v 1.0
20120919

Sistemi Operativi

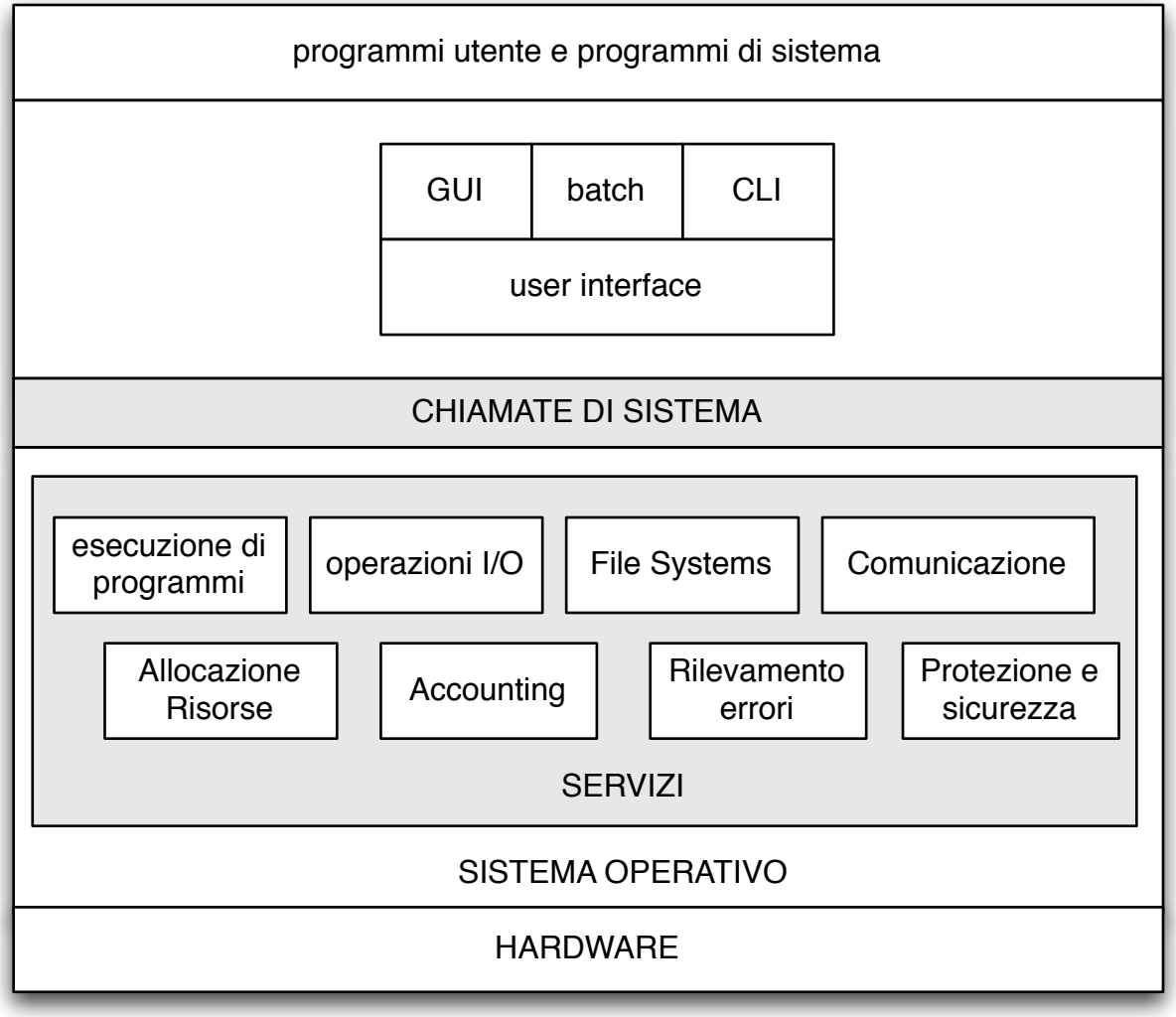
[modulo 1c]
STRUTTURA DEI SISTEMI OPERATIVI

DESCRIZIONE DEL MODULO

- Categorie di servizi offerti dal Sistema Operativo
- Chiamate di sistema
  - ciò che permette ai programmi di fruire dei servizi
  - rapporto funzioni di libreria e chiamate di sistema
- ~~Progettazione e sviluppo dei Sistemi Operativi~~
  - ~~principali architetture~~
  - ~~aspetti di implementazione, debugging~~
- ~~Generazione e booting~~

SERVIZI

SERVIZI FORNITI DAL S.O.



SERVIZI PRIMARI

- **Interfaccia con l'utente**
  - in forma di linea di comando (**CLI**, Command-line Interpreter), in forma grafica (**GUI**, Graphical User Interface), o in forma **batch interface** (comandi raccolti in script file)
- **Esecuzione dei programmi**
  - la capaciltà di caricare in memoria un programma ed eseguirlo
- **Operazioni di I/O**
  - la capacità di interfacciare programmi utente con l'I/O, mediante insieme di operazioni specifiche
- **Manipolazione del file-system**
  - servizi per leggere, scrivere, creare, e cancelllare file

SERVIZI PRIMARI

- **Comunicazione**
  - servizi per abilitare la comunicazione (scambio di informazioni) sia fra processi in esecuzione sul medesimo sistema, sia su nodi diversi di un sistema distribuito.
  - I modelli di comunicazione supportati sono tipicamente o a memoria condivisa o scambio di messaggi
- **Detecting degli errori**
  - servizi per specificare il comportamento che il sistema deve avere in caso accadano eventi anomali, causati da errori, nella CPU, nella memoria, nel sistema di I/O o nei programmi degli user

SERVIZI SECONDARI

- Servizi secondari
  - funzionalità e servizi esistono per assicurare una più efficiente gestione delle risorse ed esecuzione delle operazioni
- **Allocazione delle risorse**
  - allocazione delle risorse a più utenti o processi in esecuzione concorrentemente
- **Accounting**
  - tener traccia e memorizzare quali utenti o processi usano quali risorse come e quando
    - in modo da poter fare statistiche, attuare determinate politiche di allocazione e uso, o far pagare l'utilizzo (account billing)
- **Protezione**
  - assicurarsi che tutti gli accessi alle risorse del sistema siano controllati

INTERFACCIAMENTO CON L'UTENTE: INTEPRETE COMANDI

- Programma che permette di ottenere **comandi** dall'utente **via linea di comando** e di metterli in esecuzione
  - può essere parte del nucleo del sistema operativo, oppure semplicemente un programma come gli altri (es. UNIX, Windows XP)
    - mandato in esecuzione al login dell'utente (es. UNIX)
  - Esempi
    - C:\WINDOWS\SYSTEM32\cmd.exe ("Prompt dei comandi") nei sistemi Windows
    - /bin/sh in sistemi UNIX
  - in alcuni sistemi (es. UNIX) possono esistere vari tipi di interpreti
    - prendono il nome in questo caso di shell
    - es. Bourne shell (sh), C-Shell (csh), Bourne Again Shell (bash), Korn shell (ksh)
- I comandi concernono in massima parte la manipolazione del file system e dei processi in esecuzione
  - possono essere implementati direttamente all'interno dell'interprete oppure richiamare l'esecuzione di programmi di sistema collocati in una specifica directory del file system
    - es: /usr/bin in sistemi UNIX, C:\WINDOWS\SYSTEM32 nei sistemi Windows

