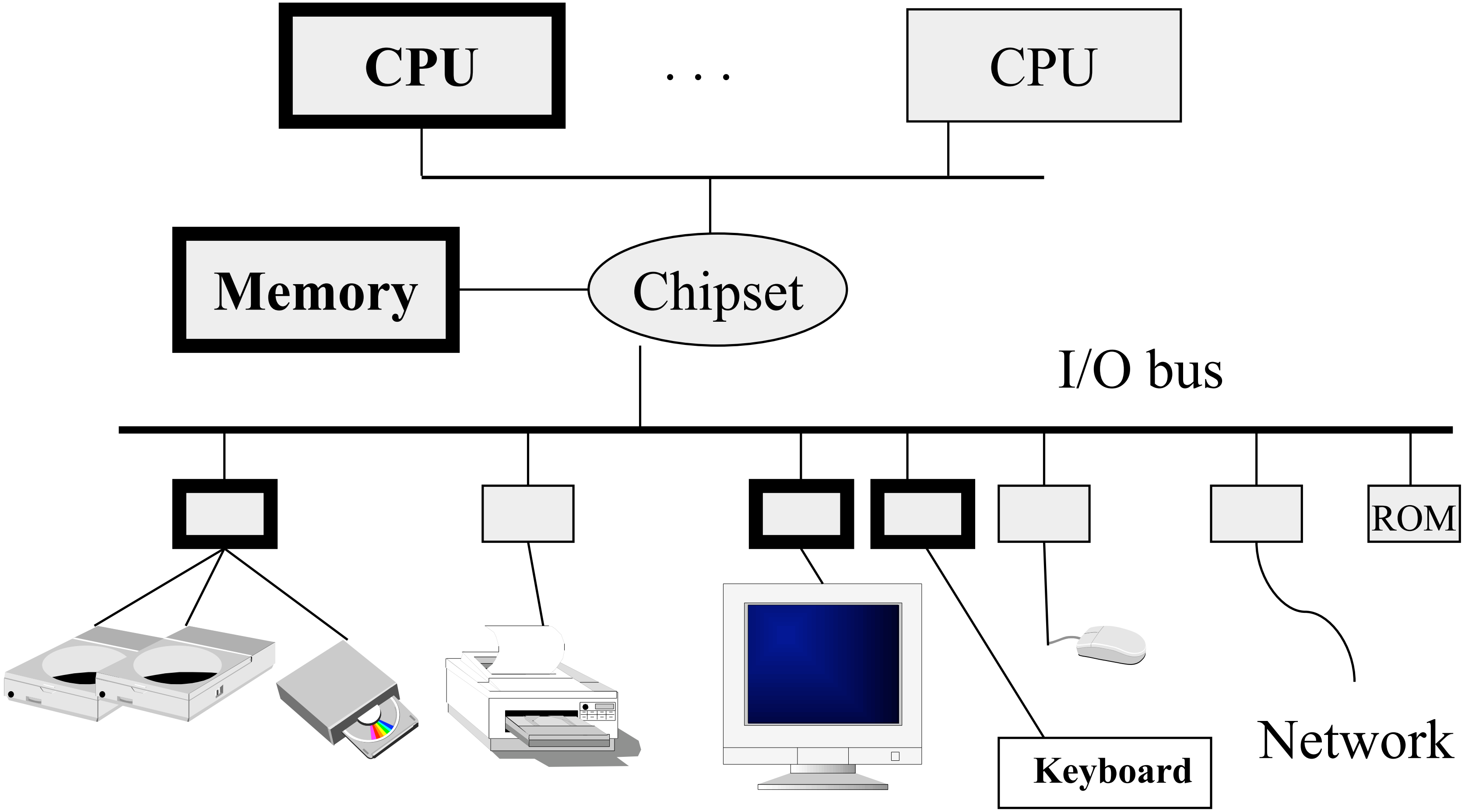
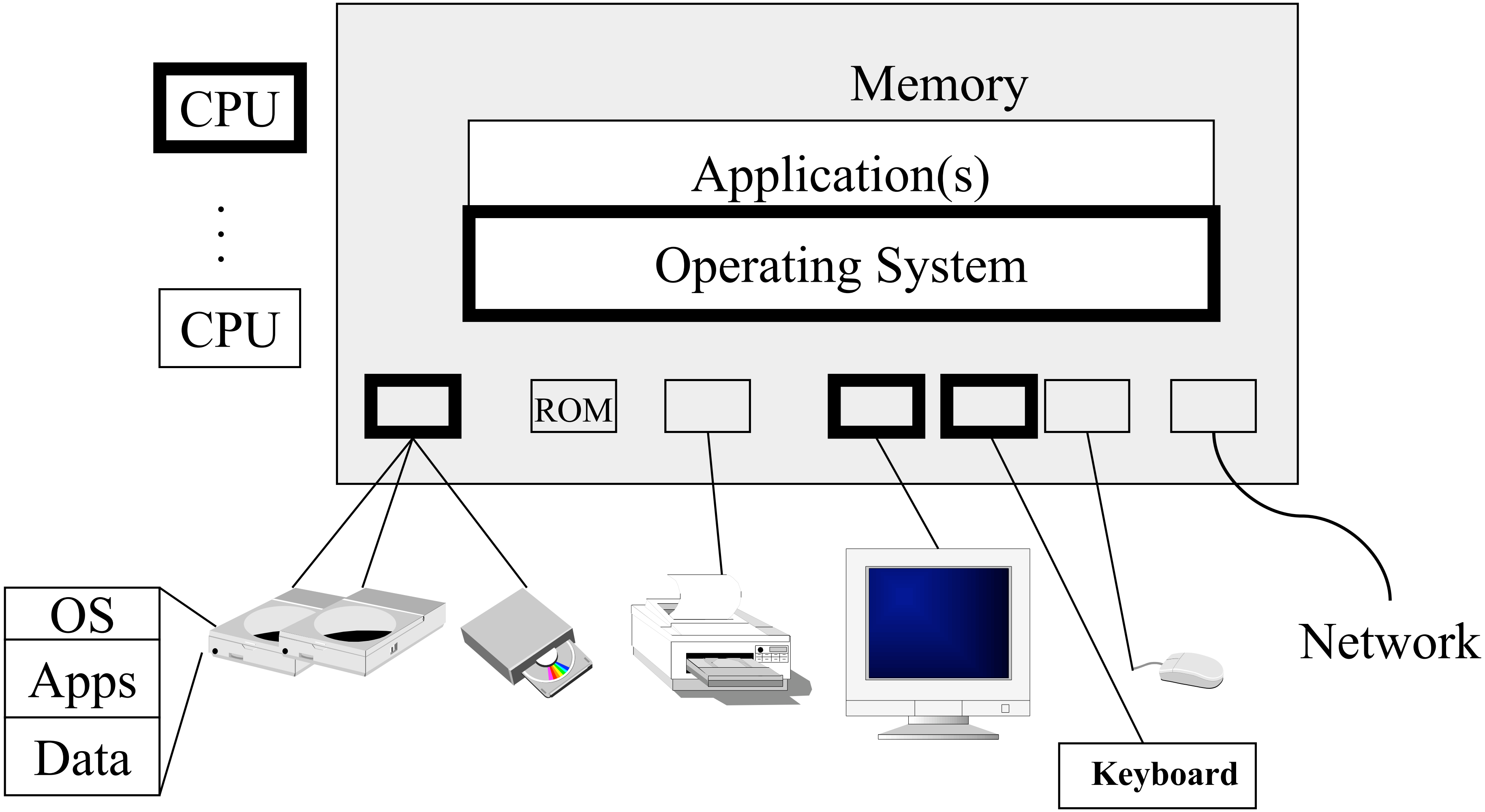
Operating System Overview

