Applied Operating System

Chapter 4: Input/output Management

Prepared By:
Amit K. Shrivastava
Asst. Professor
Nepal College Of Information Technology

I/O Sub- Systems

Concepts

▪ Management of I/O devices is a very important part of the operating system - so important and so varied that entire I/O subsystems are devoted to its operation. (Consider the range of devices on a modern computer, from mice, keyboards, disk drives, display adapters, USB devices, network connections, audio I/O, printers, special devices for the handicapped, and many special-purpose peripherals.)

▪I/O Subsystems must contend with two trends: (1) The gravitation towards standard interfaces for a wide range of devices, making it easier to add newly developed devices to existing systems, and (2) the development of entirely new types of devices, for which the existing standard interfaces are not always easy to apply.

▪Device drivers are modules that can be plugged into an OS to handle a particular device or category of similar devices.

Application I/O Interface

▪ User application access to a wide variety of different devices is accomplished through layering, and through encapsulating all of the device-specific code into ***device drivers***, while application layers are presented with a common interface for all (or at least large general categories of) devices.

Application I/O Interface(contd…)

- 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: A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.
  - Sequential or random-access: A sequential device transfers data in a fixed order that is determined by the device, whereas the user of random-access device can instruct the device to seek to any of the available data storage locations.
  - Synchronous or asynchronous: A synchronous device is one that performs data transfers with predictable response time. An asynchronous device exhibits irregular response time
  - Sharable or dedicated: A shareable device can be used concurrently by several processes or threads; a dedicated device cannot be.
  - Speed of operation: Device speeds range from few bytes to few gigabytes per second.
  - Read-write, read only, or write only: Some devices perform both input and output, whereas others support only one data direction.

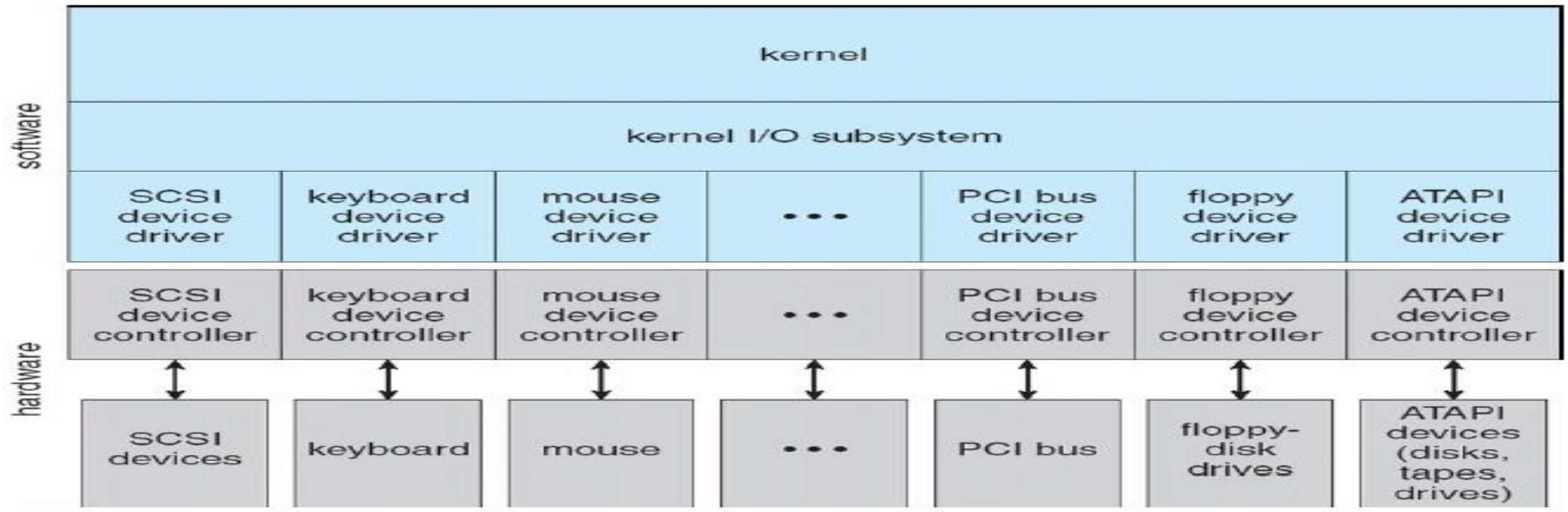
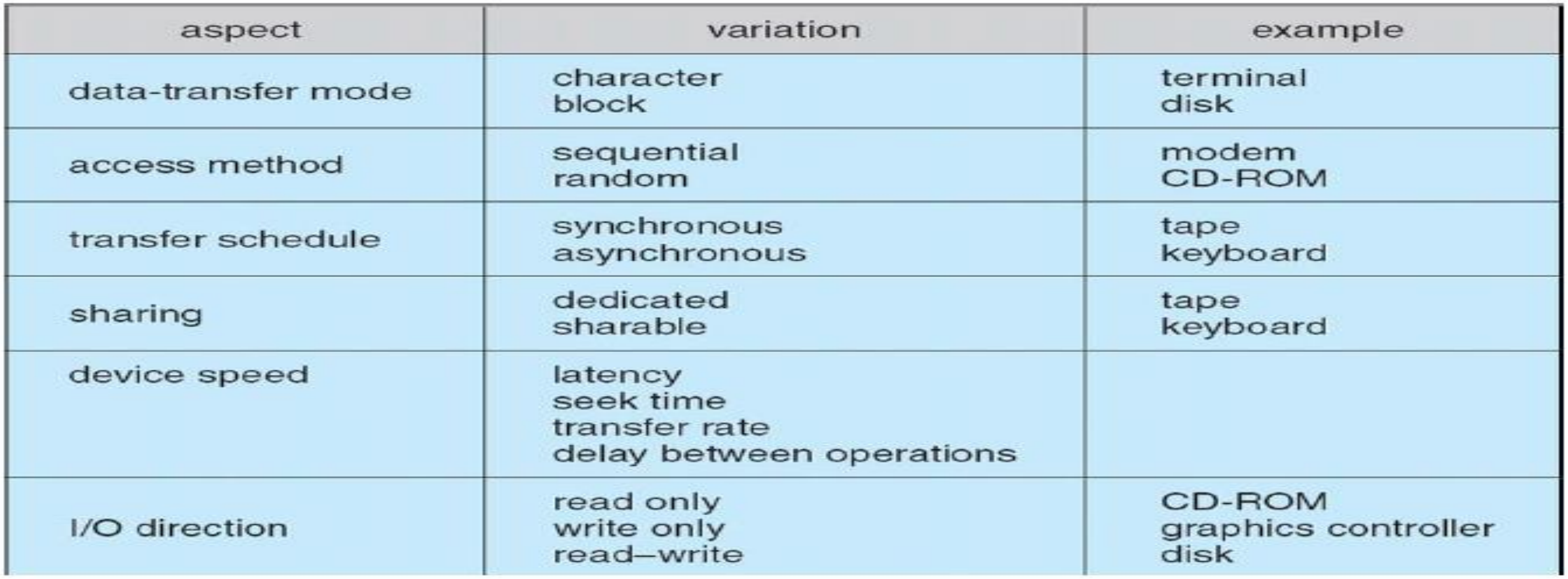


Fig: A kernel I/O structure



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

Fig: Characteristics of I/O devices

Blocking and Nonblocking I/O

- With **blocking I/O** a process is moved to the wait queue when an I/O request is made, and moved back to the ready queue when the request completes, allowing other processes to run in the meantime. It is easy to use and understand but insufficient for some needs.
- With **non-blocking I/O** the I/O request returns immediately, whether the requested I/O operation has (completely) occurred or not. This allows the process to check for available data without getting hung completely if it is not there.
- Asynchronous - process runs while I/O executes
    - Difficult to use
    - I/O subsystem signals process when I/O completed

Ways To Input/Output:

There are three fundamentals ways to do input/output

1.Programmed I/O

2.Interupt-Driven

3.DMA(Direct Memory Access)

