Università di Bologna
Laurea in Ingegneria Informatica
A.A. 2012-2013

Sistemi Operativi T

Informatica Industriale M

Prof. Anna Ciampolini

http://lia.deis.unibo.it/Courses/sot1213

Obiettivi del Corso

- Fornire i principali concetti alla base ***dei moderni sistemi operativi***
- Fornire i fondamenti e gli strumenti di base per **la programmazione concorrente**
- Illustrare le caratteristiche di alcuni ***sistemi operativi reali (UNIX, GNU/Linux e Windows XP)*** e gli strumenti a disposizione di utenti e programmatori per il loro utilizzo
- Sperimentare ampiamente in ***laboratorio*** i concetti e gli strumenti visti in aula

Capacità richieste in ingresso:

- conoscenza dei ***linguaggi C e Java***
- fondamenti di architettura degli elaboratori

Capacità ottenute in uscita:

- ***conoscenza dei concetti*** alla base dei sistemi operativi moderni
- conoscenza delle soluzioni realizzative adottate nei più diffusi sistemi operativi moderni
- capacità di ***sviluppare programmi di sistema*** e applicazioni principalmente in ambiente UNIX/ Linux.
- capacità di sviluppo di applicazioni concorrenti in **java**

Argomenti trattati

- Che cos'è un sistema operativo: ruolo e funzionalità
- ***Organizzazione*** e ***struttura*** di un sistema operativo
- ***Gestione dei Processi***
- Interazione tra processi mediante ***memoria condivisa*** e ***scambio di messaggi***
- Cenni di ***sincronizzazione*** dei processi
- Gestione della ***memoria***
- Gestione del ***file system***
- Gestione dei dispositivi di ***Input/Output***
- ***Protezione delle risorse***
- ***Metodi e strumenti per la programmazione concorrente.***

Panoramica sul Corso

Introduzione:

- Che cos'è un sistema operativo: ruolo, funzionalità e struttura
- Evoluzione dei sistemi operativi: ***batch***, ***multiprogrammazione***, ***time-sharing***
- Richiami sul funzionamento di un elaboratore: ***interruzioni e loro gestione, I/O, modi di funzionamento single e dual, system call***

Panoramica sul Corso

Organizzazione di un sistema operativo:

- Funzionalità
- Classificazione in base a struttura: sistemi ***monolitici***, ***modulari***, sistemi ***stratificati***, ***microkernel***, ***macchine virtuali***
- Cenni introduttivi di organizzazione e funzionalità di alcuni sistemi operativi reali (***UNIX/Linux, Windows, ecc.)***

Panoramica sul Corso

Gestione dei Processi:

- ***Concetto di processo*** e sua ***rappresentazione*** nel sistema operativo:
  - ✓ Processi pesanti
  - ✓ Processi leggeri (thread)
- ***Stati e ciclo di vita dei processi***
- ***Gestione dei processi pesanti/leggeri*** da parte del SO
- Operazioni sui processi
- La gestione dei ***processi in UNIX/Linux***: stati, rappresentazione, gestione (scheduling), operazioni e comandi relativi ai processi
- ***Java thread e memoria condivisa***

Panoramica sul Corso

Scheduling della CPU :

- Concetti generali: ***code, preemption, dispatcher***
- ***Criteri*** di scheduling
- ***Algoritmi di scheduling***: FCFS, SJF, con priorità, round-robin, con code multiple, …
- Scheduling in s.o. reali: Unix, Linux e Windows

Panoramica sul Corso

Interazione tra processi:

- **Mediante memoria condivisa**

  Cenni sul problema della ***sincronizzazione tra processi***. Il semaforo.

- **Mediante scambio di messaggi**
  - ***Comunicazione*** diretta/indiretta,simmetrica/ asimmetrica, buffering
  - ***Interazione tra processi UNIX***: comunicazione mediante pipe e fifo, sincronizzazione tramite segnali

Panoramica sul Corso

Gestione della memoria:

- Spazi degli indirizzi e binding
- Allocazione della memoria
  - ***Contigua***: a partizione singola e partizioni multiple; frammentazione;
  - ***Non contigua***: paginazione, segmentazione
- ***Memoria virtuale***
- Gestione della memoria in UNIX e GNU/ Linux.

Panoramica sul Corso

Gestione dei dispositivi di I/O:

- Il sottosistema di I/O: funzioni e meccanismi.
- Driver di dispositivi.

Gestione del file system:

- File system e sua realizzazione. Metodi di accesso e di allocazione.
- Il file system di UNIX: organizzazione logica e fisica, comandi e *system call* per la gestione e l'accesso a file/direttori

Panoramica sul Corso

Protezione:

- Modelli, politiche e meccanismi di protezione.
- *Domini di protezione*
- Matrice di accesso
- Controllo degli accessi
- Sistemi basati su ***capability***

Panoramica sul Corso

Programmazione concorrente:

- Algoritmi non sequenziali e programmi concorrenti
- Proprietà di programmi concorrenti
- Il deadlock
- Il problema della mutua esclusione
- Strumenti linguistici per la sincronizzazione nel modello a memoria comune: il monitor
- Monitor in java

Percorso didattico

- **Argomenti teorici**
- **Esemplificazioni**: sui sistemi operativi UNIX/Linux:
  - ***programmazione di sistema in linguaggio C***
  - sviluppo di ***file comandi in shell***
  - ***applicazioni concorrenti in Java***
- **Esercitazioni**:

Attività in laboratorio

Attività in laboratorio

- Esattamente come le lezioni in aula, è ***parte integrante dell'attività didattica!***

- Ogni settimana (a regime) verrà svolta in Lab4 una esercitazione su argomenti trattati in aula.

- L'attività sarà assistita da un **tutor**.

Attività in laboratorio

- **Programma:**
  - Uso e amministrazione del sistema GNU/linux
  - Comandi shell: uso e realizzazione di file comandi (shell scripting)
  - System Calls: realizzazione di programmi di sistema in linux:
    - Gestione di processi
    - Sincronizzazione tra processi
    - Gestione di file e direttori
  - Programmazione concorrente:
    - Semafori
    - Lock
    - Monitor
    - variabili condizione.

Accesso al Laboratorio

- L'attività si svolgerà in Lab4 su sistemi Linux Ubuntu.

Esame

Una ***prova "scritta"*** obbligatoria (in parte teorica e in parte progettuale) in Lab4:

- 10 giugno 2013 ore 9.30
- 27 giugno 2013 ore 9.30
- 11 luglio 2013 ore 9.30

- Una ***prova orale:***
  - **Facoltativa**, se il voto dello scritto è **maggiore di 22/30**
  - **Obbligatoria**, se il voto delle prova scritta dopo il superamento dello scritto è **minore o uguale a 22/30**

Materiale Didattico

- **Copia** delle diapositive mostrate a lezione (scaricabili dalle pagine Web del corso)
- **Libro adottato:**
  - P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari: ***Sistemi Operativi*** (seconda edizione), McGraw-Hill, 2008.
- **Libri consigliati**:
  - A. Silbershatz, P.B. Galvin, G. Gagne: ***Sistemi Operativi – Concetti ed Esempi*** (settima edizione), Pearson, 2006
  - A. Tanenbaum: ***I Moderni Sistemi Operativi***, Jackson Libri, 2002
  - H.M. Deitel, P.J. Deitel, D.R. Choffnes: ***Sistemi Operativi***, Pearson, 2005
  - W. Stallings: ***Sistemi Operativi***, Jackson Libri, 2000
  - K. Havilland, B. Salama: ***Unix System Programming***, Addison Wesley, 1987

Ricevimento Studenti

- **Anna Ciampolini**

  martedì ore 11:00-13:00

  c/o Deis, Edificio aule nuove, piano 2.

  E-mail: **anna.ciampolini@unibo.it**

Orario delle Lezioni

Normalmente:

- **Lun 14-16, Lab4: Esercitazione (a regime).**
- **Mar 13-16, aula 2.7B**
- **Mer 14-16, aula 2.6**

- Eventuali variazioni verranno comunicate via sito Web.

Introduzione ai Sistemi Operativi

Che cos'è un Sistema Operativo (SO)?

È un ***programma*** (o un insieme di programmi) che agisce come **intermediario** tra l'utente e l'hardware del computer:

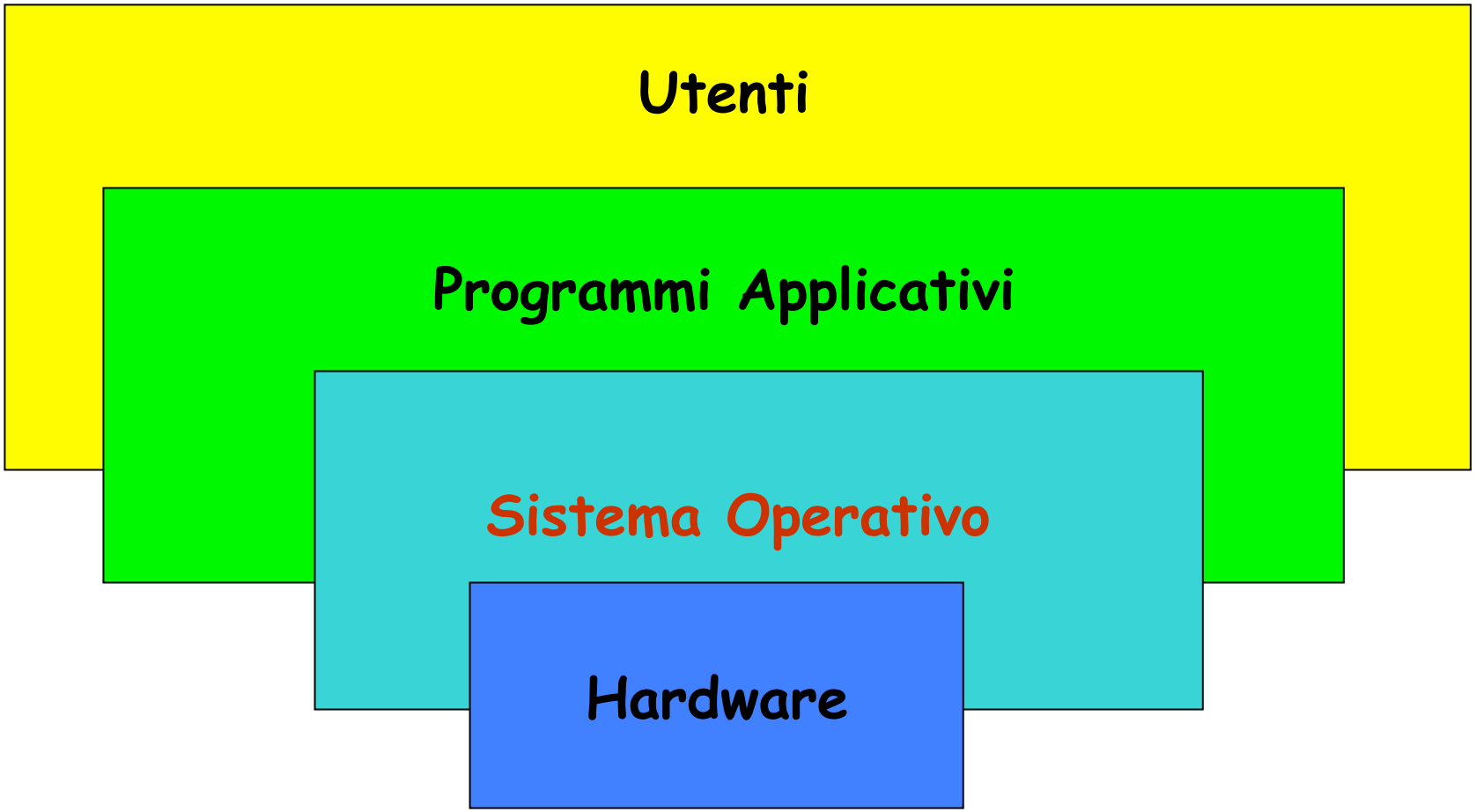
- fornisce una **visione astratta e semplificata** dell'HW
- mette a disposizione un **ambiente di esecuzione e di sviluppo** per i programmi degli utenti
- **gestisce** in modo **efficace** ed **efficiente** le risorse del sistema di calcolo

SO e Hardware

- Il SO interfaccia programmi applicativi o di sistema con le risorse HW:
  - **CPU**
  - **memoria** centrale (RAM)
  - **memoria** secondaria (es. Dischi)
  - **dispositivi** di I/O
  - **connessioni di rete**
  - …

- Il SO *mappa* le risorse HW in **risorse logiche**, accessibili attraverso interfacce ben definite:
  - ***processi*** (CPU)
  - ***file system*** (dischi)
  - ***memoria virtuale*** (memoria), …

Che cos'è un Sistema Operativo?



Che cos'è un Sistema Operativo?

- Un programma che *gestisce risorse* del sistema di calcolo in modo *efficace ed efficiente* e le *alloca* ai programmi/utenti

- Un programma che innalza il *livello di astrazione* con cui utilizzare le *risorse logiche* a disposizione

Aspetti importanti di un SO

- **Architettura**: come è organizzato il SO? Quali componenti? Quali relazioni tra componenti?
- **Condivisione**: quali risorse vengono condivise tra utenti e/o programmi? In che modo?
- **Efficienza**: come massimizzare l'utilizzo delle risorse disponibili?
- **Affidabilità/tolleranza ai guasti**: quale probabilità di malfunzionamenti? come reagisce il SO ad eventuali malfunzionamenti (HW/SW)?
- **Estendibilità**: è possibile aggiungere funzionalità al sistema?
- **Protezione** e **Sicurezza**: il SO deve impedire ***interferenze*** tra programmi/utenti e attacchi dalla rete. In che modo?
- **Conformità a standard**: portabilità, estendibilità, apertura

Alcuni cenni storici sull'evoluzione dei SO

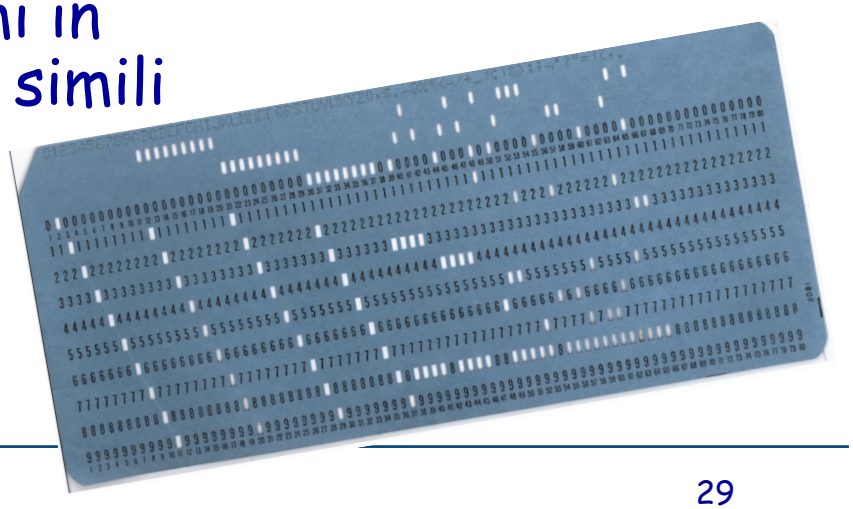
Prima generazione (anni '50)

- Architettura basata su **valvole**
- **linguaggio macchina**
- controllo del sistema completamente manuale
- non è presente il sistema operativo

Seconda generazione ('55-'65):

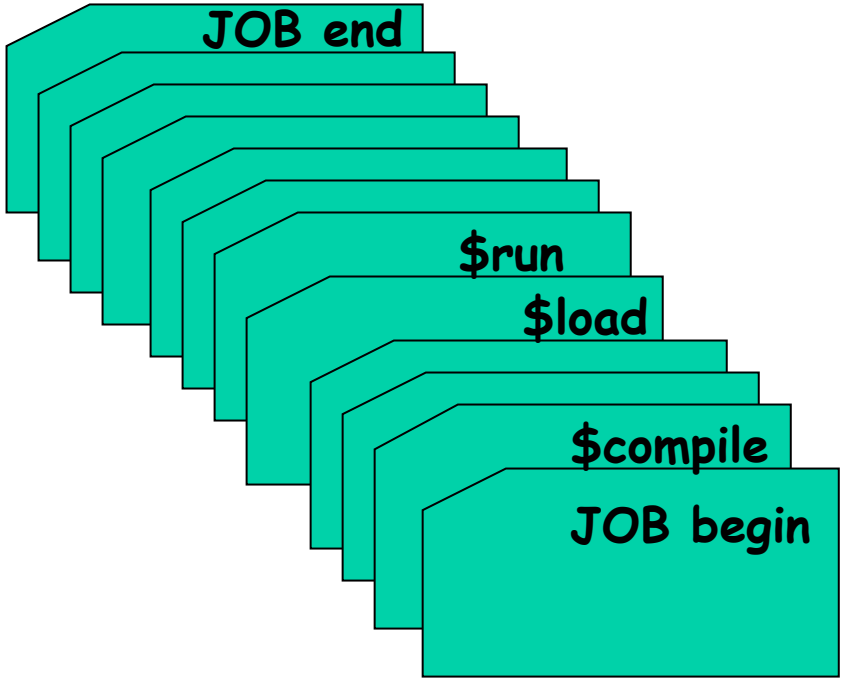
Sistemi batch semplici

- Architettura basata su **transistor**
- **linguaggio di alto livello** (*fortran*)
- input mediante schede perforate
- aggregazione di programmi in **lotti** (batch) con esigenze simili



Sistemi batch semplici

Batch: insieme di programmi (*job*) da eseguire **in modo sequenziale**



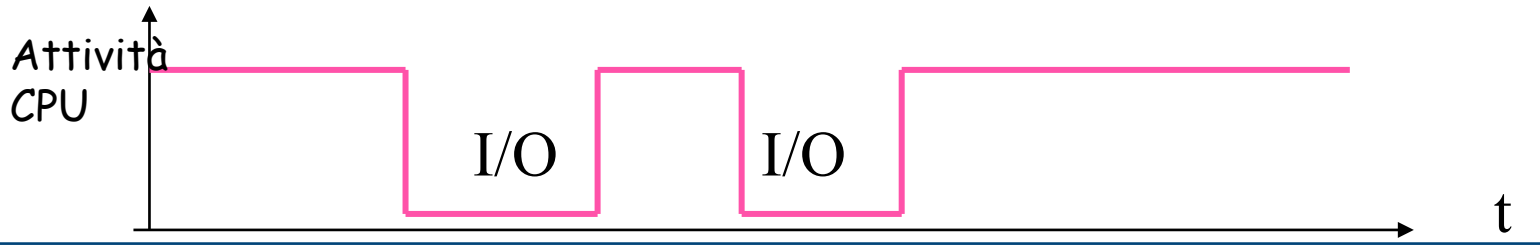
Sistemi batch semplici

Compito del SO (*monitor*):

trasferimento di controllo da un job (appena terminato) al prossimo da eseguire

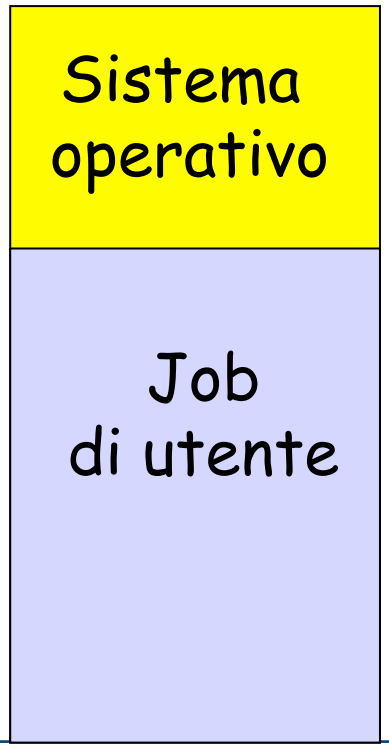
Caratteristiche dei sistemi batch semplici:

- ***SO residente in memoria*** (monitor)
- ***assenza di interazione*** tra utente e job
- ***scarsa efficienza***: durante l'I/O del job corrente, la CPU rimane inattiva !



