

Programmazione e Architetture (Modulo B)

Lezione 13 Filesystem

Filesystem

E persistenza

- Una delle astrazioni fornite da un OS è quella dei file
- Vedremo come passare da un dispositivo di memoria di massa (disco fisso o memoria flash) a fornire questa astrazione
- Vedremo questo in due passi:
 - Come passare da un disco fisso/memoria flash a un dispositivo a blocchi
 - Come creare l'astrazione dei file a partire da un dispositivo a blocchi

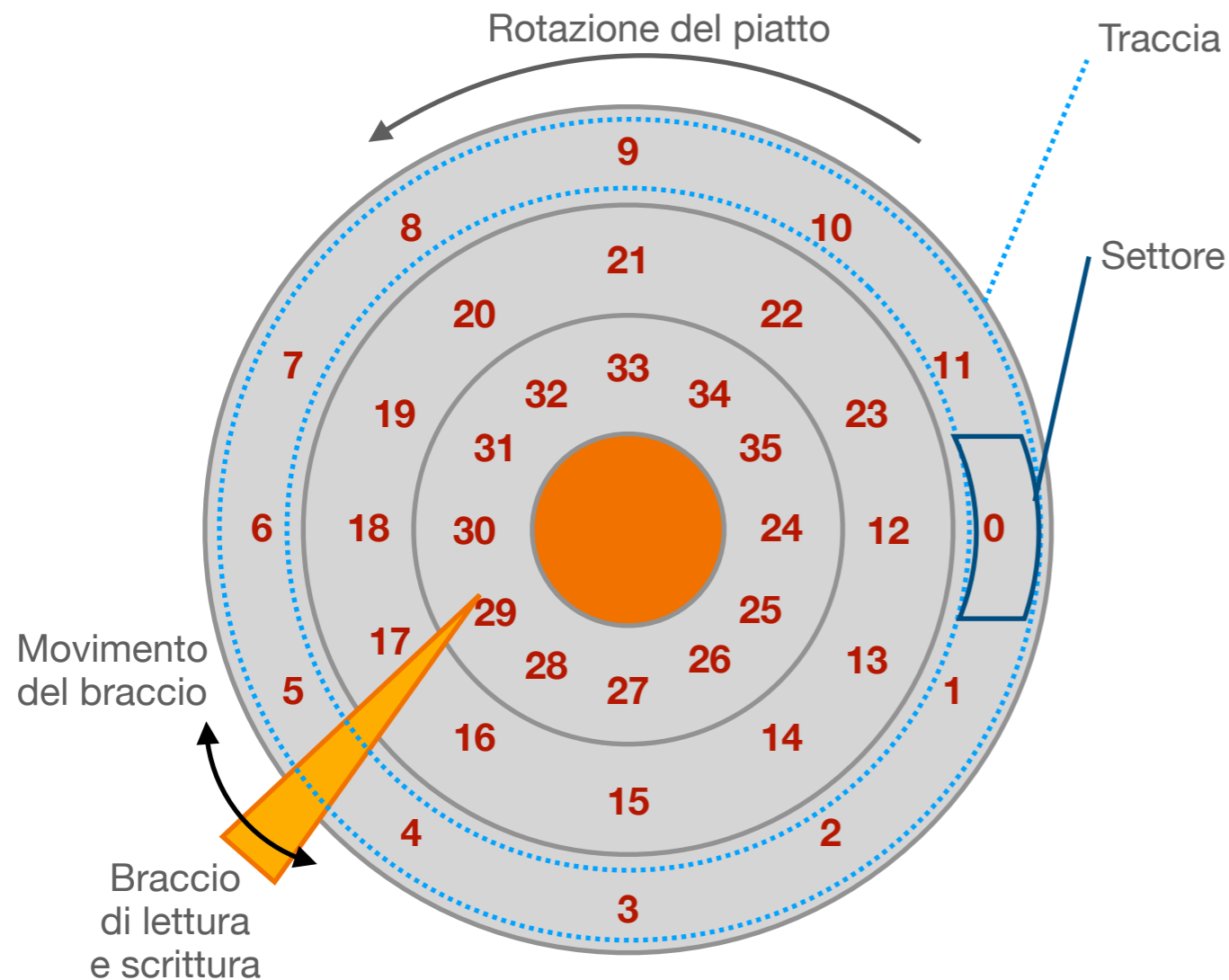
Dispositivi a blocchi

L'astrazione che vogliamo ottenere

- Per poter definire un filesystem vogliamo avere una astrazione “comoda” per rappresentare i diversi dispositivi:
 - Dischi fissi, memorie flash, dispositivi ottici, etc.
- Per questo usiamo una astrazione in cui tutto lo spazio a disposizione in ognuno di questi dispositivi è suddiviso in blocchi di dimensione fissata (e.g., 512bit, 4KB, etc)
- Ogni blocco è dotato di un indirizzo (possiamo supporre che n blocchi siano numerati $0, \dots, n - 1$)
- Per semplificare, assumiamo che il sistema operativo ci permetta di leggere/scrivere un singolo blocco alla volta

