

Sistemi Operativi

Ricapitolazione di Concetti di Base

Docente: Claudio E. Palazzi

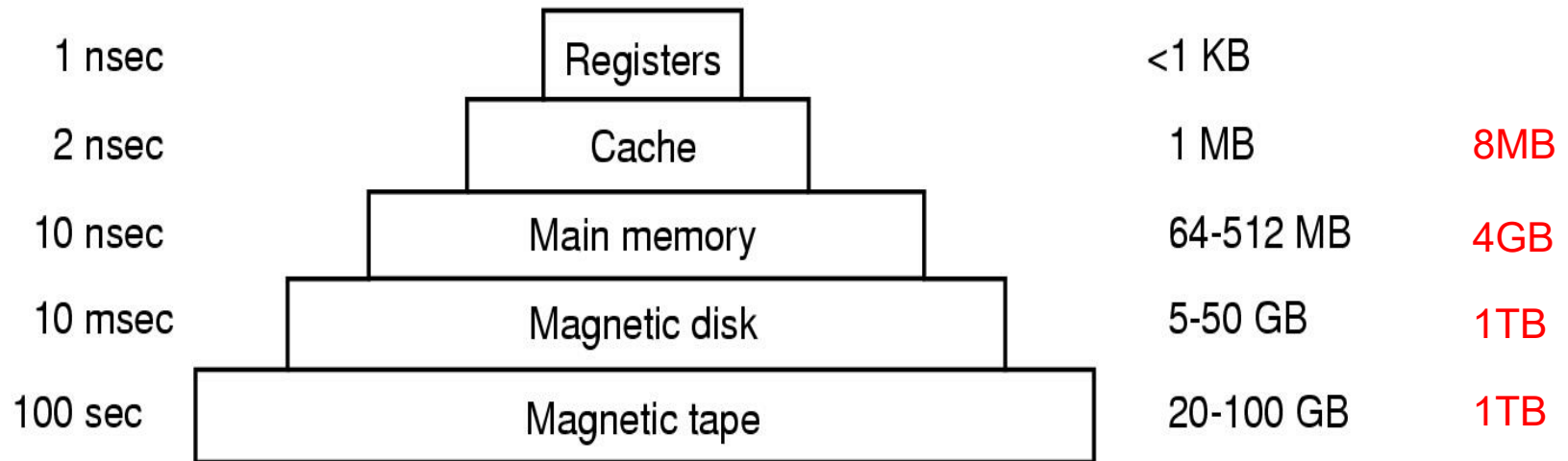
cpalazzi@math.unipd.it

Gerarchia fisica di memoria – 1

Tempo di accesso
Typical access time

Capacità tipica
Typical capacity

Ora circa...

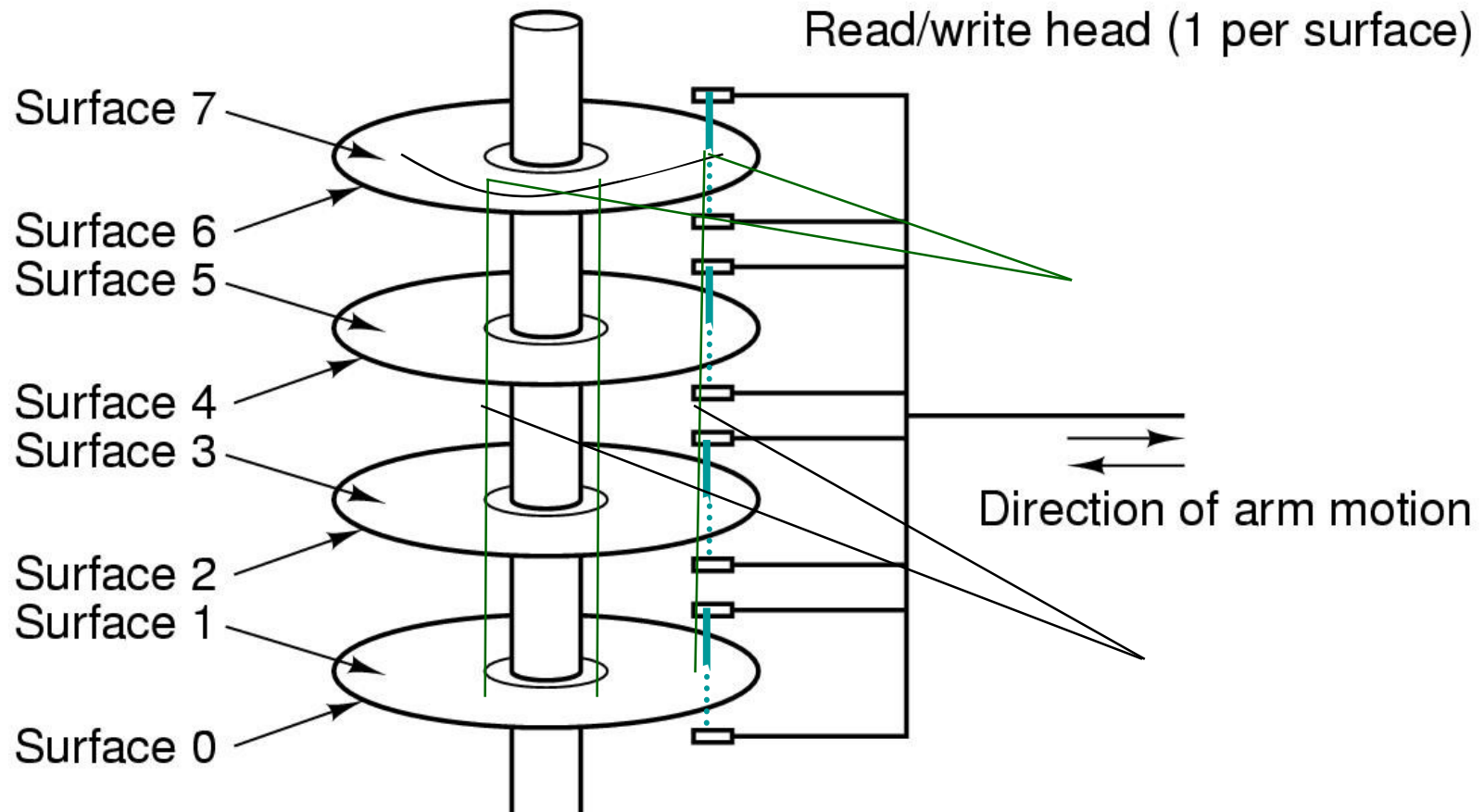


E i solid state disk?