Sistemi batch semplici

In memoria centrale, ad ogni istante, è *caricato (al più) un solo job*:



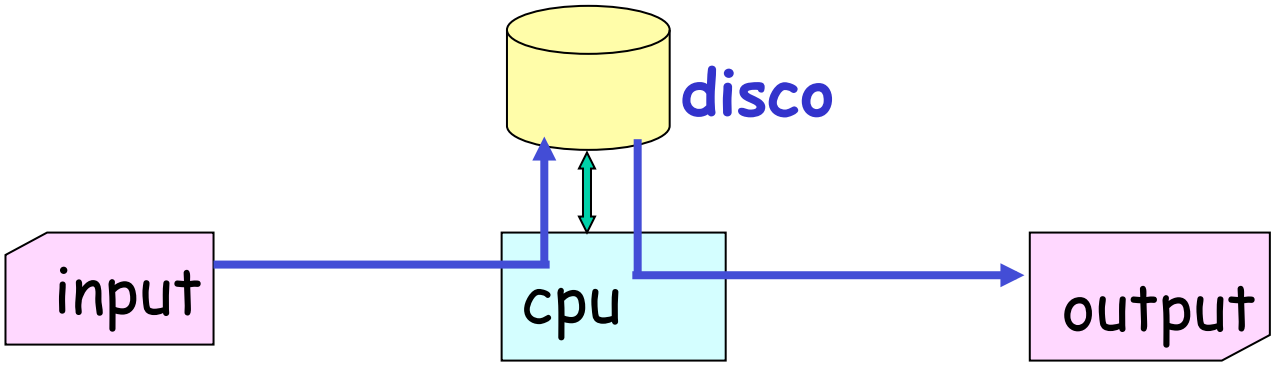
Configurazione della memoria centrale in sistemi batch semplici

Sistemi batch semplici

Spooling (Simultaneous Peripheral Operation On Line): simultaneità di I/O e attività di CPU

disco viene impiegato come **buffer** molto ampio, dove

- ***memorizzare*** in anticipo i programmi da eseguire
- **leggere** in anticipo i dati
- ***memorizzare*** temporaneamente i risultati (in attesa che il dispositivo di output sia pronto)
- caricare ***codice e dati del job successivo***: -> possibilità di **sovrapporre I/O** di un job **con elaborazione** di un altro job



Sistemi batch semplici

Problemi:

- finché il job corrente non è terminato, il ***successivo non può iniziare l'esecuzione***
- se un job si ***sospende*** in attesa di un evento, la CPU rimane ***inattiva***
- ***non c'è interazione*** con l'utente

Sistemi batch multiprogrammati

Sistemi batch semplici: ***l'attesa*** di un ***evento*** causa inattività della CPU. Per evitare il problema:

Multiprogrammazione

- Viene precaricato sul disco un insieme (***pool***) di job
- Il SO seleziona un sottoinsieme dei job appartenenti al pool, che vengono caricati in memoria centrale
- Tra i job in memoria, il SO ne sceglie uno (job corrente) a cui assegnare la CPU
- Se il job corrente si pone in attesa di un evento, il sistema operativo assegna CPU a un altro job

Sistemi batch multiprogrammati

SO è in grado di ***portare avanti*** l'esecuzione di più job ***contemporaneamente.***

- Ad ogni istante:
  - **un solo job** utilizza la CPU
  - **più job**, appartenenti al pool selezionato e caricati in memoria centrale, attendono di acquisire la CPU

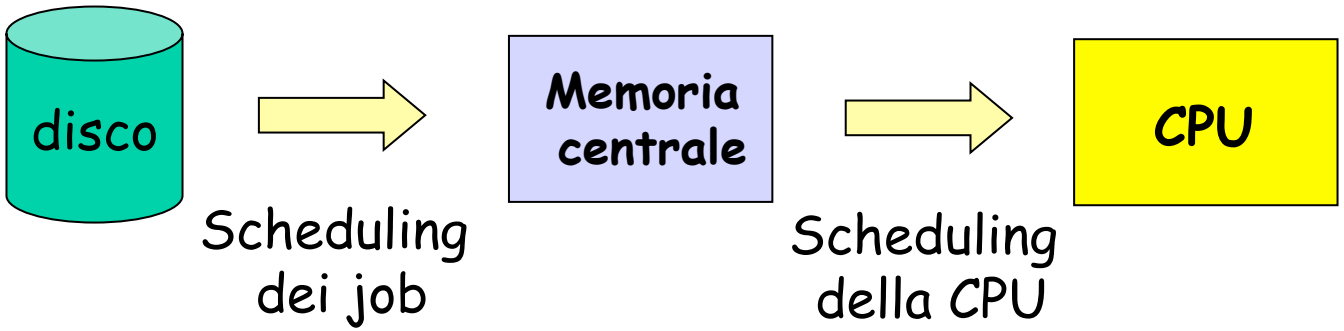
- Quando il job che sta utilizzando la CPU si ***sospende in attesa di un evento***:
  - SO ***decide*** a quale job assegnare la CPU (***scheduling***)

  ed effettua lo scambio (***context switch***)

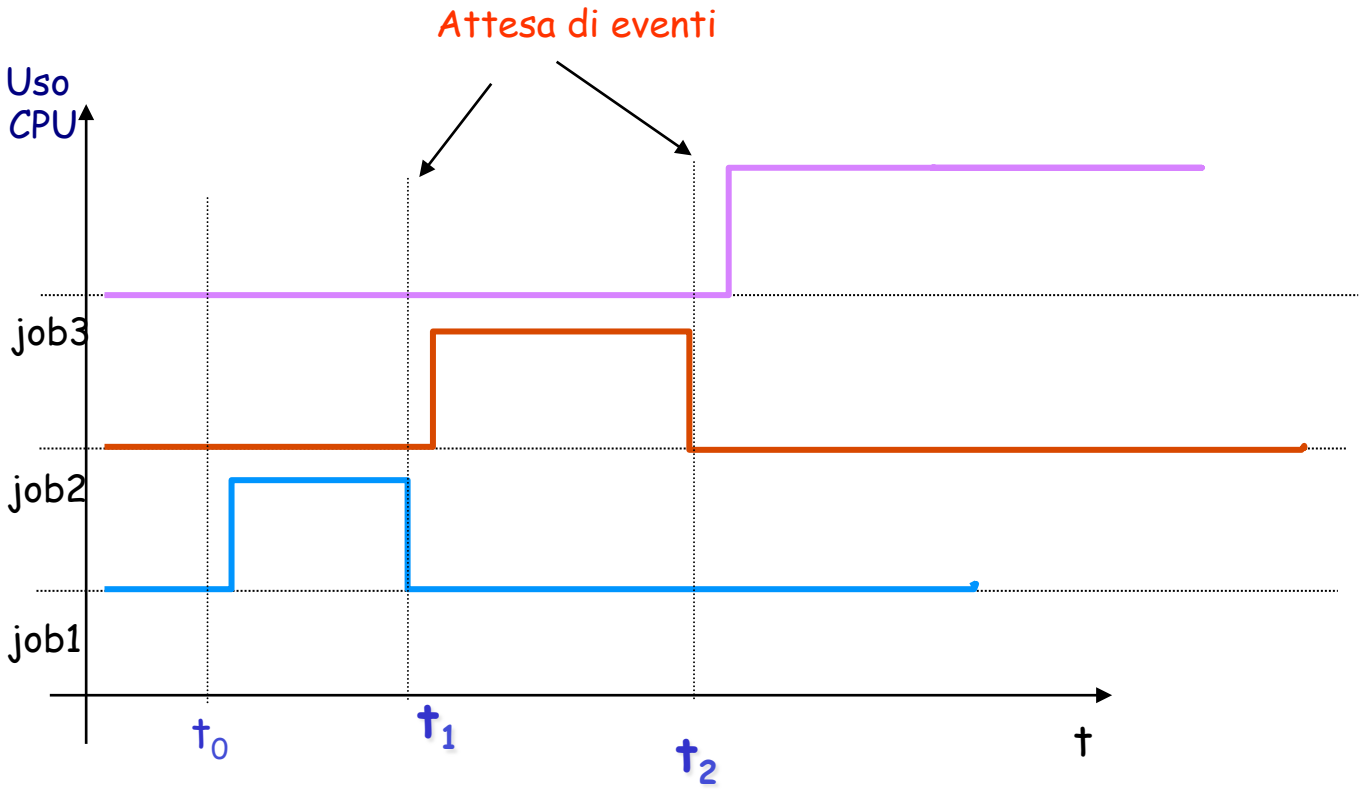
Sistemi batch multiprogrammati: scheduling

SO effettua delle scelte tra tutti i job:

- quali job caricare in memoria centrale: **scheduling dei job** (*long-term scheduling*)
- a quale job assegnare la CPU: **scheduling della CPU** o (*short-term scheduling*)

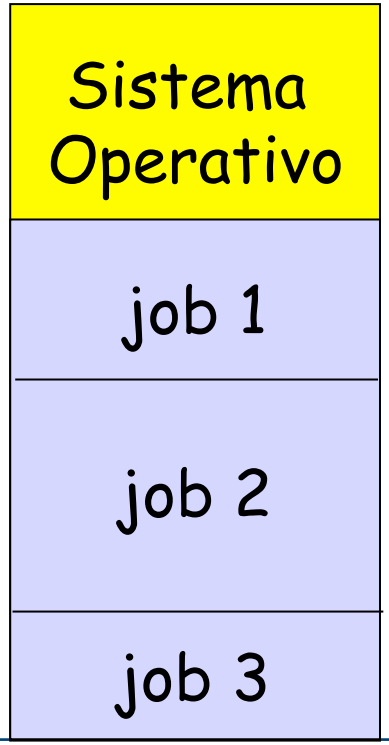


Sistemi batch multiprogrammati



Sistemi batch multiprogrammati

In memoria centrale, ad ogni istante, possono essere caricati più job:



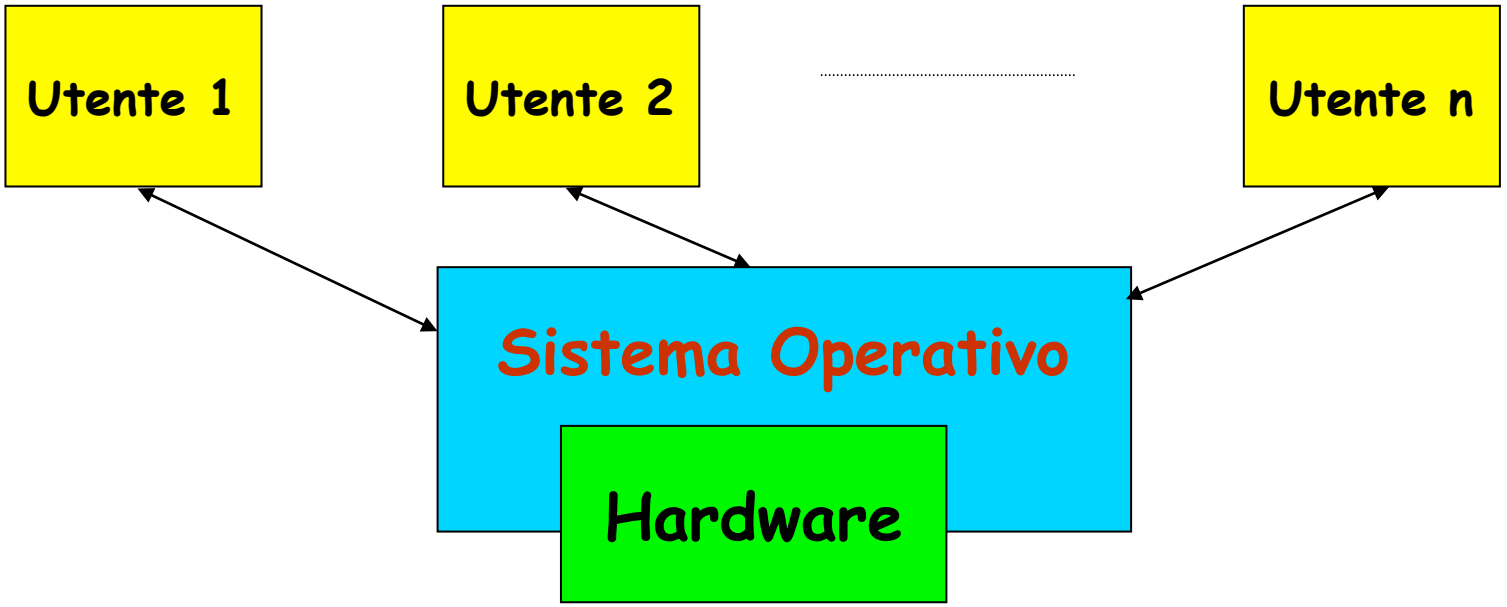
Configurazione della ***memoria centrale*** in sistemi batch multiprogrammati

Necessità di protezione

Sistemi time-sharing (Multics, 1965)

Nascono dalla necessità di:

- *interattività* con l'utente
- *multi-utenza*: più utenti interagiscono contemporaneamente con SO



Sistemi time-sharing

Multiutenza: il sistema presenta ad ogni utente una ***macchina virtuale completamente dedicata*** in termini di:

- utilizzo della CPU
- utilizzo di altre risorse, ad es. file system

Interattività: per garantire un'accettabile velocità di "reazione" alle richieste dei singoli utenti, SO ***interrompe l'esecuzione*** di ogni job dopo un intervallo di tempo prefissato (***quanto di tempo***, o ***time slice***), assegnando la CPU a un altro job

Sistemi time-sharing

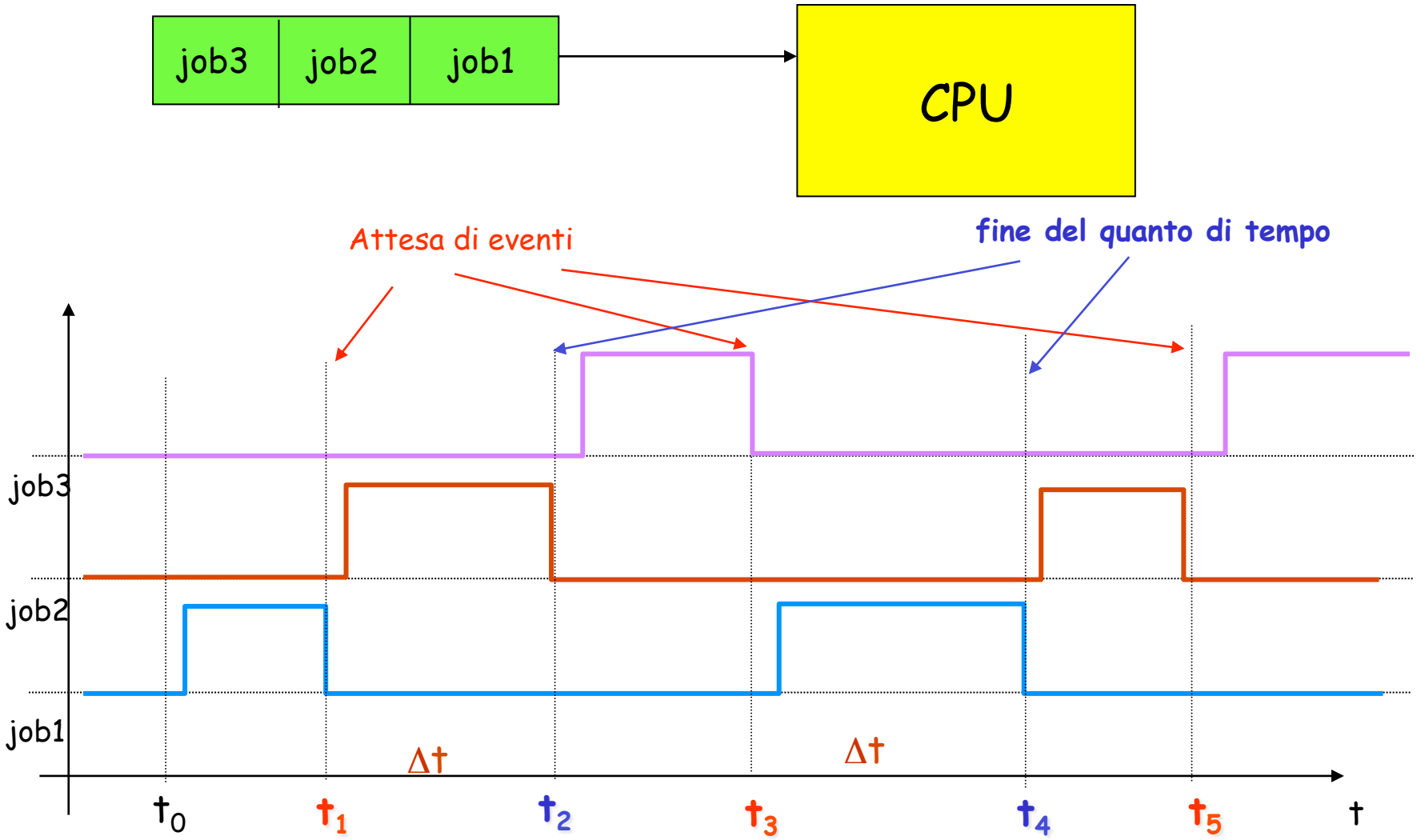
Sono sistemi in cui:

- attività della ***CPU è dedicata a job diversi*** che si alternano ***ciclicamente*** nell'uso della risorsa
- frequenza di commutazione della CPU è tale da fornire l'illusione ai vari utenti di una macchina completamente dedicata (***macchina virtuale***)

Cambio di contesto (***context switch***):

operazione di trasferimento del controllo da un job al successivo -> costo aggiuntivo (*overhead*)

Sistemi time-sharing



Time-sharing: requisiti

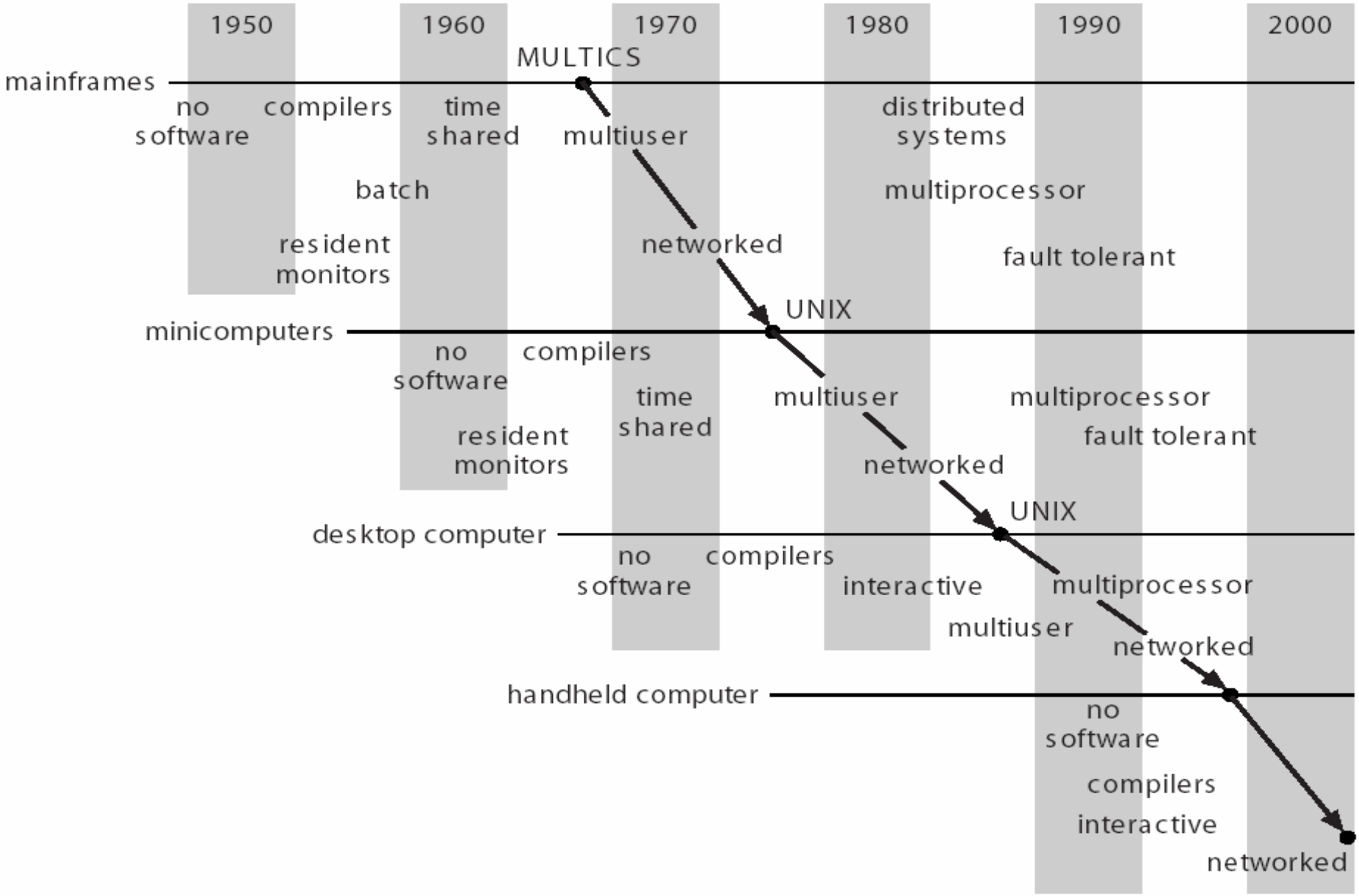
- **Gestione/protezione** della memoria:
  - trasferimenti memoria-disco
  - ***separazione degli spazi*** assegnati ai diversi job
  - molteplicità job + limitatezza della memoria

    memoria virtuale
- **Scheduling** CPU
- **Sincronizzazione/comunicazione** tra job:
  - interazione
  - prevenzione/trattamento di blocchi critici (***deadlock***)
- **Interattività**: ***accesso on-line*** al ***file system*** per permettere agli utenti di accedere semplicemente a codice e dati

Esempi di SO

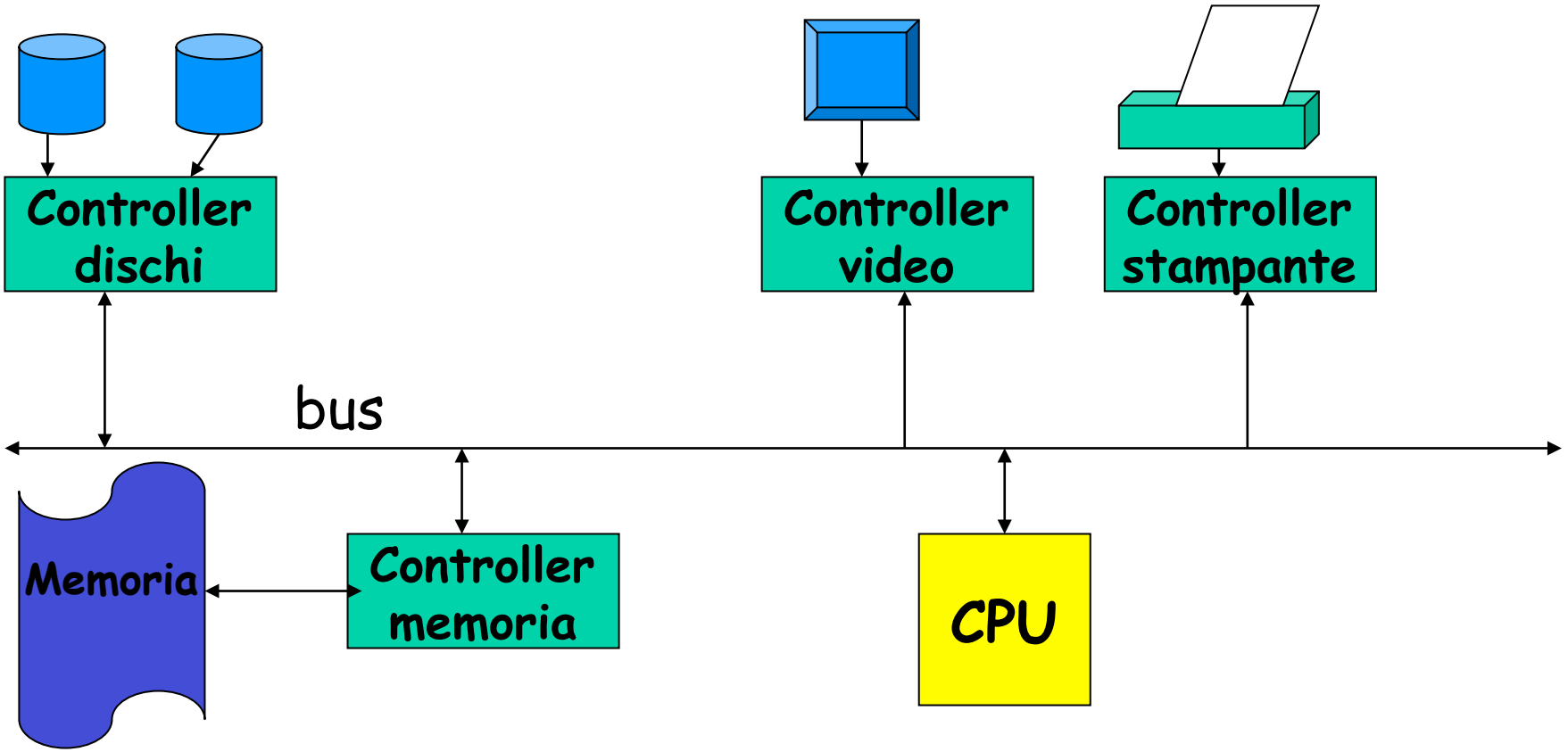
- **MSDOS**: monoprogrammato, monoutente
- **Windows 95/98, molti SO attuali per dispositivi portabili (Symbian, PalmOS, Android, etc.)**: multiprogrammato (time sharing), monoutente
- **Windows NT/2000/XP**: multiprogrammato, "multiutente"
- **UNIX/Linux**: multiprogrammato, multiutente
- **MacOSX**: multiprogrammato, multiutente

Evoluzione dei concetti nei SO



Alcuni richiami al funzionamento hardware di un sistema di elaborazione

Architettura di un sistema di elaborazione



Controller: interfaccia HW delle periferiche verso il bus di sistema

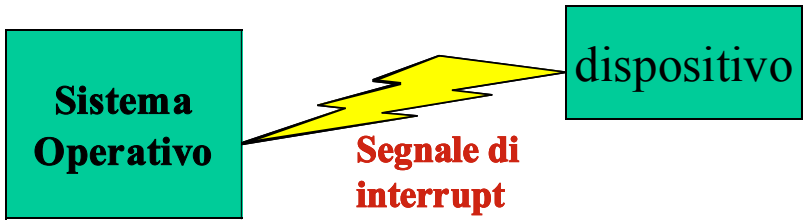
Hardware di un sistema di elaborazione

Funzionamento a interruzioni:

- le varie *componenti* (HW e SW) del sistema interagiscono con SO mediante ***interruzioni asincrone (interrupt)***
- ogni interruzione è causata da un ***evento***, ad es.:
  - ***richiesta di servizi al SO***
  - ***completamento di I/O***
  - ***accesso non consentito alla memoria***
- ad ogni interruzione è associata una ***routine di servizio (handler)*** per la ***gestione dell'evento***

Interruzioni hardware e software

- ***Interruzioni hardware***:
  dispositivi inviano segnali a CPU per notificare particolari eventi al SO (es. completamento di un'operazione)



- ***Interruzioni software***:
  programmi in esecuzione possono generare interruzioni SW
  - quando tentano l'esecuzione di ***operazioni non lecite*** (ad es. divisione per 0): ***trap***
  - quando richiedono l'esecuzione di servizi al SO - ***system call***

Gestione delle interruzioni

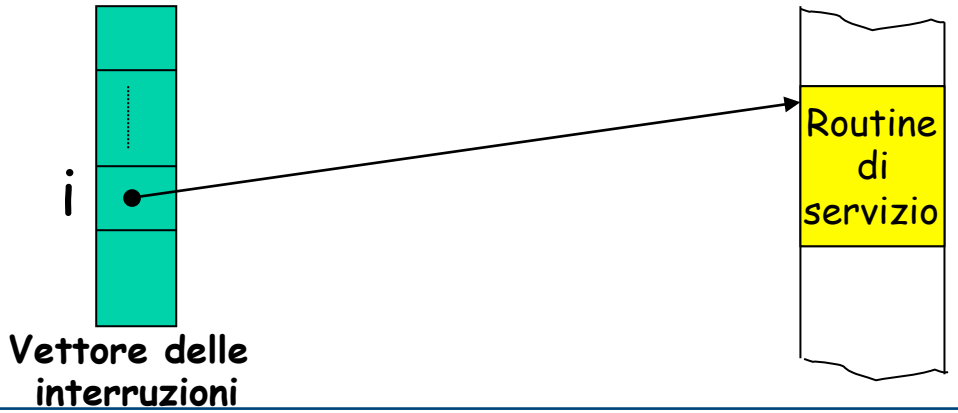
Alla ricezione di un'**interruzione**, il SO :

1] interrompe la sua esecuzione => ***salvataggio dello stato*** in memoria

2] attiva la ***routine di servizio all'interruzione*** (handler)

3] ***ripristina lo stato*** salvato

Per individuare la routine di servizio, si può utilizzare un **vettore delle interruzioni**



Input/Output

Come avviene l'I/O in un sistema di elaborazione?

Controller: interfaccia HW delle periferiche verso il bus di sistema;

ogni controller è dotato di:

- ***un registro dati*** (ove ***memorizzare temporaneamente*** le informazioni da ***leggere*** o ***scrivere***)
- alcuni ***registri speciali*** , ove ***memorizzare le specifiche delle operazioni*** di I/O da eseguire (reg. **controllo**) e l'esito delle operazioni eseguite (reg. **stato**).

Input/Output

Quando un job richiede un'operazione di I/O (ad esempio, ***lettura da un dispositivo***):

- CPU ***scrive nei registri speciali*** del dispositivo coinvolto le ***specifiche dell'operazione*** da eseguire
- controller esamina i registri e provvede a ***trasferire i dati richiesti dal dispositivo al registro dati***
- invio di ***interrupt alla CPU*** (completamento del trasferimento)
- CPU esegue l'operazione di I/O tramite la routine di servizio (***trasferimento dal registro dati del controller alla memoria centrale***)

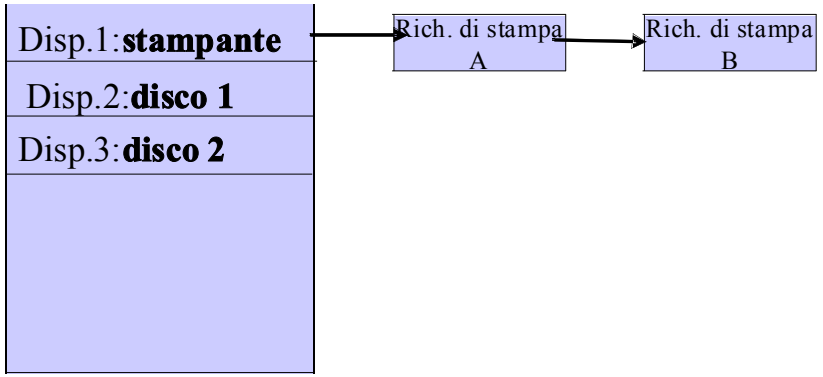
- **driver di dispositivo:** componente del SO che interagisce direttamente con il dispositivo:
  - copia nei registri del controller le informazioni relative all'operazione da effettuare
  - è l'unica componente del s.o. device-dependent (la sua struttura è strettamente dipendente dal particolare dispositivo controllato)

Input/Output

2 tipi di I/O

- **Sincrono**: il ***job viene sospeso*** in attesa del completamento dell'operazione di I/O
- **Asincrono**: il sistema restituisce ***immediatamente il controllo al job***
  - se necessario, funzionalità di blocco in attesa di completamento dell'I/O
  - possibilità di più I/O ***pendenti***
    -> tabella di stato dei dispositivi

I/O asincrono = maggiore efficienza



Direct Memory Access

Il trasferimento tra memoria e dispositivo può essere effettuato direttamente, ***senza intervento della CPU***

Introduzione di un dispositivo HW per controllare I/O: ***DMA controller***

- **driver di dispositivo (DMA)** componente del SO che
  - ***copia nei registri del DMA controller*** i dati relativi al trasferimento da effettuare
  - ***invia comando*** richiesto ***al DMA controller***
- **interrupt** alla CPU (inviato dal DMA controller) ***solo alla fine*** del trasferimento dispositivo -> memoria, usualmente di grandi quantità di dati

Protezione HW delle risorse

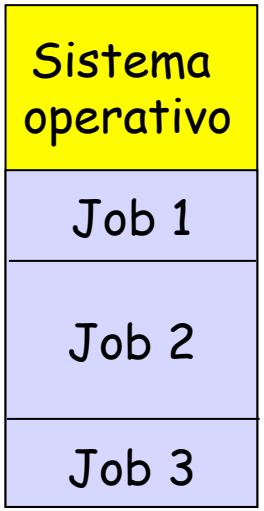
- Nei sistemi che prevedono multiprogrammazione e multiutenza sono necessari alcuni ***meccanismi HW*** (e non solo…) ***per esercitare protezione***
- Le risorse allocate a programmi/utenti devono essere protette nei confronti di ***accessi illeciti di altri programmi/utenti***:
  - dispositivi di I/O
  - memoria
  - CPU

