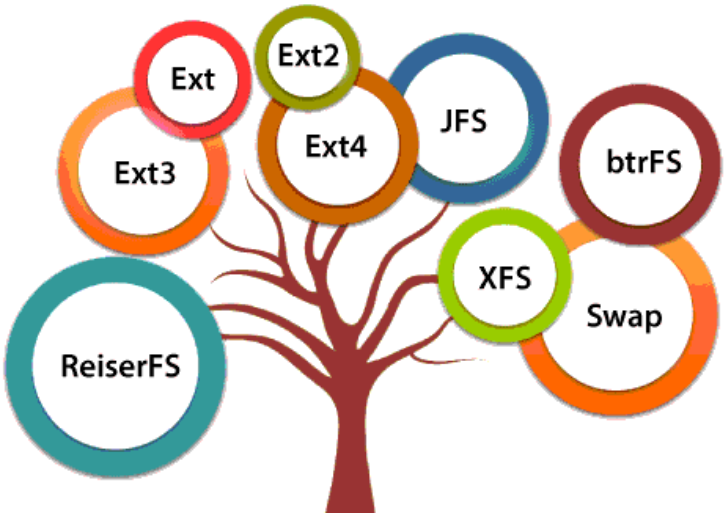
SISTEMI OPERATIVI (M-Z)



C.d.L. in Informatica

(laurea triennale)

Dip. di Matematica e Informatica

Università degli Studi di Catania

Anno Accademico

2023-2024

Prof. Mario F. Pavone

mpavone@dmi.unict.it

File System e Dischi

IL FILE SYSTEM

- Problema di base: gestire **grandi quantità** di informazioni, in modo **persistente** e condiviso **tra più processi**;
- astrazione: **file** e **directory**;
- i dettagli di gestione ed implementazione costituiscono il **file system**;
- **Un file è una sequenza di bit, byte, righe o record!!**
- esempi di **dettagli**:
  - nomenclatura;
  - tipi di file;
  - tipi di accesso;
  - metadati (o attributi);
  - operazioni supportate sui file;

I FILE SYSTEM

- **Struttura Directory** – directory structure:
  - Nome
  - Identificatore univoco
- Operazioni che possono essere eseguite su un file
  - *creare*, *leggere*, *scrivere*, *eliminare*, *riposizione*, *troncare*
- Più processi possono aprire contemporaneamente un file => uso di **due livelli di tabella interne**:
  - ***tabella per processo***
  - ***tabella a livello di sistema***
- **Conteggio delle aperture** di un file
  - *open*( ) aumenta il contatore
  - *close*( ) decrementa il contatore

I FILE SYSTEM

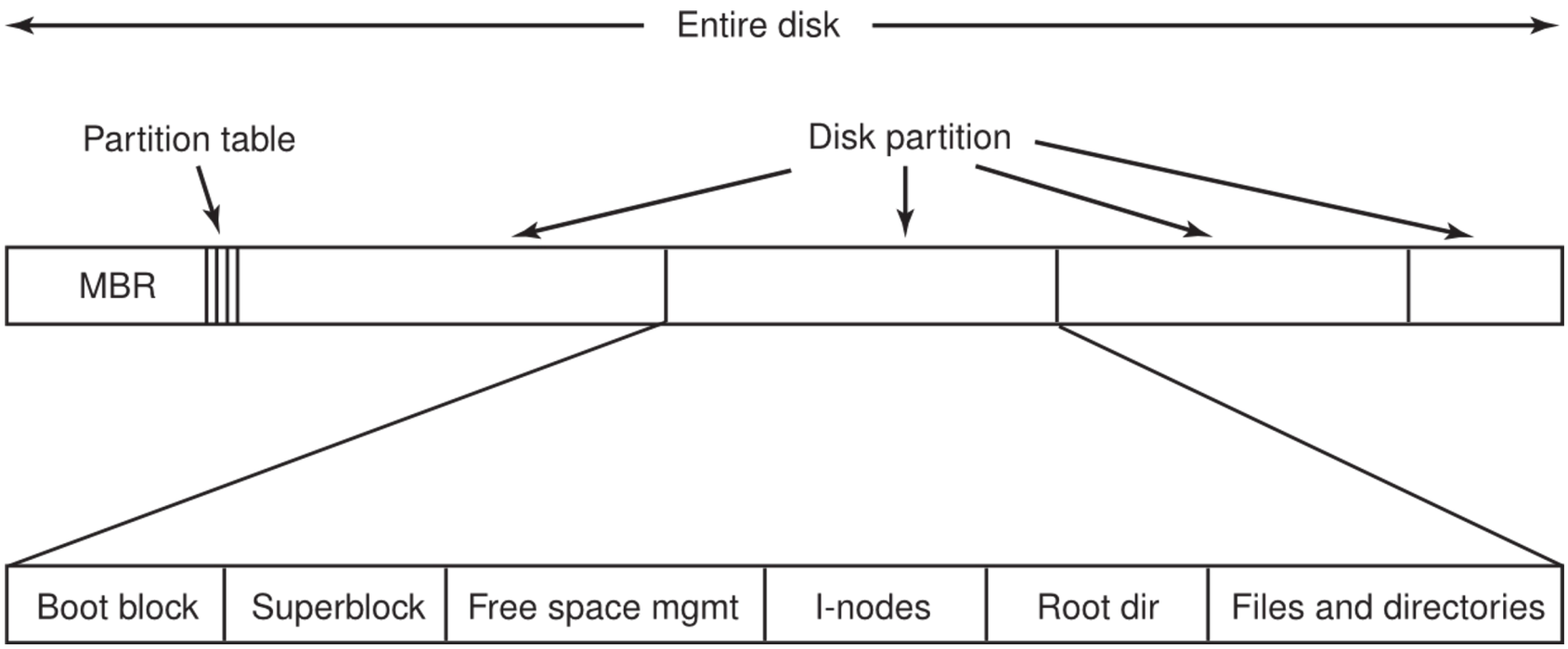
- **Informazioni** associate ad un file aperto:
  - Puntatore a file
  - Conteggio dei file aperti
  - Posizione nel disco del file
  - Diritti di accesso

- operazioni supportate sui file;
  - accesso condiviso ai file: i **lock**:
    - **shared** *vs.* **exclusive**;
    - **mandatory** *vs.* **advisory** **(obbligatori *vs.* consultivi)**;

- Tipi di file in Unix:
  - ***File regolari*** (anche in Windows)
  - ***File speciale a caratteri***
  - ***Flie speciale a blocchi***

