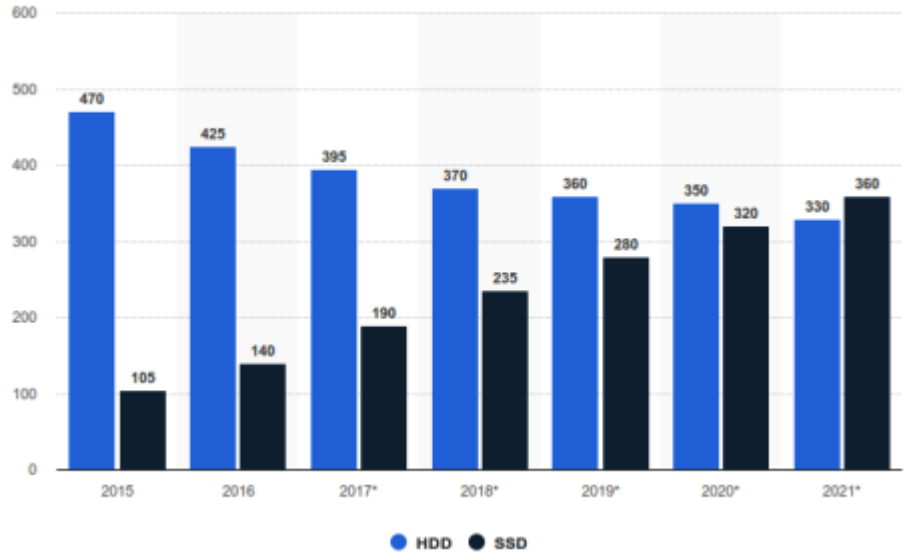
Scheduling del disco

Index

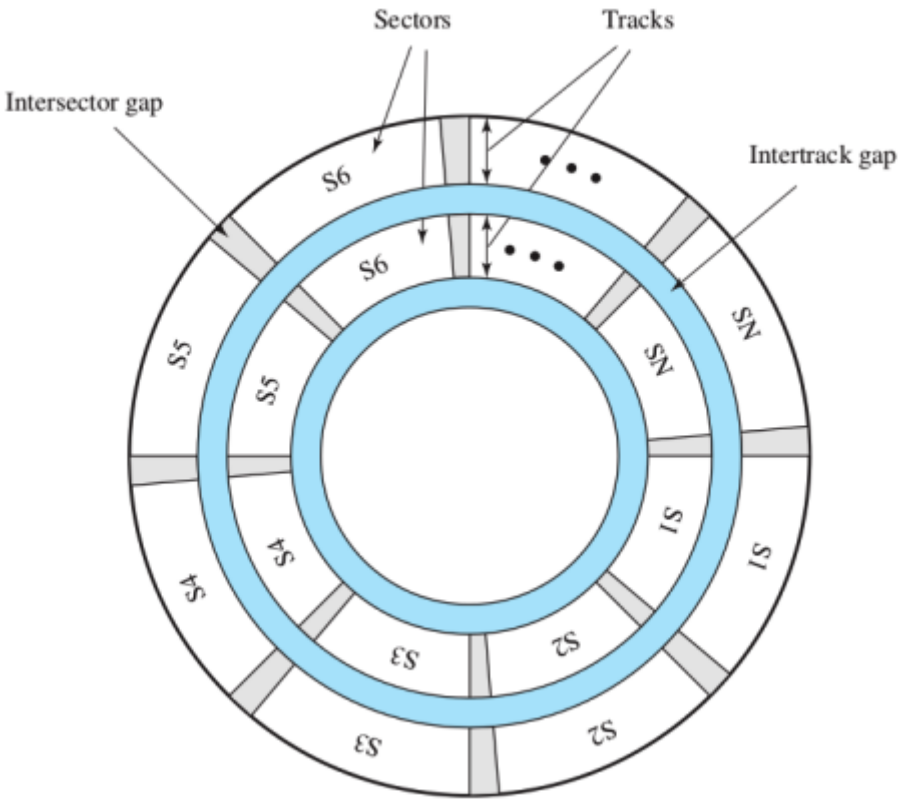
HDD vs SSD

Come abbiamo visto per gestire l'I/O il SO deve essere il più efficiente possibile.
Uno degli ambiti su cui i progettisti di sistemi operativi si sono dati più da fare è quello dei dispositivi di archiviazione di massa

