

Sistemi Operativi

Laurea in Ingegneria Informatica

Università Roma Tre

Docente: Romolo Marotta

Introduzione

1. Obiettivi e funzioni di un sistema operativo
2. L'evoluzione dei sistemi operativi
3. Elementi chiave

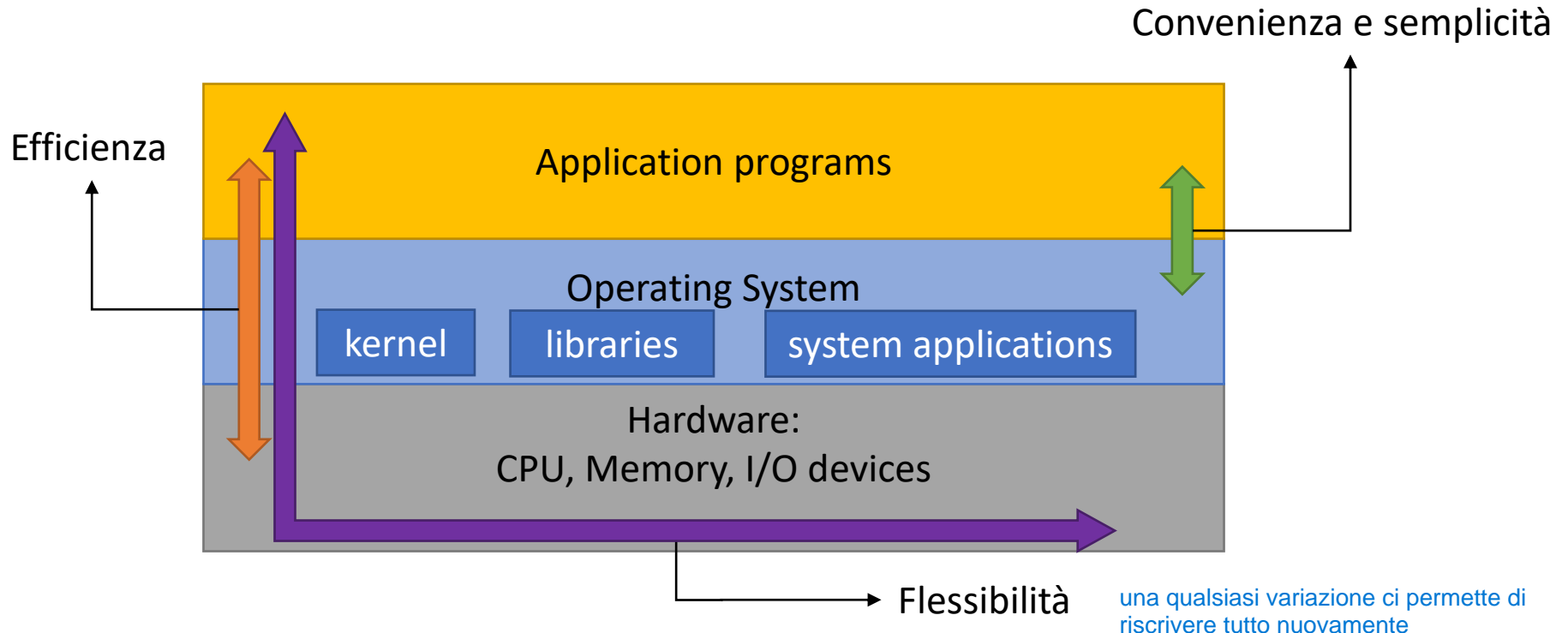
Cos'è un Sistema Operativo?

- È un **software** che:
 - **controlla** l'esecuzione di altri programmi
 - agisce come **interfaccia** tra applicazioni e hardware
- Obiettivi
 - **Convenienza e semplicità:** rendere lo sviluppo del software più semplice, mascherando peculiarità dell'hardware
 - **Efficienza:** ottimizzare l'uso delle risorse hardware da parte degli applicativi
 - **Flessibilità e manutenibilità:** un sistema operativo è costruito per facilitare lo sviluppo, testing ed aggiunta di nuove funzionalità di sistema

Cos'è un Sistema Operativo?

KERNEL: insieme di software che è in RAM che gestisce le risorse per conto delle applicazioni

- È un **software** che:
 - **controlla** l'esecuzione di altri programmi
 - agisce come **interfaccia** tra applicazioni e hardware



Prospettiva storica

- **40s – mid 50s: Serial processing**

- Nessun Sistema Operativo
- Il programma interagisce direttamente con l'hardware
- L'utente è responsabile di tutte le fasi necessarie ad avviare l'esecuzione

- caricamento del compilatore e del sorgente
- compilazione e salvataggio del compilato
- caricamento e collegamento con moduli predefiniti
- caricamento ed avvio del programma

1. fatto tutto dal
programmatore

Costo di
setup
significativo

- Nessun supporto allo **scheduling** < che processo devi andare in esecuzione

- un job alla volta
- prenotazione su fogli cartacei

Allocazione
STATICA dello slot
temporale

2. poca flessibilità

Le prime ottimizzazioni fu processare i job uno dopo l'altro, cioè che faccia tutto la macchina

• mid 50s – late 60s: Batch System

- Introduzione di un software chiamato **monitor**
 - assimilabile ad un primo sistema operativo
- L'utente sottomette i **job** ad un operatore
 - utenti non interagiscono più con l'hardware
- L'operatore prepara un **batch** (sequenza) di job memorizzati su un dispositivo di input
- Il monitor carica da dispositivo di input in memoria ed avvia il programma
- Il programma
 - carica moduli necessari alla sua esecuzione tramite comandi espressi in **job control language** per dire al monitor di caricare un pezzo di software
 - cede il controllo al monitor in caso di interazione con dispositivi, terminazione o errore

Ridotti i costi di setup



ma sprechiamo un po' di memoria per mantenere il monitor

Prospettiva storica

- **mid 50s – late 60s: Batch System**

Il monitor:

- è responsabile di avviare il successivo job nel batch
- risiede (almeno in parte) sempre in memoria

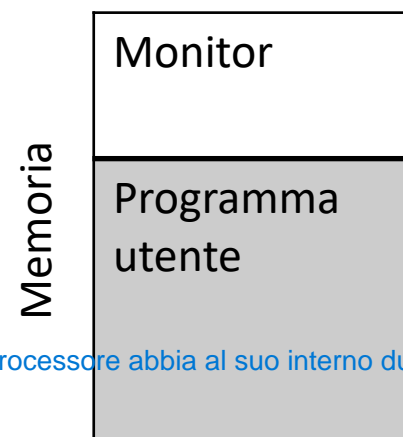
L'hardware insegue e risolve le nuove problematiche introdotte dall'uso del monitor

- Impedire ai job di corrompere l'area di memoria del monitor ⇒ **Memory protection**
- Impedire ai job di monopolizzare l'uso del sistema ⇒ **Timer**
- Impedire ai job di avere accesso incontrollato ai dispositivi di I/O ⇒ **Istruzioni Privilegiate**

Attese ridotte tra un job e l'altro

USER
MODE

KERNEL
MODE



Istruzioni Privilegiate: che solo il monitor possa eseguire

Prospettiva storica

Risolve le principali cause di sottoutilizzo di **COSTOSO** hardware nei sistemi seriali

• mid 50s – late 60s: Batch System

Benefici:

- Ridotti tempi di setup
- Ridotti tempi di inattività del sistema



Costi:

- parte della memoria è allocata per il monitor e non può essere utilizzata dai job
- parte del tempo di processore è consumato dal monitor

Prospettiva storica

- primo problema che ci sono tanti vuoti
- secondo problema stimato sottoutilizzando l'hardware

• mid 50s – late 60s: Batch System

- L'uniprogrammazione è il maggior limite

Tempo utile per eseguire attività di altri job

CPU	Run Job A		Run Job A		Run Job B		Run Job B		Run Job C	
DISK		I/O		I/O		I/O		I/O		I/O

$$\text{Utilizzazione CPU} = \frac{5}{12} = 42\%$$

$$\text{Utilizzazione I/O} = \frac{7}{12} = 58\%$$

MEMORY	Spazio libero per ospitare più job		
	15%	30%	15%
	Monitor (30%)		

$$\text{Utilizzazione Memoria} = 45\% \cdot \frac{6}{12} + 60\% \cdot \frac{4}{12} + 45\% \cdot \frac{2}{12} = 50\%$$

lo spazio libero verrà utilizzato per far lavorare i job in contemporanea

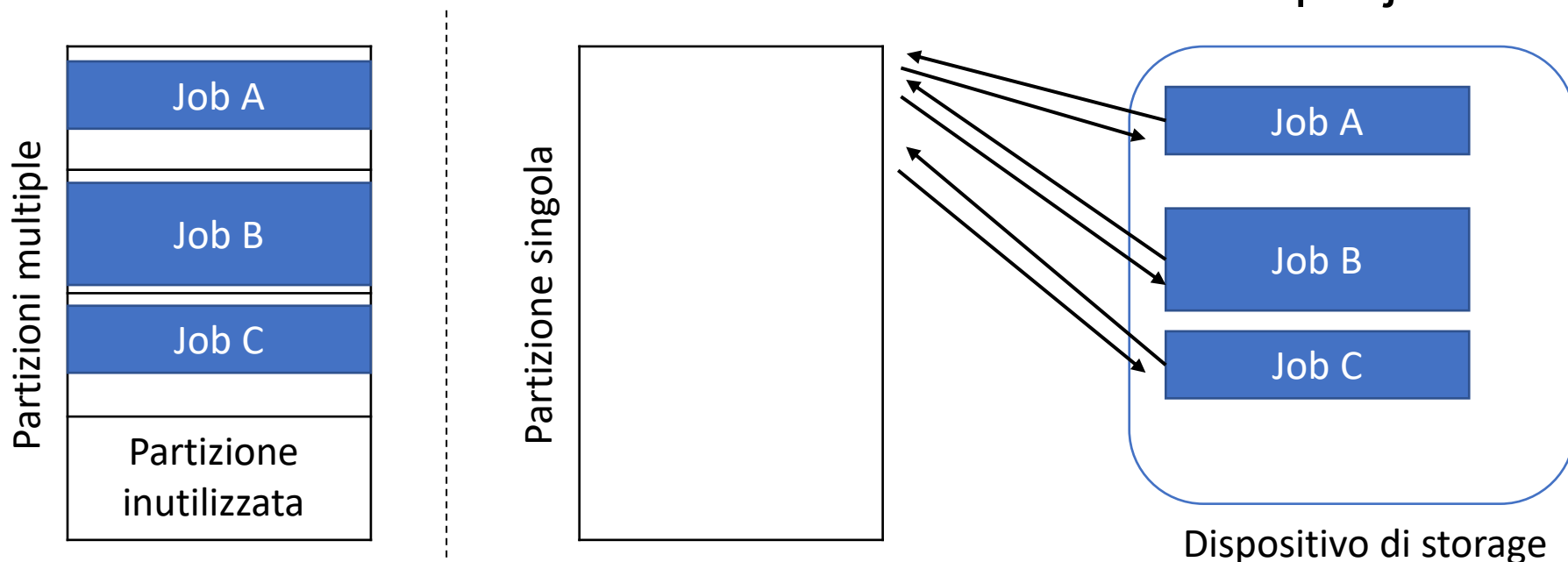
Prospettiva storica

- **mid 50s – late 60s: Batch System**

- Batch Multiprogramming più job in contemporanea, ci serve che il monitor...

Il monitor:

- è in grado di cedere il controllo da un job all'altro
- maschera le latenze di I/O con l'esecuzione di altri job
- necessita di mantenere in memoria lo stato di più job



Prospettiva storica

- mid 50s – late 60s: Batch System

CPU	Run Job A		Run Job A		Run Job B		Run Job B		Run Job C	
DISK		I/O		I/O		I/O		I/O		I/O

$$\text{Utilizzazione CPU} = \frac{5}{12} = 42\%$$

$$\text{Utilizzazione I/O} = \frac{7}{12} = 58\%$$

CPU	Run Job A	Run Job B	Run Job C	Run Job A	Run Job B	
DISK		I/O		I/O	I/O	I/O

$$\text{Utilizzazione CPU} = \frac{5}{8} = 62\%$$

$$\text{Utilizzazione I/O} = \frac{7}{8} = 87\%$$

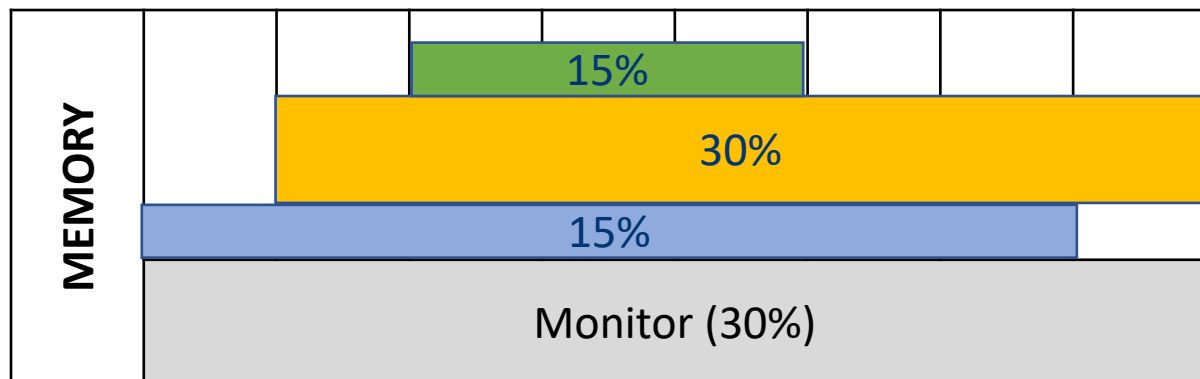
tutta l'esecuzione dei job termina prima e si comincia ha usare tutto l'hardware

TEMPO
MACCHINA/DEVICE
UTILE PER SERVIRE
ALTRI JOB

Prospettiva storica

- mid 50s – late 60s: Batch System

si aumenta l'utilizzazione



$$\text{Utilizzazione Memoria} = \frac{45\%}{8} + \frac{3 \cdot 75\%}{8} + \frac{3 \cdot 90\%}{8} + \frac{60\%}{8} = 75\%$$

