Operating System
Unit 5: Input/Output Management

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

Introduction

- The two main jobs of a computer are I/O and processing. In many cases, the main job is I/O and the processing is merely incidental. For instance, when we browse a web page or edit a file, our immediate interest is to read or enter some information, not to compute an answer.
- It must issue commands to the devices, catch interrupts, and handle errors. It should also provide an interface between the devices and the rest of the system that is simple and easy to use. To the extent possible, the interface should be the same for all devices. The I/O code represents a significant fraction of the total operating system.

Interrupt Handlers

- An interrupt is a signal sent by hardware or software to the processor for processing. The process that runs when an interrupt is generated is the Interrupt Handler. The CPU saves the state of the ongoing process and shifts its attention to the interrupt generated by giving access to the interrupt handler. This entire process is called interrupt handling.
- Interrupt Handler is a process that runs when an interrupt is generated by hardware or software. The interrupt handler is also known as the Interrupt Service Routine (ISR). ISR handles the request and sends it to the CPU. When the ISR is complete, the process gets resumed.
- For example, the keyboard has its interrupt handler, and the printer has its interrupt handler, and so on.

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.

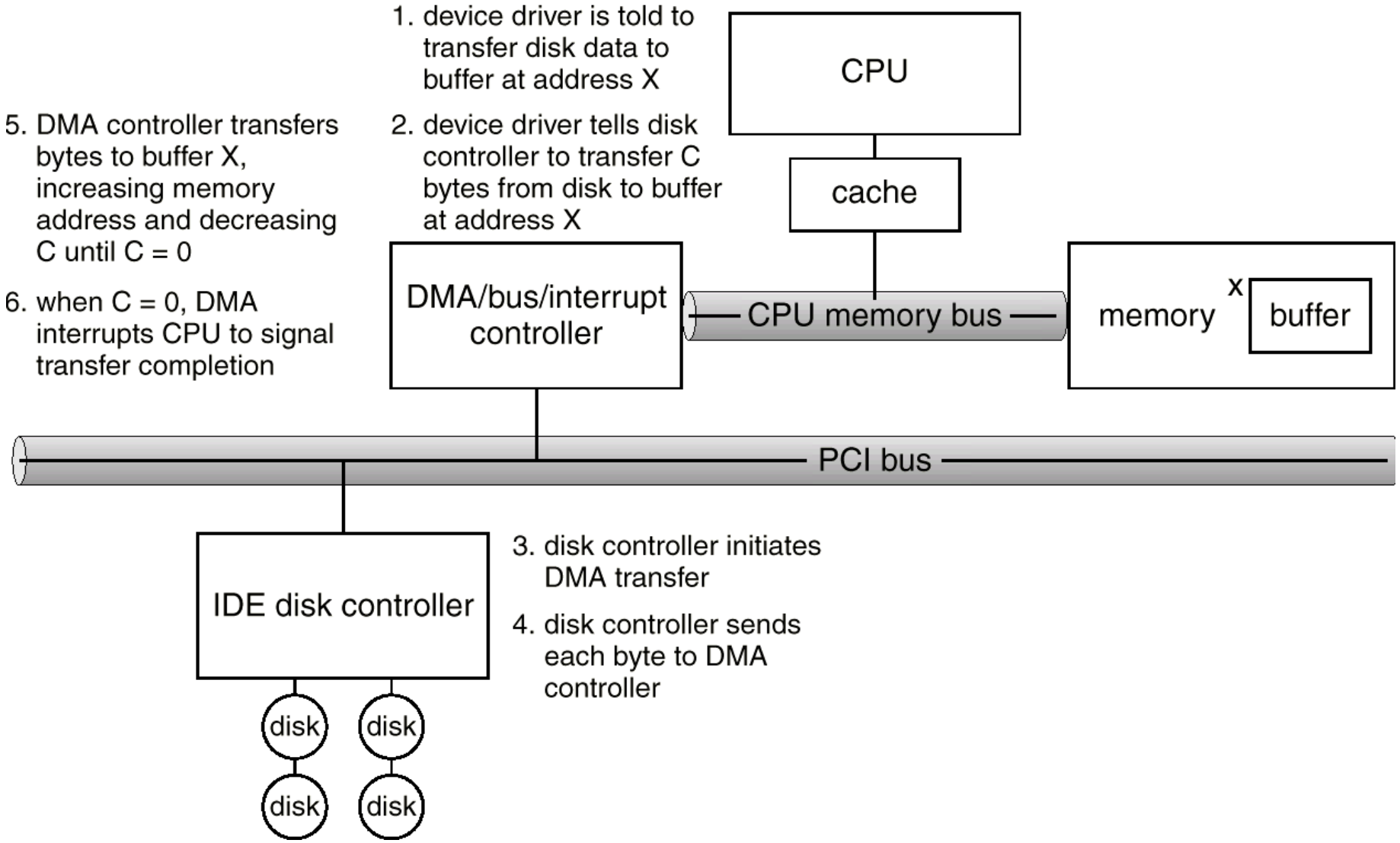
Memory –Mapped I/O

- 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.

DMA(Direct Memory Access)

- Many computers avoid burdening the main CPU with **PIO(Programmed I/O)** 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.

Six Step Process to Perform DMA Transfer



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.

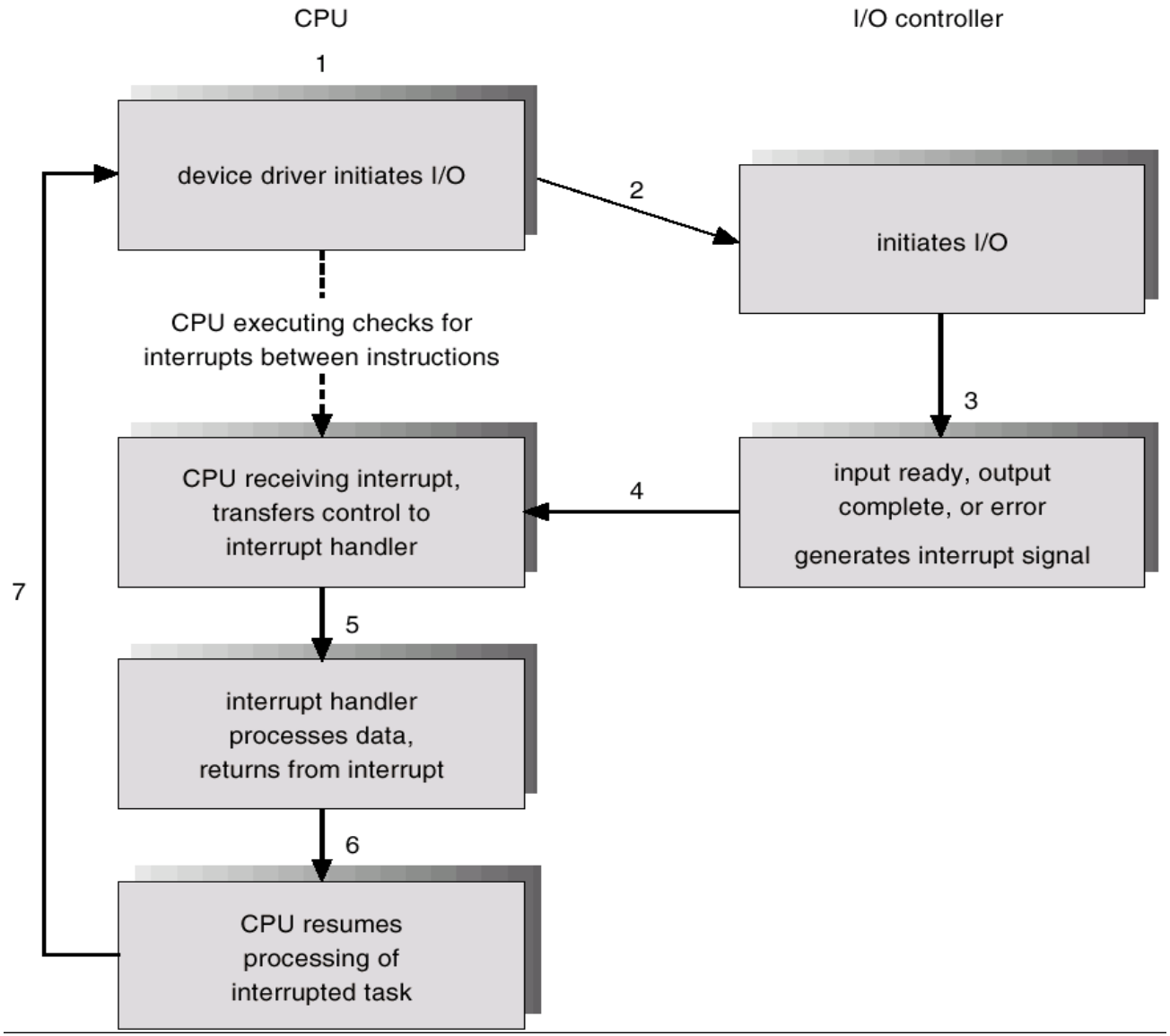
Polled I/O

- 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



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.)

