Comp 310
Computer Systems and Organization

Lecture #21

I/O Systems

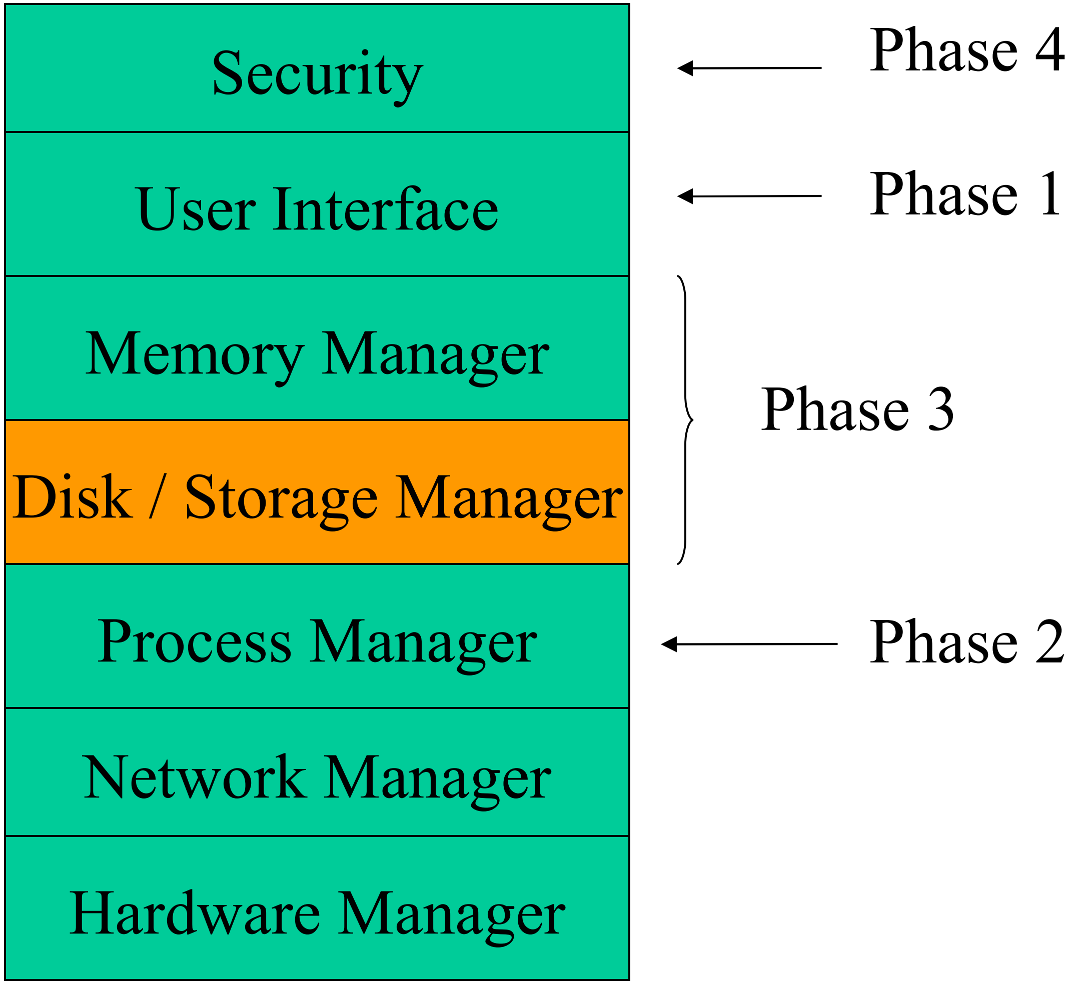
Prof. Joseph Vybihal

Announcements

- Final Exam Dec 9, 2PM
- **Course Evaluations**

Basic OS Architecture

(Course Table of Contents)



Security ← Phase 4

User Interface ← Phase 1

Memory Manager

Disk / Storage Manager

} Phase 3

Process Manager ← Phase 2

Network Manager

Hardware Manager

Part 1

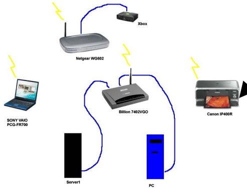
I/O Systems

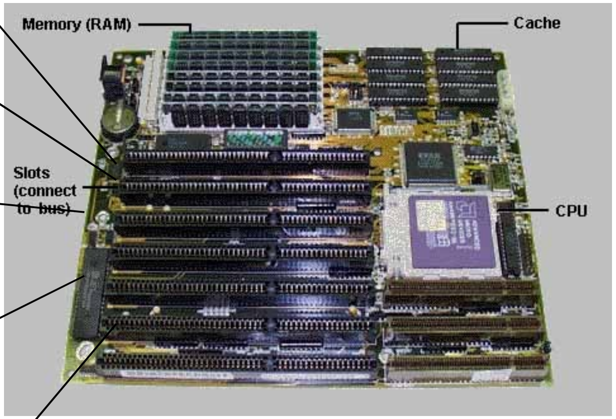
General Purpose OS

A Manager

The OS

OS manages through the system board

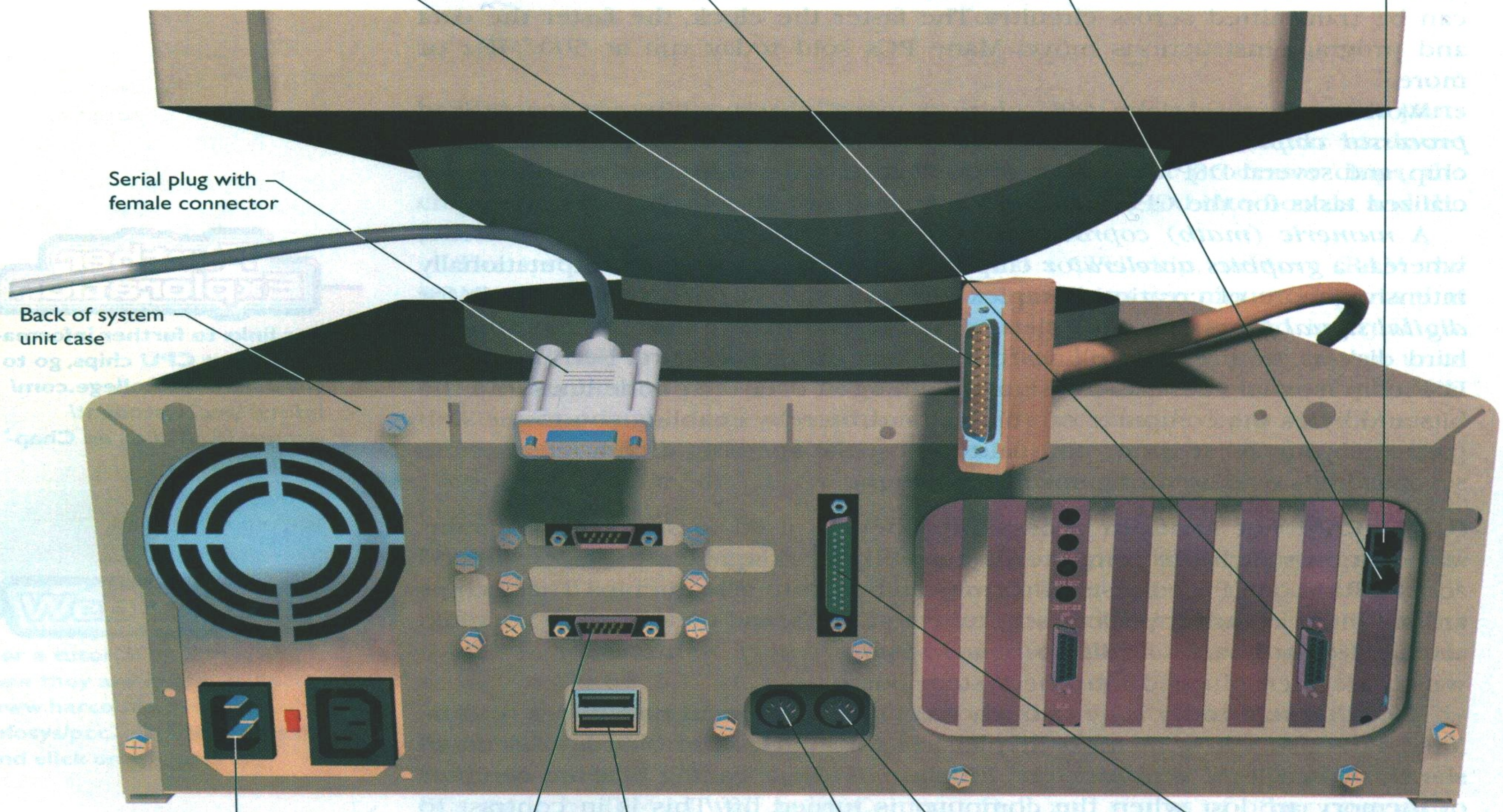




Being a Device Manager is one of the OS's biggest jobs.

