



# Sistemas Operativos Introducción

Semestre II-2013

# ¿Qué es un sistema operativo y por qué necesitamos uno?

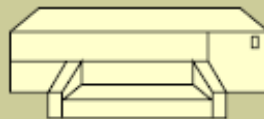
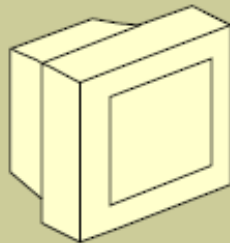
---

- ▶ ¿Qué es lo que tenemos?
  - ▶ Conjunto de recursos comunes
- ▶ ¿Qué es lo que necesitamos?
  - ▶ Una manera simple que permita a las aplicaciones utilizar estos recursos

# Recursos

---

## Hardware



## Network



# Aplicaciones

---

## Application Software

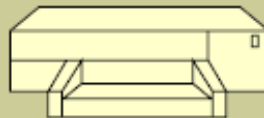
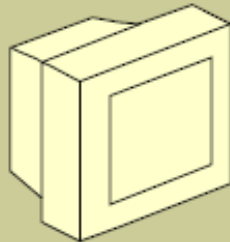
Firefox

Second Life

Yahoo  
Chat

GMail

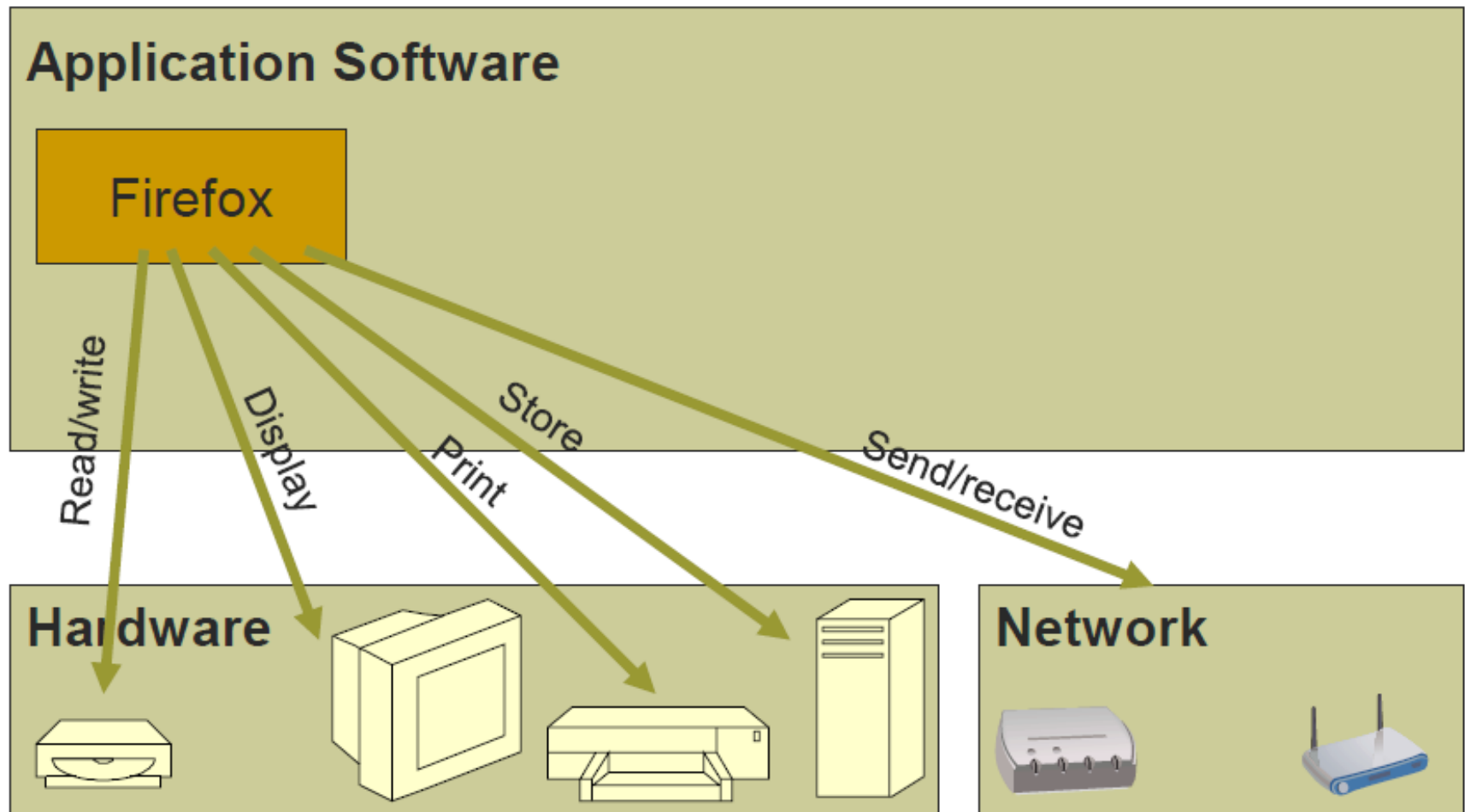
## Hardware



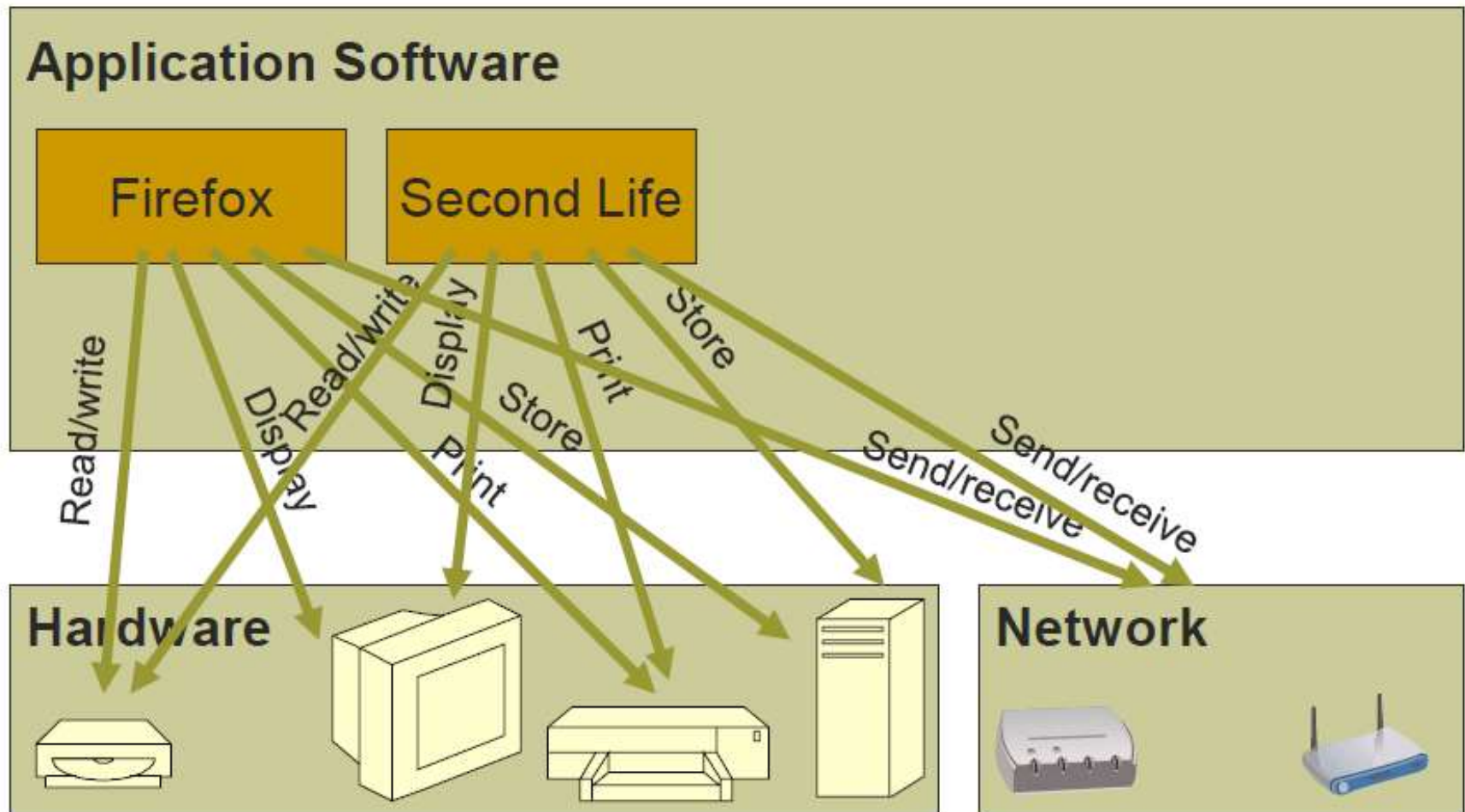
## Network



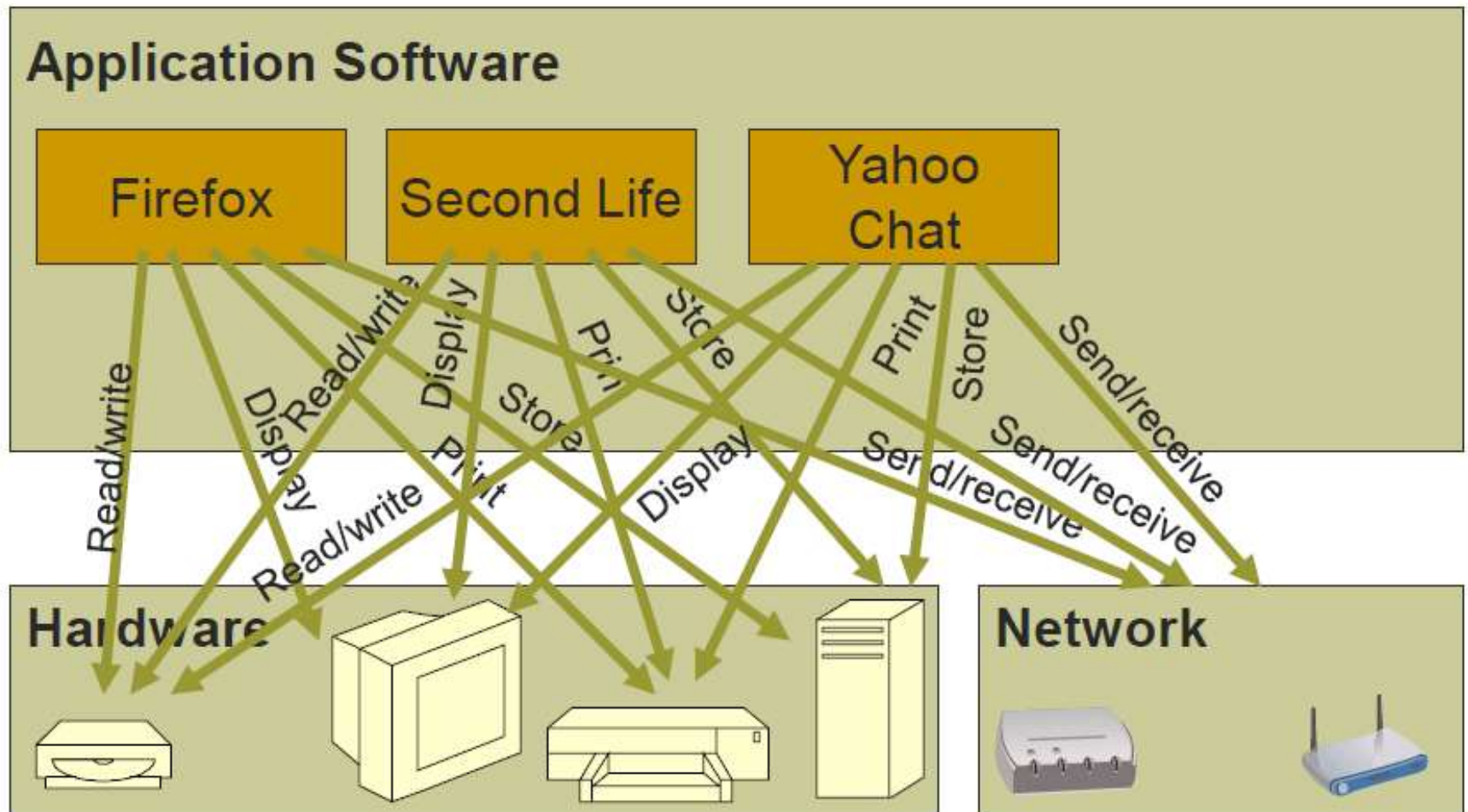
# Requerimientos de las Aplicaciones



# Dos aplicaciones

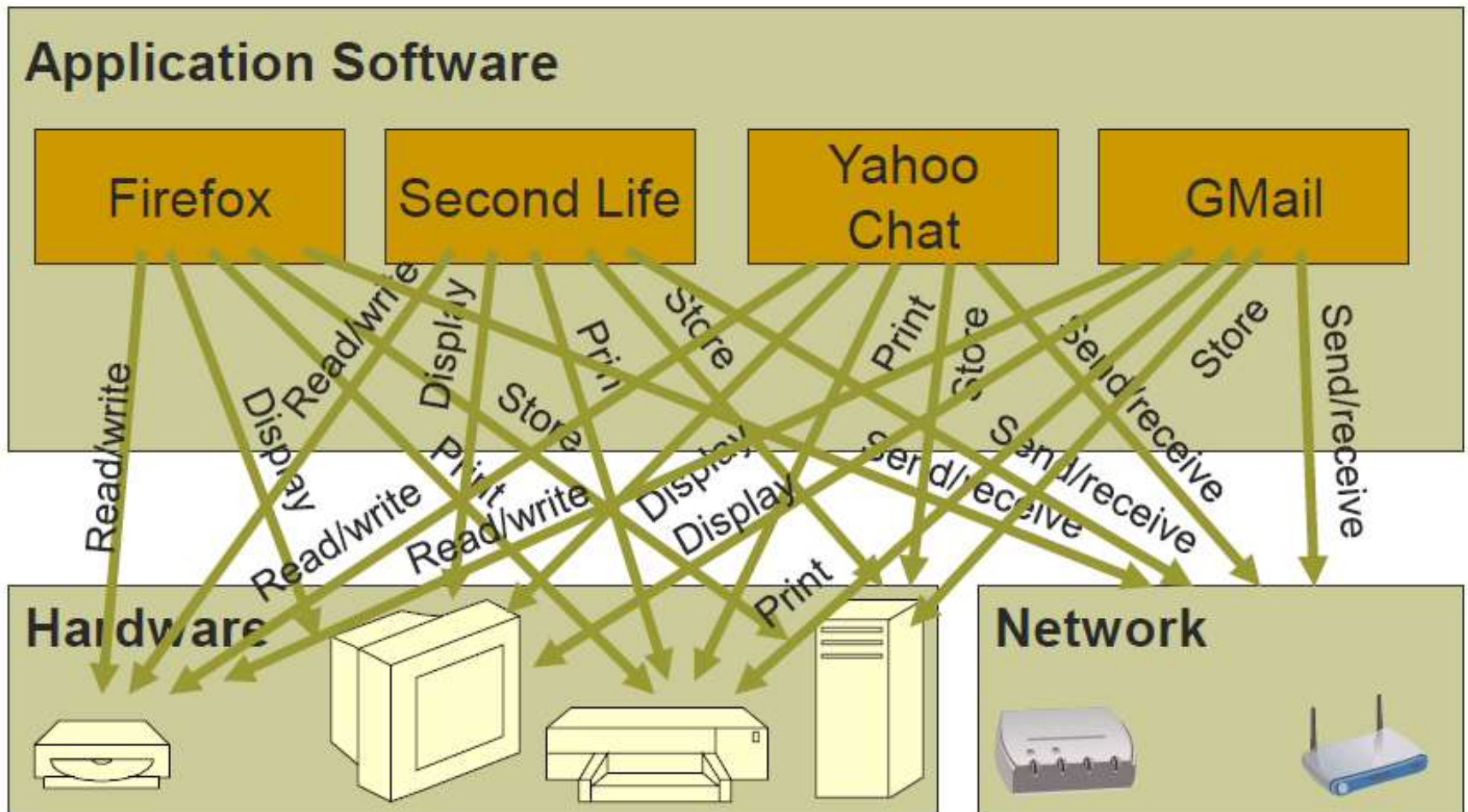


# Múltiples Aplicaciones



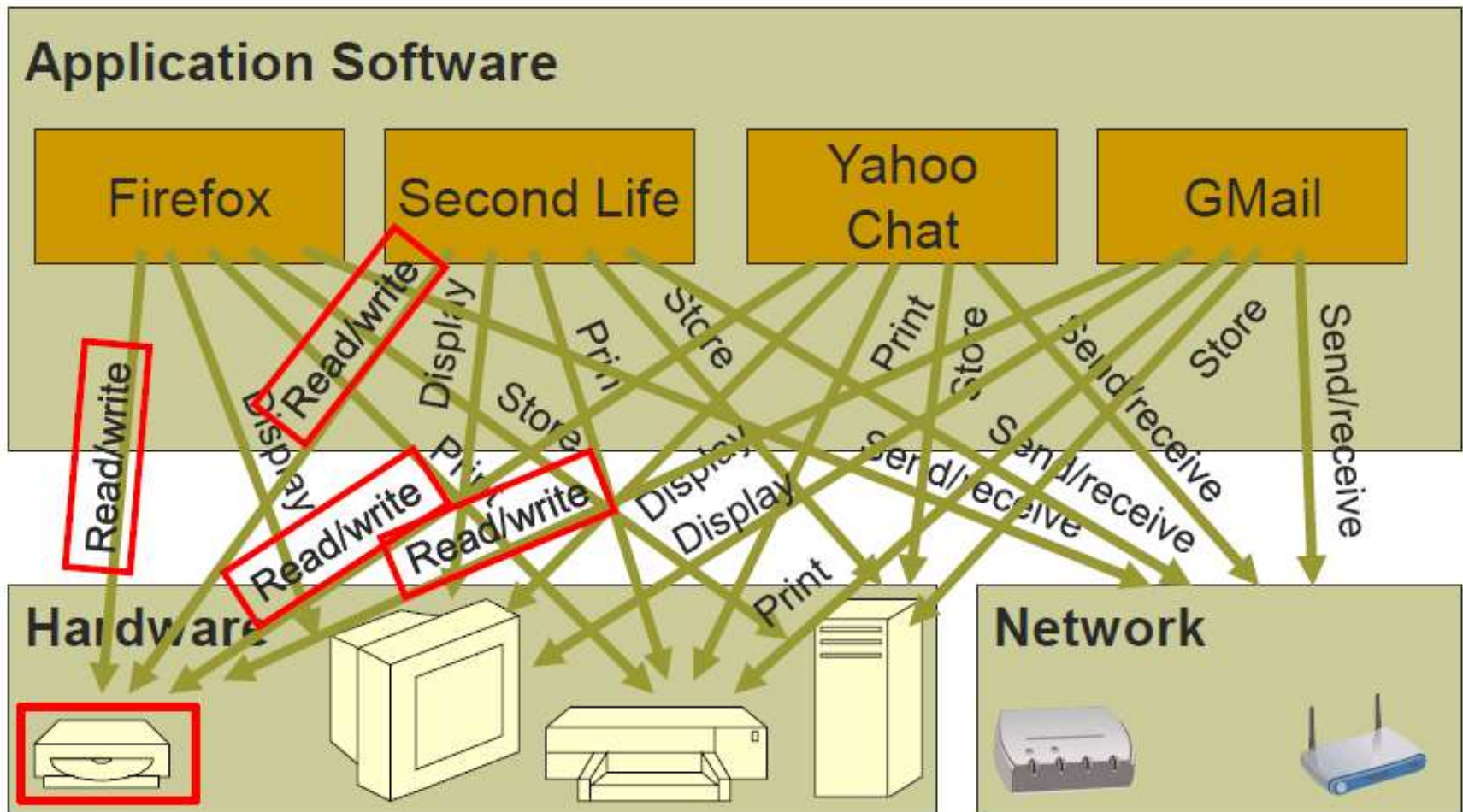


# ¡Necesitamos Ayuda!

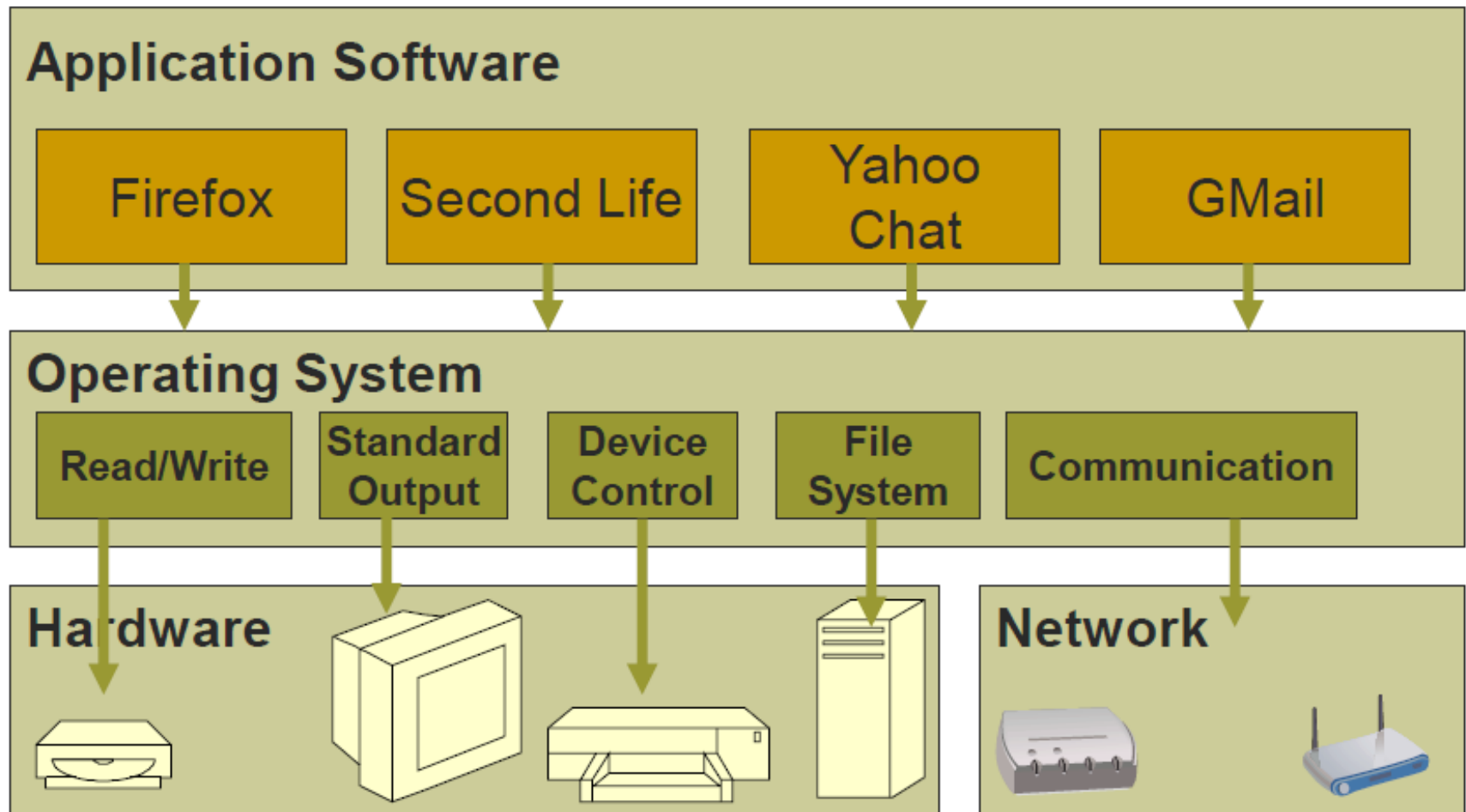




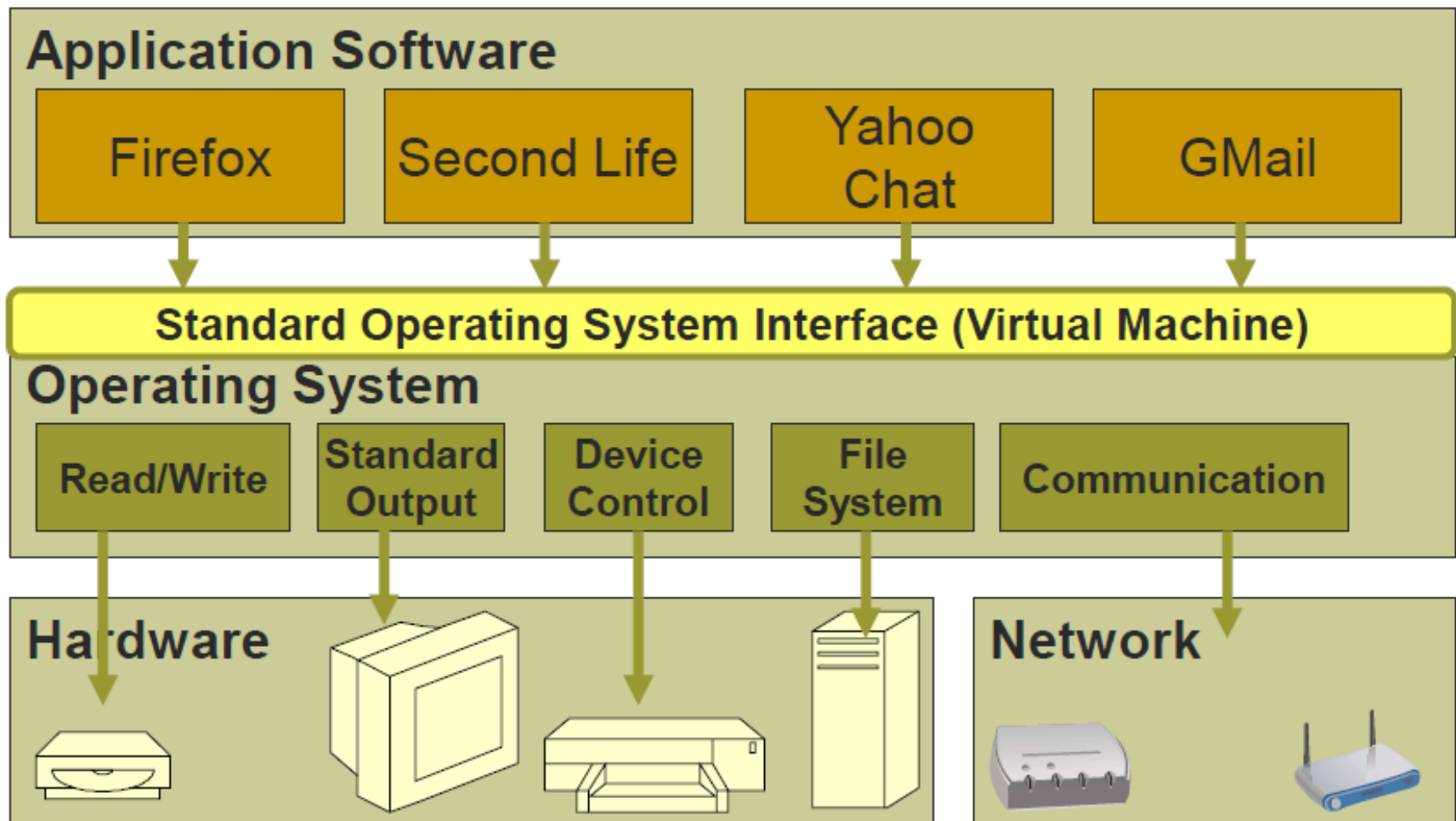
# Aproximación – Funciones Comunes



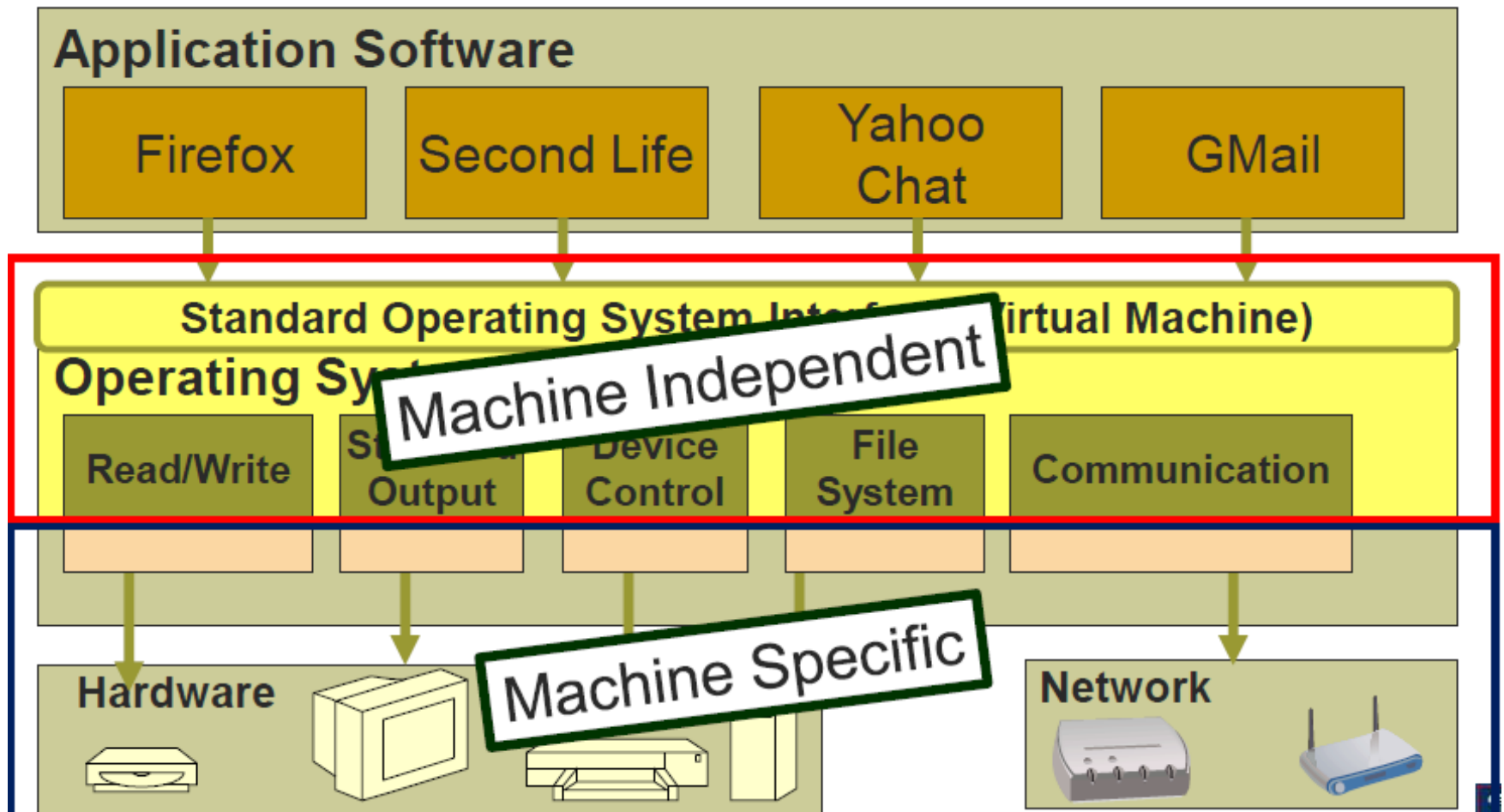
# Delegar Funciones Comunes



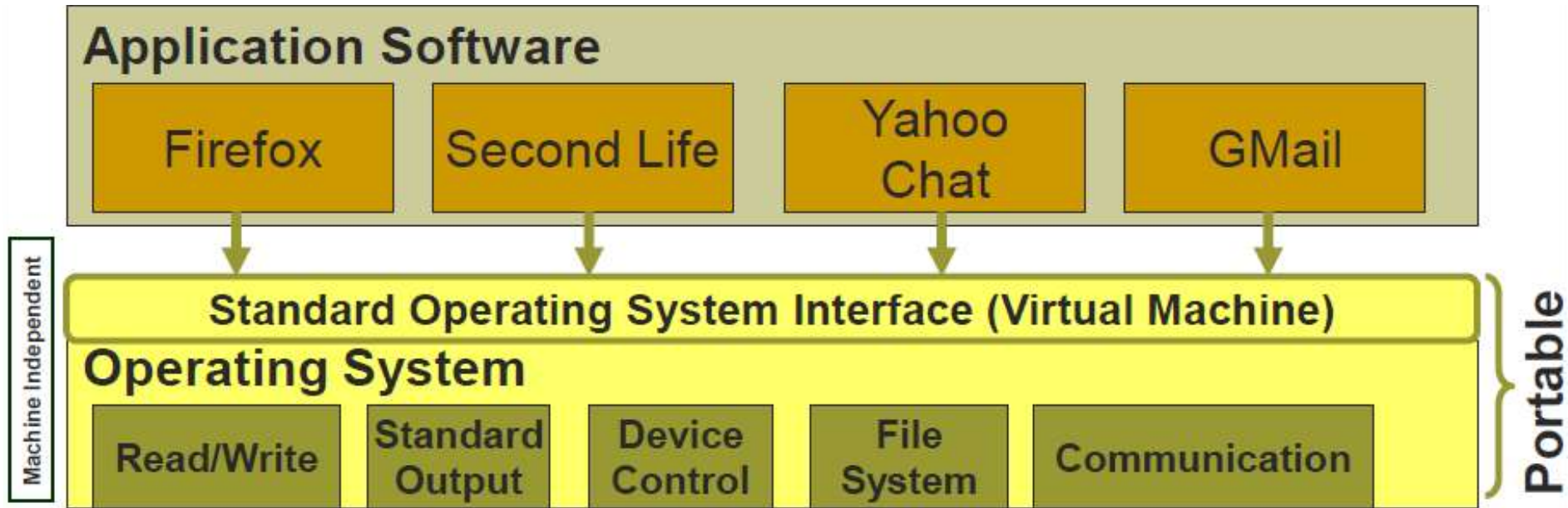
# Exportar una Interfaz Estándar



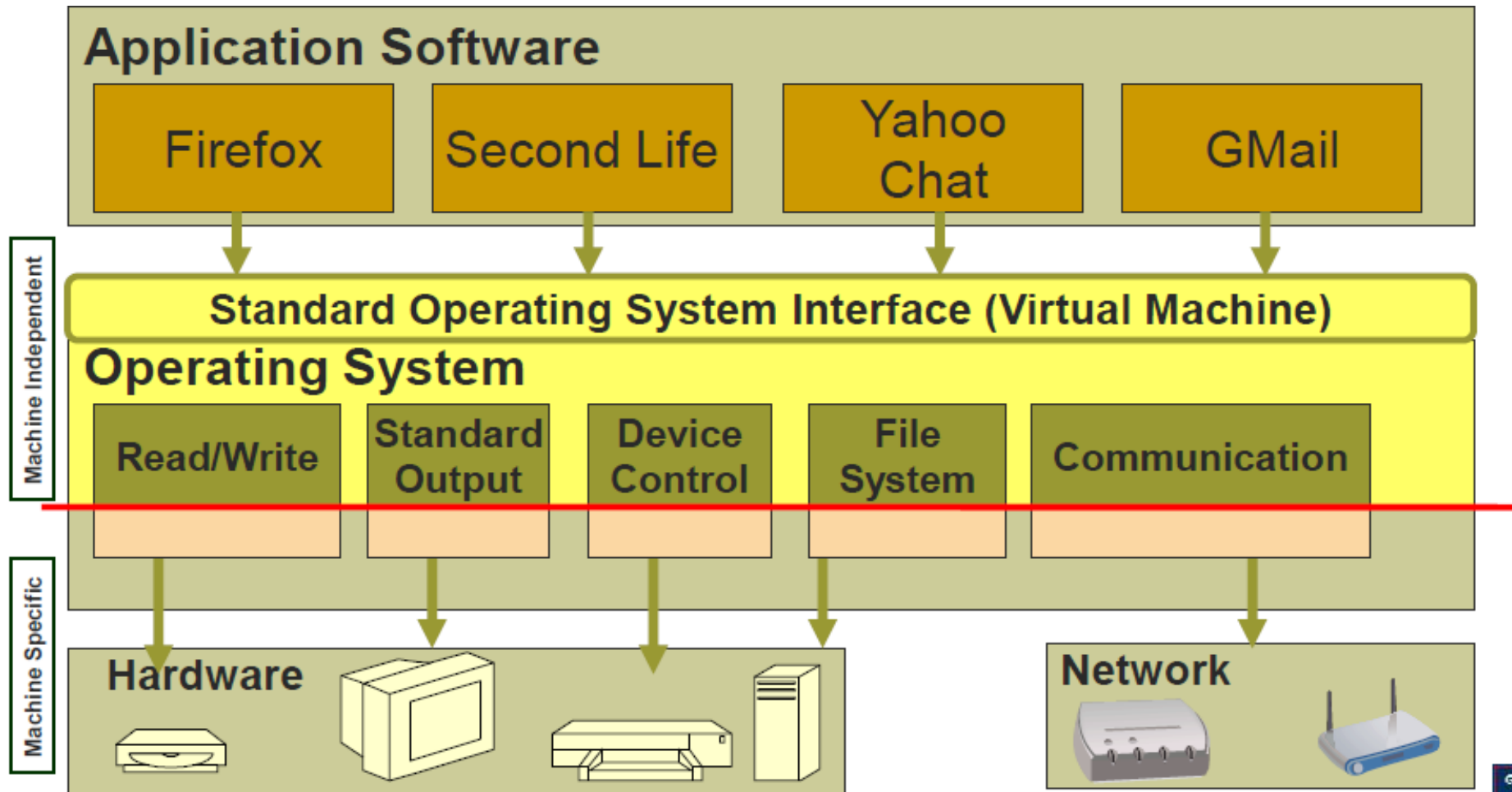
