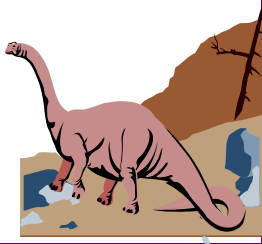
Chapter 2: Computer-System Structures (6th Edition)

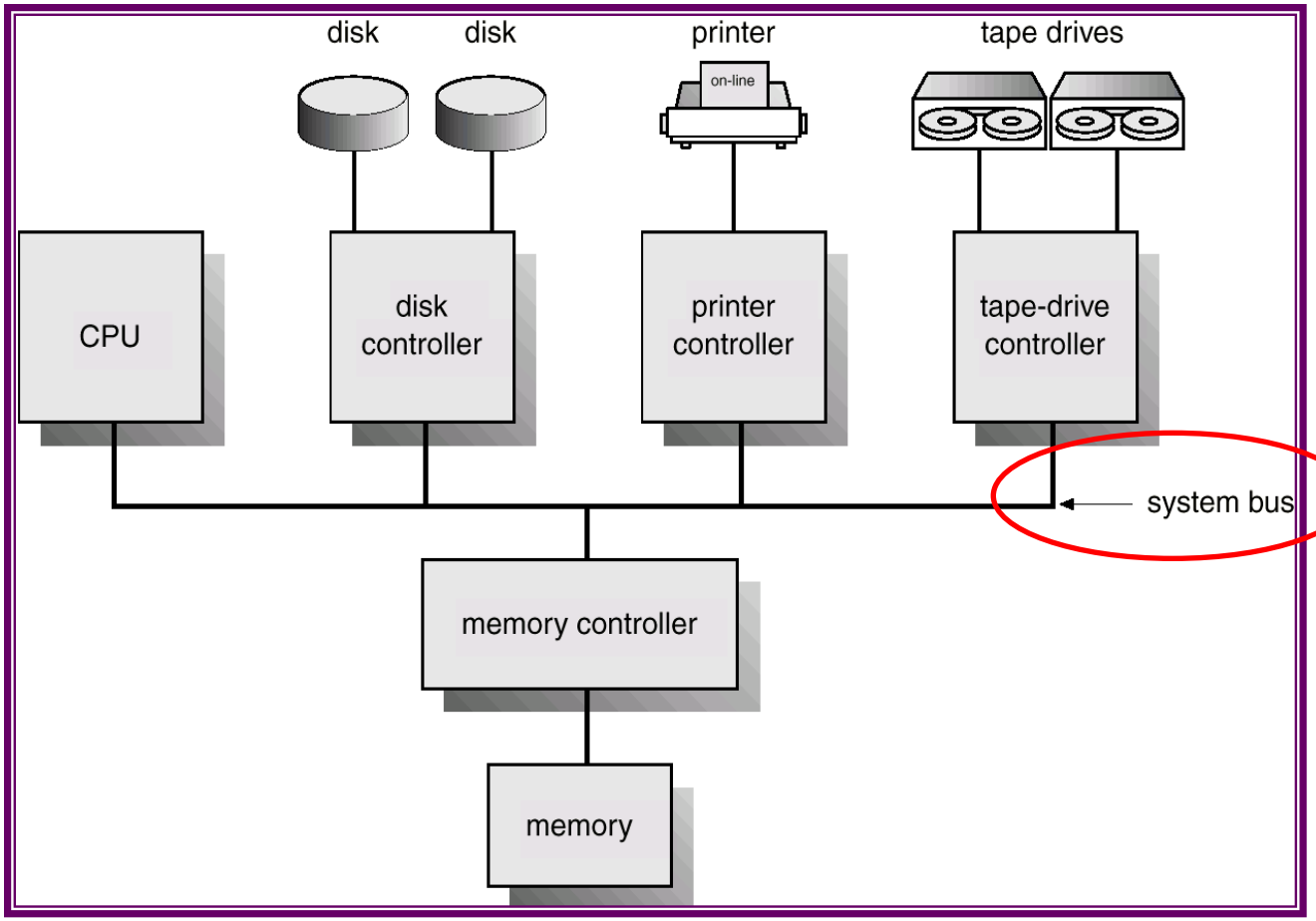
肖 卿 俊
办公室：计算机楼532室
电邮：csqjxiao@seu.edu.cn
主页：http://cse.seu.edu.cn/PersonalPage/csqjxiao
电话：025-52091023

Chapter 2: Computer-System Structures (6th Ed.)

- Computer System Operation
- I/O Structure
- Storage Structure
- Storage Hierarchy (层次结构)
- Hardware Protection
- Network Structure



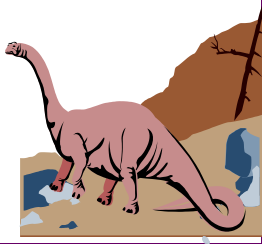
Computer-System Architecture



- How does the OS get into system?
- How do we request for services?
- How does the OS know something has happened

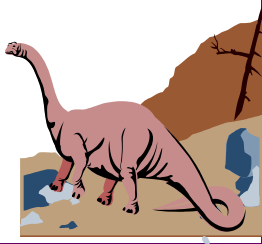
Computer-System Operation

- 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.



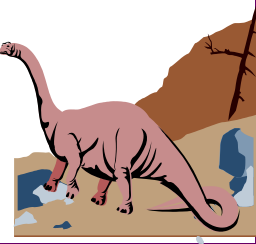
Computer-System Operation (Cont.)

- CPU moves data from/to main memory to/from local buffers of controllers
- 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*.

