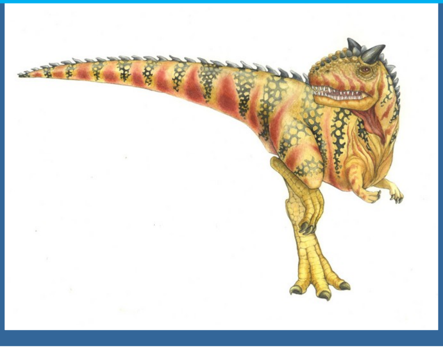
Lecture 10: File System

Hossein Asadi (asadi@sharif.edu)

Rasool Jalili (jalili@sharif.edu)



Fall 2024

Lecture 10: File-System

- File Concept
- File System Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File-System Comparison

Objectives

- To Explain Function of File Systems
- To Describe Interfaces to File Systems
- To Discuss File-System Design Tradeoffs, including Access Methods, File Sharing, File Locking, and Directory Structures
- To Explore File-System Protection

File Concept

- Contiguous Logical Address Space
- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program
- Contents Defined by File’s Creator
  - Many types
    - Consider **text file**, **source file**, **executable file**

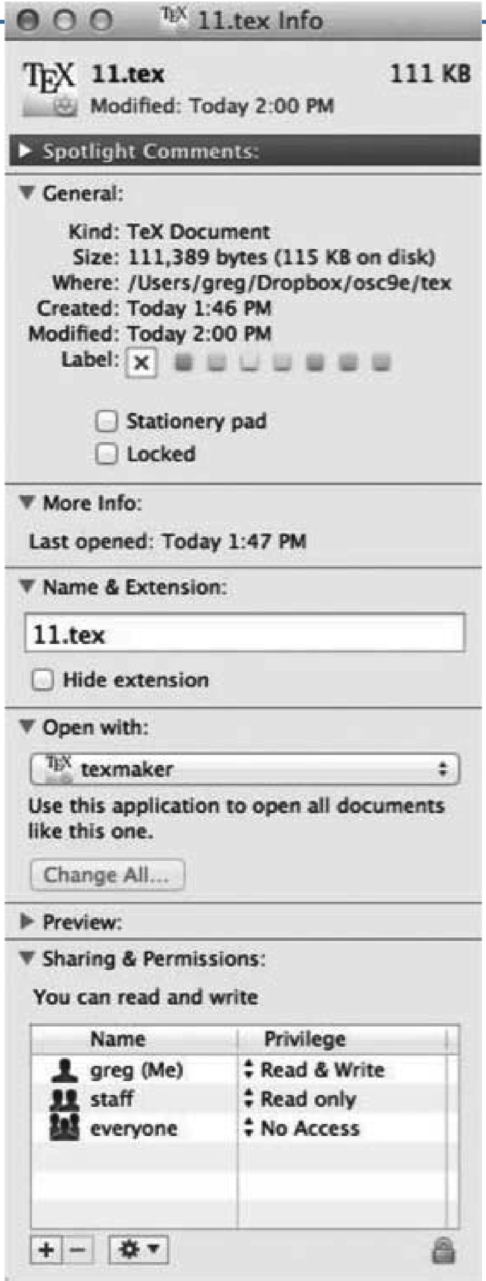
File Attributes

- **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 device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing

File Attributes (cont.)

- **Time, Date, and User Identification**
  - Data for protection, security, and usage monitoring
- Information about Files Kept in Directory Structure
  - Maintained on disk
- Many Variations
  - Including extended file attributes such as file checksum
- Information Kept in Directory Structure

File info Window on Mac OS X



11.tex 111 KB
Modified: Today 2:00 PM

▶ Spotlight Comments:

▼ General:

Kind: TeX Document
Size: 111,389 bytes (115 KB on disk)
Where: /Users/greg/Dropbox/osc9e/tex
Created: Today 1:46 PM
Modified: Today 2:00 PM
Label: ☒

□ Stationery pad
□ Locked

▼ More Info:

Last opened: Today 1:47 PM

▼ Name & Extension:

11.tex

□ Hide extension

▼ Open with:

texmaker

Use this application to open all documents like this one.

