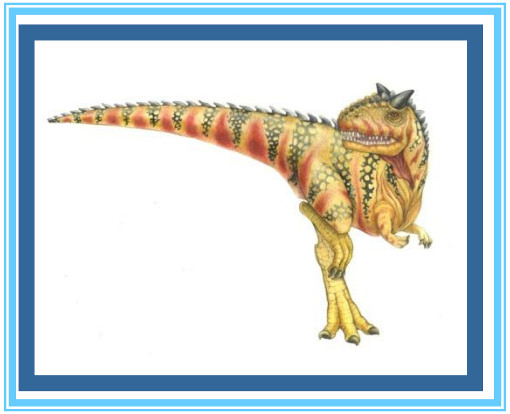
Chapter 13: I/O Systems



Chapter 13: I/O Systems

- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Streams
- Performance

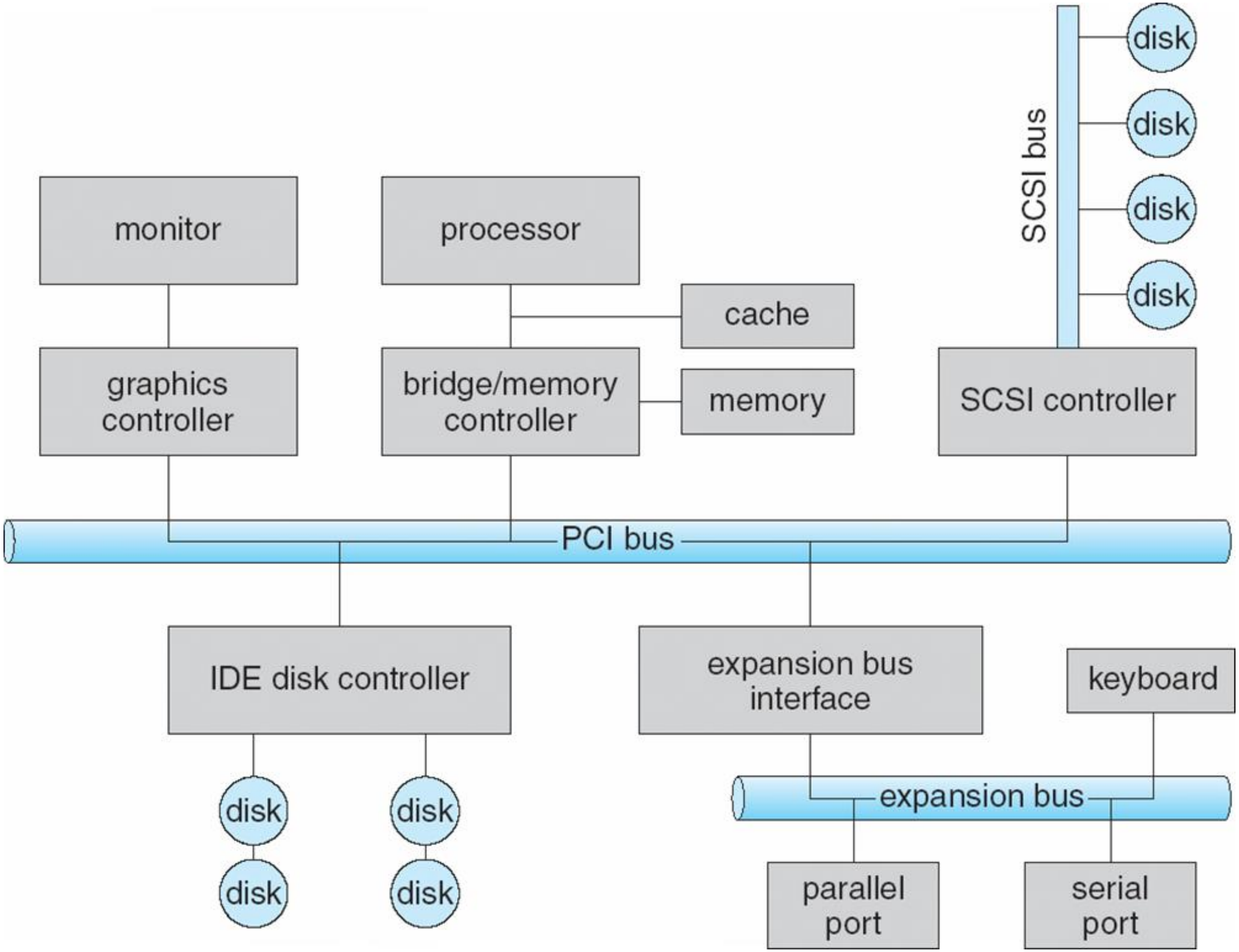
Objectives

- 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)

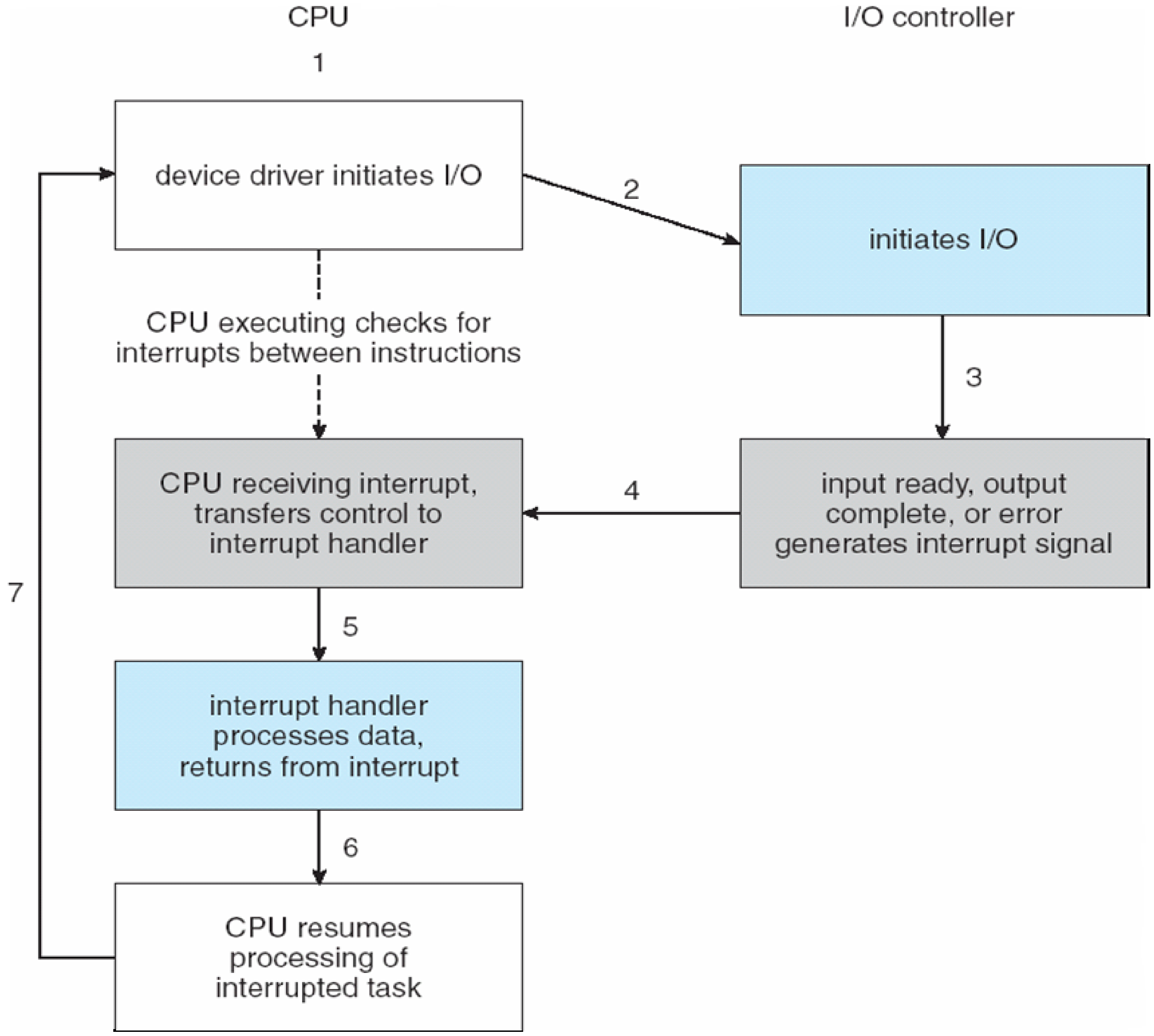
Polling

- Determines state of device
  - command-ready
  - busy
  - Error
- **Busy-wait** cycle to wait for I/O from device

