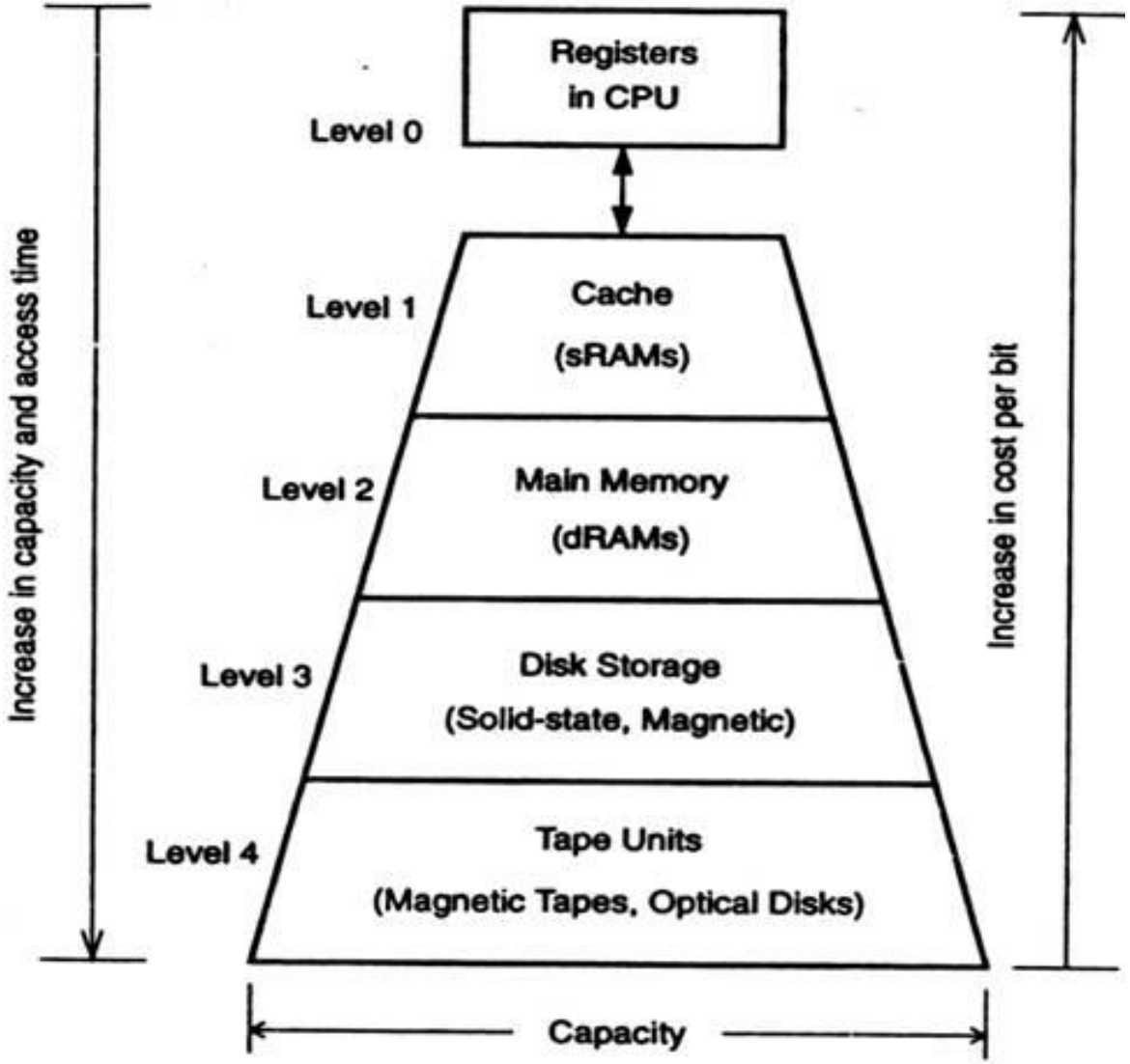
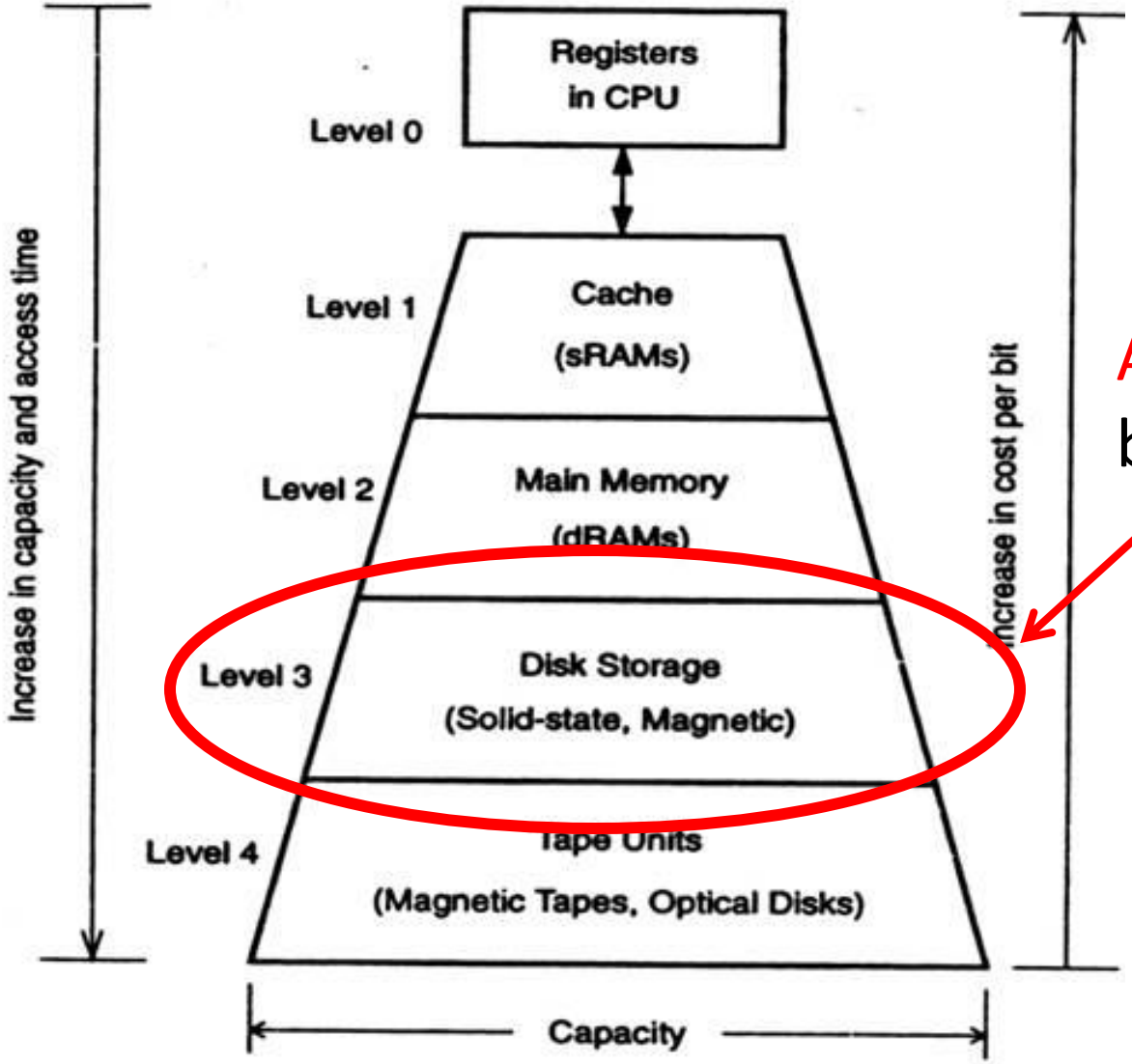
NYU
New York University

Operating Systems

File Systems II

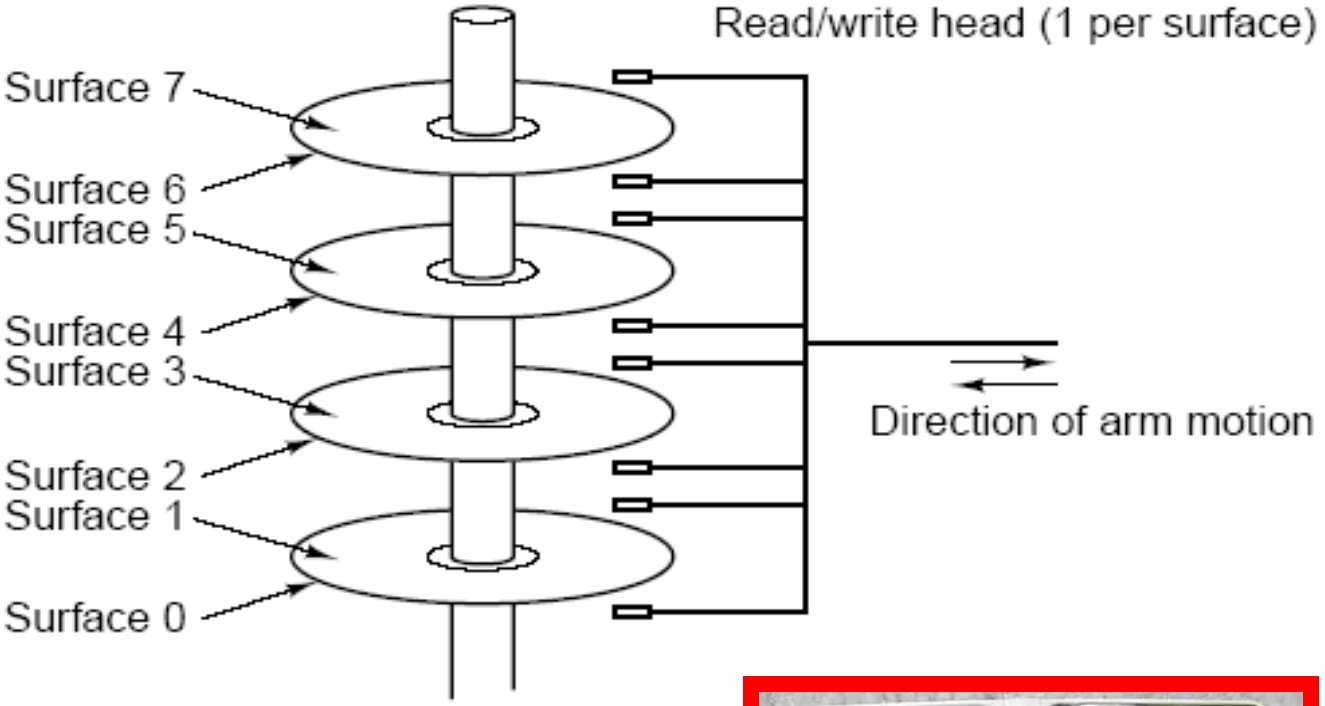
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