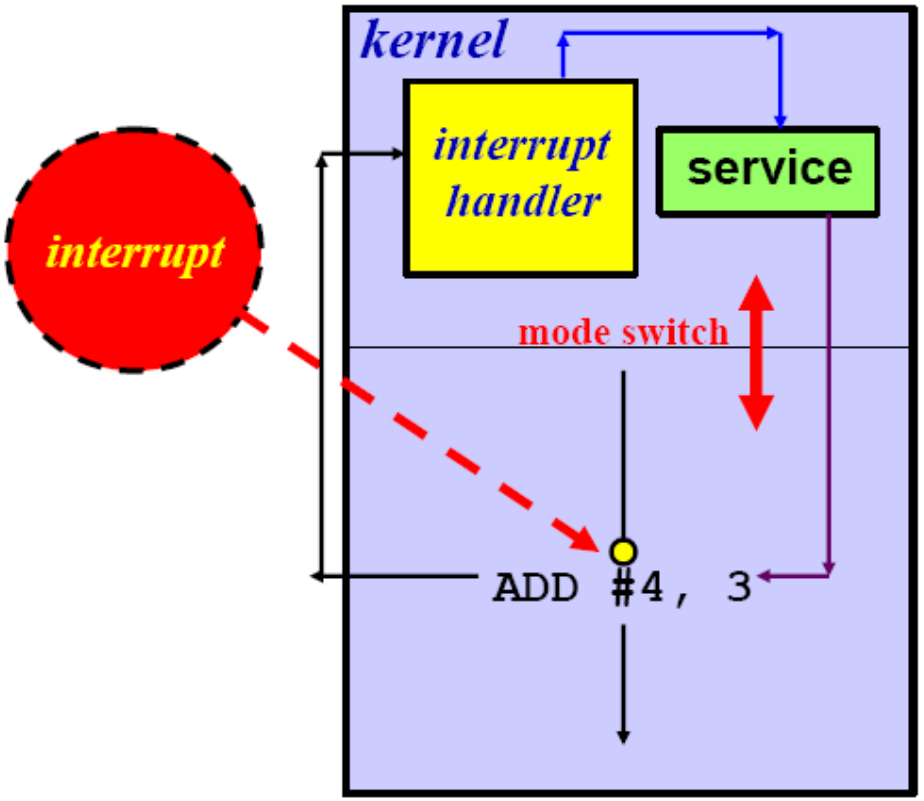


Interrupt

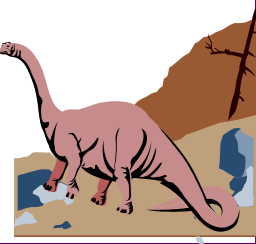
- An event that requires the attention of the OS is an *interrupt*. These events include the completion of an I/O, a keypress, a request for service, a division by zero and so on.
- Interrupts may be triggered by software.
  - An interrupt generated by software (*i.e.*, division by 0, page fault, debug breakpoint) is usually referred to as a *trap*.
- Modern operating systems are *interrupt driven*, meaning the OS is in action only if an interrupt occurs.



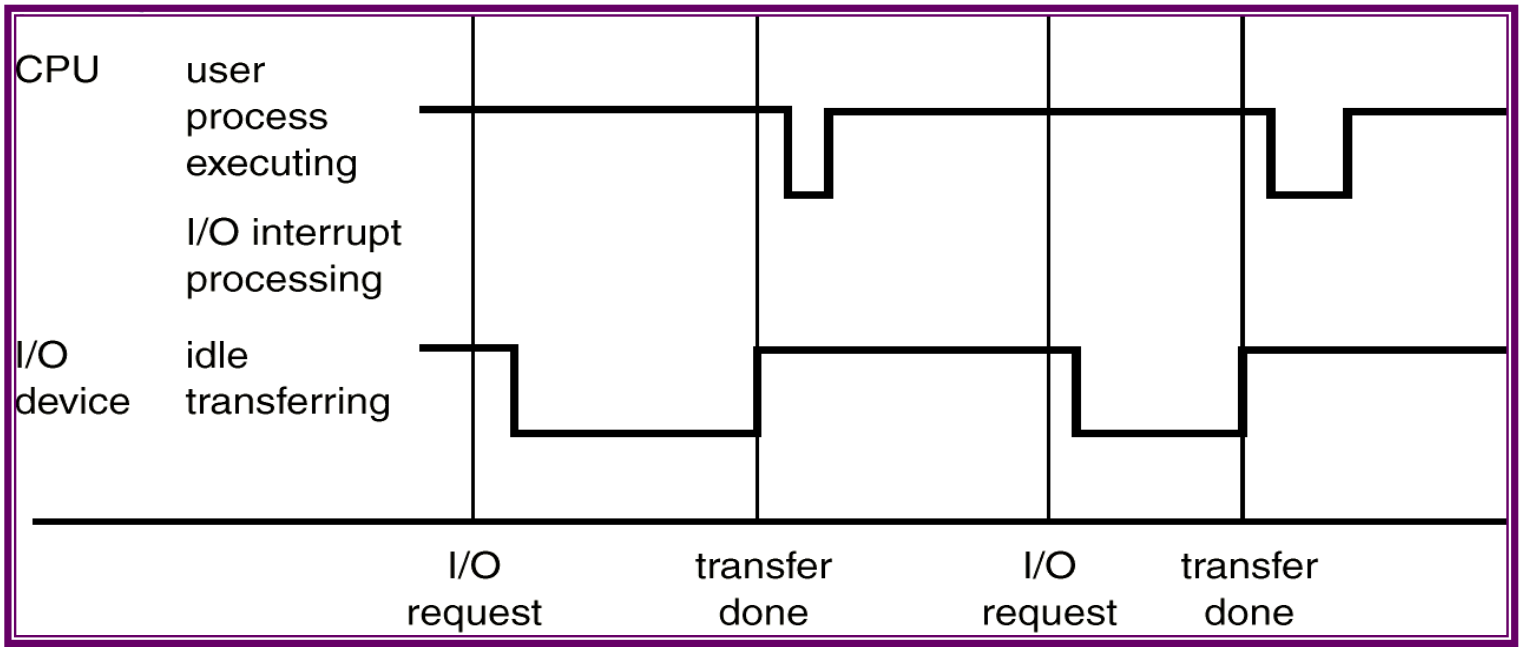
What is Interrupt driven?

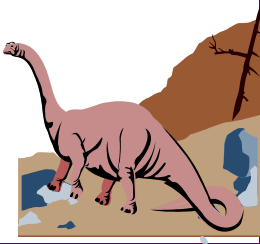


- The OS is activated by an interrupt.
- The executing program is suspended.
- Control is transferred to the OS.
- Program continues when the service completes



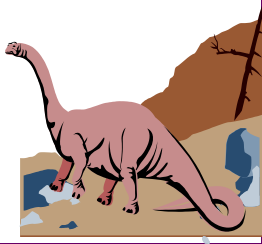
Interrupt Time Line For a Single Process Doing Output





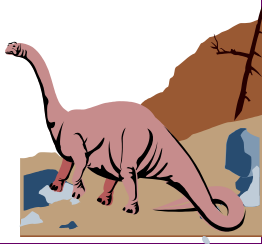
Common Functions of Interrupts

- 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*.

