# Gestione del processore

## Il modello a processi sequenziali

# Il modello a processi sequenziali

- Nei computer attuali, ci sono molte attività attive contemporaneamente (sia di SO che di utente)
  - es : stampa in corso + Word + cd player …
- Programmare un insieme di attività che interagiscono in maniera del tutto incontrollata ed arbitraria è un compito proibitivo
  - es : scrivere un programma che tenga conto della possibilità di venire interrotto da altri dopo l'esecuzione di ogni comando….

# Il modello a processi sequenziali (2)

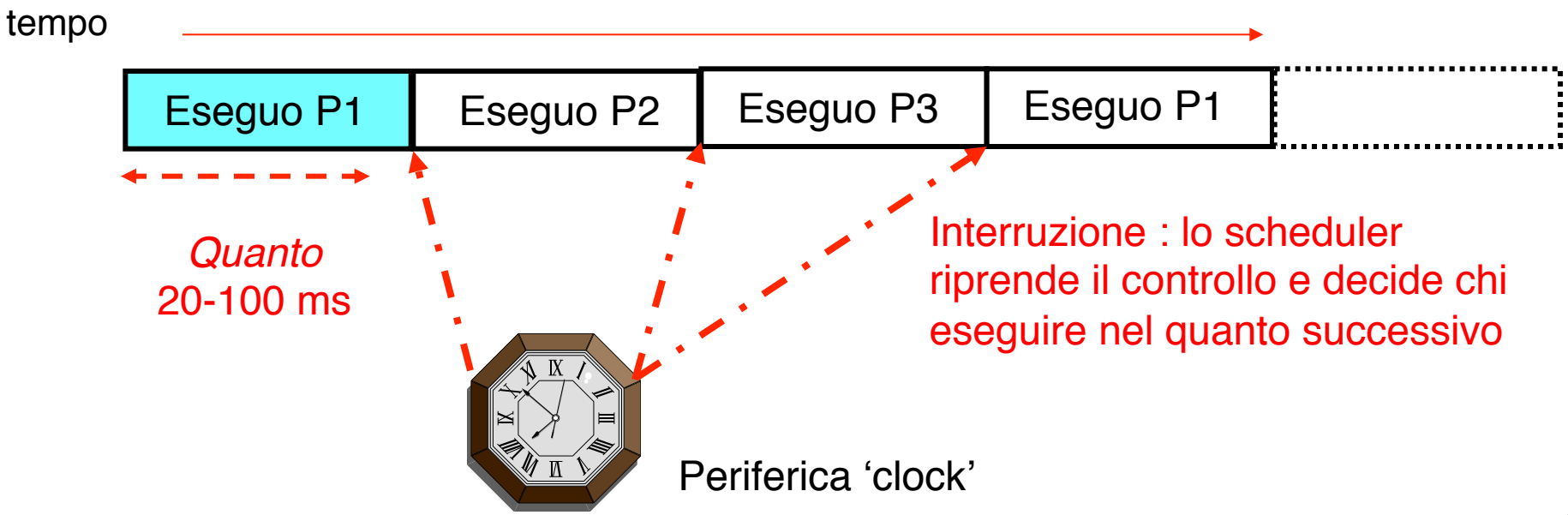
- Il modello a processi sequenziali : è un modello concettuale che permette di *esprimere in modo semplice più attività contemporanee*
  - ogni attività è incapsulata in un processo e viene programmata come se utilizzasse il calcolatore in modo esclusivo
  - l'interazione fra processi avviene solo con l'invocazione di un numero (limitato e fissato) di meccanismi di cooperazione (di IPC)

# Il modello a processi sequenziali (3)

- Con questo modello tutto il software che gira su un calcolatore è organizzato come un insieme di processi sequenziali cooperanti
  - es : un comando richiesto alla shell provoca l'attivazione di un processo che esegue il comando
  - es : `ps` voi fornisce informazioni sui processi attivi in un sistema Unix/Linux, sono decine!

# Lo scheduler

- Lo scheduler (di basso livello) si preoccupa di ripartire il processore fra i processi in maniera trasparente
  - es : 3 processi P1, P2, P3, vengono mandati in esecuzione ciclicamente



# Cos'è un processo ?

- *Un processo* è un programma in esecuzione completo del suo *stato* (spazio di indirizzamento, registri, file aperti…)
- Come si crea un processo ?
  - Richiedendo al SO l'esecuzione di una o più chiamate di sistema
    - es : `fork - exec` in Unix
    - es : CreateProcess in API Win32
- Esempi di attivazione :
  - l'invocazione di un comando, il doppio click su un'icona, il lancio di un eseguibile …
  - inizializzazione del sistema

# Terminazione di un processo

- Un processo termina :
  - Quando esegue una istruzione assembler di terminazione
  - Quando effettua una operazione illecita
    - es. cerca di accedere alla memoria privata di altri processi
  - Quando c'è un errore che non gli permette di proseguire (es. overflow, etc)
- In tutti questi casi il processore ricomincia automaticamente ad eseguire il sistema operativo ad un indirizzo prefissato

# Interruzione momentanea di un processo

- Un processo in esecuzione può cedere il processore al SO nei due casi seguenti:
  - quando il processo stesso esegue una chiamata di sistema per richiedere un servizio da parte del SO
  - quando arriva una interruzione hw

Esaminiamo in dettaglio

i due casi

# Gestione del processore

Come un processo utente ‘cede’ il controllo al SO:

*System call e interruzioni*

# Cosa sono le system call?

- Sono funzioni che permettono di invocare servizi di competenza del SO
  - es. scrittura di file, stampa su video …
- Sono disponibili in una libreria predefinita (tipicamente in C)
- Vengono invocate come normali funzioni
  - es. invocazione della chiamata di sistema read fornita da Unix/Linux per leggere i file

```
read (fd, buffer, nbytes);
```

# Cosa sono le system call? (2)

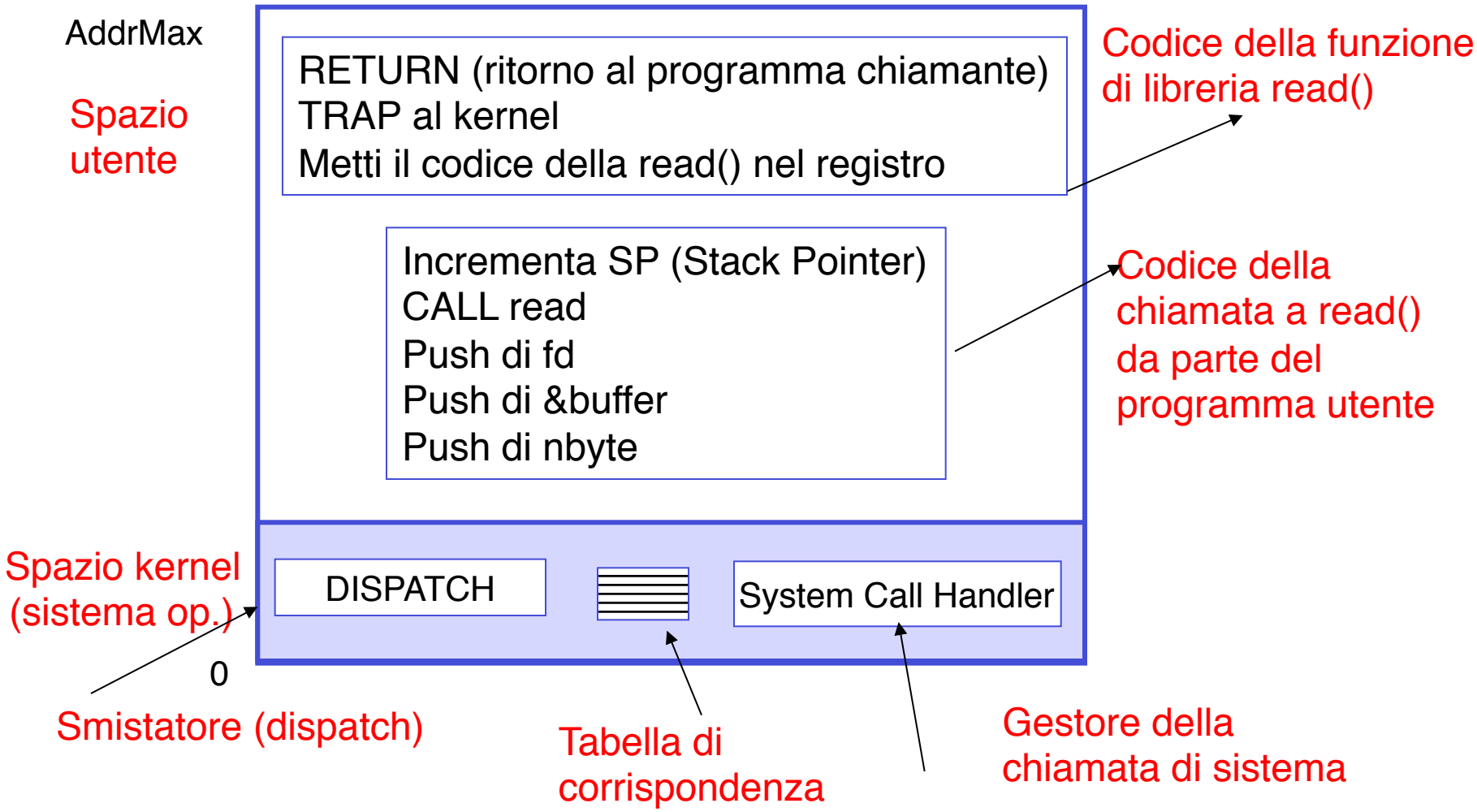
- La implementazione delle SC è però diversa da quella delle normali funzioni
  - sono scritte direttamente in assembler e utilizzano la istruzione TRAP
  - la TRAP permette di
    - passare da stato utente a stato supervisore/kernel
    - saltare ad un indirizzo predefinto all'interno del sistema operativo
  - il processo rimane in esecuzione (con accesso al suo spazio di indirizzamento) solo esegue una funzione del sistema operativo in stato kernel

