CIS 520, *Operating Systems Concepts*

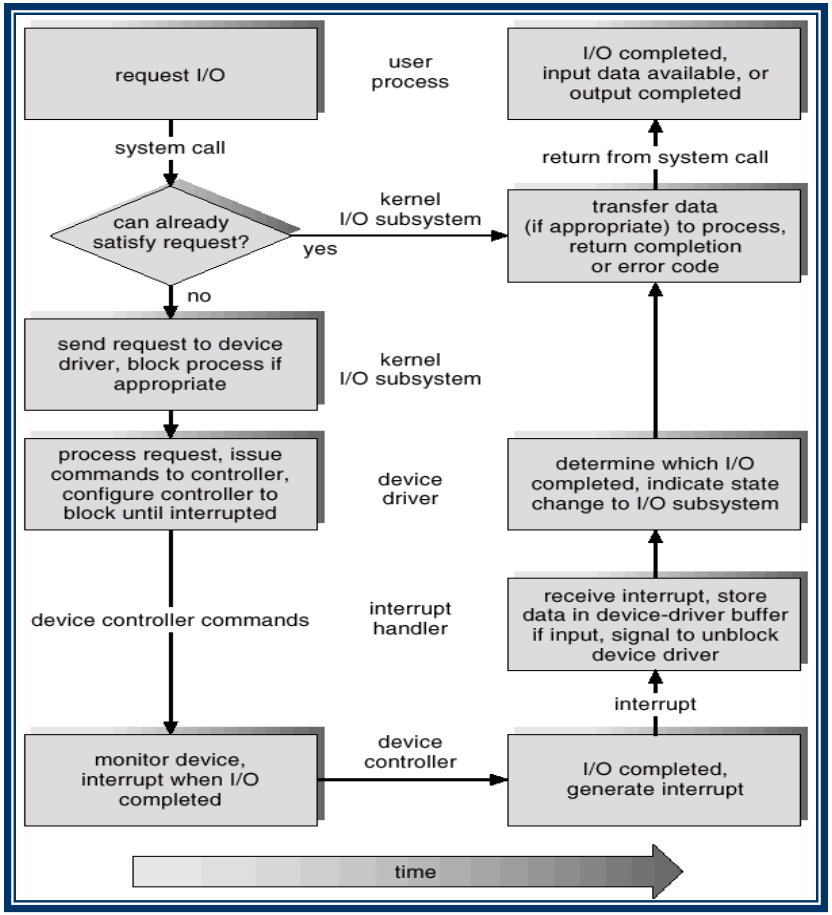
Lecture 8

Input/Output

Outline

- The I/O life cycle
- Types of I/O devices
- I/O Controllers
- Direct Memory Access (DMA)
- I/O Software Design
  - Principles and layers
  - Device drivers
- Device review
  - Terminals (monitors), keyboards, and mice
  - Magnetic disks and their scheduling algorithms, RAID technology
  - Magnetic tape
  - Solid state storage
  - Clocks
  - MIDI devices

Life Cycle of an I/O Request



from the Book

I/O Devices

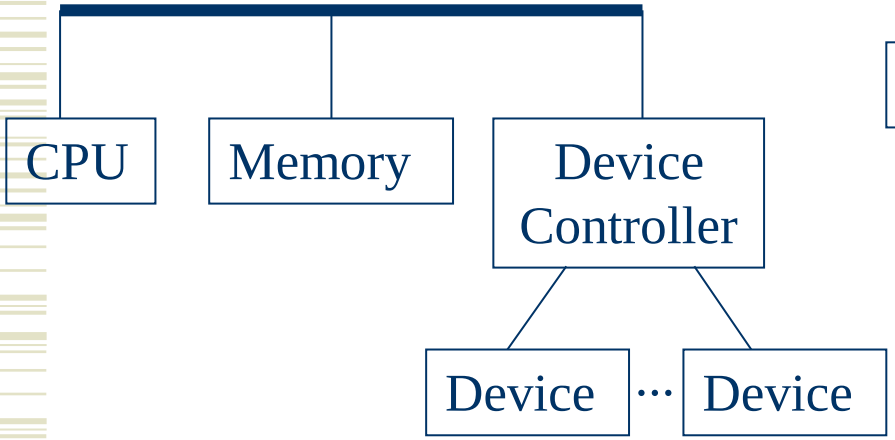
- The are two categories of the I/O devices:
  - *Block Devices* store and deliver information in blocks of bytes
    - *Disk* uses fixed-size blocks
    - *Tape* and *asynchronous terminals* use variable size blocks
  - *Character Devices* (e.g., *teletupe terminals, some printers, MIDI devices)* store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

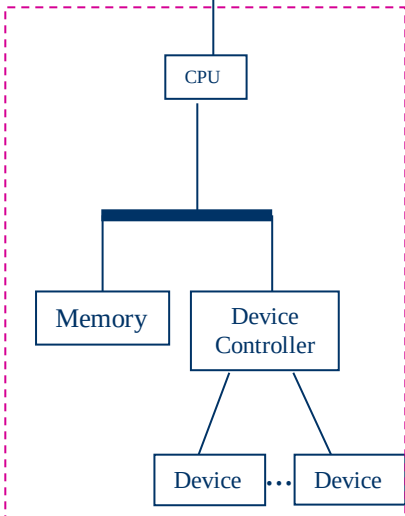
And then there are devices (e.g., *computer clocks*, *telemetric devices*, *mice)* that don't belong to either category.

