Sistemi Operativi

Introduzione ai sistemi operativi
II semestre

Renzo Davoli
Alberto Montresor

I semestre

- **Uso del S.O. UNIX: shell - command line interface** *(prova pratica + progetto)*
- **Linguaggio C** *(prova pratica + progetto)*
- **Programmazione UNIX: l'interfaccia delle System Call** *(prova pratica)*
- **Programmazione Concorrente** *(scritto "concorrenza")*
- **Shell scripting** *(prova pratica)*
- **Python** *(prova pratica)*
- **Strumenti di sviluppo */* gestione dei progetti (make, autotools, cmake, git..)** *(progetto)*
- **Progetto uMPS3 + phase 1** *(progetto)*

Cos'è un sistema operativo?

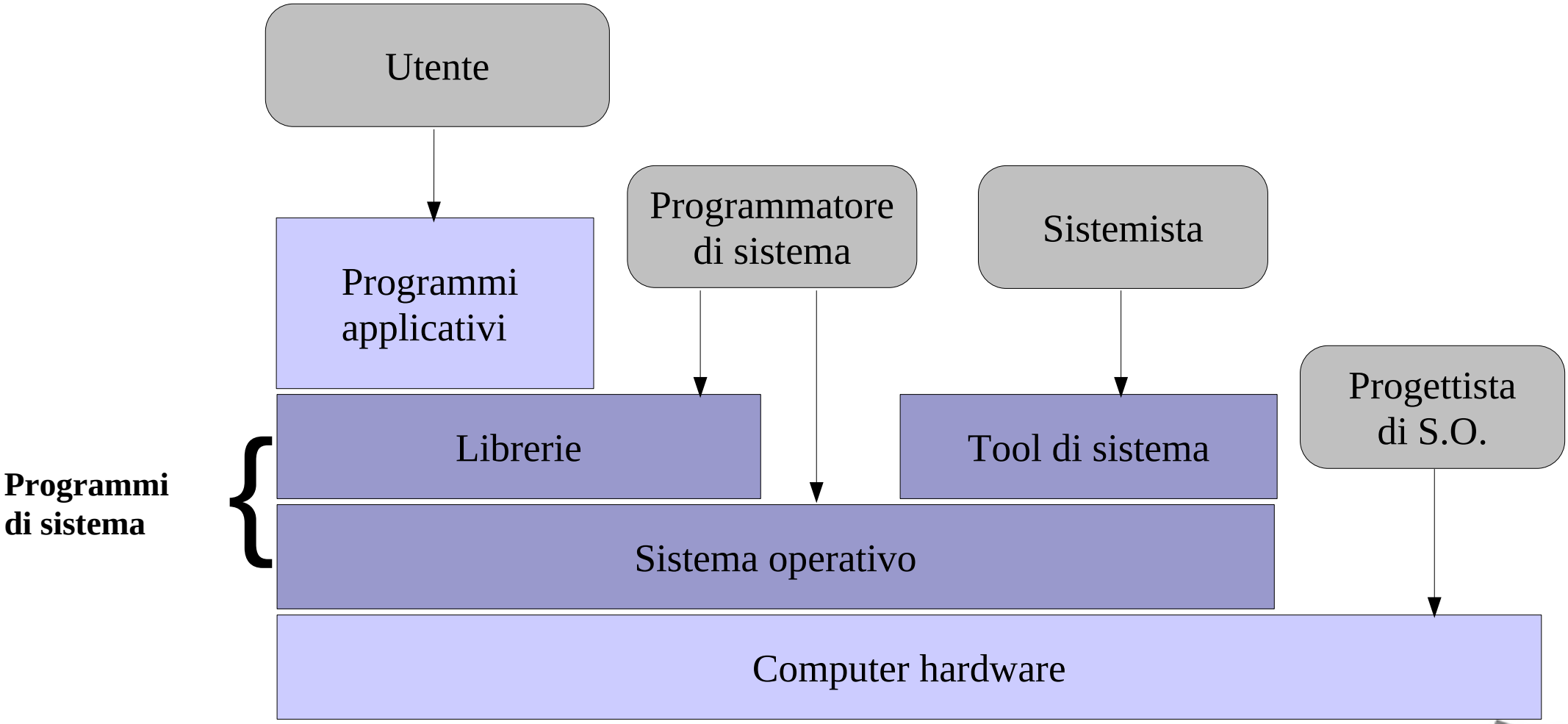
- **Definizione:**
  - Un sistema operativo è livello di astrazione:
    - realizza il concetto di processo
    - Il “linguaggio” fornito dal S.O. è definito dalle system call
    - È implementato tramite un programma che *controlla l'esecuzione di programmi applicativi* e agisce come *interfaccia tra le applicazioni e l'hardware* del calcolatore
- **Obiettivi**
  - Efficienza:
    - Un S.O. cerca di utilizzare in modo efficiente le risorse del calcolatore
  - Semplicità:
    - Un sistema operativo dovrebbe semplificare l'utilizzazione dell'hardware di un calcolatore

S.O. come gestore di risorse

- **Alcune osservazioni:**
  - Gestendo le risorse di un calcolatore, un S.O. controlla il funzionamento del calcolatore stesso...
  - ... ma questo controllo è esercitato in modo "particolare"
- **Normalmente:**
  - Il meccanismo di controllo è esterno al sistema controllato
  - Esempio: termostato e impianto di riscaldamento
- **In un elaboratore:**
  - Il S.O. è un programma, simile all'oggetto del controllo, ovvero le applicazioni controllate
  - Il S.O. deve lasciare il controllo alle applicazioni e affidarsi al processore per riottenere il controllo

S.O. come macchina estesa

- **Visione "a strati" delle componenti hardware/software che compongono un elaboratore:**



S.O. come macchina estesa

- **In questa visione, un sistema operativo:**
  - nasconde ai programmatori i dettagli dell'hardware e fornisce ai programmatori una API conveniente e facile da usare
  - agisce come intermediario tra programmatore e hardware
- **Parole chiave:**
  - Indipendenza dall'hardware
  - Comodità d'uso
  - Programmabilità

S.O. come macchina estesa

- **Esempio: floppy disk drive**
  - I floppy drive delle macchine Intel sono compatibili con il controllore NEC PD765
  - 16 comandi
    - inizializzazione, avviamento motore, spostamento testina, lettura-scrittura, spegnimento motore
    - formato: vari parametri, impacchettati in 1-9 byte
    - esempio: comando read, 13 parametri
  - al completamento, il driver restituirà 23 campi di stato e di errore racchiusi in 7 byte

S.O. come macchina estesa

- **Esempio senza S.O.**

```
li $t0, 0xDEFF12 # init

sw $t0, 0xB000.0040

li $t0, 0xFFDF # motor

sw $t0, 0xB000.0044

li $t0, 0xFFBB

sw $t0, 0xBOOO.0048

...
```

NB: Questo è esempio serve a dare un'idea, la realtà è molto più complessa....

