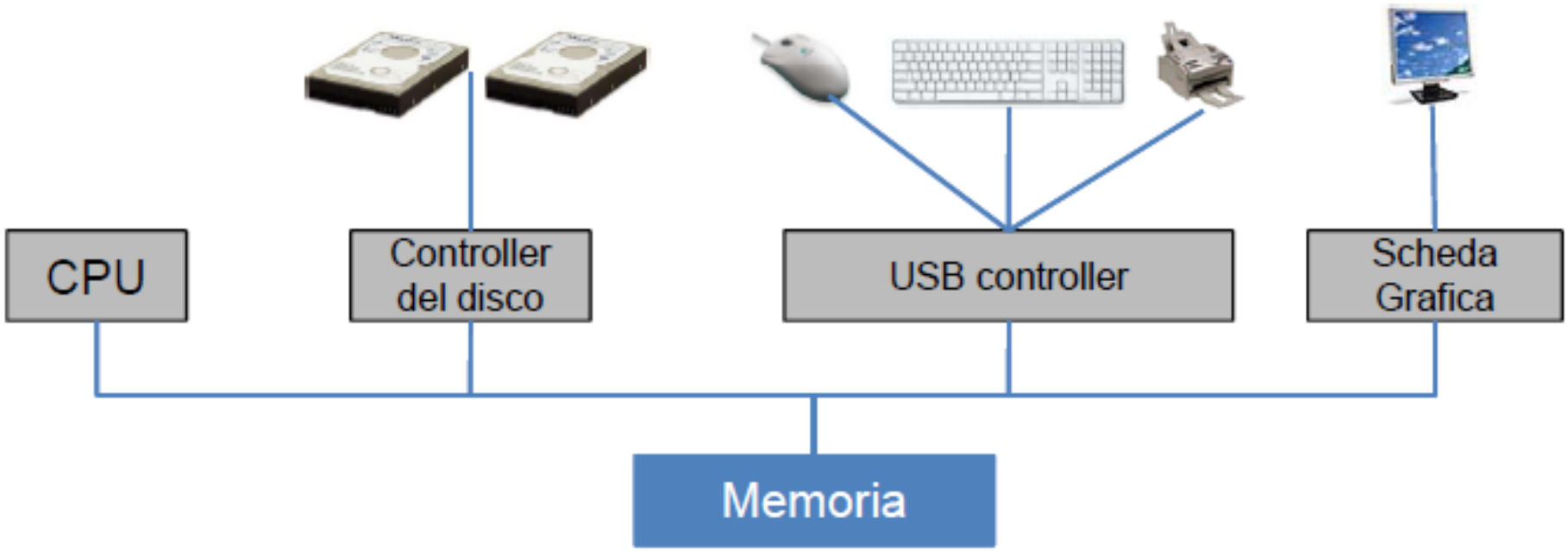
Fondamenti dei Sistemi Operativi

Struttura di un calcolatore

Sommario

- Struttura e funzionamento di un calcolatore *general purpose*.
- Protezione dell'hardware.
- Modalità di funzionamento: *user mode* e *system mode*.
- Interruzioni: *interrupt* e *trap*.
- La protezione nell'input/output.
- Concorrenza di I/O e calcolo.
- Coordinamento tra controller e CPU.
- I/O sincrono e asincrono
- *Device Status Table*
- DMA
- Memoria principale e memorie secondarie: i dischi magnetici
- Gerarchia di memorizzazione
- Protezione memoria e protezione CPU

Struttura di un generico PC



- Un sistema di calcolo è composto da una o più **CPU**, alcuni **controllori** di dispositivi di I/O (device) connessi attraverso un **bus** alla **memoria centrale**
- Questi dispositivi, assieme alla CPU, operano in maniera concorrente per ottenere cicli di accesso alla memoria

The O.S. Interface (shell)

È la parte esterna del SO, attraverso la quale gli utenti richiedono l'esecuzione dei programmi (siano essi programmi "di utilità" dello stesso SO o programmi "applicativi") o accedono ai servizi del kernel.

L'interfaccia può essere di tre tipi

.... a caratteri

L'interfaccia a caratteri o **shell** o **command interpreter** è un vero e proprio linguaggio di programmazione che permette all'utente di controllare la esecuzione di comandi, sia in modo interattivo che in modo "batch" (mediante "script" di shell). Tipici SO con interfaccia a caratteri sono il DOS, UNIX, LINUX, ecc.

.... grafica o iconica

L'interfaccia grafica è un sistema d'interazione basato su simboli (icone) che rappresentano le risorse su cui è possibile operare. Tipici SO con interfaccia iconica sono WINDOWS (in tutte le sue versioni), MAC OS, UNIX vers. OSF Motif, LINUX vers. OSF Motif, ecc.

.... a manipolazione diretta

prevede ingressi mono e *multi-touch* come *strisciate*, *tocchi* e *pizzichi* sullo schermo per manipolare gli oggetti visibili sullo stesso. Sensori hardware interni come accelerometri, giroscopi e sensori di prossimità sono utilizzati ad esempio per regolare lo schermo da verticale a orizzontale a seconda di come il dispositivo è orientato. Tipici SO con interfaccia a manipolazione diretta sono iOS, ANDROID, ecc

THE CHARACTER SHELL (*)

Una shell a caratteri si basa sull'esecuzione di tre tipi di **comandi**:

1. **oggetti eseguibili (o comandi esterni)**
   Sono file contenenti programmi in formato eseguibile. Esempio: **`sort`**
2. **comandi built-in (o comandi interni)**
   Sono comandi che realizzano funzioni semplici, eseguite direttamente dalla shell. Esempio: **`cd`**
3. **script (o comandi batch)**
   sono "programmi" in linguaggio di shell

Quando si digita un comando, la shell verifica se si tratta di un comando built-in. In caso positivo, lo esegue. Altrimenti la shell crea (fork) un nuovo processo che esegua il comando.

Oltre all'esecuzione di comandi, la shell consente altre funzioni tipiche:
- ***ridirezione*** dell'input e dell'output
- ***"pipeline"*** di comandi
- ***filtri***

(*) si fa riferimento ad una tipica interfaccia a caratteri di un SO UNIX-like

THE GRAPHICAL SHELL (*)

Un'interfaccia grafica si basa sulla manipolazione di "**risorse**"

Una risorsa è un **oggetto** che fornisce o elabora informazioni

In Windows le risorse sono rappresentate da icone ed hanno un nome

Esempi di risorse

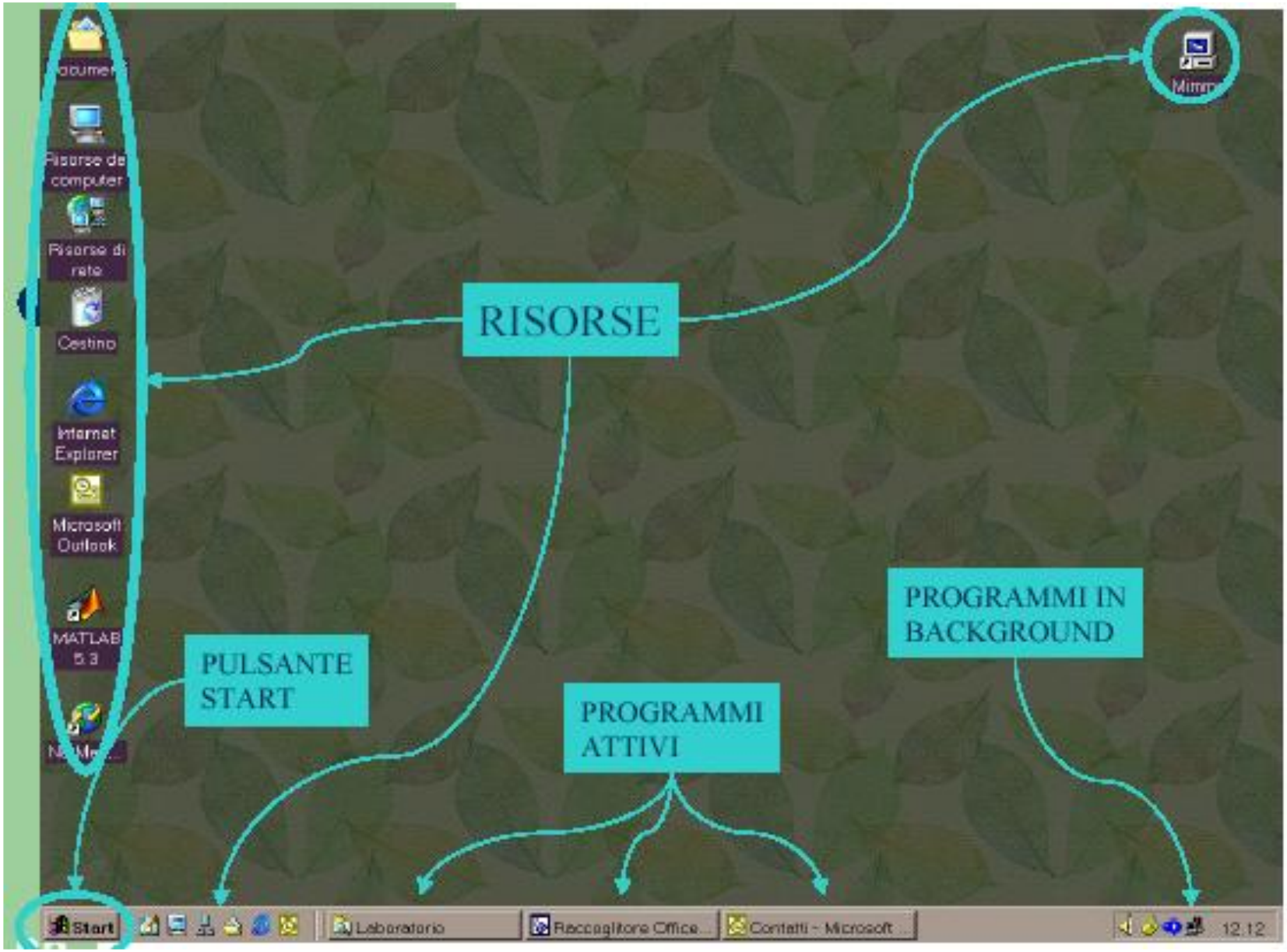
- I dischi e le stampanti
- Tutti i dispositivi esterni
- I file e le cartelle
- I programmi
- I componenti di Windows

