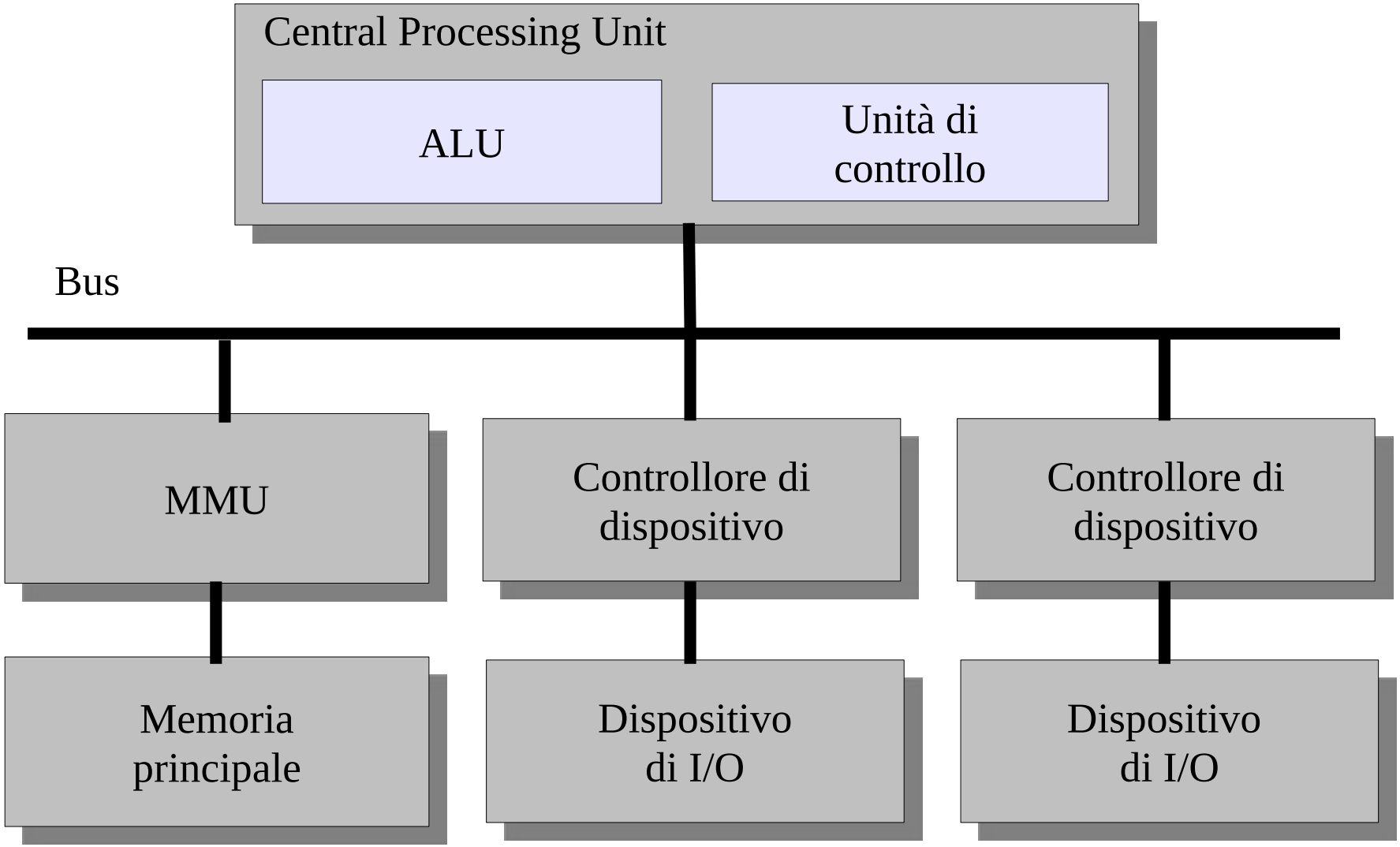
Sistemi Operativi

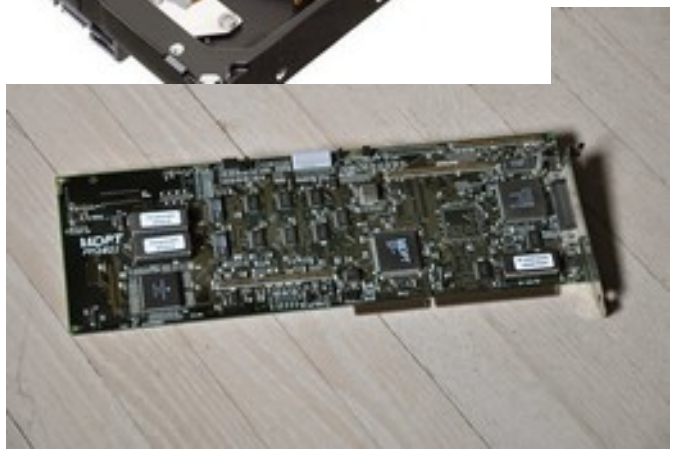
Modulo HW: richiami architettura

Renzo Davoli
Alberto Montresor

Architettura di Von Neumann



Nota



- **Device è hardware (disco/tastiera…)**
- **Device controller è la scheda elettronica di controllo del dispositivo: è hardware**
- **Device driver: è software. È il codice che viene fornito al sistema operativo per "parlare" col controller e quindi col device**
- **Memory Management Unit è hardware, è l'unità di controllo della memoria**
- **Memory Manager è software. È il codice del sistema operativo che configura e controlla le funzionalità della MMU.**

