Introduzione

Table of contents

Sistema operativo

L'obbiettivo del sistema operativo è controllare e coordinare l'utilizzo dell'**hardware**, tra le diverse applicazioni che vengono eseguite, e gli utenti.

Programmi Applicativi

Definiscono i modi in cui vengono usate le risorse del sistema, la memoria, la CPU, spazio sul disco, ecc. e servono a **risolvere** i vari problemi dell'utente. Gli utenti, infatti, faranno diversi usi di queste applicazioni.

Utenti

L'utente di un sistema di calcolo potrebbe essere una persona, una macchina oppure un altro computer. Nell'ultimo caso, infatti, potrebbe essere un altro computer che si collega in remoto.

Che cosa fanno i sistemi operativi?

Il comportamento di un sistema operativo dipende dal **punto di vista** da cui lo si osserva.

- Dal punto di vista **degli utenti** dovrebbe avere delle **buone performance** e dovrebbe essere di semplice utilizzo. L'utente, infatti, non ha interesse nel sapere che il SO sia **ben ottimizzato** , ma vuole solo che esso "funzioni adeguatamente".
- Nel momento in cui abbiamo un computer **condiviso** , a cui vi si collegano diversi utenti (anche simultaneamente), esso deve gestire ogni connessione efficientemente, in modo che ogni utente riscontri delle buone performance.
- Gli utenti che usano delle **workstation** avranno bisogno di accedere a risorse **condivise** salvate su server, ed anche in questo caso il SO deve gestire al meglio

le operazioni.

- I dispositivi **mobili** sono poveri di risorse, come batteria e capacità di calcolo, quindi il SO deve gestire in modo ottimizzato ogni operazione, in modo da ridurre l'utilizzo di CPU e batteria. Questi dispositivi hanno interfacce di I/O particolari, come **touch screen** , **riconoscimento vocale** .
- Alcuni computer hanno un'interfaccia molto ridotta o praticamente assente, come i sistemi **embedded** , utilizzati in automobili o elettrodomestici. Questo tipo di sistemi operano senza la supervisione di un utente.

Definire i sistemi operativi

Il termine **sistema operativo** può ricoprire molti ruoli.

Questo perché si possono avere molti sistemi diversi, e di conseguenza possiamo avere molti utilizzi diversi dei SO. Possiamo trovarli ovunque, dagli elettrodomestici meno costosi alle navicelle spaziali.

Non esiste una vera e propria definizione di SO, ma il concetto è che i SO sono talmente tanto diversi tra loro, che quello che viene "spacciato" per sistema operativo, è la definizione stessa di SO.

Il Kernel

Nel momento in cui ho un dispositivo in funzione, c'è un minimo di software, indipendente dai programmi applicativi, che è sempre in esecuzione (quindi in memoria), pronto ad eseguire delle operazioni non appena si verifica un evento che ne richieda l'esecuzione.

Il Sistema operativo non è solo **kernel**, ma è l'unione di tanti piccoli "pezzi", come i **programmi di sistema**, che sono dei programmi **preinstallati** nel sistema operativo, oppure **programmi applicativi**, ovvero tutti i programmi che **non sono associati** al sistema operativo.

Middlewear

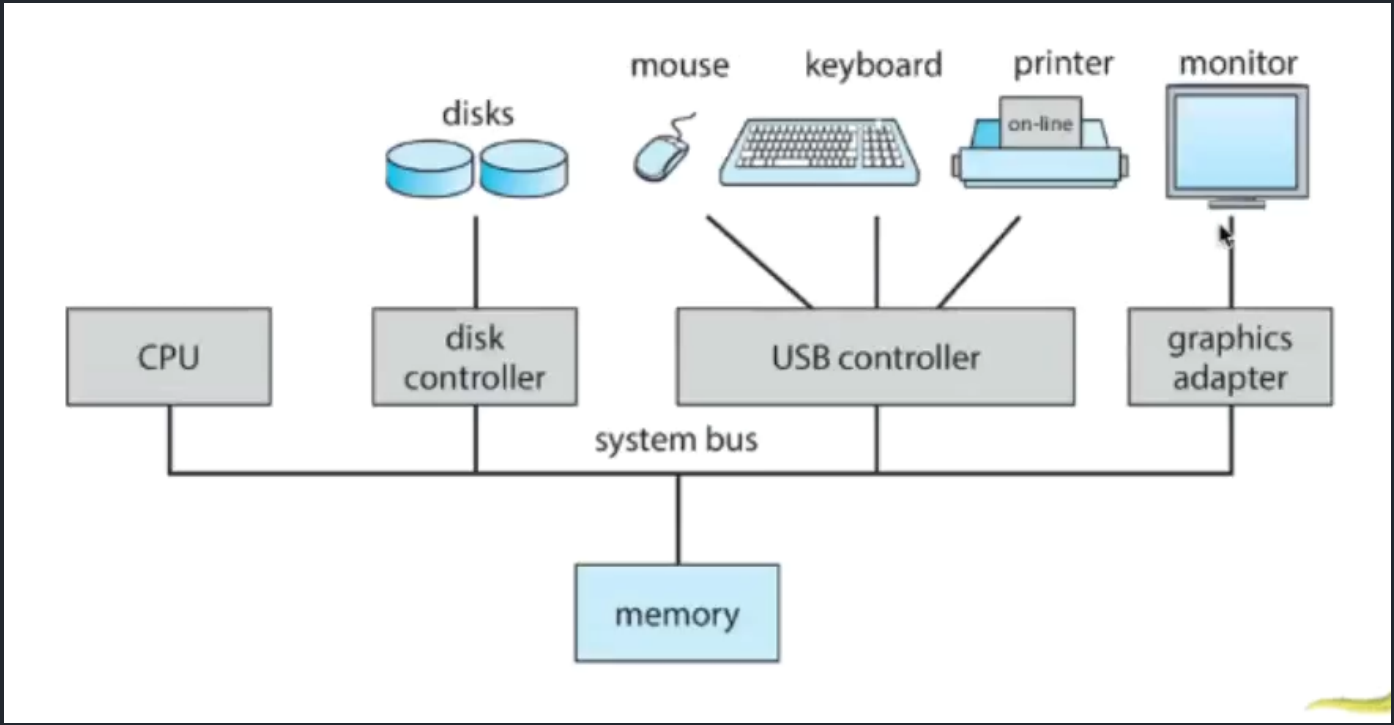
Nei sistemi operativi moderni ha preso l'uso di software chiamati **middlewear**, ovvero una via di mezzo tra **software** ed **hardware**, e sono dei software che forniscono **servizi addizionali**, per realizzare database, applicazioni multimediali ecc.