- **Esempio con S.O.**

```
fd = open("/etc/rpc");

read(fd, buffer, size);
```

S.O. come macchina estesa

- **Servizi estesi offerti da un S.O.**
- **(sono le classi di SysCall studiate nel I semestre):**
  - esecuzione di programmi
  - accesso semplificato/unificato ai dispositivi di I/O
  - accesso a file system
  - accesso a networking
  - accesso al sistema
  - rilevazione e risposta agli errori
  - accounting

Storia dei sistemi operativi

Storia dei Sistemi Operativi

- **L'evoluzione dei sistemi operativi**
  - *è stata spinta* dal progresso tecnologico nel campo dell'hardware
  - *ha guidato* il progresso tecnologico nel campo dell'hardware
- **Esempio:**
  - Gestione degli interrupt
  - Protezione della memoria
  - Memoria virtuale
  - ....

Storia dei Sistemi Operativi

- **Perché analizzare la storia dei sistemi operativi?**
  - Perché permette di capire l'origine di certe soluzioni presenti oggi nei moderni sistemi operativi
  - Perché è l'approccio didattico migliore per capire come certe idee si sono sviluppate
  - Perché alcune delle soluzioni più vecchie sono ancora utilizzate
- **Durante il corso:**
  - illustreremo ogni argomento
    - partendo dalle prime soluzioni disponibili
    - costruendo sopra di esse soluzioni mano a mano più complesse
  - non stupitevi quindi se alcune soluzioni vi sembreranno banali e ingenue; sono soluzioni adottate 10,20,30,40 o 50 anni fa!

Storia dei Sistemi Operativi

- **Generazione 1: 1945 - 1955**
  - valvole e tavole di commutazione
- **Generazione 2: 1955 - 1965**
  - transistor e sistemi batch
- **Generazione 3: 1965 – 1980**
  - circuiti integrati, multiprogrammazione e time-sharing
- **Generazione 4: 1980 – oggi**
  - personal computer

Generazione 0

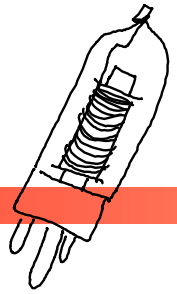
- **Babbage (1792-1871)**
  - Cerca di costruire la macchina analitica (programmabile, meccanica)
  - Non aveva sistema operativo
  - La prima programmatrice della storia e' Lady Ada Lovelace (figlia del poeta Lord Byron)

telaio Jackward 1773
a schede perforate
lì dentro c'era l'idea di esecuzione di programma

riv. francese e americana

Menabrea primo ministro regno di sardegna che si occupa di scrivere programmi per m. analitica

Generazione 1 (1944-1955)



- **Come venivano costruiti?**
  - macchine a valvole e tavole di commutazione
- **Come venivano usati?**
  - solo calcoli numerici (calcolatori non elaboratori)
  - un singolo gruppo di persone progettava, costruiva, programmava e manuteneva il proprio computer
- **Come venivano programmati?**
  - in linguaggio macchina
    - programmazione su tavole di commutazione
    - non esisteva il concetto di assembler!
- **Nessun sistema operativo!**

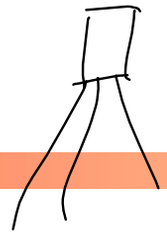
Generazione 1 (1944-1955)

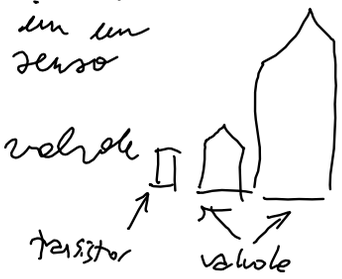
- **Principali problemi**
  - grossi problemi di affidabilità (guasti frequenti)
  - rigidità nell'assegnazione dei ruoli;
    - non esiste il concetto di programmatore come entità separata dal costruttore di computer e dall'utente
  - utilizzazione lenta e complessa; l'operatore doveva:
    - caricare il programma da eseguire
    - inserire i dati di input
    - eseguire il programma
    - attendere il risultato
    - ricominciare dal punto 1.
  - tutto ciò a causa dell'assenza del sistema operativo

Generazione 1 (1944-1955)

- **Frasi celebri**
  (da una lezione di P. Ciancarini)
  - Nel futuro i computer arriveranno a pesare non più di una tonnellata e mezzo (Popular Mechanics, 1949)
  - Penso che ci sia mercato nel mondo per non più di cinque computer (Thomas Watson, presidente di IBM, 1943)
  - Ho girato avanti e indietro questa nazione (USA) e ho parlato con la gente. Vi assicuro che questa moda dell'elaborazione automatica non vedrà l'anno prossimo
    (Editor di libri scientifici di Prentice Hall, 1947)

Generazione 2 (1955-1965)





- **Come venivano costruiti?**
  - introduzione dei transistor
  - costruzione di macchine più affidabili ed economiche
- **Come venivano usati?**
  - le macchine iniziano ad essere utilizzate per compiti diversi
  - si crea un mercato, grazie alle ridotte dimensioni e al prezzo più abbordabile
  - avviene una separazione tra costruttori, operatori e programmatori
- **Come venivano programmati?**
  - linguaggi ad "alto livello": Assembly, Fortran
  - tramite schede perforate
- **Sistemi operativi batch**

lunedì 8 Agosto 2022

dice il nonno:

e^- sono elettroni

e^+ sono positroni (antiparticelle corrispondenti agli elettroni).

lui utilizzava a fisica quando studiava Fortran su schede perforate. quindi era della generazione 2.

Generazione 2 (1955-1965)