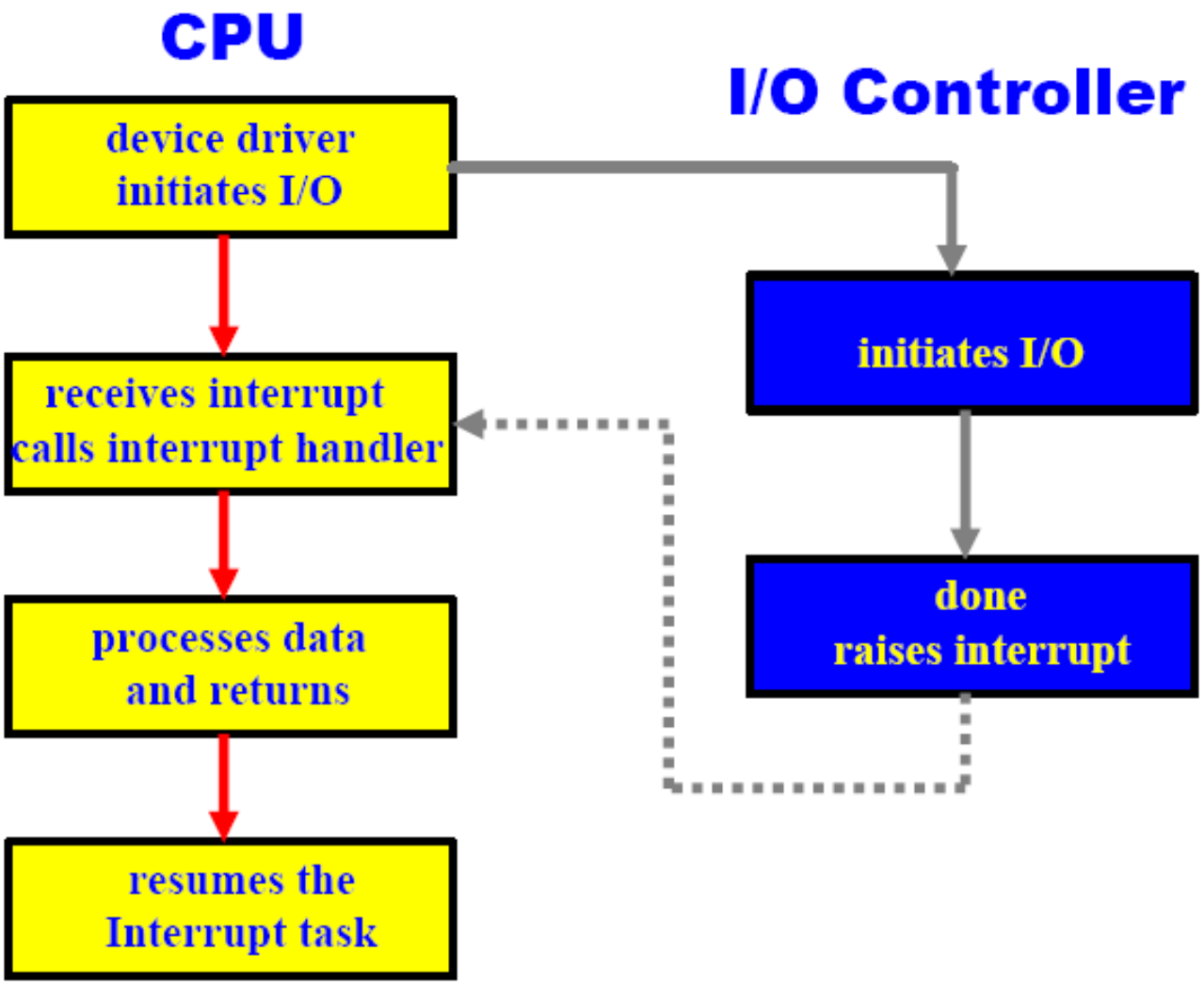


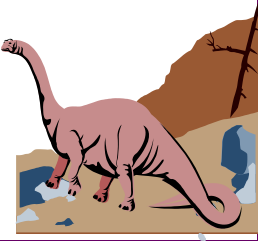
Interrupt Handling

- 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

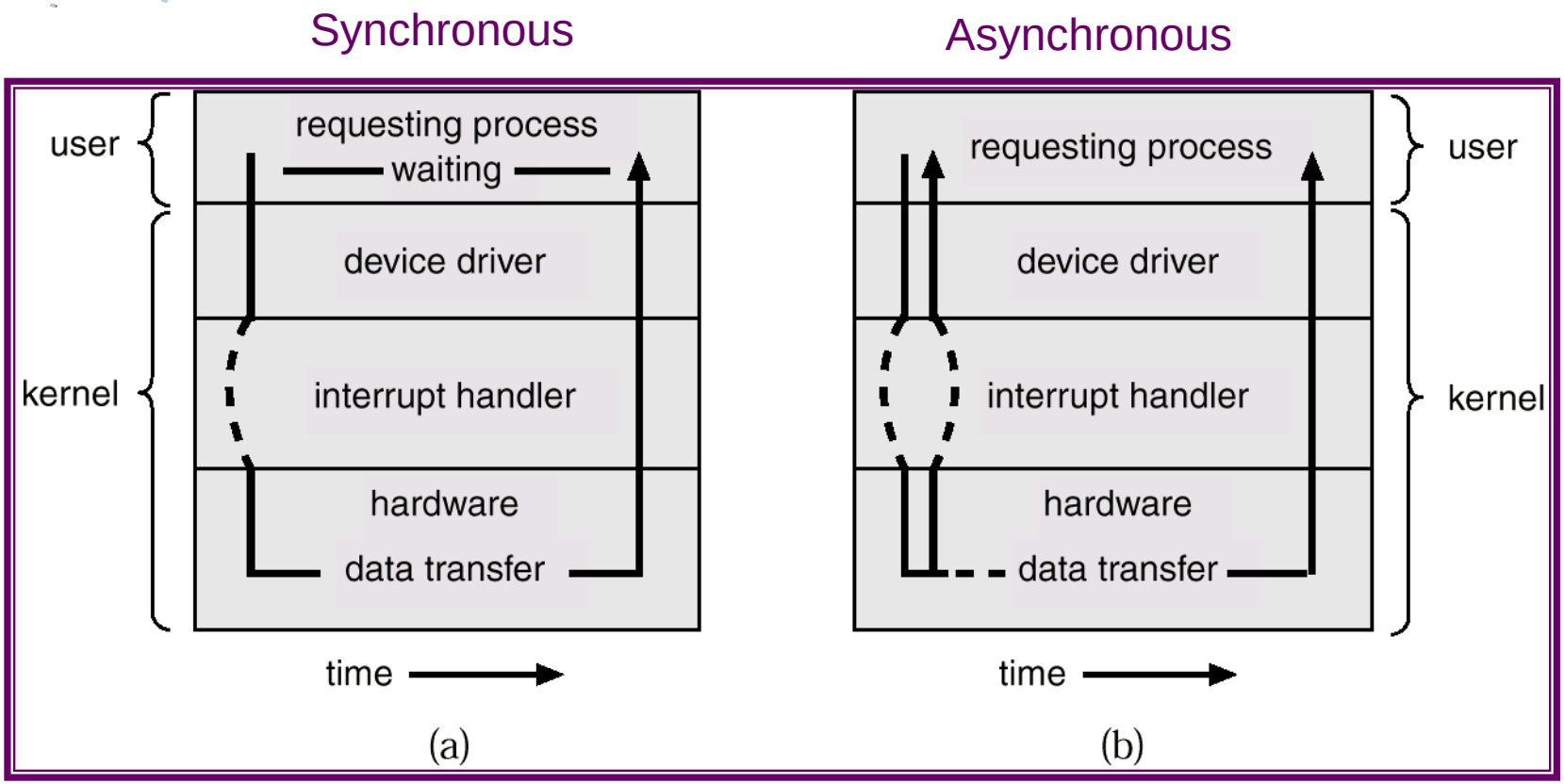


I/O Interrupt



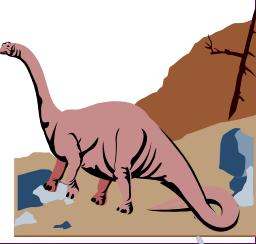


Two I/O Methods: Synchronous vs. Asynchronous



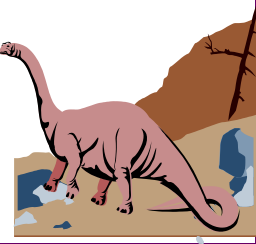
Synchronous IO from User View

- After I/O starts, control returns to user program only upon I/O completion.
  - Wait instruction idles the user process 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.

