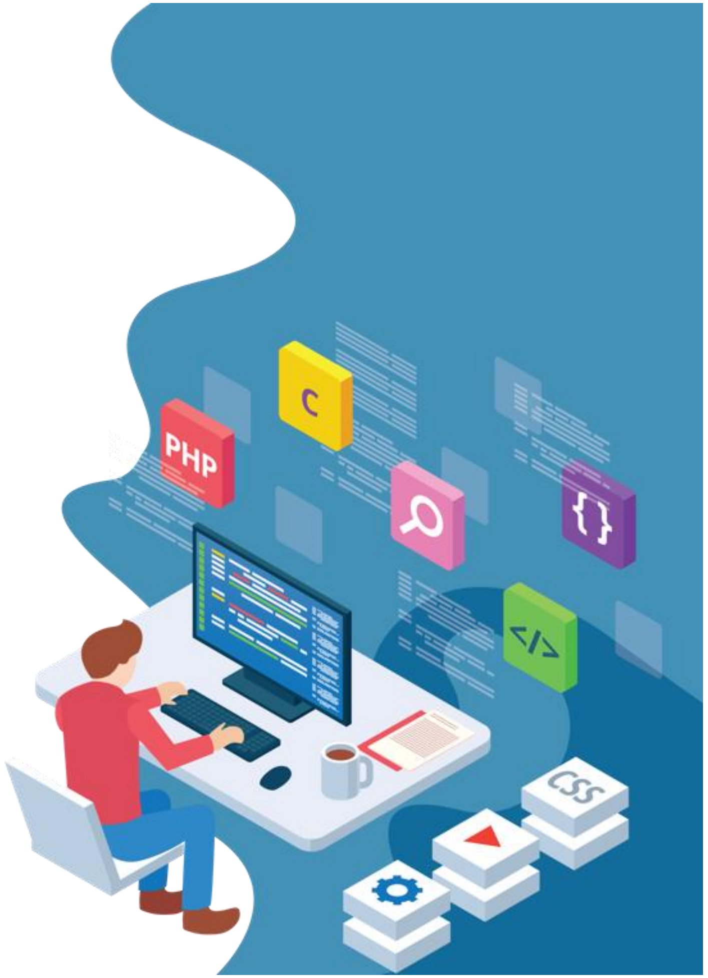
NATIONAL ECONOMICS UNIVERSITY
SCHOOL OF INFORMATION TECHNOLOGY AND DIGITAL ECONOMICS

CHAPTER 6

I/O SYSTEMS

OUTLINE

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Life Cycle of An I/O Request
- Performance



OBJECTIVES

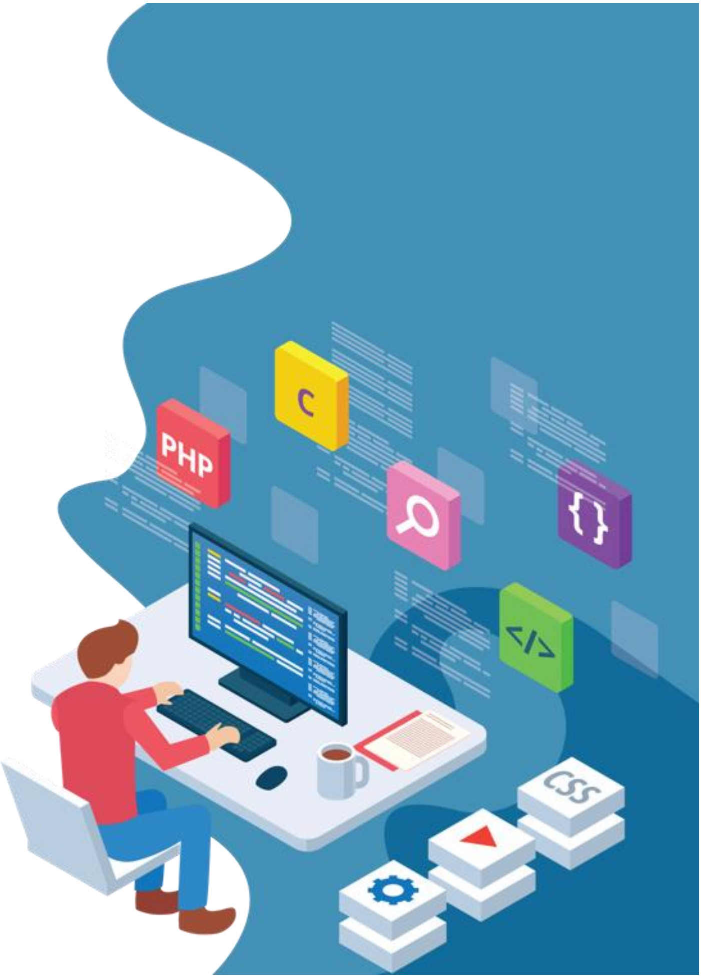
- Explore the structure of an operating system's I/O subsystem
- Discuss the principles and complexities of I/O hardware
- Explain the performance aspects of I/O hardware and software

OVERVIEW

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent
- Ports, busses, device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem

OUTLINE

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Life Cycle of An I/O Request
- Performance



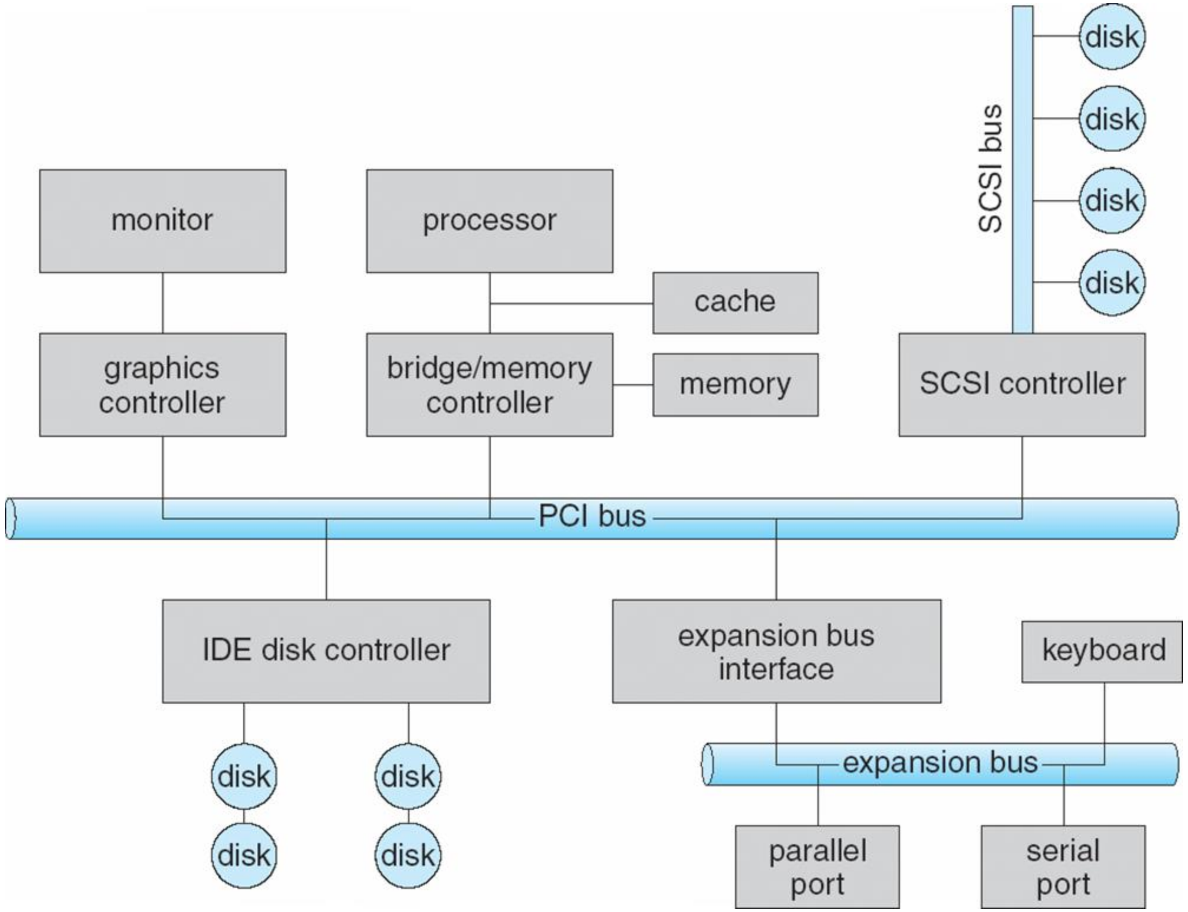
I/O HARDWARE

- Incredible variety of I/O devices
  - Storage
  - Transmission
  - Human-interface
- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus - daisy chain** or shared direct access
    - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
    - **expansion bus** connects relatively slow devices
    - **Serial-attached SCSI** (**SAS**) common disk interface

I/O HARDWARE (CONT.)

