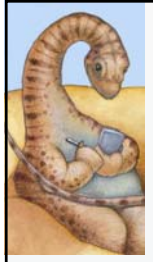
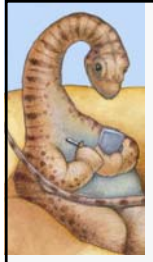
Chapter 13: I/O Systems



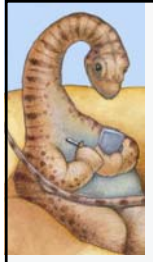
Chapter 13: I/O Systems

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- ~~STREAMS~~
- Performance



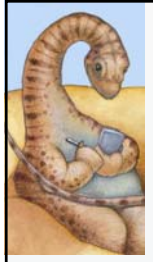
Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

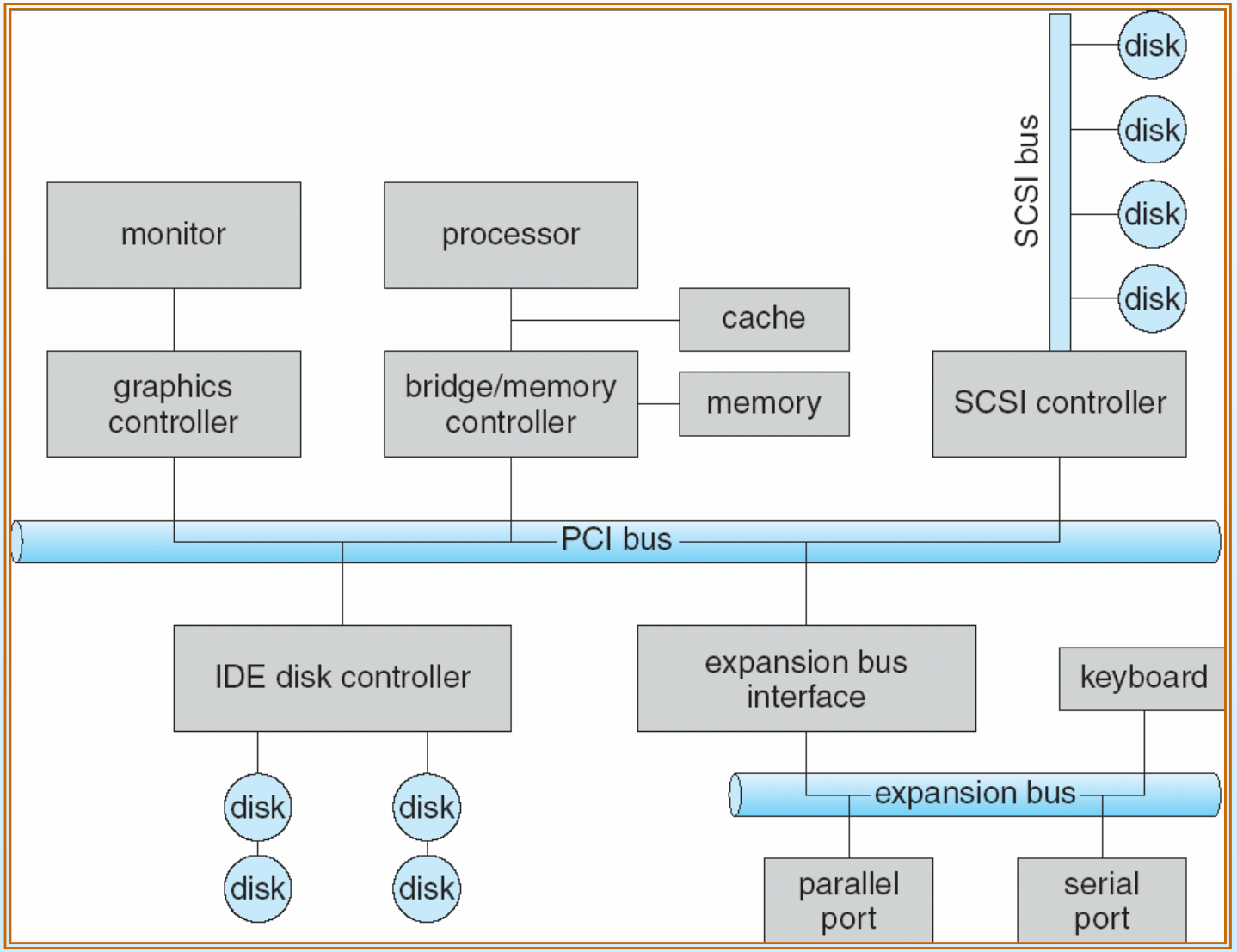


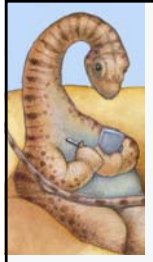
I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - **Port**
  - **Bus** (**daisy chain** or shared direct access)
  - **Controller** (**host adapter**)
- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions
  - **Memory-mapped** I/O



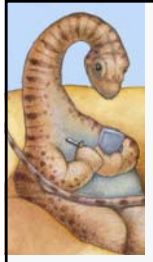
A Typical PC Bus Structure





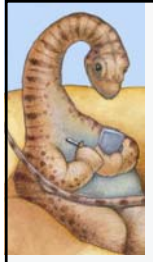
Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)



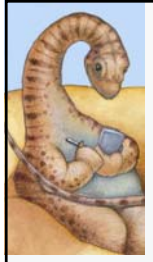
Communication between CPU and Controllers

- Controller has one or more registers for data and control signals
  - CPU communicates with controller by reading and writing bit patterns in these register
- Two approaches
  - **Direct I/O instructions**: use special I/O instruction to specify the transfer of a byte or word to an I/O port address
    - The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register
  - **Memory-mapped I/O**: the device-control registers are mapped into the address space of the CPU
    - CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers
    - For example, graphics controller



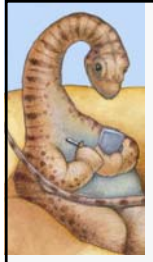
Polling

- Handshaking
  - The host repeatedly reads the *busy* bit until it becomes clear.
  - The host sets the *write* bit in the *command* register and writes a byte into the *data-out* register.
  - The host sets the *command-ready* bit.
  - When the controller notices that *command-ready* bit is set, it sets the *busy* bit.
  - The controller reads the *command* register and sees the *write* command. It reads the *data-out* register to get the byte.
  - The controller clears the *command-ready* bit, clears the *error* bit in the status register to indicate that the device I/O succeeded, and clears the *busy* bit to indicate that it is finished.
- In step 1, the host is **busy-waiting** or **polling**.



