

I/O e Gestione dei Dischi



Corso di Laurea in Ingegneria Informatica
Università degli Studi di Napoli Federico II
Anno Accademico 2024/2025, Canale San Giovanni

I/O e gestione dei dischi

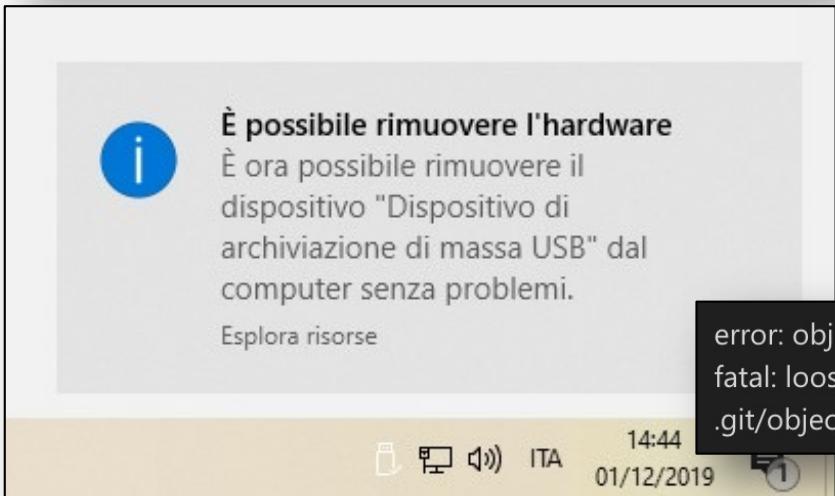
- Sommario

- Architettura del sistema di I/O
- Device driver
- I/O su disco e politiche di scheduling
- Scheduling del disco in Linux
- RAID
- SSD

- Riferimenti

- Dispensa didattica su I/O e gestione dei dischi
- Dispensa didattica su SSD
- P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari, "Sistemi Operativi", Mc-Graw-Hill (Cap.5)
- www.ostep.org, Capp. 36, 37, 38, 44
- W. Stallings "Operating Systems", Cap. 11

Importanza dello I/O



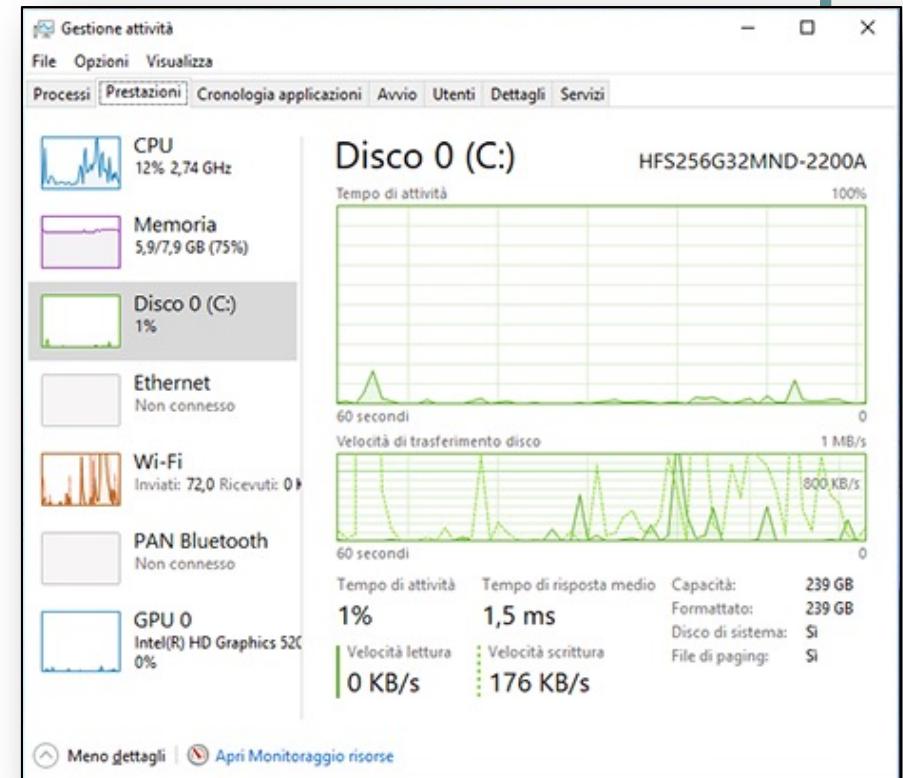
Il sistema di I/O (in particolare, lo **storage**) è spesso

- **collo di bottiglia** delle prestazioni
- causa di **perdita dei dati**

```
error: object file .git/objects/31/65329bb680e30595f242b7c4d8406ca63eeab0 is empty
fatal: loose object 3165329bb680e30595f242b7c4d8406ca63eeab0 (stored in
.git/objects/31/65329bb680e30595f242b7c4d8406ca63eeab0) is corrupt
```

Indicatori di prestazione

- Bassi **tempi di risposta (latenza)**
- Alto **numero di richieste servite (throughput)**
- Alta **percentuale di utilizzo** dei dispositivi
- **Equità** (no starvation)



Periferiche di I/O



Human readable
Interazione con
l'utente
(tastiera, video, ...)



Machine readable
Interazione con
dispositivi locali
(es. storage,
sensori)



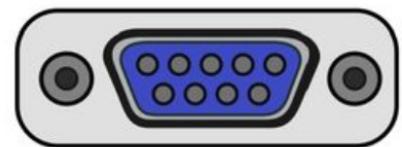
Comunicazione
Interazione con
dispositivi
remoti (es.
Ethernet card)

Periferiche di I/O

Character devices



keyboard



serial port

Accesso per **flussi di dati sequenziali**, a piccole unità (es. byte)

Block devices



hard disk drive



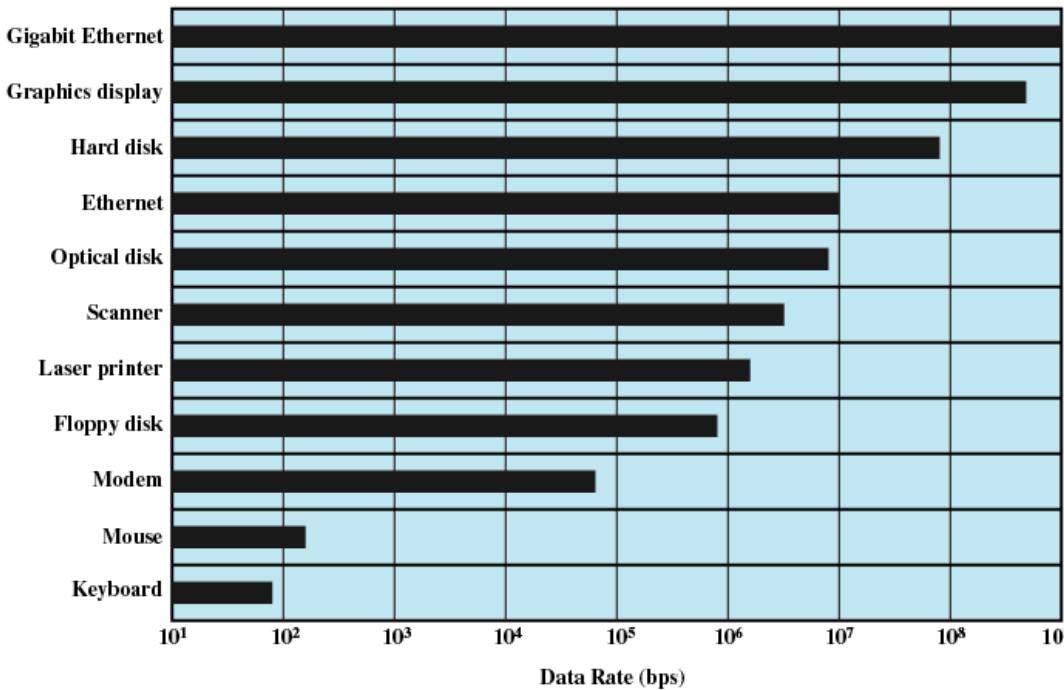
SD card



USB drive

Accesso a **posizioni arbitrarie (random)**, per blocco (es. 1 Kbyte)

Prestazioni delle periferiche di I/O



I dispositivi di I/O sono **numerosi ed eterogenei**, ed **estremamente lenti** (rispetto a CPU e memoria)

Le applicazioni sono **sensibili alla latenza**

Prestazioni delle periferiche di I/O



Inverso - The Peripheral (2022)

Virtual Reality Check: Are Our Networks Ready for VR?

<https://www.blog.adva.com/en/virtual-reality-check-are-our-networks-ready-for-vr>

Sistema di I/O

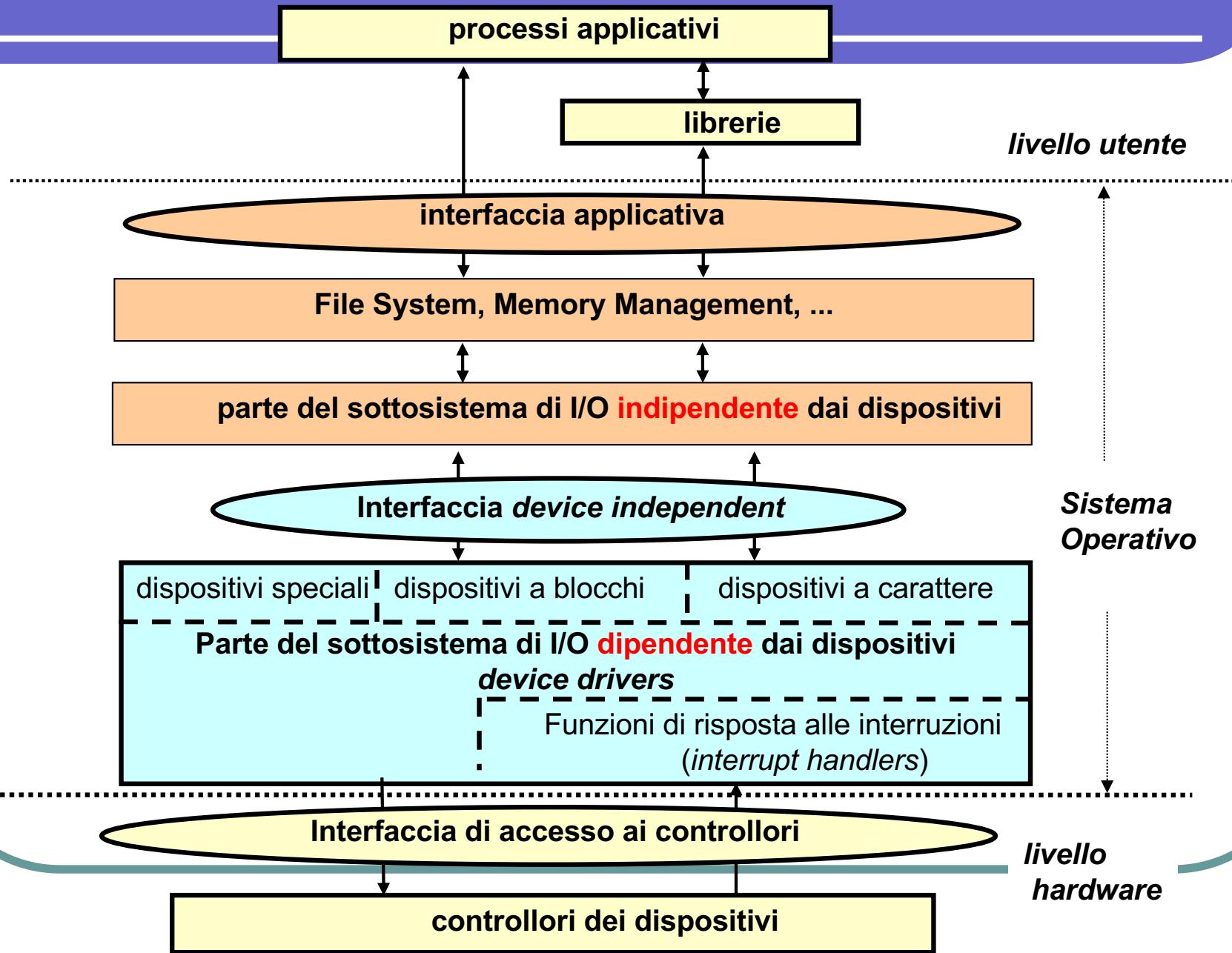
Il sistema di I/O del SO fornisce una **astrazione dei dispositivi** ai programmi utente

- I programmi accedono ai dischi usando sempre le **stesse primitive** (`read`, `write`, `open`, `close`, ...)
- Indipendentemente dalla tecnologia
- Nasconde la **eterogeneità** dei dispositivi

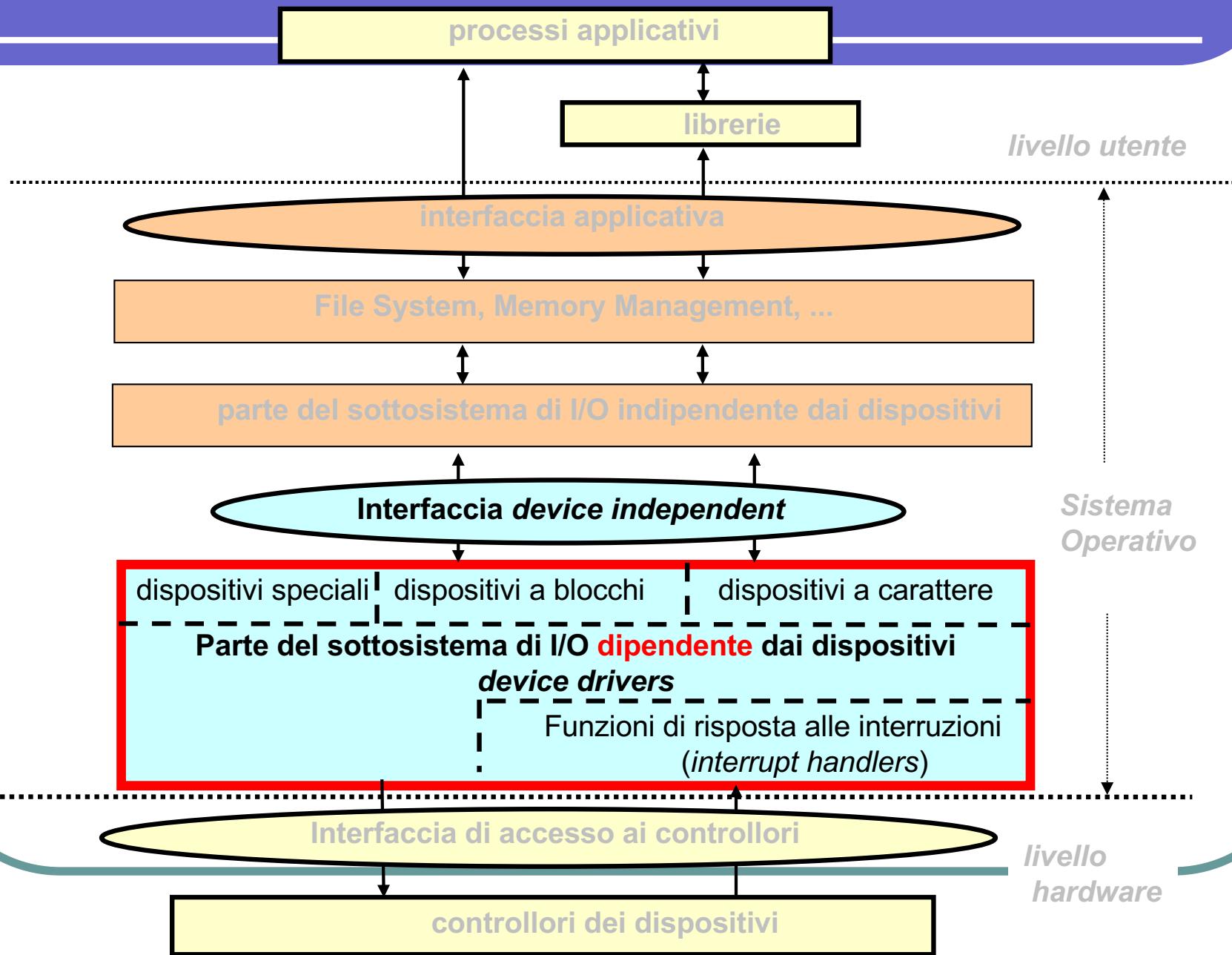
Consente una gestione **modulare**

- Es. aggiunta del supporto ad un nuovo tipo di dispositivo, tramite caricamento di un **modulo kernel**

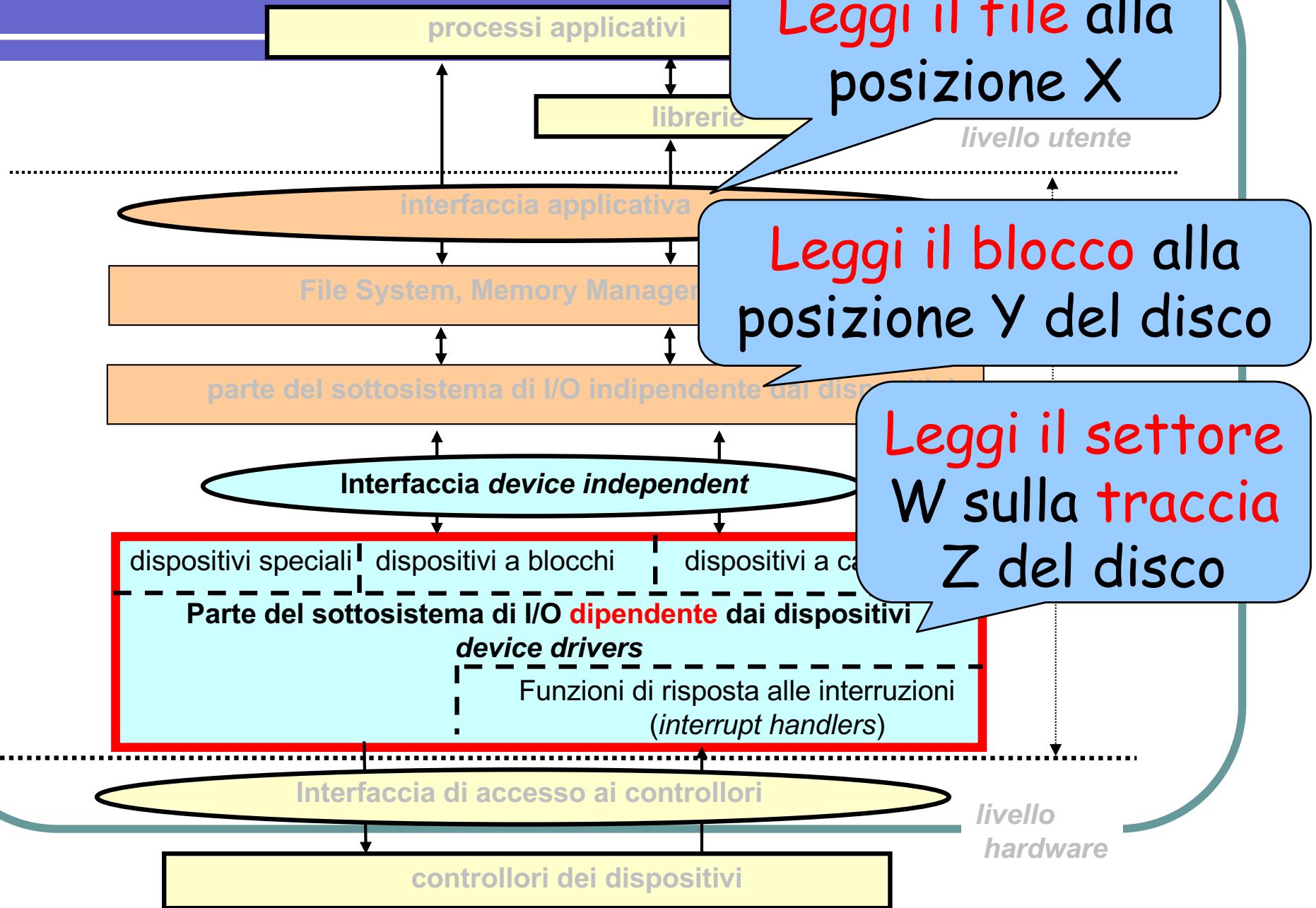
Architettura del sistema di I/O



Architettura del sistema di I/O



Architettura del sistema di I/O



Device Drivers

- I device drivers sono specifici per il dispositivo
 - es., ISR, indirizzi dei registri, comandi di I/O, ...
- Rappresentano la maggior parte del SO
 - Circa il **70% del codice** di Linux!
 - Sono tra le maggiori cause di **malfunzionamenti!**