Dischi fissi

Settori e tracce

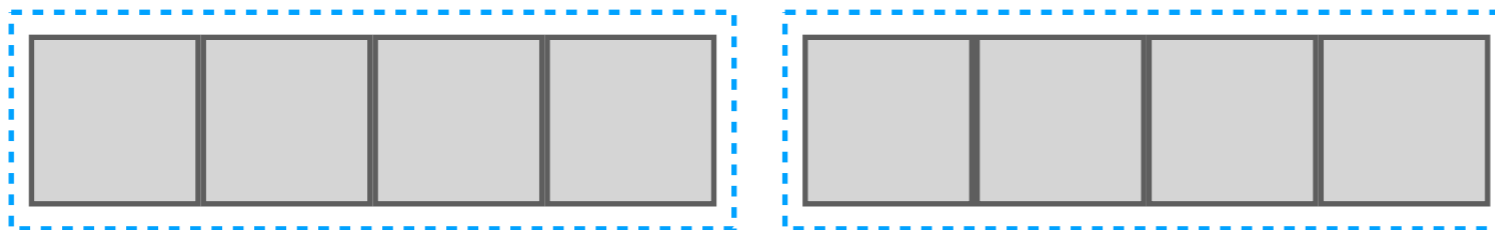


- Un disco fisso è diviso in più piatti, ognuno con un braccio che permette di leggere e scrivere
- I piatti ruotano a alcune migliaia di giri al minuto (rpm). Valori comuni sono 5400 o 7200 rpm
- Ogni piatto è diviso in più tracce concentriche
- Ogni traccia è divisa in più settori (tradizionalmente da 512 bytes l'uno)
- Per leggere o scrivere un settore il braccio deve spostarsi sulla traccia corretta (tempo di seek)
- Una volta sulla traccia corretta deve attendere che il settore corretto passi sotto la testina di lettura/scrittura

Memoria flash

Blocchi e pagine

Stessi nomi ma significati diversi rispetto a quello che abbiamo già visto



Una memoria flash è divisa in pagine raggruppate in blocchi

Pagina

00

01

02

03

04

05

06

07

Blocco

0

1

Le operazioni possibili sono:

- Lettura di una **pagina**
- Cancellazione di un **blocco**
- Programmazione (scrittura) di una **pagina** (solo su un blocco cancellato)

Notate che se vogliamo scrivere una pagina dobbiamo cancellare il blocco che la contiene!