Otto J. Anshus

A Typical Computer



A Typical Computer System



Moving data around in the machine

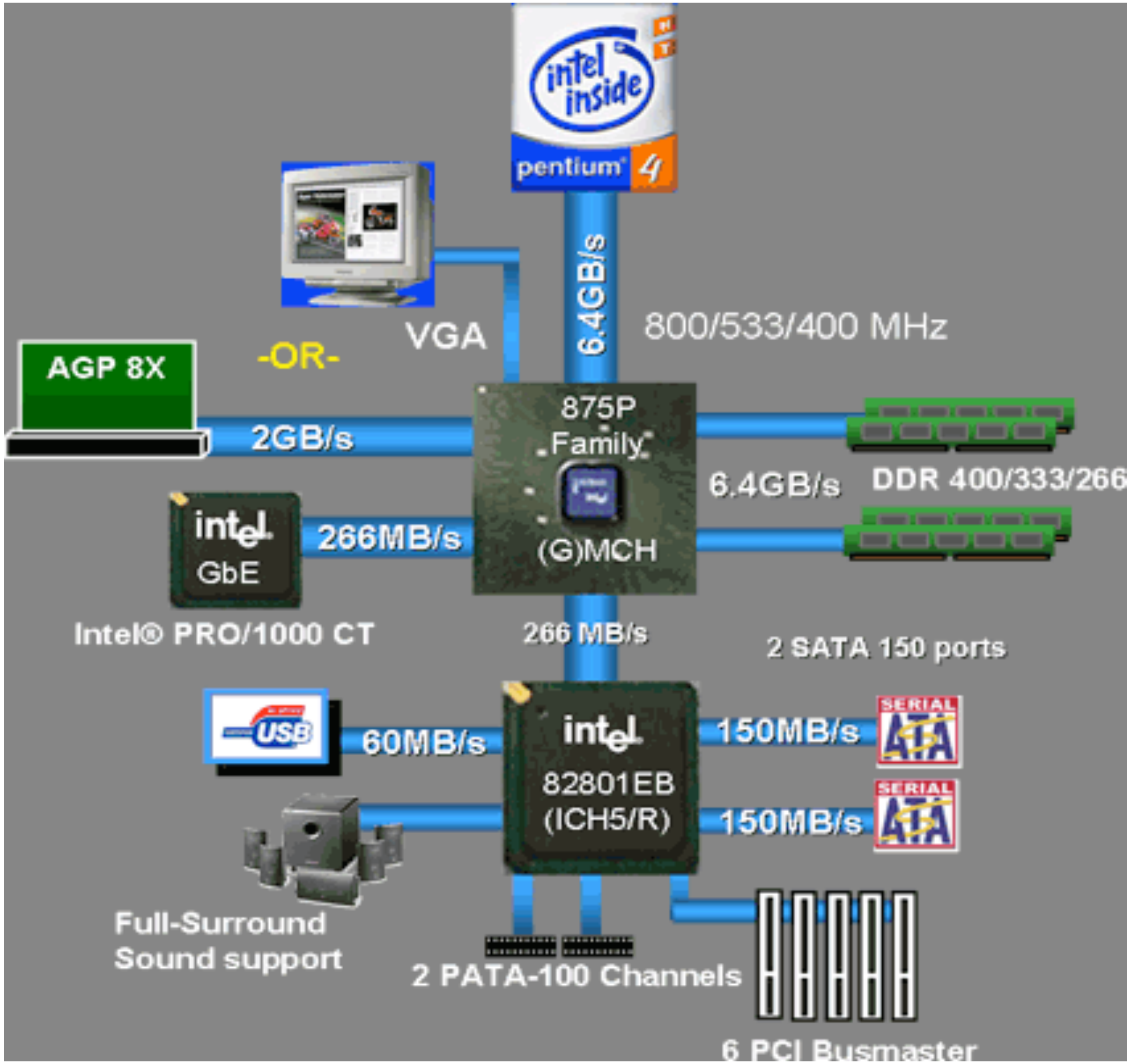
- The processor has entry/exit points for moving data in and out of the processor.
- A ”bus” is a set of wires upon which devices can connect and communicate over.
- The conceptually simplest way of connecting everything together would be to extend the wires in/out of the processor, and have everything else hook onto that
- For our systems, that would be impractical and expensive

Buses

- In/out of processor should be very fast
  - You can make it wide (more parallell wires)
  - You can increase the rate on each wire while still making sure that corresponding bits on each line arrive at same time
  - You can do other tricks
  - For achieving this you will trade a combination of cost, distance, robustness, power, etc.
  - Not needed nor practical for your diskette, keyboard, mouse and many other devices
- We need a range of bus'es (or highways) from the super-wide, super-fast bus between CPU, Memory and cache, to narrow but robust footpaths to "legacy" devices