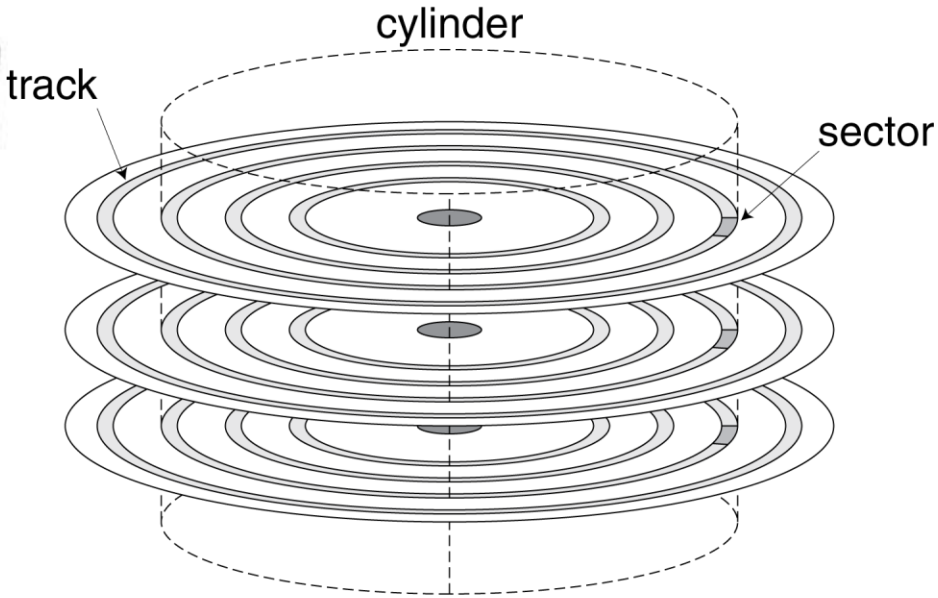

Registers
in CPU
Level 0
Level 1
Cache
(sRAMs)
Level 2
Main Memory
(dRAMs)
Level 3
Disk Storage
(Solid-state, Magnetic)
Level 4
Tape Units
(Magnetic Tapes, Optical Disks)
Increase in capacity and access time
Increase in cost per bit
Capacity



Abstracted
by OS as files

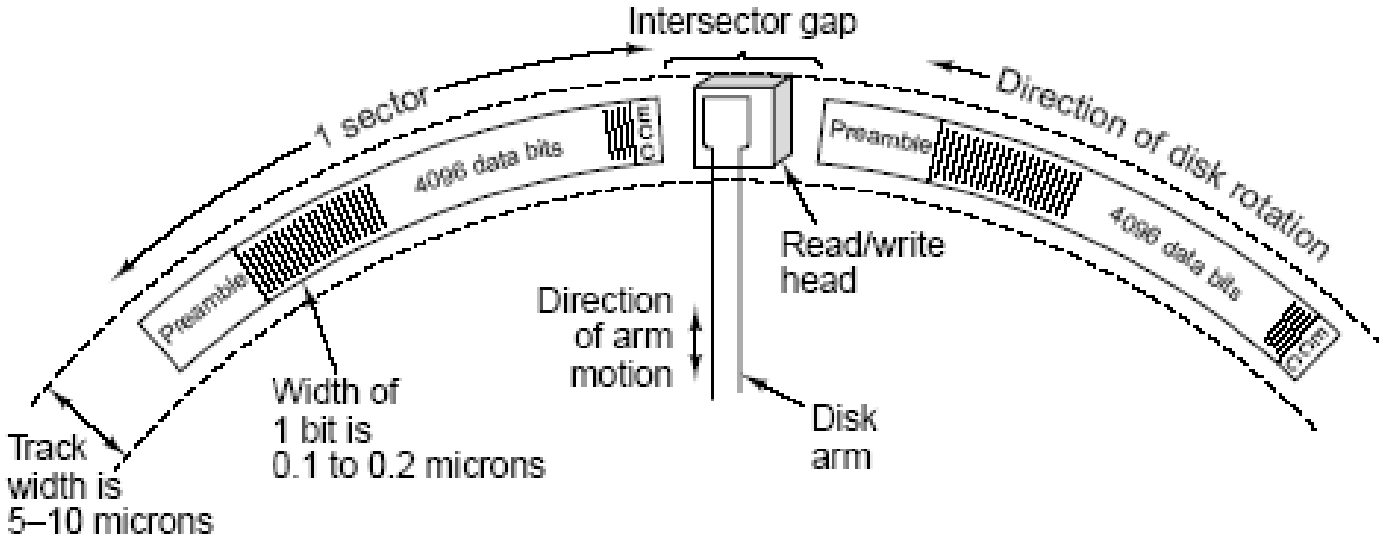
A Conventional Hard Disk (Magnetic) Structure




cylinder
track
sector

Hard Disk (Magnetic) Architecture

- **Surface** = group of tracks
- **Track** = group of sectors
- **Sector** = group of bytes
- **Cylinder**: several tracks on corresponding surfaces



This Lecture

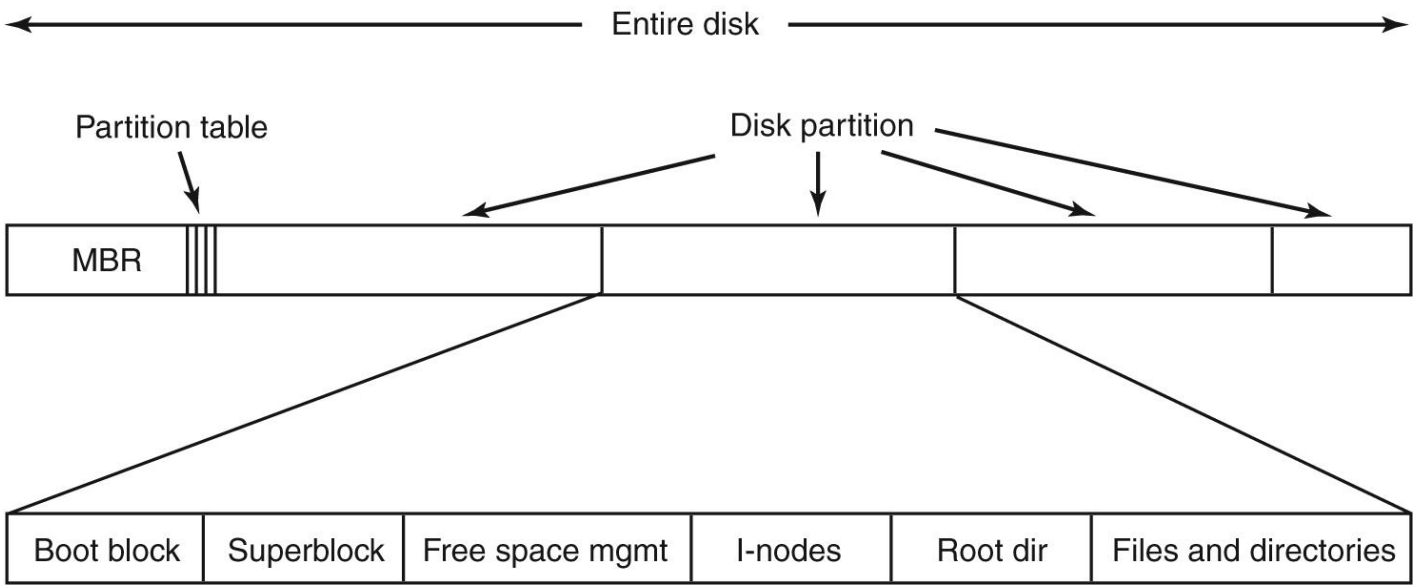
- How files and directories are stored?
- How disk space is managed?
- How to make everything work efficiently and reliably?

