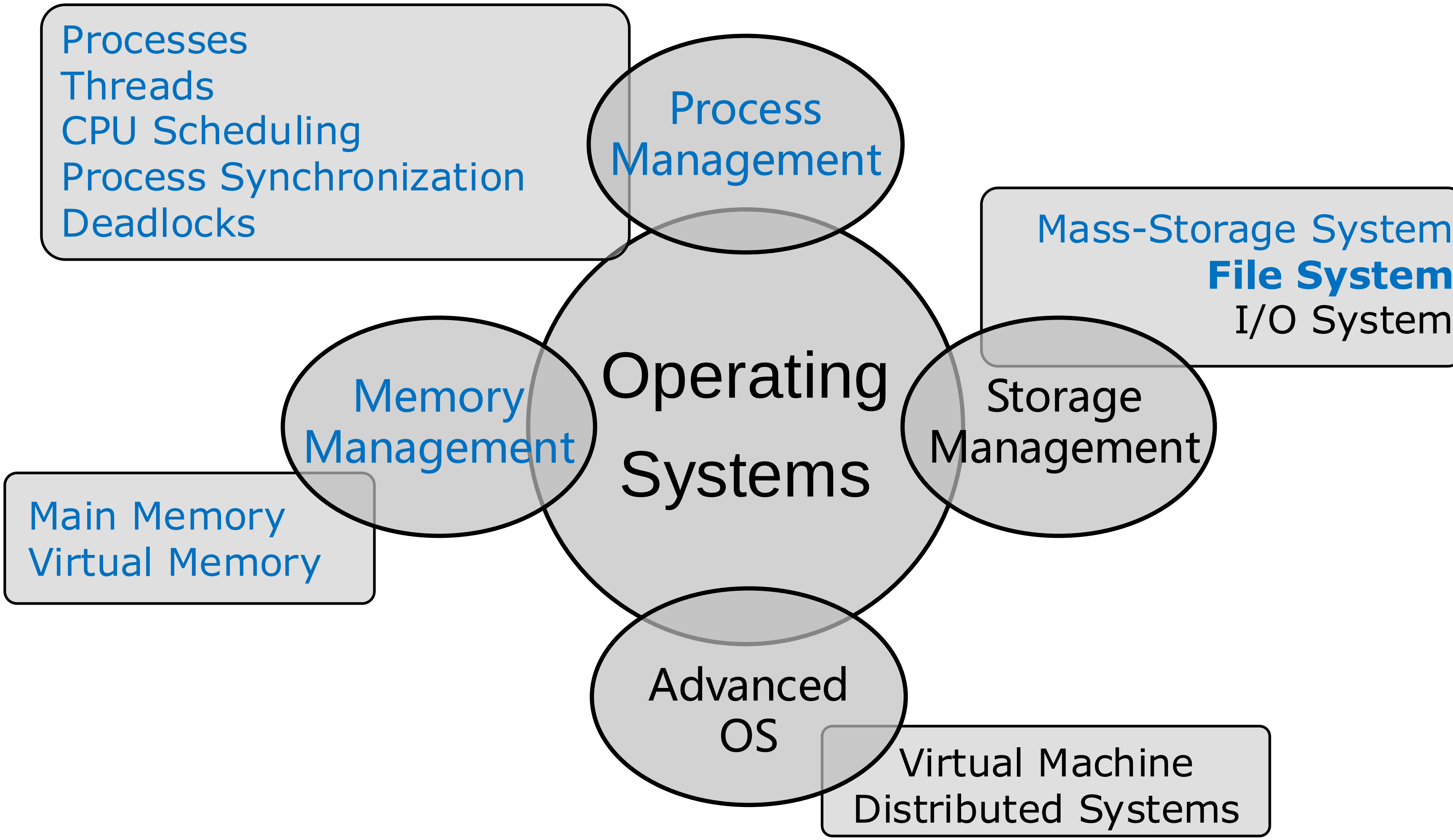
File System

Shengzhong Liu
Department of Computer Science and Engineering
Shanghai Jiao Tong University

Operating System Topics



Outline

- File Concepts
- Access Methods
- Directory Structures
- File System Structures and Operations
- Partition and Mounting
- Allocation Methods
- Free Space Management
- Virtual and Remote File Systems

File Concepts

File Concept

- A file is a named collection of related information that is recorded on secondary storage.

- Types:
  - Text
  - Source/object programs
  - Executable programs
  - Database records
  - Graphic images
  - Multimedia

File Types – Name, Extension

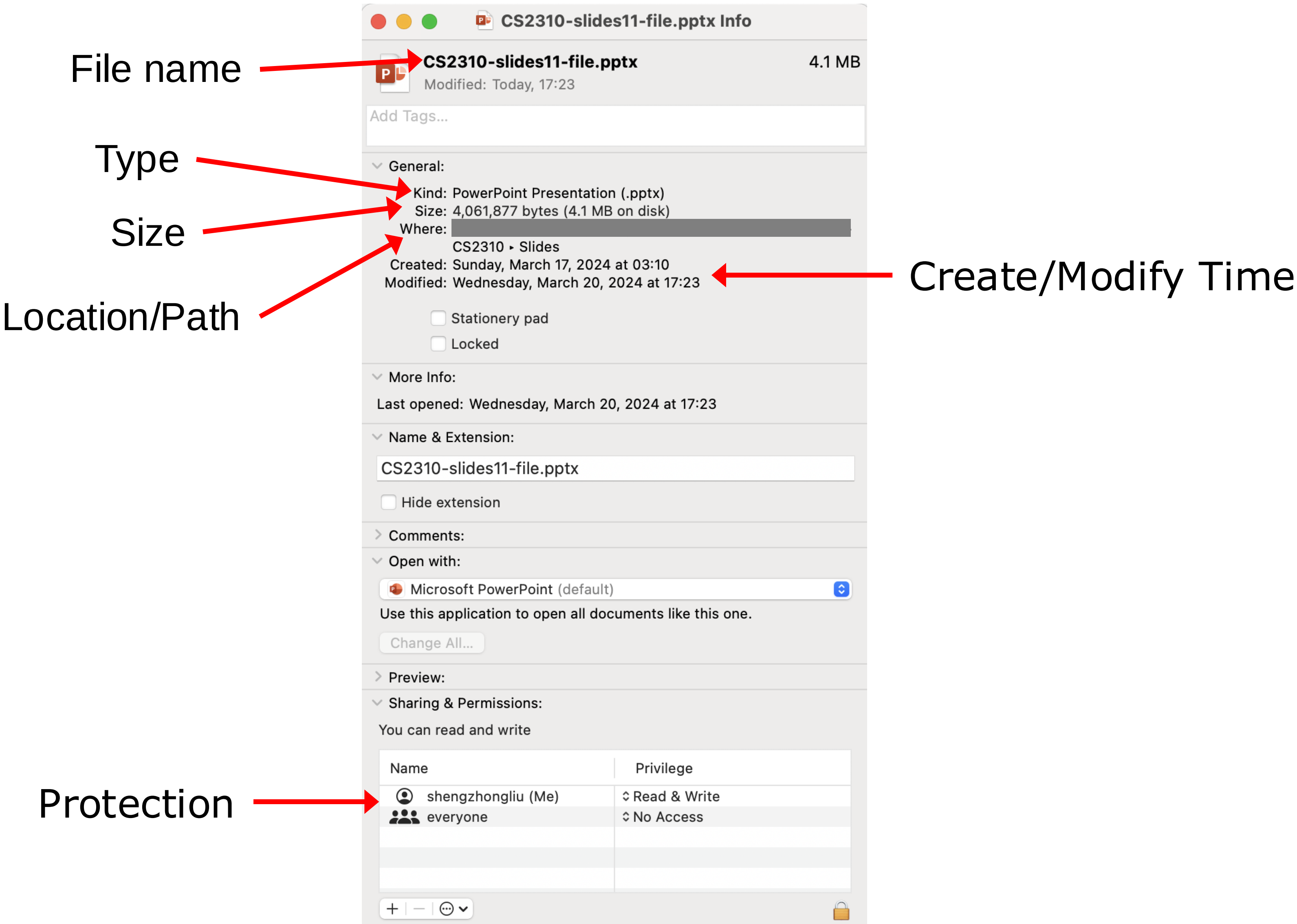
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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 Attributes

- **Name** – information kept in human-readable form (for convenience)
- **Identifier** – unique tag (number) identifies file within file system
    - Non-human-readable
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring

- Information about files are kept in the **directory structure (目录结构)**, which is maintained on the disk

File Info Window on Mac OS



File Structure

- **Option 1**: None - sequence of words, bytes
- **Option 2**: Simple record structure
  - Lines
  - Fixed length
  - Variable length
- **Option 3**: Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate the last two with the first method by inserting appropriate control characters

File Operations

- **Create:**
  - Allocate space for the file and create an entry in a directory.
- **Open:**
  - All operations except create and delete require a file open() first.
  - Return a file handle as an argument in other calls.
- **Write:**
  - Keep a **write pointer** to the file location where the **next sequential write** happens
- **Read:**
  - Keep a **read pointer** to the file location where the **next read** happens
- **Reposition within file:**
  - Current-file-position pointer of the open file is repositioned to a given value
- **Delete:**
  - Release file space and erase/mark-as-free the directory entry.
- **Truncate:**
  - Erase the contents of a file but keep its attributes

Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table:**
    - Tracks opened files
  - **File pointer**:
    - Pointer to last read/write location, per process that has the file open
  - **File-open count**:
    - Counter of the number of times a file is opened
    - Allow removal from open-file table when the last process closes it
  - **Disk location of the file**:
    - Data access information
  - **Access rights**:
    - Per-process access mode information

File Locking

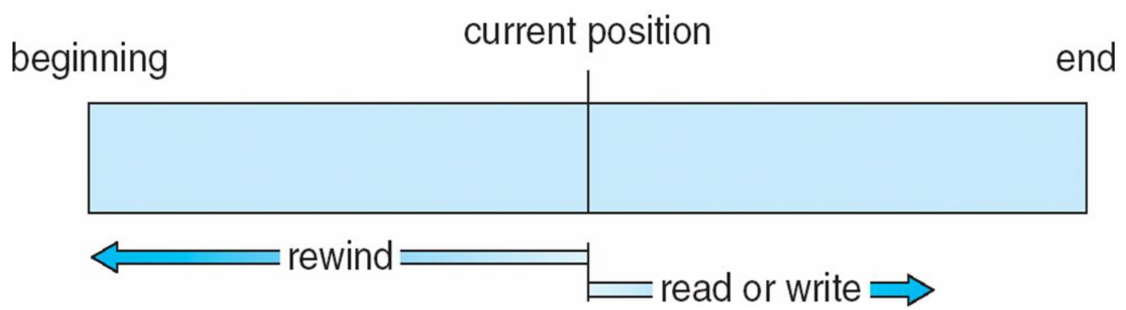
- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory file-locking mechanism:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

File Access Methods and Protection

Access Methods

- **Sequential Access**

  read_next()
  write_next()
  reset()