Gerarchia fisica di memoria – 2

- I registri sono interni alla CPU; dimensione:
 - 32 bit su processori a 32-bit
 - 64 bit su processori a 64-bit
- La *cache* è controllata da hw ed è suddivisa in blocchi chiamati *line* con ampiezza tipica 64 B
 - L1 dentro la CPU, L2 adiacente alla CPU
 - L2 condivisa (Intel) o propria di ciascun core (AMD)?
 - *Hit* (2 cicli di clock), *miss* (memoria)
 - *Write through*, *copy back*
- I dischi magnetici hanno capienza 100 volte superiore e costo/*bit* 100 volte inferiore rispetto alla RAM
 - Ma per tempo di accesso 1000 volte peggiore

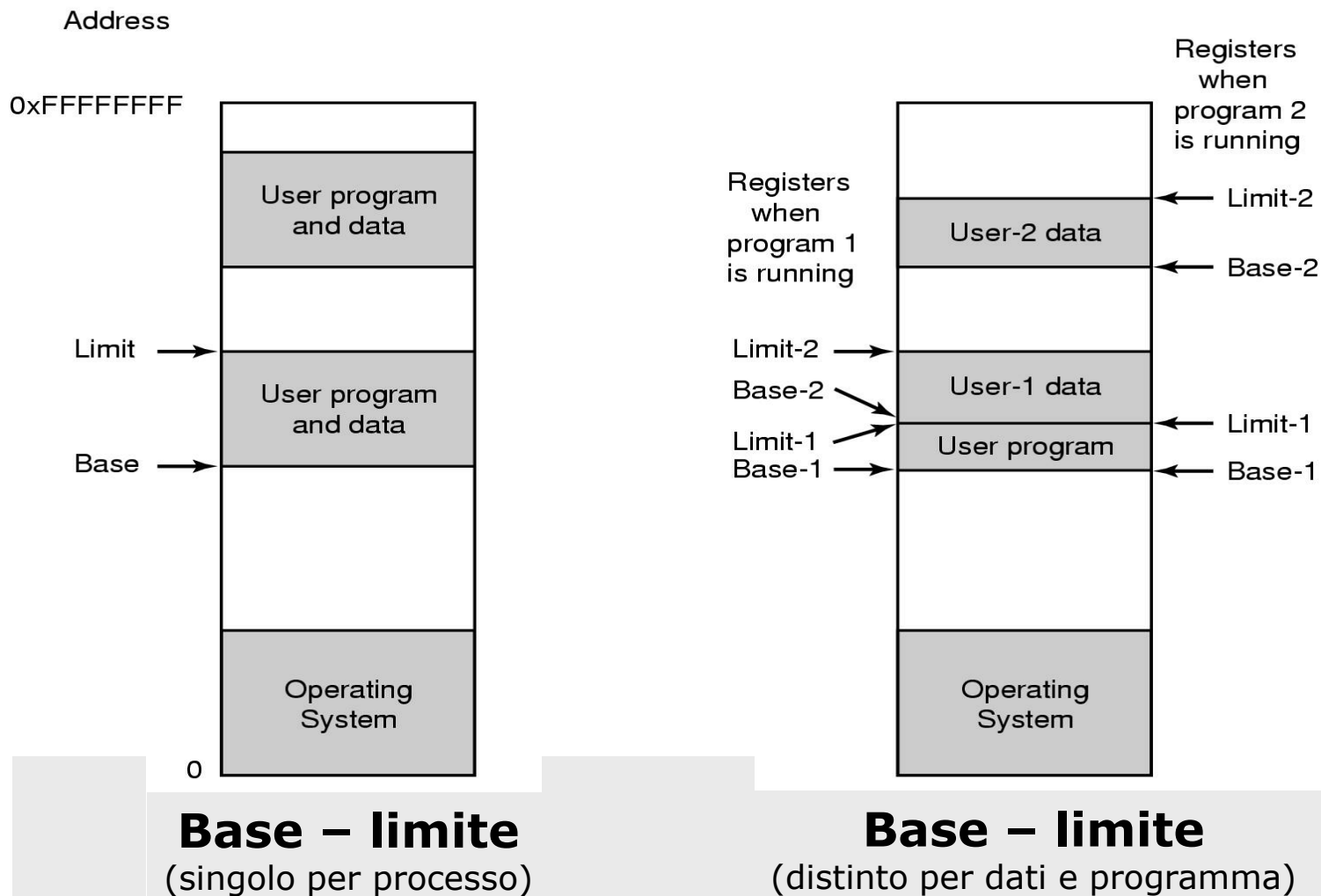
Gerarchia fisica di memoria – 3



Programmi e Memoria

- CMOS è una memoria volatile ma alimentata da una piccola batteria
 - Memorizza ora e da quale disco fare boot
 - Memorizza impostazioni BIOS diverse da default
- Interazione memoria-programmi:
 - Come proteggere i programmi tra loro e il kernel dai programmi
 - Come gestire la rilocalizzazione
 - Quando si compila un programma non si sa in che area della memoria verrà caricato (indirizzi da 0 a ...)
 - Soluzione hardware, due registri (base e limite)
 - Verifica e somma base+indirizzo costa qualche ciclo di CPU

