Historia de los sistemas operativos

Gustavo Romero

Arquitectura y Tecnología de Computadores

12 de febrero de 2009

Índice

Lecturas recomendadas

Jean Bacon	Operating Systems (2, 26)
Abraham Silberschatz	Fundamentos de Sistemas Operativos (2)
William Stallings	Sistemas Operativos (2)
Andrew Tanuenbaum	Sistemas Operativos Modernos (1,12)

Motivación

- La **arquitectura** de un SO marca de forma vital su funcionamiento.
- Cada posible elección tendrá **consecuencias** ineludibles.
- Ejemplo: el **compromiso** velocidad/espacio:

```
-----------------------------------------------------------------------
#define BYTE_SIZE 8                     /* a byte contains 8 bits          */
int bit_count(int byte)                 /* count the bits in a byte        */
{
    int i, count = 0;
    for (i = 0; i < BYTE_SIZE; i++)     /* loop over the bits in a byte    */
        if ((byte >> i) & 1)
            ++count;                    /* if this bit is a 1, add to count */
        return count;                   /* return sum                      */
}
-----------------------------------------------------------------------
#define bit_count(b) (b&1)      + ((b>>1)&1) + ((b>>2)&1) + ((b>>3)&1) +
                     ((b>>4)&1) + ((b>>5)&1) + ((b>>6)&1) + ((b>>7)&1);
-----------------------------------------------------------------------
char bits[256] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2,..};
-----------------------------------------------------------------------
```

¿Qué es un sistema operativo?

- ¿Todos los programas que vienen con el ordenador al comprarlo? $\Longrightarrow$ no.
- ¿Todo lo que viene en el CD/DVD del creador del SO? $\Longrightarrow$ no.
- Los programas que nos permiten utilizar el ordenador (... *con suerte eficientemente*) $\Longrightarrow$ si.
  - Interfaz con el ordenador:
    - desarrollo de programas
    - ejecución de programas
    - acceso a dispositivos de E/S
    - acceso al sistema de ficheros
    - protección y seguridad
    - detección y respuesta a errores
    - contabilidad
  - Gestor de recursos.

Historia

- Primera generación (1945-55)
- Segunda generación (1955-65)
- Tercera generación (1965-80)
- Cuarta generación (1980-hoy)

Primera generación (1945-55)

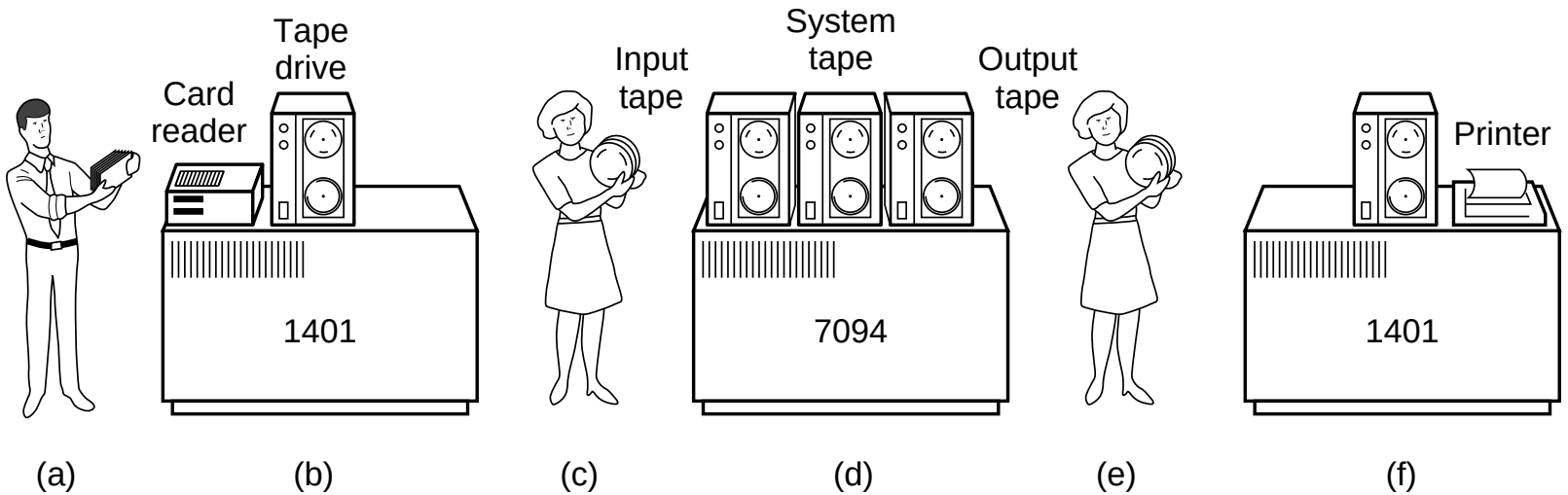
Tubos de vacio y paneles

- Utilidad: máquinas de cálculo.
- Tecnología: dispositivos mecánicos $\Rightarrow$ tubos de vacio y paneles.
- Método de programación: cables $\Rightarrow$ interruptores y tarjetas perforadas.
- Diseño/construcción/operación/programación/-mantenimiento: genios como Aiken, von Newman o Mauchley.

Segunda generación (1955-65)

Transistores y sistemas por lotes

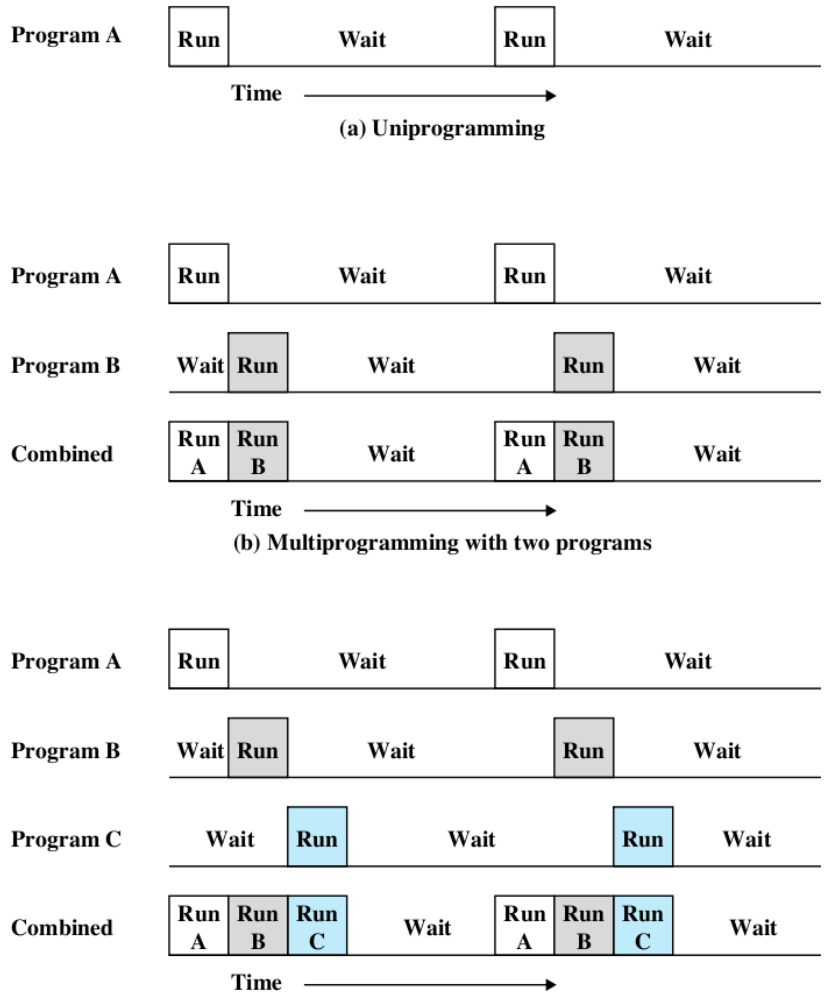
- Utilidad: cálculo científico e ingeniería.
- Tecnología: la invención del transistor redujo su tamaño y precio y los popularizó ⇒ **mainframes**/**IBM 1401**/**7094**.
- Método de programación: **ensamblador** y lenguajes de alto nivel (**FORTRAN**) sobre tarjetas perforadas.
- Paso de procesamiento **secuencial** a procesamiento **por lotes**.
- Ejemplos: **FMS** y **IBSYS**.



Tercera generación (1965-80)

Circuitos integrados y multiprogramación

- 2 usos principales:
  - cálculo científico e ingeniería.
  - procesamiento de carácteres.
- **Circuito integrado** $\Rightarrow$ +barato $\Rightarrow$ +popular $\Rightarrow$ **IBM 360**, **GE-645**, **DEC PDP-1**.
- Logros destacables:
  - **multiprogramación**.
  - **spooling**.
  - **tiempo compartido**.
- Ejemplos: **OS/360**, **CTSS**, **MULTICS**, **UNIX**.



Cuarta generación (1980-hoy)

Ordenador personal (era μ)

- **(V)LSI** $\Rightarrow$ ++barato $\Rightarrow$ ++popular $\Rightarrow$ **IBM PC**.
- μ**P**: 8080, Z80, 8086, 286, 386, 486, Pentium, Core 2, Athlon, Alpha, Ultrasparc.
- Logros destacables:
  - **GUI**.
  - **SO de red**.
  - **SMP**.
  - **SO distribuidos**.
- Ejemplos: **UNIX**, **CP/M**, **MS-DOS**, **Linux**, **MacOS**, **XP**, **NT**, **Vista**...