Interrupts

- CPU **Interrupt-request line** triggered by I/O device
- **Interrupt handler** receives interrupts
- **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some **nonmaskable**
- Interrupt mechanism also used for exceptions

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

Direct Memory Access

Used to avoid **programmed I/O** for large data movement

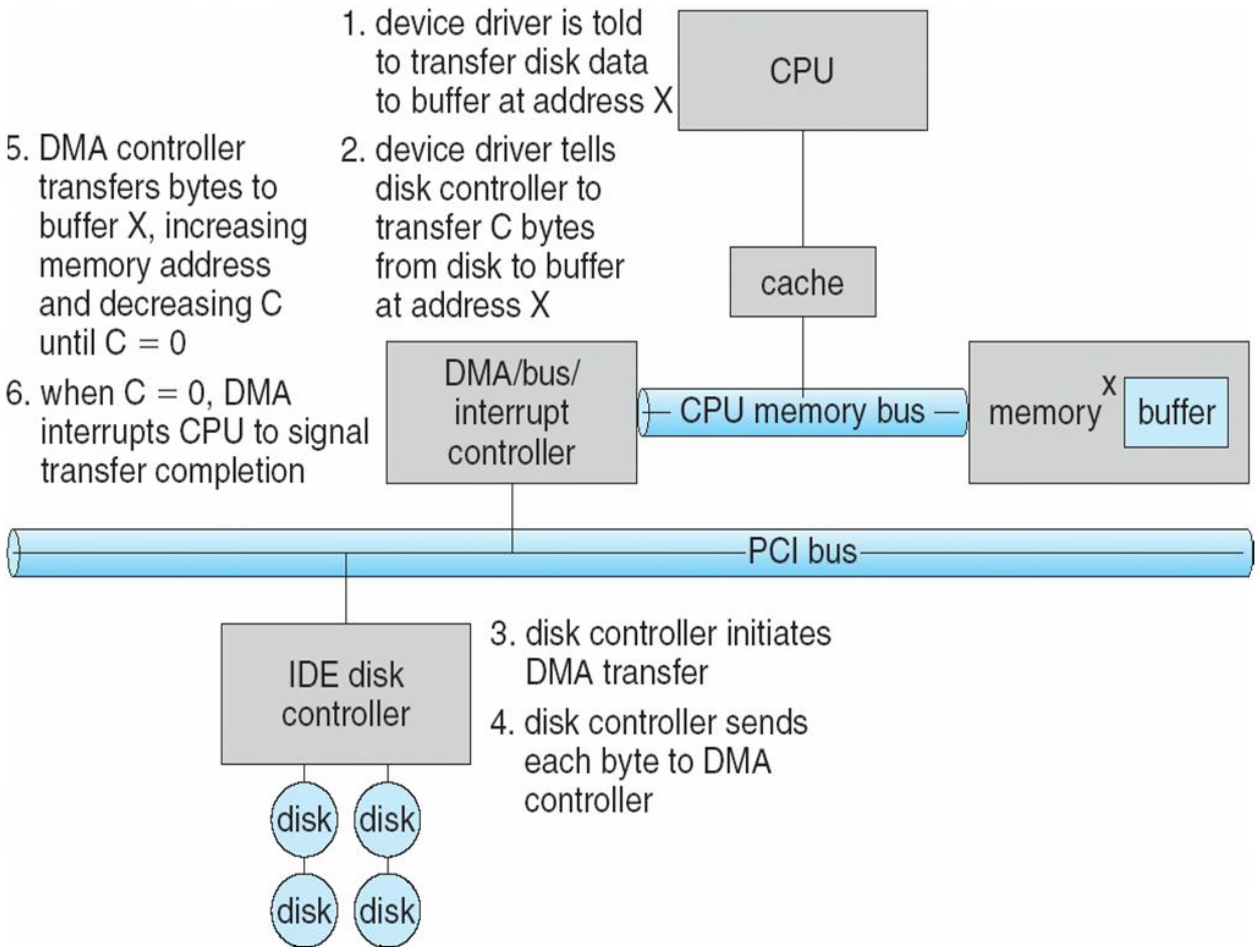
```c
/*
* example.c: very simple example of port I/O
*
* This code does nothing useful, just a port write, a pause,
* and a port read. Compile with `gcc -O2 -o example example.c',
* and run as root with `./example'.
*/
#include <stdio.h>
#include <unistd.h>
#include <sys/io.h>
#define BASEPORT 0x378 /* lp1 */
int main()
{
    /* Get access to the ports */
    if (ioperm(BASEPORT, 3, 1)) {perror("ioperm"); return 1;}
    /* Set the data signals (D0-7) of the port to all low (0) */
    outb(0, BASEPORT);
    /* Sleep for a while (100 ms) */
    usleep(100000);
    /* Read from the status port (BASE+1) and display the result */
    printf("status: %d\n", inb(BASEPORT + 1));
    /* We don't need the ports anymore */
    if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); return 1;}
    return 0;
}
/* end of example.c */
```