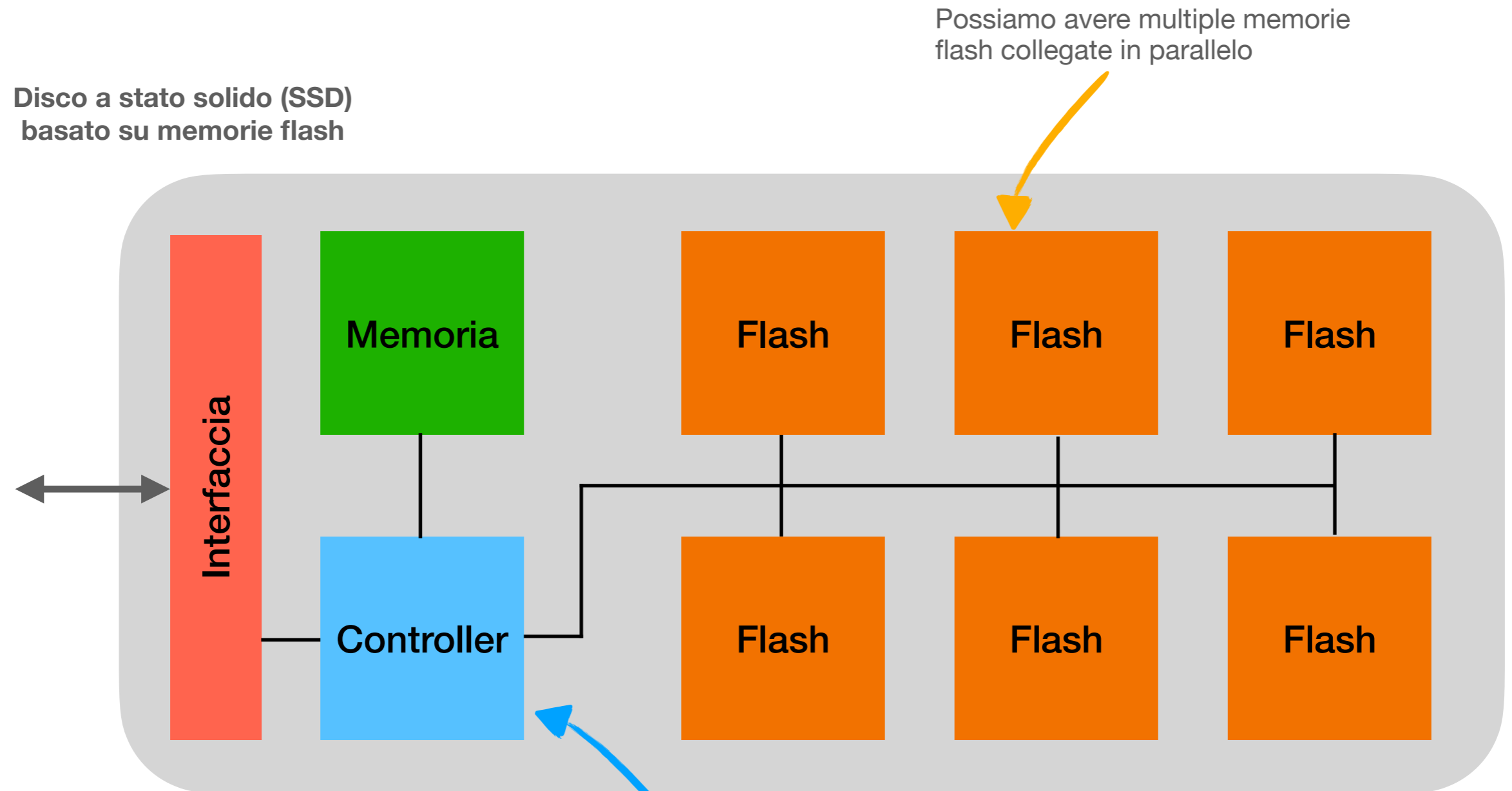
Dobbiamo salvare e ri-scrivere tutte le altre pagine del blocco che contengono informazione che vogliamo salvare

In aggiunta a questo ogni blocco ha un numero limitato di scritture possibili!

Come viene creato un disco a stato solido (SSD) usando memorie flash?

