GESTIONE DELLE PERIFERICHE D'INGRESSO/USCITA

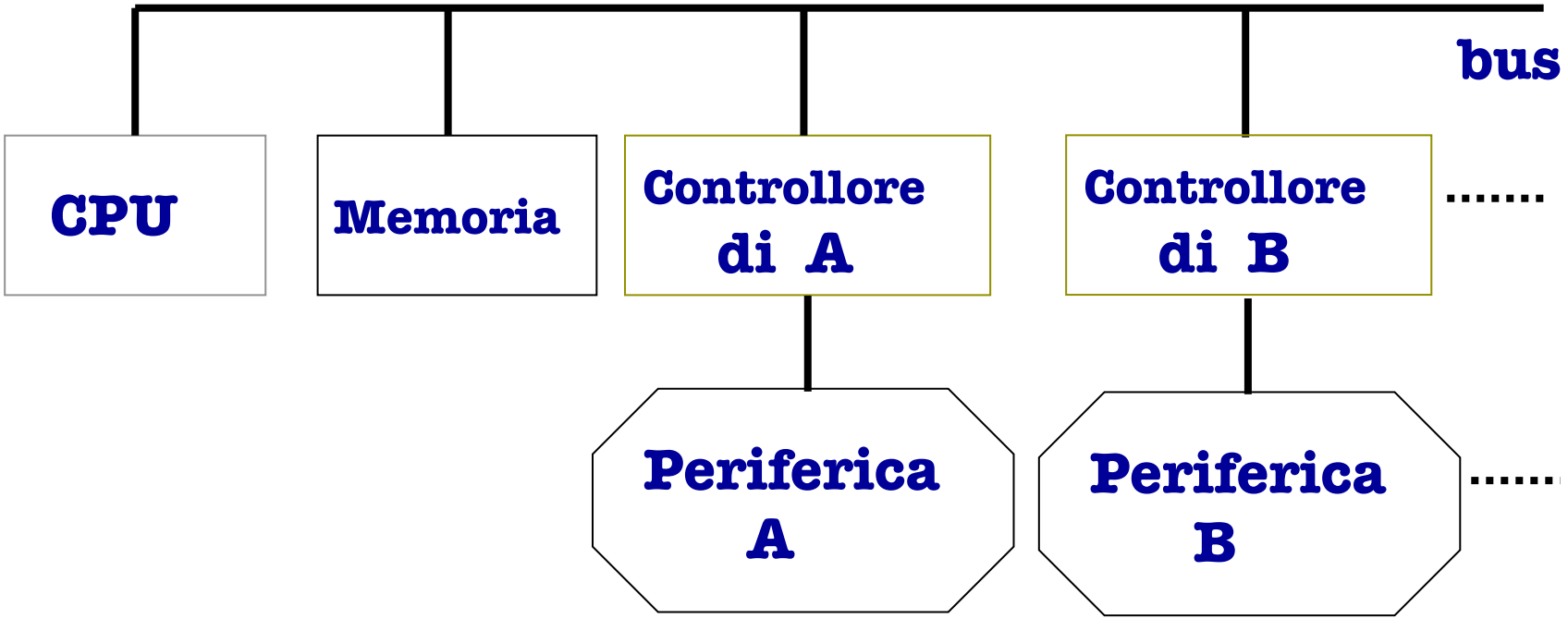
- **Compiti del sottosistema di I/O**
- **Architettura del sottosistema di I/O**
- **Gestore di un dispositivo di I/O**

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 ogni 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

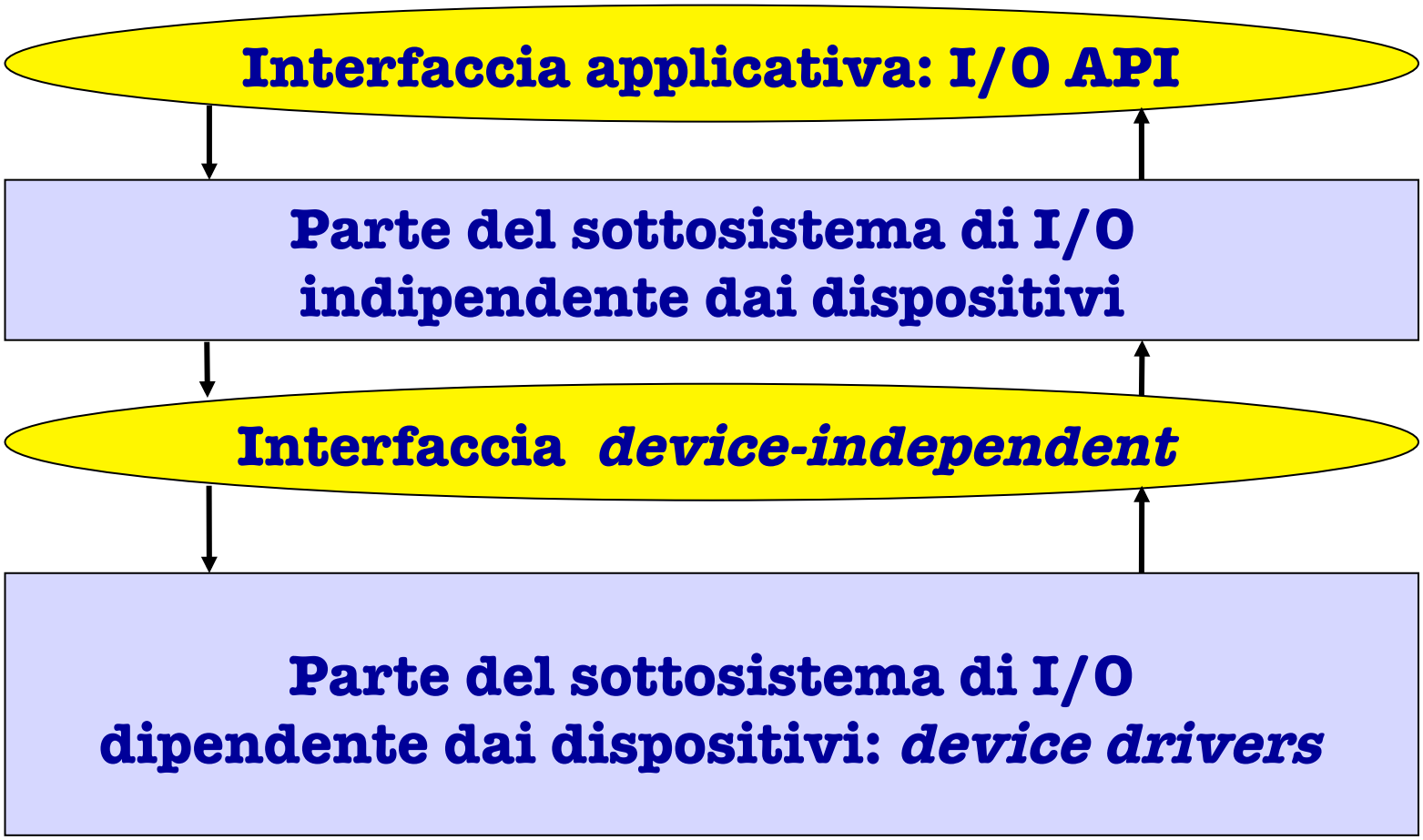
- Uso di nomi unici (valori numerici) all'interno del sistema per identificare in modo univoco i dispositivi;
- Uso di nomi simbolici da parte dell'utente (*I/O API Input/Output Application Programming Interface*);
- Uniformità col meccanismo di *naming* del file-system.