- **Controller** (**host adapter**) – electronics that operate port, bus, device
  - Sometimes integrated
  - Sometimes separate circuit board (host adapter)
  - Contains processor, microcode, private memory, bus controller, etc.
    - Some talk to per-device controller with bus controller, microcode, memory, etc.

A TYPICAL PC BUS STRUCTURE



I/O HARDWARE (CONT.)

- **Fibre channel** (**FC**) is complex controller, usually separate circuit board (**host-bus adapter**, **HBA**) plugging into bus
- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer

HOW CAN THE PROCESSOR GIVE COMMANDS AND DATA TO A CONTROLLER TO ACCOMPLISH AN I/O TRANSFER?

- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
- The processor communicates with the controller by reading and writing bit patterns in these registers.
- Two ways:
  - **I/O instructions:** specify the transfer of a byte or word to an I/O port address
  - **Memory-mapped I/O:** Device data and command registers mapped to processor address space

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)

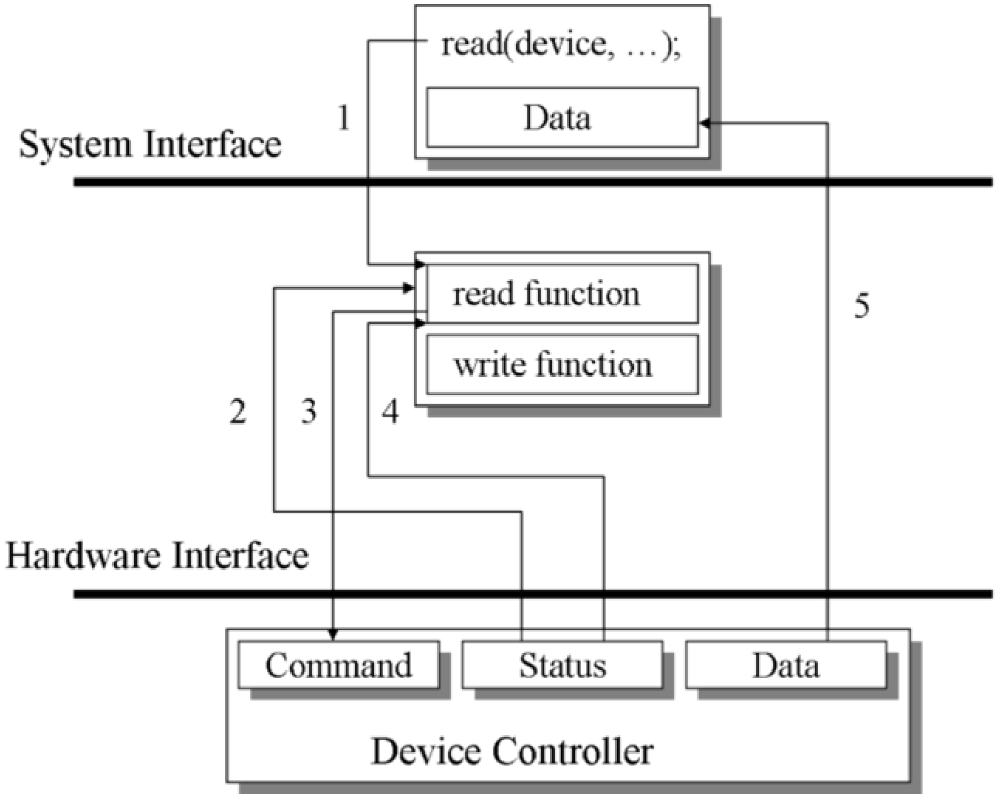
3 TYPES OF I/O:

1. Polling
2. Interrupts
3. Direct Memory Access (**DMA**)

1. POLLING

- For each byte of I/O
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and if write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done
- Step 1 is **busy-wait** cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten / lost

1. POLLING (CONT.)



1. POLLING (CONT.)

- In step 1, the host is busy-waiting or polling: it is in a loop, reading the status register over and over until the busy bit becomes clear.
- If the controller and device are fast, this method is a reasonable one. But if the wait may be long, the host should probably switch to another task.
- For some devices, the host must service the device quickly, or data will be lost.
- For instance, when data are streaming in on a serial port or from a keyboard, the small buffer on the controller will overflow and data will be lost

1. POLLING (CONT.)

- Polling can happen in 3 instruction cycles
- The basic polling operation is efficient
- It becomes inefficient when it is attempted repeatedly yet rarely finds a device ready for service, while other useful CPU processing remains undone
- It may be more efficient to arrange for the hardware controller to notify the CPU when the device becomes ready for service -> **interrupt**

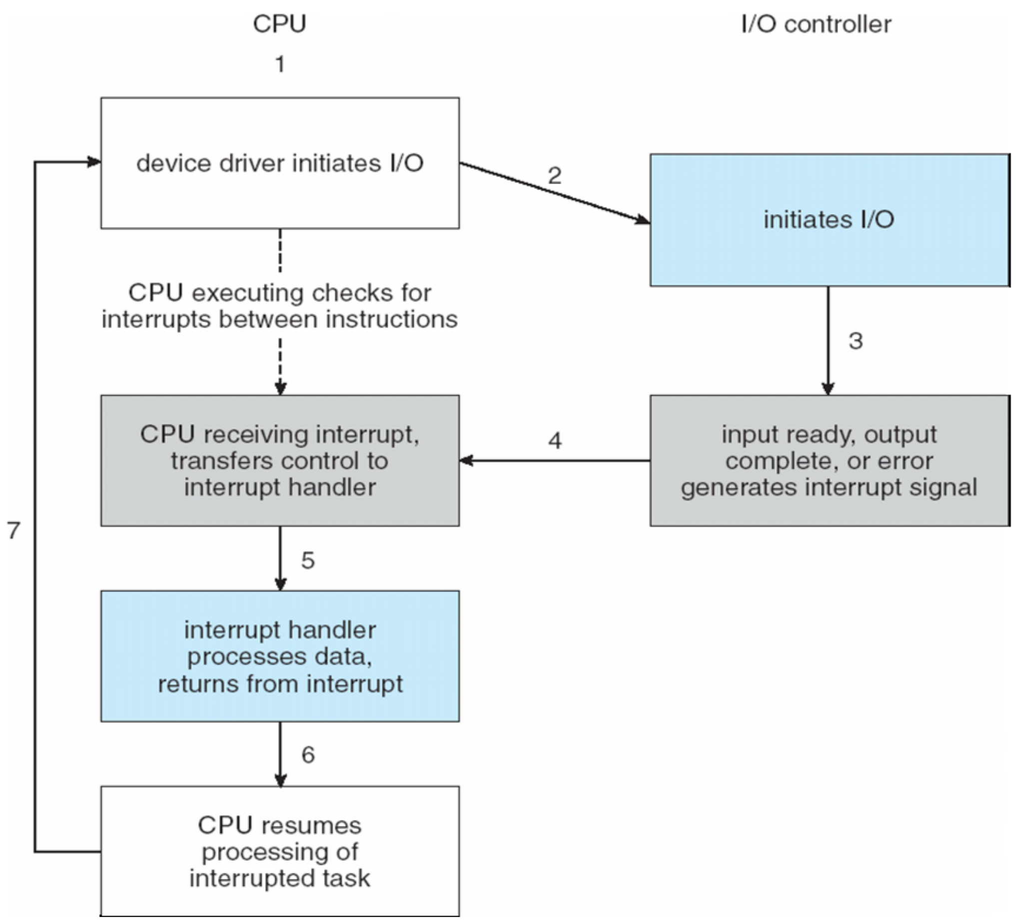
2. INTERRUPTS

- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **nonmaskable**
  - Interrupt chaining if more than one device at same interrupt number

INTERRUPTS (CONT.)