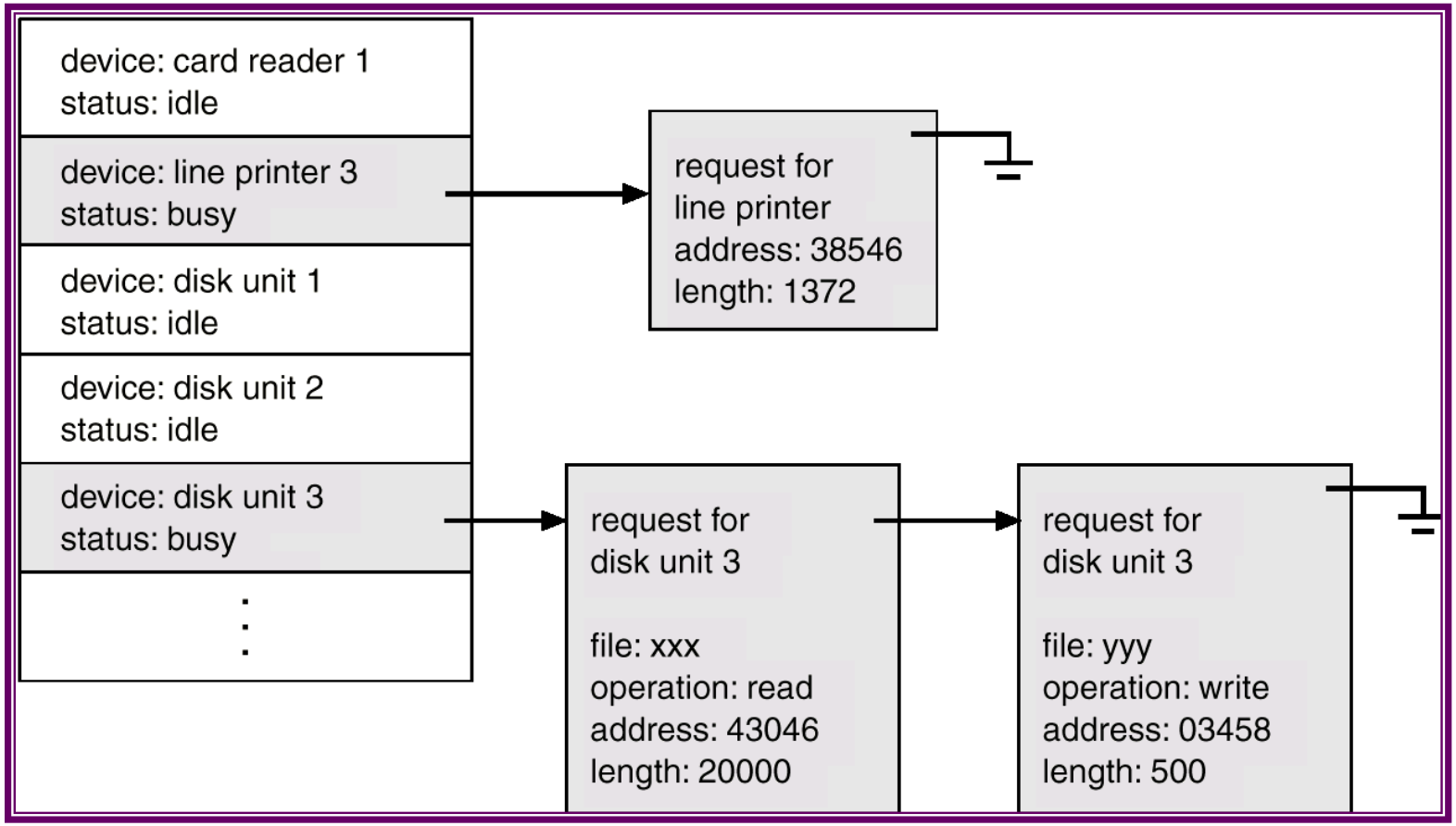


Asynchronous IO from User View

- 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.
- How does the user program receive the notification of I/O completion?

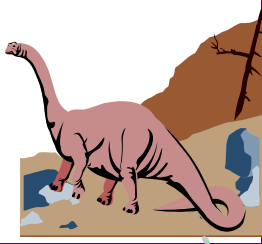


Device-Status Table



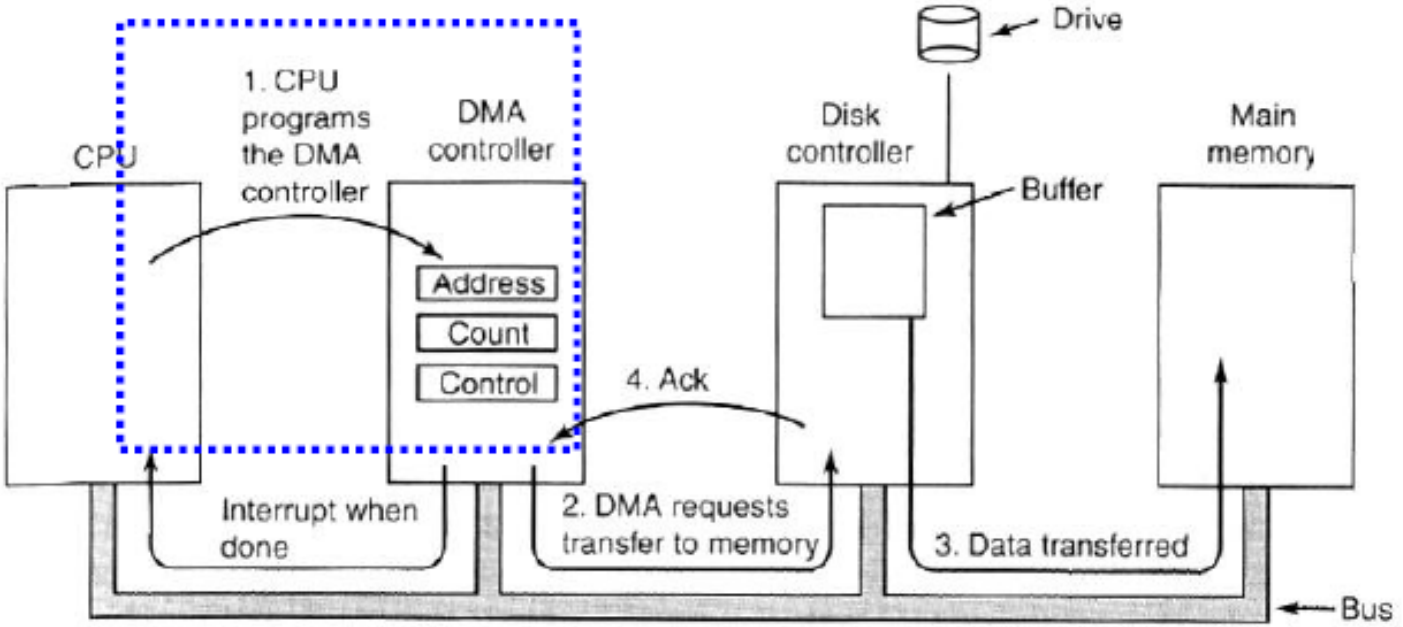
Direct Memory Access Structure (1/2)

- 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.