- **Direct Access**

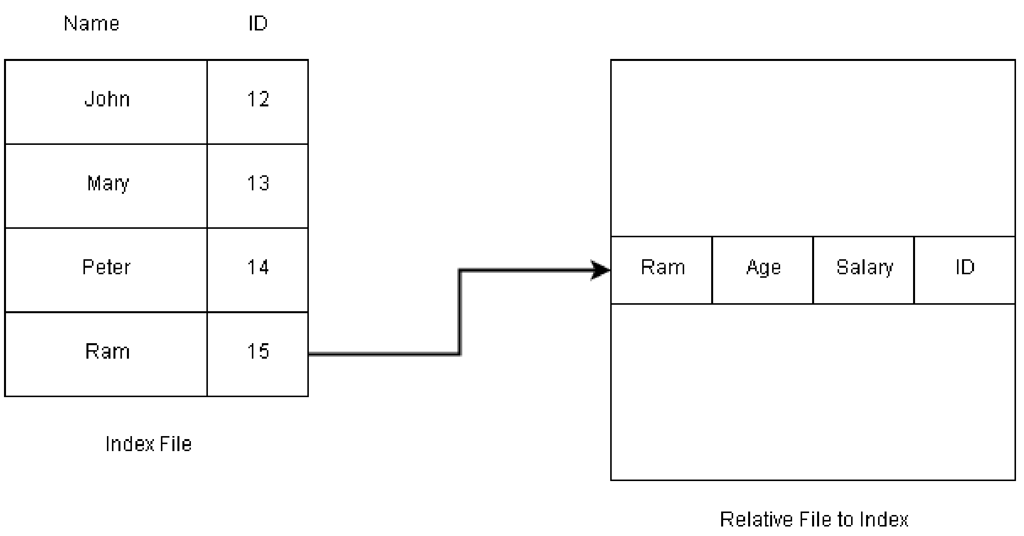
  read(n)
  write(n)

  position_file(n)
  read_next()
  write_next()
  rewrite(n)

n = relative block/byte number within the file

Other Access Methods

- Can be other access methods built on top of base methods
- Index-based access:
  - Generally involve creation of an **index** for the file
  - Keep index in memory for fast determination of location of data to be operated on
    - If the index is too large, create an in-memory index, which is an index of a disk index



Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom

- Recap: Types of access to file
  - **Read:** Read from the file
  - **Write:** Write or rewrite the file
  - **Execute:** Load the file into memory and execute it
  - **Append:** Write new information at the end of the file
  - **Delete:** Delete the file and free its space for possible reuse
  - **List:** List the name and attributes of the file
  - **Attribute change:** Changing the attributes of the file

Access Lists and Groups

- Mode of access: **`read`**, **`write`**, **`execute`**
- Three classes of users

			RWX
a) **owner access**	7	$\Rightarrow$	1 1 1
			RWX
b) **group access**	6	$\Rightarrow$	1 1 0
			RWX
c) **public access**	1	$\Rightarrow$	0 0 1

- Ask administrator to create a group (unique name), say **`G`**, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

`chmod 761 game`

Attach a group to a file

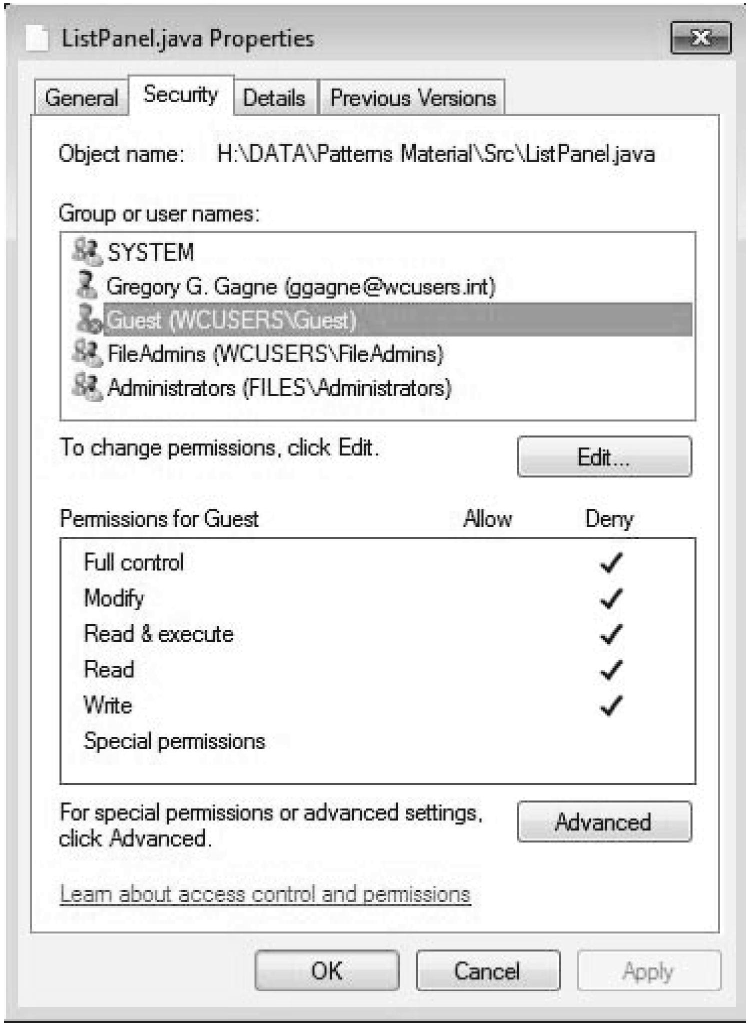
`chgrp G game`

A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

owner group public

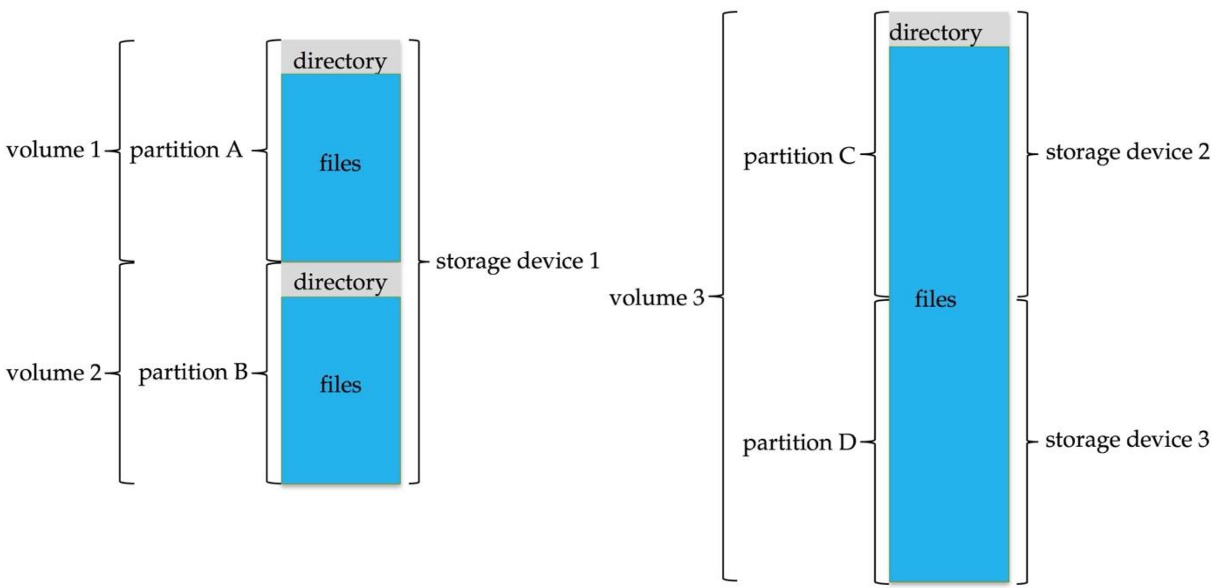
Windows 7 Access-Control List Management



Directory Structures

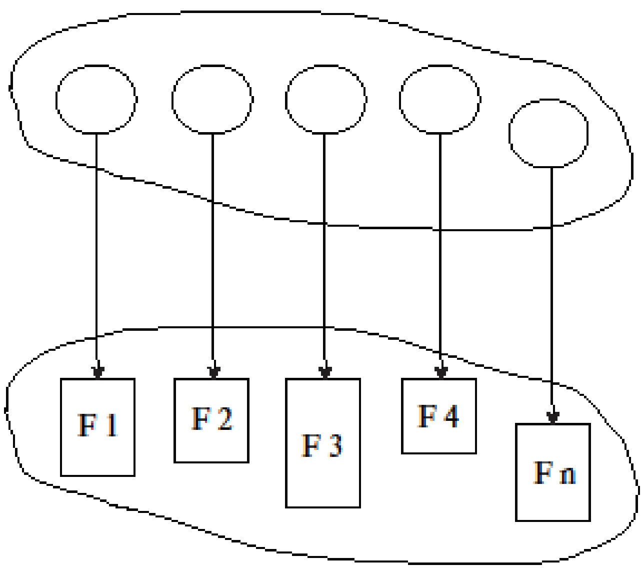
Disk Structure

- Disk can be subdivided into **partitions（分区）**
  - Disks or partitions can be **RAID** protected against failure
  - Disk or partition can be used **raw** without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system is known as a **volume**
  - Each volume containing a file system also tracks that file system's info in **device directory** or **volume table of contents**



Directory Structure

- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk

Operations Performed on Directory

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

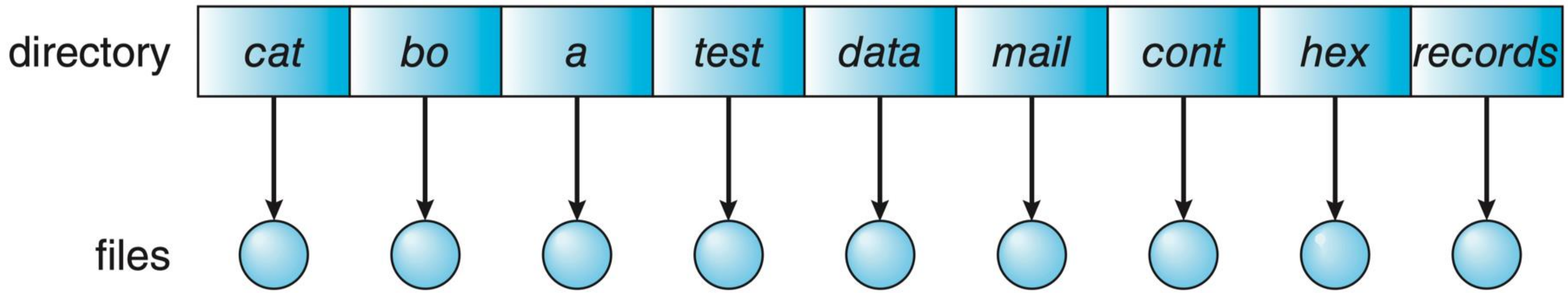
Directory Organization

The directory is organized logically to obtain

- **Efficiency** – locating a file quickly
- **Naming** – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