# Objetivo – Incrementar la Portabilidad



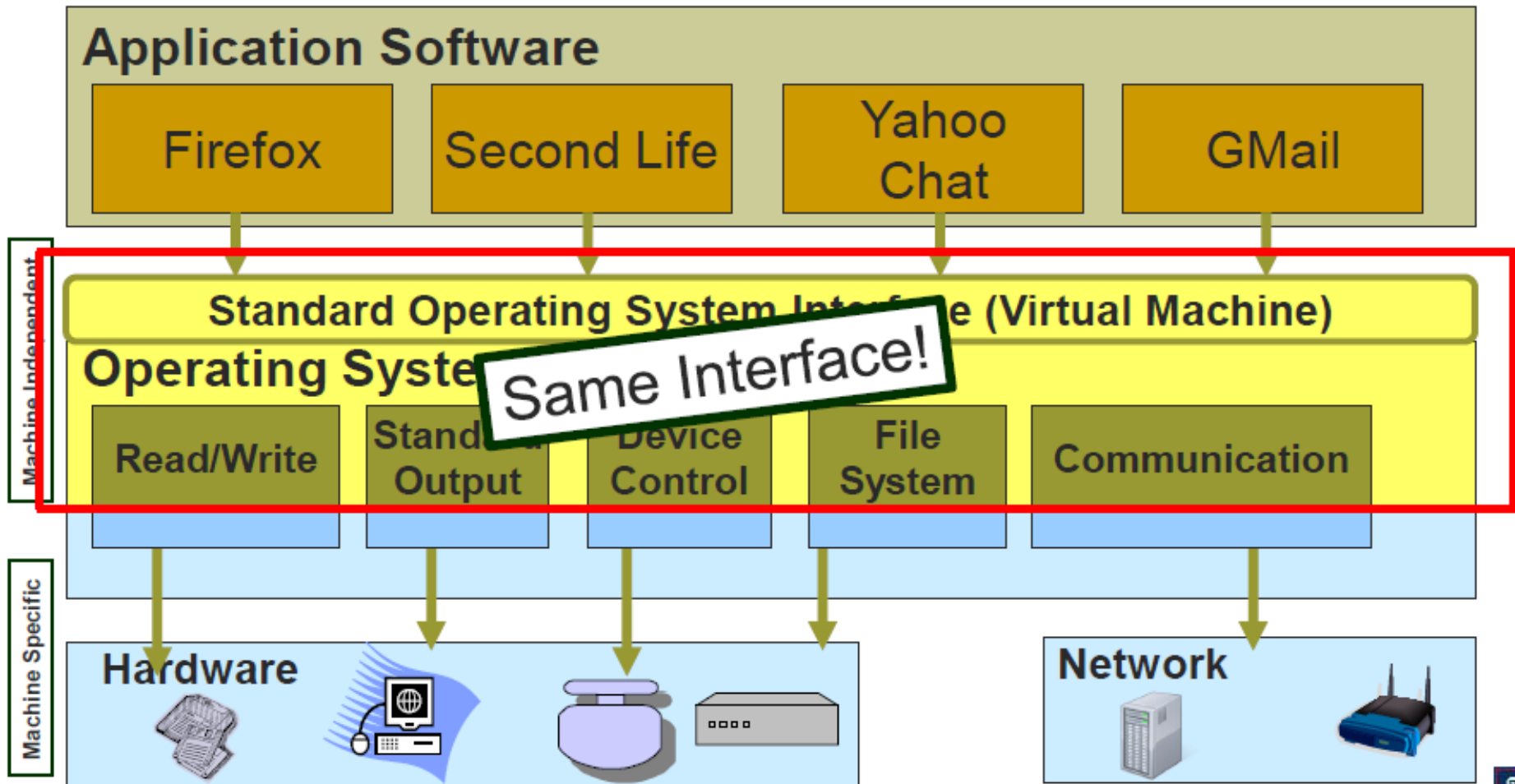
# Independencia de la Máquina => Portabilidad



# SO – Múltiples Plataformas



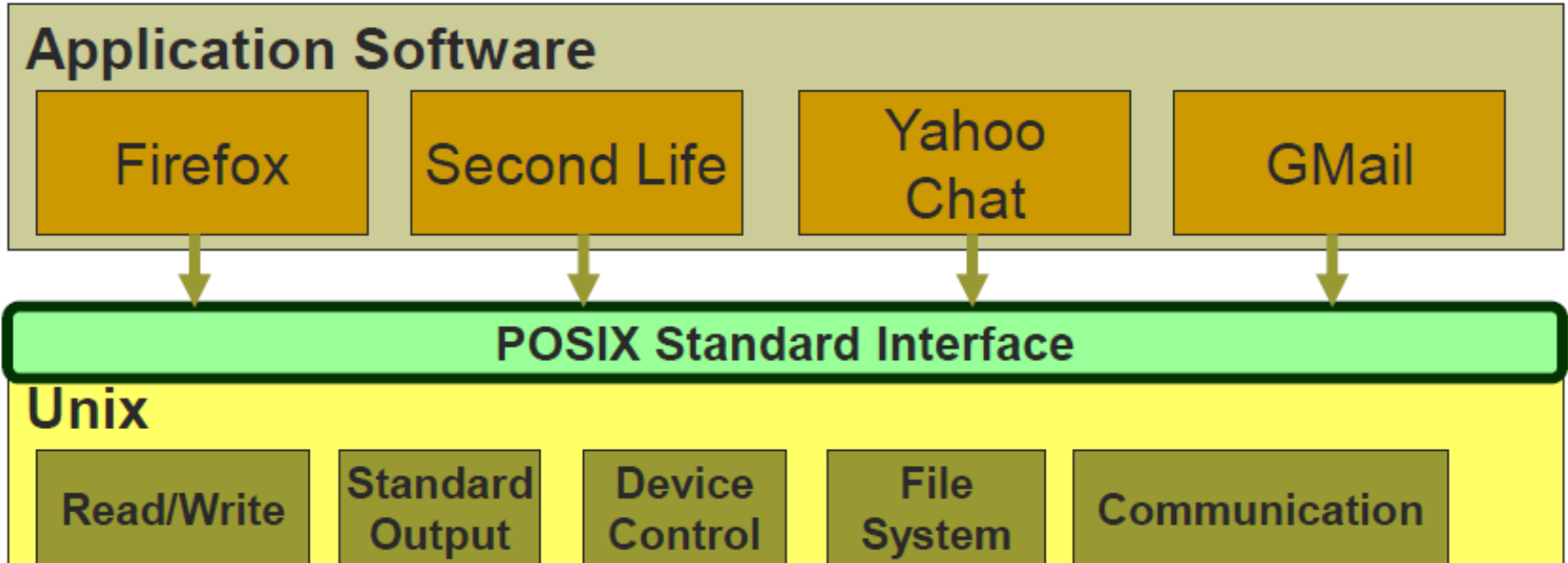
# SO – Múltiples Plataformas





# POSIX – Interfaz Estándar de UNIX

---



# ¿Qué es un Sistema Operativo?

---

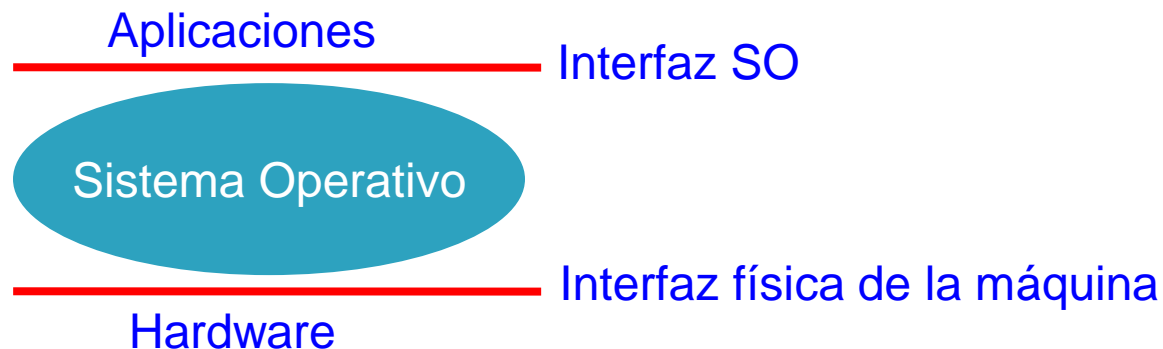
- ▶ ¡Magia!
- ▶ Existe un gran número de definiciones:
  - ▶ Pruebe en Google con define: Operating System
- ▶ Unas pocas de estas definiciones:
  - ▶ “El software del que depende el resto del software para hacer el computador funcional”
  - ▶ “El único programa que se ejecuta todo el tiempo en el computador”
  - ▶ “Un programa que administra todos los otros programas en el computador”

# ¿Qué es un Sistema Operativo?

---

## ► Definición

- El Sistema Operativo (SO) provee una máquina virtual sobre el tope del hardware real, cuya interface es más conveniente que la interfaz con el hardware desnudo
- Ventajas
  - Fácil de usar, simple de codificar, más confiables, más seguros. Se puede decir: “Escribir XYZ en el archivo ABC”



# Propósito del SO

---

- ▶ Dos funciones principales:

- ▶ Administrar los recursos físicos:

- ▶ Manejar varios dispositivos:

- Procesador, memoria, discos, redes, pantallas, cámaras, etc.

