



Dischi

Ci sono diversi tipi di dischi che sono:

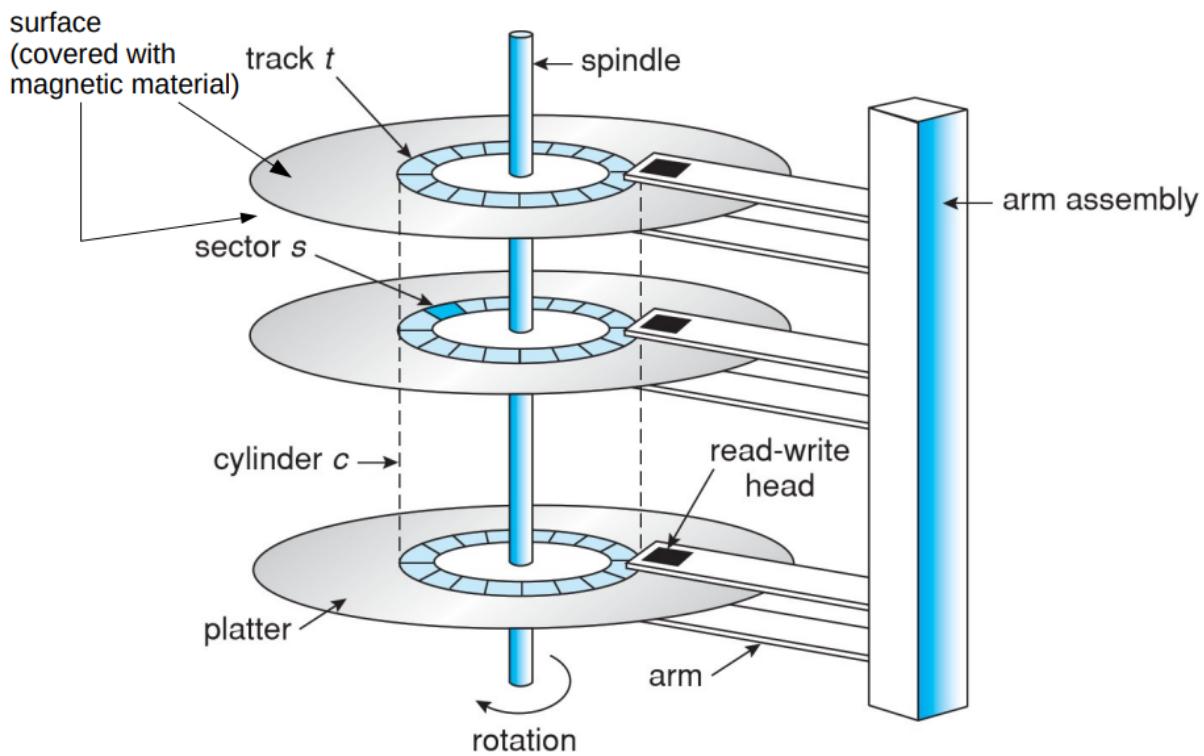
- **Dischi magnetici (HDD)**
 - Lettura e scrittura sono equamente uguali come velocità di esecuzione
- **SSD (Solid State Disks)**
 - Le operazioni di lettura sono più veloci di quelle di scrittura
- **Dischi ottici**
 - Avviene una scrittura ma multiple letture

Bisogna trovare un modo efficiente di gestire ognuna di queste, c'è un driver per ogni categoria di disco.

Dischi magnetici

Il principio di funzionamento è che abbiamo una **testina di lettura o scrittura** che è capace di magnetizzare (scrittura) o leggere lo stato di magnetizzazione, così facendo abbiamo le due operazioni più importanti.

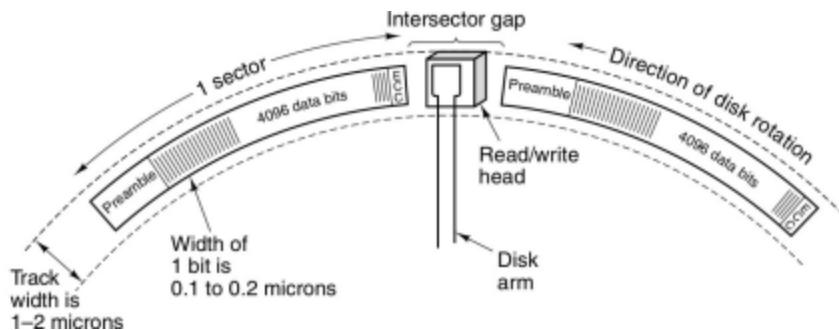
La testina ruota sul disco, ma possono esserci anche più testine per uno spazio più grande.



Il perno (**spindle**) abbiamo una serie di piatti con due facce, possono essere salvati i dati su entrambe le superfici (**surface**) di un piatto (**platter**) ogni piatto ha due facce (**head**). La testina gestisce entrambe, i dati sono salvati in punti precisi, ovvero in determinate circonferenze (**tracce**) ogni circonferenza è divisa in segmenti (**settori**). Viene chiamato **cilindro** quando tutti i bracci stanno leggendo lo stessa traccia.

Ogni **superficie** di un **piatto** è ottenuta dalla **divisione in tracce circolari**, la divisione è logica, ogni traccia è divisa in settori.

Ogni settore contiene una quantità di bit prestabilita ovvero 512 Byte, i settori sono separati da gaps dove non ci sono dati.



Ogni settore ha un **preambolo** che consente alla testina di capire **quale settore sta andando ad interrogare**, ogni settore è composto da **4096 bit** (512 Byte).

Abbiamo un codice **ECC** standard alla fine e viene testato se il codice calcolato nei dati è uguale al codice ECC.

L'ampiezza della traccia e quindi di 1 bit è tra 0.1 e 0.2 micron.

L'unità minima dei dischi magnetici in scrittura e il lettura è il **settore**, quindi possiamo solo calcolare sui multipli del settore. Quando dobbiamo cambiare un settore ma non completamente dobbiamo aggiornare il vecchio contenuto senza perdere dati presenti in precedenza.

Il controller del disco può leggere o scrivere più superfici quando si posiziona su un settore, è molto più conveniente scrivere più settori del cilindro e muovere il braccio per scrivere settori diversi della stessa traccia.

I trasferimenti dei dati è eseguito dal controller che è composto da due parti:

- **Host controller**: posizionato sulla motherboard.
- **Disk controller** posizionato sotto al disco.

Nei dischi moderni il disk controller fa la maggior parte del lavoro, mentre l'host controller invia comandi semplici. Il compenso però è **implementare tecniche di caching** ovvero portare in memoria dati che servono più spesso.

Per eseguire una operazione di I/O sul disco viene seguita questa sequenza di passi:

- Il **driver del disco scrive un comando nel command register dell'host controller**
- L'**host controller invia al disk controller i comandi** come impulsi elettrici
- La **testina ora si muove** e i dati sono **copiati dentro dei buffer** per controllare l'integrità dei dati appena presi.
- Tornano i dati all'host controller.

I dischi meccanici hanno limitazioni fisiche per le loro velocità, si sono pensate soluzioni per alleviare il problema, una prima soluzione è stata implementare più dischi per la scrittura e lettura in parallelo.

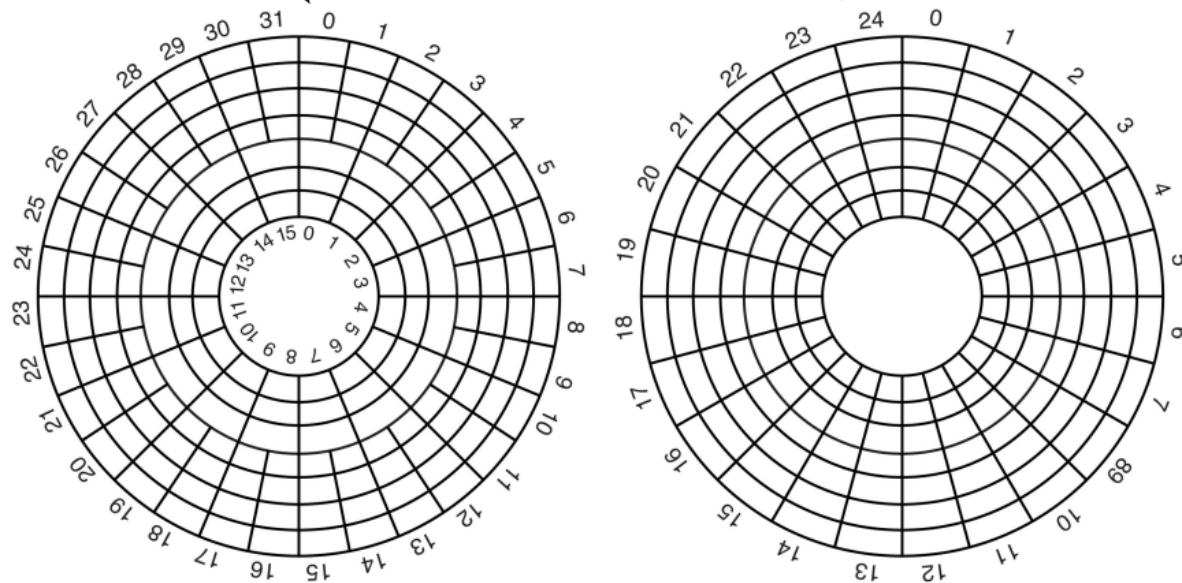
I **controller moderni** sono capaci di **fare operazioni di ricerca sovrapposte nel tempo**, quando un controller sta aspettando una per una ricerca , può inizializzare un'altra ricerca. Nel mentre i controller mentre cercano possono anche scrivere o leggere altri dati.

Ma i trasferimenti di dati tra il controller e la memoria centrale non si possono sovrapporre, questo è il nostro collo di bottiglia.

Geometria del disco

La **geometria del disco** serve per indicare lo **schema di posizionamento per la testina** sui diversi **settori**. Qualche anno fa ogni traccia aveva lo stesso numero di settori, non importa la distanza dal centro, ma questo non va bene per una spiegazione fisica ovvero più ci allontaniamo dal centro più il disco va veloce ma questa fa perdere spazio disponibile.

Nei **dischi moderni** la geometria **fisica e virtuale è diversa**, viene chiamata **Zone Bit Recording** anche conosciuta come zone multipla di registrazione ovvero quando dividiamo in più settori nello stessa traccia più ci allontaniamo dal centro.