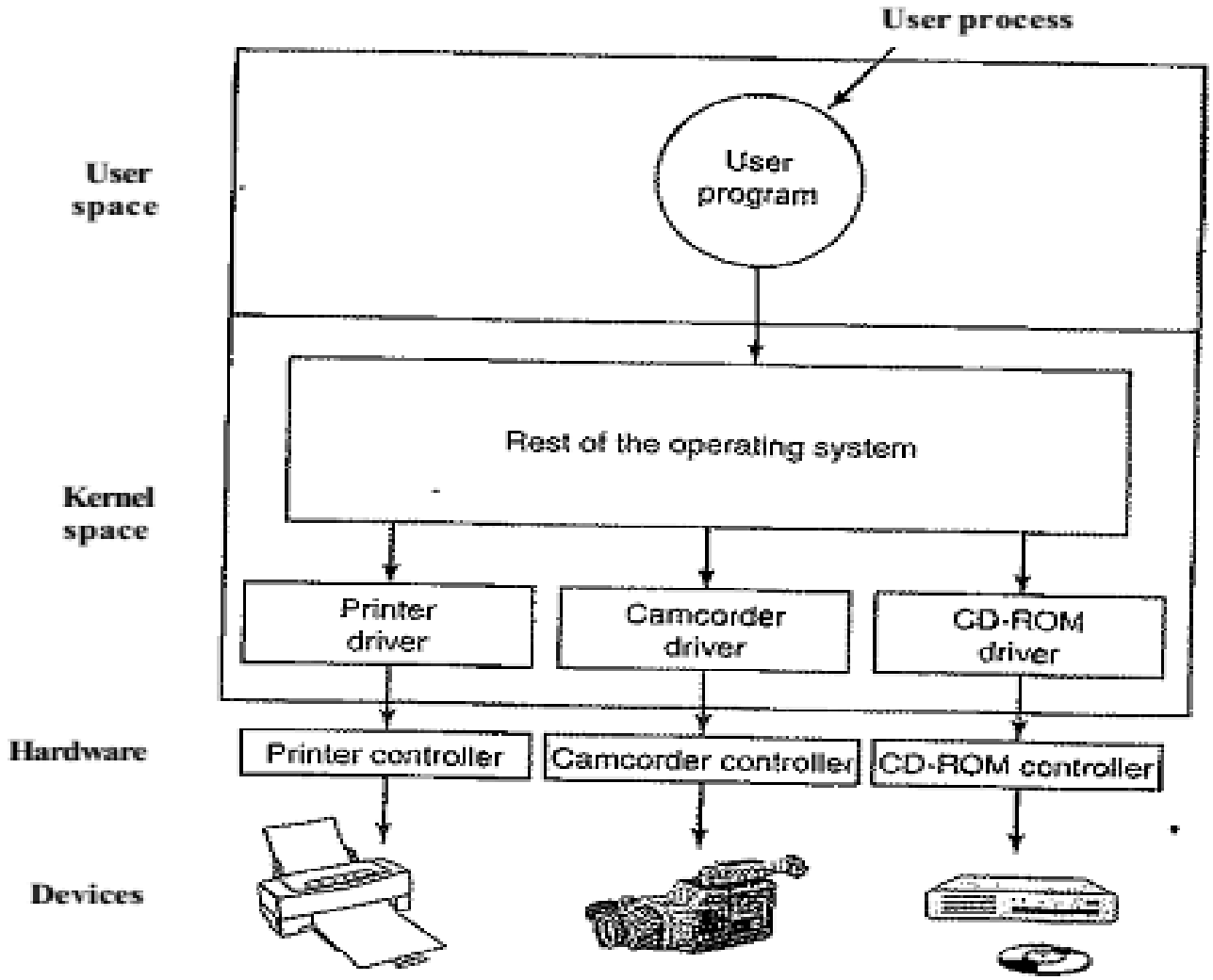


Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

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.

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 systems.

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:

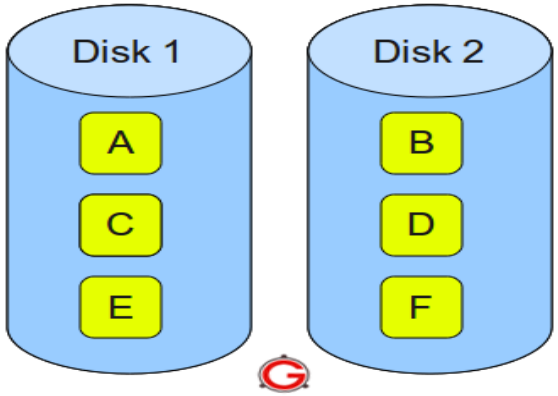
- RAID 5 is the most common secure RAID level. It requires at least 3 drives but can work with up to 16. Data blocks are striped across the drives and on one drive a parity checksum of all the block data is written. The parity data are not written to a fixed drive, they are spread across all drives,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

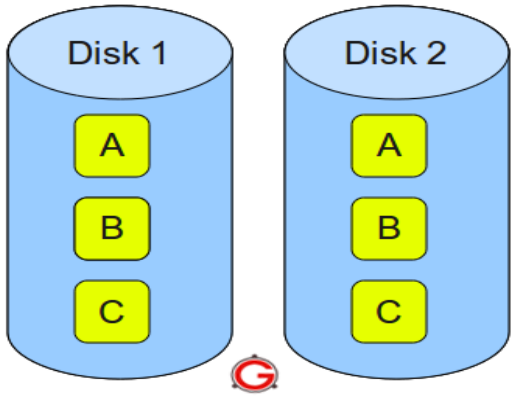
- Read data transactions are very fast while write data transactions are somewhat slower (due to the parity that has to be calculated).
- If a drive fails, you still have access to all data, even while the failed drive is being replaced and the storage controller rebuilds the data on the new drive.

Disadvantages:

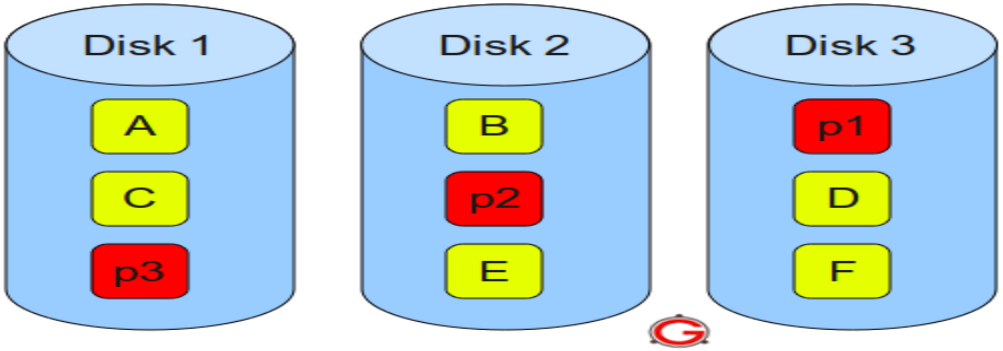
- Drive failures have an effect on throughput, although this is still acceptable.