Organizzazione dei sistemi operativi

Nei computers una o più CPU, i **controller di device** (che sono delle unità **indipendenti!**), sono connessi attraverso un **bus** comune, che fornisce accesso alla memoria condivisa.

I **controllers** sono molto importanti, tutte le unità I/O non sono infatti connesse direttamente alla coppia **CPU - Memoria**, ma nel mezzo sono appunto posti i **controllers**, che sono visibili alla CPU.

Quindi, se la CPU volesse scrivere sul disco, essa scriverebbe sui controllers, che a loro volta scriverebbero sul disco.



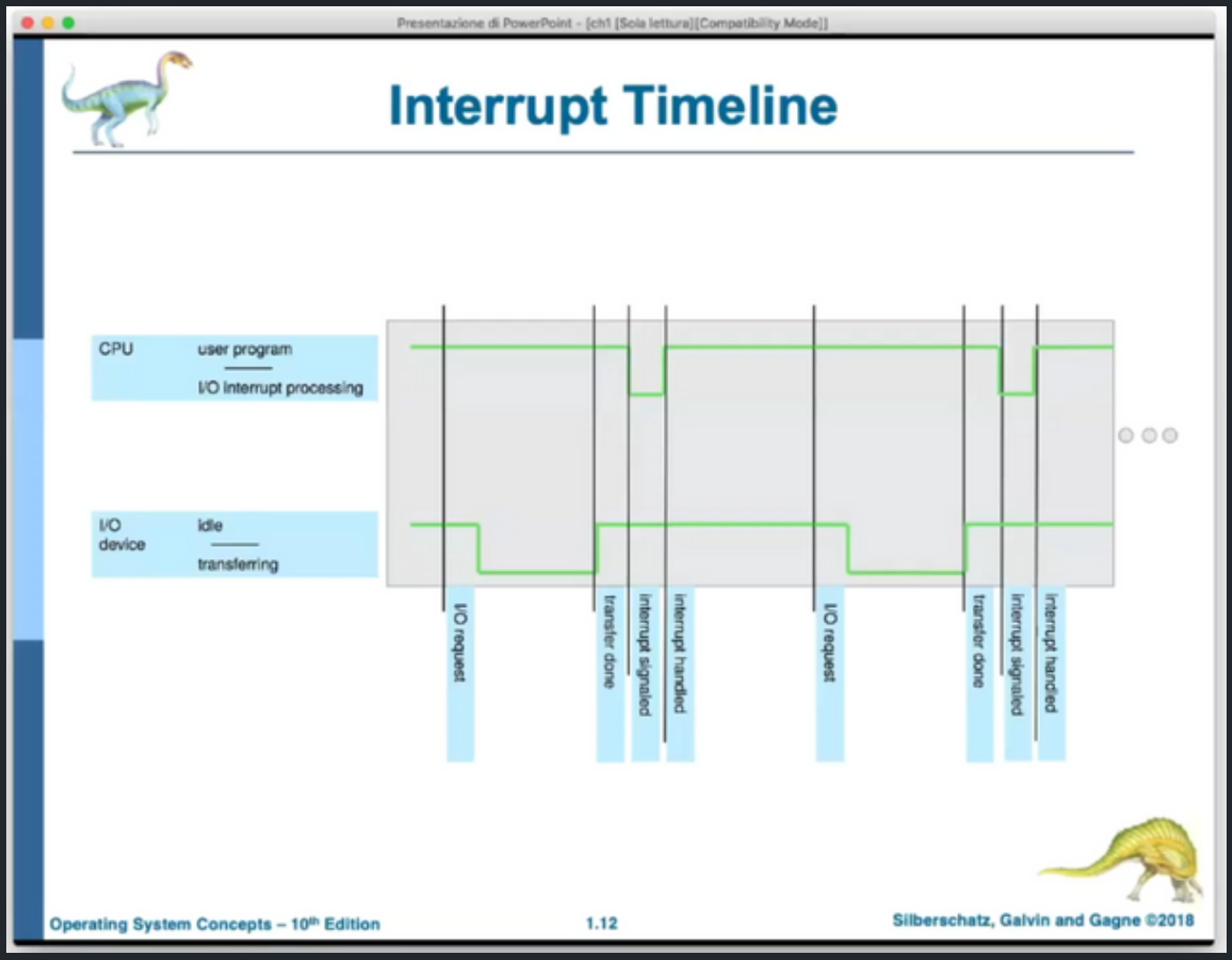
🏁 1:15 lezione 04-29

Interruzioni

Le interruzioni sono un caso particolare delle eccezioni; un'eccezione è un qualsiasi **fenomeno** che interrompe il programma in esecuzione, e sposta il controllo su di un'altra parte del SO.

Una **Trap** o **eccezione** è un tipo di interruzione particolare generata dal software, causata o da un errore o da una richiesta dell'utente. Quando avviene, il