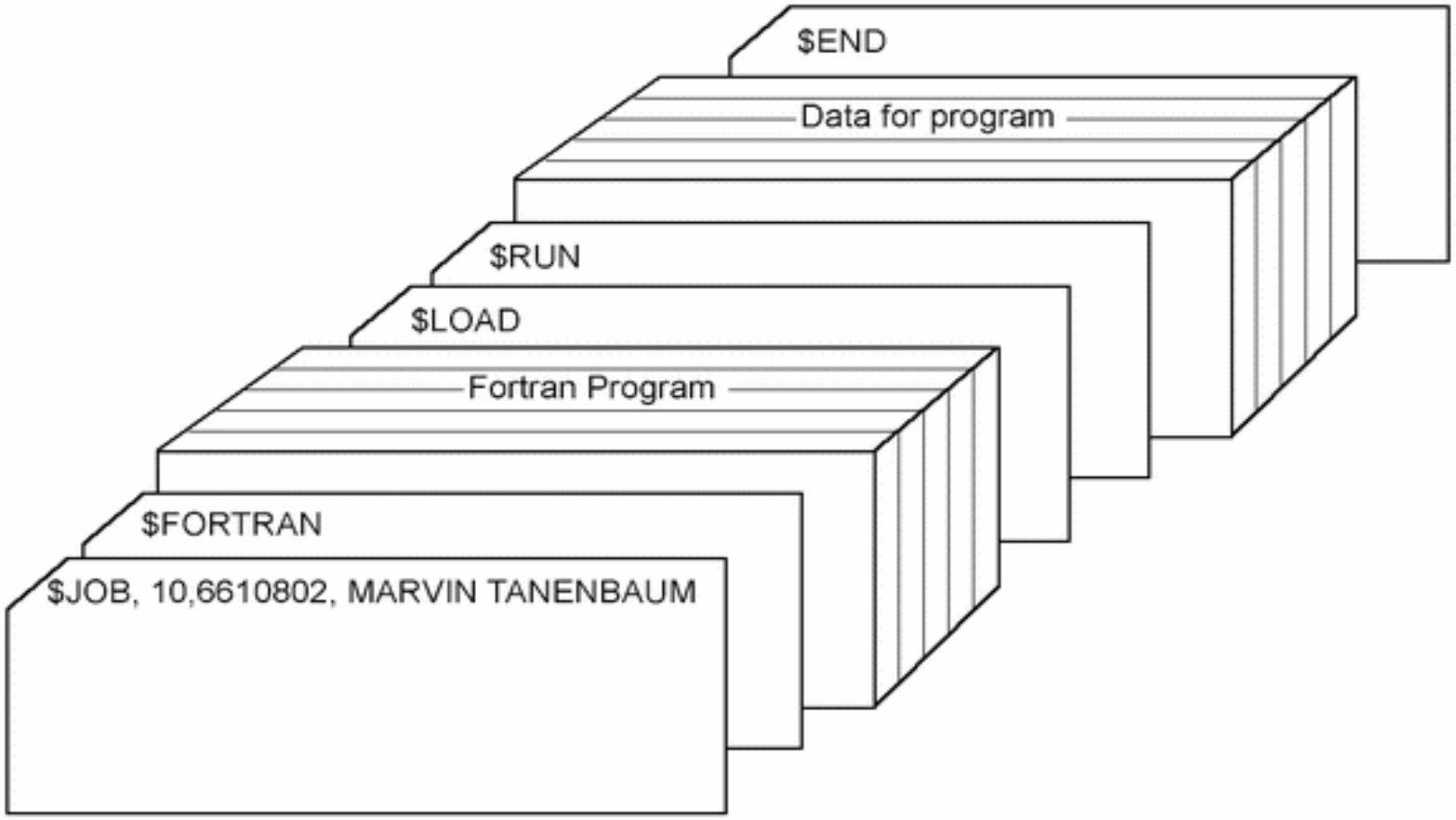
- **Definizione: job**
  - Un programma o un'insieme di programmi la cui esecuzione veniva richiesta da uno degli utilizzatori del computer
- **Ciclo di esecuzione di un job:**
  - il programmatore
    - scrive (su carta) un programma in un linguaggio ad alto livello
    - perfora una serie di schede con il programma e il suo input
    - consegna le schede ad un operatore
  - l'operatore
    - inserisce schede di controllo scritte in JCL
    - inserisce le schede del programma
    - attende il risultato e lo consegna al programmatore
- **Nota: operatore != programmatore == utente**

Generazione 2 (1955-1965)

- **Sistema operativo**
  - *primi rudimentali esempi di sistema operativo*, detti anche monitor residenti:
    - controllo iniziale nel monitor
    - il controllo viene ceduto al job corrente
    - una volta terminato il job, il controllo ritorna al monitor
  - il monitor residente è in grado di eseguire una sequenza di job, trasferendo il controllo dall'uno all'altro
  - Esegue un solo job alla volta
- **Detti anche sistemi batch ("infornata")**

Generazione 2 (1955-1965)

- **Job Control Language (JCL-FMS)**

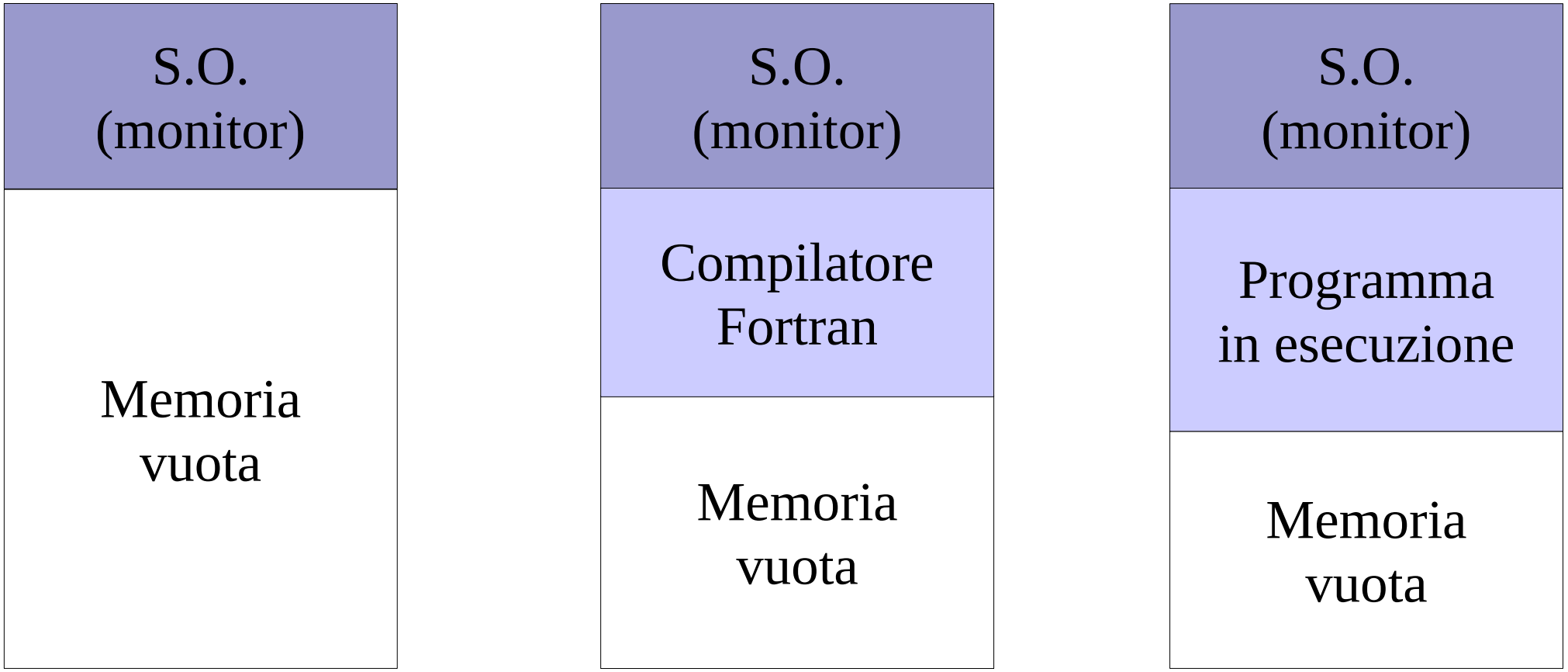


(da una lezione di A.S. Tanenbaum)

Generazione 2 (1955-1965)

monotask.