CPU	Run Job A	Run Job B	Run Job C	Run Job A	Run Job B	
DISK		I/O	I/O	I/O	I/O	I/O

$$\text{Utilizzazione CPU} = \frac{5}{8} = 62\%$$

$$\text{Utilizzazione I/O} = \frac{7}{8} = 87\%$$

- **mid 50s – late 60s: Batch System**

- Batch Multiprogramming
- L'hardware ha un ruolo fondamentale nell'avanzamento dei sistemi operativi:
 - **Direct Memory Access** (trasferimenti in memoria non richiedono l'uso della CPU)
 - **Interrupt-driven I/O** (permette al processore di essere «avvisato» riguardo l'avanzamento delle operazioni di I/O)
- Multiprogramming OS sono più complessi:
 - Memory Management
 - Politiche di scheduling più avanzate
- Limiti:
 - Alcuni classi di job possono essere svantaggiate (e.g., I/O bound) perche usano più dispositivi di I/O rispetto ad altri
 - Impossibile supportare applicazioni interattive perché l'evento è da un job ad un altro

Prospettiva storica

- **late 60s – present Time-Sharing System:**

- Piuttosto che massimizzare l'utilizzo del processore si tenta di minimizzare il tempo di risposta
- Il tempo di CPU viene preassegnato ai job secondo un certo algoritmo di **scheduling** (e.g. round robin)
- I job possono essere interrotti anche se non hanno fatto richieste di I/O (e.g. allo scadere del quanto di tempo assegnatogli)
- Condivisione delle risorse **trasparente**:
 - I job non sono consci di essere interrotti e riesumati
 - I job non sanno di condividere risorse con altri job (e.g., processore o dispositivi di I/O)

➡ Molteplici utenti possono accedere al sistema da più terminali

➡ Introduzione del concetto di processo

Sistemi Operativi Moderni

Molteplici linee di evoluzione:

- (soft-)real time esistono delle deadline temporanei che devono essere rispettati, molto complessi.
- Multicore
- Energy awarness

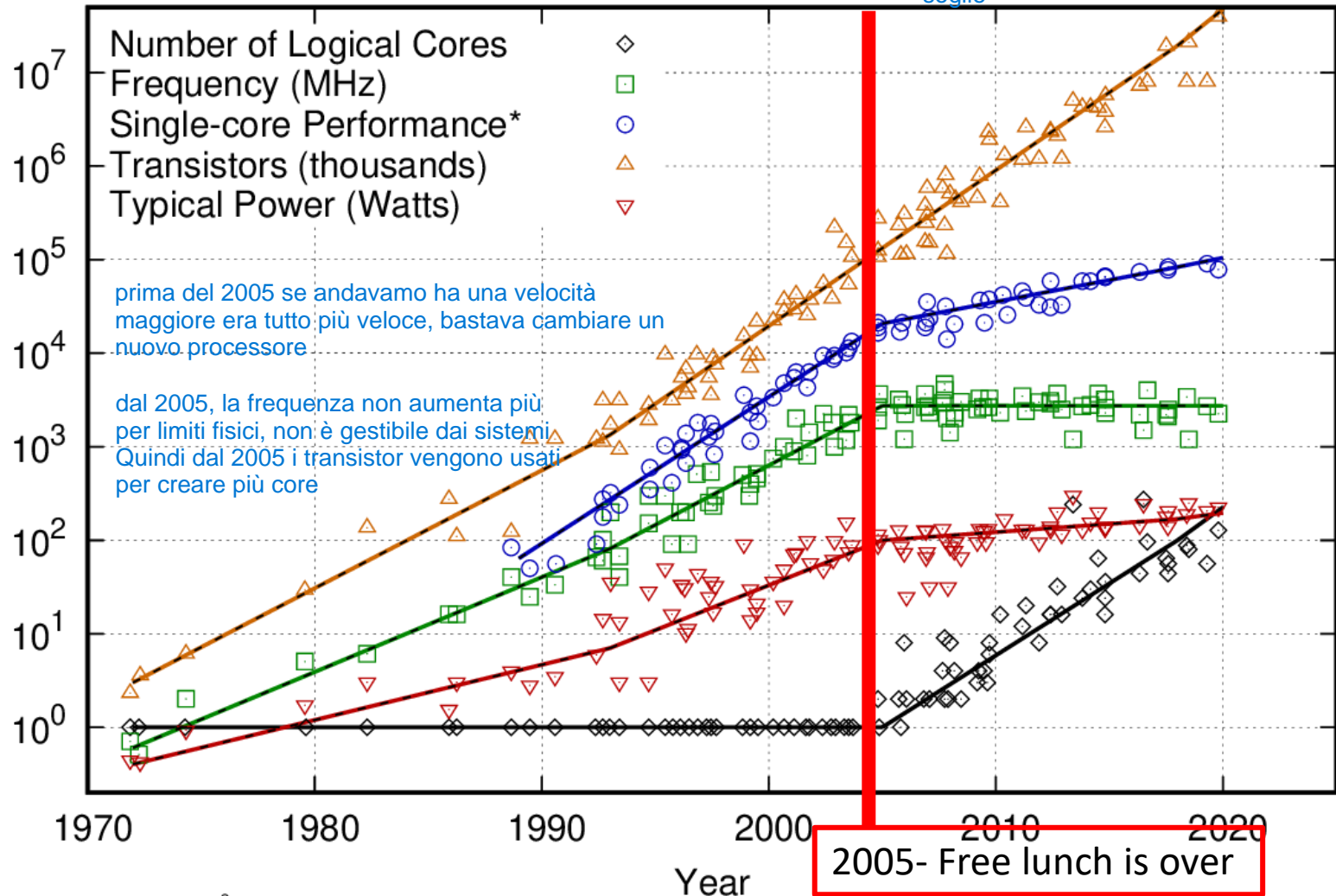
Sistemi Real-time

- Il SO deve garantire che determinate attività completino entro una specifica scadenza temporale (deadline)
- Hard real-time:
 - La deadline **deve** essere rispettata
 - Utile in sistemi vitali e/o industriali
 - Tipicamente non supportato da sistemi timesharing
- Soft real-time:
 - La deadline **dovrebbe** essere rispettata
 - Utile in applicazioni che richiedono specifiche latenze per determinate attività (e.g. registrazione e monitor audio)

deadline = scadenza temporale

Trend tecnologici delle CPU moderne

power wall perche non possiamo superare certe soglie

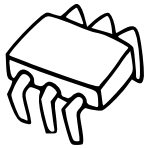


*SpecINT x 10³

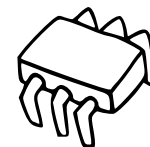
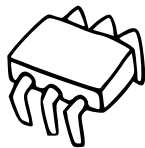
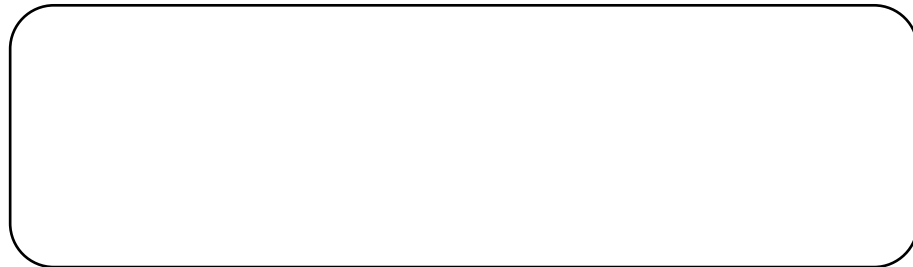
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2018 by K. Rupp

Trend tecnologici delle CPU moderne

- Implicazioni: più core in parallelo, creano dei conflitti. Una possibile soluzione è produrre meccanismi di sincronizzazione
 - Più processi possono eseguire effettivamente in parallelo ed accedere a più risorse
 - Necessità di sincronizzare tali accessi avendo come target non solo la correttezza, ma anche le performance



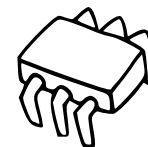
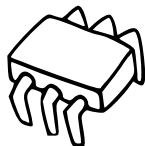
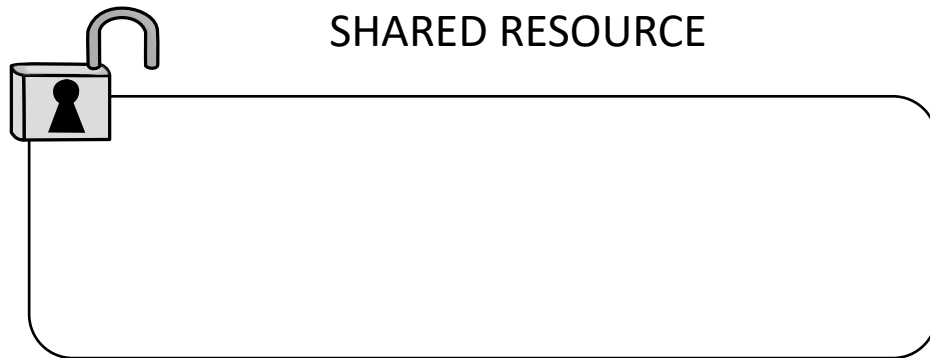
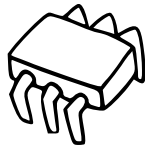
SHARED RESOURCE



Trend tecnologici delle CPU moderne

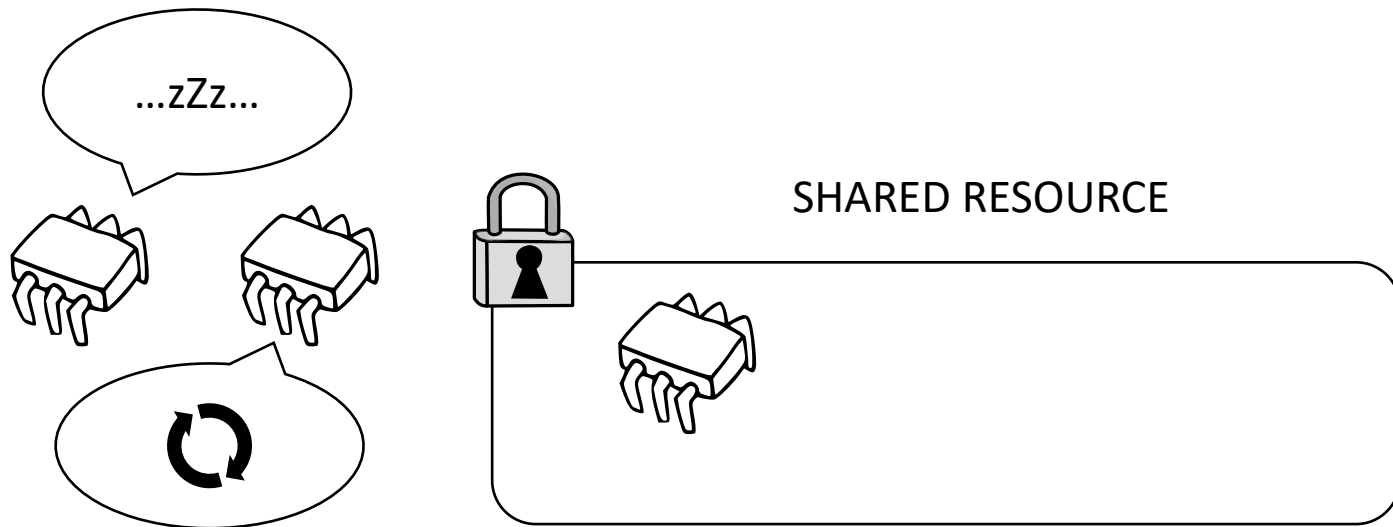
- Implicazioni:
 - Più processi possono eseguire effettivamente in parallelo ed accedere a più risorse
 - Necessità di sincronizzare tali accessi avendo come target non solo la correttezza, ma anche le performance

lock: ammette solo un core processo mentre gli altri sono in attesa, che può essere attiva oppure che sia sleep



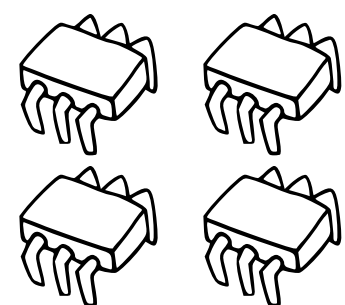
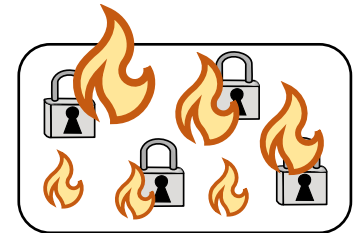
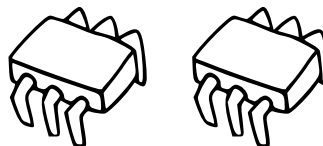
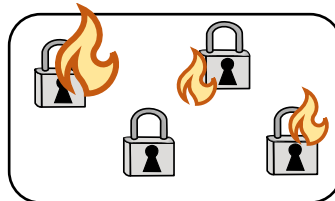
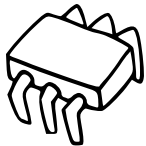
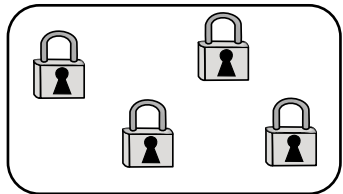
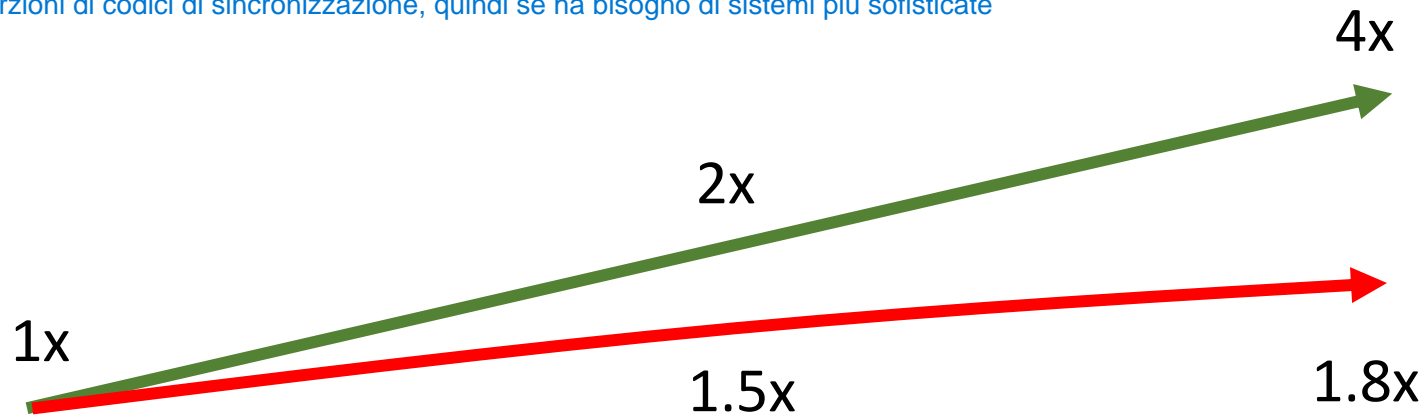
Trend tecnologici delle CPU moderne