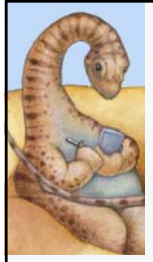
Polling (cont.)

- Determines state of device
  - command-ready
  - busy
  - error
- **Busy-wait** cycle to wait for I/O from device
  - Polling becomes inefficient when it is attempted repeatedly yet rarely finds a device to be ready for service.

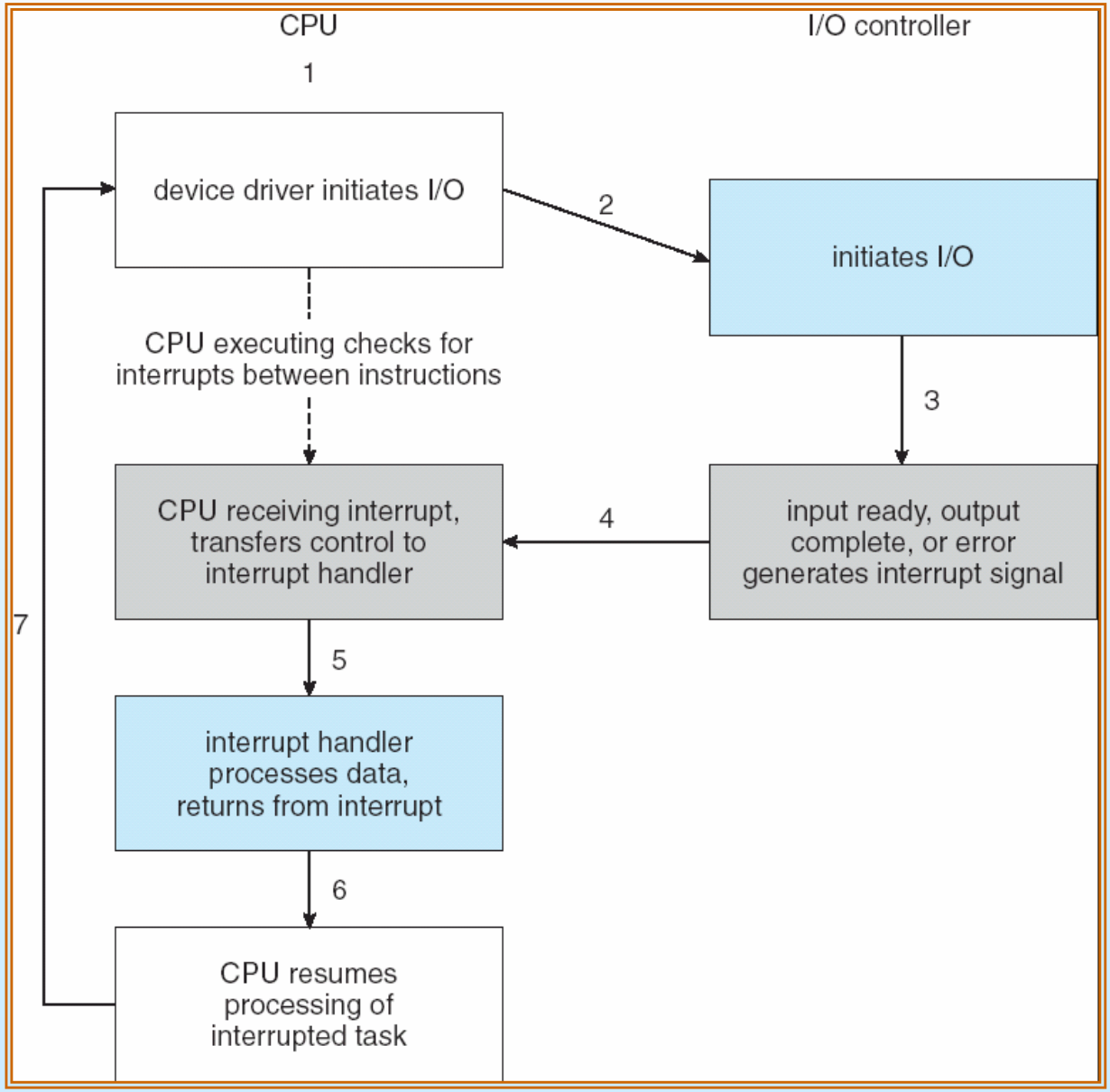


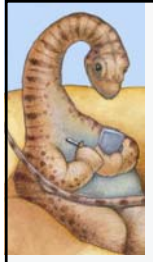
Interrupts

- CPU senses **Interrupt-request line**, triggered by I/O device, after executing every instruction.
- If asserted, CPU performs a state save and jumps to the **Interrupt handler** routine at a fixed address in memory.
  - The interrupt handler determines the cause, performs processing, performs state restore, and executes a return to the execution state prior to the interrupt.
- Need more sophisticated interrupt-handling features
  - Ability to defer interrupt handling during critical processing
  - An efficient way to dispatch to the proper interrupt handler for a device without polling all the devices to see which one raised the interrupt
  - Multilevel interrupts
    - OS can distinguish between high- and low-priority interrupts and can respond with the appropriate degree of urgency.