Ad esempio: accesso a ***locazioni esterne allo spazio di indirizzamento del programma***

Protezione della memoria

In un sistema **multiprogrammato** o **time sharing**, ogni *job* ha un suo spazio di indirizzi:

- è necessario ***impedire al programma in esecuzione di accedere ad aree di memoria esterne al proprio spazio*** (ad esempio del SO oppure di altri *job*)



Se fosse consentito: un programma potrebbe modificare codice e dati di altri programmi o, ancor peggio, del SO

Protezione

Per garantire protezione, molte architetture di CPU prevedono un ***duplice modo di funzionamento (dual mode):***

- ***user*** mode
- ***kernel*** mode (***supervisor***, ***monitor*** mode)

Realizzazione: l'architettura hardware della CPU prevede almeno un ***bit di modo***

- **kernel**: 0
- **user**: 1

Dual mode

Istruzioni privilegiate: sono quelle più *pericolose* e possono essere eseguite soltanto se il sistema si trova in ***kernel mode:***

- accesso a dispositivi di I/O (dischi,schede di rete, ...)
- gestione della memoria (accesso a strutture dati di sistema per controllo e accesso alla memoria, …)
- disabilitazione interruzioni
- istruzione di ***shutdown*** (arresto del sistema)
- …

Dual Mode

- **SO esegue in modo kernel**

- **Ogni programma utente esegue in user mode:**

  - quando un ***programma utente tenta l'esecuzione di una istruzione privilegiata***, l'hardware lo impedisce (può essere generato un ***trap***)

  - se necessita di ***operazioni privilegiate***:

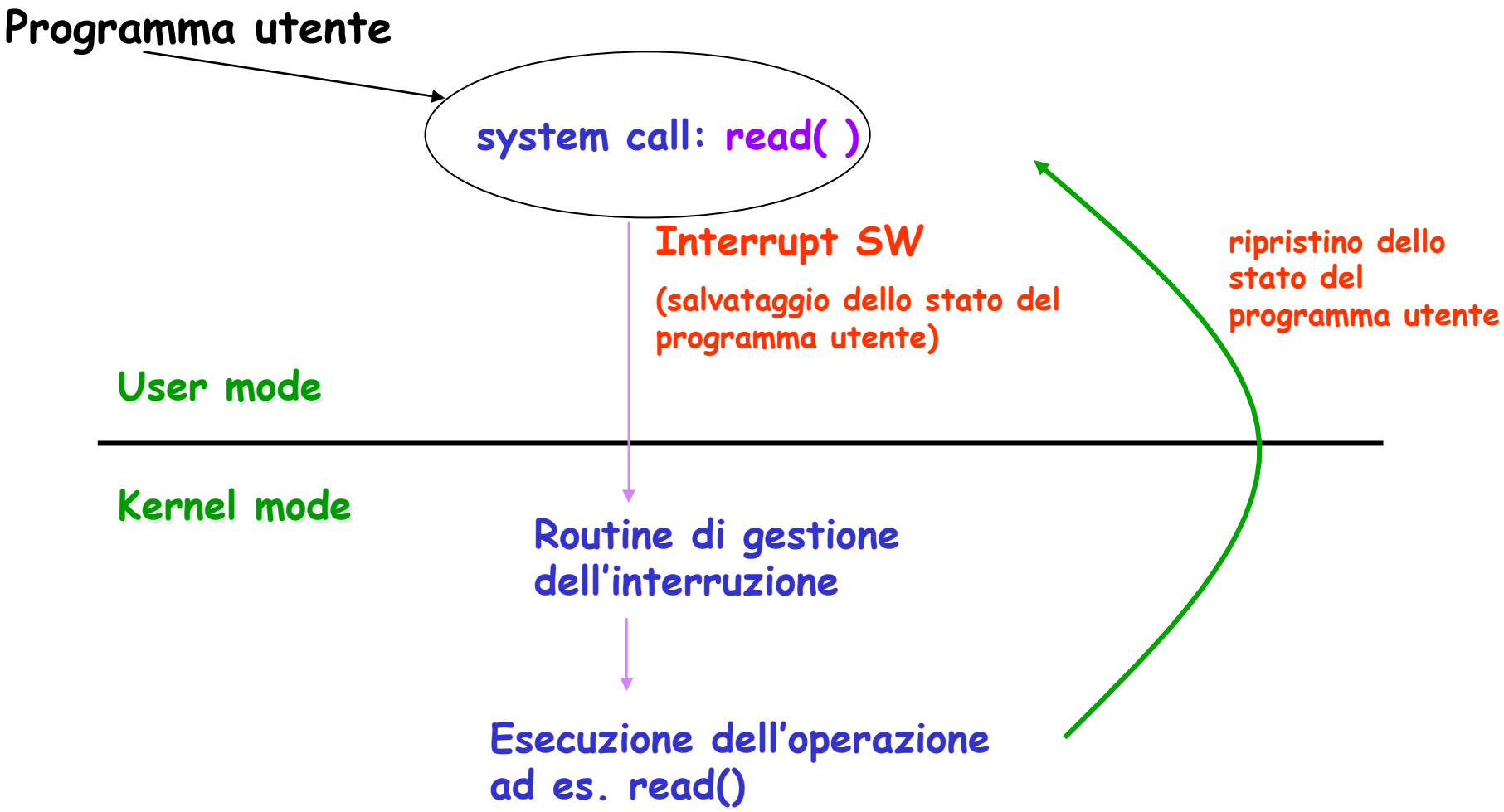
    chiamata a ***system call***

System call

Per ottenere l'esecuzione di ***istruzioni privilegiate, un programma di utente deve chiamare una system call:***

1. invio di ***un'interruzione software*** al SO
2. ***salvataggio dello stato*** (PC, registri, bit di modo, ...) del programma chiamante e trasferimento del controllo al SO
3. SO esegue in ***modo kernel*** l'operazione richiesta
4. al termine dell'operazione, il controllo ritorna al programma chiamante (***ritorno al modo user***)

System call


Programma utente
system call: read( )
Interrupt SW
(salvataggio dello stato del programma utente)
ripristino dello stato del programma utente
User mode
Kernel mode
Routine di gestione dell'interruzione
Esecuzione dell'operazione ad es. read()

Protezione

ES: Architettura Intel IA32.

- I 2 bit meno significativi del registro **CS** rappresentano il livello (ring) di privilegio corrente (Current Privilege Level):

  Ring 0 dotato dei maggiori privilegi e quindi destinato al kernel del sistema operativo -> modo kernel

  ...

  Ring 3, quello dotato dei minori privilegi e quindi destinato alle applicazioni utente -> modo user

  I sistemi operativi Windows e Linux usano solo il ring 0 (kernel mode) e il ring 3 (user mode).

Introduzione all'Organizzazione dei Sistemi Operativi

Struttura dei SO

Quali sono le ***componenti*** di un SO?

- gestione dei ***processi***
- gestione della ***memoria centrale***
- gestione di ***memoria secondaria*** e ***file system***
- gestione dell'***I/O***
- ***protezione e sicurezza***
- interfaccia utente/programmatore

Quali sono le ***relazioni mutue*** tra le componenti?

Processi

Processo = programma in esecuzione

- il ***programma*** è ***un'entità passiva*** (un insieme di byte contenente le istruzioni che dovranno essere eseguite)
- ***il processo è un'entità attiva:***
  - è ***l'unità di lavoro/esecuzione*** all'interno del sistema. ***Ogni attività all'interno del SO è rappresentata da un processo***
  - ***è l'istanza di un programma in esecuzione***

Processo = programma +
contesto di esecuzione (PC, registri, …)

Gestione dei processi

In un sistema multiprogrammato: più processi possono essere ***simultaneamente presenti*** nel sistema

Compito cruciale del SO

- ***creazione/terminazione*** dei processi
- ***sospensione/ripristino*** dei processi
- ***sincronizzazione/comunicazione*** dei processi
- ***gestione del blocco critico (deadlock)*** di processi

Gestione della memoria centrale

HW di sistema di elaborazione è equipaggiato con ***un unico spazio di memoria*** accessibile direttamente da CPU e dispositivi

Compito cruciale di SO

- ***separare gli spazi di indirizzi*** associati ai processi
- ***allocare/deallocare memoria*** ai processi
- ***memoria virtuale*** – gestire ***spazi logici di indirizzi*** di dimensioni complessivamente ***superiori allo spazio fisico***
- realizzare i collegamenti (***binding***) tra ***memoria logica e memoria fisica***

Gestione dei dispositivi di I/O

Gestione dell'I/O rappresenta una parte importante di SO:

- ***interfaccia*** tra programmi e dispositivi
- per ogni dispositivo: ***device driver***
  - ***routine per l'interazione con un particolare dispositivo***
  - contiene ***conoscenza specifica*** sul dispositivo (ad es., routine di gestione delle interruzioni)

Gestione della memoria secondaria

Tra tutti i dispositivi, la ***memoria secondaria*** riveste un ruolo particolarmente importante:

- ***allocazione/deallocazione*** di spazio
- gestione dello ***spazio libero***
- ***scheduling*** delle operazioni sul disco

Di solito:

- la **gestione dei file** usa i meccanismi di gestione della memoria secondaria
- la **gestione della memoria secondaria** è indipendente dalla gestione dei file

Gestione del file system

Ogni sistema di elaborazione dispone di uno o più dispositivi per la memorizzazione persistente delle informazioni (***memoria secondaria***)

Compito di SO

fornire una ***visione logica uniforme della memoria secondaria*** (indipendente dal tipo e dal numero dei dispositivi):

- realizzare il ***concetto astratto di file***, come unità di memorizzazione logica
- fornire una struttura astratta per ***l'organizzazione*** dei file (***direttorio***)

Gestione del file system

Inoltre, SO si deve occupare di:

- creazione/cancellazione di file e direttori
- manipolazione di file/direttori
- associazione tra file e dispositivi di memorizzazione secondaria

Spesso file, direttori e dispositivi di I/O vengono ***presentati*** a utenti/programmi ***in modo uniforme***

Protezione e sicurezza

In un sistema multiprogrammato, più entità (processi o utenti) possono utilizzare le risorse del sistema contemporaneamente:
necessità di protezione

Protezione: controllo dell'accesso alle risorse del sistema da parte di processi (e utenti) mediante

- ***autorizzazioni***
- ***modalità di accesso***

Risorse da proteggere:

- ❑ memoria
- ❑ processi
- ❑ file
- ❑ dispositivi

Protezione e sicurezza

Sicurezza:

se il sistema appartiene a una rete, la ***sicurezza misura l'affidabilità del sistema nei confronti di accessi (attacchi) dal mondo esterno***

Non ce ne occuperemo all'interno di questo corso…

Interfaccia utente

SO presenta un'interfaccia che consente l'interazione con l'utente

- **interprete comandi** (***shell***): l'interazione avviene mediante una linea di comando
- **interfaccia grafica** (graphical user interface, ***GUI***): l'interazione avviene mediante ***interazione*** mouse-elementi grafici su desktop; di solito è organizzata a finestre

Interfaccia programmatore

L'interfaccia del SO verso i processi è rappresentato dalle ***system call***:

- mediante la system call il ***processo richiede a SO*** l'esecuzione di un servizio
- la system call esegue ***istruzioni privilegiate***: passaggio da modo ***user*** a modo ***kernel***

Classi di system call:

- ❑ gestione dei processi
- ❑ gestione di file e di dispositivi (spesso trattati in modo omogeneo)
- ❑ gestione informazioni di sistema
- ❑ comunicazione/sincronizzazione tra processi

Programma di sistema **= programma che chiama system call**

Struttura e organizzazione di SO

Sistema operativo = insieme di componenti

- gestione dei processi
- gestione della memoria centrale
- gestione dei file
- gestione dell'I/O
- gestione della memoria secondaria
- protezione e sicurezza
- interfaccia utente/programmatore

Le componenti non sono indipendenti tra loro, ma interagiscono

Struttura del Sistema Operativo

Come sono organizzate le varie componenti all'interno del sistema operativo?

Varie soluzioni:

- **struttura *monolitica***
- **struttura *modulare***
- ***Microkernel***
- ***Macchine virtuali***

Struttura Monolitica

Il sistema operativo è costituito da un unico modulo contenente un **insieme di procedure**, che realizzano le varie componenti:

- l'interazione tra le diverse componenti avviene mediante il meccanismo di **chiamata a procedura**.

Esempi: *MS-DOS, UNIX, GNU/Linux*

Sistemi Operativi Monolitici

Principale Vantaggio: basso costo di interazione tra le componenti.

Svantaggio: Il SO è un sistema complesso e presenta gli stessi requisiti delle applicazioni ***in-the-large:***

- **estendibilità**
- **manutenibilità**
- **riutilizzo**
- **portabilità**
- **affidabilità**
- ...

Soluzione: organizzazione ***modulare***

Struttura modulare

Le varie componenti del SO vengono organizzate in moduli caratterizzati da interfacce ben definite.

Sistemi Stratificati (a *livelli*)

(THE, Dijkstra1968)

il sistema operativo è costituito da livelli sovrapposti, ognuno dei quali realizza un insieme di funzionalità:

- **ogni livello realizza un'insieme di funzionalità che vengono offerte al livello superiore mediante un'interfaccia**
- **ogni livello utilizza le funzionalità offerte dal livello sottostante, per realizzare altre funzionalità**

Struttura a livelli

Ad esempio: **THE (5 livelli)**

Struttura Stratificata

Vantaggi:

- **Astrazione:** ogni livello è un **oggetto astratto**, che fornisce ai livelli superiori una visione astratta del sistema (***Macchina Virtuale***), limitata alle astrazioni presentate nell'interfaccia.

- **Modularità:** le relazioni tra i livelli sono chiaramente esplicitate dalle interfacce -> possibilità di sviluppo, verifica, modifica in modo indipendente dagli altri livelli.

Struttura Stratificata

Svantaggi:

- **Organizzazione gerarchica** tra le componenti: non sempre è possibile -> difficoltà di realizzazione.
- **Scarsa efficienza:** costo di attraversamento dei livelli

Soluzione: limitare il numero dei livelli.

Nucleo del Sistema Operativo (kernel)

"E` la parte del sistema operativo che esegue in *modo kernel*"

- È la parte più *interna* del sistema operativo, che si interfaccia direttamente con l'hardware della macchina.
- Le funzioni realizzate all'interno del nucleo variano a seconda del Sistema Operativo.

Nucleo del Sistema Operativo (*kernel*)

- Tipicamente, tra le funzioni del nucleo ci sono:
  - Creazione/terminazione dei processi
  - **scheduling** della Cpu
  - gestire il **cambio** di contesti
  - Sincronizzazione/comunicazione tra processi
  - Gestione della memoria
  - Gestione dell' I/O
  - Gestione delle **interruzioni**
  - implementazione **system call**.

Sistemi Operativi a Microkernel

- La struttura del nucleo è ridotta a poche funzionalità di base.
- il resto del SO è rappresentato da processi di utente

Caratteristiche:

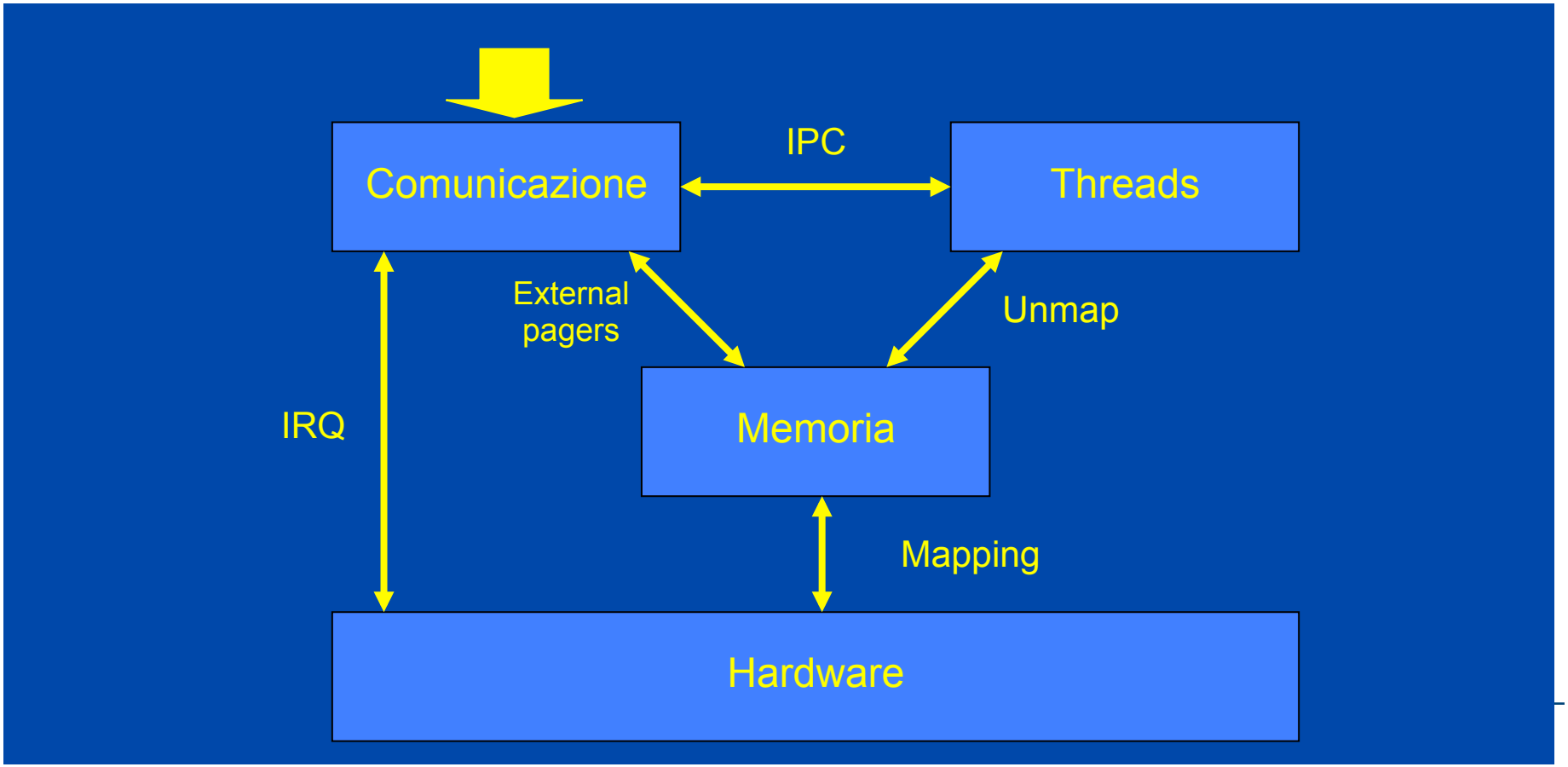
- affidabilità (separazione tra componenti)
- possibilità di estensioni e personalizzazioni
- scarsa efficienza (molte chiamate a system call)