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



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



RAID 5 – Blocks Striped. Distributed Parity.

Disk Hardware

- Disks come in a variety of types. The most common ones are the magnetic disks (hard disks and floppy disks). They are characterized by the fact that reads and writes are equally fast, which makes them ideal as secondary memory (paging, file systems, etc.). Arrays of these disks are sometimes used to provide highly reliable storage. For distribution of programs, data, and movies, various kinds of optical disks (CD-ROMs, CD-Recordables, and DVDs) are also important.

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

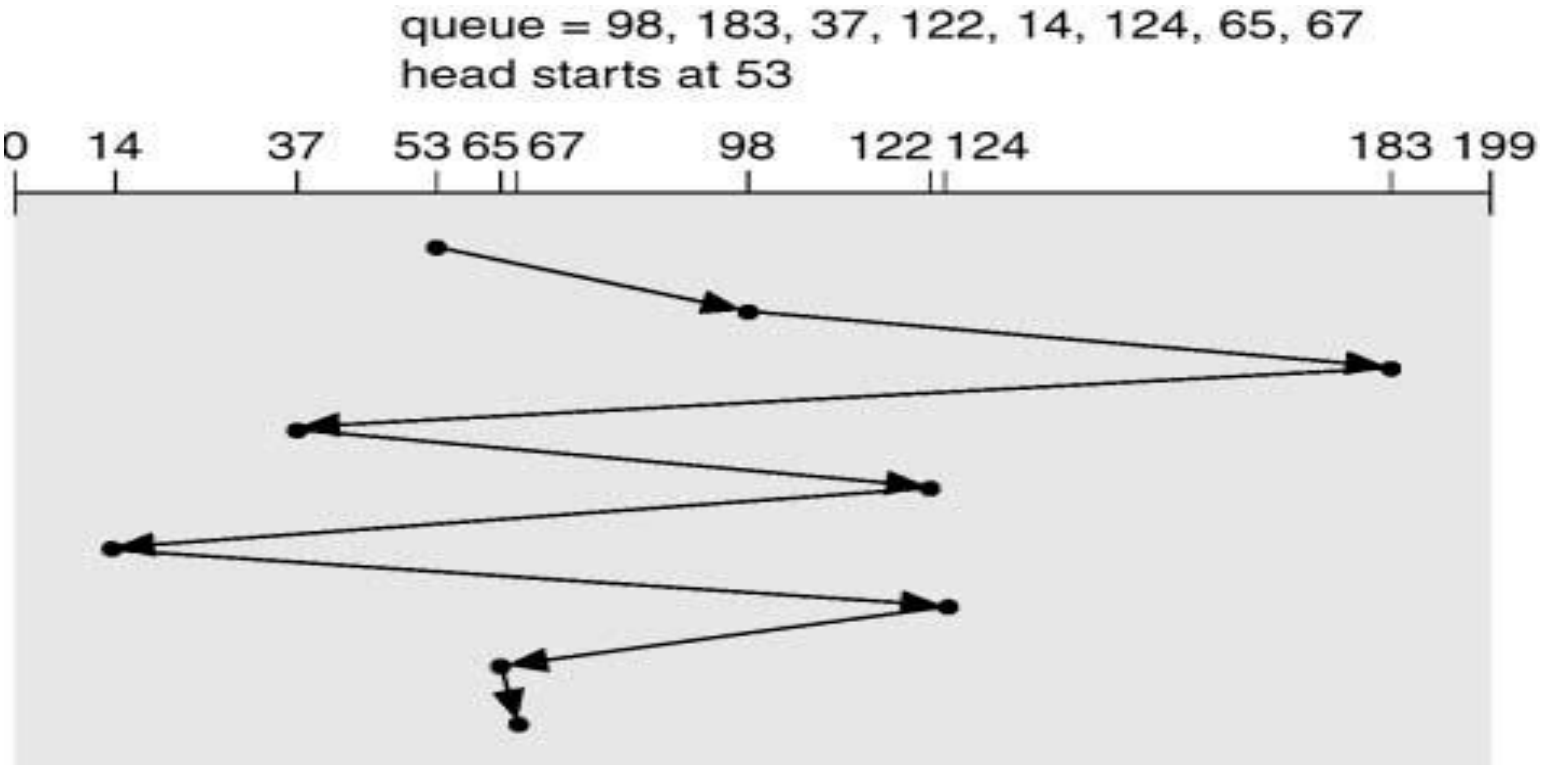
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

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

Head pointer 53

FCFS

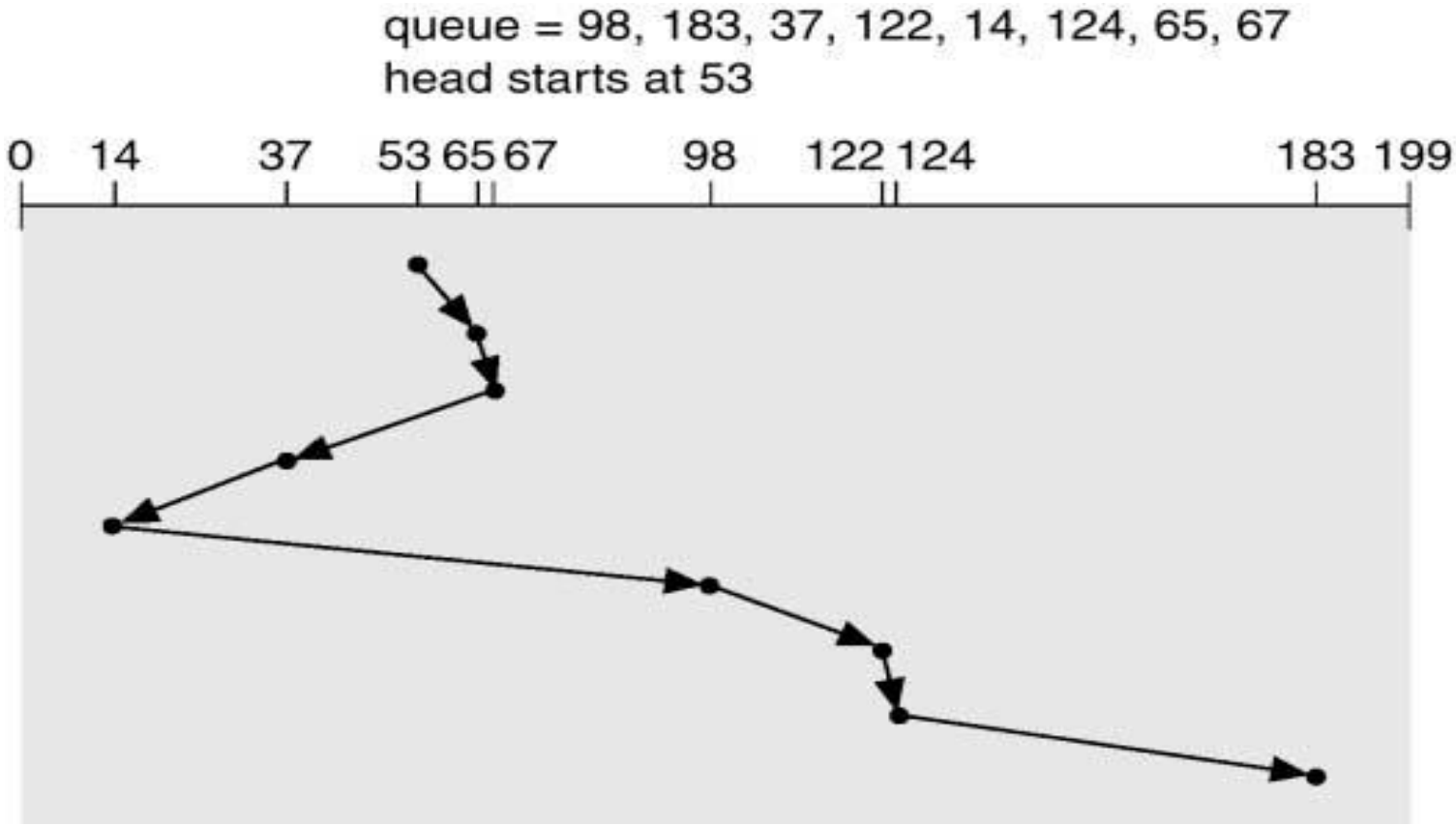
Illustration shows total head movement of 640 cylinders.



