Applied Operating System

Chapter 4: I/O Management

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

4.1 I/O Sub- Systems

4.1.1 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.

4.1.2 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.

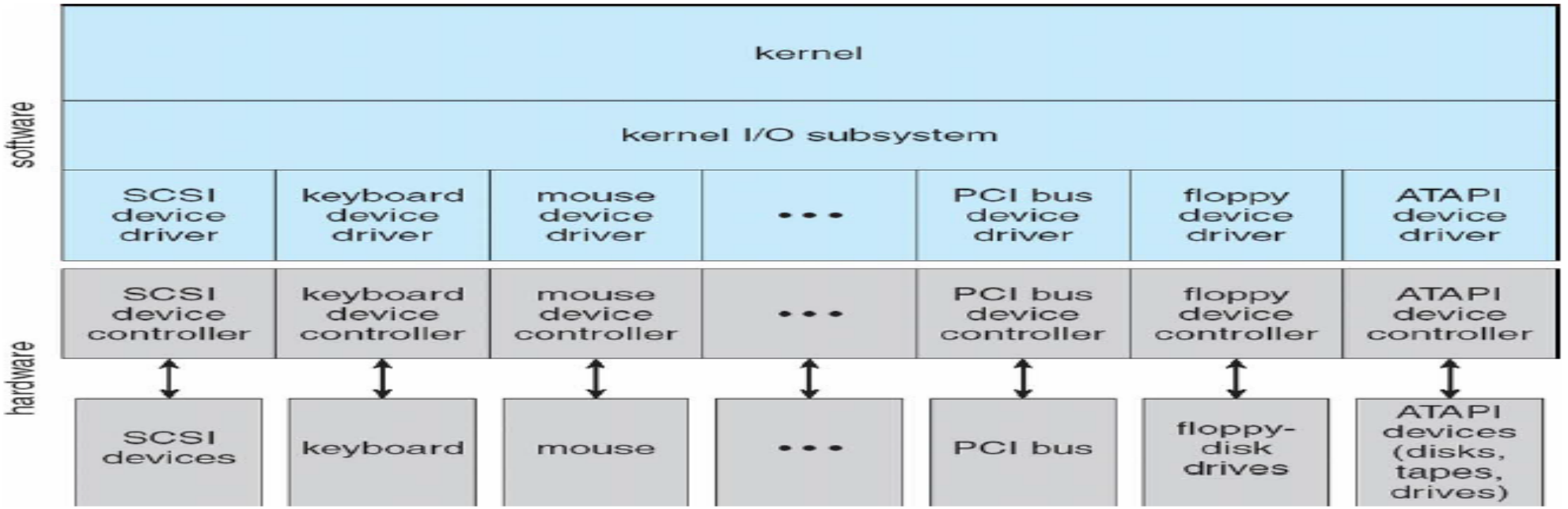


Fig: A kernel I/O structure

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

Characteristics of I/O Devices (Cont.)

• Most devices can be characterized as either block I/O, character I/O, memory mapped file access, or network sockets. A few devices are special, such as time-of-day clock and the system timer.

• Most OSes also have an escape, or back door, which allows applications to send commands directly to device drivers if needed. In UNIX this is the **ioctl**() system call (for I/O Control).

Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - **Raw I/O, direct I/O, or file-system access**
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- 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/9x/2000 include socket interface
  - Separates network protocol from network operation
  - Includes select() functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mail**lboxes**

Clocks and Timers

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- 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

- 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)

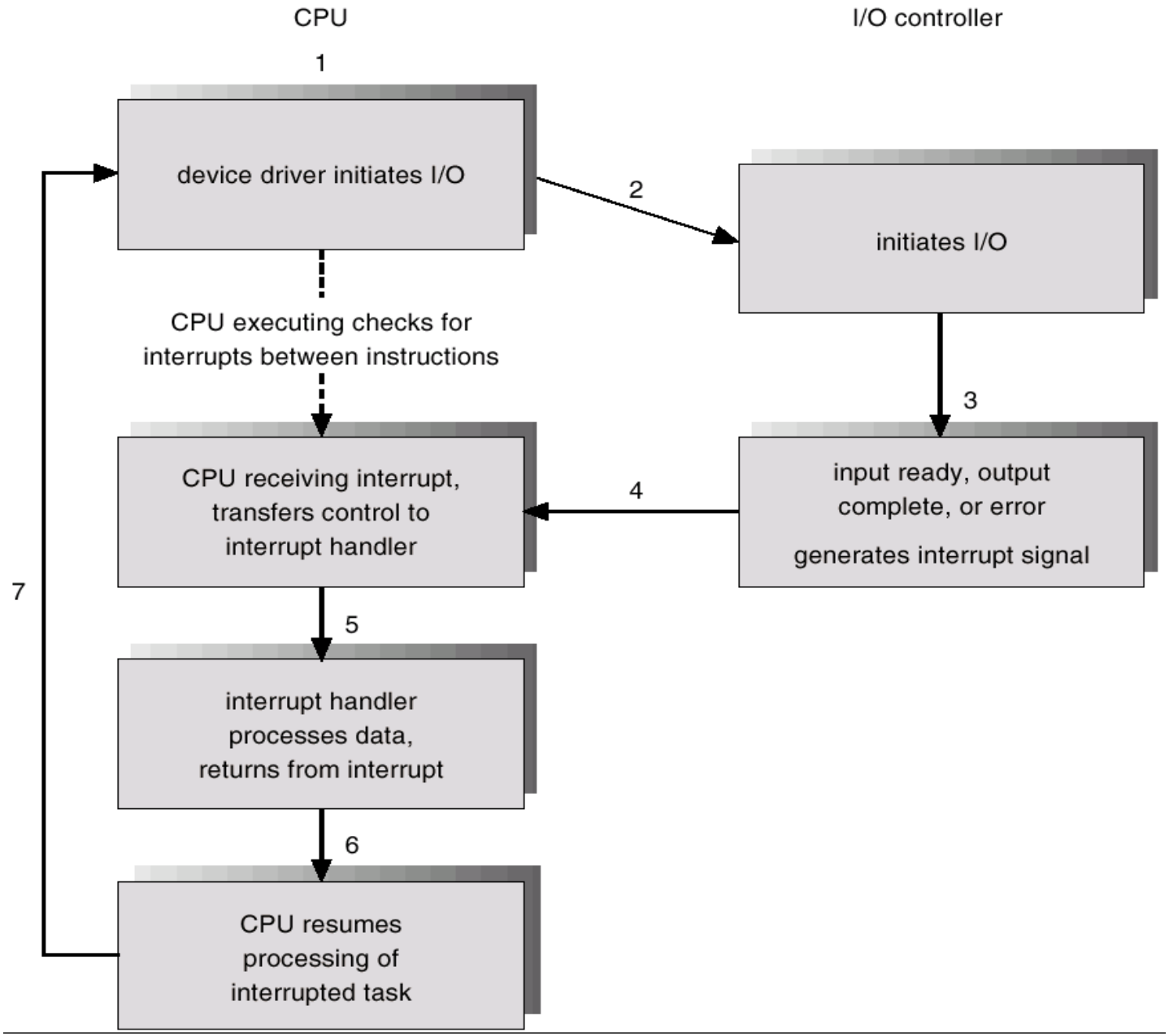
Programmed I/O:

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

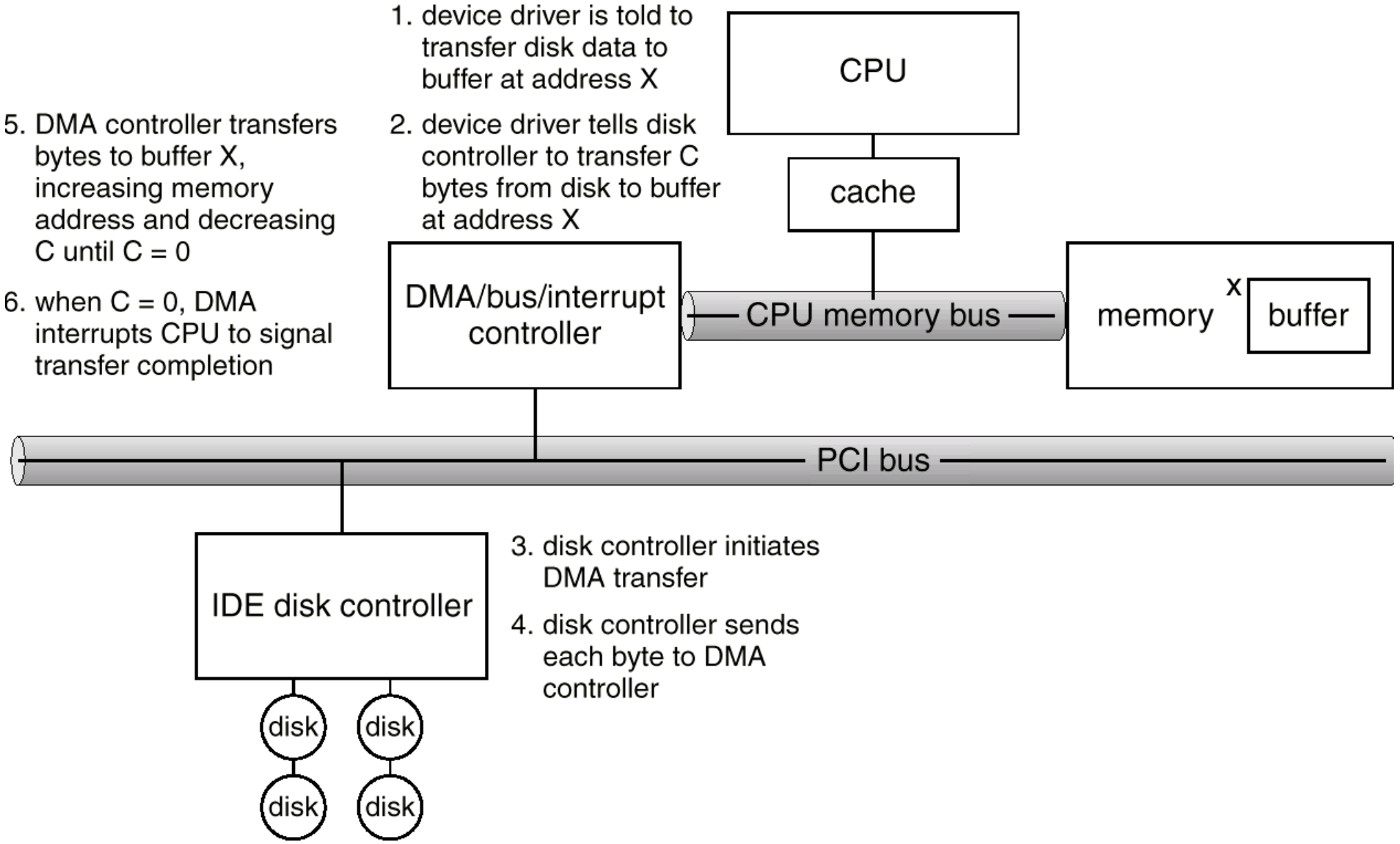
- The basic interrupt mechanism works as follows. The CPU hardware has a wire called the interrupt-request line that the CPU senses after executing every instruction. When the CPU detects that a controller has asserted a signal on the interrupt request line, the CPU saves a small amount of state, such 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



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
DMA/bus/interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

4.1.3 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

Kernel I/O Subsystem(contd..)

- **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

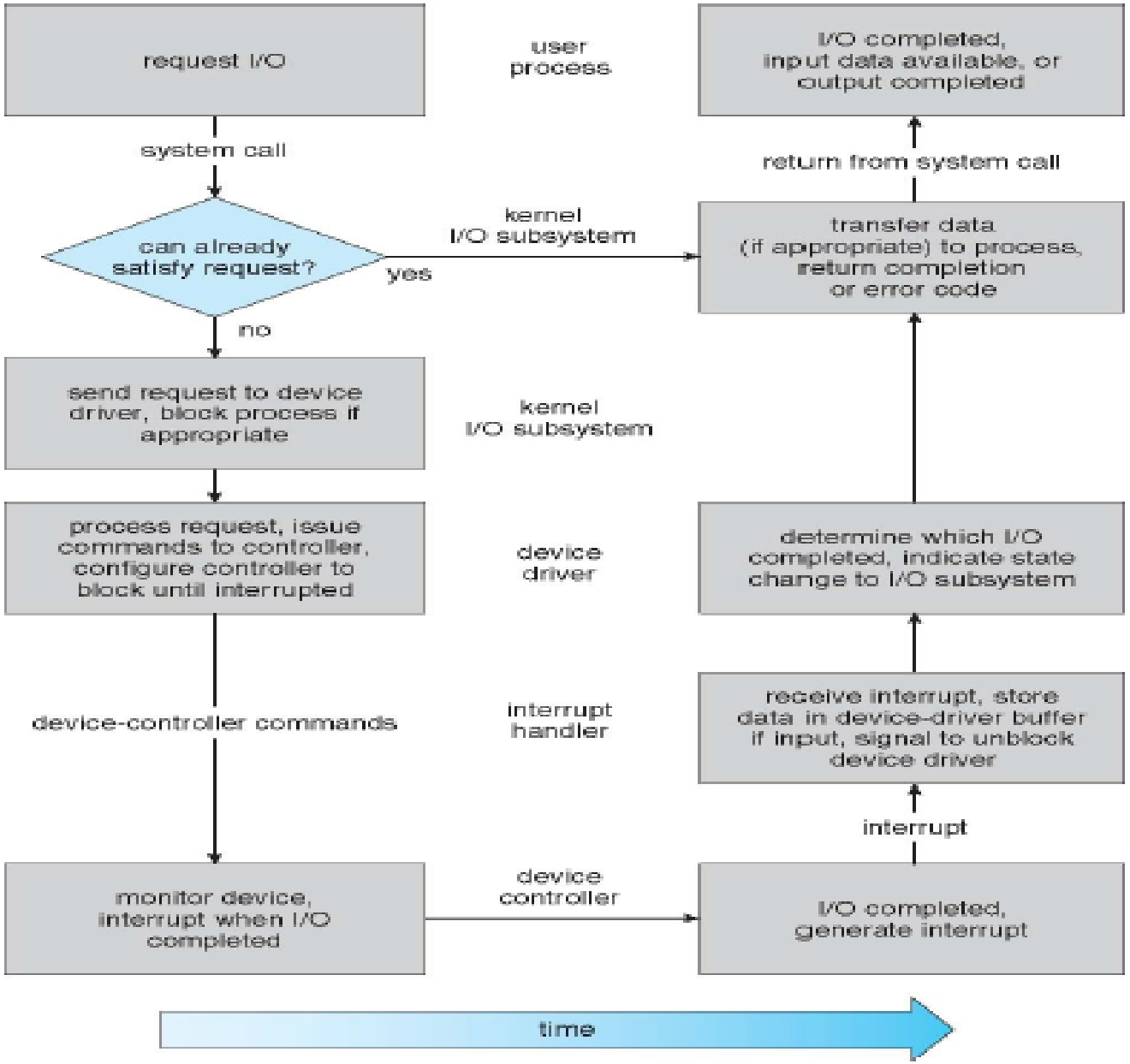
User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions

- 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

4.1.4 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



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.

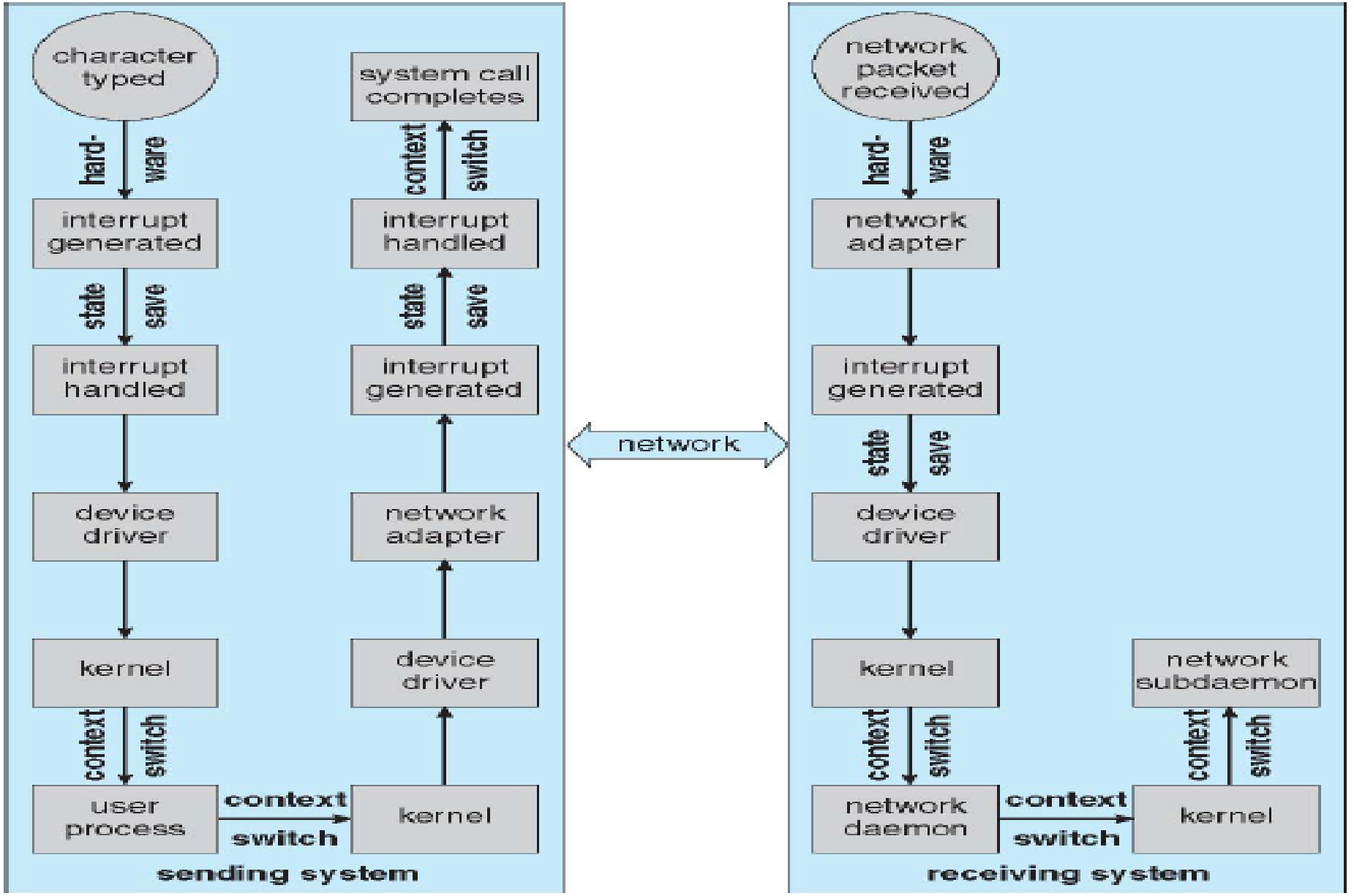


Figure1: Intercomputer Communication

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