ESEMPI: Minix, L4, Mach, Hurd, BeOS/Haiku

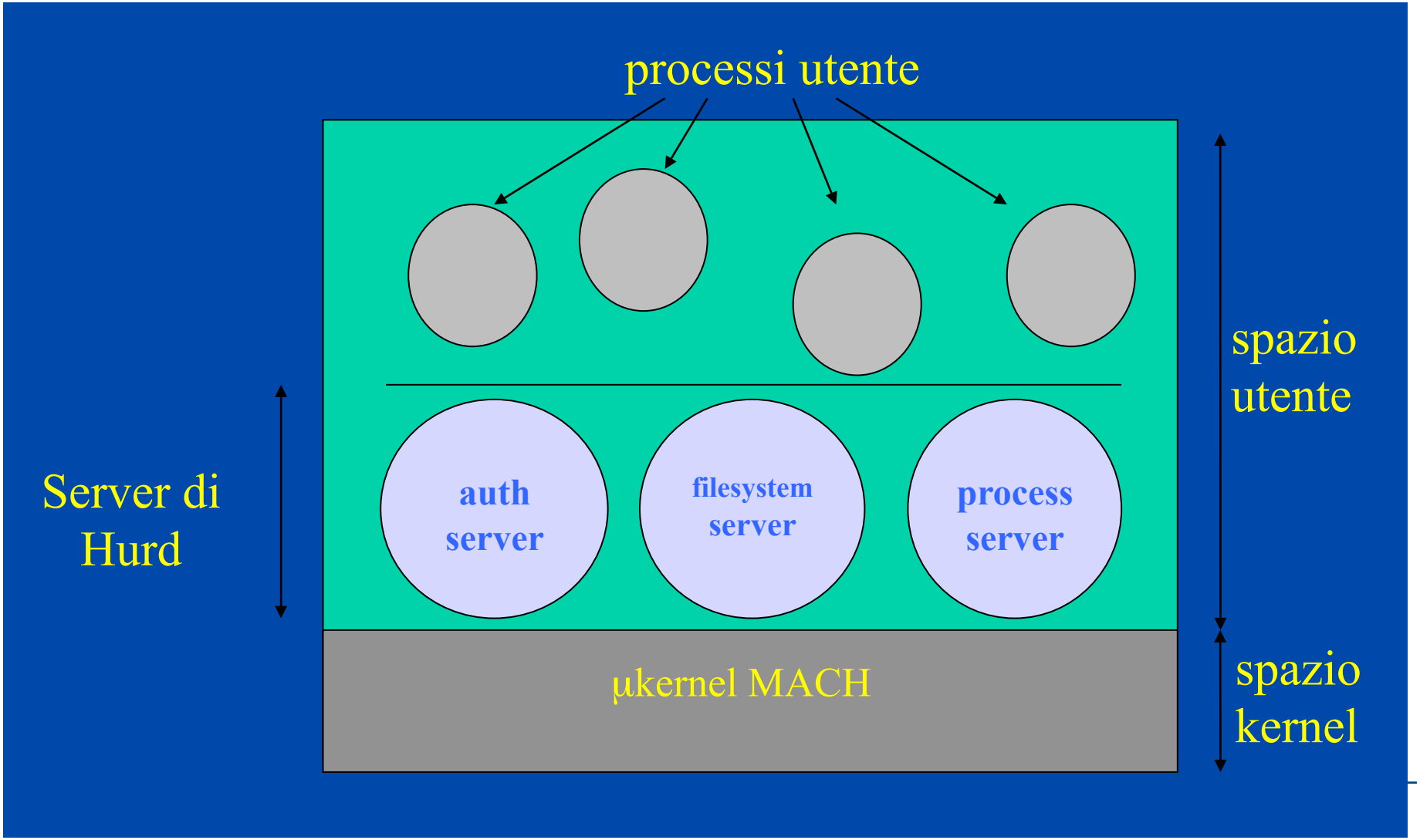
L4 μkernel

http://os.inf.tu-dresden.de

- gestione dei thread
- allocazione della memoria (pager esterni)
- Inter Process Communication



GNU/Hurd


processi utente
Server di Hurd
auth server
filesystem server
process server
μkernel MACH
spazio utente
spazio kernel

Kernel Ibridi

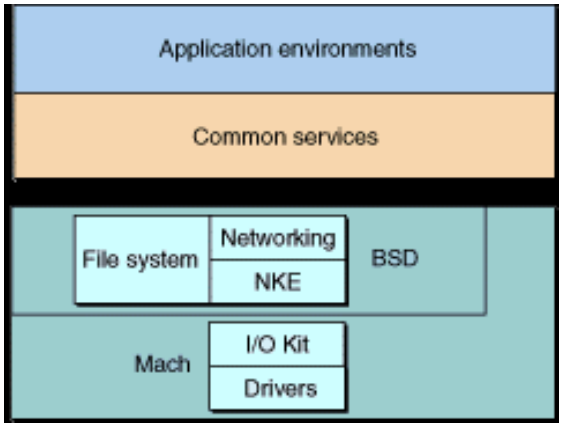
- microkernel che integrano a livello kernel funzionalità non essenziali.

Esempi:

- Microsoft Windows.
- XNU, il kernel di Mac OS X (inclusione di codice BSD in un kernel basato su Mach).

XNU (Darwin) - "X is Not Unix"

Mach:
kernel threads
Processes
pre-emptive multitasking
message-passing
Memory management



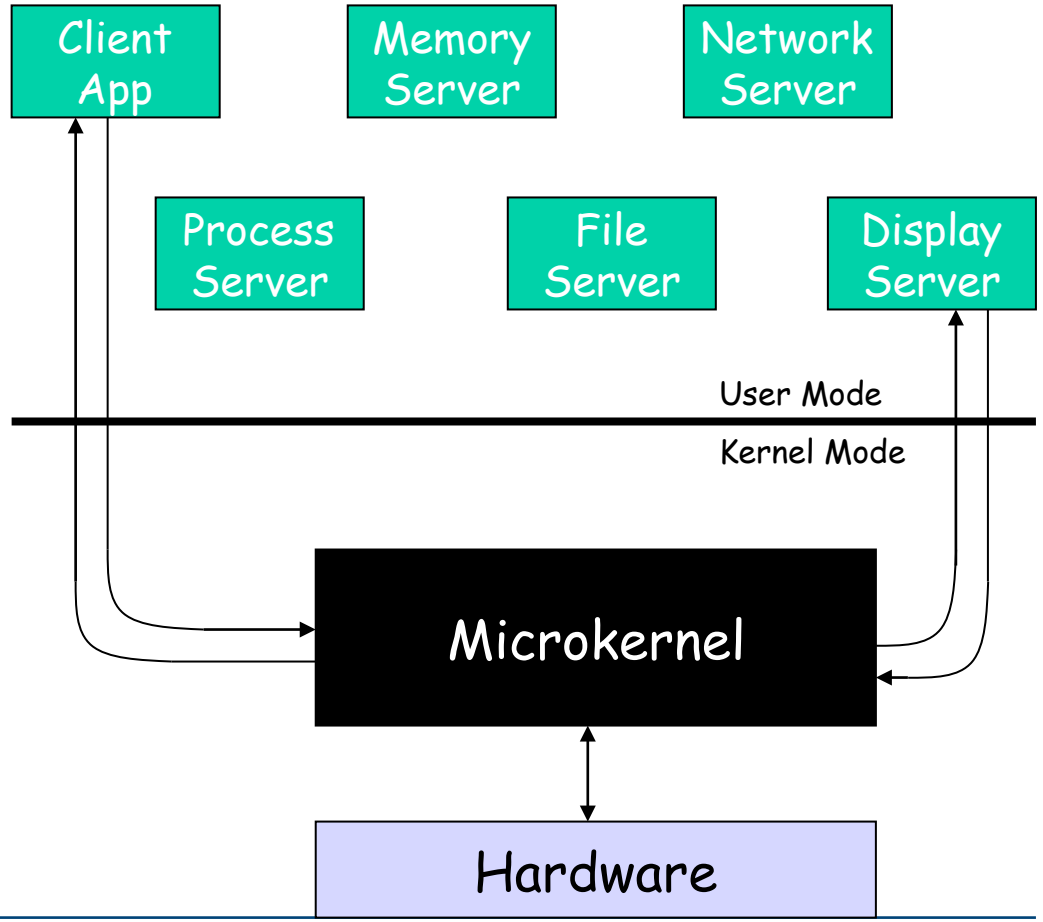
BSD: basic security policies, protezione, network stack, file system, Posix API

MSWinXP

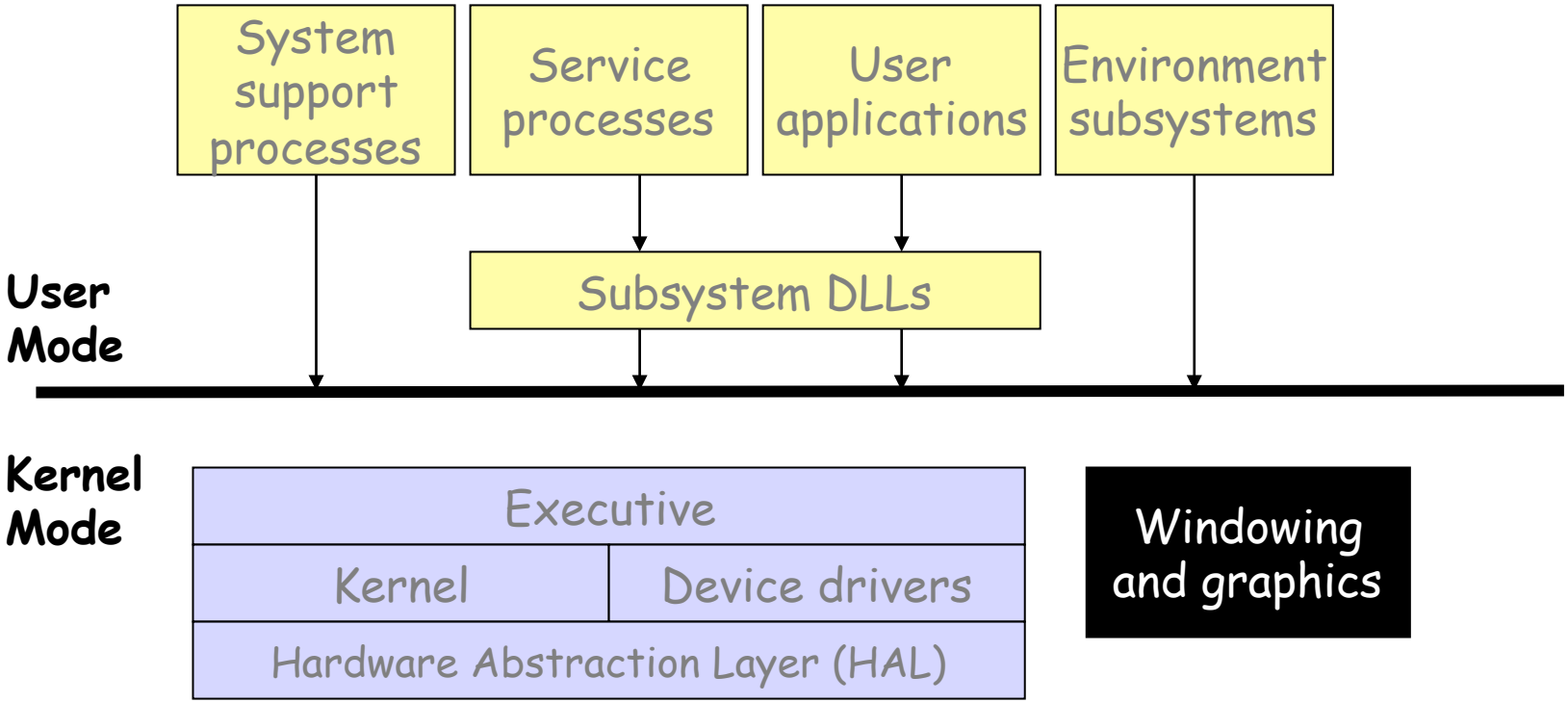
Kernel implementa:

- Scheduler
- Gestore della memoria
- Interprocess communication (IPC)

Server in user-mode



Architettura di WinXP: vista semplificata

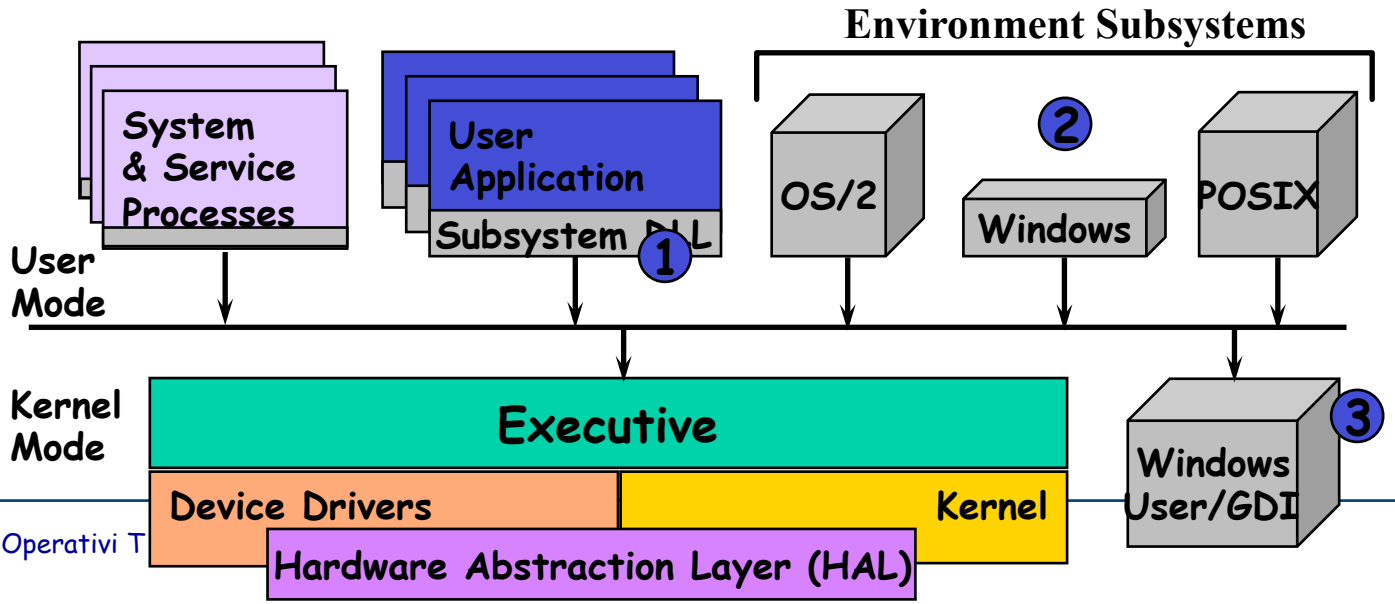


Cenni di architettura WinXP

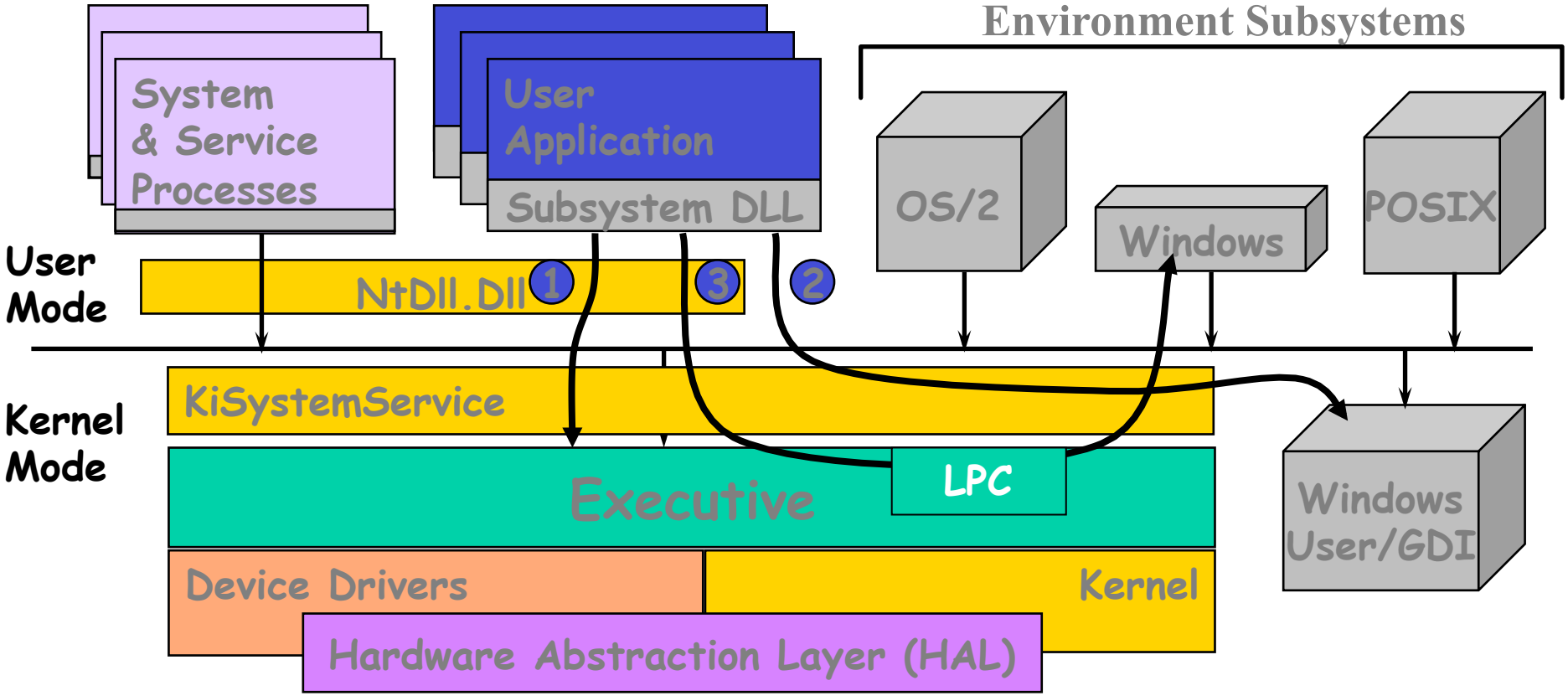
- Progettato per avere **più "personalità"**
  - Applicazioni utente non chiamano servizi di sistema direttam.
- **DLL di sottosistema** per tradurre una funzione nella corrispondente chiamata di sistema interna
- Processi di **Sottosistema (Environment Subsystem)**
  - Espongono una serie di funzionalità sottostanti alle applic.
  - Possono fare cose diverse nei diversi sottosistemi (e.g., POSIX fork)
- Originariamente tre sottosistemi: Windows, POSIX e OS/2
  - Windows 2000 include solo sottosistemi Windows e POSIX
  - Windows XP/Vista include solo il sottosistema Windows
    - "Services for Unix" offrono un sottosistema POSIX
    - Inclusi in Windows Server 2003 R2

