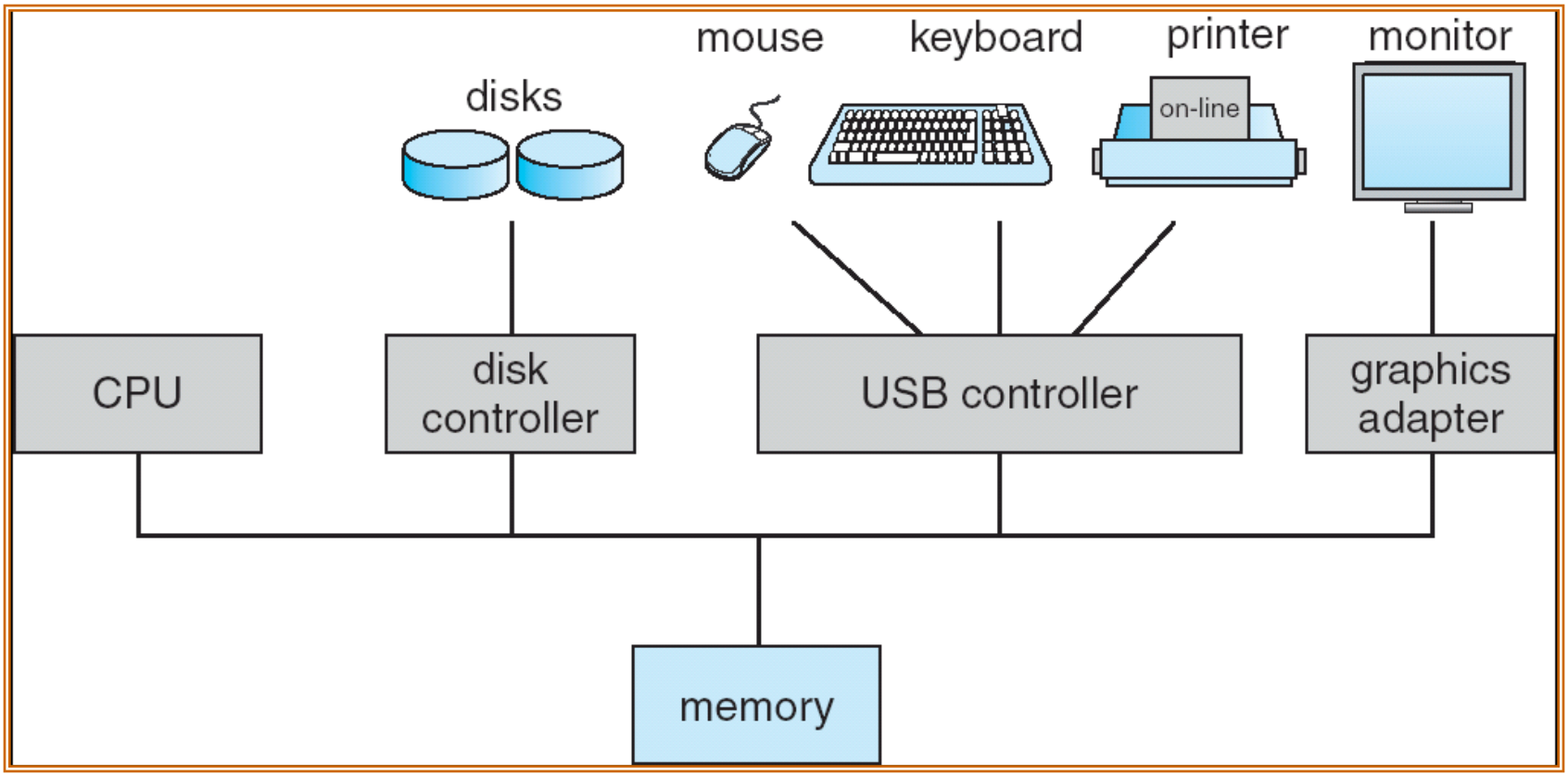
STRUTTURA DI UN ELABORATORE

Architettura di un sistema di calcolo

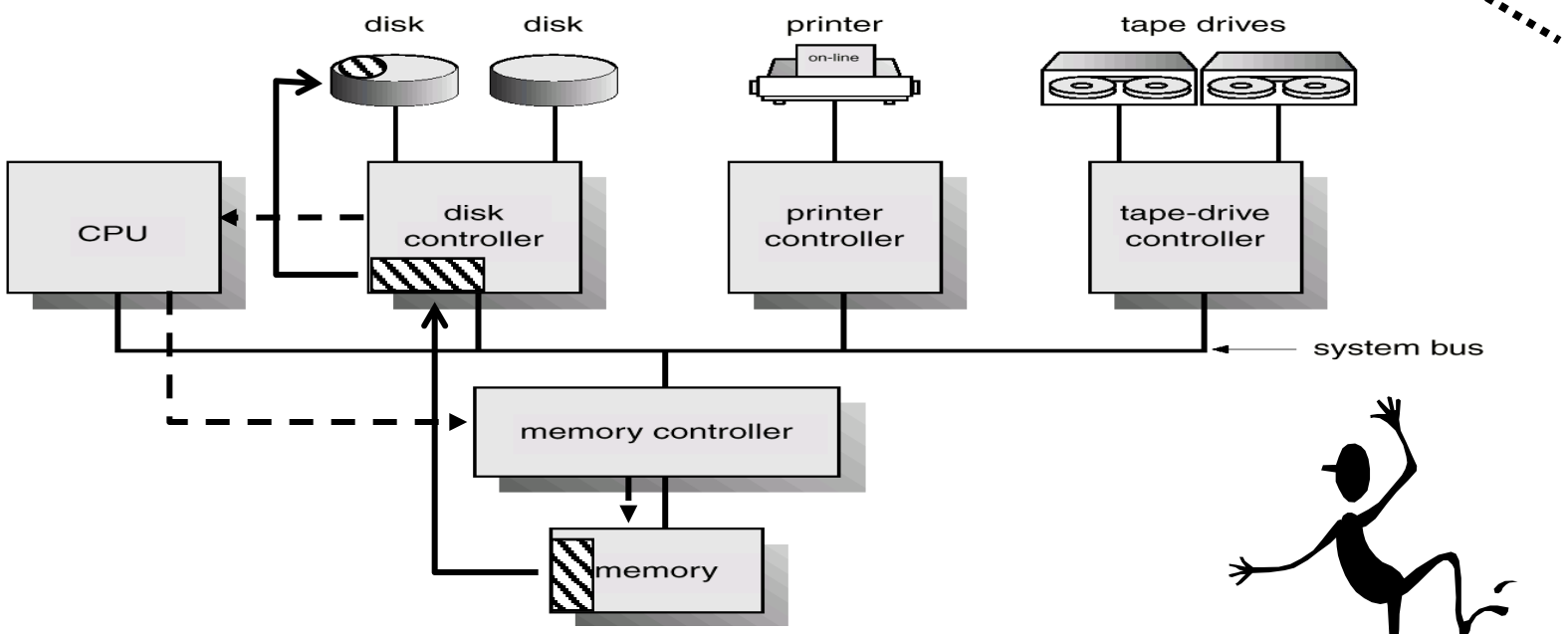


Caratteristiche generali…

- Dispositivi di I/O e CPU possono operare allo stesso tempo e competere per ***l'accesso a memoria*** (regolato dal controller di memoria, unico nel suo genere)
- I ***controller*** sono la parte ***intelligente*** di un dispositivo
- Ogni ***controller*** e' predisposto per un certo ***tipo di dispositivo***
- Esso ha ***registri*** e ***buffer locali*** per gestire le interazioni con le altre parti del sistema
- Il ***device driver*** e' la controparte del dispositivo in OS, e cioe' la ***parte di OS*** predisposta a interagire con un certo dispositivo

… e meccanismi di I/O

- La ***CPU*** sposta i dati da/a ***memoria principale*** a/da ***buffer del dispositivo***
  - ***I/O*** avviene da/a ***buffer del dispositivo*** a/da ***dispositivo***
  - Il ***controller*** informa la CPU, mediante un ***interrupt***, che ha ***finito*** l' operazione



Quanti di voi hanno mai sentito parlare di *Interrupt* ?!

… e aiutiamo allora ad alzare le altre mani…

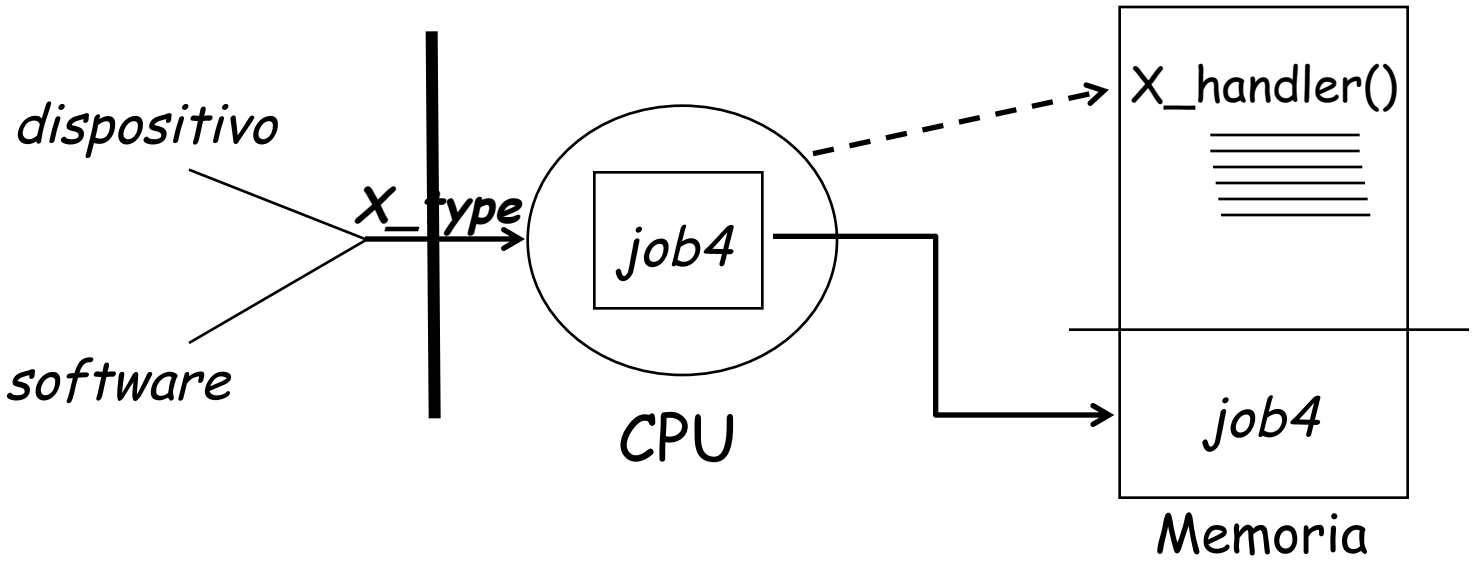
Quanti di voi hanno mai sentito parlare di *Interrupt* ?!

OS e' ***interrupt driven***

- ***Perche'?***
  - Interrupt regolano le interazioni tra OS e dispositivi e tra OS e programmi utente
  - In particolare, un interrupt viene generato **al termine di un operazione di I/O**
- ***Esempi***
  - Un interrupt viene generato dal controller di un disco quando un' operazione di scrittura e' terminata
  - Una ***trap*** (interrupt generato via software) puo' essere causata da un errore (ex. divisione per 0) o dalla richiesta esplicita (da parte di un programma utente) di un servizio del OS (system call, ex. ***mkdir***)

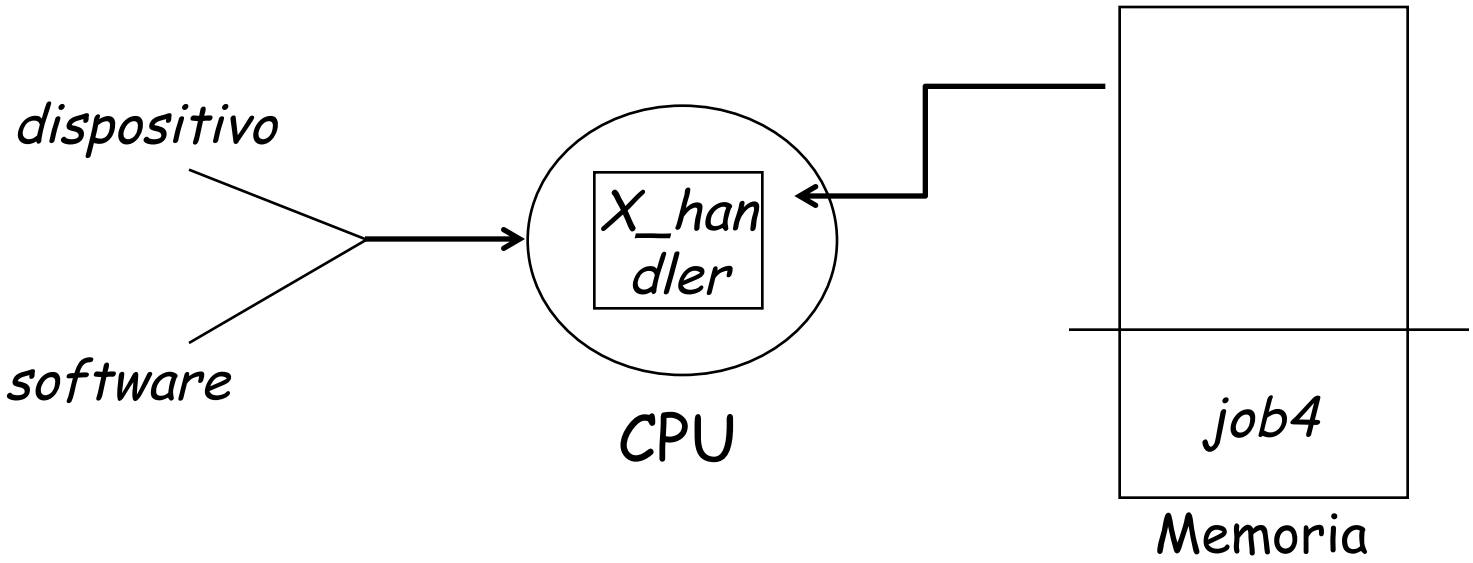
Gestione di un interrupt

- Salvataggio dell' indirizzo dell' istruzione corrente e di tutti i registri della CPU essenziali per il ***ripristino***
- Interrupt successivi sono ***disabilitati*** durante la gestione di un interrupt per evitare perdite
- Differenti segmenti di codice (routine di gestione dell' interrupt) determinano le azioni da intraprendere per ogni ***tipo di interrupt***



