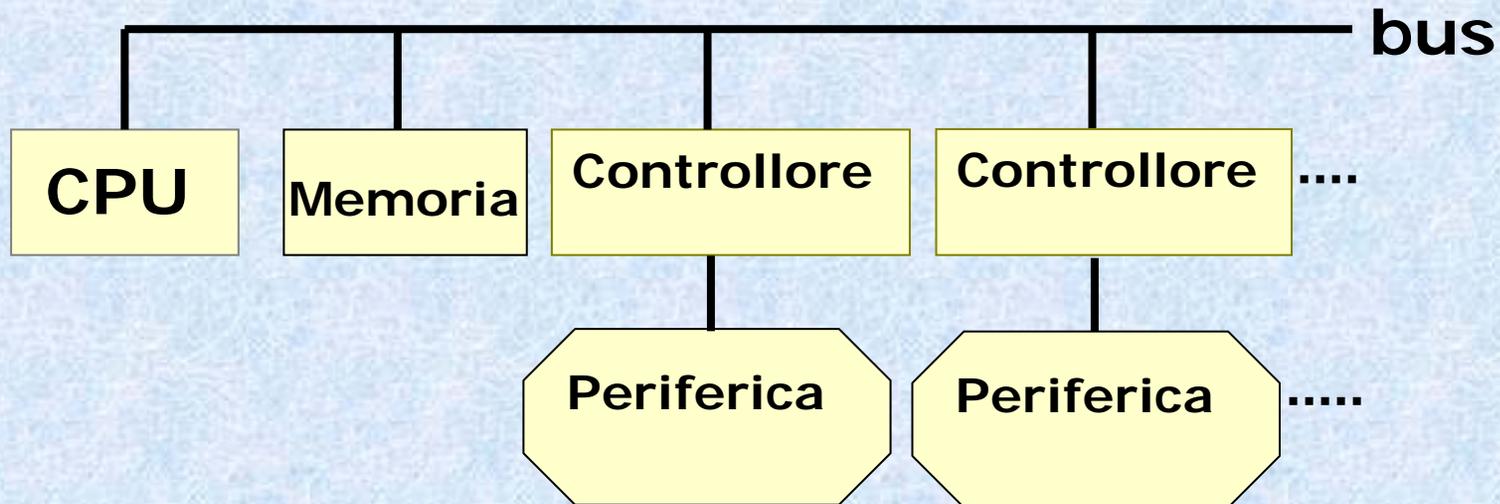


Gestione delle periferiche

- 5.1** **Concetti generali**
- 5.2** **Organizzazione del sottosistema di I/O**
- 5.3** **Gestione di un dispositivo**
- 5.4** **Gestione e organizzazione dei dischi**

Concetti generali

- **Compiti del sottosistema di I/O**
 - ✓ **Gestire i dispositivi, creando astrazioni che nascondono i dettagli dell'hardware e le particolarità dei controllori**



Caratteristiche dei dispositivi: velocità di trasferimento

dispositivo	velocità di trasferimento
tastiera	10 bytes/sec
mouse	100 bytes/sec
modem	10 Kbytes/sec
linea ISDN	16 Kbytes/sec
stampante laser	100 Kbytes/sec
scanner	400 Kbytes/sec
porta USB	1.5 Mbytes/sec
disco IDE (parallel ATA)	5 Mbytes/sec
CD-ROM	6 Mbytes/sec
Fast Ethernet	12.5 Mbytes/sec
FireWire (IEEE 1394)	50 Mbytes/sec
monitor XGA	60 Mbytes/sec
Ethernet gigabit	125 Mbytes/sec

Caratteristiche dei dispositivi: modalità di interazione

Interazioni controllate da interruzione

- il processore trasferisce singole unità di informazione dalla RAM alle periferiche e viceversa (*dispositivi a caratteri*)

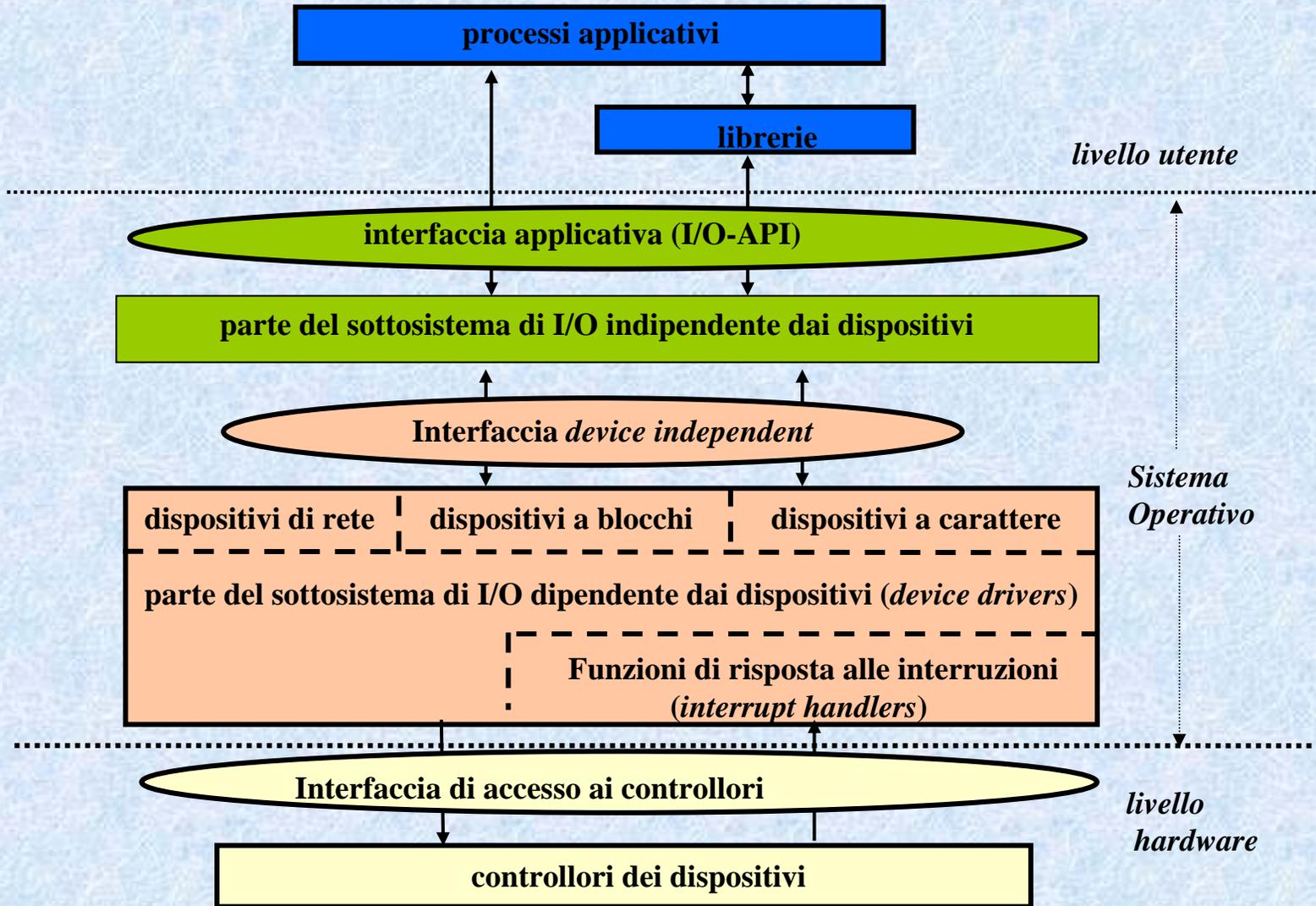
Interazioni con DMA (Direct Memory Access)

- l'interfaccia accede direttamente alla RAM per trasferire blocchi di dati (*dispositivi a blocchi*)
- possibilità di trasferire dati in memoria mentre il processore elabora
- Il controllore di DMA segnala il completamento dell'operazione con un' interruzione