Il desktop

È una risorsa speciale di Windows, realizzata come cartella in cui sono contenuti documenti (oggetti) di interesse generale ed organizzata come "scrivania" virtuale, su cui sono disposte le icone degli oggetti.

(*) si fa riferimento ad una tipica interfaccia grafica di un SO WINDOWS-like

THE WINDOWS DESKTOP



THE ACTIVITIES MANAGED BY AN O.S.

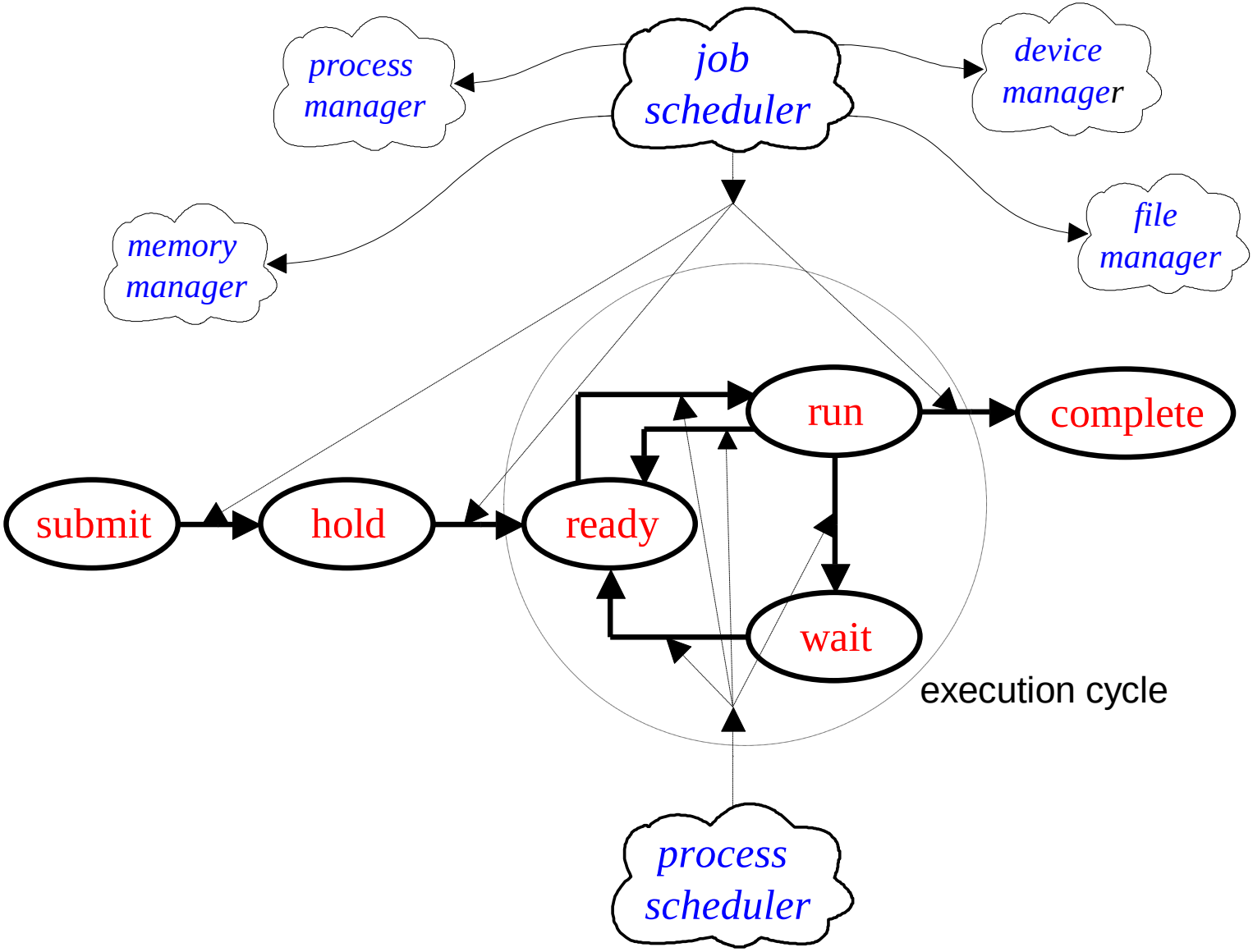
Un ***job*** è costituito dalla sequenza di uno o più job-step (programmi).

Un ***job-step*** (o programma) è costituito da uno o più task (processi cooperanti) che operano «contemporaneamente» fra loro, oltre che con altri processi (processi concorrenti), competendo nell'utilizzo delle risorse (CPU, memoria centrale, memoria di massa, dispositivi, file) presenti nel sistema di elaborazione.

Il sistema operativo stesso è costituito da processi che cooperano per rendere possibile l'esecuzione dei programmi.

Un ***task*** è una attività elementare ed indipendente, con associate le risorse (memoria, file, dispositivi) richieste per la sua esecuzione. Ad ogni task il SO associa un *Task Control Block* (TCB) o *Process Control Block* (PCB), attraverso il quale il SO ha nozione e controlla la singola attività atomica.

THE STATE DIAGRAM OF A PROGRAM



PROGRAM EXECUTION PROCEEDING (1/2)

Il sistema operativo produce l'avanzamento dell'esecuzione dei programmi.

Dapprima le richieste sottoposte vengono raggruppate, costituendo la coda dei job di cui viene richiesta l'esecuzione (coda dello **stato di submit**).

Dopo l'analisi delle risorse necessarie a soddisfare le richieste dei job nello stato di *submit*), i primi job-step di ogni job vengono ordinati – in relazione ai criteri di priorità (*macroscheduling*) del ***Job Scheduler*** – costituendo la coda dei job-step in attesa di essere eseguiti (coda dello **stato di hold**).

A partire dalla richiesta di esecuzione prioritaria nella coda di hold, il Job Scheduler, tramite i gestori delle risorse, verifica la disponibilità delle risorse necessarie e, dopo averne avuto conferma, attiva il ***Process Scheduler***. Questo trasforma la richiesta di esecuzione in processo (caricando nella memoria centrale il codice eseguibile corrispondente al programma richiesto), costruisce un blocco di informazioni (*Process* o *Task Control Block*), che, progressivamente aggiornate, permetteranno al SO di gestire il programma nel ***ciclo di esecuzione*** e immette tale blocco nella coda dei programmi pronti per l'esecuzione (coda dello **stato di ready**).

PROGRAM EXECUTION PROCEEDING (2/2)

Inizialmente il blocco delle informazioni relative al singolo programma (TCB o PCB) viene immesso nella coda dello **stato di ready**. Tale coda è organizzata con criteri di priorità (*microscheduling*) con cui il Process Scheduler ordina i vari task che, disponendo di tutte le risorse necessarie all'esecuzione, abbisognano ora della sola CPU per essere effettivamente eseguiti.

Non appena la CPU si libera (o il rispetto delle priorità lo richiede), viene operato il cambio del contesto operativo (*context switching*) della CPU e un processo passa nello **stato di run** (mentre quello che occupava finora tale stato retrocede di norma nello stato di *ready*).

L'attesa di specifici eventi può costringere il S.O. a far transitare il processo nella coda corrispondente allo **stato di wait**.

Dopo aver ciclato tra gli stati del ciclo di esecuzione, il processo arriva a conclusione e transita nella coda dello **stato di complete**.

