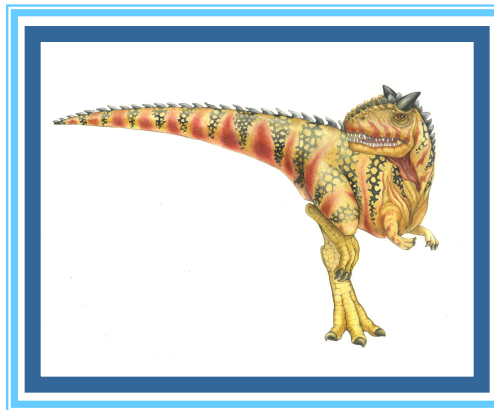


Chapter 14: File System Implementation





Outline

- File-System Structure
- File-System Operations
- Directory Implementation
- Allocation Methods
- Free-Space Management





Objectives

- Describe the details of implementing local file systems and directory structures
- Discuss block allocation and free-block algorithms and trade-offs





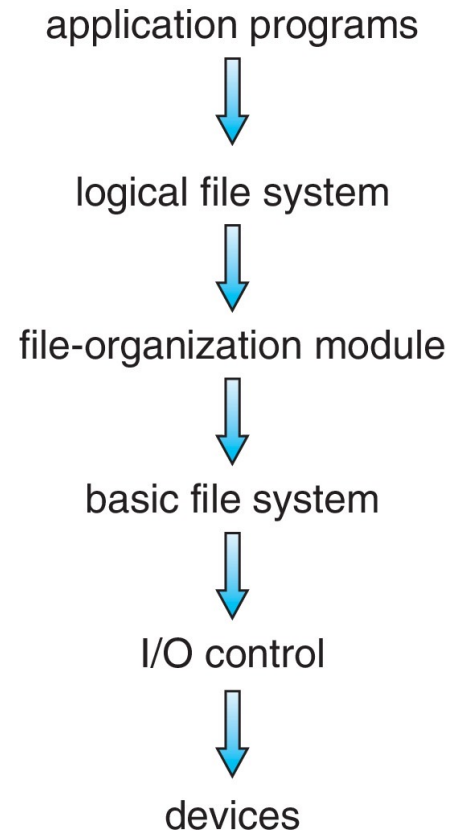
File-System Structure

- File structure
 - Logical storage unit
 - Collection of related information
- **File system** resides on secondary storage (disks)
 - Provided user interface to storage, mapping logical to physical
 - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
 - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block (FCB)** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers





Layered File System





File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
Given commands like
 read drive1, cylinder 72, track 2, sector 10, into memory location 1060
Outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like “retrieve block 123” translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
 - Buffers hold data in transit
 - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation





File System Layers (Cont.)

- **Logical file system** manages metadata information
 - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
 - Directory management
 - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
- Logical layers can be implemented by any coding method according to OS designer





File System Layers (Cont.)

- Many file systems, sometimes many within an operating system
 - Each with its own format:
 - CD-ROM is ISO 9660;
 - Unix has **UFS**, FFS;
 - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray,
 - Linux has more than 130 types, with **extended file system** ext3 and ext4 leading; plus distributed file systems, etc.)
 - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE





File-System Operations

- We have system calls at the API level, but how do we implement their functions?
 - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
 - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
 - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
 - Names and inode numbers, master file table





File Control Block (FCB)

- OS maintains **FCB** per file, which contains many details about the file
 - Typically, inode number, permissions, size, dates
 - Example