- Implicazioni:
 - Più processi possono eseguire effettivamente in parallelo ed accedere a più risorse
 - Necessità di sincronizzare tali accessi avendo come target non solo la correttezza, ma anche le performance



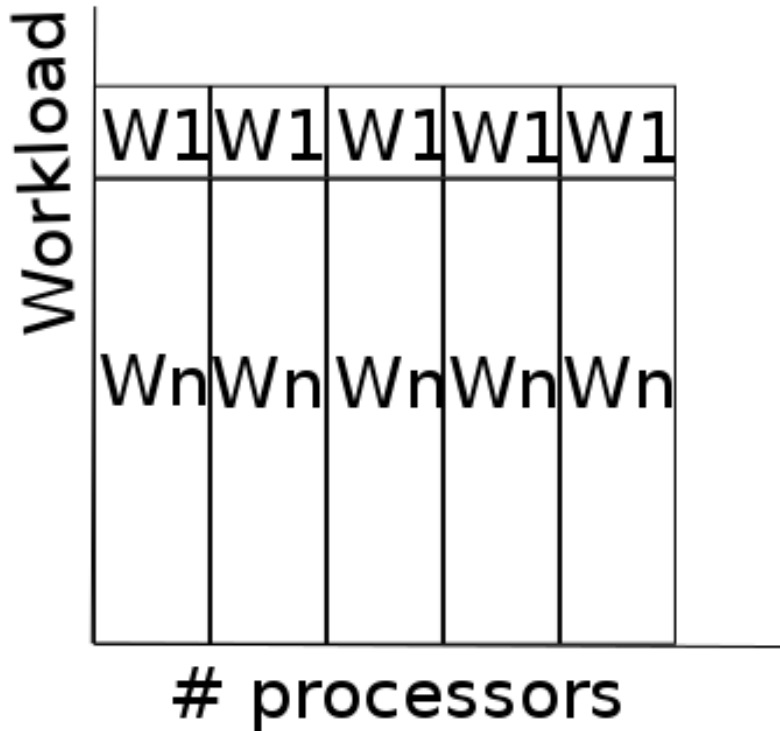
Impatto della sincronizzazione

quando passo da un processore un core a due core, lo speedup diventa 2x ma in realtà non è così perché nel software ci sono delle porzioni di codici di sincronizzazione, quindi se ha bisogno di sistemi più sofisticate

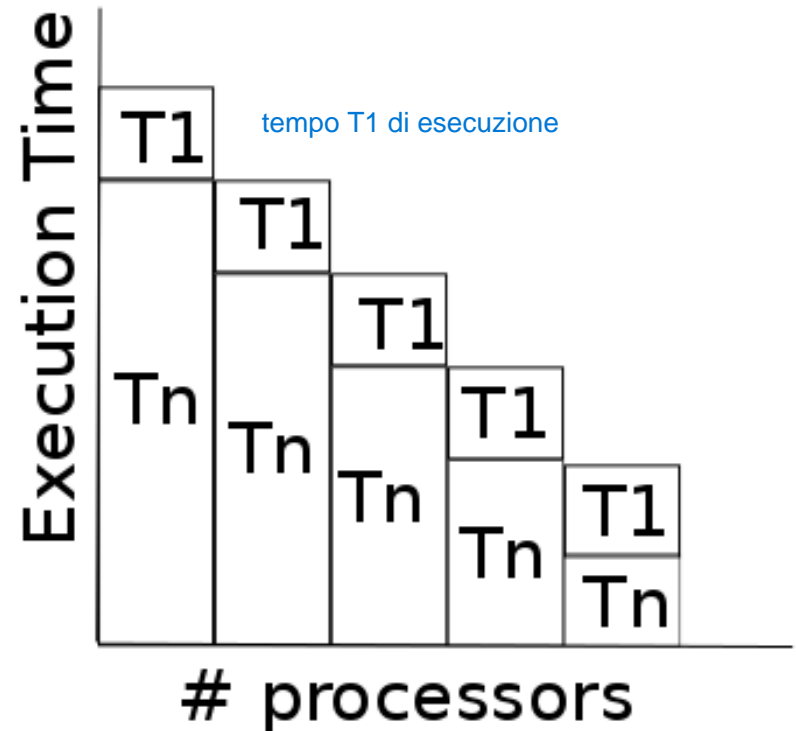


Amdahl Law – Fixed-size Model (1967)

+ processori , come cambia il tempo di esecuzione complessivo all'aumentare dei core



workload ci dice delle attività che sono necessario eseguire in maniera sequenziale ossia protetto di qualche tipo di sincronizzazione, quella parte di workload richiede un tempo t_1 di esecuzione che rimane uguale indipendentemente dei processori



la parte di W_n è parallelizzabile quindi può trarre dei vantaggi del fatto che andiamo ad aumentare il numero di processori

Amdahl Law – Fixed-size Model (1967)

2586

una linea guida per scegliere da dove ottimizzare, dal processore che prende più tempo

- Il workload è fissato: modella il comportamento di un programma al variare del numero di processori

se α è zero possiamo che lo speedup è parallelizzabile

$$S_{Amdahl} = \frac{T_s}{T_p} = \frac{T_s}{\alpha T_s + (1 - \alpha) \frac{T_s}{p}} = \frac{1}{\alpha + \frac{(1 - \alpha)}{p}}$$

- dove:

α : percentuale che non riusciamo parallelizzare

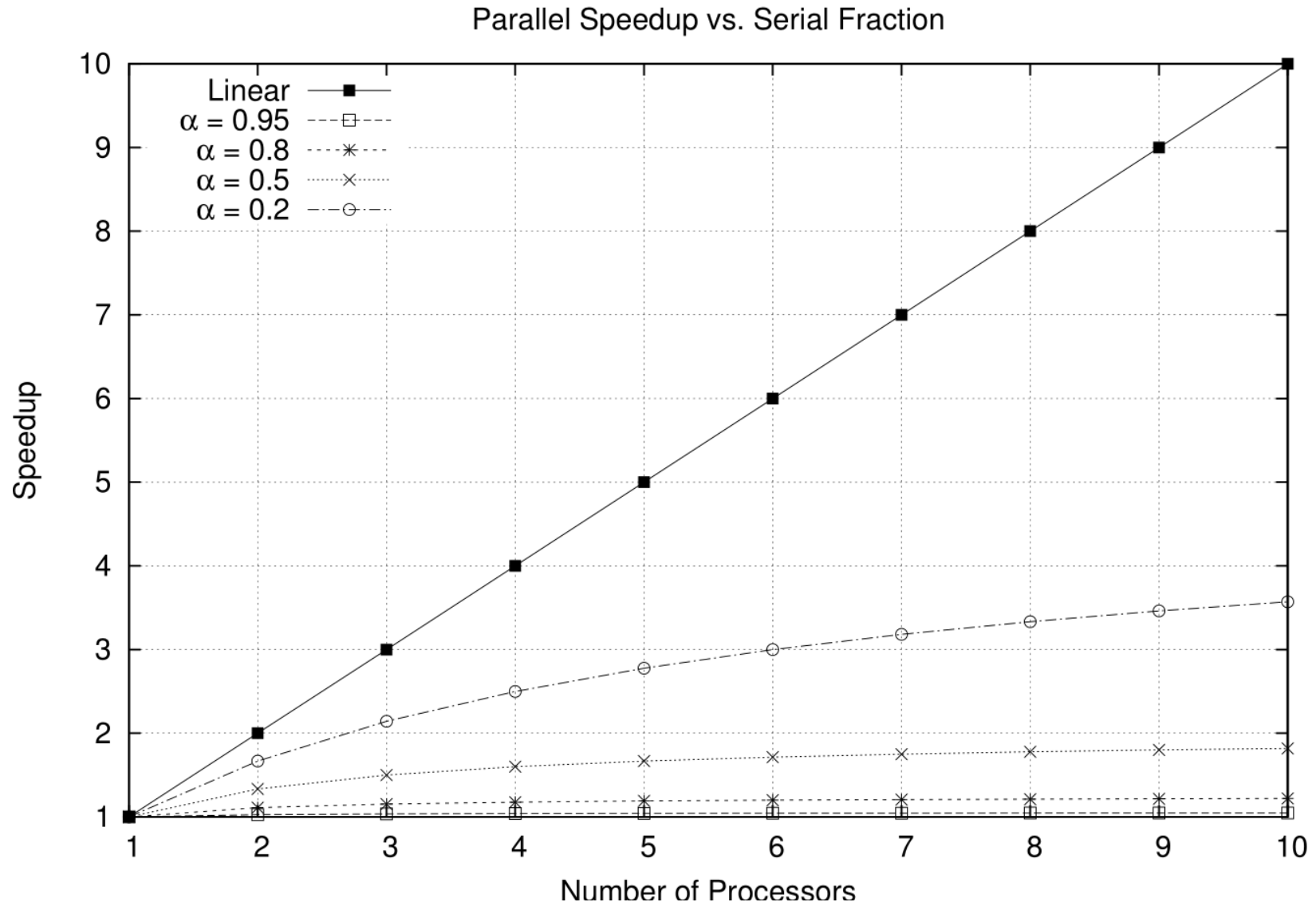
$(1 - \alpha) T_s$: è un tempo che in questo modello viene distribuito in tutti i processori

- $\alpha \in [0,1]$: frazione del programma non parallelizzabile
- $p \in N$: numero di processori
- T_s : tempo di esecuzione sequenziale
- T_p : tempo di esecuzione in parallelo

SPEED UP massimo -> processore infiniti

la legge ci dice quanto andiamo a guadagnare quando andiamo a girare con un numero maggiore di 1 di core

Amdahl Law – Fixed-size Model (1967)



Amdahl Law – Fixed-size Model (1967)

$$\lim_{p \rightarrow \infty} S_{Amdahl} = \lim_{p \rightarrow \infty} \frac{1}{\alpha + \frac{(1 - \alpha)}{p_{\text{rocessori}}}} = \frac{1}{\alpha}$$

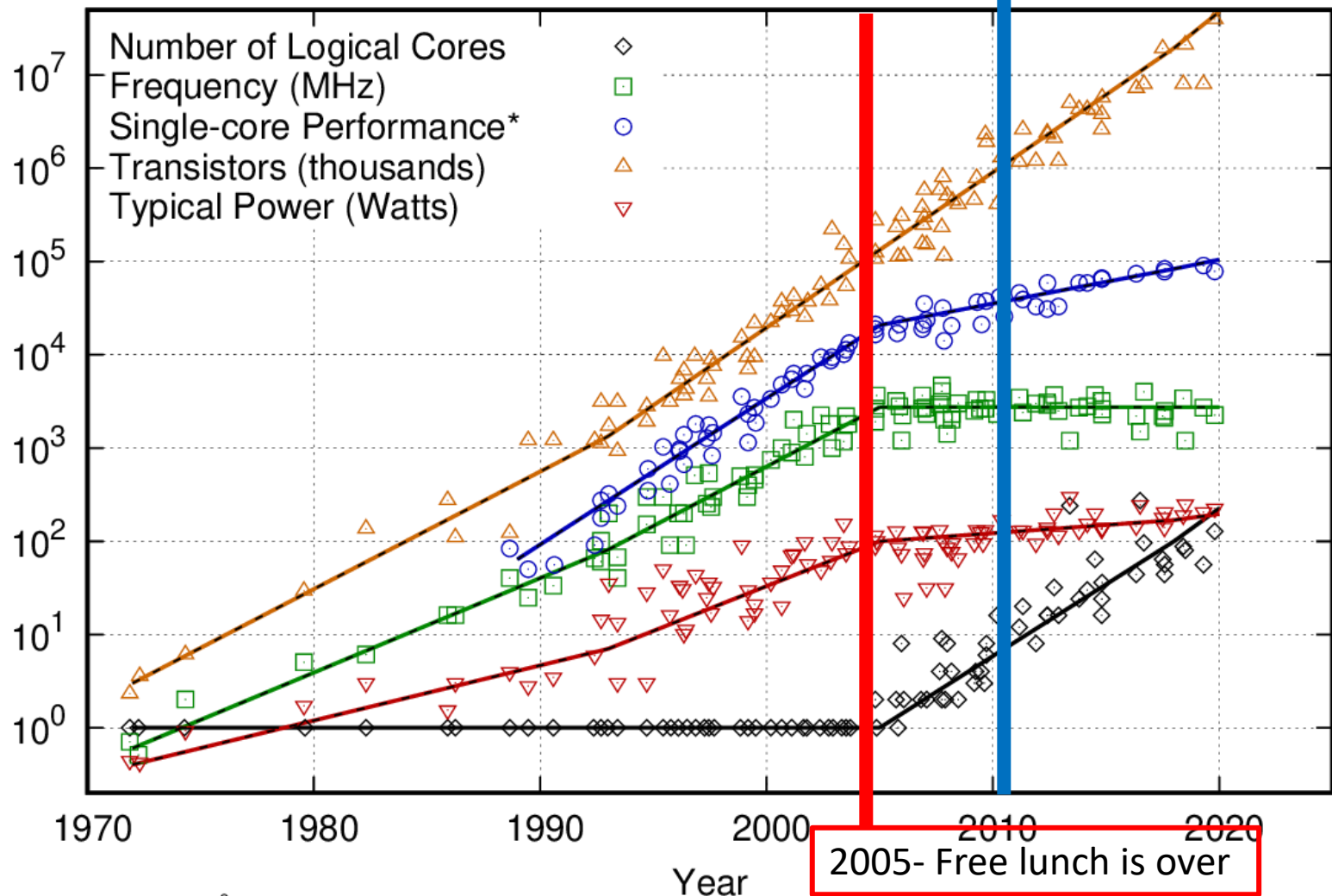
- Se la percentuale sequenziale è 20%:

$$\lim_{p \rightarrow \infty} S_{Amdahl} = \frac{1}{0.2} = 5$$

- Speedup 5 usando un numero infinito di processori!