Architettura di Von Neumann

- **L'anno scorso avete visto nei dettagli:**
  - architettura dei processori
  - concetti base relativi alla memoria
  - linguaggio assembly
- **Nei prossimi lucidi vedremo alcuni concetti relativi a:**
  - gestione della comunicazione tra processore e dispositivi di I/O
  - concetto di interrupt
  - gerarchie di memoria

Interrupt

- **Definizione:**
  - Un meccanismo che permette l'interruzione del normale ciclo di esecuzione della CPU
- **Caratteristiche**
  - introdotti per aumentare l'efficienza di un sistema di calcolo
  - permettono ad un S.O. di "intervenire" durante l'esecuzione di un programma, allo scopo di gestire efficacemente le risorse del calcolatore
    - processore, memoria, dispositivi di I/O
  - possono essere sia hardware che software
  - possono essere mascherati (ritardati) se la CPU sta svolgendo compiti non interrompibili

Interrupt vs Trap

- **Interrupt Hardware**
  - Eventi hardware asincroni, non causati dal processo in esecuzione
  - Esempi:
    - *dispositivi di I/O*
      (per notifica di eventi quali il ***completamento di una operazione di I/O***)
    - *Clock* / interval timer
      (scadenza del quanto di tempo)
  - Gli Interrupt Hardware sono generati dai controller dei dispositivi
- **Interrupt Software (Trap)**
  - Causato dal programma
  - Esempi
    - errori come divisione per 0 o problemi di indirizzamento
    - richiesta di servizi di sistema (system call)

Nota

- **Gli interrupt sono segnali inviati al processore. (ai processori)**
- **Interrupt hardware: i controller dei dispositivi mandano (elevano) gli interrupt per indicare al processore che sono avvenuti eventi che necessitano di attenzione**
  - es. fine dell'operazione di I/O
- **Interrupt software: il codice in esecuzione ha generato un evento che necessita di attenzione**
  - es. divisione per zero, istruzione illegale, system call
- **Il sistema operativo per attivare una operazione di I/O comunica col controller usando il bus di sistema. L'interrupt avviene solo al termine dell'operazione.**