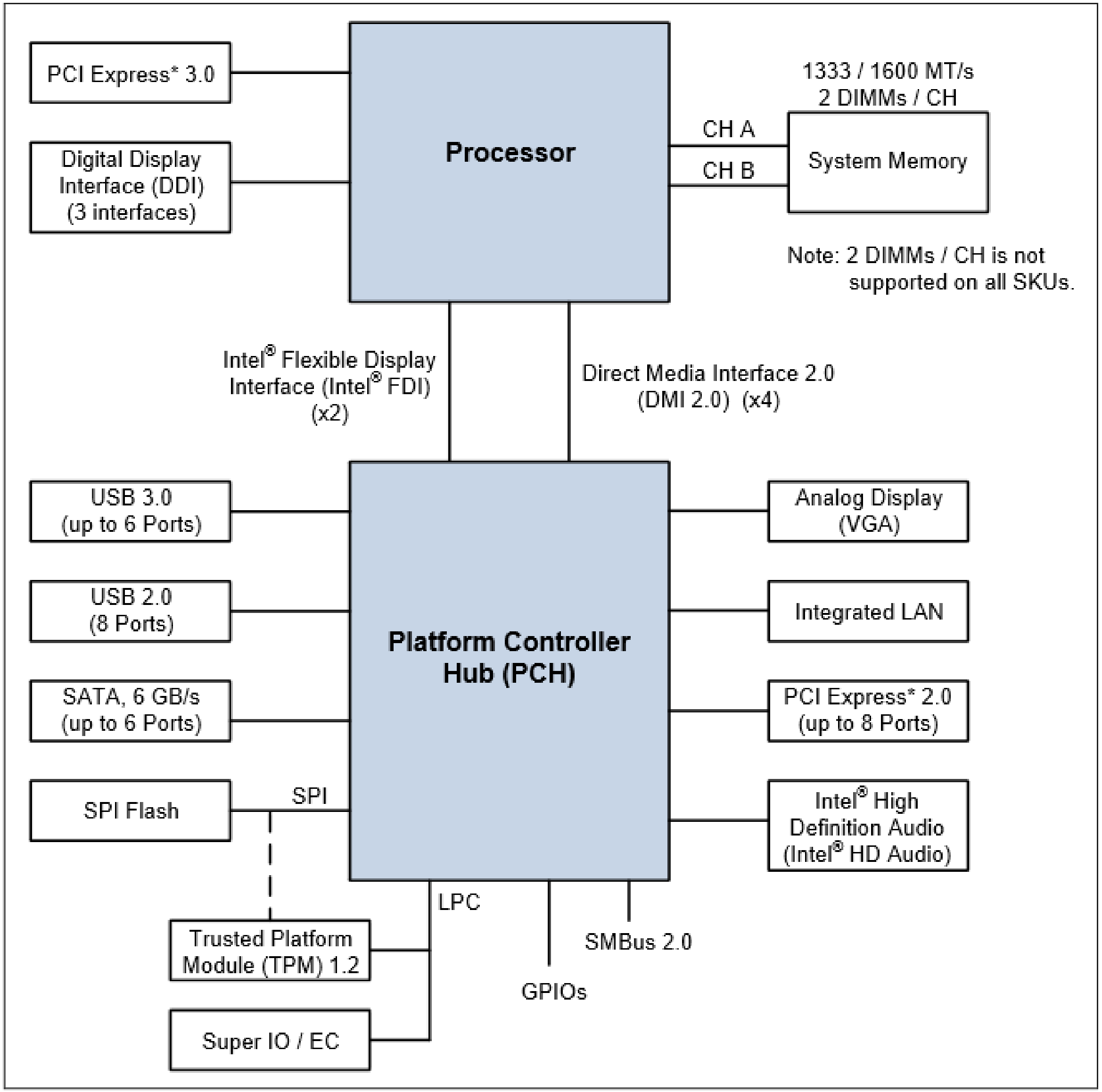
Requires **DMA** controller

Bypasses CPU to transfer data directly between I/O device and memory hank@hank-VirtualBox: ~

```
hank@hank-VirtualBox:~$ sudo cat /proc/ioports
[sudo] password for hank:
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-0060 : keyboard
0064-0064 : keyboard
0070-0071 : rtc_cmos
  0070-0071 : rtc0
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : 0000:00:01.1
  0170-0177 : ata_piix
01f0-01f7 : 0000:00:01.1
  01f0-01f7 : ata_piix
0376-0376 : 0000:00:01.1
  0376-0376 : ata_piix
03c0-03df : vesafb
03f6-03f6 : 0000:00:01.1
  03f6-03f6 : ata_piix
0cf8-0cff : PCI conf1
4000-4003 : ACPI PM1a_EVT_BLK
4004-4005 : ACPI PM1a_CNT_BLK
4008-400b : ACPI PM_TMR
4020-4021 : ACPI GPE0_BLK
d000-d00f : 0000:00:01.1
  d000-d00f : ata_piix
d010-d017 : 0000:00:03.0
  d010-d017 : e1000
d020-d03f : 0000:00:04.0
d100-d1ff : 0000:00:05.0
  d100-d1ff : Intel 82801AA-ICH
d200-d23f : 0000:00:05.0
  d200-d23f : Intel 82801AA-ICH
d240-d247 : 0000:00:0d.0
  d240-d247 : ahci
d250-d257 : 0000:00:0d.0
  d250-d257 : ahci
d260-d26f : 0000:00:0d.0
  d260-d26f : ahci
hank@hank-VirtualBox:~$
```

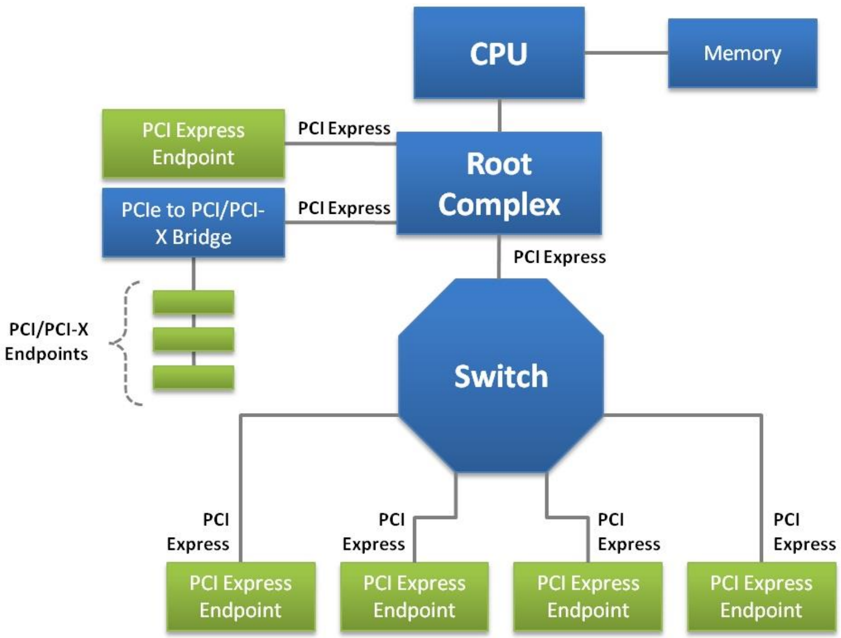
Six Step Process to Perform DMA Transfer




PCI Express* 3.0
Processor
1333 / 1600 MT/s
2 DIMMs / CH
CH A
CH B
System Memory
Digital Display
Interface (DDI)
(3 interfaces)
Note: 2 DIMMs / CH is not
supported on all SKUs.
Intel® Flexible Display
Interface (Intel® FDI)
(x2)
Direct Media Interface 2.0
(DMI 2.0) (x4)
USB 3.0
(up to 6 Ports)
Analog Display
(VGA)
USB 2.0
(8 Ports)
Integrated LAN
Platform Controller
Hub (PCH)
SATA, 6 GB/s
(up to 6 Ports)
PCI Express* 2.0
(up to 8 Ports)
SPI Flash
SPI
Intel® High
Definition Audio
(Intel® HD Audio)
LPC
SMBus 2.0
Trusted Platform
Module (TPM) 1.2
GPIOs
Super IO / EC