Però è scomodo dover scrivere un driver che tenga conto di questo, per questo dobbiamo utilizzare una **geometria virtuale** che è come se avessimo **per ogni traccia lo stesso numero di settori**, questa traduzione è nascosta al SO, lui vede

la geometria virtuale. Quando richiede un **settore** il controller traduce la **geometria virtuale** in quella **fisica**.

Esempio di tabelle:

Zone number	Sectors per track	Cylinders per zone
(outer) 0	864	3201
1	844	3200
2	816	3400
3	806	3100
4	795	3100
5	768	3400
6	768	3450
7	725	3650
8	704	3700
9	672	3700
10	640	3700
11	603	3700
12	576	3707
13	528	3060

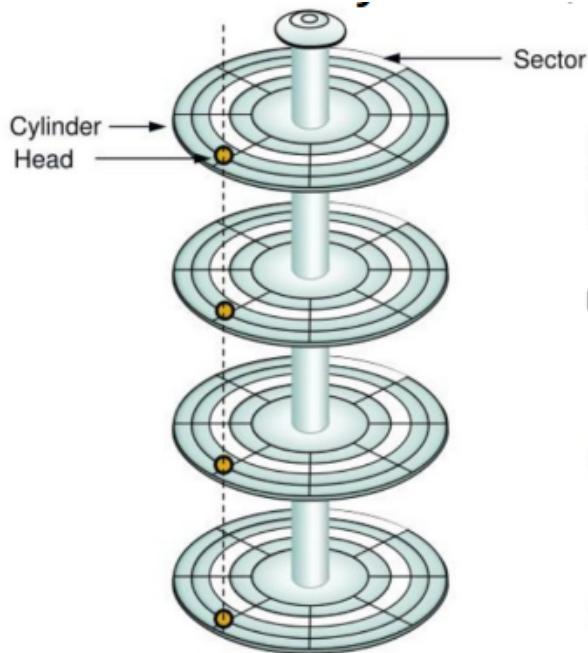
Ci sono 14 zone per disco, ogni zona ha un numero di settori, ci sono diversi cilindri per zona.

Ci sono due schemi di indirizzamento:

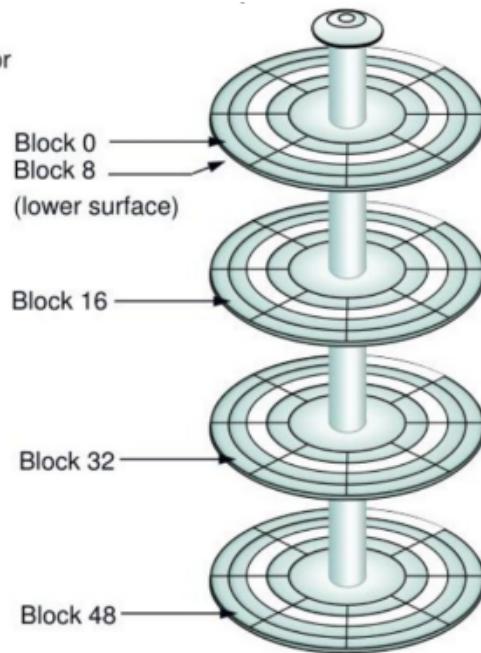
- **Cylinder Head Sector** (CHS): ogni settore (S) è identificato da una terna di valori, è su un cilindro (C), ed è su una delle due facce (H) quindi: (Cv, Hv, Sv) dove v sta per virtuale. sarà poi il controller a tradurre in indirizzi fisici. L'hard disk al tempo era 8GB la più grande indirizzabile. Ns è il numero di settori per traccia.
- **Logical Block Addressing** (LBA): ogni settore è numerato in modo progressivo è identificato da un solo numero, il controller andrà a tradurre per posizionare

la testina in maniera corretta.

Come si passa da un modo di indirizzare a un altro?



Physical Address = CHS



Logical Block Address = Block #

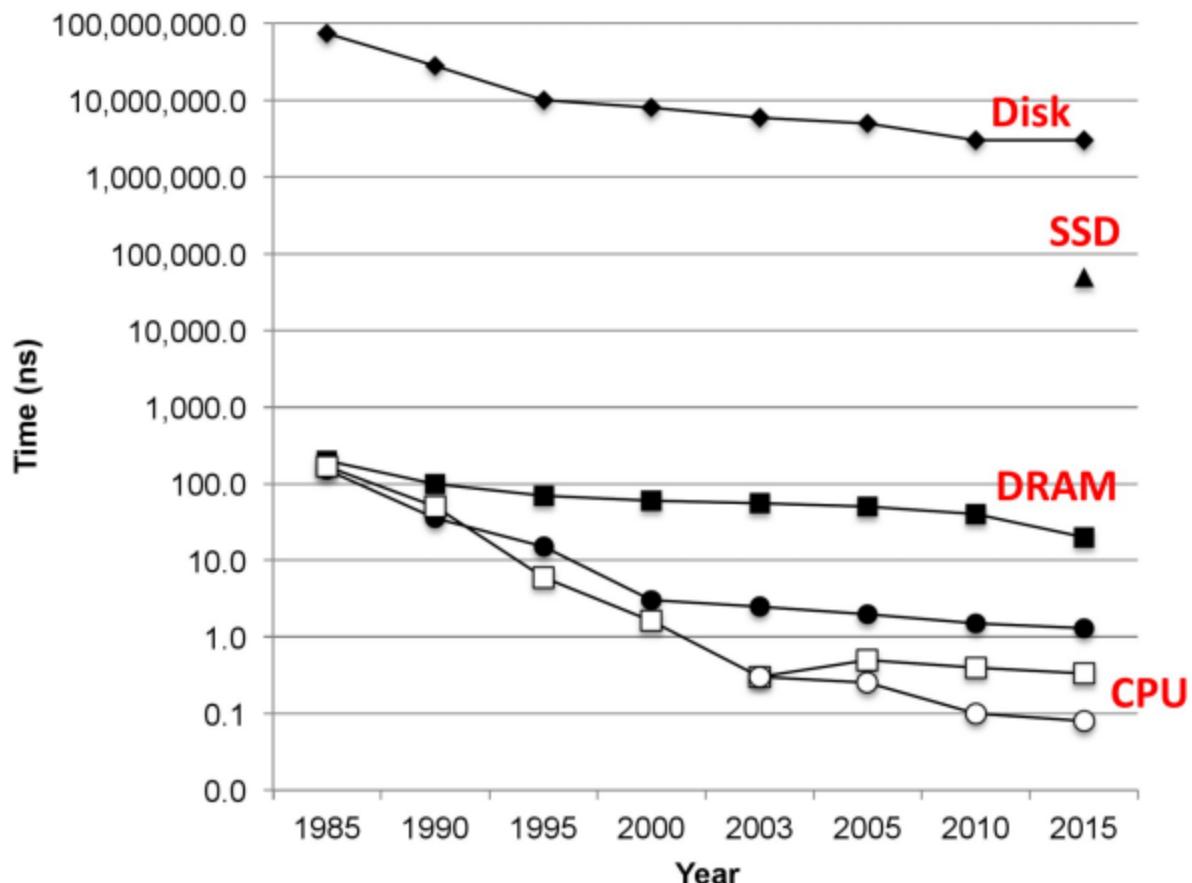
In LBA quando finiamo la traccia più esterna passiamo a quella più interna con la conta dei settori.

$$\begin{aligned} & (c, h, s) \rightarrow lba: \\ & \quad lba = (c \times N_h + h) \times N_s + (s - 1) \\ & lba \rightarrow (c, h, s): \\ & \quad c = lba \text{ div } (N_h \times N_s) \\ & \quad h = (lba \text{ div } N_s) \bmod N_h \\ & \quad s = (lba \bmod N_s) + 1 \end{aligned}$$

Per convertire da CHS a LBA ci pensa il disk controller, il driver non deve far calcoli.

LBA value	CHS tuple (c,h,s)
0	(0 , 0 , 1)
1	(0 , 0 , 2)
62	(0 , 0 , 63)
63	(0 , 1 , 1)
1007	(0 , 15 , 63)
1008	(1 , 0 , 1)
32256	(32 , 0 , 1)
66058272	(65534 , 0 , 1)
66059279	(65534 , 15 , 63)

Il problema che si è venuto a creare nel corso del tempo è un gap tra la velocità della CPU e quella di accesso alla memoria come vediamo da grafico riportato:



Le latenze di accesso sono per colpa di vari fattori come il tempo di ricerca (ovvero spostare il braccio meccanico nel giusto cilindro) il tempo di rotazione del disco e l'effettivo tempo di trasferimento.

- **Tempo posizionamento:** tempo di ricerca + latenza rotazione
- **Tempo di accesso per 1 settore:** tempo posizionamento + tempo di trasferimento
- **Tempo massimo latenza rotazione:** $1 / \text{velocità di rotazione}$
- **Tempo medio latenza rotazione:** tempo massimo latenza rotazione / 2
- **Tempo trasferimento medio:** tempo massimo latenza rotazione / numero medio di settori
- **Tempo accesso medio:** tempo medio di ricerca + tempo medio latenza rotazione + tempo trasferimento medio

Tecnologia RAID

La tecnologia RAID (**Redundant Arrays of Independent Disks**) serve per ottimizzare i tempi di accesso con svariati accorgimenti:

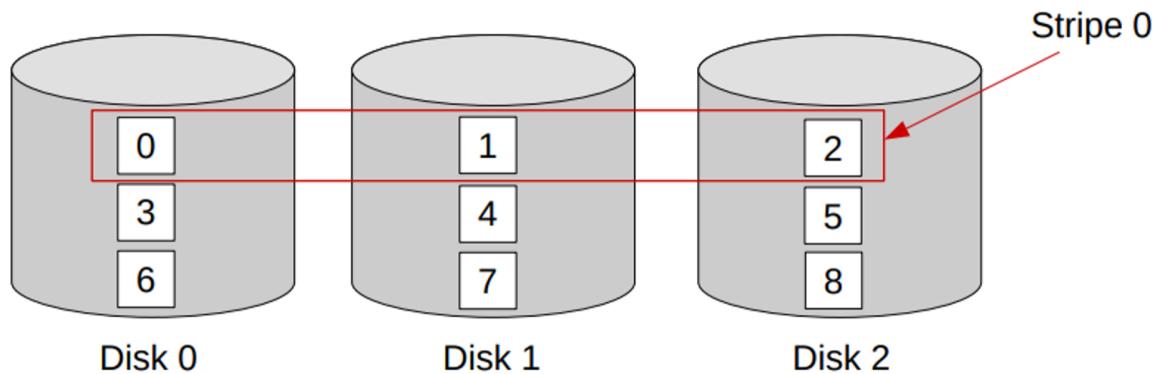
- Array di più dischi
- Controller integrato per gestire I/O
- Memoria volatile come buffer dei blocchi

Esteriormente appare come un disco unico logico, sarà quindi compito del controller di tradurre da indirizzo logico fornito dal SO a indirizzo fisico interno al RAID.