INTERFACCIAMENTO CON L'UTENTE: GRAPHICAL USER INTERFACE (GUI)

- Programma che permette di ottenere comandi dall'utente mediante un'interfaccia grafica
  - metafora del desktop
    - mouse + finestre & menu & icone
  - anche in questo caso può essere parte del nucleo del sistema operativo, oppure semplicemente un programma come gli altri (es. UNIX, Windows XP)
    - mandato in esecuizione al login dell'utente (es. UNIX)
- Esempi
  - AQUA su Mac OS X
  - Windows Explorer su sistemi Windows
  - Common Desktop Environment (CDE) e X-Windows su sistemi UNIX
  - K-Desktop Environment (KDE) e GNOME (progetto GNU) su sistemi Linux

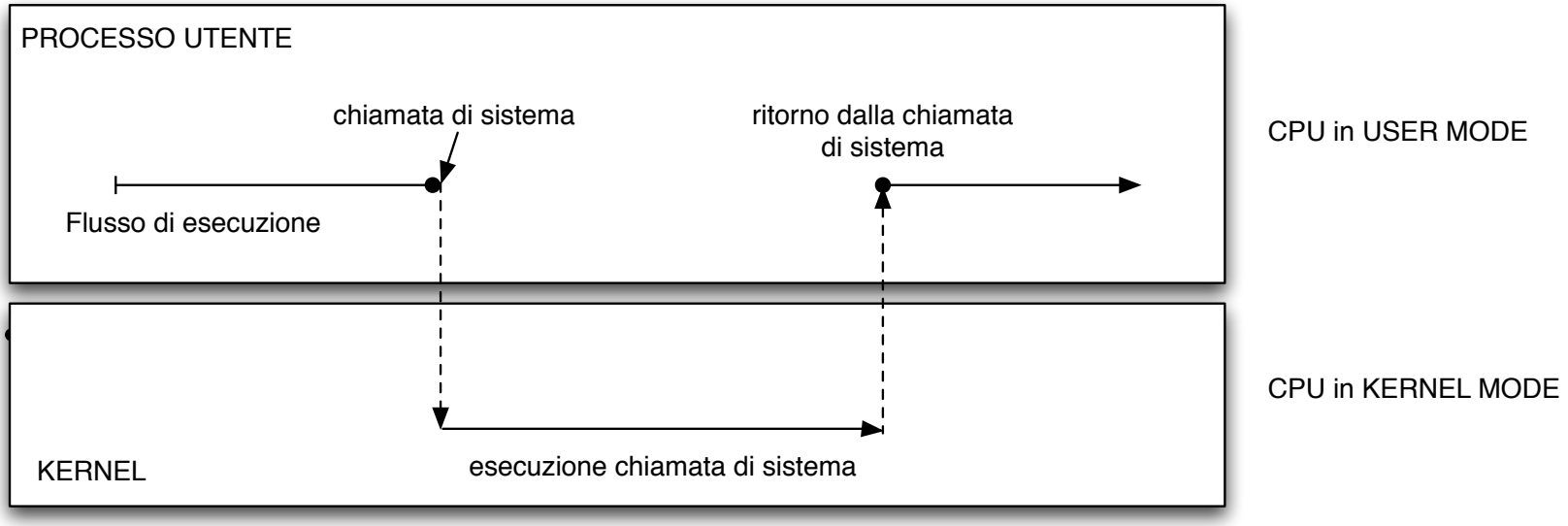
PROGRAMMI DI SISTEMA

- Programmi di utilità che forniscono complessivamente un ambiente che agevola lo sviluppo, esecuzione e debugging di programmi di alto livello, oltre che l'amministrazione del sistema
- Categorie
  - Gestione dei file
    - copiare, creare, spostare files, ...
  - Informazioni sullo stato
    - data corrente, processi in esecuzione, ...
  - File editors
  - Programmi di supporto per la programmazione
    - compilatori,...
  - Caricamento ed esecuzione programmi
    - ambienti runtime, linker, debugger...
  - Comunicazione
    - mail, web browser...

CHIAMATE DI SISTEMA (SYSTEM CALL)

CHIAMATE DI SISTEMA

- Le chiamate di sistema (**system call**) forniscono l'interfaccia con cui programmi in esecuzione (processi) possono accedere e sfruttare i servizi offerti dal sistema operativo
  - ogni possibile interazione con qualsiasi dispositivo di I/O coinvolge l'esecuzione di chiamate di sistema
    - dal mouse, al disco, alla scrittura di messaggi su console, etc.
- Comportano il cambio di modalità di esecuzione della CPU
  - da user mode a kernel mode



CHIAMATE DI SISTEMA E API

- Tipicamente un programma utente non invoca direttamente le chiamate di sistema, ma si appoggia su una **API (Application Programming Interface)** fornita dall'ambiente specifico di programmazione che si sta usando
  - API definiscono in generale un insieme di interfacce (procedure, classi, funzioni,...) utilizzata dalle applicazioni
- Esempi
  - **Win32 API**
    - API per sistemi Windows
  - **POSIX API**
    - API per sistemi POSIX
  - **Java API**
    - insieme di classi a disposizione per i programmi che devono essere eseguiti sulla JVM

