File System and Secondary Storage

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents (many types) is defined by file's creator
  - text file,
  - source file,
  - executable file

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on the device (disk)
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – information kept for creation time, last modification time, and last use time.
  - Useful for data for protection, security, and usage monitoring
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure (on disk), which consists of “inode” entries for each of the files in the system.

File Operations

- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for inode entry ***F_i***, and move the content of the entry to memory
- ***Close (F_i)*** – move the content of inode entry ***F_i*** in memory to directory structure on disk.

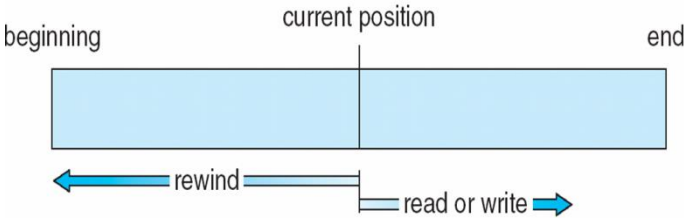
File Types – Name, Extension

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

Access Methods

Sequential Access

- General structure



- Operations:
  - **re**ad_**next ()** – reads the next portion of the file and automatically advances a file pointer.
  - **write_next () –** append to the end of the file and advances to the end of the newly written material (the new end of file).
  - r**eset** – back to the beginning of the file.

Access Methods

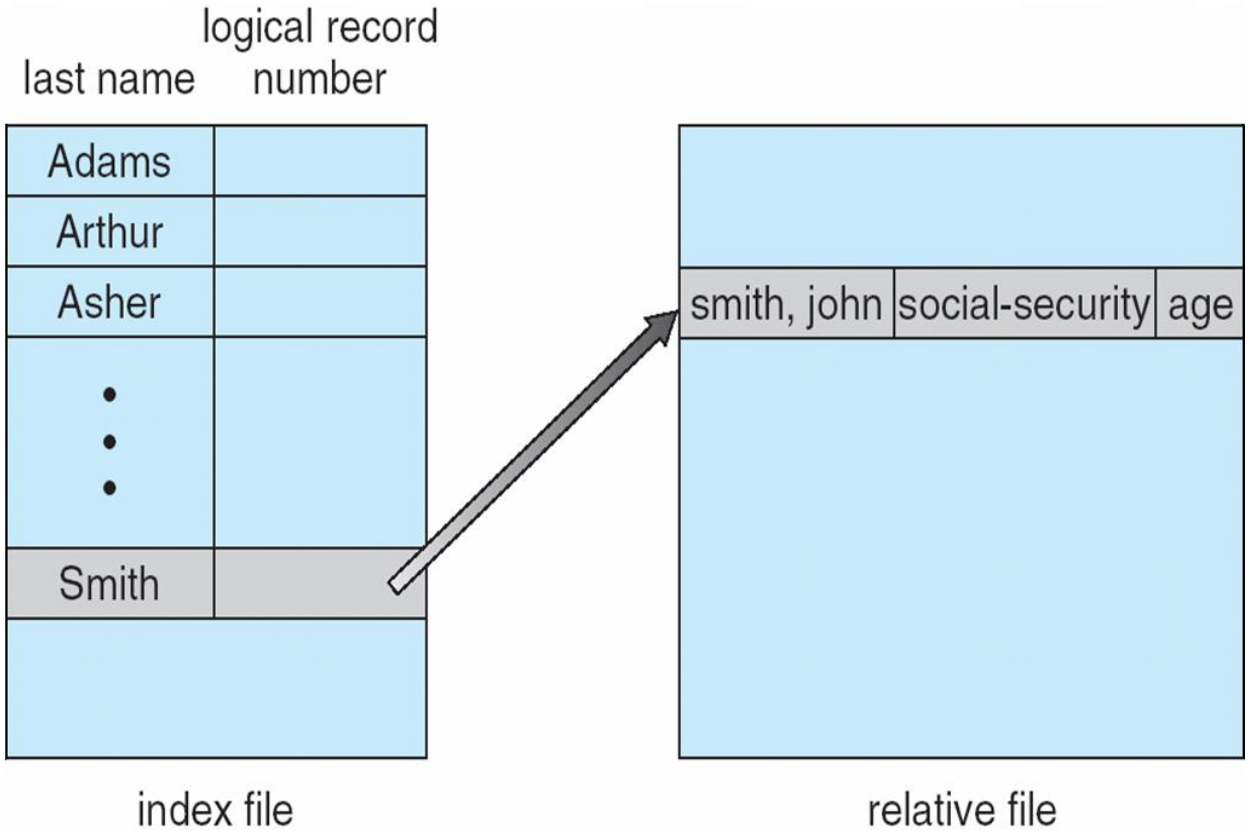
Direct Access

- File is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- File is viewed as a numbered sequence of blocks or records. For example, can read block 14, then read block 53, and then write block 7.
- Operations:
  - **read(n)** – reads relative block number n.
  - **write(n)** – writes relative block number n.
- Relative block numbers allow OS to decide where file should be placed