Componenti di sottosistema

1. DLL per le API
   - per Windows: Kernel32.DLL, Gdi32.DLL, User32.DLL, etc.
2. Processi di sottosistema
   - per Windows: CSRSS.EXE (Client Server Runtime SubSystem)
3. Solo per Windows: kernel-mode GDI code
   - Win32K.SYS - (il codice era originariamente parte di CSRSS)



Comunicazione applicazioni con SO



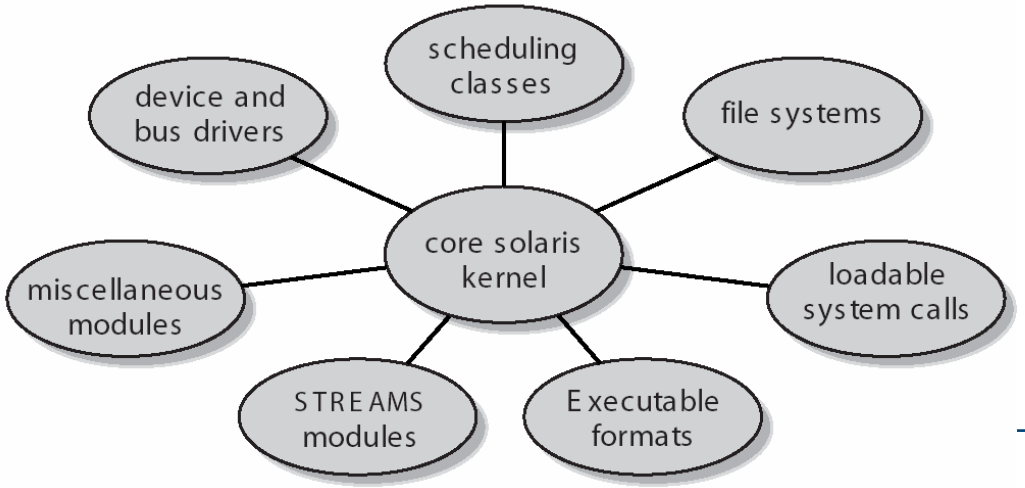
1. La maggior parte delle Windows Kernel API
2. La maggior parte delle Windows User e GDI API
3. Alcune Windows API

Modularità

Molti moderni SO implementano il ***kernel in maniera modulare***

- ogni modulo core è ***separato***
- ogni modulo interagisce con gli altri tramite ***interfacce note***
- ogni modulo può essere ***caricato nel kernel quando*** e ove ***necessario***
- possono usare tecniche object-oriented

Strutturazione simile ai livelli, ma con ***maggiore flessibilità***

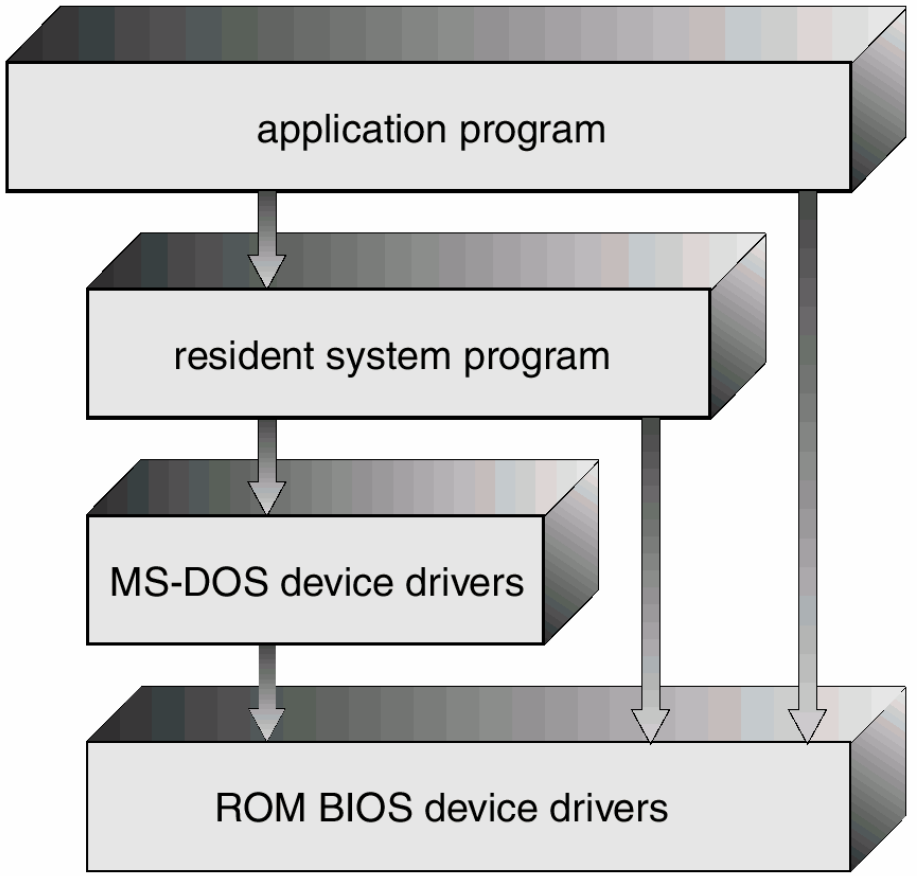


Esempio di SO Solaris di SUN

Una piccola panoramica: organizzazione di MS-DOS

MS-DOS – progettato per avere ***minimo footprint***

- ***non diviso in moduli***
- sebbene abbia una qualche struttura, ***interfacce e livelli di funzionalità non sono ben separati***

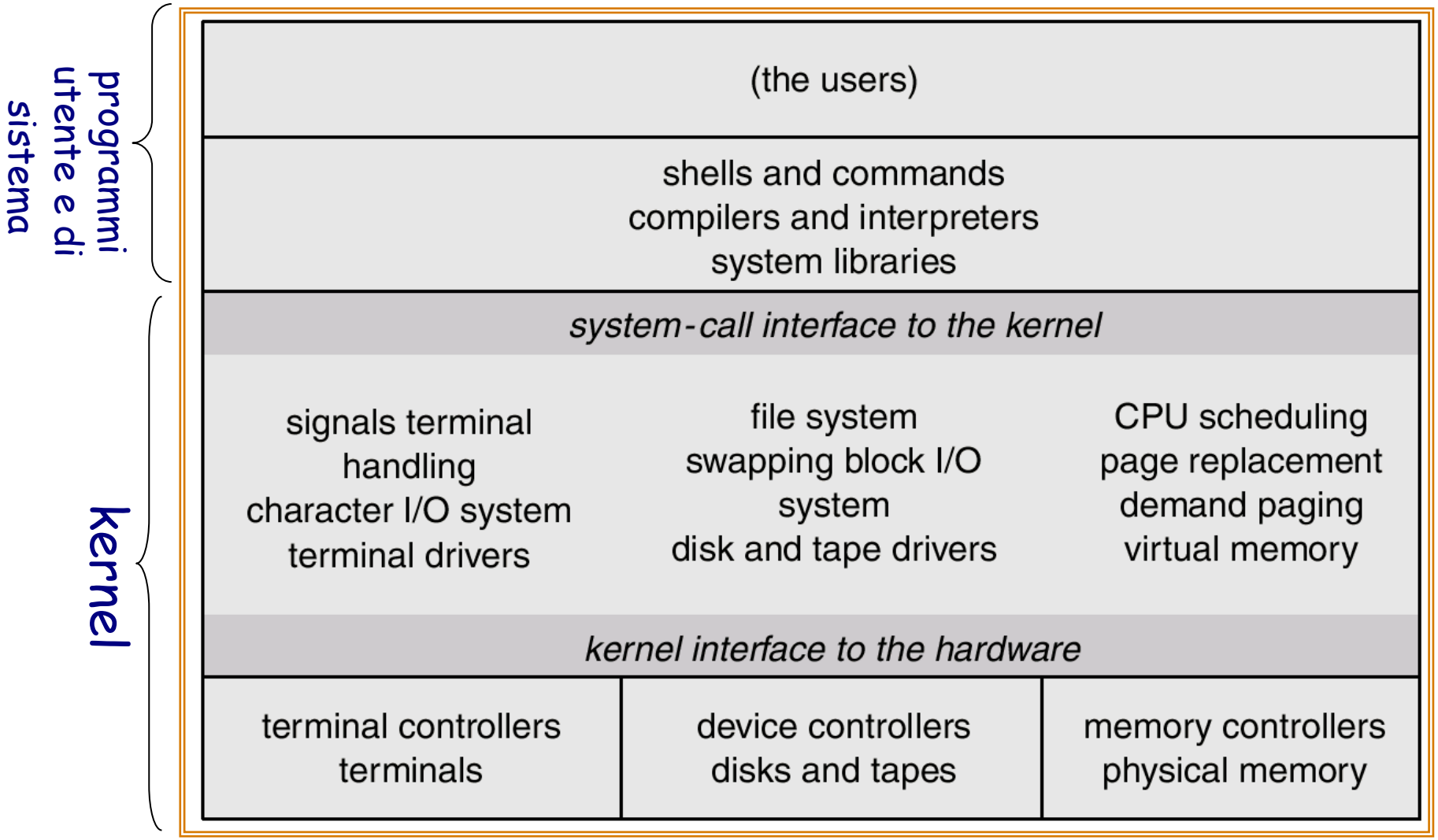


Una piccola panoramica: organizzazione di UNIX

UNIX – dati i limiti delle risorse hw dell'epoca (anni '70), originariamente UNIX sceglie di avere una ***strutturazione limitata***. Consiste di due parti separabili:

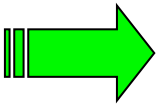
- ***programmi di sistema***
- ***kernel***
  - costituito da tutto ciò che è sotto l' interfaccia delle system-call interface e sopra hw fisico
  - fornisce funzionalità di file system, CPU scheduling, gestione memoria, …; ***molte funzionalità tutte allo stesso livello***

Organizzazione di UNIX

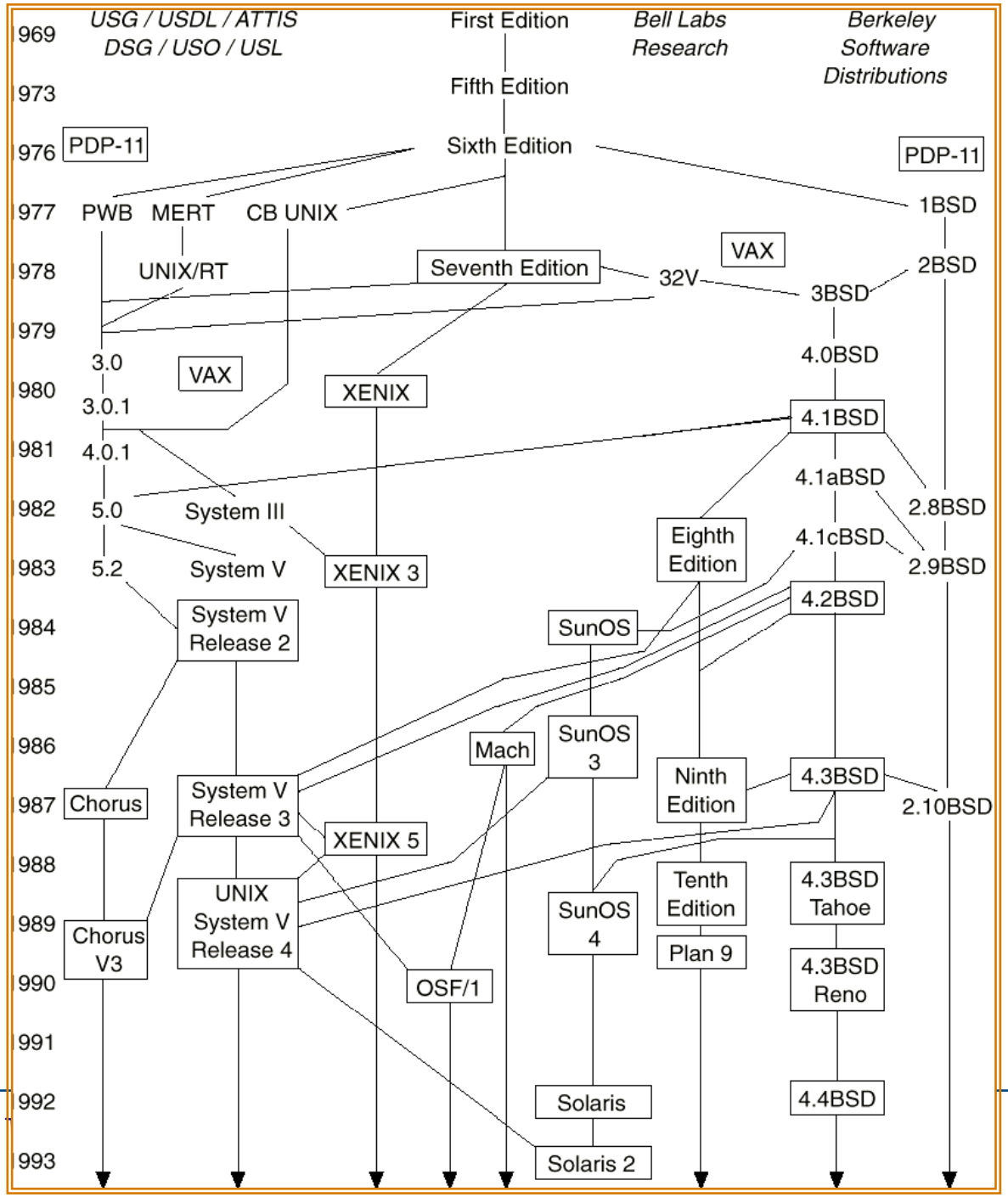


UNIX: qualche cenno storico

- Thompson e Ritchie, Bell Laboratories (1969). Raccolti diversi spunti dalle caratteristiche di altri SO contemporanei, specie ***MULTICS***
- Terza versione del sistema ***scritta in C***, ***specificamente sviluppato*** ai Bell Labs per supportare e implementare UNIX
- Gruppo di sviluppo UNIX più influente (escludendo Bell Labs e AT&T) - University of California at Berkeley (***Berkeley Software Distributions***):
  - ***4.0 BSD UNIX*** fu il risultato di finanziamento DARPA per lo sviluppo di una ***versione standard*** di UNIX
  - ***4.3 BSD UNIX***, sviluppato per VAX, influenzò molti dei SO successivi
- Numerosi progetti di ***standardizzazione*** per giungere a interfaccia di programmazione uniforme