processore **salva lo stato** del programma in esecuzione, ed effettua un **jump** ad un'altra locazione di memoria. Il metodo più utilizzato nei processori moderni è quello dell'interrupt vettorizzato: ogni causa di interruzione ha un proprio indirizzo a cui effettuare lo jump (indirizzi salvati in una tabella), questo permette di evitare di saltare ad un punto comune e solo poi determinare cosa è successo, ma saltare direttamente alla routine di quella particolare interruzione.

Un sistema operativo è **interrupt driven**, ovvero è guidato dalle interruzioni. Si può pensare al sistema operativo come un software che si mette in esecuzione nel momento in cui si verificano eccezioni o interruzioni.



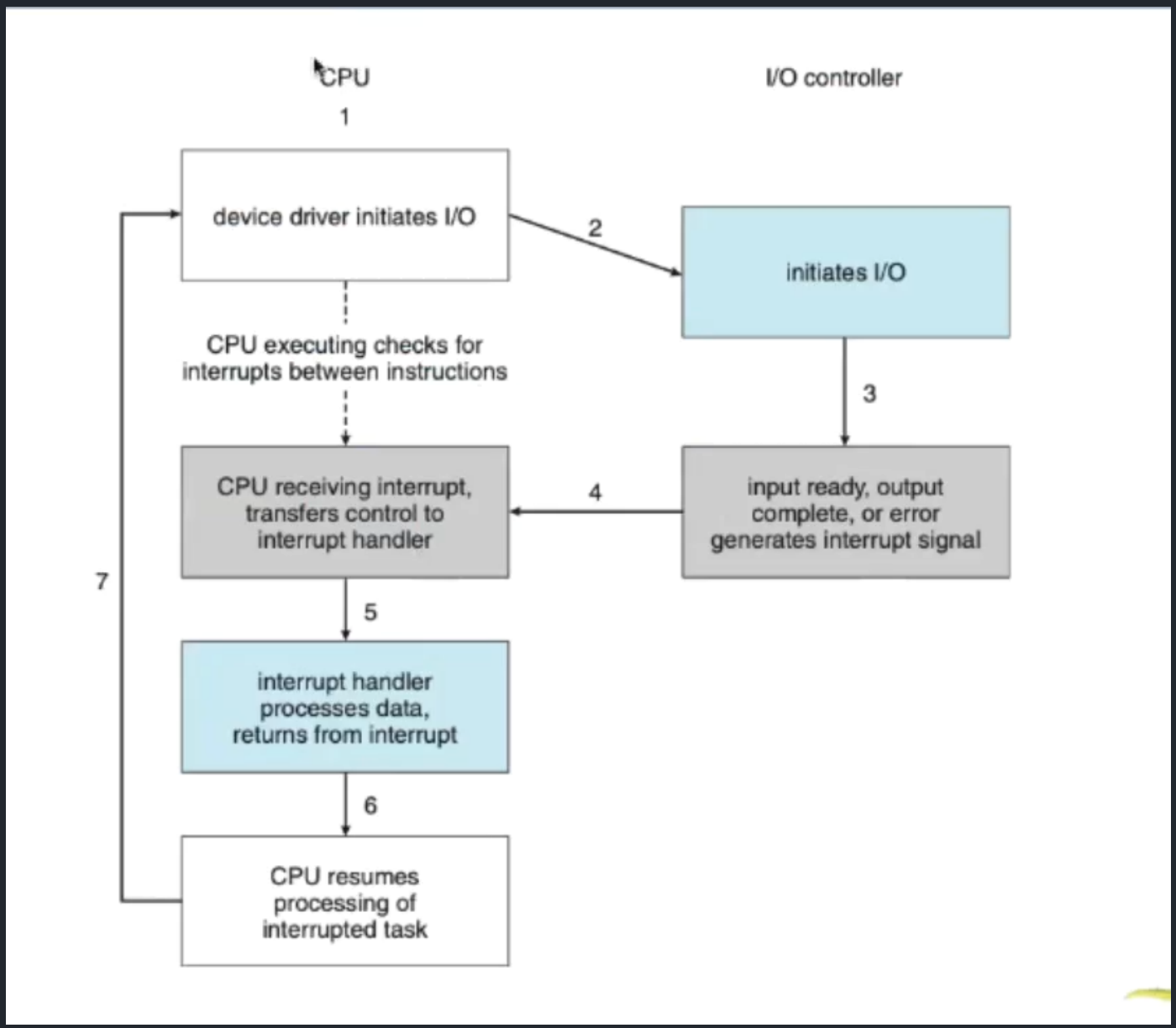
In questa immagine si vede come dopo ogni completamento di operazione I/O, viene richiesto l'intervento del sistema operativo attraverso un'interruzione. Dopo che il SO ha gestito l'interruzione, il programma utente riprende l'esecuzione.

