Sistemas operativos avanzados

Tema 1

Introducción

Contenido

- *Definición de SO*
- *Componentes del SO*
- *Tipos de arquitectura del SO*
- *Virtualización*

Definición de Sistema Operativo (*déjà vu*)

- "Ubicuos pero no hacen un trabajo útil concreto"
- Gestión segura y eficiente de los recursos del computador
  - Reparto **temporal y espacial** de recursos entre programas
- Ofreciendo abstracción de "máquina extendida"
  - Servicios que ocultan e independizan del hardware
    - Crean abstracciones de recursos hardware
- ¿Algo más?: Sólo con eso, tan inútil como máquina desnuda
  - SO de propósito general debe ofrecer **interfaz a usuarios**
    - ¿Debe considerarse interfaz como parte del SO?
    - Suele estar implementado como aplicación externa
  - Definición precisa de SO: Asunto polémico
    - Linux vs. GNU/Linux
    - Juicio antimonopolio a Microsoft por IE
  - ¿A qué nos referimos cuando hablamos de SO?

Distintas interpretaciones del término SO

- Estricta (la que usaremos en la asignatura):
  - Gestor de recursos que ofrece servicios a aplicaciones
    - Interfaz de usuario es otra aplicación más fuera del SO
  - SO = Núcleo (*kernel*): software que ejecuta en **modo sistema**
    - No aplicable a sistemas con arquitectura micronúcleo
      - Parte del SO ejecuta en modo usuario
- Amplia (concepto de distribución en Linux):
  - Todo el software que **hace operativo** al sistema
    - Software de interfaz de usuario (p.e. GUI, *bash*)
    - Herramientas del sistema (p.e. montador *ld*)
    - "Demonios del sistema" (p.e. *spooler* de impresora)
    - Bibliotecas del sistema (p.e. *libc*)

Componentes del sistema operativo

- Gestión de procesos
- Gestión de memoria
- Sistema de entrada/salida
- Sistema de ficheros
- Sistema de seguridad y protección

Gestión de procesos

- Abstracción fundamental del SO
  - Proceso: programa en ejecución
- Cada proceso tiene un conjunto de recursos asociados:
  - Flujos de ejecución internos (*threads*)
  - Mapa de memoria
  - Ficheros abiertos, puertos de comunicación, ...
- SO debe controlar:
  - Creación y destrucción de procesos
  - **Comunicación y sincronización** del proceso
    - Así como asegurar su propia sincronización interna
  - Asignación y liberación de recursos al proceso
    - Evitando los **interbloqueos**
  - **Planificación de UCP**: qué proceso ejecuta en cada instante

Gestión de memoria

- SO ofrece espacio lógico propio (mapa) a cada proceso
  - Mapa del proceso incluye todas las regiones requeridas
    - Código, datos, pilas, DLL, ficheros proyectados, etc.
- SO gestiona mem. de sistema usando esquema fijado por MMU
  - Registros base/límite, segmentación, paginación, ...
- SO implementa la técnica de memoria virtual que permite:
  - Ejecutar procesos cuyo mapa es más grande que la memoria
  - Aumentar el grado de multiprogramación

Sistema de entrada/salida

- Manejadores se encargan de gestionar los dispositivos
  - Uno por cada tipo de dispositivo
  - Ofrecen interfaz común a pesar de gran variedad de dispositivos
  - Gestionan todos los aspectos hardware (p.e. DMA)
- Implementación de aspectos avanzados
  - Control de consumo de energía del dispositivo en portátiles
  - *Hot-plugging*
- Manejador de disco crítico en SO (sirve a G. memoria y S. Fich.)
  - Algoritmos de planificación del disco

Sistema de ficheros

- Fichero: abstracción de espacio de almacenamiento
- Espacio jerárquico de nombres usando directorios
- Tendencia: dar soporte a distintos tipos de sistemas de ficheros
  - Concepto de sistema de ficheros virtual: interfaz común
- Cada tipo de sistema de ficheros específico usa estrategias para:
  - Organización del espacio ocupado por los ficheros
  - Gestión del espacio libre
  - Técnicas de prevención ante caídas (p.e. *journaling*)

Sistema de seguridad y protección

- Protección:
  - ¿Qué operaciones puede hacer usuario con recurso?
  - ¿Cómo almacenar información sobre permisos?
    - Asociada al usuario: Capacidades
    - Asociada al recurso: Listas de control de acceso
  - DAC, MAC y RBAC
- Autenticación:
  - Asegurar que un usuario es quien dice ser
- SO debe evitar amenazas a la seguridad e integridad del sistema