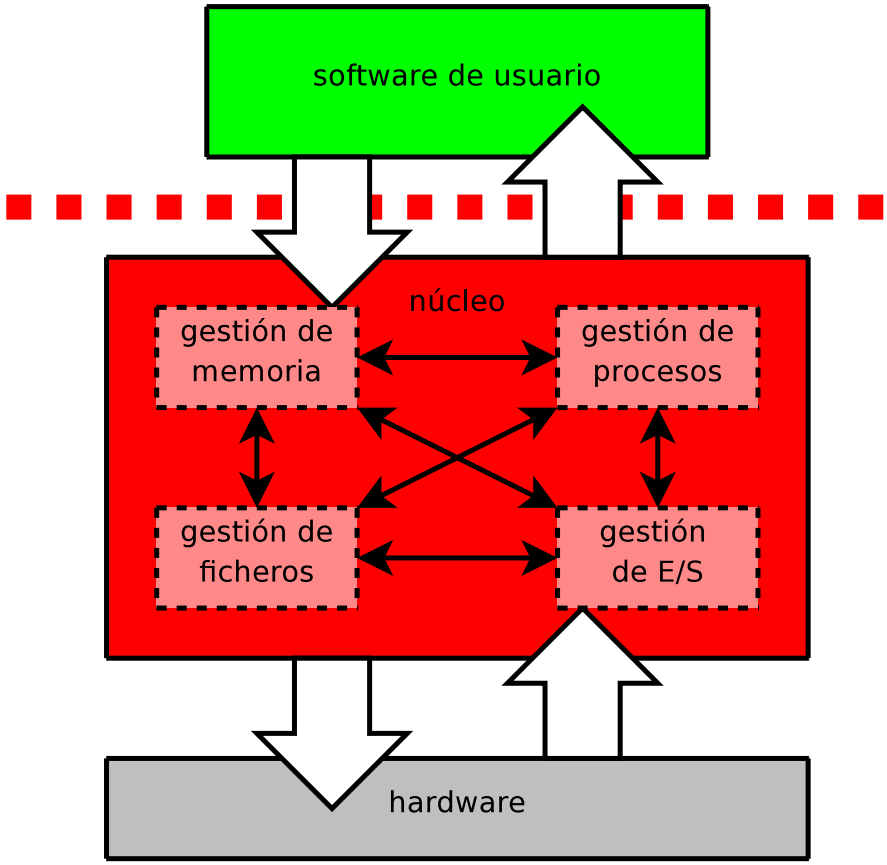
Clasificación de SO según su estructura

¿Cómo se organiza internamente el SO?

- Clasificación:
  - Desestructurados.
  - Estructura simple:
    - monolíticos
    - capas
    - modulares
  - Estructura cliente/servidor:
    - micronúcleo
    - exonúcleo
  - Máquina virtual.
  - Híbridos.
- Tendencias:
  - Núcleos extensibles.
  - Multiservidores sobre un micronúcleo.
  - Núcleos híbridos.

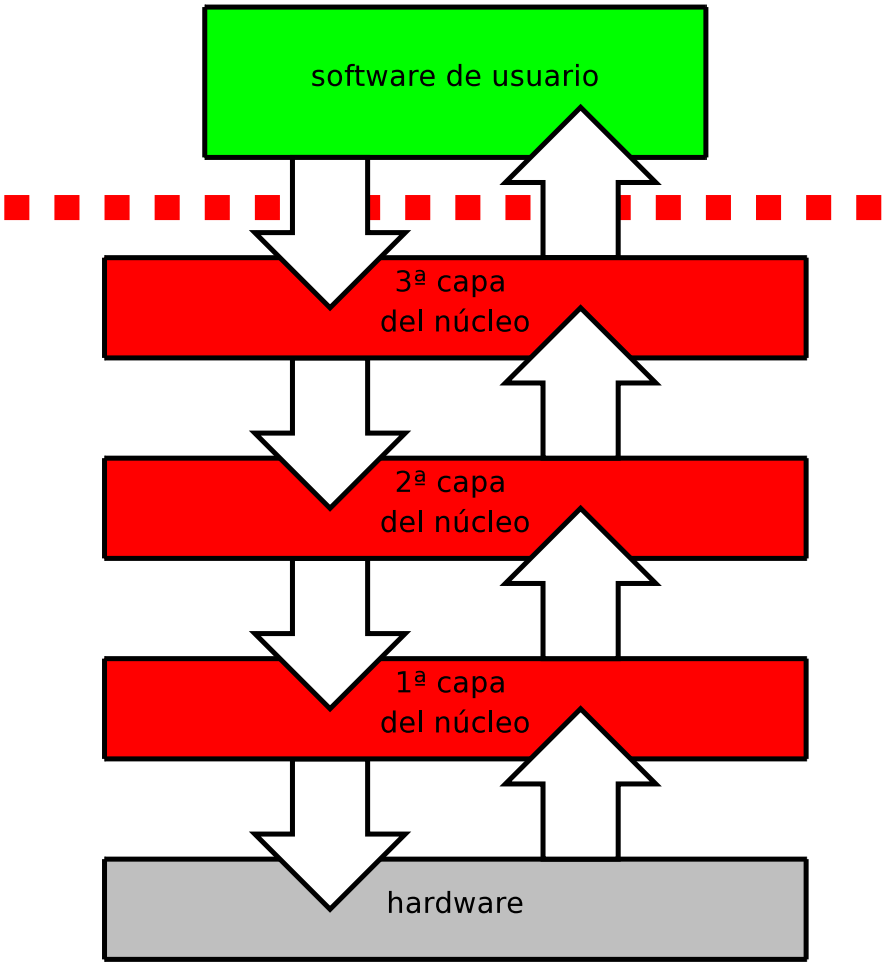
Monolítico

- El SO completo se ejecuta en modo protegido.
- Nula protección entre los componentes.
- Ventajas:
  - Economía de cambios de contexto $\Rightarrow$ **+eficiente**.
- Inconvenientes:
  - Falta de protección $\Rightarrow$ **-fiabilidad** (controladores).
  - Manejo de la complejidad: Es más sencillo escribir 10^3 programas de 10^3 líneas que uno de 10^6.



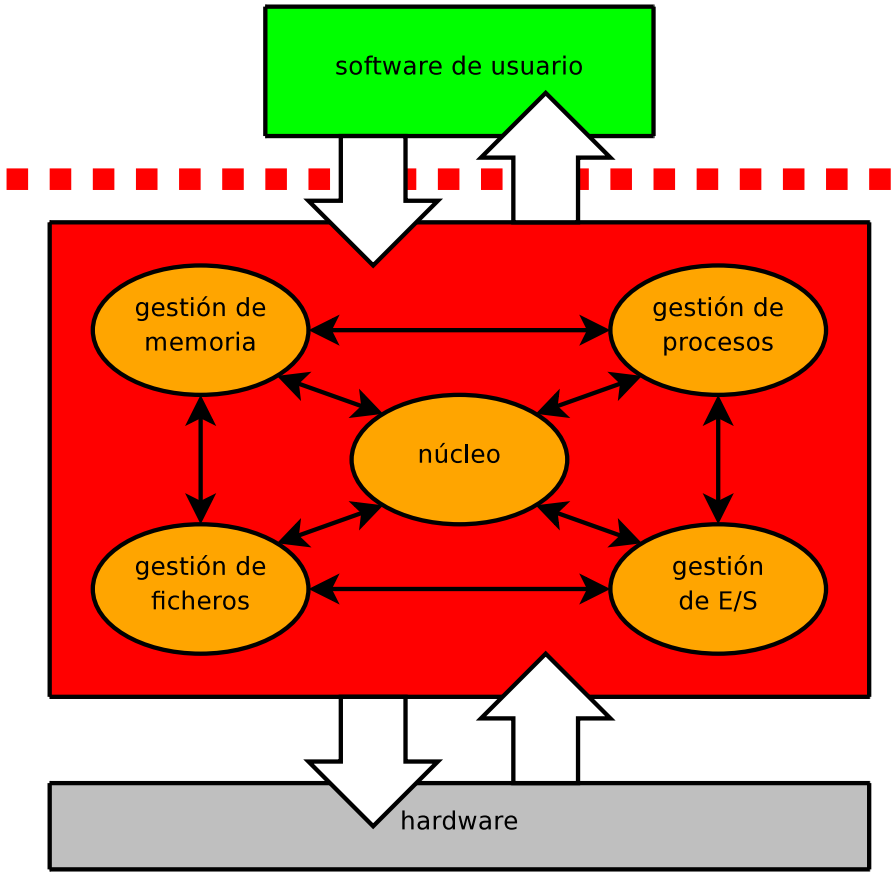
Capas/Niveles

- El SO completo se ejecuta en modo protegido.
- Escasa protección entre los componentes.
- Ventajas:
  - Economía de cambios de contexto ⇒ **+eficiente**.
  - Menor complejidad.
- Inconvenientes:
  - Falta de protección ⇒ **-fiabilidad** (controladores).
  - Menos flexible que monolítico.
- ¿Cómo subdividir en capas?