Gestione Interrupt - Panoramica

- **Cosa succede in seguito ad un interrupt**

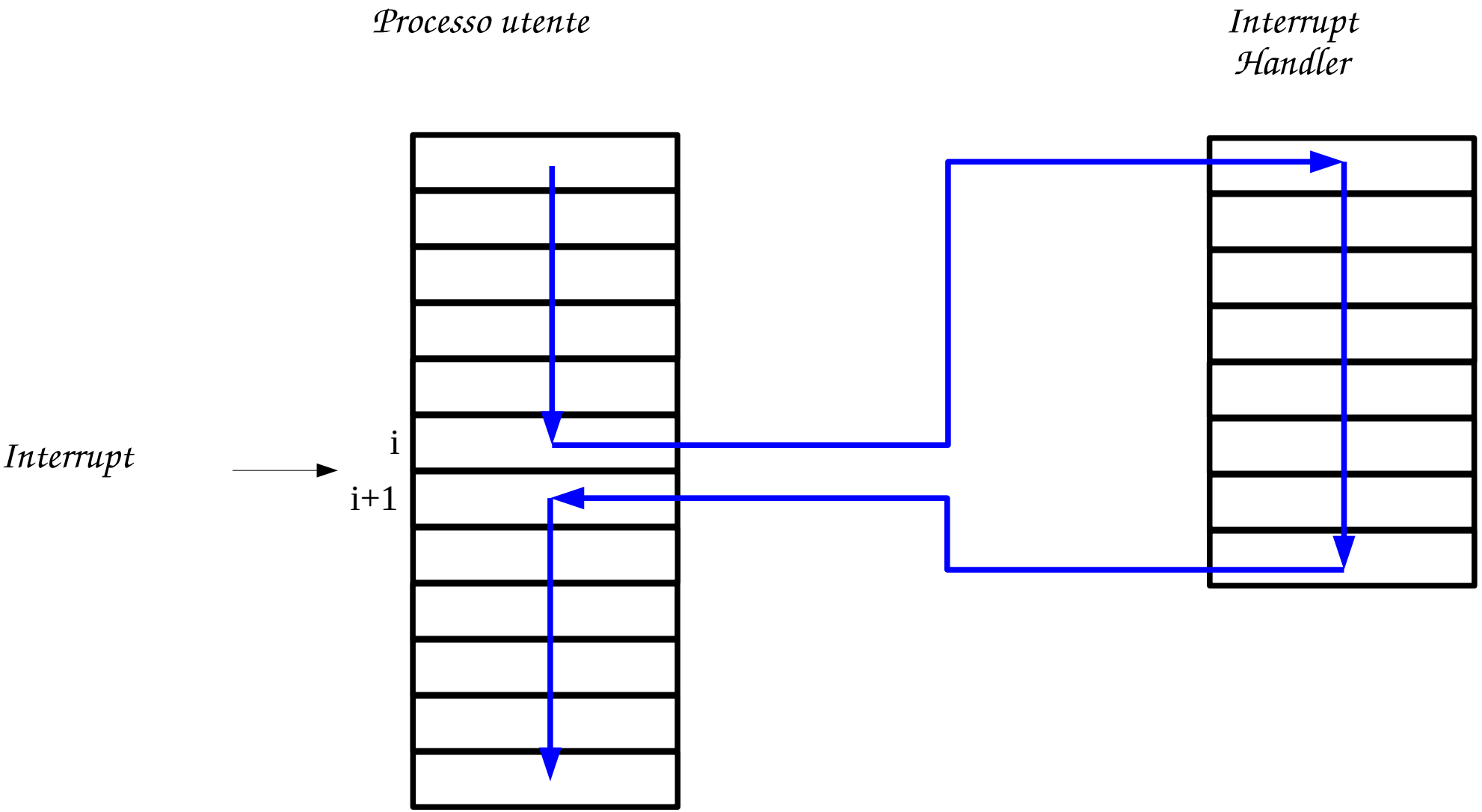
Processore

- Un segnale "interrupt request" viene spedito al processore
- Il processore
  - sospende le operazioni del processo corrente
  - salta ad un particolare indirizzo di memoria contenente la routine di gestione dell'interrupt (*interrupt handler*)

Sistema Operativo

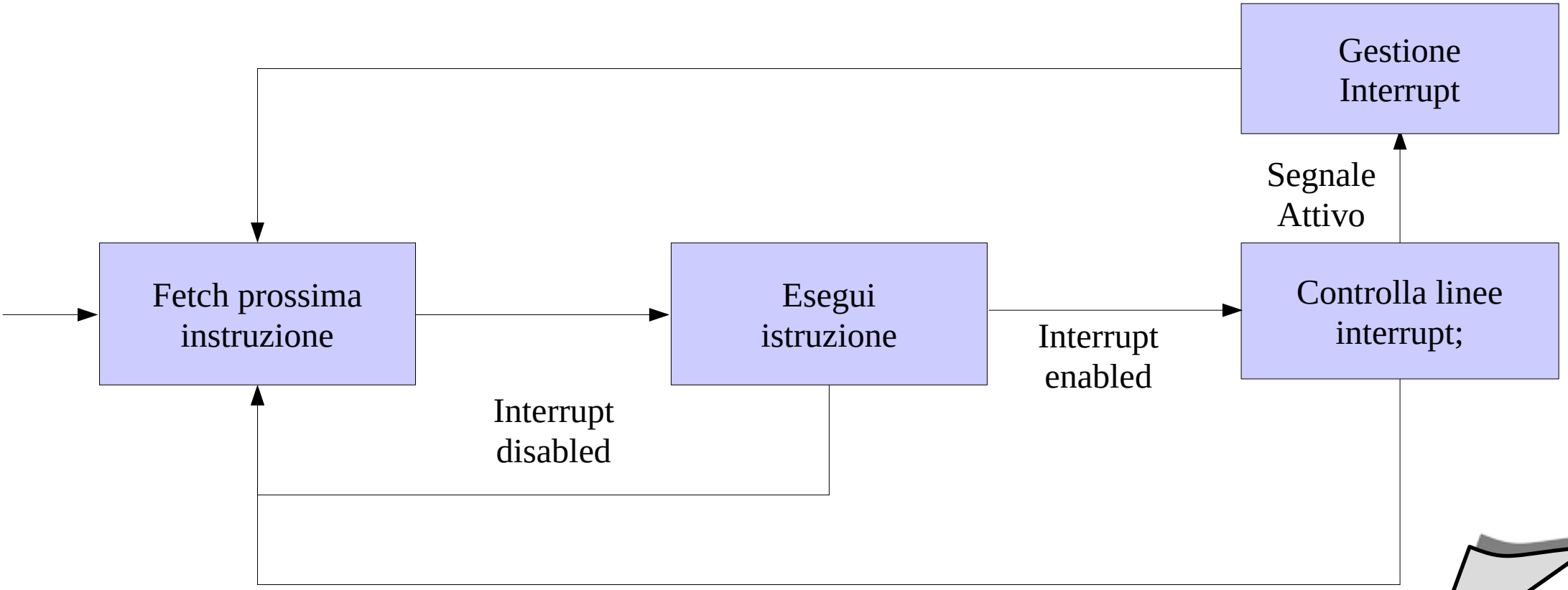
- L'interrupt handler
  - gestisce nel modo opportuno l'interrupt
  - ritorna il controllo al processo interrotto (o a un altro processo, nel caso di scheduling)
- Il processore riprende l'esecuzione del processo interrotto come se nulla fosse successo

