



# Capitolo 2: Strutture dei sistemi operativi

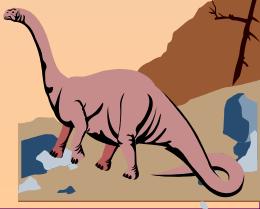
- Servizi di un sistema operativo
- Interfaccia con l'utente del sistema operativo
- Chiamate di sistema
- Categorie di chiamate del sistema
- Programmi del sistema
- Progettazione e realizzazione di un sistema operativo
- Struttura del sistema operativo
- Macchine virtuali
- Generazione di sistemi
- Avvio del sistema

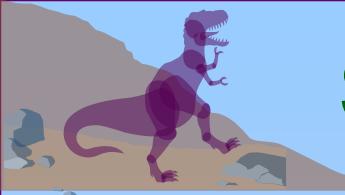




# Servizi di un sistema operativo

- Un S.O. offre un ambiente in cui eseguire i programmi e fornire servizi ai programmi e ai loro utenti.
- Ecco una lista di alcune classi di servizi comuni offerti dal S.O. per rendere più agevole la programmazione:
  - ☞ Interfaccia con l'utente:
    - ▣ interfaccia a riga di comando (CLI) - basata su stringhe che codificano i comandi, insieme ad un metodo per inserirli e modificarli,
    - ▣ interfaccia a lotti - comandi e relative direttive sono codificati nei file ed eseguiti successivamente a lotti,
    - ▣ interfaccia grafica con l'utente (GUI) - sistema grafico a finestre dotato di un dispositivo puntatore (ad es. il mouse).
  - ☞ Esecuzione di un programma – il sistema deve poter caricare un programma in memoria ed eseguirlo.

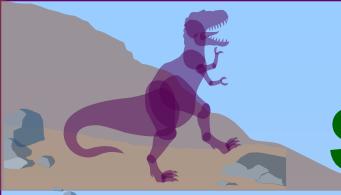




# Servizi di un sistema operativo (II)

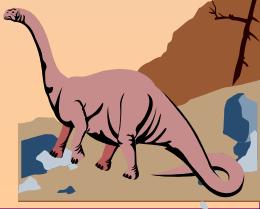
- ☞ Operazioni di I/O - i programmi utenti non possono eseguire direttamente operazioni di I/O:
  - ☞ S.O. deve fornire strumenti per permettere l'esecuzione di operazioni di I/O.
- ☞ Gestione del file system – esecuzione di operazioni di lettura, scrittura, creazione e cancellazione file.
- ☞ Comunicazioni – scambi di informazioni tra processi in esecuzione sullo stesso calcolatore o collegati tra loro per mezzo di una rete.
  - ☞ Realizzate tramite *memoria condivisa* o *scambio di messaggi*.
- ☞ Rilevamento di errori – assicurare la correttezza della computazione rilevando eventuali errori di CPU, di memoria, di I/O o in programmi utenti.





# Servizi di un sistema operativo (III)

- Esiste un'altra serie di funzioni del S.O. che non riguarda direttamente l'utente ma assicura il funzionamento efficiente del sistema stesso:
  - ☞ Assegnazione delle risorse – allocare risorse a più utenti o processi che sono concorrentemente in esecuzione.
  - ☞ Contabilizzazione dell'uso delle risorse – registrare quali utenti usino il calcolatore, segnalando quali e quante risorse impieghino.
  - ☞ Protezione e sicurezza – assicurare il controllo dell'accesso a tutte le risorse condivise di sistema identificando l'utente ad ogni suo accesso.





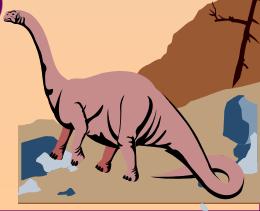
# Interfaccia utente: interprete dei comandi

- Le interfacce a linea di comando (CLI) permettono l'invio diretto di comandi:
  - ☞ quando i sistemi consentono la scelta tra molteplici interpreti dei comandi questi vengono definiti shell.
    - ☞ Unix: Bourne Shell, C Shell, Bourne-again Shell, Korn Shell...
  - ☞ A volte i comandi sono built-in (implementati nel kernel), altre sono semplici nomi di programmi speciali di sistema che vengono quindi eseguiti,
    - ☞ ad es.: `rm file.txt`.
    - ☞ Con il secondo approccio, l'aggiunta di nuove caratteristiche non richiede la modifica della shell.



# Interfaccia utente: interfaccia grafica con l'utente

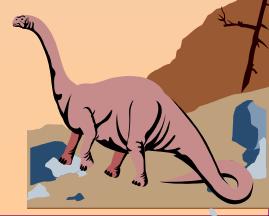
- Le interfacce grafiche con l'utente (GUI) forniscono uno strumento user-friendly:
  - ☞ I comandi sono forniti tramite mouse, tastiera, monitor.
  - ☞ Icone rappresentano file, programmi, azioni ....
  - ☞ La pressione dei tasti del mouse sugli oggetti dell'interfaccia causa varie azioni (recupero informazioni, opzioni, esecuzioni di funzioni, apertura di directory).
- Inventate negli anni '70 nei laboratori Xerox PARC, divennero comuni negli anni '80 con l'avvento degli Apple Macintosh.
- Molti sistemi oggi includono interfacce sia CLI che GUI:
  - ☞ Microsoft Windows offre una GUI ed una CLI
  - ☞ Apple Mac OS X offre una GUI (Aqua), che poggia su un kernel UNIX e mette a disposizione le shell UNIX
  - ☞ Solaris offre una CLI e optionalmente GUI (Java Desktop, KDE)





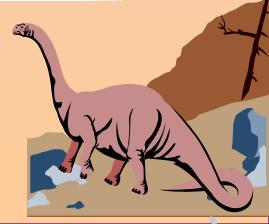
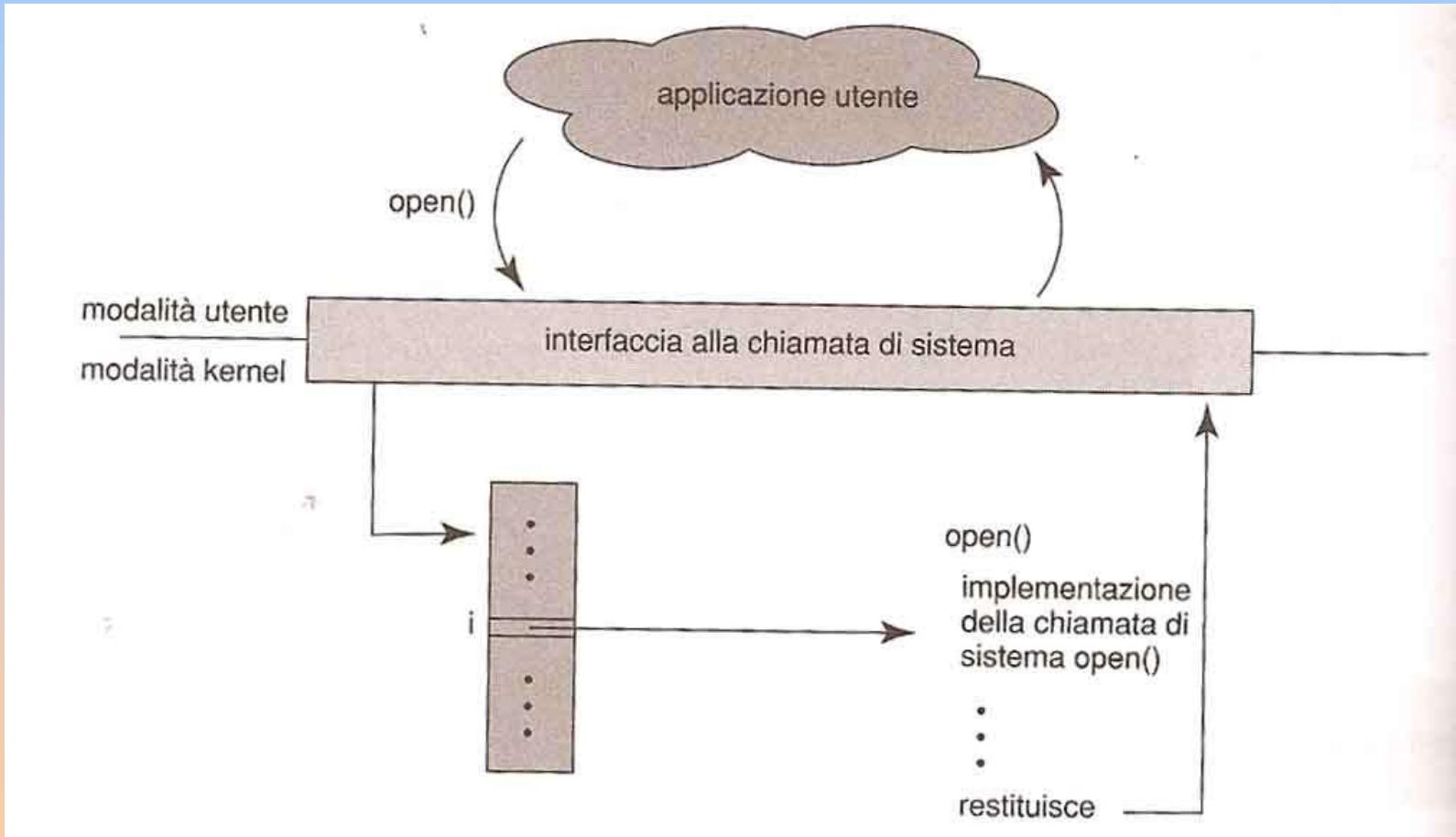
# Chiamate del sistema (*system calls*)