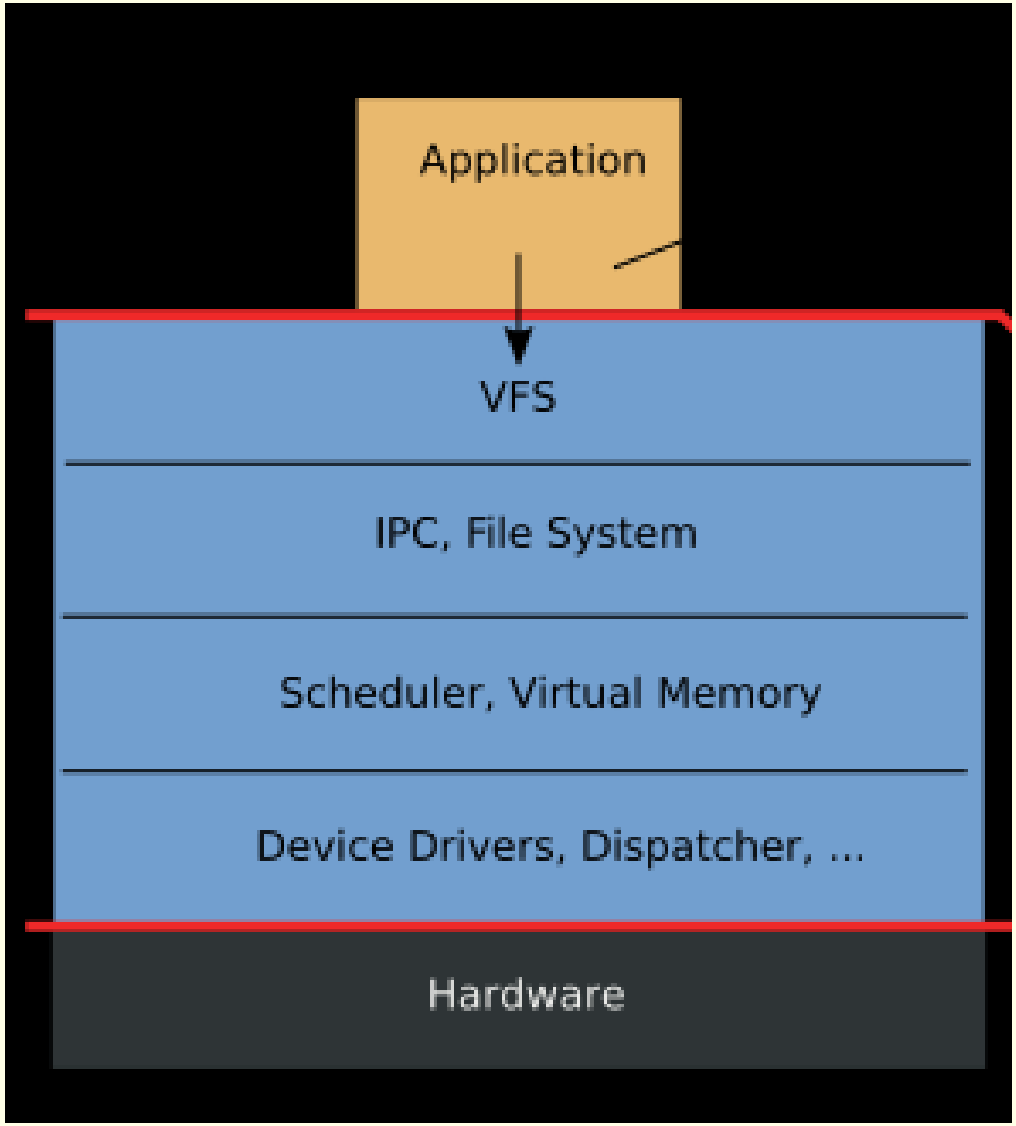
Arquitectura del sistema operativo

- Existen distintas alternativas:
  - Sistemas monolíticos
  - Sistemas por capas
  - Sistemas basados en micronúcleos
  - Sistemas híbridos
  - Exonúcleos

Sistemas operativos monolíticos

- SO = núcleo (*kernel*) → programa que ejecuta en modo sistema
  - Todo el código del SO enlazado en un único programa que
  - Ejecuta en un mismo espacio de direcciones
- Aplicaciones y programas de sistema ejecutan en modo usuario
- **Ventaja**: Eficiencia
  - Ejecución de servicio de SO:
    - Sólo requiere cambio de usuario a sistema y viceversa
- **Desventaja**: Difícil depuración y extensibilidad
  - Error en cualquier parte del SO afecta al resto
- Es la arquitectura más habitual
  - Característica de la familia UNIX (Linux)

Sistema monolítico (wikipedia)



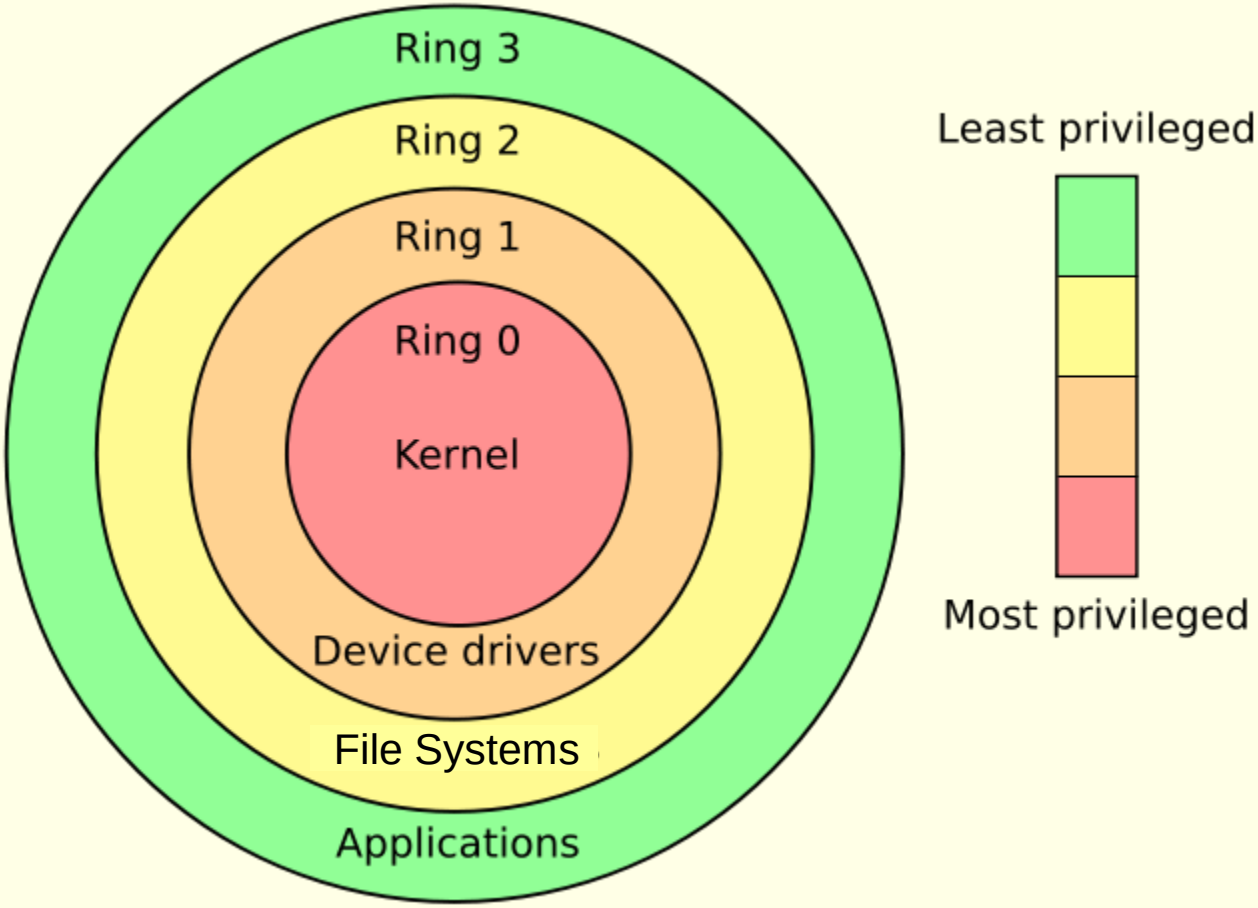
Sistemas con módulos cargables

- Mayoría de sistemas monolíticos actuales no están “cerrados”
  - Permiten cargar módulos en tiempo de ejecución
- Ejecutable del SO contiene funcionalidad básica
- Restante en módulos (manejadores, s. ficheros, protocolos, etc.)
- Módulo similar a biblioteca dinámica pero para el núcleo
  - Se incorpora a espacio de SO y comparte símbolos
  - Se mantienen los mismos problemas de fiabilidad
- **Ventajas**:
  - Facilita extensibilidad del SO
  - Permite adaptar núcleo a características de la plataforma
    - P.ej. Crear núcleo mínimo para sistema empotrado
  - Posibilita técnicas como *hot-plugging*

Sistemas por capas (o anillos)

- Organizar funcionalidad del SO en capas independientes
  - Cada capa es un ejecutable independiente que
  - Ejecuta en su propio espacio de direcciones
  - Organizadas en niveles de mayor a menor privilegio
  - Capa sólo se comunica con adyacentes usando llam. al sistema
    - Procesador verifica que se trata de capas adyacentes
- Facilita depuración y controla propagación de errores
- Requiere que procesador provea de varios niveles de privilegio
  - Pentium proporciona 4
  - S. monolíticos sólo requieren dos modos (usuario/sistema)
    - Más transportables
- Primer SO por capas: THE (Dijkstra, 1968)
  - MULTICS también usó esta arquitectura