Protezione dell'hardware

- La multiutenza, la multiprogrammazione e la condivisione di risorse producono un più efficiente sfruttamento del sistema, ma implicano l'insorgere di problemi derivanti dalla propagazione di bachi tra processi diversi e, quindi, sistemi operativi con funzioni di controllo residente.
- Nei sistemi multiprogrammati determinati errori potrebbero influenzare i programmi in maniera incrociata ed inficiare il funzionamento del SO residente. Si rende necessario prevedere forme di protezione HW:
  - Funzionamento in modalità differenziate.
  - Protezione dell'I/O.
  - Protezione della memoria.
  - Protezione della CPU.

Un SO ben progettato deve assicurarsi che un programma errato o malizioso non possa indurre errori incrociati.

In assenza di protezione HW, un sistema dovrebbe eseguire solo un processo per volta, altrimenti tutti i dati in uscita potrebbero essere considerati "sospetti".

Modalità di funzionamento: *user mode* e *system mode*

Condividere le risorse comporta la necessità di proteggere il SO da qualsiasi programma funzionante in modo scorretto.

Molti SO forniscono un supporto hardware che permette di distinguere vari ***modi di esecuzione***:

1. Modalità utente (***User mode***) – attività eseguita dall'utente.
2. Modalità di controllo (***Monitor mode***, *ma anche kernel mode* o *system mode*) – attività eseguita dal SO.

Un bit di modalità *(mode bit)* aggiunto all'hardware del computer per indicare la modalità corrente: monitor (0) o user (1).

Le interruzioni: tipi di interruzioni

- ***Interruzione interna*** (o *trap*): è generata dal software ed è causata da un errore (divisione per zero, indirizzo errato di memoria) o dalla richiesta di un servizio (I/O). Ha carattere sincrono: il processo che innesca la trap provoca la chiamata di un segmento di codice del nucleo del sistema operativo (*Supervisor Call o SVC*).

Software may trigger a trap by *executing a supervisor call* (SVC).

- ***Interruzione esterna*** (o *interrupt*): ha carattere asincrono, in quanto al verificarsi di un evento, la sua occorrenza è segnalata alla CPU tramite un segnale in ragione del quale viene fermata l'esecuzione del programma in corso di esecuzione e viene mandata in esecuzione la routine di servizio per il segnale rilevato dalla CPU.

Hardware may trigger an interrupt by *sending a signal to the CPU*.

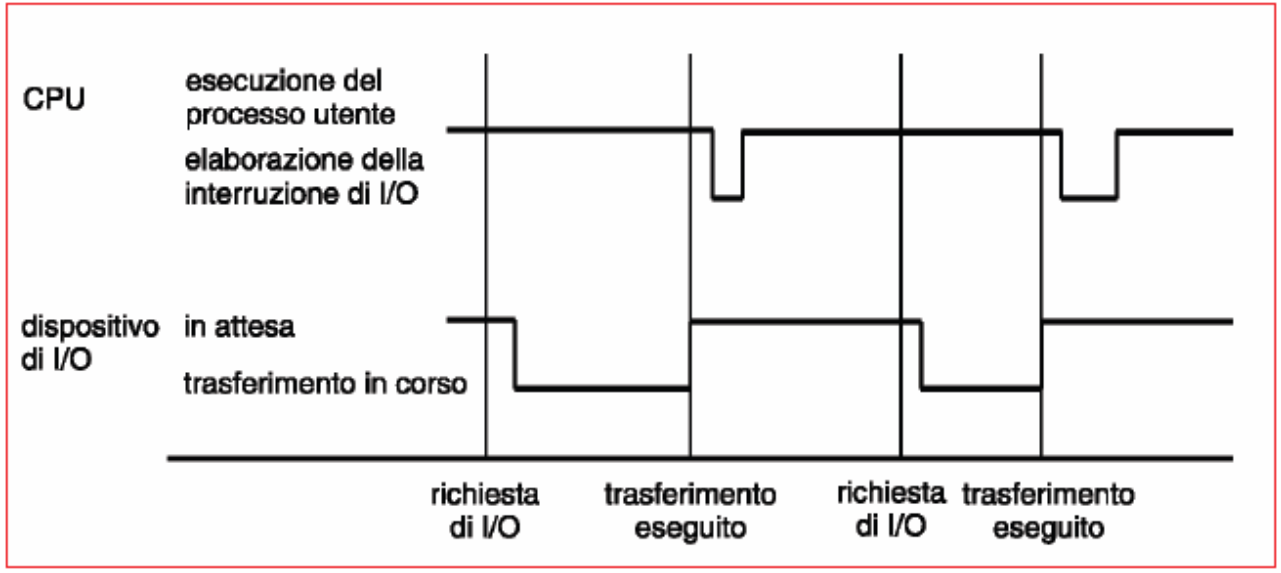
Ogni processore ha una propria architettura di SVC e interrupt.

Le interruzioni esterne: gli interrupt

L'interrupt trasferisce il controllo della CPU alla routine di gestione interrupt, generalmente attraverso un "**vettore degli interrupt**", che contiene gli indirizzi di tutte le routine di servizio.

Il SO deve salvare lo stato della CPU ed il contenuto dei registri, tra cui il PC del programma interrotto.

Gli altri interrupt (relativi alle altre periferiche) sono disabilitati (*interrupt masking*) mentre un interrupt viene trattato dal sistema, per prevenire la perdita di segnali (*lost interrupt*).



Le interruzioni interne: le SVC

Le chiamate di sistema forniscono un'interfaccia tra il programma in esecuzione e il SO.

Esse erano generalmente disponibili in assembler. I SO moderni consentono di svilupparle ora in linguaggio C o anche C++.

Una SVC viene servita facendo riferimento al vettore delle trap, analogo al vettore degli interrupt.

Tipi di chiamate di sistema

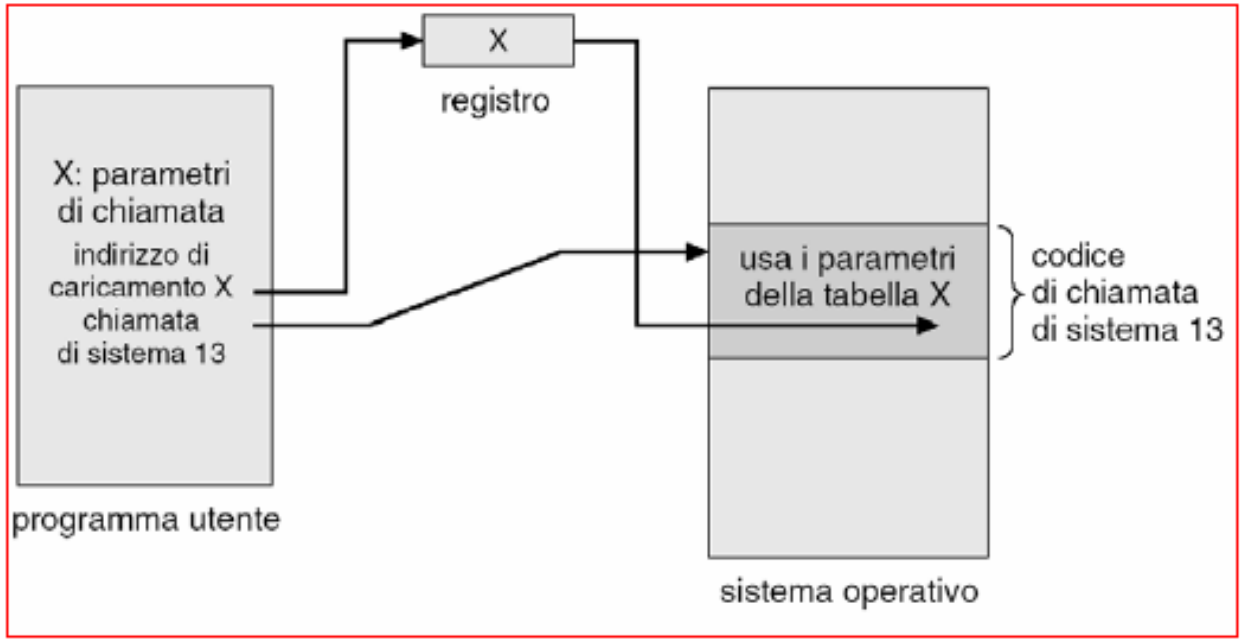
- Controllo del processo.
- Gestione dei file.
- Gestione dei dispositivi.
- Gestione delle informazioni.
- Comunicazioni.