Le tecniche che tratteremo riguardano solo e unicamente gli HDD (non gli SSD). Gli SSD hanno diversi problemi, che però non tratteremo



Il disco



Una *traccia* è una corona circolare, avvicinandosi al centro diventa sempre più piccola
Un *settore* è una parte di cerchio delimitata da due raggi

Chiaramente i settori e le tracce non sono contigui infatti si ha l'*intertrack gap* che separa due tracce, mentre l'*intersector gap* separa due settori. Sono presenti in quanto altrimenti non sarebbe possibile leggere e scrivere i dati con grande accuratezza

---

Come funziona un disco

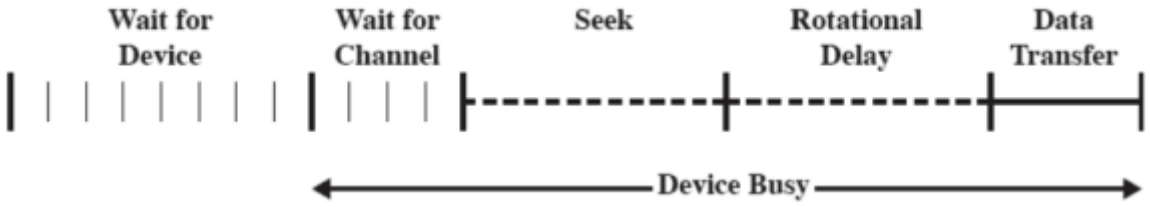
I dati si trovano sulle tracce (corone concentriche) su un certo numero di settori. Quindi per leggere/scrivere occorre sapere su quale traccia si trovano i dati, e sulla traccia, su quale settore.

Per selezionare una traccia bisogna:

- spostare la testina, se il disco ha testine mobili (*seek*)
- selezionare una testina, se il disco a testine fisse
  Per selezionare un settore su una traccia bisogna aspettare che il disco ruoti (a velocità costante)
  Se i dati sono tanti, potrebbero essere su più settori o addirittura su più tracce (tipicamente un settore misura 512 bytes)

Prestazioni del disco

La linea temporale di un disco può essere riassunta come segue



- Tempo di accesso (*access time*) → somma di:
  - Tempo di posizionamento (*seek time*) → tempo necessario perché la testina si posizioni sulla traccia desiderata
  - Ritardo di rotazione (*rotational delay*) → tempo necessario affinché l'inizio del settore raggiunga la testina
- Tempo di trasferimento (*transfer time*) → tempo necessario a trasferire i dati che scorrono sotto la testina

A parte:

- *wait for device* → attesa che il dispositivo sia assegnato alla richiesta
- *wait for channel* → attesa che il sottodispositivo sia assegnato alla richiesta (se ci sono più dischi che condividono un unico canale di comunicazione)

Politiche di scheduling per il disco

Come è successo per la RAM, nel caso in cui ci troviamo su un disco con testine mobili, sono state pensate numerose vie per poter rendere il più efficiente possibile le operazioni di lettura e scrittura

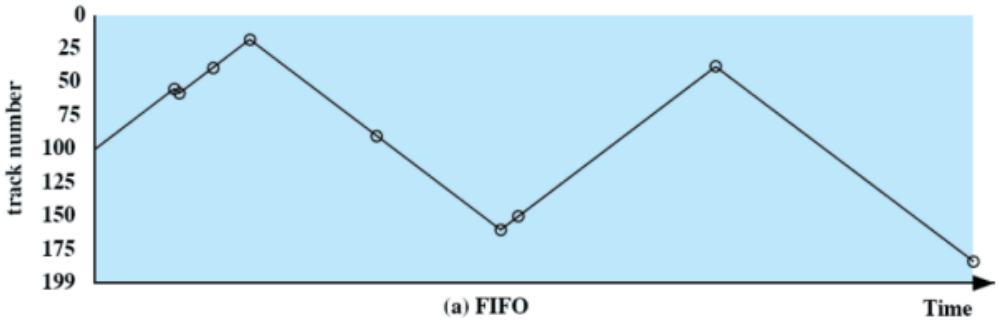
Tutte le politiche sotto elencante verranno confrontate su un esempio comune.
All'inizio, la testina si trova sulla traccia numero 100 su un totale di 200 tracce
Vengono richieste le seguenti tracce nel seguente ordine:

$$55, 58, 39, 18, 90, 160, 150, 38, 184$$

Consideriamo solo il **seek time**, che è il parametro più importante per le prestazioni.
Verranno inoltre confrontati con lo scheduling peggiore, il random

FIFO

Le richieste sono servite in modo sequenziale e risulta equo nei confronti dei processi.
Se ci sono molti processi in esecuzione, le prestazioni sono simili allo scheduling



Priorità

In questo tipo di politica l'obiettivo non è ottimizzare il disco, ma raggiungere altri obiettivi. Vengono infatti soddisfatti per primi i processi con priorità più alta.
In particolare è desiderabile fornire un buon tempo di risposta ai processi interattivi, ma i processi più lunghi potrebbero dover aspettare troppo tempo (infatti i processi batch sono corti) il che non risulta ottimale per i DBMS

LIFO

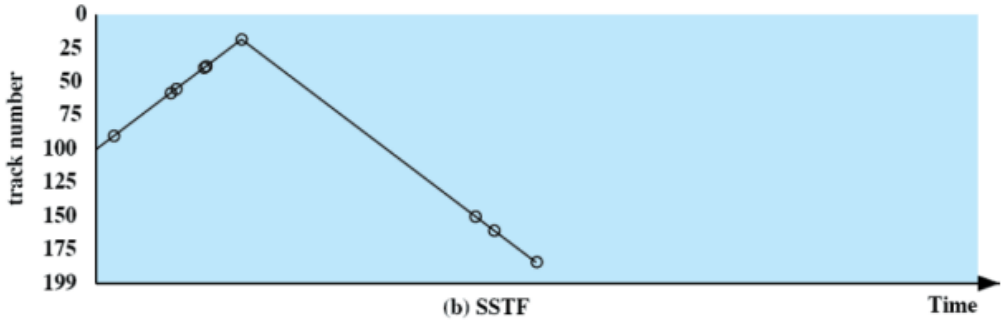
Questa politica risulta ottima per il DBMS (sequenza di istruzioni che non può essere interrotta) con transizioni.
In questo caso dunque il dispositivo è dato all'utente più recente, se quindi un utente continua a fare richieste sul disco si potrebbe arrivare alla starvation.
Il motivo per cui è usato è per il fatto che se un utente continua a fare richieste,

probabilmente sta accedendo sequenzialmente ad un file, ed è quindi più efficiente far terminare la sua lettura che altro

Minimo tempo di servizio

Da questa politica in poi, lo scopo sarà quello di minimizzare il seek time.
Per quanto riguarda il minimo tempo di servizio
Con questa politica si sceglie sempre il tempo di posizionamento minore (rispetto alle richieste attualmente pervenute)

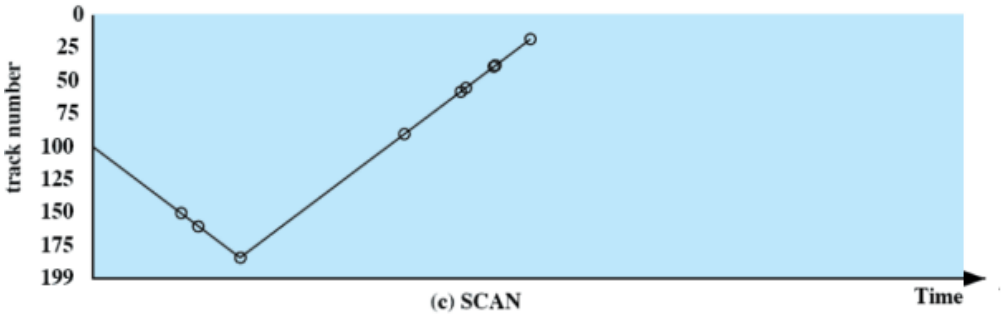
Si potrebbe però arrivare a starvation nel caso in cui arrivino continuamente richieste più vicine