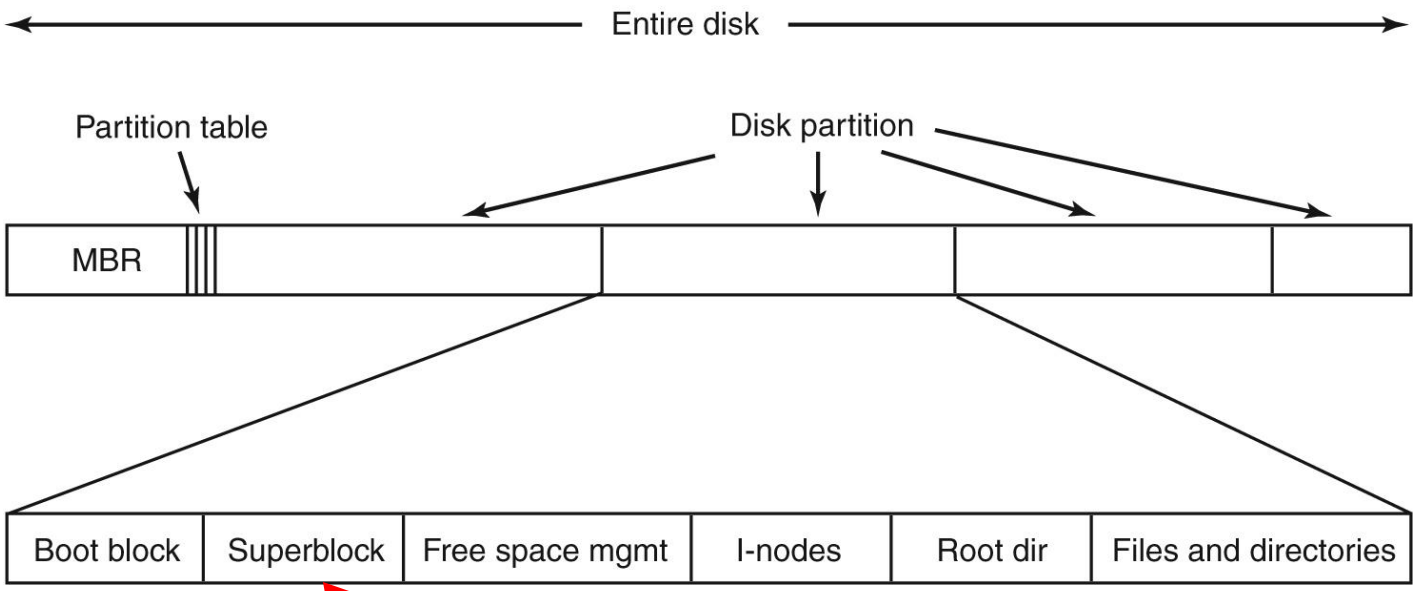
File System Layout

- Stored on disks
- Disks can have partitions with different file systems
- Sector 0 of the disk called MBR (Master Boot Record)
- MBR used to boot the computer
- The end of MBR contains the partition table

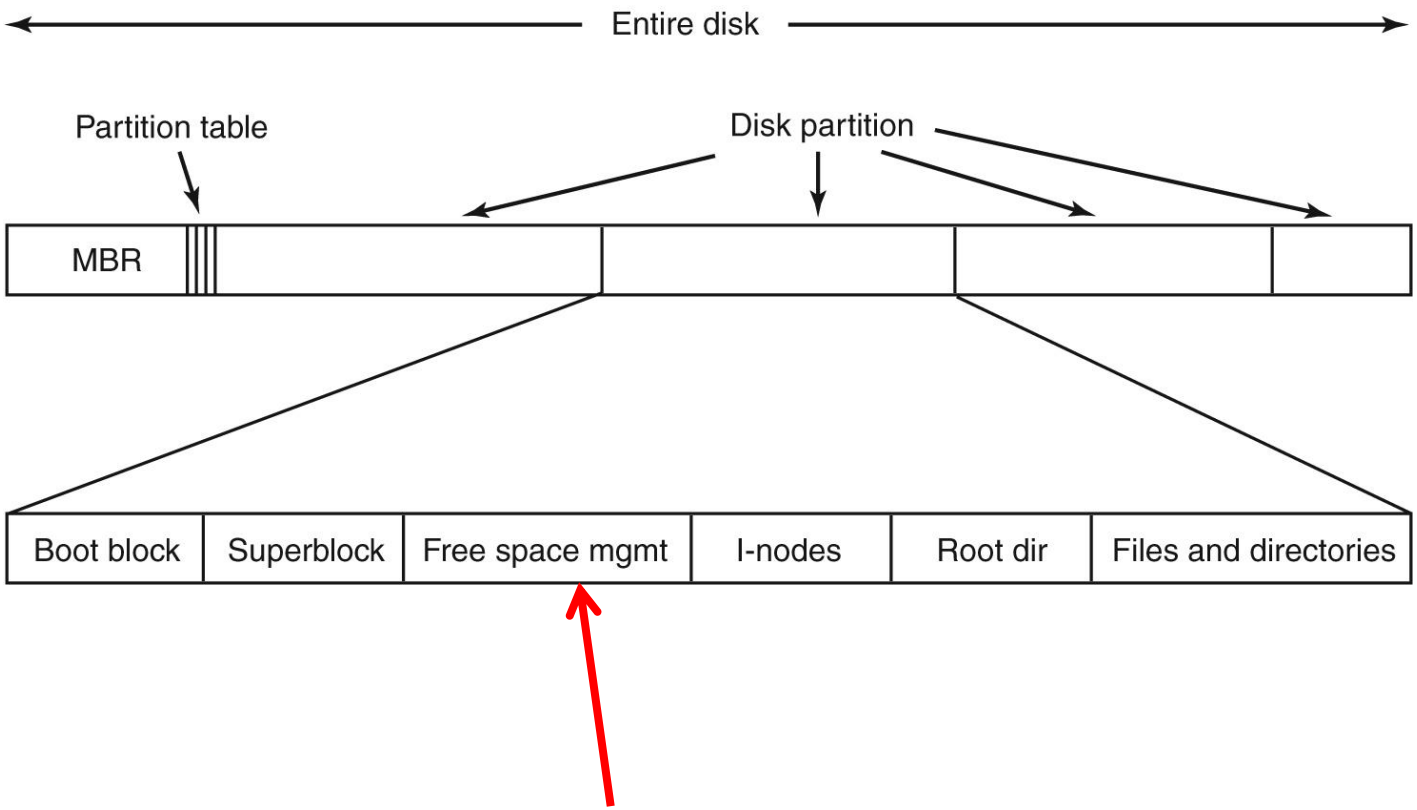
MBR and Partition Table

- Gives the starting and ending addresses of each partition
- One partition in the table is marked as active.
- When the computer is booted, BIOS executes MBR.
- MBR finds the active partition and reads its first block (called boot block) and executes it.
- Boot block loads the OS contained in that partition.