Chip-set

- Commercially standardized circuitry that ”surround” the processor and provides a set of buses and some other functionality
- Also has a programmable timer that you will set to interrupt the processor regularly
- PC chip-sets traditionally have two exit/entry areas
  - North-bridge (fast and furios)
    - CPU, Memory, AGP-port (?), now also Gb Ethernet
  - South-bridge
    - Everything else, including ”legacy buses”
    - Used to be limited by PCI-bus speed, now is much faster



intel inside
pentium 4
6.4GB/s
800/533/400 MHz
VGA
-OR-
AGP 8X
2GB/s
875P
Family
(G)MCH
6.4GB/s
DDR 400/333/266
intel
GbE
266MB/s
Intel® PRO/1000 CT
266 MB/s
2 SATA 150 ports
USB
60MB/s
intel
82801EB
(ICH5/R)
150MB/s
150MB/s
SERIAL ATA
SERIAL ATA
Full-Surround
Sound support
2 PATA-100 Channels
6 PCI Busmaster

Wrap-up: The Processor

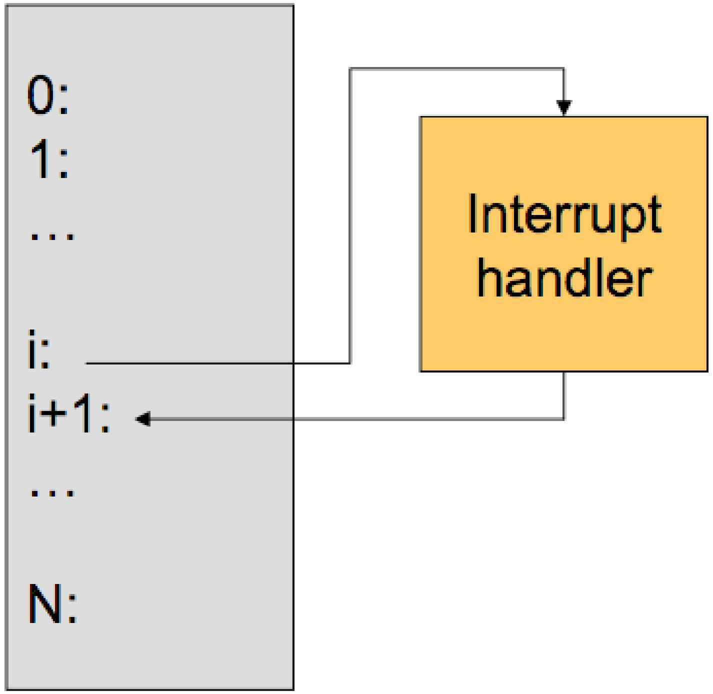
- Von Neumann architecture, stored program, instruction pointer, sequential execution ”one-at-a-time”
- Control section
  - Decodes intructions and controls the datapath
- Datapath including ALU
  - Register file, paths for moving data around internally and out/in of processor, operational units (ALU)

Wrap-up: OS-HW agreement

- We agreed with processor architect that whenever processor couldn't proceed meaningfully, it should note the exception and proceed fetching instructions from a predermined location in memory. We, the OS-writers, will make sure appropriate code resides at that location in memory
- To allow other HW to request the attention of the OS, the processor architect provides the processor with an "interupt line." The processor checks the line once every instruction-cycle. Whenever the line is set, the processor faults and gets it's next instruction at a predetermined location, where we, the OS-writers, will make sure …

Interrupts and Traps

- Interrupts
  - Raised by external events
  - CPU can resume from the interrupt handler
  - **iret** instruction: returns by popping return address from stack, and enable interrupts (IA32 instruction set)
- Traps
  - Internal events
  - System calls (syscalls)
  - Also return by **iret**

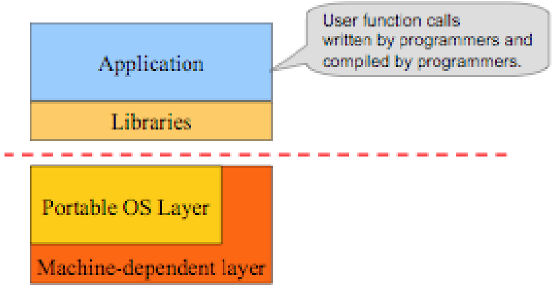


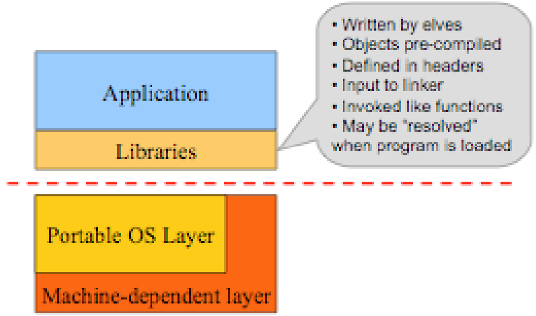
User level vs. Kernel level

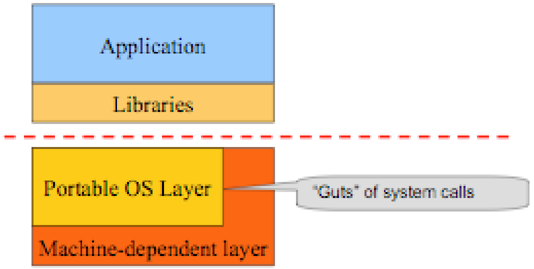
- Kernel (a.k.a. supervisory or privileged) level
    - All instructions are available
    - Total control possible so OS must say “Mine, all mine” (Daffy Duck)
- User level
    - Some instructions are not available any more
    - Programs can be modified and substituted by user