- ▶ De manera eficiente, confiable, tolerando y enmascarando las fallas, etc.

- ▶ Proveer un ambiente de ejecución para las aplicaciones corriendo sobre el computador (programas como Word, Emacs, etc.):

- ▶ Provee recursos virtuales y sus interfaces

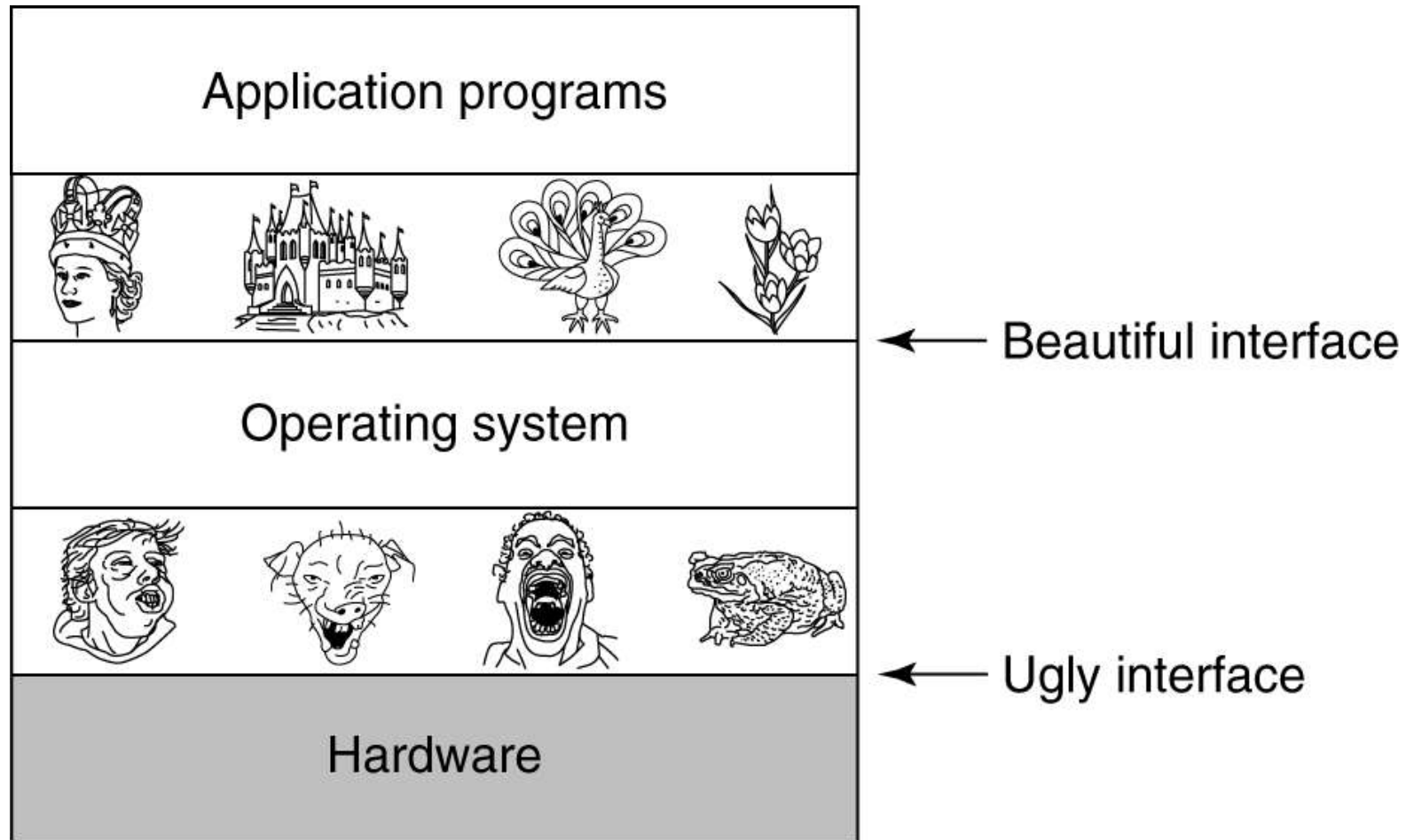
- Archivos, directorios, threads, procesos, etc.

- ▶ Simplifica la programación mediante abstracciones de alto nivel.

- ▶ Provee al usuario con un ambiente estable

# Abstracciones

---



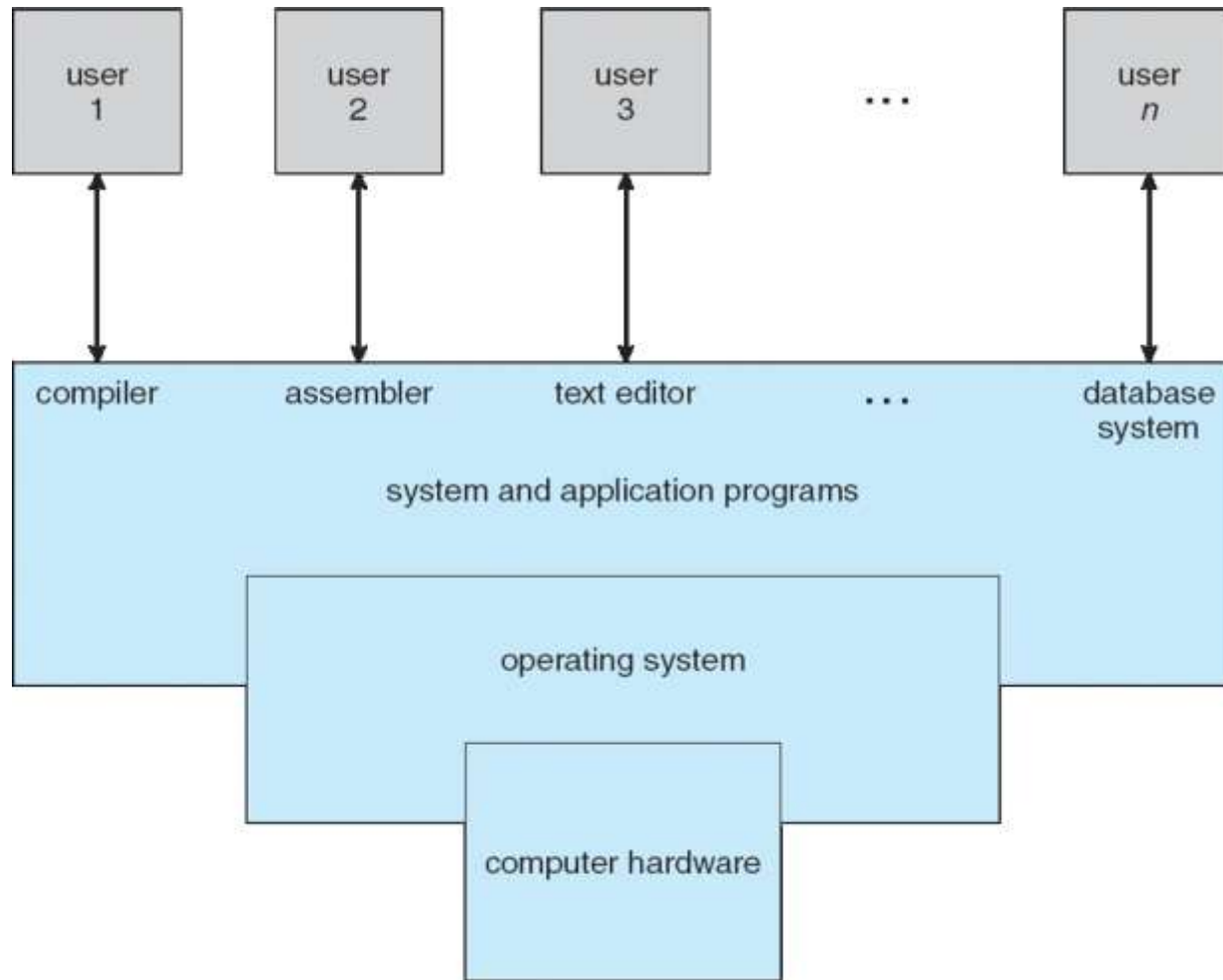
# Estructura del Sistema Computador

---

- ▶ Se puede dividir en cuatro componentes:
  - ▶ Hardware
    - ▶ Provee recursos básicos de computación
      - CPU, memoria, dispositivos de E / S.
  - ▶ Sistema Operativo
    - ▶ Controla y coordina el uso del hardware entre distintas aplicaciones y usuarios
  - ▶ Los programas de aplicación
    - ▶ Definen las formas en que los recursos del sistema se utilizan para resolver los problemas computacionales de los usuarios
      - Procesadores de texto, compiladores, navegadores web, sistemas de bases de datos, juegos de vídeo
  - ▶ Usuarios
    - ▶ Personas, máquinas, otros equipos

# Los cuatro componentes de un Sistema Computador

---





# ¿Qué hacen los Sistemas Operativos?

---

- ▶ Depende del punto de vista.
- ▶ Los usuarios quieren conveniencia y **facilidad de uso**.
  - ▶ No se preocupan por la **utilización de los recursos**.
- ▶ Pero en computadores compartidos, como un **mainframe** o una **minicomputadora**, se deben mantener contentos a todos los usuarios.
- ▶ Los usuarios de sistemas dedicados, tales como **estaciones de trabajo**, tienen recursos dedicados, pero con frecuencia utilizan los recursos compartidos de los **servidores**.
- ▶ Las computadoras portátiles tienen escasos recursos, y son optimizados para lograr facilidad de uso y duración de la batería.
- ▶ Algunos equipos tales como computadores integrados en dispositivos y los automóviles, tienen muy poca o ninguna interfaz de usuario.

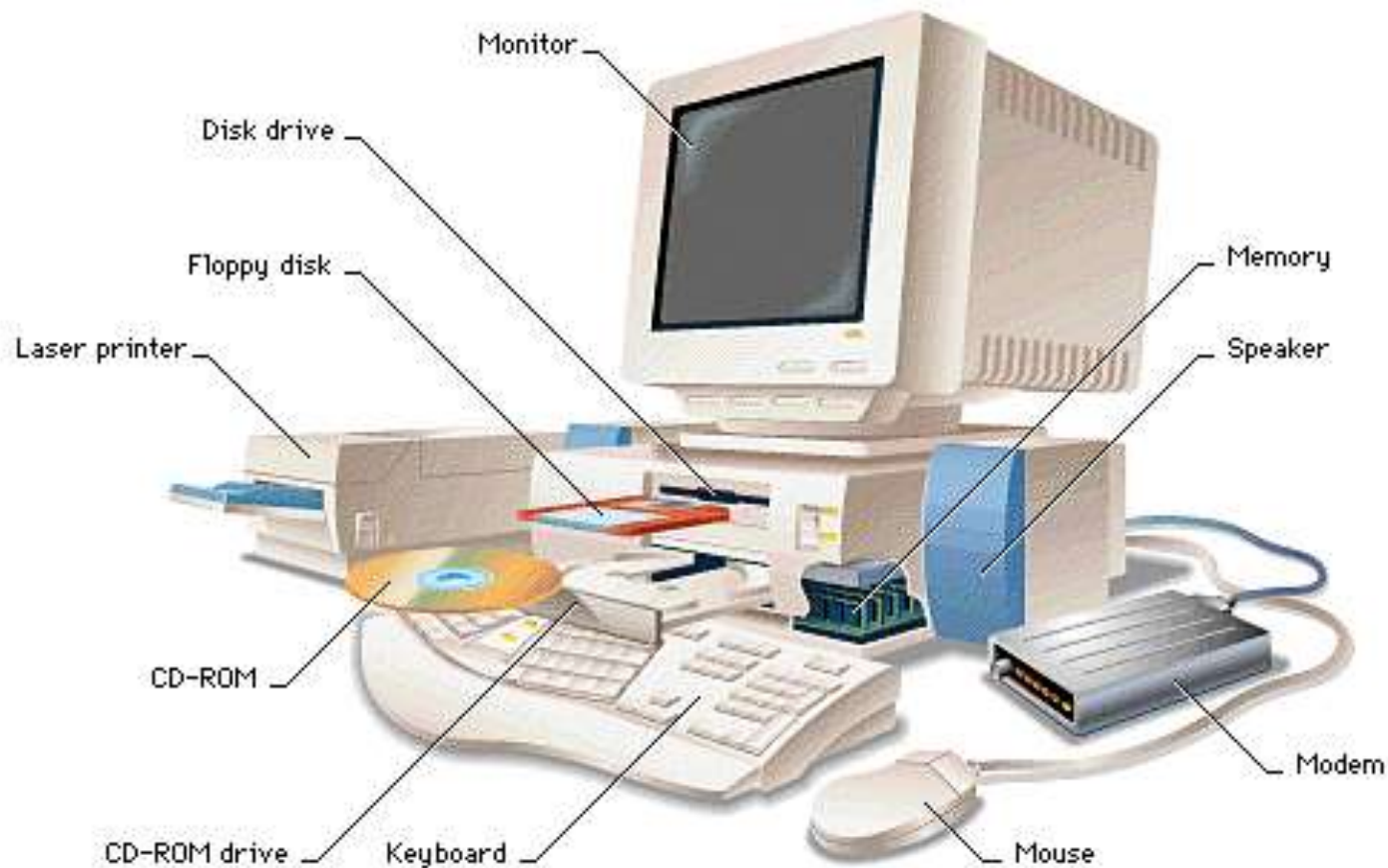
# Arranque (Startup) del Computador

---

- ▶ El **programa de arranque (bootstrap)** se carga al arrancar o reiniciar el sistema.
  - ▶ Típicamente almacena en la ROM o EPROM, generalmente conocido como **firmware**.
  - ▶ Inicializa todos los aspectos del sistema.
  - ▶ Carga el kernel del sistema operativo e inicia la ejecución.
- ▶ Memoria de solo lectura (ROM).
  - ▶ Almacena el BIOS → Ejecuta.
- ▶ BIOS.
  - ▶ Se realiza un chequeo básico del hardware de la máquina.
  - ▶ Accede al CMOS (Parámetros Generales p.e: Dispositivos Arrancables).
- ▶ Se carga el gestor de arranque.
  - ▶ ¿Cómo ubico al gestor de arranque?
- ▶ Gestor de Arranque (SO Loader).
  - ▶ Posición Fija (p.e: DD → cilindro 0, pista 0, sector 1)
- ▶ Tamaño reducido (512 bytes)
- ▶ Ejemplos
  - ▶ NTLDR, GRUB, LILO, etc.

# Organización del Sistema Computador

---

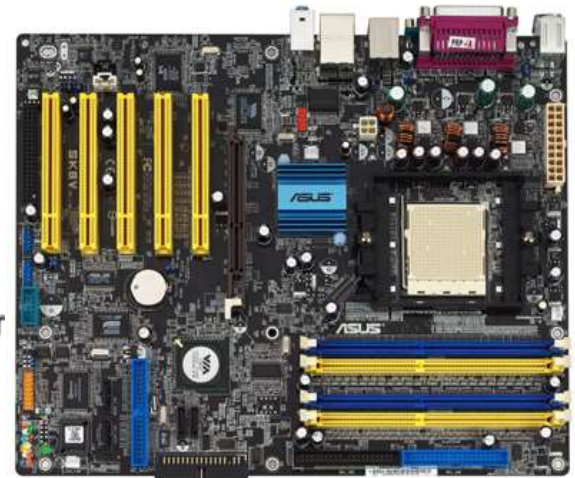
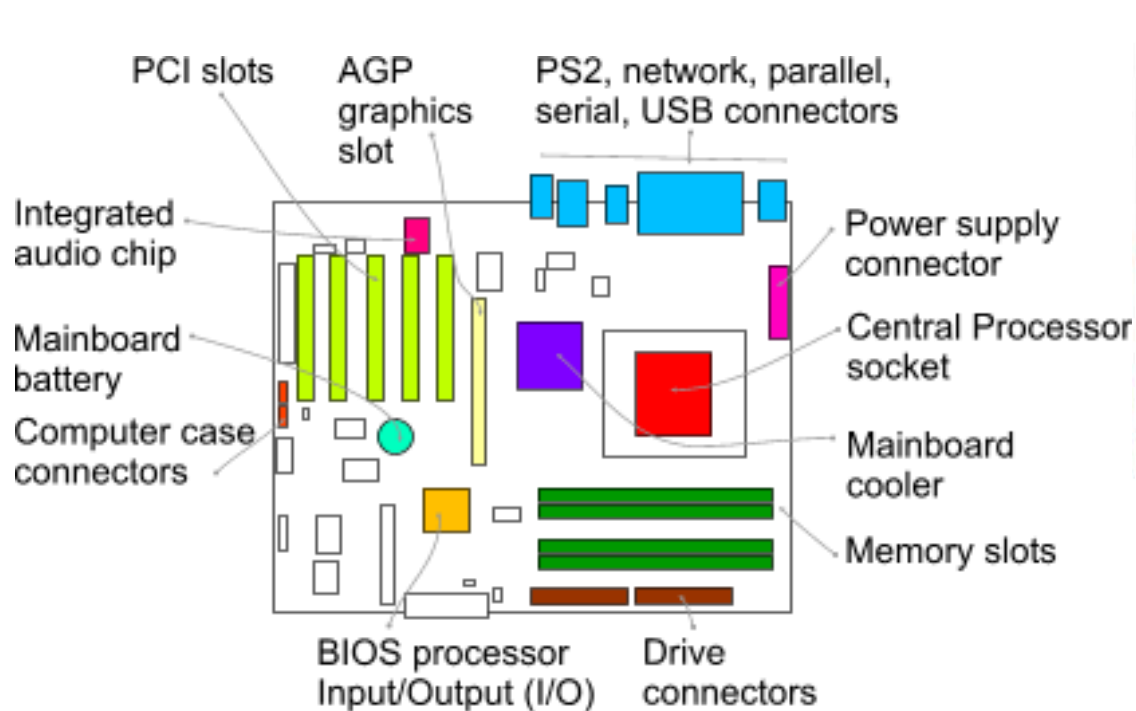


# Organización del Sistema Computador

---



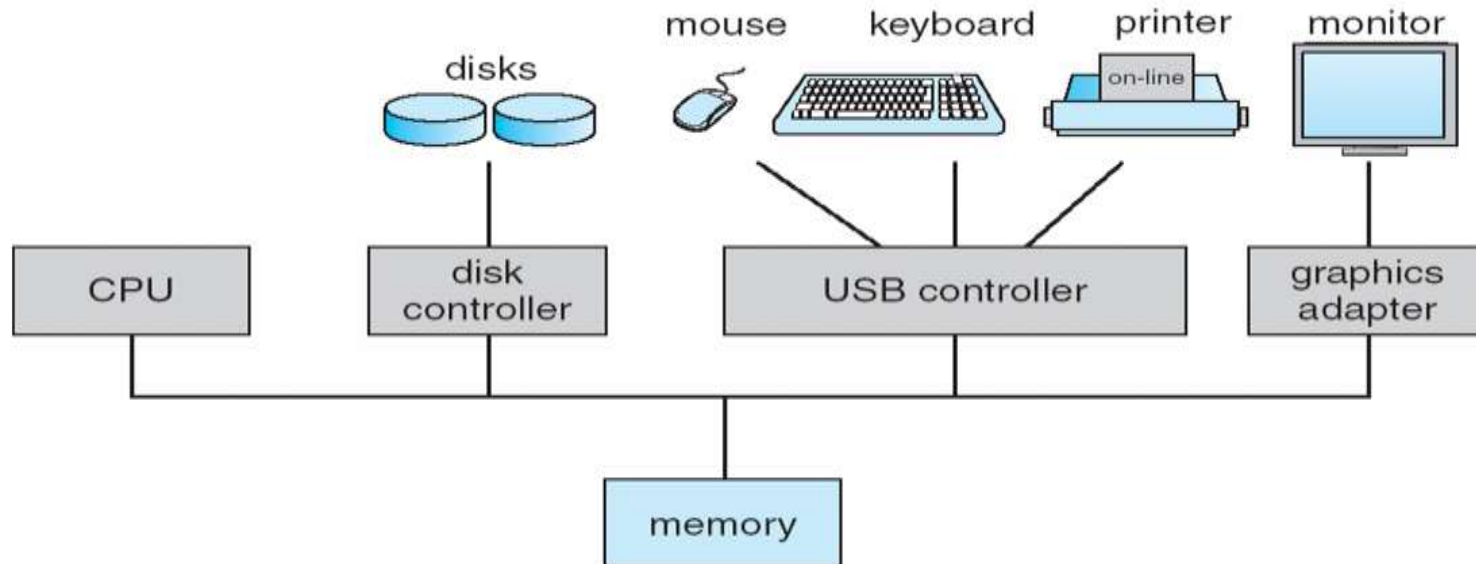
# Organización del Sistema Computador



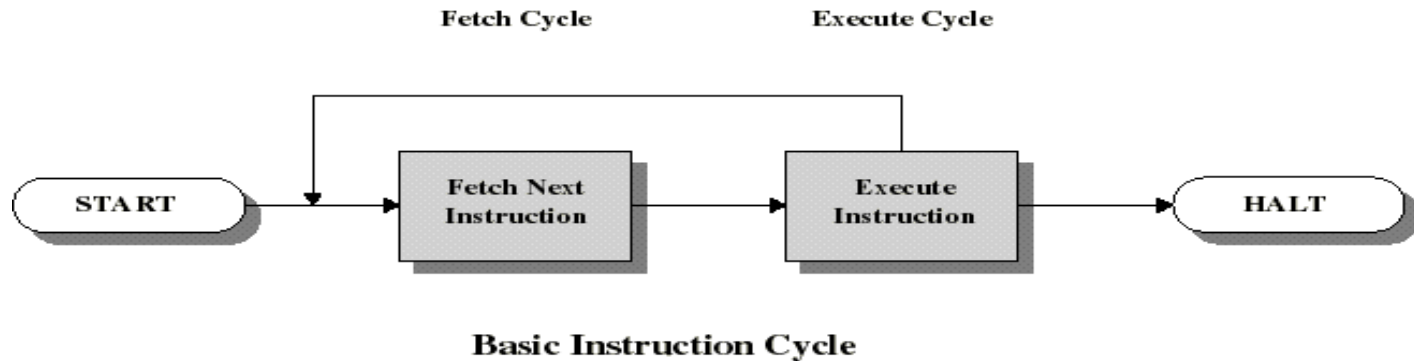
# Organización del Sistema Computador

## ► Operación del Sistema Computador

- Uno o más CPU y controladores de dispositivo se conectan a través del bus común que proporciona acceso a la memoria compartida.
- Ejecución concurrente de múltiples CPUs y dispositivos compitiendo por ciclos de memoria.