Interrupt Handling

Il sistema operativo preserva lo stato della CPU salvando i registri ed il **program counter**, in modo da poter riprendere l'esecuzione del programma. L'interruzione può essere eseguita mediante un salto ad un'unica locazione di memoria, dove è presente un software che determina il **richiedente** dell'interruzione "chiedendo"

ad ogni elemento, oppure tramite un sistema **vettorizzato**, ovvero un metodo più moderno e performante.

In questo sistema di **interrupt vettorizzato** è presente una lista di punti di ingresso delle varie routine e si salta direttamente alla routine interessata. In questo modo si salta tutta la fase di "investigazione" per capire chi ha generato l'interruzione.



Struttura dell'I/O

Nel momento in cui parte un'operazione di I/O la CPU resta bloccata nel frattempo che l'operazione venga compiuta, oppure fa altro?

Il metodo più usato è quello di **sbloccare** la CPU in maniera tale che possa fare altro. La CPU gestisce quindi l'inizio del trasferimento I/O, dopodiché esegue altre operazioni, come eseguire un programma utente, e questo significa che possono verificarsi anche altre richieste da parte di altri controller. In fine il programma utente può fermarsi, ad esempio per richiedere dei dati, e quindi è presente un'opportuna **system call** che permette di attendere il completamento dell'operazione di I/O.

Struttura dello Storage

Memoria principale (RAM)

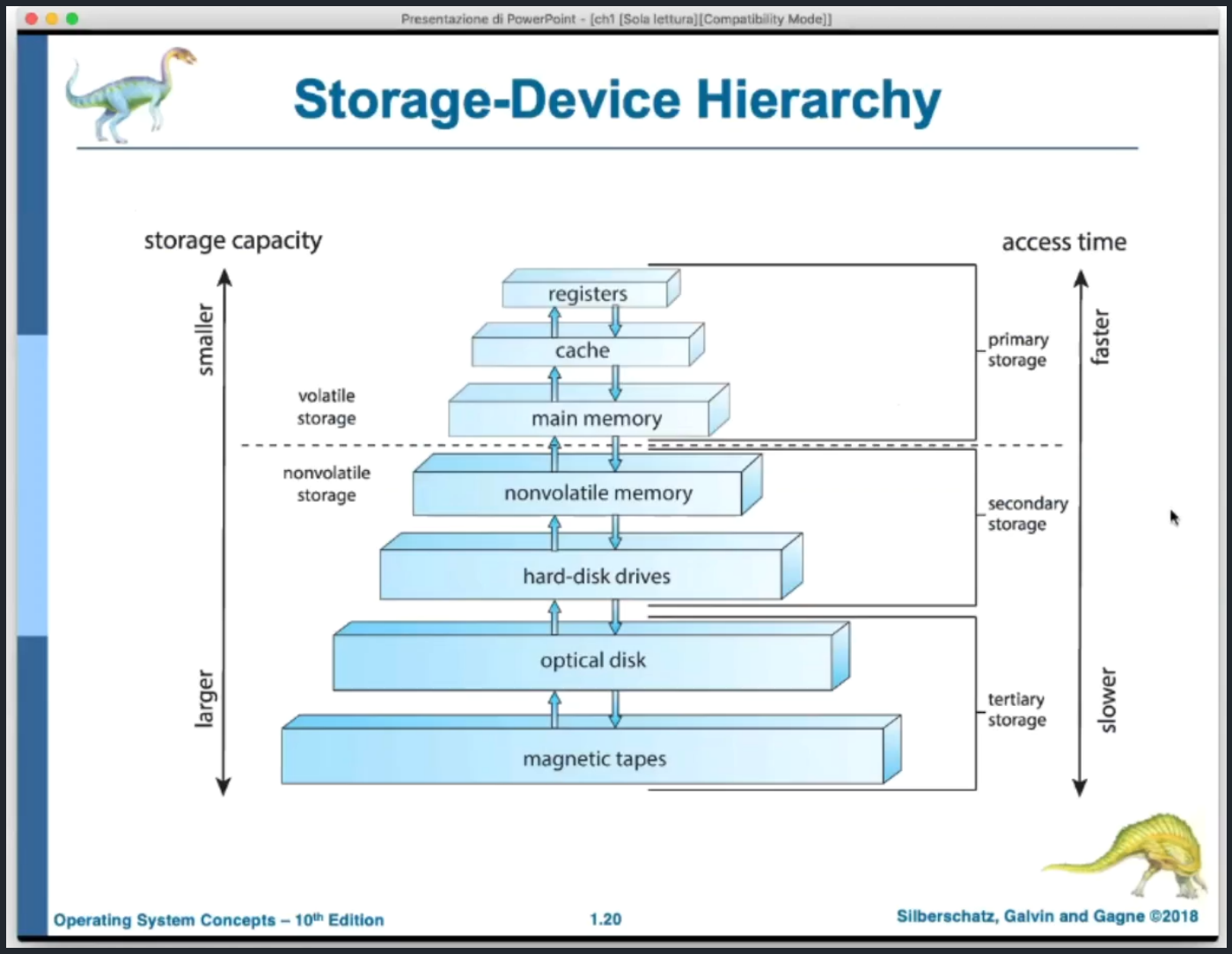
E` presente una memoria principale, ovvero l'unica memoria di grande capienza che la CPU può accedere **direttamente**, e questa memoria è composta da:

- **Random access** : memoria che consente un accesso casuale, ovvero una memoria in cui si può accedere a qualsiasi locazione di memoria negli **stessi tempi** .
- **Tipicamente Volatile** : questo tipo di memoria è volatile, quindi una volta che c'è un'interruzione di corrente, e quindi si spegne la macchina, la memoria viene azzerata.

Memoria secondaria (dischi)

Per memoria secondaria si intende tutto ciò che è sotto forma di disco (o SSD, memorie flash ecc.),.

Gerarchia della memoria



Direct Memory Access Structure

Il DMA viene usato esclusivamente per devices ad **alta velocità**, in questi casi il controller permette l'accesso alla memoria direttamente, andando ad interrompere la CPU **solo alla fine di un trasferimento**. Nel caso del DMA viene trasferito un blocco di dati intero, anche da 64KB, senza interrupt. Questo è conveniente nel momento in cui ho una grande quantità di dati da dover trasferire

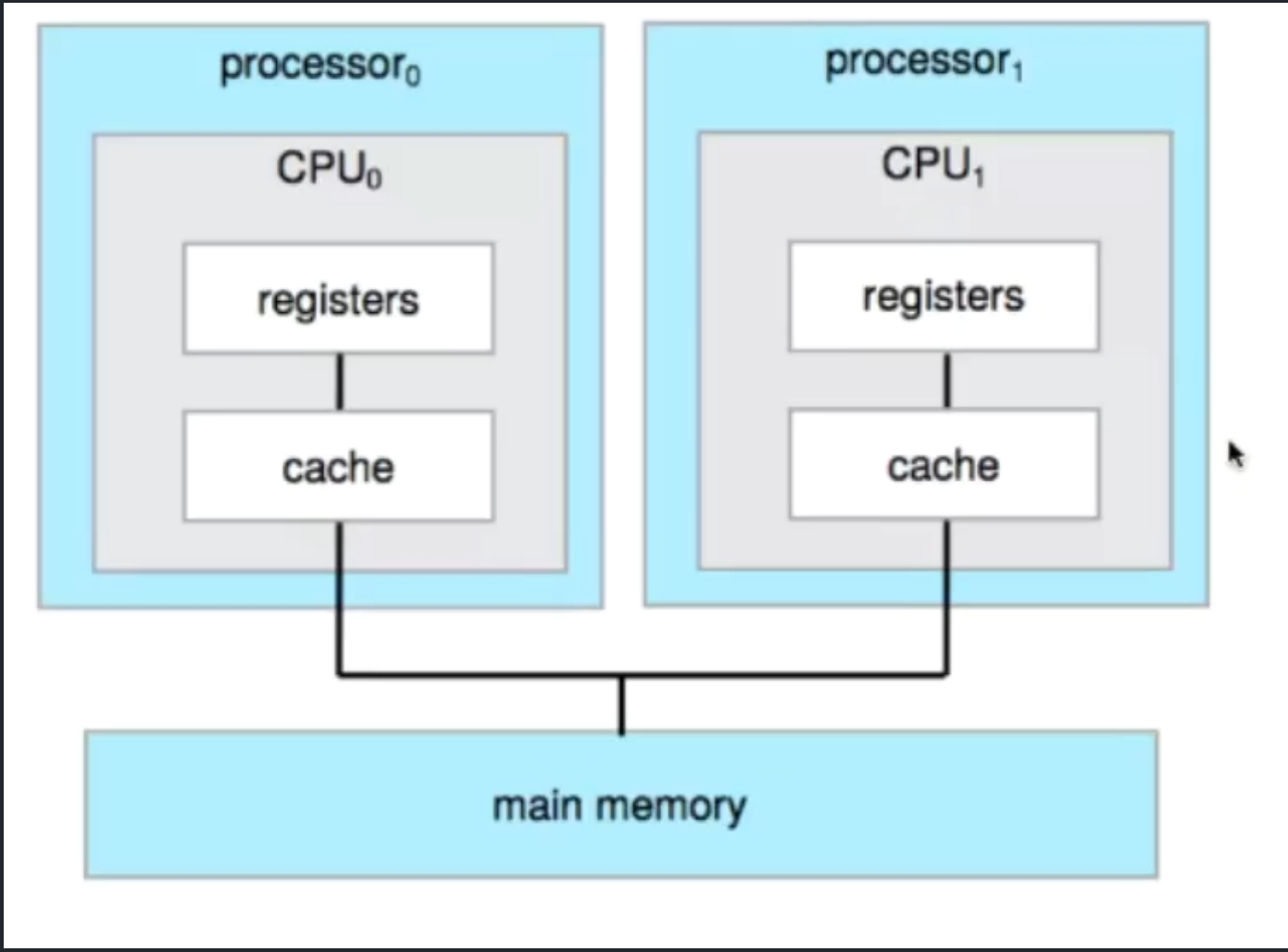
Architettura dei Computer-System

Ci sono sistemi che sono **General purpose**, ovvero sistemi che hanno più obbiettivi e destinati a diverse operazioni, e sistemi **Special purpose**, ovvero sistemi destinati ad un obbiettivo specifico.