SSD

Disco a stato solido (SSD)
basato su memorie flash



Possiamo avere multiple memorie flash collegate in parallelo

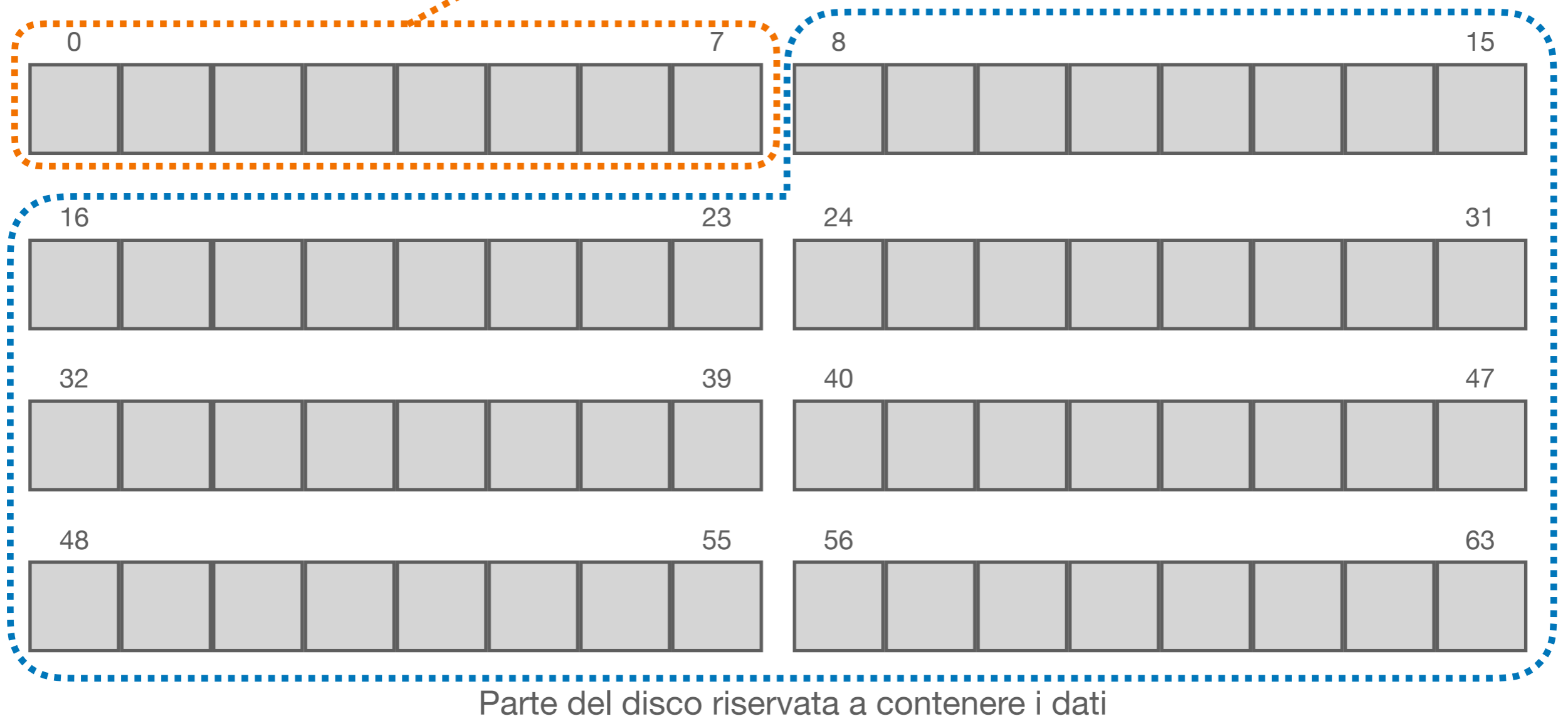
Il controller si occupa di distribuire le scritture in modo da rendere uniforme l'usura della flash (*wear levelling*) e di evitare che la quantità di scritture effettuate realmente rispetto a quelle richieste dall'OS sia bassa (limitate la *write amplification*)

Simple Filesystem

Imitare il “vecchio” Unix Filesystems

Supponiamo di avere un dispositivo a blocchi (in questo caso con 64 blocchi) tutti di dimensione fissata. Come facciamo ad ottenere file e directory?

