CH11 -DISCHI

Struttura di memoria di massa:

- Da 10GB a TB
- HDD hanno capienza superiore rispetto alle NMV memorie non volatili

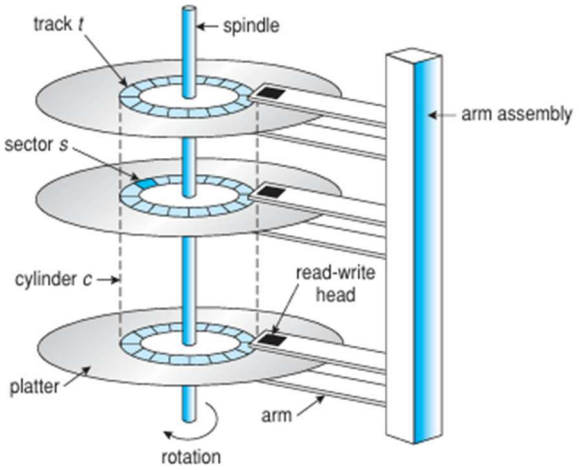
HDD

dispositivi in cui:

- Informazione immagazinata grazie a dispositivi magnetici
  - Bit si basano su bipoli magnetici
    - Isolando, su ogni piccola area, un insieme di bipoli magnetici orientandoli usando delle testine calamite

Dispositivo che ruota e una traccia a centri concentrici:

- Abbiamo dei dischi con delle piccole aree disposte su una circonferenza → **tracce**
  - bit messi in fila in diversi settori che formano una circonferenza
- Le testine sono in grado di modificare o leggere un campo magentico
- Tracce devono essere sottili
  - traccia (Fila di bit su una faccia)



Funzionamento:

- Bisogna capire dove scrivere un bit/byte:
  - **Identificare Cilindro** (fila di bit su più facce)
    - Bisogna spostare la testina ad una determinata distaza dal centro → cilindro
  - **Identificare settore**
    - Bisogna svolgere un movimento angolare per raggiungere il settore

Nota su **performance HDD**:

- Ruota da 60 a 250 volte al secondo → non molto veloce in realtà
- **Velocità di trasferimento** → quanti bit/byte vanno dal disco/verso il disco quando si legge o scrive
  - ~ 1GB/sec
- **Tempo di posizionamento** = seak time+latenza rotazionale
  - **Seek time** → tempo pre raggiungere il cilindro interessato
    - Dai 3ms ai 12ms
      - Varia in base a quello che si sta facendo e alla situazione attuale
      - seekTime medio calcolato come 1/3 delle tracce
  - **Latenza rotazionale** → tempo per raggiungere il settore, ruotando
    - dipende dal settore in cui mi trovo e dove devo andare
      - se si gira in una sola direzione:
        - caso peggiore → tempo per effettuare un giro
        - caso medio → mezzo giro

- Latency based on spindle speed
  - 1 / (RPM / 60) = 60 / RPM
- Average latency = ½ latency

- **Hard crash** → se la testina per qualche problema meccanico, non è alla distanza dovuta dal piatto e lo colpisce → buttare

- Access Latency = Average access time = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

4.17=(1/(7200/60))/2
→ conversione in Byte

Nota: frame → memoria fiscia
Pagine → memoria logica
Blocco → ora

NMV: non volatili basate su silicio → non perdono informazioni quando si spengono (SSD)

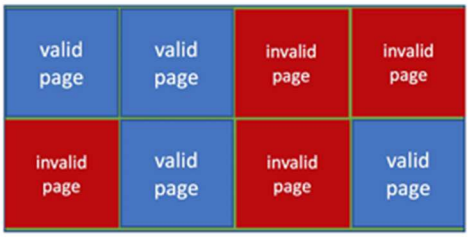
SSD

memorie DRAM con tecniche derivanti dal ROM dopo una serie di upgrade.

