CISC 7310X

# C13: Files and Directories: System's Perspective

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# File Systems: Requirements

- Long term data storage
  1. It must be possible to store a very large amount of data.
  2. Data must survive termination of process using it.
  3. Multiple processes must be able to access data concurrently.

# Storage Devices: Disks

- Long-term storage devices are abstracted as “disks”
- A disk is abstracted as a linear sequence of fix-sized blocks
  - Two operations
    - Read block k
    - Write block k

# Common Queries

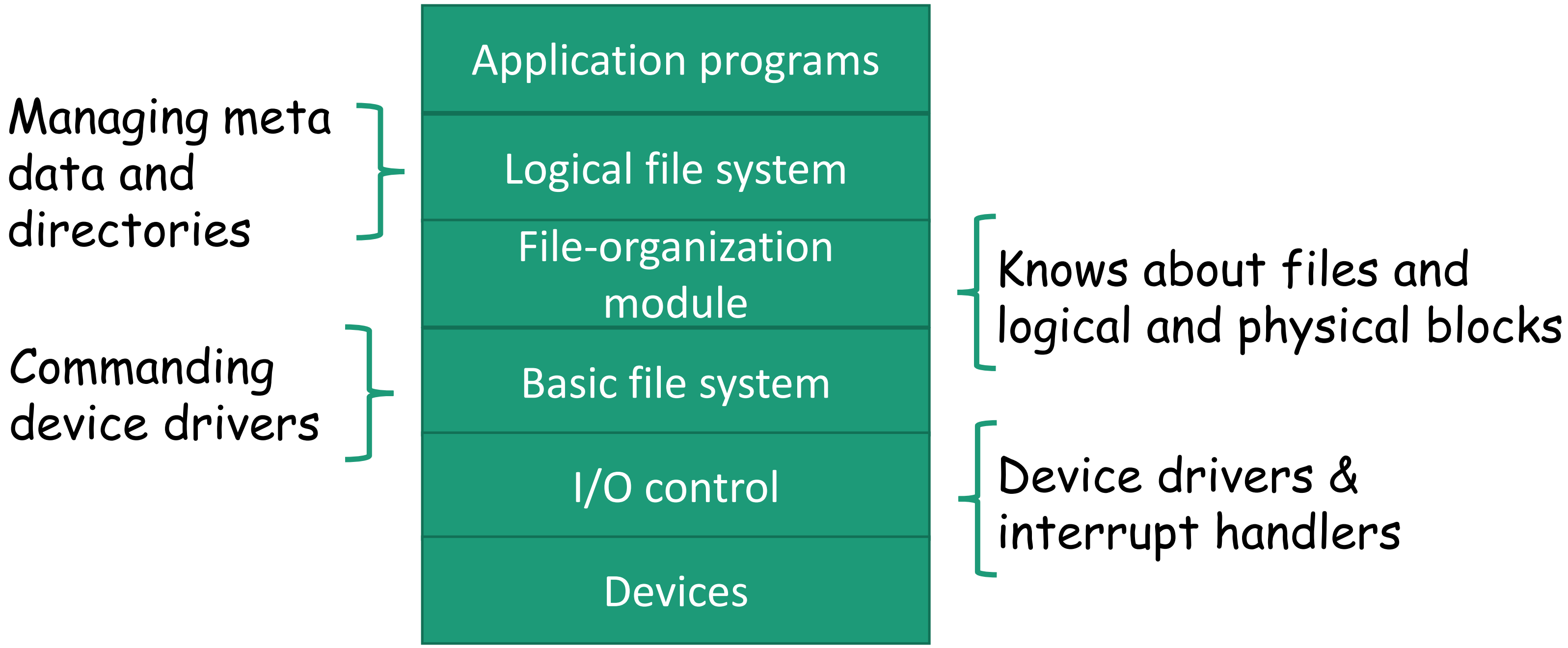
1. How do you find information?
2. How do you keep one user from reading another user's data?
3. How do you know which blocks are free?

# Outline

- Overview and design
- Space Allocation
  - Contiguous, linked, and indexed
- Free-space management
  - Bitvector, linked, grouping, counting, maps
- Efficiency and performance
  - Do we waste space, is it fast?
  - Caching, and replacement algorithms
- Recovery and backups
  - Consistency checking, log-structured, and journaling
  - Backups, full backups, incremental backups
- (space and block are interchangeable)

# File System Structure

- Layered approach



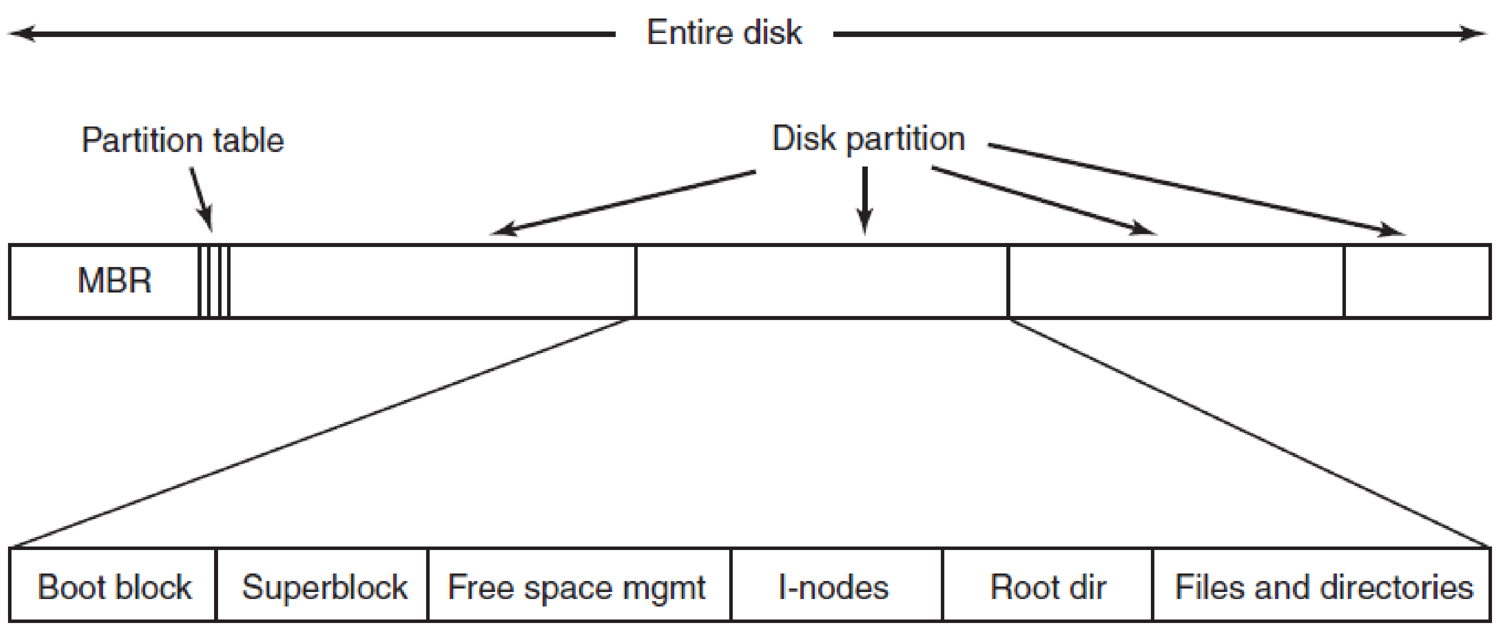
# Data Structures on Disk

- On-disk and in-memory data structures, varying on different systems
- On disk,
  - Boot control block (sector 0)
    - Master boot record, partition tables
  - Volume/partition control bock (per volume/partition)
    - Number of blocks, size of the blocks, free-block count …, superblock in some systems
  - Directory structure (per file system)
    - Associated file names, where to find the allocation of the files
  - File Control block (per file)