Vista logica della RAM – 1

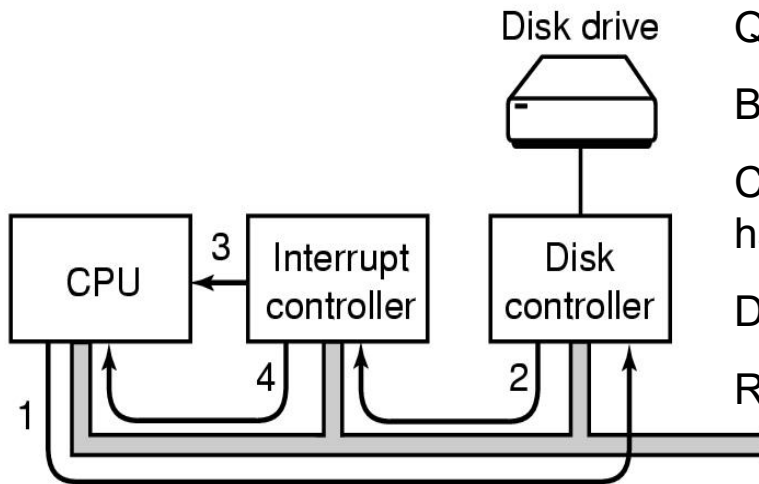


Vista logica della RAM – 2

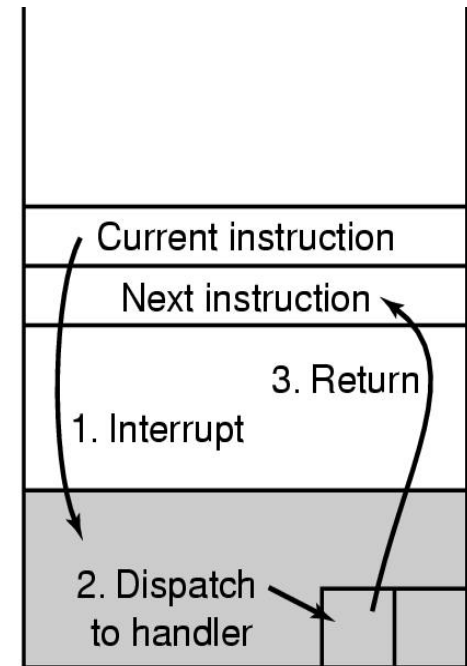
- Nella sua forma più rudimentale la ripartizione della RAM tra processi distinti utilizza 2 registri speciali
 - **Base e limite**, i cui valori formano parte importante del contesto del processo
 - L'allocazione del processo in RAM richiede **rilocazione** della sua **memoria virtuale**
- In generale la gestione dello spazio di memoria virtuale dei processi utilizza un dispositivo di **MMU** (Memory Management Unit) logicamente interposto tra CPU e memoria
 - Sotto la responsabilità del Sistema Operativo
- Attenzione: a ogni context switch la cache è piena di dati del processo precedente

Trattamento delle interruzioni – 1

1. Driver dice a controller cosa fare
2. Segnale “finito” su certe linee bus
3. Imposta pin in CPU
4. Mette nome dispositivo su bus



Quale settore?
Blocchi rovinati?
Cilindri esterni
hanno più settori
Driver?
Registri?



Interrupt vector ha indirizzo di Interrupt handler

Attivazione di un dispositivo di I/O Gestione delle interruzioni

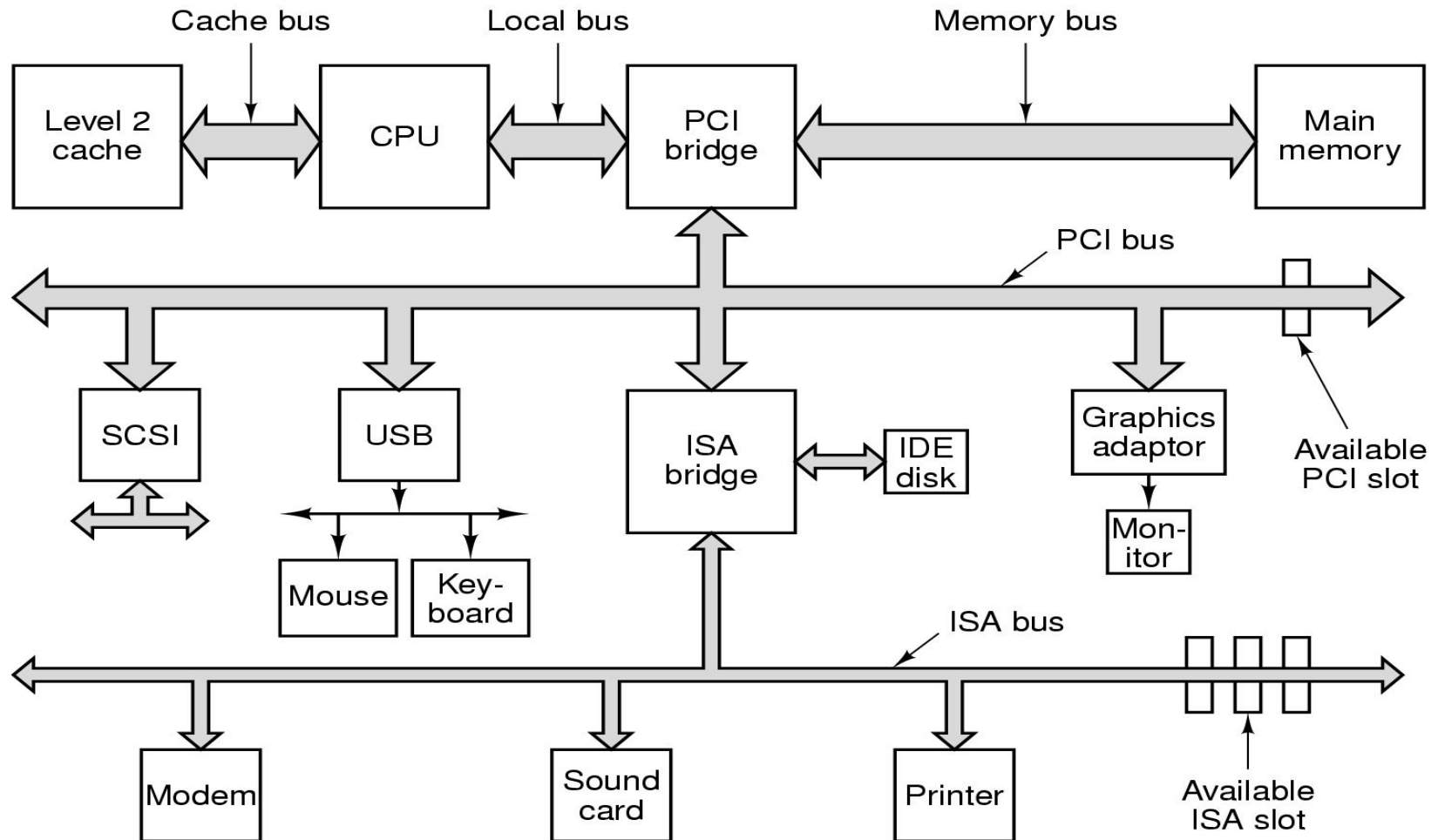
Trattamento delle interruzioni – 2

- L'uso delle interruzioni per l'interazione con i dispositivi evita il ricorso al *polling*
- L'interazione tipica avviene in 4 passi successivi come illustrato in figura
 1. Il gestore del dispositivo programma il controllore di dispositivo scrivendo nei suoi registri di interfaccia
 2. Il controllore agisce sul dispositivo e poi informa il controllore delle interruzioni
 3. Il controllore delle interruzioni asserisce un valore (*pin*) di notifica verso la CPU
 4. Quando la CPU si dispone a ricevere la notifica il controllore delle interruzioni comunica anche l'identità del dispositivo
 - Così che il trattamento dell'interruzione sia attribuito al gestore appropriato

