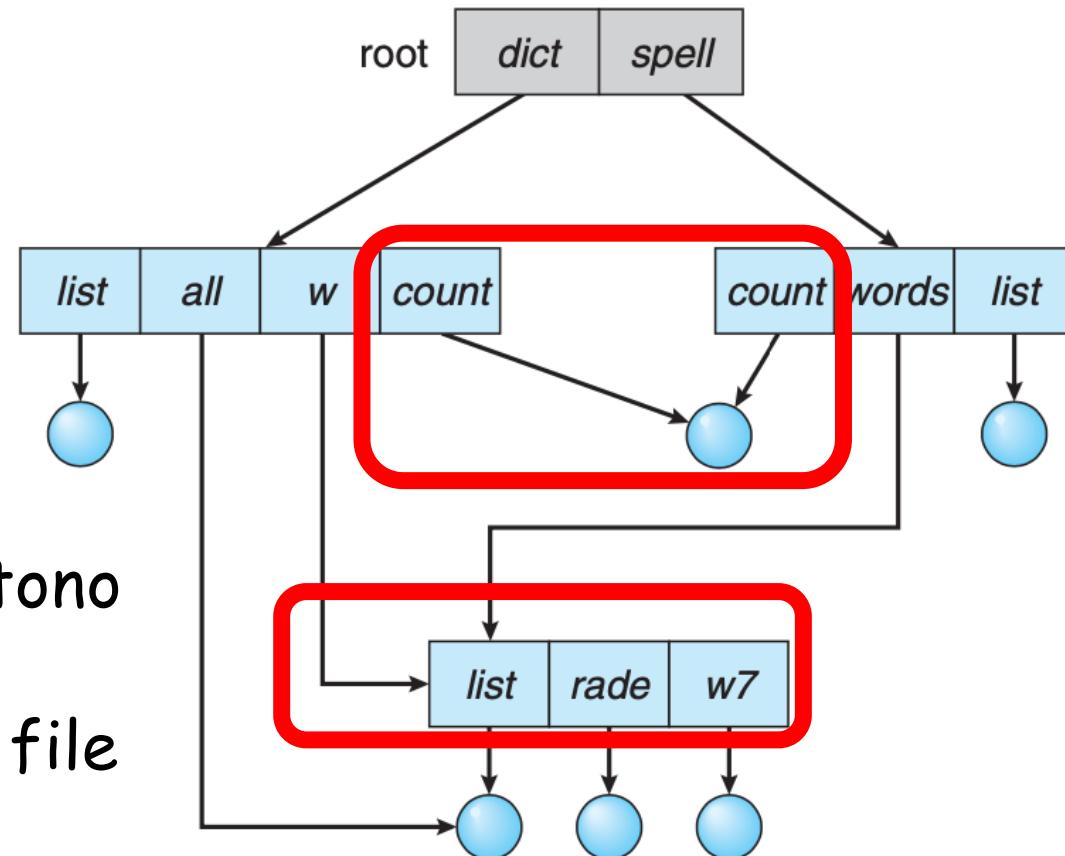
Nel caso dei **file**, è inizialmente **1**.

Nel caso delle **directory**, include:

- Il riferimento nella dir. padre
- Il riferimento della dir. a sè stessa ("**.**")
- I riferimenti nelle eventuali dir. figlie ("**..**")



Struttura di directory a grafo aciclico

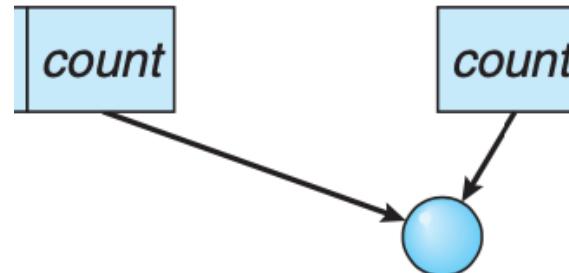


Molti sistemi permettono
a più directory di
condividere lo stesso file
o sotto-directory
(meccanismo del "link")

I link



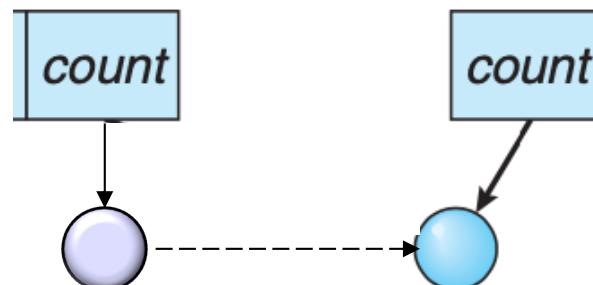
- **Hard link**: le entry nella directory hanno entrambe un **puntatore** allo **stesso settore** su disco
 - Accesso più veloce
 - Dipende dalla struttura fisica





I link

- **Soft link:** è un file "fittizio", distinto da quello condiviso
 - Ad esempio, contiene il **percorso del file condiviso**
 - Non dipende dalla posizione dei dati sul disco fisico
 - Permette di collegare file su **file system di tipo differente**
 - È usato nei sistemi Windows



📁 My Stuff	10/18/2011 2:52 PM	File folder
📁 Project	10/18/2011 2:55 PM	File folder
📄 Imp.txt	8/9/2011 11:17 AM	Text Document 2 KB
📄 index.php	12/7/2008 1:37 AM	PHP File 8 KB
📄 licensing.txt	8/15/2011 4:17 PM	Text Document 1 KB
📄 links.txt	9/6/2011 1:28 PM	Text Document 1 KB



Link in Linux e UNIX

- Hard link: `ln <file_iniziale> <file_nuovo_link>`
- Soft link: `ln -s <file_iniziale> <file_nuovo_link>`

```
so@so-vbox:~/so_esempi$ ln readme.md hard_link.md
so@so-vbox:~/so_esempi$ ln -s readme.md soft_link.md
so@so-vbox:~/so_esempi$ ls -l
total 72
drwxrwxr-x 2 so so 4096 nov 16 14:33 banker_alg
drwxrwxr-x 5 so so 4096 ott 25 12:34 code_messaggi
drwxrwxr-x 2 so so 4096 ott 21 18:43 compilazione
drwxrwxr-x 2 so so 4096 ott 21 18:42 gdb
drwxrwxr-x 2 so so 4096 ott 22 20:01 gestione_processi
-rw-rw-r-- 2 so so 85 ott 16 14:18 hard_link.md
drwxrwxr-x 4 so so 4096 ott 25 12:34 lettore_script
drwxrwxr-x 2 so so 4096 ott 16 14:20 librerie
drwxrwxr-x 2 so so 4096 nov 30 17:01 linux_mem
drwxrwxr-x 8 so so 4096 ott 25 12:34 moduli_kernel
drwxrwxr-x 7 so so 4096 ott 25 12:34 monitor
drwxrwxr-x 2 so so 4096 ott 25 12:34 produttore_coda
drwxrwxr-x 6 so so 4096 ott 25 12:34 PThreads
-rw-rw-r-- 2 so so 85 ott 16 14:18 readme.md
drwxrwxr-x 2 so so 4096 ott 25 12:34 sensori
drwxrwxr-x 2 so so 4096 ott 29 19:26 shared_memory
drwxrwxr-x 7 so so 4096 ott 21 18:39 shell_scripting
lrwxrwxrwx 1 so so 9 dic 1 21:15 soft_link.md -> readme.md
drwxrwxr-x 8 so so 4096 nov 9 15:35 utente
so@so-vbox:~/so_esempi$
```

Hard link,
indistinguibile
dal file iniziale.

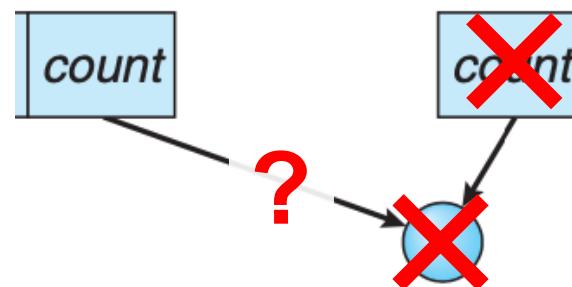
Il file iniziale
(è il primo hard link)

Soft link, non incrementa il
numero di riferimenti. È un
file "fittizio" di pochi byte.



Rimozione dei link

- Per **rimuovere un file**, occorre rimuovere la **entry** dalla directory che lo contiene
- Nel caso di **link**, occorre prestare attenzione a quando rimuovere il **file dal disco**
- C'è rischio di avere **link ad un file non più esistente** su disco!



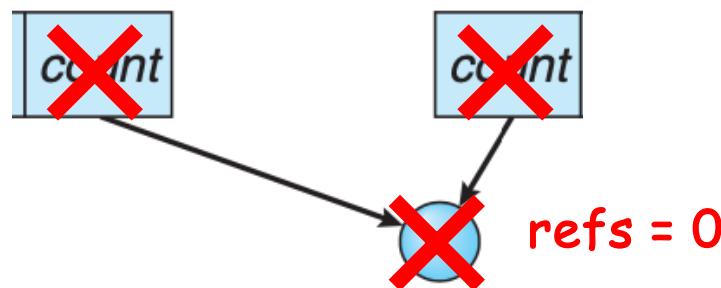


Rimozione dei link

Rerefence counting:

Il file è **conservato** su disco, fin quando esiste **almeno un riferimento** a esso (es. Linux e UNIX, system call *unlink()*).

Richiede di gestire, per ogni file, un **conteggio dei riferimenti**.



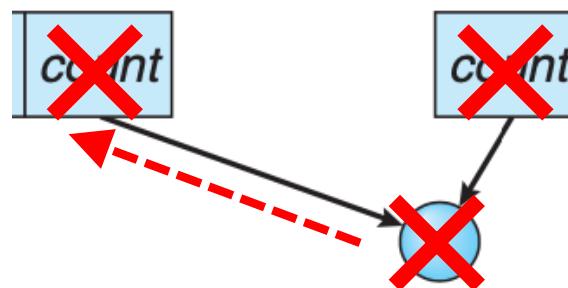


Rimozione dei link

Backpointers:

Quando si cancella un link, si **cercano e rimuovono** anche il file e tutti gli altri link.

Richiede un doppio collegamento tra directory e file





Gestione dei file in UNIX

- UNIX distingue 6 tipi di file:
 1. Ordinary: file dati (sequenza di byte)
 2. Directory
 3. Hard Link
 4. Symbolic Link
 5. Special: astrazione dei dispositivi di I/O
 6. Named Pipe: meccanismo di IPC via file



Metodi di accesso



Operazioni su file

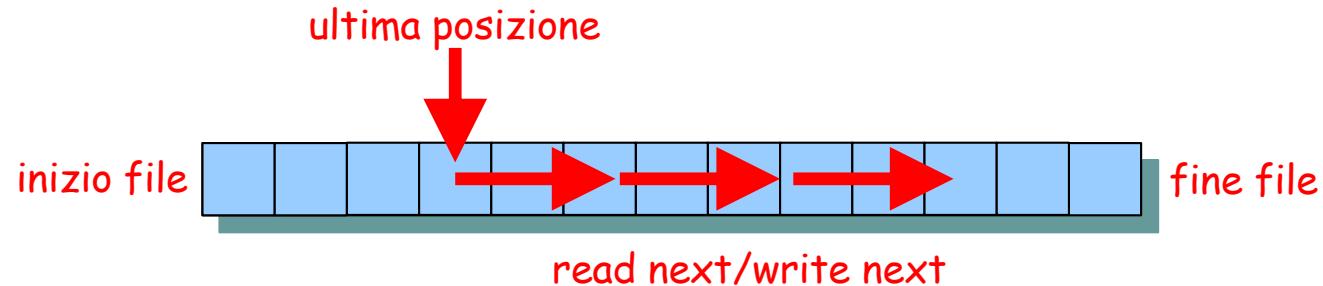
- creazione
- lettura
- cancellazione
- apertura (open)
- scrittura
- riposizionamento nel file (seek)
- chiusura (close)

Molti FS forniscono un'interfaccia conforme allo standard **POSIX**



Accesso sequenziale

Accesso sequenziale: Per ogni file aperto, il FS ha un puntatore all'ultima posizione acceduta. Dopo una operazione di lettura/scrittura, il puntatore **avanza al byte successivo**

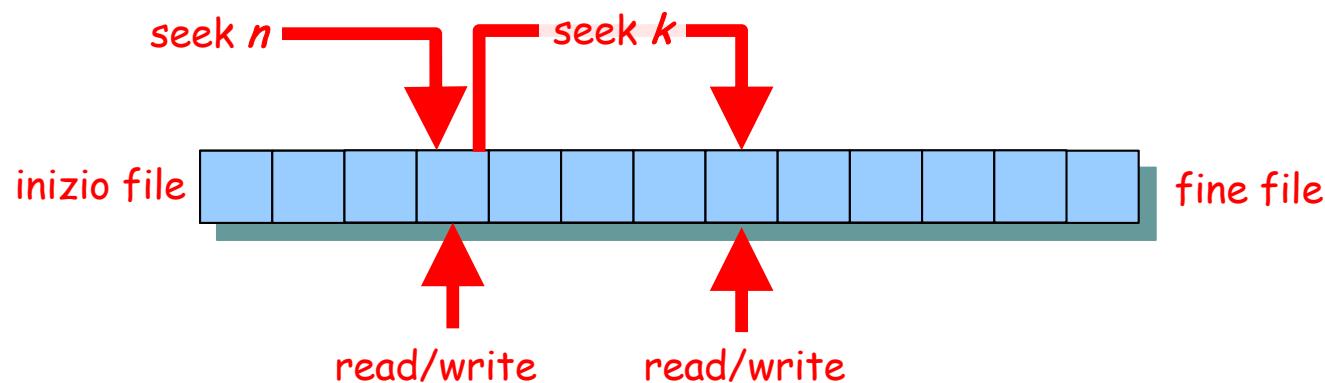


```
int fd = open("file.txt", O_RDONLY);  
...  
do {  
    byte_read = read(fd, buffer, 100);  
    ...  
} while(byte_read > 0);  
  
close(fd);
```

Accesso diretto



Accesso diretto: Il processo **seleziona (seek)** una posizione arbitraria (random) da accedere, che sarà letta/scritta alla prossima operazione

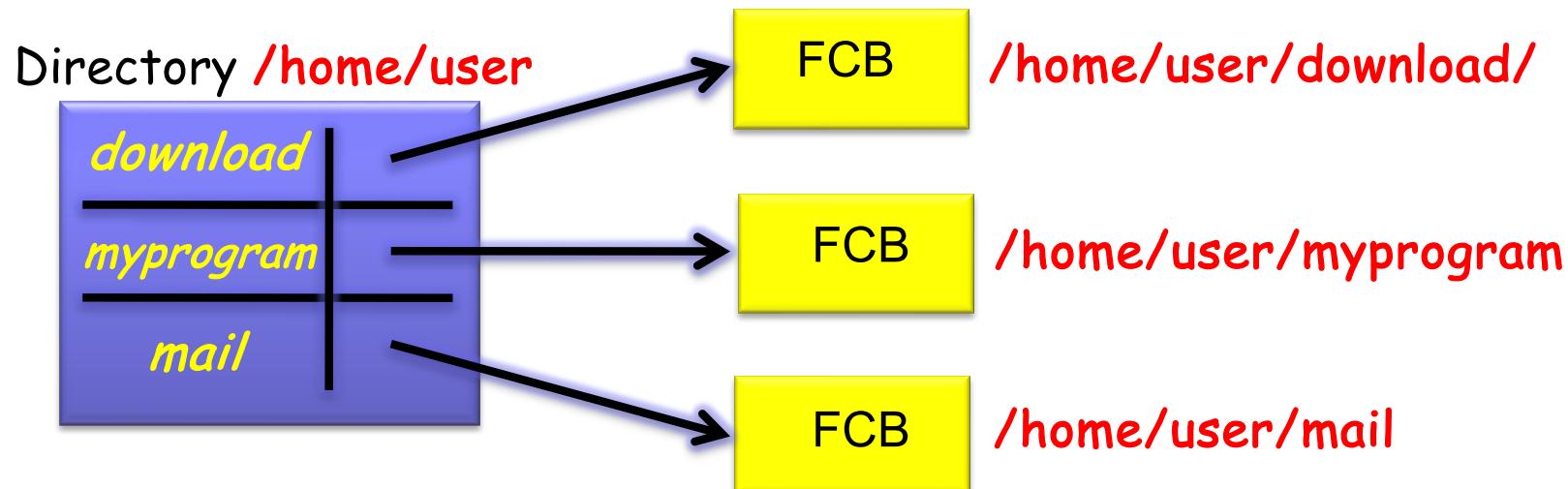


```
FILE* fd = open("file.txt", "r");  
...  
fseek(fd, 10, SEEK_SET);  
byte_read = fread(buffer, 1, 100, fd);  
...  
fclose(fd);
```