▶ Preview:

▼ Sharing & Permissions:

You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

File Operations

- File is an **Abstract Data Type**
- **Create**
- **Write –** at **Write Pointer** Location
- **Read –** at **Read Pointer** Location
- **Reposition within File - Seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search directory structure on disk for entry ***F_i***, and move content of entry to memory
- ***Close (F_i)*** – move content of entry ***F_i*** in memory to directory structure on disk

Open Files

- Several Pieces of Data Needed to Manage Open Files:
  - **Open-file table**: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of file: Info to locate file on disk kept in memory
  - Access rights: per-process access mode information

Open File Locking

- Provided by some OSs 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:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

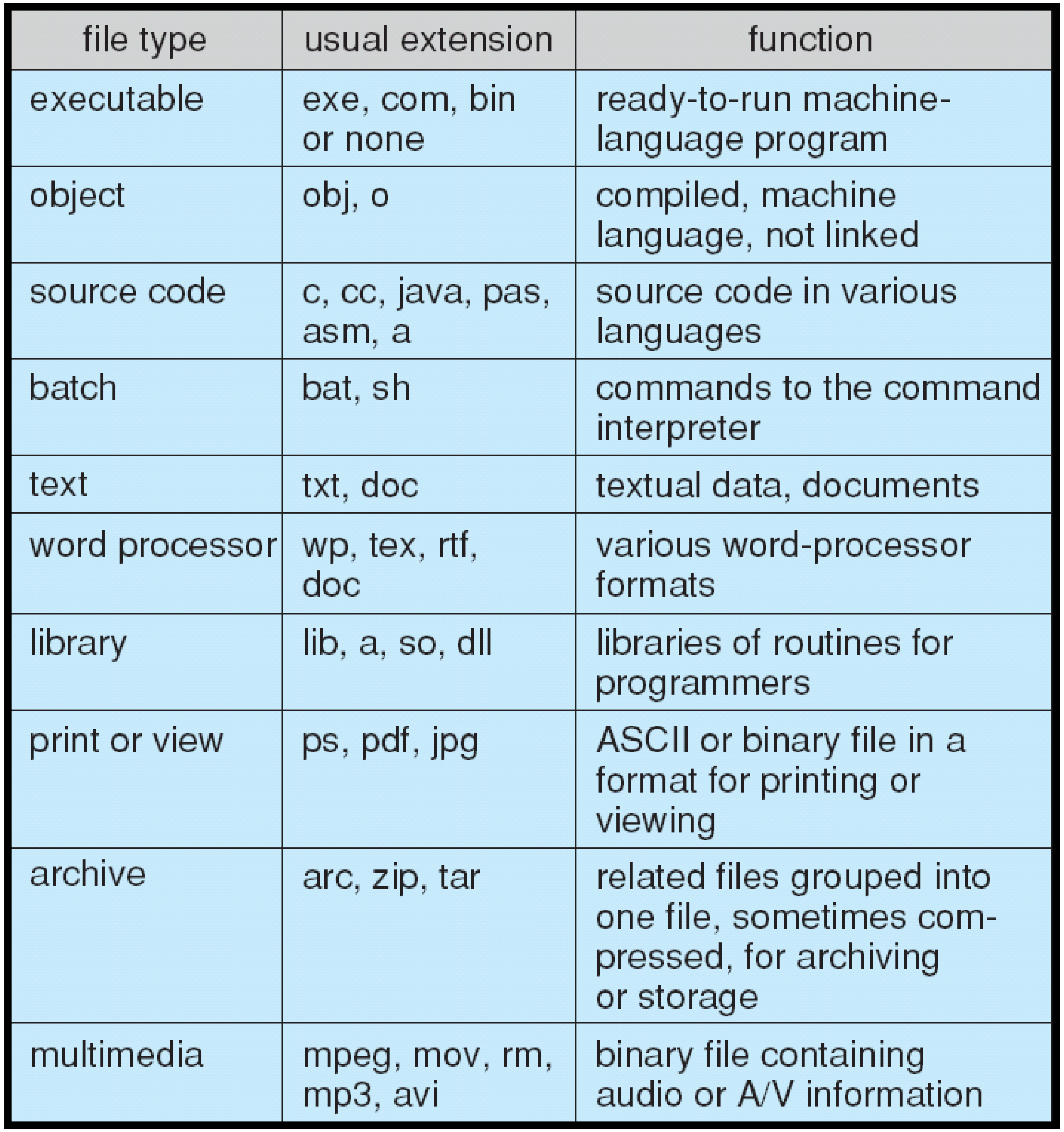
File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
            FileLock sharedLock = null;
            FileLock exclusiveLock = null;
            try {
                    RandomAccessFile raf = new RandomAccessFile("file.txt",
    "rw");
                    // get the channel for the file
                    FileChannel ch = raf.getChannel();
                    // this locks the first half of the file - exclusive
                    exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
                    /** Now modify the data . . . */
                    // release the lock
                    exclusiveLock.release();
```

File Locking Example – Java API (Cont.)

