

Introduzione ai Sistemi Operativi

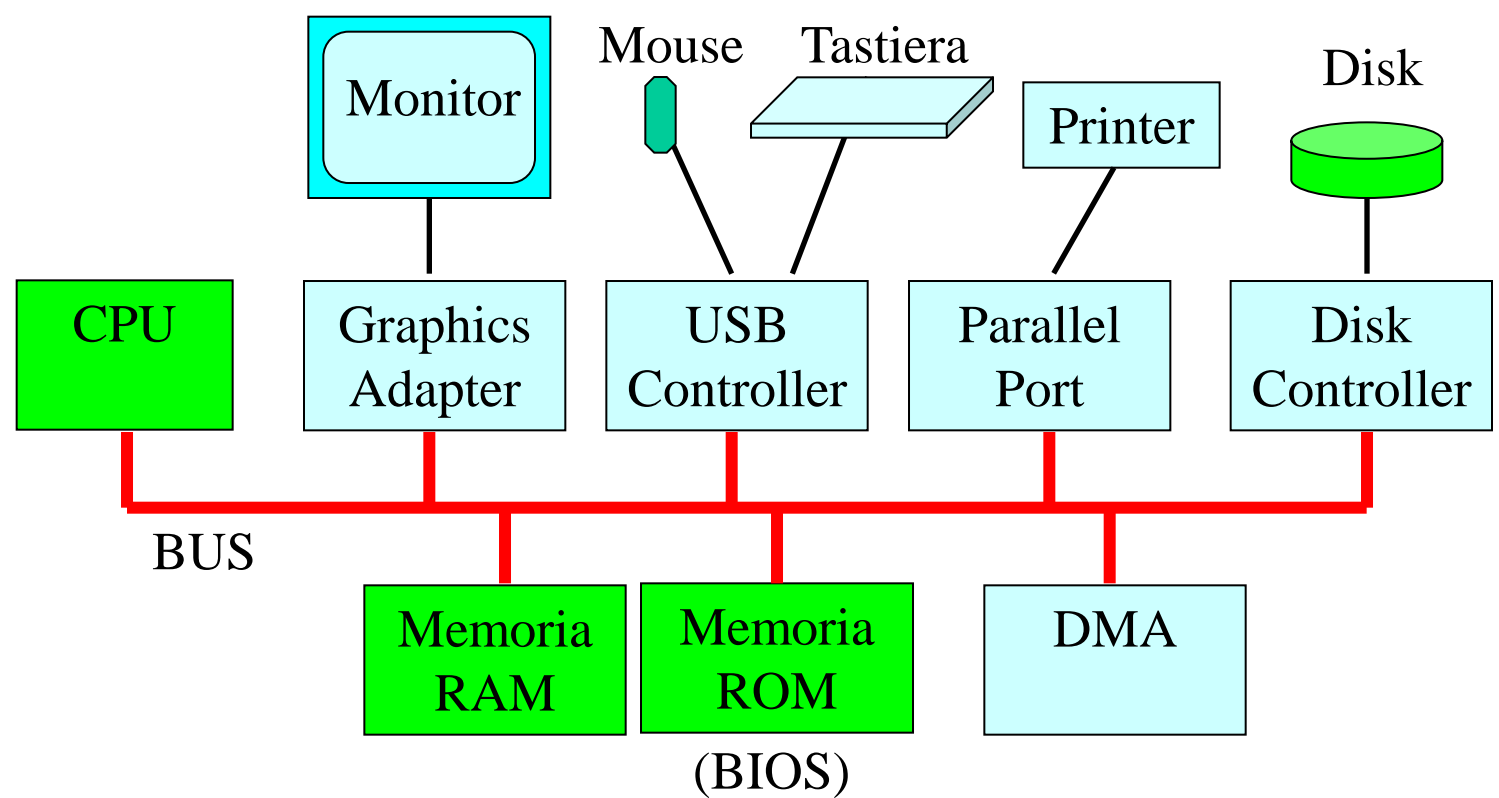
Scopo di questa parte del corso e':

- introdurre i concetti principali dei sistemi operativi,**
- identificare i requisiti hardware necessari alla gestione dei processi e del sistema operativo,**
- individuare le interfacce software su cui i sistemi operativi si appoggiano,**
- individuare i principali servizi forniti dal sistema operativo,**
- individuare le interfacce software che i sistemi operativi forniscono ai livelli superiori.**

Questa parte del corso si basa sul capitolo 1 del libro di testo “Sistemi Operativi” di A. Silberschatz, nona edizione.

Le presenti slide hanno il solo scopo di rendere più agevole seguire le lezioni.

Hardware di un Computer



CPU esegue istruzioni, effettua calcoli, controlla input/output, coordina spostam. dati.

BUS trasferisce i dati tra la memoria e gli altri dispositivi.

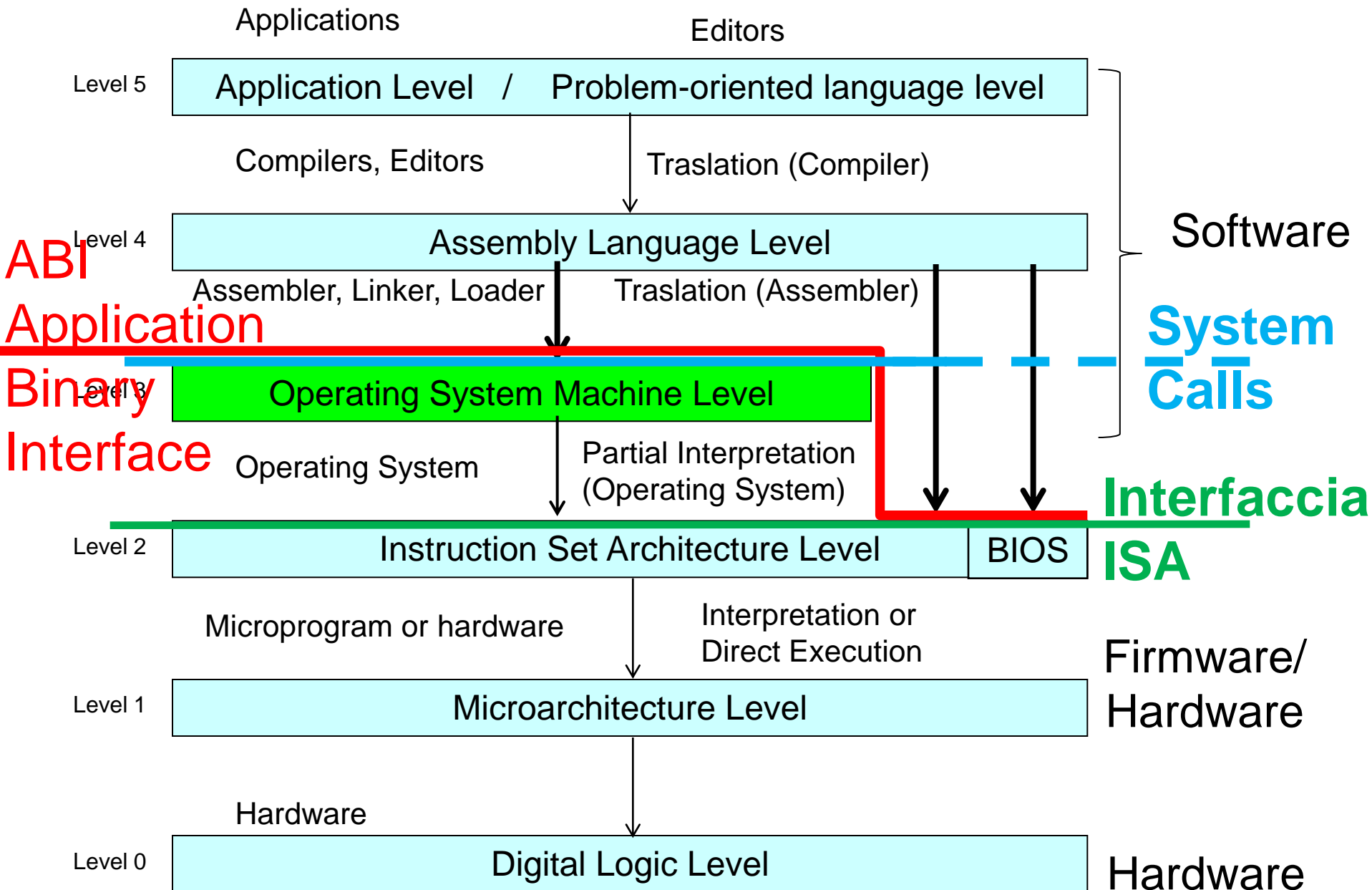
Memoria RAM (volatile) mantiene programmi (seq. di istruzioni) quando devono essere eseguiti, e i dati che i programmi usano, ma solo finché il computer è acceso.

Memoria ROM (permanente) contiene il BIOS, le istruzioni per fornire i servizi base, usate dalla CPU anche nel momento dell'accensione del computer (il BOOT).

