GESTIONE DELLE PERIFERICHE D'INGRESSO/USCITA

ARGOMENTI

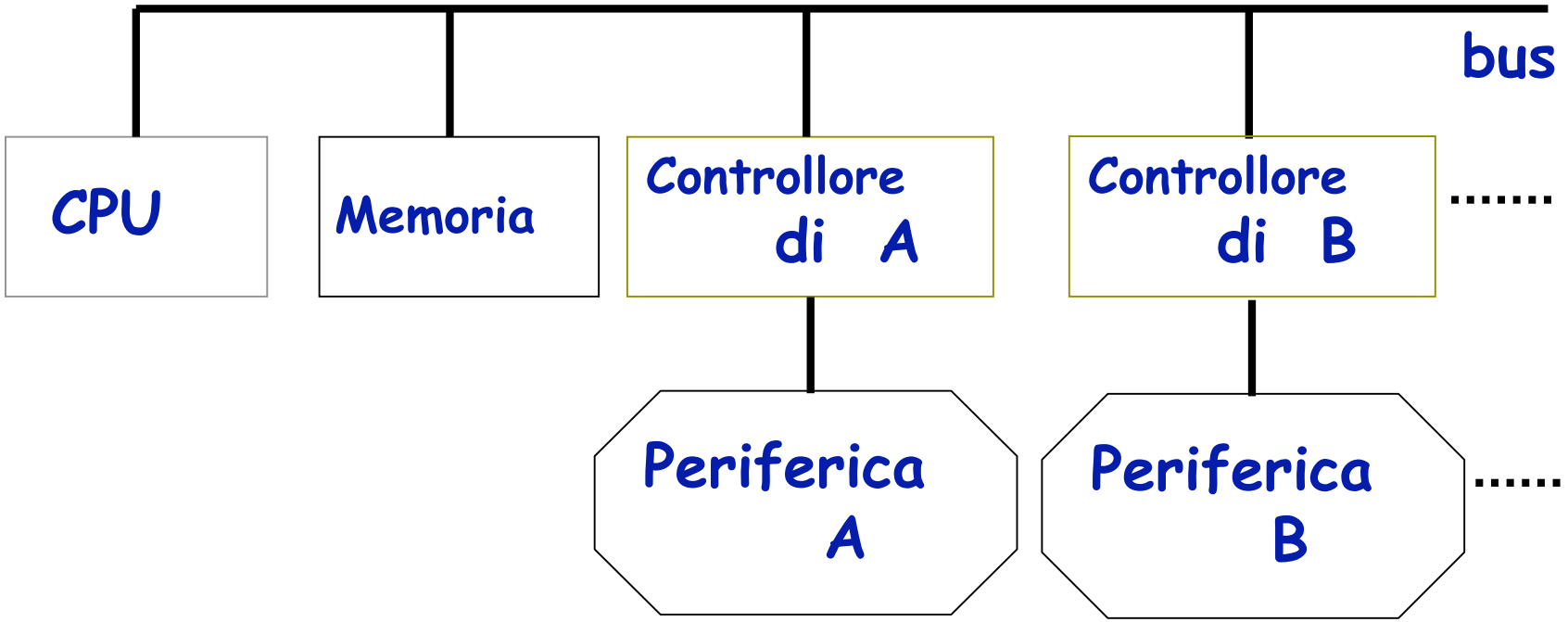
- Compiti del sottosistema di I/O
- Architettura del sottosistema di I/O
- Gestore di un dispositivo di I/O
- Gestione di unità disco

COMPITI DEL SOTTOSISTEMA DI I/O

1. Nascondere al programmatore i dettagli delle interfacce hardware dei dispositivi;
2. Omogeneizzare la gestione di dispositivi diversi;
3. Gestire i malfunzionamenti che si possono verificare durante un trasferimento di dati;
4. Definire lo spazio dei nomi (*naming*) con cui vengono identificati i dispositivi;
5. Garantire la corretta sincronizzazione tra un processo applicativo che ha attivato un trasferimento dati e l'attività del dispositivo.

COMPITI DEL SOTTOSISTEMA DI I/O

1. Nascondere al programmatore i dettagli delle interfacce hardware dei dispositivi



COMPITI DEL SOTTOSISTEMA DI I/O

2) Omogeneizzare la gestione di dispositivi diversi

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	5 Mbytes/sec
CD-ROM	6 Mbytes/sec
Fast Etherneet	12.5 Mbytes/sec
FireWire (IEEE 1394)	50 Mbytes/sec
monitor XGA	60 Mbytes/sec
Ethernet gigabit	125 Mbytes/sec

COMPITI DEL SOTTOSISTEMA DI I/O

2) Omogeneizzare la gestione di dispositivi diversi

TIPOLOGIE DI DISPOSITIVI

- Dispositivi a carattere (es. tastiera, stampante, mouse,…)
- Dispositivi a blocchi (es. dischi, nastri, ..)
- Dispositivi speciali (es. timer)

COMPITI DEL SOTTOSISTEMA DI I/O

3. Gestire i malfunzionamenti che si possono verificare durante un trasferimento di dati

TIPOLOGIE DI GUASTI

- Eventi eccezionali (es. mancanza di carta sulla stampante, end-of-file);
- Guasti transitori (es. disturbi elettromagnetici durante un trasferimento dati);
- Guasti permanenti (es. rottura di una testina di lettura/scrittura di un disco).

COMPITI DEL SOTTOSISTEMA DI I/O

4. Definire lo spazio dei nomi (*naming*) con cui vengono identificati i dispositivi

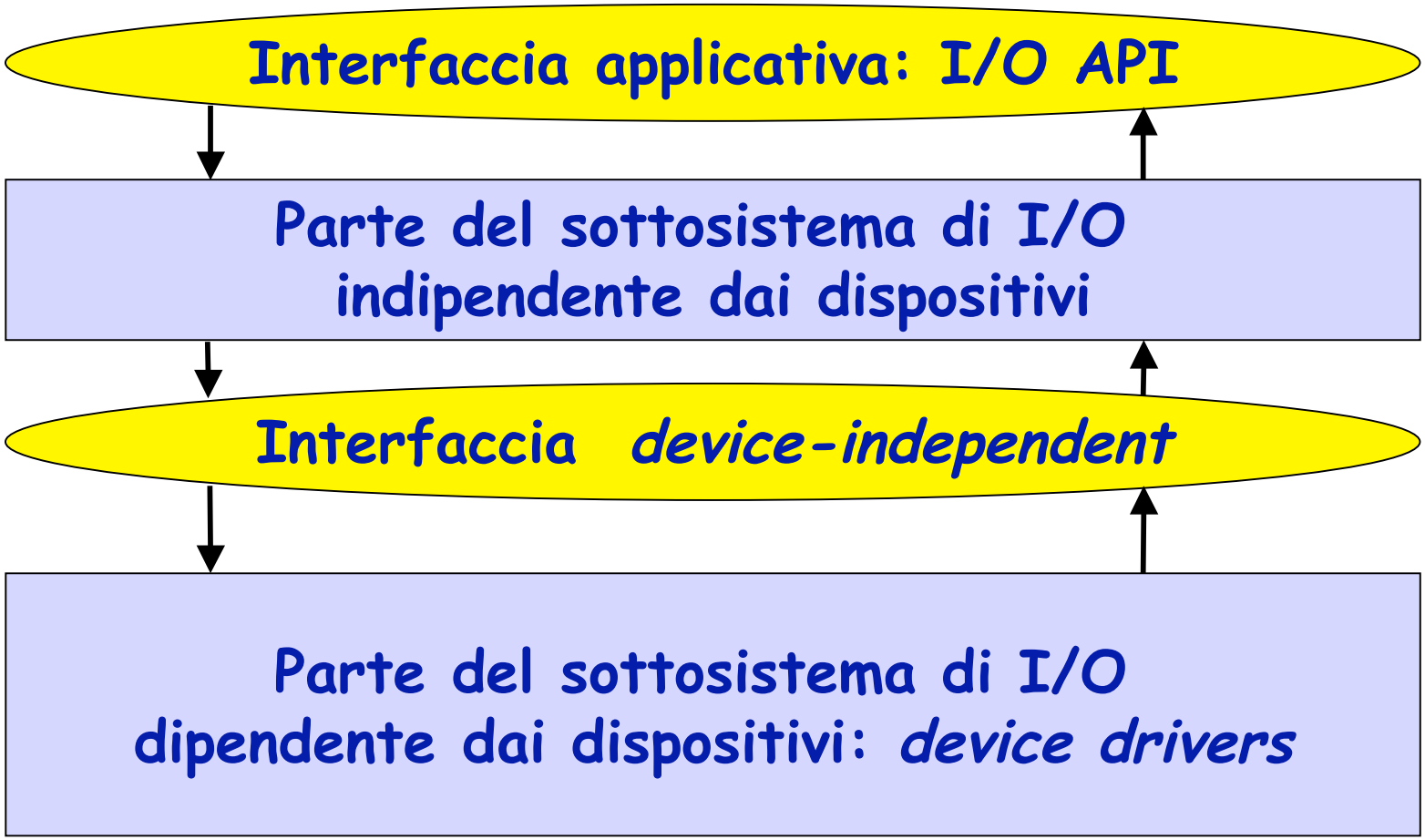
- Uso di nomi unici (valori numerici) all'interno del sistema per identificare in modo univoco i dispositivi;
- Uso di nomi simbolici da parte dell'utente (*I/O API Input/Output Application Programming Interface*);
- Uniformità col meccanismo di *naming* del file-system.

COMPITI DEL SOTTOSISTEMA DI I/O

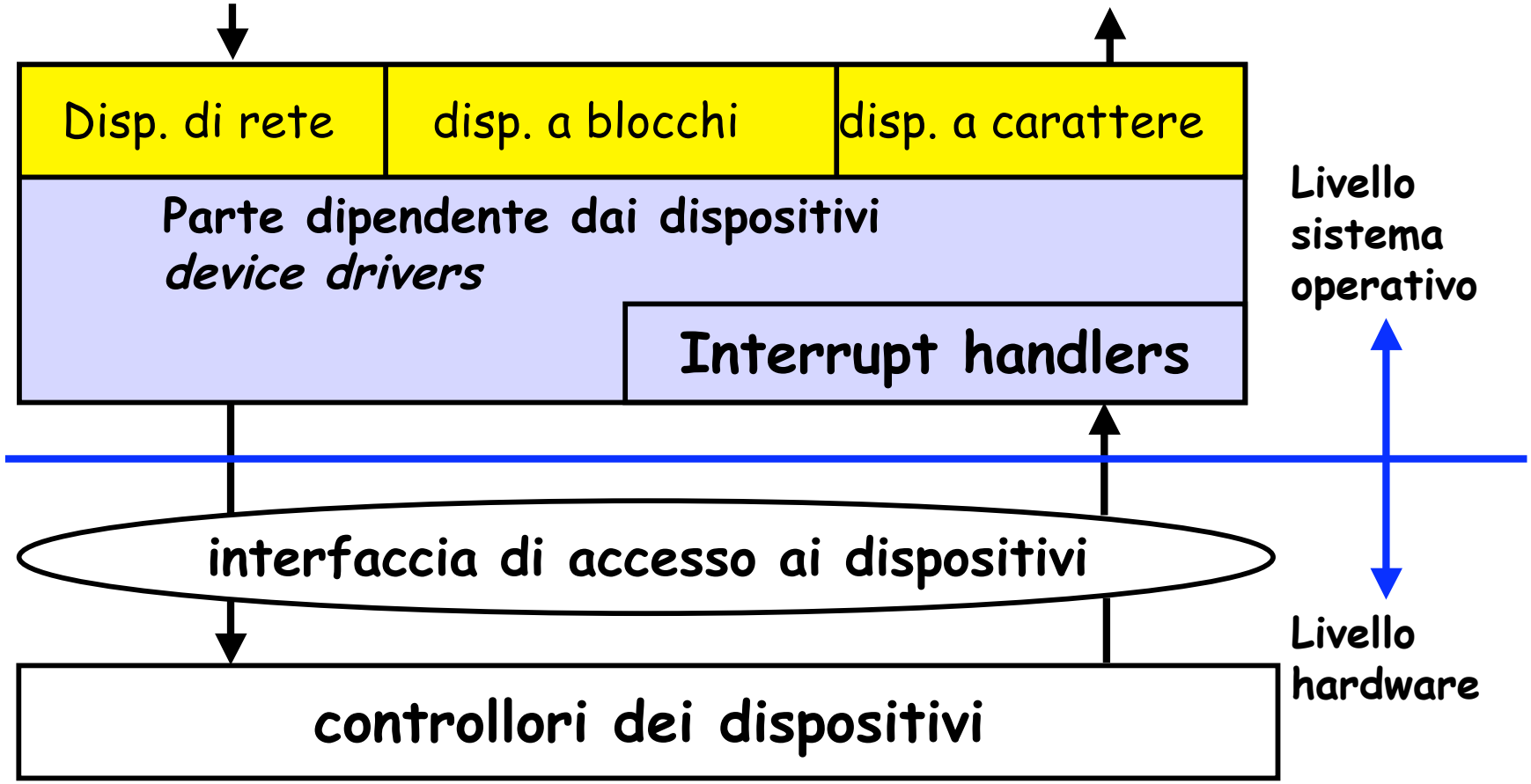
5. Garantire la corretta sincronizzazione tra un processo applicativo che ha attivato un trasferimento dati e l'attività del dispositivo.