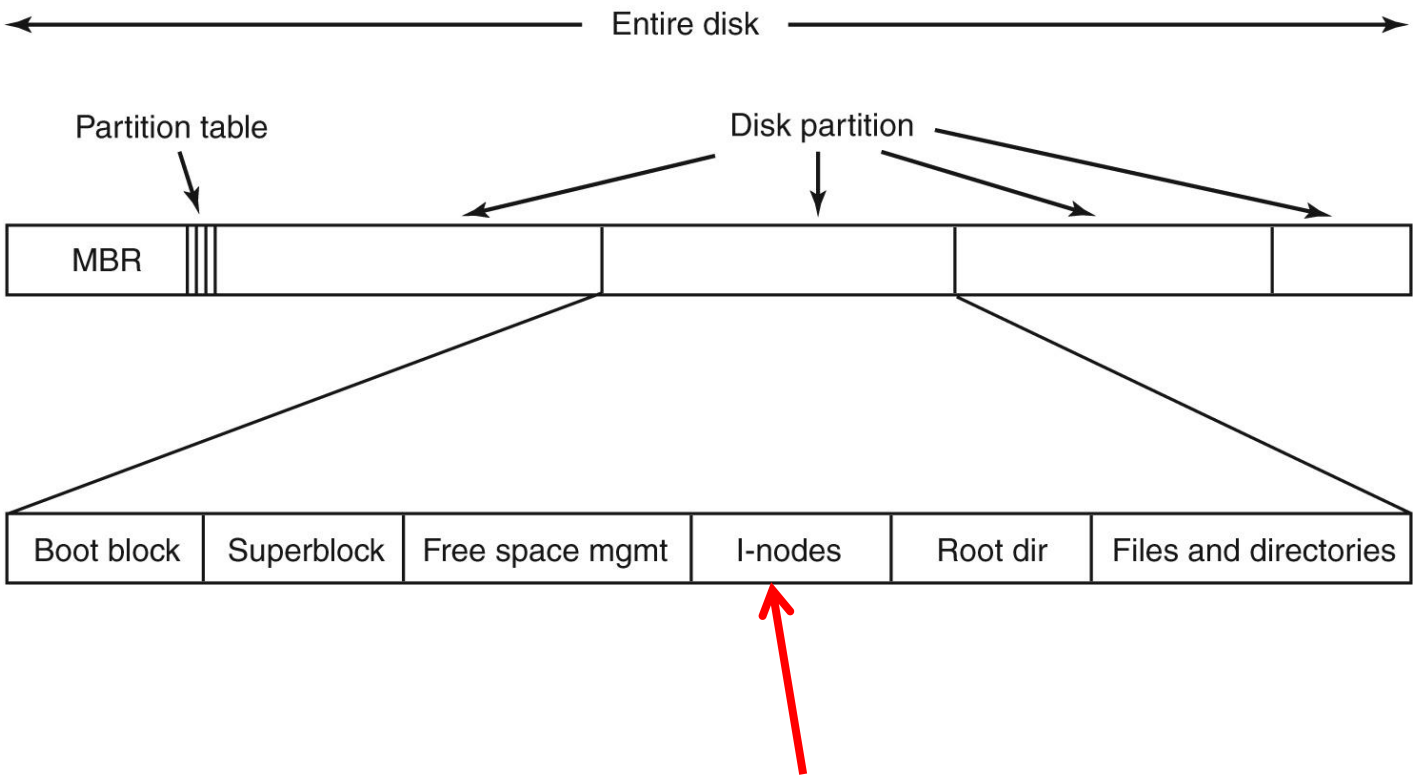

Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories



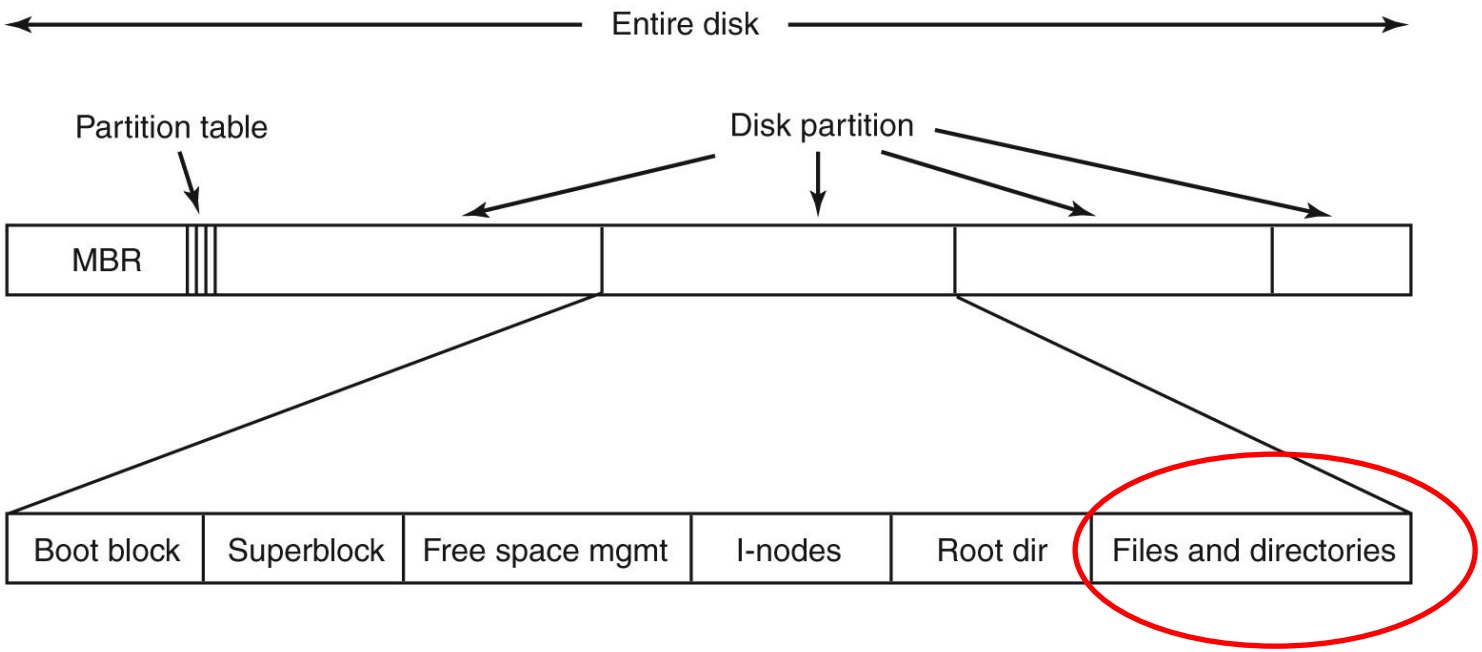
- Contains all key parameters about the file system.
- Is read into memory when computer is booted or file system is touched.



Bitmap or linked list



An array of data structures, one per file, telling about the file

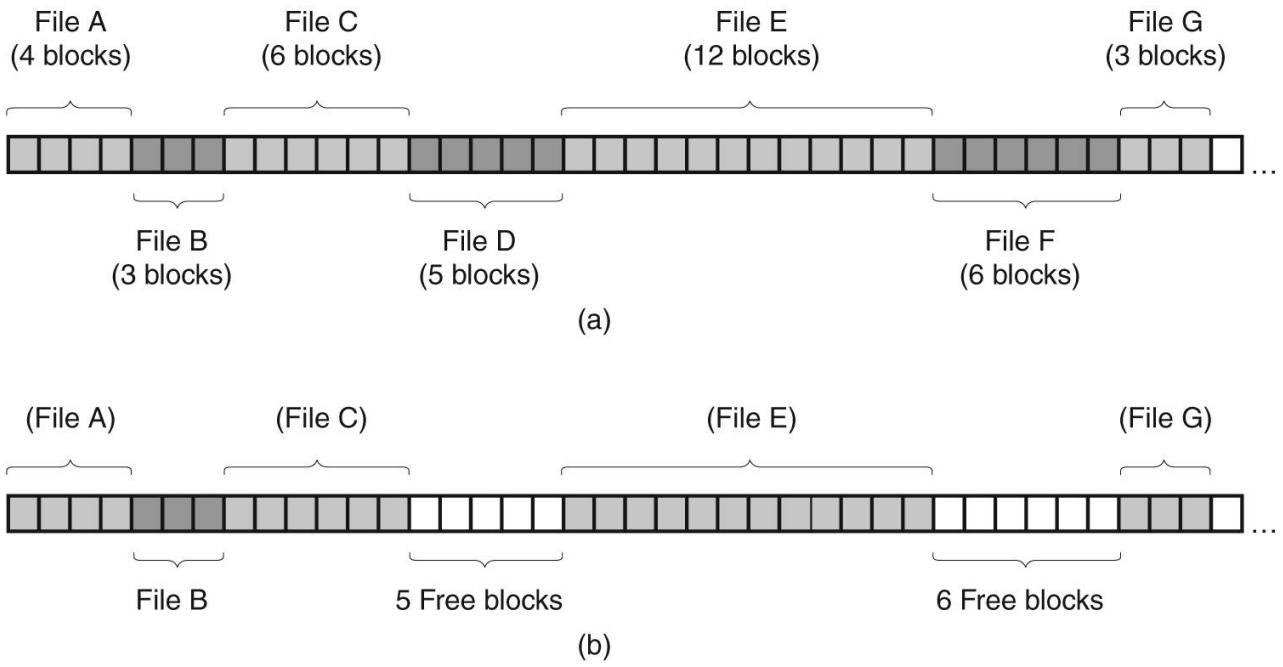


Which disk blocks go with which files?

Files

Implementing Files: Method 1: Contiguous Allocation

- Store each file as a contiguous run of disk blocks



After files D and F were deleted

Implementing Files: Contiguous Allocation

+ Simple to implement:

  Need to remember starting block address of the file and number of blocks

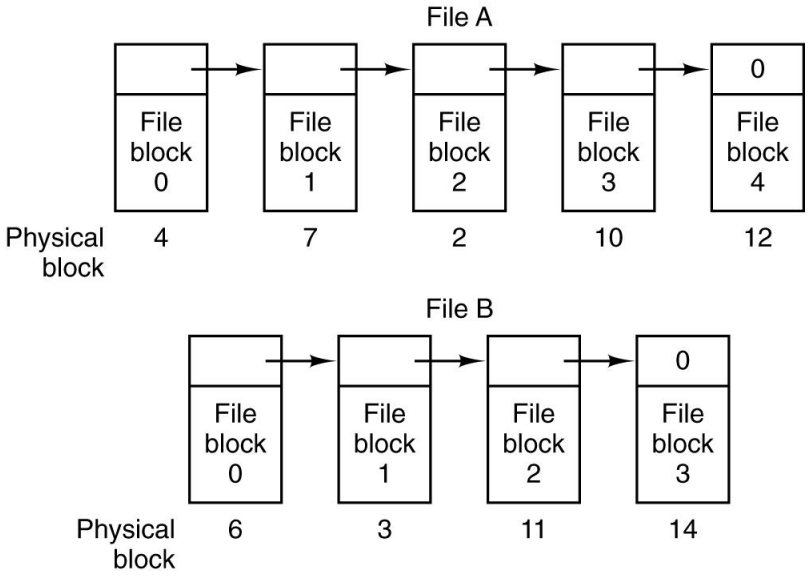
+ Read performance is excellent

  The entire file can be read from disk in a single operation.

- Disk becomes fragmented
- Need to know the final size of a file when the file is created

Implementing Files: Method 2: Linked List Allocation

- Keep a file as a linked list of disk blocks
- The first word of each block is used as a pointer to the next one.
- The rest of the block is for data.



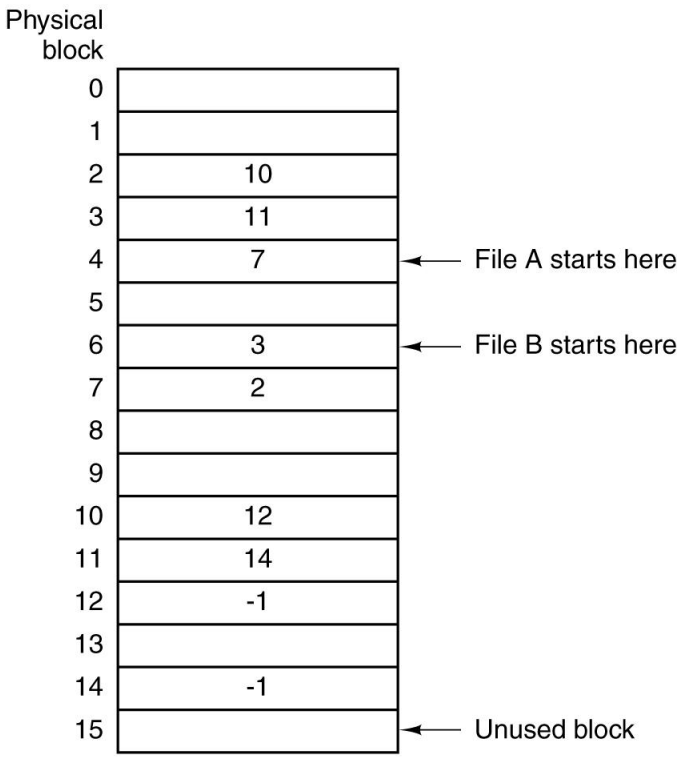
Implementing Files: Linked List Allocation

+ No (external) fragmentation
+ The directory entry needs to just store the disk address of the first block.
- Random access is extremely slow.
- The amount of data storage is no longer a power of two, because the pointer takes up a few bytes.

Implementing Files:

Method 3: Linked List Allocation Using a Table in Memory

- Take the pointer word from each block and put it in a table in memory.