```
                // this locks the second half of the file - shared
                sharedLock = ch.lock(raf.length()/2+1,
    raf.length(),                                      SHARED);
                /** Now read the data . . . */
                // release the lock
                sharedLock.release();
          } catch (java.io.IOException ioe) {
                System.err.println(ioe);
          }finally {
                if (exclusiveLock != null)
                exclusiveLock.release();
                if (sharedLock != null)
                sharedLock.release();
          }
```

File Types – Name, Extension

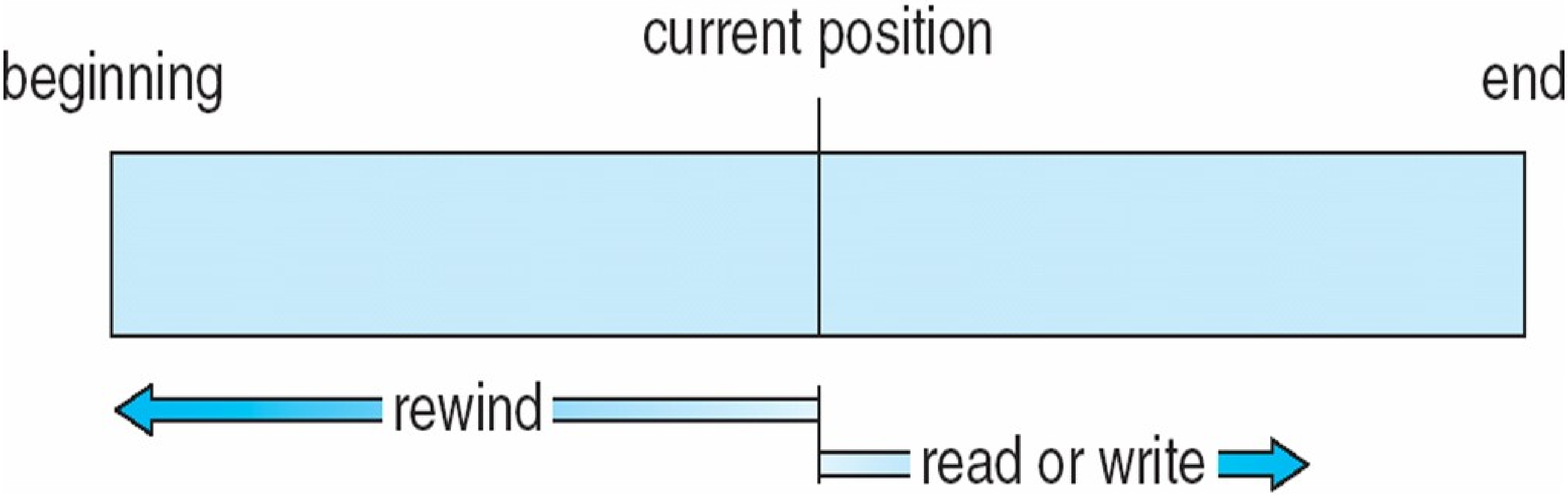


file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual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

- None - Sequence of Words, Bytes
- Simple Record Structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can Simulate Last Two with first Method
  - By inserting appropriate control characters
- Who Decides:
  - OS or program

Sequential-Access File



Access Methods

- **Sequential Access**