POSIX

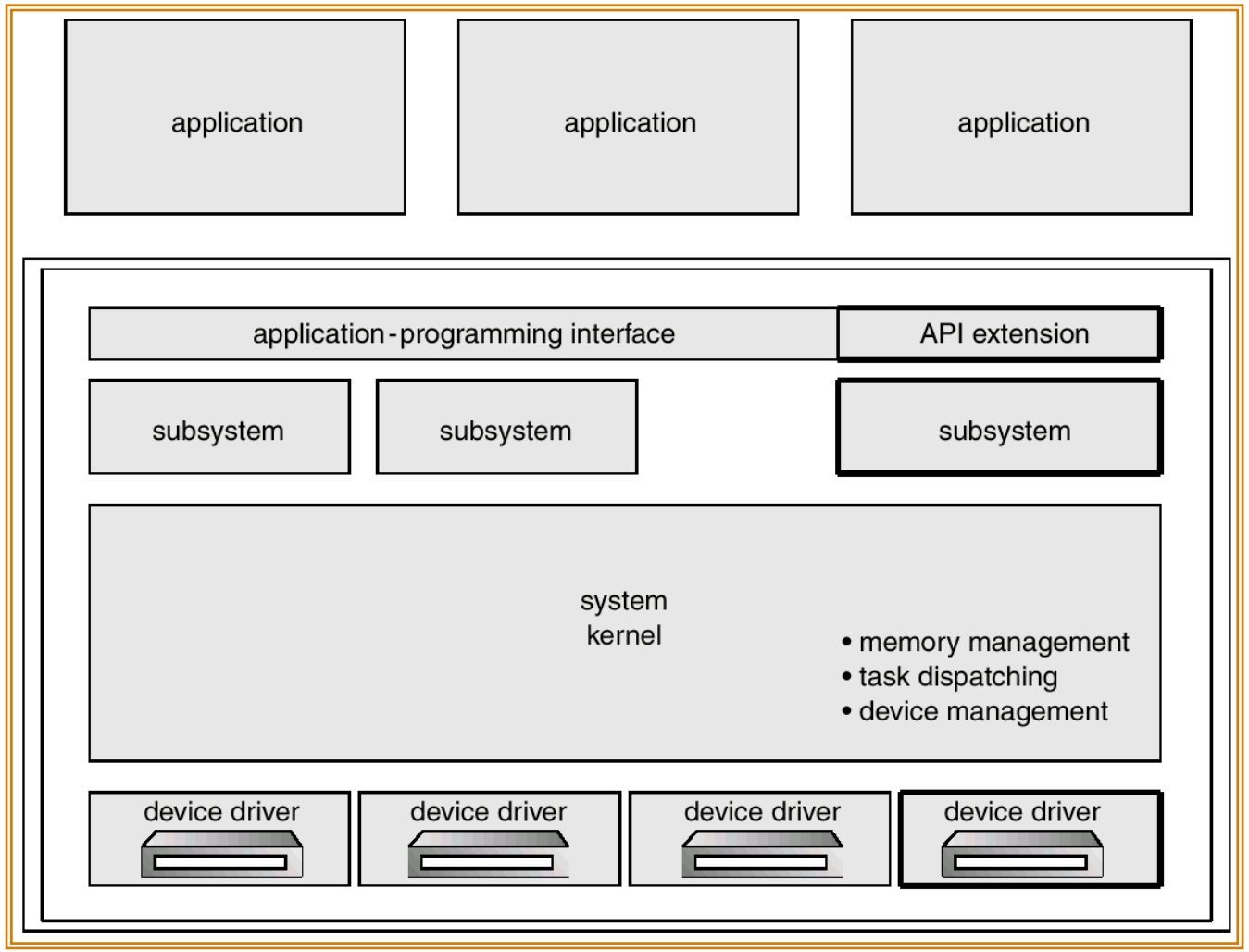

969
973
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
USG / USDL / ATTIS
DSG / USO / USL
First Edition
Bell Labs
Research
Berkeley
Software
Distributions
Fifth Edition
Sixth Edition
PDP-11
PDP-11
PWB
MERT
CB UNIX
1BSD
UNIX/RT
Seventh Edition
32V
VAX
2BSD
3BSD
3.0
VAX
4.0BSD
XENIX
3.0.1
4.1BSD
4.0.1
4.1aBSD
2.8BSD
5.0
System III
Eighth
Edition
4.1cBSD
2.9BSD
5.2
System V
XENIX 3
4.2BSD
System V
Release 2
SunOS
Mach
SunOS
3
Chorus
System V
Release 3
Ninth
Edition
4.3BSD
2.10BSD
XENIX 5
UNIX
System V
Release 4
Tenth
Edition
4.3BSD
Tahoe
Chorus
V3
SunOS
4
Plan 9
4.3BSD
Reno
OSF/1
Solaris
4.4BSD
Solaris 2

UNIX: principi di progettazione e vantaggi

- Progetto ***snello, pulito e modulare***
- Scritto in ***linguaggio di alto livello*** (linguaggio C)
- Disponibilità codice sorgente
- ***Potenti primitive di SO*** su una piattaforma a ***basso prezzo***

- Progettato per essere ***time-sharing***
- ***User interface semplice (shell),*** anche sostituibile
- File system con ***direttori organizzati ad albero***
- ***Concetto unificante di file***, come ***sequenza non strutturata*** di byte
- Supporto semplice a ***processi multipli e concorrenza***
- Supporto ampio allo ***sviluppo di programmi*** applicativi e/o di ***sistema***

Una piccola panoramica: organizzazione di OS/2



Buona strutturazione *a livelli e modulare*

Macchine virtuali

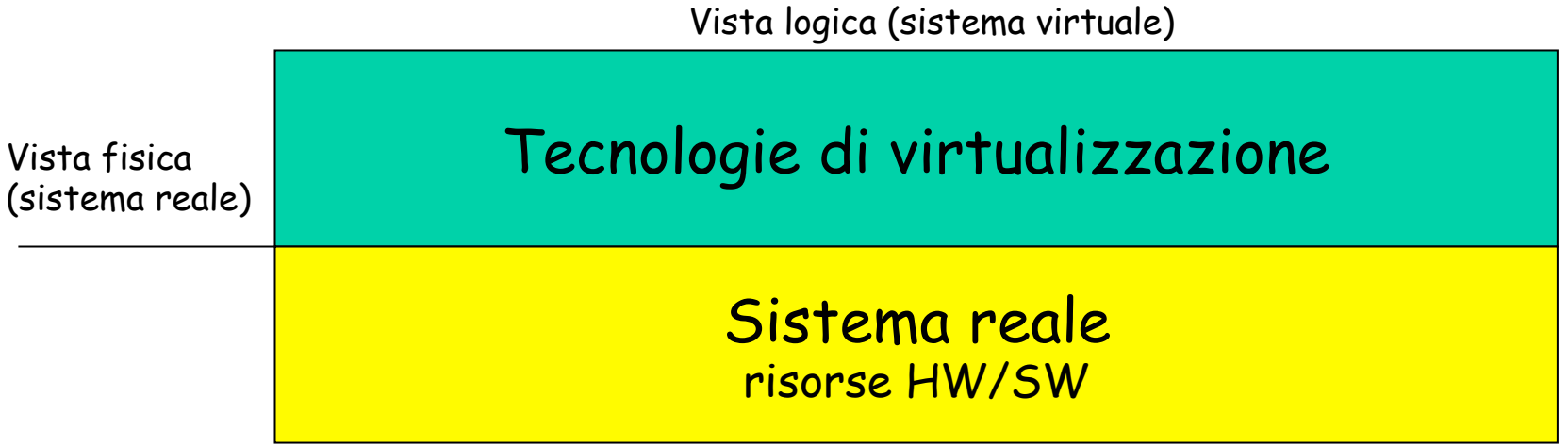
Macchine virtuali *(VMWare, VirtualBox, xen, Java, .NET)* sono la logica evoluzione dell' approccio a livelli. Virtualizzano ***sia hardware che kernel del SO***

- Creano l' illusione di ***processi multipli***, ciascuno in esecuzione sul suo ***processore privato*** e con la propria ***memoria virtuale privata***, messa a disposizione ***dal proprio kernel SO, che può essere diverso per processi diversi***

Virtualizzazione

Dato un sistema caratterizzato da un insieme di risorse (hardware e software), **virtualizzare il sistema** significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.

Ciò si ottiene introducendo **un livello di indirezione** tra la vista ***logica*** e quella ***fisica*** delle risorse.



Obiettivo: disaccoppiare il comportamento delle risorse hardware e software di un sistema di elaborazione, così come viste dall'utente, dalla loro realizzazione fisica.

Esempi di virtualizzazione

Astrazione: in generale un oggetto astratto (risorsa virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica):

- esibendo le proprietà significative per l'utilizzatore
- nascondendo i dettagli realizzativi non necessari.

Es: tipi di dato vs. rappresentazione binaria nella cella di memoria

Il **disaccoppiamento** è realizzato dalle operazioni (interfaccia) con le quali è possibile utilizzare l'oggetto.

Linguaggi di Programmazione. La capacità di portare lo stesso programma (scritto in un linguaggio di alto livello) su architetture diverse è possibile grazie alla definizione di una macchina virtuale in grado di interpretare ed eseguire ogni istruzione del linguaggio, indipendentemente dall'architettura del sistema (S.O. e HW):

- Interpreti (esempio Java Virtual Machine)
- Compilatori

Virtualizzazione a livello di processo. I sistemi multiprogrammati permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una macchina virtuale (CPU, memoria, dispositivi) dedicata. La virtualizzazione è realizzata dal **kernel** del sistema operativo.

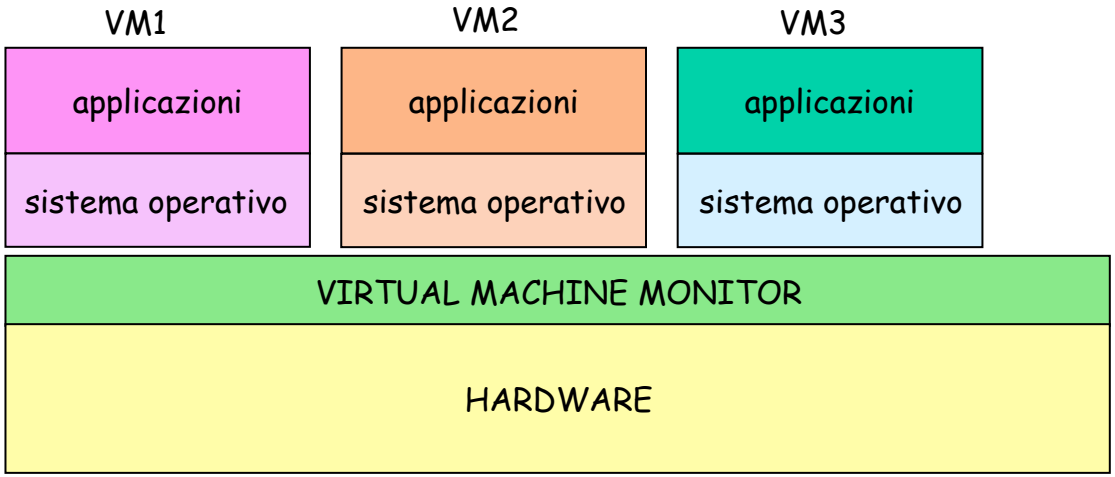
Sistemi Operativi per la Virtualizzazione

- La macchina fisica viene trasformata in n interfacce (**macchine virtuali**), ognuna delle quali e` una replica della macchina fisica:
  - dotata di tutte le istruzioni del processore (sia privilegiate che non privilegiate)
  - dotata delle risorse del sistema (memoria, dispositivi di I/O).

➔ Su ogni macchina virtuale è possibile installare ed eseguire un sistema operativo (eventualmente diverso da macchina a macchina): ***Virtual Machine Monitor***

Virtualizzazione di Sistema. Una singola piattaforma hardware viene condivisa da più sistemi operativi, ognuno dei quali è installato su una diversa macchina virtuale.

Il disaccoppiamento è realizzato da un componente chiamato Virtual Machine Monitor (VMM, o hypervisor) il cui compito è consentire la condivisione da parte di più macchine virtuali di una singola piattaforma hardware. Ogni macchina virtuale è costituita oltre che dall'applicazione che in essa viene eseguita, anche dal sistema operativo utilizzato.



Il VMM è il mediatore unico nelle interazioni tra le macchine virtuali e l'hardware sottostante, che garantisce:

- **isolamento** tra le VM
- **stabilità** del sistema

VMM di sistema vs. VMM ospitati

VMM di Sistema.

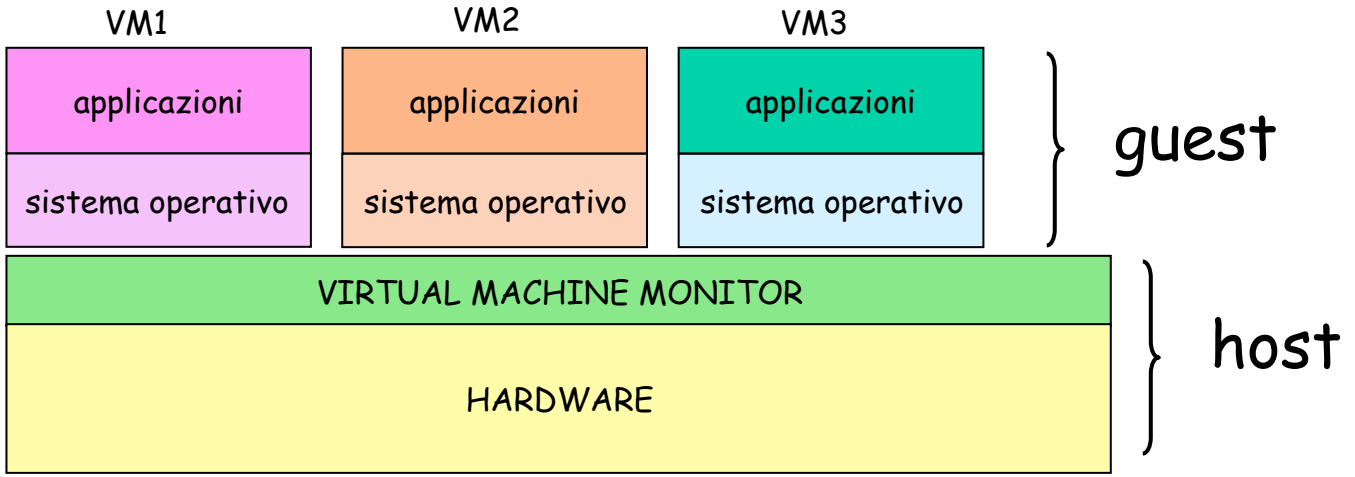
le funzionalità di virtualizzazione vengono integrate in un sistema operativo leggero, costituendo un unico sistema **posto direttamente sopra l'hardware dell'elaboratore**.

➡ E' necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche.

Esempi di VMM di sistema: Vmware ESX, xen

Host: piattaforma di base sulla quale si realizzano macchine virtuali. Comprende la macchina fisica, l'eventuale sistema operativo ed il VMM.

Guest: la macchina virtuale. Comprende applicazioni e sistema operativo



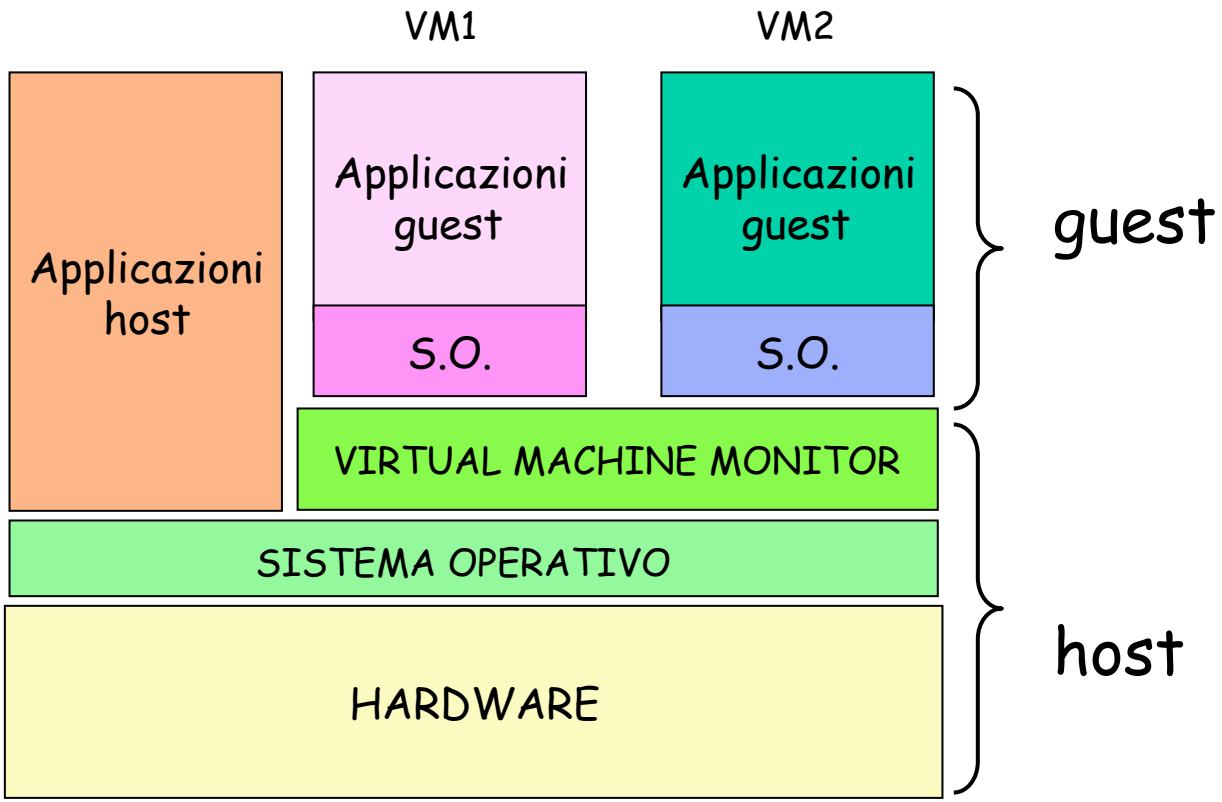
VMM di Sistema

VMM ospitato

il VMM viene installato come **un'applicazione** sopra un sistema operativo esistente, che opera nello **spazio utente** e accede l'hardware tramite le **system call** del S.O. su cui viene installato.

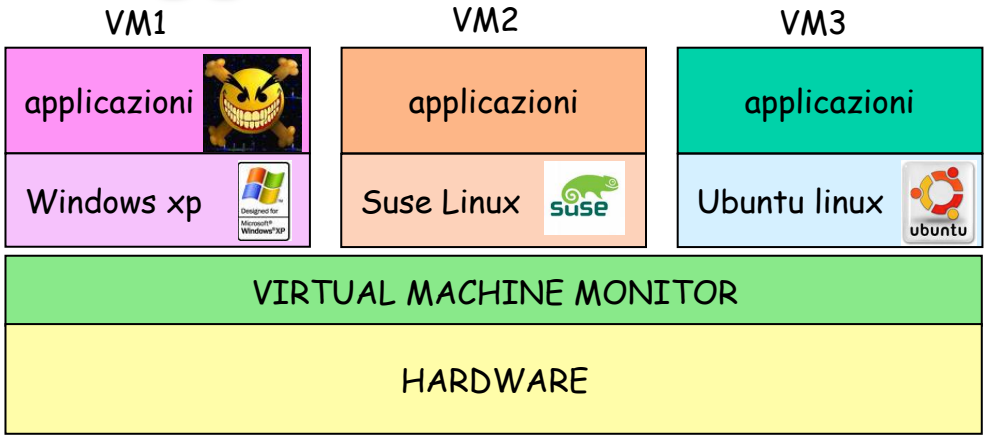
- Più semplice l'installazione (come un'applicazione).
- Può fare riferimento al S.O. sottostante per la gestione delle periferiche e può utilizzare altri servizi del S.O.(es. scheduling, gestione delle risorse.).
- Peggiore la performance.

