2. Computer-System Structures

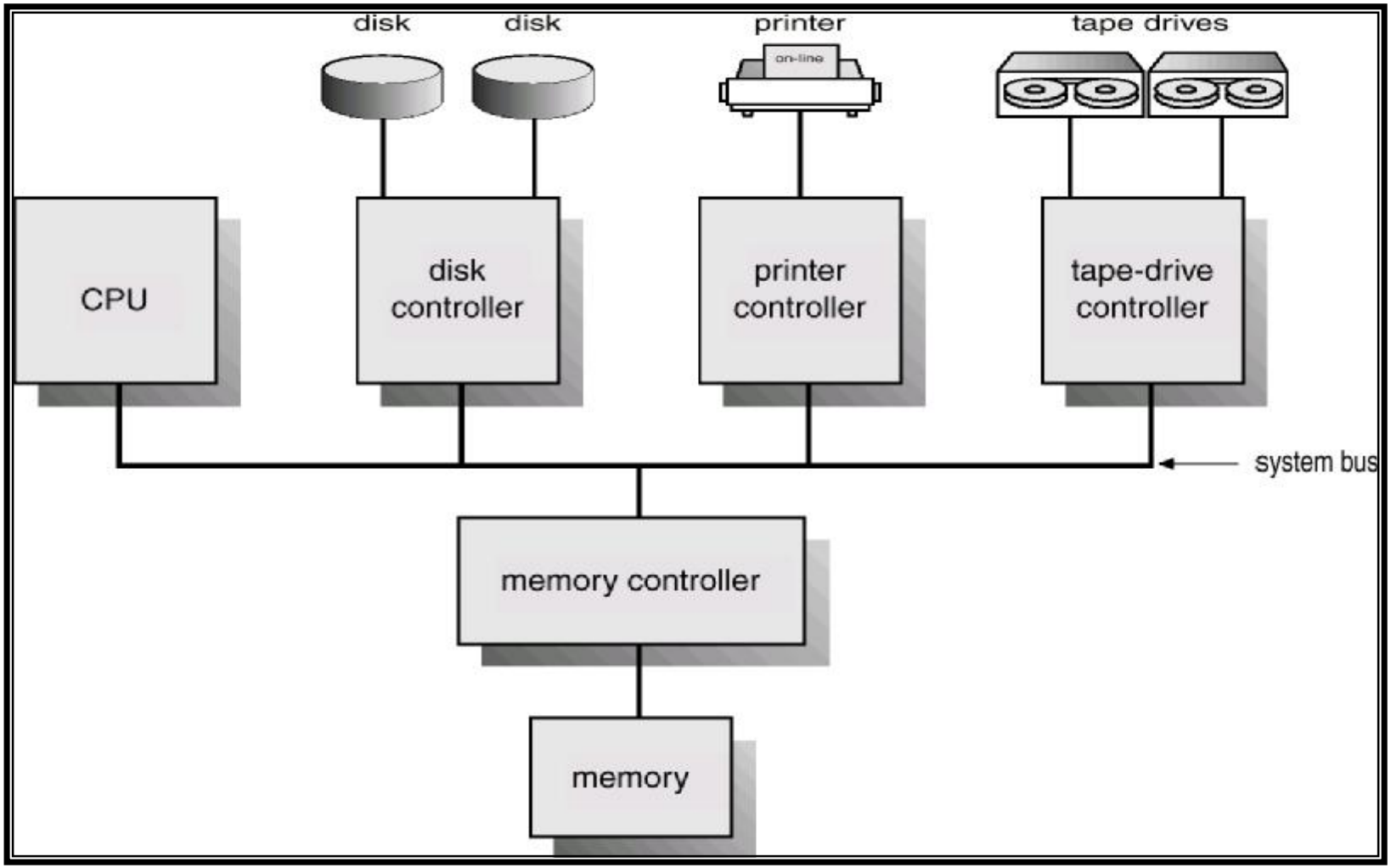
Sungyoung Lee

College of Engineering

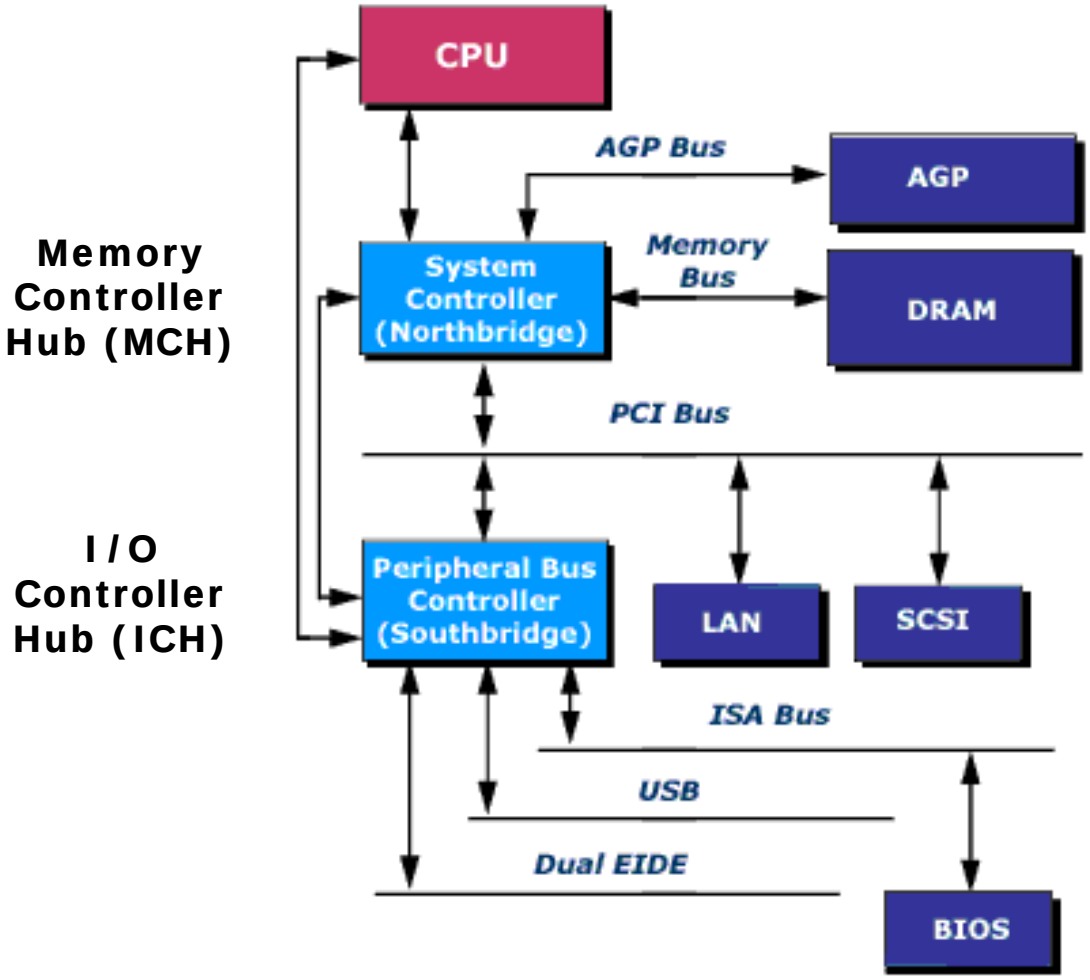
KyungHee University

Contents

Computer-System Architecture



Modern PC Architecture



CPU

- Instruction Set Architecture
  - RISC vs. CISC
  - Intel, SPARC, MIPS, PowerPC, ARM, Alpha, …
- Pipelining
  - Fetch, Decode, Execute, Write Back, etc.
- Registers
  - General-purpose registers
  - Program Counter
  - PSW (Program Status Word)
- Instruction-Level Parallelism(ILP)
  - Superscalar vs. VLIW
  - Simultaneous Multithreading