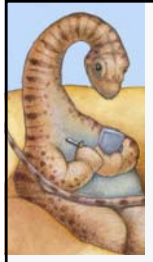
Interrupt-Driven I/O Cycle





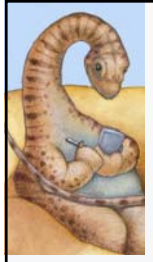
Interrupts (cont.)

- Two interrupt-request line
  - One is nonmaskable, reserved for events such as unrecoverable memory errors
  - The other one is **maskable** which can be turned off by CPU to ignore or delay
- Use address – a number that selects a specific interrupt-handling routine
  - Done by **interrupt vector** to dispatch interrupt to correct handler
    - Contains the memory addresses of specialized interrupt handlers
    - To reduce the need for a single interrupt handler to search all possible sources to determine which one needs service



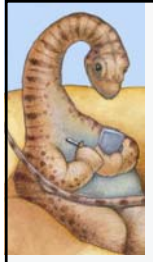
Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

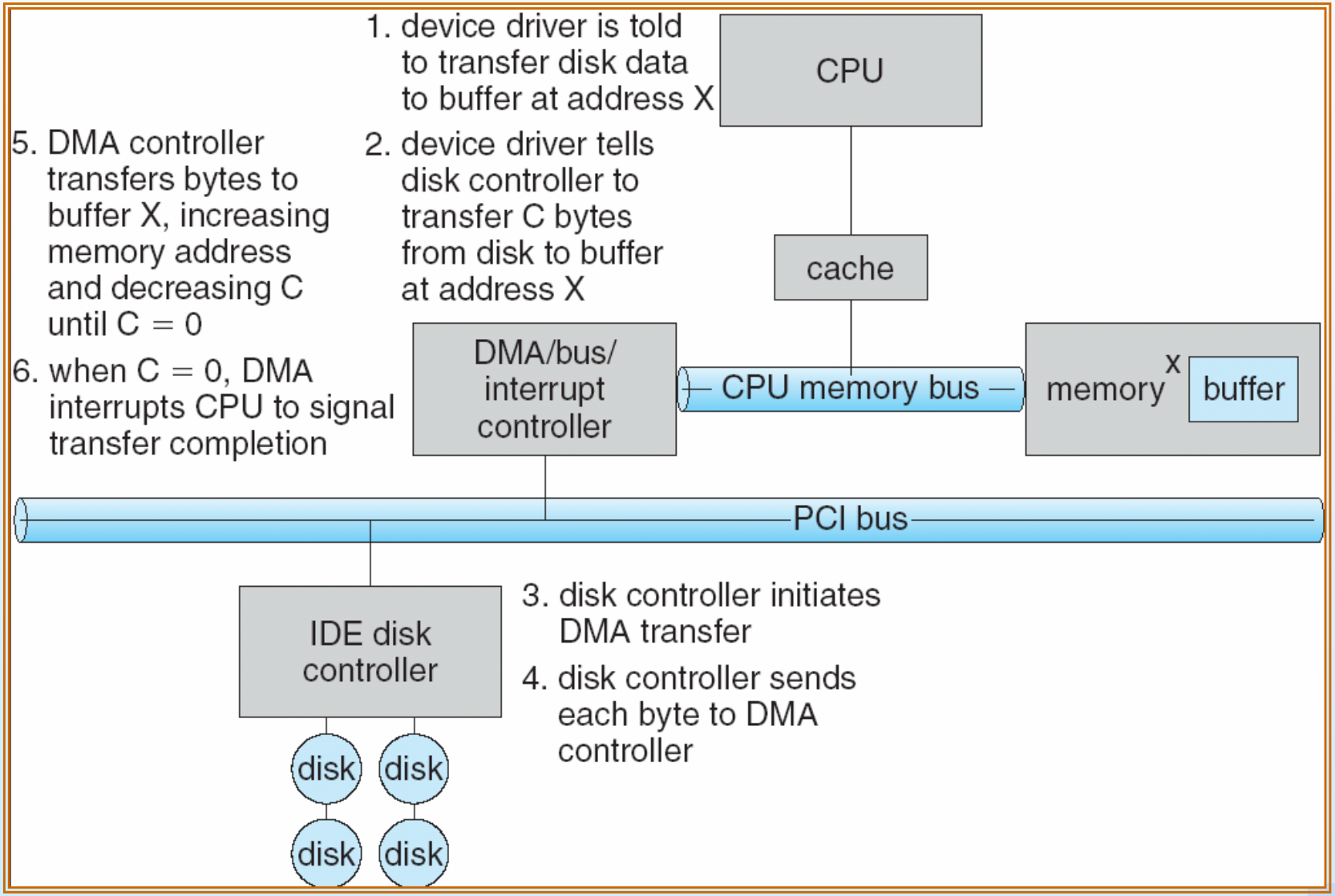


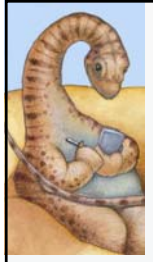
Direct Memory Access

- Used to avoid **programmed I/O** for large data movement
  - It is wasteful to use CPU to watch status bits and to feed data into a controller register one byte at a time
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory



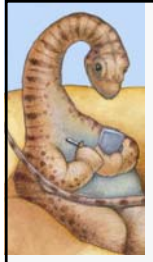
Step in A DMA Transfer



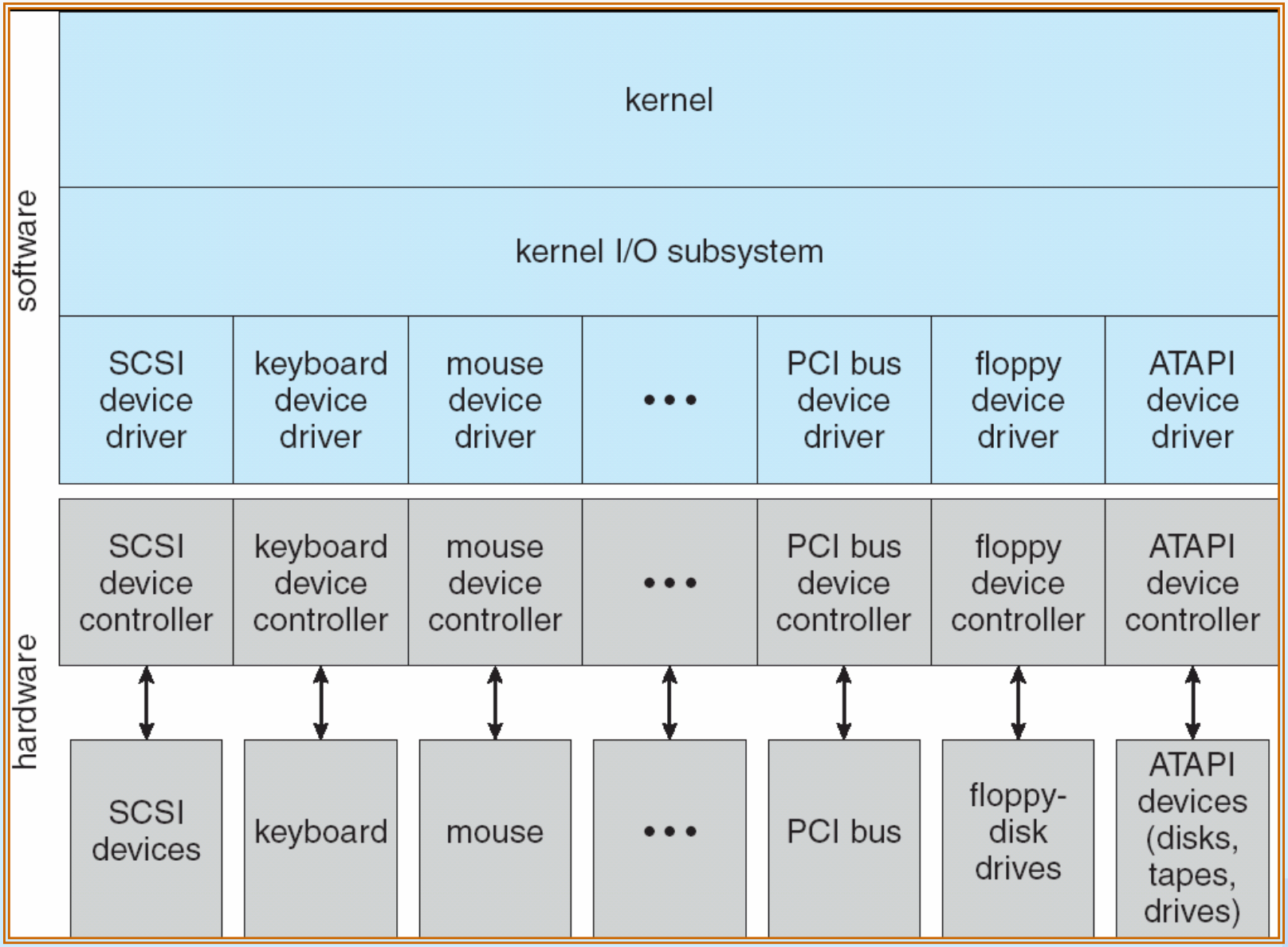


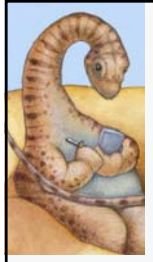
Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- Devices vary in many dimensions
  - **Character-stream** or **block**
  - **Sequential or random-access**
  - **Sharable or dedicated**
  - **Speed of operation**
  - **read-write, read only,** or **write only**



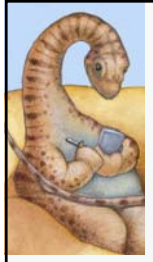
A Kernel I/O Structure





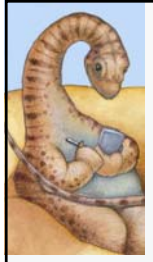
Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk



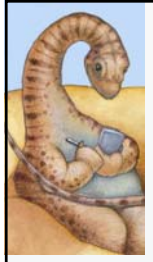
Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character devices include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top allow line editing



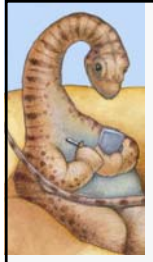
Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows NT/9*x*/2000 include socket interface
  - Separates network protocol from network operation
  - Includes `select` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)



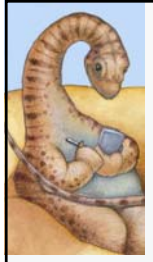
Clocks and Timers

- Provide current time, elapsed time, timer
- **Programmable interval timer** used for timings, periodic interrupts
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers



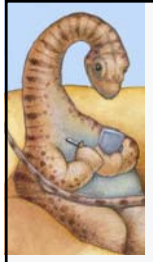
Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