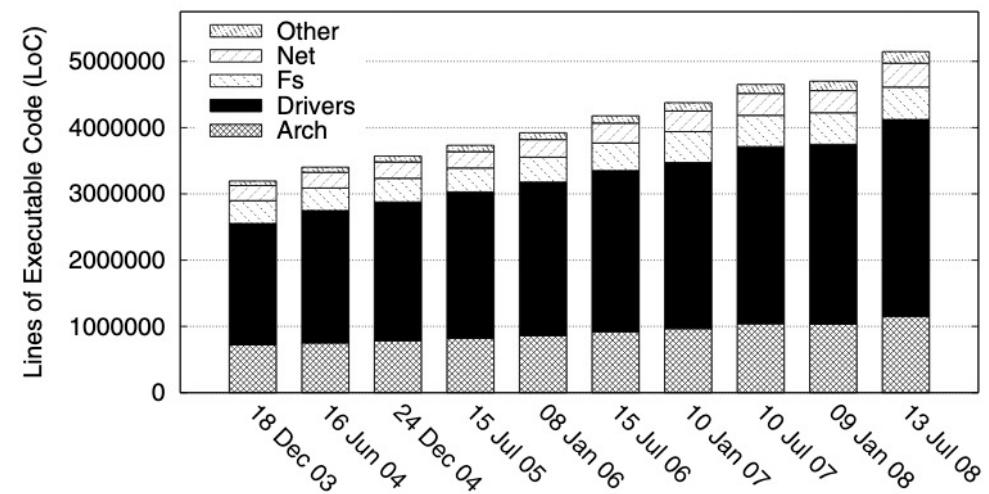
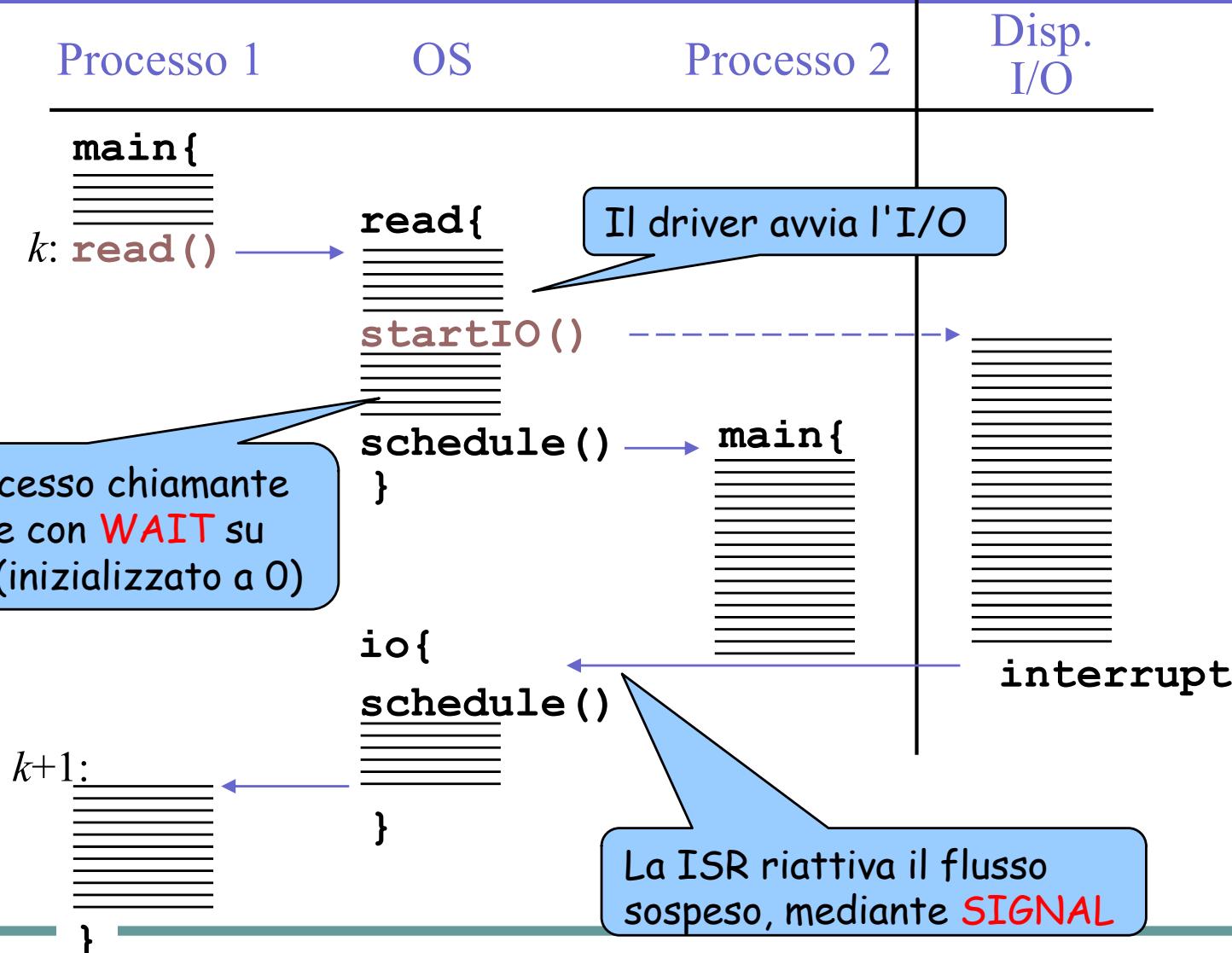


Figure 1: Growth of the Linux 2.6 kernel since its release.

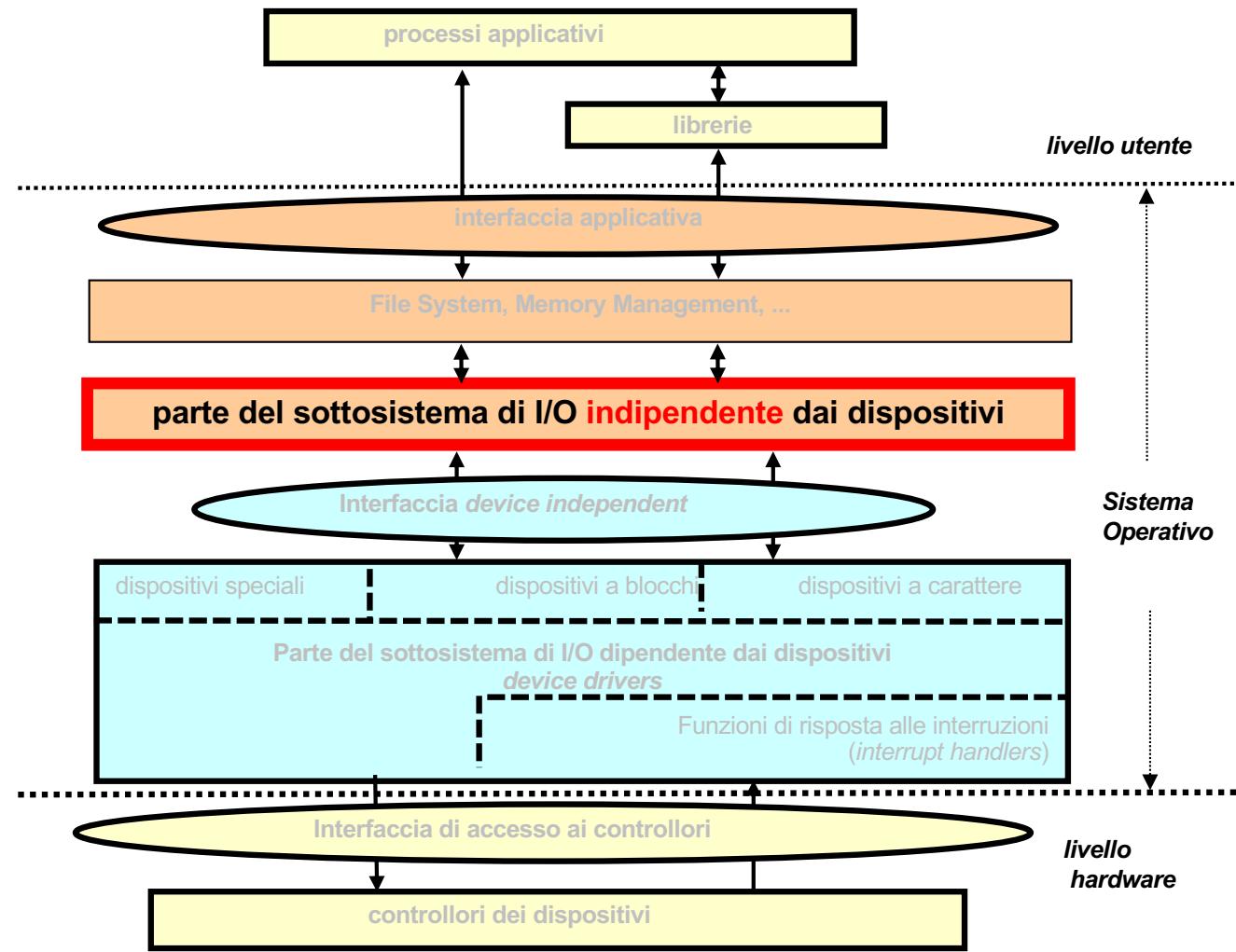
Sincronizzazione nei device drivers

- È necessario **sospendere il processo richiedente** durante le operazioni di I/O
- Un altro processo può nel frattempo **utilizzare la CPU e le altre risorse**

Sincronizzazione nei device drivers



Device-independent I/O subsystem



Responsabilità del livello indipendente dai dispositivi

- Naming
- Virtualizzazione delle periferiche
- Allocazione dei dispositivi e spooling
- I/O Scheduling
- I/O Caching e Buffering
- Gestione dei guasti

I/O su disco

- Tra tutti i dispositivi di I/O, i **dischi** risultano quelli più **critici**
 - Salvataggio dei **file** su file system
 - Gestione memoria virtuale (**swapping**)
 - Area di transito (**spooling**) verso altre periferiche



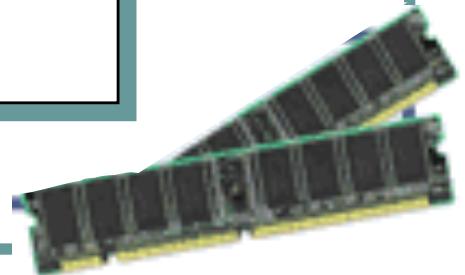
I/O su disco

- I dischi hanno una **capacità estesa**, anche se con prestazioni ridotte
- I dischi sono **non volatili**
- I dischi costano meno

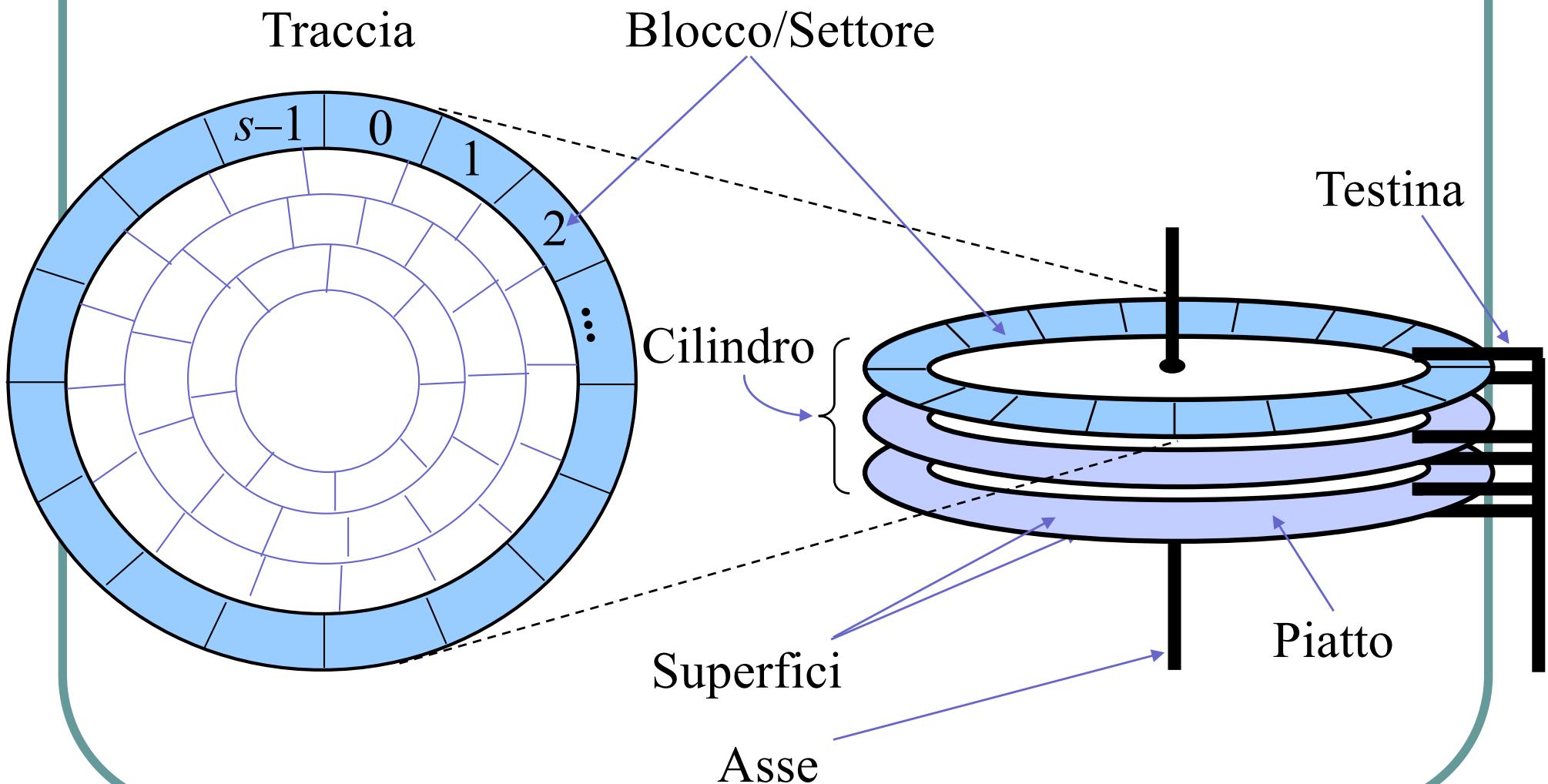


	GB/dollar	dollar/GB
RAM	0.1	\$9.5
Disks	18	50¢

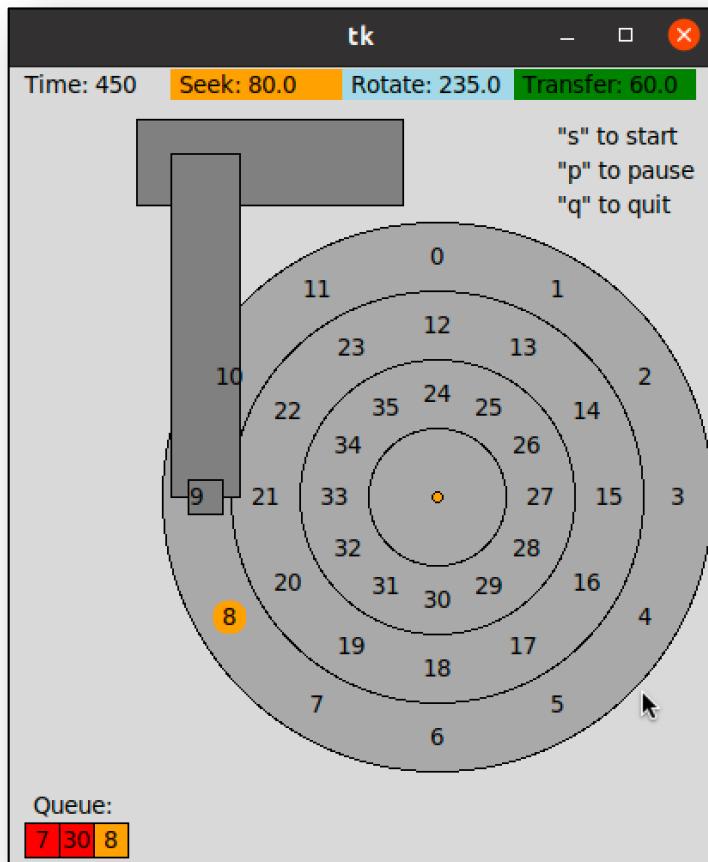
Capacità : 8GB vs. 1TB



Anatomia di un disco

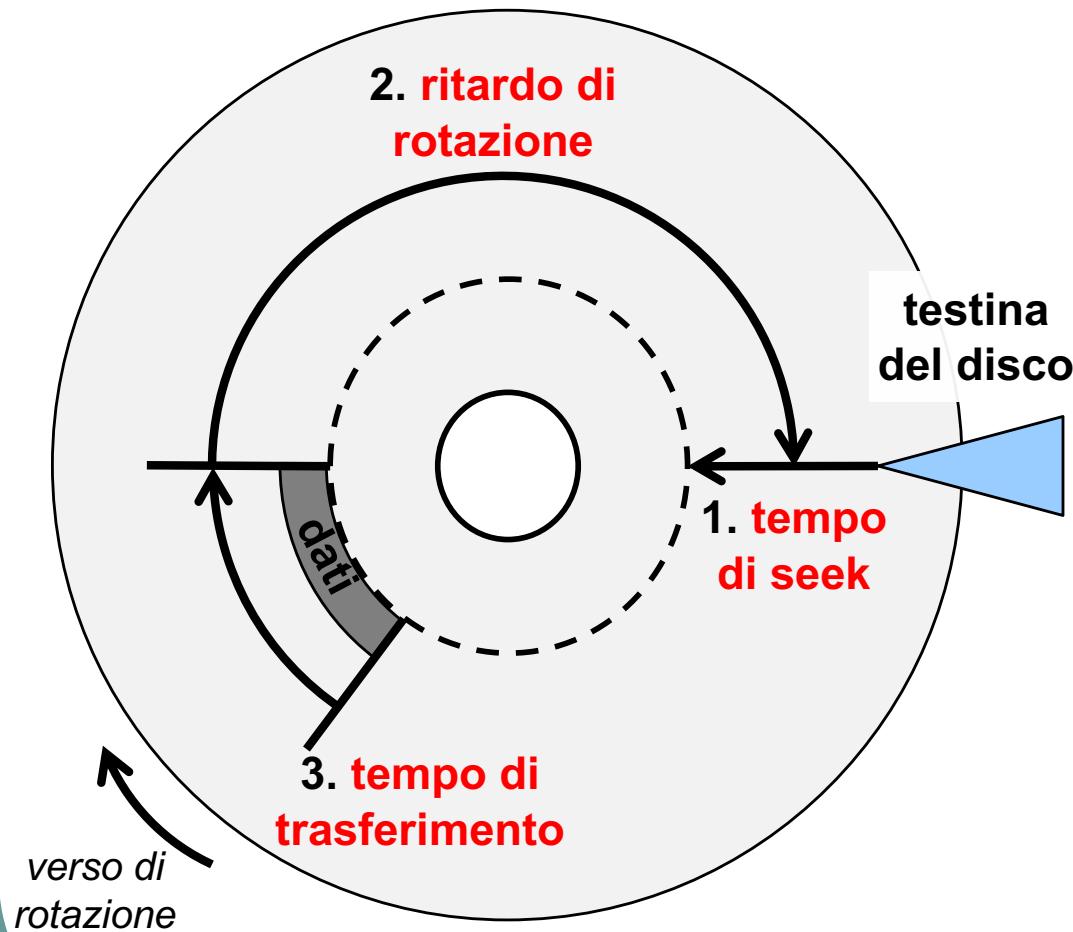


Simulatore



```
$ git clone https://github.com/  
remzi-arpacidusseau/ostep-homework.git  
_____  
$ cd ostep-homework  
$ cd file-disks  
  
$ sudo apt install python3-tk  
  
$ python3 disk.py -G -a 7,30,8
```