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*

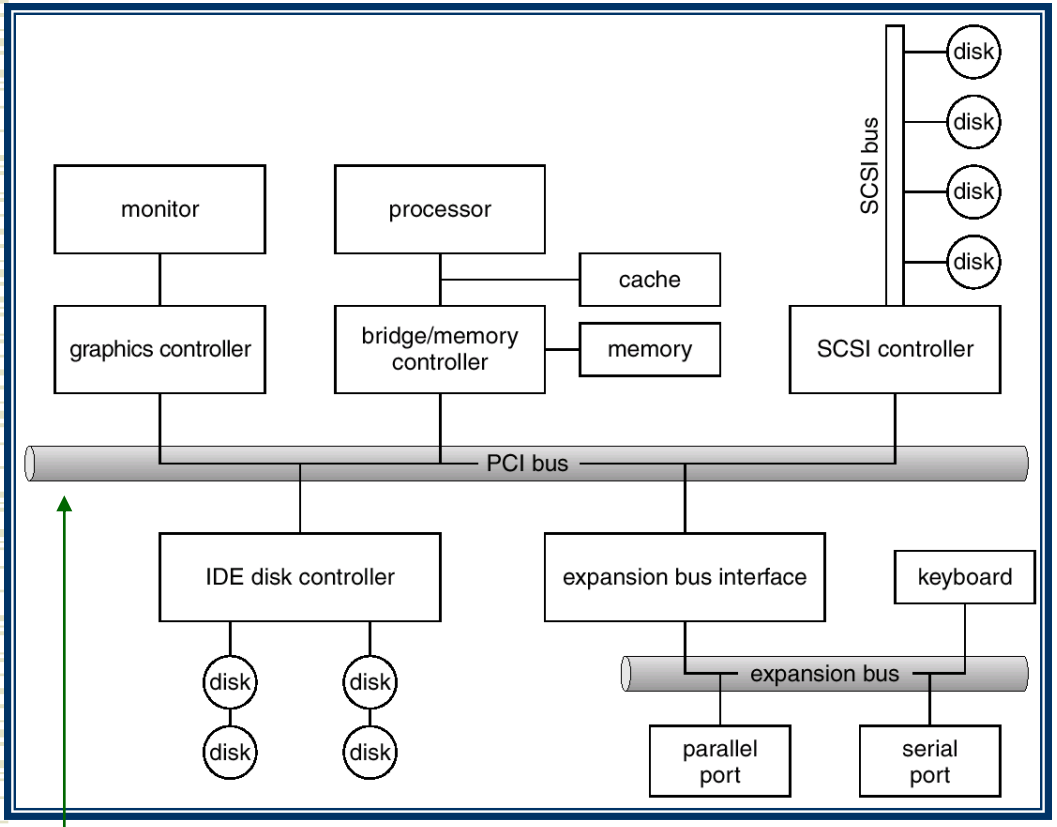
Device Controllers and I/O Channels (cont.)





I/O Channel

A Real-Life Example (from the Book)

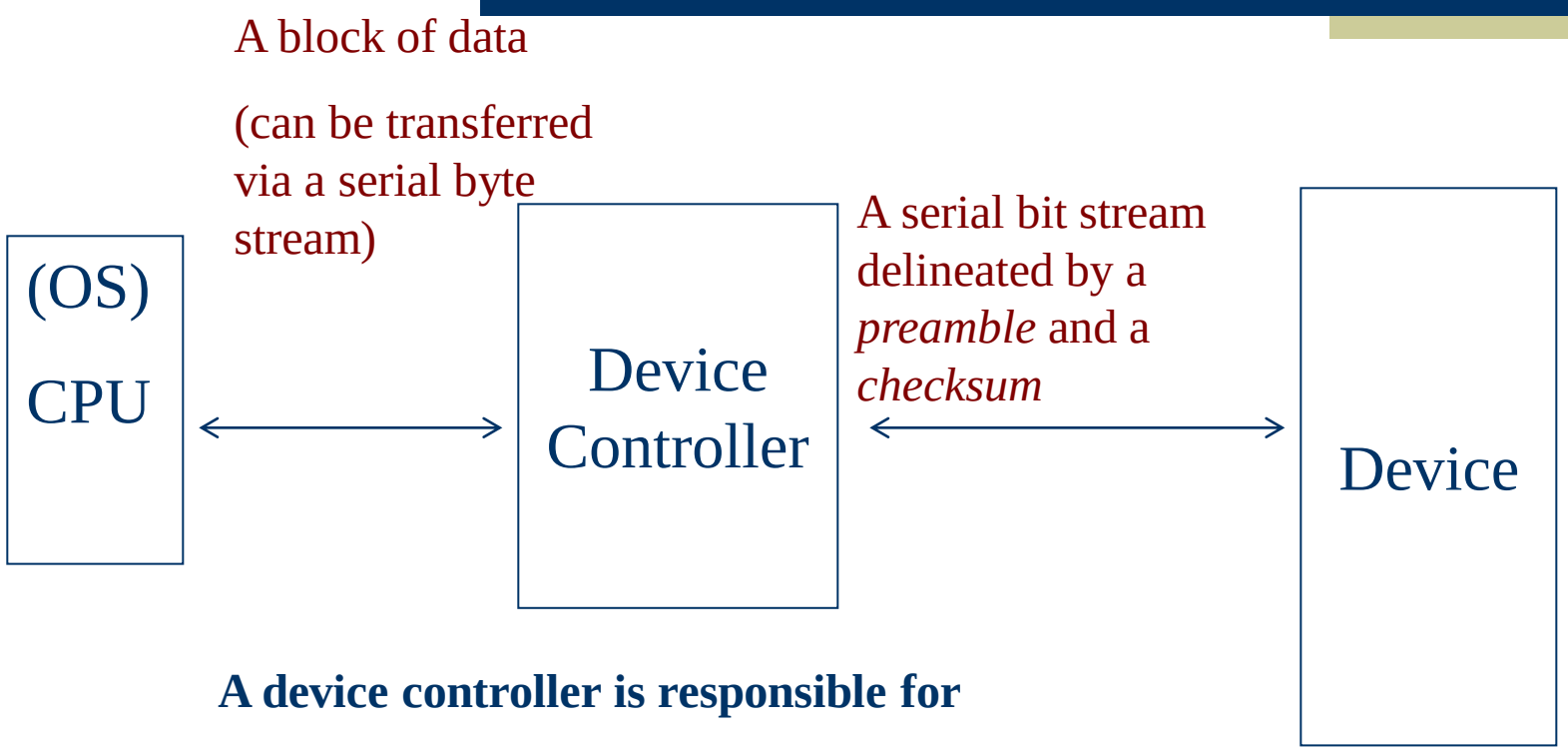


Peripheral Component Interconnection (PCI)

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

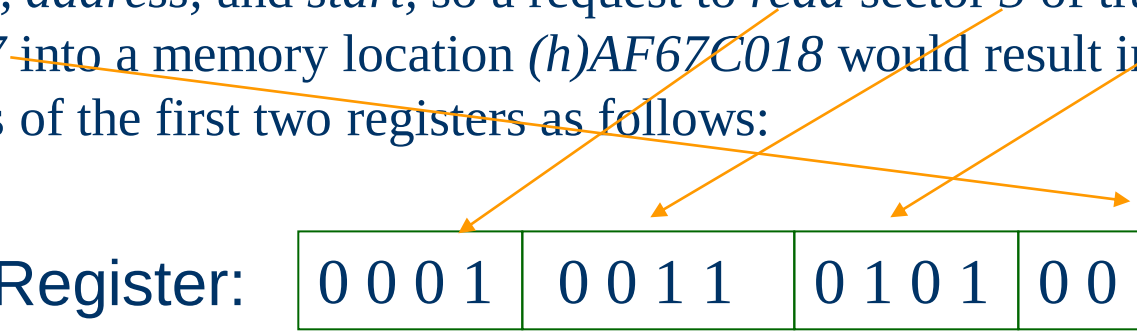
1. **Ensuring that the I/O is error-free and**
2. **Translating the OS requests into a series of low-level device commands and vice versa**