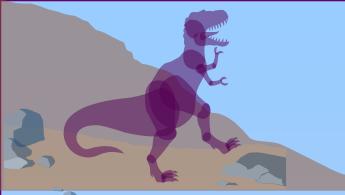
- Le system calls costituiscono l'interfaccia tra un processo ed il sistema operativo.
- Sono generalmente disponibili in forma di istruzioni in linguaggio assembly.
- In alcuni sistemi le chiamate possono essere invocate direttamente tramite funzioni scritte in programmi ad alto livello (C, C++).
- Tipicamente ad ogni chiamata di sistema è associato un numero.
  - ☞ Il sistema mantiene una tabella, indicizzata da questi numeri.





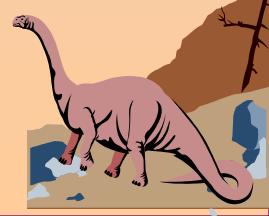
# Gestione della chiamata di sistema open()

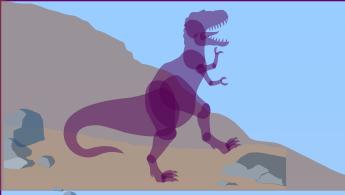




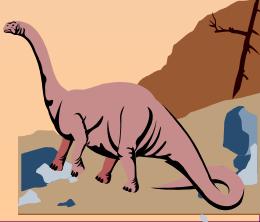
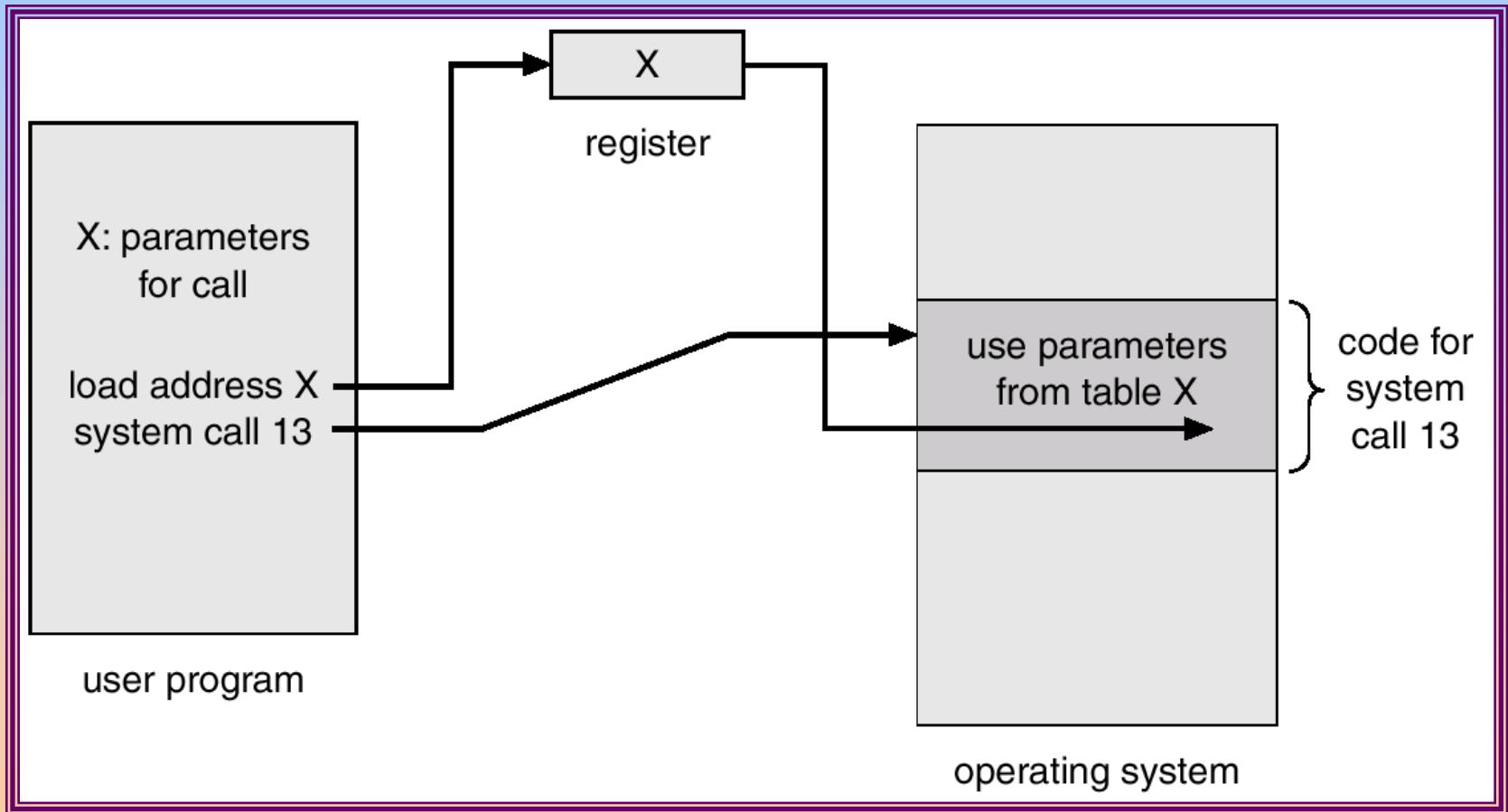
# Passaggio dei parametri

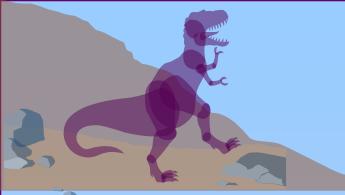
- Spesso sono richieste informazioni aggiuntive oltre all'identificativo della system call.
  - ☞ Il tipo e la quantità di informazione aggiuntivi necessari dipendono dal SO e dalla specifica chiamata di sistema.
- Per passare parametri tra un programma in esecuzione ed una chiamata al sistema operativo si può:
  - ☞ Passare i parametri in registri.
  - ☞ Memorizzare i parametri in una tabella in memoria e passare l'indirizzo della tabella come parametro in un registro.
  - ☞ Collocare (*push*) i parametri in una pila da cui sono prelevati (*pop*) dal S.O..