Equal to process and memory management.

Computer Ports



MONITOR PORT
The monitor port is used to connect a monitor.

MODEM CONNECTOR
Plug the modem jack in here and connect the other end to the phone outlet.

PHONE CONNECTOR
If you unplugged a telephone to connect your modem, plug the cable from the phone in here.

Parallel plug with male connector

Serial plug with female connector

Back of system unit case

POWER CONNECTOR
The power connector is used with a special cable to connect to a wall outlet.

SERIAL PORT
Serial ports, which have either 9 or 25 pins, are used to connect such low-speed peripherals as scanners and external modems.

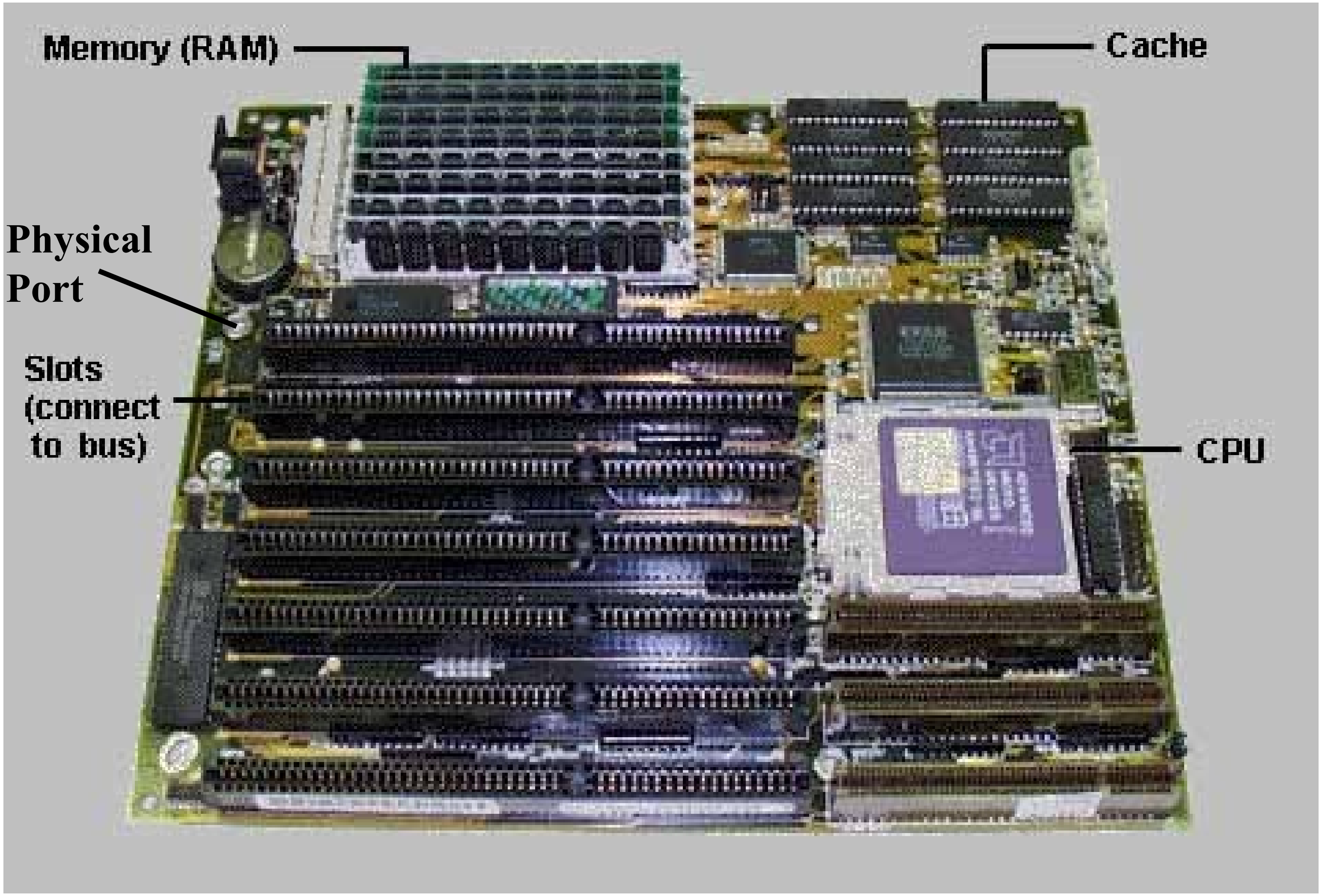
USB PORT
USB ports allow you to connect several devices to a single port. Most new computers come with two USB ports.

MOUSE PORT
The mouse port is used to connect a mouse.

KEYBOARD PORT
The keyboard port is used to connect a keyboard.

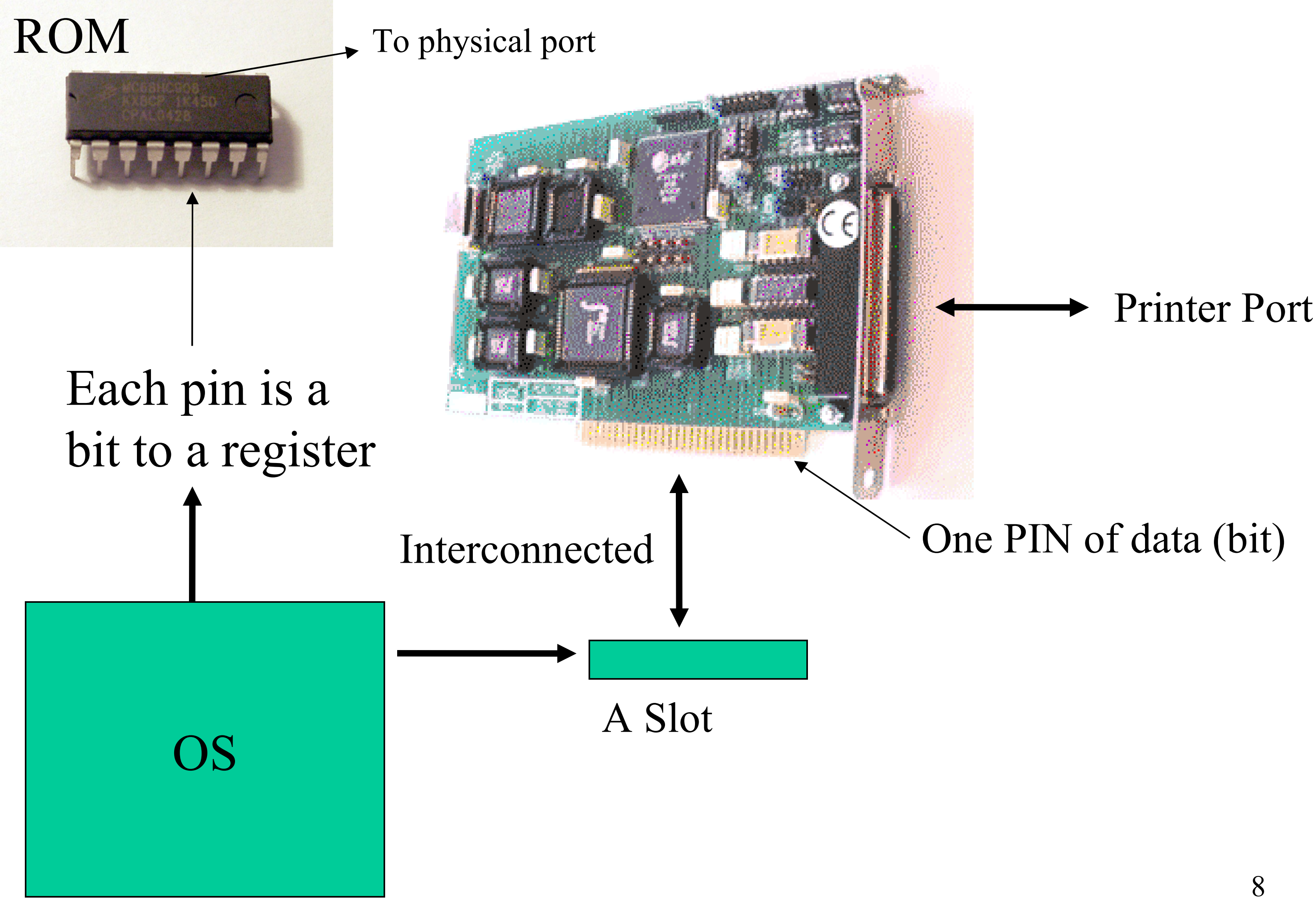
PARALLEL PORT
Parallel ports have 25 holes and are most commonly used for printers and tape drives.