- Gestione sincrona dei trasferimenti: un processo applicativo attiva un dispositivo e si blocca fino al termine del trasferimento;
- Gestione asincrona dei trasferimenti: un processo applicativo attiva un dispositivo e prosegue senza bloccarsi;
- Necessità di gestire la "*bufferizzazione*" dei dati.

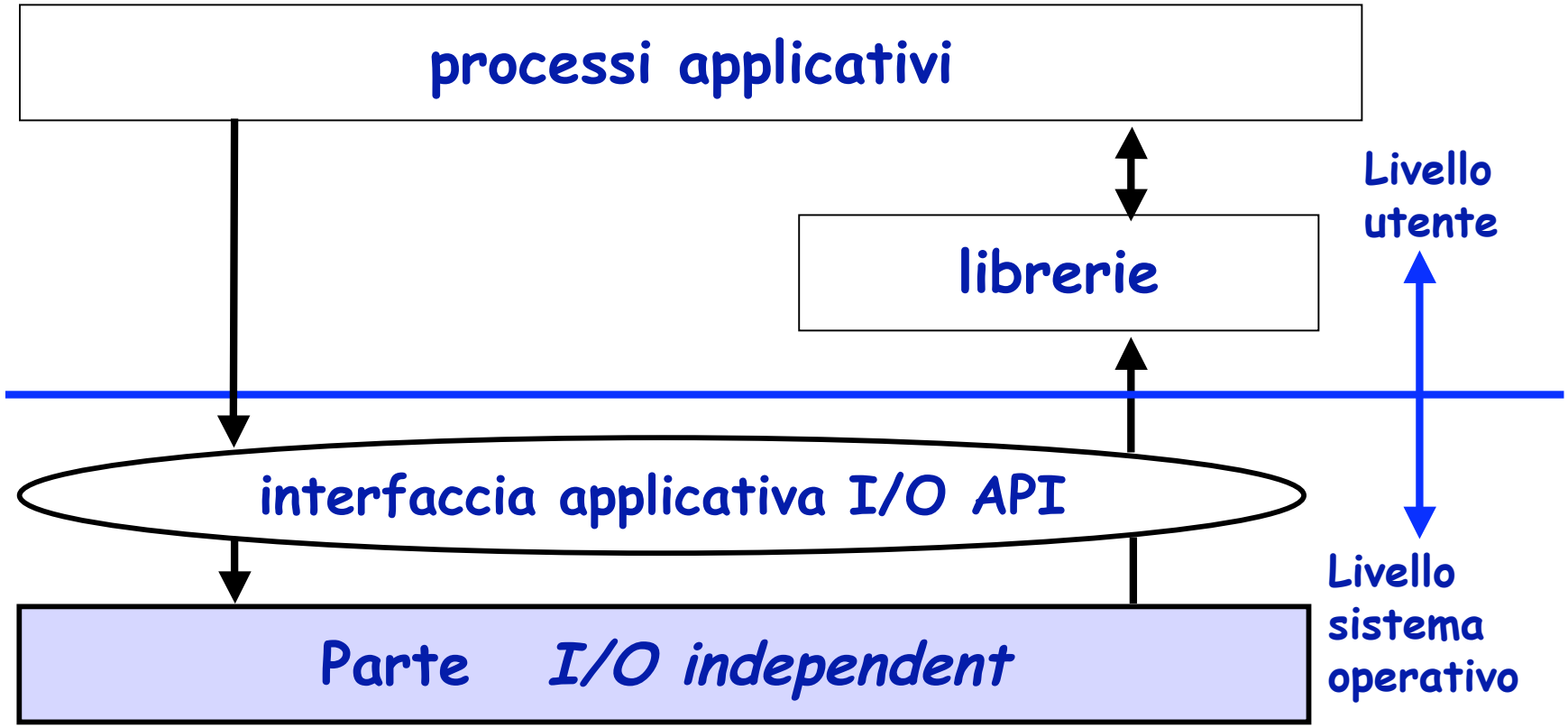
ARCHITETTURA DEL SOTTOSISTEMA DI I/O



ARCHITETTURA DEL SOTTOSISTEMA DI I/O:
parte dipendente dai dispositivi



ARCHITETTURA DEL SOTTOSISTEMA DI I/O:
parte indipendente dai dispositivi



LIVELLO INDIPENDENTE DAI DISPOSITIVI

FUNZIONI

- Naming
- Buffering
- Gestione malfunzionamenti
- Allocazione dei dispositivi ai processi applicativi

BUFFERING

Per ogni operazione di I/O il sistema operativo riserva un'area di memoria "tampone" (buffer), per contenere i dati oggetto del trasferimento.

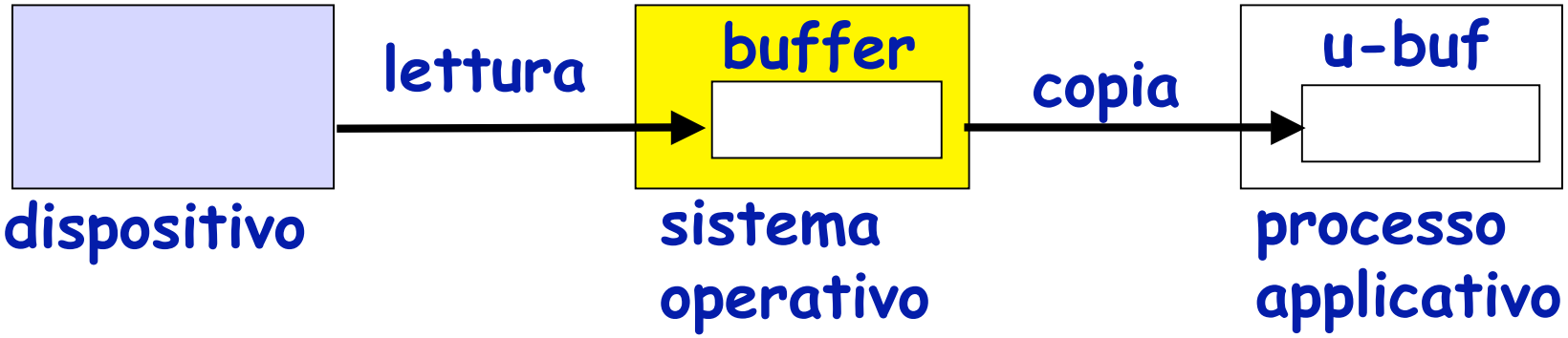
Motivazioni:

- differenza di velocita` tra processo e periferica: disaccoppiamento

- quantita` di dati da trasferire (es. dispositivi a blocchi): il processo puo` richiedere il trasferimento di una quantita` di informazioni inferiore a quella del blocco

BUFFERING

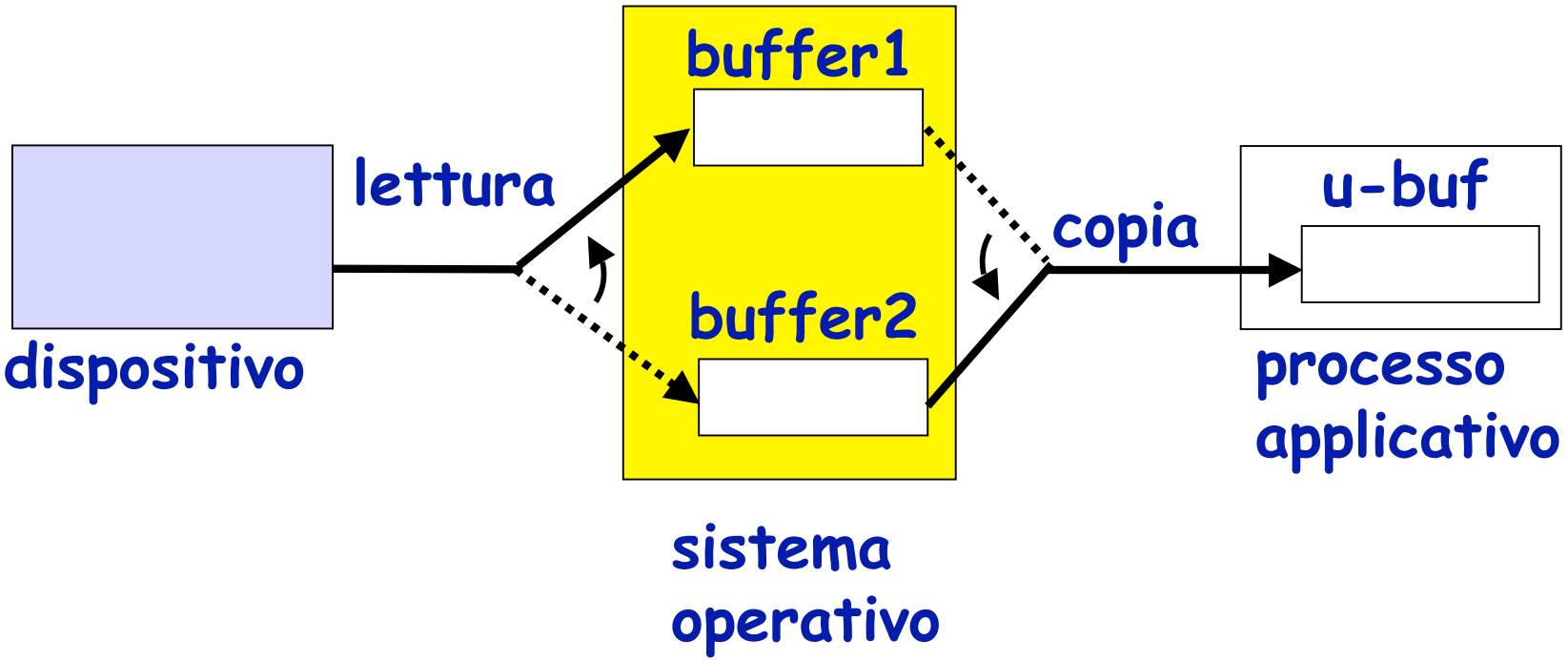
ES. operazione di lettura con singolo buffer



- Buffer: area tampone nella memoria del sistema operativo
- u-buf: area tampone nella memoria virtuale del processo applicativo

BUFFERING

ES. operazione di lettura con doppio buffer



GESTIONE MALFUNZIONAMENTI

- Tipi di gestione degli eventi anomali:
  - Risoluzione del problema (mascheramento dell'evento anomalo);
  - Gestione parziale e propagazione a livello applicativo;
- Tipi di eventi anomali:
  - Eventi propagati dal livello inferiore (es. guasto HW permanente;
  - Eventi generati a questo livello (es. tentativo di accesso a un dispositivo inesistente).