La tecnologia RAID serve per integrare il parallelismo e la ridondanza per aver i dati accessibili in modo più veloce e sicuro.

Parallelismo con i RAID

Con la tecnologia RAID è stato possibile introdurre il parallelismo grazie alla suddivisione dell'array di dischi chiamata **data striping** (strisce di dati) è una tecnica che combinata con più dischi crea le **stripe**.



Per calcolare in quale disco ci troviamo dobbiamo calcolare: **strip modulo numero di blocchi**, mentre per la **stripe**: **strip diviso numero blocchi**.

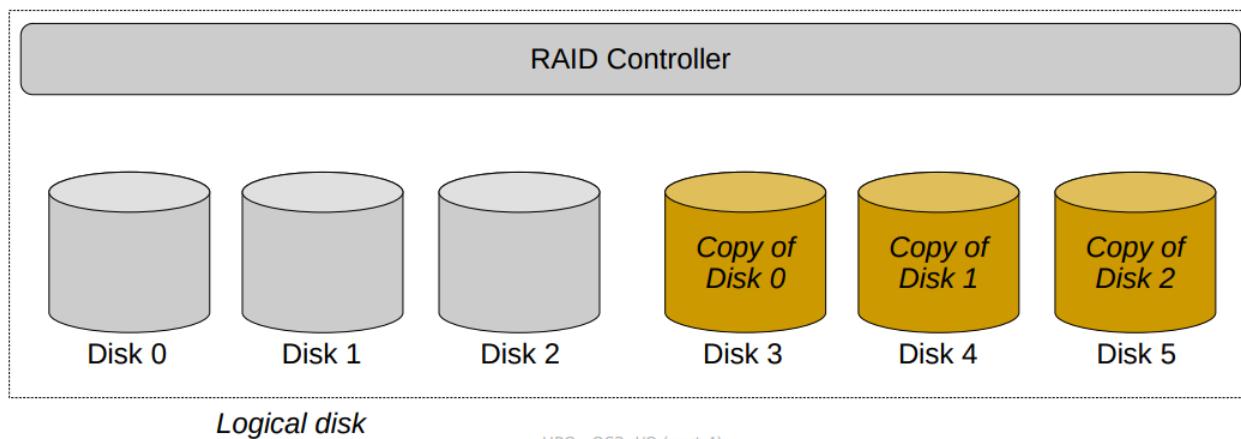
- Strip 1 → Disco = 1 modulo 3 = 1, Stripe = 1 diviso 3 = 0
- Strip 4 → Disco = 4 modulo 3 = 1, Stripe 4 diviso 3 = 1

Lo striping è gestito dal controller del blocco RAID, per rendere trasparente questo calcolo. Ci sono diverse modalità di striping come dimensione: bit, byte e blocchi, ad esempio bit-level lo striping avviene in modo che in ogni stripe ci sia 1 bit del disco.

La cosa importante da tenere a mente è che lo striping permette di leggere i **dati** in diversi dischi come se fossero strisce di dati contigue.

Affidabilità con i RAID

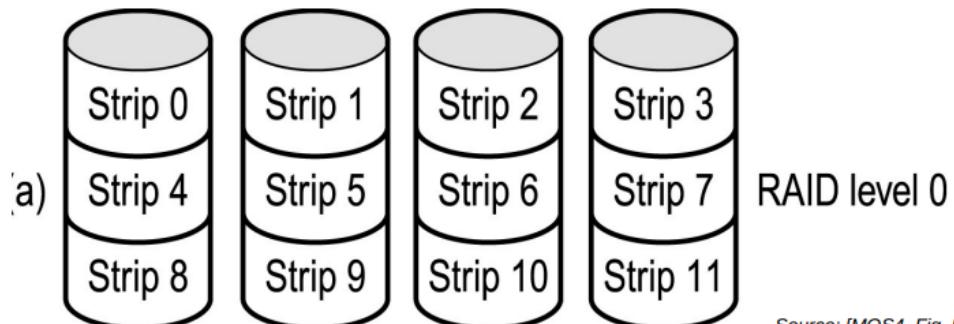
Alcune informazioni sono salvate in più per permettere di avere comunque le informazioni nonostante un disco possa smettere di funzionare. Per questo si fa una copia di ogni disco.



Non è certamente la soluzione più economica. Se anche il clone smette di funzionare allo i dati sono persi per sempre. Si calcola il **MTTR** ovvero **Mean Time To Repair**, il tempo che ci mette in media a rimpiazzare un disco e prendere i dati dal nuovo.

Tutto quello che viene scritto su un disco viene scritto anche sul suo clone, le letture avvengono anche qui in parallelo su più dischi. Questa soluzione è chiamata **mirroring**.

RAID 0

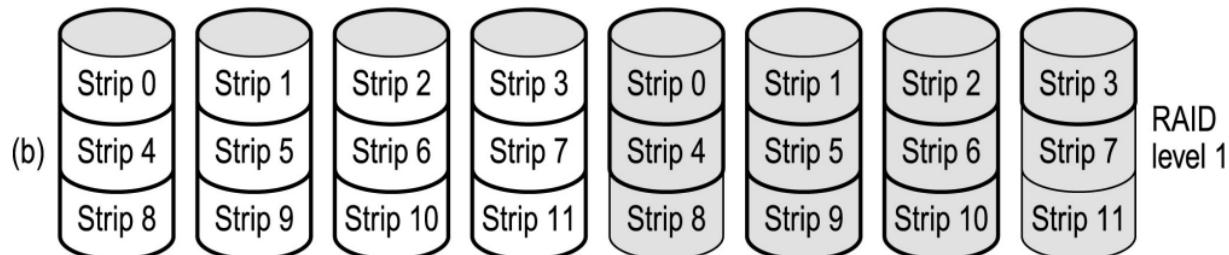


Source: IMOS4. Fia. 5-201

Il RAID 0 ha uno **striping a blocchi**, richiede dischi per $c_{\text{disks}} \times n$ e non supporta il danneggiamento di un disco.

- **Vantaggi:** alte performance a livello di scritture e letture, alta capacità
- **Svantaggi:** assenza di ridondanza, non permette sistemi con richieste piccole (data sempre maggiore della strip size)

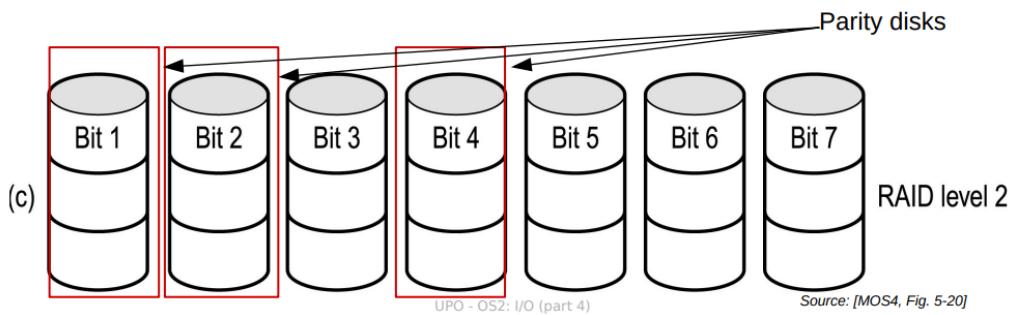
RAID 1



Supporta il mirroring, i **dati** sono in **stripe** e ogni **stripe** è **mirrored**. Numero di dischi richiesti: $n \times n$ dove n è il numero di dischi. Tollerà la perdita di un disco

- **Vantaggi:** permette di controllare le perdite, recovery di un errore, le richieste di lettura sono servite il doppio più velocemente.
- **Svantaggi:** costoso, le performance sono di poco migliorate, poca capacità confrontato al RAID 0 dato che si ha metà dello spazio a parità di dischi.

RAID 2



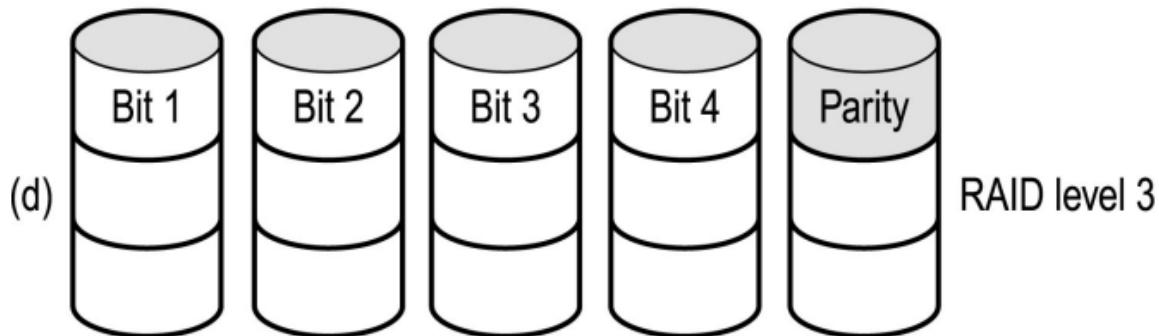
Lo striping avviene a livello bit, viene creato un **ECC (Error Correction Code)** per ogni stripe che è salvato in un disco diverso da quello dei dati per poterlo usare nel caso il disco da cui provengono i dati sia corrotto. Serve per correggere dati corrotti dentro una stripe.

Ci sono n dischi e m dischi con dentro i codici ECC, dove $n + m$ è minore di $2n$.

Le operazioni di lettura vanno a interrogare pochi dischi, ovvero dove risiedono i dati della stripe, mentre le scritture interessano tutti i dischi dato che bisogna anche salvare le ECC di ogni dato. Anche il salvataggio di un singolo bit comporta il cambiamento di un intero ECC.

- **Vantaggi:** molto affidabile grazie a ECC, ottime performance di lettura
- **Svantaggi:** molto complicato da implementare e gestire, carico di lavoro molto pesante al controller.