STRUTTURA DI UN FILE SYSTEM

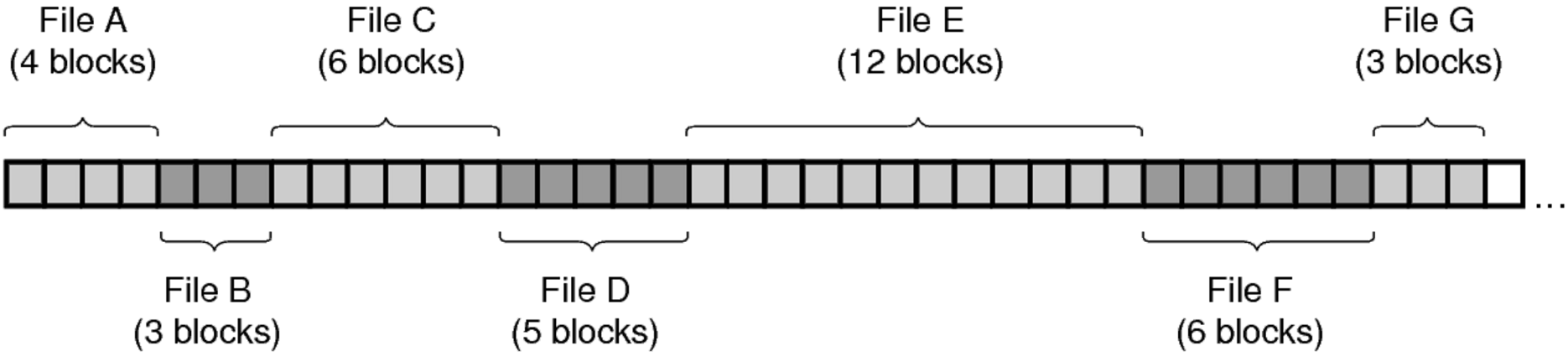
- **Master Boot Record** (MBR) => settore 0 del disco;
- partizioni e **boot record** (o boot block);
- **superblocco**;



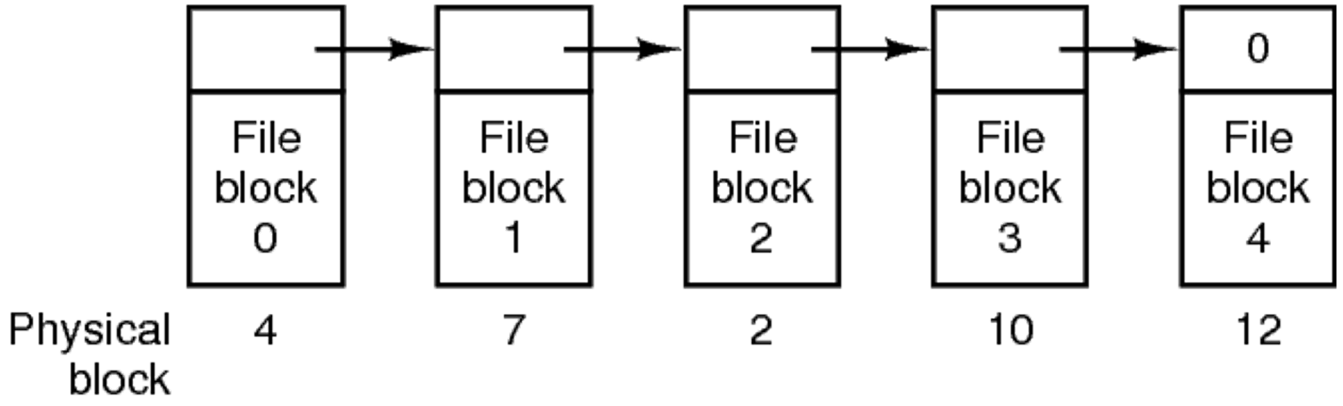
- layout moderno alternativo: **GPT** (GUID Partition Table) definito dallo standard **EFI**.

IMPLEMENTAZIONE DEI FILE

- **Allocazione contigua.**

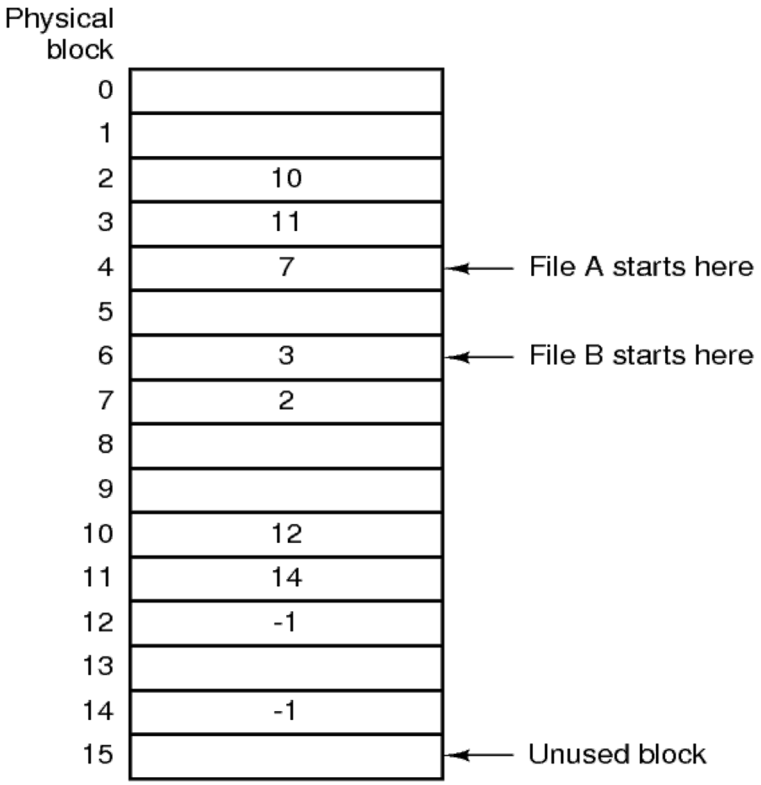


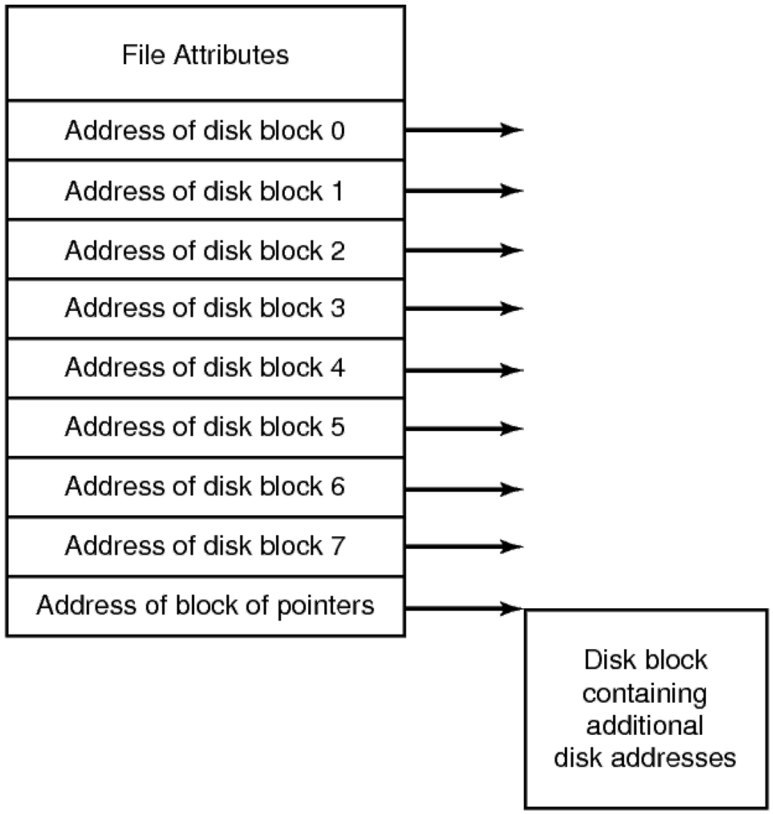
- **Allocazione con liste collegate** (o allocazione concatenata).