ALLOCAZIONE DEI DISPOSITIVI

- Dispositivi condivisi da utilizzare in mutua esclusione;
- Dispositivi dedicati ad un solo processo (*server*) a cui i processi *client* possono inviare messaggi di richiesta di servizio;
- Tecniche di *spooling* (dispositivi virtuali).

LIVELLO DIPENDENTE DAI DISPOSITIVI

Funzioni:

- fornire i gestori dei dispositivi (*device drivers*)
- offrire al livello superiore l'insieme delle funzioni di accesso ai dispositivi (interfaccia "*device-independent*"), es:

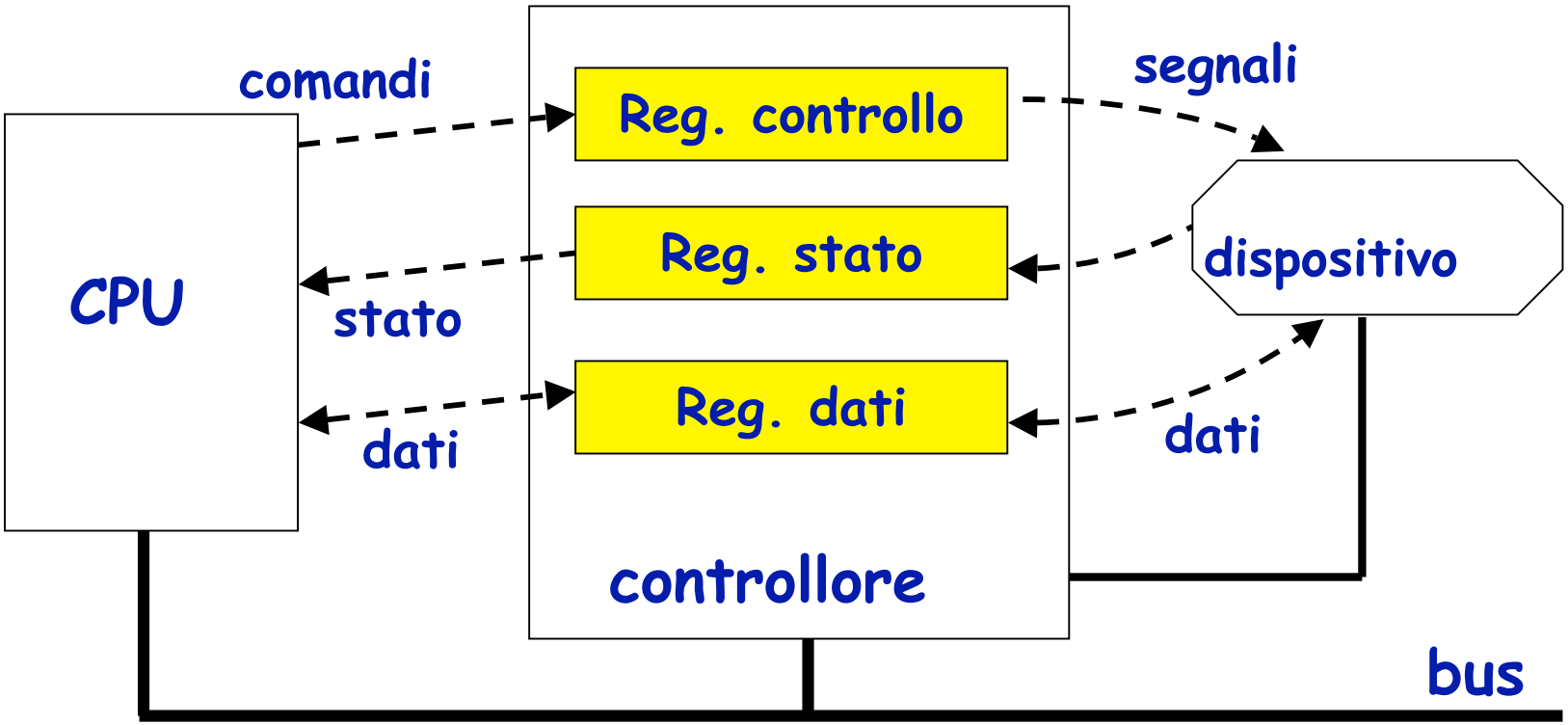
N=_read (disp, buffer, nbytes)