RAID 3



Lo striping avviene a livello di bit con il bit di parità alla fine dello stripe (nell'ultimo disco quindi). Il numero di dischi necessario è $n + 1$ perché l'ultimo è riservato al

bit di parità:

- **Bit di parità pari:** se il numero di bit a 1 è pari viene impostato a 0 altrimenti se è dispari a 1.
- **Bit di parità dispari:** se il numero di bit a 1 è pari si imposta a 1, altrimenti se è dispari a 0.

Esempio pratico:

- Stripe: {1, 0, 1, 1}, con il bit di **parità pari** mettiamo **1**, con il bit di **parità dispari** mettiamo **0**.
 - Parità pari → 1, 0, 1, 1, 1
 - Parità dispari → 1, 0, 1, 1, 0

Se viene preso male il dato in un disco e quindi con il bit di parità pari abbiamo 1, 0, **0**, 1, 1 possiamo capire che c'è stato un errore nel prendere i dati.

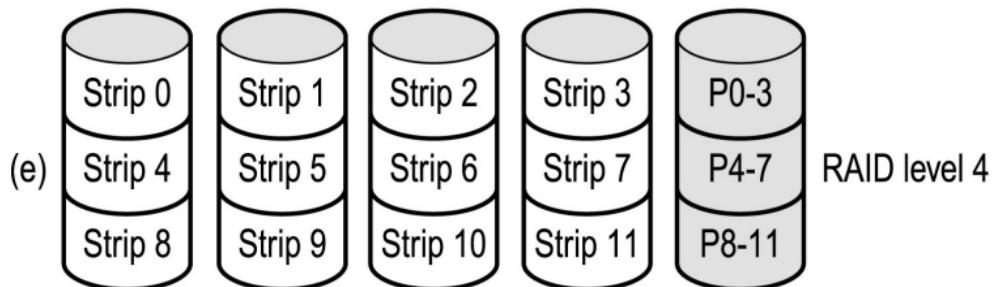
Per capire meglio, il **bit** che viene **aggiunto** mette la **condizione vera sul suo tipo**: il bit di parità pari viene messo a 1 quando con la sua aggiunta il numero di 1 diventa effettivamente pari, altrimenti se lo sono già si mette a 0.

- Esempio 1, 0, 1, 0, 1 in questo caso il bit di parità sarà 1.

Lo stesso discorso vale per il bit di disparità.

- **Vantaggi:** molto affidabile grazie al bit di parità, ottime performance grazie allo striping
- **Svantaggi:** molto complicato da implementare e gestire, carico di lavoro molto pesante al controller.

RAID 4



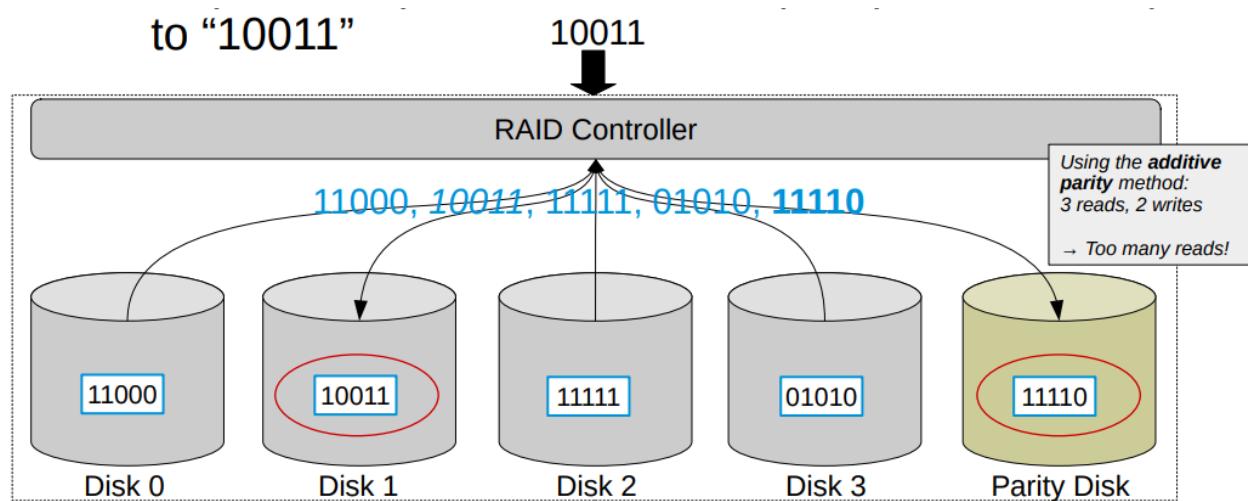
Lo striping avviene a blocchi + un blocco di parità, vengono richiesti $n + 1$ dischi per completare il raid. Non si discosta troppo dal RAID 3.

Blocco di parità: XOR tra le strips della stessa stripe. Ci sono due metodi ovvero parità **additiva** (legge dati vecchi e dati nuovi creando un nuovo blocco di parità) e **sottrattiva** (sottrae i dati vecchi con il vecchio blocco di parità con il nuovo dato).

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

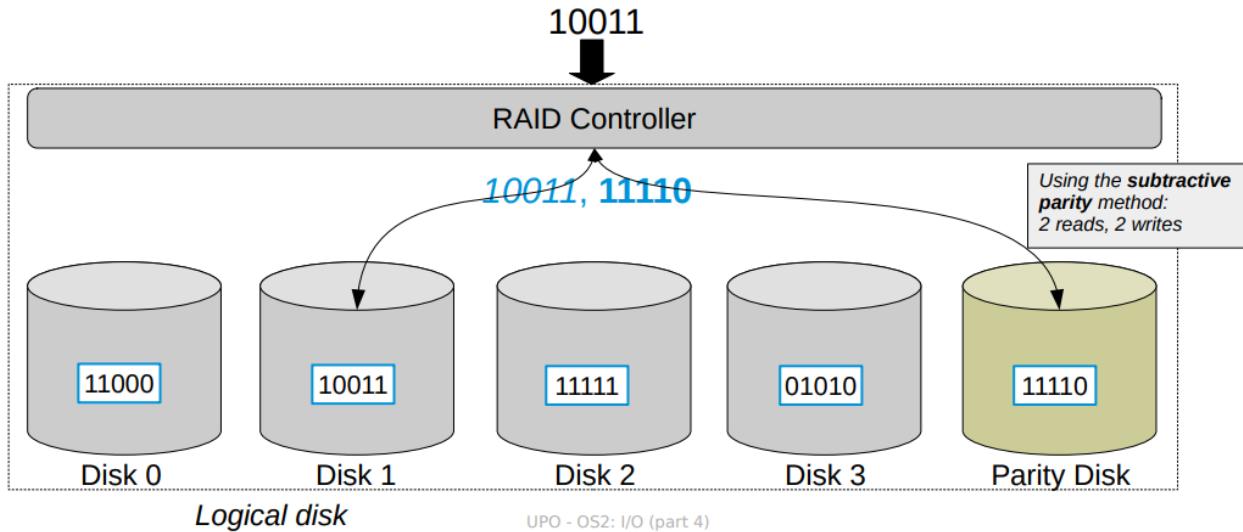
Il **problema** delle **small-write**: ovvero quando la scrittura è minore della lunghezza della stripe, quindi modifichiamo solamente una strip. al contrario abbiamo poi la **large-write** ovvero quando scriviamo una stripe intera.

Esempio di **scrittura** di 00101 con **parità additiva**:



Stiamo usando una parità additiva quindi facciamo 3 letture e 2 scritture dato che dobbiamo cambiare una strip e poi il blocco di parità che prende i dati vecchi e i dati nuovi e li somma in XOR.

Esempio di **scrittura** di 00101 con **parità sottrattiva**:



Stiamo usando una parità sottrattiva, prima legge i vecchi dati e il vecchio blocco di parità e poi va a sottrarre con i nuovi dati in arrivo. Il risultato non cambia, cambia solo il numero di scritture e letture.

Come funziona la parità sottrattiva?

Le proprietà dello XOR ha un elemento neutro ovvero 0, ad esempio:

- $A \oplus A = 0$
- $0 \oplus A = 0$

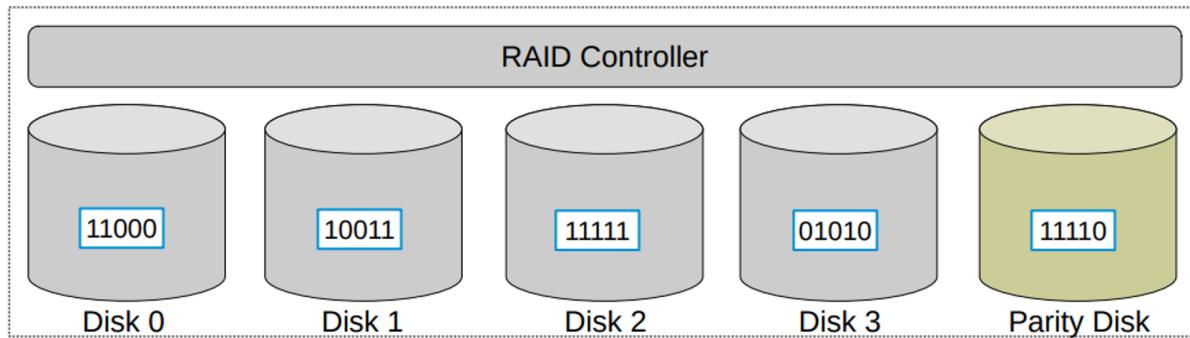
Vediamo il perchè la sottrattiva funziona:

- **NewParityStrip** = Strip0 \oplus Strip1 \oplus Strip2 \oplus Strip3 (**parità additiva** come base)
- **NewParityStrip** = Strip0 \oplus Strip1 \oplus Strip2 \oplus Strip3 \oplus OldStrip1 \oplus OldStrip1 (per le proprietà dello XOR diciamo che $OldStrip1 \oplus OldStrip1 = 0$ quindi possiamo metterla, sappiamo che però $Strip0 \oplus OldStrip1 \oplus Strip2 \oplus Strip3$ è uguale a $OldParity$)
- **NewParityStrip** = $OldParity \oplus OldStrip1 \oplus NewStrip1$ (qui abbiamo sostituito con la $OldParity$ che è calcolata avendo i vecchi dati, ci rimangono quindi la vecchia strip1 la nuova strip1 e il vecchio blocco di parità)