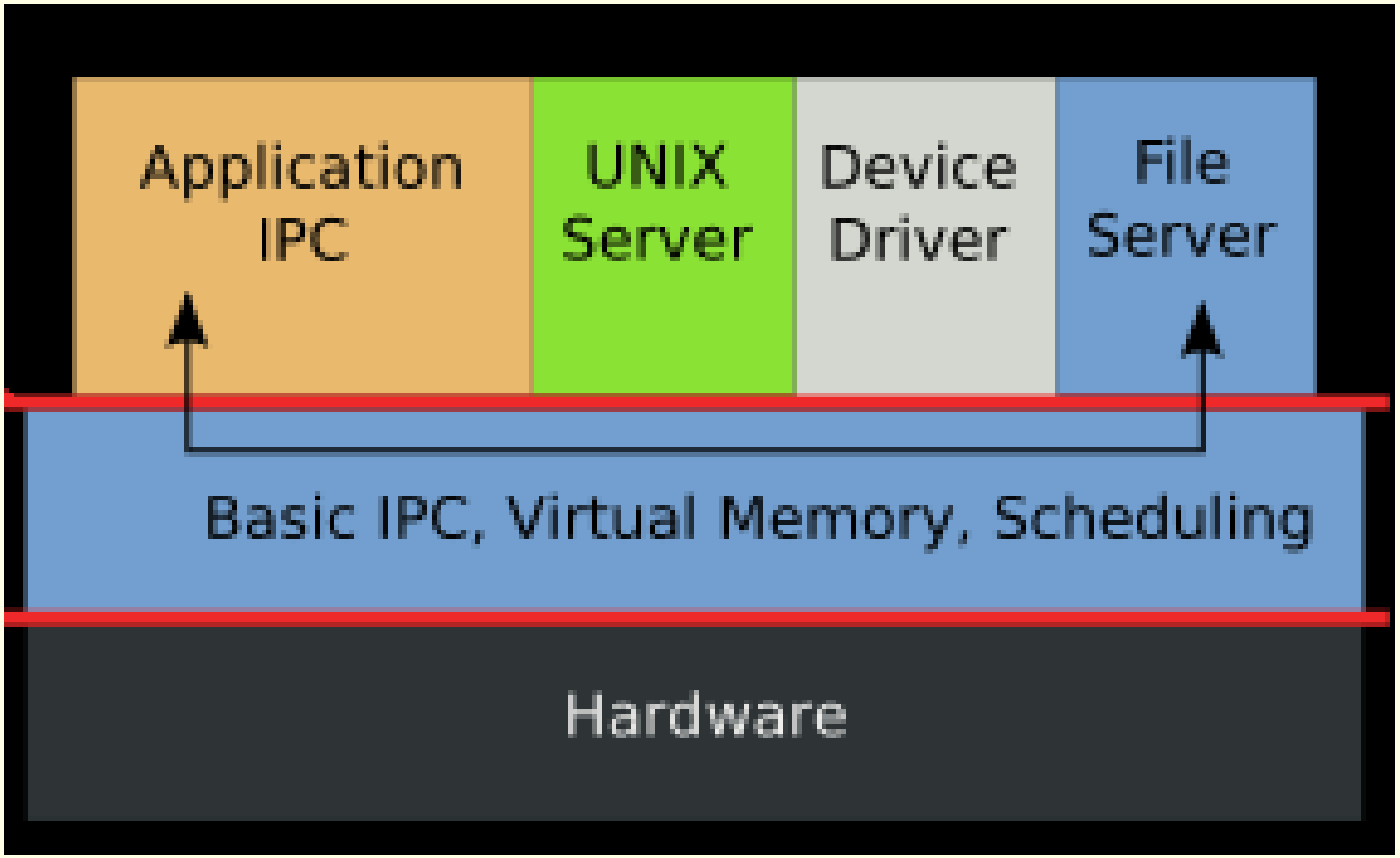
Sistema organizado en anillos (wikipedia)



Micronúcleos

- “Lo perfecto no es que no falte nada sino que no sobre nada”
- Núcleo queda reducido a funcionalidad mínima
  - Gestión de procesos y de memoria de bajo nivel + IPC
    - Micronúcleo proporciona nº muy reducido de llamadas
  - Gestión básica de hardware
    - P.e. interrupción → mensaje a proceso que la gestiona
- Funcionalidad del SO en servidores en modo usuario
  - Sistema de ficheros, gestor de memoria, manejadores, ...
  - “Llamada al sistema” de aplicación: mensaje a servidor
    - Y entre servidores si es necesario
- Aplicación no sabe si ejecuta sobre micronúcleo o monolítico

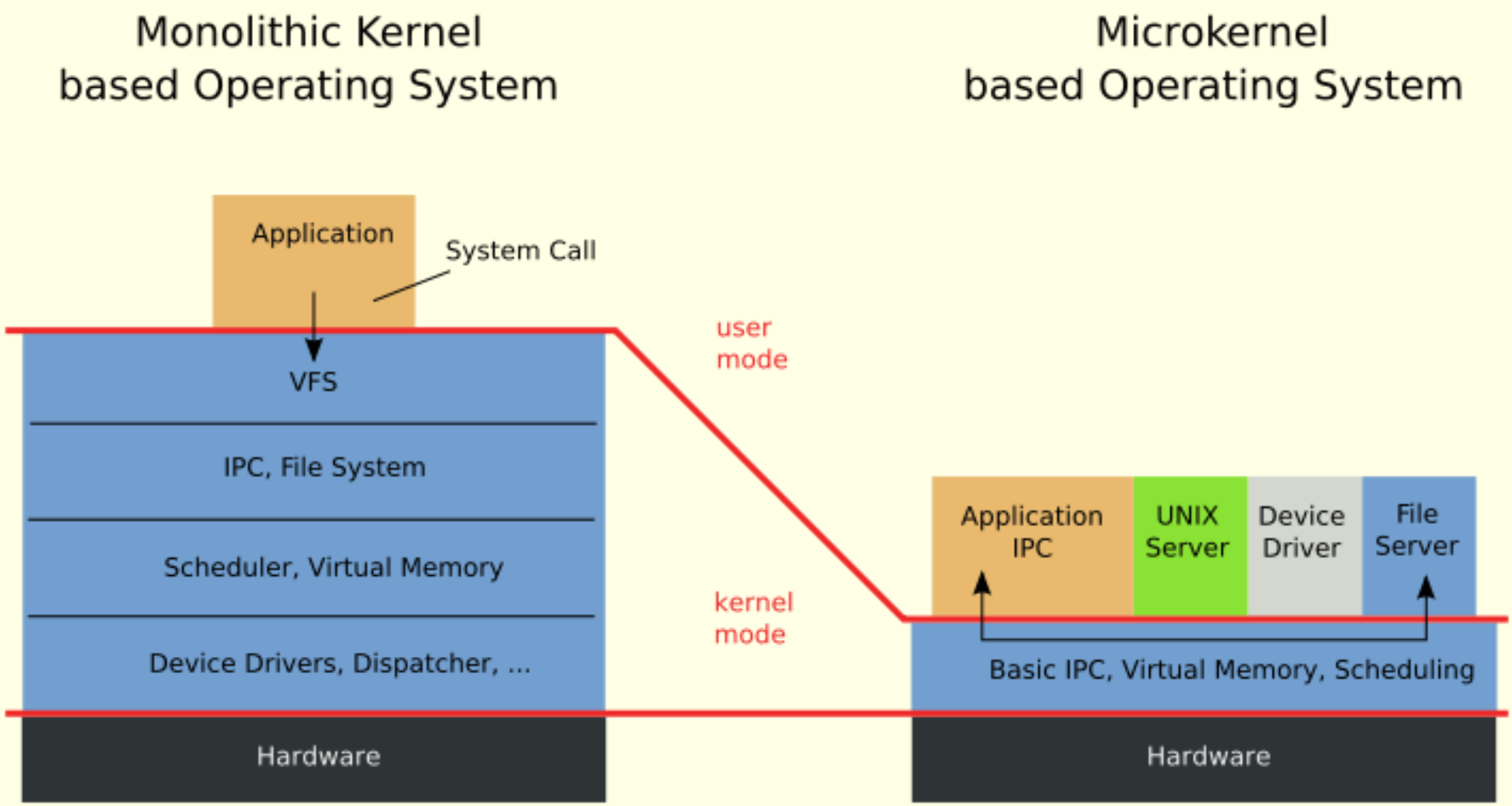
Sistema basado en micronúcleo (wikipedia)