# Passaggio di parametri in forma di tabella

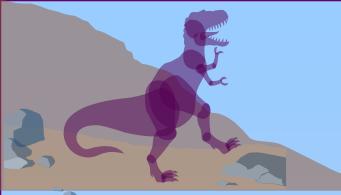




# API e Chiamate di sistema

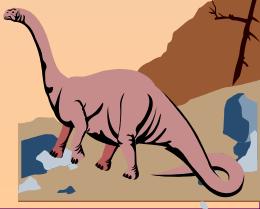
- I programmi applicativi accedono ai servizi del sistema principalmente tramite API (Application Programming Interface) piuttosto che direttamente attraverso le system call.
  - ☞ Tre API comuni:
    - ▀ la Win32 API per Windows, la POSIX API per sistemi POSIX-based (che virtualmente includono tutte le versioni di UNIX, Linux, e Mac OS X), e la Java API per la Java virtual machine (JVM).
  - ☞ Perchè usare API invece di system call direttamente?
    - ▀ Portabilità dei programmi, mascheramento dei dettagli delle system call reali, etc. .
- L'interfaccia (API) alle chiamate di sistema invoca l'opportuna chiamata di sistema e restituisce lo stato ed il valore di ritorno della chiamata stessa.
  - ☞ Il chiamante deve semplicemente deve seguire le specifiche dell'API e capire cosa il S.O. fa in seguito di una chiamata.
  - ☞ L'API nasconde al programmatore molti dettagli implementativi, gestiti dalla libreria di supporto run-time (insieme di funzioni della libreria inclusa con il compilatore).





# Categorie di chiamate del sistema

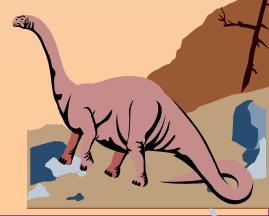
- Controllo dei processi.
  - Terminazione (normale o anormale)
  - Caricamento, esecuzione
  - Creazione ed arresto di un processo
  - Esame ed impostazione degli attributi di un processo
  - Attesa per il tempo indicato
  - Attesa e segnalazione di un evento
  - Assegnazione o rilascio di memoria
- Gestione dei file.
  - Creazione e cancellazione di file
  - Apertura, chiusura
  - Lettura scrittura posizionamento
  - Esame e impostazione degli attributi di un file

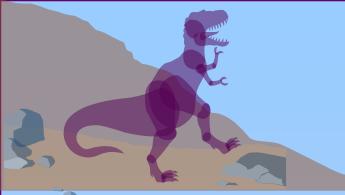




# Categorie di chiamate del sistema (II)

- Gestione dei dispositivi.
  - ☞ Richiesta e rilascio di un dispositivo
  - ☞ Lettura, scrittura, posizionamento
  - ☞ Esame e impostazione degli attributi di un dispositivo
  - ☞ Inserimento logico ed esclusione logica di un dispositivo
- Gestione delle informazioni.
  - ☞ Esame ed impostazione dell'ora e della data
  - ☞ Esame ed impostazione dei dati del sistema.
  - ☞ Esame ed impostazione degli attributi dei processi, file e dispositivi
- Comunicazione.
  - ☞ Creazione e chiusura di una connessione
  - ☞ Invio e ricezione di messaggi
  - ☞ Informazioni sullo stato di un trasferimento
  - ☞ Inserimento ed esclusione di dispositivi remoti





# Programmi di sistema

- I programmi di sistema offrono un ambiente per lo sviluppo e l'esecuzione dei programmi.
- Possono essere divisi in:
  - ☞ Gestione dei file.
  - ☞ Informazioni di stato.
  - ☞ Modifica dei file.
  - ☞ Ambienti di ausilio alla programmazione.
  - ☞ Caricamento ed esecuzione dei programmi.
  - ☞ Comunicazioni.
- Con la maggior parte dei S.O. sono anche forniti programmi che risolvono problemi o operazioni comuni:
  - ☞ *programmi di applicazione*.
- Per molti utenti l'interfaccia con S.O. è definita in termini di programmi di sistema invece che di system call.





# Progettazione e realizzazione di un S.O.

## ■ Scopi della progettazione:

### ☞ Scopi degli utenti:

- ☞ S.O. dovrebbe essere conveniente da usare, semplice da imparare, affidabile, sicuro e veloce

### ☞ Scopi del sistema:

- ☞ S.O. dovrebbe essere semplice da progettare, implementare e mantenere, ed anche flessibile, affidabile, privo di errori ed efficiente

## ■ Meccanismi e criteri:

- ☞ I meccanismi determinano come eseguire qualcosa, i criteri cosa si debba fare
- ☞ La separazione tra meccanismi e criteri permette massima flessibilità nel caso i criteri debbano essere successivamente modificati

## ■ Realizzazione:

- ☞ I S.O. ora possono essere scritti anche in linguaggi ad alto livello.

### ☞ Vantaggi

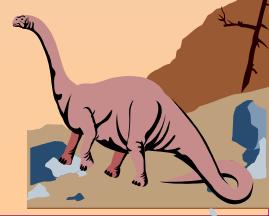
- ☞ Possono essere scritti più velocemente
- ☞ codice più compatto e più semplice da capire ai fini del debugging
- ☞ portabilità





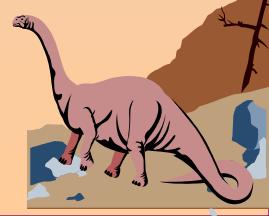
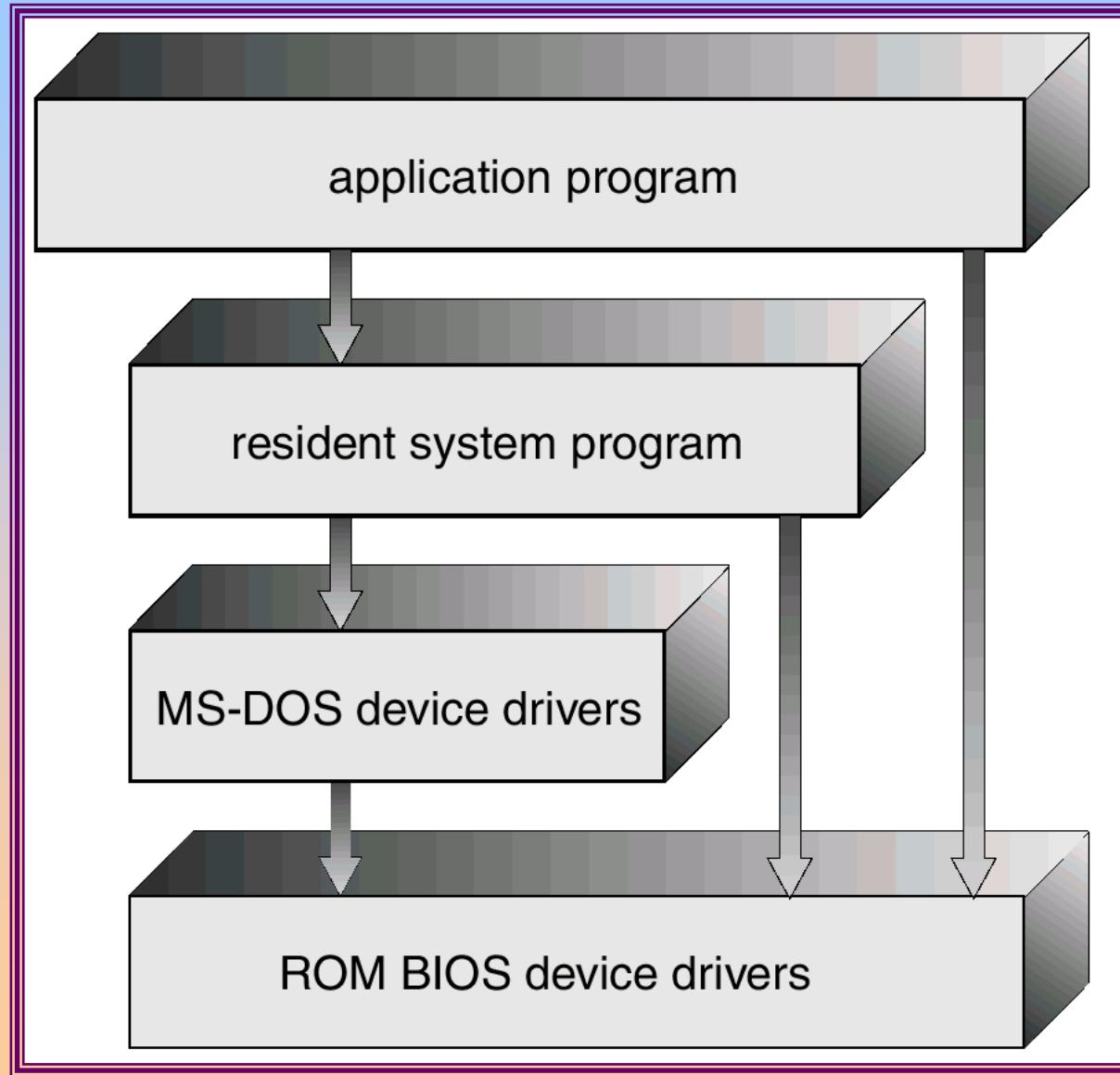
# Struttura del sistema