Trattamento delle interruzioni – 3

- All'arrivo di una interruzione
 - I registri PC (Program Counter) e PSW (Program Status Word) sono posti sullo *stack* del processo corrente
 - La CPU passa al “modo operativo protetto”
 - Il parametro principale che denota l'interruzione serve come indice nel **vettore delle interruzioni**
 - Così si individua il gestore designato a servire l'interruzione
 - La parte **immediata** del gestore esegue nel contesto del processo interrotto
 - La parte del servizio meno urgente può essere invece **differita** e demandata a un processo dedicato

Pentium: architettura fisica – 1

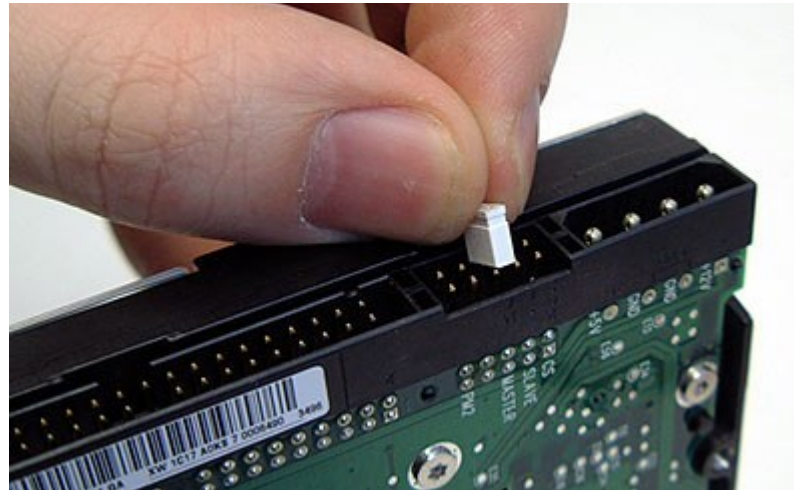


Pentium: architettura fisica – 2

- **ISA bus** (Industry Standard Architecture, 8.33 MHz, 2B/ciclo, 16.67 MB/s)
 - Non più usato
- **SATA**: I(1,5 Gbit/s), II(3 Gbit/s), III(6 Gbit/s)
- **PCI bus** (*Peripheral Component Interconnect*, 66 MHz, 8 B/ciclo, 528 MB/s)
 - Di vecchia concezione ma dotato di connettori per una grande varietà di dispositivi
 - PCI Express: 1.x 4GBB/sec, 2.x 8GB/sec, 3.x 16GB/sec
- **Memory bus** (100 Mhz, dedicated)
 - 1333Mhz è attualmente lo standard per le DDR3
- **USB bus** (*Universal Serial Bus*)
 - USB 1.0: 1,5 Mbit/s
 - USB 1.1: 12 Mbit/s
 - USB 2.0: 480 Mbit/s
 - USB 3.0: 4,8 Gbit/s
 - Per l'interconnessione di dispositivi lenti (tastiera, mouse), con 4 linee delle quali 2 di alimentazione del dispositivo
 - Unico device driver
 - *Bus* con singolo *master* centrale predefinito che interroga @ 1 ms i dispositivi collegati (collegabili)
- **SCSI bus** (*Small Computer System Interface*, ≤ 160 MB/s)
 - Per l'interconnessione di dispositivi veloci (dischi veloci, un tempo anche scanner)

Plug & Play (Pray?)

- Intel e Microsoft
- Prima ogni scheda I/O aveva un livello di interrupt fisso e un indirizzo fisso per i registri
 - Se si acquistavano due dispositivi con lo stesso valore di interrupt?
- Con Plug & Play, il sistema assegna centralmente i livelli di interrupt e gli indirizzi di I/O e poi li rivela alle schede