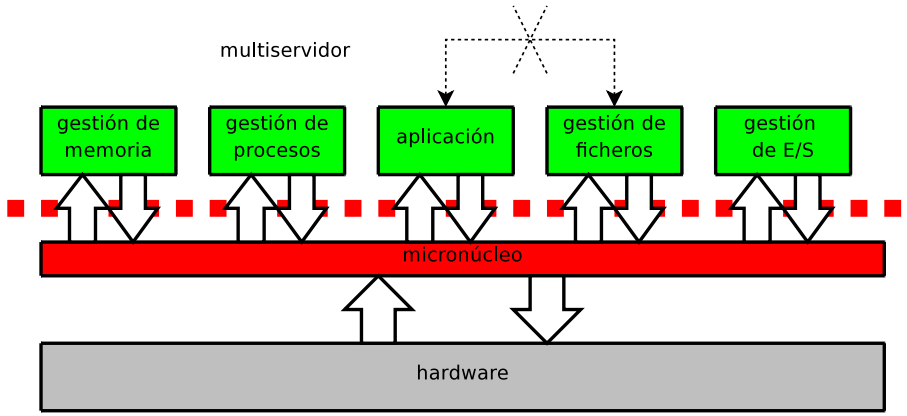
Modular

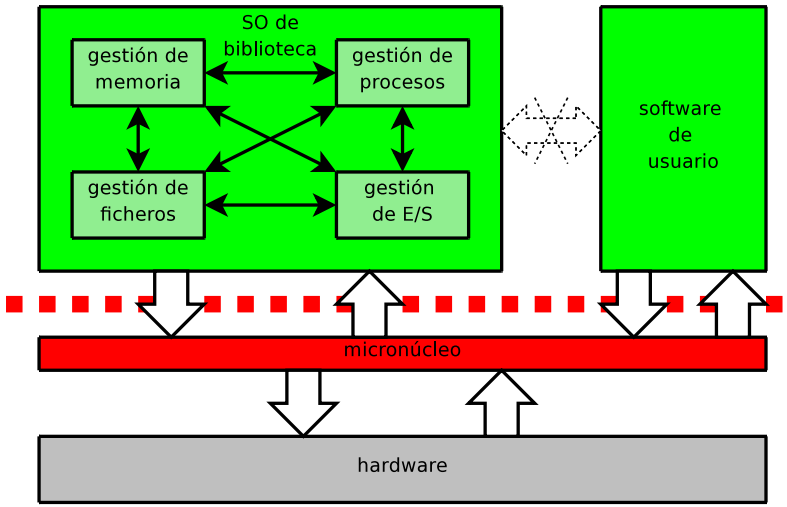
- El SO completo se ejecuta en modo protegido.
- Escasa protección entre los componentes.
- Ventajas:
  - Economía de cambios de contexto $\Rightarrow$ **+eficiente**.
  - Menor complejidad.
- Inconvenientes:
  - Falta de protección $\Rightarrow$ **-fiabilidad** (controladores).
  - Menos flexible que monolítico.
- ¿Qué colocar en el núcleo y qué en módulos?



Micronúcleo

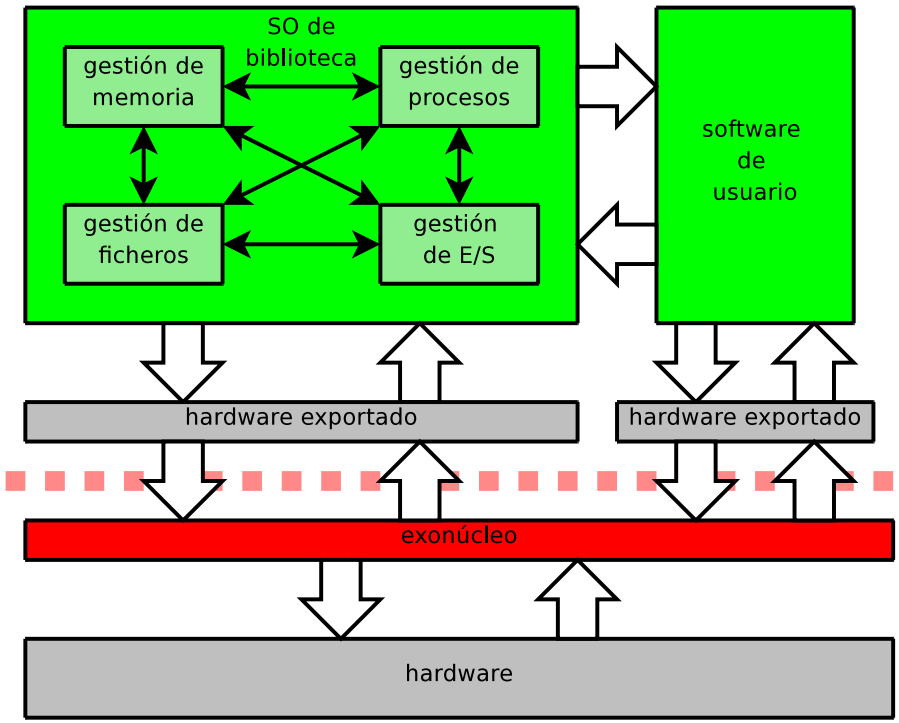
- Una mínima parte del SO se ejecuta en modo protegido.
- Ventajas:
  - Perfecta protección entre componentes ⇒ **+fiabilidad**.
  - Manejo de la complejidad.
  - Facilidad de programación.
- Inconvenientes:
  - Sobrecarga en las comunicaciones ⇒ **-eficiencia**.





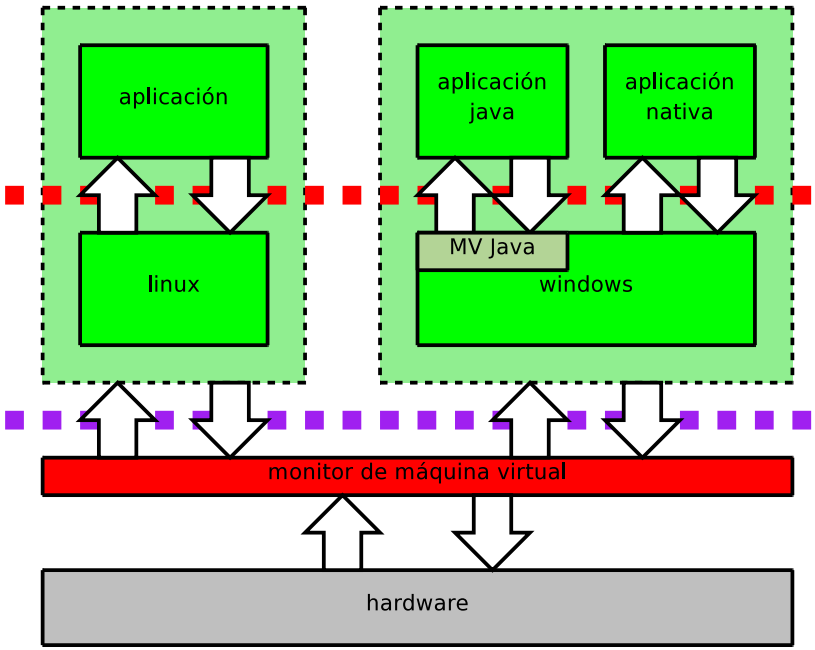
Exonúcleo

- Apenas existe SO, sólo un gestor de recursos.
- Dejamos que el software acceda directamente al hardware.
- Ventajas:
  - Perfecta protección entre componentes ⇒ **+fiabilidad**.
  - Acceso directo al hardware ⇒ **máxima eficiencia**
- Inconvenientes:
  - Pobre reutilización de código.



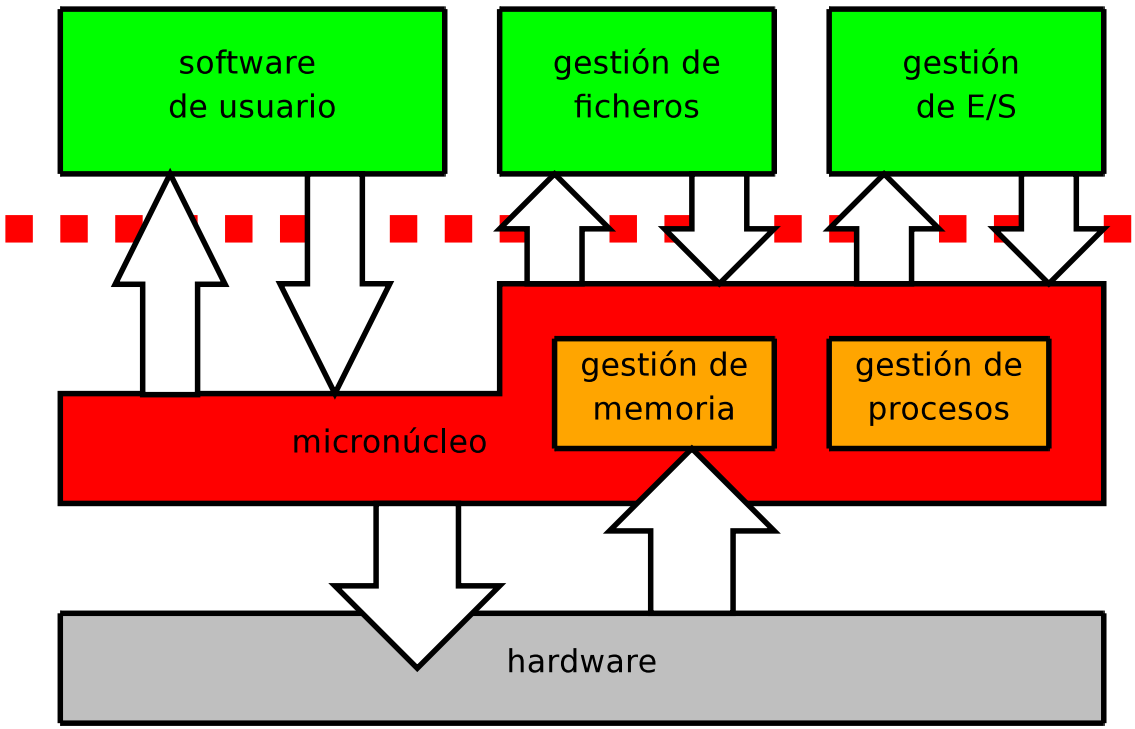
Máquina virtual

- N **copias virtuales** de la máquina real:
  - Software: Bochs, Qemu, VMWare, Xen.
  - Hardware: VMWawe, Xen.
- IBM VM/370 (1972).
- Ventajas:
  - Perfecta protección entre componentes ⇒ **+fiabilidad**.
  - Mejor aprovechamiento del hardware.
  - Máxima reutilización de código.
- Inconvenientes:
  - La simulación del hardware real es costosa ⇒ **poco eficiente**