Gestione di un interrupt

- Salvataggio dell' indirizzo dell' istruzione corrente e di tutti i registri della CPU essenziali per il ***ripristino***
- Interrupt successivi sono ***disabilitati*** durante la gestione di un interrupt per evitare perdite
- Differenti segmenti di codice (routine di gestione dell' interrupt) determinano le azioni da intraprendere per ogni ***tipo di interrupt***



Determinare il tipo di interrupt

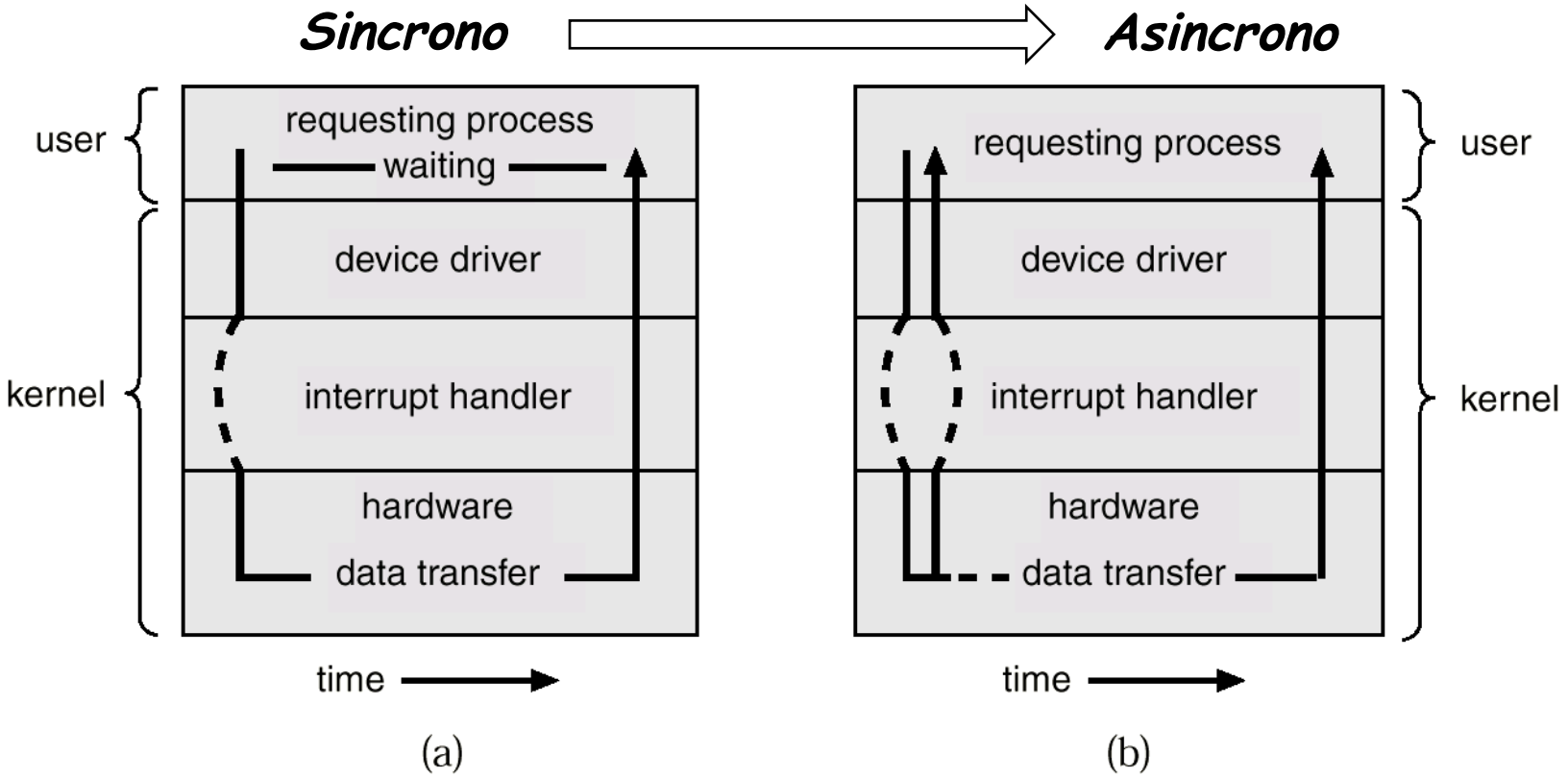
POLLING

La CPU riceve un ***anonimo interrupt*** e quindi ***interroga tutti i dispositivi*** per vedere chi ha da eseguire un' operazione di I/O e quindi esegue la routine appropriata

VECTORED INTERRUPT SYSTEM

La CPU riceve un ***interrupt di un certo tipo*** e, mediante l' interrupt vector, manda in esecuzione la routine appropriata.
Interrupt vector : mappa tipi di interrupt a indirizzi di routine di gestione interrupt

Evoluzione dell'Input/Output



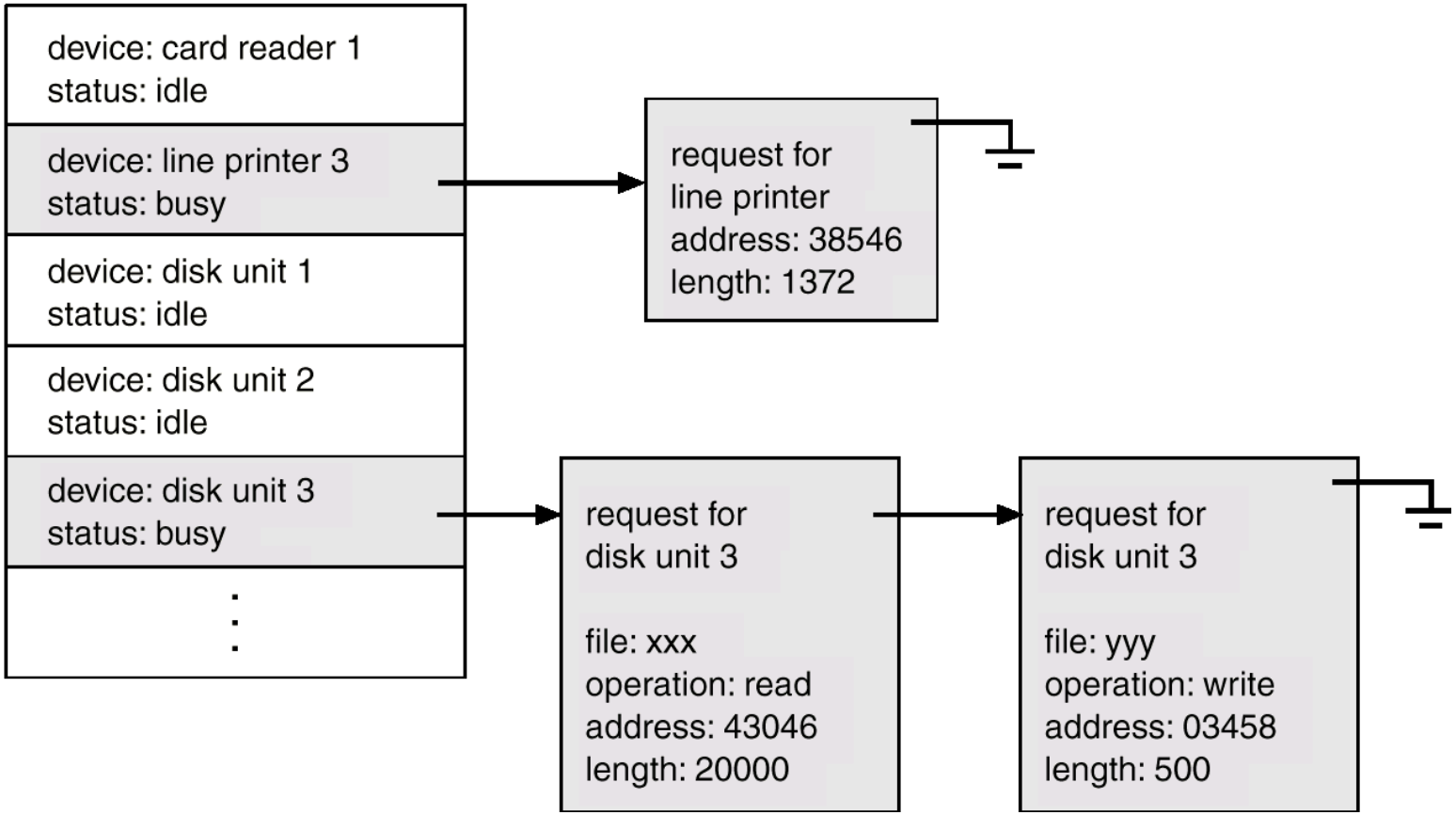
Dopo che I/O inizia, il controllo ritorna all'utente ***senza attendere il completamento del I/O***

Evoluzione dell'Input/Output

- ***Sincrono***
  - Al piu' ***una richiesta*** di I/O ***alla volta*** per ogni processo

- ***Asincrono***
  - Anche un solo processo puo' originare piu' richieste contemporanee:
    - **Device-status table** *contiene una entry per ogni dispositivo di I/O che ne indica il tipo, l'indirizzo e lo stato*
    - **OS accede alla table** *per determinare lo stato di un dispositivo e per modificare la table entry in caso di interrupt*

Device-Status Table…



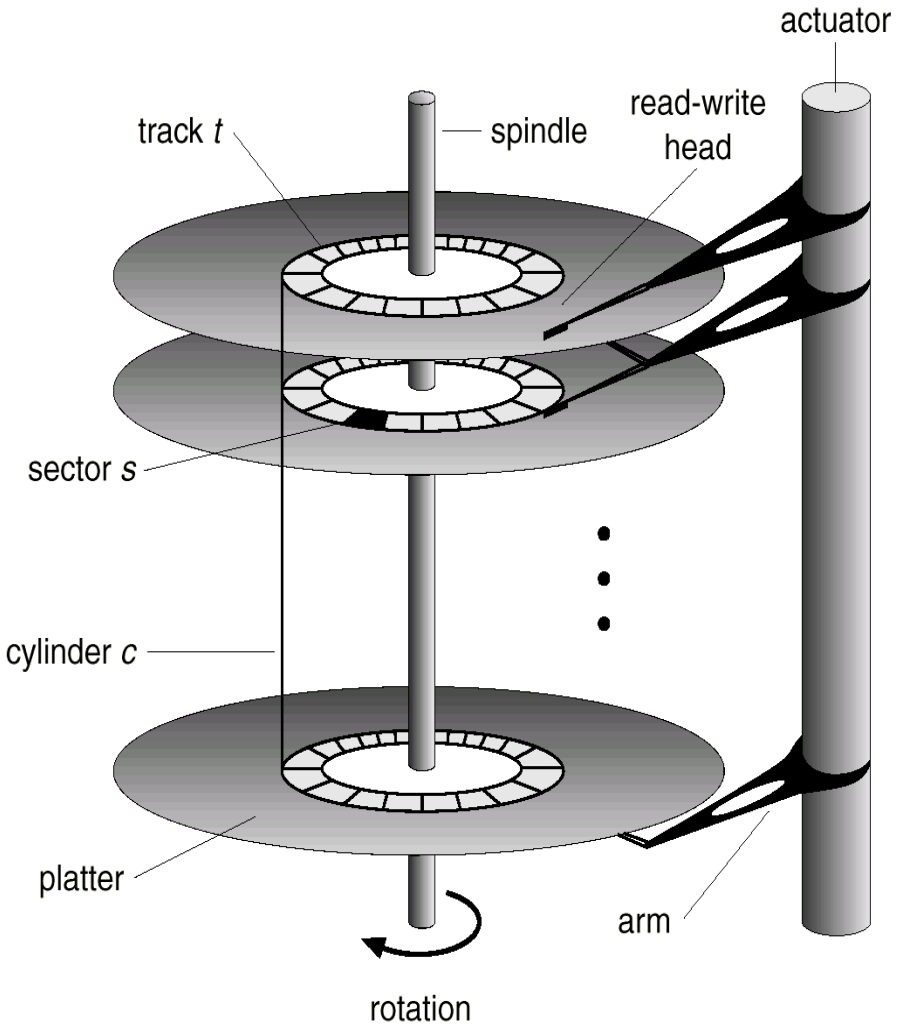
… e sua gestione

- ***Code di richieste*** in attesa su dispositivi

- Quando arriva un interrupt al OS per un certo dispositivo, OS va a ***modificare*** semplicemente ***la coda***

- ***In uscita*** da un'operazione di I/O il ***controllo*** puo' essere ***ridato*** al processo che stava in esecuzione prima dell'interrupt, a quello che stava in attesa di completamento di questa operazione, oppure a un altro processo

Gestione dell'I/O su un disco



tempo di trasferimento (MB/sec)
tempo di seek e latenza (msec)

Qualche eccezione nei meccanismi di I/O

Tastiera

Consente di ***inserire informazioni prima*** ancora che il programma cui sono diretti ne faccia ***richiesta***

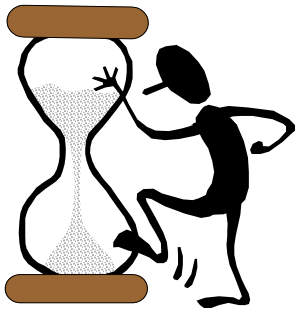
Controller della Tastiera

- Segnala la ***presenza di informazioni*** e nel frattempo…

- … utilizza un ***buffer locale*** per conservare le informazioni inserite ***in attesa della corrispondente richiesta***

Direct Memory Access (DMA) Structure

PROBLEMA : Device ***troppo veloci*** con frequenti interrupt prenderebbero ***troppa CPU***