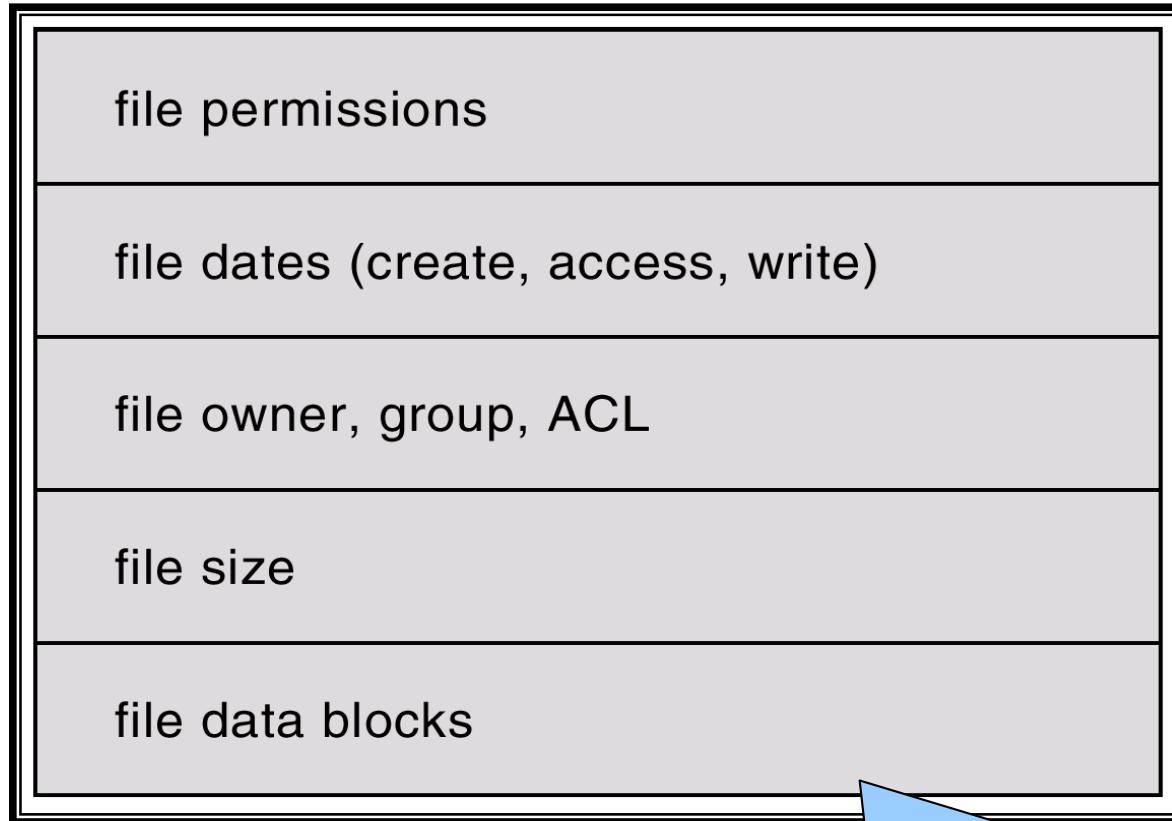
File Control Block

La **directory entry** punta ad un **File Control Block (FCB)**
È una struttura dati con le informazioni sul file





Tipico FCB (Descrittore di File)

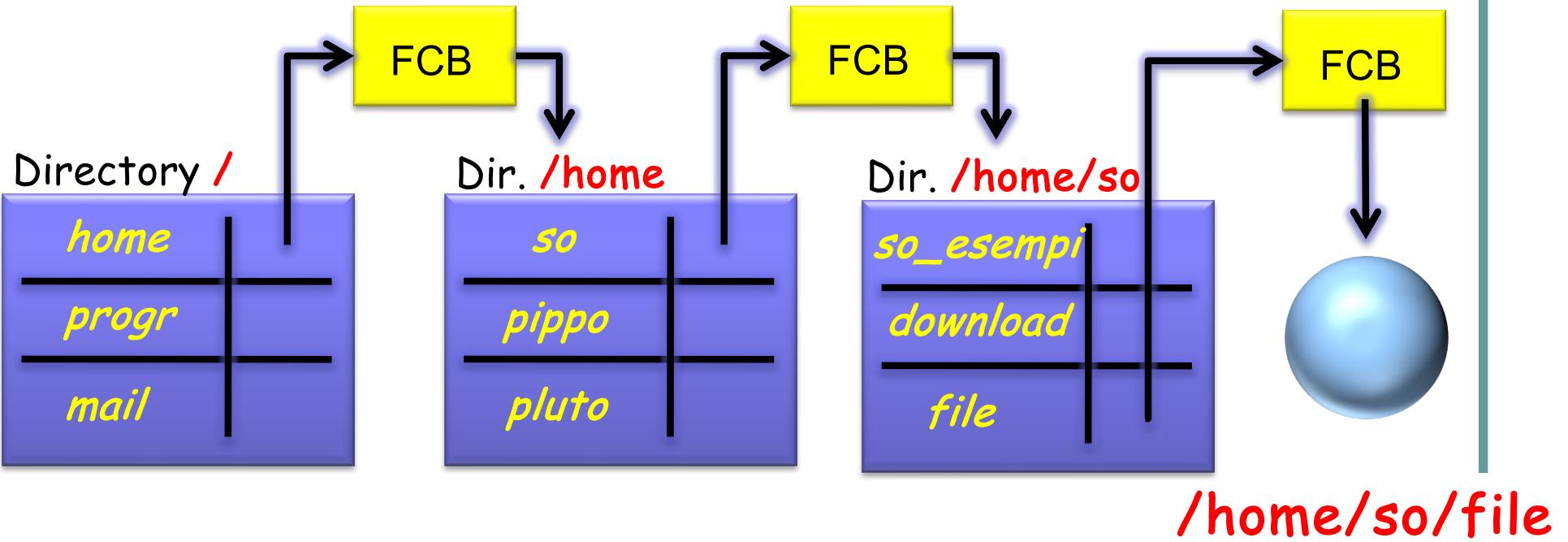


Nota: il **nome del file** non è un attributo del FCB!
Il nome è conservato nella **directory entry**



Accesso al file

- Per accedere ad un percorso (**path**), il FS:
 - Percorre tutte le directory e i FCB...
 - ...fino ad arrivare ai blocchi su disco del file



Ognuno di questi è a sua volta un **accesso al disco!**
Gli FCB più usati sono mantenuti in RAM (**caching**)

Accesso al file



L'applicazione legge un file

```
int fd = open("./file.txt", O_RDWR);  
read(fd, buffer, 100);
```

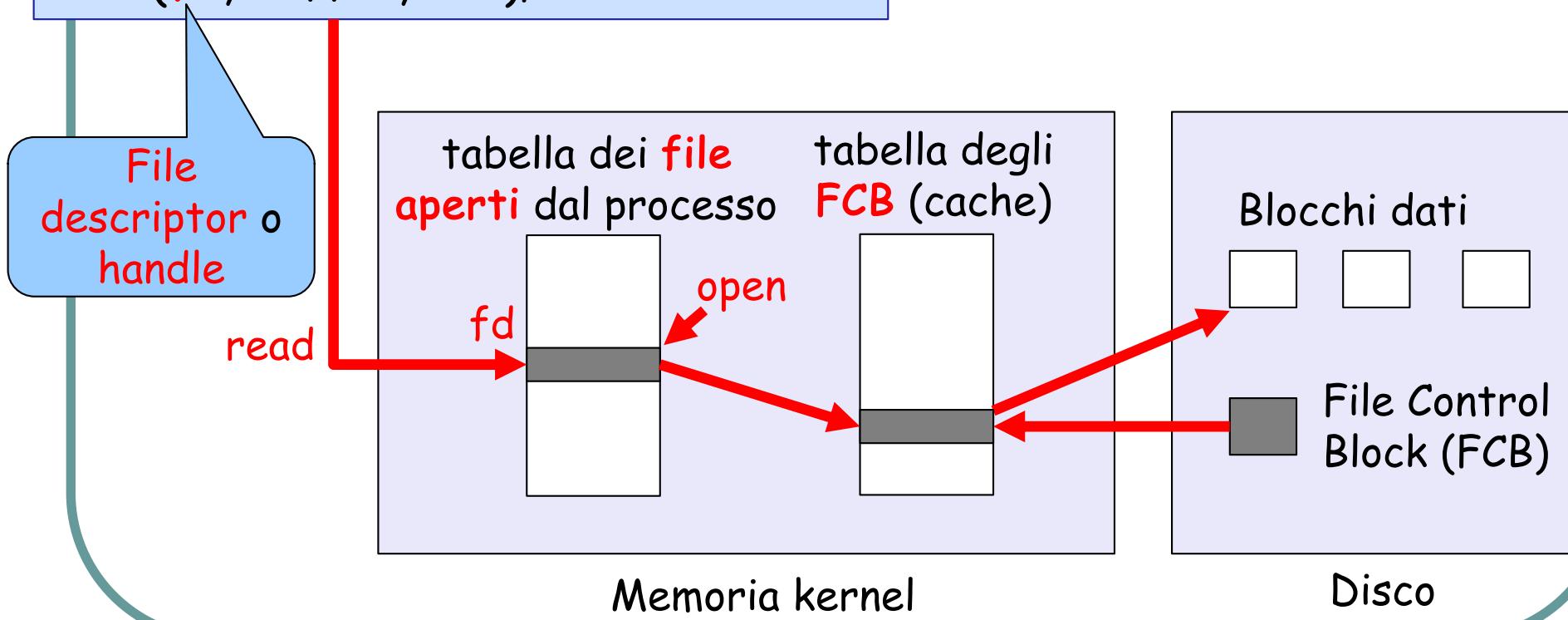




Tabella dei file aperti (Linux)

```
$ sudo ls -l /proc/2081/fd/
```

```
totale 0
lrwx----- 1 lxd vboxsf 64 nov 30 04:22 0 -> /dev/pts/0
lrwx----- 1 lxd vboxsf 64 nov 30 04:22 1 -> /dev/pts/0
lrwx----- 1 lxd vboxsf 64 nov 30 04:24 10 -> /var/lib/mysql/#ib_16384_0 dblwr
lrwx----- 1 lxd vboxsf 64 nov 30 04:24 11 -> /var/lib/mysql/#ib_16384_1 dblwr
lrwx----- 1 lxd vboxsf 64 nov 30 04:24 12 -> /var/lib/mysql/undo_001
...
...
```



Organizzazione fisica



Il layout del disco

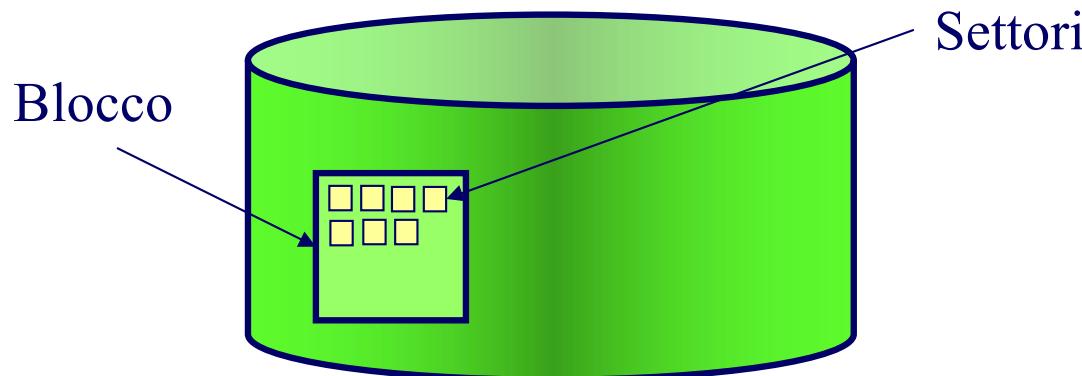
- Dal punto di vista del SO, il disco è un **vettore lineare di blocchi**





Block vs. Sector

- L'unità di gestione del FS è il **blocco**, che in genere è maggiore di quella del settore del disco
- Ogni blocco contiene un **insieme di settori contigui**

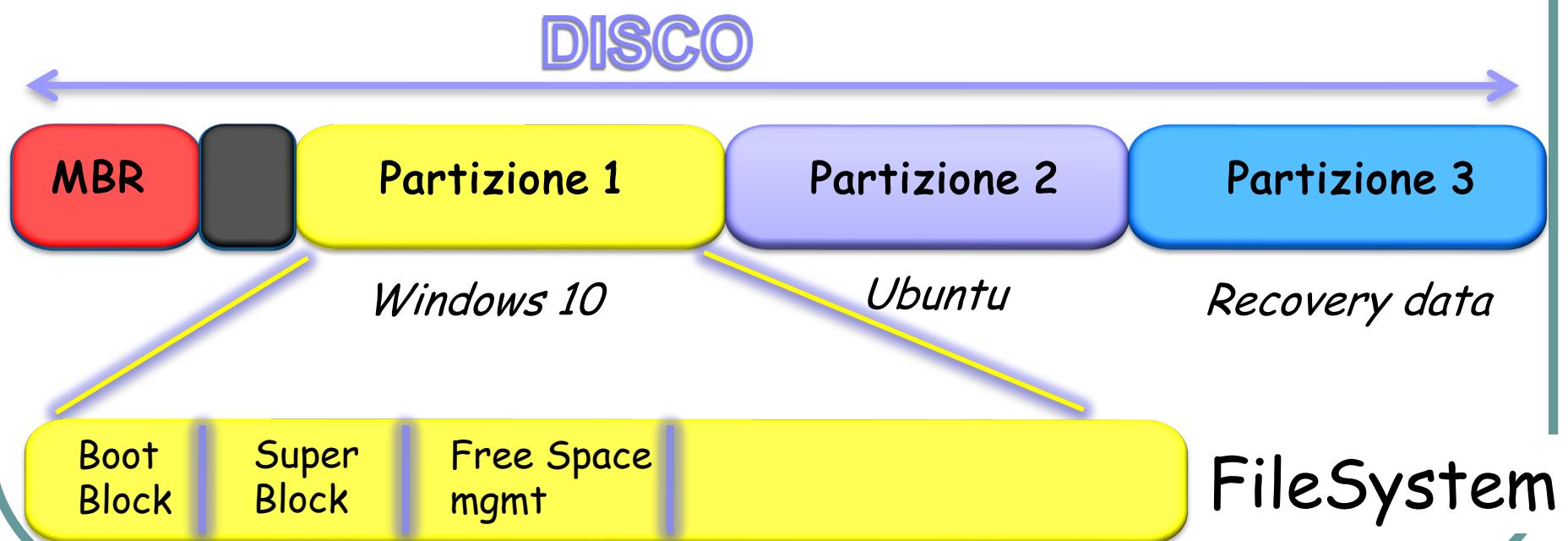


Migliora l'**efficienza** di trasferimento dei dati (evita i **tempi di seek**).
La dimensione del blocco è tipicamente uguale alla dimensione di una **pagina in memoria**



Il layout del disco

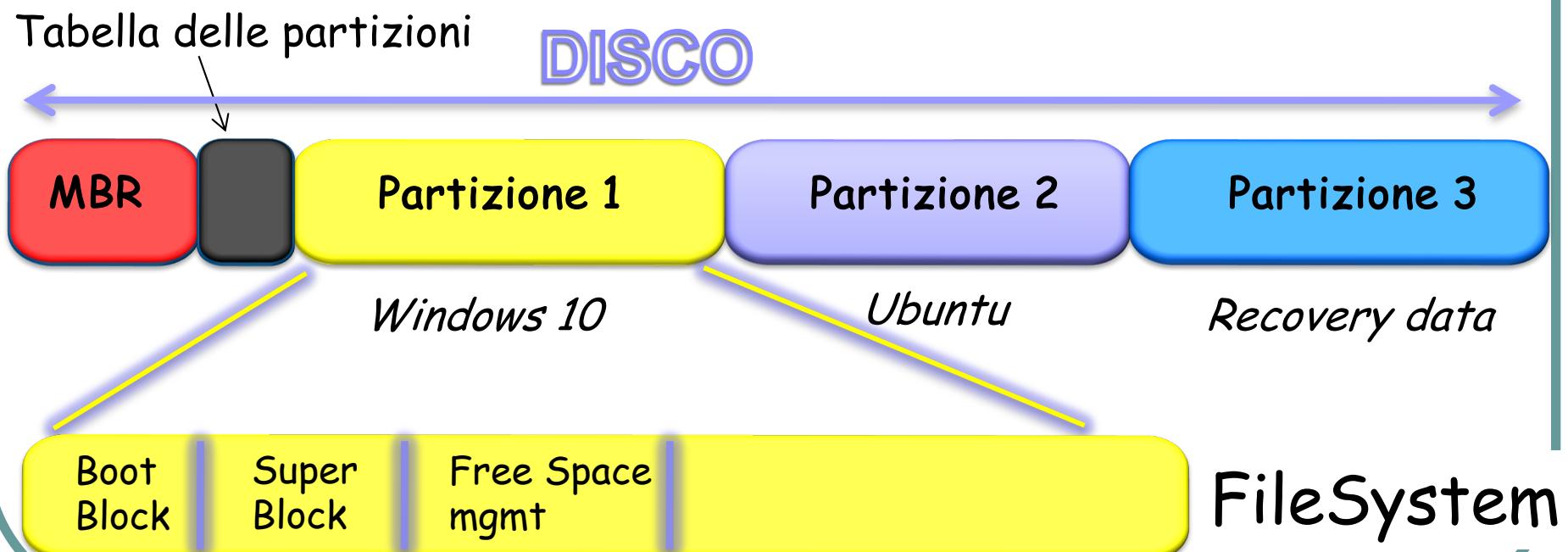
- Un disco può essere suddiviso in **partizioni**, ognuna con un proprio **file system**