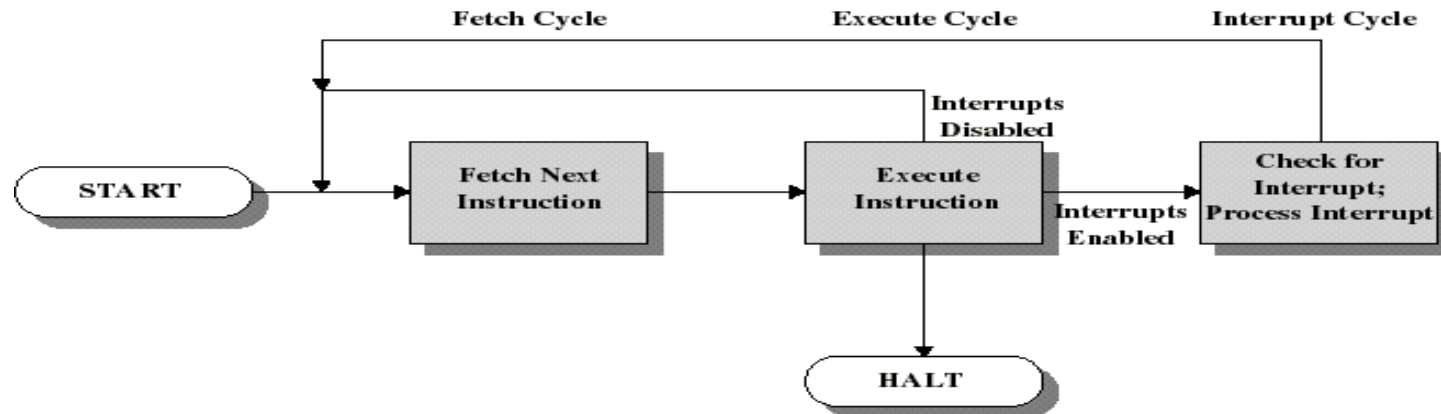
# Ciclo de Instrucción - Básico



- ▶ El CPU busca la próxima instrucción (con sus operandos) en la memoria.
- ▶ El CPU ejecuta la instrucción.
- ▶ El contador de programa (PC) mantiene la dirección de la próxima instrucción.
- ▶ El contador de programa se incrementa automáticamente después de cada búsqueda.



# Ciclo de Instrucción - Interrupciones



- ▶ El CPU verifica las solicitudes de interrupciones después de cada instrucción.
- ▶ Si no hay solicitudes de interrupción, entonces busca la próxima instrucción para el programa actual.
- ▶ Si una interrupción está pendiente, entonces suspende ejecución del programa actual, y ejecuta la rutina manejadora de la interrupción.

# Operación del Sistema Computador

---

- ▶ Los dispositivos de E/S y el CPU se puede ejecutar simultáneamente.
- ▶ Cada controlador de dispositivo está a cargo de un determinado tipo de dispositivo.
- ▶ Cada controlador de dispositivo tiene un buffer local.
- ▶ El CPU mueve datos desde/hacia la memoria principal a/desde los buffers locales.
- ▶ LA E/S es desde el dispositivo al búfer local del controlador.
- ▶ El controlador de dispositivo informa al CPU que ha terminado su operación, generando una **interrupción**.

# Manejador de interrupciones

---

- ▶ Es un programa que determina la naturaleza de la interrupción, realizando las acciones necesarias.
- ▶ El control se transfiere a este programa.
- ▶ El control debe transferirse de regreso al programa interrumpido, para que este pueda continuar desde el punto de interrupción.
- ▶ Este punto de interrupción puede ocurrir en cualquier parte en el programa.
- ▶ Por ello se debe salvar el estado del programa (contenido del PC + PSW + los registros +...).

# Funciones comunes de las interrupciones

---

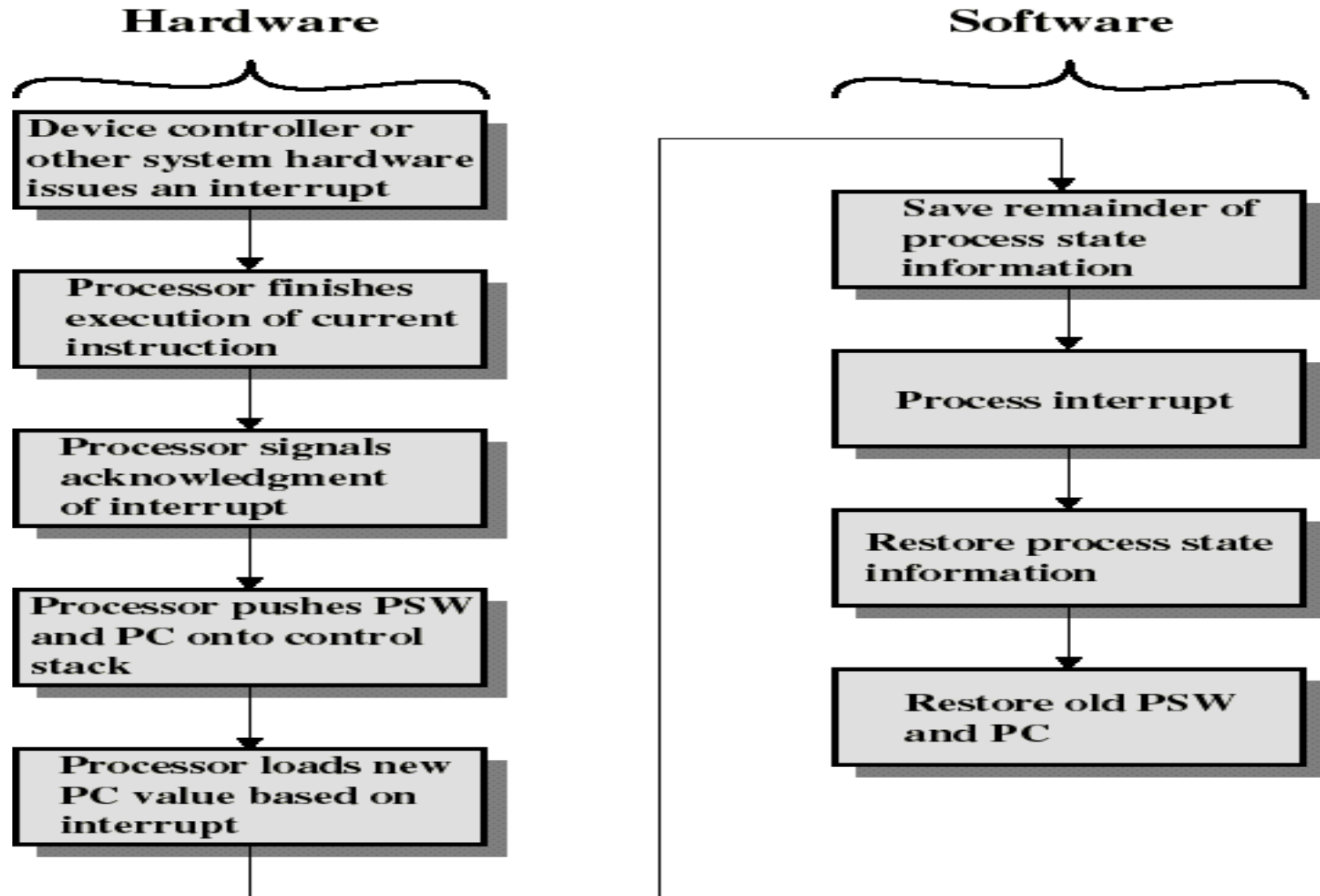
- ▶ Las interrupciones transfieren el control a la rutina manejadora de interrupciones generalmente, a través del **vector de interrupción**, que contiene las direcciones de todas las rutinas de tratamiento de las interrupciones.
- ▶ La arquitectura de interrupciones debe guardar la dirección de retorno de la instrucción interrumpida.
- ▶ Un **trap** o **excepción** es una interrupción generada por software, ya sea causada por un error o por un requerimiento del usuario.
- ▶ El sistema operativo es **manejado por interrupciones**.

# Manejo de las Interrupciones

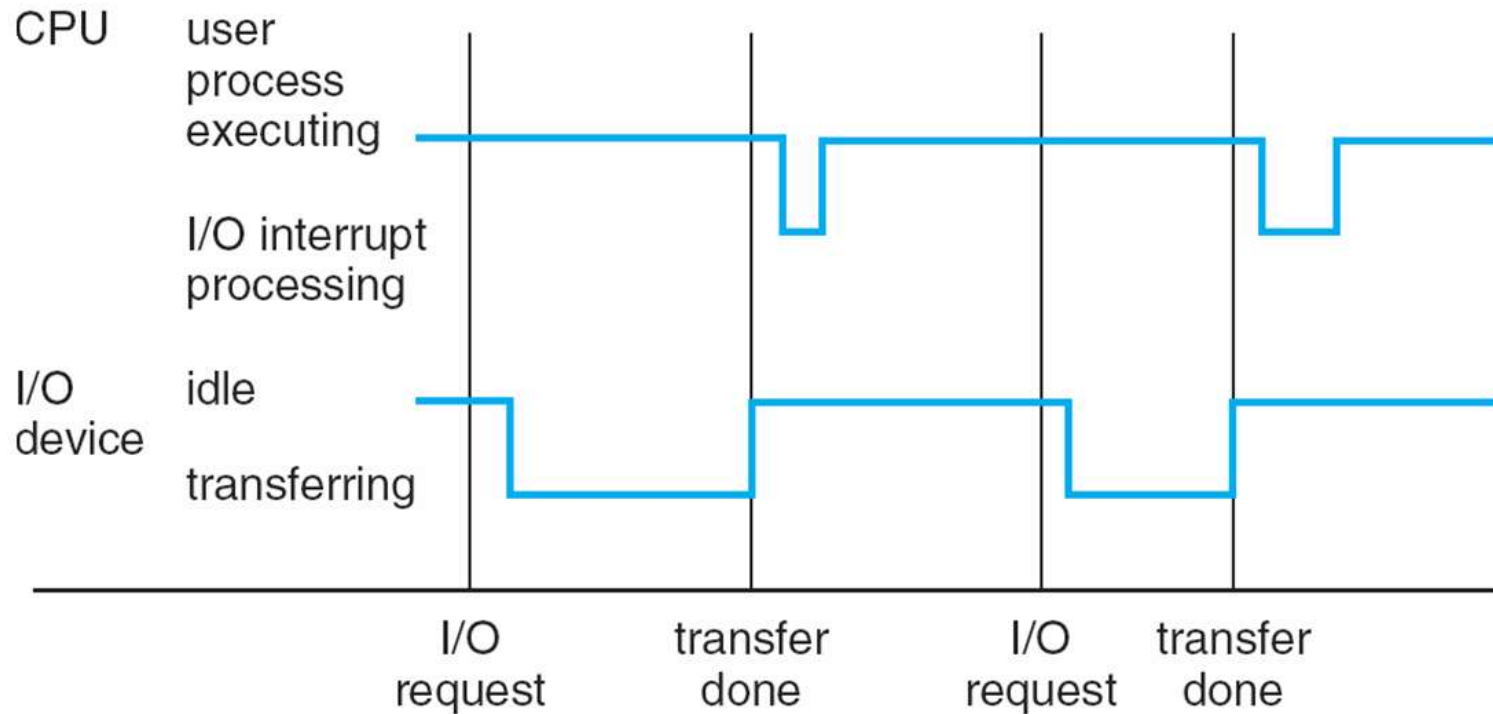
---

- ▶ El sistema operativo conserva el estado del CPU mediante el almacenamiento de los registros y el contador de programa (PC).
- ▶ Determina qué tipo de interrupción se ha producido:
  - ▶ Sondeo (**Polling**).
  - ▶ Sistema de Interrupciones **Vectorizadas**.
- ▶ Segmentos separados de código determinar qué acción se debe tomar para cada tipo de interrupción.

# Procesamiento de interrupciones



# Línea de tiempo de las Interrupciones





# Estructura de Operación de la E/S

---

## ▶ Sincrónica

- ▶ Después que la E/S comienza, el control retorna al programa de usuario sólo tras su finalización.
  - ▶ Se genera una instrucción de Wait, para colocar el CPU en espera hasta la siguiente interrupción.
  - ▶ Se entra en un ciclo de espera (contención por el acceso a memoria).
  - ▶ A lo sumo sólo una solicitud de E/S está pendiente a la vez , no existe un procesamiento simultáneo de operaciones de E/S.

# Estructura de Operación de la E/S

---

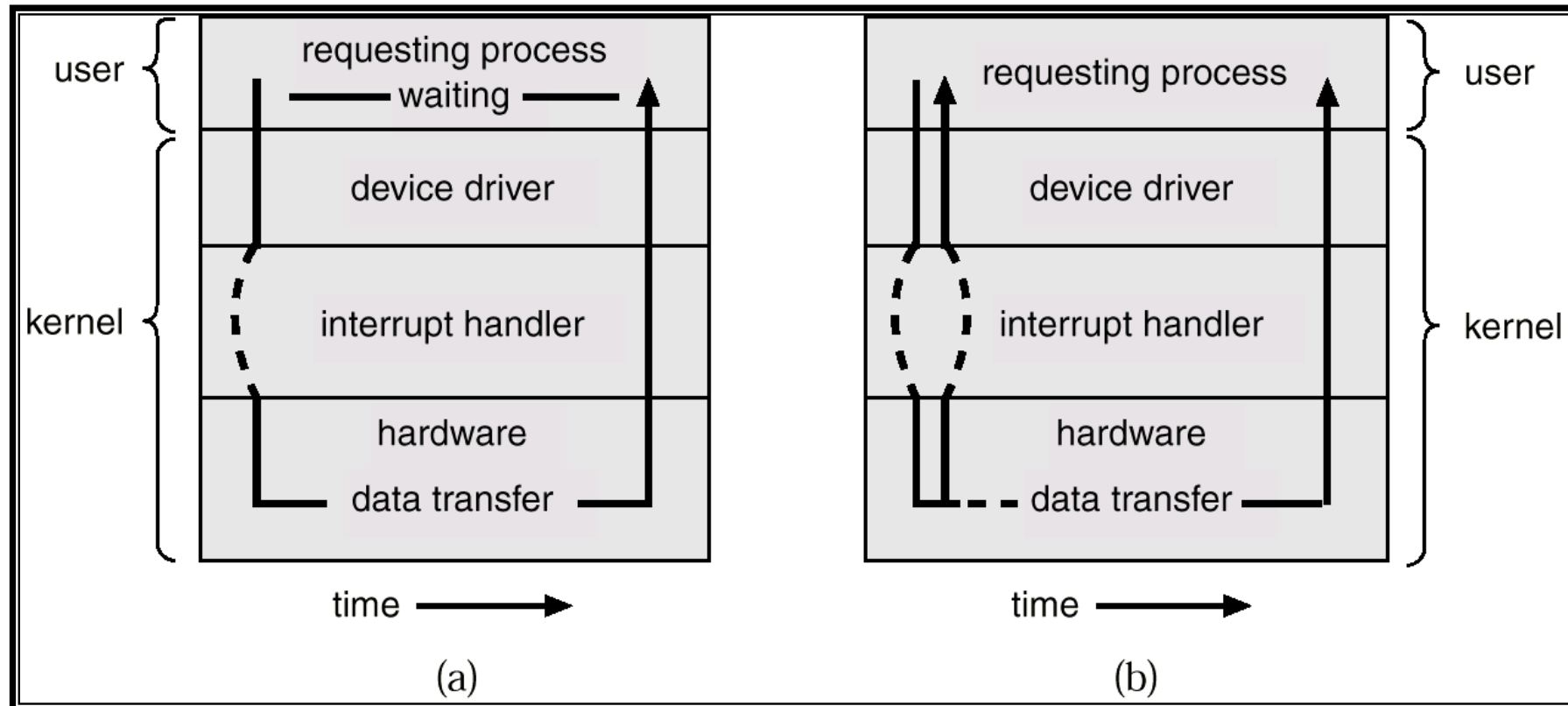
## ▶ Asincrónica

- ▶ Después que la operación de E/S comienza, el control vuelve al programa de usuario sin esperar a la finalización de la misma
  - ▶ Llamada al Sistema - requerimiento al sistema operativo para permitir al usuario esperar por la terminación de la E/S.
  - ▶ La tabla de estado de Dispositivos, contiene una entrada para cada dispositivo de E/S indicando: su tipo, dirección y estado.
  - ▶ El SO indexa en dicha tabla para determinar el estado del dispositivo y modificar la entrada de la tabla para incluir la interrupción.

# Estructura de Operación de la E/S

Sincrónica

Asincrónica



# Estructura de DMA

## (Acceso Directo a Memoria)

---

- ▶ Utilizado para los dispositivos de E/S de alta velocidad, capaces de transmitir la información a velocidades cercanas a la de la memoria.
- ▶ El controlador del dispositivo transfiere bloques de datos desde el buffer del almacenamiento directamente a la memoria principal, sin intervención del CPU.
- ▶ Sólo se genera una interrupción por bloque, en lugar de una interrupción por cada byte.

# Estructura de Almacenamiento

---

- ▶ Memoria principal - sólo almacenamiento de gran tamaño que el CPU puede acceder directamente.
  - ▶ **Acceso Aleatorio**
  - ▶ Normalmente **volátil**
- ▶ Almacenamiento secundario - extensión de la memoria principal que proporciona gran capacidad de almacenamiento **no volátil**.

# Estructura de Almacenamiento

---

- ▶ Discos magnéticos - platos rígidos de metal o de vidrio cubiertos con material de grabación magnética.
  - ▶ La superficie del disco se divide lógicamente en las **pistas**, que se subdividen en **sectores**.
  - ▶ El **controlador de disco** determina la interacción lógica entre el dispositivo y el computador.
- ▶ **Discos de estado sólido** - más rápidos que los discos magnéticos, no volátil.
  - ▶ Varias tecnologías
  - ▶ Cada vez más popular

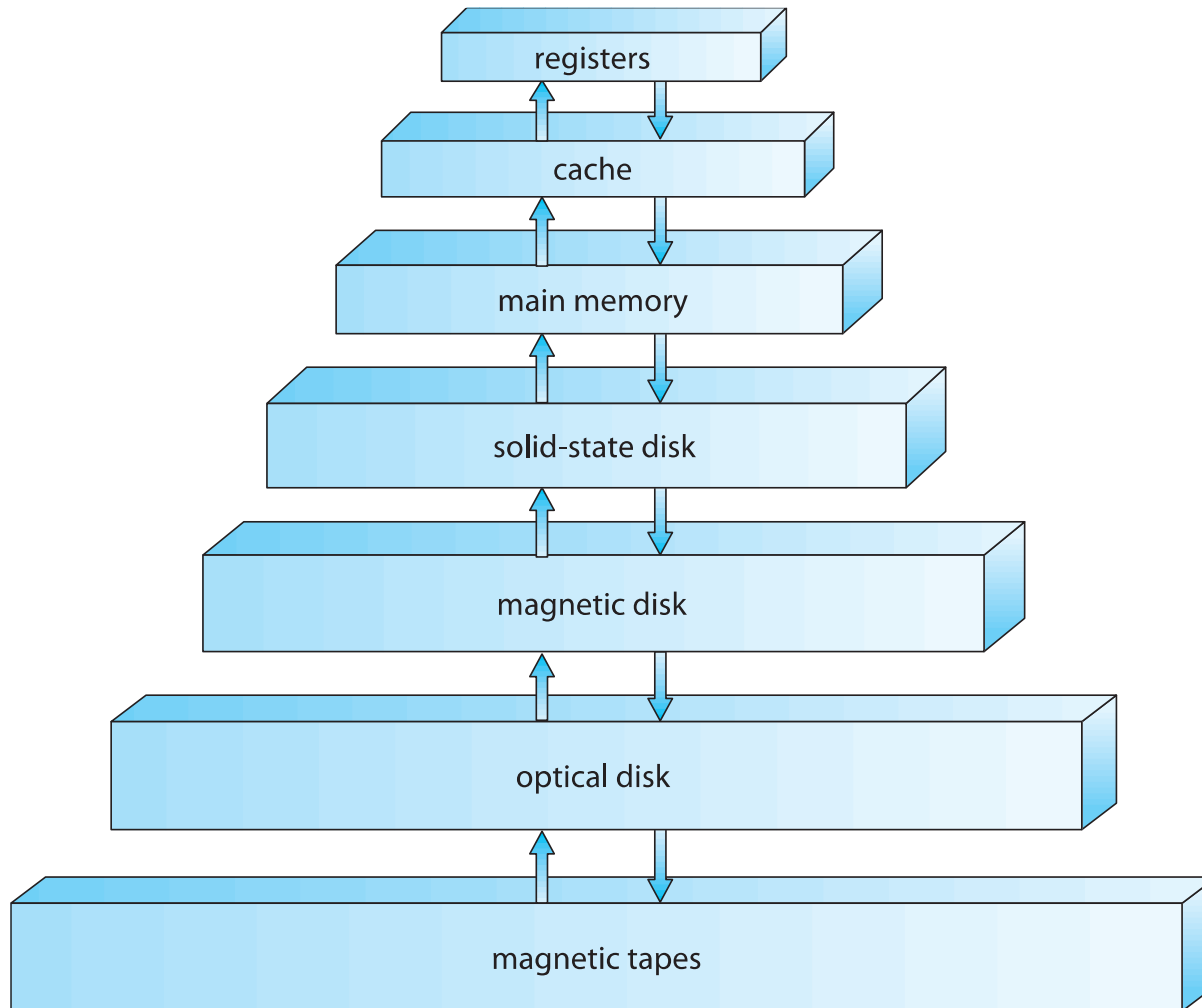
# Jerarquía de Almacenamiento

---

- ▶ El sistema de almacenamiento esta organizado en una jerarquía:
  - ▶ Velocidad.
  - ▶ Costo.
  - ▶ Volatilidad.
- ▶ **Caching** - copia información en un sistema de almacenamiento más rápido; la memoria principal se puede ver como una memoria caché para el almacenamiento secundario.
- ▶ Un **Manejador de dispositivo** para cada controlador de dispositivo administre su E/S.
  - ▶ Proporciona una interfaz uniforme entre el controlador y el kernel.

