

Lettura ottica con LASER  
Polarizzazione di campioni chimici pratico =  
lori. (CD-R / WORM)  
- DVD

Stampaggio (CD-RW)  
Polarizzazione di campioni chimici pratico =  
lori. (CD-R / WORM)

## DISCHI A STATO SOLIDO (memorie flash)

ACCESS	SSD	25 μsec	4KB READ	{ Riscontrabile un numero finito di volte.
		1.5 μsec + 250 μsec	WRITE	

Errore

7 dischi a stato solido, nonostante porti in movimento, consumano meno.

## ARCHITETTURA RAID

Articolo che si trova in Biblioteca (Non obbligatorio)

"Raid high performance, Reliable secondary storage".

RAID Redundant array of inexpensive disks.

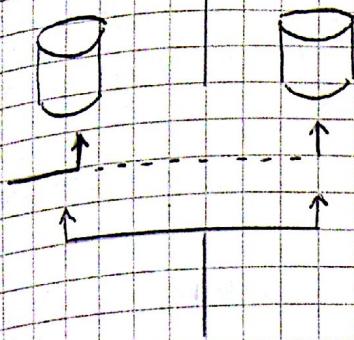
L'idea è di avere una maggiore affidabilità e una maggiore velocità. L'affidabilità è ottenuta grazie alla ricolombata.

INEXPENSIVE DISK significa che consente un aumento delle performance.

L'architettura più semplice è il RAID 1 (o mirroring).

Ha due dischi: 1) il principale 2) il disco mirror che è un back-up.

Quando leggo poco aumenta la velocità di lettura radiofrequenza fisica parallela la lettura.



la svantaggio è che paga un costo elevato per duplicare i dati.

Esistono tecniche meno costose:

1. **Striping:** ~~distribuisce~~ i dati su più dischi [Performance]



Il sistema operativo vede il RAID come un'unica unità.

Le unità di striping possono essere ~~a~~ il Byte oppure il Blocco.

Lo svantaggio dell'architettura con unità byte è che impiega la lettura tutti i dispositivi anche se aumenta la lettura parallela.

Con l'unità blocco invece la lettura solitamente è limitata ad un numero minore di blocchi.

2. **Ridondanza** [Affidabilità]

Poniamo usare una tecnica basata sul codice di parità.

Sono codici per controllare che i dati memorizzati non abbiano errori.



0

1

In questo contesto invece noi usiamo questi codici non per rilevare errori ma per correggerli.

XOR  
PARITÀ

0	1	P
0	0	0
0	1	1
1	0	1
1	1	0

Se si rompe il disco di parità si può ricomporre.

Lo stesso per il disco 2. Se si rompe faccio XOR fra disco 2 e disco parità.

Se si rompono 2 dischi non posso far nulla.

Quindi questo tipo di architettura è migliore del mirroring solo se abbiamo 2 drive con più di 2 drives.

## RAID

- 0 STRIPPING (Migliore performance ma non aggiunge affidabilità) Velocità lettura è la stessa
- 1 MIRRORING (Ridondanza massima) Velocità lettura solo lettura
- 2 è un tipo di codice di correzione - HAMMING CODE
- 3 STRIPPING con unità BYTE Parità su 1 drive
- 4 BLOCK STRIPPING. Parità su 2 drives
- 5 BLOCK STRIPPING + PARITÀ DISTRIBUITA

Il 5° è quello che troviamo ed è quello che maximizza il parallelismo fra diverse applicazioni.

Siamo di livello. sopra il disco c'è lo scheduler che riceve una serie di richieste e decide come servirle per ottimizzare il throughput.

OBIETTIVO: minimizzare il tempo di SEEK TOTALE.

OBIETTIVO SECONDARIO: avere un tempo di risposta equo.

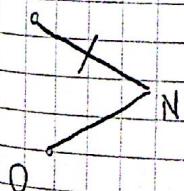
le possibili politiche sono:

1. FIFO: servirà in ordine d'arrivo. È equo ma il tempo di seek può non essere ottimo.

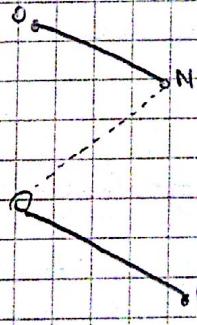
2. SSTF: Shortest seek time first. Prende dalla coda la richiesta più vicina a quella che ha appena servito. Minimizza il seek ma non è equo.

3. SCAN/C-SCAN  
L'idea dello SCAN è quella di spostare la testina da una parte all'altra del disco servendo le richieste che ~~sono~~ sono la testina su cui è presente la testina.

