CIS 520, *Operating Systems Concepts*

Lecture 8

Input/Output

I/O Devices

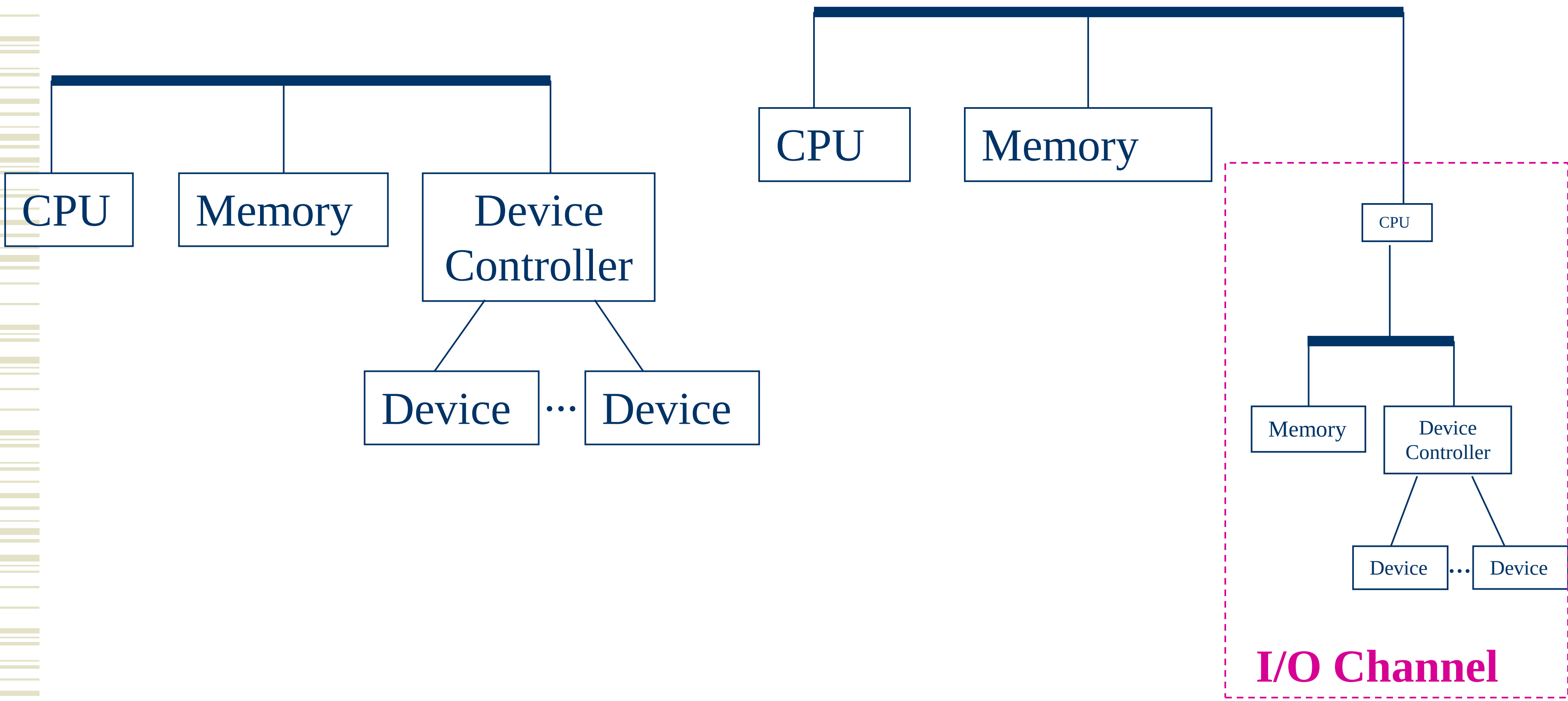
- The are two categories of the I/O devices:
  - *Block Devices* store and deliver information in blocks of bytes
    - *Disk* uses fixed-size blocks
    - *Tape* and *asynchronous terminals* use variable size blocks
  - *Character Devices* (*dumb terminals, keypunch, card readers, printers, MIDI)* store and deliver information as a stream of bytes

And then there are devices (e.g., *computer clocks*, *telemetric devices*, etc.) that don't belong to these categories.