# Cosa sono le system call? (3)

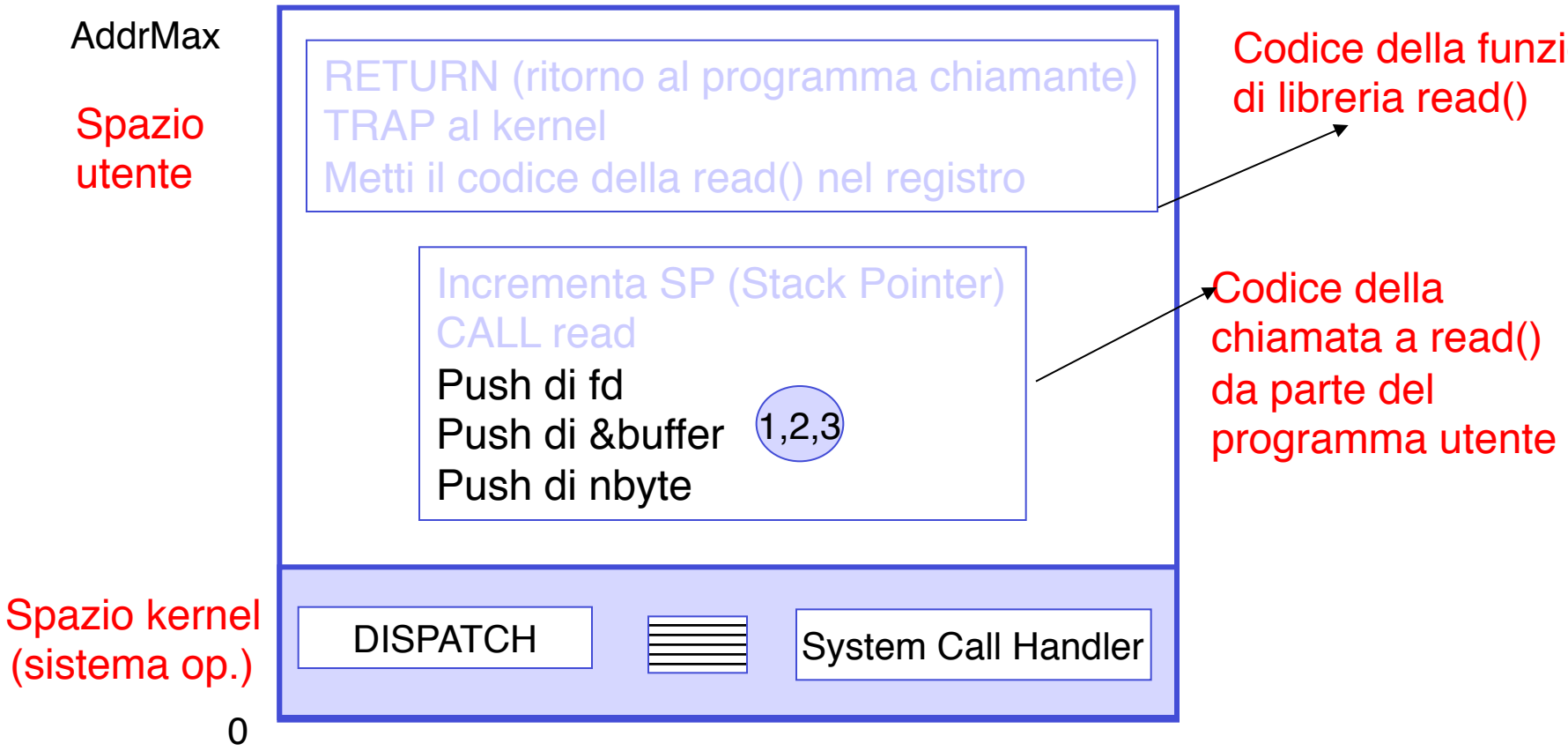
- Ma esattamente cosa accade quando viene invocata una system call ?
  - Vediamo in dettaglio come funziona una chiamata a

    read (fd, buffer, nbytes);