I sistemi multi-processore stanno diventando sempre più diffusi e vengono detti **paralleli** o ad **accoppiamento stretto**, ovvero dove diverse CPU sono connessi alla stessa **memoria**. Questo tipo di computers servono ad avere, ovviamente, delle prestazioni migliori e ad avere una maggiore affidabilità (per via del maggior numero di CPU).

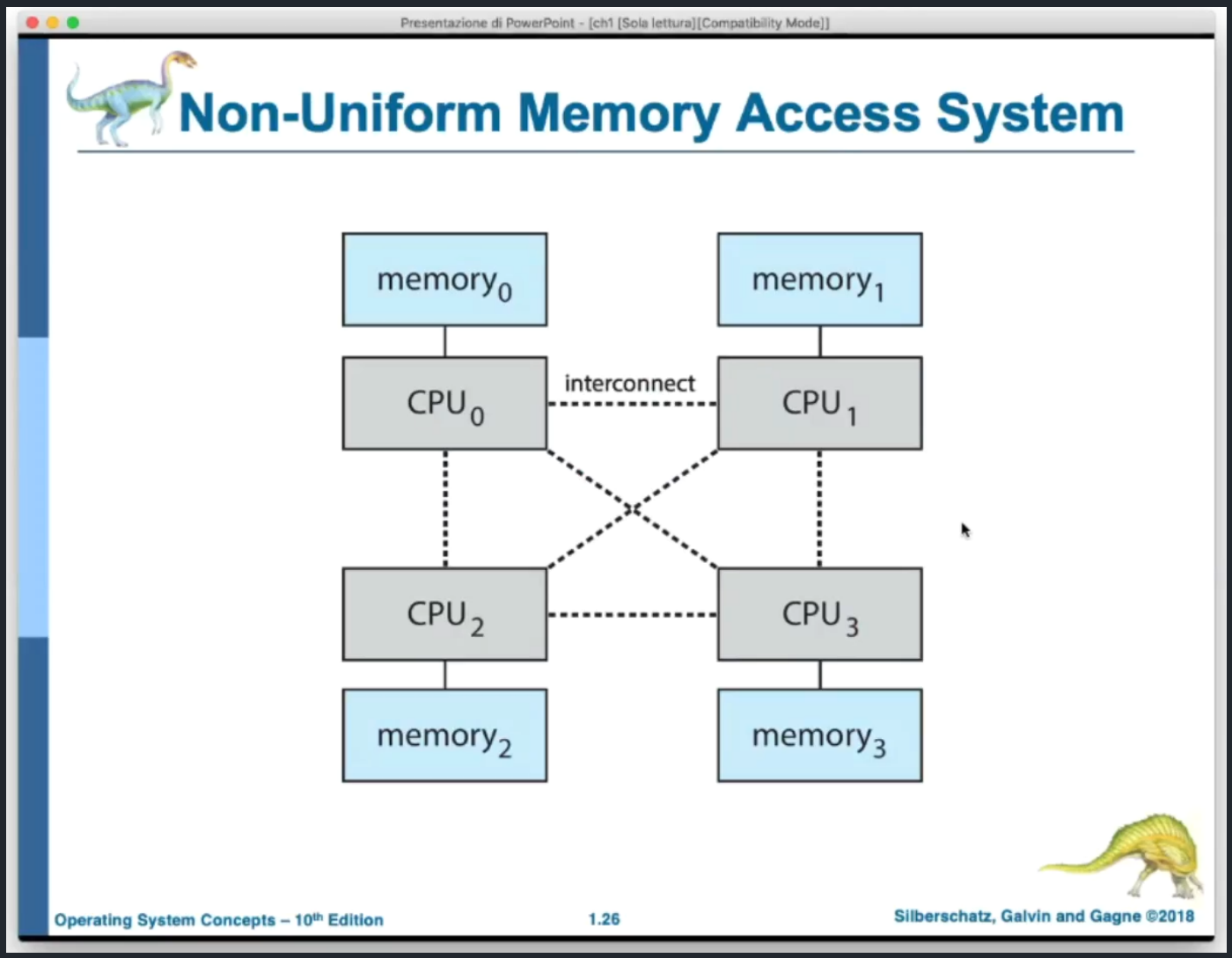
Quando ci sono CPU multiple connesse alla stessa memoria, sono possibili due schemi:

- **Schema Asimmetrico:** le CPU si dividono i compiti ("io faccio questo e tu fai quello").
- **Schema Simmetrico:** lo schema attualmente utilizzato nei sistemi moderni, permette ad ogni processore di eseguire lo stesso compito, aumentando il throughput complessivo.



> UMA: Uniform Memory Access
>
> Posso accedere a tutte le locazioni di memoria esattamente negli stessi tempi con ogni processore.