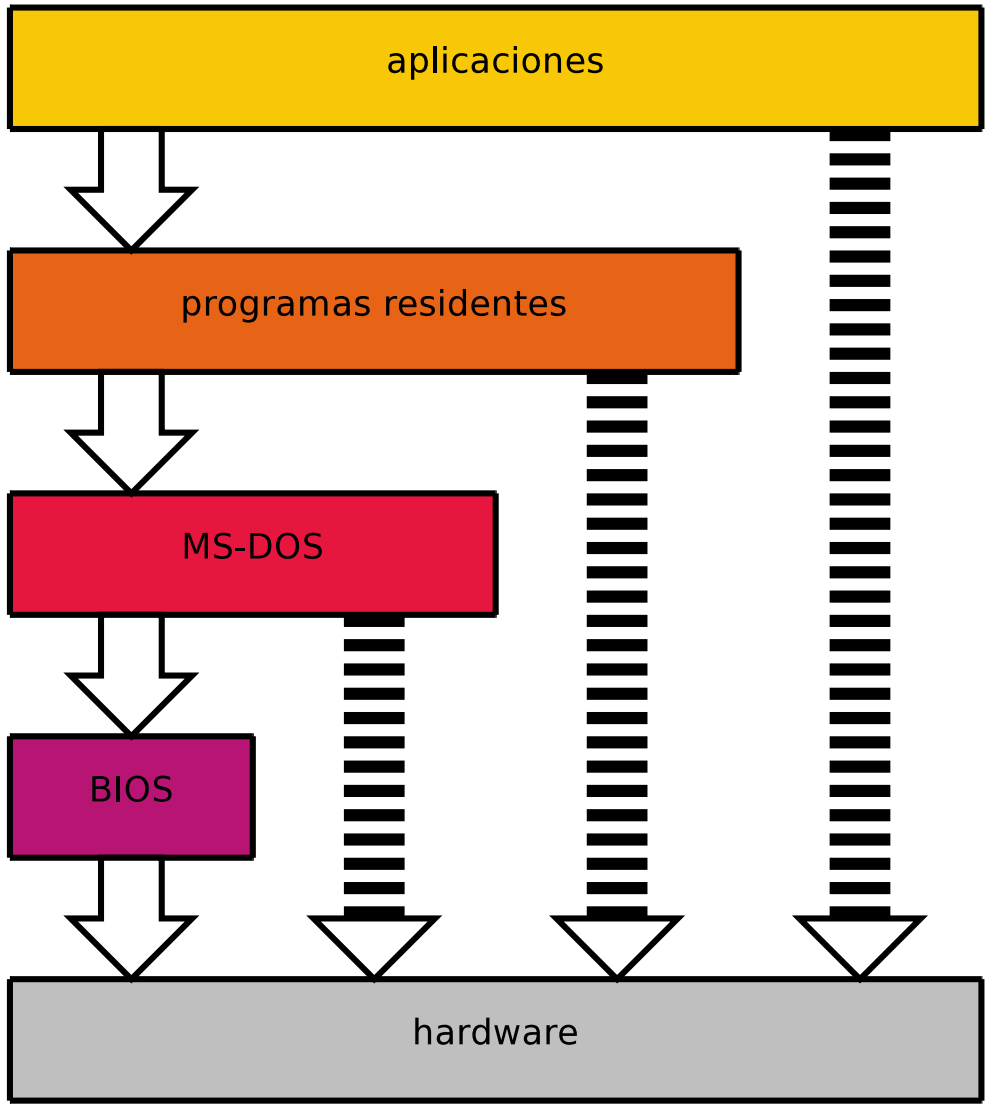


Hídrida

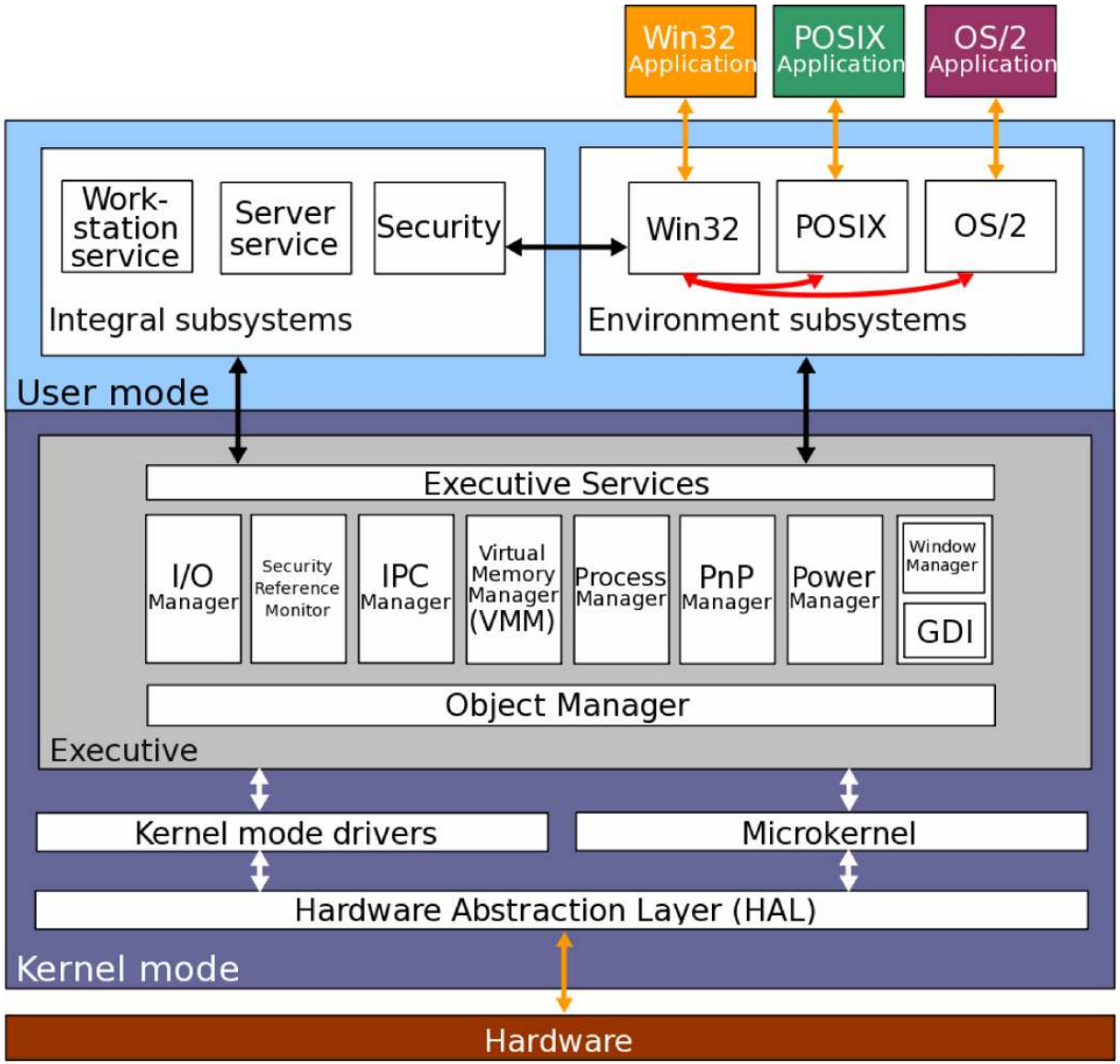
- Mezcla más frecuente: **micronúcleo + monolítico**.
- Ventaja: $\Longrightarrow$ ganamos velocidad respecto a micronúcleo.
- Inconveniente: $\Longrightarrow$ perdemos protección entre componentes.



