FILE SYSTEM

- File concept
- File support
- Access methods
- Allocation methods
- Directory systems
- File protection
- Free space management

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.

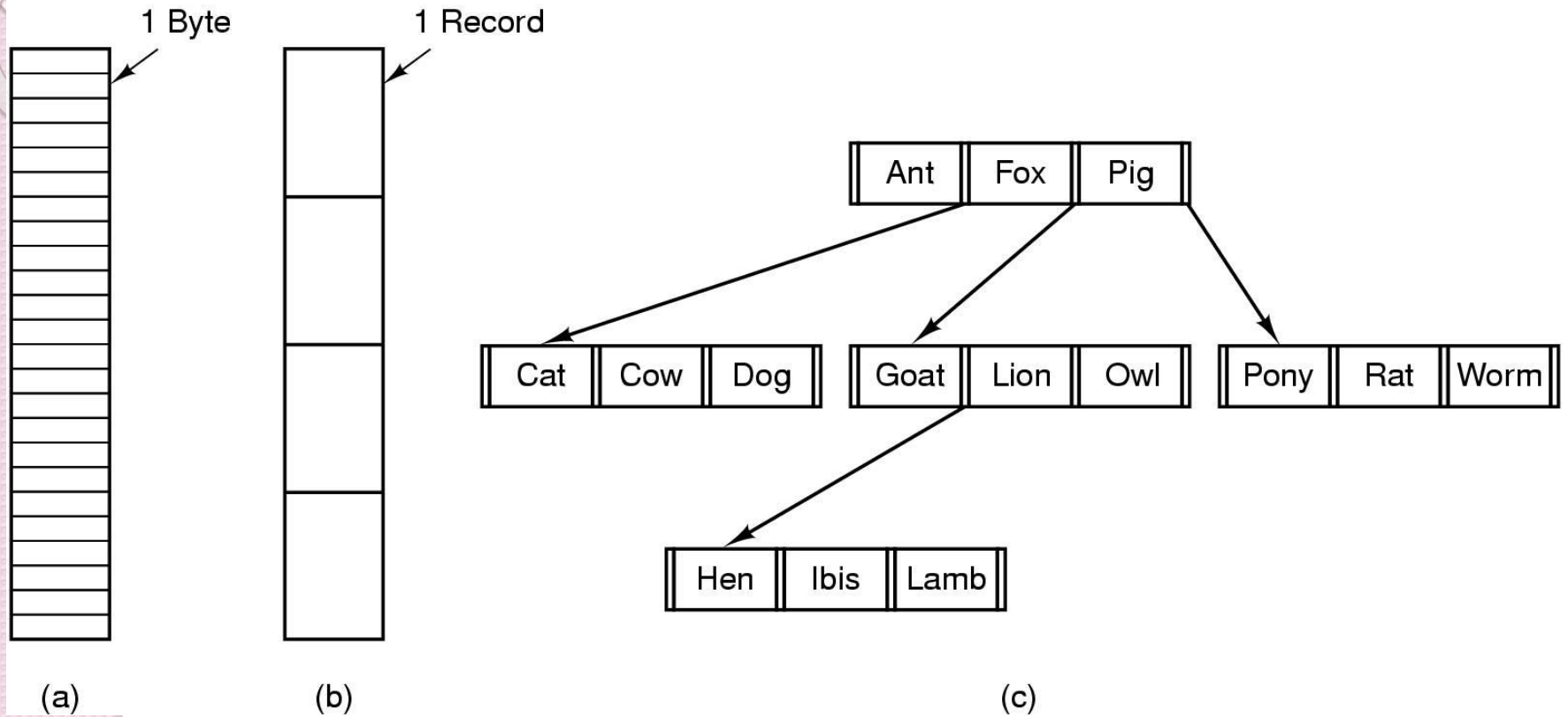
File Concept

- A file has a certain defined **structure according to its** type.
- A *text file is a sequence of characters organized into lines (and possibly* pages).
- A *source file is a sequence of subroutines and functions, each of which is* further organized as declarations followed by executable statements.
- An *object* file is a sequence of bytes organized into blocks understandable by the system's linker.
- An *executable file is a series of code sections that the loader can bring into* memory and execute.

File Structure

- A file has a certain **defined** structure, which depends on its type
- A **text file** is a sequence of characters organized into lines
- A **source code file** is a sequence of declarations, statements, and subroutine definitions
- An **object code file** and an **executable file** each contain a sequence of bytes organized into headers and tables of data and code understandable by a system linker and loader

File Structure



- Three kinds of files
  - byte sequence
  - record sequence
  - tree

File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape
- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

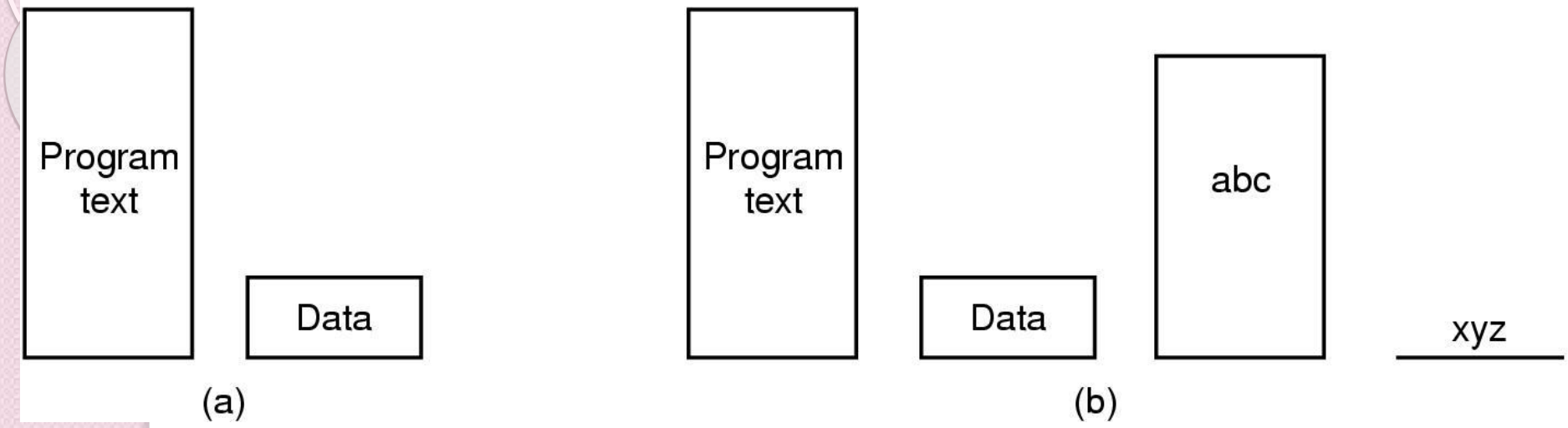
File Attributes

- **Name** – the only information kept in human-readable form
- **Identifier** – unique tag (a number) that identifies file within the file system
- **Type** – used by systems that support different types of files
- **Location** – pointer to file location on a device
- **Size** – current file size in bytes, words, or blocks
- **Protection** – access controls for who can read, write, and execute
- **Time, date, and user identification** – used for documenting file creation, last modification, and last use

File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

Memory-Mapped Files



(a) Segmented process before mapping files into its address space

(b) Process after mapping

existing file *abc* into one segment

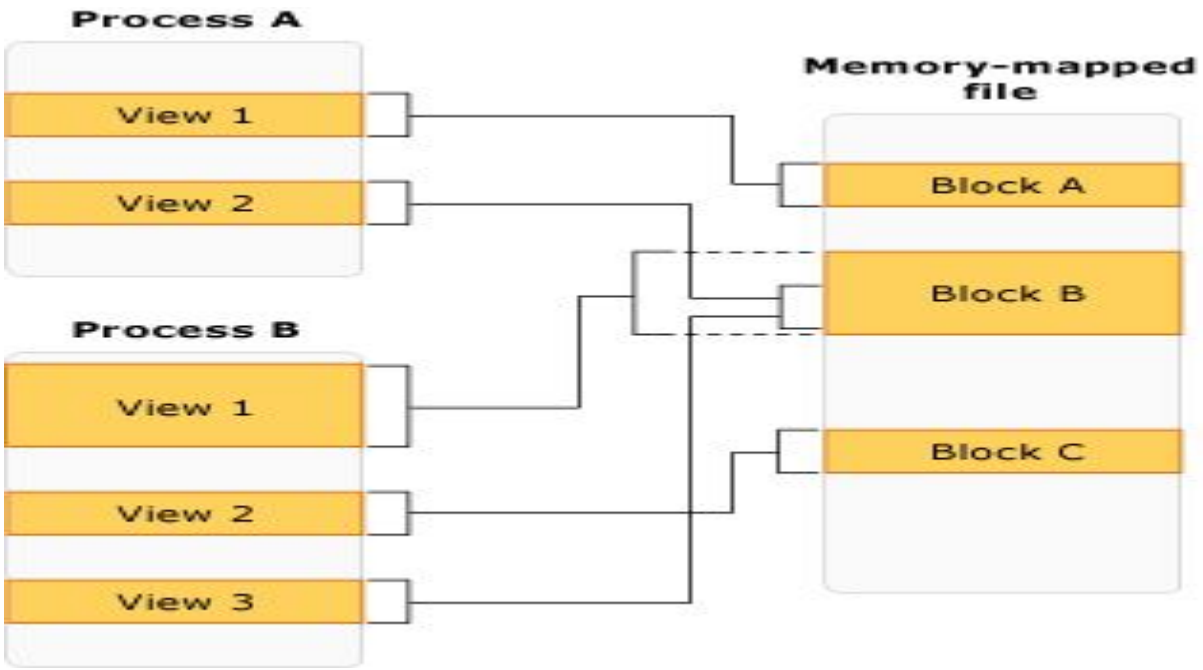
creating new segment for *xyz*

Memory-Mapped Files

- A memory-mapped file contains the contents of a file in virtual memory.
- Enables an application, including multiple processes, to modify the file by reading and writing directly to the memory.
- There are two types of memory-mapped files:
  - Persisted memory-mapped files
  - Non-persisted memory-mapped files
- Memory-mapped files can be shared across multiple processes.
- Processes can map to the same memory-mapped file by using a common name that is assigned by the process that created the file.

- To work with a memory-mapped file, you must create a view of the entire memory-mapped file or a part of it.
- You can also create multiple views to the same part of the memory-mapped file, thereby creating concurrent memory.
- For two views to remain concurrent, they have to be created from the same memory-mapped file.
- There are two types of views: stream access view and random access view.
- Use stream access views for sequential access to a file; this is recommended for non-persisted files and IPC.
- Random access views are preferred for working with persisted files.

multiple processes can have multiple and overlapping views to the same memory mapped file



Memory-Mapped Files

Persisted memory-mapped files

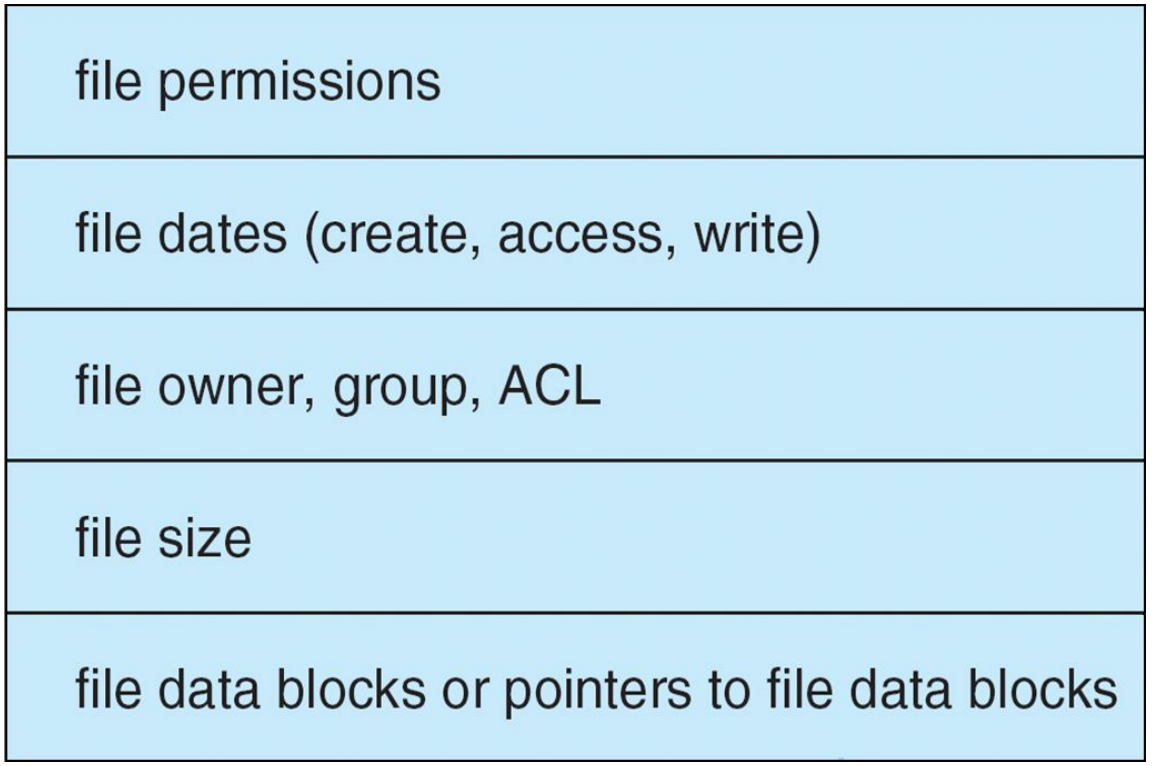
- Persisted files are memory-mapped files that are associated with a source file on a disk.
- When the last process has finished working with the file, the data is saved to the source file on the disk.
- These memory-mapped files are suitable for working with extremely large source files.

Memory-Mapped Files

Non-persisted memory-mapped files

- Non-persisted files are memory-mapped files that are not associated with a file on a disk.
- When the last process has finished working with the file, the data is lost and the file is reclaimed by garbage collection.
- These files are suitable for creating shared memory for inter-process communications (IPC).

File System Implementation



File Control Block

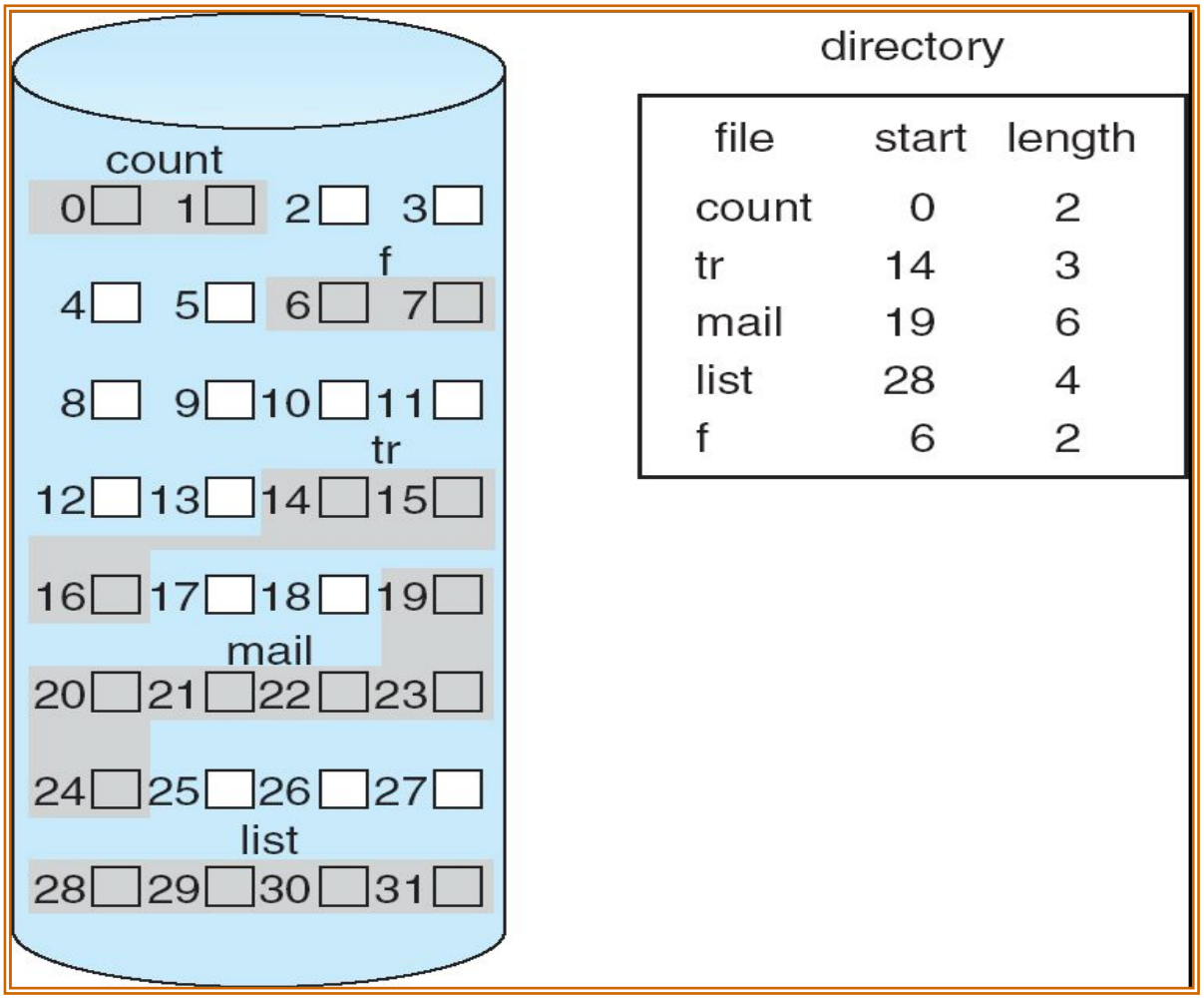
Allocation Methods

- How should file system allocate disk blocks to each file?
  1. Contiguous Allocation
  2. Linked Allocation
  3. Indexed Allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Both sequential and direct access is supported
- For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block.
- For direct access to block i of a file that starts at block b, we can immediately access block (b + i) .

Contiguous Allocation



directory

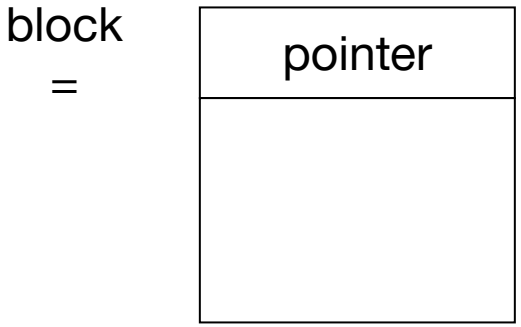
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous Allocation

- Contiguous allocation has some problems
  - Dynamic storage-allocation
    - How to satisfy a request of size *n* from a list of free blocks
  - External fragmentation
    - Free space is broken into chunks and the largest chunk is insufficient for a request
  - Determining how much space is needed for a file
    - When the file is created, the total amount of space it will need must be found and allocated. How does the creator know the size of the file to be created?
    - If Allocated too little and the file may not be extended.
    - If allocated too much and space is wasted.
  - File cannot grow

Linked Allocation

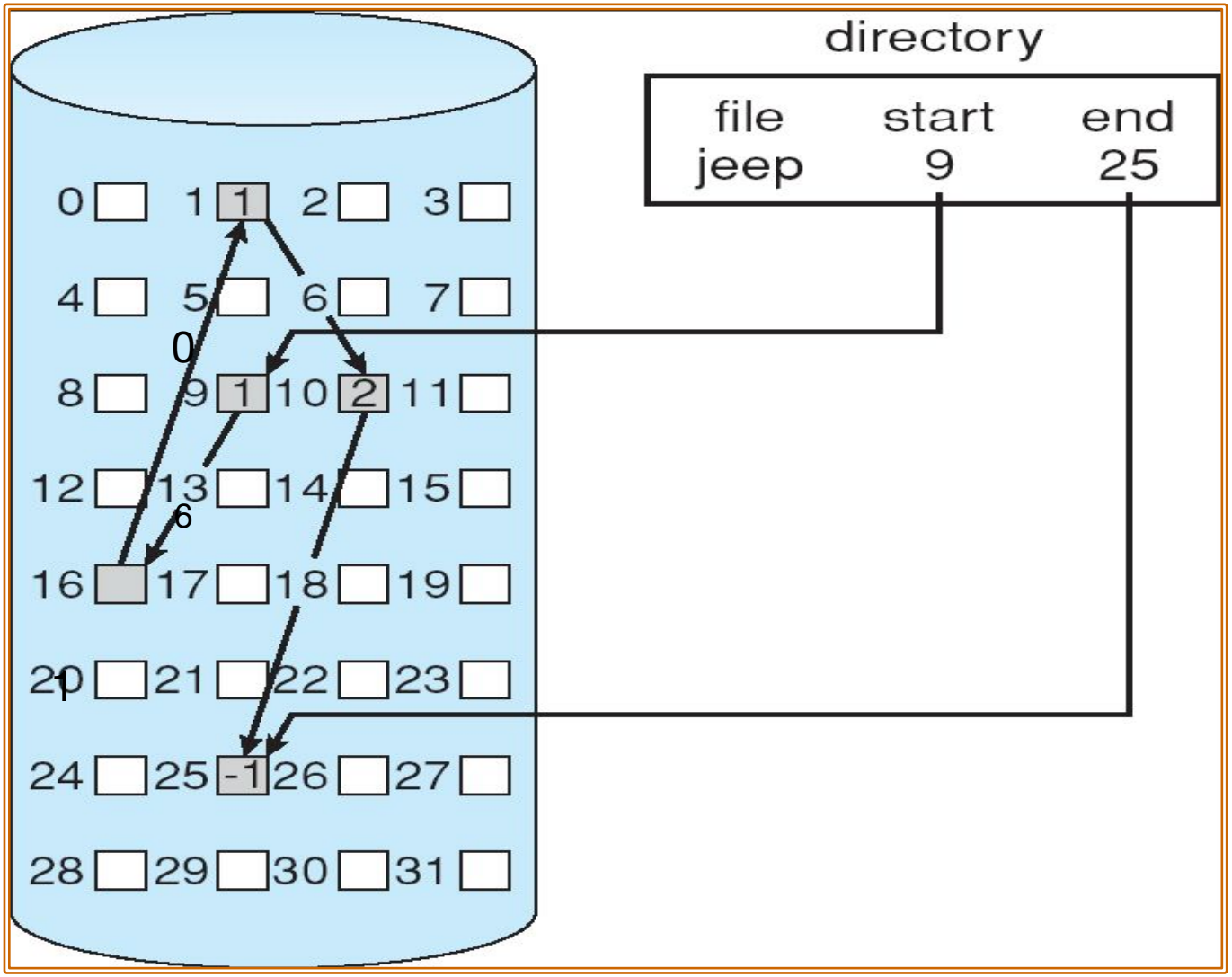
- Each file is allocated a linked list of disk blocks: blocks may be scattered everywhere on the disk.



Linked Allocation

- Simple – need only starting address
- Solves all the problems of contiguous allocation
- Size of file need not be declared at the time of creation
- Free-space management system – no wastage of space
- No external fragmentation
- Effective sequential access but no random access

Linked Allocation



Linked Allocation

To create a new file:

- Create a new entry in the directory.
- Each directory entry has a pointer to the first disk block of the file.
- This pointer is initialized to ***nil*** to signify an empty file.
- The size field is also set to 0.

To write the file:

- Search a free block .
- This new block is then written to, and is linked to the end of the file.

To read a file:

- Read blocks by following the pointers from block to block.

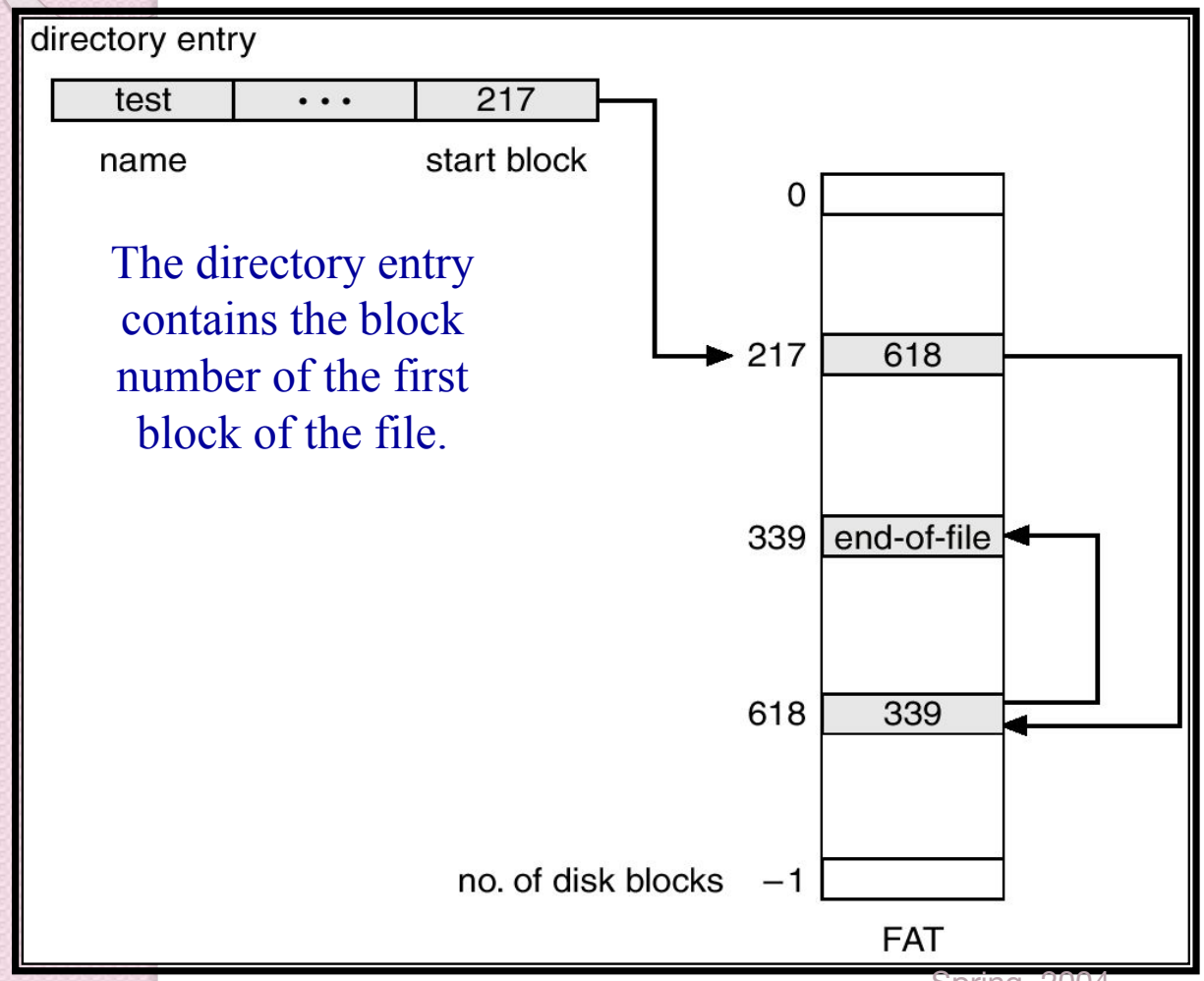
Linked Allocation

Disadvantages

- It can be used effectively only for sequential access
- Random access is expensive
  - To find the i^{th} block, we must start from the beginning of the file and follow the pointers until we get the i^{th} block
- If a pointer needs 4 bytes out of 512 byte block then, the user sees blocks of 508 bytes.
- Maximum memory is wasted for pointers.
- Pointer space: Maximum disk space is used for pointers

File-Allocation Table

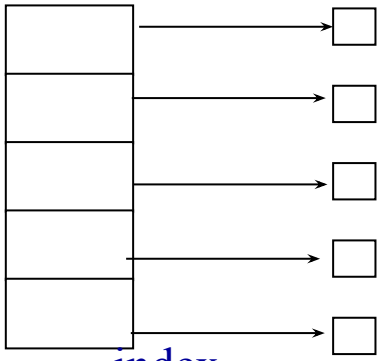
This is a variation of linked allocation



- The table has one entry for each block and is indexed by block number.
- Unused blocks are indicated by 0 table value.
- Allocating a new block involves finding the first 0 valued table entry and replacing the previous end-of-file value with the address of the new block

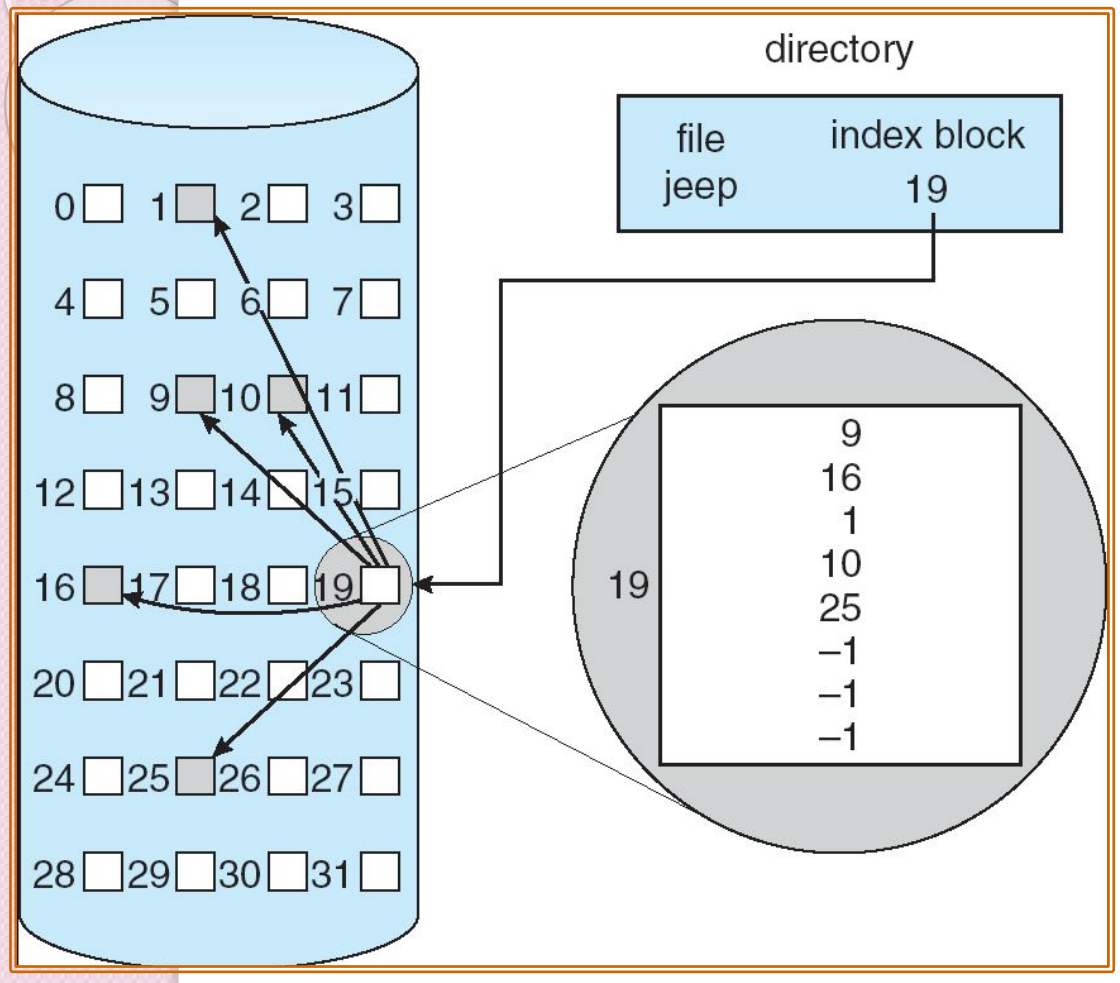
Indexed Allocation

- Each file has its own index block, which is an array of disk-block addresses
- Supports direct access by bringing all the pointers together into the index block
- To read ith block , read index block entry, gives direct access.
- Dynamic allocation without external fragmentation, but has overhead of index block.
- Logical view.



index table

Indexed Allocation



To create file:

All pointers in the index block are set to nil.

To write a block:

When the ith block is first written, a block is obtained from the free-space manager, and its address is put in the ith index-block entry.

Indexed Allocation

- The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.
- Consider a file of only two blocks. With linked allocation, we lose the space of only one pointer per block
- With indexed allocation, an entire index block must be allocated.
- The question: How large the index block should be?
- Every file must have an index block, so we want the index block to be as small as possible. If the index block is too small, however, it will not be able to hold enough pointers for a large file.

Indexed Allocation

Indexed Allocation schemes

1. Linked scheme
2. Multilevel index scheme
3. Combined scheme

Linked Scheme in Index Allocation

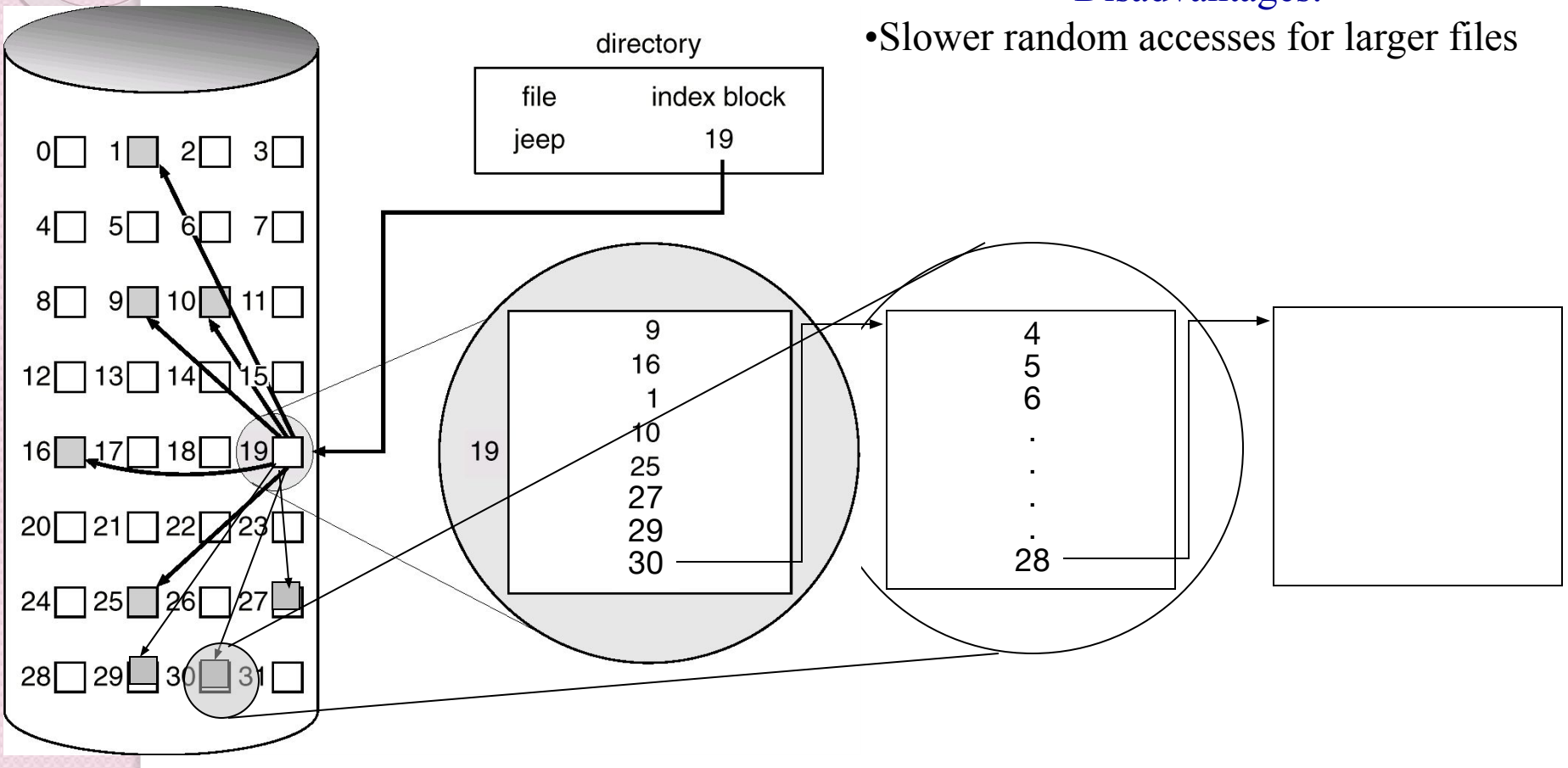
Index blocks are divided for larger files and linked to each other.