# Chiamata a read(fd, buffer, nbytes)

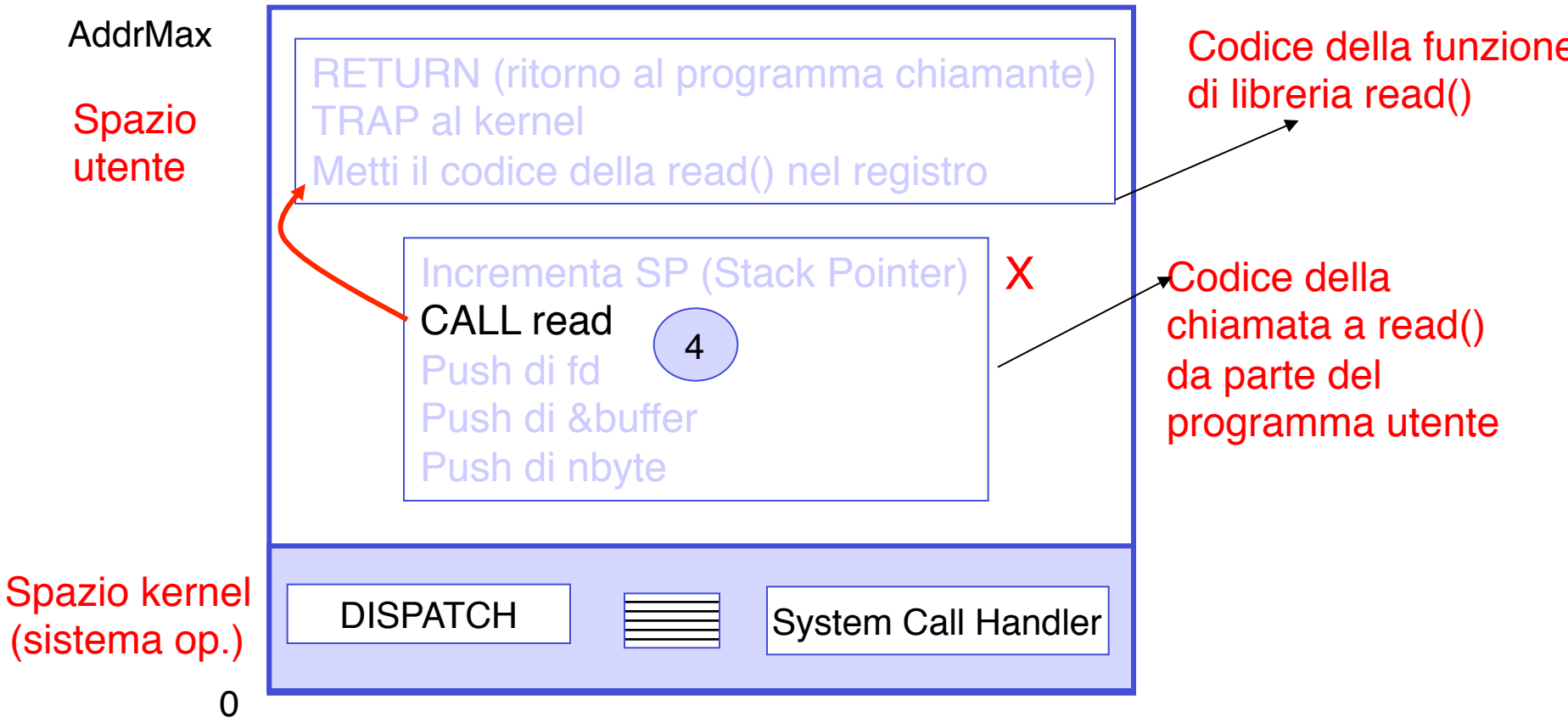


# read(fd, buffer, nbytes) (2)



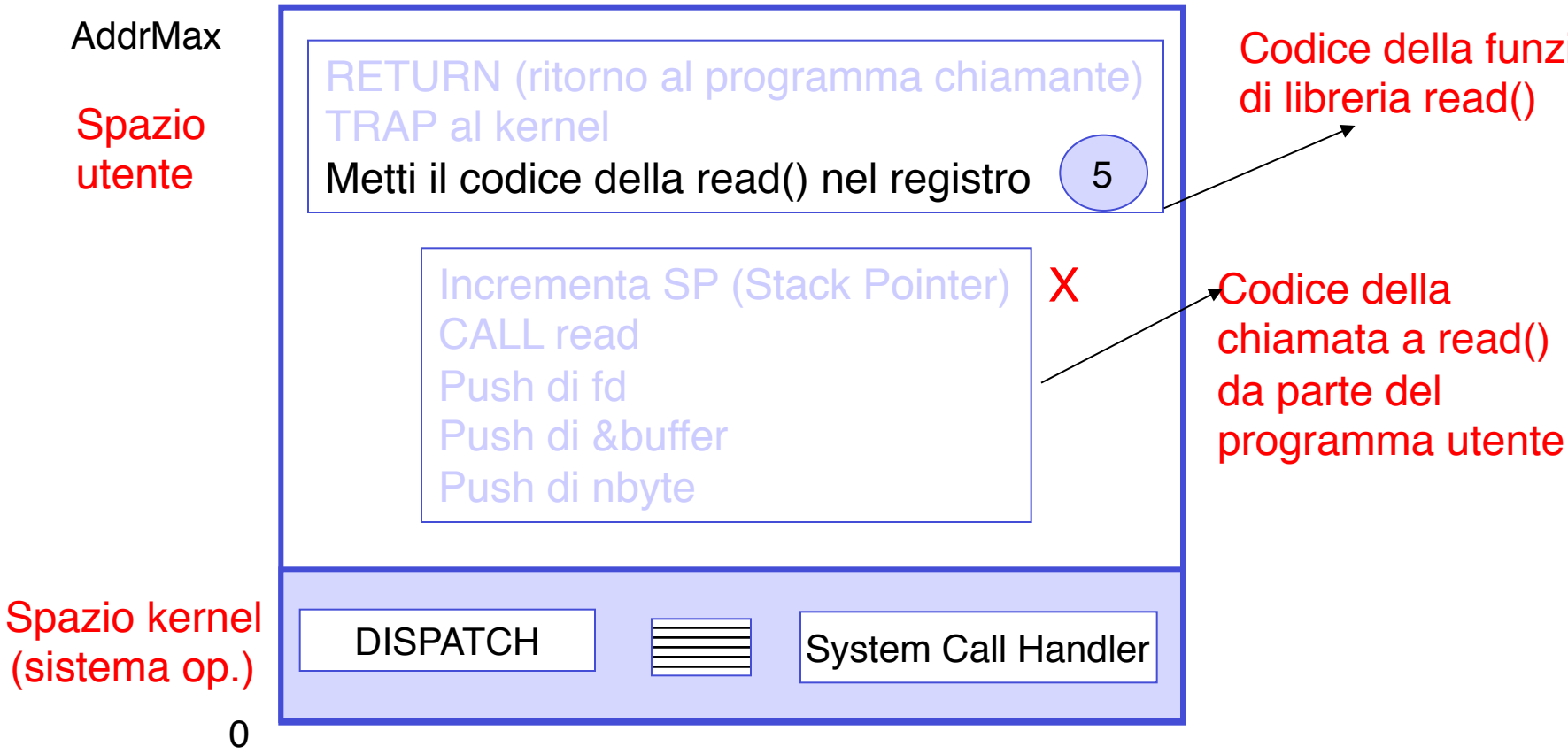

- Passi 1,2,3 : si ricopia il valore dei parametri sullo stack

# read(fd, buffer, nbytes) (3)



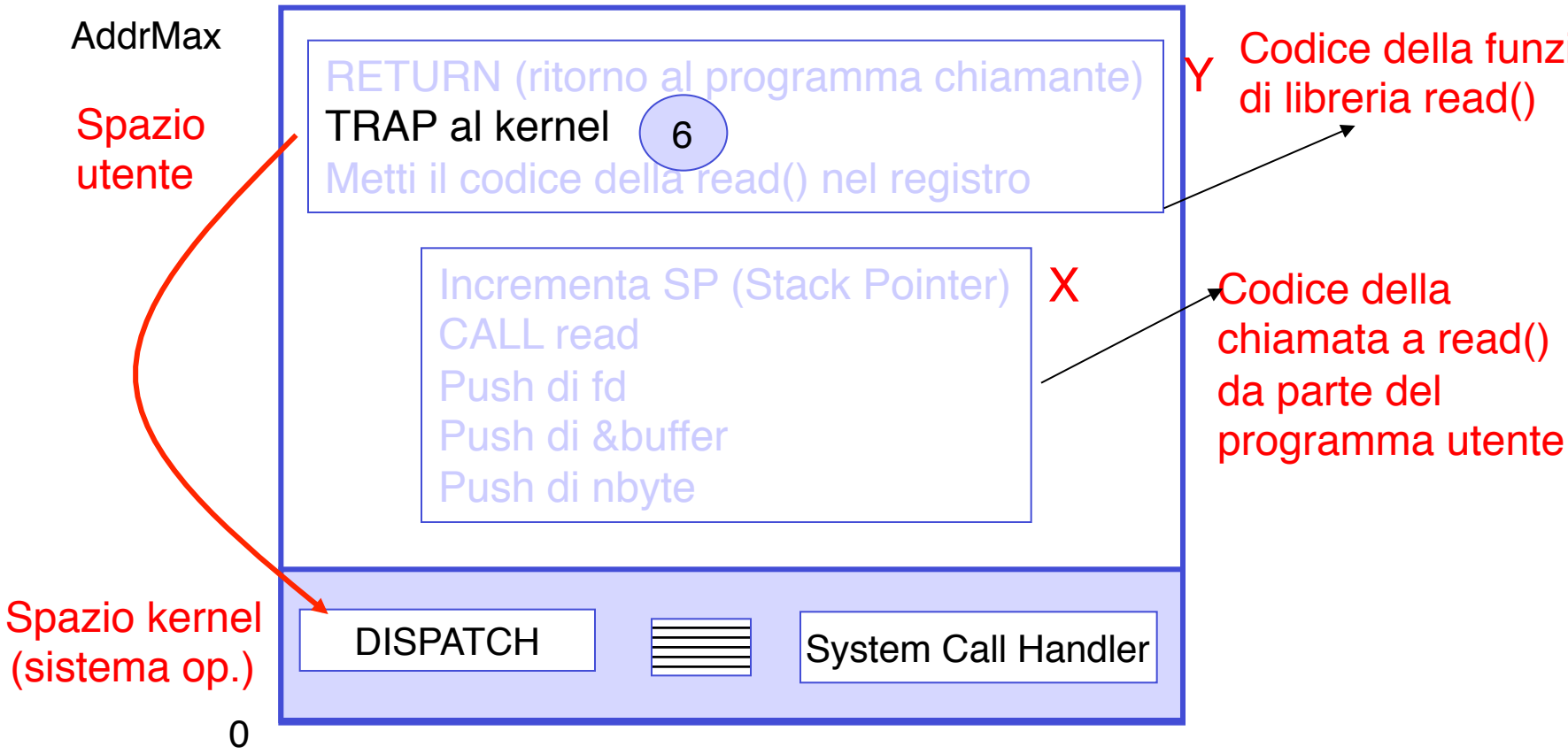

- Passo 4 : chiamata di read()
  - salto alla prima istruzione di read() + push indirizzo di ritorno (X) sullo stack

# read(fd, buffer, nbytes) (4)



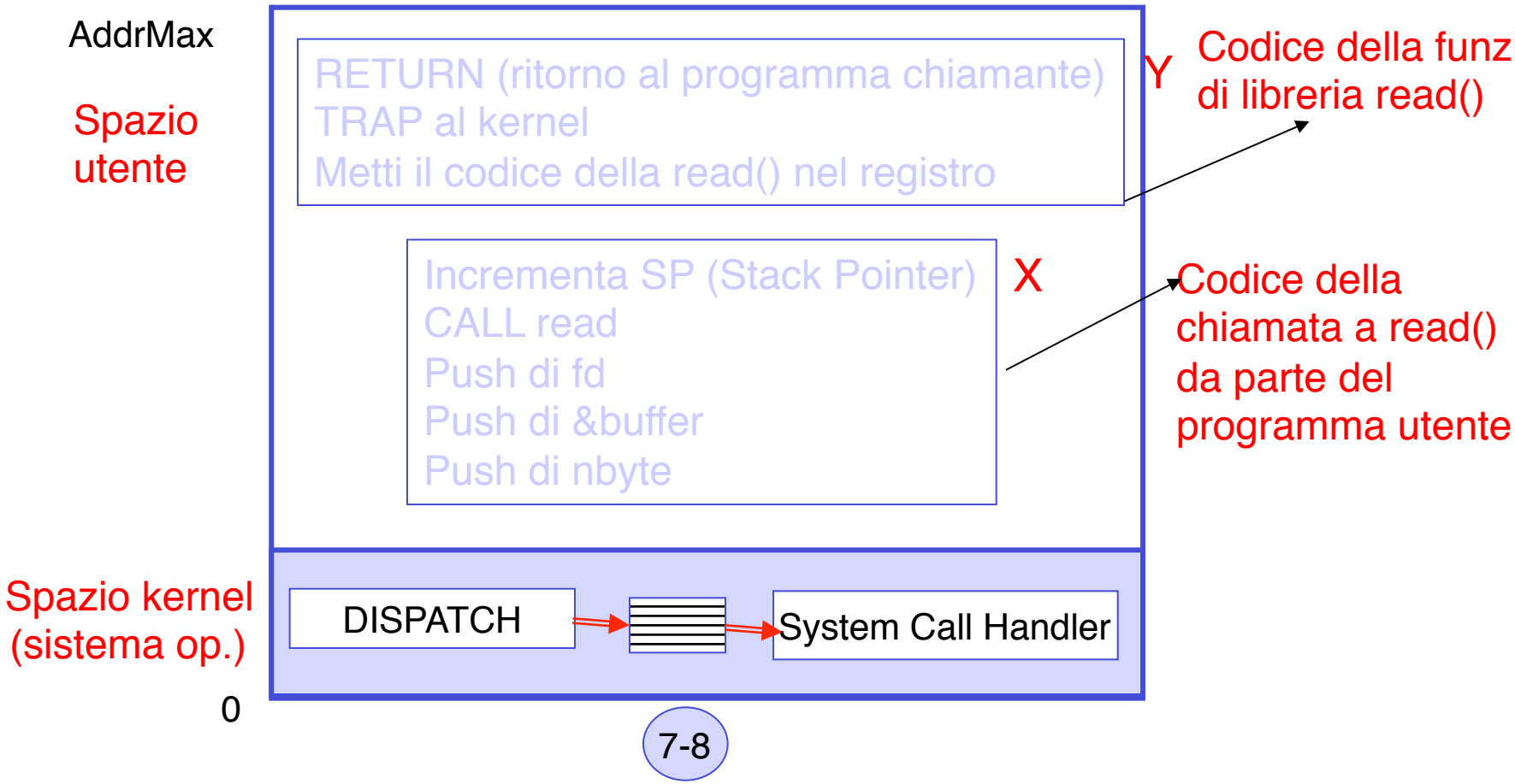

- Passo 5 : Inizia l'esecuzione della read() :
  - caricamento del codice della system call in un registro fissato Rx