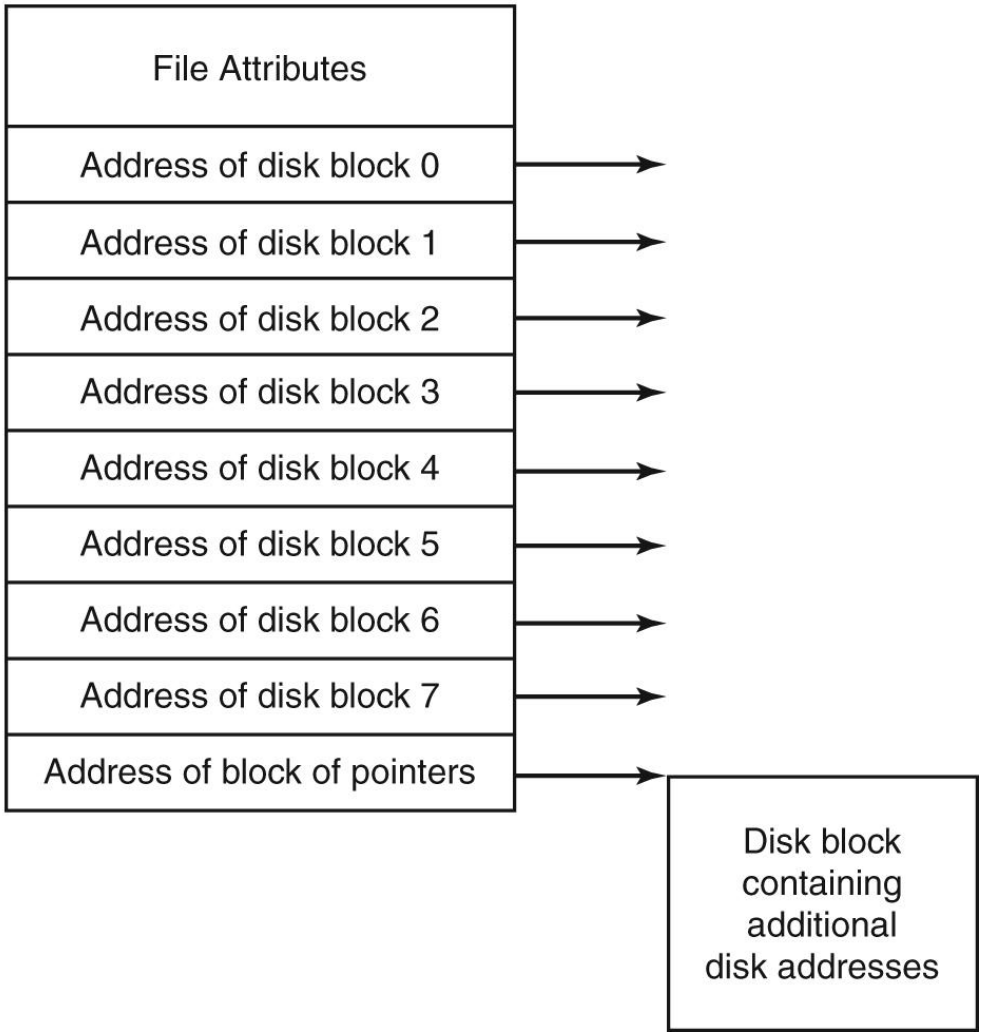
- This table is called:
  File Allocation Table (FAT)
- Directory entry needs to keep a single integer:
  (the start block number)
- The figure shows File A occupying blocks: 4, 7, 2,10, and 12.
- (-1) indicates the end of a file

Main drawback: Does not scale to large disks because the table needs to be in memory all the time.

Implementing Files: Method 4: I-nodes

- A data structure associated with each file
- Lists the attributes and disk addresses of the file blocks
- Needs only be in memory when the corresponding file is open.

Implementing Files: I-nodes

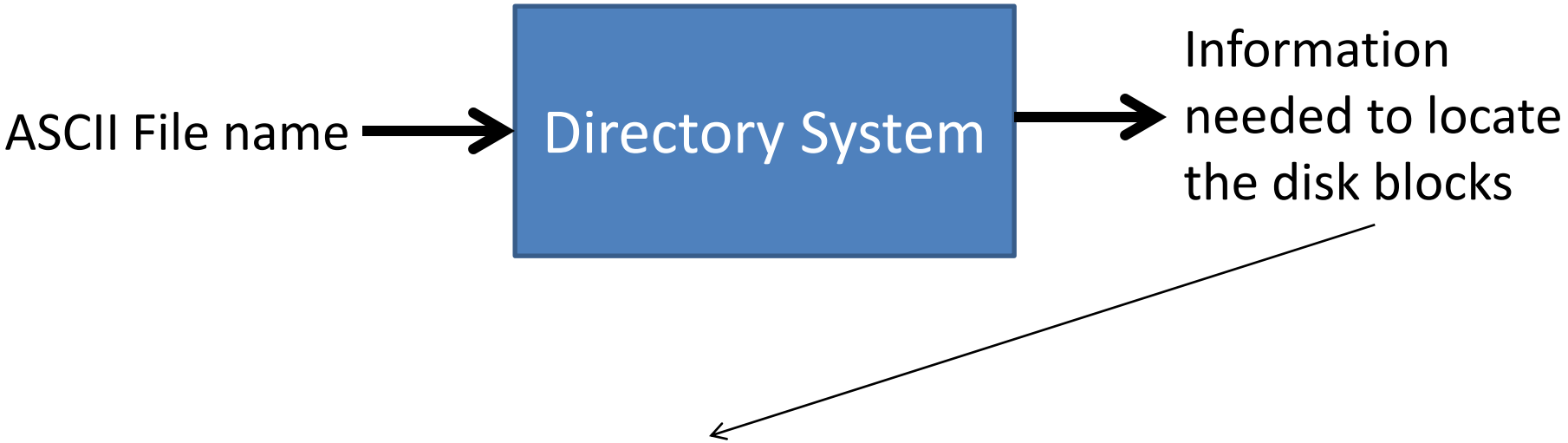


Directories

What are we talking about?

- To use a file, you need to open it.
- To open it, you must supply a file path to the OS (e.g. /home/usr1/data.txt).
- The OS uses that path to locate the directory entry on the disk.

Implementing Directories

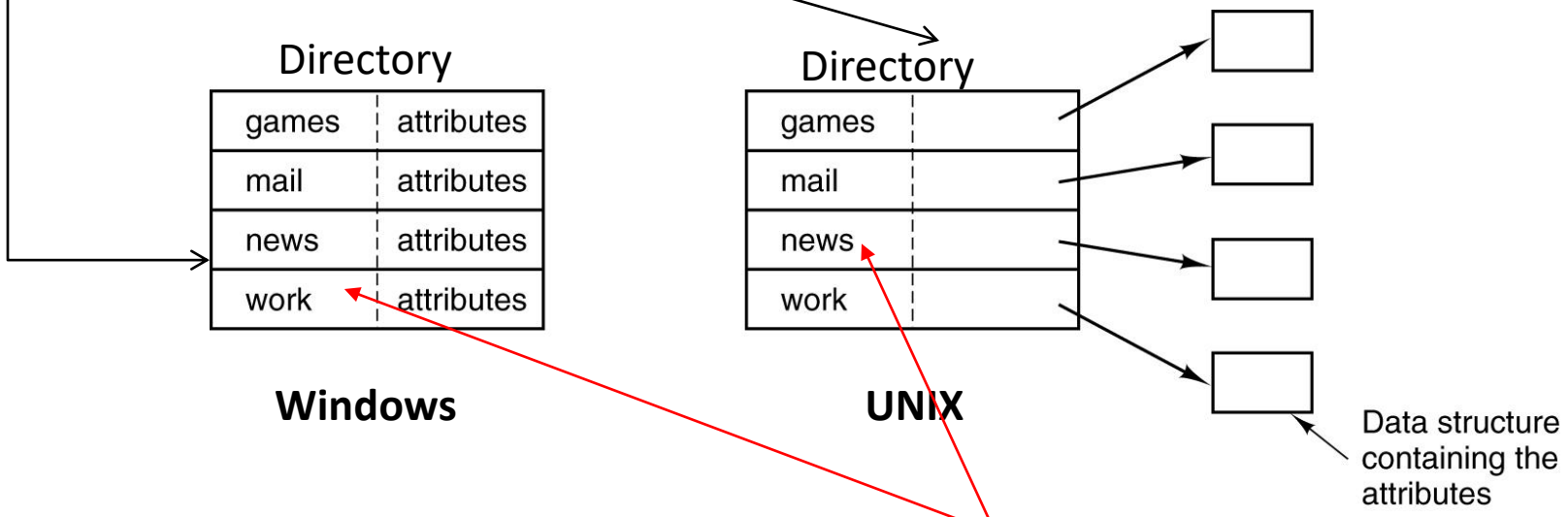


Example:

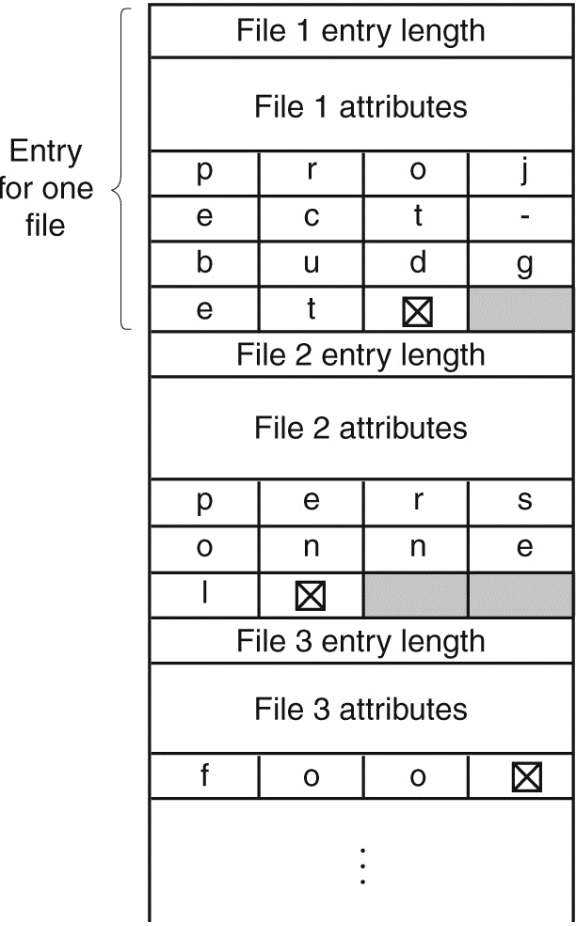
- Disk address of the file (in contiguous scheme)
- Number of the first block (in linked-list schemes)
- Number of the i-node