Advantage:

•Adjustable to any size of files

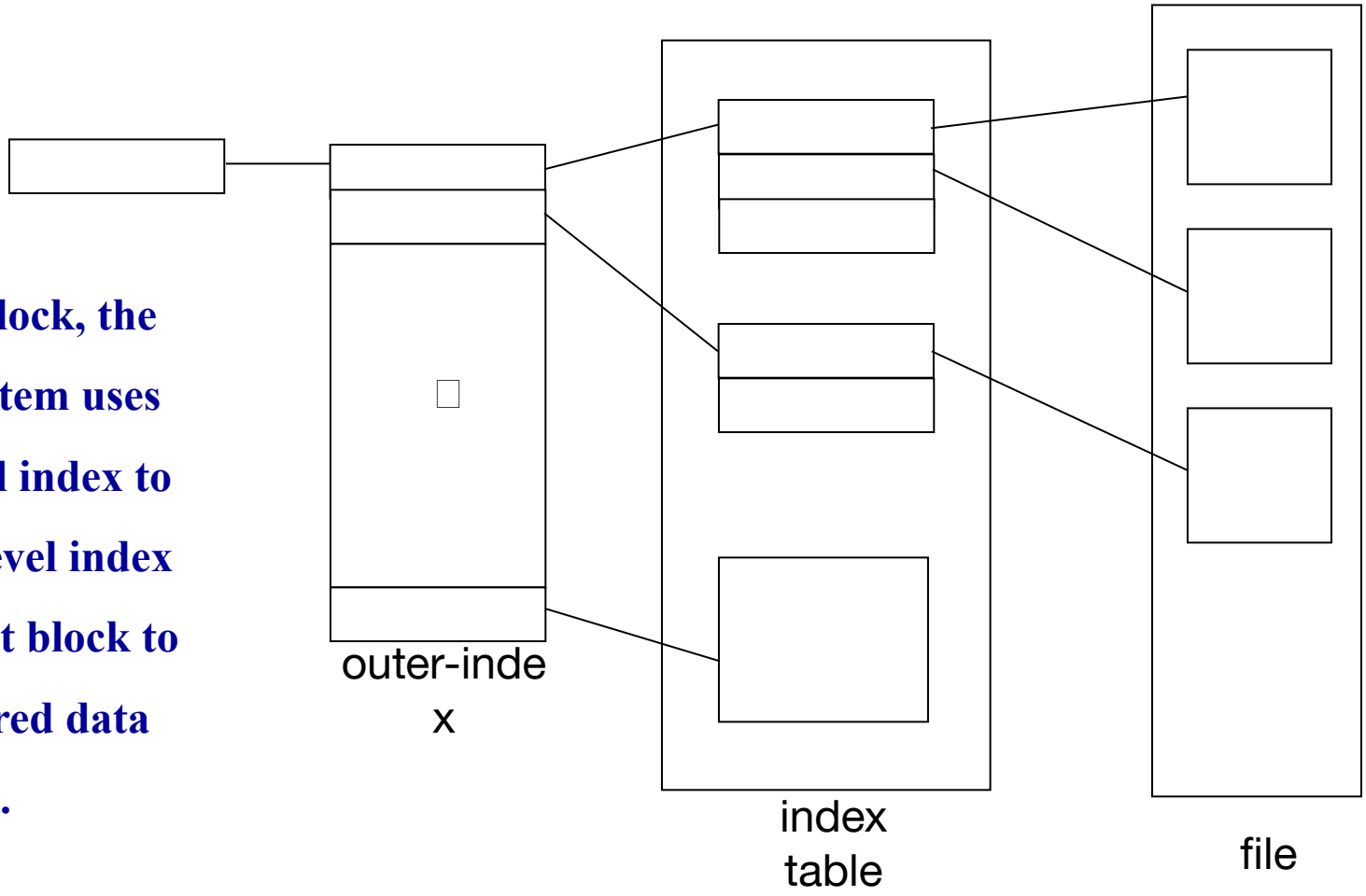
Disadvantages:

•Slower random accesses for larger files



Indexed Allocation – Multilevel indexing

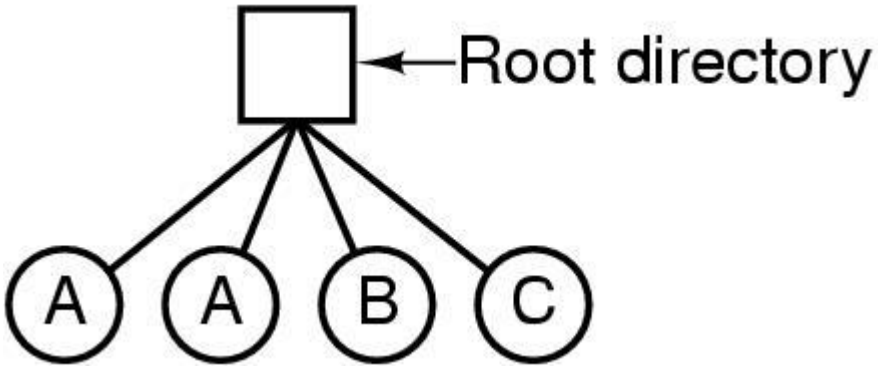
To access a block, the operating system uses the outer-level index to find a inner-level index block, and that block to find the desired data block.



Indexed Allocation – Two level indexing

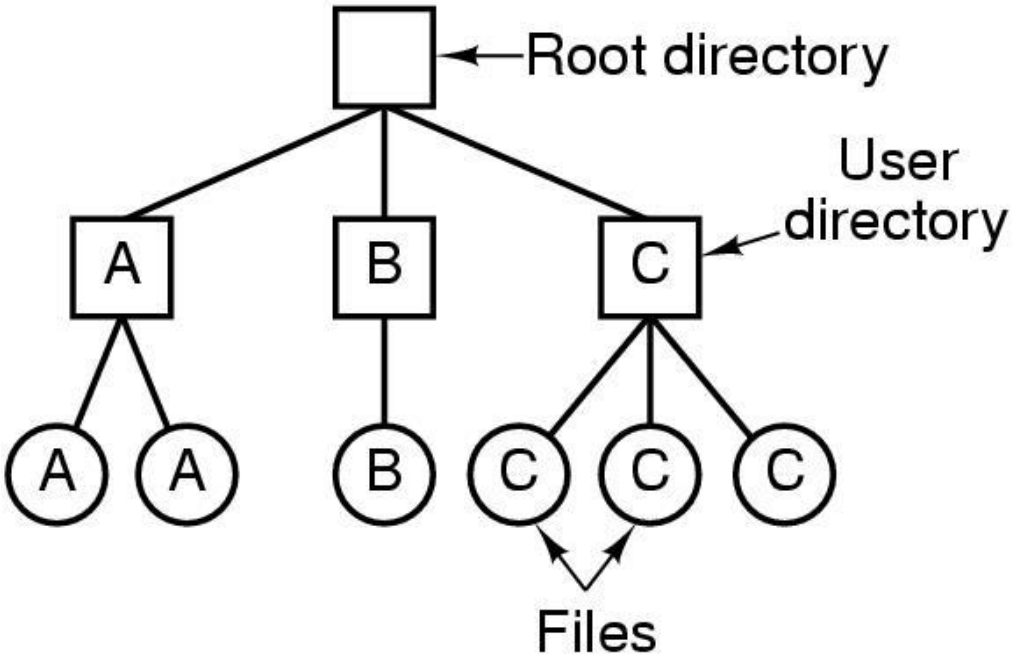
- With 4,096-byte blocks, we could store 1,024, 4-byte pointers in an index block. Two levels of indexes allow 1,048,576 data blocks, which allows a file of up to 4 GB.

Directories
Single-Level Directory Systems



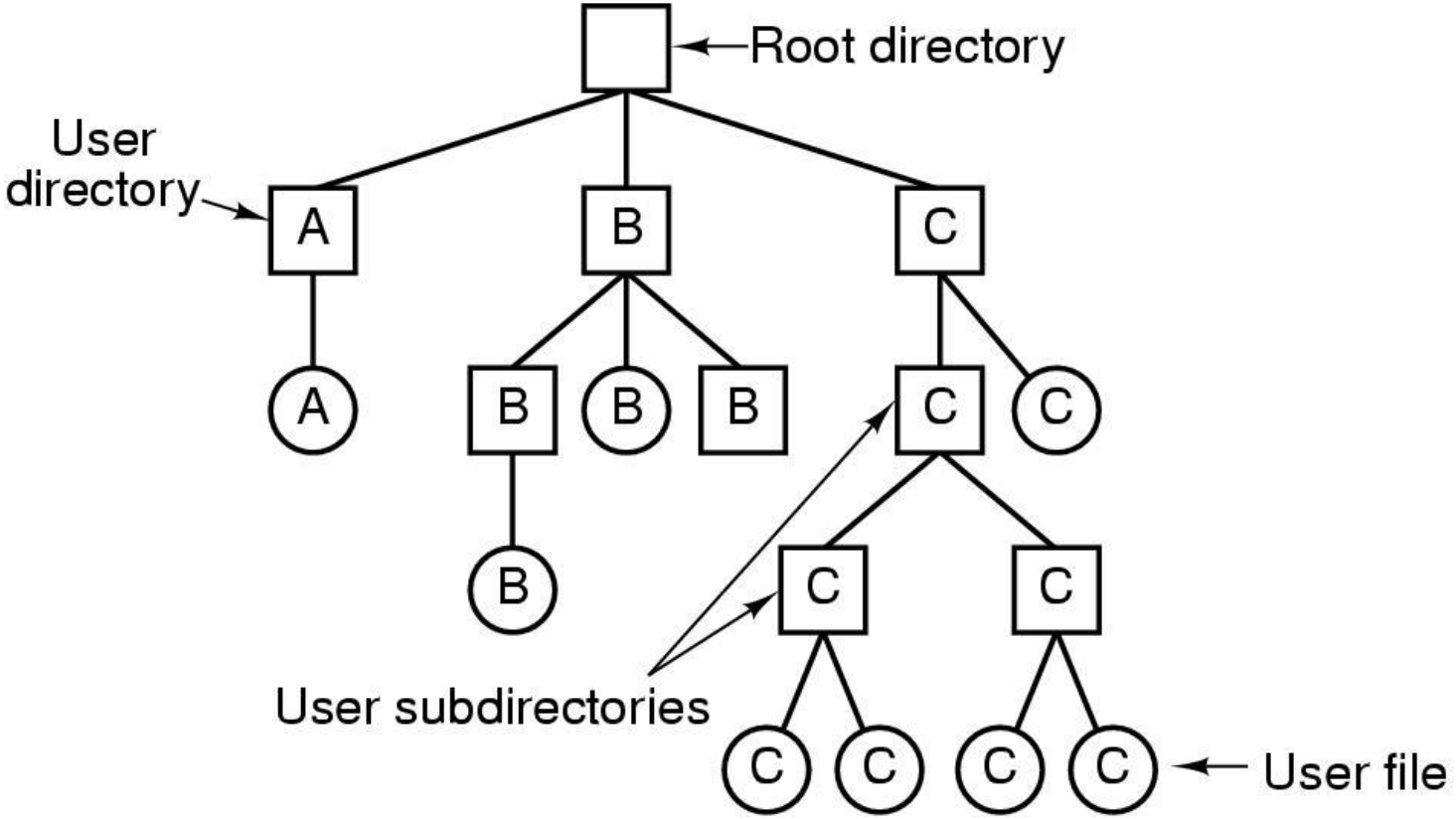
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

Two-level Directory Systems



Letters indicate *owners* of the directories and files

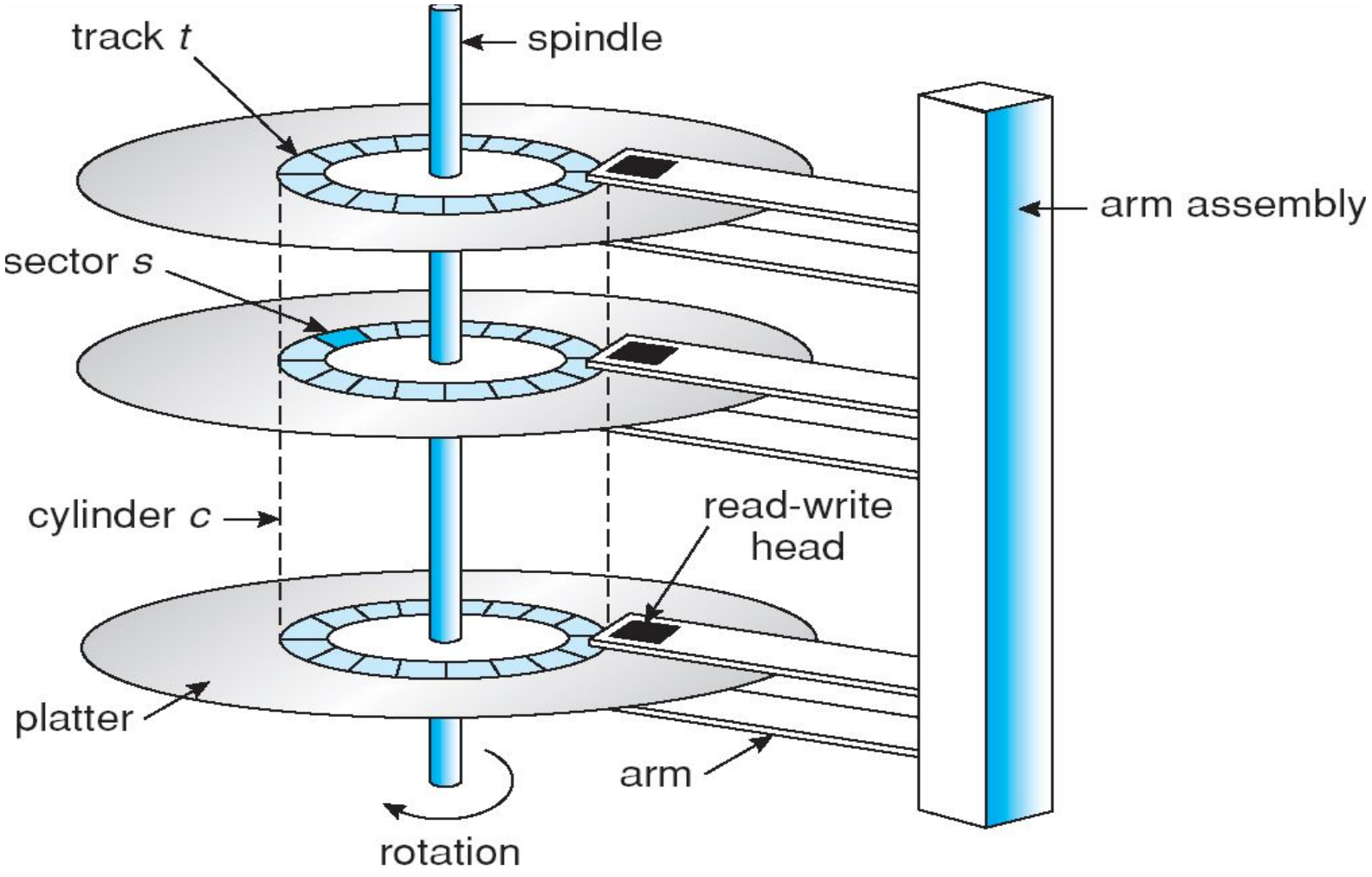
Hierarchical Directory Systems



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.

Moving-head Disk Mechanism



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 = the total number of bytes transferred/ total time between the first request for service and the completion of the last transfer.

Disk Scheduling (Cont.)

- 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

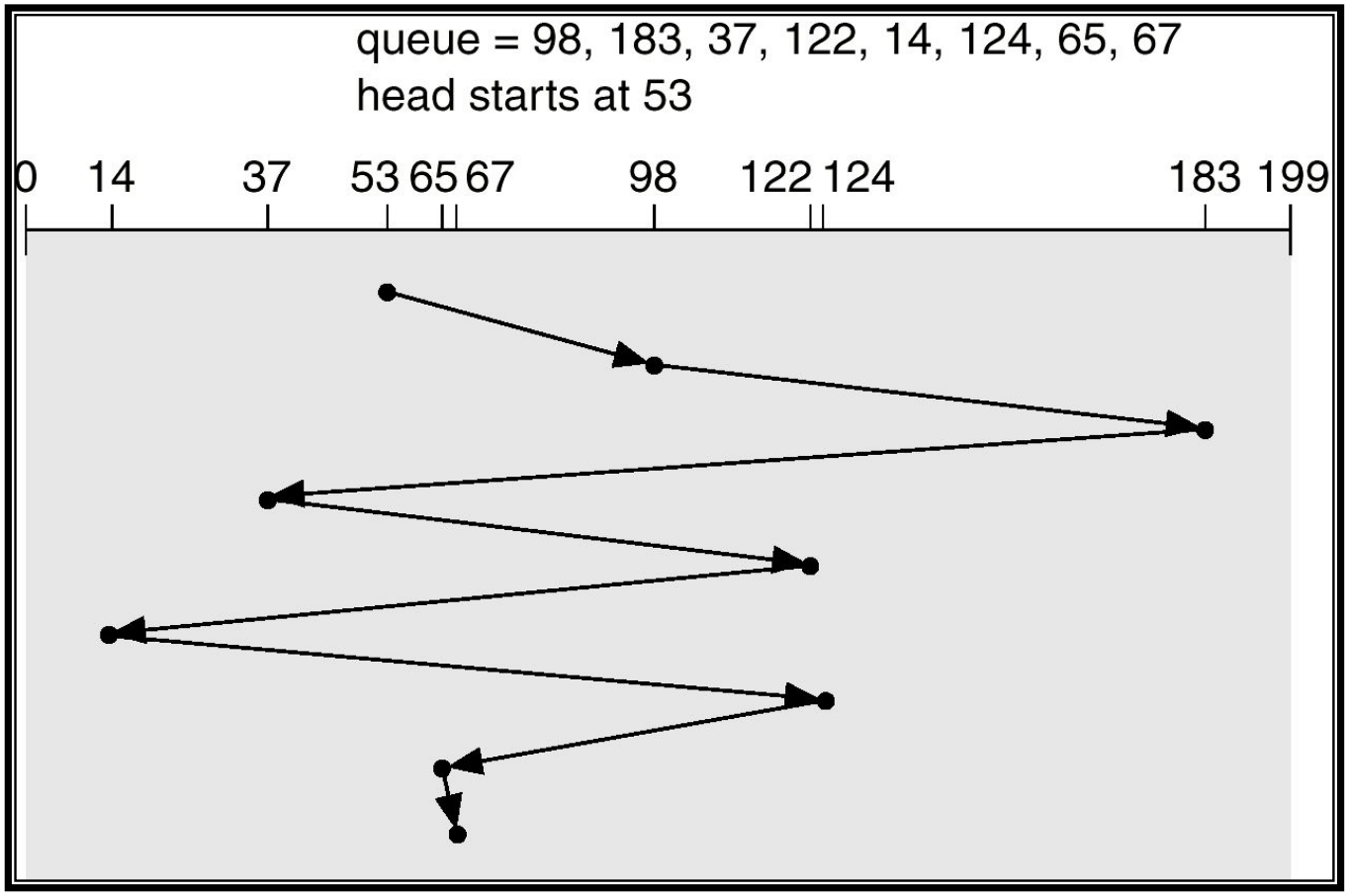


Illustration shows total head movement of 640 cylinders.

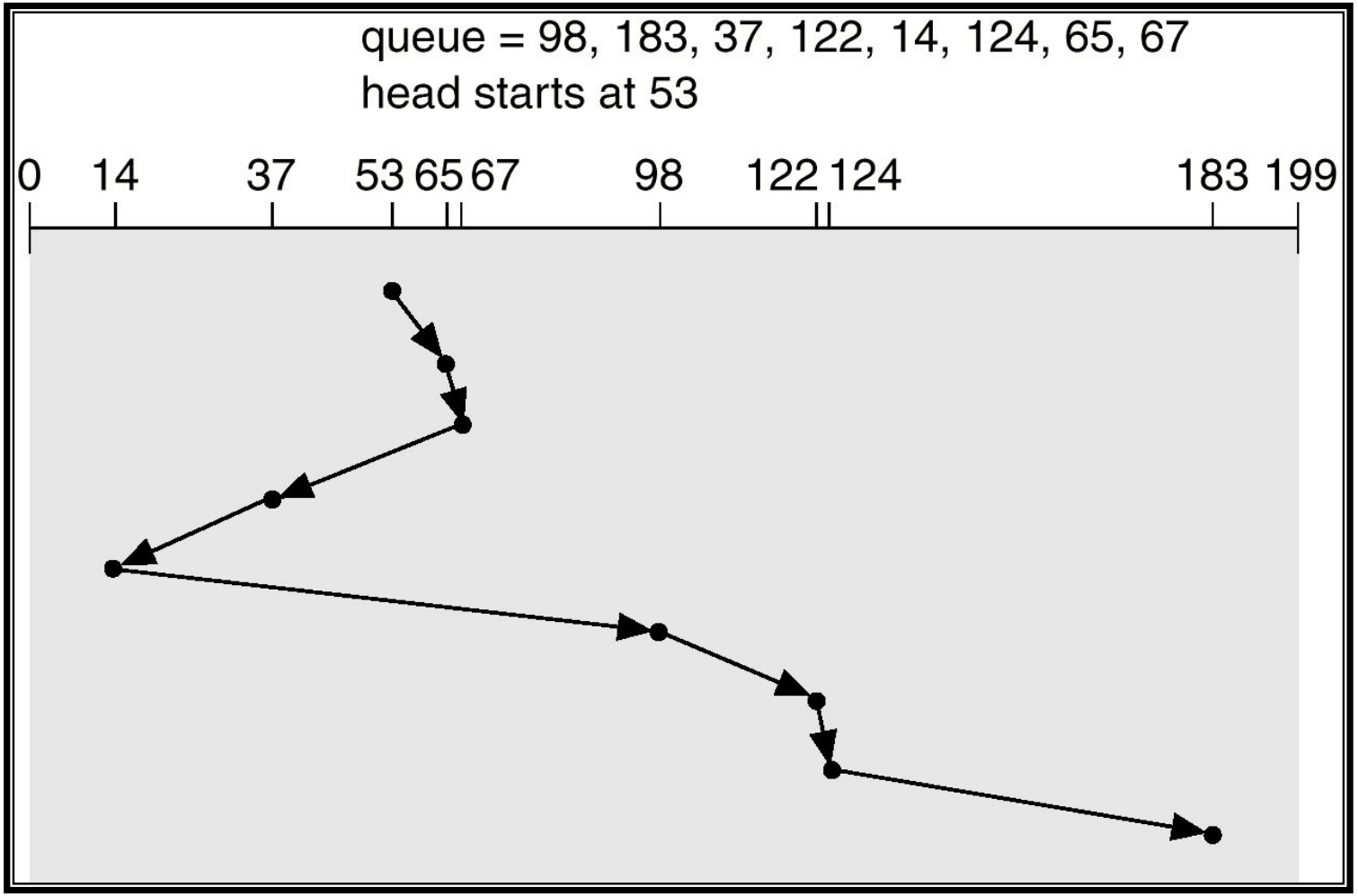
|53-98|+|98-183|+|183-37|+|37-122|+|122-14|+|14-124|+|124-65|+|65-67|

=45+85+146+85+108+110+59+2 = 640

SSTF

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

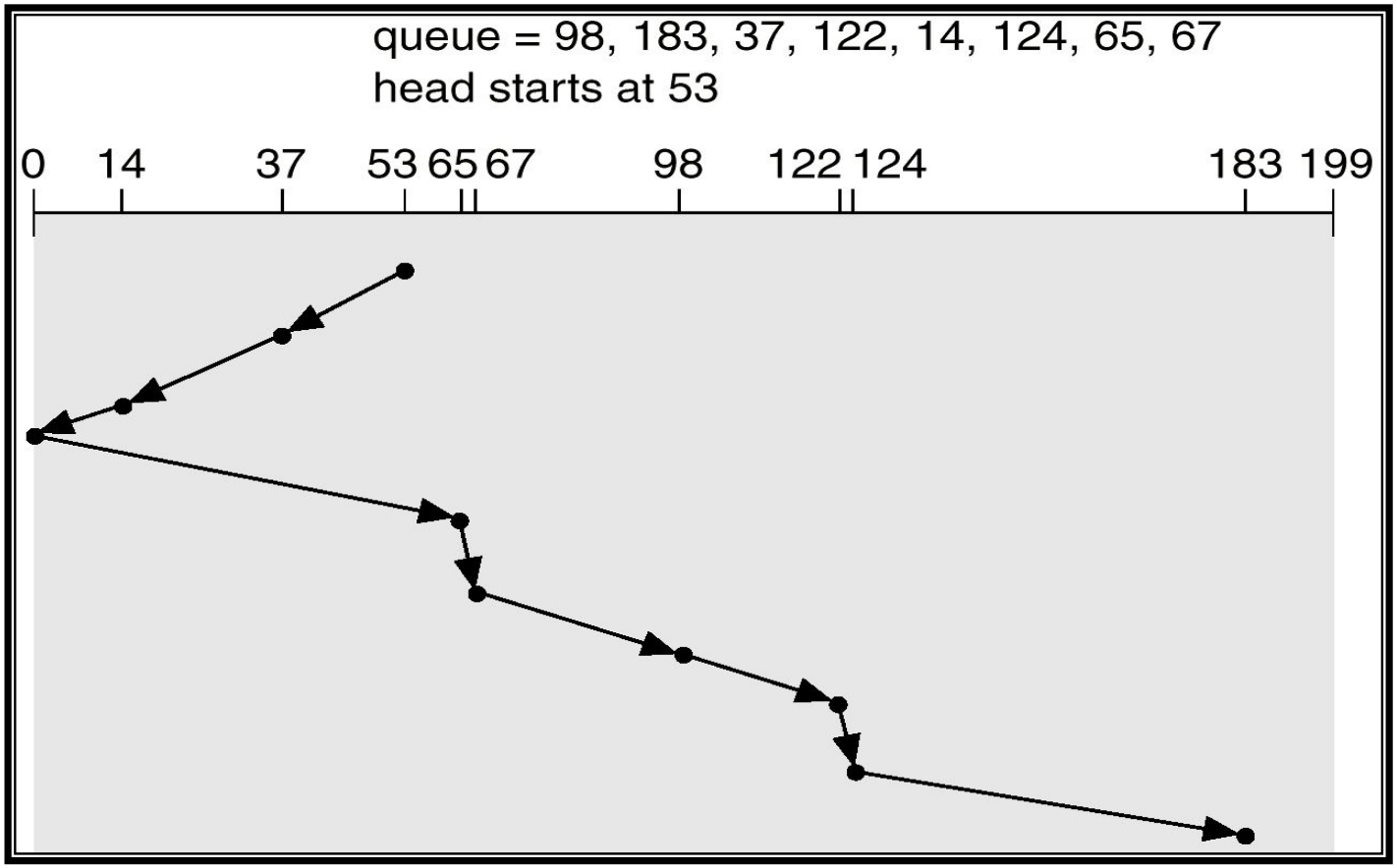
SSTF (Cont.)


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.

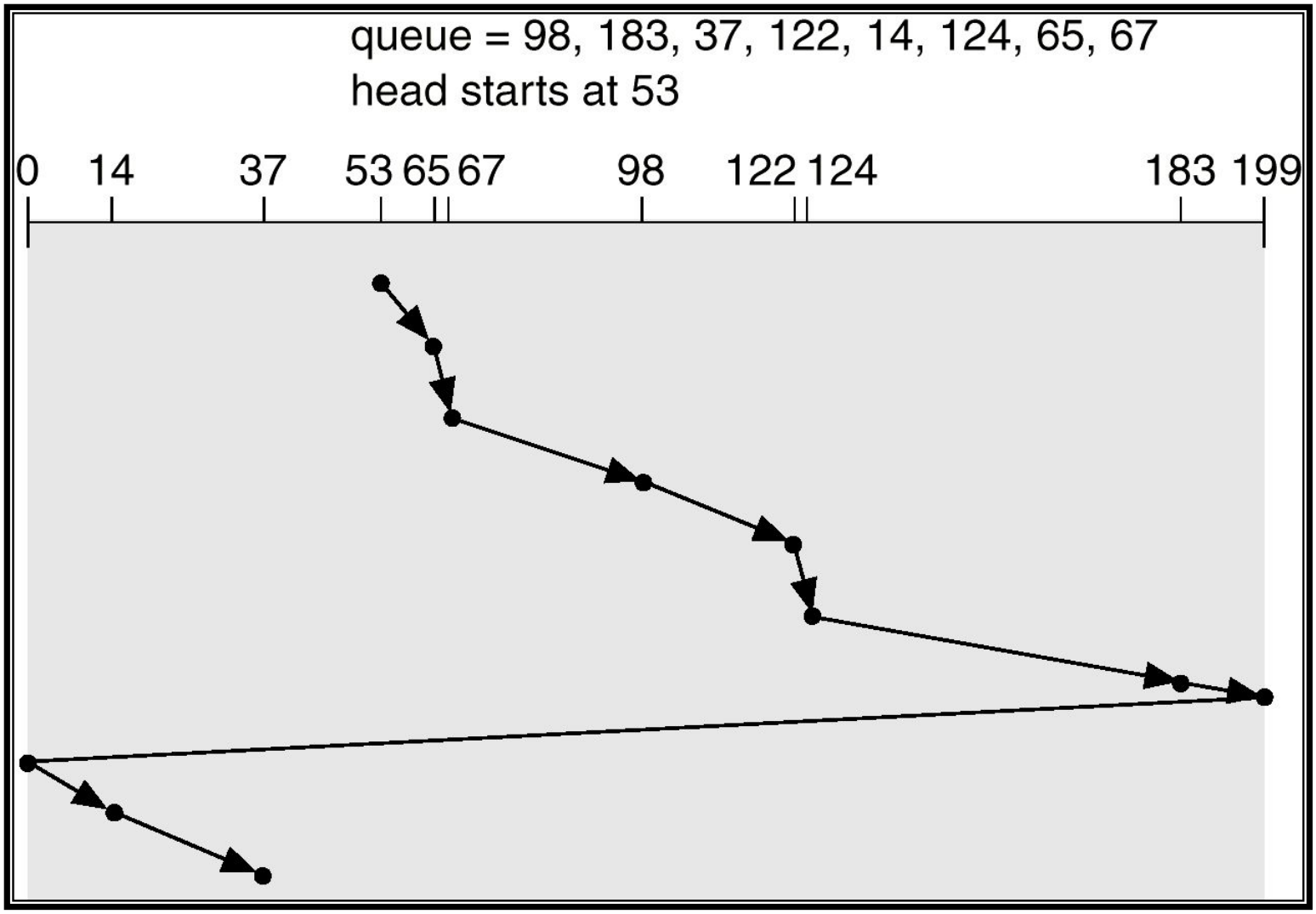
SCAN (Cont.)


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.

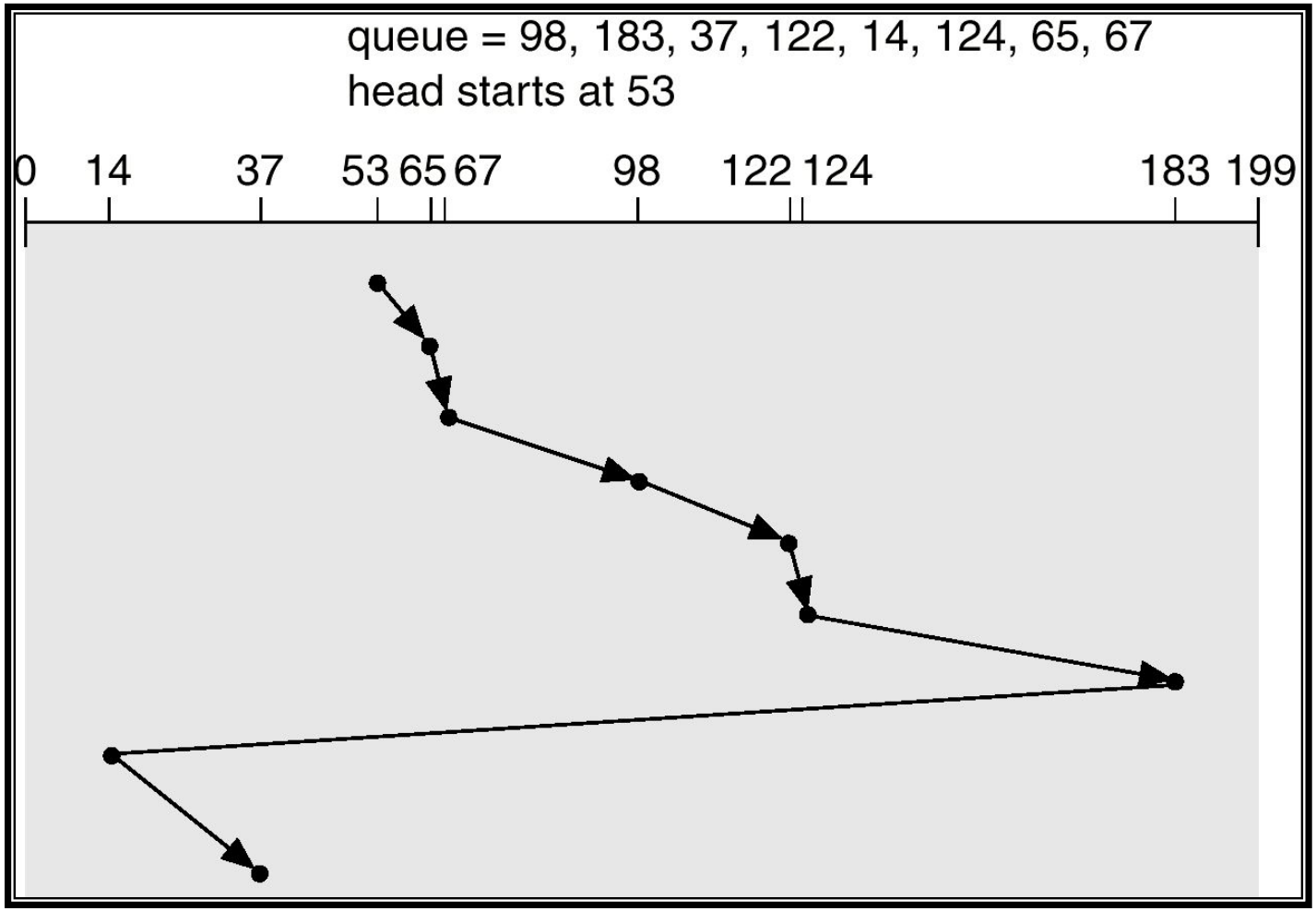
C-SCAN (Cont.)