PCI Express

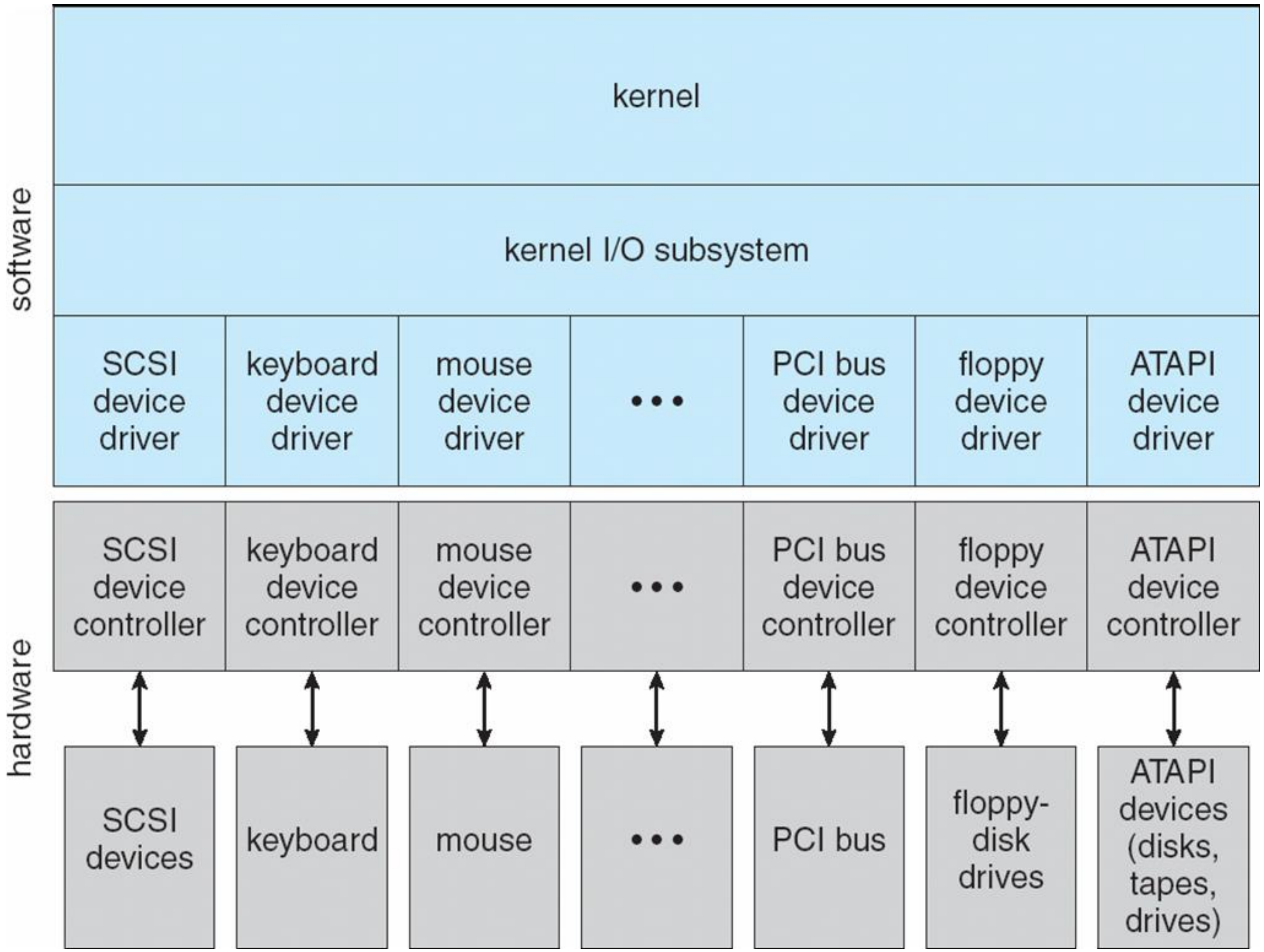
http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1



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

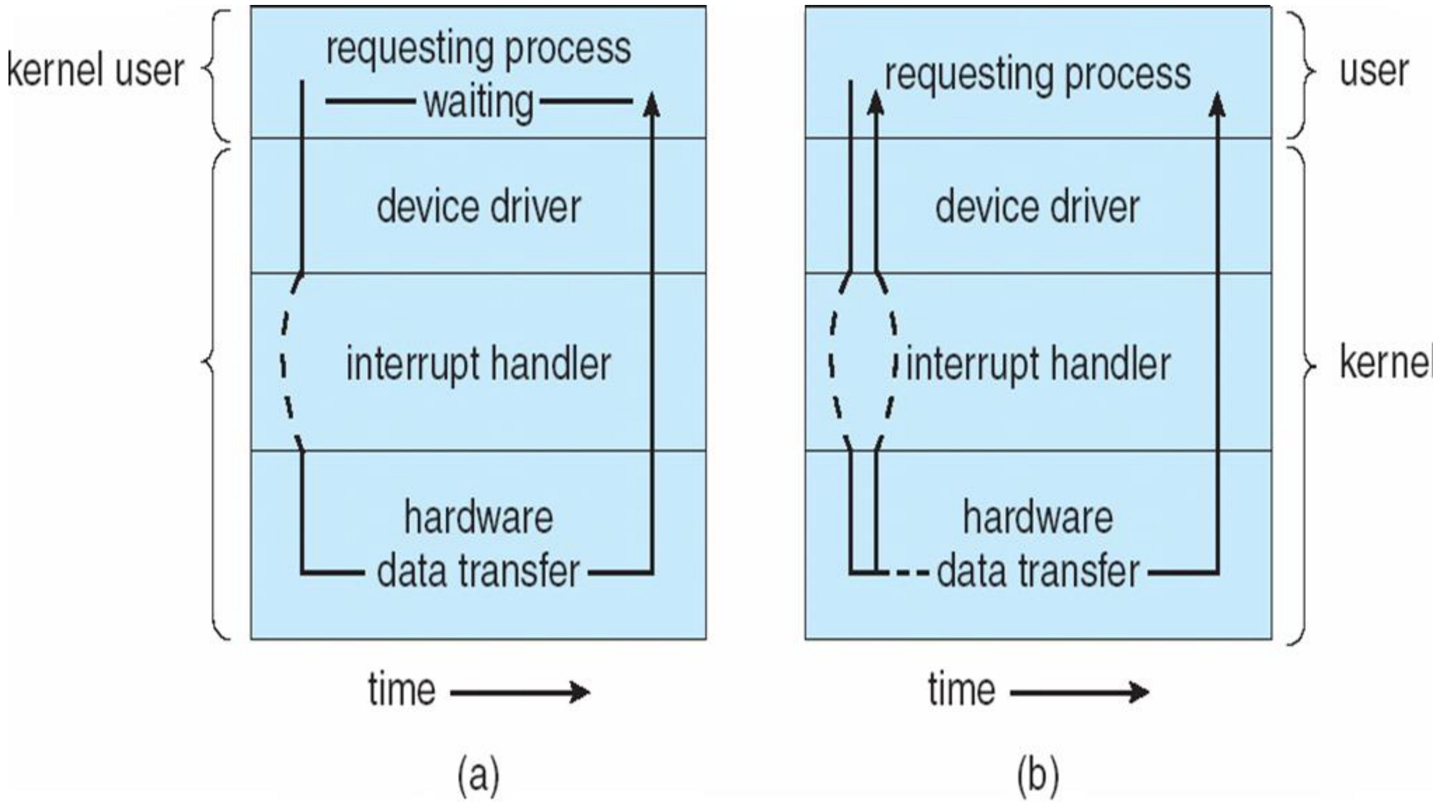
Programmable interval timer used for timings, periodic interrupts