- **Memory layout per un S.O. batch (versione semplificata)**



1. Stato iniziale

2. Dopo il caricamento del compilatore fortran

3. Dopo la compilazione

2022-02-25 SO II Sem pt1

slide: 00-intro.pdf (inizio min: 5:20)

Allora, ieri abbiamo visto queste belle cose:

- cosa abbiamo fatto
- Il Sistema Operativo è un LIVELLO di ASTRAZIONE, fornisce un API in maniera tale che il processo utente venga confinato, controllato in maniera tale che il processo utente possa (metafora con il mondo reale) convivere civilmente con processi che condividono

[[...(salto della registrazione al minuto: 6:05)...]]

gestore di risorse, rende tutto + facile, macchina estesa per rendere programmi portabili

abbiamo lavorato nella storia ...

- vorrei tornare qui (alla storia slides 10-37) un attimo perché "ciò che avviene in passato non è detto che non sia utile anche nel futuro"

questo, che sembra un architettura antica desueta

(generazione 2 (1955-1965) slide 22

(• Memory layout per un S.O. batch (versione semplificata))

in realtà è usata ma non in sistemi general purpose. Chi di voi ha giocato con un arduino, per esempio, sa che non si può dire che ha un S.O. ma ha proprio un monitor (oppure chiamato loader per l'arduino), cioé l'arduino altro non è che un micro-controllore (adesso se ne usano di vario genere

compreso adesso il nuovo chip della raspberry pi, comunque quello classico è un "at mega" di qualche genere diciamo per molti arduini è il 328) e che sarebbe un degno sistema però per inserire / caricare il programma avrebbe bisogno di un device esterno. L'arduino è di fatto un micro-controllore con in più un loader, un loader che viene pre-caricato, che viene mantenuto in una epron (non volatile eepron). Arduino "at mega" non è una macchina di Von Neumman è una macchina di Hardvard, quindi ha due memorie separate: una per i programmi e una per i dati. Nella memoria programmi che è

[[...(salto delle registrazioni al minuto: 8:36 - 8:47)...]]

non volatile. Per essere un arduino viene pre-caricato un piccolo programma, che può essere simile a questo monitor, che all'accensione o al reset per pochi secondi si ferma e guarda se sta (-nno) arrivando dati dalla seriale emulata su USB, e se in questi pochi secondi arrivano dati li interpreta come un programma da caricare, sempre nella parte programmi OLTRE il loader. Quindi assomiglia molto a questo (sempre slide 22) vedete il compilatore Fortran se voi avete l'arduino e il primo programma tipico dell'arduino "blink" viene caricato blink, fa quello che deve fare, poi resettate / riaccendete / caricate un altro programma e fate altre cose. Quindi come vedete queste architetture ritornano

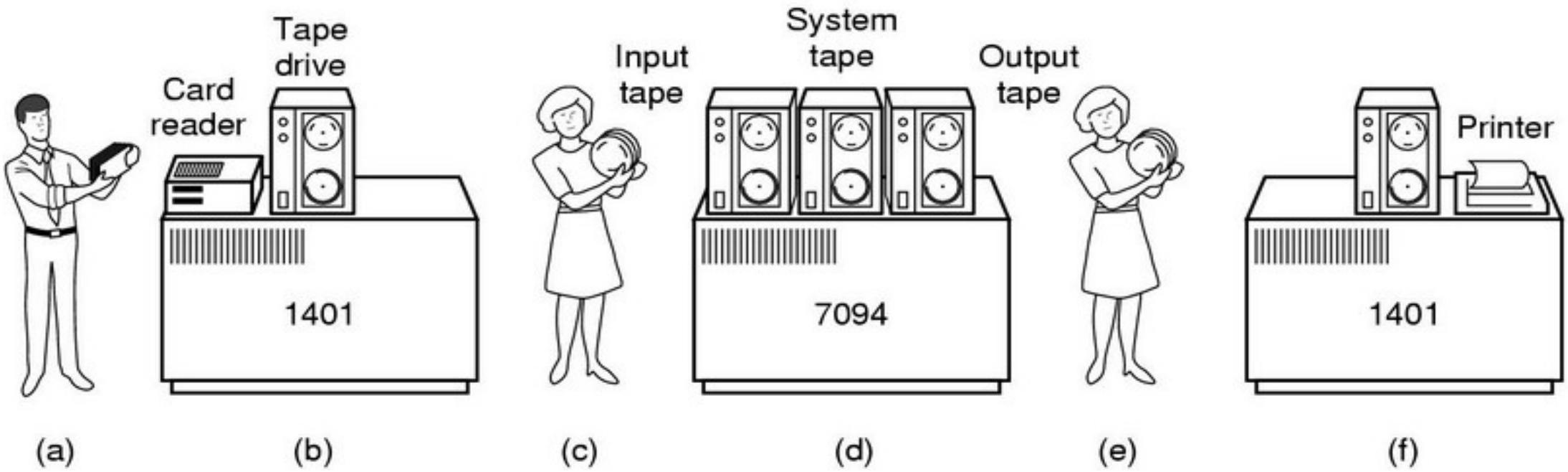
in altri ambienti. Non si può dire che sia un sistema operativo, è embrionale come sistema.

registrazioni minuto: 10:30

Generazione 2 (1955-1965)

- **Principali problemi**
  - Molte risorse restavano inutilizzate:
    - durante le operazioni di lettura schede / stampa, durante il caricamento di un nuovo job, il processore restava inutilizzato
    - parte della memoria restava inutilizzata
  - Primo miglioramento (ma non una soluzione)
    - caricamento di numerosi job su nastro (off-line)
    - elaborazione (output su nastro)
    - stampa del nastro di output (off-line)

Generazione 2 (1955-1965)



(da una lezione di A.S. Tanenbaum)

Generazione 3 (1965-1980)