Il passaggio di parametri in una SVC

La chiamata può consentire di passare i parametri:

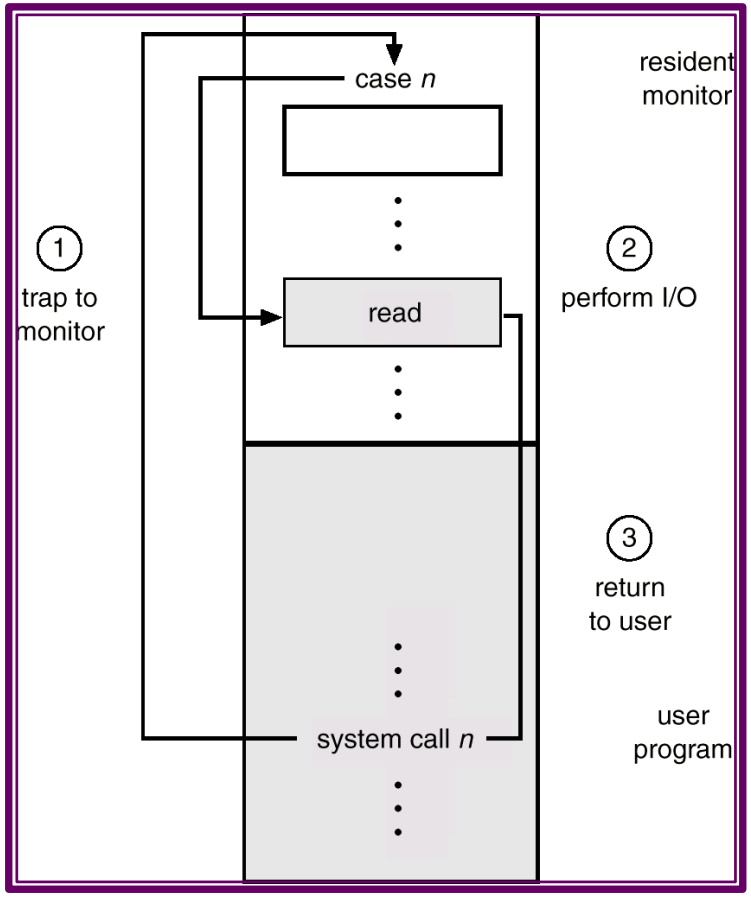
- nei registri
- memorizzandoli in una tabella in memoria e passando l'indirizzo della tabella in un registro
- facendo nel programma chiamante una push (store) dei parametri sullo stack e una pop nel SO.

Quando si presenta una trap o un interrupt l'hardware commuta dalla modalità utente a quella di sistema.



L'esecuzione delle chiamate al supervisore

Le *istruzioni privilegiate* (*System call*) possono essere eseguite solo in modalità di controllo. All'avvio, il sistema si predispone in kernel mode.

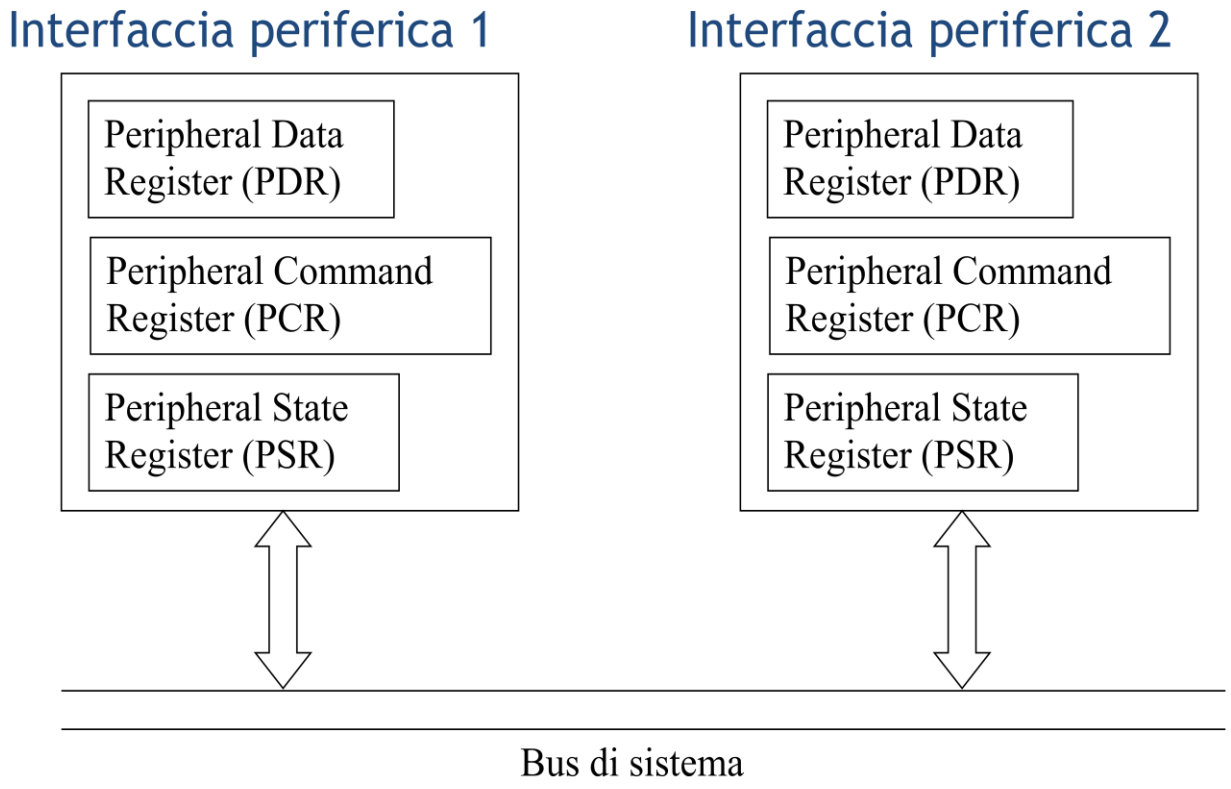


Protezione dell'I/O

- Per evitare che i processi utente effettuino I/O non consentiti, tutte le istruzioni di I/O sono privilegiate.
- I processi utente non potranno che eseguire operazioni di I/O attraverso *System call*.
- Affinchè la protezione I/O sia completa, un programma utente non deve mai poter ottenere il controllo della macchina in modalità supervisore.
  - In caso contrario, il programma potrebbe commutare in modalità supervisore tutte le volte che si presenta un interrupt o una trap, saltando all'indirizzo determinato dal vettore di interrupt.
  - In tal caso un programma utente malizioso potrebbe memorizzare (come parte della sua esecuzione) un nuovo indirizzo per la routine di gestione dell'interruzione nell'interrupt vector.
  - In tal modo al successivo interrupt il programma potrebbe via via alterare l'intero vettore di interruzioni e carpire il controllo della macchina in modalità supervisore.

Concorrenza di I/O e calcolo

- I dispositivi di I/O e la CPU possono funzionare concorrentemente
- Ogni controller di dispositivo gestisce un particolare dispositivo periferico o un set di dispositivi simili.
- Ogni controller ha una memoria temporanea locale (buffer locale)
- La CPU muove dati da/per la memoria principale per/da i buffer locali dei controller



Coordinamento tra controller di I/O e CPU

- l'Input procede dal dispositivo al buffer locale del controller, mentre l'Output procede dal buffer locale del controller al dispositivo
- Per iniziare un'operazione di I/O, la CPU carica gli appropriati registri nel device controller.
- Il device controller esamina il contenuto dei registri per determinare l'operazione da intraprendere e fa partire il trasferimento di dati richiesto.
- Appena il trasferimento è completato, il controller invia un interrupt alla CPU per segnalare la fine della operazione richiesta..