Interrupt



Gestione Interrupt - Dettagli

1. **Un segnale di interrupt request viene spedito alla CPU**
2. **La CPU finisce l'esecuzione dell'istruzione corrente**
3. **La CPU verifica la presenza di un segnale di interrupt**



Gestione Interrupt - Dettagli

4. **Preparazione al trasferimento di controllo dal programma all'interrupt handler**
   - metodo 1: salvataggio dei registri "critici"
     - informazione minima richiesta: PC + registro di stato
   - metodo 2: scambio di stato
     - fotografia dello stato del processore
5. **Selezione dell'interrupt handler appropriato**
   - a seconda dell'architettura, vi può essere un singolo interrupt handler, uno per ogni tipo di interrupt o uno per dispositivo
   - la selezione avviene tramite l'*interrupt vector*
6. **Caricamento del PC con l'indirizzo iniziale dell'interrupt handler assegnato**

Gestione Interrupt - Dettagli

- **Nota:**
  - tutte le operazioni compiute fino a qui sono operazioni hardware
  - la modifica del PC corrisponde ad un salto al codice dell'interrupt handler
  - a questo punto:
    - il ciclo fetch-execute viene ripreso
    - il controllo è passato in mano all'interrupt handler

Gestione Interrupt - Dettagli

7. **Salvataggio dello stato del processore**
   - salvataggio delle informazioni critiche non salvate automaticamente dai meccanismi hardware di gestione interrupt
8. **Gestione dell'interrupt**
   - lettura delle informazioni di controllo proveniente dal dispositivo
   - eventualmente, spedizione di ulteriori informazioni al dispositivo stesso
9. **Ripristino dello stato del processore**
   - l'operazione inversa della numero 7
10. **Ritorno del controllo al processo in esecuzione (o ad un altro processo, se necessario)**

Sistemi operativi "Interrupt Driven"

- **I S.O. moderni sono detti "Interrupt Driven"**
  - il codice del S.O. entra in funzione come interrupt handler
  - sono gli interrupt (o i trap) che guidano l'avvicendamento dei processi