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.

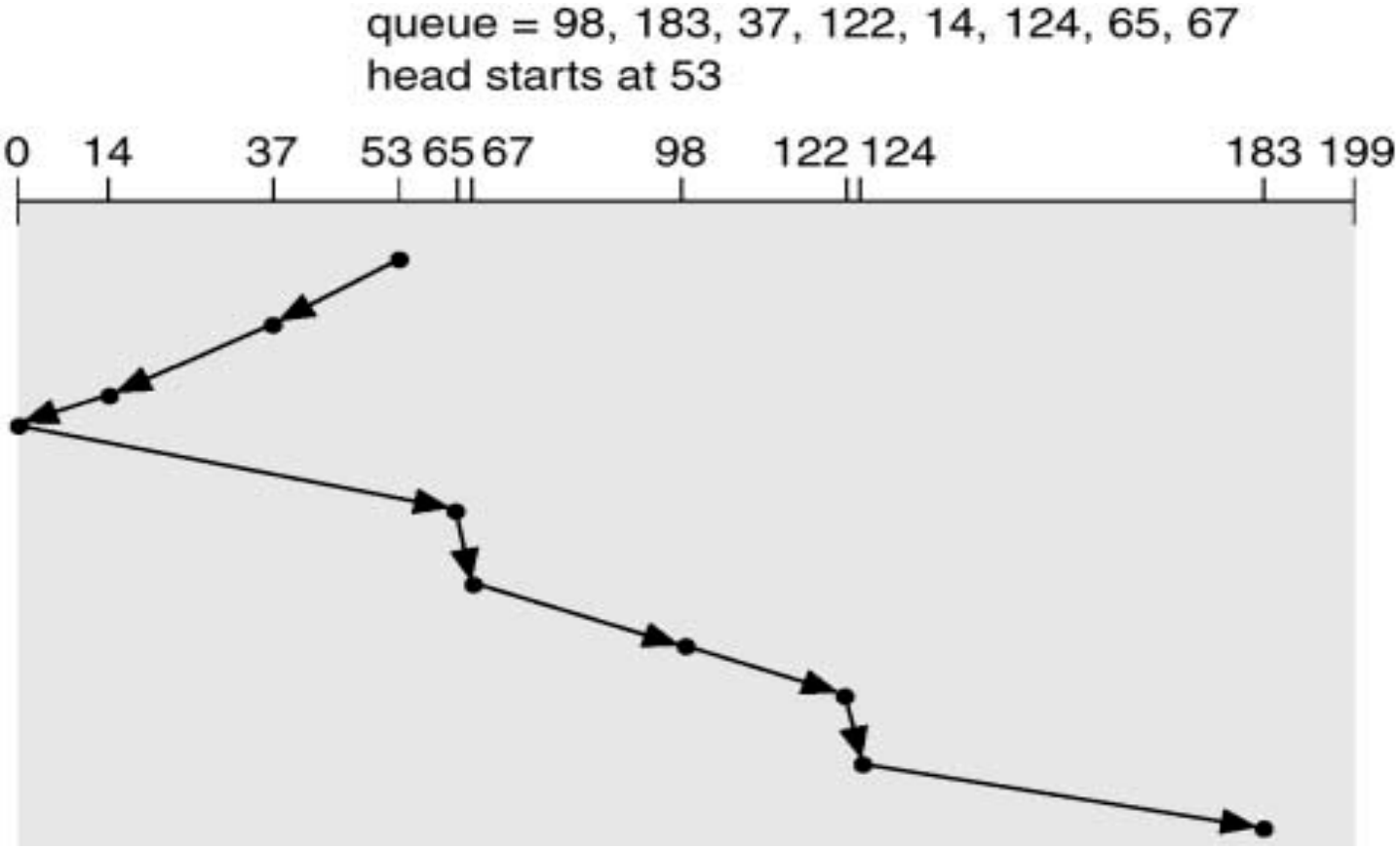
SSF (Cont.)


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.

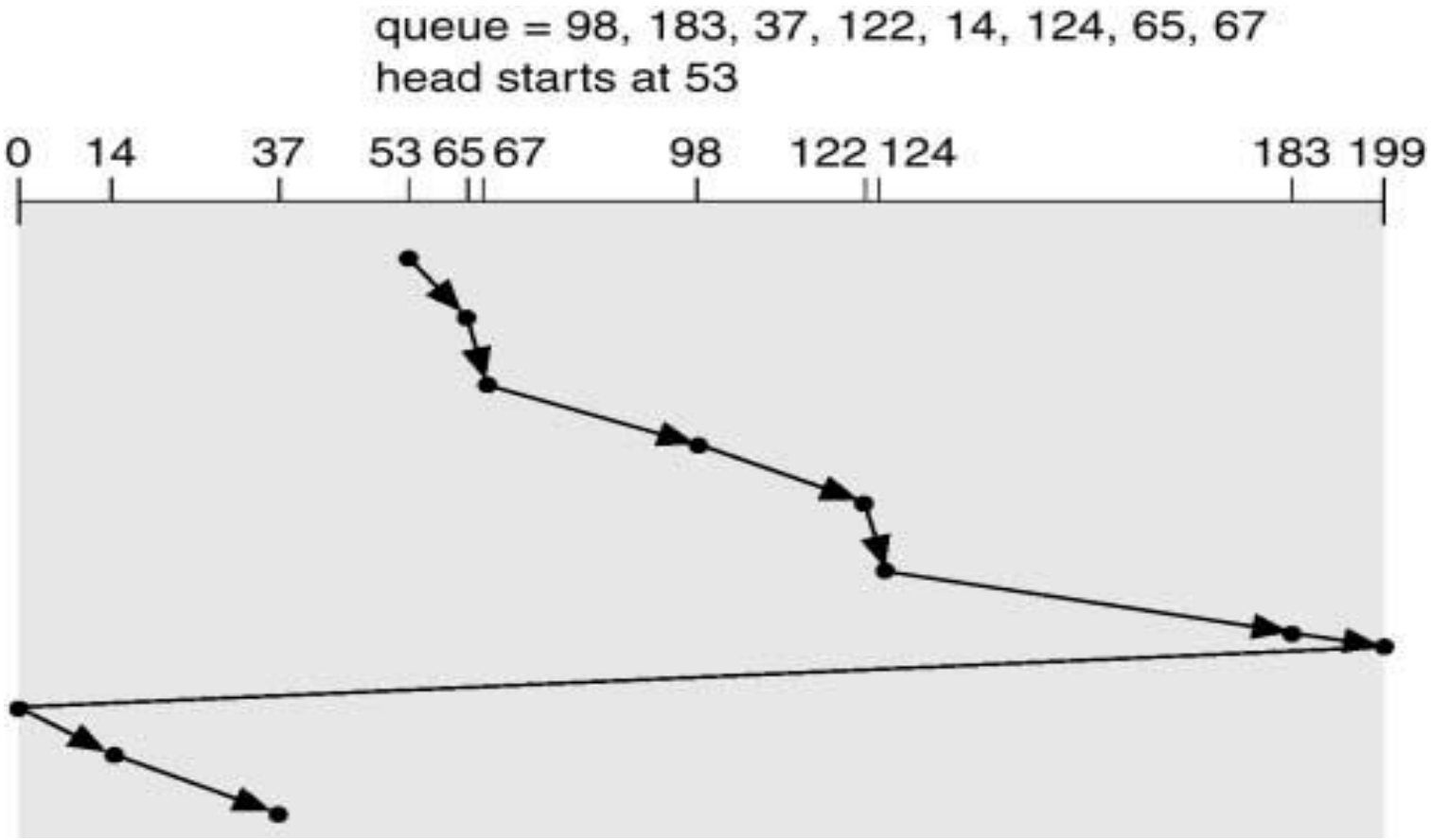
SCAN (Cont.)