```
read next
write next
reset
no read after last write
        (rewrite)
```

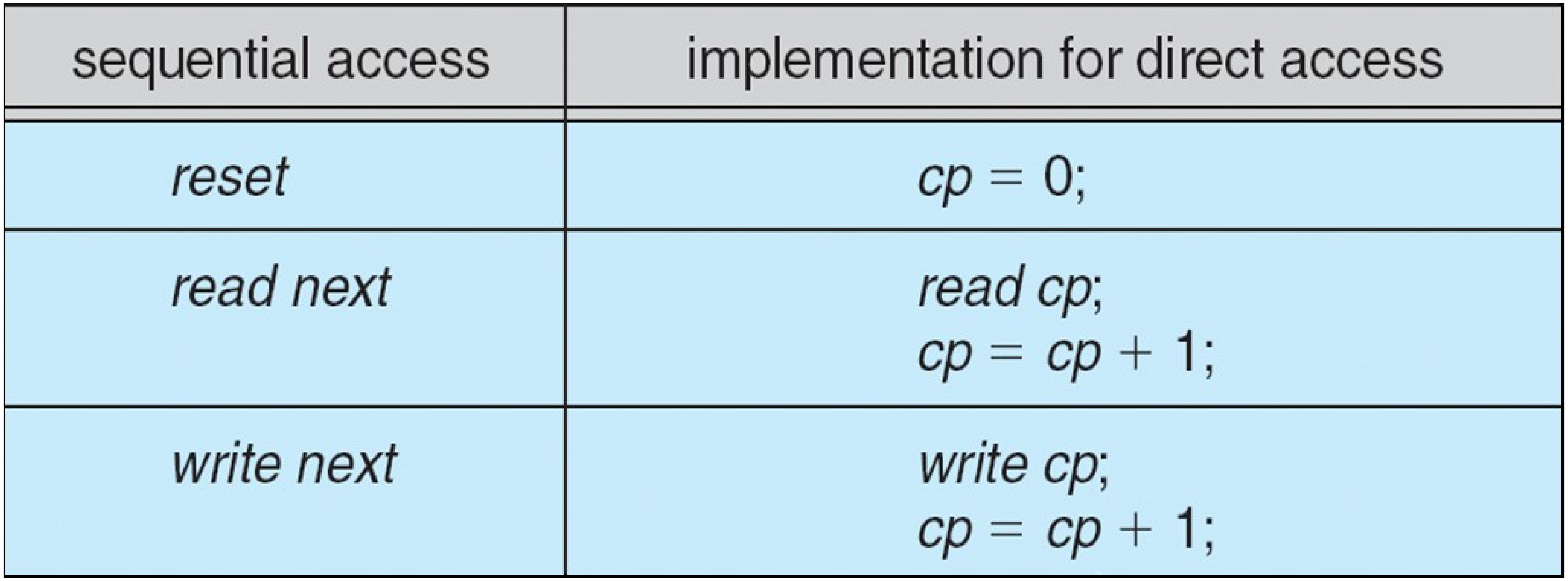
- **Direct Access – file is fixed length logical records**