IMPLEMENTAZIONE DEI FILE

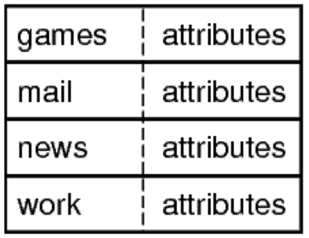
- **Allocazione con liste collegate su una tabella di allocazione dei file** (file allocation table – **FAT**) (o allocazione tabellare).
- **Allocazione con nodi indice** (index node – **i-node**) (o *allocazione indicizzata*):
  - varianti: **collegata**, **multilivello, ibrida.**

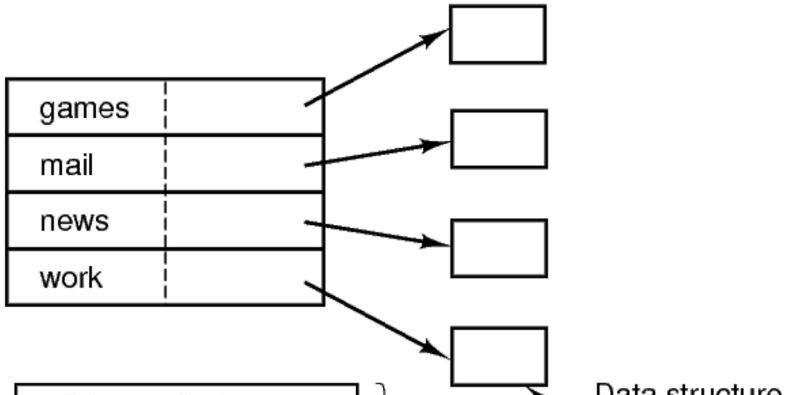




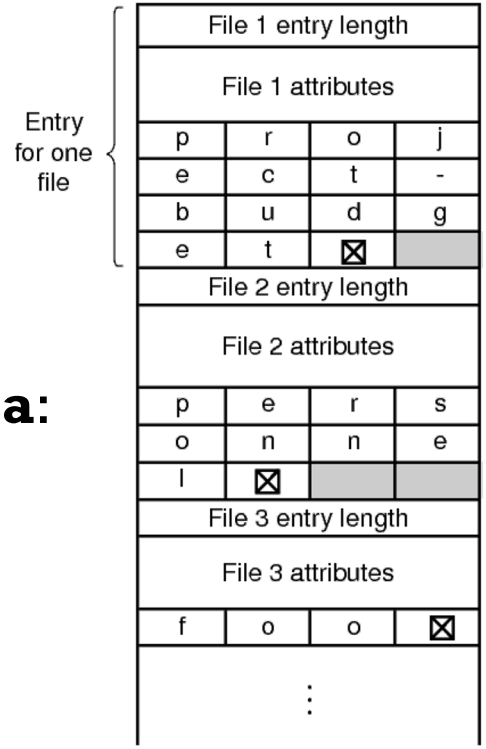
IMPLEMENTAZIONE DELLE DIRECTORY

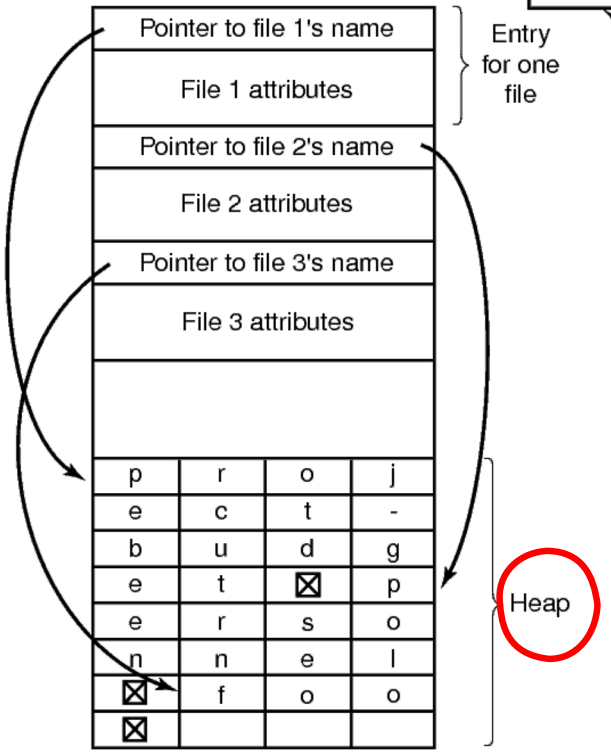
- Dove memorizzare i **metadati/attributi**?





- **nomi lunghi:**
  - lung. variabile;
  - tramite heap;

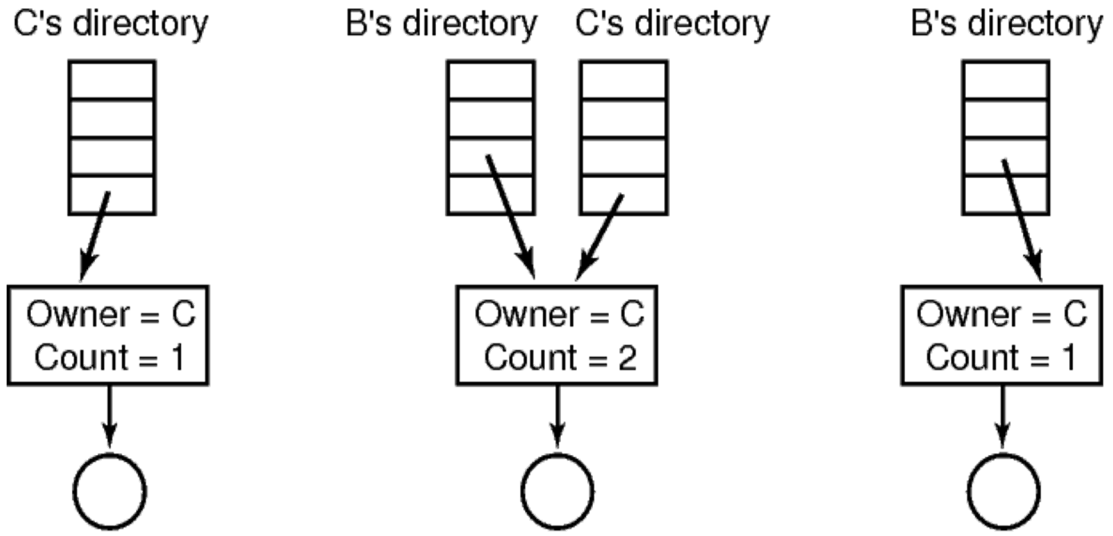




- **prestazioni ricerca**:
  - tabella hash;
  - cache.