`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

Two I/O Methods



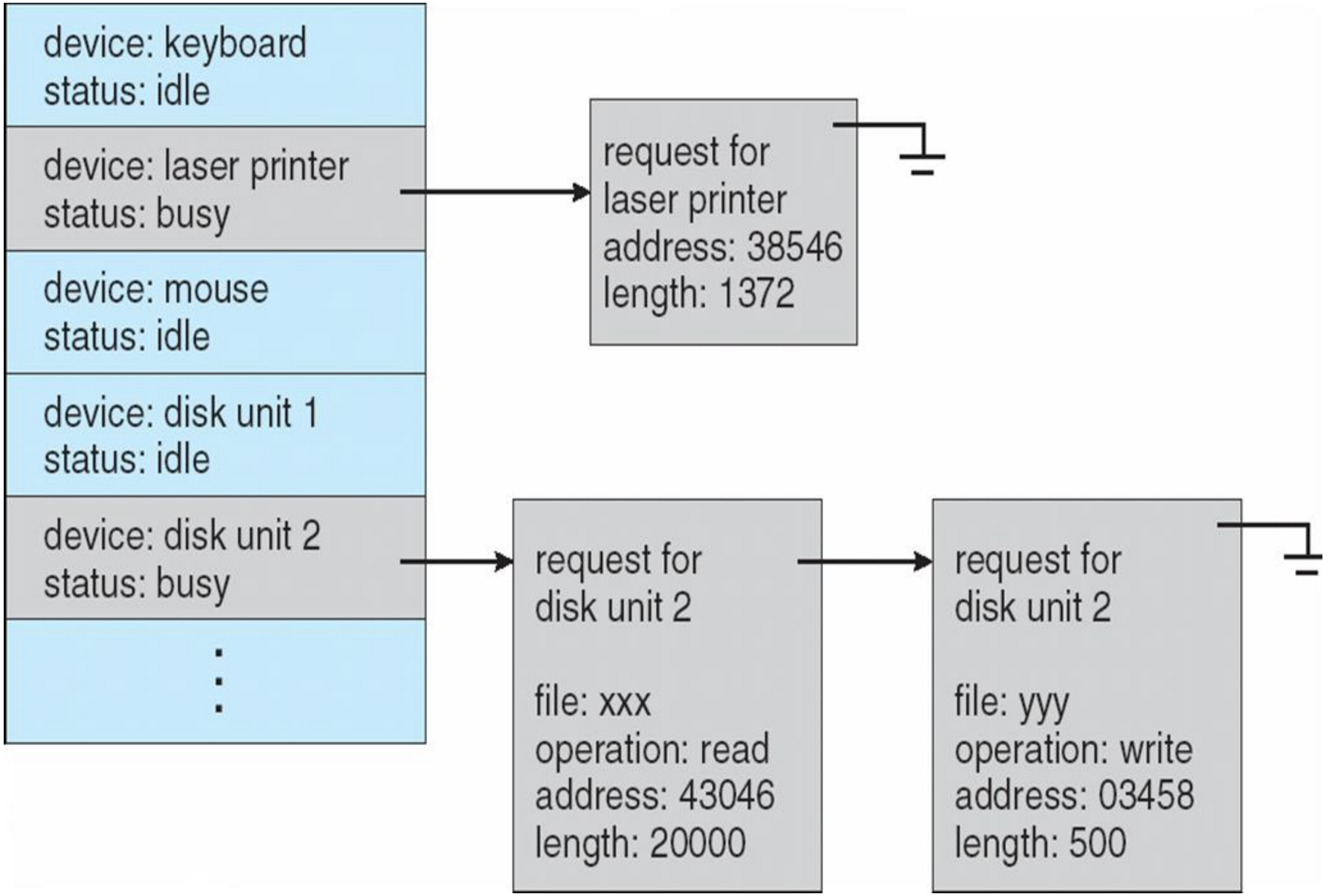
Synchronous

Asynchronous

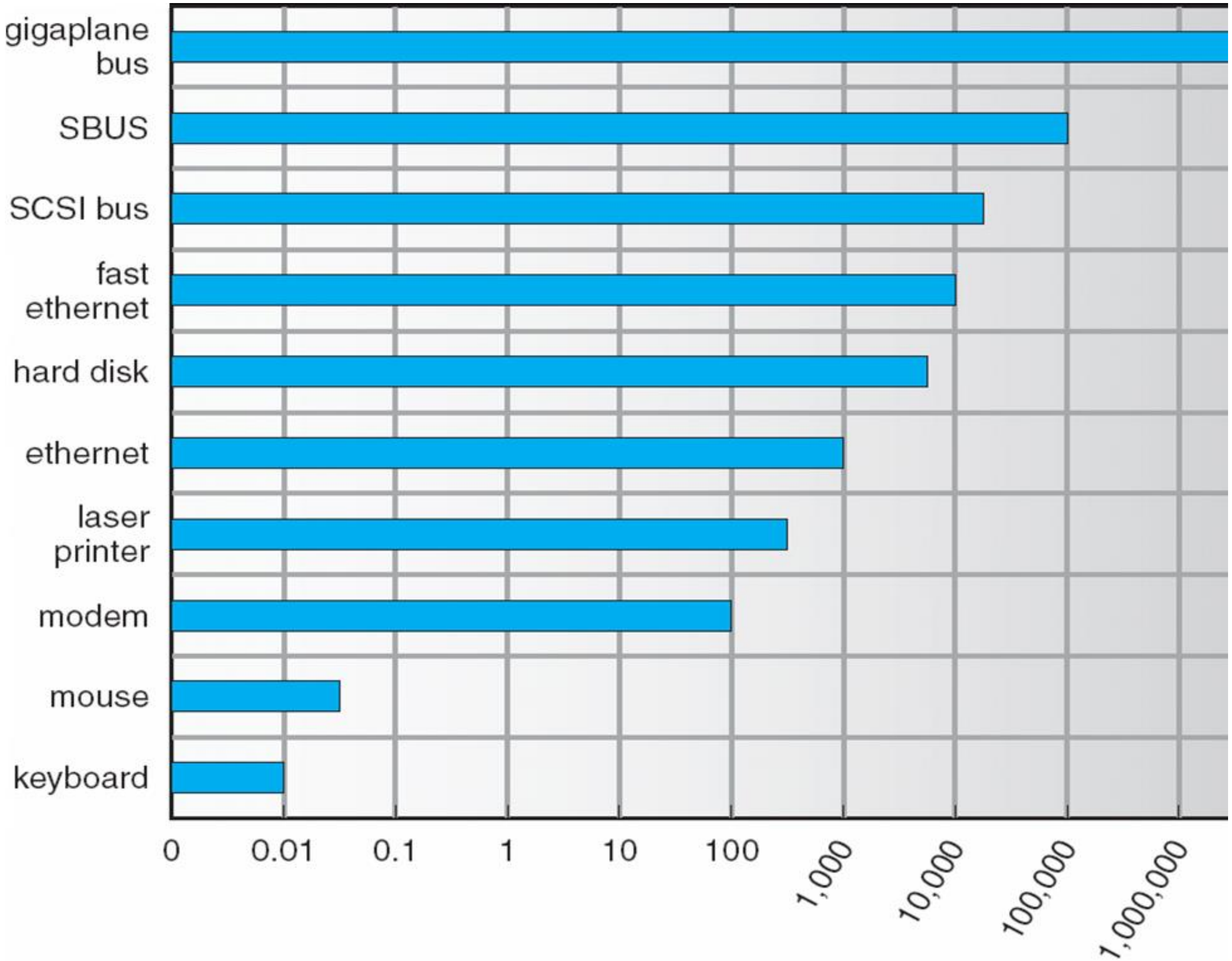
Kernel I/O Subsystem

- 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”