- Interrupt mechanism also used for **exceptions**
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

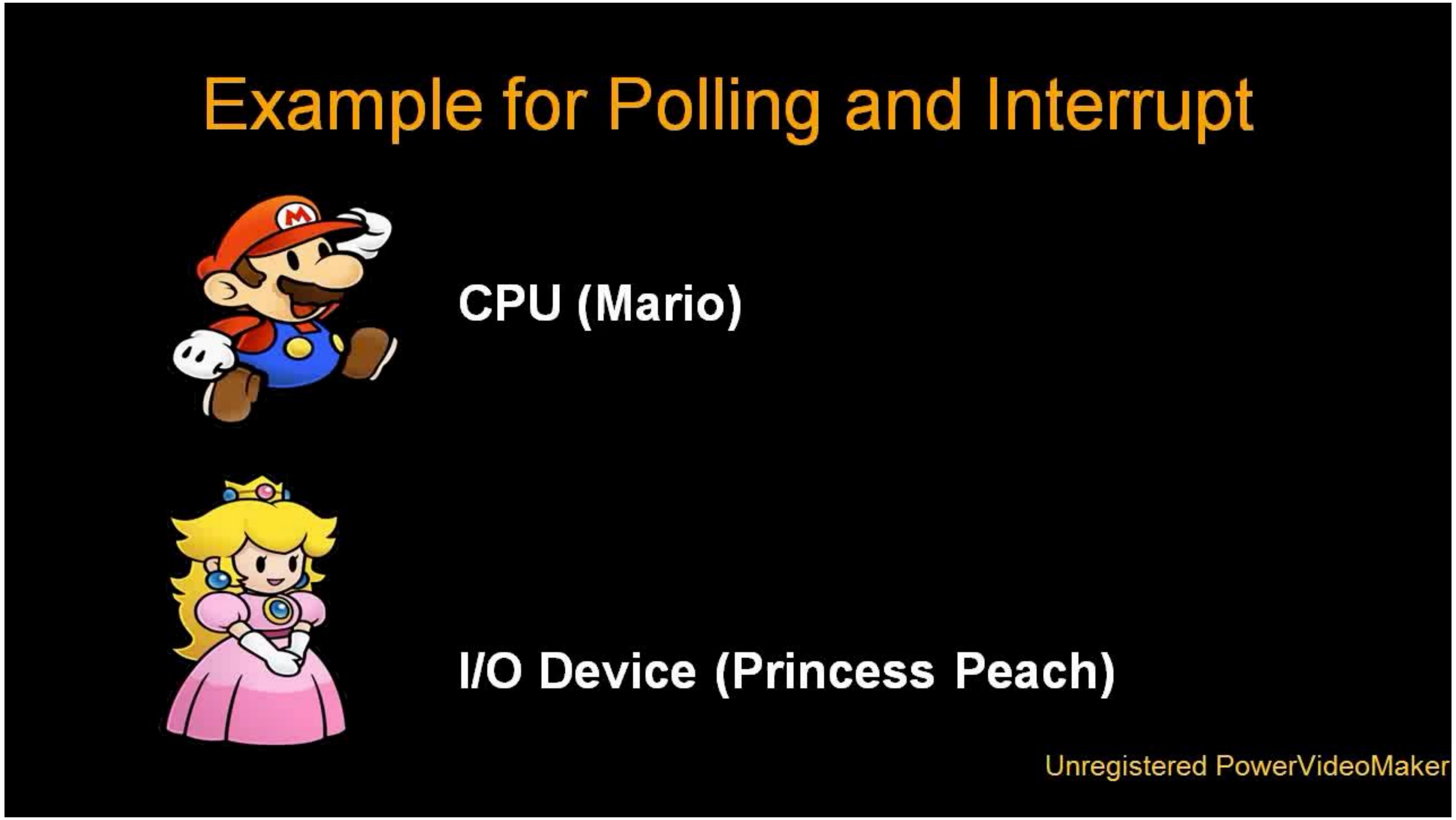
INTERRUPT-DRIVEN I/O CYCLE



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

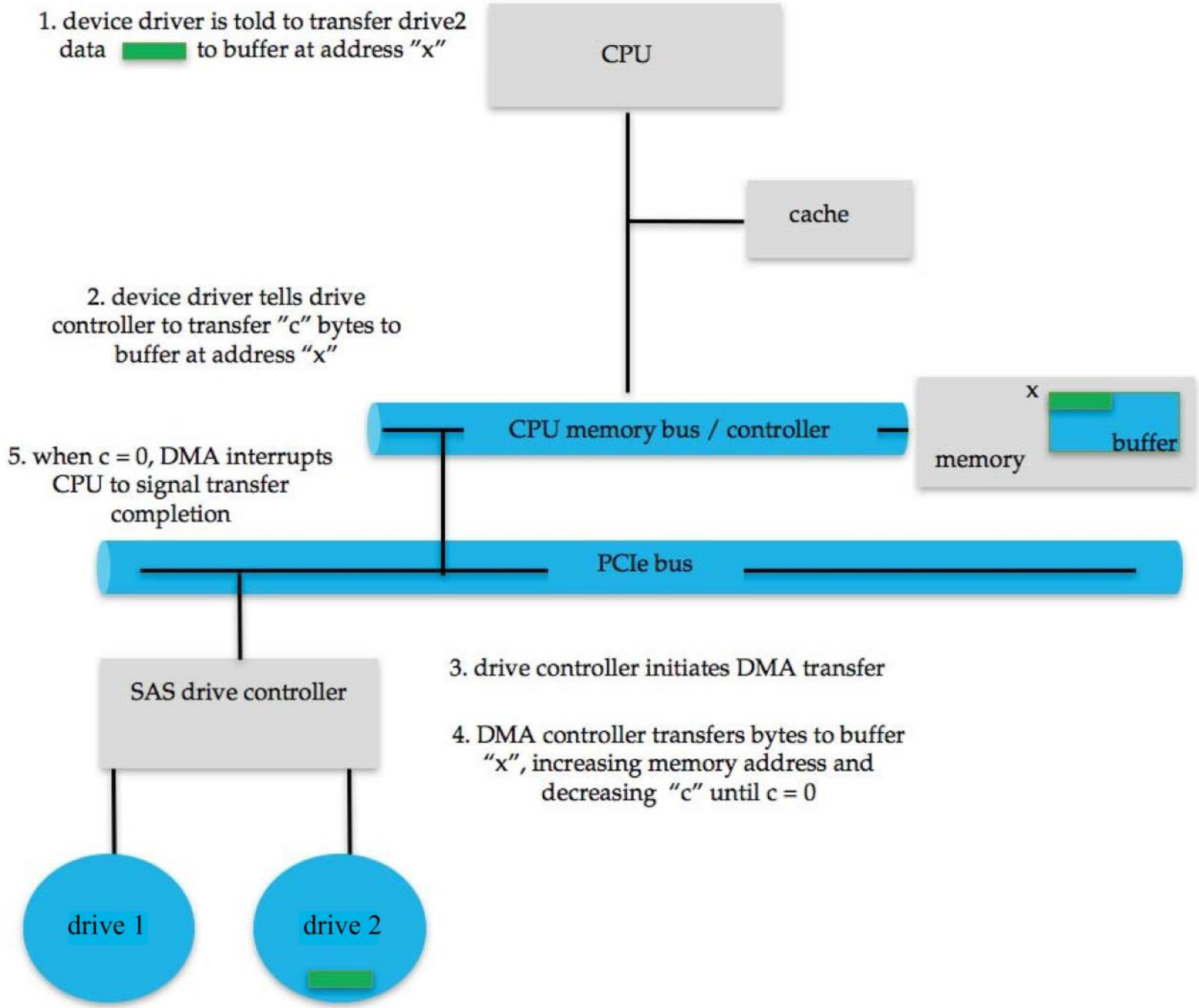
POOLING VS INTERRUPTS



3. DIRECT MEMORY ACCESS (DMA)

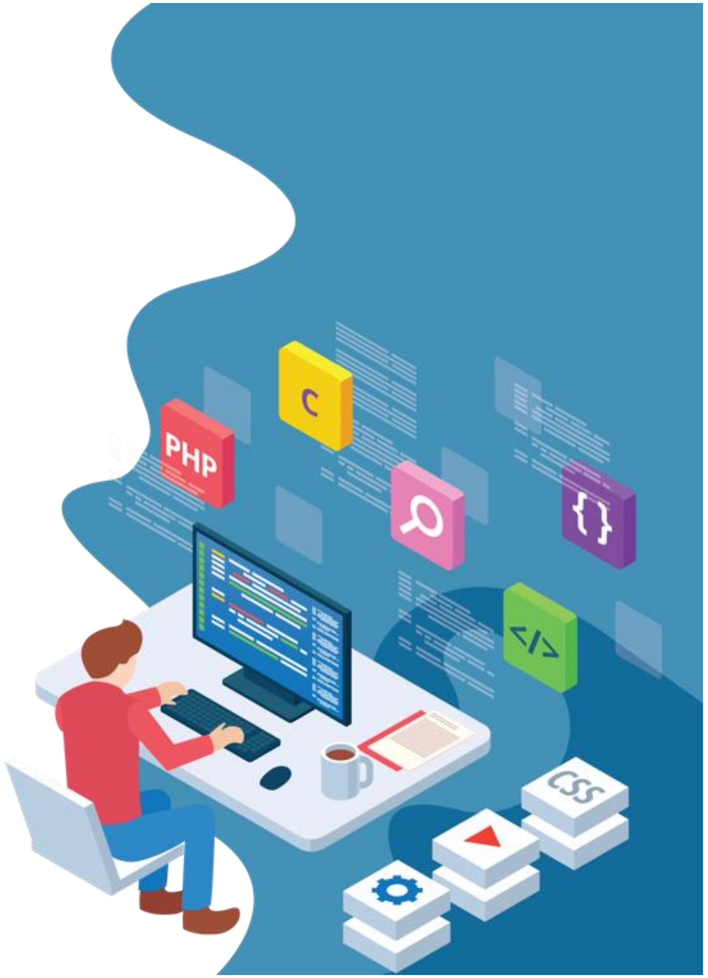
- Used to avoid **programmed I/O (Pooling)** (one byte at a time) for large data movement
- Transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
- Execute DMA command
- Transfers data to memory and removes the DMA-request signal.
- When the entire transfer is finished, the DMA controller interrupts the CPU

SIX STEP PROCESS TO PERFORM DMA TRANSFER



OUTLINE

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Life Cycle of An I/O Request