nome unico del dispositivo

Buffer di sistema

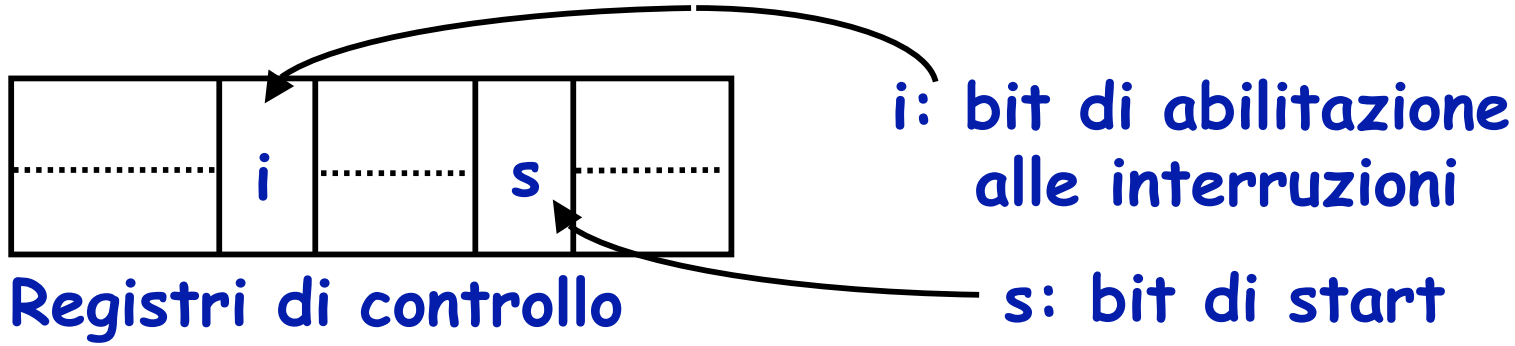
GESTORE DI UN DISPOSITIVO

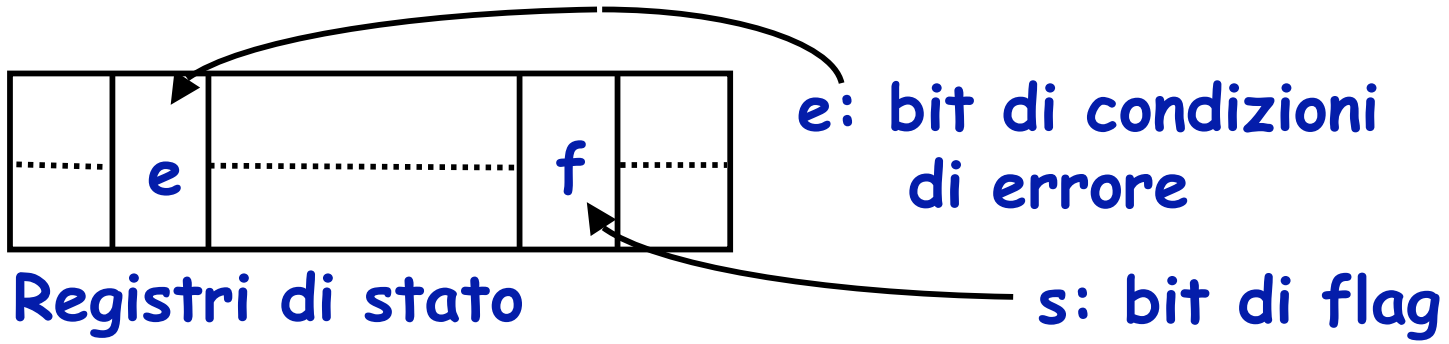
Schema semplificato di un controllore



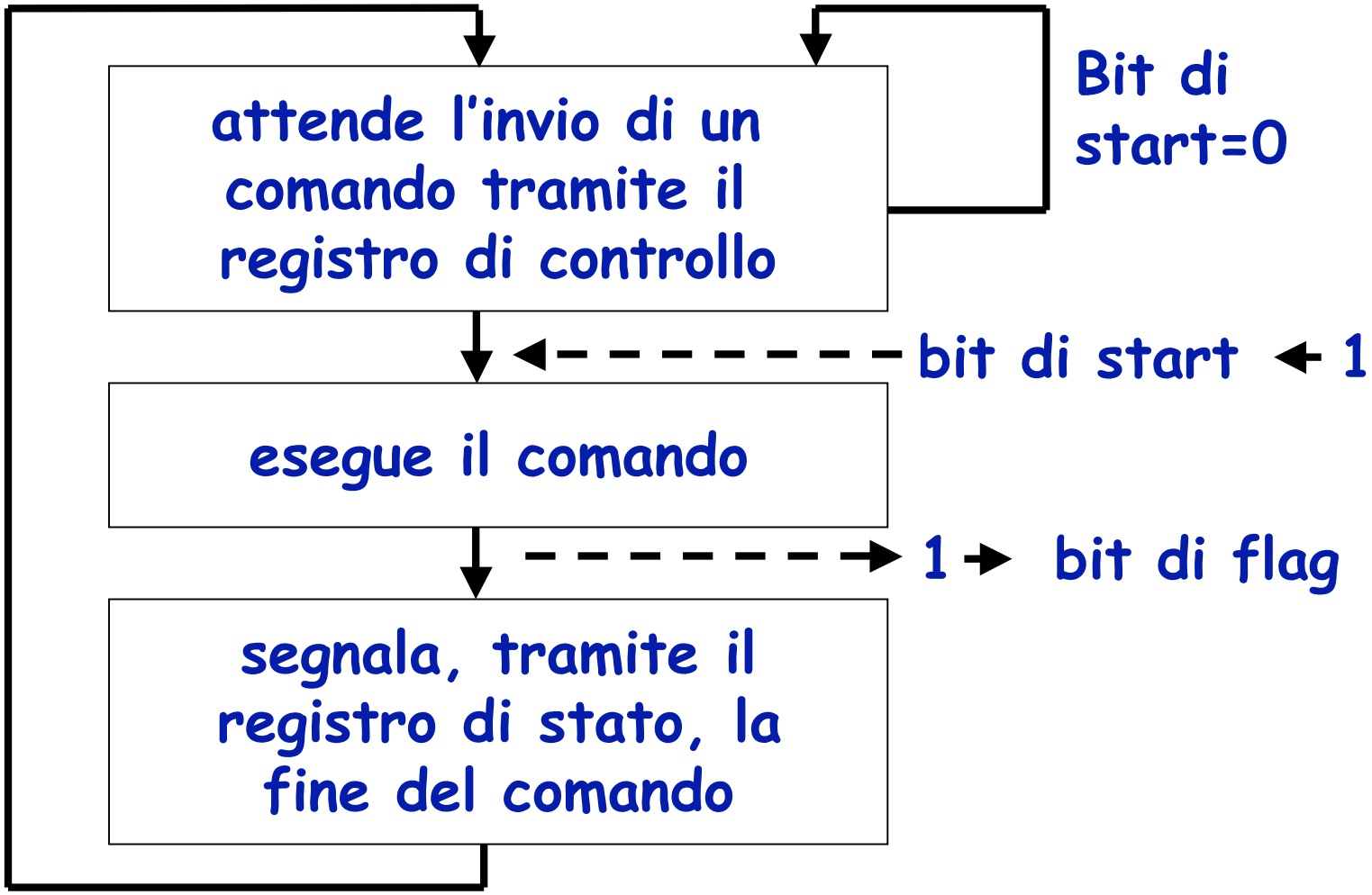
GESTORE DI UN DISPOSITIVO

Registri di stato e controllo





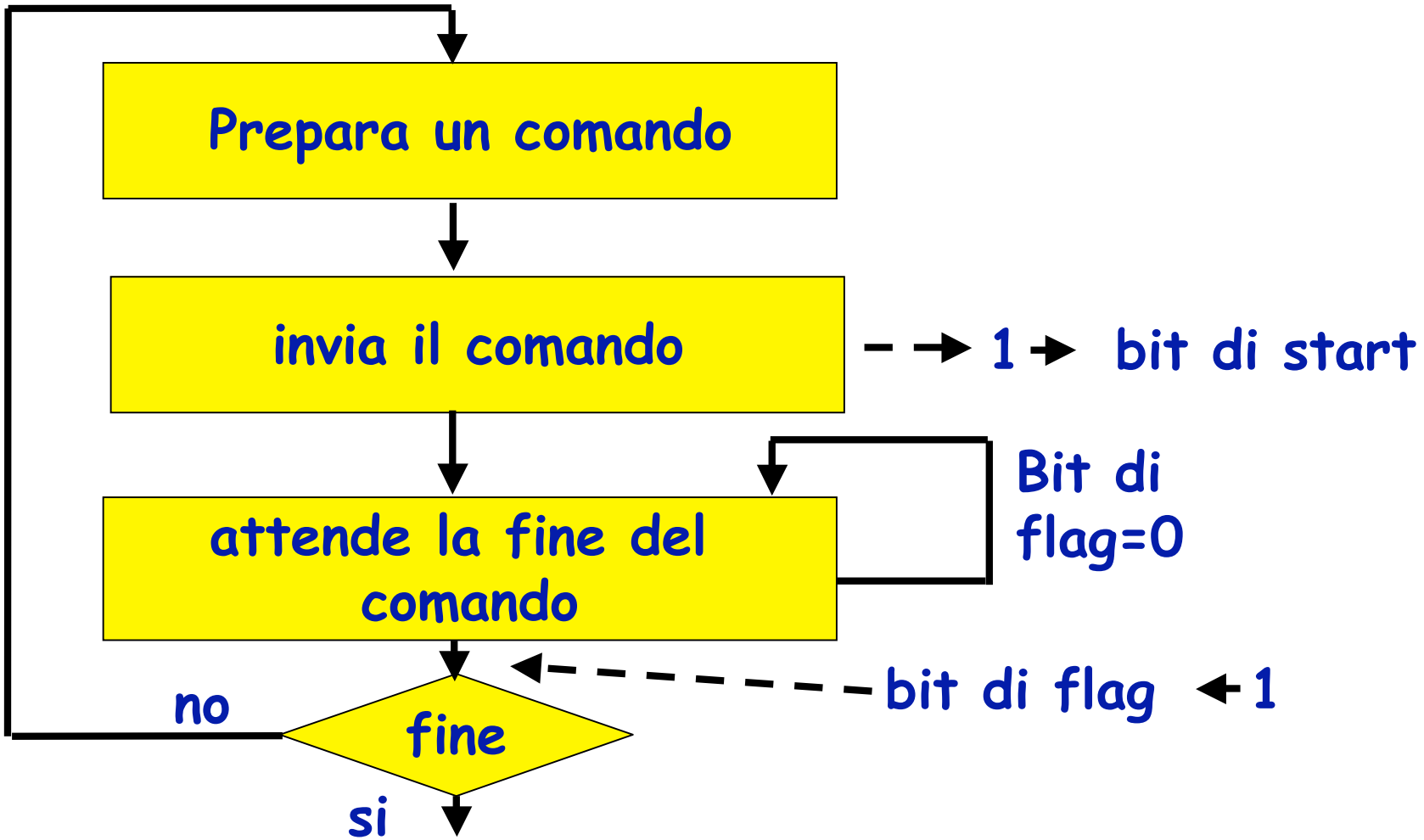
PROCESSO ESTERNO



PROCESSO ESTERNO

```
processo esterno
{
     while (true)
     {
          do{;} while (start ==0)//stand-by
          <esegue il comando>;
          <registra l'esito del comando
            ponendo flag = 1>;
     }
}
```

PROCESSO APPLICATIVO:
gestione a controllo di programma



PROCESSO APPLICATIVO

```
processo applicativo
{
      ...
      for (int i=0; i++; i<n)
      {    <prepara il comando>;
           <invia il comando>;
           do{;} while (flag ==0)
            //ciclo di attesa attiva
           <verifica l'esito>;
      }
      ...
}
```

GESTIONE A INTERRUZIONE

- Lo schema precedente viene detto anche "*a controllo di programma*".
- Non adatto per sistemi multiprogrammati a causa dei clicli di attesa attiva.
- Per evitare l'attesa attiva:
  - Riservare, per ogni dispositivo un semaforo:
    dato_disponibile
    (dato_disponibile = 0;)
  - Attivare il dispositivo abilitandolo a interrompere (ponendo nel registro di controllo il bit di abilitazione a 1).

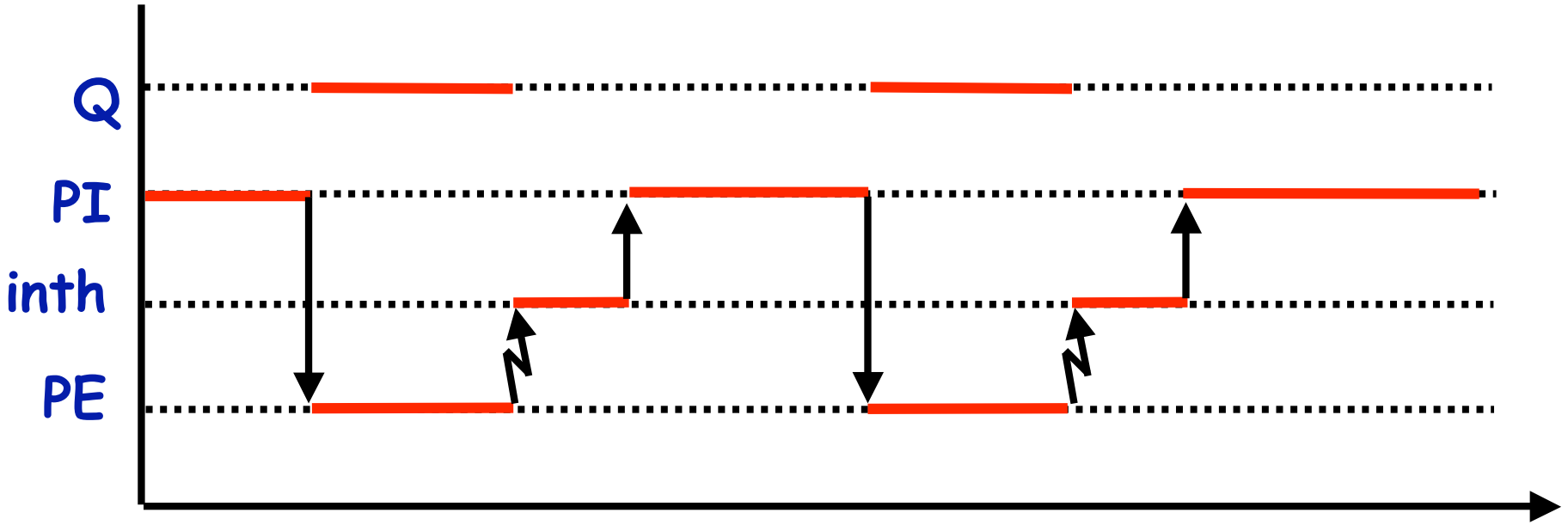
GESTIONE A INTERRUZIONE

```
processo applicativo
{
        ……..
        for (int i=0; i++; i<n)
        {    <prepara il comando>;
             <invia il comando>;
             p(dato_disponibile);
              <verifica l'esito>;
        }
         ……..
}
```

FUNZIONE DI RISPOSTA ALLE INTERRUZIONI

```
Interrupt_handler
{
      ...
      v(dato_disponibile);
      ...
}
```