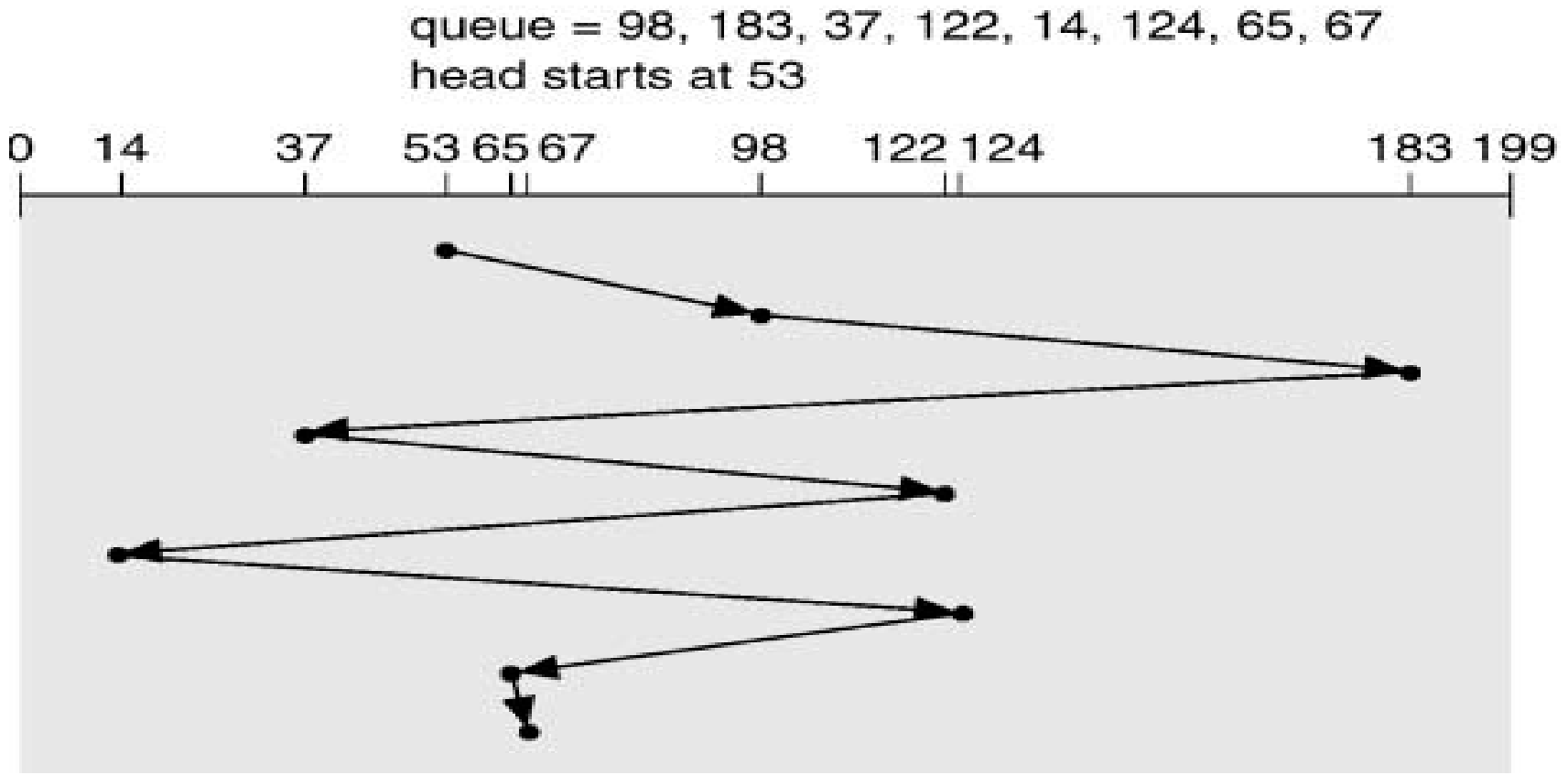
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

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

  Head pointer 53

FCFS

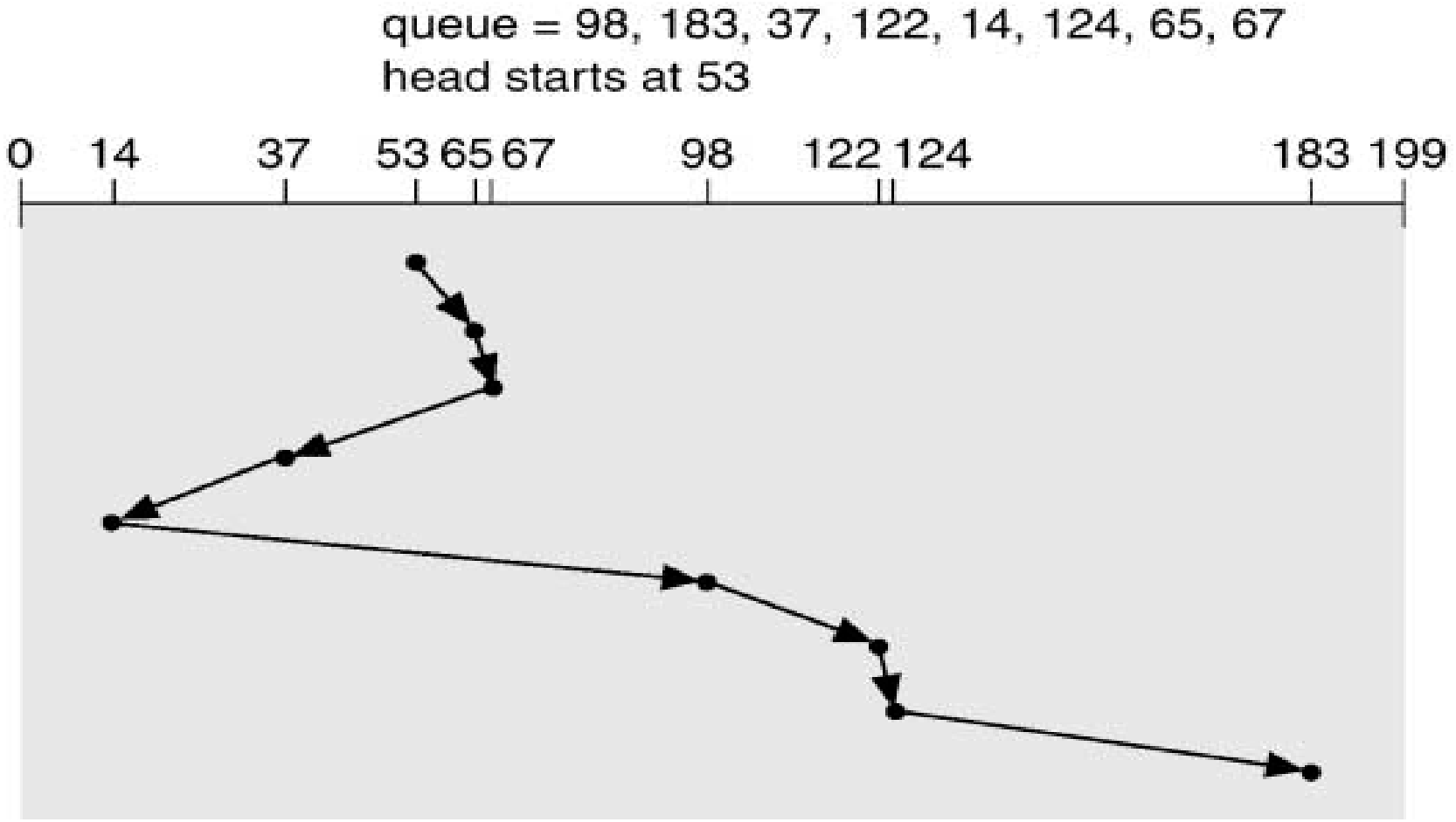
Illustration shows total head movement of 640 cylinders



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.