- Performance



APPLICATION I/O INTERFACE

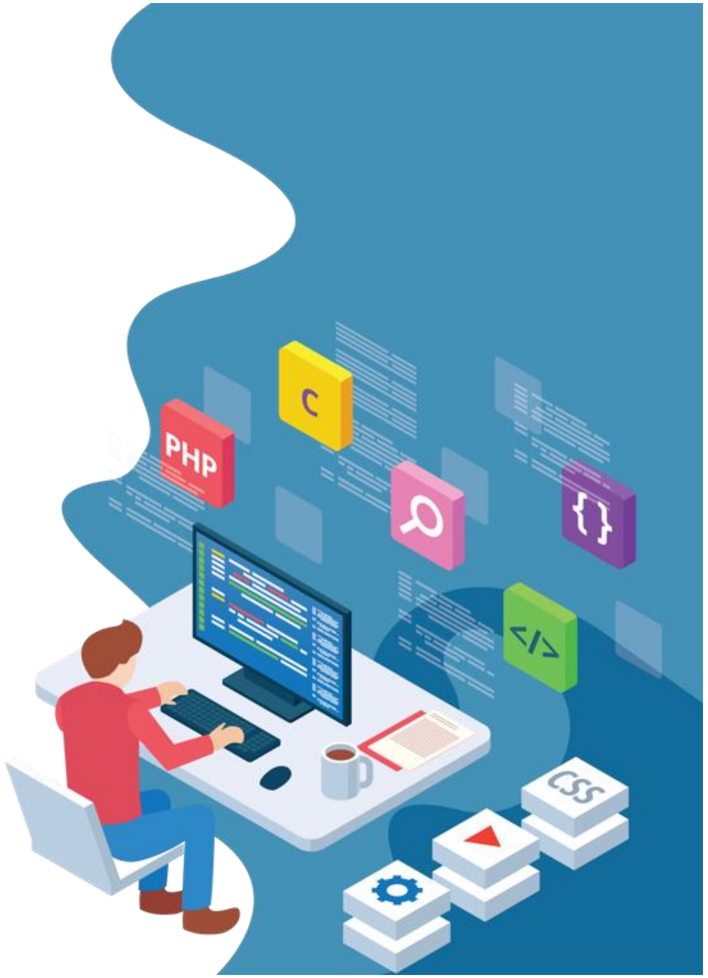
- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions:
  - **Character-stream** or **block**
  - **Sequential** or **random-access**
  - **Synchronous** or **asynchronous** (or both)
  - **Sharable** or **dedicated**
  - **Speed of operation**
  - **read-write**, **read only**, or **write only**

CHARACTERISTICS OF I/O DEVICES

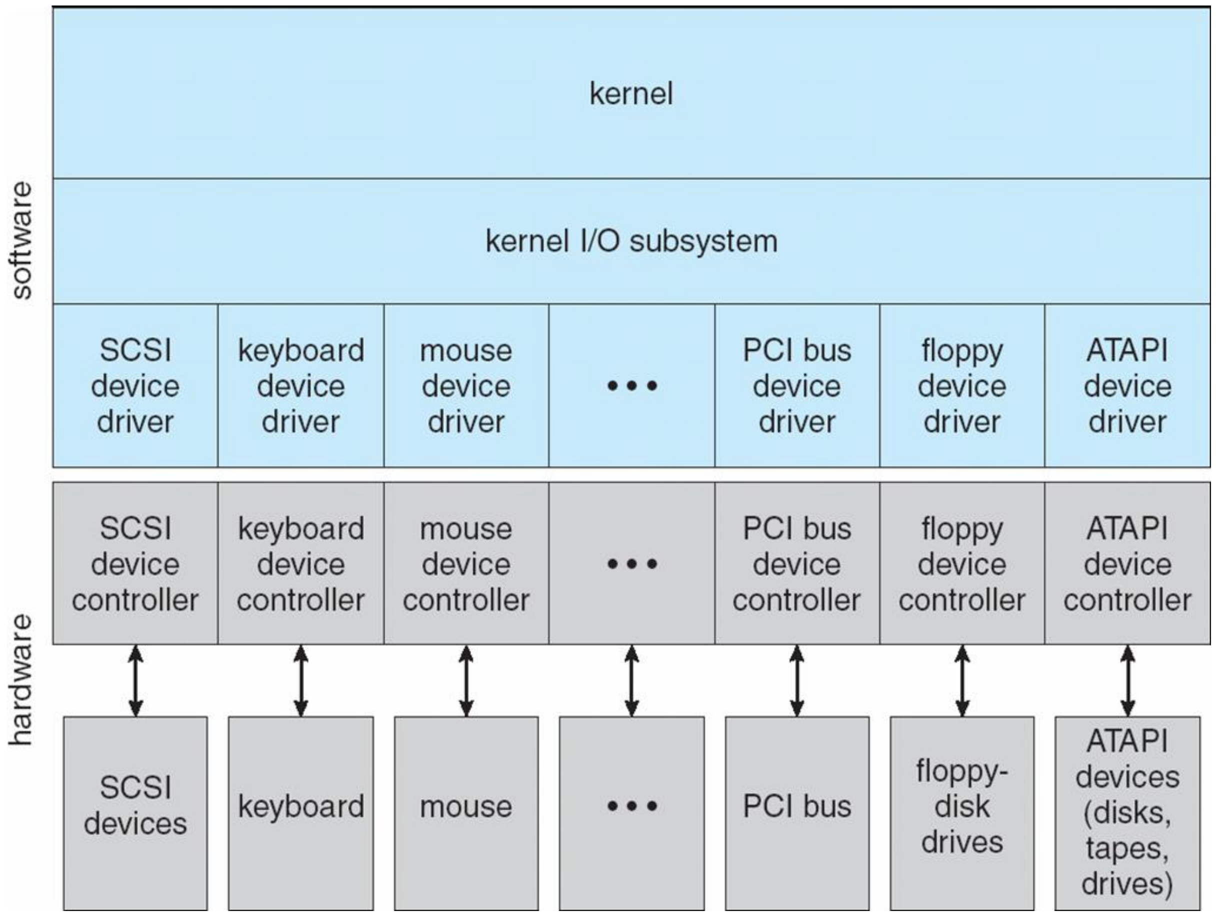
aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction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

OUTLINE

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Life Cycle of An I/O Request
- Performance



A KERNEL I/O STRUCTURE



CHARACTERISTICS OF I/O DEVICES (CONT.)