ESEMPIO IN LINUX

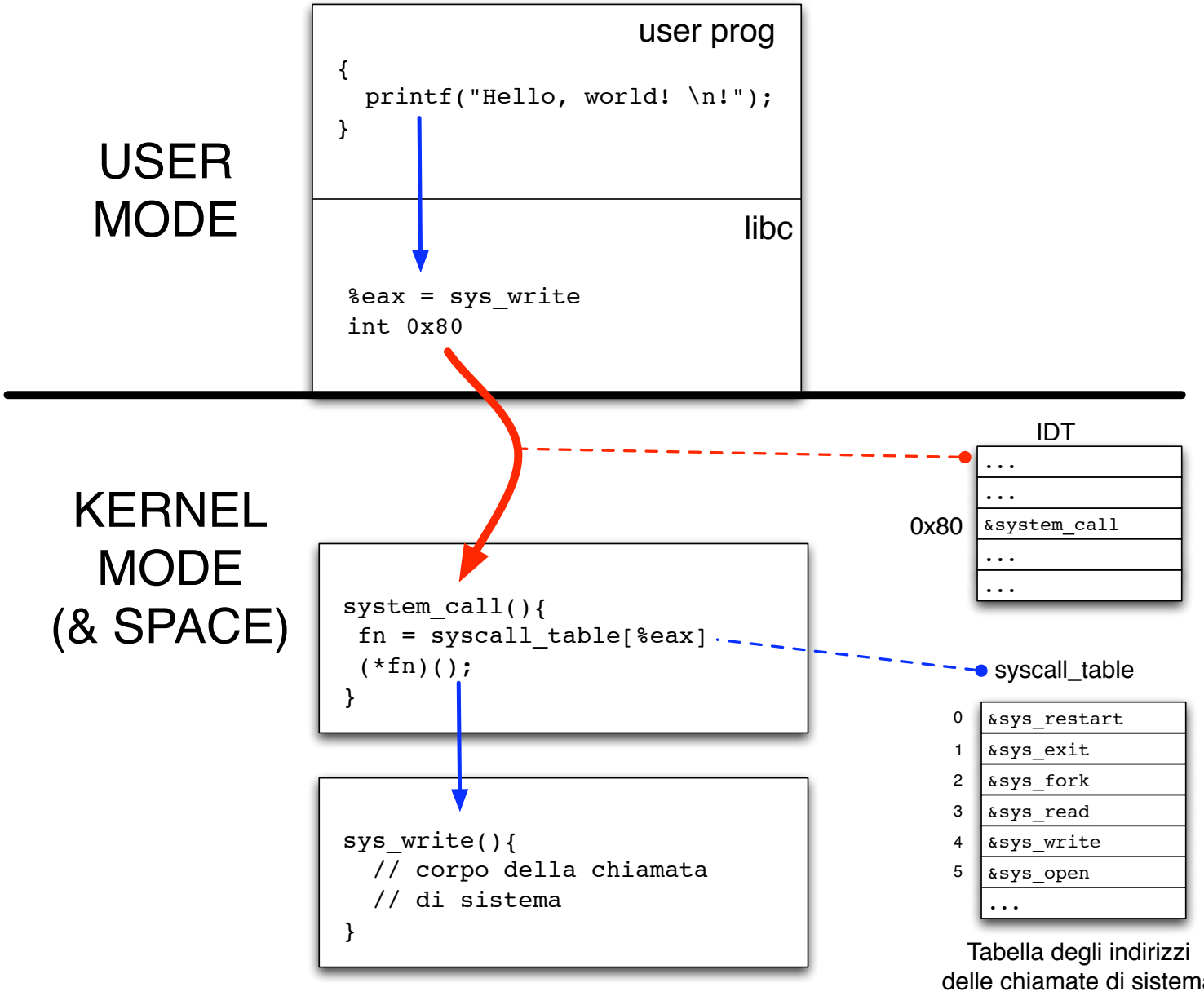


Tabella degli indirizzi delle chiamate di sistema

TIPI DI CHIAMATE DI SISTEMA

Process control & memory management
-end, abort
-load, execute
-create process, terminate process
-get process attributes, set process attributes
-wait for time
-wait event, signal event
-allocate and free memory

File management
-create file, delete file
-open,close
-read,write,reposition
-get file attribute, set file attribute

Device management
-request device, release device
-read, write, reposition
-get device attributes, set device attributes
-logically attach or detach devices

Information maintenance
-get time or date, set time or date
-get system data, set system data
-get process, file, or device attribute
-set process, file, or device attribute

Communications
-create, delete communication connection
-send, receive messages
-transfer status information
-attach or detach remote devices

DESIGN E IMPLEMENTAZIONE DI S.O.

MODELLI ARCHITETTURALI

- L'architettura di un sistema operativo definisce come sono organizzate le varie parti della struttura
  - struttura del kernel
  - componenti e collegamento fra i vari componenti
- Principali tipi di architetture:
  - **monolitica**
  - **a livelli (*layered*)**
  - **a microkernel**
  - **a moduli**
  - **exokernel**
  - **ibride**
  - **basate su macchina virtuale**

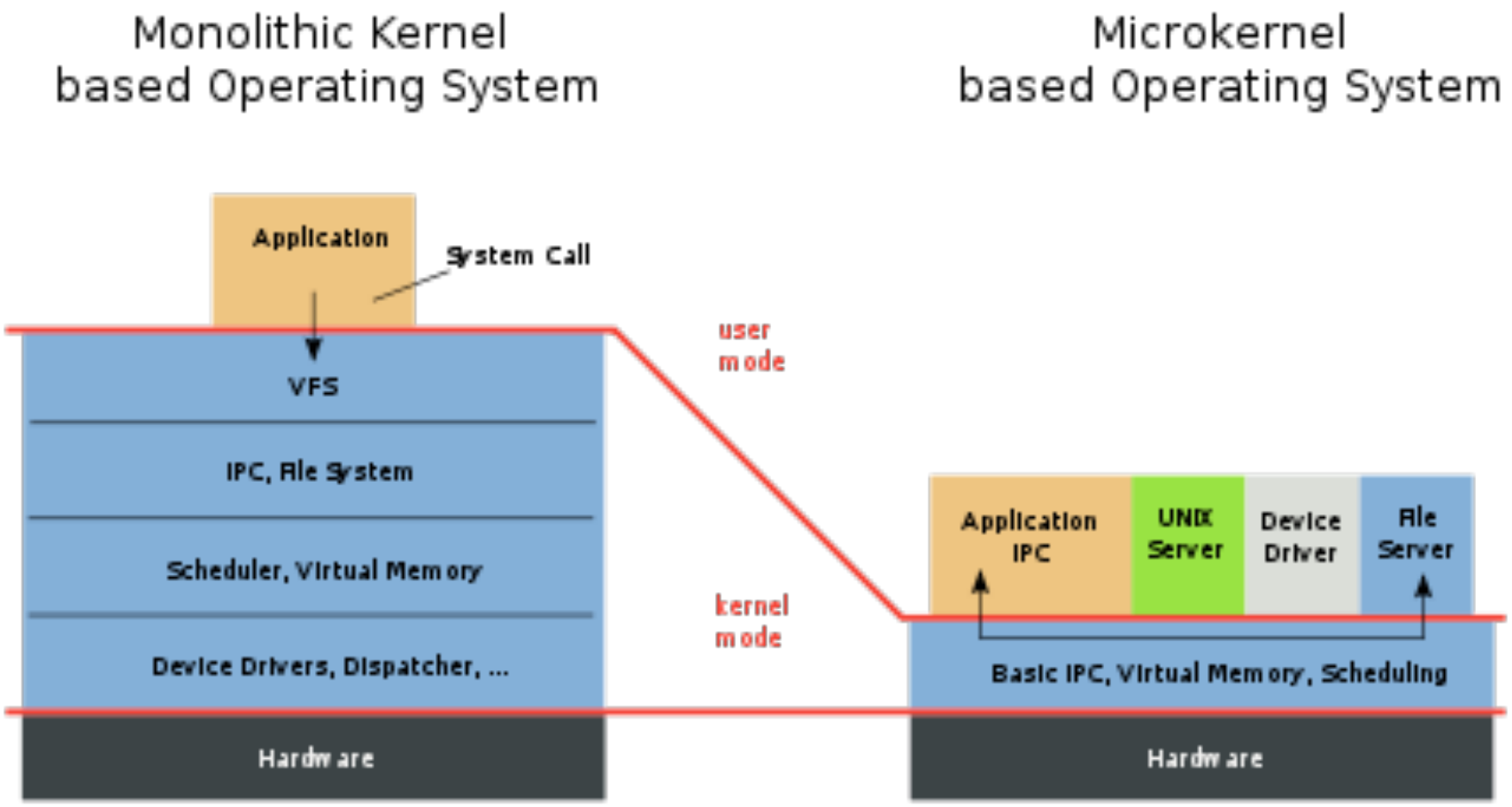
ARCHITETTURA MONOLITICA

- Architettura utilizzata dai primi sistemi operativi, con l'obiettivo di fornire le massime funzionalità avendo a disposizione risorse (memoria, capacità computazionale CPU) limitate
  - non è suddiviso in moduli
  - non c'è separazione fra interfacce e livelli di funzionalità
  - esempi:
    - IBM OS / 360
      - unico blocco, con più di un milione di linee di codice in assembly language
    - MSDOS
- Vantaggi
  - efficienza
- Svantaggi
  - comprensione e gestione dell'implementazione del sistema molto difficoltose
  - detecting di errori molto arduo
  - estendibilità del sistema difficoltosi