Non-Uniform Memory Access System NUMA



> NUMA

Quando la *CPU1* vuole accedere alla *memory3*, effettua l'operazione in maniera veloce, ma quando la stessa CPU vuole accedere, ad esempio, alla *memory3*, dovrà farlo tramite **interconnessione**. Questo vuol dire che i tempi di accesso, non sono **uniformi**, ovvero ci sono delle zone che richiedono meno tempo, e zone che richiedono più tempo.

Questo tipo di architettura non è utilizzata nei sistemi comuni, ma in sistemi che comprendono 64/128 processori, destinati ad un utilizzo di nicchia.

Sistemi Clustered (a grappolo)

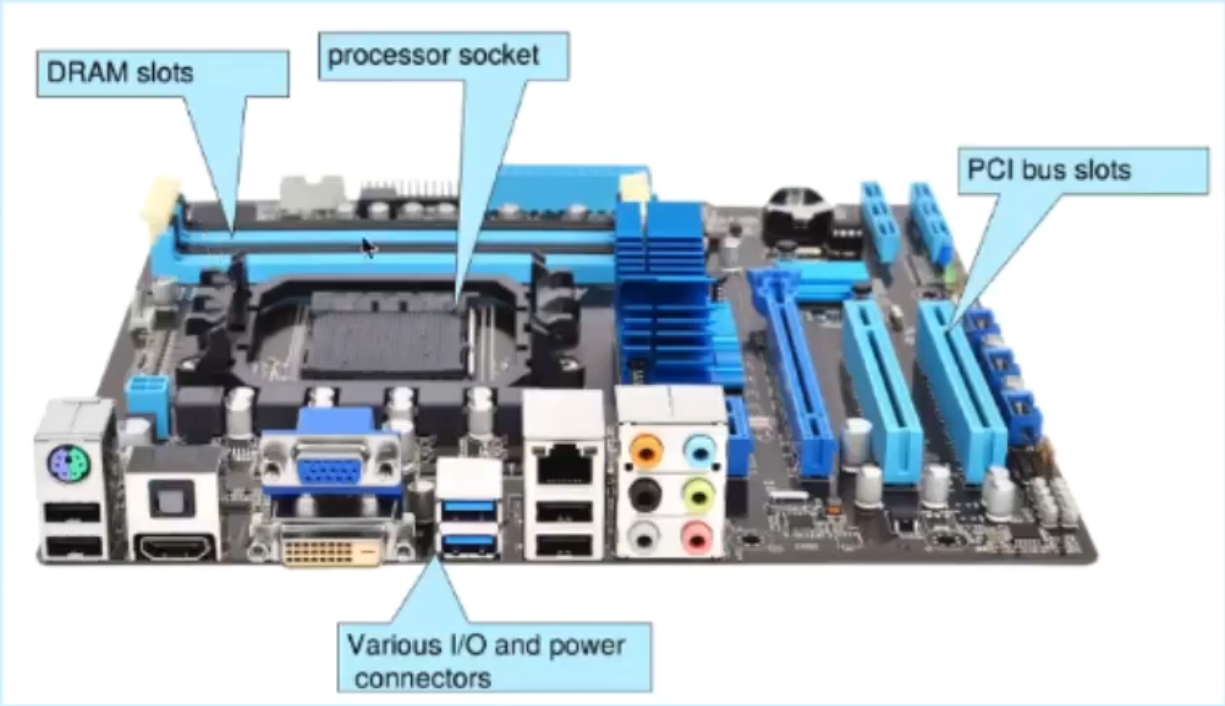
Sono sitemi in cui ci sono più computer indipendenti, che in qualche modo lavorano insieme. Con uno schema come quello precedente, possiamo costruire un cluster tramite un software che governa tutti i processori distribuendo vari compiti ad ognuno di essi. Questo tipo di sistemi possono essere utilizzati sia singolarmente che insieme.

Questi sistemi vengono utilizzati per applicazioni ad **alte prestazioni**, ovvero applicazioni che richiedono molto tempo di calcolo, e quindi richiedono una grande potenza di calcolo. Queste applicazioni vengono chiamate **HPC, ovvero high-performance computing.**

Un esempio classico per questo tipo di applicazione sono le **previsioni meteo o la ricerca di pozzi di petrolio.** Infatti queste operazioni richiedono dei calcoli enormi, e quindi spesso vengono utilizzati dei processori (sistemi singoli) organizzati in **cluster** (ovvero lavorano insieme).

Scheda madre

Consider the desktop PC motherboard with a processor socket shown below:

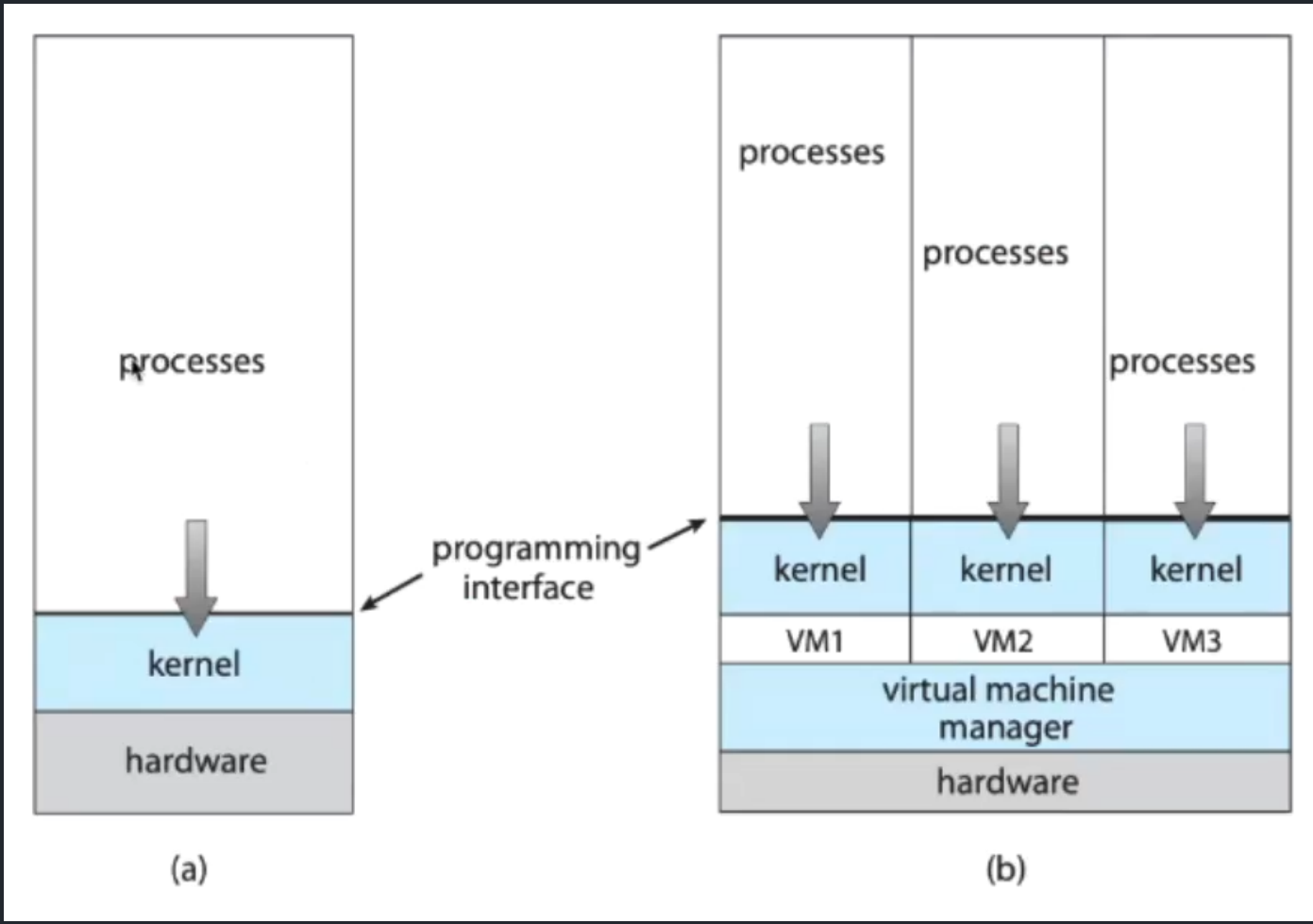


This board is a fully-functioning computer, once its slots are populated. It consists of a processor socket containing a CPU, DRAM sockets, PCIe bus slots, and I/O connectors of various types. Even the lowest-cost general-purpose CPU contains multiple cores. Some motherboards contain multiple processor sockets. More advanced computers allow more than one system board, creating NUMA systems.

Virtualizzazione

La virtualizzazione consente di gestire un SO come se stesse girando su un **hardware dedicato**. Questo nella realtà non avviene, perchè abbiamo un'unica CPU, con un set di core sui quali girano più macchine virtuali, che possono essere più sistemi operativi diversi. Questa tecnica è maggiormente usata lato **server**.

Esiste un ulteriore livello, ovvero quello dei **containers**. I containers sono una caratteristica di LINUX, che permette di avere degli ambienti di esecuzione che sono **separati** l'uno dall'altro. Ci permette di avere una soluzione più snella e leggera di una VM, ma al di sotto c'è un **unico sistema operativo.**



Sistema operativo Real Time

Esistono due distizioni di Real Time, Il **soft real time e Hard real time.**

- **Soft** : si cerca di eseguire all'interno dei vincoli di tempo tutto ciò che c'è da eseguire, ma se non si riesce non è un problema. Un esempio di questo tipo potrebbe essere un riproduttore MP3 o riproduttore video, dove si cerca di evitare di perdere frame, per evitare di avere scatti nella visione, ma se succede non è un problema.
- **Hard:** in questo tipo di aereo il processo **DEVE** essere effettuato nei limiti di

tempo. Un esempio potrebbe essere il **pilota automatico di un aereo** , il quale riceve un input dai sensori e si regola di conseguenza.

I **SO real time** non sono assolutamente sistemi come Windows o Linux, ma sono dei SO progettati appositamente. In primo luogo non hanno nulla di **virtualizzato**, perchè la virtualizzazione aggiunge delle **incertezze** nei tempi. Solitamente non hanno **dischi**, ma memoria a stato **solido**, quindi ci viene garantito con sicurezza che tutto avvenga nei limiti di tempo.