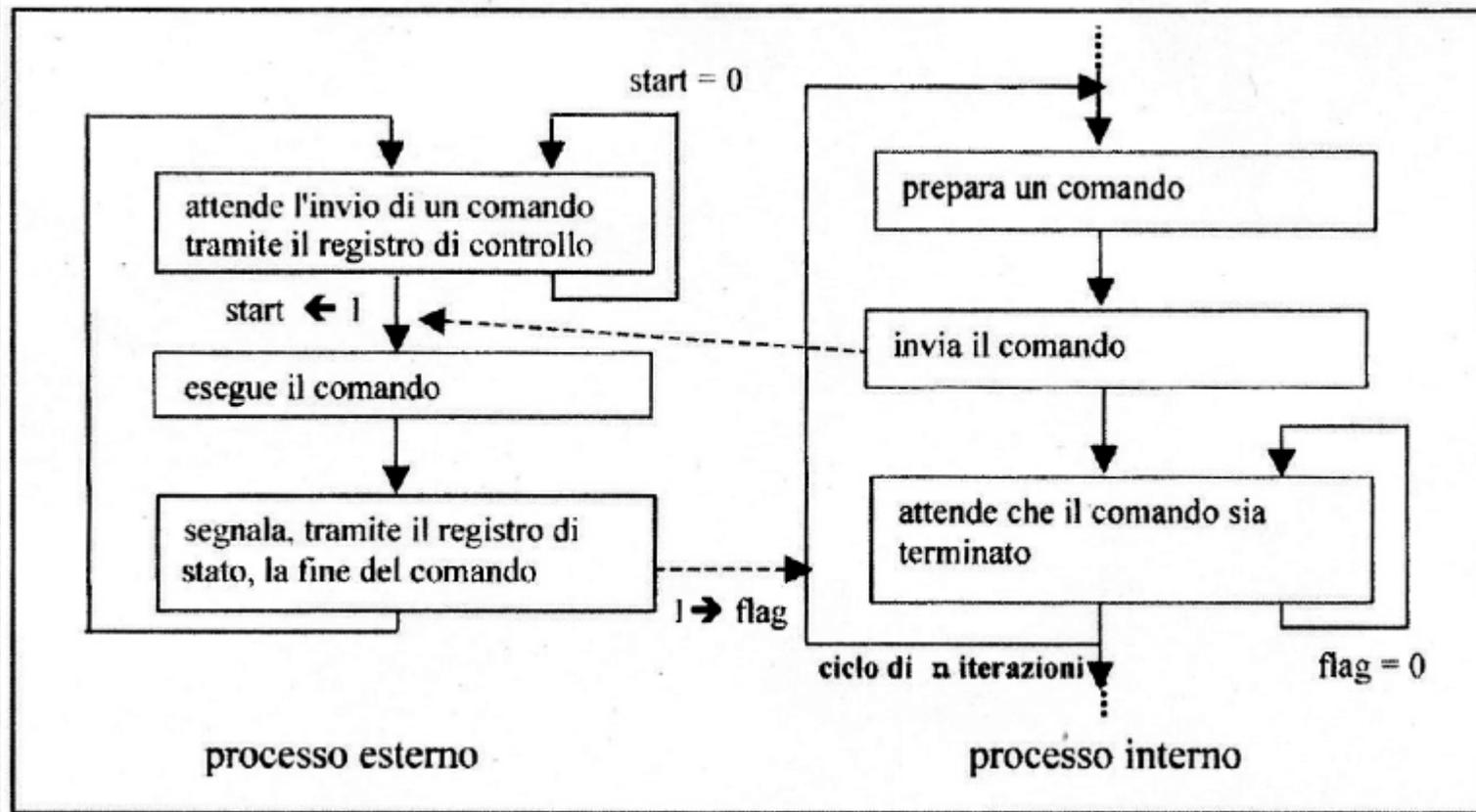
Concetti generali

- **Compiti del sottosistema di I/O**
 - ✓ **Definire lo spazio dei nomi con cui identificare i dispositivi.**
 - ✓ **Definire una interfaccia uniforme per i dispositivi, con diverse implementazioni per le differenti categorie (dispositivi a caratteri, a blocchi, di rete)**
 - ✓ **Realizzare la sincronizzazione tra l'attività di un dispositivo e quella del processo che lo ha attivato (driver del dispositivo)**
 - ✓ **Gestire i malfunzionamenti.**

Organizzazione logica del sottosistema di I/O



Astrazione del dispositivo come processo esterno



processo esterno:

```
{
  while (true) {
    do{}
    while(Start == 0); // attesa invio di un comando;
    <esegue comando>;
    <registra esito comando, ivi incluso flag=1>;
  }
}
```

Interazione con un dispositivo a caratteri

```
...
for (int i=0; i<n;i++)
{
    <prepara il comando>;
    // assembla in un registro interno alla
    // CPU il valore da trasferire nel
    // registro di controllo;

    <invia il comando>; // ponendo a uno il bit di start;

    do{;} while (flag == 0); // ciclo di attesa sul flag;

    <verifica l'esito>;
    // controlla tramite il registro di
    // stato che non ci siano stati errori e
    // preleva il dato letto dal registro
    // dati del controllore;
}
```

1) a controllo di programma

```
...
for (int i=0; i<n; i++) {
    <prepara il comando>;
    // assembla in un registro interno alla
    // CPU il valore da trasferire nel
    // registro di controllo;

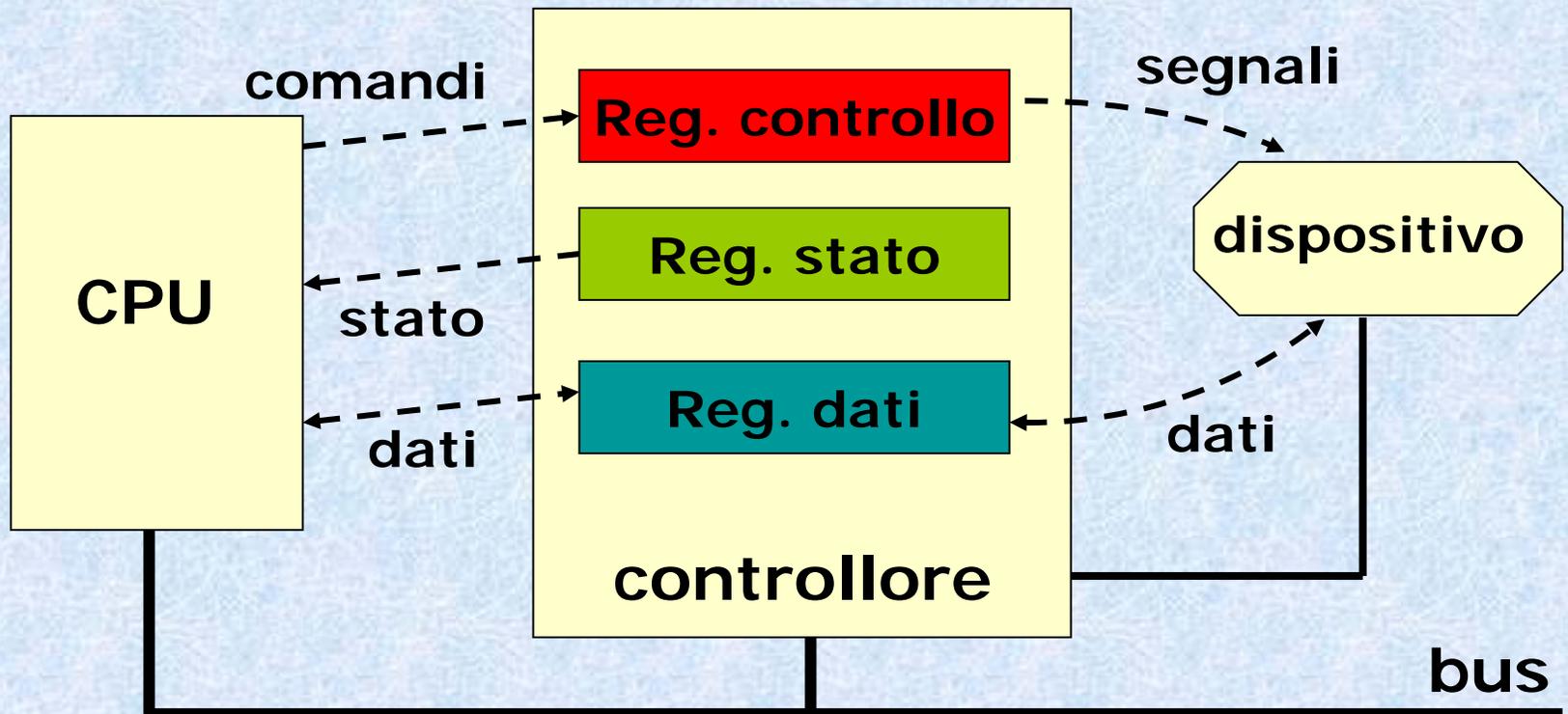
    <invia il comando>; // ponendo a uno il bit di start;

    dato_disponibile.wait(); // attesa dato;
    <verifica l'esito>; // controlla...
}
...
```

2) Mediante sistema
di interruzione

Livello dipendente dal dispositivo:

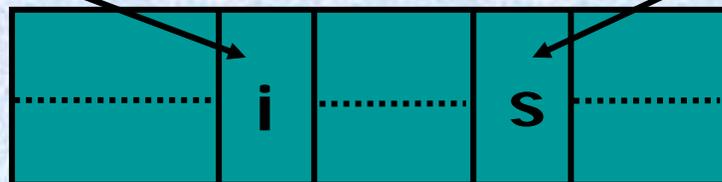
Interfaccia semplificata del controllore di un dispositivo a caratteri