The Motherboard/System Board

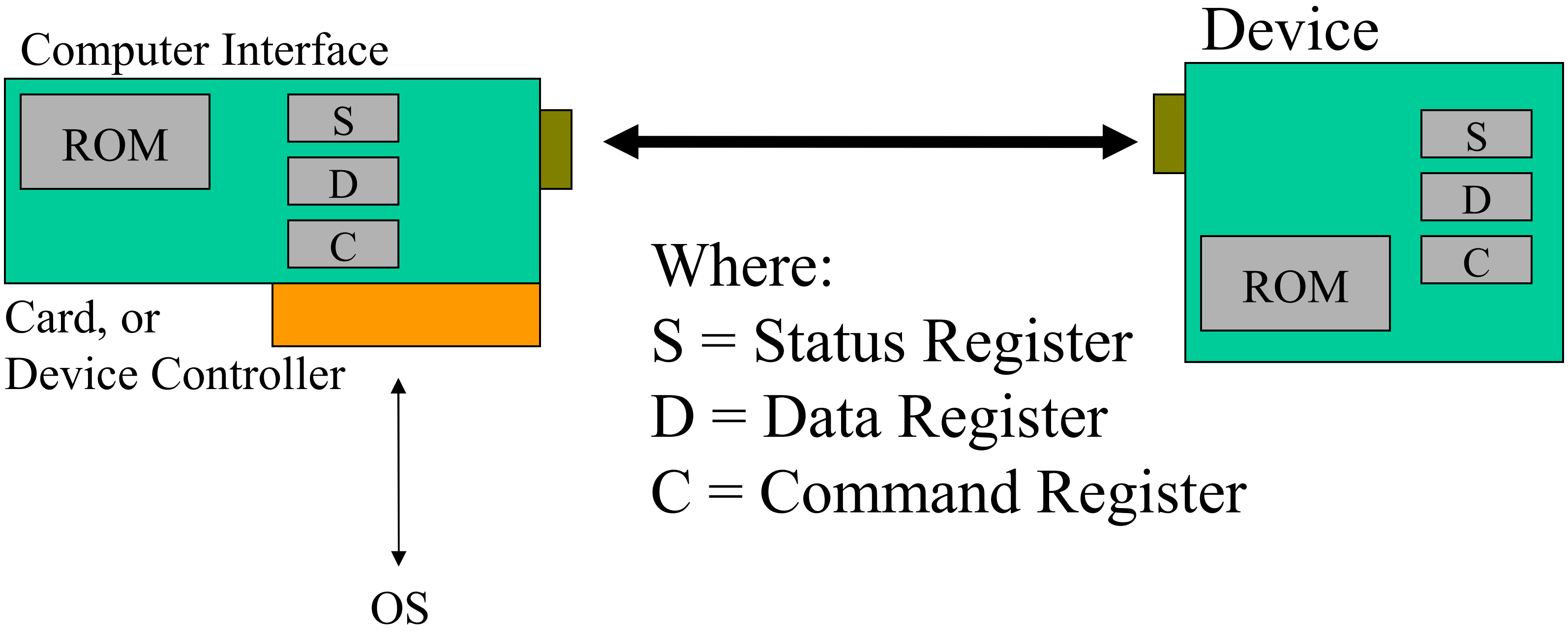


Devices connect via Physical Ports or System Slots – **no other way**

Slots, Cards, Ports & ROMS



Flow Diagram



Where:
S = Status Register
D = Data Register
C = Command Register

<u>Device Operation</u>:

2. Device updates S (optionally D)

- Waits a small unit of time and then reads C and D
  (possible data loss)

<u>Polling Method</u>:

- Check S for change in a busy loop
- If change then copy S and D to memory

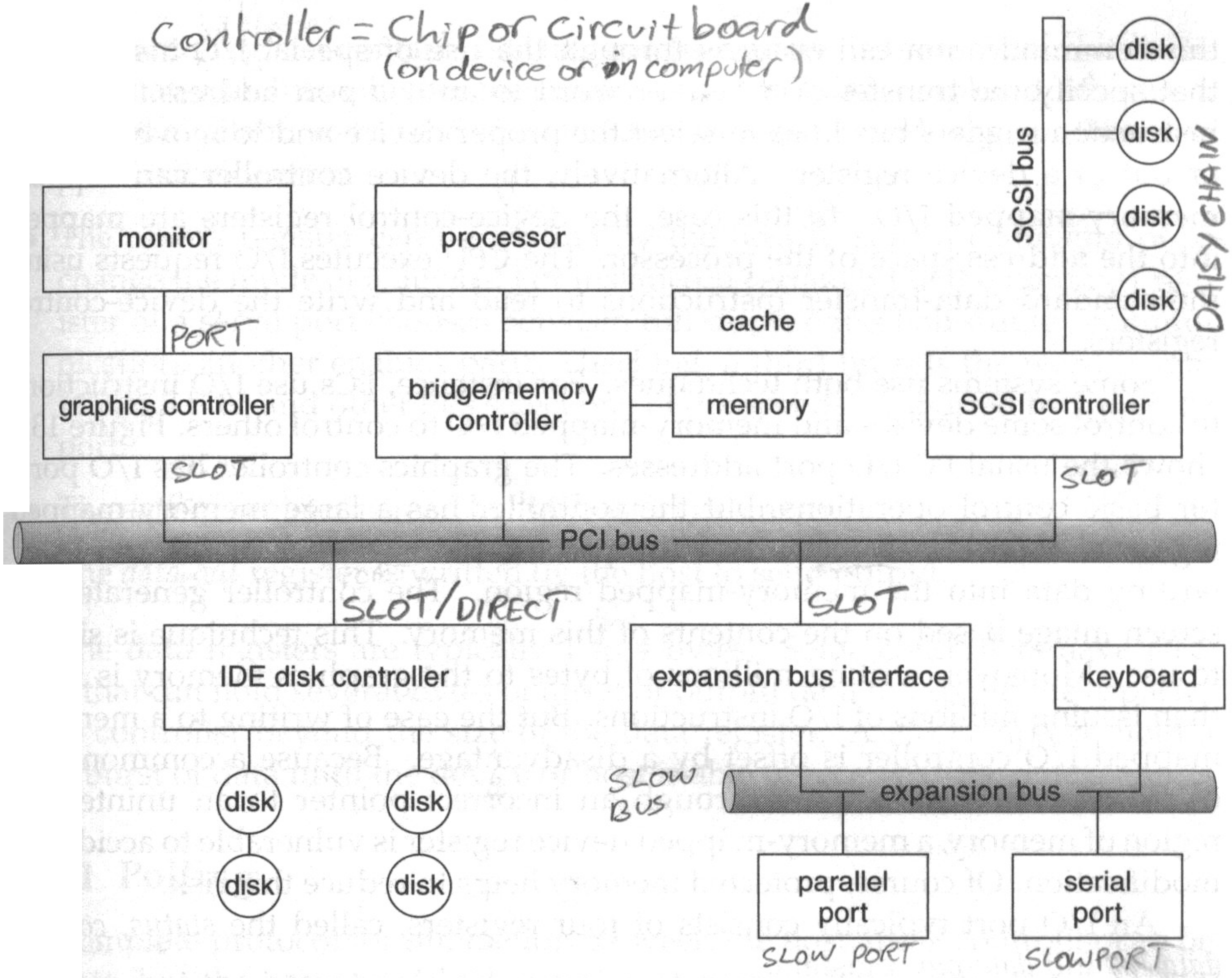
<u>Interrupt Method</u>:

- If S or D updated signal CPU
- CPU Task switch to OS

Question

- Using pseudo-assembler, how would the OS implement polling for a process?
  - How would this relate to the process’ ready or sleep queue status?
  - How would the OS manage this:
    - Queue, assembler and interrupts?