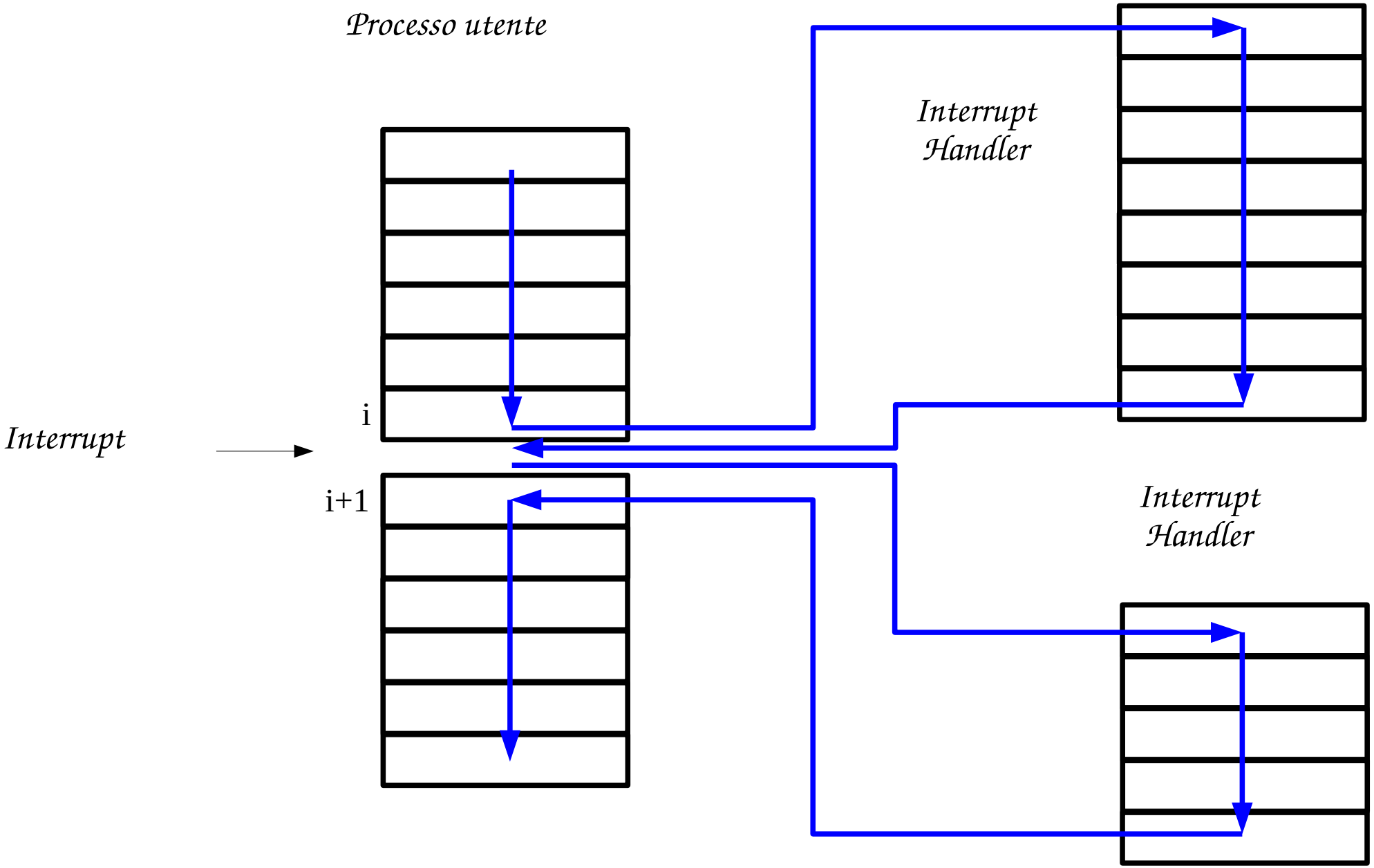
Interrupt Multipli

- **La discussione precedente prevedeva la presenza di un singolo interrupt**
- **Esiste la possibilità che avvengano interrupt multipli**
  - ad esempio, originati da dispositivi diversi
  - un interrupt può avvenire durante la gestione di un interrupt precedente
- **Due approcci possibili:**
  - disabilitazione degli interrupt
  - interrupt annidati

Interrupt Multipli - Disabilitazione Interrupt

- **Disabilitazione degli interrupt**
  - durante l'esecuzione di un interrupt handler
    - ulteriori segnali di interrupt vengono ignorati
    - i segnali corrispondenti restano pendenti
  - gli interrupt vengono riabilitati prima di riattivare il processo interrotto
  - il processore verifica quindi se vi sono ulteriori interrupt, e in caso attiva l'interrupt handler corrispondente
- **Vantaggi e svantaggi**
  - approccio semplice; interrupt gestiti in modo sequenziale
  - non tiene conto di gestioni "time-critical"

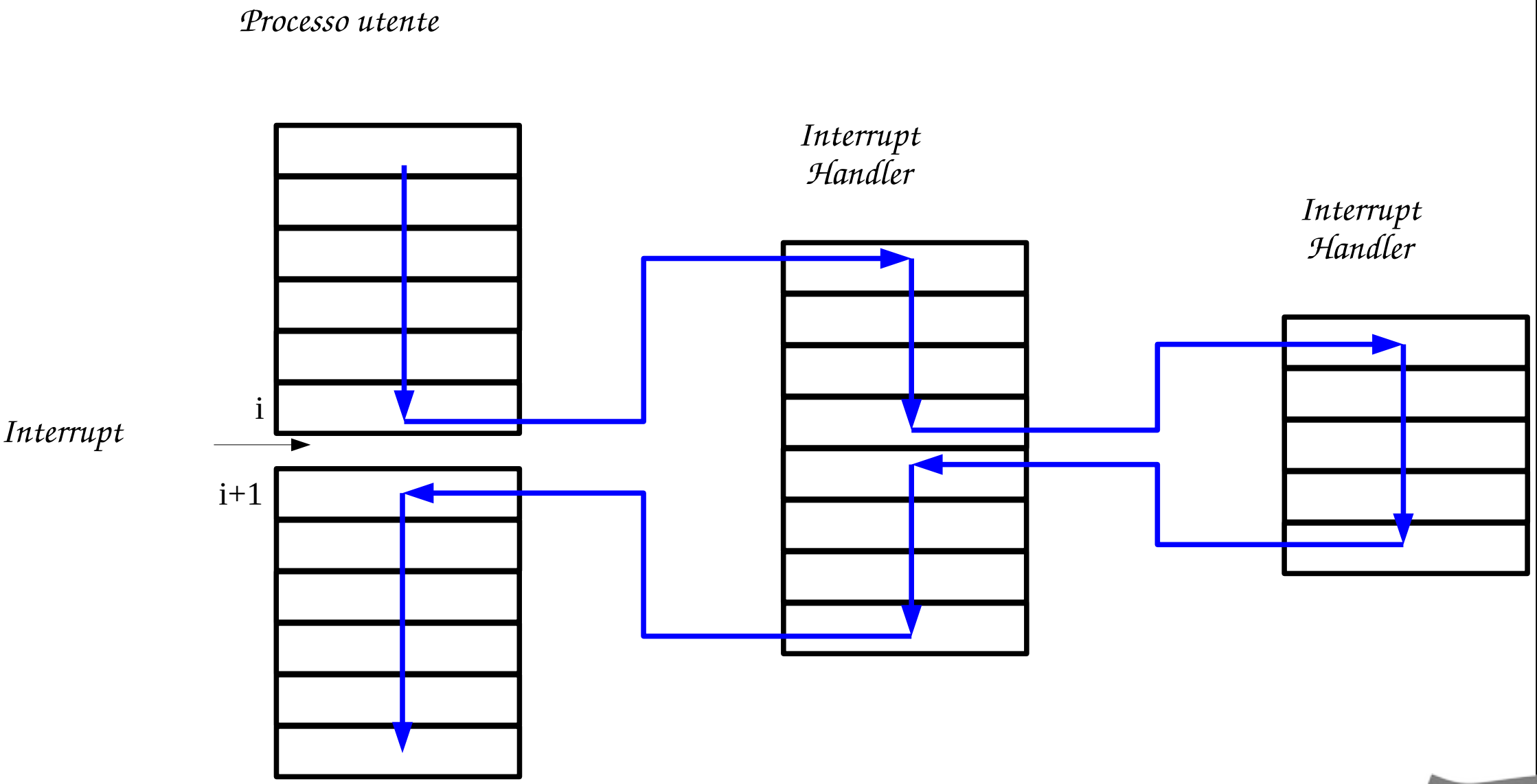
Interrupt Multipli - Disabilitazione Interrupt