# Data Structures in Memory

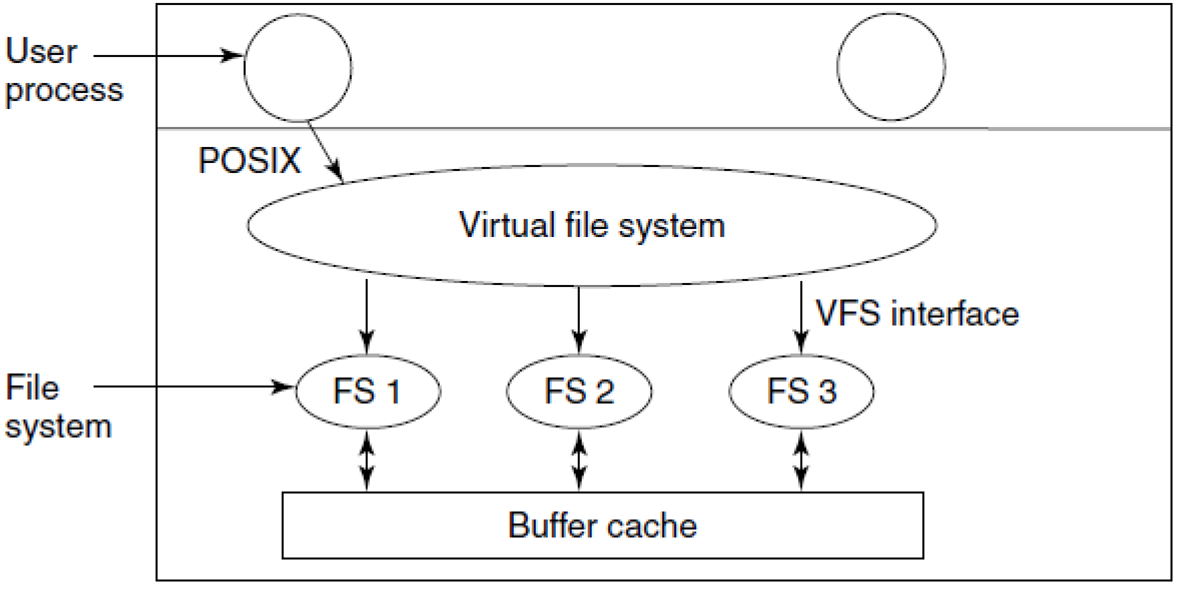
- In-memory mount table
  - Mounted volumes
- In-memory directory structure cache
  - Data about recently accessed directories
- System-wide open-file table
  - Copy of the FCB of each open file, and others
- Per-process open-file table
  - Pointers to the appropriate entry in the system-wide open-file table, and others
  - File descriptor/file handle
- Buffers hold file system blocks being read from or written to disk

# File System Layouts




- [Figure 4-9 in Tanenbaum & Bos, 2014]

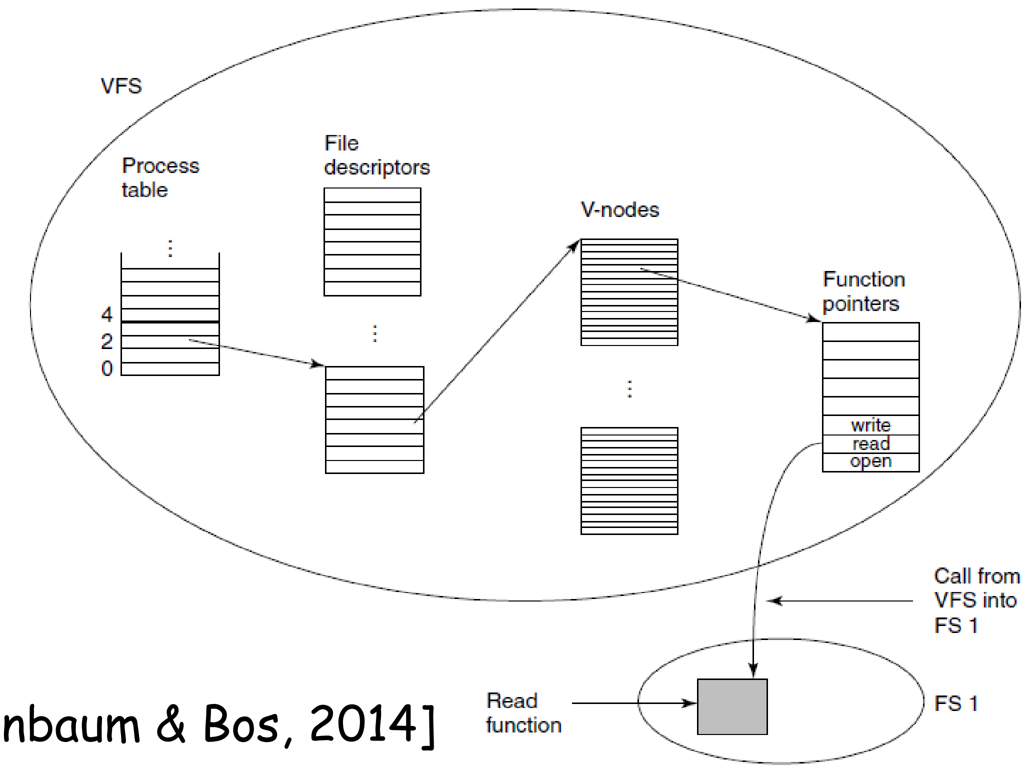
# Virtual File Systems




- [Figure 4-17 in Tanenbaum & Bos, 2014]

# VFS: Data Structures

- A simplified view




- [Figure 4-19 in Tanenbaum & Bos, 2014]

# Questions?

- Overview of file systems implementation
  - Necessary data structures on disk and in memory
- Layered approach

# Space Allocation

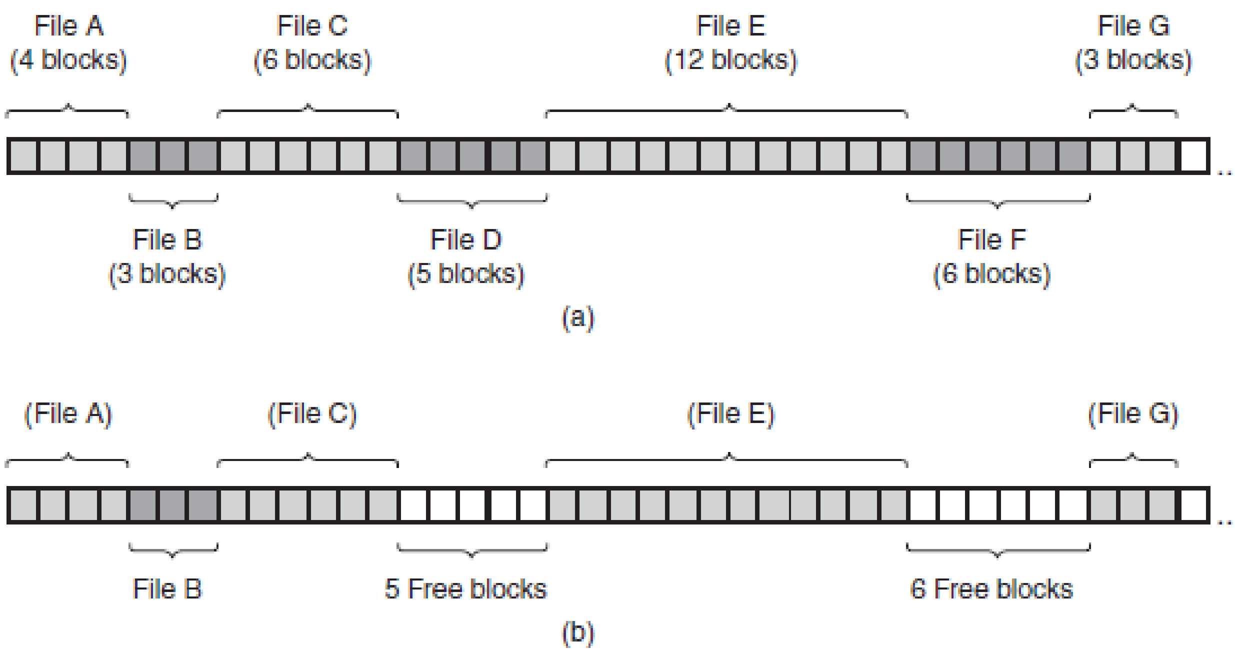
- Contagious allocation
- Linked allocation
- Indexed allocation

# Contiguous Allocation

- Each file occupies a set of contiguous blocks
  - File A start at block a with size n
    - block: a, a+1, …, a+n-1
    - Minimize disk head movement
    - FCB: a and n
- Where to allocate new file?
  - Dynamic storage-allocation, recall memory allocation
- Problem
  - External fragmentation, new file  may not fit
  - Space planning, how much space to allocate?