Il layout del disco

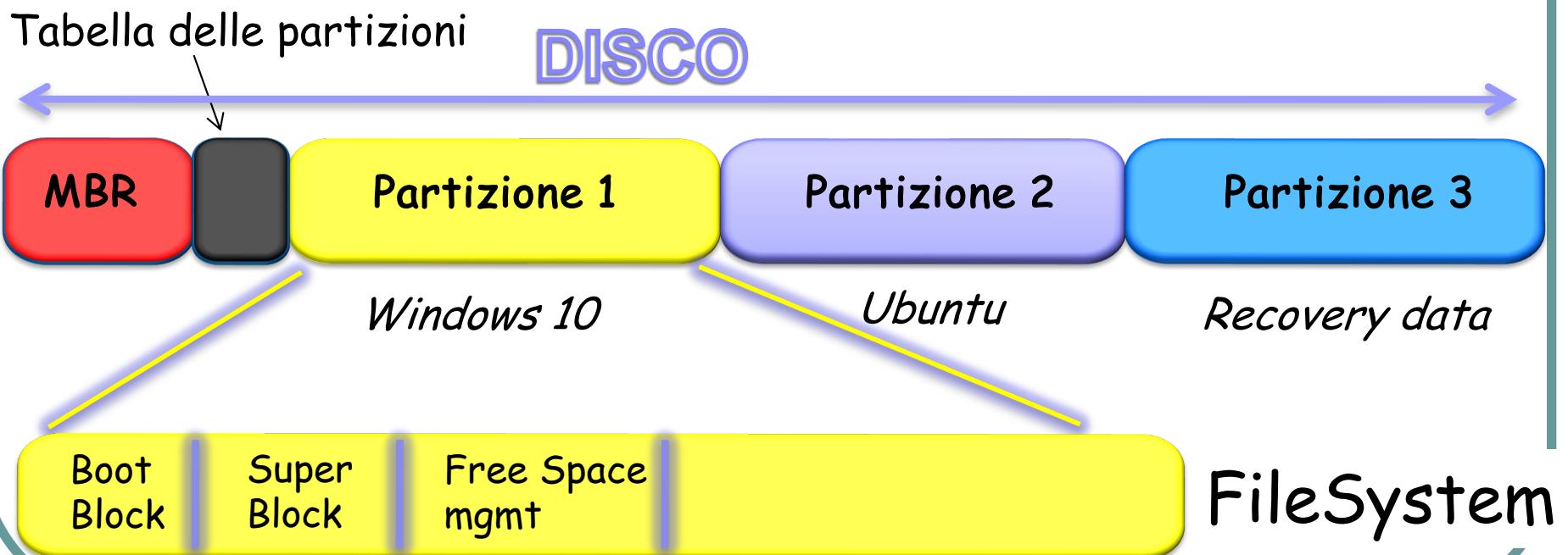
- La **tabella delle partizioni** è un'area che contiene la posizione iniziale e finale su disco delle partizioni





Il layout del disco

- **MBR (Master Boot Record)** è un settore speciale che contiene un programma di avviamento (**bootloader**)
 - Il BIOS preleva ed avvia il bootloader
 - Il bootloader carica il kernel da una partizione e lo avvia





Il termine "bootstrap"

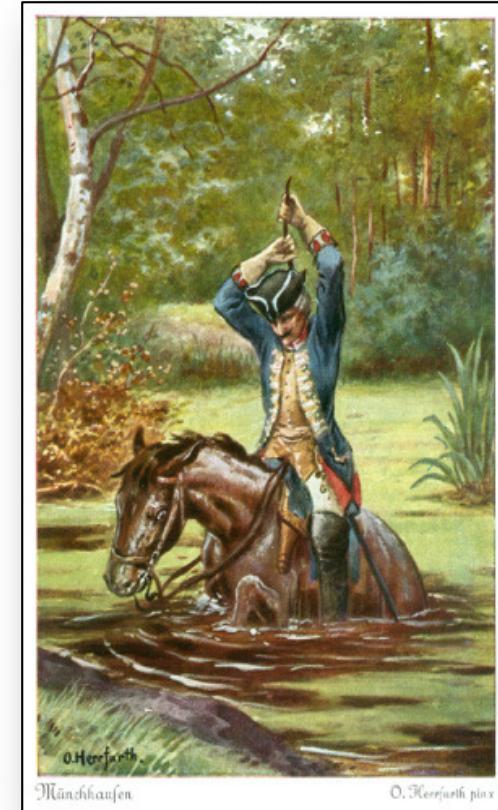
- Il termine bootstrap si riferisce in generale a **una attività che si auto-avvia** senza aiuto esterno



Il termine "bootstrap"



- L'origine del termine **bootstrap** è attribuito al libro "*Le avventure del barone di Münchhausen*"

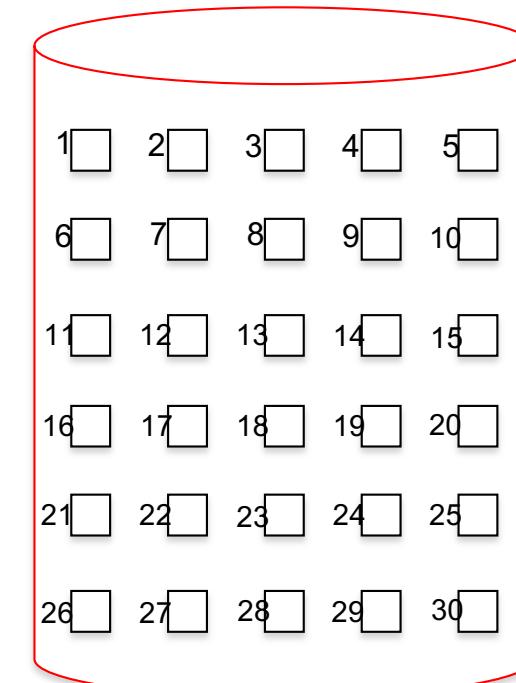


*Il barone si solleva fuori
da una palude tirandosi
per i suoi stessi capelli!*



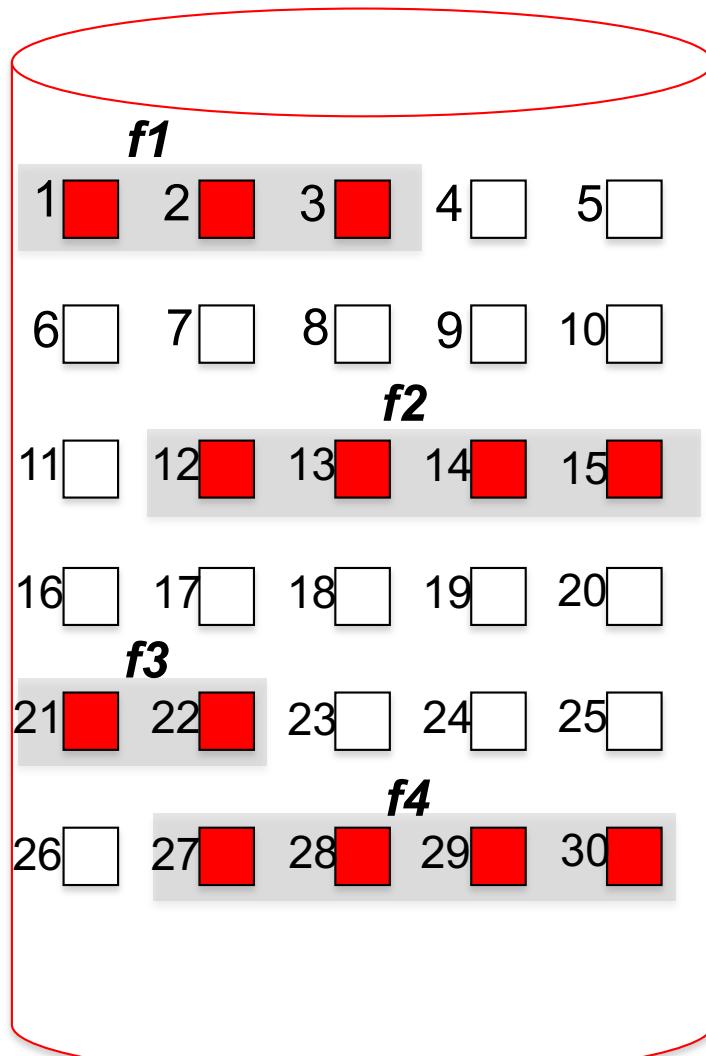
Metodi di allocazione dei file

- I **metodi di allocazione** definiscono il modo in cui i blocchi sono **assegnati ai file**
 - Allocazione Contigua
 - Allocazione Concatenata
 - Allocazione Indicizzata





Allocazione contigua



directory

File	Start	length
f1	1	3
f2	12	4
f3	21	2
f5	27	4

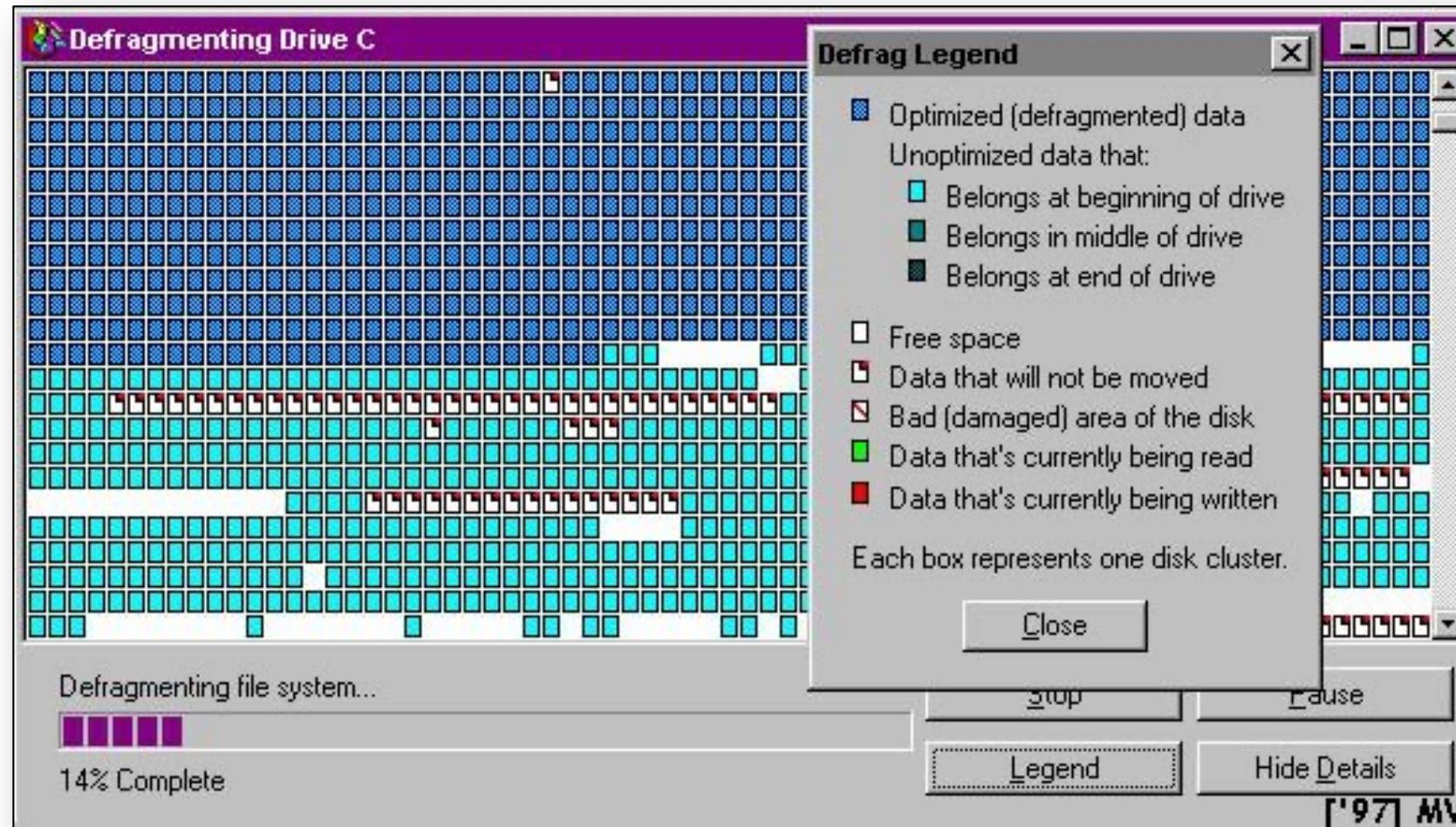


Allocazione contigua

- Ciascun file occupa un **insieme di blocchi contigui sul disco**
- **Vantaggi**
 - Favorisce sia l'accesso **sequenziale** sia **diretto**
 - Per accedere al file, è **sufficiente conoscere** solo la posizione di **inizio** (block #) e la **lunghezza** (numero di blocchi)
- **Svantaggi**
 - **Frammentazione esterna** (come nella allocazione della memoria) → Problema della compattazione del disco
 - I file non possono **crescere** facilmente



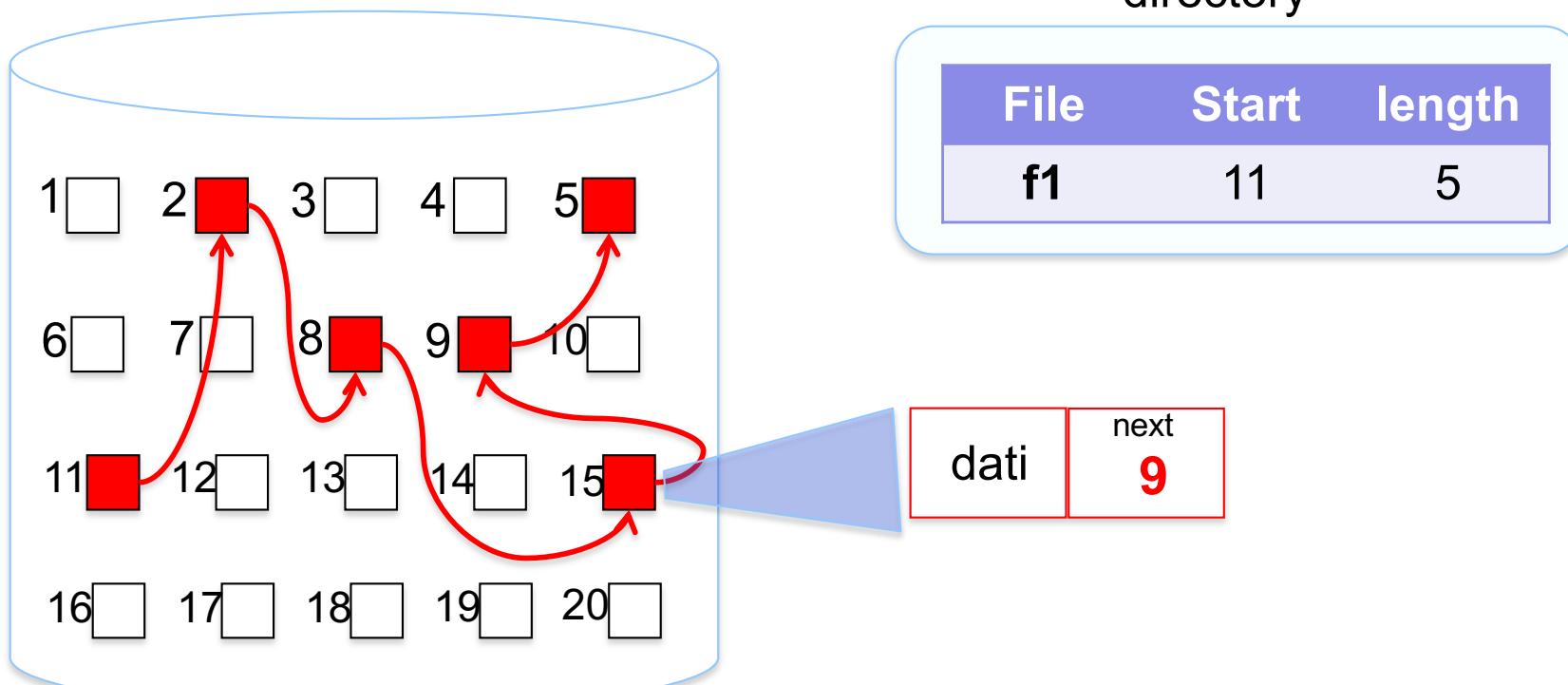
De-frammentazione del disco





Allocazione concatenata (linked)

- Ciascun file è una **lista concatenata di blocchi** su disco
- I blocchi possono essere disposti ovunque