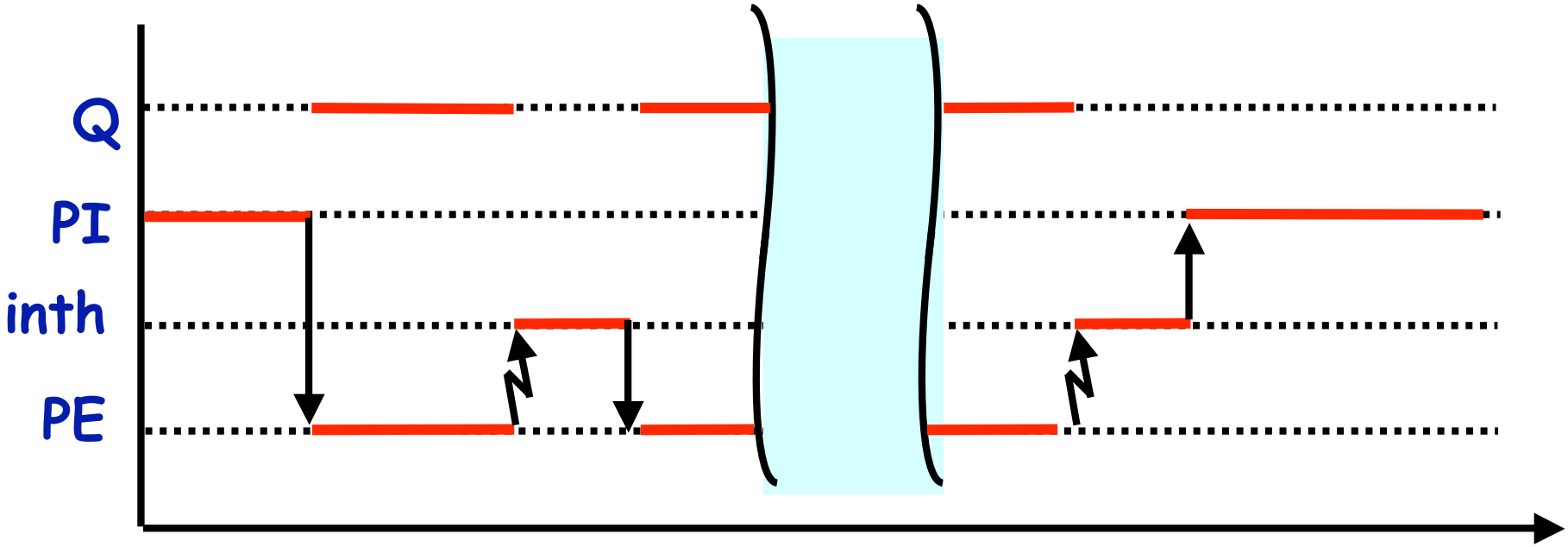
DIAGRAMMA TEMPORALE



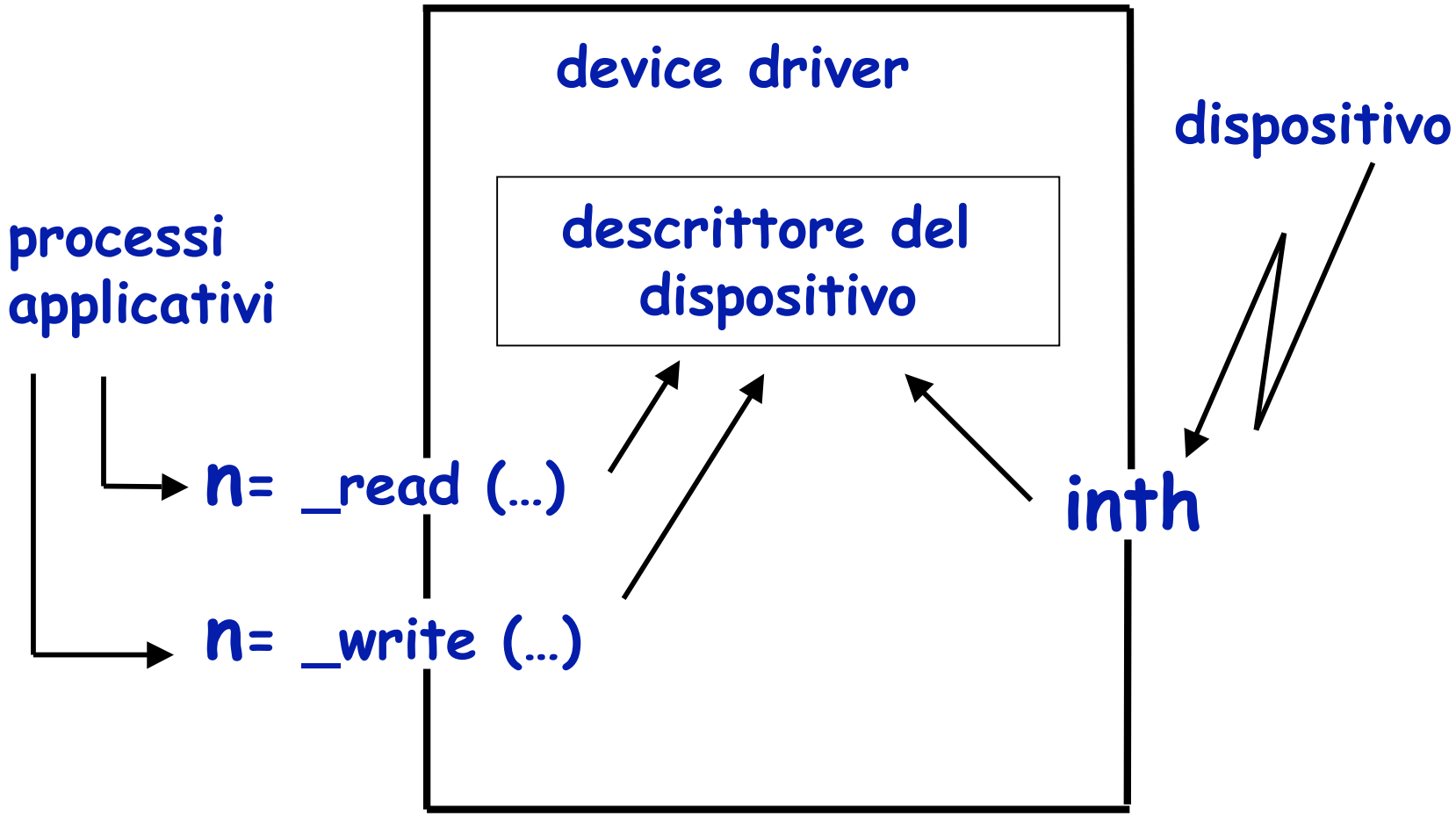
PI: processo applicativo che attiva il dispositivo
PE: processo esterno
Inth: routine di gestione interruzioni
Q: altro processo applicativo

DIAGRAMMA TEMPORALE

E` preferibile uno schema in cui il processo applicativo che ha attivato un dispositivo per trasferire n dati venga risvegliato solo alla fine dell'intero trasferimento:



ASTRAZIONE DI UN DISPOSITIVO



DESCRITTORE DI UN DISPOSITIVO

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**

DRIVER DI UN DISPOSITIVO

ESEMPIO:

```
int _read(int disp,char *buf,int cont)
```

CON:

- la funzione che restituisce -1 in caso di errore o il numero di caratteri letti se tutto va bene,
- **disp** è il nome unico del dispositivo,
- **buf** è l'indirizzo del buffer in memoria,
- **cont** il numero di dati da leggere

DRIVER DI UN DISPOSITIVO

```
int _read(int disp,char *buf,int cont)
{ descrittore[disp].contatore=cont;
  descrittore[disp].puntatore=buf;
  <attivazione dispositivo> ;
    p(descrittore[disp].dato_disponibile);
  if (descrittore[disp].esito== <cod.errore>)
        return (-1);
  return (cont-descrittore[disp].contatore);
}
```

DRIVER DI UN DISPOSITIVO

```
void inth() //interrupt handler
 { char b;
   <legge il valore del registro di stato>
   if (<bit di errore> == 0)
     {<ramo normale della funzione> }
   else
   {<ramo eccezionale della funzione> }
   return //ritorno da interruzione
 }
```

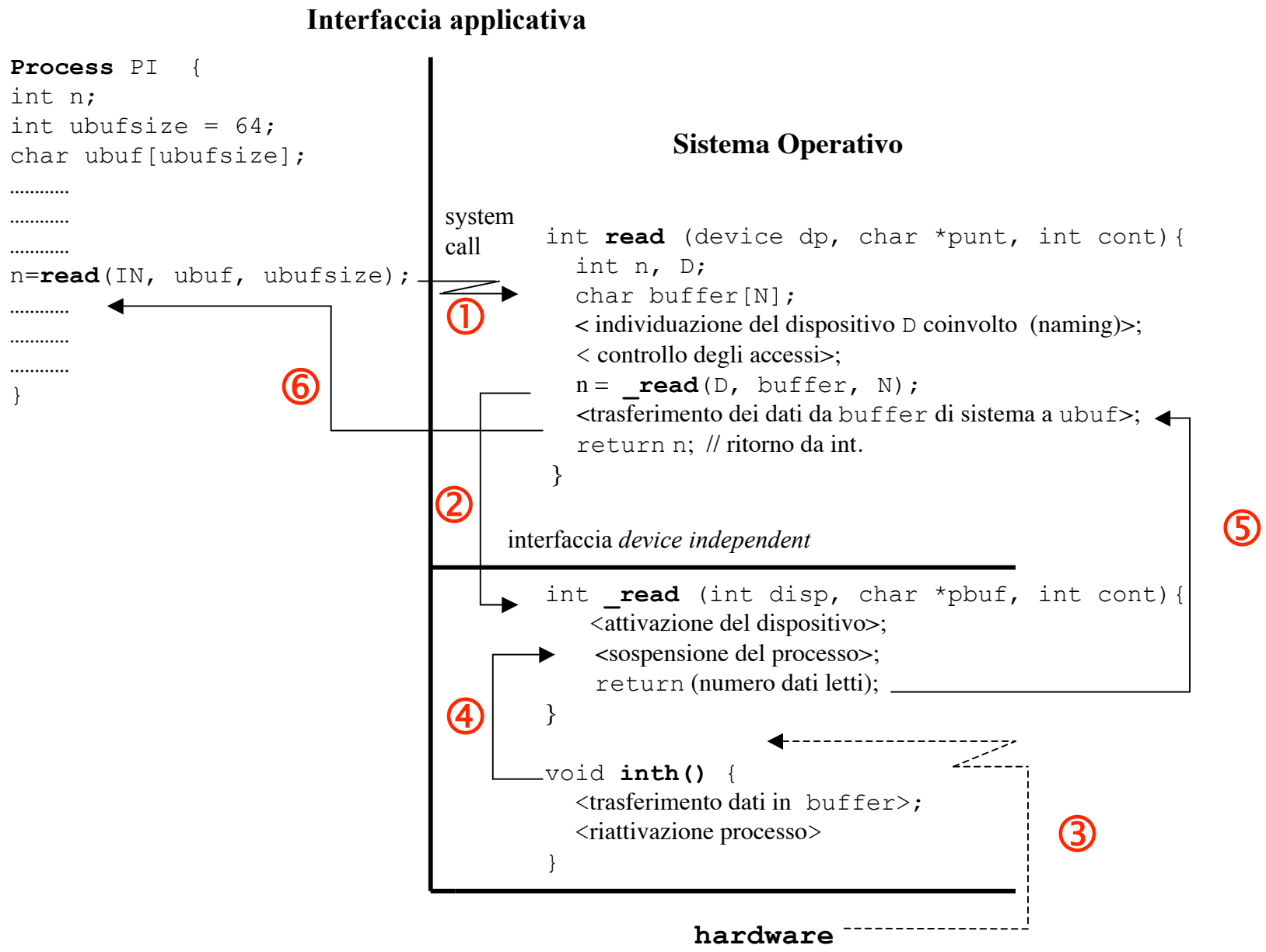
RAMO NORMALE DELLA FUNZIONE

```
{   < b = registro dati >;
    *(descrittore[disp].puntatore)= b;
    descrittore[disp].puntatore ++;
    descrittore[disp].contatore --;
    if (descrittore[disp].contatore!=0)
       <riattivazione dispositivo>;
    else
            {descrittore[disp].esito =
          <codice di terminazione corretta>;
        <disattivazione dispositivo>;
        v
(descrittore[disp].dato_disponibile);
      }
}
```

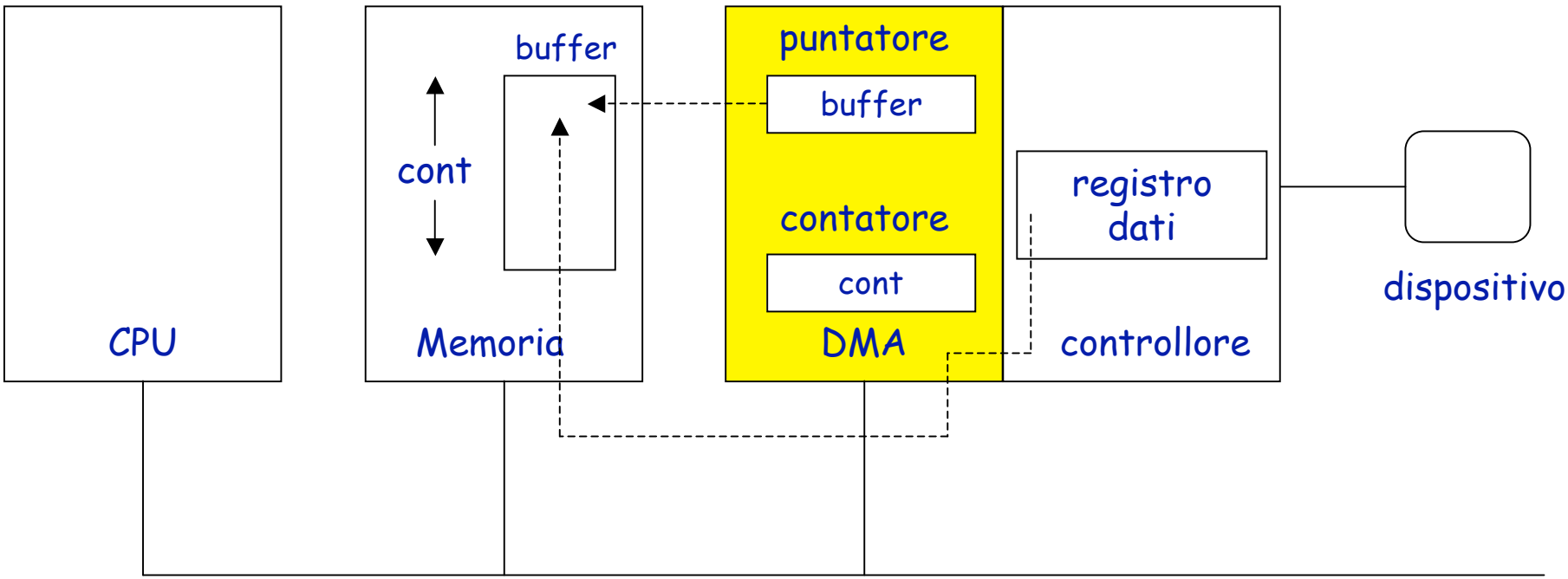
RAMO ECCEZIONALE DELLA FUNZIONE

```
{   <routine di gestione errore >;
    if (<errore non recuperabile>)
     {descrittore[disp].esito =
          <codice di terminazione anomala>;
     v(descrittore[disp].dato_disponibile);
     }
}
```

Flusso di controllo durante un trasferimento



Gestione di un dispositivo in DMA



Gestione del temporizzatore

- Per consentire la modalità di servizio a divisione di tempo è necessario che il nucleo gestisca un *dispositivo temporizzatore* tramite un'apposita procedura che, ad intervalli di tempo fissati, provveda a sospendere il processo in esecuzione ed assegnare l'unità di elaborazione ad un altro processo
- Gestione del clock: i dispositivi clock generano interruzioni periodiche (clock ticks) a frequenze stabilite; la gestione software delle interruzioni consente di ottenere alcuni servizi quali:
  - aggiornamento della data
  - gestione del quanto di tempo (sistemi time-sharing)
  - valutazione dell'impegno della CPU di un processo
  - gestione della system call ALARM
  - gestione del time-out (watchdog timers)

Il controllore del timer contiene, oltre ai registri di controllo e di stato, un registro contatore nel quale la CPU trasferisce un valore intero che viene decrementato dal timer.