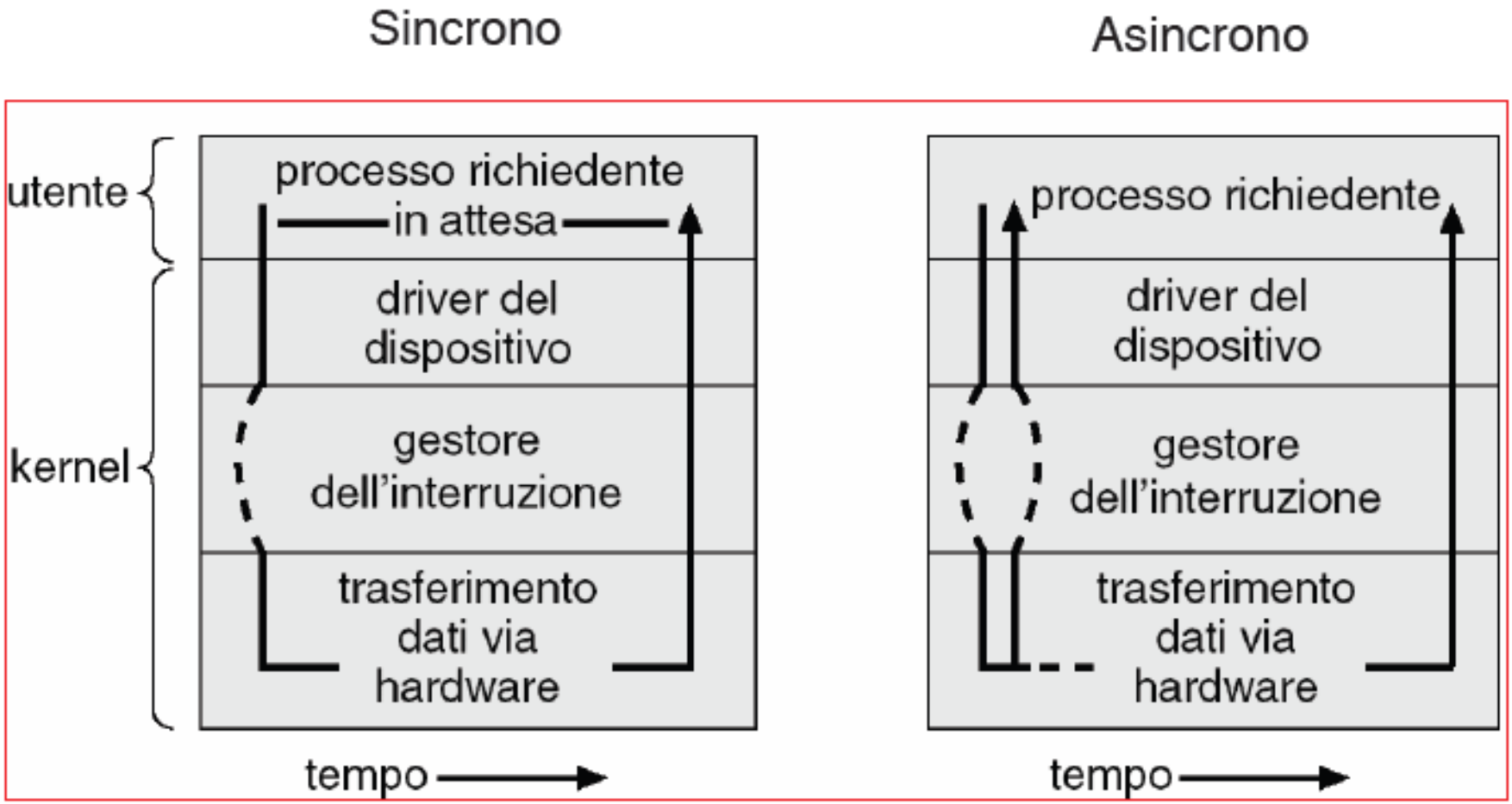
L'operazione può essere effettuata in 2 differenti modi: I/O Sincrono o Asincrono.

Synchronous and asynchronous I/O 1/2

- **I/O Sincrono** – l'operazione viene iniziata e al suo completamento il controllo viene restituito al processo utente.
  - L'istruzione speciale *wait* (oppure un ciclo stretto di attesa (*tight wait loop*) `LOOP: jmp LOOP`) rende la CPU inattiva fino all'interrupt successivo (interrupt di ritorno).
  - Solo una richiesta di I/O alla volta può essere tenuta in sospeso; non sono consentiti processi di I/O simultanei.

- **I/O Asincrono** – l'operazione viene iniziata e il controllo viene restituito al processo utente senza attendere il completamento dell'operazione.
  - Chiamata di sistema (*System call*) – richiesta al S.O. per far sì che un processo utente possa attendere il completamento dell'I/O.
  - ***Device-status table*** contiene un valore di ingresso per ogni dispositivo di I/O che ne indica il tipo, l'indirizzo e lo stato.
  - Il SO esegue una ricerca nella tabella del dispositivo di I/O per determinarne lo stato e modifica il valore in ingresso per rispecchiare l'interrupt.

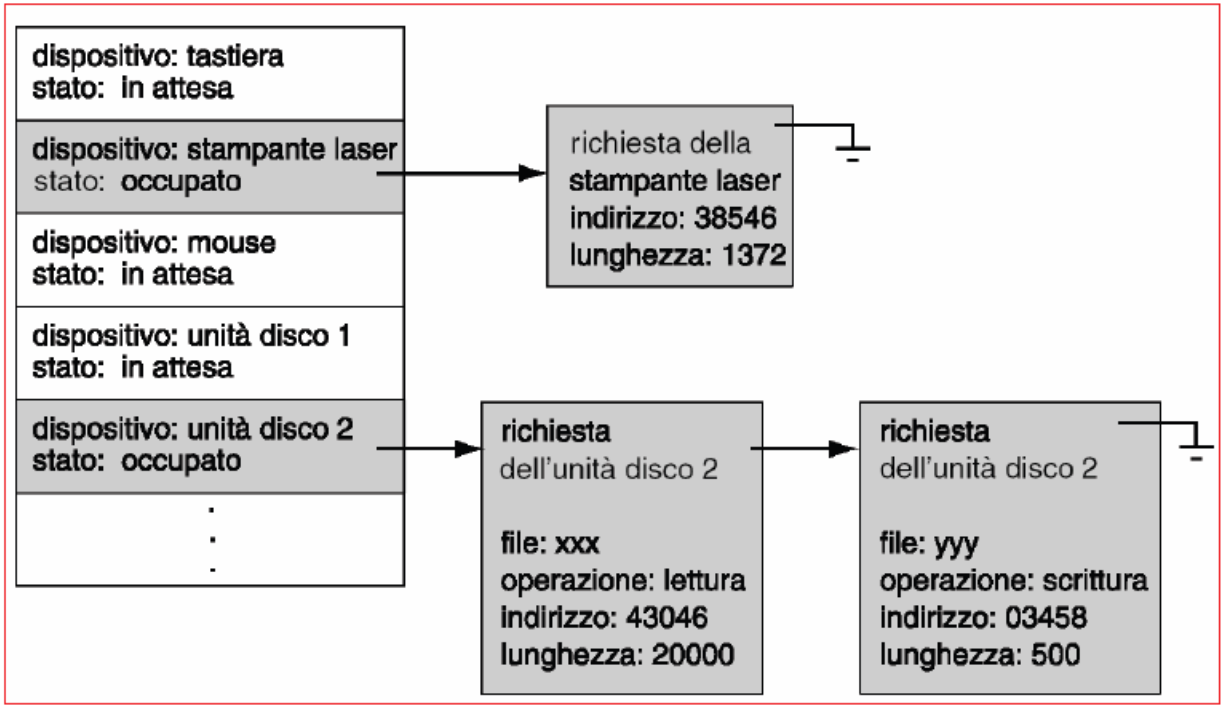
Synchronous and asynchronous I/O 2/2



Device Status Table

Per ovviare al problema dell'istruzione di wait o del ciclo di loop stretto e per tener traccia di diverse richieste di I/O allo stesso tempo, il SO usa una tavola (*Device Status Table* o DST) contenente un entry per ogni *device*, in cui è riportato il tipo, l'indirizzo e lo stato del device stesso.

Il SO accede alla DST per determinare lo stato del device e per modificare il suo contenuto. Ogni entry nella DST punta ad una coda esterna che contiene, in ordine cronologico, le richieste di I/O da quel dispositivo.