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks





In-Memory File System Structures

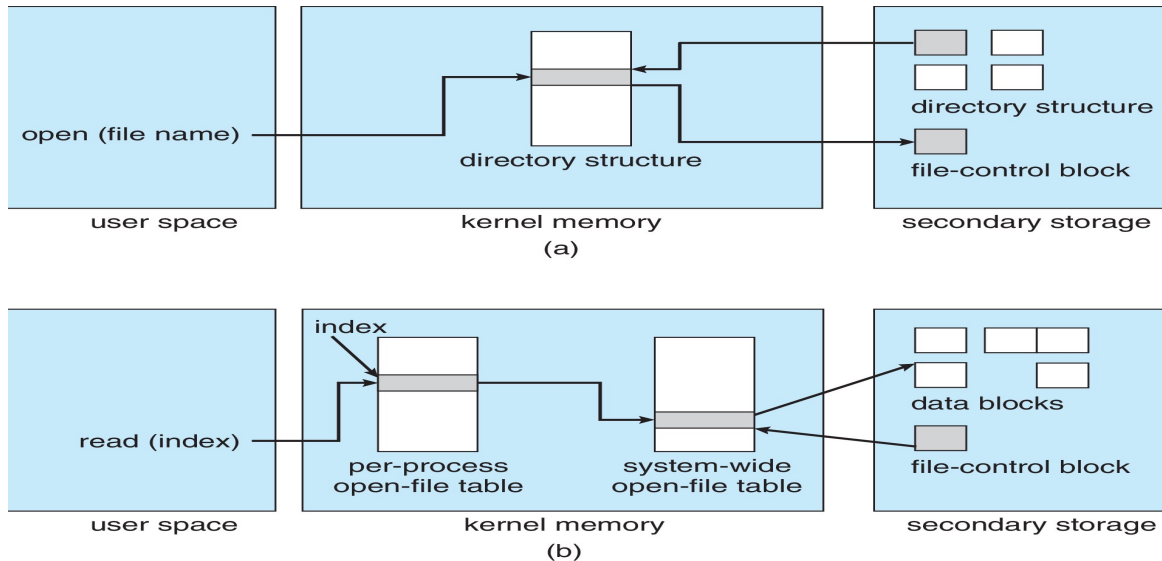
- **Mount table** storing file system mounts, mount points, file system types
- **System-wide open-file table** contains a copy of the FCB of each file and other info
- **Per-process open-file table** contains pointers to appropriate entries in system-wide open-file table as well as other info





In-Memory File System Structures (Cont.)

- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file





Directory Implementation

- **Linear list** of file names with pointer to the data blocks
 - Simple to program
 - Time-consuming to execute
 - ▶ Linear search time
 - ▶ Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
 - Decreases directory search time
 - **Collisions** – situations where two file names hash to the same location
 - Only good if entries are fixed size, or use chained-overflow method





Allocation Method

- An allocation method refers to how disk blocks are allocated for files:
 - Contiguous
 - Linked
 - Indexed





Contiguous Allocation Method

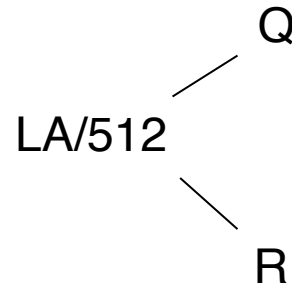
- An allocation method refers to how disk blocks are allocated for files:
- Each file occupies set of contiguous blocks
 - Best performance in most cases
 - Simple – only starting location (block #) and length (number of blocks) are required
 - Problems include:
 - ▶ Finding space on the disk for a file,
 - ▶ Knowing file size,
 - ▶ External fragmentation, need for **compaction off-line** (**downtime**) or **on-line**



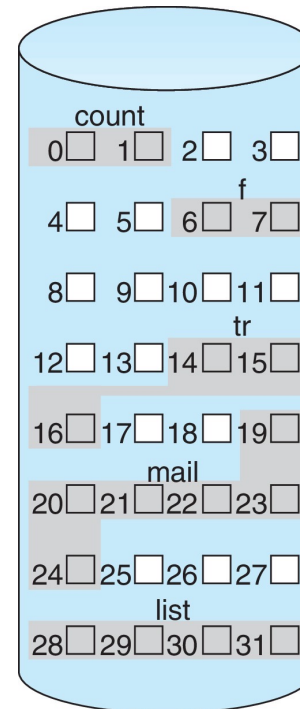


Contiguous Allocation (Cont.)