Implementing Directories

- Where the attributes of the file should be stored?
  - Directly in the directory entry
  - In the i-nodes



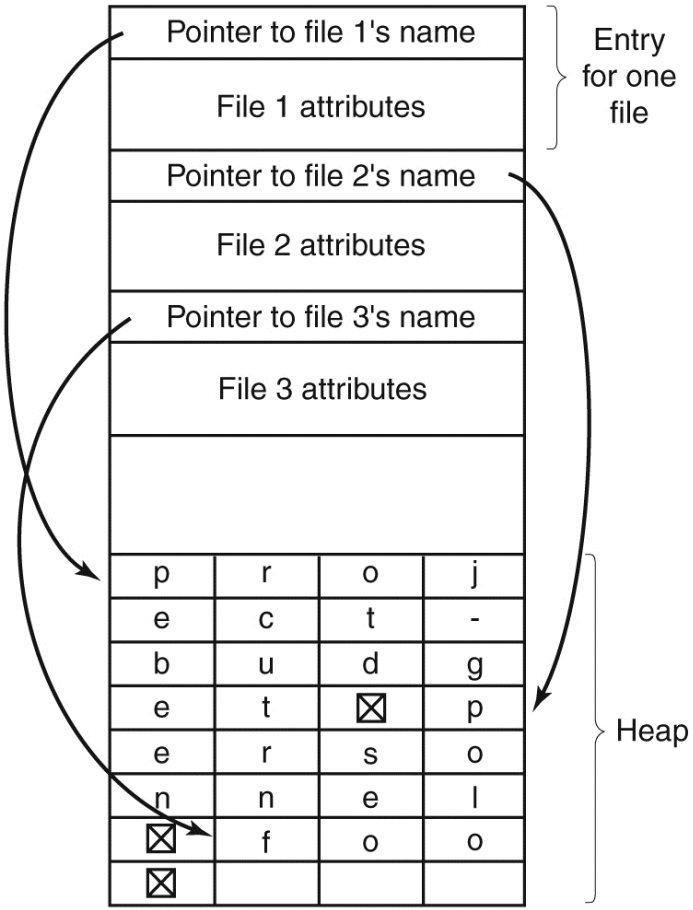
Implementing Directories: Variable-Length Filenames



Disadvantages:

- Entries are no longer of the same length.
-Variable size gaps when files are removed
- A big directory may span several pages which may lead to page faults.

Implementing Directories: Variable-Length Filenames



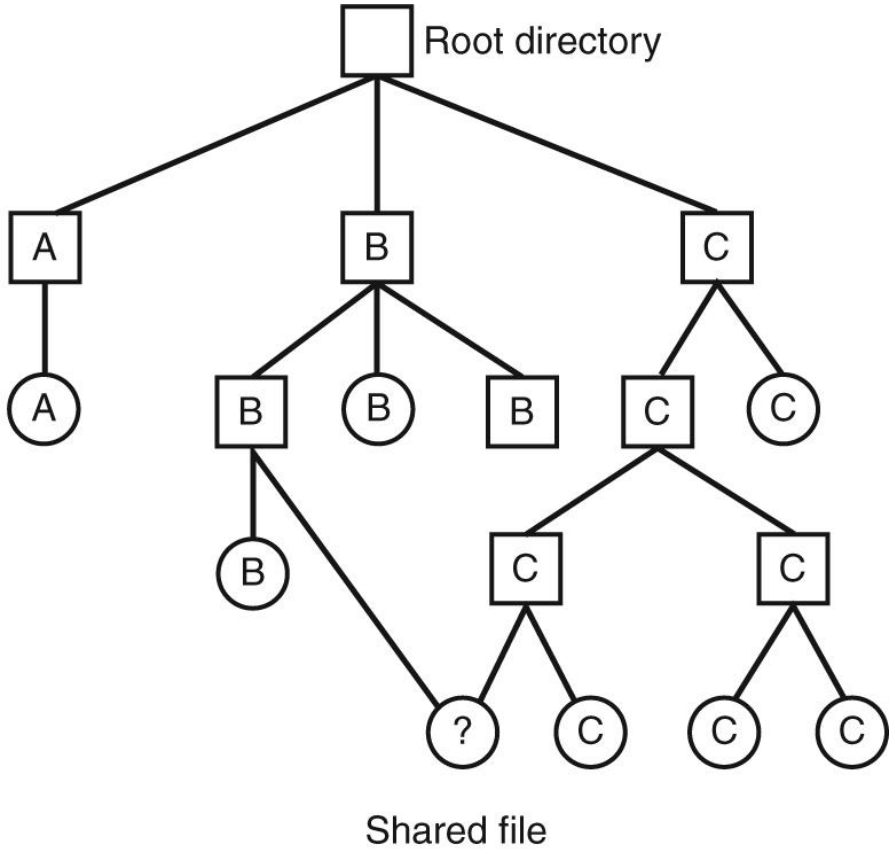
- Keep directory entries fixed length
- Keep filenames in a heap at the end of the directory.
- Page faults can still occur while accessing filenames.

Implementing Directories

- For extremely long directories, linear search can be slow.
  - Hashing can be used
  - Caching can be used

Shared Files

- Appear simultaneously in different directories

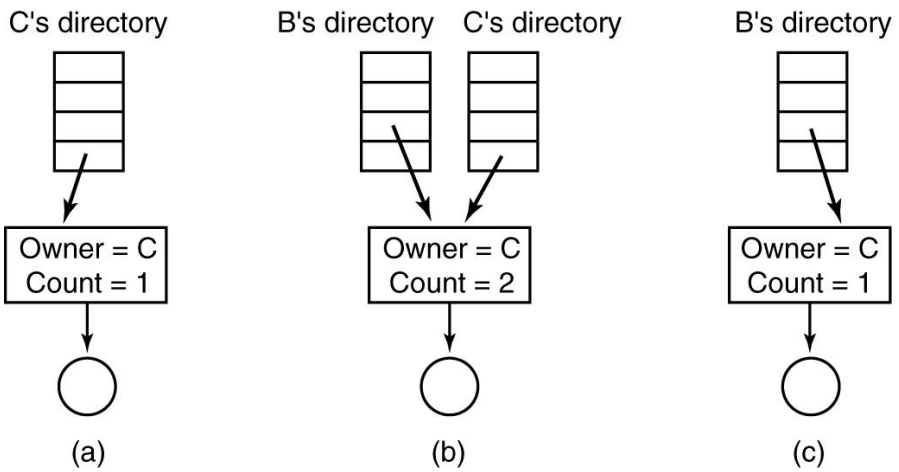


Shared file

Shared Files: Method 1

- Disk blocks are not listed in directories but in a data structure associated with the file itself (e.g. i-nodes in UNIX).
- Directories just point to that data structure.

Problematic Scenario



Shared Files: Method 2

- Have the system create a new file (of type LINK). This new file contains the path name of the file to which it is linked.
- This approach is called: symbolic linking
- The main drawback is the extra overhead.

Special File Systems

Log-Structured File Systems: What is the Motivation?

- Disk seek time does not improve as fast relative to CPU speed, disk capacity, and memory capacity.
- Disk caches can satisfy most requests
  - A disk cache is a buffer in memory that stores the most recently accessed blocks of a disk.

SO:

- In the future, most disk accesses will be writes because the reads will mostly be served by disk caches.
- Writes to disk are slow in traditional file systems.

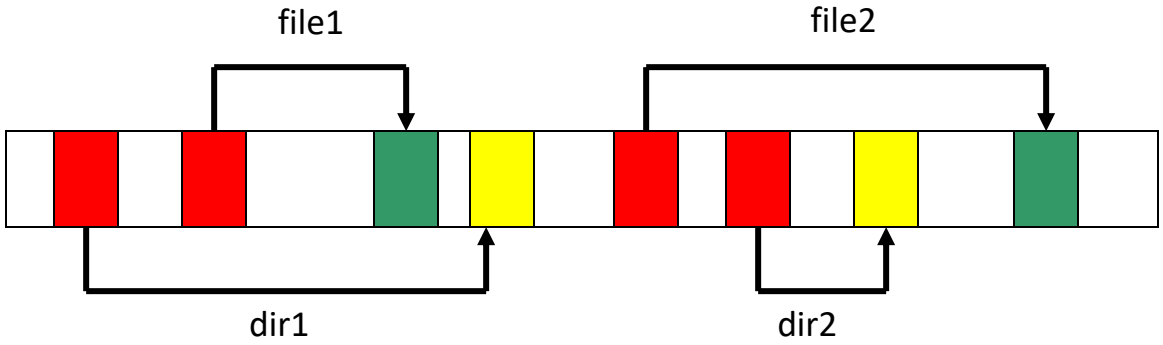
Log-Structured File Systems: The Main Idea

- Structure the entire file as a log.
  - Log is a data structure that has the data always added at the end of the log.
- Periodically (or when in need):
  - All pending writes buffered in memory are collected into a single segment.
  - The segment is written in disk at the end of the log.
  - The segment can contain data and meta data (e.g. inodes, directories, etc).
- An i-node map, indexed by i-number, is maintained.
- The map is kept on disk and is also cached.