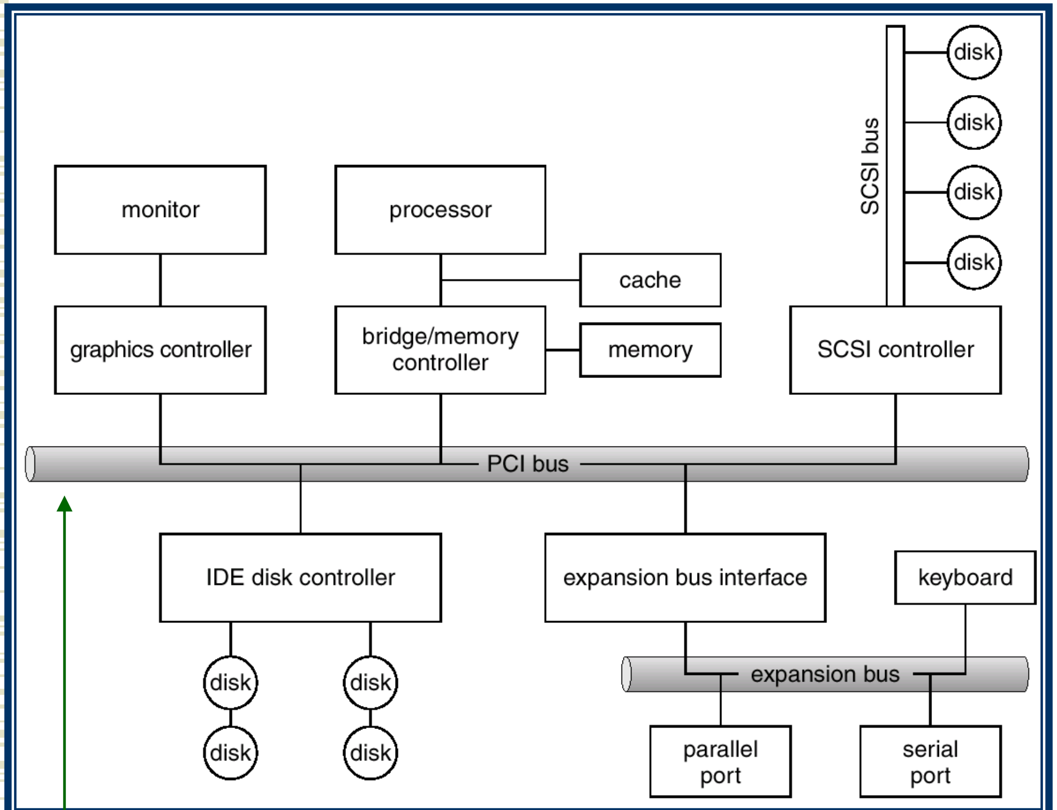
Device Controllers and I/O Channels

- The *programmable* component of a device is called a *controller*. Controllers connect to the system bus
- In some cases, a separate CPU is assigned to deal with a device (or a set of devices). Such a CPU and its supporting circuitry form an *I/O Channel*

Device Controllers and I/O Channels (cont.)



A Real-Life Example (from the Book)

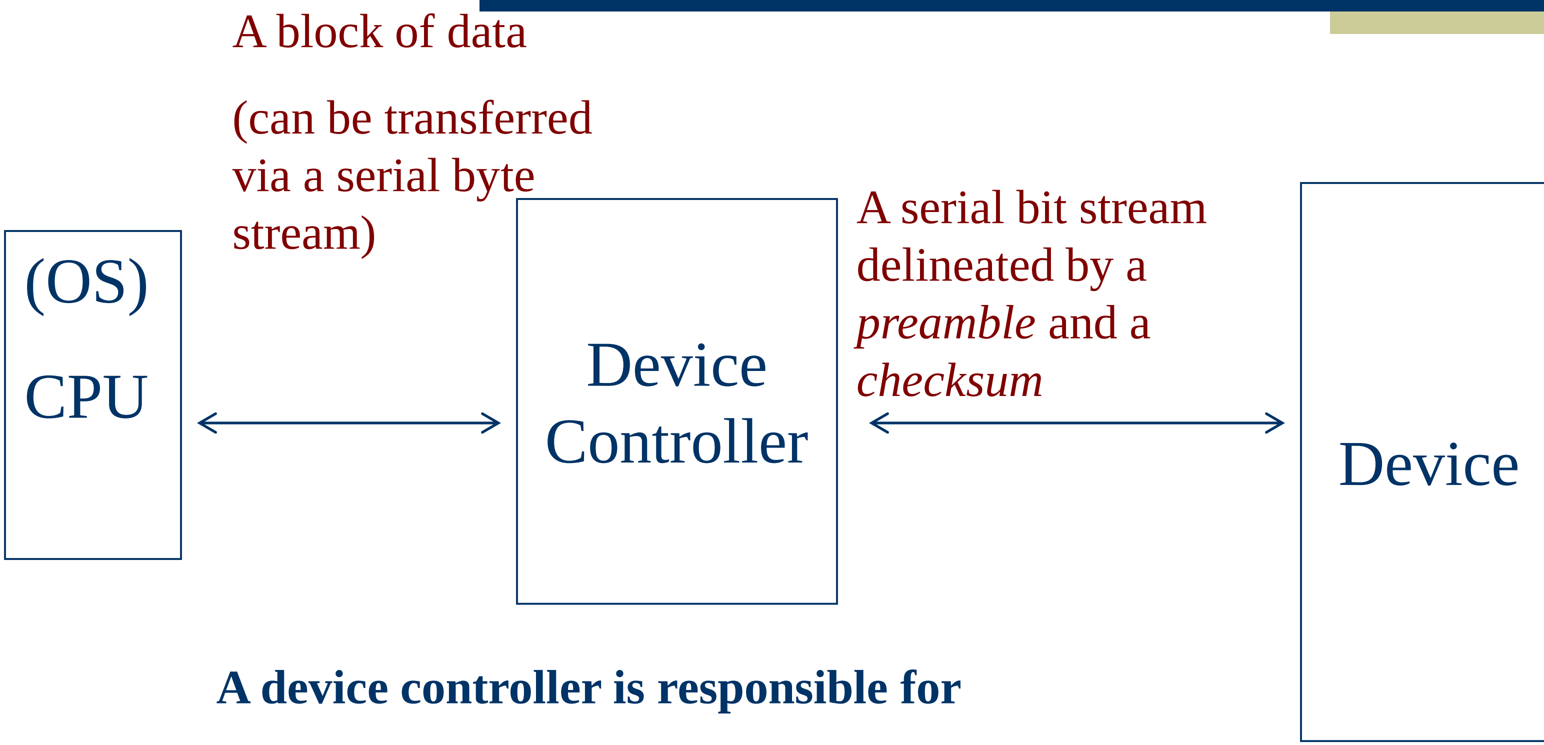


Peripheral Component Interconnection (PCI)

Note: *Integrated Disk Electronics (IDE)* is the least expensive current disk technology. IDE support is usually built into the main board.