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.

C-LOOK (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

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Comparison of Disk Scheduling Algorithms

RAID

- Redundant Array of Independent Disks
- Consists of seven levels, zero through six

•Design architectures share three characteristics:

- RAID is a set of physical disk drives viewed by the operating system as a single logical drive
- data are distributed across the physical drives of an array in a scheme known as striping
- redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure

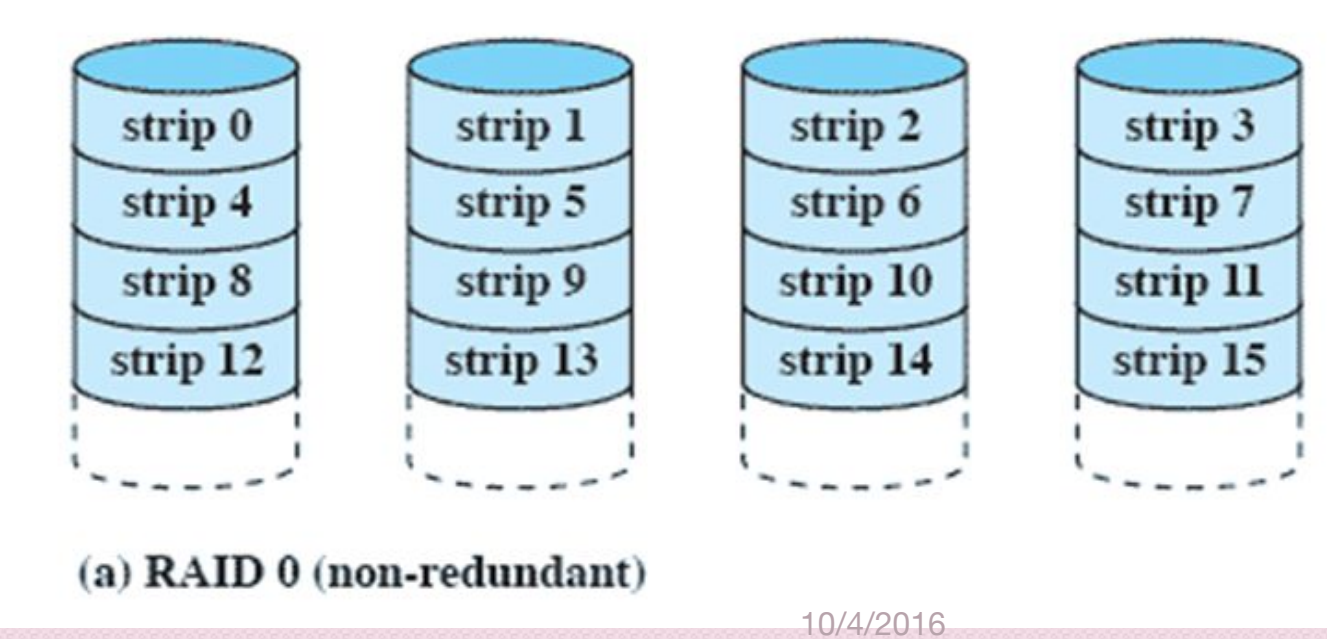
RAID LEVELS

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

N = number of data disks; m proportional to log N

RAID LEVEL 0

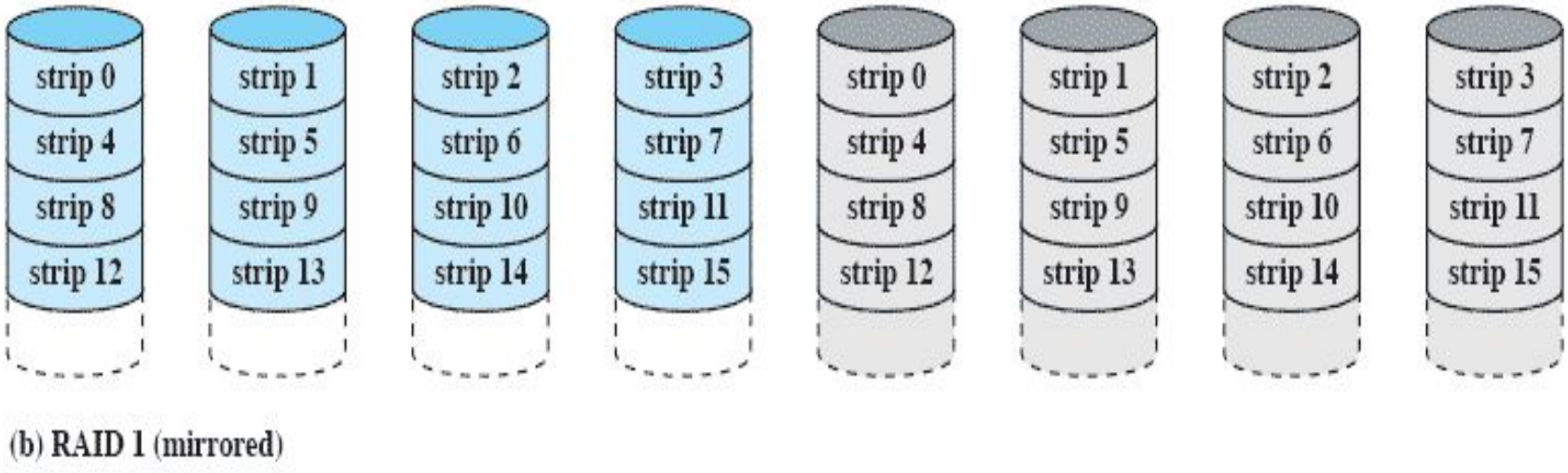
- Not a true RAID because it does not include redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
- Logical disk is divided into strips



(a) RAID 0 (non-redundant)

RAID LEVEL 1

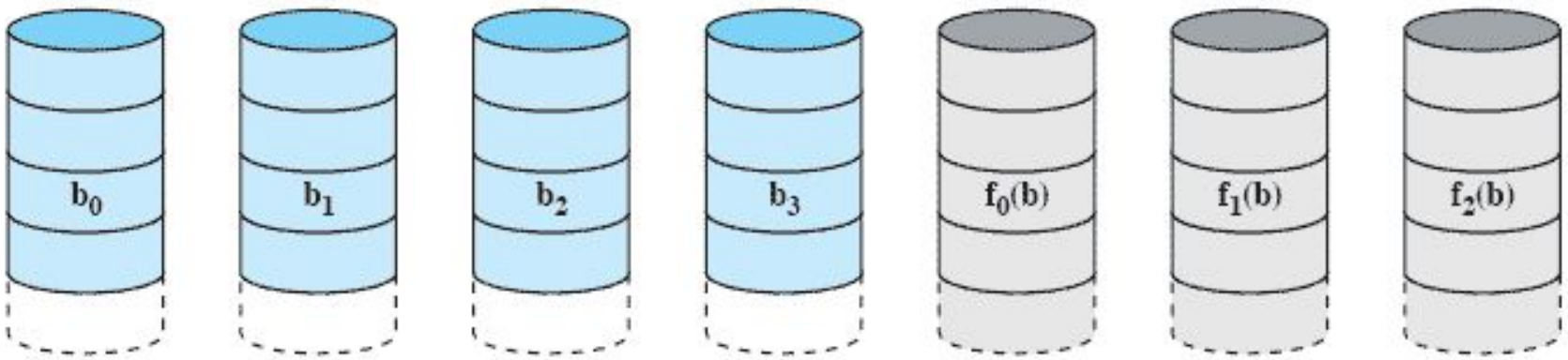
- Redundancy is achieved by the simple expedient of duplicating all the data
- When a drive fails the data may still be accessed from the second drive
- Principal disadvantage is the cost



(b) RAID 1 (mirrored)

RAID LEVEL 2

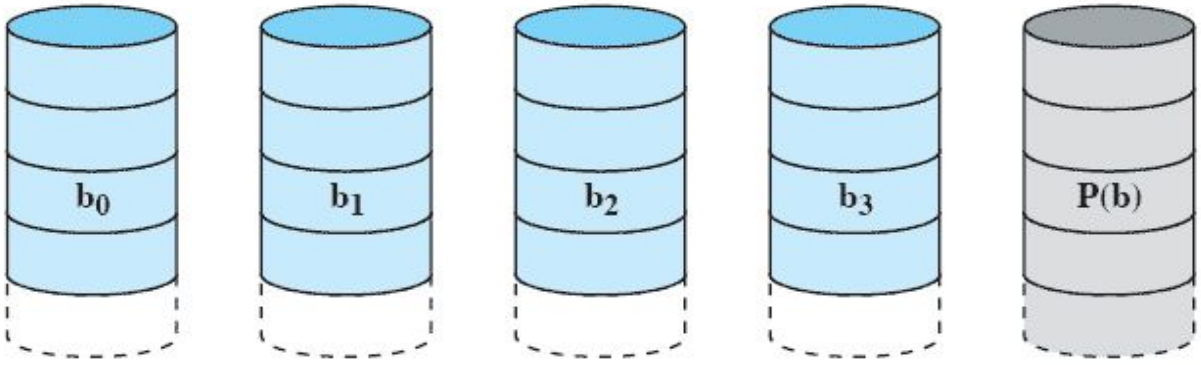
- Makes use of a parallel access technique
- Error correcting code is calculated. Typically a Hamming code is used
- No of (redundant) disks (n)=$\log_2(N)$



(c) RAID 2 (redundancy through Hamming code)

RAID LEVEL 3

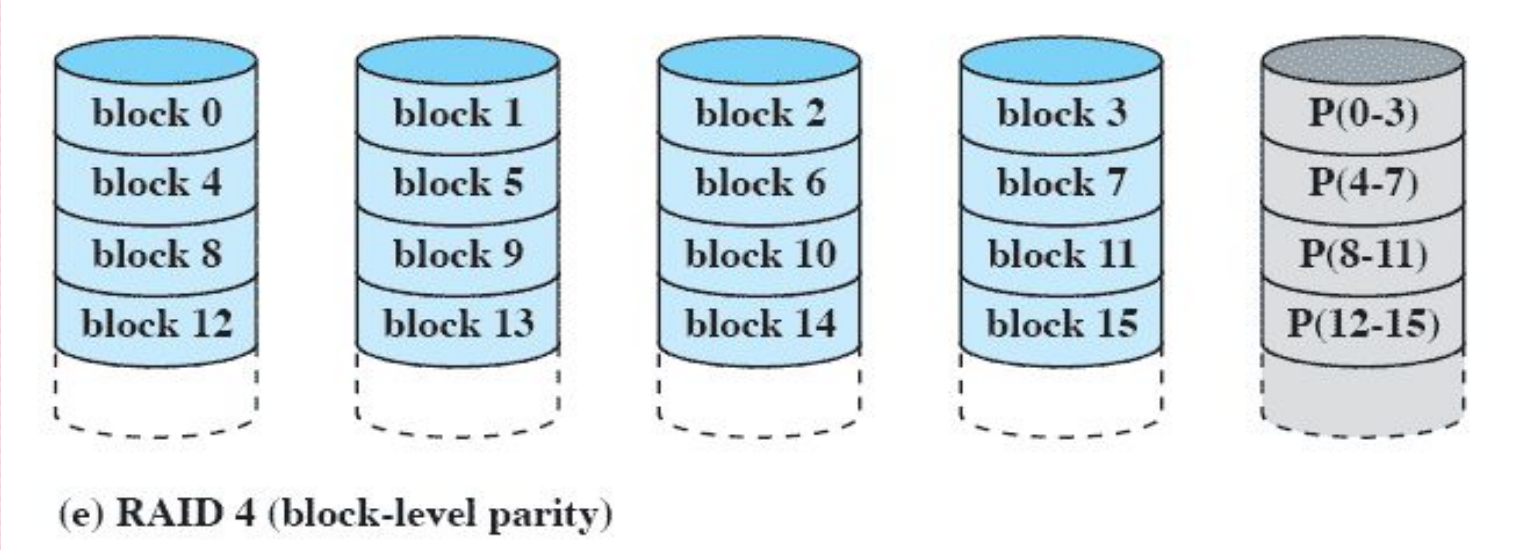
- Single redundant disk, no matter how large the disk array.
- Employs parallel access, Parity bit computed.
- X_0 through X_3 disks contain data and X_4 is the parity disk. The parity for the i^{th} bit *is* calculated as follows:
- $X_4(i) = X_3(i) \otimes X_2(i) \otimes X_1(i) \otimes X_0(i)$, where is exclusive-OR function.
- Suppose that drive X_1 has failed. If we add $X_4(i) \otimes X_1(i)$ to both sides of the preceding equation, we get $X_1(i) = X_4(i) \otimes X_3(i) \otimes X_2(i) \otimes X_0(i)$



(d) RAID 3 (bit-interleaved parity)

RAID LEVEL 4

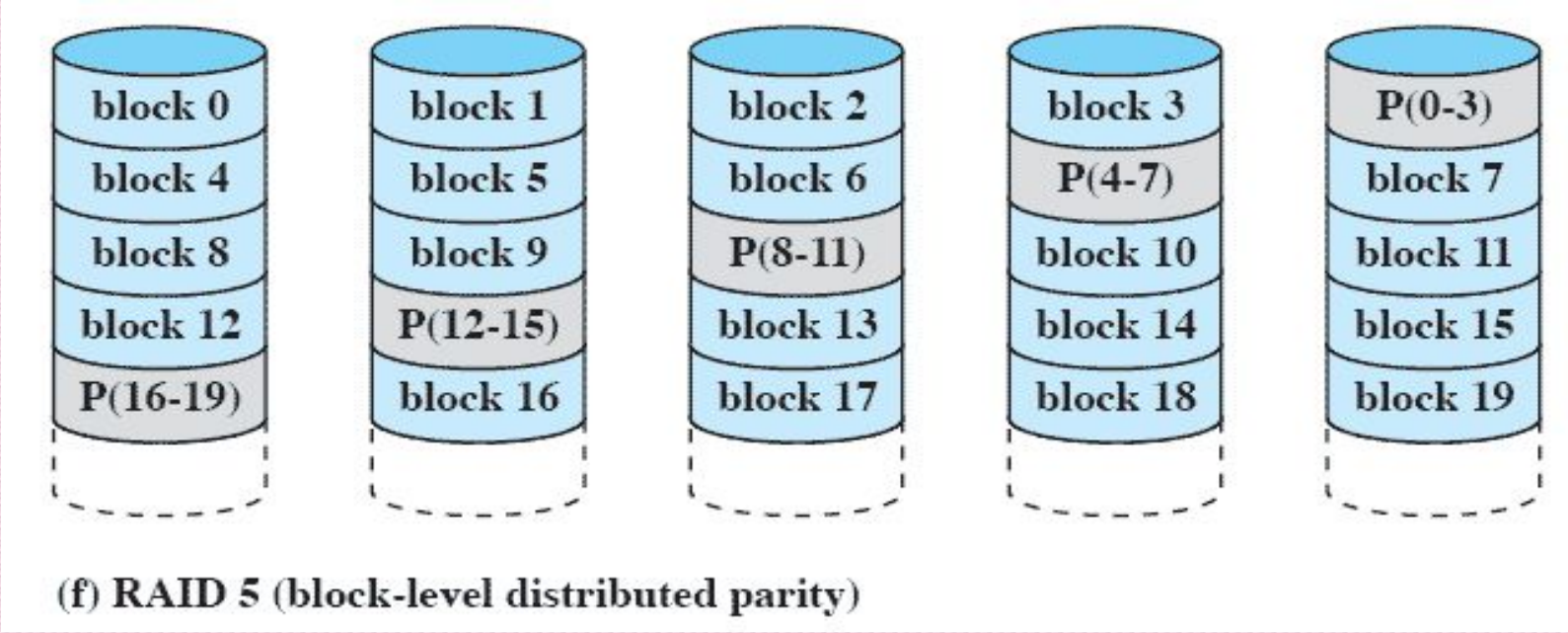
- Makes use of an independent access technique
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk



(e) RAID 4 (block-level parity)

RAID LEVEL 5

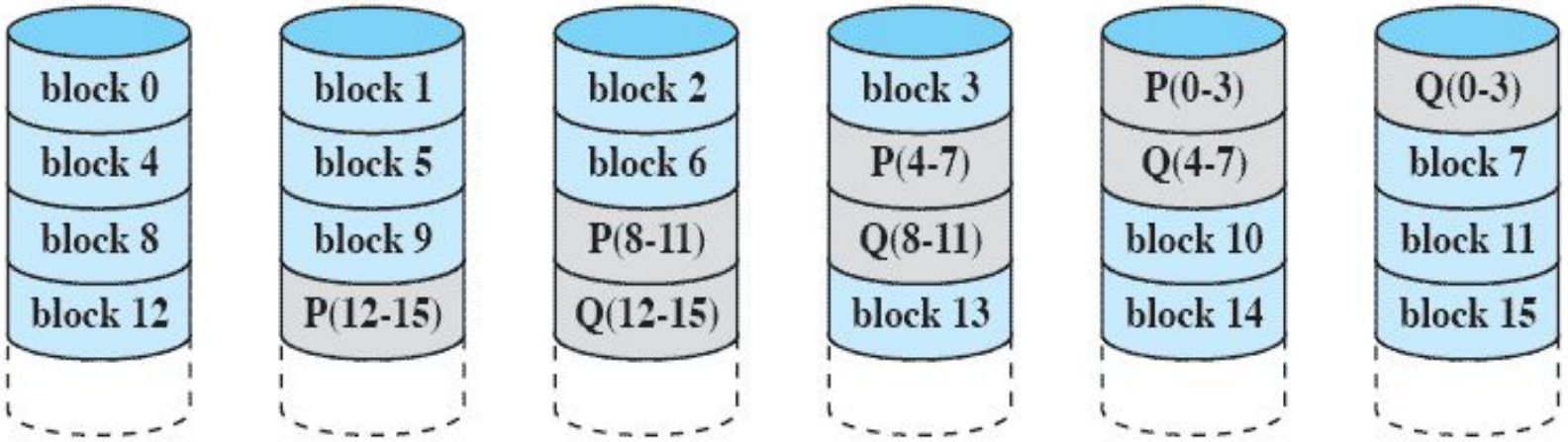
- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss
- Independent access
- no of disks= N+1



(f) RAID 5 (block-level distributed parity)

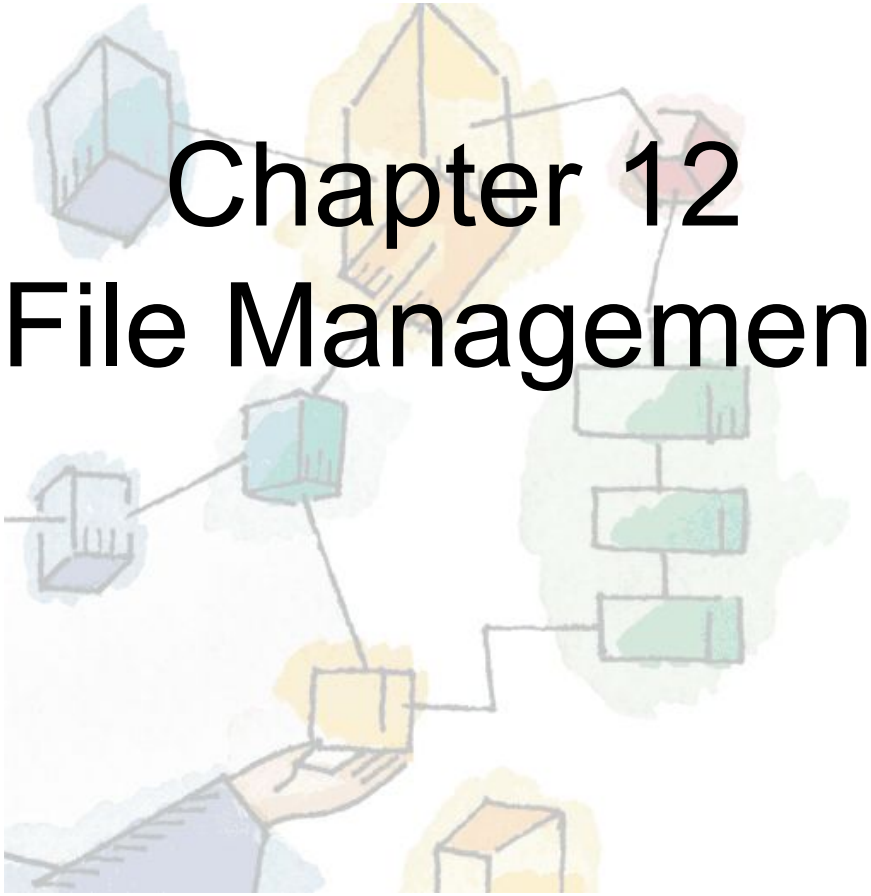
RAID LEVEL 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks
- Independent Access
- No of disks = N+2



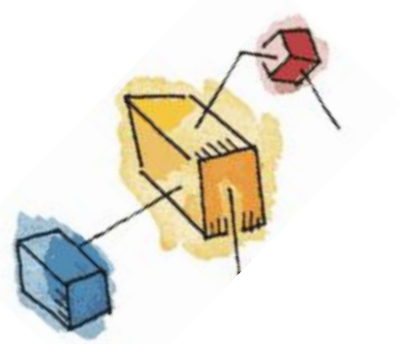
(g) RAID 6 (dual redundancy)

Operating Systems:
Internals and Design Principles, 6/E
William Stallings



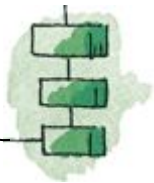
Chapter 12
File Management

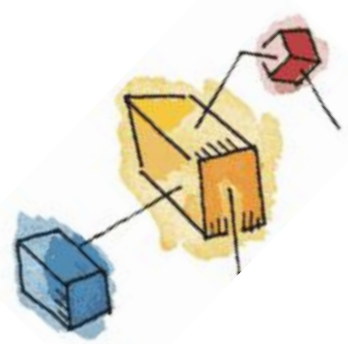
Patricia Roy
Manatee Community College, Venice,
FL



File Management

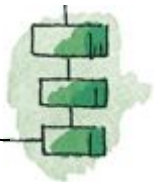
- File management system consists of system utility programs that run as privileged applications
- Concerned with secondary storage

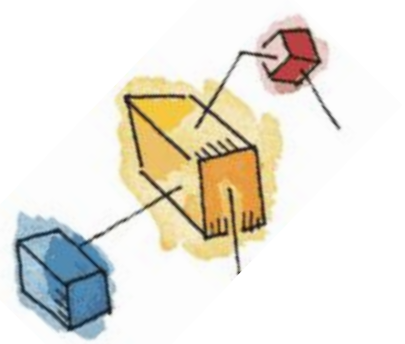




File System Properties

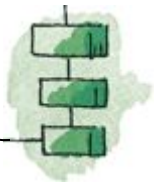
- Long-term existence
- Sharable between processes
- Structure

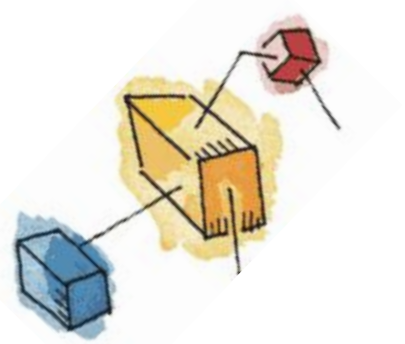




File Operations

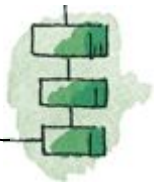
- Create
- Delete
- Open
- Close
- Read
- Write

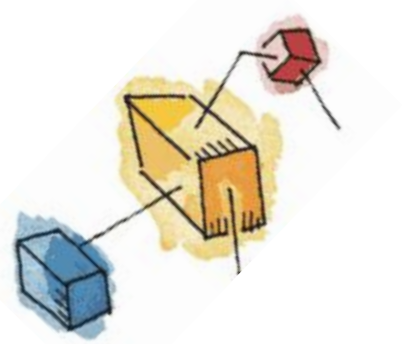




File Terms

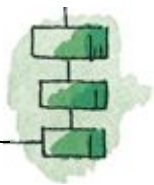
- Field
  - Basic element of data
  - Contains a single value
  - Characterized by its length and data type
- Record
  - Collection of related fields
  - Treated as a unit

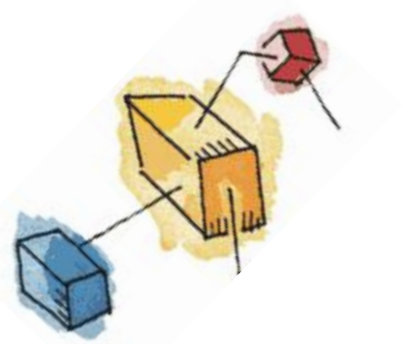




File Terms

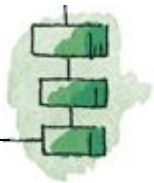
- File
  - Collection of similar records
  - Treated as a single entity
  - Have file names
  - May restrict access
- Database
  - Collection of related data
  - Relationships exist among elements

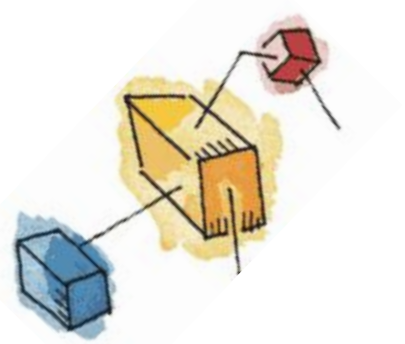




Typical Operations

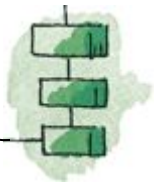
- Retrieve_All
- Retrieve_One
- Retrieve_Next
- Retrieve_Previous





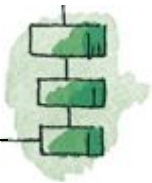
Typical Operations

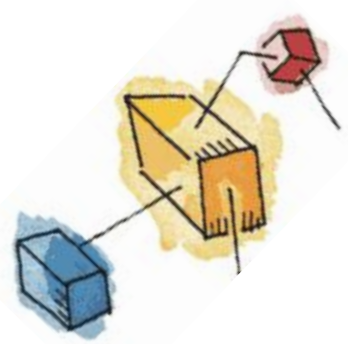
- Insert_One
- Delete_One
- Update_One
- Retrieve_Few



File Management Systems

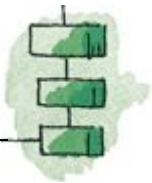
- The way a user or application may access files
- Programmer does not need to develop file management software

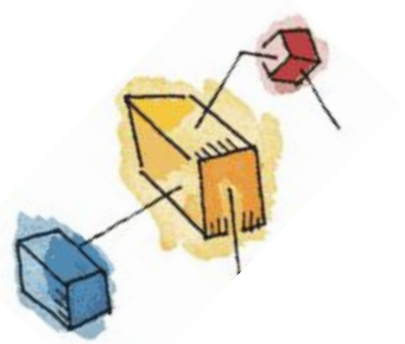




Objectives for a File Management System

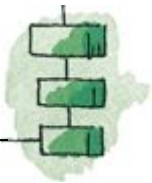
- Meet the data management needs and requirements of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types

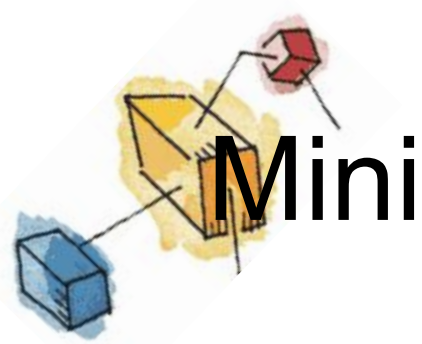




Objectives for a File Management System

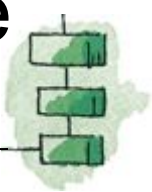
- Minimize or eliminate the potential for lost or destroyed data
- Provide a standardized set of I/O interface routines
- Provide I/O support for multiple users

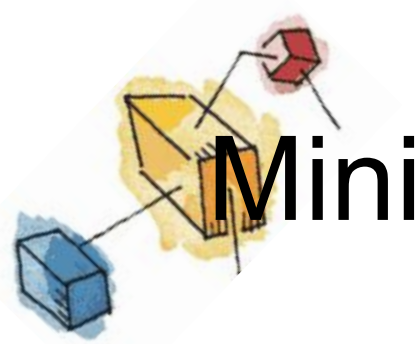




Minimal Set of Requirements

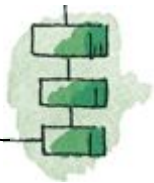
- Each user should be able to create, delete, read, write and modify files
- Each user may have controlled access to other users' files
- Each user may control what type of accesses are allowed to the users' files
- Each user should be able to restructure the user's files in a form appropriate to the problem





Minimal Set of Requirements

- Each user should be able to move data between files
- Each user should be able to back up and recover the user's files in case of damage
- Each user should be able to access the user's files by using symbolic names



Files System Software Architecture

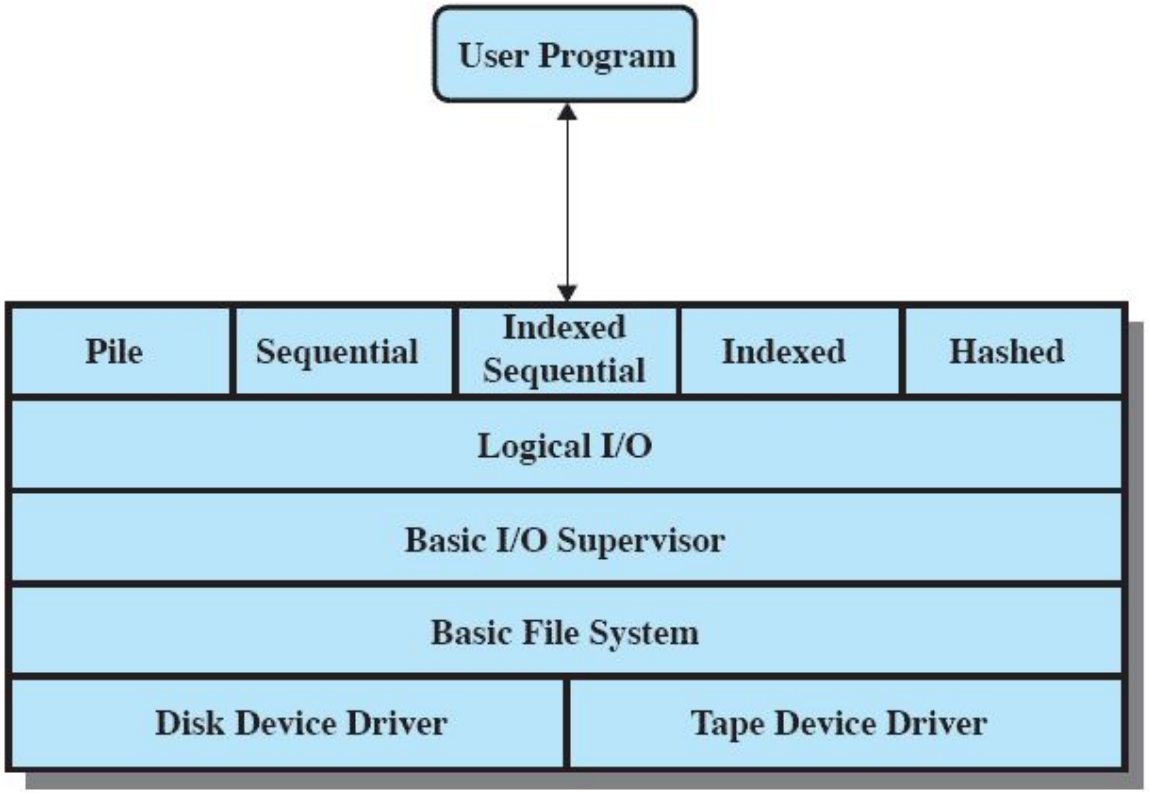
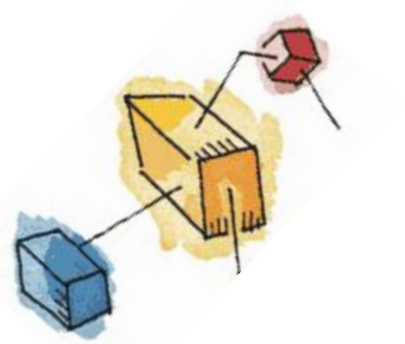
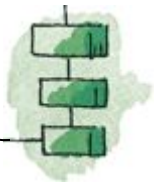


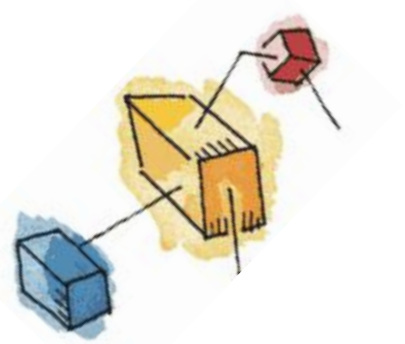
Figure 12.1 File System Software Architecture



Device Drivers

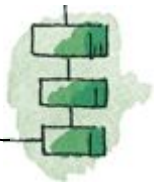
- Lowest level
- Communicates directly with peripheral devices
- Responsible for starting I/O operations on a device
- Processes the completion of an I/O request