Controller = Chip or Circuit board
(on device or in computer)

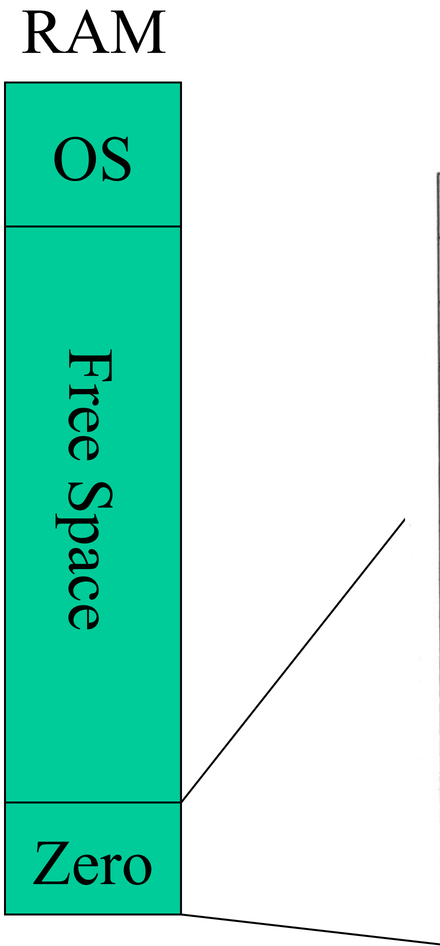


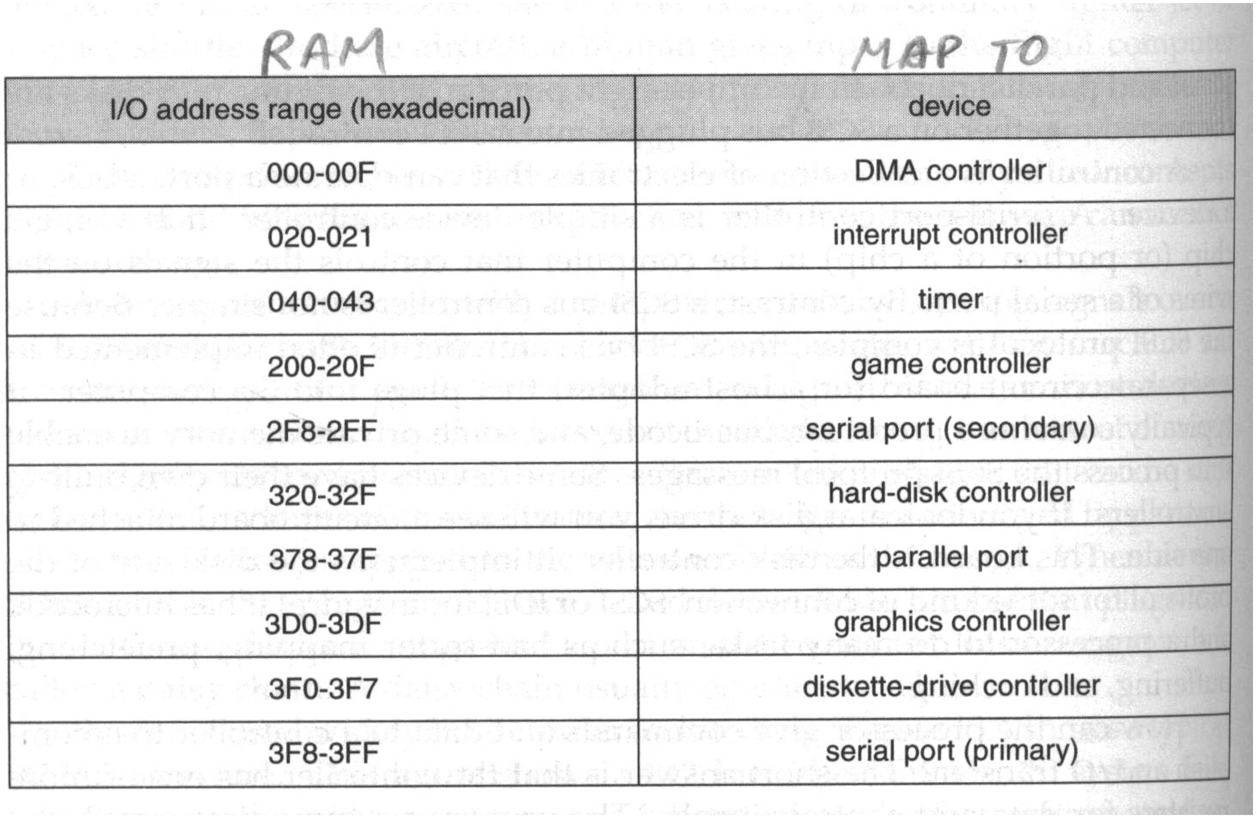
Access to Information

- So… all the information is in those cards and chips… how do I get access to them?

- Direct Memory Mapped I/O
  - A section of RAM wired to system board:
    - Slots, and
    - Physical Ports
  - Any activity that occurs in a slot or port is mirrored in this section of RAM
  - This section of RAM is a two-way path (Input and Output) to these slots and ports
- Direct Memory Access
  - The device can write directory to RAM without the OS or CPU

Memory Mapped I/O




RAM MAP TO

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)

Partial IBM PC Mapped Address Space

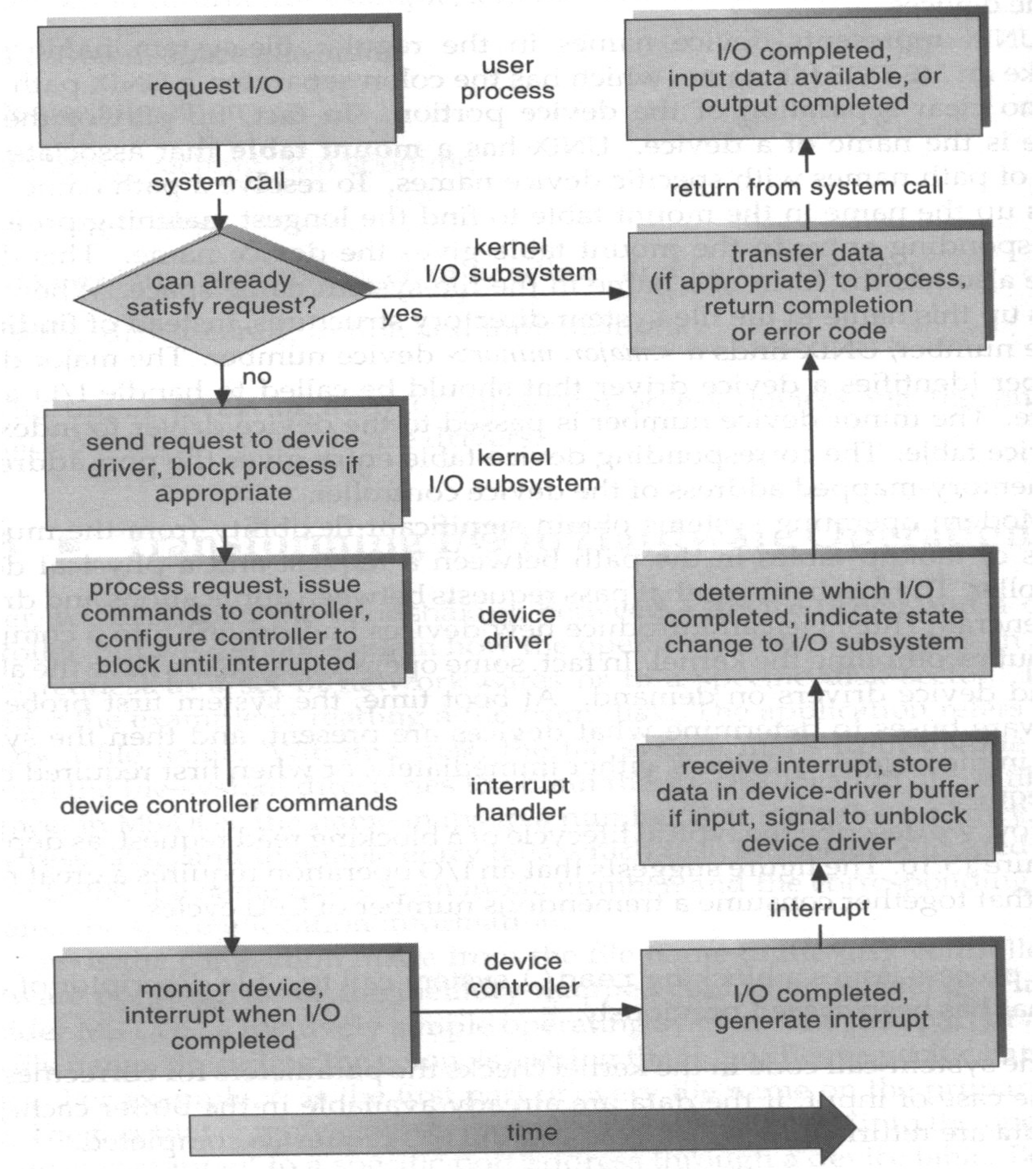
Assembler/Pointer Programming

- To take advantage of this memory mapped space requires low-level programming.
- Two low-level programming techniques are used:
  - Polling, and
  - Interrupts