CONDIVISIONE DI FILE SU UN FILE SYSTEM

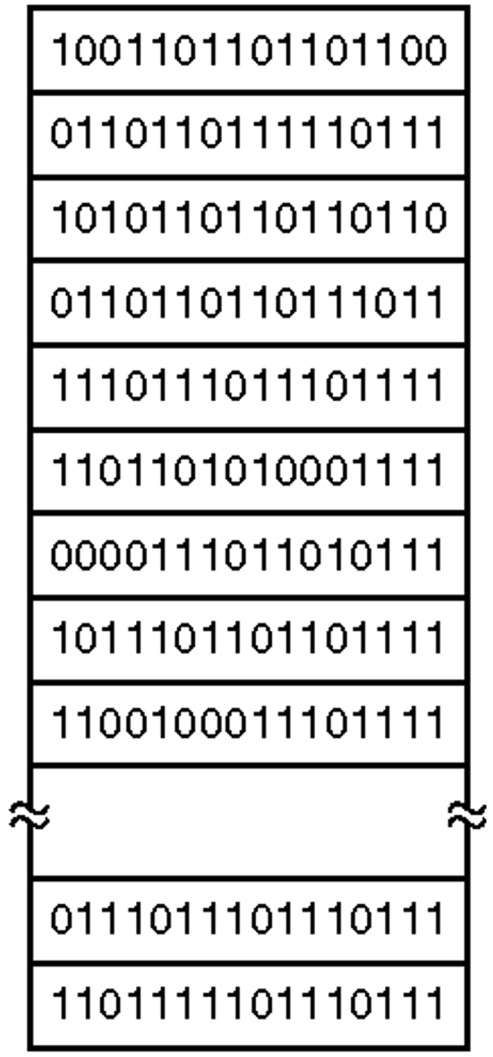
- **Scenario**: due o più utenti vogliono condividere un file;
- usando una FAT: duplicare la lista con i riferimenti ai blocchi;
  - problemi in caso di append;
- usando i-node: **hard-link**;
  - contatore dei link;
  - anomalia con accounting;



- ulteriore approccio: **soft-link**;
  - universale e permettere di fare riferimenti al di fuori del file system;
  - appesantimento nella gestione;
- eventuali **problemi** in fase di attraversamenti e backup.

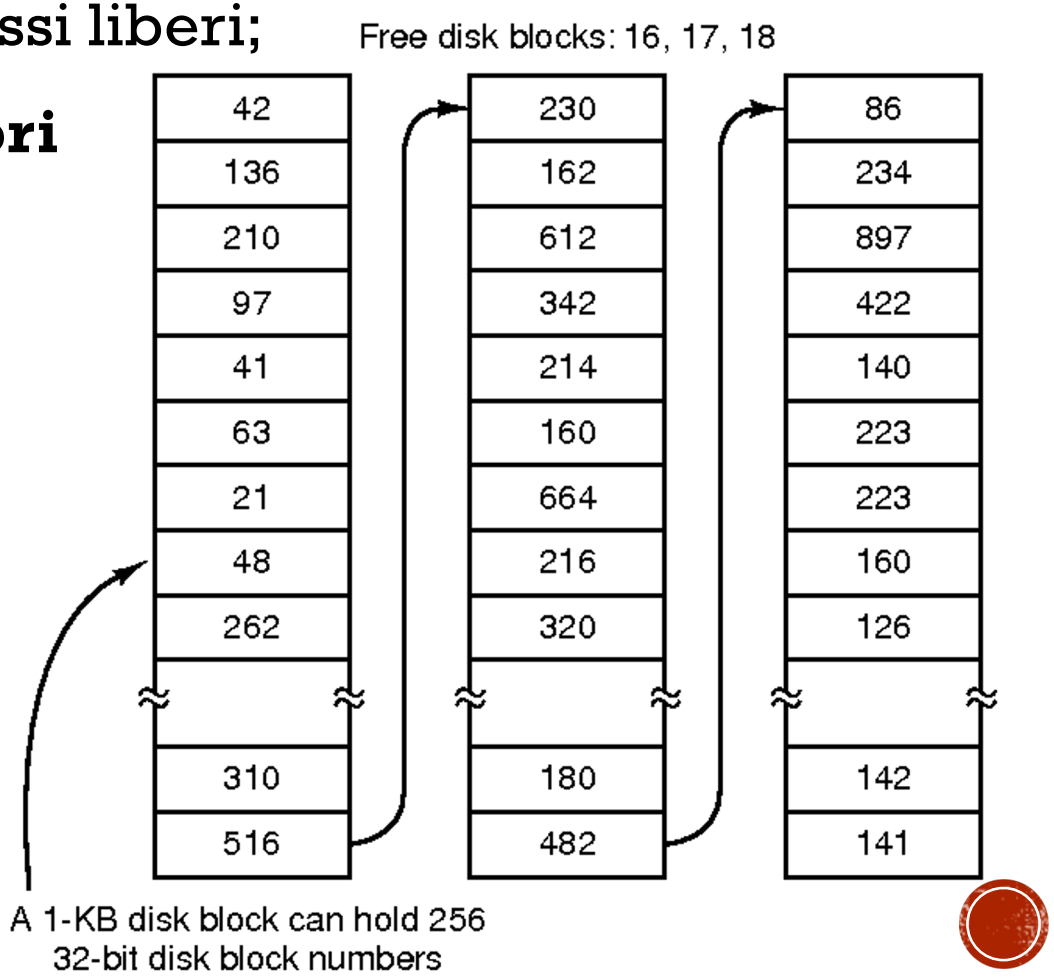
GESTIONE BLOCCHI LIBERI

- Attraverso l'uso di una **bitmap**:
  - relativamente **piccola**;
  - strategie di **allocazione in memoria**:
    - tutta in memoria, o;
    - un blocco alla volta;
      - paginata con VM;