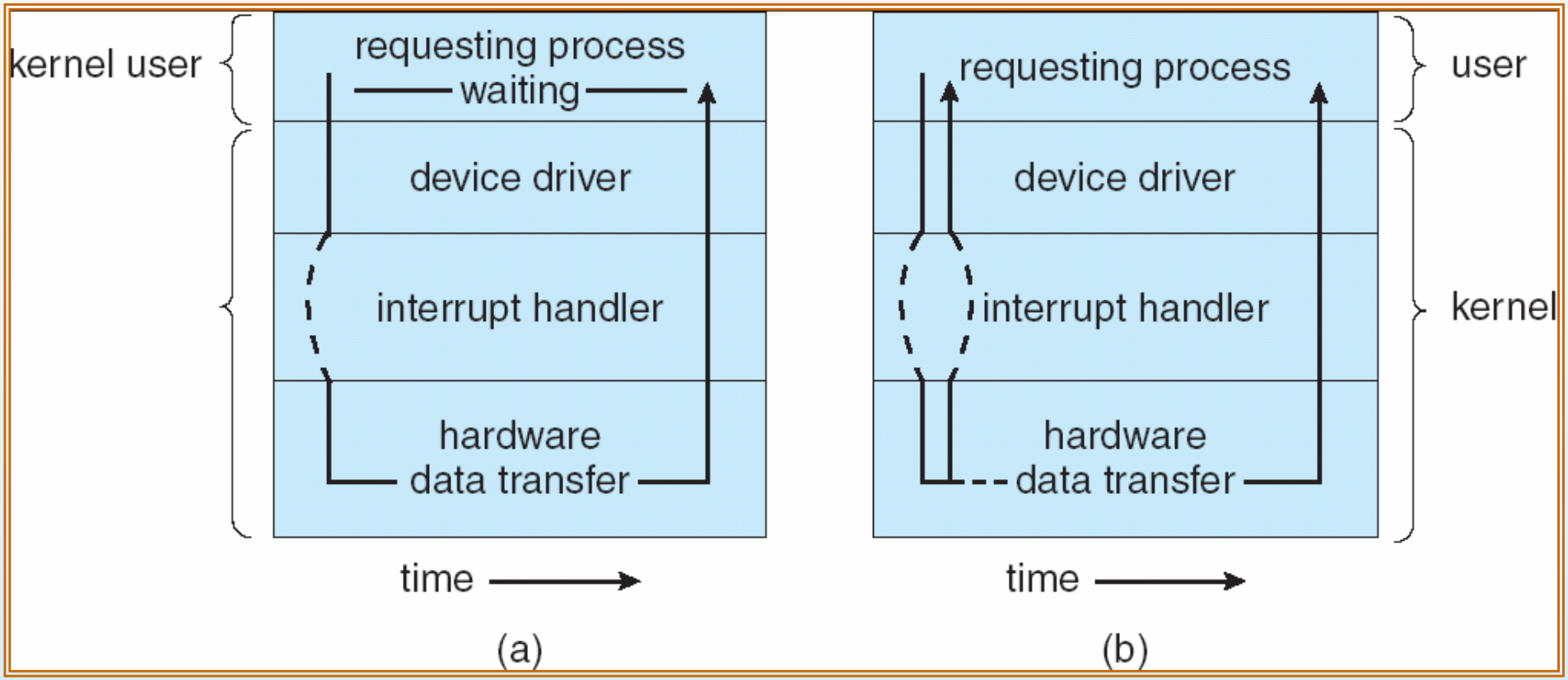


Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”