Riserviamo una parte dei blocchi a contenere informazioni sul contenuto degli altri blocchi (i.e., i metadati)



Metadati

Inodes, bitmap, superblocchi

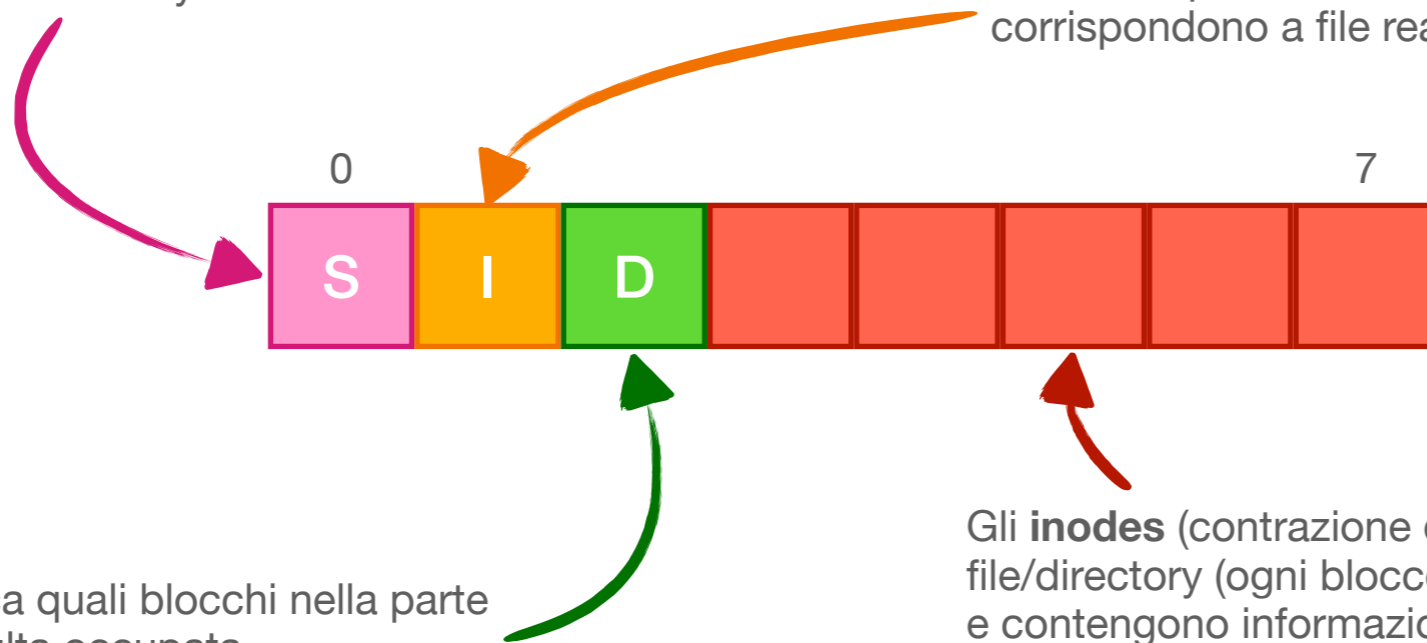
Vediamo che tipologie di metadati contiene il nostro “Simple filesystem”:

Superblocco

Il superblocco contiene informazioni relative all'intero filesystem, come quali blocchi sono dedicati ai dati, dove si trovano gli inodes, la inode bitmap e la data bitmap. Di solito contiene anche un “magic number” che permette di identificare il tipo di filesystem

Inode bitmap

Una bitmap che ci dice quali inode sono occupati (i.e., corrispondono a file realmente esistenti) e quali sono liberi



Data bitmap

Una bitmap ci indica quali blocchi nella parte dedicata ai dati risulta occupata

Gli **inodes** (contrazione di **index nodes**) sono uno per file/directory (ogni blocco ne contiene molteplici) e contengono informazioni come:

- Che blocchi occupa un certo file
- Chi è il proprietario
- I permessi
- Tempo di ultimo accesso/modifica

Inode

Informazioni su un file

Un inode è rappresentabile da una struct con diversi campi:

rwxr-xr-x	Permessi: il file può essere letto/scritto/eseguito?
1001	UID (User ID): Utente proprietario del file
4096	Dimensione
01/04/2021 11:37	Time: momento di ultimo accesso al file
31/01/2019 9:22	Ctime: momento in cui il file è stato creato
10/03/2020 22:01	Mtime: momento dell'ultima modifica del file
-	Dtime: momento in cui l'inode è stato cancellato
1005	GID (Group ID): Gruppo proprietario del file
8	Blocchi: numero di blocchi allocati per il file
-	Altri campi: ACL (access control list — permessi aggiuntivi), etc.
	Puntatori ai blocchi (solitamente decine): puntatori ai blocchi occupati dal file