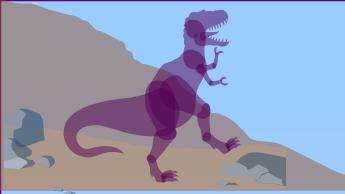
- Affinché possa funzionare correttamente ed essere facilmente modificabile un S.O. non viene in genere progettato come un sistema monolitico ma suddiviso in piccoli componenti.
- *Struttura semplice:*
  - ☞ Molti sistemi sono nati come sistemi piccoli e solo in un secondo tempo si sono accresciuti superando il loro scopo originale.
  - ☞ Ad es. MS-DOS, aveva come scopo il fornire la massima funzionalità nel minimo spazio.
    - ☒ Non è modulare
    - ☒ Nonostante la presenza di una struttura elementare le sue interfacce ed i livelli di funzionalità non sono ben separati





# Struttura degli strati di MS-DOS



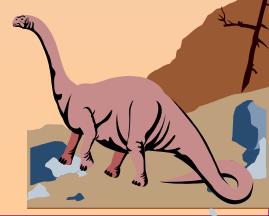
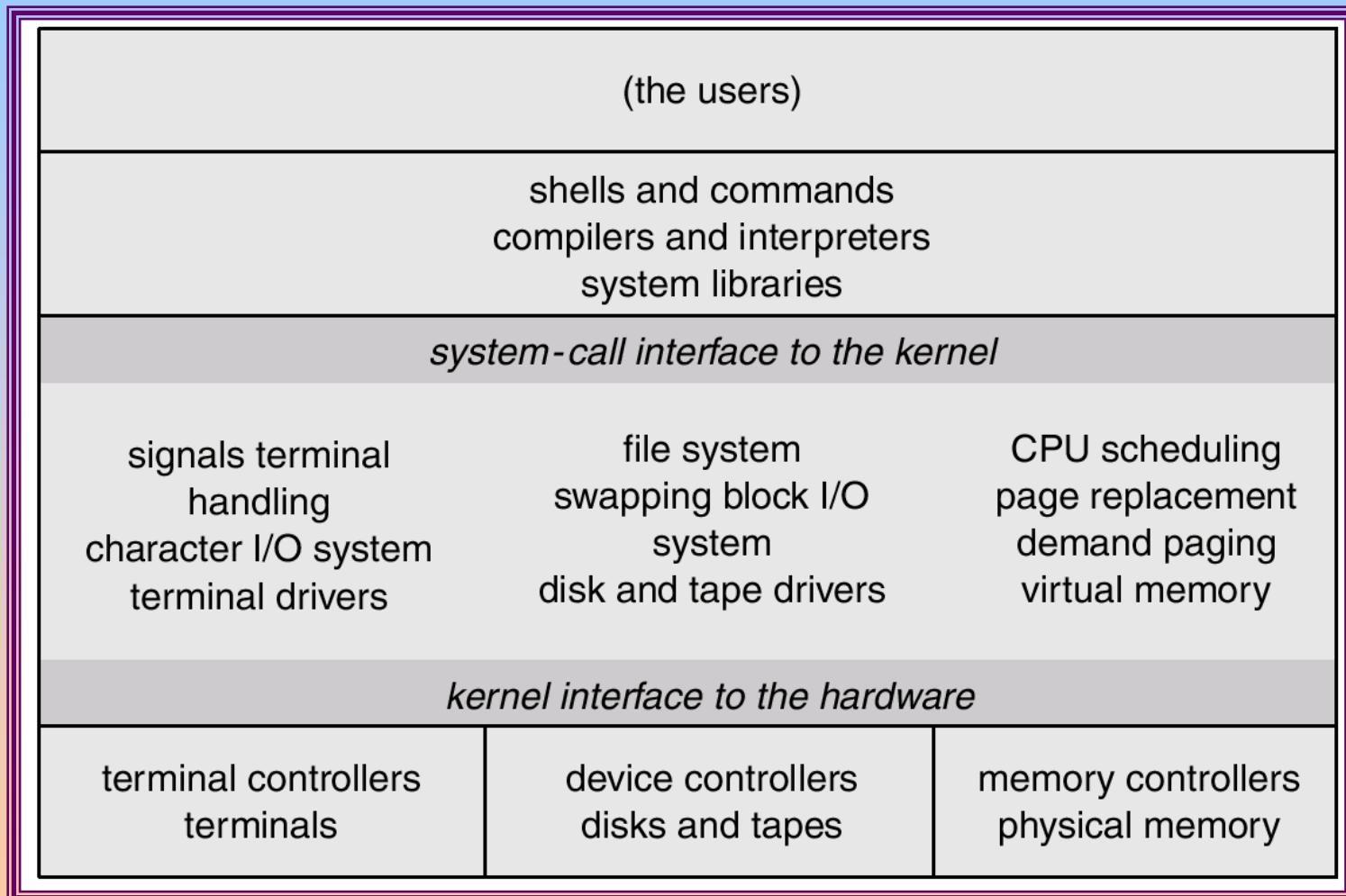


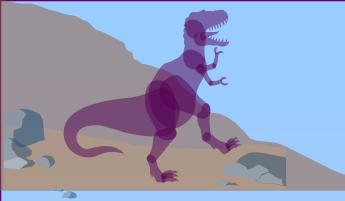
# Struttura di sistema di UNIX

- A causa dei limiti delle architetture per cui era stato progettato, anche la strutturazione di UNIX non risultò completa.
- Il S.O. UNIX consiste di due parti separate:
  - ☞ Programmi di sistema
  - ☞ Kernel
    - ☞ Consiste di tutto ciò che nel diagramma a stati di un sistema è compreso tra l'hardware e l'interfaccia delle chiamate del sistema.
    - ☞ Fornisce il file system, lo scheduling della CPU, la gestione della memoria e altre (forse troppe) funzioni.
    - ☞ Difficile da migliorare: le modifiche in una parte possono avere effetto negativo in un'altra.



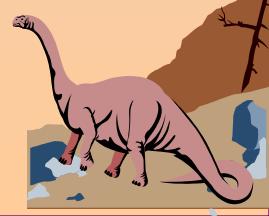
# Struttura di sistema di UNIX (II)