Interrupt Multipli - Interrupt Annidati

- **Interrupt annidati**
  - è possibile definire priorità diverse per gli interrupt
  - un interrupt di priorità inferiore può essere interrotto da un interrupt di priorità superiore
  - è necessario prevedere un meccanismo di salvataggio e ripristino dell'esecuzione adeguato
- **Vantaggi e svantaggi**
  - dispositivi veloci possono essere serviti prima (es. schede di rete)
  - approccio più complesso
  - Occorrono stack separati

Interrupt Multipli - Interrupt Annidati



Comunicazione fra processore e dispositivi di I/O

- **Comunicazione tra processore e dispositivi di I/O**
  - il controllore governa il dialogo con il dispositivo fisico
- **Esempio**
  - il controller di un disco accetta una richiesta per volta
  - l'accodamento delle richieste in attesa è a carico del S.O.
- **Due modalità possibili:**
  - *Programmed I/O*
  - *Interrupt-Driven I/O*

Programmed I/O (obsoleto)

- **Operazione di input**
  - la *CPU carica (tramite il bus)* i parametri della richiesta di input in appositi registri del controller (registri comando)
  - il dispositivo
    - esegue la richiesta
    - il risultato dell'operazione viene memorizzato in un apposito buffer locale sul controller
    - il completamento dell'operazione viene segnalato attraverso appositi registri di status
  - *il S.O. attende* (busy waiting/polling) *che il comando sia completato verificando periodicamente il contenuto del registro di stato*
  - *infine, la CPU copia i dati dal buffer locale del controller alla memoria*

Interrupt-Driven I/O

- **Operazione di input**
  - la *CPU carica (tramite il bus)* i parametri della richiesta di input in appositi registri del controller (registri comando)
  - *il S.O. sospende l'esecuzione del processo che ha eseguito l'operazione di input ed esegue un altro processo*
  - il dispositivo
    - esegue la richiesta
    - il risultato dell'operazione viene memorizzato in un apposito buffer locale sul controller
    - il completamento dell'operazione viene segnalato attraverso interrupt
  - *al ricevimento dell'interrupt, la CPU copia i dati dal buffer locale del controller alla memoria*
  - NB: l'interrupt segnala la ***fine*** dell'operazione di I/O

Programmed I/O e Interrupt-Driven I/O

- **Nel caso di operazioni di output**
  - il procedimento è similare:
    - i dati vengono copiati dalla memoria ai buffer locali
    - questa operazione viene eseguita prima di caricare i parametri della richiesta nei registri di comando dei dispositivi
- **Svantaggi degli approcci precedenti**
  - il processore spreca parte del suo tempo nella gestione del trasferimento dei dati
  - la velocità di trasferimento è limitata dalla velocità con cui il processore riesce a gestire il servizio

Direct Memory Access (DMA)

- **Il S.O.**
  - *attiva l'operazione di I/O specificando l'indirizzo in memoria di destinazione (Input) o di provenienza (Output) dei dati*
  - Il controller del dispositivo prende (output) o pone (input) i dati per l'operazione di I/O direttamente dalla memoria centrale.
  - *l'interrupt specifica solamente la conclusione dell'operazione di I/O*
- **Vantaggi e svantaggi**
  - c'è contesa nell'accesso al bus
  - device driver più semplici
  - efficace perché la CPU non accede al bus ad ogni ciclo di clock

Memoria

- **Memoria centrale (RAM)**
  - assieme ai registri, l'unico spazio di memorizzazione che può essere acceduto *direttamente* dal processore
  - accesso tramite istruzioni LOAD/STORE
  - Volatile
  - Nei sistemi moderni accesso tramite MMU (trasforma indirizzi logici in indirizzi fisici)
- **Memoria ROM**