SCAN

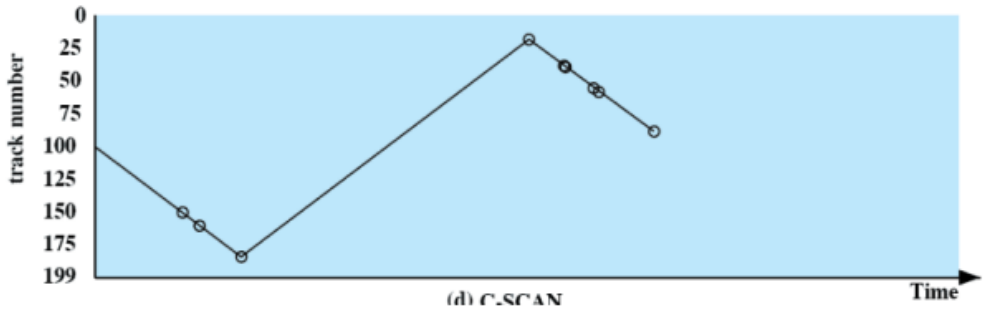
Si fa in modo che le richieste siano servite in modo tale che il braccio si muova sempre in un verso, e poi torni indietro.
In questo modo viene azzerata la starvation delle richieste ma risulta esse poco corretta poiché:

- **favorisce le richieste ai bordi** (attraversati due volte in poco tempo, in quanto vengono servite sia in discesa che in salita) → C-SCAN
- **favorisce le richieste appena arrivate**, nel senso che se ben piazzate rispetto alla testina, possono precedere richieste che attendono da molto → N-step-SCAN



C-SCAN

Come SCAN ma non si scelgono le richieste allo scendere del numero della traccia.
Quindi i bordi non sono più visitati due volte in poco tempo



FSCAN

Con questa politica vengono usate due code anziché una.
Quando SCAN inizia, tutte le richieste sono nella coda F, e l'altra coda R è vuota.
Mentre SCAN serve tutta F, ogni nuova richiesta è aggiunta ad R. Quando SCAN finisce di servire F, si scambiano F e R.
Ogni nuova richiesta deve aspettare che tutte le precedenti vengano servite, quindi le richieste vecchie non sono sfavorite rispetto alle nuove come in SCAN

N-step-SCAN

Questa politica è una generalizzazione di FSCAN a $N > 2$ code.
Si accodano le richieste nella coda i-esama finché non si riempie; poi si passa alla $(i + 1) \bmod N$.
Le richieste non sono mai aggiunte a quella attualmente servita.
Se N è alto, le prestazioni sono quelle di SCAN (ma con fairness), se $N = 1$, allora si usa il FIFO per evitare di essere unfair

Confronto prestazionale

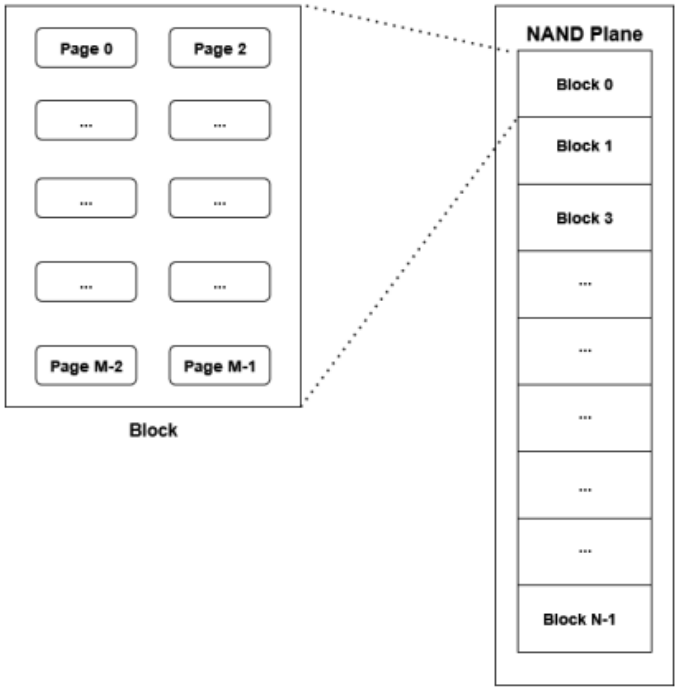
(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