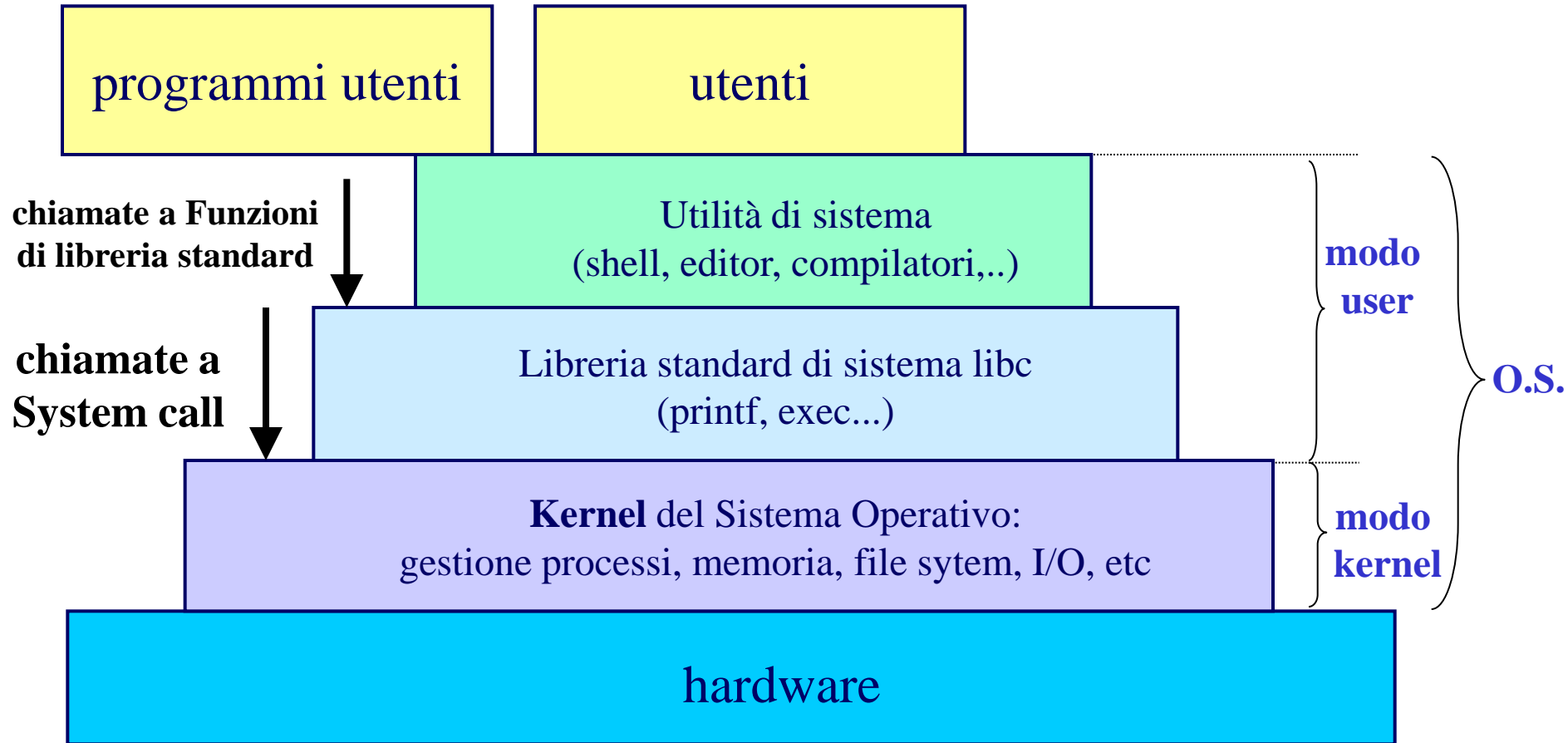
Dischi mantengono enormi quantità di programmi e dati, anche dopo lo spegnimento del computer.

Sono inoltre presenti alcune periferiche di **I/O** (Input/Output).

Interfacce dei servizi esposte alle applicazioni



Interazioni con il Sistema Operativo



Protezione hardware: Modi di esecuzione delle CPU

(es: Instruction Set Architecture of IA-32 - **Privilege Levels (Rings)**)

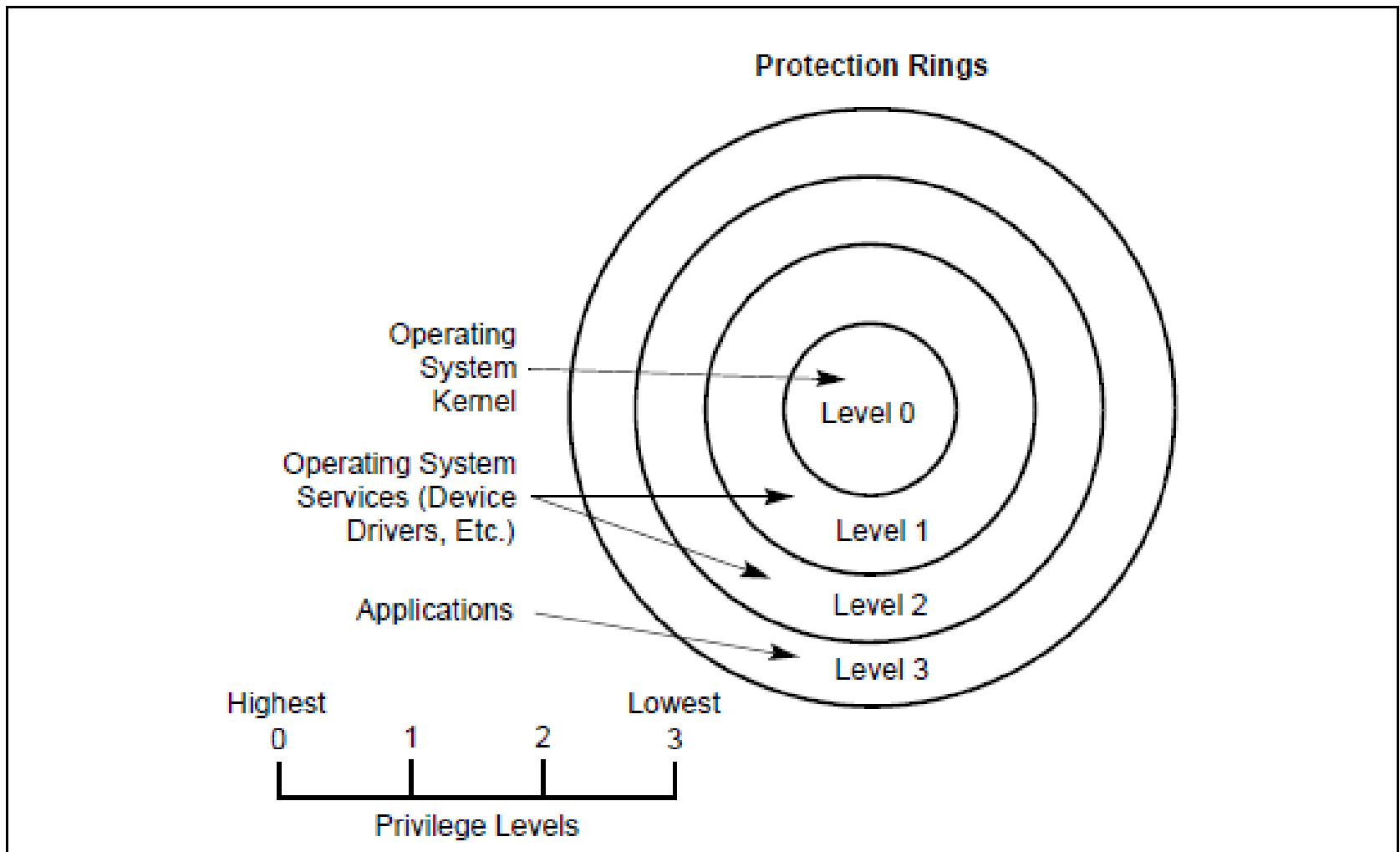
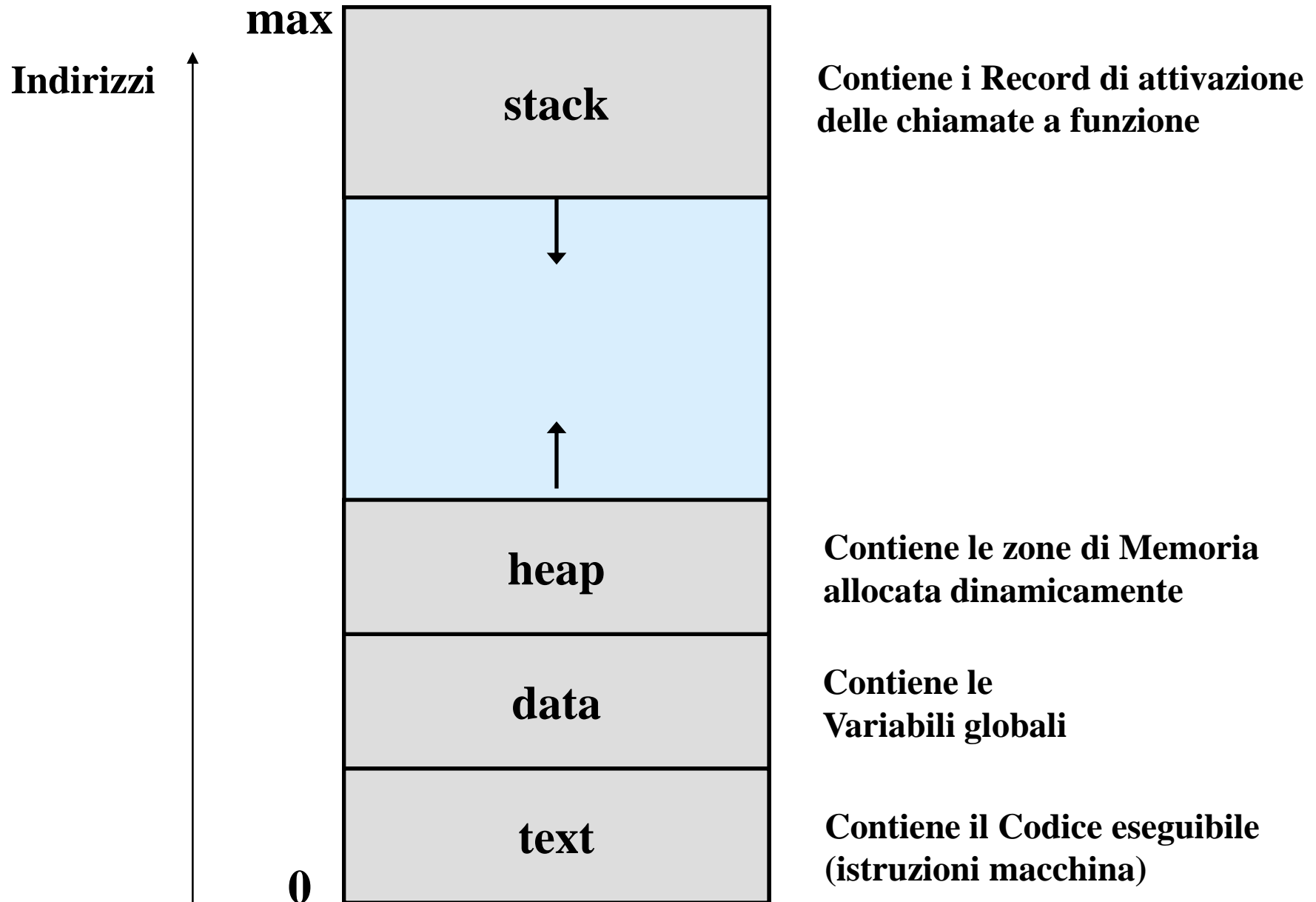


Figure 6-3. Protection Rings

Organizzazione della Memoria usata dalla parte utente dei Processi



Supporto ai Processi da parte del livello ISA

I processori moderni hanno caratteristiche comuni che permettono di fornire ai livelli superiori una interfaccia di programmazione denominata livello ISA (Instruction Level Architecture) variabile da processore a processore ma avente alcune analogie:

Modalità di protezione (modo kernel, modo utente) e istruzioni per switch tra i modi.

ALU, unità aritmetico-logica e istruzioni per comandarla.