Small Computer Systems Interface (SCSI) supports disks, tapes, and CD-ROM drives. While IDE disks provide up to one gigabyte of storage, SCSI disks are available with four to 9 gigabytes of storage.

Interfaces and Responsibilities



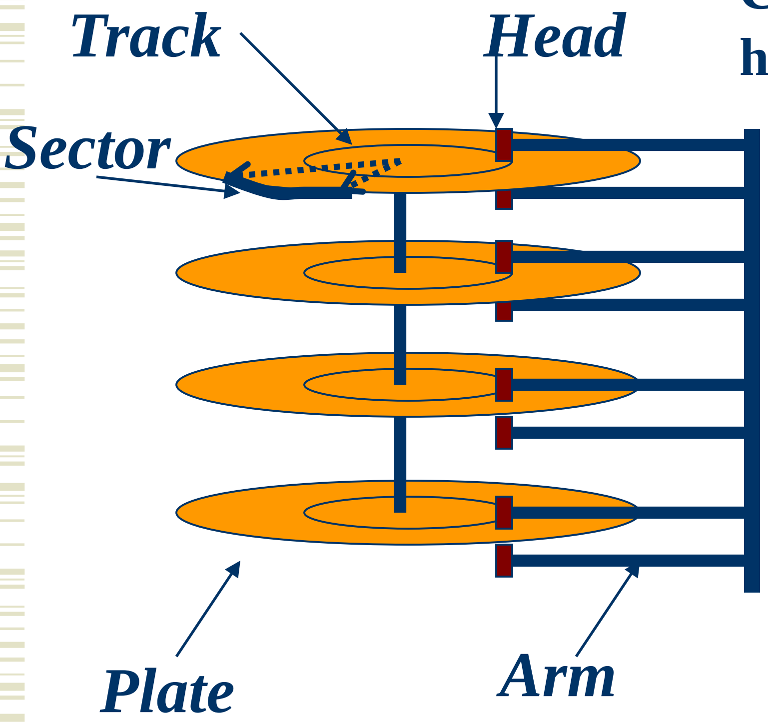
A device controller is responsible for

1. **Ensuring that the I/O is error-free and**
2. **Translating the OS requests into a series of low-level device commands and vice versa**

Memory Access

- There are two aspects to that:
  1. The *control path* aspect—sending actual commands (like *read* or *write*) from the OS to the controller. Typically, this is achieved by writing to the controller registers, which can also be *memory-mapped*
  2. The *data transfer path* aspect—moving data between the memory and the controller is typically achieved via *Direct Memory Access (DMA)*