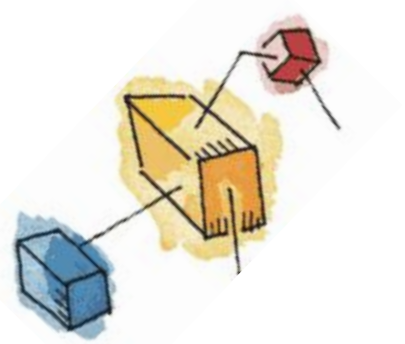




Basic File System

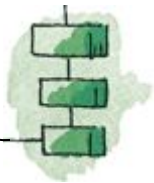
- Physical I/O
- Deals with exchanging blocks of data
- Concerned with the placement of blocks
- Concerned with buffering blocks in main memory

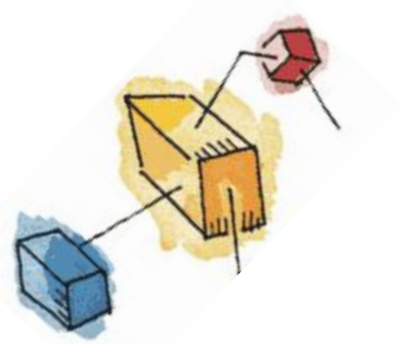




Logical I/O

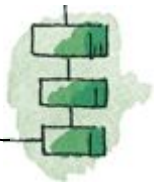
- Enables users and applications to access records
- Provides general-purpose record I/O capability
- Maintains basic data about file





Access Method

- Reflect different file structures
- Different ways to access and process data



Elements of File Management

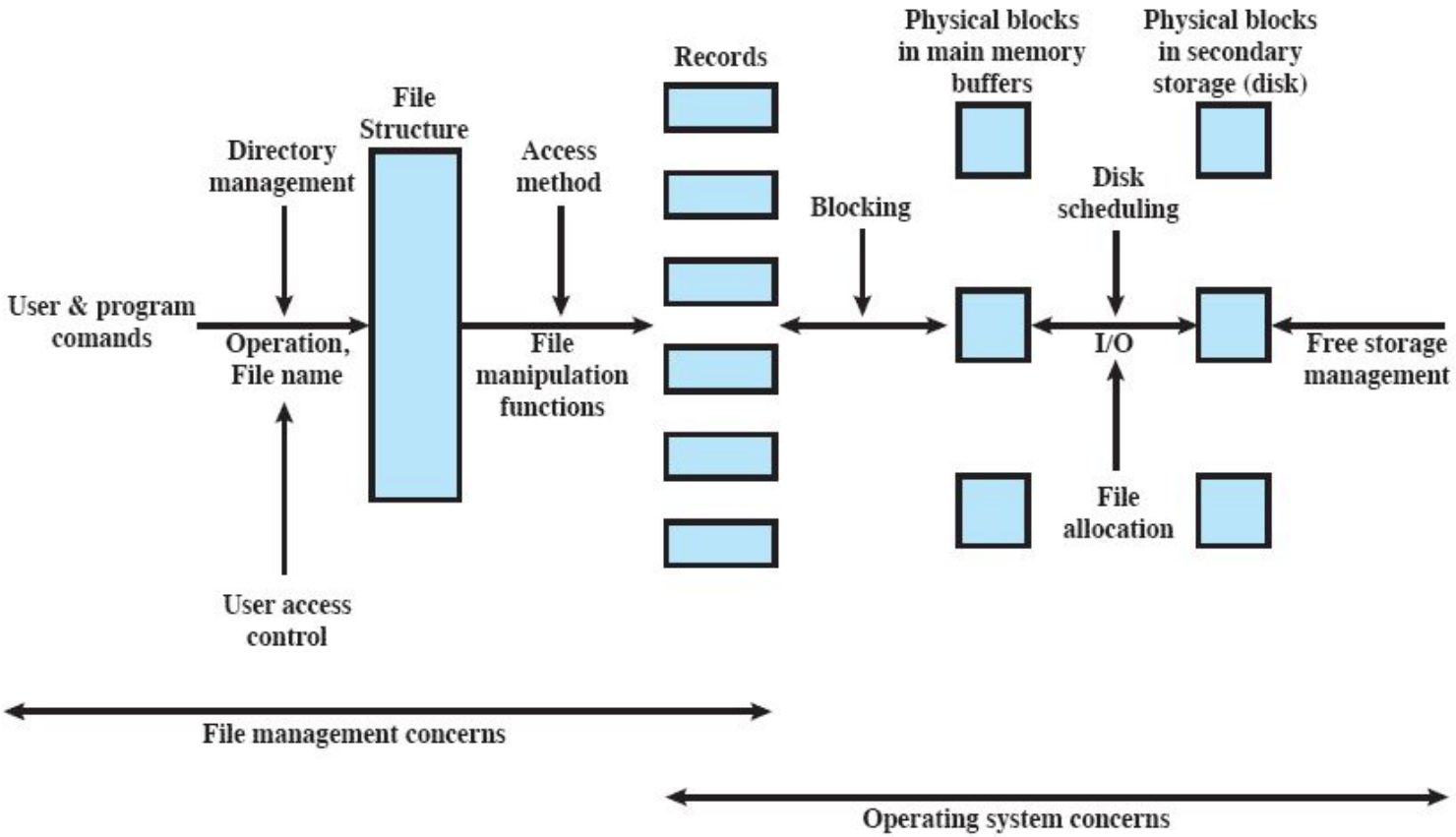
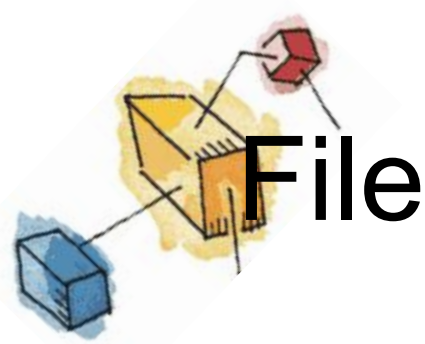
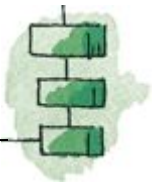


Figure 12.2 Elements of File Management



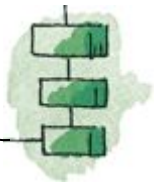
File Management Functions

- Identify and locate a selected file
- Use a directory to describe the location of all files plus their attributes
- On a shared system, describe user access control



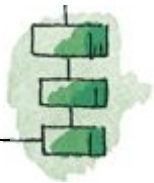
Criteria for File Organization

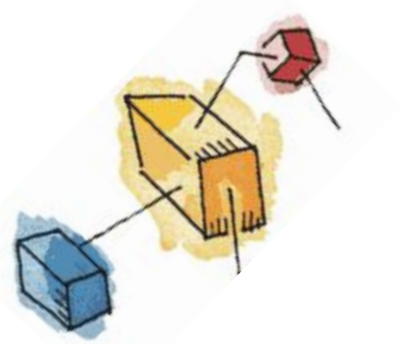
- Short access time
  - Needed when accessing a single record
- Ease of update
  - File on CD-ROM will not be updated, so this is not a concern



Criteria for File Organization

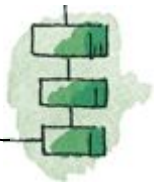
- Economy of storage
  - Should be minimum redundancy in the data
  - Redundancy can be used to speed access such as an index
- Simple maintenance
- Reliability



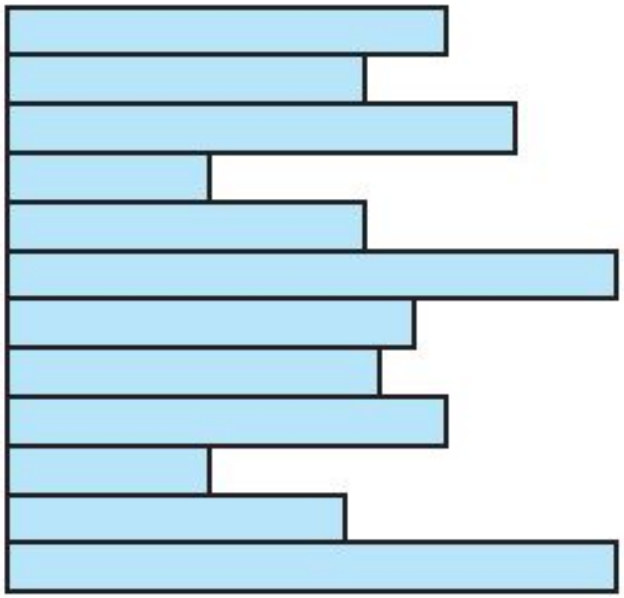


File Organization

- The Pile
  - Data are collected in the order they arrive
  - Purpose is to accumulate a mass of data and save it
  - Records may have different fields
  - No structure
  - Record access is by exhaustive search

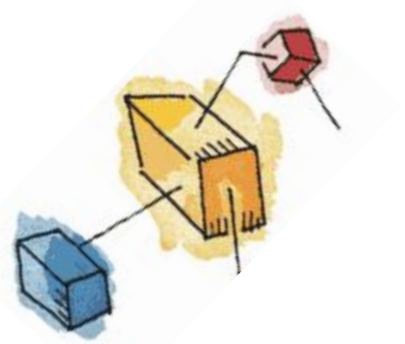


The Pile



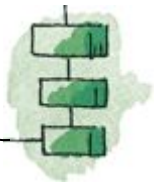
Variable-length records
Variable set of fields
Chronological order

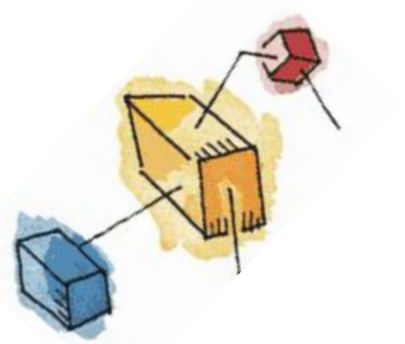
(a) Pile File



File Organization

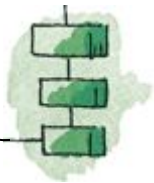
- The Sequential File
  - Fixed format used for records
  - Records are the same length
  - All fields the same (order and length)
  - Field names and lengths are attributes of the file



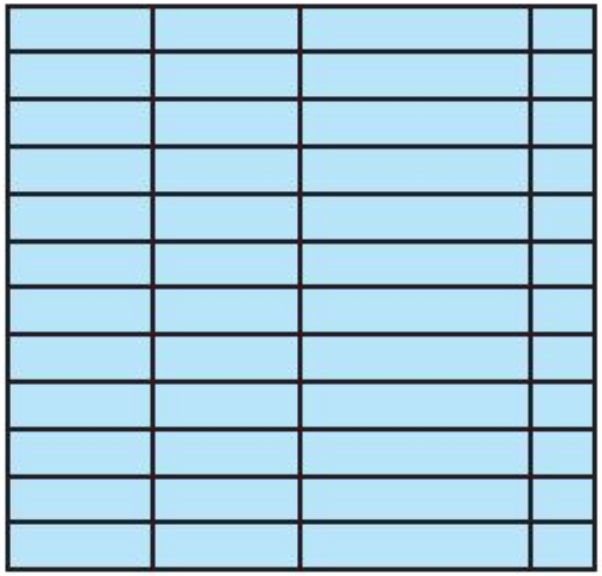


File Organization

- The Sequential File
  - One field is the key filed
    - Uniquely identifies the record
    - Records are stored in key sequence

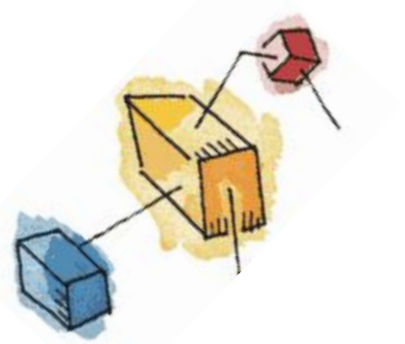


The Sequential File



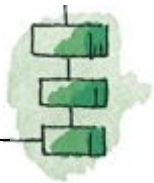
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

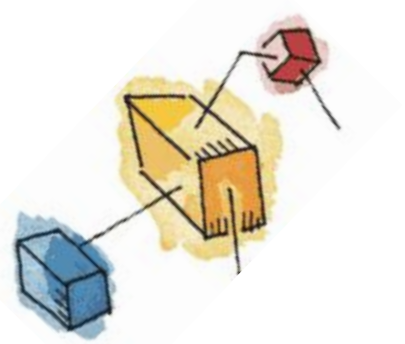
(b) Sequential File



File Organization

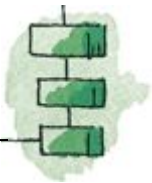
- Indexed Sequential File
  - Index provides a lookup capability to quickly reach the vicinity of the desired record
    - Contains key field and a pointer to the main file
    - Indexed is searched to find highest key value that is equal to or precedes the desired key value
    - Search continues in the main file at the location indicated by the pointer

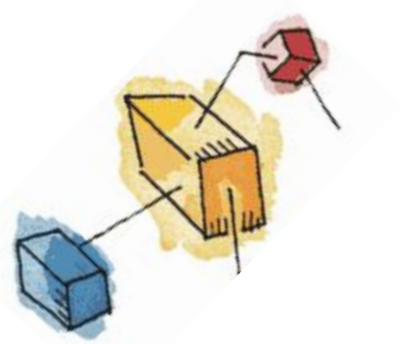




File Organization

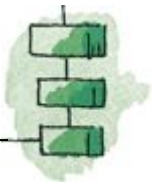
- Comparison of sequential and indexed sequential
  - Example: a file contains 1 million records
  - On average 500,00 accesses are required to find a record in a sequential file
  - If an index contains 1000 entries, it will take on average 500 accesses to find the key, followed by 500 accesses in the main file. Now on average it is 1000 accesses



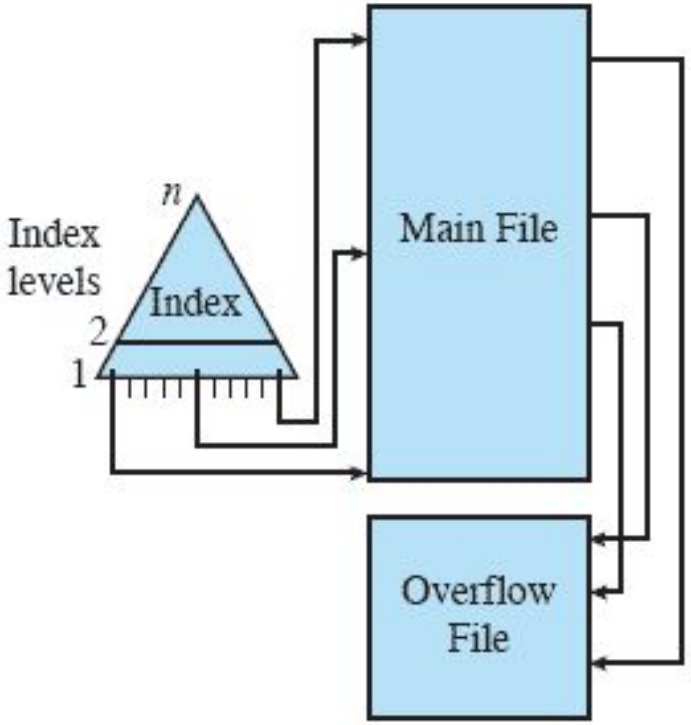


File Organization

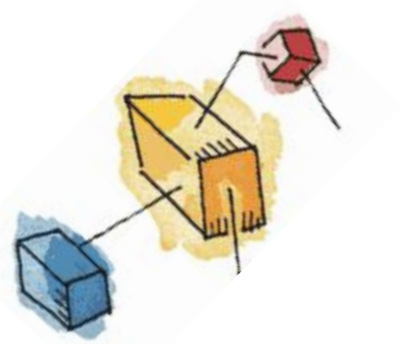
- Indexed Sequential File
  - New records are added to an overflow file
  - Record in main file that precedes it is updated to contain a pointer to the new record
  - The overflow is merged with the main file during a batch update
  - Multiple indexes for the same key field can be set up to increase efficiency



Indexed Sequential File

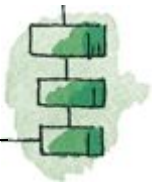


(c) Indexed Sequential File

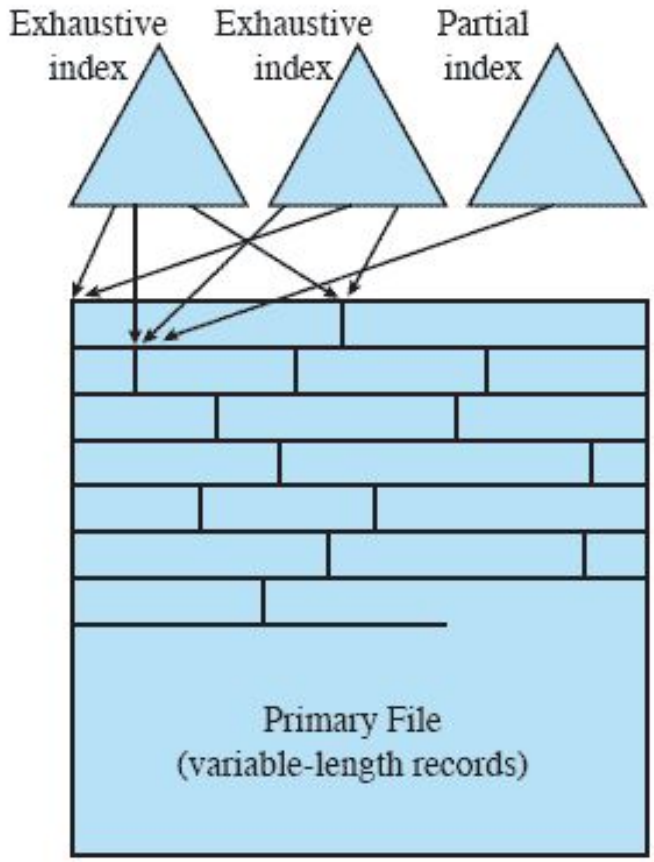


File Organization

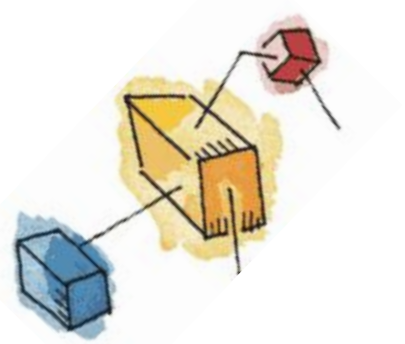
- Indexed File
  - Uses multiple indexes for different key fields
  - May contain an exhaustive index that contains one entry for every record in the main file
  - May contain a partial index



Indexed File

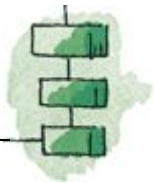


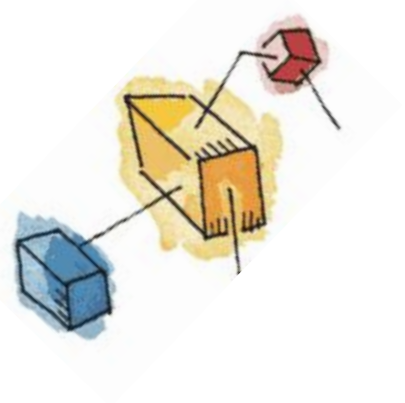
(d) **Indexed File**



File Organization

- The Direct or Hashed File
  - Directly access a block at a known address
  - Key field required for each record





Performance

Table 12.1 Grades of Performance for Five Basic File Organizations [WIED87]

	Space		Update		Retrieval		
	Attributes		Record Size				
File Method	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

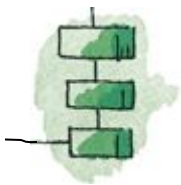
A	=	Excellent, well suited to this purpose	$\approx O(r)$
B	=	Good	$\approx O(o \times r)$
C	=	Adequate	$\approx O(r \log n)$
D	=	Requires some extra effort	$\approx O(n)$
E	=	Possible with extreme effort	$\approx O(r \times n)$
F	=	Not reasonable for this purpose	$\approx O(n^{>1})$

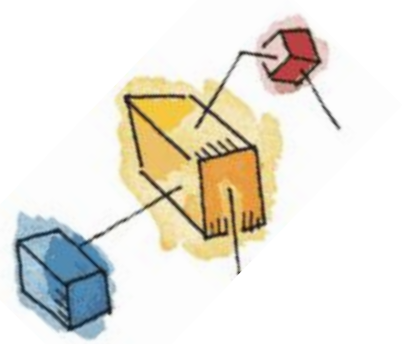
where

r = size of the result

o = number of records that overflow

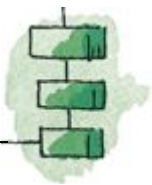
n = number of records in file

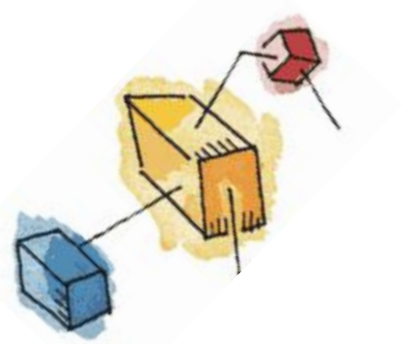




File Directories

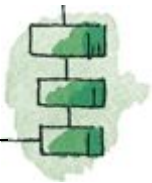
- Contains information about files
  - Attributes
  - Location
  - Ownership
- Directory itself is a file owned by the operating system
- Provides mapping between file names and the files themselves

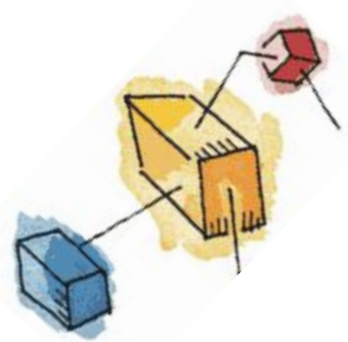




Simple Structure for a Directory

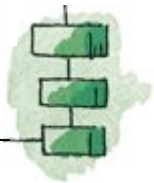
- List of entries, one for each file
- Sequential file with the name of the file serving as the key
- Provides no help in organizing the files
- Forces user to be careful not to use the same name for two different files





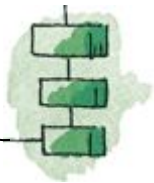
Information Elements of a File Directory

	Basic Information
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
	Address Information
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file



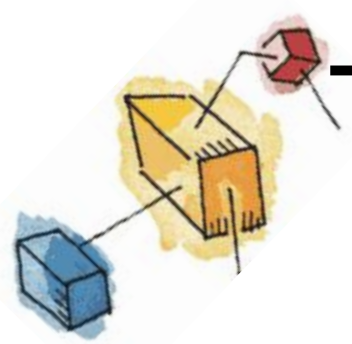
Information Elements of a File Directory

Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network



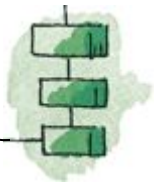
Information Elements of a File Directory

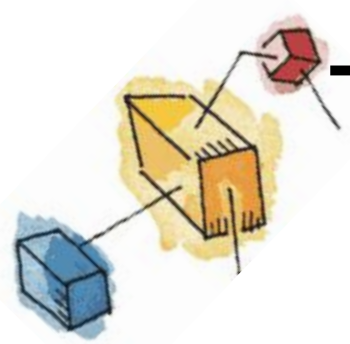
Usage Information	
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk



Two-Level Scheme for a Directory

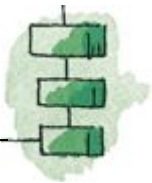
- One directory for each user and a master directory
- Master directory contains entry for each user
  - Provides address and access control information





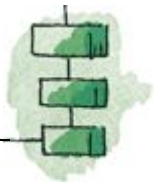
Two-Level Scheme for a Directory

- Each user directory is a simple list of files for that user
- Still provides no help in structuring collections of files



Hierarchical, or Tree-Structured Directory

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries



Tree-Structured Directory

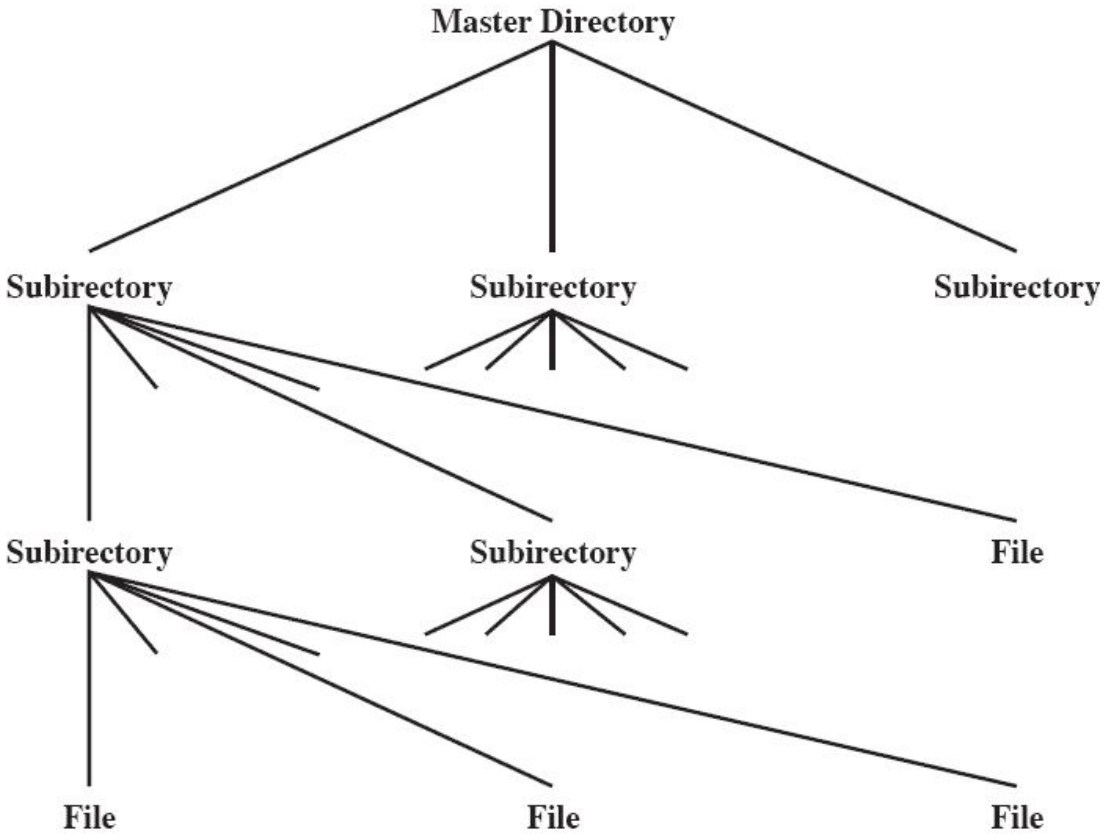


Figure 12.4 Tree-Structured Directory

Example of Tree-Structured Directory

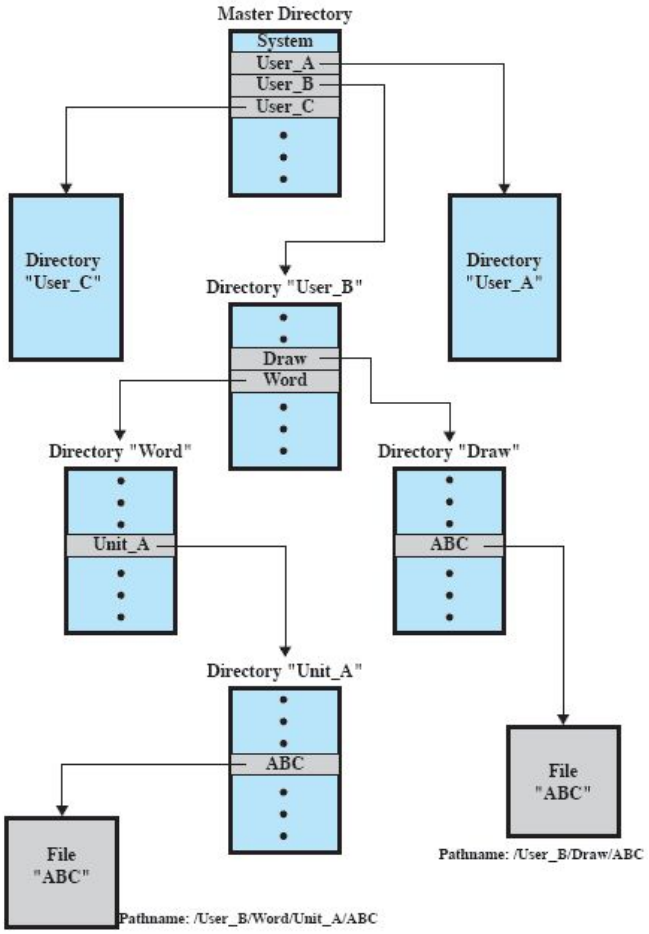
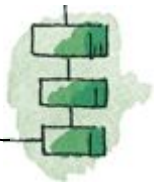


Figure 12.5 Example of Tree-Structured Directory

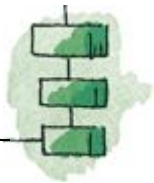
Hierarchical, or Tree-Structured Directory

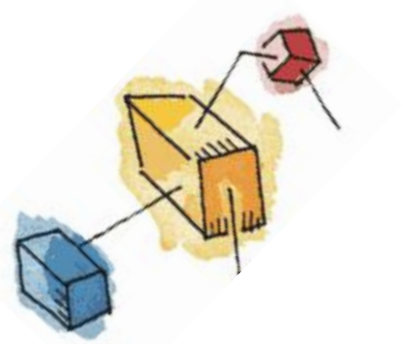
- Files can be located by following a path from the root, or master, directory down various branches
  - This is the pathname for the file
- Can have several files with the same file name as long as they have unique path names



Hierarchical, or Tree-Structured Directory

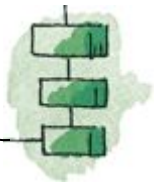
- Current directory is the working directory
- Files are referenced relative to the working directory

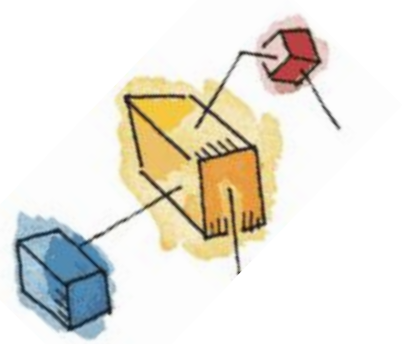




File Sharing

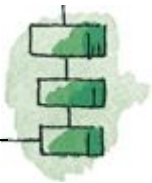
- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights
  - Management of simultaneous access

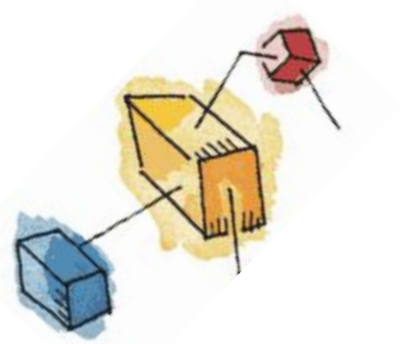




Access Rights

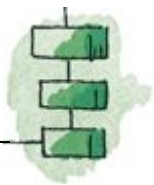
- None
  - User may not know of the existence of the file
  - User is not allowed to read the user directory that includes the file
- Knowledge
  - User can only determine that the file exists and who its owner is

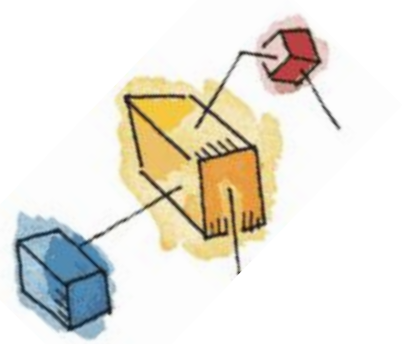




Access Rights

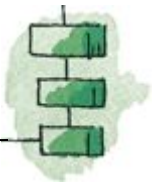
- Execution
  - The user can load and execute a program but cannot copy it
- Reading
  - The user can read the file for any purpose, including copying and execution
- Appending
  - The user can add data to the file but cannot modify or delete any of the file's contents

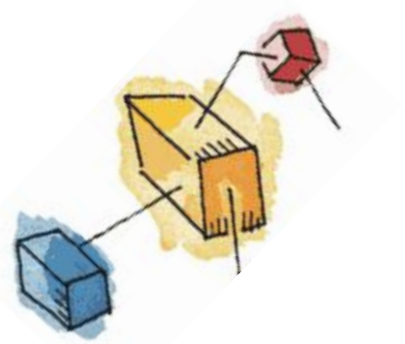




Access Rights

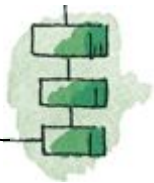
- Updating
  - The user can modify, deleted, and add to the file's data. This includes creating the file, rewriting it, and removing all or part of the data
- Changing protection
  - User can change access rights granted to other users
- Deletion
  - User can delete the file

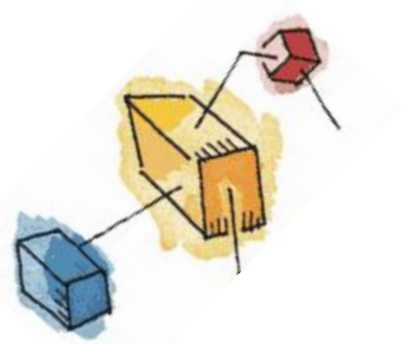




Access Rights

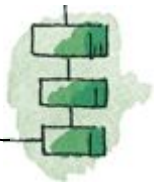
- Owners
  - Has all rights previously listed
  - May grant rights to others using the following classes of users
    - Specific user
    - User groups
    - All for public files



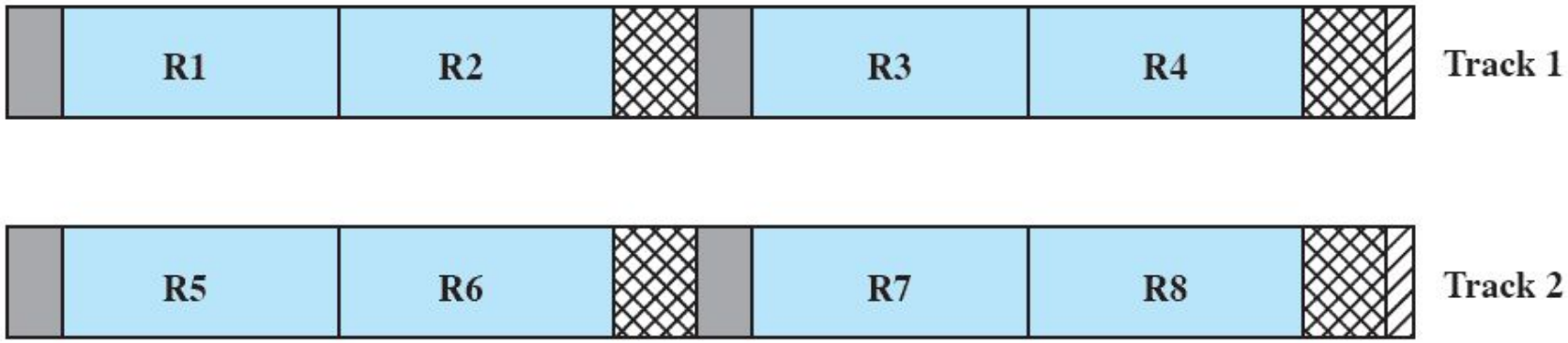


Simultaneous Access

- User may lock entire file when it is to be updated
- User may lock the individual records during the update
- Mutual exclusion and deadlock are issues for shared access