# read(fd, buffer, nbytes) (5)



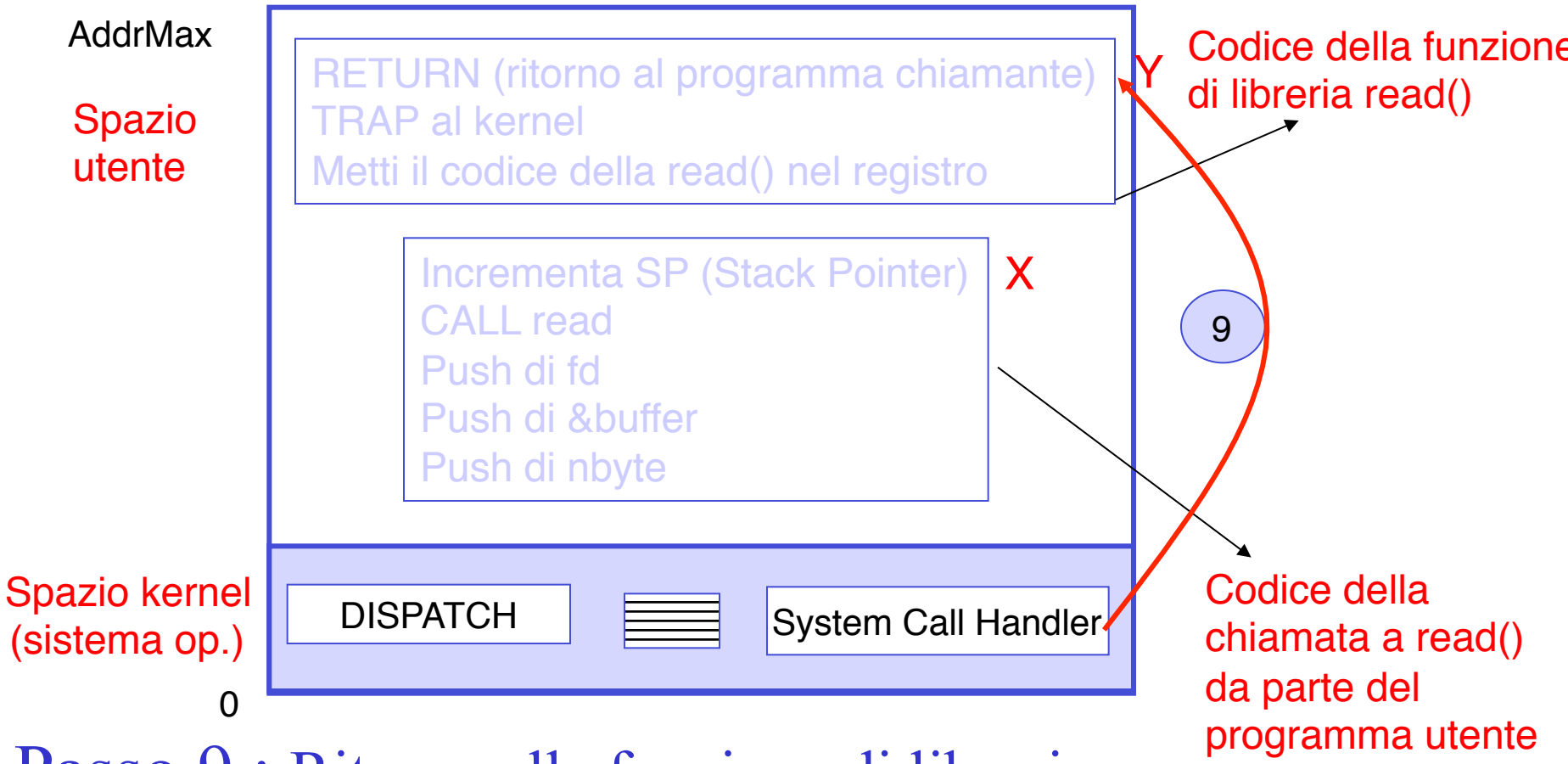

- Passo 6 : Esecuzione TRAP
  - passaggio in kernel mode, salto al codice dello smistatore

# read(fd, buffer, nbytes) (6)



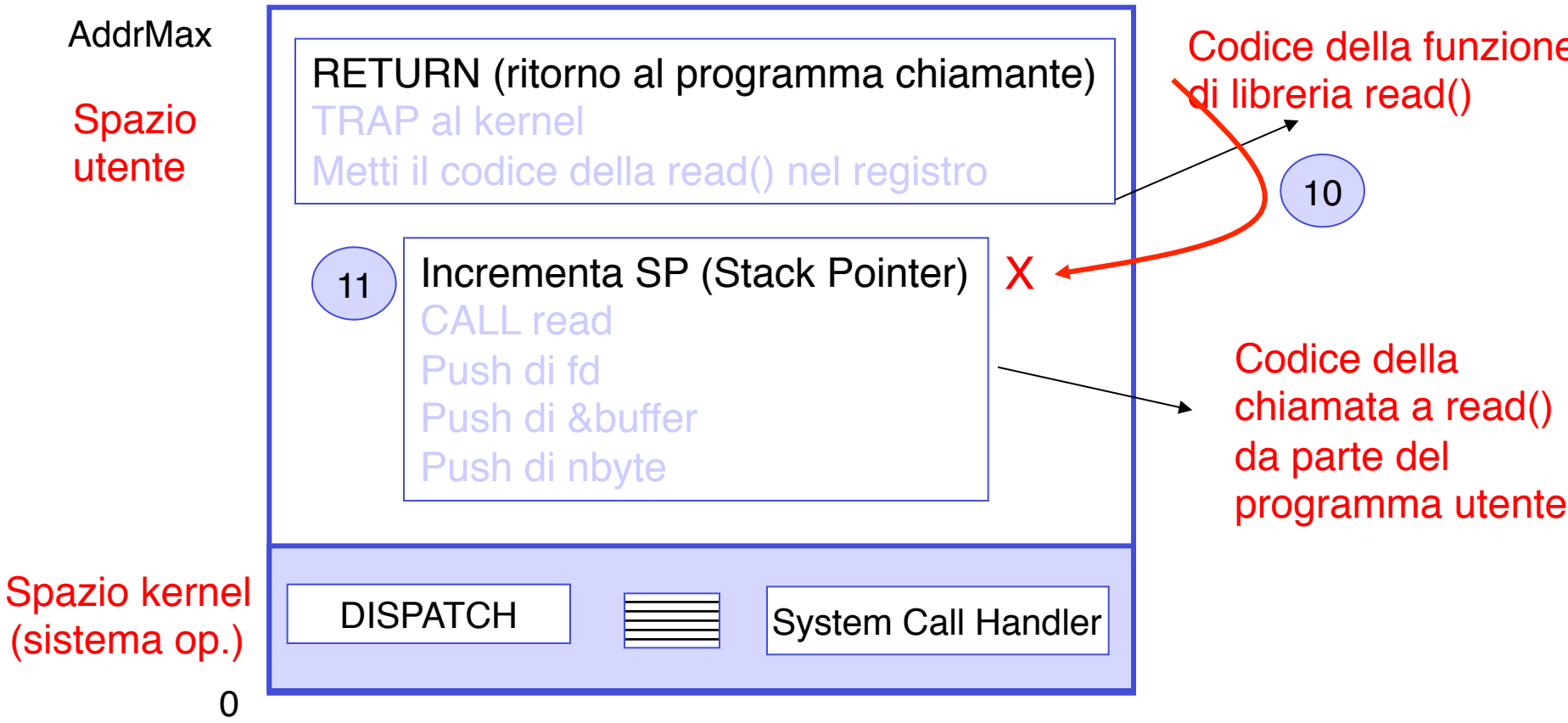

- Passi 7-8 : Selezione della SC secondo il codice in Rx

# read(fd, buffer, nbytes) (7)




- Passo 9 : Ritorno alla funzione di libreria
  - ripristino user mode, caricamento PC con l'indirizzo dell'istruzione successiva alla TRAP (Y)

# read(fd, buffer, nbytes) (8)




- Passi 10-11 : Ritorno al codice utente (nel modo usuale)
  - PC = X, SP viene incrementato per eliminare il frame della read()