Disk Hardware

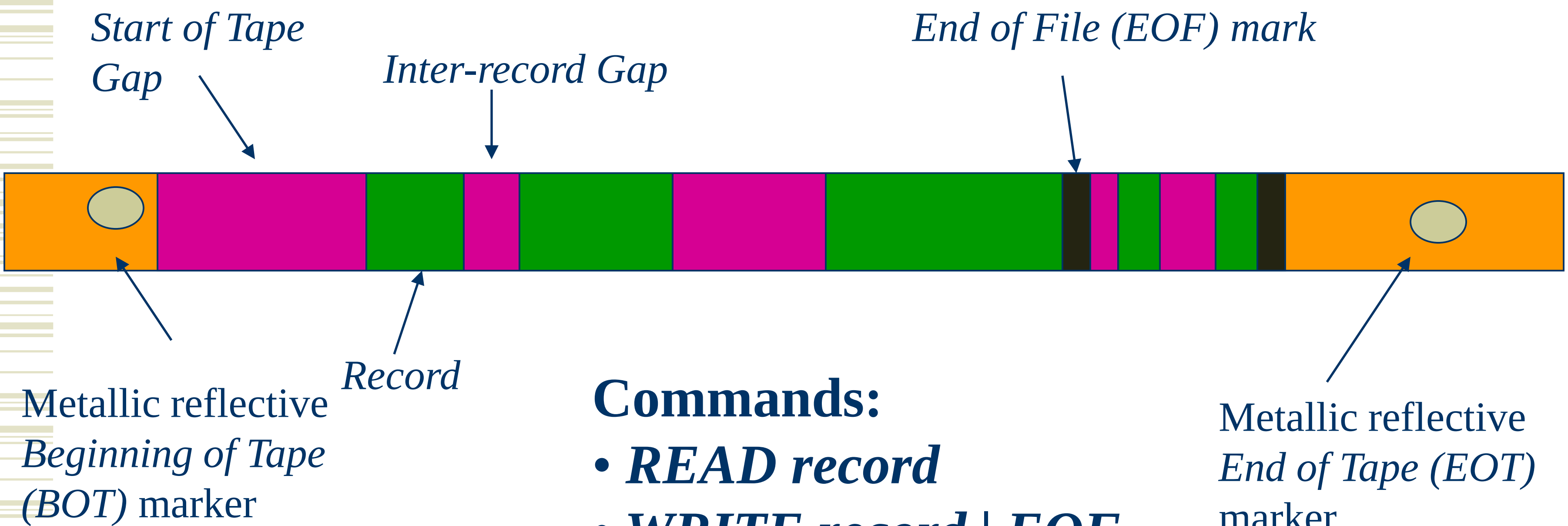


***Cylinder* = set of all *tracks* under the heads in a present position**

Commands:

- ***SEEK cylinder***
- ***READ track, [sector]***
- ***WRITE track, [sector]***
- ***RECALIBRATE***

Magnetic Tape Hardware



Commands:

- ***READ record***
- ***WRITE record | EOF***
- ***ERASE GAP***
- ***REWIND BOT | EOF***

The Control Path

- The OS (more specifically, a device driver) communicates with a controller by writing commands and their parameters in the *controller* registers. For example, a disk controller might have three registers called *command*, *address*, and *start*, so a request to *read* sector *3* of track *5* of cylinder *7* into a memory location *(h)AF67C018* would result in setting the values of the first two registers as follows:

Command Register:

0 0 0 1	0 0 1 1	0 1 0 1	0 0 1 1 1

Address Register:

1 0 1 0	1 1 1 1	0 1 1 0	0 1 1 1
1 1 0 0	0 0 0 0	0 0 0 1	1 0 0 0

and then writing anything (say *0*) to the *start* register

The Control Path (cont.)

- In addition, a *status* register would display the result of the operation after it was finished
- The *status* register could also be used for *polled I/O*, but a typical way to communicate with the disk is via an interrupt

An Example: Memory-Mapped I/O in IBM PC

I/O Controller	I/0 Address	Interrupt Vector
Clock	040-043	8
Keyboard	060-063	9
Secondary RS232	2F8-2FF	11
Hard Disk	320-32F	13
Printer	378-37F	15
Monochrome Display	380-3BF	-
Color Display	3D0-3DF	-
Diskette	3F0-3F7	14
Primary RS232	3F8-3FF	12