Direct Memory Access Structure (2/2)

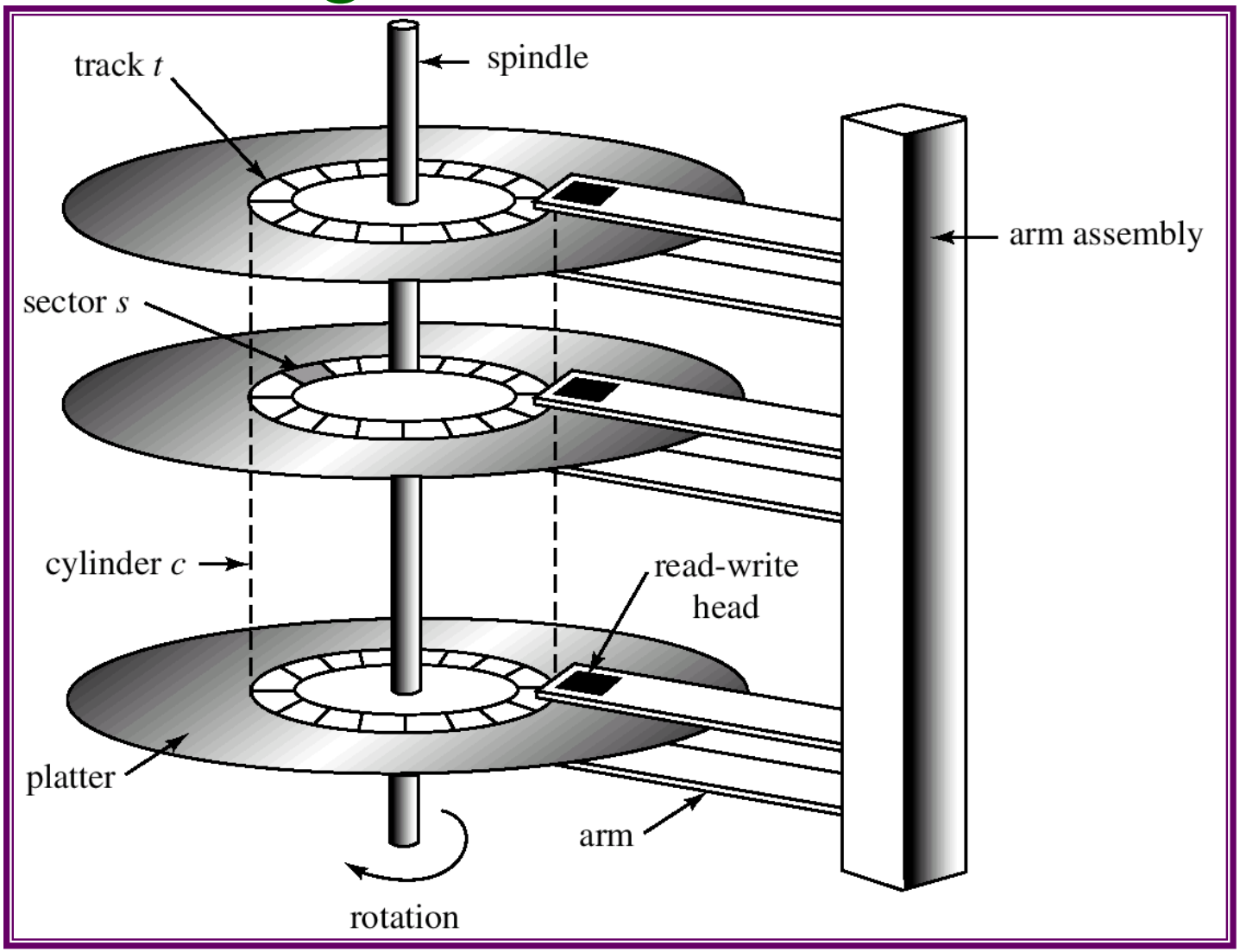
The CPU gives the controller (1) disk address, (2) memory address for storing the block, and (3) a byte count. Then, the CPU goes back to work.



Storage Structure

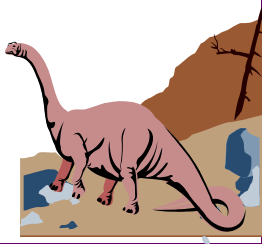
- Main memory – only large storage media that the CPU can access directly.
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into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