COMPITI DEL SOTTOSISTEMA DI I/O

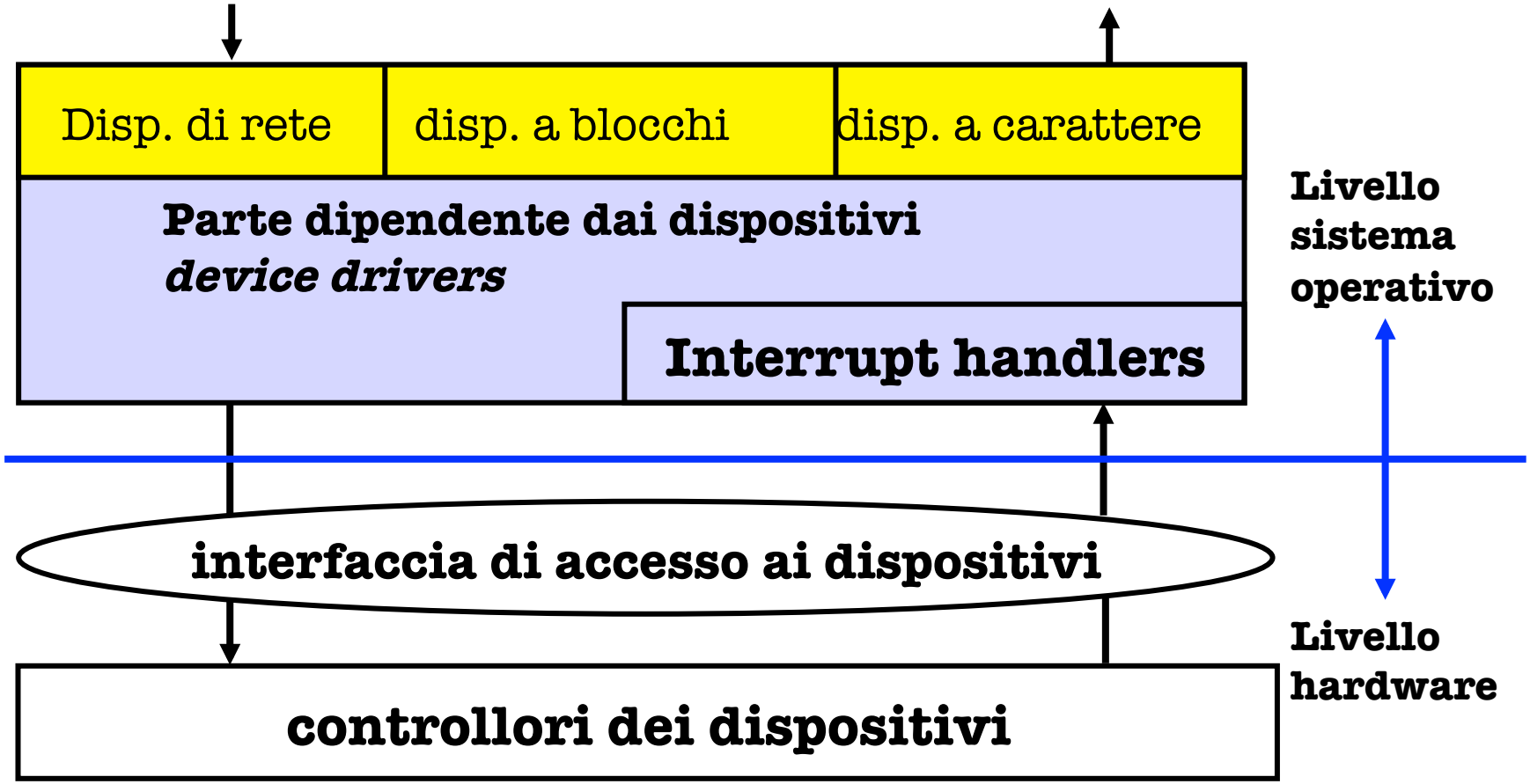
5. Garantire la corretta sincronizzazione tra un processo applicativo che ha attivato un trasferimento dati e l'attività del dispositivo.

- 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.