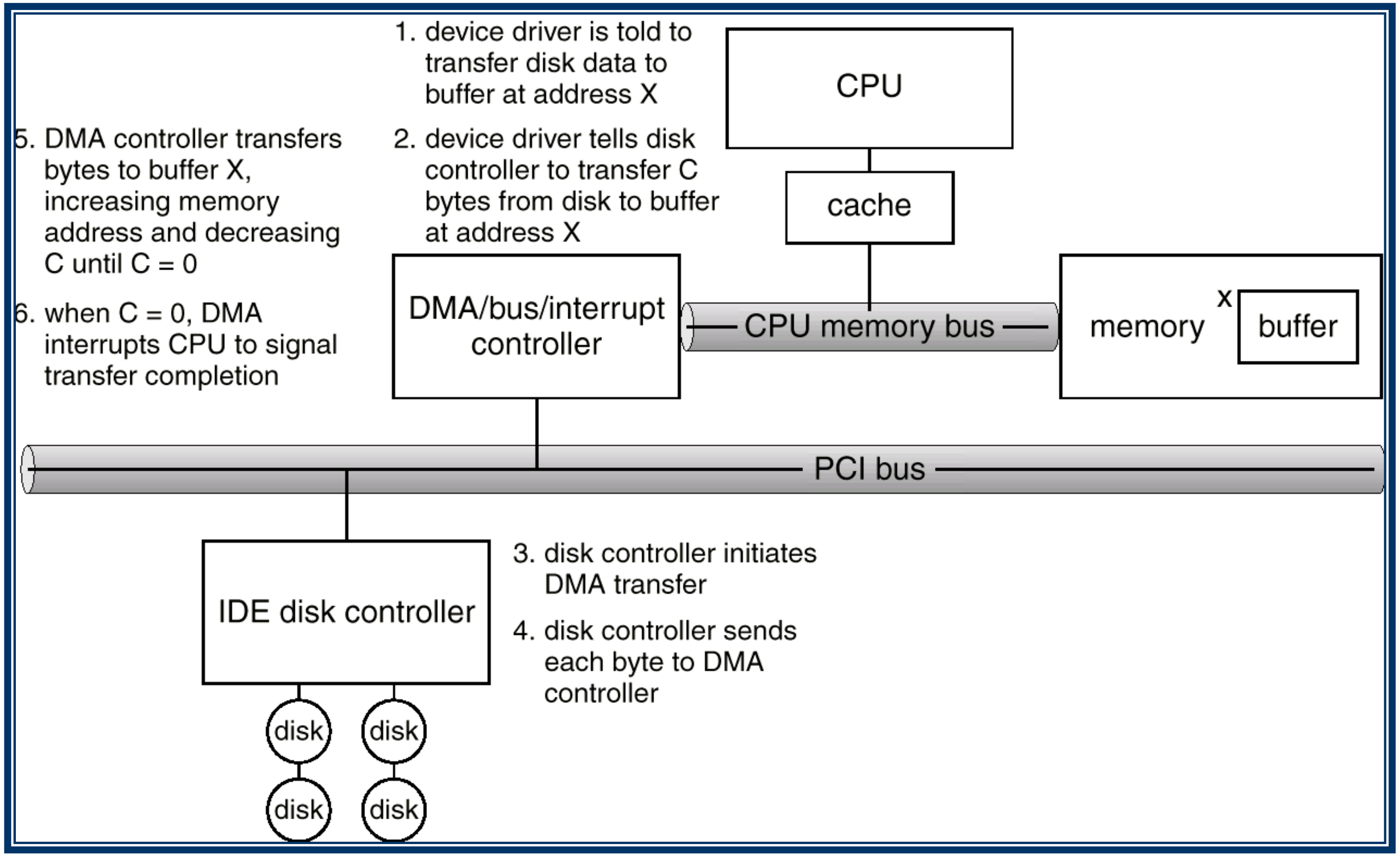
The Data Path

- The controller stores the data (read from or to be written to the device) in its internal buffer—or it would not be able to keep up with the transfer rate
- These data need to be transferred to or from the main memory
- The OS could deal with the controller by reading or writing its data, but interrupting CPU for each byte or at most word of data in a large transfer is very expensive

Direct Memory Access (DMA)

- DMA solves the problem by transferring the data between the main memory and the controller' buffer *without* getting the CPU (and OS) involved beyond ordering a transfer
- A DMA (which could be a separate device or part of the device controller) accesses main memory by *stealing* cycles from CPU

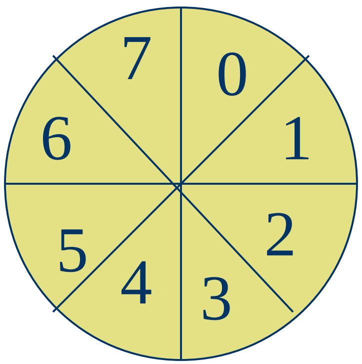
DMA Operation (from the Book)



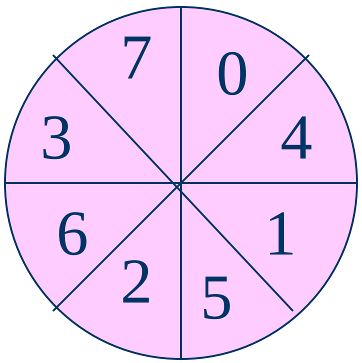
Interleaving Sectors

- While the DMA transfer of a sector's data takes place, a controller may be unable to keep up with reading the next sector (or even two sectors)
- Then, the full disk rotation must take place before the next sector is read
- This could substantially decrease performance, so the disk should be formatted so as to mitigate this problem by skipping adjacent sectors
- Such a practice is called *interleaving*

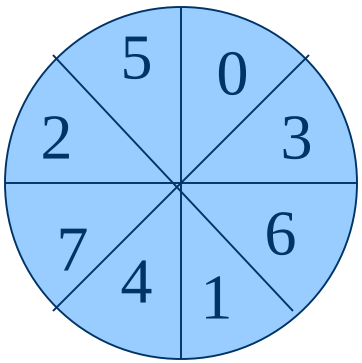
Interleaving Sectors (Example)



No Interleaving



Single Interleaving



Double Interleaving

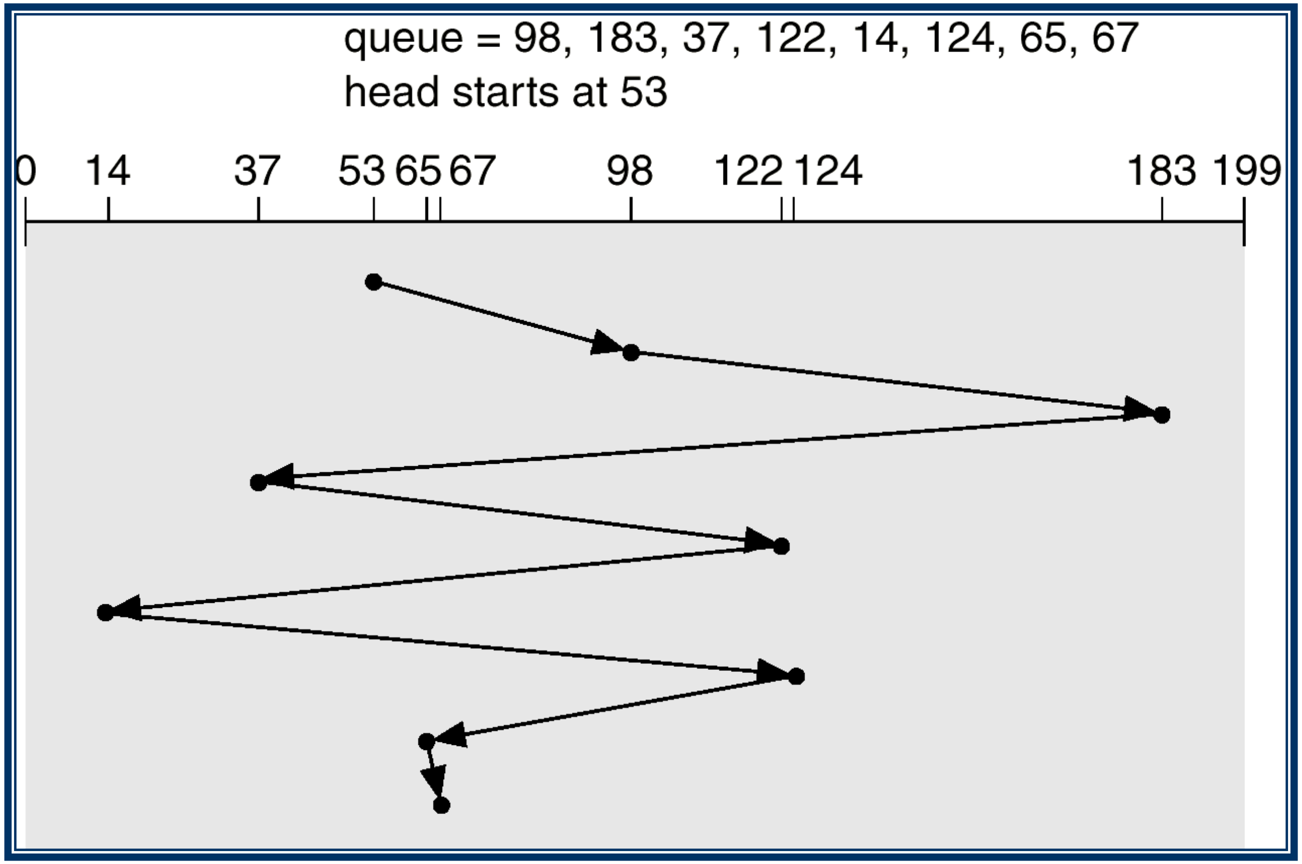
By the way, disk tracks typically have 8 to 32 sectors. The number of sectors is a constant for the whole disk.