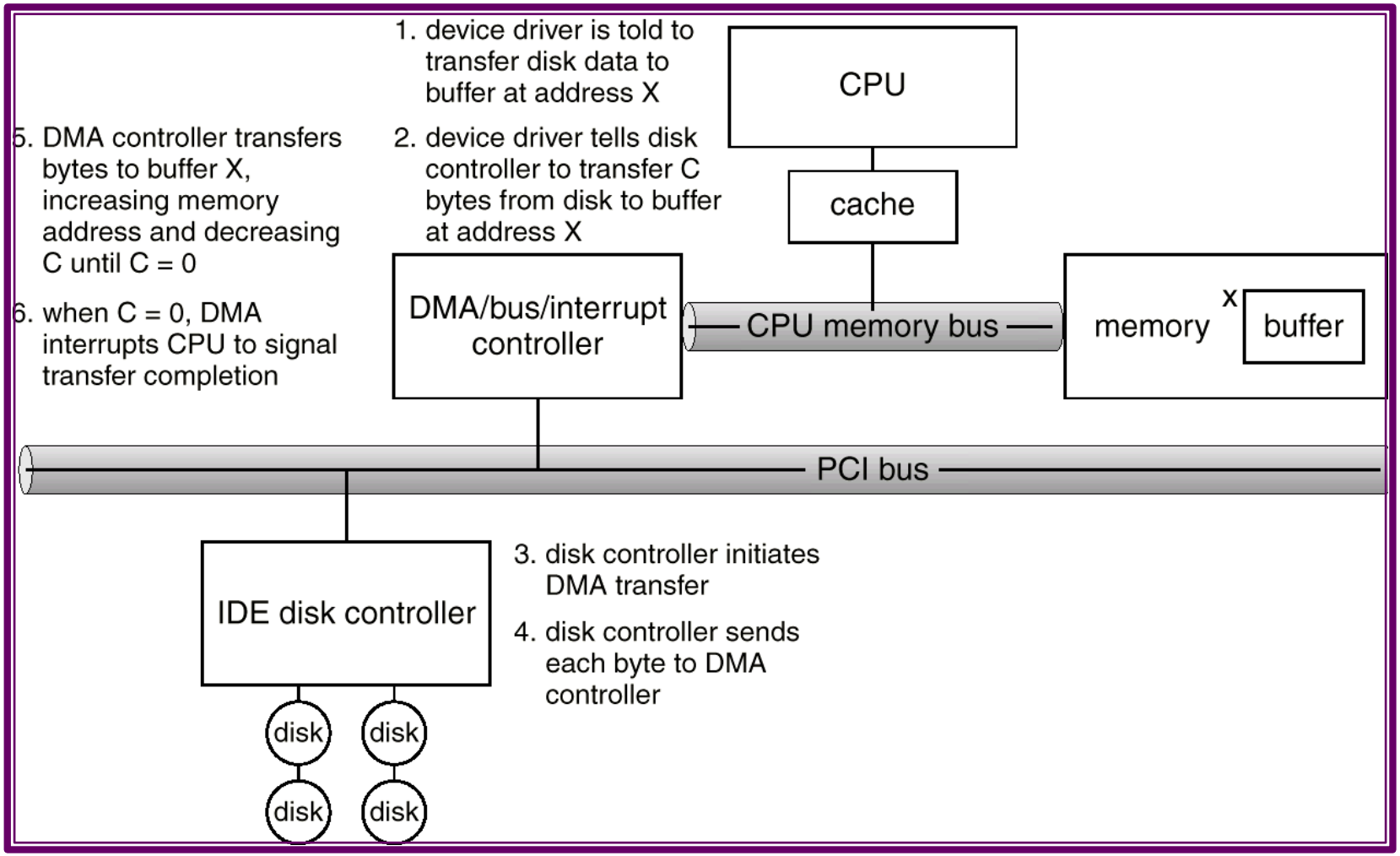
Direct Memory Access (DMA) 1/2

Una tastiera trasferisce1 byte ogni 1000 µsec, una tipica routine di interruzione per I/O occupa 2 µsec di tempo di CPU. Quindi la CPU è in *idle* per i restanti 998 µsec.

Un HD o un network ad alta velocità trasferiscono 1 byte ogni 4 µsec. La CPU è dunque in *idle* per 2 µsec. Quindi non resta molto tempo per l'esecuzione del processo.

- Il DMA è usato per dispositivi di I/O ad alta velocità, in grado di trasmettere informazioni a velocità prossime a quella della memoria.
- Funzionamento di base:
  - Un programma o il SO richiedono un trasferimento di dati.
  - Il SO identifica il buffer per il trasferimento.
  - Il modulo DEVICE DRIVER del SO inizializza i registri del controller del DMA
    - Identificazione della periferica sorgente/destinazione.
    - Identificazione degli indirizzi di memoria destinazione/sorgente.
- Il controller del dispositivo trasferisce un intero blocco di dati direttamente al/dal proprio buffer da/in memoria, senza alcun intervento da parte della CPU.
- Viene generato soltanto un interrupt per blocco, invece che un interrupt per ogni byte.

Direct Memory Access (DMA) 2/2



Memoria principale e secondaria

La Main Memory – unica unità di memorizzazione a cui la CPU può accedere direttamente via bus.

- Indirizzamento alla *word*.
- Interazione mediante sequenze di primitive *load/store* che spostano dati da/verso la memoria centrale a/da i registri della CPU.

Memoria secondaria – estensione della memoria centrale che è in grado di mantenere grandi quantità di dati in modo permanente.

Disco magnetico – il più comune tipo di memoria secondaria, costituita da piatti con profilo piano circolare in metallo rigido o vetro ricoperti da materiale ferromagnetico. Come si vedrà nella slide seguente:

- I piatti ruotano solidalmente intorno ad un asse di rotazione.
- La superficie di un piatto è divisa in modo logico in tracce, che sono suddivise in settori; il luogo delle tracce che si trovano alla stessa distanza dall'asse di rotazione viene detto cilindro.
- Il controller dei dischi (disk controller) determina l'interazione logica tra il dispositivo e il computer.

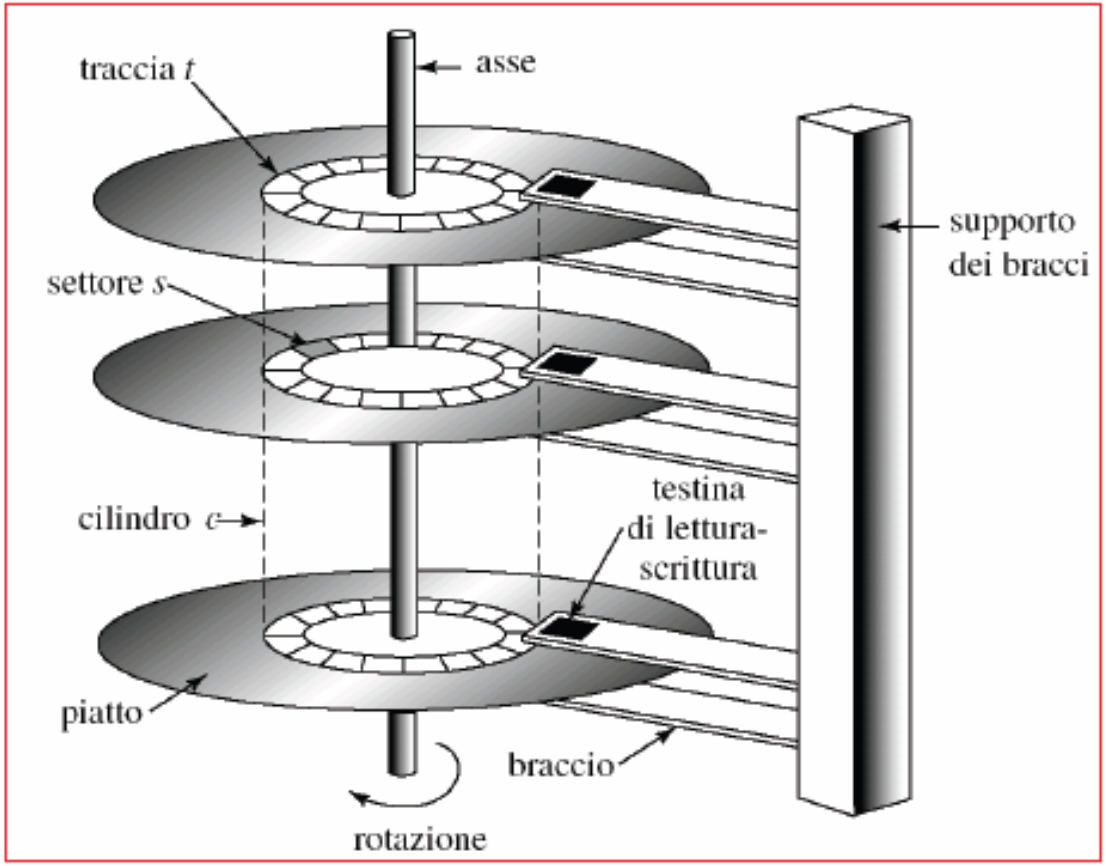
Struttura di un disco magnetico

Velocità rotazionali da 5400 a 7200 giri al minuto (rpm).

Il tempo di accesso dipende da:

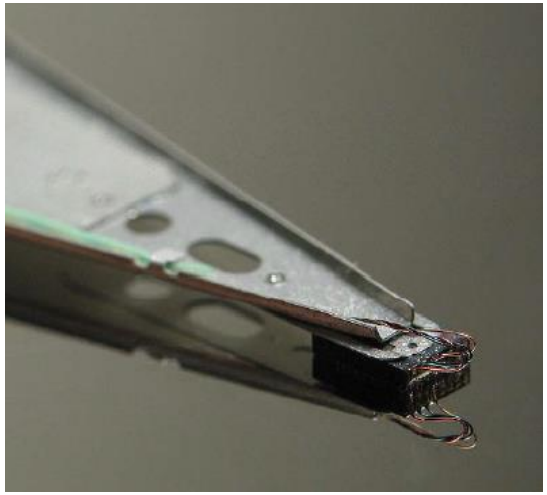
- *Transfer rate*.
- *Tempo di posizionamento* o *Seek time*: tempo di spostamento radiale delle testine per raggiungere il cilindro desiderato.
- *Latenza rotazionale* o *Search time*: tempo di rotazione per raggiungere il settore desiderato.

Un *drawback* molto comune è l'*headcrash*.