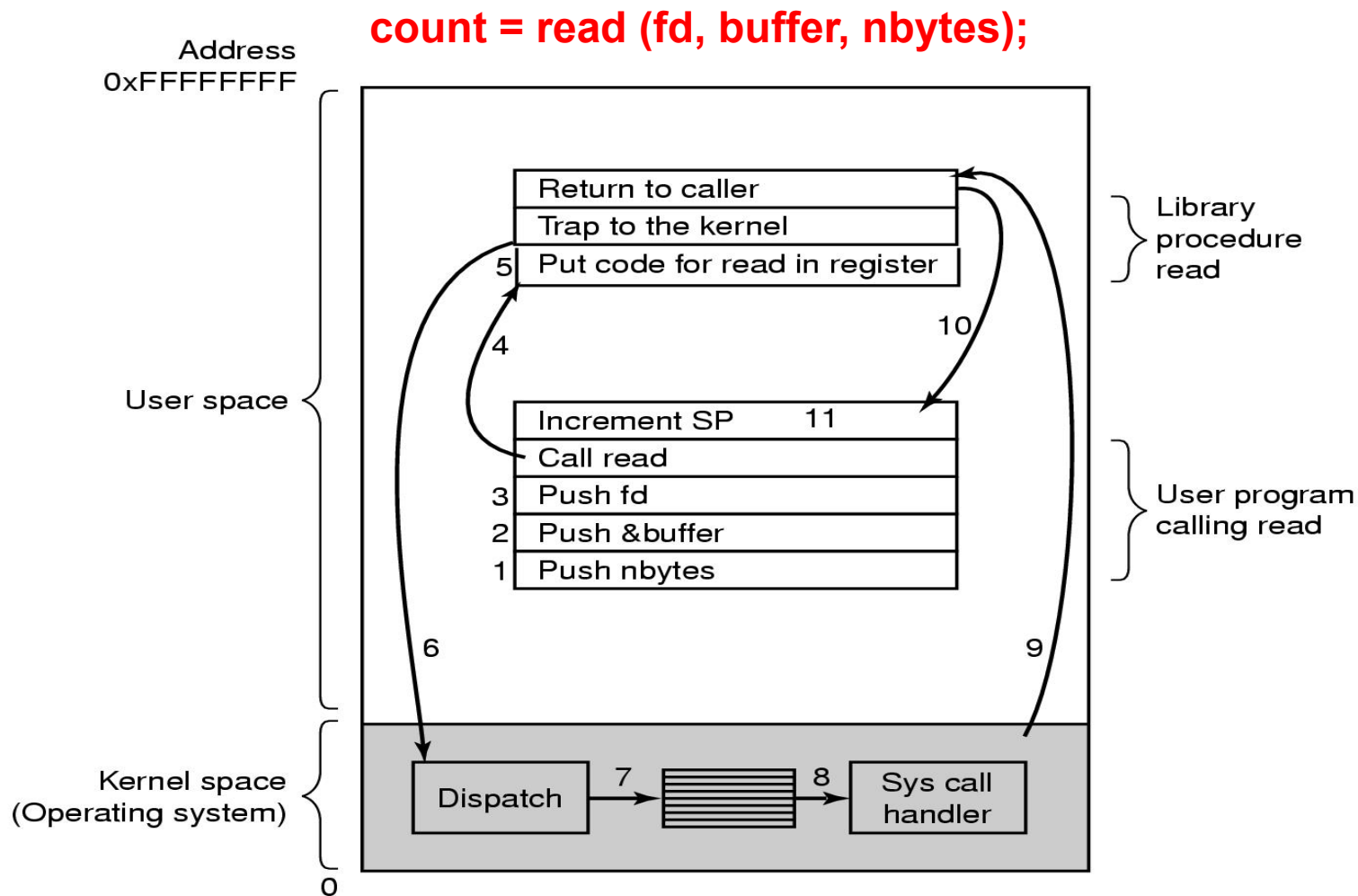
BIOS

- BIOS (Basic Input Output System)
 - Contiene sw a basso livello per la gestione di I/O
 - Viene caricato all'avvio del computer
 - Verifica quanta RAM e quali dispositivi base (tastiera,ecc) sono presenti
 - Fa scan dei bus ISA e PCI per rilevare dispositivi ad essi connessi
 - I dispositivi vecchi (prima di plug & play, detti legacy) sono rilevati e registrati
 - Vengono registrati anche i dispositivi plug & play
 - Se ci sono nuovi dispositivi dall'ultimo avvio, questi vengono configurati (assegnati livelli di interrupt e indirizzi I/O)
 - Determina il dispositivo di boot dalla lista in memoria CMOS

BOOT

- Il primo settore del dispositivo di boot viene letto in memoria ed eseguito
 - Contiene un programma che esamina la tabella di partizione e determina quale partizione sia attiva
 - Da tale tabella, viene caricato un secondo boot loader
 - Legge il sistema operativo dalla partizione attiva e lo esegue
 - Il sistema operativo interroga il BIOS per ottenere informazioni sulla configurazione del sistema
 - Per ogni dispositivo controlla l'esistenza del driver
 - Se non c'è chiede di inserire CD o Floppy
 - Se ci sono li carica nel kernel
 - Poi, esegue varie inizializzazioni ed esegue programma iniziale (login, GUI)

Chiamate di sistema – 1



(SP: Stack Pointer)

Chiamate di sistema – 2

- La maggior parte dei servizi del Sistema Operativo sono eseguiti in risposta a invocazioni esplicite di processi
 - Chiamata di sistema
- Le chiamate di sistema sono nascoste in procedure di libreria predefinite
 - L'applicazione **non** effettua direttamente chiamate di sistema
 - La procedura di libreria svolge il lavoro di preparazione necessario ad assicurare la corretta invocazione della chiamata di sistema
- La prima istruzione di una chiamata di sistema (*trap*) deve attivare il modo operativo privilegiato
 - Inizia esecuzione ad un indirizzo prefissato del kernel
 - Il parametro della chiamata designa l'azione da svolgere e la convenzione per trovare gli altri eventuali parametri
 - Il meccanismo complessivo è simile a quello già visto per il trattamento delle interruzioni
 - Le interruzioni sono **asincrone**
 - Le chiamate di sistema invece sono **sincrone**

Chiamate di sistema – 3

1. Il programma applicativo effettua una chiamata di sistema
 - (2.-3.) Prima pone sullo *stack* i parametri secondo una antica convenzione C/UNIX
4. Poi invoca la procedura di libreria corrispondente alla chiamata
5. Questa pone l'ID della chiamata in un luogo noto al S/O
6. Poi esegue l'istruzione *trap* per passare all'esecuzione in modo operativo privilegiato
7. Il S/O individua la chiamata da eseguire
8. La esegue
9. Poi ritorna al chiamante oppure a un nuovo processo
10. Ritorna come farebbe da return di procedura
11. Cancella dati nello stack facendo avanzare il puntatore

Alcune Chiamate di Sistema (POSIX)

Process management

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

File management

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

Directory and file system management

Call	Description
<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system

Miscellaneous

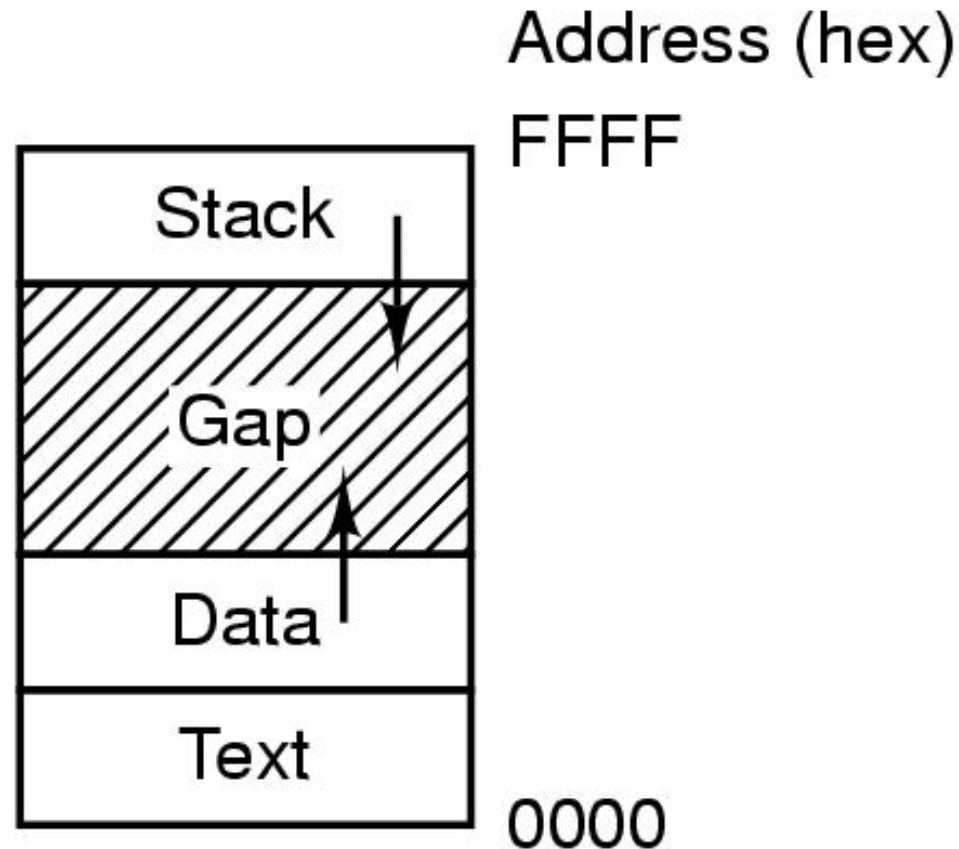
Call	Description
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>s = kill(pid, signal)</code>	Send a signal to a process
<code>seconds = time(&seconds)</code>	Get the elapsed time since Jan. 1, 1970