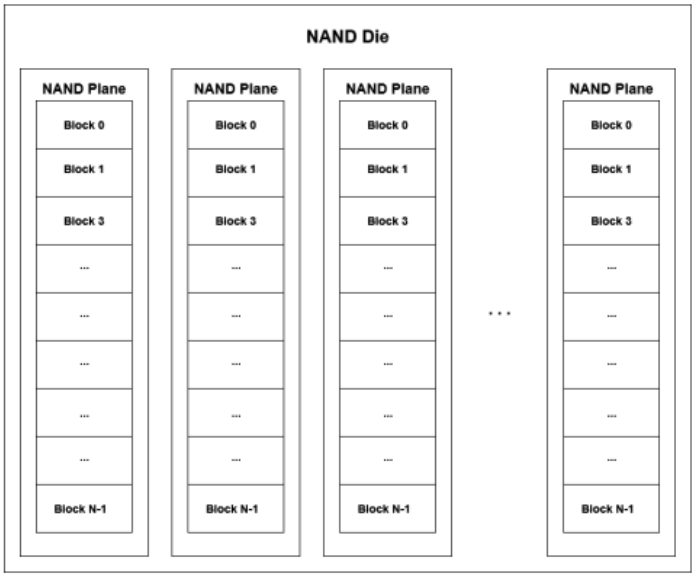
Prospetto

Algoritmo	Descrizione	Note
Selezione a cura del richiedente		
RSS	Scheduling random	Solo per simulazioni e confronti
FIFO	A coda semplice	Il più equo
LIFO	Ultimo utente	A sorpresa, può andare bene
PRI	Priorità del processo	Se la priorità è importante
Selezione sulla base del dato richiesto		
SSTF	Tempo di servizio più corto	Alto uso del disco, piccole code
SCAN	Avanti ed indietro sul disco	Miglior distribuzione del servizio
C-SCAN	Avanti, con ritorno veloce	Minore variabilità di servizio
N-step-SCAN	SCAN su N richieste	Garanzia del servizio
FSCAN	N-step-SCAN con due code	Tiene conto del carico

SSD: cenni

Gli SSD ad alto livello, sono costituiti da stack (*flash chips*) di *die* (matrici); il controller dell'SSD gestisce gli stack di die
Ciascun die ha un certo numero di *planes*. Le planes sono divise in *blocks* (blocchi). Ciascun blocco è composto da un numero variabile di *pages* (pagine $\sim 4KB$).
Infine le pagine sono composte da *cells* (celle). Le celle dunque sono le unità più piccole di un SSD e possono immagazzinare un solo bit (due per muti-level cells)





Le operazioni sugli SSD hanno granularità diversa in base al tipo di operazione:

- in lettura → l'unità di accesso minima solo le pagine
- in scrittura → l'unità di accesso minima è sempre la pagina, ma può essere scritta solo se vuota (non è possibile sovrascrittura)
  Sovrascrivere una pagine implica il dover azzerare l'intero blocco che la contiene

1. Si fa una copia dell'intero blocco
2. Si azzera il blocco
3. Si scrive la nuova pagina
4. Si copiano le pagine rimanenti della copia effettuata in precedenza

Gli SSD sono preferiti rispetto agli HDD perché estremamente veloci, infatti:

- nessun tempo richiesto per effettuare il seek (il che implica che non è meno problematico il sapere dove si trovino i dati)
- consentono accesso parallelo ai diversi flash chip

Sono state inoltre progettati algoritmi di accesso e file system progettati per massimizzare le performance di SSD (esistono addirittura dei file system dedicati, F2FS)