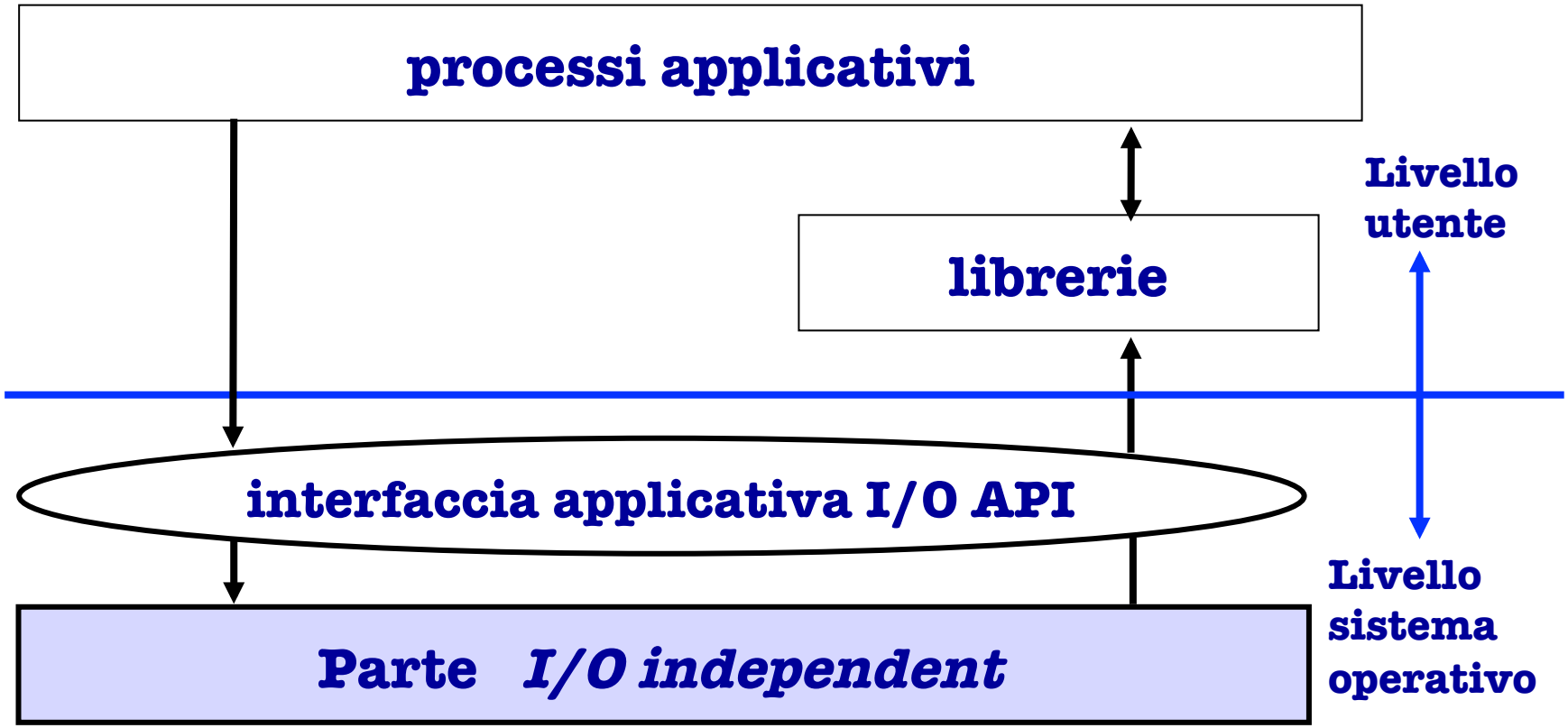
ARCHITETTURA DEL SOTTOSISTEMA DI I/O



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

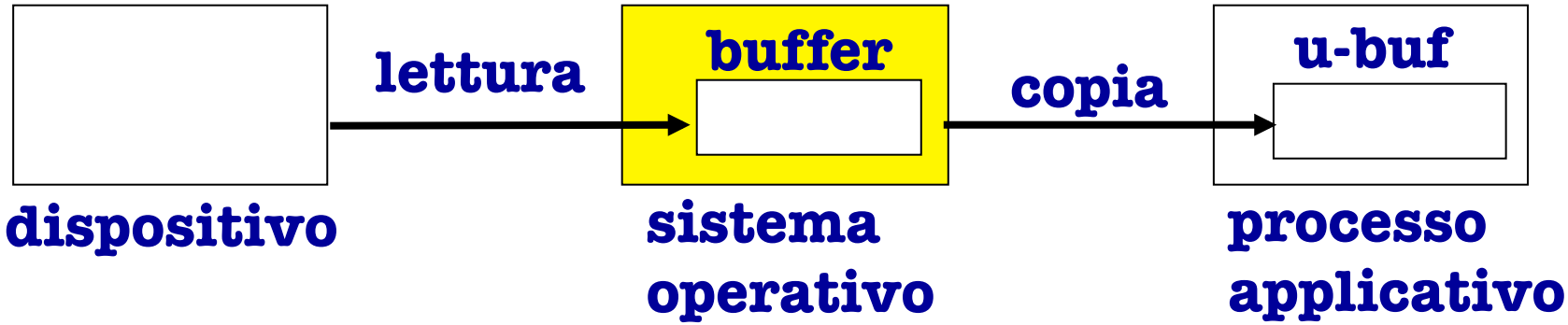
Per ogni operazione di I/O il sistema operativo riserva un'area di memoria "tampone" (buffer), per contenere i dati oggetto del trasferimento.

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

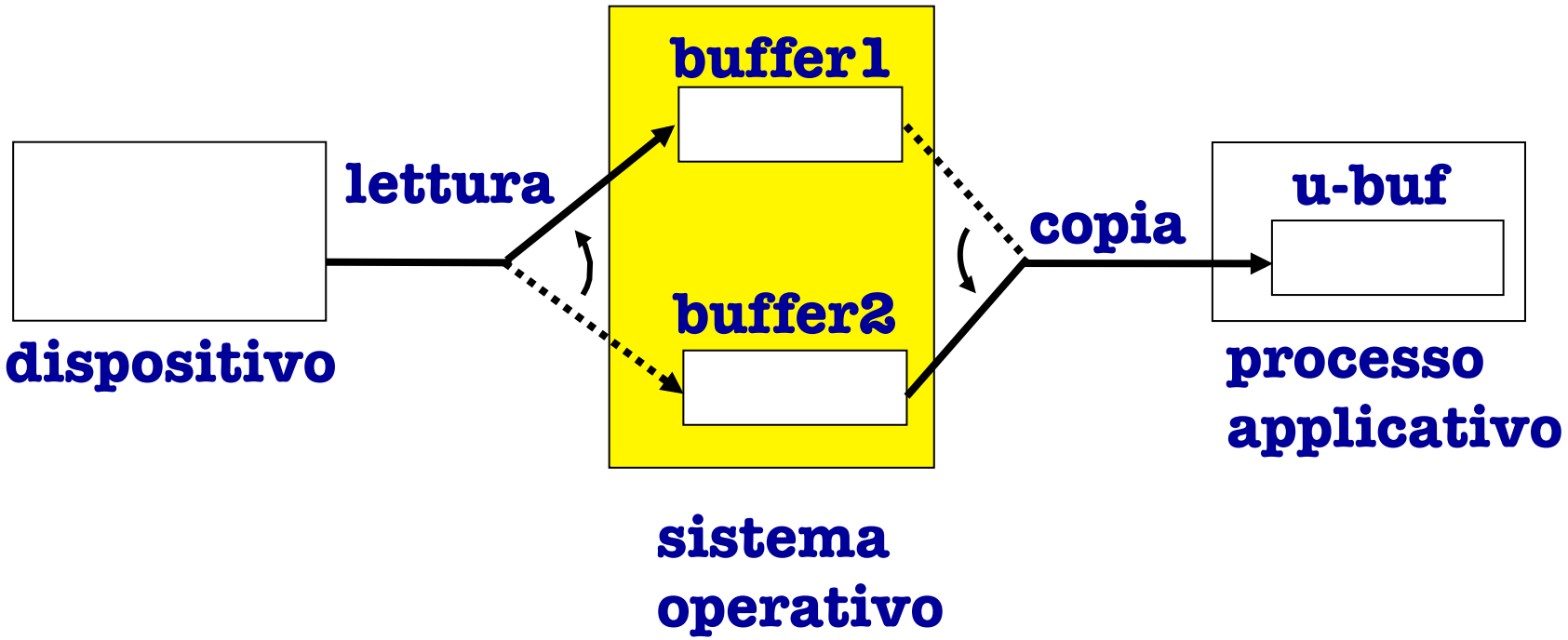
ES. operazione di lettura con singolo buffer



- **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 temporaneo o 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

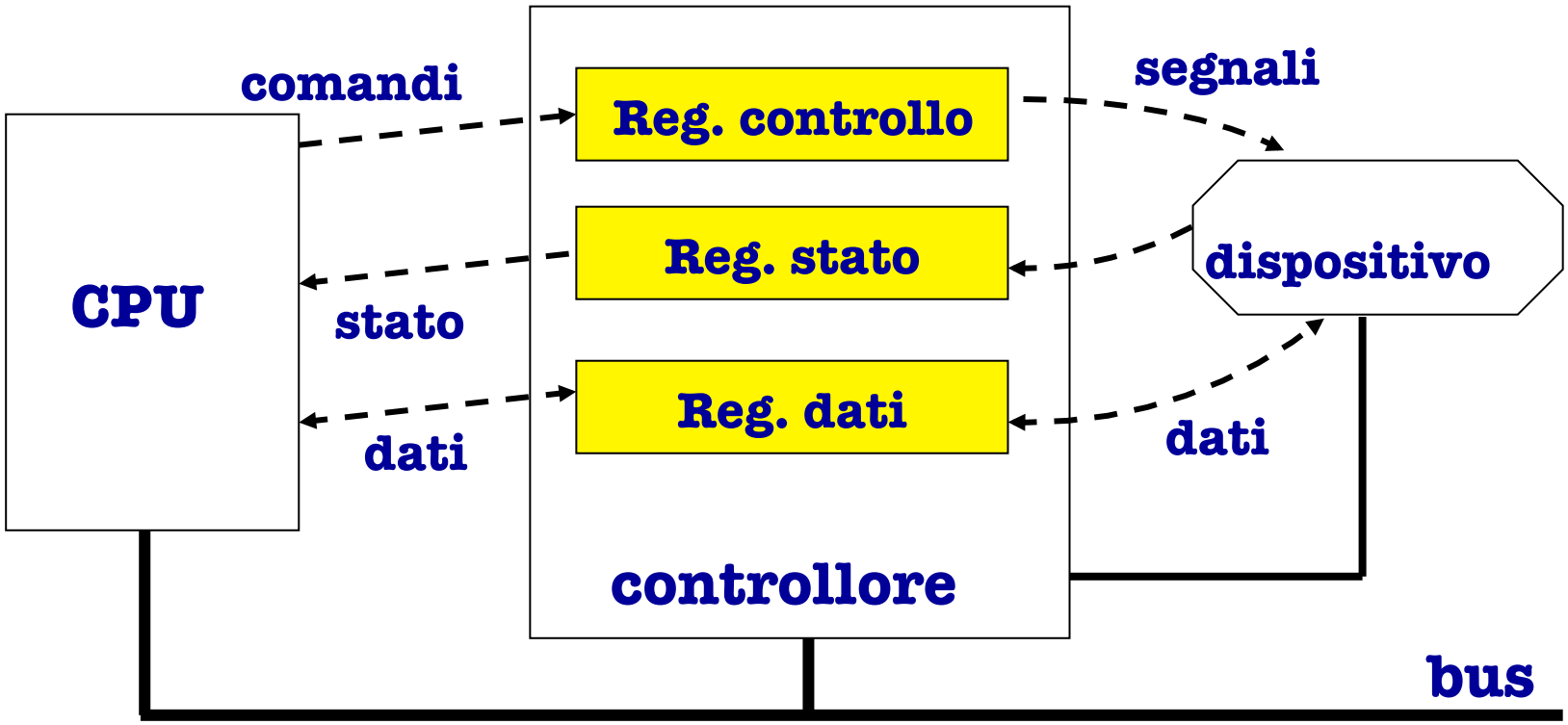
Funzioni:

- 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)

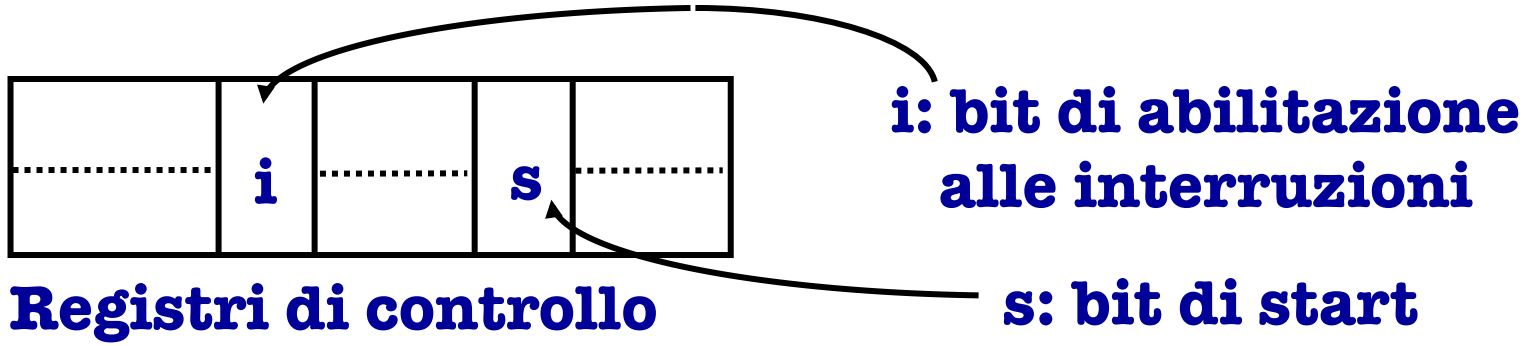
Interazione DISPOSITIVO-CPU

Schema semplificato di un controllore

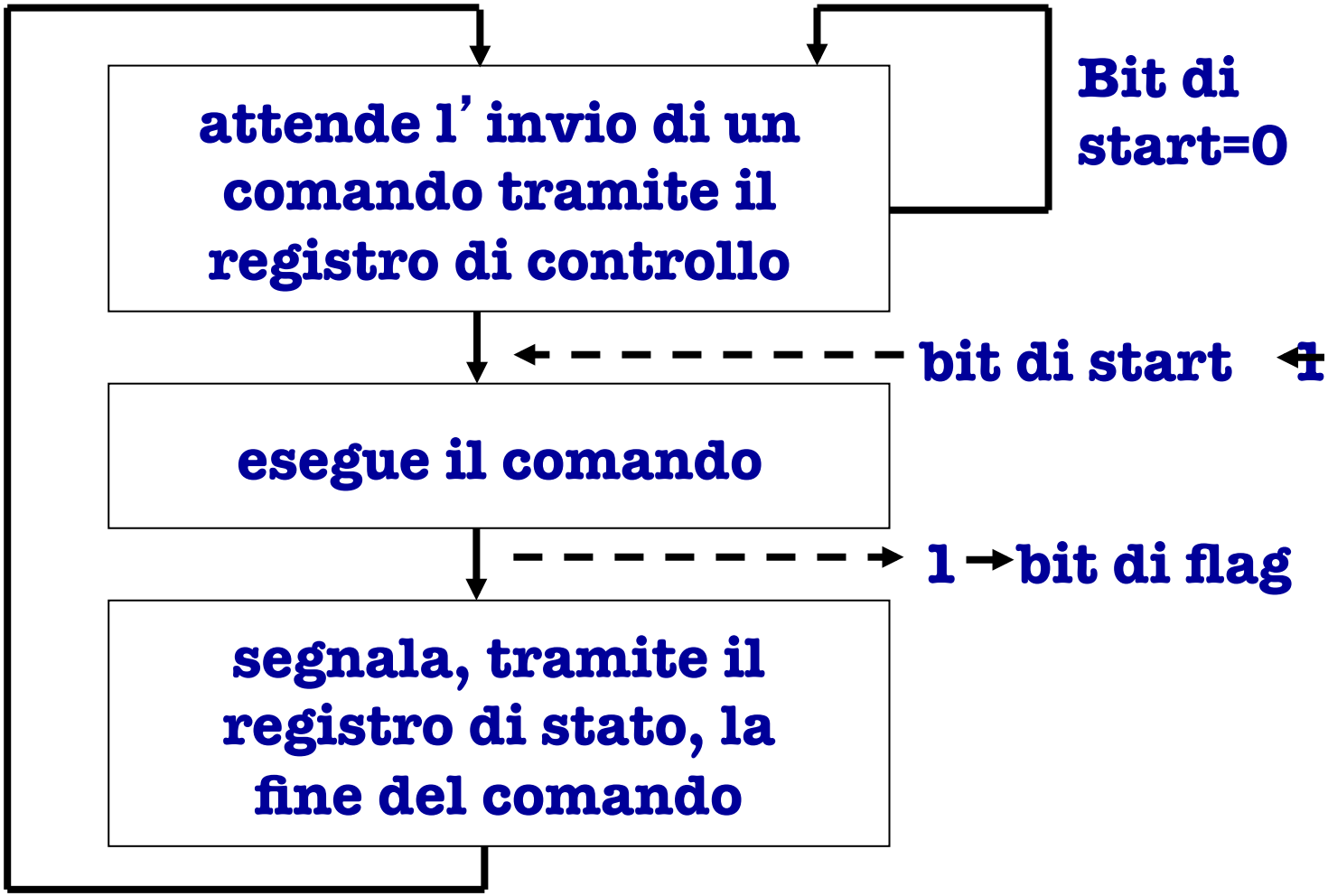


Controllore di UN DISPOSITIVO

Registri di stato e controllo



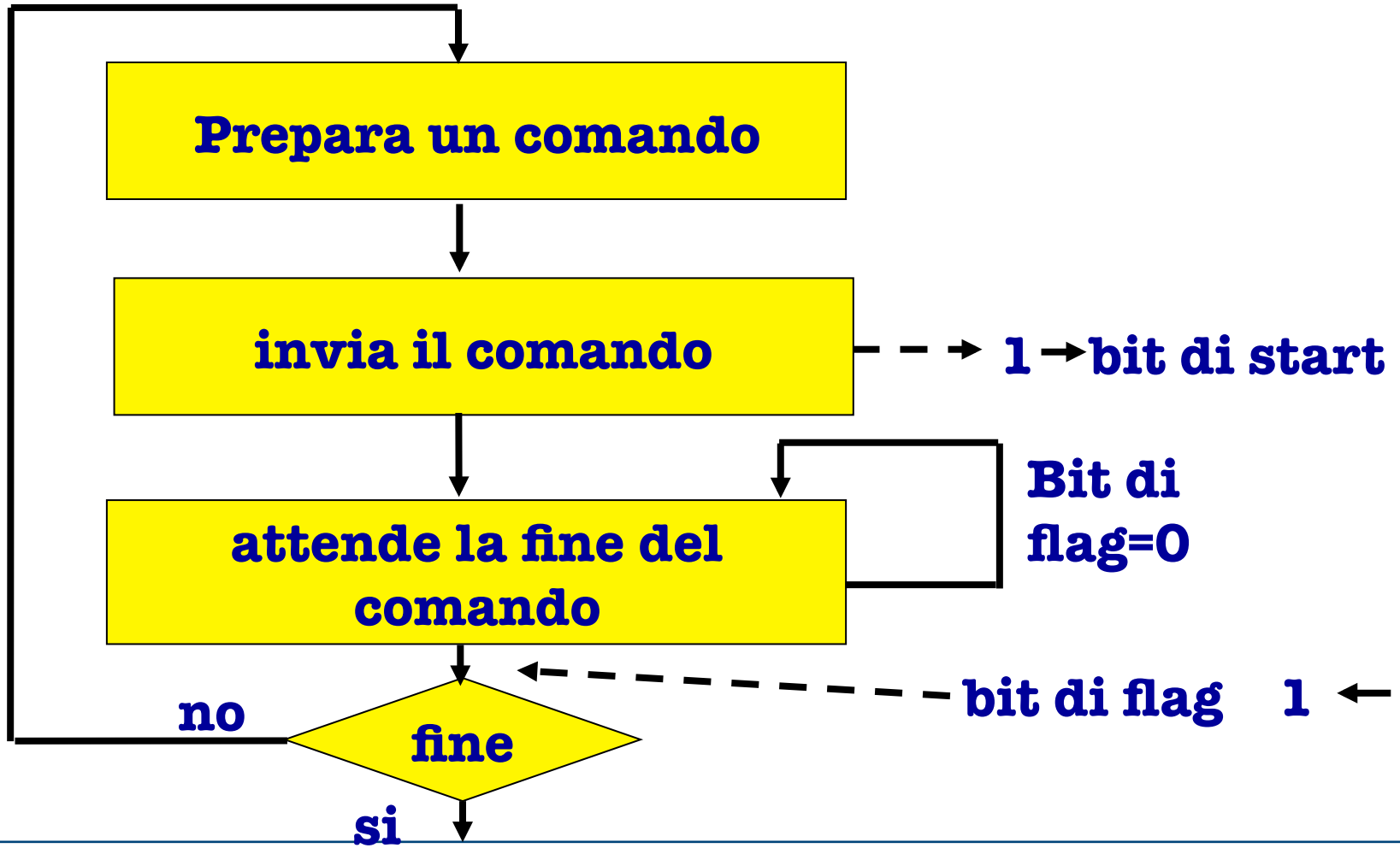
Gestione a controllo di programma
Attività del dispositivo: PROCESSO ESTERNO



Gestione a controllo di programma: PROCESSO ESTERNO

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

Gestione a controllo di programma: 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 "*a controllo di programma*" non