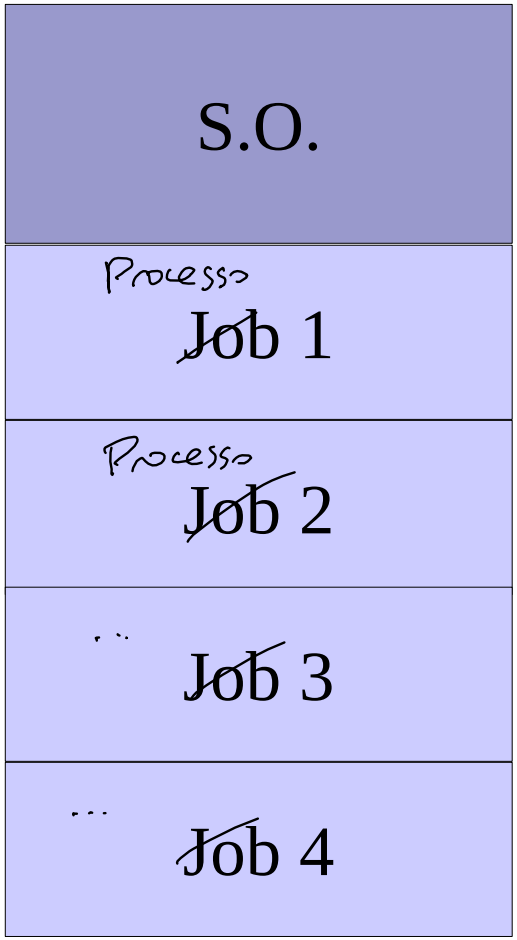
- **Come venivano costruiti?**
  - circuiti integrati
- **Come venivano usati?**
  - man mano sparisce la figura dell'operatore come "interfaccia" degli utenti verso la macchine
  - utente == operatore
- **Come venivano programmati?**
  - linguaggi ad "alto livello": C, shell scripting
  - editor testuali, editor grafici, compilatori
  - accesso al sistema da terminali
- **Quale sistemi operativi venivano usati?**
  - non più batch ma interattivi
  - multi-programmazione
  - time sharing

Generazione 3 - Multiprogrammazione

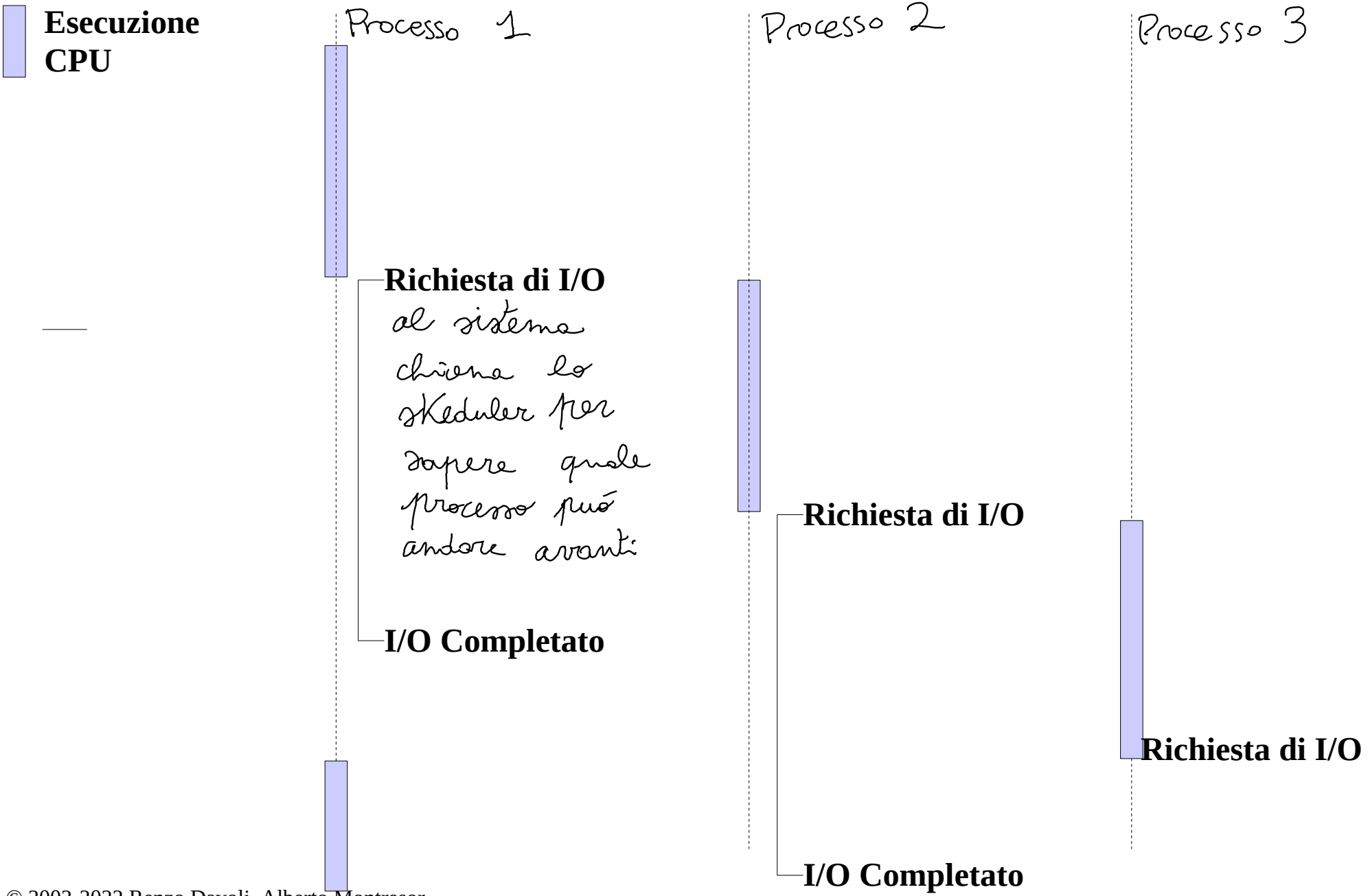
- **Definizione: *multiprogrammazione*** o multitasking
  - utilizzare il processore durante i periodi di I/O di un job per eseguire altri job
- **Vantaggi**
  - il processore non viene lasciato inattivo (*idle*) durante operazioni di I/O molto lunghe
  - la memoria viene utilizzata al meglio, caricando il maggior numero di job possibili

Generazione 3 - Multiprogrammazione

- **Caratteristiche tecniche:**
  - Più job contemporaneamente in memoria
  - Una componente del S.O. detto scheduler si preoccupa di alternarli nell'uso della CPU
  - quando un job richiede un'operazione di I/O, la CPU viene assegnata ad un altro job.



Generazione 3 - Multiprogrammazione



Generazione 3 - Multiprogrammazione

- **S.O. multiprogrammati: quali caratteristiche?**
  - routine di I/O devono essere fornite dal S.O.
  - gestione della memoria
    - il sistema deve allocare la memoria per i job multipli presenti contemporaneamente
  - CPU scheduling
    - il sistema deve scegliere tra i diversi job pronti ad eseguire
  - allocazione delle risorse di I/O
    - Il sistema operativo deve essere in grado di allocare le risorse di I/O fra diversi processi