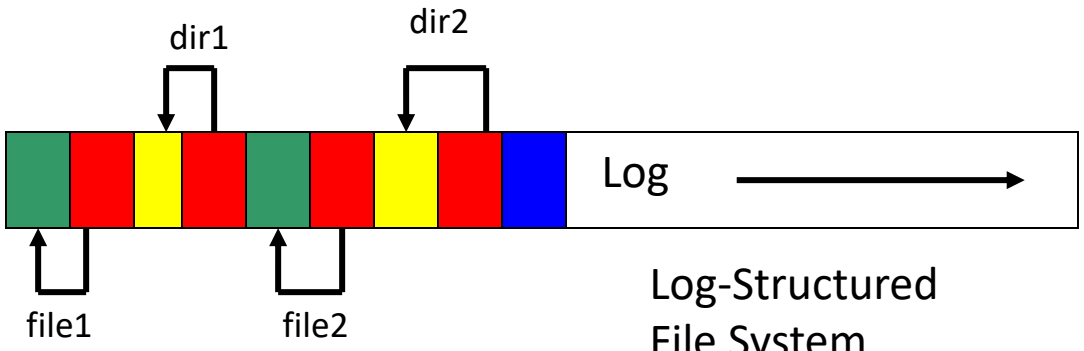
Log-Structured File Systems: The Pieces of the Puzzle

- **i-node**: as we saw before, i-node contain physical block pointers to files.
- **i-node map**: a table indicating where each i-node is on the disk.
- **segment**: can contain a mix of data blocks, i-nodes, and i-node maps.

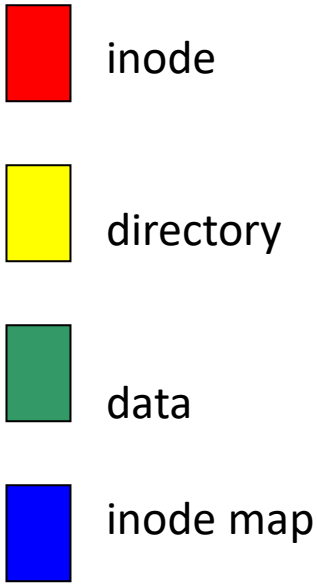
Log-Structured File System: vs Traditional Unix file system



Unix File System



Log-Structured File System



Log-Structured File Systems: Cleaning

- As files get deleted, change sizes, moved, etc, the log becomes fragmented.
- A cleaner thread scans log to compact it and discard unneeded information.

Journaling File System

- Keep a log of what the file system is going to do before it does it.
- If the system crashes before it is done, then after rebooting the log is checked and the job is finished.
- Example: Microsoft NTFS, Linux ext3

Disk Space Management

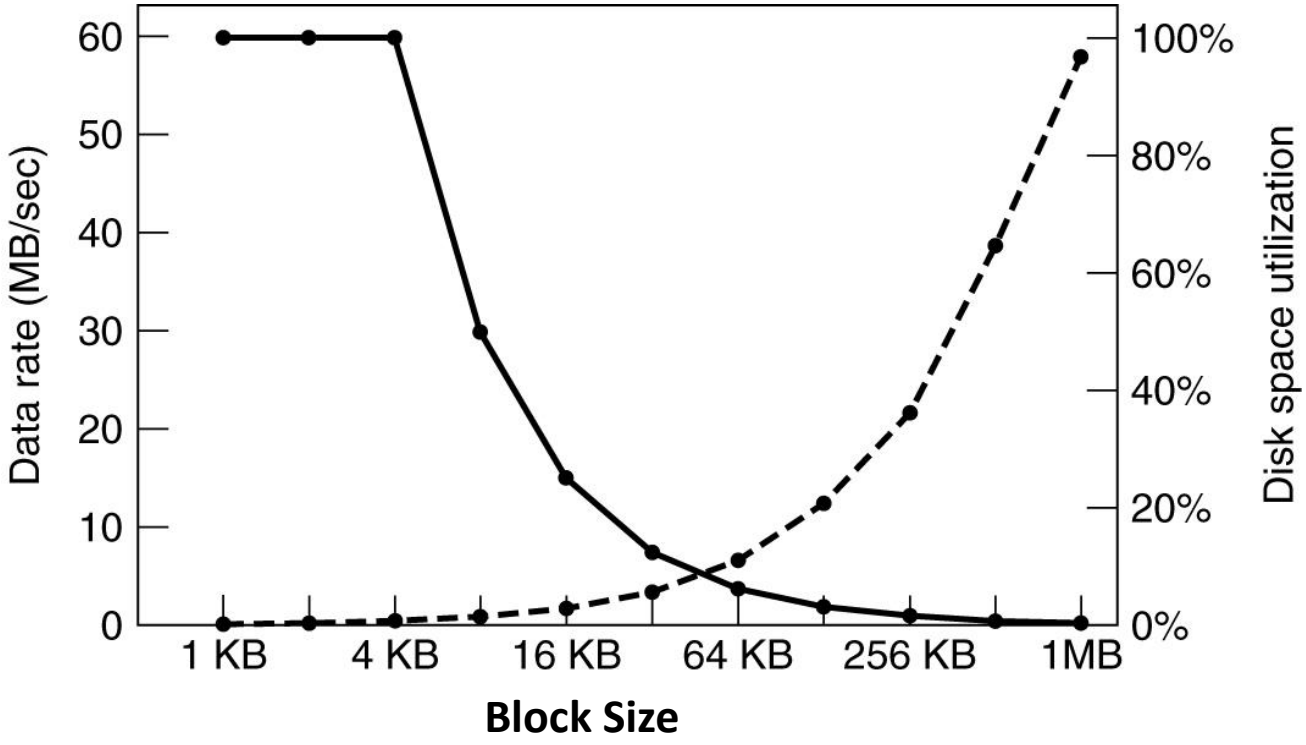
Disk Space Management

- All file systems chop files up into fixed-size blocks that need not be adjacent.
- Block size:
  - Too large -> we waste space
  - Too small -> we waste time

Access time for a block is completely dominated by the seek time and rotational delay (assume traditional hard-disk).

So … The more data are fetched the better.

(dotted)

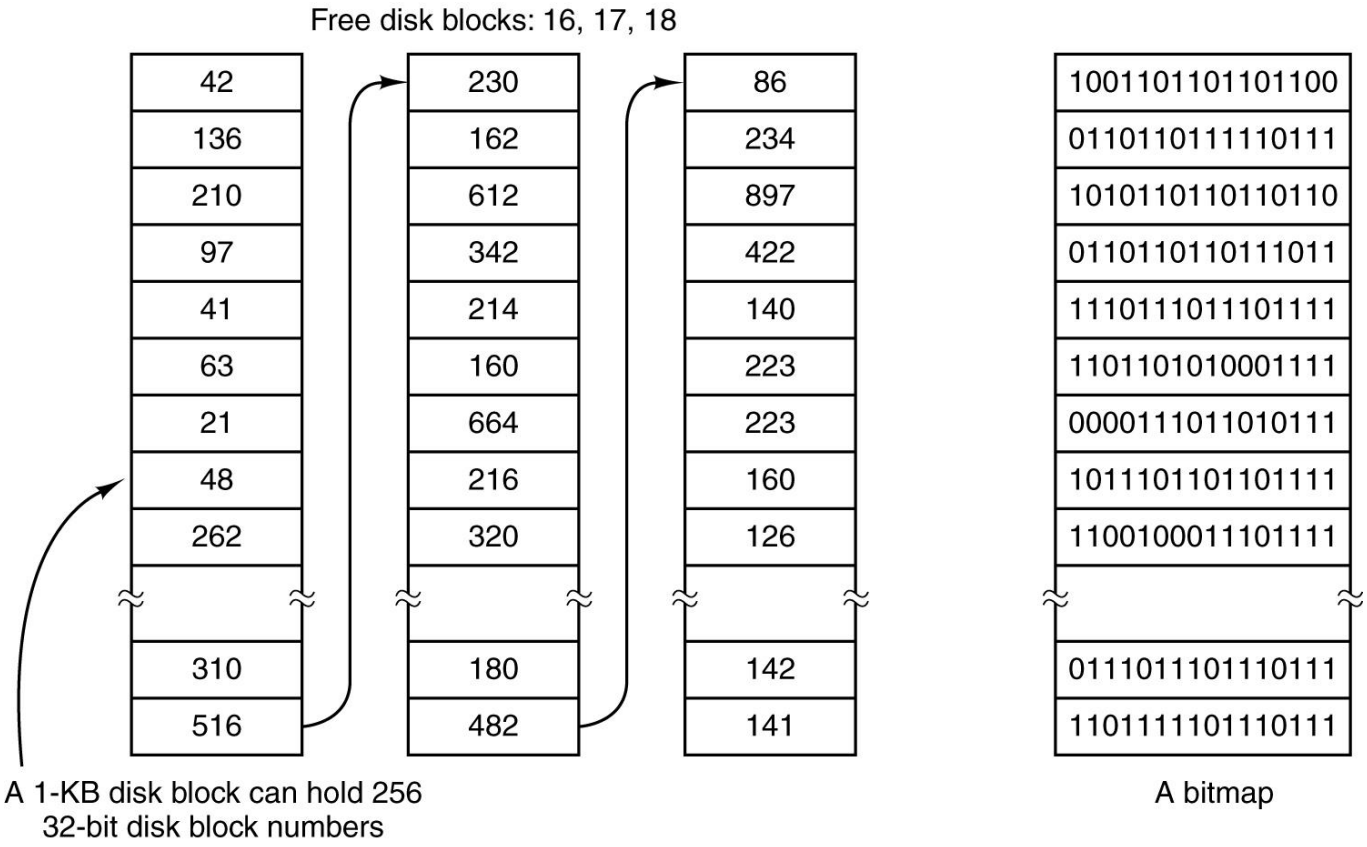


Disk Space Management: Keeping Track of Free Blocks

- **Method 1**: Linked list of disk blocks, with each block holding as many free disk block numbers as possible.