Parametri prestazionali



1. **Tempo di seek:** *tempo di ricerca per posizionare la testina sulla traccia desiderata*
2. **Ritardo di rotazione:** *tempo necessario affinché l'inizio del dato ruoti presso la testina*
3. **Tempo di trasferimento:** *tempo necessario alla rotazione dell'area del dato sotto la testina*

Tempo di trasferimento totale

$$T = T_s + T_{rl} + T_{tr}$$

$$T_{rl} = \frac{1}{2r} \text{ Ritardo di rotazione}$$

$$T_{tr} = \frac{b}{rN} \text{ Tempo di trasferimento}$$

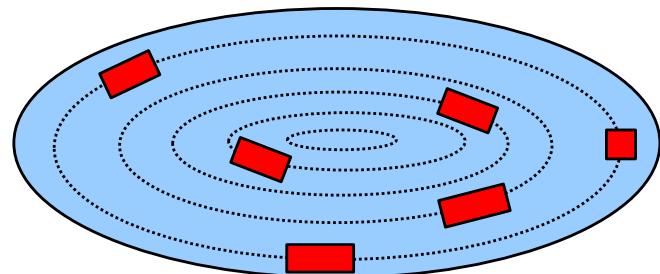
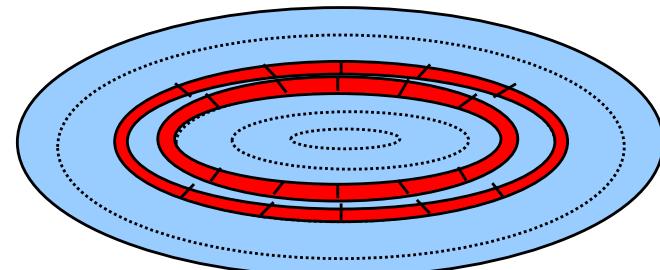
r = #rotazioni al secondo
b = byte da trasferire
N = # byte di un traccia

In media, il tempo per effettuare la **metà** di una **rotazione completa** ($1/r$)

Tempo di una **rotazione completa** ($1/r$), moltiplicato per la **frazione della traccia da percorrere** (b/N)

Esempio

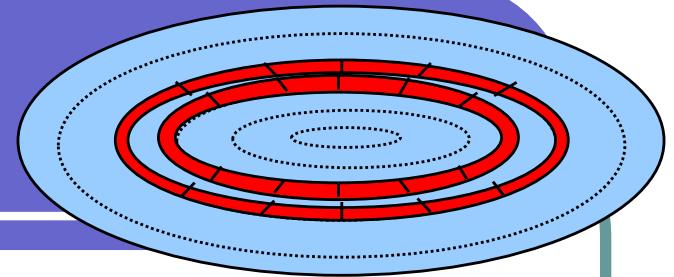
- Si vuole calcolare il tempo di trasferimento totale di un file nei due seguenti casi:
 - **Caso 1)** Il file è memorizzato in maniera compatta, occupando 5 tracce adiacenti
(organizzazione sequenziale)
 - **Caso 2)** Il file è memorizzato in settori che sono distribuiti in maniera random su disco
(organizzazione random)



Esempio

- Lettura di un file di **2500 settori** (totale ca. 1,22 Mb)
- Ipotesi:
 - Disco 1500rpm
 - Settori da 512byte
 - 500 settori per traccia
 - Tempo di seek = 4ms

Esempio



- Caso 1 (**sequenziale**, su 5 tracce consecutive)

- Tempo di lettura della prima traccia:

$$T_1 = T_s + T_{rl} + T_{tr} = 4ms + \frac{1}{2 \cdot 0.250} ms + \frac{1}{0.250} ms = 10ms$$

- Le tracce rimanenti possono essere lette consecutivamente (**senza seek time**):

$$T_2 = T_{rl} + T_{tr} = \frac{1}{2 \cdot 0.250} ms + \frac{1}{0.250} ms = 6ms$$

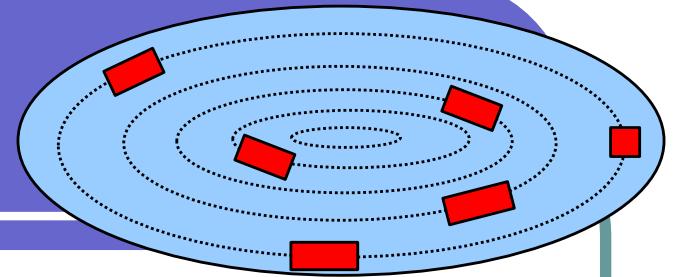
- Tempo di trasferimento totale:

$$T = T_1 + 4 * T_2 = 10 + 4 * 6 = 34 \text{ ms} = \mathbf{0.034 \text{ s}}$$

15000 rpm = 250 rotazioni per secondo

Tempo di 1 rotazione = 1/250 secondi = 1/0.250 ms

Esempio



- Caso 2 (organizzazione **random**)
 - Tempo di lettura di un singolo settore:
- $$T_i = T_s + T_{rl} + T_{tr} = 4ms + \frac{1}{2 \cdot 0.250} ms + \frac{1}{500 \cdot 0.250} ms = 6.008ms$$
- Tempo totale:
- $$T = 2500 * 6.008 \approx \mathbf{15s}$$

15000 rpm = 250 rotazioni per secondo

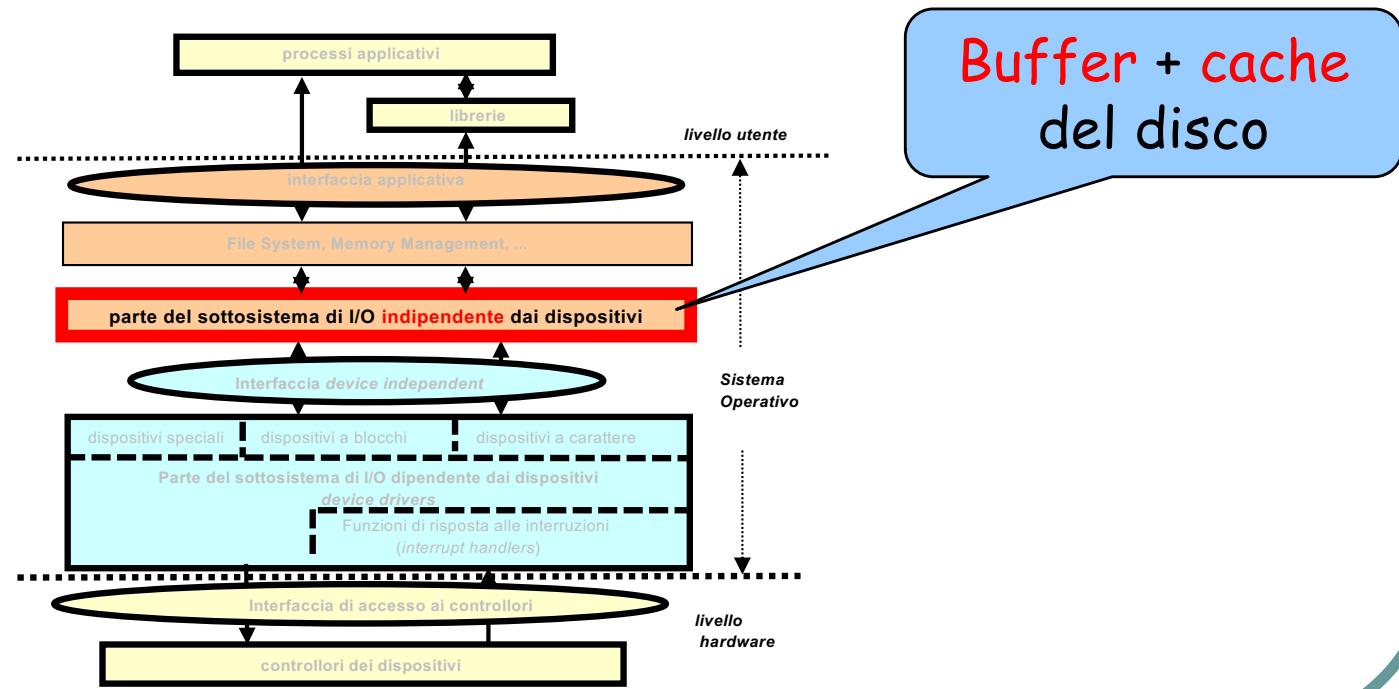
Tempo di 1 rotazione = 1/250 secondi = 1/0.250 ms

Tempo di un 500-esimo di rotazione = 1/(500*0.250) ms

I/O Caching e Buffering

- **Buffer-Cache:**

Area di memoria principale che mantiene una copia dei dati in transito da/verso il disco



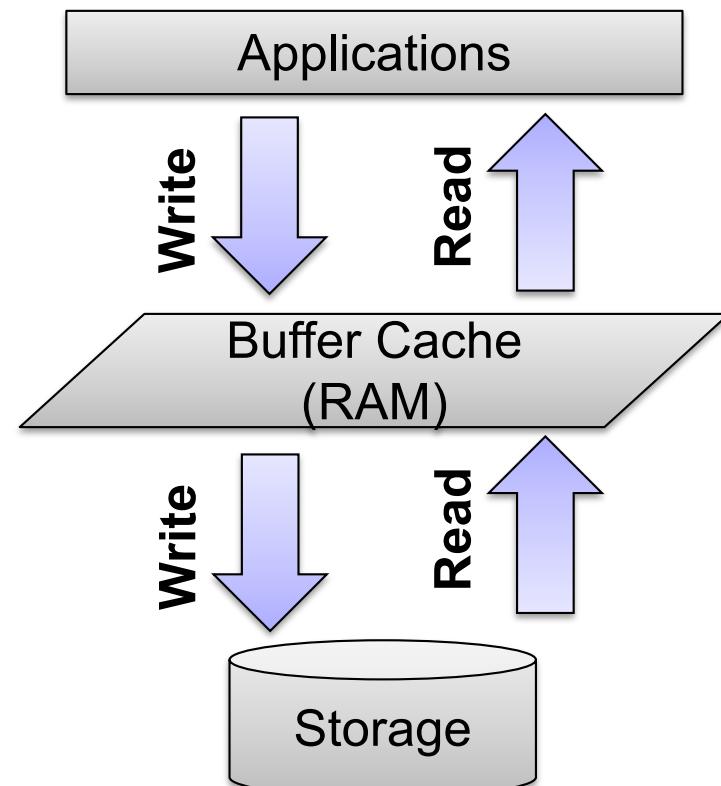
I/O Caching e Buffering

- **Buffer:**

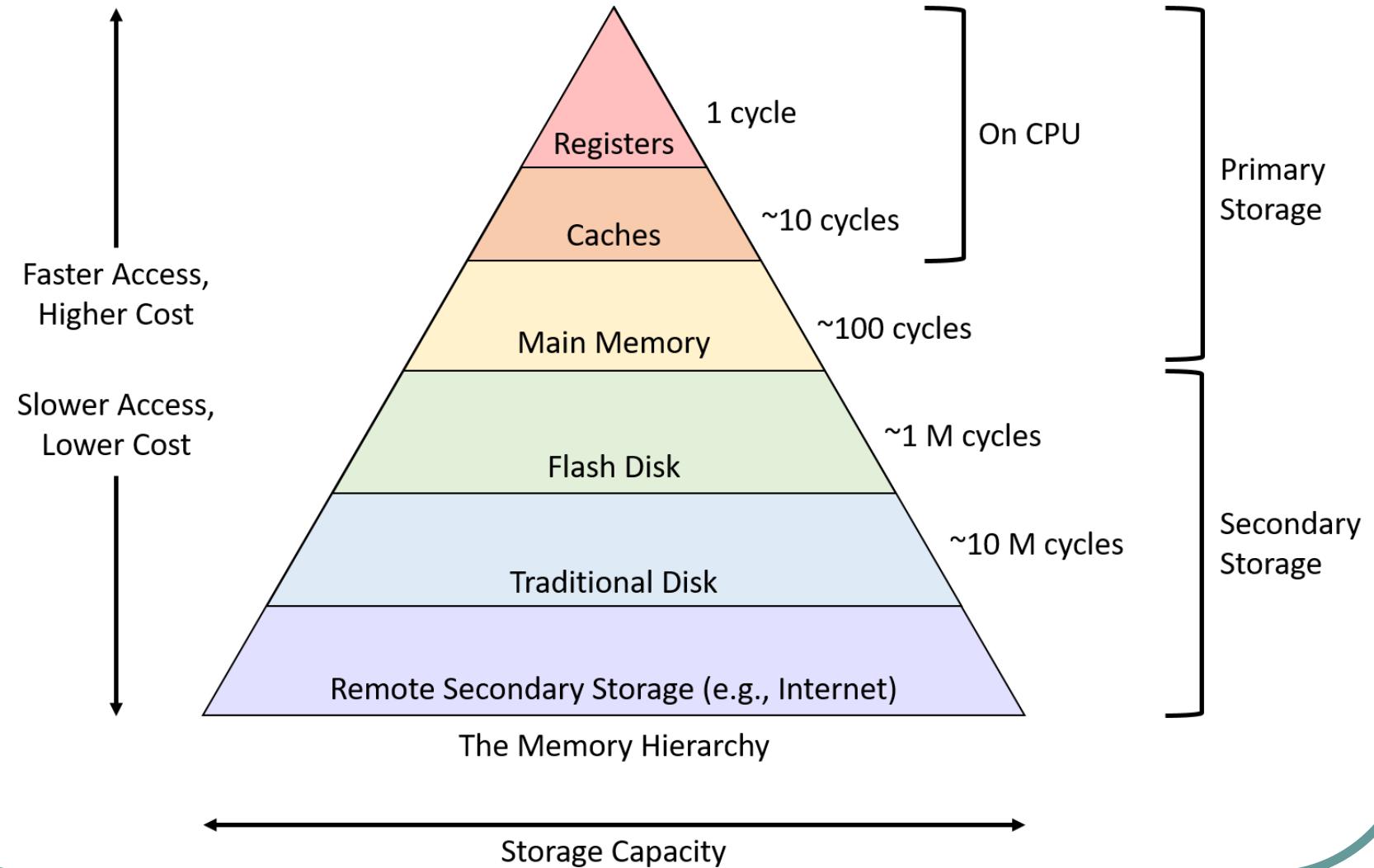
- I dati sono prima copiati in memoria
- Posticipa la scrittura di dati sul disco

- **Cache:**

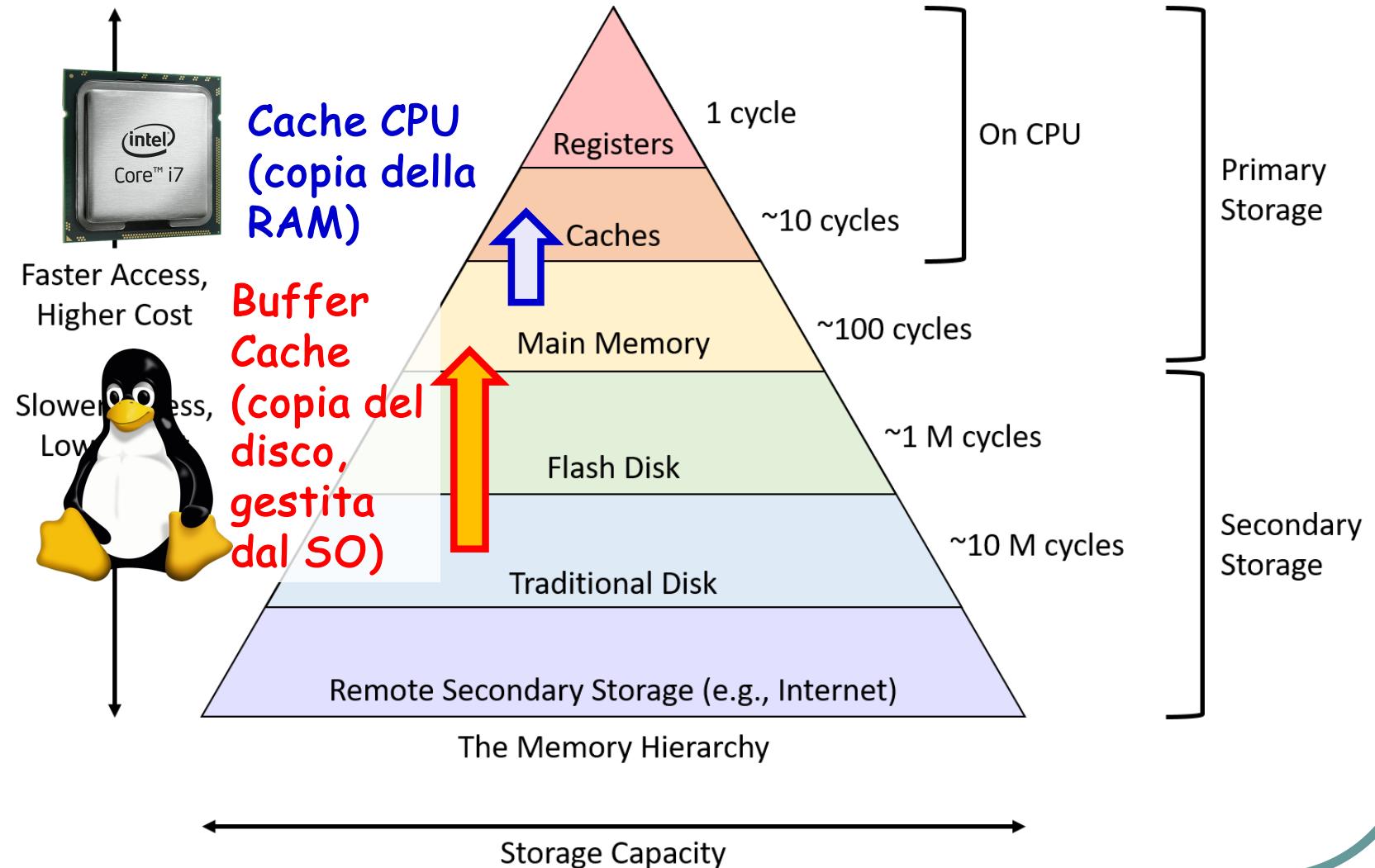
- Conserva dati recenti in RAM
- Una seconda lettura accede alla RAM, invece che al disco



I/O Caching e Buffering



I/O Caching e Buffering

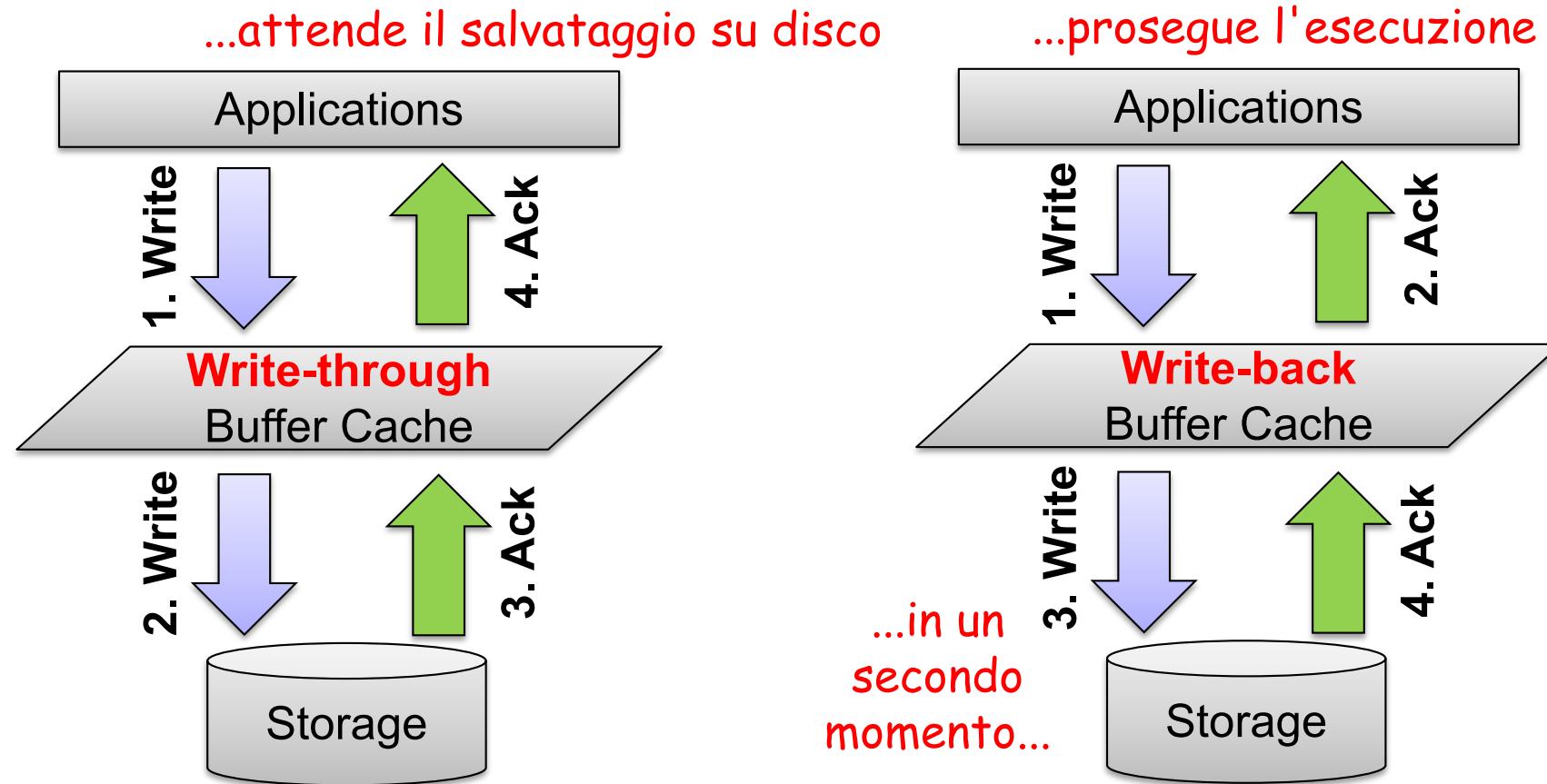


Politiche di I/O Buffering

- **Write-through**: i dati vengono scritti sia su buffer sia su disco **immediatamente ad ogni modifica**
- **Write-back**: i dati non sono scritti subito su disco, ma "svuotati" (**flush**) periodicamente o quando si accumulano