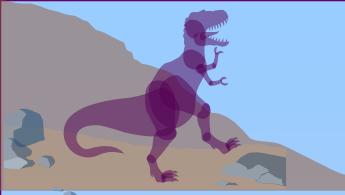




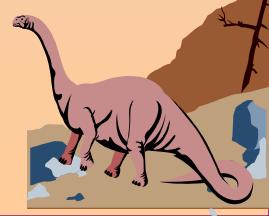
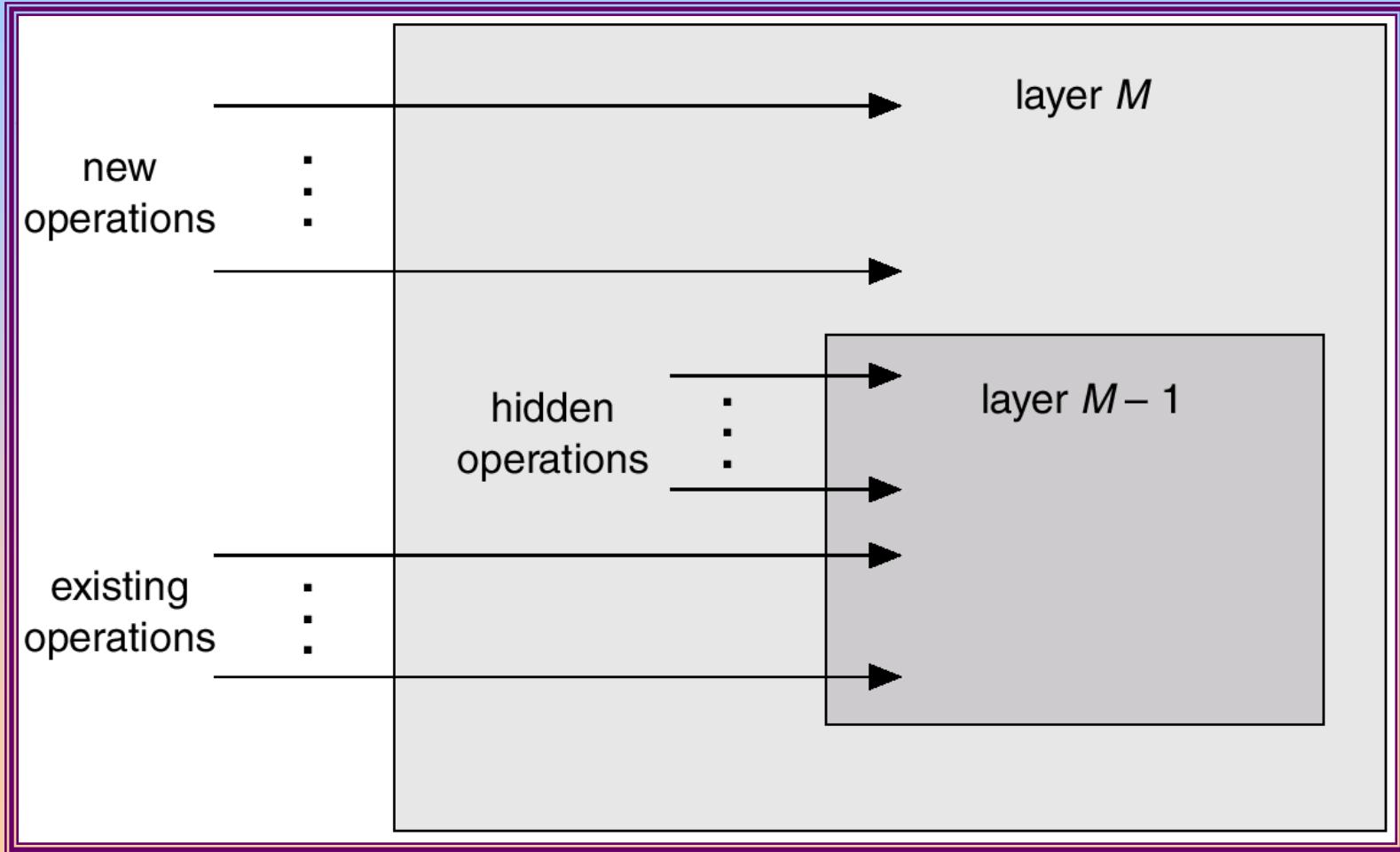
# Metodo stratificato

- In presenza di hardware appropriato si suddivide il S.O. in un certo numero di strati (livelli), ciascuno costruito sopra gli strati inferiori.
- Lo strato più basso (0) è lo strato fisico, quello più alto ( $n$ ) è l'interfaccia utente.
- Gli strati sono composti in modo che ciascuno di essi usi solo funzioni o operazioni e servizi che appartengono a strati di livello inferiore.
- Ogni strato si realizza impiegando unicamente le operazioni messe a disposizione dagli strati inferiori, considerando soltanto le azioni che compiono senza entrare nel merito di come sono realizzate.
- Ogni strato nasconde a quelli superiori l'esistenza di determinate strutture dati, operazioni ed elementi fisici.



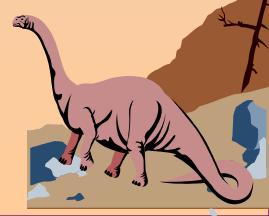


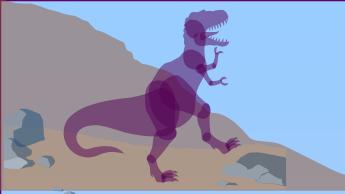
# Uno strato di sistema operativo





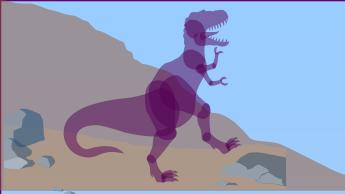
# Micronucleo

- Il kernel è il nucleo di un sistema operativo:
    - ☞ il software che ha il compito di fornire ai processi in esecuzione un accesso sicuro e controllato all'hardware.
  - Il kernel ha anche la responsabilità
    - ☞ di assegnare una porzione di tempo-macchina,
    - ☞ di permettere l'accesso alle risorse hardware a ciascun programma.
  - Nell'approccio di progettazione del S.O. basato su microkernel, si cerca di spostare i servizi dal kernel al livello dei programmi utente.
  - In generale un micronucleo offre i servizi minimi di gestione dei processi, della memoria e di comunicazione.
- 



# Micronucleo (II)

- La “filosofia” del microkernel:
  - ☞ Solo le funzioni assolutamente essenziali del nucleo del sistema operativo dovrebbero essere nel kernel;
  - ☞ I servizi meno essenziali e le applicazioni sono costruiti sopra il microkernel e vengono eseguiti in modalità utente.
- La linea di separazione fra cosa è dentro e cosa è fuori dal microkernel varia da un progetto all’altro.
- La caratteristica comune è che molti servizi che facevano parte del S.O. diventano sottosistemi esterni che interagiscono con il kernel e tra di loro,
  - ☞ Ad es. driver dei dispositivi, file system, gestore della memoria virtuale, sistema a finestre, servizi di sicurezza, etc..
- L’architettura a microkernel sostituisce la tradizionale stratificazione verticale dei SO con una orizzontale.

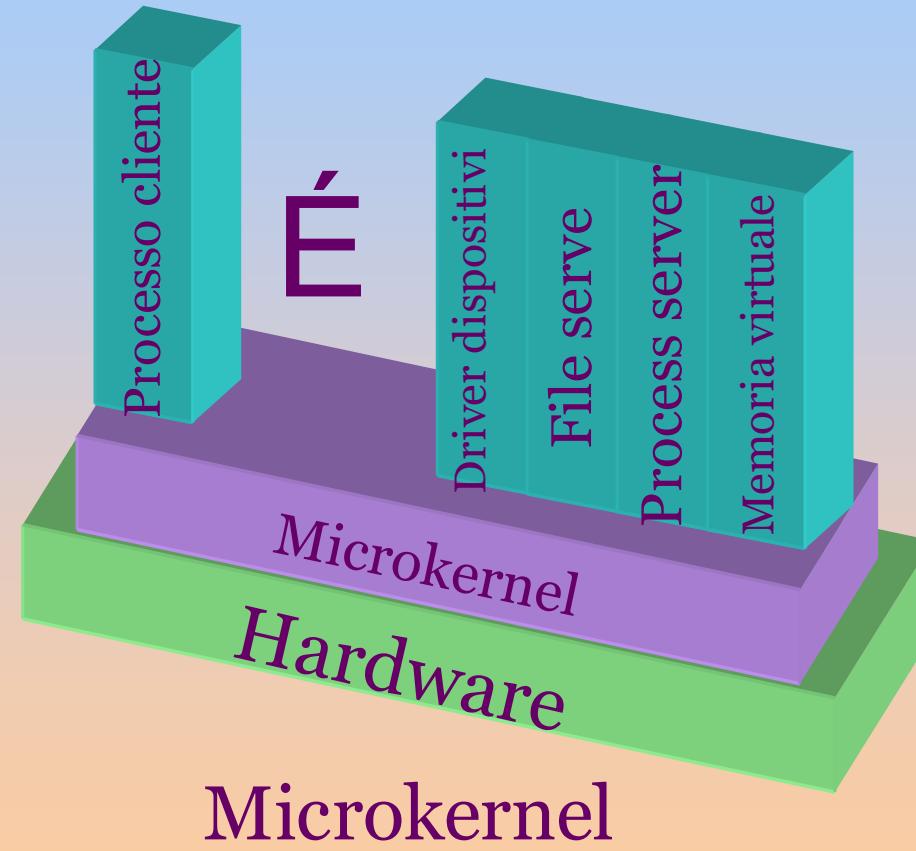


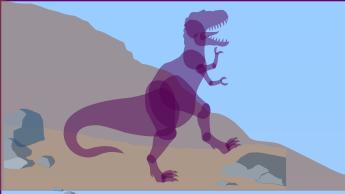
# Micronucleo (III)