- Handled by device drivers
- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets

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

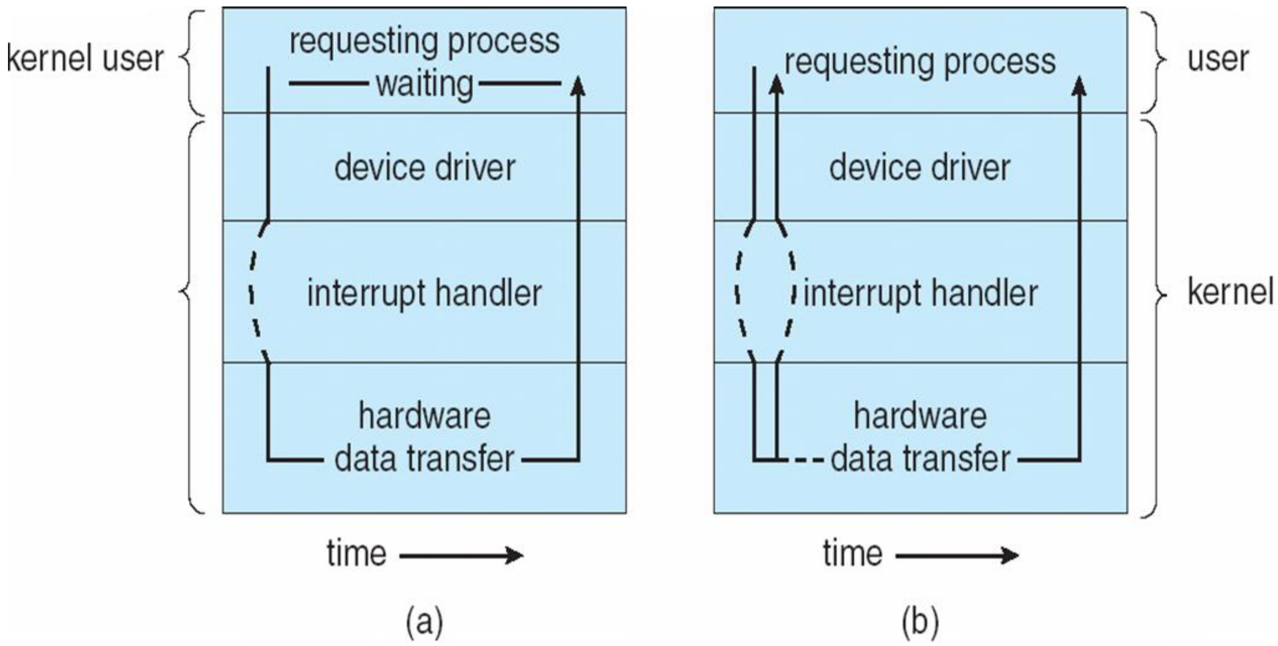
BLOCK AND CHARACTER DEVICES

- Block devices include disk drives
  - Commands include read, write, seek
  - **Raw I/O**, **direct I/O**, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- Character devices include keyboards, mouse, 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
- Most operating systems provide a network I/O interface that is different from the read()–write()–seek() interface used for disk
- One interface available in many operating systems, including UNIX and Windows, is the network **socket** interface.

TWO I/O METHODS



Synchronous

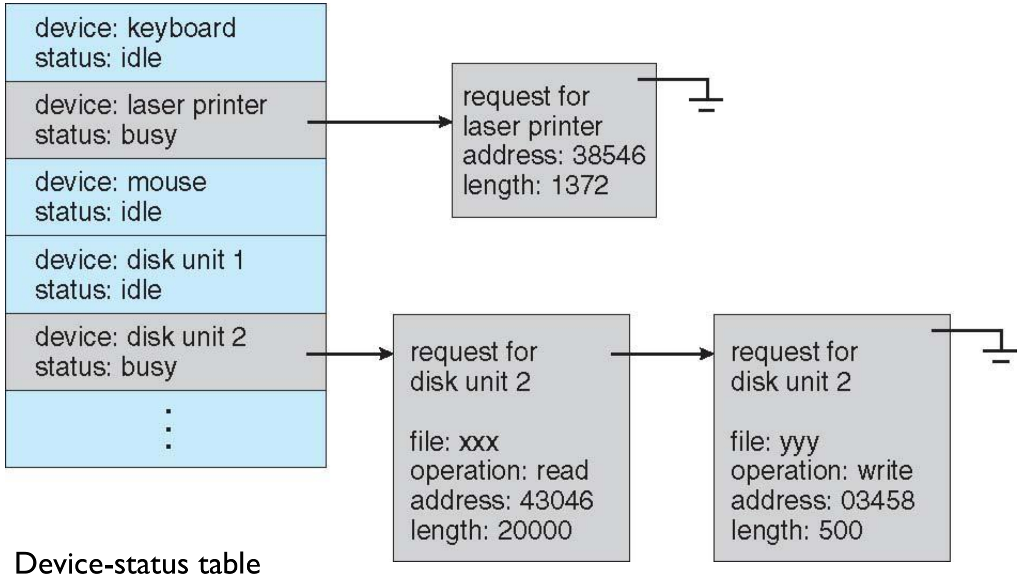
Asynchronous

KERNEL I/O SUBSYSTEM

- Kernels provide many services related to I/O.
  - Scheduling,
  - Buffering,
  - Caching
  - Spooling
  - Device reservation
  - Error handling

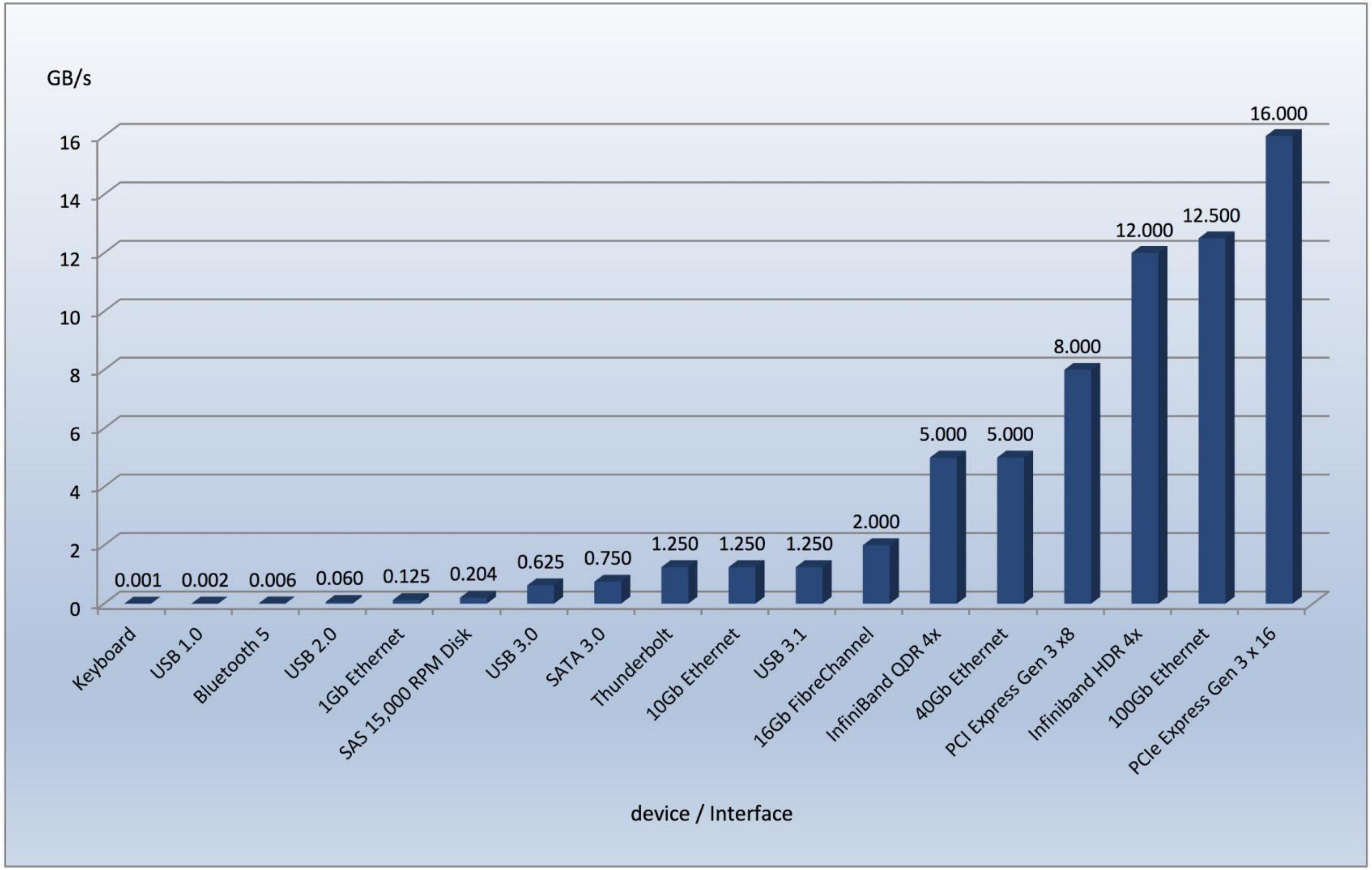
SCHEDULING

- To determine a good order in which to execute them.
- For example: Disk Scheduling
  - FCFS
  - SSTF
  - SCAN
  - C-SCAN
  - LOOK
  - C-LOOK