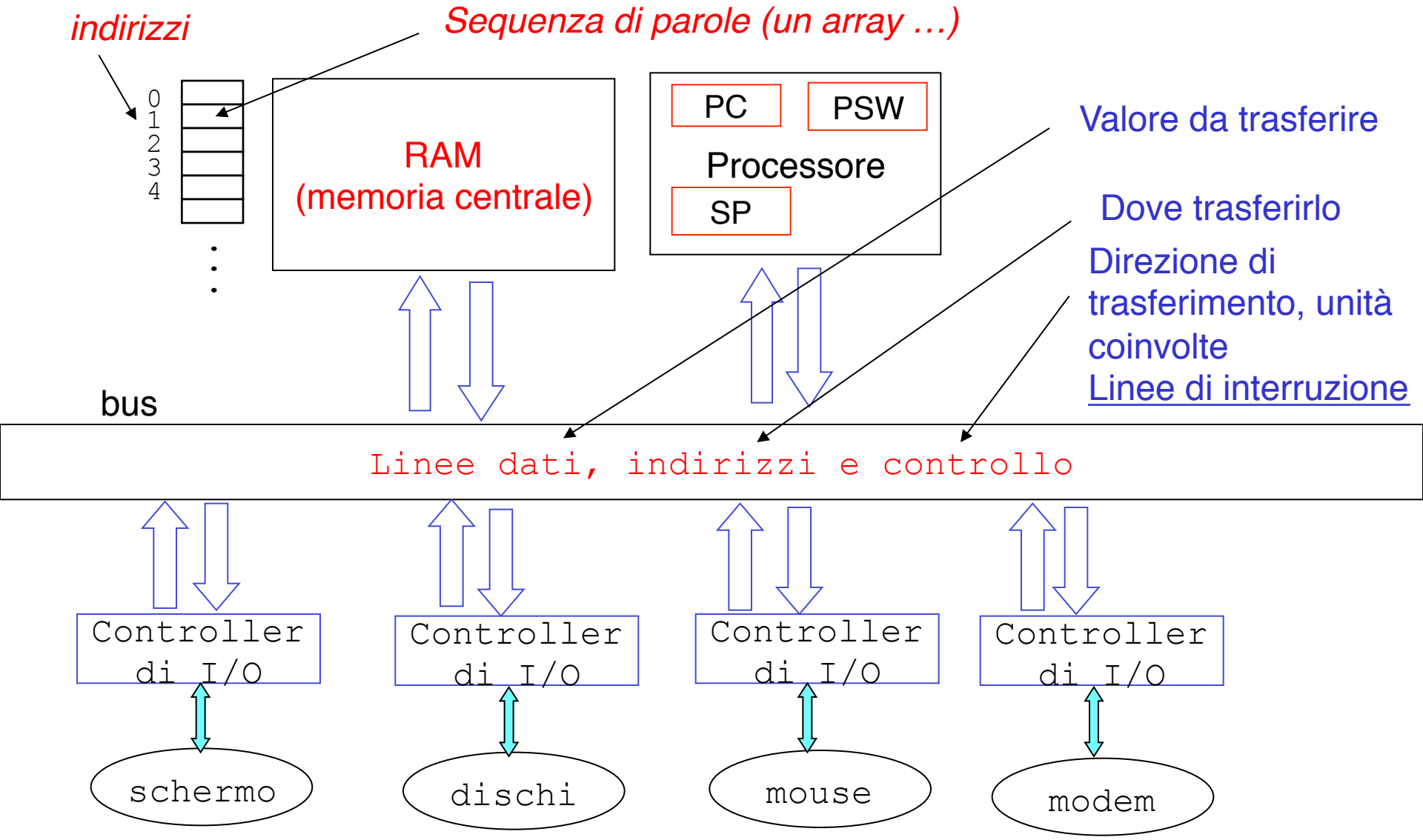
# Gestione del Processore

*Interruzioni*

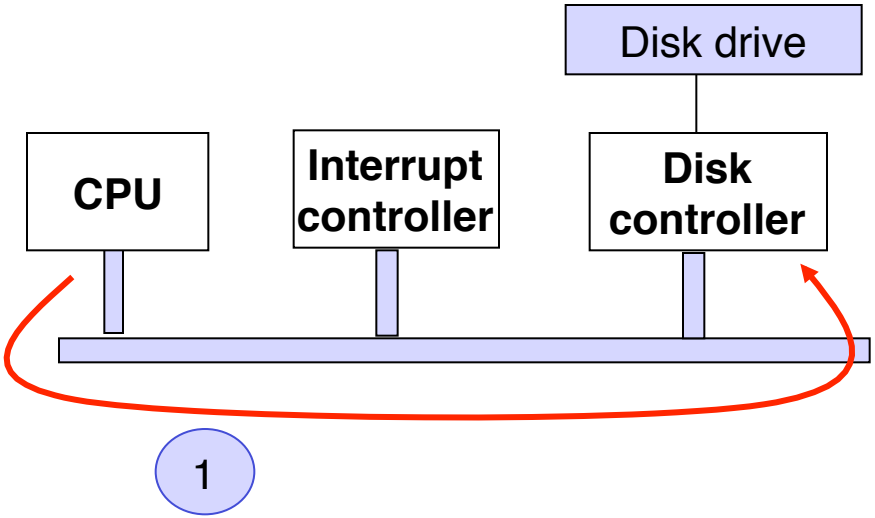
# Interruzione di un processo

- Come funzionano le *interruzioni hw*
  - ogni periferica può 'richiedere attenzione' inviando un segnale di *interruzione* usando le linee di controllo del bus
  - alla fine dell'esecuzione di ogni istruzione assembler il processore controlla la presenza di una interruzione
  - se l'interruzione è presente il controllo passa automaticamente al sistema operativo
    - vediamo più in dettaglio …..

# Struttura di un semplice PC

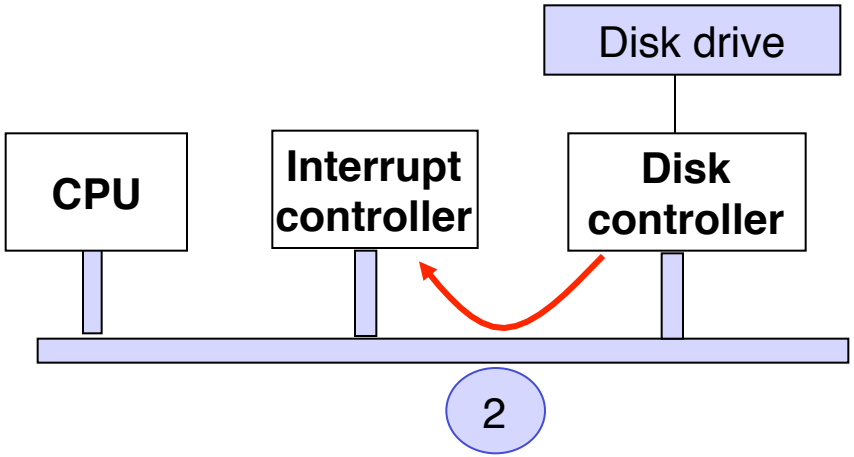


# Interruzioni (1)




Esempio : La CPU richiede una operazione al controllore del disco

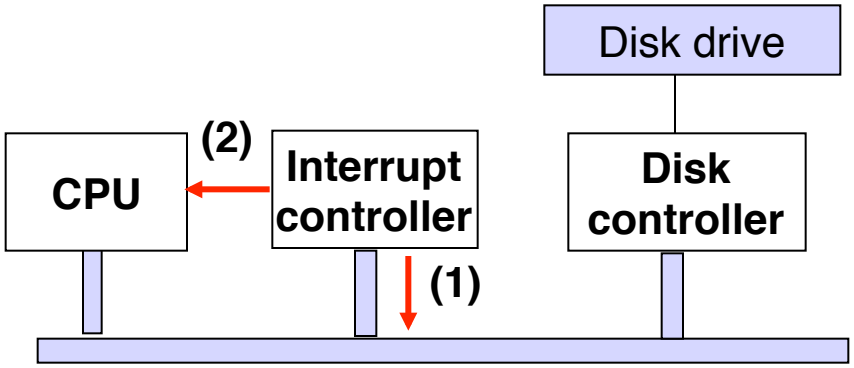
# Interruzioni (2)




Passo 2 : Quando il controllore ha finito :