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)*

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

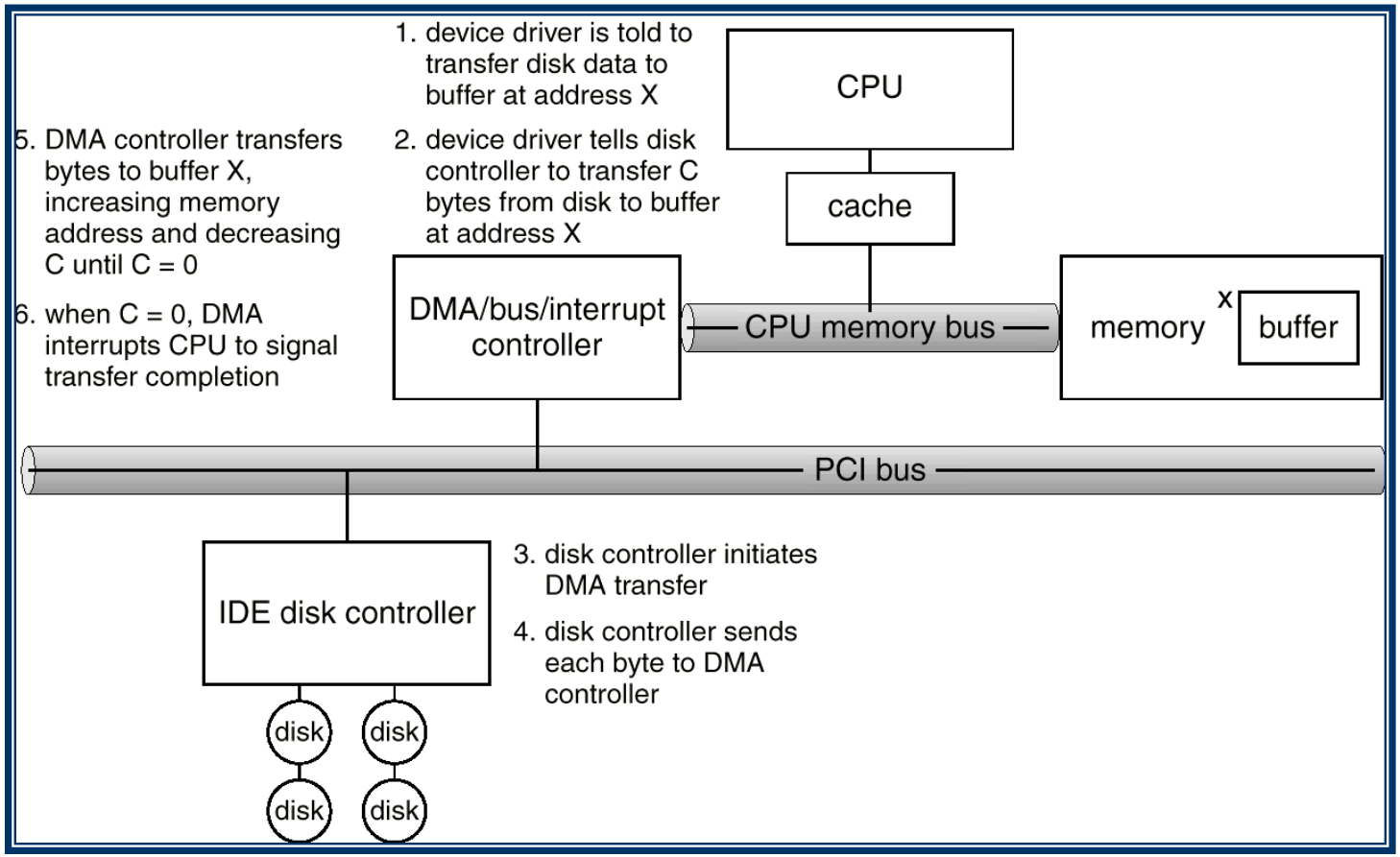
The Data Path

- 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

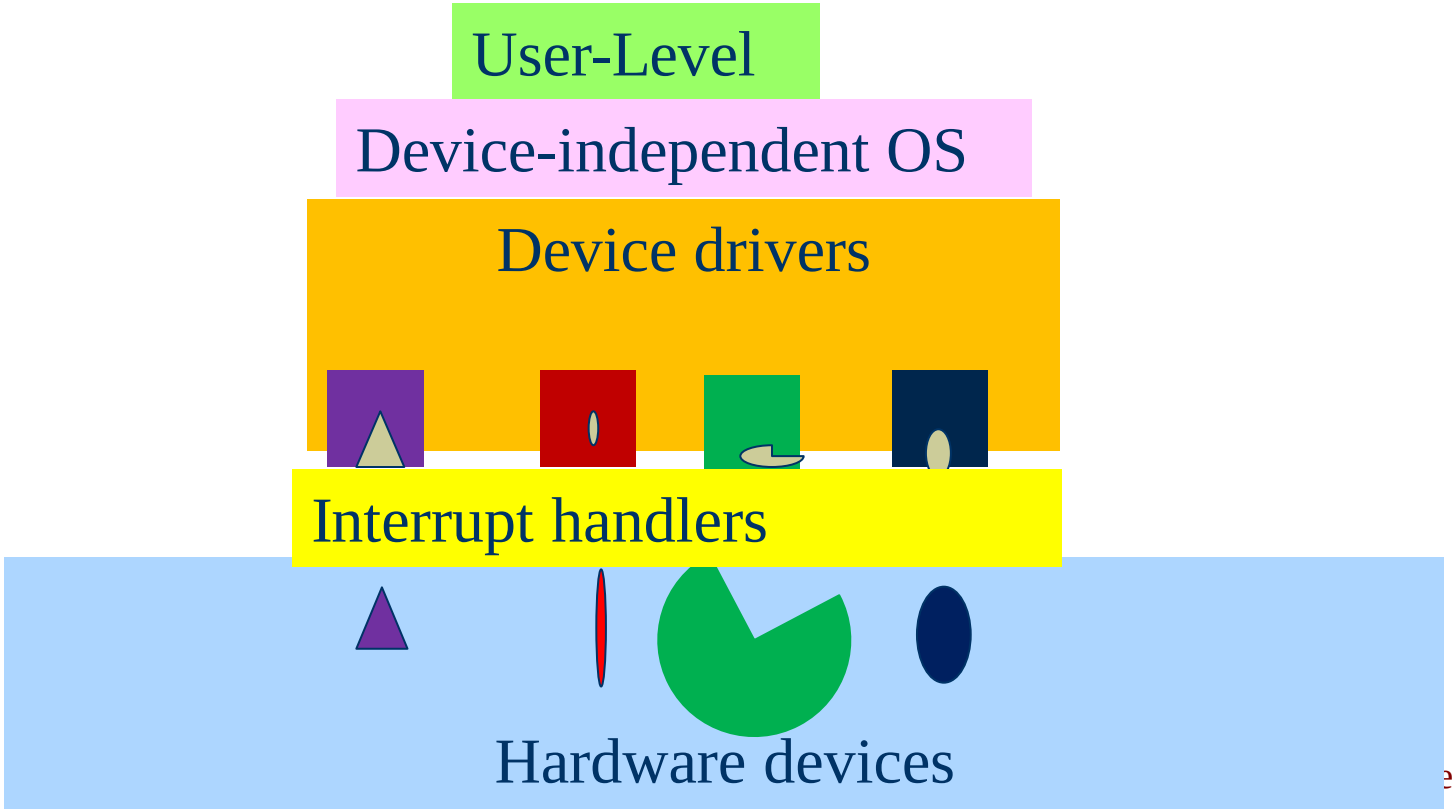
DMA Operation (from the Book)