Device-status table

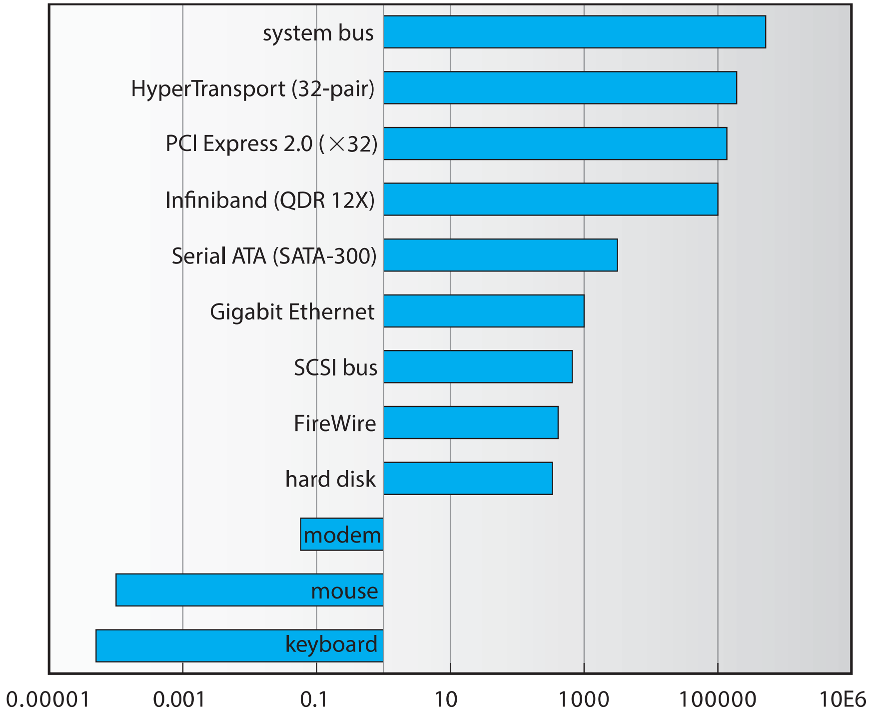
COMMON PC AND DATA-CENTER I/O DEVICES AND INTERFACE SPEEDS



BUFFERING

- Store data in memory while transferring between devices
  - To cope with device speed mismatch
    - **Double buffering** – two copies of the data
  - To cope with device transfer size mismatch
  - To support copy semantics for application I/O

SUN ENTERPRISE 6000 DEVICE-TRANSFER RATES



KERNEL I/O SUBSYSTEM

- **Caching** - faster device holding copy of data
  - Always just a copy
  - Key to performance
- Buffer may hold the only existing copy of a data item, whereas a cache holds a copy on faster storage
  - Sometimes combined with buffering

KERNEL I/O SUBSYSTEM (CONT.)

- **Spooling** - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing:
    - Although a printer can serve only one job at a time
    - Several applications may wish to print their output concurrently,
    - The operating system solves this problem by intercepting all output to the printer.
    - Each application's output is spooled to a separate disk file. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer.
- **Device reservation:** provides exclusive access to a device
    - By enabling a process to allocate an idle device and to deallocate that device when it is no longer needed

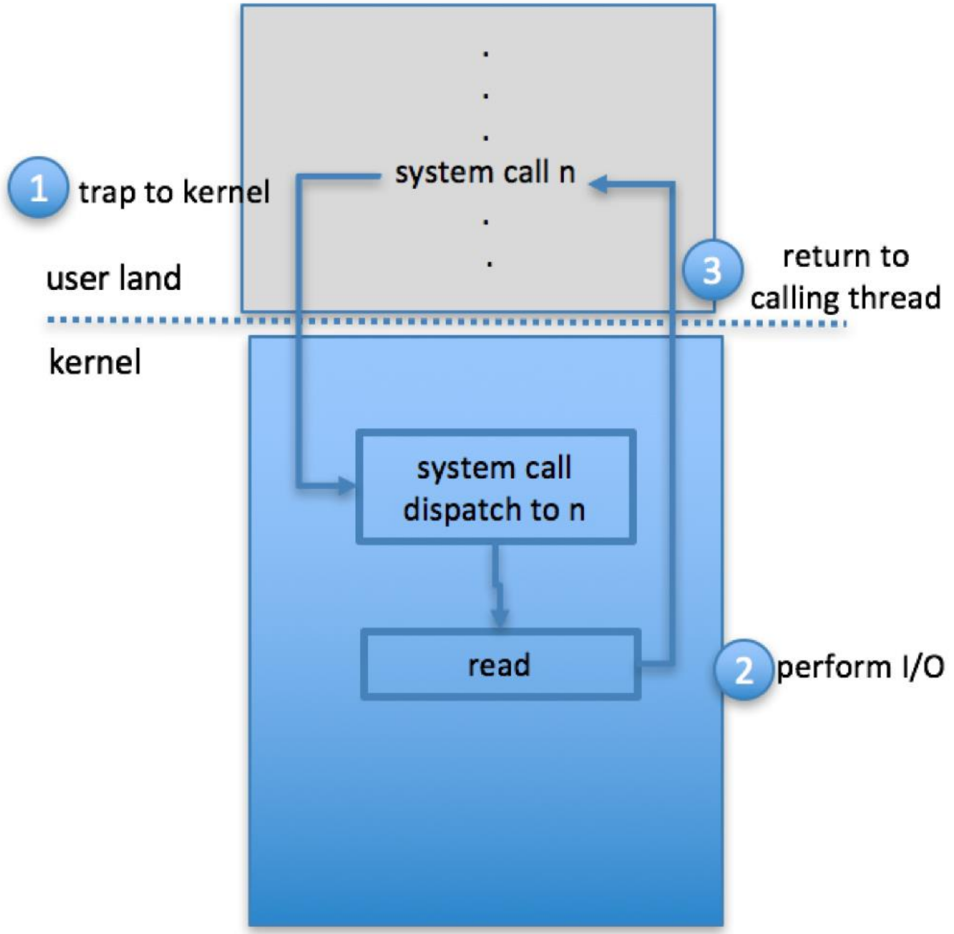
ERROR HANDLING

- Devices and I/O transfers can fail in many ways
  - transient reasons, as when a network becomes overloaded
  - permanent reasons, as when a disk controller becomes defective
- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write, for example
  - Some systems more advanced – Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

I/O PROTECTION

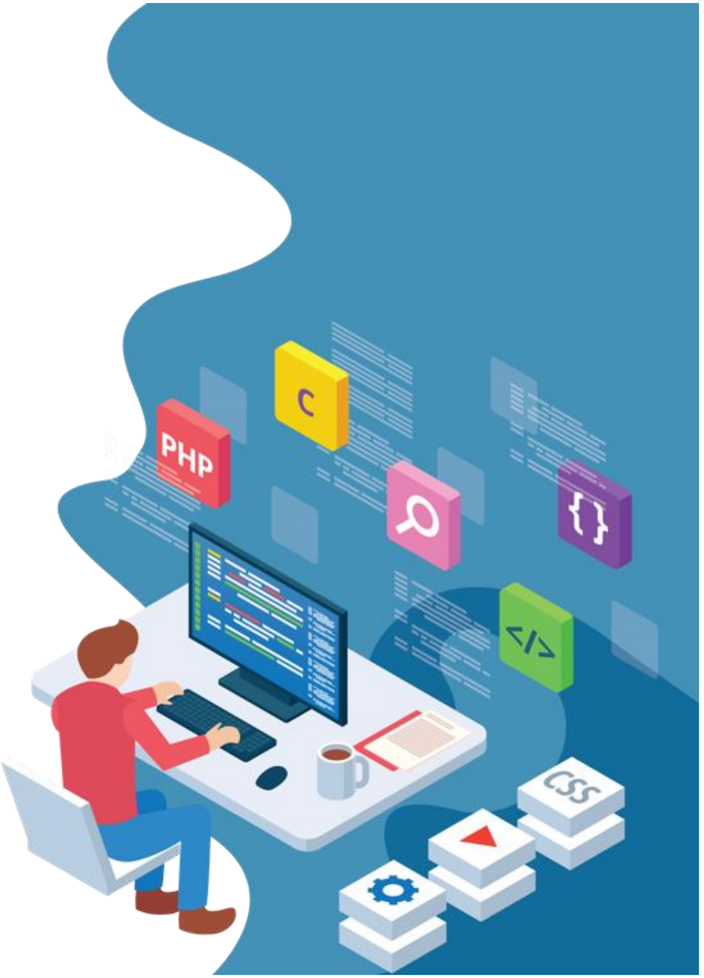
- User process may **accidentally** or **purposefully** attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
  - To do I/O, a user program executes a system call to request that the operating system perform I/O on its behalf.
  - The operating system, executing in monitor mode, checks that the request is valid and, if it is, does the I/O requested. The operating system then returns to the user.