Registri del controllore di un dispositivo a caratteri

bit di abilitazione
alle interruzioni

bit di start



Registro di controllo

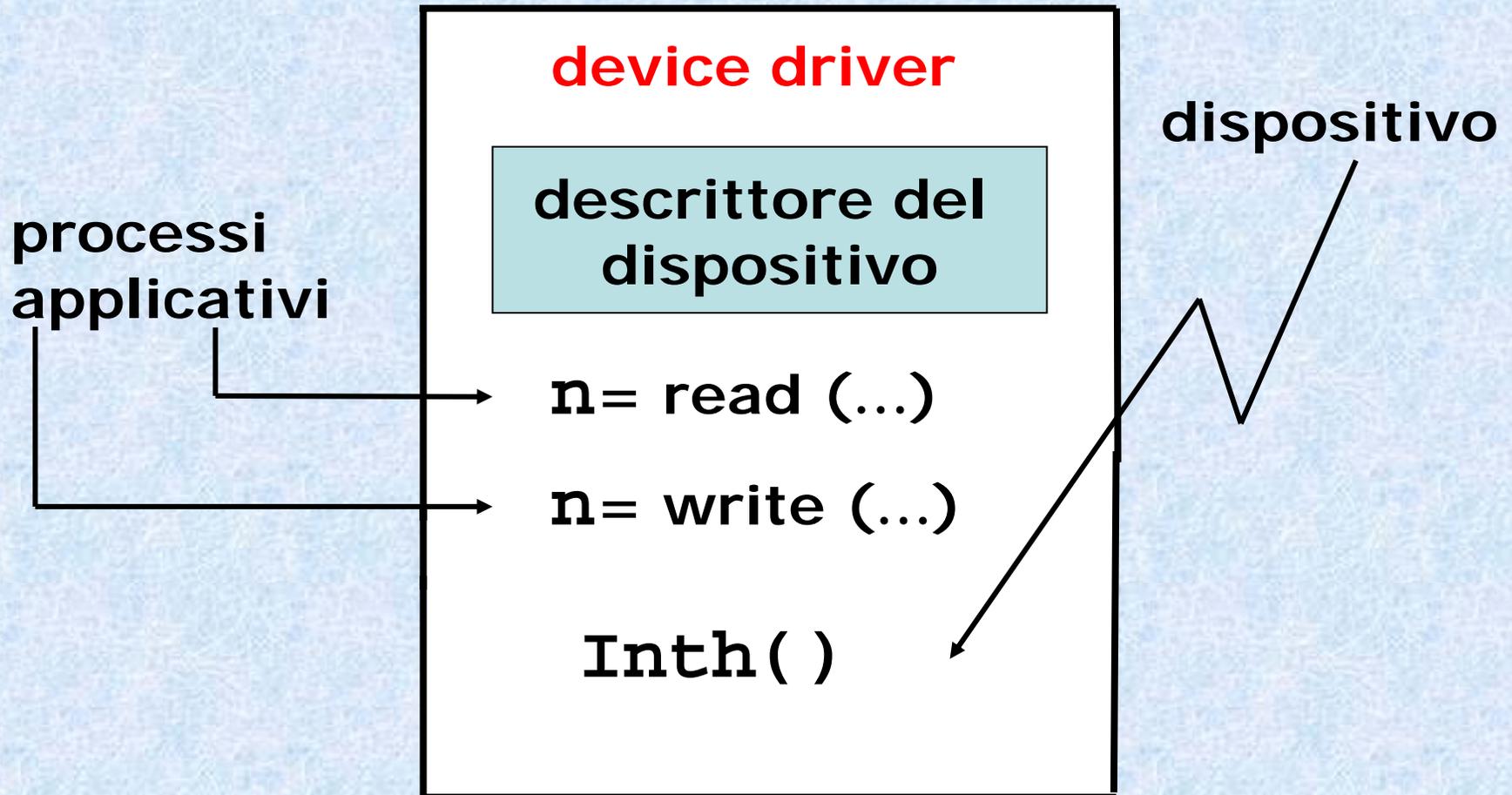
bit di condizioni
di errore

bit di flag



Registro di stato

Livello dipendente dai dispositivi: descrittore del dispositivo



Descrittore di un dispositivo a caratteri

indirizzo registro di controllo
indirizzo registro di stato
indirizzo registro dati
semaforo Dato_disponibile
contatore dati da trasferire
puntatore al buffer in memoria
esito del trasferimento

Gestione controllata da interruzione (1)

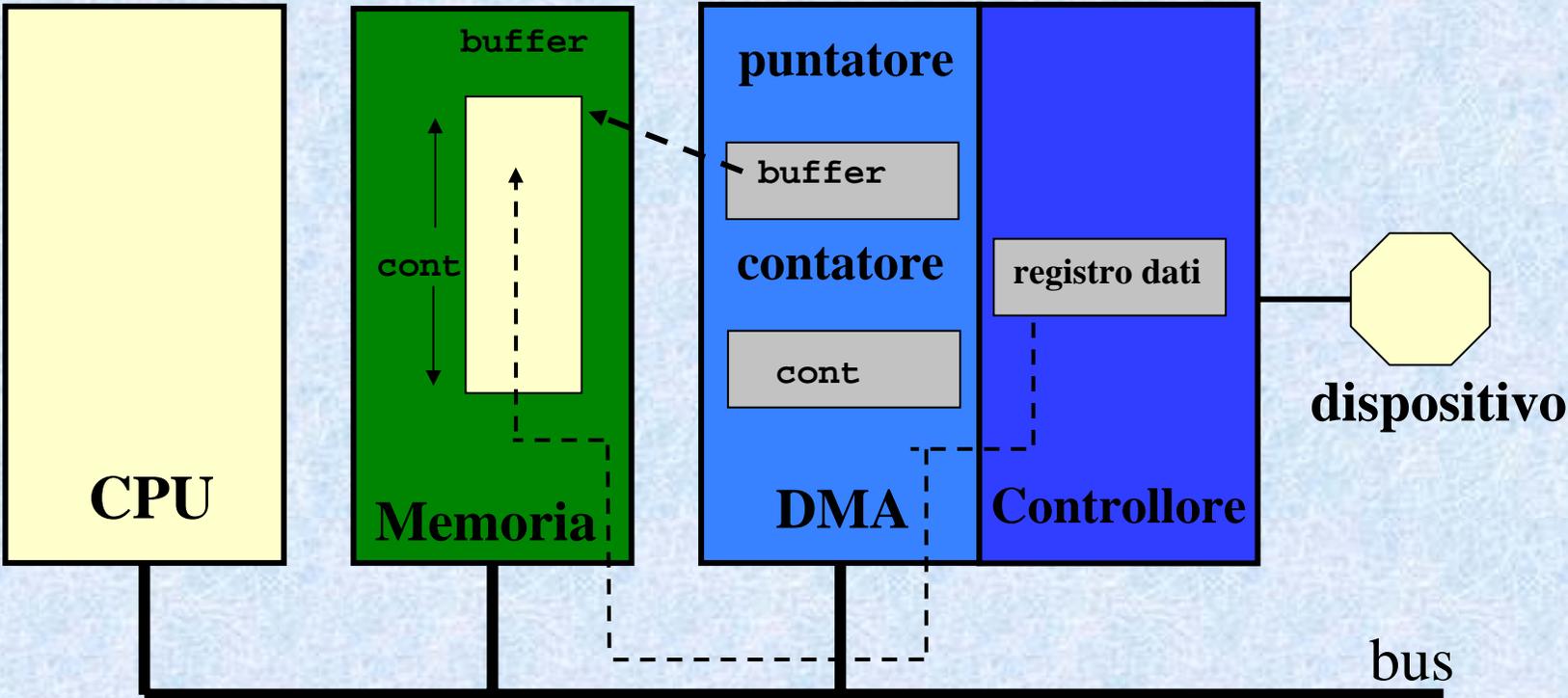
```
int read (int disp, char * pbuf, int cont)
{
    descrittore[disp].contatore = cont;
    descrittore[disp].puntatore = pbuf;
    <attivazione del dispositivo>; // bit di start=1
    // sospensione del processo
    descrittore[disp].dato_disponibile.wait();
    // in caso di errore restituisce -1
    if (descrittore[disp].esito == <codice di errore>)
        >return(-1);
    // altrimenti restituisce il numero di dati letti
    return(cont - descrittore[disp].contatore);
}
```

Gestione controllata da interruzione (2)

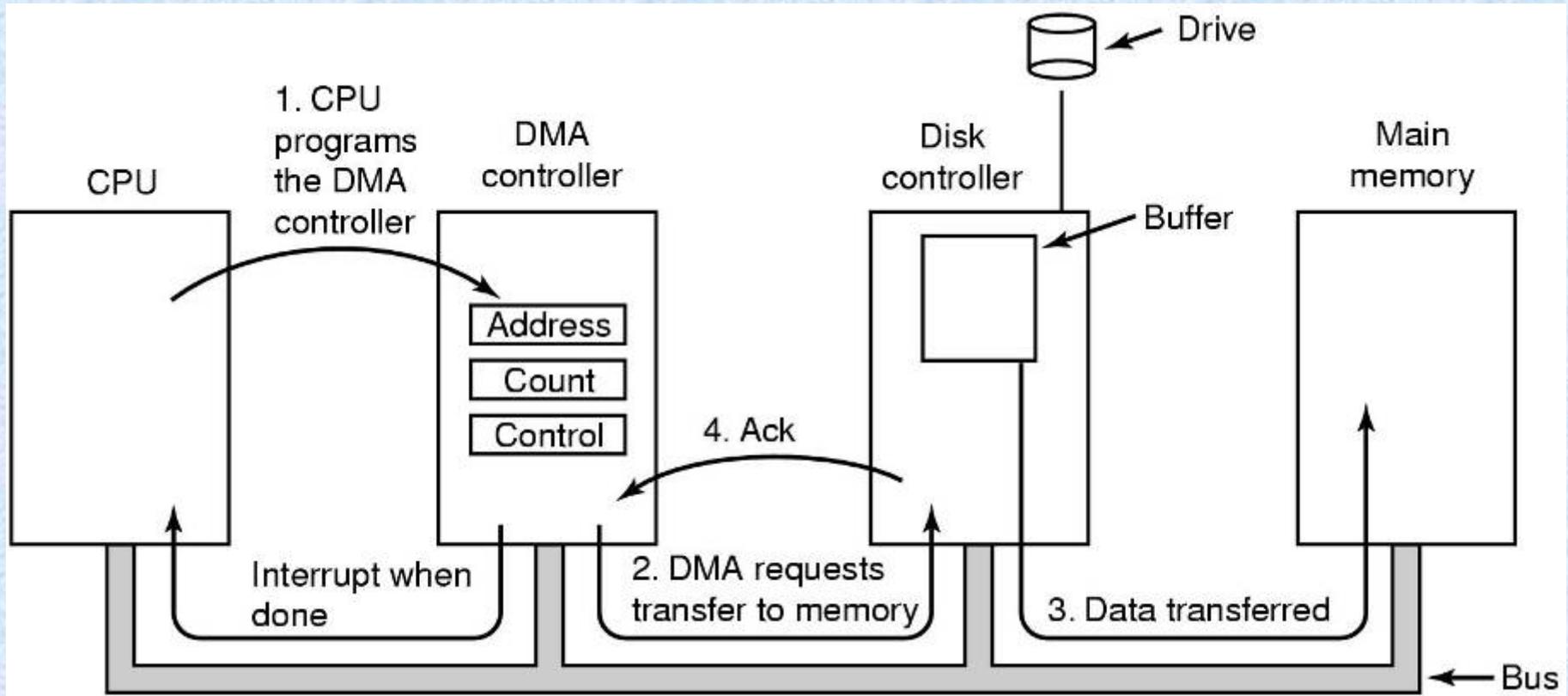
```
void inth() { // funzione di risposta alle interruzioni
    char b;
    <legge il registro di stato del controllore>;
    if (<bit di errore == 0>) { // assenza di errori

        <lettura del registro dati>
        <assegnamento alla variabile locale b>;
        *descrittore[disp].puntatore = b;
        descrittore[disp].puntatore++;
        descrittore[disp].contatore--;
        if(descrittore[disp].contatore != 0)
            <riattivazione dispositivo>;
        else {
            >descrittore[disp].esito=<terminazione corretta>;
            <disattivazione del dispositivo>;
            // riattivazione processo
            descrittore[disp].dato.disponibile.signal();
        }
    }
    else { // presenza di errori
        <routine di gestione errore>;
        if (<errore non recuperabile>)
            descrittore[disp].esito=<codice errore>;
        // riattivazione processo
        descrittore[disp].dato.disponibile.signal();
    }
}
return; // ritorno da interruzione
```