System Calls (1)

- Una shell base tramite fork (UNIX):
- ```
while (TRUE) { /* repeat forever
*/
type_prompt(); /* display prompt */
read_command (command, parameters) /* input from terminal */

if (fork() != 0) { /* fork off child process */
 /* Parent code */
 waitpid(-1, &status, 0); /* wait for child to exit */
} else {
 /* Child code */
 execve (command, parameters, 0); /* execute command */
}
}
```

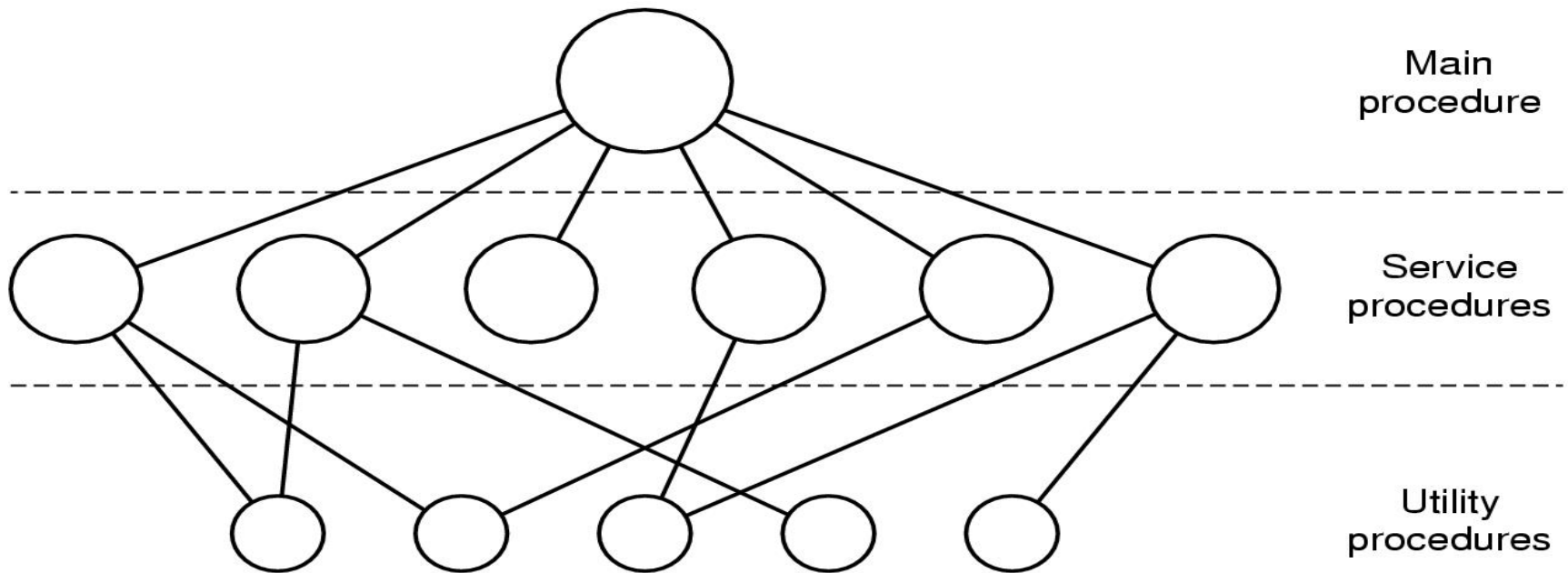
# System Calls (2)



- Processi in memoria hanno 3 segmenti: testo, dati, stack

# Architettura logica di S/O – 1

## Struttura monolitica



# Architettura logica di S/O – 2

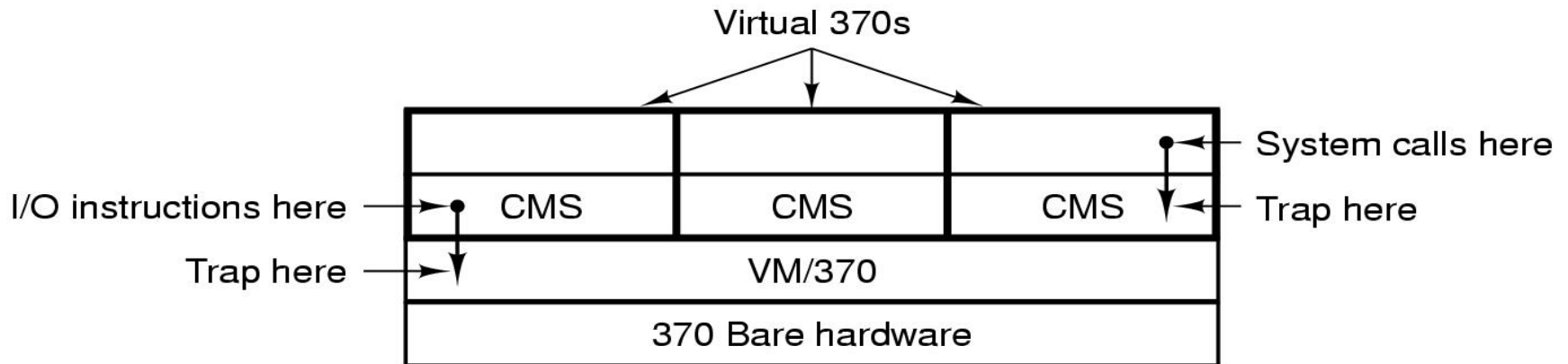
- Un'architettura monolitica **non** ha struttura
  - Il S/O è una collezione “piatta” di procedure
    - Ognuna delle quali può chiamarne qualunque altra
    - Nessuna forma di *information hiding*
  - Il S/O è un singolo .o che collega tutte le procedure che lo compongono
- L'unica struttura riconoscibile in essa è data dalla convenzione di attivazione delle chiamate di sistema
  - Parametri messi in un posto preciso (stack) e poi esegue trap

# Architettura logica di S/O – 2 bis

- Organizzazione di base:
  1. Programma principale che invoca le procedure di servizio richieste;
  2. Insieme di procedure che eseguono le system call;
  3. Insieme di procedure di utilità che sono di ausilio per le procedure di servizio.
- Generalizzazione è la struttura a strati (layers)
  - THE system (Dijkstra '68)