Other Access Methods

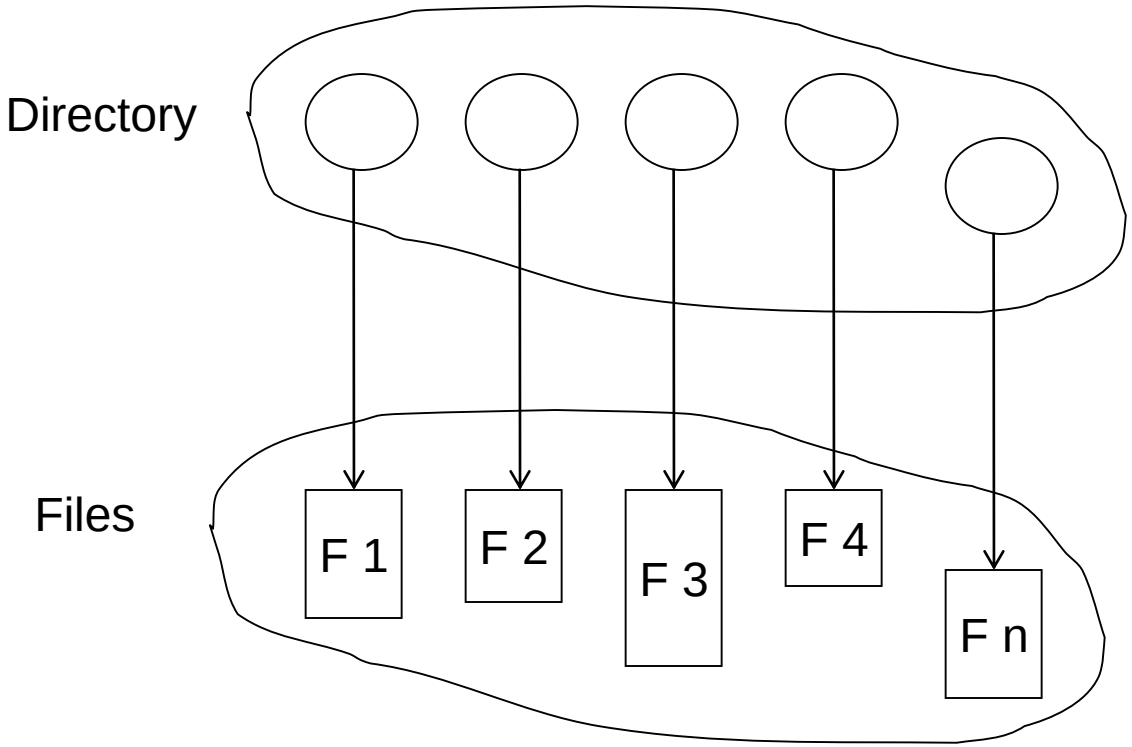
- Can be built on top of the base methods
- Generally -- involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, keep index (in memory) of the main index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example

Example of Index and Relative Files



Directory Structure

A collection of nodes containing information about all files



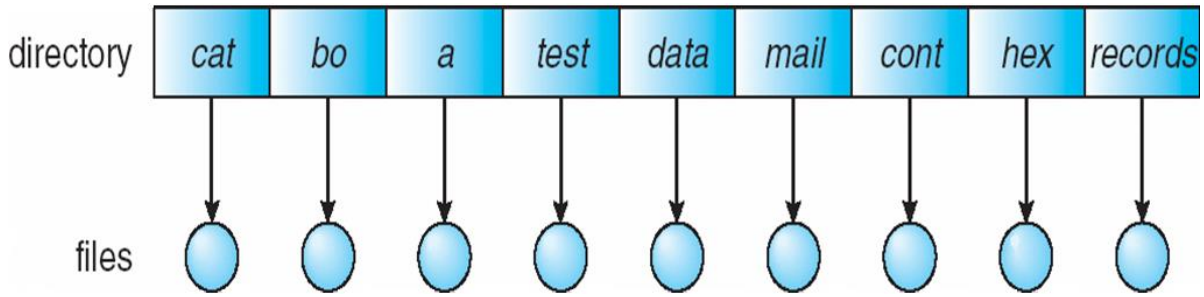
Operations performed on directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Both the directory structure and the files reside on disk