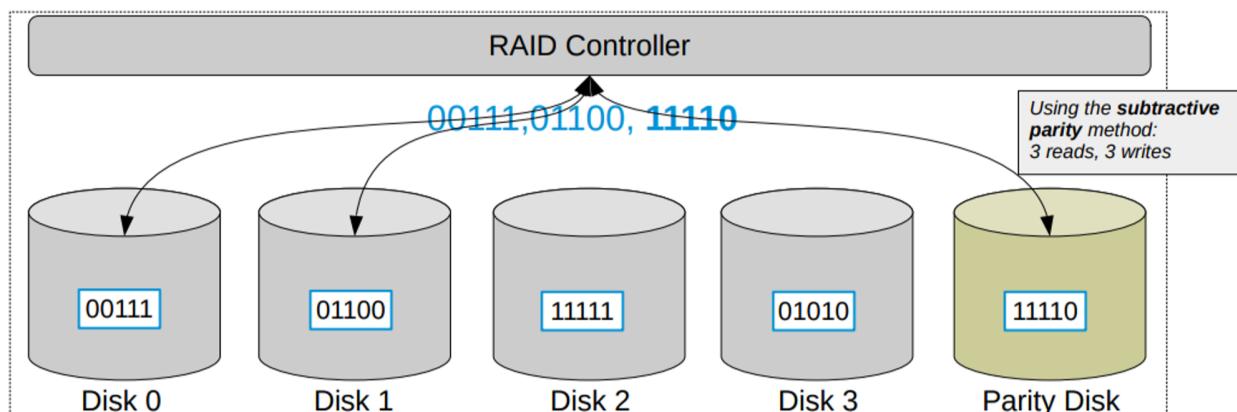
Così abbiamo dimostrato che la sottrattiva è creata partendo dalla additiva. Si dimostra sempre allo stesso modo.

Durante il compito di esame ci si aspetta il calcolo formale senza valori in bit, ma solo con i nomi dei blocchi per dimostrare che si ottengono gli stessi risultati.

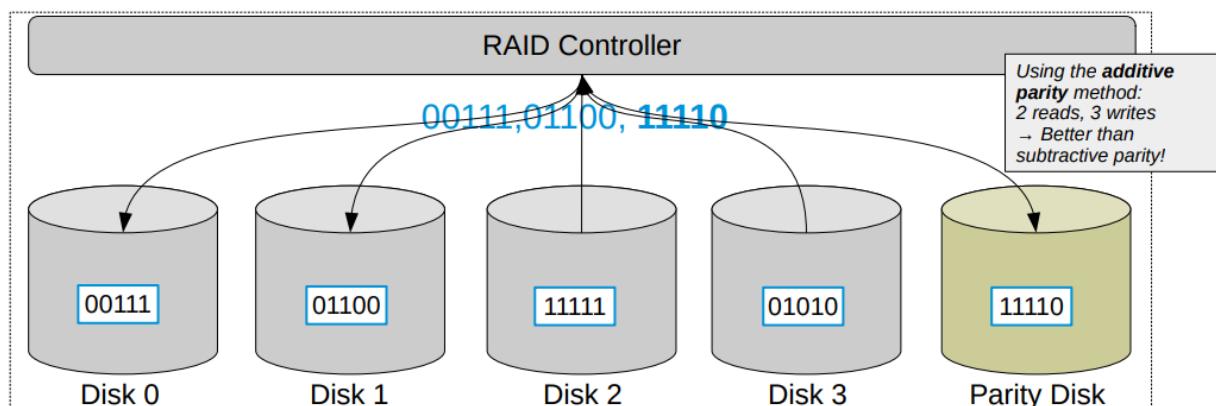
Come scegliamo tra proprietà additiva e sottrattiva



- Vogliamo sostituire 11000→00111 (strip 0) and 10011→01100 (strip 1) prima con il metodo sottrattivo:



- Abbiamo **3 letture** ovvero il vecchio blocco di parità, i due vecchi blocchi e 3 scritture ovvero nuovo blocco di parità e due nuovi blocchi.



- Qui abbiamo invece 2 letture ovvero delle vecchie strip e 3 scritture ovvero dei nuovi dati e del blocco di parità.

Regola generale

Considerando il numero di I/O (RD e WR) nel RAID 4, per scrivere B blocchi (strip) in N dischi con il numero $B < N$ (per le small-write altrimenti si usano le large-write) possiamo calcolare che la migliore è quella che mette meno operazioni di lettura.

- se il numero di dati da leggere è maggiore di B → **additiva**.
- altrimenti **sottrattiva**

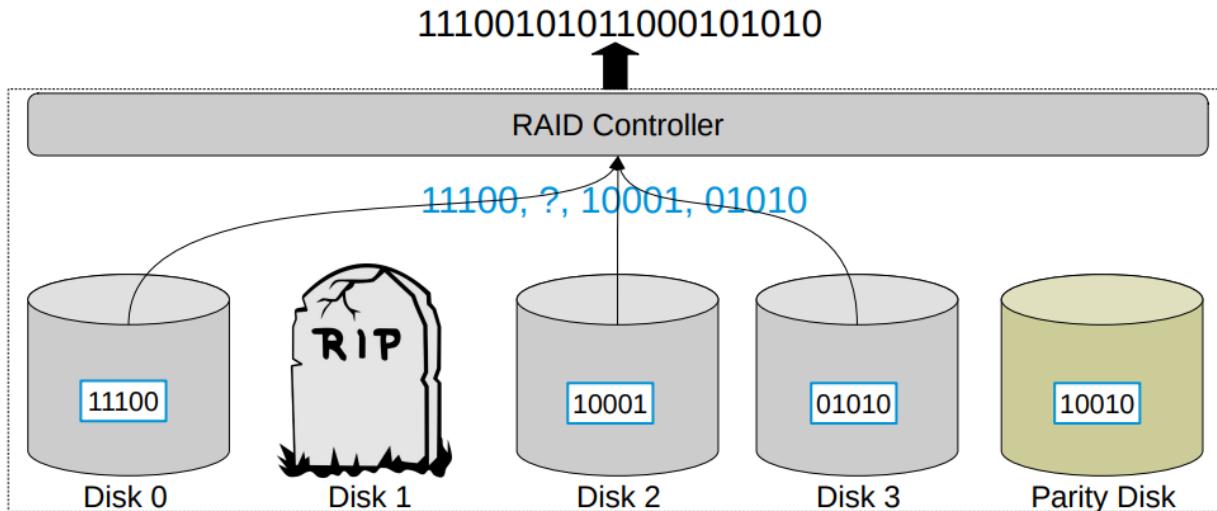
Esiste un punto di incontro tra le due: additiva e sottrattiva

- $N - 1 - B$ (additiva) = $B + 1$ (sottrattiva)
- $2B = N - 2$
- $B = \frac{N-2}{2}$

Calcolando B possiamo capire cosa usare, proprio come fa il controller.

Correzione dell'errore

Mettendo XOR bit a bit possiamo ricostruire il blocco che non c'è più infatti se mettiamo in XOR otteniamo 10101 ovvero la strip corrotta. Possiamo usare solo la parità additiva perché quella sottrattiva prevede di avere la vecchia strip e in questo caso non c'è.



Come funziona la correzione dell'errore

ParityStrip = Strip0 \oplus Strip1 \oplus Strip2 \oplus Strip3 (per la proprietà matematica di base possiamo aggiunger a ogni lato i valori di ParityStrip \oplus Strip1)

ParityStrip \oplus ParityStrip \oplus Strip1 = Strip0 \oplus Strip1 \oplus Strip2 \oplus Strip3 \oplus ParityStrip \oplus Strip1 (quando due valori sono uguali in XOR valgono 0 quindi abbiamo)

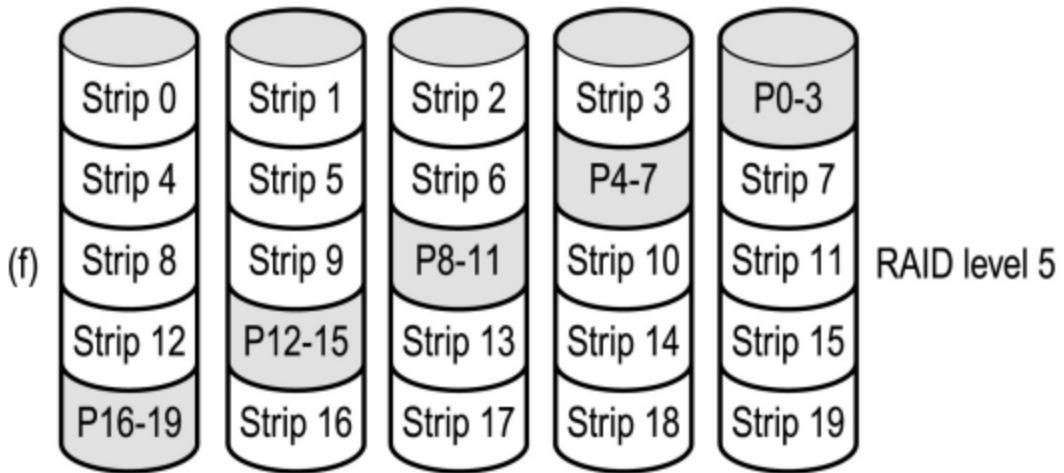
Strip1 = Strip0 \oplus Strip2 \oplus Strip3 \oplus ParityStrip



I punti di forza del RAID 4 quindi sono la alta affidabilità su un disco nel caso di suo fallimento, e ha ottime performance in larghe scritture. I problemi arrivano quando ci sono piccole scritture per quello che abbiamo visto e il disco di parità fa da **bottleneck** perchè non possiamo scrivere in parallelo ma solo in serializzazione e questo fa perdere di molto le performance.