CPU e SO

Dal 2011 Linux supporta in modo efficiente SMP (rimosso Big Kernel Lock)

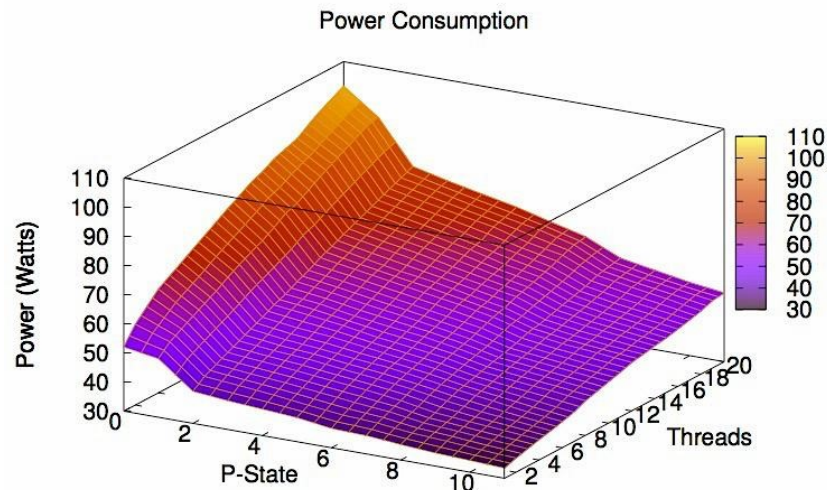


*SpecINT x 10³

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2018 by K. Rupp

Trend tecnologici delle CPU moderne

- Implicazioni:
 - Più processi possono eseguire effettivamente in parallelo ed accedere a più risorse
 - Necessità di sincronizzare tali accessi avendo come target non solo la correttezza, ma anche le performance
- Per quanto ancora questo trend?



il grafico ci da il consumo di potenza
al variare core attivi e frequenza

Conoci et al

Adaptive Performance Optimization under Power Constraint in Multi-thread Applications with Diverse Scalability
ICPE'18

Dark silicon

- Non è possibile mantenere attive tutte le unità computazionali
- Dark silicon: unità computazionali che non potranno essere attive per rispettare limiti di consumi di potenza (Thermal Design Power)
- La percentuale di dark silicon aumenterà nel tempo
 - Meccanismi di gestione dei consumi sempre più avanzati
 - Efficienza energetica tramite utilizzo di unità hardware specializzate in un insieme ridotto di funzionalità (co-processori, FPGA, GPU)

Sistemi Operativi Moderni

Molteplici linee di evoluzione:

- (soft-)real time
- Multicore
- Energy awarness

Elementi chiave:

- Interrupt-driven
- Protezione delle risorse
- Gestione dei processi
- Gestione della memoria
- Gestione di I/O e file
- Virtualizzazione

Interrupt-driven operating systems

- I sistemi batch necessitano di un meccanismo per permettere al monitor di acquisire il controllo in determinate situazioni:
 - timer scaduti
 - eseguire istruzioni privilegiate per conto dei job
 - gestione di errori software
- **Interrupt:**
 - Meccanismo per interrompere il corrente flusso di esecuzione del processore all'occorrenza di determinati eventi ed eseguire una routine di gestione dell'interrupt (**interrupt handler**)
- I sistemi operativi moderni sono **interrupt-driven**
 - Le interruzioni regolano le interazioni tra SO, dispositivi e programmi utente

Tipi di interruzioni

GENERATA DA	CAUSA	DESCRIZIONE
EVENTI ASINCRONI	INTERRUPT	
EVENTI SINCRONI	TRAP	

Tipi di interruzioni

GENERATA DA	CAUSA	DESCRIZIONE
EVENTI ASINCRONI <small>che accadono, indipendentemente di quello che succede nel processore(es. staccare il cavo di alimentazione,...)</small>	Hardware failure	generata da un problema rilevato via hardware (e.g. alimentazione scarsa, errore ECC in RAM)
	I/O	generata da un controller di I/O (e.g. terminazione di una operazione di I/O, condizione di errore del device)
	Timer	generata da un clock interno del processore
EVENTI SINCRONI <small>che dipende di quello che succede nel processore dall'istruzione che sta eseguendo il processore.</small>	Programma	generata da qualche condizione verificatasi a causa dell'esecuzione di una istruzione (e.g. divisione per zero, accesso a memoria protetta)

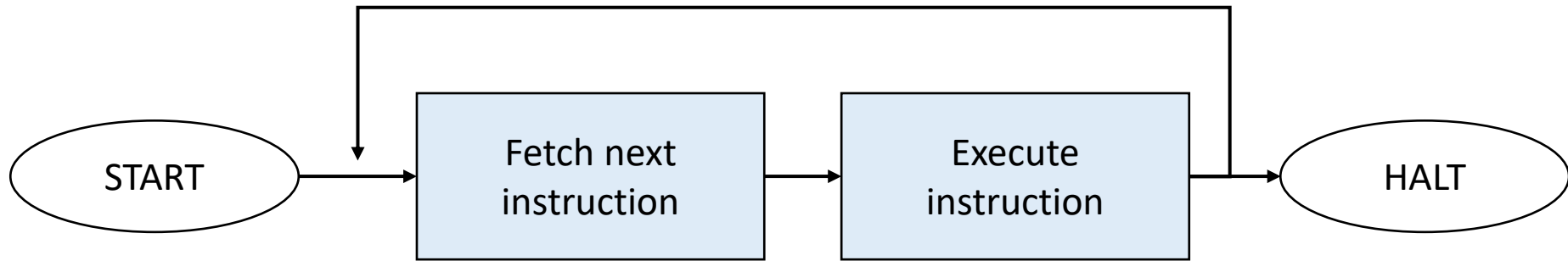
- Diverse interruzioni possono richiedere gestioni diverse
- Gli handler possono a loro volta subire interruzioni

Supporti alle interruzioni

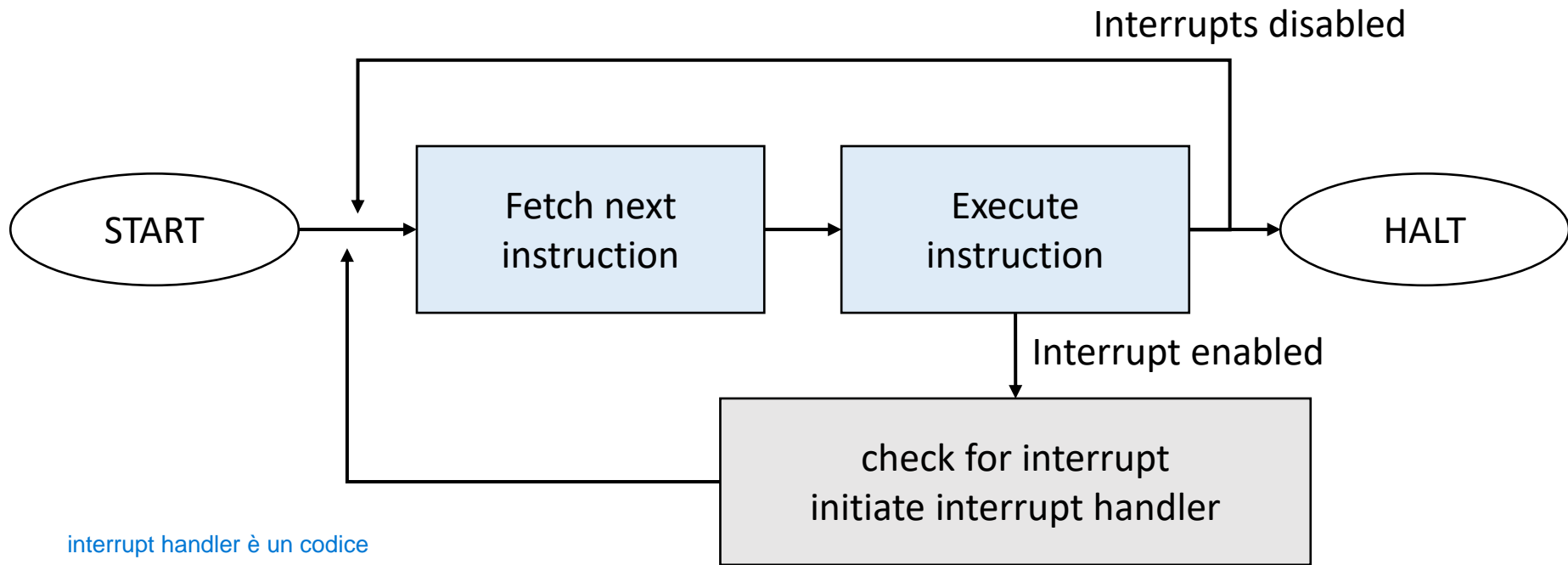
- **Ripasso: Esecuzione di un'istruzione**
 - **Stadi:**
 - Fetch
 - Decodifica
 - Esecuzione
 - Accesso a memoria
 - Write back su registri

Supporti alle interruzioni

- **Ripasso: Esecuzione di un'istruzione**
 - Uno schema semplificato



Supporti alle interruzioni



interrupt handler è un codice

controlla se al termine dell'esecuzione ce una interruzione
se c'è si va ad eseguire un interrupt handler

Supporti alle interruzioni

due modi per individuare la routine:

- **Polled:** il processore controlla tutti i dispositivi/componenti che possono lanciare interrupt per vedere cosa ha il segnale di interrupt asserito
- **Vectored:** l'interrupt ha associata un codice. Quest'ultimo è utilizzato per accedere ad un vettore che mantiene l'associazione tra codice e handler

interruzione

Program A

Il firmware salva informazioni sullo stato di processore in specifiche locazioni di memoria

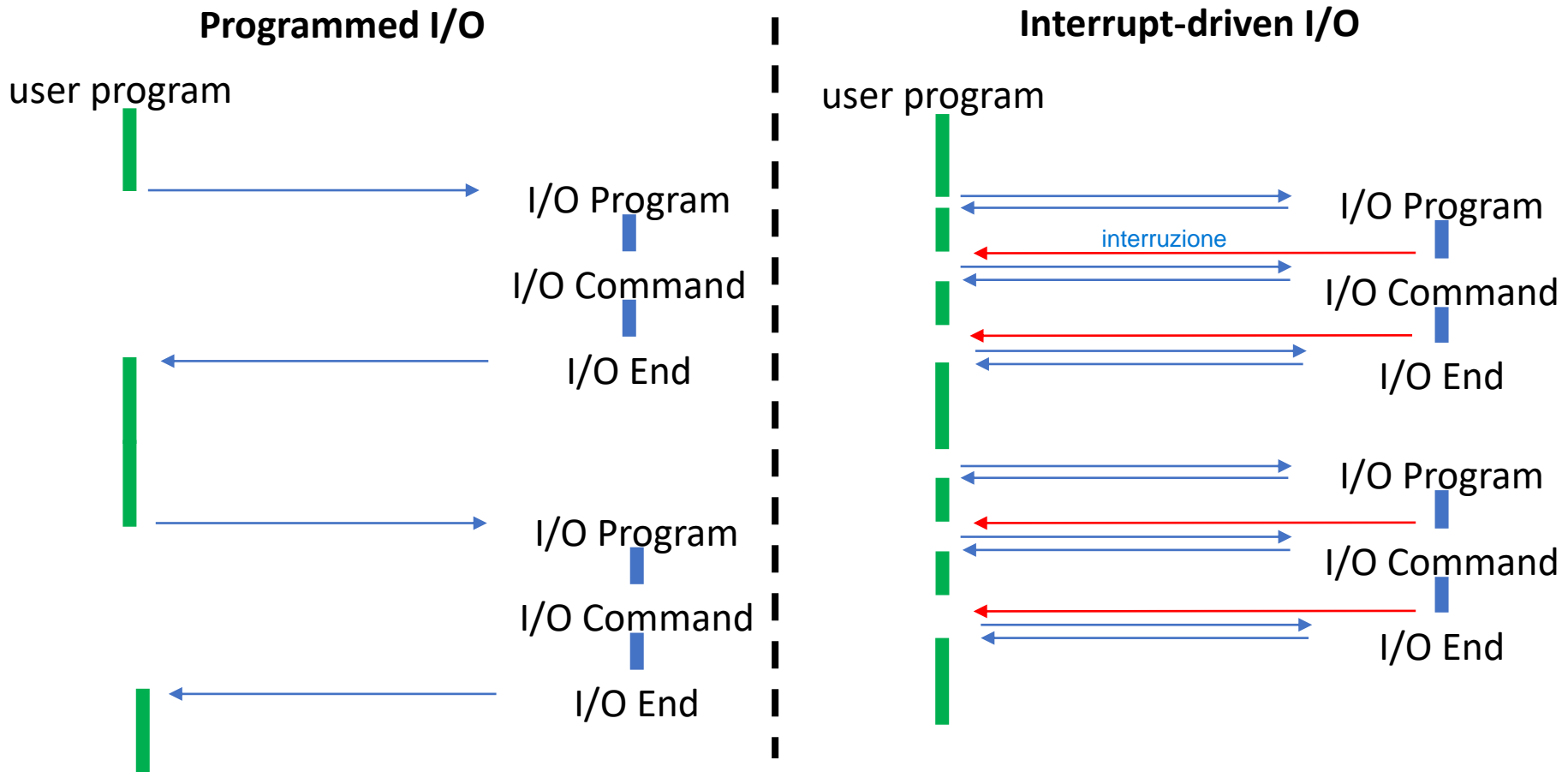
Il firmware individua la routine di sistema per la gestione dell'interrupt

Interrupt handler

Interrupt-driven operating systems

I Sistemi Operativi moderni:

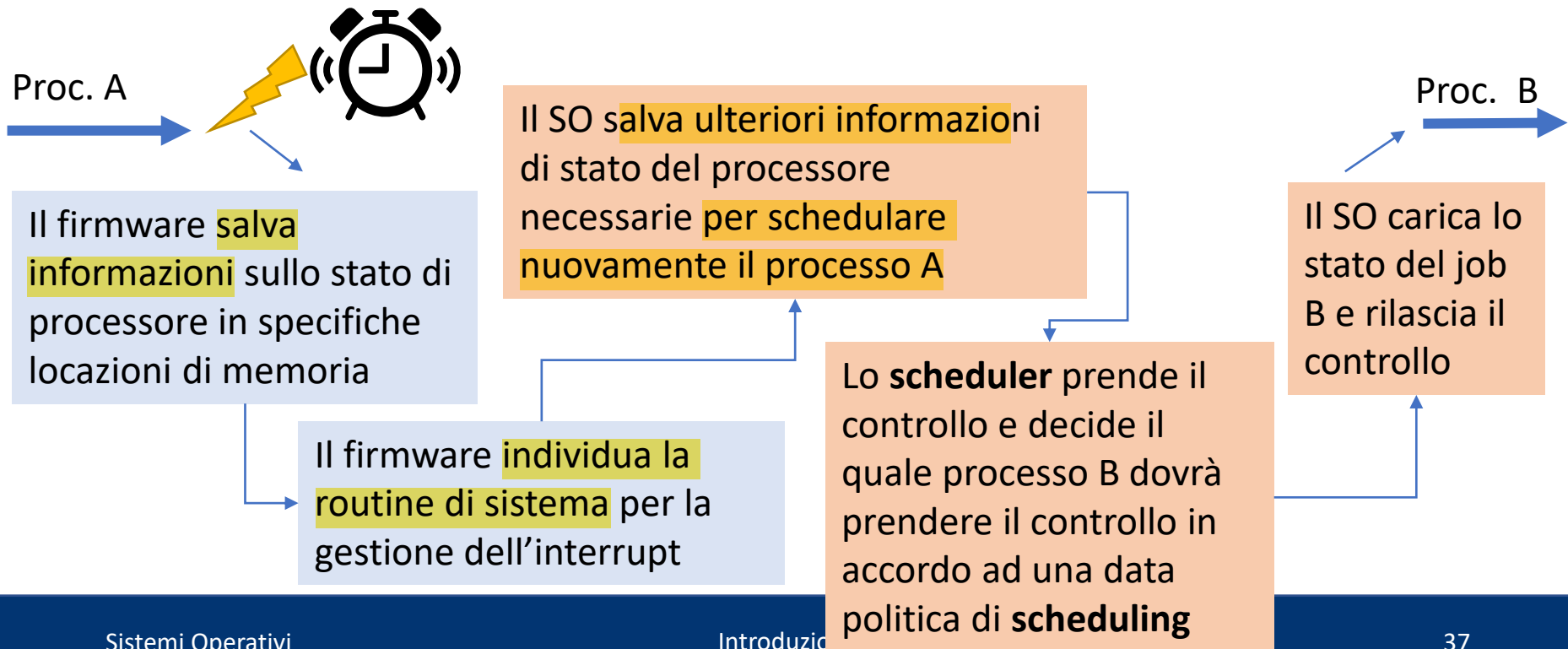
- Sono Interrupt-driven I/O



Interrupt-driven operating systems

I Sistemi Operativi moderni:

- Sono Interrupt-driven I/O
- Proteggono le risorse hardware acquisendo il controllo sotto determinate condizioni:
 - Un processo monopolizza l'uso della CPU → **Timer**



Interrupt-driven operating systems

I Sistemi Operativi moderni:

- Sono Interrupt-driven I/O
- Proteggono le risorse hardware acquisendo il controllo sotto determinate condizioni:
 - Un processo monopolizza l'uso della CPU → **Timer**
 - Proteggere l'accesso a determinate regioni di memoria (ad esempio quelle che contengono il codice del kernel e le relative strutture dati)
 - Un processo tenta di eseguire istruzioni privilegiate

rientrano

Interrupt-driven operating systems

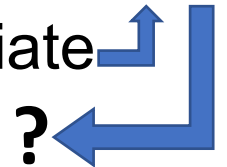
I Sistemi Operativi moderni:

- Sono Interrupt-driven I/O
- Proteggono le risorse hardware acquisendo il controllo sotto determinate condizioni:
 - Un processo monopolizza l'uso della CPU → **Timer**
 - Proteggere l'accesso a determinate regioni di memoria (ad esempio quelle che contengono il codice del kernel e le relative strutture dati) → **HW detection**
 - Un processo tenta di eseguire istruzioni privilegiate ↗

Interrupt-driven operating systems

I Sistemi Operativi moderni:

- Sono Interrupt-driven I/O
- Proteggono le risorse hardware acquisendo il controllo sotto determinate condizioni:
 - Un processo monopolizza l'uso della CPU → **Timer**
 - Proteggere l'accesso a determinate regioni di memoria (ad esempio quelle che contengono il codice del kernel e le relative strutture dati) → **HW detection**
 - Un processo tenta di eseguire istruzioni privilegiate



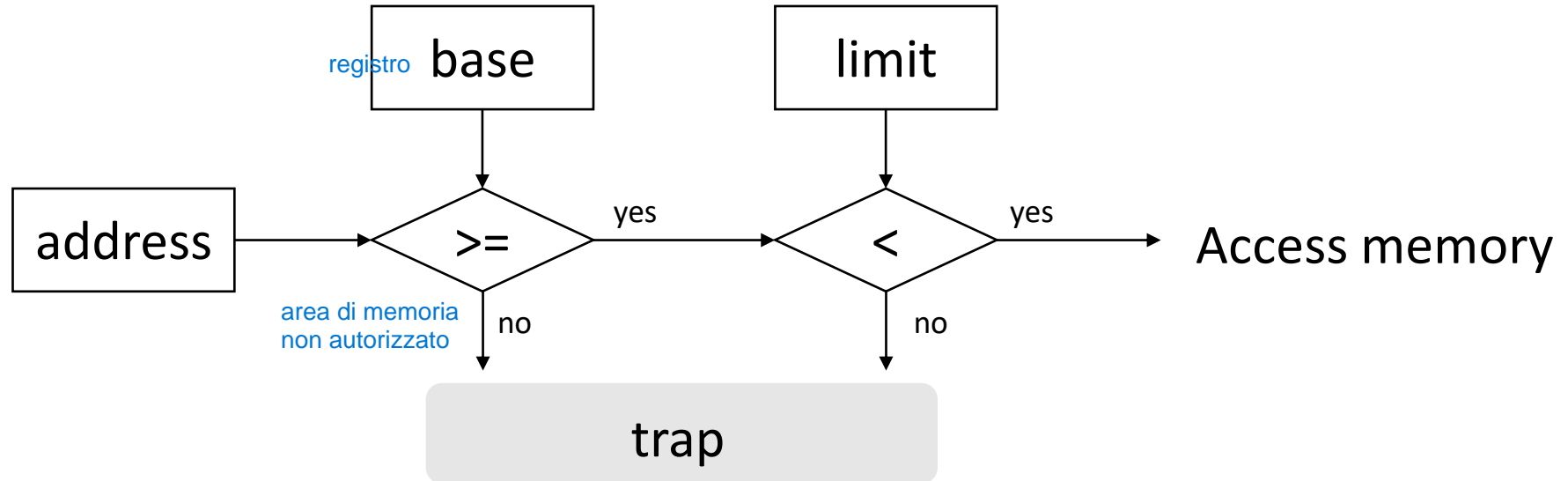
Interrupt-driven operating systems

I Sistemi Operativi moderni:

- Sono Interrupt-driven I/O
- Proteggono le risorse hardware acquisendo il controllo sotto determinate condizioni:
 - Un processo monopolizza l'uso della CPU → **Timer**
 - Proteggere l'accesso a determinate regioni di memoria (ad esempio quelle che contengono il codice del kernel e le relative strutture dati) → **HW detection**
 - Un processo tenta di eseguire istruzioni privilegiate → **Trap**

Interrupt-driven operating systems

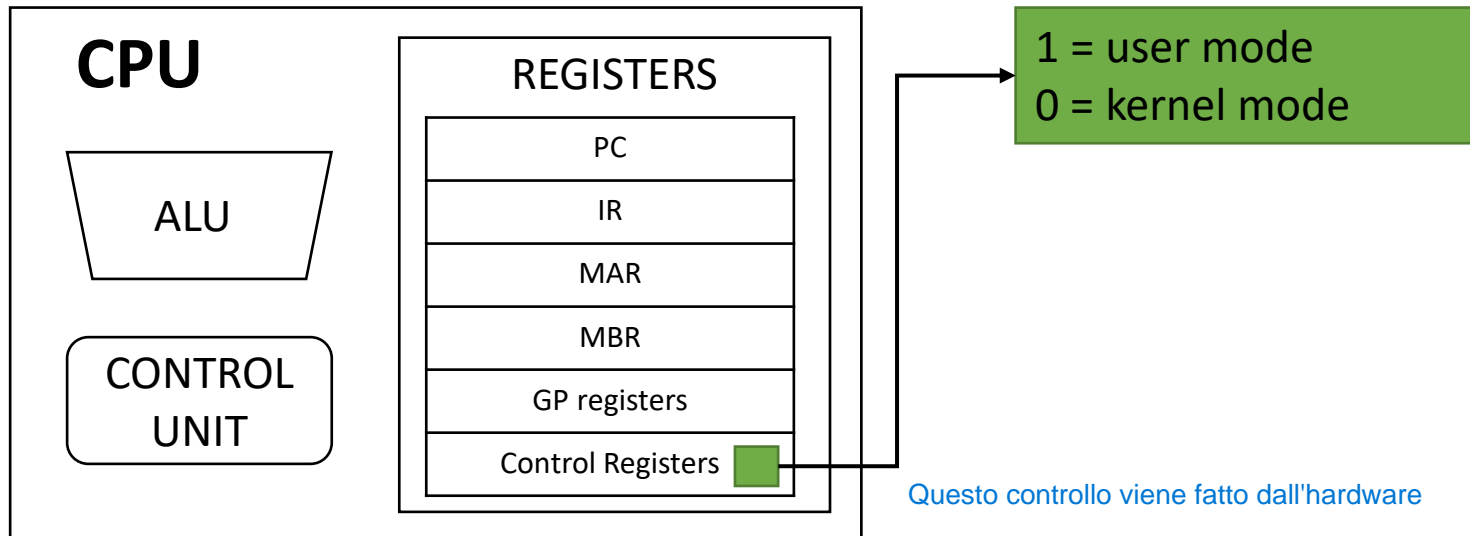
- HW memory protection



Interrupt-driven operating systems

- Istruzioni privilegiate

- Supporto hardware per avere almeno due modi esecuzione: **kernel** e **user**
- Kernel mode: nessuna restrizione sulle istruzioni ammissibili
- User Mode: alcuni istruzioni sono proibite e, se eseguite, generano una trap^e



Interrupt-driven operating systems

I Sistemi Operativi moderni:

- Sono Interrupt-driven I/O
- Proteggono le risorse hardware acquisendo il controllo sotto determinate condizioni:
 - Un processo monopolizza l'uso della CPU → **Timer**
 - Proteggere l'accesso a determinate regioni di memoria (ad esempio quelle che contengono il codice del kernel e le relative strutture dati) → **HW detection**
 - Un processo tenta di eseguire istruzioni privilegiate → **Trap**
- Forniscono delle interfacce ai programmi utente per accedere ai servizi del SO
 - **System call** permette di poter cambiare da un user mod a kernel mod

System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su ?

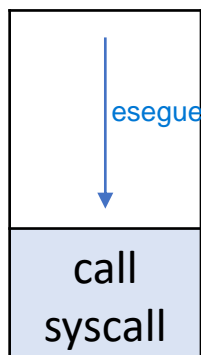
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su **trap**
 - Il SO registra (a start up) una specifica entry per gestire le syscall nel vettore di interrupt (0x80 su Linux, e 0x2E su Windows)
 - Il codice utente:
 - utente imposta i parametri per la syscall in specifici registri del processore
 - scatena una trap
 - L'interrupt handler (kernel mode):
 - assume il controllo e prende i parametri dai registri di CPU
 - invoca la reale syscall
 - rilascia il controllo al programma utente

System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap

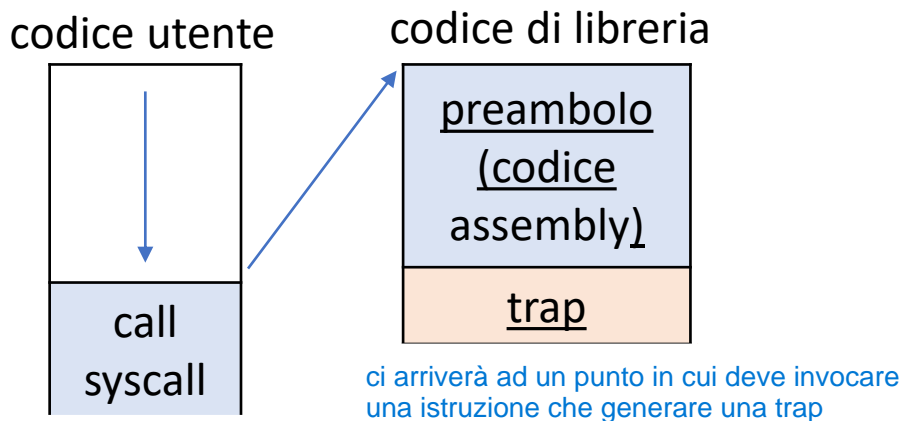
codice utente



per non generare una trap le librerie ci vengono in aiuto,
dandoci dei codici assembly

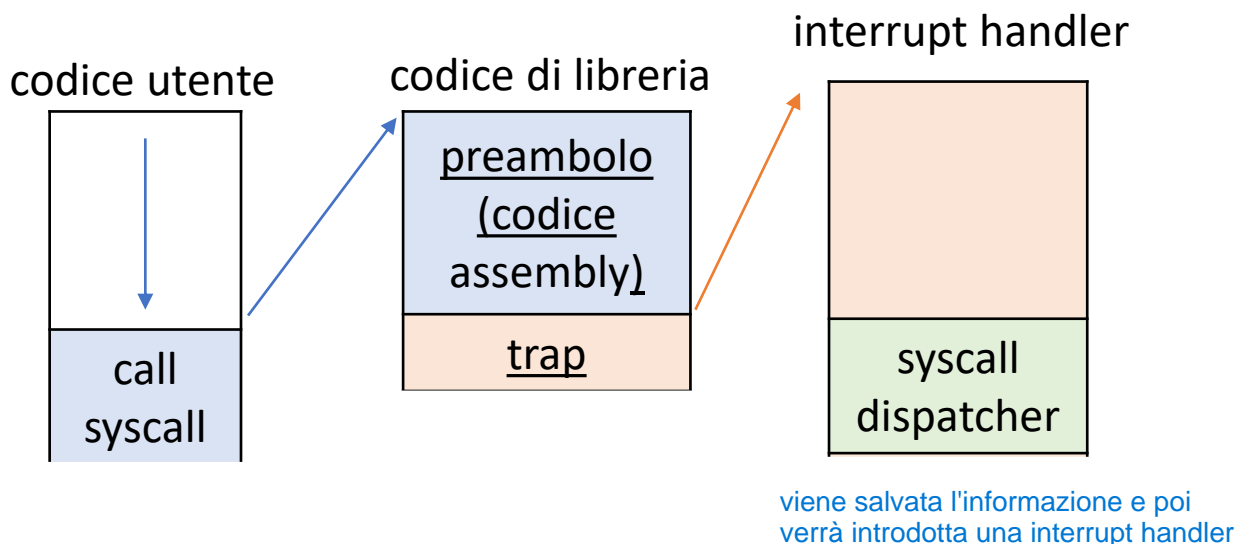
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