Schema funzionale di un disco

La testina



Geometria fisica del disco

- La *geometria fisica del disco* è costituita dalle coordinate ***CTS***, esprimibili indicando Cilindro, Traccia o testina, Settore
- In questo modo è possibile indirizzare univocamente ciascun blocco di dati presente sul disco

Ad esempio, se un disco rigido si compone di:

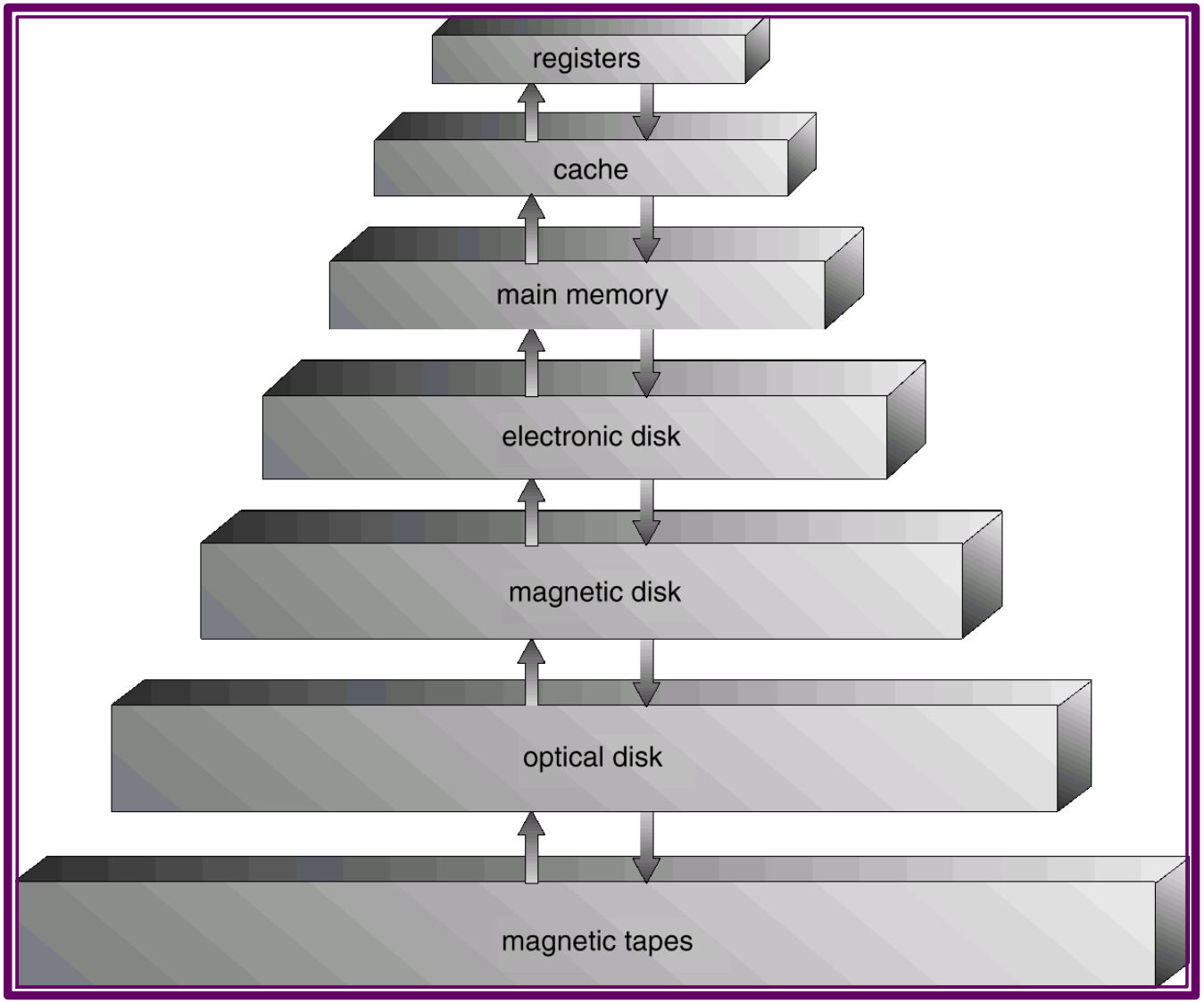
- 2 dischi (o 4 piatti)
- 16384 cilindri
- 16 settori di 4096 byte per traccia
- La capacità del disco sarà di 4×16384×16×4096 byte, ovvero 4 GB

Gerarchia di memorizzazione

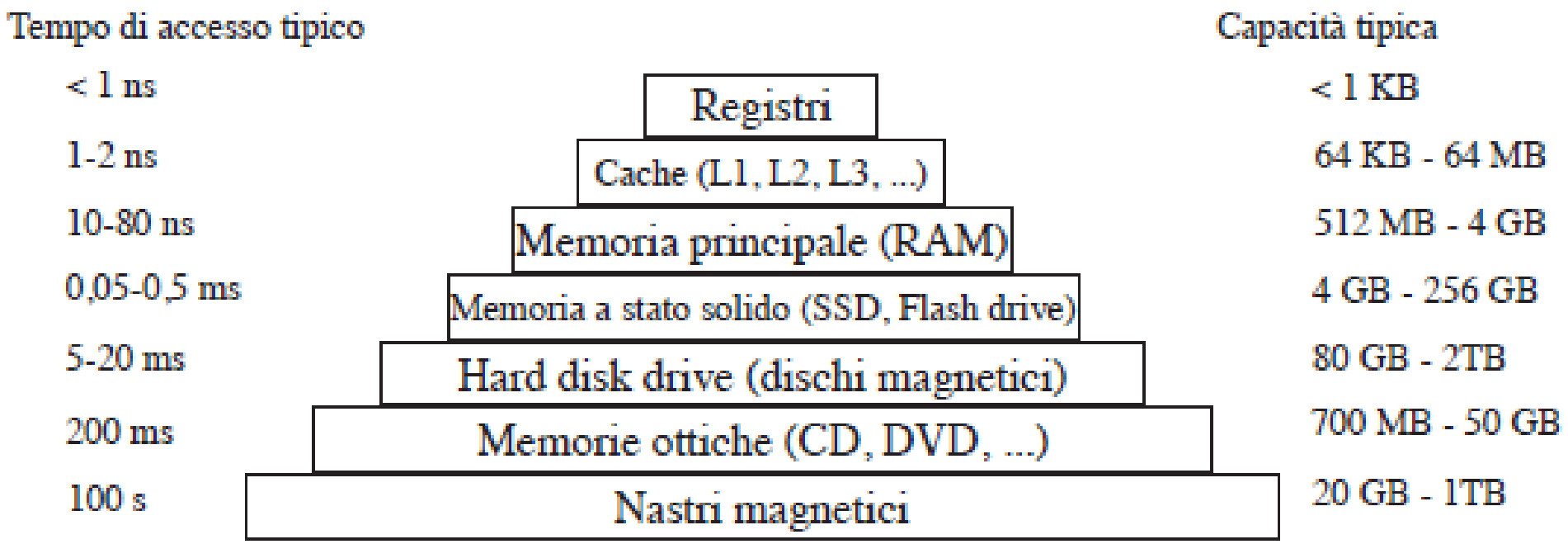
I sistemi di memorizzazione sono organizzati gerarchicamente, secondo *velocità/tempo di accesso*, *costo* e *volatilità*.

Caching – copia temporanea dell'informazione in un sistema di memorizzazione più veloce.

La memoria centrale può essere vista come *cache* veloce per dispositivi di memorizzazione secondari.



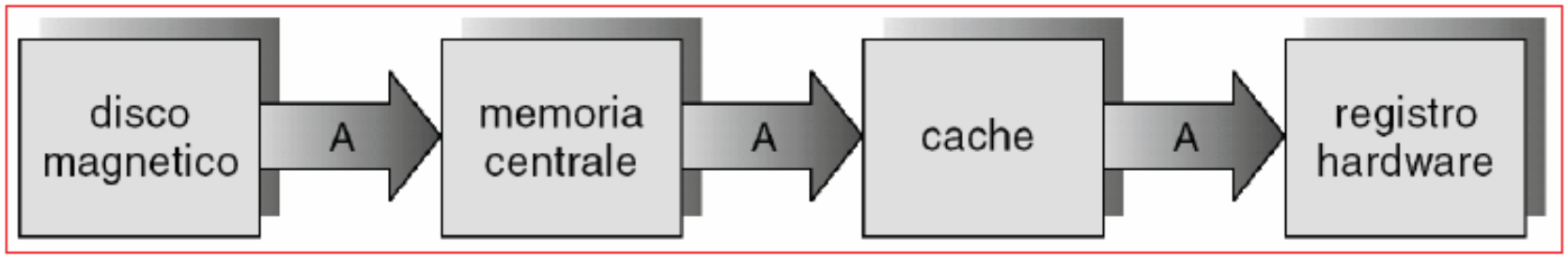
Gerarchia di memorie: caratteristiche



Cache memory

- Uso di una memoria ad alta velocità per mantenere i dati di accesso recente.
- Necessita di una politica di *gestione della cache.*
- Il movimento delle informazioni tra i livelli della gerarchia di memorizzazione può essere sia esplicito che implicito.
- Il caching introduce un altro livello nella gerarchia dei dispositivi di memorizzazione.
  - Questo richiede che i dati memorizzati simultaneamente in più di un livello siano *coerenti* e *consistenti*.