Question: Do we waste much space because of the difference in the track size?

Scheduling Disk Requests To Optimize Delay

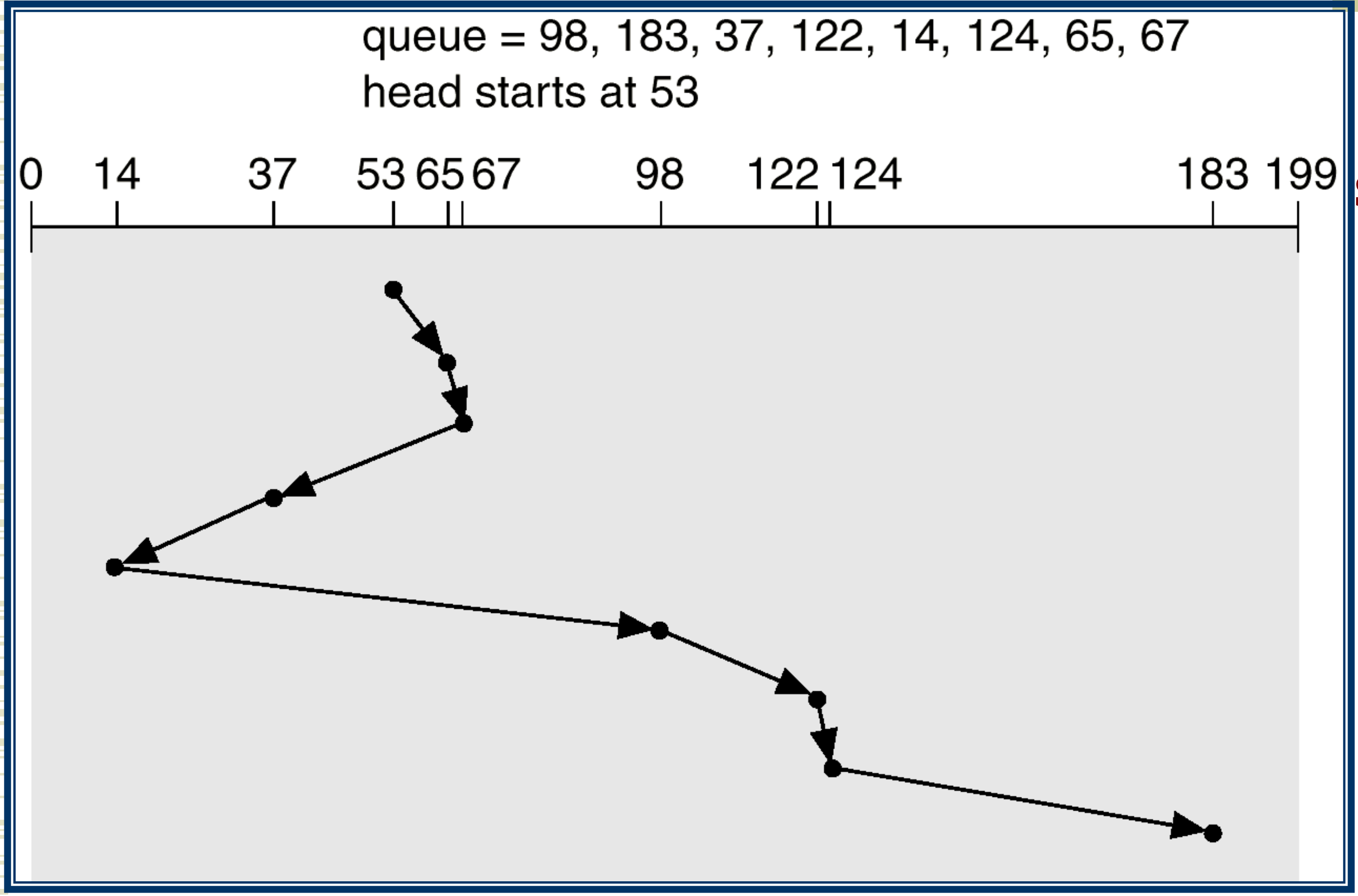
- *First-Come First-Served* algorithm optimizes nothing
- *Shortest Seek Time First (SSTF)* algorithm optimizes seek time, but tends to keep the head around the center, starving the processes that request outer tracks
- *Elevator (SCAN)* algorithm and its derivatives guarantee the upper bound on the total motion