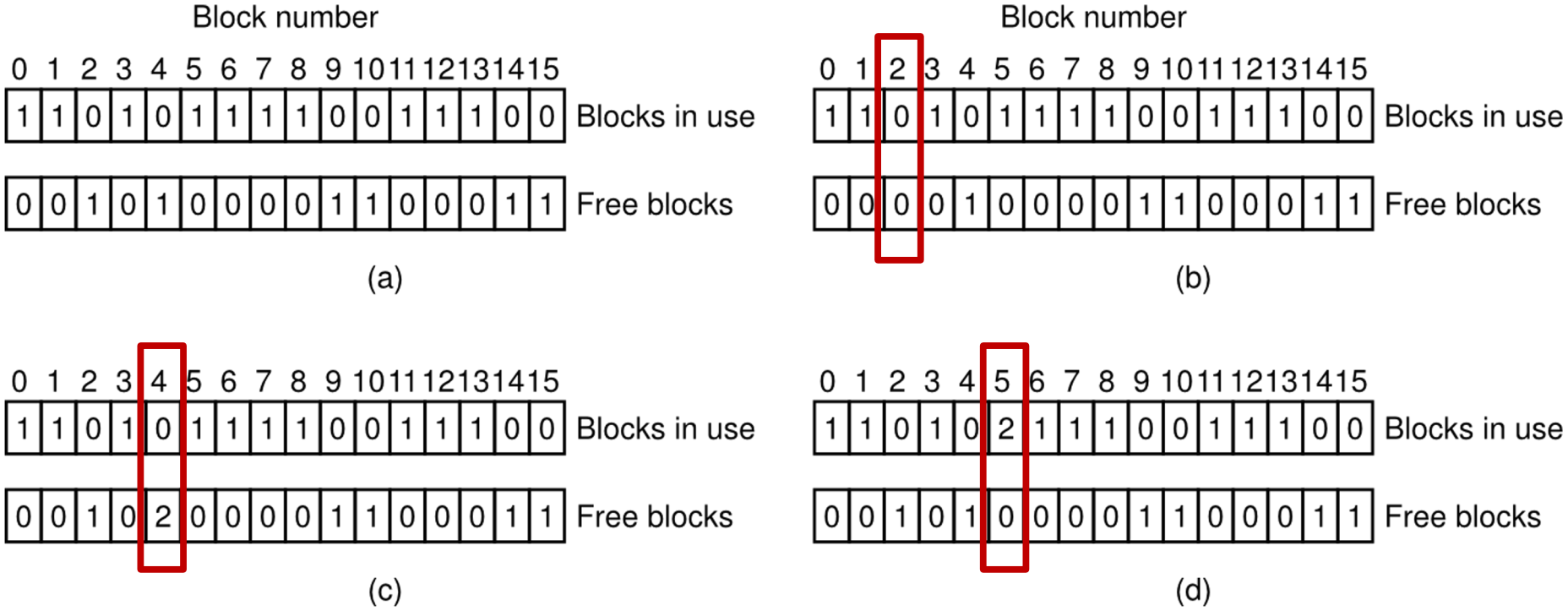
GESTIONE BLOCCHI LIBERI

- attraverso l'uso di **liste concatenate**:
  - richiede più spazio;
    - ma si sfruttano i blocchi stessi liberi;
  - possibilità di inserire **contatori** per blocchi contigui.

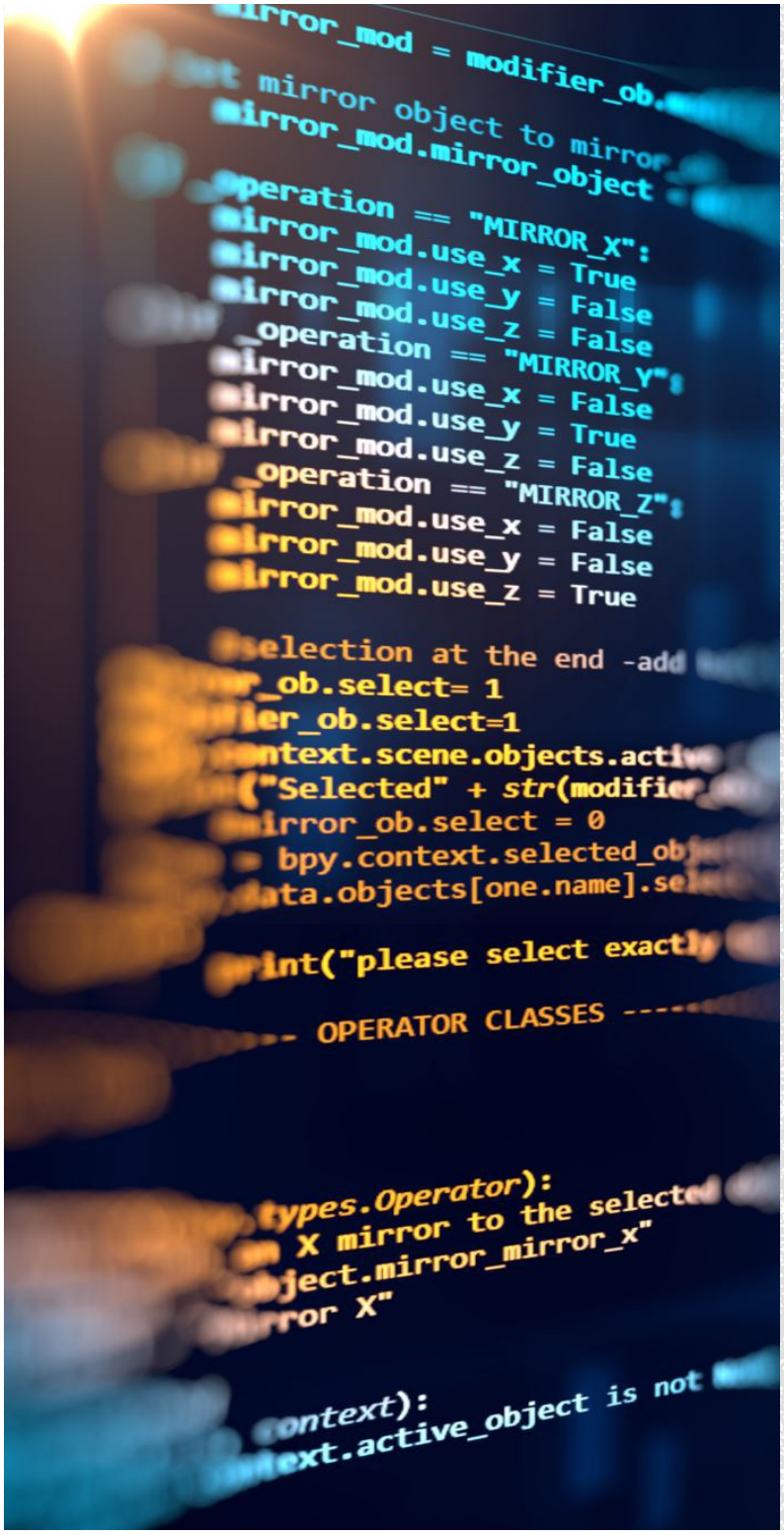


CONTROLLI DI CONSISTENZA

- A seguito di **crash** del sistema i file-system possono diventare inconsistenti;
- apposite **utility** possono effettuare dei **controlli di consistenza**:
  - sui **blocchi**;



  - sui **riferimenti agli i-node.**



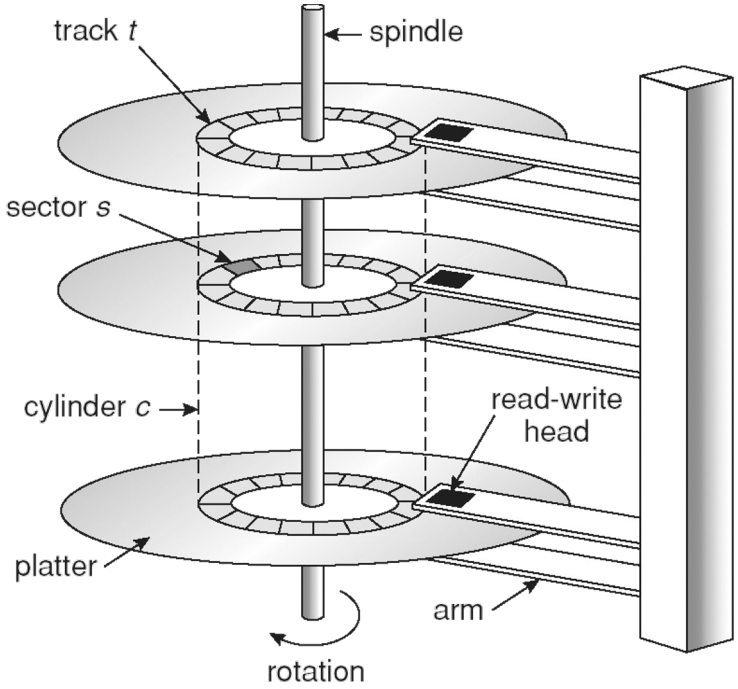
JOURNALING

- **strategia**:
  - le operazioni sui meta-dati sono preliminarmente appuntate in un log che viene poi ripulito a posteriori (subito dopo);
  - in caso di ripristino da crash: ripetere le operazioni in log;
- **benefici:**
  - maggiore robustezza dei metadati;
  - veloce ripristino da crash/reboot;
- affinché tutto funzioni bisogna operare con operazioni **idempotenti**.