Registri general purpose (di uso generale) e istruzioni per scambio dati tra registri della CPU e memoria , periferiche .

Registri Specializzati (nome differente a seconda del processore):

- Extended Instruction Pointer (EIP) serve a formare l'indirizzo in memoria (detto Program Counter (PC)) della prossima istruzione da eseguire. Contiene l'offset della prossima istruzione da eseguire, rispetto all'inizio del segmento di codice.**
- Program Status (PS) detto anche Registro di Stato (Status Register).**
- Stack Segment Register (SS). Punta all'inizio dello stack.**
- Code Segment Register (CS). Punta all'inizio della sezione text, il codice eseguibile.**
- Data Segment Register (DS). Punta all'inizio della sezione data, con le var globali.**

Registri disponibili ai Programmi nei processori Intel "a 32 bit" cioe' di tipo IA-32 (i386, Pentium, ...)

- Registers are high speed memory inside the CPU
 - Eight 32-bit general-purpose registers
 - Six 16-bit segment registers
 - Processor Status Flags (EFLAGS) and Instruction Pointer (EIP)

32-bit General-Purpose Registers

EAX
EBX
ECX
EDX

EBP
ESP
ESI
EDI

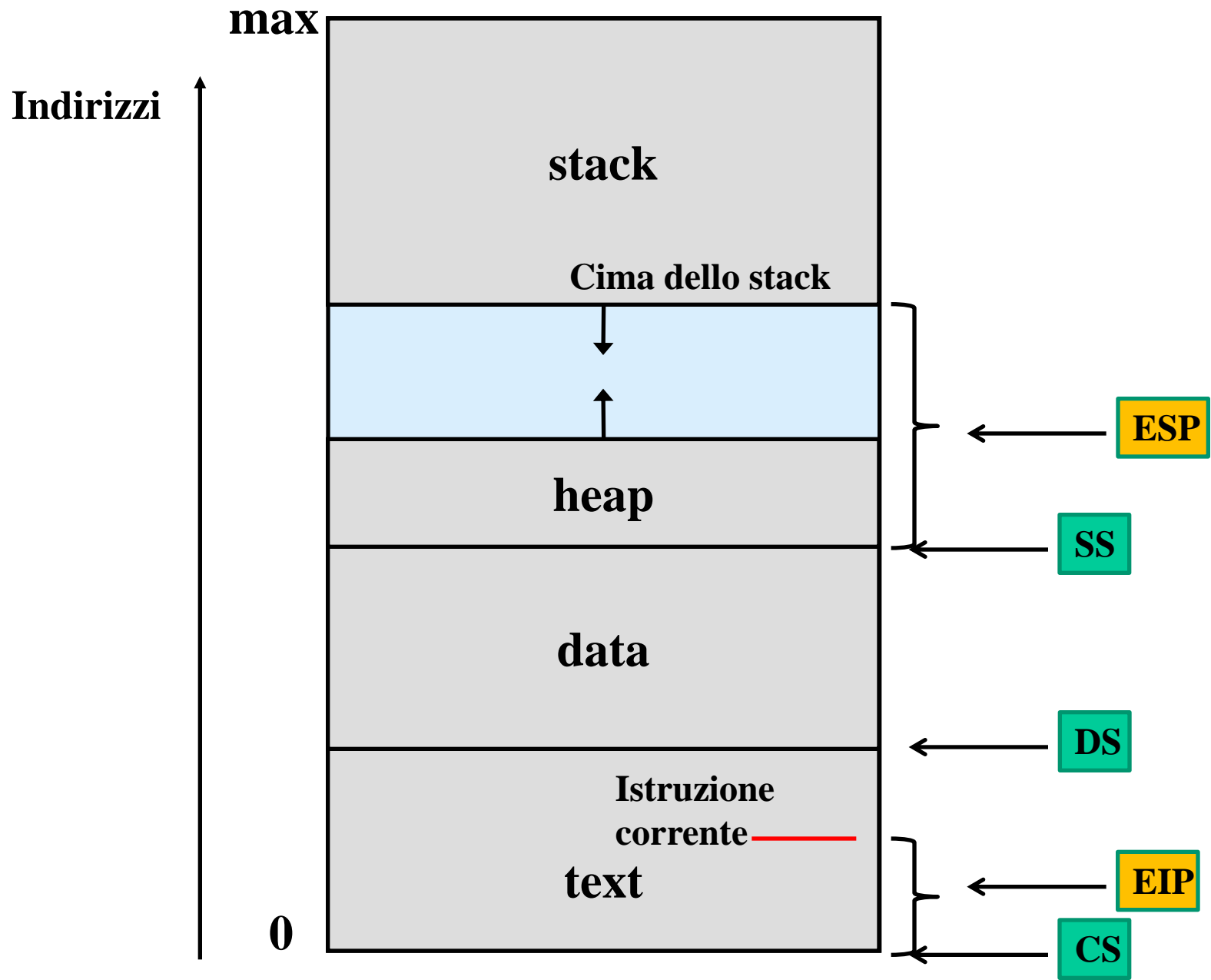
16-bit Segment Registers

EFLAGS
EIP

CS	ES
SS	FS
DS	GS

- Inoltre, Registro di Stato (Status Register), non disponibile ai programmi, definisce modalità di funzionamento, utente o kernel

Uso dei registri di segmento e di offset nella parte utente dei Processi



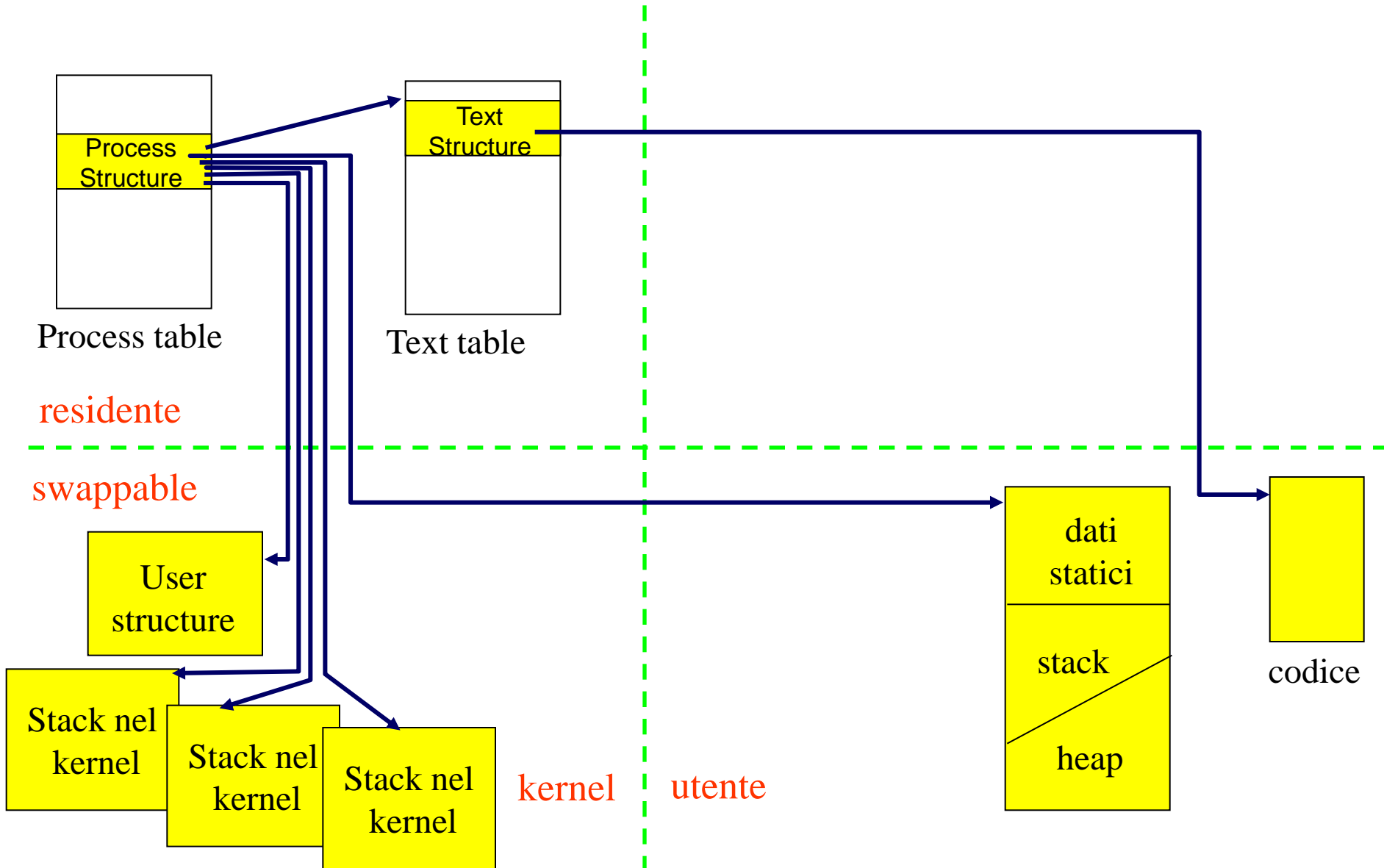
Supporto ai "modi di esecuzione della CPU" per i processi

Quando un processo esegue una chiamata ad interrupt, la CPU entra nella modalità di esecuzione kernel e deve utilizzare uno (o più) stack "di sistema" separato rispetto allo stack utilizzato per le normali chiamate a funzione.

Ciò permette al kernel di separare e proteggere i record di attivazione utilizzati nelle chiamate ad interrupt.

Ciascun processo, perciò, mantiene uno stack per ciascun livello di privilegio della CPU.

Immagine in Memoria dei Processi (spazi utente & kernel)



- Notare la presenza di uno stack per le funzioni utente nello spazio utente e di più stack per l'esecuzione delle system calls in spazio kernel.

Cosa c'e' in esecuzione? Task

In un computer sono in esecuzione:

- **alcuni processi (i programmi lanciati dagli utenti)**
- **alcuni gestori degli interrupt**
 - **per eventi asincroni originati dalle periferiche**
(scadenza timer, arrivo caratteri da tastiera, arrivo msg da DMA, da usb, da rete)
 - **per eventi sincroni generati dai processi**
(invocazioni di system call mediante chiamate esplicite alla istruzione che esegue interrupt software).
- **alcuni gestori delle eccezioni**
 - **per eventi sincroni generati dalla istruzioni eseguite dai processi**

I processi, gli interrupt e le eccezioni eseguono ciascuno in un proprio contesto denominato Task. I Task mantengono le informazioni sui segmenti di memoria utilizzati e sul valore corrente dei registri principali.

Lo scheduler del sistema operativo decide per quanto tempo eseguire ciascun task.