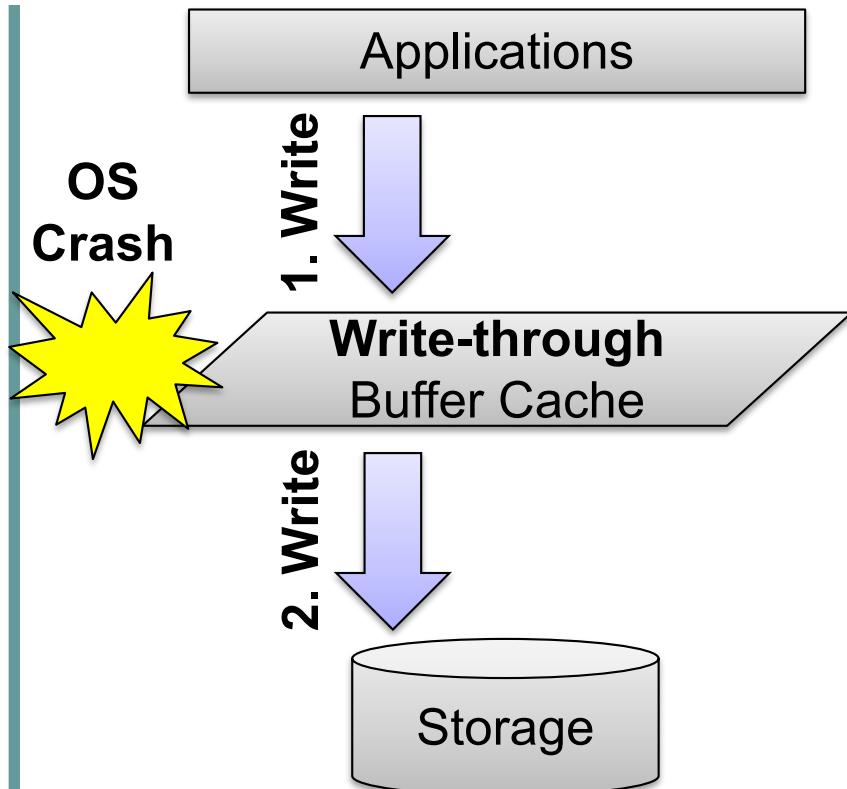
La politica **write-back** ha migliori prestazioni, ma può causare perdita di dati in caso di **guasti** (es. power outage)

Write-through vs. write-back



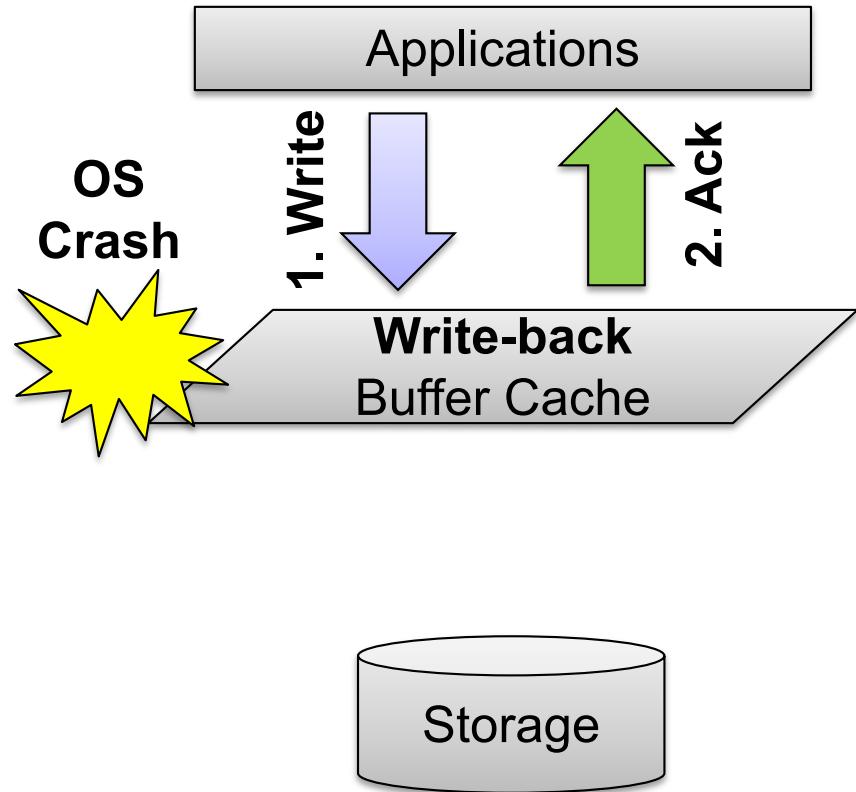
Write-through vs. write-back

...l'applicazione attende, non produce altri dati



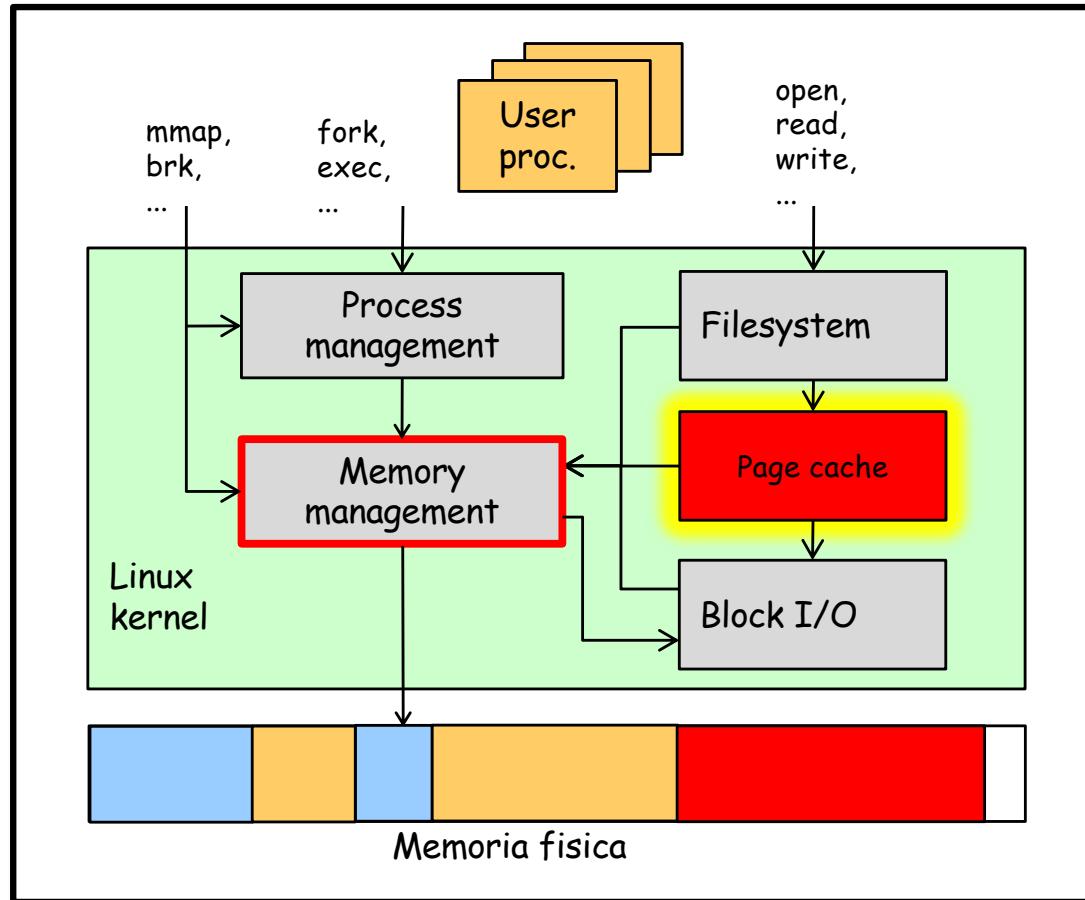
- ☺ Affidabilità dei dati
- ☹ Basse prestazioni

...l'applicazione produce altri dati



- ☹ Vulnerabile a perdita di dati
- ☺ Migliori prestazioni

Page cache in Linux



Tipicamente, tutta la **memoria RAM residua** non usata dai processi viene dedicata alla page cache

Page cache in Linux

Esempio di programma per riempire
(artificialmente) la page cache

```
find /usr -type f -exec cat {} \; > /dev/null
```

- cerca tutti i file in /usr
- ne legge il contenuto (cat)
- l'output viene ignorato (/dev/null)
- il kernel ne mantiene una copia in page cache

Page cache in Linux

2Gb
RAM

~2% memoria libera, 0% swapped

```
top - 18:41:12 up 2 days, 13:40, 3 users, load average: 0.05, 0.11, 0.26
Tasks: 120 total, 1 running, 119 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2062076k total, 2003876k used, 58200k free, 267580k buffers
Swap: 2554292k total, 0k used, 2554292k free, 1520060k cached
```

~87%
page
cache

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	2828	1296	1092	S	0.0	0.1	0:00.89	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.22	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.11	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:00.22	migration/1
7	root	20	0	0	0	0	S	0.0	0.0	0:00.12	ksoftirqd/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/1
9	root	20	0	0	0	0	S	0.0	0.0	0:00.74	events/0
10	root	20	0	0	0	0	S	0.0	0.0	0:00.65	events/1

Sistema scarico

Page cache in Linux (parametri)

- **Tempo massimo** dopo cui un blocco “sporco” in memoria viene aggiornato su disco (30 sec.)

```
# echo 3000 > /proc/sys/vm/dirty_expire_centisecs
```

- **Percentuale** di memoria “sporca” oltre la quale si forza l’aggiornamento su disco (10%)

```
# echo 10 > /proc/sys/vm/dirty_background_ratio
```

È utile ridurre la frequenza dei flush su disco per favorire **applicazioni “write-heavy”**

Swappiness

- Il parametro **Swappiness** regola l'algoritmo di **selezione delle pagine vittima** per lo swap-out

```
# echo 60 > /proc/sys/vm/swappiness
```



Tende a spostare in swap le
pagine della buffer/cache
(preserva i processi)

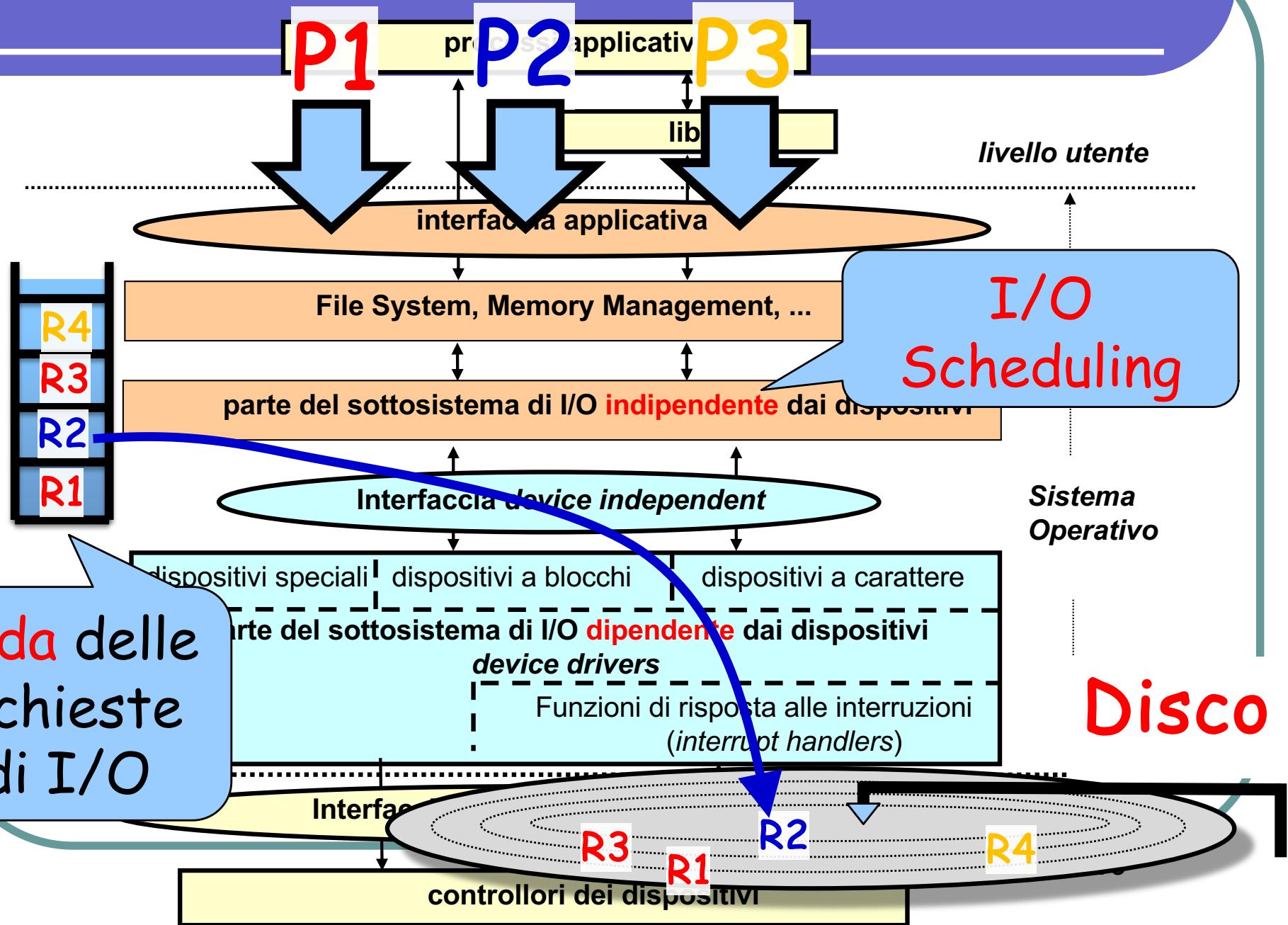


Tende a spostare in swap le
pagine dei processi
(preserva la page cache e I/O)

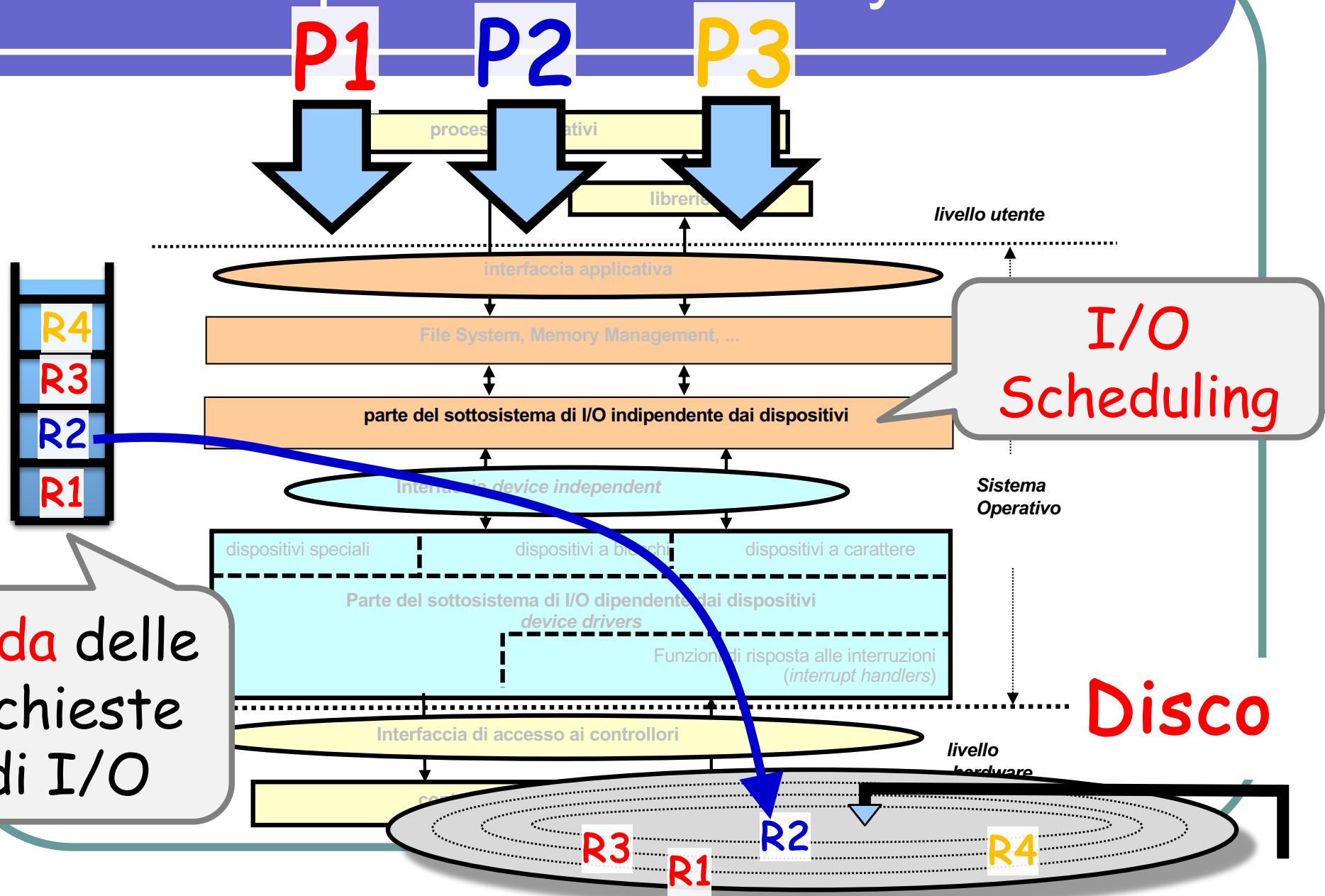
I/O Scheduling su disco

- Il SO riceve richieste di I/O da **più processi differenti** (multiprogrammazione) che accedono a **file differenti**
- Lo I/O scheduling:
 - Pone le richieste in arrivo in una **coda**
 - Seleziona l'ordine con cui servire le richieste
 - Ottimizza le **prestazioni**, evita **starvation**