OS and Architecture

- Mutual interaction
  - The functionality of an OS is limited by architectural features
    - Multiprocessing on DOS/8086?
  - The structure of an OS can be simplified by architectural support
    - Interrupt, DMA, etc.
  - Most proprietary OS's were developed with the certain architecture in mind

Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an *interrupt*

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*
- A *trap (or exception)* is a software-generated interrupt caused either by an error or a user request
- An operating system is *interrupt* driven

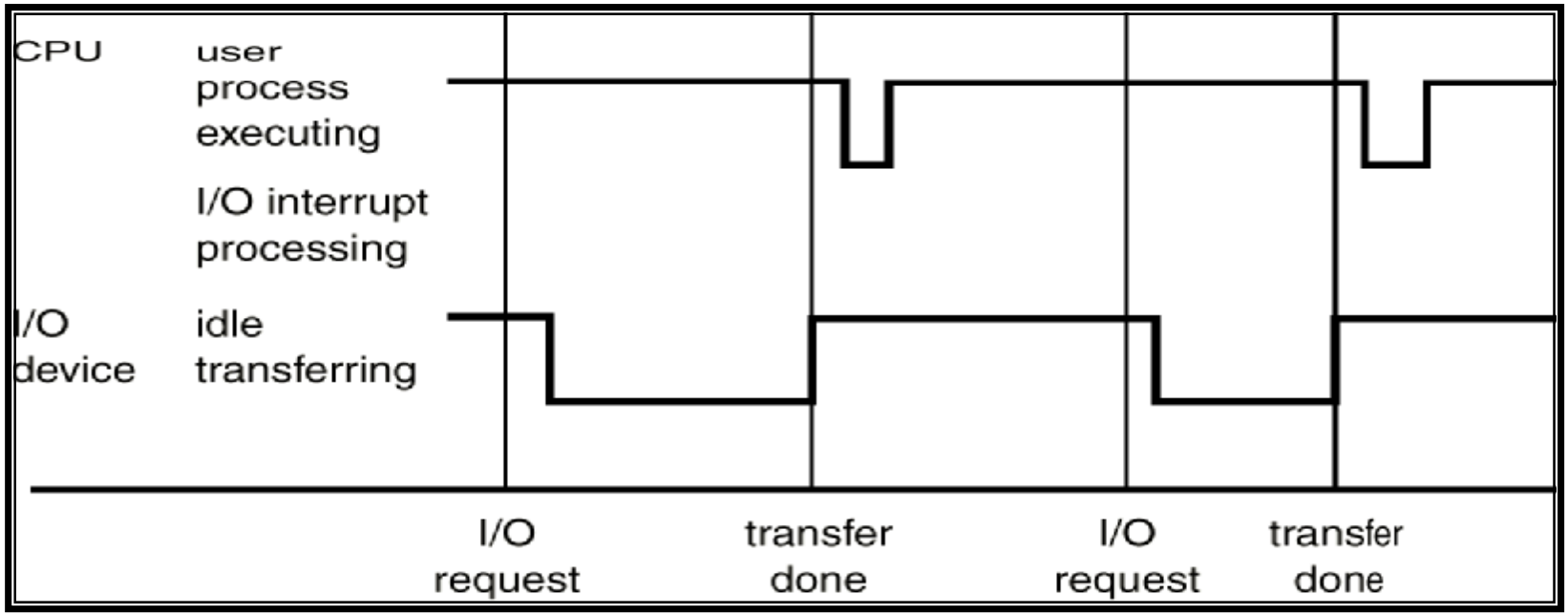
Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter

- Determines which type of interrupt has occurred:
  - *polling*
  - *vectored* interrupt system

- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Time Line For a Single Process Doing Output



Interrupts and Exceptions

- Interrupts
  - Generated by hardware devices
    - Triggered by a signal in INTR or NMI pins (Pentium)
  - Asynchronous

- Exceptions
  - Generated by software executing instructions
    - INT instruction in IA32
    - Page fault, protection fault
  - Synchronous
  - Trap (expected) or fault (unexpected)

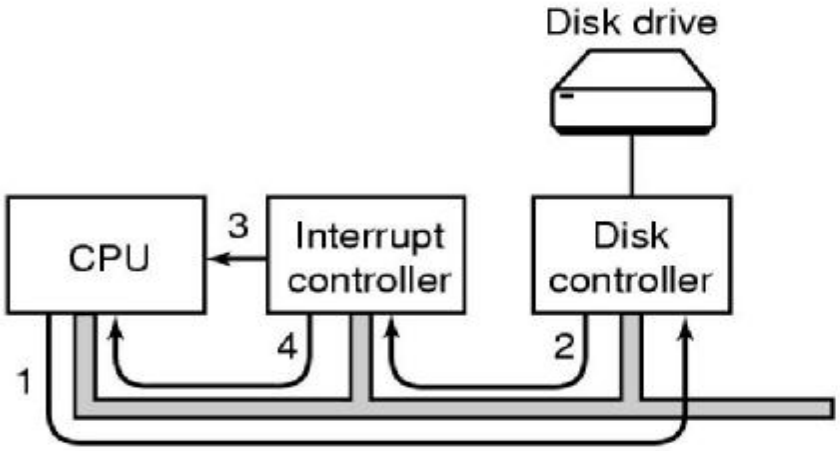
I/O Structure

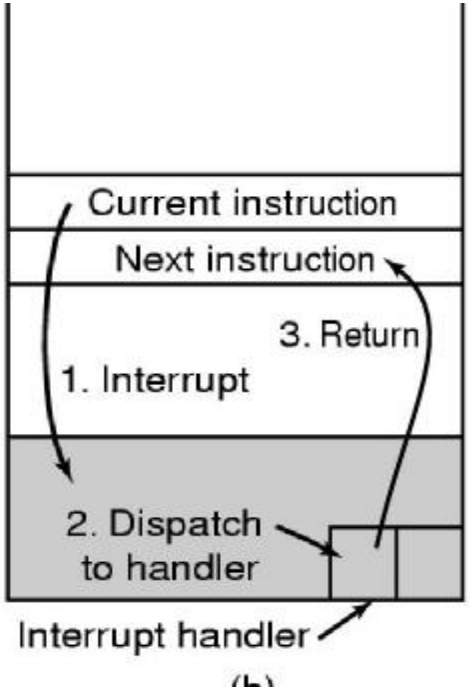
- After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing

- After I/O starts, control returns to user program without waiting for I/O completion
  - *System call* – request to the operating system to allow user to wait for I/O completion
  - *Device-status table* contains entry for each I/O device indicating its type, address, and state
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

I/O Control

- How does the kernel notice an I/O has finished?
  - Polling
  - Hardware interrupt





I/O Control (Cont'd)

Device interrupts

↓

CPU stops current operation, switches to kernel mode, and saves current PC and other state on kernel stack

↓

CPU fetches proper vector from vector table and branches to that address (to routine to handle interrupt)