- Più affidabili degli hard disk
- Costano di più
- Potrebbero avere vita inferirore
- Meno capacità
- Molto più veloci

Leggere è più facile
Per scrivere, devo prima cancellare; inoltre c'è un limite sulla riscrittura (Es: max 100mila cancellazioni)

- Alcune pagine rischiano di non essere più valide perché hanno raggiunto un certo numero di riscritture
    - Bisogna poter marcare delle pagine come valide e altre come non valide → diminuisce lo spazio disponibile
        - Serve una tabella
        - Bisogna avere un garbage collector
        - Si alloca un po' più di memoria in modo tale che se uno dei pezzi diventa non valido, ne ho ancora un po'
        - Supponendo che serva leggere e scrivere molte volte il blocco 2500 → si può fare **wearLeveling** → ogni volta che riscrivi, scrivo da un'altra parte e faccio un mapping in modo tale da non dover scrivere sempre sullo stesso



NAND block with valid and invalid pages

Memorie non valitili NVM:

- non hanno problemi di scheduling.
- Il difetto delle NVM è che le operazioni di schittura costano di più → possono rallenatare.

Memorie volatili:

è possibile, in certi contesti, che servi una memoria di massa volatile → salvare solo con dispositivo acceso e poi perdere tutto. Questo perchè ci sono informazioni che sono viste dal sw come dei file che servono solo da accensione a spegnimento. → RAM

- Molto veloci
- Molto piccole

STRUTTURA DEL DISCO:

dal punto di vista logico , dischi sono organizzati come:

- **Vettore di blocchi**

Formattazione di basso livello → crea una tabella logica con corrispondenza tra blocchi logici e dove stanno fisicamente:

- Mappato sui settori (dove settore 0: primo settore sulla prima traccia dell'esterno)
- Traduzione dovrebbe essere semplice tranne in caso di bad sector
    - La miglior cosa sarebbe avere velocità di lettura costante

HDD Scheduling:

- Dischi non utilizzati necessariamente per una sola richiesta alla volta ma essendoci concorrenza, più processi e parti kernel che hanno bisogno del disco nello stesso momento, qualcuno dovrà decidere **chi passa prima e chi passa dopo** con l'obbiettivo di:
    - Minimizzare il tempo di seek per minimizzare il tempo perso
- **SO mantiene una coda delle richieste**
    - C'è solo un processo → servito tutto
    - Ci sono più richieste → algoritmo di ottimizzazione (gestito a bordo del disco)

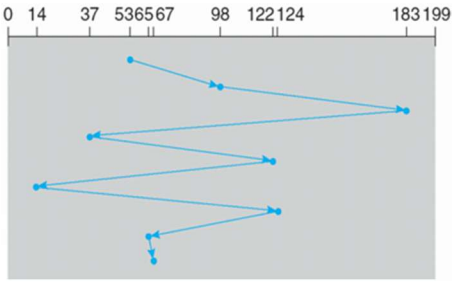
Algoritmi:

- We illustrate scheduling algorithms with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

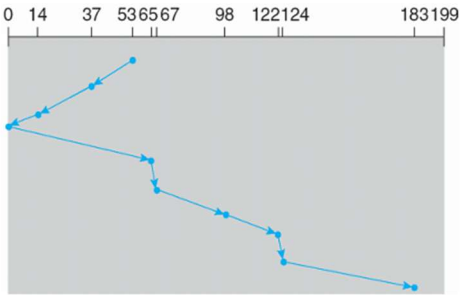
Head pointer 53

- **FCFS**: primo arrivato, primo servito

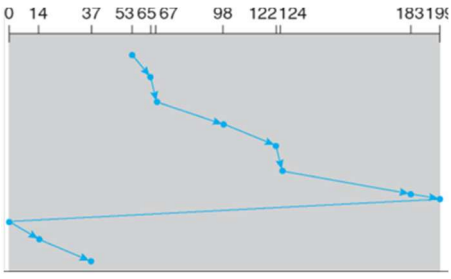


Tragittto toale di 640 cilindri

- **SCAN**: testina parte ad una delle estremità del disco e va verso l'altra raccogliendo tutti quelli che vanno in quella direzione