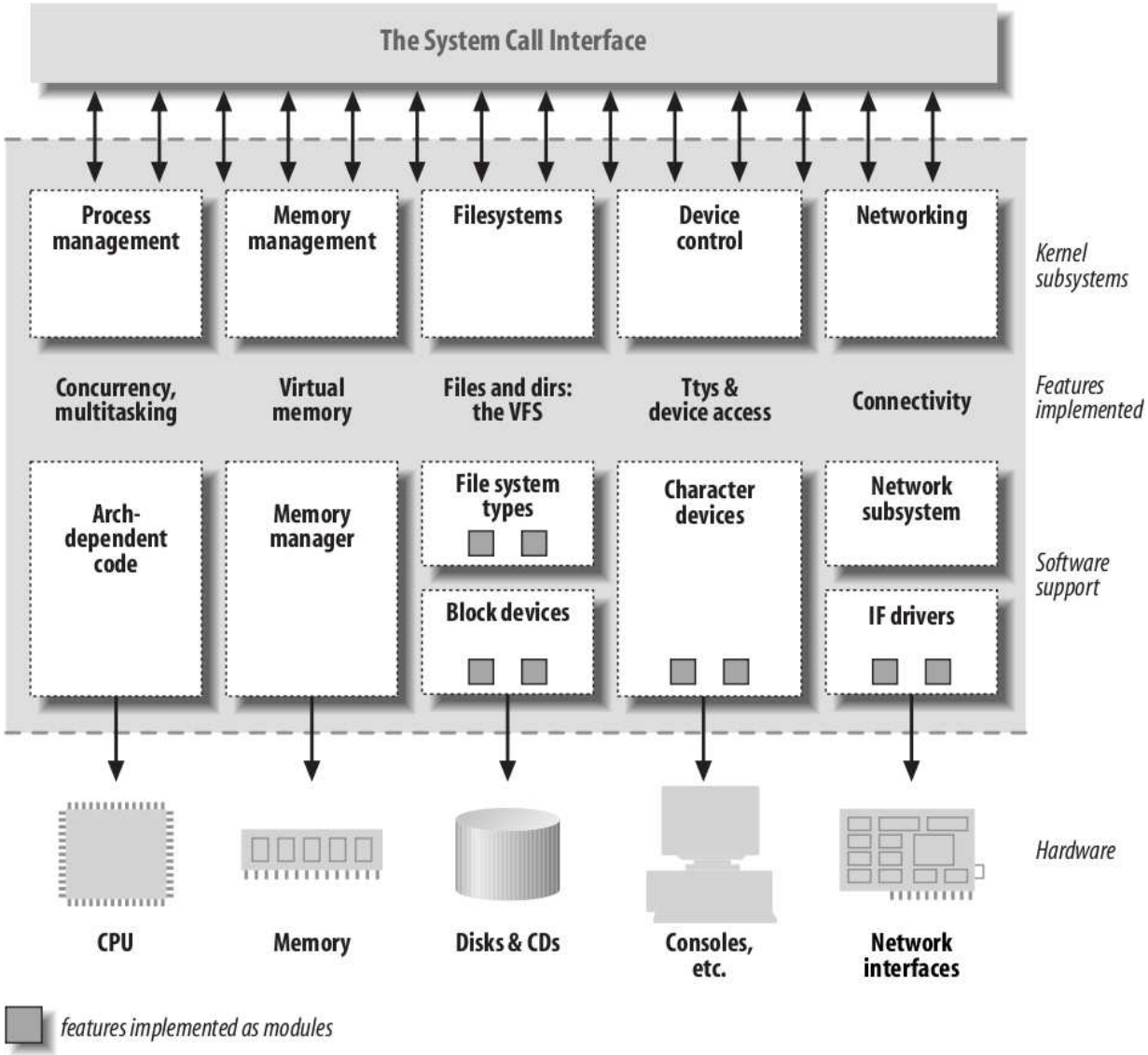
MS-DOS



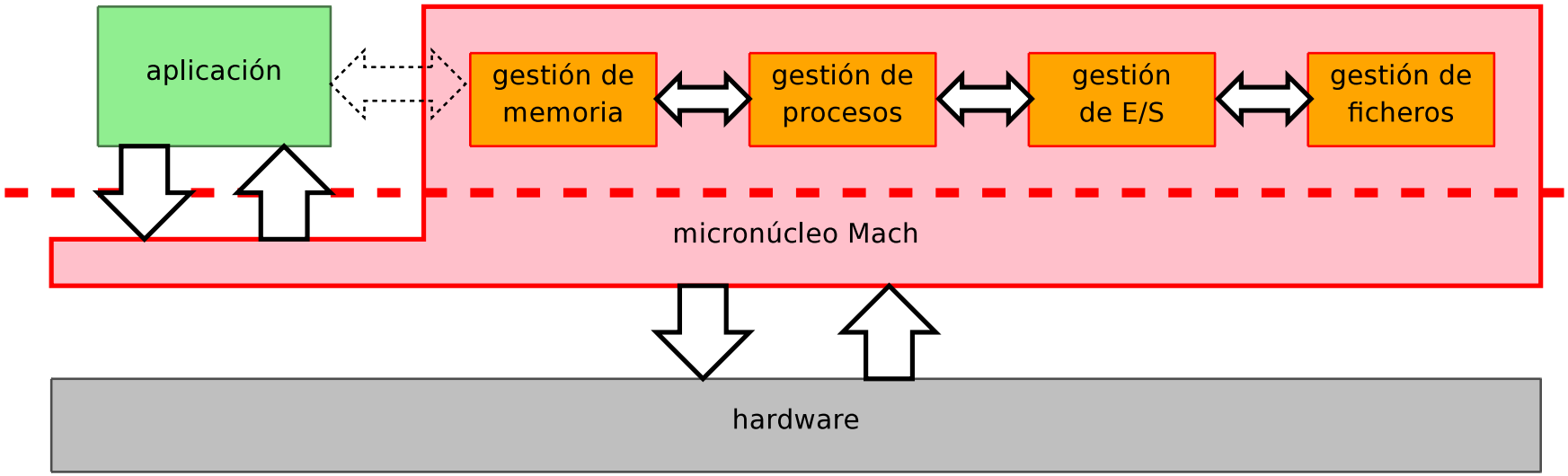
Windows 2000



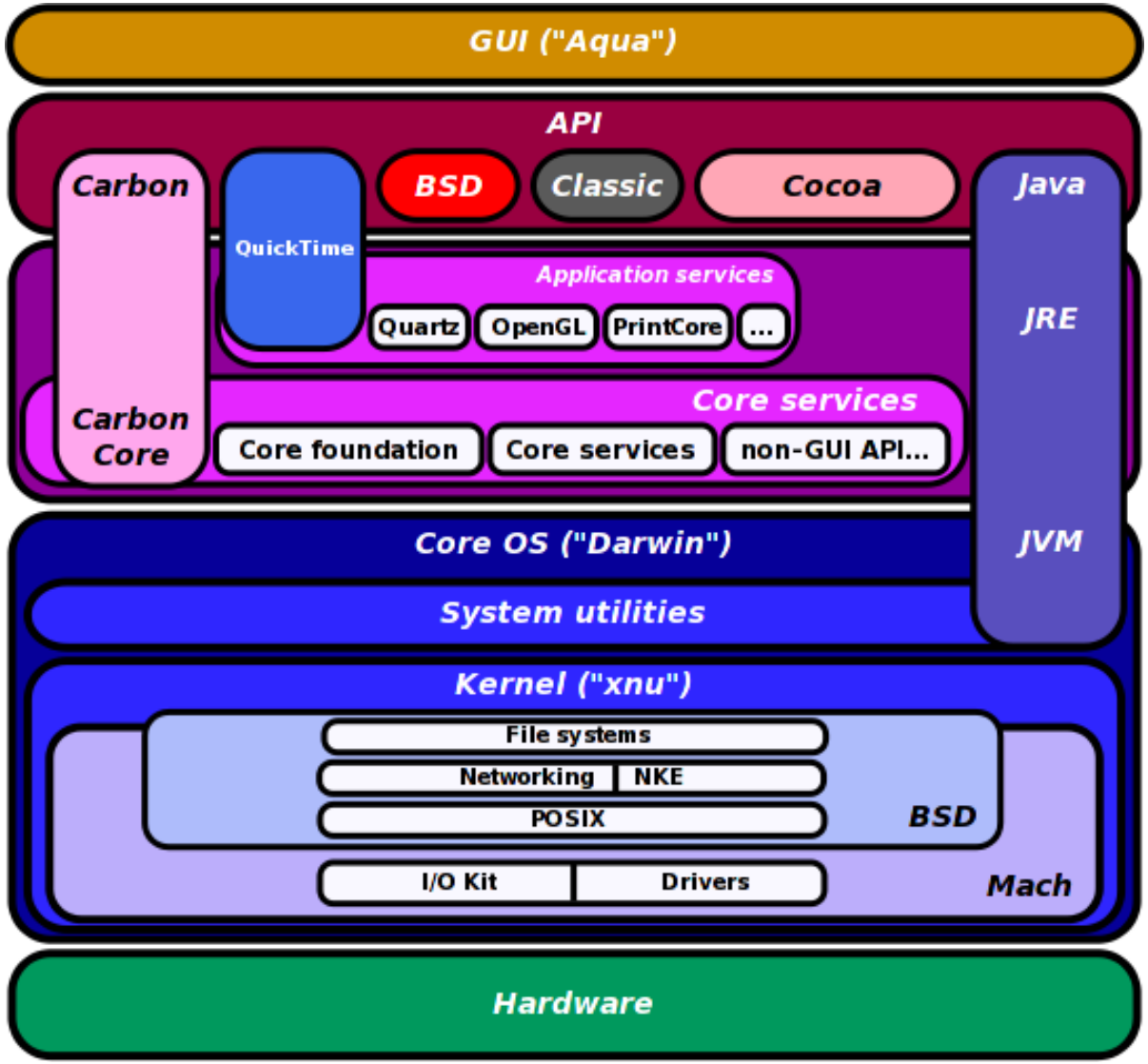
Linux


The System Call Interface
Process management
Memory management
Filesystems
Device control
Networking
Kernel subsystems
Concurrency, multitasking
Virtual memory
Files and dirs: the VFS
Ttys & device access
Connectivity
Features implemented
Arch-dependent code
Memory manager
File system types
Block devices
Character devices
Network subsystem
IF drivers
Software support
CPU
Memory
Disks & CDs
Consoles, etc.
Network interfaces
Hardware
features implemented as modules