# Holes and External Fragmentation

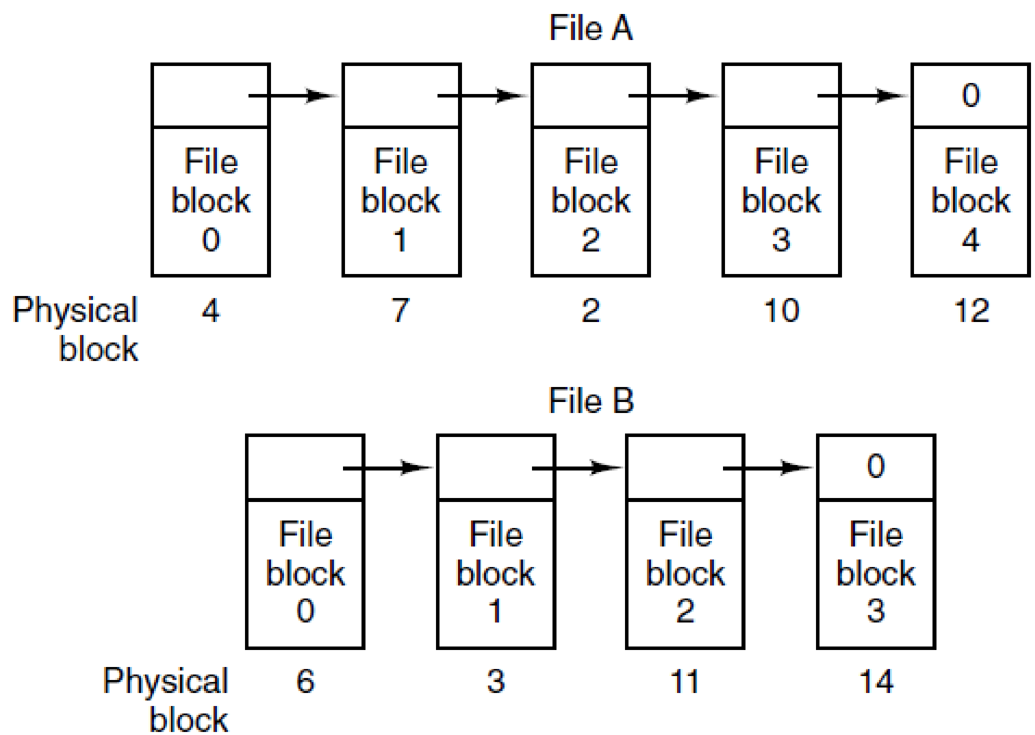
- Allocation and deallocation: holes forming




  - [Figure 4-10 in Tanenbaum & Bos, 2014]

# Linked Allocation

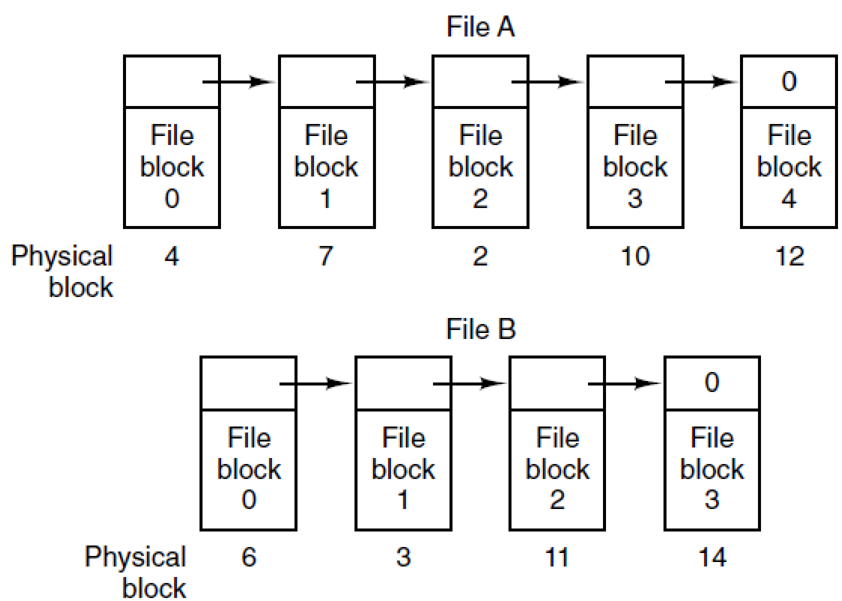
- A file is a linked list of blocks




- [Figure 4-11 in Tanenbaum & Bos, 2014]

# Linked Allocation: FCB

- FCB: a pointer to the first, the last blocks of the file, number of blocks
  - File A: 4, 12, 5
  - File B: 6, 14, 4
- New File
  - File C: nil, nil, 0



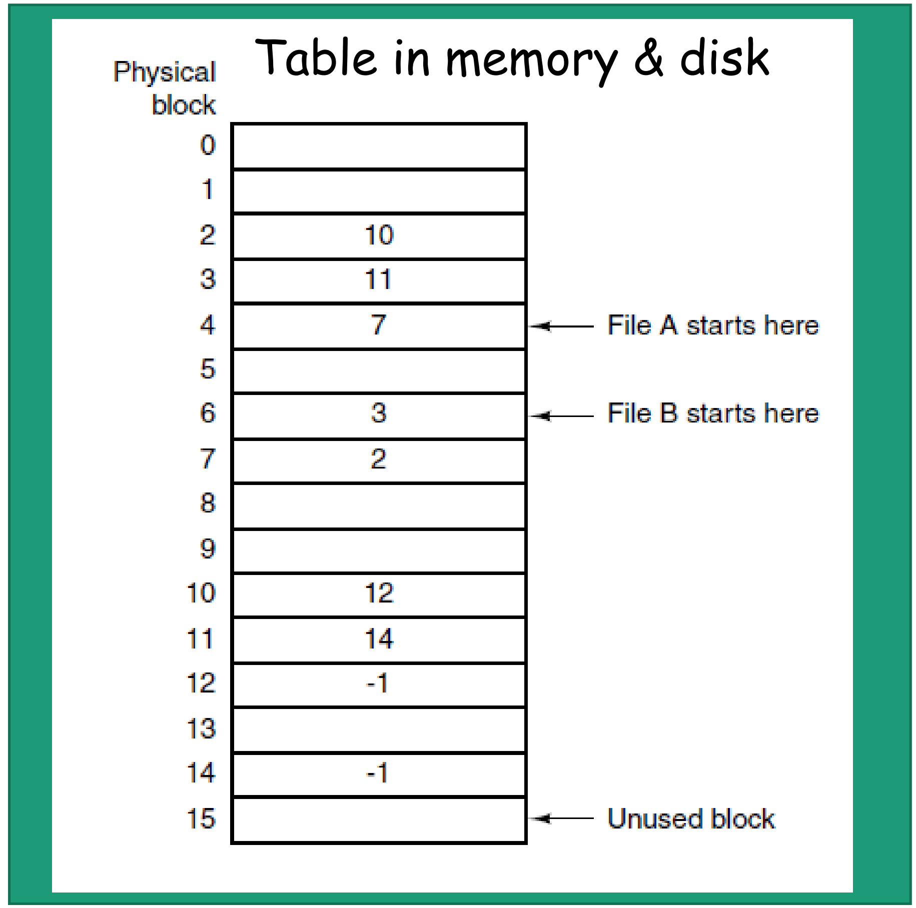

- [Figure 4-11 in Tanenbaum & Bos, 2014]

# Linked Allocation: Discussion

- No external fragmentation, all blocks can be allocated to files
- Two access types
  - Sequential access? Random access?
- Pointers occupies space
  - If a pointer requires 4 bytes (maximally, about 530 GB disk) and a block is 512 bytes, 4/512 = 0.78%
  - Clusters: use large allocation units
    - Cluster: logical block, logical-to-physical block mapping
  - Reliability: lost head link or tail link?