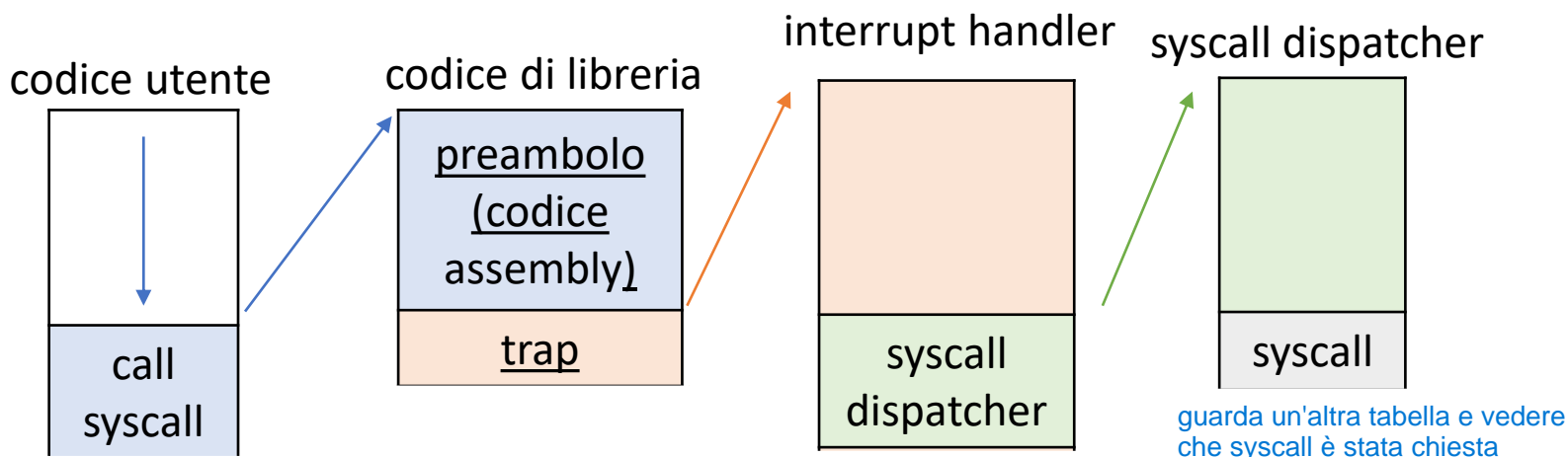
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



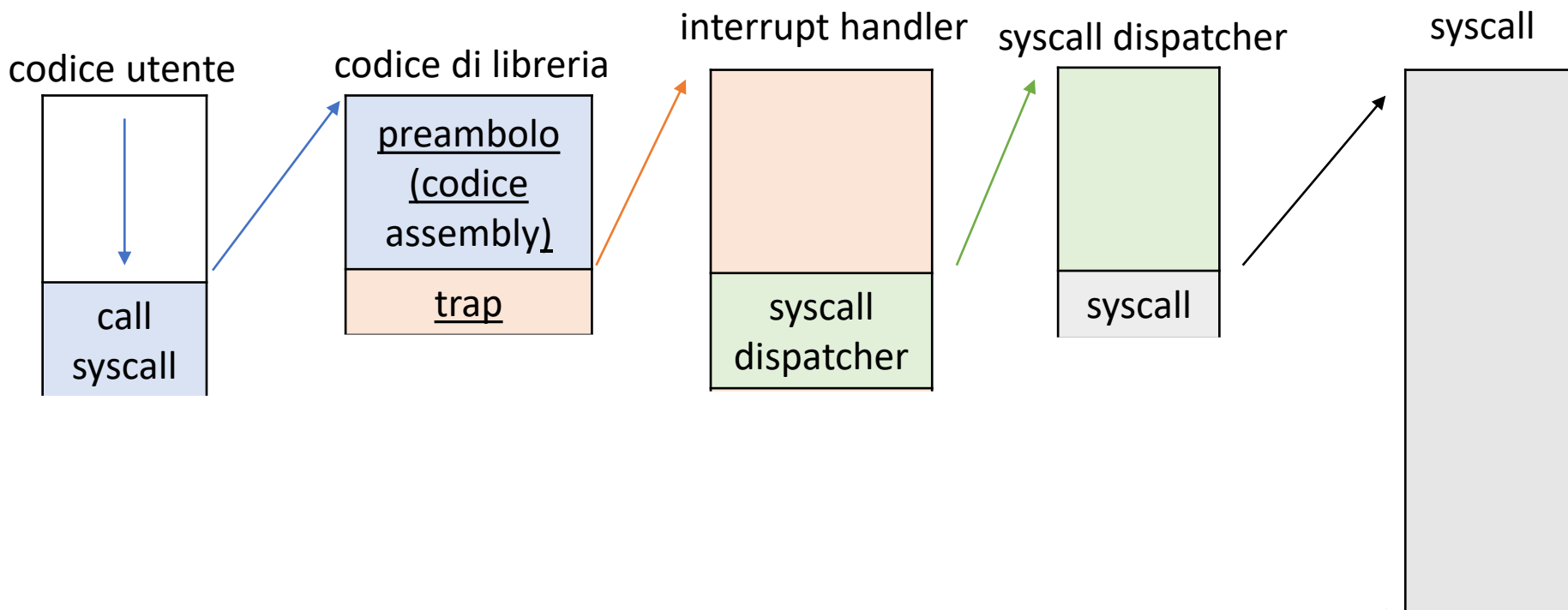
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



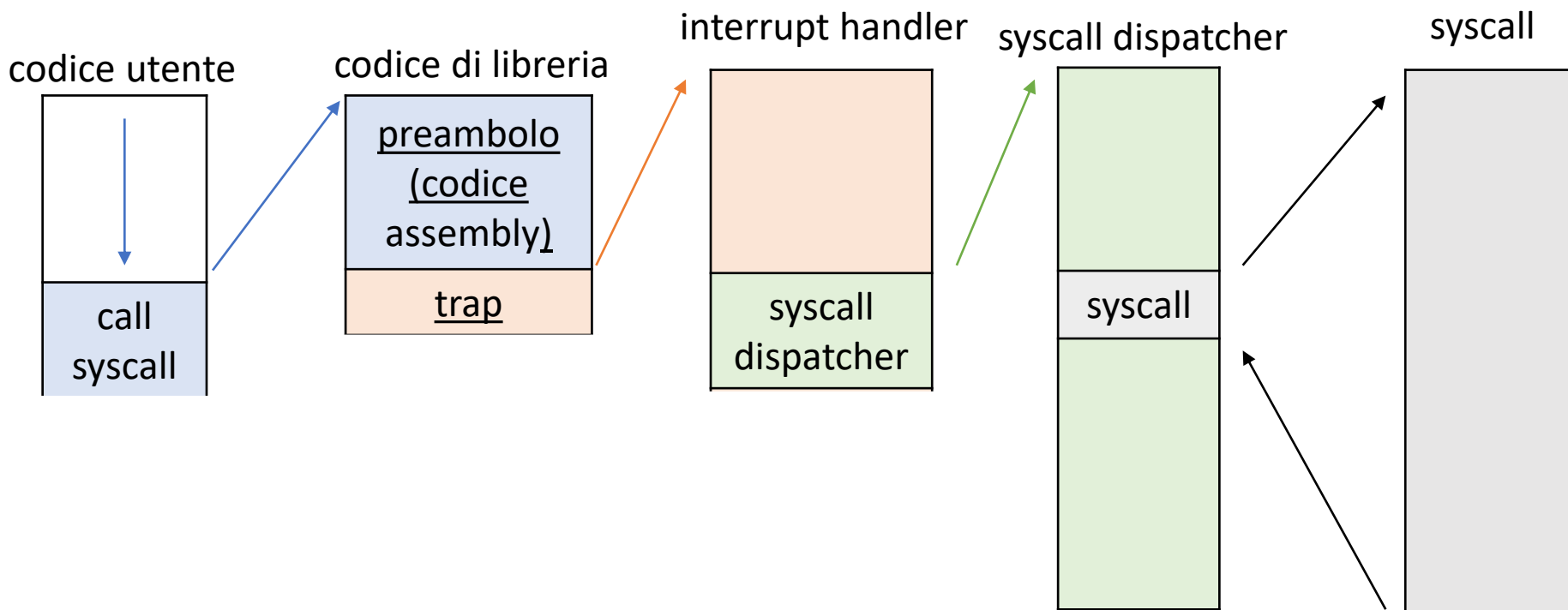
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



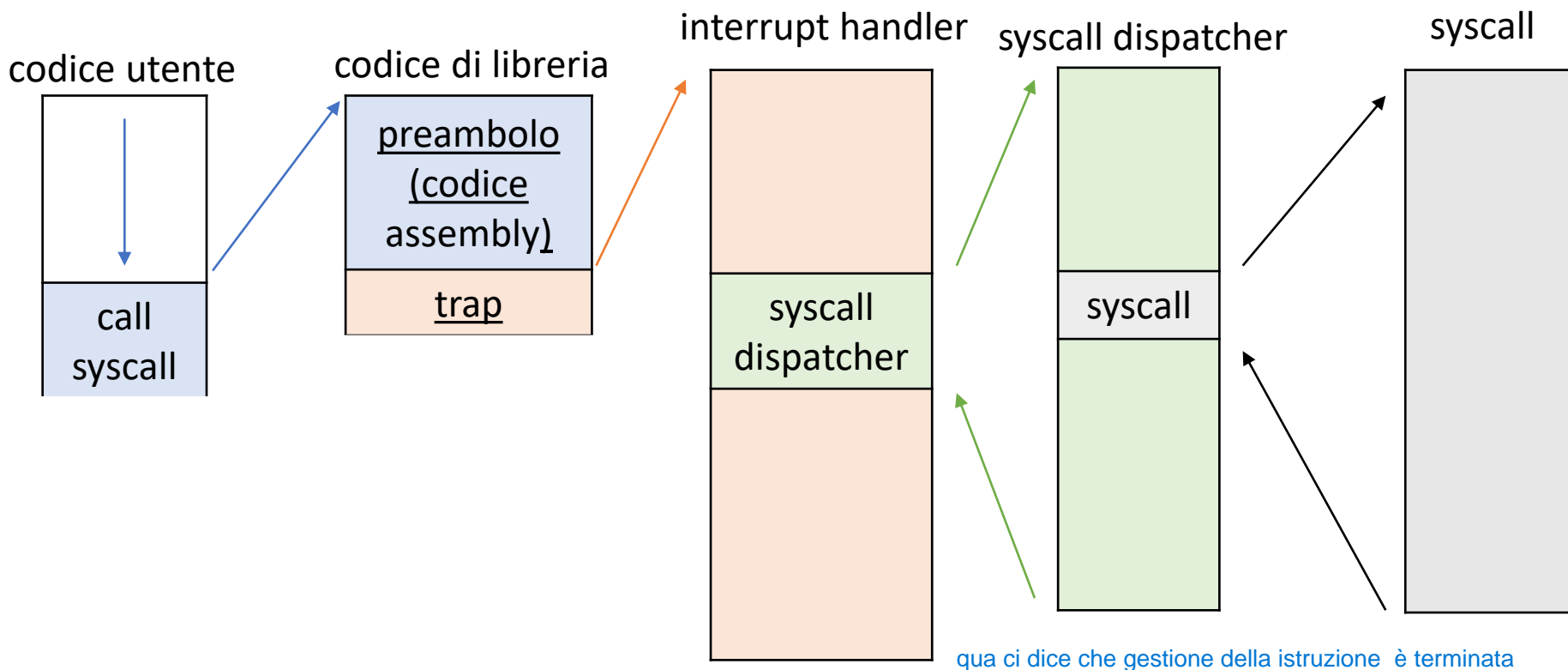
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



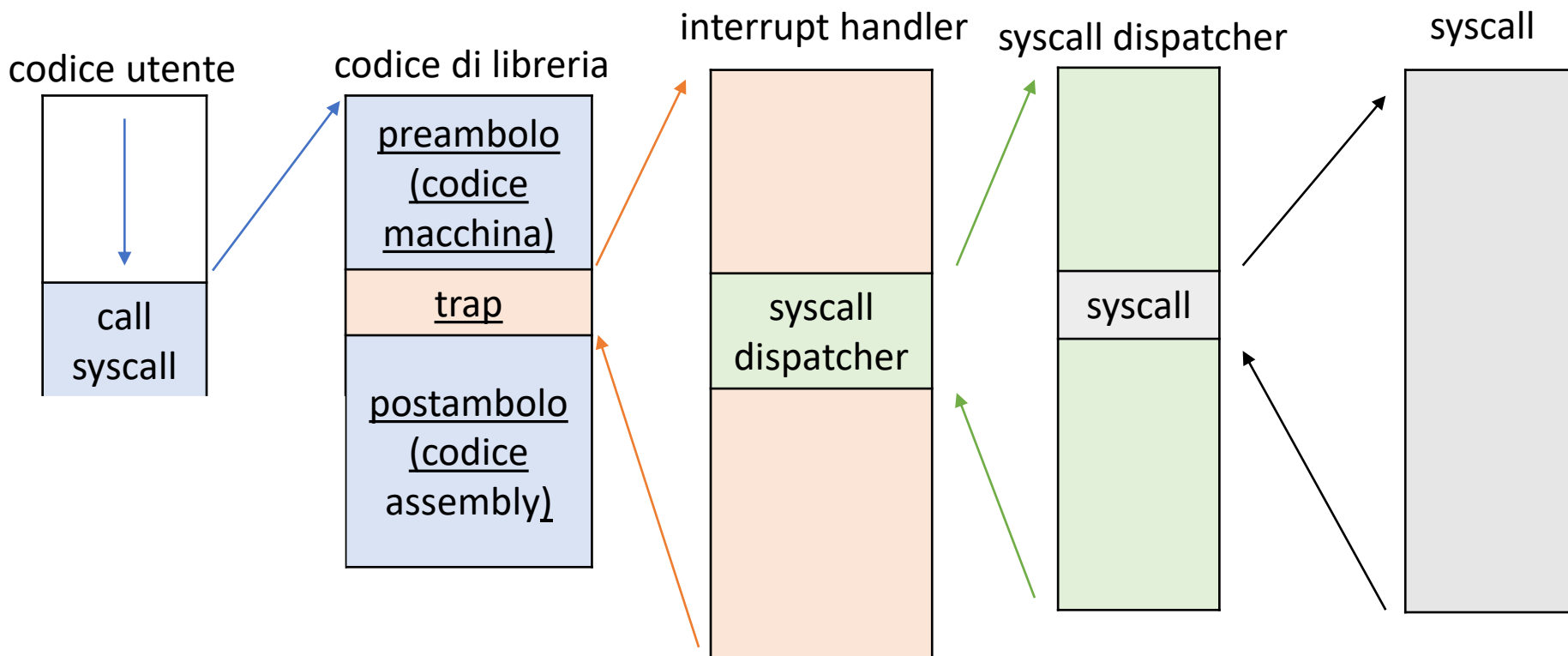
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