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

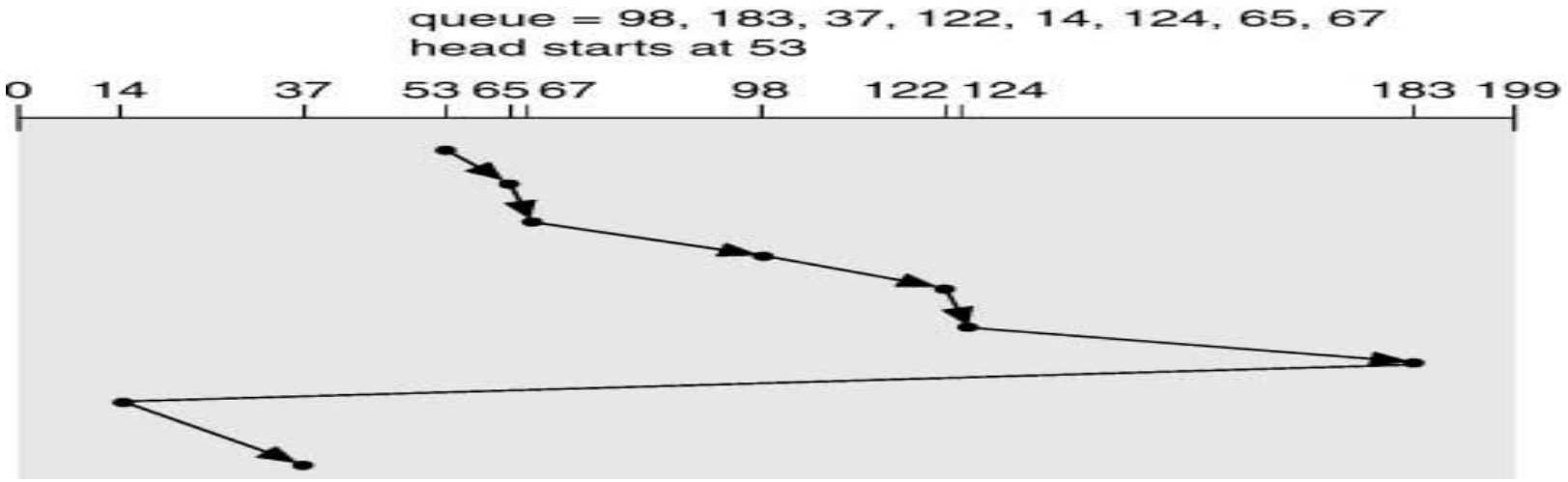
- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders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



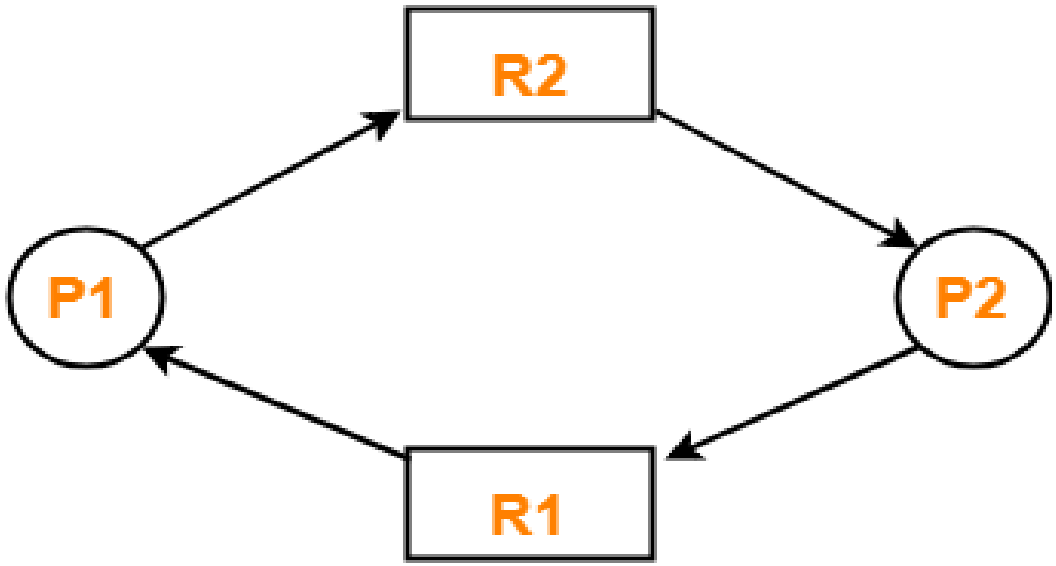
Deadlocks

Introduction:

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock

OR

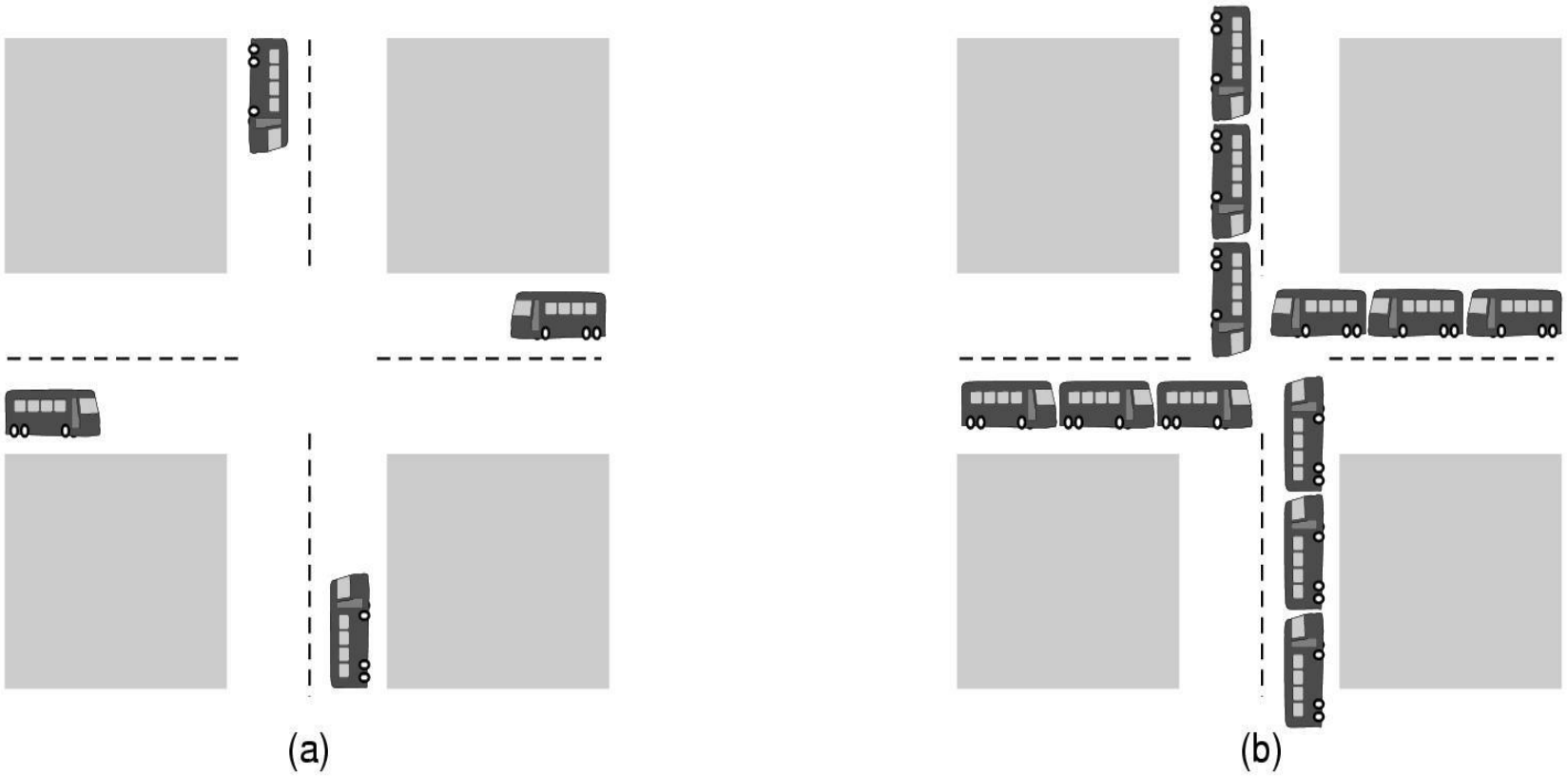
- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
  - ✦ System has 2 tape drives.
  - ✦ P1 and P2 each hold one tape drive and each needs another one.