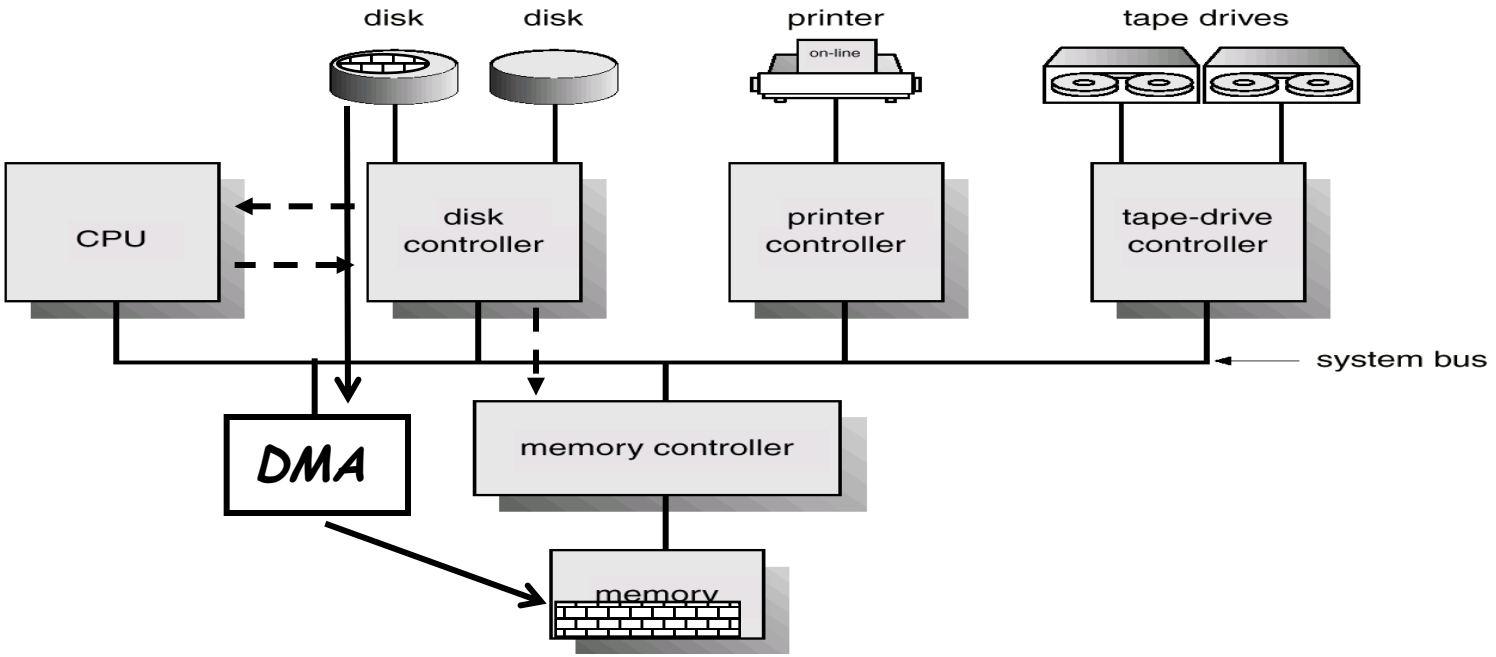


IDEA : ***un unico interrupt per un insieme piu' ampio di informazioni***

Direct Memory Access (DMA) Structure

COME:

Dopo avere settato buffers, puntatori e counters, il controller spedisce un insieme di blocchi di dati dal suo buffer alla memoria (o viceversa) senza intervento ulteriore della CPU



Struttura astratta della *memoria*

- ***Memoria centrale***
  - il solo dispositivo di memoria al quale la CPU puo' ***accedere direttamente***

- ***Memoria secondaria*** (di massa)
  - Estensione della memoria centrale che fornisce grande capacita' di ***memoria non volatile***

Memoria centrale…

Tutto cio' che "accade" in un sistema di calcolo passa per la memoria centrale

Load e ***store*** per trasferire da memoria a registri della CPU e viceversa

Memoria ***volatile***, ad ***accesso veloce***, dalla ***capacita' limitata***

Dal punto di vista OS, si tratta di gestire le possibili ***sequenze di indirizzi*** generate per l'accesso a memoria

… e rapporto con I/O

I/O mappato in memoria

Registri dei dispositivi mappati a indirizzi di memoria

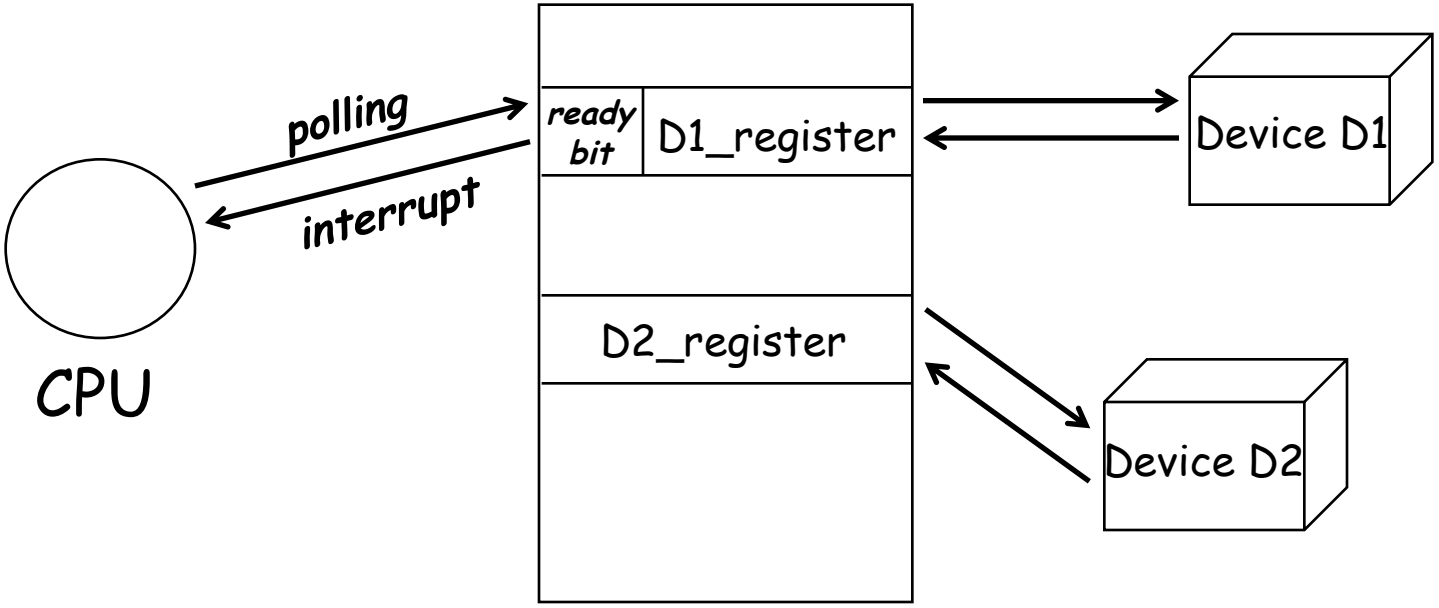
- per dispositivi con ***brevi tempi di risposta***, come un video controller (locazioni di memoria che corrispondono a posizioni sullo schermo)

- per ***porte*** seriali/parallele, in quanto e' facile mappare una porta ad un indirizzo di memoria

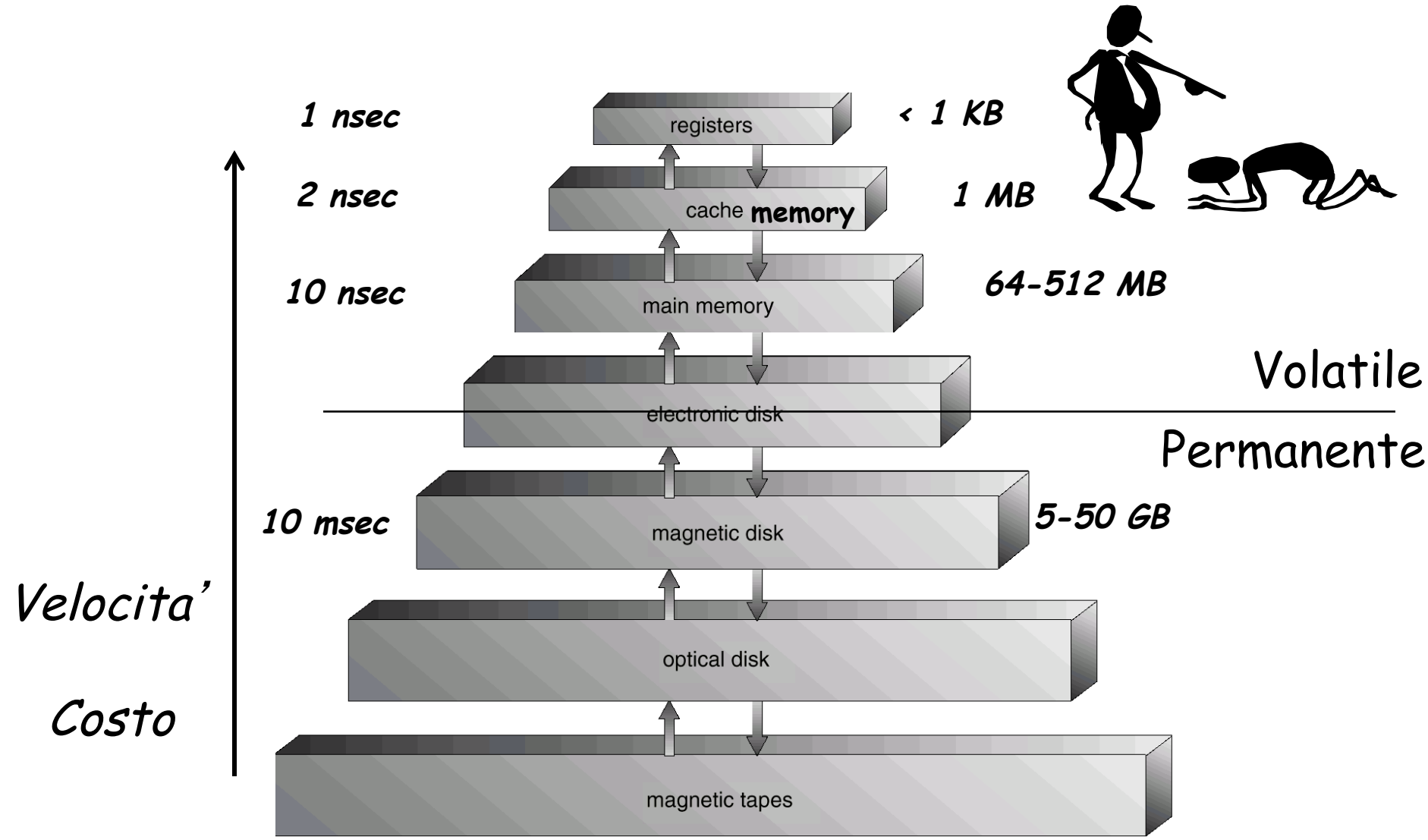
Modalita' di interazione in I/O mappato

- la CPU fa polling su un ready bit (***programmed I/O***)

- la CPU riceve un interrupt quando il dispositivo e' pronto (***interrupt driven I/O***)



Gerarchia dei dispositivi di memoria…



… qualche osservazione …

- ***Electronic disk*** e' accessibile come un file system (volatile o non volatile alimentato a batteria)

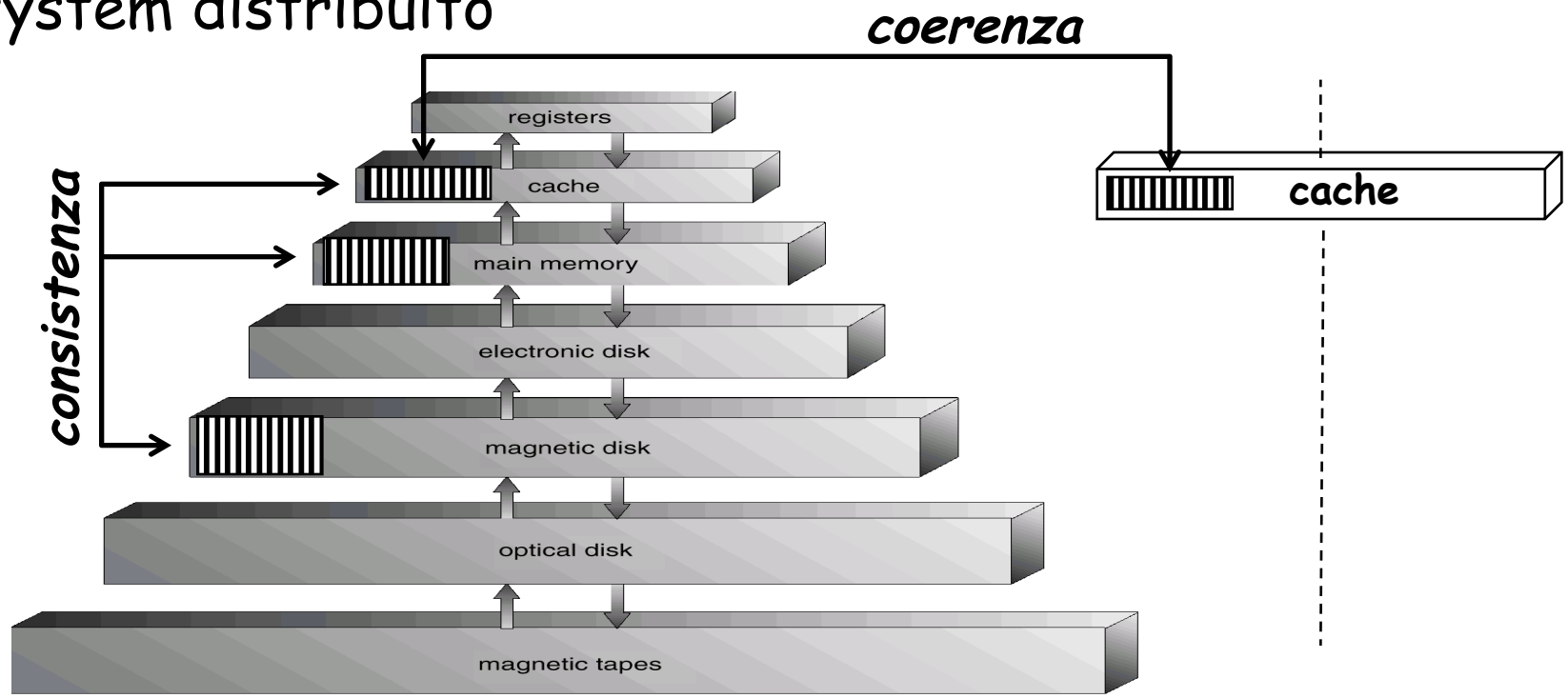
- ***Cache*** interamente gestite ***via hardware*** (ex., next instruction register)