First-Come First-Served



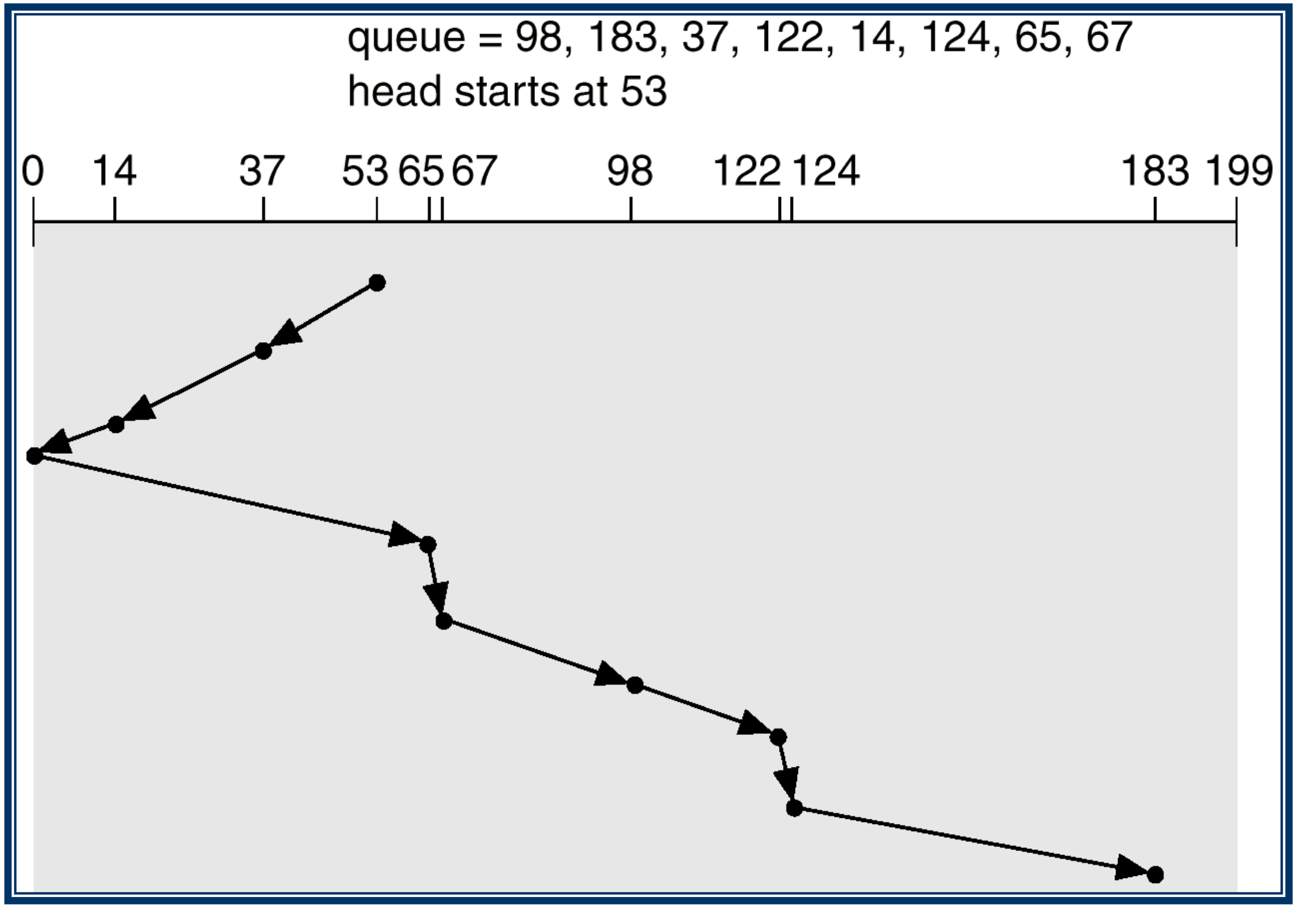
Seek Time: 640

Shortest Seek Time First (SSTF)



Seek Time: 236

Elevator (SCAN)



Seek Time: 236

A Bottleneck

• The performance of CPUs and RAM has been increasing exponentially (in time)

• The performance of the disk drives has been increasing *sub-linearly*

• In 1987, UC Berkeley researchers David Patterson, Garth Gibson, and Randy Katz have invented the *RAID* technology to deal with the problem