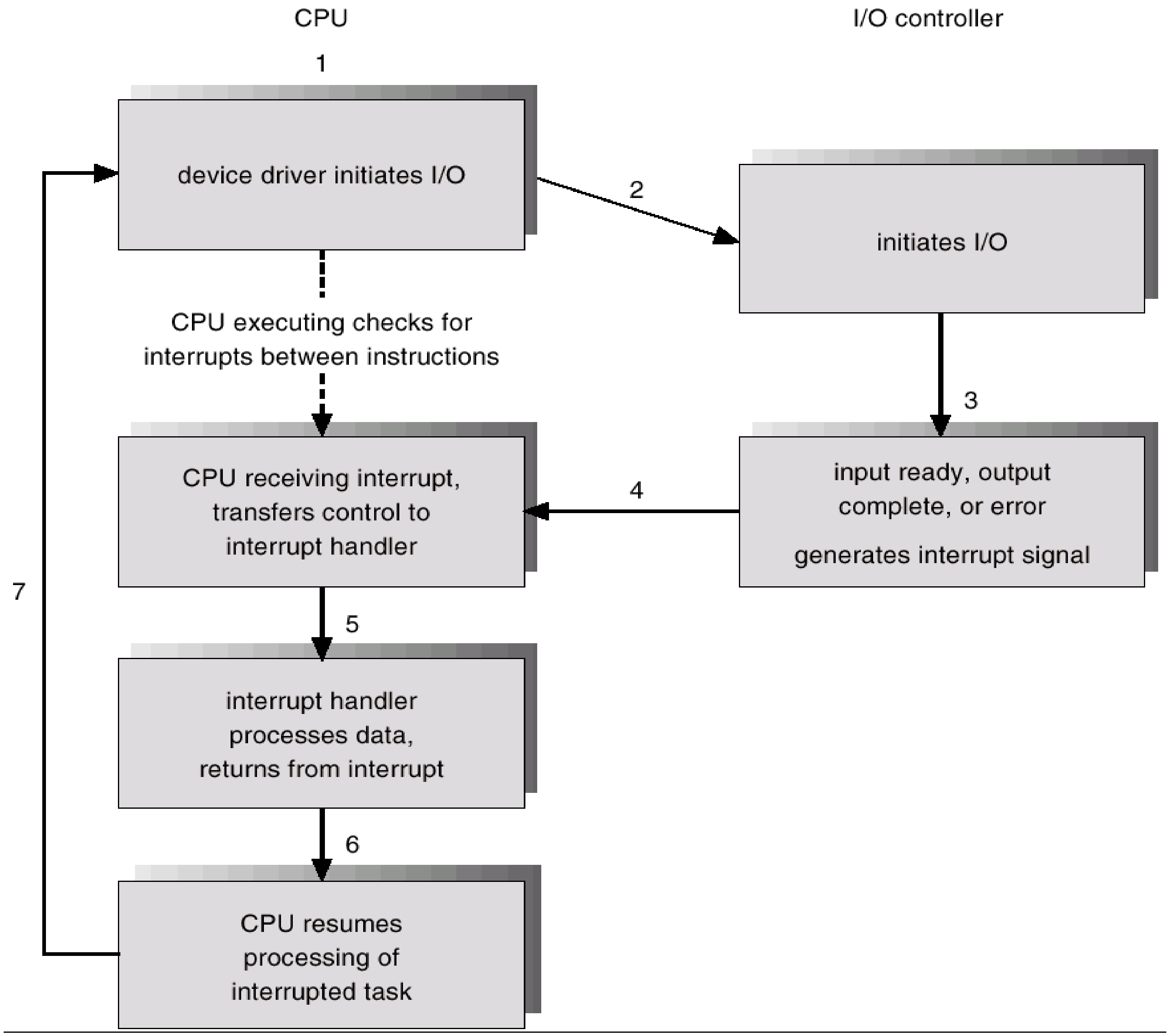
Programmed I/O:

The simplest form form of I/O is to have the CPU do all the work. This method is called **programmed I/O.** The action followed by operating system are summarized in following manner. First the data are copied to the kernel. Then the operating systems enters a tight loop outputting the characters one at a time. The essential aspect of programmed I/O is that after outputting a character, the CPU continuously polls the device to see if it is ready to accept another one. This behavior is often called **polling** or **busy waiting.**

Interrupt - Driven I/O

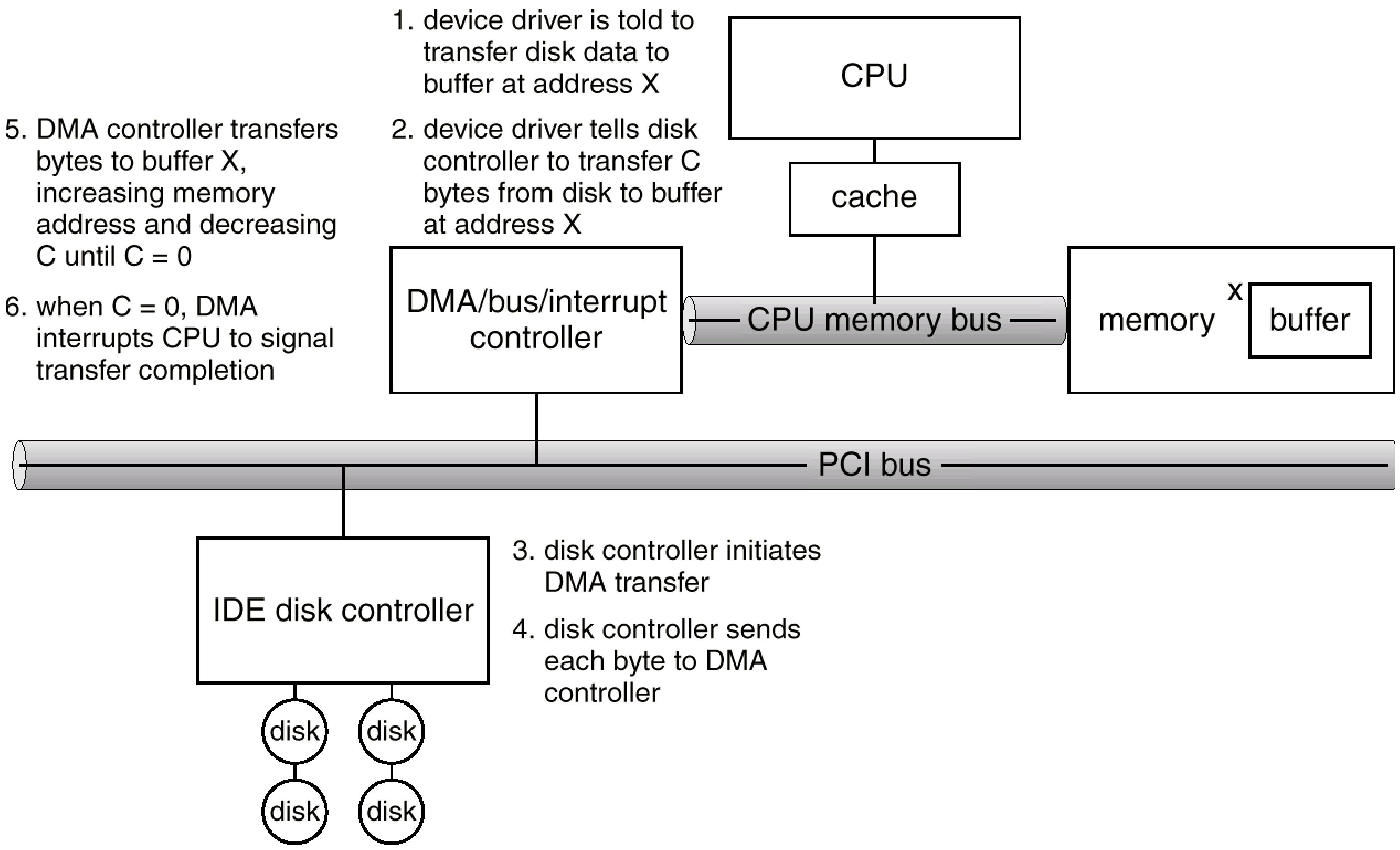
- The basic interrupt mechanism works as follows. The CPU hardware has a wire called the interrupt-request line that the CPU senses after executing every instruction. When the CPU detects that a controller has asserted a signal on the interrupt request line, the CPU saves a small amount of state, such as the current value of the instruction pointer, and jumps to the interrupt-handler routine at a fixed address in memory. The interrupt handler determines the cause of the interrupt, performs the necessary processing, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt. We say that the device controller *raises an interrupt by asserting a* signal on the interrupt request line, the CPU *catches the interrupt and dispatches* to the interrupt handler, and the handler *clears the interrupt by servicing the* device.

Interrupt-Driven I/O Cycle


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

DMA(Direct Memory Access)

- Many computers avoid burdening the main CPU with **PIO** by offloading some of this work to a special-purpose processor called a directmemory-access **(DMA**) controller. To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred.
- The CPU writes the address of this command block to the DMA controller, then goes on with other work. The DMA controller proceeds to operate the memory bus directly, placing addresses on the bus to perform transfers without the help of the main CPU. A simple DMA controller is a standard component in PCs,and bus-mastering I/O boards for the PC usually contain their own high-speed DMA hardware.