Prodotti: User Mode Linux, VMware Server/Player, Microsoft Virtual Server , Parallels, VirtualBox.



VMM ospitato

Vantaggi della virtualizzazione



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

Vantaggi della virtualizzazione

- **Consolidamento HW:** possibilita` di concentrare piu` macchine (ad es. server) su un'unica architettura HW per un utilizzo efficiente dell'hardware (es. *server farm):*
  - *Abbattimento costi hw*
  - *Abbattimento costi amministrazione*

- **Gestione facilitata delle macchine:** e` possibile effettuare in modo semplice:
  - la creazione di macchine virtuali (virtual appliances)
  - l'amministrazione di macchine virtuali (reboot, ricompilazione kernel, etc.)
  - migrazione *a caldo* di macchine virtuali tra macchine fisiche:
    - possibilita` di manutenzione hw senza interrompere i servizi forniti dalle macchine virtuali
    - disaster recovery
    - workload balancing: alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte in modo "autonomico" a situazioni di sbilanciamento

Vantaggi della virtualizzazione

- **In ambito didattico:** invece di assegnare ad ogni studente un account su una macchina fisica, si assegna una macchina virtuale.

DEIS Virtual Lab. La Facoltà di Ingegneria (DEIS) ha realizzato un laboratorio di macchine virtuali che offre ad ogni studente una macchina virtuale personale da amministrare autonomamente:

- possibilita` di esercitarsi senza limitazioni nelle tecniche di amministrazione e configurazione del sistema;
- possibilita` di installazione e testing di nuovi sistemi operativi, anche prototipali, senza il rischio di compromettere la funzionalita` del sistema.
- possibilita` di testing di applicazioni potenzialmente pericolose senza il rischio di interferire con altri utenti/macchine;
- possibilita` di trasferire le proprie macchine virtuali in supporti mobili (es: memorie USB, per continuare le esercitazioni sul computer di casa).

Dotazione hw: 5 server Intel-VT xeon (2 processori quadcore), storage unit CORAID 12TB

Software: **xen**

Unix & Linux

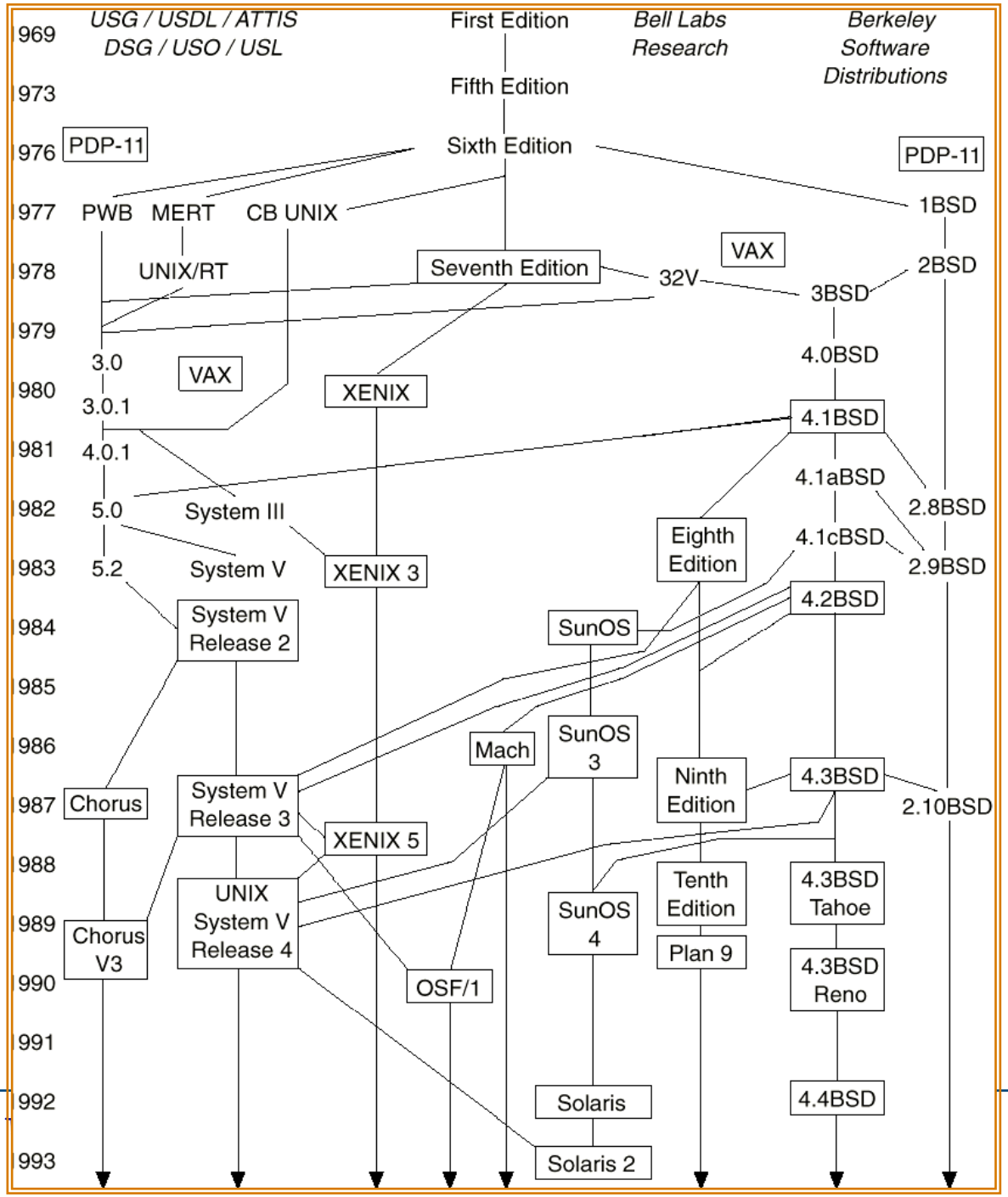
Storia di Unix

- **1969**: AT&T, sviluppo di un ambiente di calcolo multiprogrammato e portabile per macchine di medie dimensioni.
- **1970**: prima versione di UNIX (multiprogrammata e monoutente) interamente sviluppata nel linguaggio assembler del calcolatore PDP-7.
- **Anni 1970**: nuove versioni, arricchite con altre caratteristiche e funzionalità. Introduzione del supporto alla multiutenza.

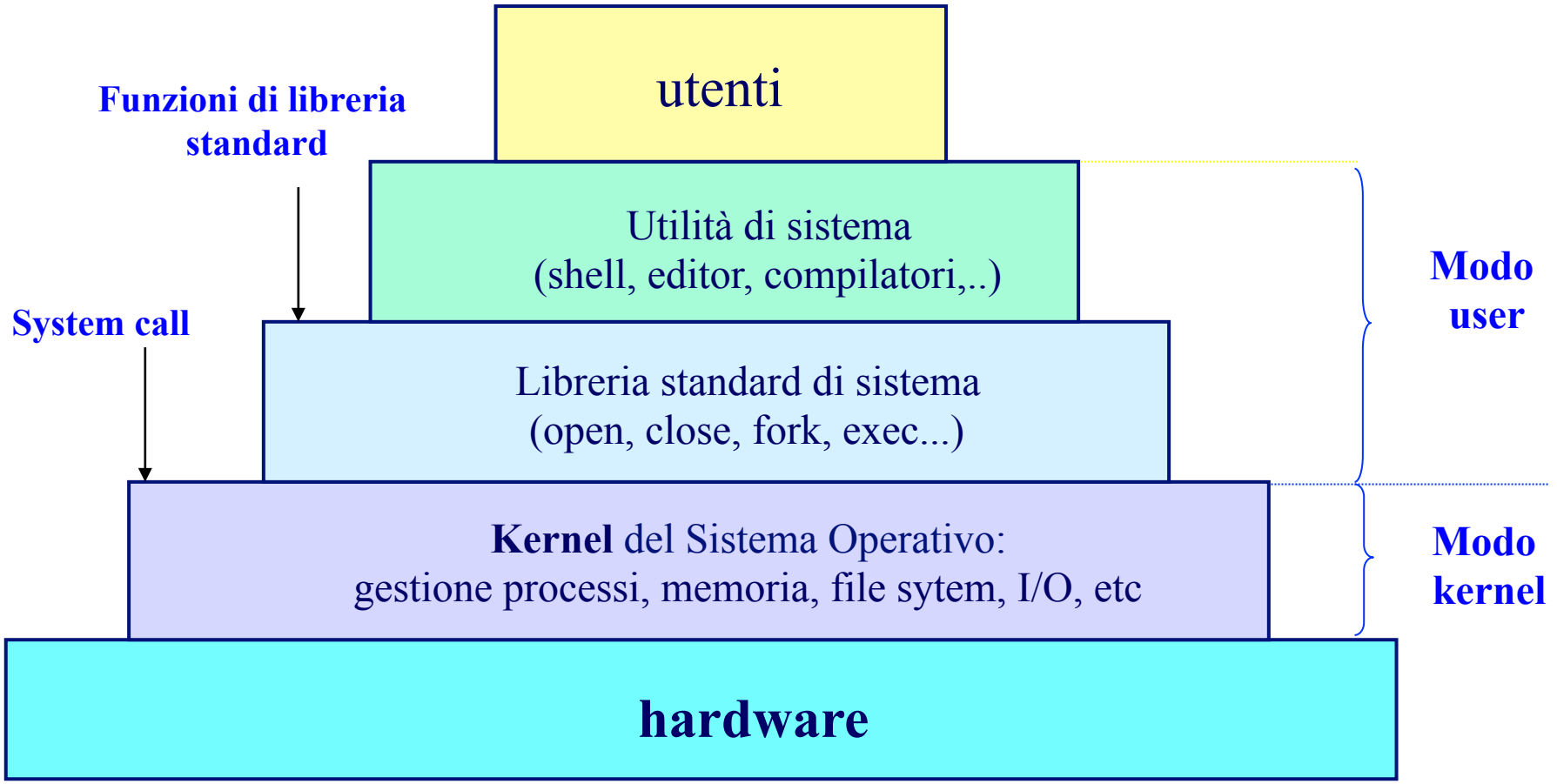
Unix e il linguaggio C

- **1973**: Unix viene realizzato nel linguaggio di programmazione C:
  - Elevata portabilità
  - Leggibilità
  - Diffusione presso la comunità scientifica e accademica.

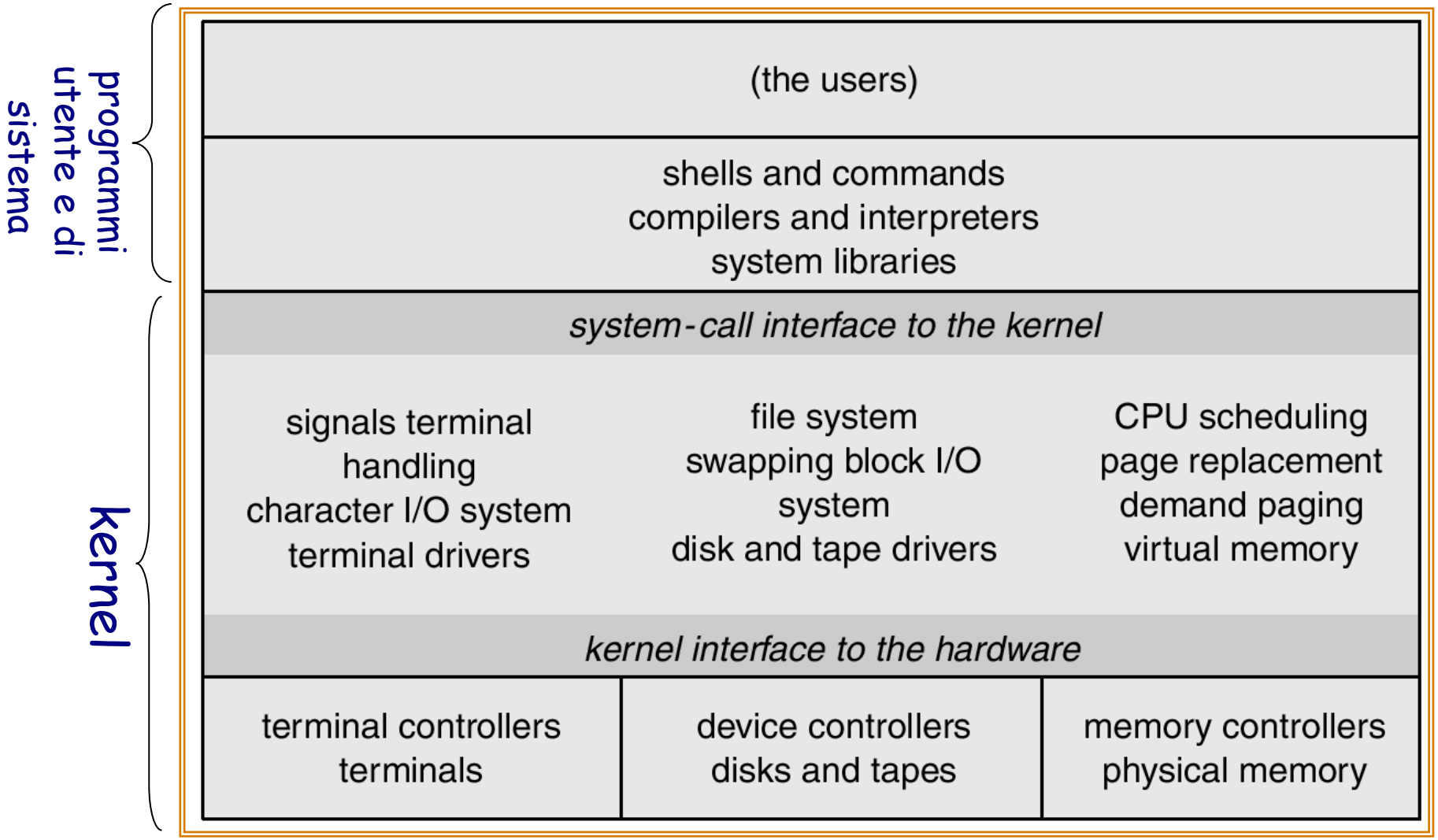
- **Anni 80**: la grande popolarità di Unix ha determinato il proliferare di versioni diverse. Due famiglie:
  - Unix System V (AT&T Laboratories)
  - Unix Berkeley Software Distributions, o BSD (University of California at Berkeley)


969
973
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
USG / USDL / ATTIS
DSG / USO / USL
Bell Labs
Research
Berkeley
Software
Distributions
First Edition
Fifth Edition
Sixth Edition
PDP-11
PWB
MERT
CB UNIX
UNIX/RT
Seventh Edition
32V
VAX
3BSD
1BSD
2BSD
3.0
VAX
XENIX
4.0BSD
3.0.1
4.1BSD
4.0.1
4.1aBSD
2.8BSD
5.0
System III
Eighth
Edition
4.1cBSD
2.9BSD
5.2
System V
XENIX 3
4.2BSD
System V
Release 2
SunOS
Mach
SunOS
3
Chorus
System V
Release 3
XENIX 5
Ninth
Edition
4.3BSD
2.10BSD
UNIX
System V
Release 4
Tenth
Edition
4.3BSD
Tahoe
Chorus
V3
SunOS
4
Plan 9
4.3BSD
Reno
OSF/1
Solaris
4.4BSD
Solaris 2

Organizzazione di Unix



Organizzazione di UNIX



Caratteristiche di Unix

- multi-utente
- time sharing
- kernel monolitico
- Ambiente di sviluppo per programmi in linguaggio C
- Programmazione mediante linguaggi comandi
- portabilità

POSIX

- **1988**: POSIX (*Portable Operating Systems Interface*) è lo standard definito dall'IEEE. Definisce le caratteristiche relative alle modalità di utilizzo del sistema operativo.
- **1990**: POSIX viene anche riconosciuto dall' International Standards Organization (ISO).
- **Anni 90**: Negli anni seguenti, le versioni successive di Unix SystemV e BSD (versione 4.3), si uniformano a POSIX.

Introduzione a GNU/Linux

- GNU project:
  - **1984**: Richard Stallman avvia un progetto di sviluppo di un sistema operativo ***libero* compatibile con Unix:**

    "GNU is Not Unix"
  - Furono sviluppate velocemente molte utilita` di sistema:
    - editor Emacs,
    - Compilatori: gcc,
    - shell: bash,
    - ...
  - lo sviluppo del kernel (Hurd), invece, subi` molte vicissitudini e vide la luce molto piu` tardi (1996)

GNU/Linux

- **1991**: Linus Tornvalds realizza un kernel Unix-compatibile (Minix) per l'architettura intel x86 e pubblica su web i sorgenti
- In breve tempo, grazie a una comunita` di *hacker* in rapidissima espansione, Linux acquista le caratteristiche di un prodotto affidabile e in continuo miglioramento.
- **1994**: Linux viene integrato nel progetto GNU come kernel del sistema operativo: nasce il sistema operativo GNU/Linux

GNU/Linux

Caratteristiche:

- Open Source / Free software
- multi-utente, multiprogrammato e multithreaded
- Kernel monolitico con possibilita` di caricamento dinamico di moduli
- estendibilita`
- affidabilita`: testing in tempi brevissimi da parte di migliaia di utenti/sviluppatori
- portabilità