```
read n
write n
position to n
        read next
        write next
rewrite n
```

***n* = relative block number**

- **Random Access (Can Access in any Arbitrary Order)**
- **Relative Block Numbers Allow OS to decide where File should be Placed**

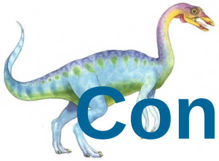
Simulation of Sequential Access on Direct-Access File



sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Unix/POSIX Idea: Everything is a "File"

- Identical Interface for:
  - Files on disk
  - Devices (terminals, printers, etc.)
  - Regular files on disk
  - Networking (sockets)
  - Local inter-process communication (pipes, sockets)
- Based on system calls **`open()`**, **`read()`**, **`write()`**, and **`close()`**
- Additional: **`ioctl()`** for custom configuration that doesn't quite fit
- Note that "Everything is a File" idea was a radical idea when proposed

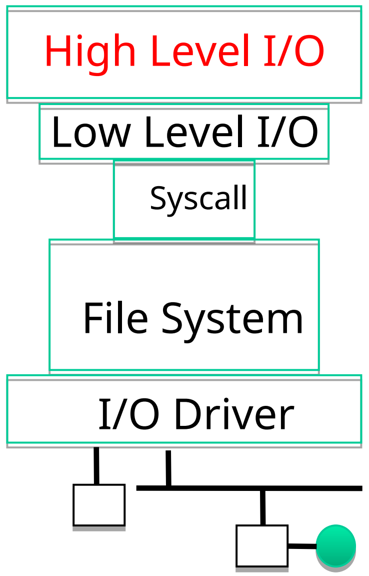


Connecting Processes, File Systems, & Users

- **Every process has *current working directory* (CWD)**
  - Can be set with system call:
    `int chdir(const char *path); //change CWD`
- Absolute paths ignore CWD
  - /home/oski/cs162
- Relative paths are relative to CWD
  - index.html, ./index.html
    - Refers to index.html in current working directory
  - ../index.html
    - Refers to index.html in parent of current working directory
  - ~/index.html, ~cs162/index.html
    - Refers to index.html in the home directory

I/O and Storage Layers

Application / Service



High Level I/O

Low Level I/O

Syscall

File System

I/O Driver

Streams (buffered I/O)

File Descriptors

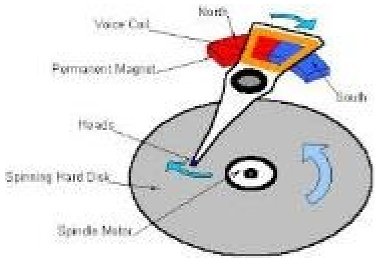
open(), read(), write(), close(), …

Open File Descriptions

Files/Directories/Indexes

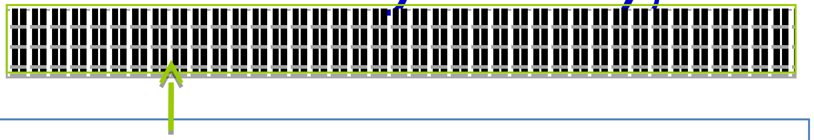
Commands and Data Transfers

Disks, Flash, Controllers, DMA



C High-Level File API – Streams

- Operates on “streams” – unformatted sequences of bytes (text or binary data), with a position:



```
#include <stdio.h>
FILE *fopen( const char *filename, const char *mode );
int fclose( FILE *fp );
```

Mode Text	Binary	Descriptions
r	rb	Open existing file for reading
w	wb	Open for writing; created if does not exist
a	ab	Open for appending; created if does not exist
r+	rb+	Open existing file for reading & writing.
w+	wb+	Open for reading & writing; truncated to zero if exists, create otherwise
a+	ab+	Open for reading & writing. Created if does not exist. Read from beginning, write as append

- Open stream represented by pointer to a FILE data structure
  - Error reported by returning a NULL pointer

C API Standard Streams – `stdio.h`

- Three predefined streams are opened implicitly when the program is executed.
  - `FILE *stdin` – normal source of input, can be redirected
  - `FILE *stdout` – normal source of output, can too
  - `FILE *stderr` – diagnostics and errors
- STDIN / STDOUT enable composition in Unix
- All can be redirected
  - `cat hello.txt | grep “World!”`
  - **`cat`**’s **`stdout`** goes to **`grep`**’s **`stdin`**

C High-Level File API

```
// character oriented
int fputc( int c, FILE *fp );                 // rtn c or
EOF on err
int fputs( const char *s, FILE *fp );   // rtn > 0 or EOF

int fgetc( FILE * fp );
char *fgets( char *buf, int n, FILE *fp );