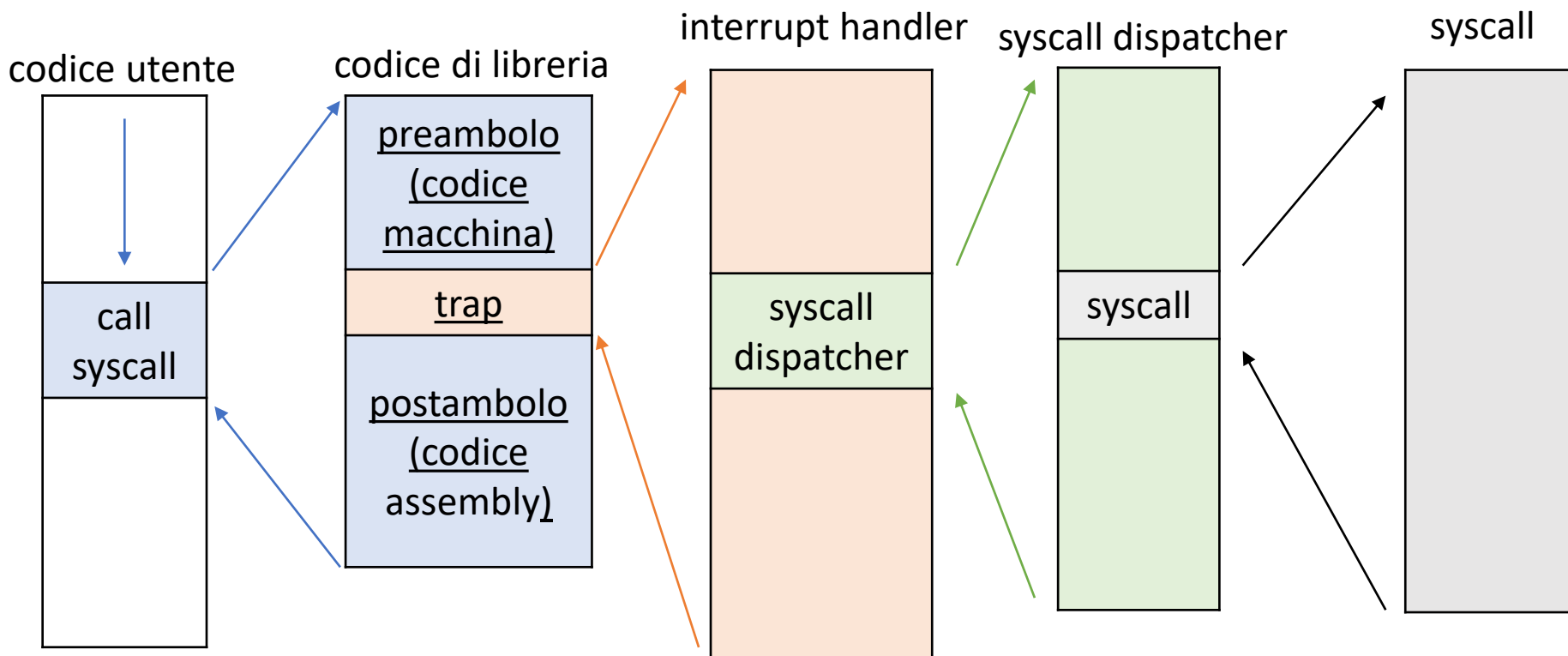
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap



System call – x86 Linux example

Interrupt Descriptor Vector (codice Kernel)

<https://elixir.bootlin.com/linux/v5.14.7/source/arch/x86/kernel/idt.c#L79>

qua definiamo un array a tempo di compilazione

```
static const __initconst struct idt_data def_idts[] = {  
    INTG(X86_TRAP_DE,          asm_exc_divide_error),  
    .  
    .
```

può essere generata **solo** in kernel mode

```
    INTG(X86_TRAP_GP,          asm_exc_general_protection),  
    .  
    .
```

può essere generata in user mode

```
    SYSG(IA32_SYSCALL_VECTOR, entry_INT80_32),  
    .
```

#define IA32_SYSCALL_VECTOR 0x80

in quale posizione si trova

System call – x86 Linux example

Codice di libreria

Nel mondo linux le syscall vengono rappresentate da numeri, quindi invocare una syscall devo usare questo preambolo mettere su %eax (un registro) el numero della syscall che voglio invocare ed esegue ENTER_KERNEL (una macro)

<https://sourceware.org/git/?p=glibc.git;a=blob;f=sysdeps/unix/sysv/linux/i386/syscall.S;>

```
ENTRY (syscall)
    PUSHARGS 6 /* Save register contents. */
    _DOARGS_6(44) /* Load arguments. */
    movl 20(%esp), %eax /* Load syscall number into %eax. */
    ENTER_KERNEL /* Do the system call. */
    POPARGS 6 /* Restore register contents. */
    cmpl $-4095, %eax /* Check %eax for error. */
    jae SYSCALL_ERROR_LABEL /* Jump to error handler if error. */
    ret /* Return to caller. */
PSEUDO_END (syscall)
```

<https://sourceware.org/git/?p=glibc.git;a=blob;f=sysdeps/unix/sysv/linux/i386/sysdep.h;#l122>

```
#define ENTER_KERNEL int $0x80
```

sono delle direttive per il preprocessore(guarda le direttive e fa delle sostituzioni testuali. ESEMPIO: cioè queste direttive vengono sostituite con tutto il testo sopra

\$0x80: vettore per poter accedere alle system call in Linux

con int se entra in modo kernel e int e quello istruzione per scatenare una trap

System call – x86 Linux example

Interrupt Handler (codice kernel)

https://elixir.bootlin.com/linux/v5.14.7/source/arch/x86/entry/entry_32.S#L969

```
SYM_FUNC_START(entry_INT80_32)
```

```
/* retrieve syscall number from eax, parameters from  
 * other registers, and setup for  
 * subsequent call  
 */
```

```
call    do_int80_syscall_32
```

```
.
```

```
.
```

```
.
```

```
iret
```

Istruzione per terminare
la gestione della trap

La syscall comunicherà il risultato tramite un registro!

do_syscall_32_irqs_on

```
regs->ax = ia32_sys_call_table[unr](regs);
```

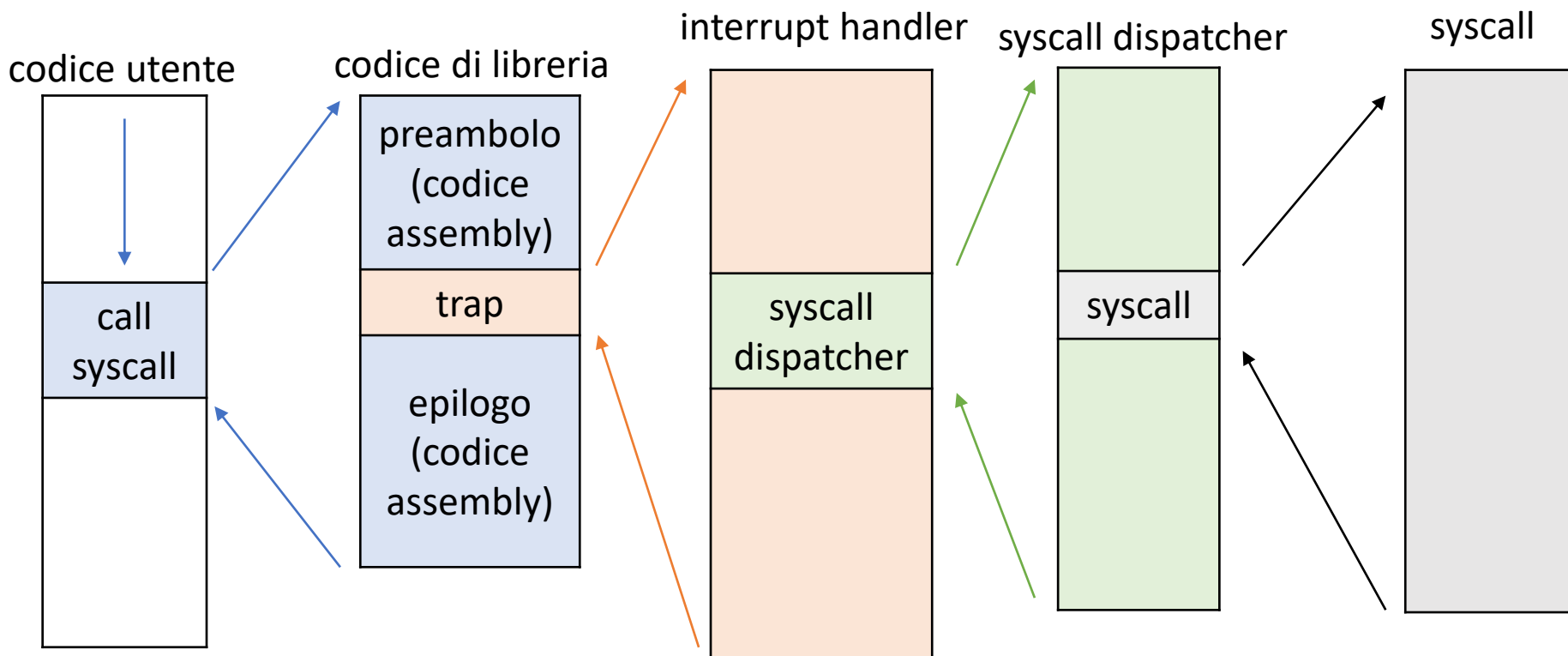
SYSCALL TABLE: tabella di
puntatori a funzione che
implementano l'effettiva system call

System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap

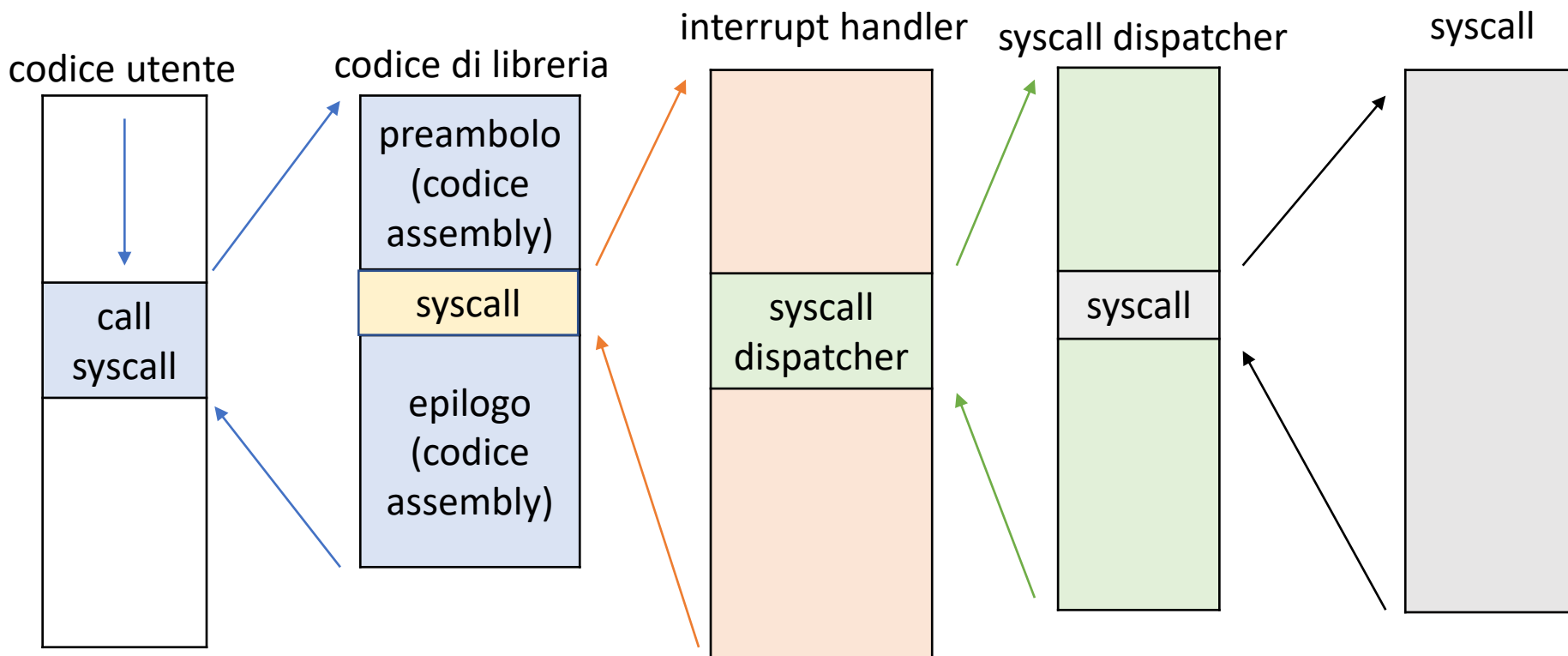
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap o **apposite istruzioni macchina**



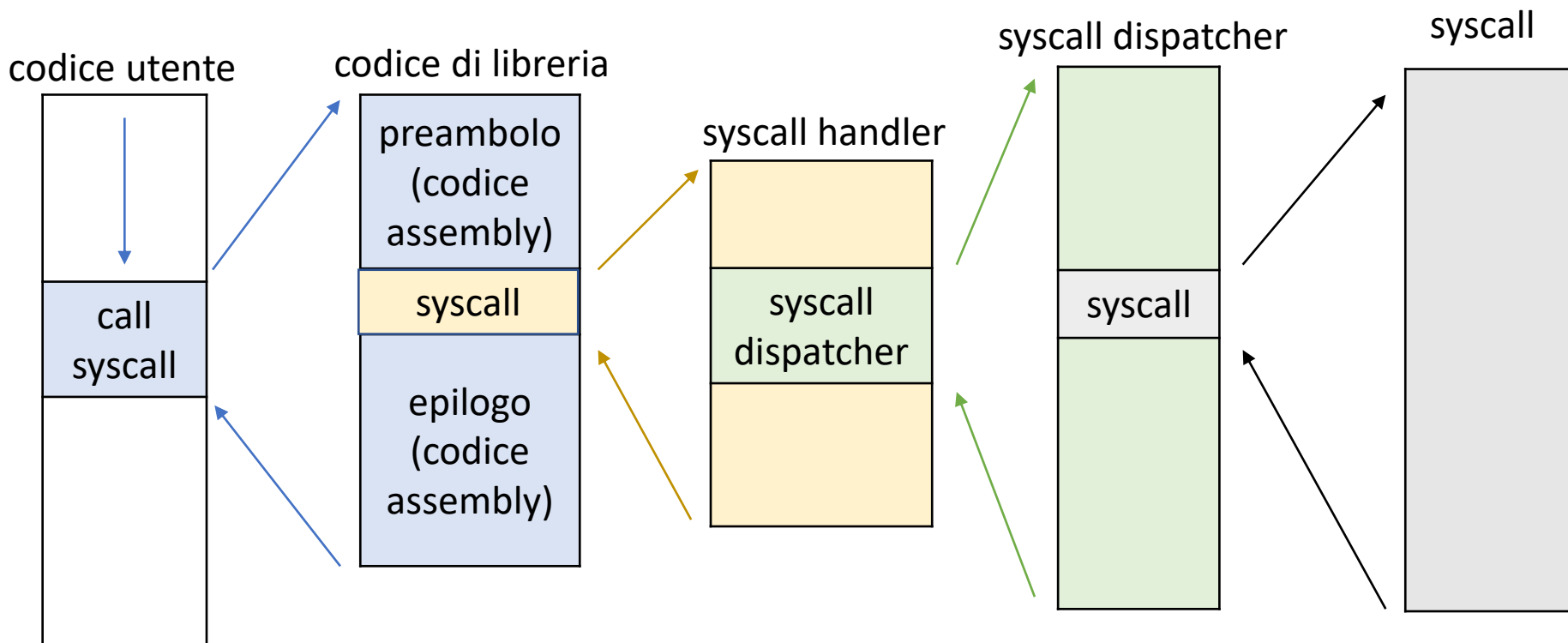
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap o **apposite istruzioni macchina**
che ci permette di invocare direttamente una syscall