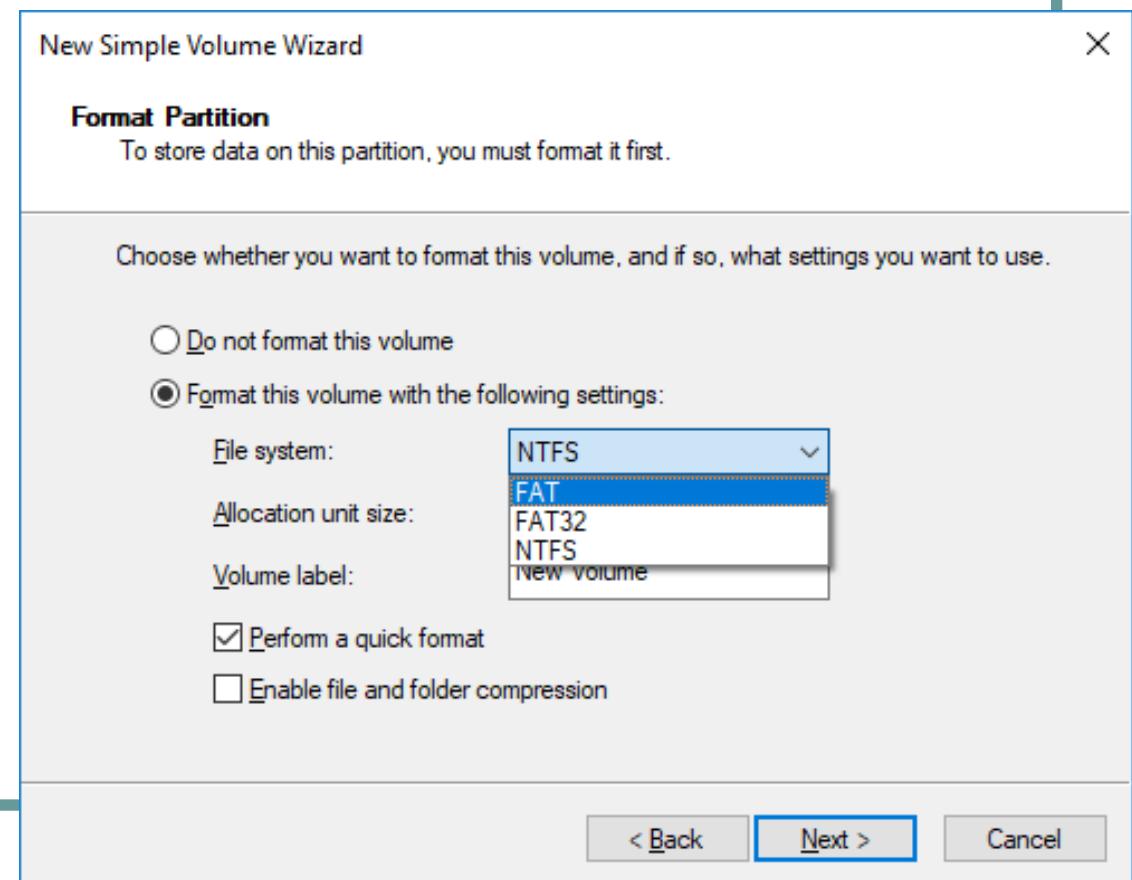
Allocazione concatenata

- I blocchi del file sono organizzati in una lista concatenata
- **Vantaggi**
 - ✓ non c'è frammentazione esterna
 - ✓ maggiore facilità nella scelta dei blocchi da allocare
 - ✓ maggiore flessibilità (es. crescita dei file)
 - ✓ favorisce l'accesso sequenziale
- **Svantaggi**
 - ✓ possibilità di errore se un link viene danneggiato
 - ✓ maggior **consumo di spazio** (spazio occupato dai puntatori)
 - ✓ sfavorisce l'**accesso diretto (casuale)**



FAT (File Allocation Table)

- Introdotto in MS-DOS, usato in Windows fino a ME
- Oggi è una "lingua franca" per dischi acceduti da più SO
- es. **dischi esterni USB**



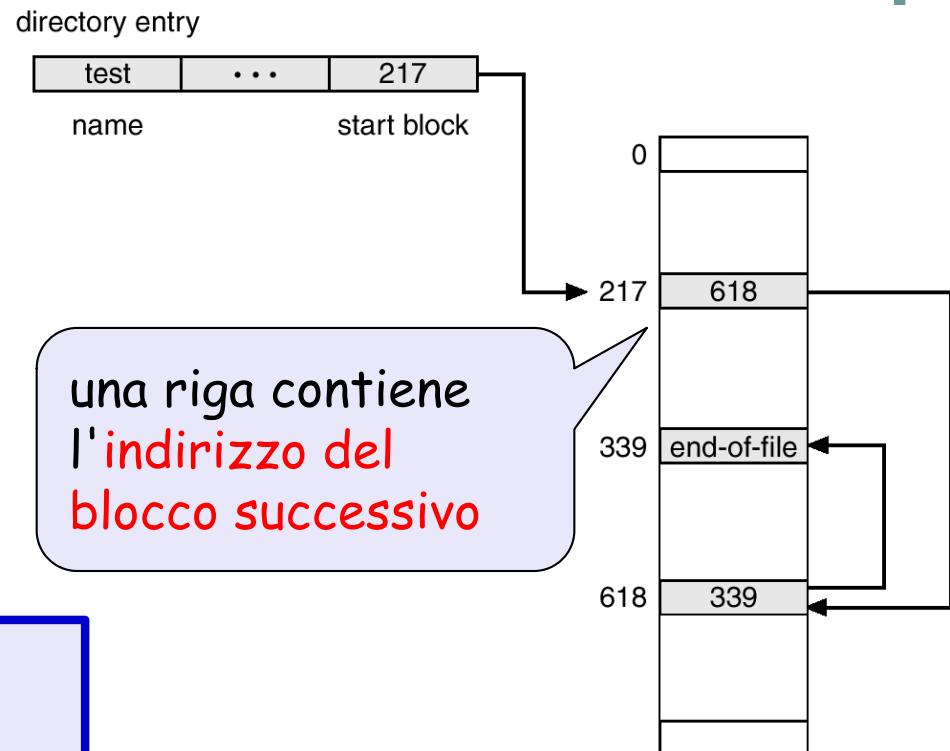


FAT (File Allocation Table)

- Allocazione concatenata
- La FAT table **concentra i puntatori** in una **unica area sul disco**
- Invece che sparsi nei blocchi

Migliore **affidabilità**
(la tabella può essere replicata)

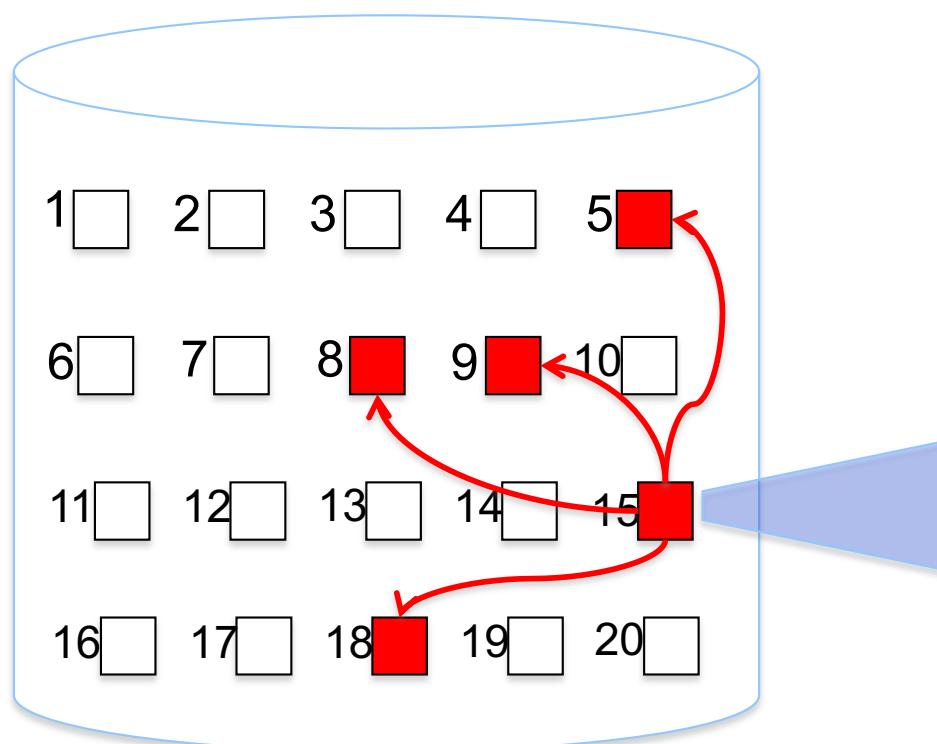
Velocizza l'**accesso diretto**
(evita il seek tra blocchi diversi)





Allocazione indicizzata

- Raggruppa tutti i puntatori in un blocco indice ("index node")



File	Index Block
f1	15

blocks
5
18
9
8
-1
-1
-1
-1

directory



Allocazione indicizzata

- **Vantaggi**

- ✓ non c'è frammentazione esterna
- ✓ maggiore facilità nella scelta dei blocchi da allocare
- ✓ maggiore flessibilità (es. crescita dei file)
- ✓ favorisce l'**accesso diretto (casuale)**

- **Svantaggi**

- ✓ **consumo di spazio** per i blocchi indice
- ✓ problema della **dimensione del blocco indice**

Un **indice piccolo** evita lo spreco all'interno dell'indice, utile per i file piccoli.

Un **indice grande** può indicizzare file di maggiori dimensioni (molti blocchi).

Si utilizzano schemi basati su **blocchi indice concatenati e/o a più livelli**.

inodes



- In UNIX, ogni file ha una struttura di controllo (FCB) denominata **inode (index node)**
- Ogni inode ha un numero univoco (**inode number**)

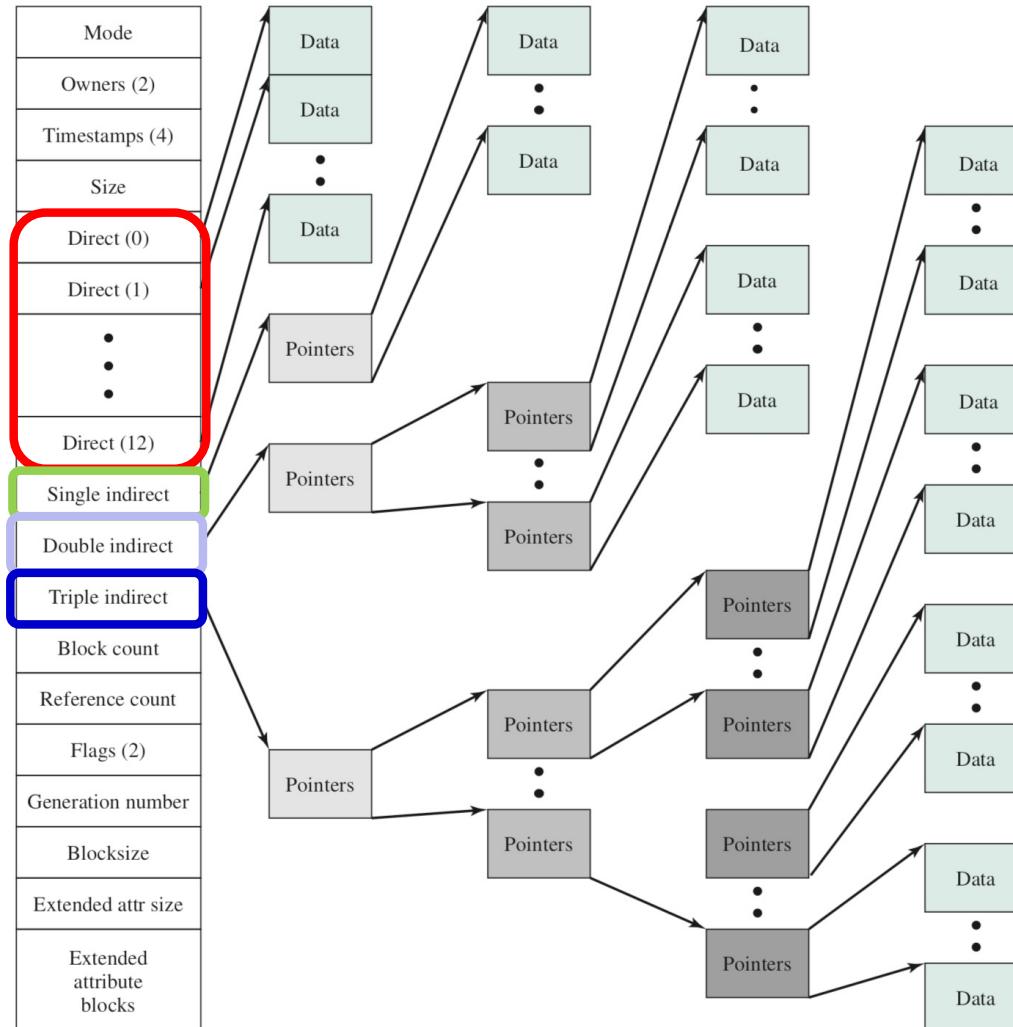
The screenshot shows a terminal window with the following content:

```
so@so-vbox:~/so_esempi$ ln readme.md hard_link.md
so@so-vbox:~/so_esempi$ ls -i *.md
1455388 hard_link.md 1455388 readme.md
so@so-vbox:~/so_esempi$
```

The inode numbers for the two files are highlighted with red boxes: 1455388 for both "hard_link.md" and "readme.md".

Nota: Lo stesso inode può avere più riferimenti (hard link)

inode in UNIX

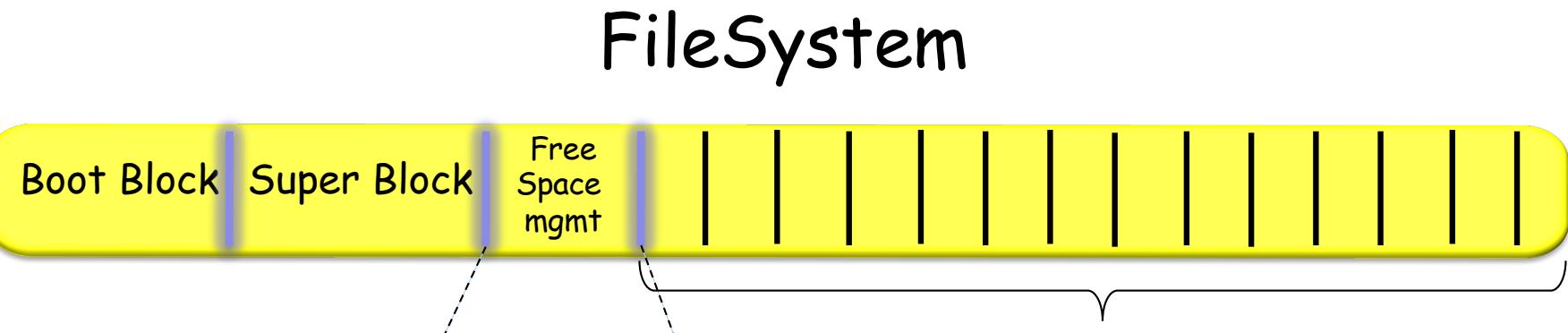


UNIX adotta inodes
con puntatori **sia**
diretti sia indiretti
(multilivello)

Se un file è **piccolo**,
è sufficiente che il
FS utilizzi **solo i**
puntatori diretti



Gestione dello spazio libero



Bitmap

(1 bit per ogni blocco-dati del disco)

$\text{bitmap}[i] = 0 \rightarrow \text{blocco}[i]$ è libero

$\text{bitmap}[i] = 1 \rightarrow \text{blocco}[i]$ è occupato



Gestione dello spazio libero

- La mappa dei bit consuma a sua volta dello **spazio su disco**
- Esempio:

block size = 2^{12} byte (4K)

disk size = 2^{30} byte (1 gigabyte)

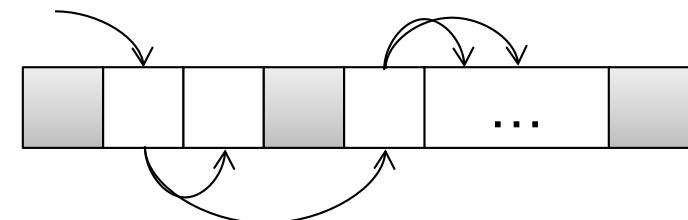
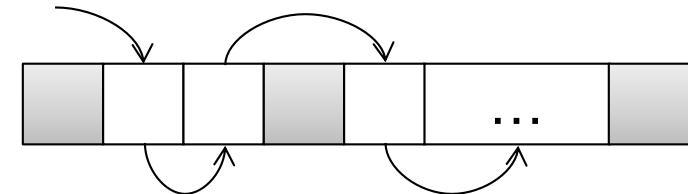
$$\text{n. blocchi} = 2^{30}/2^{12} = 2^{18} \text{ (262.144 blocchi)}$$

$$\begin{aligned}\text{bitmap} &= 2^{18} \text{ bit} \\ &= 2^{18}/8 \text{ byte} \\ &= 32\text{Kb (8 blocchi)}\end{aligned}$$

Altri approcci di gestione dello spazio libero



- **Lista concatenata (*free list*)**: ogni blocco libero contiene un puntatore al successivo blocco libero
- **Raggruppamento (variante della free list)**: ogni blocco libero ha un puntatore ad n blocchi liberi; l'ultimo di essi indirizza altri blocchi, e così via
- **Conteggio**: Per ogni gruppo di blocchi liberi, si memorizza in una lista l'**indirizzo del primo blocco libero**, e il numero di altri blocchi liberi (contigui) da cui è seguito



Inizio	N
2	2
5	1
...	...