USE OF A SYSTEM CALL TO PERFORM I/O

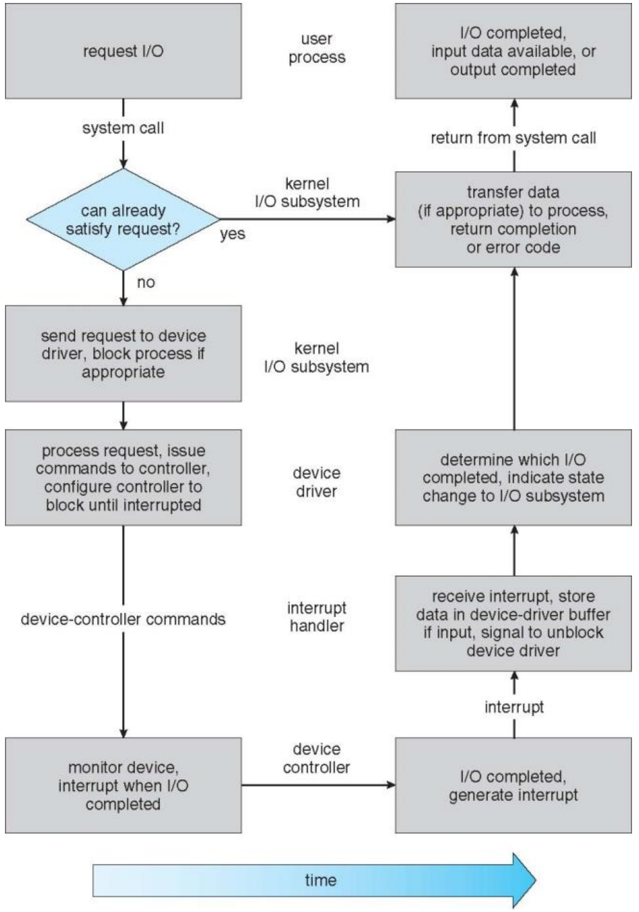


OUTLINE

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Life Cycle of An I/O Request
- Performance



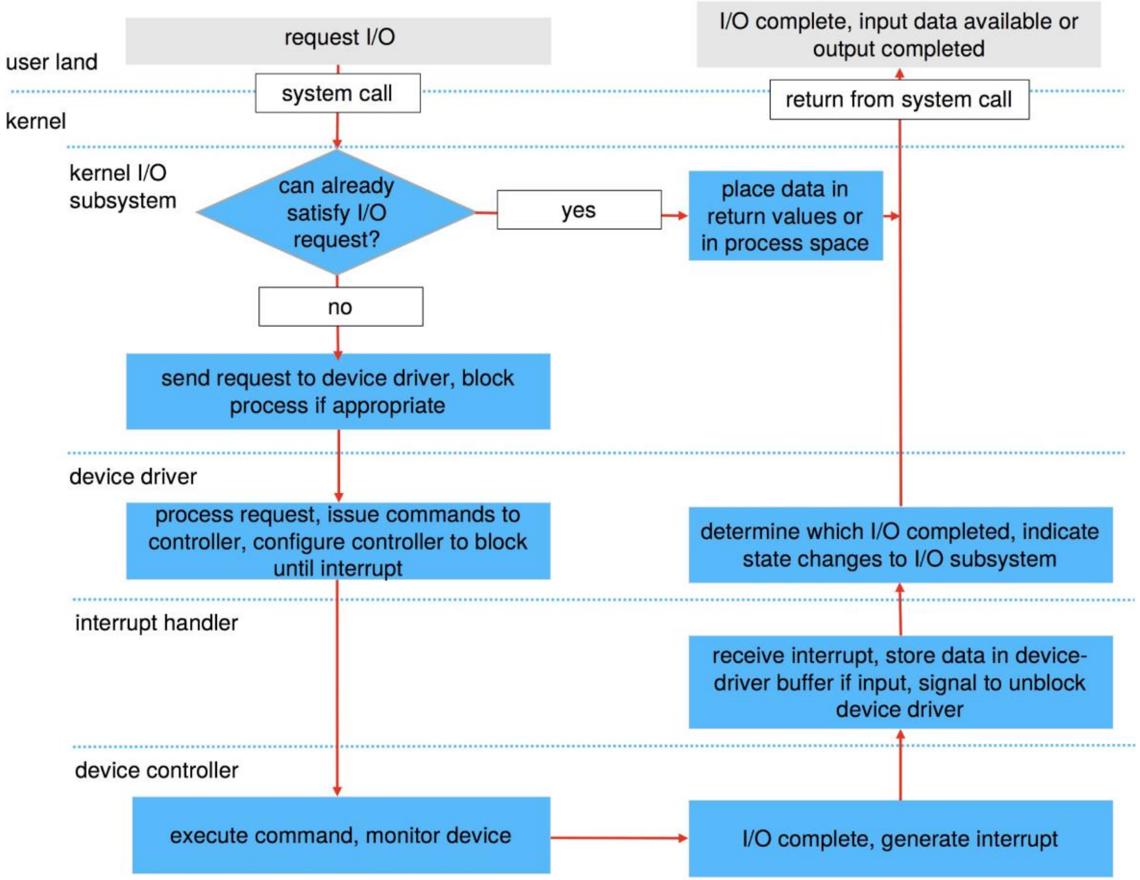
LIFE CYCLE OF AN I/O REQUEST



LIFE CYCLE OF AN I/O REQUEST

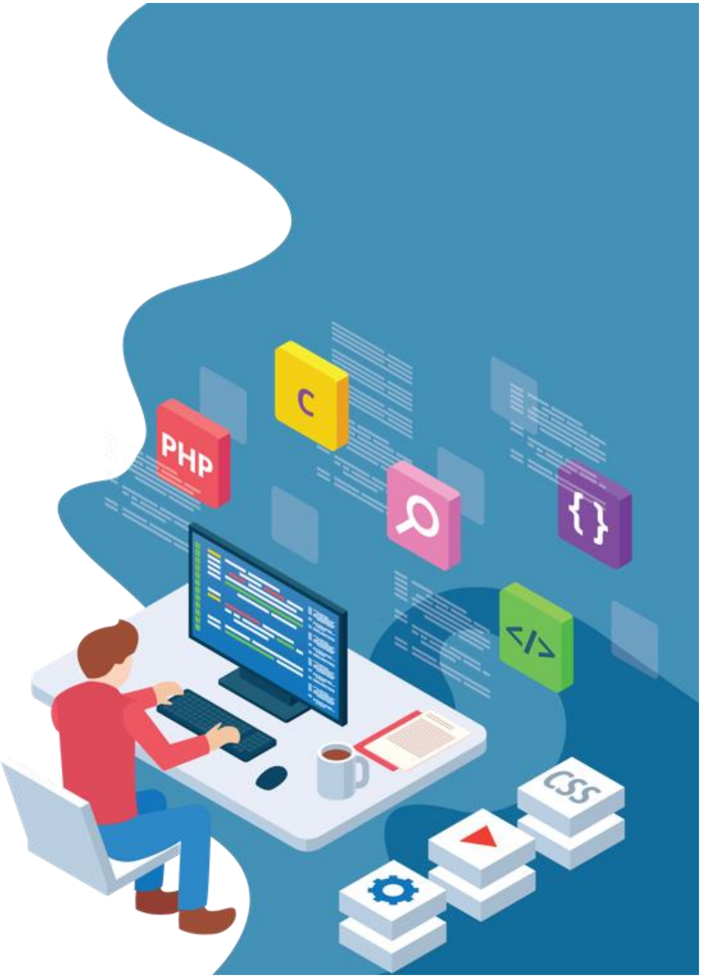
1. A process **issues a read()** system call to a file descriptor of a file that has been opened
2. The system-call code in the kernel checks the parameters for correctness. if the data are already available in the **buffer cache**, the data are returned to the process
3. Otherwise, a physical I/O must be performed. The process is removed from the run queue and is **placed on the IO queue**, and the I/O request is scheduled.
4. The device driver allocates kernel **buffer space** to receive the data. The driver sends commands to the device controller by writing into the device-control registers.
5. The device controller operates the device hardware to perform the **data transfer**.
6. The driver may **poll** for status and data, or it may have **set up a DMA transfer** into kernel memory. An interrupt may be generated when the transfer completes.
7. The **interrupt handler** receives the interrupt, stores necessary data, signals the device
8. The **device driver receives the signal**, determines which I/O request has completed, the request's status, and signals that the request has been complede
9. The kernel transfers data or **return codes to the address space** of the requesting process and moves the process from the wait queue back to the **ready queue**.
10. The process **resumes** execution at the completion of the system call.