- Mapping from logical to physical
(block size = 512 bytes)



- Block to be accessed = starting address + Q
- Displacement into block = R



directory

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





Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
 - Extents are allocated for file allocation
 - A file consists of one or more extents





Linked Allocation

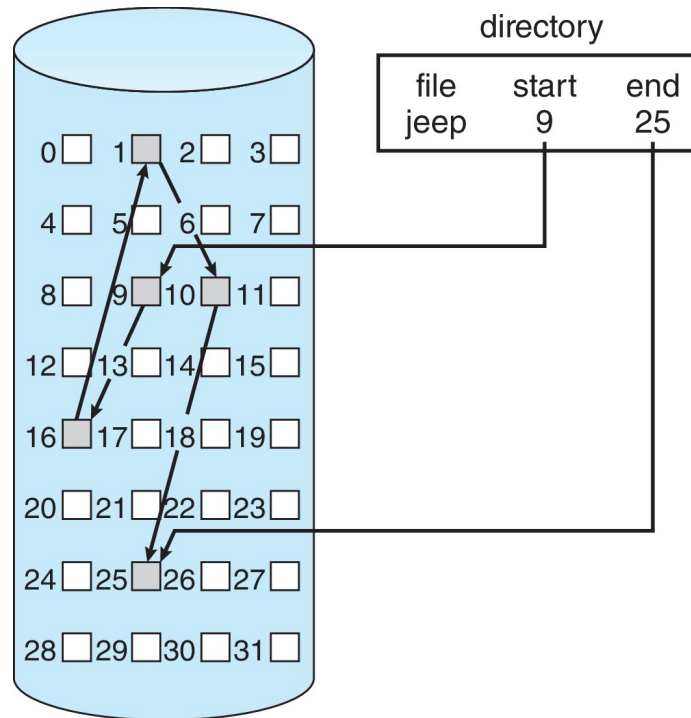
- Each file is a linked list of blocks
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks





Linked Allocation Example

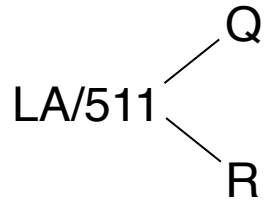
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk
- Scheme





Linked Allocation (Cont.)

- Mapping



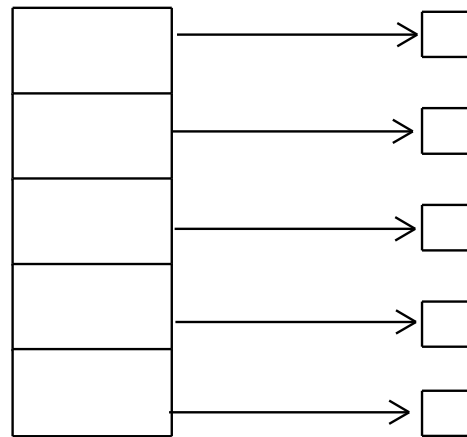
- Block to be accessed is the Q^{th} block in the linked chain of blocks representing the file.
- Displacement into block = $R + 1$





Indexed Allocation Method

- Each file has its own **index block**(s) of pointers to its data blocks
- Logical view