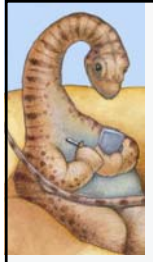


Two I/O Methods:(a)Synchronous and (b)asynchronous

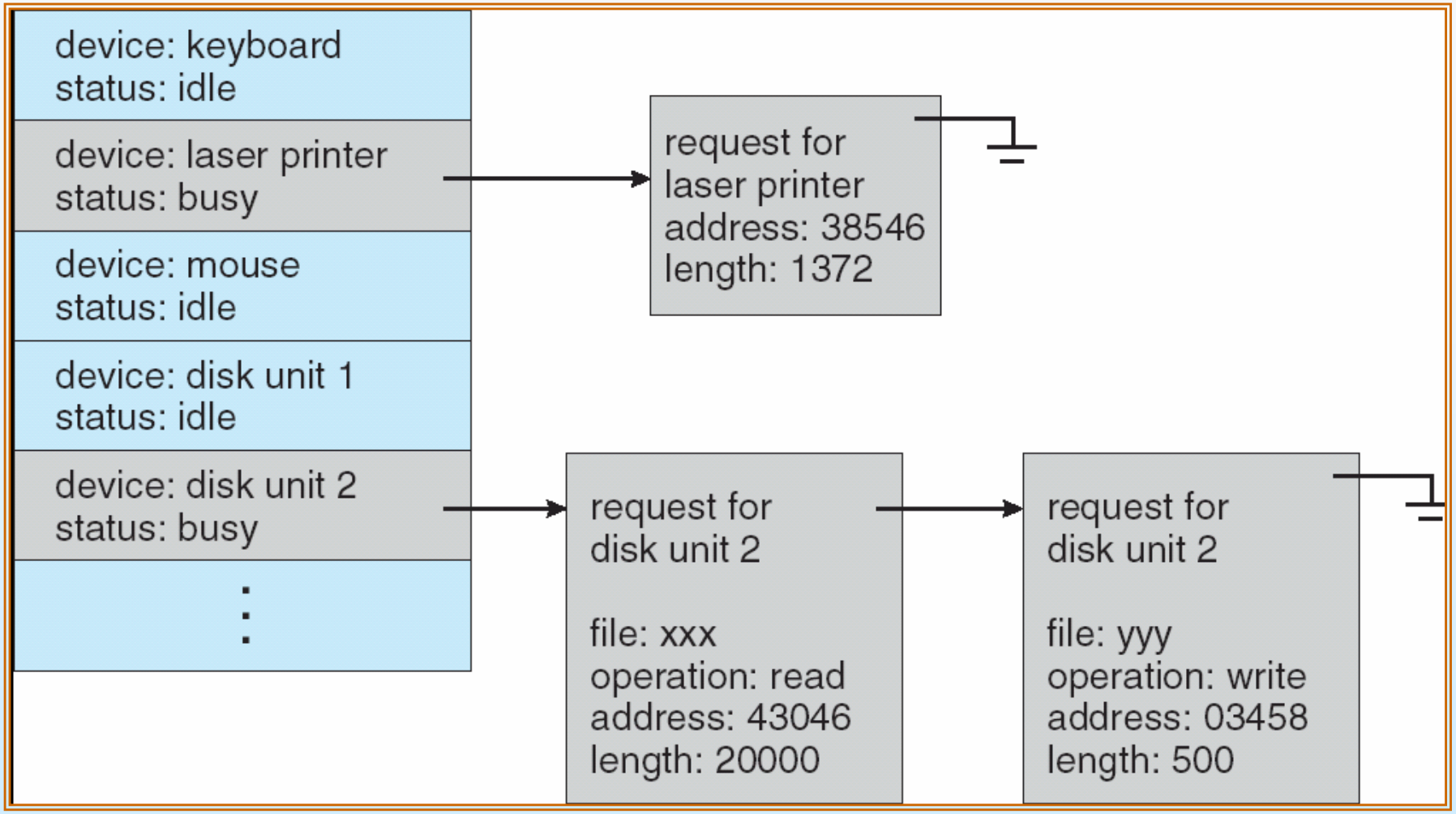


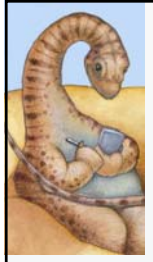
Synchronous

Asynchronous

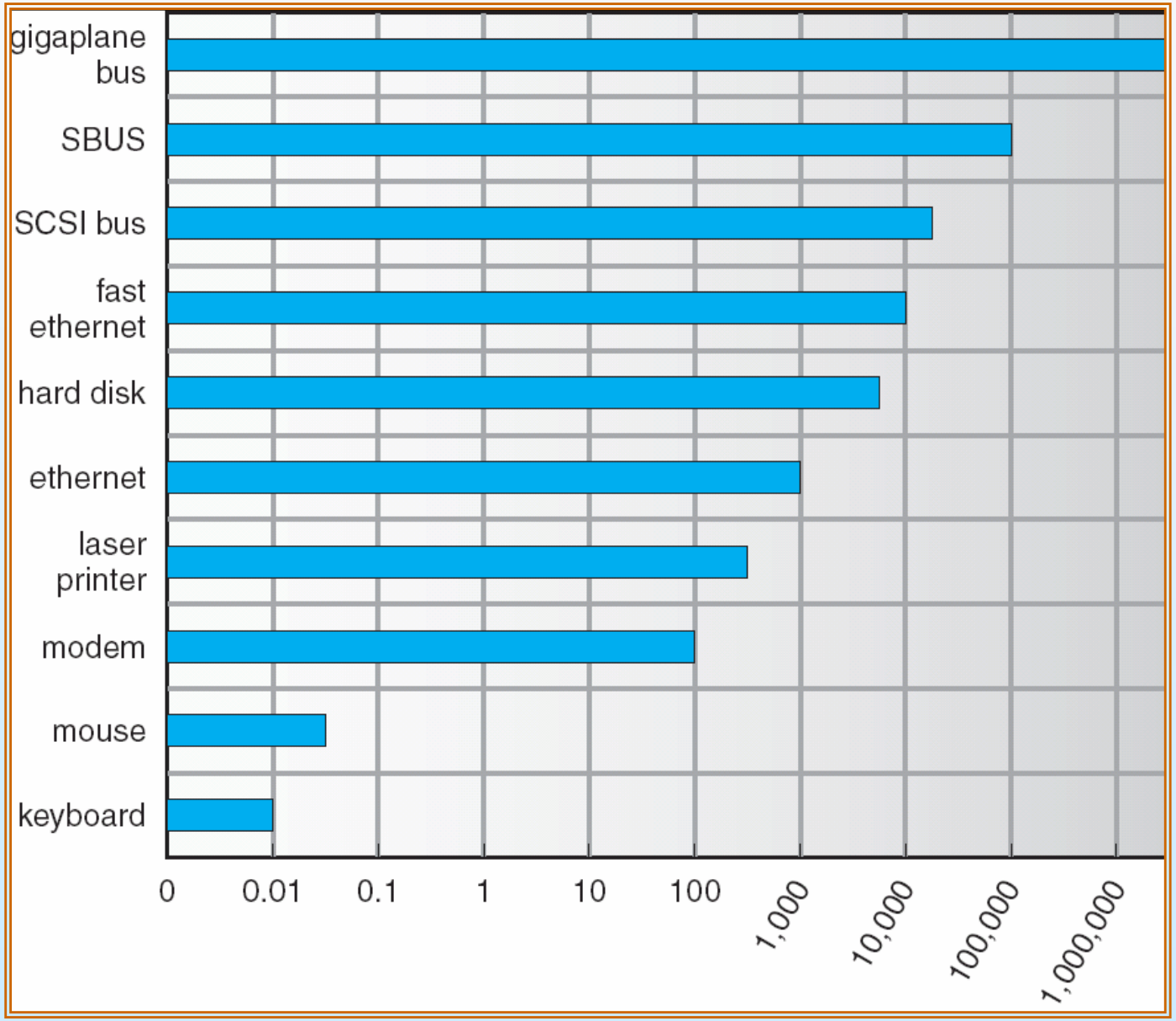


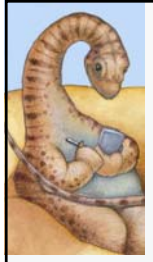
Device-status Table





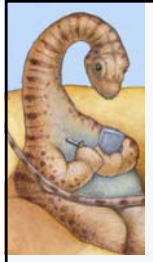
Sun Enterprise 6000 Device-Transfer Rates(logarithmic)





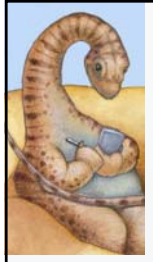
Error Handling

- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports



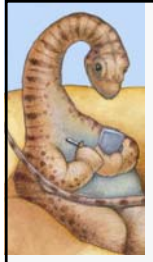
I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