LIFE CYCLE OF AN I/O REQUEST (CONT.)



OUTLINE

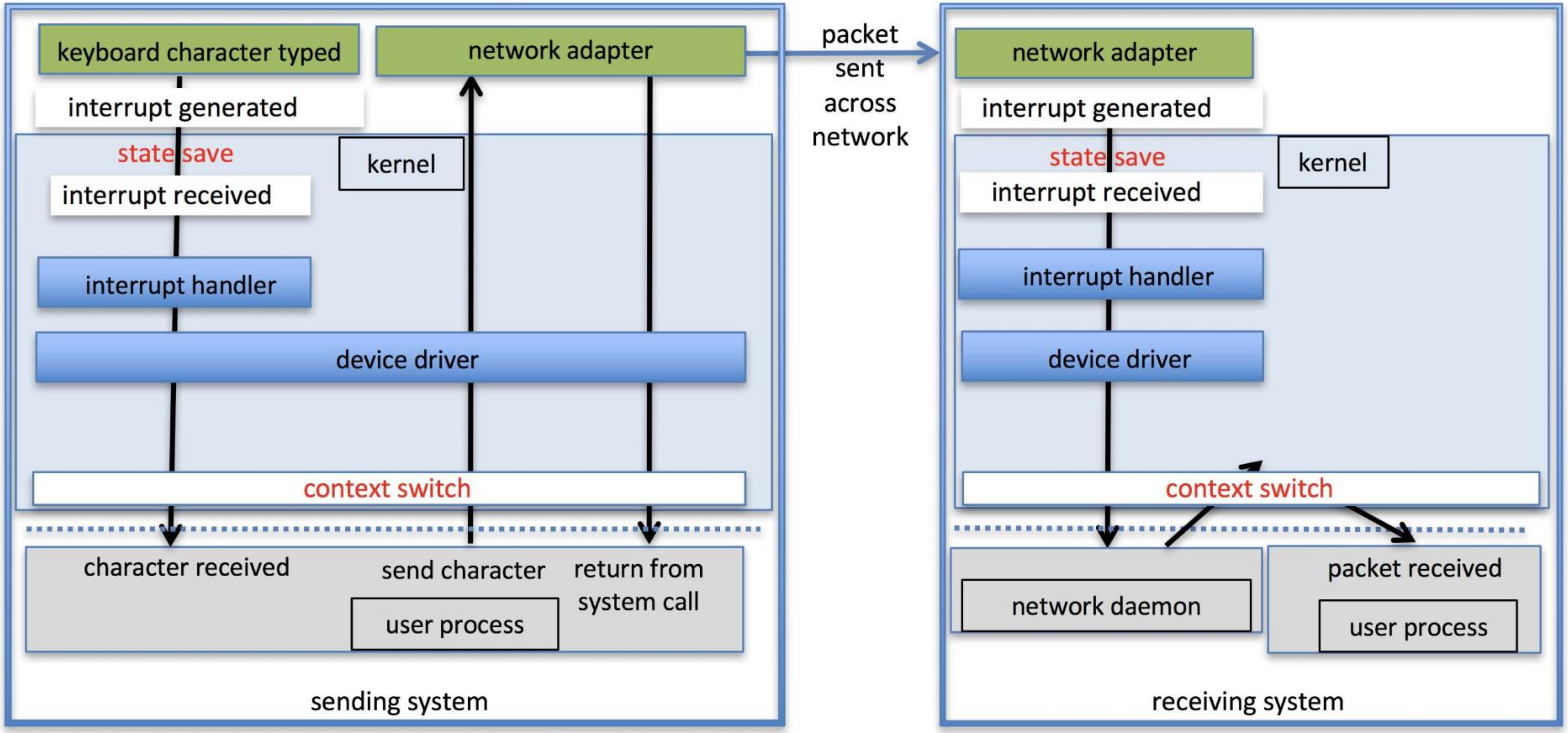
- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Life Cycle of An I/O Request
- Performance



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

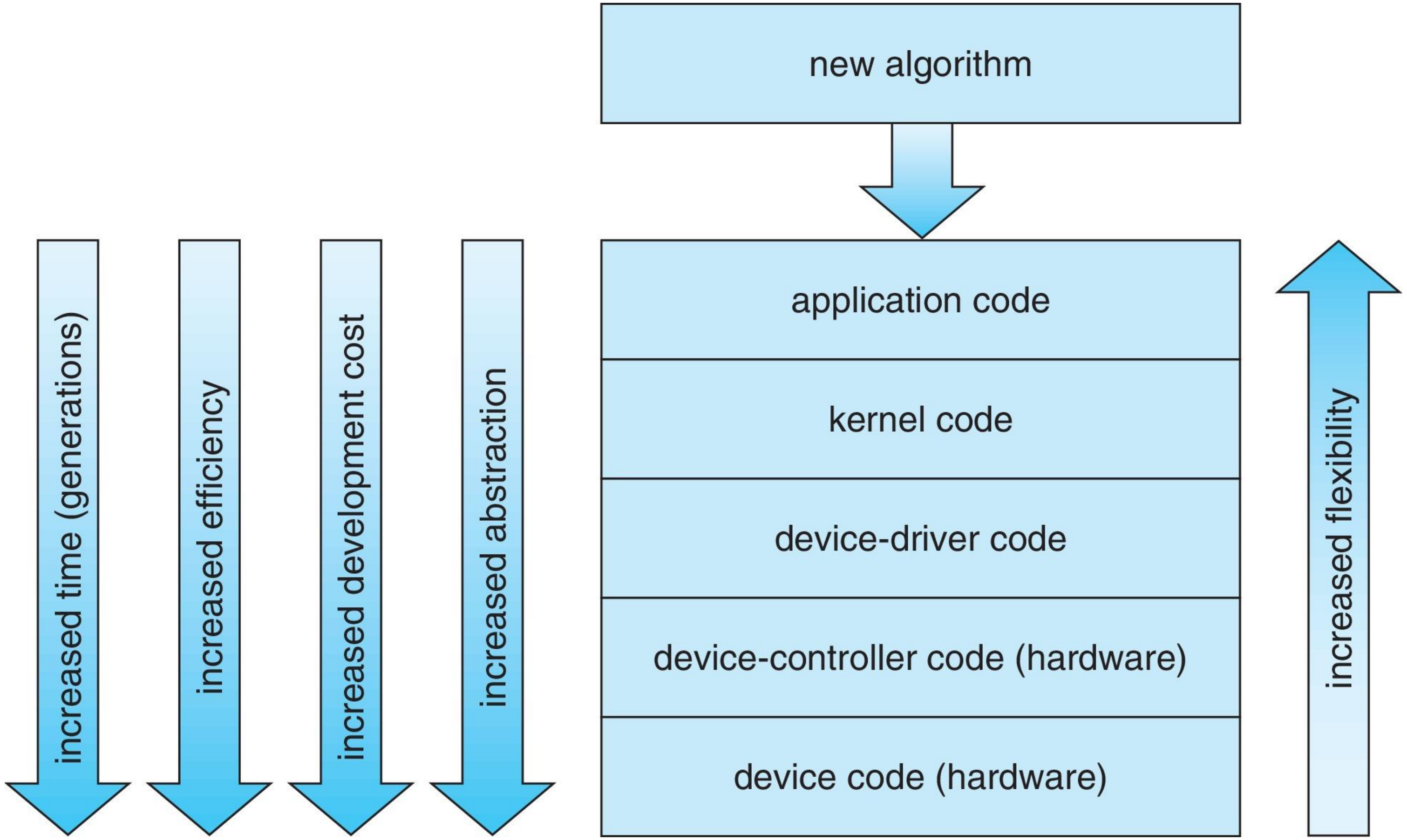
INTERCOMPUTER COMMUNICATIONS



IMPROVING PERFORMANCE

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput
- Move user-mode processes / daemons to kernel threads

DEVICE-FUNCTIONALITY PROGRESSION



I/O PERFORMANCE OF STORAGE (AND NETWORK LATENCY)