Salvare e ripristinare lo stato di un task consente di intervallare l'esecuzione di piu' task che potranno cosi' utilizzarne a turno la CPU (interleaving**) dando un'impressione di un'esecuzione contemporanea.**

Instruction Set Architecture of IA-32

Task

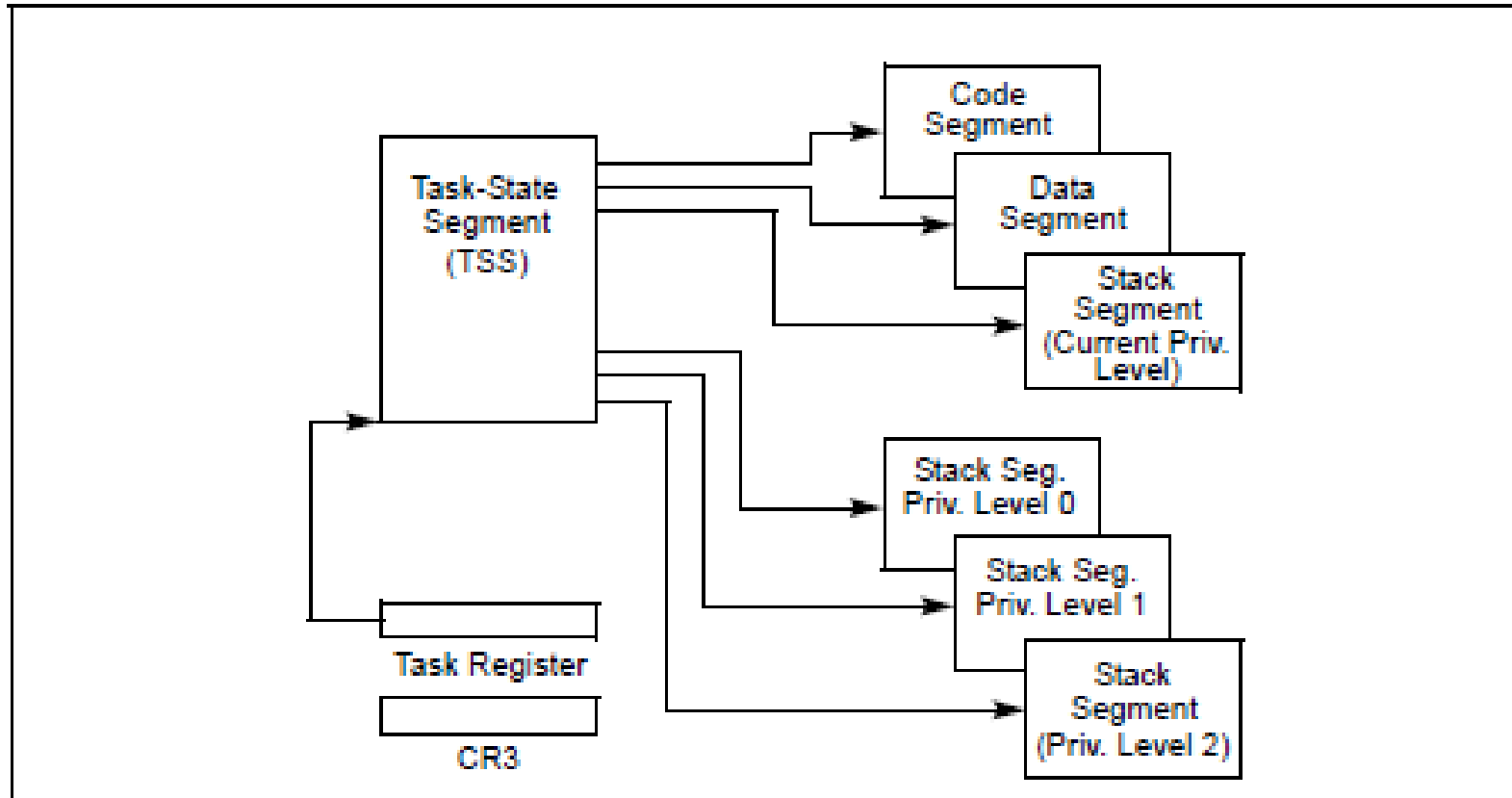


Figure 7-1. Structure of a Task

ISA of IA-32

Task- State Segment

31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0


 Reserved bits. Set to 0.

Figure 7-2. 32-Bit Task-State Segment (TSS)

Il sistema operativo mette a disposizione diversi servizi:

Gestione della Protezione:

- protezione della CPU - sfrutta la capacità della CPU di operare in due diverse modalità:
 - **modalità kernel**, che permette di utilizzare tutte le istruzioni ISA, tutti gli indirizzi di memoria, tutti i registri, tutte le periferiche.
 - **modalità utente**, che limita le istruzioni eseguibili, l'accesso alla memoria ai registri e alle periferiche.
- protezione della Memoria.

Gestione della CPU:

- scheduling (quale task deve usare la CPU e per quanto tempo?)

Gestione della Memoria:

- Indirizzamento, Segmentazione, Protezione, Memoria Virtuale, Paging.

Gestione dell' Input/Output:

- Periferiche I/O (tastiera, video, rete, dischi), Storage (organizzazione del filesystem).

Chiamate di Sistema (System Calls), Librerie di sistema, programmi di utilità:

- Servizi messi a disposizione in diverse modalità. Le system call sono disponibili mediante chiamate ad interrupt effettuabili in assembly.