Principles of I/O Software Design

- Device independence
  - All devices of the same type are handled by the same code
  - At the user-level, the API is as abstract as possible (e.g., `sort <input >output`)
  - All device-specific code goes into *device drivers*
- Uniform naming
- Error handling
- Blocking (synchronous vs. asynchronous handling)
- Buffering

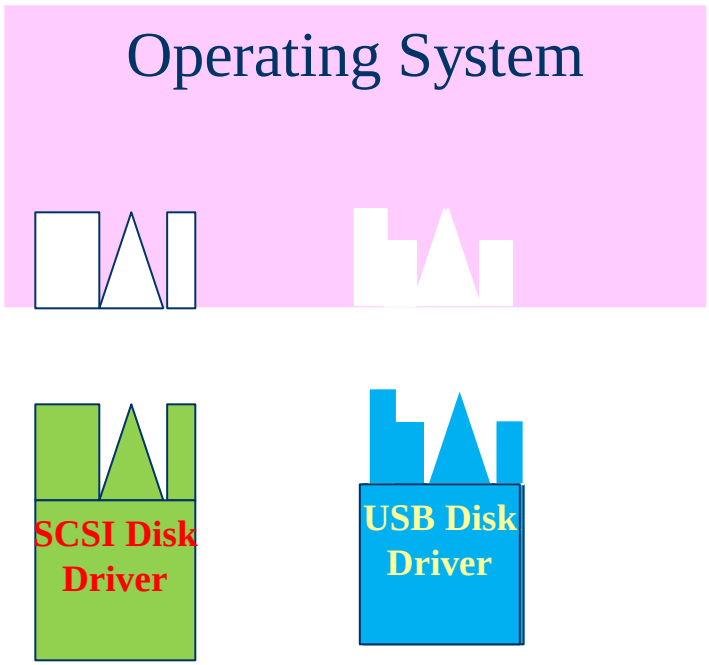
I/O Software Layers

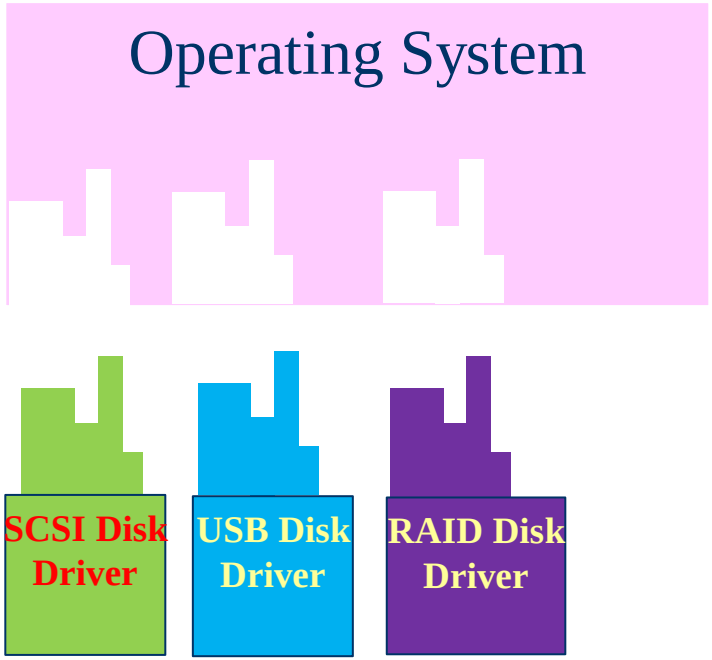


Device-Independent I/O Software

- Uniform interface for device drivers
- Buffering
- Error Reporting
- Allocation and release of the devices (e.g., *mount*)
- Maintaining a device-independent block size

Uniform Interface for Device Drivers (after A. Tanenbaum)

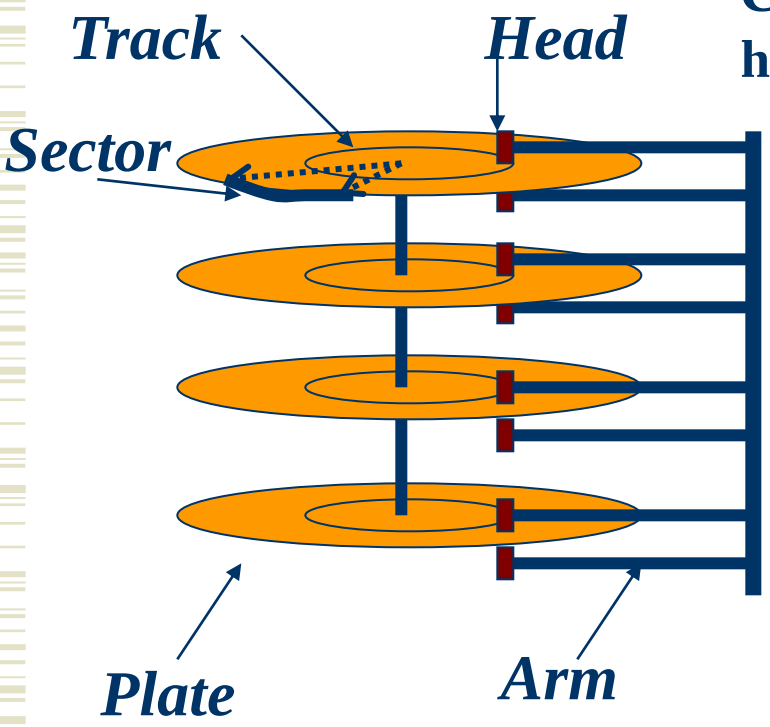




Device Drivers

- Most often still implemented as part of the OS kernel (inflexible)
- Can be partly implemented in the user space in micro-kernel systems (such as *MINIX*-- https://www.operating-system.org/betriebssystem/_english/bs-minix.htm)
- Can be plugged into some systems (a potential security risk)