Architettura del sistema di I/O



Device-independent I/O subsystem



Politiche di Scheduling del disco

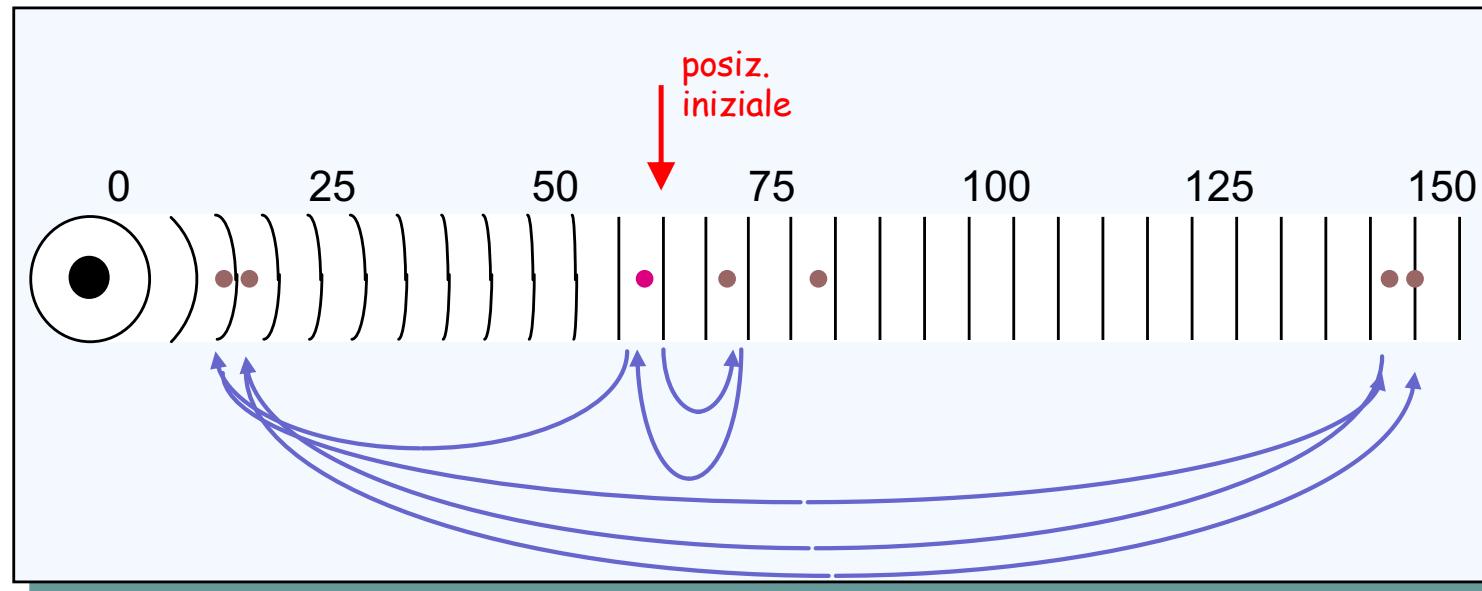
- 2 categorie:
 - Selezione secondo il richiedente:
 - FIFO, priorità, LIFO
 - Selezione secondo l'elemento richiesto:
 - SSTF, SCAN, C-SCAN, N-step-SCAN, FSCAN

Algoritmo FIFO

- First-in, first-out (FIFO)
 - Le richieste vengono servite in maniera sequenziale
 - Politica “fair” su tutti i processi
 - Ha basse prestazioni in presenza di un alto numero di richieste

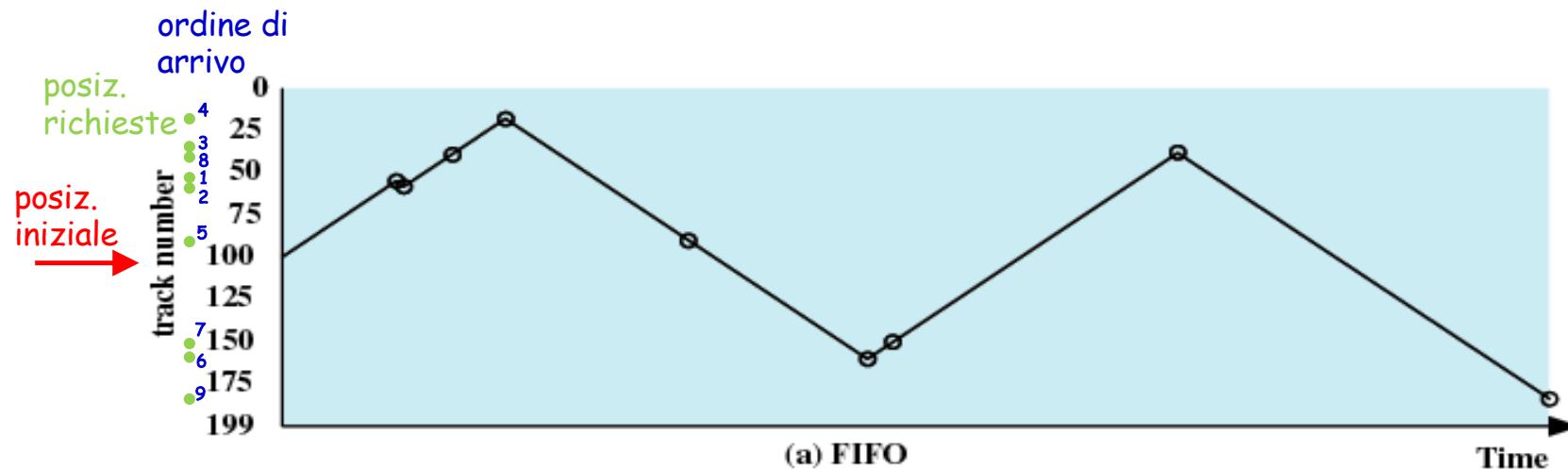
Algoritmo FIFO

Coda richieste: [83 72 14 147 16 150]



MOLTI MOVIMENTI SUPERFLUI DELLA TESTINA DEL DISCO!

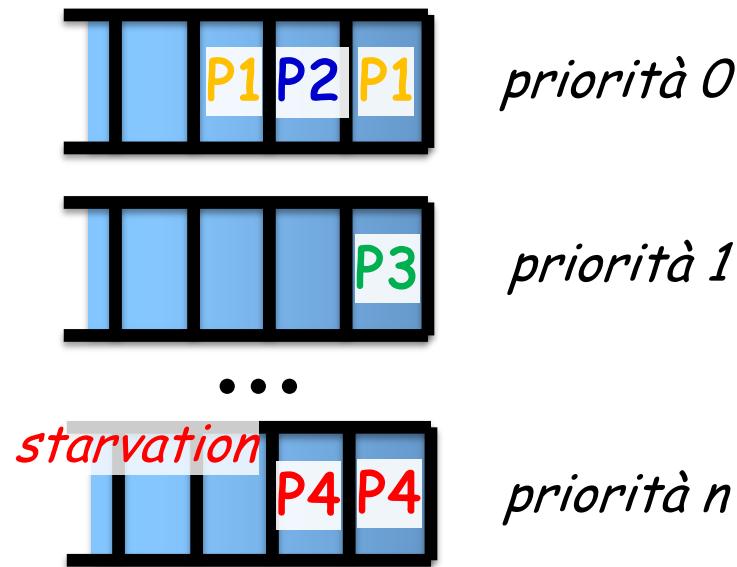
Algoritmo FIFO



Algoritmo a Priorità

- **Algoritmo a Priorità**

- Come FIFO, ma i processi sono organizzati in **code diverse** a seconda della loro priorità
- Ad esempio, si privilegiano i processi interattivi
- Possibile **starvation** per processi a bassa priorità



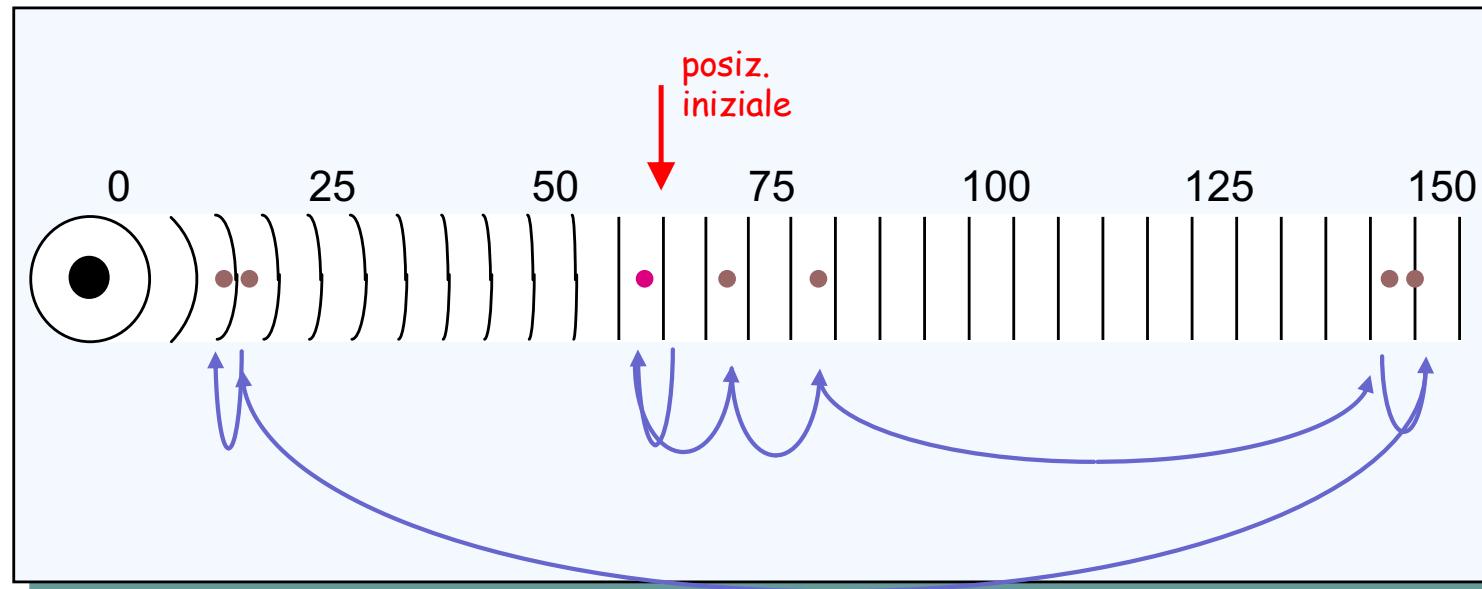
Algoritmo SSTF

- Shortest Service Time First (SSTF)
 - Seleziona la richiesta che richiede il **minimo movimento** dei braccetti del disco rispetto alla **posizione corrente (minimo seek time)**
 - Ottime prestazioni, ma vulnerabile alla **starvation**

Algoritmo FIFO

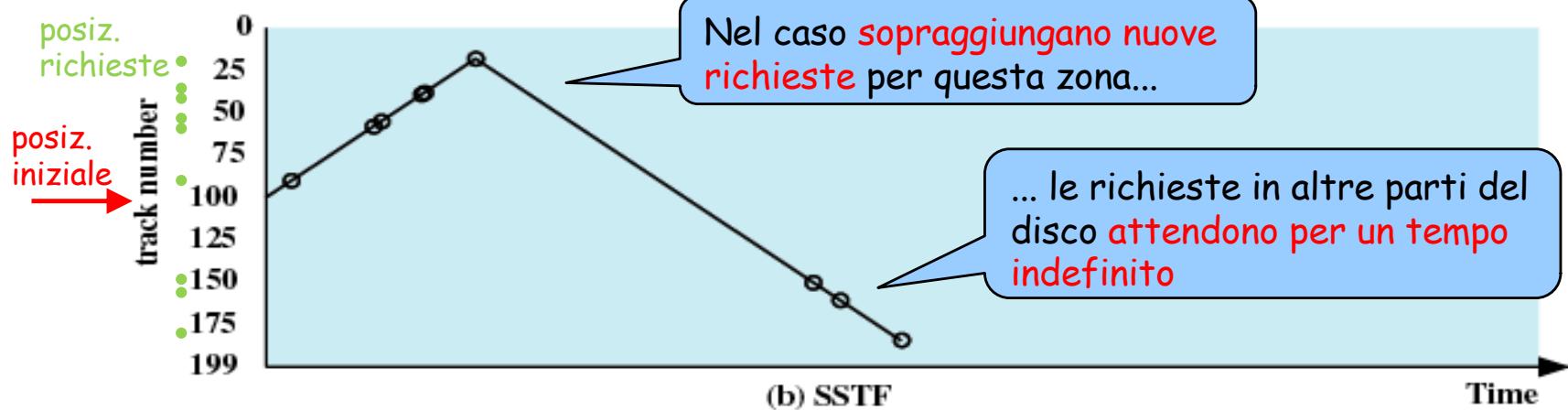
Coda richieste:

83	72	14	147	16	150
----	----	----	-----	----	-----



SE IN UNA ZONA DEL DISCO ARRIVANO MOLTE RICHIESTE,
L'ALGORITMO "TRASCURA" LE ALTRE RICHEISTE

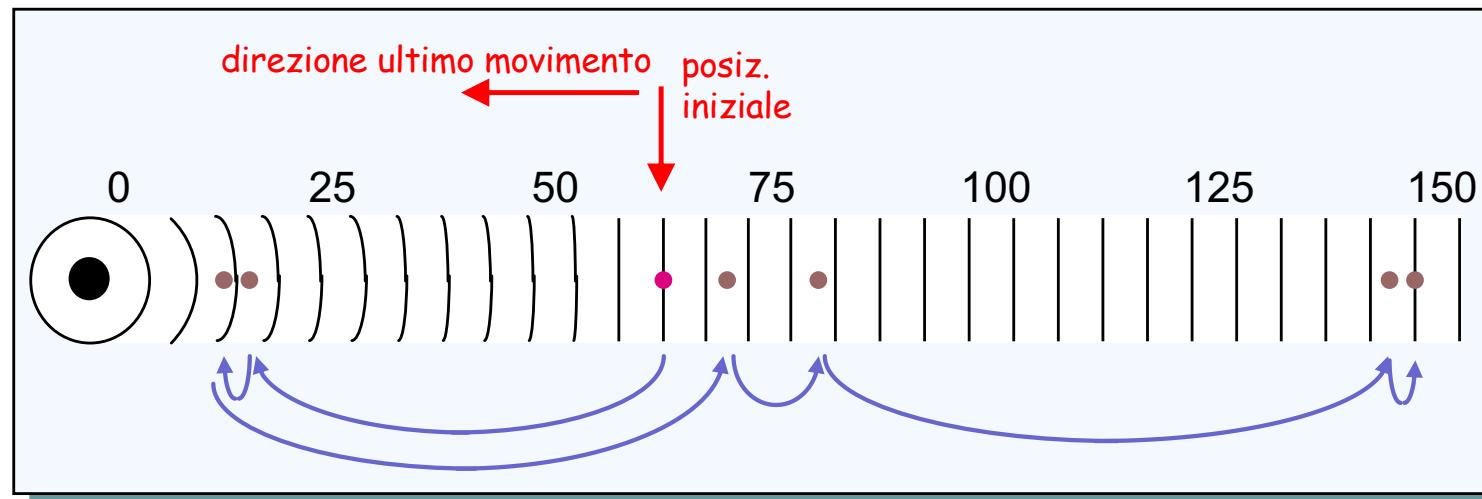
Algoritmo SSTF



L'algoritmo dell'ascensore

- **SCAN (detto anche Algoritmo dell'ascensore)**
 - I bracci del disco si muovono **solo in una direzione**, soddisfacendo tutte le richieste pendenti
 - Il braccio si muove nella direzione fino a raggiungere l'ultima traccia
 - La direzione viene poi invertita

Coda richieste: 



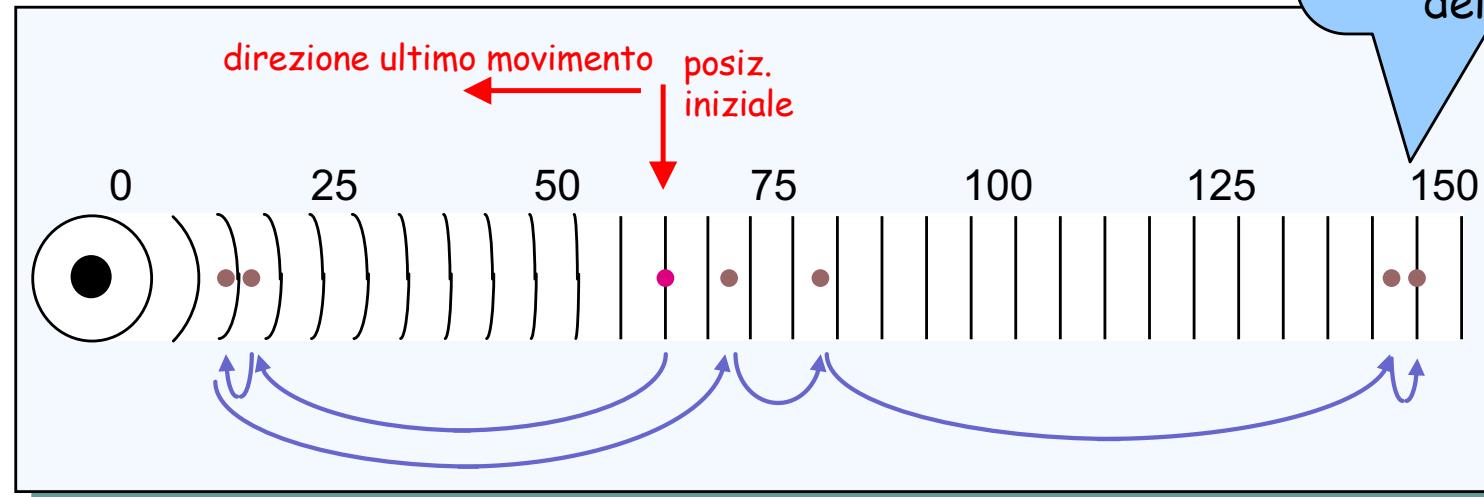
L'algoritmo dell'ascensore

- **SCAN** (detto anche **Algoritmo dell'ascensore**)
 - L'algoritmo opera **ordinando** le richieste in base alla posizione

Coda richieste: 

Ordinamento: 

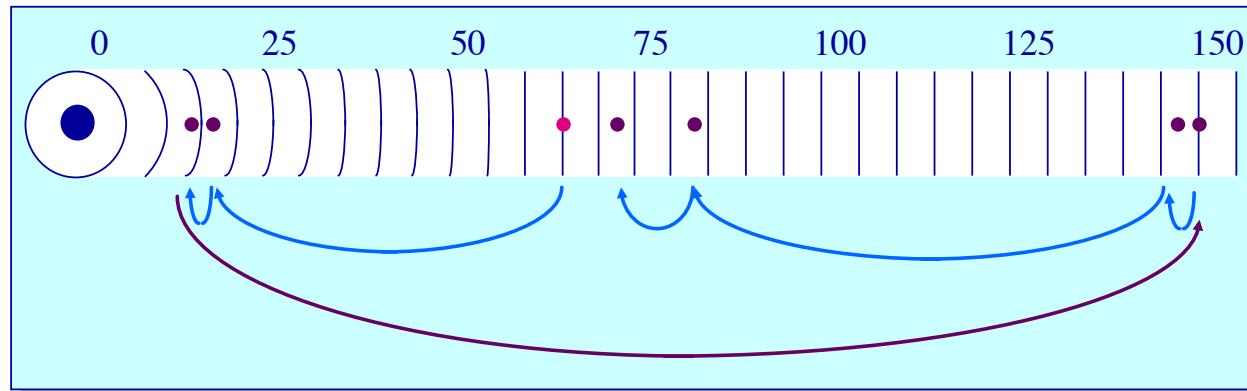
La **starvation** viene ridotta, ma è ancora possibile (es. alle estremità del disco)



Politiche di Scheduling del disco

- **C-SCAN (Circular SCAN)**

- Applica lo scanning secondo una **unica direzione**
- Quando viene raggiunta l'**ultima traccia**, i braccetti del disco portano la testina alla **parte opposta** del disco, ricominciando così lo scanning

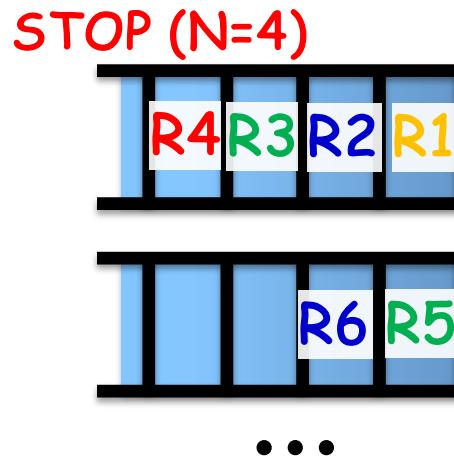


Politiche di Scheduling del disco

- N-step-SCAN

- Più code di richieste, di lunghezza N
- Le code sono servite una alla volta, con SCAN
- Mentre la coda corrente è servita, le nuove richieste sono accumulate in un'altra coda

N=4



SCAN opera sulla prima coda...