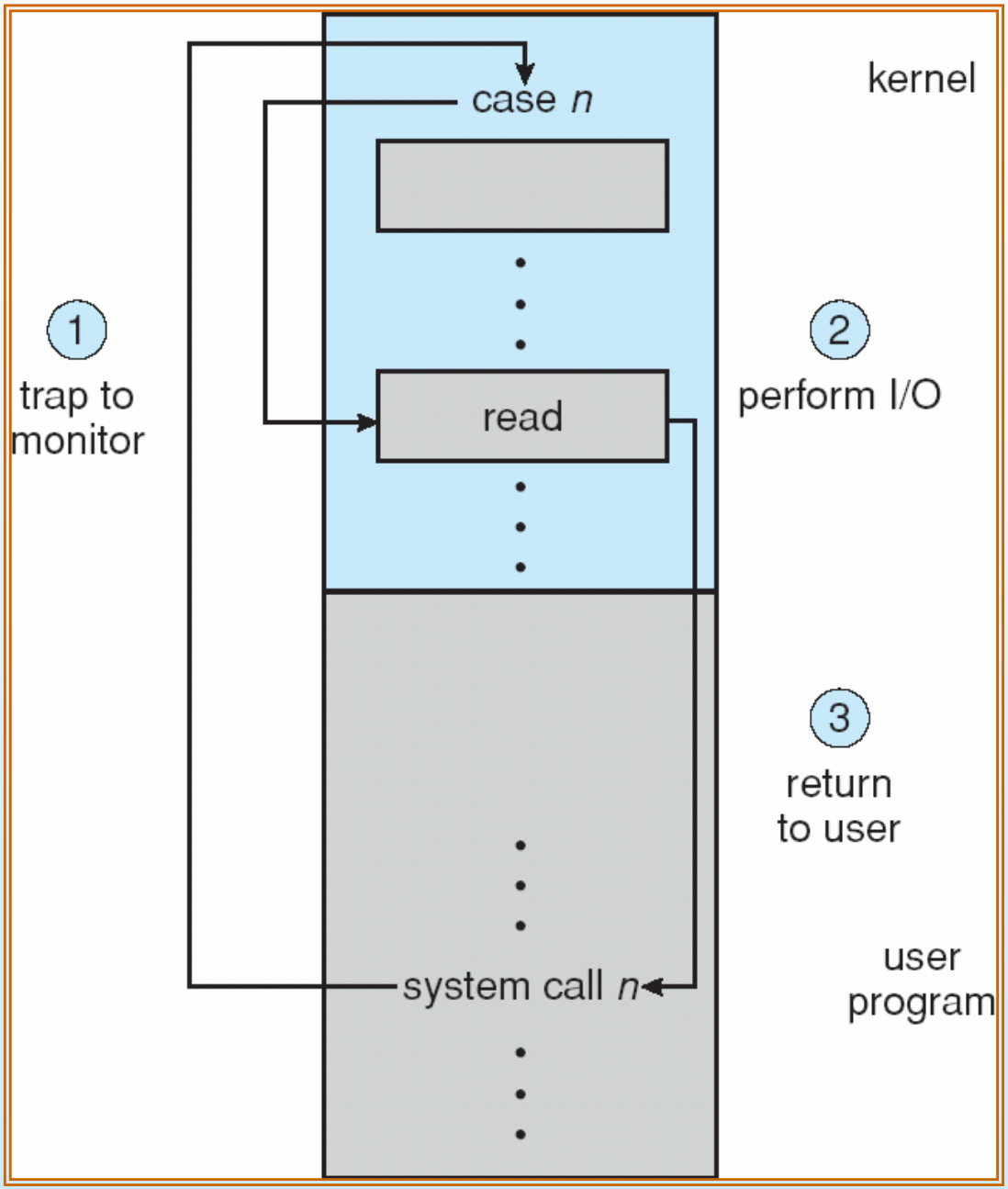


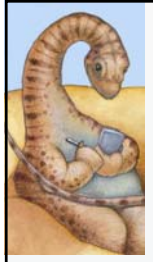
Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- Some use object-oriented methods and message passing to implement I/O