ISA of IA-32

System-Level Registers and Data Structures used for Task Management

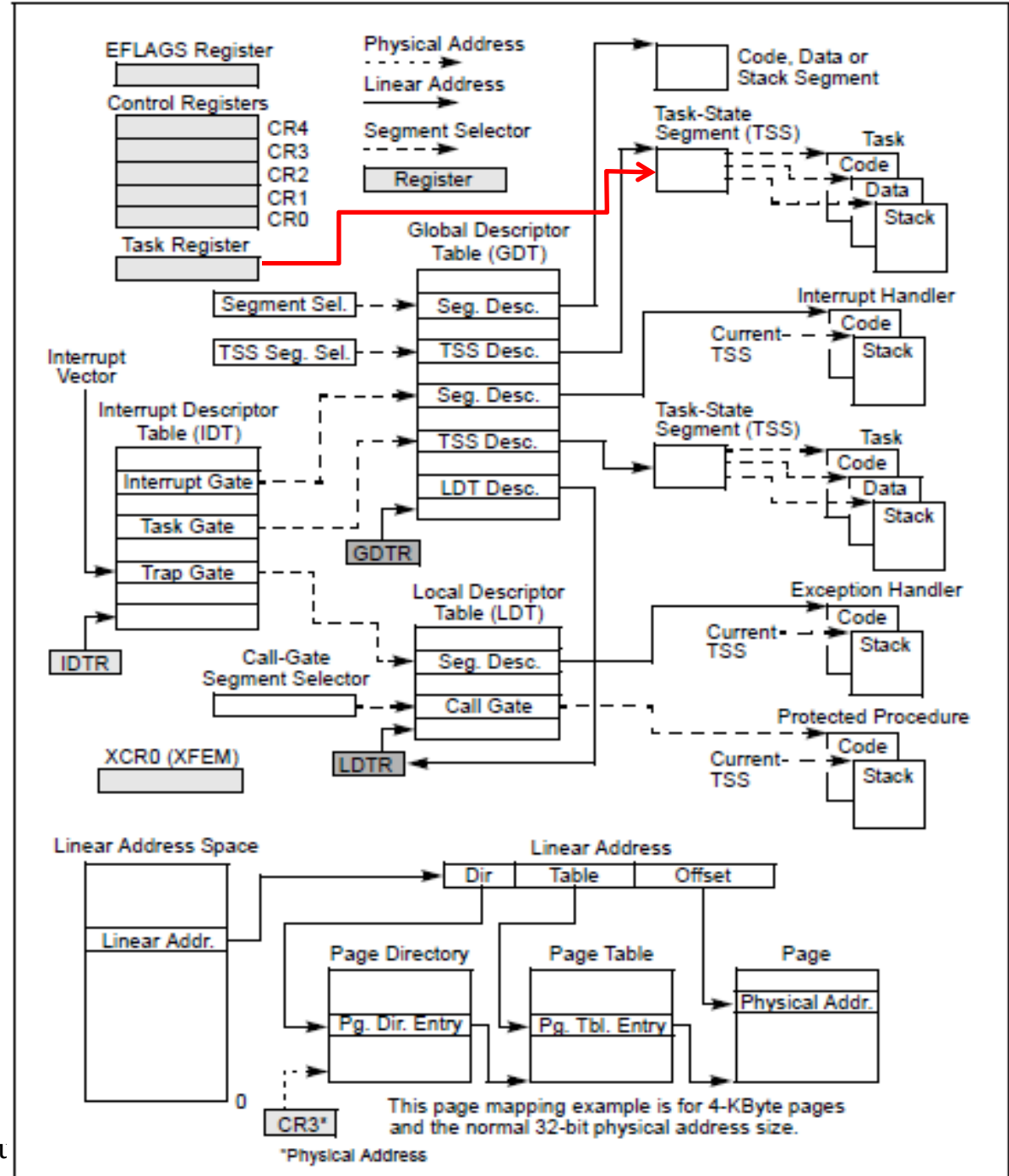


Figure 2-1. IA-32 System-Level Registers and Data Structures

ISA of IA-32

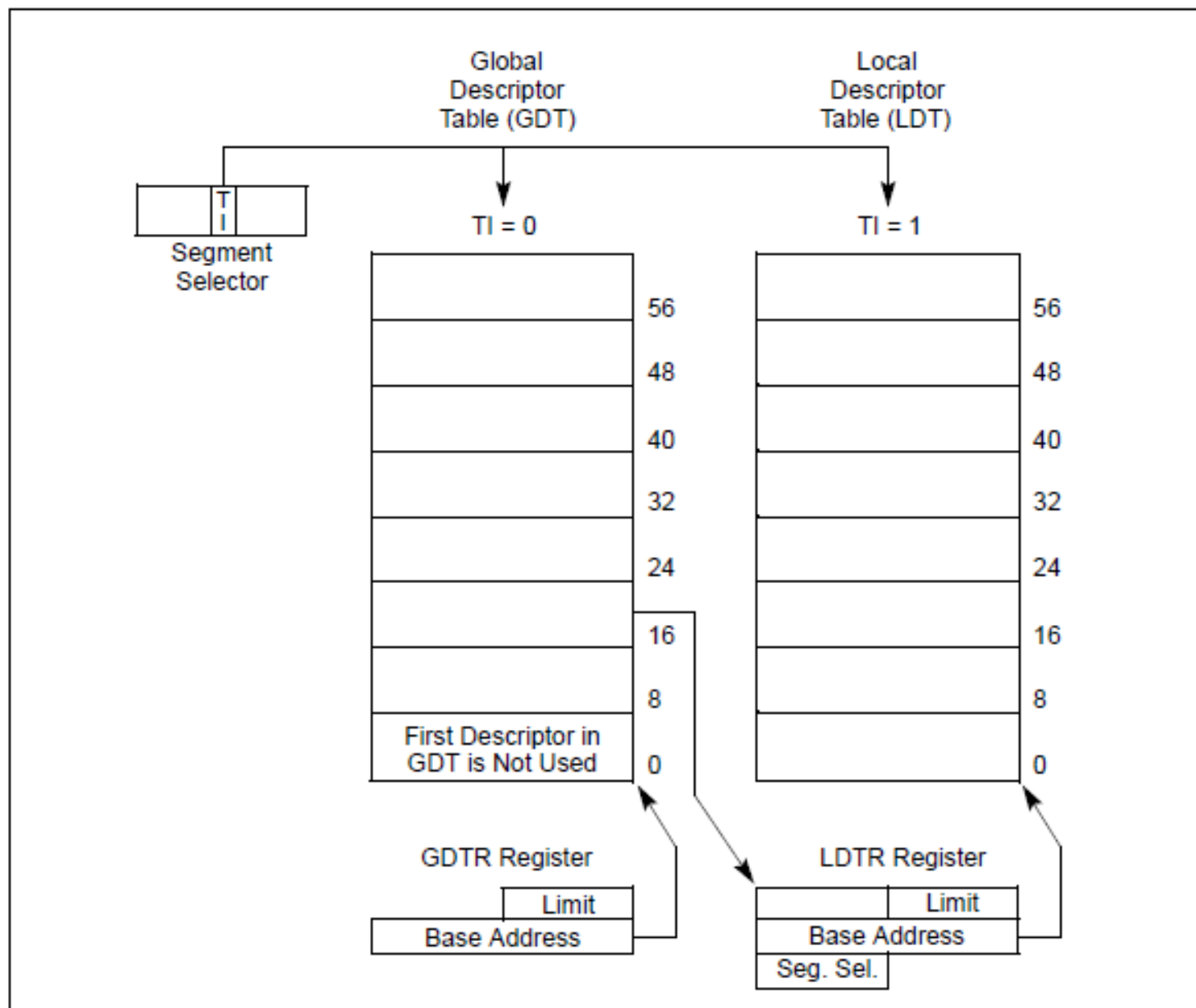
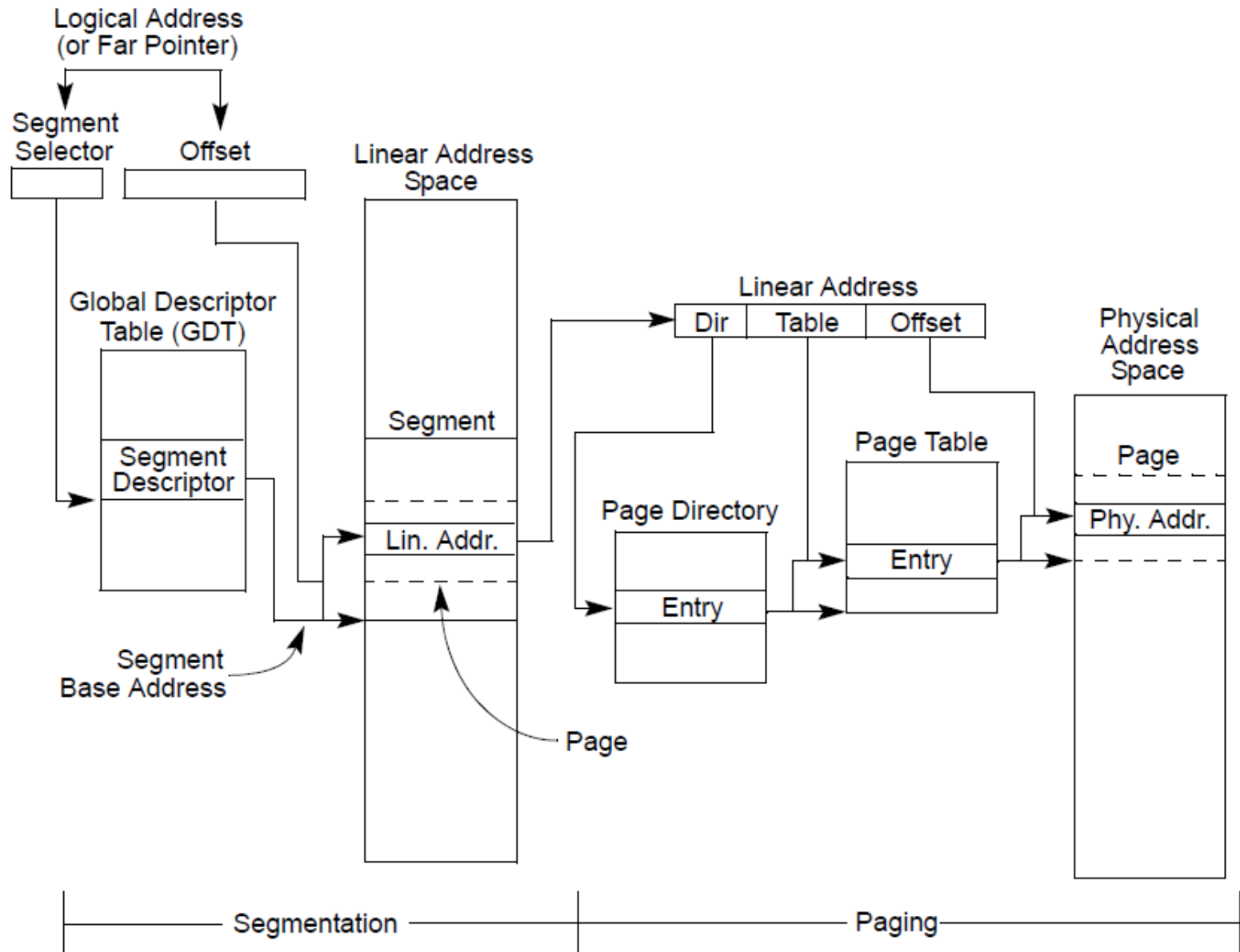


Figure 3-10. Global and Local Descriptor Tables

ISA – Segmentazione e Paginazione in IA-32



Processi

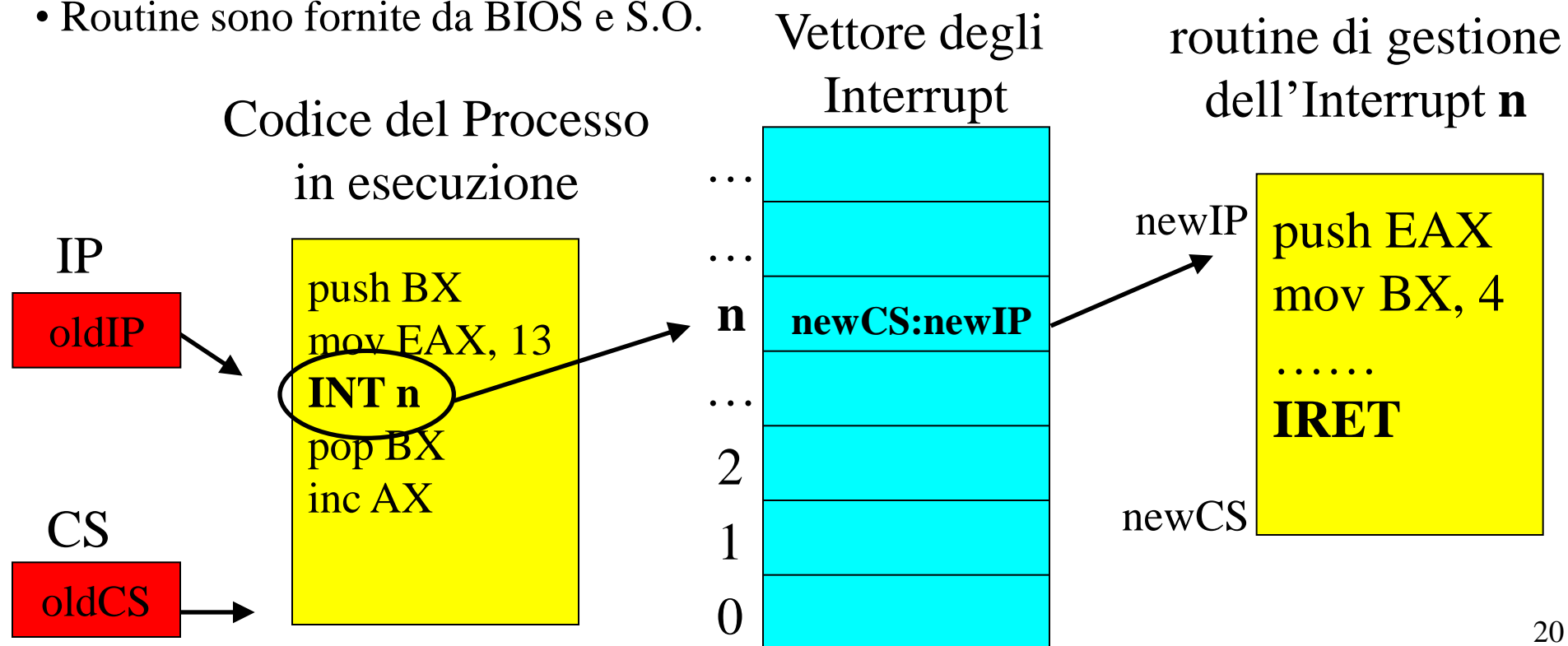
Un processo è l'istanziamento in memoria di un programma eseguibile.

Quando un programma viene lanciato, il **loader** (che fa parte del sistema operativo) :

- **crea i segmenti di memoria** per il processo popolando la tabella con le informazioni sui diversi segmenti,
- carica in memoria il codice eseguibile (sezione text) del programma,
- carica in memoria la sezione Data,
- **crea lo stack utente** per l'esecuzione delle funzioni del programma,
- **crea gli stack di sistema per l'esecuzione delle system calls** che verranno invocate dal programma utente,
- copia eventuali argomenti passati a riga di comando mettendoli a disposizione del main
- infine ordina alla CPU di eseguire la prima istruzione eseguibile corrispondente al main del programma.