Micronúcleos vs. monolíticos

- Ventajas:
  - Extensibilidad, fiabilidad y facilidad de depuración
  - Error en parte del SO sólo afecta al servidor involucrado
  - Posible convivencia de varios SS.OO.
  - Facilita programación funcionalidad del S.O.
    - Programación convencional vs. Programación módulos Linux
- Desventajas:
  - Eficiencia.
  - Coste de llamada:
    - Sobrecarga mensajes y cambios de proceso

Monolítico versus micronúcleo (wikipedia)



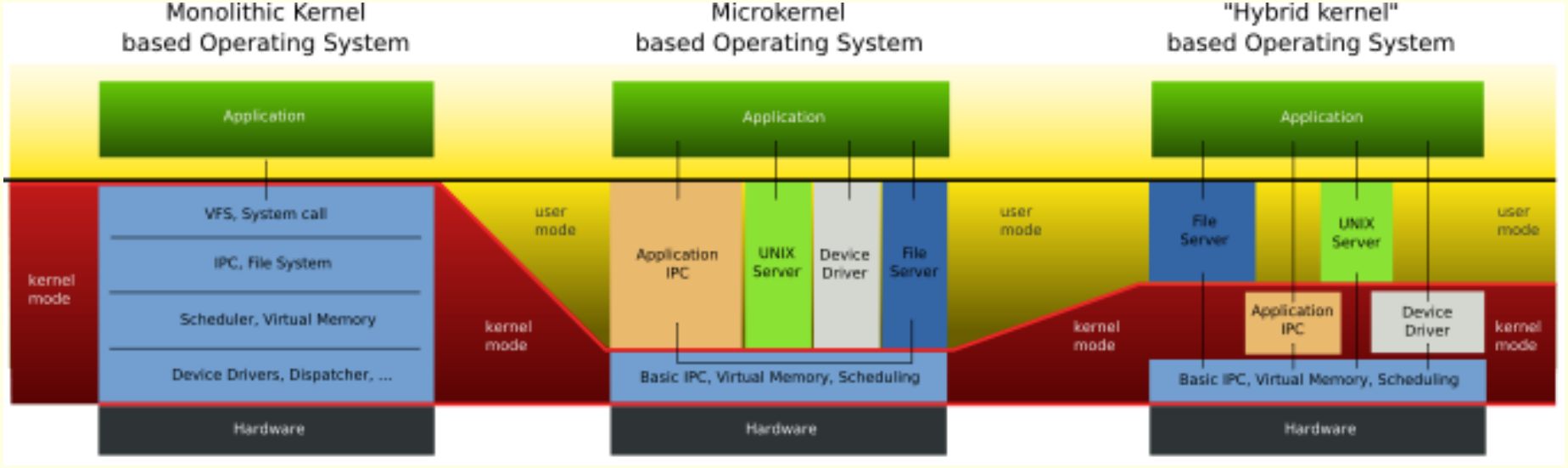
Evolución de los micronúcleos

- Primeros sistemas: mono-servidor
  - Único servidor en modo usuario proporciona todos servicios
  - Menos sobrecarga por mensajes y cambios de contexto
  - Pero pierde muchas de las ventajas de micro-núcleos
- Eficiencia ha mejorado en 2ª generación (L4) frente a 1ª (Mach):
  - Mach (1987): IPC ≈ 100 μs; L4 (1995): IPC ≈ 1 μs
- Tamaño del núcleo ha ido disminuyendo:
  - Mach ≈ 100 llamadas; L4 ≈ 10 llamadas
  - Mach ≈ 100.000 líneas de código; L4 ≈ 10.000
- 3ª generación: núcleos verificados formalmente (SeL4)