è adatto per sistemi multiprogrammati, a causa dei cicli 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 ) ;
      ..................
}
```

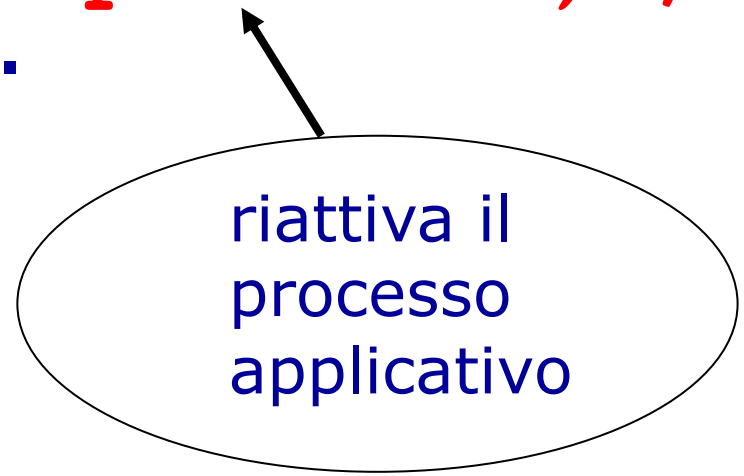
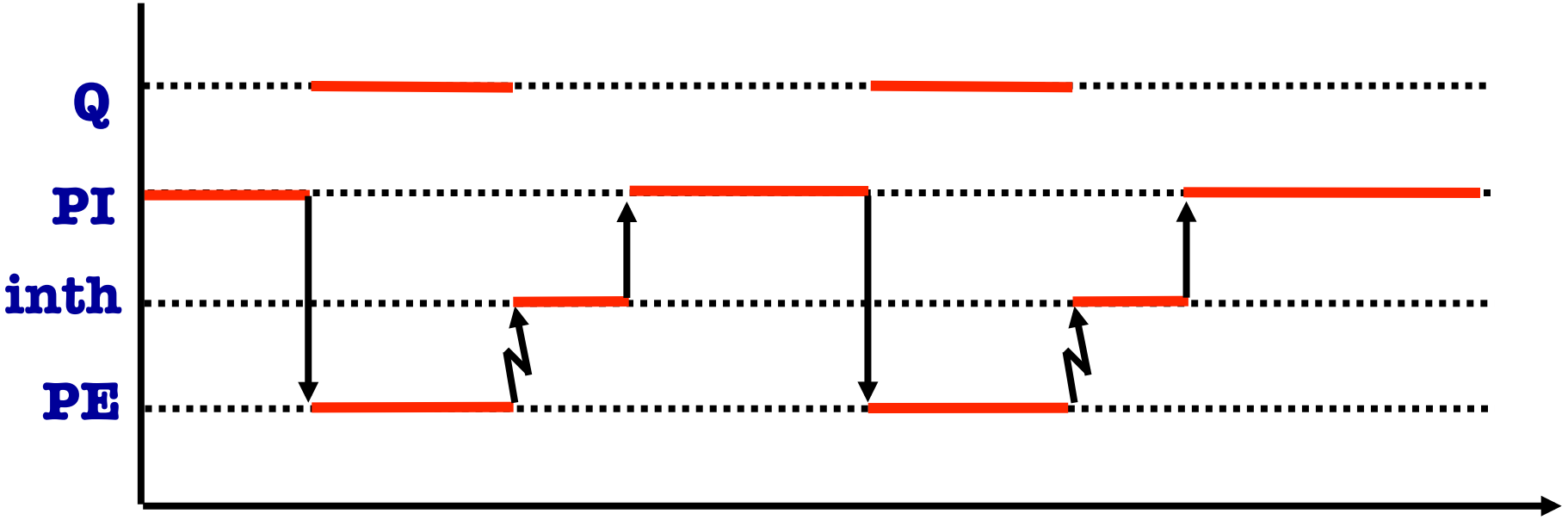


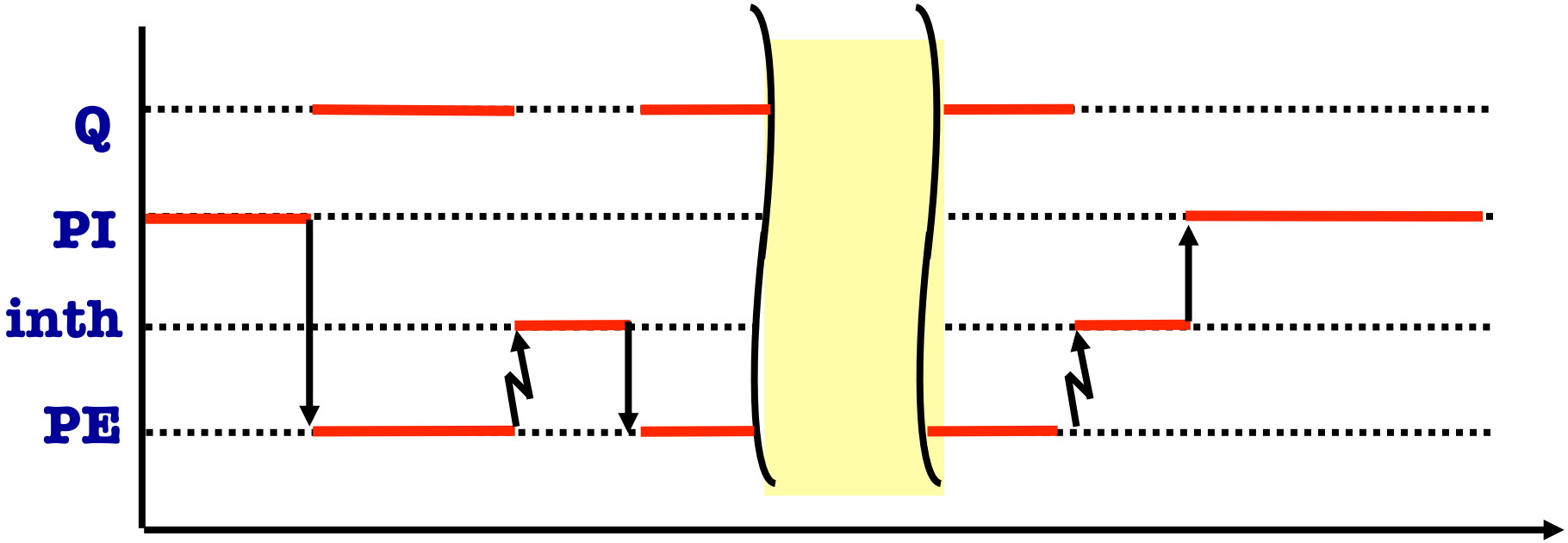
DIAGRAMMA TEMPORALE



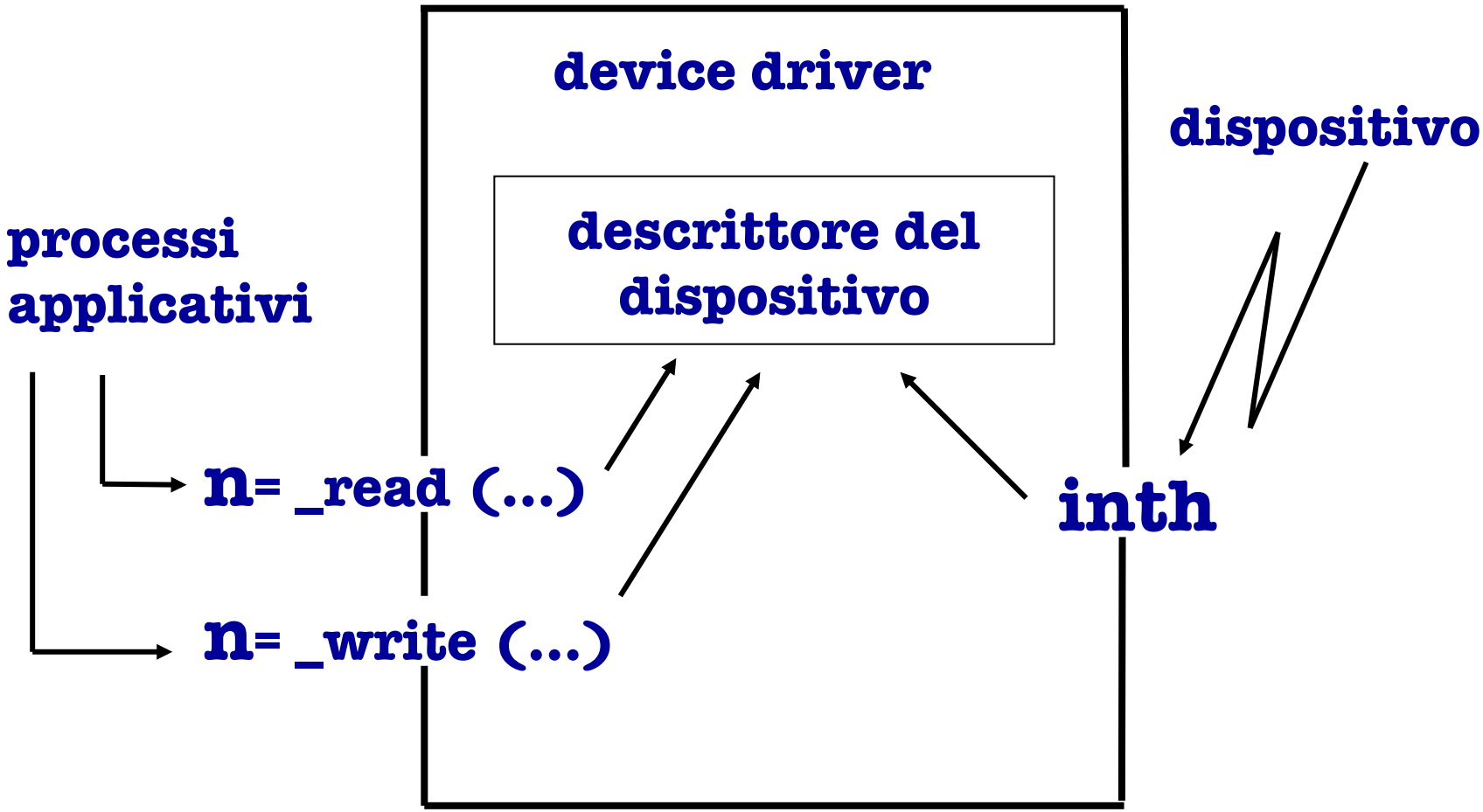
PI: processo applicativo che attiva il dispositivo
PE: processo esterno
Inth: routine di gestione interruzioni
Q: altro processo applicativo

DIAGRAMMA TEMPORALE

E` preferibile uno schema in cui il processo applicativo (PI) 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
 }
```