Disk Hardware

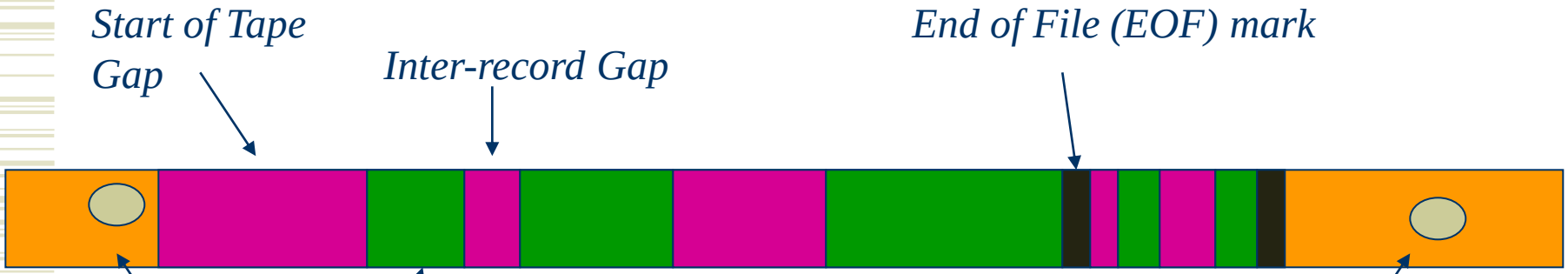


***Cylinder* = set of all *tracks* under the heads in a present position**

Commands:

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

Magnetic Tape Hardware



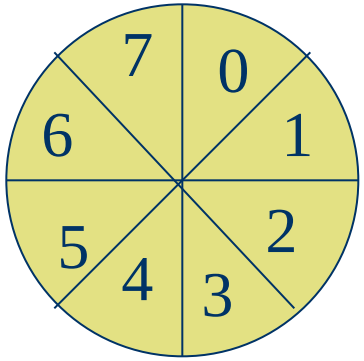
Commands:

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

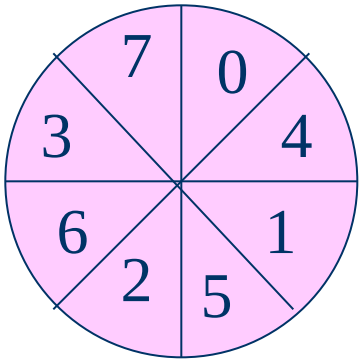
Interleaving Sectors

- 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*

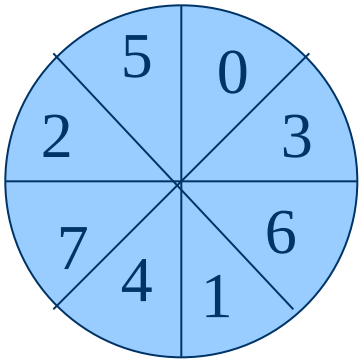
Interleaving Sectors (Example)



No Interleaving



Single Interleaving



Double Interleaving

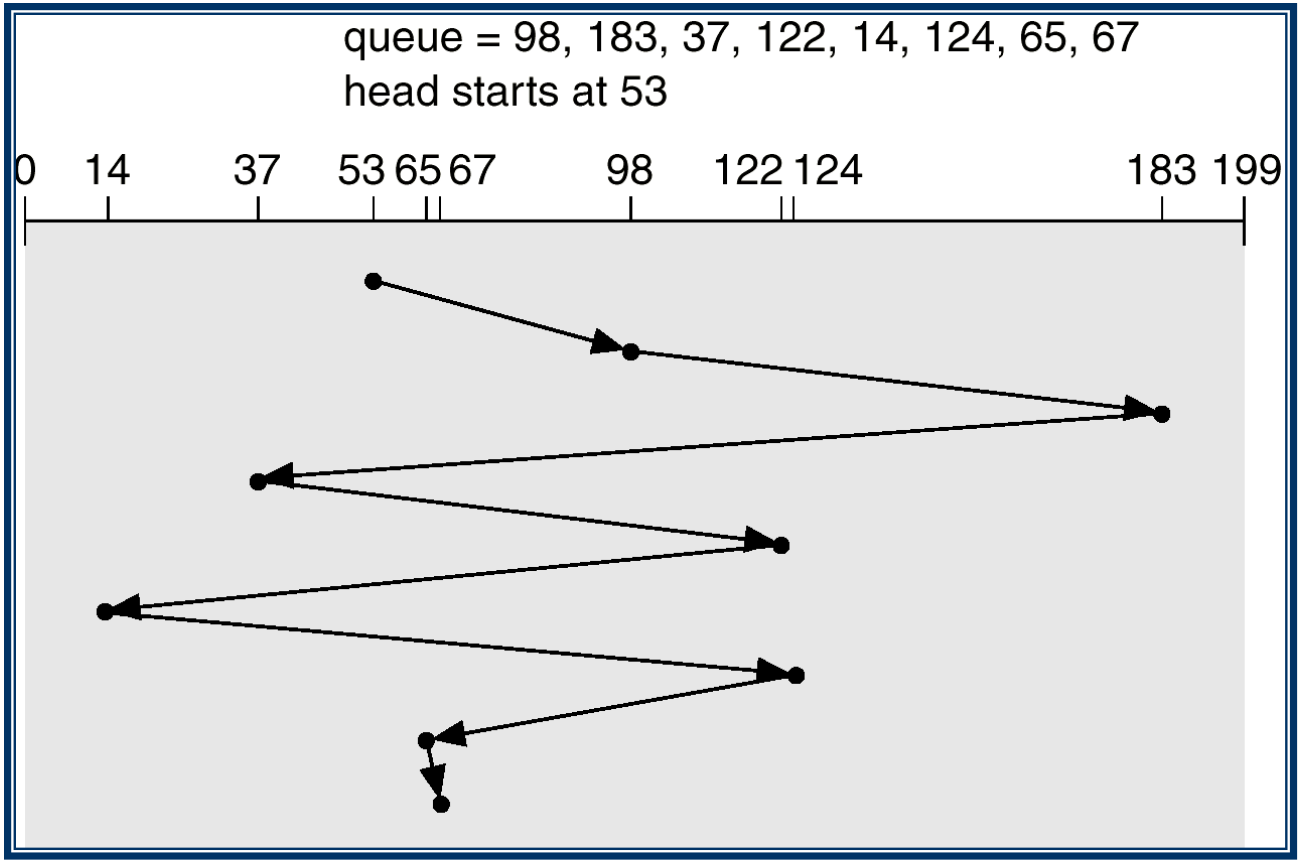
By the way, disk tracks typically have 8 to 32 sectors. The number of sectors is a constant for the whole disk.