Device-status Table



Sun Enterprise 6000 Device-Transfer Rates



Kernel I/O Subsystem

Caching - fast memory holding copy of data

- Always just a copy
- Key to performance

Spooling - hold output for a device

- If device can serve only one request at a time
- i.e., Printing

Device reservation - provides exclusive access to a device

- System calls for allocation and deallocation
- Watch out for deadlock

Error Handling

OS can recover from disk read, device unavailable, transient write failures

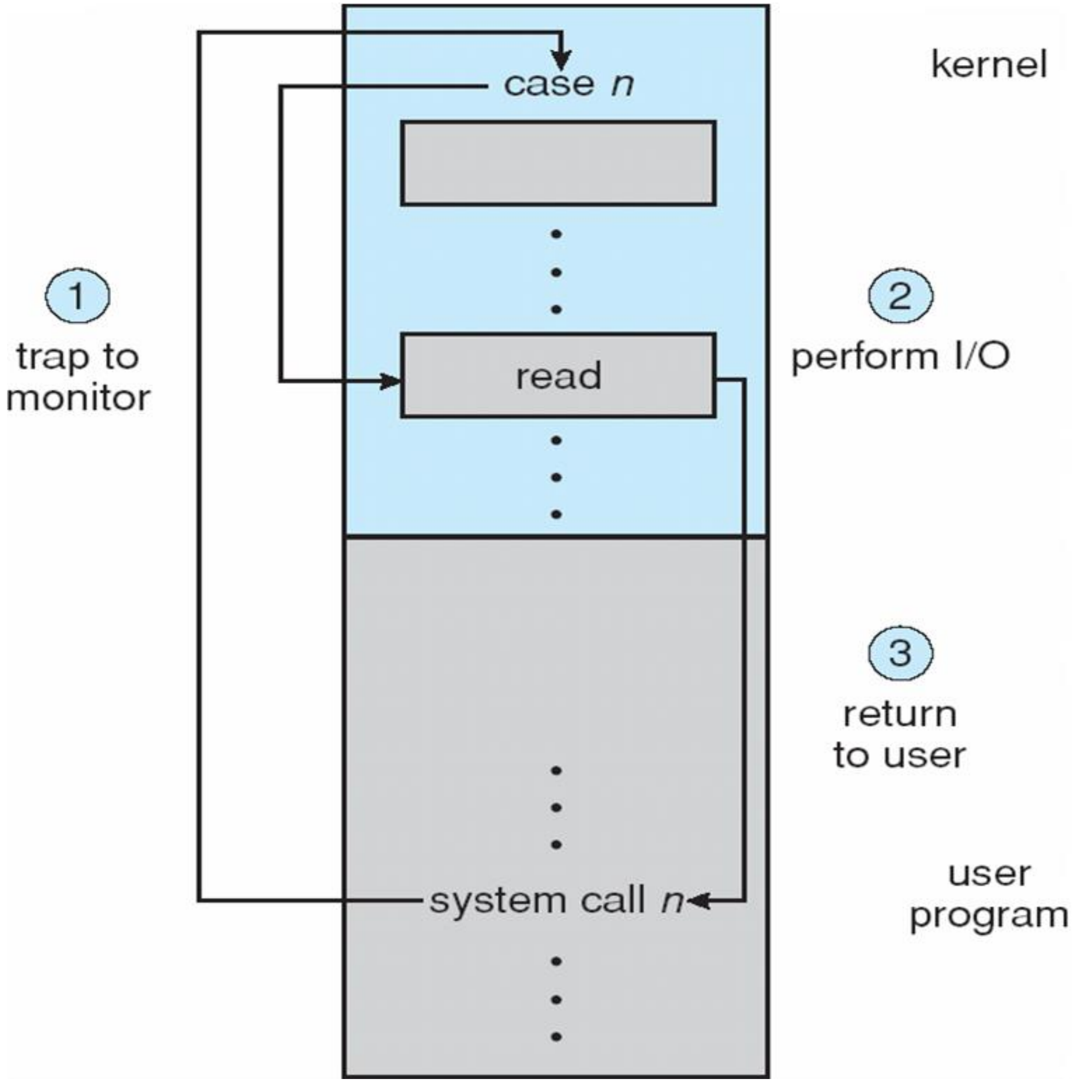
Most return an error number or code when I/O request fails