- A single directory for all users



Efficiency problem

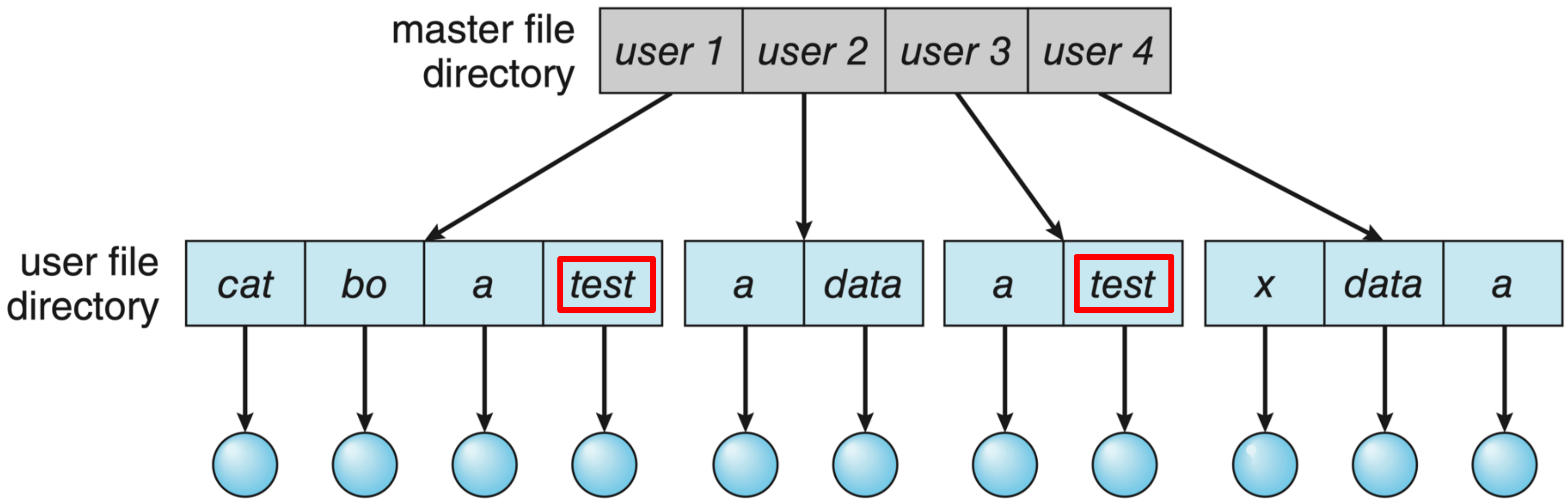
Naming problem

Protection of users' private files

Grouping problem

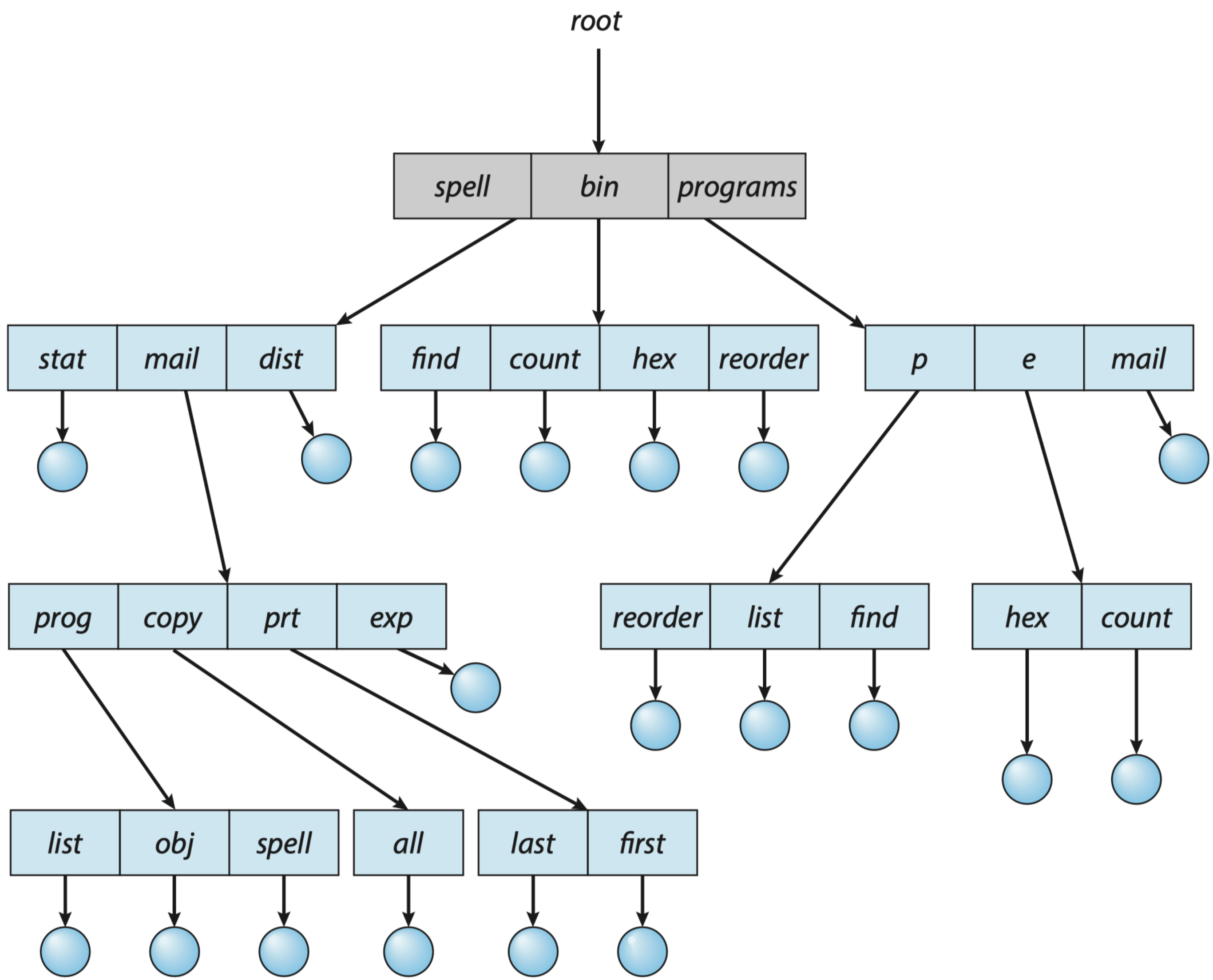
Two-Level Directory

- Separate directory for each user



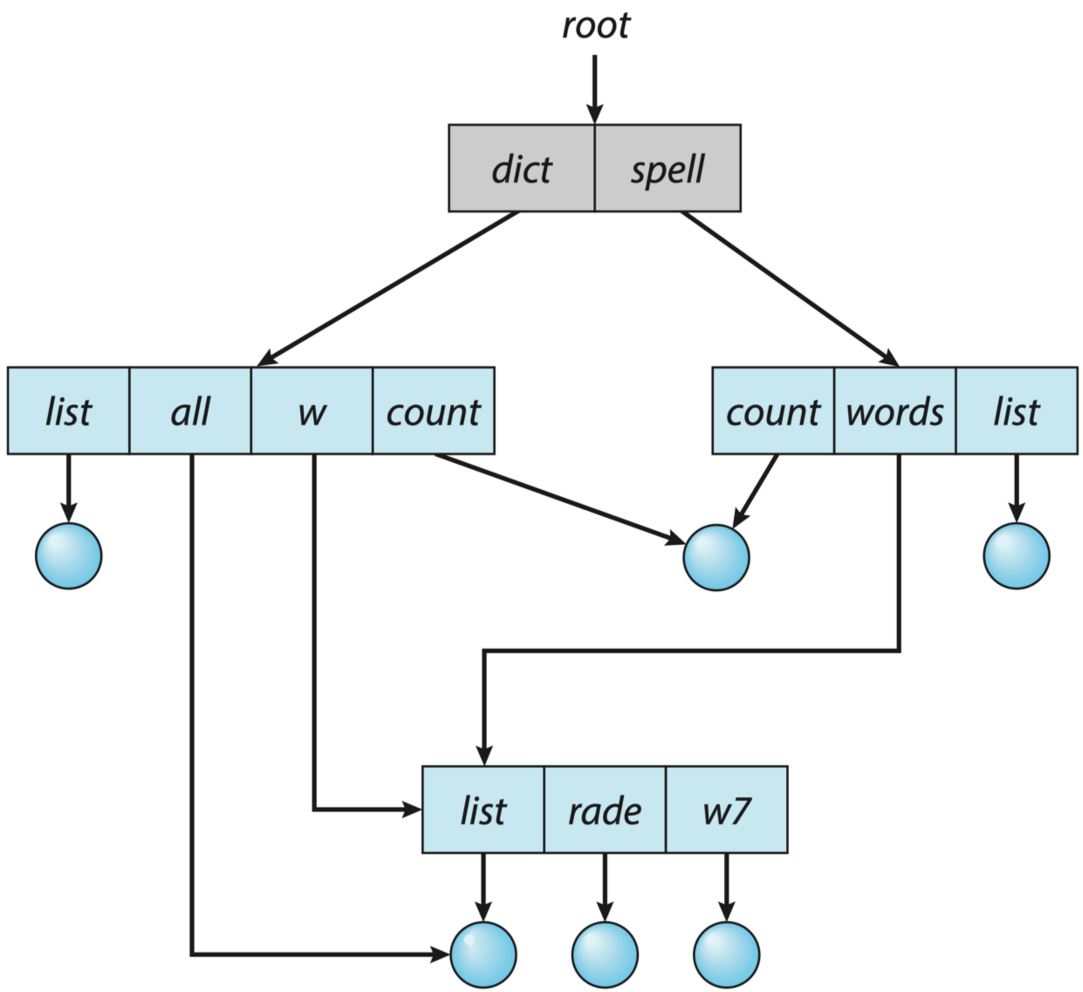
- Can have the same file name for different user
- A little bit more efficient searching
- Path name
- No grouping capability

Tree-Structured Directories



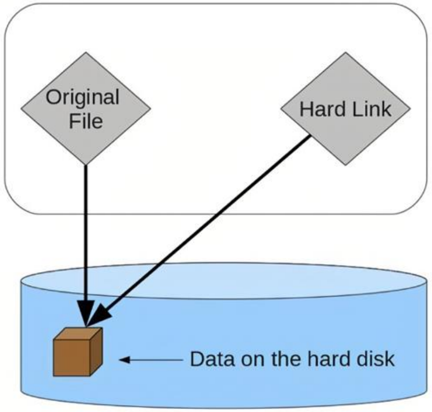
Acyclic-Graph Directories

- Have shared subdirectories and files



Acyclic-Graph Directories: Implementations

- Two implementations for shared files:
  - **Option 1**: Create a new directory entry, called **link**.
    - Link is a pointer to another file or subdirectory
      - The name of the real file is included in the link information.
    - When a reference to a link is made, it is resolved by using the stored path name to locate the real file.
  - **Option 2**: Simply duplicate all information about shared files in both sharing directories.
    - Both entries are identical and equal.
    - Issue: Have to maintain consistency when a file is modified.
- In practice, the **link method** is more commonly used.