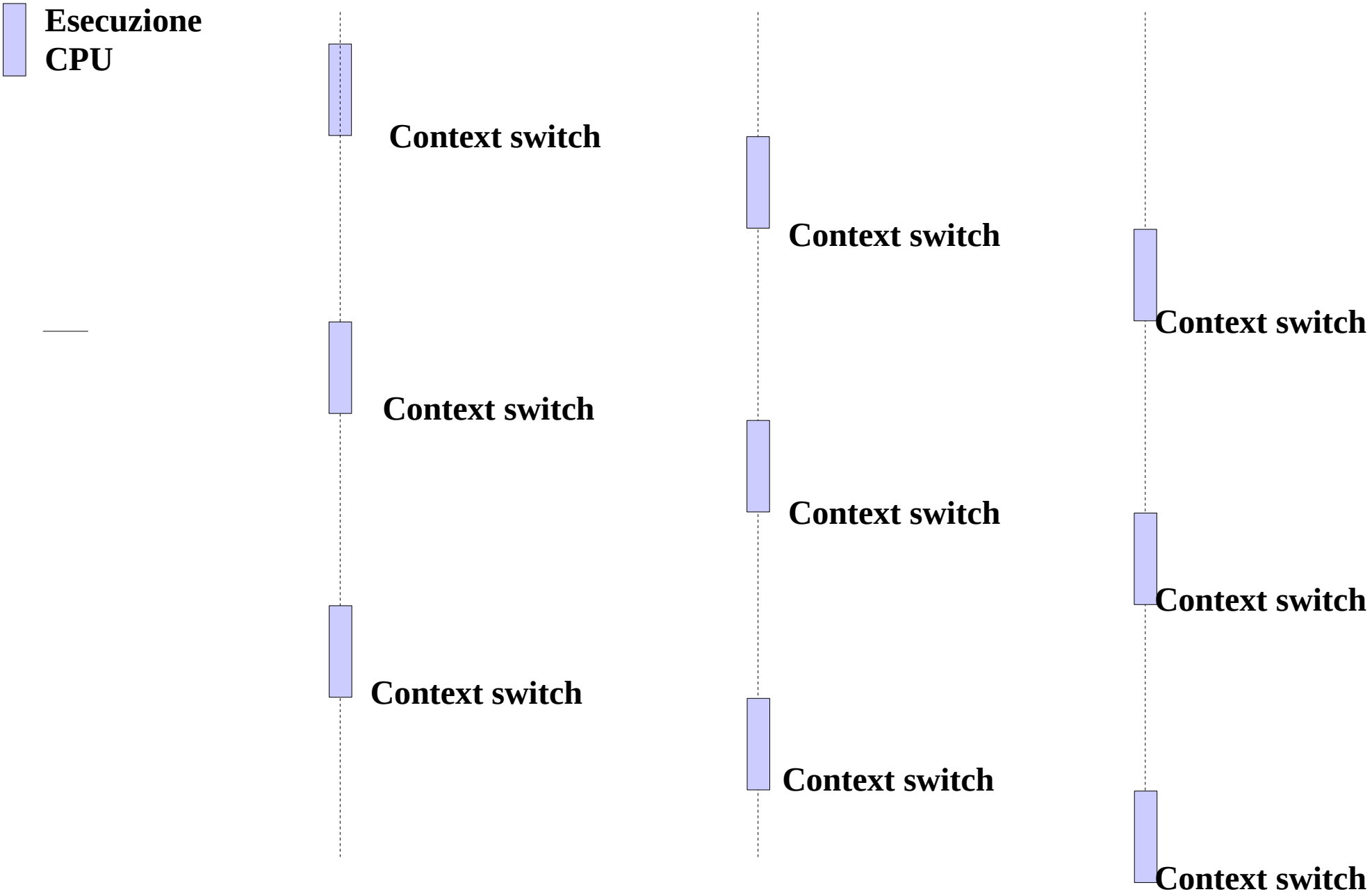
Generazione 3 - Time-sharing

vogliamo farli interagire
"l'umano se non vede che è cambiato niente entro 10 secondi fa reboot del sistema"

- **Definizione - Time sharing**
  - E' l'estensione logica della multiprogrammazione
  - L'esecuzione della CPU viene suddivisa in un certo numero di quanti temporali
  - Allo scadere di un quanto (corto tempo), il job corrente viene interrotto e l'esecuzione passa ad un altro job
    - anche in assenza di richieste di I/O
  - Necessità di un dispositivo hardware (interval timer) (3 millisecondi)
  - I passaggi (*context switch*) avvengono così frequentemente che più utenti possono interagire con i programmi in esecuzione

(mette il processo in coda alla Ready Queue)

Generazione 3 - Time-sharing



Generazione 3 - Time-sharing

- **S.O. time-sharing: quali caratteristiche?**
  - *Gestione della memoria*
    - Il numero di programmi eseguiti dagli utenti può essere molto grande; si rende necessario la gestione della *memoria virtuale*
  - *CPU Scheduling*
    - Lo scheduling deve essere di tipo *preemptive* o *time-sliced*, ovvero sospendere periodicamente l'esecuzione di un programma a favore di un altro
  - *Meccanismi di protezione*
    - La presenza di più utenti rende necessari meccanismi di protezione (e.g. protezione nel file system, della memoria, etc.)

il sistema operativo può dire no tu basta metto avanti quell'altro.

Generazione 3 - Storia

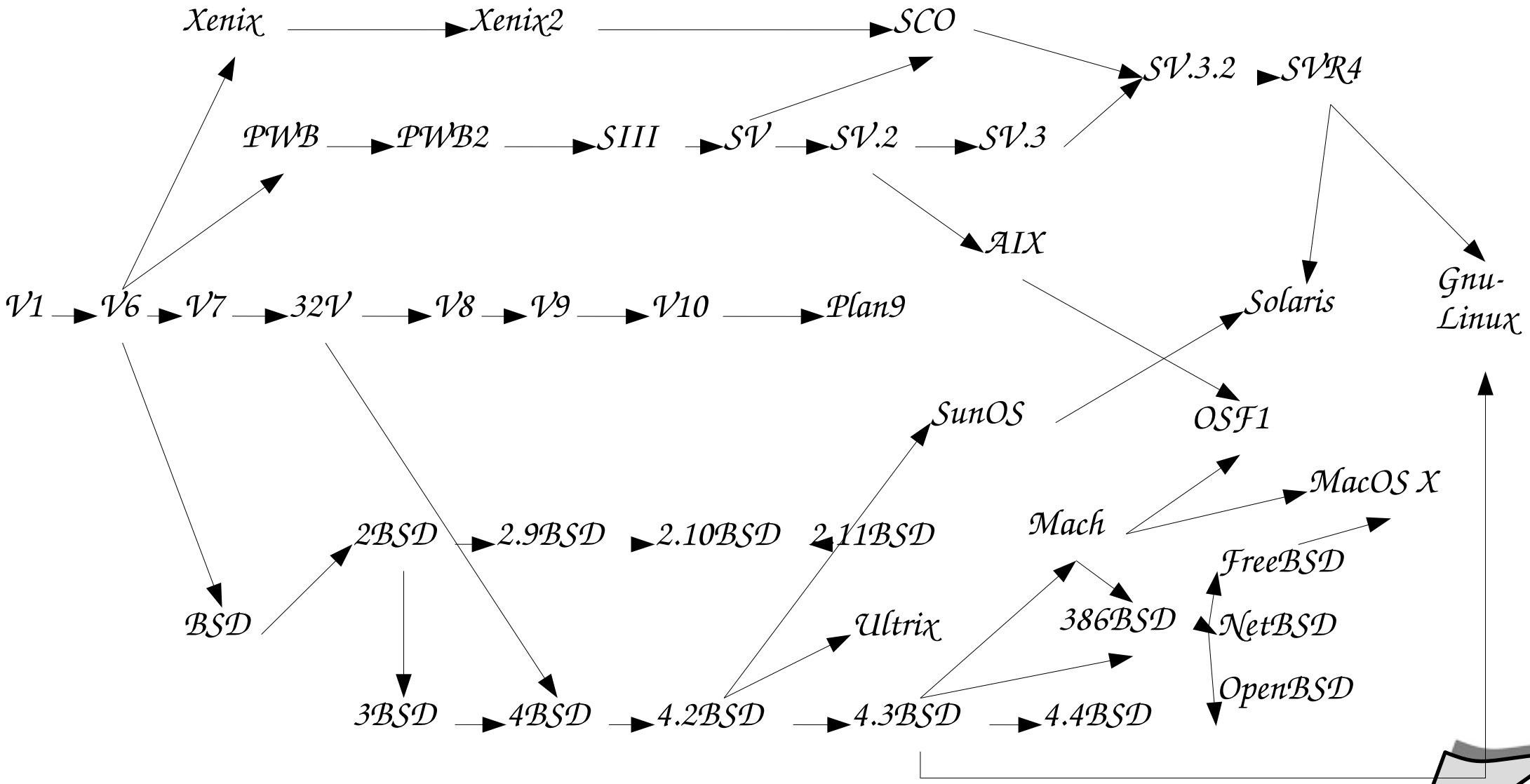
- **Compatible Time-Sharing System (CTSS) (1962)**
  - introdusse il concetto di multiprogrammazione
  - introdusse il concetto di time-sharing
  - Codice sorgente rilasciato nel settembre 2004: http://www.piercefuller.com/library/ctss.html?id=ctss
- **Multics (1965)**
  - introduzione del concetto di processo
- **Unix (1970)**
  - derivato da CTSS e da Multics
  - sviluppato inizialmente ai Bell-labs su un PDP-7

Unix – Un po' di storia

- **La storia di UNIX – in breve**
  - portato dal PDP-7 al PDP-11
    (1ª volta che un S.O viene utilizzato in due architetture diverse)
  - riscritto in linguaggio C per renderlo portabile
    (anche questa una 1ª volta, visto che i S.O. venivano scritti in assembly)
  - Inizialmente, veniva usato solo all'interno di Bell Labs
  - Nel 1974, viene pubblicato un articolo
    - licenze proprietarie
    - licenze “libere” alle università
  - Due varianti
    - Bell Labs
    - BSD (Berkeley Software Distribution, ora OpenBSD e FreeBSD)

teoria dell' universo pulsante

Unix – Un po' di storia



Generazione 4 (1980-) - Personal computer

- **Ancora una frase celebre**
  - Non c'è ragione per cui qualcuno possa volere un computer a casa sua
    (Ken Olson, fondatore di Digital, 1977)
- **Punti chiave**
  - I personal computer sono dedicati a singoli utenti
  - L'obiettivo primario diventa la facilità d'uso
  - Essendo dedicati a singoli utenti, i sistemi operativi per PC sono in generale più semplici
  - Interfacce utente (ma NON fanno parte del S.O.)
  - Grave errore: sottovalutare la sicurezza

Sistemi paralleli

- **Definizione - *Sistema parallelo***
  - un singolo elaboratore che possiede più unità di elaborazione
- **Tassonomia basata sulla struttura**
  - *SIMD - Single Instruction, Multiple Data*
    - Le CPU eseguono all'unisono lo stesso programma su dati diversi
  - *MIMD - Multiple Instruction, Multiple Data*
    - Le CPU eseguono programmi differenti su dati differenti
- **Tassonomia basata sulla dimensione**
  - A seconda del numero (e della potenza) dei processori si suddividono in
    - *sistemi a basso parallelismo*
      pochi processori in genere molto potenti
    - *sistemi massicciamente paralleli*
      gran numero di processori, che possono avere anche potenza non elevata

REGISTRATE 2022-02-25 10:33 - 13:00

Abbiamo visto tante funzionalità che possono avere i S.O.i.
I S.O.i si riconoscono dall'interfaccia delle systemcall (dalla loro API).
Abbiamo visto che ci sono delle funzionalità che ci possono ESSERE o NON ESSERE e che se ci sono o non ci sono si evince dal fatto se l'interfaccia delle systemcall PREVEDE o meno FUNZIONALITÀ chiamate SPECIFICHE per queste funzioni. Se manca tutta la parte (di systemcall) per definire chi è l'utente del processo corrente, cambiare l'utente del processo corrente il sistema non sarà MULTIUSER ma sarà pensato per eseguire programmi senza protezione tra utenti. Oltre alle "cose classiche" altre funzionalità del S.O. che possono esserci o non esserci e queste caratterizzano S.O. per USI SPECIFICI, ad esempio i S.O. possono supportare o no il PARALLELISMO. Cosa è il PARALLELISMO? È la capacità di fare più operazioni contemporaneamente. Un processore "standard, normale" un Core FA UNA COSA SOLA IN UNITÀ DI TEMPO.
I S.O. capaci di funzionare su un singolo processore o eseguono codice del KERNEL o eseguono CODICE APPLICATIVO e passano dall'uno all'altro: o da KERNEL a USER CARICANDO UNO STATO o da USER a KERNEL perché avviene una RICHIESTA del processo o un altro evento asincrono. Come possono essere i sistemi PARALLELI? SIMD detti anche Sistemi vettoriali...

Sistemi paralleli

Perché non si fanno sistemi TIGHTLY COUPLED con 256 processori?
Più processori ci sono più ci sono processori che NELLO STESSO MOMENTO RICHIEDERANNO ACCESSO IN MEMORIA, quindi il BUS diventa COLLO di BOTTIGLIA.

- sistemi *tightly* coupled
  - Bus / memoria condivisa
  - Pochi processori / basso livello di parallelismo. Bus: collo di bottiglia

quindi sono sistemi dove i Core collaborano all'interno di un architettura che prevede il BUS di comunicazione condiviso, memoria condivisa. É questo il modello dei PERSONAL COMPUTER MULTICORE, avete 2/4/8 core che condividono il BUS, la MEMORIA.

- sistemi *loosely* coupled
  - Processori con memoria privata interconnessi da canali
  - Tanti processori / alto livello di parallelismo.

dove ogni unità di elaborazione é un processore con la SUA MEMORIA e i SUOI canali di comunicazione ed é collegato con altri processori secondo una determinata topologia.

sperimentazioni Connection machine che aveva 65.536 processori collegati tra loro con un cubo a 16 dimensioni. quindi ogni processore può parlare con un altro in non più di 16 "passi" LA CONNECTION MACHINE É FALLITA!

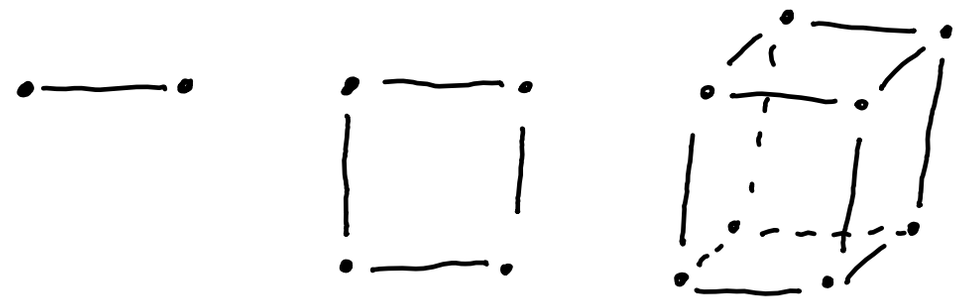
- Vantaggi dei sistemi paralleli
  - incremento delle prestazioni

Perché? ogni processore (anche se non é una valvola) ha la sua possibilità di GUASTARSI, l'altro fattore é che tutti gli algoritmi che andiamo a scrivere sono fatti per funzionare "lì" sopra. Immaginate un problema sequenziale imperativo standard.

← andrà piano come su 1 singolo Processore.

REGISTRAZIONI 18:40

(2) Mentre invece ci sono state sperimentazioni per esempio la CONNECTION MACHINE una macchina che aveva 65.536 Processori collegati tra loro con una topologia a cubo. Che cos'é un CUBO?
"Non si capisce un cubo ☺" (disegna alla lavagna...)

Cubo a  e poi potete continuare!

dimensione 1 2 3 ... N

esistono cubi a tutte le dimensioni. Per fare il cubo a dimensione N+1 prendete 2 cubi a dimensione N e collegate tra loro i nodi corrispondenti. Ciò é correlato ai numeri binari. Il cubo a dim. 1 rappresenta 0 e 1. Poi avete due copie e il primo bit rappresenta a quale copia accedere e il secondo é il bit all'interno del cubo precedente. (quello meno significativo)

Tutti i processori vengono gestiti nello stesso modo e "NON C'È UN PROCESSORE PIÙ UGUALE DEGLI ALTRI" citando Orwell più diffusi

- **Symmetric multiprocessing (SMP)**
  - Ogni processore esegue una copia identica del sistema operativo
  - Processi diversi possono essere eseguiti contemporaneamente
  - Molti sistemi operativi moderni supportano SMP
- **Asymmetric multiprocessing**
  - Ogni processore è assegnato ad un compito specifico; un processore master gestisce l'allocazione del lavoro ai processori slave
  - Più comune in sistemi estremamente grandi

Se ci sono più processori che vogliono eseguire il Kernel questi processori hanno strutture dati in comune, e quindi occorre fare in modo che si coordinino nell'accedere alle strutture dati e (ripasso dal I sem) siccome i vari Core hanno disabilitazione degli interrupt indipendenti occorre utilizzare altre DAVOLERIE/soluzioni per creare sistemi di mutua esclusione ed è per questo che per poter fare sistemi operativi occorre che i processori forniscano delle istruzioni consentono gli spinlock (test&set atomic swap, e su processori RISC link e load, store conditional).

(sono noti gli spin lock)

Sistemi distribuiti

- **Definizione - *Sistema distribuito***
  - Sono sistemi composti da più elaboratori indipendenti (con proprie risorse e proprio sistema operativo), collegati da una rete, appaiono come se fossero un unico sistema

Nuovi supercomputer. avete sentito Leonardo?

- **Vantaggi dei sistemi distribuiti**
  - Condivisione di risorse
  - Suddivisione di carico, incremento delle prestazioni
  - Affidabilità
  - (latenza maggiore rispetto ai sistemi paralleli)

a causa del fatto che usano una RETE invece di un "bus dedicato".

- **Sistemi operativi**
  - forniscono condivisione di file
  - forniscono la possibilità di comunicare
  - ogni computer opera indipendentemente dagli altri
- **Sistemi operativi distribuiti** → esempio nostro laboratorio (con nomi di personaggi d'opera)
  - minore autonomia tra i computer
  - dà l'impressione che un singolo sistema operativo stia controllando tutti gli elaboratori che fanno parte del sistema distribuito

un esempio il nostro laboratorio

occorre:
- condivisione efficace del File system
- database condivisi (utenti, password, gruppi, ...)
- distribuire il carico tra computer se si vuole fare elaborazione massiva/massiccia

NON SIGNIFICA VELOCE! può essere tranquillamente lento dipende dal problema che dovete risolvere

- **Definizione: *sistemi real-time***
  - Sono i sistemi per i quali la correttezza del risultato non dipende solamente dal suo valore ma anche dall'istante nel quale il risultato viene prodotto

real time

correttezza risultato — valore, tempo in cui viene prodotto

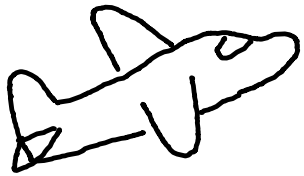
citazione: ♡

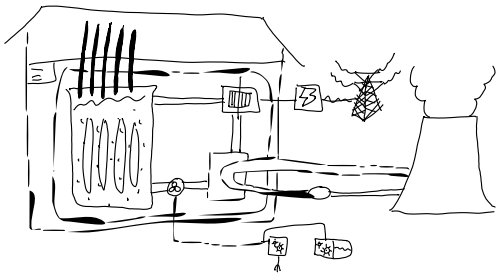
Quindi tecnicamente se io progetto un sistema per dire "devo trovare la risposta alla vita, l'universo ed ogni cosa ma non ci deve mettere meno di 200 anni" è REAL TIME, perché se quello mi risponde a 199 anni la risposta è sbagliata perché gli avevo creato un vincolo temporale.
(Scherzo con citazione a Douglas Adams) perché ovviamente quello che interessa è avere un risultato ENTRO UN DETERMINATO TEMPO. L'importante è che sia deterministico non ci interessa che sia veloce, sia veloce abbastanza per il problema da risolvere.

Infatti abbiamo due tipi di sistemi REAL TIME

Sistemi real-time

- **I sistemi real-time si dividono in:**
  - *hard real-time*:
    - se il mancato rispetto dei vincoli temporali può avere effetti catastrofici
    - e.g. controllo assetto velivoli, controllo centrali nucleari, apparecchiature per terapia intensiva





Chernobyl

  - *soft real-time*:
    - se si hanno solamente disagi o disservizi
    - e.g. programmi interattivi

→ best effort

- **N.B.**
  - real-time non significa necessariamente esecuzione veloce