- causa un'interruzione asserendo una linea del bus che gli è stata assegnata

# Interruzioni (3)

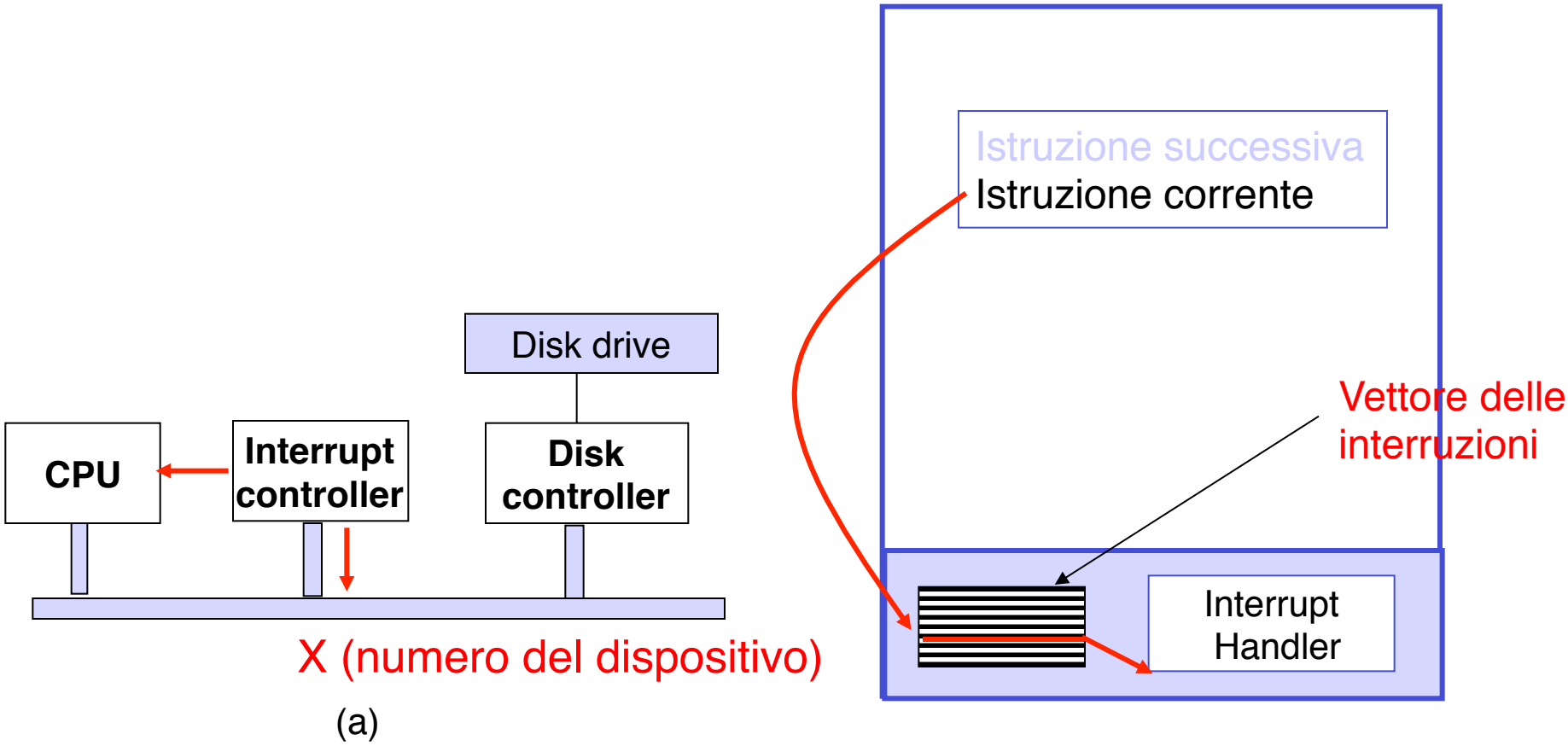



X (numero del dispositivo)

Passo 3 : Se IC può gestire l'interruzione :

- imposta il segnale di interruzione per la CPU (2)
- imposta sulle linee indirizzo del bus, un numero (X) che specifica il dispositivo di cui sta servendo l'interruzione (1)

# Interruzioni (4)



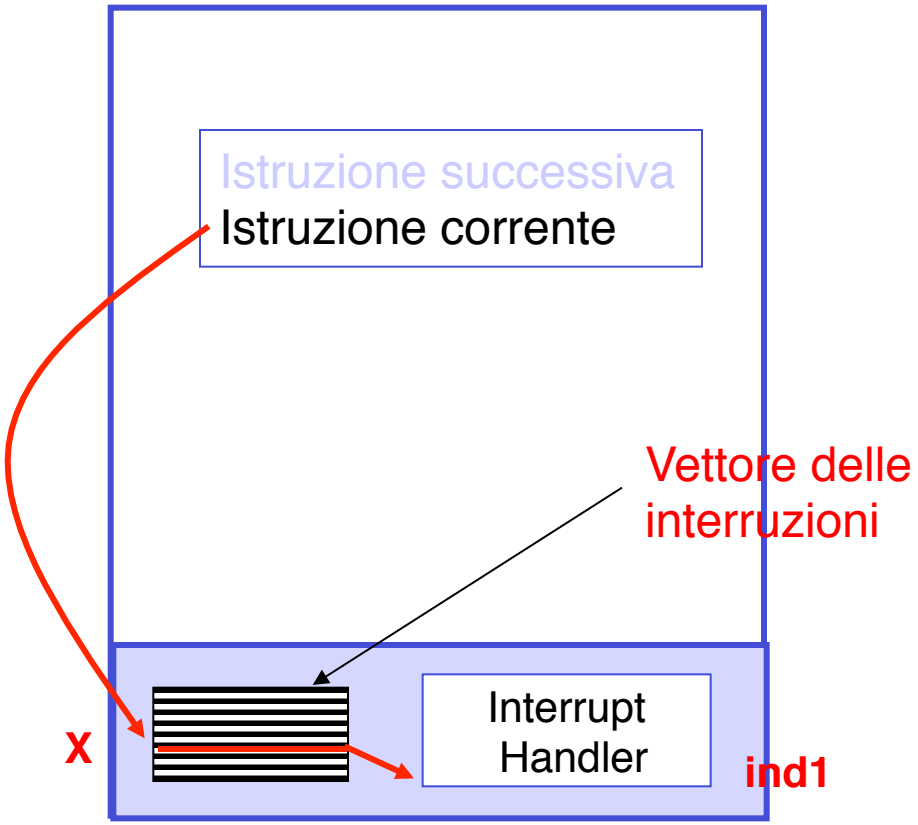

Passo 4 : Il processore interrompe il flusso di esecuzione corrente e manda in esecuzione il gestore delle interruzioni di tipo X, vediamo come...

# Interruzioni (4)

## Passo 4 :

4.1 (hw) si rileva il segnale di interruzione presente, il contenuto di PC/PSW viene salvato sulla pila, si passa in modo kernel

4.2 (hw) l'indirizzo del dispositivo (il valore X sul bus) viene usato come indice nel *vettore delle interruzioni*



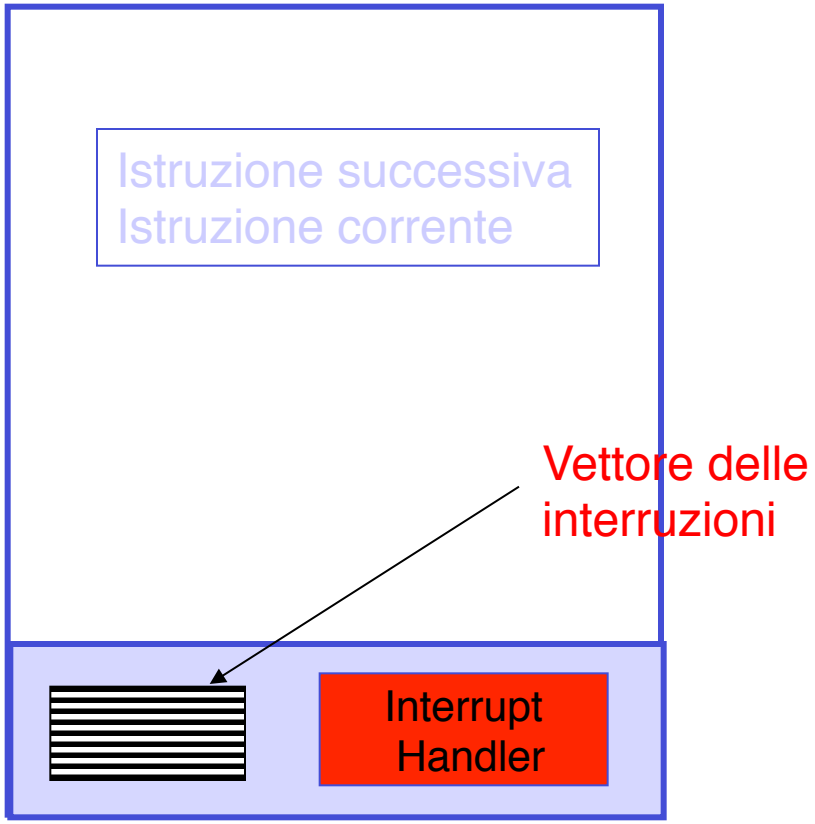

Il contenuto della posizione X del vettore delle interruzioni è l'indirizzo della prima istruzione del gestore delle interruzioni di tipo X (ind1) e viene caricato nel PC.