Example of a deadlock

Bridge Crossing Example

- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.



Preemptable and Nonpreemptable Resources

- A **Preemptable** resource is one that can be taken away from the process owning it with no ill effects. Such types of resources are reusable.
- Memory, printers, tape drives are example of Preemptable resources.

- A **Nonpreemptable** resource, in contrast, is one that cannot be taken away from its current owner without causing the computation to fail.
- If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD.

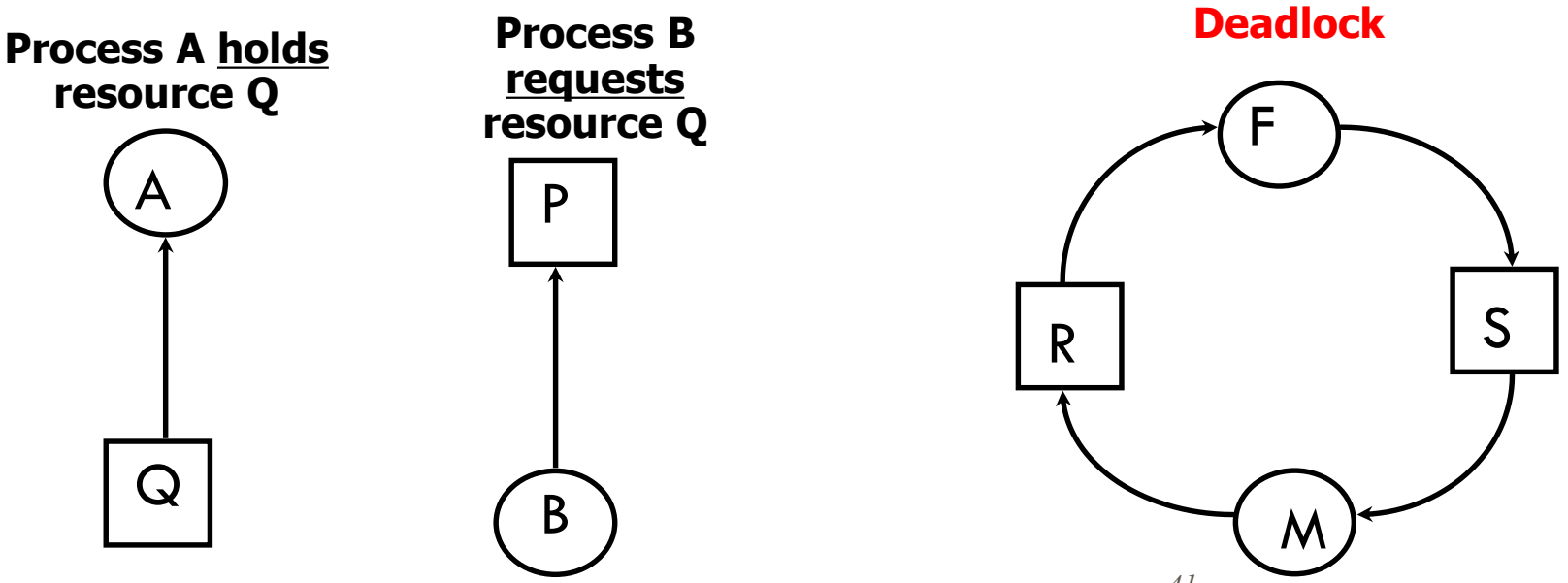
Conditions for Deadlock

Deadlock can arise if four conditions hold simultaneously.

- Mutual exclusion: only one process at a time can use a resource.
- Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes.
- No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- Circular wait: there exists a set {P0, P1, ..., P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ..., Pn–1 is waiting for a resource that is held by Pn, and P0 is waiting for a resource that is held by P0.

Modeling deadlocks

- modeled by a ***directed graph*** (resource graph)
  - Requests and assignments as *directed edges*
  - Processes and Resources as vertices
- ***Cycle*** in graph means ***deadlock***

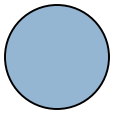
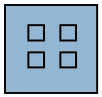


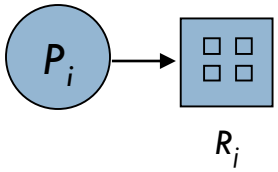
Resource-Allocation Graph (RAG)

A set of vertices V and a set of edges E

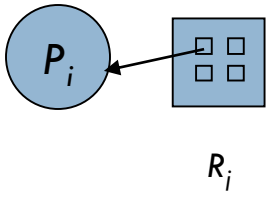
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- Two types of E:
  - request edge – directed edge $P_i \rightarrow R_j$
  - assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

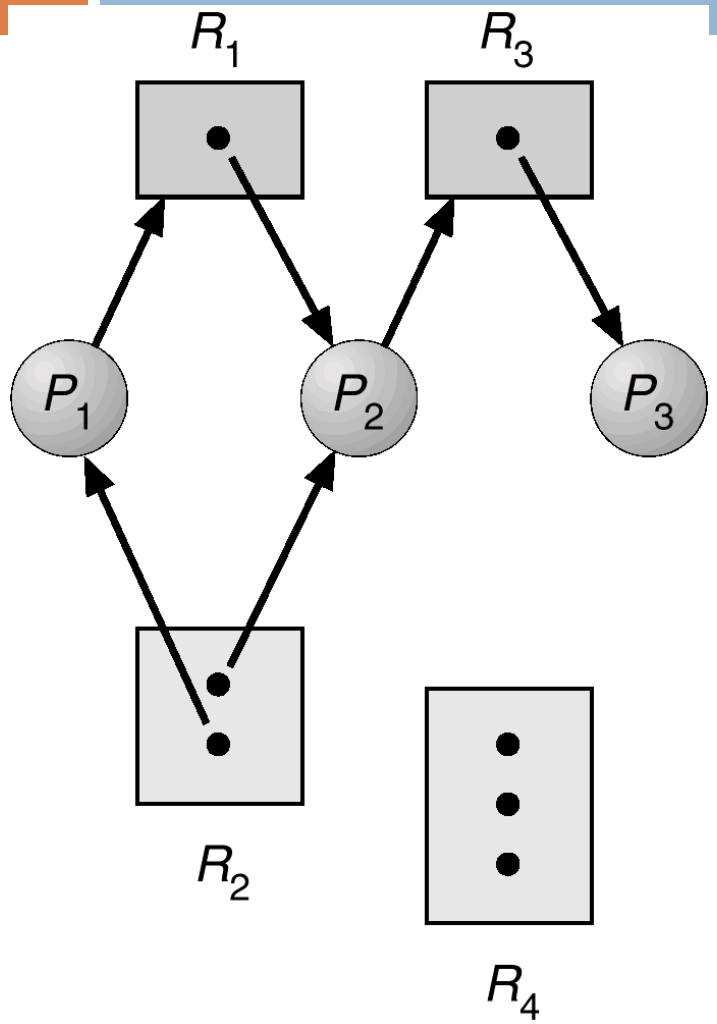
- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_i