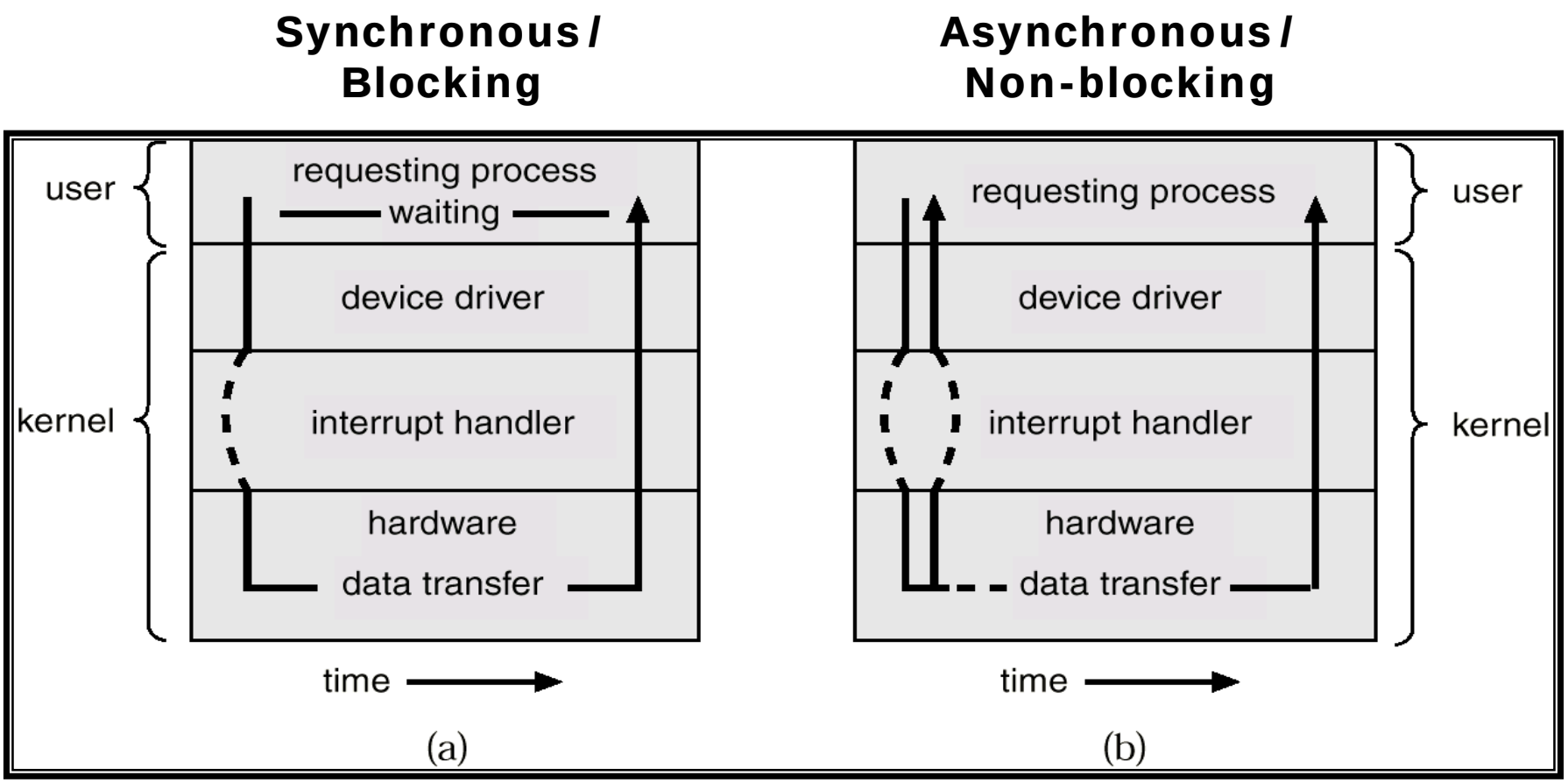
↓

Interrupt routine examines device database and performs action required by interrupt

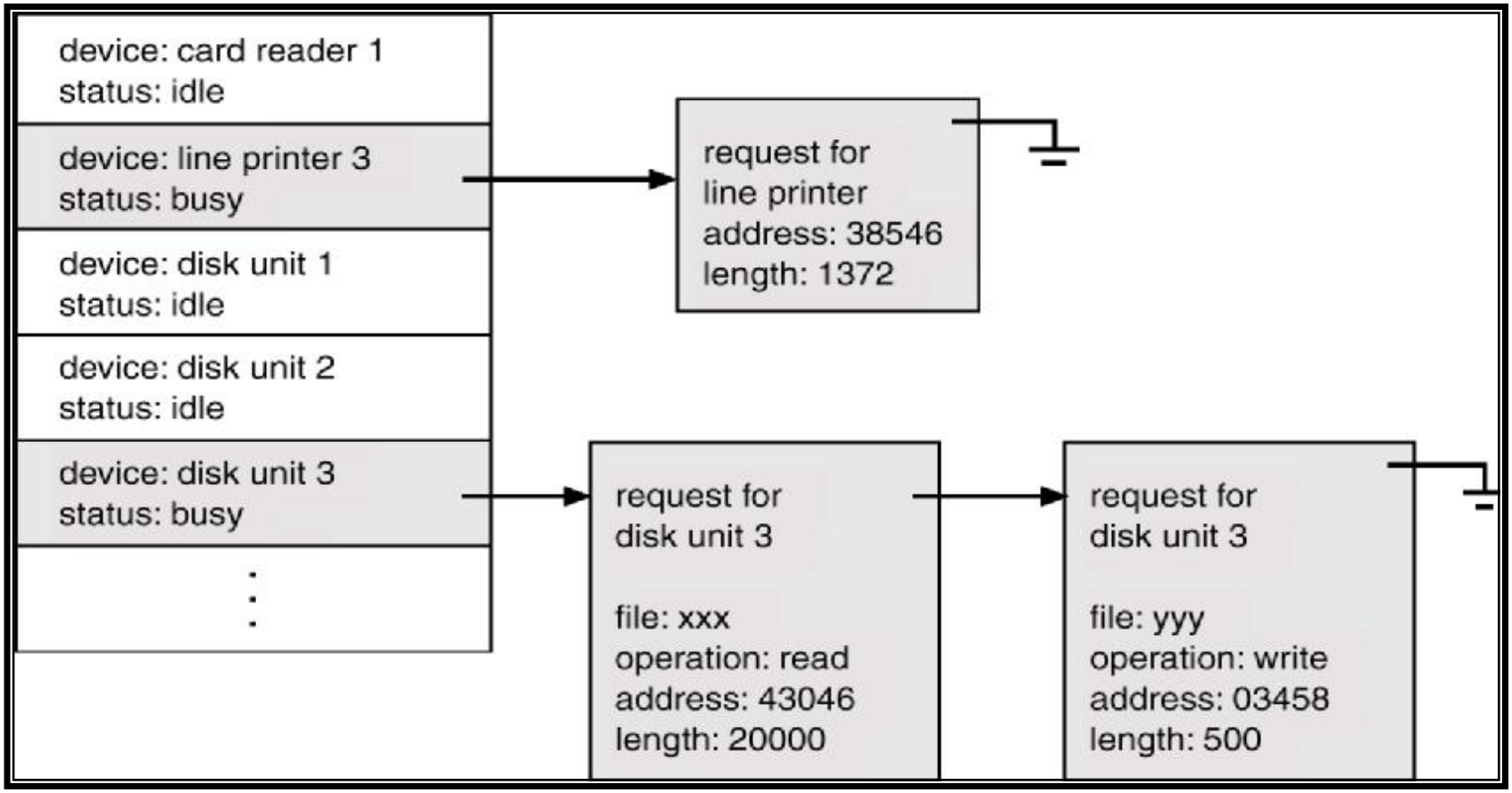
↓

Handler completes operation, restores saved (interrupt state) and returns to user mode (or calls scheduler to switch to another program)