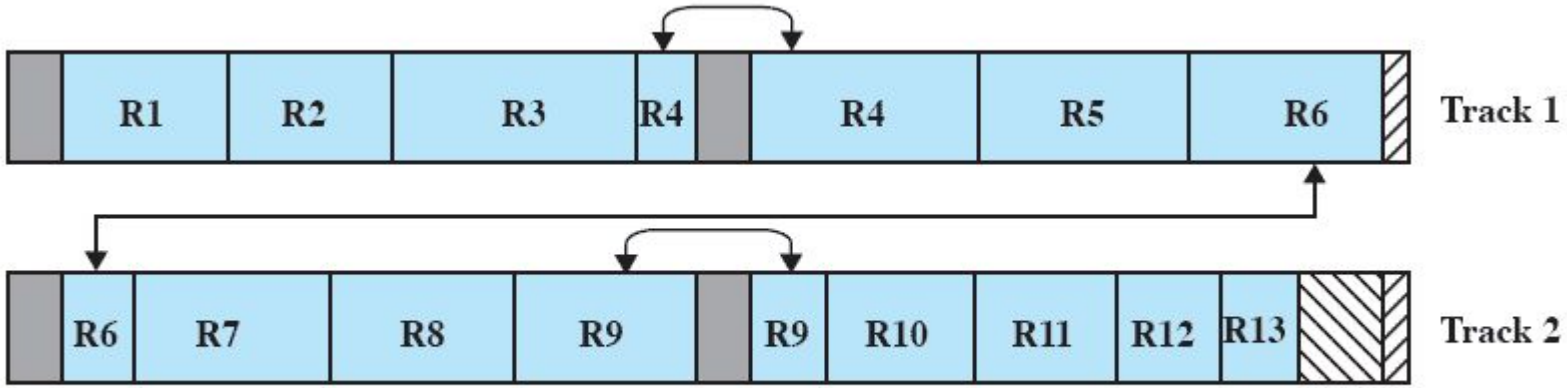
Fixed Blocking



Fixed Blocking

- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

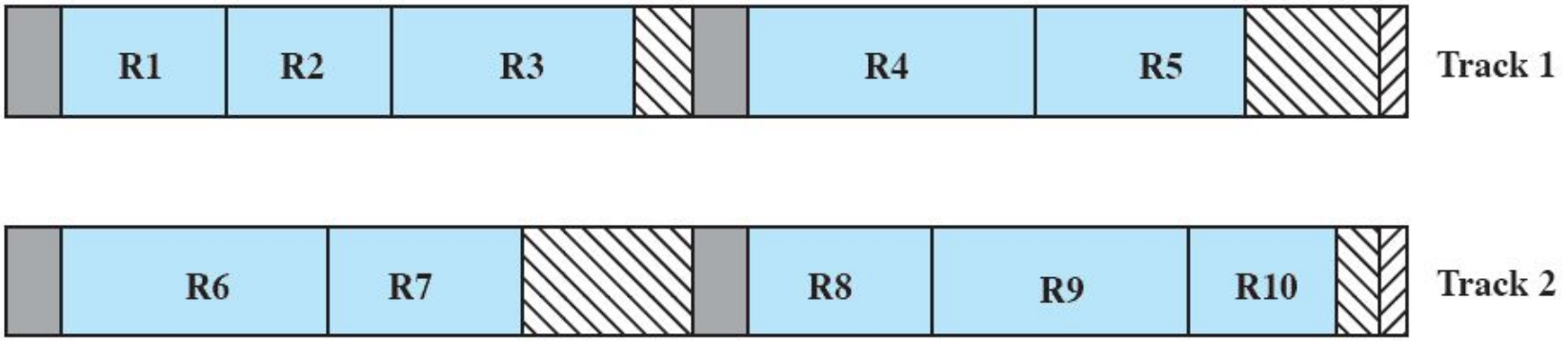
Variable Blocking: Spanned



Variable Blocking: Spanned

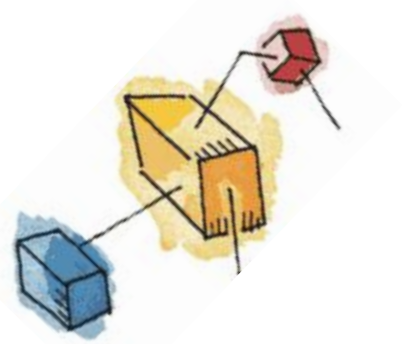
- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

Variable Blocking: Unspanned



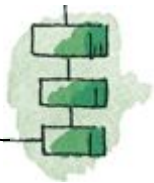
Variable Blocking: Unspanned

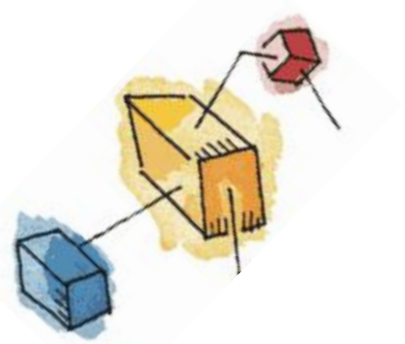
- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size



Secondary Storage Management

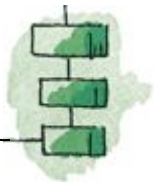
- Space must be allocated to files
- Must keep track of the space available for allocation

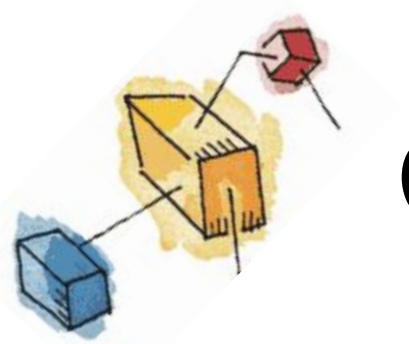




Preallocation

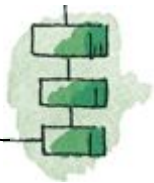
- Need the maximum size for the file at the time of creation
- Difficult to reliably estimate the maximum potential size of the file
- Tend to overestimate file size so as not to run out of space



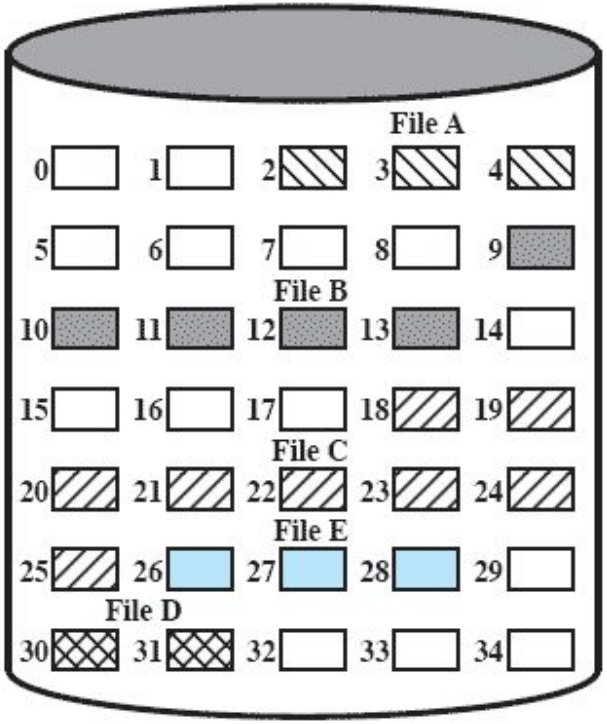


Contiguous Allocation

- Single set of blocks is allocated to a file at the time of creation
- Only a single entry in the file allocation table
  - Starting block and length of the file
- External fragmentation will occur
  - Need to perform compaction



Contiguous File Allocation

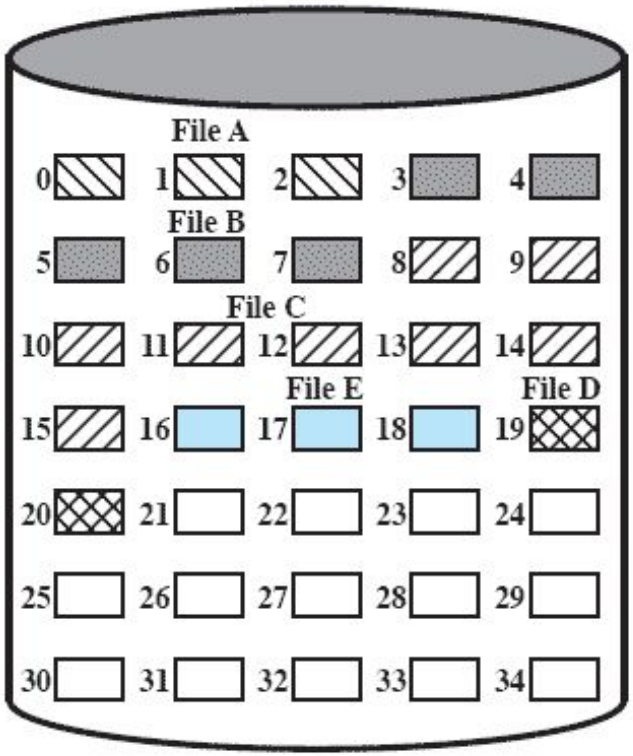


File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.7 Contiguous File Allocation

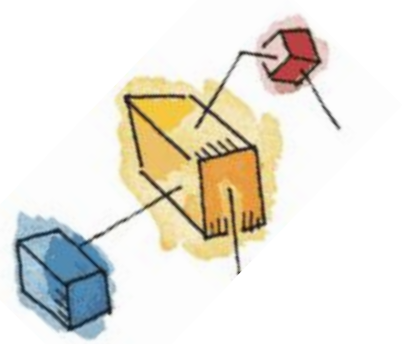
Contiguous File Allocation



File Allocation Table

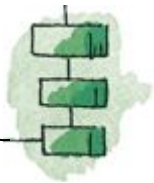
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

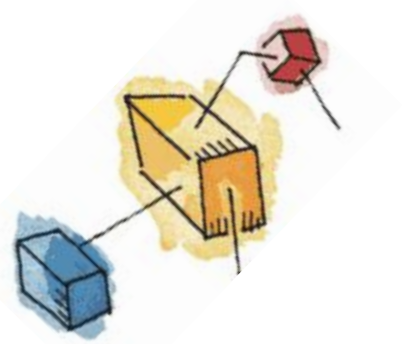
Figure 12.8 Contiguous File Allocation (After Compaction)



Chained Allocation

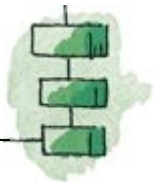
- Allocation on basis of individual block
- Each block contains a pointer to the next block in the chain
- Only single entry in the file allocation table
  - Starting block and length of file



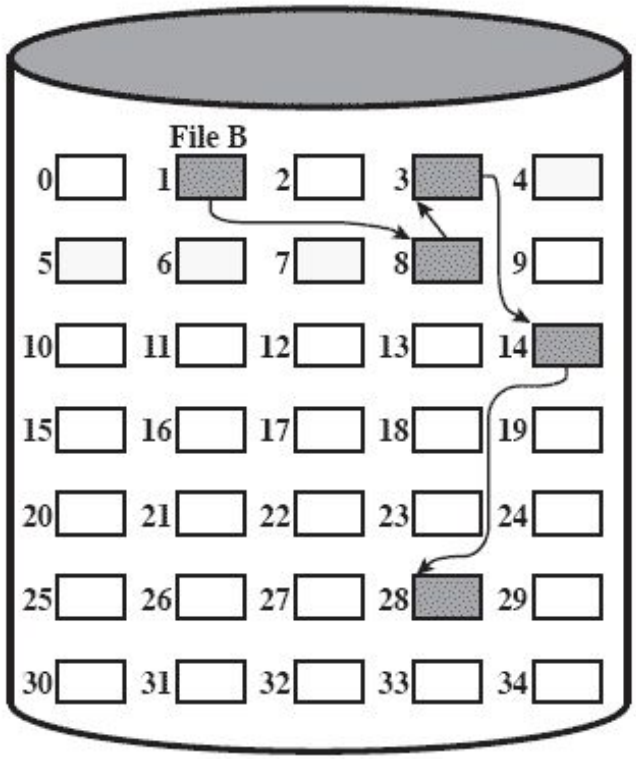


Chained Allocation

- No external fragmentation
- Best for sequential files
- No accommodation of the principle of locality



Chained Allocation

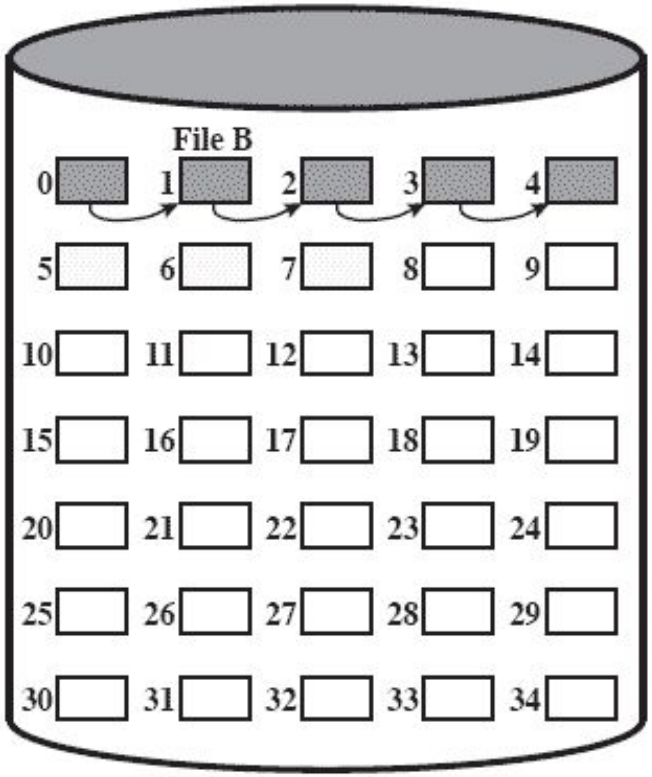


File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.9 Chained Allocation

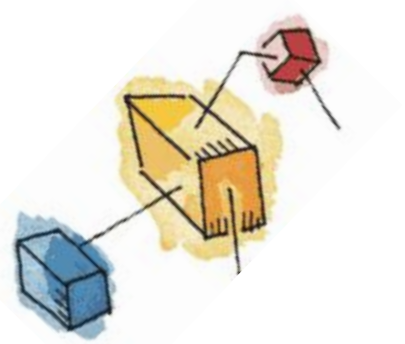
Chained Allocation



File Allocation Table

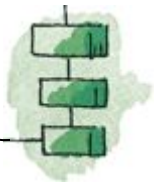
File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.10 Chained Allocation (After Consolidation)



Indexed Allocation

- File allocation table contains a separate one-level index for each file
- The index has one entry for each portion allocated to the file
- The file allocation table contains block number for the index



Indexed Allocation

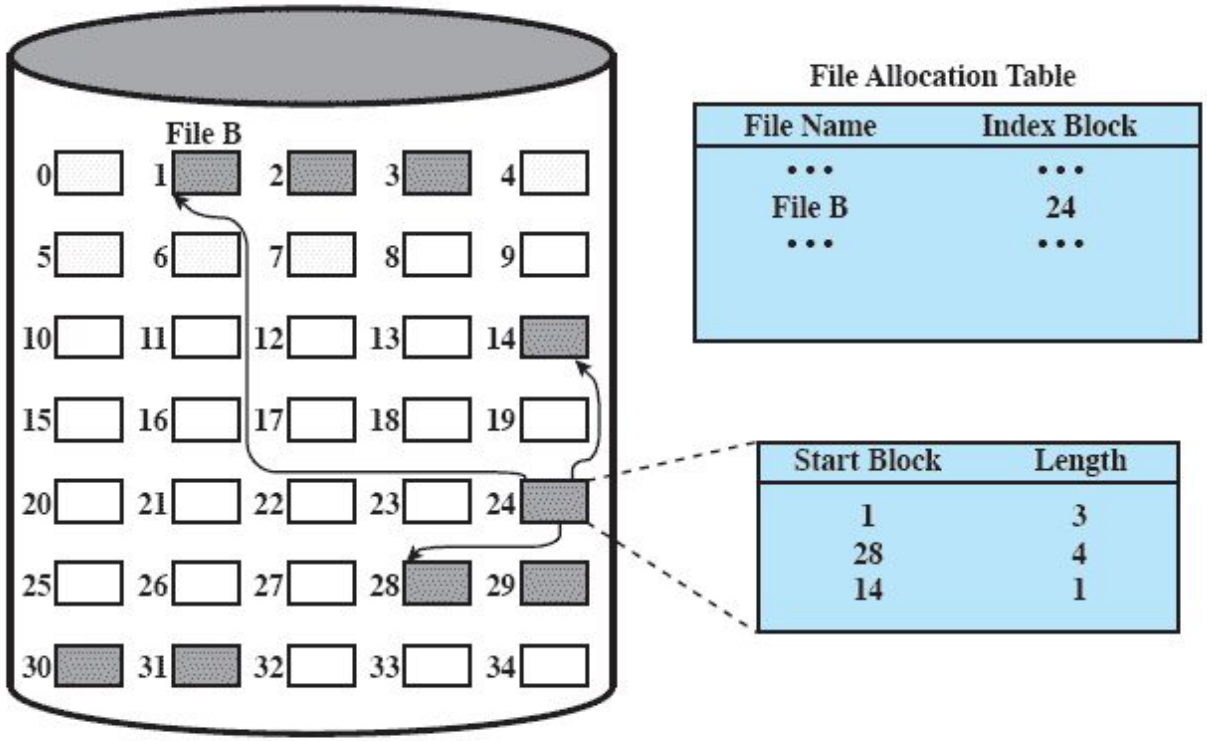


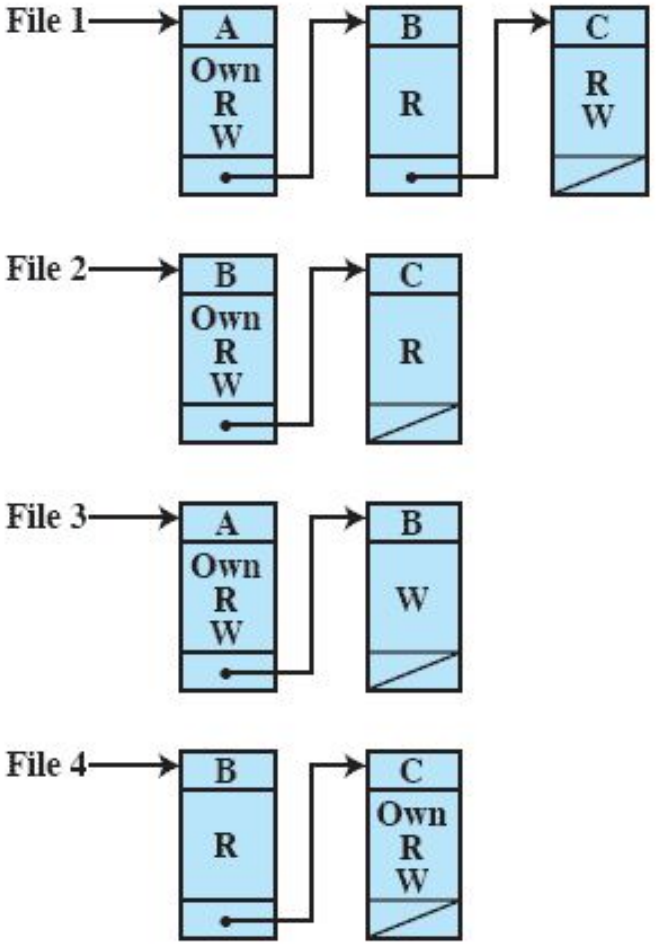
Figure 12.12 Indexed Allocation with Variable-Length Portions

Access Matrix

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

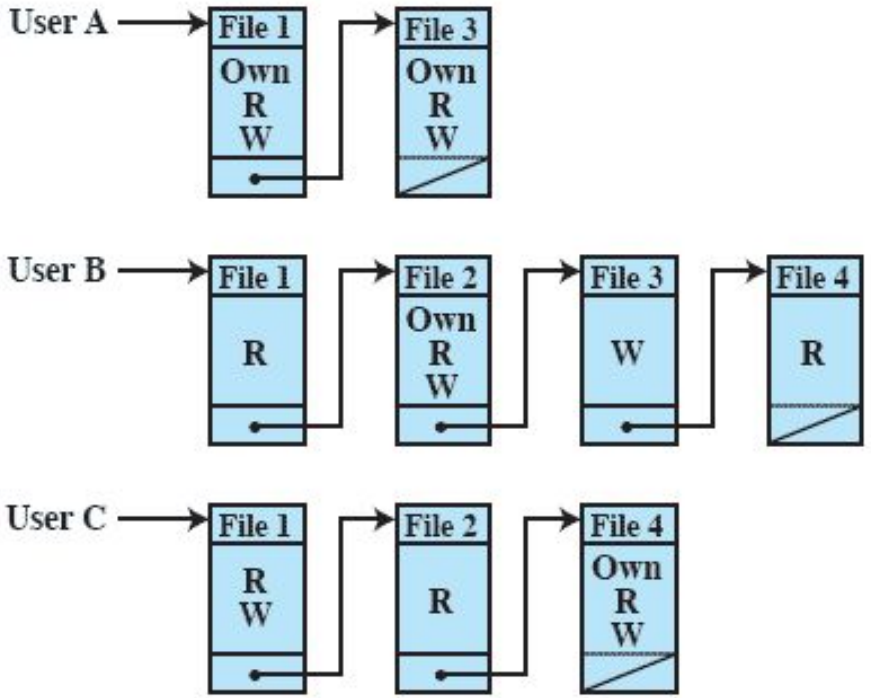
(a) Access matrix

Access Control List



(b) Access control lists for files of part (a)

Capability Lists



(c) Capability lists for files of part (a)

Operating Systems: Internals and Design Principles

Chapter 11
I/O Management and Disk Scheduling

Seventh Edition
By William Stallings

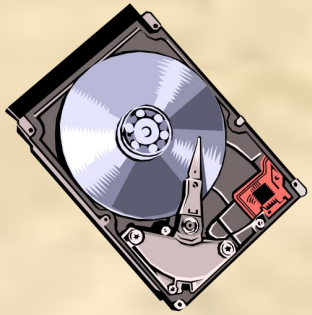
Operating Systems: Internals and Design Principles

An artifact can be thought of as a meeting point—an "interface" in today's terms between an "inner" environment, the substance and organization of the artifact itself, and an "outer" environment, the surroundings in which it operates. If the inner environment is appropriate to the outer environment, or vice versa, the artifact will serve its intended purpose.

— *THE SCIENCES OF THE ARTIFICIAL,*
Herbert Simon

Categories of I/O Devices

External devices that engage in I/O with computer systems can be grouped into three categories:



Human readable

- suitable for communicating with the computer user
- printers, terminals, video display, keyboard, mouse

Machine readable

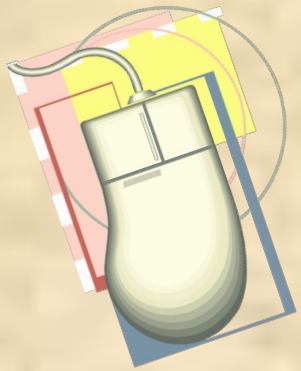
- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers

Communication

- suitable for communicating with remote devices
- modems, digital line drivers

Differences in I/O Devices

- Devices differ in a number of areas:



Data Rate

- there may be differences of magnitude between the data transfer rates

Application

- the use to which a device is put has an influence on the software

Complexity of Control

- the effect on the operating system is filtered by the complexity of the I/O module that controls the device

Unit of Transfer

- data may be transferred as a stream of bytes or characters or in larger blocks

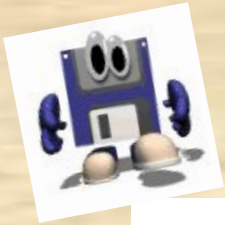
Data Representation

- different data encoding schemes are used by different devices

Error Conditions

- the nature of errors, the way in which they are reported, their consequences, and available range of responses differs from one device to another

the



Data Rates

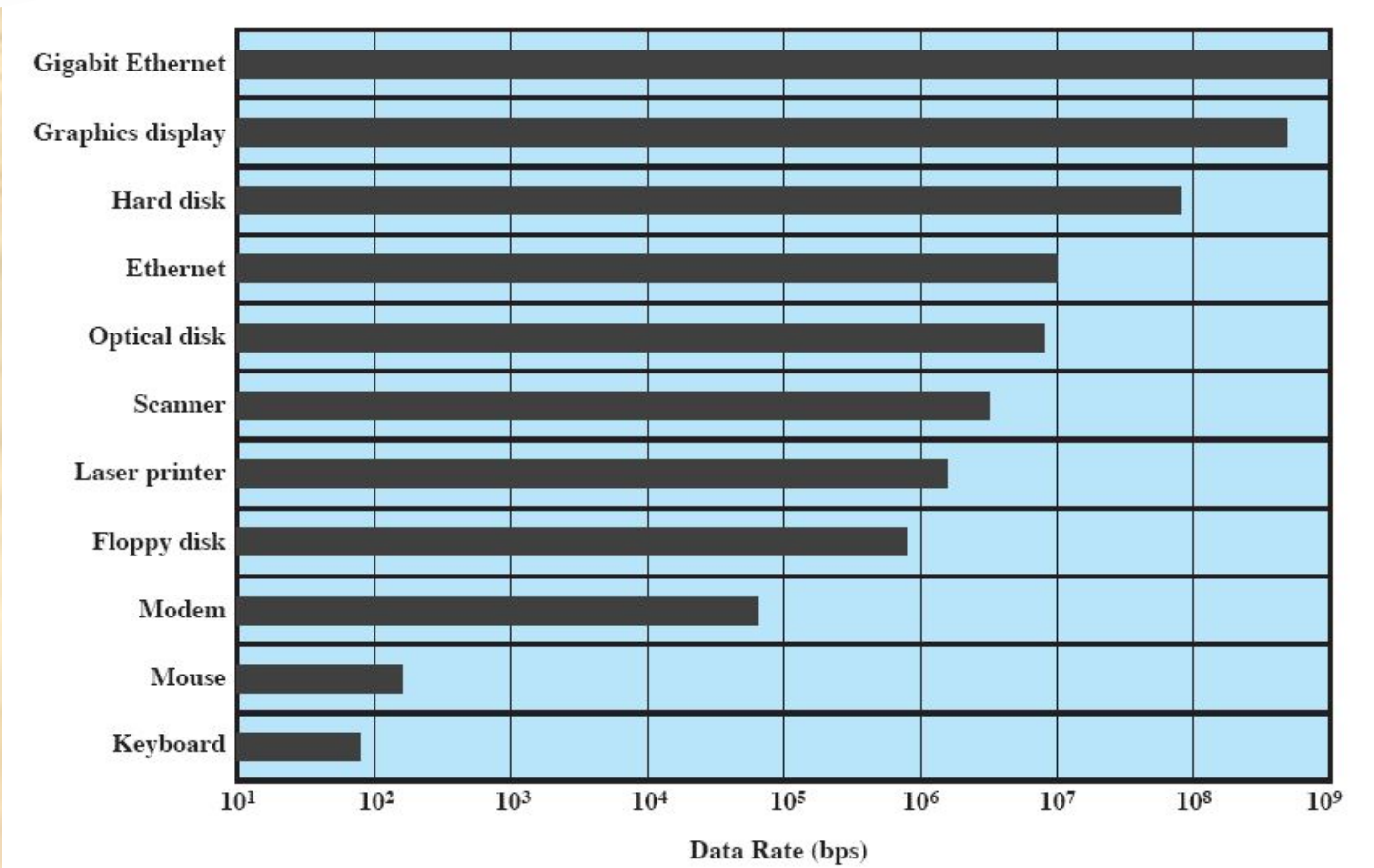


Figure 11.1 Typical I/O Device Data Rates

Organization of the I/O Function

- Three techniques for performing I/O are:
- **Programmed I/O**
  - the processor issues an I/O command on behalf of a process to an I/O module; that process then busy waits for the operation to be completed before proceeding
- **Interrupt-driven I/O**
  - the processor issues an I/O command on behalf of a process
    - if non-blocking – processor continues to execute instructions from the process that issued the I/O command
    - if blocking – the next instruction the processor executes is from the OS, which will put the current process in a blocked state and schedule another process
- **Direct Memory Access (DMA)**
  - a DMA module controls the exchange of data between main memory and an I/O module

Techniques for Performing I/O

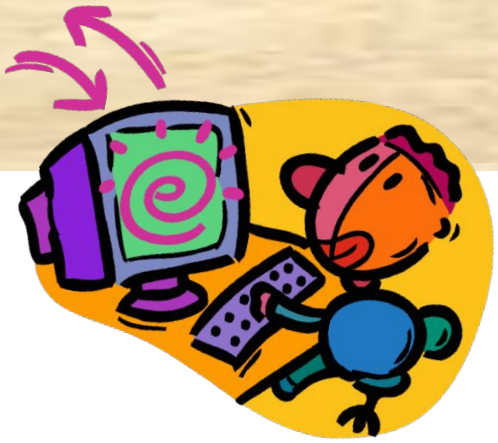


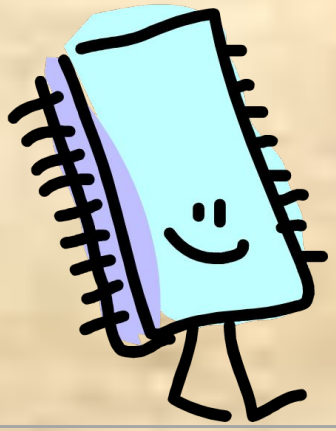
Table 11.1 I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Evolution of the I/O Function

1. Processor directly controls a peripheral device
2. A controller or I/O module is added
3. Same configuration as step 2, but now interrupts are employed
4. The I/O module is given direct control of memory via DMA
5. The I/O module is enhanced to become a separate processor, with a specialized instruction set tailored for I/O
6. The I/O module has a local memory of its own and is, in fact, a computer in its own right

Direct Memory Access



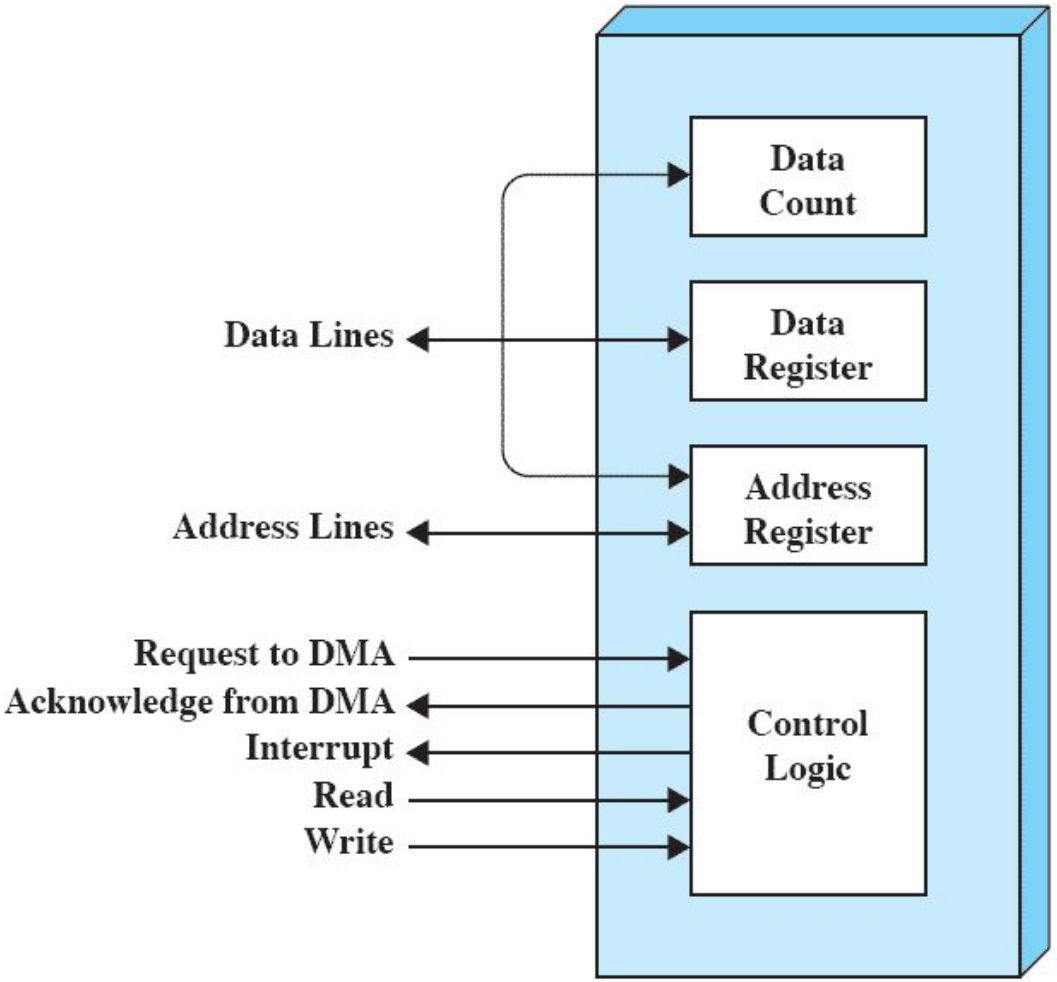
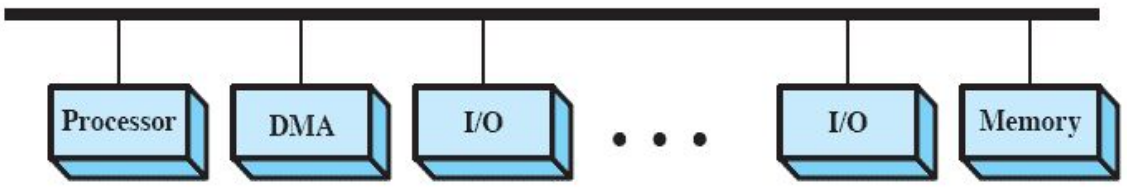