# Architettura logica di S/O – 3



## Sistema a macchina virtuale

# Architettura logica di S/O – 4

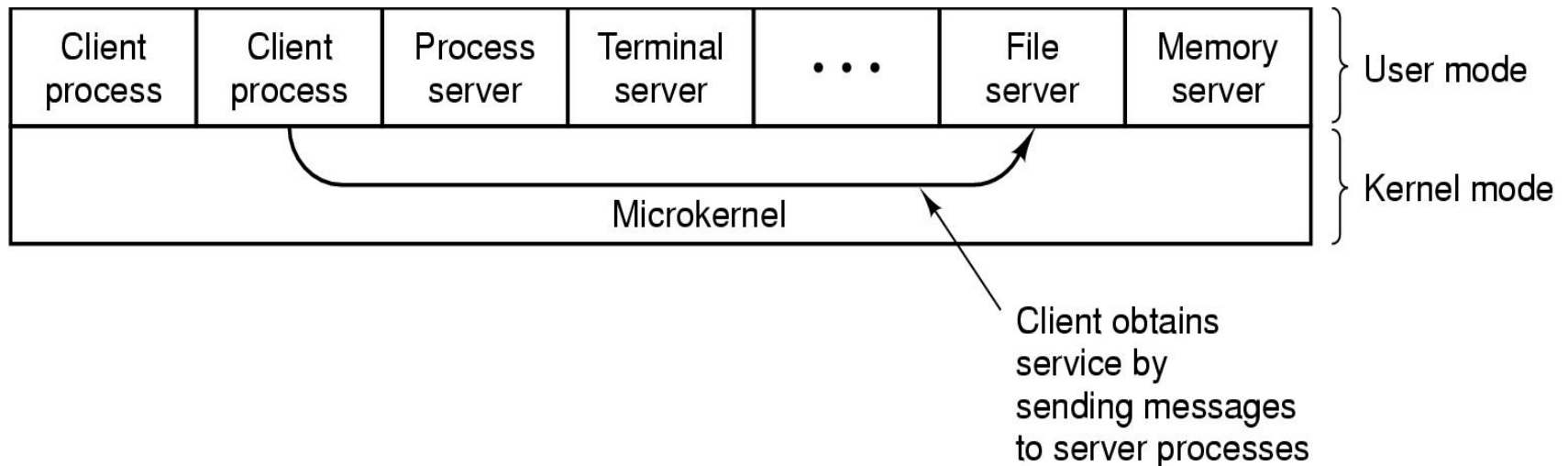
- Primi anni '70
  - **VM/370** sviluppato da IBM *Scientific Center*, Cambridge, Massachusetts per fornire time sharing su sistemi batch dell'azienda
  - Basato sull'intuizione che un S/O a divisione di tempo in realtà realizza 2 fondamentali funzioni
    1. Multiprogrammazione
    2. Virtualizzazione dell'elaboratore fisico
  - Separandole e ponendo 2. alla base si possono offrire copie identiche di “**macchine virtuali**” (copie dell'hardware) a S/O distinti che realizzano 1. secondo un criterio loro proprio

# Architettura logica di S/O – 5

- **CMS** (*Conversational Monitor System*)
  - S/O interattivo a divisione di tempo mono-utente
  - Esegue sopra una macchina virtuale realizzata da VM/370
- L'idea della virtualizzazione di elaboratori logici o fisici ha avuto notevole seguito
  - Intel: modo 8086 virtuale su Pentium
  - MS Windows & co.: ambiente virtuale di esecuzione MS-DOS
  - JVM: architettura portabile di elaboratore logico

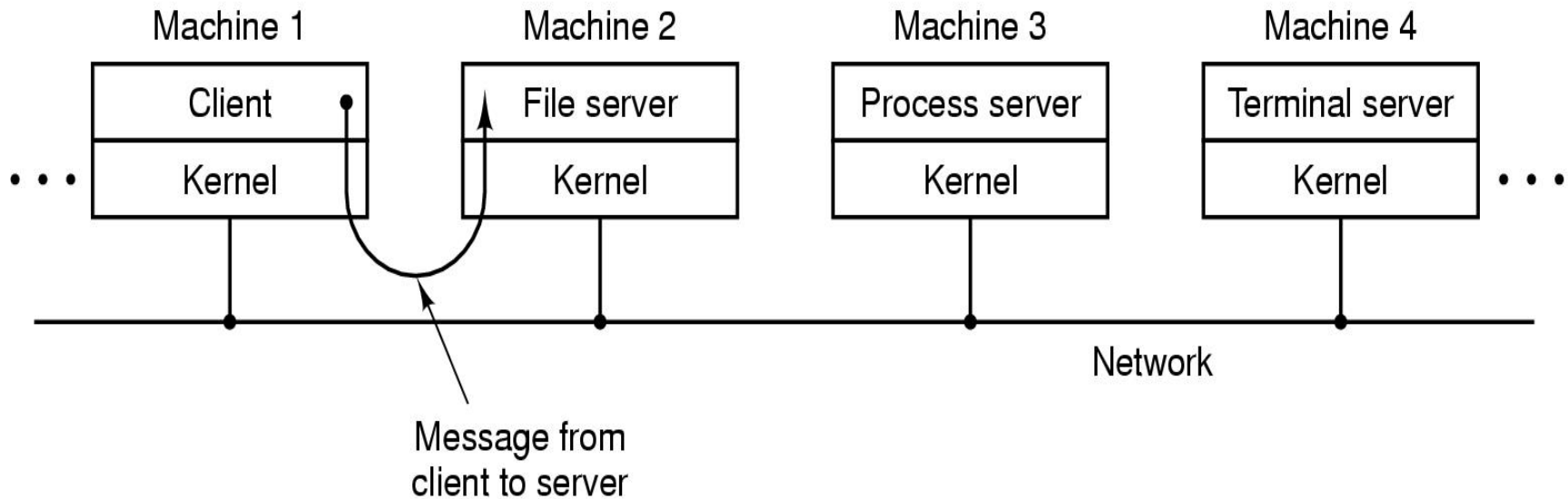
# Architettura logica di S/O – 6

## Struttura di tipo cliente-servente



# Architettura logica di S/O – 8

## Struttura distribuita



# Architettura logica di S/O – 7

- L'architettura di S/O a modello cliente-servente è anche detta a *micro-kernel*
- L'idea portante è di limitare al solo essenziale le responsabilità del nucleo delegando le altre a processi di sistema nello spazio di utente
  - I processi di sistema sono visti come serventi
  - I processi utenti sono visti come clienti
- Il ruolo del nucleo di S/O è di gestire i processi e supportare le loro comunicazioni
- Se un servizio va in crash difficilmente lo farà tutto il sistema
- Idea “pulita” ma prestazioni generalmente scadenti

# Monolitico vs. Microkernel

- [http://www.dina.dk/~abraham/Linus\\_vs\\_Tanenbaum.html](http://www.dina.dk/~abraham/Linus_vs_Tanenbaum.html)
- i kernel monolitici sono più semplici da realizzare e mantenere.
- i microkernel consentono gestione più flessibile
  - montare un certo servizio (es. un hard disk) solo per un utente invece che per tutti
    - il “montaggio” diventa un’operazione livello utente, non più livello kernel
- i microkernel hanno problemi di sincronizzazione tra le varie componenti, che ne rallentano sviluppo e mantenimento
- Linux è Monolitico, Minix è Microkernel, Windows e Mac OS/X sono ibridi.