# Jerarquía de Dispositivos de Almacenamiento

---



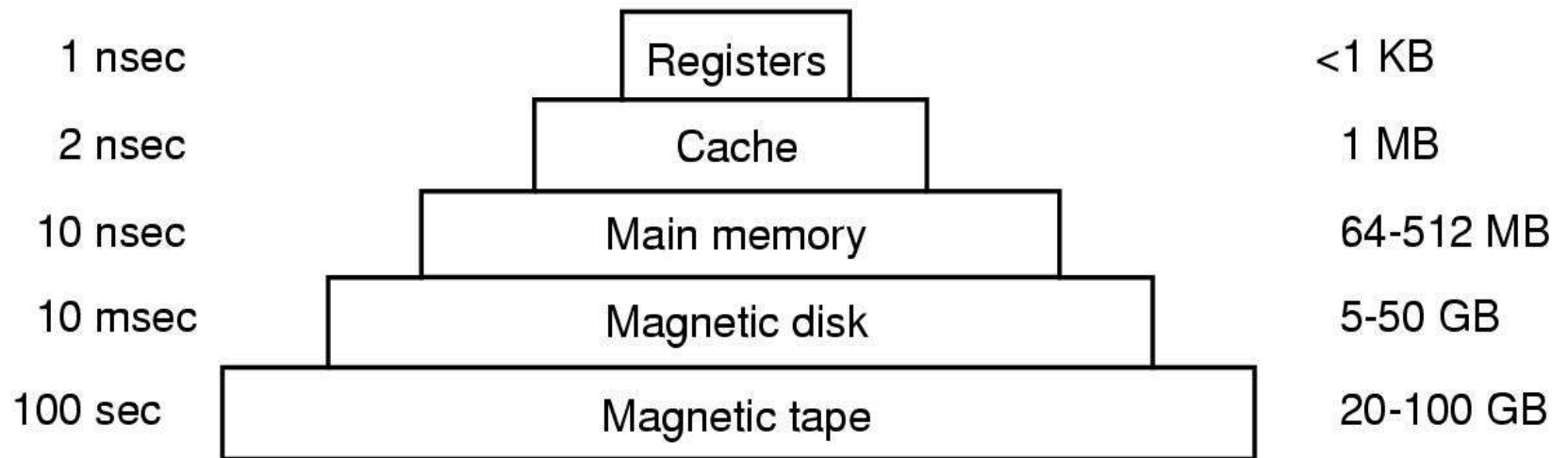


# Jerarquía de Almacenamiento

---

Typical access time

Typical capacity



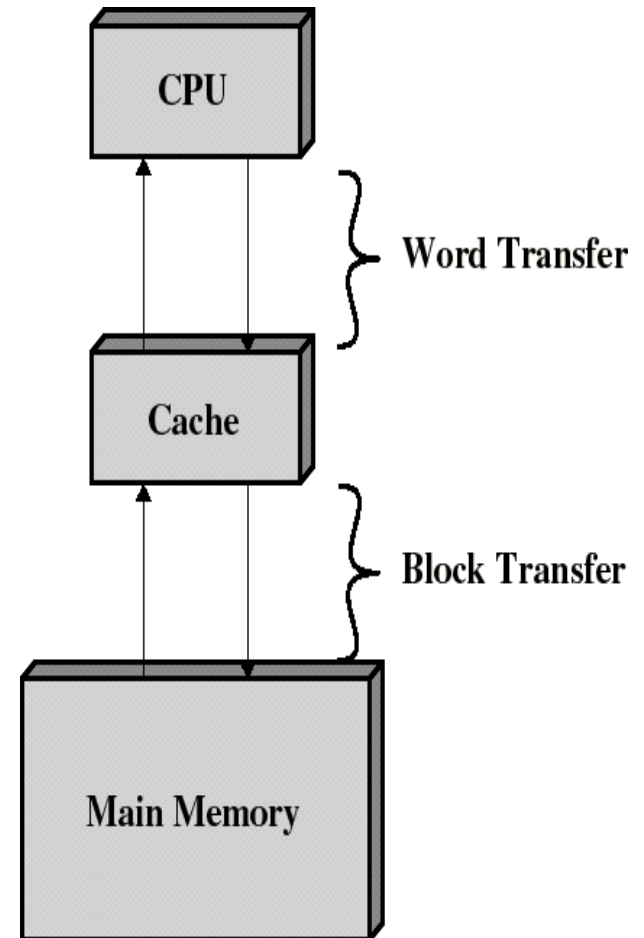
# Memoria Cache

---

- ▶ Principio importante, utilizado en muchos niveles en un computador (hardware, sistema operativo, software).
- ▶ La memoria caché es más pequeña que el almacenamiento al que sirve.
  - ▶ La gestión de la memoria caché es un problema de importante de diseño.
  - ▶ El tamaño de la memoria caché y la política de sustitución también son tópicos importantes

# Memoria Cache

- ▶ Pequeñas memorias caras pero muy rápidas que interactúan con otra memoria más lenta pero mucho más grande.
- ▶ Invisible al SO y el programa de usuario, pero interactúa con otro hardware de manejo de memoria.
- ▶ El procesador verifica primero si la palabra referenciada está en el cache.
- ▶ Si no se encontró en el cache, un bloque de memoria que contiene la palabra se mueve al cache.



# Arquitectura del Computador

---

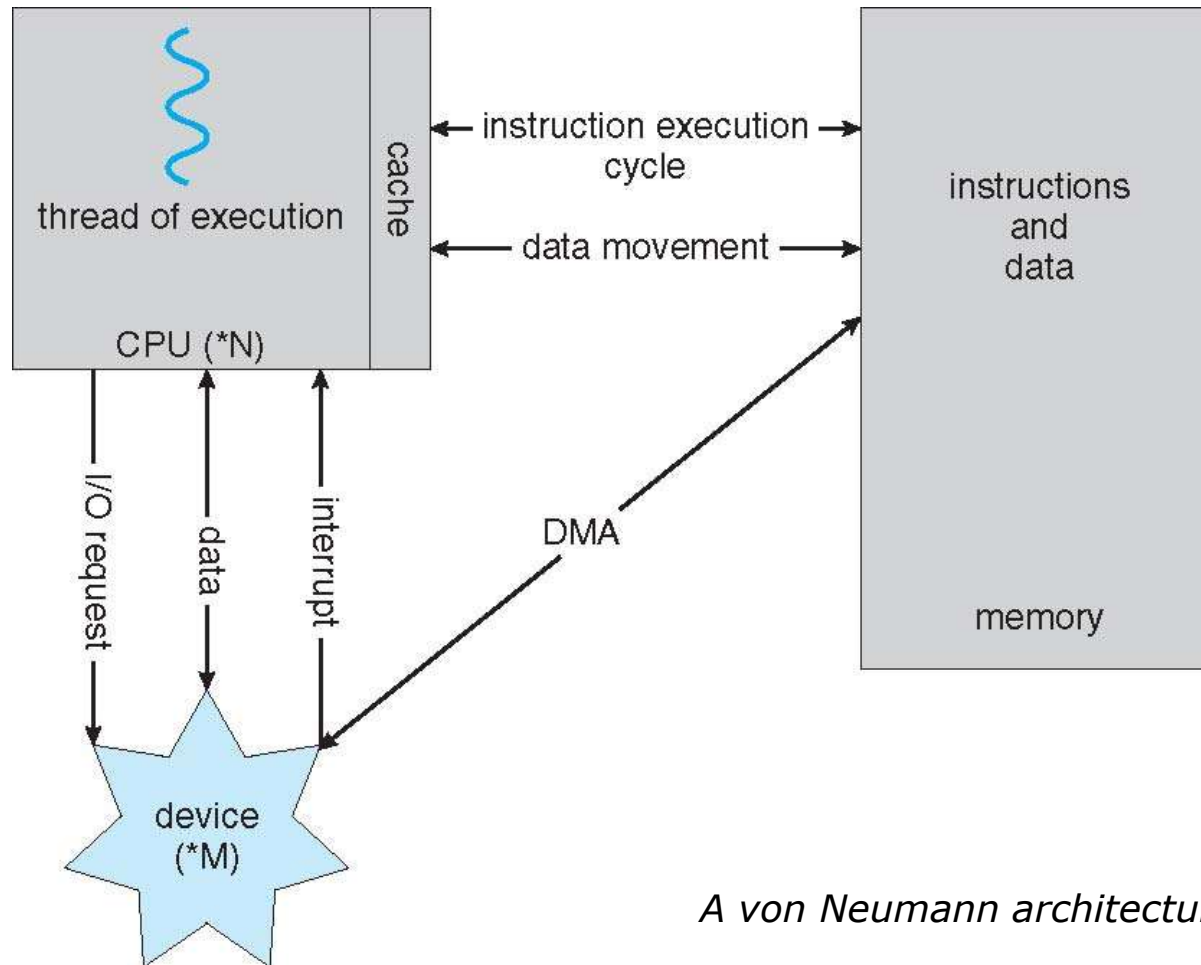
- ▶ La mayoría de los sistemas usan un solo procesador de propósito general (desde PDAs hasta mainframes).
- ▶ La mayoría de los sistemas cuentan también con procesadores de propósito específico.

# Arquitectura del Computador

---

- ▶ Los **Sistemas Multiprocesadores** han crecido en uso e importancia.
  - ▶ Son también conocidos como sistemas paralelos o sistemas estrechamente acoplados.
  - ▶ Las ventajas incluyen:
    1. Mayor rendimiento
    2. Economía de escala
    3. Mayor fiabilidad - degradación progresiva o la tolerancia a fallos
  - ▶ Dos tipos:
    1. Multiprocesamiento Asimétrico
    2. Multiprocesamiento Simétrico

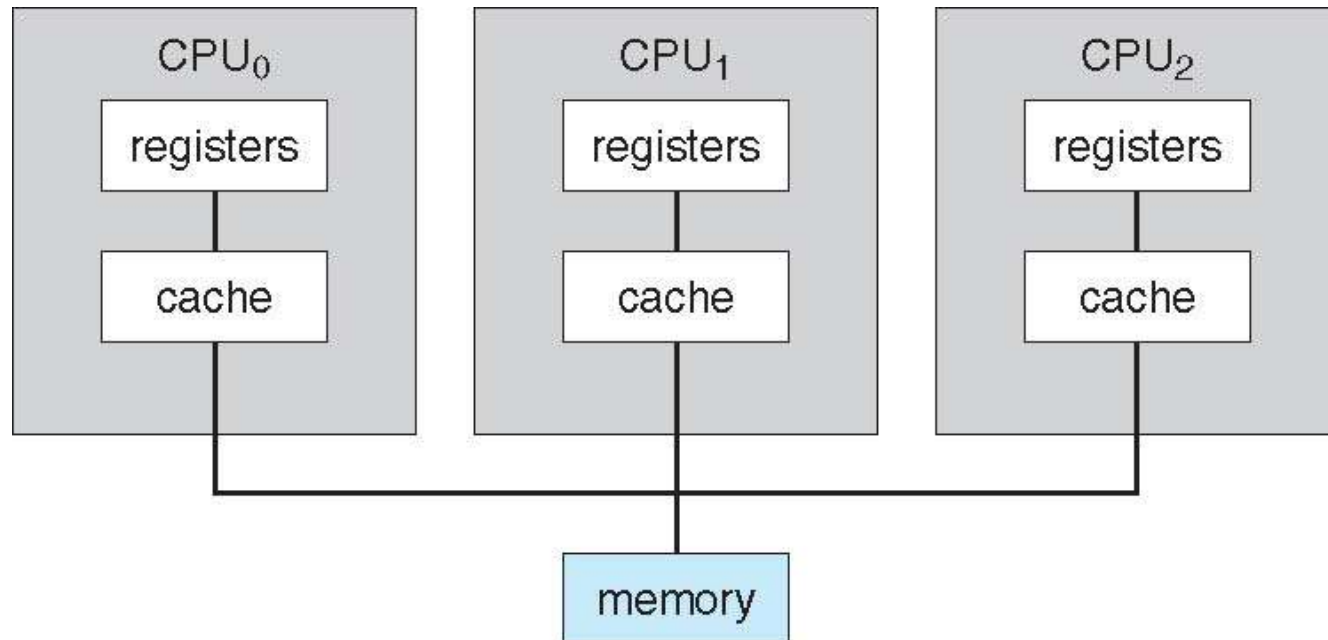
# Funcionamiento de un Computador Moderno



*A von Neumann architecture*

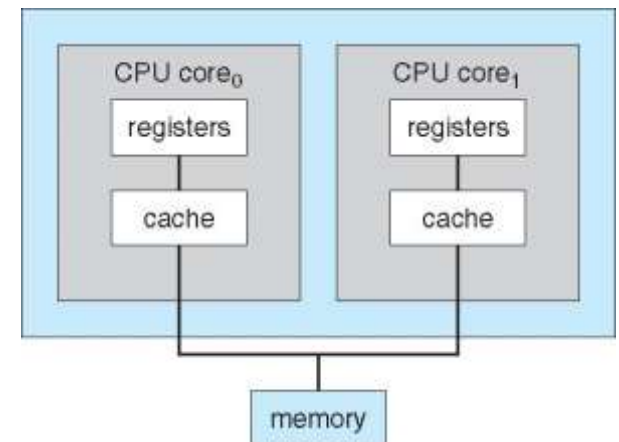
# Arquitectura de Multiprocesamiento Simétrico

---



# Un Diseño Dual-Core (Doble Núcleo )

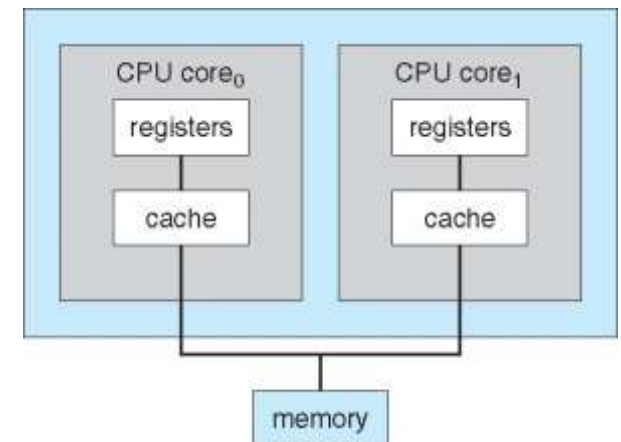
- ▶ **Variaciones de las Arquitecturas **UMA** y **NUMA**.**
- ▶ **Múltiples chips y múltiples núcleos (**multicore**).**
- ▶ **Sistemas todo en un chip (**SoC**) vs **servidores blade**.**
  - ▶ **Chasis que contienen múltiples Sistemas Separados.**





# Un Diseño Dual-Core (Doble Núcleo )

- ▶ **Variaciones de las Arquitecturas **UMA** y **NUMA**.**
- ▶ **Múltiples chips y múltiples núcleos (**multicore**).**
- ▶ **Sistemas todo en un chip (**SoC**) vs **servidores blade**.**
  - ▶ **Chasis que contienen múltiples Sistemas Separados.**



# Sistemas en Clúster

---

- ▶ Parecidos a los sistemas multiprocesador, pero se trata de múltiples sistemas trabajando juntos.
- ▶ Por lo general, comparten el almacenamiento a través de una **red de área de almacenamiento (SAN)**.
- ▶ Proveen un servicio de **alta disponibilidad** que es resistente a fallas.
  - ▶ **Clustering asimétrico**, en el cuál una máquina esta en modo de “espera en caliente” (hot-standby).
  - ▶ **Clustering simétrico**, todos los nodos ejecutan varias aplicaciones, monitoreándose entre sí.

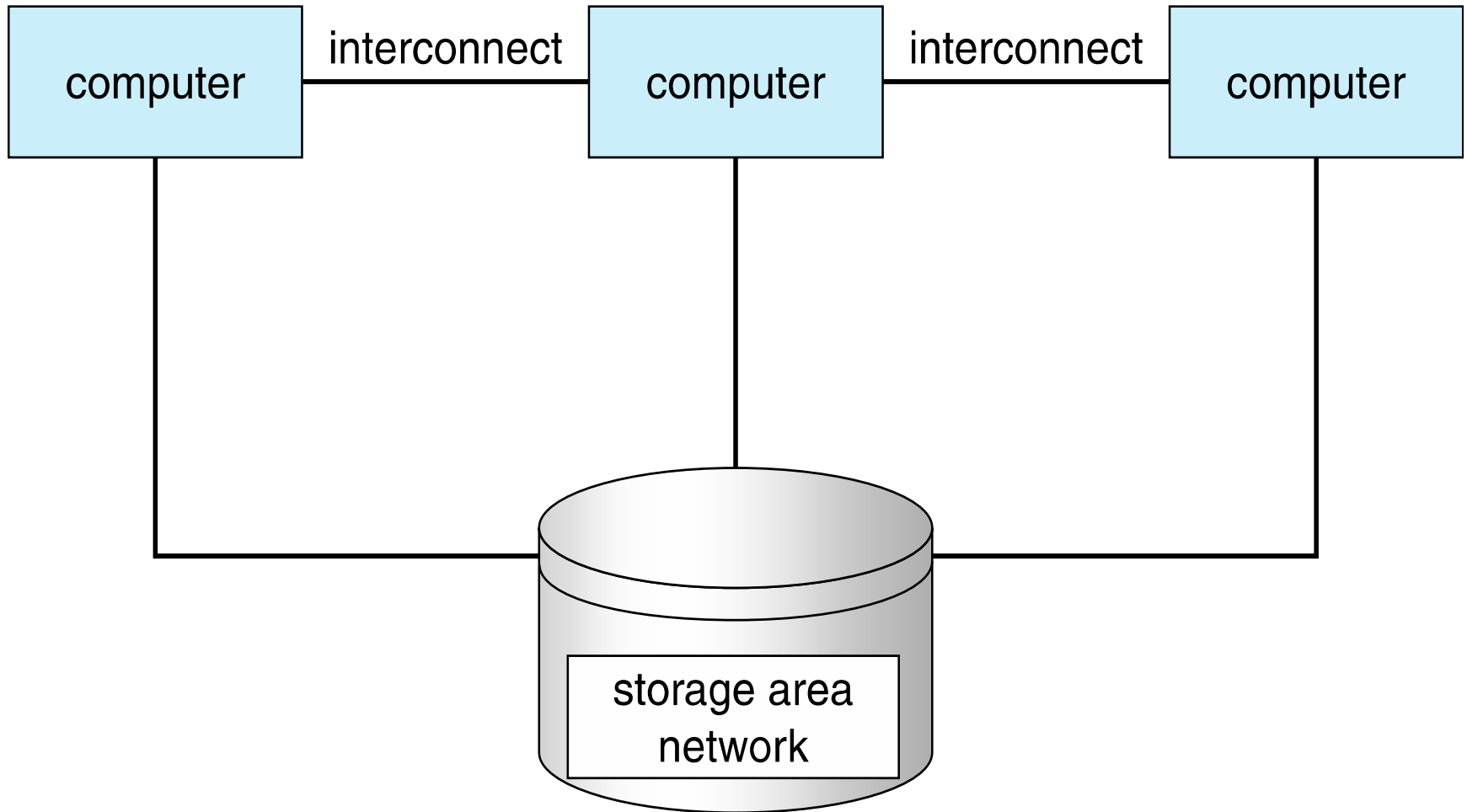
# Sistemas en Clúster

---

- ▶ Parecidos a los sistemas multiprocesador, pero se trata de múltiples sistemas trabajando juntos.
- ▶ Algunos Clústers son para **computación de alto rendimiento (HPC)**.
  - ▶ Las aplicaciones deben ser por escritas para usar **paralelización**.
- ▶ Algunos tienen un gestión distribuida de bloqueos (**Distributed Lock Manager, DLM**) para evitar operaciones conflictivas.

# Sistemas en Clúster

---



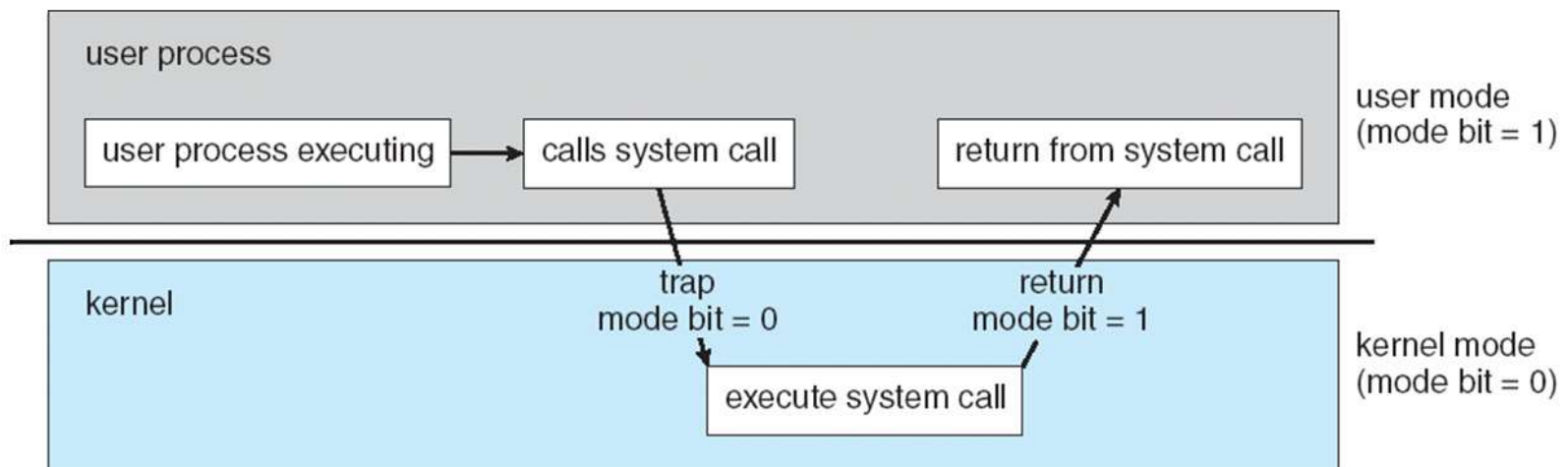
# Operación del Sistema Operativo

---

- ▶ **Manejo de Interrupciones** por hardware.
- ▶ Se crean **excepciones** o **traps**, por errores de software o por requerimientos.
  - ▶ La división por cero, la solicitud de servicios del sistema operativo.
- ▶ Otros problemas de los procesos incluyen los ciclos infinitos, la modificación de los procesos entre sí o al sistema operativo.
- ▶ La operación en **Modo Dual**, permite al SO protegerse a sí mismo y a otros componentes del sistema.
  - ▶ **Modo de usuario** y **modo kernel**.
  - ▶ **Bit de Modo** proporcionado por hardware.
    - ▶ Proporciona la capacidad de distinguir cuando el sistema está ejecutando código de usuario o código del kernel.
    - ▶ Algunas instrucciones designadas como privilegiadas, sólo son ejecutables en modo kernel.
    - ▶ Una Llamada al Sistema cambiará a modo kernel, el retorno de la llamada restaura el modo usuario.
- ▶ Cada vez más, los CPUs soportan operaciones multi-modo
  - ▶ Por ejemplo, modo gestión de máquina virtual (**virtual machine manager, VMM**), para máquinas virtuales en el sistema.

# Transición de Modo User a Modo Kernel

- ▶ Un Temporizador previene ciclos infinitos y/o de que un proceso acapare los recursos.
  - ▶ Se coloca la interrupción para un periodo de tiempo específico.
  - ▶ El SO decrementa el contador.
  - ▶ Cuando el contador es cero, se genera una interrupción.
  - ▶ Se coloca antes de Planificar el proceso para recuperar el control o dar por terminado el programa que excede el tiempo asignado.