Chiamata ad Interrupt da Programma

- I programmi chiamano Interrupt per ottenere servizi da sistema operativo o BIOS.
- Per chiamare un interrupt da programma, si usa l'istruzione assembly: **INT n**
- n è un numero intero maggiore o uguale a zero che viene usato come indice per accedere alla n-esima posizione del Vettore degli Interrupt.
- Il Vettore degli Interrupt è posizionato a partire dall'indirizzo 0 in memoria.
- L' n-esimo elemento del Vettore di Interrupt contiene diverse informazioni, tra cui l'indirizzo (CS:IP) della prima istruzione eseguibile (codice macchina) della routine di gestione dell'interrupt di indice n.
- Routine sono fornite da BIOS e S.O.



Chiamata ad Interrupt da Programma – Cambio di Contesto

E' in esecuzione (1) il codice di un processo che sta utilizza lo stack utente.

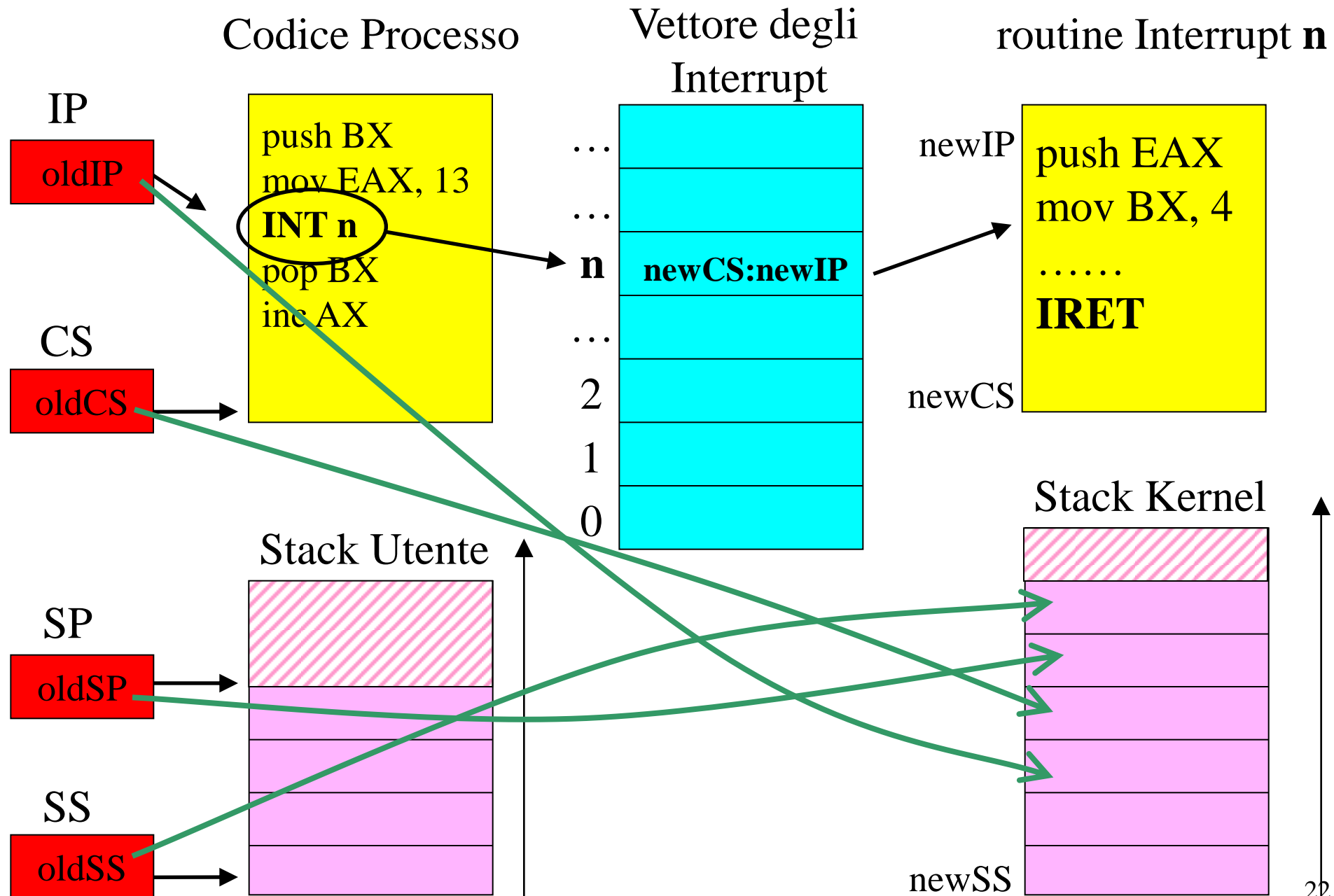
Quando la CPU esegue l'istruzione INT n :

- l' Instruction Pointer viene fatto puntare all'istruzione successiva ad INT.
- il processore passa in modalità kernel con il livello di privilegio da usare.
- il processore salva i valori attuali dei registri SS, SP, CS, IP nello stack di sistema del processo che ha chiamato l'interrupt, cioè lo stack dedicato all'esecuzione in modalità kernel per quel processo.
- il processore carica i registri SS e SP con i valori dello **stack di** sistema per quel livello di privilegio.
- il processore carica i registri CS e IP con l'indirizzo della routine di gestione dell'interrupt trovato nell'n-esima posizione del vettore di interrupt.
- A questo punto (2) viene eseguita la routine dell'Interrupt, nello stack di sistema.

La routine dell'Interrupt termina (3) con l'istruzione IRET (Interrupt Return) che :

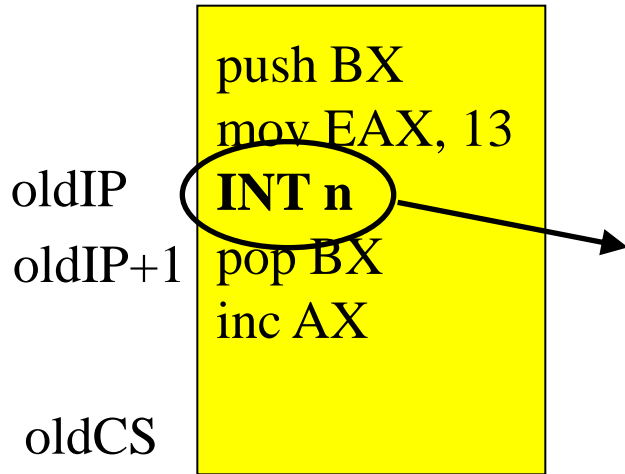
- carica i registri IP, CS, SP e SS con i valori salvati sullo stack di sistema.
- fa tornare il processore in modalità utente.
- così l'esecuzione ricomincia (4) dall'istruzione del programma successiva all'istruzione INT n utilizzando lo stack utente del processo.