# Linked Allocation: FAT

- FAT = File Allocation Table
- Linked list as a table
  - Each entry, a block
  - Stored in disk
  - Loaded to memory
    - Random access is to follow the linked list in memory



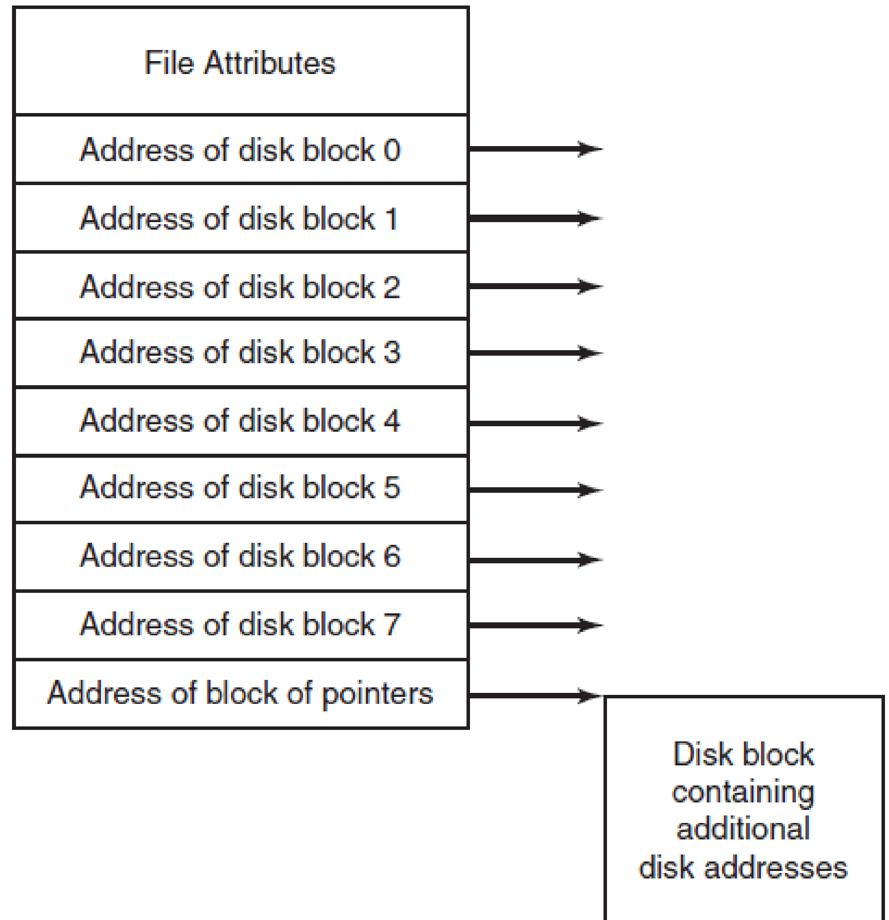

- [Figure 4-12 in Tanenbaum & Bos, 2014]

# Indexed Allocation

- Indexed block
  - contains pointer to each block in a file
  - An array of disk block numbers
- Each file has its own indexed block
- Pointer overhead is greater than the linked allocation
  - How do we make index block smaller?
    - Linked scheme
    - Multilevel scheme
    - Combined schem

# Index-Node (i-node)

- A linked & multi-leveled index allocation
- Loaded to memory when the file is "open"



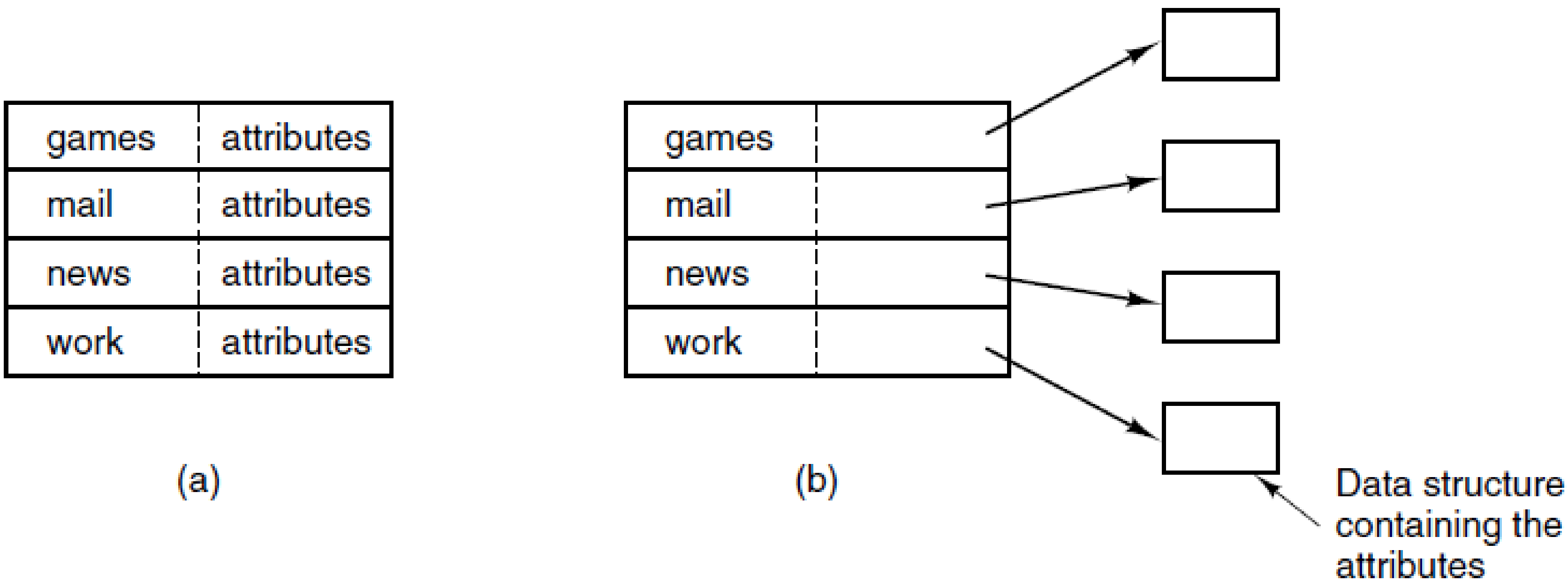

- [Figure 4-13 in Tanenbaum & Bos, 2014]

# Questions

- Free space allocation
  - Contiguous
  - Linked
  - Indexed

# Directories

- Fixed and linked

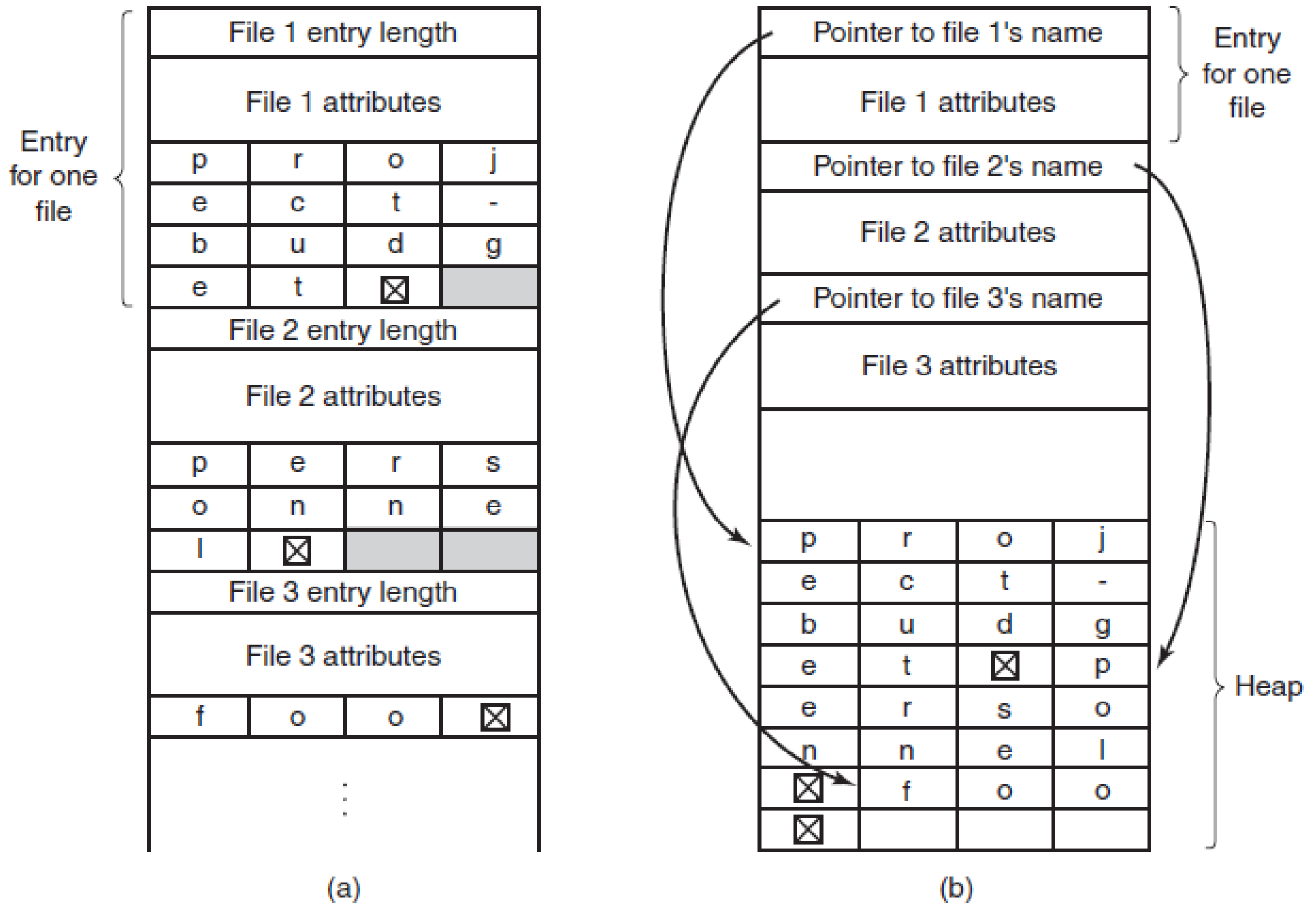



- [Figure 4-14 in Tanenbaum & Bos, 2014]