Redundant Array of Inexpensive (or Independent) Disks (RAID)

- *RAID* is the technology supporting storage systems' resilience to disk failure through the use of multiple disks and by the use of data distribution and correction techniques
- A *RAID array* is a collection of drives that collectively act as a single storage system, which can tolerate the failure of a drive without losing data, and which can operate independently of one another
- Different *levels* of RAID are defined according to how the data and parity information is distributed among the disks
- Parity information supports error correction

(Example: With 38 disks in the system, using bits 1, 2, 4, 8, 16, and 32 for Hamming code, the system can recover from a single disk failure)

Clocks

There are two types:

- Simpler clocks follow the AC frequency of the power line and cause an interrupt at 50-60 Hz
- More useful clocks use a crystal oscillator, which generates signals of high accuracy in the range of 5 to 100 MHz
  - These signals are fed into the *counter*, which counts down to zero
  - At zero count, the counter causes an interrupt (a *tick*)
  - Also at zero count (and initially), the counter is set to the value of the settable *holding register.* In *one-shot* mode, that happens automatically; in *square-wave* mode, the clock stops at the interrupt and waits for a reset

Clock Drivers

- The *clock driver* is responsible for
  - Maintaining the time of day
  - Preempting processes
  - Accounting for resource use
  - Handling *Alarm* calls
  - Profiling, monitoring, and gathering statistics

Terminals

- There are three basic types:
  1. Memory-mapped terminals
  2. Serial (synchronous) terminals connected over the RS-232 interface
  3. Asynchronous terminals (although they have been pretty much replaced by PCs now), which support messages of various length
- *Keyboard* and *display* are two separate devices, and are treated as such by the operating system

Musical Instrument Digital Interface (MIDI) Devices

- In August of 1983, music manufacturers agreed on a document that is called "MIDI 1.0 Specification".
- The MIDI protocol supports interworking of keyboards, synthesizers, and sequencers built by different manufacturers
- It is important to remember that MIDI transmits commands--not audio signals

Physical Interface

- MIDI uses serial interface. The *serial interface* was chosen by MIDI manufacturers because it is less expensive than parallel interface.
- The speed of a MIDI serial interface is 31,250 bits per second.
- There are 10 bits needed for every MIDI digital word or 3125 messages per second.

MIDI Data examples

- Note on/off (on a particular channel)
- Additional effects (like sustain pedal)
- Dynamic range of the note

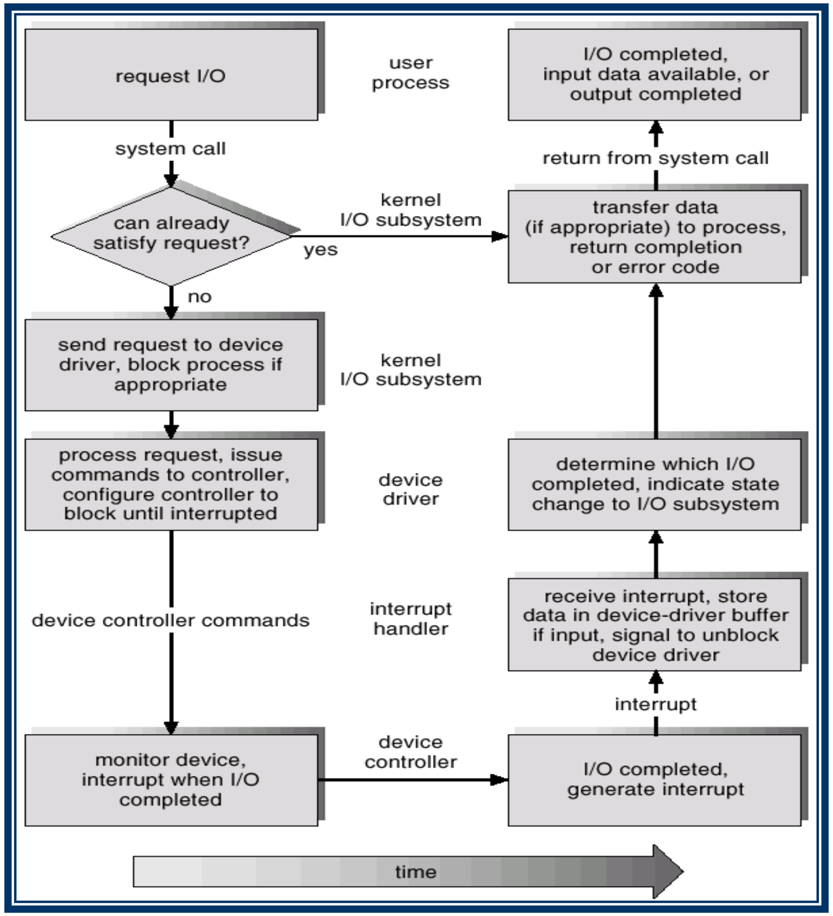
90	3C	72

Note ON, Channel 1   Middle C   Velocity: 72

Principles of the I/O Software Design

1. Device independence
   - At the user-level, the API is as abstract as possible (e.g., `sort <input >output`)
   - All device-specific code goes into *device drivers*
2. Uniform Naming (whence *Universal Resource Identifier [URI]* came)

Life Cycle of an I/O Request



from the Book