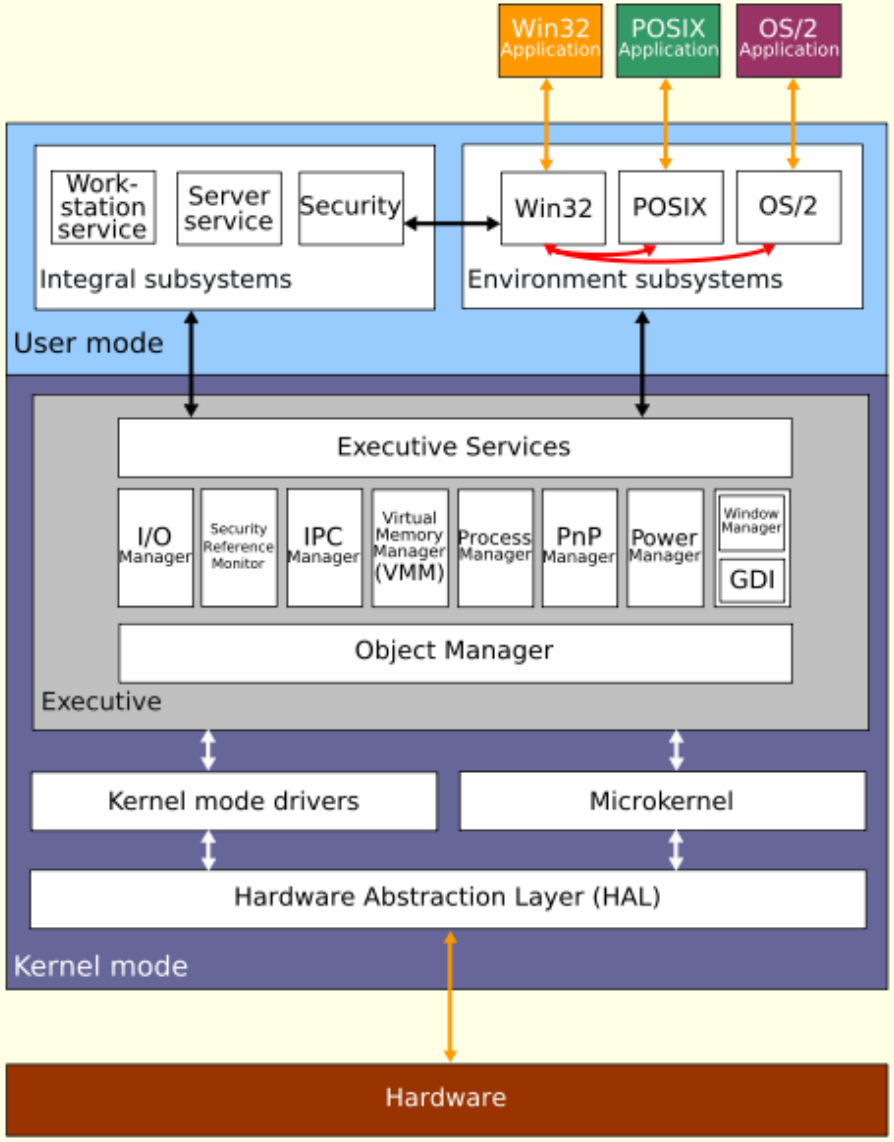
Sistemas híbridos

- Algunos s. micronúcleo permiten servidores en modo sistema
  - Más eficiente pero rompe la filosofía micronúcleo
  - Servidores son programas independientes pero
  - Ejecutan en mismo espacio de direcciones del micronúcleo
    - Y no usan IPCs para comunicarse
- Categoría discutida
  - Puristas consideran que son monolíticos
- Ejemplo: Mac OS X
  - Interfaz UNIX
  - Micronúcleo Mach con servidores en m. sistema
- ¿Y Windows?
  - Difícil clasificación

Monolítico | Micronúcleo | Híbrido (wikipedia)



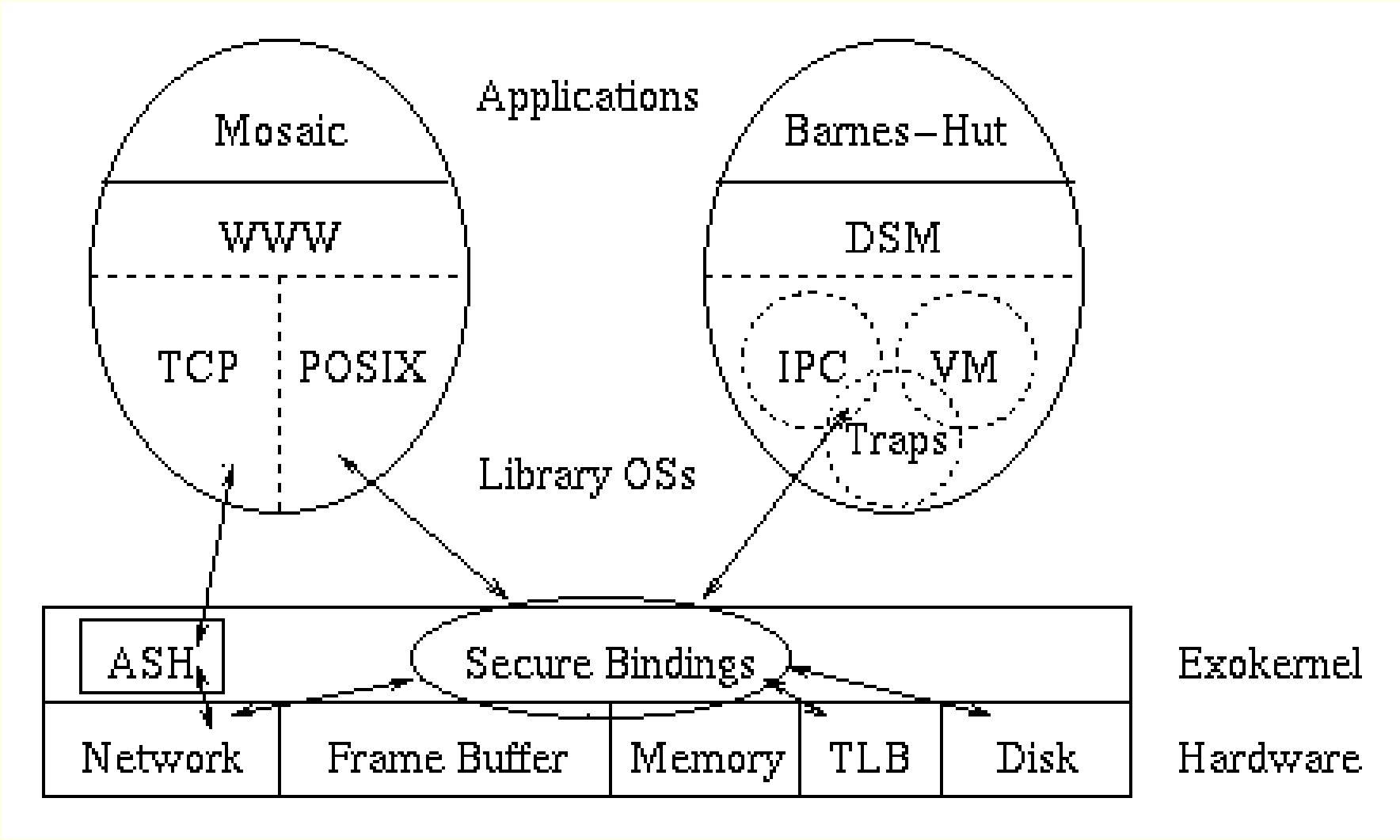
Estructura de Windows 2000 (wikipedia)



Exonúcleos

- Motivación:
  - No todas las aplicaciones necesitan ver mismas abstracciones
    - Gestor base de datos mejor maneja bloques de disco que ficheros
      - Uso de abstracciones inadecuadas es ineficiente
- Propuesta: *Exokernel* (MIT, 1995)
  - Núcleo provee abstracciones básicas (página, bloque, ...)
  - Funcionalidades de tipo SO en bibliotecas en modo usuario
  - Cada aplicación se enlaza con las bibliotecas que requiera
    - Gestor base de datos no requiere de sistema de ficheros
  - Aplicación sabe que está ejecutando sobre un exonúcleo
- Aplicación del principio "*end-to-end*"

Exonúcleo (MIT)



http://www.cs.cornell.edu/home/ulfar/ukernel/ukernel.html

Virtualización

- Máquinas virtuales de sistema
- Hipervisor tipo 1 vs. tipo 2
- Virtualización completa
- *Paravirtualización*
- Virtualización asistida por hardware
- Virtualización a nivel de SO: contenedores