Spostamenti totali di 208 cilindri



- **CSCAN**: più uniforme percè vanno in un'unica direzione
    - Discesa veloce senza vedere nulla
    - Salita lenta fornendo i servizi richiesti
- **SSTF** → vado a quello più vicino a me

Starvation → impossibilità perpetua, da parte di un processo pronto all'esecuzione, di ottenere le risorse sia hardware sia software di cui necessita per essere eseguito.

Gestione errori

È possibile che ci siano informazioni che si guastino o falliscano per qualche motivo (avevi sctitto un valore e ne leggi un altro o un valore non stabile), dunque:

DETECTION: determinare se c'è stato un problema:

- **Bit di parità**
  - Se sbagliano un numero pari di bit → non se ne accorge
- **Checksum**
  - Usa aritmetica modulare per calcolare, salvare e comparare valori di parole di lunghezza fissa

Codice per correzione:

- Oltre a detection, possono fare correzione sfruttando rindondanza
  - Es: bit di parità sia per riga che per colonna
    - Se becchi incrocio RIGA-COLONNA errato → cambia valore

Storage Device Management:

Gestione del disco:

- Disco va **formattato**: nel passaggio di un disco **da** semplice sequenza di blocchi **a** file system organizzato, ci sono molte posizione intermedie
  - **Liv fisico** (Low-level): dividere il disco **in settori** che possono essere trattati come parti da leggere o scrivere in modo individuale
    - Ad ogni settore si possono aggiungewre delle informazioni per rilevamento, gestione, correzione dell'errore (~kb)
  - **Liv logico**

Per usare un disco per contenere file, il sistema operativo ha bisogno di **memorizzare la sua struttura dati:**

- Partizionando il disco in uno o più gruppi di cilindri, ognuno trattato come un **disco logico**
  - Su ognuno di questi o su gruppi, si potrà fare una formattazione logica → **filesystem**
- Esiste la possibilità di raggruppare i blocchi in cluster in grado di aumentare l'efficienza

Una delle partizioni è la **root partition**, contenente il S.O.
Altre partizioni possono ospitare altri s.o., parti di s.o o dati.
Le partizioni possono essere:

- Mounted/agganciate ad un S.O. in esecuzione alla fase di bootstrap
- Agganciate successivamente in modo manuale/automatico

Quando si fa **mount**, c'è una fase di verifica di consistenza → dati siano corretti.

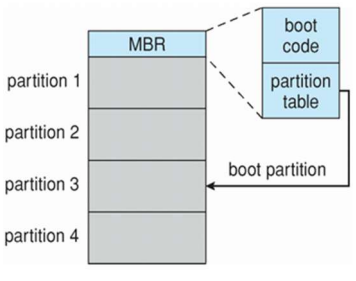
Nella **partizione di boot**, c'è un blocco di bout: blocco che va copiato direttamente in RAM per far partire il blocco di bootstrap:

- Carica da disco il kernel/BootLoader che carica il pezzo di disco da kernel

BootLoader: invece che mettere direttamente tutto il kernel, si fa partire un pezzo di codcie che carica il kernel in memoria RAM.

C'è una parte della RAM, detta **ROM** (contiene il BIOS), che fa in modo che parta l'operazione alla partenza: il **BIOS legge e carica in RAM il MBR** dove c'è un pezzo di codice detto bootLoader, partition table → infomraizoni su come far partitre il disco e edove trovare le informazioni da caricare in ram per far partire tutto.

Un disco è formattato fisicamente e logicamente ed è anche partizionato.



