Operazione Rif. PA 2023-19410/RER approvata con DGR 1317/2023 del 31/07/2023 finanziata con risorse del Programma Fondo sociale europeo Plus 2021-2027 della Regione Emilia –Romagna.

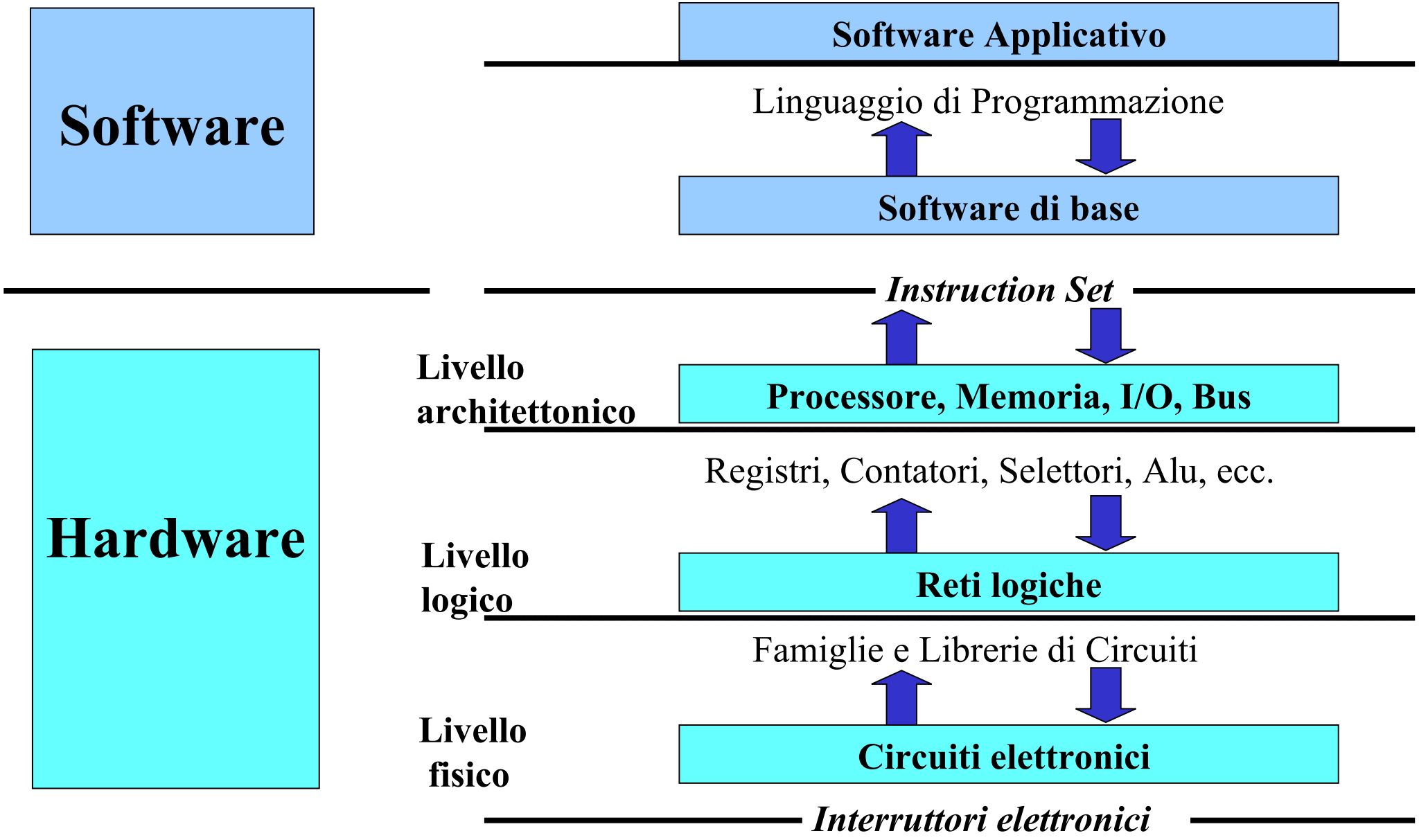
Progetto n. **1** - Edizione n. **1**

TECNICO PER LA PROGETTAZIONE E LO SVILUPPO DI APPLICAZIONI INFORMATICHE

MODULO: N. 5 Titolo: SICUREZZA DEI SISTEMI INFORMATICI E DISPIEGO DELLE APPLICAZIONI
DURATA: 21 ORE DOCENTE: MARCO PRANDINI

ARCHITETTURE DEI CALCOLATORI E DEI SISTEMI OPERATIVI

Strati di astrazione del progetto



La rappresentazione dell'informazione

Informazione - **Stringa** di lunghezza finita formata da simboli s_i appartenenti ad un **alfabeto** di definizione **A**:

$$s_1\, s_2\, s_3 \ldots s_i \ldots\ldots s_{n-1}\, s_n \quad \text{con } s_i \in A:\{a_1,\ a_2,\ ..,\ a_m\}$$

Esempi:

"testo" e caratteri

"numero" e cifre

"musica" e note

"immagine", pixel e toni di grigio

"disegno" e pend./lung. di tratti

"misura" e posizione di un indice

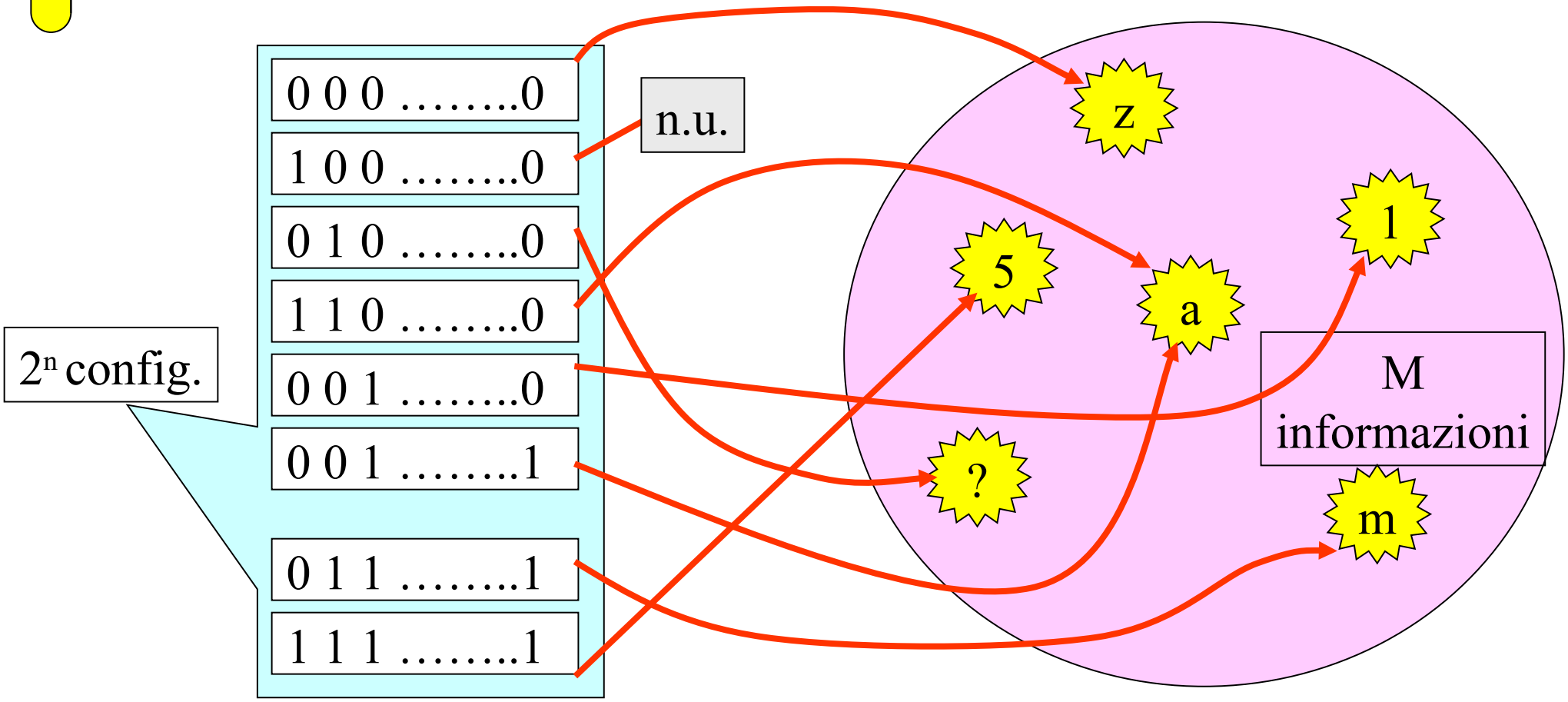
"parlato" e fonemi

Simboli di informazione - **Variabili a cui può essere assegnato come valore uno qualsiasi degli elementi dell'alfabeto A**

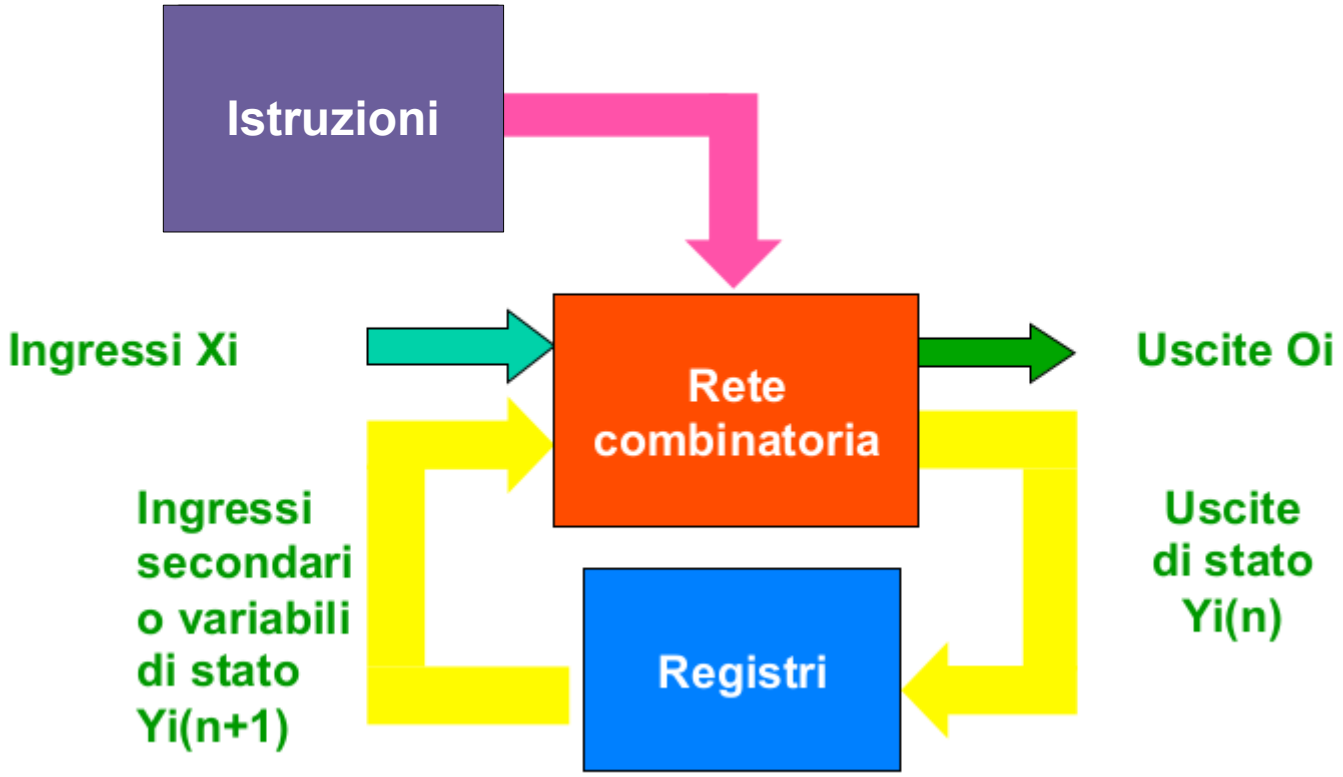
La codifica binaria della informazione

Codice binario

<u>Codice binario</u> - Funzione dall'insieme delle 2^n configurazioni di n bit ad un insieme di M informazioni (*simboli alfanumerici*, *colori*, *eventi*, *stati interni*, ecc.).
Condizione necessaria per la codifica: $2^n \geq M$



Macchine programmabili a stati



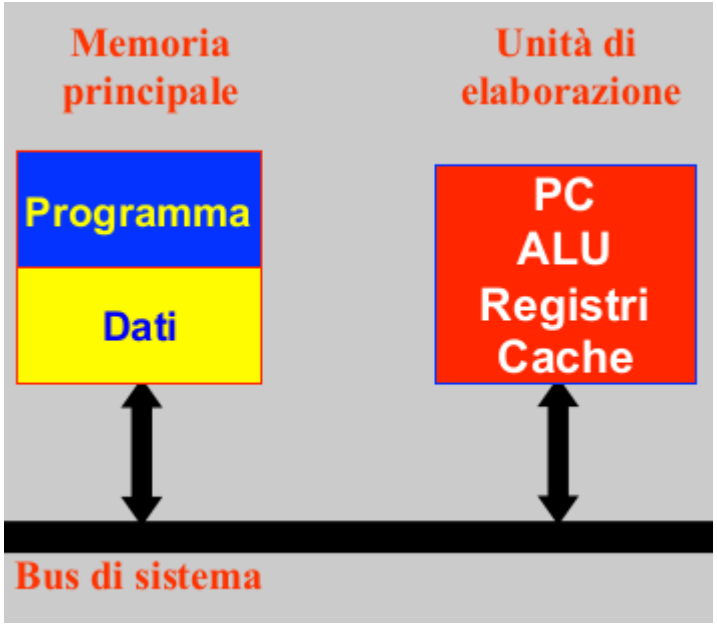
Dati e istruzioni sono rappresentati in modo omogeneo da sequenze di bit
Possiamo costruire una rete elettronica alimentata istante per istante con

- dati da elaborare
- istruzioni su che operazione eseguire
- il contenuto di una memoria (registro) che rappresenta lo *stato* del sistema, cioè una sintesi di come si è evoluto nel passato fino all'istante corrente

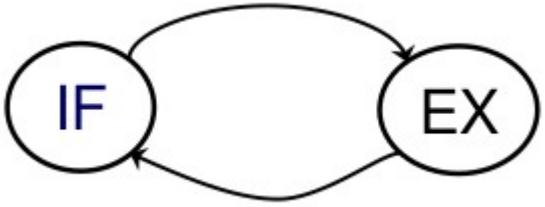
che produca un risultato immediato e memorizzi nei registri il nuovo stato (es: somma in colonna)

Architettura astratta di un calcolatore secondo il modello di Von Neumann

- Potremmo inserire a mano i dati e le istruzioni, ma...
- se invece la rete sequenziale, tra i tanti risultati che può produrre, fosse capace di comandare un dispositivo che trova in una *memoria* la prossima istruzione da eseguire e i dati su cui lavorare?

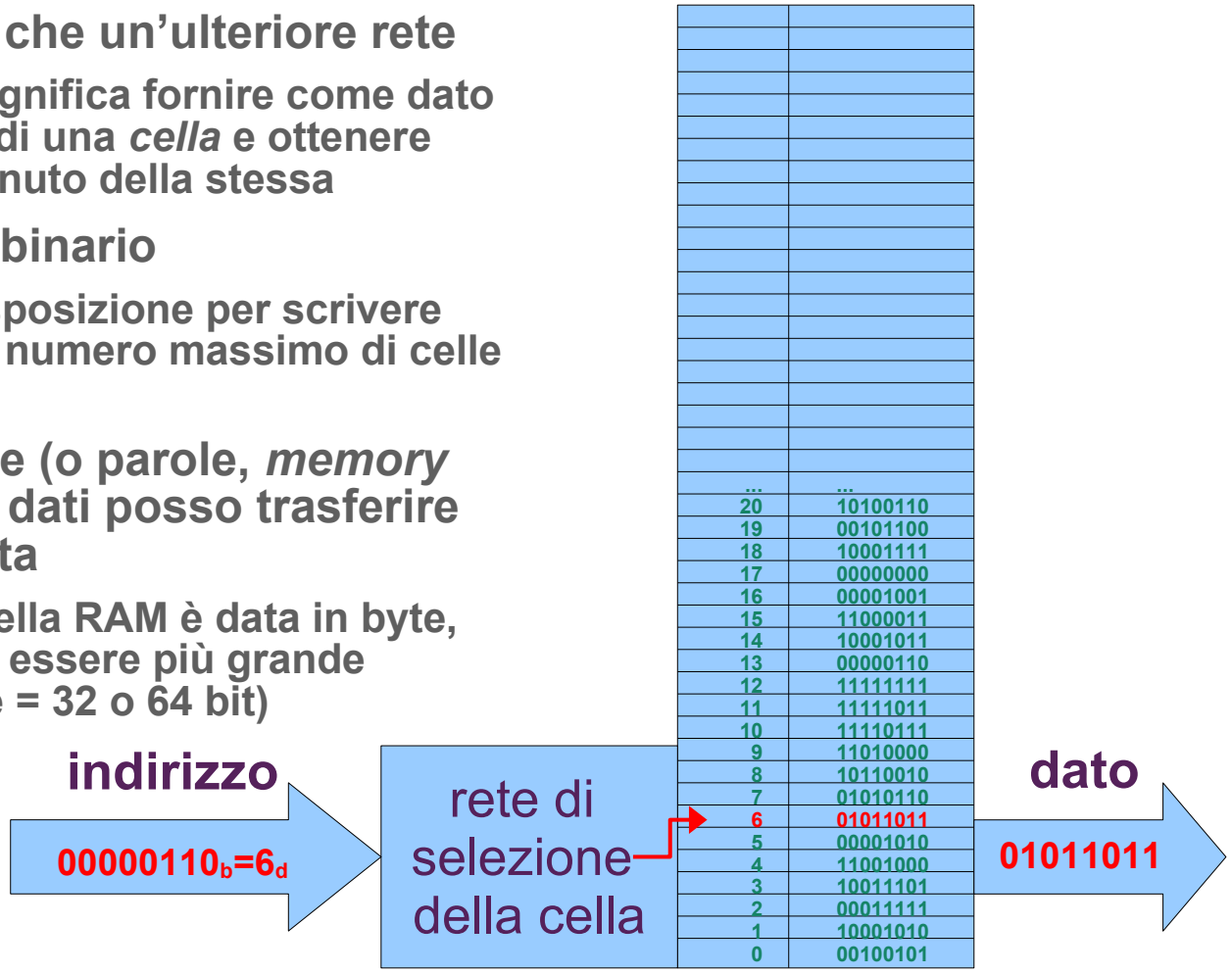


- Supponiamo che la rete, appena accesa, sappia "dove guardare"
- La CPU preleva alcuni bit dalla memoria (Instruction Fetch - IF)
- La sua rete interna produce risultati in funzione di quei bit (Execution- EX)
- Tra i risultati, c'è la richiesta alla memoria di fornire la prossima istruzione – siamo tornati a IF



Memoria

- Cosa significa “prelevare dalla memoria”?
- La memoria non è altro che un’ulteriore rete
  - leggere il contenuto significa fornire come dato in ingresso l’*indirizzo* di una *cella* e ottenere come risultato il contenuto della stessa
- l’indirizzo è un numero binario
  - il numero *N* di bit a disposizione per scrivere l’indirizzo determina il numero massimo di celle identificabili: 2^N
- la dimensione delle celle (o parole, *memory word*) determina quanti dati posso trasferire con una singola richiesta
  - la dimensione totale della RAM è data in byte, ma singola parola può essere più grande (tipicamente 4 o 8 byte = 32 o 64 bit)



Il Program Counter

- Quindi: per poter eseguire le istruzioni in sequenza la CPU dispone al suo interno di un contatore detto Program Counter (PC)
- Il PC viene incrementato ad ogni FETCH
- Il PC contiene l'indirizzo in memoria della prossima istruzione da leggere nella prossima fase di FETCH; in questo modo il PC individua la prossima istruzione da eseguire

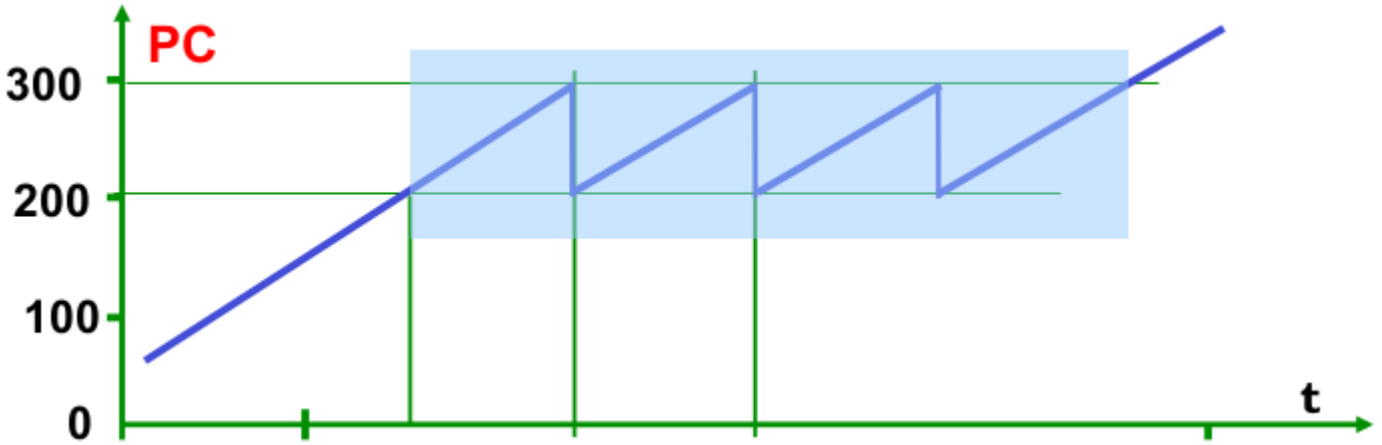


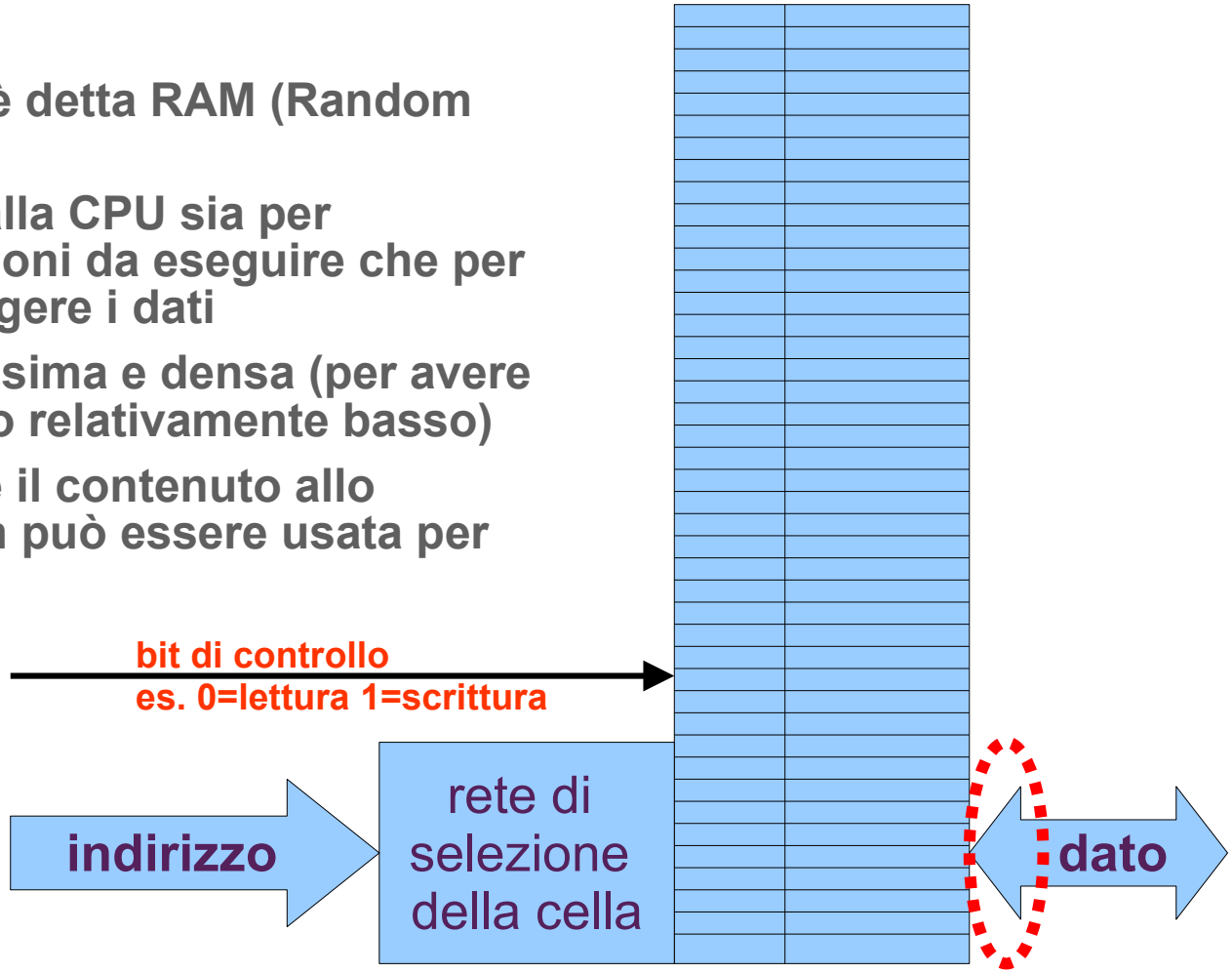
Grafico che mostra qualitativamente la dinamica del PC quando il calcolatore entra in un loop, esegue per 4 iterazioni le istruzioni dalla cella 200 alla 300 della memoria, e poi prosegue

Tipi di memoria

- Questo modello di esecuzione pone alcune domande fondamentali:
  - come fa la CPU a sapere da dove iniziare a prendere le istruzioni?
  - chi mette le prime istruzioni da eseguire nella memoria?
  - la CPU può usare la memoria anche per immagazzinare i risultati che produce, oltre che per prelevare istruzioni e dati?
- Per dare risposta in modo efficiente a tutte queste domande, nei calcolatori esistono diversi tipi di memoria

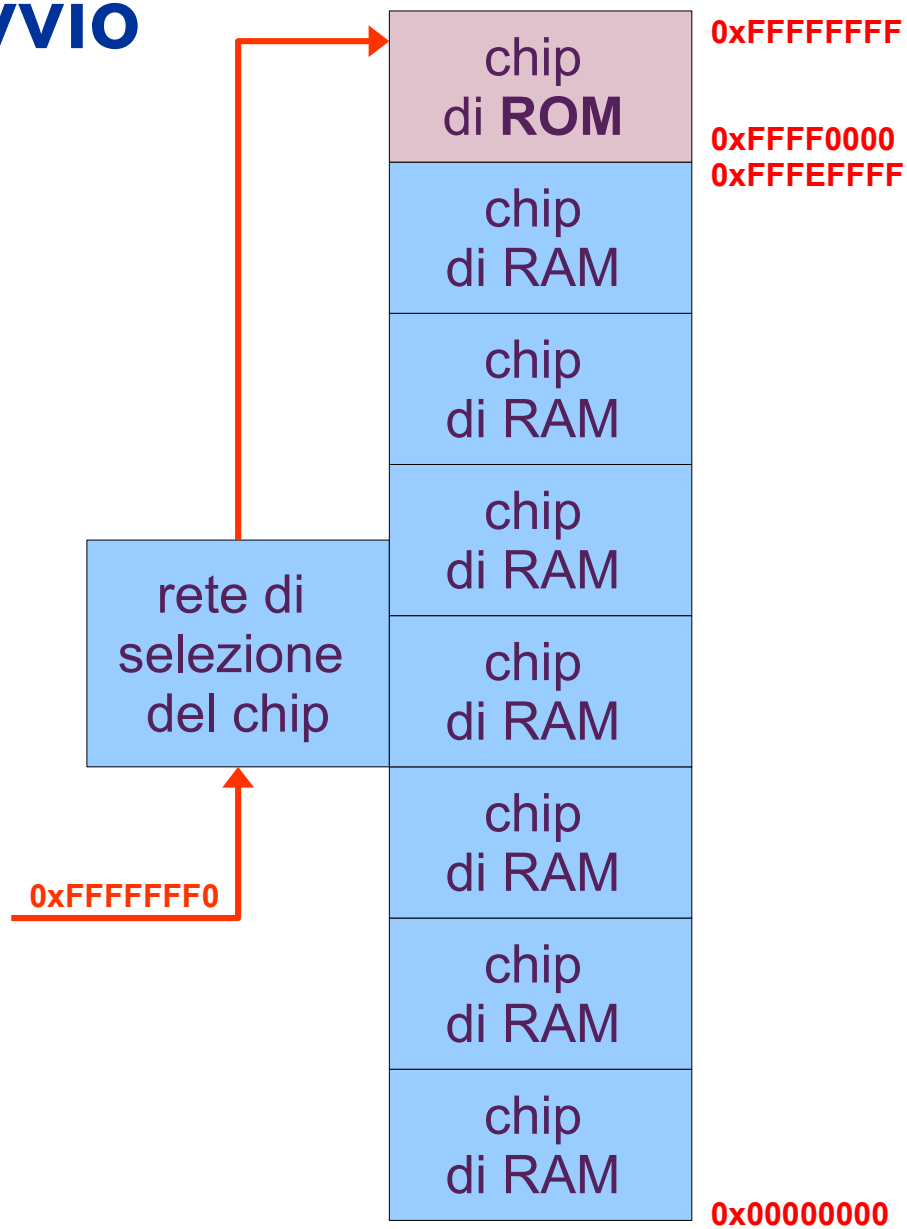
RAM

- La memoria principale è detta RAM (Random Access Memory)
  - È la più utilizzata dalla CPU sia per recuperare le istruzioni da eseguire che per memorizzare e rileggere i dati
  - Deve essere velocissima e densa (per avere alta capacità e costo relativamente basso)
  - È *VOLATILE* : perde il contenuto allo spegnimento → non può essere usata per avviare il computer!