Moving-Head Disk Mechanism



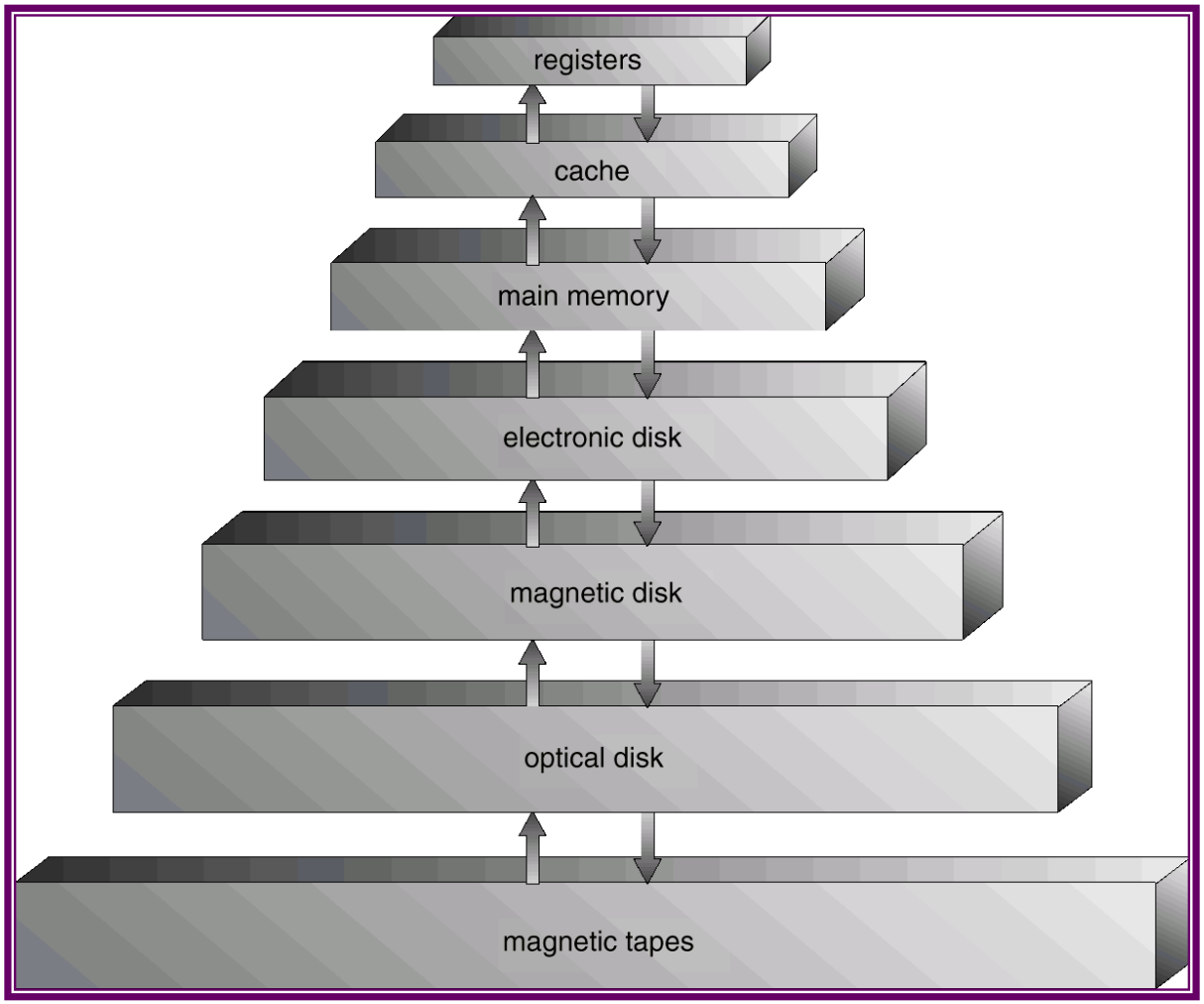
Storage Hierarchy

- 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.

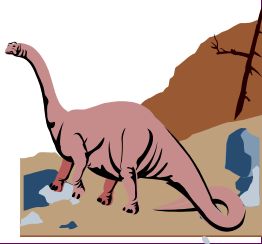


Storage-Device Hierarchy

Speed

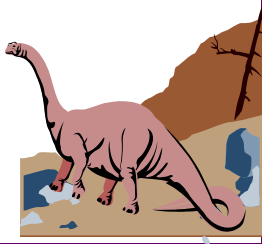


Volume

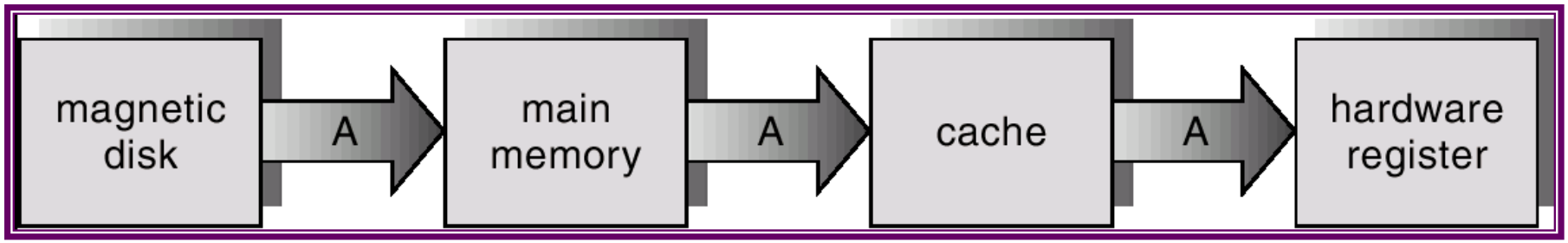


Caching

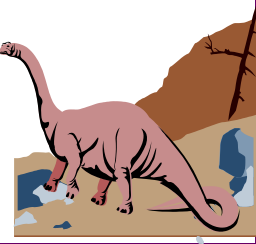
- Use of high-speed memory to hold recently-accessed data.
- Requires a *cache management* policy.
- Caching introduces another level in storage hierarchy. This requires data that is simultaneously stored in more than one level to be *consistent*.



Migration of A From Disk to Register

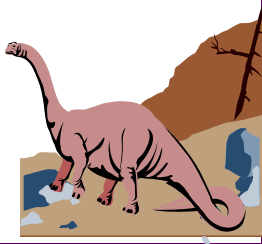


How to maintain the Cache Consistency?



Hardware Protection

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

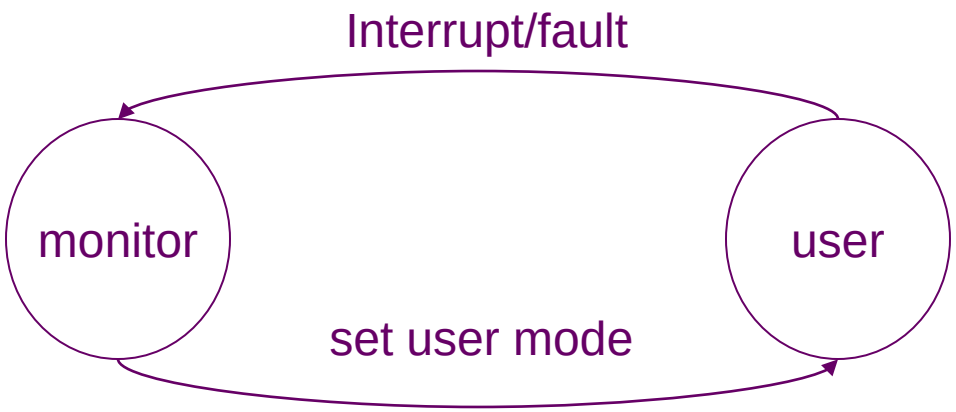


Dual-Mode Operation

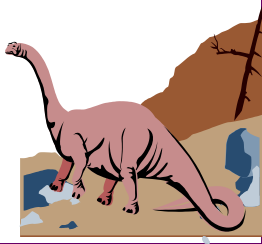
- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
  1. *User mode* – execution done on behalf of a user.(用户态或目态)
  2. *Monitor mode* (also *kernel mode* or *system mode*) – execution done on behalf of operating system. (核心态或管态)