Politiche di Scheduling del disco

- **FSCAN** ("freeze" scan)
 - Due code
 - Le code non hanno un limite numerico
 - All'inizio di uno scan, serve le richieste di una coda
 - L'altra coda è inizialmente vuota ed è dedicata alle nuove richieste entranti

Politiche di Scheduling del disco

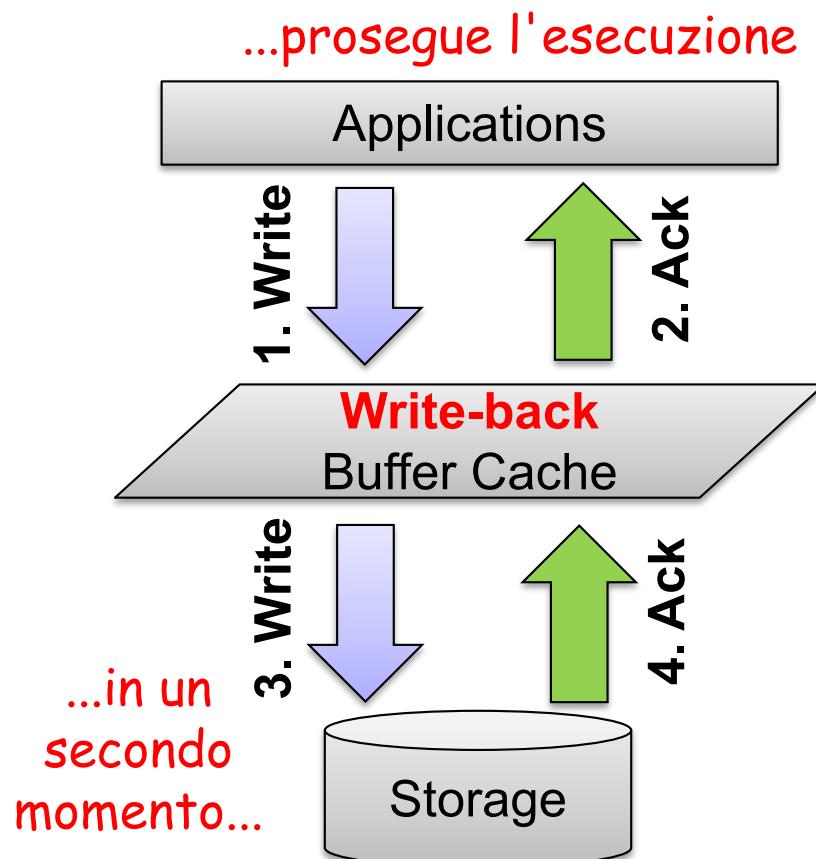
Table 11.2 Comparison of Disk Scheduling Algorithms

(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	<u>146</u>	184	<u>24</u>	18	<u>20</u>	90	<u>32</u>
Average seek length	55.3	Average seek length	27.5	Average seek length	27.8	Average seek length	35.8

Scheduling del disco in Linux: Linus Elevator

- Linux ha adottato dalla v2.4 l'algoritmo **C-SCAN** ("Linus Elevator")
 - **I/O sorting**: ordina le operazioni secondo il blocco richiesto
 - **I/O merging**: unifica due operazioni su due blocchi vicini
- Gli sviluppatori di Linux riscontrarono una particolare forma di starvation, detta **writes-starving-reads**
 - La quantità di **scritture** è molto superiore alle **letture**
 - Le letture sono penalizzate

Writes-starving-reads



Le scritture sono **bufferizzate** (es. politica write-back).

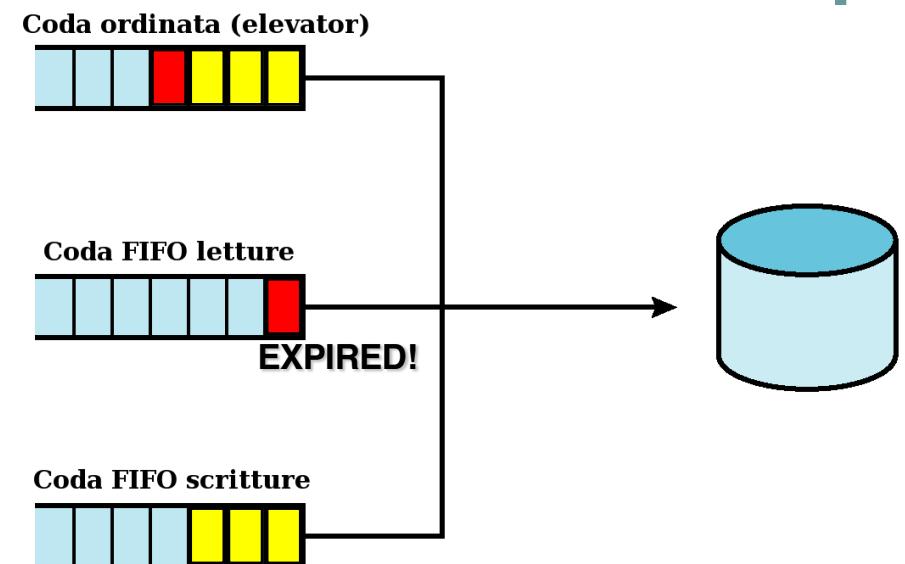
Un processo può fare molte scritture **senza sospendersi** per attendere la fine delle operazioni

Le letture sono **non sono bufferizzabili** (op. sincrone).

Un processo che legge si deve **sospendere**, poiché ha bisogno del dato per continuare ad eseguire.

Il deadline scheduler (Linux 2.6)

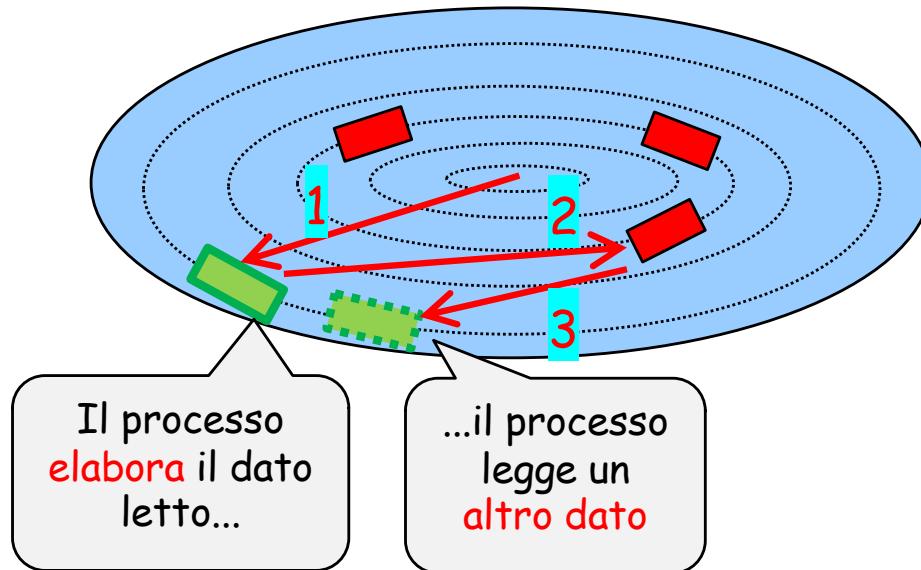
- Linus Elevator modificato
- In aggiunta, le richieste sono inserite **anche in una di 2 code FIFO (*lettura e scrittura*)**
- Ogni coda FIFO ha un **tempo massimo di attesa (deadline)**
 - 500ms per read
 - 5s per write, per favorire le read
- Se la deadline scade, la richiesta è servita immediatamente



Think time delle letture

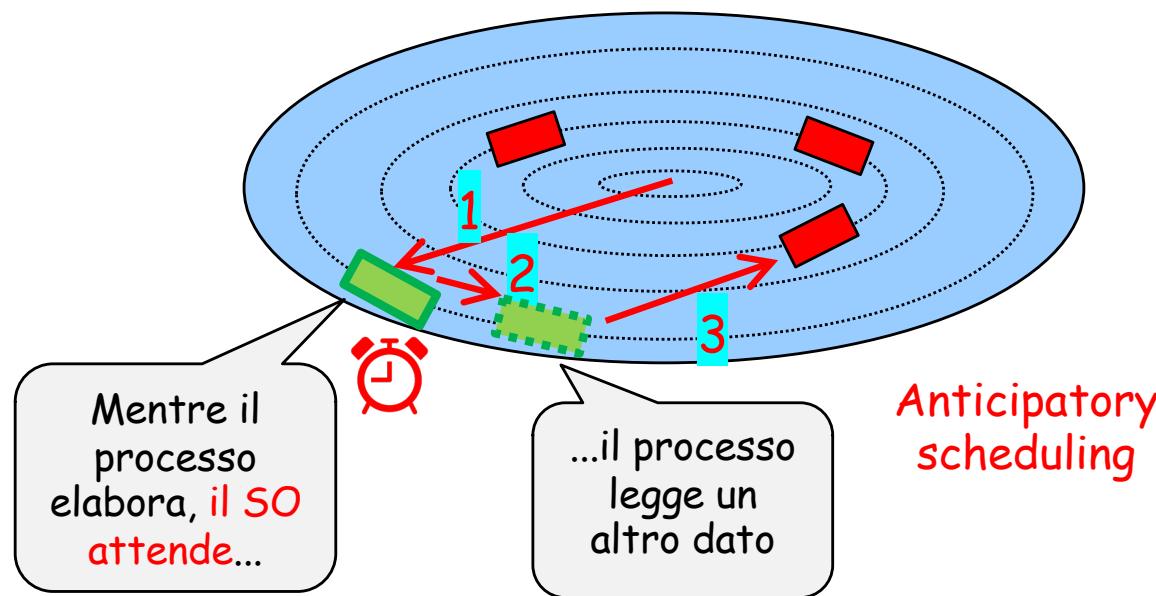
- Ulteriore problema con le letture:
 1. dopo aver letto un blocco, il processo si riattiva
 2. il processo elabora il dato letto ("think time")
 3. nel frattempo, il braccio del disco si sposta
 4. il lettore effettua una nuova lettura, vicina al blocco appena letto in precedenza
 5. La nuova lettura subisce lunghi tempi di seek

Think time delle letture



Anticipatory scheduling

- Con il meccanismo **anticipatorio**, dopo che una lettura è terminata, **attende senza servire altre richieste**
- Il disco rimane inutilizzato per brevi periodi
- Le prestazioni complessive migliorano



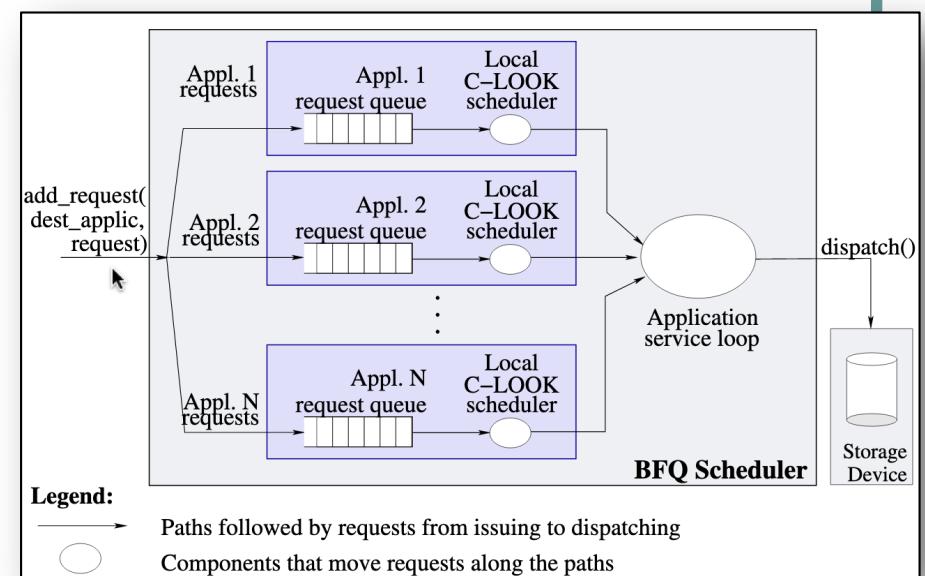
Writes-starving-reads

	Deadline + Anticipatory	Linus Elevator
Tempo medio di compilazione dei sorgenti del kernel Linux	7.149 s	55.057 s

L'algoritmo anticipatorio previene il fenomeno del **writes-starving-reads**, conseguendo dei tempi di servizio inferiori di quasi un ordine di grandezza.

Budget Fair Queueing (BFQ)

- Le versioni recenti di Linux usano **BFQ**, che incorpora le tecniche degli algoritmi precedenti
- Le richieste sono separate su **più code**, una per processo
- Ogni processo ha un "budget" di **numero di settori da attraversare**, proporzionale alla priorità

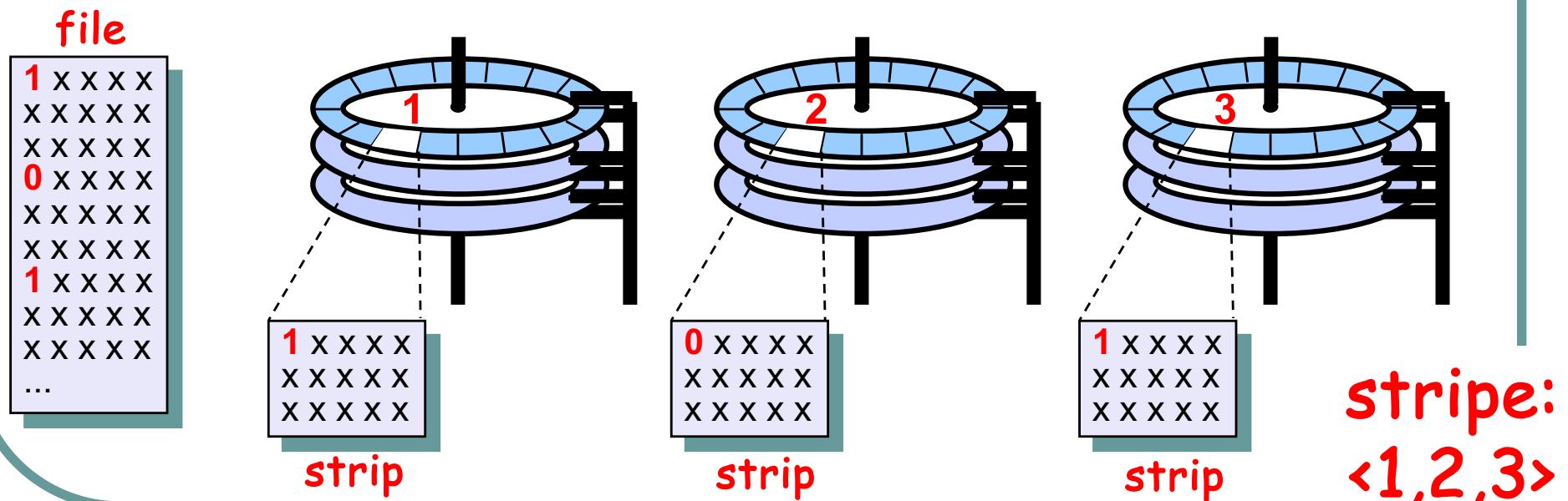


RAID

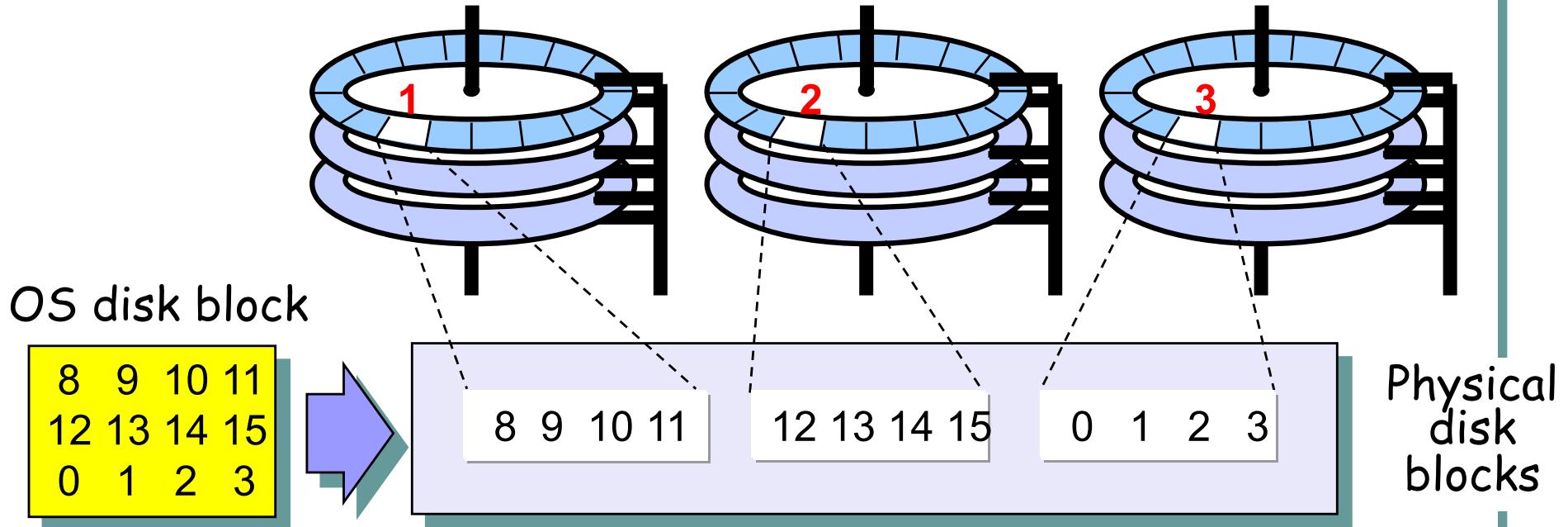
- Redundant Array of Independent Disks
- Un insieme di dischi, visto dalle applicazioni come una unica unità logica
- Perchè dischi ridondanti?
 - Aumentare le prestazioni (accesso in parallelo)
 - Aumentare l'affidabilità (ridondanza dei dati)
- Parte dello spazio dei dischi è usata per varie forme di ridondanza dei dati

Data striping

- I dati di ogni **blocco logico** sono divisi su più **blocchi fisici ("strip" o "chunk")** di stessa dimensione, e distribuiti su più dischi
- Le **strip** possono essere **ridondanti**
- Una **stripe** è un insieme di **strip** su posizioni omologhe



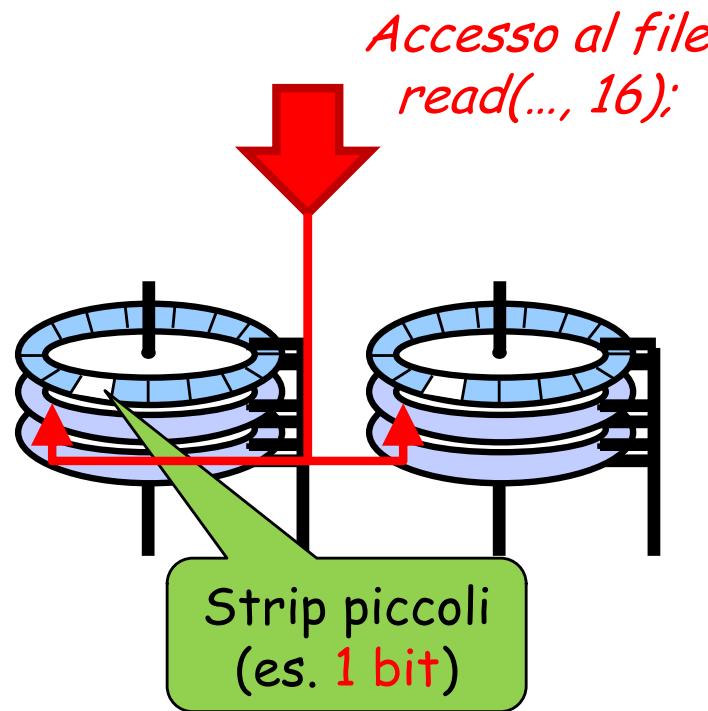
RAID 0 (non-ridondante)