Gestione con accesso diretto alla memoria (DMA)



Accesso diretto alla memoria (DMA)



Funzioni del livello indipendente dai dispositivi

- **Naming (comprende controllo degli accessi)**
- **Buffering**
- **Gestione dei malfunzionamenti.**
- **Allocazione dei dispositivi ai processi applicativi.**
- **Spooling**
- **Scheduling**

Funzioni del livello indipendente dai dispositivi

Naming

Associare ai nomi dei dispositivi i rispettivi gestori (*device drivers*), attivati dai livelli superiori attraverso l'interfaccia “*device-independent*”.

N= read (disp, buffer, nbytes)

nome unico
del dispositivo



```
graph BT; A[nome unico del dispositivo] --> B[disp];
```

buffer di sistema



```
graph BT; C[buffer di sistema] --> D[buffer];
```

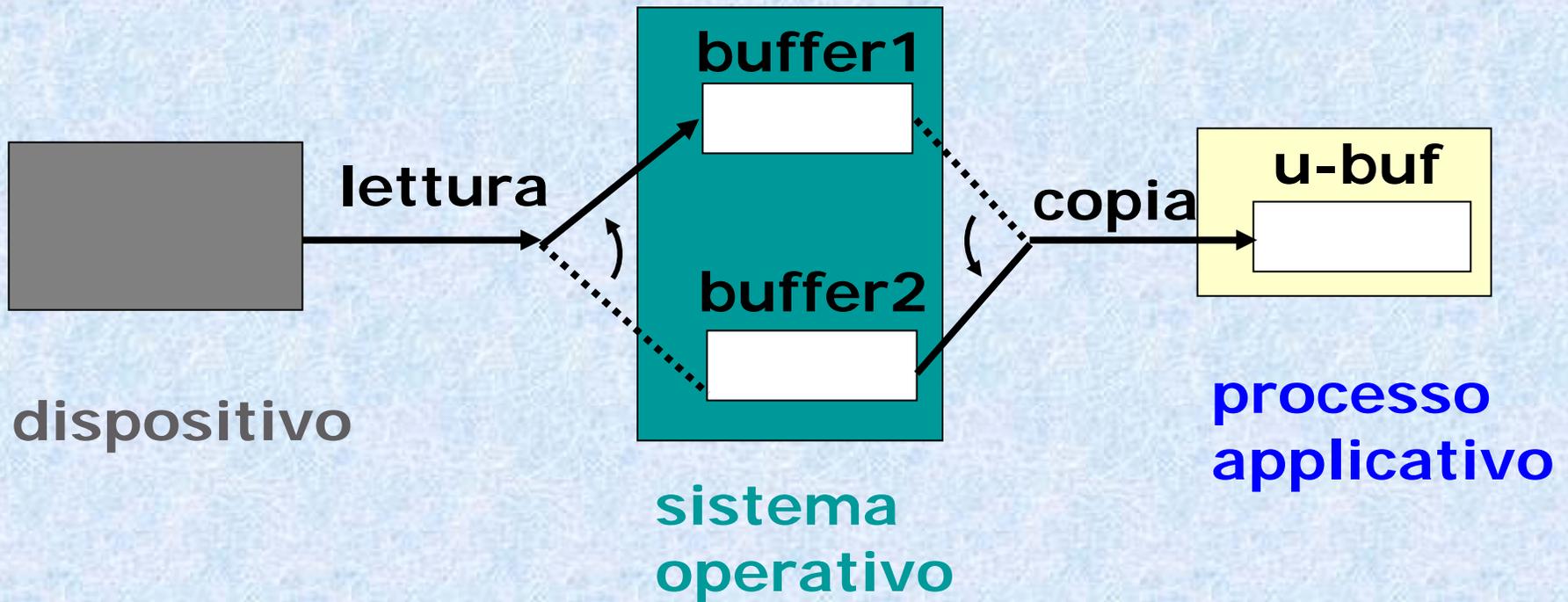
Funzioni del livello indipendente dai dispositivi

Buffering (1)

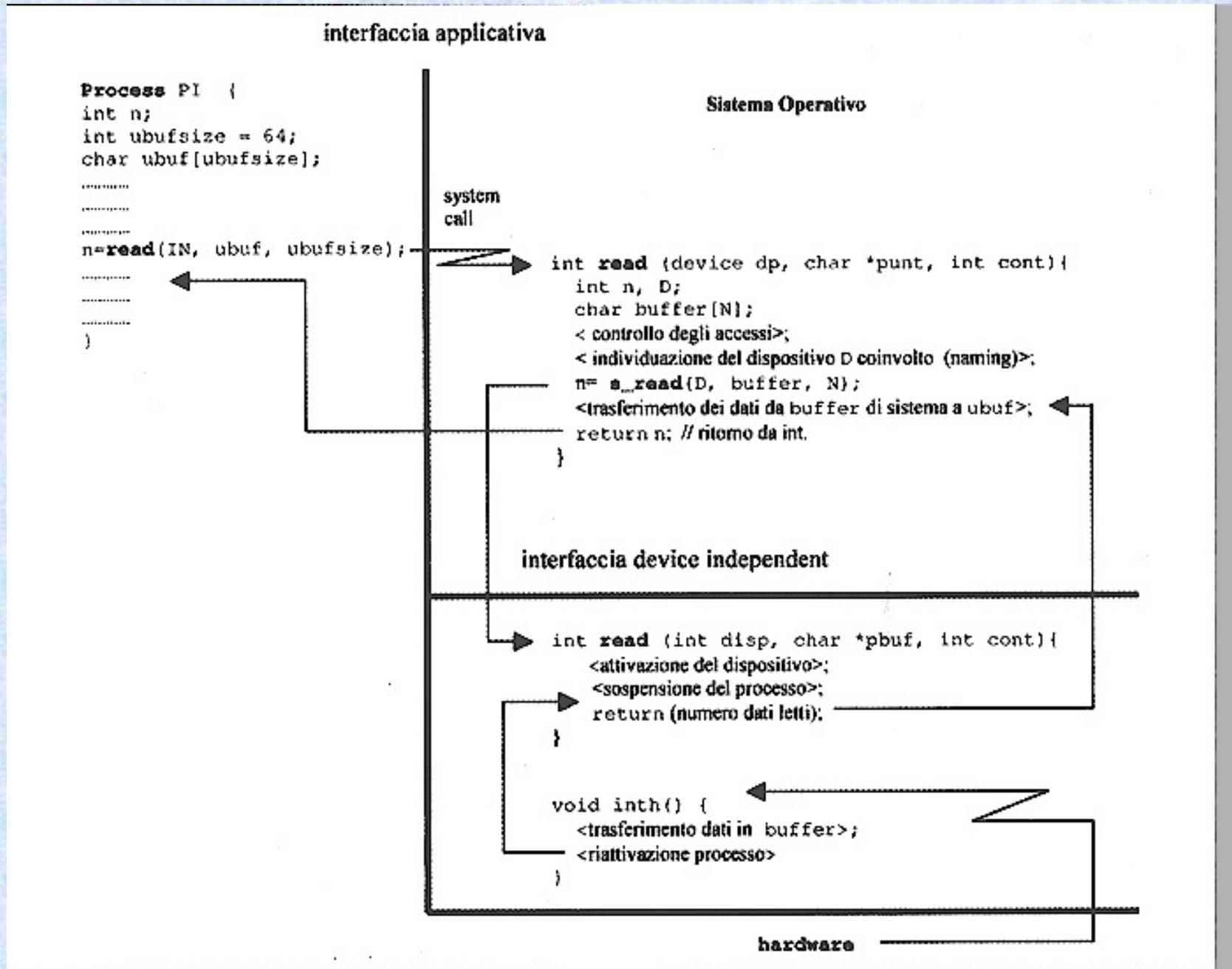


Funzioni del livello indipendente dai dispositivi

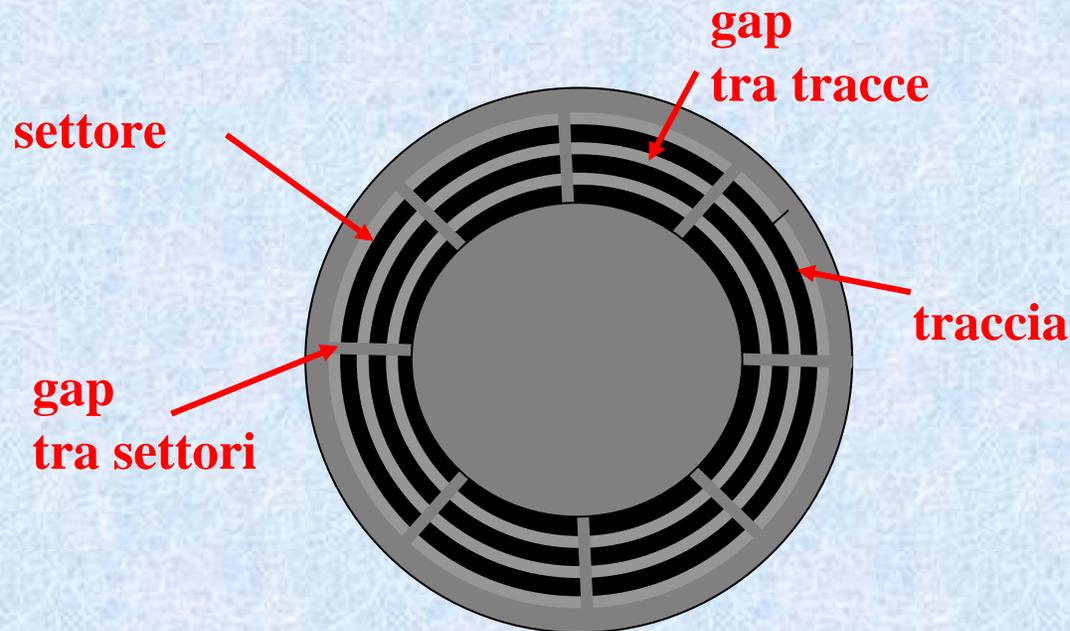
Buffering (2)