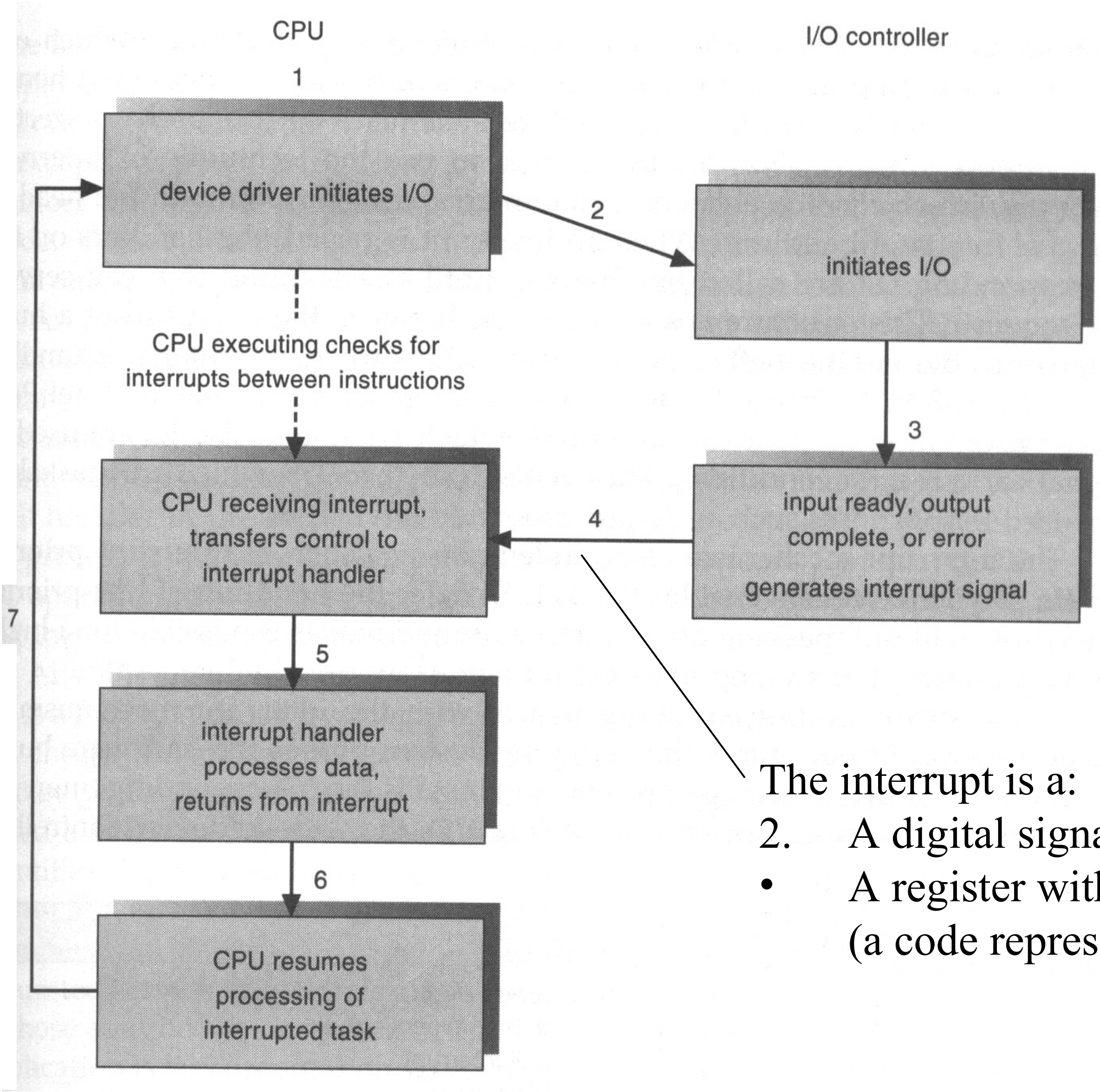
Polling

- Handshaking
  - OS waits for device to indicate that it is not busy
  - OS then deposits a command and data to device registers
  - Device now performs action & OS waits
  - Device returns status information
  - OS reads this information
- Waiting
  - Is a while-loop (a busy loop) that does nothing but loop until the wait is over
  - The OS knows when the wait is over because the device updates the Status register to indicate its state.
- Simple mechanism inserted directly into any OS fn

Hardware Communication


request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
kernel I/O subsystem
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

The Interrupt Process


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
input ready, output complete, or error generates interrupt signal
4
CPU receiving interrupt, transfers control to interrupt handler
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

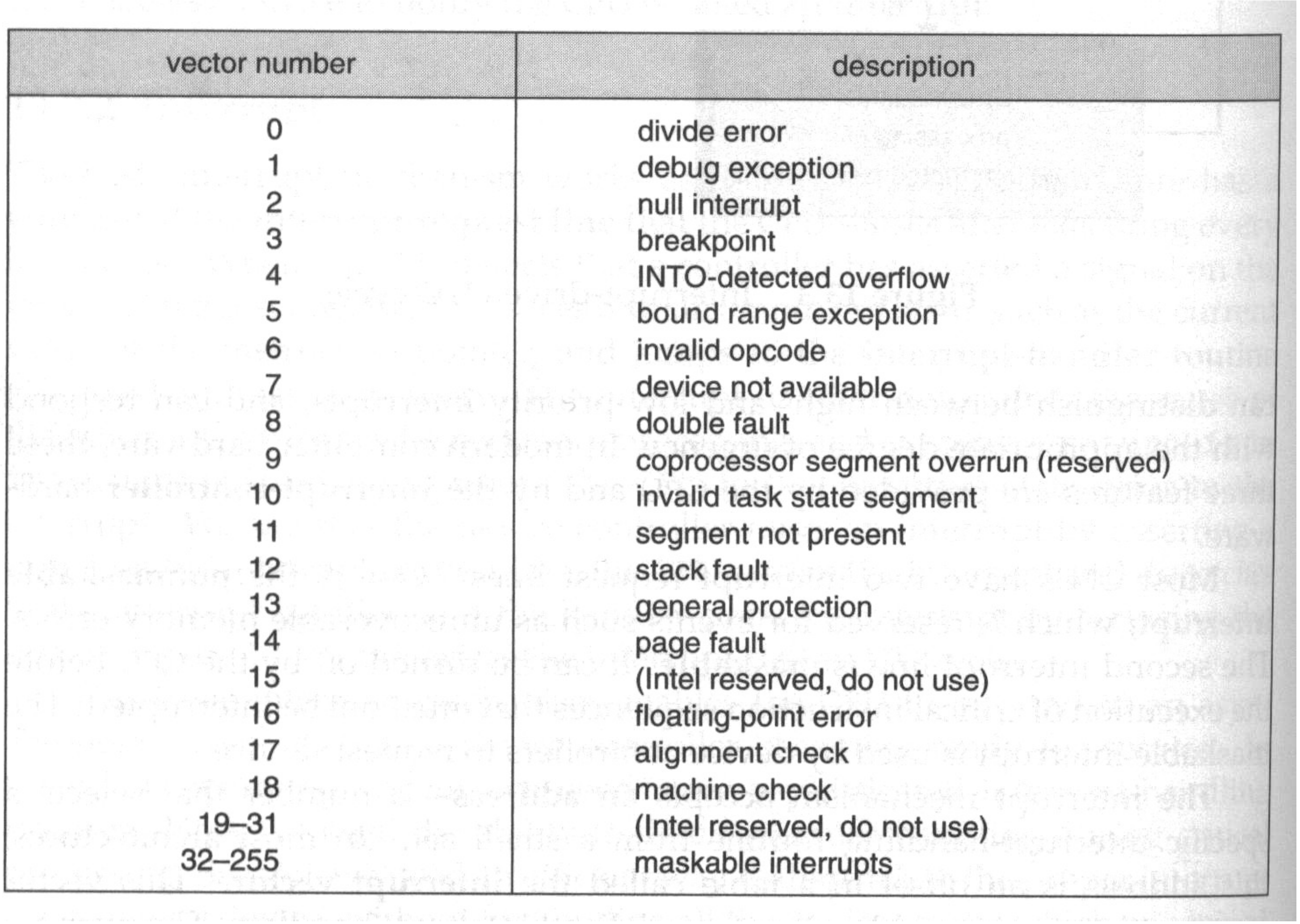
The interrupt is a:

2. A digital signal to stop the CPU

• A register with an integer number (a code representing a message)