- I componenti del SO esterni al microkernel sono implementati come processi server:
  - ☞ interagiscono fra di loro su una base di parità, tramite passaggio di messaggi attraverso il microkernel.
- Il microkernel gestisce lo scambio dei messaggi.
- Scopo principale del micronucleo è fornire funzioni di comunicazione tra i programmi client ed i vari servizi,
  - ☞ anch'essi eseguiti nello spazio utente, tramite scambio di messaggi.
- Vantaggi:
  - ☞ facilità di estensione del sistema (che va fatta nello spazio utente)
  - ☞ S.O. può essere semplicemente adattato a nuove architetture
  - ☞ Più affidabile (meno codice) e sicuro



# Microkernel

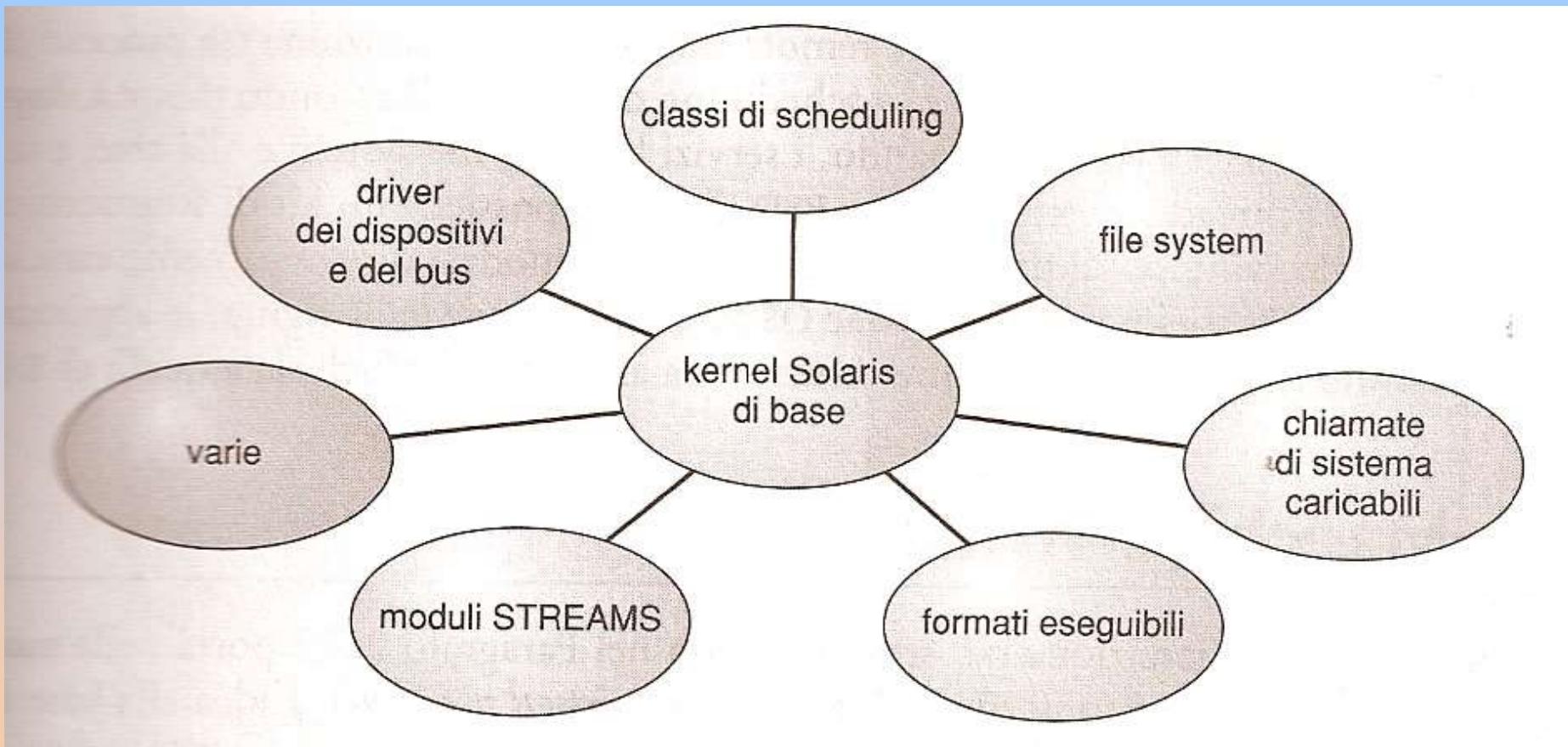




# Moduli

- Molti sistemi operativi moderni implementano un kernel modulare:
    - ☞ usano un approccio object-oriented.
    - ☞ Ogni componente è separata.
    - ☞ Ognuna di essa dialoga con le altre componenti attraverso interfacce.
    - ☞ Ciascuna di esse può essere caricata quando necessaria all'interno del kernel.
    - ☞ L'approccio dei moduli è molto simile a quello stratificato (livelli) ma presenta maggiore flessibilità.
-  Ad es. Solaris.