(1) Prima di Chiamata ad Interrupt da Programma

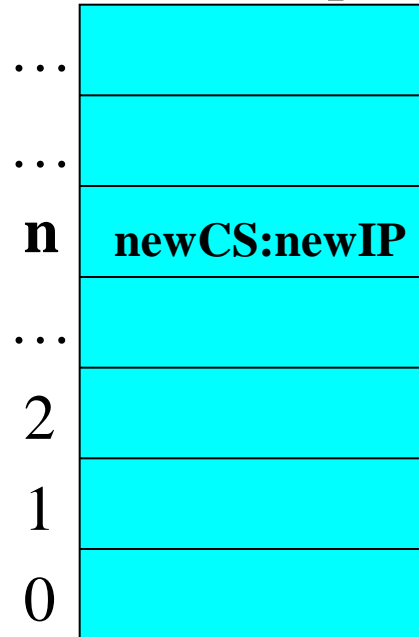


(2) Dopo esecuzione istr. INT n -

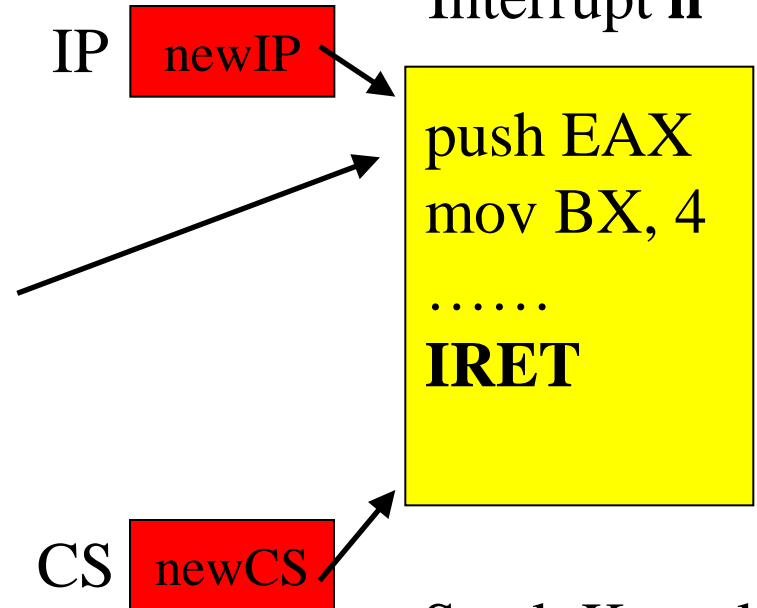
Codice Processo



Vettore degli Interrupt



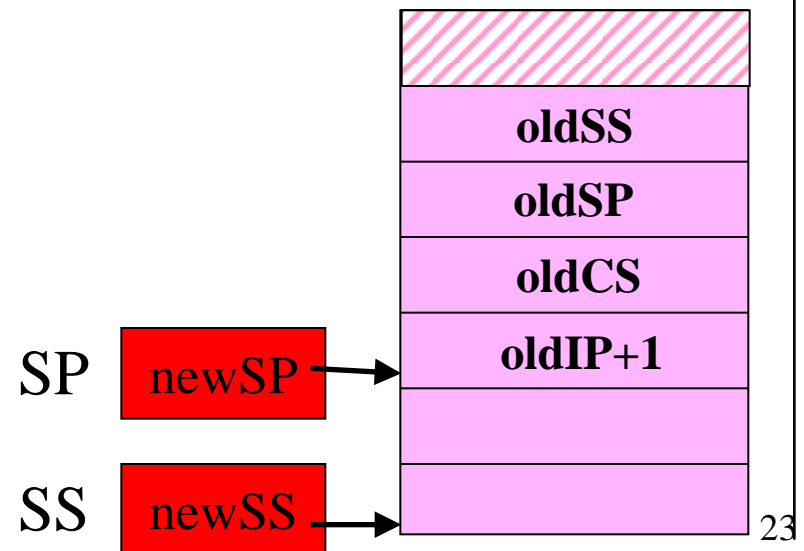
Routine di Interrupt n



Stack Utente

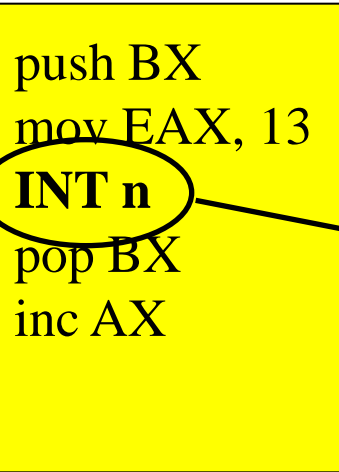


Stack Kernel

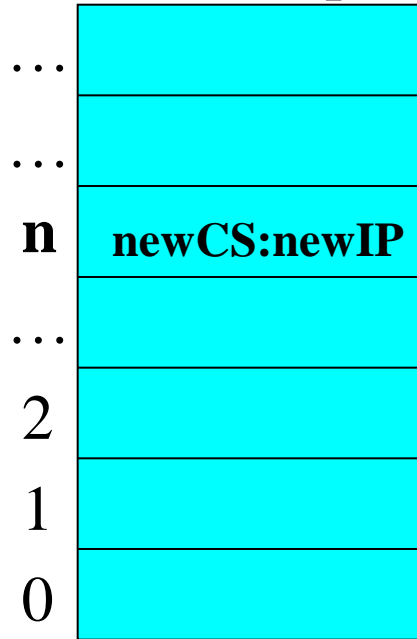


(3) Prima di esecuzione istr. IRET

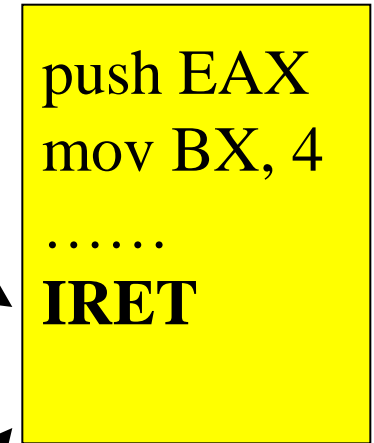
Codice Processo



Vettore degli
Interrupt



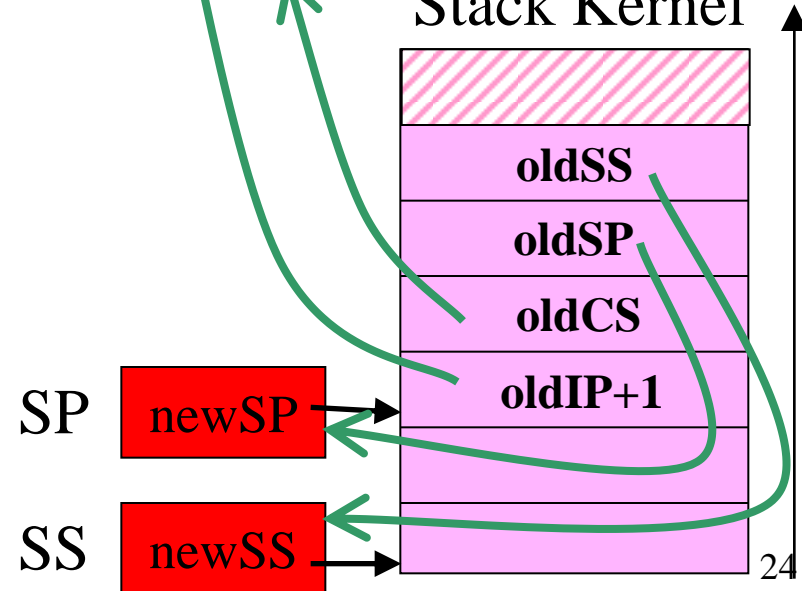
Routine di
Interrupt **n**



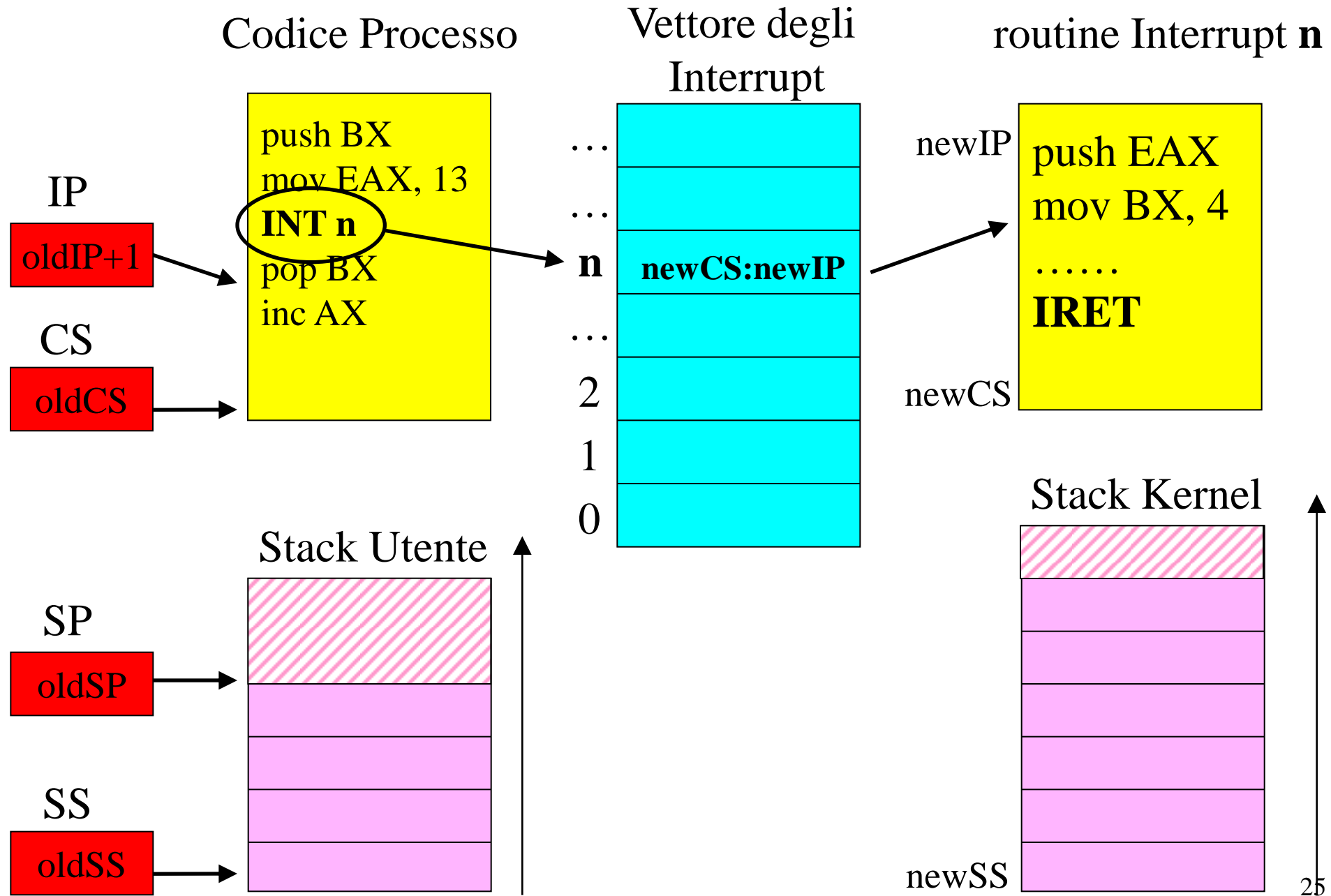
Stack Utente



Stack Kernel



(4) Dopo Esecuzione istr. IRET



Il meccanismo delle System Calls

- un programma utente, durante l'esecuzione può invocare una system call.
- la system call è implementata nella routine di gestione di un certo interrupt **n**.
- la system call viene invocata eseguendo una istruzione INT n, con un certo n intero
- il programma passa i parametri all'interrupt mettendoli in registri (EAX e altri)
- chiamata ad interrupt
- switching di contesto – da modo utente a modo kernel
 - salvataggio del contesto
- le routine di gestione degli interrupt.
 - puntatori a funzione nel vettore degli interrupt.
- esecuzione su stack del kernel (per separare contesto di esecuzione)
- ritorno a modo utente
 - ripristino del contesto

Instruction Set Architecture (ISA) & Sistema Operativo

Interrupt e Eccezioni

- Eccezioni (Exceptions) e Interruzioni (Interrupts)
 - entrambe gestite mediante routine rintracciabili a partire dal Vettore degli Interrupt
- Eccezioni (Exceptions)
 - sono sincrone rispetto alle istruzioni eseguite dalla CPU, ovvero
 - sono provocate (**involontariamente**) da una istruzione macchina eseguita dalla CPU che provoca un errore o un caso particolare da gestire
 - **TRAP**
 - l'istruzione che ha causato il trap viene momentaneamente sospesa, ma alla fine della gestione del trap viene portata a buon fine.
 - Es: trap per debugging
 - **FAULT**
 - l'istruzione che ha causato il trap viene interrotta, e viene eseguita la routine di gestione. In un successivo momento l'istruzione verrà rieseguita da principio.
 - Es: Page Fault, la pagina di memoria che contiene l'indirizzo a cui stiamo cercando di accedere è swappata su disco. La pagina viene caricata in memoria. Prima o poi l'istruzione verrà nuovamente eseguita.
 - **ABORT**
 - Causata da errore irrimediabile, l'istruzione viene abortita e il processo killato.
 - Es: divisione di un numero per zero.
 - Es: **segmentation fault** (nonostante il nome è una eccezione di tipo Abort)

Instruction Set Architecture (ISA) & Sistema Operativo

Interrupt e Eccezioni

- Interrupt

Considerando il momento in cui vengono eseguiti, gli Interrupt possono essere

- **SINCRONI (interrupt software)**

- esplicitamente chiamati dalla CPU mediante l'istruzione INT n.
- Es: la CPU ordina di fare output su video di un carattere chiamando una syscall.

- **ASINCRONI (interrupt hardware)**

- scatenati da una periferica che, mediante il BUS, avvisa la CPU che è accaduto un evento che deve essere gestito.
- Ad esempio, la tastiera avvisa che l'utente ha premuto un tasto.
- La periferica invia un segnale che codifica un numero intero, tale numero è l'indice dell'Interrupt da chiamare.
- La CPU interrompe quello che sta facendo e gestisce l'Interrupt.