COSA USANO I NOSTRI SISTEMI OPERATIVI?

- **Windows**:
  - exFAT su unità removibili;
  - NTFS su dischi fissi:
    - file-system moderno, molto complesso;
    - journaling, cifratura, compressione, copia shadow (CoW), dischi multipli (RAID) …
- **Linux**:
  - **ext-4**:
    - journaling, allocazione efficiente;
  - **BTRFS**:
    - recente file-system evoluto
    - journaling, check-sum dati e metadati, compressione, volumi, clonazione (CoW), dischi multipli (RAID), …
- **MacOS**: HFS+ recentemente soppiantato da APFS.

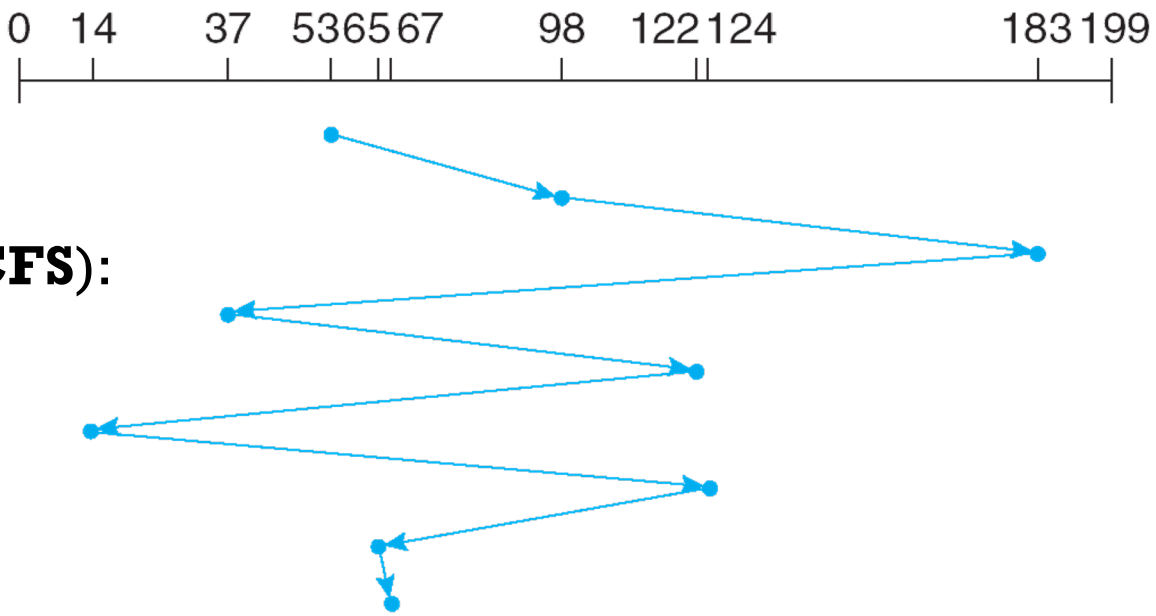
SCHEDULING DEL DISCO



- **Obiettivi**:
  - massimizzare il numero di richieste soddisfatte in una unità di tempo (**throughput**);
  - minizzare il **tempo medio di accesso**;
- in un sistema, soprattutto se multiprogrammato, si vengono a creare **varie richieste di I/O su disco** che però, tipicamente, possono essere inviate al controller del disco solo una alla volta. Si crea quindi una **coda** di **richieste pendenti**;
- Il S.O. può adottare varie **politiche di selezione** della prossima richiesta da mandare;
- si può ottimizzare per:
  - **tempo di posizionamento** (seek-time);
  - **latenza di rotazione**

OTTIMIZZARE IL SEEK-TIME

- Vedremo varie politiche di scheduling su uno specifico **esempio**:
  - lista delle richieste in ordine di arrivo e per # di cilindro:
    98, 183, 37, 122, 14, 124, 65, 67
  - posizione iniziale della testina: cilindro ***53***

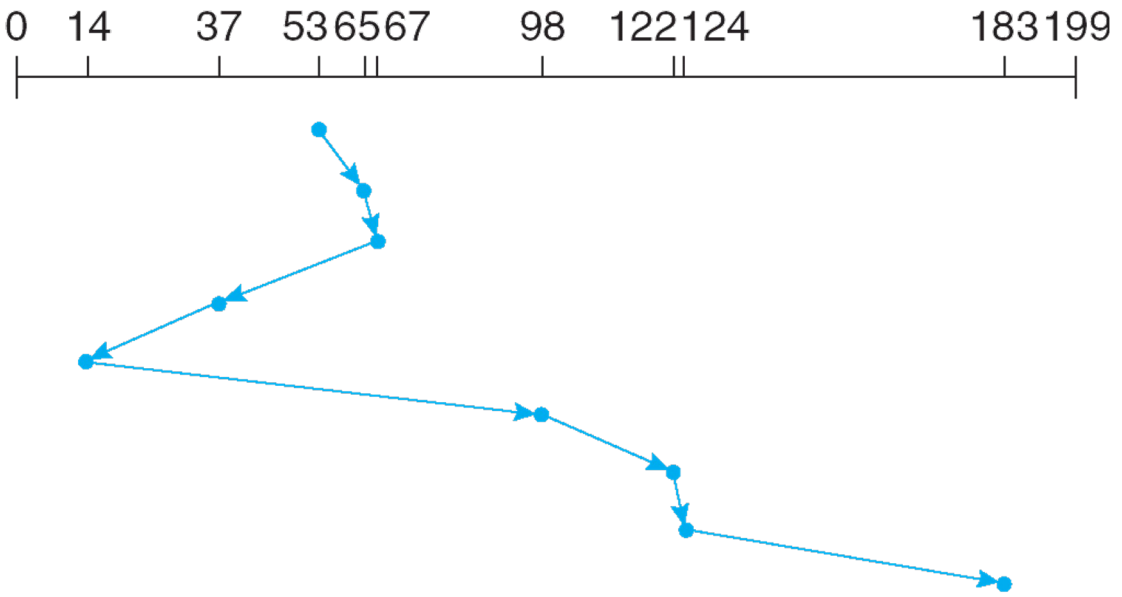


- **First Come First Served** (**FCFS**):
  - distanza totale percorsa:
    640 tracce;
  - **semplice** da realizzare;
  - **equo**;
  - **inefficiente**;

OTTIMIZZARE IL SEEK-TIME

- **esempio**: coda ***98, 183, 37, 122, 14, 124, 65, 67***; cilindro iniziale ***53***;