# Interruzioni (5)

## Passo 4 :

4.3 (sw) Il gestore prende il controllo e svolge le operazioni necessarie

a) (assembler) il contenuto dei registri del processore viene salvato (in una struttura del SO riservata a quel processo: PCB o Tabella dei Processi (TP))



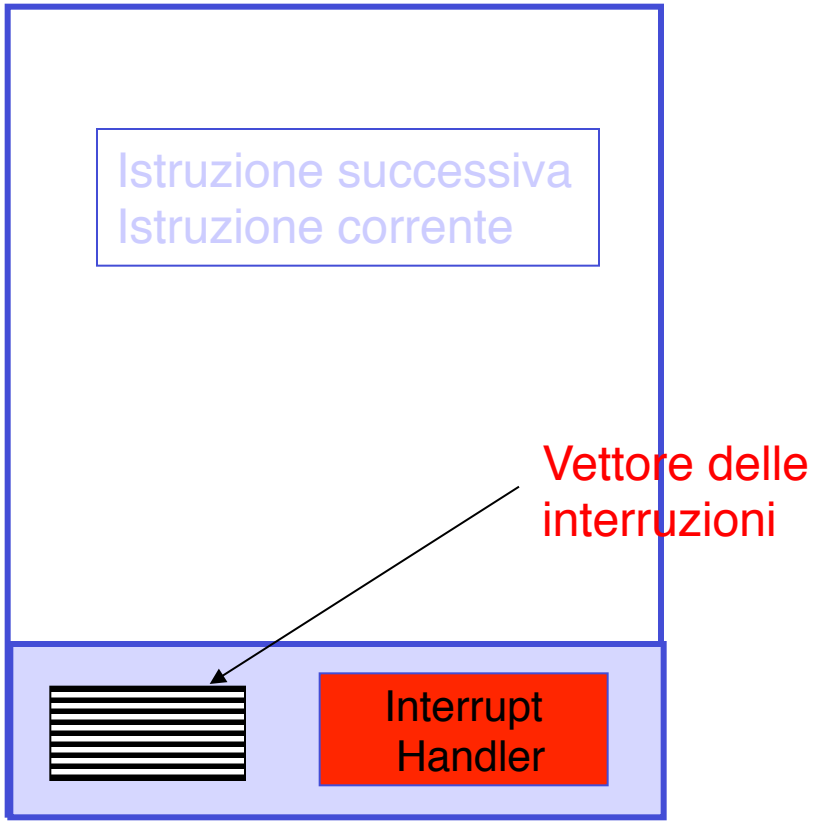
# Interruzioni (6)

## Passo 4 :

4.3 (sw) Il gestore prende il controllo e svolge le operazioni necessarie

b) (assembler) rimuove dallo Stack l'informazione salvata dall'hw e la salva sempre nella TP

c) (assembler) salva il contenuto di SP in TP e setta SP in modo che punti a uno stack temporaneo nel kernel



# Interruzioni (7)

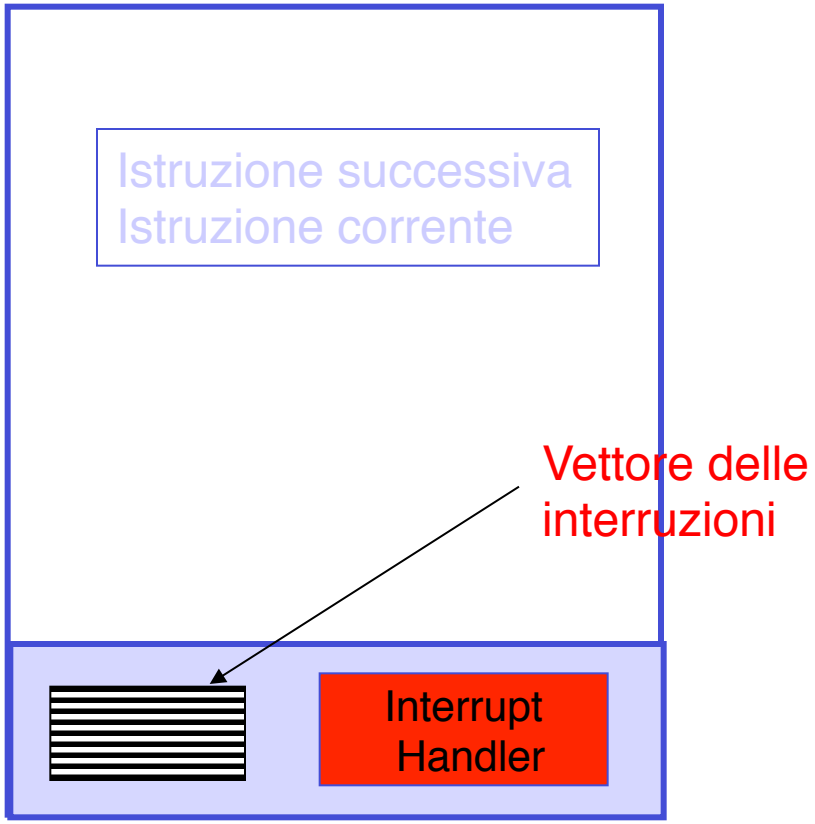
## Passo 4 :

4.3 (sw) Il gestore prende il controllo e svolge le operazioni necessarie

d) (C) effettua il lavoro specifico del tipo di interruzione verificato, tipicamente :

- salva i dati del controller
- segnala il fatto un processo in attesa del verificarsi di quell'evento

e) (C) viene chiamato lo scheduler

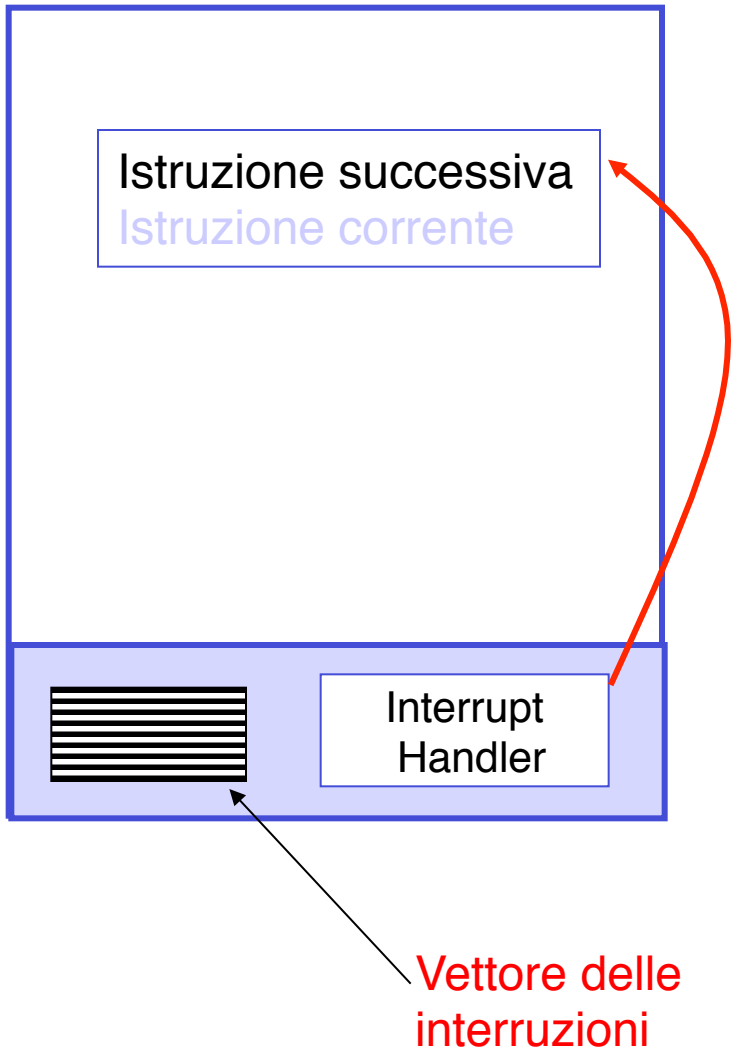


# Interruzioni (8)

## Passo 5 :

4.4 Quando il scheduler ha deciso quale processo mandare in esecuzione (di solito non quello interrotto)

a) (assembler) viene ripristinato il valore dei riegistri PC/PSW , si ritorna in modo utente. In pratica, si esegue l'istruzione successiva a quella interotta