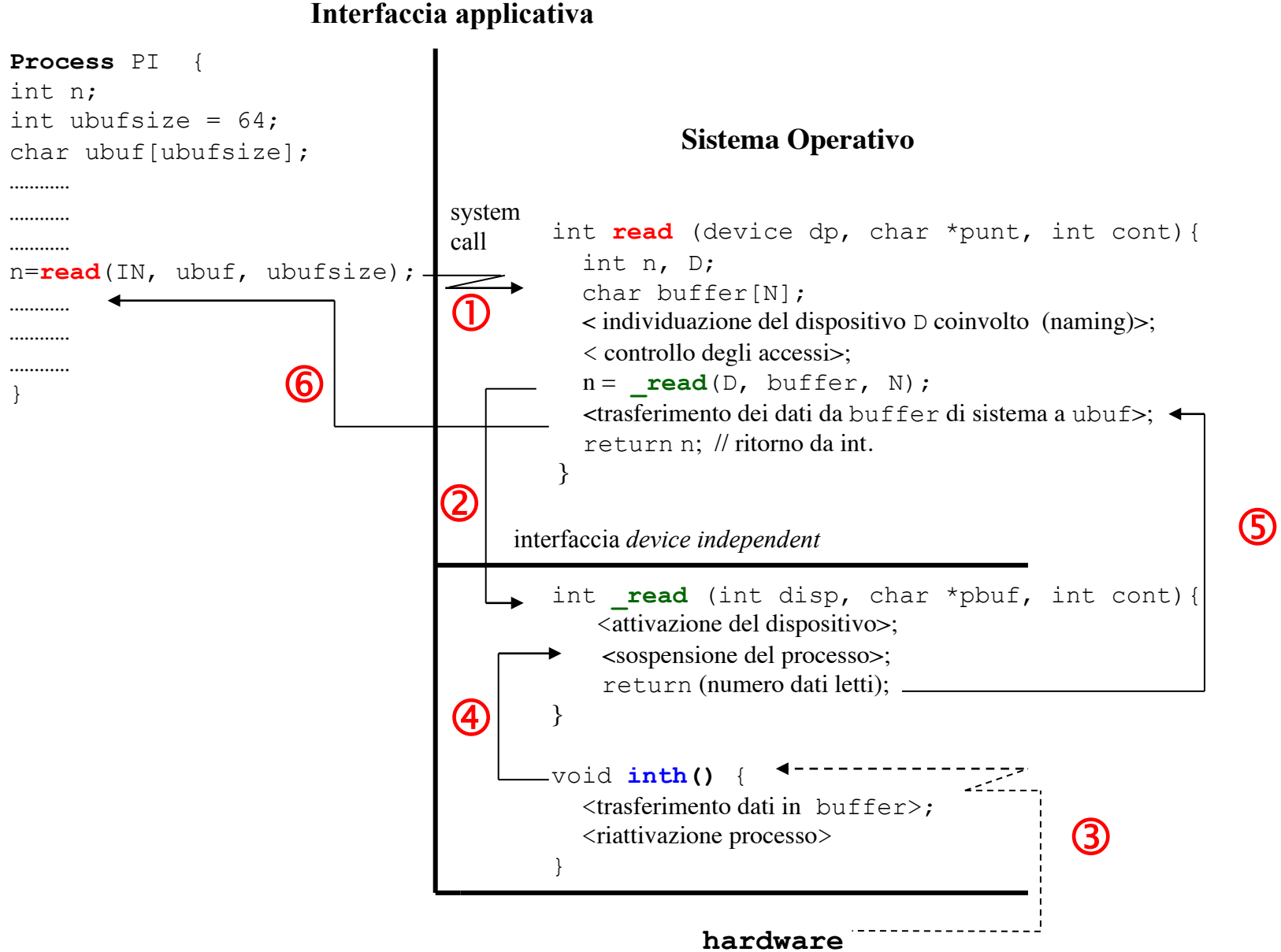
RAMO NORMALE DELLA FUNZIONE

```
{   < 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);
        }
}
```