ARCHITETTURA A MICROKERNEL

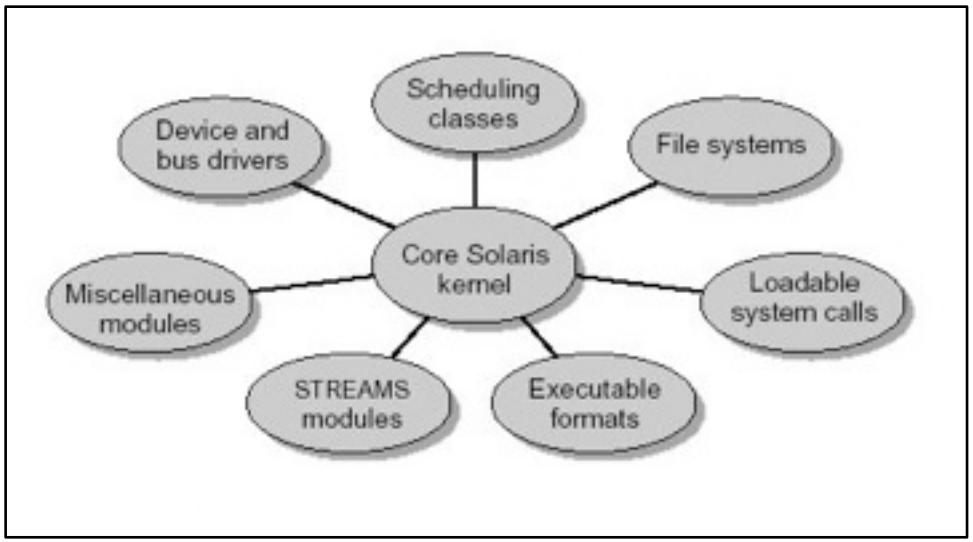
- Modello architetturale presente nei sistemi operativi più moderni, che integra modularità e separazione *politiche/meccanismi*
- Componenti principali:
  - **microkernel**
    - di piccole dimensioni, contiene l'insieme dei *meccanismi fondamentali* del sistema operativo, usati dai componenti server
    - unico componente ad essere eseguito in supervisor mode
  - componenti **server** (detti anche **manager**)
    - implementano le specifiche *politiche di gestione delle risorse e funzionalità,* che utilizzano i meccanismi del microkernel
    - eseguiti in user mode
  - comunicazione fra componenti e fra programmi e sistema operativo (system call) basata su *scambio di messaggi*

MONOLITICA VS. MICROKERNEL



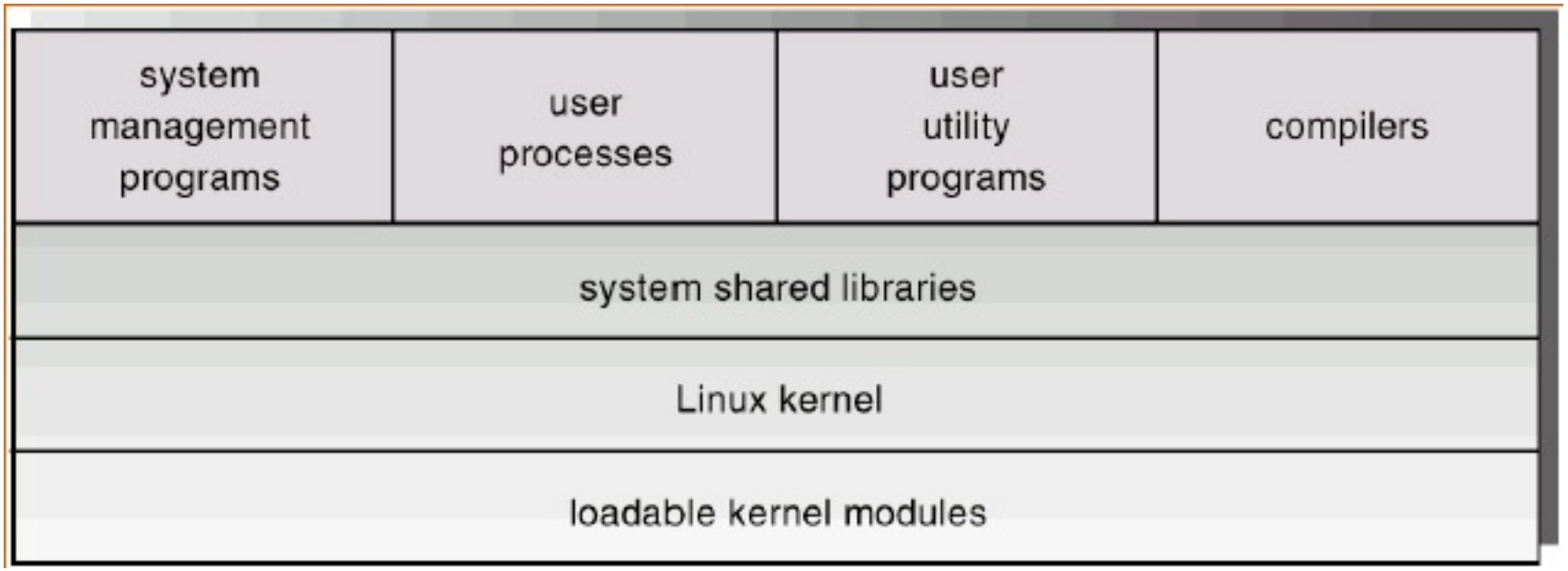
ARCHITETTURA A MODULI DINAMICI

- Architettura in cui il kernel è realizzato come insieme di moduli che possono essere caricati anche dinamicamente *(dynamically loadable modules)*
- Approccio utilizzato in Solaris, Linux e Mac OS X.
  - in Solaris ad esempio c'è un core kernel minimale e sette tipi di moduli kernel caricabili dinamicamente



ARCHITETTURA LINUX

- Monolitica con kernel suddiviso in moduli
  - dimensioni notevoli del kernel (2.6 ~ 6M loc)