- ***Movimento*** tra i vari livelli ***sia implicito che esplicito***

- In generale ***caching*** e' l'operazione di ***copia di informazioni in dispositivi di memoria piu' veloci*** (es. la memoria centrale puo' essere vista come una cache per la memoria secondaria)

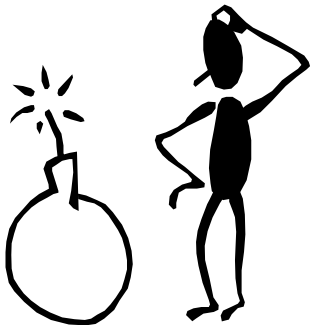
… e qualche attributo

- ***Consistenza*** tra copie dello stesso item in diverse memorie
- ***Coerenza*** tra diverse cache (in un multiprocessor, per esempio) oppure tra diverse repliche di un file in un file system distribuito



Hardware Protection

1. Dual-Mode Operation
2. I/O Protection
3. Memory Protection
4. CPU Protection



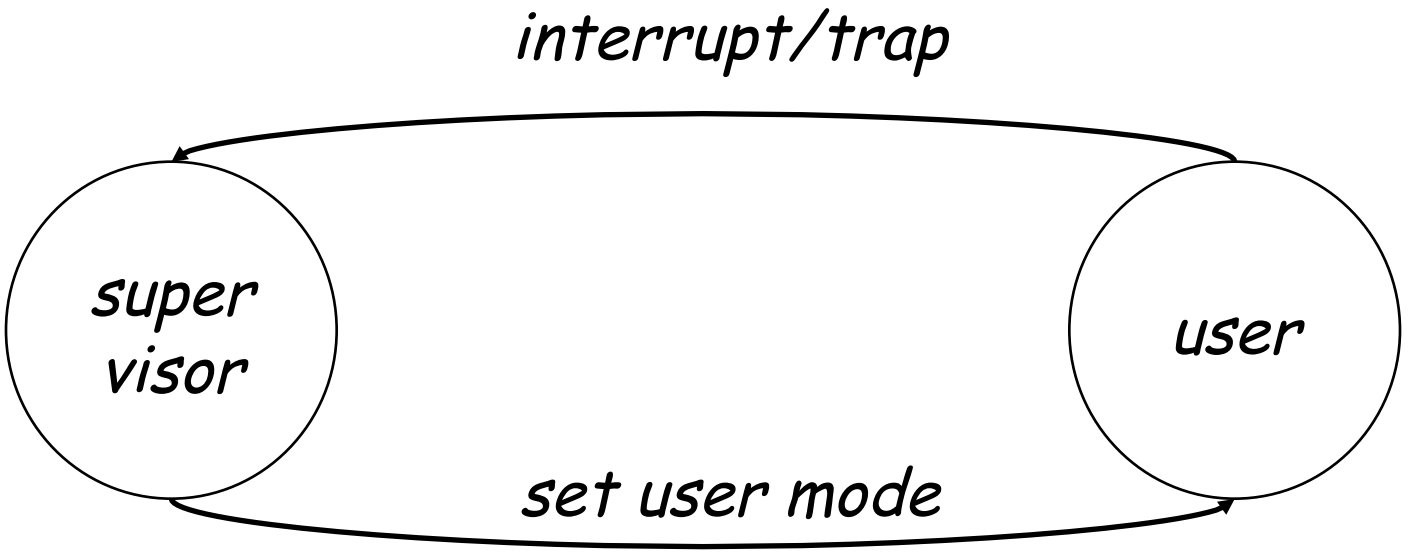
1. Dual-Mode Operation…

In un sistema a condivisione di risorse OS deve assicurare che ***un programma non possa danneggiare*** (non solo se' stesso ma anche) ***gli altri programmi***

Gestione "normale" di errore: emissione di una ***trap*** , dump del core

Supporto hardware per avere (almeno) due modi di operare:
Modo utente - da parte di un utente
Modo supervisor - da parte di OS

… e relative transizioni



Le ***istruzioni*** "pericolose" (***privilegiate***) possono essere eseguite ***solo in modo supervisor***

2. I/O Protection

- Regola: un programma ***utente*** non deve ***mai guadagnare*** controllo del computer in ***supervisor mode***

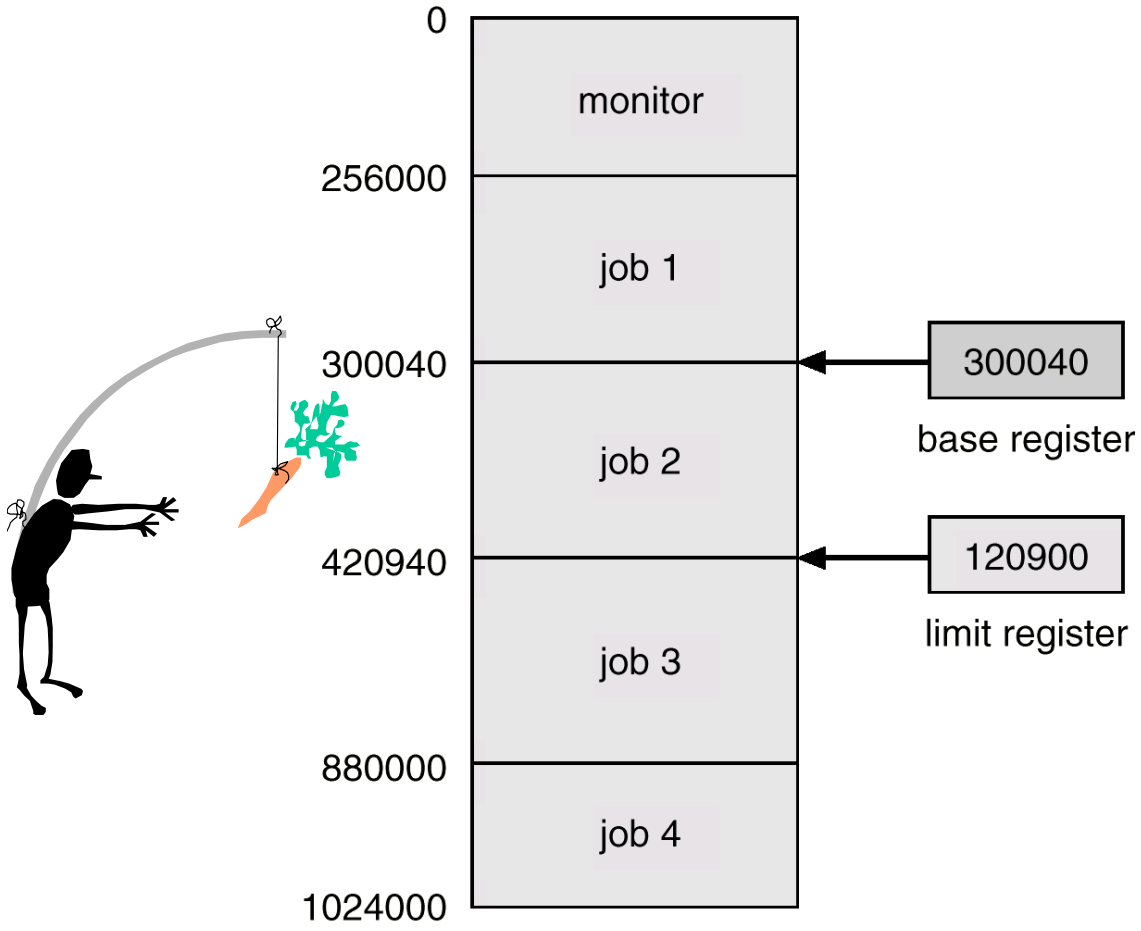
- Come potrebbe accadere:
  - ***scrivendo*** un ***indirizzo*** che appartiene al proprio spazio di indirizzi nell' interrupt vector
  - ***modificando*** il ***codice*** di un interrupt handler per deviarlo ad eseguire nel proprio spazio di indirizzi

- Soluzione: tutte le ***istruzioni di I/O sono privilegiate***

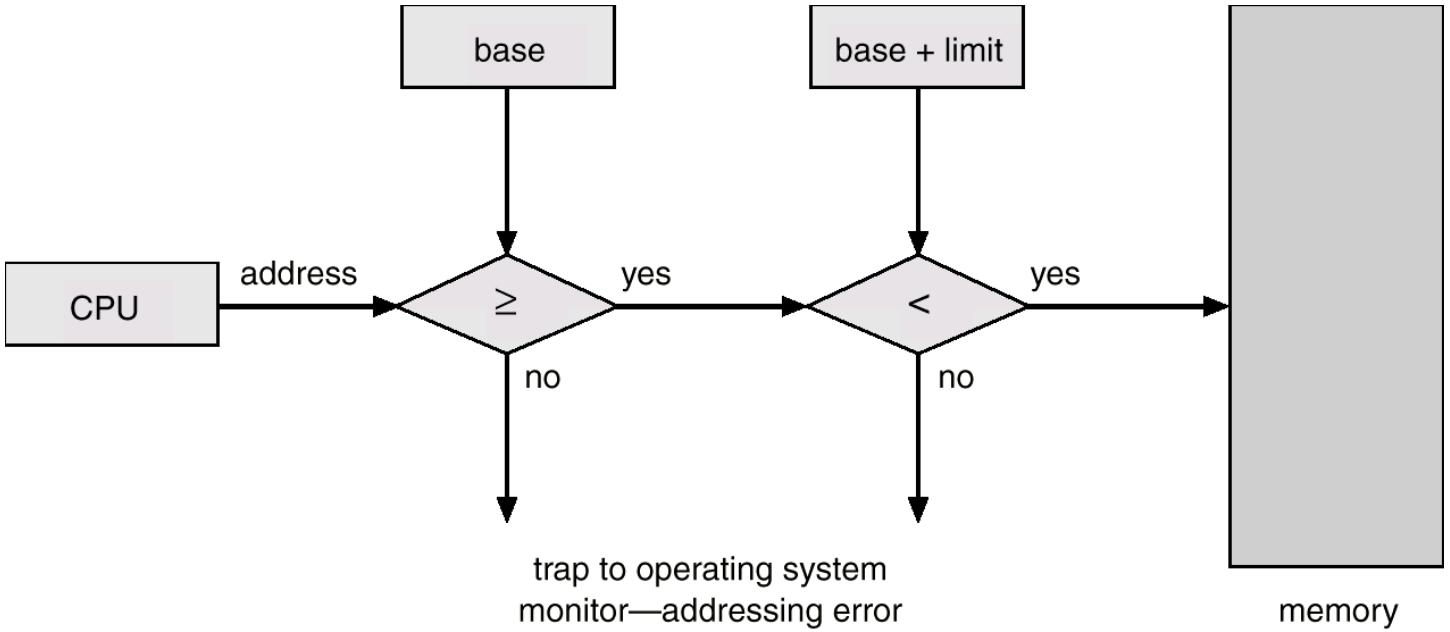
3. Memory Protection

- Proteggere interrupt vector e codice delle interrupt routines
- Due registri che determinano il range di indirizzi leciti di un processo:
  - **base register** – il piu' piccolo indirizzo fisico di memoria lecito del processo
  - **limit register** – taglia del range di indirizzi del processo
- La memoria al di fuori del range deve essere protetta

Spazio di indirizzi determinati da *base* e *limit*



Congegno di protezione (hardware) basato su *base e limit*



- In modo supervisor si ha accesso sia allo spazio user che quello supervisor
- Le ***istruzioni*** che manipolano ***i registri base e limit*** sono ***privilegiate***

4. CPU Protection

Regola : Evitare che un ***processo*** possa detenere per un ***tempo illimitato*** il controllo della ***CPU***

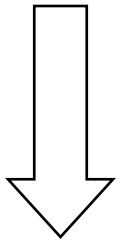
Soluzione : un timer che interrompe l' esecuzione dopo un certo tempo e restituisce il controllo a OS (ex. quantum in time sharing)

Le istruzioni di ***manipolazione del timer*** sono ***privilegiate***

RUOLO DEL SISTEMA OPERATIVO

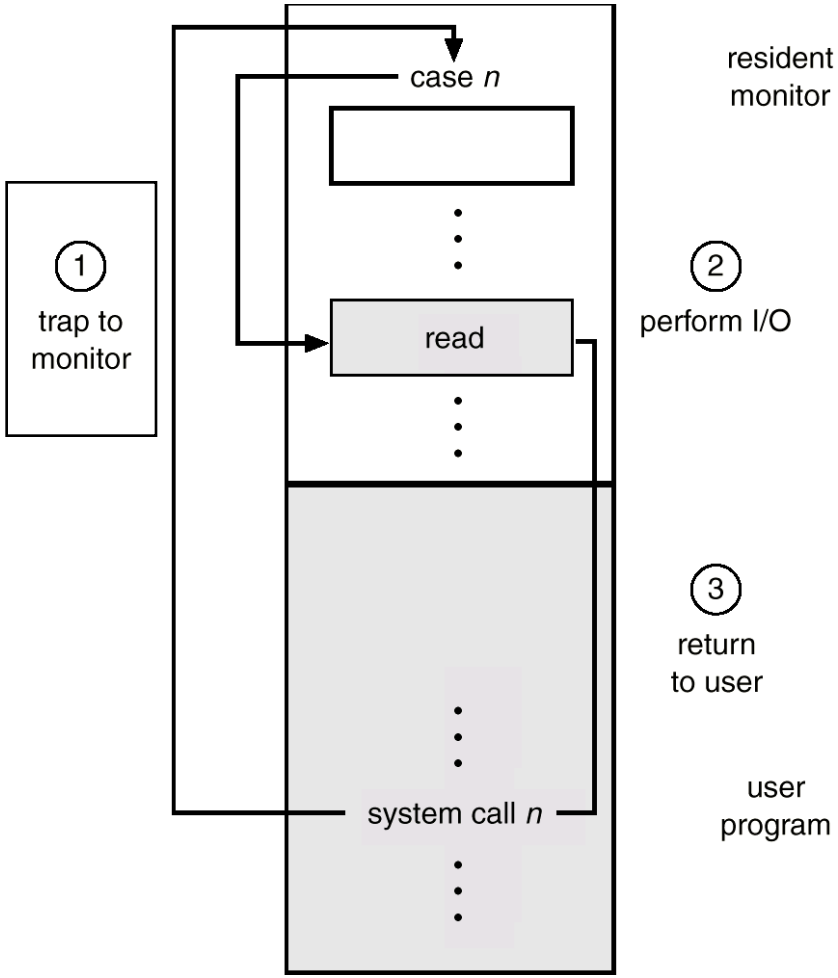
Meccanismo per l'esecuzione di istruzioni privilegiate

- OS deve sempre mantenere il controllo
- Come fa un processo utente ad accedere alle risorse se le istruzioni relative sono tutte privilegiate?