- Data striping **senza ridondanza**
- Se un disco è guasto, la sua porzione di dati viene **persa**
- Migliori prestazioni rispetto a un disco singolo, mediante **lavoro parallelo** su più dischi
- Dischi fisici = Quantità di dati logici = N

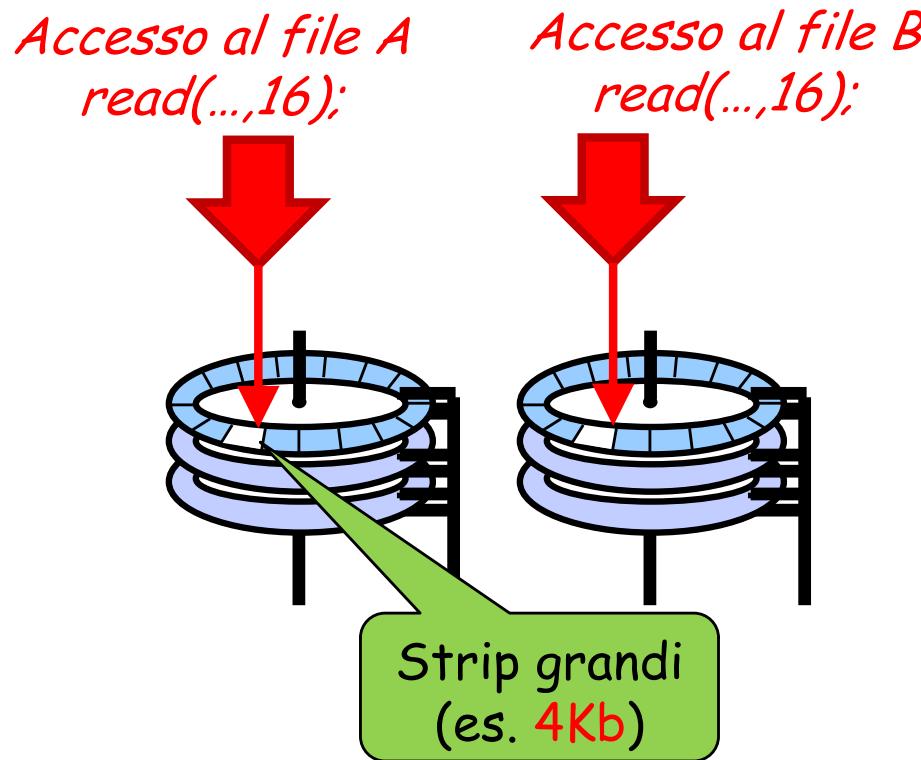
Dimensionamento degli strip

- Con **strip piccoli**, un singolo file tende ad essere sparpagliato su più dischi
 - riduce il tempo medio di servizio delle singole richieste

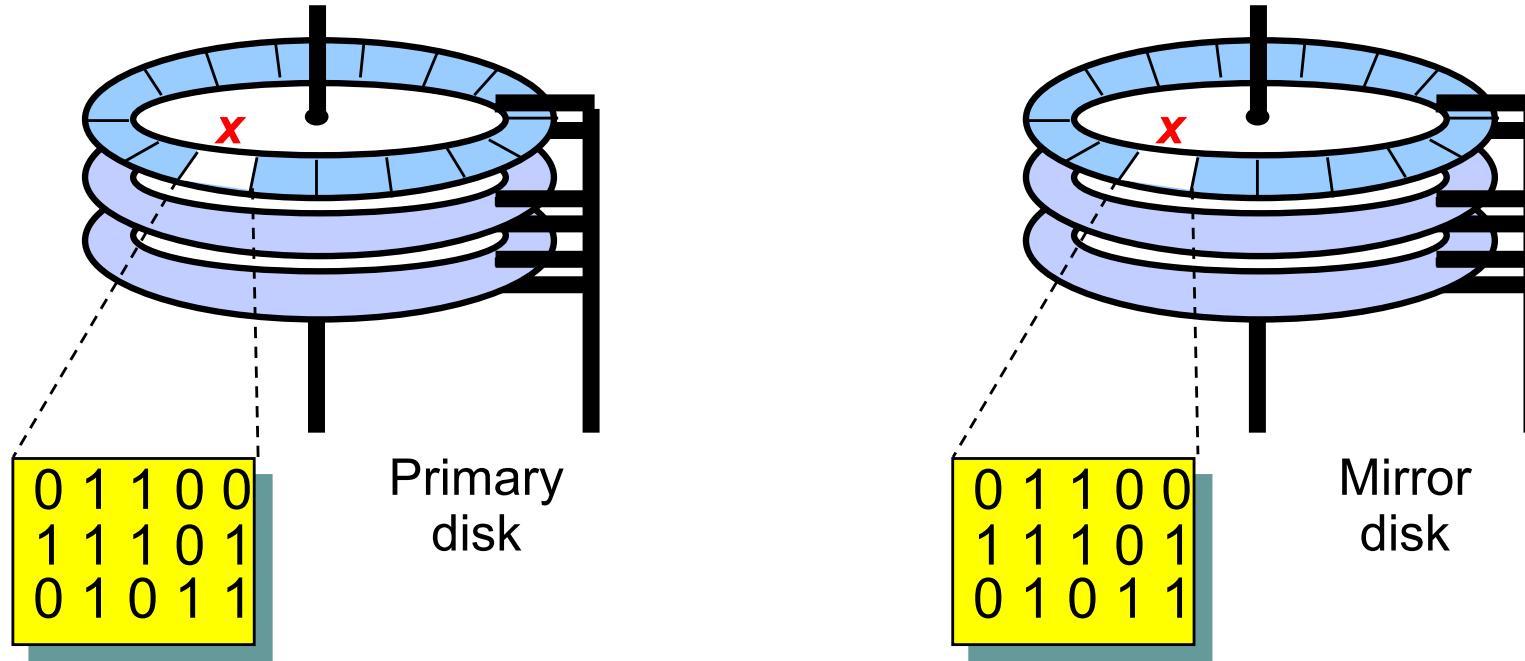


Dimensionamento degli strip

- Con **strip grandi**, un file può entrare per intero in un singolo strip su un singolo disco
 - **massimizza il throughput** di richieste servite

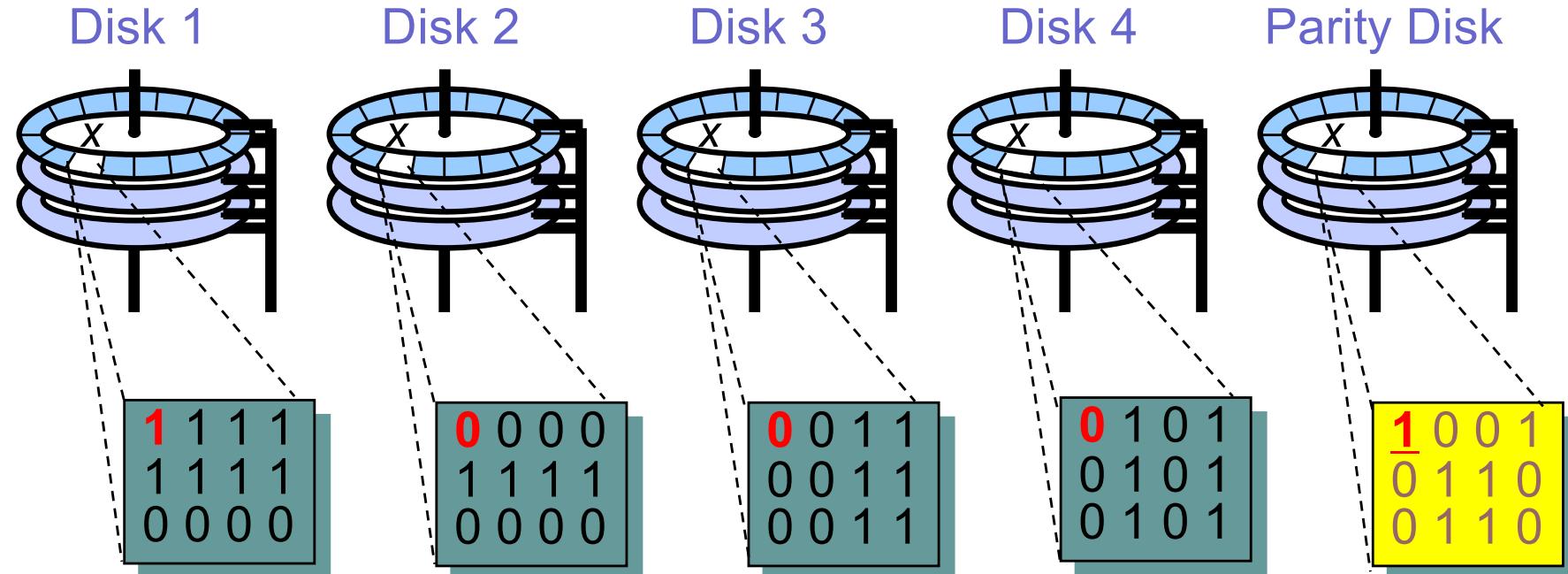


RAID 1 (mirroring)



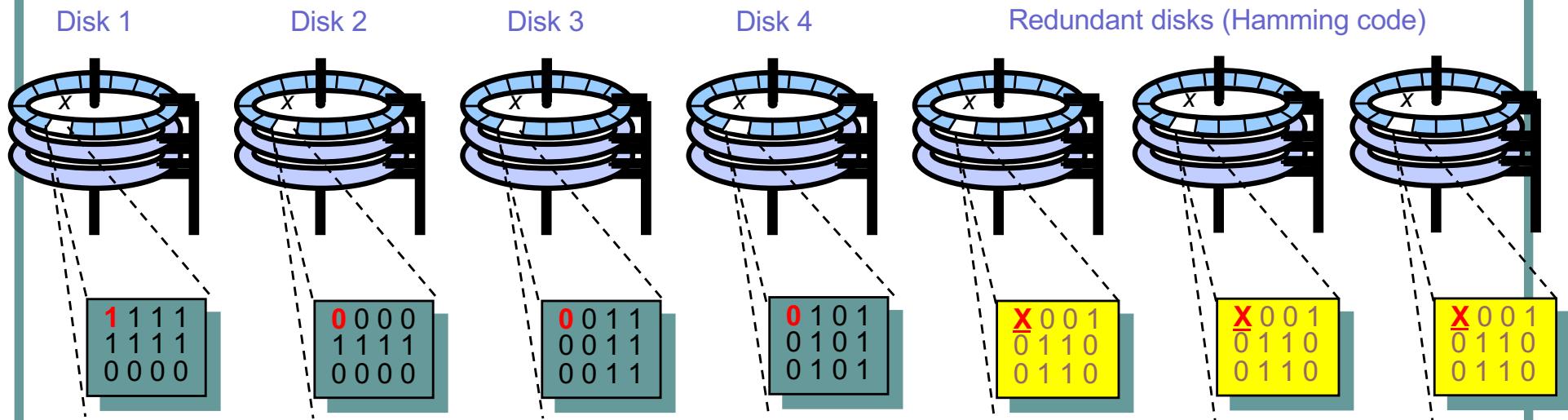
- Ridondanza dei dati con tecnica di **mirroring**
- Buona affidabilità (migliore di RAID 2,3,4,5; minore di RAID 6)
- Migliora throughput e tempi di servizio, evita le penalità degli altri RAID
- Dischi richiesti = $2N$

RAID 3 (parità bit-interleaved)



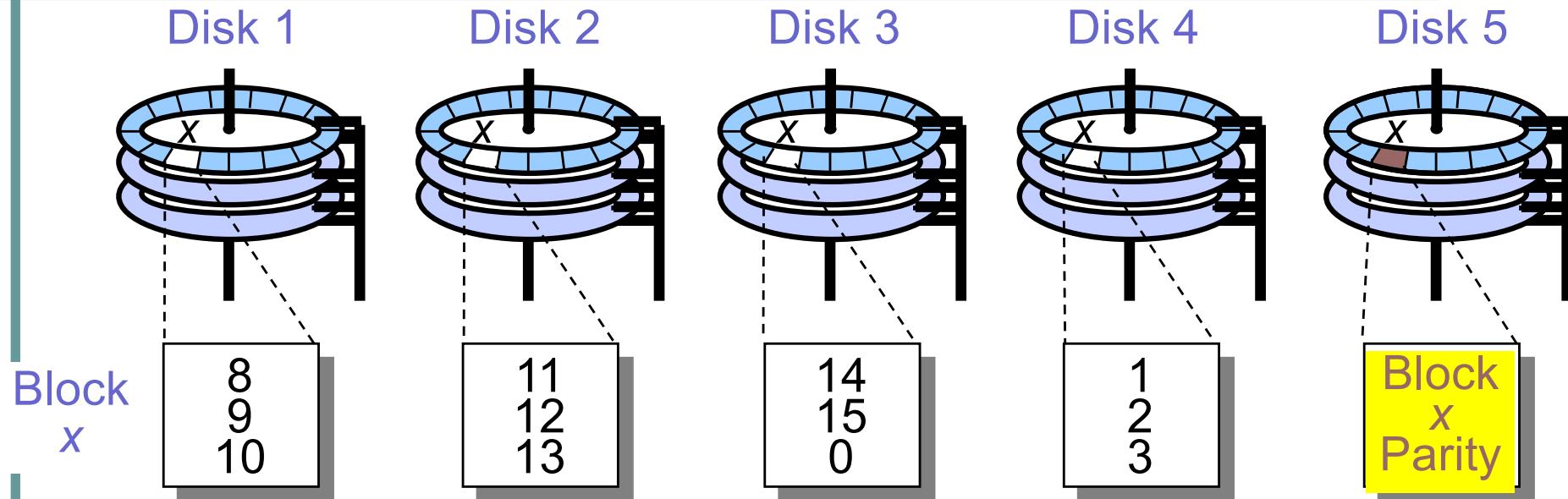
- Striping a livello di bit
- Ridondanza dei dati con **bit di parità**
- Buona affidabilità (comparabile a RAID 4 o 5)
- Basso **tempo di servizio**, tutti i dischi sono **acceduti in parallelo**
- Dischi richiesti = N+1

RAID 2 (codici di Hamming)



- Striping a livello di bit
- Ridondanza dei dati con **codici di Hamming**
- Elevata **affidabilità** (migliore di RAID 3,4)
- Può rilevare **guasti in più di un disco**, ma è considerata una soluzione eccessiva se i singoli dischi sono molto affidabili
- Basso **tempo di servizio**, come RAID 3
- Dischi richiesti = $N+m$

RAID 4 (parità a livello di blocco)



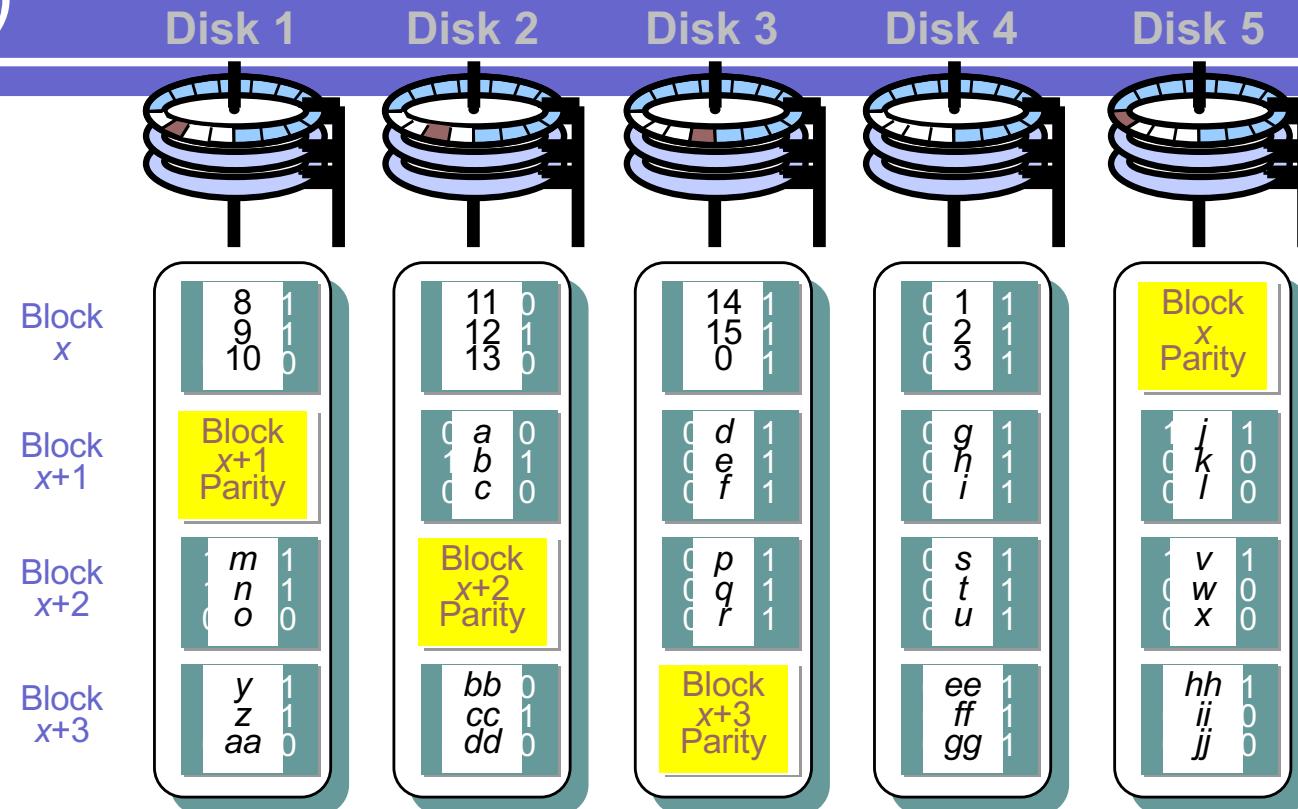
- Striping su **blocchi di dati**
- Ridondanza dei dati con bit di parità per blocco
- Buona affidabilità (comparabile con RAID 2,3,5)
- Throughput elevato per le letture (parallelismo), simile al RAID 0
- **Penalizza le scritture** (il disco di parità è un **collo di bottiglia**)
- Dischi richiesti = N+1

Il problema del read-modify-write

- Quando si utilizza un bit di parità, è necessario effettuare **4 operazioni di I/O** per ogni scrittura:
 - Lettura della **strip dati originaria**
 - Lettura della **strip di parità**
 - Scrittura della **nuova strip dati**
 - Scrittura della **nuova strip di parità**

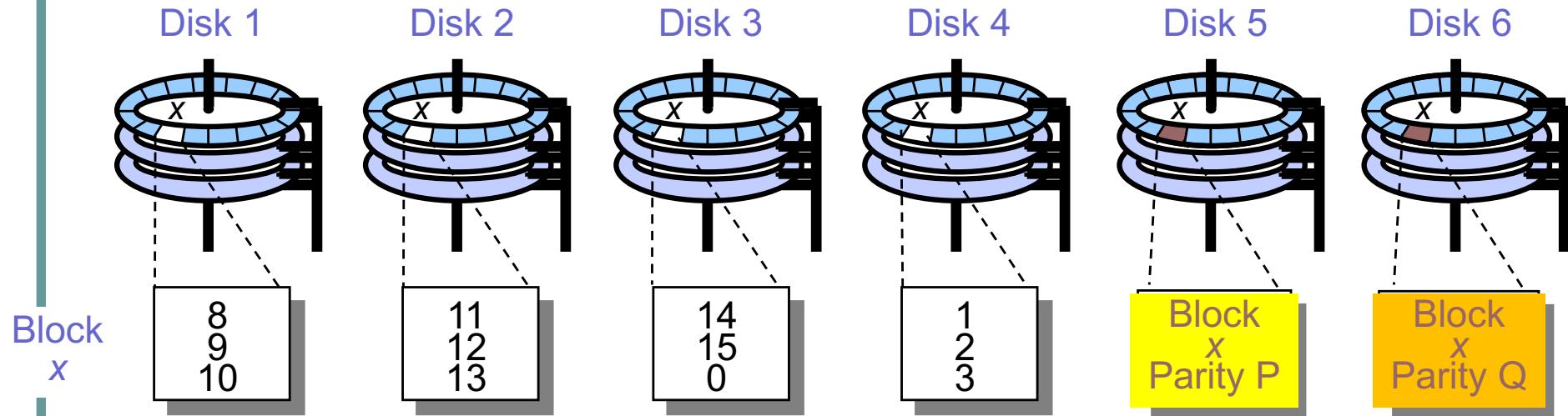
$$X'_P(i) = X_P(i) \oplus X_D(i) \oplus X'_D(i)$$