# Solaris: moduli



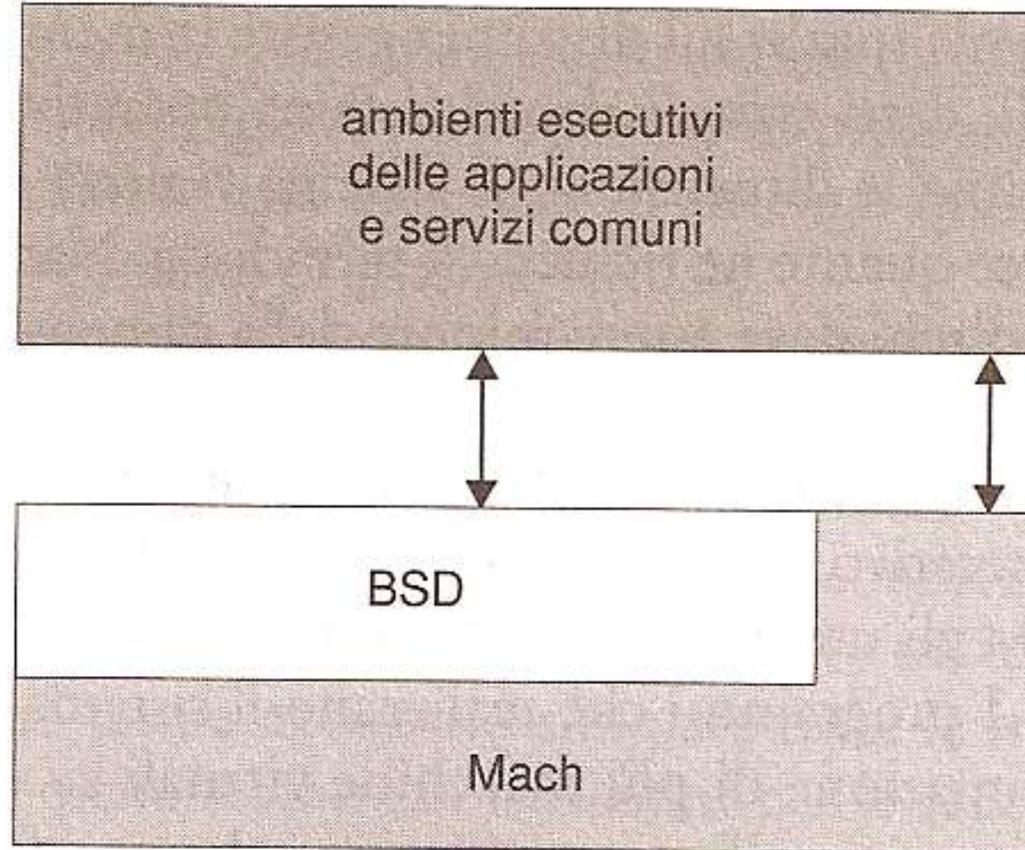


# Mac OS X

- Il S.O. Mac Os X (Darwin) adotta una struttura ibrida.
    - ☞ E' organizzato in strati, uno dei quali contiene il microkernel Mach.
  - Gli strati superiori comprendono gli ambienti esecutivi delle applicazioni ed un insieme di servizi che offre un'interfaccia grafica per le applicazioni.
  - Il kernel si trova in uno strato sottostante ed è costituito dal microkernel Mach e dal kernel BSD.
    - ☞ Il primo cura la gestione della memoria, le chiamate di procedure remote (RPC), la comunicazione tra processi - compreso lo scambio di messaggi - e lo scheduling dei thread.
    - ☞ Il secondo mette a disposizione un'interfaccia BSD a riga di comando, i servizi legati al file system e alla rete, e le API POSIX, compreso Pthreads.
  - Oltre a Mach e BSD, il kernel possiede un kit di strumenti connessi all'I/O per lo sviluppo di driver dei dispositivi e di moduli dinamicamente caricabili detti estensioni del kernel nel gergo di Mac OS X.
  - Le applicazioni ed i servizi comuni possono accedere direttamente sia ai servizi di Mach sia a quelli di BSD.
- 

# Mac OS X (II)

ambiente kernel

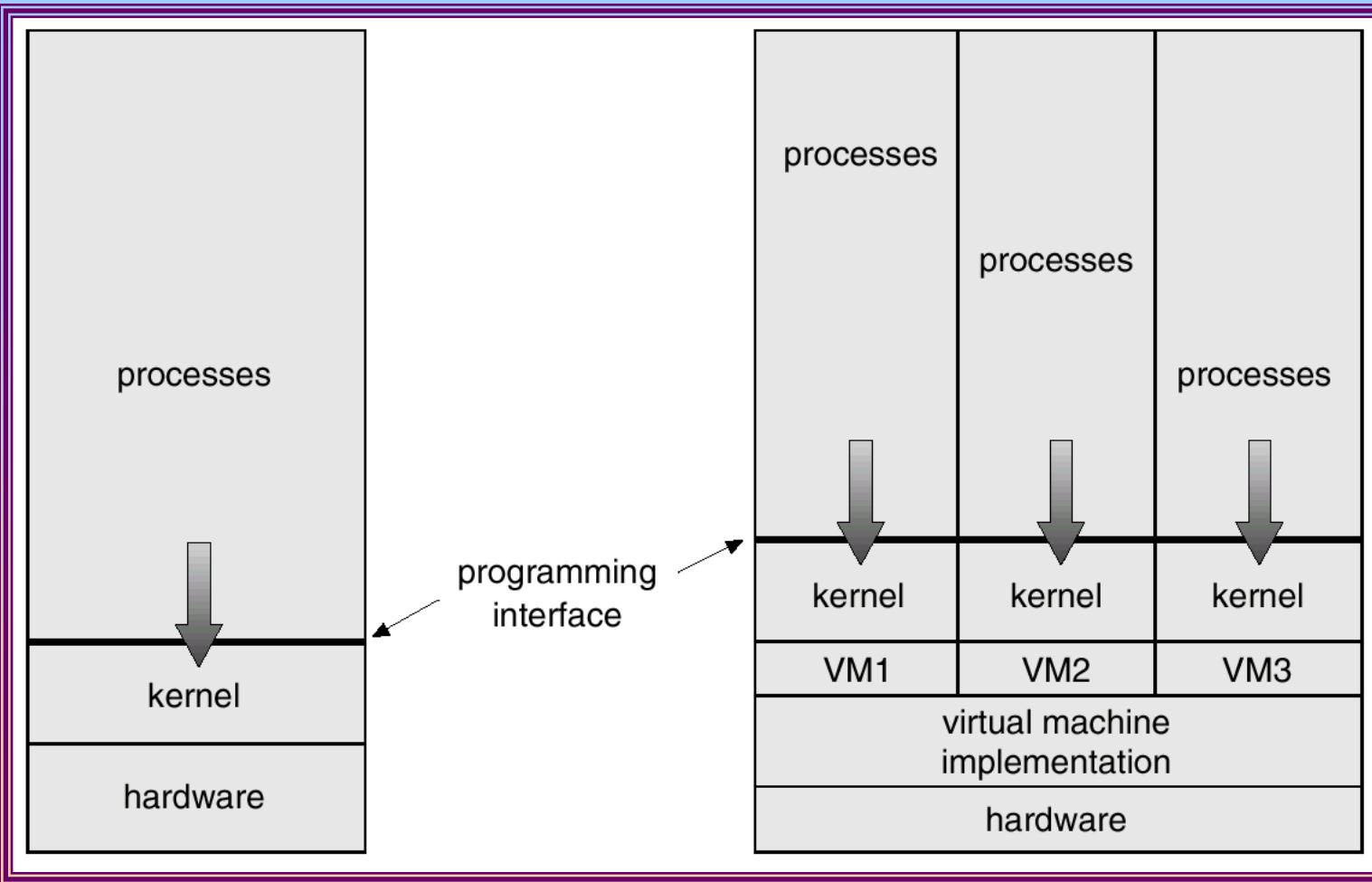




# Macchine virtuali

- Una *macchina virtuale* tratta hardware e S.O. come se fossero entrambi hardware.
  - S.O. crea l'illusione di processi multipli, ciascuno in esecuzione con un suo processore (virtuale) e la sua memoria (virtuale).
  - Le risorse del sistema vengono condivise per creare le macchine virtuali:
    - ☞ Lo scheduling della CPU crea l' illusione che ciascun utente disponga della propria CPU e memoria.
    - ☞ I meccanismi di spooling ed il file system possono fornire dispositivi di I/O “virtuali”.
    - ☞ Un normale terminale utente serve come “console virtuale” della macchina virtuale.
- 