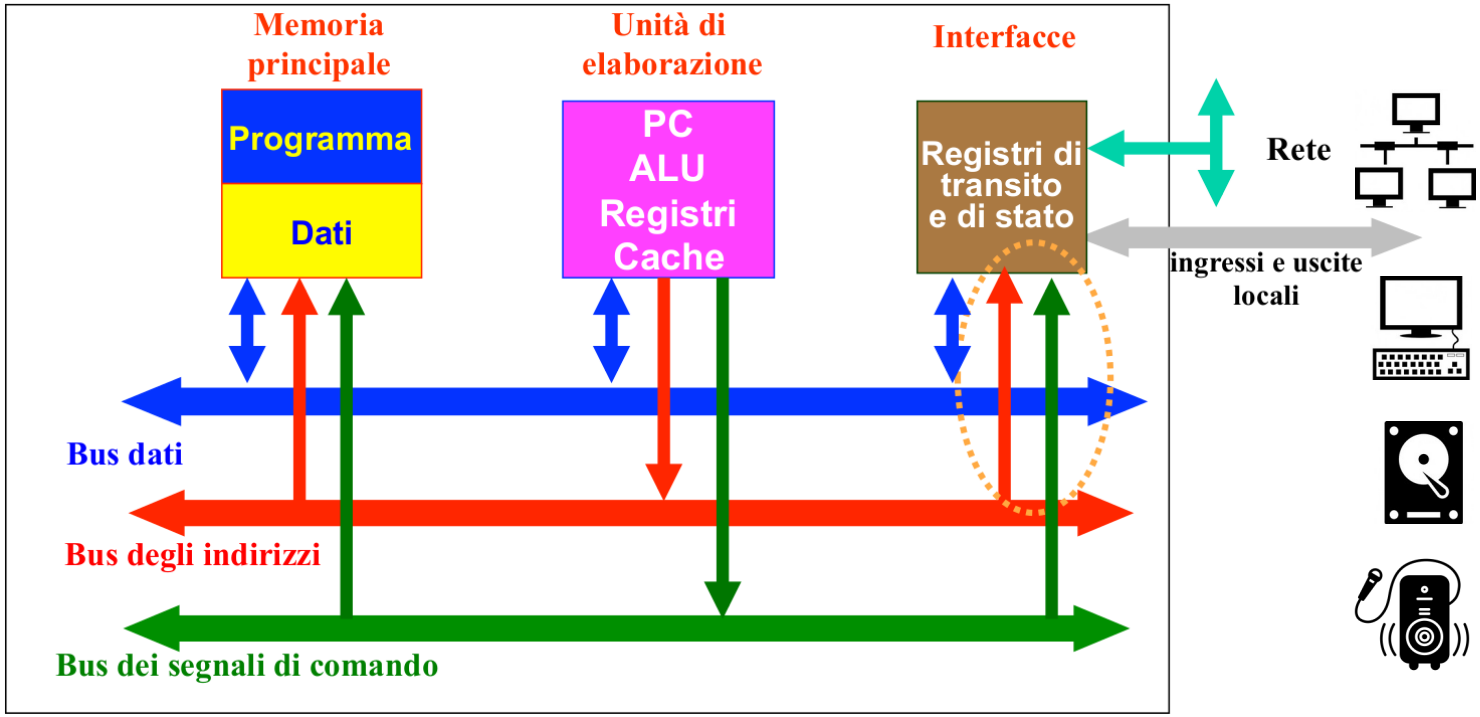
ROM e avvio

- Per memorizzare in modo permanente le istruzioni basilari che la CPU deve eseguire appena accesa, si usa una ROM (Read-Only Memory)
  - ne esistono diverse varianti: PROM, EPROM, EEPROM, Flash, ...
- Sfruttando il principio dello spazio di indirizzamento composto, si può illudere la CPU di avere un'unica memoria, che però
  - in parte è ROM col programma di avvio
    - **l'indirizzo della prima istruzione da caricare è "scritto a fuoco" nella CPU e deve cadere nello spazio in cui è collocata la ROM**
  - in parte è RAM per il normale funzionamento



Uscire dall'isolamento

- Leggere e scrivere dalla memoria e fare calcoli è utile, ma come forniamo dall'esterno i dati e come portiamo all'esterno i risultati?
- Servono circuiti elettronici specifici: le **interfacce di ingresso/uscita (I/O)**
  - La CPU può raggiungerle con gli stessi canali che già usa per la memoria
  - La CPU sa solo leggere e scrivere: *cosa mandare per far succedere qualcosa* deve essere deciso da un programma apposito! (torneremo sull'argomento in seguito)



Implementazioni dell’architettura

La main board del “classico” computer

alloggia i tre componenti e nasconde i bus

Connettori per periferiche*

Socket per la **CPU**

Slot (PCI) per schede aggiuntive*

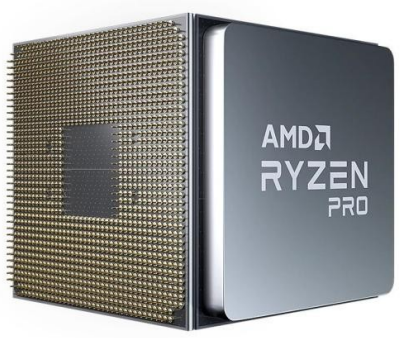
Slot per la **RAM**

Connettori per hard disk SATA*

* pilotati da specifiche **interfacce** integrate nel *chipset*

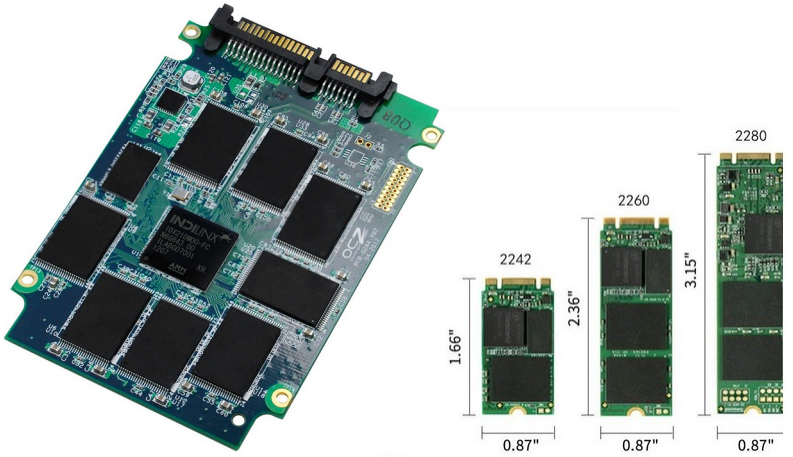
Elementi del computer - CPU

- Le CPU moderne sono miracoli di integrazione, contano miliardi di transistor (dai 3-10 delle CPU Intel Core e AMD Ryzen, agli oltre 100 della CPU Apple M2)
- Possono elaborare e trasferire parole di 64 bit, e utilizzano 48 bit per indirizzare le celle di memoria – se si considerano i segnali di controllo e la necessità di distribuire l'alimentazione elettrica in più punti, necessitano quindi di centinaia di connettori fisici (pin)



Elementi del computer – Hard disk

- Sono essenziali per memorizzare i dati e i programmi anche senza alimentazione elettrica
- Gli hard disk “tradizionali” sono composti di uno o più piatti con superficie magnetica, su cui una testina simile al braccio di un giradischi memorizza o legge bit
  - Consumano, scaldano, sono soggetti a (rari) guasti meccanici
- I Solid State Drive (SSD) sono invece totalmente elettronici
  - Più costosi
  - La riscrittura delle celle di memoria può deteriorarle (centinaia di migliaia di cicli, ma è facile raggiungerli)



Elementi del computer - memorie a confronto (ordini di grandezza – dati 2023)

	RAM (DDR5)	RAM (DDR4)	SSD (nVME)	SSD (SATA)	HDD
Capacità tipica	8-128 GB **(1)**	4-64 GB **(1)**	0,5-8 TB	0,5-8 TB	1-22 TB
Latenza **(2)**	15 ns	13 ns	20 µs	100-200 µs	10 ms
Throughput **(3)**	32-64 GB/s	12-24 GB/s	4-16 GB/s	500 MB/s	100-200 MB/s
Costo per GB	4-6 €	3€	0,05-0,10 €	0,05 €	0,02-0,03 €

Note generali

- Sono confrontati RAM e Hard Disk – ma ricordate che la CPU interagisce direttamente solo con la RAM (o ROM) mentre per interagire con un HD deve prima essere caricato in RAM un programma!
- La tecnologia procede rapidamente e ci sono estremi che non sono stati riportati, es. il disco SSD ExaDrive da 100TB (40.000$). I valori indicati sono quelli più comuni sui dispositivi facilmente reperibili.

Note alla tabella

(1) capacità di 1 modulo - sulle mainboard più comuni si possono installare fino a 4 moduli, ma i sistemi server di fascia alta possono ospitarne decine. Lo standard DDR5 può raggiungere 512GB su singolo modulo, annunciato al mercato da Samsung (2021) ma di fatto introvabile.
(2) il tempo che trascorre tra il comando di lettura e l'arrivo dei dati
(3) la velocità a cui fluiscono i dati letti sequenzialmente, una volta trascorsa la latenza
Per la scrittura valgono le stesse considerazioni (tipicamente con prestazioni inferiori)

Esigenze diverse, computer diversi

Nel mondo del calcolo in generale, e nella Internet of Things, ci sono esigenze talmente differenziate che sarebbe insensato affrontarle con un'unica implementazione. In estrema sintesi:

- "Things": bassissime esigenze di prestazioni, scarsa necessità di riprogrammarli una volta messi in opera, interfacciamento diretto col mondo esterno, bassissimo costo → ***microcontrollori*** *(es. Arduino)*
- "Things evolute": prestazioni contenute, possibilità di interfacciamento tipiche di un PC + interfacciamento diretto col mondo esterno, bassi consumi e ingombri → ***single board computer*** *(es. Raspberry Pi)*
- Workstation: PC standard, in cui le prestazioni di alcuni sottosistemi possono essere anche molto elevate (es. grafica per CAD e gaming), ma con scarse necessità di multitasking → **PC con pochi core, pochi GB di RAM, dischi standard**
- Server: sistemi pensati per sfruttare l'economia di scala di moltissimi task contemporanei → **molti core, TB di RAM, sistemi di storage parallelo, ridondanza di tutti i componenti per tolleranza ai guasti**

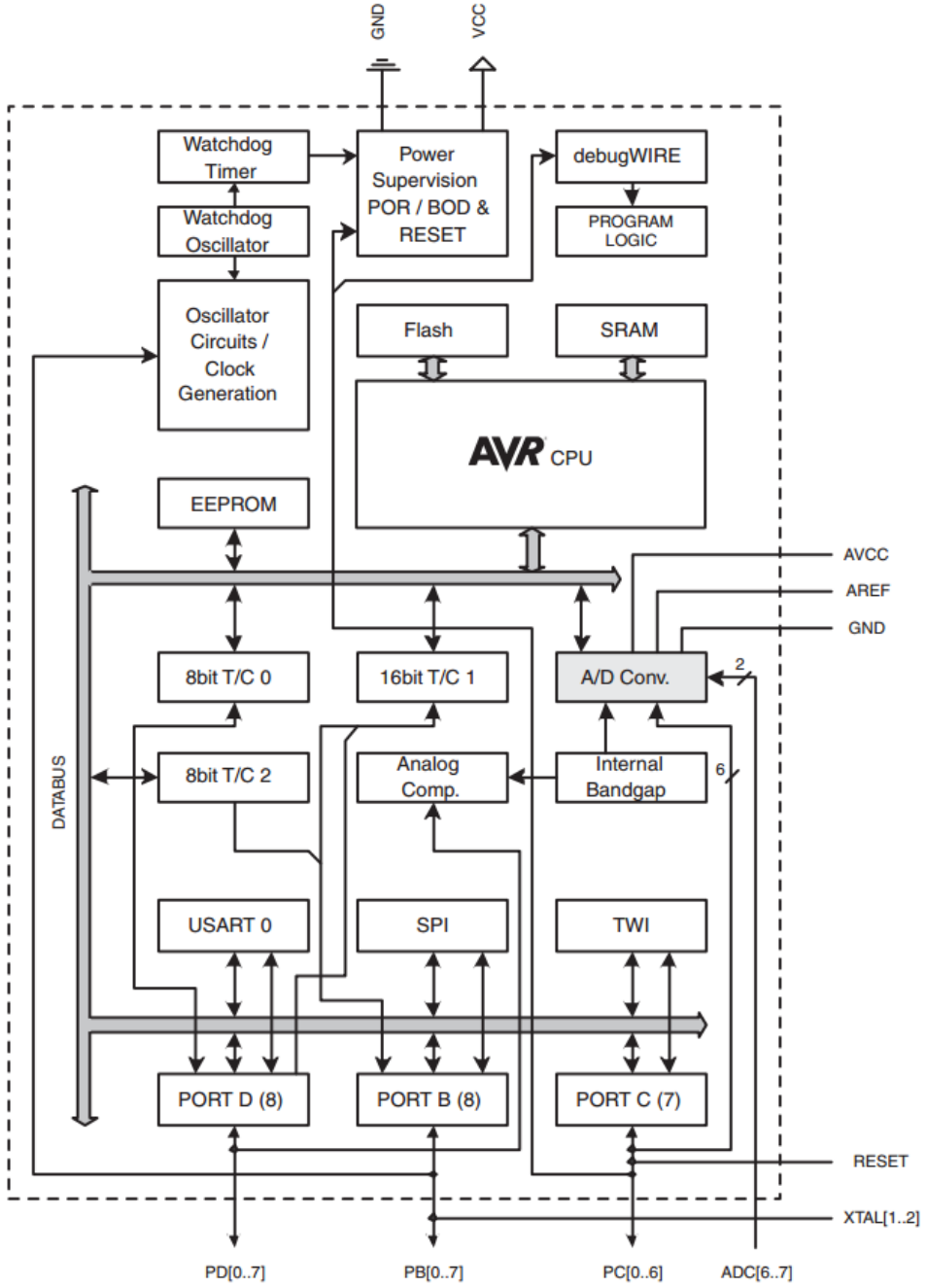
Microcontrollori

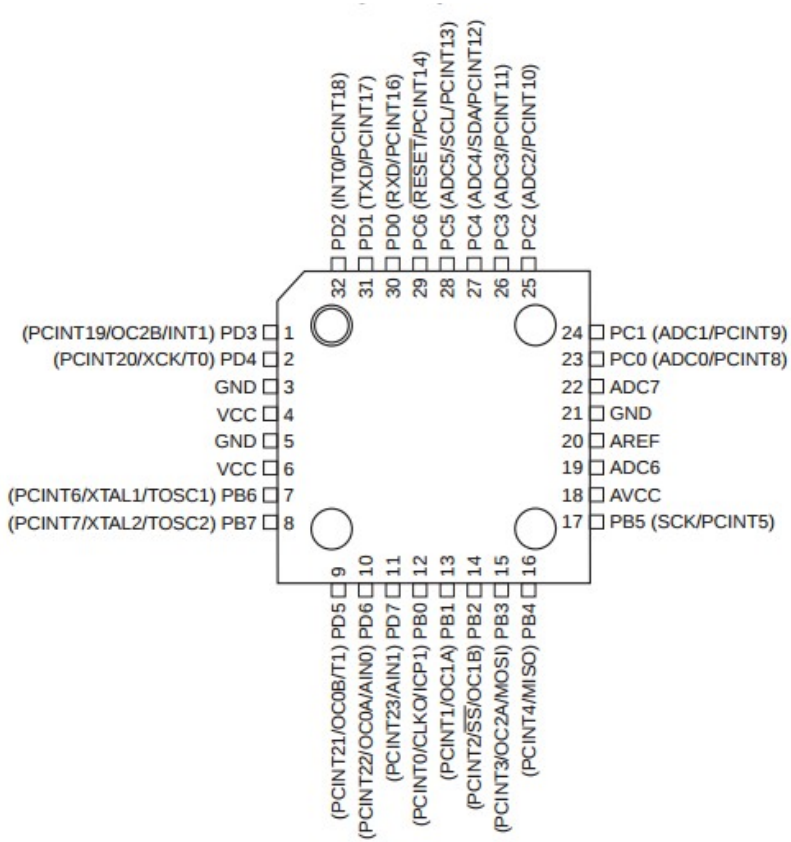
- I microcontrollori condensano in un unico package quasi tutti gli elementi di un sistema CPU-mainboard-memoria-I/O
  - Tipicamente con dimensioni, prestazioni e costi molto più limitati
- Per esempio il **uC base per Arduino** a confronto con una **CPU Intel comune**

ATmega328P	ATmega328P	Intel Core i7 1065G7
CPU	1 core RISC 8 bit	4 core CISC 64 bit
Clock	20 MHz	3900 MHz
Storage	32 kB Flash	--
ROM	1 kB EEPROM	--
RAM	2 kB	-- *tecnicamente 8 MB, ma solo per cache*
Ingressi analogici	8	--
I/O digitali	23 pin *programmabili e utilizzabili per leggere interruttori e azionare dispositivi*	64 bit *data bus, no accesso esterno*
Prezzo	2 €	500€

ATmega328P

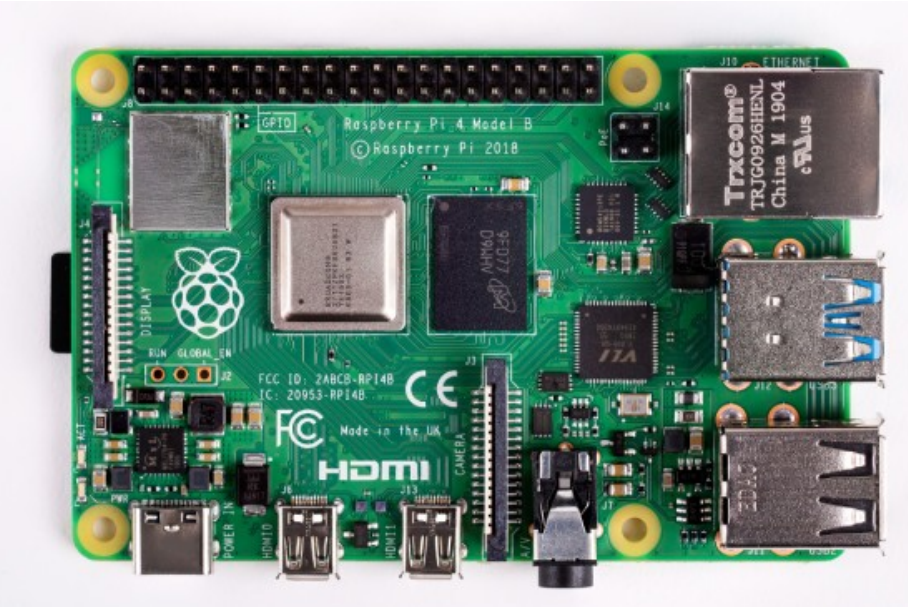
http://www.atmel.com/Images/doc8161.pdf





Single-board computer

- Basati sullo stesso principio dei microcontrollori, integrano però componenti tipici dei PC. Ad esempio il Raspberry Pi 4 model B è basato sull'integrato Broadcom BCM2711, un *System on a Chip (SoC)* con
  - 4 core ARM 64 bit a 1800MHz
  - 1, 2, 4 o 8 GB di RAM
  - 1 GPU BC VideoCore VI
    - 2 uscite micro-HDMI 4k
  - Uscita audio stereo
  - Connettore MIPI DSI per display
  - Connettore MIPI CSI per camera
  - 4 porte USB (2x 2.0, 2x 3.0)
  - Interfacce di rete
    - Gigabit Ethernet
    - WiFi 802.11ac
    - Bluetooth 5.0
  - 40 pin GPIO
  - Manca memoria non volatile, sostituita da un'interfaccia per schede SD



Single-board computer

- Sono oggetti in grado di sostituire al 100% un PC per uso domestico o ufficio
  - Su Raspberry Pi sono state portate diverse distribuzioni complete di Linux e Windows Embedded

  al costo di poche decine di euro, con un consumo di pochi watt
- I sistemi a microcontrollore, enormemente meno potenti non necessariamente molto più economici, mantengono come vantaggi
  - L'interfacciamento diretto verso il mondo analogico
  - Il consumo 10-20 volte inferiore
- Raspberry ha tracciato la rotta

  https://www.raspberrypi.org/
- Molti altri hanno accettato la sfida

  http://www.computerworlduk.com/galleries/it-vendors/move-over-raspberry-pi-9-single-board-computers-for-geeks-3544497/

Dispositivi mobili

- I dispositivi mobili (smartphone, tablet) sono dei single-board computer
  - es. la CPU del Raspberry è esattamente la stessa montata sullo smart watch Samsung Galaxy Gear e su di una decina di telefoni
  - tipicamente gli smartphone sono più potenti di un SBC!

  resi autonomi da
  - un'interfaccia utente (touchscreen, videocamera, speaker)
  - una batteria
  - un set di interfacce wireless (GSM / LTE / WiFi …)

  e semplificati eliminando GPIO e altre possibilità di espansione diverse dalla porta USB (in qualche variante proprietaria)
- Il sistema operativo è derivato da una versione standard per computer (MacOS → iOS, Linux → Android, Windows → Windows Phone) ottimizzato sull'hardware specifico e quindi legato al produttore

Workstation

- I PC “desktop” racchiudono in forma abbastanza compatta tutti i componenti necessari, con utili (ma non ampie) possibilità di configurazione ed espansione

Agganci per mainboard, interfacce integrate in posizione utile

Schede (video, altri controller)

alimentatore

“Baie” per unità ottiche

Slot per fissaggio hard disk interni

Server (“classici” x86)

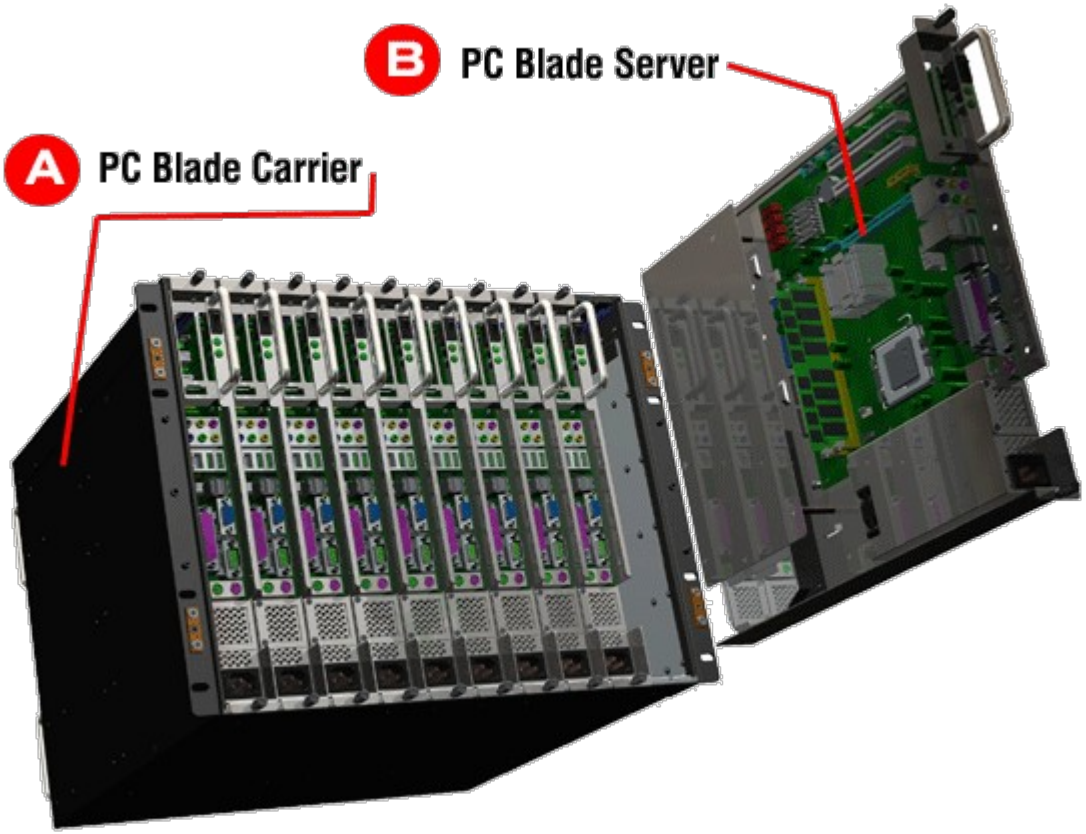
- Architetturalmente non sono molto diversi da una workstation, ma sono orientati a
  - Prestazioni
    - Mainboard multi-CPU (2/4, eccezionalmente 8)
  - Espandibilità
    - Più slot per RAM
    - Più slot per HDD
    - Montaggio su rack
  - Affidabilità
    - Doppia alimentazione
    - RAM ECC
    - Multiple interfacce di rete

Intel® Server Board S2600WT2 or
Intel® Server Board S2600WTT

- Supports **two Intel® Xeon® processor E5-2600 v3 family**
- **24 memory sockets** support LR/U/R/NV-DIMMs, up to 1.5 TB total memory (using 64 GB DIMMS)
- 80 PCIe* Gen 3 and four PCIe Gen 2 lanes available for maximum I/O capacity
- Options to support up to **24 2.5-inch or 12 3.5-inch hot-swap hard drives**
- 750 W AC **redundant-capable power supply** (80 PLUS* Platinum efficiency), 750 W DC redundant-capable power supply (80 PLUS Gold efficiency), or 1100 W AC redundant-capable power supply (80 PLUS Platinum efficiency), second power supply sold separately
- Two full-height half-length PCIe* Gen 3 x16 slots
- **Dual 1 GbE LAN or dual 10 GbE LAN** options

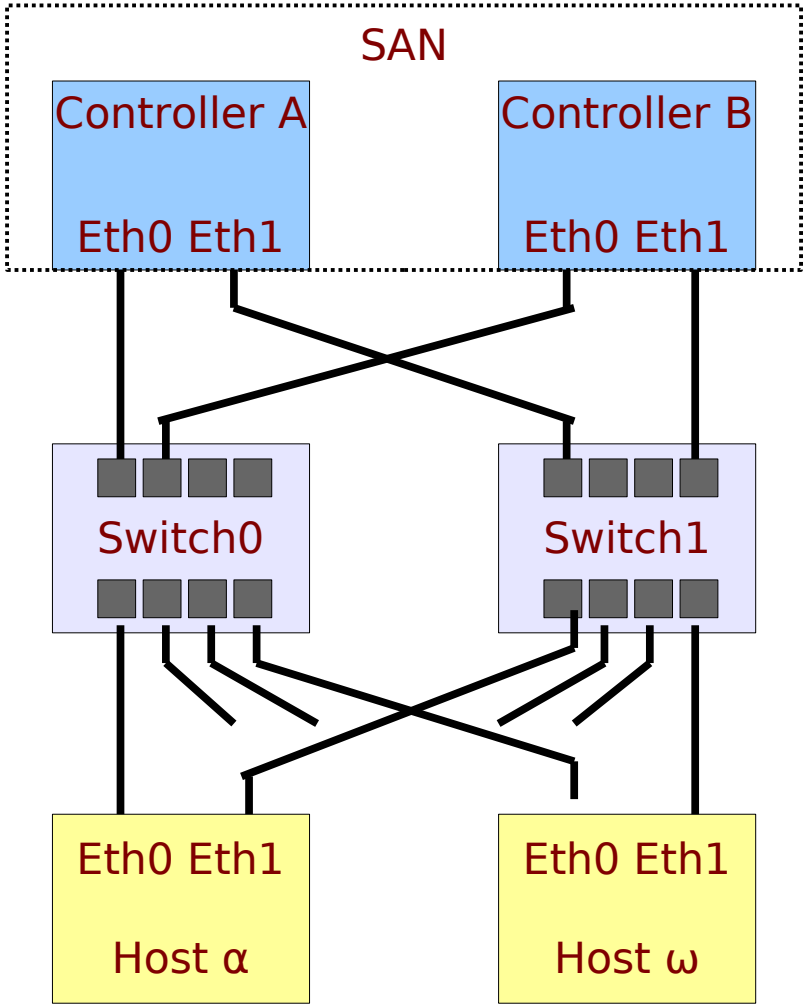
Server blade

- Mettono a fattor comune alimentazione e connessione KVM (Keyboard, Video, Mouse) tra diverse “lame”, per incrementare la densità



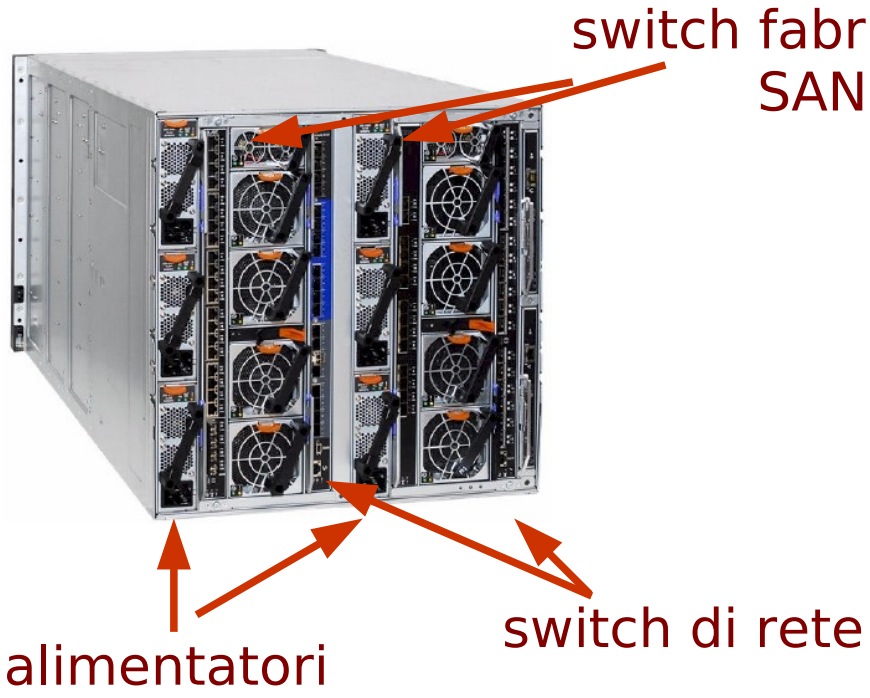
Cluster

- Per erogare servizi in modo affidabile e con prestazioni molto superiori a quelle ottenibili da un singolo sistema, l'affidabilità, molti computer vengono affiancati per svolgere collettivamente il compito richiesto
- Tipicamente il cuore del cluster è un sistema di storage condiviso + un sistema per la realizzazione di connessioni ridondanti (doppie schede su ogni server, doppi switch, ecc….)
- La parte più complessa è gestire l'accesso concorrente alle risorse e il monitoraggio dei nodi, perché in caso di guasto di uno i servizi vengano riavviati su quelli ancora attivi



Datacenter-in-a-box

- Vista la preponderanza dei cluster, i sistemi blade si sono evoluti per ospitare tutti i componenti necessari



Ruolo e struttura del sistema operativo

slide in parte tratte dal materiale di Anna Ciampolini e Paolo Bellavista

Che cos'è un Sistema Operativo (SO)?

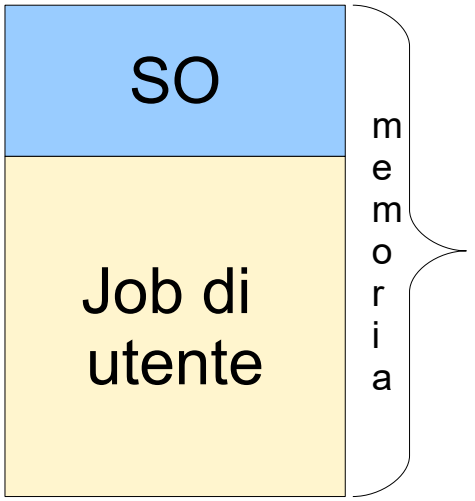
- È un programma (o un insieme di programmi) che agisce come intermediario tra l'utente e l'hardware del computer:
  - fornisce un ambiente di esecuzione per i programmi applicativi
  - fornisce una visione astratta dell'hardware
  - gestisce efficientemente le risorse del sistema di calcolo

- Per comprenderne il funzionamento, ricorriamo ad alcune semplificazioni e ricordiamo le caratteristiche basilari dell'hardware
  - la CPU è in grado di eseguire un'istruzione per volta
    - trascuriamo il caso di multi-CPU o multi-core: i principi sono gli stessi
  - la memoria è una sequenza uniforme di celle numerate
  - i dispositivi di I/O sono pilotati da programmi che con le loro istruzioni (sempre una dopo l'altra!) scrivono sulle interfacce comandi (numeri) che provocano il comportamento desiderato del circuito elettronico

Evoluzione dei SO

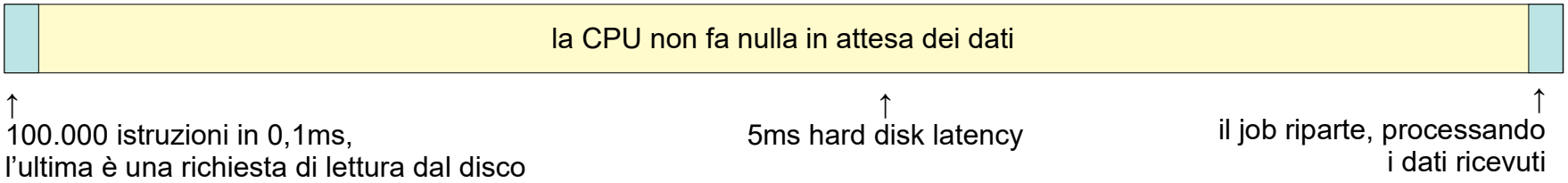
- Prima generazione (primi anni ‘50)
  - computer troppo poco potenti per potersi “permettere” un SO
  - programmi scritti direttamente nel linguaggio della CPU
  - dati e programmi memorizzati su schede perforate

- Seconda generazione (1955-1965)
  - sistemi batch semplici
  - comparsa di linguaggi di alto livello (fortran)
  - aggregazione di programmi in lotti (batch) con esigenze simili
  - il sistema operativo resta caricato in memoria e ha l’unico ruolo di trasferire dalle schede alla memoria un programma quando il precedente è terminato



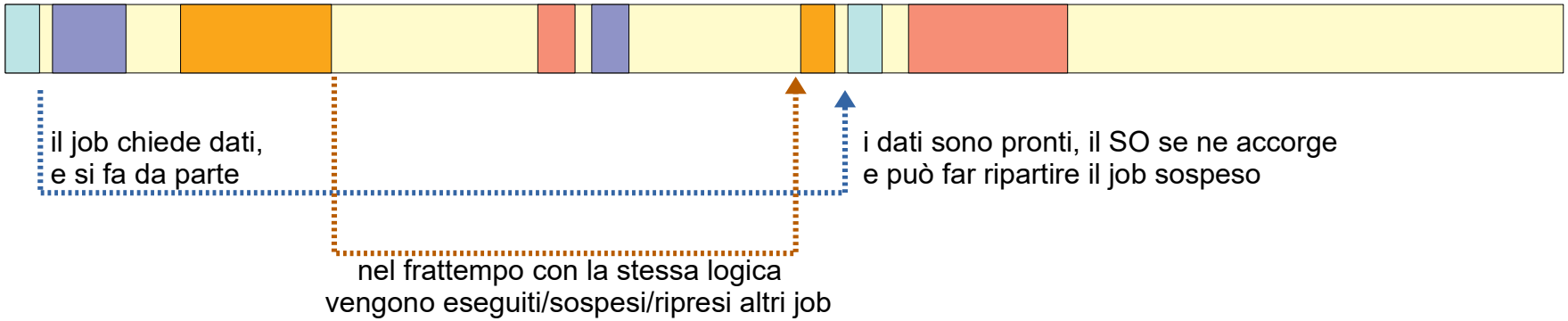
Evoluzione dei SO

- **Difetti dei sistemi batch**
  - **non ci può essere interazione con l'utente**
  - **se un programma alterna calcoli e uso dei dispositivi di I/O, essendo questi tipicamente molto più lenti della CPU, quest'ultima è inutilizzata**



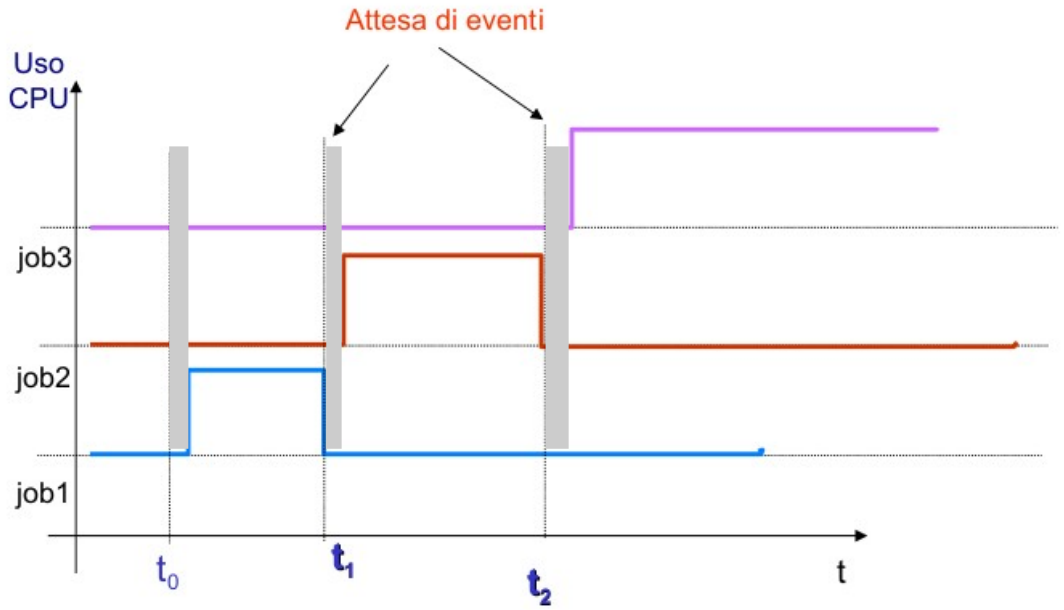
Sistemi batch multiprogrammati

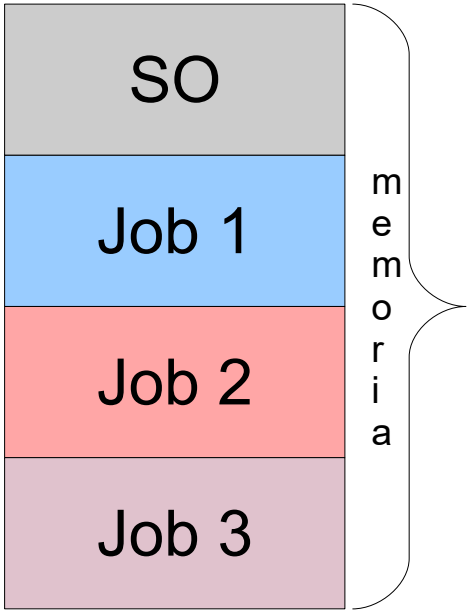
- SO è in grado di portare avanti l'esecuzione di più job contemporaneamente
  - Ad ogni istante:
    - un solo job utilizza la CPU
    - più job sono caricati in memoria centrale, attendono di acquisire la CPU
  - Quando il job che sta utilizzando la CPU chiede l'uso di un dispositivo, si sospende in attesa di un evento che dichiari il completamento dell'operazione
    - SO decide a quale job assegnare la CPU ed effettua lo scambio (scheduling)



Sistemi batch multiprogrammati

- Quali esigenze crea questo modello?
  - per garantire che il SO possa gestire le sospensioni e controllare se il servizio è terminato, **le richieste di I/O devono passare da lui**
  - la presenza di più programmi in memoria genera la necessità di **proteggerli l'uno dall'altro** contro accessi indesiderati





Sistemi time-sharing

- Cosa succede in un sistema batch multiprogrammato se un job non fa mai richieste di I/O al sistema operativo?
  - il job monopolizza il sistema
  - e se non termina mai...

- Introduzione di un tempo limite (**quanto di tempo**)
  - superato il quale il SO riprende forzatamente il controllo
  - se abbastanza breve, l'alternanza tra job è così rapida da dare all'utente **l'illusione** che i diversi programmi siano **simultaneamente** in esecuzione → adatto all'interattività e alla *multiutenza*
  - il passaggio da un job all'altro, a volte potrebbe non essere necessario, ma richiede comunque operazioni svolte dal SO → *overhead*

Riassunto dei requisiti di un SO

1) Il SO deve essere il punto di passaggio obbligato per accedere all'hardware

2) Il SO deve poter proteggere le aree di memoria e le altre risorse dedicate a un job dall'interferenza degli altri

3) Il SO deve poter intervenire quando si verificano eventi importanti, impossessandosi della CPU con prelazione rispetto a qualsiasi job

Soluzioni: (1) device driver

- Avere un punto di passaggio obbligato per accedere all'hardware è un'opportunità ghiotta anche per gli sviluppatori, rispetto al diritto/dovere di gestire ogni possibile dispositivo! Diversamente dovrebbero:
  - includere in ogni applicazione il codice per manipolare ogni interfaccia esistente
  - aggiornarle ogni volta che un nuovo dispositivo I/O viene immesso sul mercato
- Soluzione: il SO include una libreria di **device driver (DD)**
  - un DD è software installato sul sistema, specifico dell'hardware presente, in modo che gli sviluppatori non abbiano bisogno di includerlo nelle applicazioni ...
  - ... perché espone un'interfaccia standard di programmazione, specifica del SO. Un'applicazione sviluppata per un SO userà comandi "generici" per chiedere al DD di svolgere un'operazione; il DD li converirà in specifiche istruzioni in linguaggio macchina, con cui la CPU potrà pilotare attaverso i bus il modello di interfaccia effettivamente installata

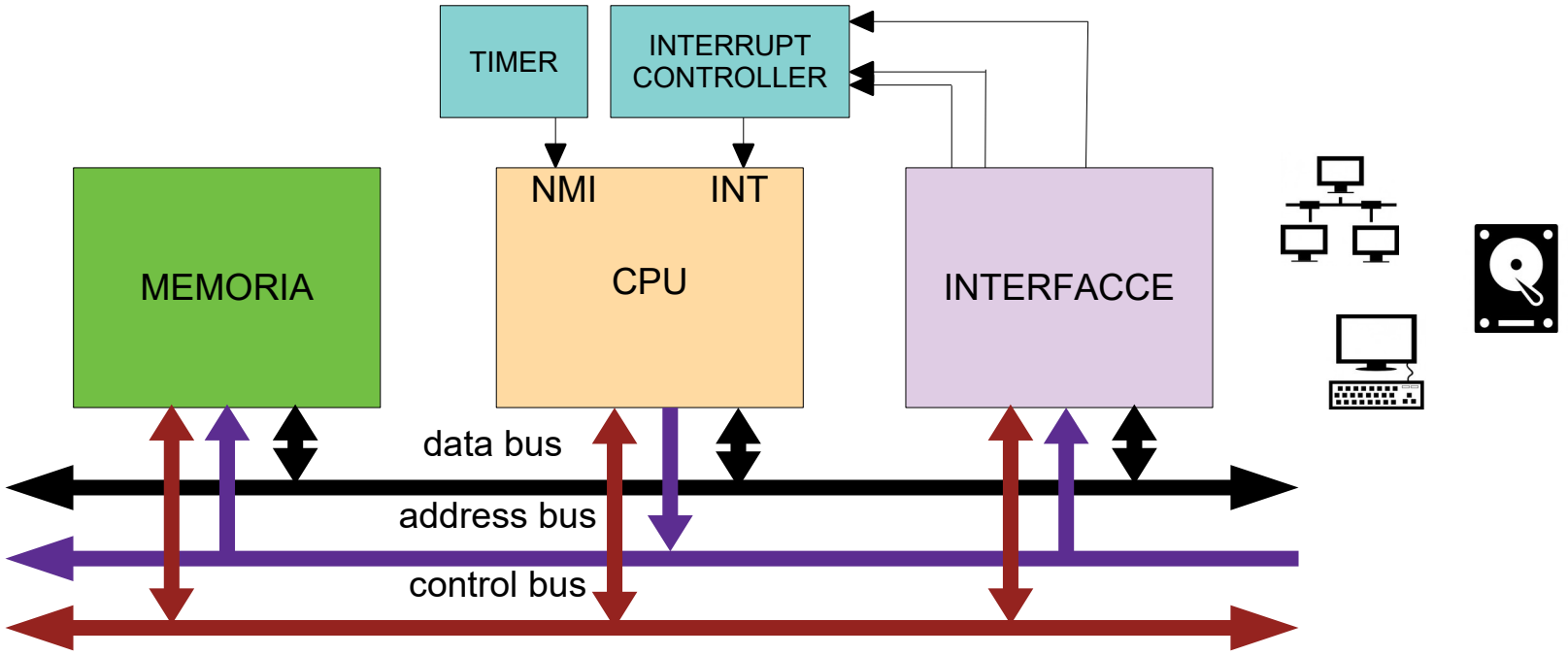
Soluzioni: (1+2) modi della CPU

- Come impedire a uno sviluppatore di mettere nella propria applicazione, direttamente e aggirando i device driver, comandi per azionare un'interfaccia, o per leggere una cella di memoria che non lo riguarda?

- I costruttori di CPU le progettano e realizzano perchè abbiano diversi *modi di funzionamento*. Semplificando al massimo, almeno due:
  - un modo privilegiato (detto anche ***supervisor mode*** o ***kernel mode***)
    - quando la CPU si trova in questo stato, può eseguire qualsiasi istruzione
  - un modo ristretto (detto anche ***user mode***)
    - quando la CPU si trova in questo stato, può eseguire solo istruzioni che non riguardano la gestione hardware e la mappatura delle aree di memoria

- All'avvio del computer, la CPU è in supervisor mode, e inizia a eseguire il SO
- Quando il SO ha caricato in memoria i job, prima di passare il controllo al primo, commuta la CPU in user mode

Soluzioni: (3) il modello a interruzioni



- La CPU può ricevere segnali elettrici dall'esterno che interrompono quel che sta facendo e istruiscono il Program Counter a "saltare" in una diversa zona di memoria, dove troverà istruzioni del sistema operativo
  - per gestire eventi segnalati da interfacce di I/O (configurabili)
  - per gestire lo scadere del quanto di tempo (Non Maskable Interrupt)

Soluzioni: (1+2+3) ultimi dettagli

- Manca solo un tassello: ovviamente, quando la CPU è in user mode, l'applicazione in esecuzione non ha il diritto di riportarla in supervisor mode
  - come può allora funzionare un device driver invocato da un'applicazione per accedere a un dispositivo esterno?
  - come può il SO riservare o liberare memoria quando devono essere avviate o terminate applicazioni?

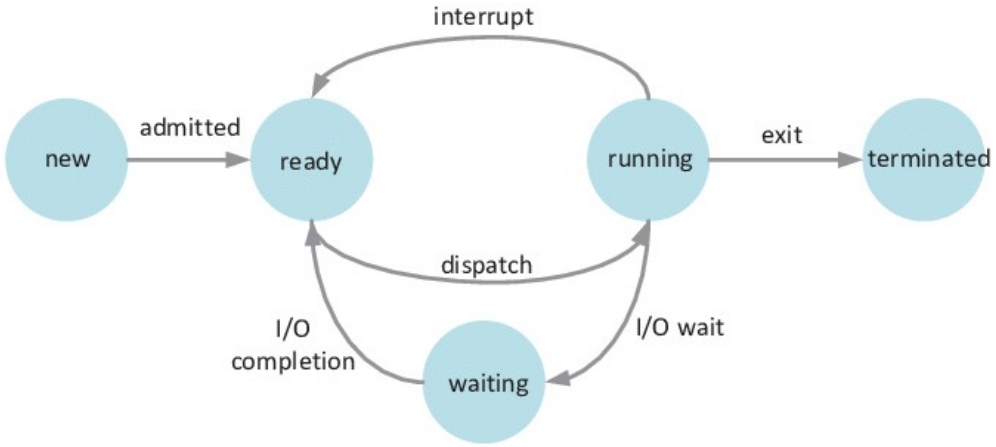
- Soluzione:
  - i DD e il codice che gestisce le interruzioni sono caricati in una zona protetta della memoria
  - possono essere eseguiti solo attraverso il meccanismo delle interruzioni, che automaticamente riporta la CPU in supervisor mode
    - ciò avviene automaticamente quando la CPU riceve un interrupt fisico
    - quando un'applicazione ha bisogno dei servizi di un DD, emula lo stesso fenomeno per mezzo di un'istruzione speciale: un interrupt software

Il ciclo di vita di un processo

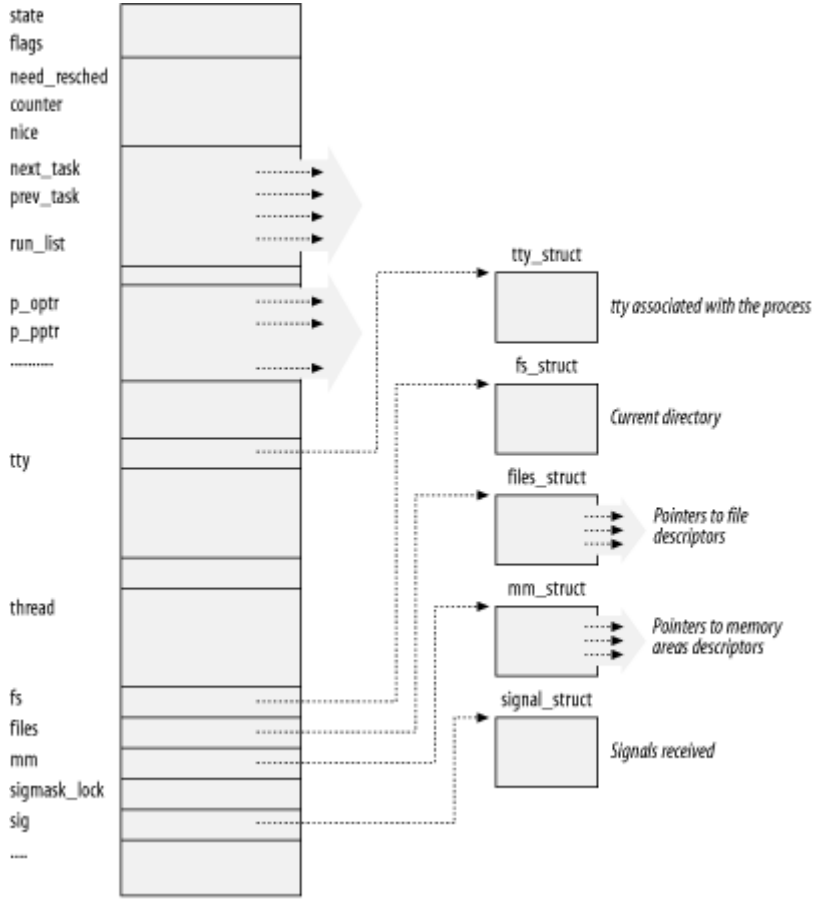
- Abbiamo chiarito che
  - le istruzioni possono essere lette solo dalla memoria principale
    - inizialmente ROM → indispensabile per avere qualcosa da eseguire appena il computer viene acceso
    - successivamente RAM → indispensabile per avere flessibilità di caricarci qualsiasi cosa l'utente desideri
  - la RAM è volatile, quindi tutti i programmi e i dati durevoli devono essere salvati su hard disk di qualche natura
- Definiamo meglio i termini
  - un **PROGRAMMA** è un'entità passiva: un elenco di istruzioni comprensibili dalla CPU, memorizzato su di un supporto persistente, che non ha nessun effetto finchè non viene
    - copiato nella RAM
    - equipaggiato di una serie di metadati che ne descrivono il progresso
    - eseguito, facendo sì che il Program Counter punti alle celle di memoria che lo contengono per fare il *fetch* delle istruzioni
  - in quel momento, diventa un **PROCESSO**

Il ciclo di vita di un processo

- Il sistema operativo come detto non è altro che un programma
  - viene caricato in memoria dal BIOS, il programma presente in ROM
- Si occupa poi di gestire tutte le fasi che
  - trasformano un programma in un processo
  - curano l'esecuzione, la sospensione, la protezione delle risorse di ogni processo



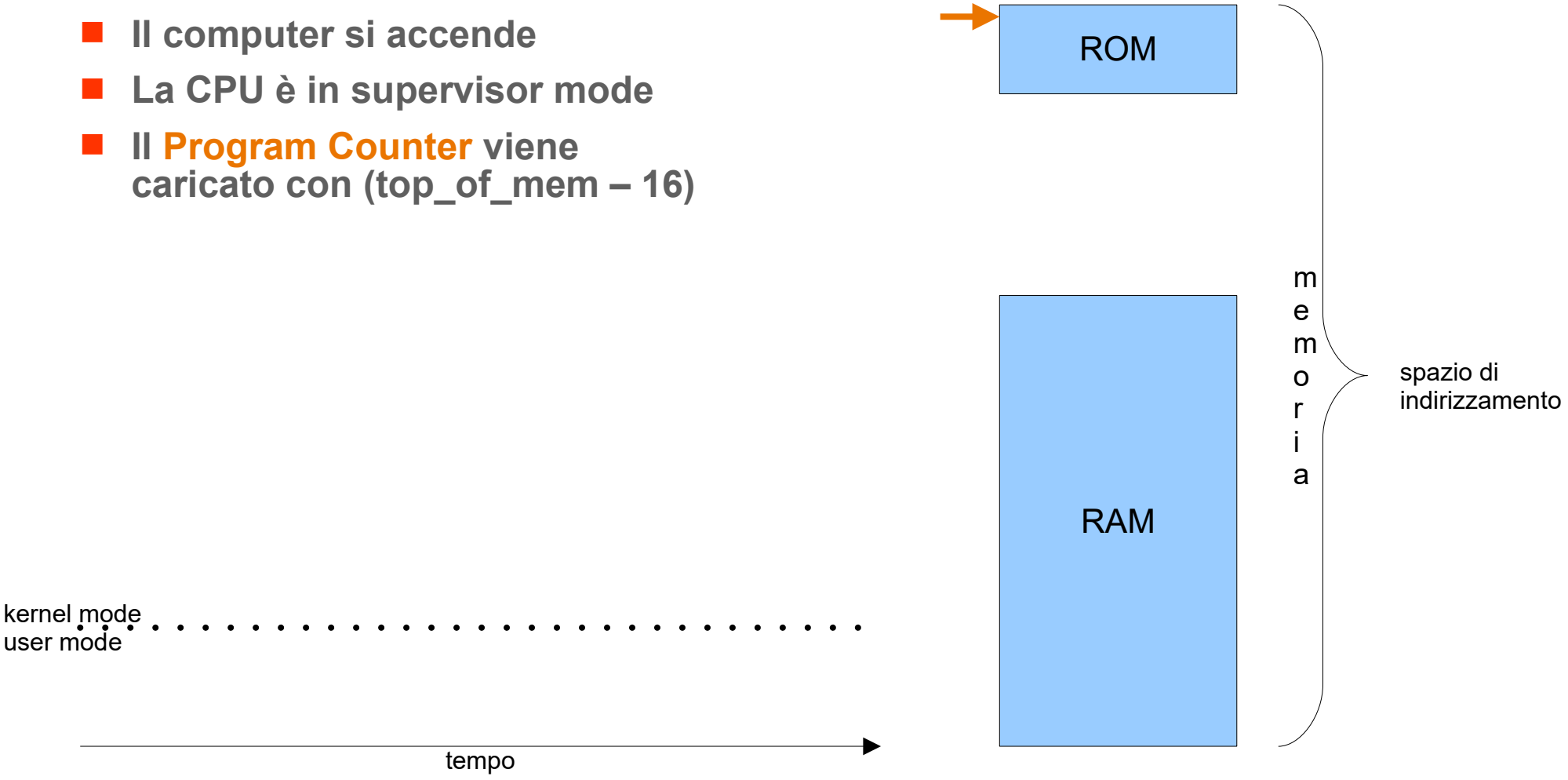
I principali stati di un processo



Il descrittore di un processo in Linux
da Bovet, Cesati
"Understanding the Linux Kernel, Second Edition"
O'Reilly

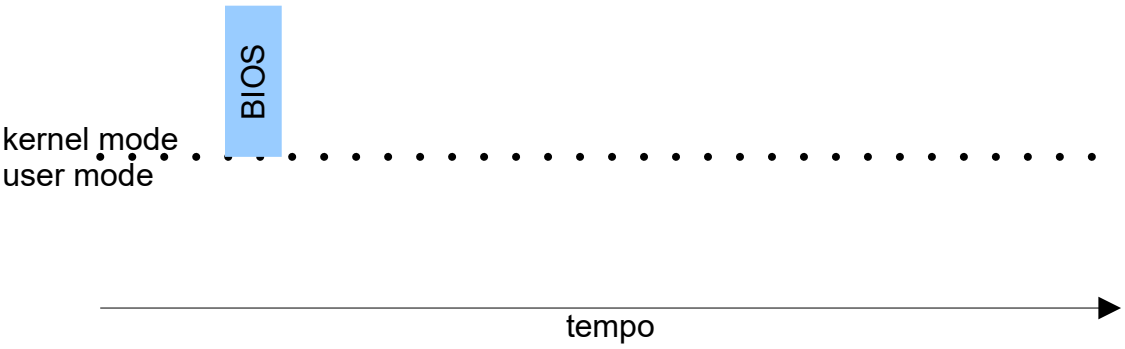
Un esempio

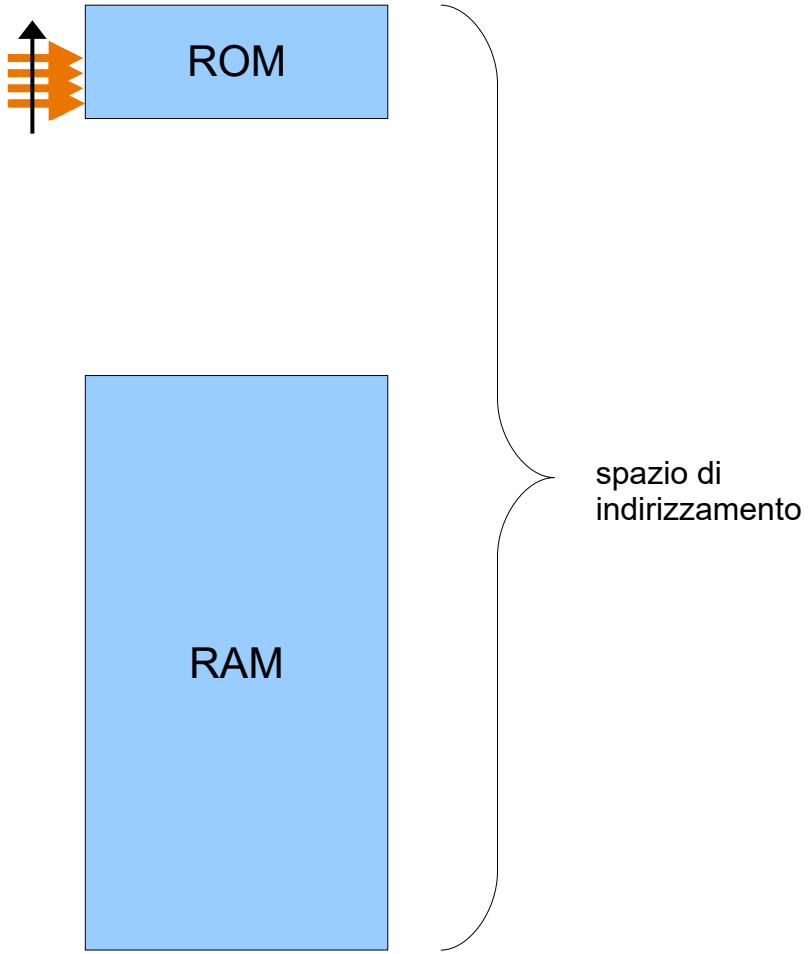
- Il computer si accende
- La CPU è in supervisor mode
- Il **Program Counter** viene caricato con (top_of_mem – 16)



Un esempio

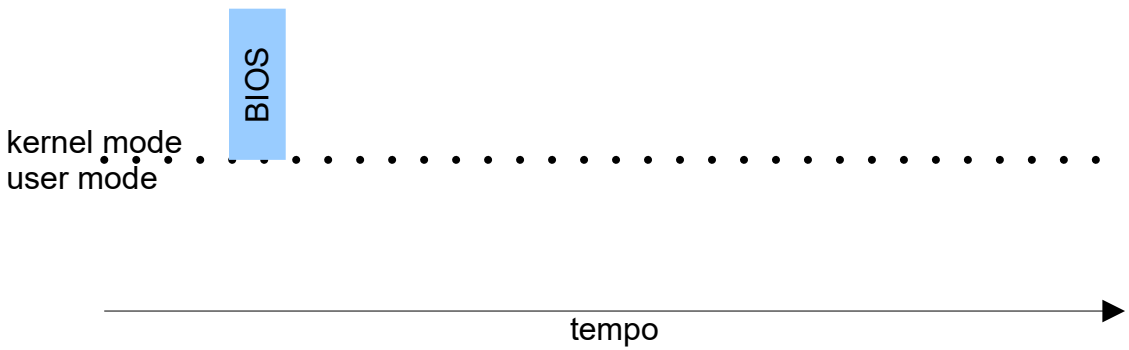
- Il computer si accende
- La CPU è in supervisor mode
- Il **Program Counter** viene caricato con (top_of_mem – 16)
- Viene eseguito il BIOS
  - che fa moltissime cose...

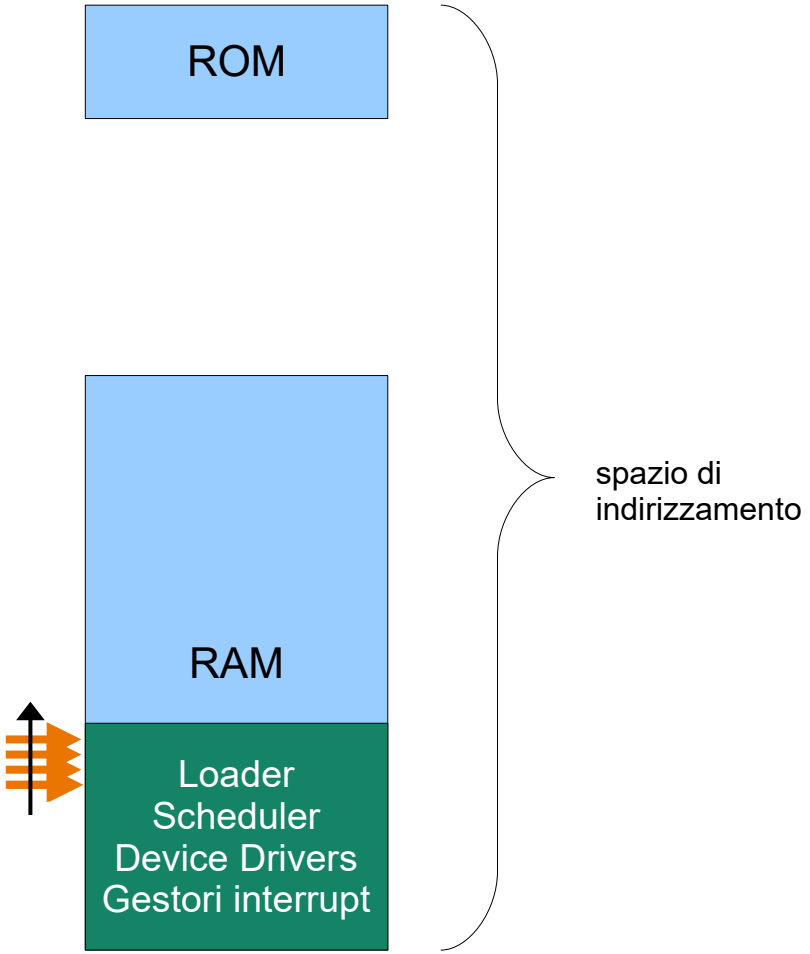




Un esempio

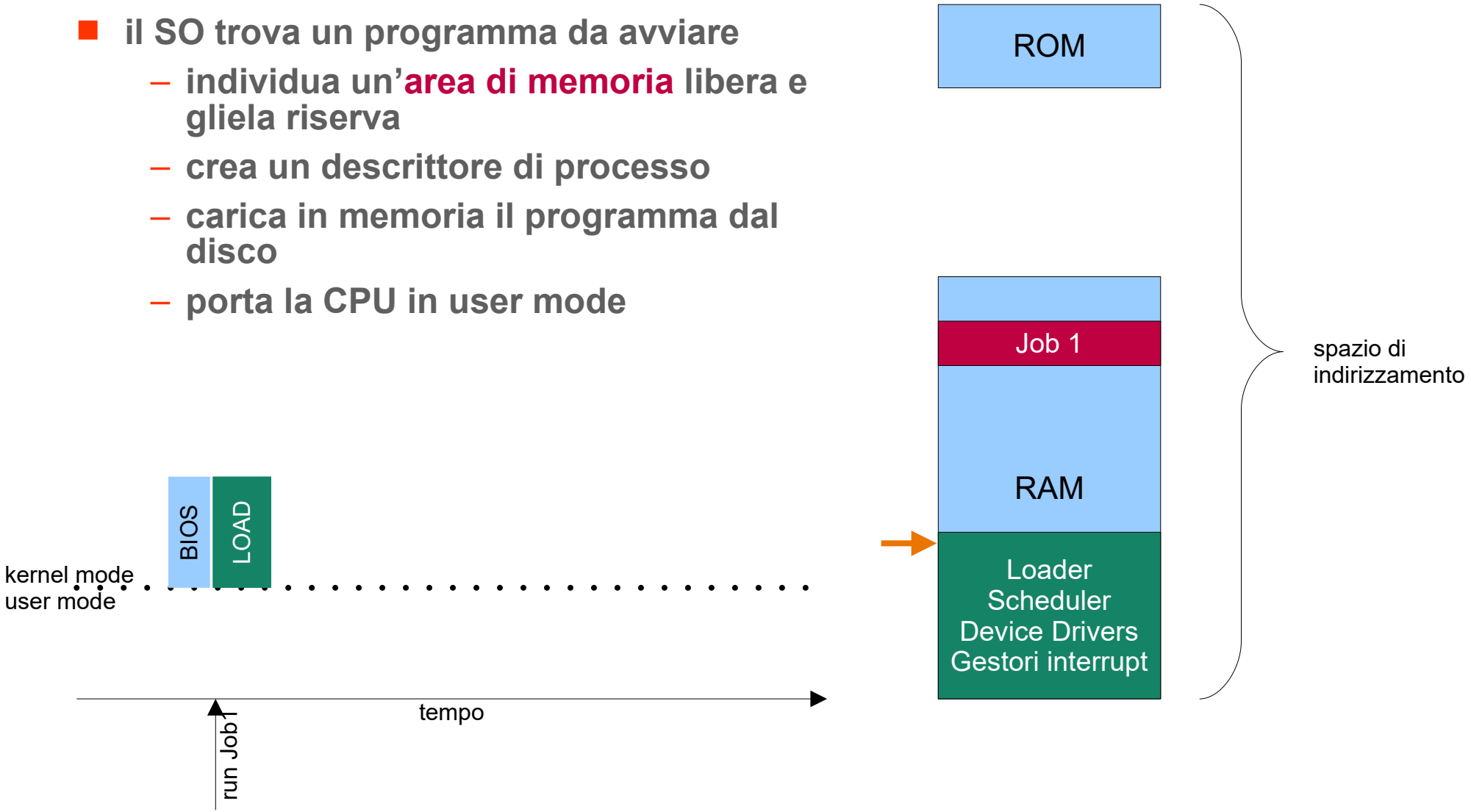
- Il computer si accende
- La CPU è in supervisor mode
- Il **Program Counter** viene caricato con (top_of_mem – 16)
- Viene eseguito il BIOS
  - che fa moltissime cose...
  - alla fine:
- è stato caricato il **SO**
  - il SO cerca un programma da avviare





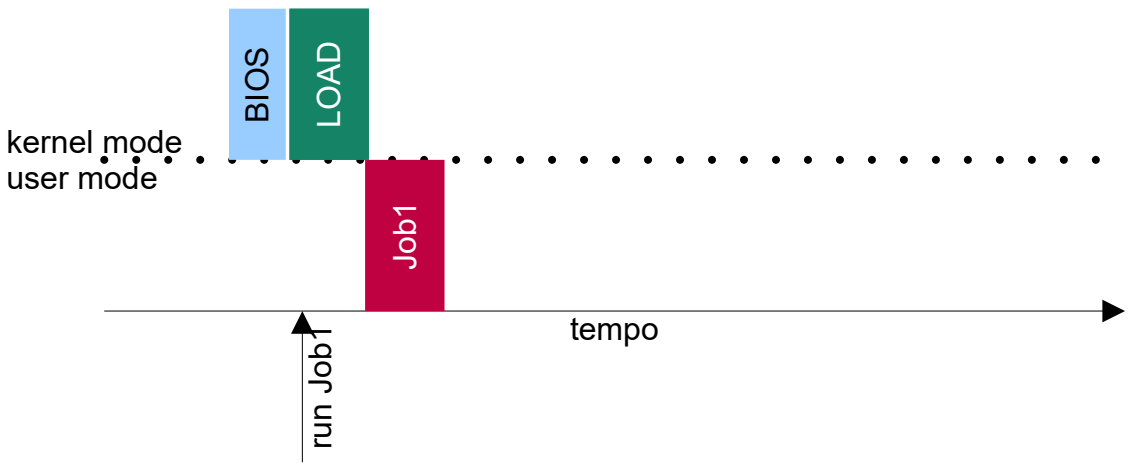
Un esempio

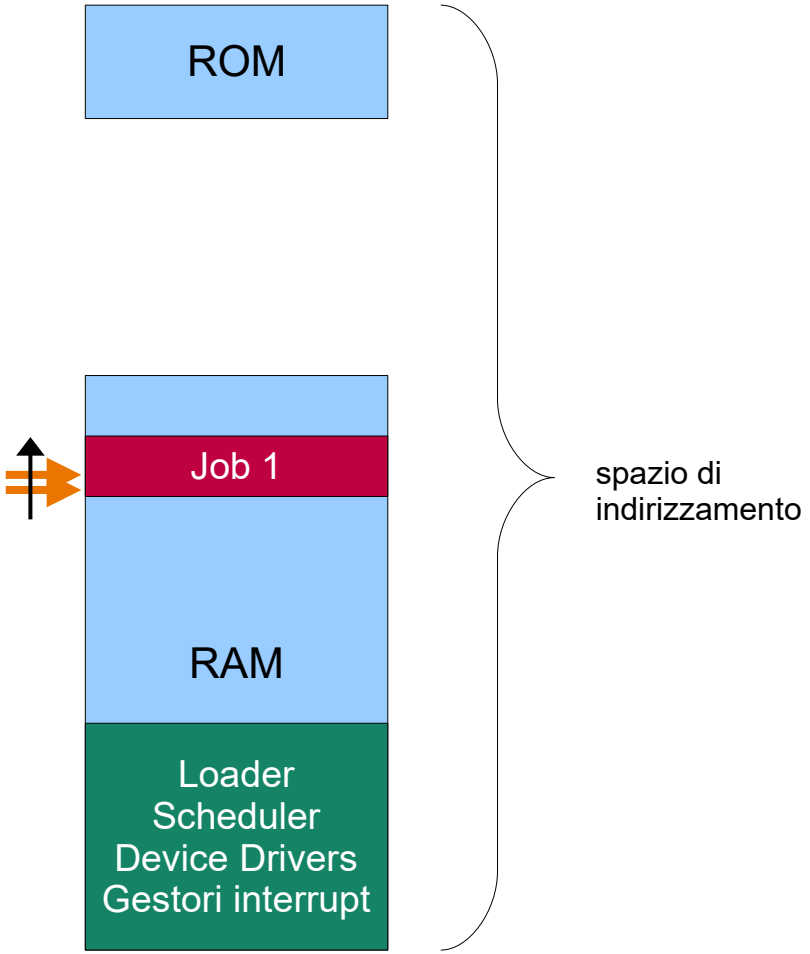
- il SO trova un programma da avviare
  - individua un'**area di memoria** libera e gliela riserva
  - crea un descrittore di processo
  - carica in memoria il programma dal disco
  - porta la CPU in user mode



Un esempio

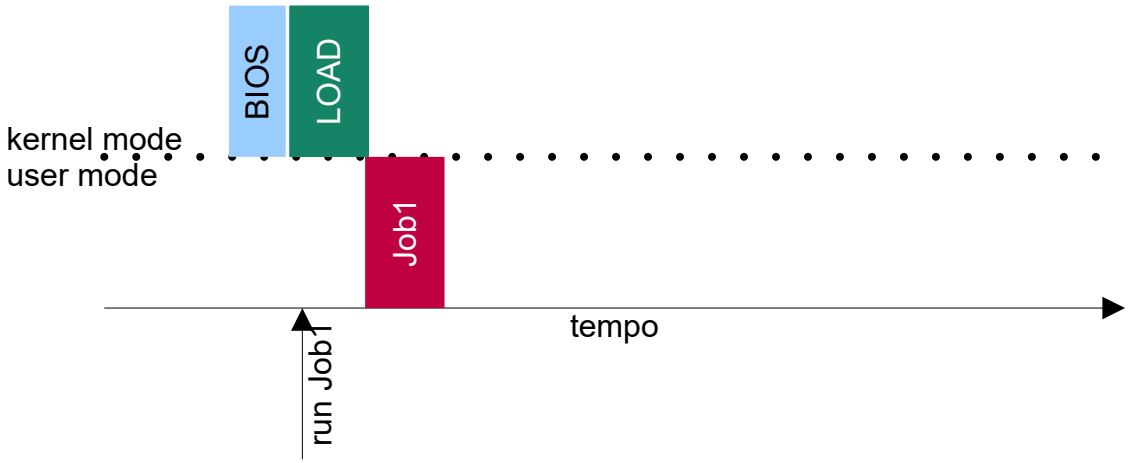
- il SO avvia il processo, caricando nel PC l'indirizzo di memoria dove è presente la prima istruzione del programma
- Il processo avanza istruzione per istruzione lungo il programma

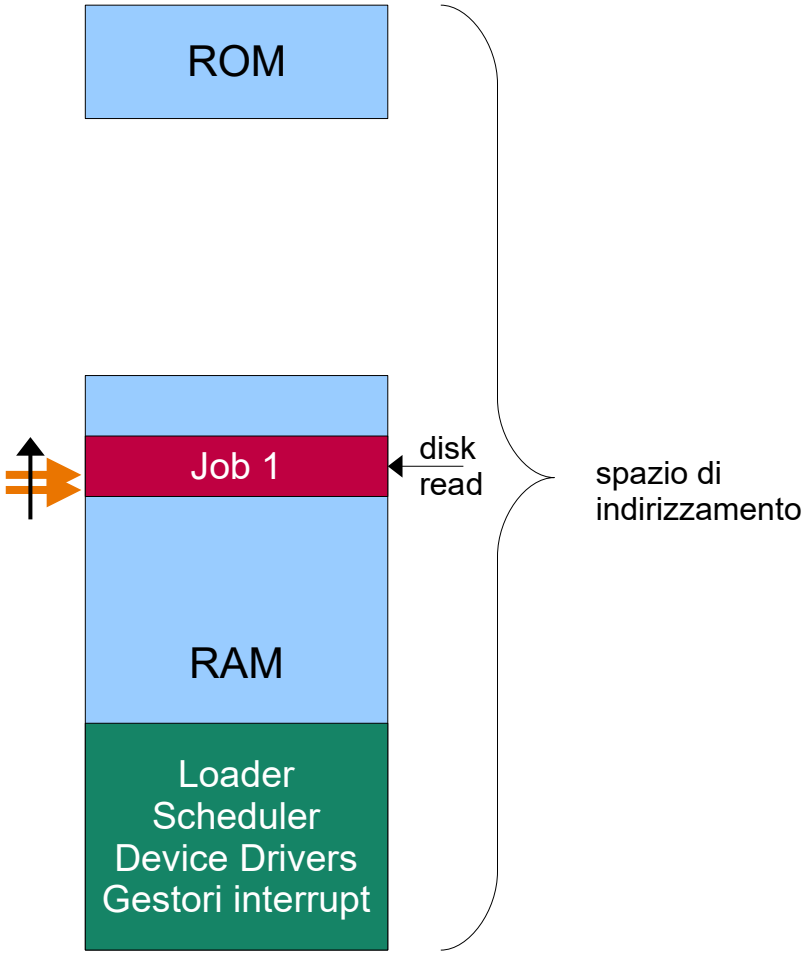




Un esempio

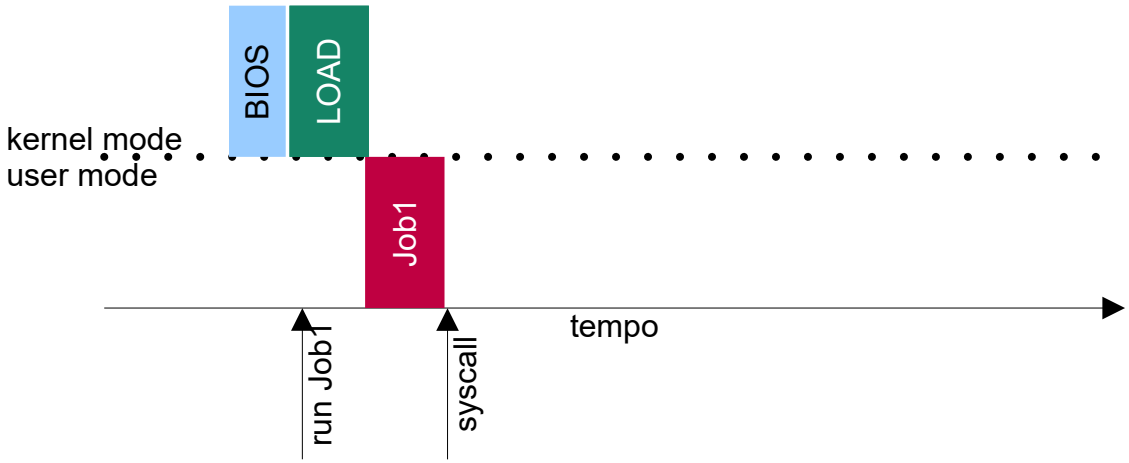
- Il processo avanza istruzione per istruzione lungo il programma
  - incontra un'istruzione che necessita di accedere al disco
  - predispone i parametri per una *system call* (invocazione di una funzione del SO)

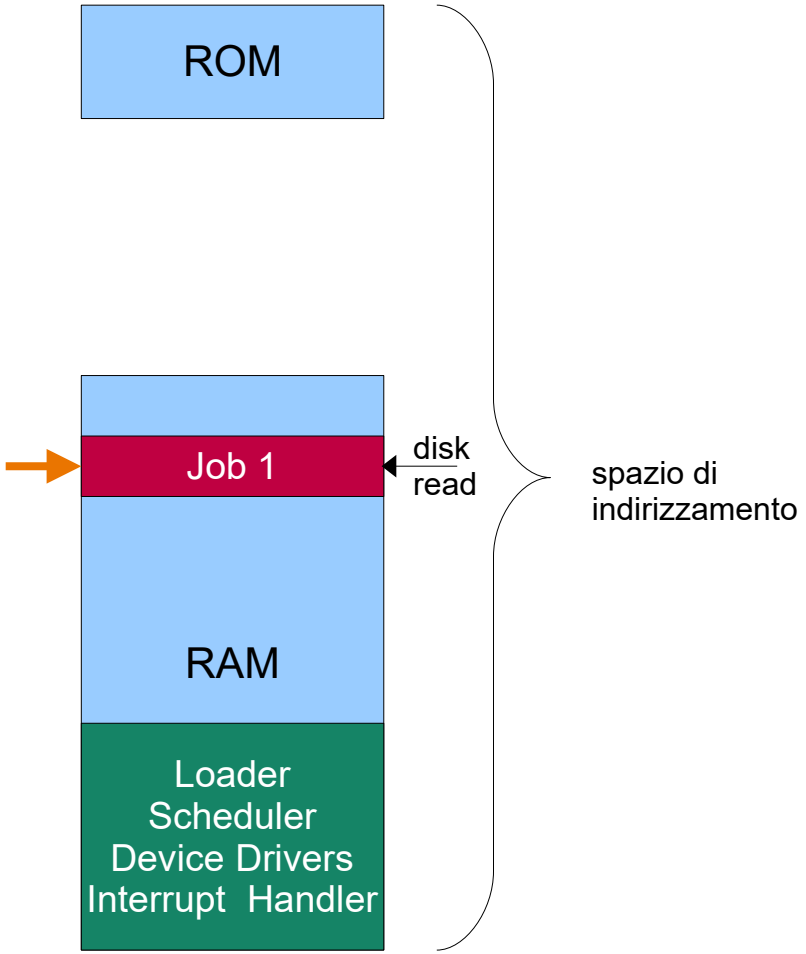




Un esempio

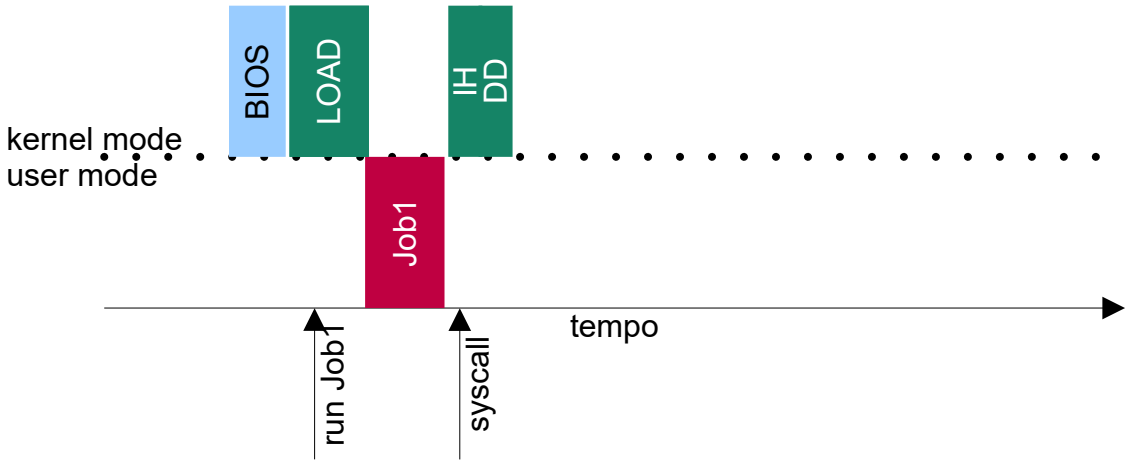
- Il processo avanza istruzione per istruzione lungo il programma
  - incontra un'istruzione che necessita di accedere al disco
  - predispone i parametri per una *system call* (invocazione di una funzione del SO)
  - invoca l'interruzione software per chiedere l'esecuzione della syscall

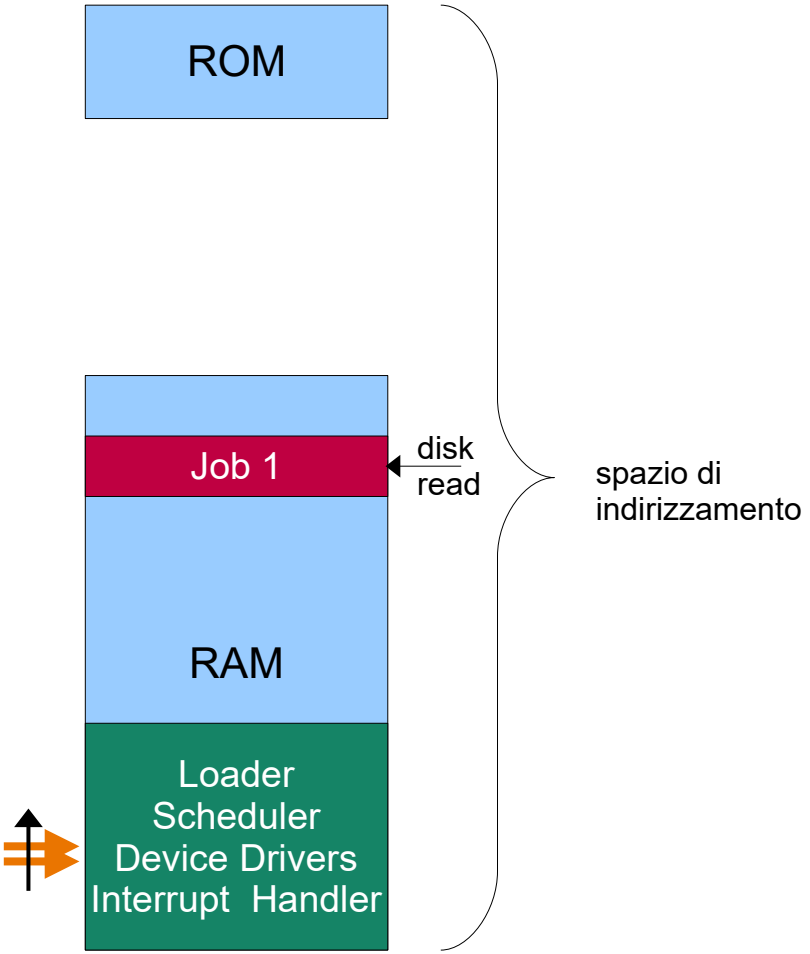




Un esempio

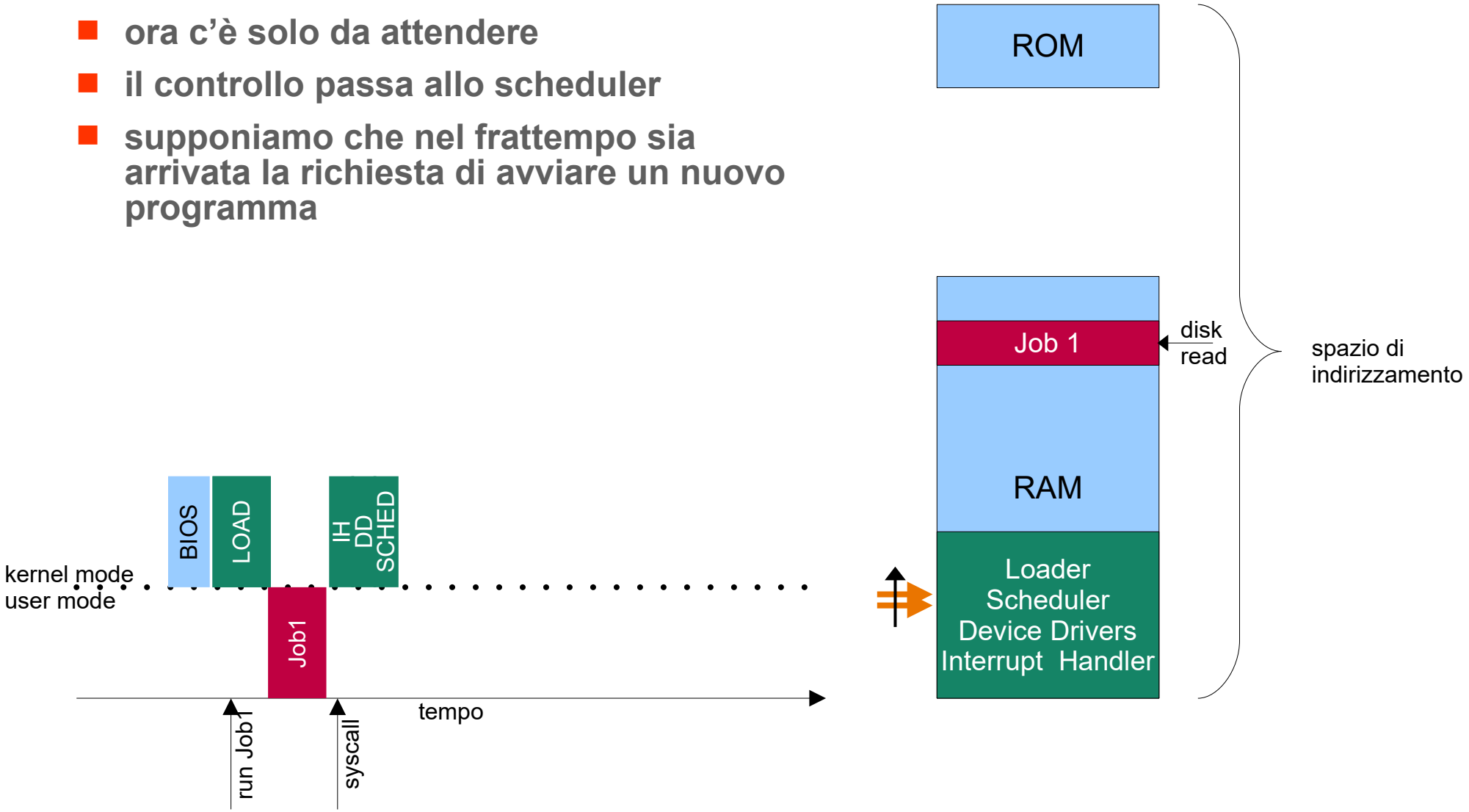
- il soft interrupt porta la CPU in supervisor mode
- il SO prende nota (nel descrittore di processo) del valore che aveva il PC, cioè a che punto era arrivata l'esecuzione di Job 1
- il SO carica nel PC l'indirizzo della system call
- il codice della syscall viene eseguito e invia al disco la richiesta di dati, utilizzando il DD corrispondente





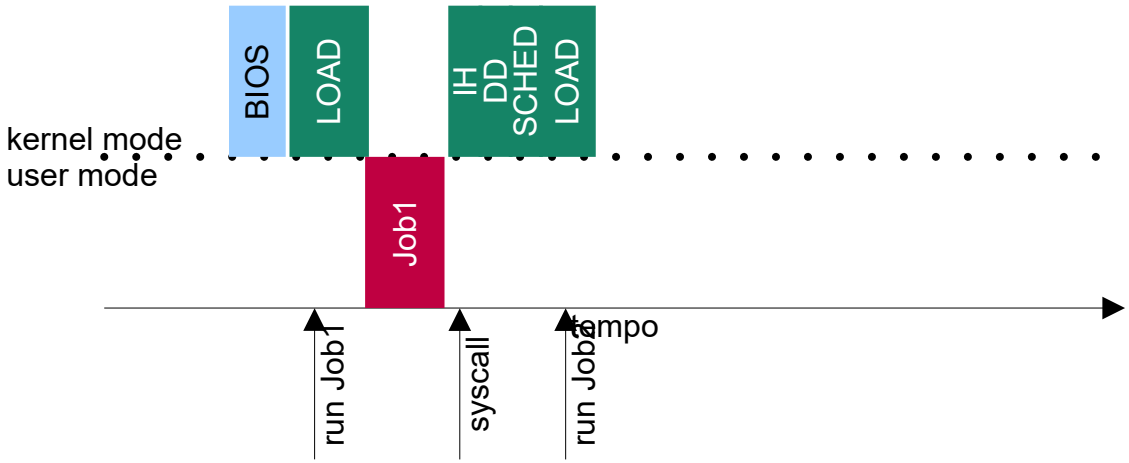
Un esempio

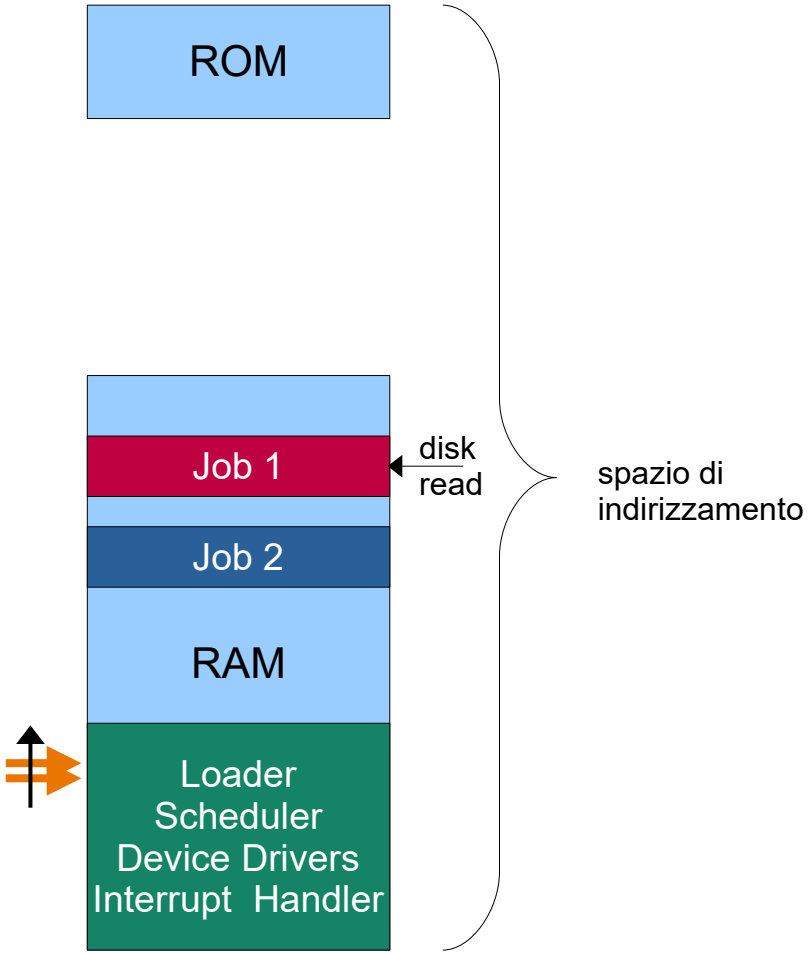
- ora c’è solo da attendere
- il controllo passa allo scheduler
- supponiamo che nel frattempo sia arrivata la richiesta di avviare un nuovo programma



Un esempio

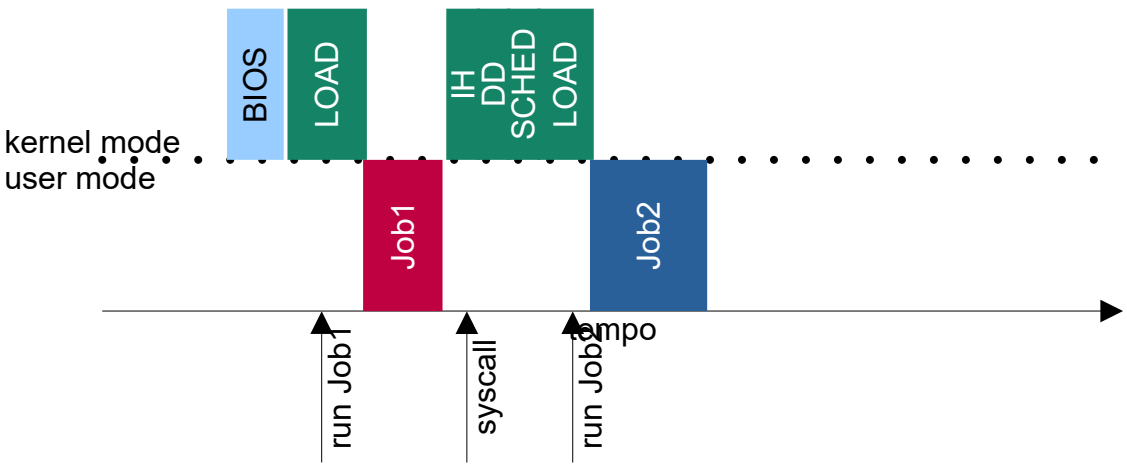
- supponiamo che nel frattempo sia arrivata la richiesta di avviare un nuovo programma
  - si ripete quanto visto per Job 1
  - il SO individua un'area di memoria libera e gliela riserva
  - crea un descrittore per il processo
  - carica in memoria il programma dal disco
  - porta la CPU in user mode

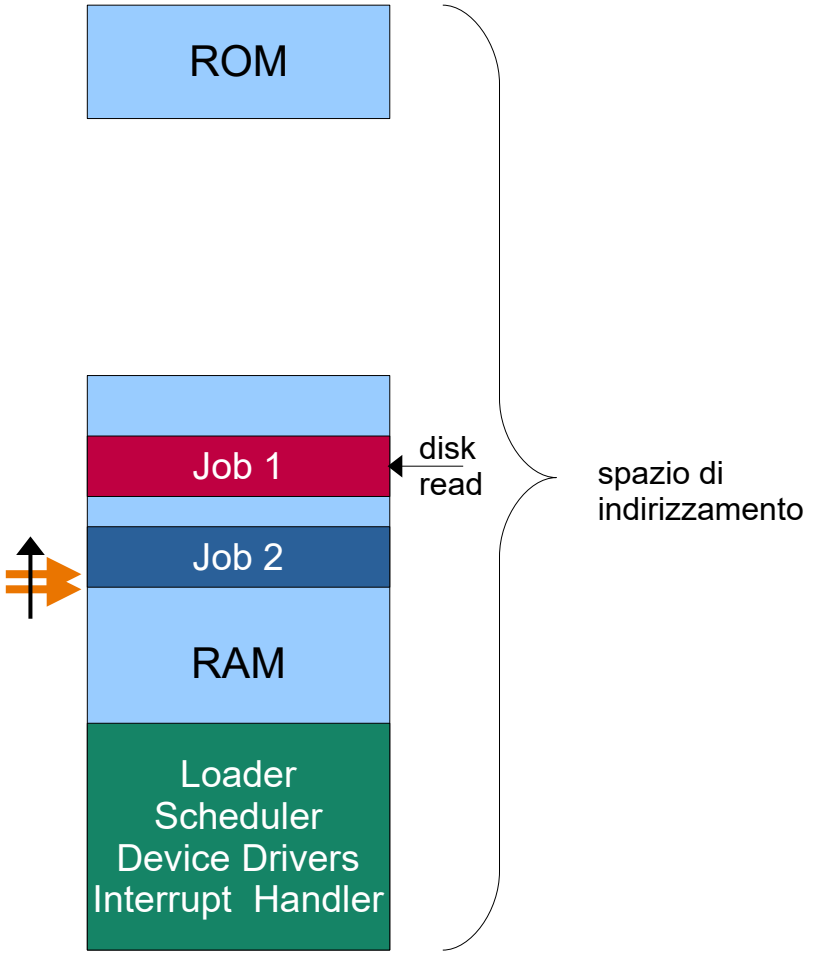




Un esempio

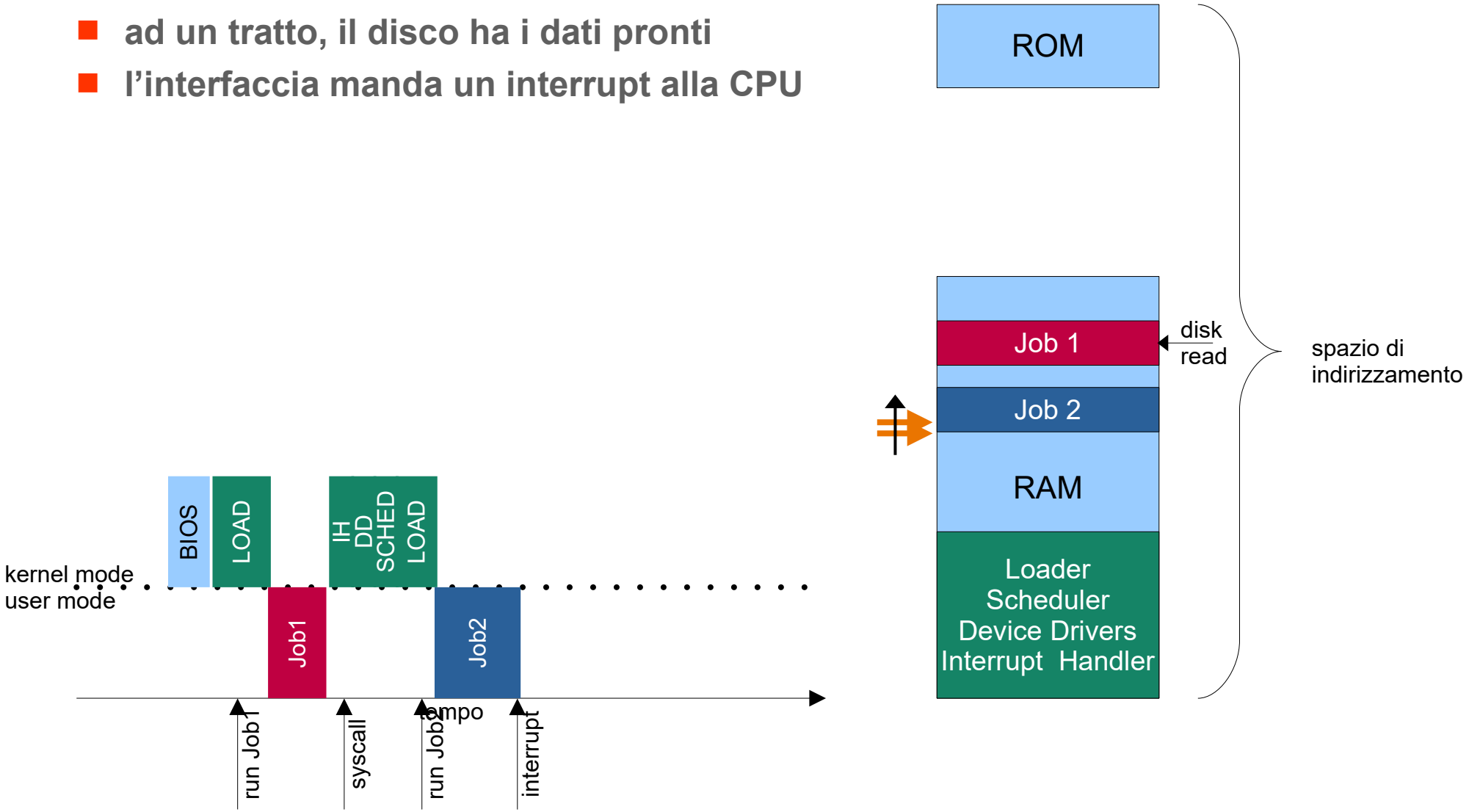
- il SO avvia il processo, caricando nel PC l'indirizzo di memoria dove è presente la prima istruzione del programma
- Il processo avanza istruzione per istruzione lungo il programma





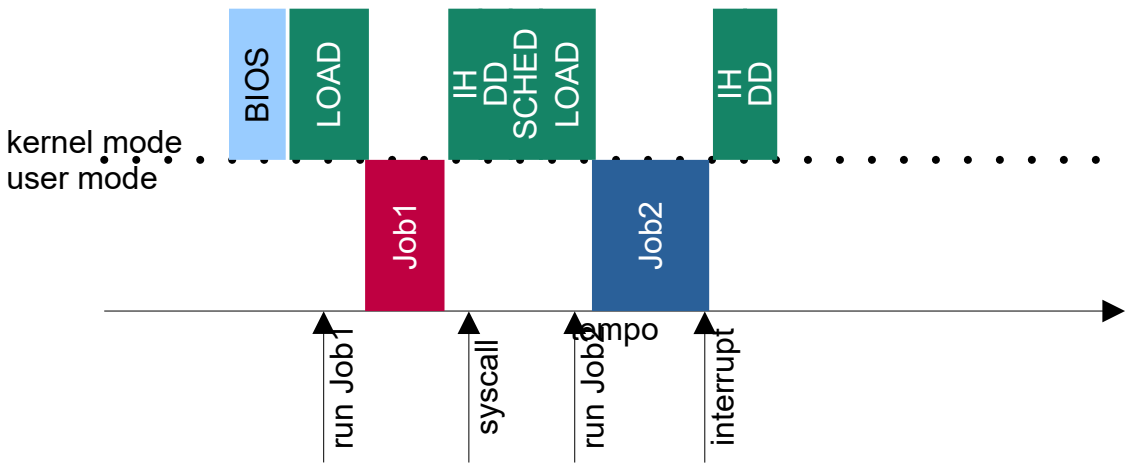
Un esempio

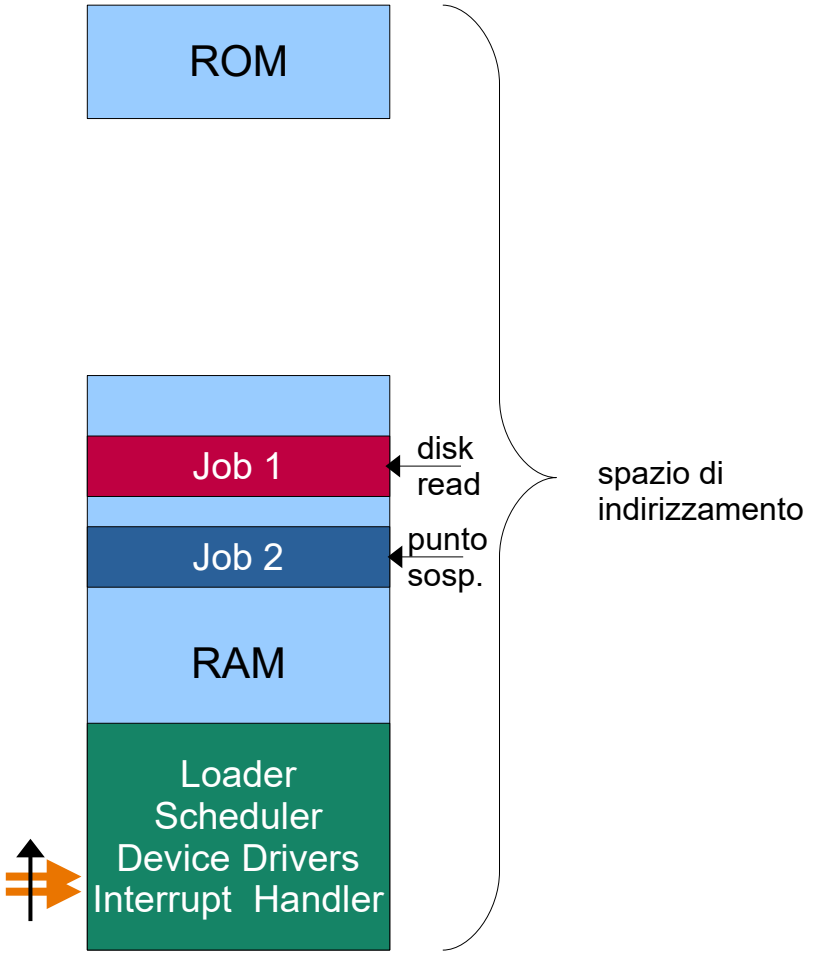
- ad un tratto, il disco ha i dati pronti
- l'interfaccia manda un interrupt alla CPU



Un esempio

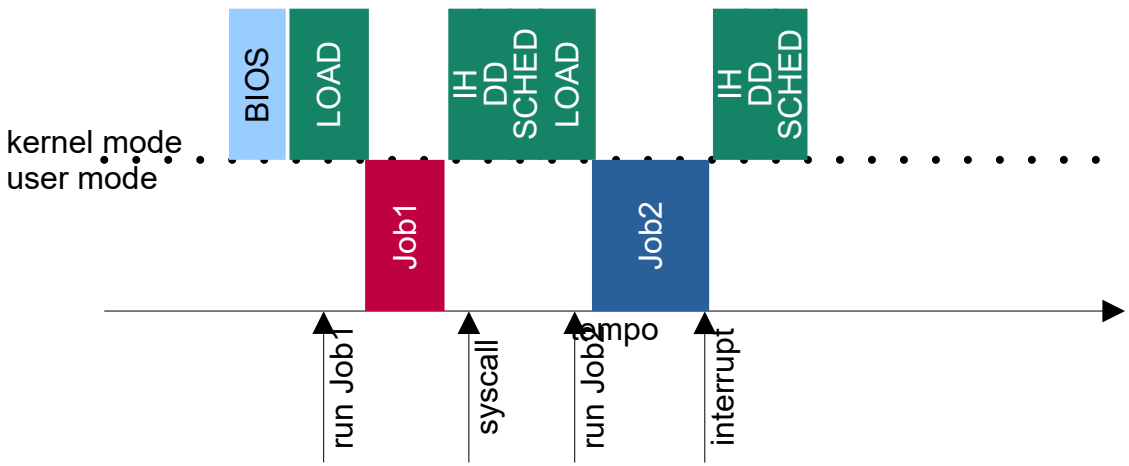
- il disco ha i dati pronti
- l'interfaccia manda un interrupt alla CPU
- l'esecuzione di Job 2 viene interrotta, il SO prende nota (nel descrittore di processo) del valore raggiunto dal PC fino a quel momento
- la CPU si porta in supervisor mode
- viene eseguito il gestore dell'interrupt
  - legge i dati dall'interfaccia e li scrive nella memoria accessibile a Job 1

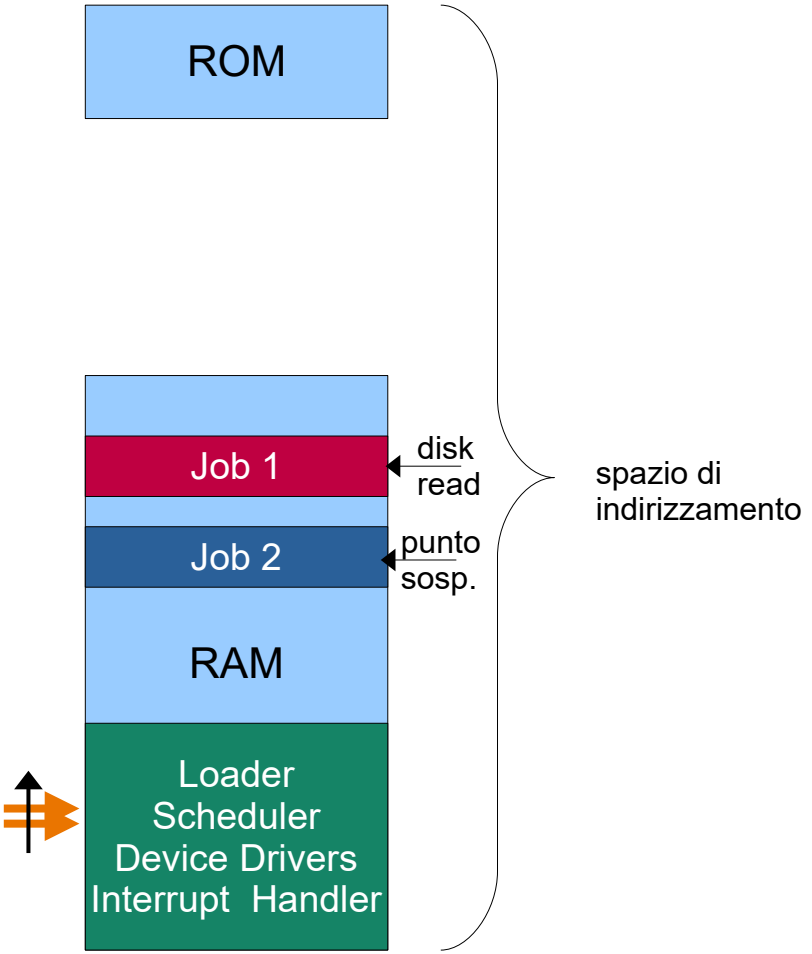




Un esempio

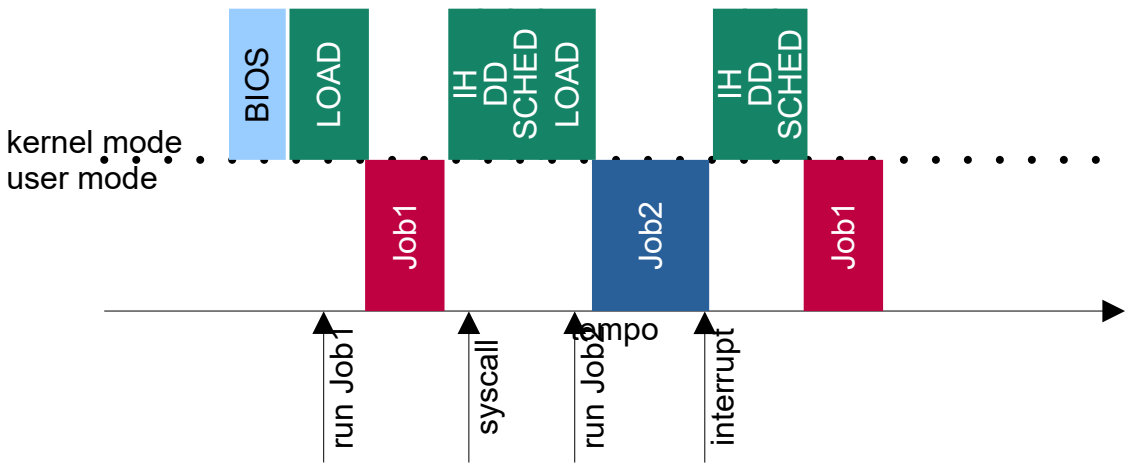
- completata la gestione dell'interrupt, il controllo torna allo scheduler
- a questo punto i due processi potrebbero proseguire entrambi
- lo scheduler seleziona quello a maggior priorità, supponiamo Job 1

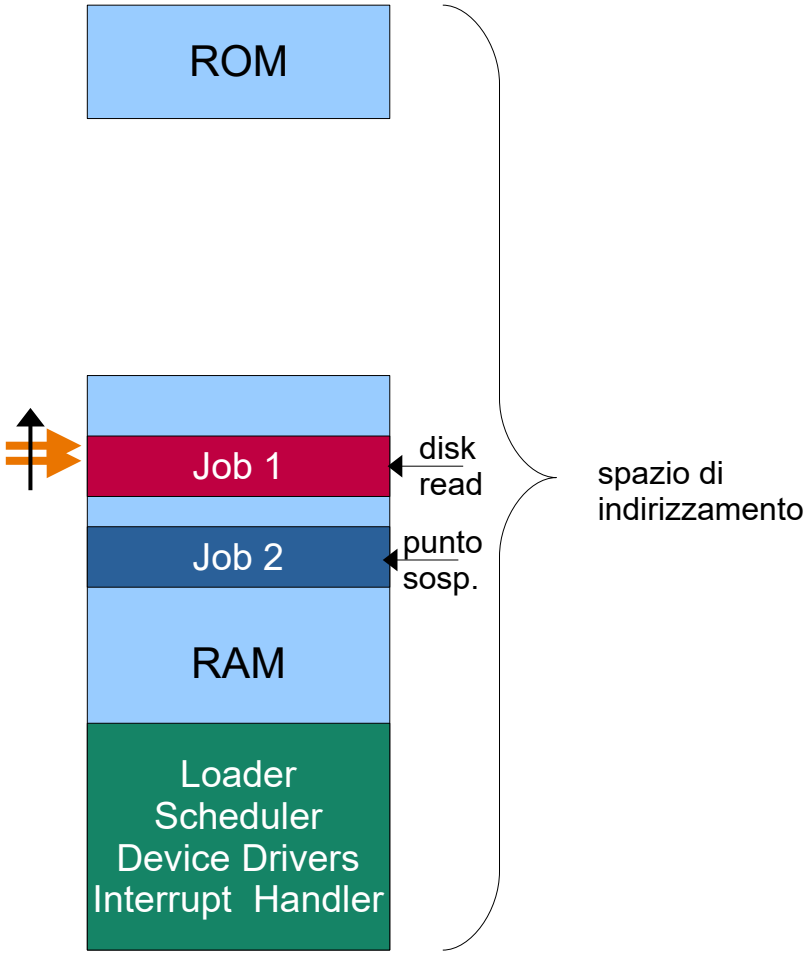




Un esempio

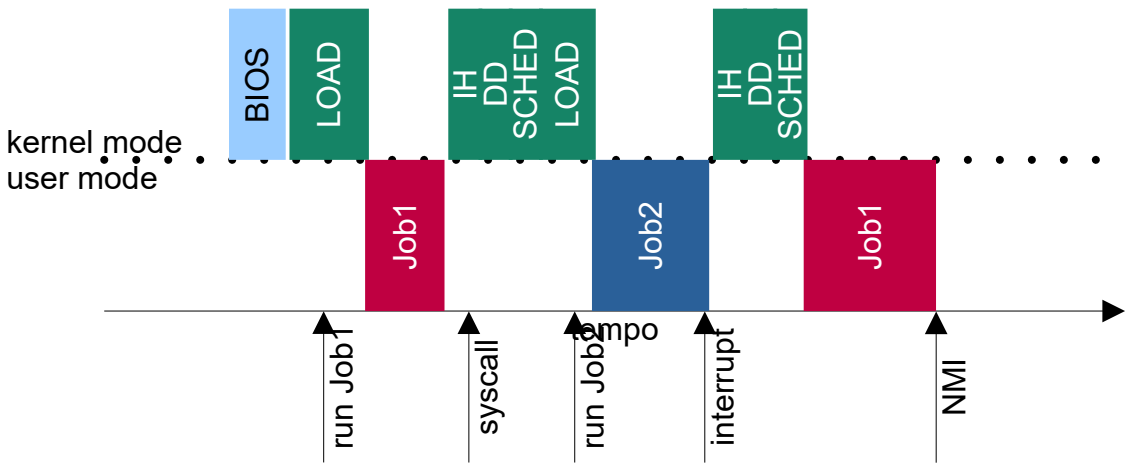
- la CPU viene portata in user mode
- Job 1 riprende esattamente da dove si era fermato, caricando nel PC l'indirizzo salvato nel descrittore del processo

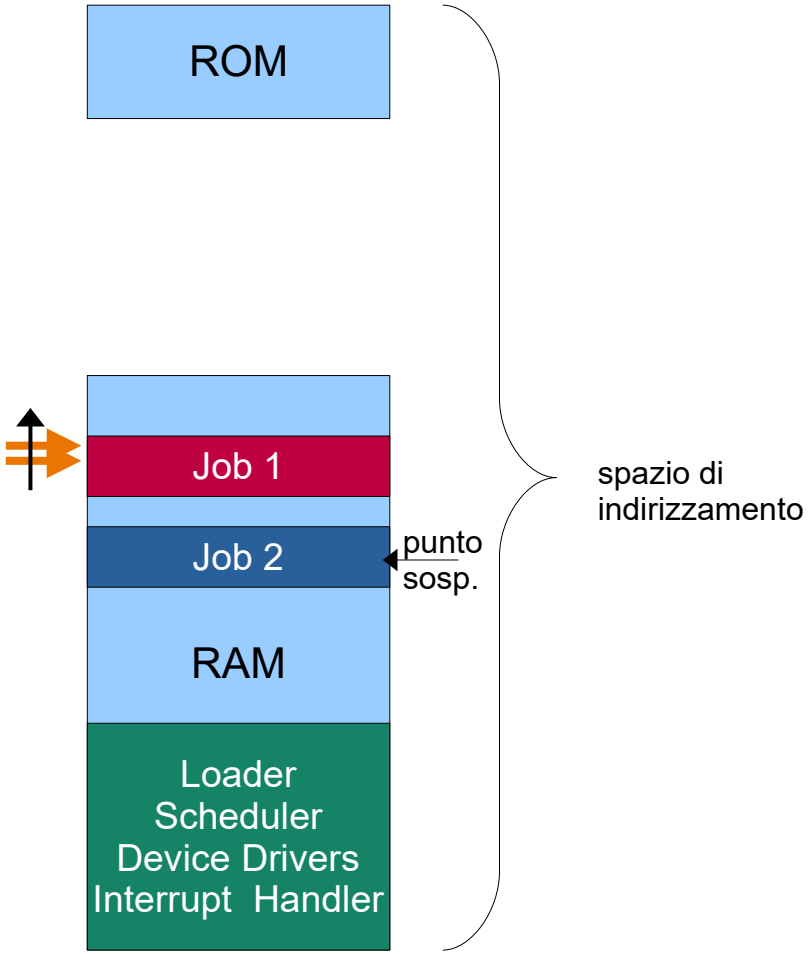




Un esempio

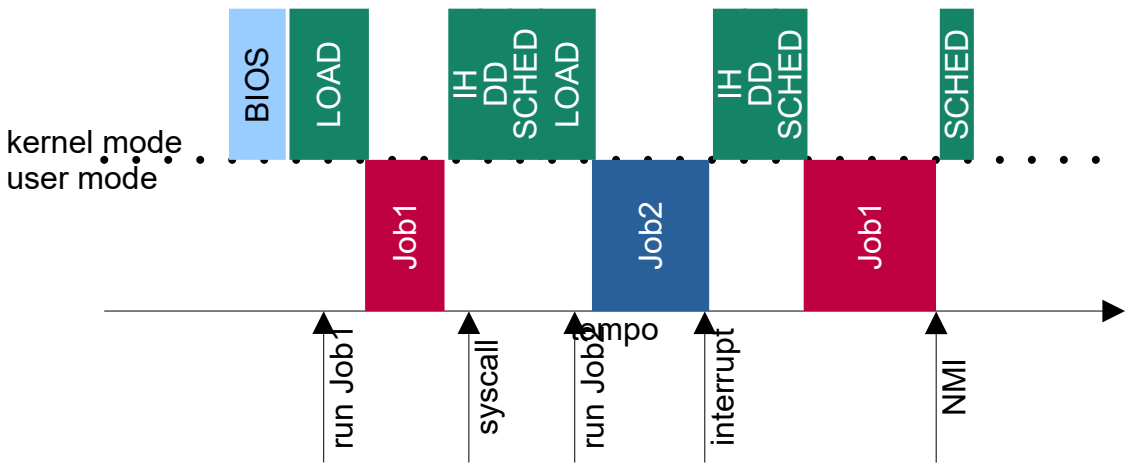
- la CPU viene portata in user mode
- Job 1 riprende esattamente da dove si era fermato, caricando nel PC l'indirizzo salvato nel descrittore del processo all'atto della chiamata di sistema
- supponiamo che proceda senza sosta fino a esaurire il proprio quanto di tempo
- arriva un NMI dal timer

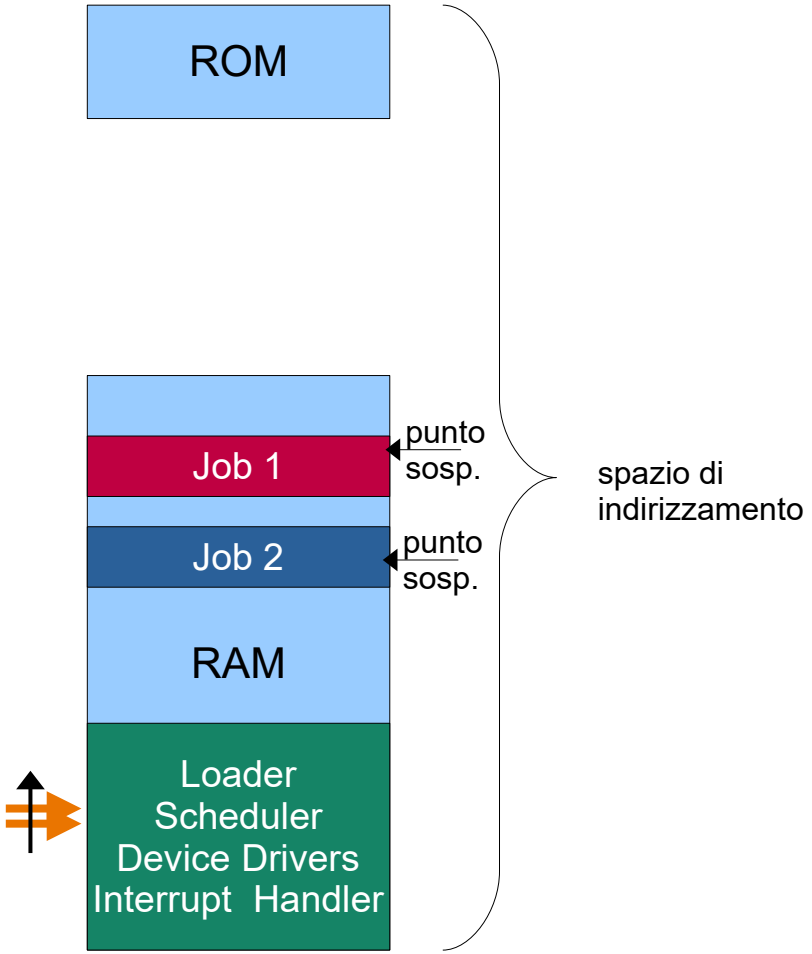




Un esempio

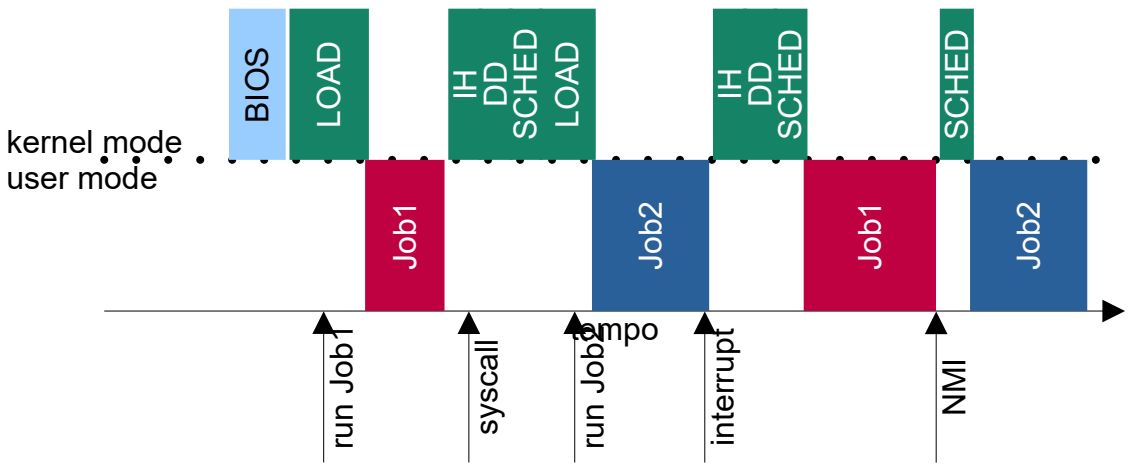
- l'esecuzione di Job 1 viene interrotta, il SO salva di nuovo nel descrittore del processo il valore raggiunto dal PC fino a quel momento
- la CPU si porta in supervisor mode
- viene eseguito il gestore dell'interrupt, che in questo caso invoca lo scheduler

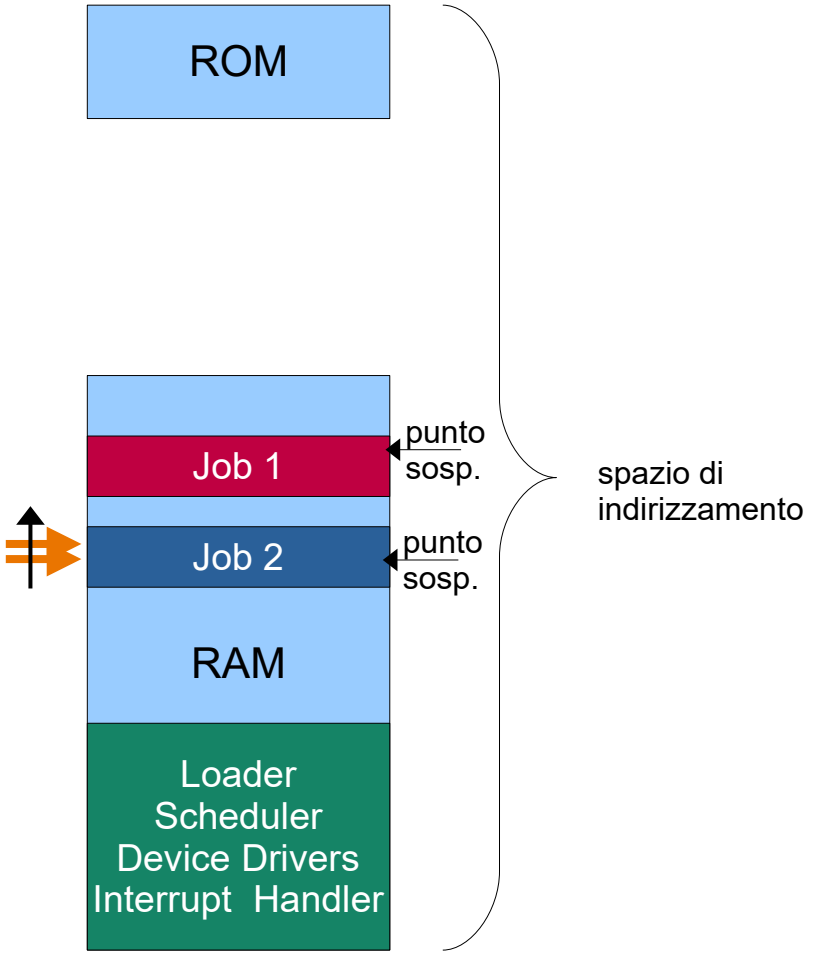




Un esempio

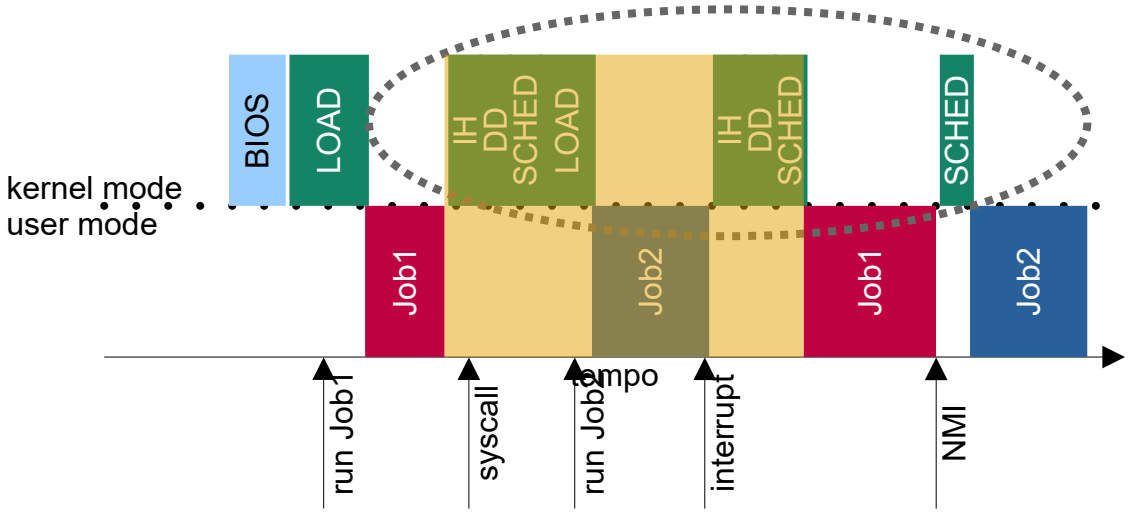
- lo scheduler decide che è il turno di Job 2
- riporta la CPU in user mode
- riavvia Job 2 da dove era stato sospeso, caricando nel PC il valore salvato nel descrittore del processo all'atto dell'esecuzione del gestore di interrupt
- ecc. ecc. ...





Un esempio

- Nota: tutto quel che c'è nella parte alta della timeline è overhead!
  - La CPU viene usata per eseguire le istruzioni del SO invece di quelle dei processi dell'utente
- Ma senza quell'overhead, in questo periodo la CPU sarebbe rimasta in ozio invece che eseguire una parte di Job 2
- Il grafico non è in scala:
  l'overhead è molto variabile ma dell'ordine dei pochi %

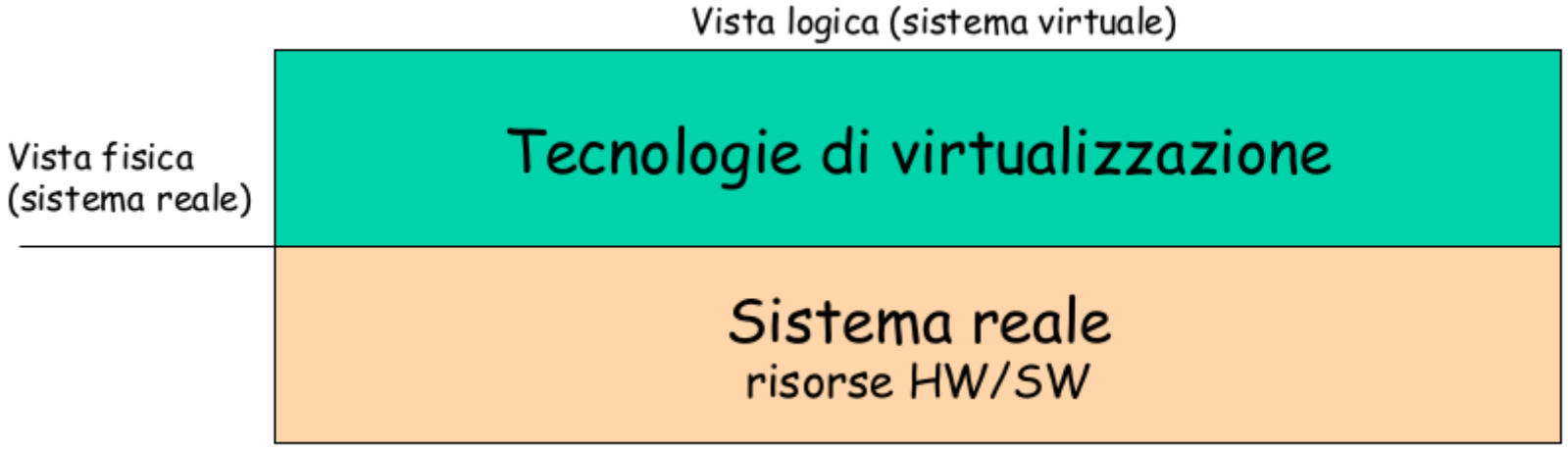


Virtualizzazione e cloud

slide in parte tratte dal materiale di Anna Ciampolini, Antonio Corradi e Luca Foschini

Virtualizzazione

- Dato un sistema caratterizzato da un insieme di risorse (hardware e software), virtualizzare il sistema significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.
- Ciò si ottiene introducendo un livello di indirezione tra la vista logica e quella fisica delle risorse.
- **Obiettivo: disaccoppiare il comportamento delle risorse di un sistema di elaborazione offerte all'utente dalla loro realizzazione fisica.**

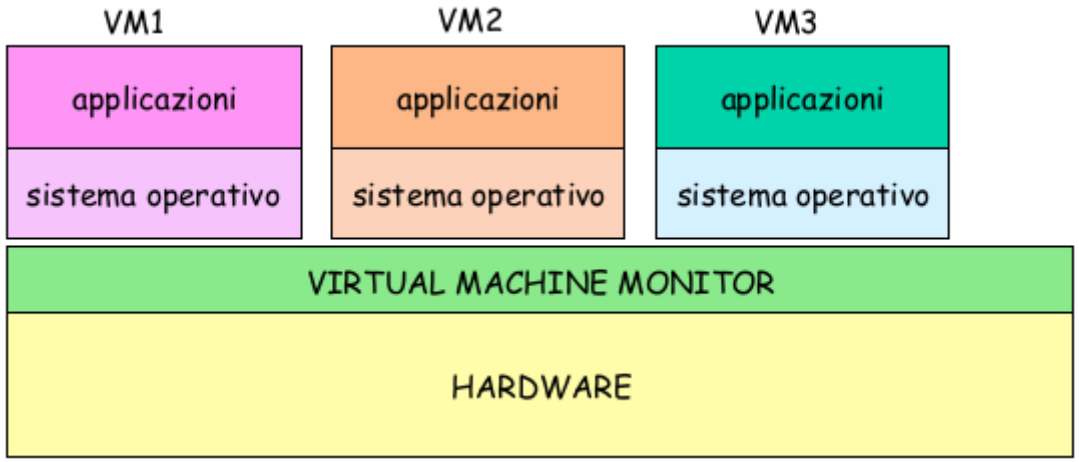


Esempi di virtualizzazione

- Il SO effettua una virtualizzazione delle risorse a livello di processo
  - Ogni processo interagisce con l'hardware solo con la mediazione del SO
  - Il SO mostra al processo ciò che preferisce
    - lo illude di avere a disposizione la CPU
    - gli mostra dispositivi virtuali attraverso i device driver invece di quelli fisici
    - gli mostra uno spazio di indirizzi di memoria indipendenti dallo spazio fisico effettivamente a disposizione
- Alcuni linguaggi virtualizzano a loro volta il SO
  - i programmi vengono eseguiti in un runtime environment
  - il runtime esegue le istruzioni del programma trasformandole in istruzioni specifiche per l'SO/HW su cui è installato
    - non è necessario riscrivere migliaia di programmi per portarli su architetture diverse
    - basta che sia stato sviluppato il runtime per ogni SO/HW di interesse

Virtualizzazione di Sistema

- Una singola piattaforma hardware viene condivisa da più elaboratori virtuali (**macchine virtuali o VM**) ognuno gestito da un proprio sistema operativo.
- Il disaccoppiamento è realizzato da un componente chiamato Virtual Machine Monitor (**VMM, o hypervisor**) il cui compito è consentire la condivisione da parte di più macchine virtuali di una singola piattaforma hardware.
- Ogni VM contiene un proprio sistema operativo, che definisce un ambiente di esecuzione distinto e isolato della altre macchine virtuali, che consente l'esecuzione di applicazioni all'interno di esso

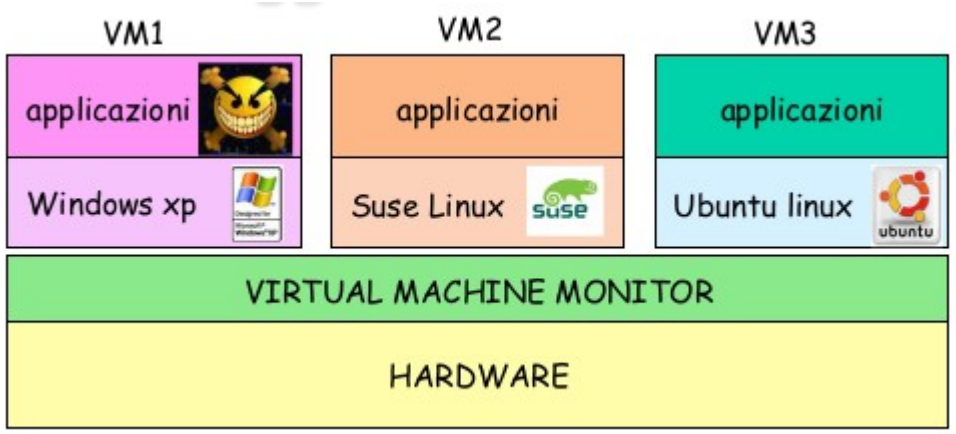


Evoluzione

- La virtualizzazione non è un concetto nuovo
  - Anni 60: IBM (CP/CMS, VM/370)
    - mainframe costosissimi → VMM eccellente modo di condividerli
  - Anni 70: SO multitasking
    - multiutenza → riduce in molti casi la necessità di avere diverse VM
  - Anni 80-90: PC e crollo prezzi HW
    - “one service, one server” dove è necessaria separazione
    - costa relativamente poco ma sottoutilizza le risorse e complica la gestione
  - Anni 00: razionalizzazione
    - le potenze di calcolo sono tali che diventa possibile avere tante VM su un singolo elaboratore grande
    - enorme convenienza su tanti elaboratori piccoli (utilizzo, energia, gestione)
  - Anni 10: cloud
    - dettagliato in seguito

Vantaggi della virtualizzazione

- Uso di più S.O. sulla stessa macchina fisica: più ambienti di esecuzione (eterogenei) per lo stesso utente:
  - Legacy systems
  - Possibilità di esecuzione di applicazioni concepite per un particolare SO
- Isolamento degli ambienti di esecuzione: ogni macchina virtuale definisce un ambiente di esecuzione separato (sandbox) da quelli delle altre:
  - possibilità di effettuare testing di applicazioni preservando l'integrità degli altri ambienti e del VMM
  - sicurezza: eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale

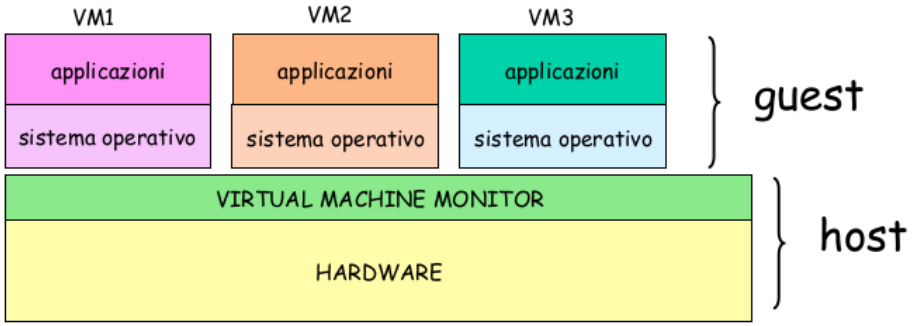


Vantaggi della virtualizzazione

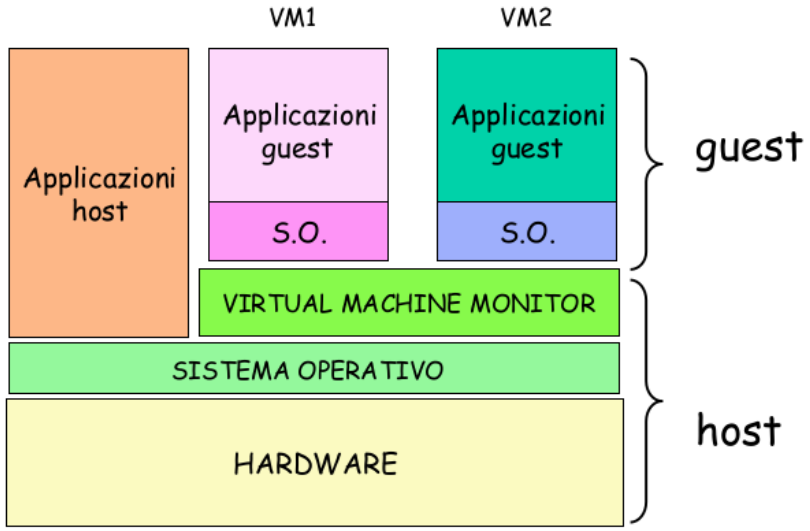
- Consolidamento HW: possibilità di concentrare più macchine (ad es. server) su un'unica architettura HW per un utilizzo efficiente dell'hardware (es. server farm):
  - Abbattimento costi hw
  - Abbattimento costi amministrazione
- Gestione facilitata delle macchine: e` possibile effettuare in modo semplice:
  - la creazione di macchine virtuali (virtual appliances)
  - l'amministrazione di macchine virtuali (reboot, ricompilazione kernel, etc.)
  - migrazione a caldo di macchine virtuali tra macchine fisiche:
    - possibilita` di manutenzione hw senza interrompere i servizi forniti dalle macchine virtuali
    - disaster recovery
    - workload balancing: alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte in modo “autonomico” a situazioni di sbilanciamento
- In ambito didattico: invece di assegnare ad ogni studente un account su una macchina fisica, si assegna una macchina virtuale, per esercitarsi:
  - senza limitazioni nelle tecniche di amministrazione e configurazione del sistema;
  - potendo installare e testarenuovi sistemi operativi, anche prototipali, senza il rischio di compromettere la funzionalità del sistema.
  - su applicazioni potenzialmente pericolose senza il rischio di interferire con altri utenti/macchine;
  - possibilita` di trasferire le proprie macchine virtuali in supporti mobili (es: penne USB, per continuare le esercitazioni sul computer di casa).

Modalità di virtualizzazione

- Non entriamo nei dettagli. Le più usate comunemente:



VMM di sistema
(più efficiente per server)



VMM ospitato
(comune su workstation)

Prefazione
-
Filosofia del software libero e aperto e origini del sistema GNU/Linux

Elementi

- GNU (GNU's Not Unix) è un progetto di sviluppo software
  - finalizzato a riprodurre l'ecosistema Unix
  - caposaldo del modello di licenza open source
- Linux è un sistema operativo
  - Unix-like
  - multitasking
  - multiuser
  - 32 e 64 bit
  - funziona su un'ampia varietà di piattaforme hardware
  - è distribuito sotto una licenza open source

Licenze

- Troppe per citarle tutte...
  http://www.gnu.org/philosophy/license-list.html
  - La proliferazione delle licenze è uno dei problemi del software libero, perchè complica il meccanismo fondamentale della creazione di nuovi prodotti per integrazione di quelli esistenti
- In sintesi estrema:
  - **copyleft**: attestano le libertà fondamentali ed impongono che i prodotti derivati siano distribuiti sotto la stessa licenza
  - **free**: attestano le libertà fondamentali
  - **public domain**: non impongono nessuna licenza al prodotto

Le proprietà delle OSL

1) **Free redistribution**
   The application can be redistributed, either on its own or as a part of a bundle of other applications. No royalty or other fees are required to do this.

2) **Source Code**
   The Source Code for the project must be easily available, if it is not downloaded with the compiled code (human readable code compiled into code the machine can read) clear instructions must be given as to where and how the source code can be obtained.

3) **Derived Works**
   The license must allow applications to be modified and distributed again with the same license as the original application.

4) **Integrity of the Author's Source Code**
   If the author wishes to keep their Source Code as is, they may stipulate that modified Source Code cannot be distributed, only if they then allow files (patches) to be distributed with the application that will modify it at runtime.

5) **No Discrimination against Persons or Groups**
   The author may not discriminate against any person or group; the product must be made available to all people who want to use it.

Le proprietà delle OSL

6) **No Discrimination against fields of Endeavour**
The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7) **Distribution of License**
The same license applies to all the people who the application is redistributed to.

8) **License must not be specific to a Product**
If an application is taken from a bundle of applications that was released as Open Content, then that application has the same license as the original bundle of applications.

9) **License must not restrict other Software**
The License the application is released under cannot specify that it can only be distributed or used in conjunction with Open Source software.

10) **License must be Technology Neutral**
For example: the license cannot specify that to use the software you must download it from a web page, or from a CD-ROM. The license must allow modification of the application so that it can be used in all environments.

GPLv3

- Esaminiamo alcuni aspetti fondamentali della licenza sotto cui è distribuita la gran parte di un sistema GNU/Linux:

  GNU General Public License version 3
  - http://www.gnu.org/licenses/gpl-faq.html

- Come interagiscono i termini di questa licenza nel processo di sviluppo di una software house?

GPLv3

- Binari/sorgenti
  - Software GPL distribuito in forma binaria sotto qualunque forma (ad esempio in un dispositivo hardware) **deve essere accompagnato dal codice sorgente**, o almeno dalle istruzioni su come ottenerlo
- Estensioni ed aggregati
  - ogni estensione di un programma GPL deve essere rilasciata sotto la stessa licenza (attenzione alle librerie!)
  - un aggregato è un set di componenti chiaramente separabili, ad esempio un'applicazione che gira su un kernel; un aggregato può contenere parti proprietarie e parti GPL, purchè sia possibile per chi lo riceve far valere su queste ultime i diritti sanciti dalla licenza

GPLv3

- Strumenti e prodotti
  - Software GPL (es. editor, compilatori, ...) può essere usato per sviluppare software proprietario
  - Attenzione ai casi in cui **nell'ouput è incluso parte dello strumento** (es. script+interprete, come nel caso di bison)
- Librerie
  - Un programma che usa librerie ”pure GPL” **deve essere rilasciato sotto una licenza GPL-compatibile**.
  - La stragrande maggioranza delle librerie GNU, però, sono rilasciate
    - o sotto la GPL + una specifica eccezione che consente il link con qualunque applicazione, anche non free
    - **o sotto la GNU Lesser GPL**

LGPL

- La Gnu Lesser GPL (LGPL, GPL attenuata) è stata studiata per consentire la creazione di software proprietario basato su librerie free
  - secondo la GPL infatti, poichè effettivamente all'atto dell'esecuzione il codice comprende la libreria, il totale sarebbe un'estensione e non un aggregato
- la LGPL impone di rispettare alcuni vincoli, essenzialmente a garanzia che le librerie usate non vengano ”chiuse” per effetto della distribuzione con l'applicazione proprietaria
  - http://www.gnu.org/licenses/lgpl.html

1991

- All'inizio degli anni '90, quindi sono disponibili PC economici e realmente potenti per le necessità dell'epoca (avviene il salto generazionale da 16 a 32 bit con l'Intel i386)
- ma per queste piattaforme sono popolari solo S.O.
  - completamente chiusi e davvero limitati
    - (March 1993) MS-DOS 6.0
    - (December 1993) Windows for Workgroups 3.11
  - completamente chiusi ed embrionalmente evoluti
    - (October 1993) Windows NT 3.1 Workstation & Server
  - simil-Unix ma limitati, gratis ma non modificabili
    - minix

Linus Torvalds

- Uno studente finlandese di nome Linus Torvalds, stanco delle limitazioni di Minix e desideroso di studiare le funzionalità dei nuovi processori a 32bit, si imbarca in un progetto che si trasforma rapidamente in un kernel ispirato a Unix.
- L'annuncio è dato il 25 agosto 1991:

"Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(."

Linux

- **La disponibilità di tutti i tool GNU, principalmente il compilatore GCC, rende l'opera di Torvalds enormemente più semplice**
- **Inizialmente l'autorità indiscussa dell'epoca, Andrew S. Tanenbaum, docente universitario ed autore di Minix, scartò Linux come malamente progettato**
  - il design sembrava troppo legato al processore
  - il modello di sviluppo aperto alla comunità fu da AST tacciato di ingestibilità
  - all'epoca sembrava tramontata l'era dei kernel monolitici, Minix e Hurd erano microkernel

Linux - i primi sviluppi

- Nel giro di un anno e mezzo, Linux riceve sufficiente attenzione da giungere ad una versione (0.99) di eccellente qualità
- Viene rilasciata nel dicembre '92 **con licenza GPL**, e questa decisione decreta il suo successo
  - si possono legalmente **integrare Linux e gli strumenti GNU**
  - l'uno ha tutto ciò che manca all'altro
- dopo qualche controversia, viene quindi accettato globalmente che la denominazione corretta del sistema risultante sia GNU/Linux
- Nel 1993 conta più di 100 sviluppatori attivi
- Nel 1994 Torvalds lo considera abbastanza stabile e completo da meritare la versione 1.0

Le prime versioni del kernel

0.99 → **1.0** ('94): first stable release
(schema di numerazione A.B.C con B pari = stabile, B dispari = sviluppo per il successivo)

1.0 → 1.2 ('95): porting su DEC, Sparc e MIPS

1.2 → **2.0** ('96): modularizzazione, supporto SMP

2.0 → 2.2 ('99): forte miglioramento del supporto SMP, porting su m68k e PowerPC, nuovi filesystem

2.2 → 2.4 ('01): supporto a hw "dinamico" (PnP, USB, PCcard), filesystem complessi (RAID, LVM)

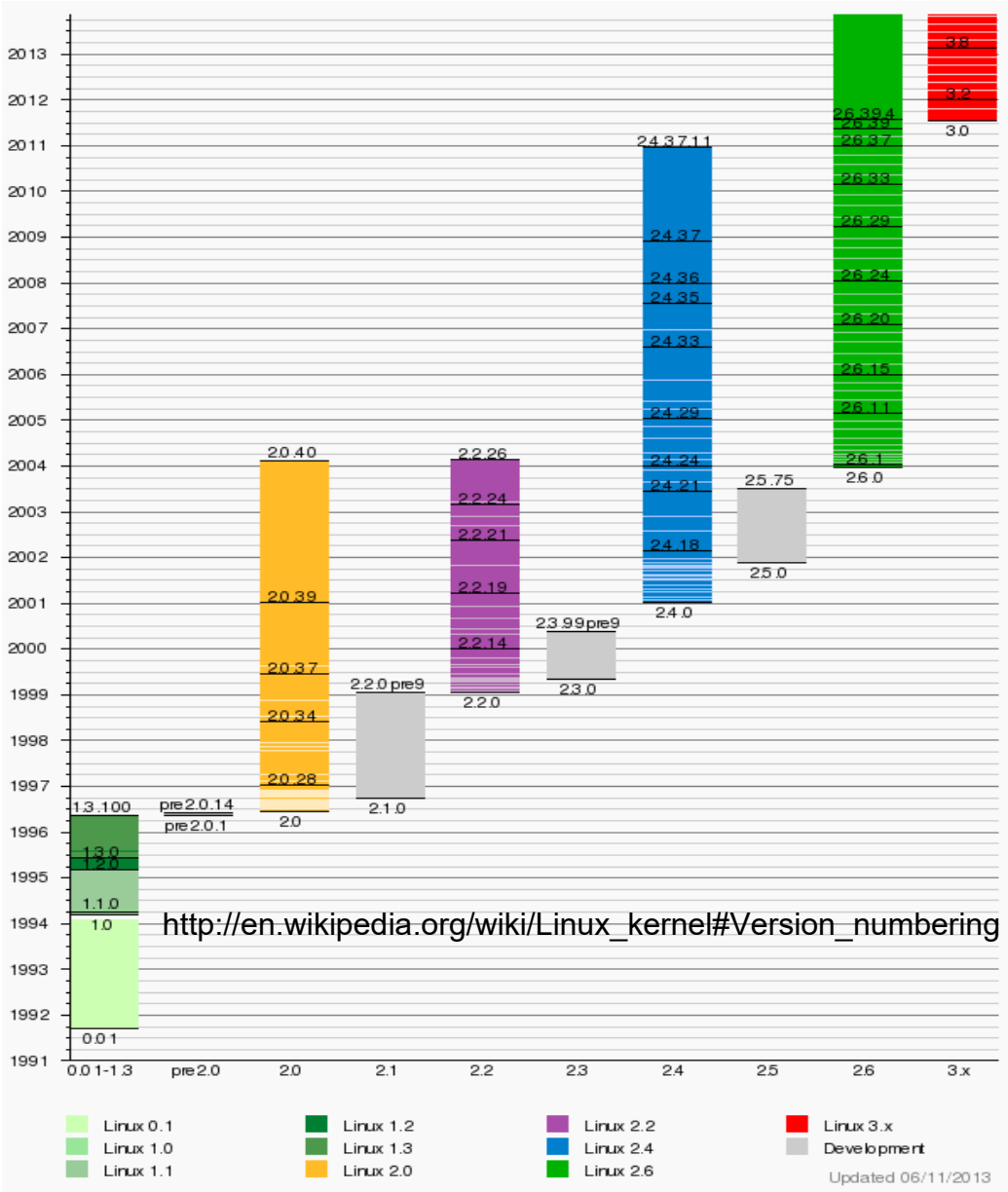
2.4 → 2.6 ('03): innumerevoli aggiunte (passaggio da 16 a 32 bit di uid e pid, supporto proc. 64bit, kernel-based vm, vari filesystem, SELinux, …)

(schema di numerazione time-based A.B.C.D – D=patch release)

2.6 → **3.0** ('11): terzo decennio!

(ripristino di A.B.C dove "3" sostituisce "2.6")

4.0 → oggi: grazie all'efficacia di git nel tracciamento delle modifiche, diventa superfluo distinguere stabile/sviluppo; i numeri vengono incrementati secondo il gusto di Linus :-)



www.kernel.org (maggio 2023)

The Linux Kernel Archives

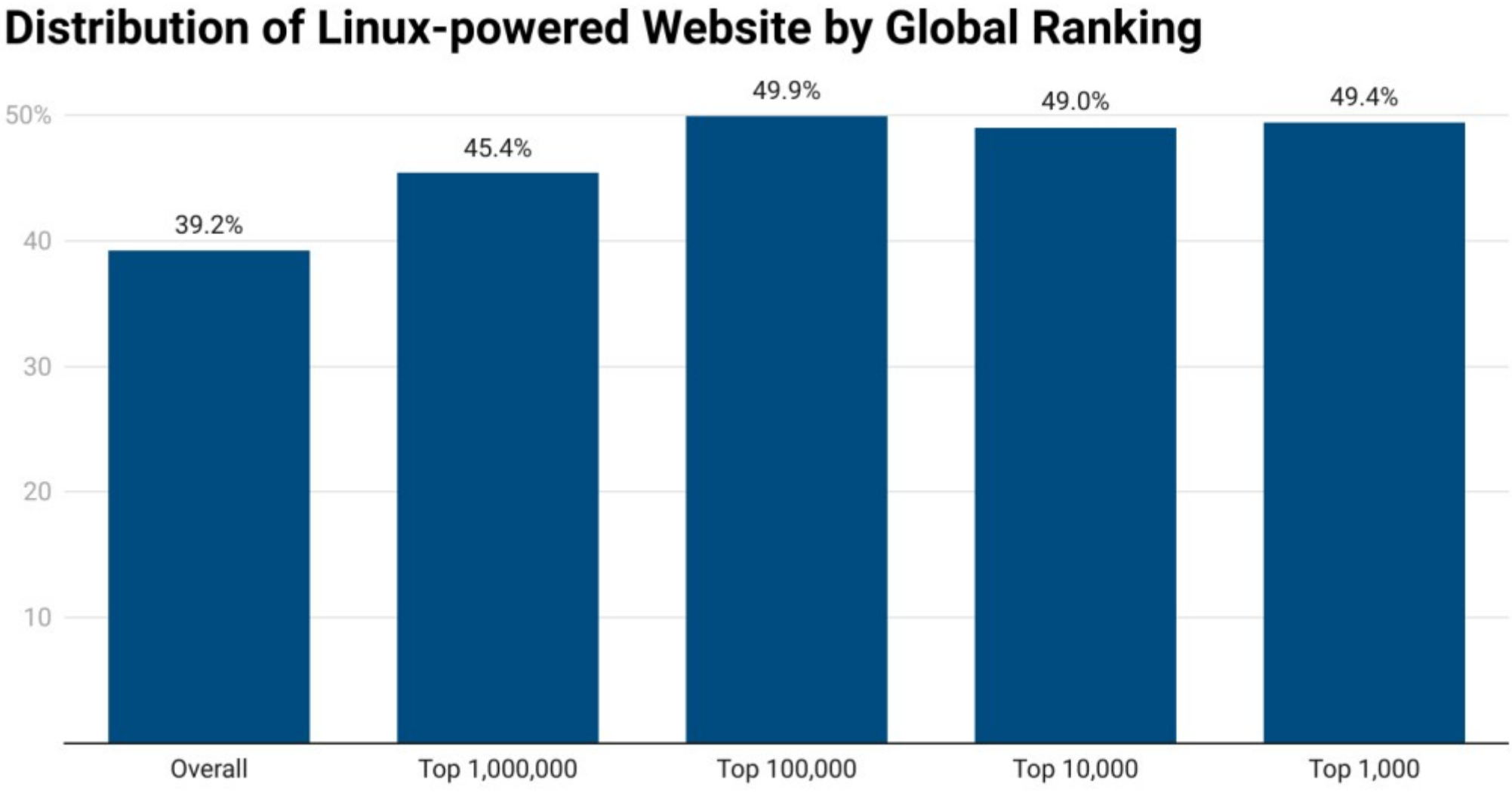
About Contact us FAQ Releases Signatures Site news

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Release
6.3.2

mainline:	6.4-rc1	2023-05-07	[tarball]		[patch]		[view diff]	[browse]
stable:	6.3.2	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
stable:	6.2.15	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.1.28	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.15.111	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.10.179	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.4.242	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.19.282	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.14.314	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
linux-next:	next-20230512	2023-05-12						[browse]

Quanto è diffuso Linux su Internet



Quanto vale Linux?

- Nel 2006, uno studio finanziato dall'UE ha stimato il costo teorico per riscrivere da zero il kernel versione 2.6.8 in **882 milioni di €**

- Nel 2008 la Linux Foundation ha stimato [1] che per ricreare un intero sistema come Fedora 9 (204.5 millioni di linee di codice in 5547 pacchetti applicativi) sarebbero necessari circa
  60,000 anni-uomo,
  distribuiti nell'arco di **25 anni**
  con un budget di **10,8 miliardi di US$**

- [1] https://web.archive.org/web/20081025072913/http://www.linuxfoundation.org/publications/estimatinglinux.php

Distribuzioni

- GNU/Linux è un insieme composto di elementi sostanzialmente fissi, reperibili in forma sorgente:
  - kernel
  - librerie di sistema
  - utilità di base
  - software applicativo di uso comune
- Per portare su di un sistema questi componenti però serve
  - o un altro sistema funzionante su cui scaricare e compilare tutti i pezzi
  - o una **distribuzione**
    - raccoglie in un supporto fisico pratico tutti i componenti, già compilati per l'architettura su cui si vuole installare GNU/Linux
    - fornisce gli strumenti per l'avvio autonomo del sistema da installare
    - tramite un installer propone la corretta sequenza dei passi di predisposizione del sistema
    - fornisce strumenti per la gestione del software installato/installabile: il grande valore aggiunto è la **gestione delle dipendenze**

Criteri per la scelta: stabilità

- Stabilità vs. Aggiornamento
  - il processo di rilascio frequente e continuo del software nel mondo GNU/Linux ha come conseguenza inevitabile che le versioni più aggiornate possano essere meno stabili
  - vi sono distribuzioni che hanno come filosofia l'inclusione dei pacchetti più recenti (e quindi con funzionalità maggiori) anche a costo di una minor robustezza, ed altre che garantiscono l'inclusione solo di software ben collaudato
    - Alcune distribuzioni sono “versionate”: durante il ciclo di vita di una versione vengono forniti solo aggiornamenti correttivi, tutte le novità vengono testate e accumulate per la pubblicazione in una nuova versione (che va installata sovrascrivendo la precedente)
    - Altre sono “rolling”: ogni volta che c’è una novità viene testata e distribuita, quindi in ogni momento il sistema è alla versione più recente
- Supporto e durata
  - La disponibilità di supporto garantito è tipica delle distribuzioni commerciali, ma anche con le distribuzioni gratuite più diffuse, in virtù della dimensione della relativa comunità di utenti, è semplice risolvere eventuali problemi
  - Per installazioni di tipo server esistono varianti denominate LTS (Long Term Support): per 5/7 anni chi cura la distro garantisce che gli aggiornamenti non modifichino le API (tipicamente viene garantito solo il backporting dei security fix, non quello di tutti i bug fix)

Predisporre l'installazione

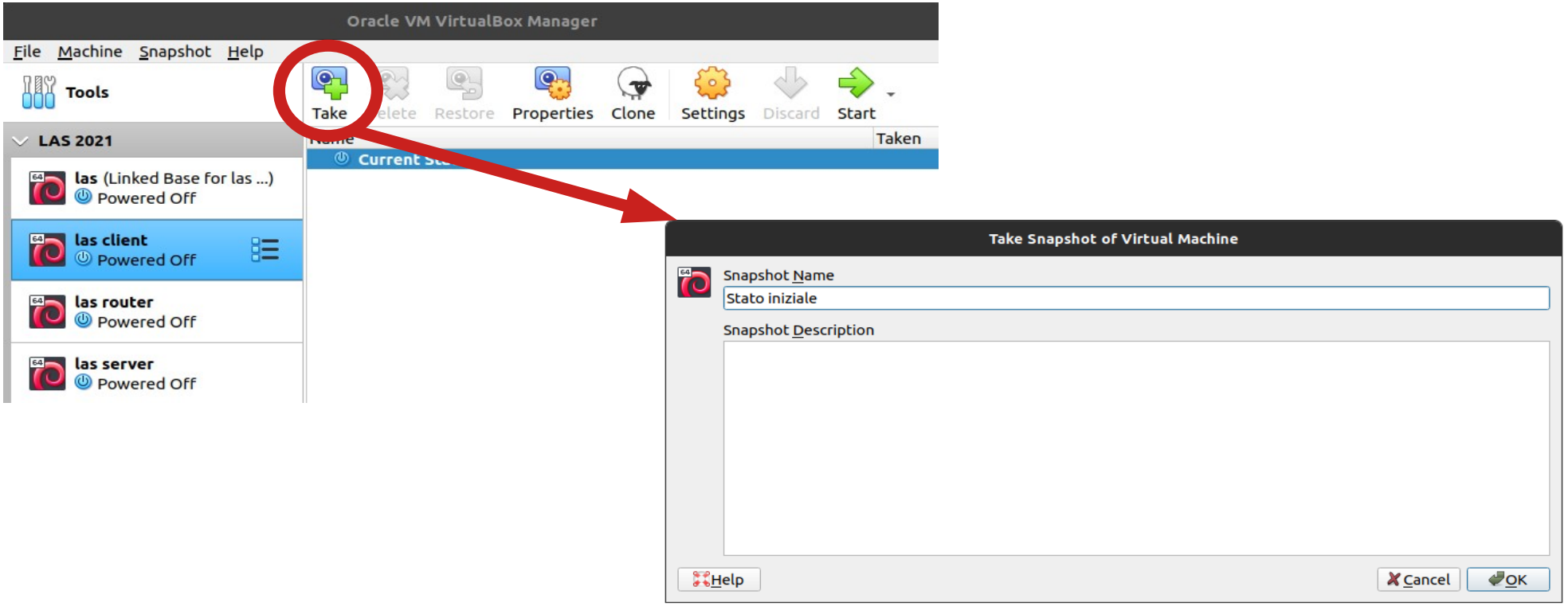
- Il modo più comune è scaricare da Internet un file
  - di tipo ISO: una copia esatta di quel che trovereste su di un media ottico (CD, DVD, BluRay)
    - si può ricreare il media ottico
    - più tipicamente si scrive a basso livello su un pendrive USB (non ci si copia semplicemente il file)
    - in entrambi i casi si ottiene un dispositivo da cui il computer può essere avviato, facendo partire una procedura per l'installazione
  - di tipo OVA: un archivio che contiene
    - un descrittore delle caratteristiche di una macchina virtuale
    - una copia esatta dei dischi di una macchina col sistema già installato
    - si importa nel virtualizzatore e si avvia pronto all'uso
  - di tipo VDI o VMDK o simile
    - una copia esatta dei dischi di una macchina col sistema già installato
    - la configurazione della VM va scelta manualmente
  - la prima opzione è utilizzabile sia su computer fisici che virtuali, le altre, evidentemente, solo per generare macchine virtuali
- Molti vendor offrono sistemi con Linux pre-installato

Laboratorio

- Esploriamo VirtualBox
- Installiamo su VB
  - una VM Kali pronta
    - sotto forma di disco VDI
    - è su classroom
    - viene da https://www.kali.org/get-kali/#kali-virtual-machines
  - una VM Ubuntu Desktop seguendo tutto il processo di installazione
    - versione 22.04.2
    - il file ISO è su classroom
    - viene da https://ubuntu.com/download/desktop
  - una VM Fedora Server seguendo tutto il processo di installazione
    - versione 38
    - il file ISO è su classroom
    - viene da https://fedoraproject.org/server/download/

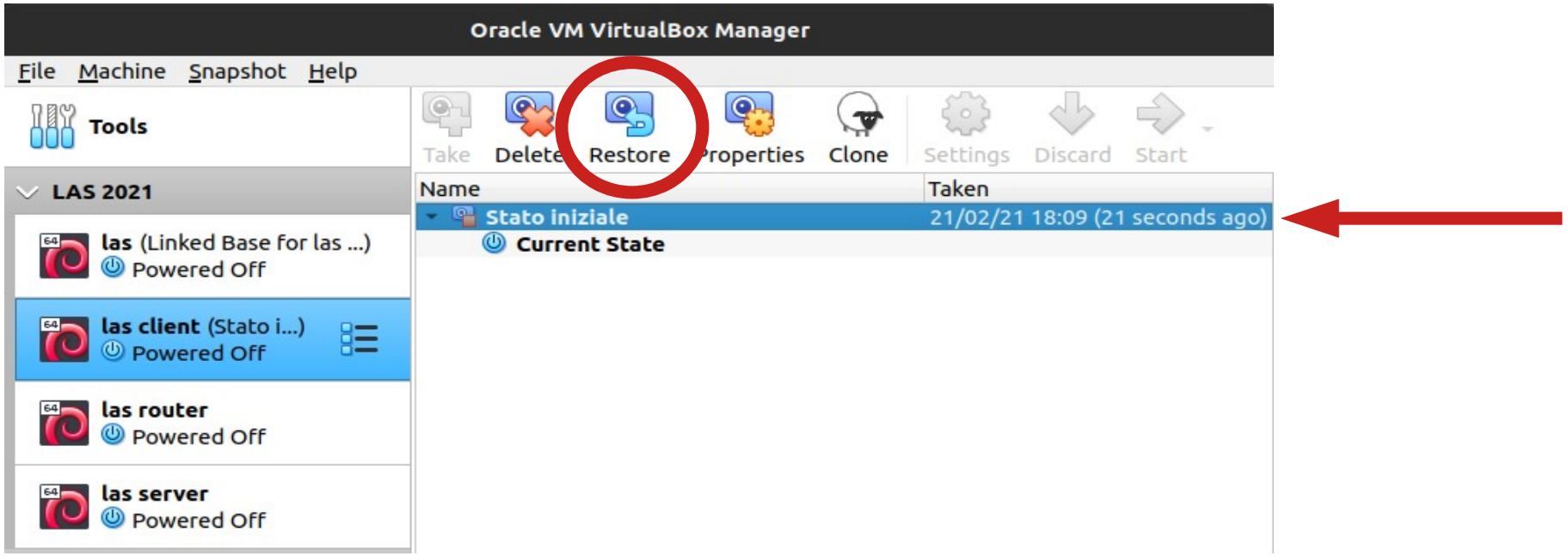
Snapshot (della VM Host)

- È possibile congelare lo stato di una VM in modo da poterlo ripristinare in seguito a modifiche con esito indesiderato
- Selezionare una VM
- Con il tasto “Take” scattare uno snapshot e chiamarlo, ad esempio, “Stato iniziale”



Recupero snapshot

- La macchina ora funzionerà usando il disco virtuale “Current State”, per ripristinarla a Stato Iniziale basta selezionarlo e cliccare sul tasto “Restore”



Accedere al sistema
e utilizzare la riga di comando

Login e interprete dei comandi

- Lo strumento più potente,flessibile e soprattutto più standard con cui amministrare un sistema Unix è l'interfaccia testuale a **riga di comando**.

- Componente essenziale è la **shell** o interprete dei comandi, che permette di svolgere compiti quali job control, mette a disposizione variabili e strutture di controllo per scrivere semplici programmi, e permette di invocare gli eseguibili installati nel sistema.
  - Possiamo definire shell qualsiasi interfaccia tra utente e OS

- Una delle shell più usate è **bash**, (Bourne-again shell) un'evoluzione della classica Bourne Shell di Unix.

- Una volta inserite le credenziali, il sistema presenta un *prompt* che indica la disponibilità dell'interprete ad accettare comandi

Chi sono, chi c’è

- Poiché è del tutto comune disporre di differenti identificativi utente con cui lavorare, è utile disporre di un comando per sapere quale e' l'identificativo con il quale si sta operando:
  - `whoami` indica il proprio username
  - `id` dà informazioni sull'identità e sul gruppo di appartenenza
  - `who` indica chi e' attualmente collegato alla macchina

Esecuzione dei comandi

- La shell indica il proprio stato di 'pronto' con una stringa di caratteri visualizzati nella parte iniziale della prima linea vuota. Questa stringa e' detta 'prompt'.
  - I caratteri digitati dall'utente dopo il prompt e terminati dal fine linea (ritorno a capo) costituiscono la command line.
  - Questa, tipicamente, contiene comandi e argomenti.
- I comandi digitati sulla command line possono essere:
  - keyword
  - built-in
  - comandi esterni
  - alias
  - funzioni

Tipi di comandi

- Una **keyword** è un comando che ha il compito di modificare l'esecuzione di altri comandi (es. misurare il tempo di esecuzione, innescare un ciclo)
- Un built-in e' un comando **direttamente eseguito dal codice interno della shell** che lo interpreta e lo 'converte' in azioni sul sistema operativo. Un esempio tipico è costituito dai comandi per la navigazione del filesystem.
- Un comando esterno e' un **file eseguibile che viene localizzato e messo in esecuzione dalla shell** (tipicamente in un processo figlio). La shell può attenderne o meno la conclusione prima di accettare nuovi comandi.
- Un **alias** è una stringa che viene sostituita da un'altra
- Una **funzione** è un'intera sequenza di comandi shell, con un nome, a cui possono essere passati parametri

Ricerca dei comandi tra i diversi tipi

- Come distinguere quale tipo di comando viene eseguito?
  - comando `type`

```
$ type -a echo
echo is a shell builtin
echo is /bin/echo
```

- Come alterare l'ordine di default?
  - backslash `\` davanti al comando: previene solo l'espansione degli alias
  - keyword `builtin`: previene l'espansione degli alias e l'uso di funzioni e invoca l'esecuzione del builtin specificato, se esiste
  - keyword `command`: utilizza un comando esterno anche se esiste una funzione con lo stesso nome
  - comando `unalias`: cancella un alias definito in precedenza

Alias

- La shell mette a disposizione alcuni sistemi per memorizzare linee di comando complesse in comandi più semplici da invocare. Uno di questi è l'**alias** che permette di attribuire un nome ad una command line:

```
alias miols='ls -l'
```

- Le associazioni definite con alias vanno perse al termine della sessione, vedremo come renderle persistenti.

Alias vs. builtin e comandi

- Una volta definiti, gli alias (o come descritto in seguito, le funzioni) questi hanno la priorità rispetto ai builtin ed i comandi omonimi.
- Per analizzare e pilotare la configurazione, sono utili i comandi `type`, `builtin`, `command` e `unalias`

```
$ alias echo='echo ~~~'
$ echo test
~~~ test
$ \echo test
test
$ builtin echo test
test
$ type echo
echo is aliased to `echo ~~~'
$ unalias echo
$ type -a echo
echo is a shell builtin
echo is /bin/echo
```

`\` previene solo l'espansione degli alias,
se `echo` fosse stato rifedinito come funzione
sarebbe stata usata quest'ultima

`builtin` invece fa override sia delle definizioni
di alias che di funzioni;
vale solo per riaccedere ai builtin:
se è stato ridefinito un comando esterno,
si usi `command` per invocare la versione originale

Ricerca dei comandi esterni

- Per lanciare un eseguibile lo si può individuare col percorso completo
  - assoluto

    `/usr/local/bin/top`
  - relativo

    `./mycommand`
- Tra le variabili d'ambiente comuni, la shell utilizza PATH per eseguire la ricerca dei comandi nel file system. La sua struttura è quella di un elenco di directory separate da **:**

  `PATH=/bin:/usr/bin:/sbin`
- Nel caso di più eseguibili omonimi in directory diverse?
  - lista ordinata, il sistema usa la prima istanza che trova
  - `which` Permette di sapere quale versione si sta usando:

    ```
    # which passwd
    /usr/bin/passwd
    ```
- Per consentire automaticamente l'esecuzione di programmi presenti nella directory corrente la variabile PATH deve contenere la directory **.**
  - Non è una buona norma, è facile lanciare per distrazione comandi errati
  - Meglio usare il percorso esplicito (vedi inizio slide)

Impostazioni di default

- Per ottenere automaticamente all'avvio della shell
  - assegnazione di valori a variabili come PATH
  - definizione di alias di comandi
  - … esecuzione di qualsiasi comando shell utile per inizializzare l'ambiente

  si ricorre ai file di configurazione di bash
- Si vedano (quasi all'inizio) la sezione INVOCATION e (quasi in fondo) la sezione FILES di *man bash*
- File globali

  `/etc/profile` `/etc/bash.bashrc`
- File personali (nella home)

  `.bash_profile` `.bash_login` `.profile` `.bashrc`

Documentazione dei comandi (e non solo)

- **man pages** – ogni applicazione installa "pagine di manuale" relative al suo utilizzo e configurazione.
  - Le man pages si leggono con il comando *man*
  - I builtin, non essendo programmi installati indipendentemente, non hanno man page.
    - Un sommario del loro funzionamento può essere visualizzato con help <builtin>
    - Inoltre, naturalmente, sono documentati nella man page `bash(1)`
- **info files** – a metà strada tra la man page e l'ipertesto, si leggono con il comando *info*, che invoca l'editor emacs appositamente esteso per gestire questi file
- **HOWTO** – documenti specifici per la risoluzione dei più svariati problemi pratici, sono raccolti in un pacchetto e vengono tutti installati in */usr/[share/]doc/HOWTO*
  Inoltre, sotto */usr/doc/HOWTO/translations/it* si possono trovare la maggior parte degli HOWTO tradotti in italiano.
- **on-line** – troppe fonti per citarle... un punto di partenza può essere http://tldp.org/ The Linux Documentation Project

Categorie di man pages

Una installazione standard di unix mette a disposizione innumerevoli **pagine di manuale** raggruppate in sezioni:

(1) User commands
(2) Chiamate al sistema operativo
(3) Funzioni di libreria, ed in particolare
(4) File speciali (/dev/*)
(5) Formati dei file, dei protocolli, e delle relative strutture C
(6) Giochi
(7) Varie: macro, header, filesystem, concetti generali
(8) Comandi di amministrazione riservati a *root*
(n) Comandi predefiniti del linguaggio Tcl/Tk

Accesso alle man pages

- L'accesso alle pagine si ottiene con il comando
- **man** <nome della pagina>

- Spesso il nome della pagina coincide con il comando o il nome del file di configurazione che essa documenta.

- Alcune opzioni utili sono qui riassunte:
  - *man* **–a** *<comando>* — cercherà in tutte le sezioni
  - *man* **<sez.>** *<comando>* — cercherà nella sezione specificata
  - *man* **–k** *<keyword>* — cercherà tutte le pagine attinenti alla parola chiave specificata

Per avere altre informazioni sul comando man è ovviamente sufficiente usare man man.

Gli argomenti

- Ogni comando, sia built-in che esterno, può accedere ai caratteri che seguono la propria invocazione sulla command line.
  - La shell inserisce in memoria l'ARGV prima di generare il processo
- I gruppi di caratteri, **separati da spazi**, rappresentano gli **argomenti**, cioè i dati su cui si vuole che il comando operi.
- Un argomento che inizia con il carattere '-' è chiamato solitamente *opzione*
  - normalmente non è un vero e proprio dato da elaborare
  - è un modo di specificare una variante al comportamento del comando
  - più opzioni possono essere solitamente raggruppate in un'unica stringa.

Esempi di argomenti e opzioni

ls /home	arg #1="/home" indica la directory di cui elencare il contenuto
ls -l /home	arg #1=opzione l indica che si desidera un listato in forma “lunga”
ls -l -a /home/alex	arg #1 e #2 = opzioni l (lunga) ed a (all)
ls -la /home/alex	arg #1 = opzioni concatenate l+a (identico a prima)

Digitare interattivamente la command line

- Iniziando a scrivere un comando (come primo elemento) o un nome di file (come argomento), e battendo TAB, bash completa automaticamente la stringa se non ci sono ambiguità, o suggerisce come completarla correttamente.

  Es.: al prompt digito `pass`<TAB> → compare `passwd` seguito da spazio ad indicare che passwd è l'unico completamento possibile

- Se ci sono ambiguità, una seconda pressione di <TAB> mostra i completamenti possibili.

  Es.
  - digito `ls /etc/pam`<TAB> → compare `/etc/pam.`
  - digito <TAB> → vengono elencati `pam.conf` e `pam.d/`
  - aggiungo "`c`" e digito <TAB> → compare `/etc/pam.conf`

Richiamare comandi passati

- `history` mostra l'elenco di tutti i comandi eseguiti in un terminale. Per richiamarli sulla command line, basta usare la freccia-su, appariranno (editabili) dal più recente al più vecchio
- La history è anche ricercabile interattivamente: al prompt basta digitare **CTRL-r** per far apparire un prompt (reverse-i-search); digitando una stringa, verrà mostrato il comando più recente che la contiene.
- Per navigare verso comandi impartiti precedentemente basta digitare nuovamente **CTRL-r**.
- Individuato il comando desiderato, si può lanciare direttamente premendo invio, o renderlo editabile sulla command line con freccia-destra o freccia-sinistra

Documentare le attività svolte sulla shell

- Il comando `script` permette di catturare in un file una sessione di lavoro al terminale, esattamente come compare a video, quindi sia i comandi impartiti che il loro risultato.

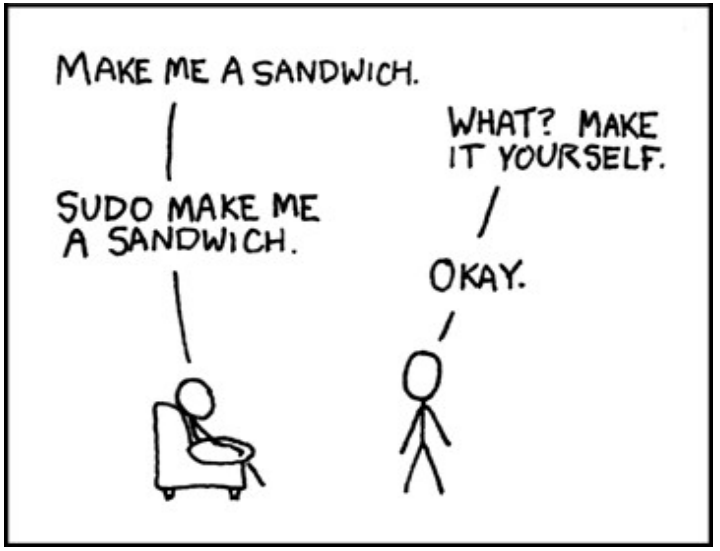
- Per terminare l'attività, digitare `exit` o premere **CTRL-d**

Accesso

- I sistemi come GNU/Linux e Microsoft Windows sono multiutente
  - ogni utente ha il permessi limitati di uso delle risorse
  - esiste un utente che ha permessi illimitati: ***root***
- Anche quando si lavora come sysadm, la quantità di operazioni da svolgere coi privilegi di root è molto limitata
  - è fortemente consigliabile utilizzare sempre l'account standard, che anche in caso di errori può provocare danni molto limitati
- Successivamente al login, è possibile cambiare identità
  - Il comando `su` (switch user) permette di "diventare" l'utente specificato come parametro, a patto di conoscerne la password
  - Senza parametri, equivale a `su root`
  - Consigliabile usare il parametro `-` per caricare l'ambiente dell'utente di destinazione (quindi il comando più comune sarà `su -`)

Accesso

- Oltre a `su`, esiste un modo migliore di utilizzare l'identità di root
  - `sudo` (super-user do) permette di eseguire un singolo comando come altri utenti (incluso root) senza conoscerne la password, a patto di essere stati preventivamente autorizzati



sudo

- Normalmente l'utente che avete creato in fase di installazione (o quello predefinito dell'immagine di una VM pronta all'uso) è già autorizzato a utilizzare sudo
  - Si lanciano comandi privilegiati con `sudo <comando>`
  - Si può diventare root con `sudo -i`
- Se si desidera configurare `sudo`
  - Diventare root (con `su -` nel caso che root abbia una password valida e l'utente standard non sia ancora autorizzato a usare sudo, con `sudo -i` altrimenti)
  - Lanciare il comando `visudo`

sudo

- Due possibilità:
  - Autorizzare l'utente “standard”, supponiamo si chiami `las`, a operare come qualsiasi utente, con la riga

    ```
    las ALL=(ALL:ALL) ALL
    ```

  - Autorizzare i membri del gruppo `sudo` a operare come qualsiasi utente, con la riga

    ```
    %sudo ALL=(ALL:ALL) ALL
    ```

    - Normalmente esiste già questa seconda riga nel file di configurazione di default
    - In questo caso verificare che l’utente (es. `las`) sia nel gruppo `sudo` (comando `id`)
    - Se non c’è, aggiungerlo (lanciare sempre come *root*)

      ```
      usermod -aG sudo las
      ```

    - Efficace solo dal successivo login

sudoers – qualche dettaglio in più

(da tenere buono per più avanti)

- Il file di configurazione di `sudo` è `/etc/sudoers`
- Leggete la man page… o anche no
  - forse l'unica che si sia mai vista che dice "don't despair…" dopo 200 righe di introduzione, prima di lanciarsi in 2600 ulteriori righe di delirio.
- Cose utili (grazie a https://toroid.org/sudoers-syntax)
  - formato delle direttive:

    ```
    User Host = (Runas) Tag: Command
    ```

  - da leggere come "User puo eseguire Command coi privilegi di Runas su Host"
    - Runas si può specificare come *utente*[:*gruppo*] ; se manca, è implicito root (e nessun altro)
    - Tag: è facoltativo e può essere un'opzione che modifica alcuni aspetti dell'esecuzione
    - Host è utile per condividere lo stesso file *sudoers*, quindi la stessa security policy, su tutte le macchine dell'organizzazione, pur differenziandola
- Qualche esempio:
  - `%sudo ALL=(ALL:ALL) ALL`

    i membri del gruppo sudo
    su qualsiasi host
    possono eseguire qualsiasi comando
    a nome di qualsiasi utente:gruppo
  - `marco ALL= NOPASSWD:/bin/backup`

    marco può eseguire (solo come root) */bin/backup* senza digitare la propria password
  - `ams ALL=(ALL) sha224:IkotndXGTmZtH5ZNFtRfIwkG0WuiuOs7GoZ+6g== /bin/ls`

    ams può eseguire (come utente a sua scelta) */bin/ls* solo se il programma ha un digest corrispondente a quello memorizzato

Shutdown

- Anche lo spegnimento di una macchina Linux (e unix in generale) richiede alcune operazioni.
- Si può abbandonare semplicemente la sessione di lavoro con exit (dalla shell di login) o con logout. Il sistema rimane comunue attivo e pronto ad altri accessi.
- Lo spegnimento richiede invece, di norma, l'accesso di root.
  - Sempre parlando di terminale. I sistemi grafici lo consentono all'utente.
  - Alcune installazioni permettono a chiunque si trovi in possesso della console (tastiera) di eseguire lo spegnimento (shutdown) premendo CTRL-ALT-CANC.
- root puo' invocare direttamente lo spegnimento, tipicamente con il comando:

  `shutdown [-h|-r] now`

  - `-h` indica la richiesta di arrestare il sistema (altrimenti: comando halt),
  - `-r` indica la richiesta di riavviare il sistema (altrimenti: comando reboot).
  - `now` indica quando eseguire l'operazione. E' possibile, ad esempio, lasciare agli utenti collegati alcuni minuti per terminare il proprio lavoro.