- P_i is holding an instance of R_i



Multiple Resources of each Type



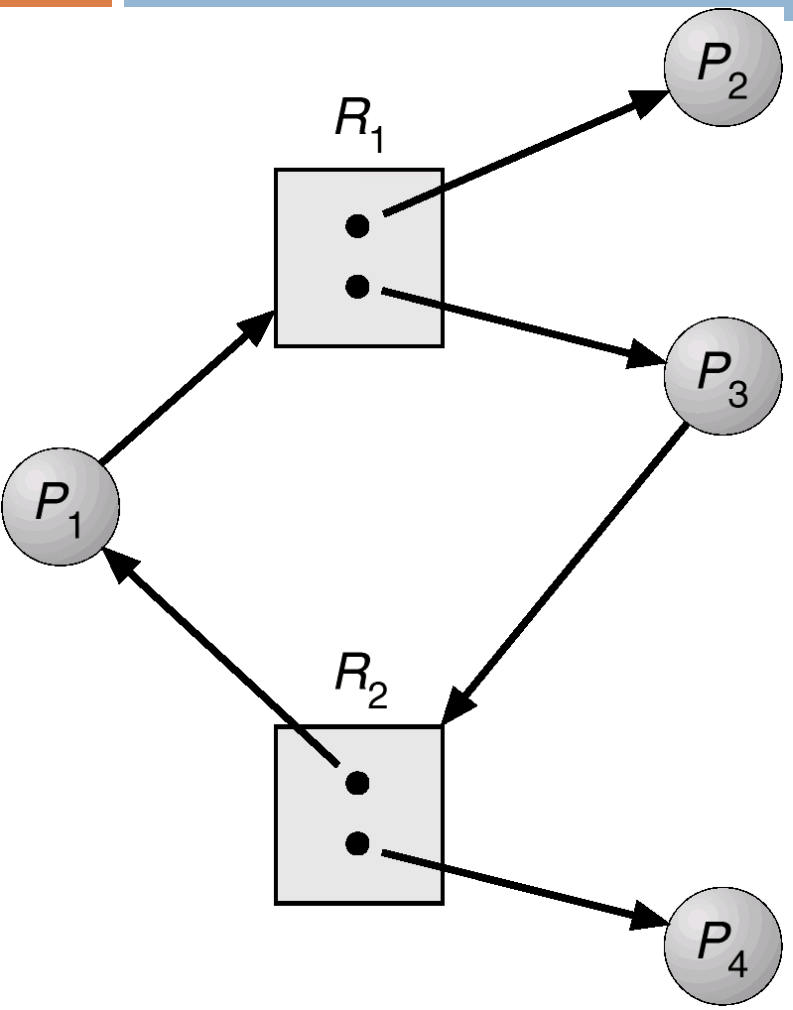
The sets P, R and E:

- P == {P1, P2, P3}
- R== {R1, R2, R3, R4}
- E == $\{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$
- Resource instances:
  - One instance of resource type R1
  - Two instances of resource type R**2**
  - One instance of resource type R3
  - Three instances of resource type R4

Process states:

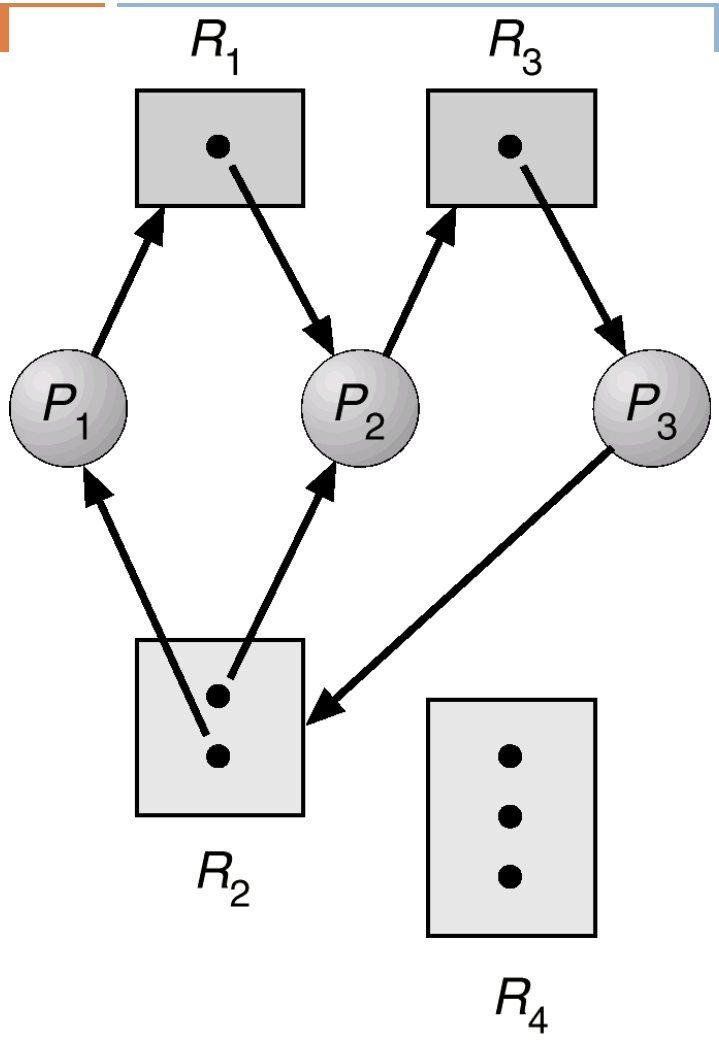
- Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
- Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
- Process **P**3 is holding an instance of R3 .

A Directed Cycle But No Deadlock



- We have a cycle $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- However, there is no deadlock.
- Observe that process $\mathbf{P_4}$ may release its instance of resource type R_2. That resource can then be allocated to P_3, breaking the cycle.

Resource Allocation Graph With A Deadlock



Two minimal cycles exist in the system:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

- Processes P_1, P_2, and P_3 are deadlocked.
- Process P_2 is waiting for the resource R_3, which is held by process P_3.
- Process P_3 is waiting for either process P_1 or process P_2 to release resource R_2.
- In addition, process P_1 is waiting for process P_2 to release resource R_1
- Deadlock Exist

Basic Facts

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, deadlock possible.

- If a resource-allocation graph does not have a cycle, then the system is not in a deadlocked state.
- If there is a cycle, then the system may or may not be in a deadlocked state.
- This observation is important when we deal with the deadlock problem.

Methods for Handling Deadlocks

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then Recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX**(Ostrich Algorithm)**.

Deadlock Ignorance(Ostrich Algorithm):

- Stick your head in the sand and pretend there is no problem at all, this method of solving any problem is called Ostrich Algorithm. This Ostrich algorithm is the most widely used technique in order to ignore the deadlock and also it used for all the single end-users uses. If there is deadlock in the system, then the OS will reboot the system in order to function well. E.g: Unix and Windows OS.

Advantages:

1.Simplicity: Ignoring the possibility of deadlock can make the design and implementation of the operating system simpler and less complex.

2.Performance: Avoiding deadlock detection and recovery mechanisms can improve the performance of the system, as these mechanisms can consume significant system resources.

Disadvantages:

1.Unpredictability: Ignoring deadlock can lead to unpredictable behavior, making the system less reliable and stable.

2.System crashes: If a deadlock does occur and the system is not prepared to handle it, it can cause the entire system to crash, resulting in data loss and other problems.

3.Reduced availability: Deadlocks can cause processes to become blocked, which can reduce the availability of the system and impact user experience.

Deadlock Prevention

As we know, for a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these conditions cannot hold i.e eliminate any one of following conditions, we can prevent the occurrence of a deadlock.

- **Denying Mutual Exclusion-** The mutual-exclusion condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. Sharable resources, on the other hand, do not require mutually exclusive access, and thus cannot be involved in a deadlock
- **Denying Hold and Wait –** must guarantee that whenever a process requests a resource, it does not hold any other resources.
  - ✦ Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
  - ✦ Disadvanages: Low resource utilization; starvation possible.

Deadlock Prevention (Cont.)

- **Denying No-Preemption –**
  - ✦ If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
  - ✦ Preempted resources are added to the list of resources for which the process is waiting.
  - ✦ Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

- **Denying Circular Wait –** impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance(Banker's Algorithm)

Possible side effects of preventing deadlocks by Deadlock-prevention algorithms method, however, are low device utilization and reduced system throughput.