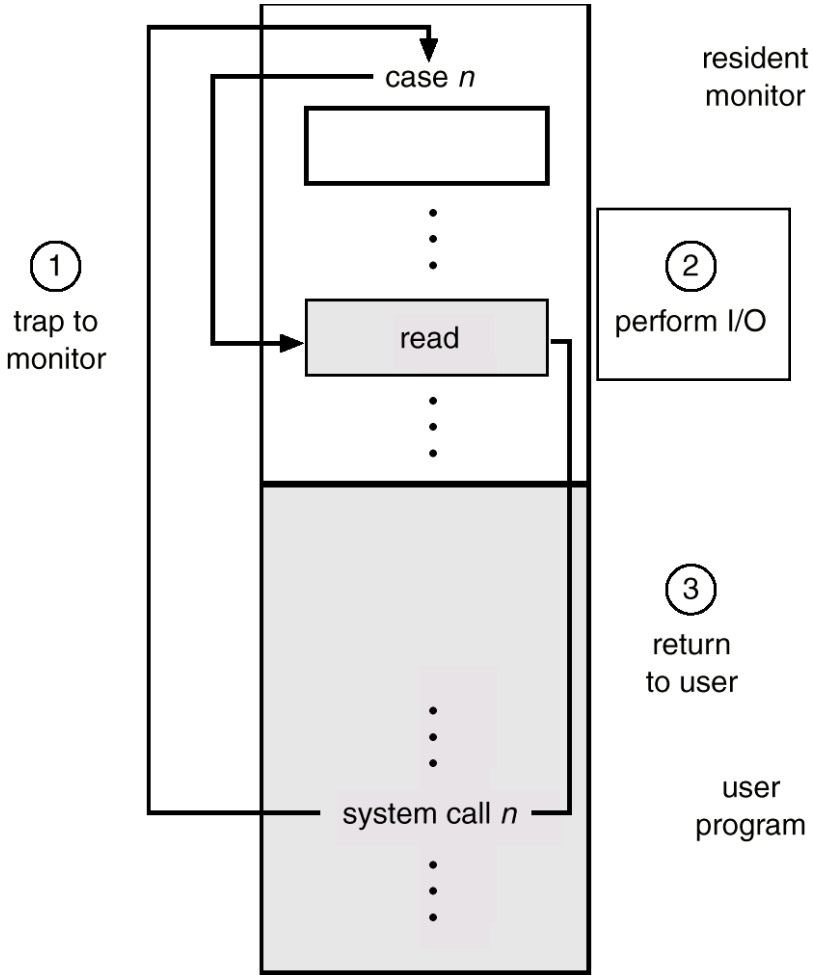
System call – il metodo usato da un processo per richiedere un'azione di OS

Uso di una system call per I/O



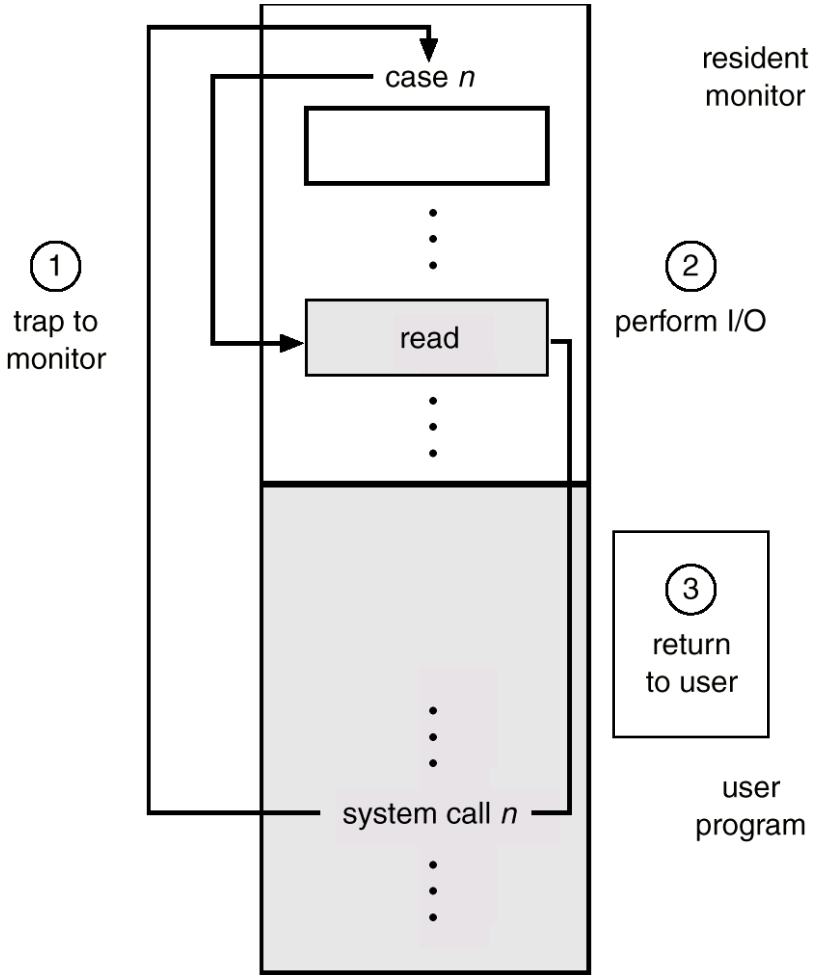
1 - Usualmente prende la forma di una ***trap*** ad una specifica locazione dell' interrupt vector

Uso di una system call per I/O



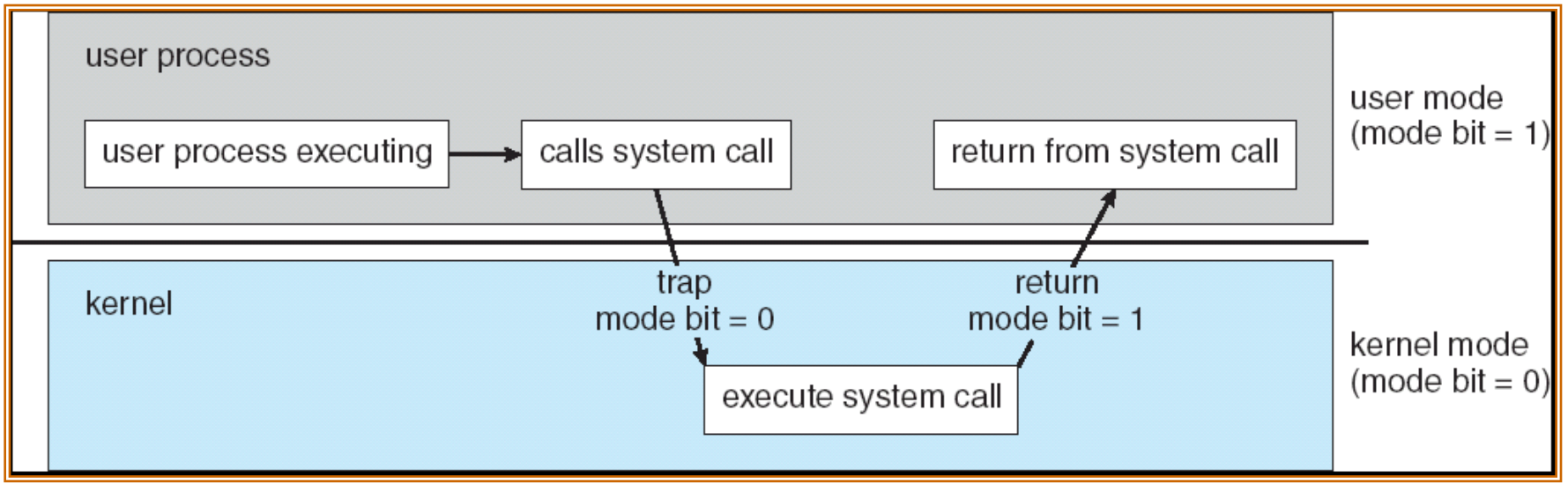
2 - Il controllo passa ad una ***routine di servizio di OS***, ed il ***mode*** viene settato ***a supervisor***

Uso di una system call per I/O

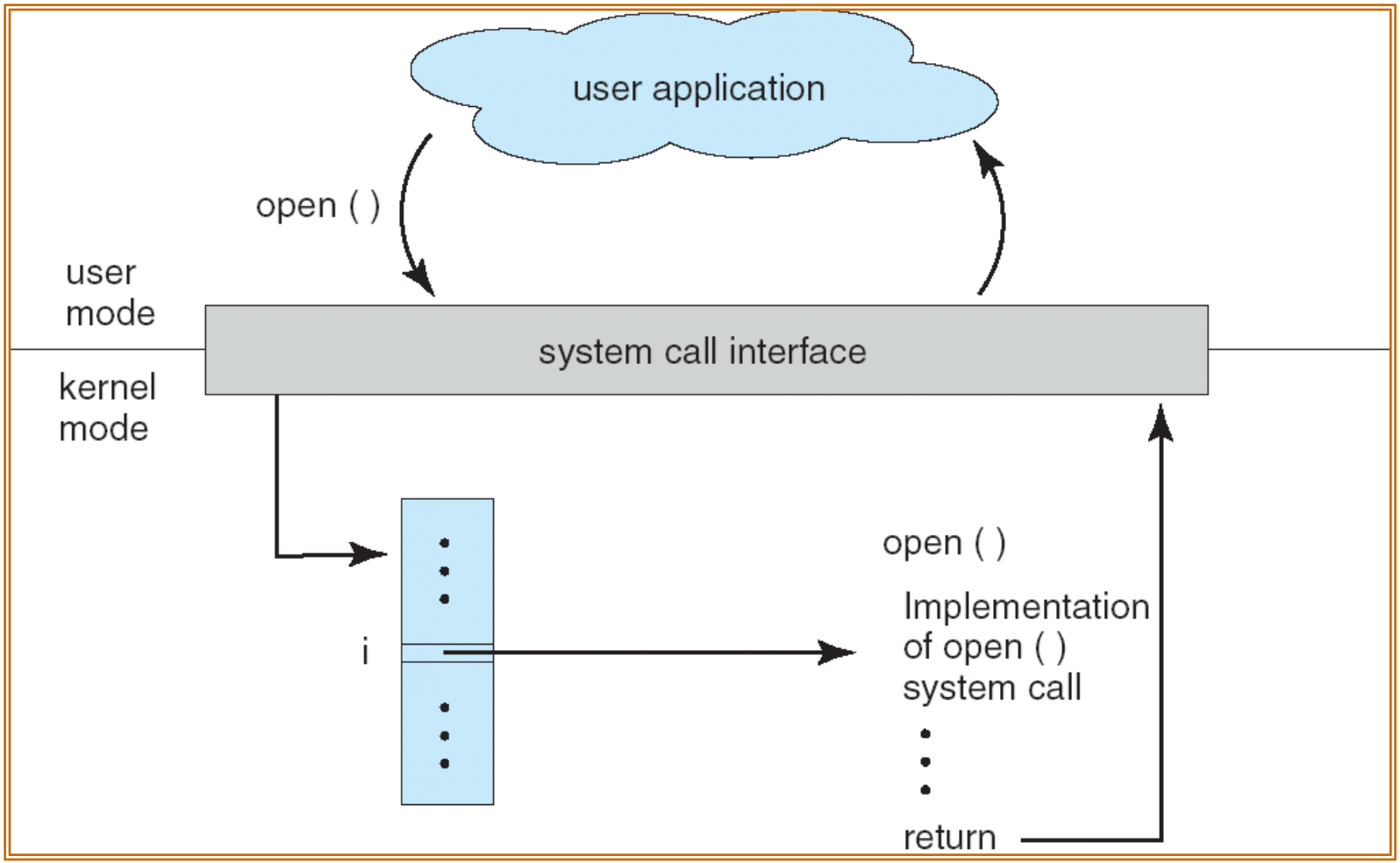


3 - OS ***verifica*** che i ***parametri*** (caricati in qualche registro) sono ***corretti e leciti***, esegue la richiesta, e ***ritorna*** il controllo ***a livello utente***

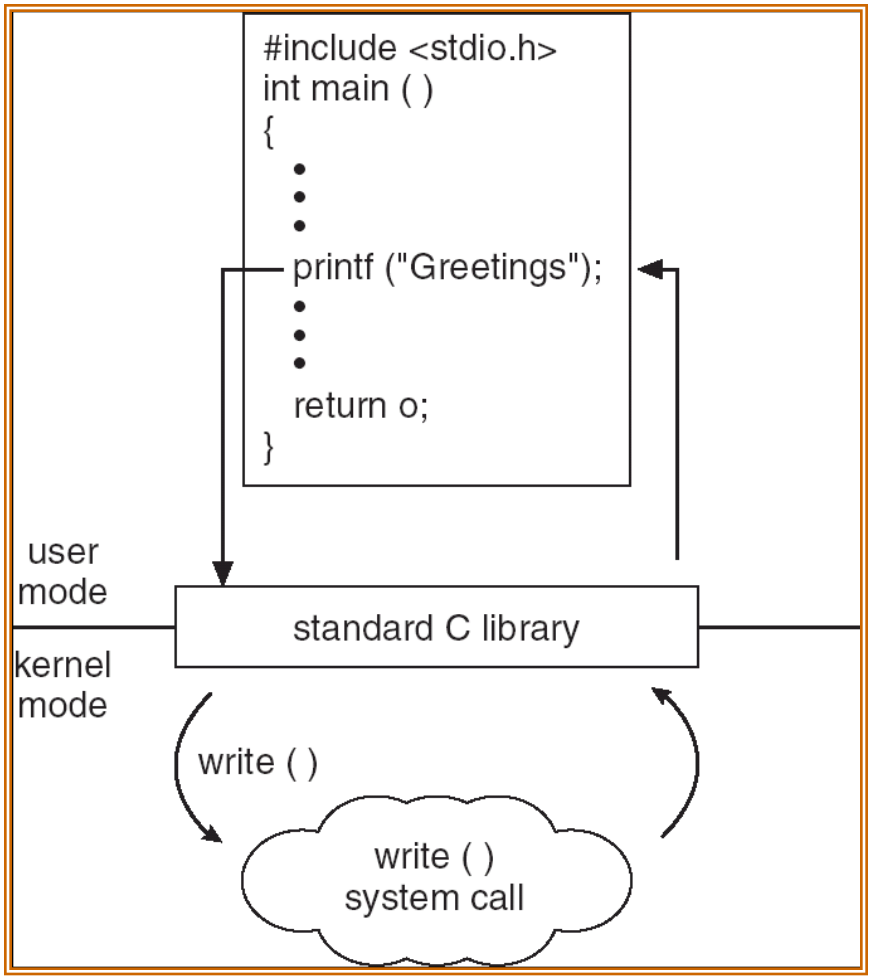
In sintesi, da user a kernel mode



Gestione di una open()...