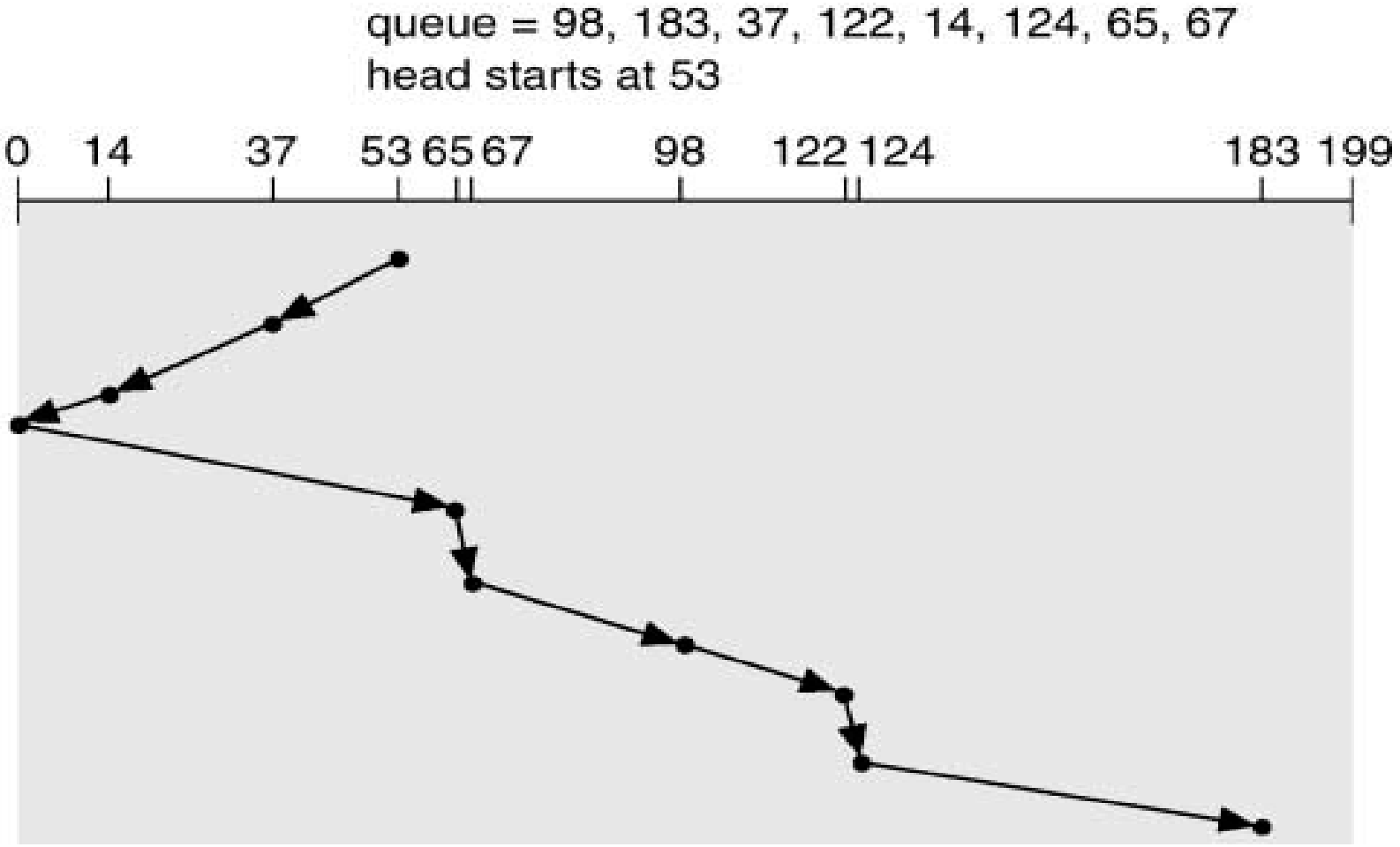
SSF (Cont.)


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.

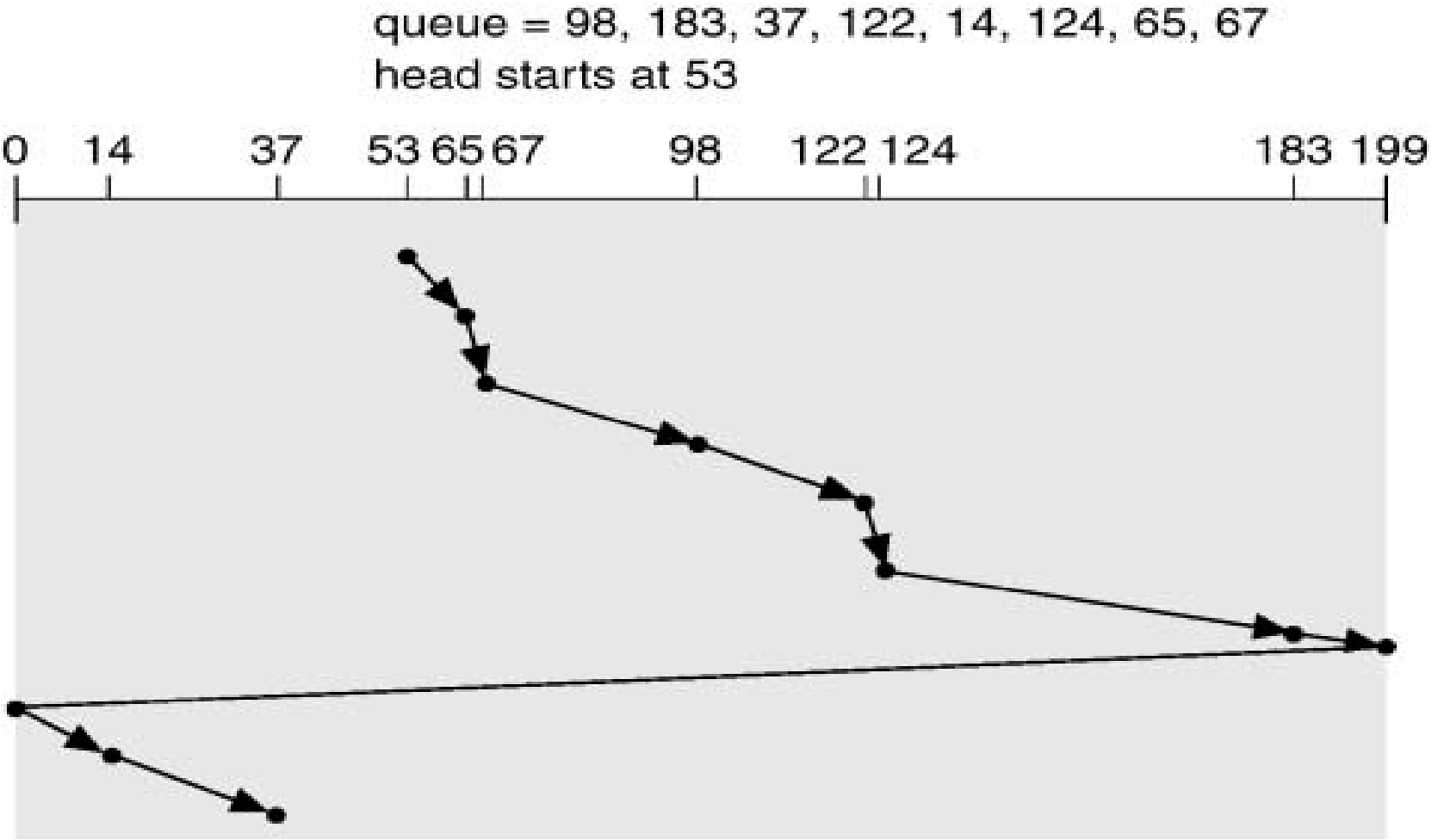
SCAN (Cont.)