Quando il registro contatore raggiunge il valore zero il controllore lancia un segnale di interruzione.

Nel descrittore della periferica timer sono presenti:

- un array di N semafori privati (fine_attesa[N]). Ciascun semaforo viene utilizzato per bloccare il corrispondente processo che chiama la delay.
- un array di interi utilizzato per mantenere aggiornato il numero di quanti di tempo che devono ancora passare prima che un processo possa essere riattivato

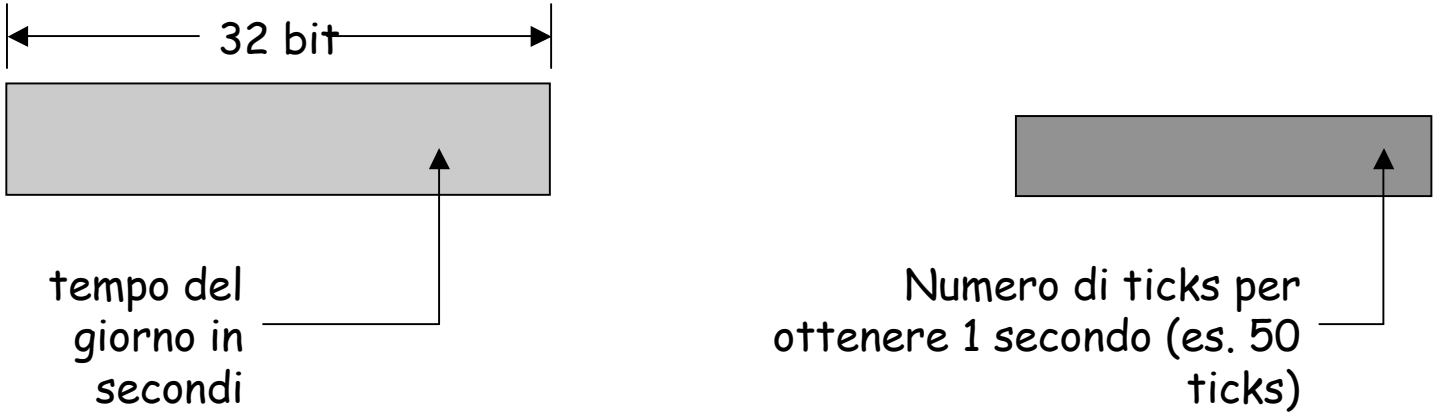
Descrittore del timer

```
void delay (int n) {
        int proc;
        proc=<indice del processo in esecuzione>;
         descrittore.ritardo[proc]= n;
        //sospensione del processo
        descrittore.fine_attesa[proc].wait();
}



void inth(){
        for(int=0; int<N, i++)
                if (descrittore.ritardo[i]!=0){
                        descrittore.ritardo [i]--;
                        if (descrittore.ritardo[i]==0)
                           descrittore.fine_attesa[proc].signal();
                }
}
```

- **Aggiornamento della data.** Il tempo del giorno viene mantenuto in secondi in un registro a 32 bit. Un contatore secondario conta i ticks (es.: frequenza 50 Hz) fino ad ottenere un secondo.

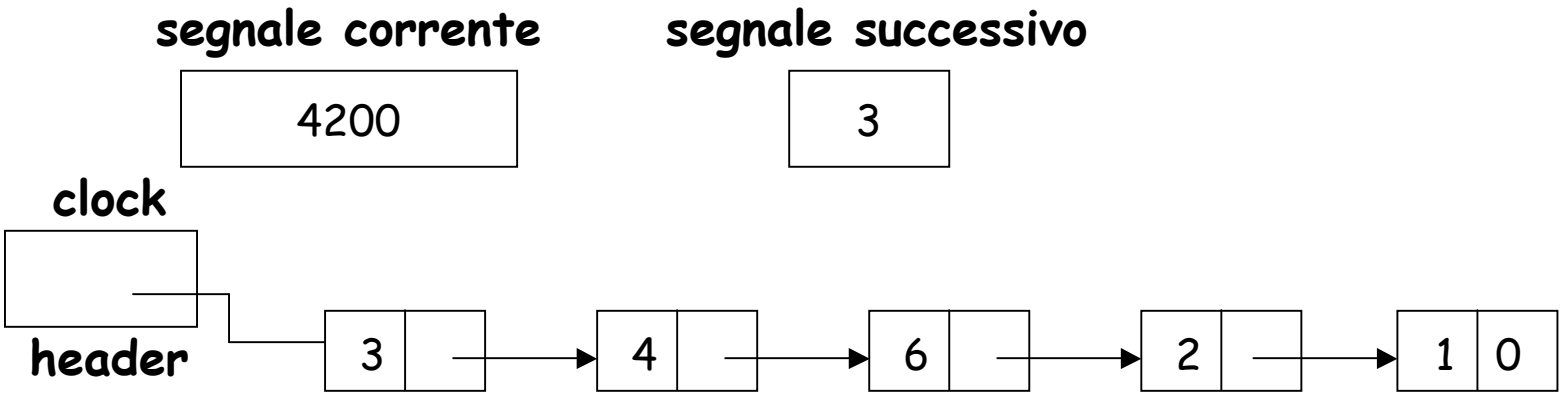


2^{32} secondi è circa 136 anni. Nei sistemi Unix il tempo si conta dal 1 gennaio 1970. Si avrà overflow del clock nel 2038 (l'intero a 32 bit è con segno).

• **Gestione del quanto di tempo.** Quando un processo inizia l'esecuzione, viene inizializzato un contatore con il valore del quanto di tempo espresso in clock ticks. Ad ogni interruzione il contatore è decrementato di 1. Quando raggiunge il valore zero, lo scheduler sottrae la CPU al processo.

• **Valutazione dell'impegno di CPU di un processo.** Ad ogni clock tick viene incrementato di 1 un campo contenuto nel descrittore del processo in esecuzione (problema delle interruzioni che possono avvenire durante l'esecuzione del processo).

- **System Call ALARM.** Un processo può richiedere al S.O. un segnale, un'interruzione, un messaggio, etc. dopo un certo tempo (es. pacchetto inviato sulla rete deve essere ritrasmesso, se non riconosciuto, entro un intervallo di tempo).

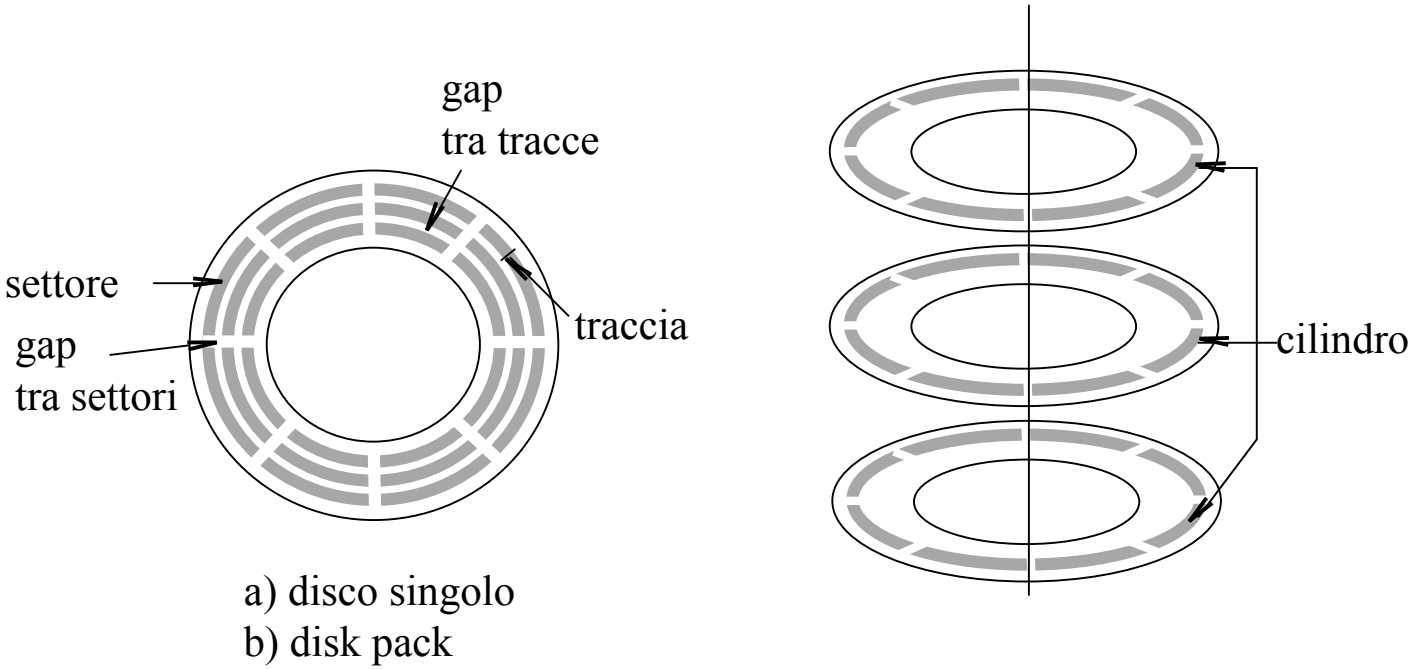


•Nel caso siano attivi più segnali, si simula la presenza di più clock tramite una lista ordinata. Ogni elemento della lista definisce quanti clock ticks occorre attendere per il prossimo segnale dopo il precedente. I segnali sono attesi a 4203, 4207, 4213, 4215, 4216.

•Ad ogni tick il valore di "segnale successivo" (nell'esempio, 3) viene decrementato di 1. Quando diventa zero, viene generato il segnale corrispondente al primo elemento della lista. Questo viene rimosso dalla lista e "segnale successivo" viene aggiornato (nell'esempio, 4).

Gestione e organizzazione dei dischi

Organizzazione fisica



a) disco singolo
b) disk pack

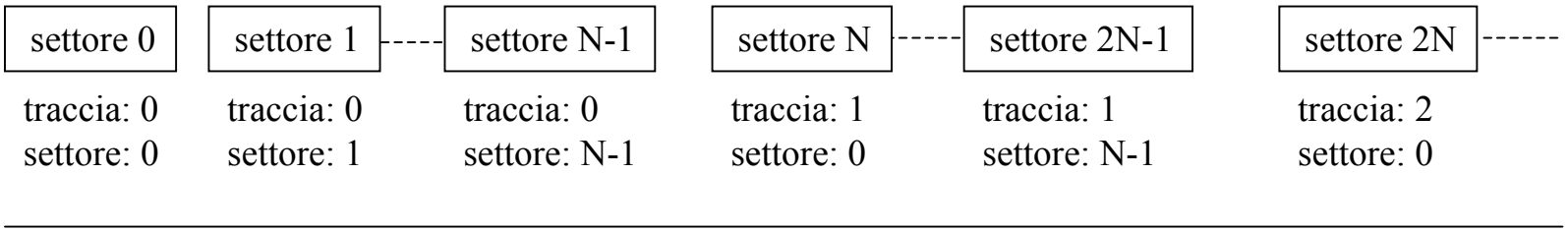
Indirizzo di un settore (blocco fisico):

(f,t,s)

dove:

f numero della ***faccia***, t numero della ***traccia*** nell'ambito della faccia, s numero del ***settore*** entro la faccia.