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
CPU
cache
CPU memory bus
memory
X
buffer
DMA/bus/interrupt controller
PCI bus
IDE disk controller
disk
disk
disk
disk

Kernel I/O Subsystem

- Kernel provide many services related to I/O. The services that we describe are I/O scheduling, buffering, caching, spooling, device reservation, and error handling.
- **Scheduling**
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
  - Some implement Quality Of Service (i.e. IPQOS)
- **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”
  - Double buffering – two copies of the data
    - Kernel and user
    - Varying sizes
    - Full / being processed and not-full / being used
    - Copy-on-write can be used for efficiency in some cases
- **Caching** - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering
- **Spooling -** hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing

<u>Kernel I/O Subsystem(contd..)</u>

- **Device reservation** - provides exclusive access to a device
    - System calls for allocation and de-allocation
    - Watch out for deadlock
- **Error Handling -** 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

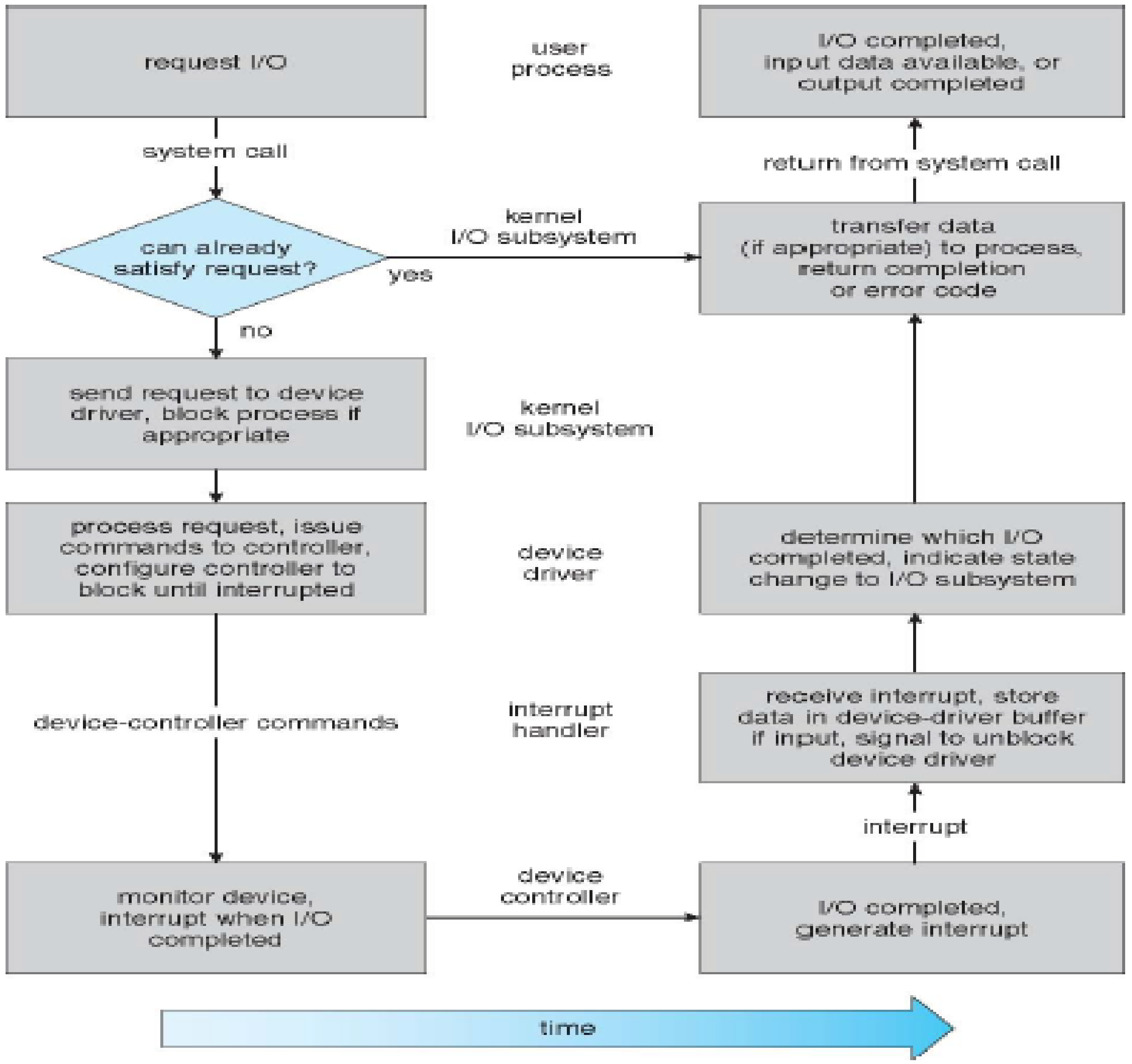
User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions

- All I/O instructions defined to be privileged
- I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

I/O Requests Handling

- Users request data using file names, which must ultimately be mapped to specific blocks of data from a specific device managed by a specific device driver.
- 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


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

4.1.5 Performance

▪ The I/O system is a major factor in overall system performance, and can place heavy loads on other major components of the system (interrupt handling, process switching, memory access, bus contention, and CPU load for device drivers just to name a few.)

▪ Interrupt handling can be relatively expensive (slow), which causes programmed I/O to be faster than interrupt-driven I/O when the time spent busy waiting is not excessive.

▪ Network traffic can also put a heavy load on the system. Consider for example the sequence of events that occur when a single character is typed in a telnet session, as shown in figure 1.

▪ Several principles can be employed to increase the overall efficiency of I/O processing:

- Reduce the number of context switches.
- Reduce the number of times data must be copied.
- Reduce interrupt frequency, using large transfers, buffering, and polling where appropriate.
- Increase concurrency using DMA.
- Move processing primitives into hardware, allowing their operation to be concurrent with CPU and bus operations.
- Balance CPU, memory, bus, and I/O operations, so a bottleneck in one does not idle all the others.

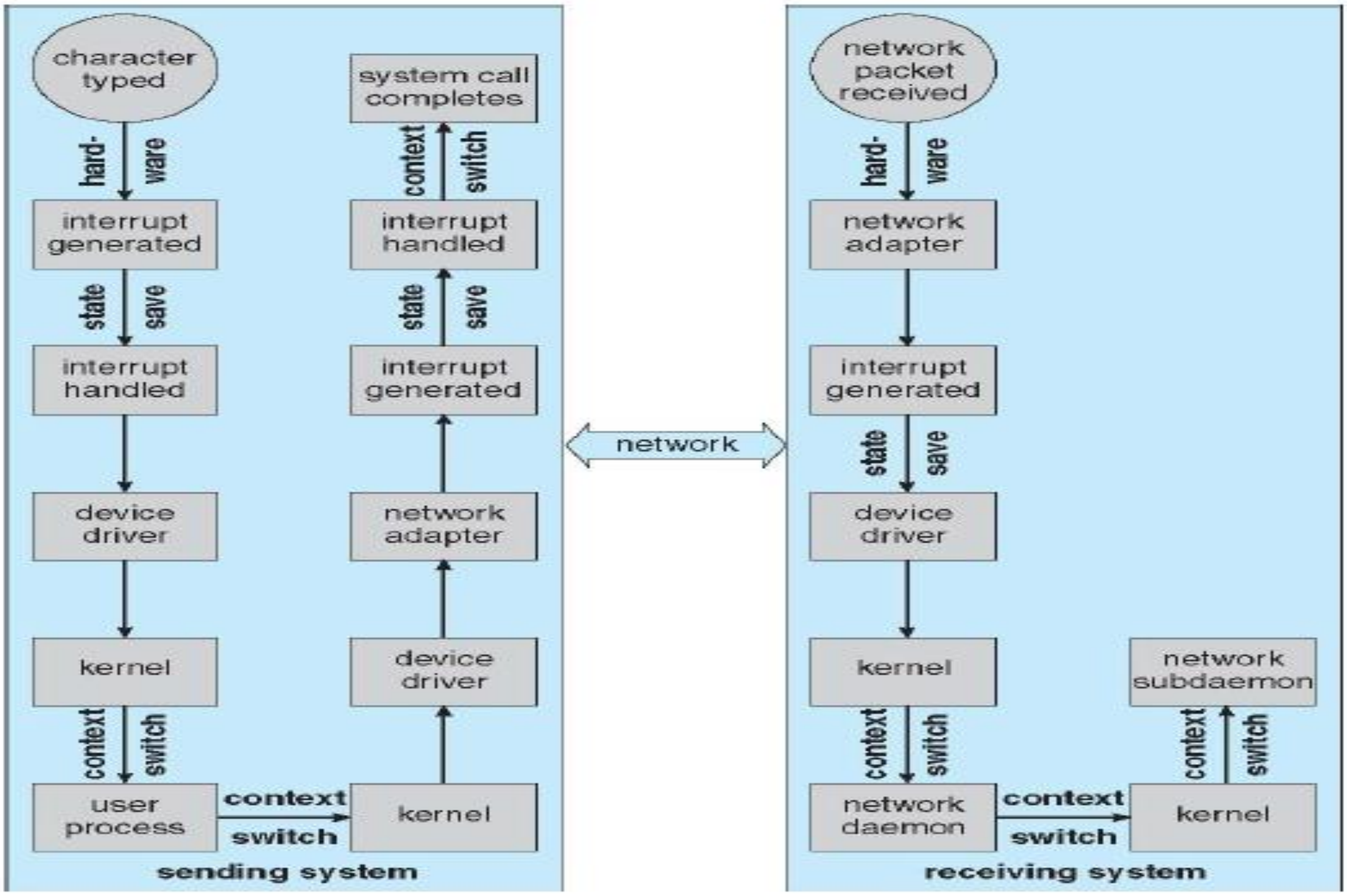


Figure1: Intercomputer Communication

Principles of I/O hardware

- Different people look at I/O hardware in different ways. Electrical engineers look at in terms of chips, wires, power supplies, motors, and all the other physical components that make up the hardware.
- Programmers look at the interface presented to the software- the commands the hardware accepts, the functions it carries out, and the errors that can be reported back.
- Here we are concerned with programming I/O devices, not designing, building, or maintaining them.

I/O Devices

- I/O devices can be roughly divided into two categories: block devices and character devices.
- A block device is one that stores information in fixed- size blocks, each one with its own address. Common block sizes range from 512 bytes to 32,768 bytes. All transfers are in units of one or more entire blocks. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. Hard disks, CD-Roms, and USB sticks are common block devices.
- A character device delivers or accepts a stream of characters, without regard to any block structure. It is not addressable and does not have any seek operation. Printers, network interfaces, mice and most other devices that are not disk- like can be seen as character device.