Two I/O Methods



Device-Status Table



Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

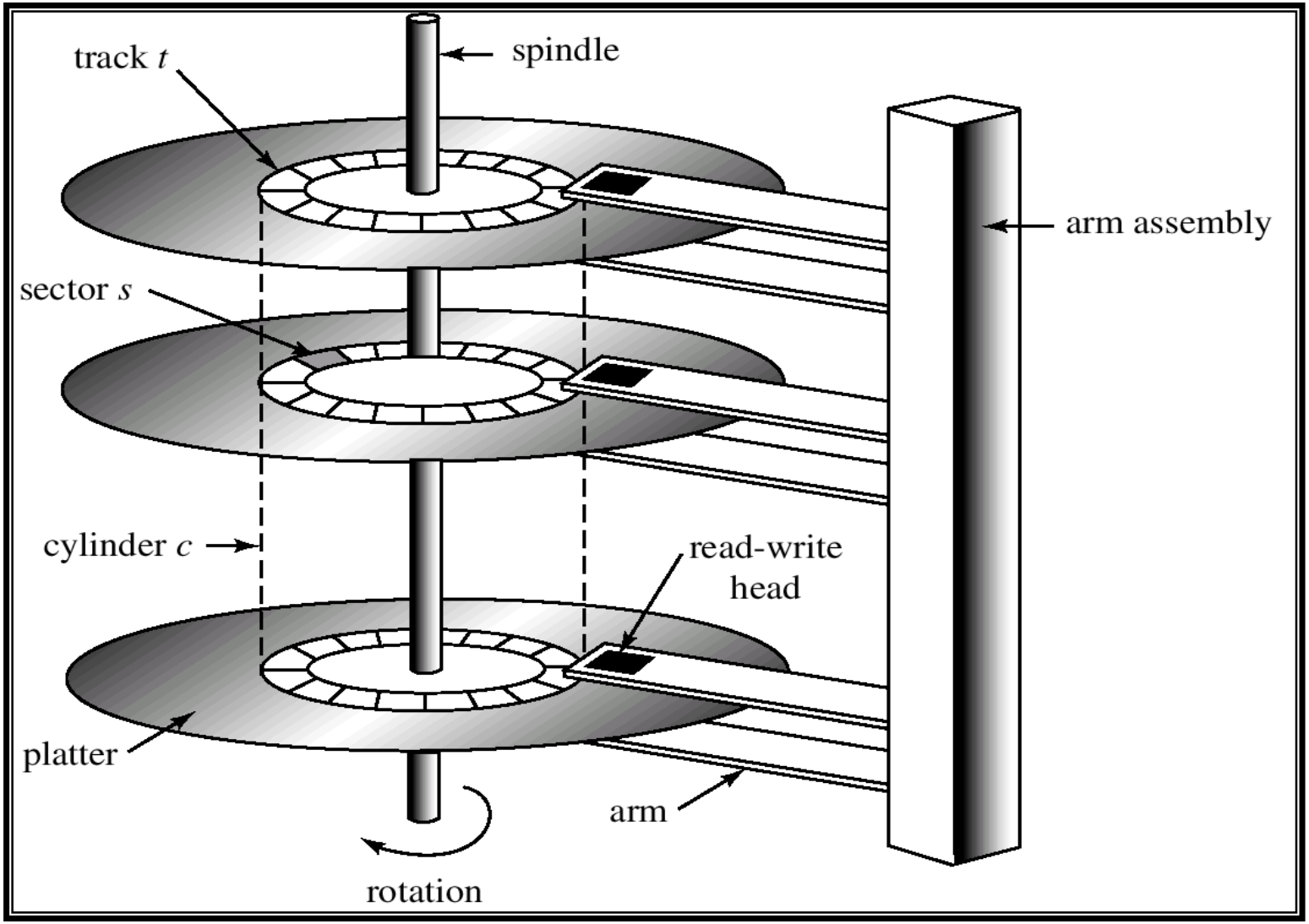
DMA

- Data Transfer Modes in I/O
  - Programmed I/O (PIO)
    - By special I/O instructions
    - Memory-mapped I/O
  - DMA (Direct Memory Access)
    - Used for high-speed I/O devices able to transmit information at close to memory speeds
    - Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
    - Only an interrupt is generated per block

Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - von Neumann architecture vs. Harvard architecture
  - SRAM vs. DRAM
  - DDR, QDR, RDRAM
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*
  - The *disk controller* determines the logical interaction between the device and the computer

Moving-Head Disk Mechanism



A Modern Disk Drive

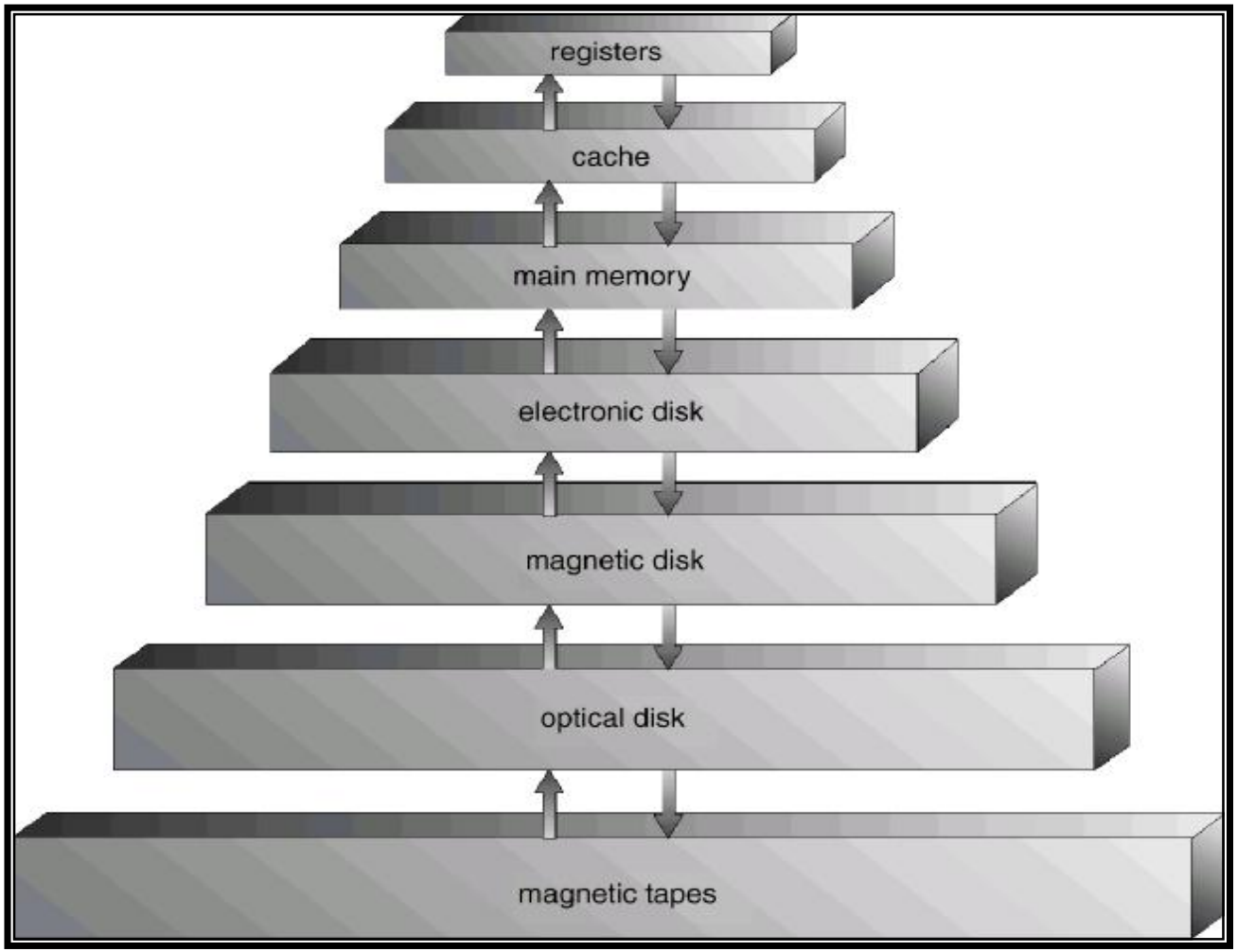
- IBM Deskstar 120GXP (120GB)
  - Disks/Head: 3/6
  - Max. areal density: 29.7 Gbits/sq.inch
  - Max. recording density: 524K BPI (bits/inch)
  - Track density: 56.7K TPI (tracks/inch)
  - Seek time: 8.5ms average
  - Rotational latency: 7200rpm (8.3ms/rotation)
  - Max. media transfer rate: 592 Mbits/sec
  - Max. interface transfer rate: 100MB/sec
  - Sustained data rate: 48 to 23 MB/sec