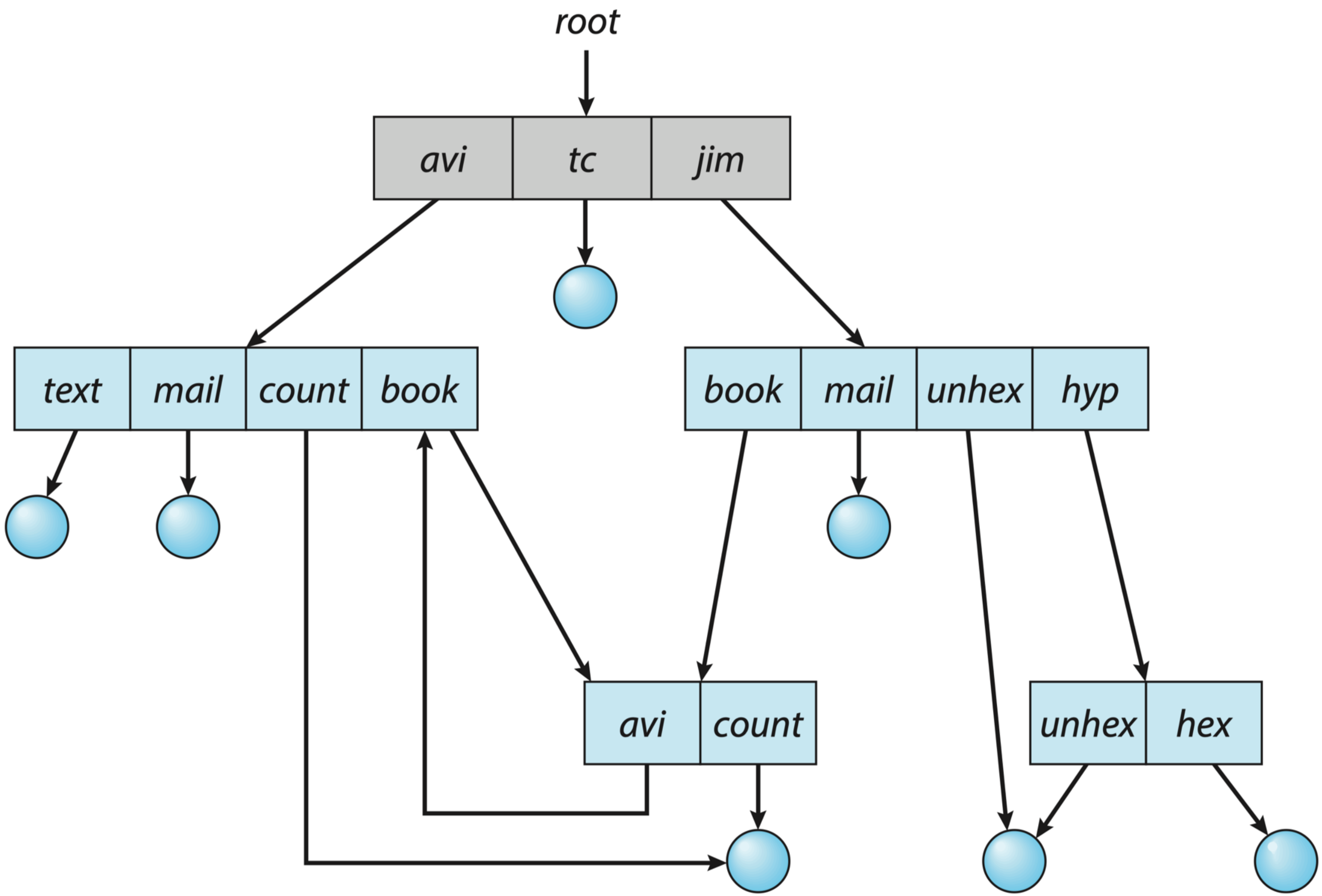


Acyclic-Graph Directories: Deletion

- **Dangling pointer (悬空指针)**:
  - If we remove the file whenever anyone deletes it, then this action may leave dangling pointers to the non-existent file.
  - If the remaining file pointers contain actual disk addresses, and the space is then reused for other files, these dangling pointers may point to other files.

- Countermeasures:
  - **Option 1**:
    - If the file entry itself is deleted, the space for the file is deallocated, and we can search for its links and remove them as well.
    - Or we can leave the links until an attempt to use them fails.
  - **Option 2**:
    - We preserve the file until all references to it are deleted.
    - We could keep a list of all references to a file (directory entries or symbolic links), or just a count of file reference numbers

General Graph Directory



General Graph Directory (Cont.)

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

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

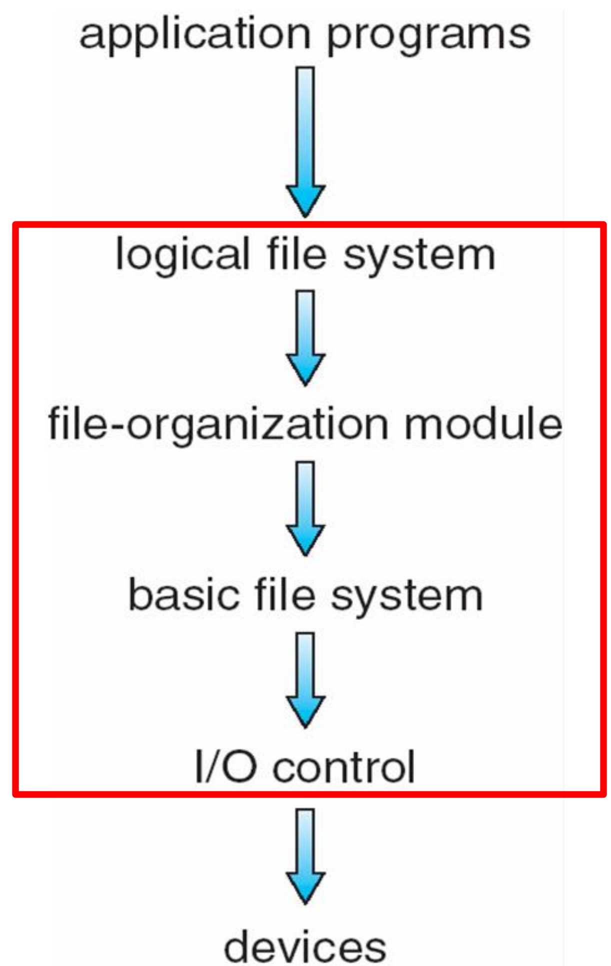
File System Structure and Operations

File-System Structure

- File system designs:
  - **Frontend**: How the file system should look to the user?
  - **Backend**: Creating algorithms and data structures to map the logical file system onto the physical secondary storage devices.
- **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, and retrieved easily
- There could be many file systems within an operating system
  - Each with its own format:
  - Examples:
    - Unix has UFS, FFS
    - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray
    - Linux has more than 130 types, extended file system ext3 and ext4 leading
    - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

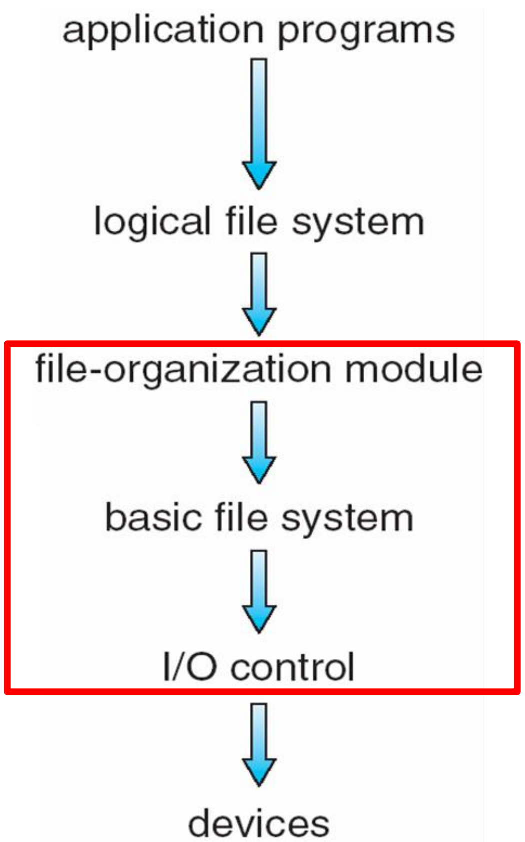
Layered File System

- File system organized into layers
  - Pros:
    - Useful for **reducing complexity and redundancy**,
  - Cons:
    - **Adds overhead** and can **decrease performance**
  - Logical layers can be implemented by any coding method according to OS designer



File System Layers

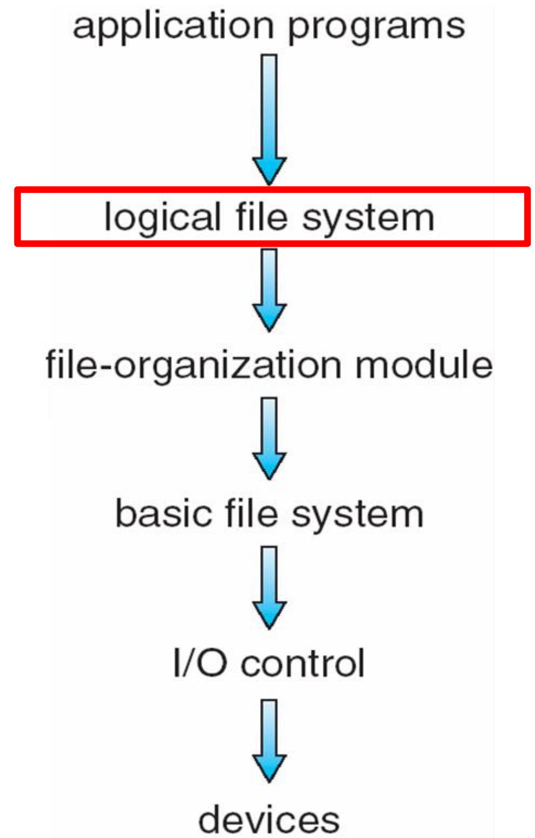
- **Device drivers** manage I/O devices at the I/O control layer, like a translator
  - **Input**: "`retrieve block 123`" (logical address)
  - **Output**: "`read drive1, cylinder 72, track 2, sector 10, into memory location 1060`", low-level hardware-specific commands to hardware controller
- **Basic file system** issues generic commands to the device driver to read and write blocks on the storage device based on logical addresses
  - In charge of I/O request scheduling
  - Manages memory buffers and caches. Buffers hold data in transit, and caches hold frequently used data
- **File organization module** understands files and their logical blocks (from 0 to N)
  - Also includes a free-space manager that tracks unallocated blocks and allocates them



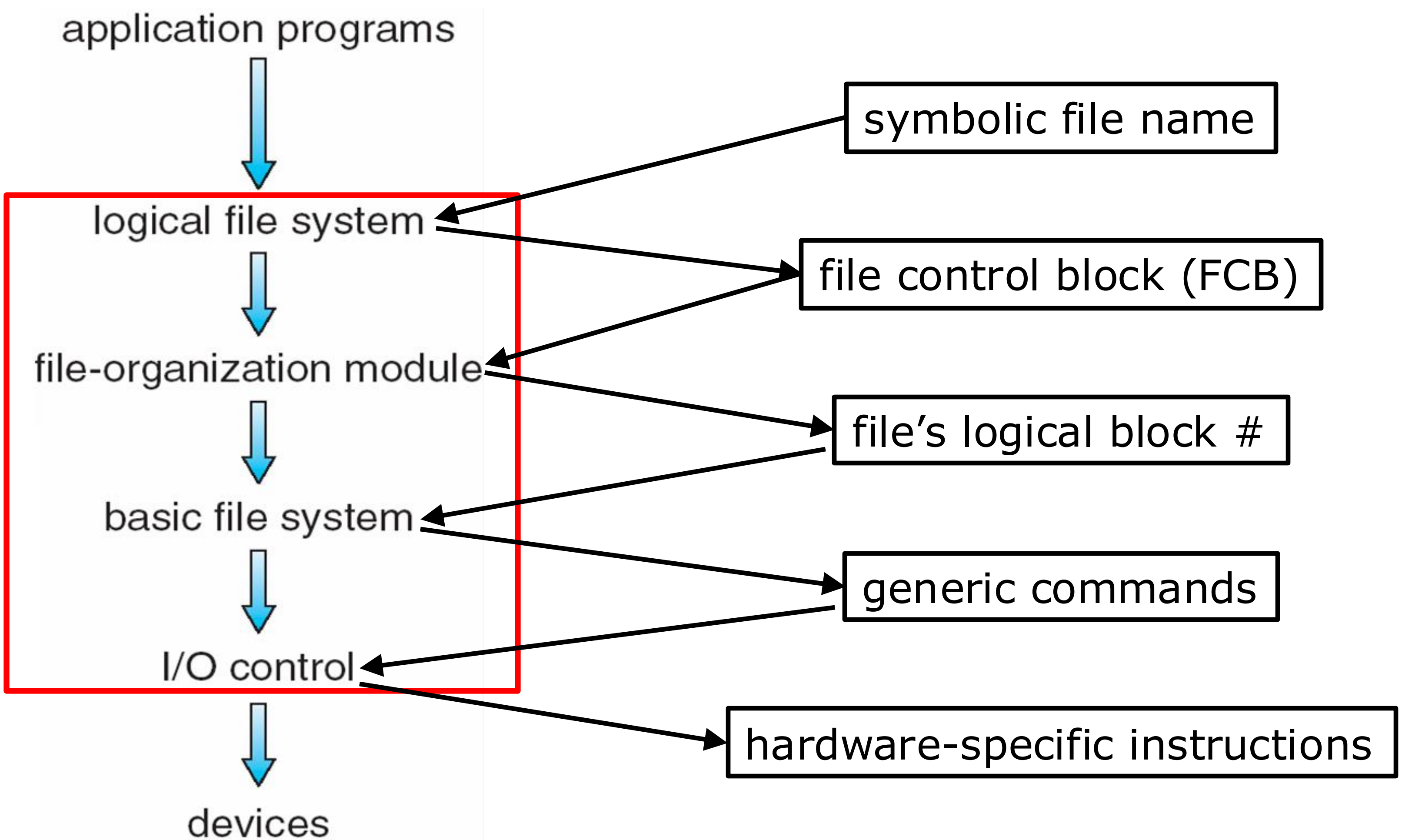
File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, and location by maintaining **file control blocks** (also called **inodes** in Unix)
  - Directory management
  - Protection
- OS maintains **FCB** per file, which contains many details about the file
  - Typically, FCB stores 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



