introduzione ai sistemi operativi

capitolo 1 del libro (VII ed.)

Argomenti

cos'è un sistema operativo (SO)?

storia dei SO

materiale tratto dal I capitolo del libro: Sistemi Operativi, di H. M. Deitel, P. Deitel, D. R. Choffnes (Pearson/Prentice Hall)

strutture dei sistemi di calcolo (I/O, memoria, protezioni HW)

system call

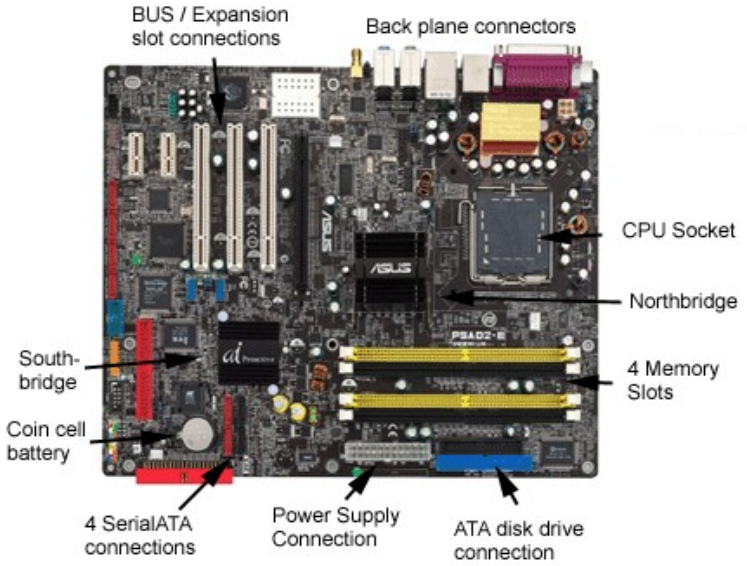
macchine virtuali

Cos'e` e a cosa serve un SO

Un SO è un insieme di programmi (software) che gestiscono gli elementi fisici (hardware) di un elaboratore

Cos'e` e a cosa serve un SO

Un SO è un insieme di programmi (software) che gestiscono gli elementi fisici (hardware) di un elaboratore



ASUS P5AD2-E Premium Motherboard
http://www.computerhope.com

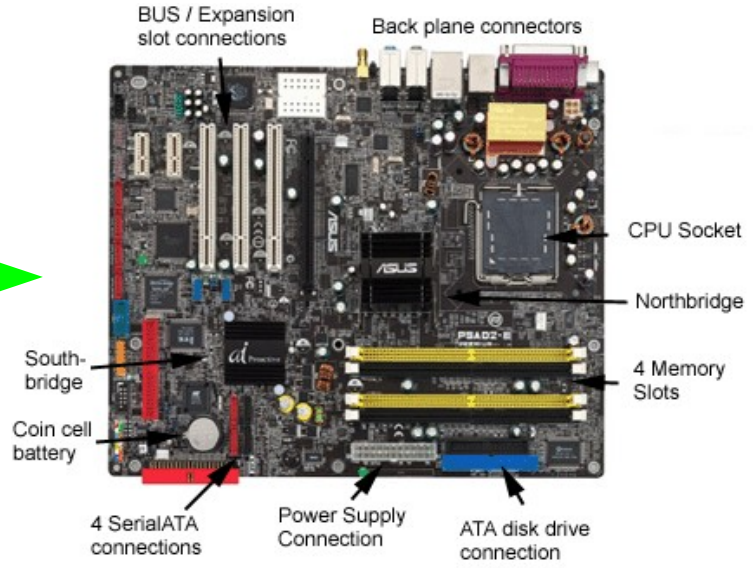
Hardware

Cos'e` e a cosa serve un SO

Un SO è un insieme di programmi (software) che gestiscono gli elementi fisici (hardware) di un elaboratore

User

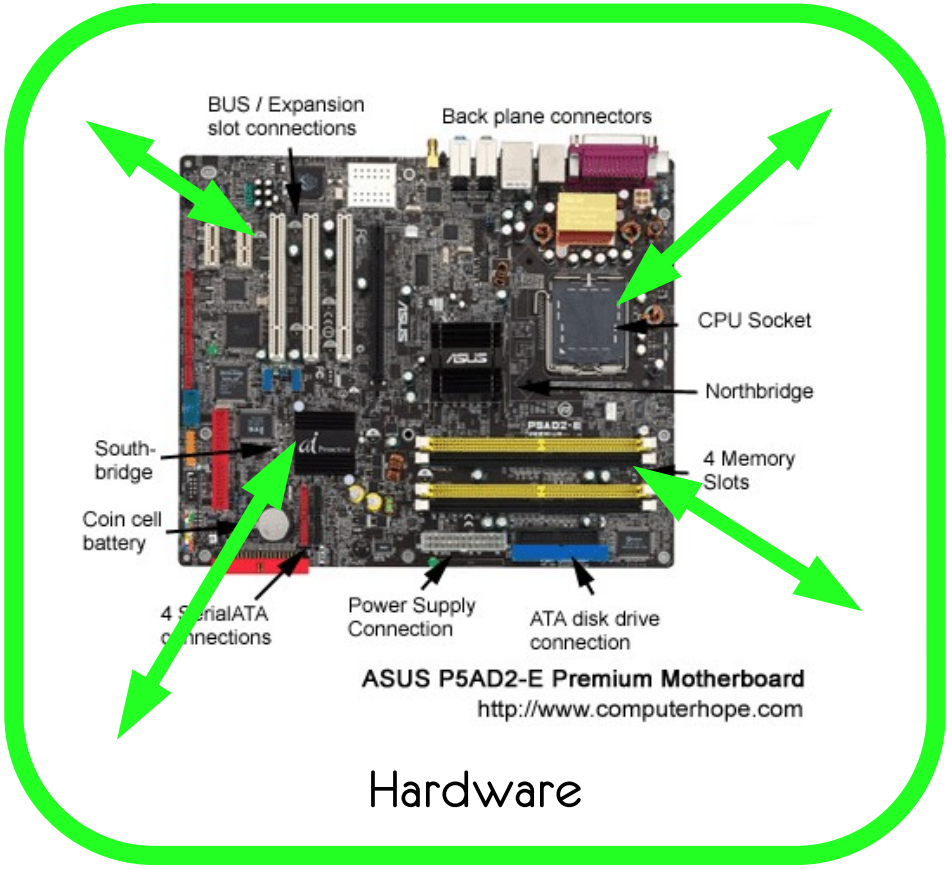
?



Hardware

Cos'e` e a cosa serve un SO

Un SO è un insieme di programmi (software) che gestiscono gli elementi fisici (hardware) di un elaboratore



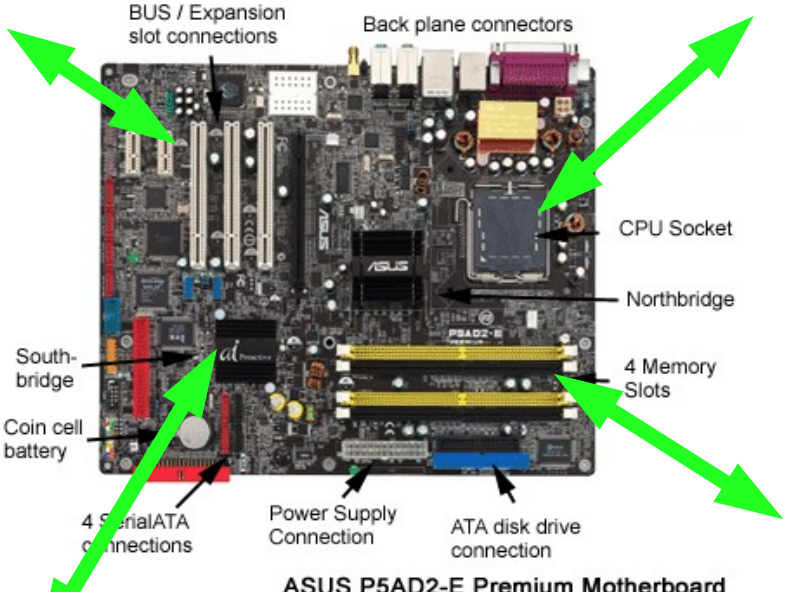
Software di gestione

Cos'e` e a cosa serve un SO

Un SO è un insieme di programmi (software) che gestiscono gli elementi fisici (hardware) di un elaboratore

User

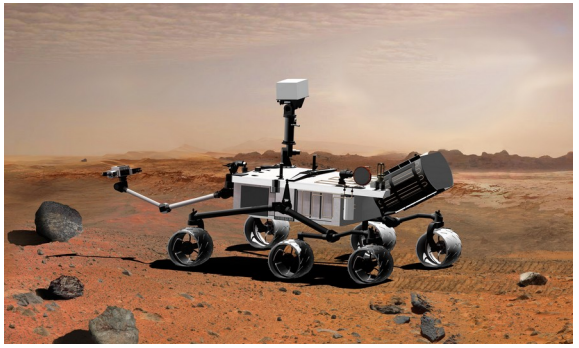
:-)



Hardware

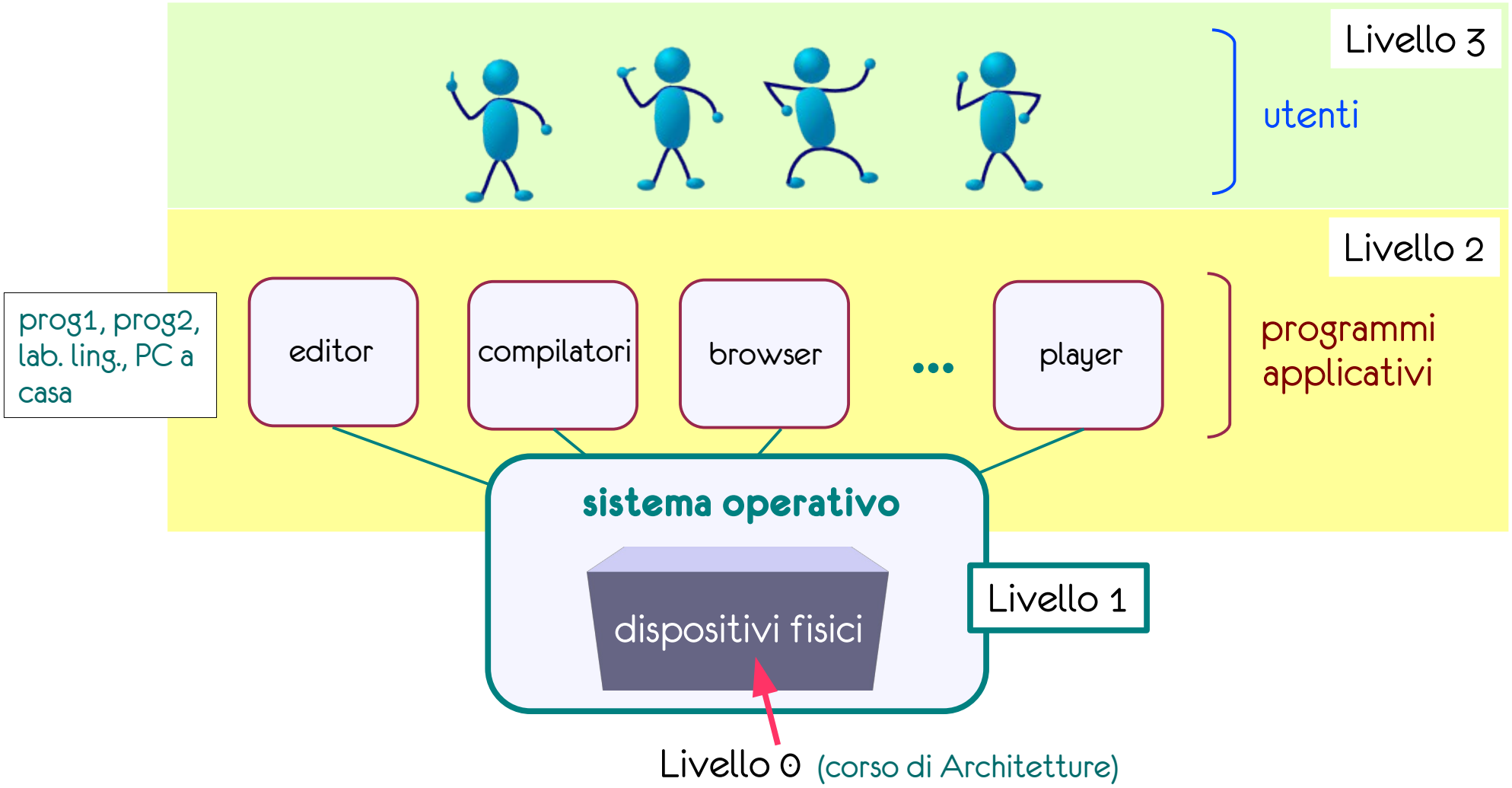
Software di gestione

Diversi tipi di sistema operativo



piu` precisamente

- Un SO è un insieme di programmi (software) che gestiscono gli elementi fisici (hardware) di un elaboratore



Livelli di un elaboratore

Livello 0

CPU, memoria, dispositivi di I/O, ...

Livello 1

dal punto di vista del sistema il SO è un assegnatore di risorse e un programma di controllo che gestisce l'esecuzione dei programmi applicativi

dal punto di vista dell'utente il SO è un ambiente di lavoro

Livello 2

tutti i programmi usati dagli utenti (es. browser, mailer, player di musica e video, editor, compilatori e interpreti, web server, filtri anti-spam, ...)

Assegnatore di Risorse?

Le **risorse** costituiscono il sistema e consentono la risoluzione di problemi

- **risorse hardware**

  CPU, memoria primaria e secondaria, dispositivi di I/O

- **risorse software**

  programmi applicativi e altre astrazioni software (es. semafori e code di messaggi) ma anche tempo di calcolo

Programma di controllo?

il SO gestisce molte cose:

- l'esecuzione dei programmi avviati dai diversi utenti,
- l'accesso a risorse quali le periferiche (es. stampanti),
- gli utenti accreditati su di una macchina,
- le risorse di sincronizzazione,
- ecc.

SO come gestore di
ogni tipo di risorsa

Programma di controllo?

ad esempio il SO si preoccupa di:

- esecuzione dei programmi:

  l'esecuzione è terminata? Attende il caricamento di dati? E` stata interrotta da un segnale?

- accesso a risorse:

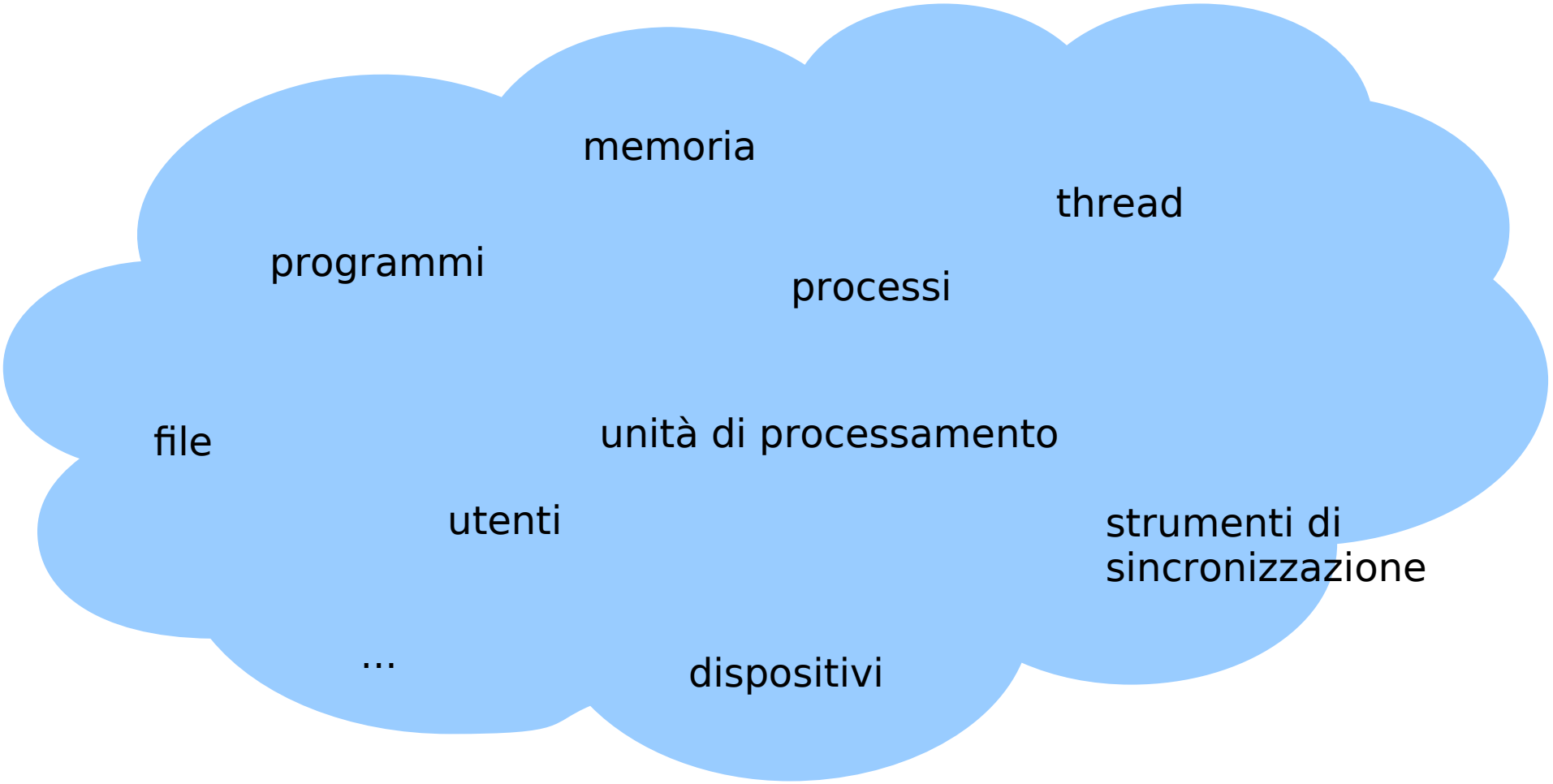
  la risorsa è sottoutilizzata? E`richiesta da più processi?

- utenti accreditati:

  l'utente X sta cercando di accedere a memoria allocata ad altri utenti? L'utente ha I privilegi per svolgere l'operaizone richiesta?

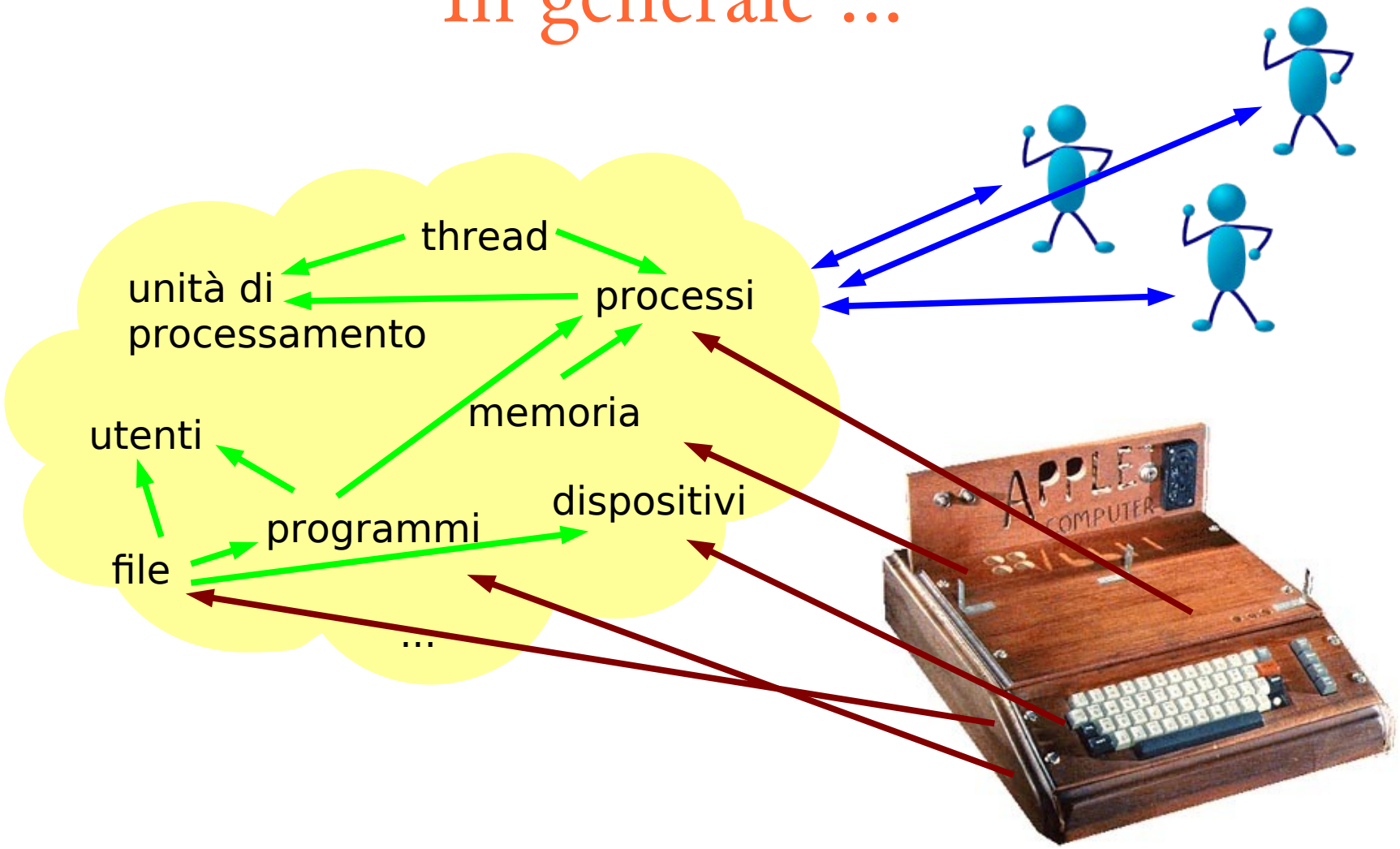
- le risorse di sincronizzazione,
- ecc.

In generale ...



in generale il SO definisce delle astrazioni software di tutti i tipi di oggetti che occorre rappresentare e gestire per far funzionare un computer (compresi gli utenti), implementa opportuni algoritmi di controllo e contiene interfacce sia verso gli utenti sia verso i dispositivi fisici

In generale ...



in generale il SO definisce delle astrazioni software di tutti i tipi di oggetti che occorre rappresentare e gestire per far funzionare un computer (compresi gli utenti), implementa opportuni algoritmi di controllo e contiene interfacce sia verso gli utenti sia verso i dispositivi fisici

un po' di storia ...

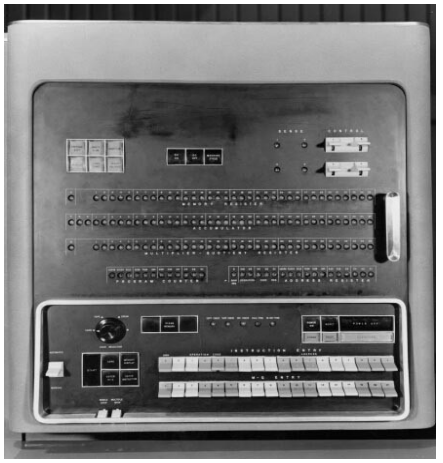
perche' per capire
le soluzioni occorre,
innanzi tutto,
capire i problemi ...

Evoluzione 1/2

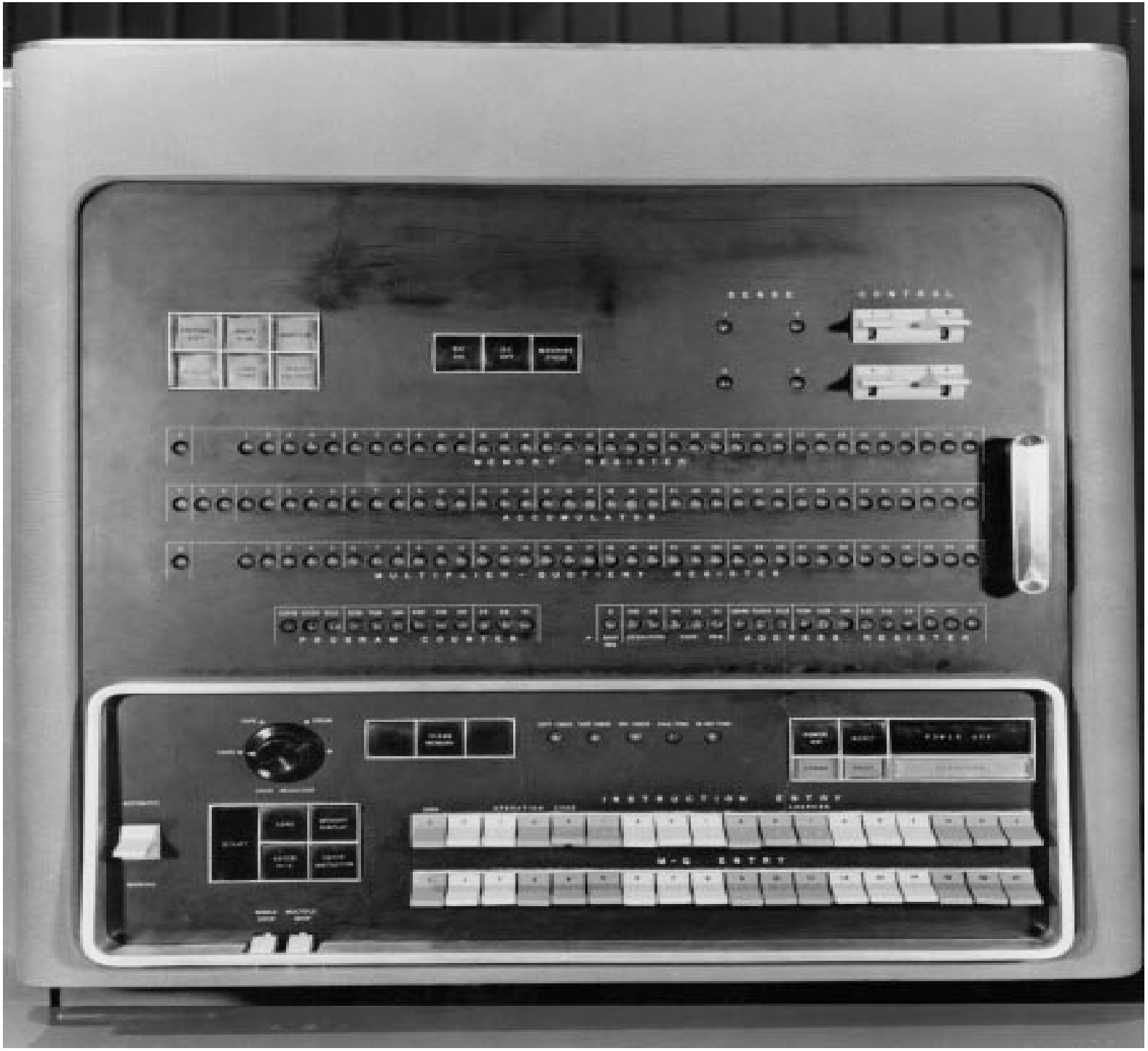
astrazioni software, algoritmi di controllo e interfacce si sono evoluti nel tempo

- anni '50: nasce il primo SO per IBM 701
  - single-stream batch-processing system
  - job
- anni '60:
  - multiprogrammazione
  - timesharing
  - processi
- anni '70:
  - sviluppo delle reti, nascita di TCP/IP
  - inizia la diffusione dei personal computer

IBM 701



IBM 701 control panel


SENSE
CONTROL
MEMORY REGISTER
ACCUMULATOR
MULTIPLIER - QUOTIENT REGISTER
PROGRAM COUNTER
ADDRESS REGISTER
INSTRUCTION ENTRY
M-Q ENTRY

Evoluzione 2/2

l'evoluzione continua

- anni '80:
  - sviluppo di workstation (macchine commerciali potenti come mainframe)
  - nascono le Graphical User Interface (GUI)
  - nascono applicativi tipo word processor e fogli di calcolo
- anni '90:
  - diffusione di internet e nascita di gopher (~'91)
  - nasce il world wide web (browser Mosaic ~'94)
  - si diffonde la tecnologia a oggetti
  - cresce il movimento open source
- anni '00:
  - web service, thread

Approfondimento

anni '50

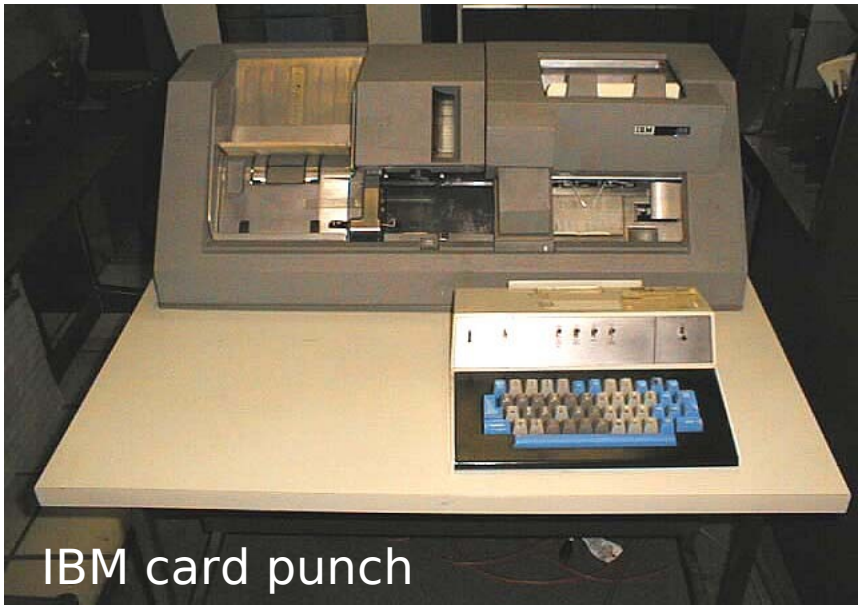
job: insieme di istruzioni di un programma dedicate a un certo compito computazionale (es. gestione di un archivio).

single-stream batch-processing system: i job venivano memorizzati in una sequenza su un nastro o un disco. Di qui venivano letti ed eseguiti in successione (in un solo flusso), senza interazione con l'utente. Lettura ed esecuzione sequenziale automatiche.

funzione del SO:
funzione di controllo, quando un job termina avvia il successivo

Sistema batch

fonte: http://www.catb.org/~esr/writings/taouu/html/ch02s01.html


IBM card punch

"Submitting a job to a batch machine involved, first, preparing a deck of punched cards describing a program and a dataset. Punching the program cards wasn't done on the computer itself, but on specialized typewriter-like machines that were notoriously balky, unforgiving, and prone to mechanical failure.
The software interface was similarly unforgiving, with very strict syntaxes meant to be parsed by the smallest possible compilers and interpreters.

Once the cards were punched, one would drop them in a job queue and wait.

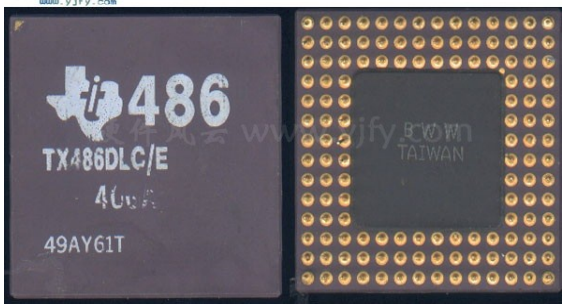
Eventually. operators would feed the deck to the computer, perhaps mounting magnetic tapes to supply a another dataset or helper software. The job would generate a printout, containing final results or (all too often) an abort notice with an attached error log. Successful runs might also write a result on magnetic tape or generate some data cards to be used in later computation.

The turnaround time for a single job often spanned entire days."

Approfondimento

```
fscanf(fp, “%s%d”, str, &i);
```

occorre del tempo affinché il dispositivo restituisca il dato di interesse



CPU

DISPOSITIVO DI MEMORIA

Approfondimento

anni '50

job: insieme di istruzioni di un programma dedicate a un certo compito computazionale (es. gestione di un archivio).

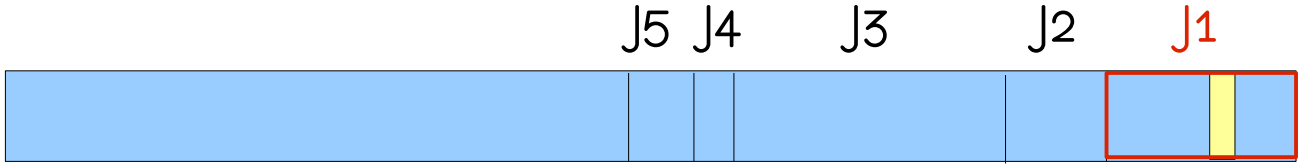
single-stream batch-processing system: i job venivano memorizzati in una sequenza su un nastro o un disco. Di qui venivano letti ed eseguiti in successione (in un solo flusso), senza interazione con l'utente. Lettura ed esecuzione sequenziale automatiche.

criterio di bontà del SO:
turn-around time = istante_restituzione_job – istante_sottomissione_job

possibilità di miglioramento del tempo di turn-around? Quando un job esegue dell'I/O non usa la CPU perché interagisce con strumenti (device) di I/O

funzione del SO:
funzione di controllo e di ottimizzazione dell'uso della risorsa CPU

I/O e CPU



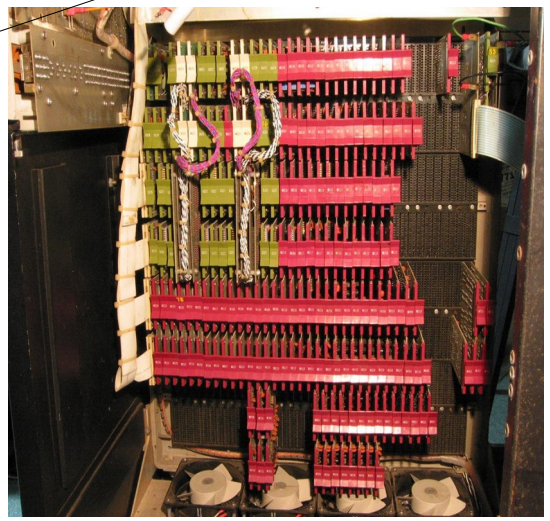
job attivo

nastro contenente il lotto
di job da eseguire

istruzione di lettura

IDEA:
non posso allocarla
a qualche altro job?

Occorrono strutture
ausiliarie (memoria,
rappresentazione e
meccanismi di gestione
dei job sospesi)



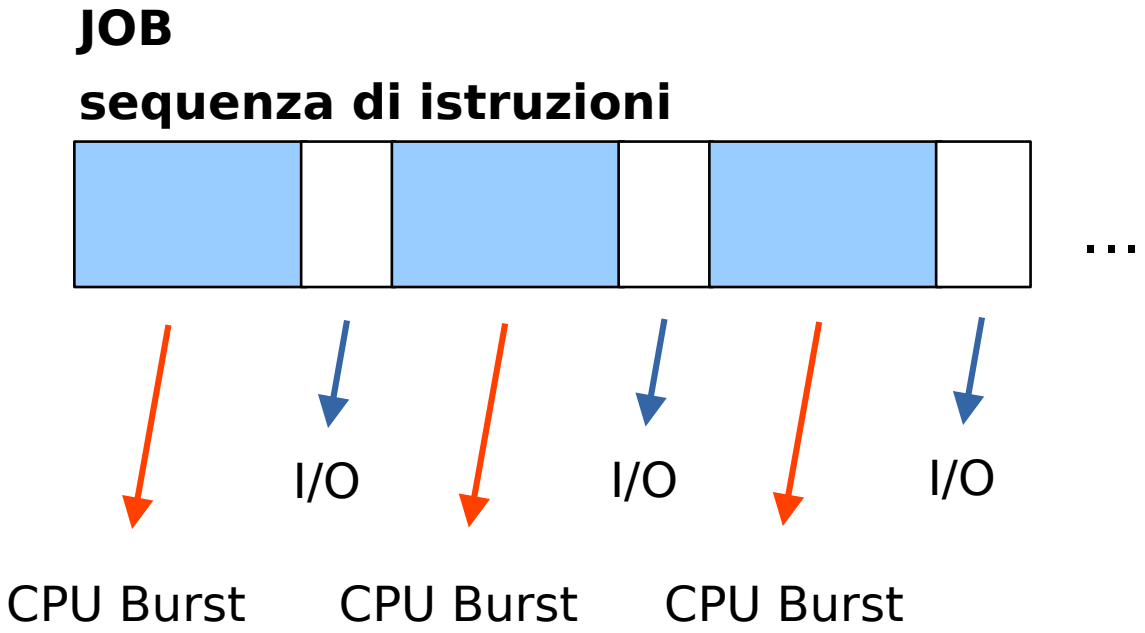
CPU, PDP8 (1965)

fintantoché la lettura
non è completa, la CPU
rimane inutilizzata!!

(1951)

la lettura
richiede tempo

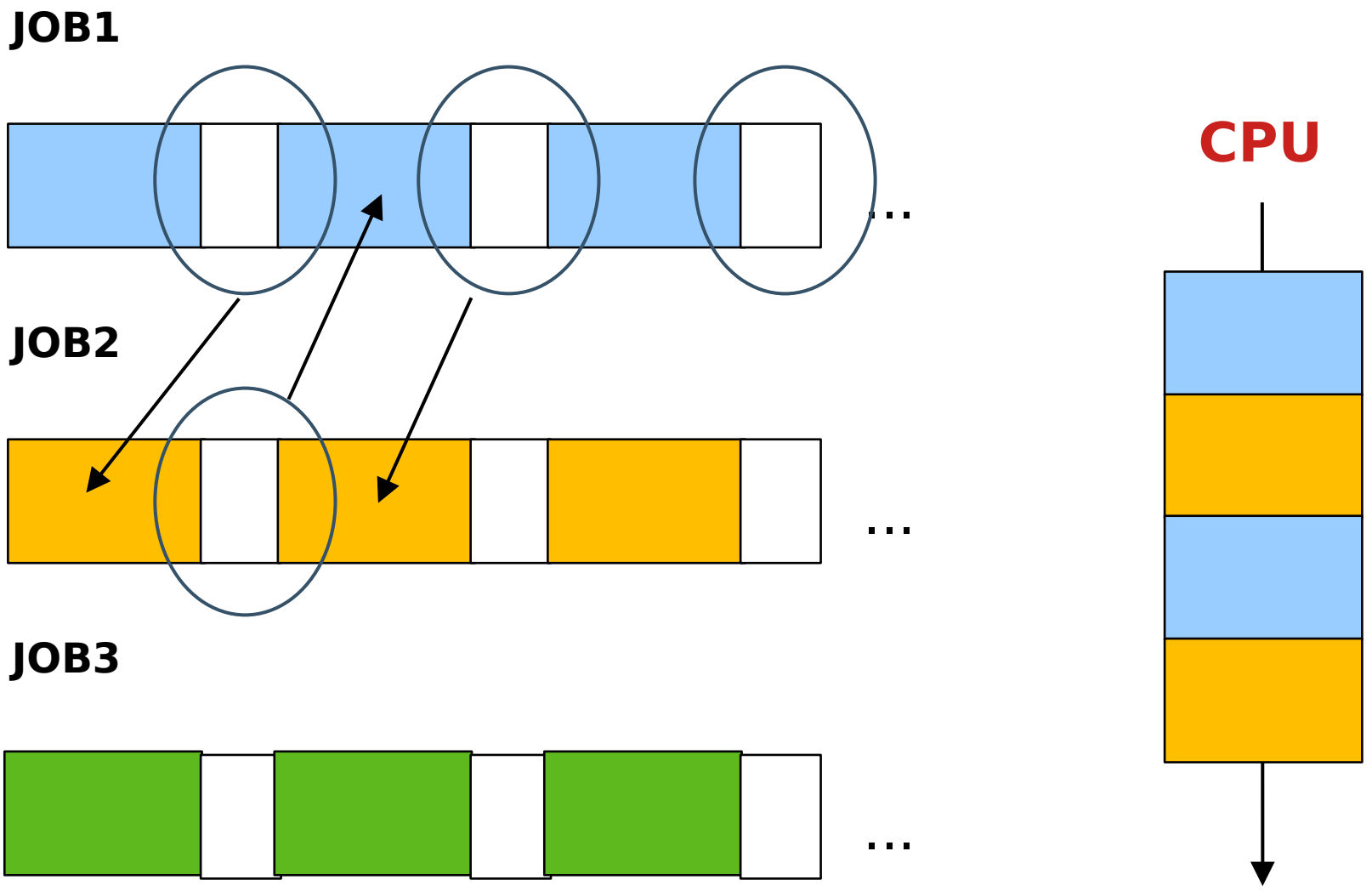
Interleaving (interfogliamento) delle istruzioni



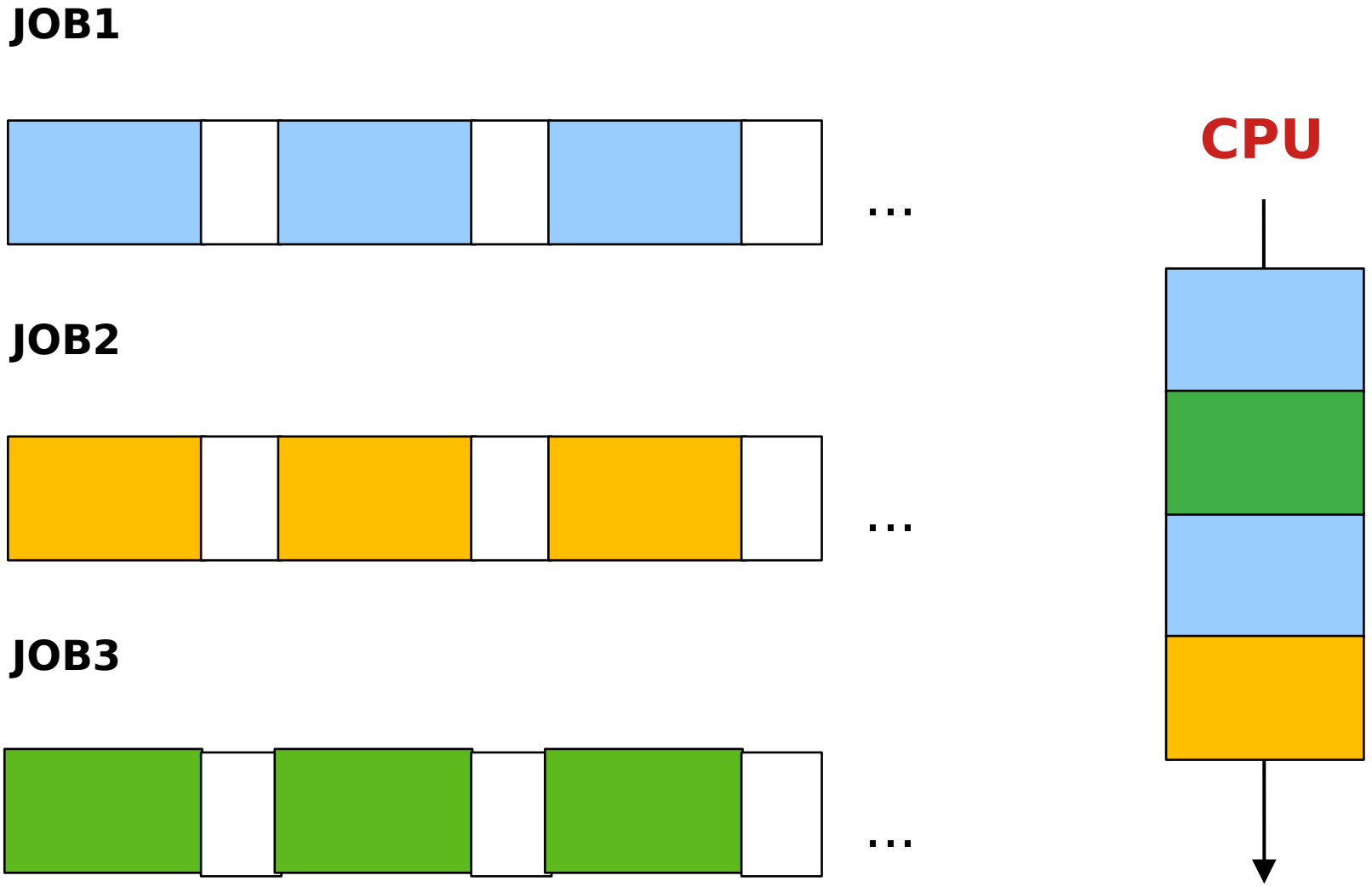
CPU Burst: sequenza di istruzioniche impegnano la CPU

I/O: operazione di input o di output

Interleaving (interfogliamento) delle istruzioni

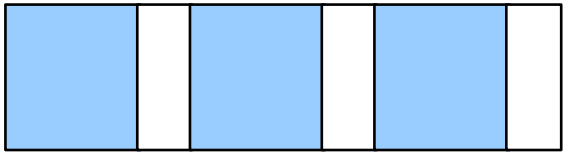


Parallelismo virtuale



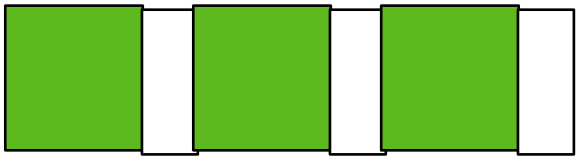
Parallelismo virtuale

JOB1 (in esecuzione)



JOB2 (pronto)

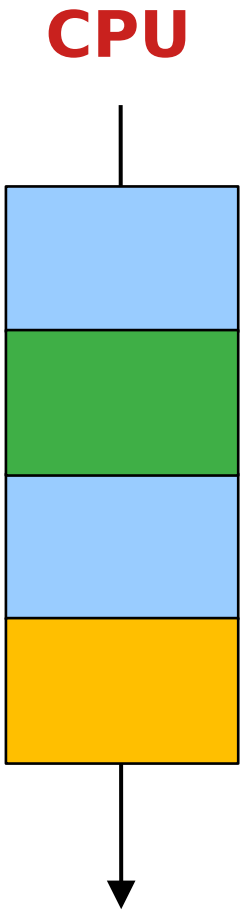
JOB3 (pronto)



COMMUTAZIONE DI CONTESTO

- Salvo lo stato del job in esecuzione
- Interrompo il job in esecuzione
- Carico lo stato di un job pronto
- Continuo esecuzione job caricato

Ecc.



Quesito

un semplice SO anni '50 è un programma del genere di quelli che avete imparato a sviluppare?

cosa deve fare?

Quesito

un semplice SO anni '50-'60 è un programma del genere di quelli che avete imparato a sviluppare?

cosa deve fare?

quando un job termina, caricane un altro;

quando il job in esecuzione richiede dell'input:

- salva lo stato dell'esecuzione
- rimetti il job in coda
- carica un altro job e mandalo in esecuzione

quando un dispositivo di input ha caricato in memoria i dati richiesti da un job sospeso ... ecc.

Migliore gestione di I/O e CPU

anni '60

nasce la multiprogrammazione: invece di riservare l'intera struttura di calcolo ad un solo job alla volta, si caricano in memoria principale diversi job tutti pronti per l'esecuzione. Quando un job richiede dell'I/O si cede il controllo della CPU ad un altro job del pool.

nasce il concetto di scheduling: scelta dell'ordine di esecuzione di elementi di interesse. Es. job scheduling: scelta dell'ordine di esecuzione di un pool di job. Può essere programmata. Si può decidere il criterio da adottare sulla base del tipo di job gestiti.

sistemi batch multiprogrammati: il SO
- controlla lo scheduling dei job
- garantisce la protezione dei job (no interferenze, no spionaggio)

Gli utenti interagiscono con i programmi

anni '60, es. IBM 704

sistemi operativi più avanzati furono sviluppati per consentire agli utenti di seguire in modo interattivo i job in esecuzione su di una certa macchina: sistemi in time-sharing

gli utenti interagivano con i programmi in esecuzione seduti ad appositi terminali, puri dispositivi di I/O senza capacità di calcolo, collegati con l'elaboratore

Sistema interattivo

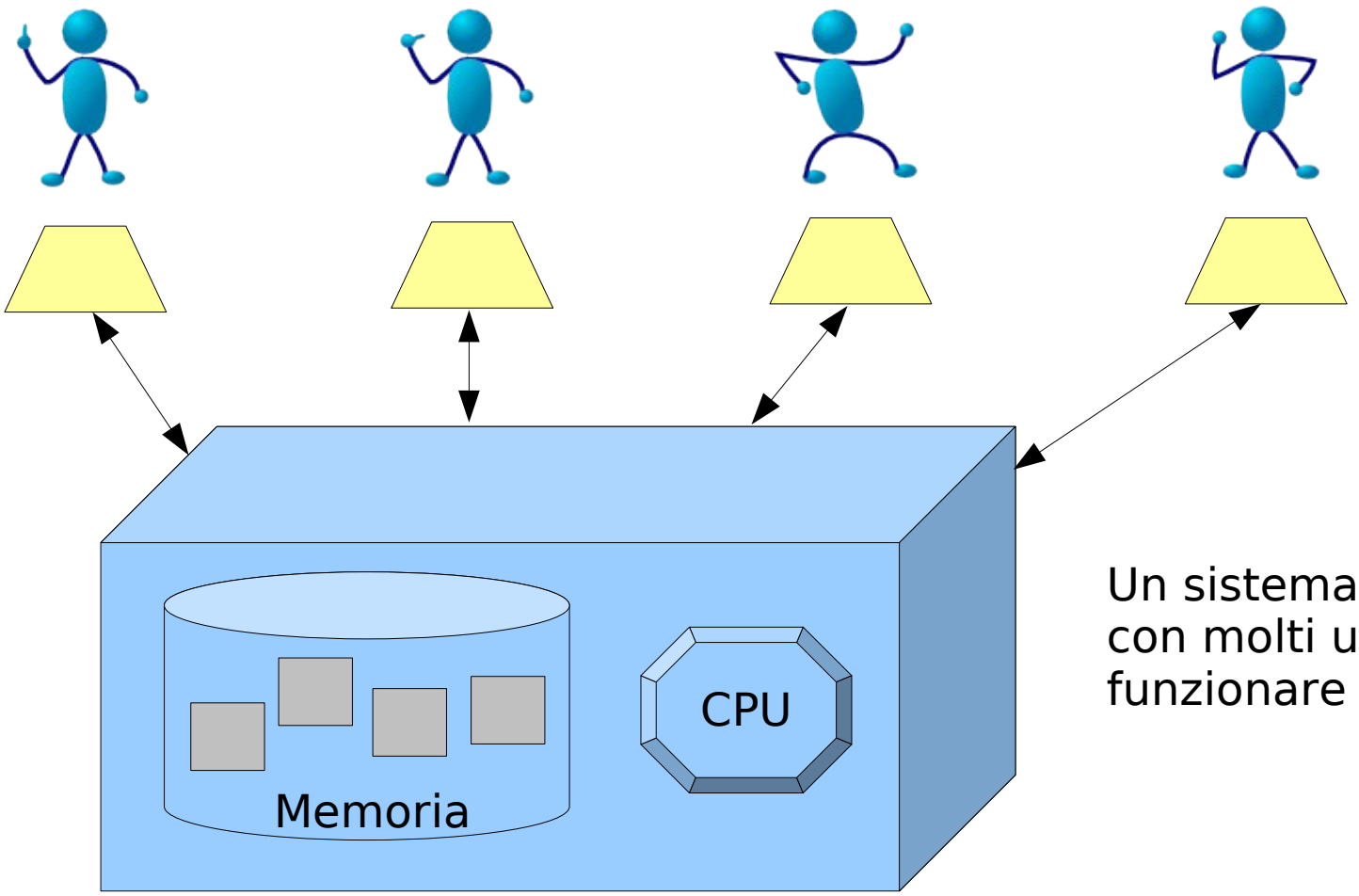
fonte: http://www.catb.org/~esr/writings/taouu/html/ch02s02.html

Terminale telescrivente

“Command-line interfaces (CLIs) evolved from batch monitors connected to the system console. Their interaction model was a series of request-response transactions, with requests expressed as textual commmands in a specialized vocabulary. Latency was far lower than for batch systems, dropping from days or hours to seconds. Accordingly, command-line systems allowed the user to change his or her mind about later stages of the transaction in response to real-time or near-real-time feedback on earlier results. Software could be exploratory and interactive in ways not possible before. But these interfaces still placed a relatively heavy mnemonic load on the user, requiring a serious investment of effort and learning time to master.

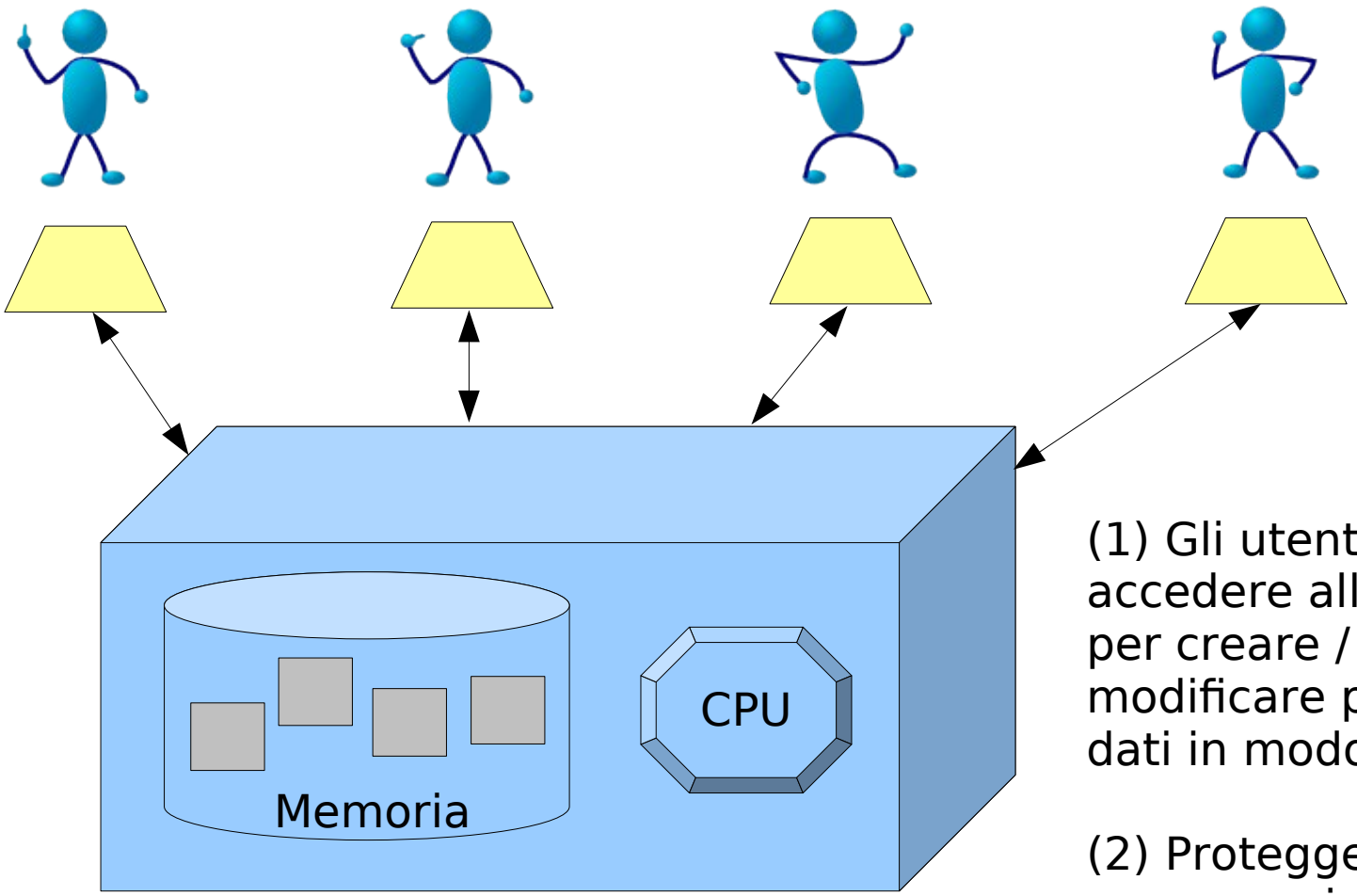
Command-line interfaces were closely associated with the rise of timesharing computers. The concept of timesharing dates back to the 1950s; the most influential early experiment was the MULTICS operating system after 1965”

Nuove Esigenze 1



Un sistema interattivo con molti utenti può funzionare batch?

Nuove Esigenze 2



(1) Gli utenti devono poter accedere alla memoria per creare / salvare / modificare programmi e dati in modo agevole

(2) Proteggere dati e programmi dagli altri utenti

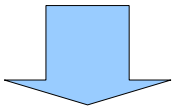
Time-sharing e file system

con i sistemi time-sharing nascono alcuni concetti fondamentali:

file: collezione di dati correlati

directory: raggruppamento (logico) di file

file system: organizzazione (logica) strutturata della memoria



il SO deve avere opportune astrazioni che gli consentono di rappresentare e gestire file, directory e file system

Quesito

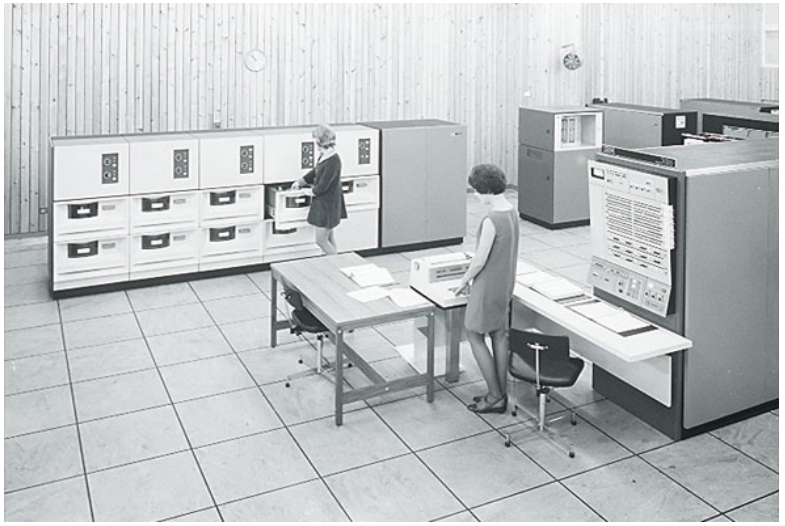
- gli elaboratori che abbiamo descritto erano più o meno “potenti” computazionalmente di quelli attuali (compreso il vostro portatile o desktop personale)?
- potevano fare più o meno cose?

Erano uguali

anni '60, es. IBM 704

sistemi operativi più avanzati furono sviluppati per consentire agli utenti di seguire in modo interattivo i job in esecuzione su di una certa macchina: sistemi in time-sharing

gli utenti interagivano con i programmi in esecuzione seduti ad appositi terminali, puri dispositivi di I/O senza capacità di calcolo, collegati con l'elaboratore



Nel 1962 *John Larry Kelly, Jr* presso i Bell Labs utilizzò un IBM 704 per la sintesi del parlato. Ricreò la canzone *"Daisy Bell"*, con accompagnamento musicale di Max Mathews. Arthur C. Clarke, autore di *"2001: A Space Odyssey"* si trovava in quel mentre in visita a un amico, collega di Kelly, ed assistette a una dimostrazione e ne fu così colpito che HAL 9000, il computer, canta la stessa canzone.

Ed Thorp utilizzò l'IBM 704 per per formulare una teoria del gioco del blackjack.

(fonte: wikipedia)

Personal Computer e TCP/IP

anni '70

- l'evoluzione dell'elettronica comporta la miniaturizzazione delle componenti e la nascita di computer meno cososi e ingombranti. Nasce il concetto di “personal computer”: calcolatore dedicato a un singolo utente
- Quali effetti??

Personal Computer e TCP/IP

anni '70

- l'evoluzione dell'elettronica comporta la miniaturizzazione delle componenti e la nascita di computer meno cososi e ingombranti. Nasce il concetto di “personal computer”: calcolatore dedicato a un singolo utente
- I SO regrediscono:
  - non gestiscono la multiutenza
  - si torna alla monoprogrammazione
  - niente meccanismi di protezione
- Nascono lo standard di comunicazione TCP/IP e ethernet (Xerox), si riducono i costi delle LAN (local area networks)

Colpo d'occhio: per avere un'idea

apple-1 (1976)

realizzato da Steve Wozniak e da Steve Jobs
(precedentemente sviluppatore di giochi per Atari)

ha un tale successo che la Commodore cerca di comperare la Apple

Home Computer

collegabile a una TV (da usare come monitor)
display rate: 60 caratteri al secondo
RAM: 1K
ROM: 256 byte
CPU: MOS Technology 6502
Porte I/O: monitor, lettore di nastri, tastiera
tastiera venduta separatamente
niente grafica, suoni, colori
prezzo: 666.66 $

commodore PET2001
(1977)

Professional computer

RAM: 4K
ROM: 14K
CPU 6502
monocromo, 40x25 caratteri
built-in: registratore di nastri
costo: 700 £

fonte: http://www.old-computers.com/museum

In epoca piu` moderna ...

driver x il floppy
contenente il SO

driver per il floppy
contentente il programma

IBM PC (1981)

apple macintosh SE (1987)

RAM: 64K
ROM: 64K
CPU: Intel 8088 (opzionale coprocessore matematico)
monocromo
grafica: 300x200 oppure 640x200
suoni: generatore di toni
costo: 1736 £

RAM: 1M (estendibile a 4)
ROM: 256K
CPU: Motorola MC 68000
monocromo
grafica: 512 x 342
suoni: 8-bit mono sound chip
costo: 3400 $

fonte: http://www.old-computers.com/museum

Infine ...

olivetti M24 (1984)

quando vendevamo
PC agli americani

Computer professionale
RAM: 128K (espandibile a 640K)
ROM: 16K
CPU: Intel 8086 (con coprocessori matematico e grafico opzionali)
mod. grafica: 320x200 a 4 colori/640x200 mono/ 640x400 mono
suoni: generatore di toni
possibilità di uso del mouse

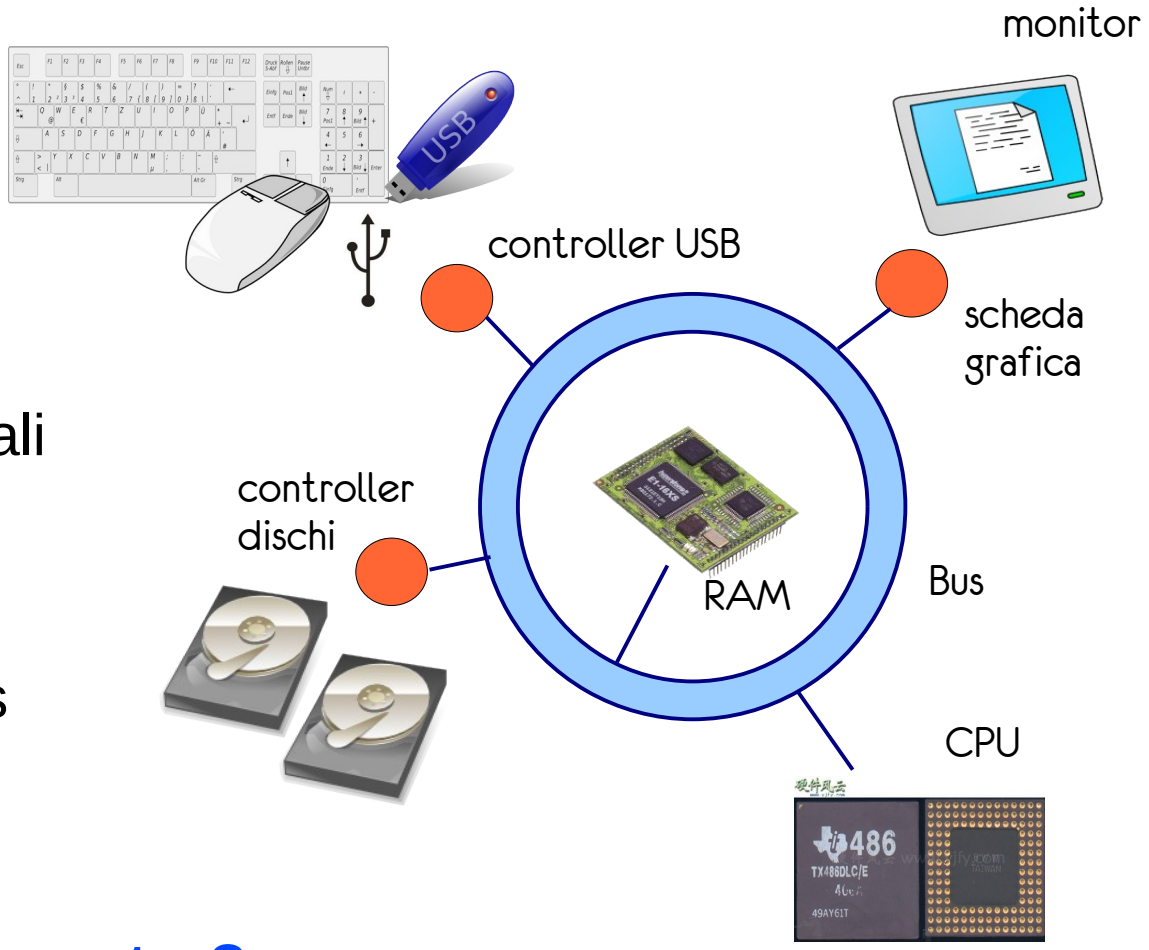
fonte: http://www.old-computers.com/museum

Strutture dei sistemi di calcolo

Organizzazione di un sistema di calcolo

- CPU
- memoria principale (RAM)
- vari dispositivi, ognuno dei quali ha un apposito controllore

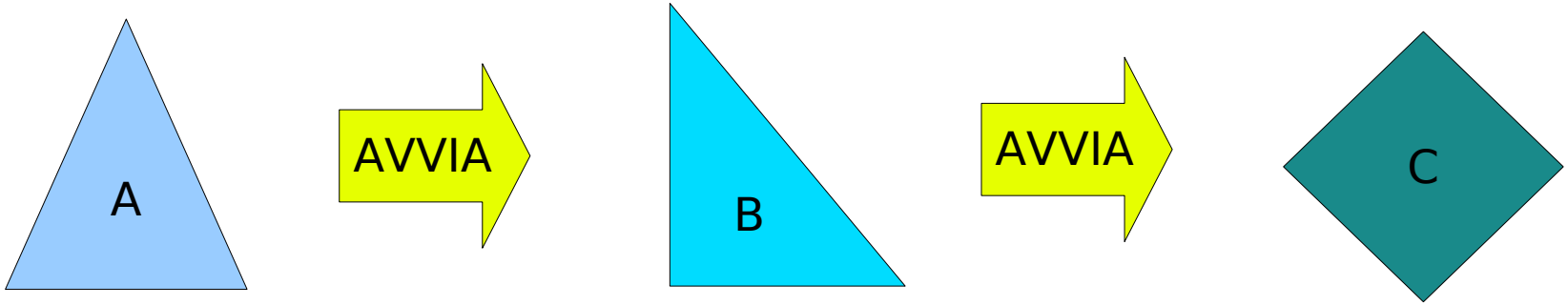
Controllori di dispositivo, RAM e CPU sono connessi da un Bus



Come si avvia un computer?

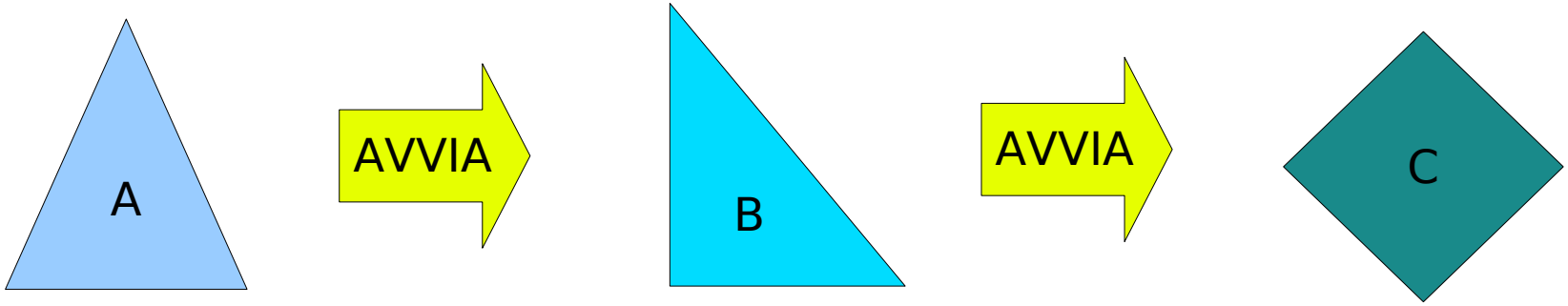
Bootstrap di un sistema di calcolo

BOOTSTRAP: meccanismo per cui un processo ne avvia un altro.
Il SO di un sistema di calcolo è avviato dal BIOS (Basic I/O System), programma tipicamente contenuto in una memoria di tipo ROM (read-only mem.) o EEPROM (electrically erasable programmable mem.)



Bootstrap di un sistema di calcolo

BOOTSTRAP: meccanismo per cui un semplice processo ne avvia un altro.
Il SO di un sistema di calcolo è avviato dal **BIOS** (Basic I/O System), programma tipicamente contenuto in una memoria di tipo ROM (read-only mem.) o EEPROM (electrically erasable programmable mem.)



Bootstrap di un sistema di calcolo

Il BIOS:
- inizializza tutte le componenti della macchina
- individua il SO, lo carica in memoria principale e ne avvia l'esecuzione (bootstrap)

Il SO:
- avvia l'esecuzione del primo processo di elaborazione
- rimane in attesa di un evento

Bootstrap di un sistema di calcolo

Il BIOS:
inizializza tutte le componenti della macchina
individua il SO, lo carica in memoria principale e ne avvia l'esecuzione (bootstrap)

Il SO:
avvia l'esecuzione del primo processo di elaborazione
rimane in attesa di un evento

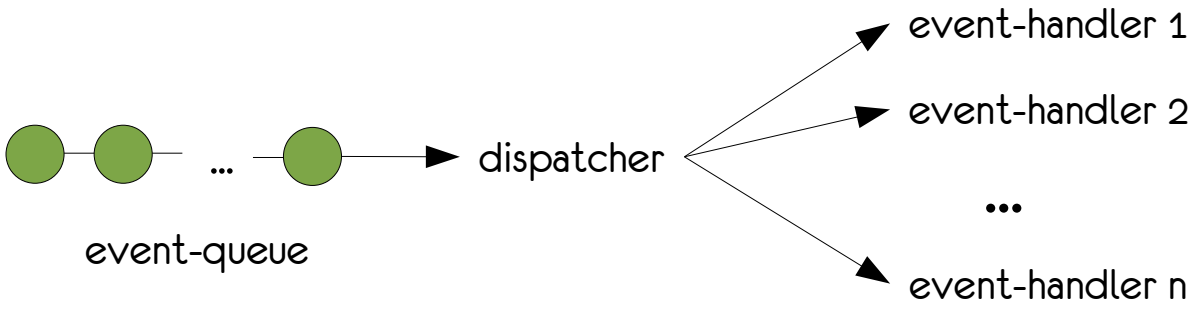
I SO sono programmi **event-driven** (guidati dagli eventi). Sono **reattivi**

Tipi di eventi:

- **interrupt**
  sono causati dai dispositivi fisici, sono segnali inviati alla CPU attraverso il bus
- **trap**
  sono causati dai programmi, possono codificare errori (*eccezioni*) oppure corrispondere richieste di esecuzioni speciali (*system call* o *supervisor call*)

Programmi event-driven

Come funziona un programma event-driven?



di base possiamo immaginare il codice come un semplice ciclo:

```
forever
      // attendi
      wait
      // all'occorrenza di un evento
      // gestiscilo
      on trigger handle(event)
end
```

a ogni evento corrisponde una routine di gestione predefinita detta “**event-handler**”

gestire un evento significa effettuare un “**dispatch**”, cioè individuare ed eseguire la routine associata all'evento

In generale gli eventi possono essere accumulati in una coda

Eventi ed efficienza

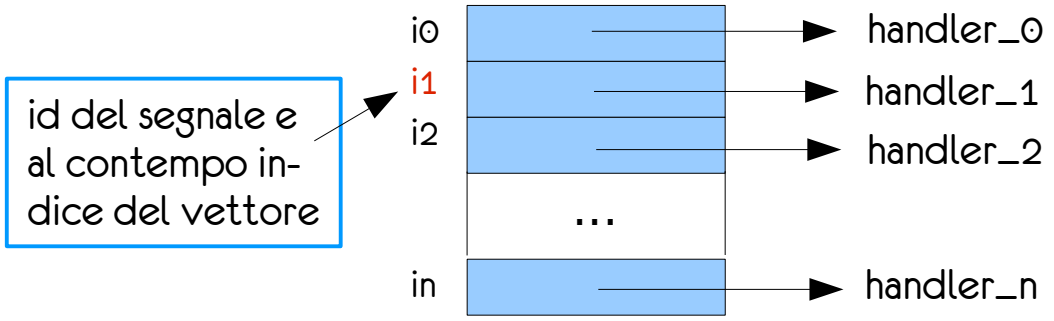
- La gestione degli eventi **deve** essere molto efficiente in un SO.
- Considerando che l'elenco dei possibili eventi da gestire è **predefinito**, è possibile implementare il dispatch attraverso un vettore di puntatori agli event_handler: basta associare a ogni evento un numero (id dell'evento e al contempo indice nel vettore). Il vettore è detto **vettore delle interruzioni** (interrupt vector)

```
baroglio@esmeralda:~/BACKUP/LAB_SO/Slide$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0      0 994084 157708 572248    0    0   299   205  295  815 12  2 78  7
baroglio@esmeralda:~/BACKUP/LAB_SO/Slide$
```

in: numero di interrupt per secondo
cs: numero di context switch per secondo

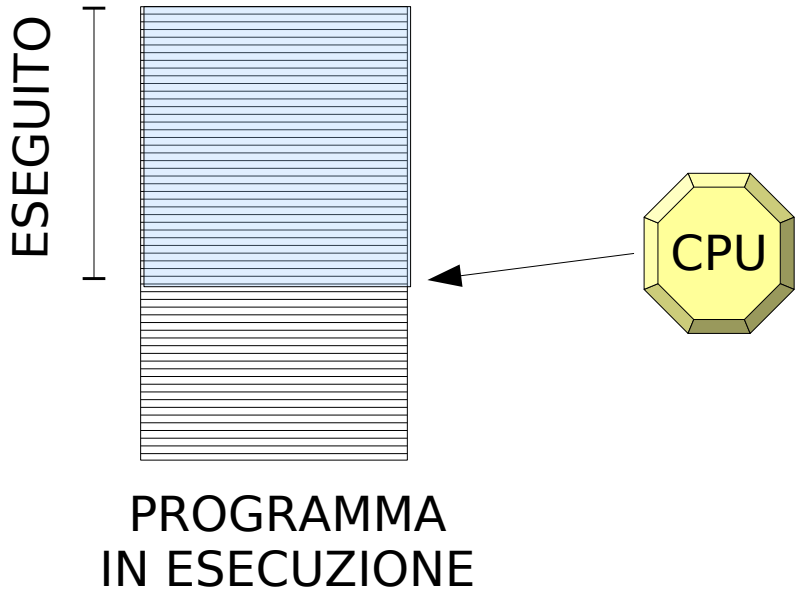
Eventi ed efficienza

- La gestione degli eventi **deve** essere molto efficiente in un SO.
- Considerando che l'elenco dei possibili eventi da gestire è predefinito, è possibile implementare il dispatch attraverso un vettore di puntatori agli event_handler: basta associare a ogni evento un numero (id dell'evento e al contempo indice nel vettore). Il vettore è detto **vettore delle interruzioni** (interrupt vector)
- Il vettore delle interruzioni è mantenuto nella memoria bassa (es. prime 100 posizioni), l'accesso avviene in modo diretto tramite un **indice codificato nello stesso segnale che notifica l'evento**, senza bisogno di un vero dispatcher, basta il segnale stesso



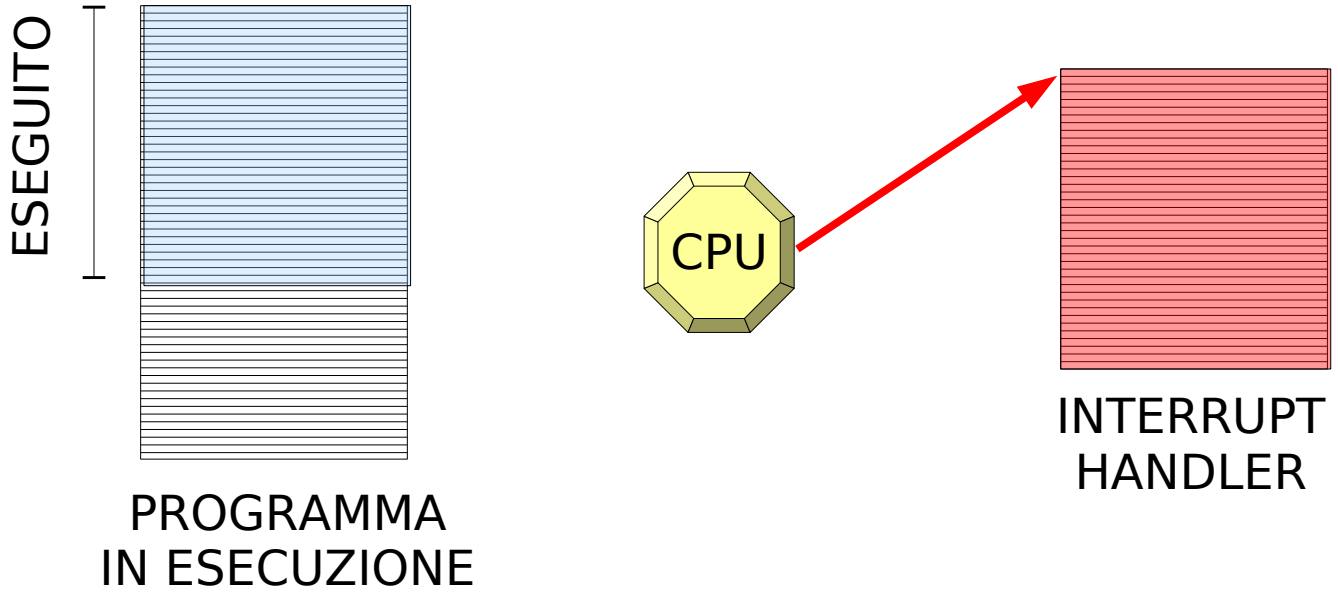
Eventi ed efficienza

- la gestione di un evento può interrompere la normale esecuzione di un programma, per questo motivo l'istruzione interrotta va memorizzata per poi riprenderne l'esecuzione appena possibile.
- La gestione di un'interruzione causa un **context switch** (cambiamento di contesto)



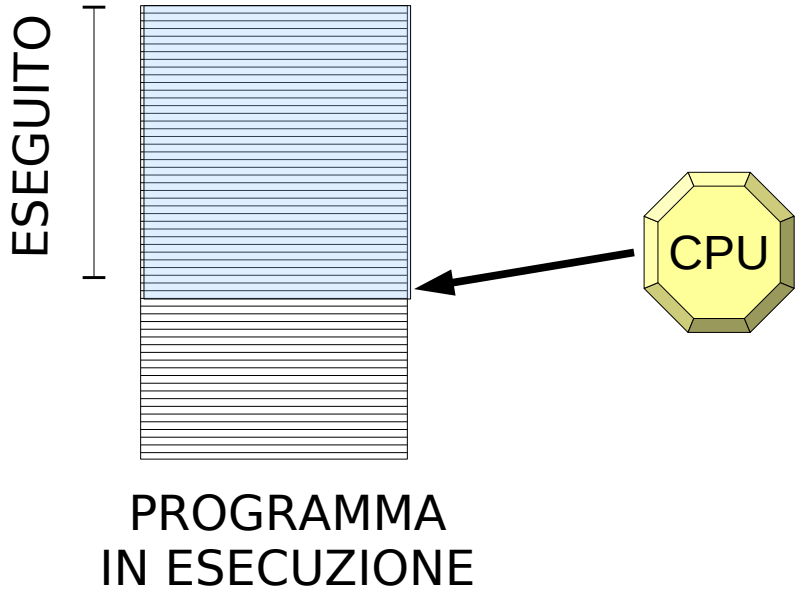
Eventi ed efficienza

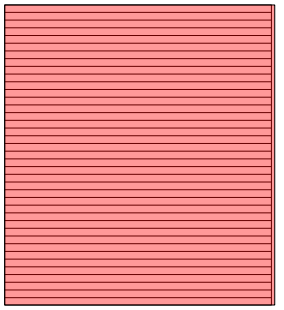
- la gestione di un evento può interrompere la normale esecuzione di un programma, per questo motivo l'istruzione interrotta va memorizzata per poi riprenderne l'esecuzione appena possibile.
- La gestione di un'interruzione causa un **context switch** (cambiamento di contesto)



Eventi ed efficienza

- la gestione di un evento può interrompere la normale esecuzione di un programma, per questo motivo l'istruzione interrotta va memorizzata per poi riprenderne l'esecuzione appena possibile.
- La gestione di un'interruzione causa un **context switch** (cambiamento di contesto)





INTERRUPT
HANDLER

Quando la routine termina occorre riprendere l'esecuzione del programma dal punto in cui si è interrotta