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

Use of a System Call to Perform I/O



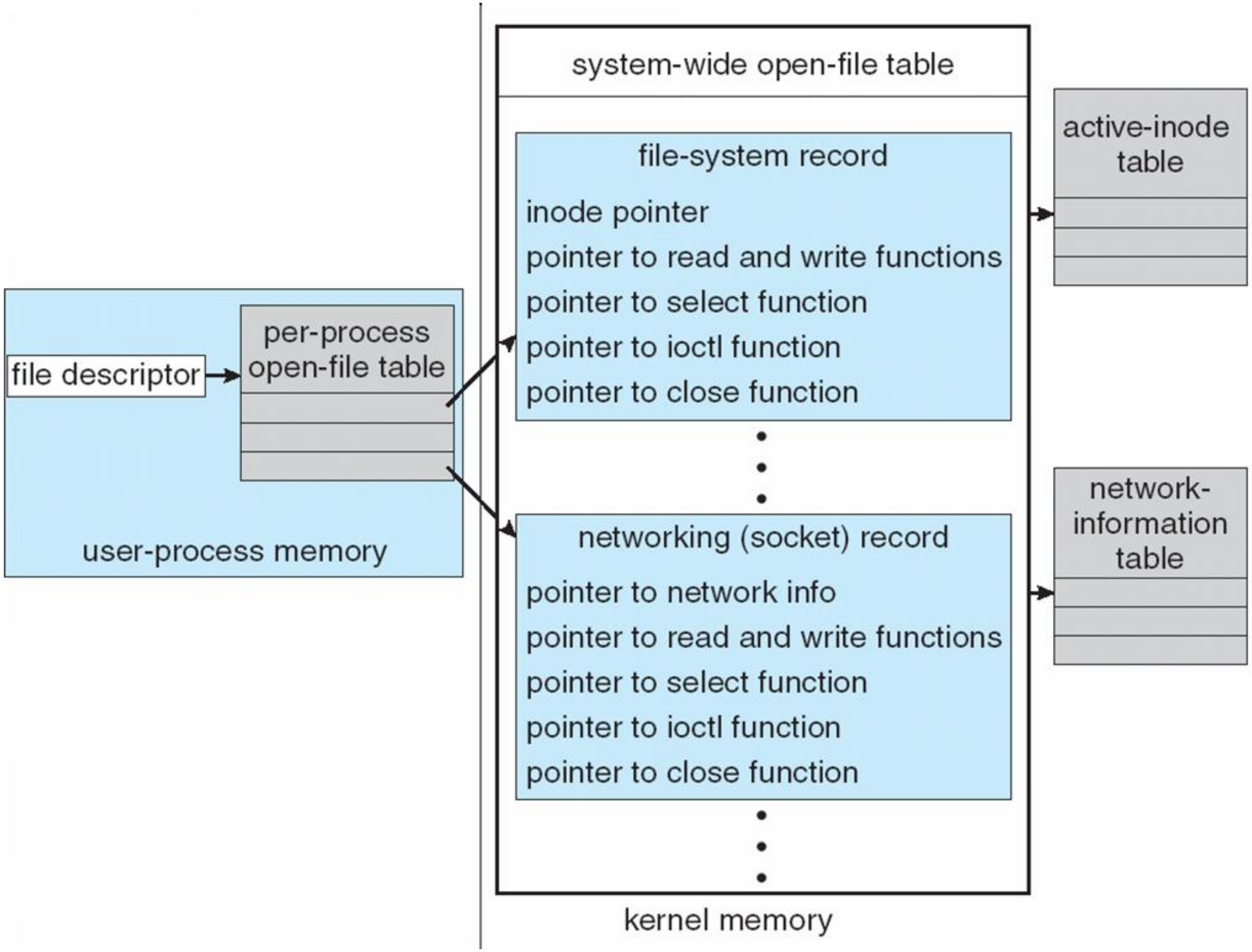
Kernel Data Structures

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

UNIX I/O Kernel Structure

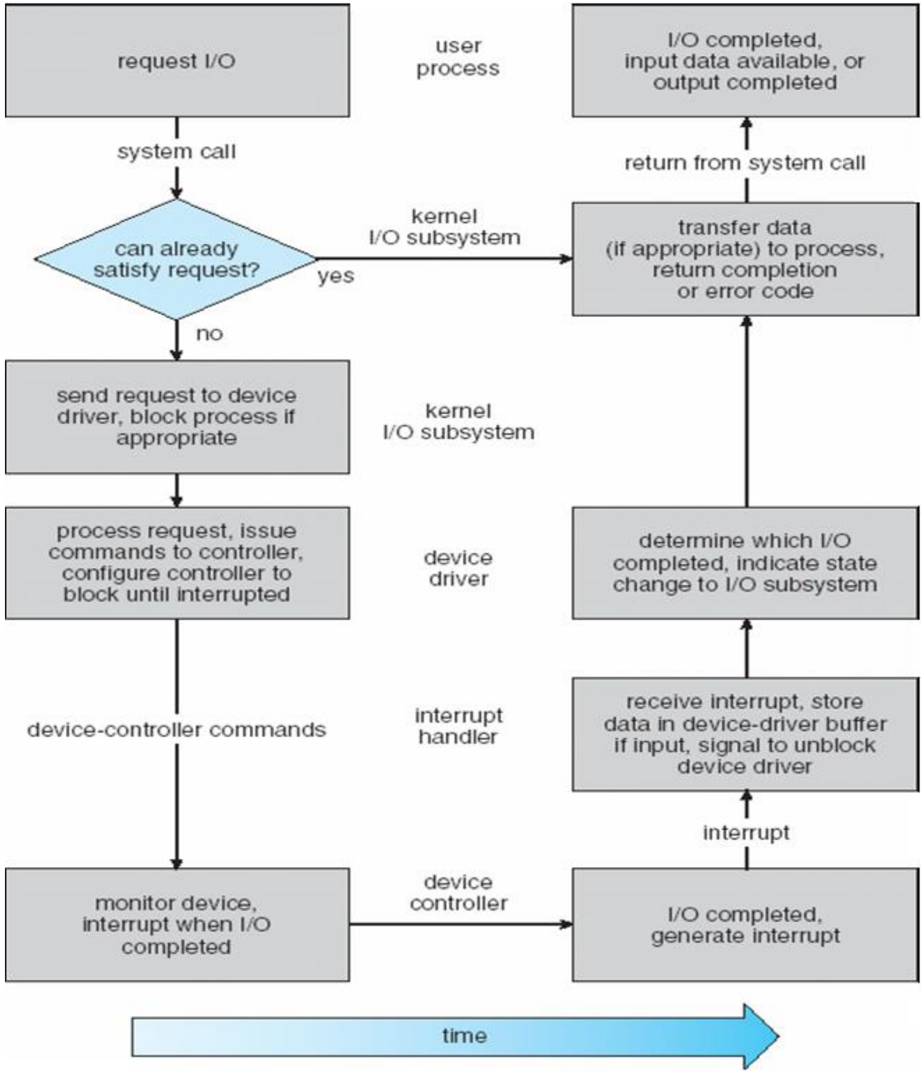


I/O Requests to Hardware Operations

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

Life Cycle of An I/O Request



STREAMS

STREAM – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond

A STREAM consists of:

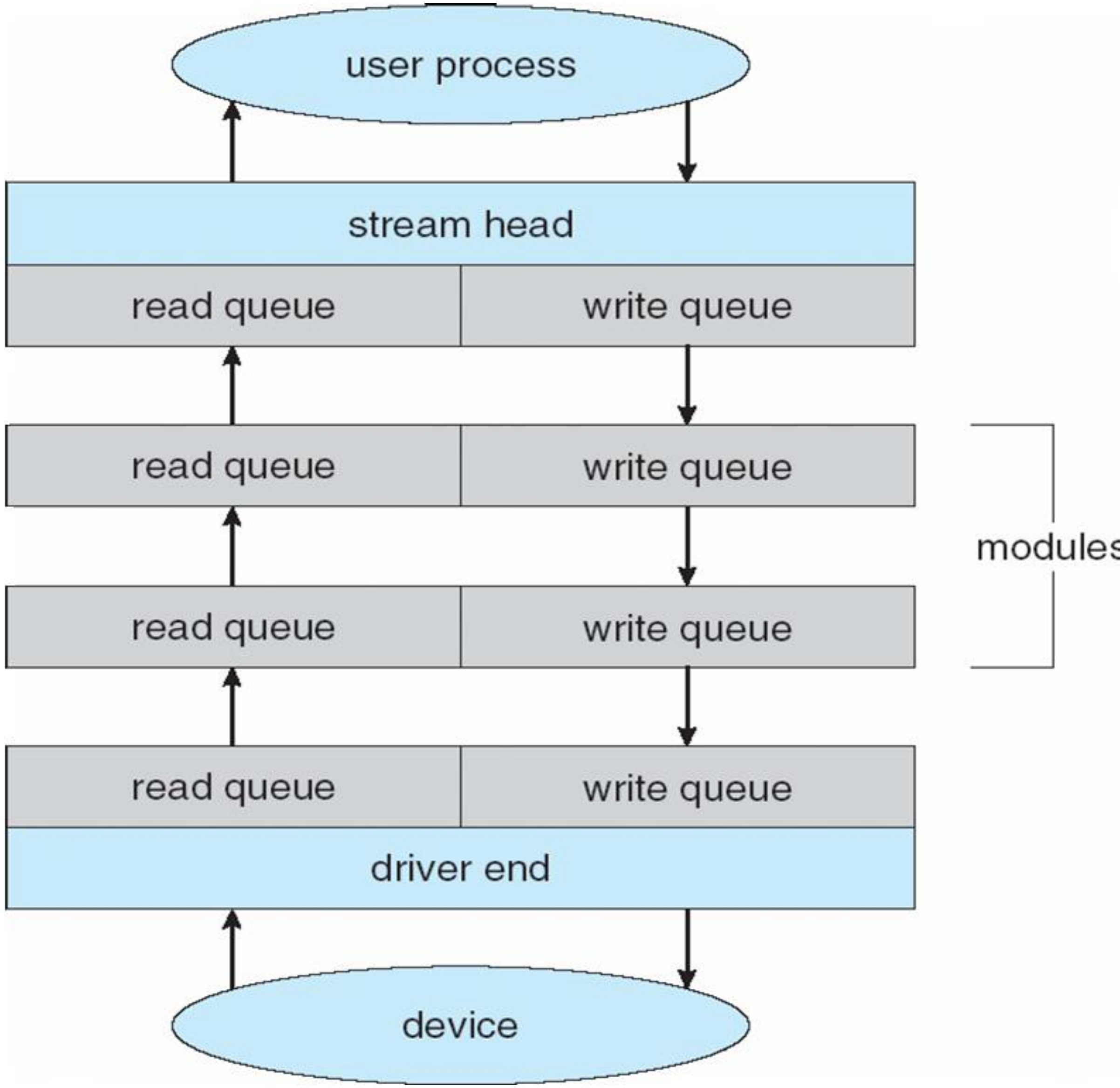
- STREAM head interfaces with the user process

- driver end interfaces with the device
- zero or more STREAM modules between them.