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.

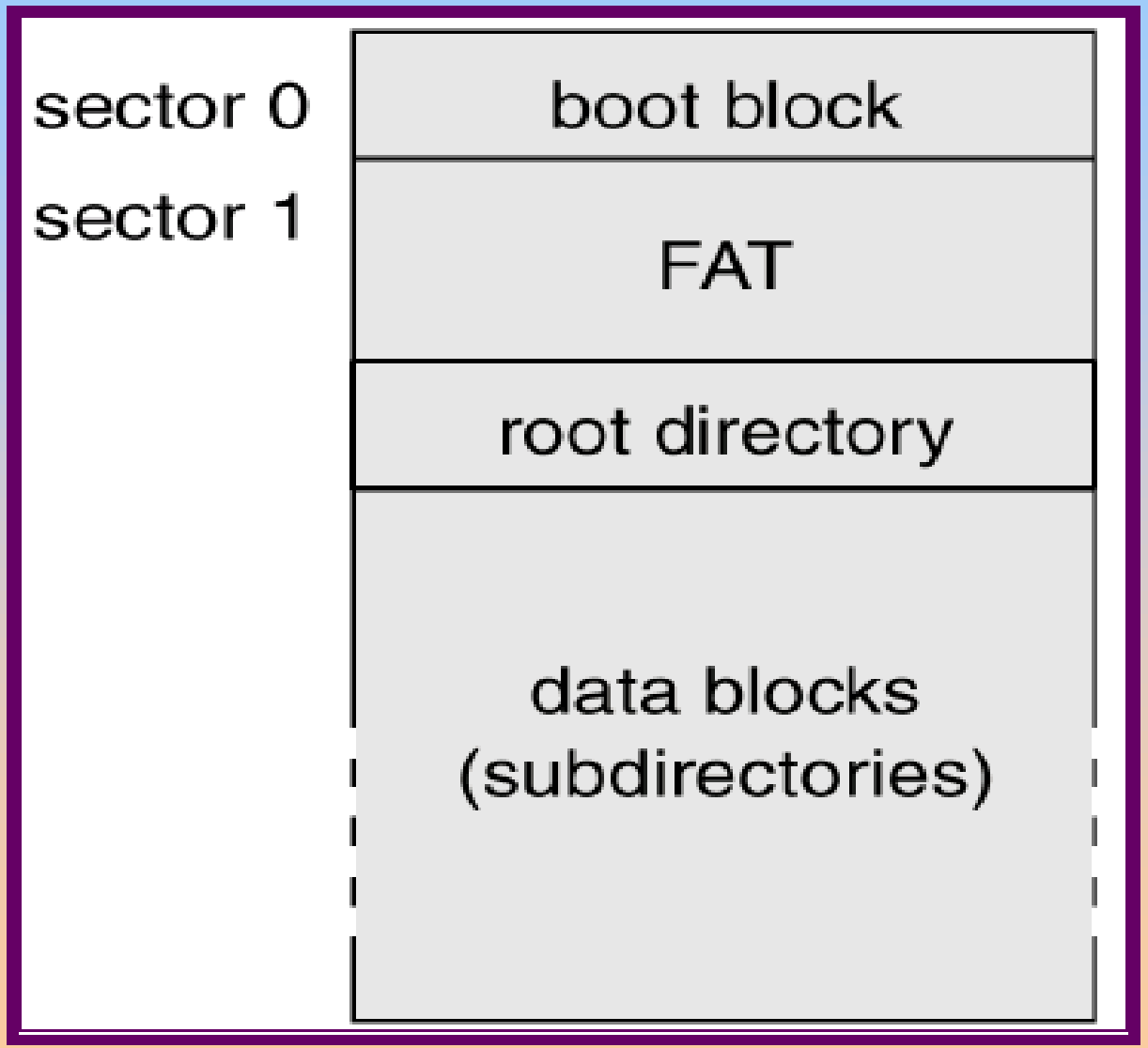
C-SCAN (Cont.)


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

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.

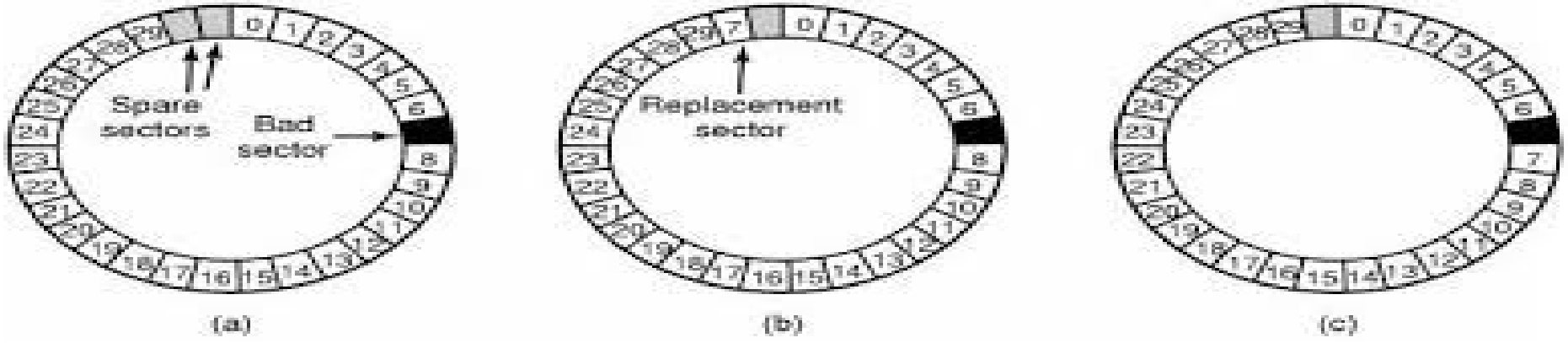
•A typical bad-sector transaction might be as follows:

- ✓ 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.