Memory Mapped I/O

- **Un dispositivo è completamente indirizzabile tramite bus**
  - i datii gestiti dal dispositivo vengono mappati su un insieme di indirizzi direttamente accessibili tramite il bus di sistema
  - una lettura o scrittura su questi indirizzi causa il trasferimento di dati da o verso il dispositivo
- **Esempio:**
  - video grafico nei PC
- **Vantaggi e svantaggi**
  - gestione molto semplice e lineare
  - necessita di tecniche di *sincronizzazione di accesso*

Dischi

- **Caratteristiche**
  - dispositivi che consentono la memorizzazione non volatile dei dati
  - accesso diretto (random, i.e. non sequenziale)
  - per individuare un dato sul disco (dal punto di vista fisico) occorre indirizzarlo in termini di cilindro, settore, testina

Dischi

- **Operazioni gestite dal controller**
  - READ (head, sector)
  - WRITE(head, sector)
  - SEEK(cylinder)
- **L'operazione di seek**
  - corrisponde allo spostamento fisico del pettine di testine da un cilindro ad un altro ed è normalmente la più costosa
- **L'operazione di read e write**
  - prevedono l'attesa che il disco ruoti fino a quando il settore richiesto raggiunge la testina

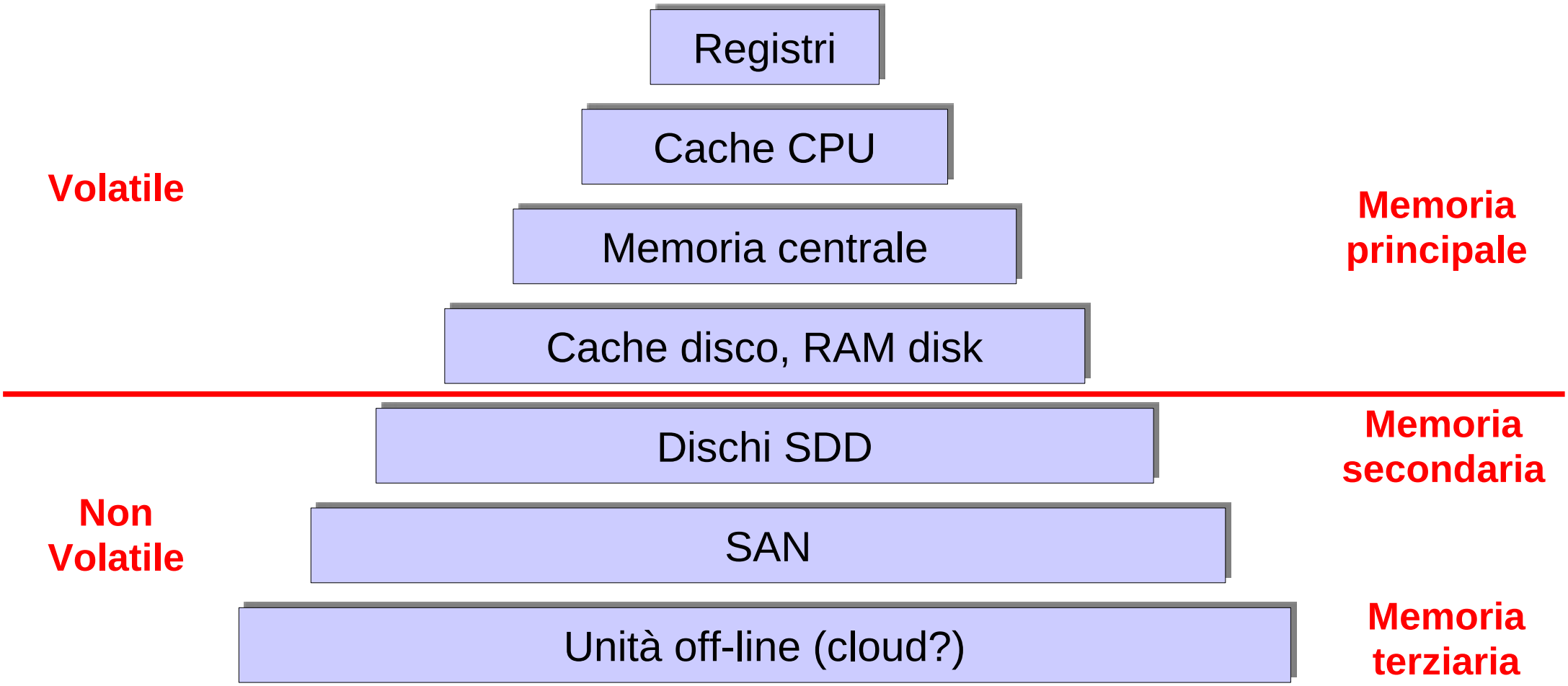
SSD - SDcard

- **Caratteristiche**
  - dispositivi per la memorizzazione non volatile dei dati
  - Hanno un numero di cicli di scrittura limitato
  - Si leggono a blocchi
  - Si scrivono a banchi (numerosi blocchi)