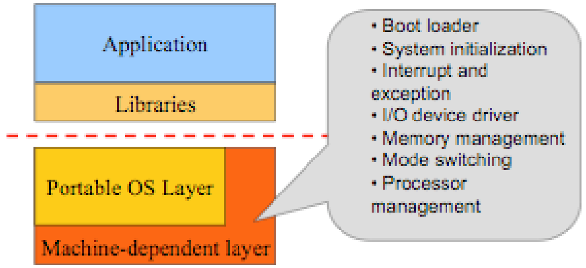
In theory, but not always in practice

Typical Unix OS Structure

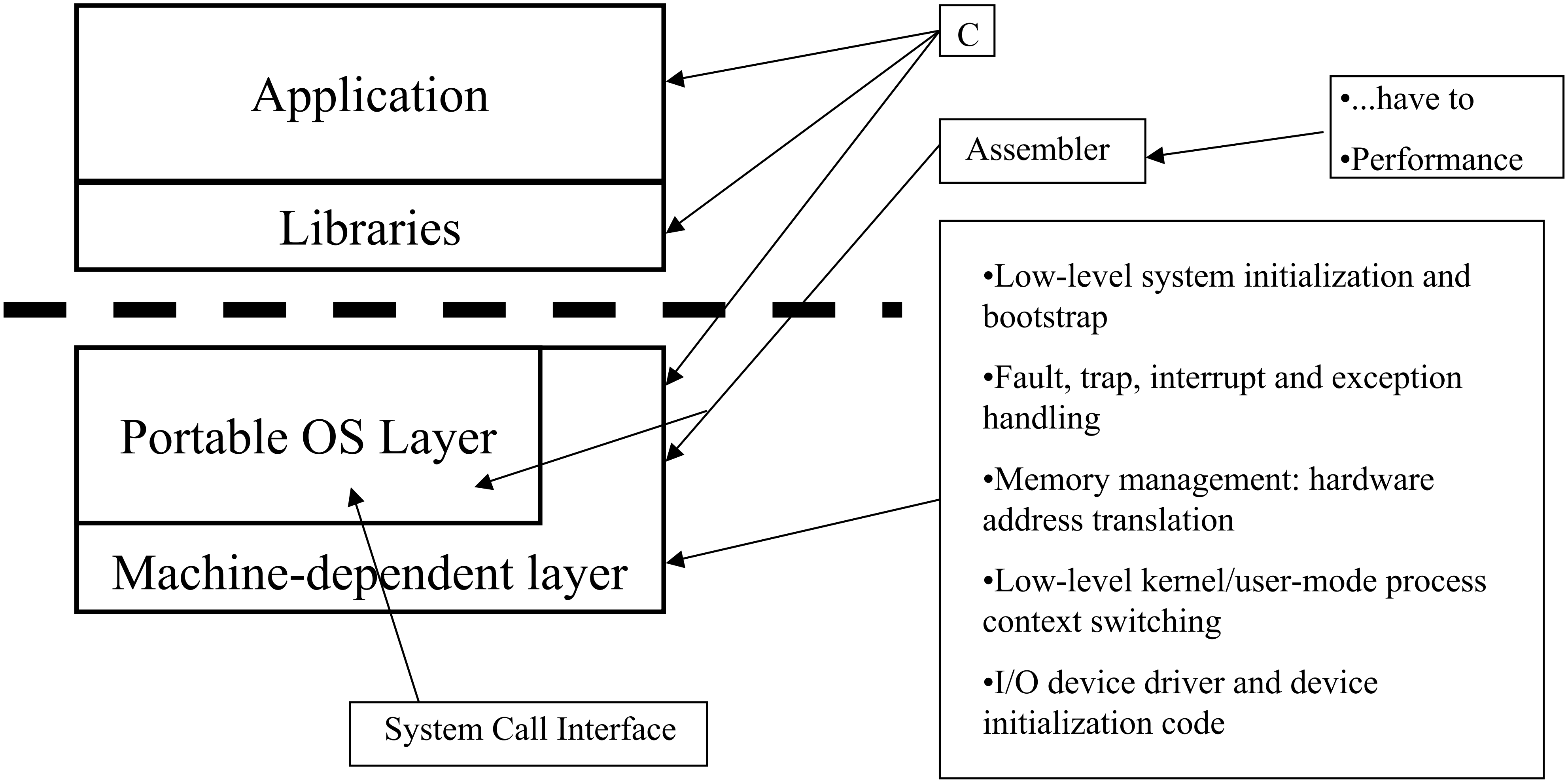




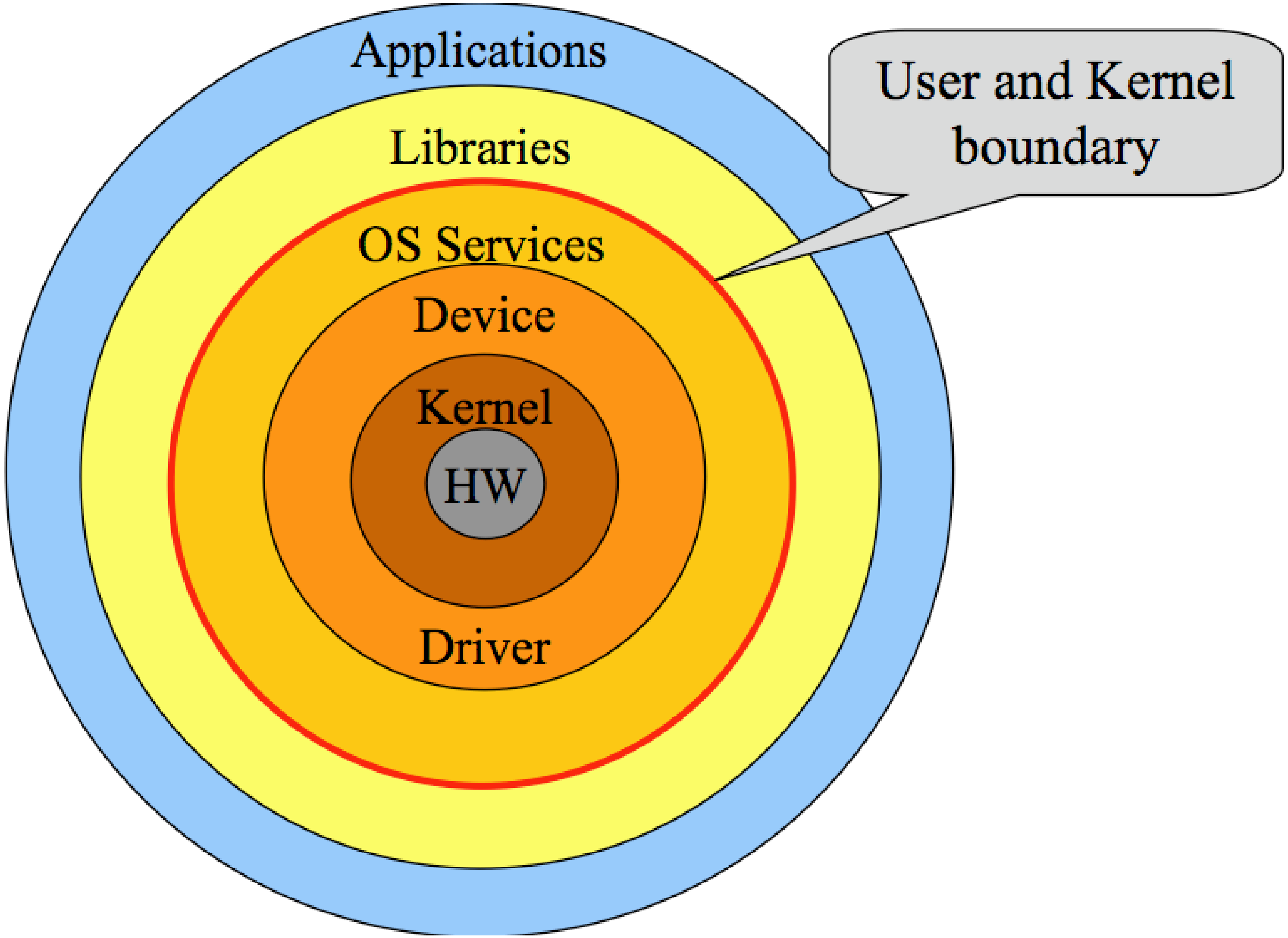




Typical Unix OS Structure



Software "Onion"

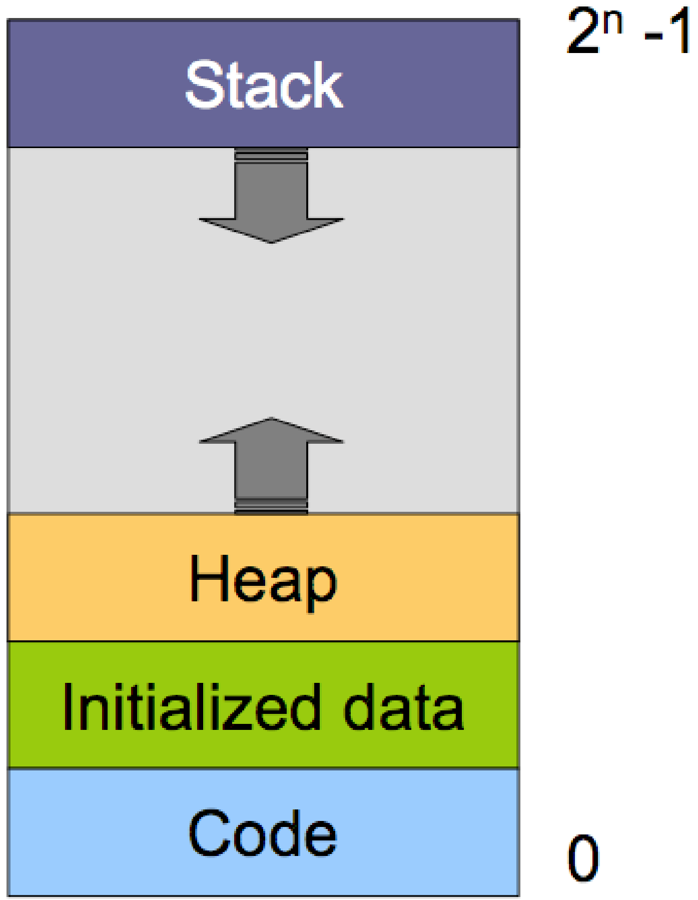


Linux Kernel version 2.0

- 500,000 lines of C code and 8000 lines of assembler
  - “Micro kernel” (process & memory management): 5%
  - Device drivers: 90%
  - Network, file systems, initialization, etc.: 5%

The Application: A **process**

- Four segments
  - Code/text: instructions
  - Data: variables
  - Stack