RAID 5

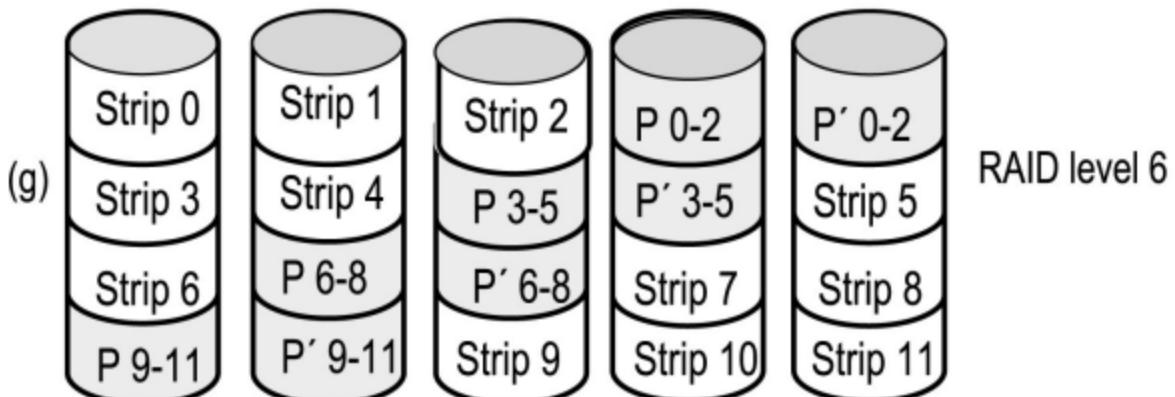
Il problema del collo di bottiglia nel disco di parità è stato molto sentito, il RAID 5 integra tutto del RAID 4, quello che cambia è che i blocchi di parità sono distribuiti tra tutti i blocchi. Per ogni stripe calcoliamo comunque il suo blocco di parità ma non viene messo in un determinato disco ma finiscono tutti in altri blocchi con schema diagonale.



Essendo che non sono tutti in un blocco possiamo accedere a più dischi senza la serializzazione, il problema è che abbiamo un minimo di due letture e due scritture. Questo perchè 1 lettura e 1 scrittura è per il nuovo strip e 1 lettura e 1 scrittura invece è per il blocco di parità come nel RAID 4.

RAID 6

Sempre striping a livello di blocchi, si usano due blocchi di parità per un solo stripe. Si chiamano blocchi di Reed-Solomon i quali sono molto più complicati dello XOR. Quindi servono $n + 2$ dischi e vengono tollerati i malfunzionamenti di 2 dischi.





Ha buone performance ma non come RAID 5, questo perchè deve scrivere due blocchi di parità quindi in un RAID con n dischi noi possiamo usarne solo n - 2. Ha un'alta affidabilità chiaramente avendo due dischi di parità, la lettura è comparabile al RAID 5 mentre la scrittura no per la doppia strip di parità.

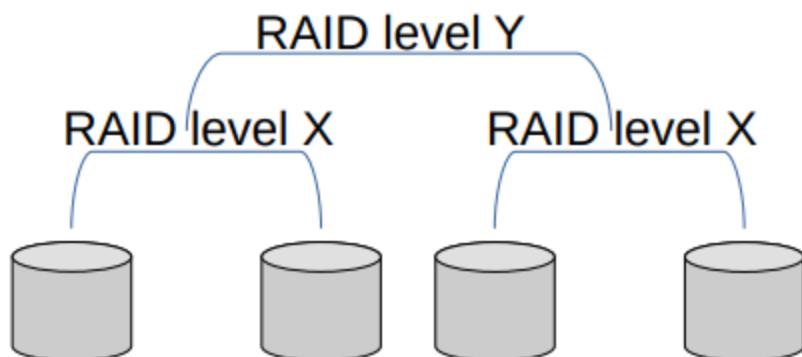
Non esiste una soluzione ottimale a seconda delle necessita dobbiamo usare RAID 0, 1, 5, 6. I RAID 2, 3 e 4 sono usciti di scena da anni perchè non molto usati.

RAID Ibrido (Nested RAID)

Possiamo tuttavia combinare più livello di RAID assieme in modo da creare nuove tipologie, la gerarchia acquisisce le caratteristiche dei singoli.

Il modo più comune di denotare la gerarchia è:

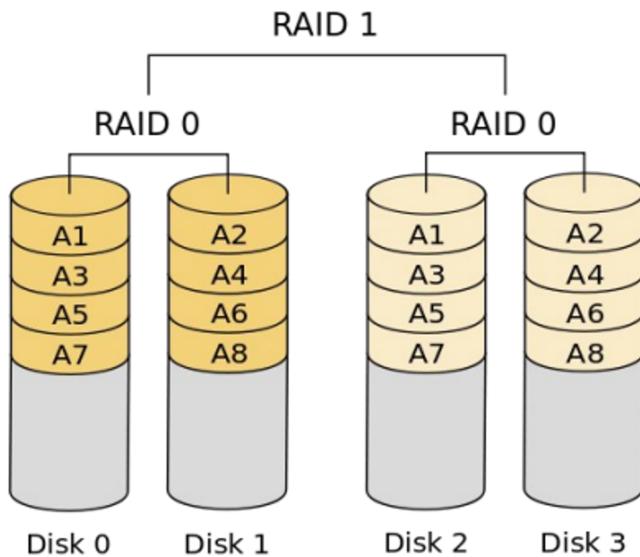
- Il RAID più a **sinistra** è il livello più **basso**
- Il RAID più a **destra** è il livello più **alto**



RAID 0+1

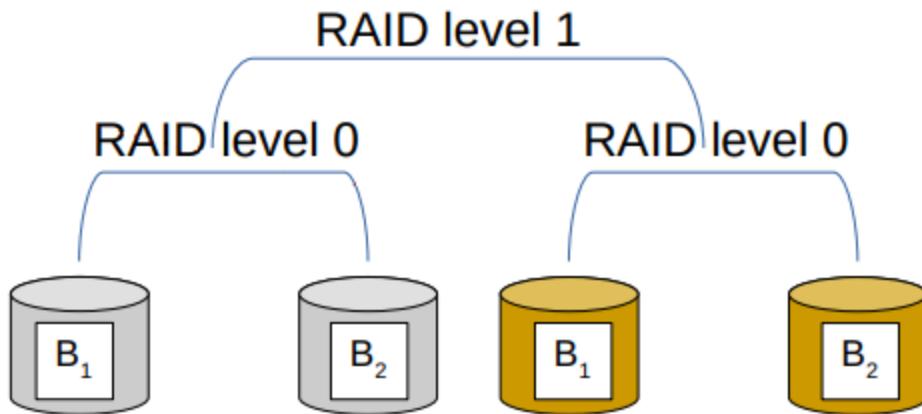
Mirroring delle stripes: abbiamo un set di n dischi con tecnica di stripe (RAID 0), ogni stripe è mirrored (RAID 1) in un'altra sequenza di dischi:

RAID 0+1



Ha alte performance in lettura, richieste 2 scritture per ogni stripe questo perchè viene salvata in due dischi una singola stripe. Questa soluzione è molto costosa.

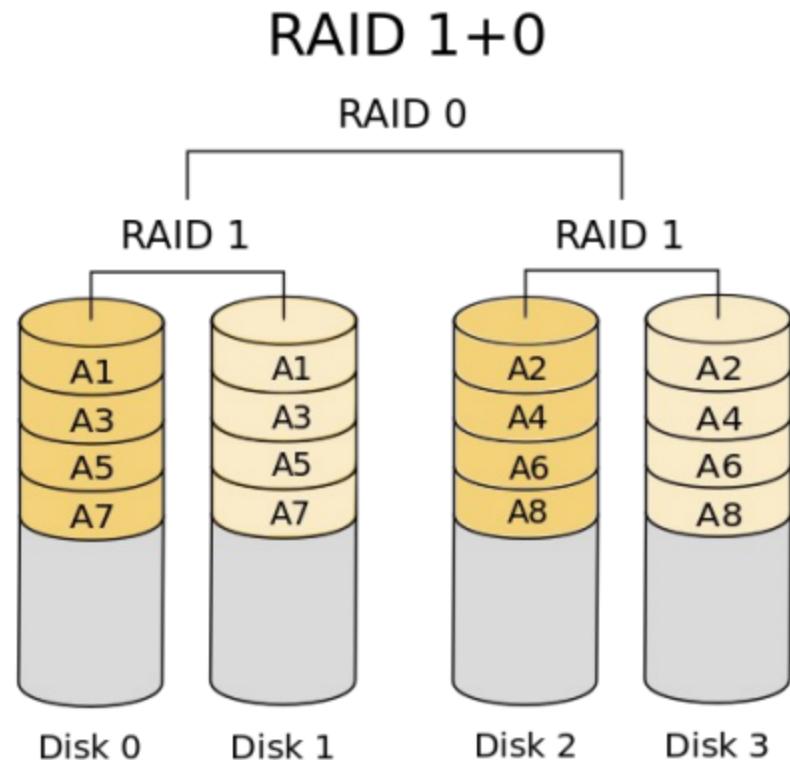
Le letture posso esser fatte su entrambi i set, ma una volta iniziata non possiamo usare altri dischi in altri set, a meno che non ci siano guasti



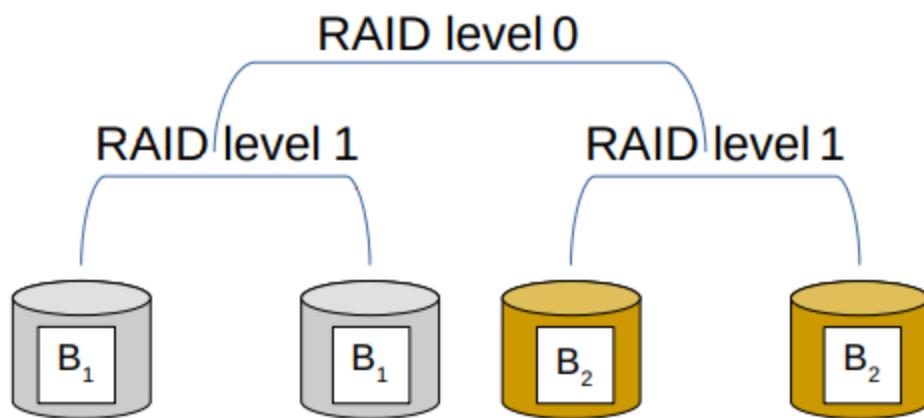
Quando effettuiamo quindi la lettura di <B1, B2> possiamo attingere o al primo o al secondo set, non possiamo però prendere un po' da un set e un po' da un altro. In caso di crash di un singolo disco allora l'intero set diventa inutilizzabile.

RAID 1+0

Striping del mirroring: abbiamo un set di n dischi che sono mirrored (RAID 1) in coppie formando un set e sono stripe (RAID 0) tra i set diversi.



Alto costo ma è anche molto affidabile, possiamo leggere uno dei due dischi di un set, se un disco ha un errore e non possiamo recuperare i dati da esso possiamo usare l'altro nello stesso set.