File System Layers



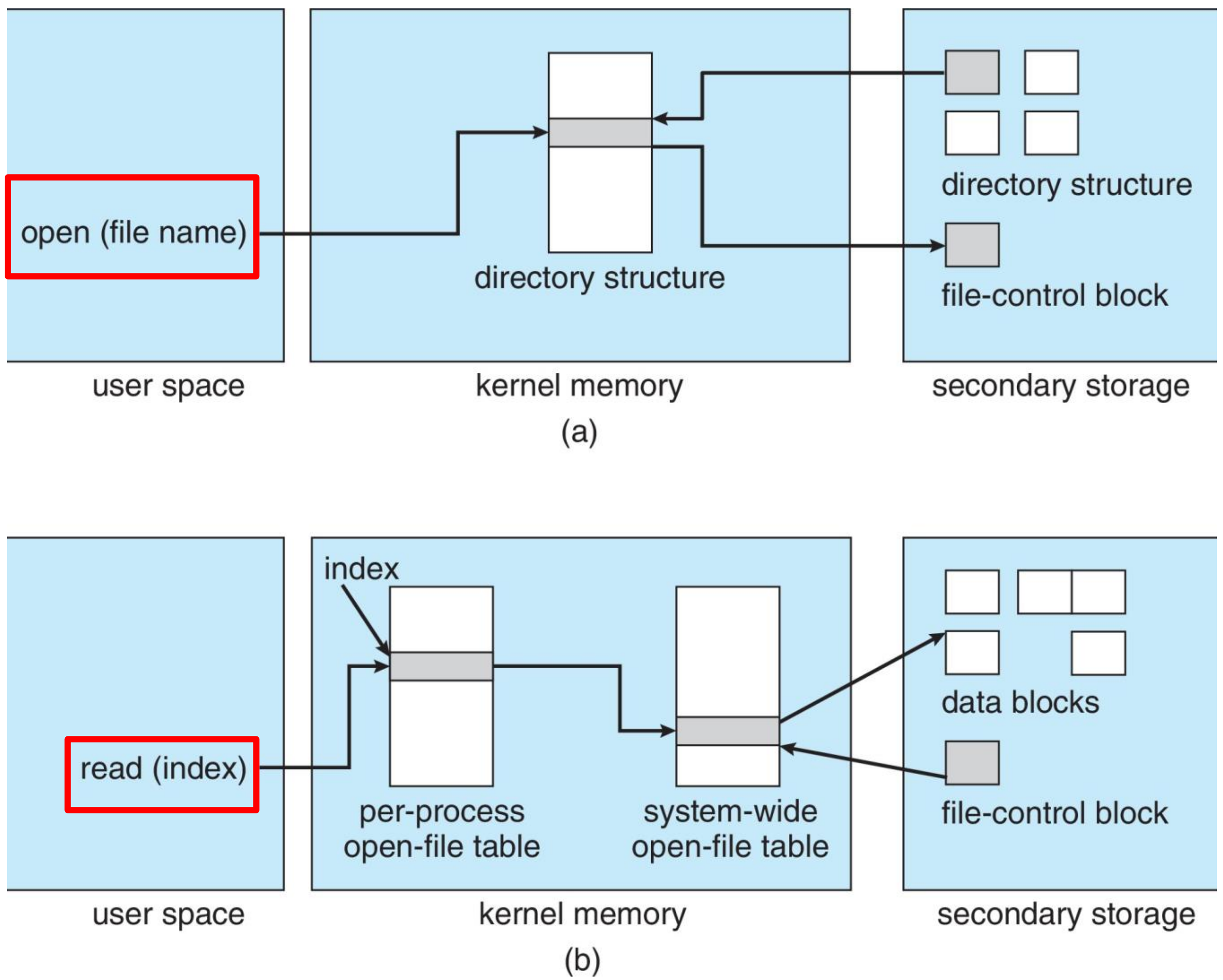
File-System Operations

- **On-storage data structures** for file system operations:
  - **Boot control block** (per volume) contains info needed by the system to boot OS from that volume
    - Needed if volume contains OS, usually first block of volume
  - **Volume control block** (per volume) contains volume details
    - Total # of blocks, # of free blocks, block size, free block pointers or array
  - **Directory structure** (per file system) organizes the files
    - Names and inode numbers, master file table
  - **File control block** (per file) contains details about a file, including the unique ID associated with a directory entry.

File-System Operations

- **In-memory data structures** for file system operations:
  - **Mount table** contains information about each mounted volume
  - **Directory-structure cache** holds the directory information of recently accessed directories
  - **System-wide open-file table** contains a copy of the FCB of each open file, as well as other information.
  - **Per-process open-file table** contains pointers to the appropriate entries in the system-wide open-file table,
    - And other information for all files the process has open.
  - **Buffers** hold file-system blocks when they are being read from or written to a file system.

File System Structure Implementation

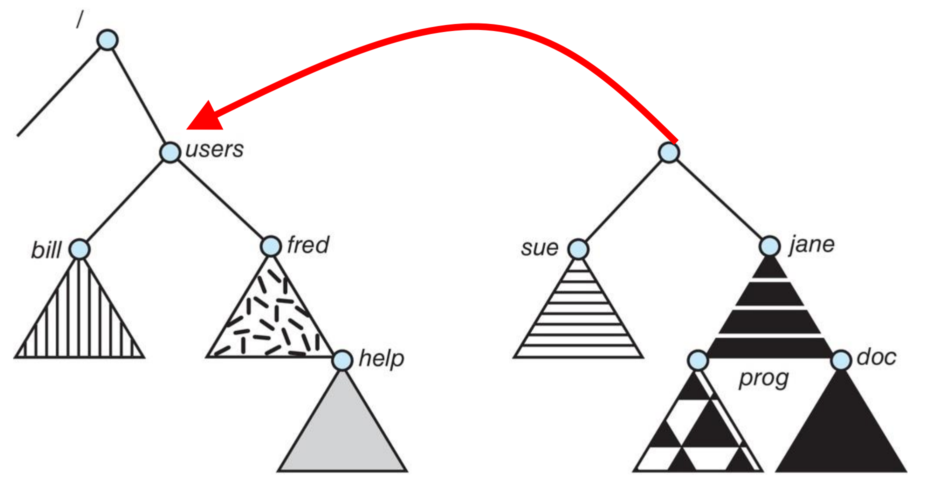


Partitions and Mounting

- Partition can be a **volume** containing:
  - **A file system ("cooked")** – If a partition contains a bootable file system, then the partition also needs boot information
  - **Raw** – just a sequence of blocks with no file system
- **Boot block** can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-OS booting
- **Root partition** contains the OS, other partitions can hold other types of OS, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually

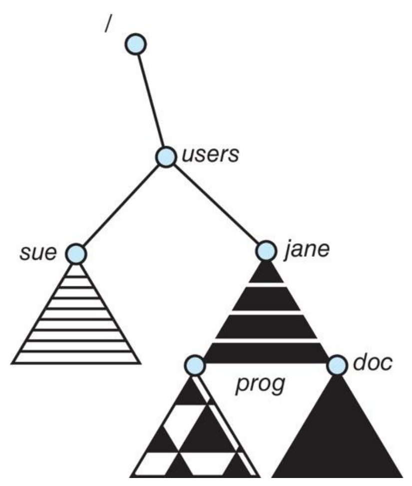
File Systems and Mounting

(a) Unix-like file system directory tree
(b) Unmounted file system



(a) (b)

After mounting (b) into the existing directory tree at `/users`



Allocation Methods

File Allocation Methods

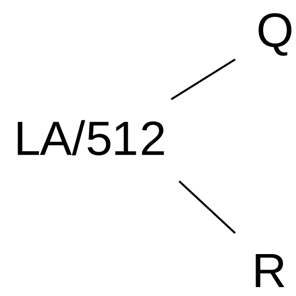
- An allocation method refers to **how disk blocks are allocated for files**
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Allocation Methods: (1) Contiguous

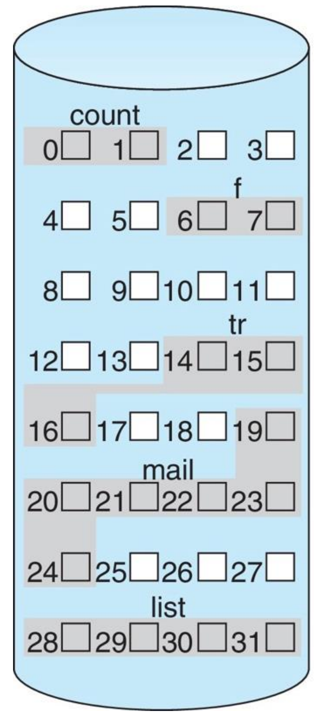
- **Contiguous allocation** – each file occupies a set of contiguous blocks
  - **Benefits**:
    - Simple – only starting location (block #) and length (number of blocks) are required
    - Best performance in most cases
  - **Problems** include
    - Finding space for file
    - Need to know file size
    - External fragmentation: Free storage space is broken into little pieces
      - need for **compaction off-line** (**downtime**) or **on-line**

Allocation Methods: (1) Contiguous

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



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



directory

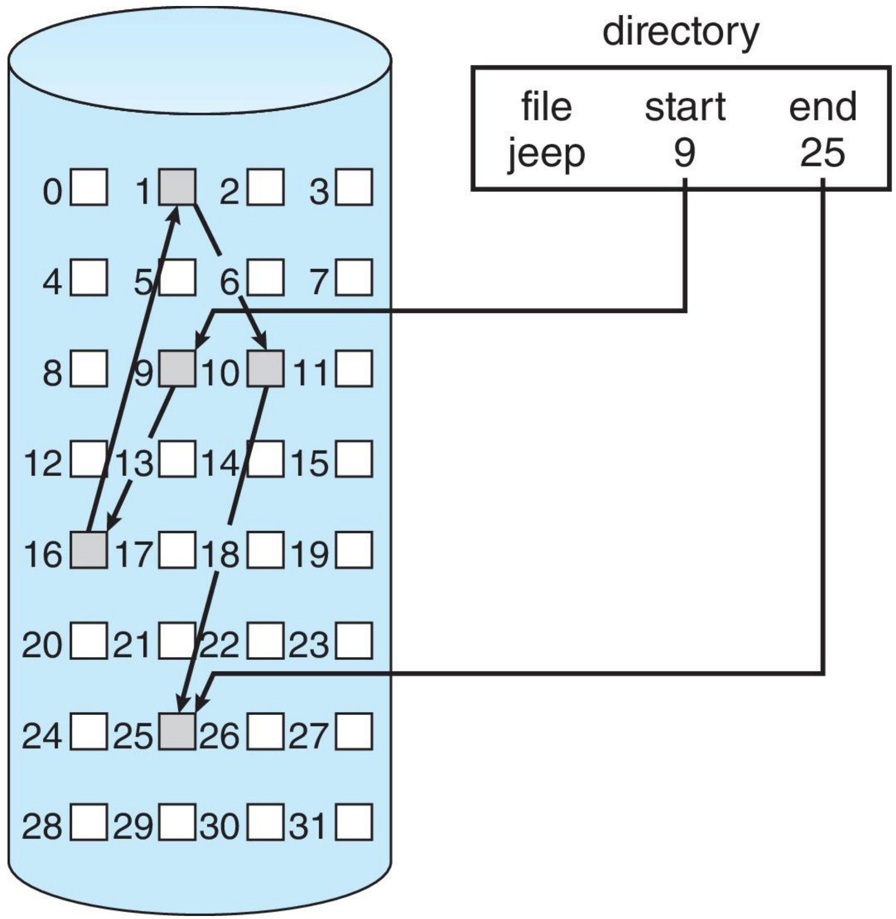
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Extension: 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**
  - A contiguous chunk of space is allocated initially
  - If that amount is not large enough, another chunk of contiguous space (an extent) is added
- An **extent** (区间) is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Allocation Methods: (2) Linked