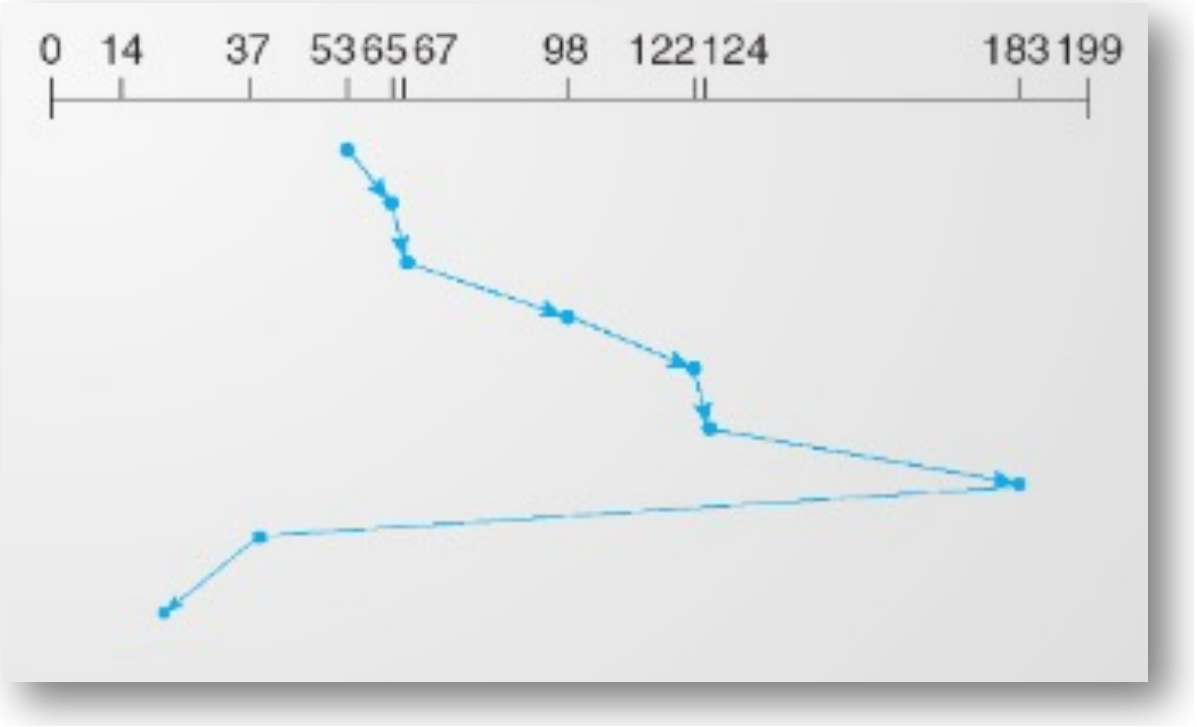
- **Shortest Seek Time First** (**SSTF**):
  - ordine usato: 65, 67, 37, 14, 98, 122, 124, 183;
  - distanza totale percorsa: 236 tracce;
  - **buone prestazioni** (ma non ottimale);
  - **non equo** (*starvation*);



OTTIMIZZARE IL SEEK-TIME

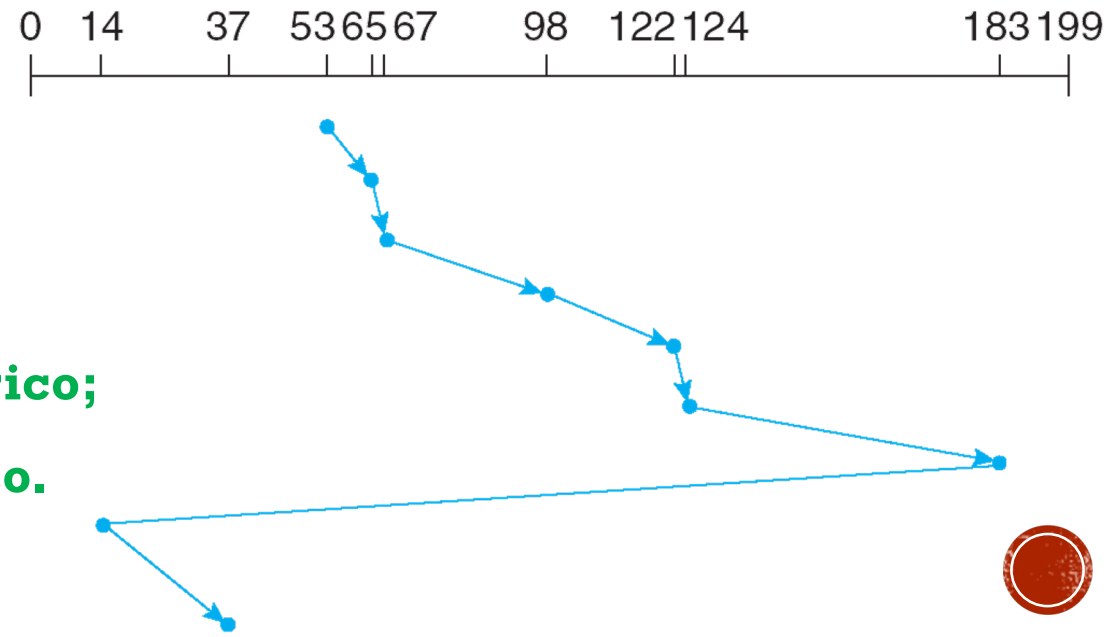
- **esempio**: coda ***98, 183, 37, 122, 14, 124, 65, 67***; cilindro iniziale ***53***;
- **Scheduling per scansione** (**algoritmo dell'ascensore**):
  - mantiene un verso fino all'ultima richiesta in tale direzione;
  - ordine usato: 65, 67, 98, 122, 124, 183, 37, 14
  - distanza totale percorsa: 299 tracce;
  - scansione uniforme;

garantisce comunque una **attesa massima** per ogni richiesta



OTTIMIZZARE IL SEEK-TIME

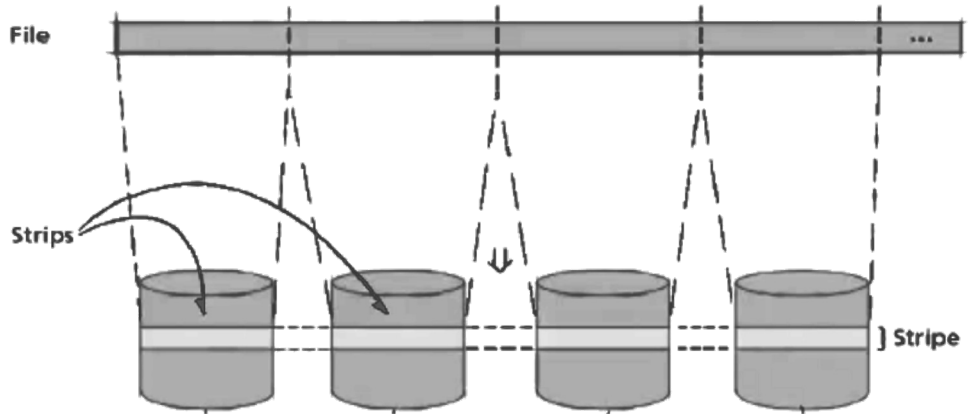
- **esempio**: coda ***98, 183, 37, 122, 14, 124, 65, 67***; cilindro iniziale ***53***;
- **Scheduling per scansione circolare**:
  - considera le posizioni come collegate in **modo circolare**: **arrivato alla fine del disco torna sul primo cilindro senza servire alcuna richiesta**;
  - ordine usato: 65, 67, 98, 122, 124, 183, 14, 37
  - garantisce un **tempo medio di attesa** più **basso** in presenza di tante richieste;