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

Scheduling Disk Requests To Optimize Delay

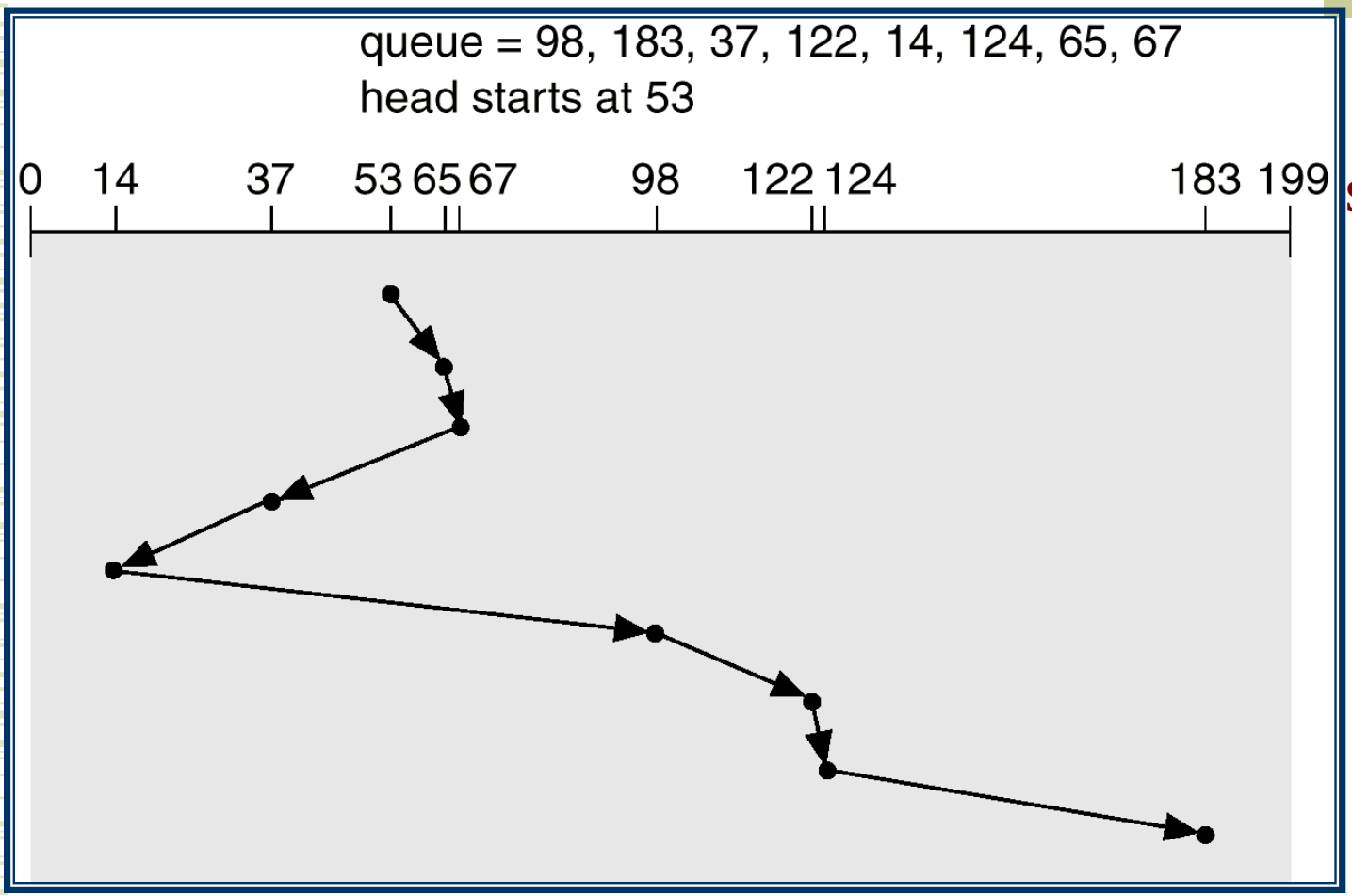
- *First-Come First-Served* algorithm optimizes nothing
- *Shortest Seek Time First (SSTF)* algorithm optimizes seek time, but tends to keep the head around the center, starving the processes that request outer tracks
- *Elevator (SCAN)* algorithm and its derivatives guarantee the upper bound on the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