Flusso di controllo per l'esecuzione di un comando di I/O



5.4 Gestione e organizzazione dei dischi

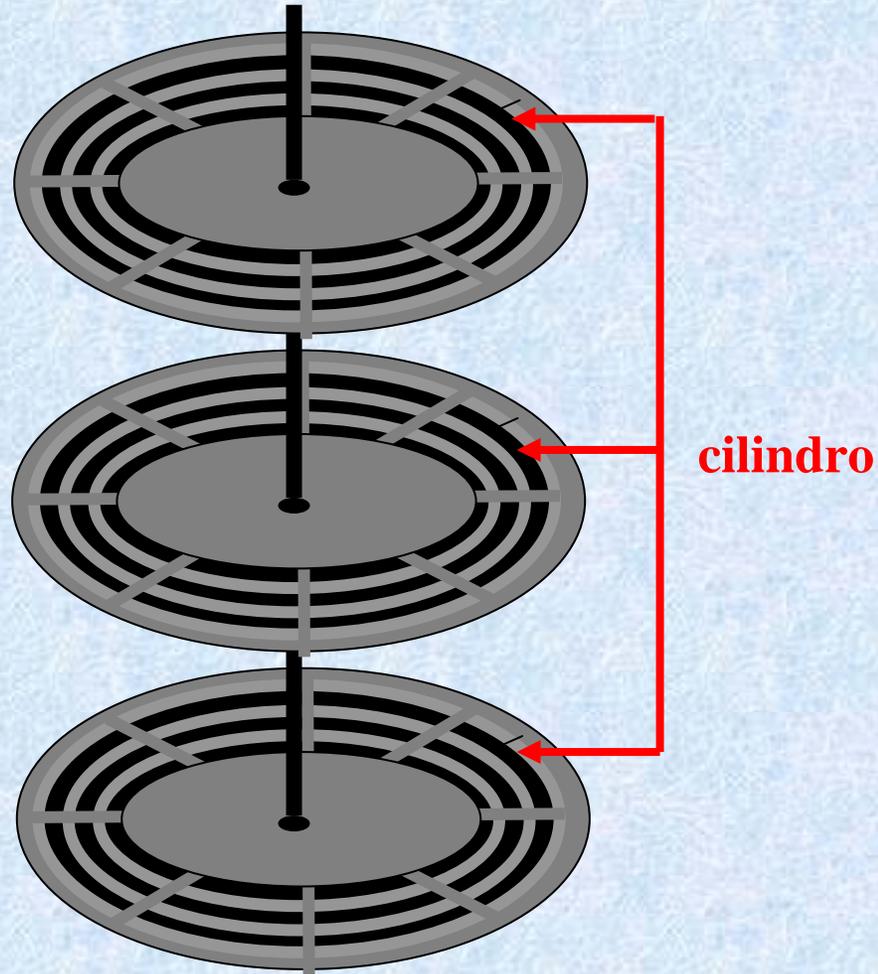


a) disco singolo

5.4 Gestione e organizzazione dei dischi

b) disk pack

Indirizzo:
(cilindro, faccia, settore)



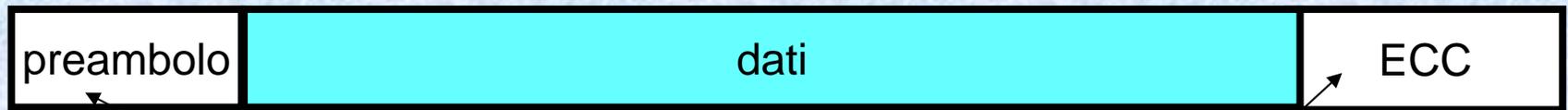
Struttura fisica di un disk pack



Formattazione del disco

- Formattazione di basso livello: predispone i settori e le tracce nel disco

Struttura di un settore



Permette alla testina di riconoscere l'inizio di un settore e fornisce il numero del settore

Codice correttore di errore :
usa la ridondanza per rilevare e/o correggere errori

Formattazione del disco

- Formattazione di basso livello
 - Definisce tracce e settori
 - Inserisce Master Boot Record (settore 0) e codice di boot
 - Definisce *partizioni* e tabella delle partizioni
 - Se esiste *partizione di boot*, marcata attiva
- Formattazione di alto livello
 - inserisce un file system vuoto nella partizione
 - definisce *boot block* (primo blocco della partizione)
- Al *bootstrap*
 - BIOS carica ed esegue MBR
 - carica *boot block* dalla partizione attiva
 - carica SO memorizzato nella partizione e lo lancia

Parametri tipici dei dischi

Parametri	AC2540	WDE18300
Numero cilindri (N. di tracce per ogni faccia)	1048	13614
Tracce per cilindro	4	8
Settori per traccia	252	320
Byte per settore	512	512
Capacità	540 MB	18.3 GB
Tempo minimo di seek (tra cilindri adiacenti)	4 msec.	0.6 msec.
Tempo medio di seek	11 msec.	5.2 msec.
Tempo di rotazione	13 msec.	6 msec.
Tempo di trasferimento di un settore	53 msec.	19 msec.

Struttura fisica e struttura logica del disco

Struttura fisica del disco

cilindri, facce, settori

Parametri: n_{Cilindri} , n_{Facce} , n_{Settori}

Indirizzi fisici: terne (c, f, s)

Struttura logica del disco

vettore di blocchi

Indirizzi logici: indici di blocco

Indice di blocco: intero nell'intervallo $[0, n_{\text{cilindri}} * n_{\text{facce}} * n_{\text{settori}})$

Indirizzo logico b del blocco di indirizzo fisico (c, f, s)

$$b = c * (n_{\text{Facce}} * n_{\text{Sett}}) + f * n_{\text{Sett}} + s$$

Indirizzo fisico (c, f, s) del blocco di indirizzo logico b

$$c = b \text{ div } (n_{\text{Facce}} * n_{\text{Sett}})$$

$$f = (b \text{ mod } (n_{\text{Facce}} * n_{\text{Sett}})) \text{ div } n_{\text{Sett}}$$

$$s = (b \text{ mod } (n_{\text{Facce}} * n_{\text{Sett}})) \text{ mod } n_{\text{Sett}}$$

Esecuzione dei comandi al disco

Il tempo necessario per leggere o scrivere un blocco è determinato da tre fattori

1. Tempo di *seek*
 2. Ritardo rotazionale (Rotational delay)
 3. Tempo di trasferimento dei dati
- Il tempo di seek è il ritardo dominante
 - Il controllo degli errori viene svolto dal controller

Algoritmi di scheduling per il disco (1)

Obiettivo: minimizzare il tempo medio di esecuzione dei *comandi pendenti*

Problema: l'insieme dei comandi pendenti è dinamico: mentre i comandi vengono eseguiti, nuovi comandi entrano nell'insieme

Politiche più comuni:

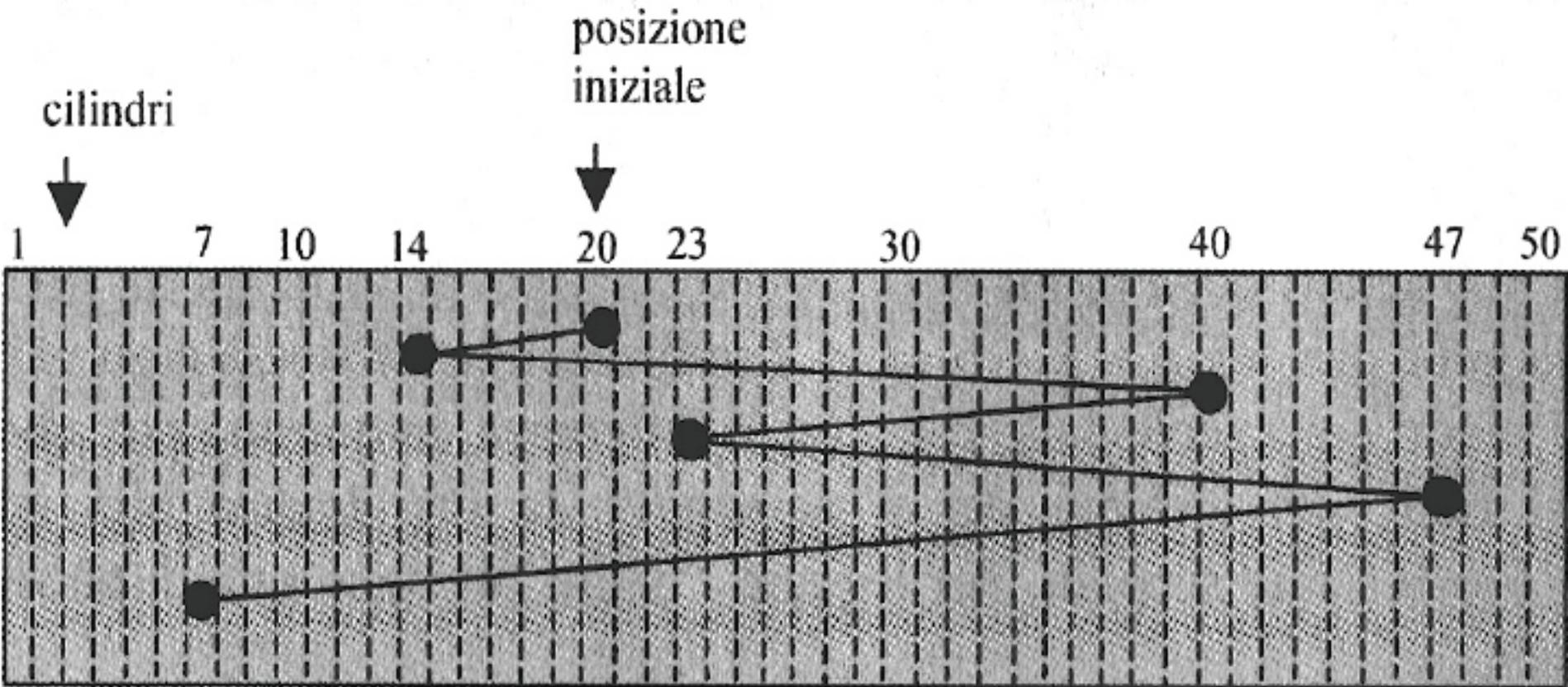
- *First Come First Served* ovvero *First In First Out (FCFS, ovvero FIFO)*
- *Shortest Seek Time First (SSTF, ovvero SSF)*
- *Scanning (SCAN)*

Obiettivo di SSTF e SCAN: ridurre il valore medio del tempo di seek

Algoritmi di scheduling per il disco

1) Scheduling con politica *FCFS*

- Sono arrivati nell'ordine comandi relativi ai cilindri 20, 14, 40, 23, 47, 7
- Tutti pendenti al tempo iniziale, quando le testine di lettura/scrittura sono posizionate sul cilindro 20

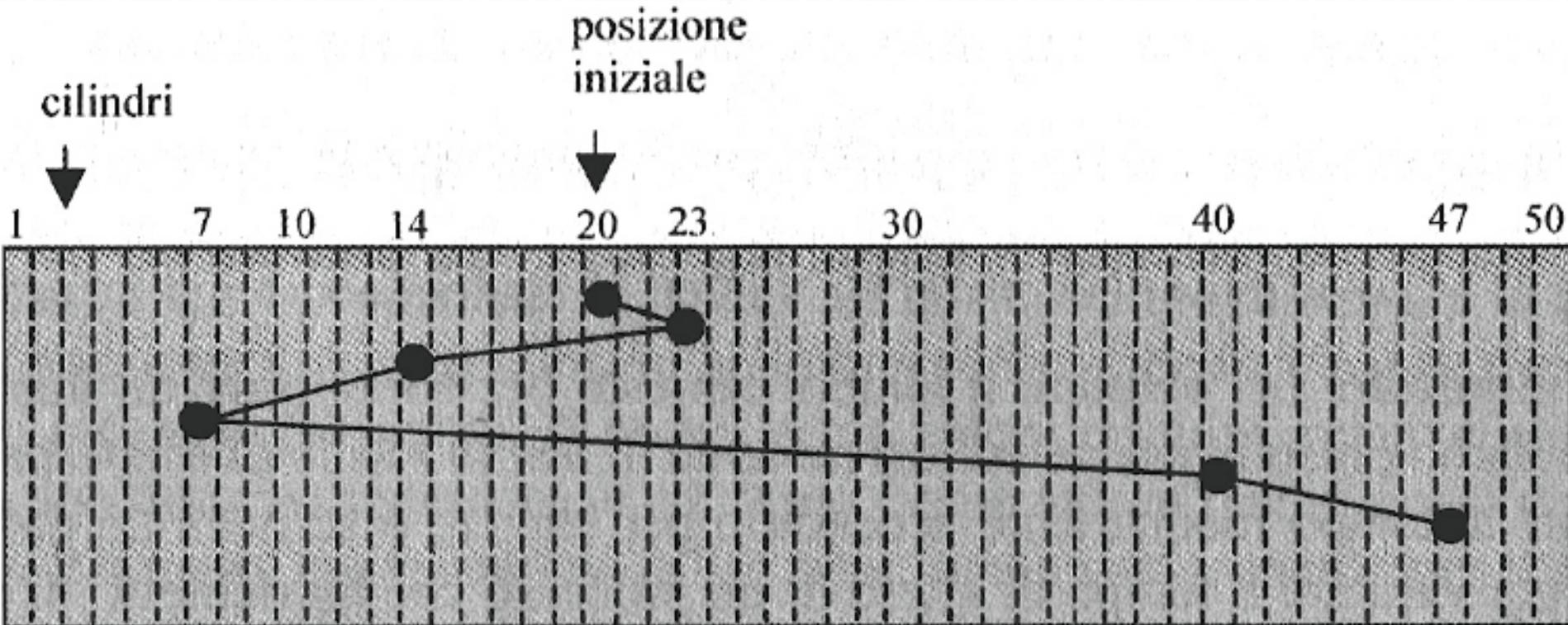


Algoritmi di scheduling per il disco

2) Scheduling con politica SSTF

- Sono arrivati nell'ordine comandi relativi ai cilindri 20, 14, 40, 23, 47, 7
- Tutti pendenti al tempo iniziale, quando le testine di lettura/scrittura sono posizionate sul cilindro 20

==> Possibilità di attesa indefinita

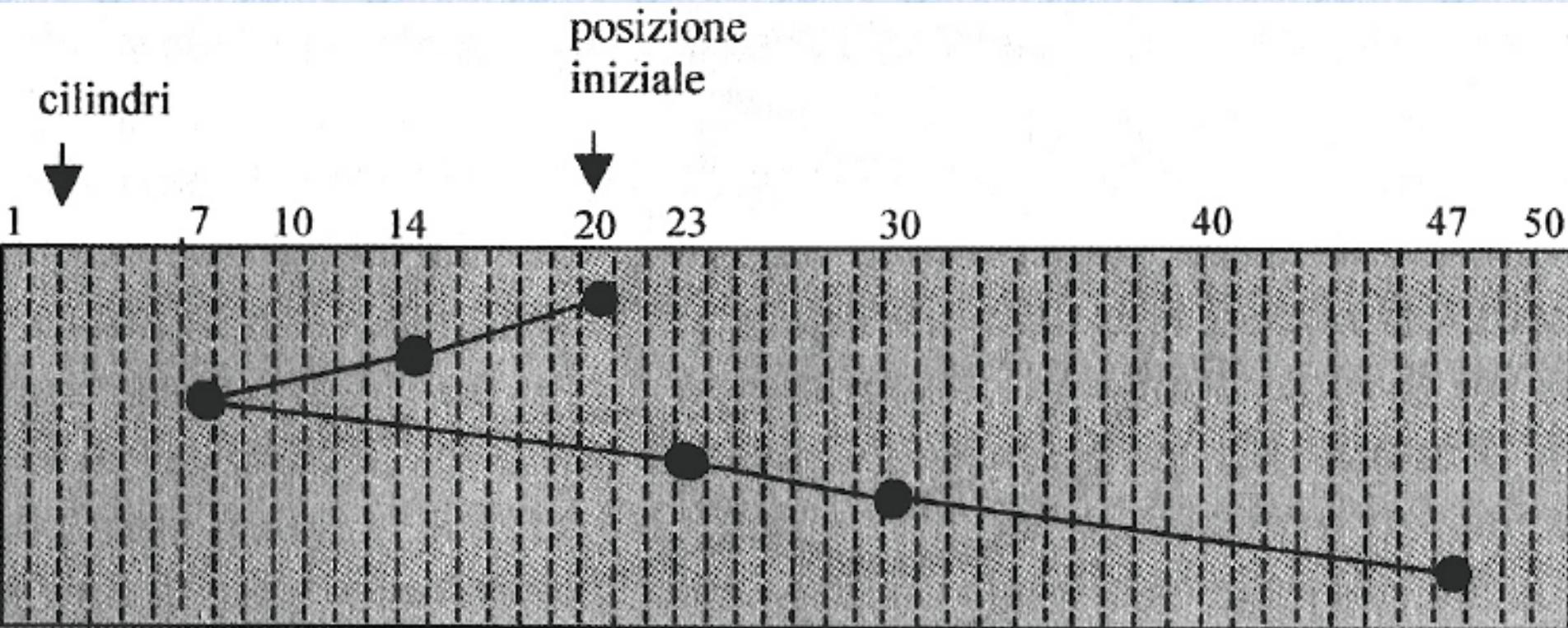


Algoritmi di scheduling per il disco

3) Scheduling con politica SCAN

- Sono arrivati nell'ordine comandi relativi ai cilindri 20, 14, 40, 23, 47, 7
- Tutti pendenti al tempo iniziale, quando le testine di lettura/scrittura sono posizionate sul cilindro 20