… e di una printf() in linguaggio C



A. Gestione dei processi

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

- Fork
- Execute
- End
- Abort
- …

Debugging e monitoring

- Esecuzione step-by-step
- Profilo temporale del program counter

B. Manipolazione dei file

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

- Create
- Delete
- Open
- Read
- ...

C. Manipolazione di dispositivi

- Request
- Release
- Alloc
- …

D. Manutenzione delle informazioni

- Time
- Date
- …

E. Comunicazioni…

- Get Hostid
- Open connection
- Close connection
- Send
- …

Componenti comuni di un OS

1. Gestione dei processi
2. Gestione della memoria centrale
3. Gestione della memoria secondaria
4. Manipolazione dei file
5. Gestione dell' I/O
6. Sistema di protezione
7. Gestione di rete
8. Interprete dei comandi

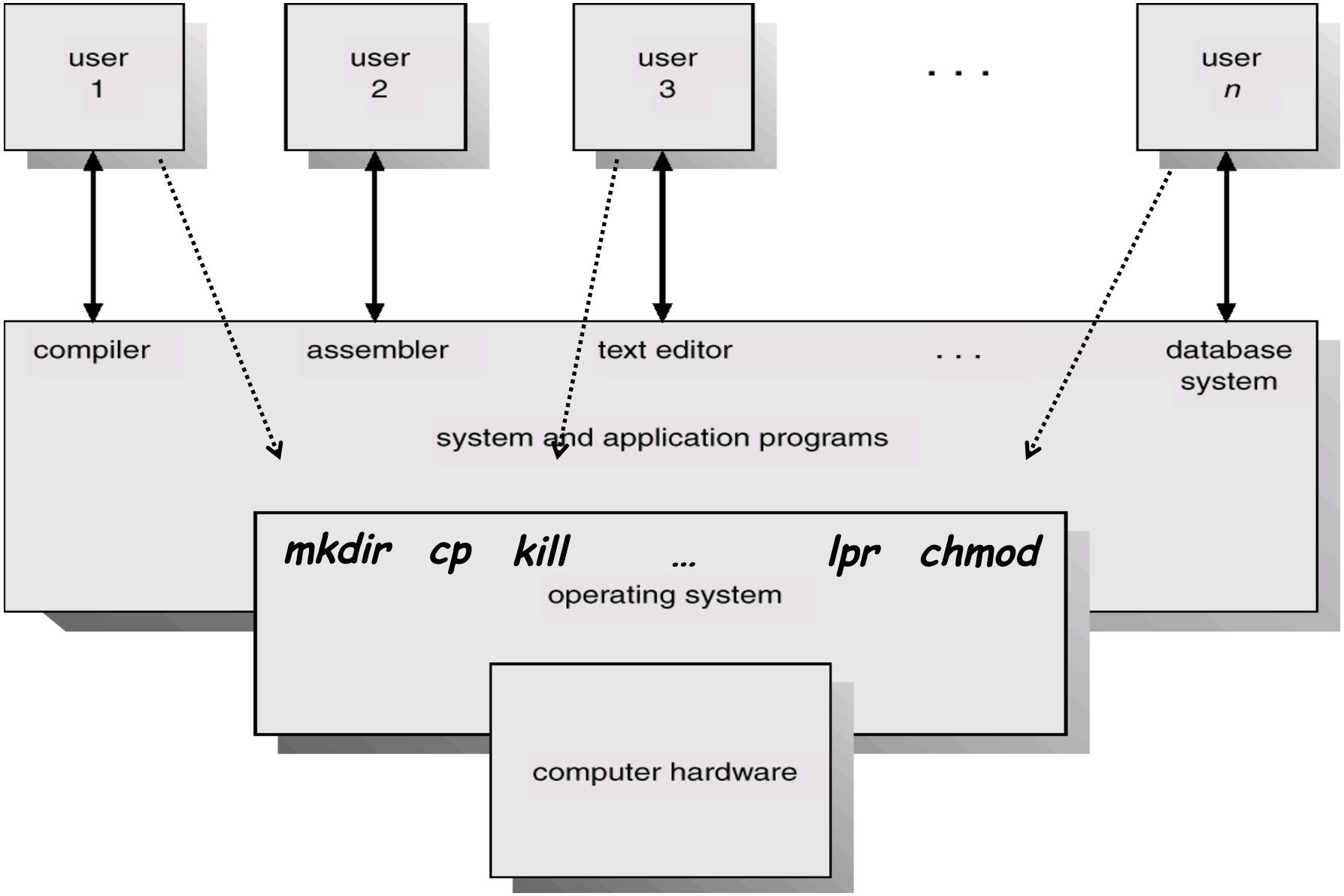
8. Interprete dei comandi (shell)

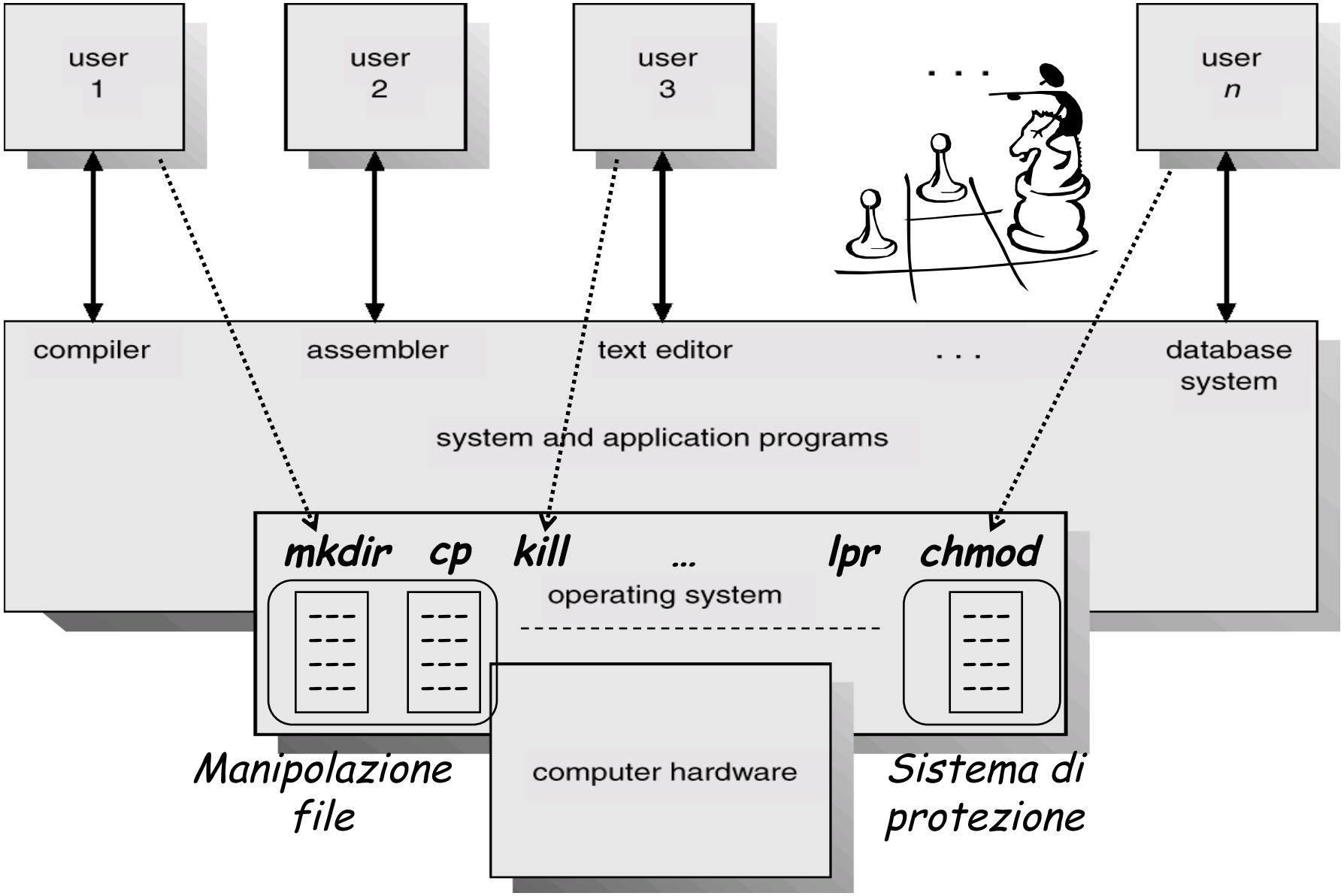
La ***shell*** e' il ***programma*** che si occupa di accettare, interpretare ed ***eseguire il prossimo comando***

L' insieme di comandi costituisce il ***linguaggio di colloquio*** tra ***utente*** e OS

Piu' o meno ***facili e user-friendly*** shell, a seconda di OS

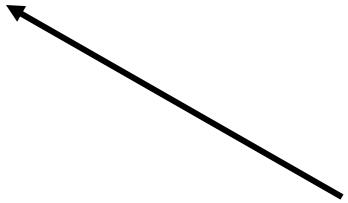
Esempio : UNIX shell




user 1
user 2
user 3
. . .
user n
compiler
assembler
text editor
. . .
database system
system and application programs
mkdir
cp
kill
...
lpr
chmod
operating system
computer hardware
Manipolazione file
Sistema di protezione

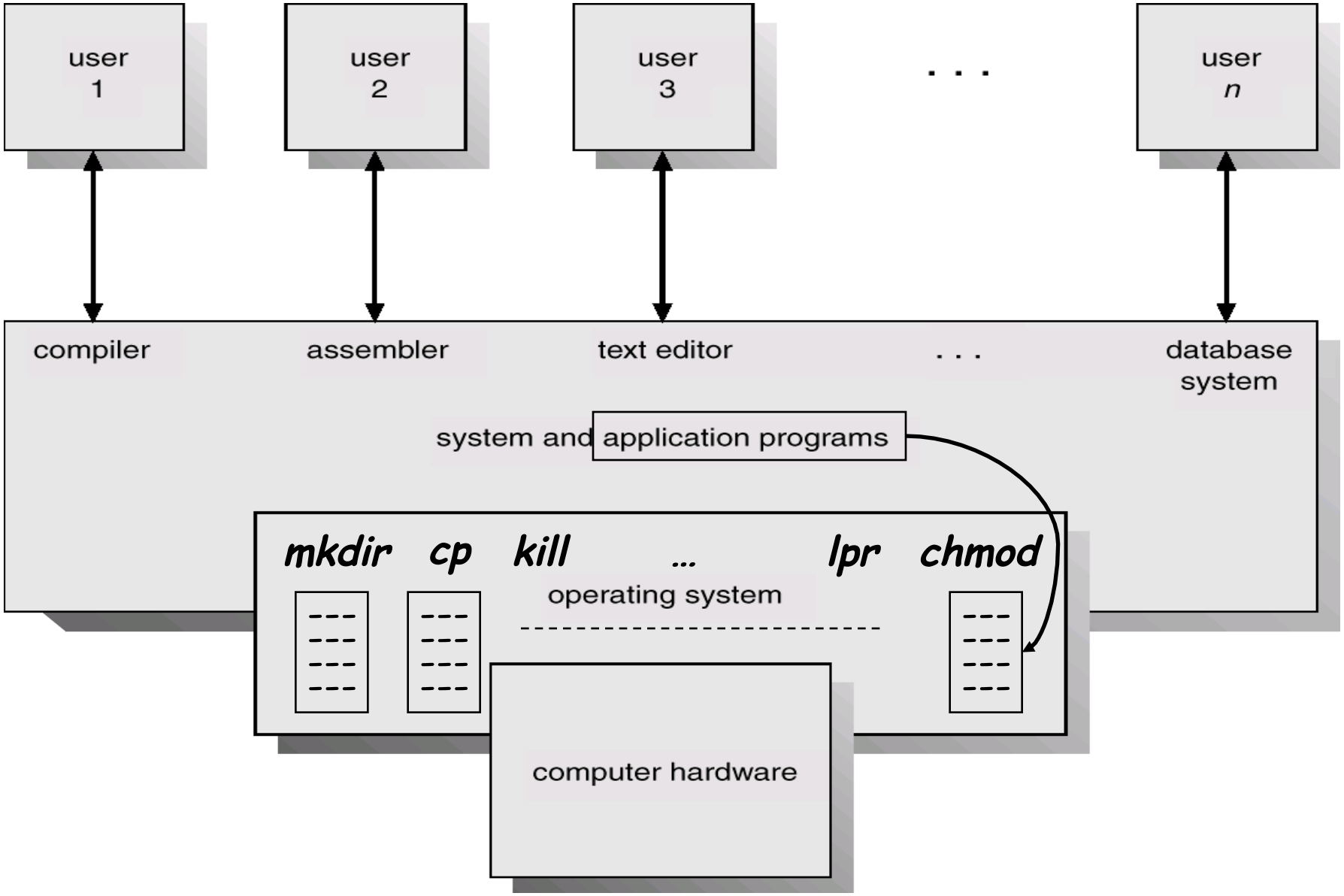
System Calls

Forniscono ***l'interfaccia*** tra un ***programma in esecuzione*** ***e OS***



La maggior parte dei moderni linguaggi di programmazione permette la ***chiamata diretta di system call*** nel linguaggio stesso

Ad ogni system call ***corrisponde*** in genere ***un comando di OS***, che quindi e' disponibile anche direttamente all'utente in ***shell***


user 1
user 2
user 3
. . .
user n
compiler
assembler
text editor
. . .
database system
system and application programs
mkdir
cp
kill
...
lpr
chmod
operating system
computer hardware