An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested

- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.
- For safe State:

  need <= Available

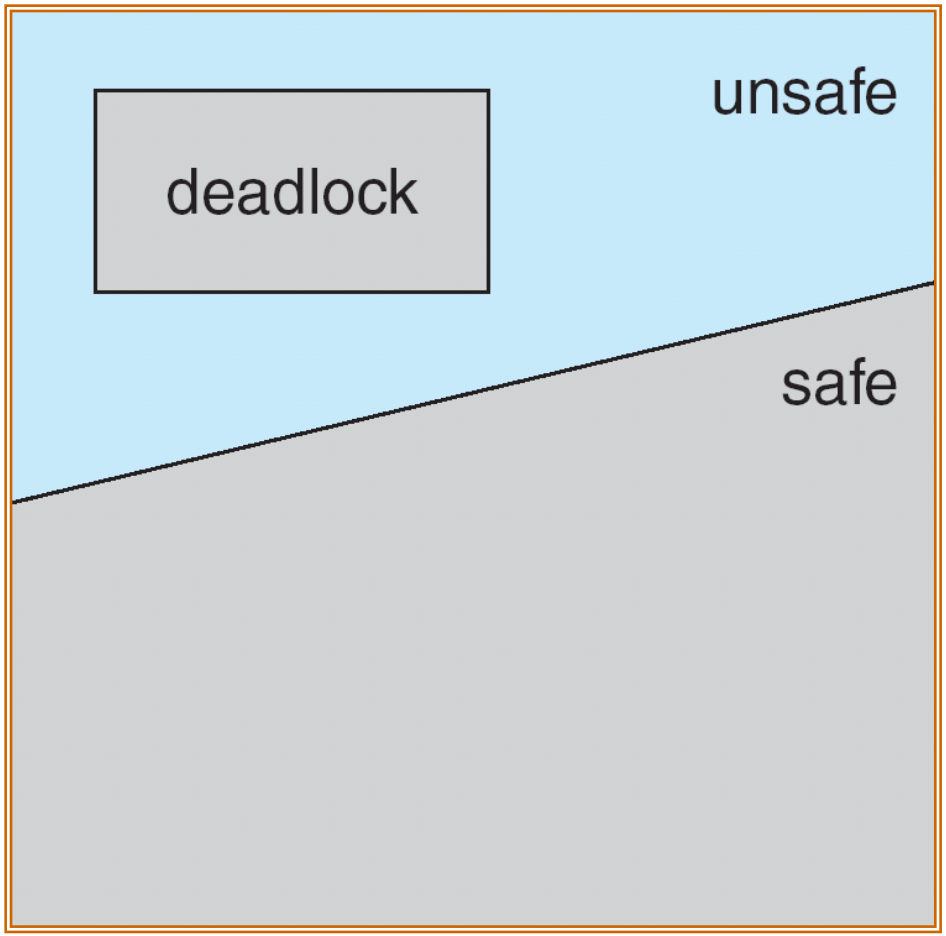
Safe State

- The system must decide if immediate allocation of a resource leaves the system in a safe state
- System is in safe state if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of processes in the system such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all P_j with $j < i$
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.
- If no such sequence exists, then the system state is said to be unsafe.

Basic Facts

- If a system is in safe state ⇒ no deadlocks
- If a system is in unsafe state ⇒ possibility of deadlock
- Avoidance ⇒ ensure that a system will never enter an unsafe state

Safe, Unsafe , Deadlock State



Safe and Unsafe States

Lets Consider that we have altogether 10 resources for 3 processes:

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1
(b)

	Has	Max
A	3	9
B	0	—
C	2	7

Free: 5
(c)

	Has	Max
A	3	9
B	0	—
C	7	7

Free: 0
(d)

	Has	Max
A	3	9
B	0	—
C	0	—

Free: 7
(e)

Sequence of run:

1. Process B take 2 resource and leads to figure (b). (Free=1)
2. Process B completes, obtain (c). (Free = 5)
3. Scheduler run process C (Take 5 resource) and leading to (d). (Free= 0)
4. When process completes, get (e). (Free=7)
5. Now process A can get 6 instances.
6. Thus (a) is safe state.

Figure: Demonstration that the state in (a) is safe

Safe and Unsafe States

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Figure: Demonstration that the sate in (b) is not safe

Note that: *An unsafe state is not a deadlocked state*. System can run for a while. From a safe state, the system can guarantee that all processes will finish; from an unsafe state, no such guarantee can be given.

Example of Banker's Algorithm

- 5 processes: P_0 through P_4
  3 resource types: A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Example (Cont.)

- The matrix *Need* is defined to be *Max* – *Allocation*

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0>$ satisfies safety criteria

Practice Numerical

1

Q. Consider the following Snapshot of a system:

	Allocation	Max	Available
	ABCD	ABCD	ABCD
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

Answer the following questions using the Banker's algorithm:

1. What is the content of the matrix need?
2. Is the system in a safe state?
3. If the request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately?

Example

Answer of 1 & 2.

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P1	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	3	8	8	6	0	0	2	0
P4	0	0	1	4	0	6	5	6	3	14	11	8	0	6	4	2
									3	14	12	12				

Total Resource: A=3, B=14, C=12 and D=12

Sequence of Process Execution: P0,P2,P1,P3,P4 (System is safe)

Example

Answer of 3.

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	1	0	0	0	0	0	0
P1	1	4	2	0	1	7	5	0	1	1	1	2	0	3	3	0
P2	1	3	5	4	2	3	5	6	2	4	6	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	3	8	8	6	0	0	2	0
P4	0	0	1	4	0	6	5	6	3	14	11	8	0	6	4	2
									3	14	12	12				

Total Resource: A=3, B=14, C=12 and D=12

Sequence of Process Execution: P0,P2,P1,P3,P4 (System is safe)

Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - ✦ How often a deadlock is likely to occur?
  - ✦ How many processes will need to be rolled back?
    - ✓ one for each disjoint cycle

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

If preemption is required to deal with deadlocks, then three issues need to be addressed:

- Selecting a victim – Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost.

- Rollback – return to some safe state, restart process for that state.

- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Issues Related to Deadlock:

- **Two Phase Locking**
- Although both avoidance and prevention are not terribly promising in the general case.
- As an example, in many database systems, an operation that occurs fre--quently is requesting locks on several records and then updating all the locked records. When multiple processes are running at the same time, there, there is a real danger of deadlock.
- The Approach often used is called two-phase locking.
- In the first phase, the process tries to lock all the records it needs, one at a time. If it succeeds, it begins the second phase, performing its updates and releasing the locks.

Issues Related to Deadlock(contd..):

- **Non Resource Deadlocks**
- This type of deadlock can occur in communication systems, in which two or more processes communicate by sending messages.
- A common arrangement is that process A sends a request message to process B, and then blocks until B sends back a reply message.
- Suppose that the request message gets lost.
- A is blocked waiting for the reply.B is blocked waiting for a request asking it to do something. We have a deadlock. Infact, there are no resources at all in the sight.

- **Starvation**
- In a dynamic system, requests for resources happen all the time. Some policy is needed to make a decision about who gets which resource when. This policy, although seemingly reasonable, may lead to some processes never getting service even though they are not deadlocked.
- This is known as starvation

RAM Disks

- Refers to RAM that has been configured to simulate a disk drive. You can access files on a RAM disk as you would access files on a real disk. RAM disks, however, are approximately a thousand times faster than hard disk drives. They are particularly useful, therefore, for applications that require frequent disk accesses.
- Because they are made of normal RAM, RAM disks lose their contents once the computer is turned off. To use a RAM disk, therefore, you need to copy files from a real hard disk at the beginning of the session and then copy the files back to the hard disk before you turn the computer off. Note that if there is a power failure, you will lose whatever data is on the RAM disk. (Some RAM disks come with a battery backup to make them more stable.)
- A RAM disk is also called a *virtual disk* or a *RAM drive.*