# Entornos de Computación – Traditional

---

- ▶ Concebidos como Sistemas Aislados de Máquinas de propósito general
- ▶ Sin embargo, desdibujado ya que la mayoría de los sistemas se interconectan con otros sistemas (por ejemplo, la Internet)
- ▶ **Portales web** proporcionan acceso a los sistemas internos
- ▶ **Las computadoras de red (clientes ligeros)** son como terminales web
- ▶ Las computadoras se interconectan móviles a través de **redes inalámbricas**
- ▶ Las Redes se están convirtiendo en omnipresentes - incluso los sistemas caseros utilizan **firewalls** para proteger a los computadores personales de ataques desde Internet

# Entornos de Computación – Móviles

---

- ▶ Teléfonos inteligentes portátiles, tabletas, etc
- ▶ ¿Cuál es la diferencia funcional entre ellos y un ordenador portátil "tradicional"?
- ▶ Características adicionales - más funciones del SO (GPS, giroscopio)
- ▶ Permite los nuevos tipos de aplicaciones tales como la **realidad aumentada**  
Usa el estándar IEEE 802.11 inalámbrico o redes de datos móviles para la conectividad  
Los líderes son **iOS de Apple** y **Android de Google**



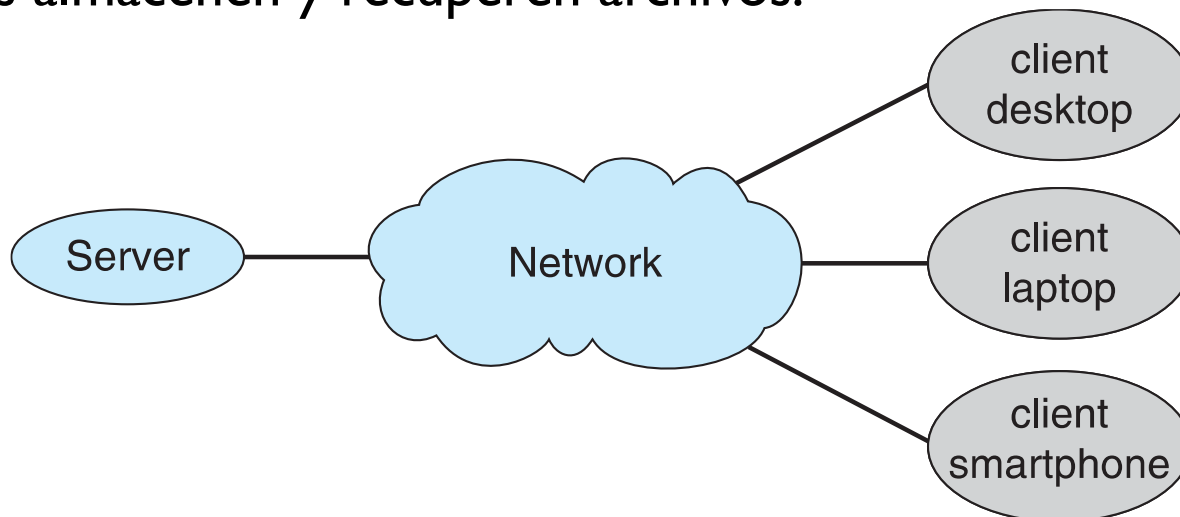
# Entornos de Computación – Distribuido

---

- ▶ Colección de sistemas separados, posiblemente heterogéneos, conectados en red
- ▶ **La red** es una ruta de comunicaciones, **TCP/IP** es el protocolo más común
  - ▶ Red de área local (LAN)
  - ▶ Red de área amplia (WAN)
  - ▶ Red de área Metropolitana (MAN)
  - ▶ Red de área personal (PAN)
- ▶ El **Sistema operativo de red** proporciona funciones entre los sistemas a lo largo de la red
  - ▶ El esquema de comunicación permite a los sistemas el intercambio de mensajes
  - ▶ Se crea la ilusión de un sistema único

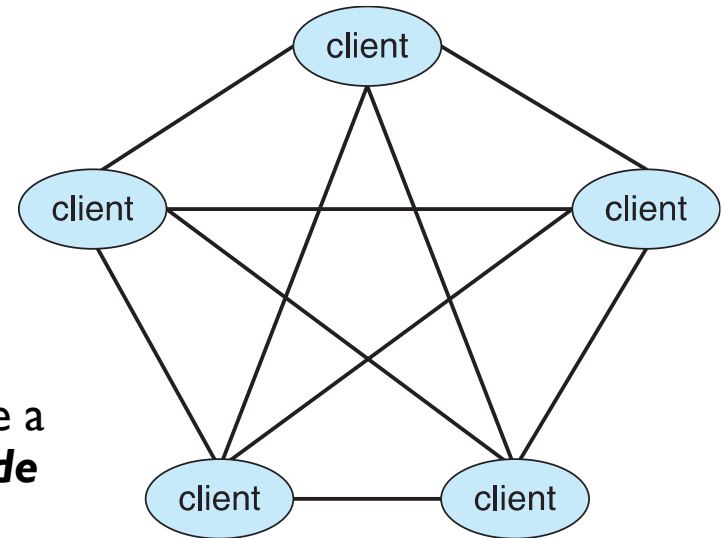
# Entornos de Computación – Cliente-Servidor

- ▶ Terminales tontos son sustituidos por PCs inteligentes.
- ▶ Muchos sistemas, ahora **Servidores**, responden a los requerimientos generados por los clientes.
- ▶ **Servidores de Procesamiento**, proporciona una interfaz con el cliente para solicitar servicios (p.e., base de datos)
- ▶ **Servidores de Archivos**, proporciona una interfaz para que los clientes almacenen y recuperen archivos.



# Entornos de Computación – Redes de Iguales (Peer-to-Peer, P2P)

- ▶ Otro modelo de sistema distribuido
- ▶ P2P no distingue los clientes y servidores
  - ▶ En su lugar todos los nodos se consideran iguales
  - ▶ Pueden ser que cada uno actúen como cliente, servidor o ambos
  - ▶ El Nodo debe unirse a la red P2P
    - ▶ Registra su servicio con el servicio de búsqueda Central en la red, o
    - ▶ Difunde una petición de servicio y responde a las solicitudes de servicio vía el **protocolo de descubrimiento**
- ▶ Los ejemplos incluyen Napster, Torrents y Gnutella, **voz sobre IP (VoIP)** como Skype



# Entornos de Computación – Virtualización

---

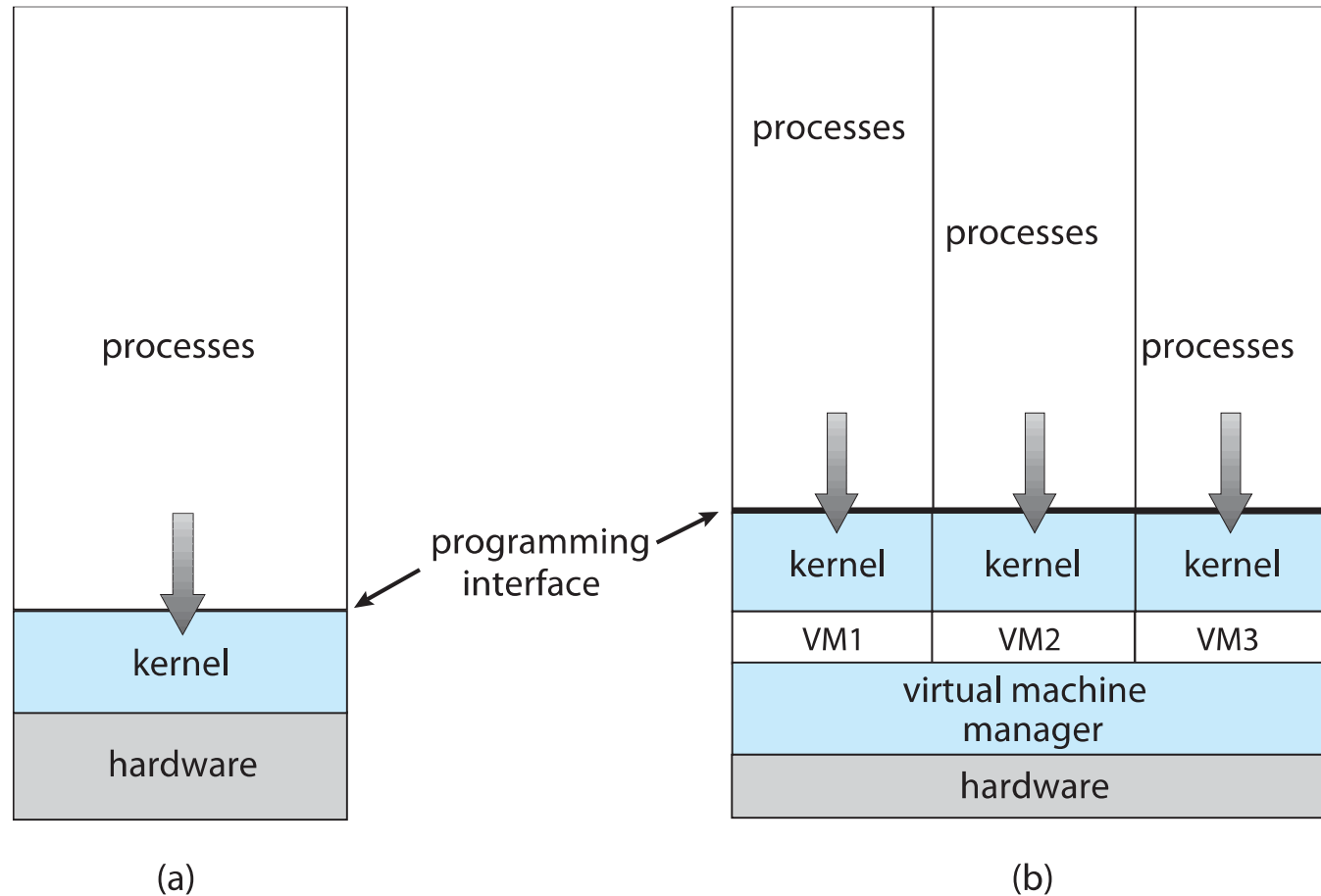
- ▶ Permite a los sistemas operativos correr aplicaciones dentro de otros SOs
  - ▶ Industria enorme y en crecimiento
- ▶ La **Emulación** se utiliza cuando el tipo de CPU fuente es diferente al tipo de CPU destino (p.e., PowerPC a Intel x86)
  - ▶ Generalmente, es el método más lento
  - ▶ Cuando el lenguaje del computador no es compilado a código fuente, se trata de una – **Interpretación**
- ▶ **Virtualización** – Un SO, compilado para el CPU, corre SOs **huéspedes** también compilados para ese CPU
  - ▶ Considere VMware, corriendo múltiples **huéspedes** WinXP, cada uno ejecutando aplicaciones y todas ellas sobre un **anfitrión** SO WinXP
- ▶ **VMM** proporciona servicios de virtualización

# Entornos de Computación – Virtualización

---

- ▶ Los casos de uso involucran laptops y desktops que ejecutan múltiples sistemas operativos para exploración o compatibilidad
  - ▶ Portátil Apple con Mac OS X como anfitrión, Windows como huésped
  - ▶ Permite el desarrollo de aplicaciones para múltiples sistemas operativos sin necesidad de tener múltiples sistemas
  - ▶ Pruebas de Control de Calidad de las aplicaciones sin necesidad de tener múltiples sistemas
  - ▶ Ejecución y administración de entornos de cómputo en los centros de datos
- ▶ VMM se puede ejecutar de forma nativa, en cuyo caso son también el anfitrión
  - ▶ Si no hay ningún anfitrión de propósito general entonces (VMware ESX y Citrix XenServer)

# Entornos de Computación – Virtualización



# Entornos de Computación – Computación en Nube (Cloud Computing)

---

- ▶ Proporciona procesamiento, almacenamiento, incluso aplicaciones como un servicio a través de una red
- ▶ Extensión lógica de la virtualización debido a que esta basado en la virtualización
  - ▶ Amazon **EC2** tiene miles de servidores, millones de máquinas virtuales, PetaBytes de almacenamiento disponible a través de Internet, pagados en función del uso

# Entornos de Computación – Computación en Nube (Cloud Computing)

---

## ► Modelos de Despliegue

- **Nube Pública** - disponible vía Internet a cualquiera que esté dispuesto a pagar
- **Nube Comunitaria** - compartida entre un conjunto de empresas/comunidades bajo algún tipo de convenio
- **Nube Privada** - ejecutado por una empresa para uso propio de la empresa
- **Nube Híbrida** - incluye componentes de nubes públicas y privadas



# Entornos de Computación – Computación en Nube (Cloud Computing)

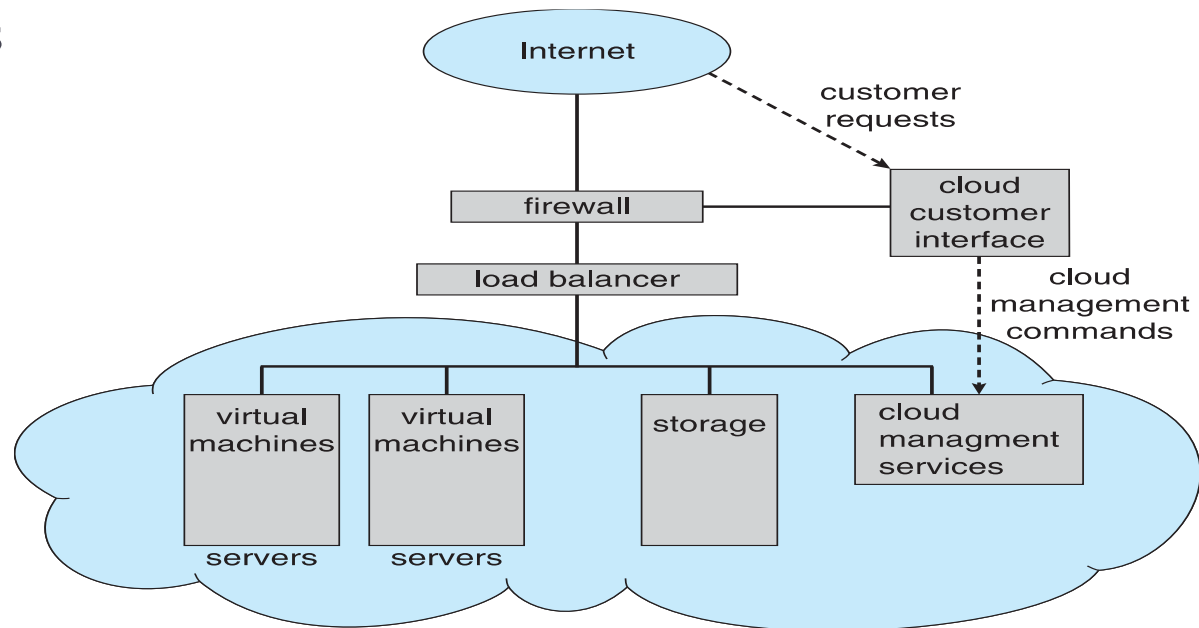
---

## ► Modelos de Servicio

- Software como Servicio (**SaaS**) - una o más aplicaciones disponibles a través de Internet (p.e., procesador de palabras)
- Plataforma como servicio (**PaaS**) - pila de software listo para el uso de aplicaciones a través de Internet (p.e., un servidor de base de datos)
- Infraestructura como Servicio (**IaaS**) - servidores o almacenamiento disponible a través de Internet (p.e., almacenamiento disponibles para el uso como respaldo)

# Entornos de Computación – Computación en Nube (Cloud Computing)

- ▶ El entorno de Computación en Nube se compone de sistemas operativos tradicionales, además de VMM, además de herramientas de gestión de la nube
- ▶ Conectividad a Internet requiere de seguridad como firewalls
- ▶ Balanceadores de carga distribuyen el tráfico a través de las aplicaciones



# Entornos de Computación – Computación en Nube (Cloud Computing)

Representación visual de la definición de la NIST del cloud computing  
<http://www.csrc.nist.gov/groups/SNS/cloud-computing/index.html>



# Servicios del Sistema Operativo

---

- ▶ El Sistema Operativo provee un ambiente de ejecución para programas, además de servicios para dichos programas y usuarios
- ▶ Un conjunto de servicios del SO provee funciones de utilidad a los usuario:
  - ▶ Interfaz de Usuario - UI (User Interfaces)
    - ▶ Varían entre **Command-Line (CLI)**, **Graphics User Interfaces (GUI)**, **Batch**
  - ▶ Ejecución de programas – El sistema debe ser capaz de cargar el programa en memoria para su posterior ejecución, y terminar su ejecución, bien sea de manera normal o anormal (indicando la condición de error)

# Servicios del Sistema Operativo

---

- ▶ Un conjunto de servicios del SO provee funciones de utilidad a los usuarios:
  - ▶ Operaciones I/O – Un programa en ejecución pudiese requerir una I/O, la cual puede requerir la manipulación de un archivo o de un dispositivo
  - ▶ Manipulación del sistema de archivos – El sistema de archivos resulta particularmente interesante. Programas necesitan leer y escribir archivos y directorios, crearlos y eliminarlos, buscarlos, listar su información, manejar sus permisos, etc.
  - ▶ Comunicaciones – Los procesos pueden requerir el intercambio de información, en la misma máquina o entre máquinas a través de una red de comunicaciones
    - ▶ Memoria compartida, paso de mensajes, etc.

# Servicios del Sistema Operativo

---

- ▶ Un conjunto de servicios del SO provee funciones de utilidad a los usuarios:
  - ▶ Detección de errores – El SO necesita estar atento de manera constante ante posibles errores
    - ▶ Estos pueden ocurrir en el CPU, memoria, dispositivos de I/O, programas de usuario
    - ▶ Para cada tipo de error, el SO debe tomar las acciones necesarias para asegurar el funcionamiento correcto y consistente del sistema
    - ▶ Facilidades de depuración pueden ser de gran utilidad para que usuarios y programadores usen de manera eficiente el sistema

# Servicios del Sistema Operativo

---

- ▶ Otro conjunto de funcionalidades exportadas por el SO para asegurar un funcionamiento correcto en la compartición de recursos son:
  - ▶ Asignación de recursos – Cuando múltiples usuarios o trabajos se encuentra en ejecución concurrentemente, los recursos deben ser asignados de manera coherente
    - ▶ Distintos tipos de recursos – Algunos (como ciclos de CPU, memoria principal, y archivos de almacenamiento) deben tener códigos especiales de asignación, mientras que otros (como dispositivos de I/O) deben tener códigos de solicitud y liberación generales
  - ▶ Contabilidad – Es requerido saber qué recursos y qué cantidad ha utilizado un usuario

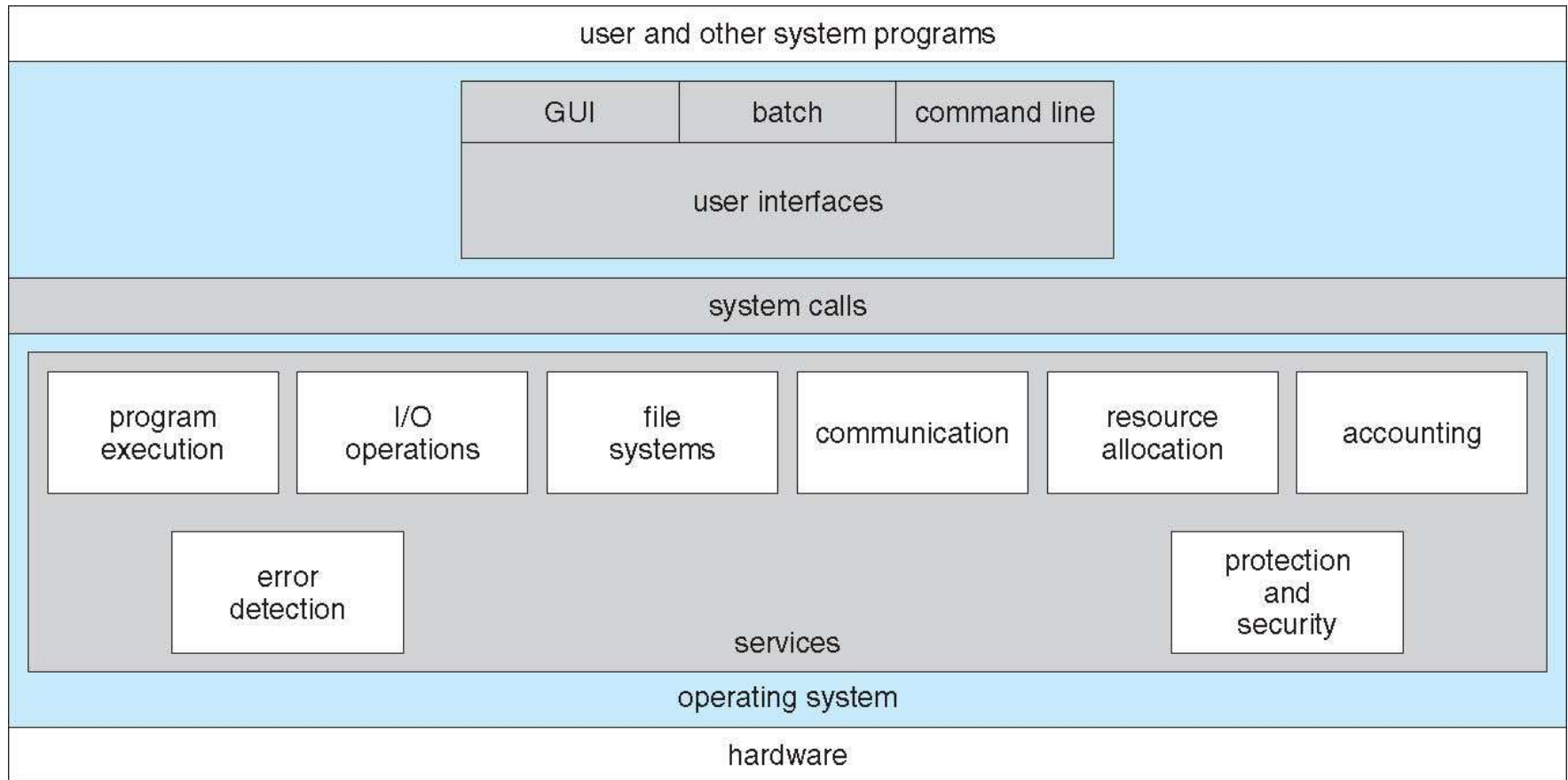
# Servicios del Sistema Operativo

---

- ▶ Otro conjunto de funcionalidades exportadas por el SO para asegurar un funcionamiento correcto en la compartición de recursos son:
  - ▶ Protección y seguridad – Los propietarios de la información almacenada en sistema multiusuario o de red pueden desear tener control sobre el uso de la información, del mismo modo procesos concurrente no deben interferir entre si
    - ▶ La protección involucra todo aquello que se vincule con el acceso a los recursos del sistema de manera controlada, generalmente se considera de ámbito interno
    - ▶ La seguridad del sistema contra extraños (ámbito externo) requiere que los usuario se autentiquen, se extiendan las defensas a accesos externos no autorizados, etc.
    - ▶ Una cadena es tan fuerte como lo es su eslabón más débil