Timestamps

Vediamo come funzionano i puntatori ai blocchi e gli indici multi-livello

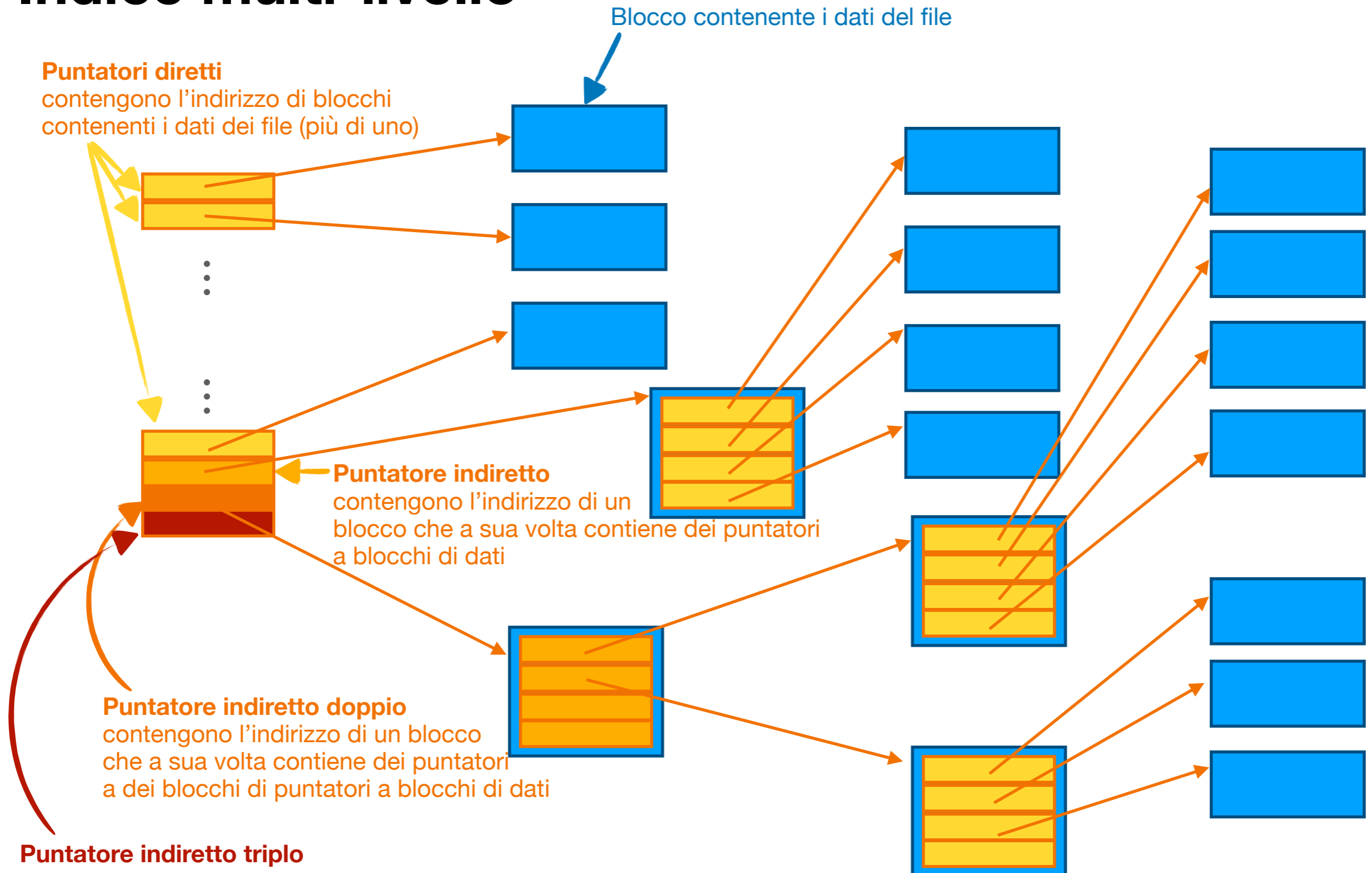
Puntatori ai blocchi occupati

Indice multi-livello

Blocco contenente i dati del file

Puntatori diretti

contengono l'indirizzo di blocchi contenenti i dati dei file (più di uno)