Single-Level Directory

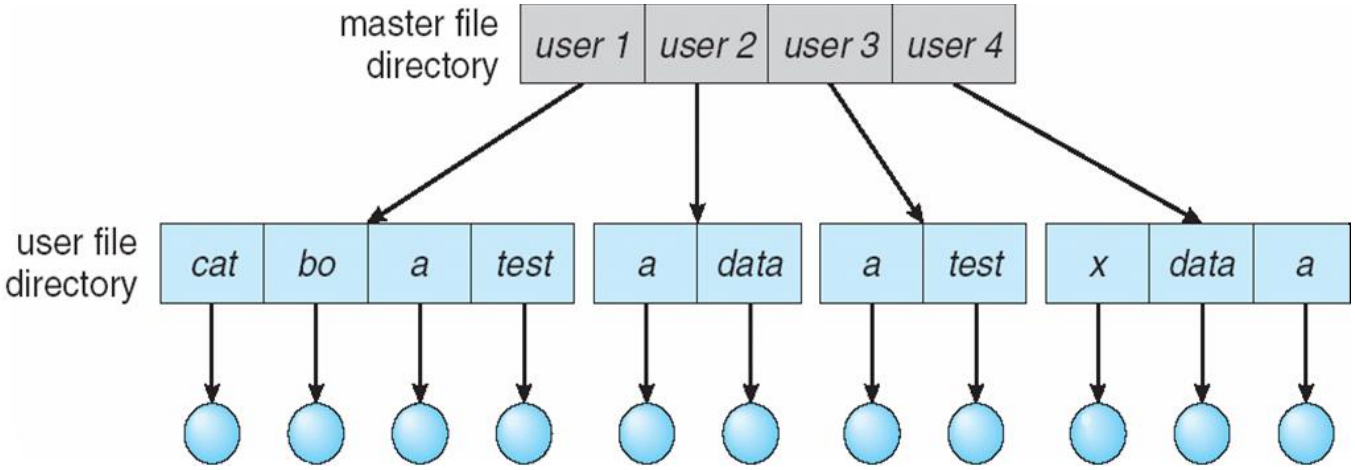
- A single directory for all users



- Naming problem
- Grouping problem

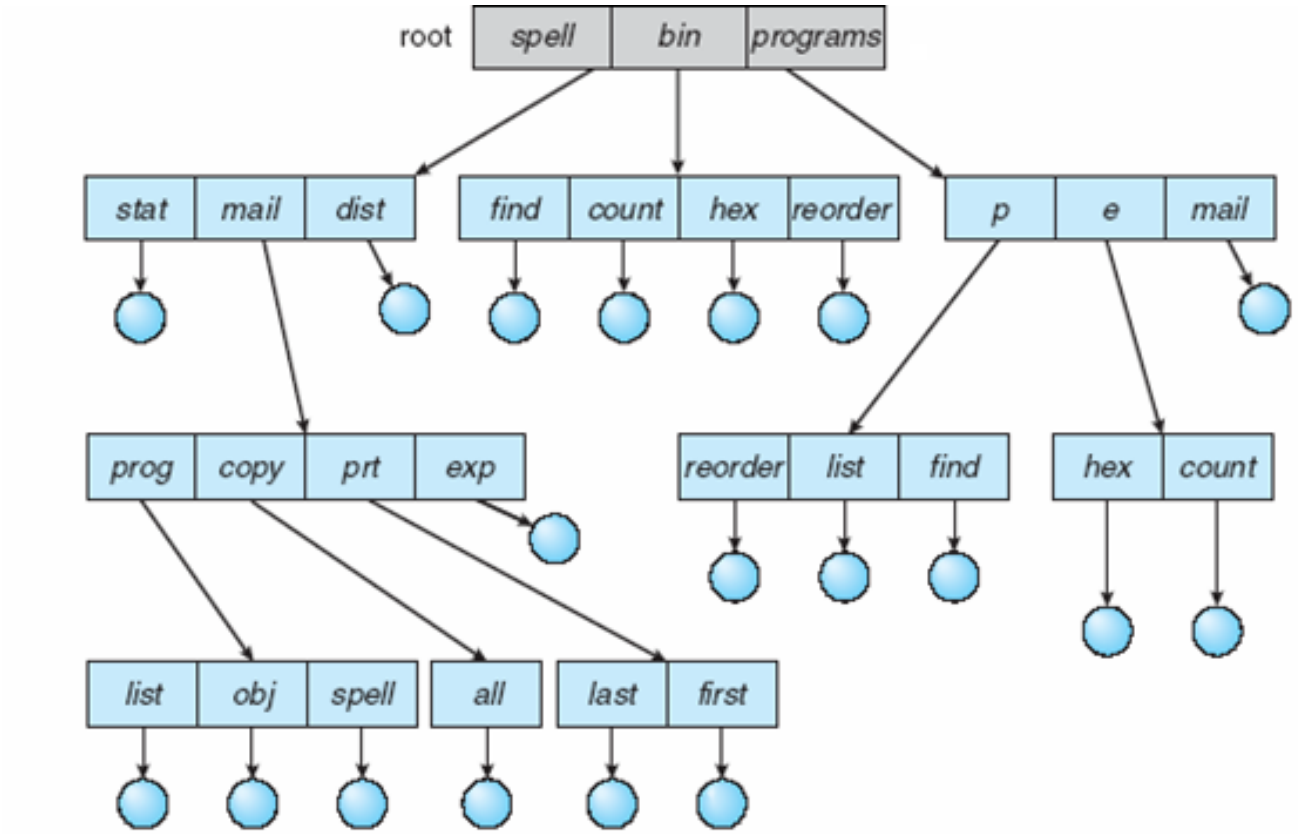
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

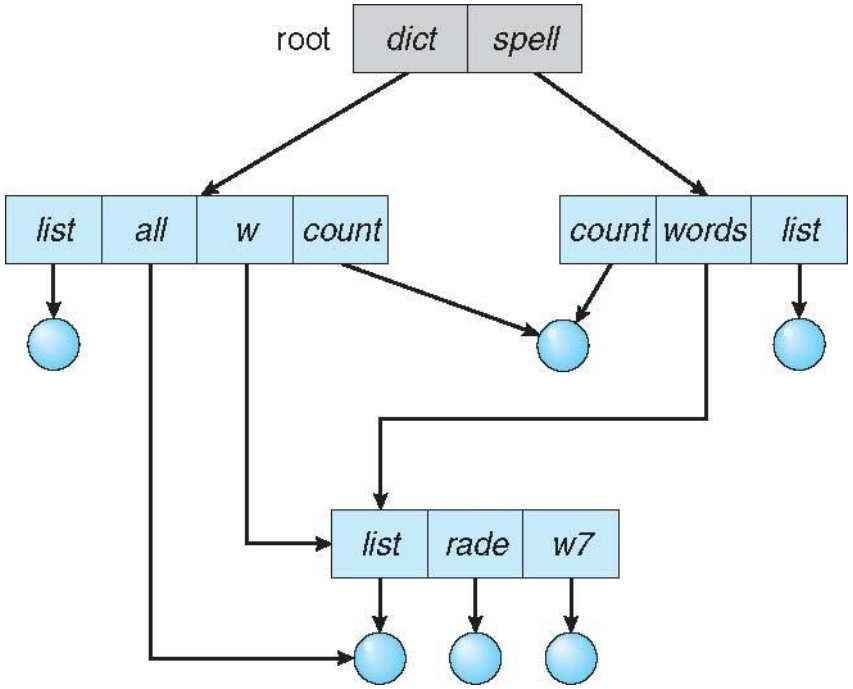
Tree-Structured Directories



- Efficient searching
- Grouping Capability

Acyclic-Graph Directories

Have shared subdirectories and files



- Files/subdirectories have two different names (aliasing)
  - Only one actual file exists, so any changes made by one person are immediately visible to the other.

Acyclic-Graph Directories (Cont.)

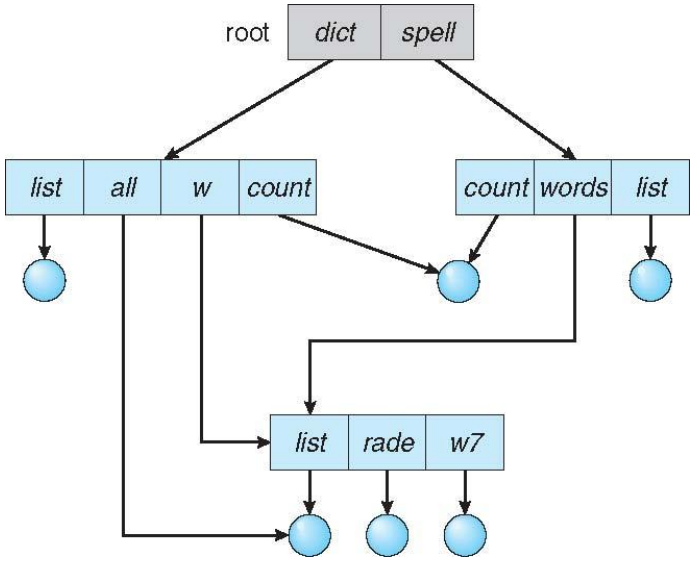
- Shared files and subdirectories can be implemented in several ways:
  - Can duplicate the *inode* information about the shared file/subdirectory in each of the subdirectories that “point’ to the shared structure.
    - If some information changes about the file it had to be updated in several places.
  - Create a new directory entry called a link.
    - **Link** – another name (pointer) to an existing file or subdirectory.
    - When a reference to a file is made, the directory is searched. If the directory entry is marked as a link, then the name of the real file is included in the link information.
    - We **resolve** the link by using that path name to locate the real file.
    - The operating system ignores these links when traversing directory trees to preserve the acyclic structure of the system.