# Una Vista de los Servicios del SO

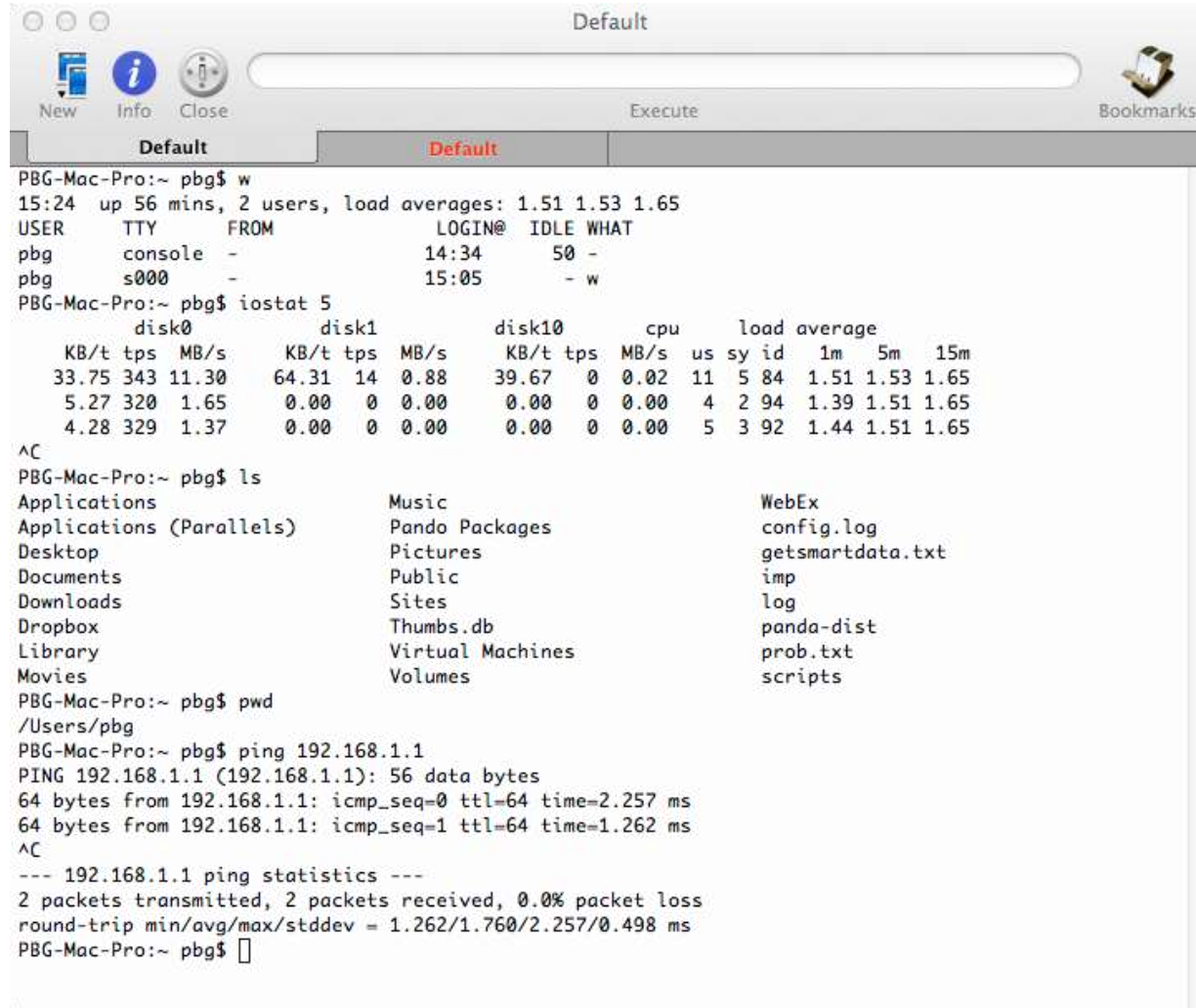


# Interfaz de Usuario del SO - CLI

---

- ▶ Un CLI o interprete de comandos permite la entrada de comandos de manera directa
  - ▶ Algunas implementaciones se encuentran en el kernel, otras a nivel de los programas de sistema
  - ▶ Diferentes implementaciones (shell)
    - ▶ Bash, sh, cmd, bourne, etc.
  - ▶ Los usuarios interactúan vía la ejecución de comandos
    - ▶ En algunas ocasiones los comandos son propios del sistema, en el resto de los casos invocan a programas de terceros
      - Esto permite agregar nuevas funcionalidades a la shell sin necesidad de modificar su estructura básica

# Interprete de Comandos Bourne



```
PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER      TTY      FROM          LOGIN@  IDLE   WHAT
pbg       console -              14:34    50    -
pbg       s000    -              15:05    -    w

PBG-Mac-Pro:~ pbg$ iostat 5

            disk0            disk1            disk10            cpu            load average
      KB/t tps MB/s      KB/t tps MB/s      KB/t tps MB/s  us sy id  1m  5m  15m
    33.75 343 11.30    64.31 14  0.88    39.67  0  0.02  11  5 84  1.51 1.53 1.65
     5.27 320  1.65      0.00  0  0.00      0.00  0  0.00   4  2 94  1.39 1.51 1.65
     4.28 329  1.37      0.00  0  0.00      0.00  0  0.00   5  3 92  1.44 1.51 1.65

^C
PBG-Mac-Pro:~ pbg$ ls
Applications                Music                        WebEx
Applications (Parallels)    Pando Packages             config.log
Desktop                     Pictures                    getsmartdata.txt
Documents                   Public                      imp
Downloads                   Sites                       log
Dropbox                     Thumbs.db                  panda-dist
Library                     Virtual Machines            prob.txt
Movies                      Volumes                    scripts

PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

# Interfaz de Usuario del SO - GUI

---

- ▶ **Escritorio amigable basado en metáforas**
  - ▶ Involucran manejo de ratón, teclado y pantalla
  - ▶ Los archivos, programas y acciones se representan mediante iconos
  - ▶ Diferentes botones del ratón provocan reacciones diferentes sobre los objetos, haciendo posible la ejecución de diversas acciones (proveer información, visualizar opciones, etc.)
  - ▶ Paradigma inventado por Xerox PARC y comercializado por Apple Inc.

# Interfaz de Usuario del SO - GUI

---

- ▶ Muchos sistemas incluyen interfaces tipo CLI y GUI
  - ▶ Microsoft Windows
    - ▶ Microsoft Windows → GUI
    - ▶ Command → CLI
  - ▶ Apple Mac OS
    - ▶ Aqua → GUI
    - ▶ Interprete estilo UNIX embebido → CLI
  - ▶ Unix y Linux
    - ▶ GNOME, KDE, XFCE → GUI
    - ▶ Interprete de comandos UNIX embebido → CLI

# Interfaz de Usuario del SO - GUI

- ▶ Los dispositivos de pantalla táctil requieren de nuevas interfaces
  - ▶ El uso de un mouse no es posible o no es deseable
  - ▶ Las acciones y selecciones se basan en gestos
  - ▶ Uso de teclado virtual



# GUI de MAC OS X



# Llamadas al Sistema

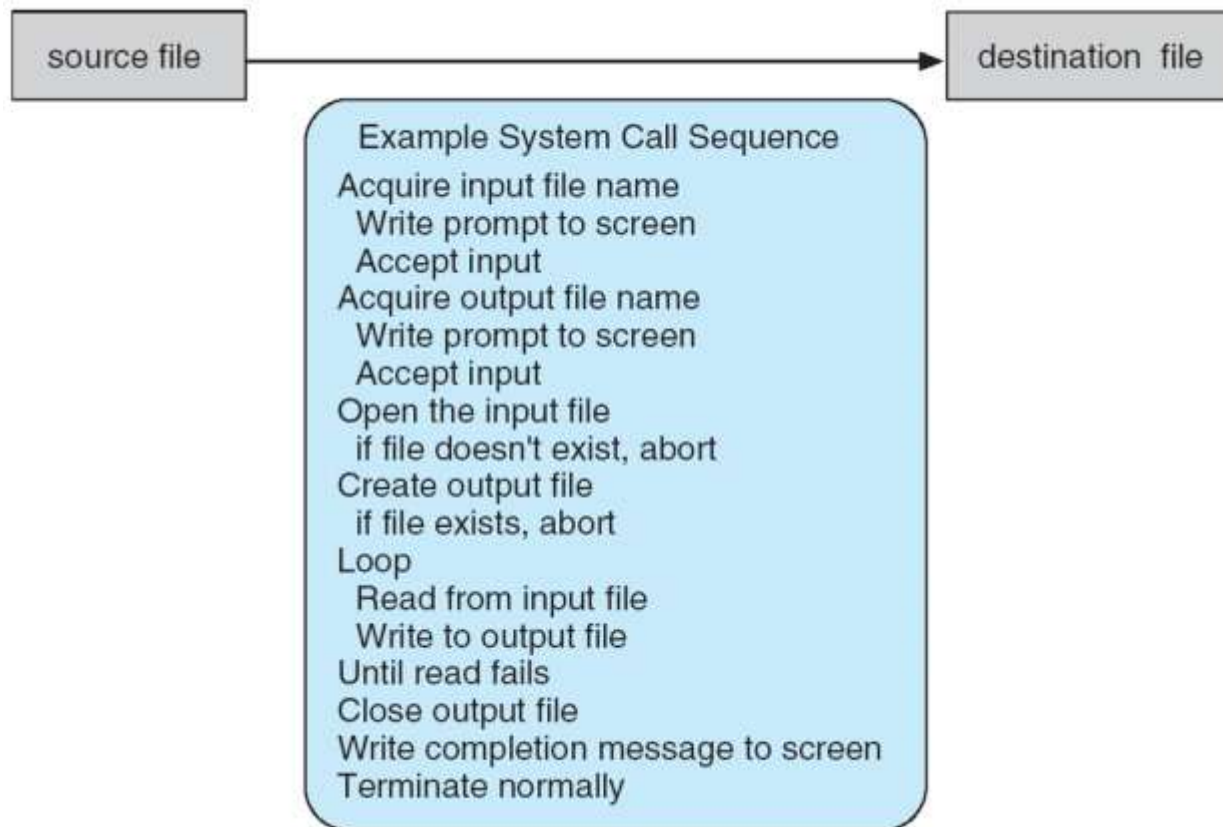
---

- ▶ Interfaz de programación de los servicios proporcionados por el SO
- ▶ Típicamente escrita en un lenguaje de alto nivel (C o C++)
- ▶ Usada frecuentemente por programas de usuario vía un API (Application Program Interface) de alto nivel, en lugar de realizar llamadas de manera directa
- ▶ Los APIs más comunes
  - ▶ API Win32 → Windows
  - ▶ API Posix → UNIX, Linux, Mac OS
  - ▶ API Java → Java Virtual Machine



# Ejemplo de una Llamada al Sistema

- Secuencia de una llamada al sistema para copiar el contenido de un archivo a otro



# Ejemplo de un API Estándar

## EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t  read(int fd, void *buf, size_t count)
```

ssize_t	read	(int fd, void *buf, size_t count)
return value	function name	parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

# Llamada al Sistema - Implementación

---

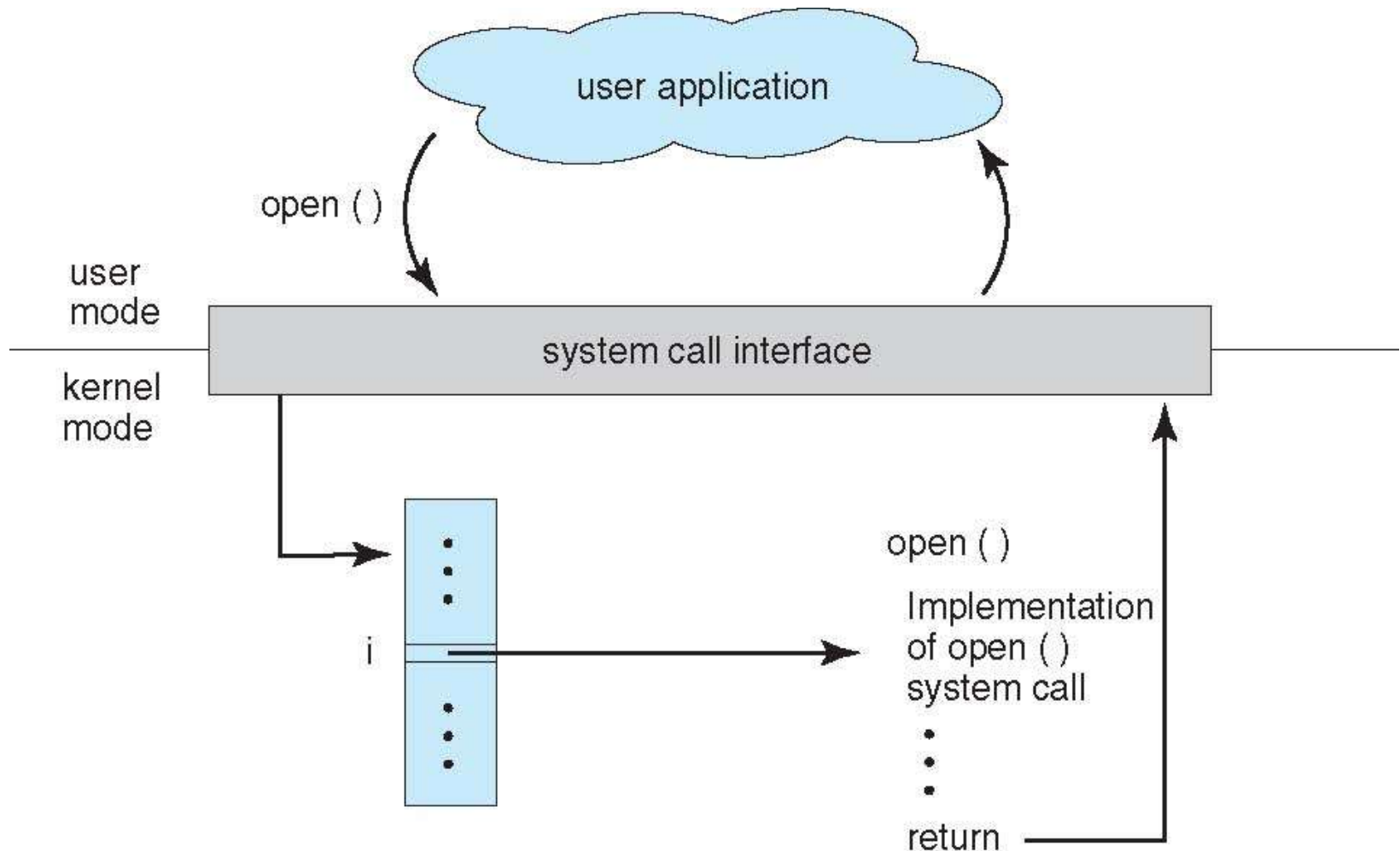
- ▶ Típicamente, un número es asociado con cada llamada al sistema
  - ▶ La **interfaz de llamada al sistema** mantiene una tabla indexada de acuerdo a estos números
- ▶ La interfaz de llamada al sistema invoca a la funcionalidad requerida en el kernel del SO y devuelve tanto su estado como los valores de retorno

# Llamada al Sistema - Implementación

---

- ▶ El invocador no requiere saber nada acerca de cómo se implementa la llamada al sistema invocada
  - ▶ Solo requiere interactuar con el API y entender que devolverá el SO como resultado
  - ▶ La mayoría de los detalles de la interfaz del SO se ocultan al programador mediante el API

# Relación API – Llamada al Sistema



# Estructura de los SO

---

- ▶ Los SO de propósito general son programas muy extensos
- ▶ Existen diversas maneras de estructurarlos, a continuación estudiaremos algunas de ellas

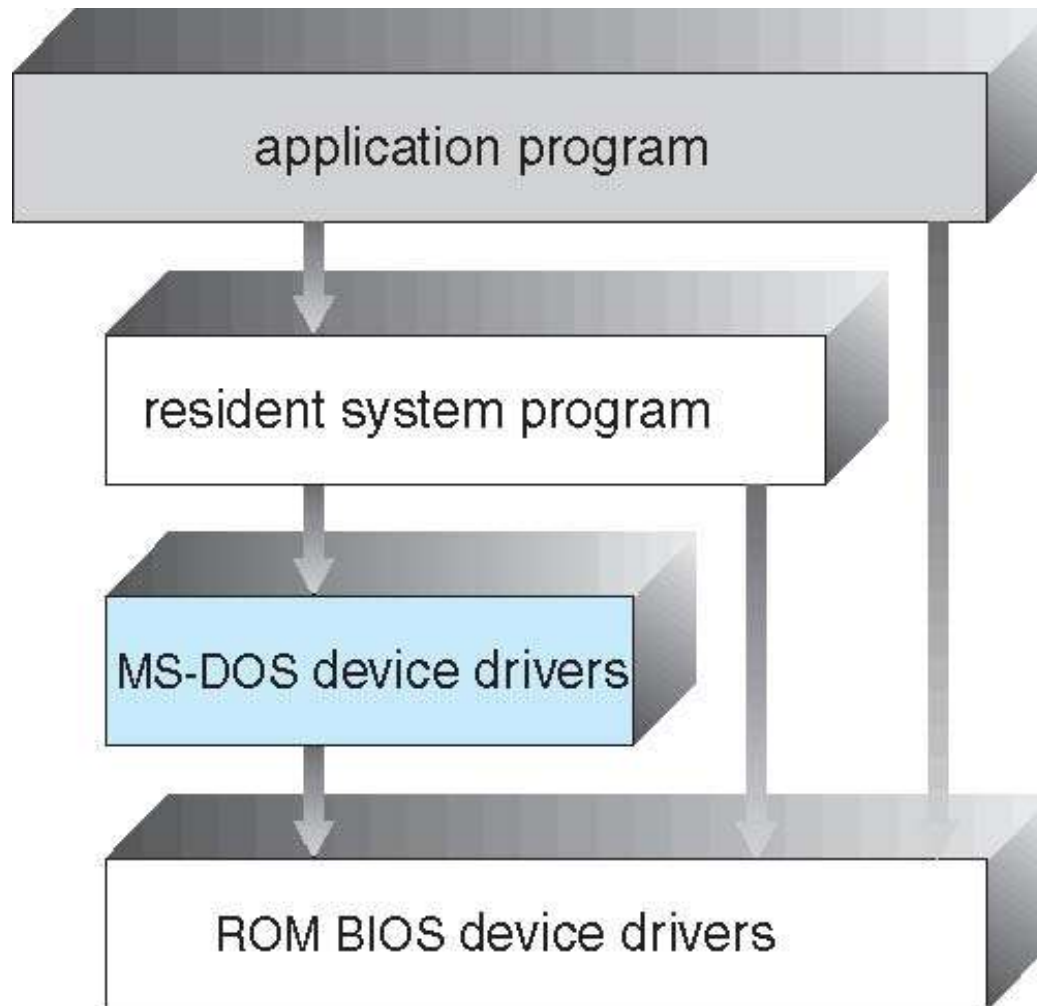
# Estructura Simple

---

- ▶ Por ejemplo, MS-DOS – Escrito para proveer la mayor cantidad de funcionalidades en el menor espacio posible
  - ▶ No esta dividido el módulos
  - ▶ Aunque MS-DOS posee una estructura, sus interfaces y funcionalidades no están claramente separadas

# Estructura Simple

---



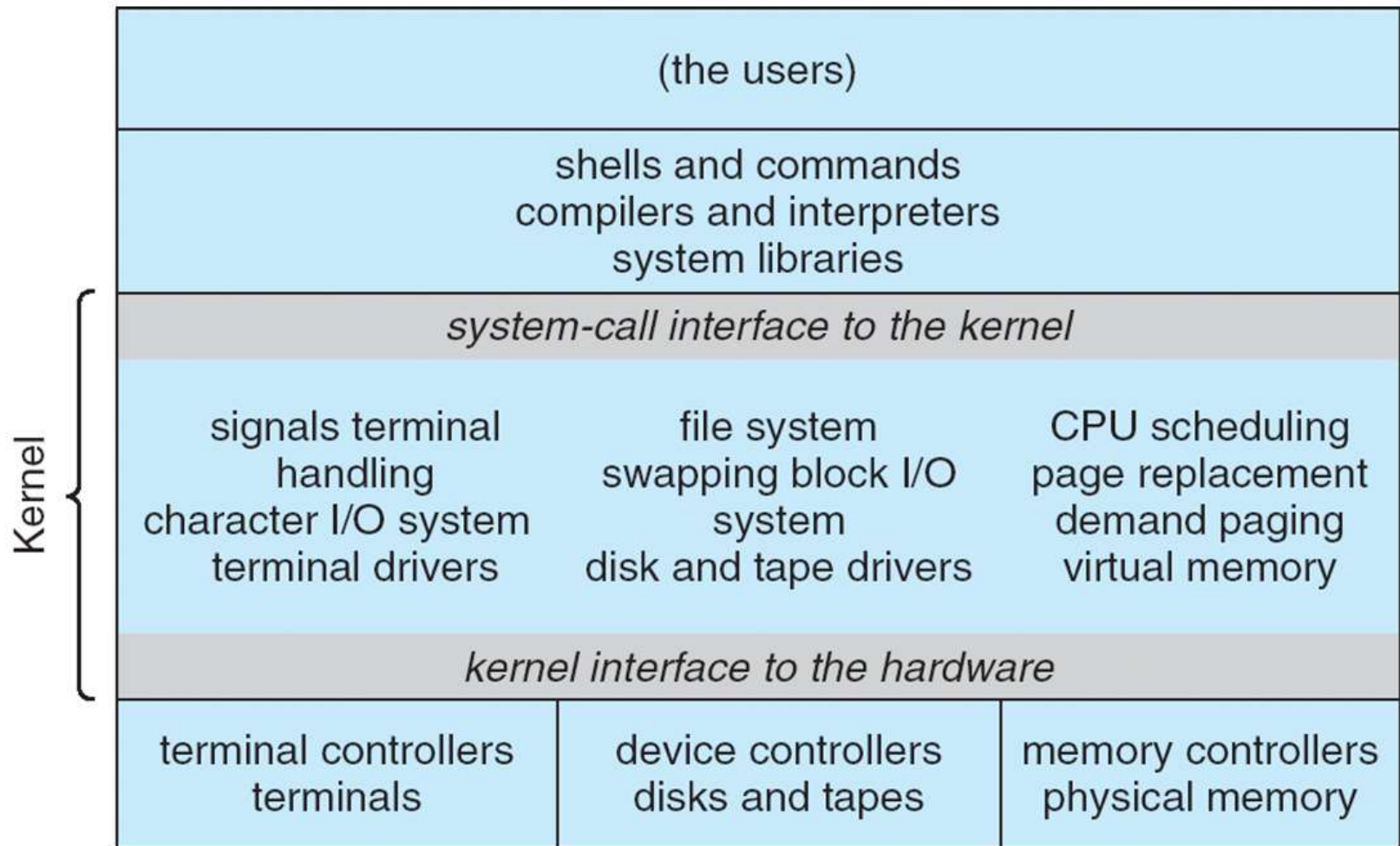


# UNIX

---

- ▶ UNIX – Limitado por las funcionalidades de hardware, el UNIX original poseía una estructura limitada.
- ▶ El SO UNIX consistía se estructuraba en dos partes separadas:
  - ▶ Los programas de sistema
  - ▶ El *kernel*
    - ▶ Consiste de todo aquello que resida por debajo de la interfaz de llamadas al sistemas y por encima del hardware
    - ▶ Provee sistemas de archivos, planificación de CPU, manejo de memoria, y otras funciones del SO; de hecho una gran cantidad de funcionalidades para un solo nivel