Device Controllers

- I/O units typically consist of a mechanical component and an electronic component. The electronic component is called the device controller or adapter. On personal computers, it often takes the form of a chip on the parent board or a printed circuit card that can be inserted into a (PCI) expansion slot.
- The controller card usually has a connector on it, into which a cable leading to the device itself can be plugged. Many controllers can handle two, four, or even eight identical devices.
- The interface between the controller and device is often a very low-level interface. What actually comes off the drive, however, is a serial bit stream and finally a checksum. The controller's job is to convert the serial bit stream into a block of bytes and perform any error correction necessary. The block of bytes is typically first assembled, bit by bit, in a buffer inside the controller. After its checksum has been verified and the block has been declared to be error free, it can be copied to main memory.

Memory –Mapped I/O

- Each controller has a few registers that are used for communicating with the CPU. By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action. By reading from these registers, the operating system can learn what the device state is, whether it is prepared to accept a new command, and so on.
- The issue thus arises of how the CPU communicates with the control registers and the device data buffers. For this we map all the control registers into memory space. Each control register is assigned a unique memory address to which no memory is assigned. This system is called **memory-mapped I/O**. Usually, the assigned address are at the top of the address space.

Goals of the I/O Software

- A key concept in the design of I/O software is known as **device independence.** It means that is should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on hard disk, a CD-ROM, a DVD, or a USB stick without having to modify the programs for each different device.
- Closely related to device independence is the goal of **uniform naming.** The name of a file or a device should simply be a string or an integer and not depend on the device in any way.

Goals of the I/O Software(contd…)

- Another important issue for I/O software is **error handling.** In general, errors should be handled as close to the hardware as possible. If the controller discovers a read error, it should try to correct the error itself if it can. If it cannot, then the device driver should handle it, perhaps by just trying to read the block again.
- Another key issue is that **synchronous**(blocking) versus **asynchronous**(interrupt-driven) transfers. Most physical I/O is asynchronous- the CPU starts the transfer and goes off to do something else until the interrupt arrives.
- Another issue for the I/O software is **buffering.** Often data that come off a device cannot be stored directly in its final destination.For example, when a packet comes in off the network, the operating system does not know where to put it until it has stored the packet somewhere and examined it.

CUI and GUI

- GUI and CUI are two types of User Interfaces.
- **GUI**: GUI stands for **Graphical User Interface**. This is a type of user interface where the user interacts with the computer using graphics. Graphics include icons, navigation bars, images, etc. A mouse can be used while using this interface to interact with the graphics. It is a very user-friendly interface and requires no expertise. Eg: Windows has GUI.
- **CUI**: CUI stands for **Character User Interface**. This is a user interface where the user interacts with a computer using only a keyboard. To perform any action a command is required. CUI is a precursor of GUI and was used in most primitive computers. Eg: MS-DOS has CUI

Device Drivers

- We know that device controller has some device registers used to give it commands or some device registers used to read out its status or both. The number of device registers and the nature of the commands vary radically from device to device. For example, a mouse driver has to accept information from the mouse telling how far it has moved and which buttons are currently depressed. In contrast, a disk driver may have to know all about sectors, tracks, cylinders, heads, arm motion, motor drives, and all the other mechanics of making disk work properly. Obviously, these drivers will be very different.
- As a consequence, each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the **device driver,** is generally written by device's manufacturer and delivered along with the device. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

Device Drivers(contd.)

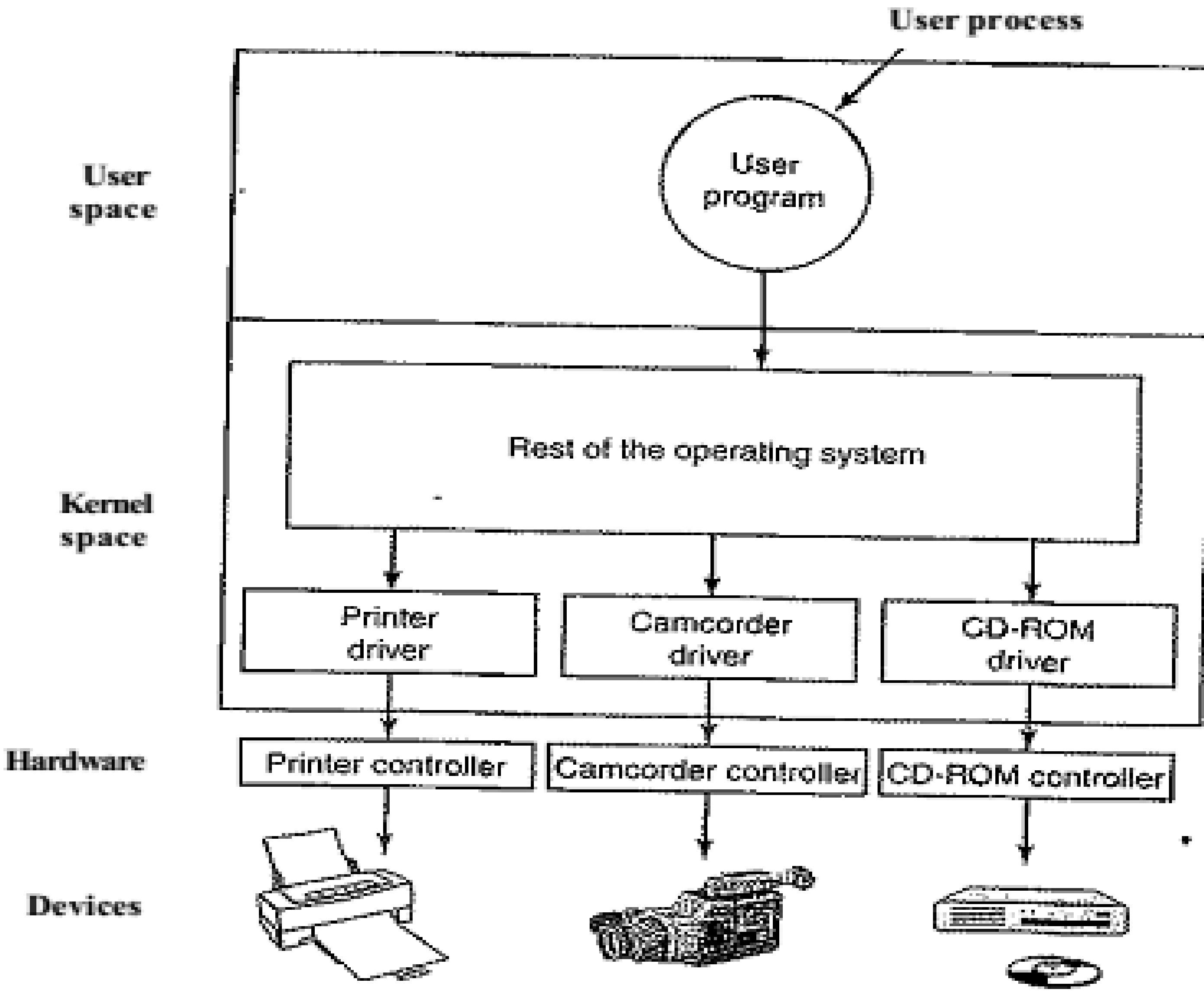


Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

Device-Independent I/O Software

- Although some of the I/O software is device specific, other parts of it are device independent. The exact boundary between the drivers and the device- independent software is system (and device) dependent, because some functions that could be done in a device-independent way may actually be done in the drivers, for efficiency or other reasons. The functions shown in Fig. below are typically done in the device-independent software.

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

- The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software.

User Space I/O Software

- Although most of the I/O software is within the operating system, a small portion of it consists of libraries linked together with user programs, and even whole programs running outside the kernel. System calls, including the I/O system calls, are normally made by library procedures.
- I/O Libraries(e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example printf(), scanf() are example of user level I/O library stdio available in C programming.

4.2 Mass-Storage Device

▪ A mass storage device (MSD) is any storage device that makes it possible to store and port large amounts of data across computers, servers and within an IT environment. MSDs are portable storage media that provide a storage interface that can be both internal and external to the computer.

4.2.1 Disk Structure :

▪Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.