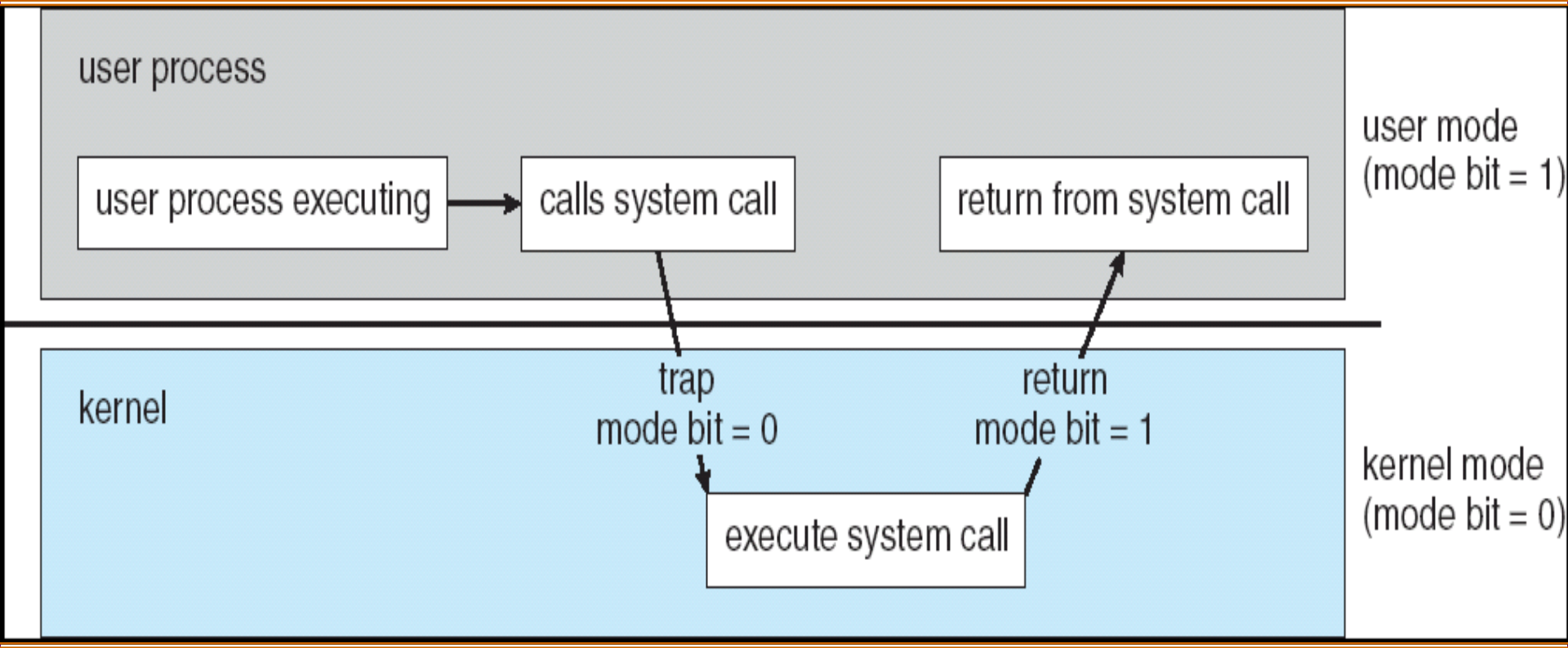
Dual-Mode Operation (Cont.)

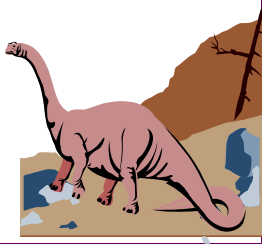
- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.



Privileged instructions can be issued only in monitor mode.

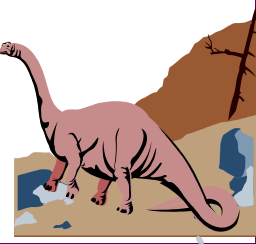



user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

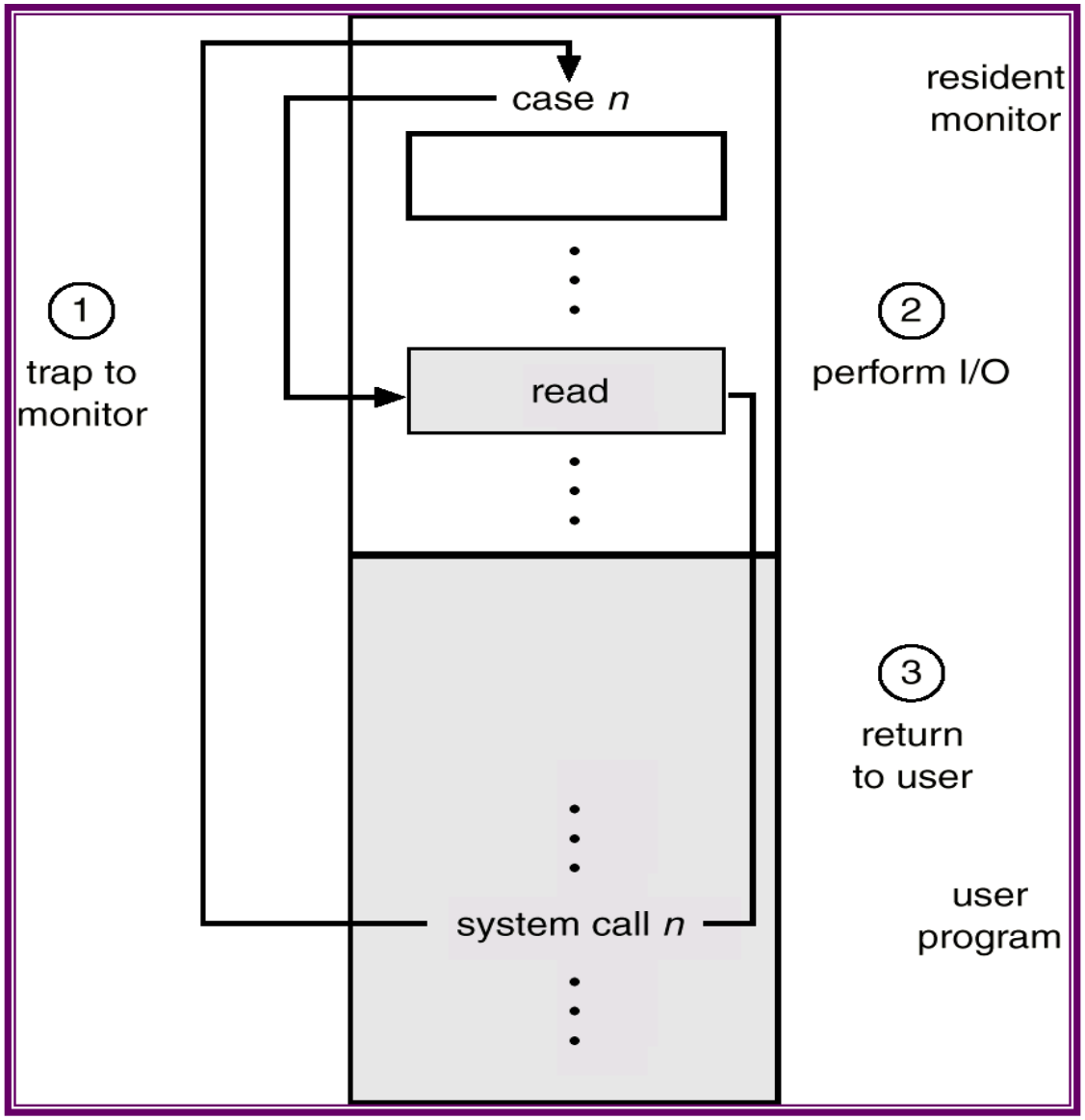


I/O Protection

- All I/O instructions are privileged instructions（特权指令）, e.g., send commands to IO controllers
- 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).