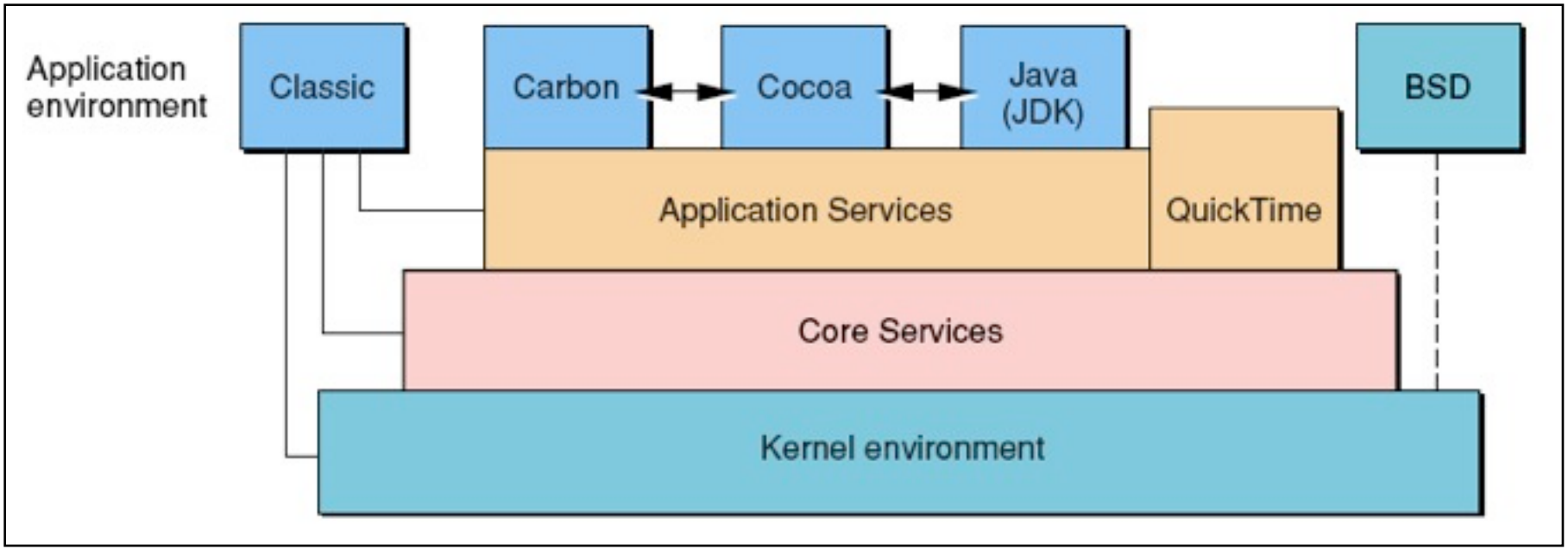


ARCHITETTURE IBRIDE

- Sistemi come Windows XP e Mac OS X adottano **architetture ibride,** in cui uniscono aspetti relativi ad architetture diverse, al fine di guadagnare in efficienza
- **Windows** unisce aspetti caratteristici delle architetture a livelli con aspetti caratteristici delle architetture a moduli
- **Mac OS X** adotta una tecnica a livelli, in cui uno dei livelli è dato dal Mach microkernel
  - i livelli superiori includono ambienti applicativi e insiemi di servizi per le applicazioni.
  - a livello sottostante c'è l'ambiente del kernel, basato su Mach microkernel e BSD Unix kernel
    - al primo è affidata gestione memoria, processi (thread) e comunicazione inter-processo
    - al secondo shell, gestione file system e rete, con implementazione dell'interfaccia POSIX standard.
  - a latere di Mach e BSD, l'ambiente del kernel mette a disposizione kit di I/O per lo sviluppo di device driver e moduli caricabili *dinamicamente* nel kernel (**kernel extension**).

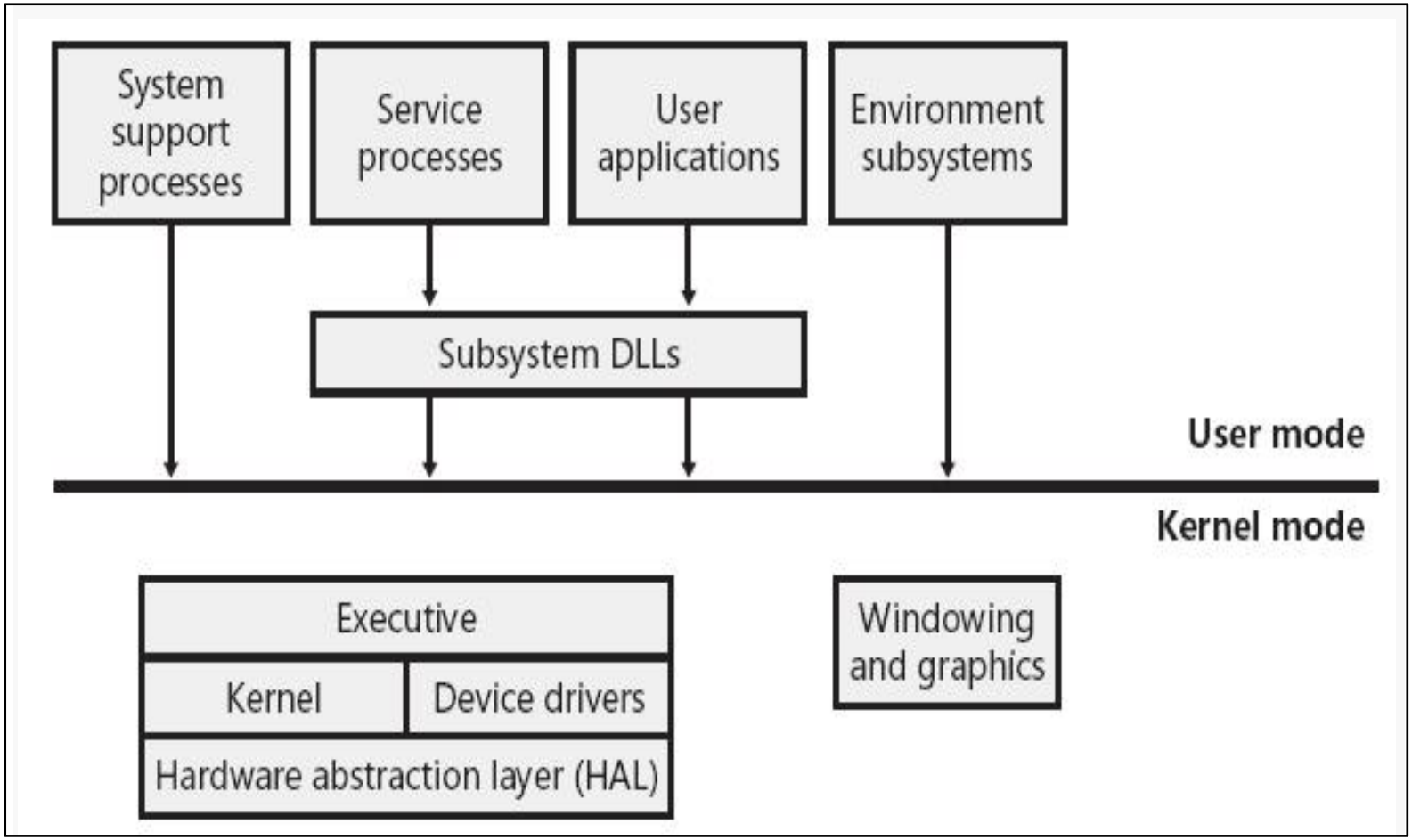
ARCHITETTURA Mac OS X

- Mac OS X architecture is composed by the following **layers**
  - Kernel Environment
  - Core Services
  - Application Services
  - Application Environment