Migrazione di un dato A dall'HD ai registri HW



Protezione della memoria

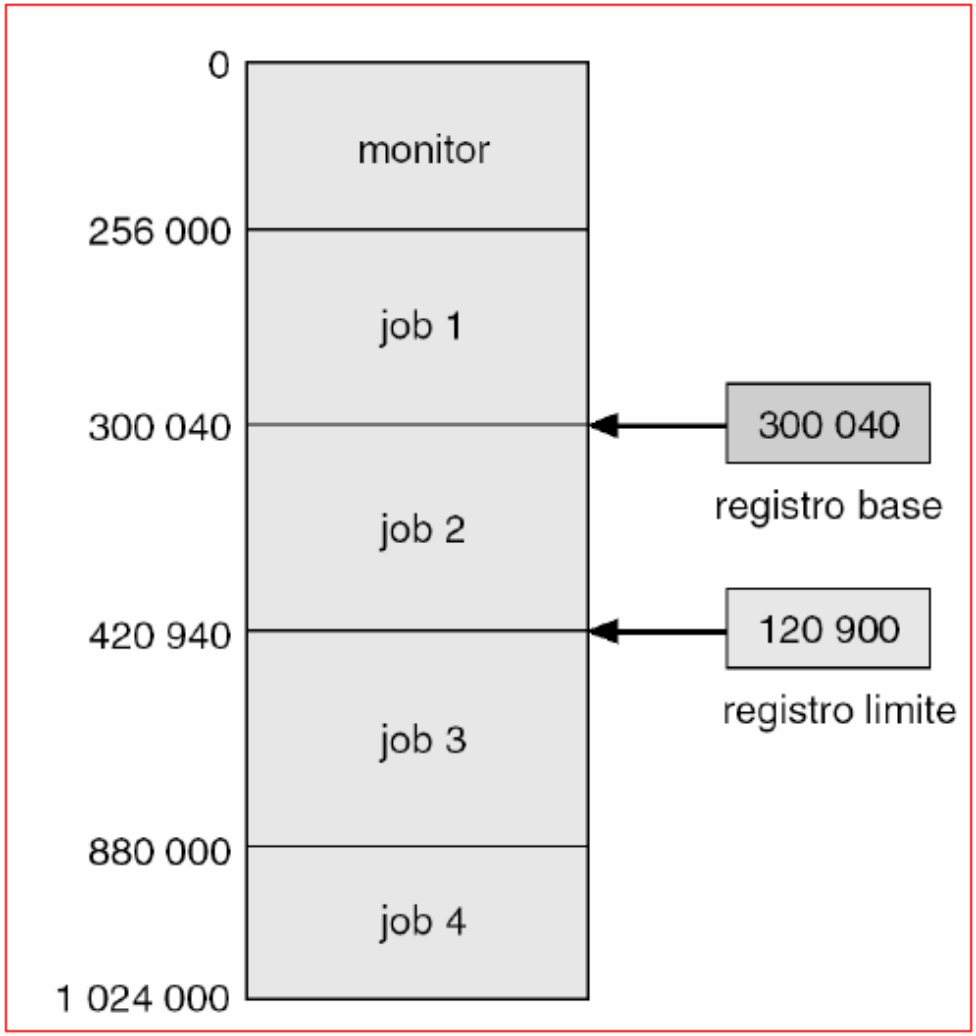
- È necessario assicurare la protezione della memoria quanto meno per il vettore di interrupt e le routine di gestione degli interrupt del SO.
- Per avere protezione della memoria, vengono aggiunti due registri che determinano il range degli indirizzi fisici a cui un programma può accedere:
  - ***Registro base*** – contiene il più piccolo indirizzo legale della memoria fisica.
  - ***Registro limite*** – contiene la dimensione della gamma degli indirizzi.

La memoria al di fuori del range definito è protetta.

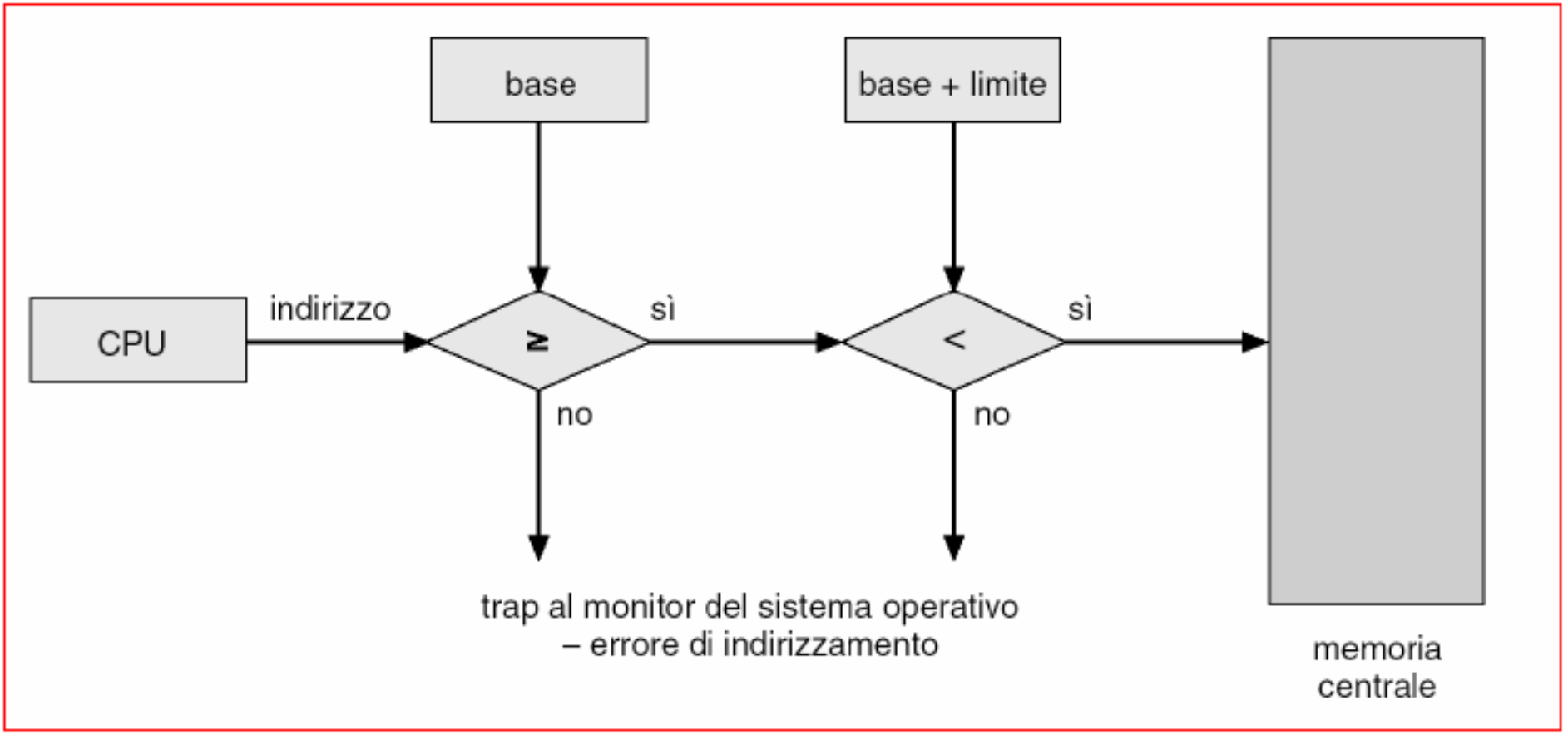
L'HW di CPU confronta ogni indirizzo generato in modalità utente con il contenuto dei registri.

I registri base/limite possono essere aggiornati solo dal SO mediante un'istruzione privilegiata speciale.

Registri base/limite



Protezione dell'indirizzo hardware



Essendo eseguito in modo monitor, il sistema operativo ha libero accesso a tutta la memoria, sia di sistema sia utente.

Le istruzioni di caricamento dei registri base e limite sono privilegiate.

Protezione della CPU

- Un programma utente potrebbe bloccare la CPU in un ciclo infinito. Per evitarlo si utilizza un ***timer*** (fisso o variabile).
- Il *timer* interrompe l'elaborazione dopo un periodo di tempo specificato per far sì che il sistema operativo mantenga il controllo.
  - Il contatore viene decrementato ad ogni ciclo di clock (comunemente 1/50 di secondo).
  - Quando il contatore raggiunge lo 0, genera un interrupt.
- All'azzerarsi del timer, il SO tratta l'interruzione valutando la possibilità di generare un *fatal error* o estendere il *time slice* riservato ad un processo utente.
- L'uso più comune del temporizzatore è quello che permette di realizzare il *time-sharing*. Al termine di ogni *time slice* scandita dal timer il SO esegue il cambio di contesto computazionale.
- Un ulteriore uso del temporizzatore consiste nel calcolare il tempo corrente rispetto ad un dato istante iniziale.
- Il caricamento del temporizzatore è un'istruzione privilegiata.