From: ast@cs.vu.nl (Andy Tanenbaum)

Newsgroups: comp.os.minix

Subject: LINUX is obsolete

Date: 29 Jan 92 12:12:50 GMT

I was in the U.S. for a couple of weeks, so I haven't commented much on LINUX (not that I would have said much had I been around), but for what it is worth, I have a couple of comments now.

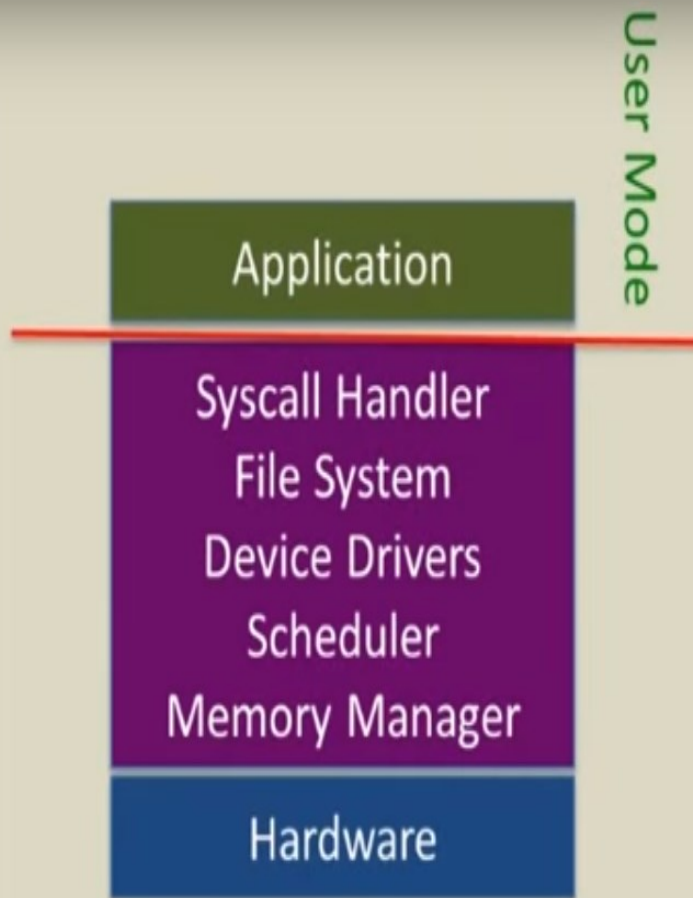
As most of you know, for me MINIX is a hobby, something that I do in the evening when I get bored writing books and there are no major wars, revolutions, or senate hearings being televised live on CNN. My real job is a professor and researcher in the area of operating systems. 

As a result of my occupation, I think I know a bit about where operating are going in the next decade or so. Two aspects stand out:

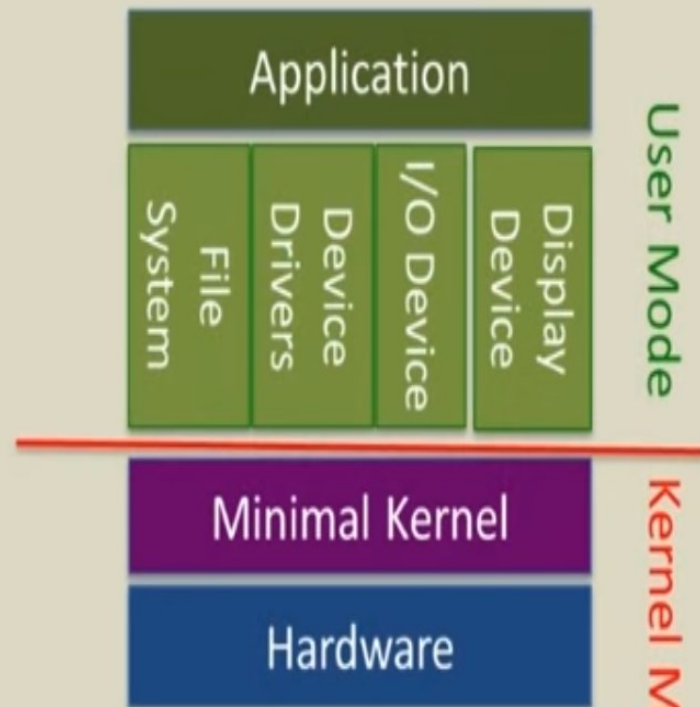


(Picture from 1998)





**Monolithic Kernel**



**Microkernel**

# Unità metriche

| Exp.       | Explicit                    | Prefix | Exp.      | Explicit                          | Prefix |
|------------|-----------------------------|--------|-----------|-----------------------------------|--------|
| $10^{-3}$  | 0.001                       | milli  | $10^3$    | 1,000                             | Kilo   |
| $10^{-6}$  | 0.000001                    | micro  | $10^6$    | 1,000,000                         | Mega   |
| $10^{-9}$  | 0.000000001                 | nano   | $10^9$    | 1,000,000,000                     | Giga   |
| $10^{-12}$ | 0.0000000000001             | pico   | $10^{12}$ | 1,000,000,000,000                 | Tera   |
| $10^{-15}$ | 0.0000000000000001          | femto  | $10^{15}$ | 1,000,000,000,000,000             | Peta   |
| $10^{-18}$ | 0.0000000000000000001       | atto   | $10^{18}$ | 1,000,000,000,000,000,000         | Exa    |
| $10^{-21}$ | 0.0000000000000000000001    | zepto  | $10^{21}$ | 1,000,000,000,000,000,000,000     | Zetta  |
| $10^{-24}$ | 0.0000000000000000000000001 | yocto  | $10^{24}$ | 1,000,000,000,000,000,000,000,000 | Yotta  |