# Long and Variable Length Filenames

- Filenames stored in directory data structure
- Fixed length
  - Set up a limit (e.g., 255)
    - Wasting space, cannot have even longer names
- Variable length
  - Inline and heap
    - Inline: external fragmentation
    - Heap: need space management for names
- Speed up search by filenames?

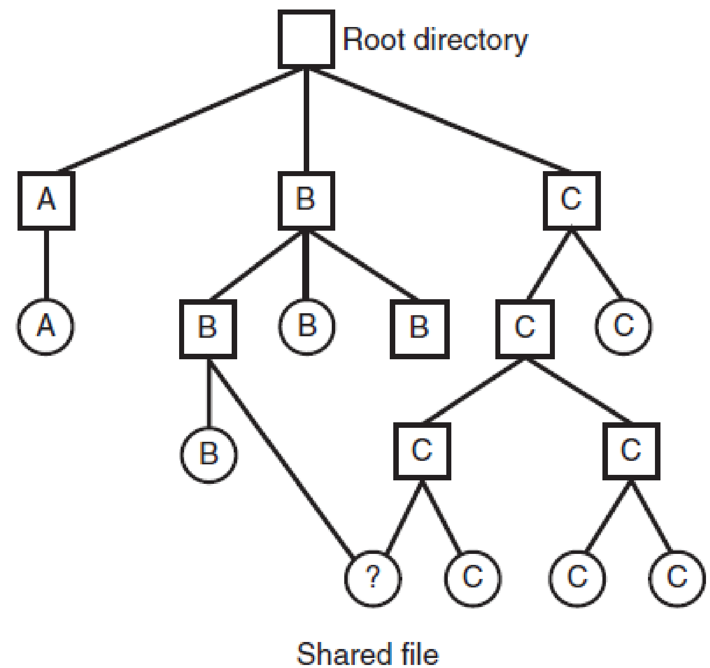
# Long Filenames: Example




- [Figure 4-15 in Tanenbaum & Bos, 2014]

# Shared Files

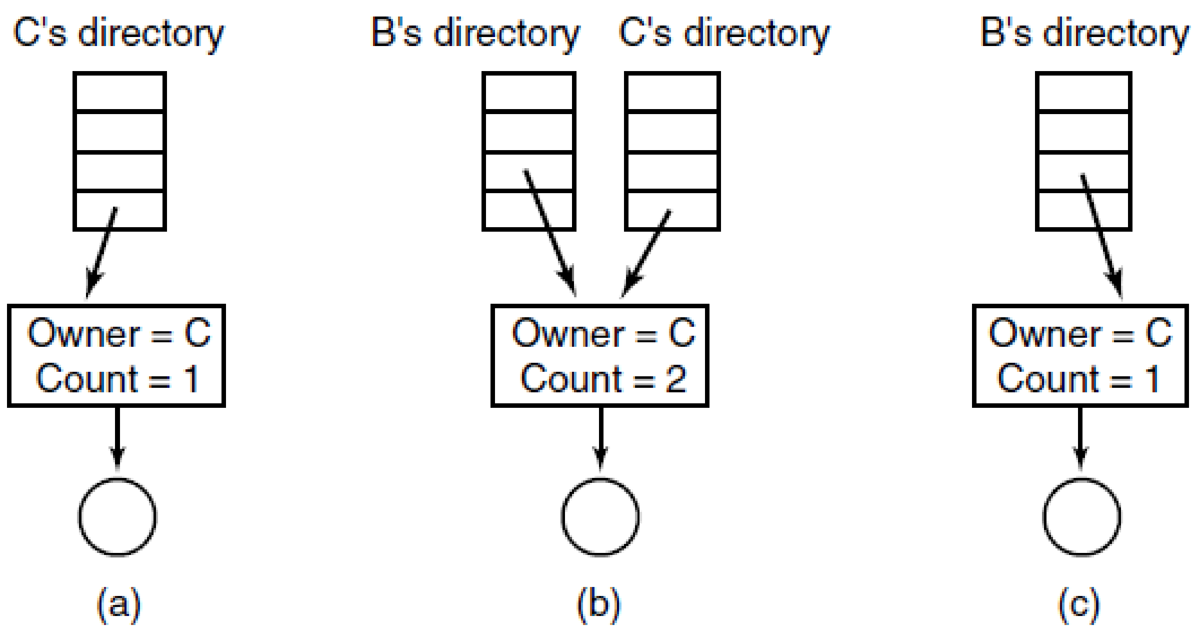
- A file can be "shared" in multiple directories




- [Figure 4-16 in Tanenbaum & Bos, 2014]

# Shared Files: Links

- (a) prior to linking; (b) created; (c) removed




- [Figure 4-17 in Tanenbaum & Bos, 2014]

# Questions?

- Design and implementing directories
  - Fixed and linked
- Long and variable length filenames
- Shared files

# Free Space Management

- File size distribution
  - How big should an allocation unit to be?
- Tracking free space

# File Size Distribution: Empirical Results

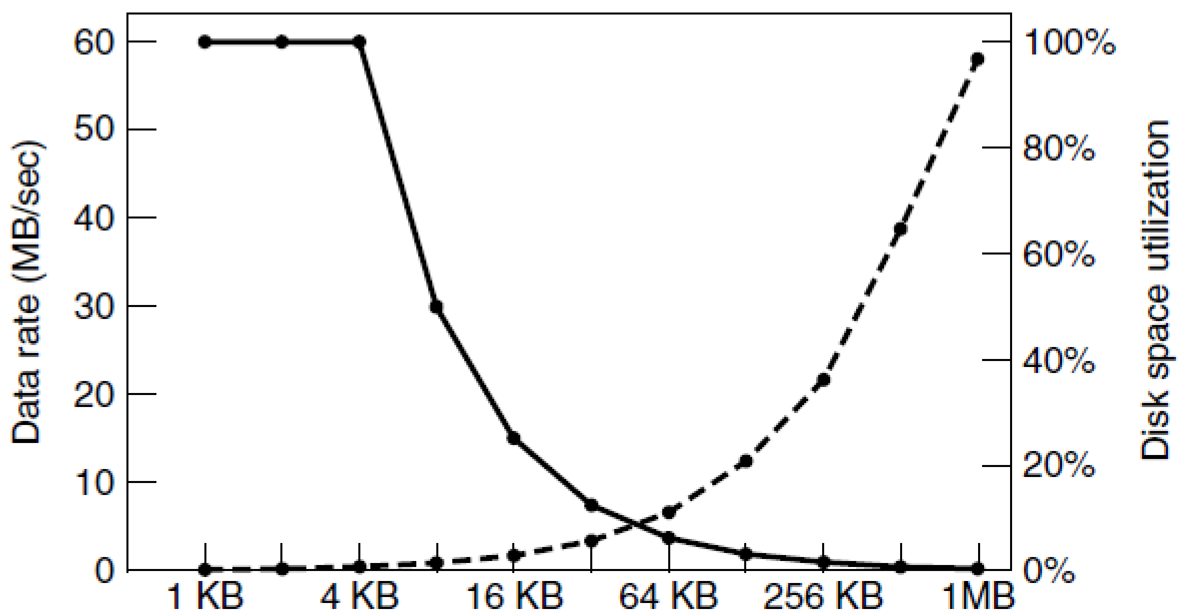
| Length | VU 1984 | VU 2005 | Web |
|---|---|---|---|
| 1 | 1.79 | 1.38 | 6.67 |
| 2 | 1.88 | 1.53 | 7.67 |
| 4 | 2.01 | 1.65 | 8.33 |
| 8 | 2.31 | 1.80 | 11.30 |
| 16 | 3.32 | 2.15 | 11.46 |
| 32 | 5.13 | 3.15 | 12.33 |
| 64 | 8.71 | 4.98 | 26.10 |
| 128 | 14.73 | 8.03 | 28.49 |
| 256 | 23.09 | 13.29 | 32.10 |
| 512 | 34.44 | 20.62 | 39.94 |
| 1 KB | 48.05 | 30.91 | 47.82 |
| 2 KB | 60.87 | 46.09 | 59.44 |
| 4 KB | 75.31 | 59.13 | 70.64 |
| 8 KB | 84.97 | 69.96 | 79.69 |