SCAN



C-SCAN



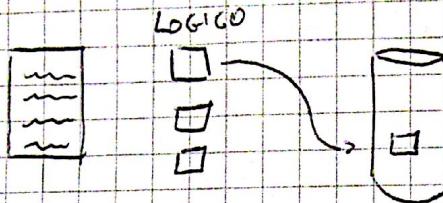
→ Va in una sola direzione. Non serve le richieste tornando indietro.

Tornando subito all'inizio dopo aver terminato il percorso è più equo perché torniamo a servire quelli che sono in attesa da più tempo.

La variazione LOOK/C-LOOK evita di spostare la testina all'indietro fino alla fine se non ci sono richieste per quelle tracce.

## ALLOCAZIONE DEI FILES

Abbiamo dei Blocco di allocazione. Il blocco è l'unità minima di scavo del so. Il blocco è un multiplo del settore.



Il file viene spezzato in blocchi.

Quello che percepisce l'utente è un file come una stringa di bytes.

Il S.O. deve determinare il blocco logico ( $N \cdot \text{BLOCKSIZE}$ ) + OFFSET

- determinare il blocco fisico
- leggere

Mappare il file su un blocco logico il quale dovrà essere mappato sul disco fisico.

Ma come allochiamo i file?

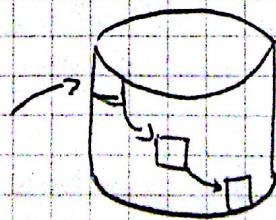
### PRO

1. Allocazione continua: file allocati sequentialmente, costo minimo per lettura sequenziale / random

### CONTRO

Estendere il file è difficile e costoso, causa anche frammentazione. Per risolvere il problema della frammentazione possiamo ricompattare.

2. Organizzazione linkata



Quando dobbiamo estendere il file prendiamo un blocco e lo linkiamo.

PRO: massimo utilizzo al 100% del disco.

CONTRO: Non riusciamo mantenere ordine

"i-blocks" ma soprattutto non possono fare una lettura random che si trasforma quindi in una lettura sequenziale.

### 3. INDEXED

Blocco indice

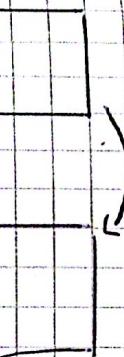


Vi è un blocco che contiene solo puntatori ai blocki. Se poi bisogna dell' N-esimo blocco, il puntatore dovrà N-uni gli elementi del blocco indice.

Se blocco indice ha dimensioni finite e quindi ha possibilità di contenere molti puntatori.

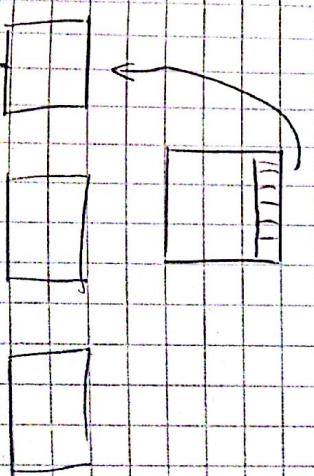
### OVERFLOW LINKED:

Una possibile soluzione per ovviare l'inabilità di essere presenti ad un altro index block.



### GERARCHICO

Va bene per file di grandi dimensioni.



→ DATI

### SCHEMI COMBINATI

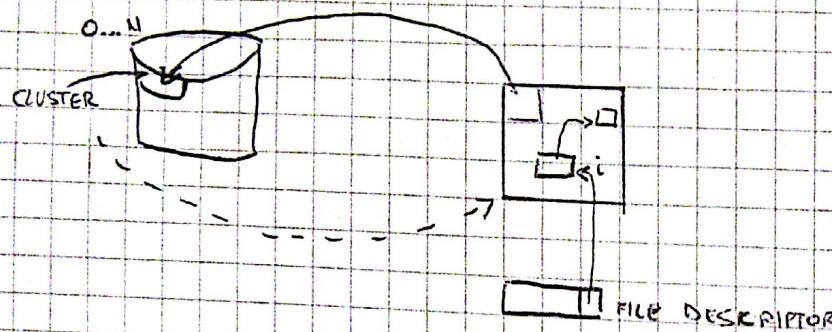
Unix ha un file descriptor con 32 puntatori diretti a block file.

- 1 puntatore ad un index block
- 1 puntatore ad un index block a 2 livelli

quindi abbiamo un compromesso ragionevole tra esigenze diverse.

## FAT

L'idea è di mantenere una tabella di allocazione in memoria.



Ed ogni cluster corrisponde un accesa nella FAT. Se bisogna accedere all'i-esimo elemento della FAT che referisce al cluster.

Tra i costi che abbiamo a volutare non vi sarà il tempo perché come abbiamo visto dipende da molti fattori ed è difficile calcolarlo.

Il nostro criterio di costo sarà il "numero di accesso alle pagine".

# PAGINE  
~ BLOCCO LOGICO  
2KB → 16 KB dimensione pagina.