▪The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially

- Sector 0 is the first sector of the first track on the outermost Cylinder.
- Mapping proceeds in order through that track, then the rest Of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.
- Logical to physical address should be easy
  - Except for bad sectors
  - Non-constant # of sectors per track via constant angular velocity

4.2.2 Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
  - " *Seek time is the time for the disk are to move the heads to* the cylinder containing the desired sector.
  - " *Rotational latency is the additional time waiting for the disk* to rotate the desired sector to the disk head.
- Minimize seek time
- Seek time ≈ seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Arm Scheduling Algorithm

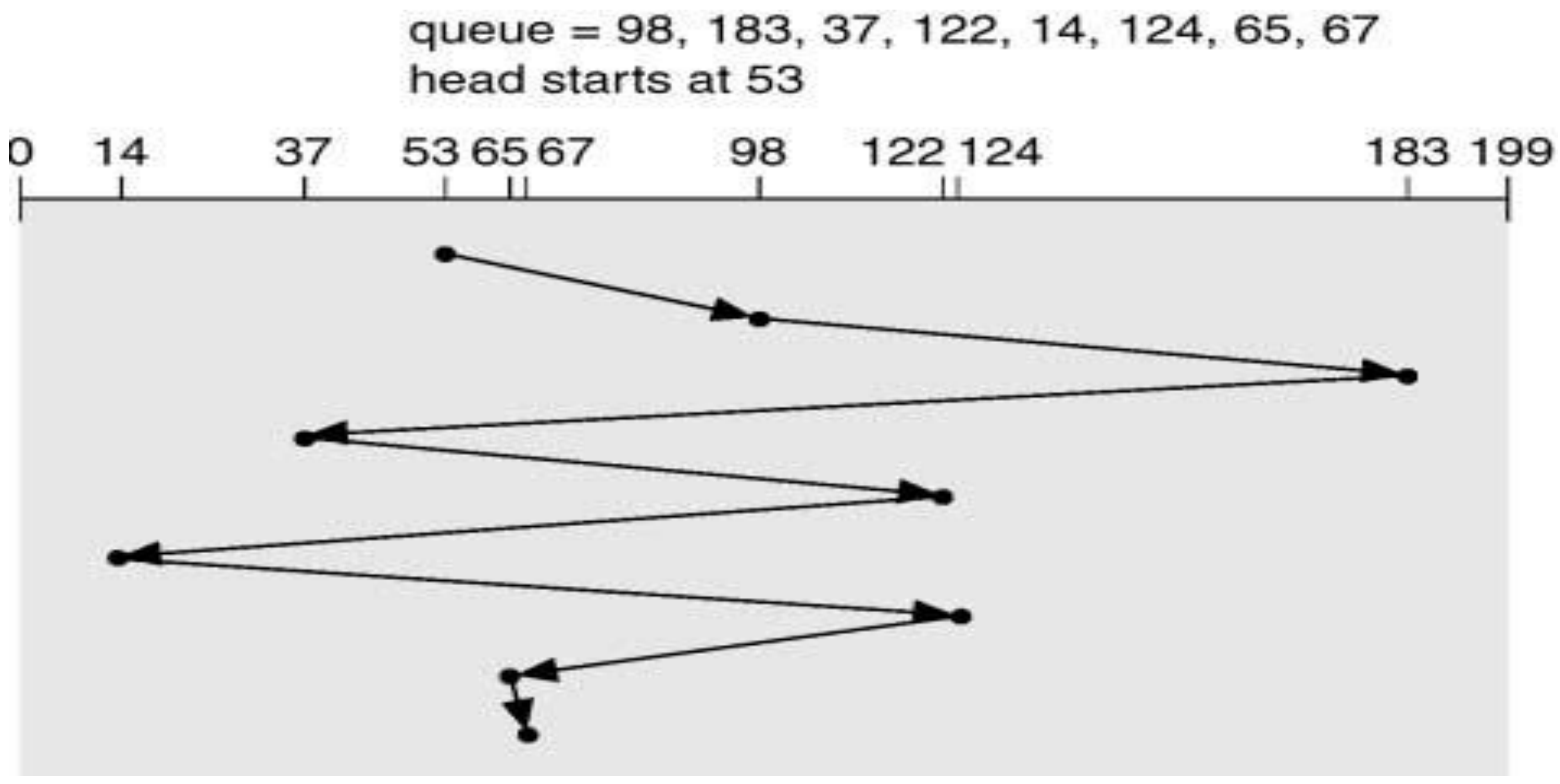
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

    98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS

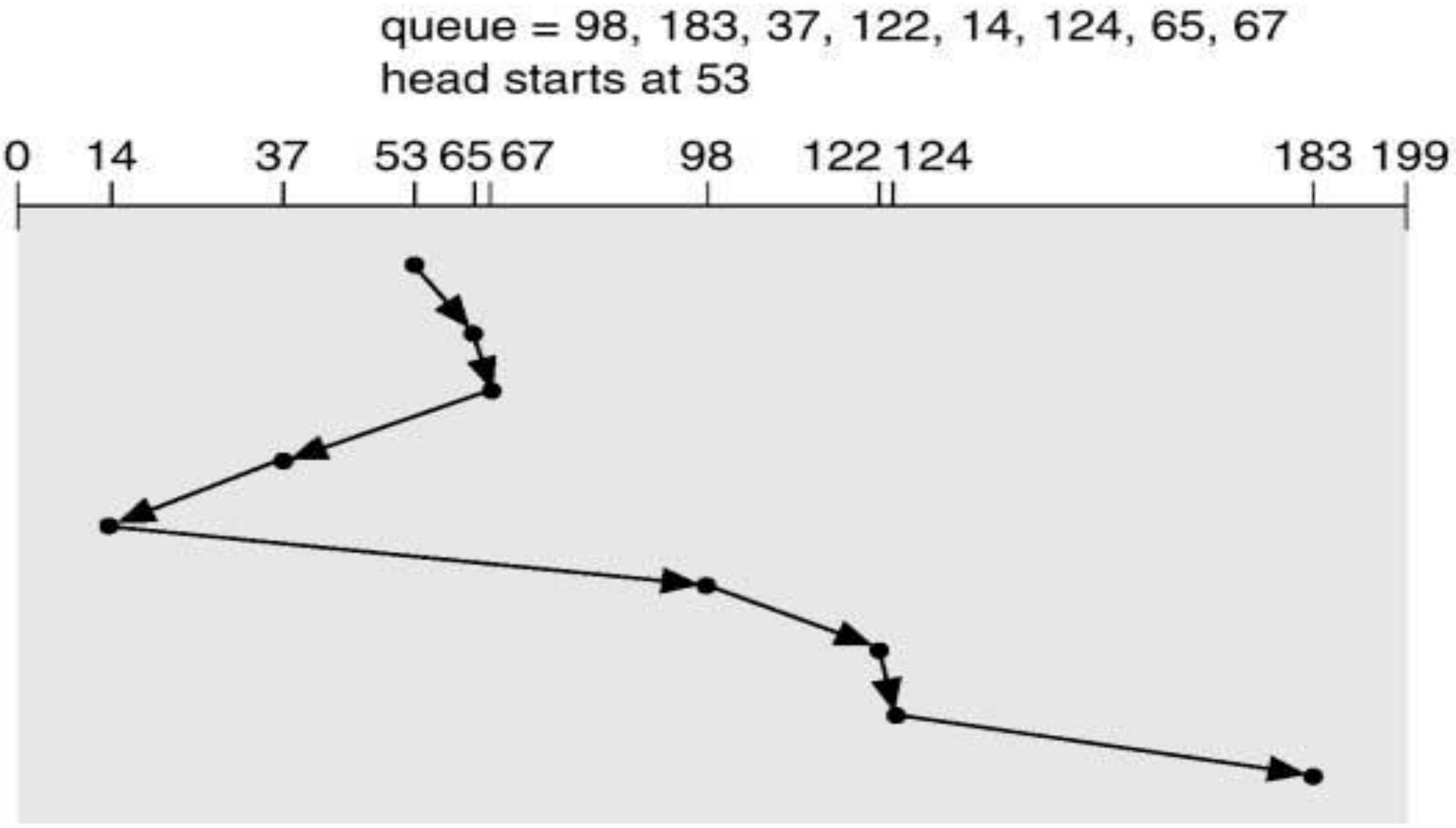
Illustration shows total head movement of 640 cylinders



Shortest Seek First(SSF)

- Selects the request with the minimum seek time from the current head position.
- SSF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

SSF (Cont.)