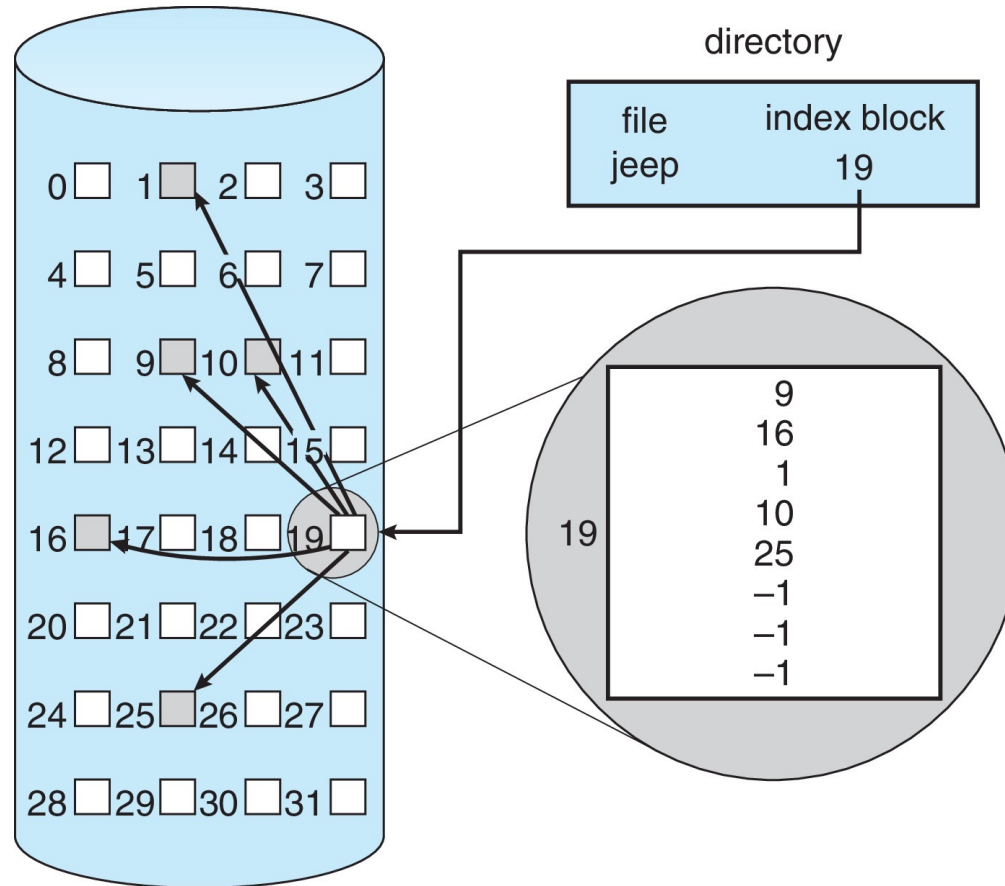


index table





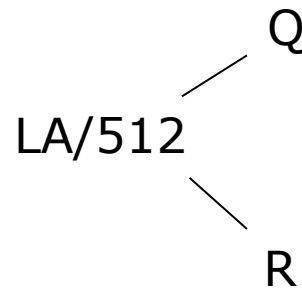
Example of Indexed Allocation





Indexed Allocation – Small Files

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table



- Calculation:
 - Q = displacement into index table
 - R = displacement into block





Indexed Allocation – Large Files

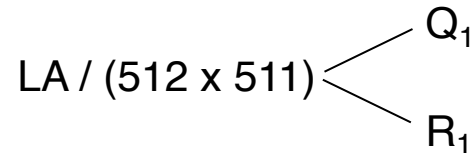
- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
 - Linked scheme – Link blocks of index table (no limit on size)
 - Multi-level indexing



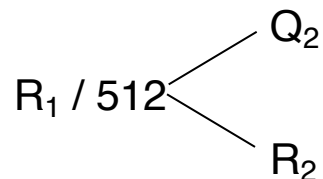


Indexed Allocation – Linked Scheme

- Link blocks of index table (no limit on size)



- Outer-level mapping scheme
 - Q_1 = block of index table
 - R_1 is used as follows



- Inner-level mapping scheme
 - ▶ Q_2 = displacement into block of index table
 - ▶ R_2 displacement into block of file





Indexed Allocation – Two-level Scheme

- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)

$$LA / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

- Mapping scheme for outer-index:
 - Q_1 = displacement into outer-index
 - R_1 is used as follows:

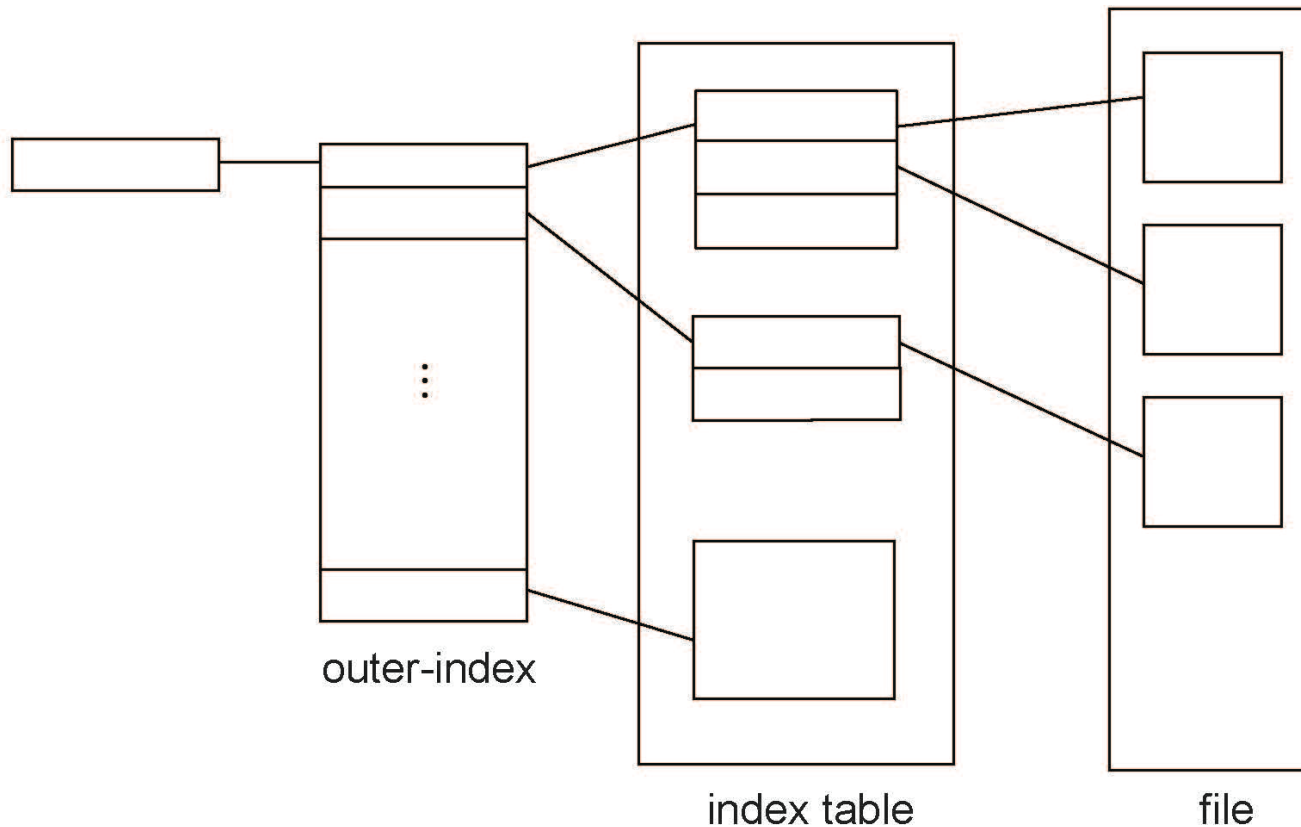
$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

- Mapping scheme for index level:
 - Q_2 = displacement into block of index table
 - R_2 displacement into block of file





Indexed Allocation – Two-Level Scheme





Performance

- Best method depends on file access type
 - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation
 - Select either contiguous or linked
- Indexed more complex
 - Single block access could require 2 index block reads then data block read
 - Clustering can help improve throughput, reduce CPU overhead





- Diagram illustrating a bit vector structure. The vector consists of bits indexed from 0 to $n-1$. The first three bits are labeled 0, 1, and 2, and the last bit is labeled $n-1$. An ellipsis (...) indicates intermediate bits. Below the vector, a definition for $\text{bit}[i]$ is provided:

$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

$$\begin{aligned} & (\text{number of bits per word}) * \\ & (\text{number of 0-value words}) + \\ & \text{offset of first 1 bit} \end{aligned}$$