Considerando la priorità e la possibilità di essere ritardati, gli Interrupt possono essere

- **MASCHERABILI**

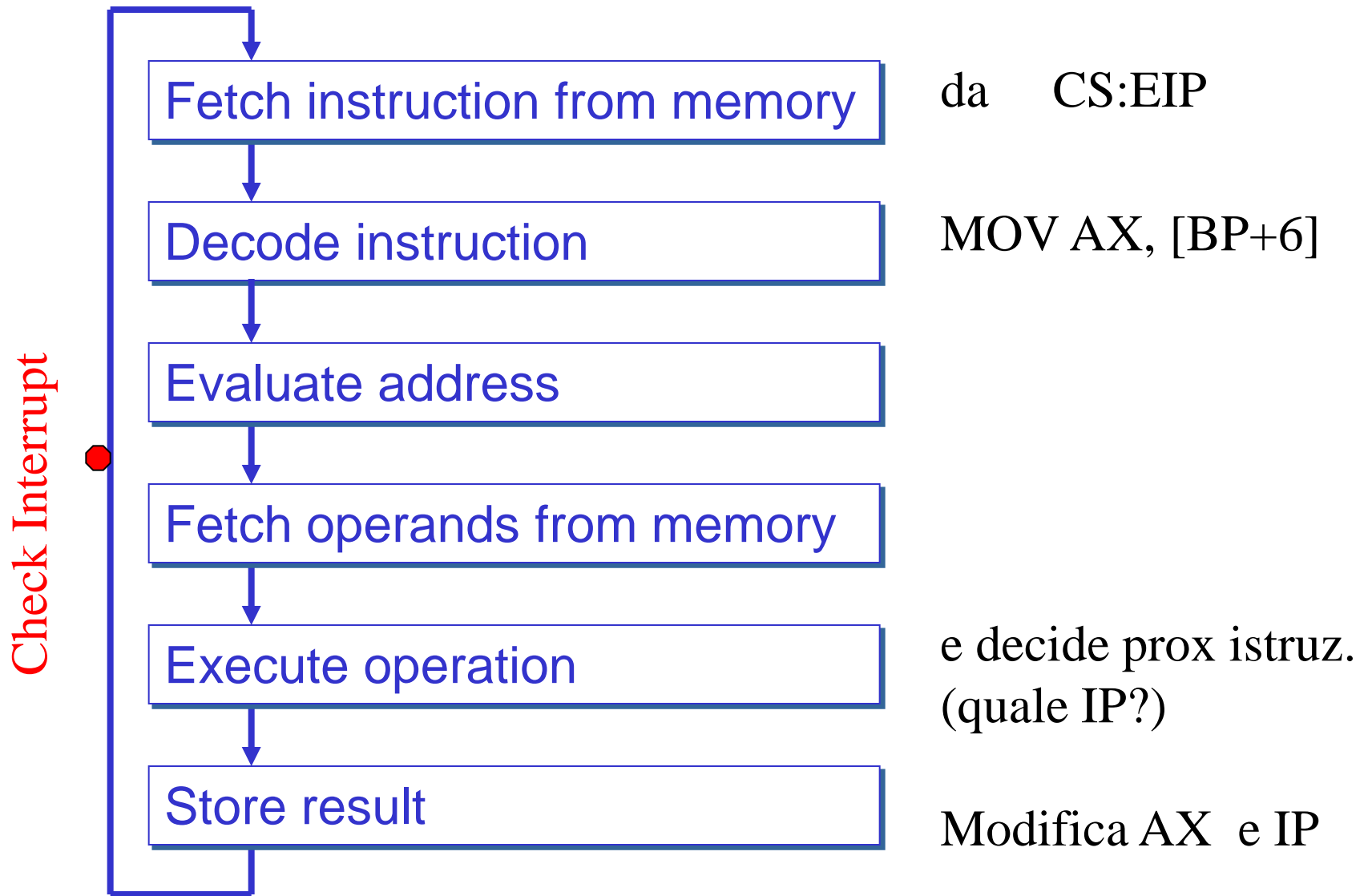
- Es: E' arrivato un carattere da tastiera, ma se c'è qualcosa più urgente da fare è meglio svolgere i compiti più urgenti.

- **NON MASCHERABILI**

- Es: Avviso di errore sul Bus, bisogna gestirlo immediatamente.

- Durante l'esecuzione di una routine di gestione degli Interrupt, la CPU setta un bit (un piedino) che disabilita la gestione degli Interrupt Mascherabili.

ISA - Instruction Processing - Ciclo di Neumann



Sistemi operativi **MultiTasking**: sfruttare il **Time Sharing**

Nei moderni sistemi operativi, più processi, più thread, più gestori di interrupt ed gestori di eccezioni, sono presenti in memoria contemporaneamente (**MultiTasking**), e cercano di eseguire concorrentemente contendosi l'uso della CPU (**Time Sharing**).

- Lo scheduler è la componente del S.O. che si occupa di alternarli nell'uso della CPU.
- L'esecuzione della CPU viene suddivisa in un certo numero di quanti temporali (detti **time slice**).
- Allo scadere di un quanto di tempo, il processo (o thread) corrente viene interrotto e l'esecuzione passa ad un altro processo (o thread) .
- Quando un processo (o thread) richiede un'operazione di I/O, la CPU viene assegnata ad un altro processo, mentre le periferiche attendono il completamento delle operazioni di I/O.
- Quando l'operazione di I/O è stata effettuata dalle periferiche, il processo che la richiedeva si rimette a disposizione dello scheduler e attende il suo turno per l'utilizzo della CPU.

I passaggi (context switch) avvengono così frequentemente che i programmi sembrano essere eseguiti contemporaneamente.

Sistemi operativi "Interrupt Driven"

I moderni S.O. sono detti "Interrupt Driven" poichè gran parte delle funzionalità del kernel del S.O. viene eseguito all'interno di una qualche routine di gestione di un interrupt, cioè viene eseguito come reazione al verificarsi di un evento che ha sollevato un interrupt o una eccezione.

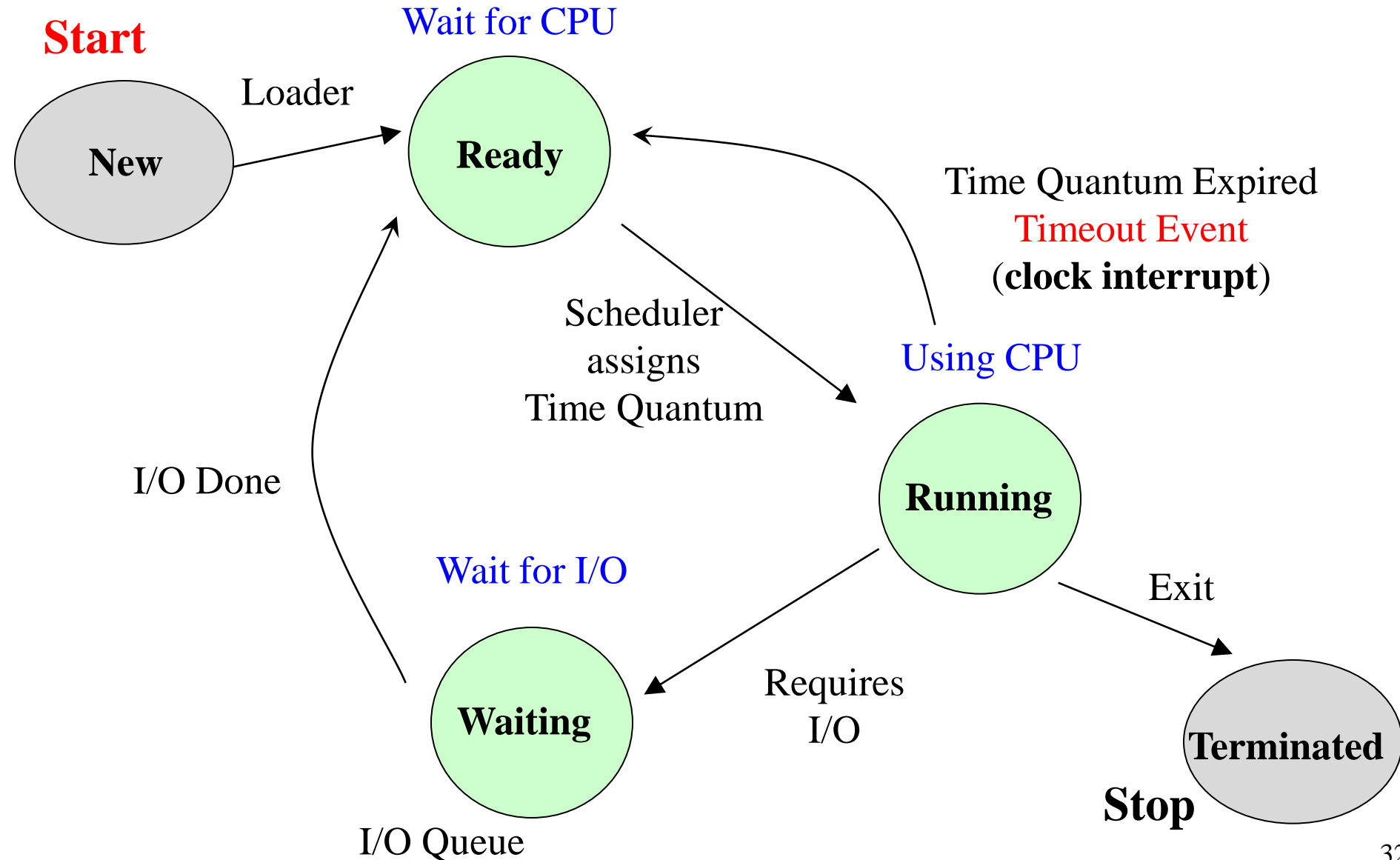
Uno dei più importanti eventi che sollevano un interrupt è lo scadere del "quanto" di tempo di esecuzione che lo scheduler ha concesso ad un processo (o ad un thread all'interno di un processo).

Allo scadere del quanto di tempo, il timer di sistema genera un interrupt che risveglia lo scheduler.

Lo scheduler decide quale altro processo (o thread) dovrà essere eseguito e passa il controllo a tale processo (thread).

Sistema Operativo - Scheduler

Ciclo di Vita dei Processi (passaggio di stato dei processi)



Passaggio di stato dei processi, sintesi

Ammissione (new → ready):

una volta creato e collocato in memoria dal loader, il processo è inserito nella coda dei processi ready.

Dispatch (ready → running):

il processo è scelto dallo scheduler come il prossimo ad eseguire ed è messo in esecuzione.

Interrupt di tempo (running → ready):

il processo termina il suo quanto di tempo e torna in coda nella coda dei ready

Richiesta (running → waiting):

il processo effettua una chiamata a sistema bloccante e viene inserito nell'opportuna coda di attesa

Evento (waiting → ready):

l'attesa termina quando si verifica un evento e il processo torna nella coda dei ready

Conclusione (running → terminated):

il processo esegue l'ultima istruzione e termina