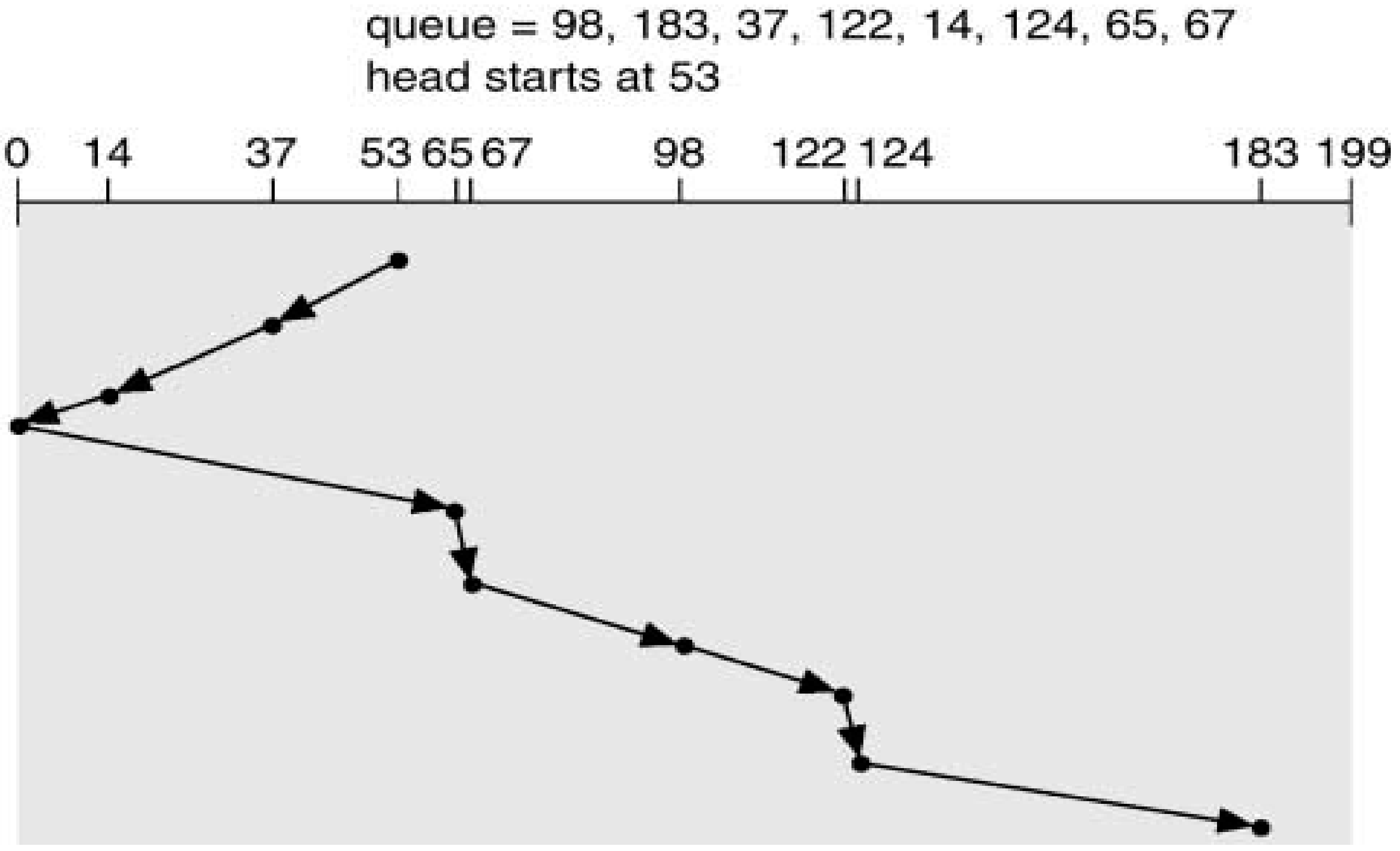


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm.*
- Illustration shows total head movement of 208 cylinders.

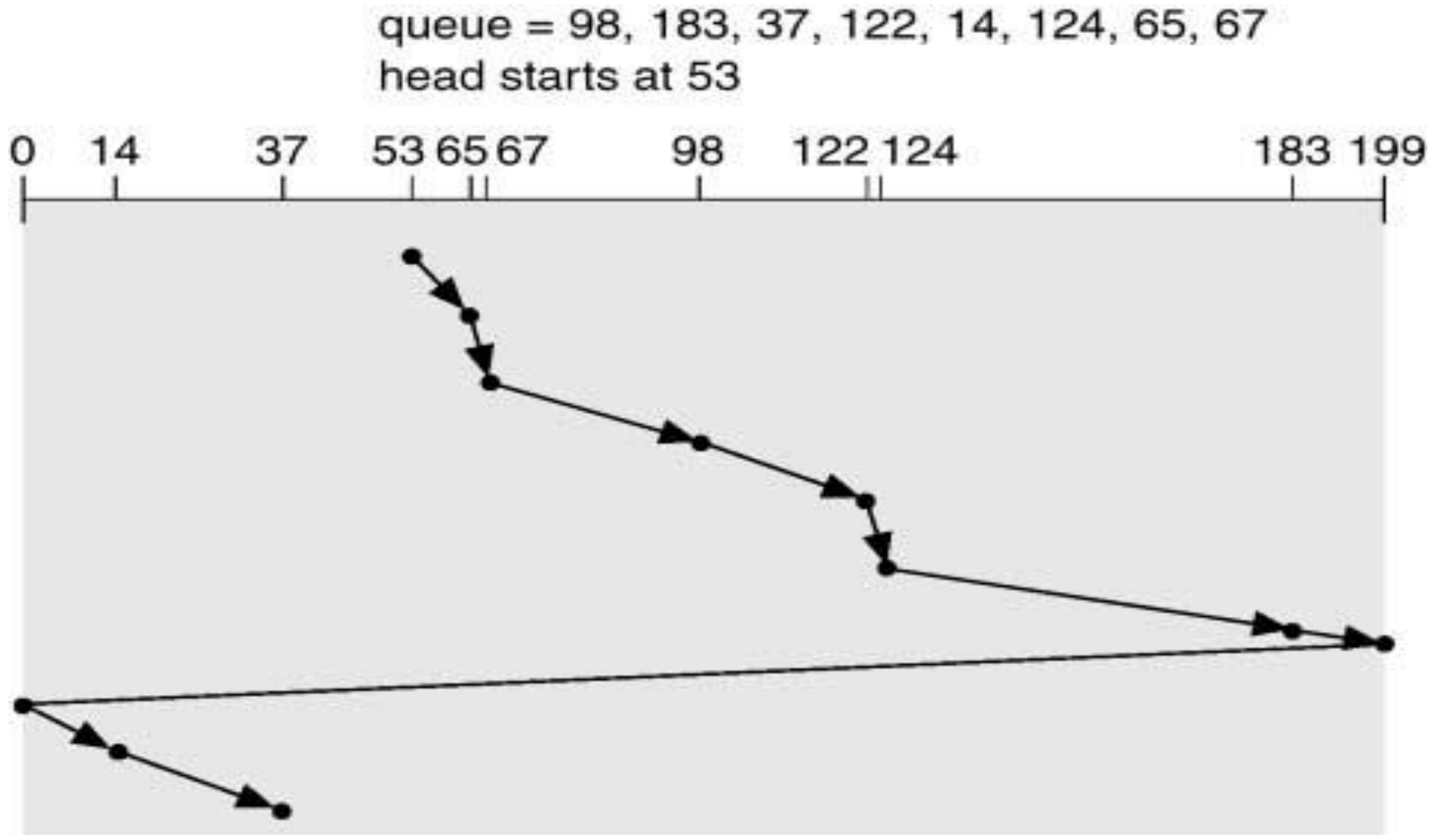
SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN

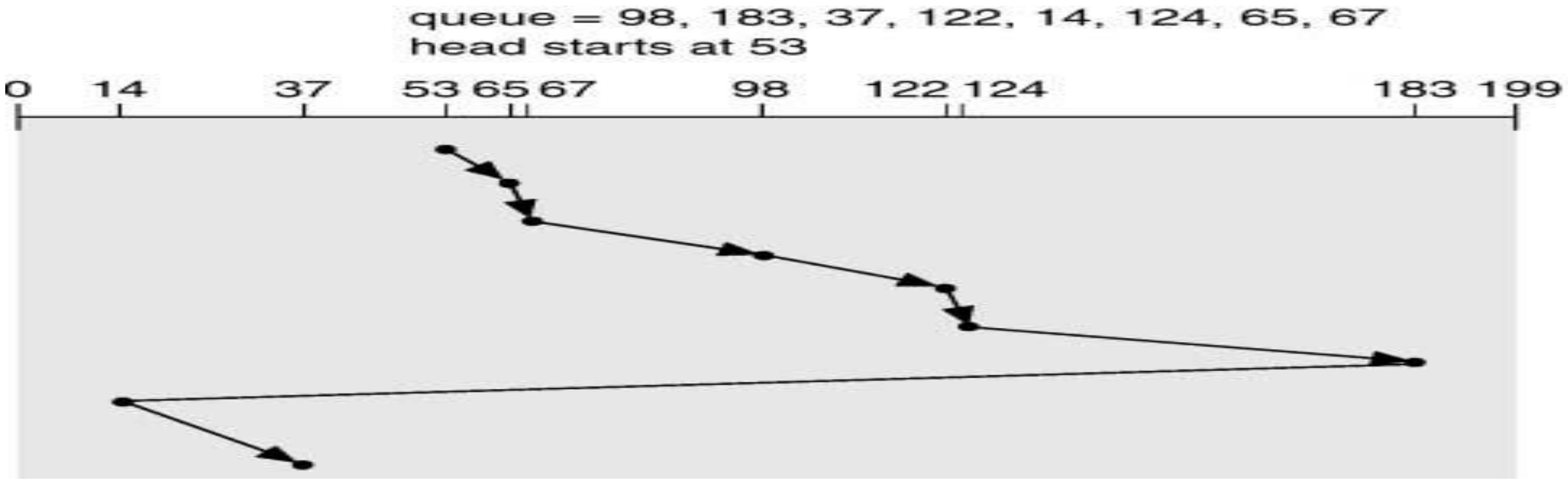
- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK

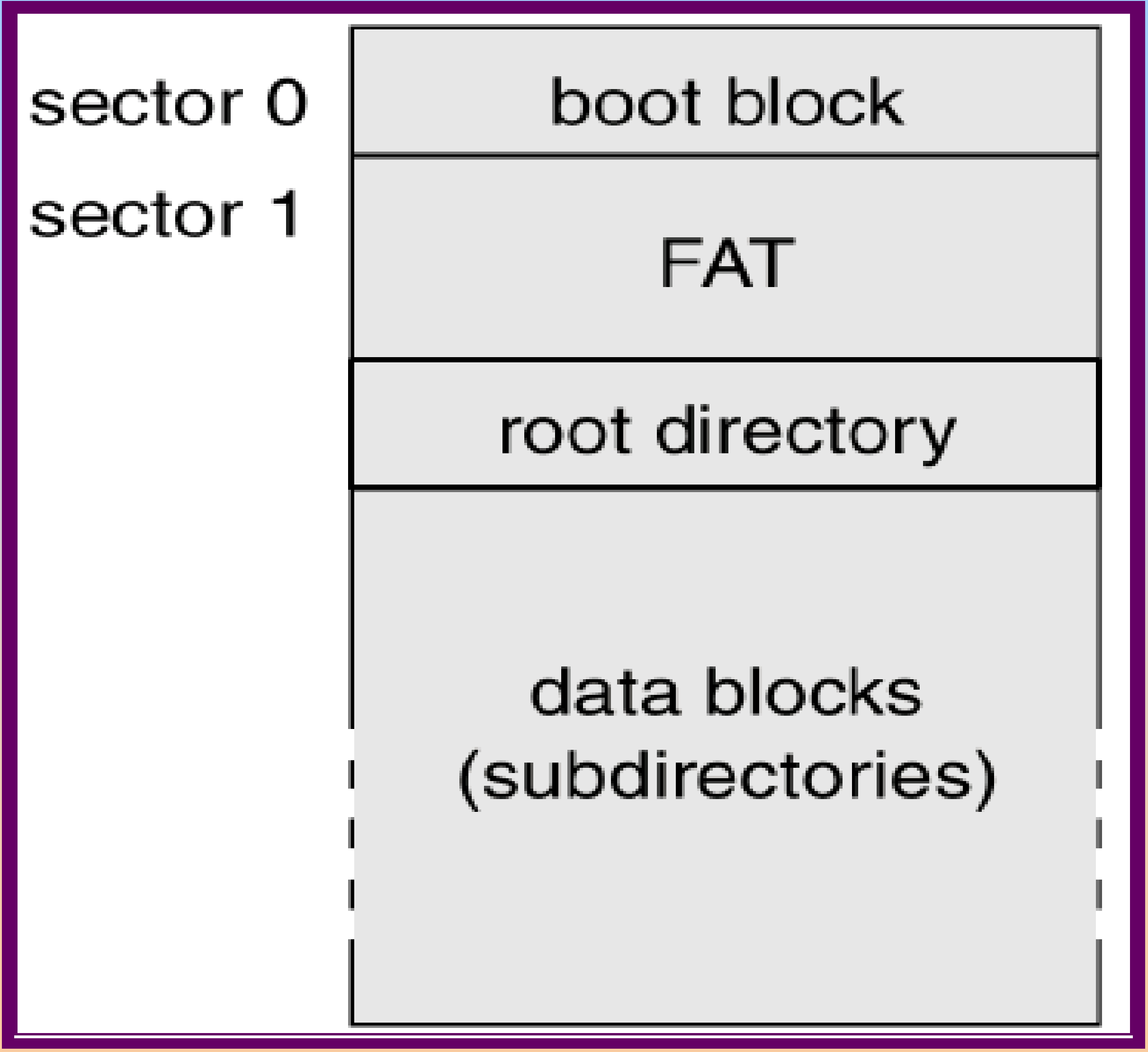
- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



4.2.3 Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (ECC)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - Partition the disk into one or more groups of cylinders, each treated as a logical disk
  - Logical formatting or “making a file system”
  - To increase efficiency most file systems group blocks into clusters
    - ➢ Disk I/O done in blocks
    - ➢ File I/O done in clusters
- Boot block initializes system
  - The bootstrap is stored in ROM
  - Bootstrap loader program stored in boot blocks of boot partition
- Methods such as sector sparing used to handle bad blocks

MS-DOS Disk Layout



Bad Blocks:

▪The disk with defected sector is called as bad block.

▪Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: “Handled manually

• If blocks go bad during normal operation, a special program must be run manually to search the bad blocks and to lock them away as before. Data that resided on the bad Blocks usually are lost.

Method 2: “sector sparing or forwarding”

•The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

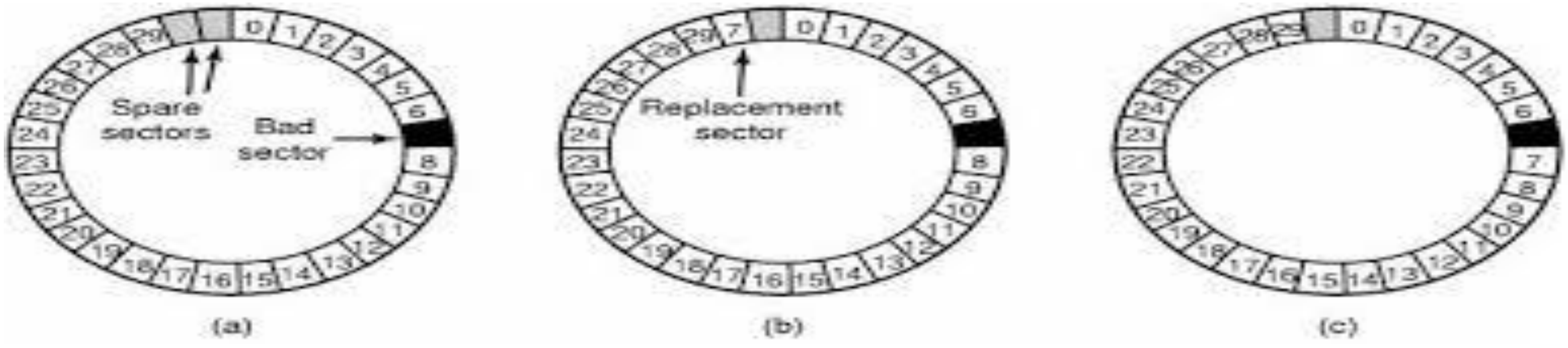
•A typical bad-sector transaction might be as follows:

- ✓ The oper ating system tries to read logical block 7.
- ✓ The controller calculates the ECC and finds that the sector is bad.
- ✓ It reports this finding to the operating system.
- ✓ The next time that the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.
- ✓ After that, whenever the system requests logical block 7, the request is translated into the replacement sector's address by the controller.

Bad Block handling(contd…)

Method 3: “sector slipping”

- For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.



Interleaving:

•Interleaving is a process or methodology to make a system more efficient, fast and reliable by arranging data in a noncontiguous manner. There are many uses for interleaving at the system level, including:

- Storage: As hard disks and other storage devices are used to store user and system data, there is always a need to arrange the stored data in an appropriate way.
- Error Correction: Errors in data communication and memory can be corrected through interleaving.

Interleaving is also known as sector interleave.

• When used to describe disk drives, it refers to the way *sectors* on a disk are organized. In one-to-one interleaving, the sectors are placed sequentially around each track. In two-to-one interleaving, sectors are staggered so that consecutively numbered sectors are separated by an intervening sector.

•The purpose of interleaving is to make the disk drive more efficient. The disk drive can access only one sector at a time, and the disk is constantly spinning beneath the **read/write head**. This means that by the time the drive is ready to access the next sector, the disk may have already spun beyond it. If a data file spans more than one sector and if the sectors are arranged sequentially, the drive will need to wait a full rotation to access the next chunk of the file. If instead the sectors are staggered, the disk will be perfectly positioned to access sequential sectors.

Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory.
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses swap maps to track swap-space use
  - Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
    - File data written to swap space until write to file system requested
    - Other dirty pages go to swap space due to no other home
    - Text segment pages thrown out and reread from the file system as needed
- What if a system runs out of swap space?
- Some systems allow multiple swap spaces

Swap-Space Management: An Example

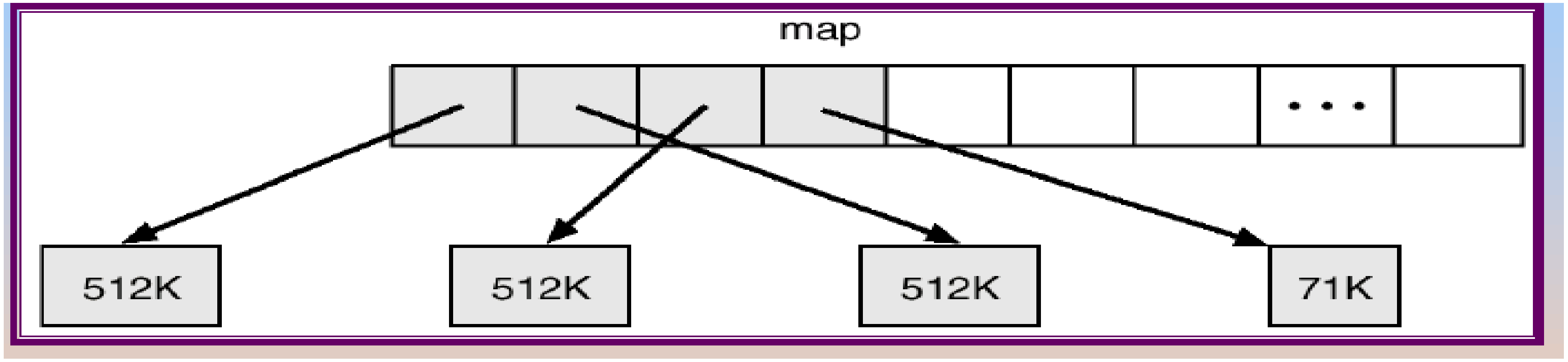


Fig1.1: 4.3BSD text-segment swap map.