| Length | VU 1984 | VU 2005 | Web |
|---|---|---|---|
| 16 KB | 92.53 | 78.92 | 86.79 |
| 32 KB | 97.21 | 85.87 | 91.65 |
| 64 KB | 99.18 | 90.84 | 94.80 |
| 128 KB | 99.84 | 93.73 | 96.93 |
| 256 KB | 99.96 | 96.12 | 98.48 |
| 512 KB | 100.00 | 97.73 | 98.99 |
| 1 MB | 100.00 | 98.87 | 99.62 |
| 2 MB | 100.00 | 99.44 | 99.80 |
| 4 MB | 100.00 | 99.71 | 99.87 |
| 8 MB | 100.00 | 99.86 | 99.94 |
| 16 MB | 100.00 | 99.94 | 99.97 |
| 32 MB | 100.00 | 99.97 | 99.99 |
| 64 MB | 100.00 | 99.99 | 99.99 |
| 128 MB | 100.00 | 99.99 | 100.00 |

- [Figure 4-20 in Tanenbaum & Bos, 2014]

# Efficiency & Performance

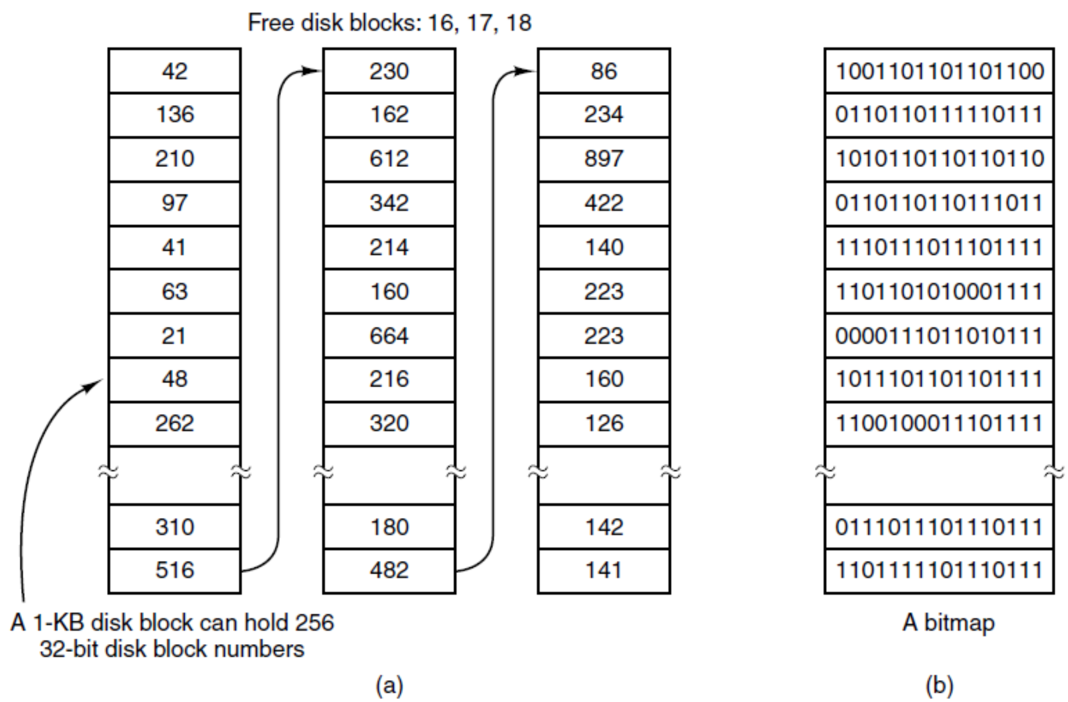
- Dashed: the data rate of a disk; Solid the disk space efficiency. All files are 4 KB.




- [Figure 4-21 in Tanenbaum & Bos, 2014]

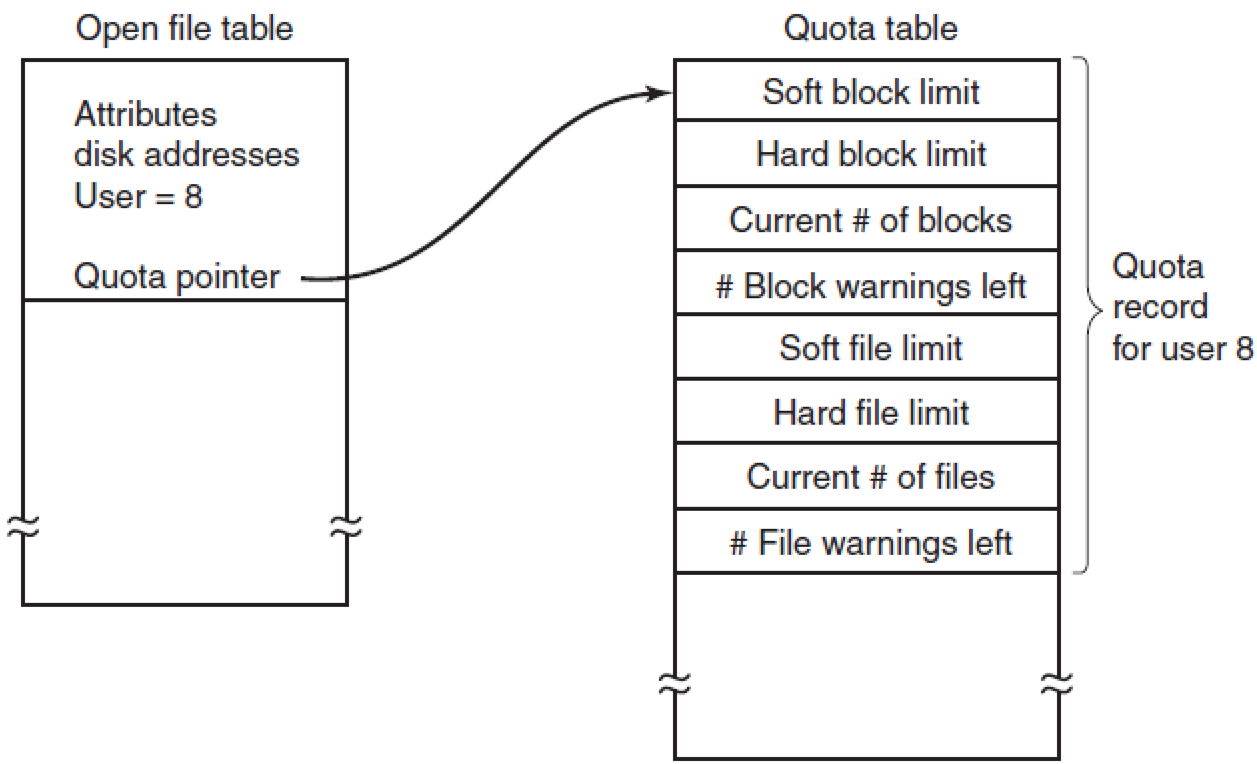
# Tracking Free Blocks

- Bit vector (bitmap) and linked list




- [Figure 4-22 in Tanenbaum & Bos, 2014]

# Disk Quotas




- [Figure 4-24 in Tanenbaum & Bos, 2014]

# Questions?

- Logical and physical block size: trade off between efficiency and performance
- Free space tracking
  - Bit vector (bitmap) and linked list
- Disk quotas