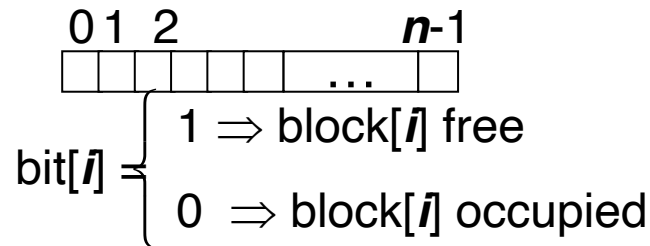
bit

A detailed illustration of a Spinosaurus dinosaur, showing its long, yellowish-brown body, a large, sail-like structure on its back, and a long, thin tail. It is depicted in a walking pose, facing right.



Free-Space Management

- File system maintains **free-space list** to track available blocks
- **Bit vector** or **bit map** (n blocks)



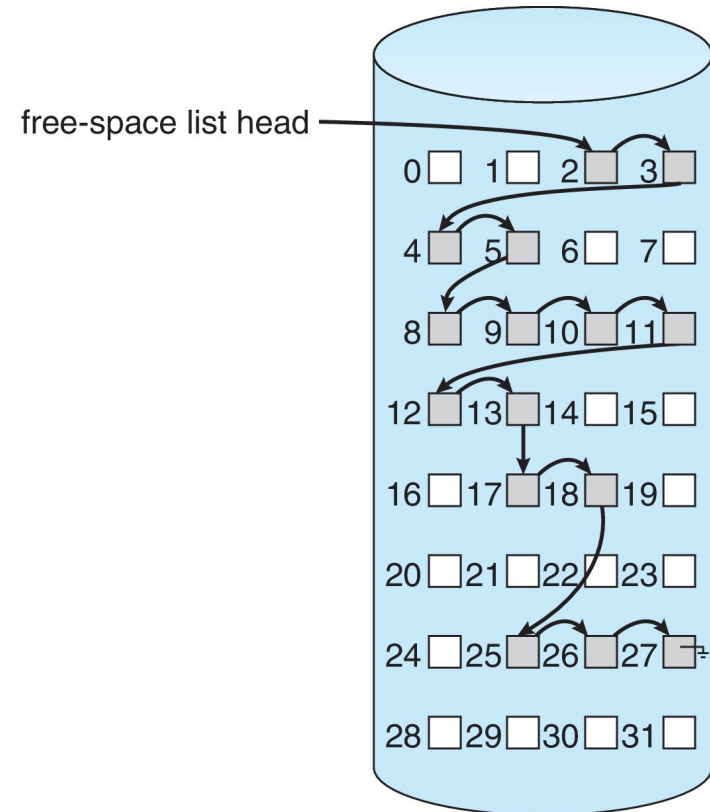
- Bit map requires extra space
 - Example:
 - block size = 4KB = 2^{12} bytes
 - disk size = 2^{40} bytes (1 terabyte)
 - $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)
 - if clusters of 4 blocks \rightarrow 8MB of memory
- Easy to get contiguous files





Linked Free Space List on Disk

- Linked list (free list)
 - Cannot get contiguous space easily
 - No waste. Linked Free Space List on Disk of space
 - No need to traverse the entire list (if # free blocks recorded)





Free-Space Management (Cont.)

- 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





Free-Space Management (Cont.)

- Space Maps
 - Used in **ZFS**
 - Consider meta-data I/O on very large file systems
 - ▶ Full data structures like bit maps cannot fit in memory → thousands of I/Os
 - Divides device space into **metaslab** units and manages metaslabs
 - ▶ Given volume can contain hundreds of metaslabs
 - Each metaslab has associated space map
 - ▶ Uses counting algorithm
 - But records to log file rather than file system
 - ▶ Log of all block activity, in time order, in counting format
 - Metaslab activity → load space map into memory in balanced-tree structure, indexed by offset
 - ▶ Replay log into that structure
 - ▶ Combine contiguous free blocks into single entry



End of Chapter