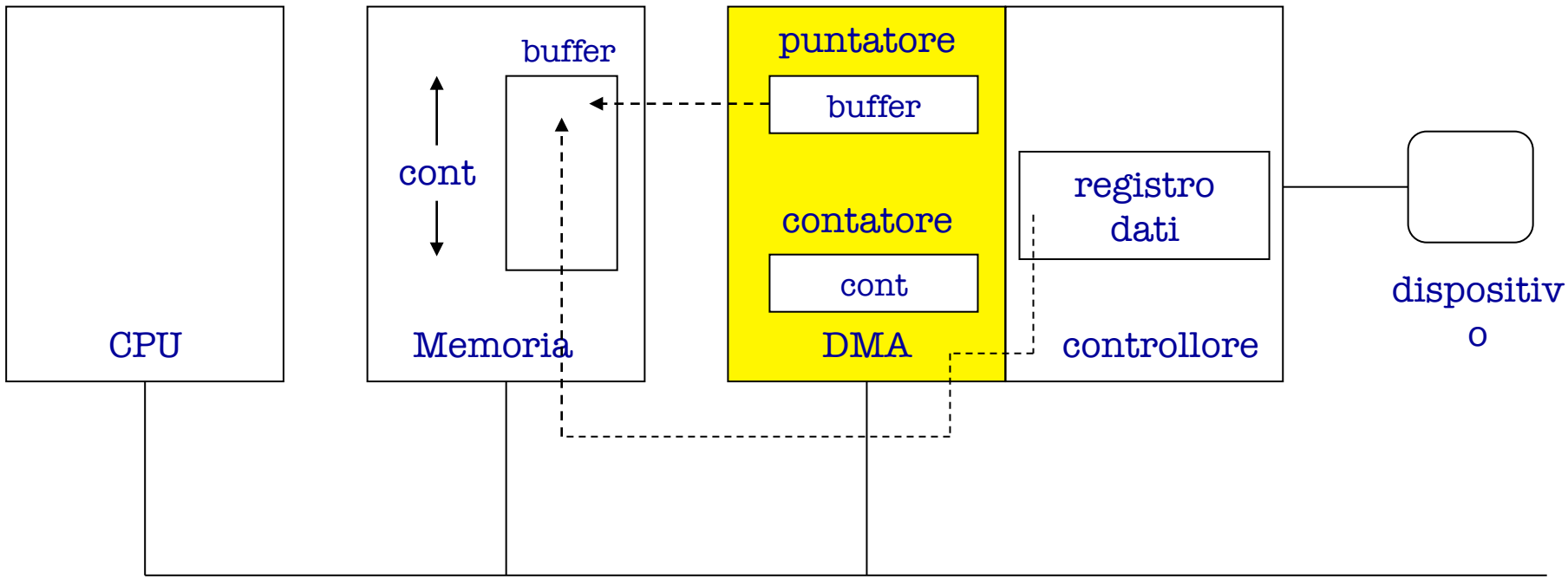
RAMO ECCEZIONALE DELLA FUNZIONE

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

Flusso di controllo durante un trasferimento



Gestione di un dispositivo in DMA



Gestione di dispositivi speciali : il temporizzatore

- Per consentire la modalità di servizio a divisione di tempo è necessario che il nucleo gestisca un *dispositivo temporizzatore* (timer o clock).
- 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)
- Esempio di operazione sul clock: impostazione di un timeout

```
delay(time);
```

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:

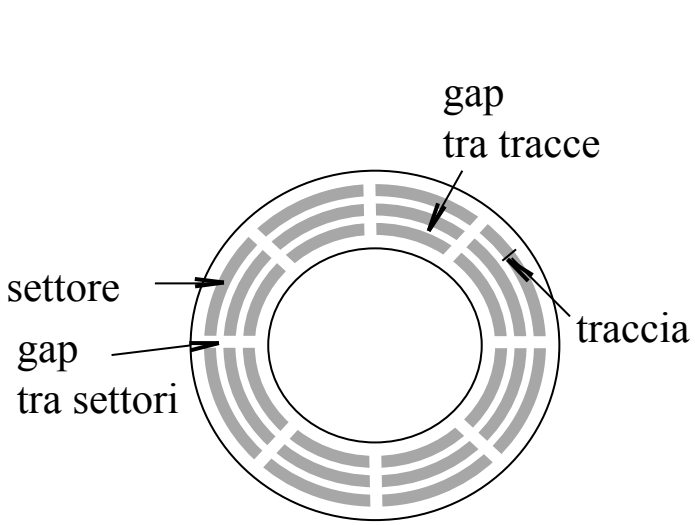
- un array di N semafori (fine_attesa[N]) inizializzati a zero. Ciascun semaforo viene utilizzato per bloccare il corrispondente processo che chiama la delay.
- un array di interi utilizzato per mantenere aggiornato il numero di quanti di tempo che devono ancora passare prima che un processo possa essere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