Tutti i settori che compongono un disco (o un pacco di dischi), vengono trattati come un array.



Indicando con

M il numero di tracce per faccia

N numero di settori per traccia

un settore di coordinate (f,t,s) viene rappresentato nell'ambito dell'array con l'indice i

$$i = f*M*N + t*N + s$$

Scheduling delle richieste di trasferimento

TF= TA+TT

TF tempo medio di trasferimento di un settore (per leggere o scrivere un settore)
TA tempo medio di accesso (per posizionare la testina di lettura/ scrittura all'inizio del settore considerato)
TT tempo di trasferimento dei dati del settore

TA= ST + RL

ST tempo di *seek* (per posizionare la testina sopra la traccia contenente il settore considerato)
RL *rotational latency* (tempo necessario perché il settore ruoti sotto la testina)

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 μs	19 μs

Tabella 5.2 parametri caratterizzanti i due dischi WD AC2540 e WDE18300.

TT tempo necessario per far transitare sotto la testina **l'intero settore**. Indicando con **t** il tempo necessario per compiere un giro, **s** il numero di settori per traccia, si ha

TT= t/s (valore approssimato, ms).

Quindi:

TF= ST+RL+TT

Il tempo medio di trasferimento dipende sostanzialmente dal tempo medio di accesso (ST e RL).

Due modi di intervento:

- Criteri con cui i *dati sono memorizzati* su disco (metodo di allocazione dei file)
- Criteri con cui *servire le richieste* di accesso (politiche di scheduling delle richieste)

Politiche di Scheduling delle Richieste

Nella valutazione del tempo medio di attesa di un **processo**, e` **necessario** tenere in conto anche il tempo durante il quale il processo attende che la sua richiesta di accesso venga servita.

Le richieste in coda ad un dispositivo possono essere servite secondo diverse politiche:

- First-Come-First-Served (FCFS)
- Shortest-Seek-Time-First (SSTF)
- SCAN algorithm
- C-SCAN (Circular-SCAN)

FCFS. Le richieste sono servite rispettando il tempo di arrivo. Si evita il problema della *starvation*, ma non risponde ad alcun criterio di ottimalità.

SSTF. Seleziona la richiesta con **tempo di seek minimo** a partire dalla posizione attuale della testina; può provocare situazioni di *starvation*

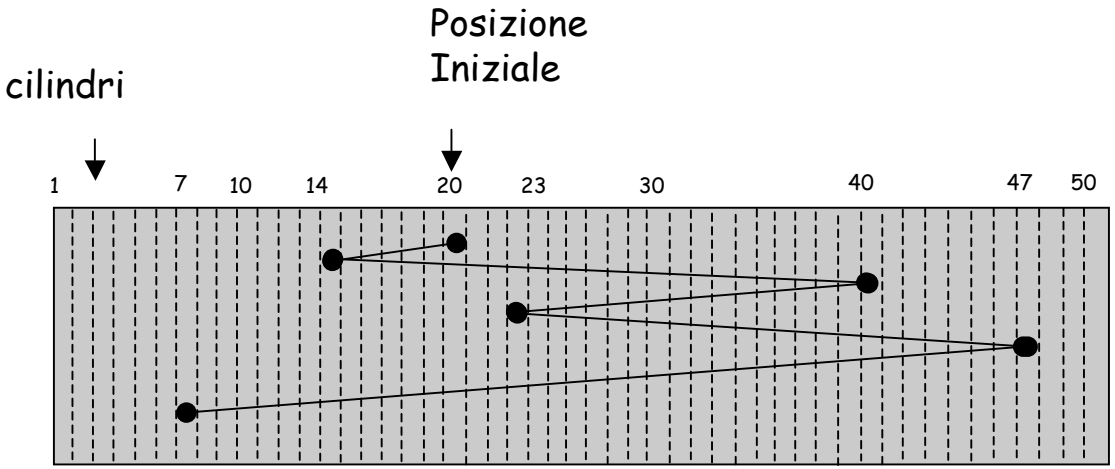
SCAN. La testina si porta ad una estremità del disco e si sposta verso l'altra estremità, servendo le richieste man mano che viene raggiunta una traccia, fino all'altra estremità del disco. Quindi viene invertita la direzione.

CSCAN. Fornisce un tempo di attesa più uniforme. Arrivata alla fine del disco la testina, essa torna immediatamente all'inizio del disco.

Testina posizionata sul cilindro 20. Richieste presenti in coda: 14, 40, 23, 47, 7

Algoritmo di scheduling FIFO

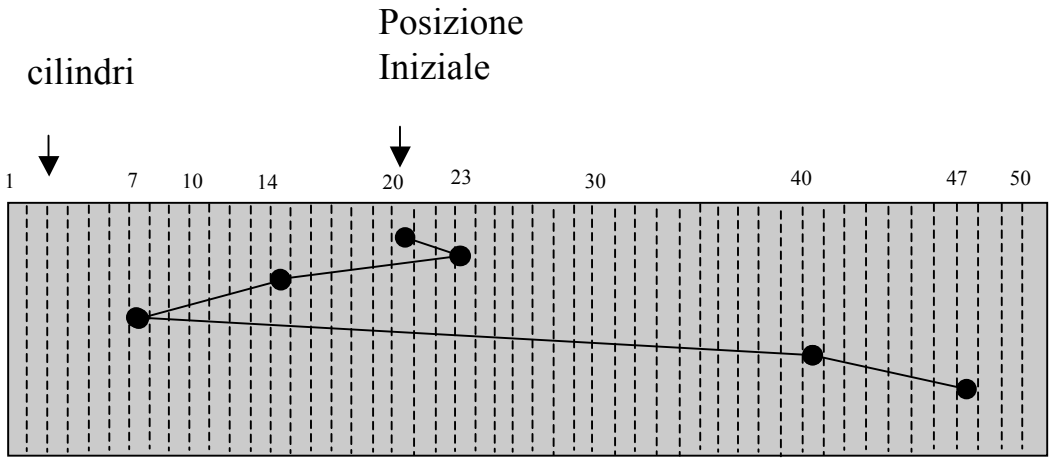
Testina posizionata sul cilindro 20. Richieste presenti in coda: [14, 40, 23, 47, 7]



Spostamento totale = 113cilindri

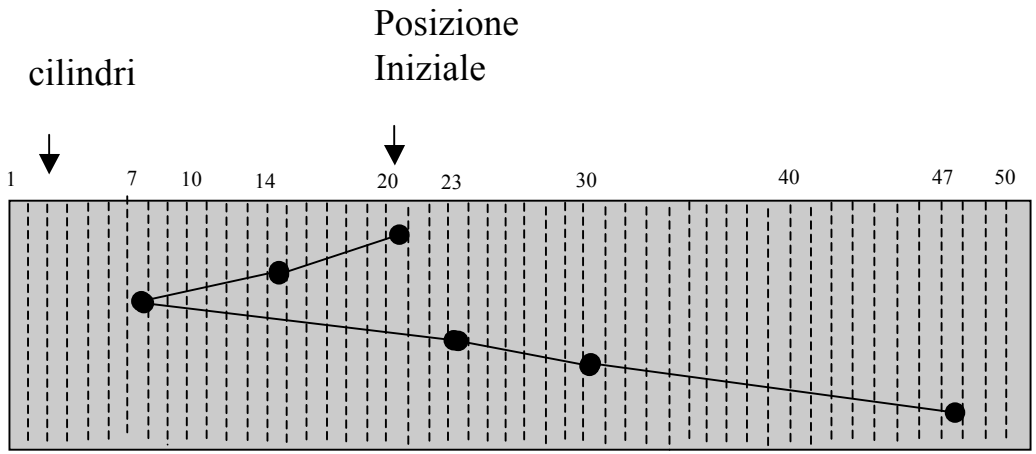
Algoritmo di Scheduling SSTF

Testina posizionata sul cilindro 20. Richieste presenti in coda: [14, 40, 23, 47, 7]



Spostamento totale = 59 cilindri

Algoritmo di Scheduling SCAN



Spostamento totale = 53 cilindri

C-SCAN (circular scan).
Ipotizzando una distribuzione uniforme per le richieste relative alle varie tracce, quando la testina inverte la direzione (SCAN) sono presenti poche richieste in *quanto servite di recente.*

La maggior densità di richieste è presente *all'altra estremità del disco*. Queste richieste sono quelle con *maggior tempo di attesa.*

CSCAN-> la direzione non viene mai invertita: alla fine di una scansione si ricomincia dall'estremità opposta.

Dischi RAID (Redundant Array of Independent Disks)

- Miglioramento delle prestazioni delle memorie di massa: problemi tecnologici.
- **Soluzione**: utilizzo di più dischi contemporaneamente che consentano di operare in parallelo .
- Organizzazione dei dati sui dischi in modo da ottenere **parallelismo** (= efficienza) **e ridondanza** (= affidabilita`).

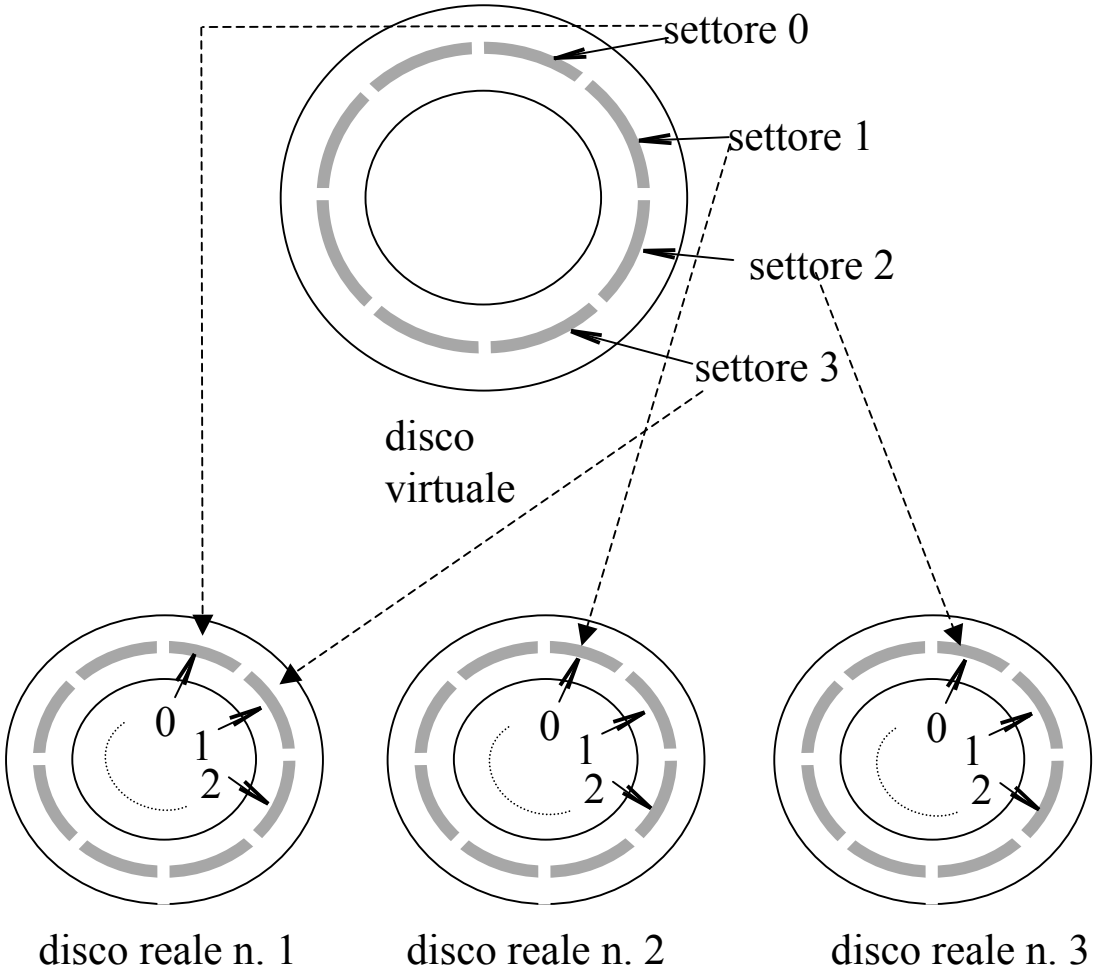
- Definizione di **standard** per la organizzazione dei dati.