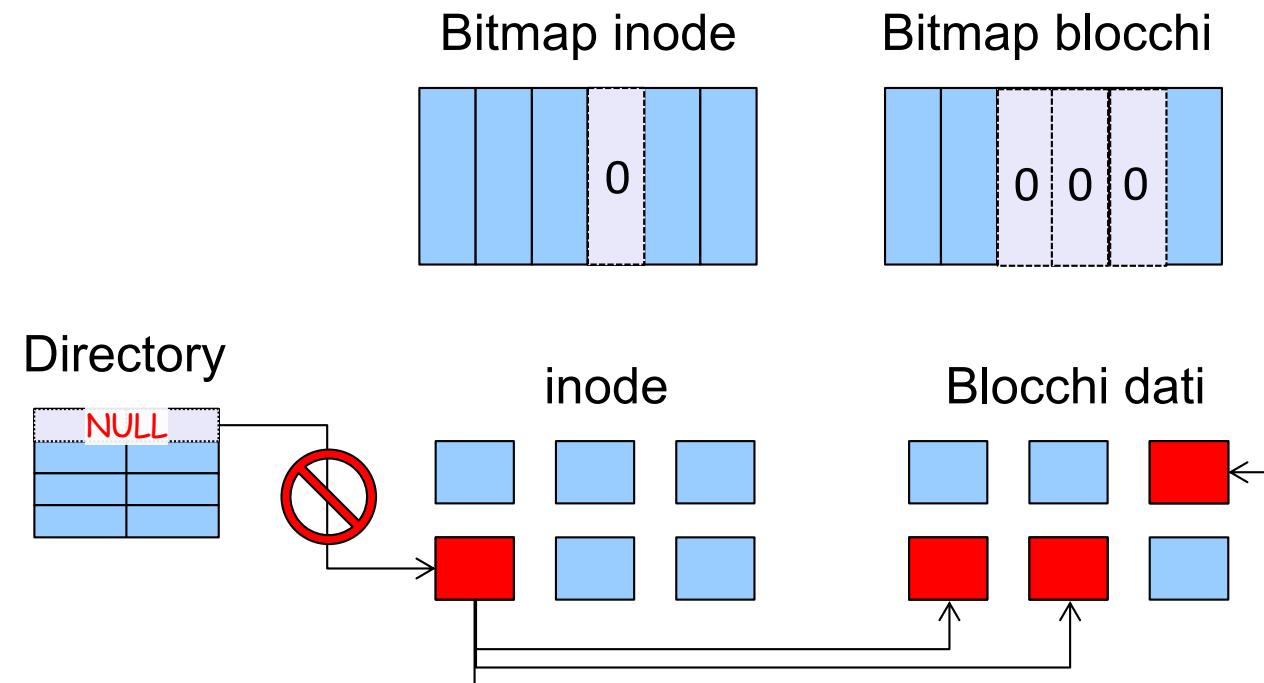


Cancellazione di file

- Per cancellare un file **velocemente**, il FS si limita a **cancellare solo i metadati dei file**
- I blocchi dati sono **marcati come disponibili**
 - contengono ancora i dati del file cancellato
 - ...finché non sono sovrascritti con altri dati



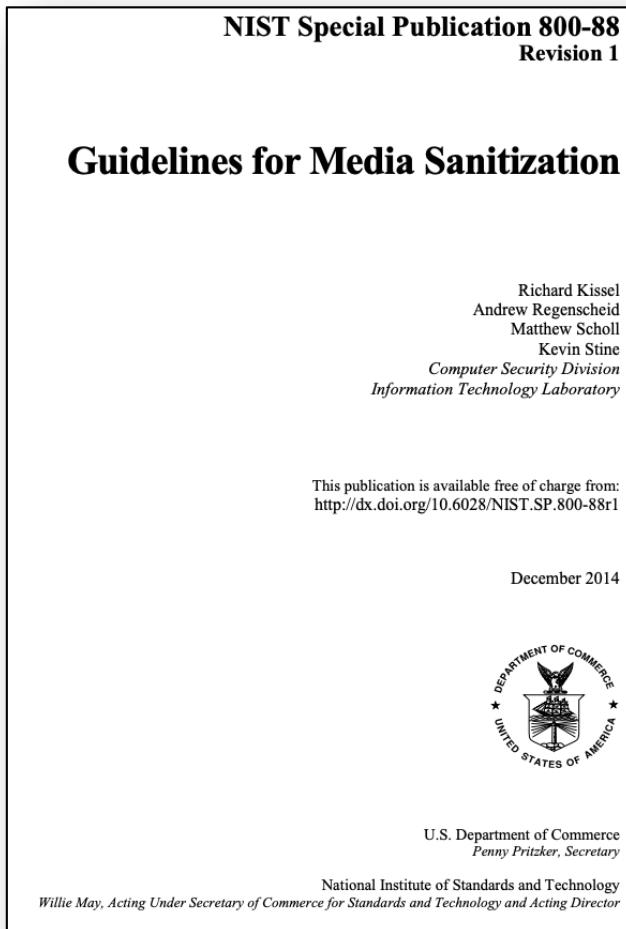
Cancellazione di file



I blocchi dati non sono più raggiungibili, ma hanno ancora il vecchio contenuto (finchè non sono sovrascritti da altri file)



Cancellazione sicura di file



NIST – Guidelines for Media Sanitization

<https://www.nist.gov/news-events/news/2006/08/how-guide-removing-data-storage-media>



Il file system ext2

- Il *second extended filesystem (ext2)* supporta tutte le caratteristiche di un FS standard Unix
- Presenta diverse estensioni rispetto al FS Unix:
 - Dimensione dei blocchi configurabile in fase di installazione
 - Attributi dei file ereditabili dalle directory
 - Link simbolici veloci (memorizzati nell'inode, non in un blocco dati)
 - Supporto a file immutabili (*read only*) e *append-only*
 - Gestione dei nomi di file lunghi (256 caratteri, estensibili a 1012) con una dimensione massima dei file di 4 Tb



Il file system ext2

- Una **partizione ext2** divide il disco in **gruppi di blocchi**
- Un gruppo di blocchi include (approssimativamente) blocchi di una **stessa traccia o tracce vicine** del disco, per favorire la **località degli accessi**





Il file system ext2

I **blocchi-dati** e lo **inode** di uno stesso file sono allocati preferibilmente nello **stesso gruppo**

Gli **inode** sono preferibilmente allocati nello stesso gruppo della **directory** che contiene il file

Le **directory** sono **distribuite uniformemente** tra i gruppi

Block
group 0

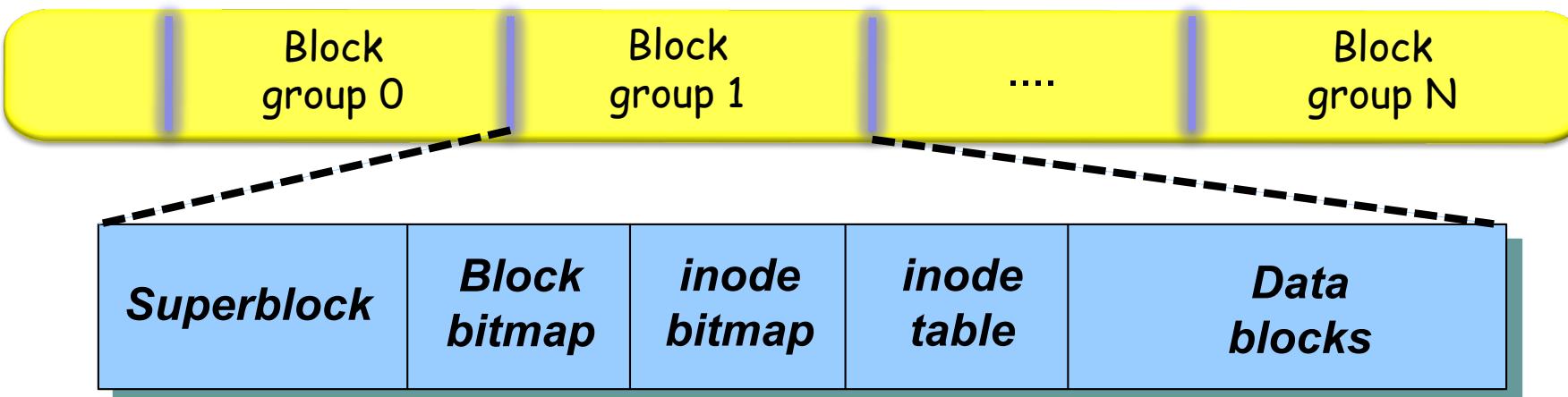
Block
group 1

....

Block
group N



Il file system ext2

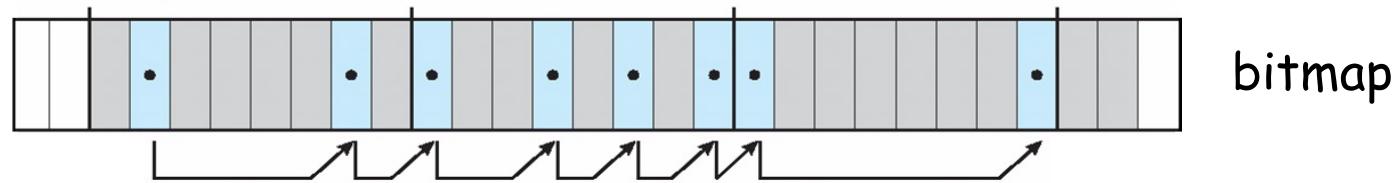


- Ogni gruppo di blocchi (**block group**) include:
 - **superblock**: contiene **informazioni sulla partizione** (ripetute in ogni gruppo), es. numero totale di blocchi e di inode, dimensione dei block group, etc.
 - **inode table**: contiene gli inode dei file nel gruppo
 - **data blocks**: blocchi che memorizzano i dati
 - **block bitmap**: tabella di bit per la gestione dello spazio dati libero
 - **inode bitmap**: tabella di bit per la gestione dello spazio in inode table

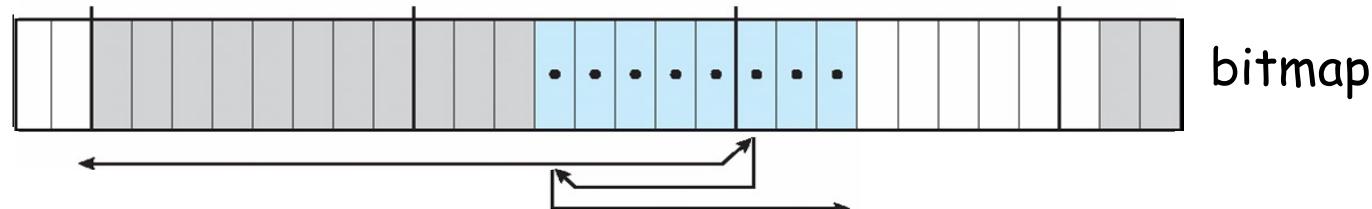


Allocazione dei blocchi in ext2

- La allocazione dello spazio libero ai nuovi file avviene tramite bitmap
 - **Alloca blocchi "vicini" al primo blocco, anche se non-contigui** (sono ancora nella stessa traccia, e accessibili senza fare seek)



- Altrimenti, cerca un gruppo di **blocchi contigui liberi** (un "byte libero" nella bitmap)





Il comando df

- "df" elenca, per tutte le partizioni attive nel SO:
 - il dispositivo fisico (nel caso di FS per I/O sui dischi)
 - il loro **mountpoint** (percorso di accesso)
 - la quantità di spazio disponibile

```
$ df
```

File system	1K-blocchi	Usati	Disponib.	Uso%	Montato su
udev	989580	0	989580	0%	/dev
tmpfs	203552	1400	202152	1%	/run
/dev/sda1	523248	4	523244	1%	/boot/efi
/dev/sda5	40503552	9294864	29121520	25%	/
tmpfs	1017752	0	1017752	0%	/dev/shm
tmpfs	5120	4	5116	1%	/run/lock
tmpfs	1017752	0	1017752	0%	/sys/fs/cgroup
tmpfs	203548	20	203528	1%	/run/user/1000
...					



Il comando df

- "df" elenca, per tutte le partizioni attive nel SO:
 - il dispositivo fisico (nel caso di FS per I/O sui dischi)
 - il loro **mountpoint** (percorso di accesso)
 - la quantità di spazio disponibile

```
$ df
```

File system	1K-blocchi	Usati	Disponib.	Uso%	Montato su
udev	989580	0	989580	0%	/dev
tmpfs	203552	1400	202152	1%	/run
/dev/sda1	523248	4	523244	1%	/boot/efi
/dev/sda5	40503552	9294864	29121520	25%	/
tmpfs	1017752	0	1017752	0%	/dev/shm
tmpfs	5120	4	5116	1%	/run/lock
tmpf	1017752	0	1017752	0%	/sys/fs/cgroup
tmp	203548	20	203528	1%	/run/user/1000
:					

Mountpoint
(partizione di "root")

File "speciale", rappresenta una **partizione del disco**.
I tool gestiscono la partizione accedendo a questo file

Nota: alcune partizioni sono **fittizie**, non sono davvero sul disco ma in RAM (es. tmpfs)



Il comando df

- "**df -i**" mostra informazioni sull'utilizzo degli inode del filesystem
 - In ext2 e successori, è una quantità massima fissata
 - Anche se lo spazio su disco non è esaurito, è possibile che si riempa la tabella degli inode!

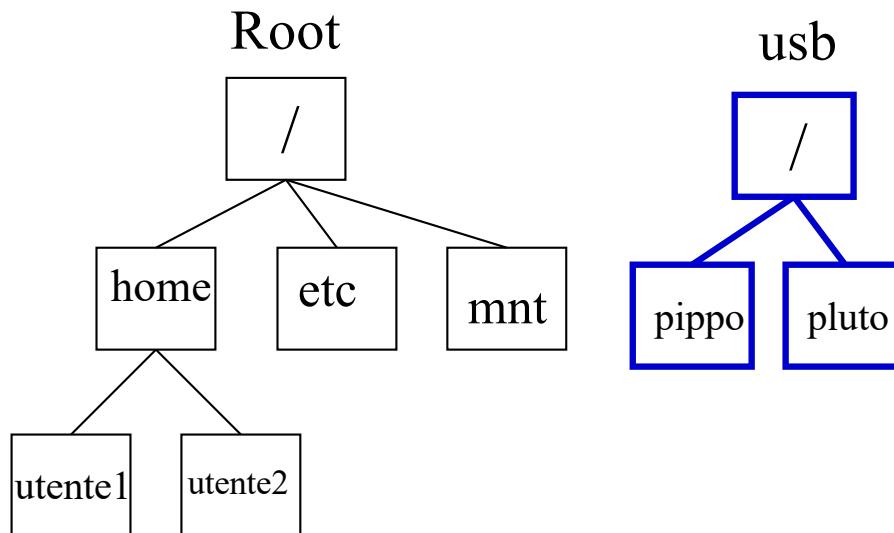
```
$ df -i
```

File system	Inode	IUsati	ILiberi	IUso%	Montato su
udev	247395	491	246904	1%	/dev
tmpfs	254438	870	253568	1%	/run
/dev/sda1	0	0	0	-	/boot/efi
/dev/sda5	2588672	185190	2403482	8%	/
tmpfs	254438	1	254437	1%	/dev/shm
tmpfs	254438	5	254433	1%	/run/lock
tmpfs	254438	18	254420	1%	/sys/fs/cgroup
tmpfs	254438	86	254352	1%	/run/user/1000
...					