Each module contains a **read queue** and a **write queue**

Message passing is used to communicate between queues

The STREAMS Structure

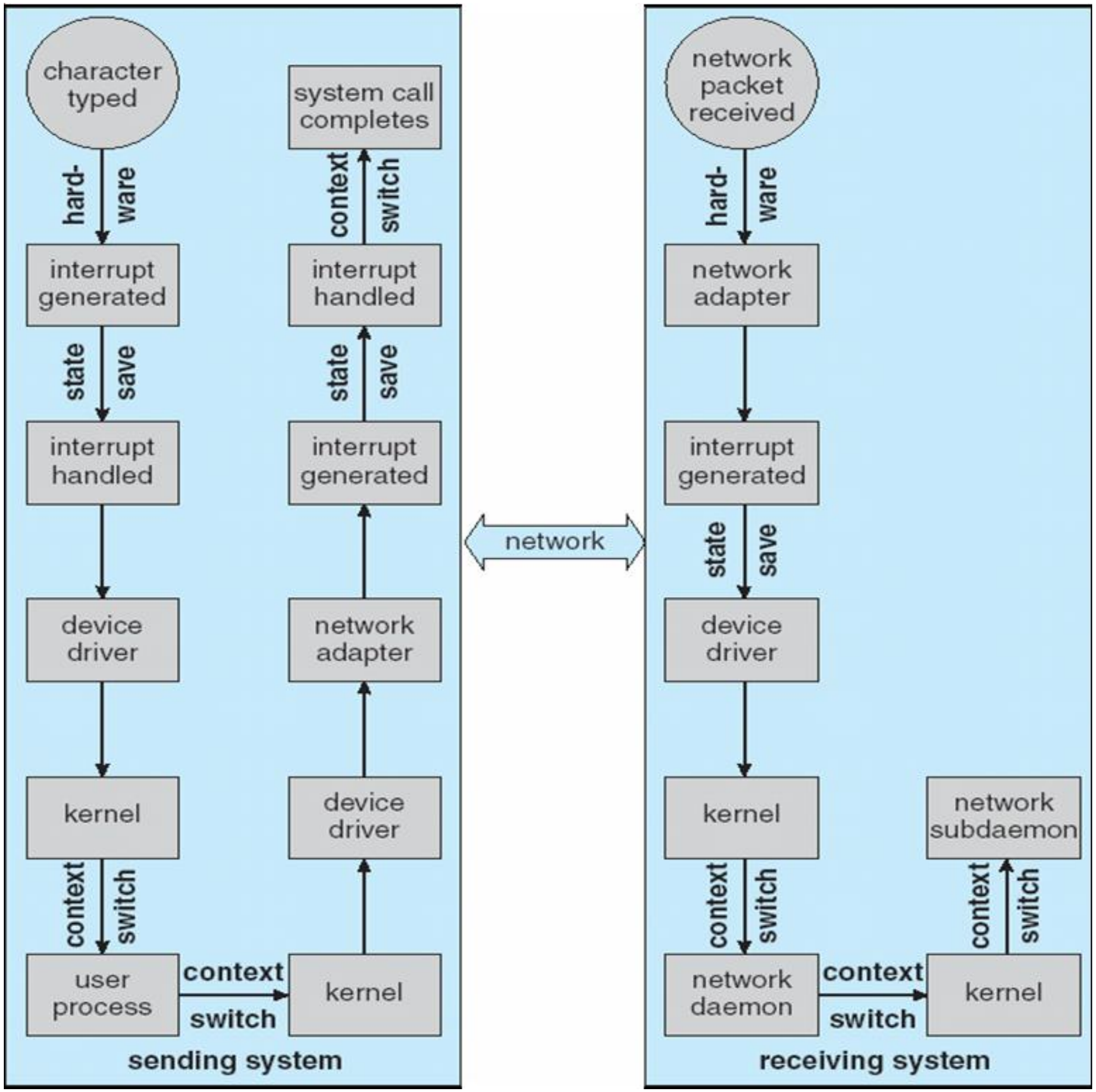


Performance

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

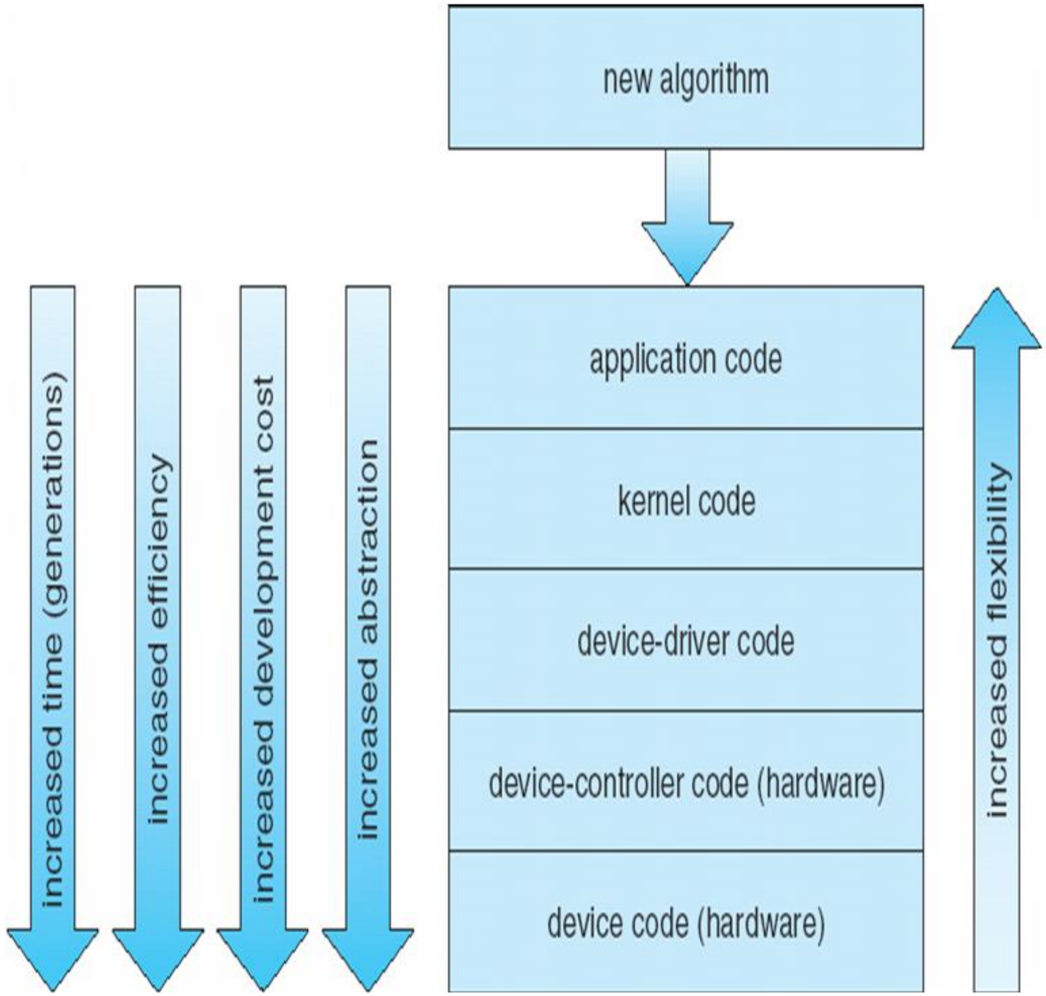
Intercomputer Communications



Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression



End of Chapter 13