==> Evita l'attesa indefinita



Dischi RAID

Redundant Array of Independent Disks

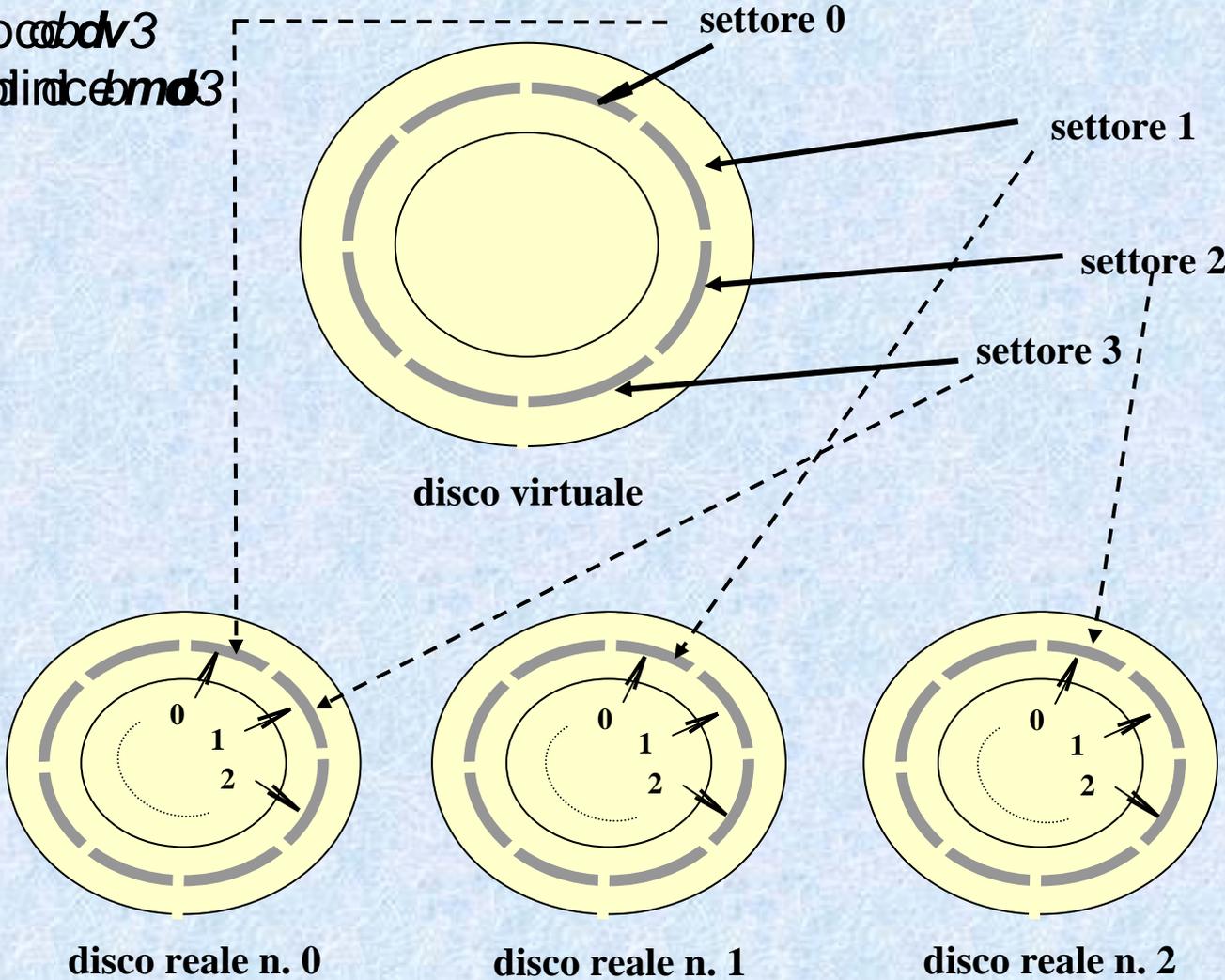
Architettura RAID:

- Realizza un *disco virtuale* di capacità superiore a quella dei singoli dischi
 - l'interfaccia è quella di un unico disco
- sfrutta il parallelismo per ottenere un accesso più veloce
 - i blocchi consecutivi di uno stesso file sono distribuiti sui dischi dell'array in modo da permettere operazioni contemporanee
- sfrutta la ridondanza per accrescere l'affidabilità
 - la ridondanza permette di correggere gli errori di certe classi

Diversi livelli di architettura RAID, con diversi livelli di ridondanza

Dischi RAID

È il disco
virtuale
che il controller
RAID
vede come un
disco fisico



Dischi RAID

Livello 0: Dischi asincroni, nessuna ridondanza

- Si possono effettuare contemporaneamente operazioni indipendenti

Livello 1: Dischi asincroni, ogni disco ha una copia ridondante (*mirror*)

- Si possono effettuare contemporaneamente operazioni indipendenti e correggere errori

Livello 2: Dischi sincroni, i dischi ridondanti contengono codici per la correzione degli errori

- si possono effettuare contemporaneamente operazioni indipendenti e correggere errori

Livello 3: Dischi sincroni, un solo disco ridondante

- contiene la parità del contenuto degli altri dischi
- si possono effettuare contemporaneamente operazioni indipendenti e correggere errori

Livello 4: Dischi asincroni; un disco ridondante

- contiene la parità del contenuto degli altri dischi
- si possono effettuare contemporaneamente operazioni indipendenti e correggere errori
- Il disco ridondante è sovraccarico nei piccoli aggiornamenti

Livello 5: Come livello precedente, ma parità distribuita tra tutti i dischi

- permette un miglior bilanciamento del carico tra i dischi

Dischi RAID

Dischi asincroni:

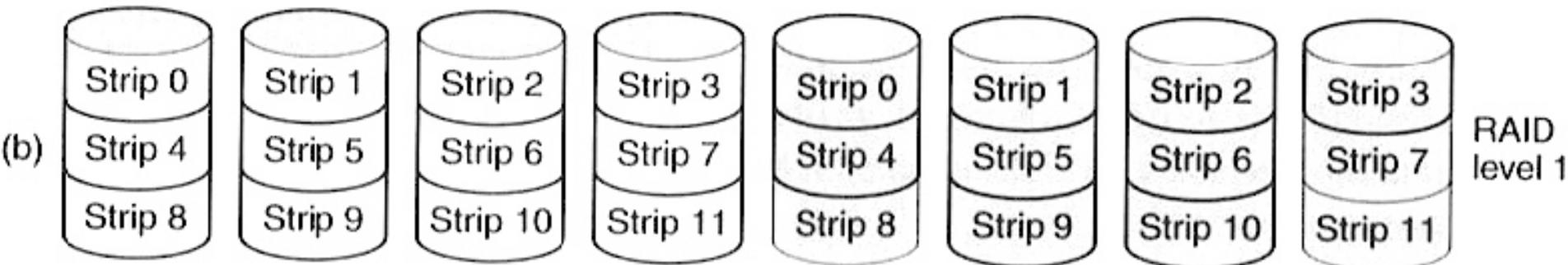
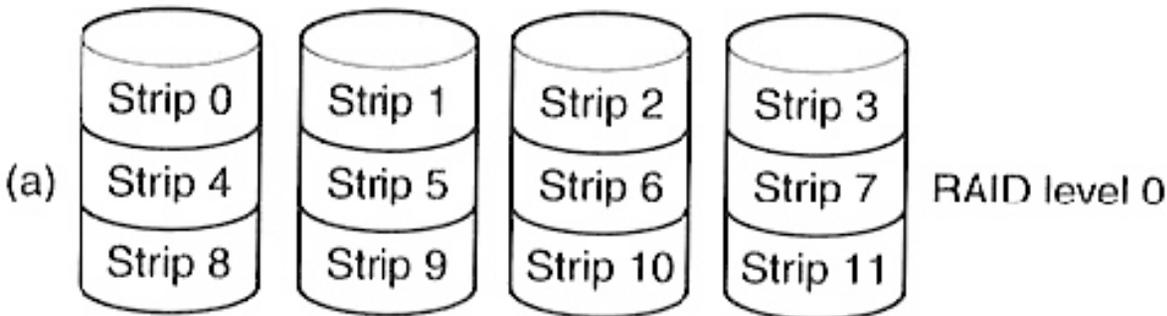
- vengono distribuite le *strips* (singoli settori o sequenze di settori contigui)

Livello 0: Nessuna ridondanza

- possibilità di eseguire contemporaneamente operazioni indipendenti

Livello 1: Ogni strip ha una copia ridondante (mirror)

- possibilità di eseguire contemporaneamente operazioni indipendenti e di correggere errori
- Esempio: dischi 4, 5, 6, 7 *mirrors* dei dischi 0, 1, 2, 3, 4



Dischi RAID

Dischi sincroni:

- vengono distribuiti i bit

Livello 2: Ridondanza con codice correttore di errori

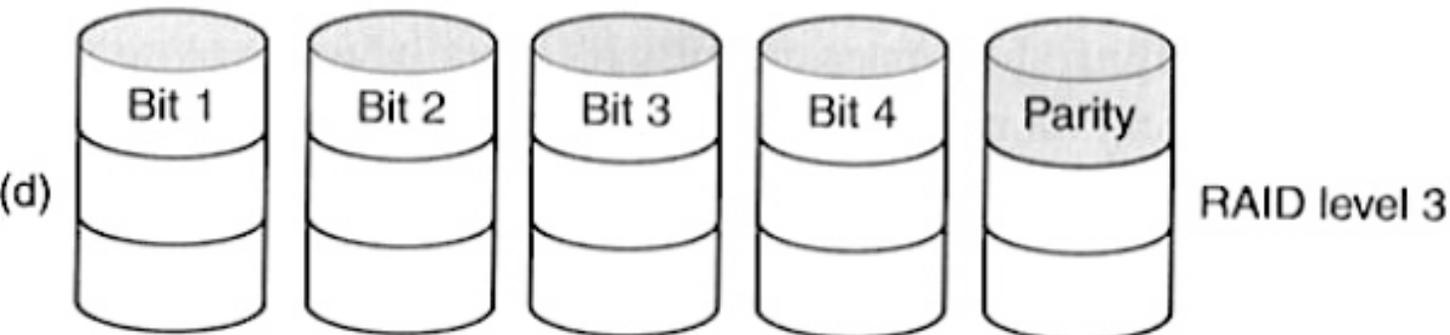
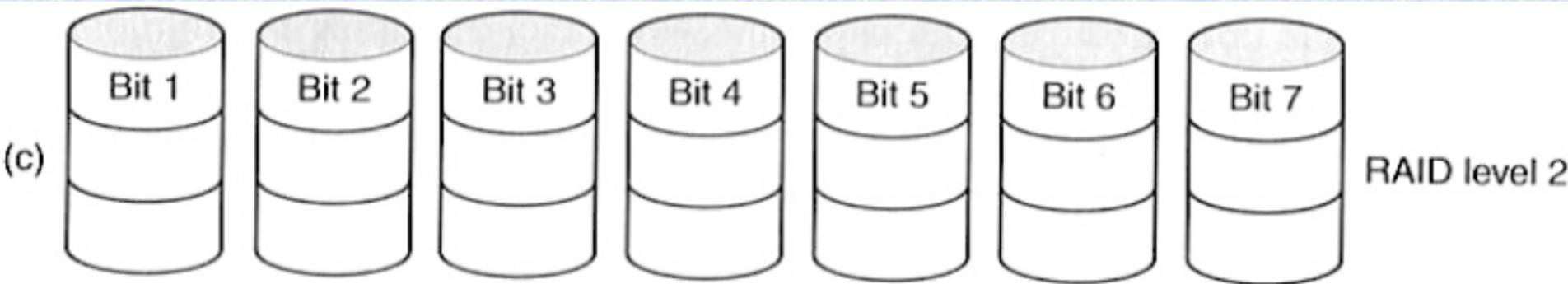
Esempio bit 1, 2, 3 e 4 di informazione, bit 5, 6 e 7 ridondanti

- Lettura o scrittura contemporanea di tutti i bit della strip

Livello 3: Ridondanza con bit di parità

Esempio: bit 5 parità dei bit 1, 2, 3, 4

- Lettura o scrittura contemporanea di tutti i bit della strip e correzione di *crash faults*



Dischi RAID

Dischi asincroni, vengono distribuite le *strips*

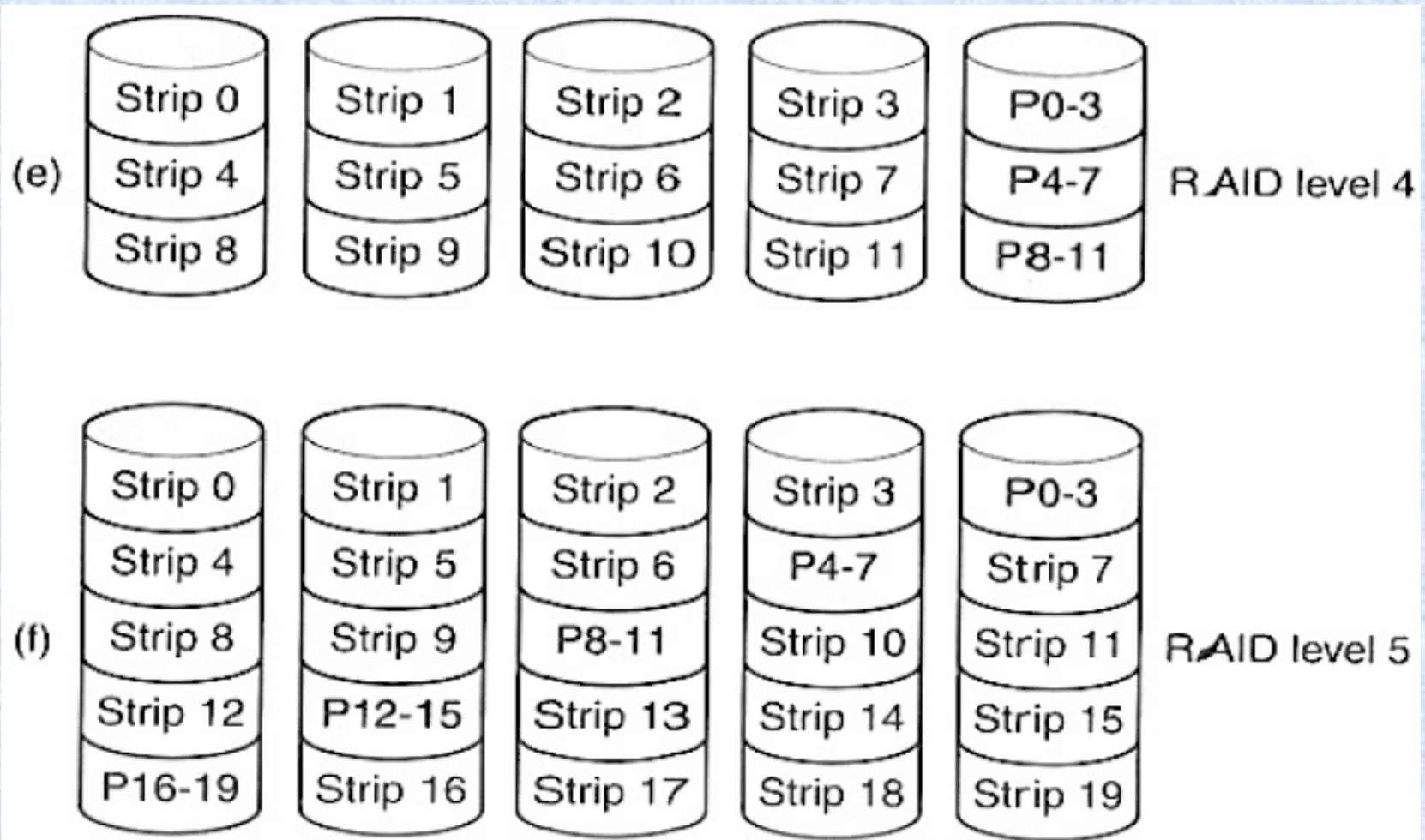
Livello 4: Ridondanza con *strip* di parità

Esempio: disco 4 contiene le strip di parità delle strips omologhe dei dischi 0, 1, 2, 3

- possibilità di operazioni indipendenti contemporanee e di correzione di *crash faults singoli*

Livello 5: Come livello 4, ma strip di parità distribuite nei vari dischi

- Permette un miglior bilanciamento del carico tra i dischi



Dischi RAID di livello 4: esempio (1)

Un disco RAID di livello 4 è composto da 5 dischi fisici, numerati da 0 a 4. I blocchi del disco virtuale V sono mappati nei dischi 0, 1, 2, 3: precisamente il blocco b del disco V è mappato nel blocco $b \text{ div } 4$ del disco fisico di indice $b \text{ mod } 4$. Il disco 4 è ridondante e il suo blocco di indice i contiene la parità dei blocchi di indice i dei dischi 0, 1, 2, 3.

Il gestore del disco virtuale accetta comandi (di lettura o scrittura) che interessano più blocchi consecutivi : ad esempio $read(buffer, PrimoBlocco, NumeroBlocchi)$ legge un numero di blocchi pari a $NumeroBlocchi$ a partire da quello di indice logico $PrimoBlocco$ e li scrive nel buffer di indirizzo iniziale $buffer$.

Ad esempio, l'operazione $read(buffer\ 12, 3)$ legge i blocchi 12, 13, 14 del disco virtuale, mappati nel blocco 3 dei dischi fisici 0, 1, 2

✓ trattandosi di un'operazione che interessa dischi fisici indipendenti, può essere eseguita in un solo tempo di accesso.

Dischi RAID di livello 4: esempio (2)

Supponiamo che i blocchi di indice 3 dei dischi fisici 0, 1, 2 e 3 abbiano i contenuti mostrati in tabella: di conseguenza il blocco di indice 3 del disco fisico 4 contiene la parità del contenuto dei blocchi omologhi dei dischi fisici 0, 1, 2 e 3.

Disco 0	0	1	0	0	1	1	0	1
Disco 1	1	0	1	1	0	0	0	1
Disco 2	0	1	1	0	1	0	0	1
Disco 3	0	1	1	1	1	0	0	1
Disco 4	1	1	1	0	1	1	0	0

Se la lettura dal disco fisico 1 fallisce a causa di un *crash fault*, l'evento viene rilevato dal controllore, che restituisce un blocco vuoto. Il contenuto del blocco 3 del disco fisico 1 può essere ricostruito come parità dei contenuti dei blocchi omologhi dei dischi 0, 2, 3 e 4.

CONTENUTO RESTITUITO

Disco 1	-	-	-	-	-	-	-	-
---------	---	---	---	---	---	---	---	---

CONTENUTO RICOSTRUITO

Disco 1	1	0	1	1	0	0	0	1
---------	---	---	---	---	---	---	---	---

Se si esegue l'operazione *write(buffer 13, 1)*, che scrive il contenuto del *buffer* nel blocco di indice 3 del disco fisico 1 e il buffer contiene 11010111, è necessario modificare come mostrato in tabella anche la parità contenuta nel blocco omologo del disco Fisico 4. La parità può essere ricalcolata in base alla differenza tra il vecchio e il nuovo contenuto del blocco 3 del disco 1, senza la necessità di leggere i blocchi omologhi dei dischi 0, 2 e 3.

PRIMA DELL'OPERAZIONE

Disco 0	0	1	0	0	1	1	0	1
Disco 1	1	0	1	1	0	0	0	1
Disco 2	0	1	1	0	1	0	0	1
Disco 3	0	1	1	1	1	0	0	1
Disco 4	1	1	1	0	1	1	0	0

DOPO L'OPERAZIONE

Disco 0	0	1	0	0	1	1	0	1
Disco 1	1	1	0	1	0	1	1	1
Disco 2	0	1	1	0	1	0	0	1
Disco 3	0	1	1	1	1	0	0	1
Disco 4	1	0	0	0	1	0	1	0