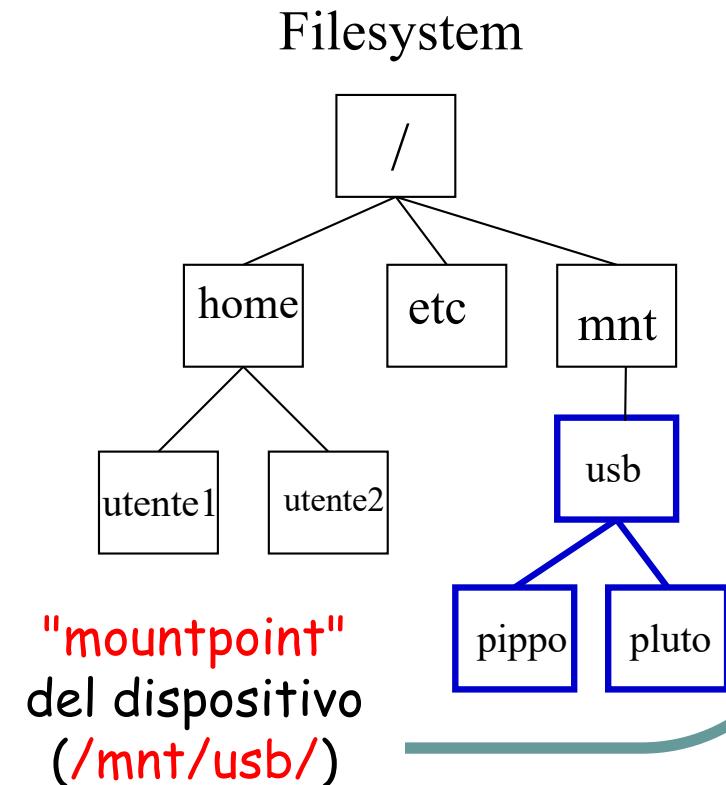


File system “montabile”

- In UNIX/Linux, i FS di tutti i dispositivi sono collegati (“**mount**”) ad un **unico albero** di file e directory



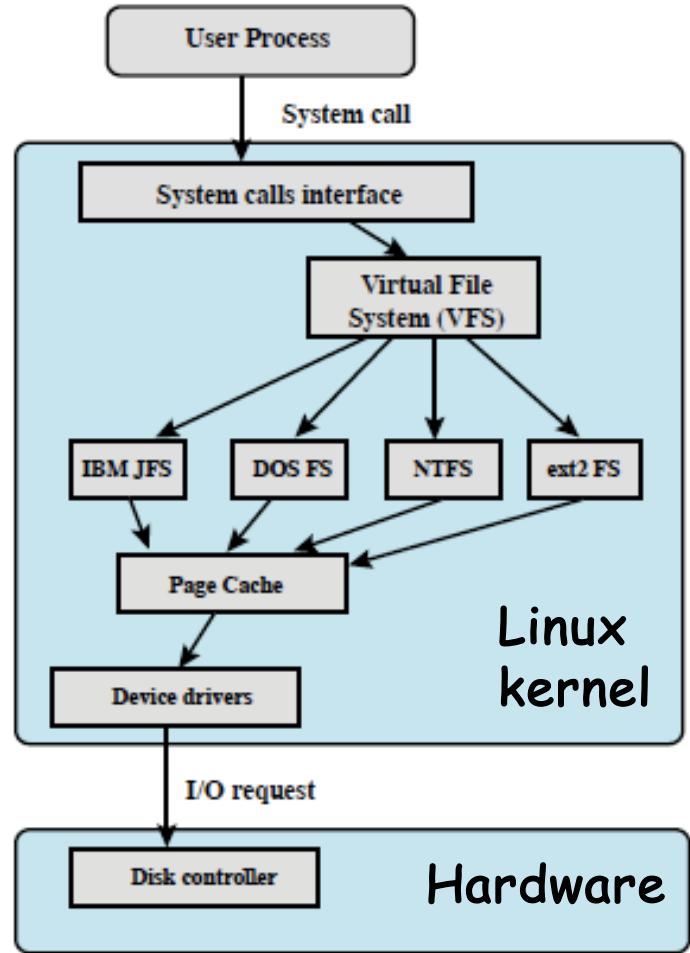
I singoli FS su **due dispositivi** (es., root su **disco interno**, e un secondo FS su **disco esterno**)



“mountpoint”
del dispositivo
(/mnt/usb/)



Il Virtual File System (VFS) di Linux



- **Virtual File System (VFS)** in Linux riceve le chiamate di sistema (**read**, **write**, ...)
- Le "instrada" verso il FS dove si trova il file
- In base al percorso del file, trova il **mountpoint** e il filesystem che lo gestisce (ext2, FAT, etc.)



Il comando mount

- Comando "**mount- Ubuntu Linux (e molte altre distribuzioni) utilizzano "**ext4**", un successore di ext2**

```
$ mount
```

```
· sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,noexec,relatime,size=989580k,nr_inodes=247395,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=203552k,mode=755)
· /dev/sda1 on /boot/efi type vfat (rw,relatime,fmask=0077,dmask=0077,codepage=437, ...
/dev/sda5 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
...
```



Comando cfdisk

Avvertenza! Non effettuare operazioni di modifica sul disco di "root" della macchina virtuale mentre in esecuzione!



- Comando "**sudo cfdisk**

The screenshot shows the terminal window of a Linux virtual machine (so@so-vbox) displaying the output of the **sudo cfdisk** command. The terminal title is "so@so-vbox: ~". The output shows the following information:

Disk: /dev/sda
Size: 40 GiB, 42949672960 bytes, 83886080 sectors
Label: dos, identifier: 0x454f0f2c

Dispositivo	Avvio	Start	Fine	Settori	Size	Id	Tipo
>> /dev/sda1	*	2048	1050623	1048576	512M	b	W95 FAT32
/dev/sda2		1052670	83884031	82831362	39,5G	5	Esteso
└/dev/sda5		1052672	83884031	82831360	39,5G	83	Linux

Partition type: W95 FAT32 (b)
Attributes: 80
Filesystem UUID: 3978-8663
Filesystem: vfat
Mountpoint: /boot/efi (mounted)

[Avviabile] [Elimina] [Resize] [Esci] [Tipo] [Guida] [Scrivi] [Dump]

Quit program without writing changes



Comando cfdisk

Avvertenza! Non effettuare operazioni di modifica sul disco di "root" della macchina virtuale mentre in esecuzione!



- Comando "**sudo cfdisk**

The screenshot shows the output of the cfdisk command for disk /dev/sda. The table lists the partitions:

Dispositivo	Avvio	Start	Fine	Settori	Size	Id	Tipo
>> /dev/sda1	*	2048	1050623	1048576	512M	b	W95 FAT32
/dev/sda2		1052670	83884031	82831362	39,5G	5	Esteso
└/dev/sda5		1052672	83884031	82831360	39,5G	83	Linux

- I file (virtuali) "**/dev/sda***" rappresentano le **partizioni** create dal sistema operativo Ubuntu
- Il prefisso "**sd**" indica un disco di tipo SATA
- "**sda5**" è una sotto-partizione



Esempio di un nuovo FS su Linux

- Aggiunta di un nuovo disco (virtuale, tramite VMware o VirtualBox)
- Modifica della tabella delle partizioni sul nuovo disco
- Formattazione della partizione
- Mount della partizione

Esempio di un nuovo FS su Linux



The image shows two windows from the Oracle VM VirtualBox application:

- Oracle VM VirtualBox Gestore**: The main window showing a list of virtual machines. One machine, "Ubuntu-Linux-2...", is listed as "Spenta" (Powered Off). The toolbar includes buttons for Strumenti, Crea, Elimina, Ripristina, Proprietà, Clona, Impostazioni, Scarta, and Avvia.
- Ubuntu-Linux-20-04-SO - Archiviazione**: A detailed configuration dialog for the selected virtual machine. It shows the storage settings for the "Controller: SATA" device. The "Dispositivi di archiviazione" pane lists "Controller: IDE" (empty) and "Controller: SATA" (containing "Ubuntu-Linux-20-04-SO.vmdk"). The "Attributi" pane shows the configuration:
 - Nome: SATA
 - Tipo: AHCI
 - Numero porte: 1
 - Usa cache I/O dell'hostA yellow tooltip "Aggiunge disco fisso." points to the "+" button in the "Controller: SATA" list.



Esempio di un nuovo FS su Linux

- Il secondo disco è connesso allo stesso controller SATA, ed è rappresentato dal kernel con "**sdb**"
- Per utilizzare il disco, occorre **creare prima una partizione** (es. tramite "cfdisk")

```
$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda5  /dev/sdb

$ sudo cfdisk /dev/sdb
```

Nota: per capire quale sia il **nuovo disco** e quale sia il **disco di root**, possiamo usare il comando "**mount**"



Esempio di un nuovo FS su Linux

- Con cfdisk, **aggiungiamo una nuova partizione**, che occupi per intero il disco che abbiamo appena creato ("**sdb1**")
- Selezionare "**Scrivi**" e poi digitare "**yes**" per salvare le modifiche alla tabella delle partizioni

```
so@so-vbox: ~
Disk: /dev/sdb
Size: 2 GiB, 2147483648 bytes, 4194304 sectors
Label: gpt, identifier: C1AD43AF-5A28-0A42-A213-4B9DD9F77D2B

Dispositivo      Start      Fine      Settori      Size  Tipo
>>  /dev/sdb1        2048    4194270    4192223     2G  Linux filesystem

Partition UUID: DB83C7A2-78D6-F342-A18C-0D83500F0E51
Partition type: Linux filesystem (0FC63DAF-8483-4772-8E79-3D69D8477DE4)

[ Elimina ] [ Resize ] [ Esci ] [ Tipo ] [ Guida ] [ Scrivi ] [ Dump ]
Scrivi la tabella delle partizioni sul disco (i dati potrebbero venir persi)
```



Esempio di un nuovo FS su Linux

- Usare il comando "mkfs" (es. **mkfs.ext4**) per **formattare** la nuova partizione usando ext4

```
$ sudo /sbin/mkfs.ext4 -L PROVA /dev/sdb1
```

```
mke2fs 1.45.5 (07-Jan-2020)
Creazione del file system con 524027 4k blocchi e 131072 inode
Etichetta del file system=06c3c41f-172e-447c-9356-ecdd6bf789a6
Backup del superblocco salvati nei blocchi:
    32768, 98304, 163840, 229376, 294912
```

```
Allocating group tables: fatto
Scrittura delle tavole degli inode: fatto
Creating journal (8192 blocks): fatto
Scrittura delle informazioni dei super-blocchi e dell'accounting del file system: fatto
```



Esempio di un nuovo FS su Linux

- Creare una nuova cartella (es. `/home/so/mydisk`) che sarà il "**mountpoint**"
- Usare il comando "**mount**" per connettere il nuovo filesystem al mountpoint

```
$ mkdir mydisk

$ sudo mount /dev/sdb1 ./mydisk/

$ mount | grep sdb
/dev/sdb1 on /home/so/mydisk type ext4 (rw,relatime)

$ df

File system      1K-blocchi    Usati Disponib. Uso% Montato su
/dev/sda1          523248        4    523244    1% /boot/efi
/dev/sda5        40503552  9294860  29121524   25% /
/dev/sdb1          2030396     6144   1903064    1% /home/so/mydisk
```



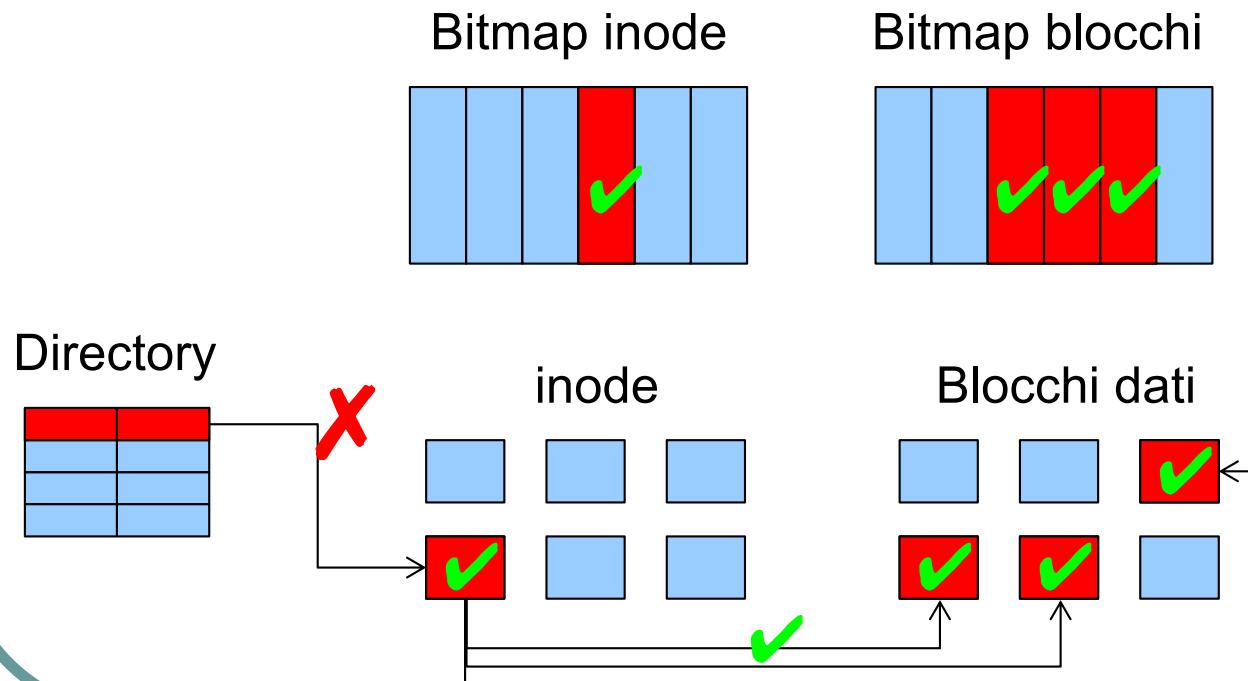
Il file system ext3

- Introdotto nel 2001 nel kernel Linux 2.4.15
- Rispetto al suo predecessore, introduce
 - il journaling
 - ridimensionamento del file system "a caldo"
 - per le directory molto grandi, adotta un'indicizzazione basata su strutture ad albero



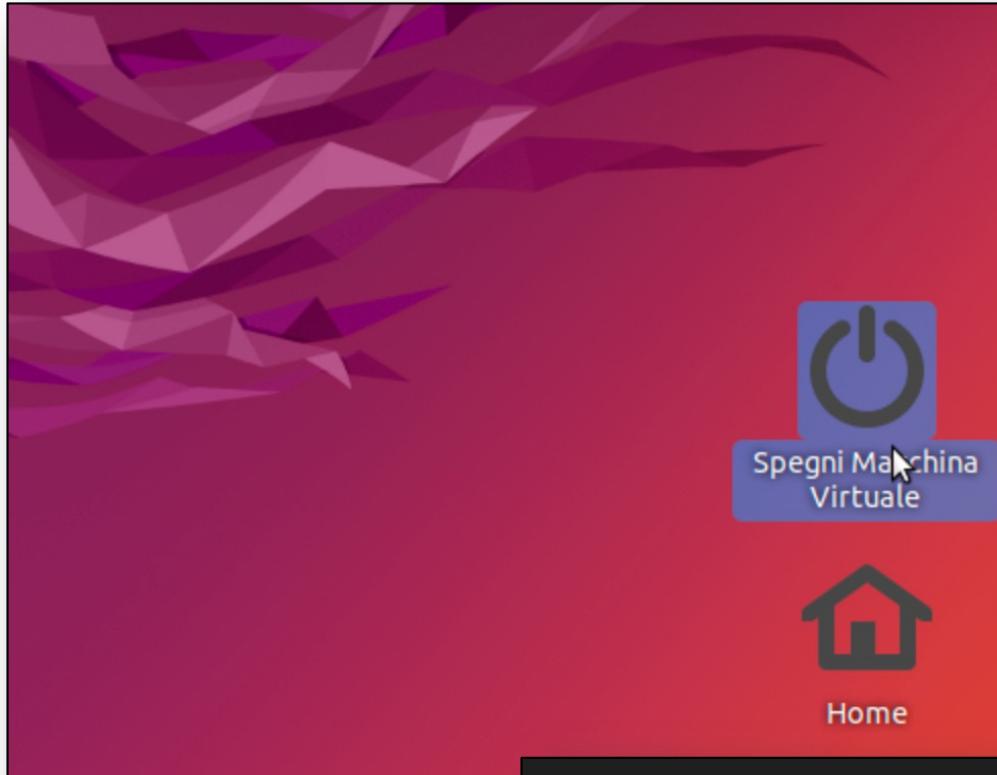
In caso di malfunzionamenti...

- I **malfunzionamenti** (sia hardware sia software) possono interrompere le operazioni sul FS
- Si creano **inconsistenze** tra le varie strutture dati





In caso di malfunzionamenti...



È opportuno **spegnere correttamente** il PC per evitare inconsistentez sul disco!