Sample INTEL Interrupt Codes

Note: not dependent on OS, but CPU & Device



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

INTEL Pentium Processor event-vector

Interrupt Handling

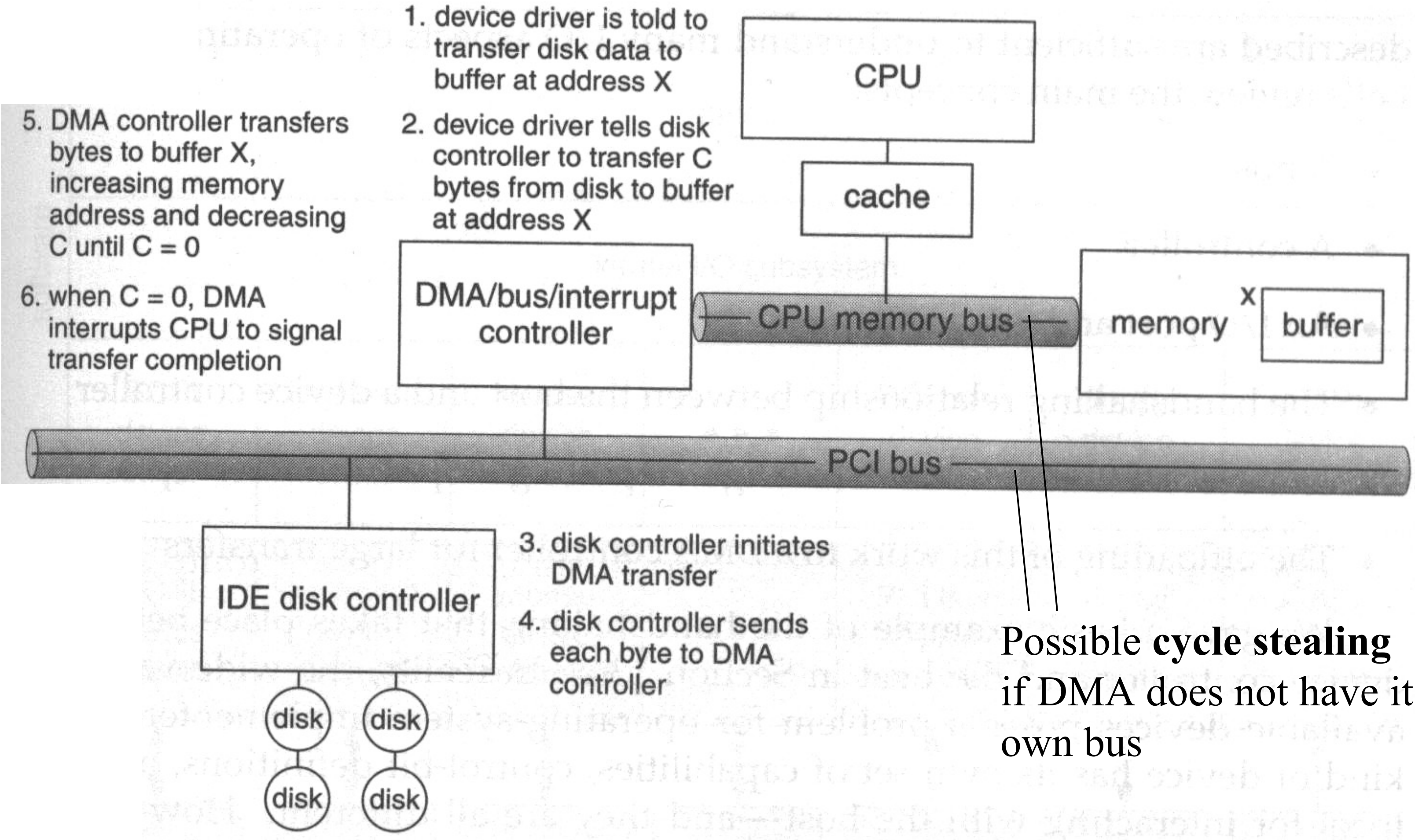
- Interrupt Handler
  - A special function called to process an interrupt from a specific device
  - Each device needs a custom handler
  - The handler is aware of the registers and codes used by that specific device (this is true for polling as well)
- Interrupt Messages
  - Stored as codes in registers
  - Two code types:
    - Integers (integer code number representing a message)
      - Processed by a switch statement
    - Flags (a bit set to 1 for true or 0 for false)
      - Processed by bit masking

Interrupt Notes

- Software can also issue interrupts
  - Timers
  - Java Action Listeners
- Interrupts can be interpreted by the OS as having priority
  - Ordering how the OS will process them
    - User interface interrupts have higher priority
- Interrupts that are not handled quickly by the OS can be overwritten by the next hardware signal
- Each CPU and device has their own event-vector table with their own unique codes
  - Microsoft Mouse
  - Logitech Mouse

MDA

(Direct Memory Addressing)



1. device driver is told to transfer disk data to buffer at address X

2. device driver tells disk controller to transfer C bytes from disk to buffer at address X

3. disk controller initiates DMA transfer

4. disk controller sends each byte to DMA controller

5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0

6. when C = 0, DMA interrupts CPU to signal transfer completion

Possible **cycle stealing** if DMA does not have its own bus

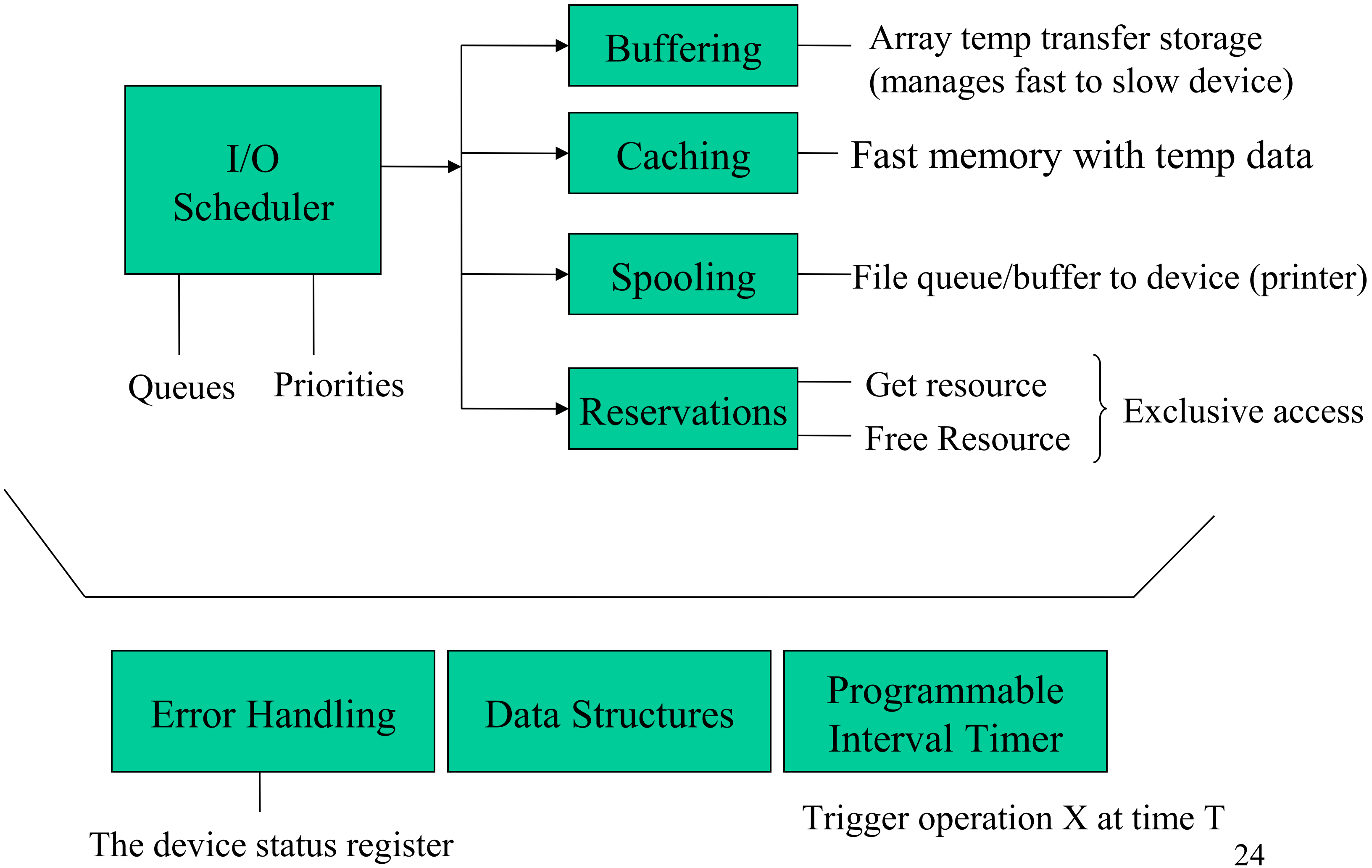
Part 2

The OS Point-of-view

Management Issues

- Device hardware specifications
- Device communication features
- Motherboard data transfer features
- Optimization & Impact on Process
  - Speed
  - Memory
- Security