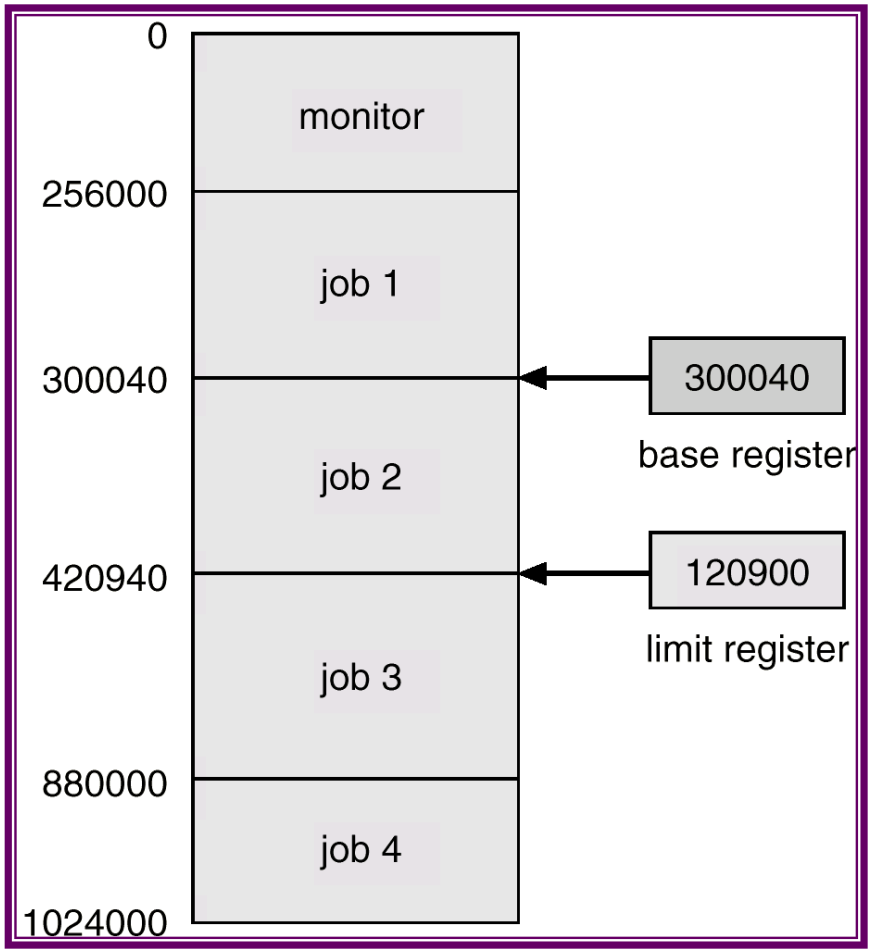
Use of A System Call to Perform I/O

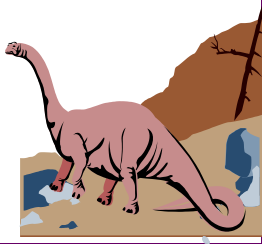


Memory Protection

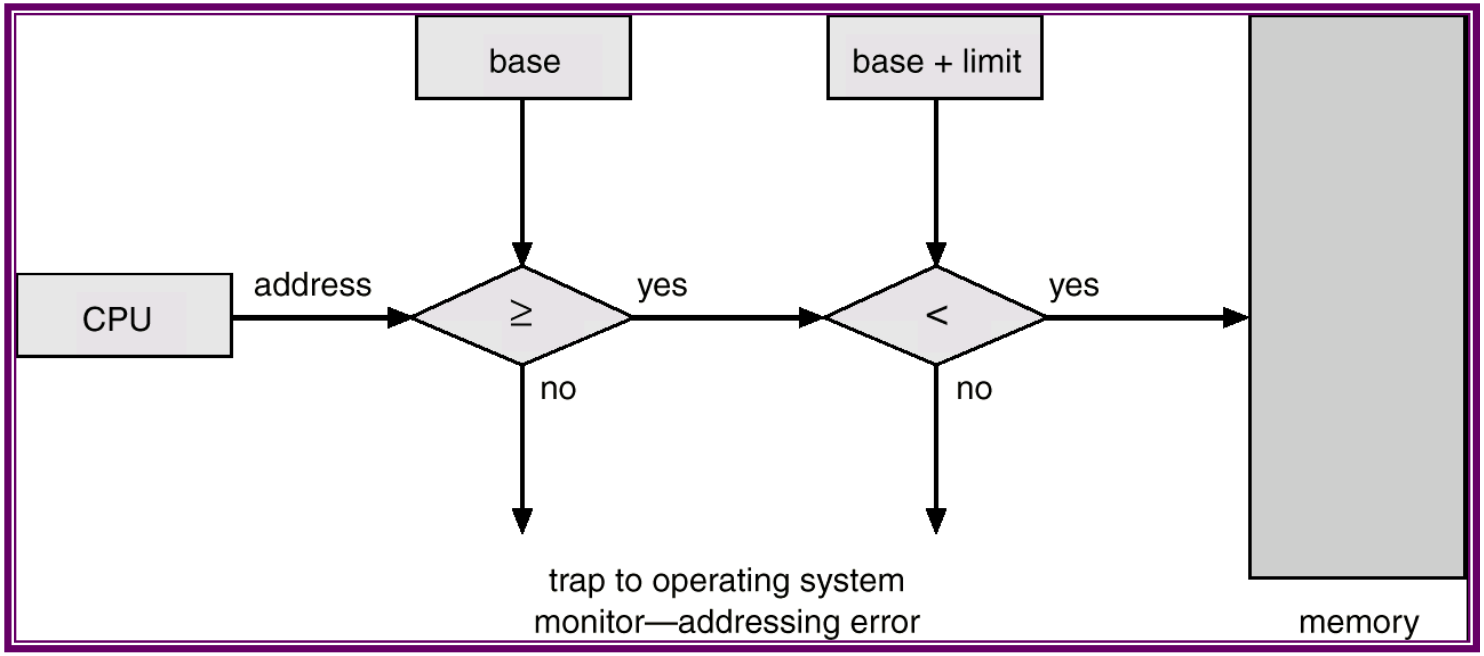
- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - **Base register**（基址寄存器）– holds the smallest legal physical memory address.
  - **Limit register**（界限寄存器） – contains the size of the range
- Memory outside the defined range is protected.

Use of A Base and Limit Register



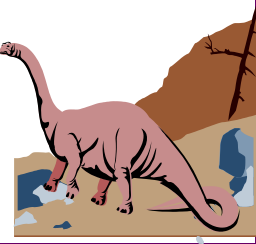


Hardware Address Protection



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 (特权指令).



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.
- Timer also used to compute the current time.
- Load-timer is a privileged instruction.