Costruite un programma (in Java/C/C++/…) che legga il contenuto di un file di testo e scriva tale contenuto in un altro file

Individuate, all'interno del programma realizzato, quante e quali sono le system call necessarie

Eseguite, poi, quest'operazione direttamente mediante comandi di OS, ed elencate quali comandi avete dovuto eseguire per completare l'operazione

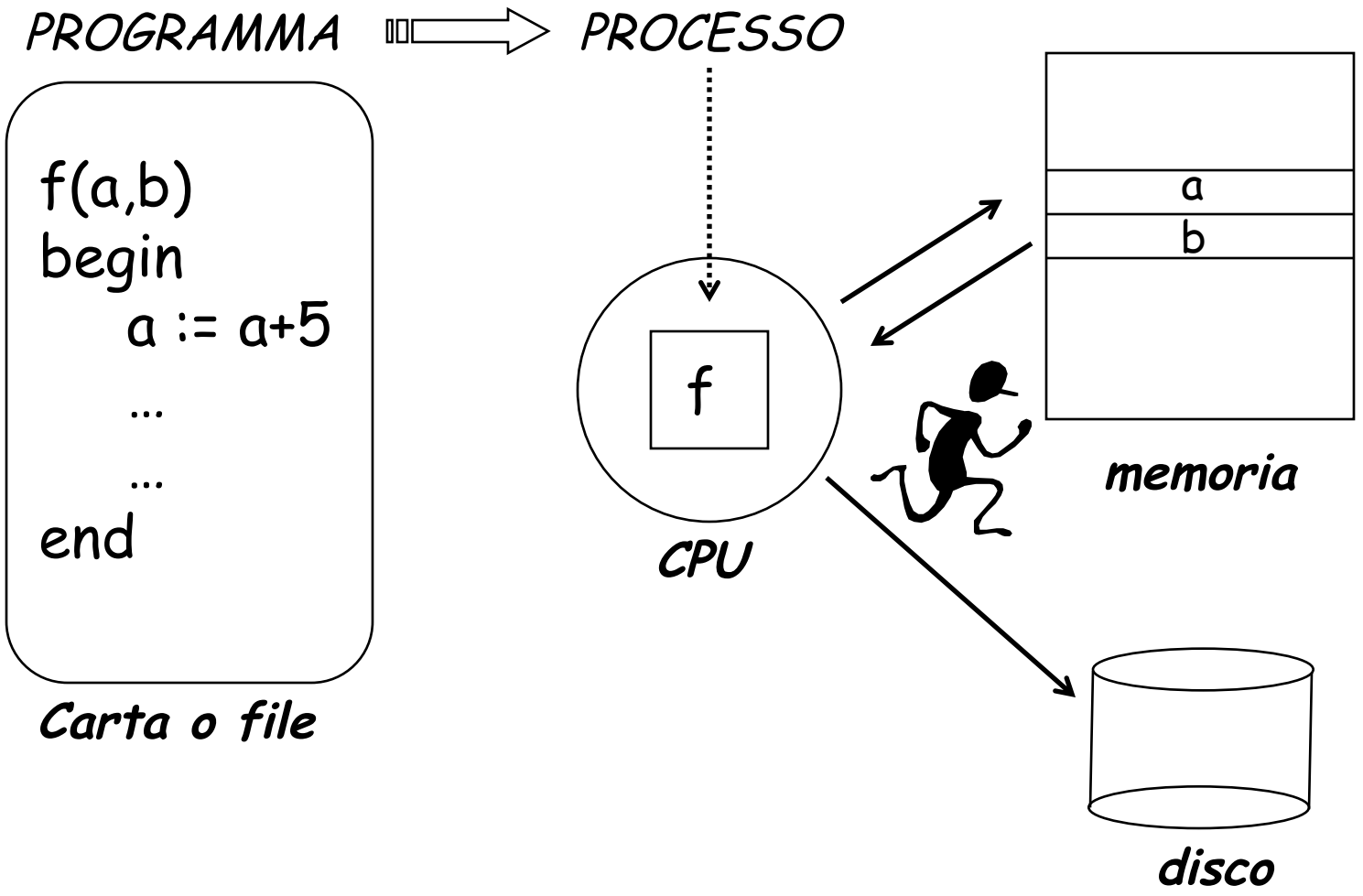
1. Gestione dei processi

Un ***processo*** e' un programma in esecuzione

Un ***processo utente*** deriva da un programma eseguito da un utente; un ***processo di sistema*** corrisponde ad una routine di OS

Un processo ha bisogno di ***risorse***, quali tempo di CPU, memoria, files e dispositivi di I/O per svolgere il suo compito

Programma e processo



Compiti di OS rispetto ai processi

Creazione e cancellazione

Sospensione e ripristino (scheduling)

Sincronizzazione (deadlock)

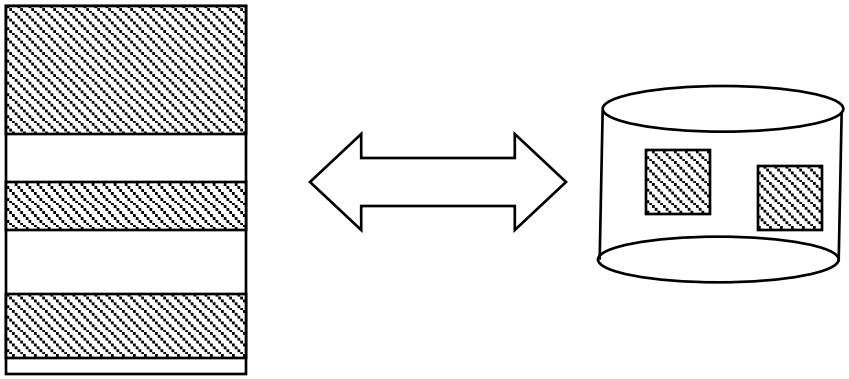
Comunicazione

2. Gestione della memoria centrale

Tenere traccia di quali parti della memoria sono correntemente usate e da chi

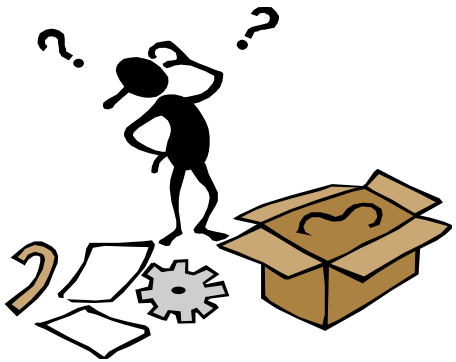
Decidere quali processi caricare da disco quando diventa disponibile spazio di memoria

Allocare e deallocare spazio di memoria quando necessario



3. Gestione della memoria secondaria

La ***memoria secondaria*** serve da ***backup per la memoria centrale*** (troppo piccola e volatile per conservare informazioni in maniera permanente)



In genere si tratta di ***dischi***, aventi caratteristiche proprie quali la velocita' di accesso, la capienza, etc…

Compiti di OS rispetto alla memoria secondaria

Gestione dello spazio libero

Allocazione della memoria

Scheduling delle richieste

4. Manipolazione dei file

Un ***file*** e' una ***collezione di informazioni*** (in qualche modo ***in relazione*** tra esse) definita dal suo creatore

Un file comunemente contiene o ***programmi*** o ***dati*** o ***altri file*** (directory)

Il file e' lo ***strumento di astrazione*** dalle memorie secondarie

Compiti di OS rispetto ai file

Creazione e cancellazione di file/directory

Primitive per la manipolazione (open, read, etc.)

Mapping di file a dispositivi di memoria

Backup di files su dispositivi appositi

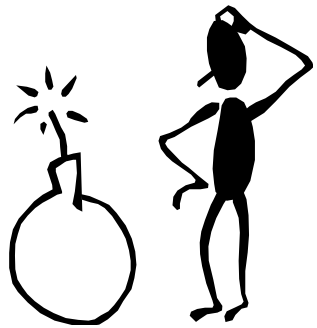
5. Gestione dell'I/O

- Sistema di buffering, caching e spooling
  - ***Buffering*** : preparare i dati di I/O in predisposti registri e aree di memoria
  - ***Caching*** : conservare in memorie veloci i dati usati piu' di frequente per I/O (ex. indirizzi per I/O mapped)
  - ***Spooling*** : virtualizzazione dei dispositivi di I/O mediante uso di aree di memoria per caricare e scaricare informazioni
- ***Driver*** per ogni tipo di dispositivo : routine di I/O che colloquia con il controller del dispositivo
- ***Interfaccia generale per i driver*** : programma che si invoca per iniziare qualsiasi operazione di I/O

6. Sistema di protezione

Una ***risorsa*** e' ***un'entita' passiva***, quindi non e' in grado di proteggersi da sola contro usi impropri ed errori

Piu' in generale un ***sistema di protezione*** si occupa di ***controllare gli accessi*** di programmi, processi, utenti a OS e risorse



Compiti di OS per la protezione

Distinguere tra ***usi/utenti autorizzati*** e non

Specificare i ***controlli*** da imporre

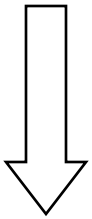
Fornire un meccanismo per il ***rispetto dei vincoli*** imposti dalle protezioni

7. Gestione di rete

Un ***sistema distribuito*** e' una collezione di "elaboratori" (possibilmente ***non omogenei*** tra loro) che ***non condividono memoria e clock***, e che sono collegati mediante una ***rete di comunicazione***

Compiti di OS per una rete

Fornire accesso alle risorse di rete
da parte degli utenti

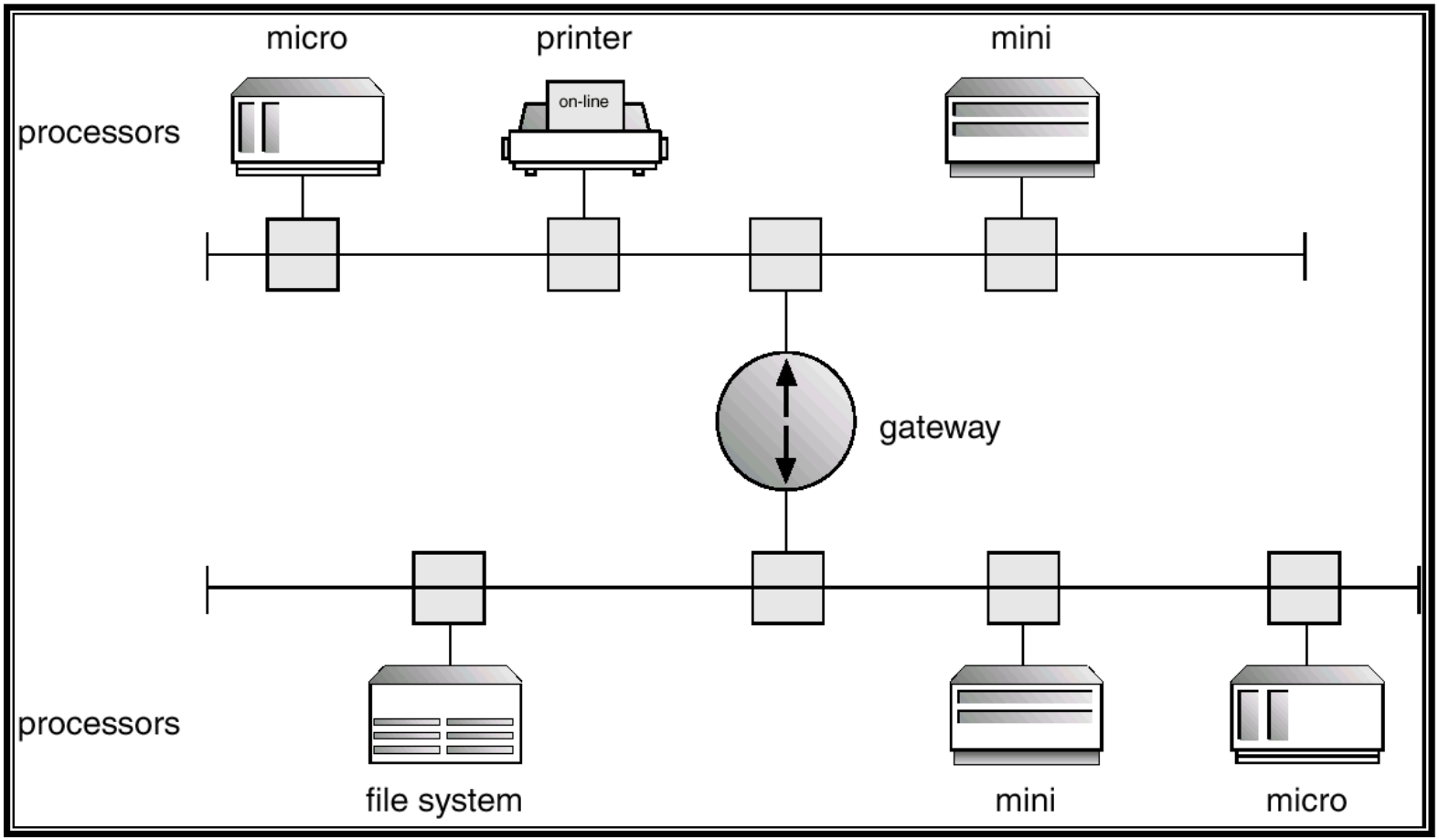


Velocizzazione dell' esecuzione

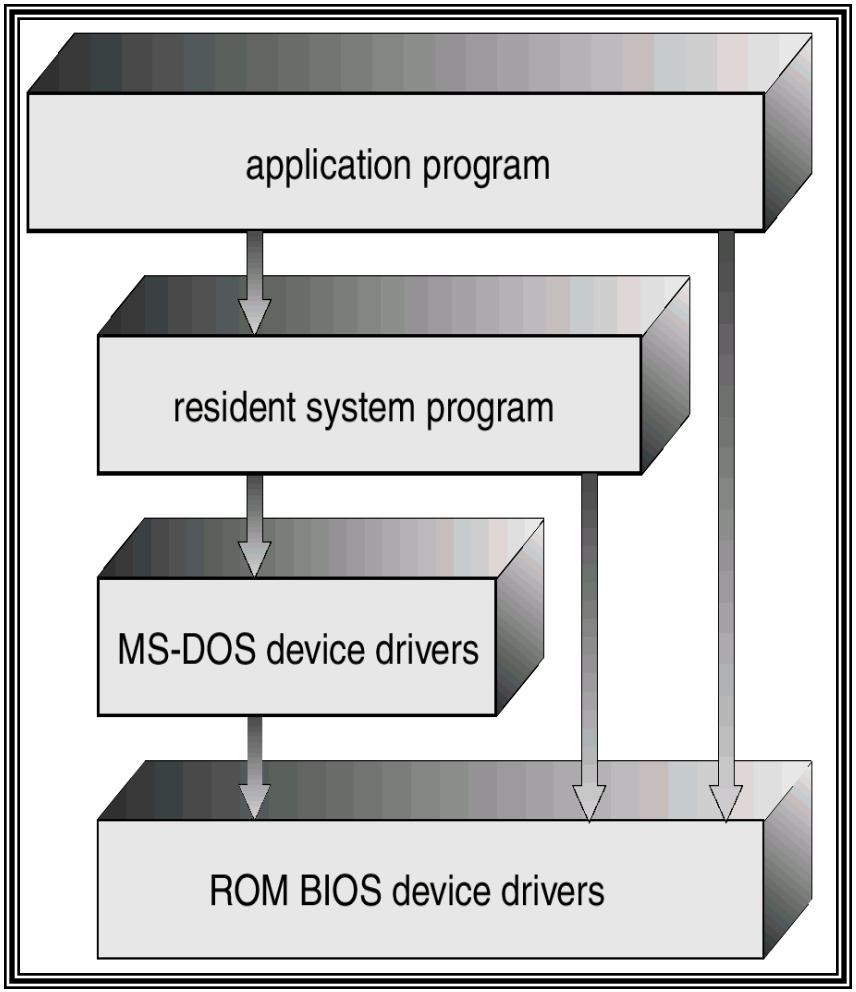
Maggiore ***disponibilita'*** di informazioni