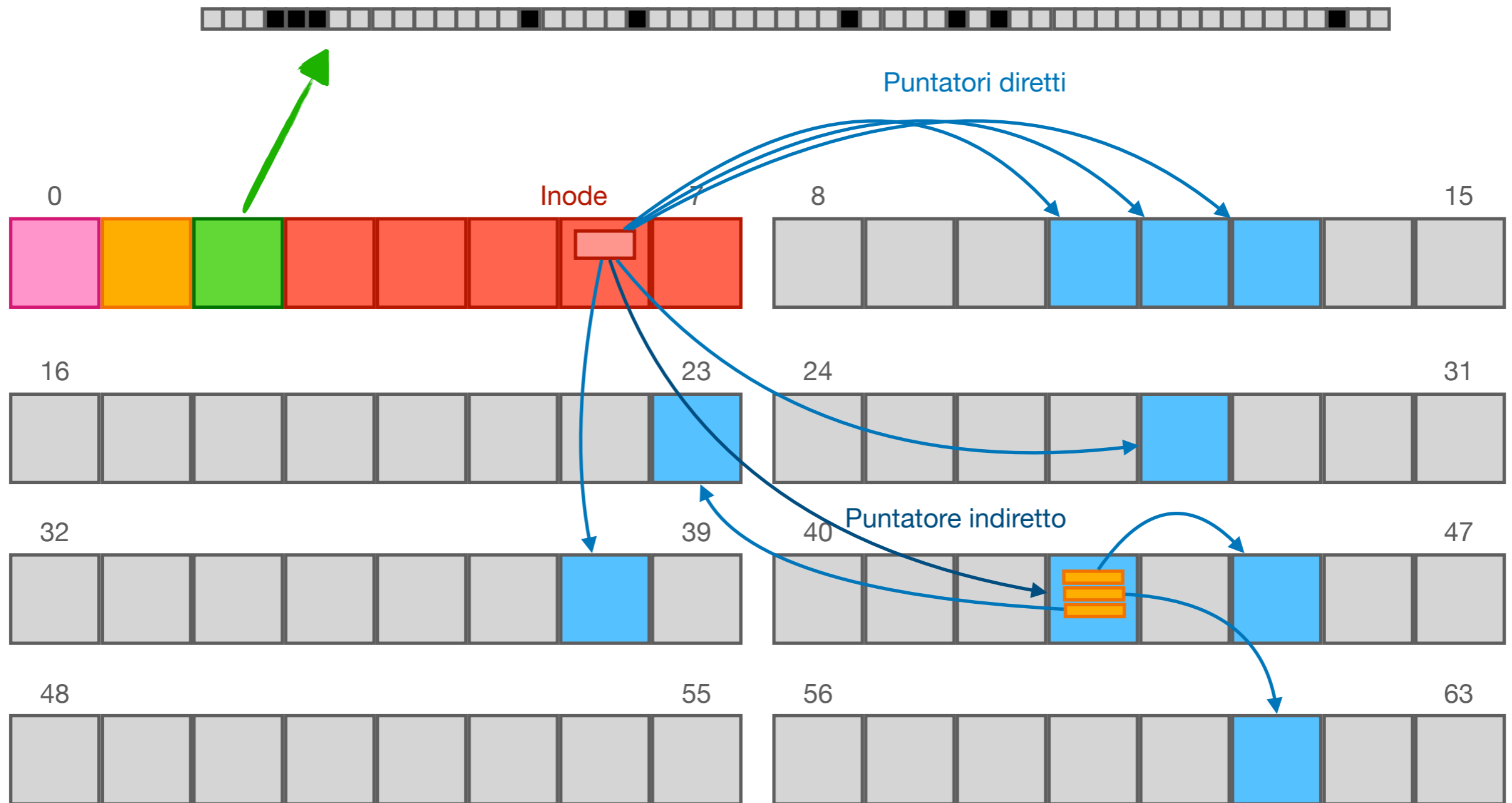
Perché questa struttura multi-livello?