FAMIGLIA WINDOWS (VISTA, XP, 7, 8)

- Architettura ibrida basata su livelli e moduli



LA NOZIONE DI MACCHINA VIRTUALE IN GENERALE

- In realtà la nozione di macchina virtuale è oggi utilizzata in vari ambiti al di là dei sistemi operativi, con significati diversi a seconda del contesto
  - **Application Virtual Machine**
  - **Hardware Virtual Machine**
    - chiamate anche System Virtual Machine

APPLICATION VIRTUAL MACHINES

- Application virtual machine are a piece of computer software that isolates the application being used by the user from the computer.
- Because versions of the virtual machine are written for various computer platforms, any application written for the virtual machine can be operated on any of the platforms
  - instead of having to produce separate versions of the application for each computer and operating system.
- The application is run on the computer using an interpreter or Just In Time compilation.
- Examples
  - first examples: USCD Pascal VM and Smalltalk Virtual Machine
    - 1970--1975
  - modern example: Sun Microsystem's **Java Virtual Machine (JVM)**
    - executing Java byte code
  - another one is the Microsoft **CLR (Common Language Runtime)**, at the core of **.NET** infrastructure
    - esecuting the MS-IL (MicroSoft-Intermediate Language)

HARDWARE VIRTUAL MACHINES

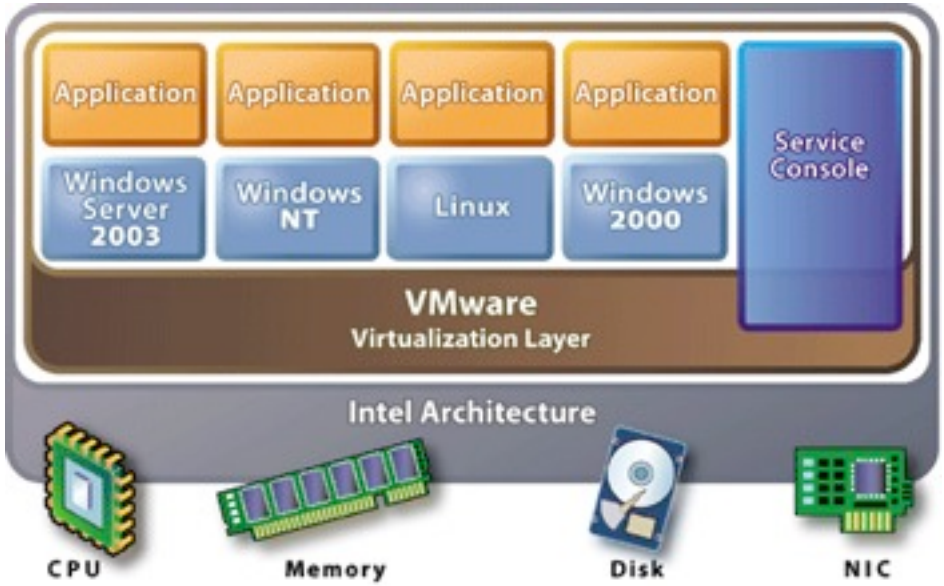
- The original meaning of virtual machine is that of a number of different identical execution environments on a single computer, each of which exactly emulates the host computer
  - this provides each user with the illusion of having an entire computer, but one that is their "private" machine, isolated from other users, all on a single physical machine.
- This is nowadays much better referred to by using the terms **virtualization** and **virtual servers**.
- The host software which provides this capability is often referred to as a **virtual machine monitor (VMM)** or **hypervisor**

HARDWARE VIRTUAL MACHINES

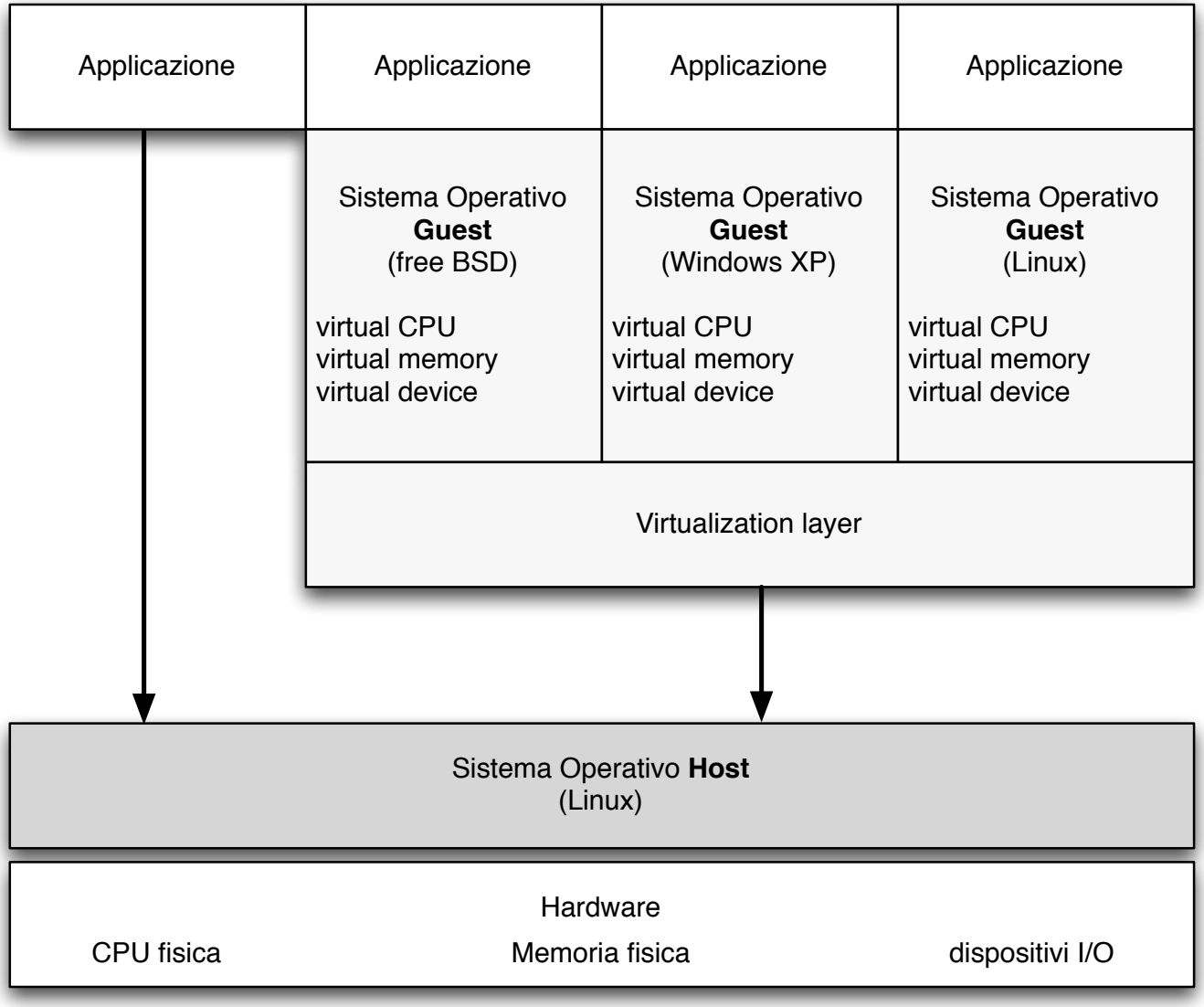
- The term virtual machine is now also used to refer to the environment created by an emulator, where software is used to emulate an operating system for the end user.
  - this is done to allow applications written for one OS to be executed on a machine which runs a different OS;
  - or to provide execution **sandboxes** which provide a greater level of isolation between processes than is achieved when running multiple processes on the same instance of an OS.
- This can be done using three major ways:
  - **full virtualization** — the virtual machine simulates the complete hardware, allowing an unmodified OS for a completely different CPU to be run.
    - Example: QEMU
  - **paravirtualization** — the virtual machine does not simulate hardware but instead offers a special API that requires OS modifications.
    - Example: XEN
  - **native virtualization** — the virtual machine only partially simulates enough hardware to allow an unmodified OS to be run in isolation, but the guest OS must be designed for the same type of CPU.
    - Example: VMWare