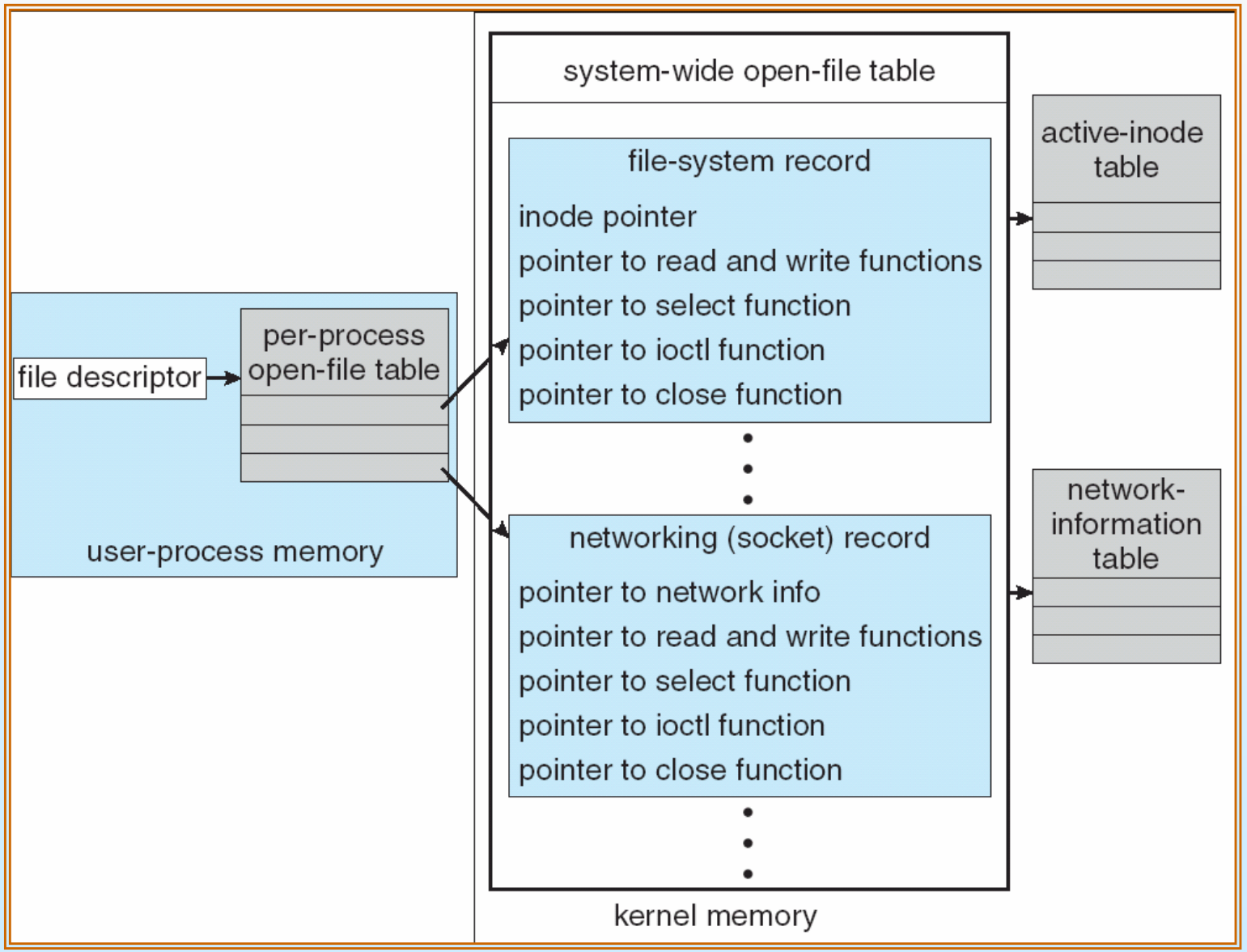


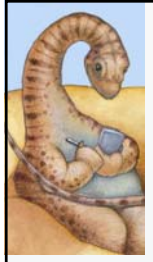
Use of a System Call to Perform I/O





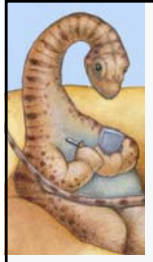
UNIX I/O Kernel Structure



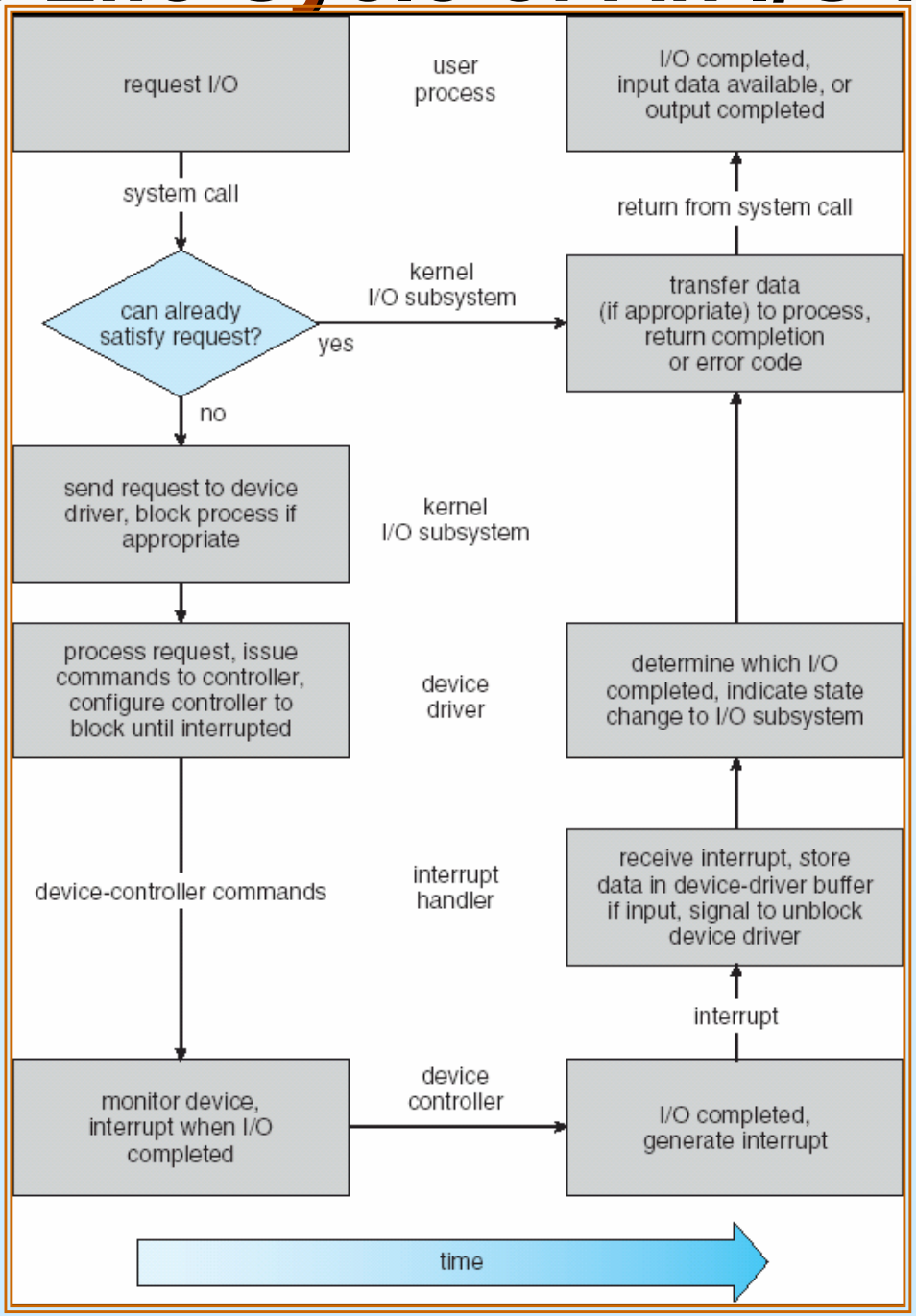


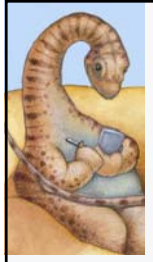
Transforming I/O Requests to Hardware Operations

- Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process



The Life Cycle of An I/O Request





Performance

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

End of Chapter 13