Journaling

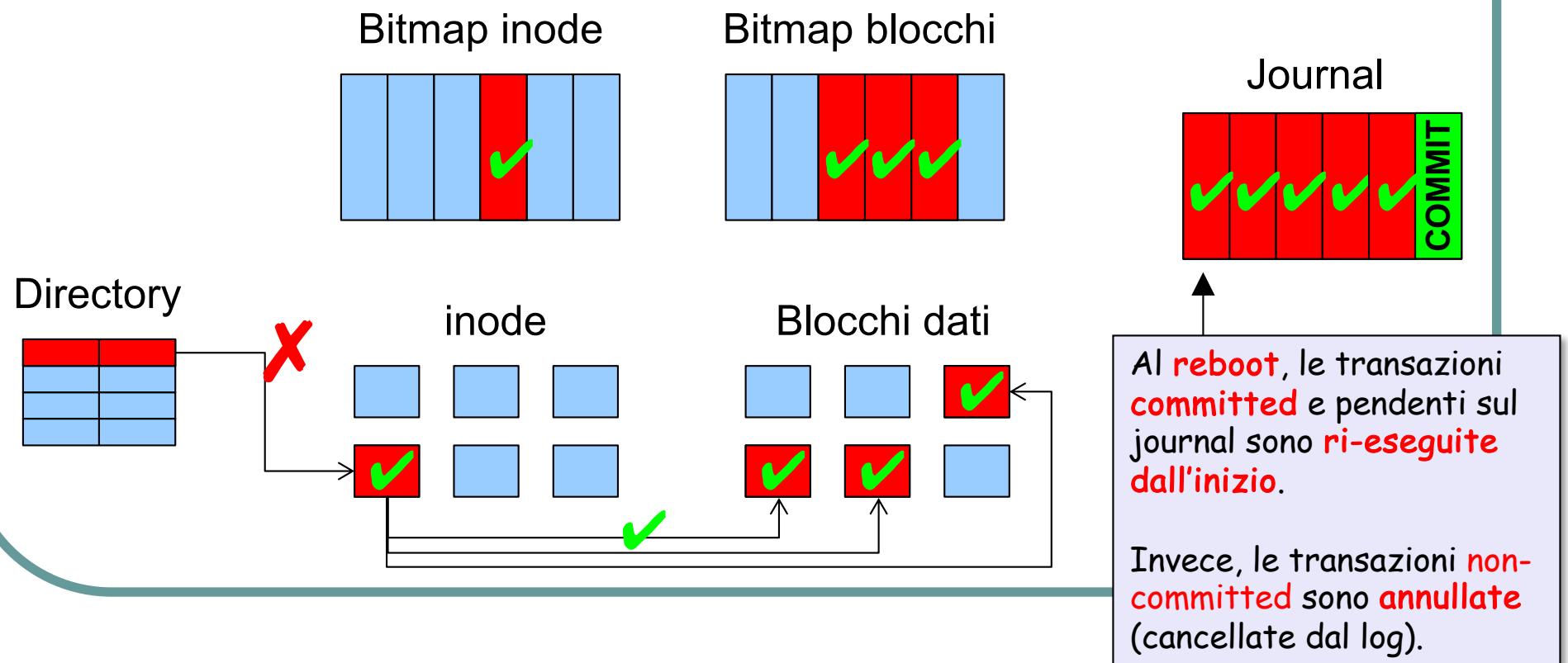


- Il *journaling* consiste nel gestire un "diario" (journal) delle modifiche su disco in maniera transazionale
 - Ogni operazione è fatta in modo "tutto o niente"
 - Evita che un guasto durante una operazione possa lasciare i file in uno **stato inconsistente**
 - Evita di ricorrere a **tool di recupero**, come e2fsck



Journaling

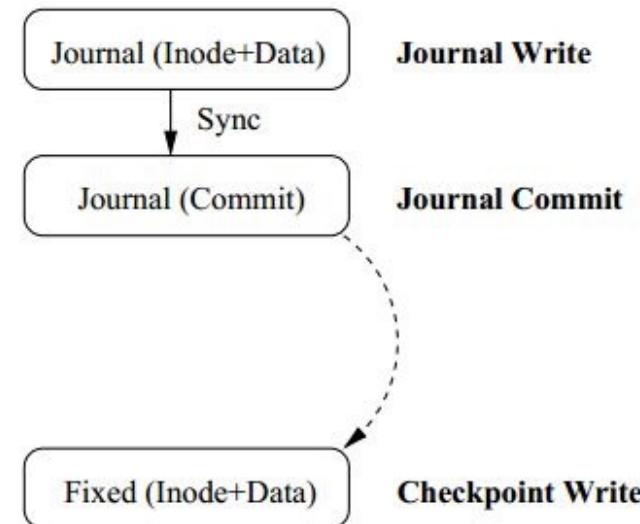
- In un journaling FS, le modifiche ai metadati e ai dati vengono **prima annotate (sequenzialmente) nel journal**
- Se la scrittura su journal ha successo, il journal ha un marcitore (**commit**)
- La scrittura dei dati/metadati nella loro **posizione effettiva su disco (checkpointing)** può avvenire subito o essere posticipata





Modalità di journaling in ext3

DATA

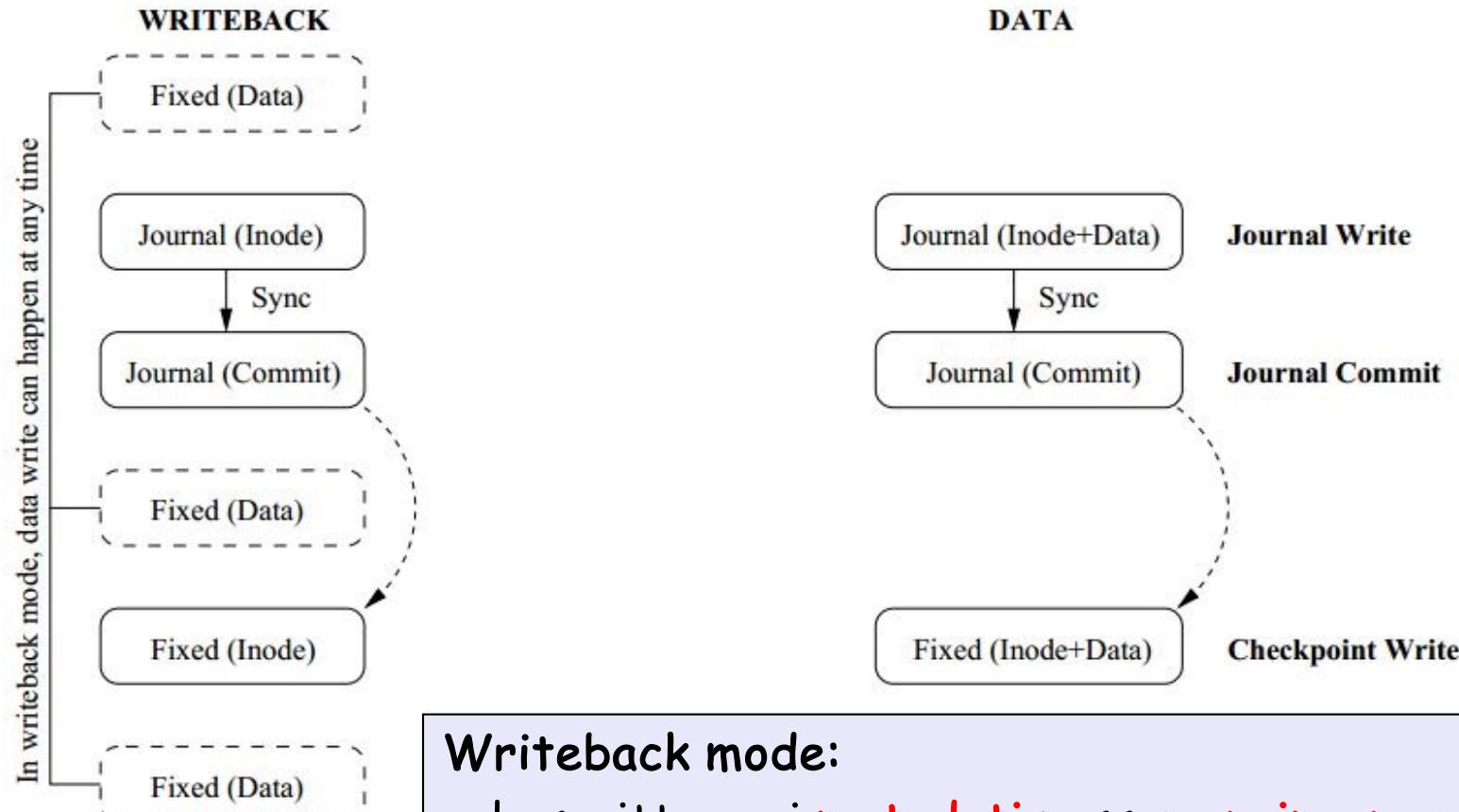


Data mode:

- **sia i dati, sia i metadati**, passano prima per il journal, e poi per alla loro posizione effettiva ("fixed")
- **migliore affidabilità** a scapito delle prestazioni



Modalità di journaling in ext3

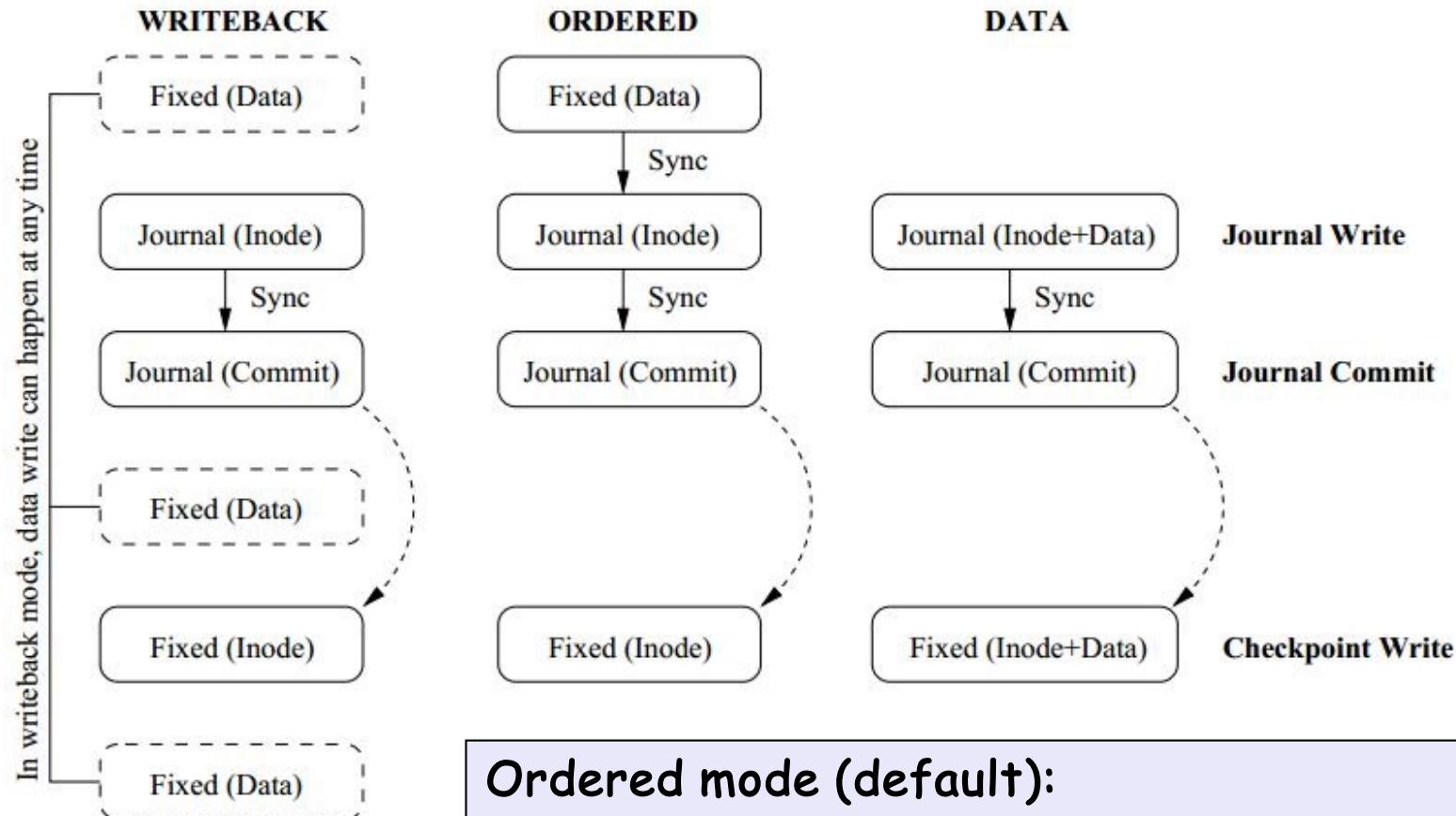


Writeback mode:

- le scritture ai **metadati** passano **prima per il journal**, e poi per la loro posizione effettiva
- le scritture ai **dati** possono avvenire **sia prima, sia dopo** le modifiche ai metadati
- **migliori prestazioni**, scarsa affidabilità



Modalità di journaling in ext3



Ordered mode (default):

- si scrivono **inizialmente i dati** nella loro posizione effettiva, **senza passare** per il journal
- poi, si **salvano i metadati** passando **per il journal**
- compromesso tra affidabilità e prestazioni



Il file system ext4

- Introdotto nel 2006 nel kernel Linux 2.6.19
- Rispetto ad ext3, aggiunge:
 - Il supporto a **volumi grandi**, fino a 16 TB
 - L'utilizzo di gruppi contigui di blocchi, noti come **extent** per memorizzare file di grosse dimensioni (la massima dimensione di un extent è di 128 MB)
 - L'allocazione **multiblocco** (in ext3 la decisione su dove allocare i blocchi di un file era fatta blocco per blocco)
 - L'allocazione **ritardata**, per effettuare scelte di allocazione ottimizzate solo quando si decide effettivamente di salvare il file



Il file system NTFS di Windows

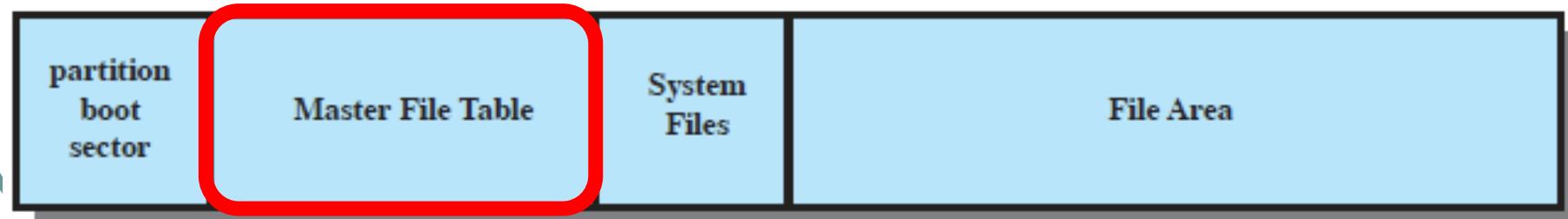
- Introdotto negli anni novanta con Windows NT
- Principali caratteristiche:
 - **Recuperabilità**, attraverso un sistema **transazionale** (basato su "journal") per la gestione delle modifiche, e introducendo ridondanza per le strutture dati critiche
 - **Sicurezza**: ad ogni file è associato un descrittore per specificare attributi di sicurezza e ACL secondo il Windows Object Model
 - Supporto a file e dischi di **grosse dimensioni**
 - Supporto a **nomi di file lunghi**



Master File Table (MFT)

- La **MFT** contiene la lista di tutti i file e le cartelle
- Organizzata come **database relazionale**
- Ogni riga della MFT è di 1024 byte, e contiene
 - Gli attributi del file o cartella, trattati come metadati
 - I dati del file, se il file è abbastanza piccolo da essere contenuto in una singola entry di MFT
 - I puntatori ai cluster che contengono i dati del file o a dei cluster "indice" (**allocazione indicizzata multilivello**)

Layout di un volume NTFS





Unità di storage di NTFS

- **Volume**: una partizione logica del disco
- **Sector**: la più piccola unità di storage (512 byte)
- **Cluster**: uno o più settori contigui; la dimensione effettiva dei cluster varia da 1 a 128 settori, e dipende dalla dimensione complessiva del volume (a volumi grandi corrispondono cluster grandi e viceversa)

Layout di un volume NTFS





System Files

- La regione "System Files" del volume contiene:
 - **MFT-2**: un mirror delle prime tre righe del MFT (utile per recuperare l'accesso alla MFT in caso di guasto di un settore)
 - **Log file**: il journal in cui sono memorizzati i passi delle transazioni, per garantire la recuperabilità
 - **Cluster bitmap**: per la gestione dello spazio libero
 - **Attribute definition table**: definisce gli attributi supportati dal volume (come lo "schema" di un database)

Layout di un volume NTFS





Filesystem per SSD

- Gli SSD hanno un "contratto non scritto" diverso dai dischi magnetici
 1. uniformità dei tempi di accesso in lettura
 2. maggiore velocità e durata di vita delle celle, in caso di scritture sequenziali



Filesystem per SSD

- I filesystem di tipo log-structured (LFS) sfruttano queste caratteristiche
- Nacquero per ottimizzare l'I/O sequenziale e "write-bound" sui dischi, sono oggi usati per SSD