Acyclic-Graph Directories -- Deletion

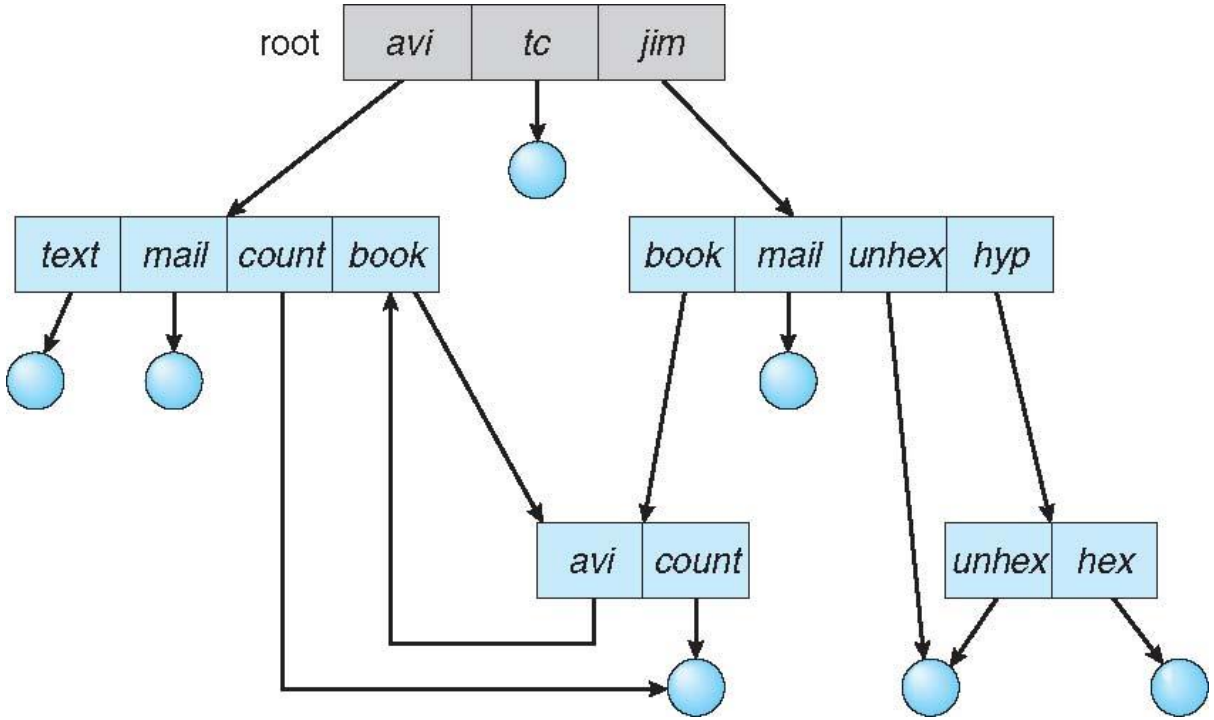
- If ***dict lcount*** is deleted ⇒ dangling pointer

  Solutions:

  - Backpointers -- so we can delete all pointers.
    - Variable size records a problem
  - Backpointers using a daisy-chain organization
  - Entry-hold-count solution



General Graph Directory



- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

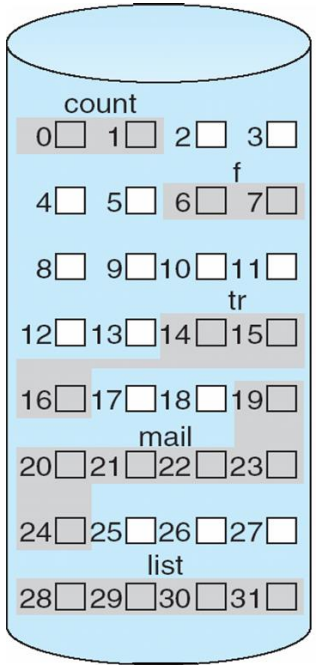
Disk Space Allocation Methods

- A disk is a direct-access device and thus gives us flexibility in the implementation of files.
- The main issue is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating disk space are in wide
  - Contiguous
  - Linked
  - Indexed

Contiguous Allocation

Each file occupies a set of contiguous blocks

- Best performance in most cases
- Simple – only starting location (block #) and length (number of blocks) are required
- Problems include:
  - Finding space for file,
  - Knowing the max file size,
  - External fragmentation,
    - Need for compaction



directory

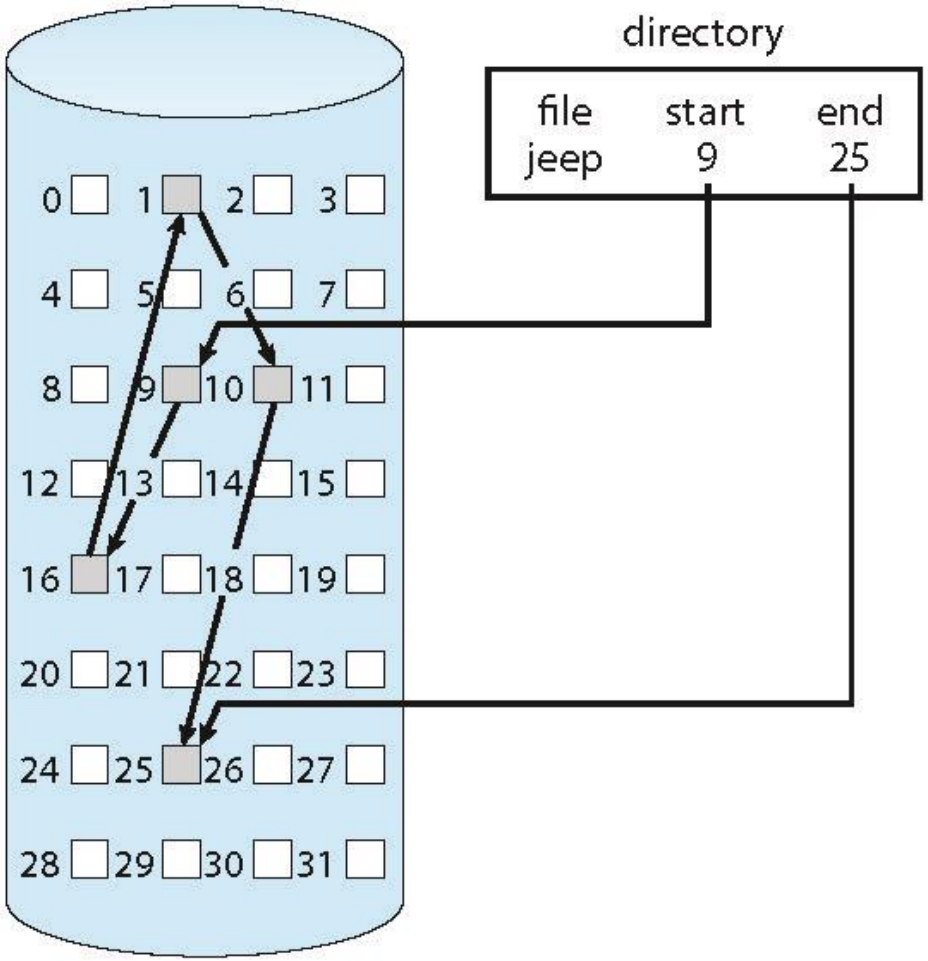
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked Allocation