# Gestione del processore

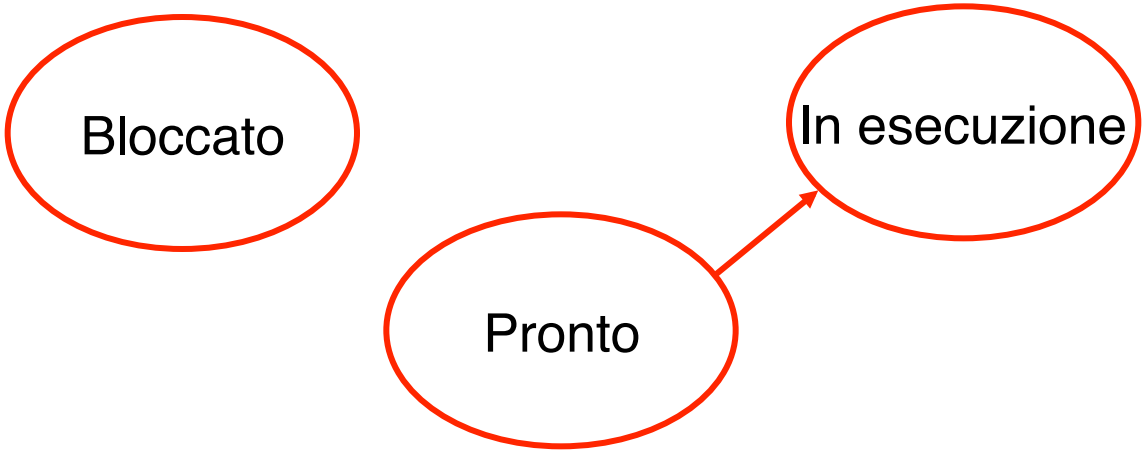
Stati di un processo

Implementazione del modello a processi

# Stati di un processo

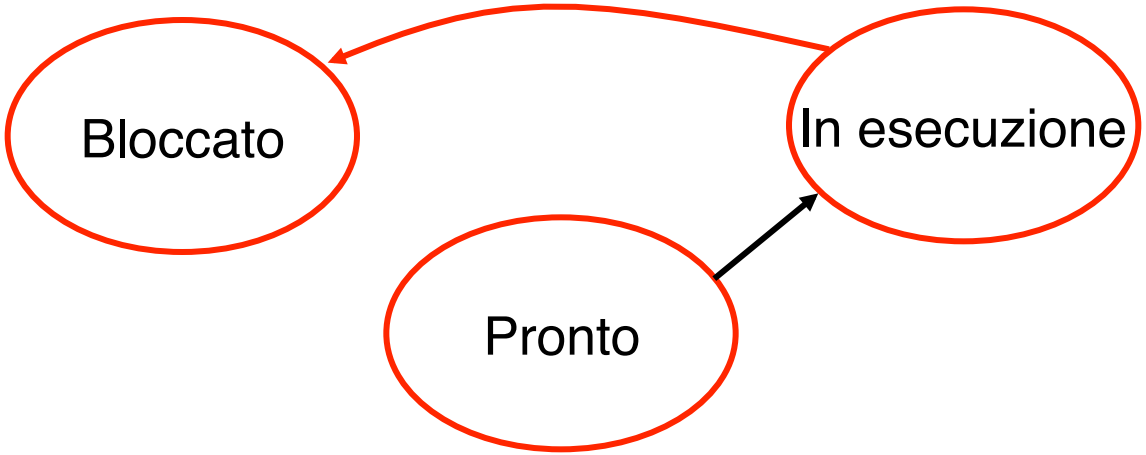

- Come si passa da uno stato all'altro ?

# Stati di un processo (2)



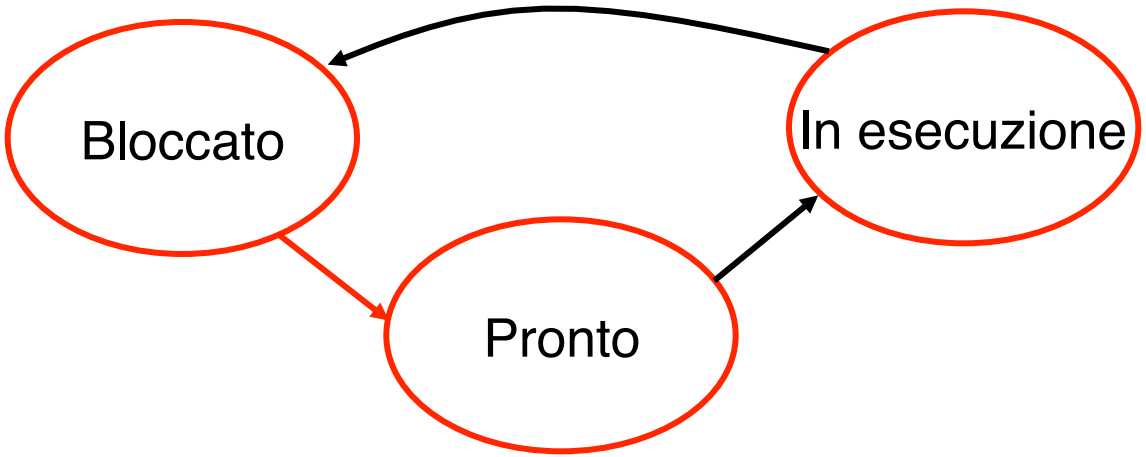

- Lo scheduler ha deciso di mandare in esecuzione proprio questo processo

# Stati di un processo (3)



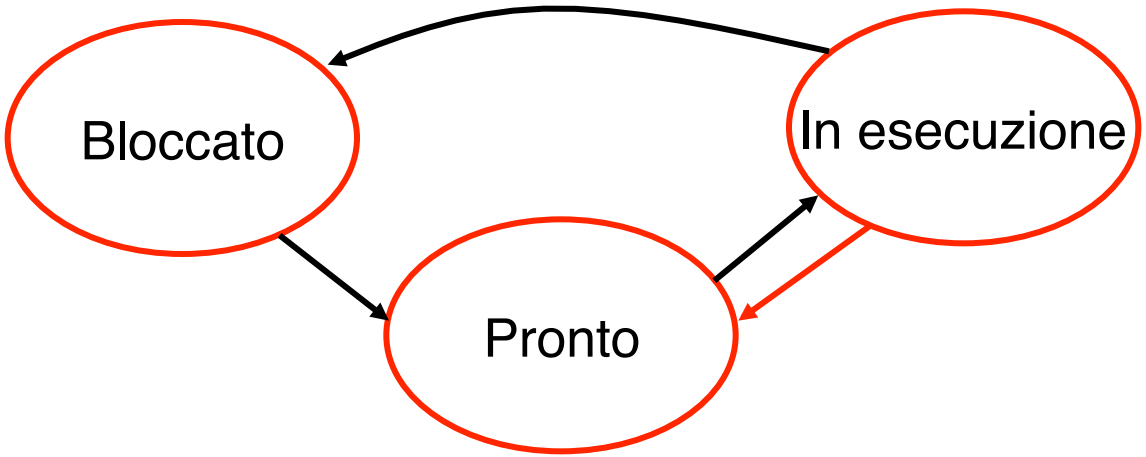

- Il processo ha eseguito una SC e il servizio non può essere completato

# Stati di un processo (4)



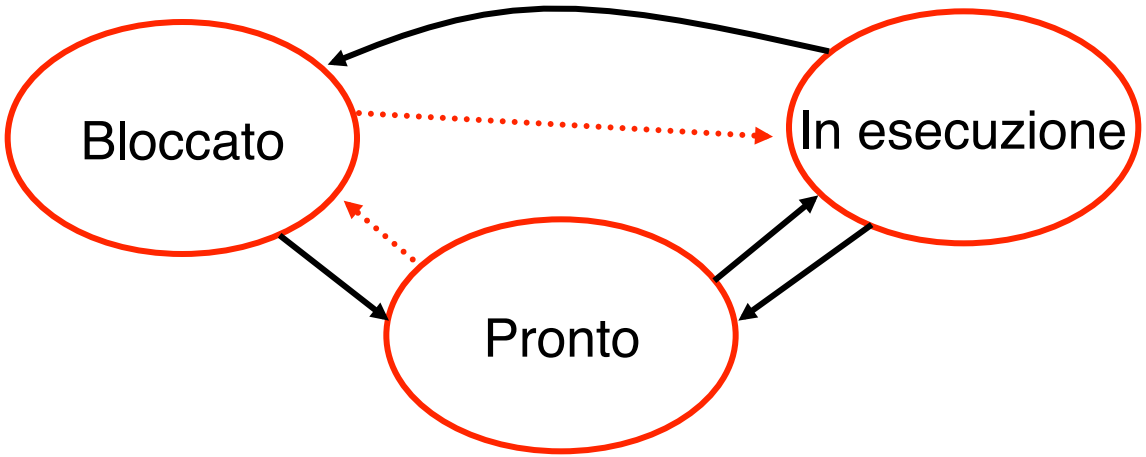

- Il servizio richiesto è stato completato
  - es : è arrivata una interruzione dal dispositivo, ...

# Stati di un processo (5)




- Lo scheduler ha deciso di assegnare il processore ad un altro processo

# Stati di un processo (6)




- Le altre due transizioni tipicamente non hanno senso

# Implementazione di processi (1)

- Le informazioni relative a tutti i processi attivi ad un dato istante sono mantenute nella *Tabella dei processi*:
  - un array di strutture (record)
  - una struttura per ogni processo
    - terminologia : la singola struttura può essere denominata anche PCB, noi useremo la terminologia corrente Unix/Linux e chiameremo tabella dei processi anche la singola struttura

# Implementazione di processi (2)

- La TP contiene tutte le informazioni sul processo diverse dal suo spazio di indirizzamento
  - valore PC, SP, PSW, registri generali
    - significativo solo quando il processo non è in esecuzione
  - stato (pronto, bloccato …)
  - informazioni relative ai file che il processo sta utilizzando

# Implementazione di processi (3)

- La TP contiene tutte le informazioni sul processo diverse dal suo spazio di indirizzamento (cont.)
  - informazioni relative alla RAM occupata dal processo
    - es : valore dei registri base e limite …
  - altre informazioni dipendenti dal particolare SO
    - es. quantità di tempo CPU utilizzato di recente (algoritmi di scheduling), informazioni legate a meccanismi di IPC (es. segnali Unix)

# Implementazione di processi (4)

- *Cambio di contesto (context switch)* : è ciò che accade quando un processo passa in esecuzione
  - il processore deve caricare i propri registri interni con le informazioni relative al nuovo processo da mandare in esecuzione
  - l'hw che realizza rilocazione e protezione deve essere aggiornato
    - es : registri base e limite, i sistemi attuali usano meccanismi più sofisticati

# Implementazione di processi (5)

- *Cambio di contesto (context switch)* : è ciò che accade quando un processo passa in esecuzione (cont.)
  - la cache deve essere invalidata e ricaricata con le informazioni relative al processo in esecuzione
  - varie tabelle e liste interne al SO devono essere aggiornate

# Implementazione di processi (6)

- Il cambio di contesto è una operazione molto costosa!
  - Il costo in sistemi reali è dell'ordine del millisecondo!
  - Quindi lo scambio non può essere effettuato troppo spesso (20-200ms in sistemi reali Unix Linux)