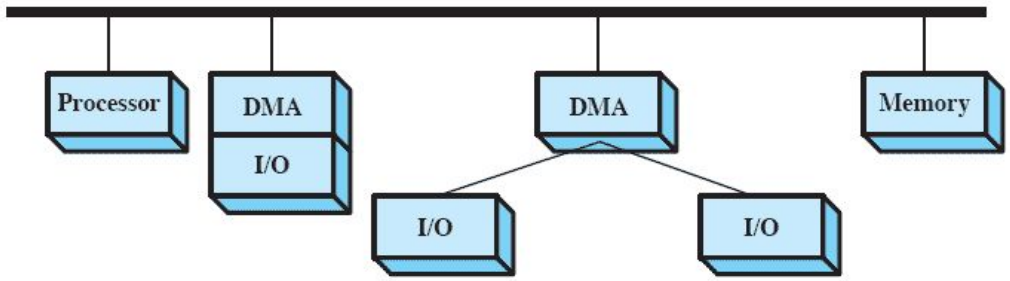
Figure 11.2 Typical DMA Block Diagram

DMA

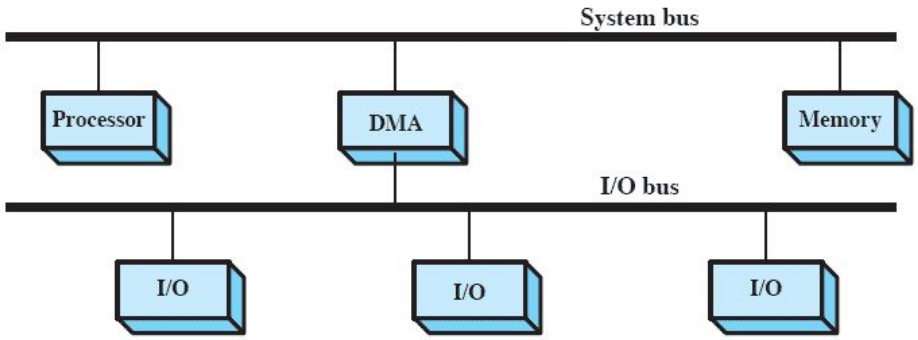


(a) Single-bus, detached DMA

Alternative



(b) Single-bus, Integrated DMA-I/O



(c) I/O bus

Configurations

Design Objectives

Efficiency

- Major effort in I/O design
- Important because I/O operations often form a bottleneck
- Most I/O devices are extremely slow compared with main memory and the processor
- The area that has received the most attention is disk I/O

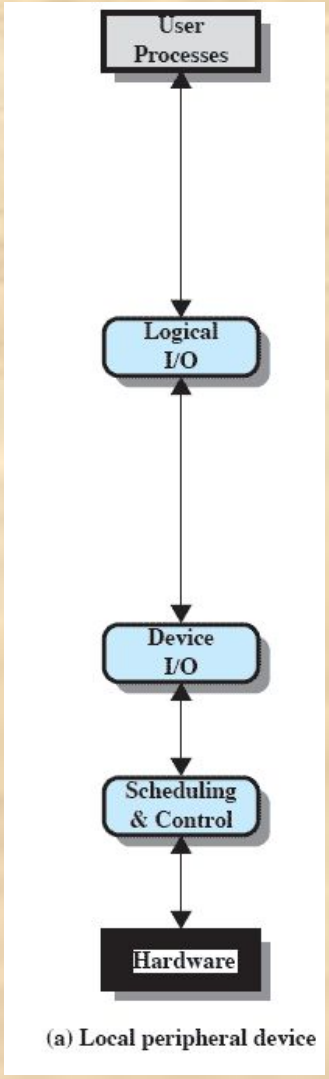
Generality

- Desirable to handle all devices in a uniform manner
- Applies to the way processes view I/O devices and the way the operating system manages I/O devices and operations
- Diversity of devices makes it difficult to achieve true generality
- Use a hierarchical, modular approach to the design of the I/O function

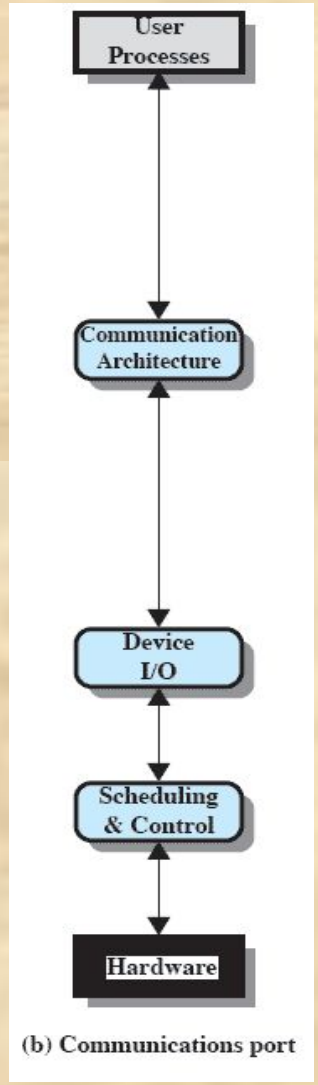
Hierarchical Design

- Functions of the operating system should be separated according to their complexity, their characteristic time scale, and their level of abstraction
- Leads to an organization of the operating system into a series of layers
- Each layer performs a related subset of the functions required of the operating system
- Layers should be defined so that changes in one layer do not require changes in other layers

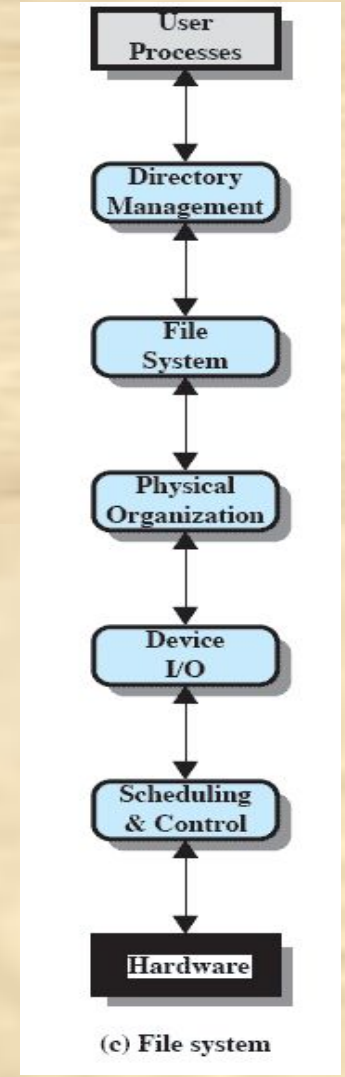
A Model of I/O Organization



(a) Local peripheral device



(b) Communications port



(c) File system

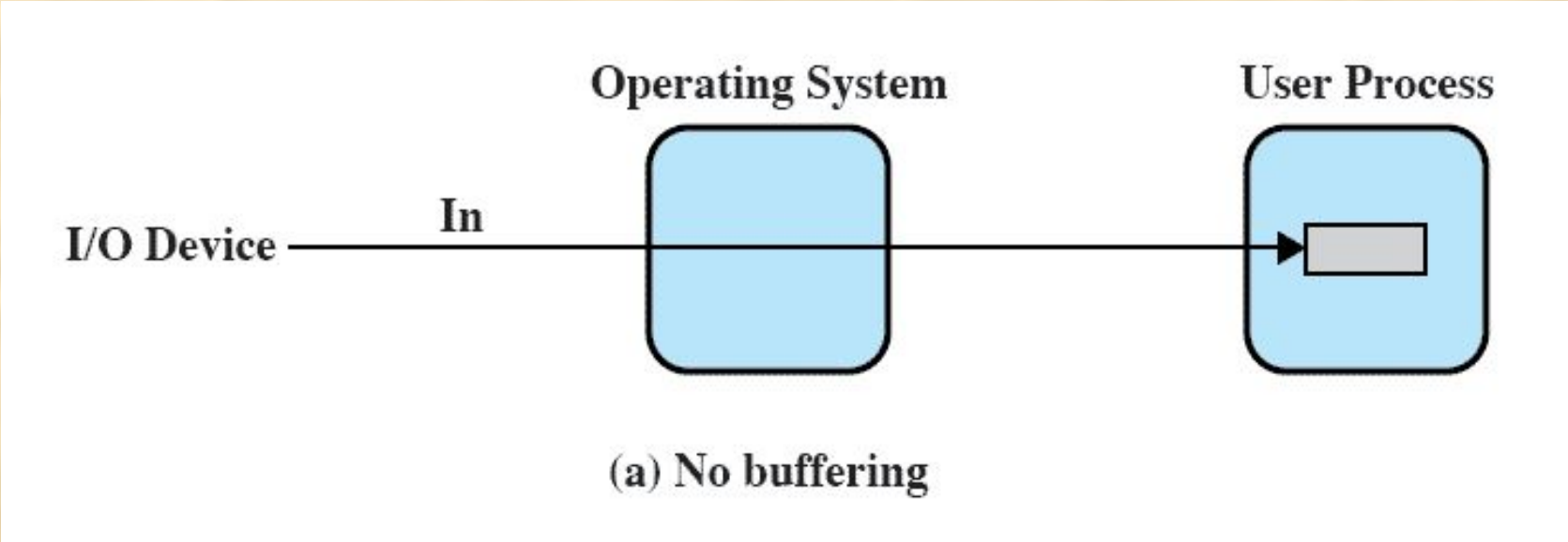
Buffering

- Perform input transfers in advance of requests being made and perform output transfers some time after the request is made

Block-oriented device	Stream-oriented device
• stores information in blocks that are usually of fixed size • transfers are made one block at a time • possible to reference data by its block number • disks and USB keys are examples	• transfers data in and out as a stream of bytes • no block structure • terminals, printers, communications ports, and most other devices that are not secondary storage are examples

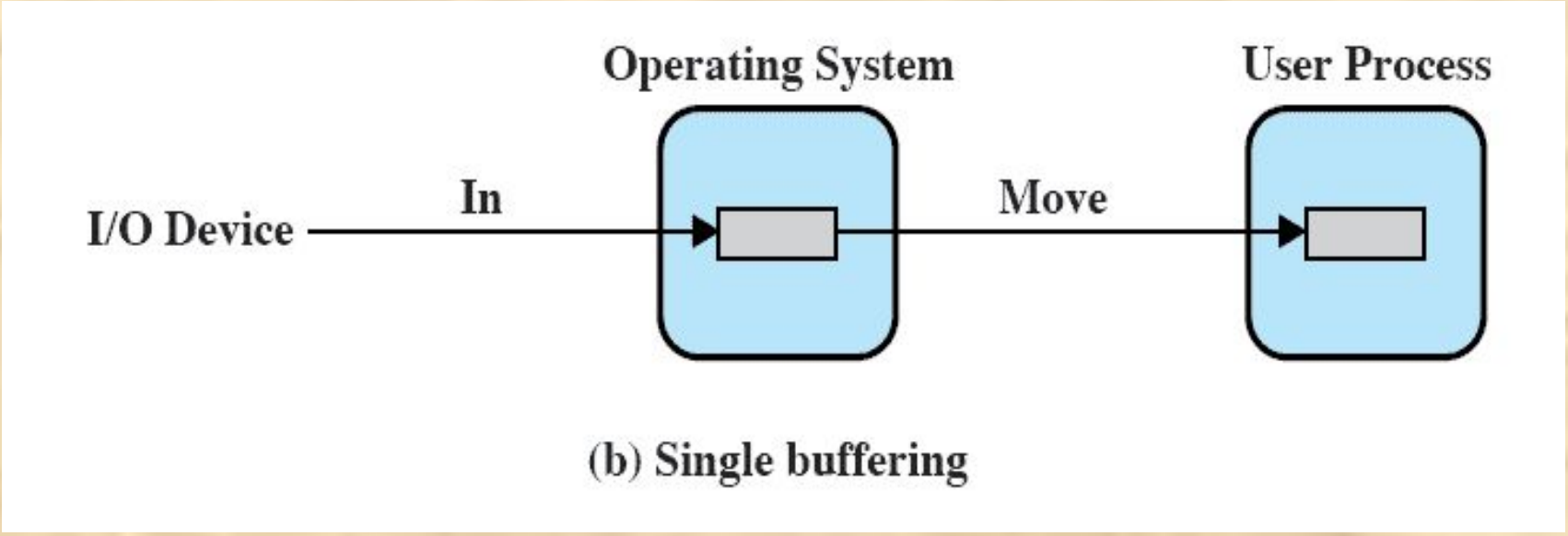
No Buffer

- Without a buffer, the OS directly accesses the device when it needs



Single Buffer

- Operating system assigns a buffer in main memory for an I/O request



(b) Single buffering

Block-Oriented Single Buffer

- Input transfers are made to the system buffer
- Reading ahead/anticipated input
  - is done in the expectation that the block will eventually be needed
  - when the transfer is complete, the process moves the block into user space and immediately requests another block
  - Generally provides a speedup compared to the lack of system buffering
  - Disadvantages:
  - complicates the logic in the operating system
  - swapping logic is also affected

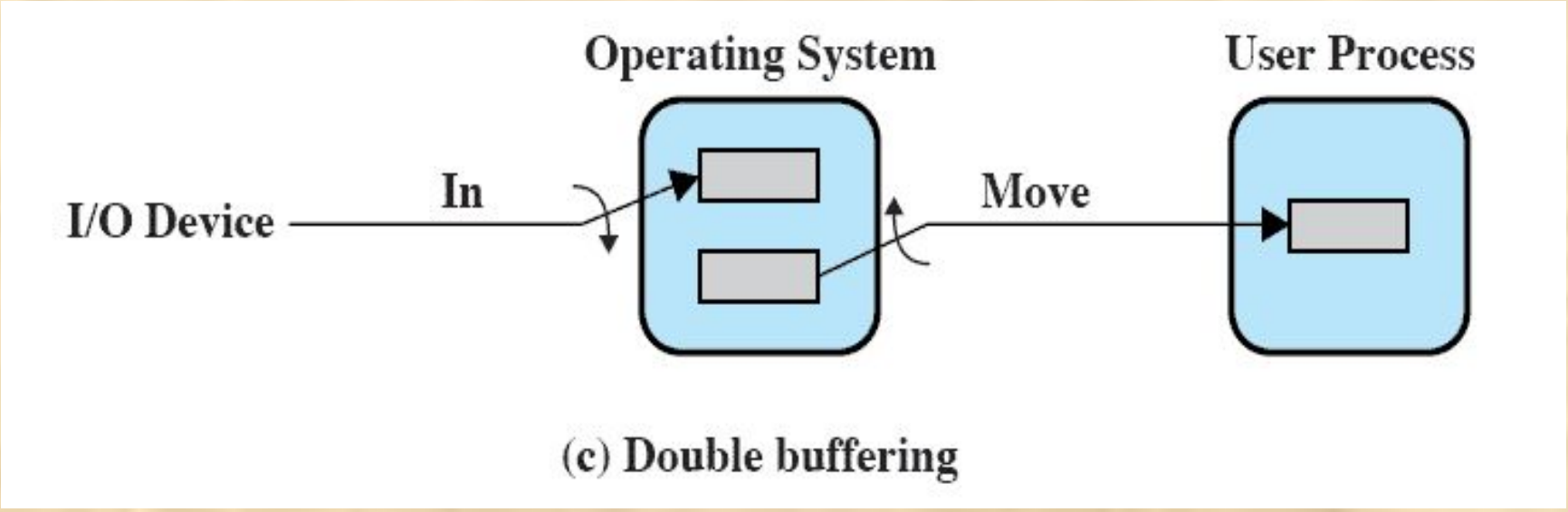
Stream-Oriented Single Buffer

- Line-at-a-time operation
  - appropriate for scroll-mode terminals (dumb terminals)
  - user input is one line at a time with a carriage return signaling the end of a line
  - output to the terminal is similarly one line at a time

- Byte-at-a-time operation
- used on forms-mode terminals
- when each keystroke is significant
- other peripherals such as sensors and controllers

Double Buffer

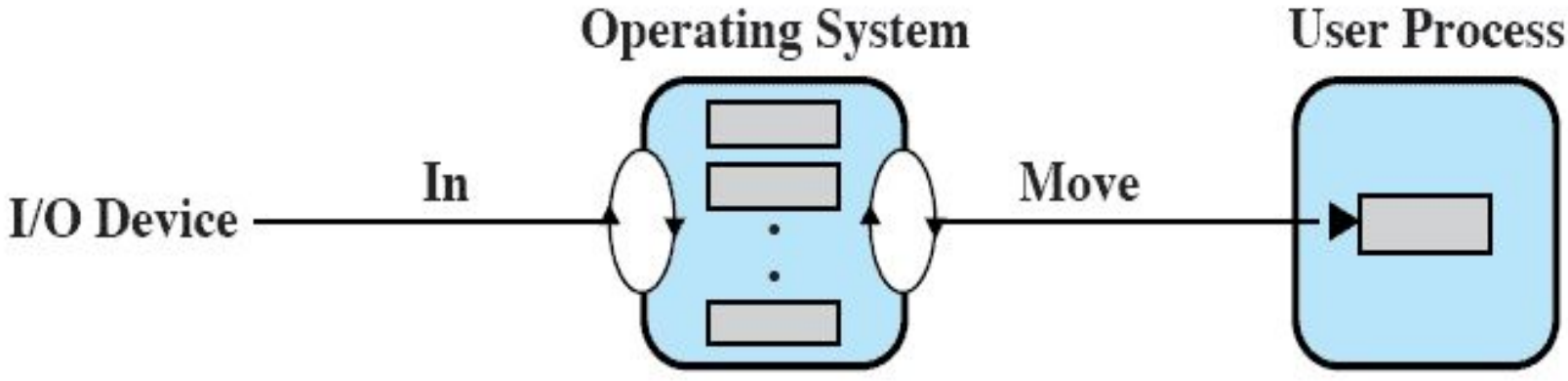
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer
- Also known as buffer swapping



(c) Double buffering

Circular Buffer

- Two or more buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process



(d) Circular buffering

The Utility of Buffering

- Technique that smoothes out peaks in I/O demand
  - with enough demand eventually all buffers become full and their advantage is lost
- When there is a variety of I/O and process activities to service, buffering can increase the efficiency of the OS and the performance of individual processes

Disk Performance Parameters

- The actual details of disk I/O operation depend on the:
  - computer system
  - operating system
  - nature of the I/O channel and disk controller hardware

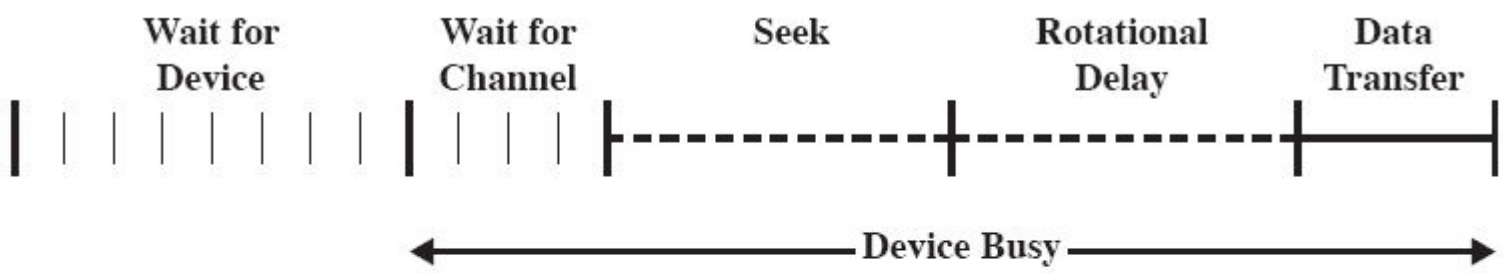


Figure 11.6 Timing of a Disk I/O Transfer

Positioning the Read/Write Heads

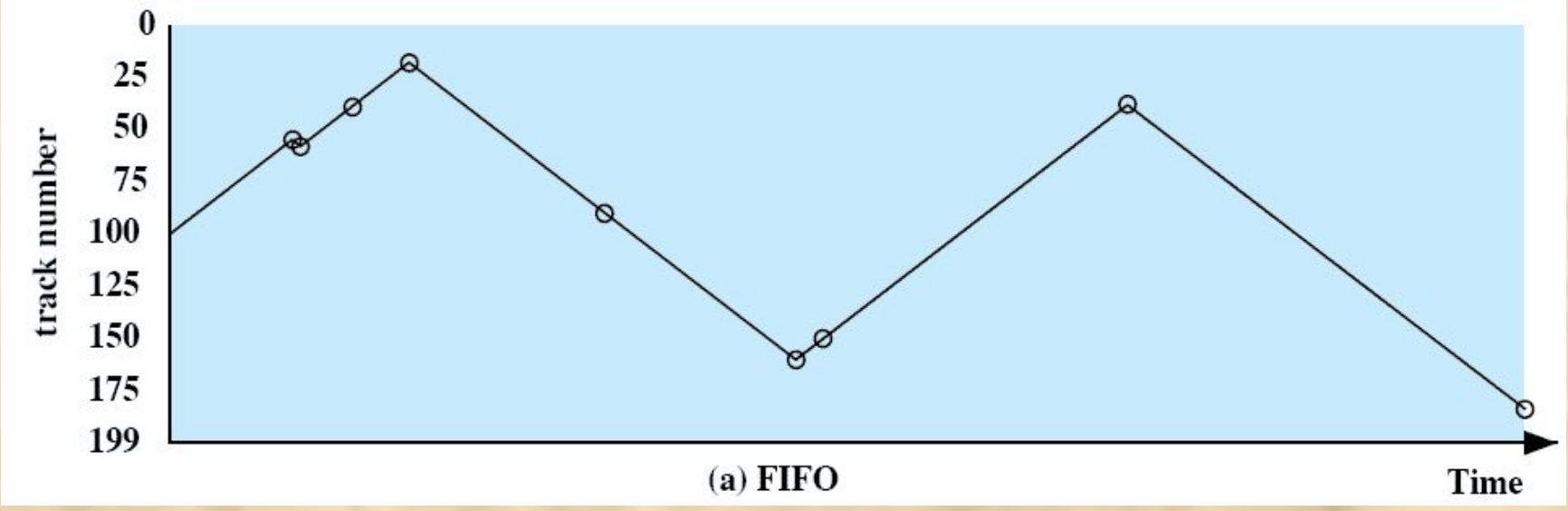
- When the disk drive is operating, the disk is rotating at constant speed
- To read or write the head must be positioned at the desired track and at the beginning of the desired sector on that track
- Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system
- On a movable-head system the time it takes to position the head at the track is known as **seek time**
- The time it takes for the beginning of the sector to reach the head is known as **rotational delay**
- The sum of the seek time and the rotational delay equals the **access time**

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Table 11.2 Comparison of Disk Scheduling Algorithms

First-In, First-Out (FIFO)

- Processes in sequential order
- Fair to all processes
- Approximates random scheduling in performance if there are many processes competing for the disk



Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N = queue size at beginning of SCAN cycle	Load sensitive

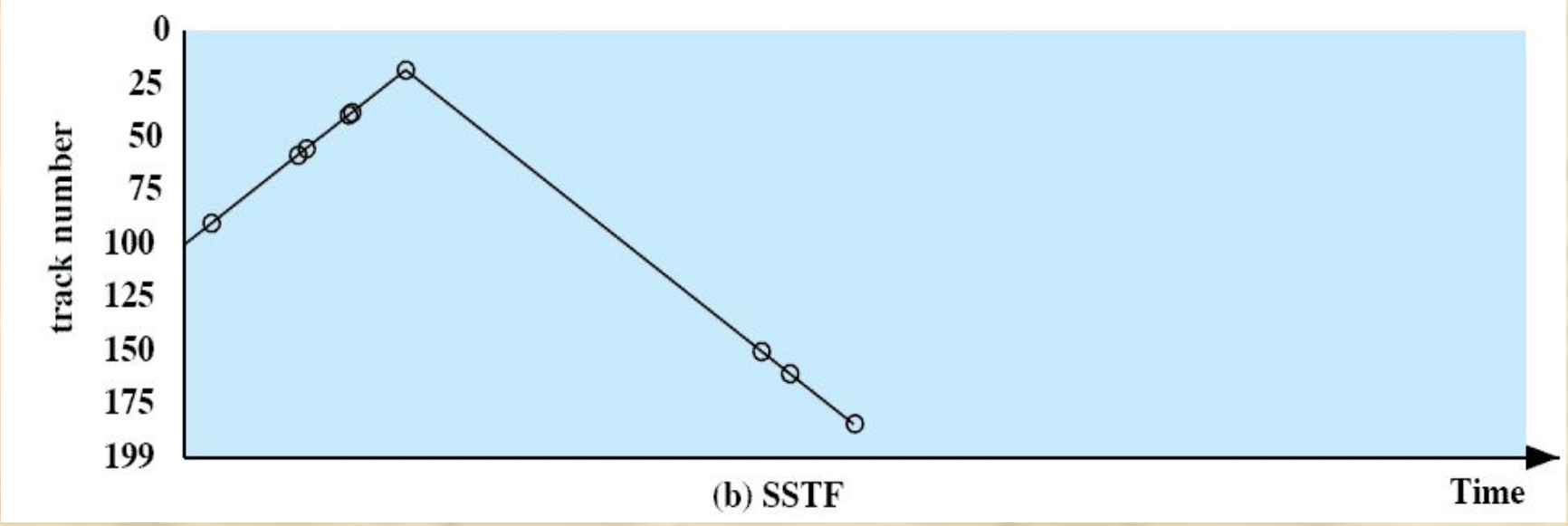
Table 11.3 Disk Scheduling Algorithms

Priority (PRI)

- Control of the scheduling is outside the control of disk management software
- Goal is not to optimize disk utilization but to meet other objectives
- Short batch jobs and interactive jobs are given higher priority
- Provides good interactive response time
- Longer jobs may have to wait an excessively long time
- A poor policy for database systems

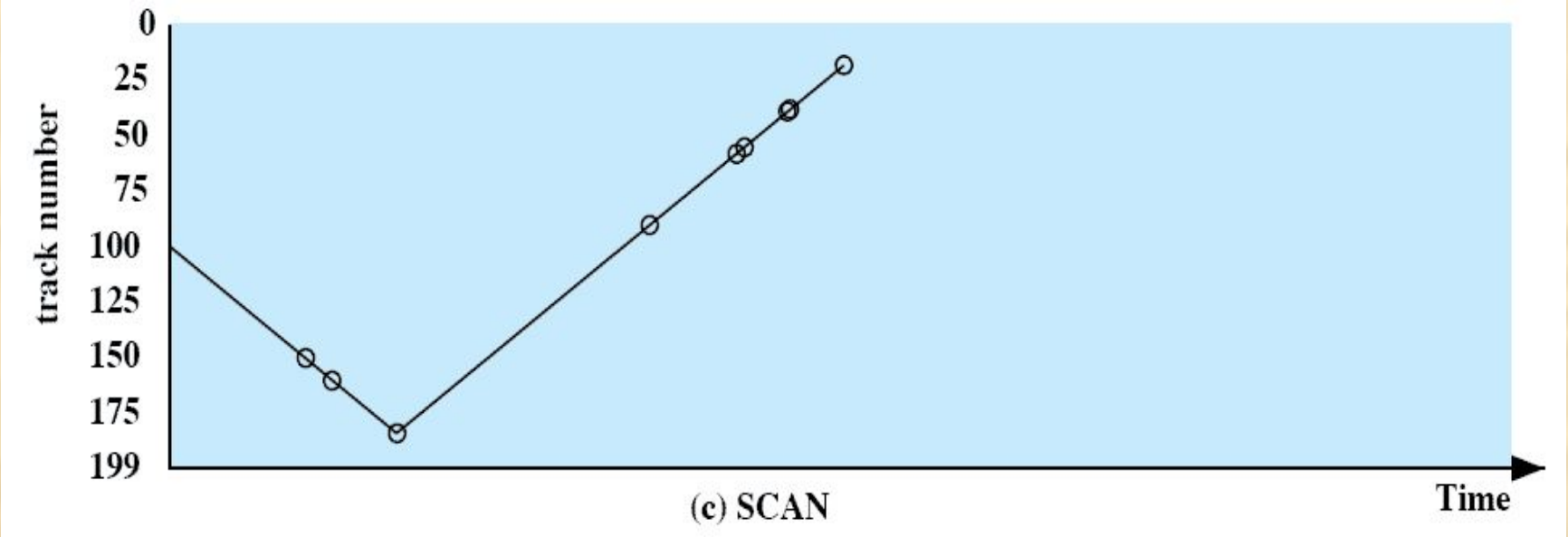
Shortest Service Time First (SSTF)

- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum seek time



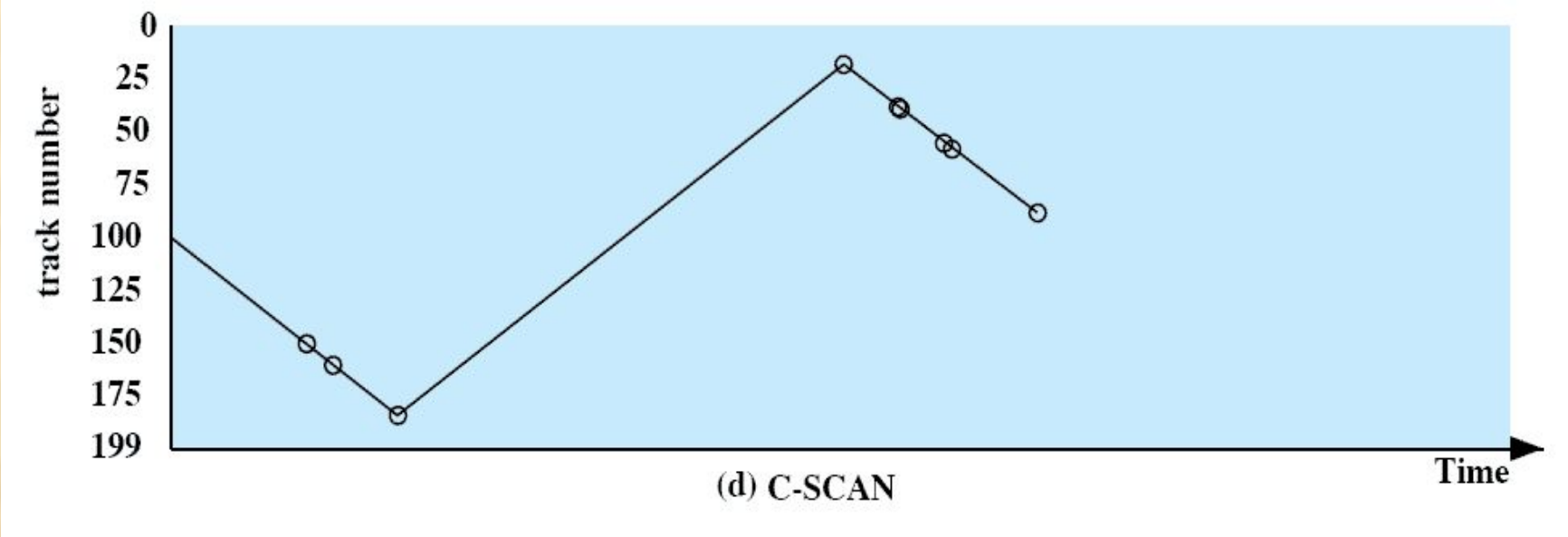
SCAN

- Also known as the elevator algorithm
- Arm moves in one direction only
  - satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed
- Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks



C-SCAN (Circular SCAN)

- Restricts scanning to one direction only
- When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again



N-Step-SCAN

- Segments the disk request queue into subqueues of length *N*
- Subqueues are processed one at a time, using SCAN
- While a queue is being processed new requests must be added to some other queue
- If fewer than *N* requests are available at the end of a scan, all of them are processed with the next scan

FSCAN

- Uses two subqueues
- When a scan begins, all of the requests are in one of the queues, with the other empty
- During scan, all new requests are put into the other queue
- Service of new requests is deferred until all of the old requests have been processed

RAID

- Redundant Array of Independent Disks
- Consists of seven levels, zero through six