Obiettivo: **unico disco virtuale** caratterizzato da grande *capacità*, alta *velocità di ingresso* e alta *affidabilità*.

➔ **RAID:** standard più diffuso.

- *7 diverse varianti* dello standard RAID: **livelli 0,1,2,3,4,5,6** che dipendono dal grado di affidabilità e rapidità di accesso.

Schema utilizzato nello standard RAID per la parallelizzazione degli accessi (livello 0)



Organizzazione dei dati

- Tutti i dati sono visti come appartenenti ad un *disco virtuale.*

- Le informazioni residenti su disco virtuale sono memorizzate suddividendole sui dischi reali, ad esempio a livello di *settore.*

- Una **traccia del disco virtuale** contiene un numero di settori pari ad **n volte quello di una traccia di un disco reale**.

- I settori sono distribuiti ***round-robin.***

- Possibilità di velocizzare tutte le operazioni di I/O che richiedono di operare su un insieme contiguo di settori del disco virtuale. Livello massimo di parallelismo pari a n.

Livello 0

Non prevede alcun livello di ridondanza dei dati

Livello 1

• Ridondanza ottenuta tramite la **duplicazione dei dati**.
Ogni settore virtuale viene mappato su una coppia di dischi fisici (**mirroring**).
➔ **Alta efficenza, alta affidabilità ma costi elevati.**

Operazioni di lettura: su uno qualunque dei due dischi, quello che richiede il minor tempo di ricerca.
Possibilita` di lettura in parallelo di settori allocati sullo stesso disco.
Operazioni di scrittura: possono procedere in parallelo su entrambi i dischi(la durata dell'operazione è vincolata dal tempo di scrittura più lungo.

• Si usa come funzione di *backup* per dati critici.

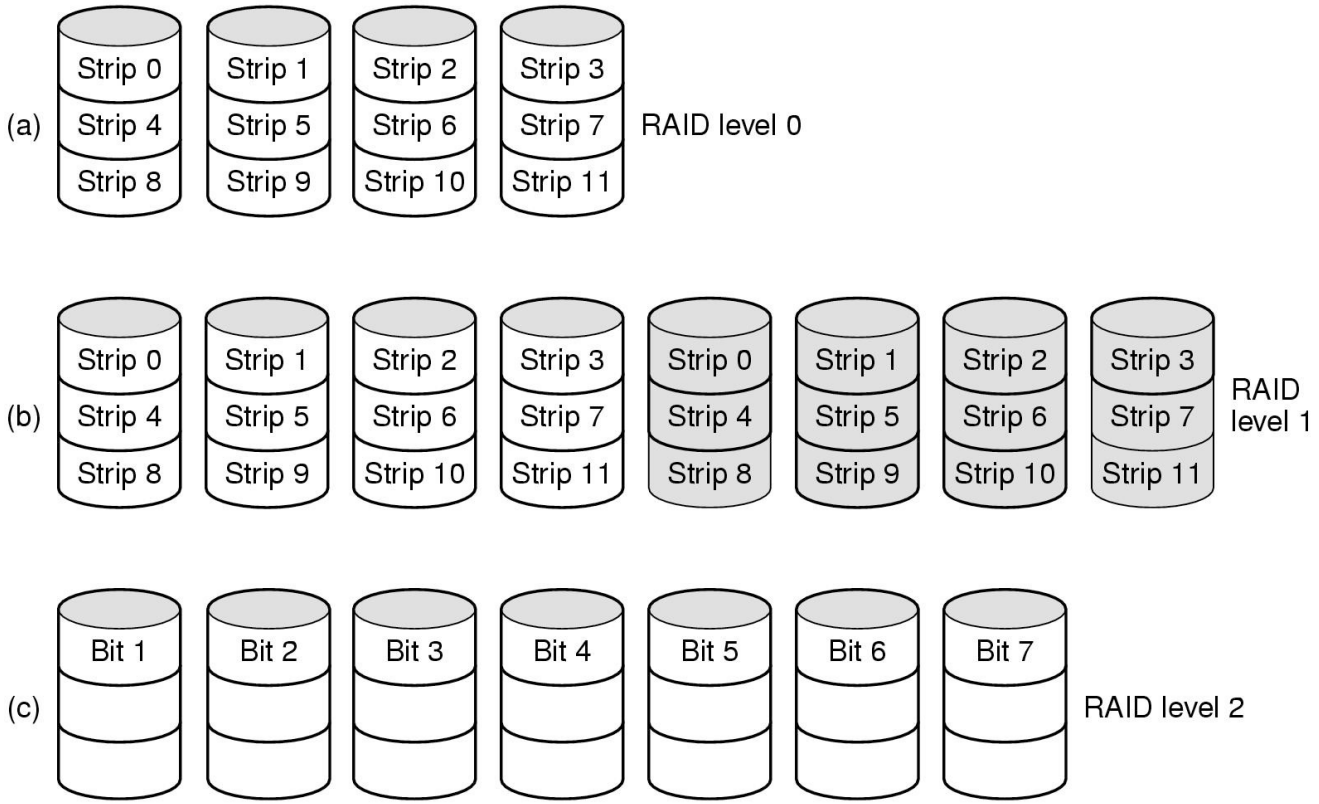
I livelli successivi si differenziano dal precedente schema per il livello di ridondanza richiesto e per i meccanismi di rilevazione e recupero da guasti (schemi di codifica Hamming o meccanismi di controllo di parità).

Livello 2

• *Il codice di correzione di errore* viene calcolato sui bit corrispondenti di ogni disco e i bit del codice sono memorizzati nelle corrispondenti posizioni *in dischi di parità multipli* (utilizzo del codice di Hamming, corregge errori di bit singoli e rileva errori di bit doppi).

➔Bit diversi della stessa parola sono allocati su dischi diversi

Costi elevati: necessita` di sincronizzare tutti i dischi (posizione della testina e rotazione).


(a)
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
RAID level 0
(b)
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
RAID level 1
(c)
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
RAID level 2

Livelli 3,4,5

• Utilizzo di un insieme di bit di parità per l'insieme (**parola** nel livello 3, **settore** nei livelli 4-5) di tutti i bit che si trovano in posizioni corrispondenti su tutti i dischi.

• In caso di un guasto di un disco si accede al disco di parità e i dati vengono ricostruiti utilizzando i dischi rimanenti.

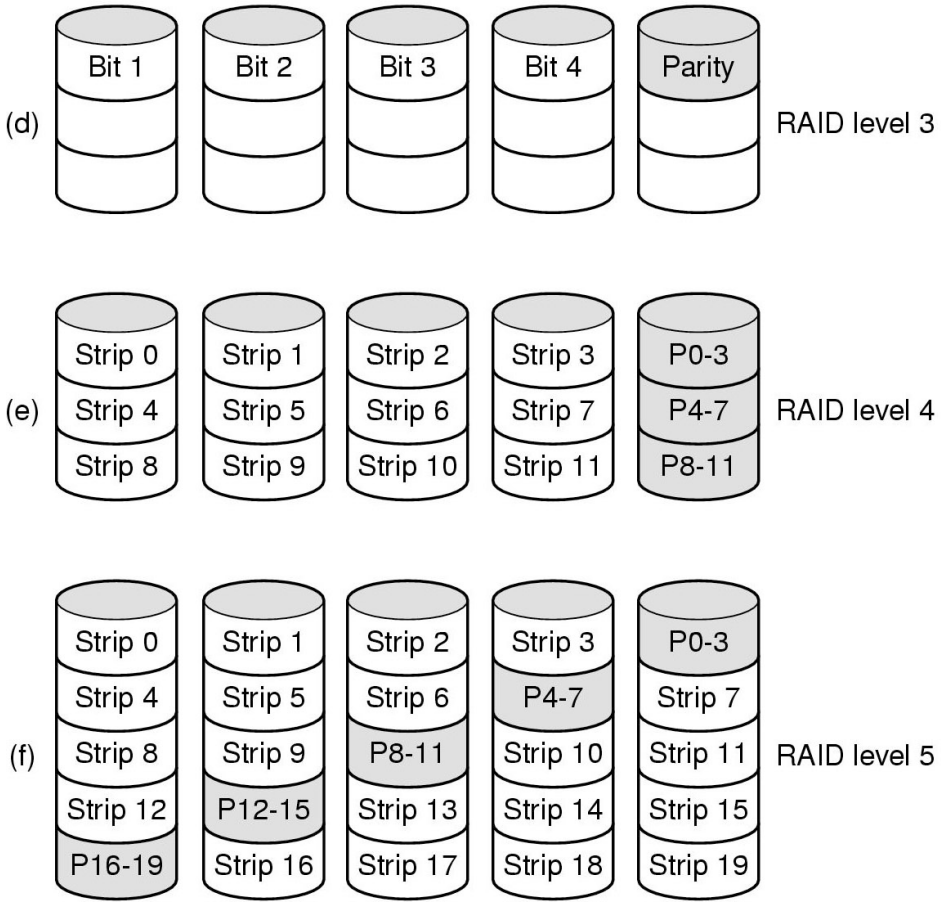
Esempio: Array di 5 dischi: x0,x1,x2,x3 contegono dati, x4 è il disco di parità.

Parità per il bit i-esimo:

$$x4(i) = x3(i) \oplus x2(i) \oplus x1(i) \oplus x0(i)$$

da cui

$$x1(i) = x4(i) \oplus x3(i) \oplus x2(i) \oplus x0(i)$$


(d)
Bit 1
Bit 2
Bit 3
Bit 4
Parity
RAID level 3
(e)
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
P0-3
P4-7
P8-11
RAID level 4
(f)
Strip 0
Strip 4
Strip 8
Strip 12
P16-19
Strip 1
Strip 5
Strip 9
P12-15
Strip 16
Strip 2
Strip 6
P8-11
Strip 13
Strip 17
Strip 3
P4-7
Strip 10
Strip 14
Strip 18
P0-3
Strip 7
Strip 11
Strip 15
Strip 19
RAID level 5

• Nel caso di n dischi, per ogni gruppo di n settori consecutivi, memorizzati sugli n dischi dell'array, viene calcolato un n+1esimo settore contenente *bit di parità.*

• Il primo bit del settore di parità corrisponde al bit di parità calcolato sui primi bit degli n settori regolari. Analogamente per gli altri bit di ogni settore.

Vantaggi per l'affidabilità: se uno qualunque dei settori di un disco si corrompe per un guasto il suo contenuto può essere recuperato utilizzando il corrispondente *settore di parità.*

➔Ogni operazione di scrittura coinvolge il settore di parità.

Livelli 3,4

I bit di parità vengono memorizzati su un nuovo disco (**disco di parità**). Criticità

Livello 5 (Block interleaved parity)

I settori di parità vengono allocati su tutti i dischi esistenti in modo circolare (*round-robin*).