  - Heap
- Why?
  - Separate code and data
  - Stack and heap grow toward each other



The Application

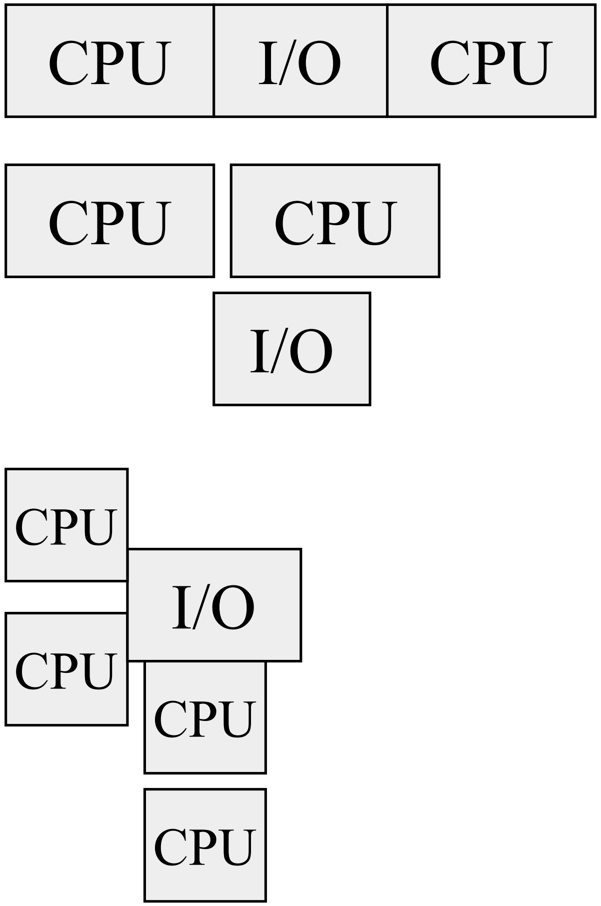
- Stack
  - Layout by compiler
  - Allocate at process creation (fork)
  - Deallocate at process termination
- Heap
  - Linker and loader specify the starting address
  - Allocate/deallocate by library calls such as **malloc()** and **free()** called by application
- Data
  - Compiler allocate statically
  - Compiler specify names and symbolic references
  - **Linker** translate refs and relocate addresses
  - **Loader** finally lay them out in memory