I/O Sub-System Architecture



I/O Scheduler

Queues

Priorities

Buffering — Array temp transfer storage (manages fast to slow device)

Caching — Fast memory with temp data

Spooling — File queue/buffer to device (printer)

Reservations — Get resource, Free Resource — Exclusive access

Error Handling

The device status register

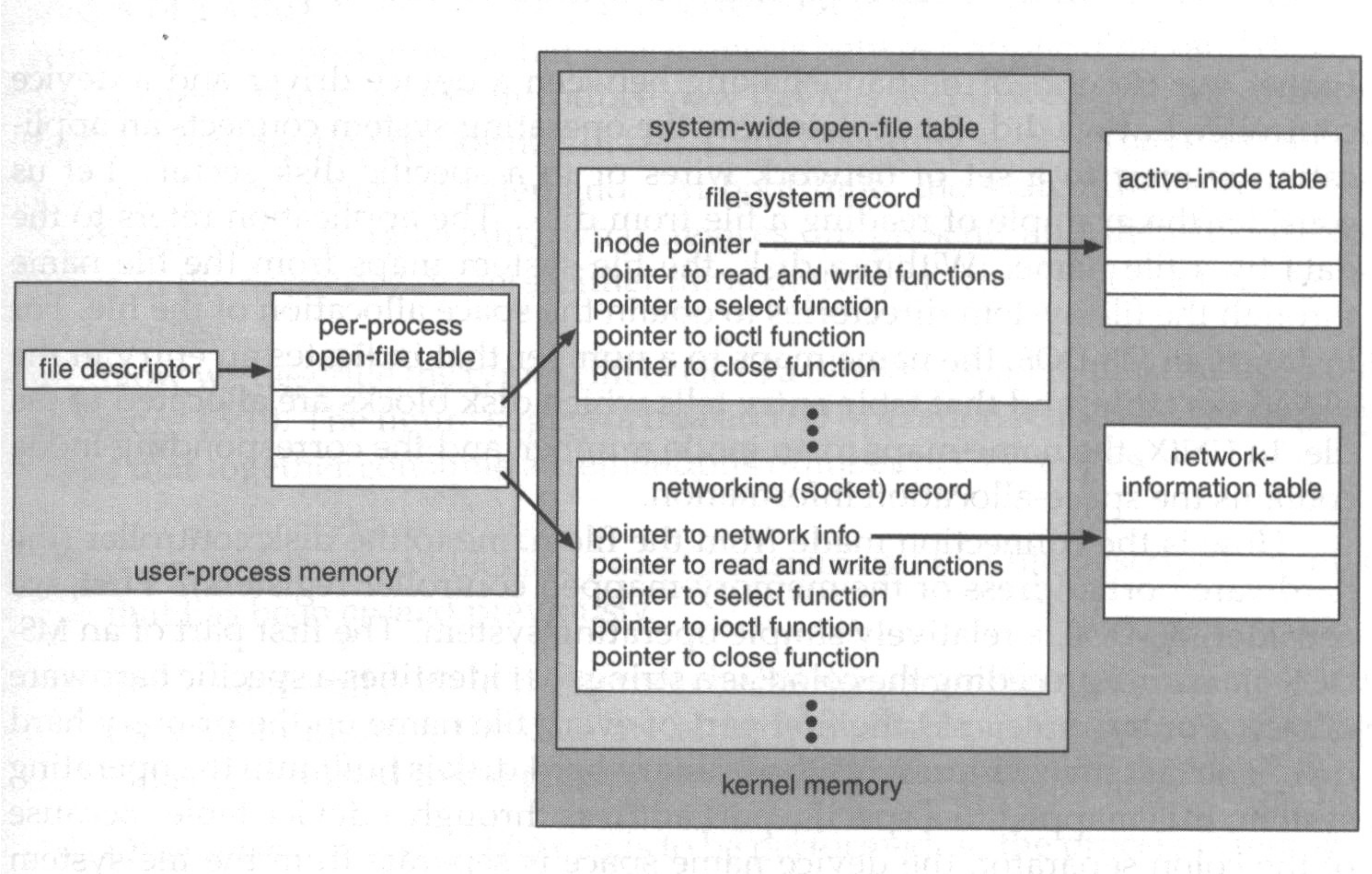
Data Structures

Programmable Interval Timer

Trigger operation X at time T

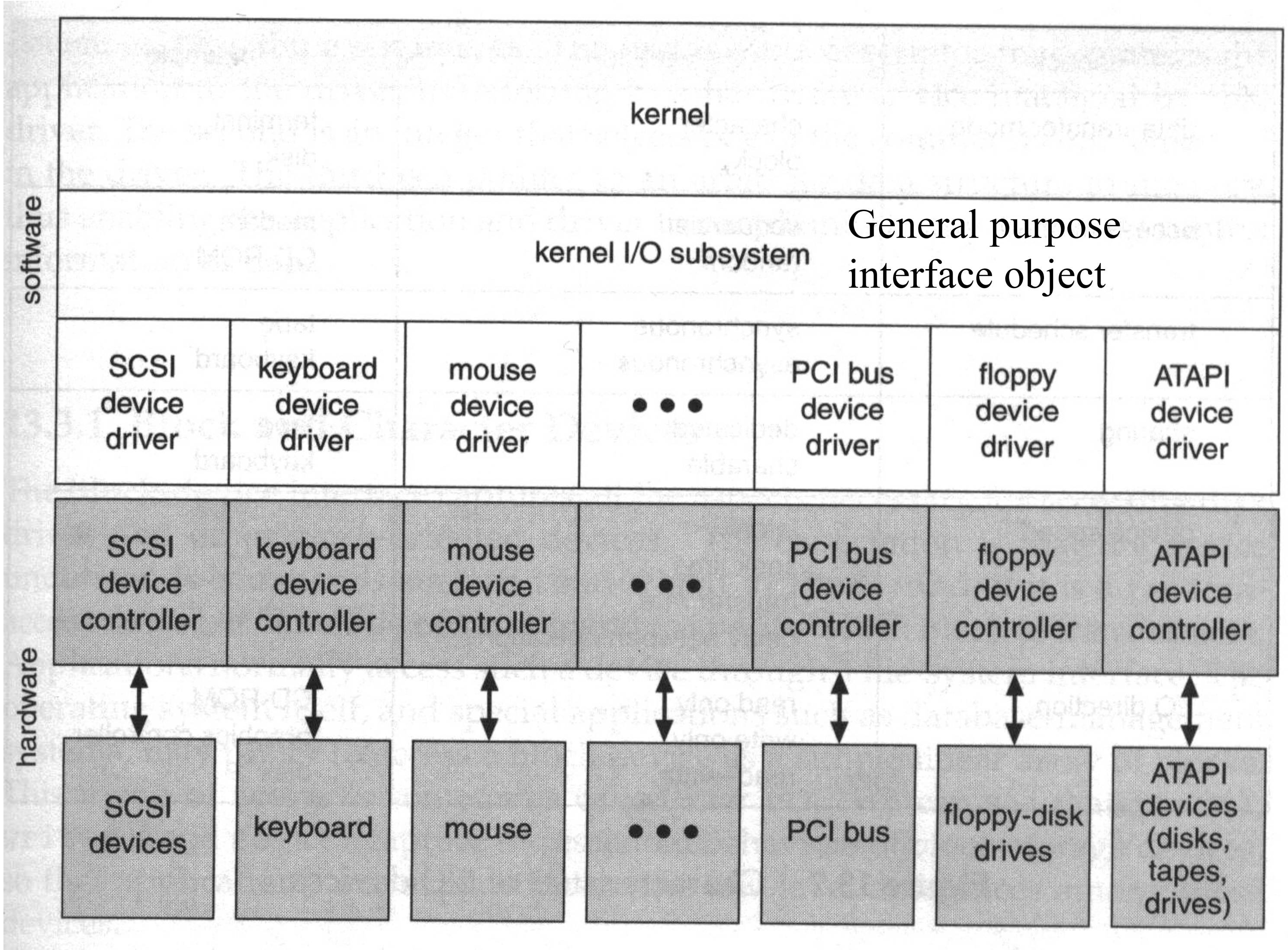
UNIX I/O Kernel Data Structure

(Example – partial structure)


system-wide open-file table
file-system record
inode pointer
pointer to read and write functions
pointer to select function
pointer to ioctl function
pointer to close function
active-inode table
file descriptor
per-process open-file table
networking (socket) record
pointer to network info
pointer to read and write functions
pointer to select function
pointer to ioctl function
pointer to close function
network-information table
user-process memory
kernel memory