// block oriented
size_t fread(void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);
size_t fwrite(const void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);
// formatted
int fprintf(FILE *restrict stream, const char *restrict
format, ...);
int fscanf(FILE *restrict stream, const char *restrict
```

C Streams: Char-by-Char I/O

```
int main(void) {
  FILE* input = fopen(“input.txt”, “r”);
  FILE* output = fopen(“output.txt”, “w”);
  int c;
  c = fgetc(input);
  while (c != EOF) {
    fputc(output, c);
    c = fgetc(input);
  }
  fclose(input);
  fclose(output);
}
```

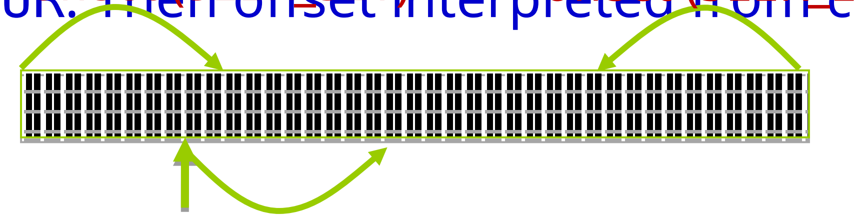
C Streams: Block-by-Block I/O

```
#define BUFFER_SIZE 1024
int main(void) {
  FILE* input = fopen("input.txt", "r");
  FILE* output = fopen("output.txt", "w");
  char buffer[BUFFER_SIZE];
  size_t length;
  length = fread(buffer, BUFFER_SIZE, sizeof(char), 
input);
  while (length > 0) {
    fwrite(buffer, length, sizeof(char), output);
    length = fread(buffer, BUFFER_SIZE, sizeof(char), 
input);
  }
  fclose(input);
  fclose(output);
}
```

C High-Level File API: Positioning Pointer

```
int fseek(FILE *stream, long int offset, int whence);
long int ftell (FILE *stream)
void rewind (FILE *stream)
```

- For `fseek()`, the `offset` is interpreted based on the whence argument (constants in `stdio.h`):
  - SEEK_SET: Then offset interpreted from beginning (position 0)
  - SEEK_END: Then offset interpreted backwards from end of file
  - SEEK_CUR: Then offset interpreted from current position



- Overall preserves high-level abstraction of a uniform stream of objects

Low-Level File I/O: RAW system-call interface

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int open (const char *filename, int flags [, mode_t mode])
int creat (const char *filename, mode_t mode)
int close (int filedes)
```

Bit vector of:
- Access modes (Rd, Wr, …)
- Open Flags (Create, …)
- Operating modes (Appends, …)

Bit vector of Permission Bits:
- User|Group|Other X R|W|X

- Integer return from `open()` is a *file descriptor*
  - *Error indicated by return < 0:* the global `errno` variable set with error (see man pages)
- Operations on *file descriptors*:
  - Open system call created an *open file description* entry in system-wide table of open files
  - *Open file description* object in the kernel represents an instance of an open file
  - Why give user an integer instead of a pointer to the file description in kernel?

Low-Level File API

- Read data from open file using file descriptor:

```
ssize_t read (int filedes, void *buffer, size_t maxsize)
```

  - Reads up to `maxsize` bytes – **might actually read less!**
  - returns bytes read, 0 => EOF, -1 => error

- Write data to open file using file descriptor

```
    ssize_t write (int filedes, const void *buffer, size_t
size)
```

  - returns number of bytes written

- Reposition file offset within kernel (this is independent of any position held by high-level FILE descriptor for this file!