Storage Hierarchy

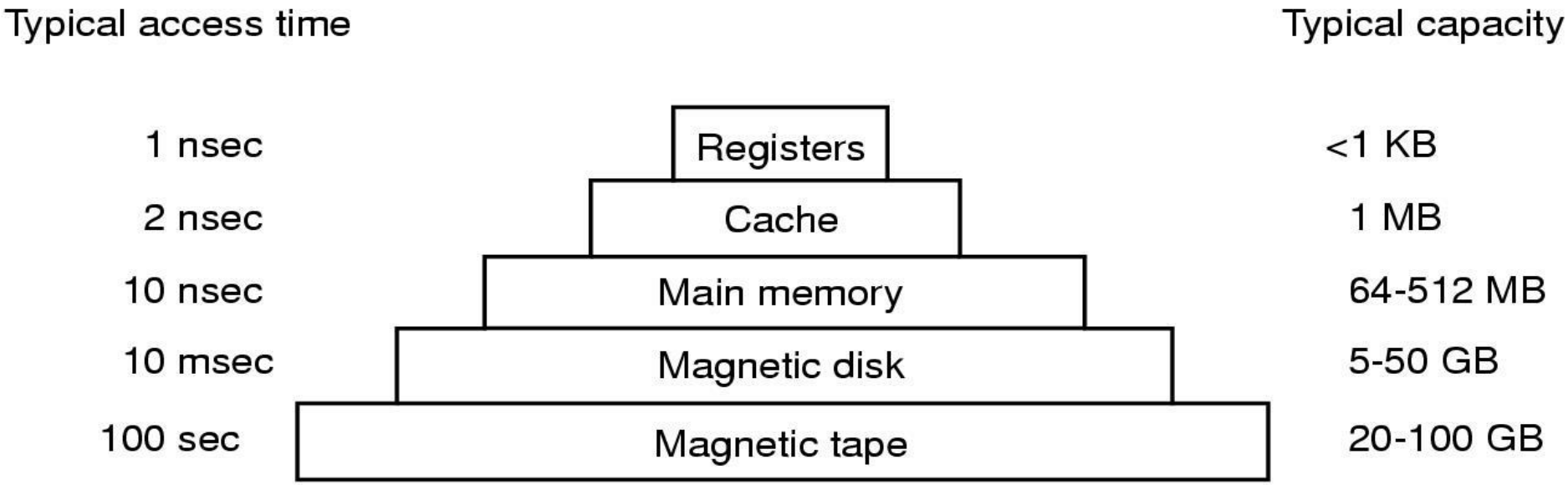
- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

- *Caching*
  - copying information into faster storage system
  - main memory can be viewed as a last *cache* for secondary storage

Storage-Device Hierarchy



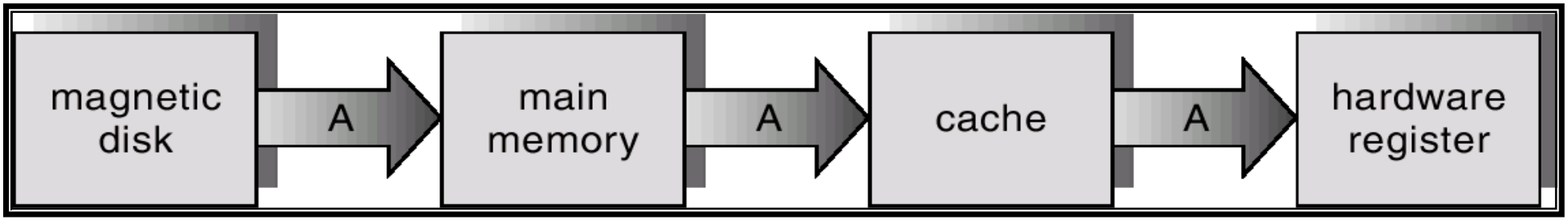
Memory Hierarchy



Caching

- Use of high-speed memory to hold recently-accessed data
- Requires a *cache management* policy
  - Write-through vs. Write-back
- Caching introduces another level in storage hierarchy. This requires data that is simultaneously stored in more than one level to be *consistent*
  - Cache coherency

Migration of A From Disk to Register



Hardware Protection

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection

Dual-Mode Operation

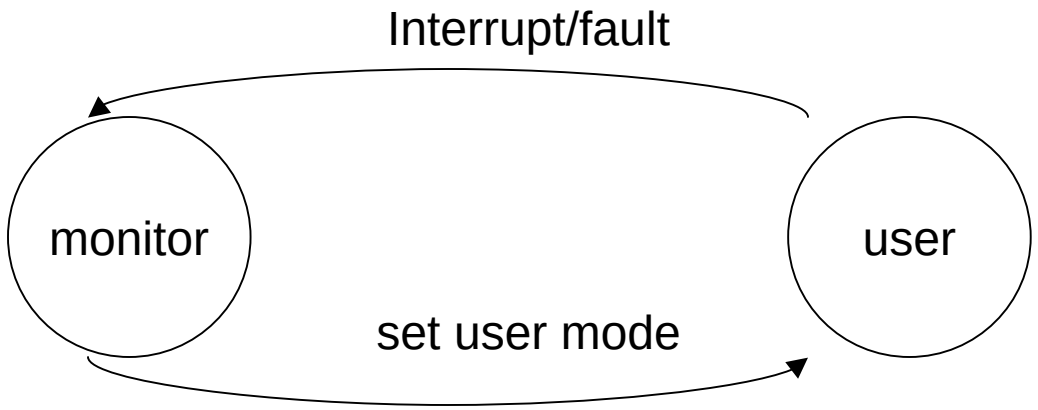
- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly

- Provide hardware support to differentiate between at least two modes of operations
  1. *User mode*
     - execution done on behalf of a user
  2. *Monitor mode* (also *kernel mode* or *system mode*)
     - execution done on behalf of operating system

Dual-Mode Operation (Cont'd)

- *Mode bit* added to computer hardware to indicate the current mode:
  - monitor (0) or user (1)