# Modelli di sistema



Semplice

Macchina virtuale

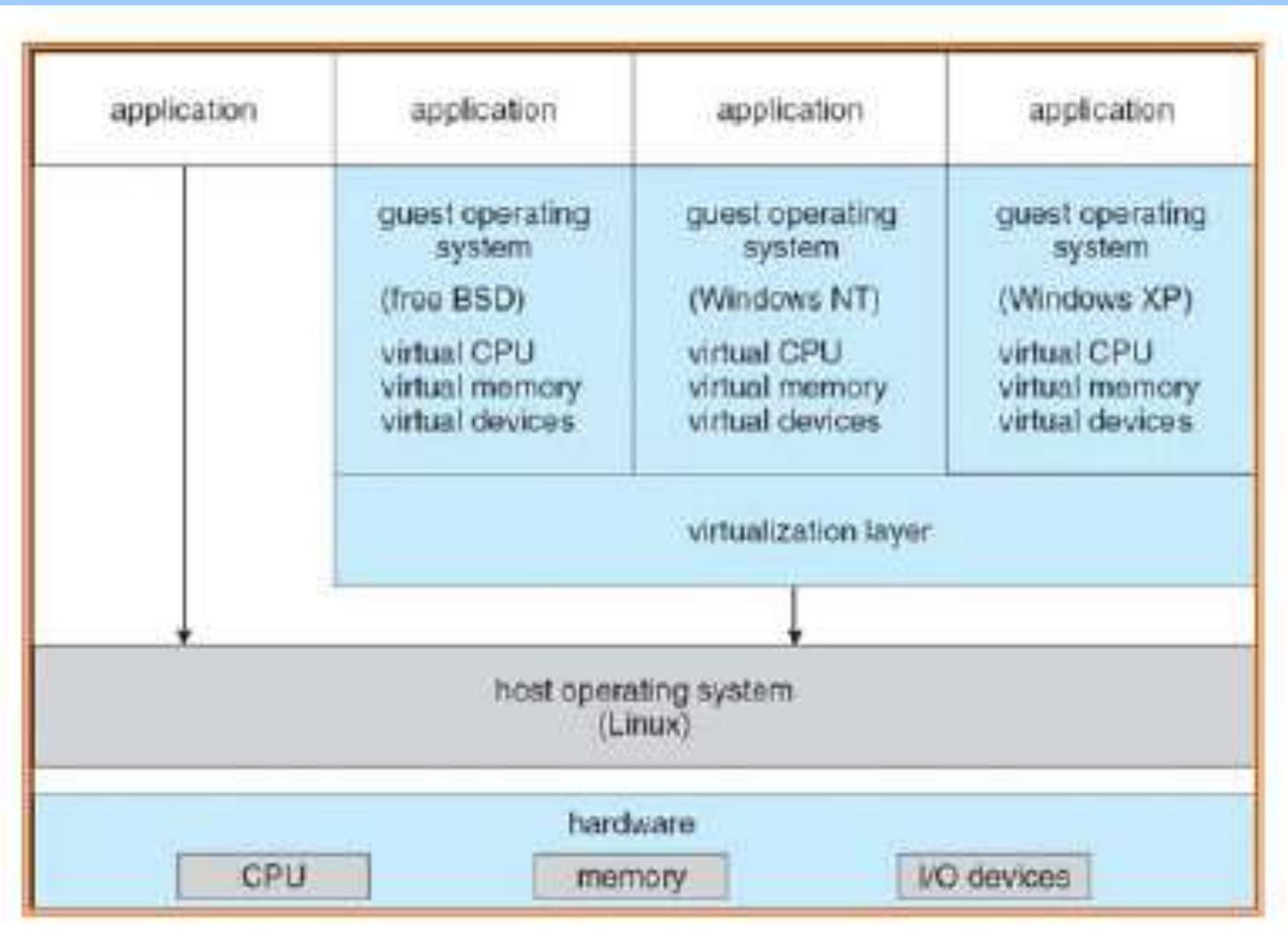


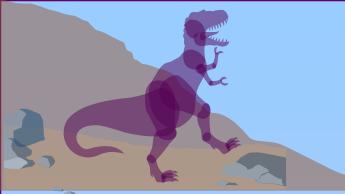
# Vantaggi e svantaggi delle macchine virtuali

- Il concetto di macchina virtuale provvede una completa protezione delle risorse del sistema:
  - ☞ ciascuna macchina virtuale è isolata dalle altre.
- Questo isolamento non permette però la condivisione diretta di risorse.
- Una macchina virtuale è un ottimo strumento per la ricerca e lo sviluppo di nuovi S.O.
  - ☞ Lo sviluppo di un nuovo S.O. può essere fatto su una macchina virtuale, senza interrompere le normali operazioni del sistema).
- Non è semplice implementare macchine virtuali a causa della necessità di fornire un esatto duplicato della architettura fisica sottostante.

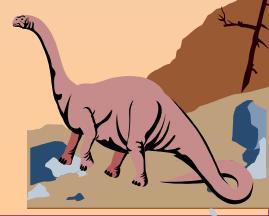


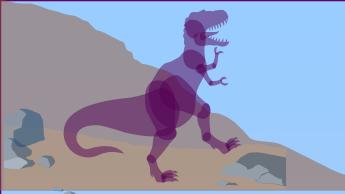
# VMware



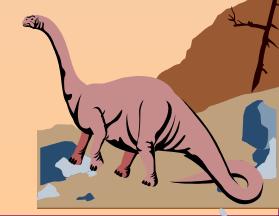
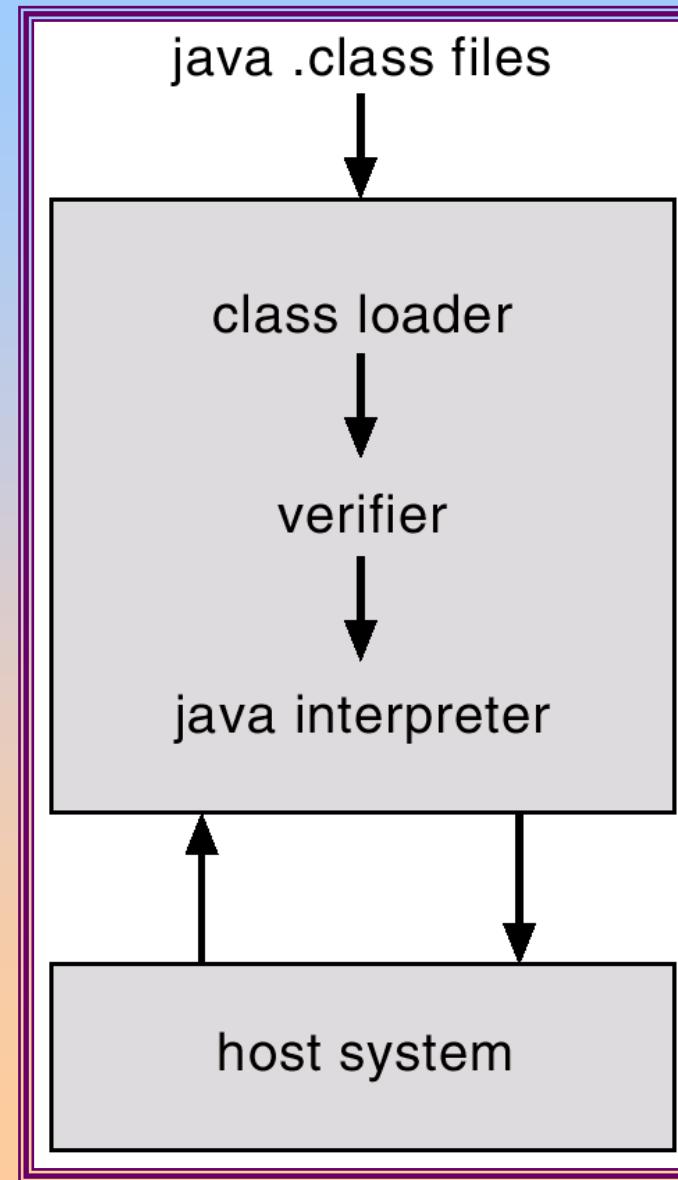


# Java

- I programmi Java vengono compilati in un formato “neutro” che verrà eseguito da una macchina virtuale Java (Java Virtual Machine - JVM).
  - JVM consiste in:
    - caricatore delle classi
    - verificatore delle classi
    - interprete del linguaggio
  - Compilatori Just-In-Time (JIT) possono migliorare le prestazioni
- 



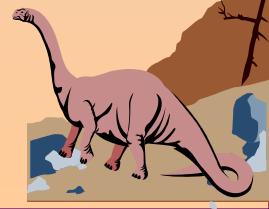
# La macchina virtuale Java

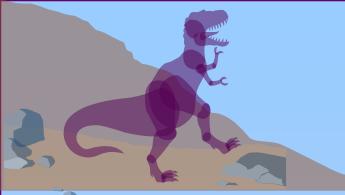




# Generazione di sistemi (SYSGEN)

- I S.O. sono disegnati per macchine di una stessa classe:
  - ☞ il sistema deve essere configurato per ciascuna specifica architettura.
- Il processo di generazione di sistemi (SYSGEN) recupera informazioni relative alla configurazione hardware e software specifica:
  - ☞ CPU, quantità di memoria disponibile, dispositivi disponibili, opzioni del S.O. richieste per i valori dei parametri che è necessario usare, etc..
- Approcci alla configurazione:
  - ☞ modifica del codice sorgente e ricompilazione,
  - ☞ selezione del “sottoinsieme” utile (codice precompilato),
  - ☞ selezione dinamica tramite tabella (cod. precompilato)





# Avvio del sistema

- La procedura di avviamento di un calcolatore attraverso il caricamento del kernel è nota come avviamento (booting) del sistema.
- Nella maggior parte dei sistemi esiste una piccola porzione di codice – programma di avvio (bootstrap loader) - che localizza il kernel, lo carica in memoria centrale, e ne avvia l'esecuzione.
- Qualche volta il processo è diviso in due fasi:
  - ☞ un blocco di boot in una locazione fissa del disco contiene il bootstrap loader.
  - ☞ Quando il computer viene acceso, l'esecuzione parte da una locazione fissa di memoria e
  - ☞ del firmware (ROM, EPROM) viene usato per caricare in memoria il blocco di boot.