First-Come First-Served



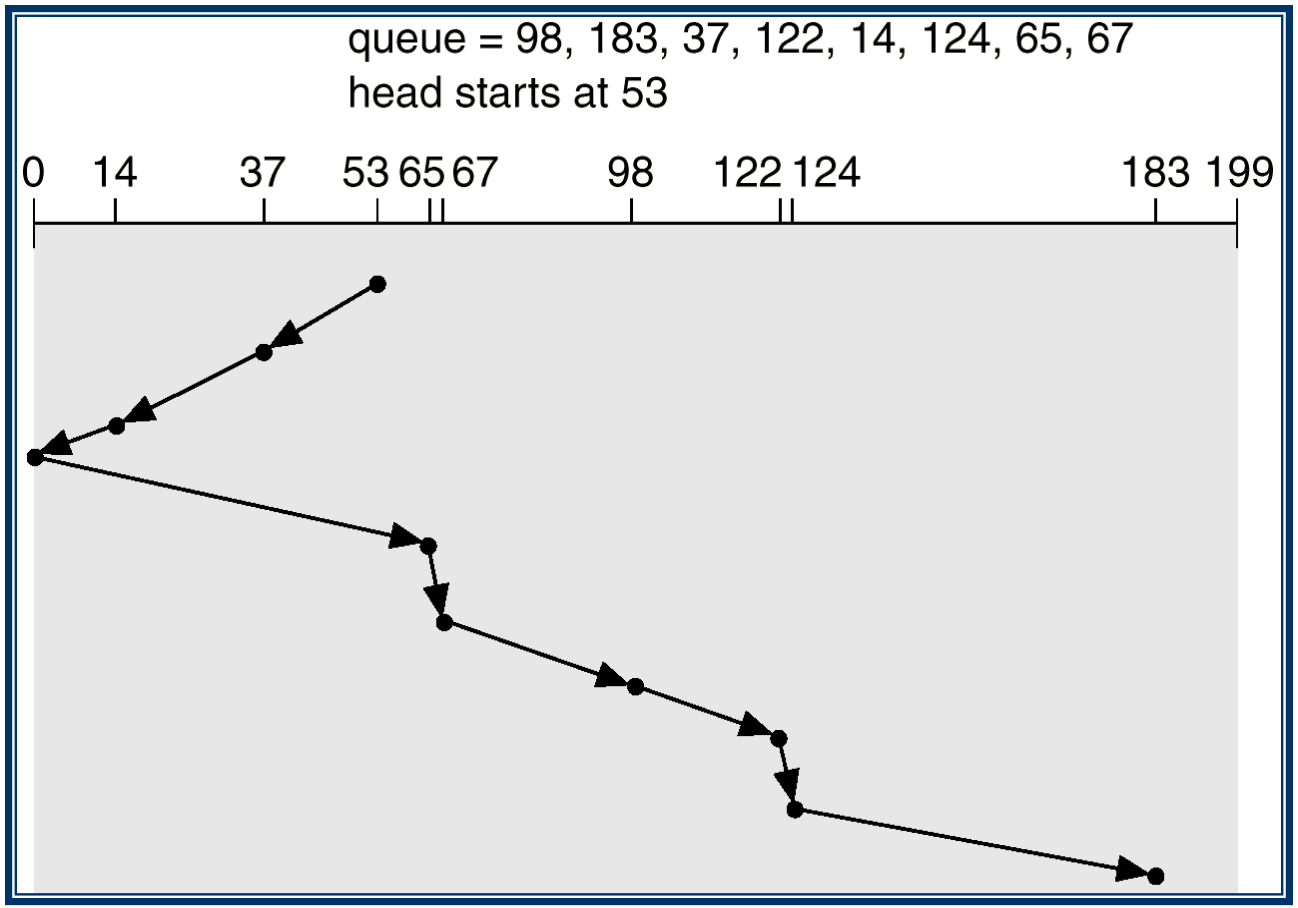
Seek Time: 640

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*

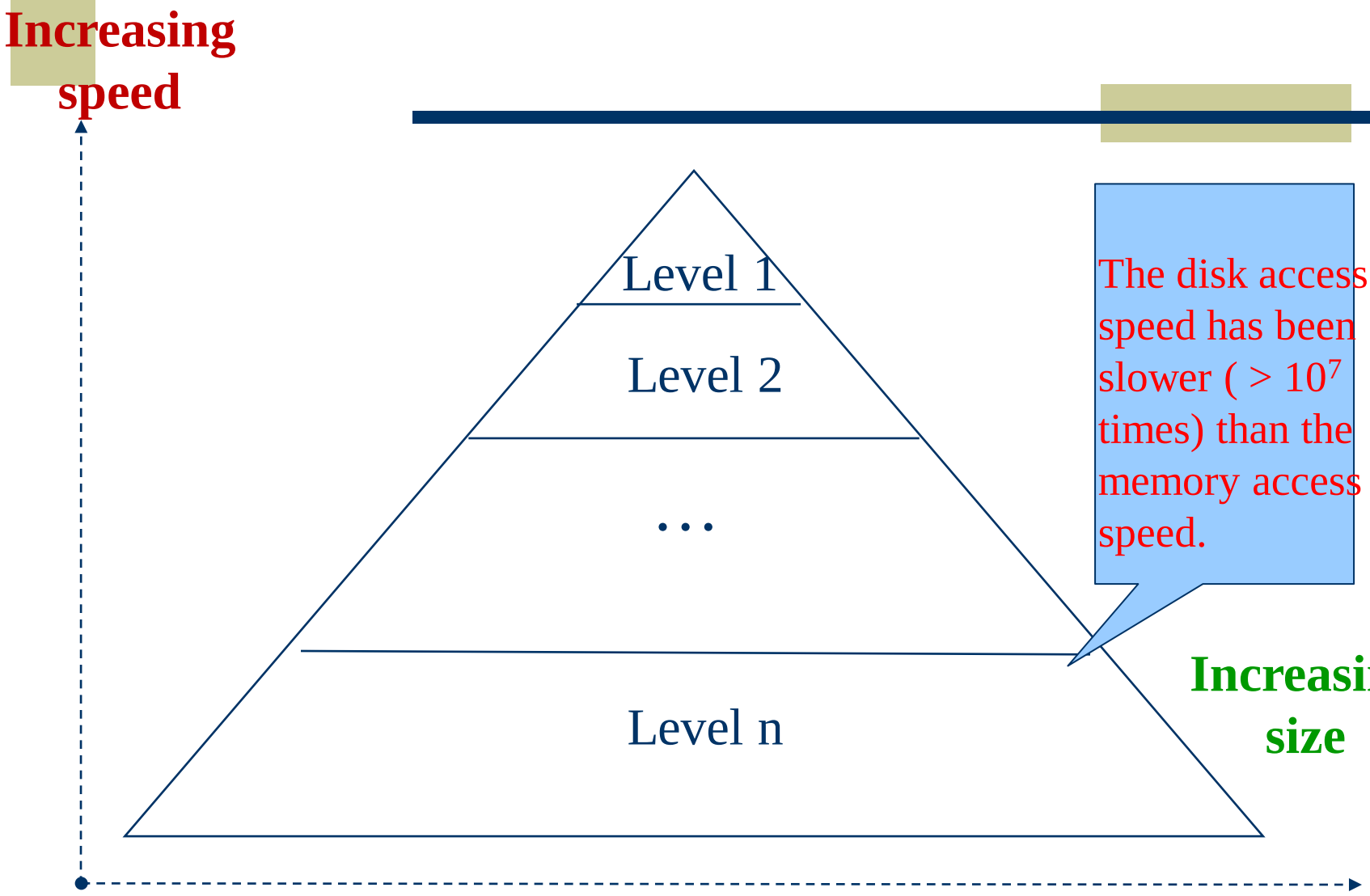
• In 1987, UC Berkeley researchers David Patterson, Garth Gibson, and Randy Katz have invented the *RAID* technology to deal with the problem

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)

Storage as memory hierarchy



Speeding things up—Solid State Storage

- Much more suited to sequential access than to random access
- Faster (from 100 to 1,000 times) and sturdier than hard discs, but more expensive
- Becoming an option as prices are dropping

A comparison of storage technologies

	Relative Access Time	Relative Cost	Retention Time
SRAM	$\sim 10^{-7}$	$\sim 10^{4}$	the duration when it is powered
DRAM	$\sim 10^{-5}$	$\sim 10^{2}$	<< 1 second
Flash	$\sim 10^{-2}$	~10	years
Magnetic Disk	1	1	years

SRAM: Static Random Access Memory (used for *cache*)
DRAM: Dynamic Random Access Memory (used for main memory)

Dynamic RAM, Static Ram, Flash

- In DRAM data are stored as electric charge in capacitors. Since the charge leaks over time, the capacitors need to be refreshed regularly (hence *dynamic).*
- In SRAM data are stored in transistors rather than in capacitors. There is no need for refreshing, which makes SRAM faster than DRAM.
- Both SRAM and DRAM are volatile—they need power
- Flash memory is not volatile.

EEPROM vs. EPROM

- Flash memory is a kind of *Electrically-Erasable Programmable Read-Only Memory* (*EEPROM*).
  - The “Electrically-erasable” feature is significant because the earlier technology erasable programmable read only memory (EPROM) required exposure to ultraviolet light (for over ten minutes) to erase content.
  - To programming EPROM one needed a special device.
- The data are kept in floating-gate islands, which can retain electric charge for a long period of time (i.e., years).