# Recovery, Backup, and Restore

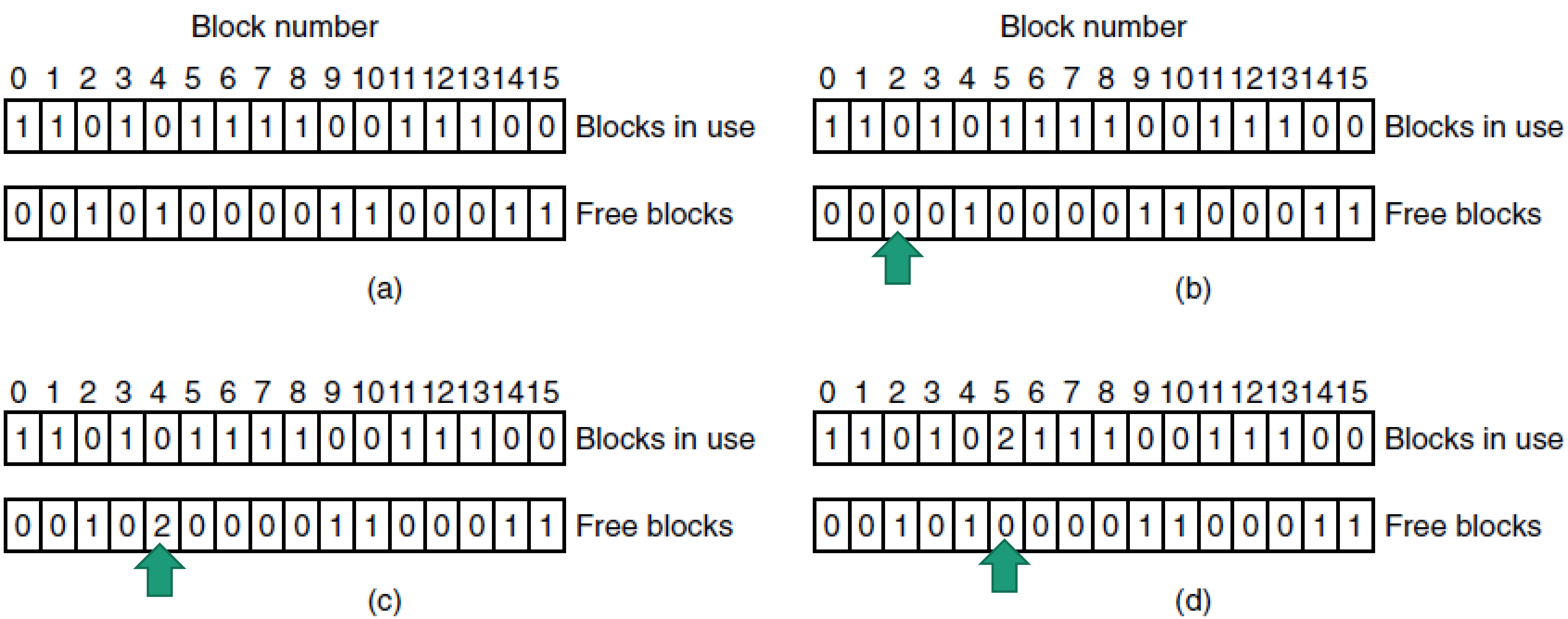
- Disaster may happen
  - Examples
    - Natural disaster
    - Power shortage
    - User's mistakes
- Recovery
  - Consistency checks, log-structured, and journaling
- Backup and restore

# Consistency Check

- Scan of all metadata on each file system
- Operating systems often have programs to conduct a consistency check
  - Examples
    - Unix-like systems: fsck
    - Windows: chkdsk

# Consistency Check: Example

- (a) consistent, (b) missing blocks, (c) duplicate blocks in free list, (d) duplicate data block




- [Figure 4-27 in Tanenbaum & Bos, 2014]

# Journaling File Systems

- All metadata changes are written sequentially to a log
- Each set of operations for performance a specific disk task is a transaction
- The operating of a transaction is written to the log, the transaction is committed
- Each step in the committed transaction is then carried out, and update the log for progress
- When a transaction is completed, it is removed from the log

# Transaction: An Example

- Steps to remove a file in Unix-like OSes:

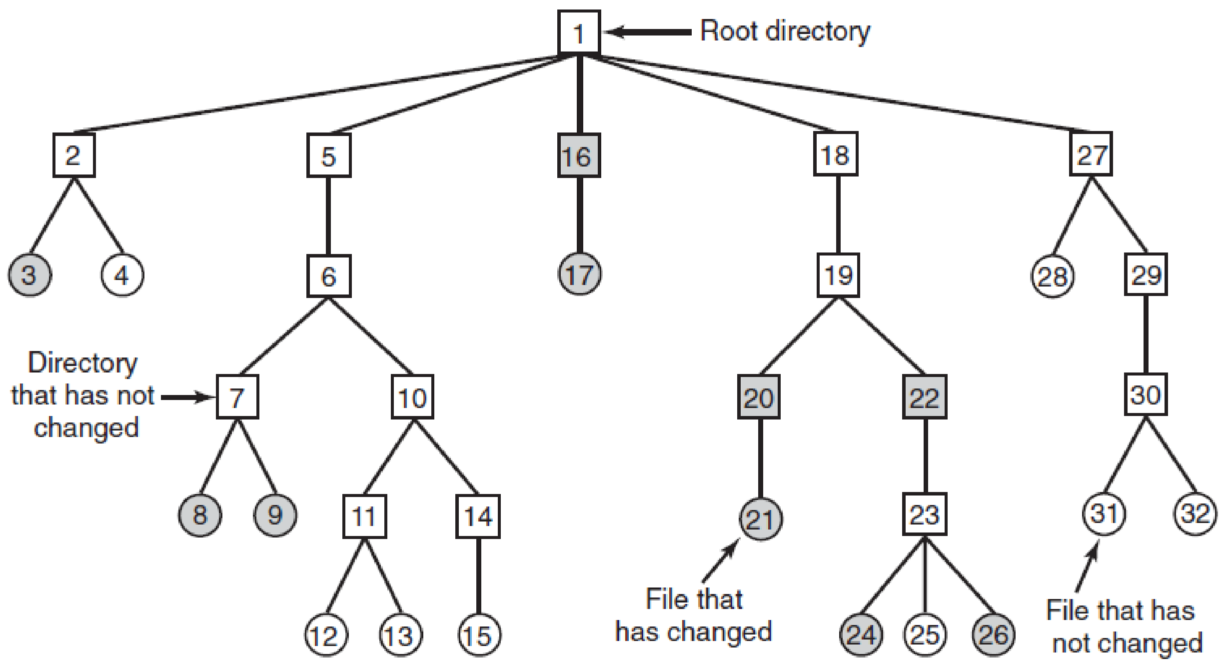
1. Remove file from its directory.
2. Release i-node to the pool of free i-nodes.
3. Return all disk blocks to pool of free disk blocks.

# Back-up

- Backups to tape are generally made to handle one of two potential problems:
    1. Recover from disaster.
    2. Recover from stupidity.
- Backup is slow, how to do it efficiently?
    - Full backup
    - Incremental backup
        - Tracking modified

# Logical dump: Example

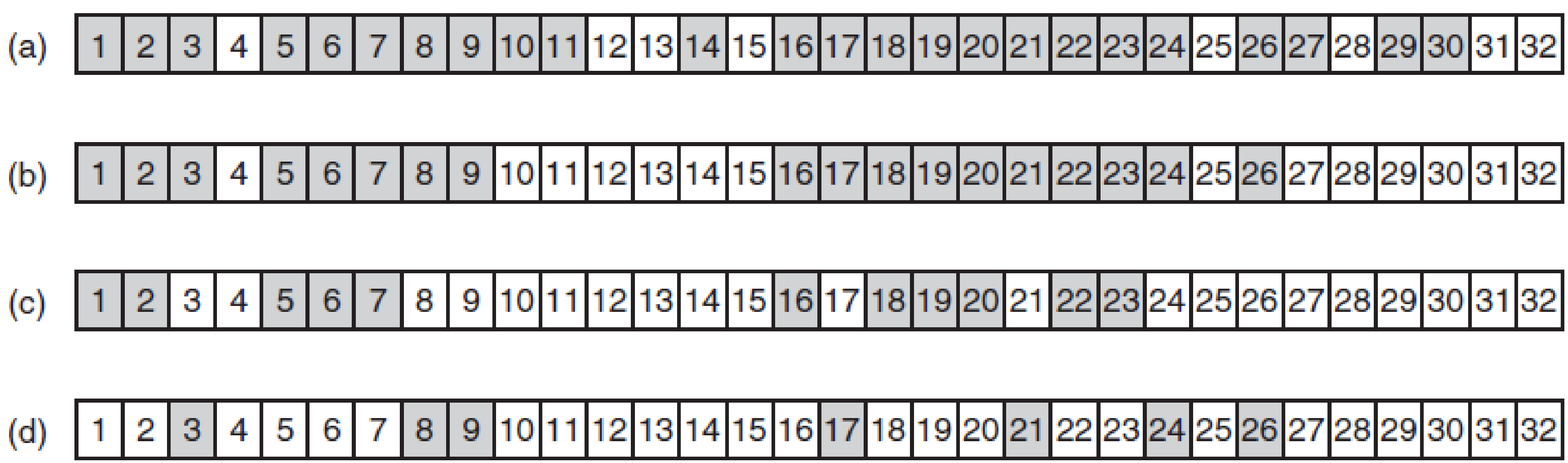
- An incremental backup, dump only changed files and directories




- [Figure 4-25 in Tanenbaum & Bos, 2014]

# Logical Dump: Example Algorithm

- Bitmap: (a) mark all directories and all modified files, (b) unmark directories that have no modified files, (c) dumping directories, (d) dumping files.