4.2.4 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

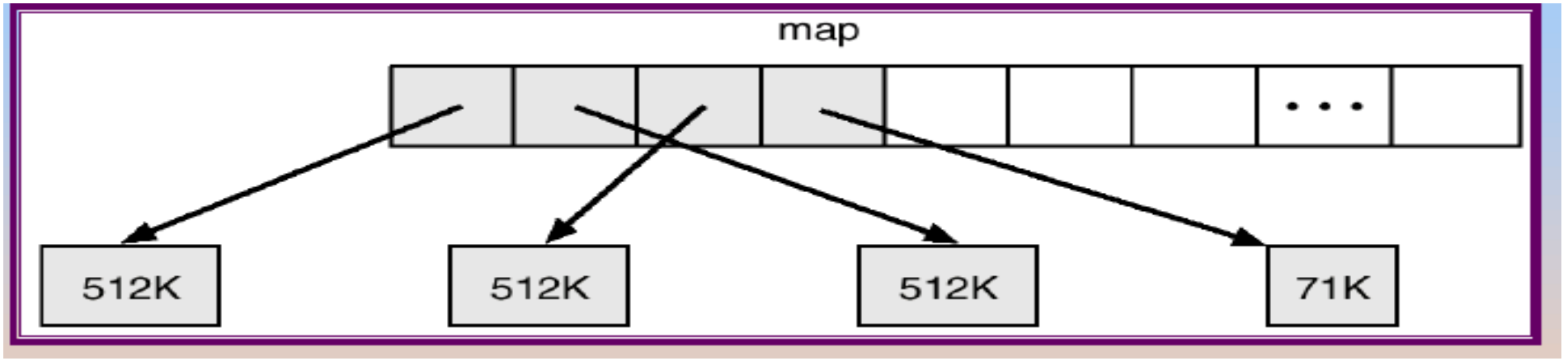


Fig1.1: 4.3BSD text-segment swap map.

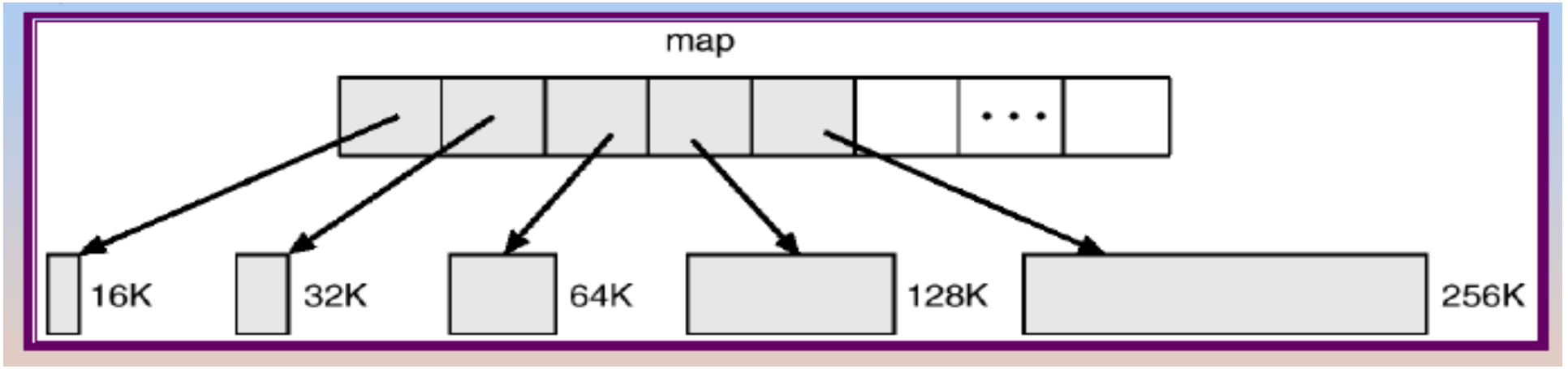


Fig 1.2: 4.3BSD data-segment swap map

- 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.

4.2.5 Stable-Storage Implementation

- Write-ahead log scheme requires stable storage
- To implement stable storage:
  - Replicate information on more than one nonvolatile storage media with independent failure modes
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery.

A disk write results in one of three outcomes:
1. Successful completion: The data were written correctly on disk.
2. Partial failure: A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted.
3. Total failure: The failure occurred before the disk write started, so the previous data values on the disk remain intact.

Stable-Storage Implementation(contd…)

- Whenever a failure occurs during writing of a block, the system needs to detect it and invoke a recovery procedure to restore the block to a consistent state. To do that, the system must maintain two physical blocks for each logical block. An output operation is executed as follows:

1. Write the information onto the first physical block.
2. When the first write completes successfully, write the same information onto the second physical block.
3. Declare the operation complete only after the second write completes successfully.

- During recovery from a failure, each pair of physical blocks is examined. If both are the same and no detectable error exists, then no further action is necessary. If one block contains a detectable error then we replace its contents with the value of the other block. If neither block contains a detectable error, but the blocks differ in content, then we replace the content of the first block with that of the second. This recovery procedure ensures that a write to stable storage either succeeds completely or results in no change.

4.3.6 Tertiary Storage Devices

- Low cost is the defining characteristic of tertiary storage
- Generally, tertiary storage is built using removable media
- Common examples of removable media are floppy disks and CD-ROMs; other types are available

Removable Disks

- Floppy disk — thin flexible disk coated with magnetic material, enclosed in a protective plastic case
  - Most floppies hold about 1 MB; similar technology is used for removable disks that hold more than 1 GB
  - Removable magnetic disks can be nearly as fast as hard disks, but they are at a greater risk of damage from exposure
- A magneto-optic disk records data on a rigid platter coated with magnetic material
  - Laser heat is used to amplify a large, weak magnetic field to record a bit
  - Laser light is also used to read data (Kerr effect)
  - The magneto-optic head flies much farther from the disk surface than a magnetic disk head, and the magnetic material is covered with a protective layer of plastic or glass; resistant to head crashes
- Optical disks do not use magnetism; they employ special materials that are altered by laser light

WORM Disks

- The data on read-write disks can be modified over and over
- WORM ("Write Once, Read Many Times") disks can be written only once
- Thin aluminum film sandwiched between two glass or plastic platters
- To write a bit, the drive uses a laser light to burn a small hole through the aluminum; information can be destroyed by not altered
- Very durable and reliable
- Read-only disks, such ad CD-ROM and DVD, com from the factory with the data pre-recorded

Tapes

- Compared to a disk, a tape is less expensive and holds more data, but random access is much slower.
- Tape is an economical medium for purposes that do not require fast random access, e.g., backup copies of disk data, holding huge volumes of data.
- Large tape installations typically use robotic tape changers that move tapes between tape drives and storage slots in a tape library
    - stacker – library that holds a few tapes
    - silo – library that holds thousands of tapes
- A disk-resident file can be archived to tape for low cost storage; the computer can stage it back into disk storage for active use.