- When an interrupt or fault occurs hardware switches to monitor mode



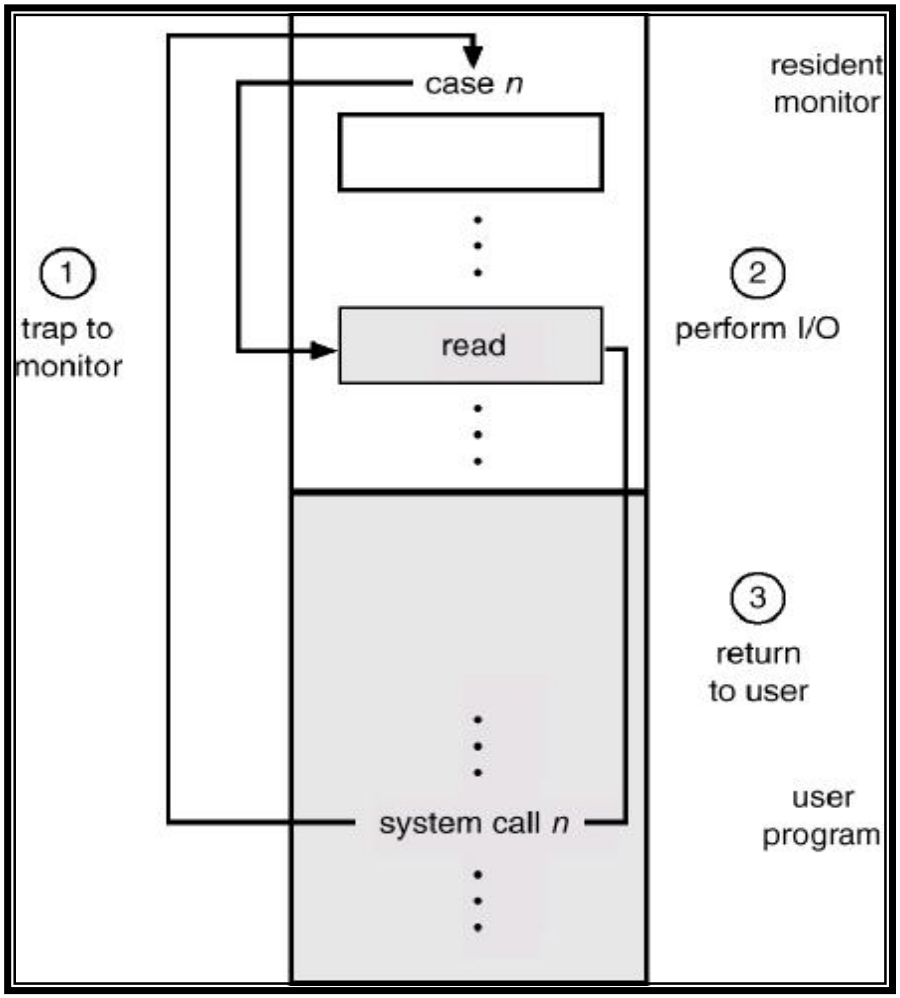
Privileged instructions can be issued only in monitor mode

I/O Protection

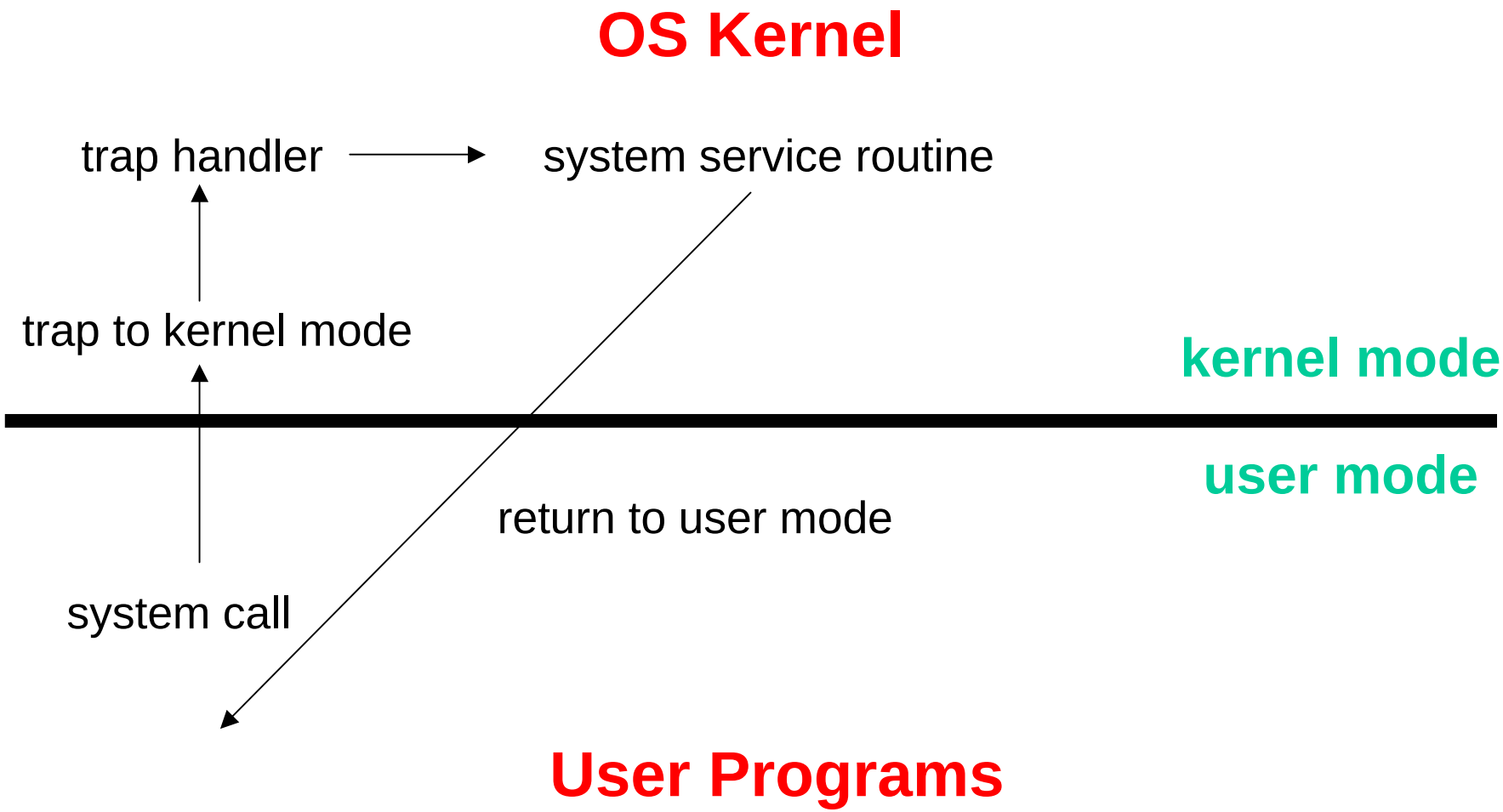
- All I/O instructions are privileged instructions

- Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector)