Implementazione del RAID

- **Implementazione del controller:** parte hardware ed è un circuito da collegare e parte software nel SO
- **Hot Spare Disk:** disco che serve quando c'è un altro disco rotto, dove metto dati per evitare un intervento umano.

Svolgere esercizi da slide 273 a 279.

Low Level Formatting

Si definisce la geometria fisica del disco (settori, traccia) prima di essere usati come tali i dischi devono ricevere una formattazione di basso livello.

Ogni segmento è suddiviso in tre parti:

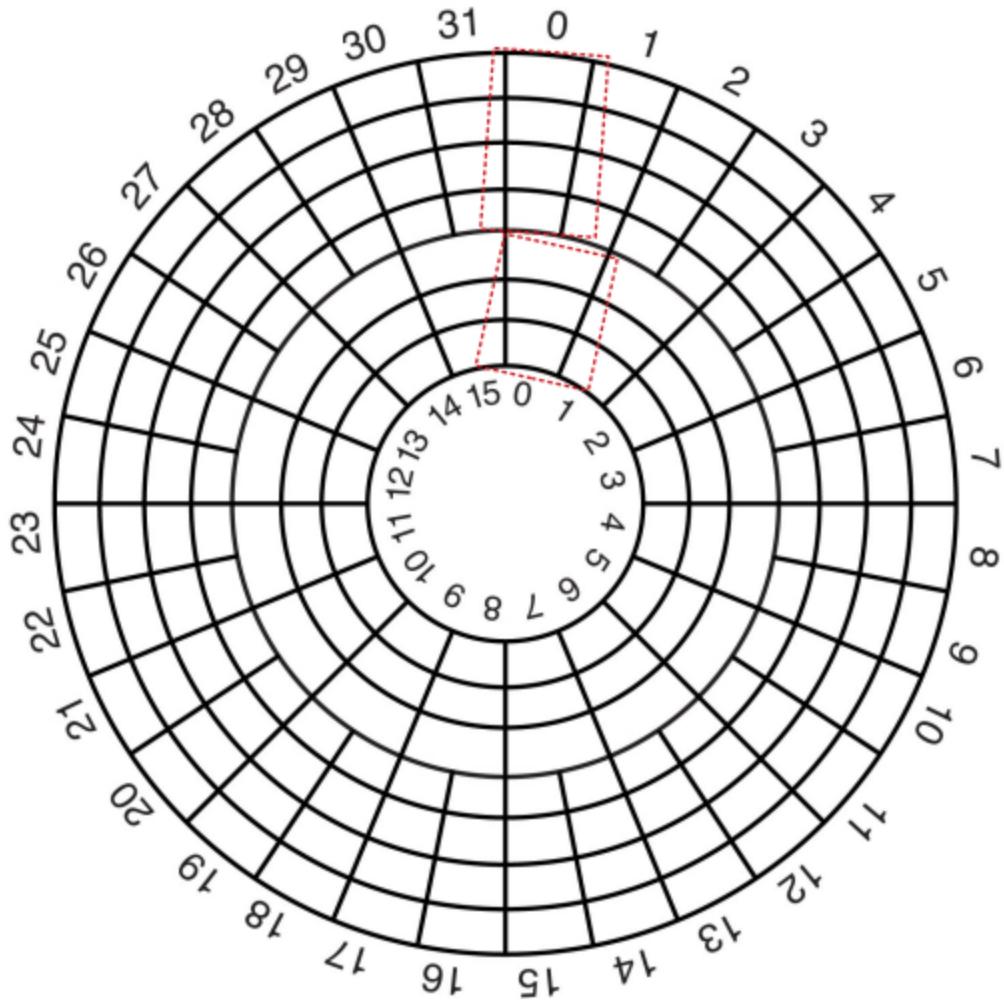
- **Preambolo:** identifica l'inizio del settore, ha un pattern di bit che permette all'hardware di capire che è iniziato un nuovo settore.
- **Data:** dove si salvano effettivamente i dati, solitamente la dimensione di questo blocco è 512 Byte di dimensione.
- **ECC:** salva un codice per poter correggere eventuali errori, molti dischi usano 16 Byte di dimensione.

Abbiamo le due principali istruzioni, le uniche che si possono fare ovvero **write** e **read** in un settore

- **Write:** quando scriviamo dei dati, il controller del disco aggiorna anche l'ECC, il valore è calcolato da tutti i dati dal campo data.
- **Read:** quando leggiamo dei dati il controller del disco va a verificare se i dati non sono corrotti ricalcolando l'ECC e confrontandolo con quello salvato. Se è i dati sono corrotto il controller del disco è in grado di stabilire quale non va e correggerlo.

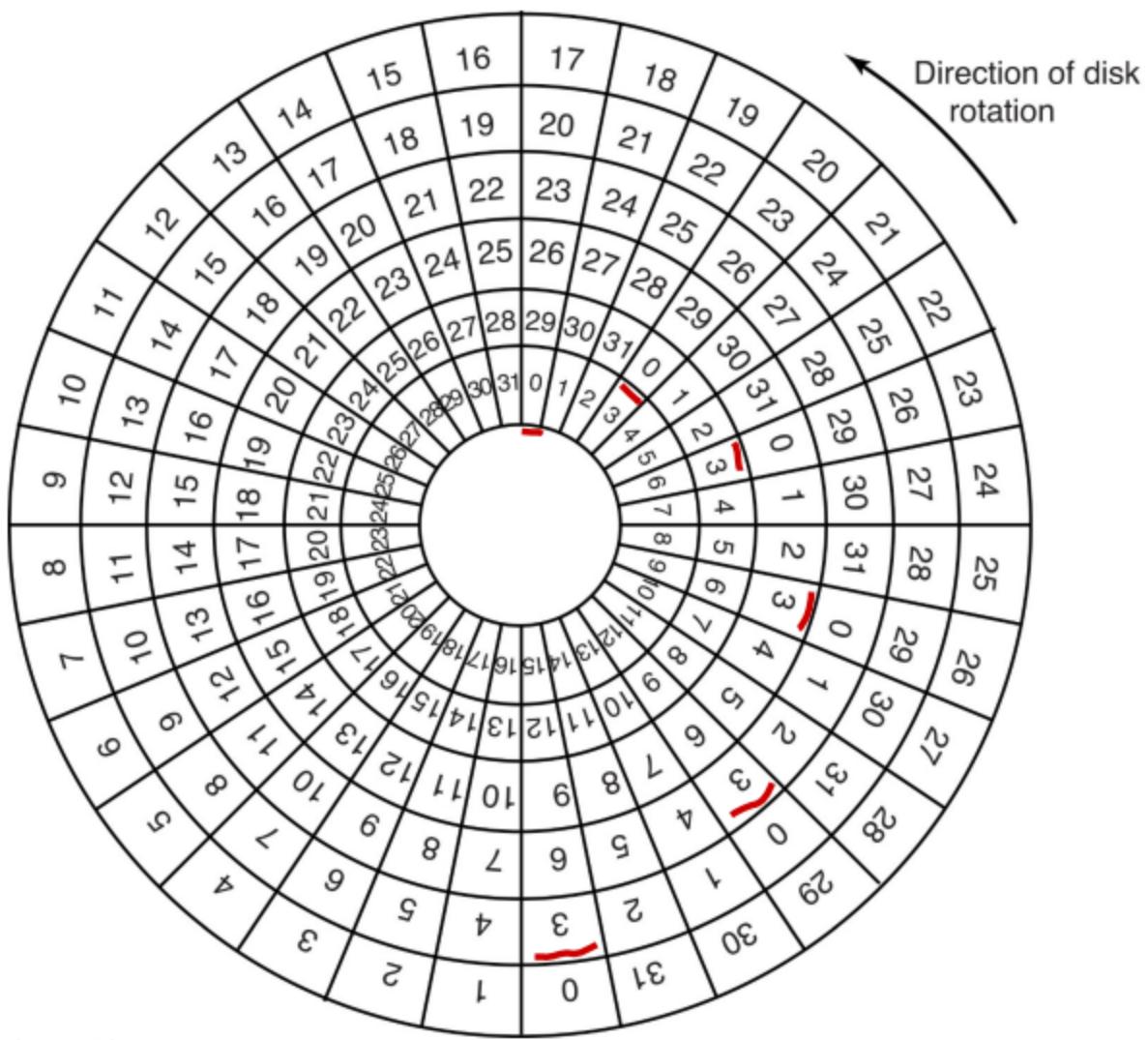
Come viene assegnato un valore al settore

Si inizia a numerare i settori nella stessa posizione nella traccia (la traccia è il cerchio in cui si trova dal centro) e viene impostato un offset.



Abbiamo così un problema di prestazioni, ogni volta che dobbiamo leggere o scrivere dobbiamo spostare il cilindro e questo riduce i tempi, per questo si è ideato il **cylinder skew** ovvero quando i dati sono organizzati in modo da ridurre gli spostamenti.

$$Cylinder Skew = \frac{Track Seek Time}{Sector Arrival Time}$$



Come possiamo vedere ogni settore dopo questa formattazione è impostato con un offset, proprio per ridurre il numero di rotazioni necessarie. Dopo la formattazione il disco è di dimensioni ridotte perché bisogna impostare il preambolo e l'ECC. Ma a questo punto deve avvenire un'altra formattazione di alto livello e deve anche esser partizionato.

Partizionamento

Separa un disco in diversi dischi logici, permettendo a diversi SO di esistere (dual boot ad esempio), tutto questo viene salvato in componente chiamato **MBR** (Master Boot Record) che tiene conto della tabella di partizionamento.

Il **MBR** viene salvato nel primo settore del disco, contiene il boot loader ovvero codice eseguibile che crea il vero SO, situato nel boot sector. Questo **boot sector** è situato in una **partizione apposita** chiamata **Boot Partition** o **Active Partition**.

La **partition table** supporta fino a 4 partizioni primarie, 1 di queste può essere una Extended Partition ovvero una partizione speciale che può contenere diverse partizioni logiche. A loro volta sono descritte dall'EBR, tanti componenti EBR formano una **linked list data structure**, dove ogni EBR punta al successivo.