Memorizzare efficacemente i file

- Supponiamo di avere blocchi di 4KB e ogni puntatore occupa 4 byte
- Supponiamo di avere 10 puntatori diretti
- Con 10 puntatori diretti possiamo indirizzare 10 blocchi (40KB)
- Ogni blocco, se usato per memorizzare puntatori, può contenere 1000 indirizzi di altri blocchi
- Un primo livello di indirettezza ci consente 1000 blocchi (4MB) aggiuntivi
- Un secondo livello di indirettezza ci consente 1000 blocchi ognuno con riferimenti a 1000 blocchi, consentendo 4GB aggiuntivi (i.e., 10^6 blocchi)
- Un terzo livello di indirettezza ci consente 1000 blocchi ognuno con un 1000 riferimenti a blocchi ognuno con 1000 riferimenti a blocchi, consentendo 4TB aggiuntivi (i.e., 10^9 blocchi)
- La dimensione massima di un file è quindi poco oltre 4TB

Dove si trova il file?

La Data Bitmap ci indicherà quali blocchi sono occupati



Accedere a un file

E modificare il contenuto

- Per poter accedere a un file abbiamo bisogno di sapere l'inode associato (che supponiamo di avere)
- Se vogliamo accedere ad una specifica posizione dobbiamo prima individuare in quale blocco è locata. Se il blocco è accessibile direttamente possiamo leggerlo, altrimenti è necessario leggere uno o più blocchi di puntatori
- Se cambiamo la dimensione del file aggiungendo o togliendo contenuto dobbiamo aggiornare i dati nell'inode e i dati nella data bitmap
- Se il file viene cancellato non è necessario eliminare solo l'inode ma anche segnare come liberi tutti i blocchi che occupava nella data bitmap

E le directory?

Directory come file

- Possiamo vedere una directory come un normale file
- Quello che cambia è il contenuto del file
- Nella sua forma più semplice una directory è un file il contenuto è una sequenza di coppie (**numero inode, nome file**):
 - (123, Esempio.txt)
 - (281, Esercizi)
 - ...
- Possiamo quindi memorizzare le directory come normali file, usando solo una interpretazione diversa del loro contenuto

Performance

E perché non è tutto così semplice

- Implementato in questo modo un filesystem che utilizzasse un disco fisso sarebbe molto lento
- I blocchi in cui è salvato il contenuto di un file sono salvati in modo “casuale” sul disco. Ogni lettura richiederebbe di riposizionare la testina e attendere il passaggio del settore sotto di essa
- Per questo i sistemi moderni tendono a cercare di allocare i file in blocchi consecutivi, in modo da poter leggere il contenuto spostando la testina del disco il meno possibile
- Questo ovviamente non vale per le memorie flash!

Consistenza

Conservare i dati in caso di spegnimento inatteso

- Cosa succede se il sistema si spegne mentre stiamo facendo una lettura/scrittura?
- Per esempio potremmo aver aggiornato un inode ma non la bitmap e il filesystem sarebbe in uno stato inconsistente
- Come risolvere?
 - Creare un programma di riparazione del filesystem (fsck). Approccio usato da ext2, ufs, fat32, ...
 - Ordinare le scritture per limitare il tipo di inconsistenze che si possono presentare (soft updates). Approccio usato da UFS2 con soft updates (FreeBSD)

Consistenza (2)

Conservare i dati in caso di spegnimento inatteso

- **Filesystem journaled.**

Tenere un journal delle operazioni che si stanno per svolgere, svolgerle, cancellare il journal. Se il journal non è vuoto si possono ripetere (o annullare) le operazioni per arrivare a uno stato consistente.

Questo approccio è usato da ext3, ext4, xfs, hfs+, ntfs,...

- **Log-based filesystem.**

Non si sovrascrivono mai i dati precedenti ma si aggiungono solo. Serve però avere un processo di “pulizia” che elimina i dati non più raggiungibili.

- Simili ai log-based filesystem sono i filesystem copy-on-write, che copiano (senza sovrascrivere) i blocchi.