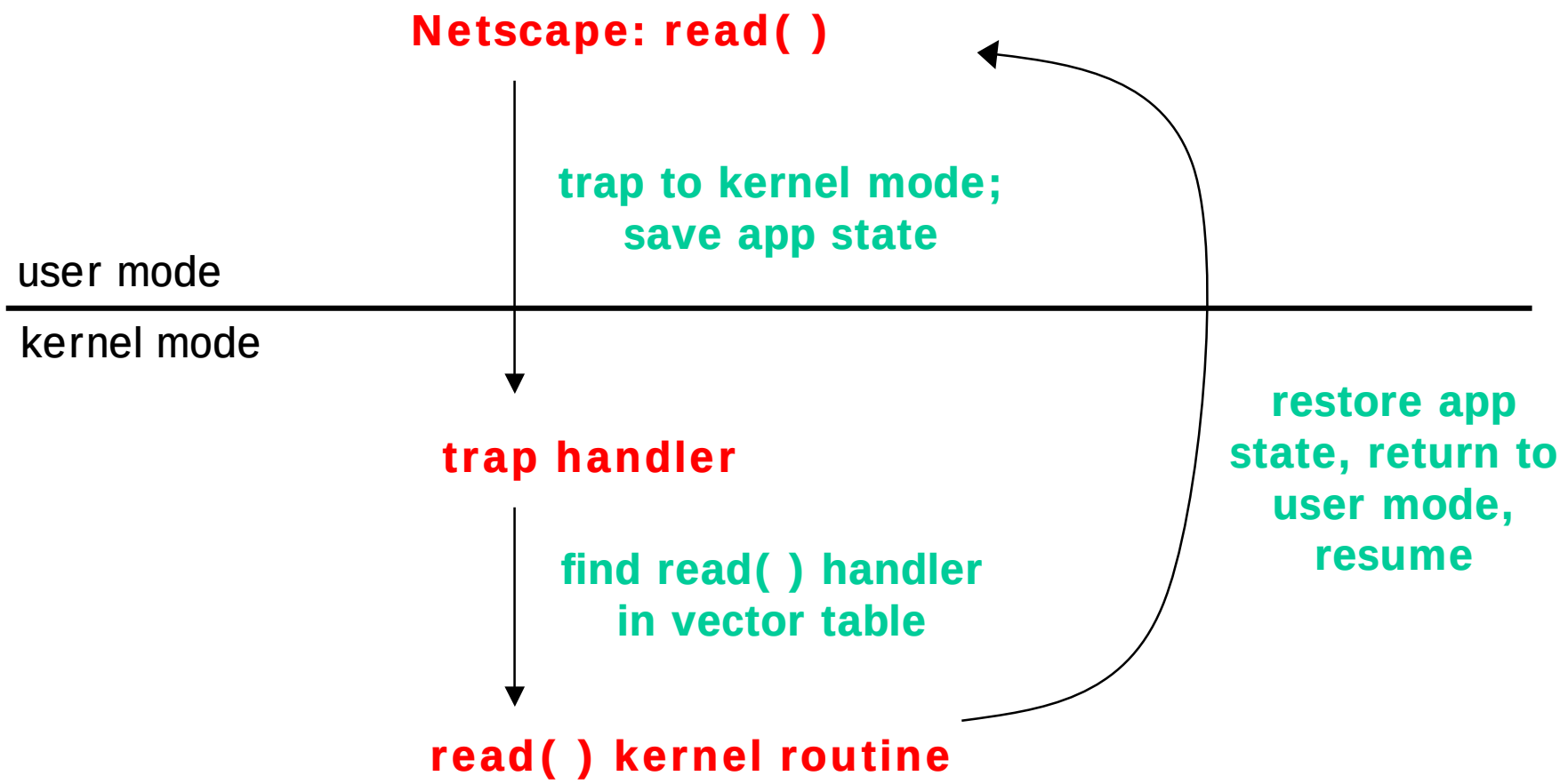
Use of A System Call to Perform I/O



System Call


OS Kernel
trap handler
system service routine
trap to kernel mode
kernel mode
user mode
return to user mode
system call
User Programs

System Call (Cont'd)



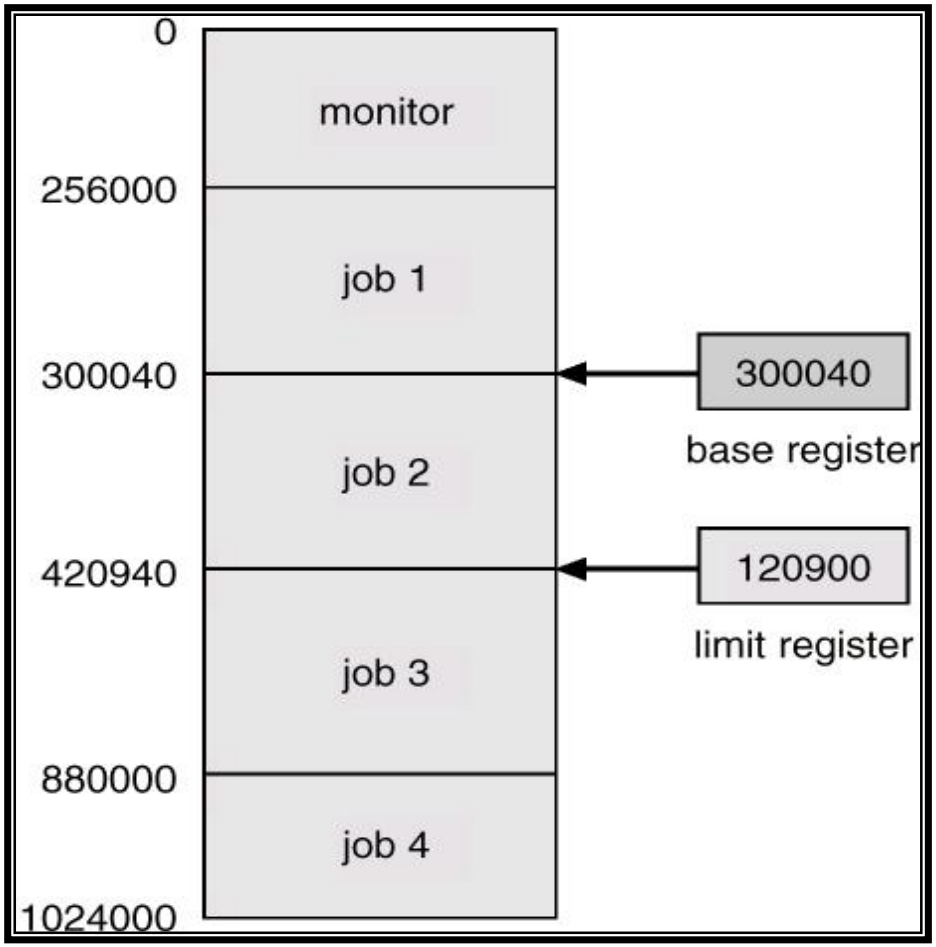
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines

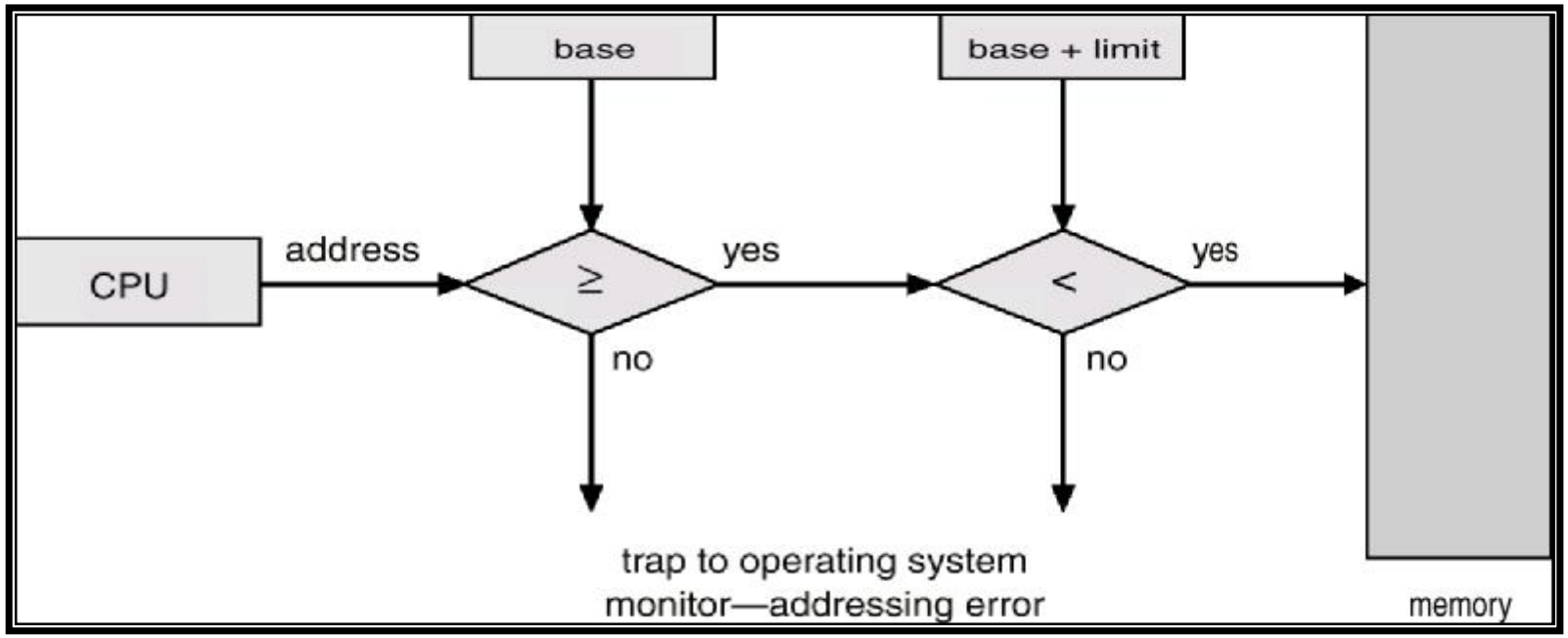
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - **Base register** – holds the smallest legal physical memory address
  - **Limit register** – contains the size of the range

- Memory outside the defined range is protected

Use of A Base and Limit Register



Hardware Address Protection



Hardware Protection

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory

- The load instructions for the *base* and *limit* registers are privileged instructions

Memory Protection

- MMU (Memory Management Unit)
  - Memory management hardware provides more sophisticated memory protection mechanisms
    - base and limit registers
    - page table pointers, page protection, TLBs
    - virtual memory
    - segmentation
  - Manipulation of memory management hardware are protected (privileged) operations

CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control
  - Timer is decremented every clock tick
  - When timer reaches the value 0, an interrupt occurs
- Timer commonly used to implement time sharing
- Time also used to compute the current time
- Load-timer is a privileged instruction

Timers

- How does the OS take control of CPU from the running programs?
  - Use a hardware timer that generates a periodic interrupt
  - The timer interrupt transfers control back to OS
  - The OS preloads the timer with a time to interrupt. (“quantum”)
    - 10ms for Linux
    - Cf) time slice or time quantum
  - The timer is privileged.
    - Only the OS can load it

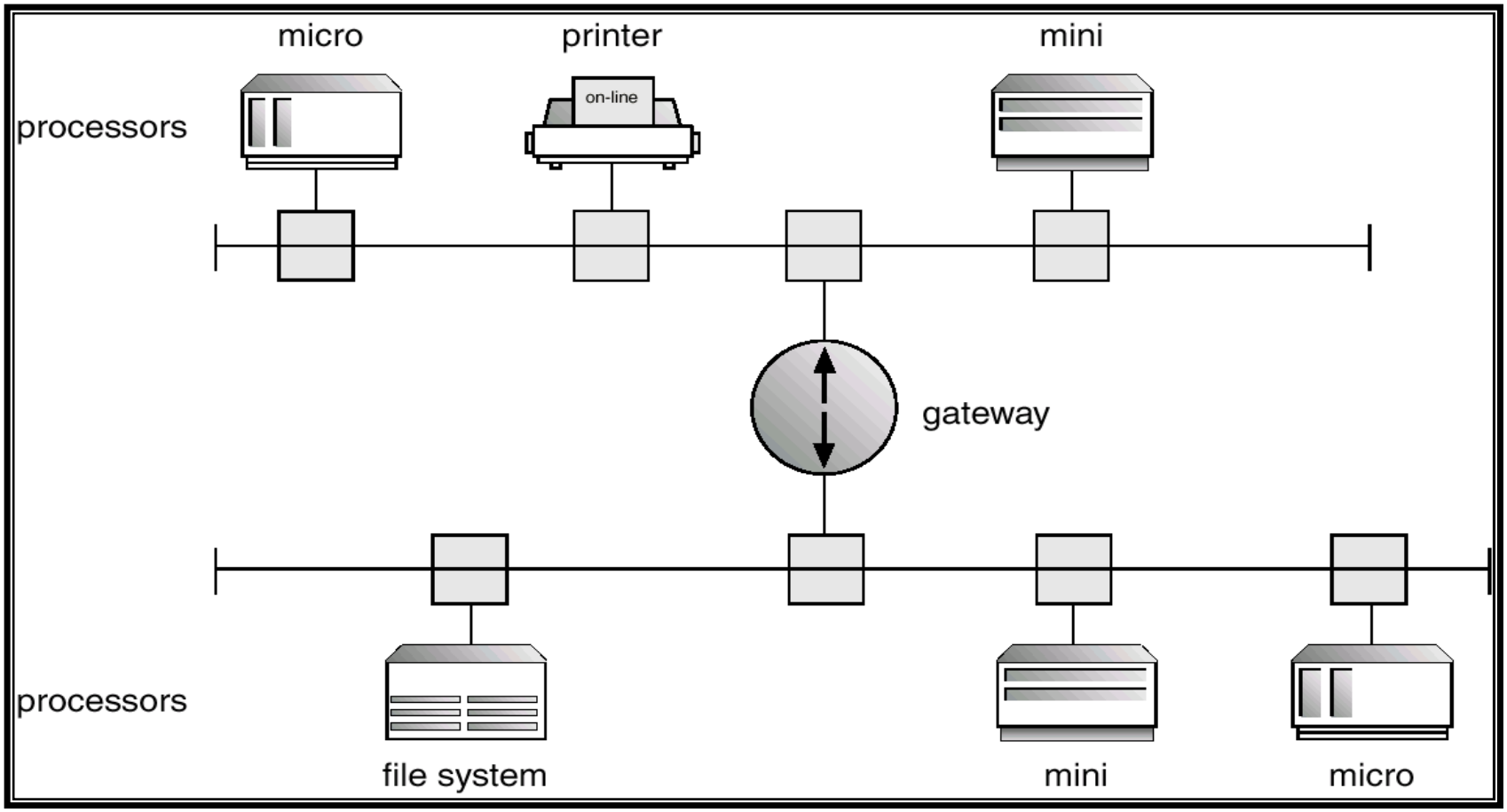
Synchronization

- Problems
  - Interrupt can occur at any time and may interfere with the interrupted code
  - OS must be able to synchronize concurrent processes
- Synchronization
  - Turn off/on interrupts
  - Use a special atomic instructions
    - read-modify-write (e.g. INC, DEC)
    - test-and-set

Network Structure

- Local Area Networks (LAN)
  - Relay, bridge, gateway, router, switch, hub
- Wide Area Networks (WAN)

Local Area Network Structure



Wide Area Network Structure