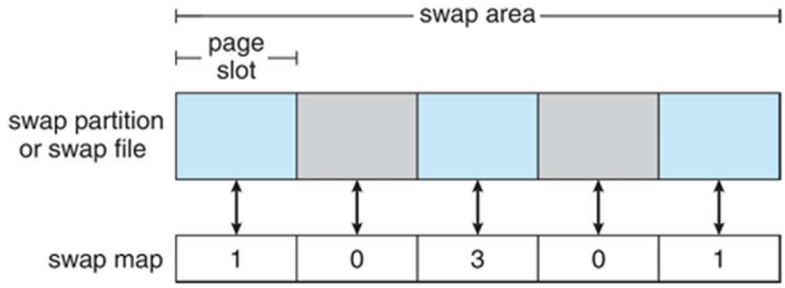
Booting from secondary storage in Windows

SwapSpaceManagement

Utilizzato per spostare interi processi (swapping) o pagine (paging) dalla DRAM alla memoria secondaria quando la DRAM non è abbastanza grande per tutti i processi.

Pezzo di memoria secondaria (+ lenti di DRAM), può essere vista come un vettore di blocchi (e non filesystem)

- Ho bisogno semplicemente di vedere un vettore di blocchi per vedere le strutture dati attraverso una swap map che indica quali sono i blocchi attivi e quali no
    - Cosa c'è e come viene usato quel blocco
- Dunque, lo swap non richiede file system



Ci sono delle organizzazioni di dischi che coinvolgono pesantemente la rete → **Storage Attachment:**

- **Host** attached →a bordo del computer che si utilizza
    - attaccabili in vari modi, BUS o altro
- Quando si parla di **Network** Attach Storage → in rete si vedono proprio dei dischi (D,C,F)
    - Attraverso una LAN, noi siamo in aula ma vediamo un disco fornito da un dipartimento
- Su **cloud**, non si vedono dei dischi ma dei servizi → livello di itnerfaccia più vicino all'applicazione
    - C'è client, fornitore di servizi e rete ma ci sono una serie di itnerfacce e API che forniscono varie cose

RAID: organizzazione particolare di dischi che mischia la ridondanza e la molteplicità dei dischi con le strategie viste per gestione errori

- Invece di mettere un disco, metto 2/4/8 dischi → miglioro affidabilità in base a ridondanza
  - **Striping**: non veramente ridondante
    - Se hai un'informazione e questa informazione vale 100, dividila in 4 info d 25 e mettila in 4 posti diversi → diminuisce la probabilità di perdere tutto
      - Se si guasta un disco, perdi ¼ dell'informazione
  - **Mirroring**: ridondanza piena
    - Duplica l'informazione
      - Memorizzo su un disco
      - Memorizzo sulla sua copia
    - Garantisce che, quando un disco si guasta, non perdi l'informazione perché c'è una sua copia

- RAID – redundant array of inexpensive disks
  - multiple disk drives provides reliability via redundancy
- Increases the mean time to failure
- Mean time to repair – exposure time when another failure could cause data loss
- Mean time to data loss based on above factors
- If mirrored disks fail independently, consider disk with 1300,000 mean time to failure and 10 hour mean time to repair
  - Mean time to data loss is 100, 000^2 / (2 * 10) = 500 * 10^6 hours, or 57,000 years!

MTTF/probabilità che fallisca il secondo

Mirrored RAID: 2 disks

$MTTF_{(1st\_fail)} = MTTF/2 = 100,000/2 = 50,000$ hours

$Prob_{(2nd\_fail\_during\_repair)} = MTTR/MTTF = 10/100,000 = 10^{-4}$

Mean time to data loss = $MTTF_{(1st\_fail+2nd\_fail\_during\_repair)}$

$MTTF_{(fail+fail\_during\_repair)} = MTTF_{(1st\_fail)} / Prob_{(2nd\_fail\_during\_repair)}$

$= MTTF^2/(2*MTTR) = 10^{10}/(2*10)$

$= 5*10^8$ hours

(hours) mean time to failure and 10 ho… time to repair

- Mean time to data loss is $100, 000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!

- Frequently combined with NVRAM to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively