Capitolo 5 – Parte 1
Le infrastrutture software

Funzioni del sistema operativo

- Rendere utilizzabili le risorse fisiche presenti nel sistema informatico:
  - garantire la **correttezza** e la **precisione** nell'elaborazione e nella trasmissione dell'informazione;
  - consentire all'utente di superare il problema della localizzazione delle risorse sfruttando opportunamente la presenza di una rete che permetta di accedere alle applicazioni da **ogni luogo** e in **ogni momento** (**anywhere**, **anytime**);
  - garantire il massimo livello di **affidabilità**, **disponibilità** e **sicurezza** dei sistemi;
  - assicurare la **privatezza** dei dati;
  - consentire la realizzazione di soluzioni aperte, che permettano **l'interoperabilità** fra dispositivi forniti da diversi produttori di hardware e di software;
  - superare i problemi legati alla **limitazione del numero di risorse** e, al tempo stesso, regolamentarne l'impiego evitando conflitti di accesso.
- Il sistema operativo può essere inteso come uno strumento che **virtualizza** le caratteristiche dell'hardware sottostante, offrendo di esso la visione di una **macchina astratta** più potente e più semplice da utilizzare di quella fisicamente disponibile.

SO: funzionalità

- **SO come GESTORE DELLE RISORSE**, controlla tutte le risorse del calcolatore e le gestisce in modo efficiente:
  - tiene traccia di chi utilizza la risorse
  - accetta e soddisfa le richieste di utilizzo di una risorsa
  - fa da mediatore fra le risorse che risultano in conflitto.
- **SO come MACCHINA ESTESA**:
  - costituisce la base su cui è possibile scrivere i programmi applicativi.
  - presenta all'utente una macchina estesa più facile da programmare dell'HW sottostante.

Funzioni di servizio del SO

- **Esecuzione di applicazioni**
  - caricamento del programma (istruzioni e dati) nella memoria centrale,
  - inizializzazione dei dispositivi di ingresso/uscita,
  - preparazione e gestione di altre risorse come la rete di comunicazione,
  - ...;
- **Accesso ai dispositivi di ingresso/uscita**
  - gestione dei segnali necessari per il trasferimento dei dati,
  - consente all'utente di ragionare in termini di operazioni astratte di lettura e scrittura;
- **Archiviazione di dati e programmi**
  - fornire un'organizzazione logica dei dati sotto forma di cartelle (**directory**) e file,
  - gestire le operazioni di basso livello per il relativo ingresso/uscita;
- **Controllo di accesso**
  - condivisione di risorse da parte di più utenti o applicazioni,
  - meccanismi di protezione e politiche di risoluzione degli eventuali conflitti d'uso;
- **Contabilizzazione**
  - ottimizzare il tempo di risposta dei programmi interattivi,
  - fatturare agli utenti i costi dell'impiego del sistema;
- **Gestione dei malfunzionamenti**
  - rilevare e, se possibile, di risolvere eventuali malfunzionamenti provocati da guasti hardware, o da operazioni scorrette compiute dal software applicativo.

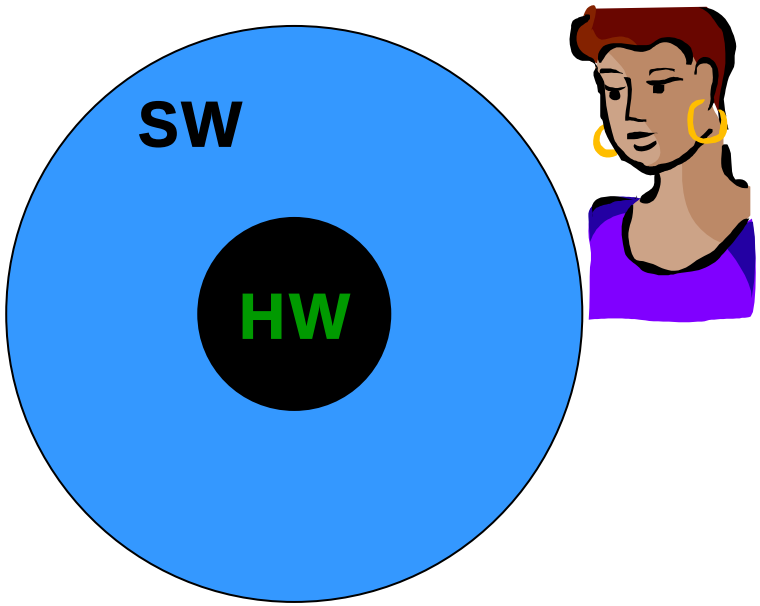
Vantaggi di un SO

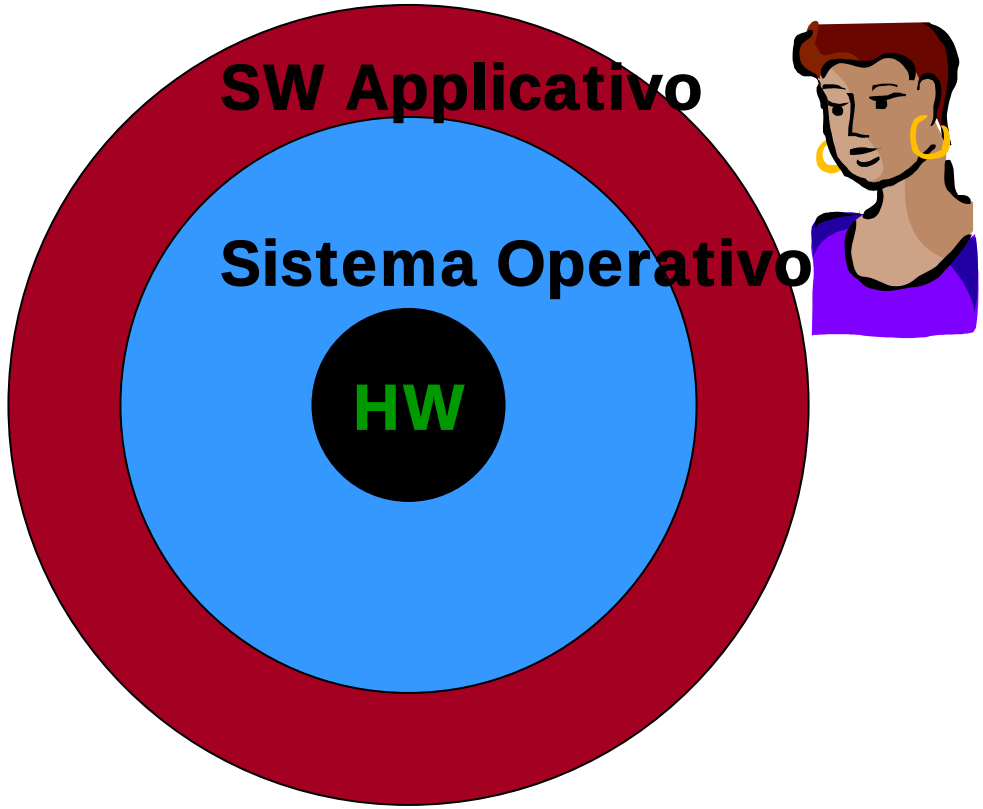
- Sono legati alla possibilità di definire modalità standard di interfaccia con i dispositivi fisici, cosicché sia possibile:
  - sviluppare programmi in modo semplice, modulare ed indipendente dallo specifico calcolatore su cui viene fatto funzionare il sistema operativo;
  - aggiornare il software di base e l'hardware in modo trasparente ai programmi applicativi e all'utente, ossia senza che vengano influenzati dall'operazione.

Visioni fornite da un SO

- **Dall'alto**: il sistema operativo fornisce all'utente un'interfaccia conveniente.
- **Dal basso**: gestisce tutti le parti di un sistema complesso, allocando in modo ordinato le diverse risorse della macchina: processori, memorie, dischi, interfacce di rete, stampanti e altre periferiche.

Il software



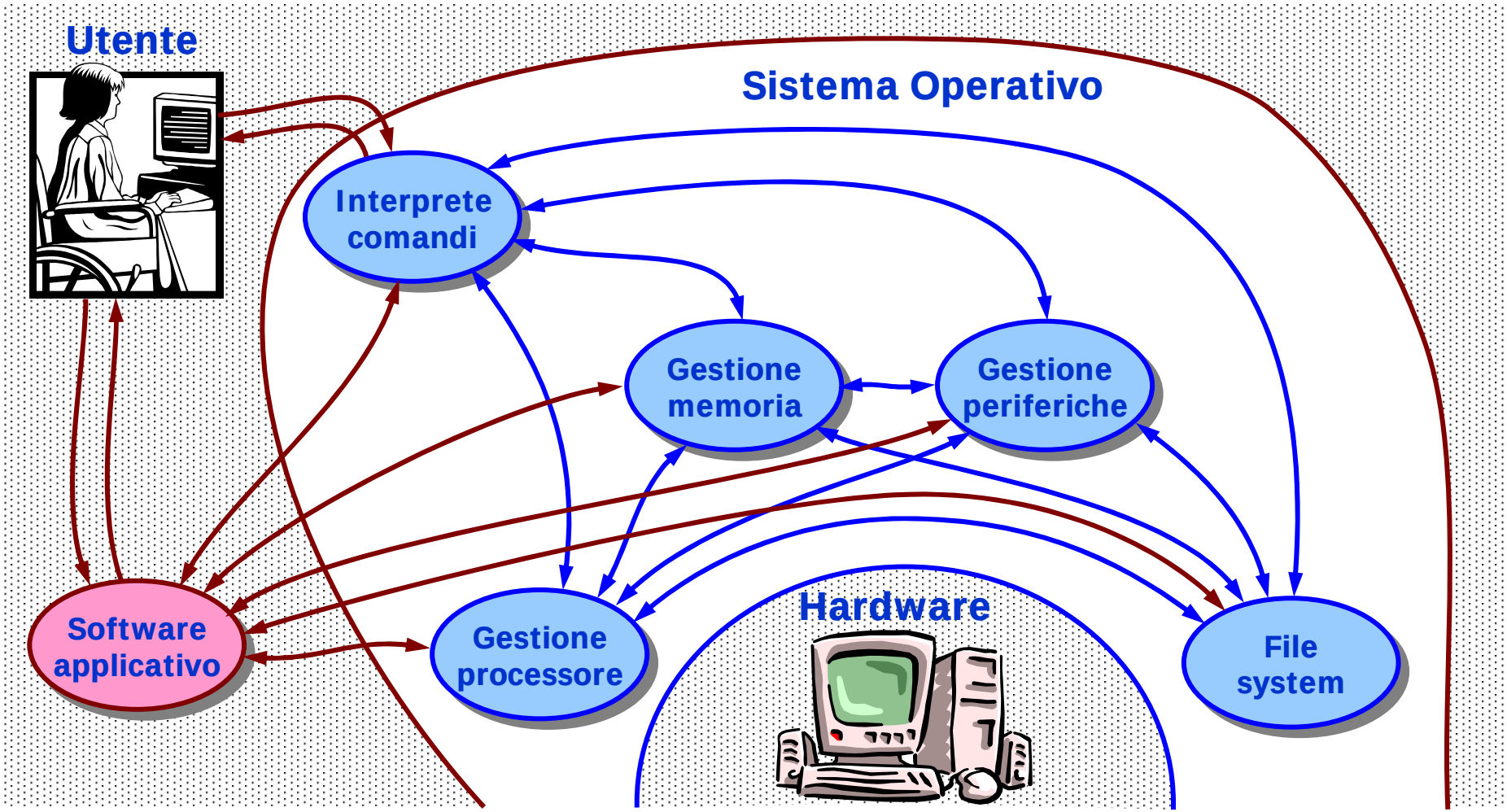


- SW= Sistema Operativo + SW applicativo
- Il S.O. come necessario **intermediario**

Elementi di un SO

- Sistema di **gestione del processore**,
  - controlla l'unità centrale di elaborazione (CPU);
  - definisce quali programmi sono da eseguire e quali compiti sono da assegnare alla CPU;
- Sistema di **gestione della memoria**,
  - controlla l'allocazione della memoria di lavoro ai diversi programmi che possono essere contemporaneamente in esecuzione;
- Sistema di **gestione delle periferiche**,
  - garantisce l'accesso ai dispositivi di ingresso/uscita,
  - maschera i dettagli di basso livello e gli eventuali conflitti che possono insorgere nel caso che diverse richieste arrivino contemporaneamente a uno stesso dispositivo;
- Sistema di **gestione dei file** (**file system**)
  - consente l'archiviazione e il reperimento dei dati sfruttando le periferiche che costituiscono la memoria di massa;
- Sistema di **gestione degli utenti e dei relativi comandi** (**interprete comandi**),
  - interfaccia diretta con gli utenti,
  - permette agli utenti di accedere in maniera semplice e intuitiva alle funzionalità disponibili.

Elementi di un SO



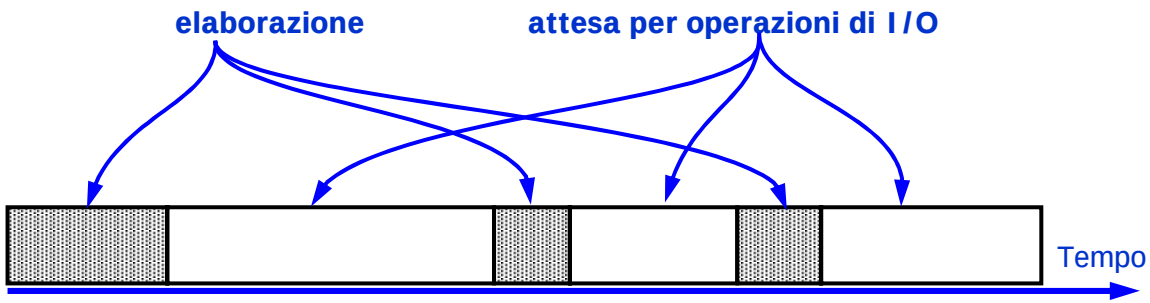
SO vs applicazioni

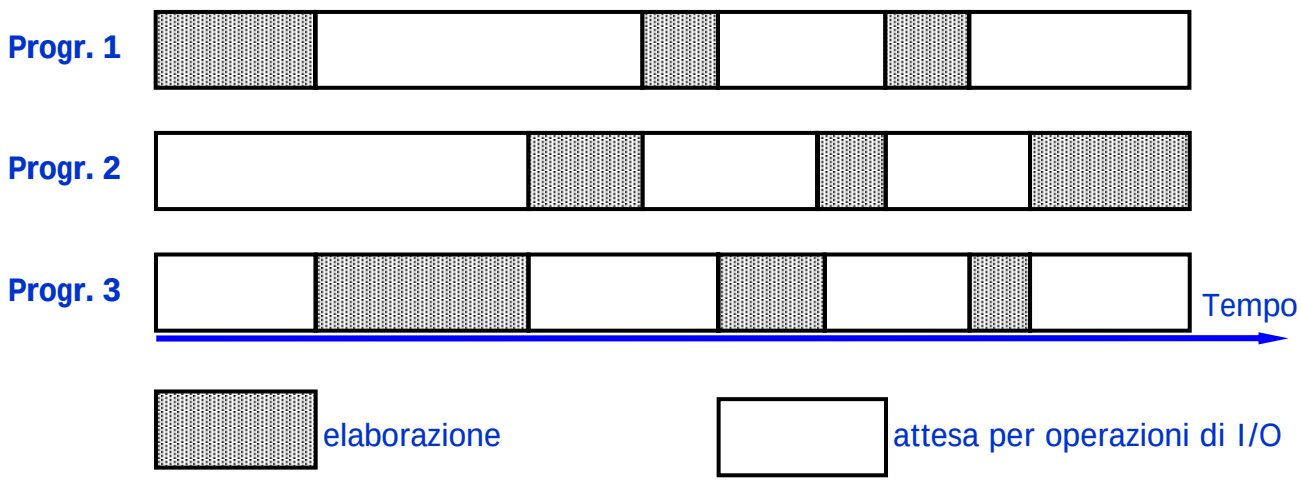
- Programmi applicativi
  - hanno accesso a un insieme ridotto di risorse;
  - possono utilizzare solo un sottoinsieme delle istruzioni del processore (esecuzione in **modalità utente**);
  - non possono decidere autonomamente quando e come avere accesso alle risorse del sistema (richiedono al sistema operativo l'esecuzione di alcuni servizi);
  - ...
- Sistema operativo
  - ha accesso a tutte le risorse;
  - può utilizzare tutte le istruzioni del processore (esecuzione in **modalità supervisore**);
  - stabilisce in che ordine e come le richieste che riceve devono essere soddisfatte;
  - ...

Evoluzione dei sistemi operativi

- Seconda metà degli anni ’40
  - non c’era un sistema operativo: il programmatore interagiva direttamente con i dispositivi fisici,
  - introdurre nel software applicativo anche le funzionalità di gestione dell’hardware.
  - esecuzione di un solo programma alla volta
  - problemi da risolvere:
    - evoluzione dell’interfaccia utente
    - miglioramento dell’efficienza di gestione delle risorse disponibili
- Sistemi a lotti (**batch**),
  - impiego di un componente software, detto **monitor**, in grado di automatizzare l’avviamento e l’esecuzione di programma;
  - apposito linguaggio, detto **Job Control Language** (**JCL**);
  - le operazioni di ingresso/uscita vengono eseguite tramite file;
  - ottimizzano lo sfruttamento delle risorse di elaborazione del sistema con un’accurata distribuzione del tempo di calcolo tra i vari programmi da eseguire
  - svantaggio principale: **tempo di latenza** che intercorre fra il momento in cui un job è sottoposto al sistema e il momento in cui vengono presentati i risultati dell’elaborazione di quel job.
- Aggiunta di altre caratteristiche (sistemi **multiprogrammati** in cui, in ogni istante, la memoria centrale può contenere più di un programma applicativo):
  - **interattività** con un utente umano con tempi di risposta accettabili;
  - accurata **protezione** della memoria;
  - **temporizzazione** dell’esecuzione (evitare che un programma monopolizzi il sistema);
  - gestione delle attività di **ingresso/uscita**;
  - **maggiore sfruttamento** (anche in parallelo da parte di diversi programmi), delle risorse disponibili nel sistema.

Multiprogrammazione





Multiprogrammazione

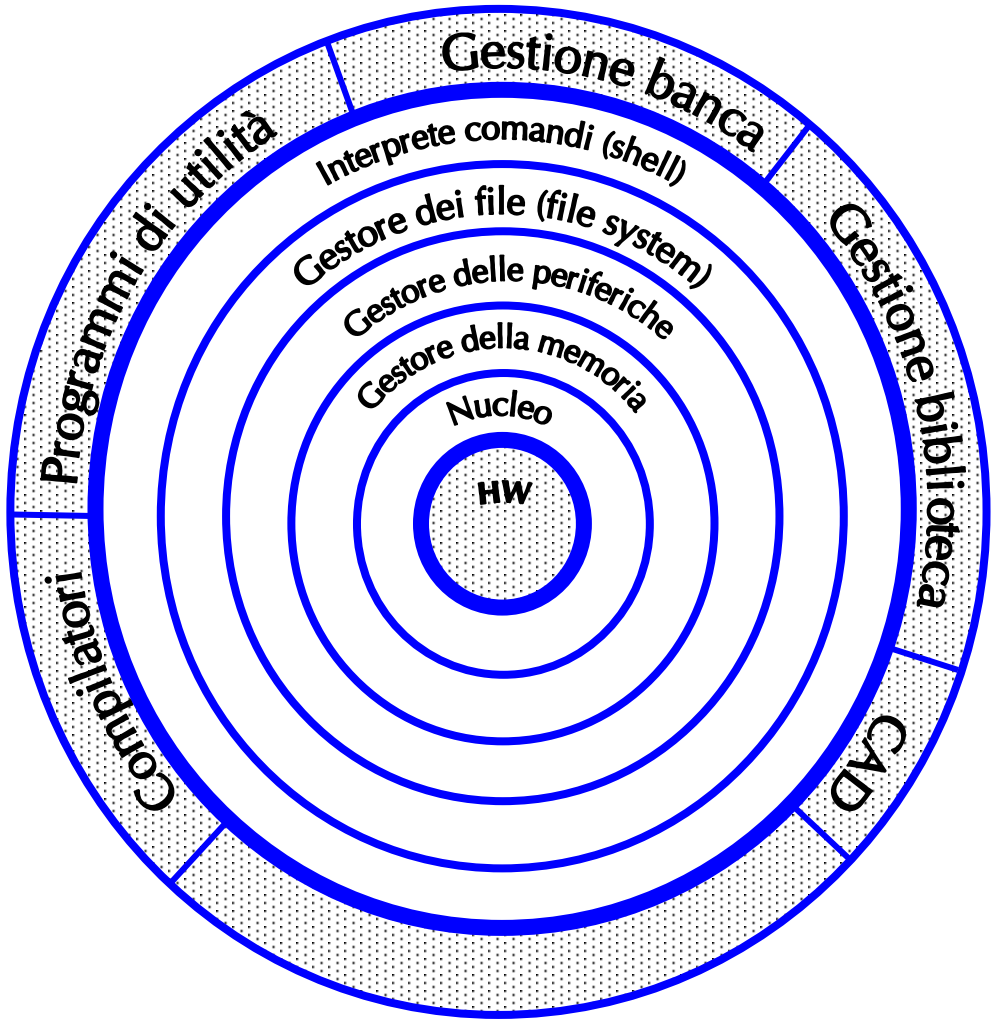
- Nel sistema sono presenti diversi programmi, ognuno con un proprio tempo di elaborazione e propri tempi di attesa per le operazioni di ingresso/uscita.
- Per evitare che la CPU venga utilizzata in modo esclusivo (o per troppo tempo) da parte di un solo programma, il tempo viene idealmente suddiviso in unità elementari, dette **quanti**, da assegnare secondo opportune politiche a tutti i programmi.
- **Round-robin**: assegnare a rotazione la disponibilità di un quanto di tempo della CPU ai vari programmi presenti contemporaneamente in memoria.
- La durata del quanto di tempo incide significativamente sia sulle prestazioni del sistema che sull'efficacia del quasi parallelismo, che tende a scomparire se la durata diviene eccessiva e degrada nella sequenzializzazione dei programmi. D'altra parte, pur migliorando in generale le proprietà di parallelismo la scelta di un valore molto piccolo può comportare un degrado delle prestazioni complessive del sistema, qualora il tempo di commutazione fra programmi sia dello stesso ordine della durata del quanto di tempo (un valore tipico per il sistema operativo Unix è 100 ms).

Processo vs programma

- **Programma**:
  entità statica composta dal codice eseguibile dal processore.
- **Processo**:
  entità dinamica che corrisponde al programma in esecuzione, composto da:
  - codice (il programma);
  - dati (quelli che servono per l'esecuzione del programma);
  - stato (a che punto dell'esecuzione ci si trova, cosa c'è nei registri, ...).

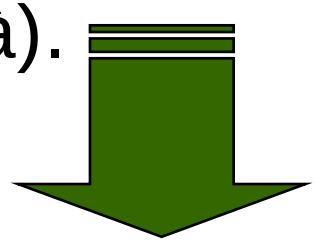
Organizzazione di un SO

- Gerarchia di “macchine virtuali”
- La visione della macchina virtuale a livello **n** è quella fornita dall’HW e dagli strati del SO fino all’ennesimo (incluso)



Organizzazione a “strati”

- Ogni macchina virtuale è un insieme di programmi che realizza delle funzionalità che utilizzano i servizi forniti a livello inferiore.
- Ogni macchina virtuale ha il compito di gestire risorse specifiche di sistema regolandone l’uso e mascherandone i limiti.
- I **meccanismi** che garantiscono la correttezza logica sono separati dalle **politiche** di gestione (maggiore flessibilità).



ogni “strato” risolve un problema specifico

La gestione dei processi

Elaborazione parallela

- Il concetto di **elaborazione parallela** si riferisce specificamente:
  - **ai dati**, che potrebbero essere trattati parallelamente;
  - **alle istruzioni**, che potrebbero essere eseguite contemporaneamente nel caso di operazioni indipendenti che trattano dati distinti;
  - **ai programmi**, che potrebbero essere in esecuzione nello stesso tempo.
- Il parallelismo a livello di dati e di istruzioni è possibile solo con l'impiego di architetture di elaborazione parallela, basate sulla presenza di più unità di elaborazione in grado di eseguire istruzioni in modo concorrente ma anche, per esempio, di adeguati linguaggi di programmazione.
- Il parallelismo a livello di programma ricade nell'ambito dei sistemi operativi.
- Le condizioni che un sistema operativo deve soddisfare sono:
  - **efficienza**: occorre assicurare un impiego ottimale di tutte le risorse;
  - **interattività**: il tempo di risposta è un fattore determinante per definire la qualità del sistema operativo;
  - **sincronizzazione/cooperazione**: gestire la conseguente presenza di più **agenti** in grado di operare sul sistema in modo concorrente, al fine di evitare possibili malfunzionamenti (per esempio il blocco di un programma a causa di operazioni errate compiute da altri).

Linguaggio di Programmazione

- La notazione con cui è possibile descrivere gli algoritmi.
- Programma: è la rappresentazione di un algoritmo in un particolare linguaggio di programmazione.
- Ogni linguaggio di programmazione dispone di un insieme di “parole chiave” (keywords)
- Ogni linguaggio è caratterizzato da una sintassi e da una semantica

Il linguaggio macchina

- Il linguaggio macchina è direttamente eseguibile dall'elaboratore, senza nessuna traduzione.
  - Istruzioni ed operandi relativi al programma in esecuzione sono caricati in memoria e quindi sono memorizzati in forma *binaria*.
  - Vincolo: conoscenza dei metodi di rappresentazione delle informazioni utilizzati.

- Istruzione: carica nell'accumulatore

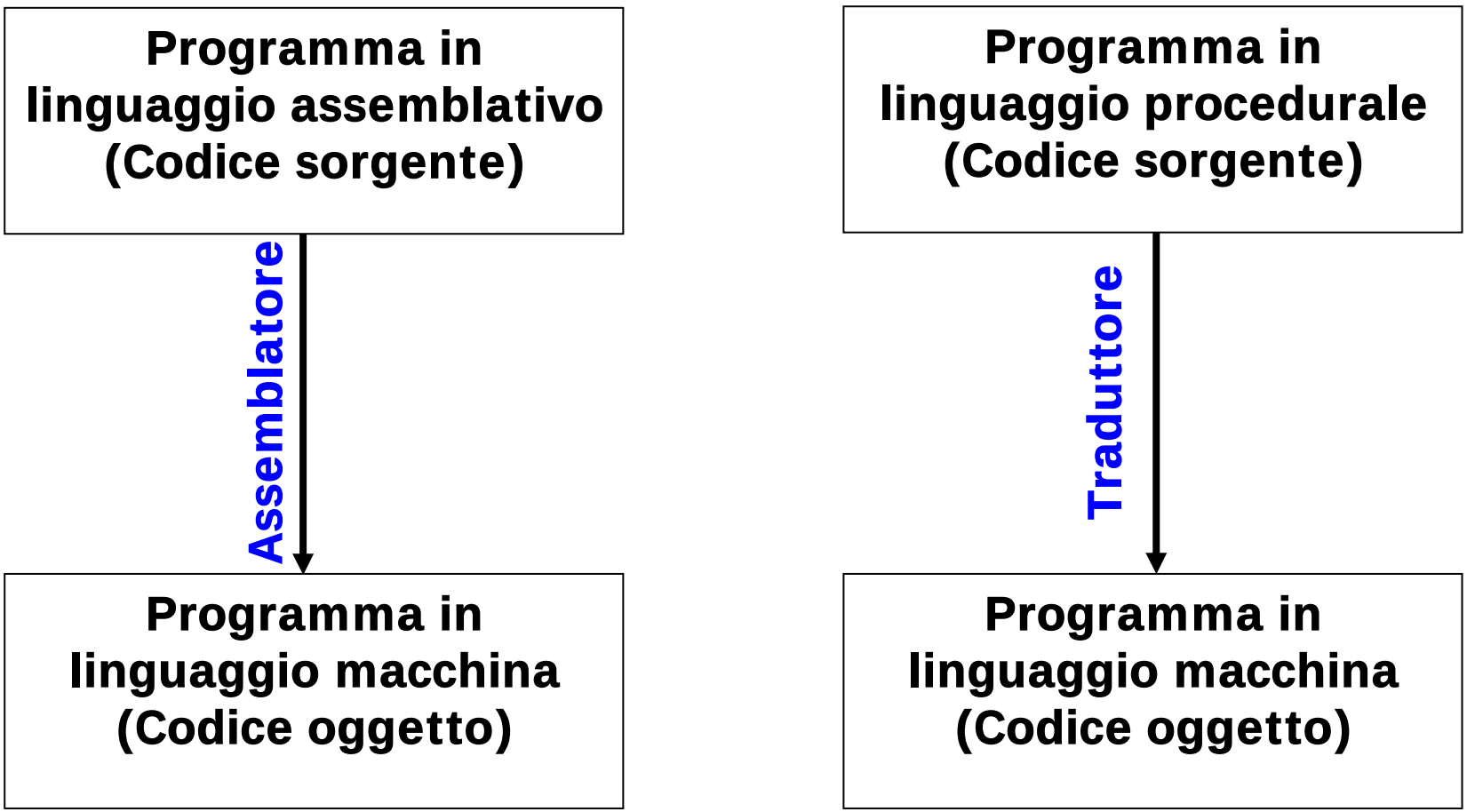
Il linguaggio Assembler

- Le istruzioni corrispondono univocamente a quelle macchina, ma vengono espresse tramite nomi simbolici (parole chiave).
- Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina (**assemblatore**).
- Vincolo: necessità di conoscere in dettaglio le caratteristiche della macchina (registri, dimensione dei dati, set di istruzioni)
- Anche semplici algoritmi implicano la specifica di molte istruzioni

I linguaggi di alto livello

- Sono i linguaggi di terza generazione. Le istruzioni esprimono una serie di azioni. Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina (**traduttore**)
- Il programmatore può astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico
- Sono indipendenti dalla macchina (astrazione)

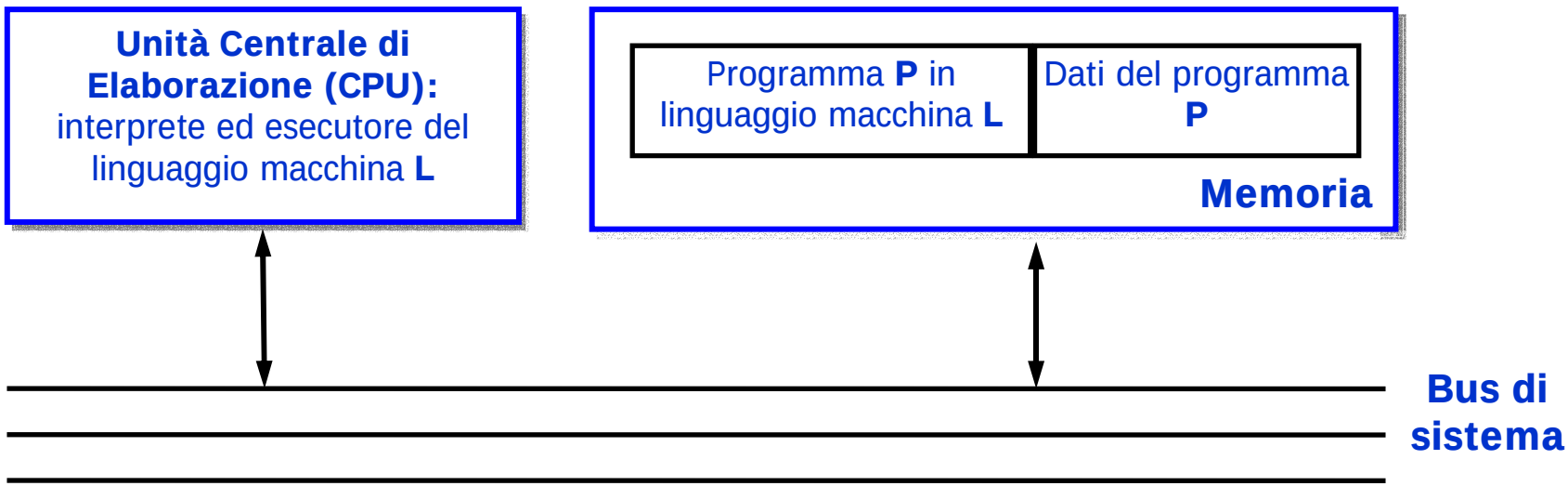
Linguaggi di II/III generazione



Sviluppo di un Programma

- I **traduttori** sono programmi particolari che provvedono a convertire il codice di programmi scritti in un dato linguaggio di programmazione (sorgenti), nella corrispondente rappresentazione in linguaggio macchina (eseguibili)

CPU come interprete del suo linguaggio macchina

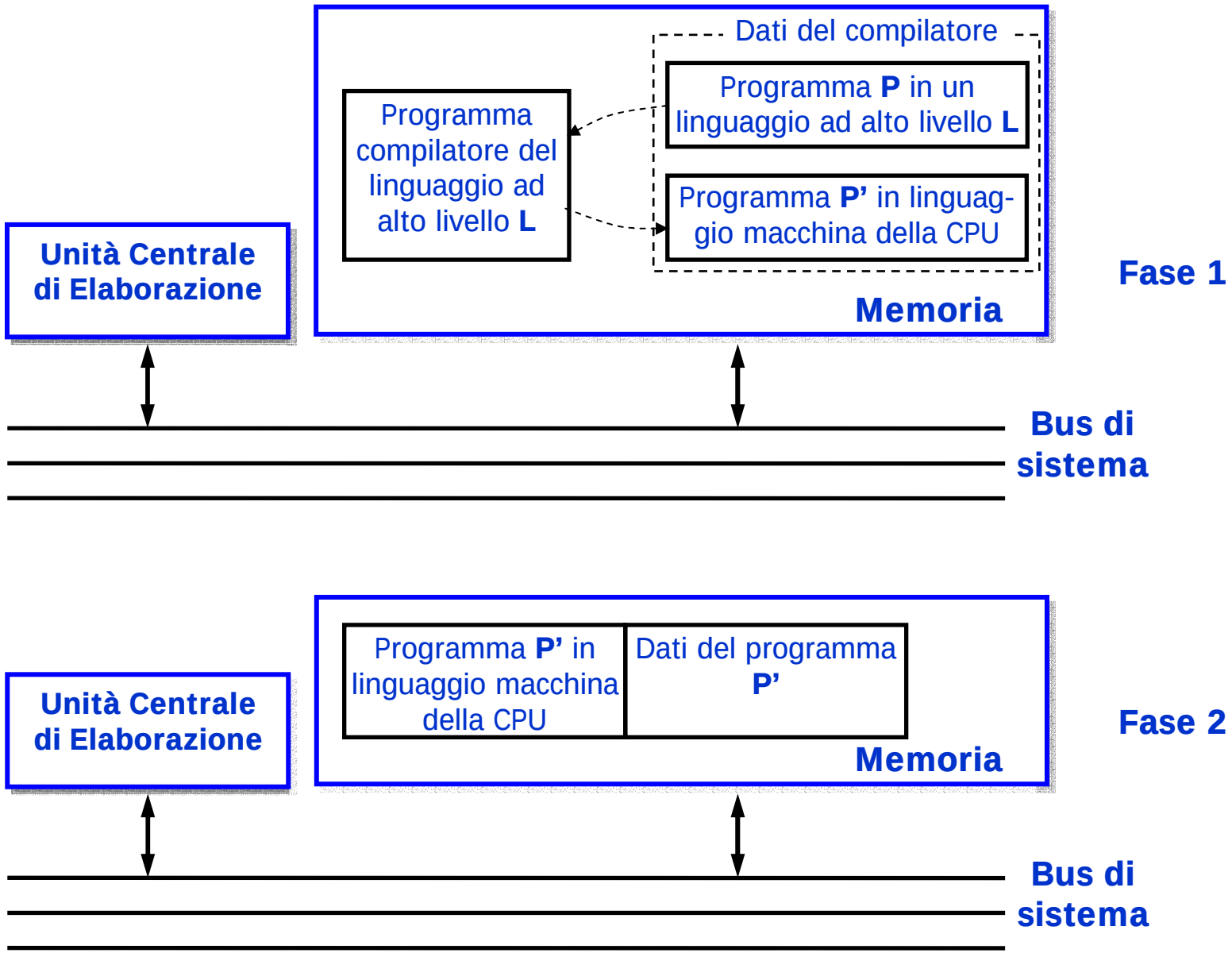


Due tipi di traduttori

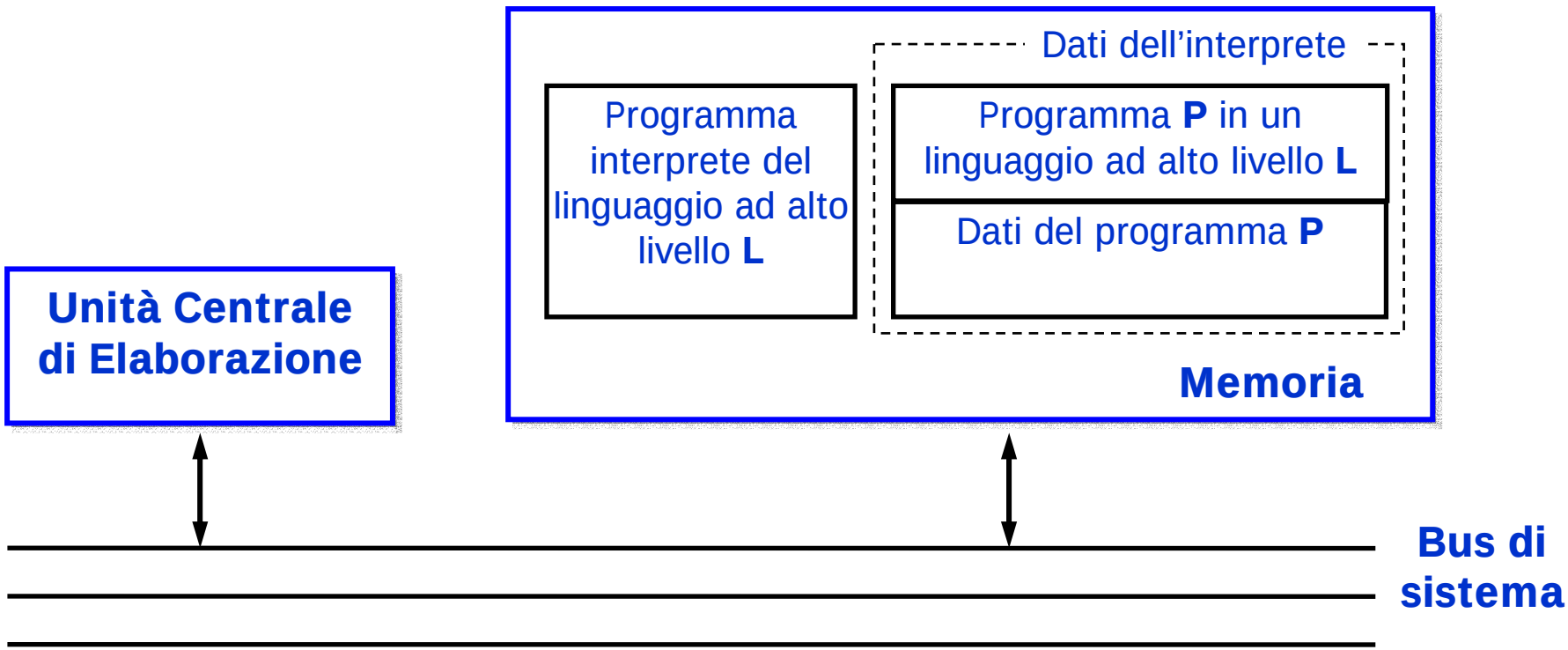
- **Compilatori**
  - Accettano in ingresso l'intero programma e producono in uscita la rappresentazione dell'intero programma in linguaggio macchina.

- **Interpreti**
  - Traducono ed eseguono direttamente ciascuna istruzione del programma sorgente, istruzione per istruzione.

Compilatore



Interprete


Dati dell'interprete
Programma interprete del linguaggio ad alto livello **L**
Programma **P** in un linguaggio ad alto livello **L**
Dati del programma **P**
Memoria
Unità Centrale di Elaborazione
Bus di sistema

Due tipi di traduttori /2

- Compilatori
  - Per ogni programma da tradurre, lo schema viene percorso una volta sola prima dell'esecuzione.
- Interpreti
  - Lo schema viene attraversato tante volte quante sono le istruzioni che compongono il programma; ad ogni attivazione dell'interprete su una particolare istruzione, segue l'esecuzione dell'istruzione stessa.
- L'esecuzione di un programma compilato è più veloce dell'esecuzione di un programma interpretato.

Interprete vs compilatore

- Quale delle due soluzioni è la migliore?
  - nella **compilazione** la traduzione viene effettuata una sola volta, in una fase che precede l'esecuzione; si ottengono in genere **migliori prestazioni** nell'esecuzione;
  - nella **compilazione** si possono attuare processi di **ottimizzazione** dell'eseguibile, dato che il compilatore opera sull'intero programma e non istruzione per istruzione come l'interprete;
  - per ogni modifica del programma sorgente, la compilazione deve essere ripetuta completamente per rigenerare il codice eseguibile; **l'interprete** consente invece di eseguire il programma non appena il codice sorgente sia stato aggiornato, per cui il costo di eventuali modifiche al programma è pressoché nullo.
- **Compilazione**
  - applicazioni più veloci
  - maggior lavoro nel processo di messa a punto e manutenzione
  - OK per i prodotti commerciali a larga diffusione.
- **Interpretazione**
  - consente tempi di sviluppo più contenuti,
  - produce programmi meno efficienti;
  - OK in fase di prototipazione dei programmi che, una volta ultimati, venivano compilati prima del rilascio commerciale.

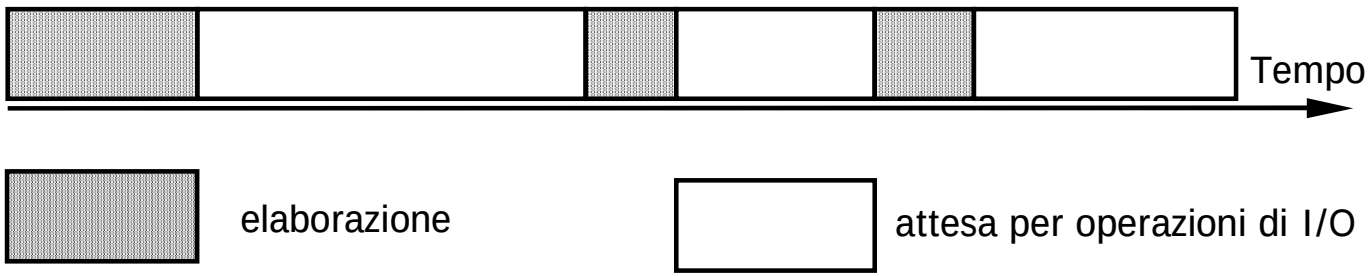
Interprete vs compilatore

- Considerare le applicazioni che debbono essere eseguite:
  - **I/O bound**:
    - programmi che eseguono molte operazioni di ingresso/uscita intervallate da brevi periodi di elaborazione;
    - convenientemente eseguiti da un interprete senza penalizzazioni nell'efficienza (il tempo di ingresso/uscita è indipendente dalle modalità di esecuzione del programma),
    - appartengono a questa categoria la maggioranza dei programmi gestionali.
  - **CPU bound**:
    - programmi che eseguono poche operazioni di ingresso/uscita rispetto alla mole delle elaborazioni effettuate.
    - sono programmi prevalentemente di tipo scientifico o ingegneristico,
    - sono scarsamente efficienti se eseguiti da un interprete e la loro compilazione risulta pressoché indispensabile.
- Considerazioni commerciali
  - la distribuzione del codice eseguibile garantisce al programmatore che l'acquirente non sarà facilmente in grado di modificare e/o riutilizzare parti del programma sorgente, soggette ovviamente a vincoli di proprietà intellettuale.
- Considerazioni tecnologiche
  - la diffusione di Internet e la disponibilità di calcolatori ad alte prestazioni e a costi contenuti hanno portato a una rinascita dell'uso di interpreti software (portabilità e indipendenza dall'hardware sul quale vengono eseguite le applicazioni).

SO in time sharing

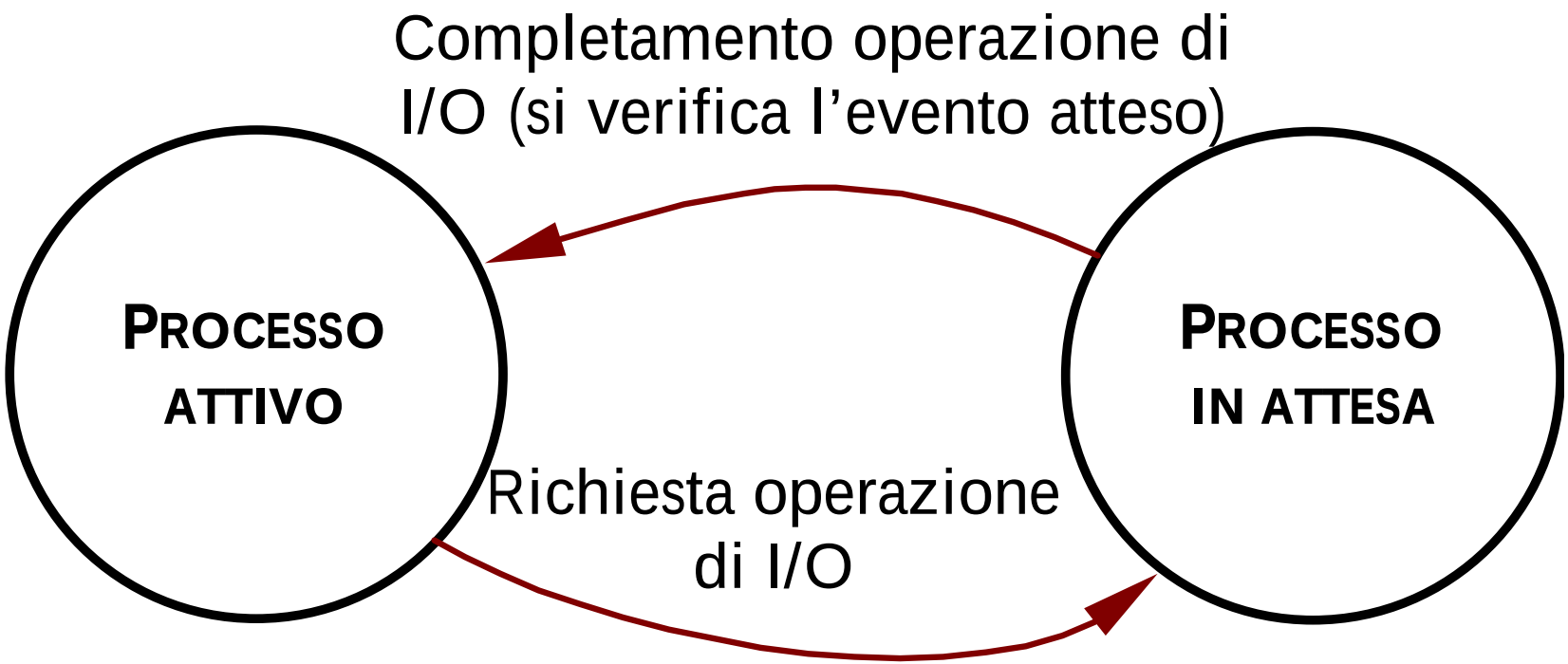
- Permette la condivisione della CPU tra più processi interattivi
- Il tempo di esecuzione del processore è condiviso tra più utenti
- Ogni processo in esecuzione ha a disposizione un quanto di tempo di utilizzo della CPU, al termine del quale viene sospeso per lasciare il posto ad un altro processo in attesa di esecuzione

Esecuzione di un processo



- Un processo utente può effettivamente essere in esecuzione sulla CPU
- Ogni operazione di I/O consiste in una chiamata al sistema operativo e quindi in una sospensione del processo utente per l'esecuzione dell'operazione di I/O da parte del kernel

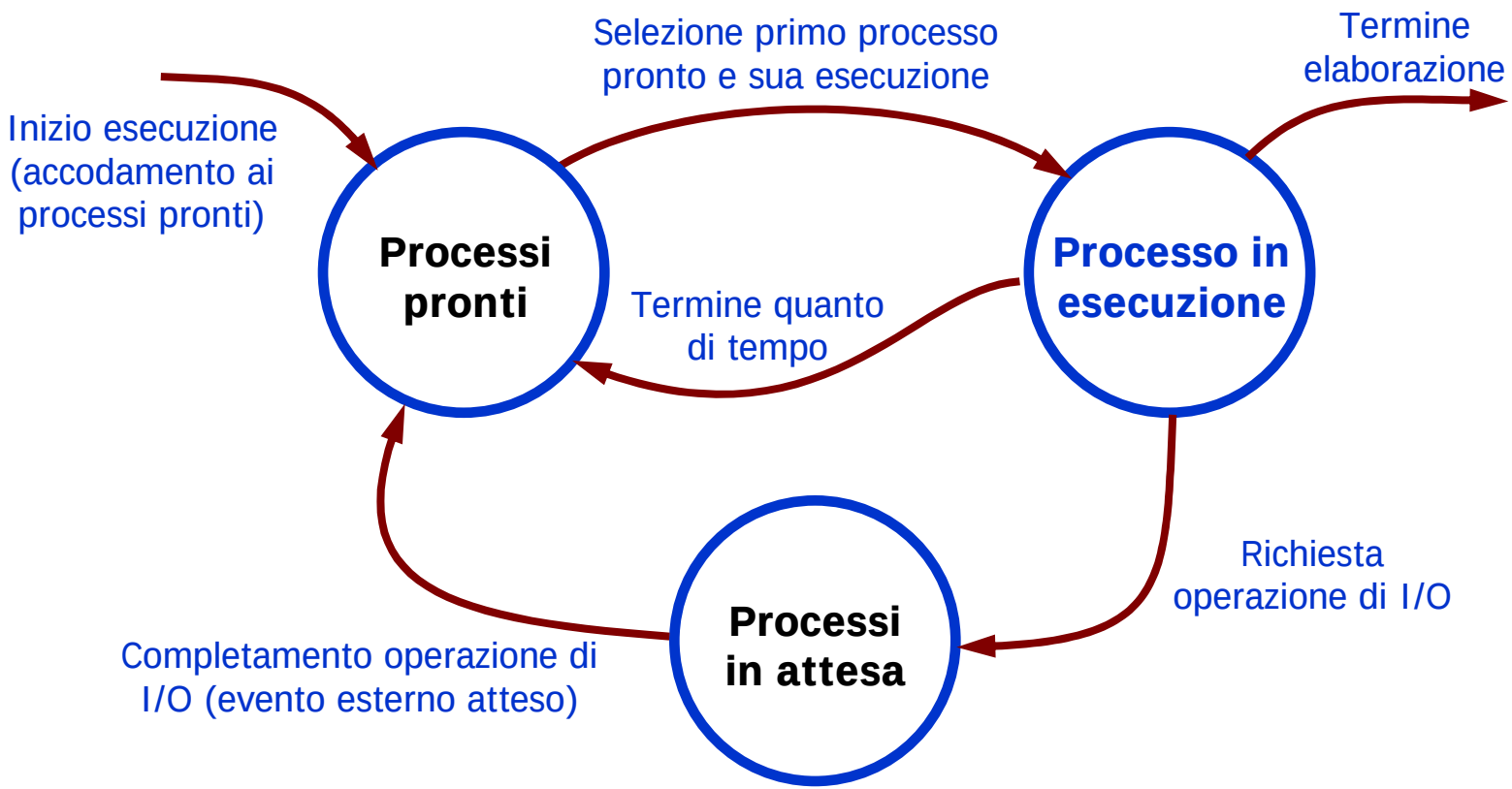
Stati di un processo



Processi non in esecuzione

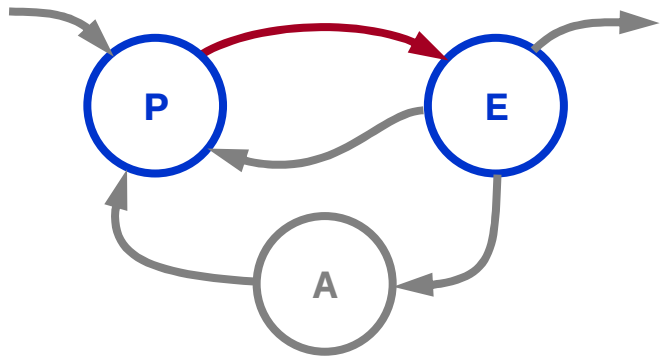
- Processi in attesa di un evento esterno (ad esempio I/O)
- Processi pronti ad essere eseguiti in attesa della CPU
- Si tratta di due stati diversi: PRONTO e ATTESA realizzati con due code diverse

Diagramma a tre stati

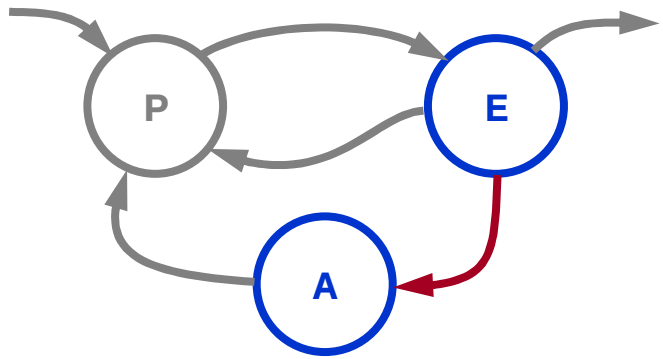


Transizioni di stato

- **Pronto → Esecuzione**
  - Il SO stabilisce quale dei processi “pronti” debba essere mandato in “esecuzione”.
  - La scelta è fatta dall’algoritmo di scheduling che deve bilanciare **efficienza** e **fairness.**

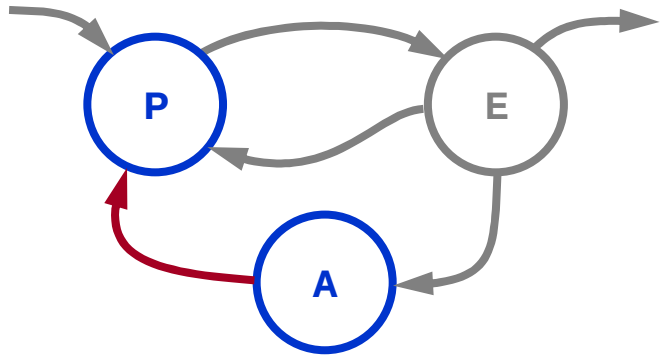


- **Esecuzione → Attesa**
  - il processo chiede delle risorse che non sono disponibili o attende un evento
  - il SO salva tutte le informazioni necessarie a riprendere l’esecuzione e l’informazione relativa all’evento atteso nella tabella dei processi

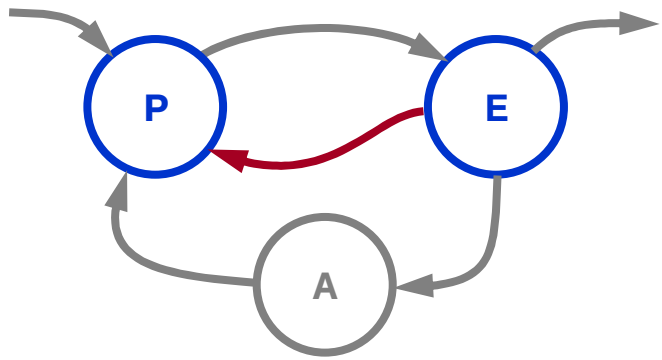


Transizioni di stato

- **Attesa → Pronto**
  - Si verifica l'evento atteso dal processo e il SO sposta quel processo nella coda dei processi pronti.



- **Esecuzione → Pronto**
  - Termina il quanto di tempo e il processo in "esecuzione" lascia spazio a un altro processo "pronto".
  - Il SO salva (nella tabella dei processi) tutte le informazioni per riprendere l'esecuzione del processo dal punto in cui viene interrotta.
  - Contemporaneamente un altro processo passa da "pronto" a "esecuzione".



Una possibile evoluzione

1. Il processo viene creato e viene posto nella coda dei processi pronti;
2. il primo processo tra i processi pronti viene posto in esecuzione;
3. il processo in esecuzione dispone delle risorse del sistema fino a
   a. il termine del quanto di tempo
      - il nucleo interrompe il processo e lo mette in coda ai processi pronti;
      - quando arriva in cima alla coda dei processi pronti, il processo torna in stato in esecuzione, proseguendo con l'elaborazione dell'istruzione successiva a quella su cui era stato interrotto;
   b. la richiesta di un'operazione di ingresso/uscita
      - il nucleo sposta il processo attivo nello stato di attesa;
      - quando l'operazione di ingresso/uscita si completa il processo può proseguire l'elaborazione e viene messo in fondo alla coda dei processi in pronti e prosegue come nel punto precedente;
   c. il termine delle propria esecuzione (istruzione finale)
      - il processo viene eliminato e rimosso dall'elenco dei processi esistenti;

   in ogni caso il nucleo provvede a sostituirlo con il primo dei processi pronti.

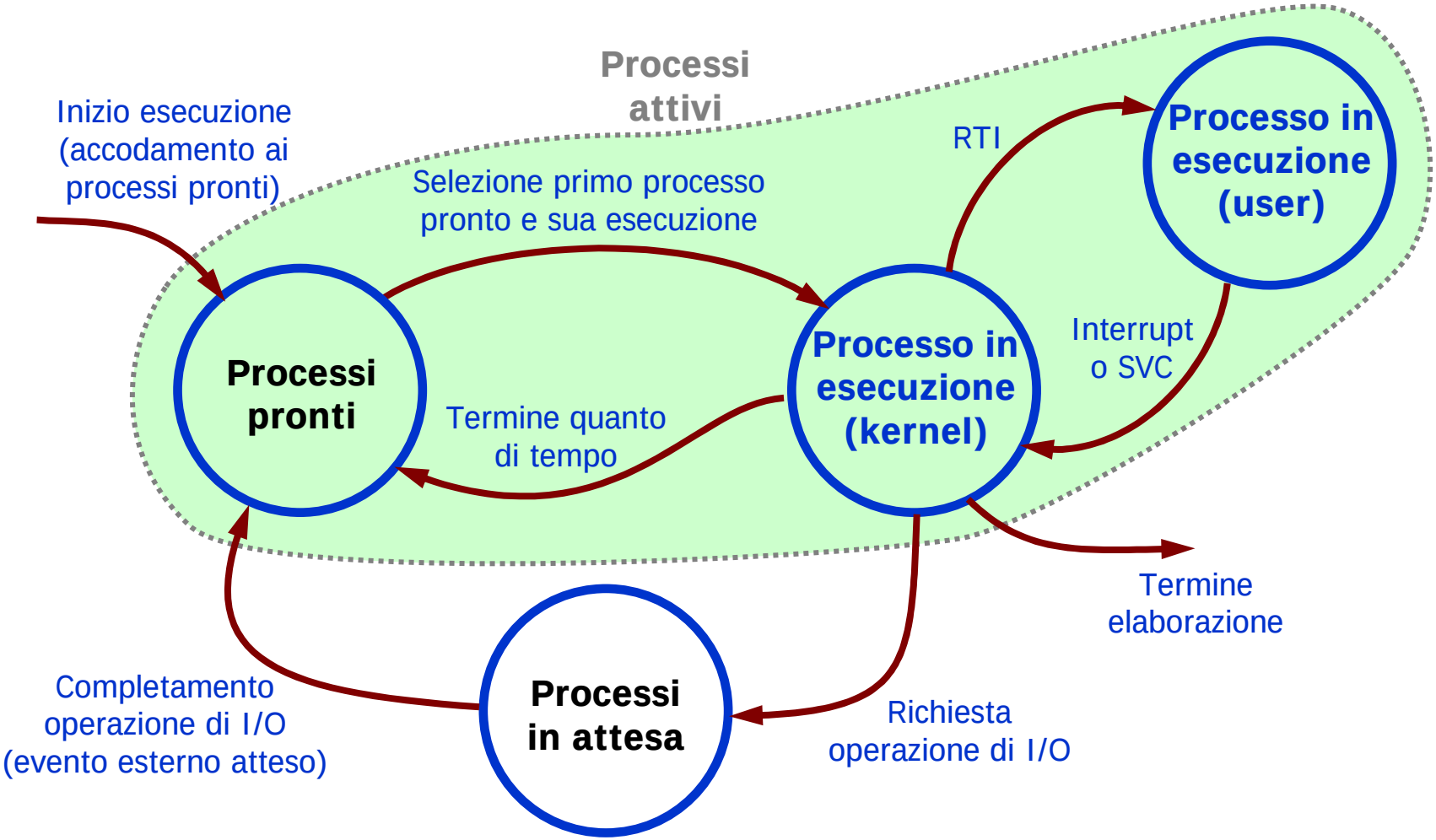
Context swapping

- Il processo non si rende conto delle interruzioni:
  - il nucleo maschera al processo come effettivamente la sua elaborazione evolve nel tempo;
  - il nucleo rende trasparente la presenza delle operazioni di interruzione e di riassegnamento del processore a un processo.
- Contesto di un processo
  - insieme dei dati che rappresentano lo “stato” del processo: situazione della memoria, contenuto dei registri, livello di priorità, ...
  - quando un processo viene interrotto (esce dallo stato di esecuzione) il nucleo provvede a **salvare** del suo contesto (in una struttura dati chiamata **descrittore del processo**);
  - quando un processo torna nello stato di esecuzione il nucleo provvede a **ripristinare** il suo contesto (recuperando i dati precedentemente salvati nel descrittore).
- Cambio di contesto (context swapping)
  - si verifica quando un processo (e.g. P1) in esecuzione viene sostituito da un altro processo P2 (il primo del processi pronti);
  - il nucleo provvede a
    1. salvare il contesto di P1 e gestirne l’evoluzione (pronto vs attesa);
    2. ripristinare il contesto di P2 per consentirgli una corretta evoluzione.

Modalità user e modalità kernel

- I processi possono essere eseguiti in modalità kernel (riservata ai servizi forniti dal sistema operativo) o user (programmi applicativi)
  - non basta più un solo stato di esecuzione, è necessario distinguere le due situazioni;
  - due nuovi stati: **esecuzione user** e **esecuzione kernel**.
- Ci sono due nuove transizione di stato:
  - **esecuzione user → esecuzione kernel**
    - richiesta di servizi al sistema operativo da parte del processo utente, detta **chiamata di sistema** (SuperVisor Call – SVC)
    - evento esterno segnalato da una periferica (**Interrupt**)
  - **esecuzione kernel → esecuzione user**
    - termine gestione interrupt (return from interrupt – **RTI**)

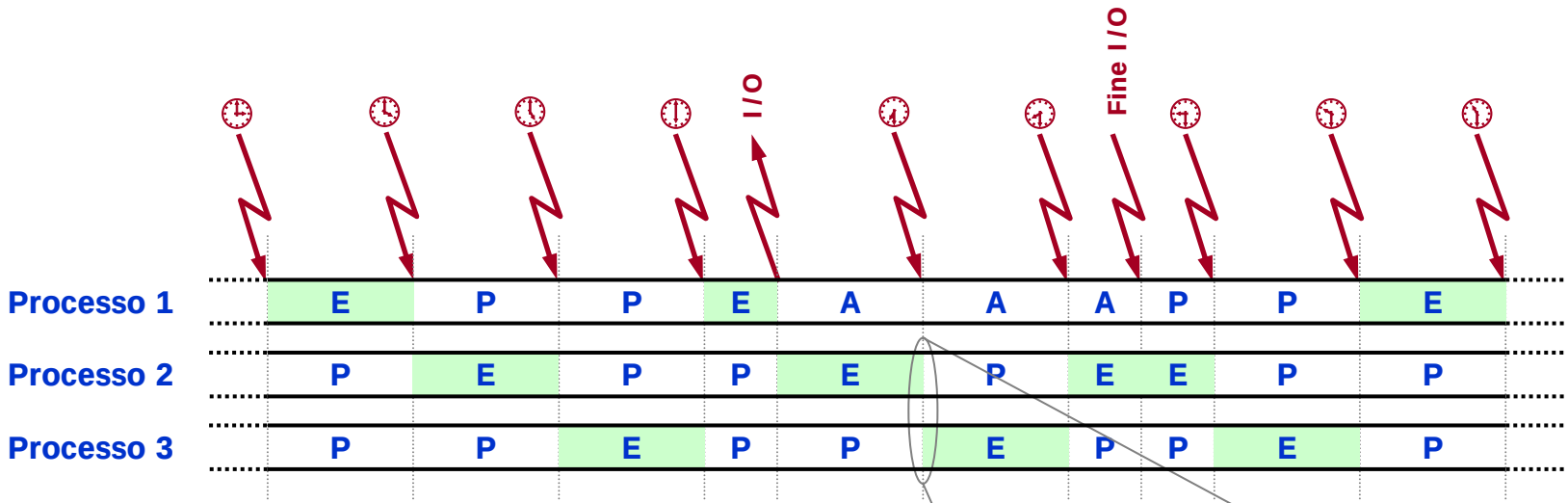
Transizioni di stato

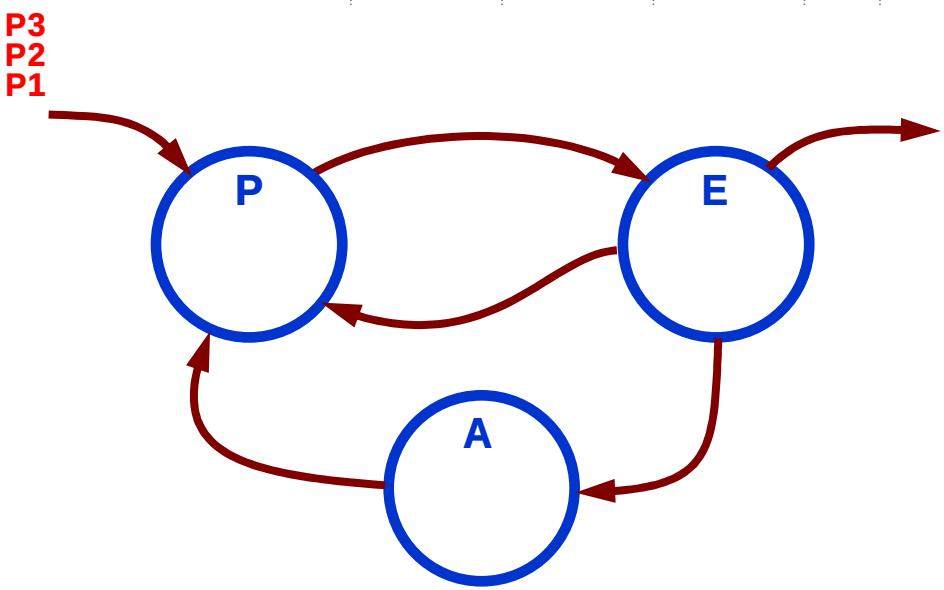


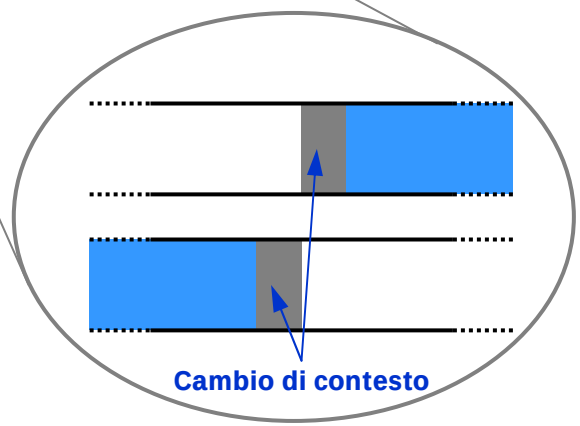
Esempio: esecuzione di P1 e P2

1. **P1 in esecuzione (user):**
   - esegue una serie di istruzioni;
   - richiede un servizio di I/O, e.g. lettura di un carattere da tastiera [**SVC**].
2. **SO in esecuzione (kernel):**
   - decide di sospendere P1 finché non termina l'operazione di I/O richiesta;
   - salva il contesto di P1 nel descrittore di P1 e sposta P1 in attesa di "carattere da tastiera".
   - sceglie P2 tra i processi pronti per mandarlo in esecuzione;
   - ripristina il contesto di P2 leggendo i dati dal descrittore di P2;
   - manda in esecuzione P2 restituendogli il controllo [**RTI**].
3. **P2 in esecuzione (user):**
   - esegue una serie di istruzioni;
   - arriva un interrupt da tastiera (evento esterno) e il controllo passa al SO [**interrupt**].
4. **SO in esecuzione (kernel):**
   - legge il carattere in arrivo dalla tastiera e lo scrive in un buffer di sistema;
   - sposta P1 nella coda dei processi pronti;
   - restituisce il controllo a P2 [**RTI**].
5. **P2 in esecuzione (user):**
   - esegue una serie di istruzioni;
   - arriva un interrupt da orologio (evento esterno) e il controllo passa al SO [**interrupt**].
6. **SO in esecuzione (kernel):**
   - decide di sospendere P2 perché è scaduto il suo quanto;
   - salva il contesto di P2 nel descrittore di P2 e sposta P2 tra i processi pronti;
   - sceglie P1 tra i processi pronti per mandarlo in esecuzione (ipotesi: ci sono solo P1 e P2);
   - ripristina il contesto di P1 leggendo i dati dal descrittore di P1;
   - manda in esecuzione P1 restituendogli il controllo [**RTI**].
7. **P1 in esecuzione (user) … … … …**

Round Robin


I/O
Fine I/O
Processo 1
Processo 2
Processo 3
E P P E A A A P P E
P E P P E P E E P P
P P E P P E P P E P


P3
P2
P1
P
E
A


Cambio di contesto

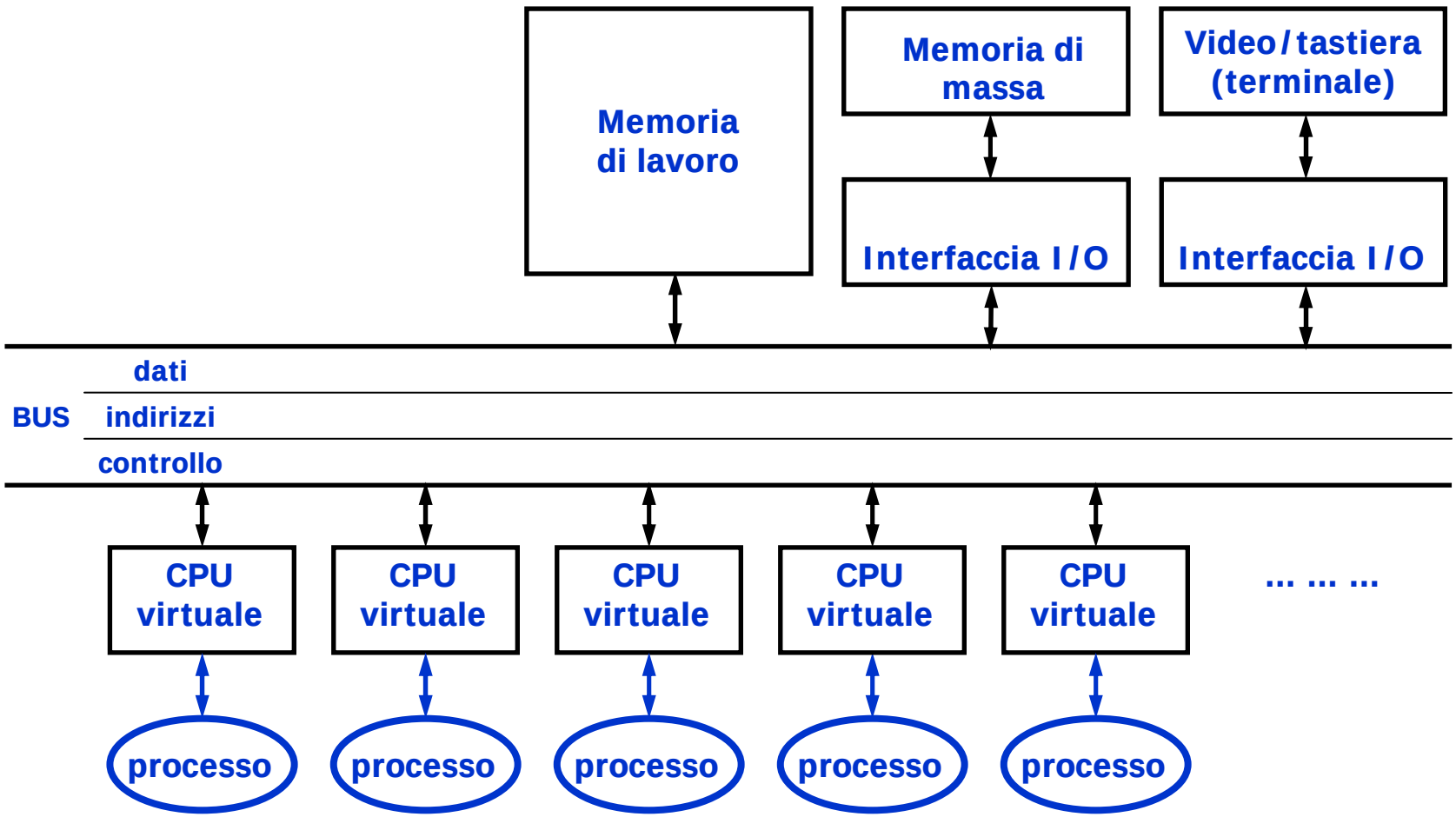
“concorrenza” fra processi

- Vantaggi dell’esecuzione concorrente di più processi:
  - impiegare in maniera trasparente una o più CPU (sia inserite in un solo calcolatore che in più calcolatori, collegati in rete);
  - aumentare l’utilizzo della CPU nei sistemi a partizione di tempo, ove si eseguono più lavori in quasi parallelismo;
  - condividere la stessa risorsa fisica fra diversi utenti in modo del tutto trasparente ma controllato;
  - accedere contemporaneamente, da parte di diversi utenti, a una base di dati comune e centralizzata;
  - ...
- Problemi
  - **starvation**: un processo non riesce ad accedere ad una risorsa perché la trova sempre occupata da altri processi (che per esempio possono avere un livello di priorità maggiore);
  - **blocco critico**: un insieme di processi rimane bloccato perché ciascuno di essi aspetta delle risorse che sono occupate da un altro processo compreso in questo stesso insieme (**vincolo circolare**).
  - Evitare (prevenzione) o risolvere (eliminazione) situazioni di blocco critico o di starvation riduce le prestazioni complessive del sistema.

Interazioni tra processi

- Le **interazioni** fra processi sono classificabili in:
  - **indesiderate e (spesso) impreviste**:
    i processi **competono** per le risorse che servono per completare l'esecuzione (causando eventualmente problemi quali blocco critico e starvation);
  - **desiderate e previste**:
    legate a meccanismi di **cooperazione** fra processi che hanno l'obiettivo di giungere alla soluzione di un problema complesso.
- La **gestione delle interazioni** fra i processi implica
  - la **sincronizzazione** fra le varie attività che ogni singolo processo deve svolgere in modo parallelo rispetto agli altri
  - la **comunicazione**, ovvero una modalità per lo scambio dei dati fra i processi
- **Modalità di funzionamento** dei processi:
  - **in foreground**, quando il processo è abilitato all'interazione con l'utente;
  - **in background**, quando il processo non è in grado, almeno temporaneamente, di interagire direttamente con l'utente; questo è lo stato in cui si trovano parecchi dei processi relativi alle funzioni interne del sistema operativo
    - sono attivati automaticamente all'accensione;
    - sono chiamati **daemon** in Unix e **agenti** o **servizi** in altri sistemi.

Nucleo: macchina astratta



La gestione della memoria

Gestore della memoria

- Applica tecniche per gestire il conflitto fra dimensione della memoria fisica e spazio complessivo richiesto dai programmi che devono essere eseguiti in modo concorrente e dai relativi dati.
- Combina le seguenti strategie:
  - consentire il caricamento di un programma a partire da un indirizzo qualunque della memoria;
  - ridurre la necessità di spazio tenendo in memoria solo una porzione dei programmi e dei dati;
  - condividere parte delle istruzioni (codice eseguibile) fra diversi processi corrispondenti a uno stesso programma.

- Il gestore della memoria
  - garantisce ai vari processi uno **spazio di indirizzamento virtuale** in cui lavorare, che può essere superiore alla memoria fisica presente nel calcolatore
  - mette in atto dei meccanismi di protezione che tutelano la privatezza dello spazio di lavoro assegnato a ogni processo.

La rilocabilità del codice

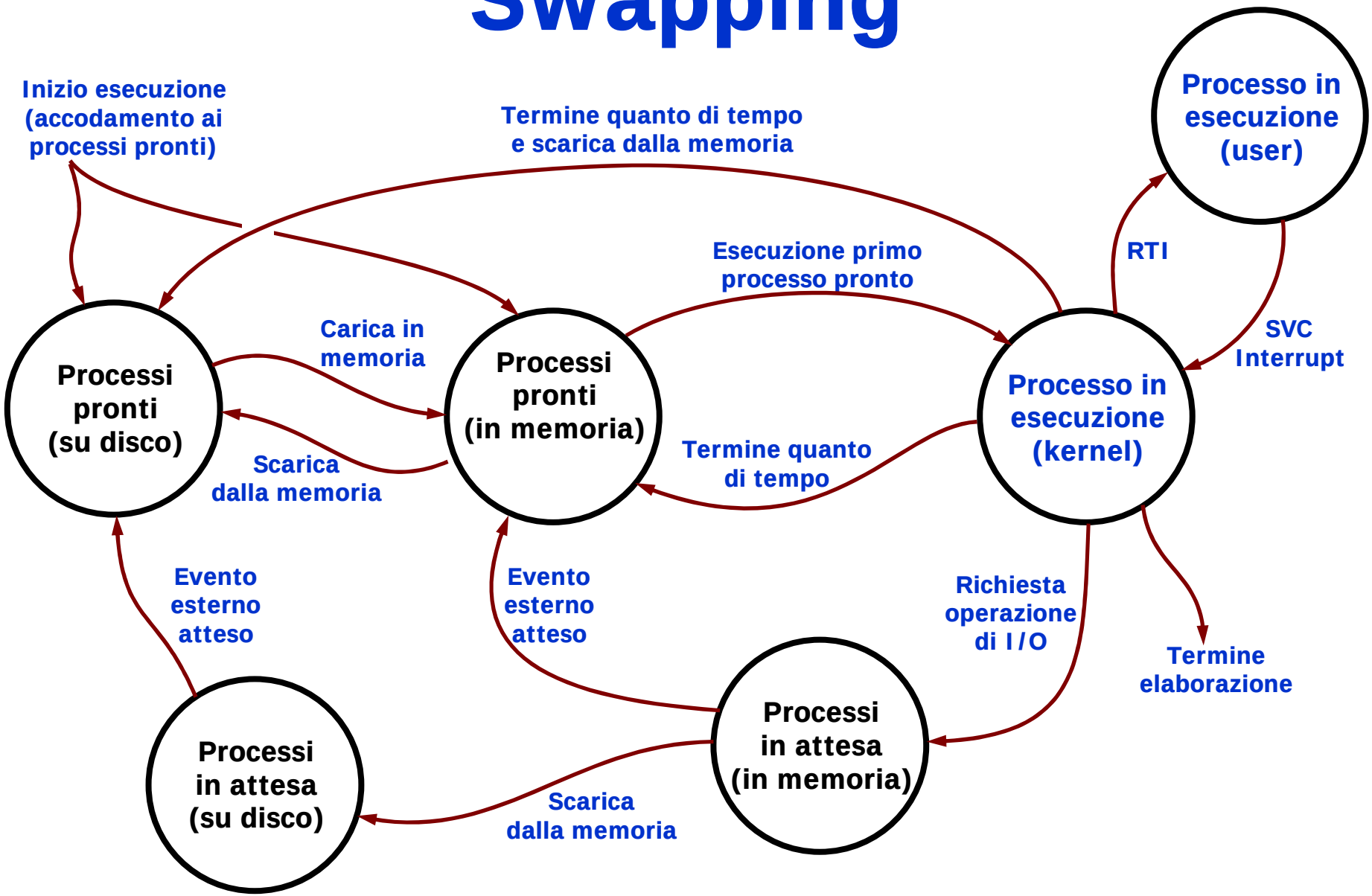
- Durante la compilazione i nomi simbolici e i riferimenti a celle di memoria sono stati risolti:
  - le istruzioni sono in formato macchina
  - tutti i riferimenti a istruzioni e dati sono espressi nella forma di indirizzi.
- Due spazi di memoria
  - **spazio logico**: intervallo di celle contigue che partono dall'indirizzo 0 in cui si immagina siano collocate le istruzioni durante la compilazione;
  - **spazio fisico**: lo spazio di memoria in cui risiede effettivamente il codice.

- Per far funzionare il programma caricandolo a partire da una posizione arbitraria della memoria bisogna effettuare una **rilocazione**: sommare a tutti gli indirizzi presenti nel programma un valore (**spiazzamento**) corrispondente alla differenza fra l'indirizzo a partire dal quale verrà effettivamente caricato il programma e il valore a partire dal quale sono stati calcolati gli indirizzi.
  - rilocazione statica: fatta direttamente dal linker previa indicazione dell'indirizzo di caricamento del programma;
  - rilocazione dinamica: eseguita durante l'esecuzione di ogni istruzione, ma che richiede l'esistenza di opportuni dispositivi interni alla CPU.

Swapping

- Spesso la memoria centrale non ha dimensioni tali da contenere tutti i programmi che occorre eseguire in modo concorrente:
  - manca spazio per attivare nuovi processi
  - manca spazio per consentire l'evoluzione di qualche processo già in esecuzione
- Soluzione: trasferire il contenuto di un'area della memoria centrale in un'area della memoria di massa (**area di swap**)
  - OK per processi in attesa
  - ~ OK per i processi pronti

Swapping


Inizio esecuzione (accodamento ai processi pronti)
Termine quanto di tempo e scarica dalla memoria
Processo in esecuzione (user)
RTI
SVC Interrupt
Esecuzione primo processo pronto
Carica in memoria
Processi pronti (su disco)
Processi pronti (in memoria)
Processo in esecuzione (kernel)
Scarica dalla memoria
Termine quanto di tempo
Evento esterno atteso
Evento esterno atteso
Richiesta operazione di I/O
Termine elaborazione
Processi in attesa (su disco)
Processi in attesa (in memoria)
Scarica dalla memoria

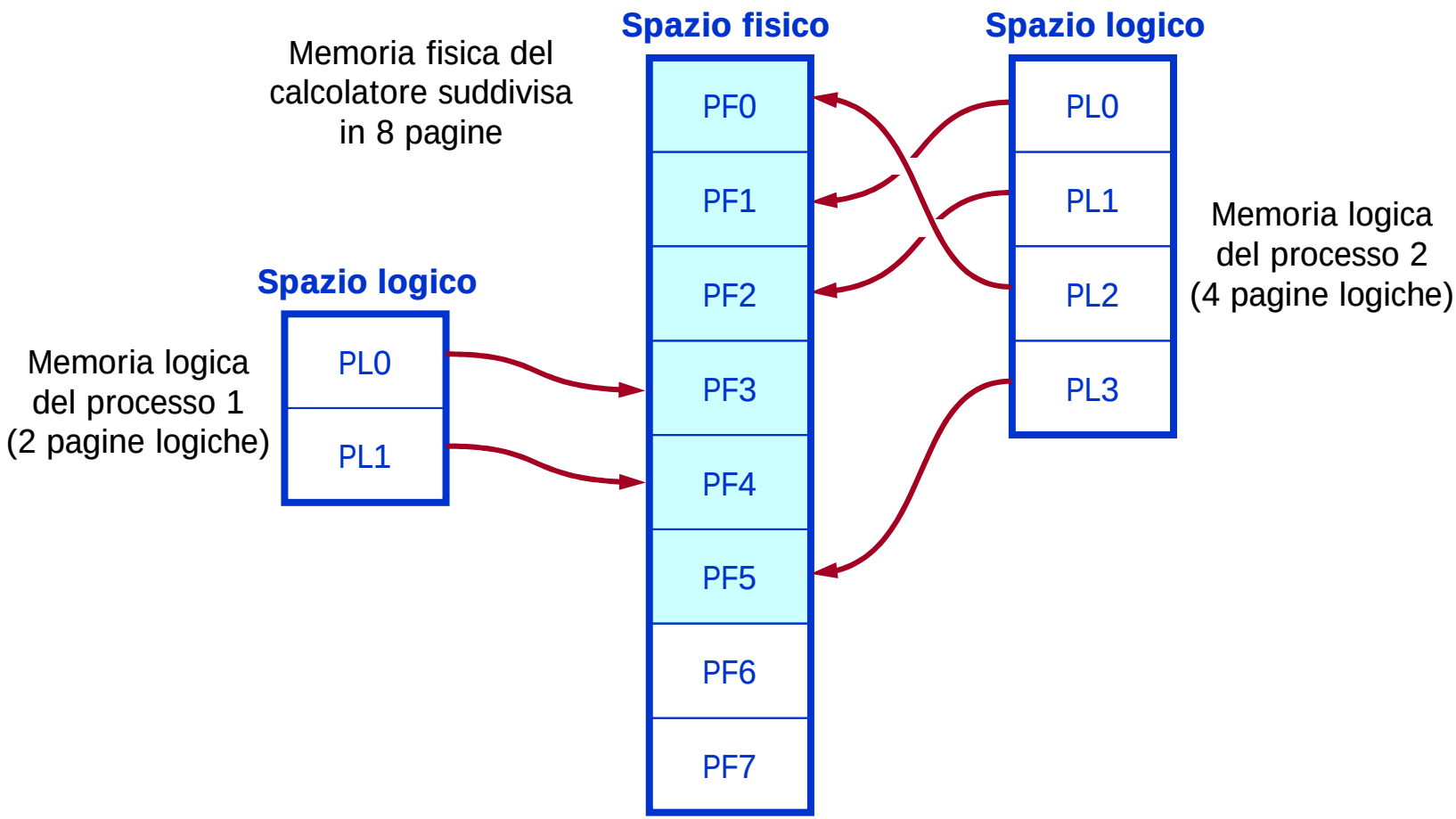
Paginazione

- Frammentazione della memoria (logica e fisica) in blocchi di dimensioni prefissate: **le pagine.**
- Lo spazio logico di indirizzamento del processo è suddiviso in sezioni, di dimensioni fisse e uguali fra loro, dette **pagine logiche**
- Lo spazio fisico di indirizzamento disponibile nel calcolatore è anch'esso suddiviso in **pagine fisiche**, della stessa dimensione delle pagine logiche.
- È possibile così:
  - estendere la dimensione di un processo utilizzando delle zone di memoria non necessariamente contigue;
  - tenere in memoria solo la porzione ridotta del programma che si sta utilizzando.

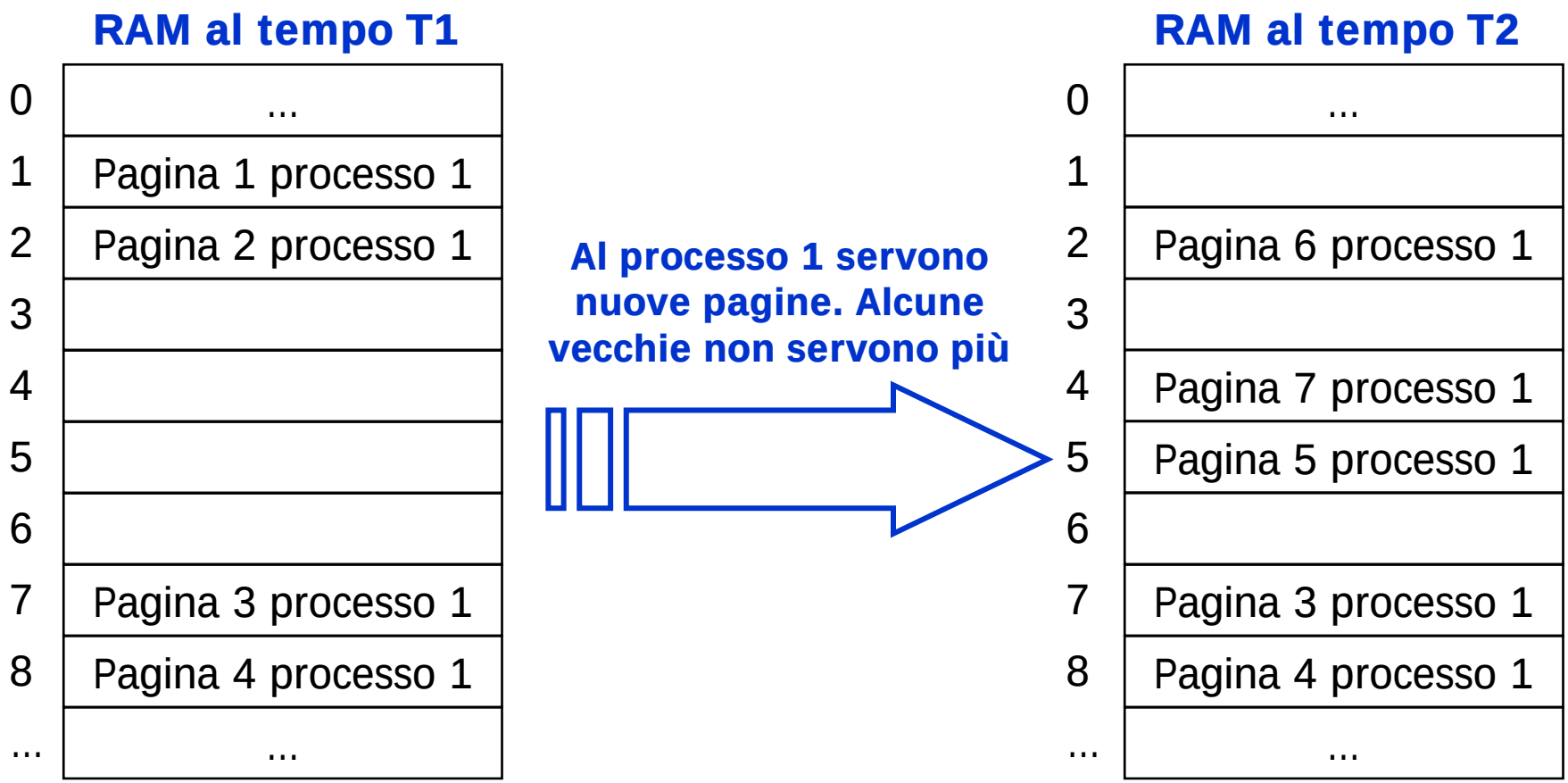
Paginazione

- Si basa sul principio di località spazio-temporale
  - Non vale la pena di caricare in RAM tutto il codice poiché l'esecuzione in un dato istante di tempo si limita ad una sezione limitata del codice stesso che, spesso, viene rieseguita.
    - Principio approssimativo del 10-90: il 10% del codice richiede circa il 90% dell'intera elaborazione
- Meccanismo: Vengono caricate, in alcune pagine fisiche su RAM, solo alcune pagine logiche del codice in esecuzione. Le pagine logiche necessarie vengono caricate di volta in volta, in base all'esigenza.
  - Quante pagine caricare e/o quante e quali pagine sostituire dipende dalla politica adottata.

Corrispondenza tra pagine logiche contigue e pagine fisiche non contigue

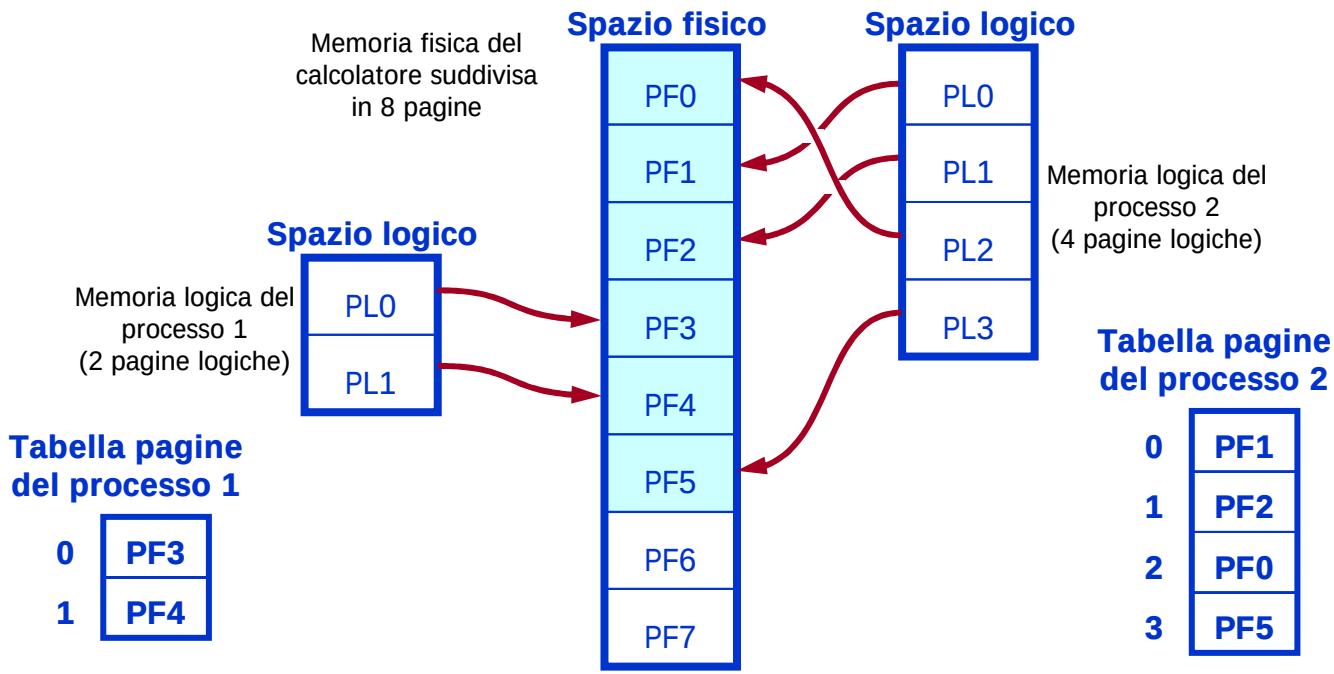


Paginazione



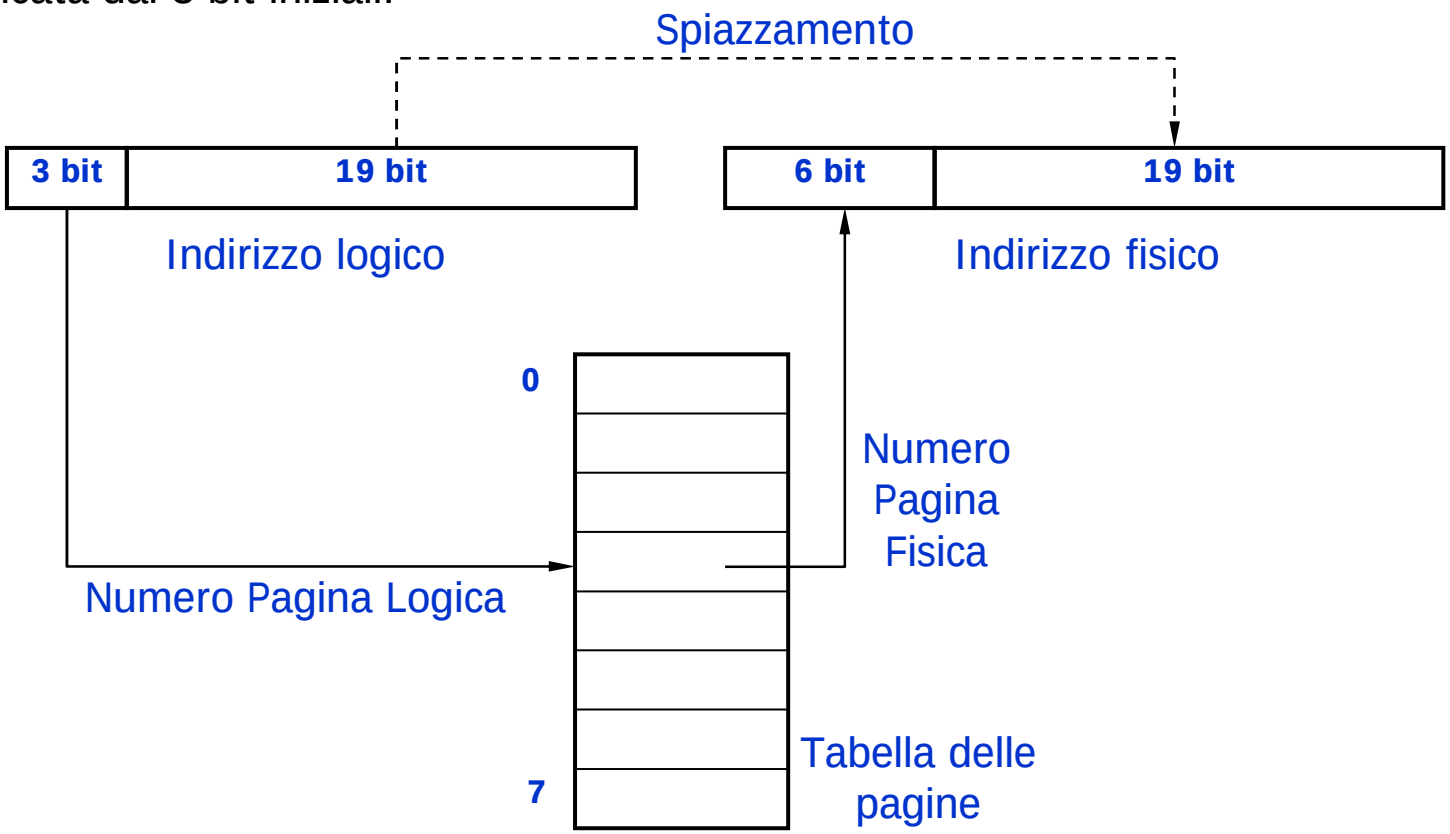
Memory Management Unit - MMU

- Serve un dispositivo dispositivo hardware aggiuntivo in grado di convertire gli indirizzi logici cui fa riferimento il programma nei corrispondenti indirizzi fisici: **Memory Management Unit.**
- La MMU utilizza una **tabella delle pagine**:
  - mantiene la relazione tra ogni pagina logica e l'indirizzo della pagina fisica corrispondente.



Memory Management Unit - MMU

- Dimensioni:
  - Memoria fisica di 32 MByte (indirizzata dunque con 25 bit)
  - Memoria logica di 4 MByte (indirizzo di 22 bit)
  - Pagine lunghe 512 KByte (indirizzo di 19 bit).
- I primi 3 dei 22 bit dell'indirizzo logico selezionano una delle $2^3=8$ righe della tabella delle pagine, il cui contenuto rappresenta l'indirizzo della pagina fisica corrispondente, mentre i restanti 19 bit identificano lo spiazzamento relativo all'inizio della pagina specificata dai 3 bit iniziali.



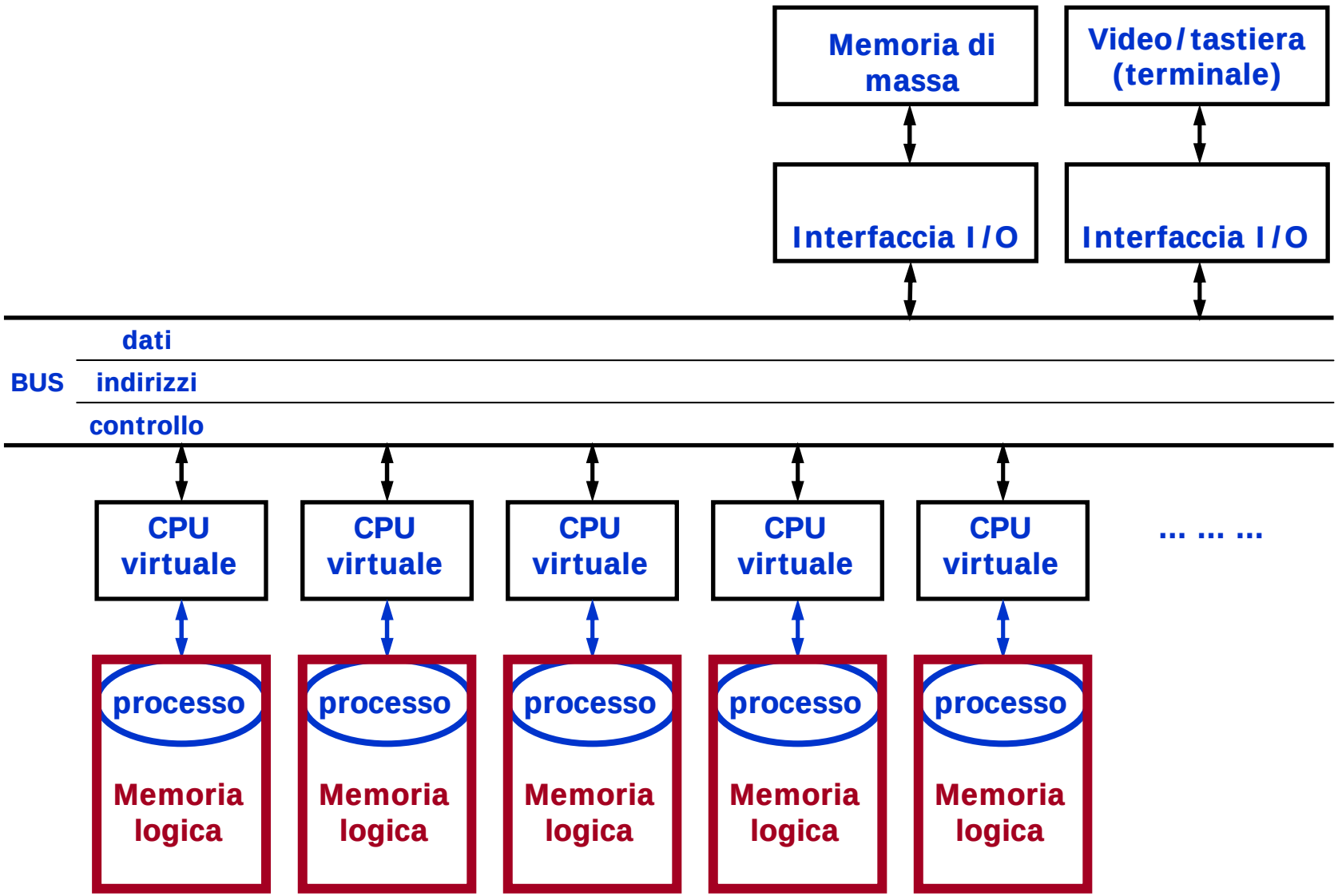
Ottimizzare l’uso della memoria

- Si possono tenere in memoria solo le pagine logiche di un programma a cui si sta accedendo più frequentemente.
  - quando un processo cerca di accedere a una pagina non in memoria si verifica un evento di **page fault**;
  - il sistema operativo libera dello spazio in memoria (per esempio scaricando delle pagine non utilizzate) e a caricare la pagina richiesta.
- Un opportuno dimensionamento su base statistica del numero di pagine logiche da tenere in memoria consente in genere di ridurre il numero di page fault, che comportano un rilevante sovraccarico per il sistema operativo.

Paginazione

- La paginazione risolve contemporaneamente tre problemi:
  1. Dove mettere il processo in memoria
     - non ha importanza quale sia il posto dove è allocato il processo;
     - la conversione delle pagine logiche in pagine fisiche maschera l'allocazione.
  2. Superare il numero di processi che posso gestire contemporaneamente
     - Se i processi fossero messi interamente in memoria lo spazio sarebbe sufficiente solo per pochi processi, non per tanti.
  3. Superare la dimensione fisica della memoria di lavoro
     - Non c'è relazione tra le pagine logiche di un processo e le pagine fisiche che gli vengono messe a disposizione dal SO;
     - Un processo può avere molte più pagine logiche di quante siano le pagine fisiche disponibili in RAM.

Gestore memoria: macchina astratta



Gestione periferiche I/O

Gestore delle periferiche

- Comunicazione tra l'ambiente CPU-RAM ed i dispositivi esterni.
  - Asincronicità tra ambiente e calcolatore
  - Gestione dell'accesso contemporaneo al calcolatore da parte di diverse periferiche.
- Mascherare ai processi l'esistenza di un numero limitato di risorse.
  - Esempio: stampa da più processi (es. due word ed un excel) su di una unica stampante.
- Mascherare ai processi la differenza tra risorse dello stesso tipo (o di tipo simile)
  - Esempio: stampante laser da un plotter e da un terminale video

Gestione periferiche I/O

- Comandi **ad alto livello** per accedere alle periferiche che usano meccanismi quali:
  - **i controller**, dispositivi hardware per effettuare le operazioni di trasferimento dati;
    - dipendono dalle caratteristiche fisiche delle periferiche che gestiscono
    - l'interfaccia per la gestione di un mouse è sicuramente diversa da quella utilizzata per controllare il funzionamento di un lettore di CD-ROM.
  - **i driver**, programmi software per la gestione delle periferiche;
    - mascherano le caratteristiche specifiche dei controller,
    - forniscono un insieme di primitive ad alto livello per la gestione delle operazioni di ingresso/uscita utilizzabili dai programmi applicativi e dagli utenti.
- I sistemi operativi comprendono i driver per la gestione delle periferiche più comuni:
  - tastiera, video, mouse, ...
  - stampanti, scanner, ...
- Ogni aggiunta o modifica alla configurazione standard comporta l'installazione di software addizionale (driver aggiuntivi).

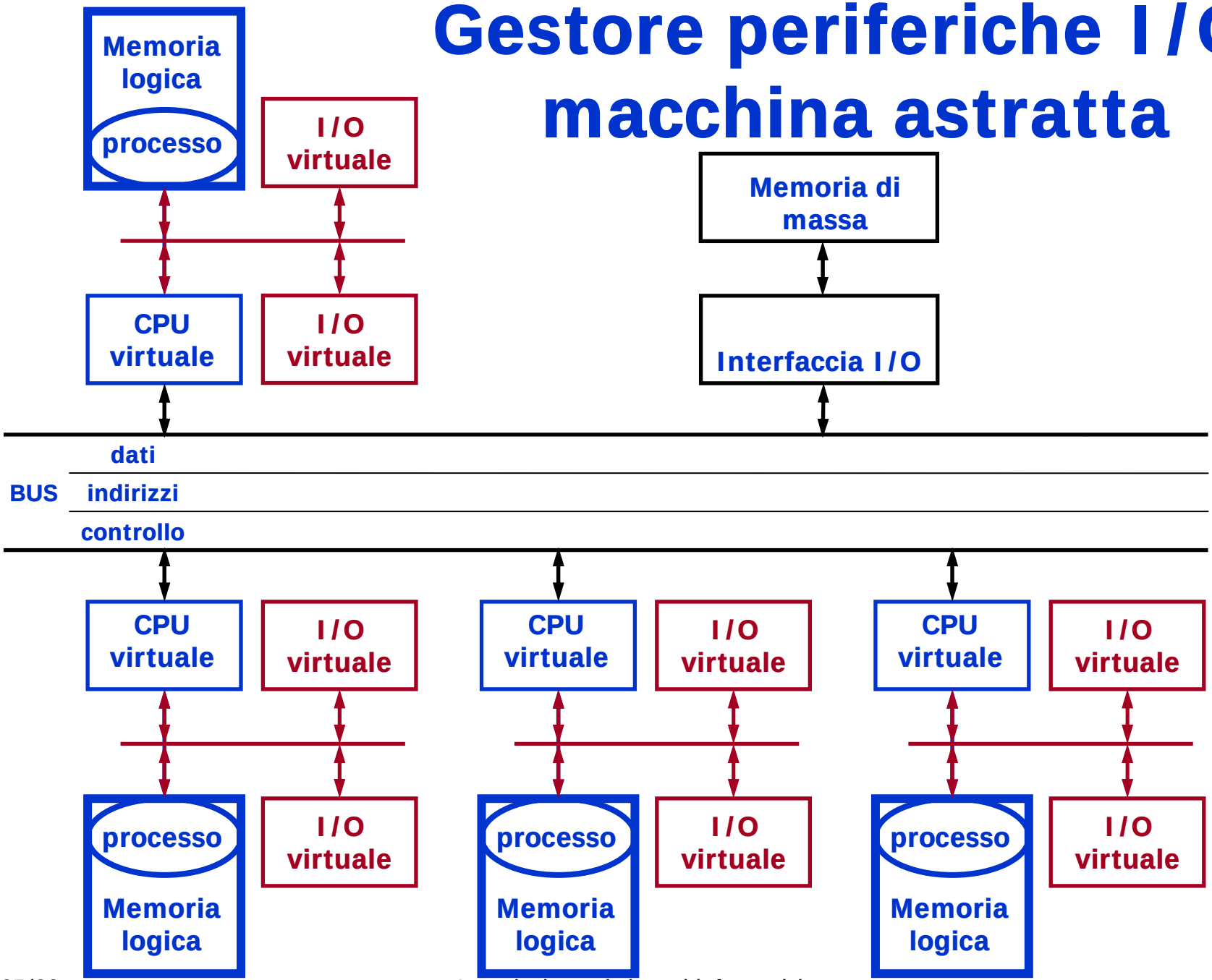
Plug&Play

- I sistemi operativi più recenti sono dotati di funzioni di **Plug&Play** (**PnP**) che permettono la configurazione automatica dei driver:
  - all'attivazione il sistema operativo scandisce ed esamina tutte le periferiche collegate al sistema;
  - le periferiche si fanno riconoscere specificando quali driver servono;
  - il sistema operativo installa gli opportuni driver per la loro gestione.
- Un sistema **PnP** consente di aggiungere (**plug**) nuove periferiche al sistema che possono essere utilizzate (**play**), senza necessità di intervento da parte dell'utente per la selezione e l'installazione dei driver.

Spooling

- I driver servono anche a virtualizzare la presenza di più periferiche intrinsecamente non condivisibili, tramite la tecnica di **spooling**.
- Esempio: gestione di una stampante
  - quando un processo desidera stampare un file, lo invia al driver,
  - il driver lo accoda in un'opportuna directory di spooling,
  - i file contenuti nella directory di spooling vengono stampati in ordine di arrivo (a meno che siano stabilite delle politiche di gestione delle priorità);
  - quando la directory di spooling si svuota il driver rimane in memoria in attesa che un processo cerchi di stampare.
- Questa soluzione
  - consente di disaccoppiare il programma che deve stampare e la periferica
  - rende possibile l'uso della stampante da parte di molti processi senza attese inutili.

Gestore periferiche I/O: macchina astratta



File System

Ovvero il **sistema di gestione** della **memoria di massa**.

Gestione memoria di massa

- **Obiettivo**:
  presentare all'utente l'organizzazione logica dei dati e le operazioni che è possibile compiere su di essi.
- Operazioni di base di un file system:
  - **recupero** di dati precedentemente memorizzati;
  - **eliminazione** (**cancellazione**) di dati obsoleti;
  - **modifica/aggiornamento** di dati preesistenti;
  - **copia** di dati (e.g. da HD a FD) per backup o per il trasferimento;
  - ...
- I servizi vengono forniti sia ai **programmi applicativi** che direttamente agli **utenti**.

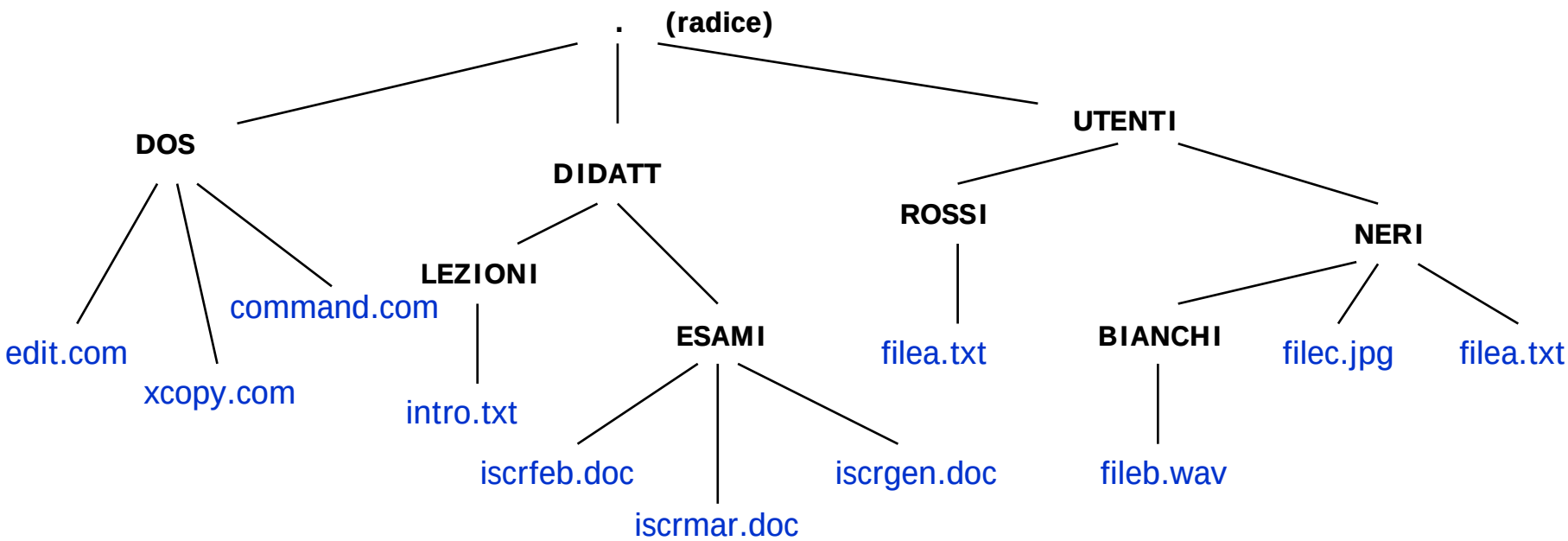
File system

- **FILE**:
  - contenitore logico di informazioni (dati o istruzioni);
  - oggetto a “lunga vita”, per conservare le informazioni anche dopo la terminazione del processo che lo ha generato.
- Per ogni file:
  - Identificatore (**nome.estensione**)
  - Periferica (**drive**) e percorso sulla periferica
  - Data creazione
  - Dimensione
  - Posizione effettiva dei dati nella memoria di massa
  - Altre informazioni
    - applicazione che consente all’utente di “usare” il file
    - data di ultima modifica
    - diritti di accesso al contenuto del file
    - ...

File

- I nomi dei file sono in genere composti da due parti:
  - nome (vero e proprio), che viene assegnato dall'utente
  - estensione, associata al programma che ha generato il file e consente quindi di identificare la tipologia dei dati contenuti nel file
    - i file eseguibili hanno estensioni exe,
    - i file generati da Word hanno estensione doc,
    - i file di testo generici hanno estensione txt.
- Ogni sistema operativo pone dei vincoli sulla lunghezza dei filename e sui caratteri di cui possono essere costituiti
  - MS-DOS imponeva una lunghezza massima di 8+3 caratteri per nomi ed estensioni
  - Windows ha un limite di 254 caratteri (compreso il path)
- I file sono generalmente organizzati in cartelle (directory) e sottocartelle in una gerarchia ad albero (o, al limite, a grafo aciclico).

Un esempio di struttura



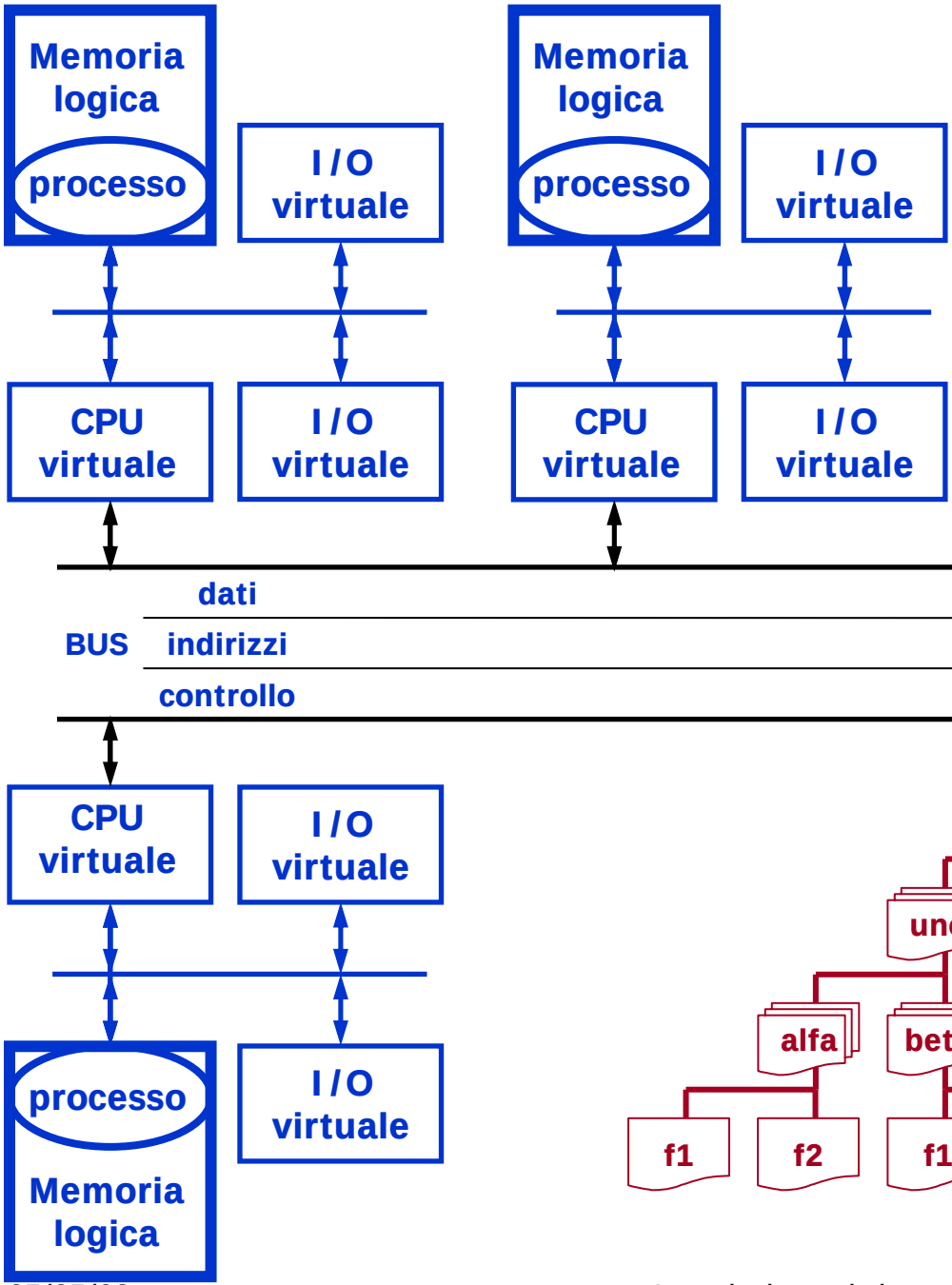
Organizzazione fisica dei dati

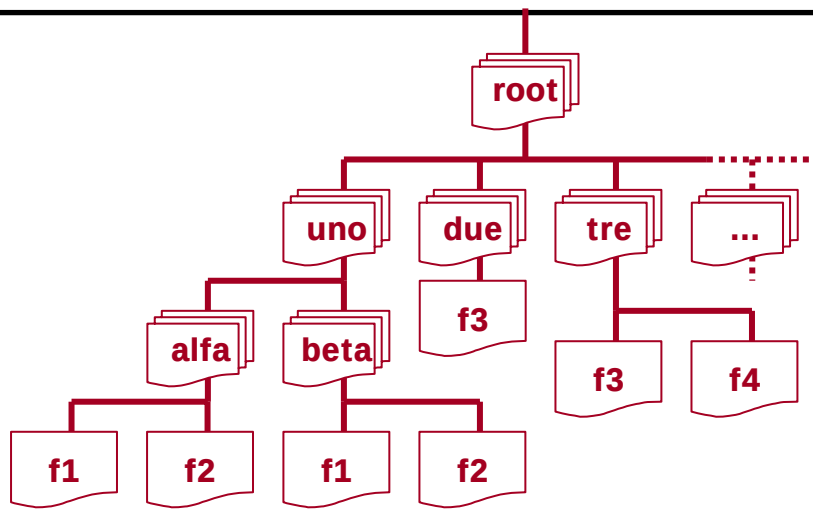
- Come mantenere la corrispondenza tra il nome del file e i blocchi su disco che ne contengono i dati:
  - **lista concatenata** (e.g. Windows 95/98)
    - a partire dal primo elemento, in coda a ogni blocco di dati viene riportato l'indirizzo del successivo),
    - la lista può essere memorizzata in una particolare area del disco: **File Allocation Table**, FAT
    - l'indirizzo del primo blocco dei dati si trova nel **descrittore di file**
    - per arrivare a conoscere l'effettivo indirizzo su disco di un dato è necessario analizzare, qualora il file sia composto da N blocchi, un numero di blocchi proporzionale a N.
  - **i-node** (e.g. UNIX)
    - se il file è piccolo, l'i-node contiene l'indicazione dei blocchi di dati,
    - se il file è grande, l'i-node identificare un insieme di altri i-node, che a loro volta specificano blocchi di dati;
    - se il file è molto grande si ripete il procedimento con un livello ulteriore di indirezione; ...
    - per arrivare a conoscere l'effettivo indirizzo su disco di un dato è necessario analizzare, qualora il file sia composto da N blocchi, un numero di i-node proporzionale a log(N).

Il controllo degli accessi

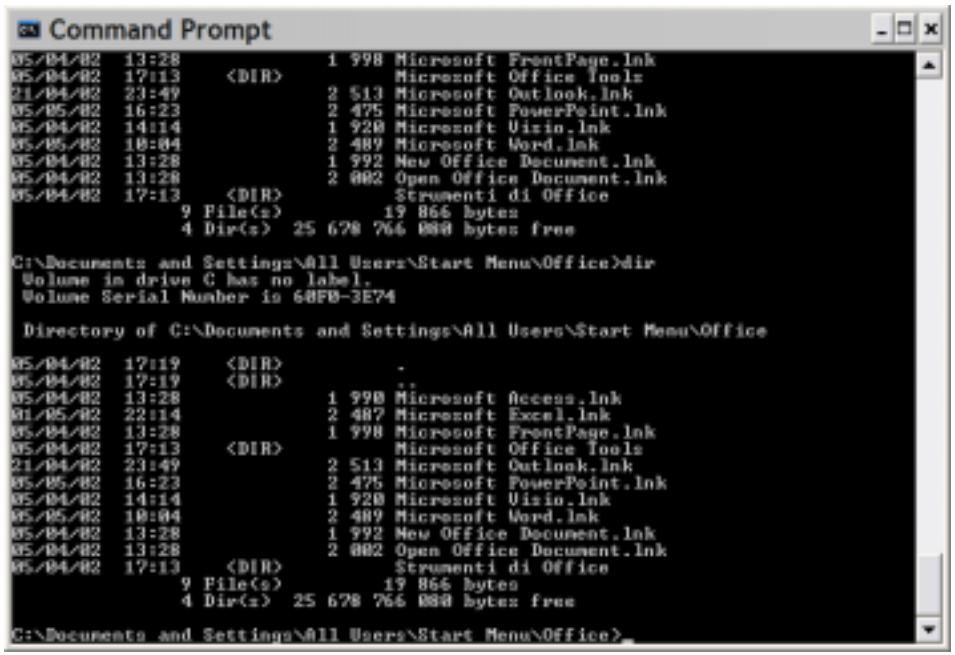
- Identificazione degli accessi al sistema.
  - Associare a ogni utente un **account** (**login**) e una parola d'ordine (**password**).
  - All'interno del sistema operativo, in un apposito file, è contenuta la lista di tutti gli account e delle relative password: solo se viene specificato un account fra quelli previsti (utente abilitato) e la password corrisponde a quella memorizzata (certificazione di identità) viene consentito l'accesso al sistema.
- Questo consente di **personalizzare** il sistema, per esempio definendo:
  - la distribuzione dei costi di gestione fra i vari utenti;
  - la visibilità del sistema in termini di porzione del file system complessivo, periferiche e programmi applicativi disponibili;
  - la personalizzazione dell'ambiente operativo.
- Consente di controllare gli accessi ai file:
  - livello di **protezione** a livello di file o di directory;
  - e.g. UNIX:
    - tre tipi di utenti: il **proprietario**, il **gruppo** e il "**resto del mondo**".
    - tre abilitazioni: **lettura** (R), **scrittura** (W) ed **esecuzione** (X).
  - altro metodo: **Access Control List**
    - nel SO esiste una tabella in cui ogni riga corrisponde a una diversa risorsa del sistema (programmi, stampanti, directory...),
    - la riga contiene una lista che specifica tutti gli utenti abilitati all'uso della corrispondente risorsa e le modalità per la sua fruizione (per esempio a un utente potrebbe essere consentito solo leggere ma non scrivere in una directory, oppure proibito l'uso di alcune stampanti collegate in rete).

File System: macchina astratta





Il gestore delle interfacce - Shell



Interfaccia a caratteri

Interfaccia grafica

(point & click)

“Information hiding”

- Principio su cui si basa l’evoluzione delle tecnologie, in particolare ICT
- L’informazione “inutile” per l’utente, che rende inutilmente complesso l’uso del calcolatore, viene mantenuta nascosta all’utente

- Il parallelo con altre tecnologie …