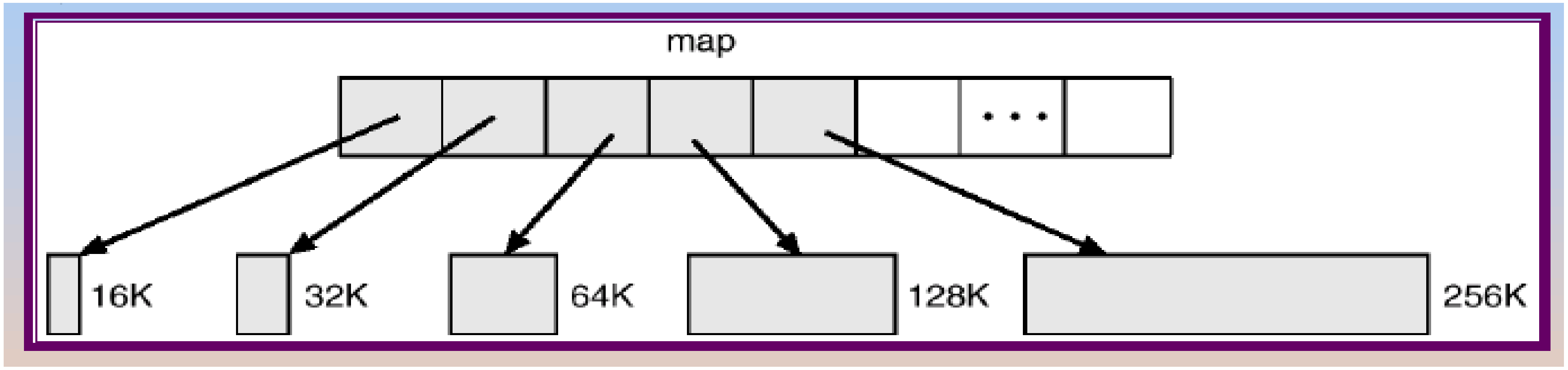


Fig 1.2: 4.3BSD data-segment swap map

- In 4.3 BSD, swap space is allocated to a process when the process is started. Enough space is set aside to hold the program, known as the text pages or the text segment, and the data segment of the process. Preallocating all the needed space in this way generally prevents a process from running out of swap space while it executes. When a process starts, its text is paged in from the file system. These pages are written out to swap when necessary, and are read back in from there, so the file system is consulted only once for each text page. Pages from the data segment are read in from the file system, or are created (if they are uninitialized), and are written to swap space and paged back in as needed. One optimization (for instance, when two users run the same editor) is that processes with identical text pages share these pages, both in physical memory and in swap space.
- Two per-process swap maps are used by the kernel to track swap-space use. The text segment is a fixed size, so its swap space is allocated in 512 KB chunks, except for the final chunk, which holds the remainder of the pages, in 1 KB increments(fig 1.1)
- The data-segment swap map is more complicated, because the data segment can grow over time. The map is of fixed size, but contains swap addresses for blocks of varying size. Given index i, a block pointed to by swap-map entry i is of size 2' x 16 KB, to a maximum of 2 MB. This data structure is shown in Figure 1.2. (The block size minimum and maximum are variable, and can be changed at system reboot.) When a process tries to grow its data segment beyond the final allocated block in its swap area, the operating system allocates another block, twice as large as the previous one. This scheme results in small processes using only small blocks. It also minimizes fragmentation. The blocks of large processes can be found quickly, and the swap map remains small.

RAID

- Disk drives have continued to get smaller and cheaper, so it is now economically feasible to attach a large number of disks to a computer system. Having a large number of disks in a system presents opportunities for improving the rate at which data can be read or written, if the disks are operated in parallel. Furthermore, this setup offers the potential for improving the reliability of data storage, because redundant information can be stored on multiple disks. Thus, failure of one disk does not lead to loss of data. A variety of disk-organization techniques, collectively called redundant arrays of inexpensive disks (RAID), are commonly used to address the performance and reliability issues.

RAID LEVELS

RAID level 0 – Striping:

- In a RAID 0 system data are split up into blocks that get written across all the drives in the array. By using multiple disks (at least 2) at the same time, this offers superior I/O performance. This performance can be enhanced further by using multiple controllers, ideally one controller per disk.

Advantages:

- RAID 0 offers great performance, both in read and write operations. There is no overhead caused by parity controls.
- All storage capacity is used, there is no overhead.
- The technology is easy to implement.

Disadvantages:

- RAID 0 is not fault-tolerant. If one drive fails, all data in the RAID 0 array are lost. It should not be used for mission-critical

RAID LEVELS

RAID level 1 – Mirroring:

- Data are stored twice by writing them to both the data drive (or set of data drives) and a mirror drive (or set of drives). If a drive fails, the controller uses either the data drive or the mirror drive for data recovery and continues operation. You need at least 2 drives for a RAID 1 array.

Advantages

- RAID 1 offers excellent read speed and a write-speed that is comparable to that of a single drive.
- In case a drive fails, data do not have to be rebuild, they just have to be copied to the replacement drive.
- RAID 1 is a very simple technology.

Disadvantages

- The main disadvantage is that the effective storage capacity is only half of the total drive capacity because all data get written twice.

RAID LEVELS

RAID level 5:

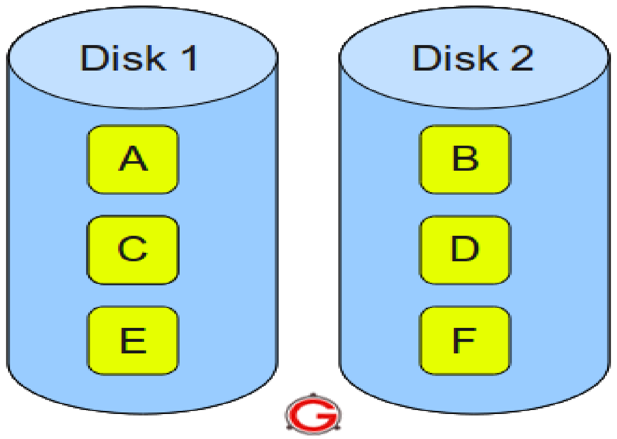
- RAID 5 is the most common secure RAID level. It requires at least 3 drives but can work with up to 16. Data blocks are striped across the drives and on one drive a parity checksum of all the block data is written. The parity data are not written to a fixed drive, they are spread across all drives, as the drawing below shows. Using the parity data, the computer can recalculate the data of one of the other data blocks, should those data no longer be available. That means a RAID 5 array can withstand a single drive failure without losing data or access to data.

Advantages:

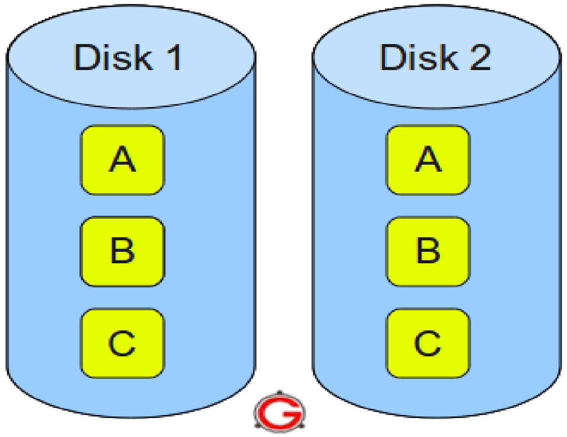
- Read data transactions are very fast while write data transactions are somewhat slower (due to the parity that has to be calculated).
- If a drive fails, you still have access to all data, even while the failed drive is being replaced and the storage controller rebuilds the data on the new drive.

Disadvantages:

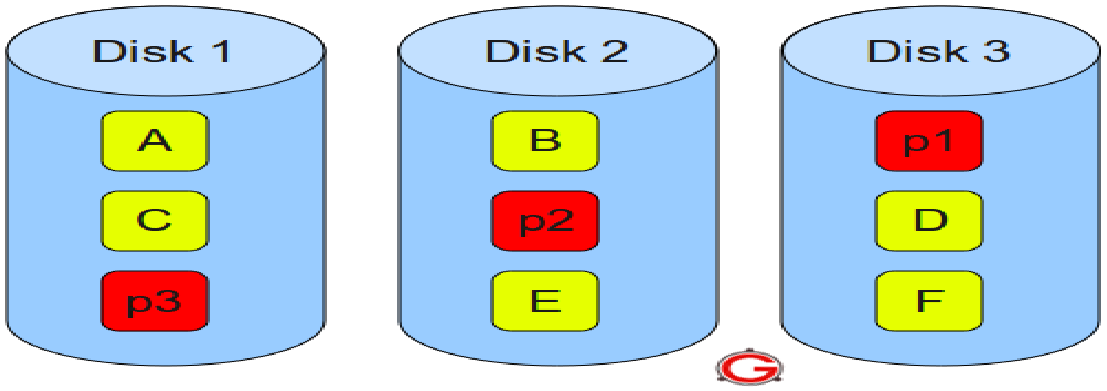
- Drive failures have an effect on throughput, although this is still acceptable.



RAID 0 – Blocks Striped. No Mirror. No Parity.



RAID 1 – Blocks Mirrored. No Stripe. No parity.



RAID 5 – Blocks Striped. Distributed Parity.