NATIVE VIRTUALIZATION EXAMPLES

- **VMWare**
  - Hardware Virtual Machine, virtualizing the Intel architecture
    - running as a simple application (not as an OS)
  - Similar products: Parallels, Virtual PC,...
- **Virtual Box**
  - x86 Virtualization, open source & well documented
  - http://www.virtualbox.org/



ARCHITETTURA VMWARE



PARA-VIRTUALIZATION: XEN example

- Developed and mantained by University of Cambridge
  - http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
- Xen system structure
  - Xen hypervisor as the lowest and most privileged layer
  - above this layer are one or more guest operating systems, which the hypervisor schedules across the physical CPUs
    - The first guest operating system, called in Xen terminology "domain 0" (dom0), is booted automatically when the hypervisor boots and given special management privileges and direct access to the physical hardware.
    - The system administrator logs into dom0 in order to start any further guest operating systems, called "domain U" (domU) in Xen terminology.
- By means of paravirtualization technique, Xen achieves high performance and low performance penalties, typically around 2%, with worst-case scenarios at 8% performance penalty
- Many operating systems have been already XEN-ed
  - FreeBSD, Linux, Windows XP,...

IMPLEMENTAZIONE DI UN S.O.

- I primi sistemi operativi erano implementati completamente in assembly language
- I sistemi operativi moderni sono implementati in massima parte in linguaggi di programmazione di alto livello, es: **linguaggio C** e **C++**
  - Linux, Windows, Darwin sono scritti per lo più in C, con piccole parti in assembly language
- Vantaggi
  - portabilità
  - menutenibilità, estendibilità
- In ambito di ricerca esistono numerosi sistemi operativi sperimentali implementati con linguaggi di più alto livello.
  - esempi:
    - Choices, implementato in C++
    - BeOS, implementato in C++
    - JNode, implementato in Java Singularity - S.O. Microsoft implementato in massima parte in Sing#, estensione di C#

DEBUGGING DI SISTEMI OPERATIVI

- Per **debugging** si intende l'attività con cui si cercano e rimuovono errori (bugs) in un sistema
  - siccome problemi relativi alle performance sono considerati bugs, il debugging è spesso associato al **profiling** (performance tuning), per rimuovere colli di bottiglia
- Analisi delle failure
  - nella maggior parte dei S.O., quando un processo fallisce, vengono scritte informazioni circa l'errore in un log file e il S.O. produce un **core dump**, una fotografia della memoria del processo memorizzata in un **core image file** per successive analisi
    - nel caso sia il kernel a fallire, il fallimento prende il nome di **crash**
- il tool **debugger** può essere utilizzato per analizzare i core dump
  - es. **gdb** *(GNU debugger)*

TRACING/PROFILING DEL S.O.

- Per fare il tuning delle performance del sistema servono opportuni strumenti di monitoraggio e analisi
  - logging eventi significativi, con tempo annotato, e analisi a posteriori con opportuni tools
  - oppure uso strumenti interattivi come **top**
- Fra i vari tool: **DTrace**
  - strumento inizialmente introdotto nel sistema operativo Solaris, poi adottato anche da altri sistemi operativi
    - es: FreeBSD, Mac OS X
  - permette di aggiungere dinamicamente dei **probe** (punti di monitoraggio) ad un sistema in esecuzione, *sia nello user space, sia nel kernel space*
  - un probe può essere interrogato mediante script (realizzati in un linguaggio di programmazione chiamato D) per determinare informazioni relativi al kernel, lo stato del sistema, l'attività dei processi, etc...
    - un probe attivato emette dati gestiti dal kernel
    - nel kernel, a fronte di nuovi dati di probe, vengono mandate in esecuzione opportune azioni (chiamate ECB, enabling control blocks) definite dai programmi in D

GENERAZIONE E BOOTING

GENERAZIONE DI UN SISTEMA OPERATIVO

- Concerne il processo di **generazione** o **configurazione** del sistema operativo per uno specifico sistema di elaborazione
  - il S.O. è distribuito normalmente su disco - CD-ROM / DVD oppure file ISO
  - per generare il sistema, si utilizza un programma speciale - definito genericamente *SYSGEN* - che legge la configurazione dell'elaboratore/ sistema ospitante da file oppure la chiede in input all'operatore, oppure esaminando direttamente l'hw
- Informazioni da determinare
  - tipo di CPU, quali set di istruzioni supportate, quante CPU
  - come verrà formattato il boot disk, quante partizioni
  - quanta memoria è disponibile
  - quali dispositivi sono disponibili, il loro indirizzo, le interruzioni interessate, il tipo e modello per ognuno
  - quali opzioni a livello di S.O.
    - dimensione buffer, parametri per lo scheduling, max. num processi,...