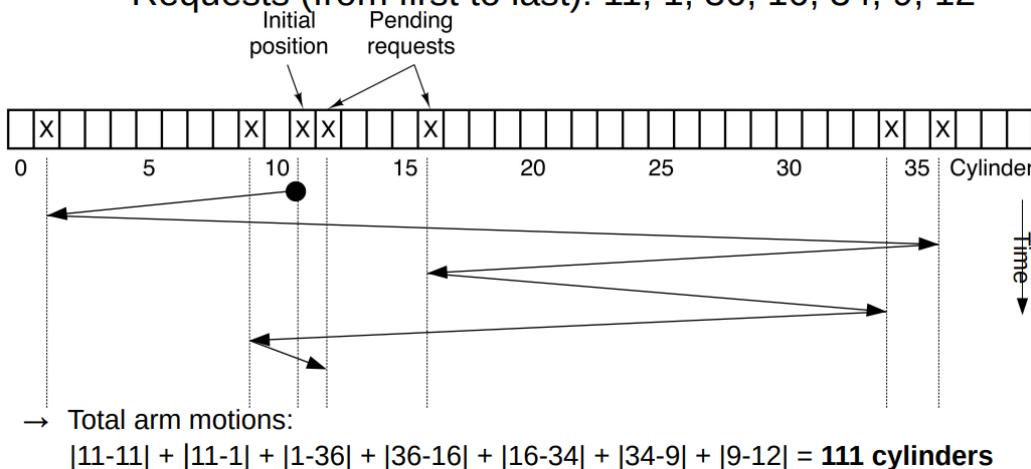
Disk Arm Scheduling

Sceglie quale richiesta servire tra le pendenti di un disco, se quando arriva una richiesta I/O il controller è occupato, dobbiamo mettere in coda nel disk driver. La coda è una tabella con i numeri del cilindro come identificatori ogni pezzo contiene le richieste del suo cilindro.

Tipi di scheduling:

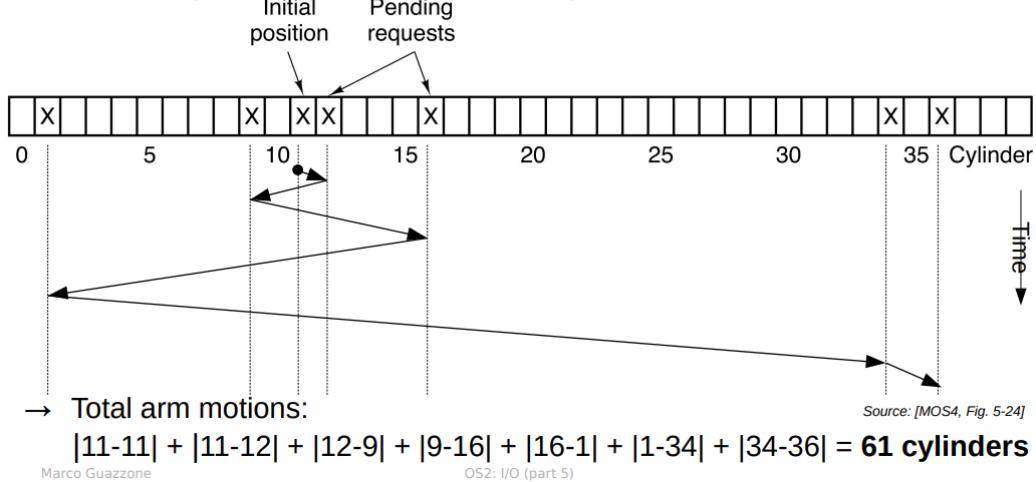
- **FCFS**: First Come First Serverd ovvero il primo che arriva.

- Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



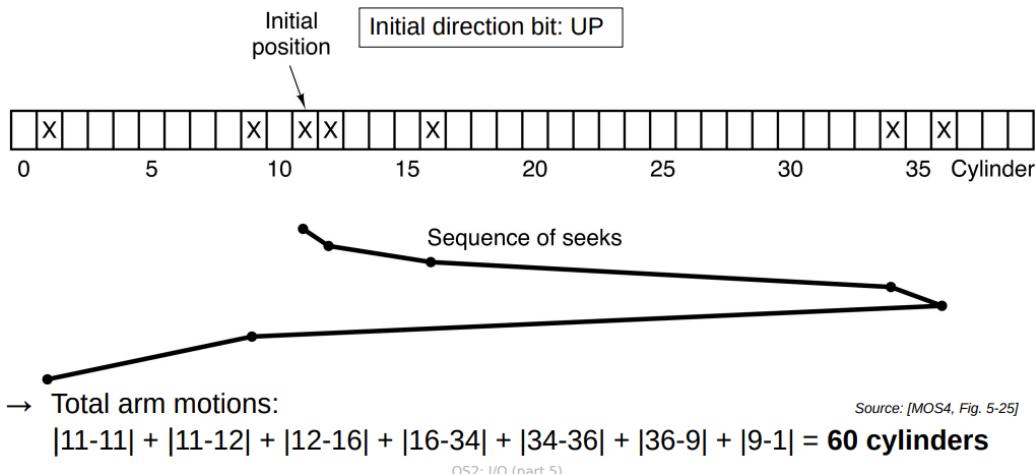
- **SSF**: Shortest Seek First ovvero quello che mette meno differenza tra uno e l'altro

- Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



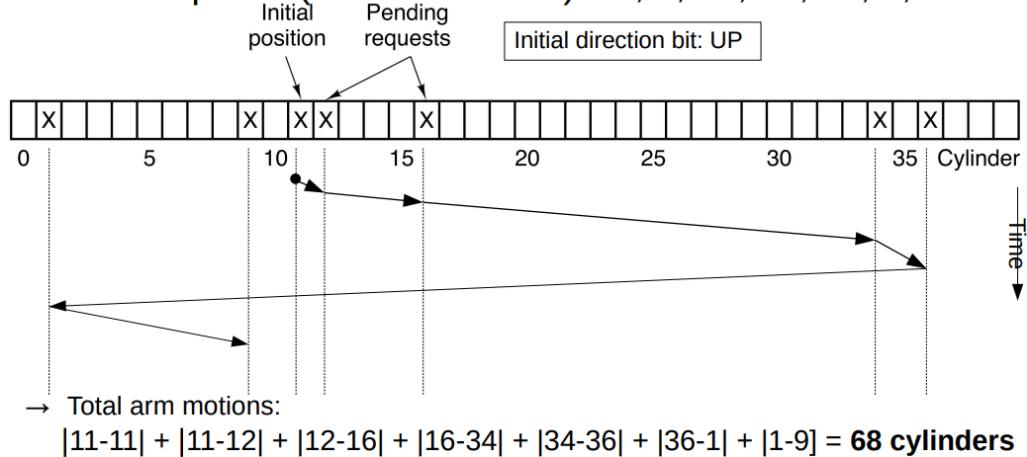
- **Look Algorithm:** prende la richiesta che è più vicina alla direzione attuale

- Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



- **C-Lock:** ovvero quando arriva all'ultimo cilindro e poi torna indietro senza servire

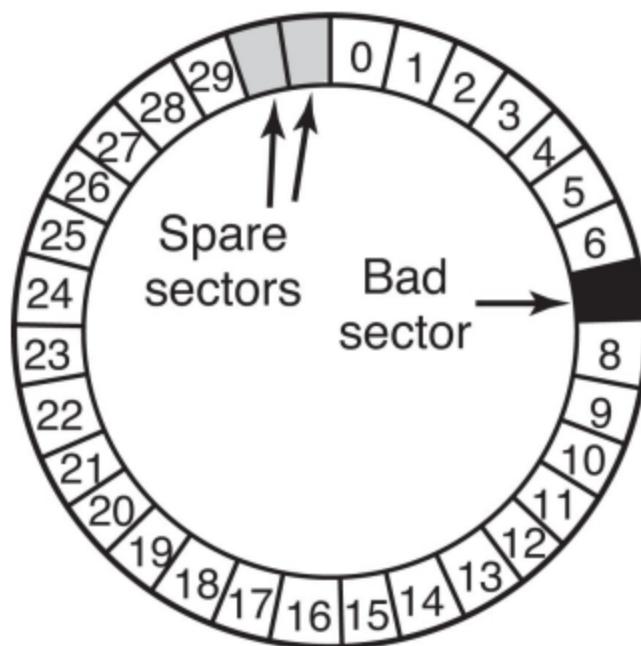
- Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



Error Handling

La superficie del disco è piena di regioni magnetizzate, che rappresentano il singolo bit, più aumenta la densità più aumenta la capacità ma aumenta anche la complessità di gestione. quando abbiamo un errore su un settore lo chiamiamo **bad sector** e per risolvere abbiamo due approcci, fisico e hardware:

- **Correzione fisica:** vengono riservati dei settori per aggiungere ridondanza, sono chiamati **spare sector** e grazie a questi possiamo aggiungere dati ridondanti



Quando il controller vede che c'è un bad sector, lo nota grazie all'ECC presente nel settore. Può sostituire quello danneggiato con uno spare, abbiamo due tecniche: sector **forwarding** o sector **slipping**.

- **Sector forwarding:** il controller esegue una ri-mappatura ovvero imposta l'indirizzo del settore danneggiato nel settore di spare. Il problema è che settori che sono logicamente consecutivi ora non lo sono più, rompe la sequenzialità.
- **Sector slipping:** il controller esegue uno shift di tutti gli indirizzi dei settori, così facendo non viene interrotta la sequenzialità. Il problema qua è che l'operazione è molto più lunga.

In entrambi i casi il controller deve sapere quale settore ha l'errore, bisogna tenere una **mappatura**, possiamo usare una **tabella** oppure **sovrascrivere il preambolo**.

Il controller quando legge i dati, prima passa per i **dati** e poi controlla il codice **ECC**, se sono **diversi** è stato **rilevata la presenza di un errore** dato che devono combaciare, si esegue anche un secondo controllo questo perchè il metallo dentro si può dilatare e quindi ci sono degli errori fisici e non logici.

Non tutti i controller sono in grado di gestire in maniera trasparente questa correzione, molte volte non ci sono neanche abbastanza spare sector per rimpiazzare i bad sector, per essere sicuri che l'utente non possa accedervi il SO nasconde un file system con i bad sector. Il SO ha una mappatura con i bad sector interna.

Errore nella **seek**: per la dilatazione possiamo avere problemi di ricerca, se c'è una minima dilatazione termina del braccio finiamo su un cilindro differente. Dobbiamo avere una **ricalibratura quindi del braccio**, viene impostato un bit di errore nello status register. Il driver inizia a ricalibrare il braccio ovvero lo sposta sul cilindro 0.