Descrittore del timer

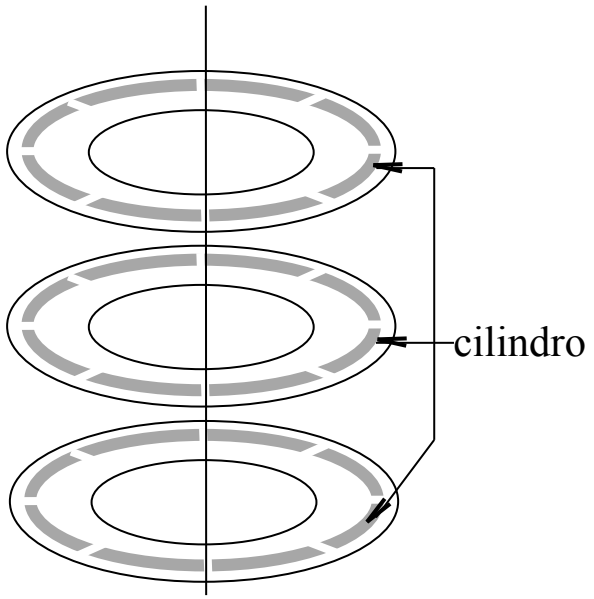
```
void delay (int n) {
        int proc;
        proc=<indice del processo in esecuzione>;
         descrittore.ritardo[proc]= n;
        //sospensione del processo:
        descrittore.fine_attesa[proc].p();
}



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

Gestione e organizzazione dei dischi

Organizzazione fisica



disco singolo



disk pack

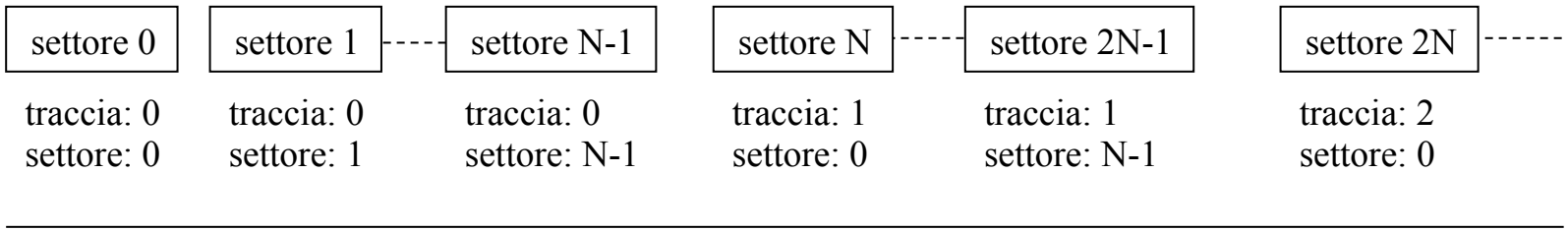
Indirizzo di un settore (blocco fisico):

(f,t,s)

dove:

f numero della ***faccia***, t numero della ***traccia*** nell'ambito della faccia, s numero del ***settore*** all'interno della faccia.

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



Indicando con

M il numero di tracce per faccia

N numero di settori per traccia

un blocco 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, ruotando, si posizioni 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, μs).

Quindi:

TF= ST + RL + TT

TA

Il tempo medio di trasferimento dipende sostanzialmente dal tempo medio di accesso (TA=ST + RL):

- RL è un parametro che dipende dalle caratteristiche fisiche del dispositivo
- ST dipende da come il disco viene gestito -> possibilità di gestione mirata alla riduzione del valore di ST

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.

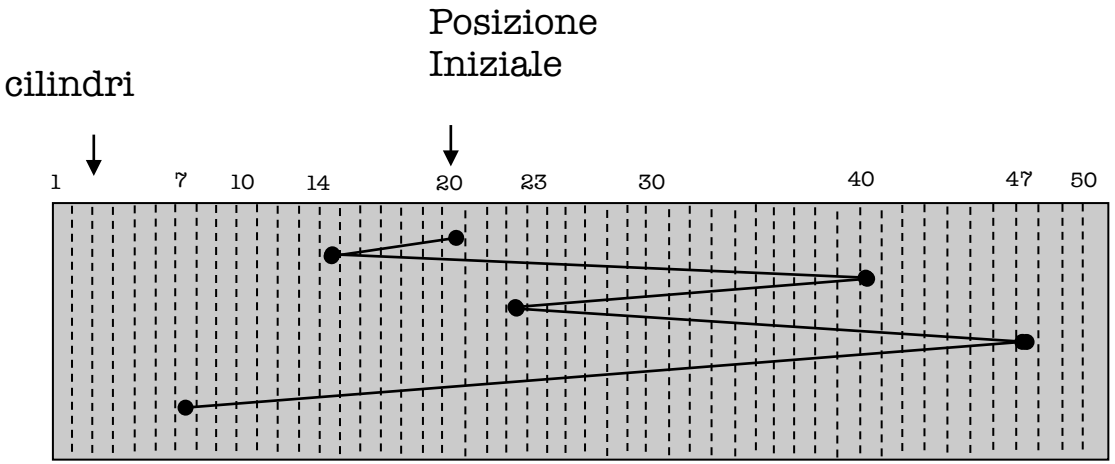
SSTF. Seleziona la richiesta con **tempo di seek minimo** a partire dalla posizione attuale della testina; può provocare situazioni di *starvation*

SCAN. La testina si porta ad una estremità del disco e si sposta verso l'altra estremità, servendo le richieste man mano che vengono raggiunte le tracce indicate, fino all'altra estremità del disco. Quindi viene invertita la direzione.
-> **Minimizzazione del ST medio**

CSCAN. E' una variante dello SCAN mirata a fornire un tempo di attesa medio (dei processi) più basso.

Algoritmo di scheduling FCFS

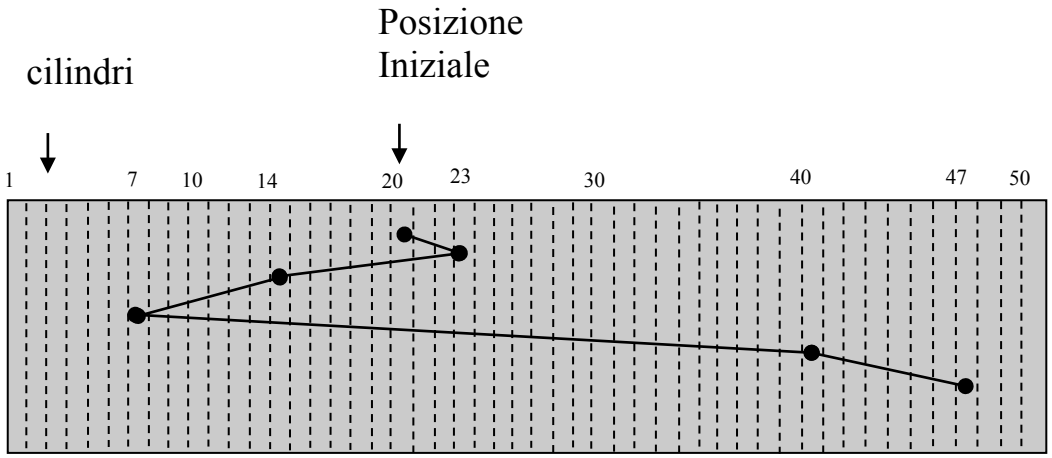
Testina posizionata sul cilindro 20. Richieste presenti in coda: [14, 40, 23, 47, 7]



Spostamento totale = 113cilindri

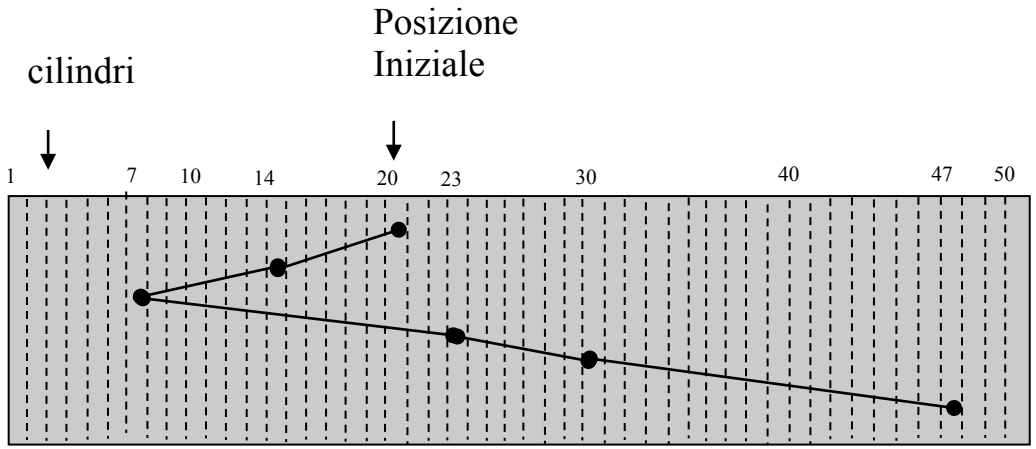
Algoritmo di Scheduling SSTF

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)

L'algoritmo SCAN non considera il tempo di attesa dei processi. Quando viene invertita la direzione per iniziare una nuova scansione, le richieste più "vecchie" riguardano tracce più lontane, che quindi verranno servite alla fine della scansione.

CSCAN è una variante dello SCAN, nella quale l'insieme delle tracce viene scansionato sempre nella stessa direzione (non c'è inversione di direzione): arrivata all'ultima traccia, la testina ritorna immediatamente all'inizio del disco per una nuova scansione a partire dalla prima traccia. Rispetto allo SCAN, fornisce un tempo di attesa medio più basso.