GENERAZIONE DI UN SISTEMA OPERATIVO

- Le informazioni possono essere usate in modo diverso:
  - ad un estremo, possono portare a modificare il codice sorgente del S.O., che viene ricompilato
  - una via meno radicale e più flessibile consiste nell'usare le info per compilare di tabelle di configurazione e alla selezione dei moduli che devono essere installati
  - all'altro estremo, il processo è totalmente table-driven: tutto il codice è sempre parte del S.O. e la selezione delle parti avviene a tempo di esecuzione, quando il sistema viene lanciato
    - la generazione in questo caso consiste solo nel compilare le tabelle

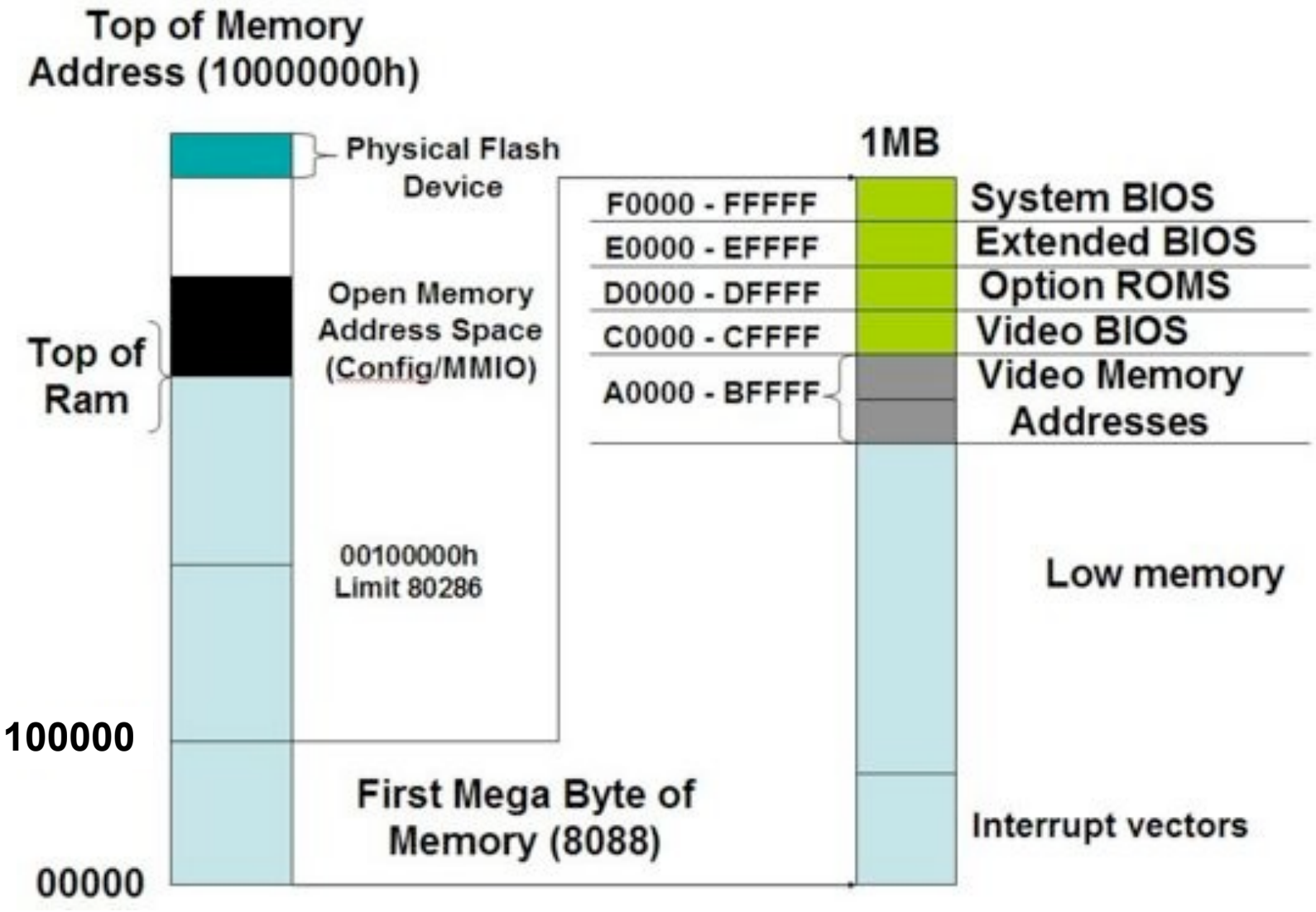
BOOTING

- Una volta generato, il sistema operativo deve poter essere caricato ed eseguito sul sistema hw.
- La procedura con cui si fa partire il computer caricando il kernel prende il nome di **booting** del sistema
- Il booting avviene grazie un programma denominato **bootstrap program** o **bootstrap loader**
  - ha il compito di recuperare il kernel dal dispositivo ove è memorizzato (es. disco fisso), caricarlo in memoria e mandarlo in esecuzione
- In alcuni sistemi, come i PC, questo processo avviene in due step:
  - un semplice bootstrap loader carica un boot program più complesso dal disco
  - il boot program a sua volta carica il kernel

BOOTING - DETTAGLI

- Quando la CPU riceve un segnale di reset (es. al power on), l'instruction registry è caricato con una locazione di memoria predefinita in **ROM** - dove risiede il programma di bootstrap iniziale, che fa parte del **BIOS**
  - l'esecuzione parte da quel punto
  - es: nei processori Intel 16 bit è all'indirizzo lineare 0xFFFF0
- Task del bootstrap program
  - eseguire una diagnostica sullo stato della macchina **(POST)**
  - inizializzare HW (registri, memoria, dispositivi,...)
  - far partire il sistema operativo

BIOS NEI PROCESSORI INTEL



BOOTING DI S.O. SU DISPOSITIVI

- In alcuni sistemi - tipicamente di piccole dimensione, come PDA, cellulari, console videogiochi - *tutto il S.O. è in ROM*
  - problema: cambiamenti del bootstrap code richiedono cambio dei ROM chip
  - soluzione: **EPROM** (erasable programmable ROM) - read-only ma con la possibilità di essere scrivibili in seguito a specifici comandi
- Tutte le forme di ROM prendono il nome di **firmware** essendo a mezza via fra HW e SW
  - un problema dell'avere il codice in firmware è che è più lento della RAM
  - ragion per cui alcuni sistemi copiano il S.O. da firmware a RAM una volta fatto il boot

BOOTING DI S.O. GENERAL-PURPOSE

- Per S.O. di grandi dimensioni / general-purpose o per sistemi che possono cambiare frequentemente, *solo il bootstrap loader è su firmware* e il S.O. è su disco
- In questo caso, il bootstrap esegue la diagnostica e ha giusto le istruzioni che servono per caricare un singolo blocco da disco (chiamato **boot block**) in una locazione prefissata (es: blocco 0) in memoria, e quindi eseguire il codice del boot block.
  - il programma nel boot block può essere fatto in modo tale da caricare l'intero sistema operativo in memoria e mandarlo in esecuzione.
  - in realtà, nella maggior parte dei casi si limita a caricare il resto del bootstrap program, conoscendone indirizzo su disco e la sua lunghezza.
    - **GRUB** è un esempio di bootstrap program open-source per Linux
- Un disco con una partizione con bootstrap program viene chiamato **boot disk** o **system disk**
- Una volta che il bootstrap program è stato completamente caricato, può navigare il file system del disco per trovare e caricare in memoria centrale il kernel del S.O., e iniziare la sua esecuzione.
  - da questo punto in poi il sistema si dice **running**