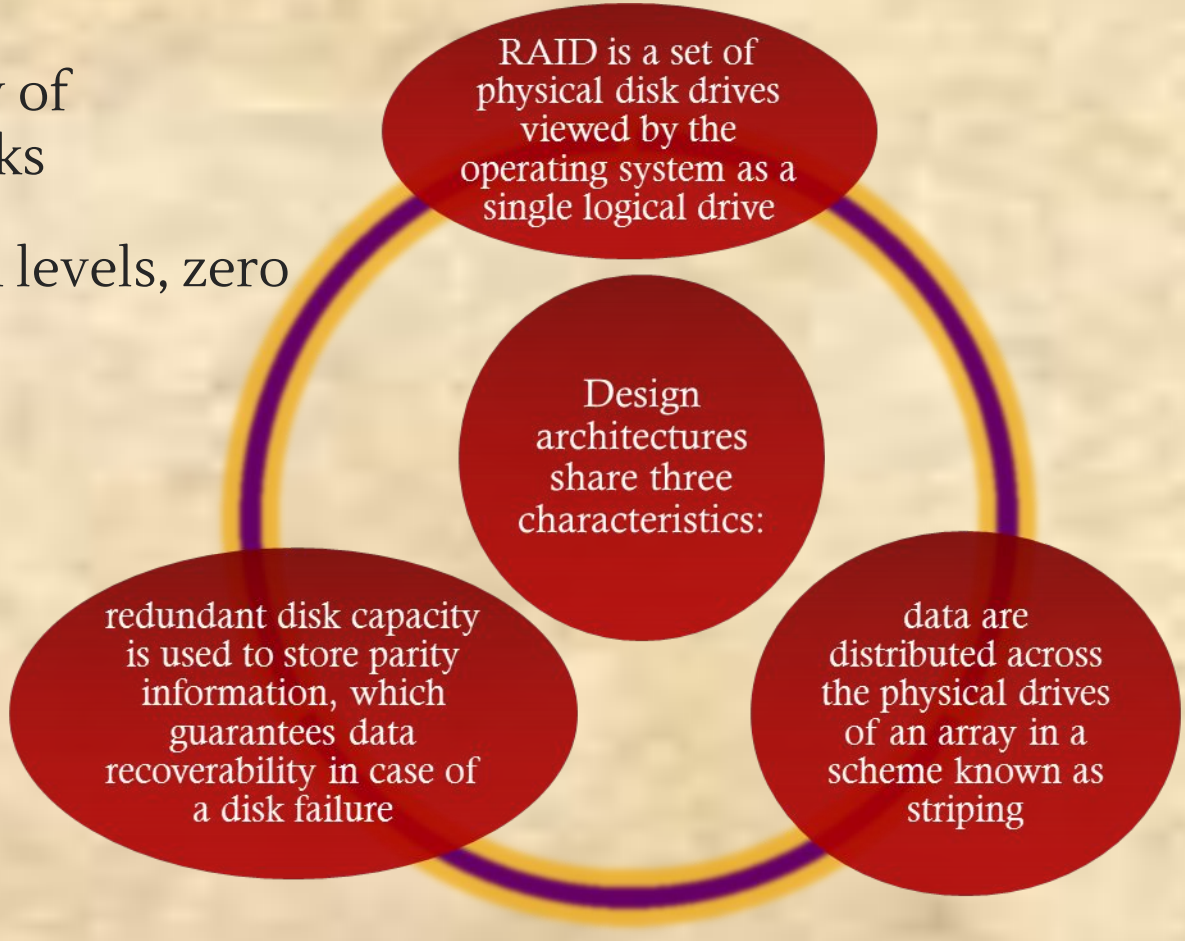


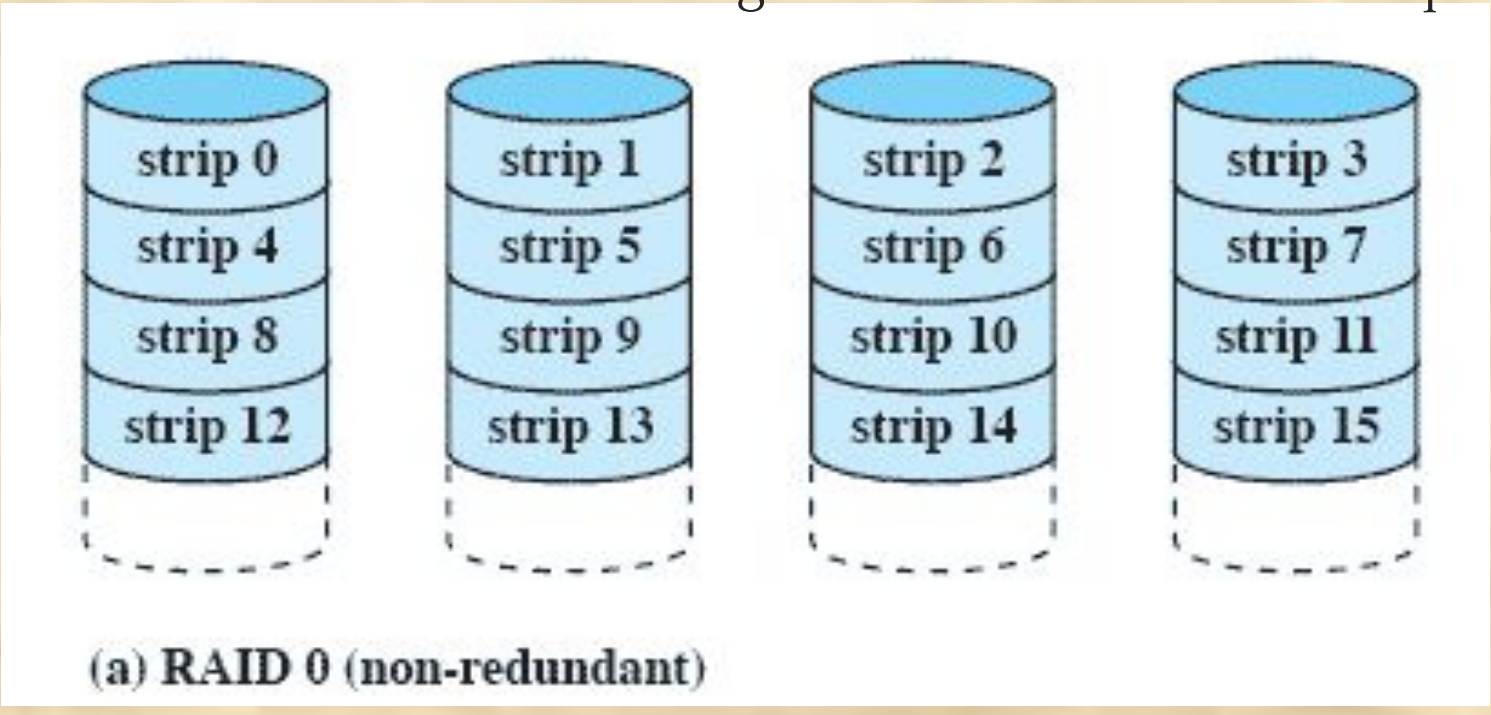
Table 11.4 RAID Levels

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

N = number of data disks; m proportional to log N

RAID Level 0

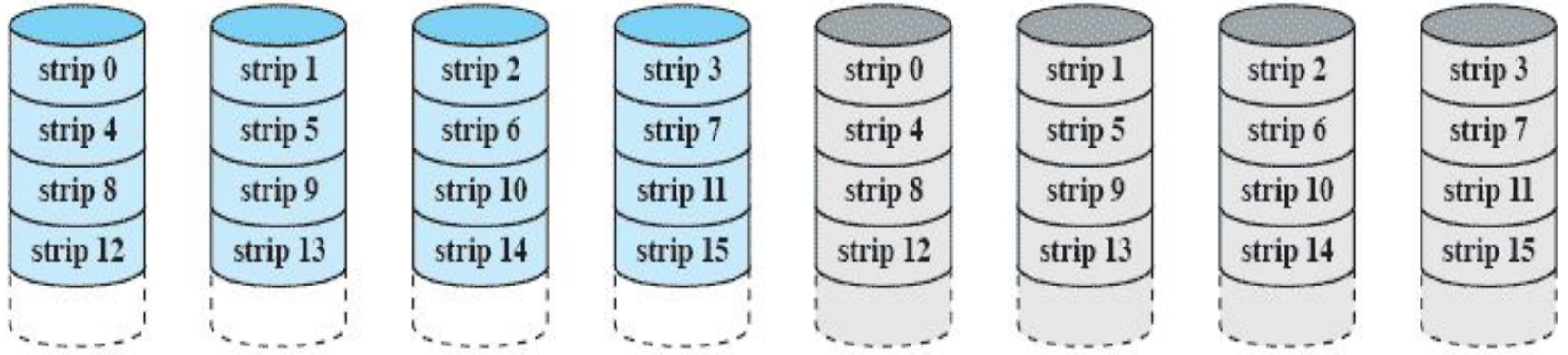
- Not a true RAID because it does not include redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
- Logical disk is divided into strips



(a) RAID 0 (non-redundant)

RAID Level 1

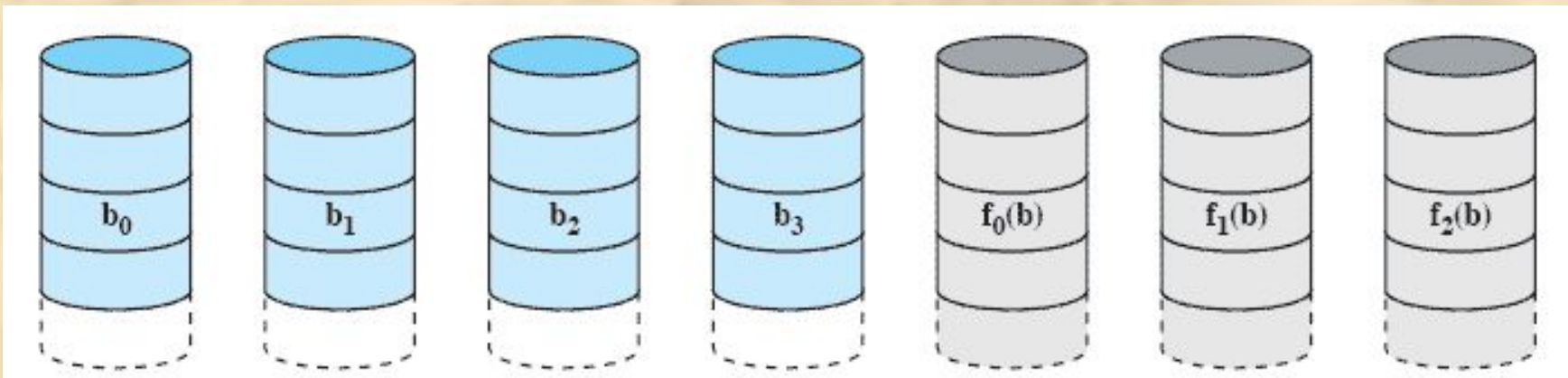
- Redundancy is achieved by the simple expedient of duplicating all the data
- There is no “write penalty”
- When a drive fails the data may still be accessed from the second drive
- Principal disadvantage is the cost



(b) RAID 1 (mirrored)

RAID Level 2

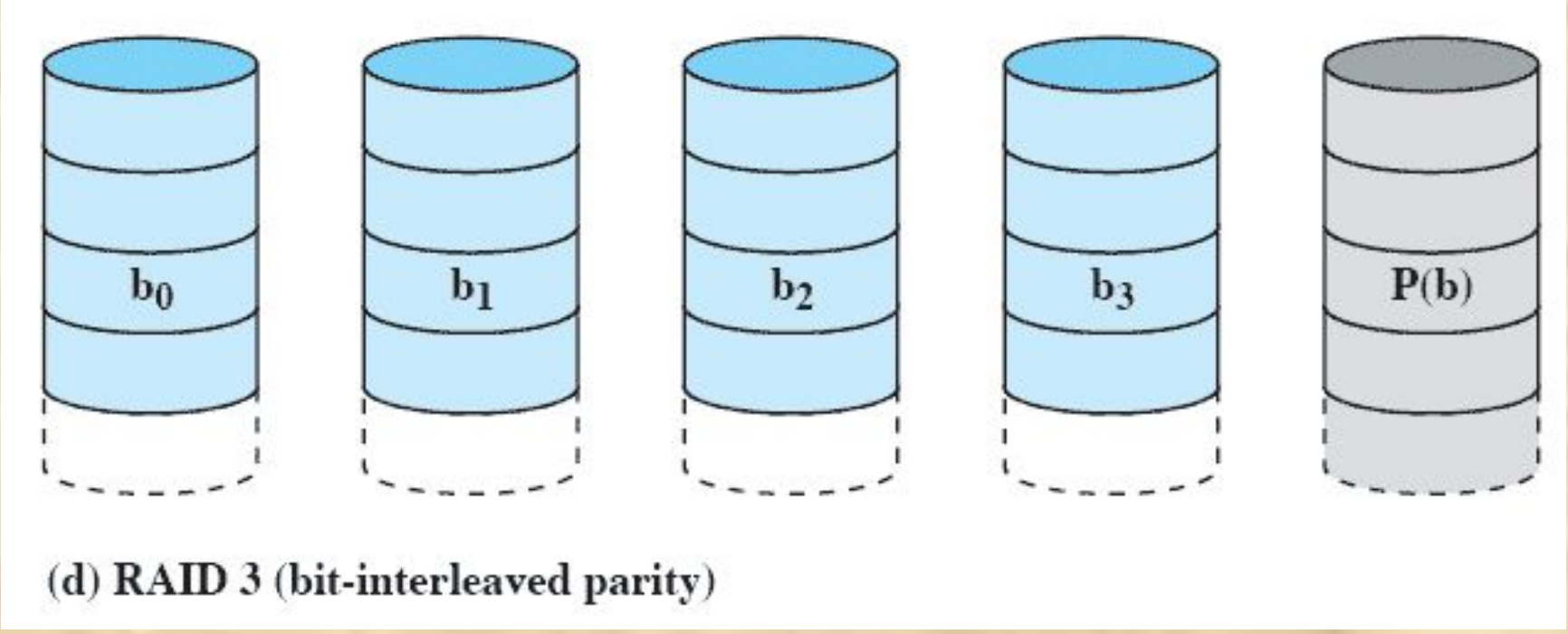
- Makes use of a parallel access technique
- Data striping is used
- Typically a Hamming code is used
- Effective choice in an environment in which many disk errors occur



(c) RAID 2 (redundancy through Hamming code)

RAID Level 3

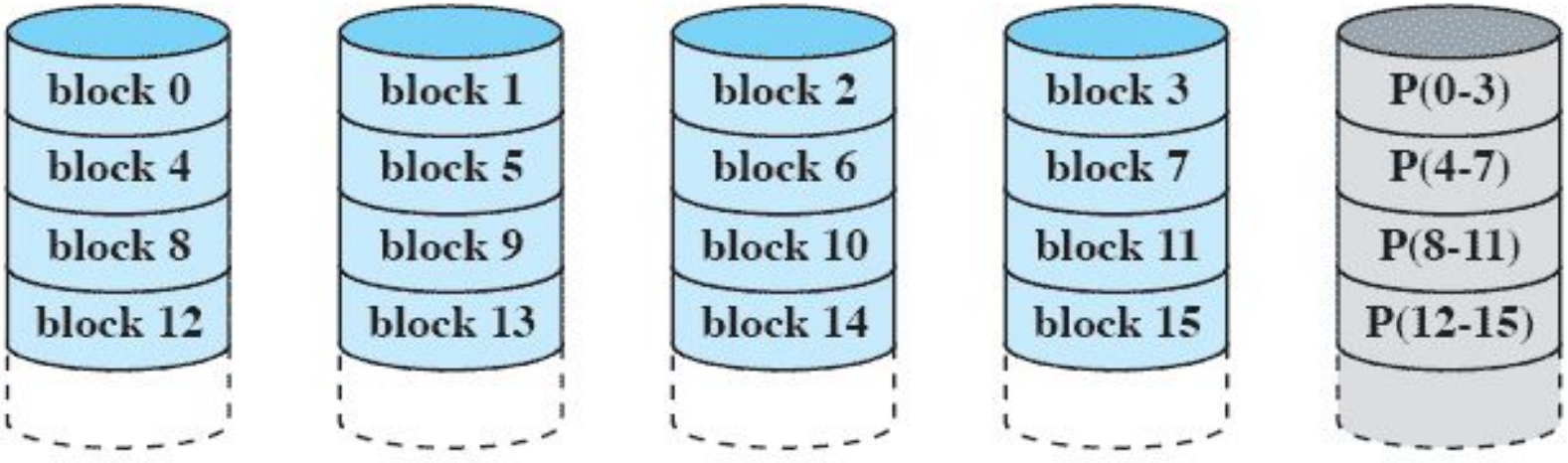
- Requires only a single redundant disk, no matter how large the disk array
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates



(d) RAID 3 (bit-interleaved parity)

RAID Level 4

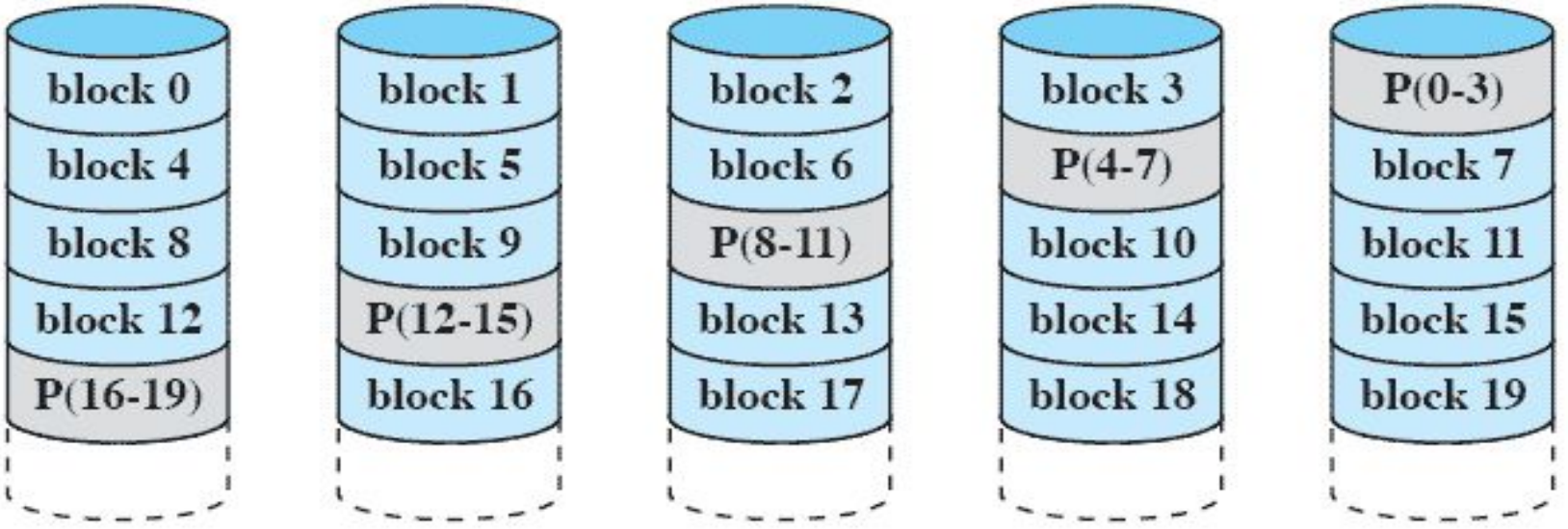
- Makes use of an independent access technique
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- Involves a write penalty when an I/O write request of small size is performed



(e) RAID 4 (block-level parity)

RAID Level 5

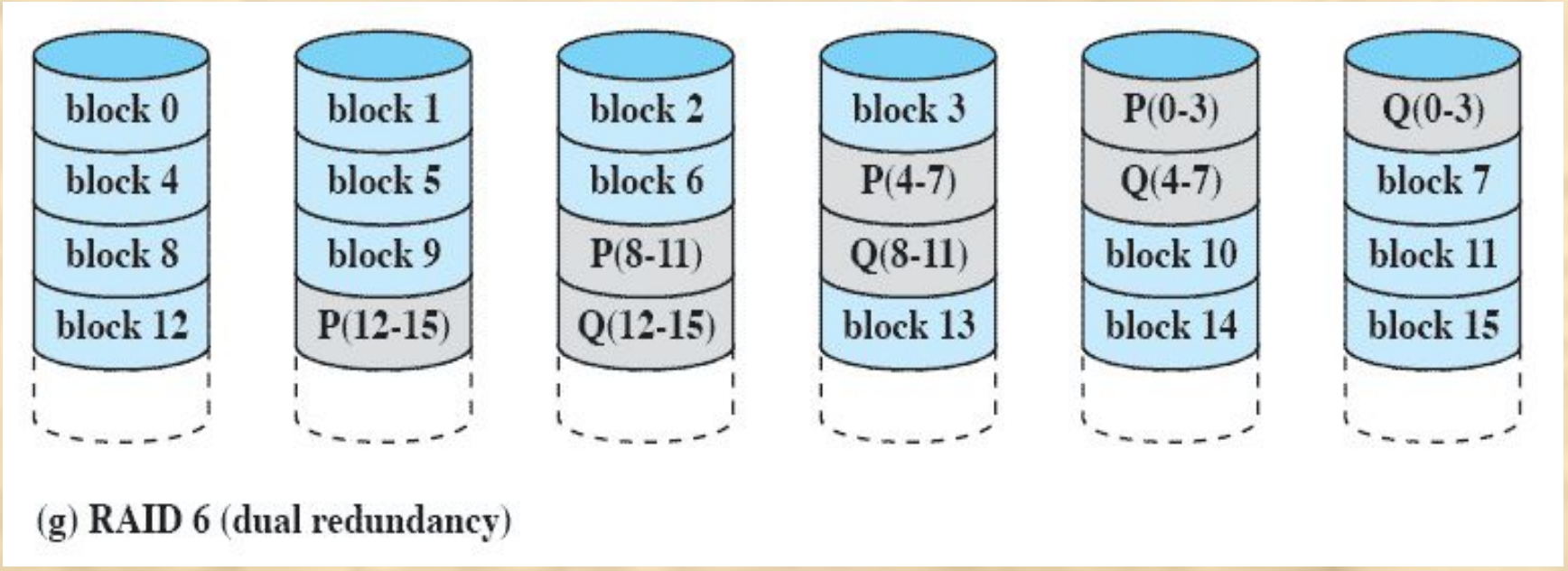
- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss



(f) RAID 5 (block-level distributed parity)

RAID Level 6

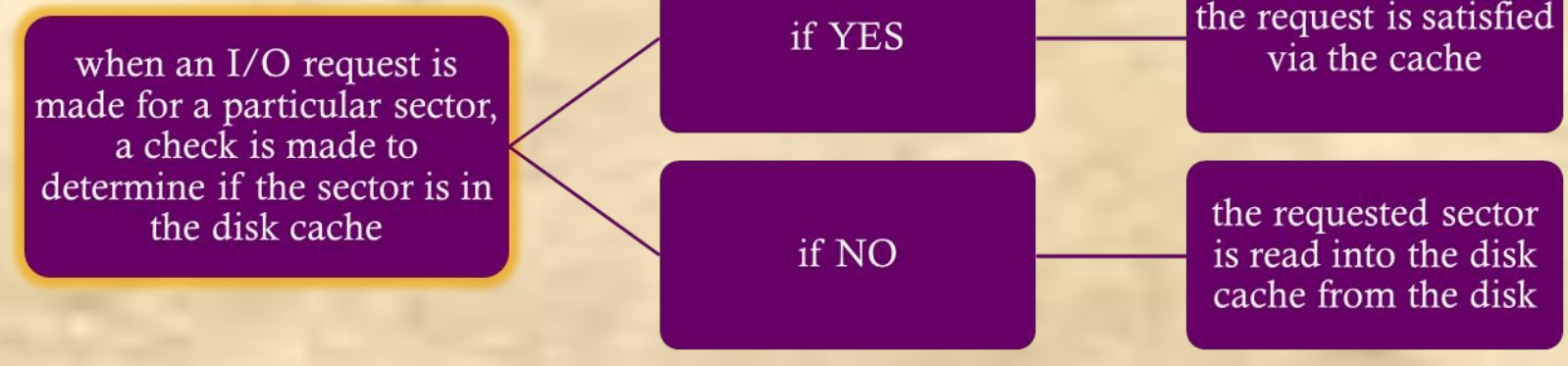
- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks



(g) RAID 6 (dual redundancy)

Disk Cache

- *Cache memory* is used to apply to a memory that is smaller and faster than main memory and that is interposed between main memory and the processor
- Reduces average memory access time by exploiting the principle of locality
- *Disk cache* is a buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk

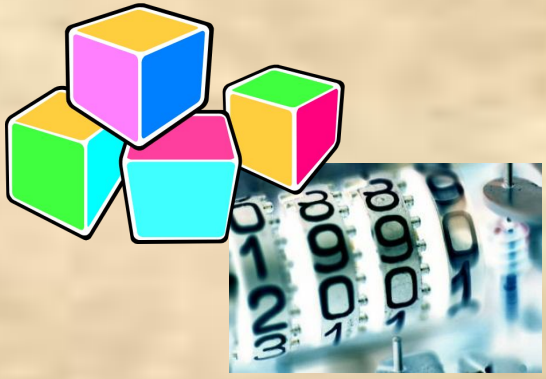


Least Recently Used (LRU)

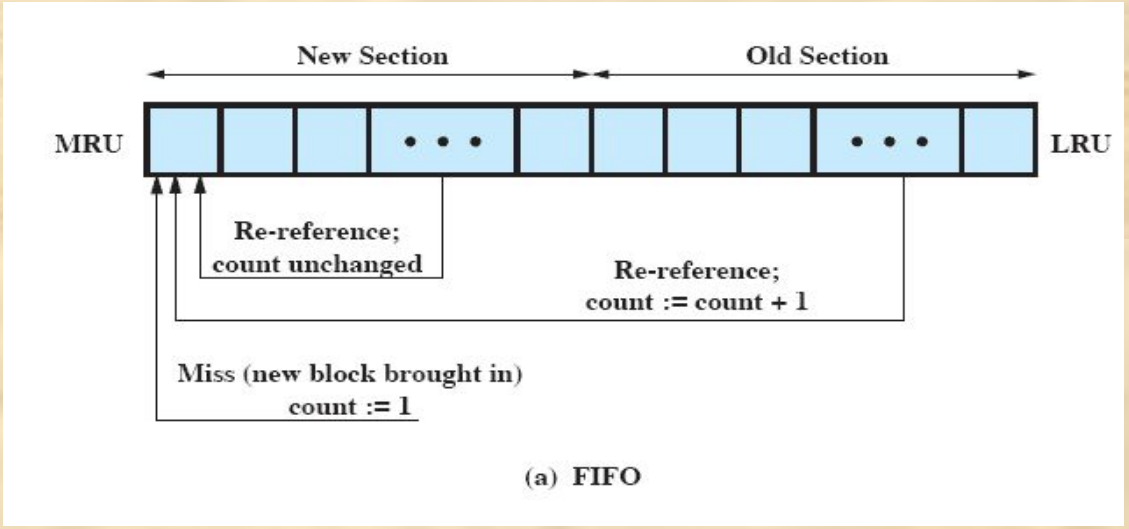
- Most commonly used algorithm that deals with the design issue of replacement strategy
- The block that has been in the cache the longest with no reference to it is replaced
- A stack of pointers reference the cache
  - most recently referenced block is on the top of the stack
  - when a block is referenced or brought into the cache, it is placed on the top of the stack

Least Frequently Used (LFU)

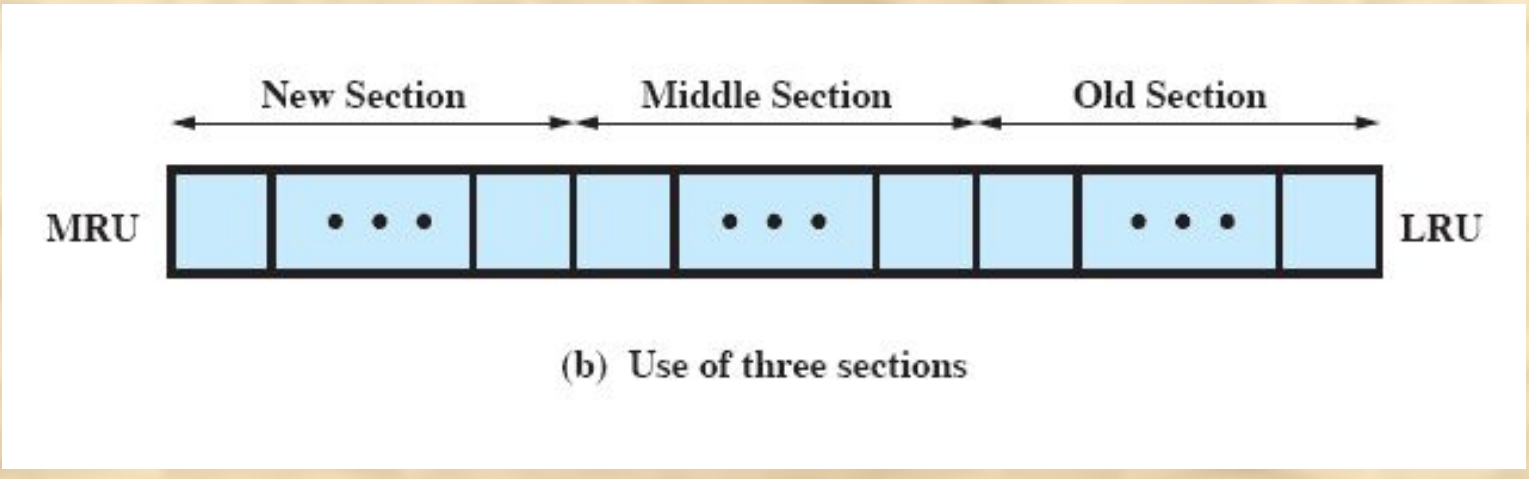
- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block is accessed
- When replacement is required, the block with the smallest count is selected



Frequency-Based Replacement



(a) FIFO



(b) Use of three sections

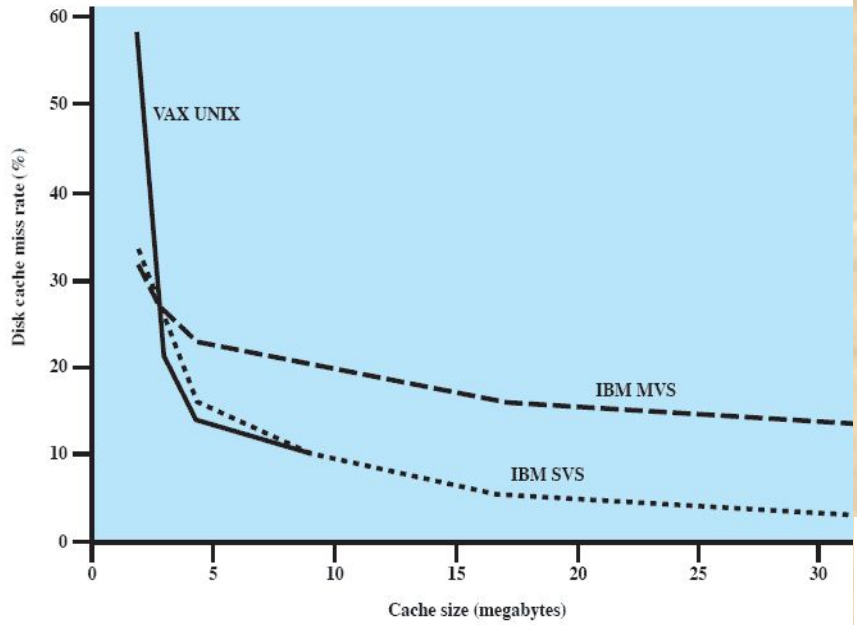


Figure 11.10 Some Disk Cache Performance Results Using LRU

LRU Disk Cache Performance

Frequency-Based Replacement

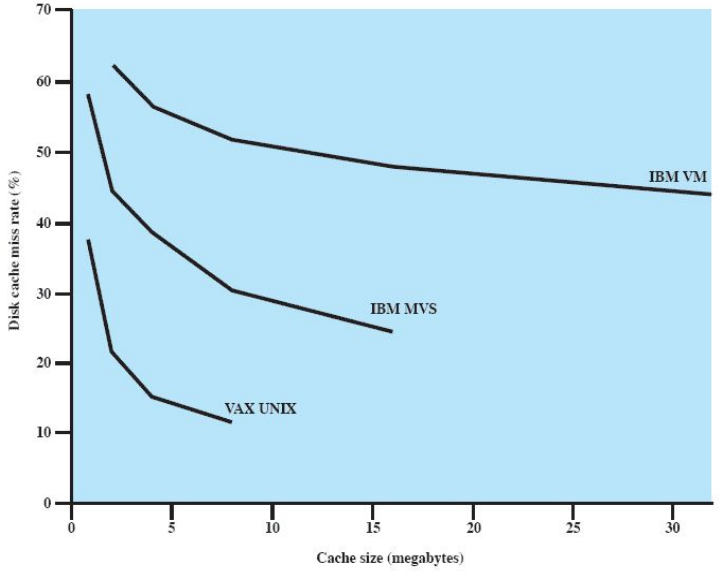


Figure 11.11 Disk Cache Performance Using Frequency-Based Replacement [ROBI90]

UNIX SVR4 I/O

- Two types of I/O
  - Buffered
    - system buffer caches
    - character queues
  - Unbuffered

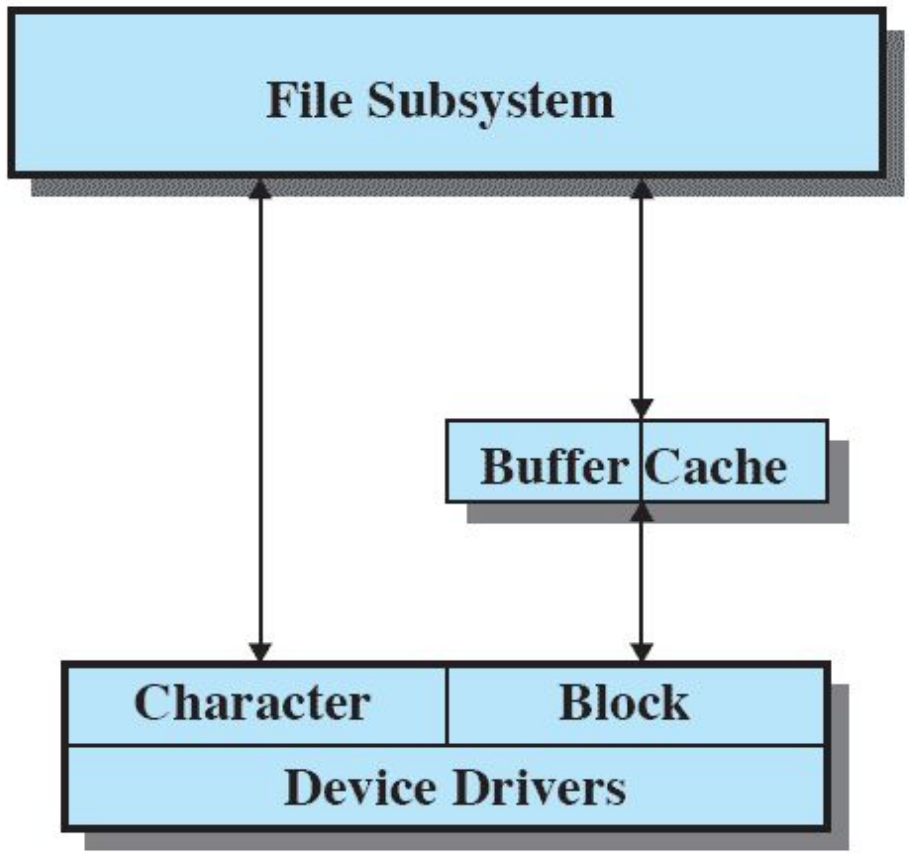


Figure 11.12 UNIX I/O Structure

Buffer Cache

- Three lists are maintained:
  - free list
  - device list
  - driver I/O queue

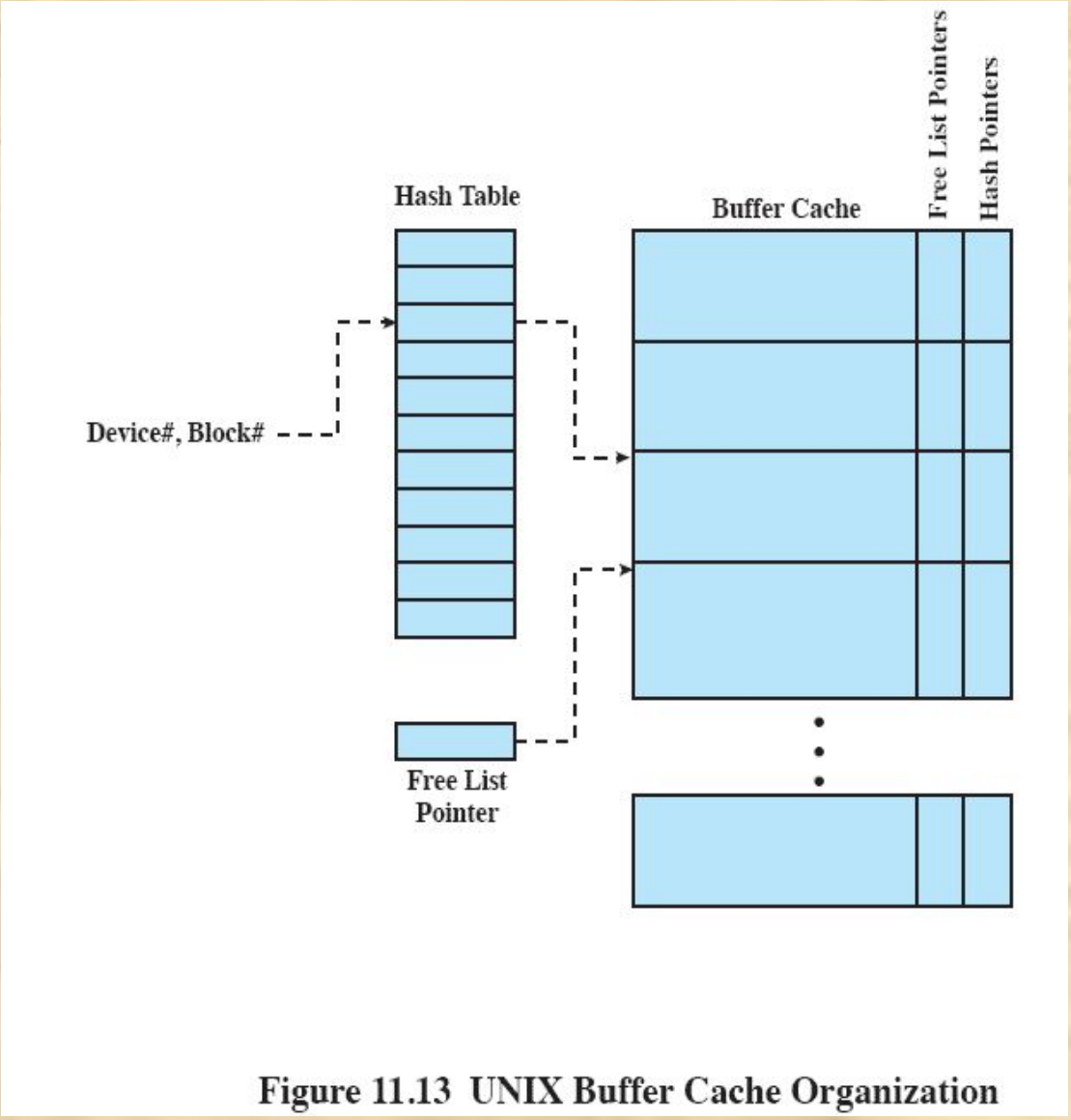


Figure 11.13 UNIX Buffer Cache Organization

Character Queue

Used by character oriented devices

terminals and printers

Either written by the I/O device and read by the process or vice versa

producer/consumer model is used

Character queues may only be read once

as each character is read, it is effectively destroyed

Unbuffered I/O

- Is simply DMA between device and process space
- Is always the fastest method for a process to perform I/O
- Process is locked in main memory and cannot be swapped out
- I/O device is tied up with the process for the duration of the transfer making it unavailable for other processes