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

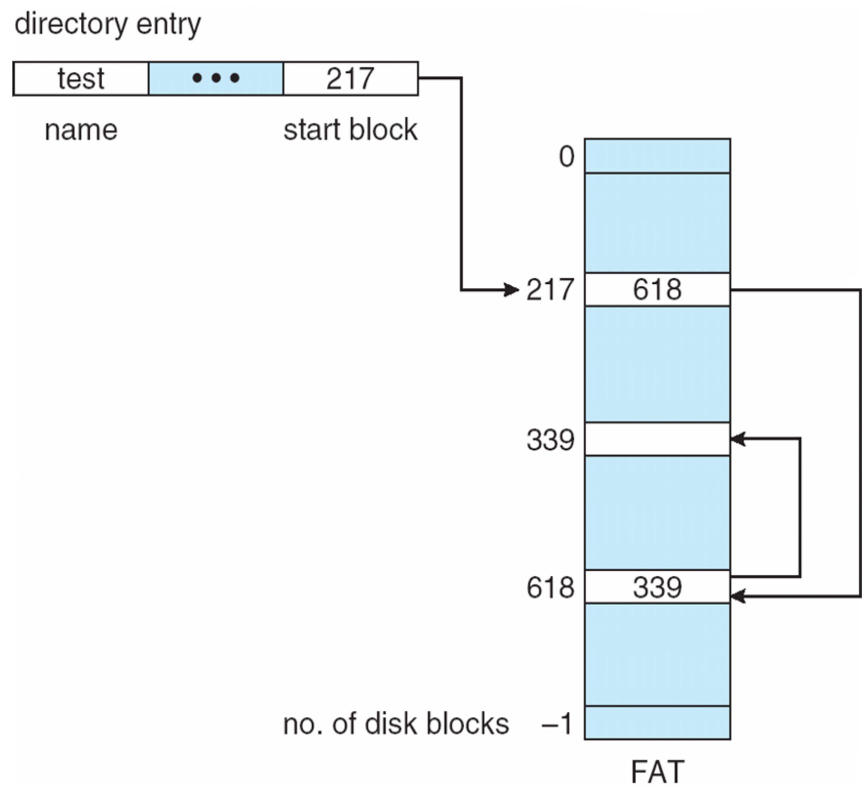


Allocation Methods: (2) Linked

- **Linked allocation** – each file is a linked list of blocks
  - Each block contains pointer to next block
  - File ends at nil pointer
  - Free space management system called when new block needed
- **Benefits**:
  - No external fragmentation
  - Can improve efficiency by clustering blocks into groups but increases internal fragmentation
- **Limitations**:
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

Extension: File-Allocation Table (FAT)

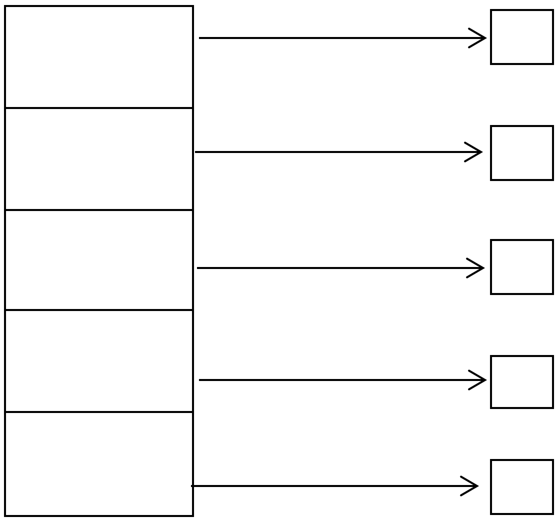
- **FAT (File Allocation Table)** variation
  - Beginning of volume has a table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple
- DOS / Windows



Allocation Methods: (3) Indexed

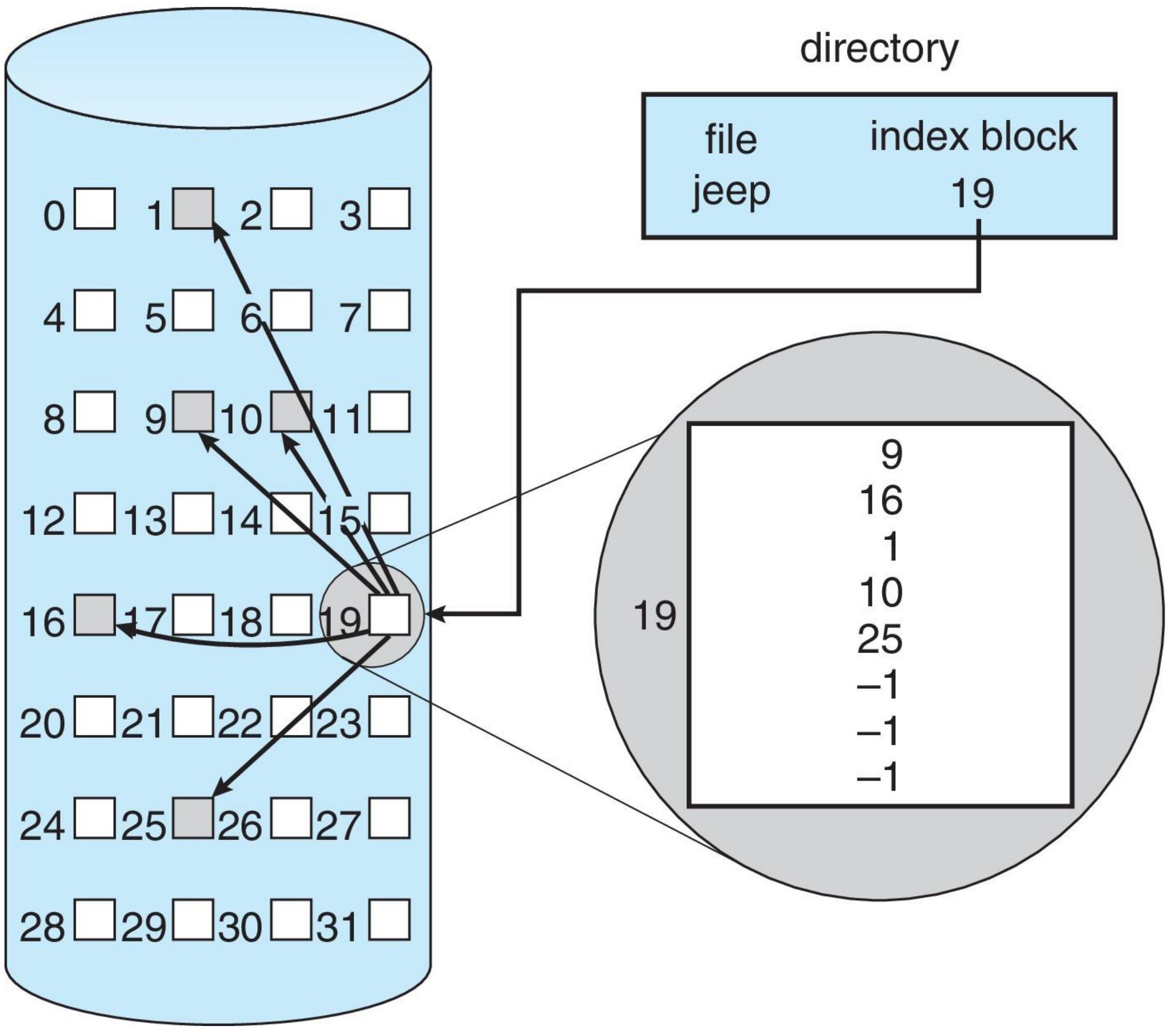
- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks
  - Index block is an array of storage-block addresses