Gerarchia di memoria

- **Trade off**
  - Quantità
  - Velocità
  - Costo
- **Limitazioni**
  - tempo di accesso più veloce, costo maggiore
  - maggiore capacità, costo minore (per bit)
  - maggiore capacità, tempo di accesso maggiore
- **Soluzione:**
  - utilizzare una gerarchia di memoria

Gerarchia di memoria



Cache

- **Un meccanismo di caching**
  - consiste nel memorizzare *parzialmente* i dati di una memoria in una seconda più costosa ma più efficiente
  - se il numero di occorrenze in cui il dato viene trovato nella cache (memoria veloce) è statisticamente rilevante rispetto al numero totale degli accessi, la cache fornisce un notevole aumento di prestazione
- **E' un concetto che si applica a diversi livelli:**
  - cache della memoria principale (DRAM) tramite memoria bipolare
  - cache di disco in memoria
  - cache di file system remoti tramite file system locali

Cache

- **Meccanismi di caching**
  - hardware
    - ad es. cache CPU; *politiche non modificabili dal S.O.*
  - software
    - ad es. cache disco; *politiche sotto controllo del S.O.*
- **Problemi da considerare nel S.O.**
  - algoritmo di replacement
    - la cache ha dimensione limitata; bisogna scegliere un algoritmo che garantisca il maggior numero di accessi in cache
  - coerenza
    - gli stessi dati possono apparire a diversi livelli della struttura di memoria

Protezione Hardware

- **I sistemi multiprogrammati e multiutente richiedono la presenza di *meccanismi di protezione***
  - bisogna evitare che processi concorrenti generino interferenze non previste...

    ma soprattutto:
  - *bisogna evitare che processi utente interferiscano con il sistema operativo*
- **Riflessione**
  - i meccanismi di protezione possono essere realizzati totalmente in software, oppure abbiamo bisogno di meccanismi hardware dedicati?

Protezione HW: Modo utente / Modo kernel

- **Modalità kernel / supervisore / privilegiata / ring 0:**
  - i processi in questa modalità hanno accesso a tutte le istruzioni, incluse quelle *privilegiate*, che permettono di gestire totalmente il sistema
- **Modalità utente**
  - i processi in modalità utente non hanno accesso alle istruzioni privilegiate
- **"Mode bit" nello status register per distinguere fra modalità utente e modalità supervisore**
- **Esempio:**
  - le istruzioni per disabilitare gli interrupt è privilegiata

Protezione HW: Modo utente / Modo kernel

- **Come funziona**
  - alla partenza, il processore è in modalità kernel
  - viene caricato il sistema operativo (*bootstrap*) e si inizia ad eseguirlo
  - quando passa il controllo ad un processo utente, il S.O. cambia il valore del mode bit e il processore passa in modalità utente
  - tutte le volte che avviene un interrupt, l'hardware passa da modalità utente a modalità kernel

Protezione HW: Protezione I/O

- **Le istruzioni di I/O devono essere considerate privilegiate**
  - il S.O. dovrà fornire agli utenti primitive e servizi per accedere all'I/O
  - tutte le richieste di I/O passano attraverso codice del S.O. e possono essere controllate preventivamente
- **Esempio:**
  - accesso al dispositivo di memoria secondaria che ospita un file system
  - vogliamo evitare che un qualunque processo possa accedere al dispositivo modificando (o corrompendo) il file system stesso

Protezione HW: Protezione Memoria

- **La protezione non è completa se non proteggiamo anche la memoria**
- **Altrimenti, i processi utente potrebbero:**
  - modificare il codice o i dati di altri processi utenti
  - modificare il codice o i dati del sistema operativo
  - modificare l'interrupt vector, inserendo i propri gestori degli interrupt
- **La protezione avviene tramite la Memory Management Unit (MMU)**

Protezione HW: MMU

- **Traduzione indirizzi logici in indirizzi fisici**
  - ogni indirizzo generato dal processore corrisponde ad un indirizzo logico
  - l'indirizzo logico viene trasformato in un indirizzo fisico a tempo di esecuzione dal meccanismo di MMU
  - un indirizzo viene protetto se non può mai essere generato dal meccanismo di traduzione

Protezione HW - System call

- **Problema**
  - poiché le istruzioni di I/O sono privilegiate, possono essere eseguite unicamente dal S.O.
  - com'è possibile per i processi utenti eseguire operazioni di I/O?
- **Soluzione**
  - i processi utenti devono fare richieste esplicite di I/O al S.O.
  - meccanismo delle *system call*, ovvero trap generate da istruzioni specifiche

Protezione HW - System call

Codice utente

```
li $a0, 10
li $v0, 1
syscall
```

```
... altro
codice...
```

Interrupt handler

```
...salvataggio registri ...
...gestione interrupt...
...operazioni di I/O...
...ripristino registri...
rfe/eret
// return from exception
```

Interrupt vector