Maggiore ***affidabilita'***

Un esempio di Local Area Network



Struttura interna di un OS
Un esempio di design : MS-DOS



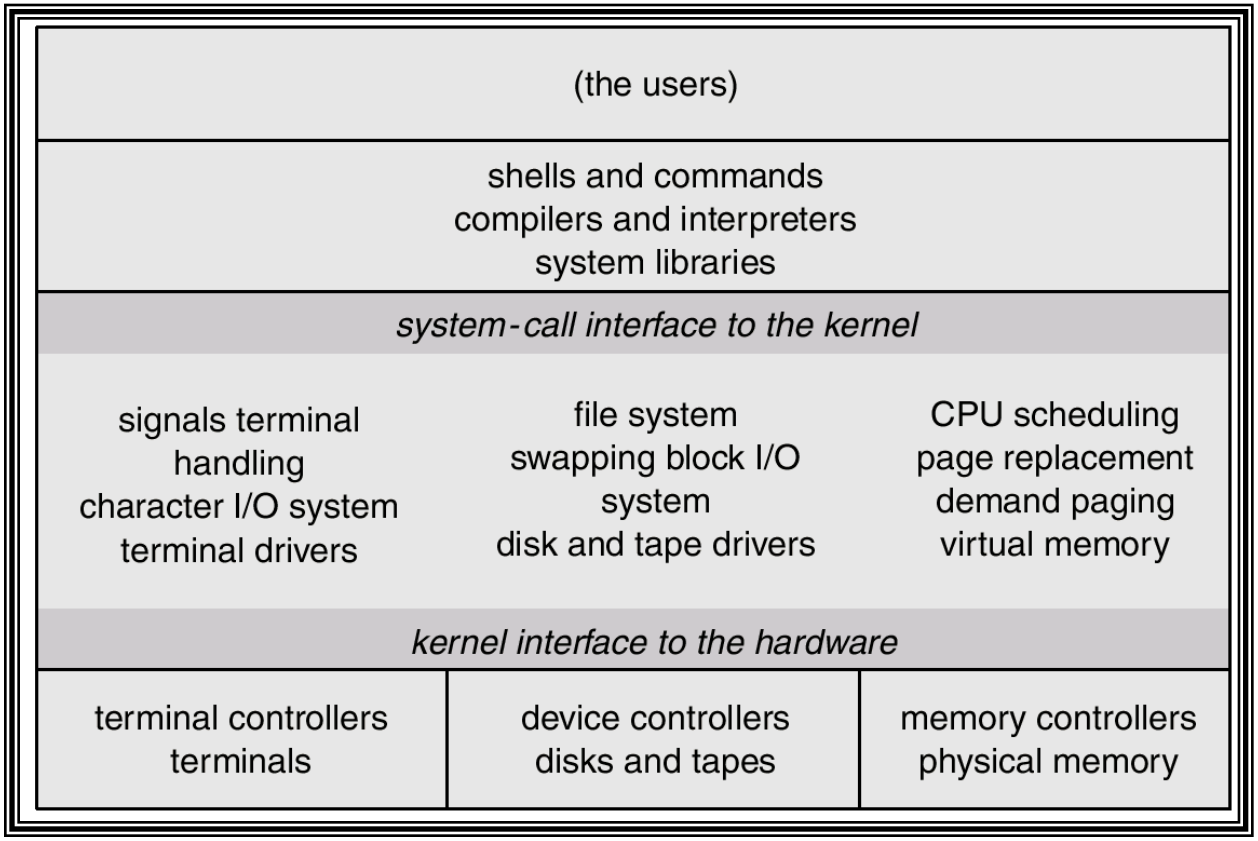
Interfacce e livelli di funzionalita' non sono ben separati

Un criterio generale di design: approccio stratificato

- OS e' suddiviso in un numero di livelli, ognuno ***costruito sui sottostanti***

- Il livello piu' basso e' ***l'hardware*** (livello 0) e il piu' alto e' ***l'interfaccia utente*** (livello N)

… ed un esempio: UNIX

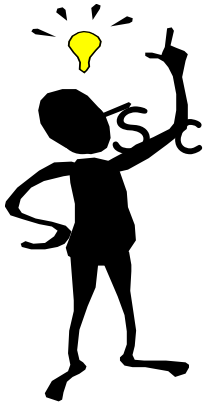


kernel e' una parola usata (impropriamente) in questo caso per indicare tutto il Sistema Operativo

Da stratificazione a MACCHINA VIRTUALE

Usualmente si definisce macchina virtuale l' insieme costituito dall' ***hardware*** e dal ***kernel di OS***

Ogni utente ha una ***copia identica dell' immagine della macchina*** che sta utilizzando



E' il punto di arrivo della stratificazione:
si crea ***l' illusione*** ad ogni utente che sta utilizzando una ***macchina dedicata***

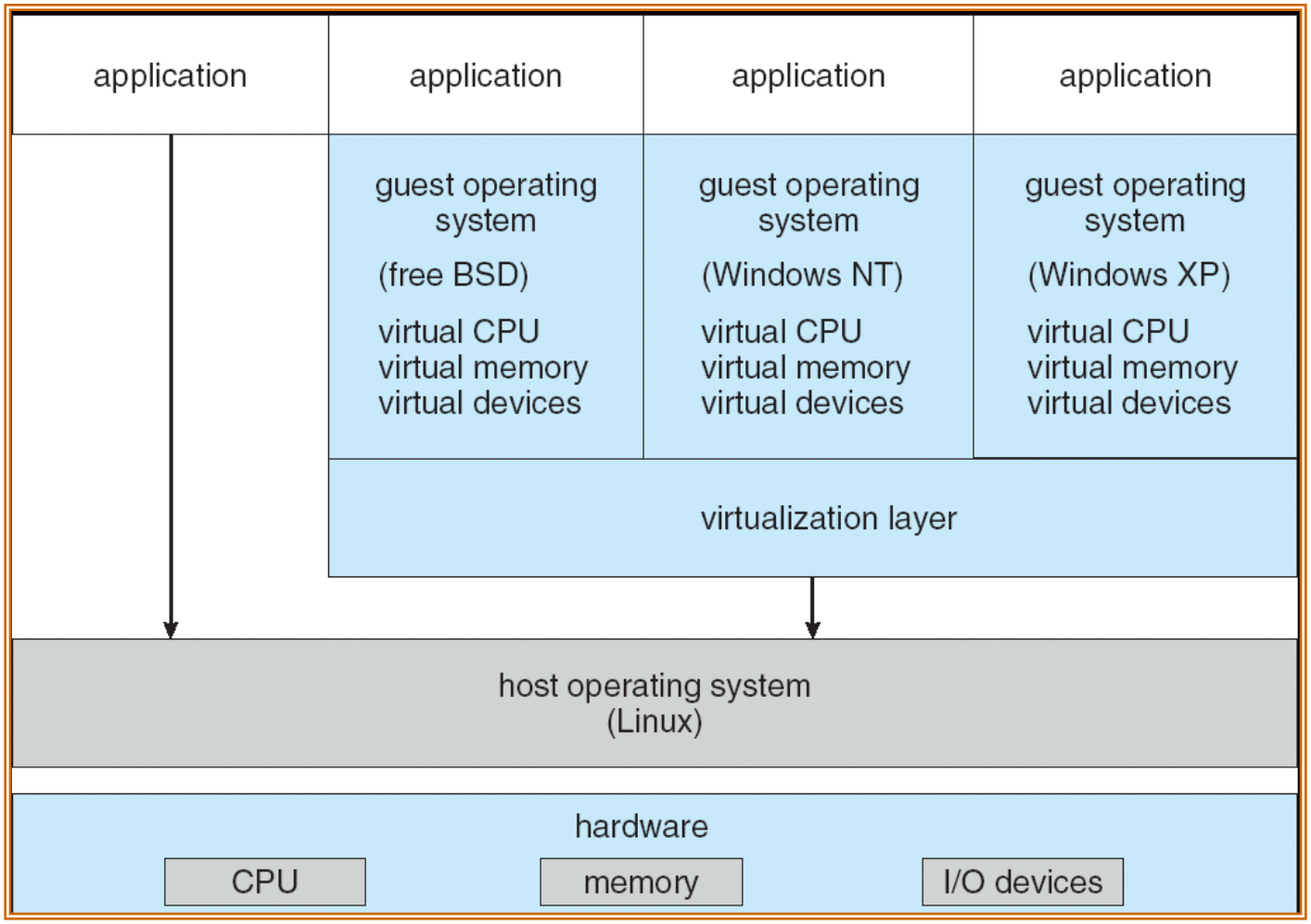
Come si virtualizza…

Le risorse della macchina fisica vengono condivise e gestite in modo da creare **risorse o intere macchine** virtuali

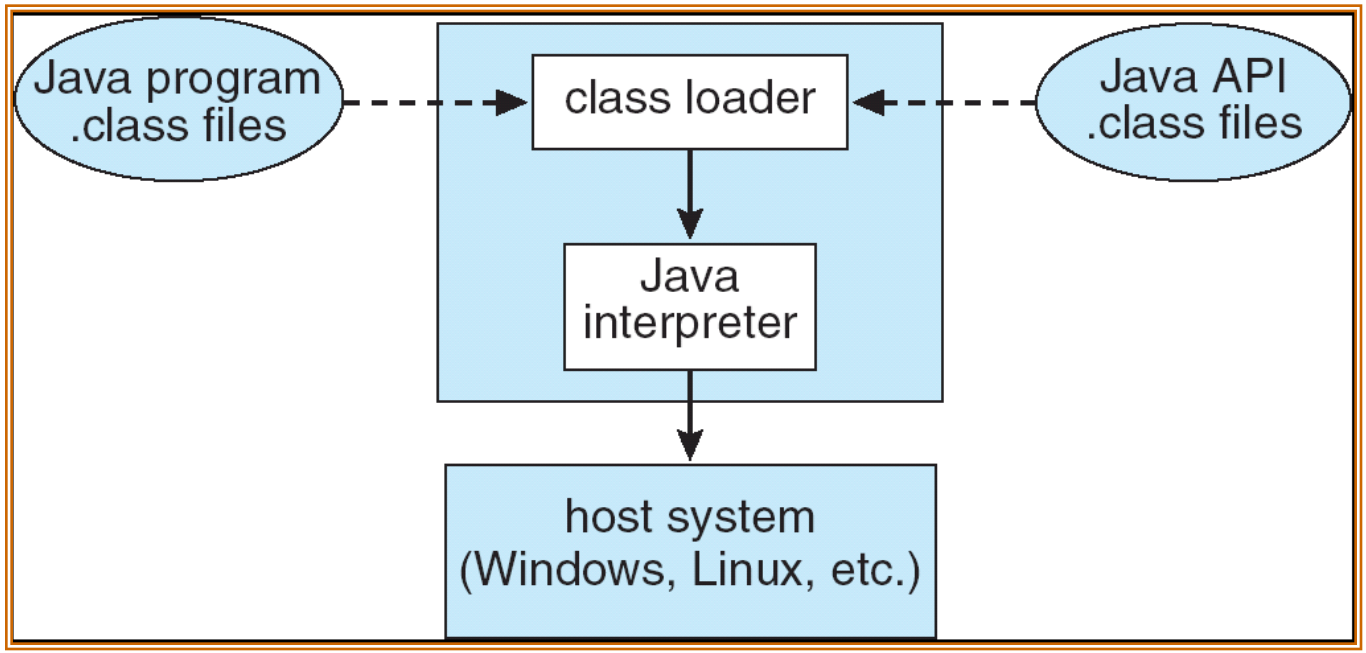
ESEMPI

- Il time-sharing della CPU crea l' illusione che ogni utente abbia la sua propria CPU
- Una tecnica di spooling ed un file system permettono di virtualizzare una stampante
- Le comuni partizioni non sono altro che una virtualizzazione dei dischi

VMware Architecture



The Java Virtual Machine



Scelte di progetto

Meccanismo

Strumento atto a svolgere un certo compito

Politica

Modo di utilizzo dello strumento

La ***separazione tra meccanismo e politica*** permette flessibilita' :
politiche potranno essere modificate senza modificare i meccanismi

ESEMPIO

Meccanismo

Strutture dati per la gestione di piu' processi pronti ad essere eseguiti

Politica

Strategia che decide in ogni momento quale processo deve essere eseguito

Installazione e partenza

- OS deve essere ***configurato*** per ogni specifica piattaforma di installazione
- Input della configurazione da adottare
- ***Due alternative*** : ricompilazione del kernel o tabelle di selezione di moduli

- ***Booting*** – far partire un computer caricando in memoria il kernel di OS
- ***Bootstrap program*** – codice residente in ROM capace di localizzare il kernel, caricarlo in memoria e avviarne l'esecuzione