Each file is a linked list of blocks

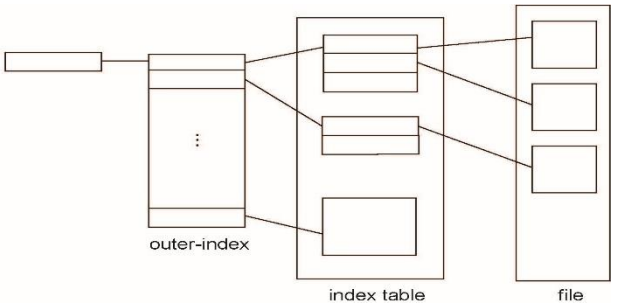
- No external fragmentation
- Each block contains pointer to next block
- Free space management system called when new block needed
- Locating a block can take many I/Os and disk seeks
- Reliability can be a problem (what if one of the pointers get corrupted?)
- Improve efficiency by clustering blocks into groups but increases internal fragmentation

Linked Allocation (Cont.)

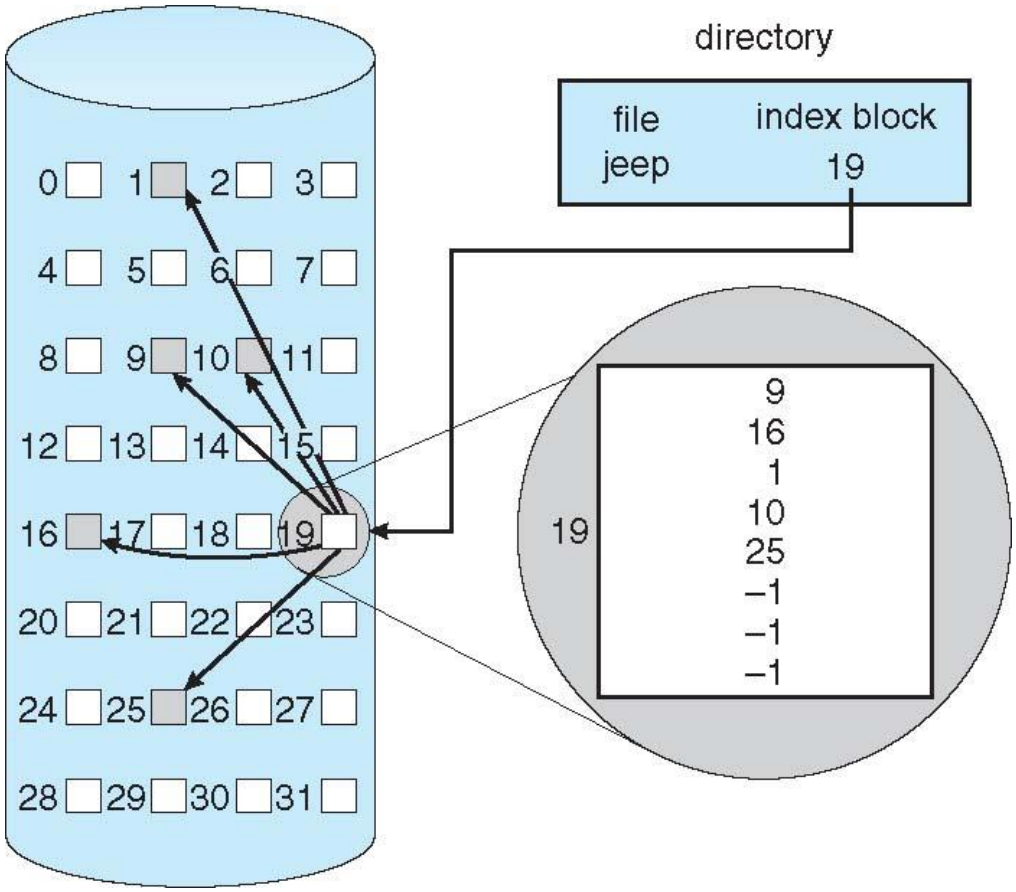


Indexed Allocation

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks
  - In linked allocation, pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order.
    - Indexed allocation solves this problem by bringing all the pointers together into one location: the index block.
- Need an index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Can be accomplished with 3 differe
  - Linked scheme.
  - Multilevel index
  - Combined scheme.



Example of Indexed Allocation

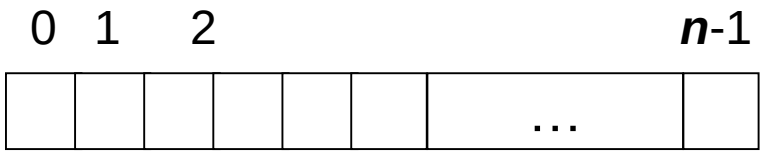


Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term “block” for simplicity)
- Four different methods:
  - Bit map
  - Free list
  - Grouping
  - Counting

Free-Space Management – bit map

- **Bit vector** (or **bit map**) (***n*** blocks)



bit[***i***] = {
- 1 ⇒ block[***i***] free
- 0 ⇒ block[***i***] occupied

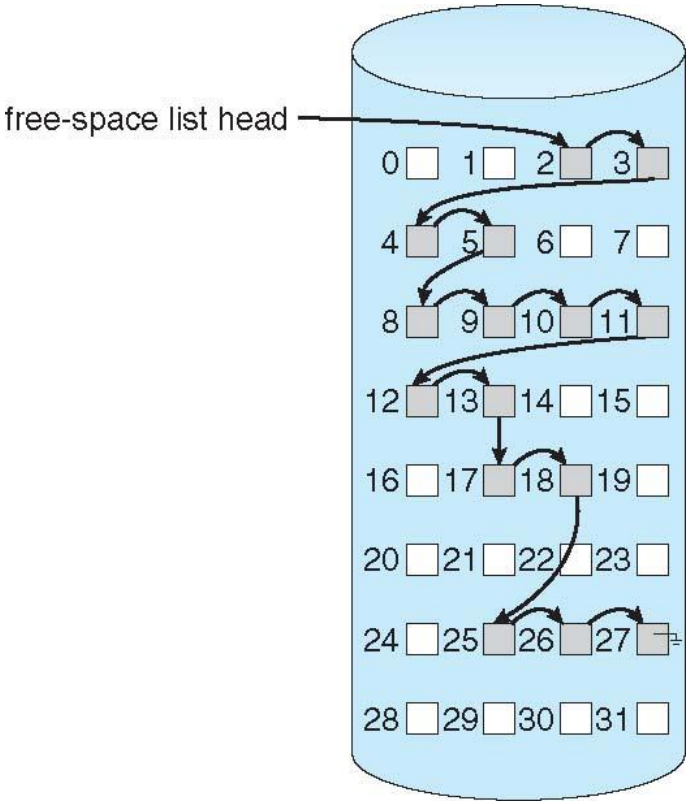
Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

CPUs have instructions to return offset within word of first “1” bit

Free List

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Grouping and Counting

- Grouping
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

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
- Minimize seek time
- Seek time ≈ seek distance
  - Seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- Rotational latency
  - It is the additional time waiting for the disk arm to rotate the desired sector to the disk head.
- Disk **bandwidth**
  - It is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must be queued.
  - Optimization algorithms only make sense when a queue exists

Disk Scheduling (Cont.)

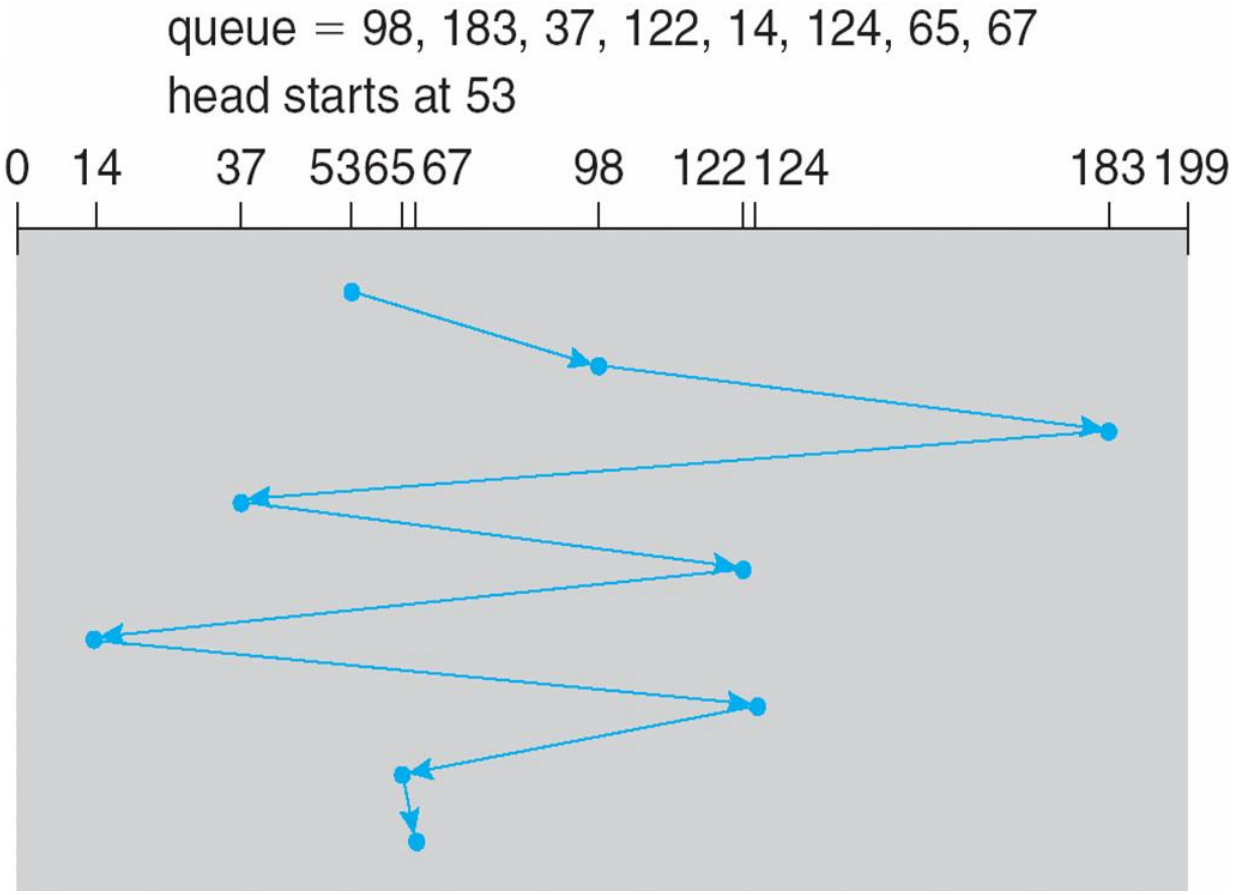
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

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

Head pointer 53

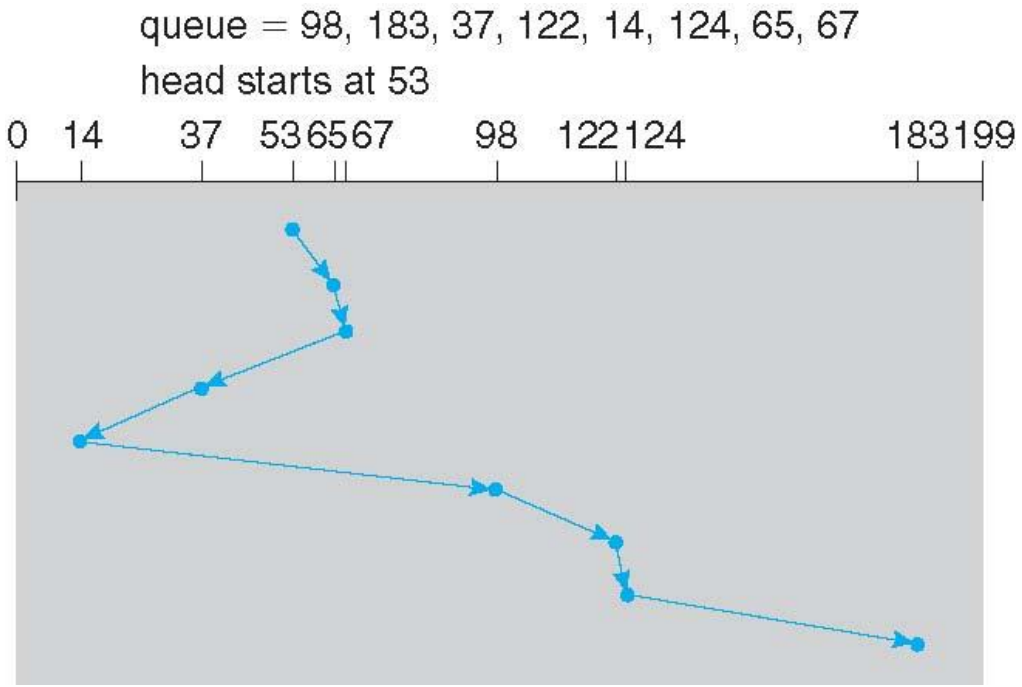
FCFS

Illustration shows total head movement of 640 cylinders



SSTF

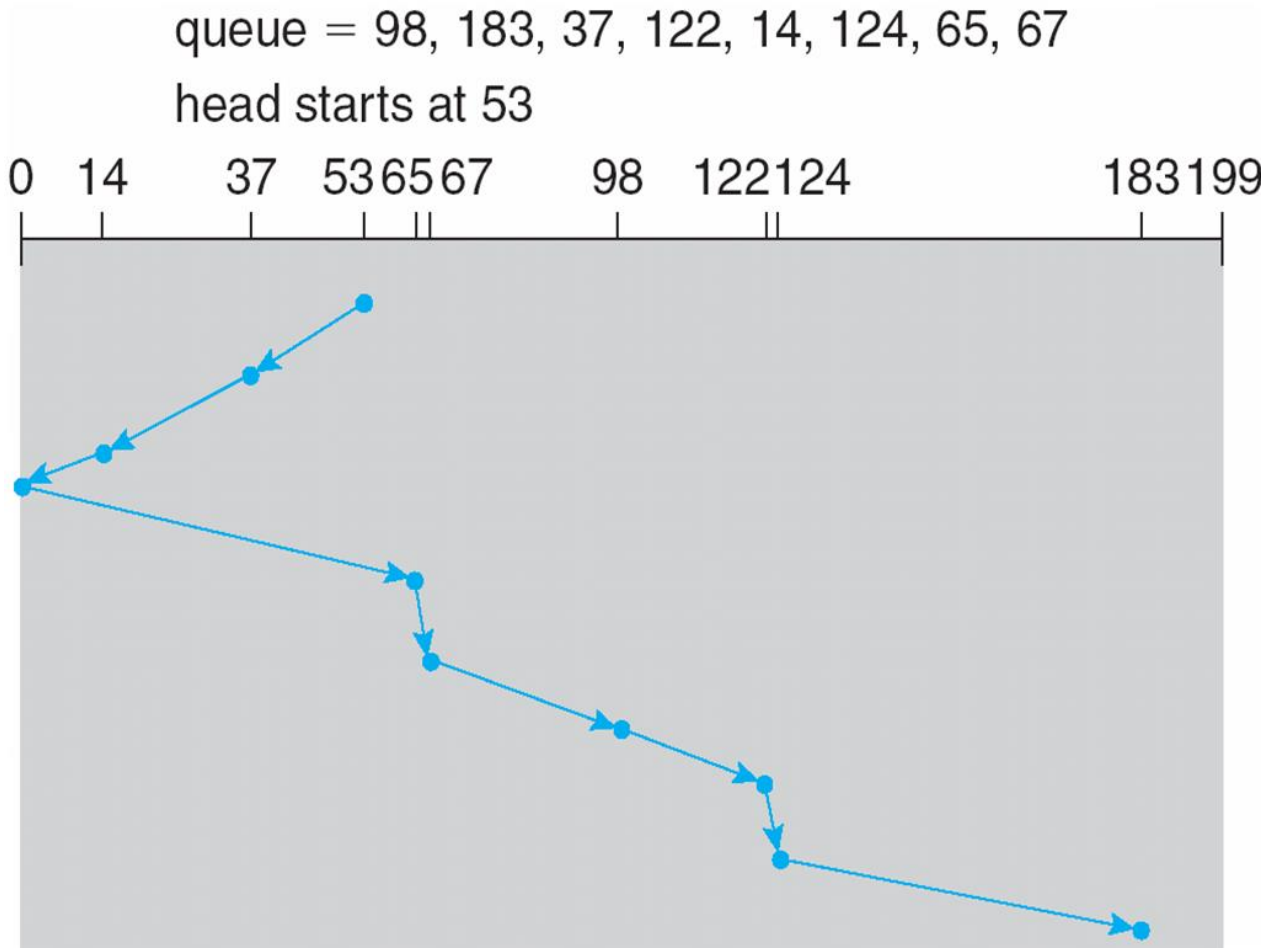
- Shortest Seek Time First -- selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders



SCAN

- **SCAN algorithm**. 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 it is called the **elevator algorithm**
- Illustration shows total head movement of 208 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

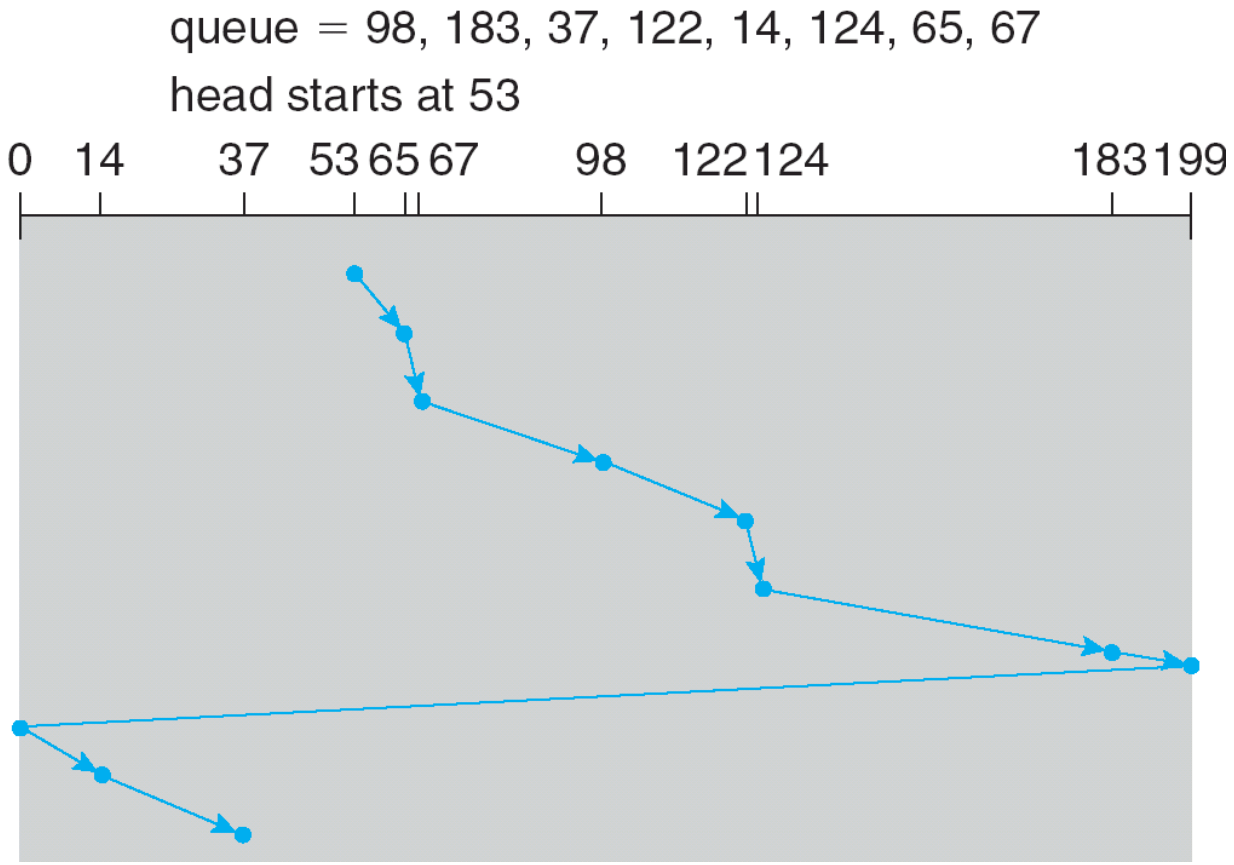
SCAN (Cont.)


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

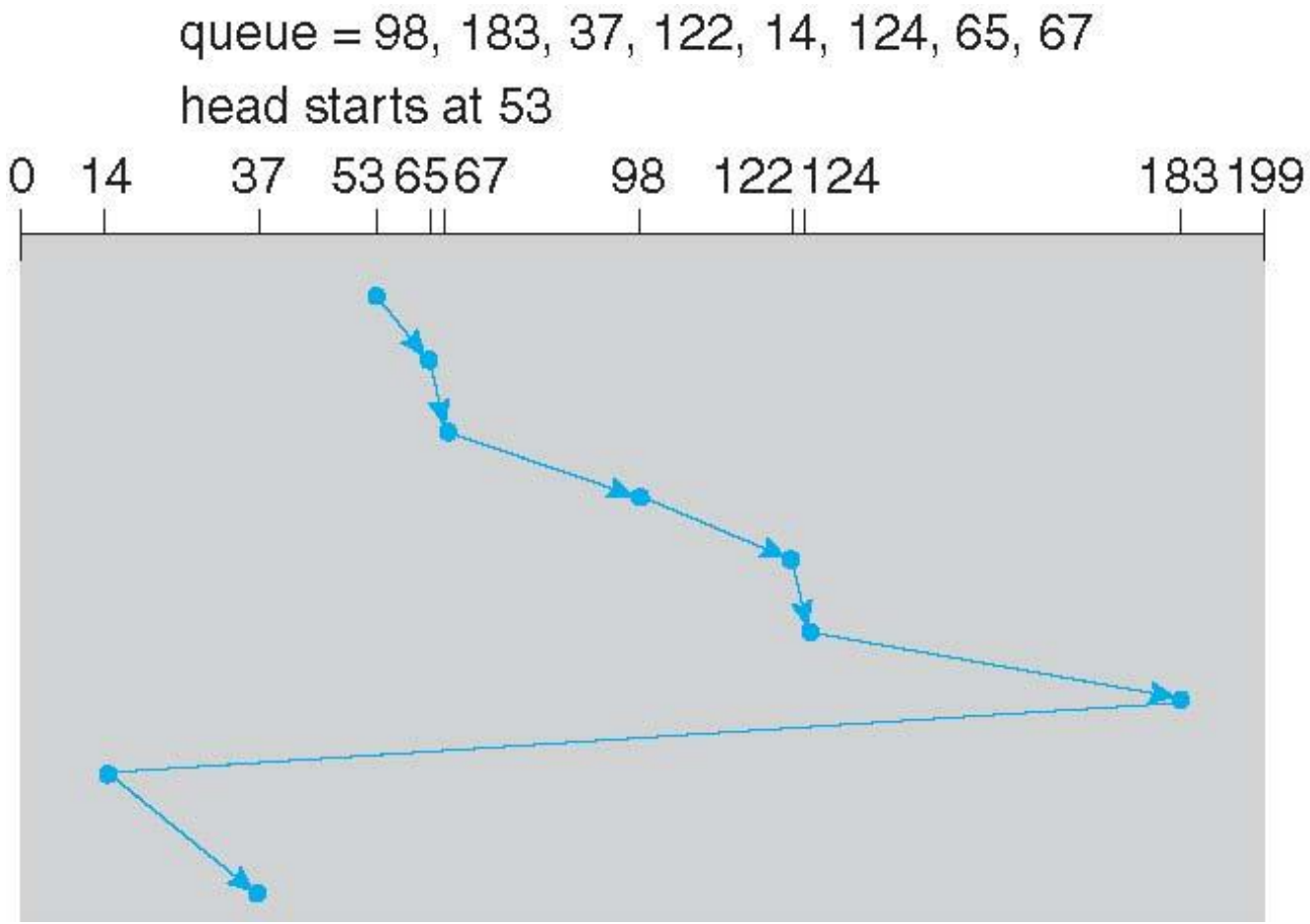
C-SCAN (Cont.)


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

LOOK and C-LOOK

- **LOOK** is a version of 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** a version of C-SCAN. Arm only goes as far as the last request in one 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

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout

Disk-Scheduling Algorithm (Cont.)

- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queuing effect OS queue ordering efforts?

References

Operating Systems Concepts by Silberschatz, Galvin, and Gagne