Virtualización

- Presente en todos los ámbitos de un sistema
  - Proceso *virtualiza* procesador
  - Mapa de memoria de proceso *virtualiza* memoria del sistema
  - Fichero *virtualiza* disco
  - Socket *virtualiza* red
- *"All problems in computer science can be solved by another level of indirection"*
- S.O. crea una máquina virtual *extendida*
  - Con un nivel de abstracción mayor que la máquina real
- **Máquina virtual de sistema (VM)**
  - Crea entorno de ejecución con **mismo nivel** de abstracción
    - Un **duplicado** del hardware real
  - ***Virtual Machine Monitor* (VMM) (aka Hipervisor)**
    - Componente que crea y gestiona las máquinas virtuales

Máquinas virtuales de sistema

- Pionero CP-40 (IBM, 1967): ¿t. compartido y SO monousuario?
  - CP crea 1 MV/usuario: SO monousuario CMS sobre cada MV
- Popek & Goldberg (P&G 1974): 3 condiciones
  - **Fidelidad**: SW mismo comportamiento sobre VM que en real
    - Exceptuando aspectos vinculados con el tiempo
  - **Seguridad**: VMM tiene control total de recursos virtualizados
  - **Rendimiento**: mayoría de instrucciones ejecutadas por m. real
  - MV: duplicado **eficiente** y aislado de una máquina real
- Técnica de nuevo en auge actualmente
  - Capacidad de procesamiento actual palia ineficiencia de MV
  - Procesadores incluyen soporte para la misma

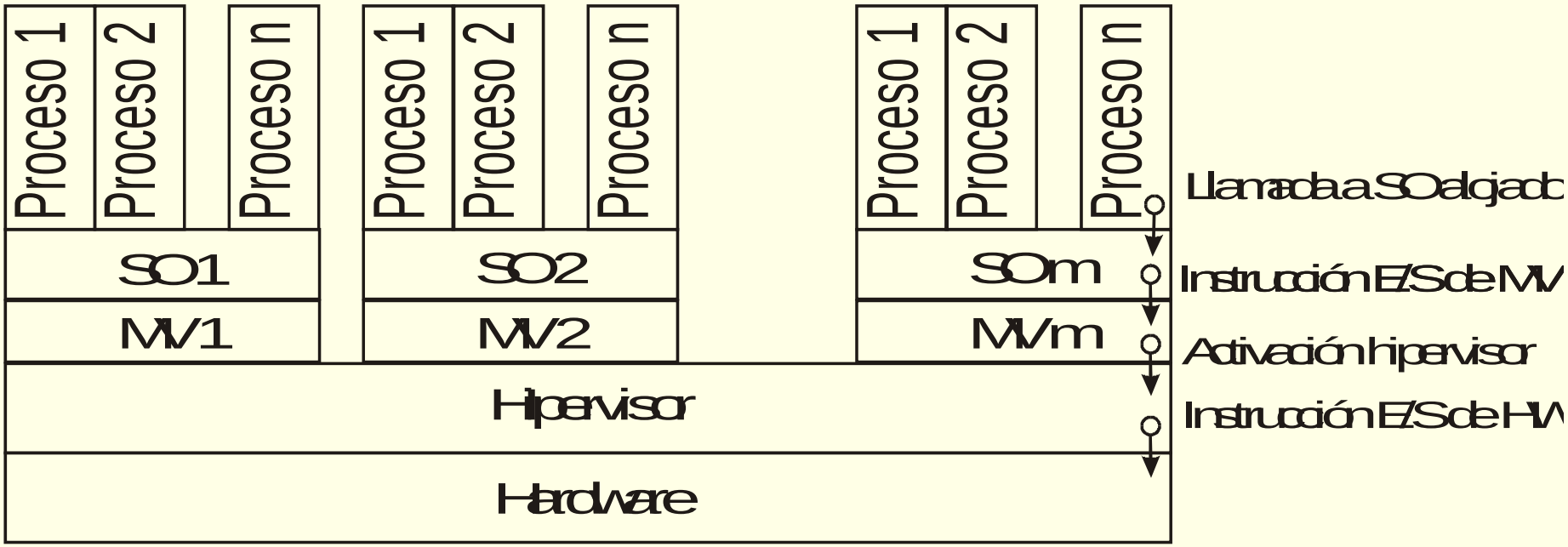
Ventajas del uso de máquinas virtuales de sistema

- Coexistencia de distintos SO
- Aislamiento (*sandboxing*)
- Satisfacción de QoS
- Ejecución de *legacy systems*
- Desarrollo de funcionalidad del SO
- Disponibilidad
- Mejor aprovechamiento del hardware
- Ahorro energético
- Consolidación de sistemas
- Migración en vivo
- Facilita administración de sistemas
- Uno de los pilares del *Cloud Computing*