RAID 5 (parità a livello di blocco, distribuita tra i dischi)



- Ridondanza dei dati con bit di parità per blocco **distribuiti su più dischi**
- Buona affidabilità (comparabile con RAID 2,3 o 4)
- In lettura, la capacità di trasferimento è simile al RAID 0
- La scrittura è penalizzata
- Dischi richiesti = N+1

RAID 6 (ridondanza duale)



- Ridondanza con doppio bit di parità per blocco, con due schemi differenti P e Q, e **distribuiti su più dischi**
- Tra le soluzioni presentate, RAID-6 è quella che ha il miglior livello di affidabilità
- In lettura la capacità di trasferimento è simile al RAID 0, la scrittura è penalizzata (è il peggiore di tutti per le prestazioni in scrittura)
- Dischi richiesti = N+2

Livelli di RAID

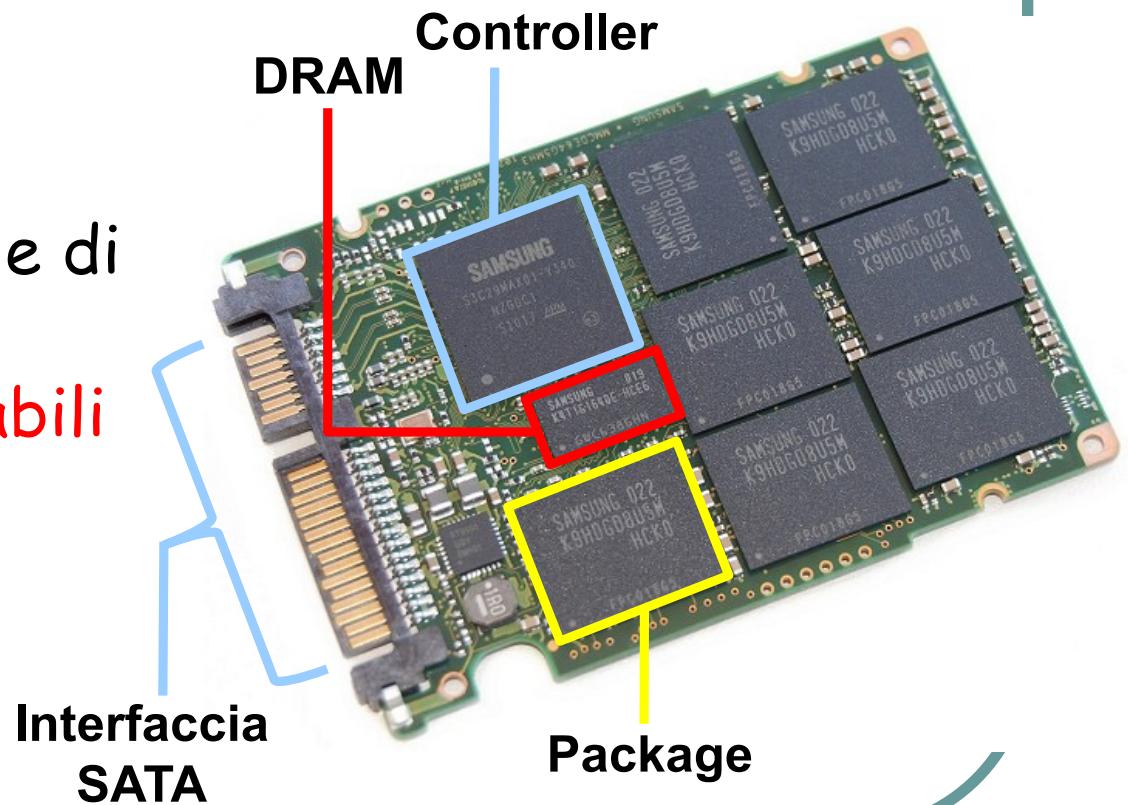
Table 11.4 RAID Levels

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

N = number of data disks; m proportional to $\log N$

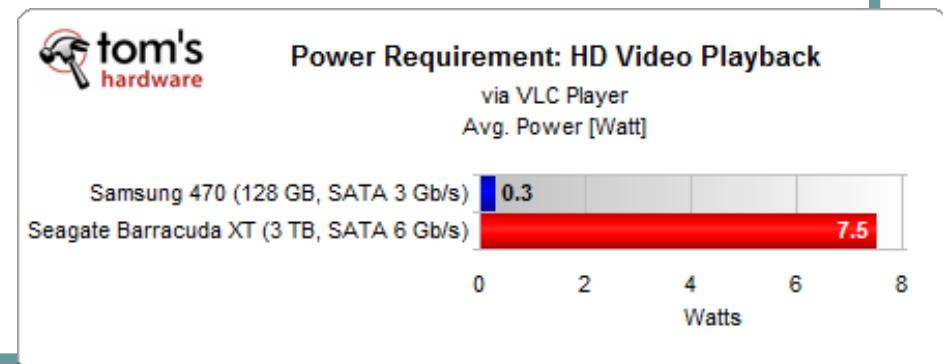
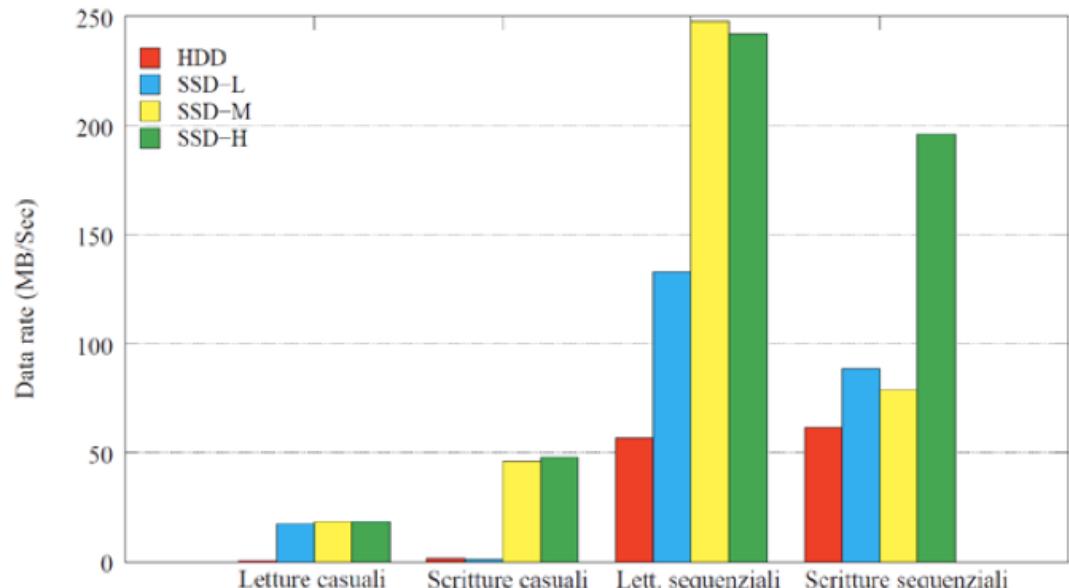
Solid-state drives (SSD)

- I "dischi" a stato solido (**SSD**) sono dispositivi di memorizzazione non volatili basati su tecnologia *Flash*
- Un SSD è fatto di celle di transistor ad **accesso random e riprogrammabili**

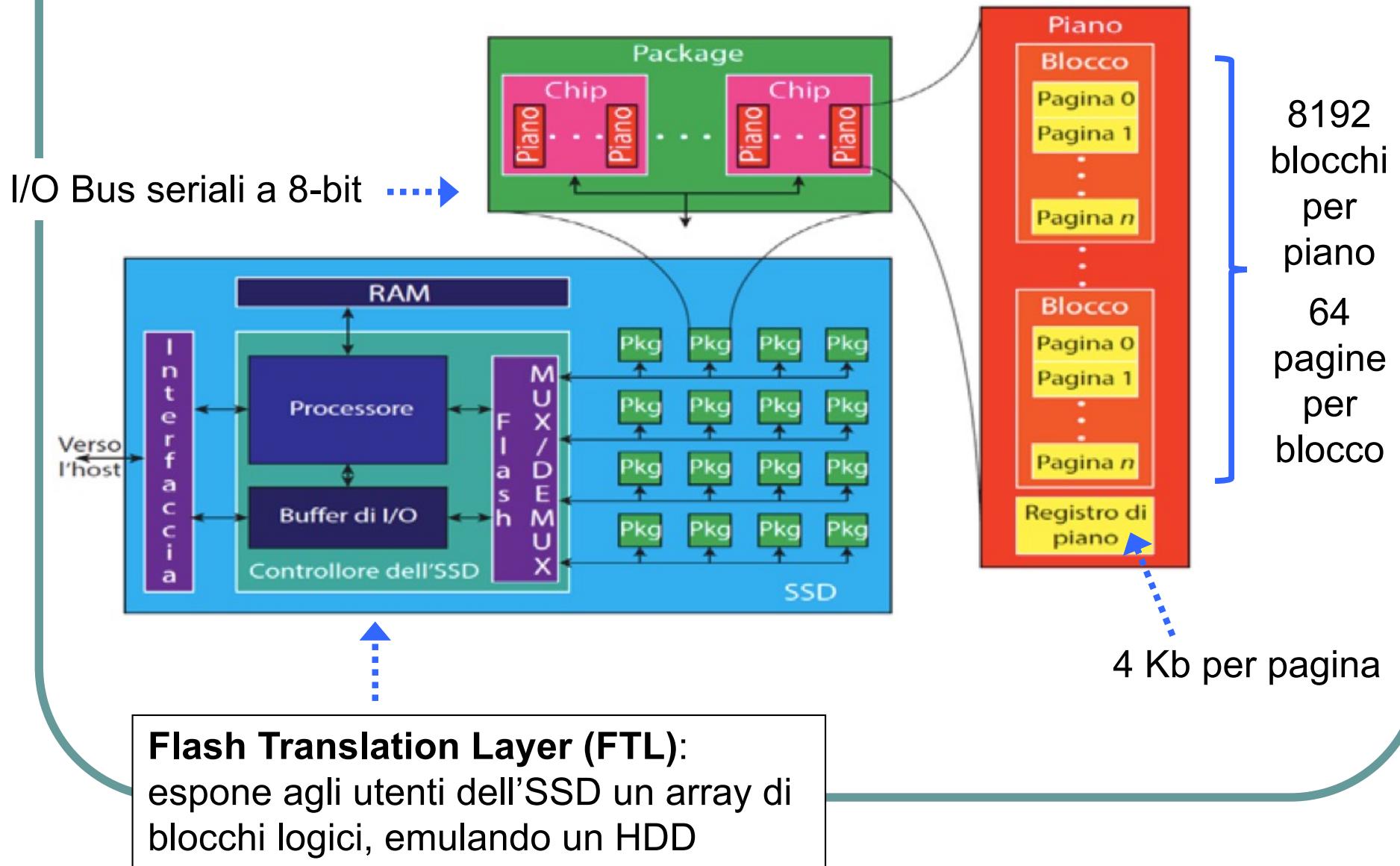


SSD: Caratteristiche

- Confronto tra HDD (hard-disk) e SSD:
 - Migliori prestazioni
 - Maggiore robustezza fisica
 - Risparmio energetico
 - Costi più elevati
 - Usura delle celle
- Sono oggi utilizzati come **alternativa** agli HDD nei dispositivi portabili e PC, e come **cache** negli HDD ibridi



Architettura di un SSD



Il “contratto non scritto” tra il SO e i dispositivi di storage (HDD)

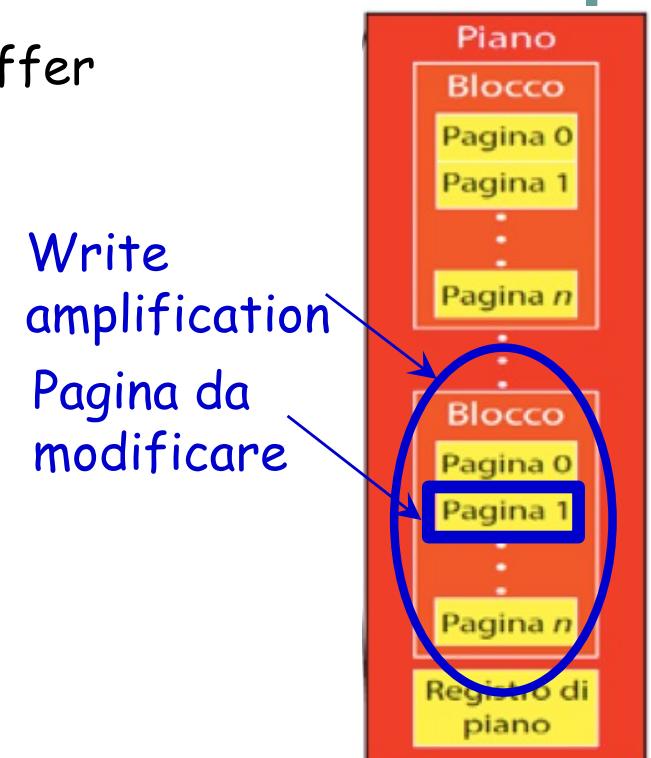
- I SO sono progettati in base alle seguenti **ipotesi implicite** sul comportamento degli HDD:
 1. Gli **accessi sequenziali** sono molto più performanti, sia per le letture sia per le scritture
 2. Il tempo per accedere ad un settore **aumenta con la distanza** dall'ultimo settore acceduto
 3. Viene **scritto solo il settore** che è strettamente richiesto
 4. La **vita del supporto** non dipende dal numero di operazioni effettuate
 5. La **attività di background** sul dispositivo è inesistente

Il “contratto non scritto” tra il SO e i dispositivi di storage (SSD)

- Alcune di queste ipotesi **non sono più valide** negli SSD:
 1. Gli accessi sequenziali sono molto più performanti, ma questo vale più per le scritture che per le letture
 2. Il tempo per accedere ad un settore non aumenta con la distanza dall'ultimo settore acceduto
 3. Viene scritto non solo il settore che è strettamente richiesto (**write amplification**)
 4. La vita del supporto non dipende dal numero di operazioni effettuate
 5. La attività di background sul dispositivo non è trascurabile

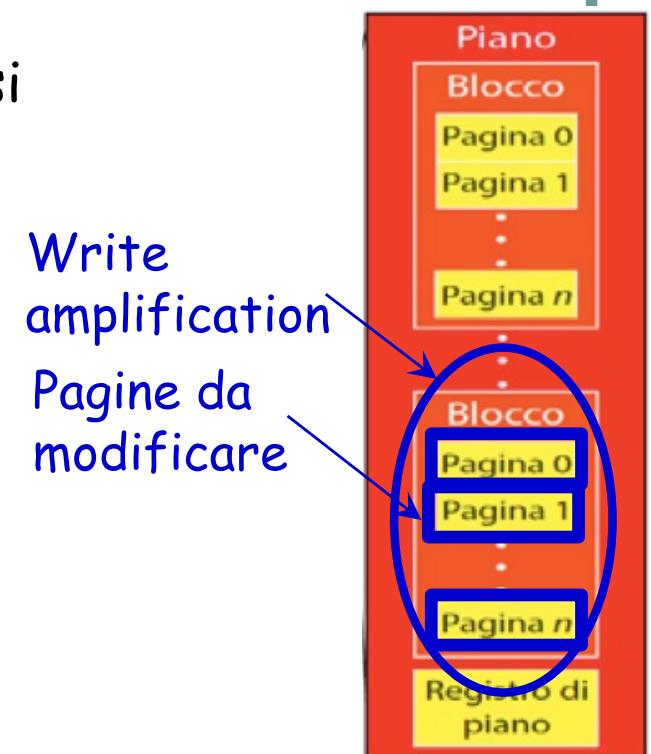
Write amplification

- Negli SSD, anche se si modifica una **singola pagina**, le scritture avvengono sui **interi blocchi** (**write amplification**):
 - lettura di un **intero blocco di pagine** in un buffer
 - modifica sul buffer
 - sovrascrittura dell'intero blocco
- Gli accessi **casuali (random)**, essi tendono a **peggiорare le prestazioni** e causare **usura**



Write amplification

- Quando si scrive **sequenzialmente** su tutto il blocco, si evita questo overhead
- Si **accumulano** molte scritture **vicine**, e le si scrive tutte insieme



Scheduling di I/O per SSD

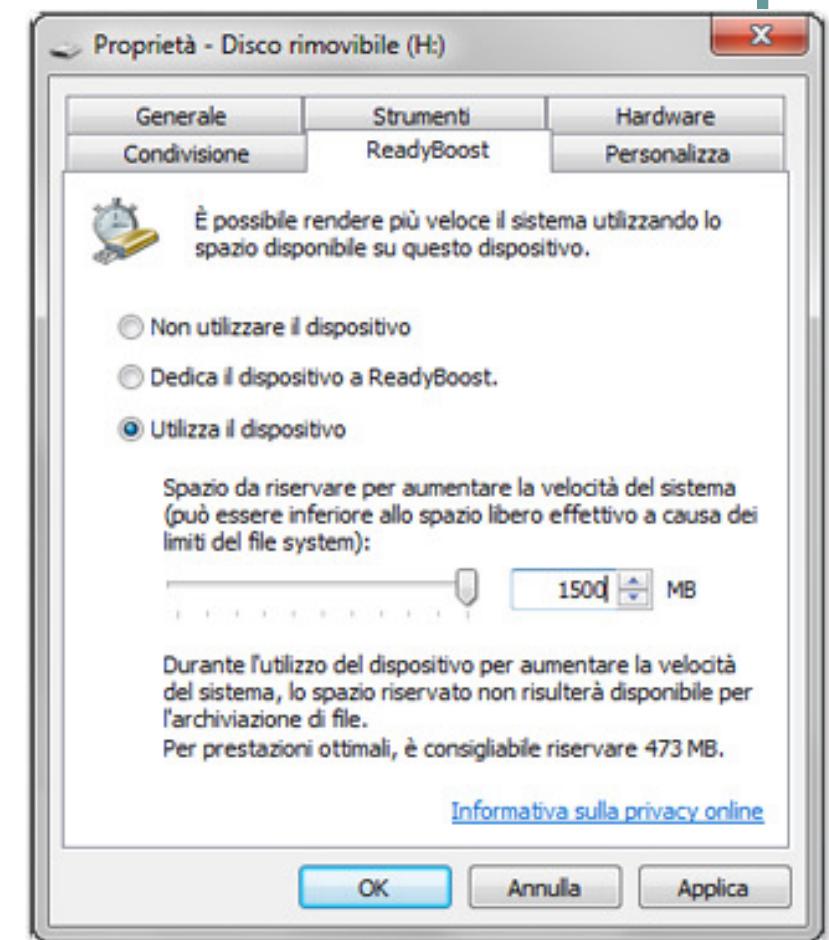
- Essendo le SSD ad accesso diretto, viene **meno la necessità di ordinare** gli accessi in base alla loro posizione
- È possibile anche utilizzare uno scheduler **FIFO**
- Il kernel Linux fornisce uno scheduler (opzionale) di tipo FIFO denominato ***noop (no-operation)***

Effetto dell'ascensore e dell'anticipazione

- L'ascensore può favorire la sequenzialità degli accessi
- L'anticipazione protegge dalla starvation i processi che effettuano letture
- Poiché gli SSD sono più veloci, occorre ridurre/disabilitare i tempi di attesa per non penalizzare gli altri processi

I/O Caching con SSD

- **HDD ibridi** includono una SSD come cache, invisibile al SO
 - Apple Fusion Drive
- È possibile usare un **SSD indipendente** come cache, affiancata ad un HDD
 - Windows ReadyBoost
 - Linux Bcache
- Permette un **risparmio di RAM** e la **persistenza** dei dati frequentemente acceduti



Quiz

1. Quali delle seguenti affermazioni riguardo il buffering di tipo "write-back" sono vere? (selezionare più di una)

- Migliora le prestazioni in lettura rispetto a "write-through", penalizzando l'affidabilità
- Migliora le prestazioni in scrittura rispetto a "write-through", penalizzando l'affidabilità
- Non migliora le prestazioni rispetto a "write-through", ma migliora l'affidabilità
- È una possibile causa del "writes-starving-reads"



<https://forms.office.com/r/8QBBsuWJnd>