Flash Memory Types

- A relatively new flash memory technology was invented by **Fujio Masuoka** when he worked at Toshiba in the 1980s.
- It was named *flash* because of its capability of erasing a big chunk of memory fast—*in a flash*!
- Two types were invented in 1984 and 1987, respectively.
  1. *NOR flash* – fast (at least faster than the hard disk), randomly addressable to a given byte. Its storage density is limited, however.
  2. NAND flash removed that limitation, and it also allowed random access in the units larger than a byte
- The NAND flash has made a splash in consumer electronics, and it is used much more widely than the NOR flash—in digital cameras, portable music players, and smart phones.

Where NAND Flash is going

- NAND flash is increasingly placed between the SRAM and the hard disk in the memory hierarchy (**A NAND flash 1.000 times faster than any disk in random operations!).**
- The drive emulates a hard-disk drive to help integrating with the existing systems.
- NAND flash is still more expensive than hard disc drives (in cost per byte) but it is useful when the access is random because of diversified workloads (think Cloud Computing).

Major Problems to Overcome

- A *write* operation over the existing content requires that this content be erased first. (This makes write operations much slower than read operations.)
- *Erase* operations are performed on a **block** basis, while write operations are performed on a **page** basis.

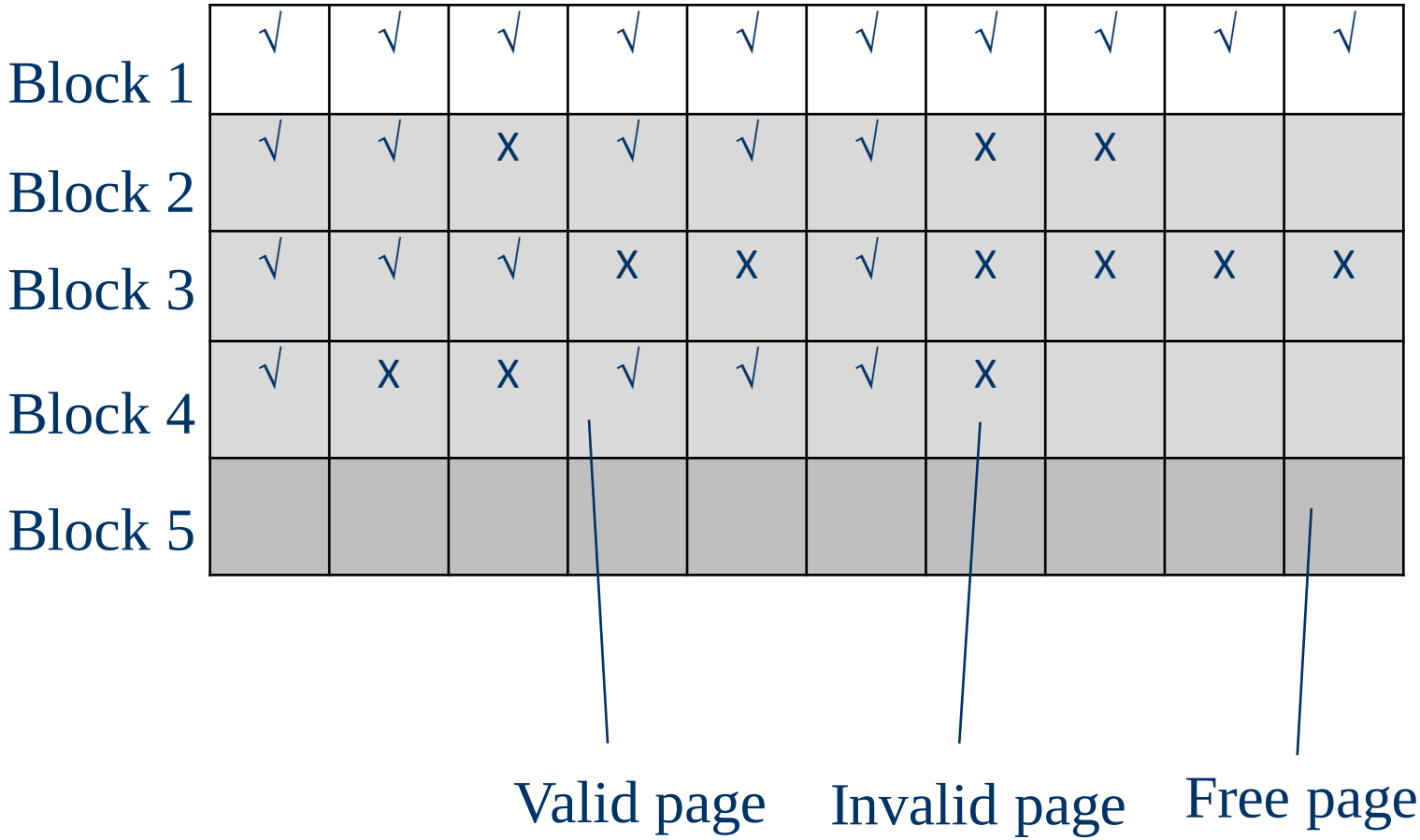
  Pages are grouped together into **blocks**. A typical block contains 32, 64, or 128 pages. (It tends to be larger the block size in a file system.) Page sizes vary. A page may contain, for example, 512, 2,048 or 4,096 bytes.

- Memory cells wear out after a limited number of write-erase cycles

How do we deal with this now

- Direct updating of the content of a page causes high latency because the need for reading, erasing, and reprogramming the entire block.
- To eliminate this, there is the practice of *relocate-on-write* (or *out-of-place write*): A free page is written with the latest data, while the old page is marked *invalid*.
- Thus write performance is improved at the expense of an ever-growing number of invalid pages. If not reclaimed, the invalid pages deplete the storage space quickly. To reclaim the storage, *garbage collection is necessary*.

A State of the NAND flash



Garbage Collection

- For purely sequential write operations, garbage collection is straightforward: Blocks can be invalidated and reclaimed one-by-one as data are written page-by-page. No extra write-erase operations are incurred.
- For random *write* operations, the situation is much more complex: an algorithm cold
  1. maximize the number of reclaimed pages, or
  2. minimize the number of additional read- and write operations.

The effectiveness of a garbage collection algorithm depends on the degree of write amplification that it incurs.

Techniques to Reduce the Write Amplification and Increase the Lifetime of a NAND Flash

.

- Over-provisioning: limiting the user address space to a fraction of the raw memory capacity. (The method is effective and does not depend on special support from the operating system (or the file system).
- A special *ATA* command: informs the underlying storage what data are deleted. The command makes a great difference in the case of NAND flash as it saves time that would be otherwise wasted during garbage collection.
- To extend a NAND flash's lifetime, a practice of "wear leveling" is used to spread write-erase operations as evenly as possible.

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 (Keyboard + Display)

- 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

Mouse

- Can be a *trackball* or *optical*
- When it has moved a *configurable* minimum distance (typically 1mm) or when a button is “clicked,” it outputs
  - The position change (Δx, Δy)
  - The status of all buttons (double-click including)

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