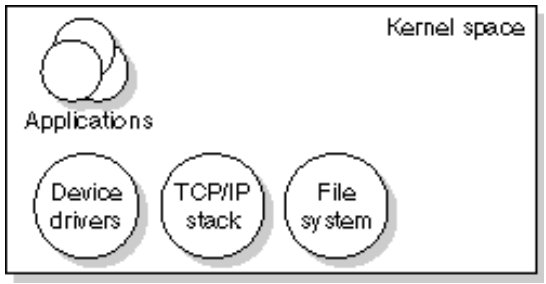
Mach

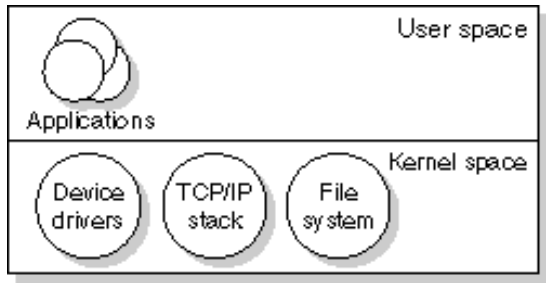

aplicación
gestión de memoria
gestión de procesos
gestión de E/S
gestión de ficheros
micronúcleo Mach
hardware

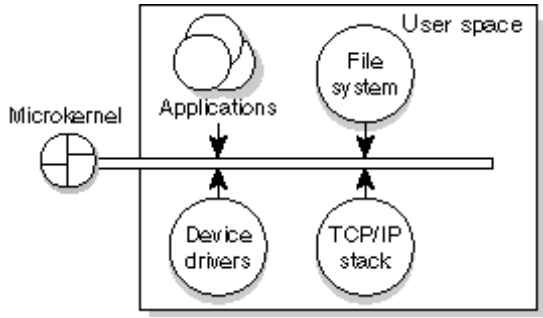
MacOS X

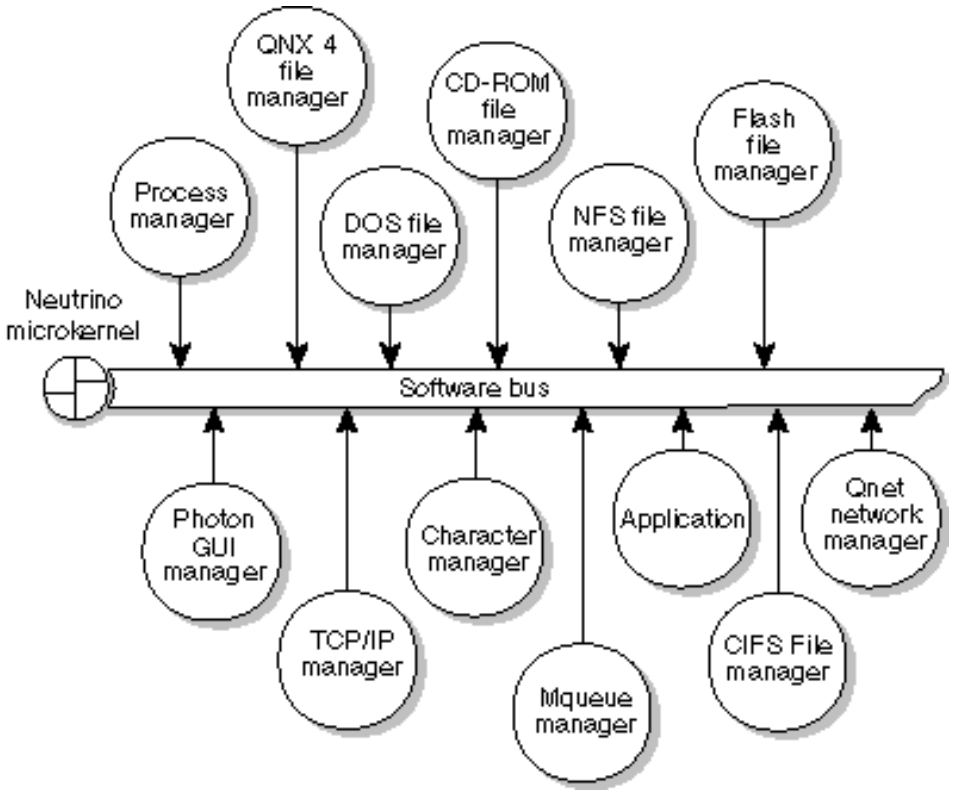

GUI ("Aqua")
API
Carbon
QuickTime
BSD
Classic
Cocoa
Java
Application services
Quartz
OpenGL
PrintCore
...
JRE
Core services
Carbon Core
Core foundation
Core services
non-GUI API...
Core OS ("Darwin")
JVM
System utilities
Kernel ("xnu")
File systems
Networking
NKE
POSIX
BSD
I/O Kit
Drivers
Mach
Hardware

QNX


Kernel space
Applications
Device drivers
TCP/IP stack
File system

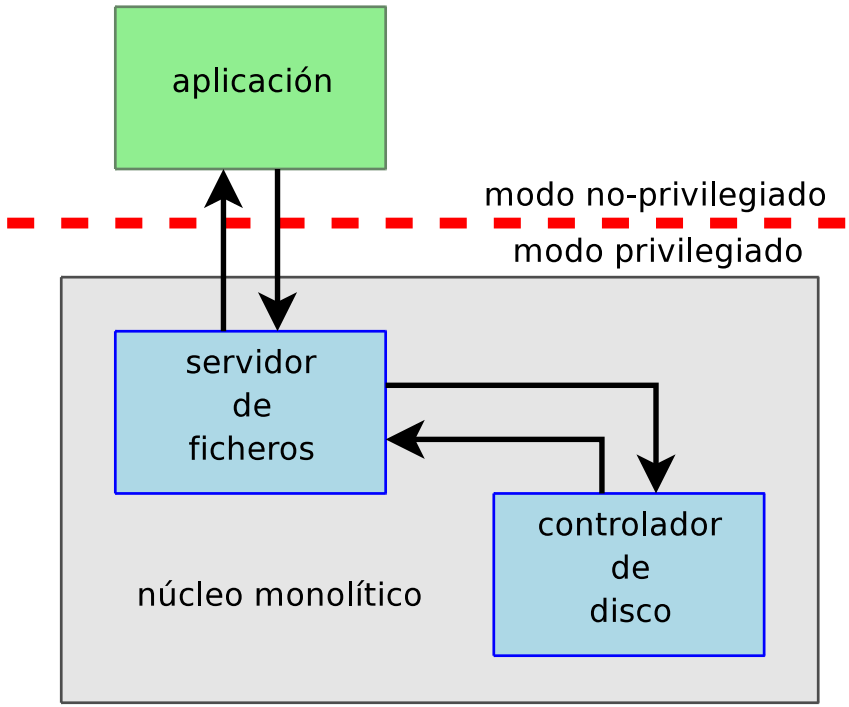

User space
Applications
Kernel space
Device drivers
TCP/IP stack
File system


User space
Microkernel
Applications
File system
Device drivers
TCP/IP stack


QNX 4 file manager
CD-ROM file manager
Flash file manager
Process manager
DOS file manager
NFS file manager
Neutrino microkernel
Software bus
Photon GUI manager
TCP/IP manager
Character manager
Mqueue manager
Application
CIFS File manager
Qnet network manager

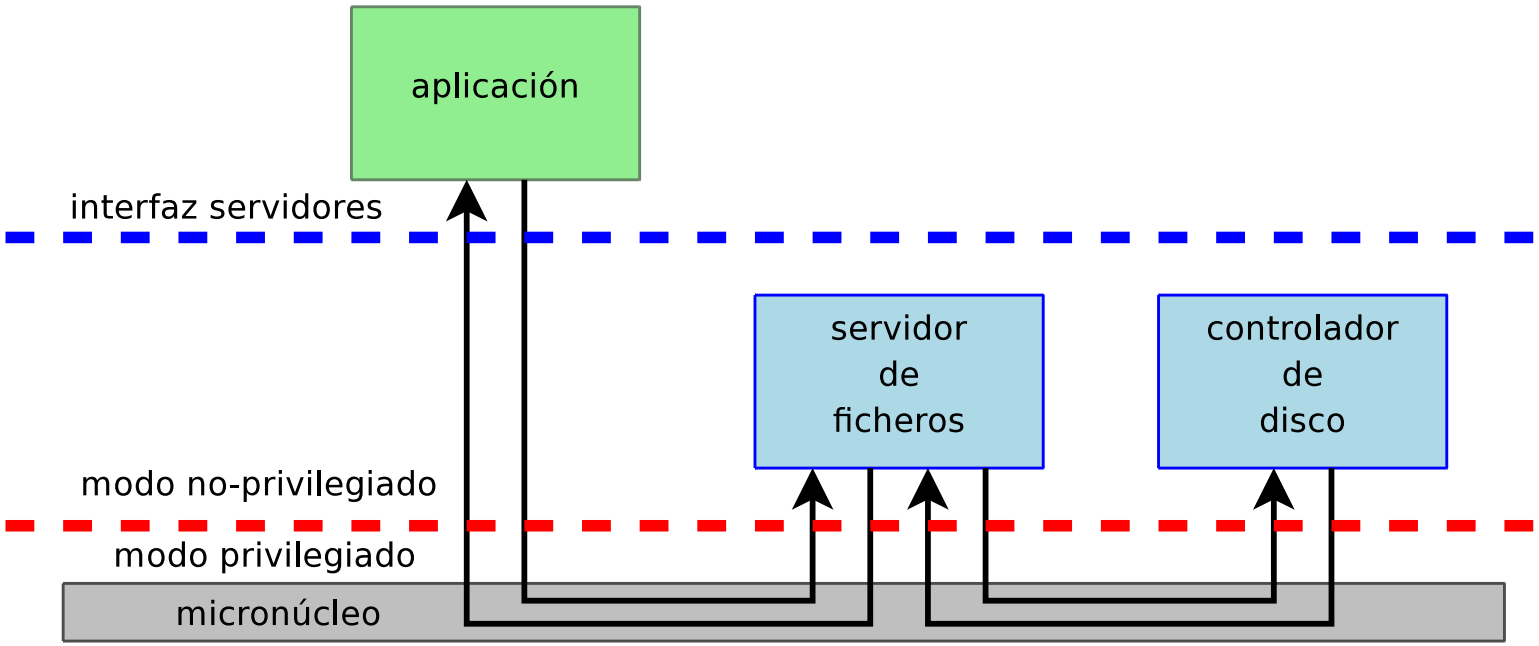
Coste estructural: monolítico

- **1 llamada al sistema:**
  - entrada al núcleo.
  - cambio al espacio de direcciones del núcleo.
  - recuperar el espacio de direcciones original.
  - salida del núcleo.
- **1 llamada a procedimiento:** llamada y retorno en el interior del espacio de direcciones del núcleo y pudiendo compartir información.