- Logical view

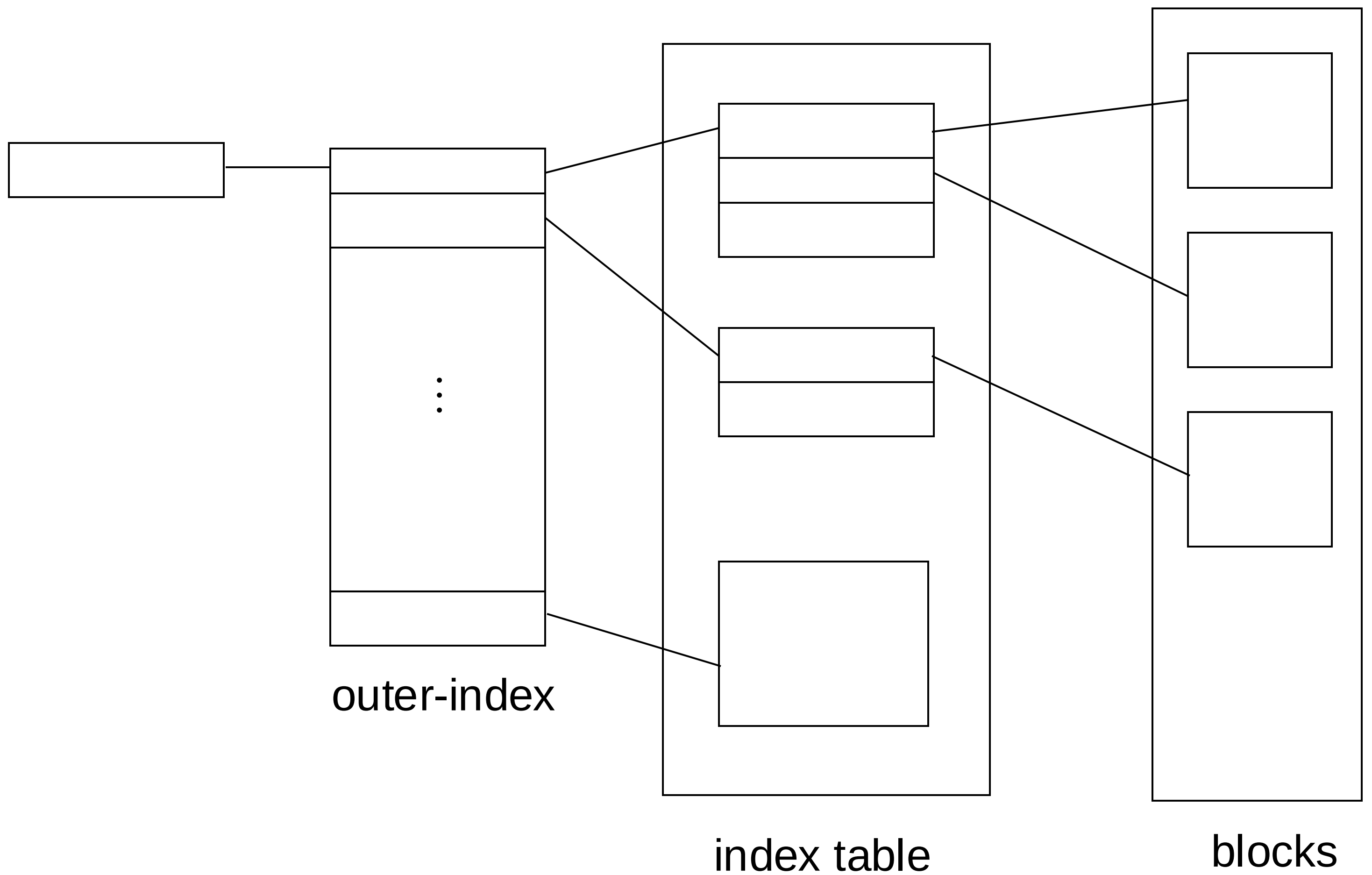


index table

Example of Indexed Allocation

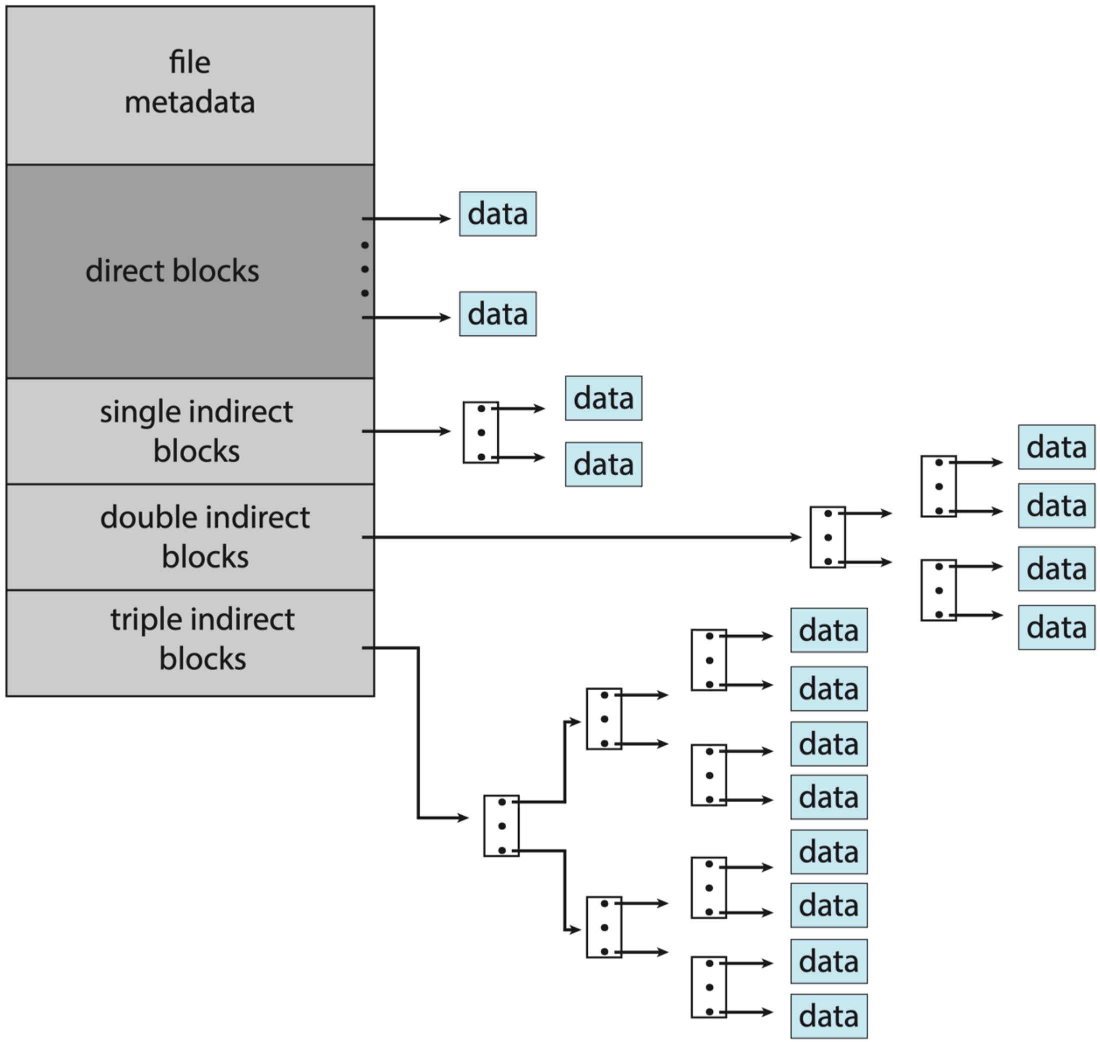


Indexed Allocation – Multilevel Index



Multilevel Index

Indexed Allocation – Combined Scheme



Combined Scheme with **UNIX I-node**

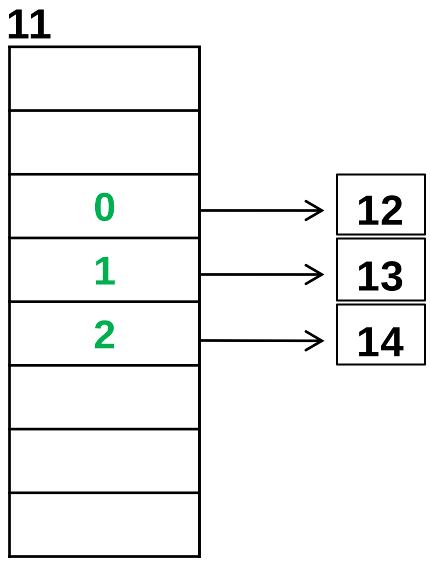
I-node Example

- Suppose a file system is constructed using blocks of 32 bytes. A pointer needs 4 bytes. The I-node structure is as follows (word, value):

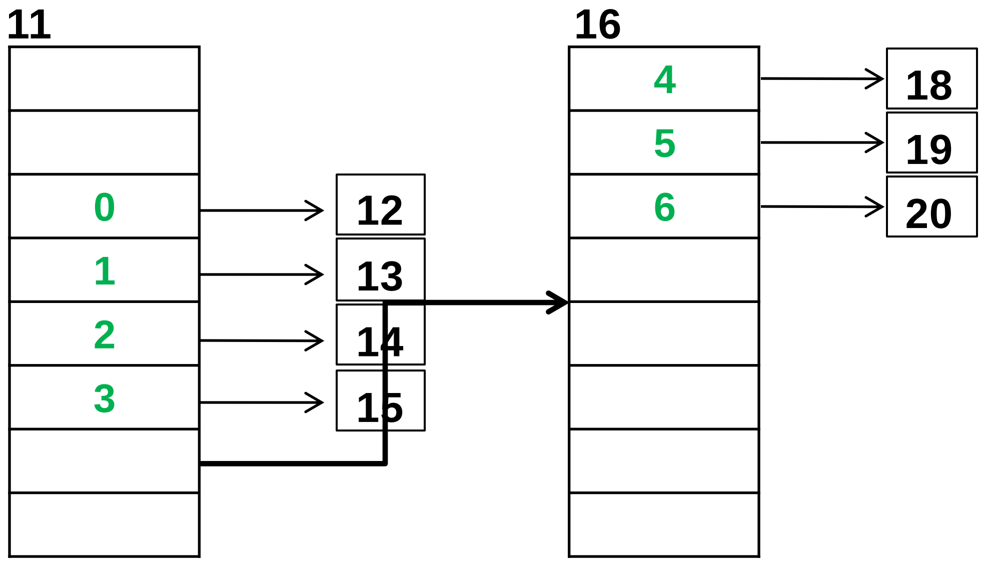
0	Permission word
1	File Size
2	Direct block
3	Direct block
4	Direct block
5	Direct block
6	Single-indirect
7	Double-indirect

- Assume that free blocks are allocated in logical order starting with block 11. Also it has been determined that blocks 17 and 32 are bad and cannot be allocated.
- Draw a block diagram showing the structure of the I-node and the blocks that are allocated for
  - Original file size of 3 blocks
  - Adding 4 blocks
  - Adding 17 blocks

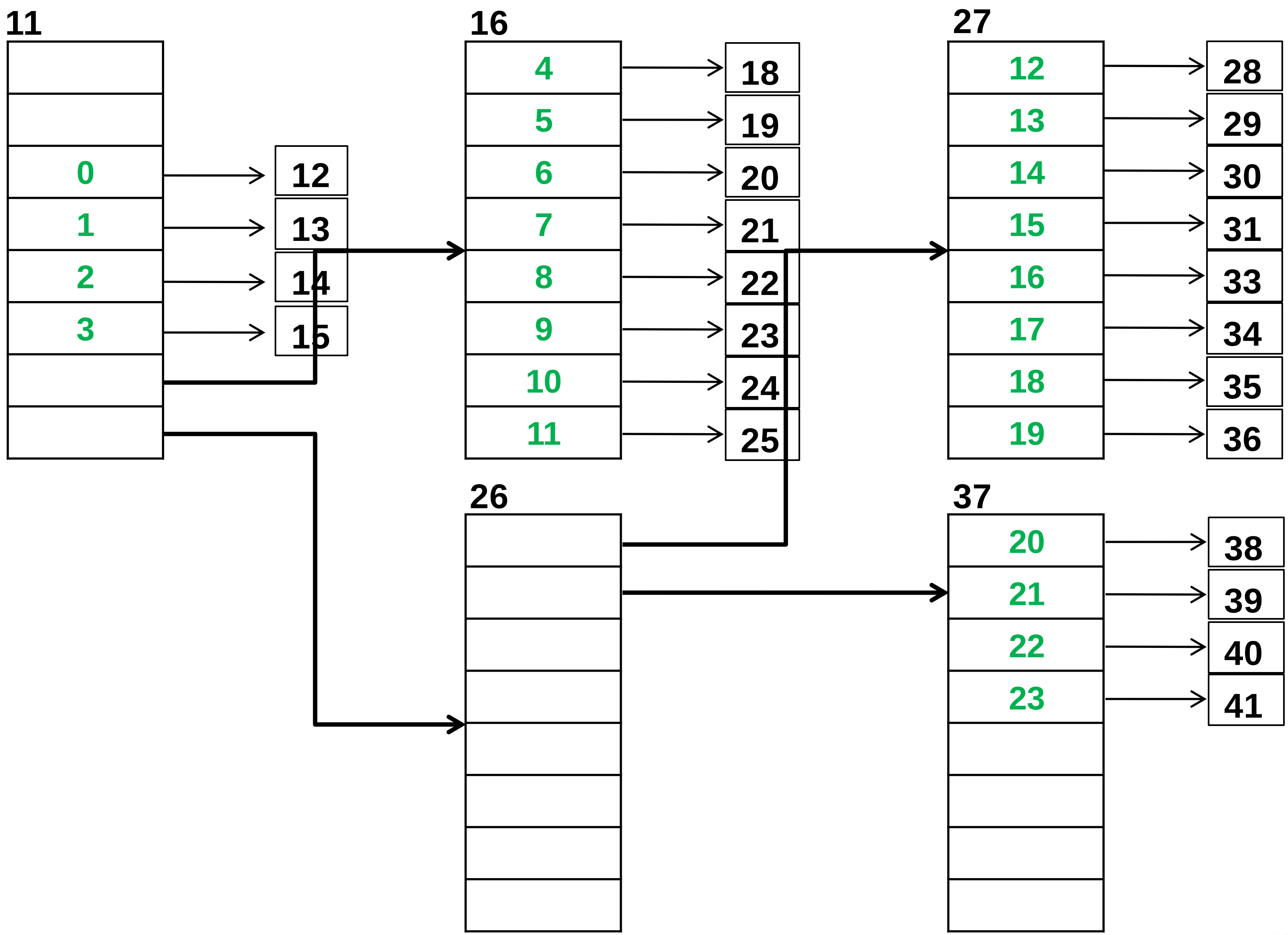
I-node: Original file size of 3 block



I-node: Adding 4 blocks



I-node: Adding 17 blocks



Combined with Disk Scheduling

- Suppose that a single-platter and single-side hard disk is used to store the file system.