OS Service Examples

- Examples of services **not** provided at user level
  - System calls
    - File open, close, read and write
  - Control the CPU so that users can't take over by doing
    - `while ( 1 ) ;`
  - Protection:
    - Keep user programs from crashing OS
    - Keep user programs from crashing each other
- Examples of services running at user level
  - Read time of the day
  - Protected user level stuff

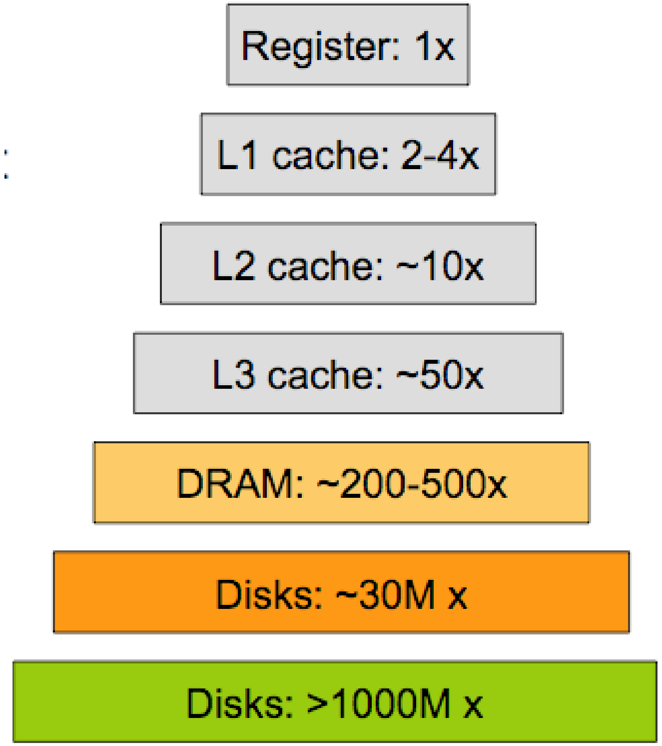
Processor Management

- Goals
  - Overlap between I/O and computation
  - Time sharing
  - Multiple CPU allocations
- Issues
  - Do not waste CPU resources
  - Synchronization and mutual exclusion
  - Fairness and deadlock free

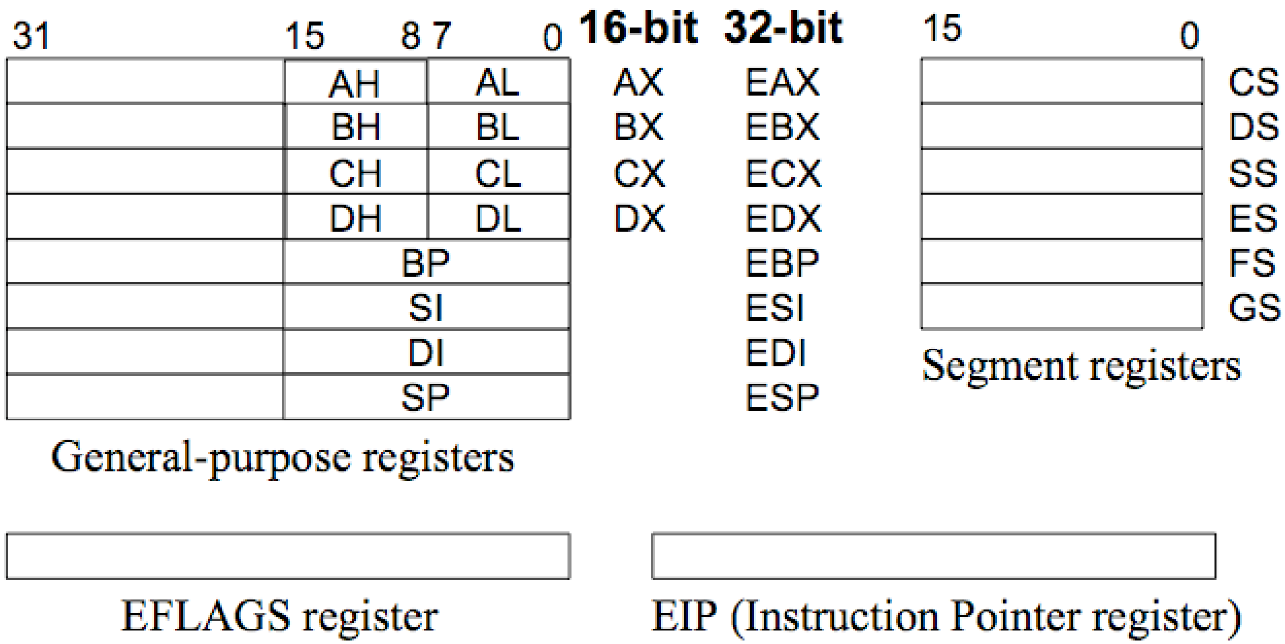


Memory Management

- Goals
  - Support programs to run
  - Allocation and management
  - Transfers from and to secondary storage
- Issues
  - Efficiency & convenience
  - Fairness
  - Protection



IA32 Architecture Registers

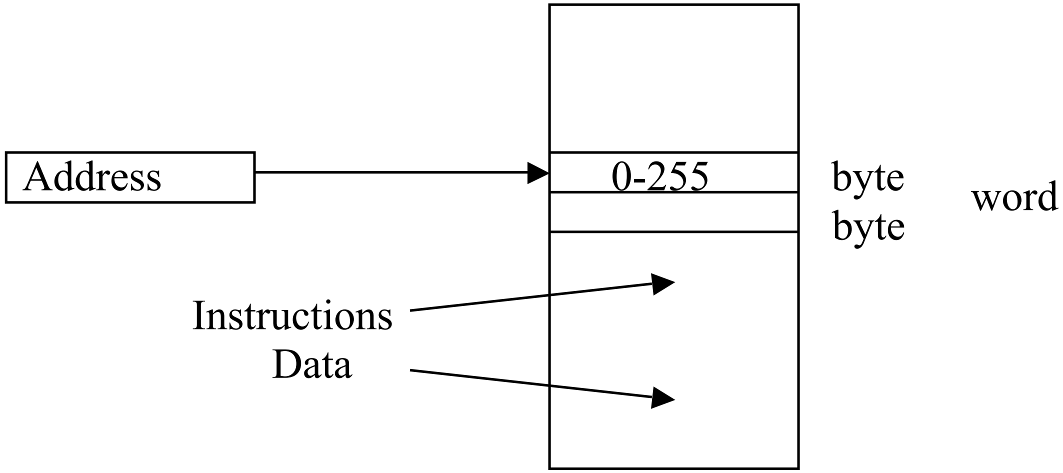


Memory

Intel architecture is "little endian": little end in first

Power PC (and Sun SPARC) is "***bi***endian", but Apple is using it as a "big endian"