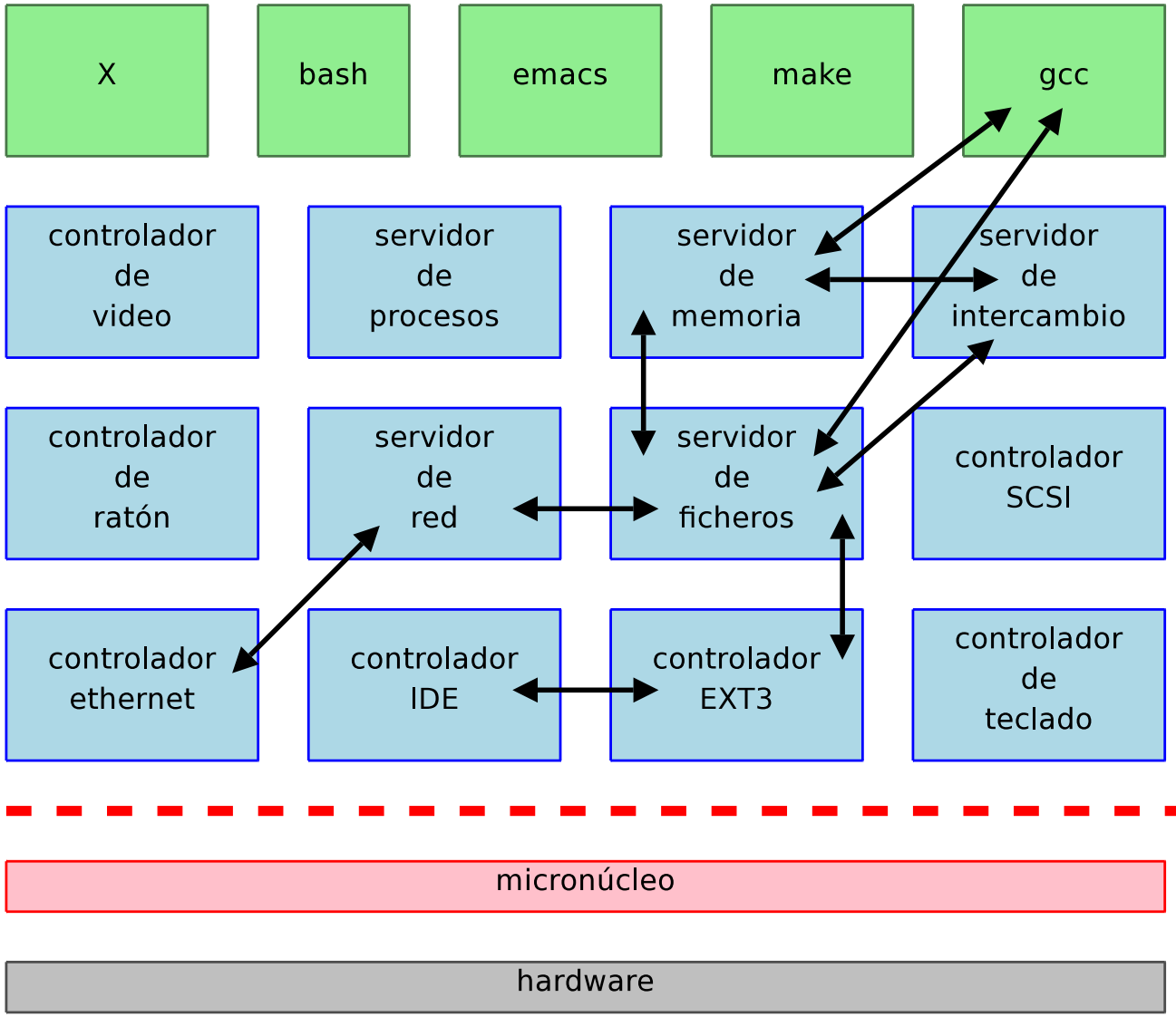


Coste estructural: micronúcleo

- **4 llamadas al sistema:**
  - entrada al micronúcleo.
  - cambio al espacio de direcciones del micronúcleo.
  - transferencia del mensaje.
  - recuperar el espacio de direcciones original.
  - salida del micronúcleo.

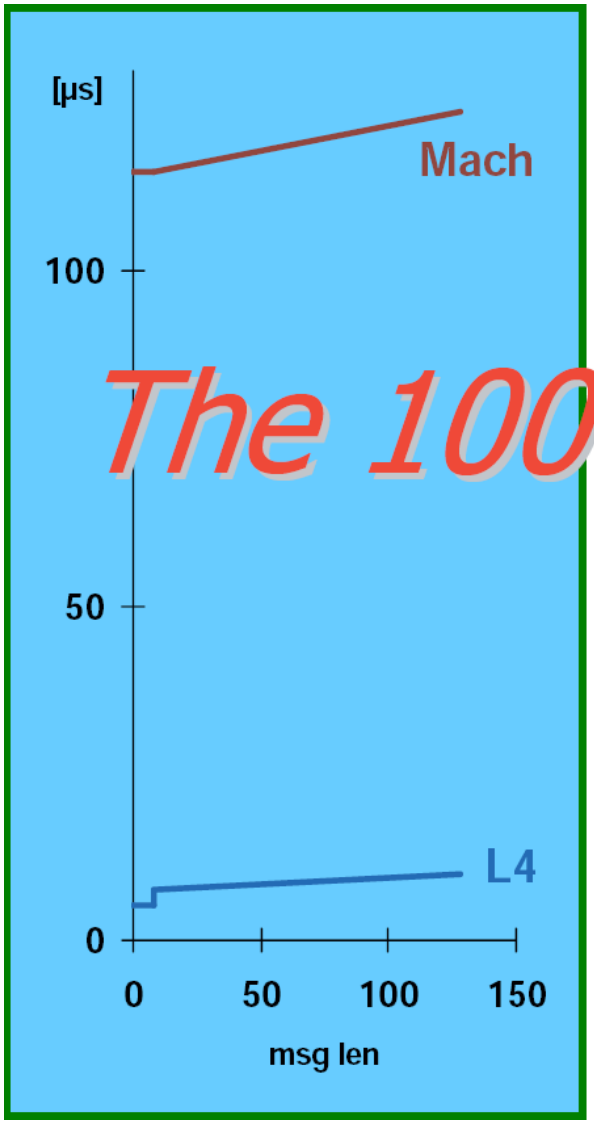


Coste estructural: multiservidor

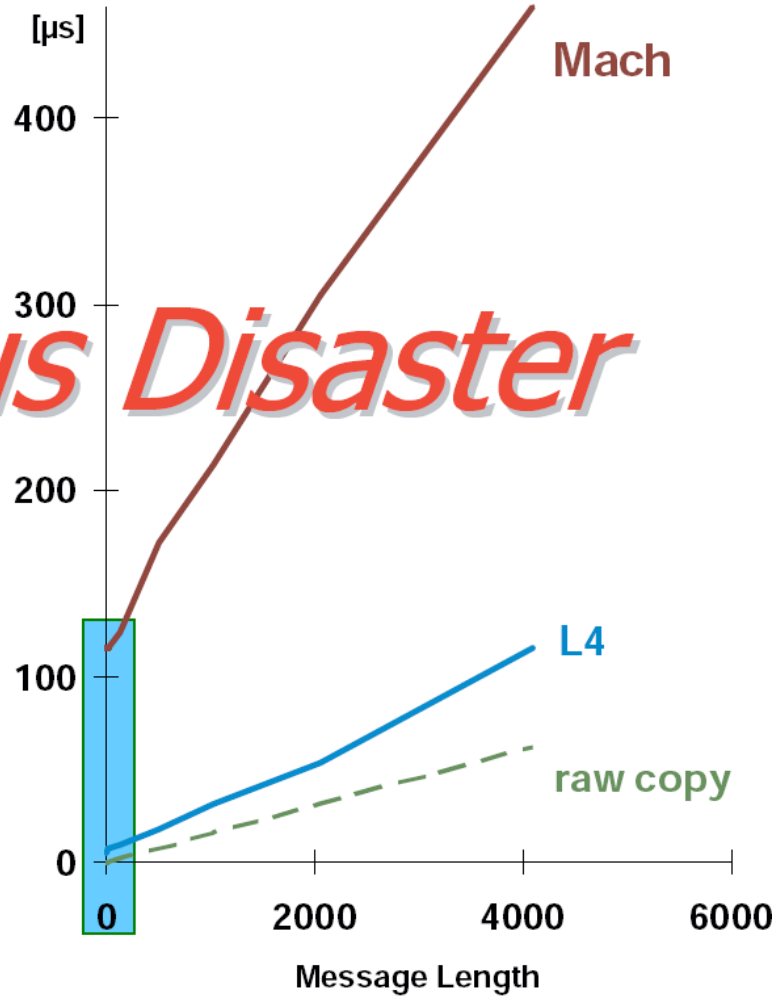


El desastre de los 100 μs (micronúcleos de 1ª generación)