- [Figure 4-26 in Tanenbaum & Bos, 2014]

# Questions?

- Recovery
  - Consistency check
  - Journaling file systems
    - Transaction
    - Log replay
- Restore
  - Backup
  - Restore (starting with an empty file system)

# Performance

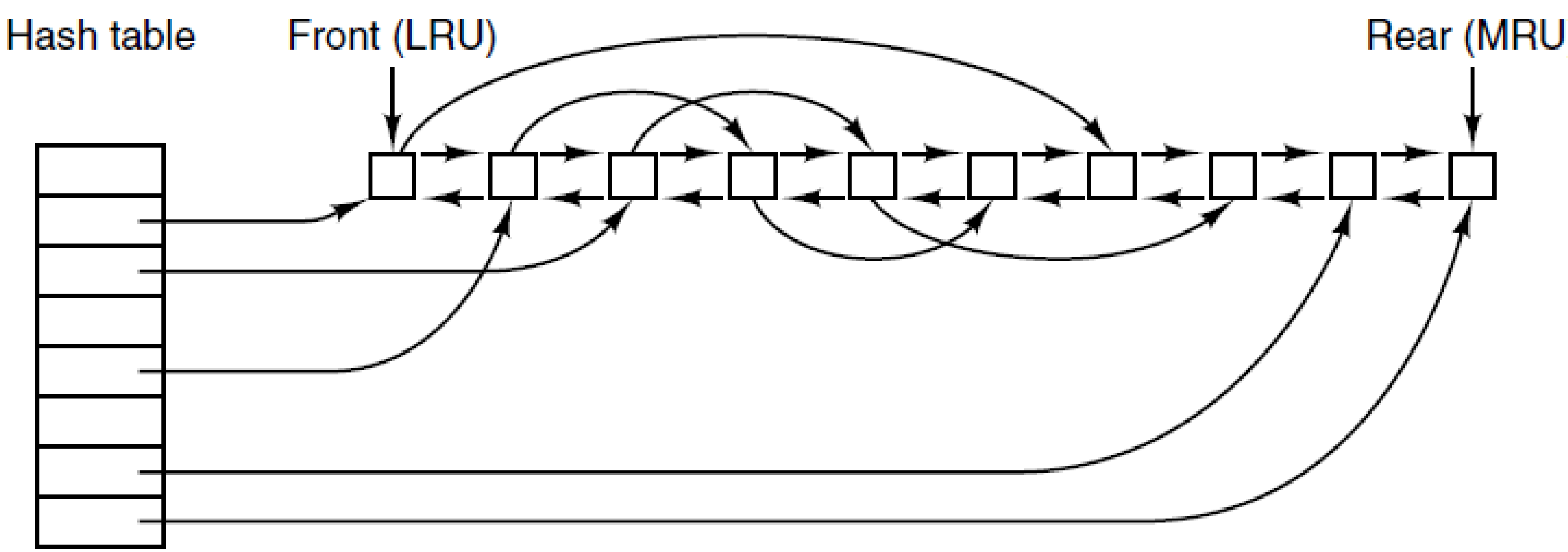
- Caching
- Block read ahead
- Device specific
    - Disk arm

# Block Cache (Buffer Cache)

- Access to disk is much slower than access to memory
- A collection of blocks kept logically in memory
  - Locality

# Caching: Example

- Create a hash table, each entry is the hash of the device and the block
- If a block in cache, read it from the cache




- [Figure 4-28 in Tanenbaum & Bos, 2014]

# Block Replacement

- Cache memory may be full, select a block in the cache and eject it
- Similar to paging
  - FIFO
  - Second change
  - LRU

# Performance and Consistency Consideration

- Some blocks rarely referenced two times within a short interval.
- Leads to a modified LRU scheme, taking two factors into account:
    1. Is the block likely to be needed again soon?
    2. Is the block essential to the consistency of the file system?

# Preventing Data Loss and Inconsistency

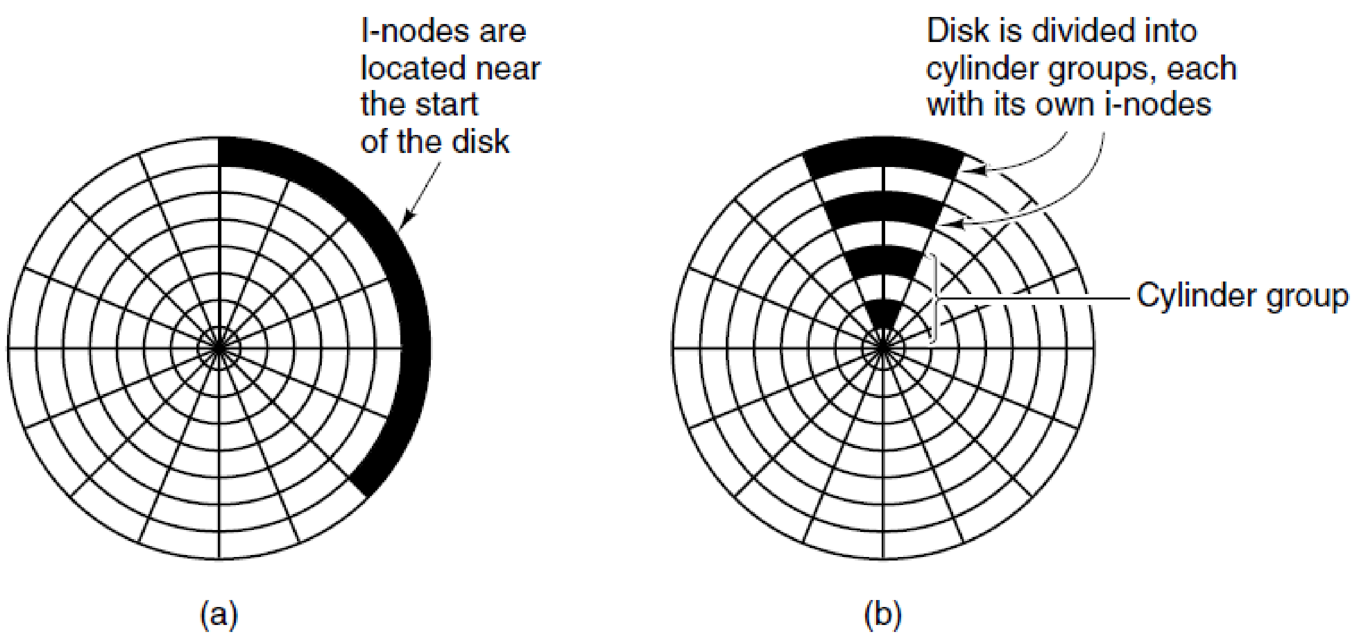
- Sync file systems with cache periodically
  - Example: Unix
- Write-through cache
  - Only read is from cache
  - Write will be done in both cache and disk (write to disk immediately)
- Example: if it is a USB drive, what do you choose and why?

# Block Read Ahead

- Read blocks into cache before they are needed
- Future read will be in cache
- Example
  - Considering that most file access are sequentially, when read block k, file system will also read k+1 to cache if it isn't in the cache

# Reduce Disk Arm Motion

- If it is hard drives …
- Location of i-nodes?




- [Figure 4-29 in Tanenbaum & Bos, 2014]

# Questions

- Performance improvement
  - Block cache
  - Block read ahead
  - Reduce disk arm motion
    - How about SSD?