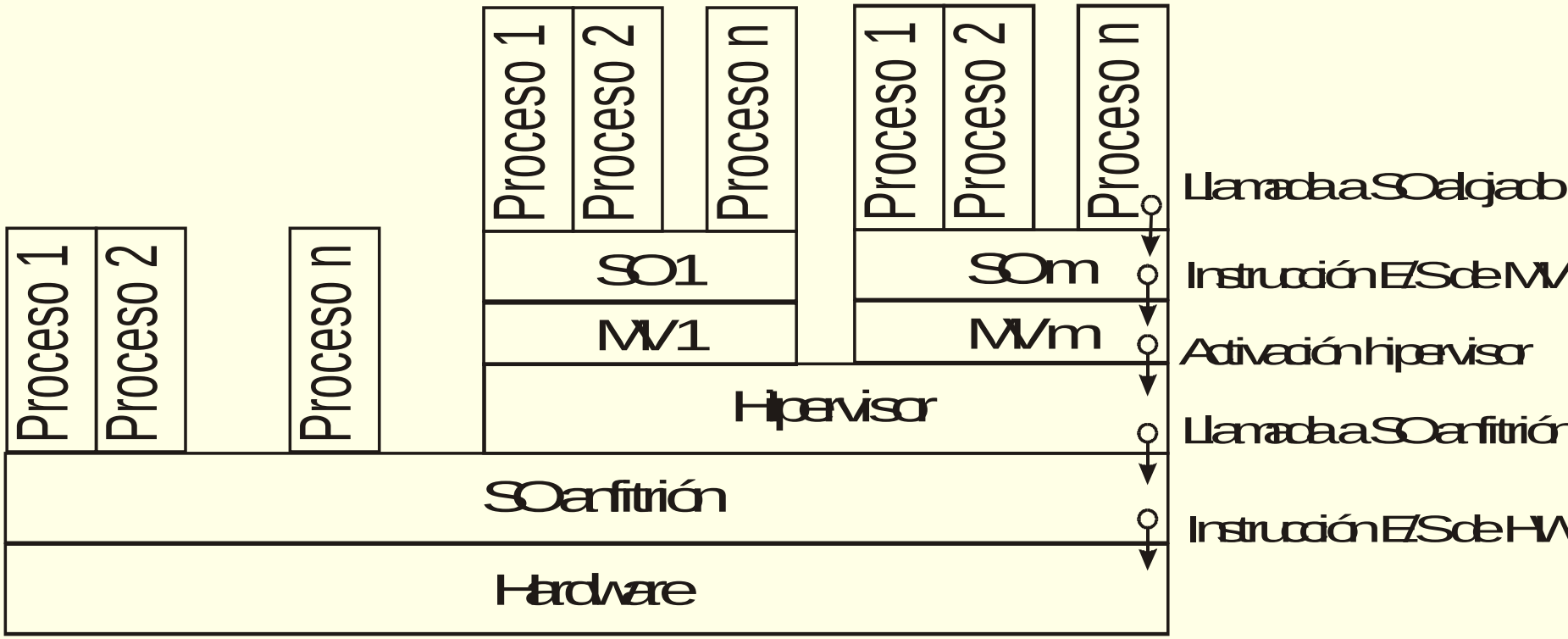
Tipos de máquinas virtuales de sistema

- Tipo 1 (nativa o *bare-metal*):
  - Hipervisor ejecuta sobre máquina desnuda
  - Sobre cada MV ejecuta **SO alojado**
  - VMware ESX/ESXi, Citrix XenServer, MS-Hyper-V,…
  - Normalmente implementado mediante módulo software
    - En algunos sistemas por HW y firmware (LPAR de IBM)
- Tipo 2 (alojada):
  - Hipervisor ejecuta sobre **SO anfitrión**
    - Un proceso más del sistema
  - Sobre cada MV ejecuta **SO alojado** (posiblemente distinto)
  - VMware Workstation, VMware Player, VirtualBox,…
  - Hipervisor invierte la abstracción de SO anfitrión
    - P.ej. Interrupción real → Señal UNIX → Interrupción MV

MV de sistema tipo I (nativa)



MV de sistema tipo II (alojada)



Técnicas de implementación de MV de sistema

- Emulación completa
  - Intérprete emula juego de instrucciones del procesador
    - Procesador emulado puede ser distinto que real
  - **No cumple 3ª condición** de P&G (**rendimiento**)
  - Ejemplo: Bochs emula x86 mediante interpretación
- *Trap&emulate* (implementación "clásica")
- Traducción binaria
- Paravirtualización
- Soporte hardware

- **NOTA**: Pueden ser técnicas complementarias

Trap&emulate

- Para mantener control y aislamiento entre MVs, SO alojado:
  - No puede ejecutar instr. privilegiadas (p.e. inhibir interrup.)
  - No debe acceder directamente a ciertas direcciones
    - P.e. modificar una entrada de una tabla de páginas
- 3 modos ejecución: usuario, privilegiado virtual, privilegiado real
- Solución: SO alojado ejecuta con UCP en modo no privilegiado
  - Si inst. privilegiada o acceso a dirección no permitida → ***trap***
    - VMM toma control y **emula** comportamiento de instrucción
  - Si procesador con sólo dos modos:
    - SO alojado convive en modo usuario con apps pero protegido
  - Si varios modos, SO alojado usa modo intermedio
    - Nivel(Hipervisor) > Nivel(SO alojado) > Nivel(Aplicaciones)
    - Pentium: VMM anillo 0, SO alojado anillo 1, Apps anillo 3
- Sobrecarga por procesamiento de *traps* (excepciones)

Virtualización del x86: ¿misión imposible?

- Popek & Goldberg (1974): distingue entre tipos de instrucciones
  - **privilegiadas**: causan *trap* si ejecutadas modo no privilegiado
  - **sensibles**:
    - **de control**: cambian configuración de recursos del sistema
    - **de comportamiento**: dependen de conf. de recursos del sistema
  - Máquina virtualizable: si sensibles subconjunto de privilegiadas
- Pentium no lo cumple
  - P.e. POPF se comporta diferente en modo usuario y privilegiado
    - No *trap*: sólo diferente comportamiento
  - Además TLB hardware y sin información de proceso
    - Dificulta la virtualización de la memoria
  - ¿Es imposible virtualizar Pentium?
    - Traducción binaria vs. paravirtualización

Traducción binaria

- *trap&translate&emulate* (VMware)
- Traductor dinámico sustituye instrucciones sensibles por *traps*
- Cumple 3ª condición P&G (**eficiencia**):
  - Traducción muy sofisticada y eficiente
  - Similar a compiladores JIT
  - Sólo para código del SO alojado
- Sobrecarga por procesamiento de *traps*
  - Traducción inteligente puede evitar algunos
- Resuelve el problema de virtualizar x86
  - Sin requerir modificar el SO alojado

Paravirtualización

- "Trampas": incumplimos 1ª condición P&G (**fidelidad**) (Xen):
  - Se modifica SO alojado para adaptarlo a VMM
- Modificaciones SO alojado llama a API del VMM
  - *hypercalls*
  - En vez de traducir en t. ejecución instr. sensibles conflictivas
    - Se eliminan a priori y se incluyen *hypercalls* en SO alojado
  - Solución más eficiente (disminuye nº de *traps*)
  - Pero requiere cambios en el SO
    - En algunos casos (Windows) es imposible
    - Y aunque sea posible, requiere incluirlos en cada versión del SO
  - Además cambios son específicos de cada VMM

Soporte hardware

- Procesadores actuales ofrecen soporte para virtualización
  - AMD-V, Intel-VT
  - Algunas características:
    - Instrucciones sensibles conflictivas → ***trap***
    - Nuevos modos de ejecución del procesador
      - 4 anillos para modo alojado y 4 para modo nativo
      - SO alojado en anillo 0 como en sistema no virtualizado
    - Redirección de interrupciones
    - Soporte de tablas de páginas anidadas
    - Soporte hardware de estructuras de control para las VMs
    - IOMMU
- Objetivo: eliminar traducción binaria o paravirtualización
  - Aunque podrían usarse puntualmente para optimizar

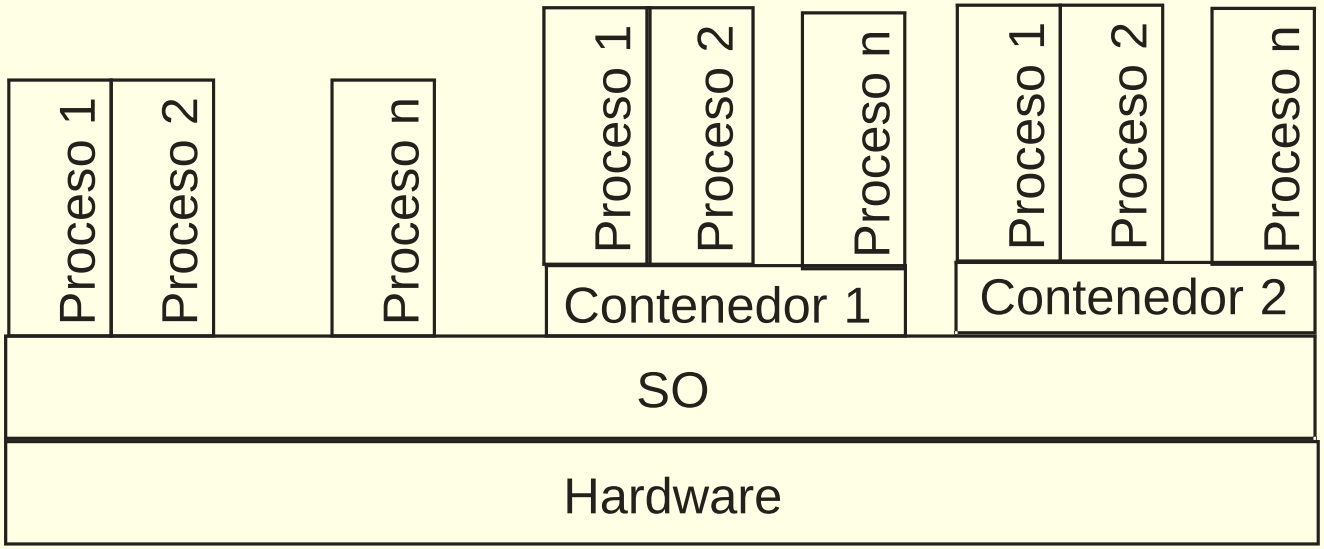
Aspectos de implementación de MV de sistema