- There are 20 tracks logically numbered from 0 to 19.
  - Each track contains 8 blocks.
  - Specifically, track 0 holds blocks 0-7, track 1 holds blocks 8-15, so on and so forth.

- We assume that all the I/O requests are for the above file in the disk queue, and the **block access sequence** is shown as follows.

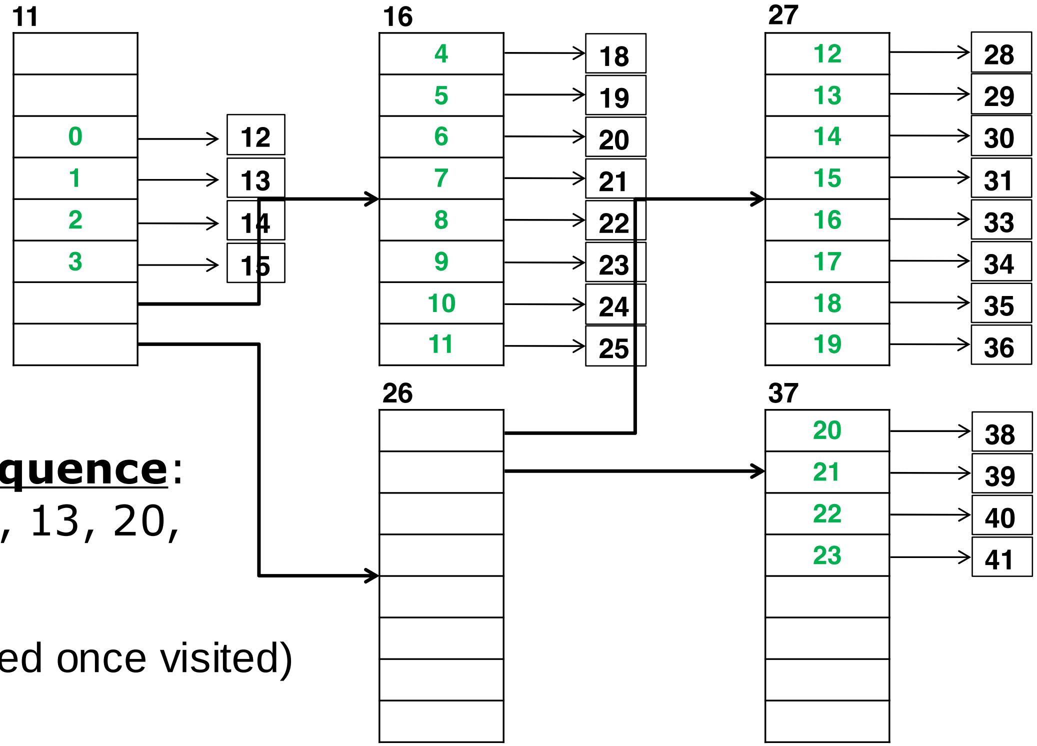
  0, 1, 2, 6, 7, 8, 9, 13, 20, 16, 6, 2

  Suppose that the head just finished serving an I/O request at track 5.

- Please consider the disk scheduling when First-Come, First-Served (FCFS) scheduling algorithm is used.

- **An index block is cached once visited**.

Combined with Disk Scheduling



Block access sequence:
0, 1, 2, 6, 7, 8, 9, 13, 20, 16, 6, 2

(index block is cached once visited)

- Physical block access sequence considering index blocks (index blocks in **red**):
  - **11**, 12, 13, 14, **16**, 20, 21, 22, 23, **26**, **27**, 29, **37**, 38, 33, 20, 14
- Track access sequence :
  - **1**, 1, 1, 1, **2**, 2, 2, 2, 2, **3**, **3**, 3, **4**, 4, 4, 2, 1
- Remove duplicated track numbers:
  - 1, 2, 3, 4, 2, 1

Performance

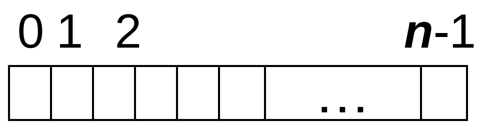
- Best method depends on **file access type**
  - Contiguous great for sequential and random
  - Linked good for sequential, not random
- Declare access type at creation
  - Select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead
- For NVM, no disk head so different algorithms and optimizations needed
  - Using old algorithm uses many CPU cycles trying to avoid non-existent head movement
  - Goal is to **reduce CPU cycles and overall path needed for I/O**

Free Space Management

Free-Space Management: (1) Bit Vector/Map

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)

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



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

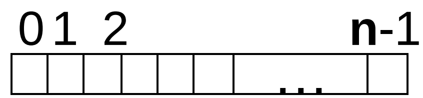
Block number calculation

(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-Space Management: (1) Bit Vector/Map

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



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

- Bit map requires extra space
  - Example:

    block size = 4KB = 2^{12} bytes

    disk size = 2^{40} bytes (1 terabyte)

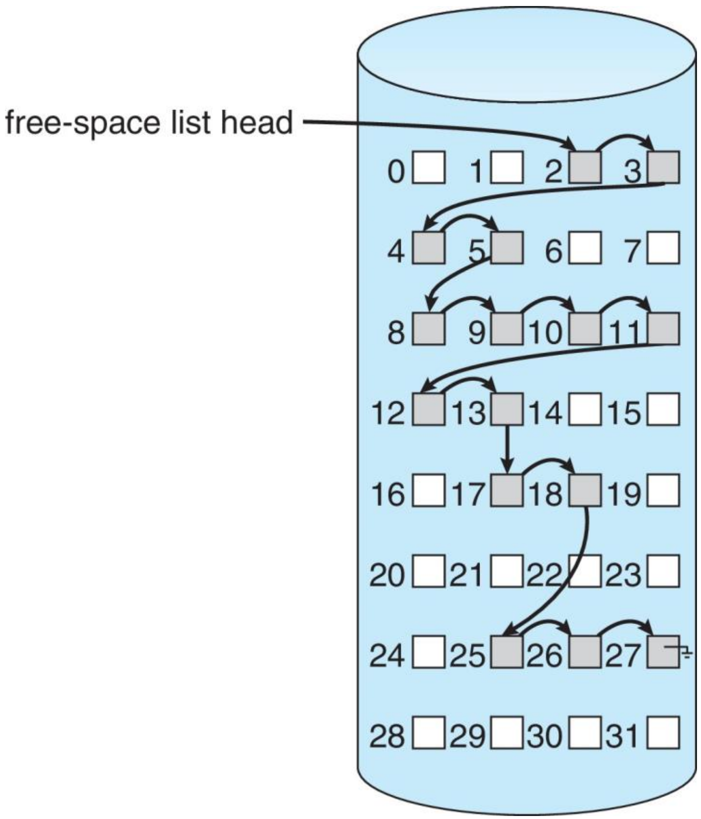
    $\mathbf{n} = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if **record in clusters of 4 blocks -> 8MB of memory**

- Easy to get contiguous files

Free-Space Management: (2) Linked List

- **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: (3) 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

Free-Space Management: (4) Space Maps

- **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
    - A given volume can contain hundreds of metaslabs
    - Each metaslab has an associated space map, using counting algorithm
  - 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

Virtual and Remote File Systems

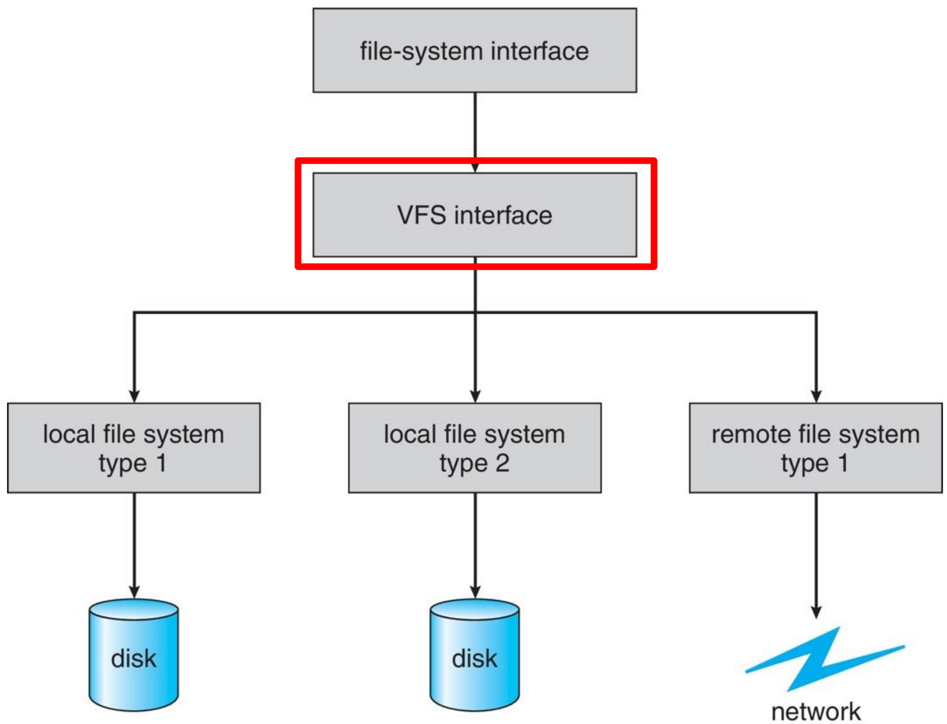
Virtual File Systems

- **Virtual File Systems (VFS)** on Unix provide an object-oriented way of implementing file systems

- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one file system type or network file system
    - Implements **vnodes** that hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems (Cont.)

- Virtual file system (VFS) layer:
  - Separates file-system-generic operations from their implementations.
  - Provides a mechanism for uniquely representing a file throughout a network.
- Example



Virtual File System Implementation

- For example, Linux has four object types:
  - **inode, file, superblock, dentry**
- VFS defines set of operations on the objects that must be implemented
  - Every object has a pointer to a function table
    - Function table has addresses of routines to implement that function on that object
    - For example:
      - **`int open(. . .)`**—Open a file
      - **`int close(. . .)`**—Close an already-open file
      - **`ssize t read(. . .)`**—Read from a file
      - **`ssize t write(. . .)`**—Write to a file
      - **`int mmap(. . .)`**—Memory-map a file

Remote File Systems

- **Sharing of files** across a network

- Three methods:
  - **Method 1**: Manually sharing each file – programs like **ftp**
  - **Method 2**: Use a **distributed file system (DFS)**
    - Remote directories visible from local machine
  - **Method 3**: **World Wide Web**
    - A bit of a revision to first method
    - Use a browser to locate file/files and download /upload
    - Anonymous access doesn't require authentication

Client-Server Model

- Sharing between a **server** (providing access to a file system via a network protocol) and a **client** (using the protocol to access the remote file system)
  - Identifying each other via network ID can be spoofed, encryption can be performance expensive
- Use NFS an example
  - User auth info on clients and servers must match (UserIDs for example)
  - Remote file system mounted, file operations sent on behalf of user across network to server
  - Server checks permissions, file handle returned
  - Handle used for reads and writes until file closed

Summary

- A **file** is a sequence of logical records defined and implemented by the OS.
- Access to files can be controlled by creating user groups.
- **Directories** are created to organize files: (1) Single-level structure, (2) Two-level structure
- OS maps the logical file concept onto physical storage devices.
- File systems are often implemented in a layered structure.
- The various files within a file system can be allocated space on the storage device in three ways: contiguous, linked, or indexed allocation.
- Free-space allocation methods include **bit vectors** and **linked lists**.
- **Virtual file system** (VFS) allows the same APIs to be used for different types of file systems.
- **Remote file systems** enable sharing of files across a network.

Homework

- Reading
  - Chapter 13, 14, 15

- Homework
  - Please check Canvas for HW4 release, due in one week!