# Estructura – UNIX Tradicional



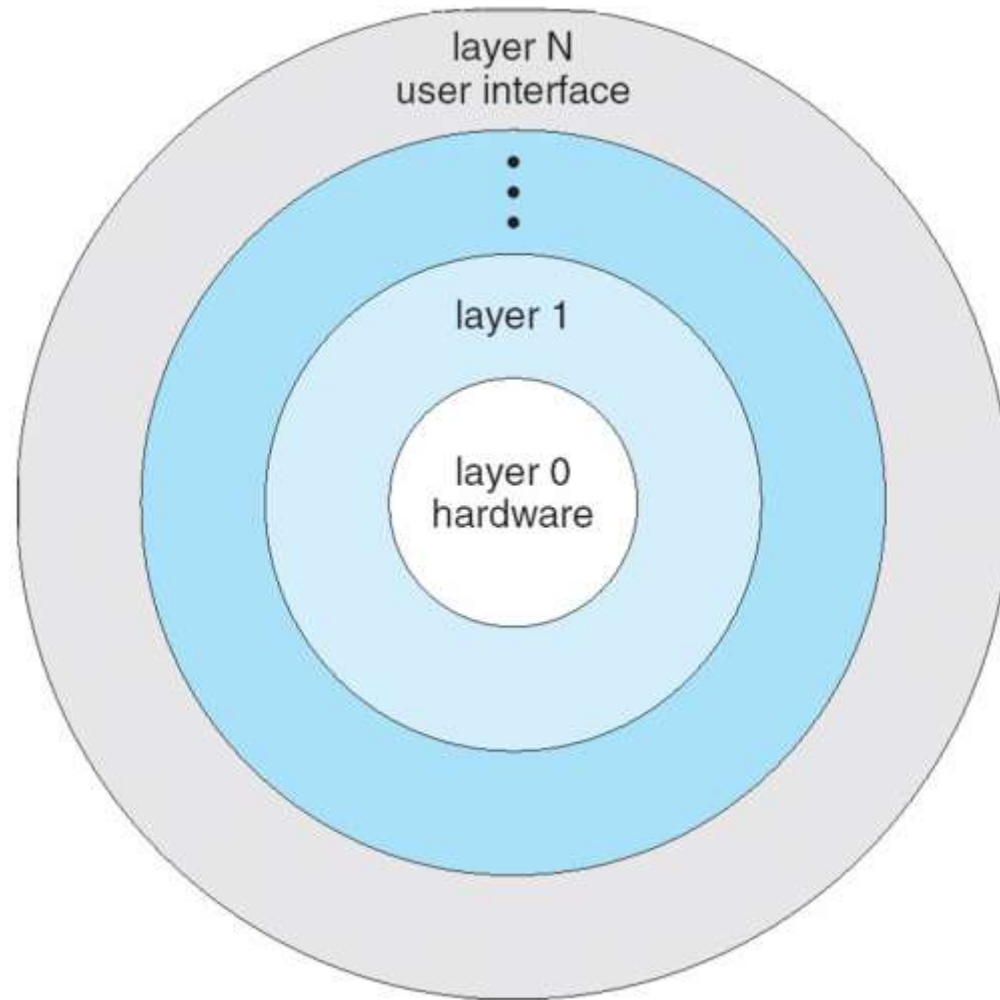
# Enfoque por Capas

---

- ▶ El SO es dividido en un número de capas (niveles), cada una es encima de la otra. La capa más baja (nivel 0) es el hardware, mientras que la capa más alta (nivel n) es la interfaz de usuario
- ▶ Haciendo uso de la modularidad producto de este enfoque, las capas son seleccionadas, así como las funciones y servicios que ellas ofrecen a sus superiores

# Enfoque por Capas

---



# Microkernel

---

- ▶ La idea es mover una gran cantidad de funcionalidades al espacio de usuario
  - ▶ ¿Por qué?
- ▶ Mach es un ejemplo de microkernel
  - ▶ El kernel de Mac OS X (Darwin) esta parcialmente basado en Mach
- ▶ La comunicación se lleva a cabo mediante el pase de mensajes entre los módulos que residen en el espacio de usuario

# Microkernel

---

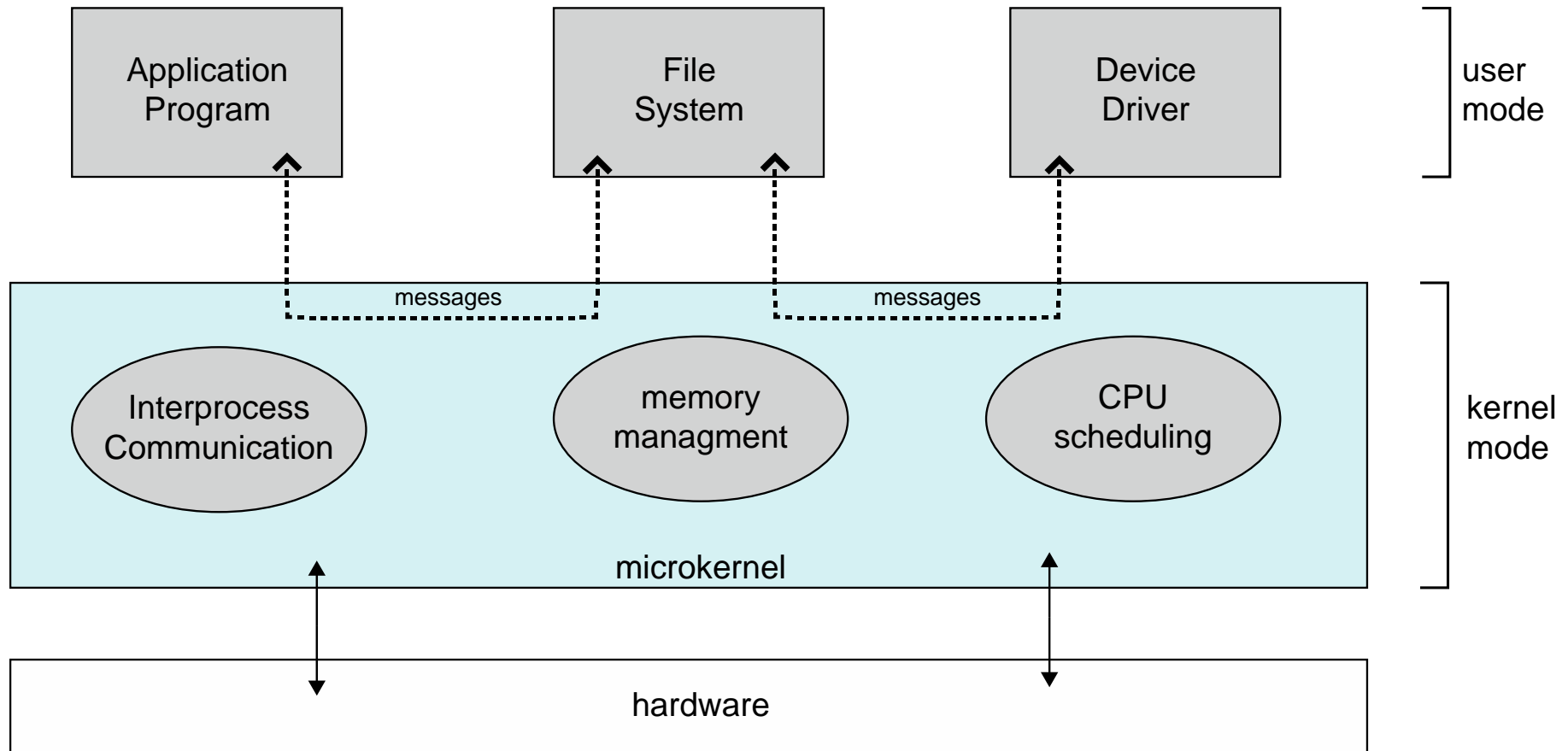
## ▶ Beneficios:

- ▶ El sencillo extender las funcionalidades del microkernel
- ▶ El fácil portar el SO a nuevas arquitecturas
- ▶ Mayor confiabilidad
  - ▶ Menos código ejecutándose en el espacio del kernel
- ▶ Mayor seguridad

## ▶ Desventajas:

- ▶ Sobrecarga ocasionada por la constante comunicación entre los módulos que se ejecutan en el espacio de usuario

# Microkernel



# Módulos

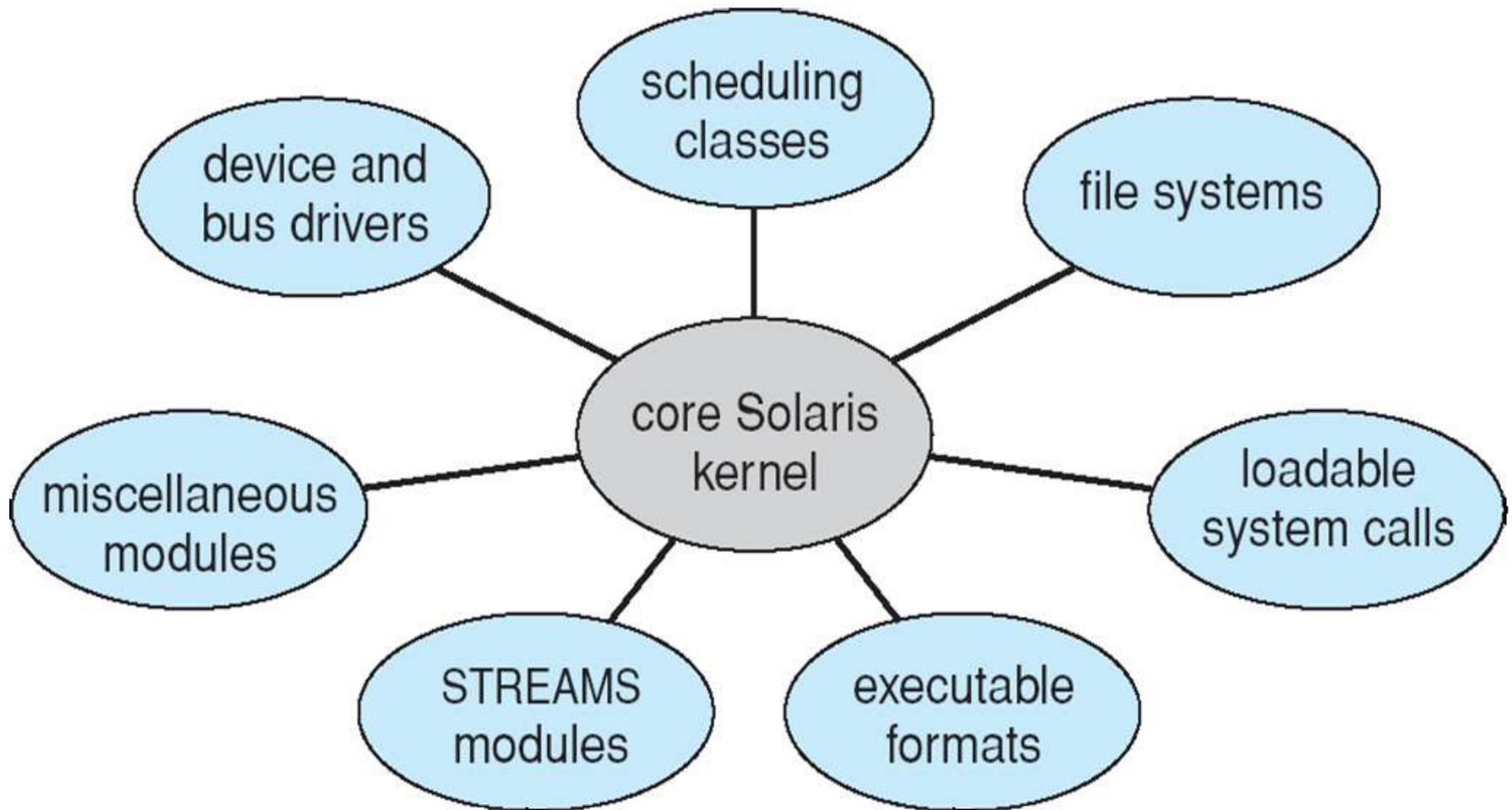
---

- ▶ Los SO modernos implementan el soporte de carga de módulos kernel
  - ▶ Usan un enfoque orientado a objetos
  - ▶ Cada componente principal se encuentra separado
  - ▶ La comunicación se realiza mediante interfaces bien conocidas
  - ▶ Es posible carga o descargar funcionalidades del kernel en caliente
- ▶ Su arquitectura es muy similar a la estructura de capas pero más flexible
  - ▶ Linux, Solaris, etc.



# Enfoque Modular en Solaris

---



# Sistemas Híbridos

---

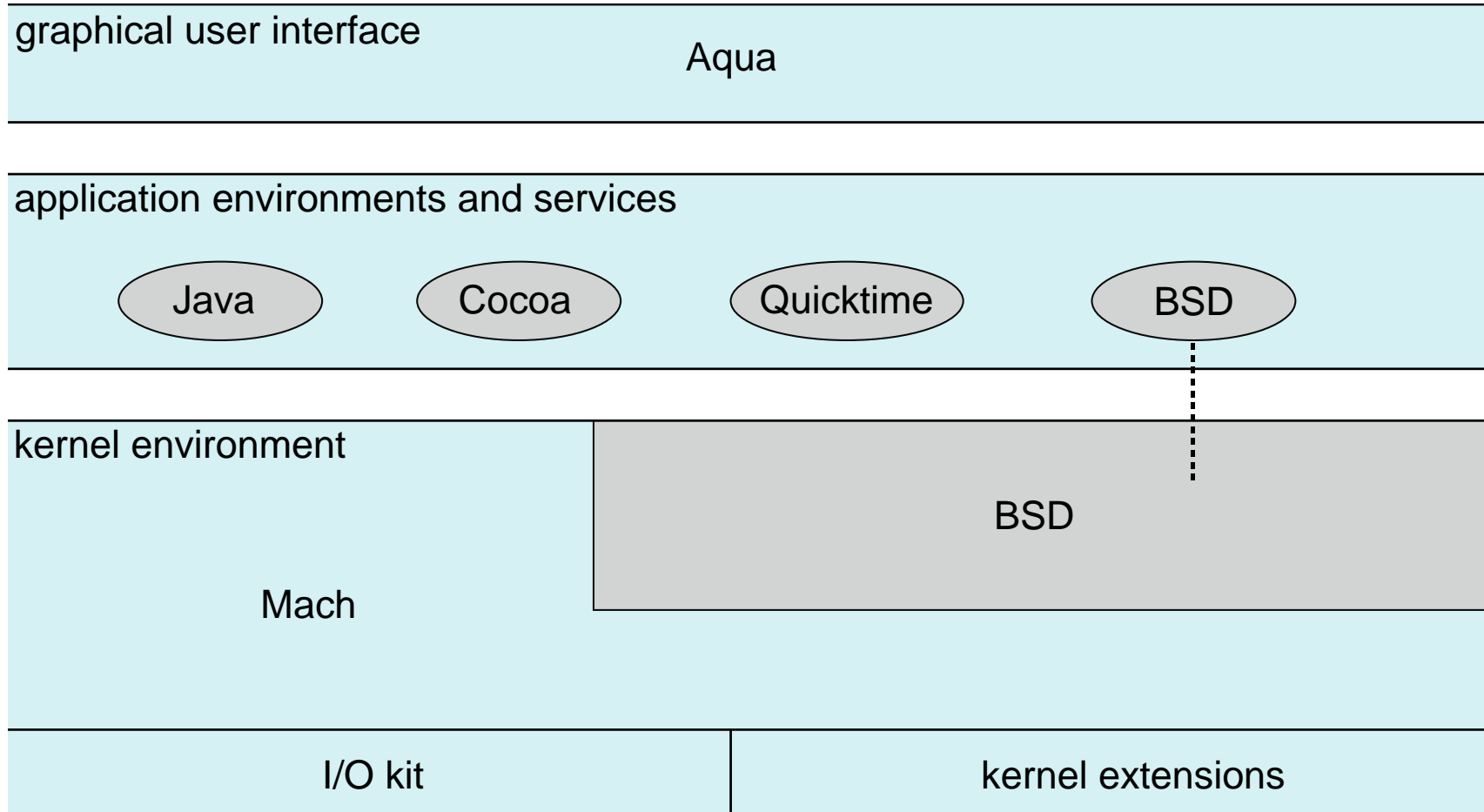
- ▶ La mayoría de los SO modernos no implementa un modelo puro
  - ▶ Los modelos híbridos combinan diferentes enfoques con la finalidad de mejorar en desempeño, seguridad, etc.
  - ▶ Linux y Solaris utilizan un enfoque monolíticos con soporte de módulos
  - ▶ Windows es básicamente monolítico, con una estructura de microkernel para diferentes subsistemas

# Sistemas Híbridos

---

- ▶ Mac OS x utiliza un enfoque híbrido de capas, una interfaz gráfica (Aqua), más un ambiente de programación (Cocoa)
- ▶ Por debajo el kernel se estructura como un microkernel estilo Mach, y un conjunto de partes de UNIX BSD. Adicionalmente el kernel soporta la carga y descarga de módulos, funcionalidad conocida como soporte de extensiones de kernel y extensiones de I/O

# Estructura de Mac OS



# iOS

---

- ▶ SO para dispositivos móviles Apple (iPhone, iPad)
  - ▶ Estructurado estilo Mac OS X, además de otras funcionalidades
  - ▶ No ejecuta aplicaciones de Mac OS X de forma nativa
    - ▶ Funciona en diferentes arquitecturas (ARM vs. Intel)
  - ▶ Cocoa Touch – API basado en C para el desarrollo de aplicaciones
  - ▶ Media Services – Capa de manejo de gráficos, audio, y video
  - ▶ Core Services – Provee soporte de *cloud computing*, base de datos
  - ▶ El core del SO, basado en el kernel de Mac OS X

Cocoa Touch

Media Services

Core Services

Core OS

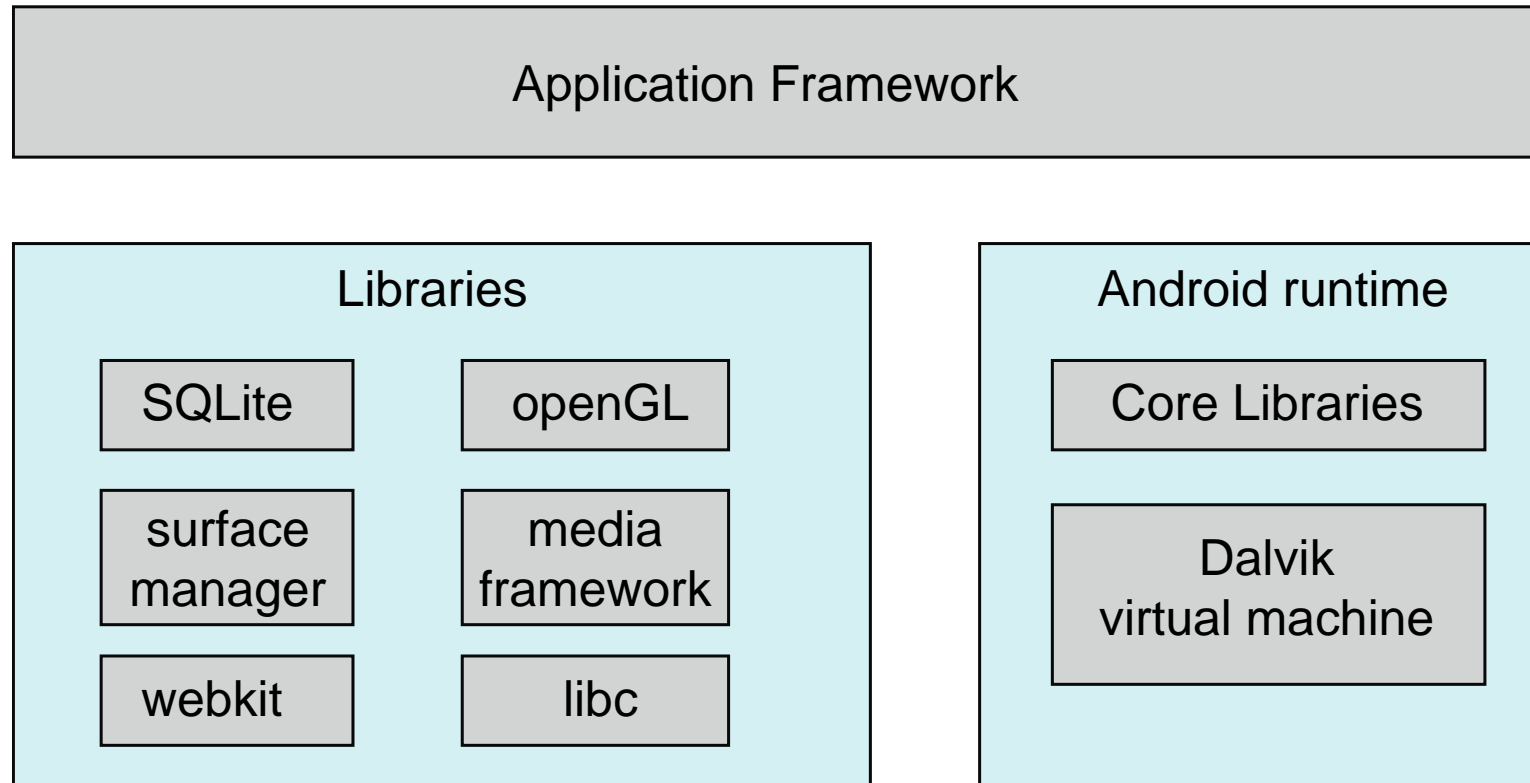
# Android

---

- ▶ Desarrollado por Google en alianza con la Open Handset Alliance
  - ▶ Código abierto
- ▶ Estructurado en capas
- ▶ Basado en el kernel de Linux, pero con modificaciones importantes
  - ▶ Provee manejo de procesos, memoria, drivers de dispositivos
  - ▶ Mejoras en el manejo de energía
- ▶ El ambiente de ejecución incluye un conjunto de bibliotecas y la máquina virtual Dalvik

# Android

- ▶ Las bibliotecas incluyen un ambiente de trabajo para navegación web, base de datos, multimedia, libc, etc.



# Arquitectura de Android