- Los estudiaremos a lo largo de la asignatura
- Virtualización del procesador
  - Reparto de tiempo entre las MV
- Virtualización de la memoria
  - Memoria física SO alojado es espacio virtual creado por VMM
  - 3 niveles de direcciones (en vez de dos):
    - d. lógicas, d. físicas del sistema alojado, d. físicas reales
- Virtualización de la E/S
  - ¿DMA usa dir. física del sistema alojado o dir. física real?
- Virtualización de dispositivos, almacenamiento, red,…

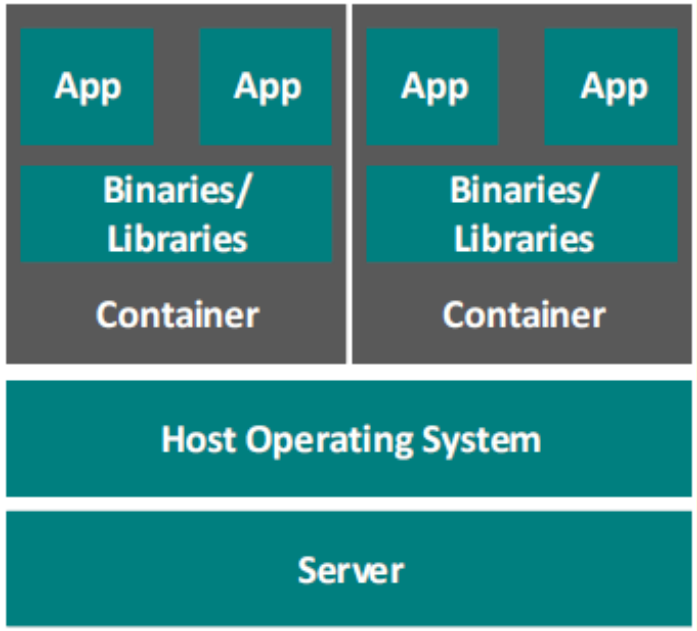
Virtualización a nivel SO: Contenedores

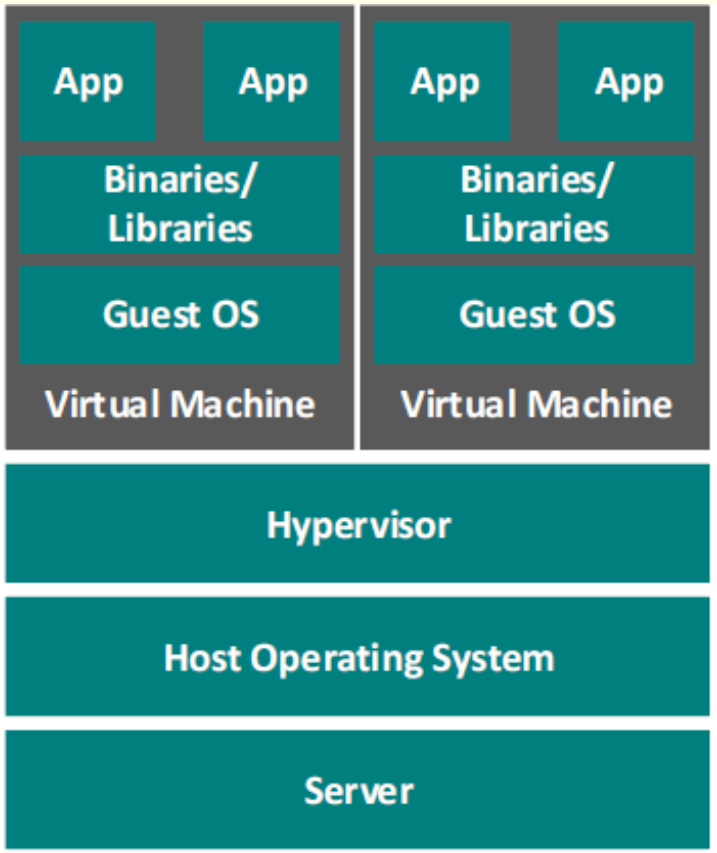
- Máquinas virtuales son "pesadas":
  - Requieren múltiples recursos por cada instancia
    - Una copia de SO alojado/instancia aunque sea el mismo
  - Arranque lento
- A veces no se requiere esa funcionalidad y se usa mismo SO
- Virtualización a nivel SO (LXC, Docker, *Solaris containers*, …):
  - El propio SO implementa la virtualización
  - SO crea múltiples sistemas sobre una máquina real
    - Sólo una copia del SO
  - Cada instancia suele denominarse **contenedor**
  - Cada contenedor recibe conjunto de recursos independiente
  - Cada contenedor usa un espacio de nombres independiente
  - Más eficiente que MV: pueden crearse muchas más instancias
  - Pero ↓ seguro: error SO hace vulnerables todos los contenedores

Virtualización a nivel SO



Virtualización a nivel SO vs hipervisor tipo II





https://wiki.aalto.fi/download/attachments/109392027/Linux%20Containers.pdf?version=2&modificationDate=1450548416183&api=v2

Virtualización a nivel SO: requisitos

- SO debe incluir funcionalidades para soporte de contenedores
- Contenedor: conjunto de procesos que usan sus propios recursos
- Se necesita:
  - SO gestione espacios de nombres separados para, entre otros:
    - PIDs
    - Nombres de usuarios
    - Nombre de la máquina
    - La red: nombre de interfaces de red, puertos,…
    - Las operaciones de montaje que realizan los procesos
  - SO asigne y controle uso de recursos a un grupo de procesos
    - CPU, memoria, discos, red,..
- Contenedor usa su propio espacio de nombres y grupo de procesos
- En Linux, LXC usan *namespaces* y *cgroups*

Máquinas virtuales de proceso

- Ejecuta como aplicación de SO y da soporte a un único proceso
- Proporciona juego de instrucciones ≠ real
- Uso habitual:
  - Crear entorno ejecución independiente de HW y SO real
    - Ejecución de proceso aislada de otras aplicaciones (*sandbox*)
  - Ejemplos
    - *Java Virtual Machine*, *Common Language Runtime* de .NET