Only shows file I/O

The Kernel I/O Structure

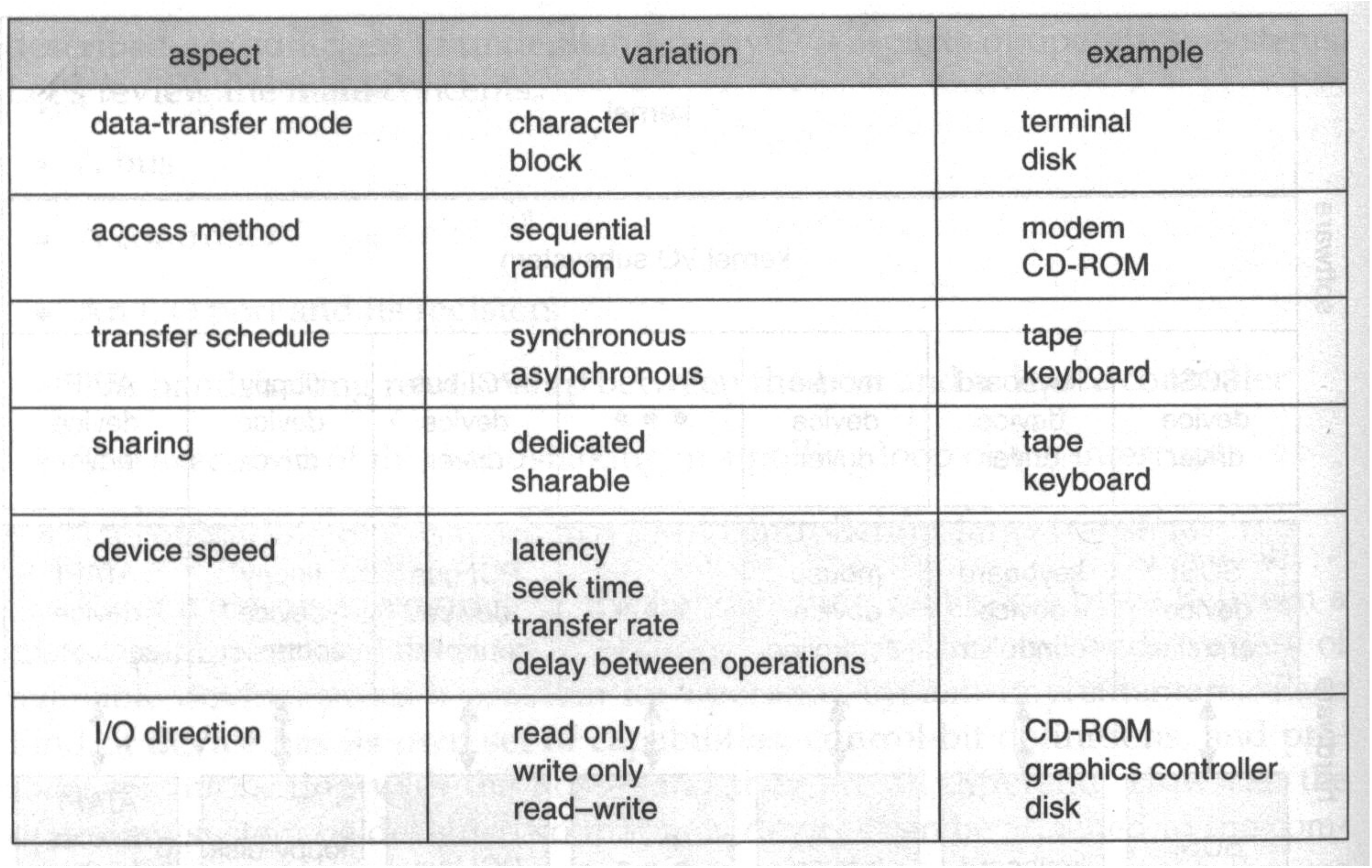


General purpose interface object

Multiple specific drivers

Device Communication Features

Each device speaks differently



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

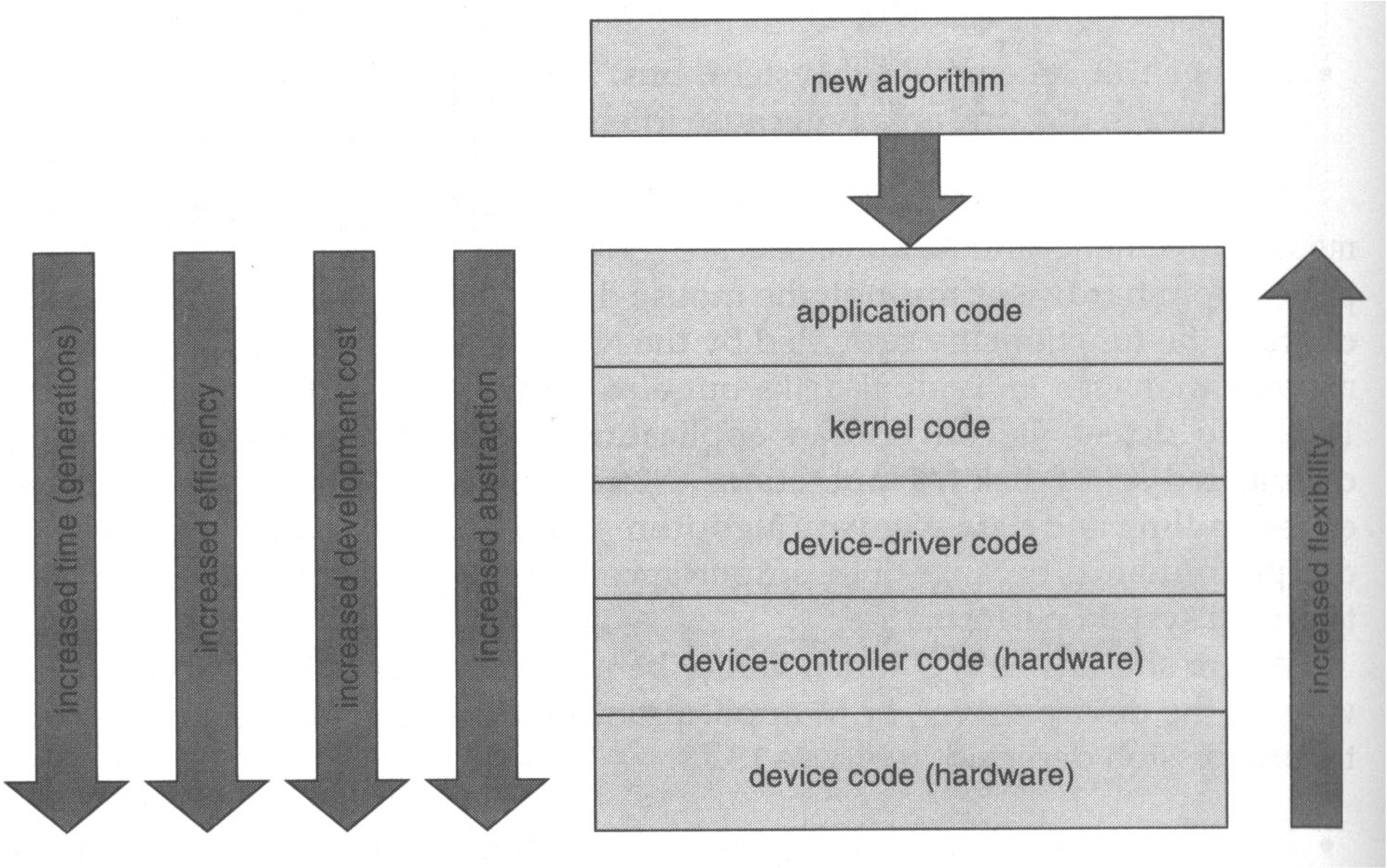
Device Communication Modes

- Block Driven
  - I/O cannot function by byte
  - I/O access by a fixed number of bytes called a block
  - The entire set of bytes is loaded into a buffer as a single unit
- Character Stream
  - I/O access is by byte
  - A pointer increments past each byte
- Socket Based
  - An object that references a specific physical port
  - The user's application has a reference to this object
  - The object manages communication with the device

Device Communication Modes

- Blocking I/O
  - Stop executing process
  - Switch to driver
- Non-Blocking I/O
  - Device data accesses as a running process
    - Keyboard
    - Mouse

Programming Complexity



Part 3

At Home

Things to try out

1. Have you every installed a driver?
2. Web Resources (I/O Systems):
   - http://www.cs.mun.ca/~paul/cs3725/material/web/notes/node28.html
   - http://download.oracle.com/docs/cd/B19306_01/server.102/b14211/iodesign.htm
   - http://www.freebsd.org/doc/en_US.ISO8859-1/books/design-44bsd/overview-io-system.html