Java: big endian (most significant byte)



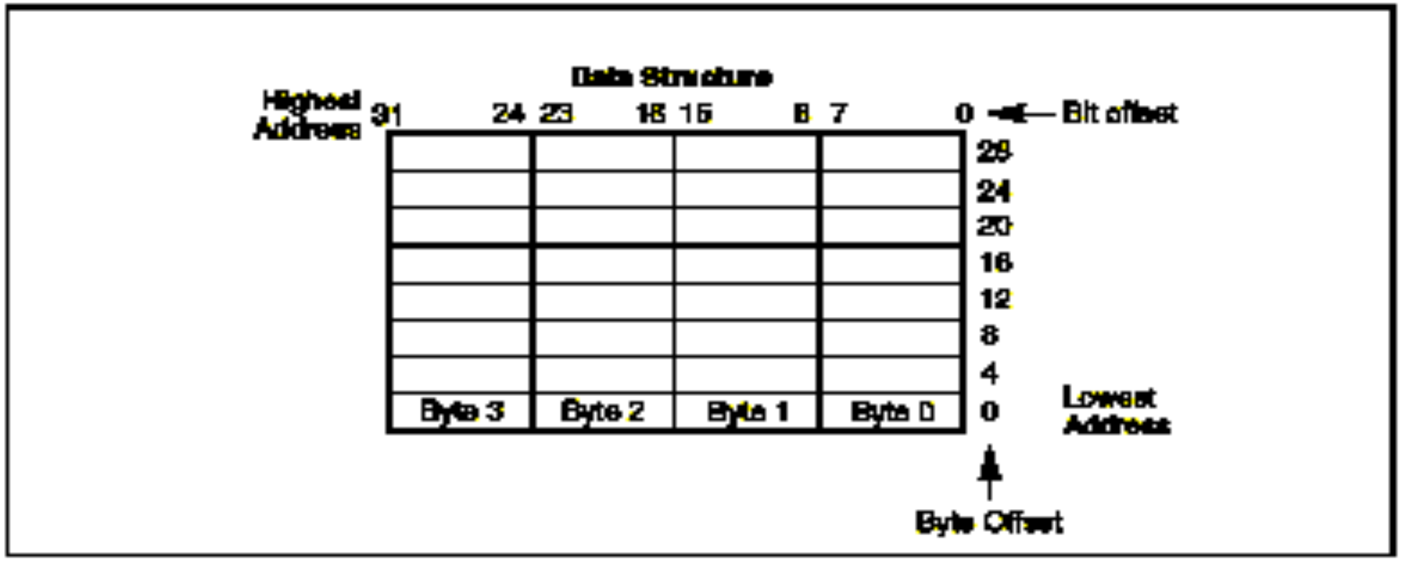
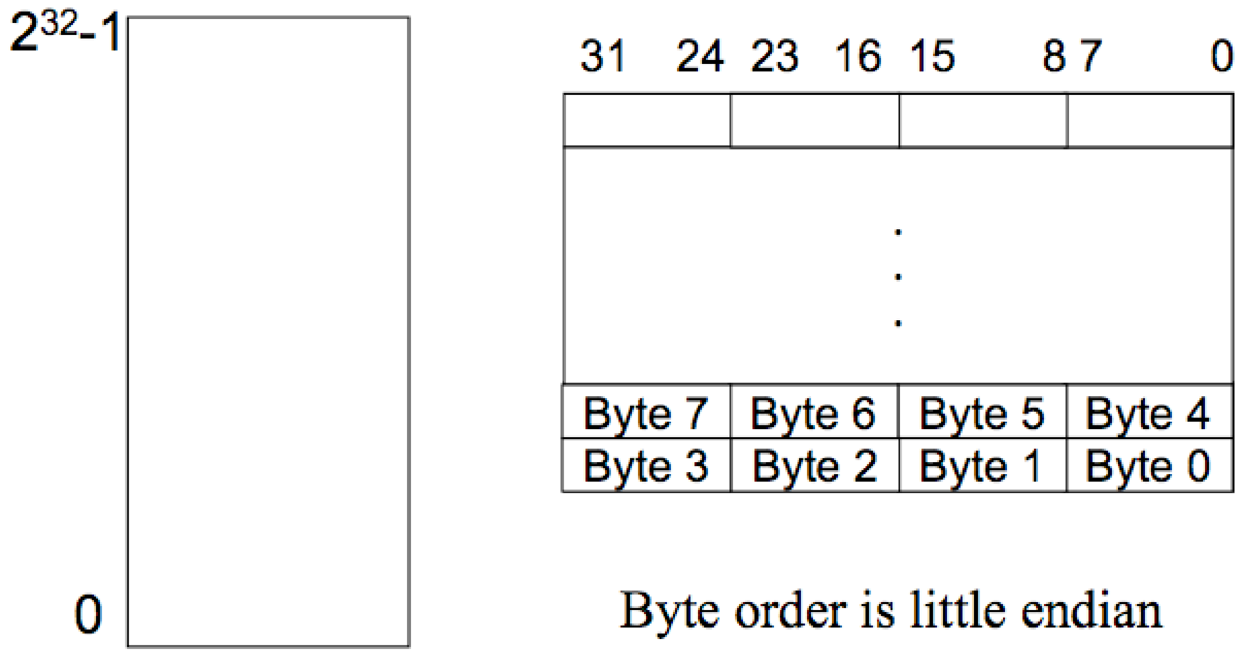