Operating System Support

- Major OS jobs are to manage physical devices and to present a virtual machine abstraction to applications
- For hard disks, the OS provides two abstraction:
  - Raw device – an array of data blocks
  - File system – the OS queues and schedules the interleaved requests from several applications

Application Interface

- Most OSs handle removable disks almost exactly like fixed disks — a new cartridge is formatted and an empty file system is generated on the disk
- Tapes are presented as a raw storage medium, i.e., and application does not open a file on the tape, it opens the whole tape drive as a raw device
- Usually the tape drive is reserved for the exclusive use of that application
- Since the OS does not provide file system services, the application must decide how to use the array of blocks
- Since every application makes up its own rules for how to organize a tape, a tape full of data can generally only be used by the program that created it

Tape Drives

- The basic operations for a tape drive differ from those of a disk drive.
- ***locate*** positions the tape to a specific logical block, not an entire track (corresponds to ***seek***).
- The ***read position*** operation returns the logical block number where the tape head is.
- The ***space*** operation enables relative motion.
- Tape drives are “append-only” devices; updating a block in the middle of the tape also effectively erases everything beyond that block.
- An EOT mark is placed after a block that is written.

File Naming

- The issue of naming files on removable media is especially difficult when we want to write data on a removable cartridge on one computer, and then use the cartridge in another computer.
- Contemporary OSs generally leave the name space problem unsolved for removable media, and depend on applications and users to figure out how to access and interpret the data.
- Some kinds of removable media (e.g., CDs) are so well standardized that all computers use them the same way.

Hierarchical Storage Management (HSM)

- A hierarchical storage system extends the storage hierarchy beyond primary memory and secondary storage to incorporate tertiary storage — usually implemented as a jukebox of tapes or removable disks.
- Usually incorporate tertiary storage by extending the file system
  - Small and frequently used files remain on disk
  - Large, old, inactive files are archived to the jukebox
- HSM is usually found in supercomputing centers and other large installations that have enormous volumes of data.

4.3.7 I/O in UNIX

There are two main categories of I/O units in UNIX, block devices and character devices. In addition there are sockets that are used for network communication.

Block devices

• Devices that addresses blocks of a fixed size, usually disk memories.
• Data blocks are buffered in the buffer cache.
• Block devices are usually called via the file system, but are also available as special files (for example /dev/hde1).

Character devices

• Terminals and printers, but also everything else (except sockets) that do not use the block buffer cache.
• There are for example /dev/mem that is an interface to physical memory.

UNIX I/O System

• Device drivers are called via a switch table. There is one switch table for block devices and one for character devices. A hardware device is identified by its type (block or character) and a device number.
• Device numbers consist of two parts: major device number and minor device number. Major device number is used as an index in the switch table to locate the correct device driver.
• Minor device number is forwarded to the device driver and used to select correct subunit. (For example correct file system partition if the disk is divided in several partitions)

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:

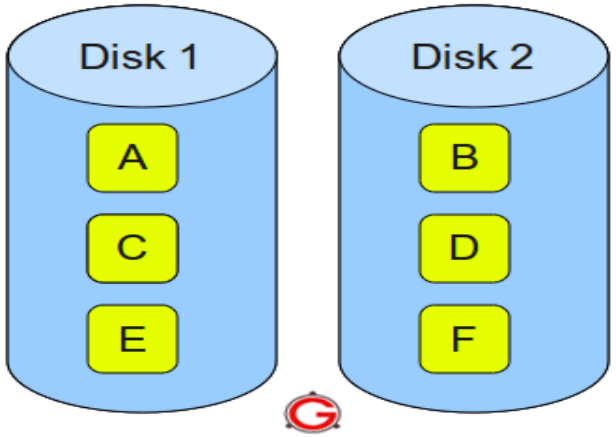
- RAID 5 is the most common secure RAID level. It requires at least 3 drives but can work with up to 16. Data blocks are striped across the drives and on one drive a parity checksum of all the block data is written. The parity data are not written to a fixed drive, they are spread across all drives, 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:

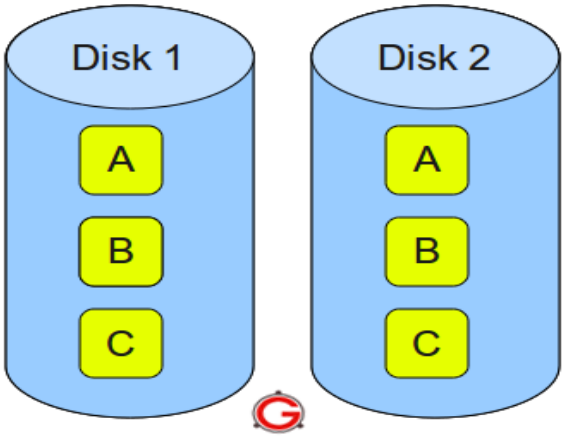
- Read data transactions are very fast while write data transactions are somewhat slower (due to the parity that has to be calculated).
- If a drive fails, you still have access to all data, even while the failed drive is being replaced and the storage controller rebuilds the data on the new drive.

Disadvantages:

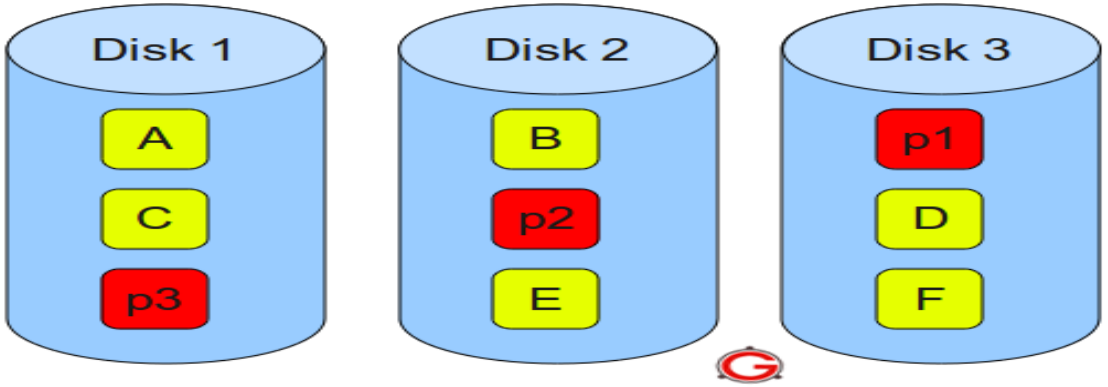
- Drive failures have an effect on throughput, although this is still acceptable.



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



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



RAID 5 – Blocks Striped. Distributed Parity.