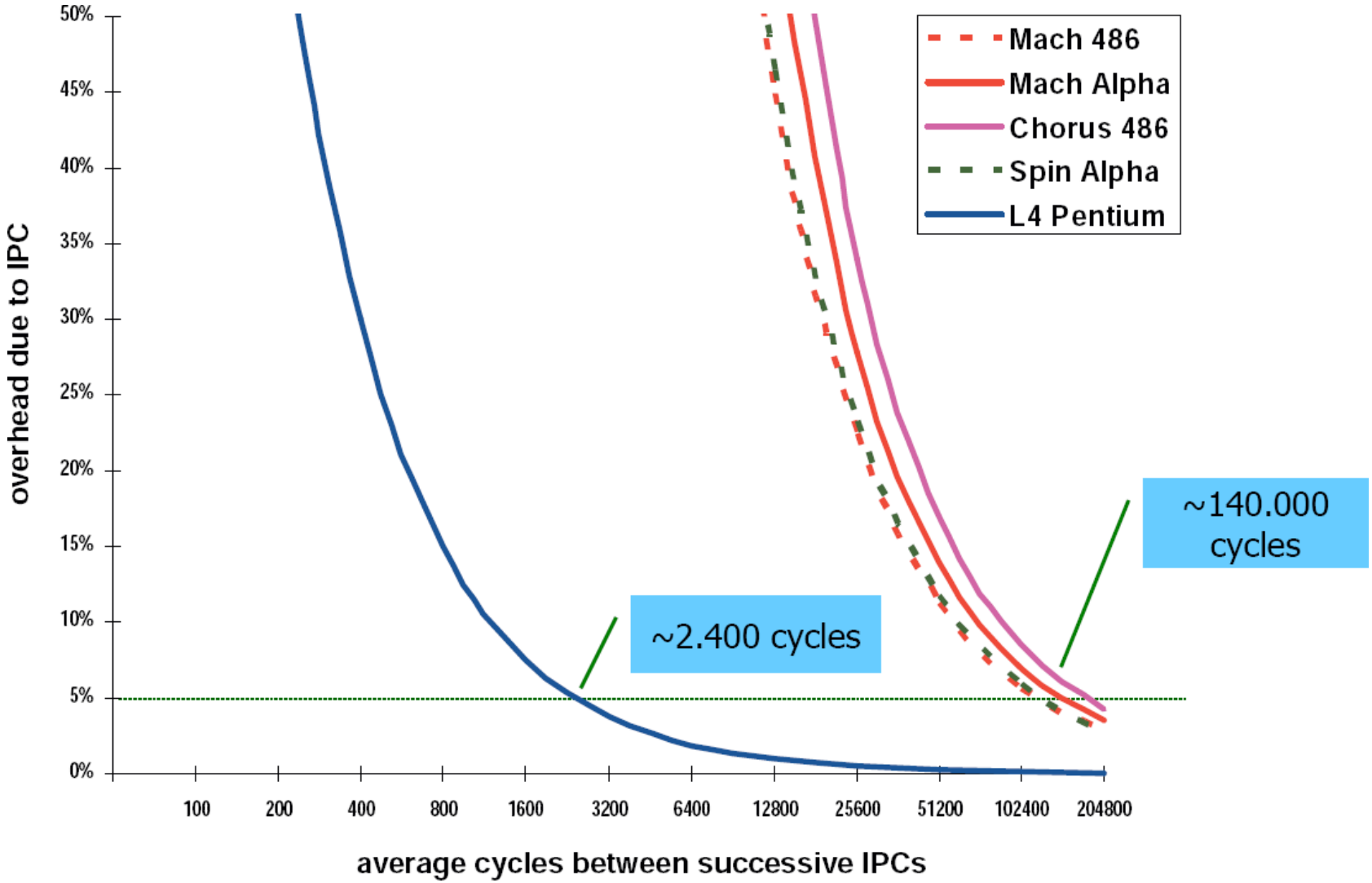
SYSCALL $\leftrightarrow$ *RPC* $\approx 2 \times$ *IPC*: Mach_{IPC}=115μs, Linux_{IPC}=20μs, L4_{IPC}=5μs (486 50MHz)



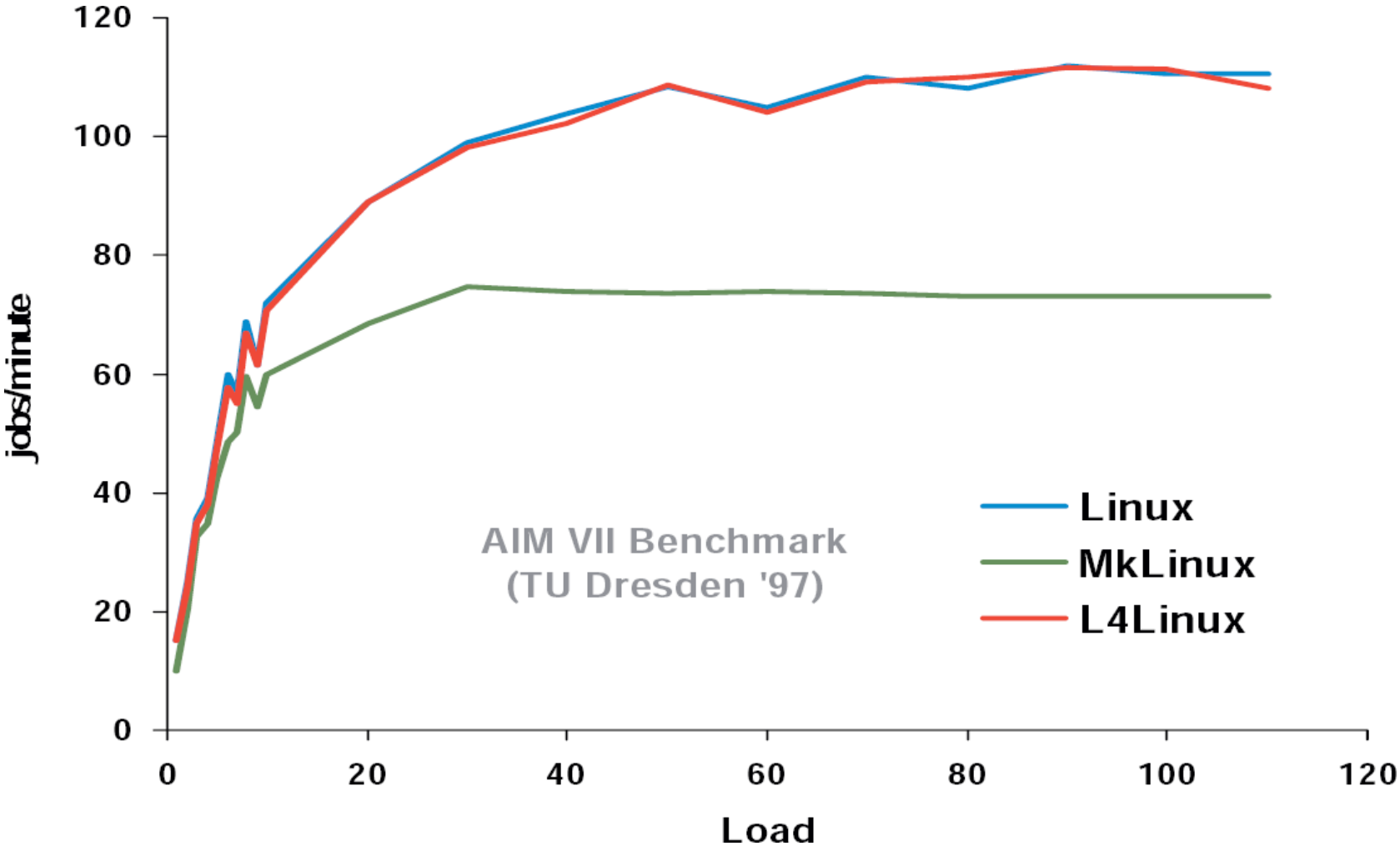
The 100-µs Disaster



Sobrecarga por comunicación entre procesos



L4Linux


120
100
80
60
40
20
0
jobs/minute
0
20
40
60
80
100
120
Load
AIM VII Benchmark
(TU Dresden '97)
Linux
MkLinux
L4Linux

Coste estructural: cambio de contexto

- Linux 2.4.21: 13200 ciclos/5.4μs en un Pentium 4 a 2.4GHz
- L4 (Liedtke, Achieved IPC performance):

	Pentium		**R4600**		**Alpha**	
	instructions	*cycles*	*instructions*	*cycles*	*instructions*	*cycles*
enter kernel mode (trap)	1	52	23	25	1	5
ipc code	43	23	47	50	60	38
segment register reload	4	16	–	–	–	–
exit kernel mode (ret)	1	20	9	11	1	2
total	50	**121**	79	**86**	62	**45**
	166 MHz: 0.73 μs		100 MHz: 0.86 μs		433 MHz: 0.10 μs	

		Pentium		**R4600**		**Alpha**	
		cache lines	*L1 cache usage*	*cache lines*	*L1 cache usage*	*cache lines*	*L1 cache usage*
kernel code	(I-cache)	6	2.3%	14	2.7%	13	5.1%
global kernel data	(D-cache)	2	0.8%	1	0.2%	0	0.0%
thread kernel data	(D-cache)	2×2	1.6%	2×2	0.8%	2×2	1.6%
total	(I+D-cache)	**12**	**2.3%**	**19**	**1.9%**	**17**	**3.3%**