System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap o **apposite istruzioni macchina**



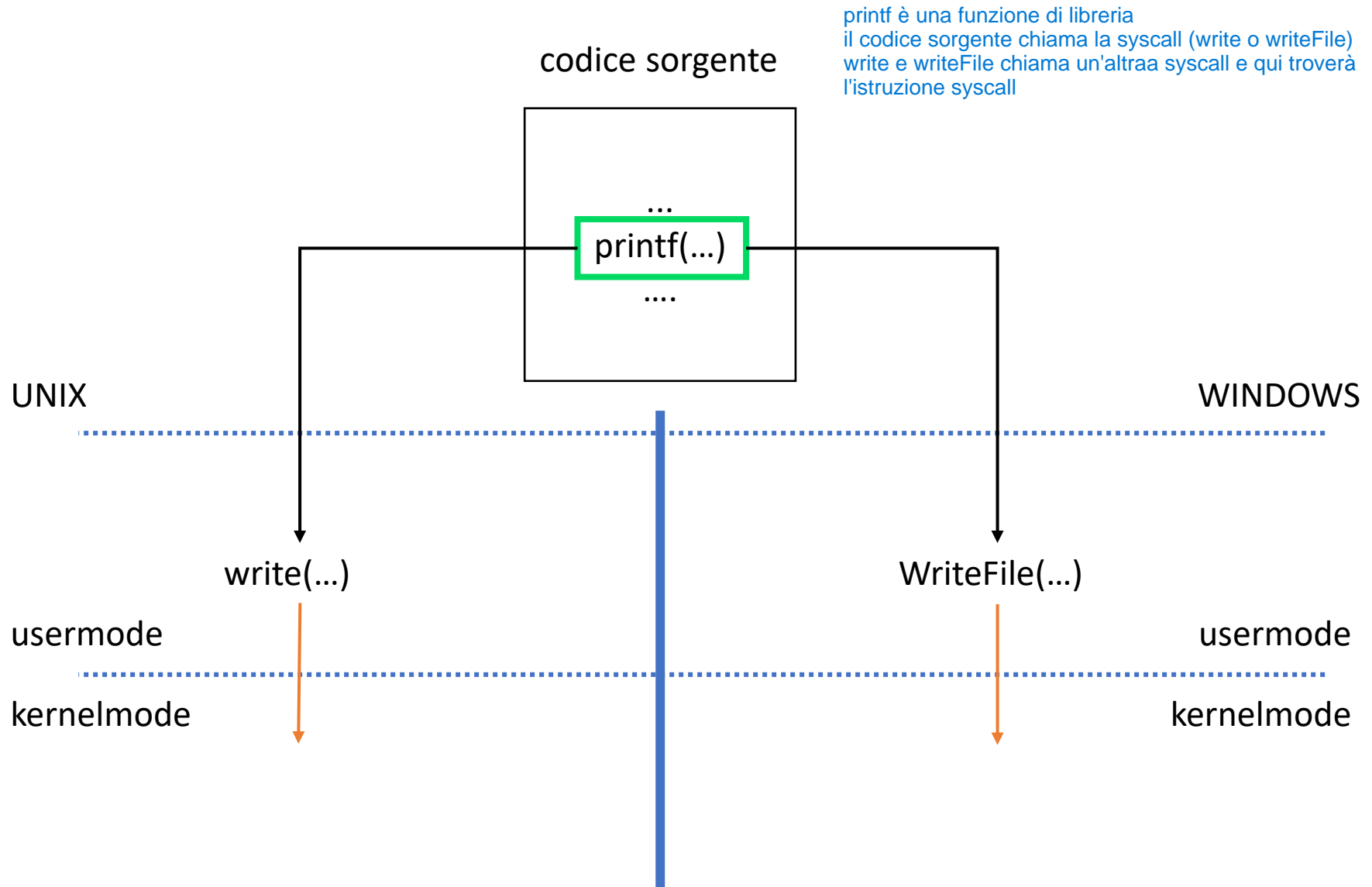
System call

- Il meccanismo delle chiamate di sistema (system call o syscall) è lo strumento messo a disposizione dal SO per accedere ed eseguire il suo codice
- Basato su trap
 - risulta necessario utilizzare codice che dipende dallo specifico hardware (ISA) per eseguire una data funzione del kernel
- E la convenienza?



- Offrono un ulteriore livello di astrazione al programmatore, il quale può invocare una system call come se fosse un'ordinaria funzione C (e.g., `write()`)
- Il programmatore dovrà solo preoccuparsi di compilare verso uno specifico ISA e sistema operativo

Funzioni di libreria standard – esempio



Funzioni di libreria standard

- Gli standard definiscono:
 - le system call offerte ai programmatori tramite libreria
 - le funzioni di libreria offerte ai programmatori (e.g., strcpy)
- Molteplici standard:
 - Posix:
 - ampiamente adottato da sistemi UNIX
 - parzialmente supportato in Windows
 - Win32 API:
 - sistemi operativi Windows
 - ReactOS [propone un OS open source](#)

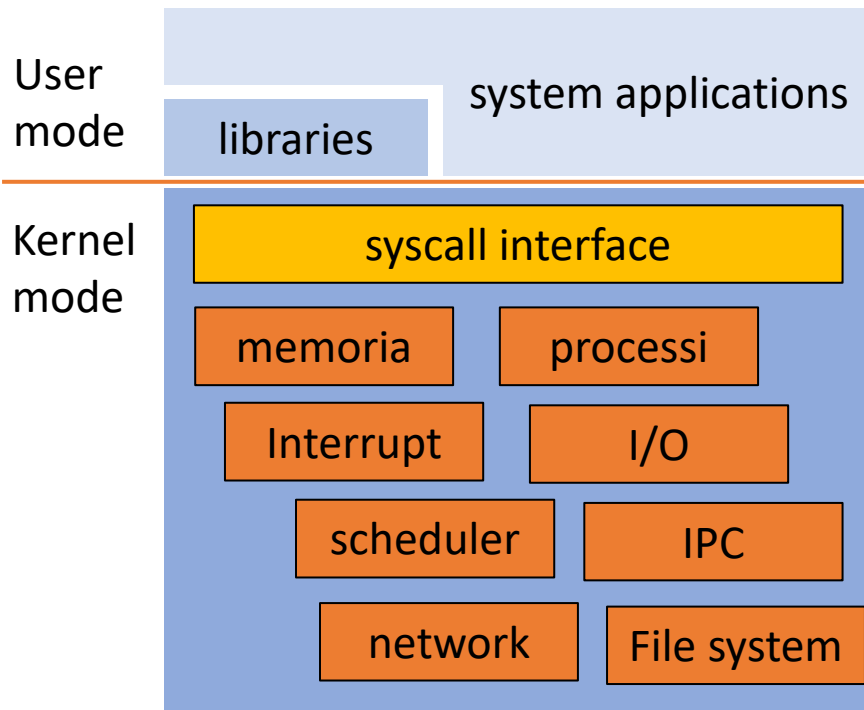
Architettura di un sistema operativo

libraries(codice)

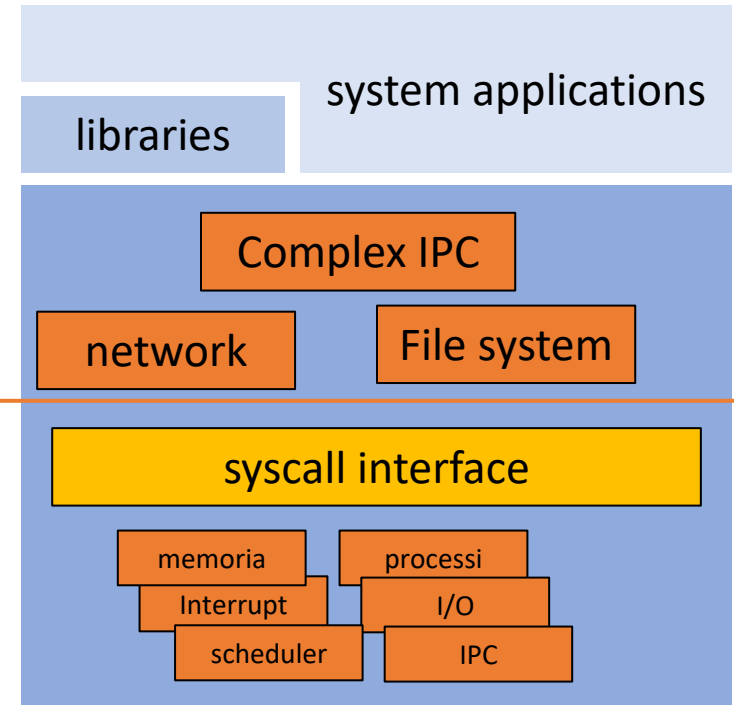
applicazione di sistema (più corretto applicazioni)

syscall interface: ci permette di accedere a tante funzionalità
come: la gestione di memoria, processi, interruzioni, I/O, scheduler,
IPC(inter process communication)

file system: si incarica di gestire i file



Kernel monolitico



MESSAGE PASSING: costo maggiore per colpa delle richieste,
perchè per poter passare un messaggio deve passare attraverso il
kernel

microkernel

kernel più ristretto

Architettura di un sistema operativo

Architettura a **microkernel**: insieme ristretto di funzionalità è nell'immagine del kernel

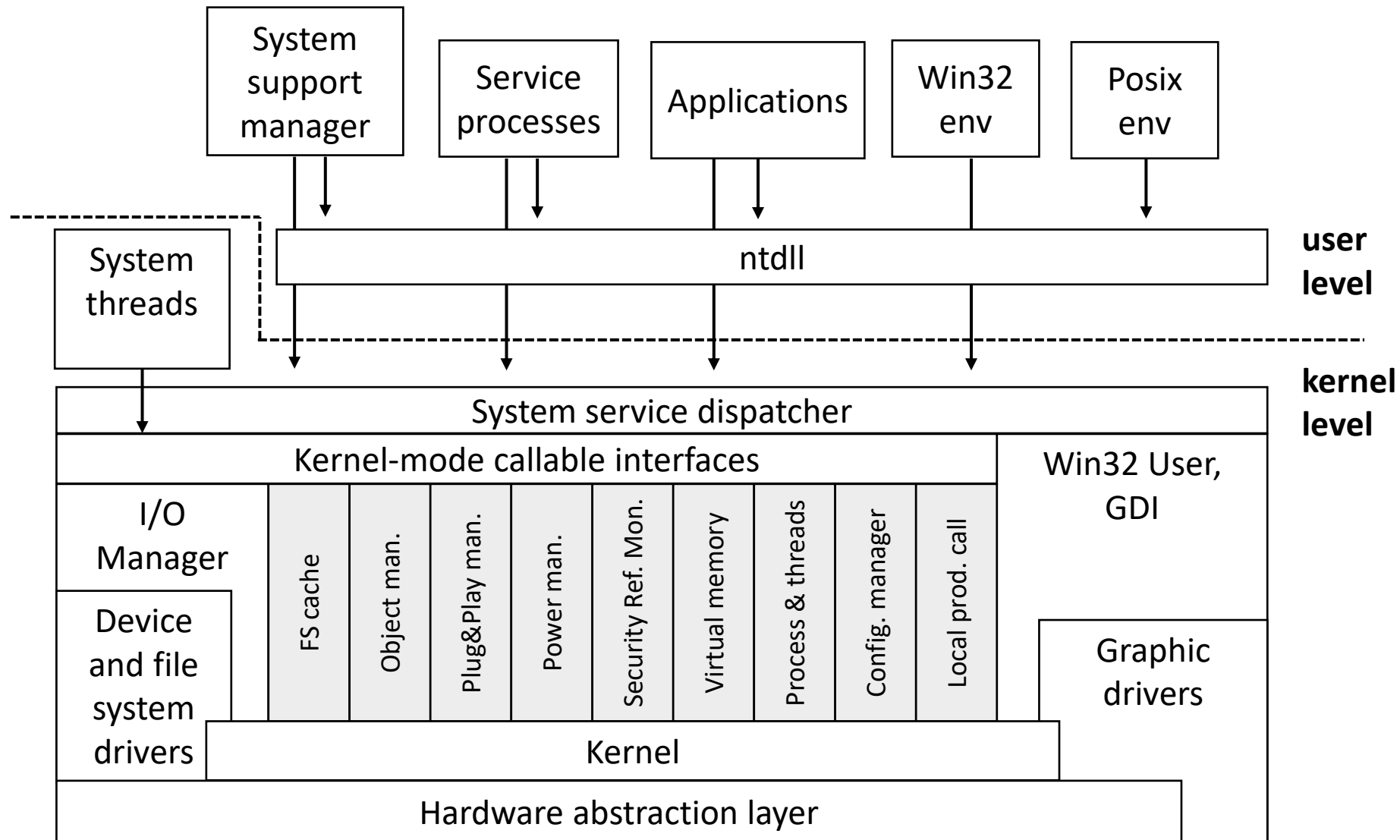
- Pros: Estendibilità, Modularità, Verifica formale del codice (o almeno di singole componenti)
- Cons: Minori performance legate alla frammentazione (non c'è condivisione di strutture dati tra le componenti nel kernel e quelle fuori dal kernel)

Architettura a **kernel monolitico**: tutte le funzionalità del sistema operativo sono incluse nel kernel

- Pros: Performance
- Cons: Codice potenzialmente più complesso da mantenere e/o estendere

Non c'è una netta distinzione

Un esempio reale – Windows



Un esempio reale – UNIX