Device I/O in UNIX

	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers	X		X

Linux I/O

- Very similar to other UNIX implementation
- Associates a special file with each I/O device driver
- Block, character, and network devices are recognized
- Default disk scheduler in Linux 2.4 is the Linux Elevator

For Linux 2.6 the Elevator algorithm has been augmented by two additional algorithms:

- the deadline I/O scheduler
- the anticipatory I/O scheduler

Deadline Scheduler

- Uses three queues:
  - incoming requests
  - read requests go to the tail of a FIFO queue
  - write requests go to the tail of a FIFO queue
- Each request has an expiration time

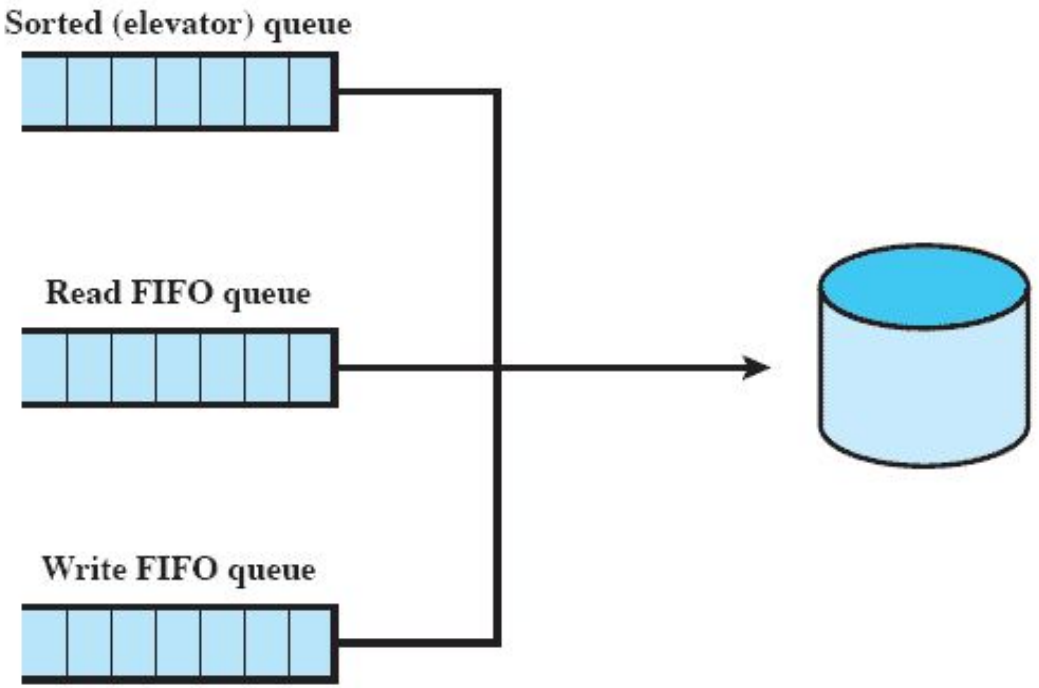


Figure 11.14 The Linux Deadline I/O Scheduler

Anticipatory I/O Scheduler

- Elevator and deadline scheduling can be counterproductive if there are numerous synchronous read requests
- Is superimposed on the deadline scheduler
- When a read request is dispatched, the anticipatory scheduler causes the scheduling system to delay
  - there is a good chance that the application that issued the last read request will issue another read request to the same region of the disk
    - that request will be serviced immediately
    - otherwise the scheduler resumes using the deadline scheduling algorithm

Linux Page Cache

- For Linux 2.4 and later there is a single unified page cache for all traffic between disk and main memory
- Benefits:
  - dirty pages can be collected and written out efficiently
  - pages in the page cache are likely to be referenced again due to temporal locality

Windows I/O Manager

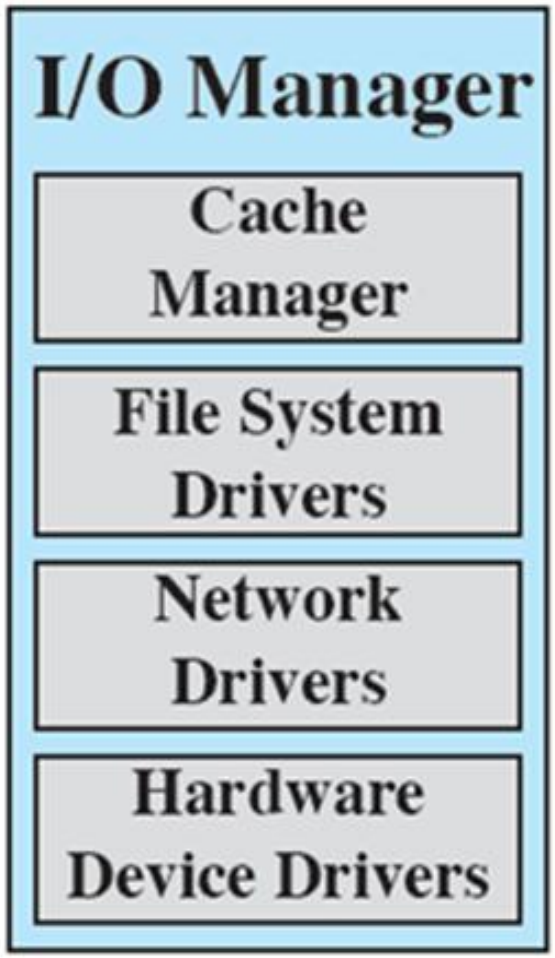
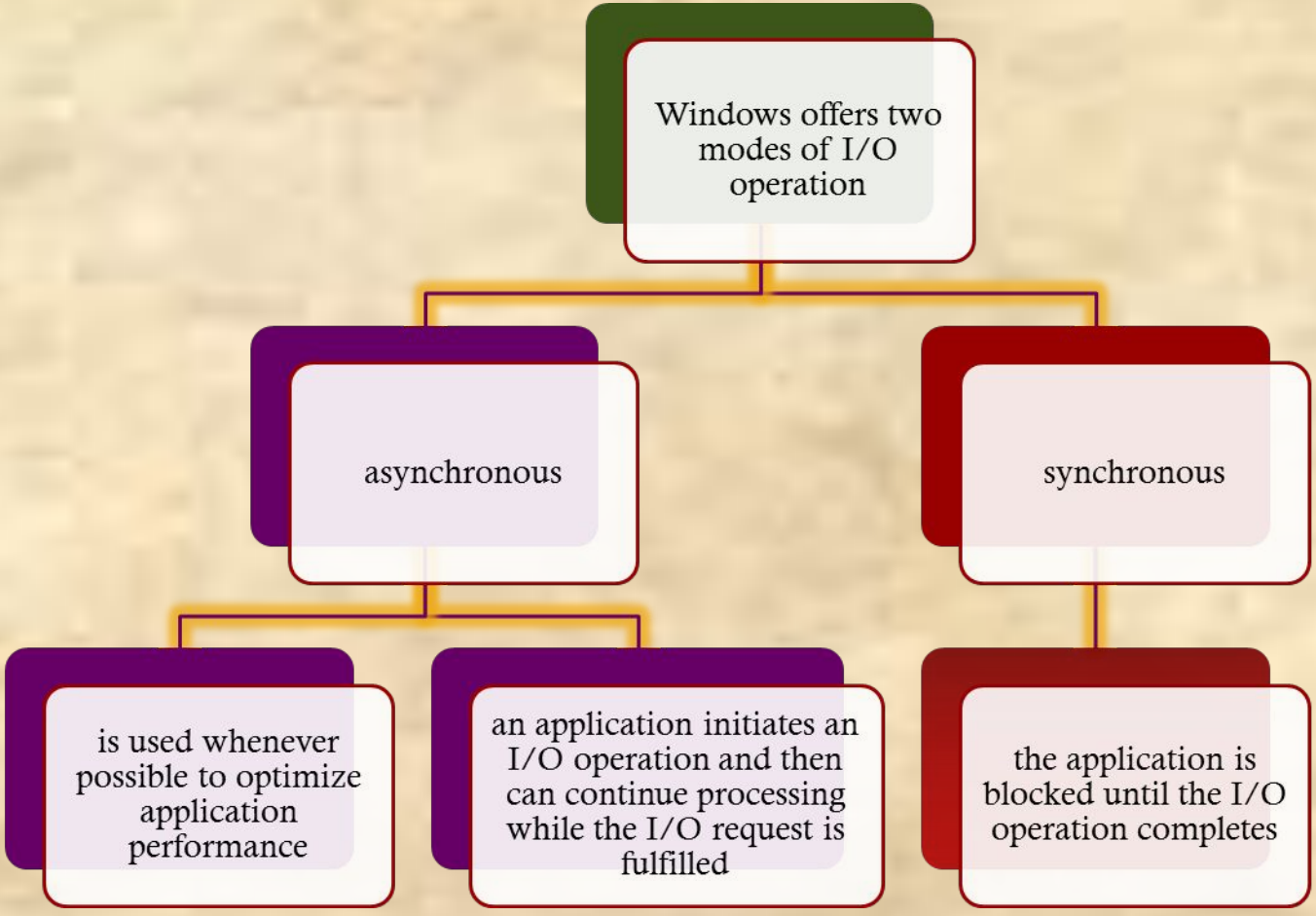


Figure 11.15 Windows I/O Manager

Basic I/O Facilities

- **Cache Manager**
  - maps regions of files into kernel virtual memory and then relies on the virtual memory manager to copy pages to and from the files on disk
- **File System Drivers**
  - sends I/O requests to the software drivers that manage the hardware device adapter
- **Network Drivers**
  - Windows includes integrated networking capabilities and support for remote file systems
  - the facilities are implemented as software drivers
- **Hardware Device Drivers**
  - the source code of Windows device drivers is portable across different processor types

Asynchronous and Synchronous I/O



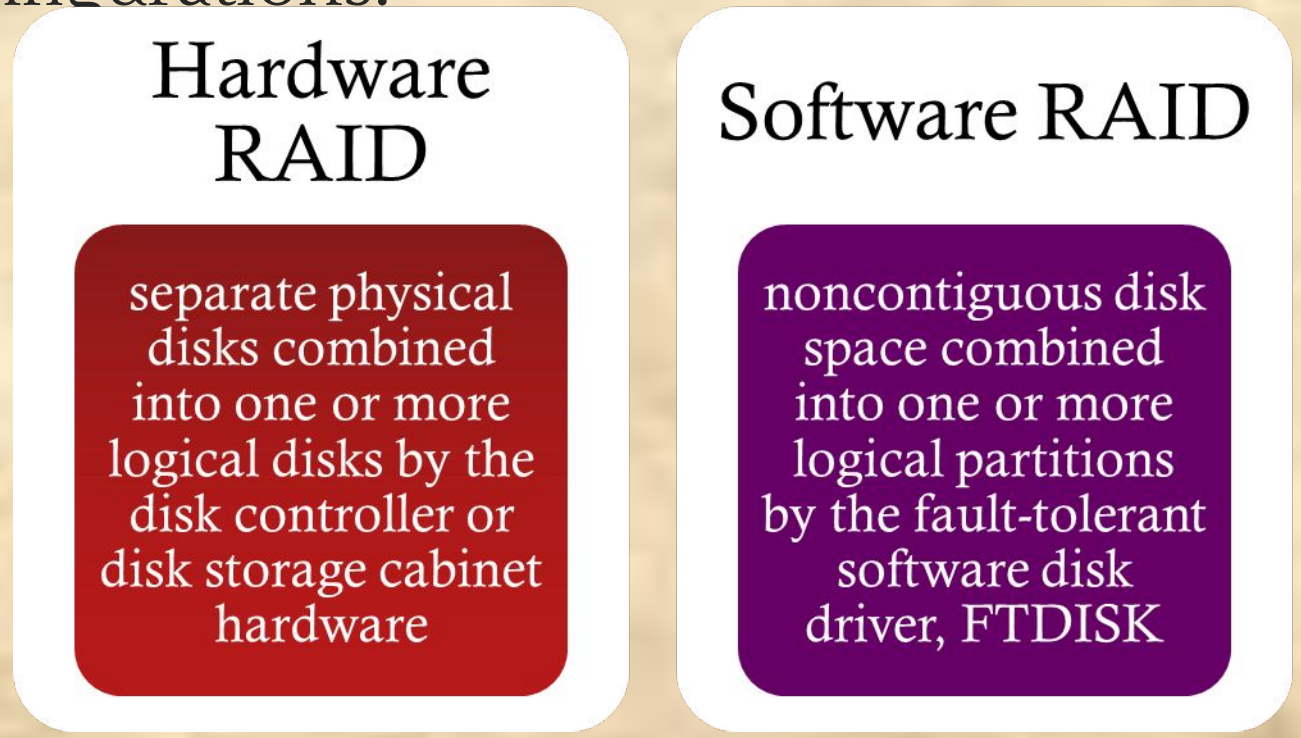
I/O Completion

- Windows provides five different techniques for signaling I/O completion:

1. Signaling the file object
2. Signaling an event object
3. Asynchronous procedure call
4. I/O completion ports
5. Polling

Windows RAID Configurations

- Windows supports two sorts of RAID configurations:



Volume Shadow Copies and Volume Encryption

- **Volume Shadow Copies**
  - efficient way of making consistent snapshots of volumes so they can be backed up
  - also useful for archiving files on a per-volume basis
  - implemented by a software driver that makes copies of data on the volume before it is overwritten

- **Volume Encryption**
  - Windows uses BitLocker to encrypt entire volumes
  - more secure than encrypting individual files
  - allows multiple interlocking layers of security

Summary

- I/O architecture is the computer system's interface to the outside world
- I/O functions are generally broken up into a number of layers
- A key aspect of I/O is the use of buffers that are controlled by I/O utilities rather than by application processes
- Buffering smoothes out the differences between the speeds
- The use of buffers also decouples the actual I/O transfer from the address space of the application process
- Disk I/O has the greatest impact on overall system performance
- Two of the most widely used approaches are disk scheduling and the disk cache
- A disk cache is a buffer, usually kept in main memory, that functions as a cache of disk block between disk memory and the rest of main memory

Disk Scheduling

Presented by:

Vaibhav Kumar Gupta

2004EE50416

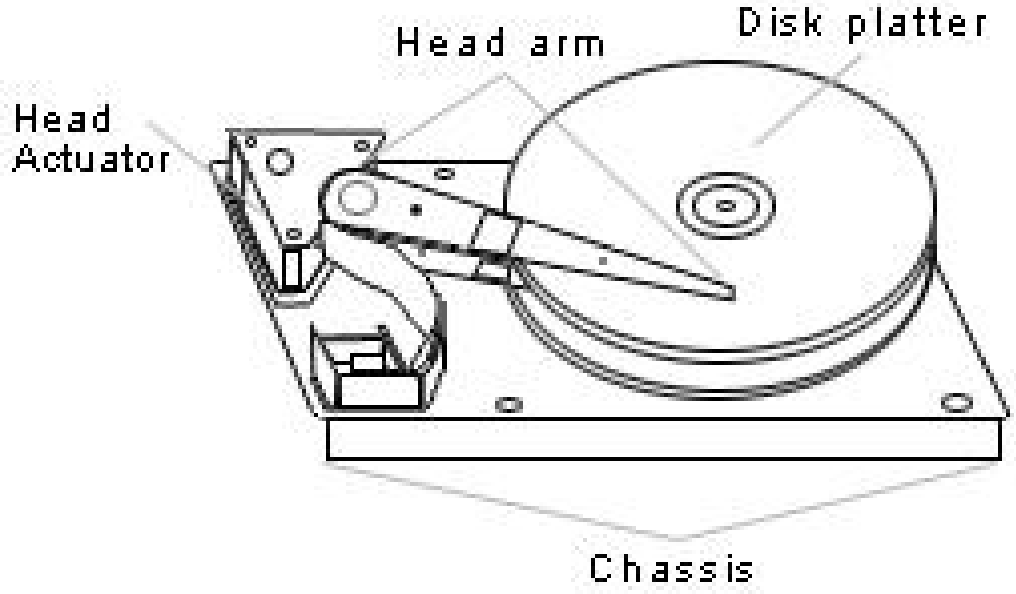
Overview

- Introduction
- Various Scheduling algorithms
  - FCFS
  - SSTF
  - SCAN Scheduling
  - C-SCAN Scheduling
  - LOOK Scheduling

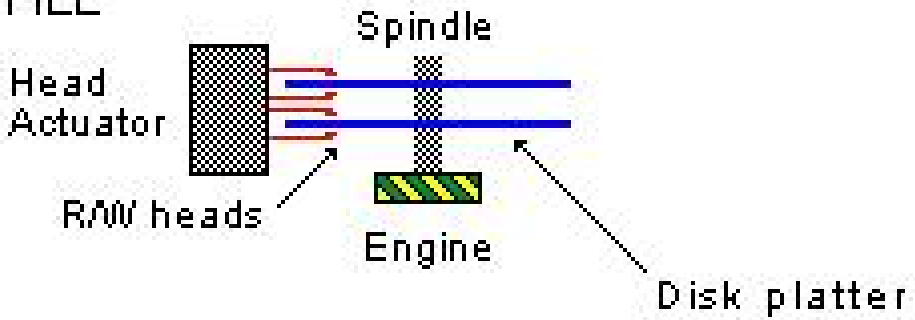
Disk Scheduling

- What is disk scheduling?
  - Servicing the disk I/O requests
- Why disk Scheduling?
  - Use hardware efficiently
- Includes
  - Fast access time (seek time+ rotational latency)
  - Large disk bandwidth

INSIDE DISK



PROFILE



Disc Scheduling

- I/O request issues a system call to the OS.
  - If desired disk drive or controller is available, request is served immediately.
  - If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the OS has to choose which pending request to service next.

FCFS Scheduling

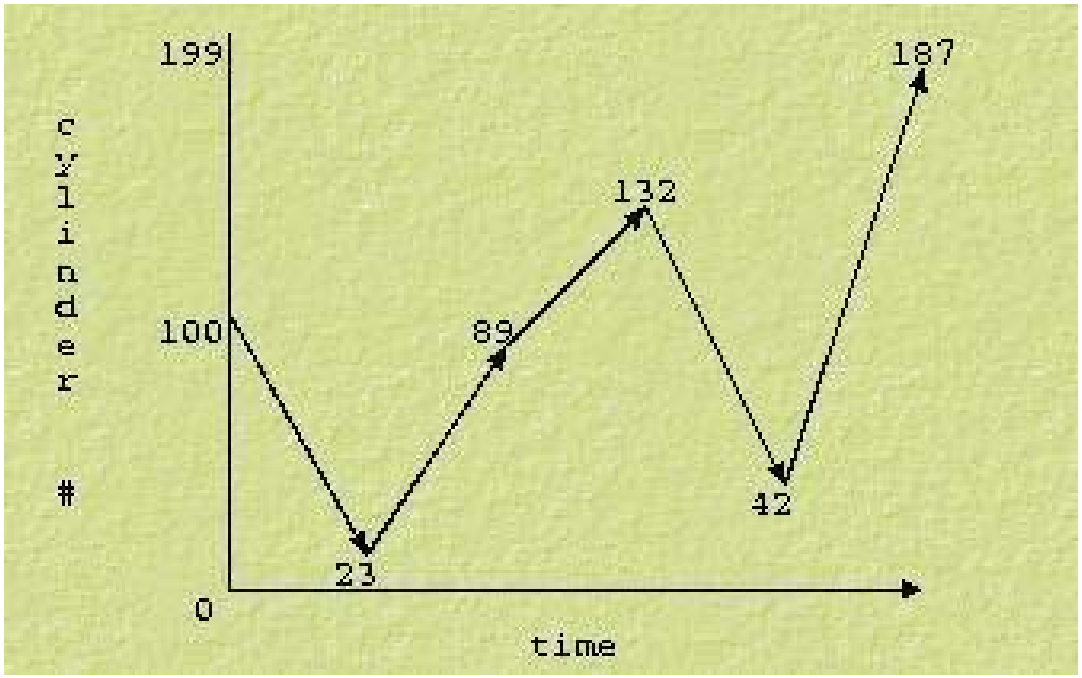
- Simplest, perform operations in order requested
- no reordering of work queue
- no **starvation**: every request is serviced
- Doesn't provide fastest service
- Ex: a disk queue with requests for I/O to blocks on cylinders

  23, 89, 132, 42, 187

  With disk head initially at 100

FCFS

23, 89, 132, 42, 187



77+66+43+90+145=421

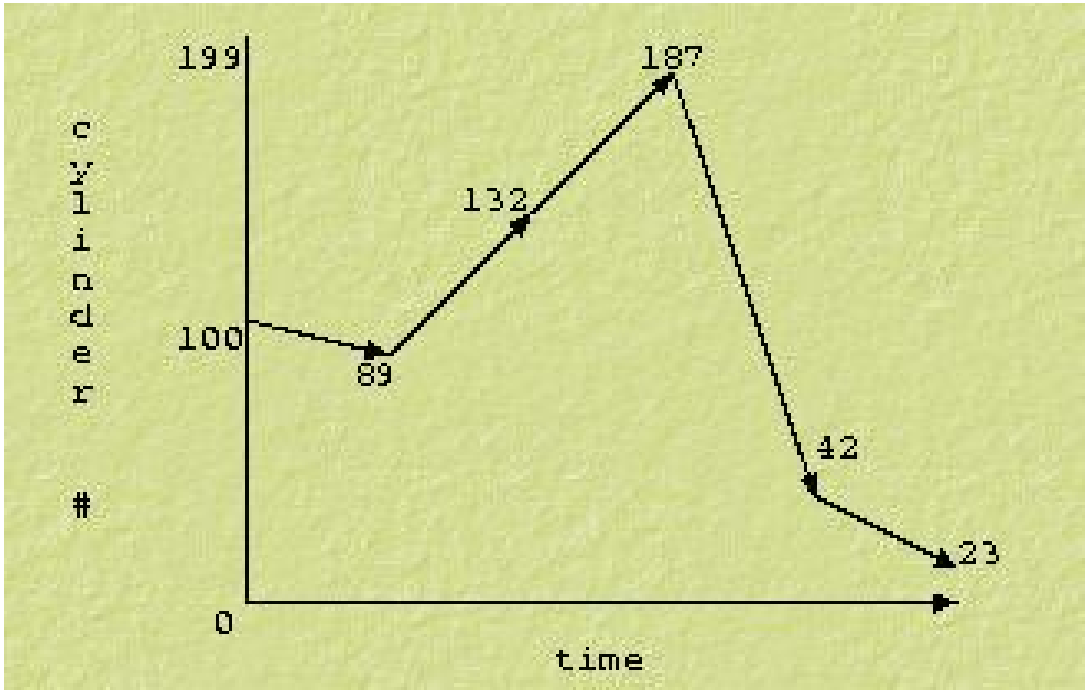
If the requests for cylinders 23 and 42 could be serviced together, total head movement could be decreased substantially.

SSTF Scheduling

- Like SJF, select the disk I/O request that requires the least movement of the disk arm from its current position, regardless of direction
- reduces total seek time compared to FCFS.
- Disadvantages
  - **starvation** is possible; stay in one area of the disk if very busy
  - switching directions slows things down
  - Not the most optimal

SSTF

23, 89, 132, 42, 187



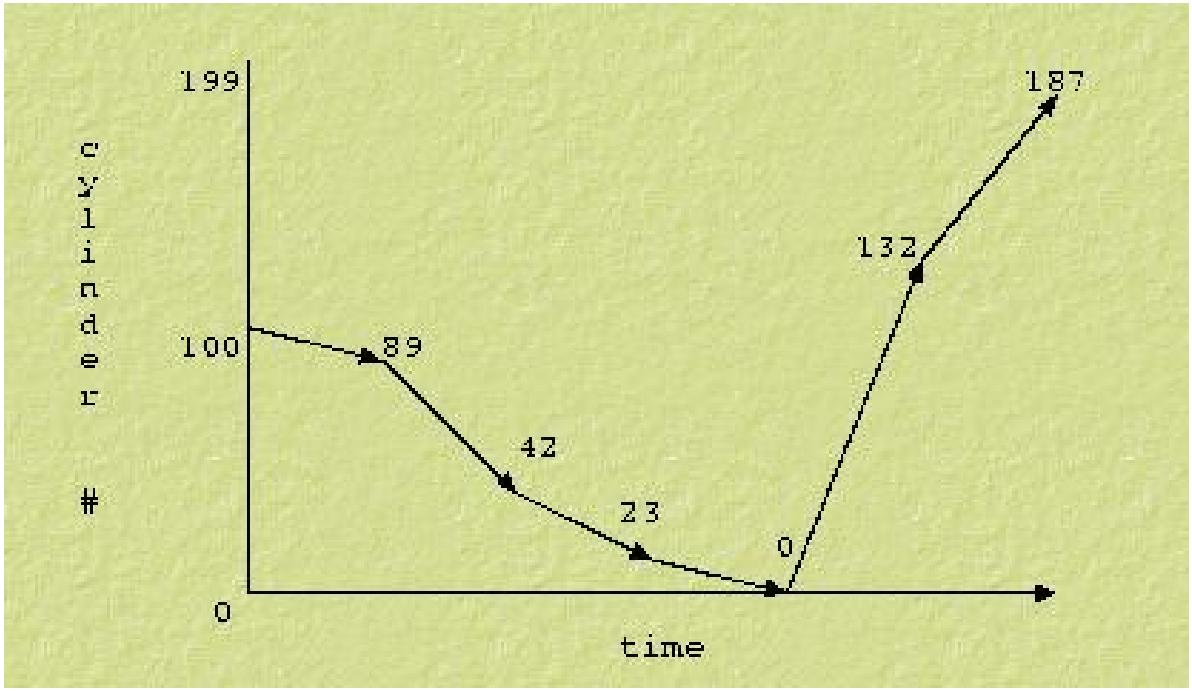
11+43+55+145+19=273

SCAN

- go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests.
- Sometimes called the elevator algorithm.
- Reduces variance compared to SSTF.
- If a request arrives in the queue
  - just in front of the head
  - Just behind

SCAN

23, 89, 132, 42, 187



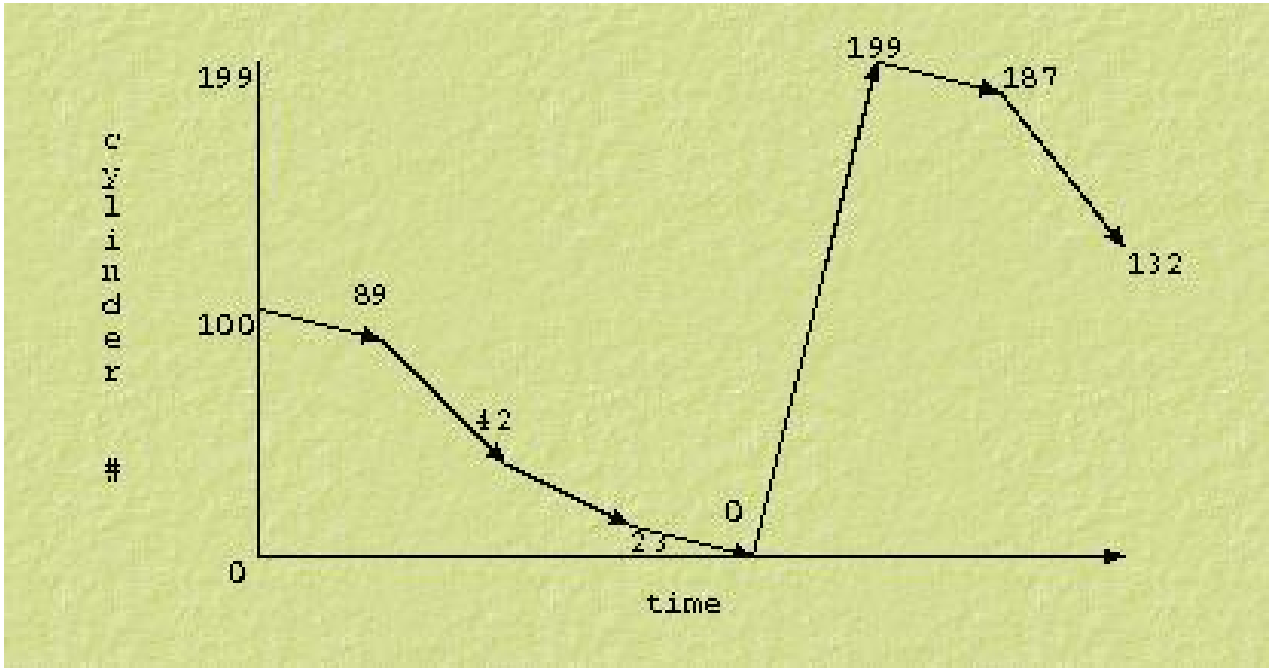
11+47+19+23+132+55=287

C-SCAN

- Circular SCAN
- moves inwards servicing requests until it reaches the innermost cylinder; then jumps to the outside cylinder of the disk without servicing any requests.
- Why C-SCAN?
  - Few requests are in front of the head, since these cylinders have recently been serviced. Hence provides a more uniform wait time.

C-SCAN

23, 89, 132, 42, 187



11+47+19+23+199+12+55=366

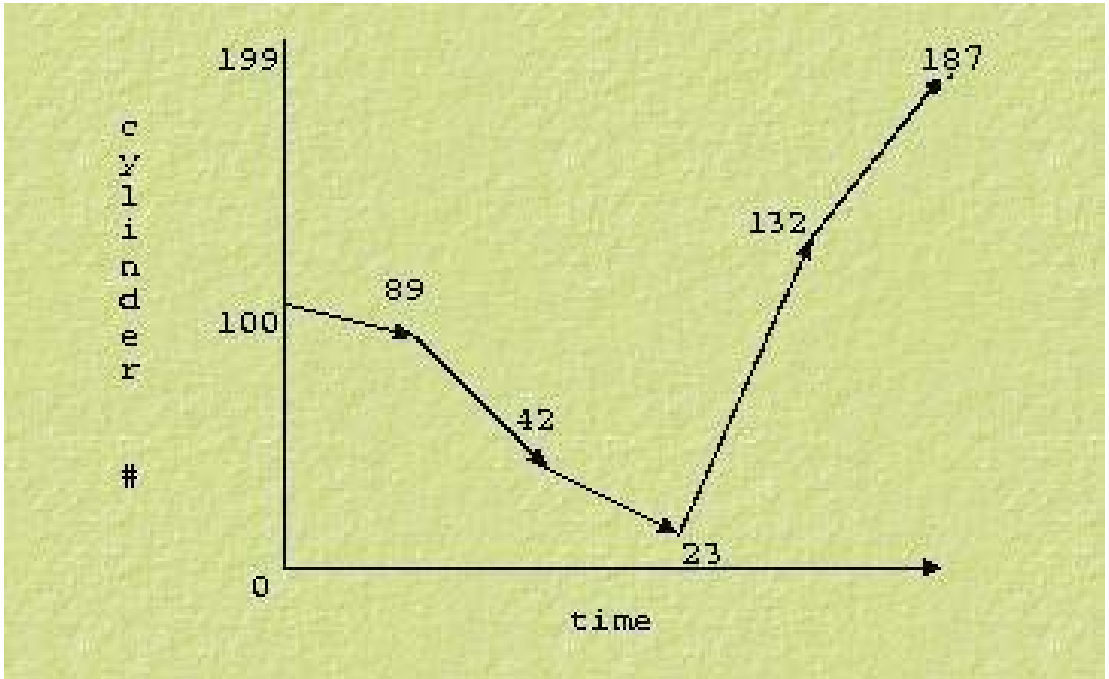
Head movement can be reduced if the request for cylinder 187 is serviced directly after request at 23 without going to the disk 0

LOOK

- like SCAN but stops moving inwards (or outwards) when no more requests in that direction exist

LOOK

23, 89, 132, 42, 187



11+47+19+109+55=241

Compared to SCAN, LOOK saves going from 23 to 0 and then back.

Most efficient for this sequence of requests

Which one to choose?

- Performance depends on number and type of requests.
- SSTF over FCFS.
- SCAN, C-SCAN for systems that place a heavy load on the disk, as they are less likely to cause starvation.
- Default algorithms, SSTF or LOOK

THANK YOU

REFERENCES:

Operating System Principles, Silberschatz, Galvin, Gagne
http://www.dmresearch.net/document/book/Introduction-to-Operating-Systems/notes/io/node8.html
http://hem.passagen.se/communication/ide.html