- **Method 2**: Using a bitmap

Disk Space Management: Keeping Track of Free Blocks

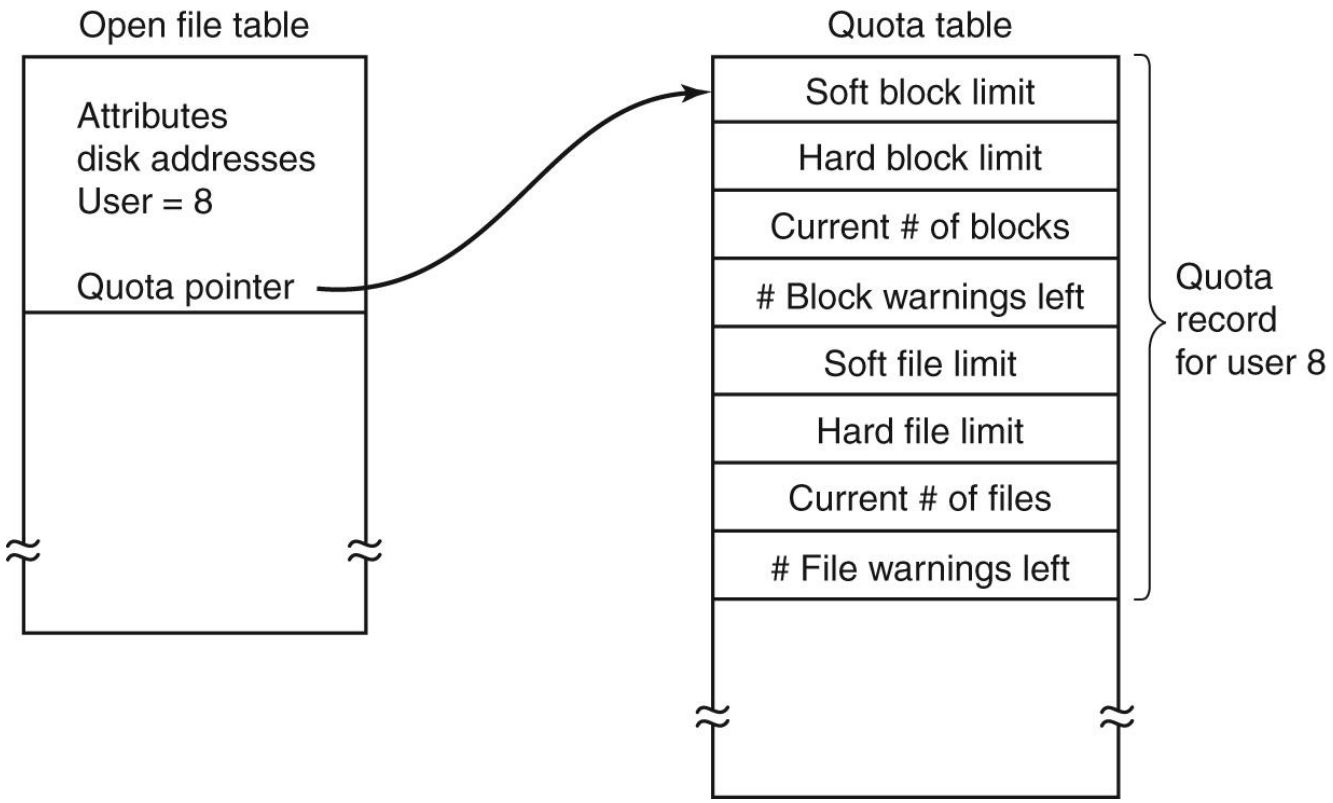


Free blocks are holding the free list.

Disk Space Management: Disk Quotas

- When a user opens file
  - The attributes and disk addresses are located
  - They are put into an open file table in memory
  - A second table contains the quota record for every user with a currently open file.

Disk Space Management: Disk Quotas



File System Reliability: Backup Consistency

File System Backups

- It is usually desirable to back up only specific directories and everything in them than the entire file system.
- Since immense amounts of data are typically dumped, it may be desirable to compress them.
- It is difficult to perform a backup on an active file system.
- Incremental dump: backup only the files that have been modified from last full-backup

File System Backups: Physical Dump

- Starts at block 0 of the disk
- Writes all the disk blocks onto the output tape (or any other type of storage) in order.
- Stops when it has copied the last one.

+ Simplicity and great speed

- Inability to skip selected directories and restore individual files.

File System Backups: Logical Dump

- Starts at one or more specified directories
- Recursively dumps all files and directories found there and have changed since some given base date.

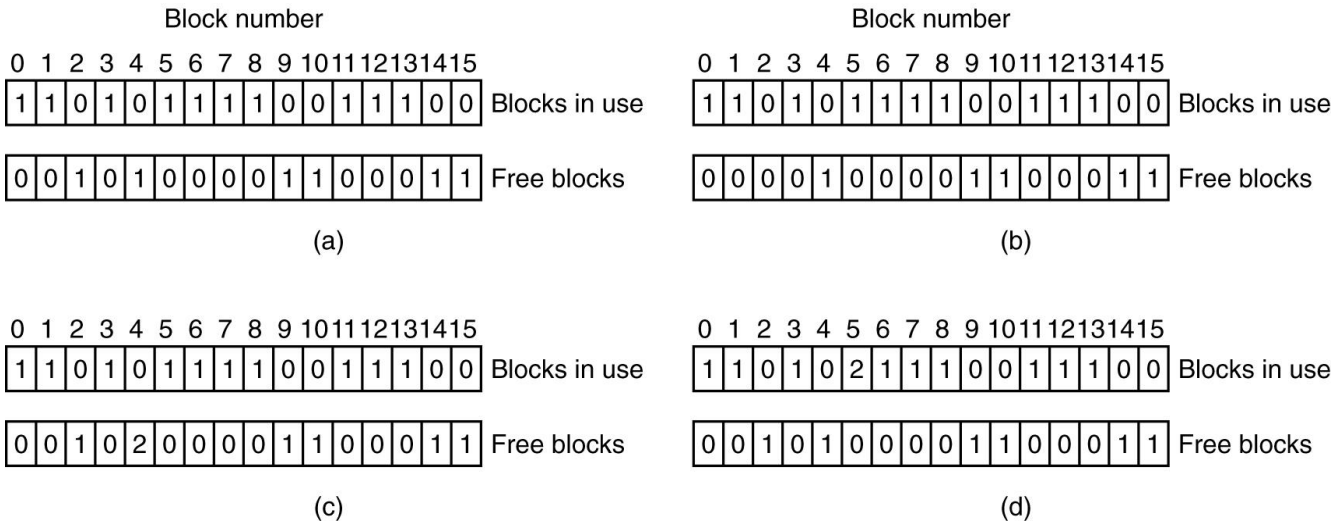
File System Consistency

- Two kinds of consistency checks
  - Blocks
  - Files

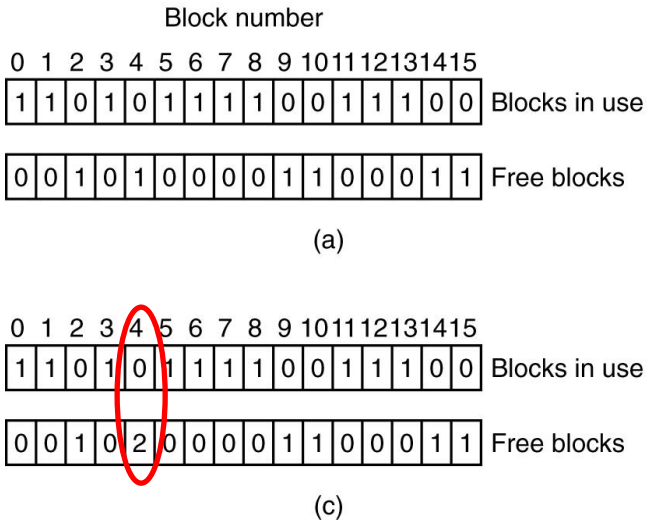
File System Consistency: Blocks

- Build two tables, each one contains a counter for each block, initially 0
- Table 1: How many times each block is present in a file
- Table 2: How many times a block is present in the free list
- A consistent file system: each block has 1 either in the first or second table

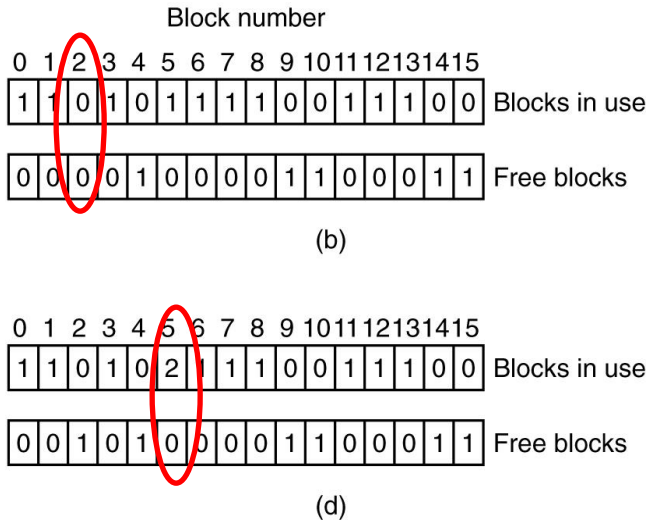
File System Consistency: Blocks



File System Consistency: Blocks



Add the block to the free list



Rebuild the free list

Allocate a free block, make a copy of that block and give it to the other file.

File System Consistency: Files

- Table of counters; a counter per file
- Counts the number of that file's usage count.
- Compares these numbers in the table with the counts in the i-node of the file itself.
- Both counts must agree.

File System Consistency: Files

- Two inconsistencies:
  - count of i-node > count in table
  - count of i-node < count in table
- Fix: set the count in i-node to the correct value

File System Performance

File System Performance

- Caching:
  - Block cache: a collection of blocks kept in memory for performance reasons
- Block Read Ahead:
  - Get blocks into the cache before they are needed
- Reducing Disk Arm Motion:
  - Putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder
- Defragmentation

Conclusions

- Files and file system are major parts of an OS.
- There are different ways of organizing files, directories, and their attributes.
- Files and File system are the OS way of abstracting storage.