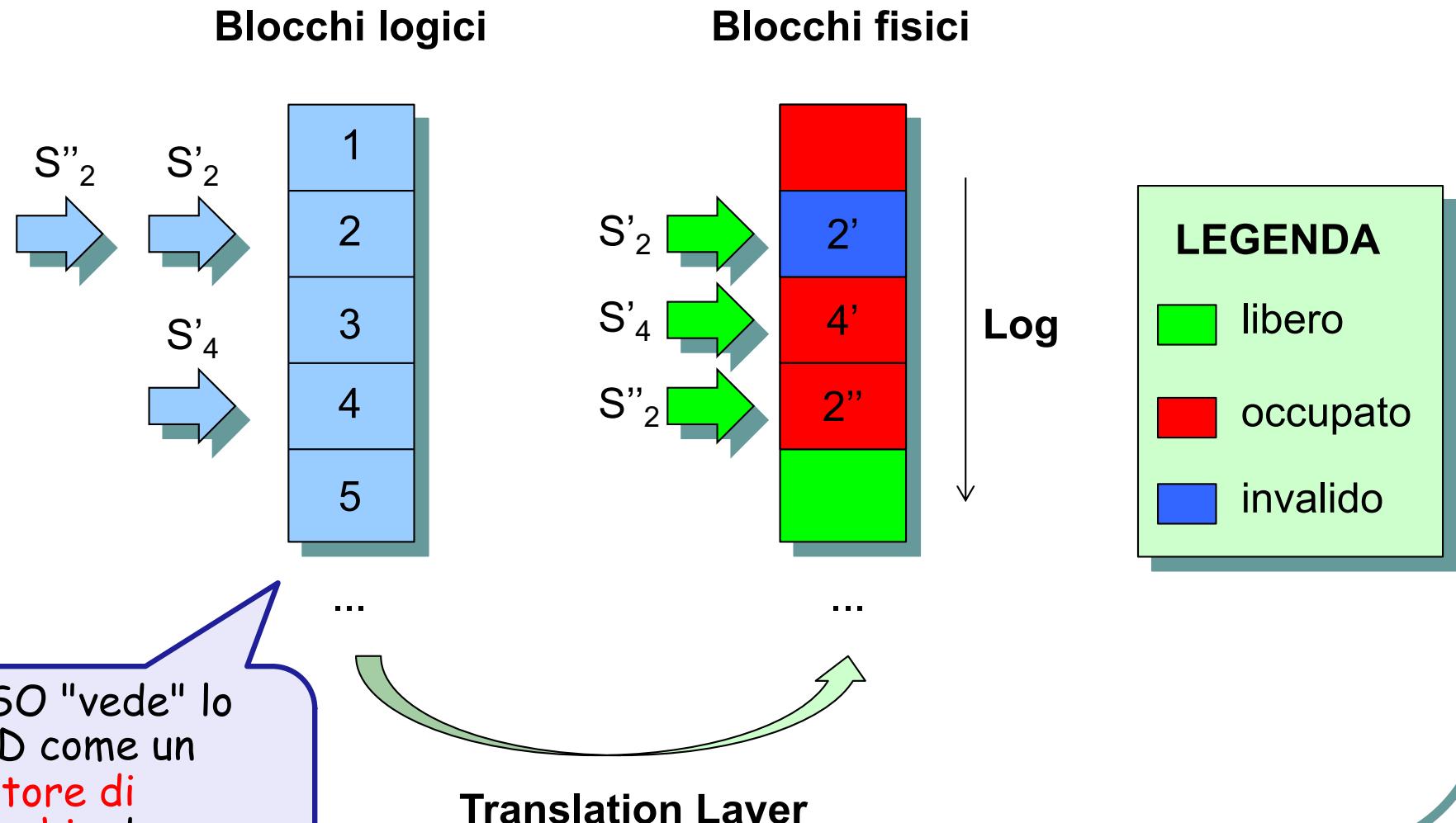


Log-structured File System

- Blocchi fisici gestiti come una **coda circolare** ("log")
- Le scritture avvengono sempre nel blocco fisico
successivo nella coda circolare
- Il log è indipendente dalla posizione "logica" dei dati



Esempio di scrittura in LFS





Gestione dei blocchi in LFS

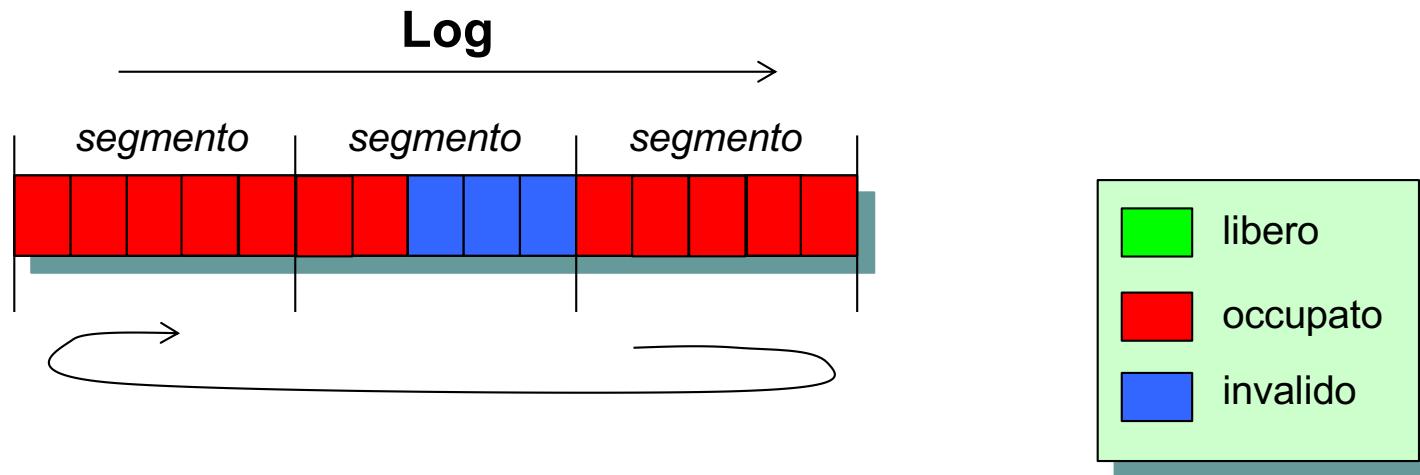
- La **sovrascrittura** di un blocco logico **invalida** il blocco fisico con la versione precedente
- Solo l'ultima versione è "valida"
- I blocchi invalidi possono essere **riutilizzati** per altre scritture



Gestione dei blocchi invalidi

1. Log "filettato"

- Una volta esauriti i blocchi fisici liberi, il FS **riusa i blocchi invalidi** gestendoli ancora come una coda circolare (saltando i blocchi occupati)
- Tende a **frammentare il log**, spezzando la sequenzialità degli accessi

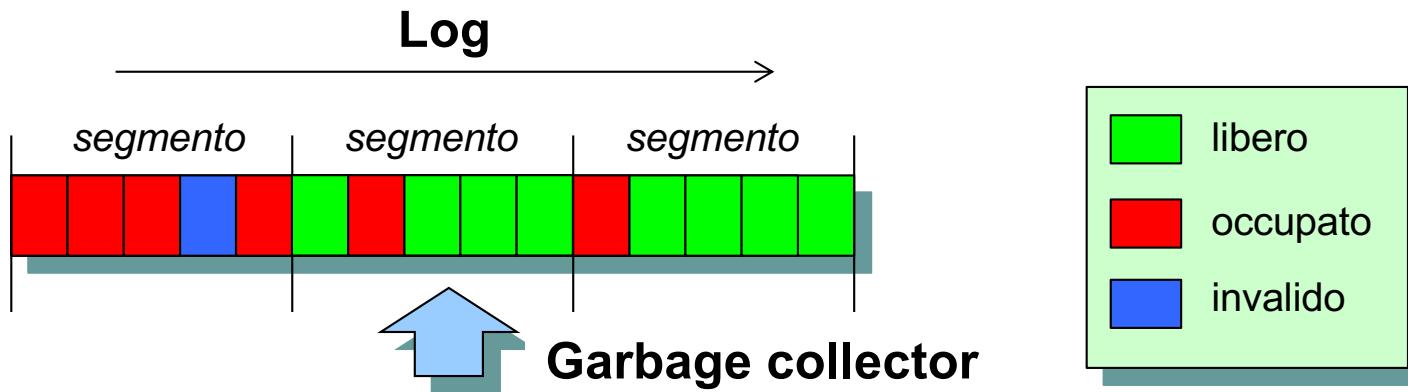




Gestione dei blocchi invalidi

2. Garbage collection

- Un algoritmo esamina periodicamente i blocchi a gruppi (**segmenti**)
- Blocchi validi in un segmento frammentato sono **spostati**
- Si "reclama" i segmenti con soli **blocchi invalidi**, dichiarandoli **liberi**



Favorisce **scritture sequenziali** su gruppi di blocchi liberi **contigui**
Genera attività di "**background**" (con overhead)



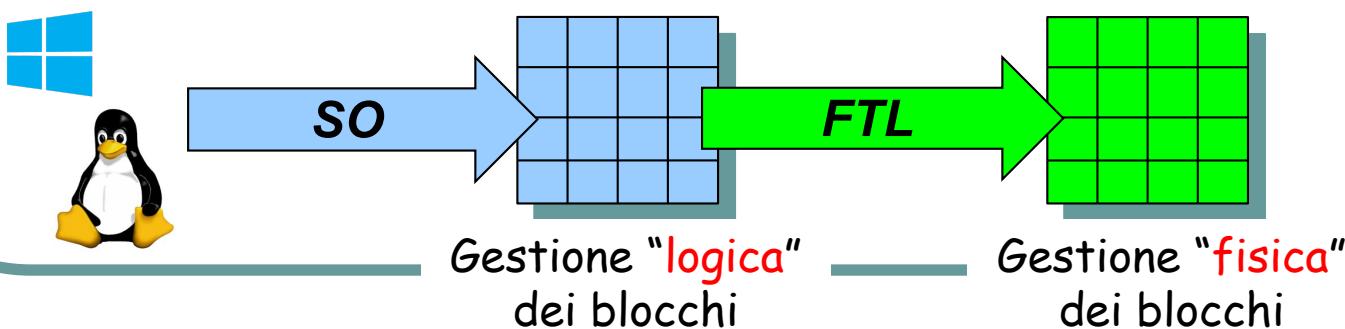
Politiche di garbage collection

- Collection del segmento **in testa** al log (elevato overhead nel caso di molti blocchi validi nel gruppo)
- Collection “**greedy**”, si libera il segmento con il minor numero di blocchi validi
- Collection basata sulla **dinamicità** dei dati (**caldi VS freddi**)
 - I segmenti sono liberati se il numero di blocchi validi è **sotto una soglia**
 - I segmenti con dati **freddi** hanno soglia più **alta** e sono liberati **prima**



Filesystem SSD

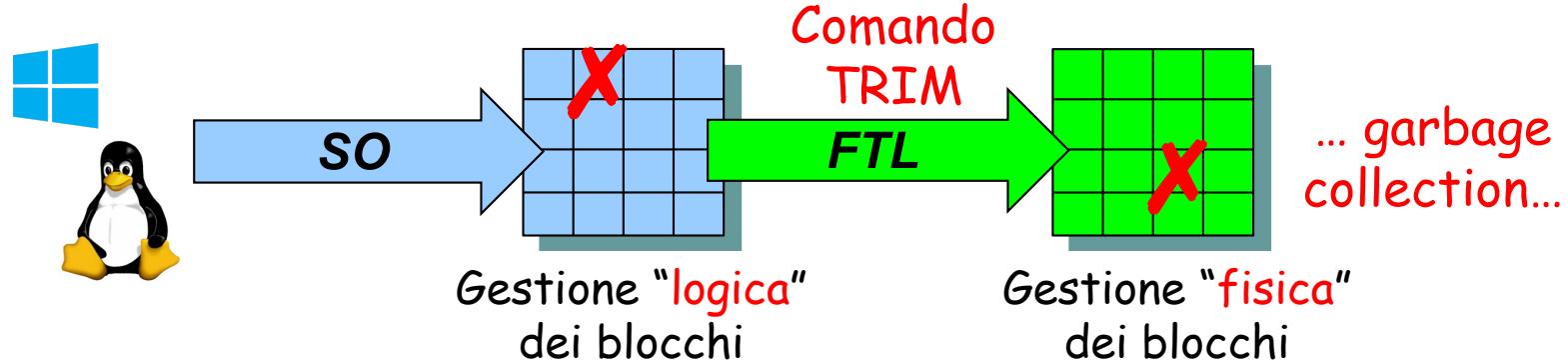
- Sia il **SO** sia il **controller SSD (FTL)** possono adottare un log-structured FS:
 1. Filesystem LFS gestito dal SO (**JFFS, YAFFS**), nessun filesystem nel controller
 2. Filesystem tradizionale gestito dal SO (**NTFS, FAT, ...**)
+ Filesystem LFS gestito dal controller
 3. Filesystem LFS gestito dal SO (**F2FS**) + Filesystem LFS gestito dal controller
 - 1. JFFS, YAFFS, ... (log-structured)
 - 2. NTFS, FAT, EXT4, ... (no log)
 - 3. F2FS (log-structured)
 - 1. No file system (blocchi logici = fisici)
 - 2. File system log-structured interno a FTL
 - 3. File system log-structured interno a FTL



Comando TRIM



- È necessario che un **FS tradizionale su SSD** (es. NTFS) informi la FTL sulla **cancellazione di un blocco logico**
 - Consente la **garbage collection** del blocco fisico
 - Altrimenti, FTL vede il blocco fisico come ancora "occupato"





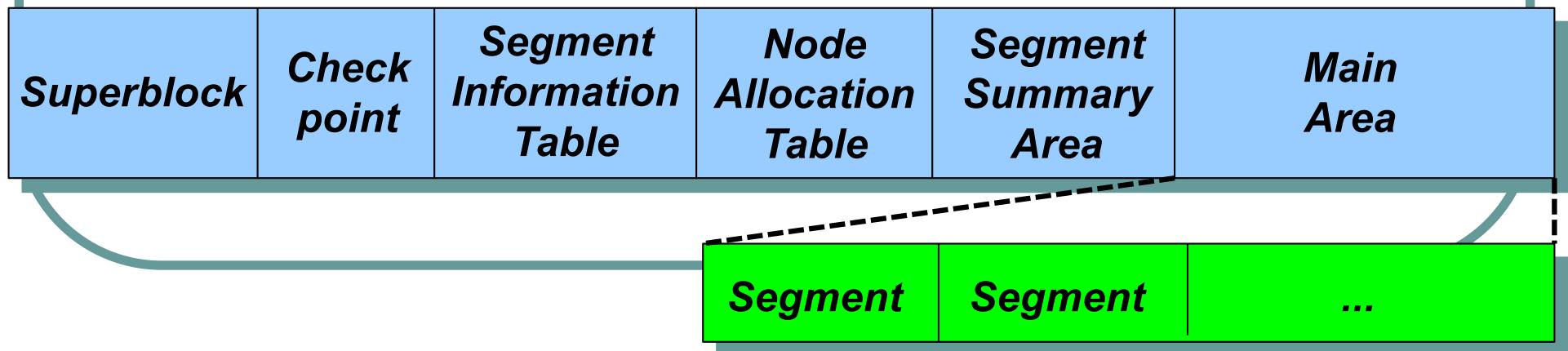
Il filesystem F2FS

- Il **Flash Friendly File System (F2FS)** è un FS sviluppato da Samsung per il SO Linux
 - Usato per SSD in dispositivi **Android** (es., smartphone)
 - Assume che il controller SSD usi un FS log-structured
 - Favorisce ulteriormente la **sequenzialità delle scritture**
 - "Allinea" le strutture dati alle componenti dello SSD
 - Ottimizza la gestione degli indici ai blocchi validi
 - Garbage collection efficiente



Layout di un filesystem F2FS

- **Superblocco**: info su partizione e parametri di F2FS
- **Main Area**: a sua volta divisa in **6 log**, in base alla **"dinamicità"** dei blocchi (per ottimizzare garbage collection)
- **Segment Information Table (SIT), Segment Summary Area**: info sui segmenti, tra cui bitmap e numero totale dei blocchi validi nel segmento
- **Node Allocation Table (NAT)**: indirizzi dei blocchi “*nodi*” (inode, blocchi-puntatore)
- **Checkpoint**: bitmap per SIT e NAT, info sullo stato del FS





Gestione dei log in F2FS

- F2FS utilizza **più log separati**, distinguendo tra **nodi** (ossia inode e puntatori di 1^o e 2^o livello) e **dati**:
 1. **Nodi caldi**: blocchi con nodi diretti verso le cartelle
 2. **Nodi tiepidi**: blocchi con nodi diretti, e non caldi
 3. **Nodi freddi**: blocchi con nodi indiretti
 4. **Dati caldi**: blocchi dati di tipo cartella
 5. **Dati freddi**: blocchi con dati multimediali oppure blocchi che sono stati migrati dall'algoritmo di garbage collection
 6. **Dati tiepidi**: blocchi contenenti dati non classificati né come caldi né come freddi
- La garbage collection è effettuata sia **on-demand** (quando i segmenti liberi sono sotto una soglia, con politica **greedy**) sia **periodicamente** (basato sulla **dinamicità** dei dati)

Quiz



1. Quali delle seguenti affermazioni riguardo la tabella dei file aperti sono vere? (selezionare più di una)

- È condivisa fra tutti i processi
- Viene riempita una nuova riga quando il processo chiama "open()"
- Viene riempita una nuova riga quando il processo chiama "read()"
- Una sua riga contiene il nome di un file aperto
- Una sua riga punta allo FCB (File Control Block) di un file aperto

<https://forms.office.com/r/EyAwYAF9Hr>