- **Come scegliere?**
  - **Scansione circolare ad alto carico;**
  - **Scansione o SSTF a basso carico.**

SISTEMI RAID

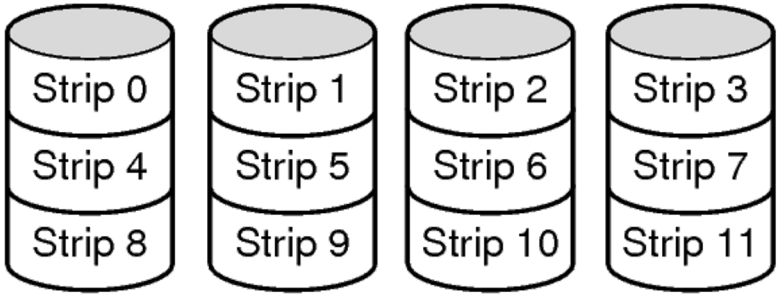
- Un altro modo per aumentare le **prestazioni** è sfruttare il **parallelismo** anche per l'I/O su disco:
  - servono più **dischi indipendenti**;
  - si suddividono i dati relativi ad una unità logica (un file, in generale un volume) su più dischi: **striping**;
    - suddivisione **trasparente all'utente**;



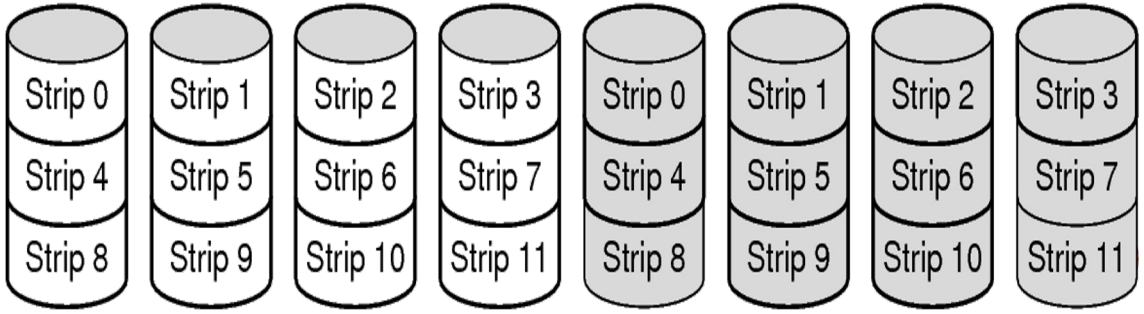
- problema: aumenta la probabilità che si **verifichi un guasto** sul volume logico RAID;
  - soluzione: aggiungiamo ridondanza per ottenere migliore **affidabilità**;
  - sostituzione automatica: dischi **spare**;
- **Redundant Array of Inexpensive Disks** (**RAID**);
  - noti anche come **Redundant Array of Independent Disks**;
- vedremo vari schemi di gestione che bilanciano questi due aspetti;
  - **livelli RAID**;
- via **hardware** (trasparente al S.O.) o via **software** (con carico sulla CPU).

SISTEMI RAID

- **RAID 0** (*striping*):
  - fa ***striping*** in modalità round-robin;
  - semplice con **prestazioni ottimali** con letture di grandi volumi;
  - niente ridodanza: **maggiore vulnerabilità**;

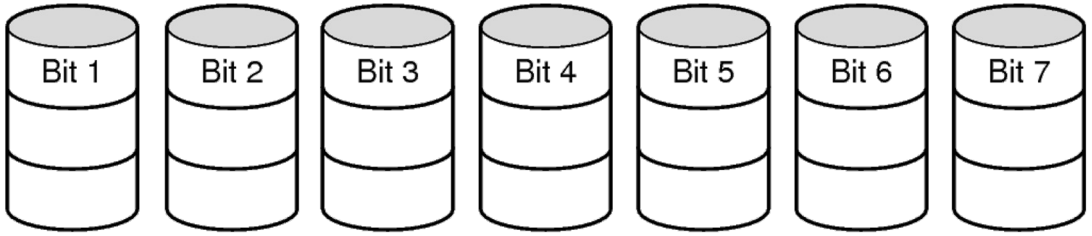


- **RAID 1** (*mirroring*):
  - gli eventuali stripe vengono anche duplicati (***mirroring***);
  - può anche essere usato senza striping;
  - raddoppio prestazioni in lettura;
  - migliore ***fault tollerance***;
  - alto **overhead** di storage;

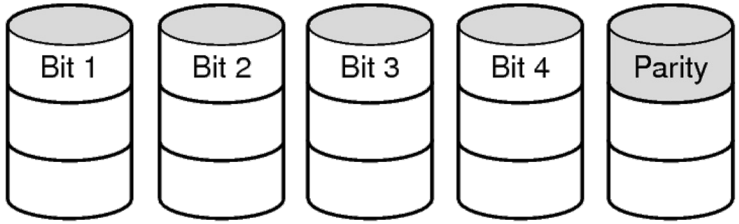


SISTEMI RAID

- **RAID 2** (*striping a livello di bit con ECC*):
  - lavora sulle parole o byte applicando un codice di correzione degli errori – ECC (tipo **codice di Hamming** per singoli bit di errore);
  - esempio: 4 bit dati + 3 bit ridondanza;
  - buone prestazioni;
  - ottima **fault tollerance**;
  - serve **sincronizzare** le rotazioni dei dischi;



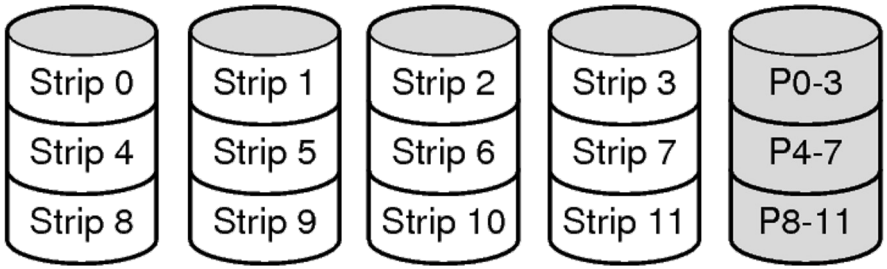
- **RAID 3** (*striping a livello di bit con bit di parità*):
  - usa un solo disco con raccolti i singoli **bit di parità**;
  - in realtà permette anche di **recuperare** i dati e offre la stessa capacità di fault tollerance del RAID 2;
  - serve ancora sincronizzazione;



- fare striping a livello di bit è comunque pesante se non gestito a livello hardware;

SISTEMI RAID

- **RAID 4** (*striping a livello di blocchi con XOR sull'ultimo disco*):
  - basato su **stripe a blocchi**;
  - disco extra = **XOR** degli stripe;
  - non necessità sincronizzazione;
  - ottima fault tollerance;
  - **aggiornamento** lento in caso di modifica di un blocco?
    - **ottimizzazione**: il nuovo blocco di parità si può calcolare dal blocco sovrascritto e dal vecchio blocco di parità;



- **RAID 5** (*striping a livello di blocchi ma con informazioni di parità distribuite*):
  - il blocchi di parità del RAID 4 vengono distribuiti su tutti i dischi;
  - di fatto sostituisce il RAID 4.