Figure 1-1. Bit and Byte Order

IA32 Memory



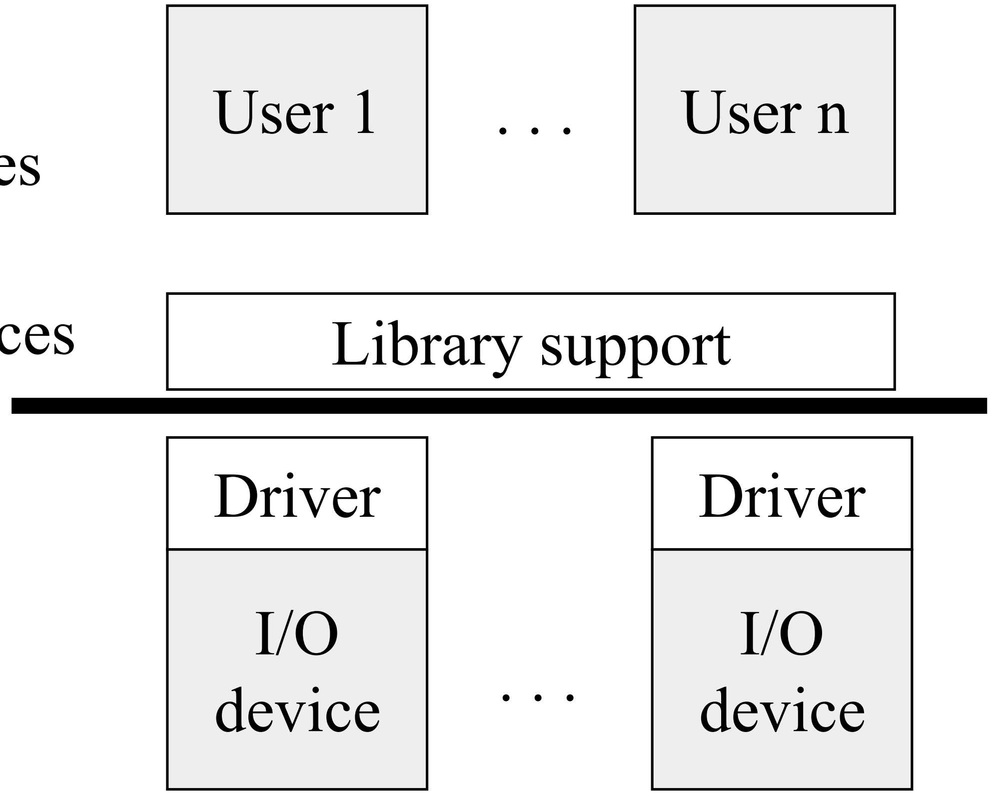
Byte order is little endian

Hexadecimal

- **16 decimal is base**
    - 0, 1, 2,…,9, A, B, C, D, E, F
- **C4AFh=50351d**
    - $C*16^3+ 4*16^2+A*16^{1+} F*16^0$
    - $12*16^3+ 4*16^2+10*16^{1+} 15*16^0 = 50351d$
- **2^8-1=11111111b =255d =FFh**
- **2^{16}-1=1111111111111111b =65535d =FFFFh**
- **2^{32}-1=1111111111111111…1b =4294967295d =FFFFFFFFh**

I/O Device Management

- Goals
  - Interactions between devices and applications
  - Ability to plug in new devices
- Issues
  - Efficiency
  - Fairness
  - Protection and sharing

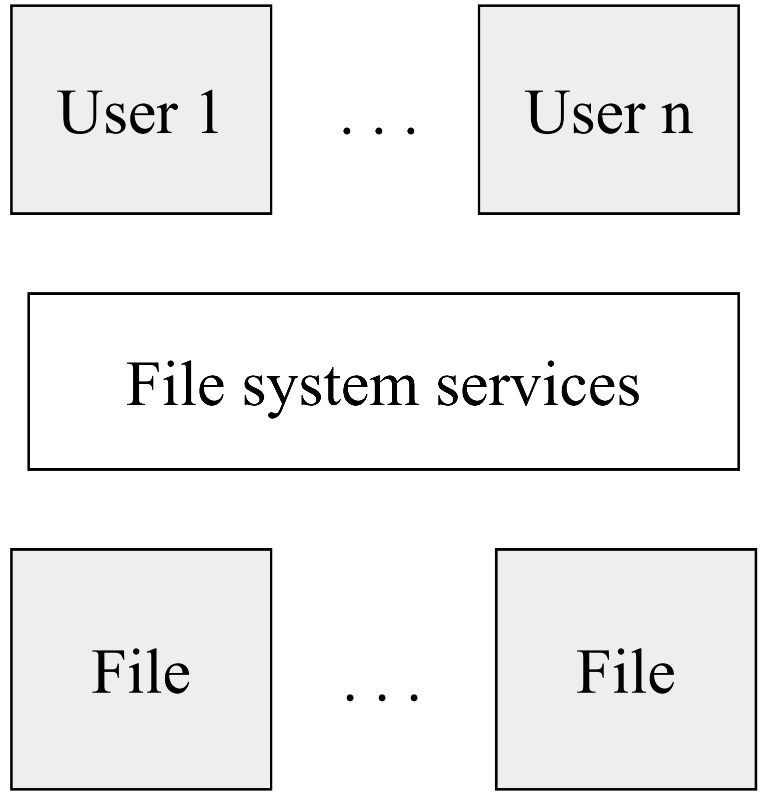


Window Systems

- All in the kernel (Windows)
  - Pro: efficient
  - Con: difficult to develop new services
- All at user level
  - Pro: easy to develop new services
  - Con: protection
- Split between user and kernel (Unix)
  - Kernel: display driver and mouse driver
  - User: the rest

File System

- A typical file system
  - Open a file with authentication
  - Read/write data in files
  - Close a file
- Can the services be moved to user level?



User level FS?

- Yes: Minix
  - FS as a “server” at user level
    - almost a user process...
    - ...but booted together with OS
    - …and never terminates
    - …and gets higher CPU priority
    - …and a new server means recompiling the kernel
  - disk drivers at Kernel level
- NO: Unix and Windows NT
  - File system at Kernel level