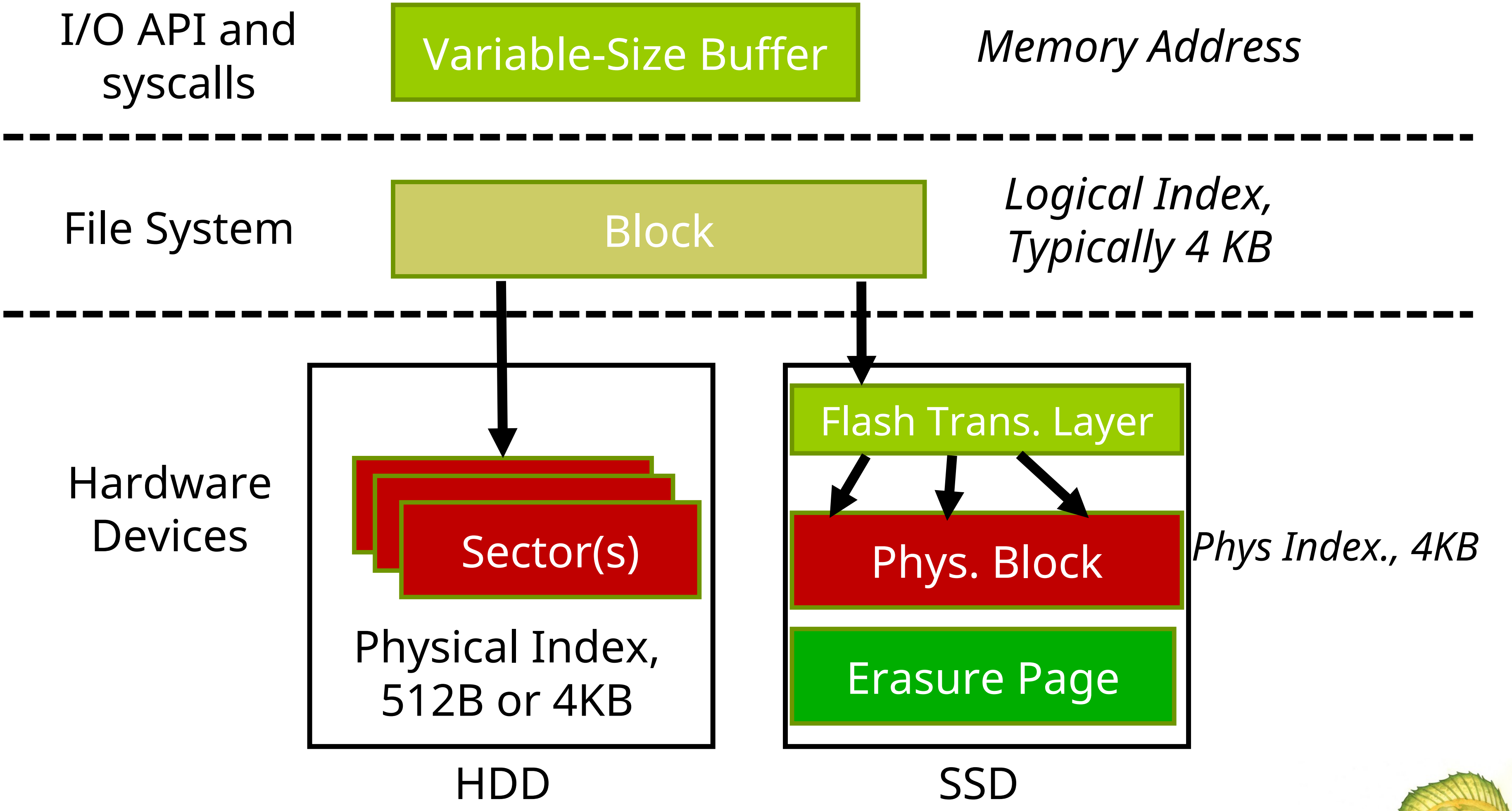
```
off_t lseek (int filedes, off_t offset, int whence)
```

Example: lowio.c

```
int main() {
  char buf[1000];
  int     fd = open("lowio.c", O_RDONLY,
S_IRUSR | S_IWUSR);
  ssize_t rd = read(fd, buf, sizeof(buf));
  int    err = close(fd);
  ssize_t wr = write(STDOUT_FILENO, buf,
rd);
}
```

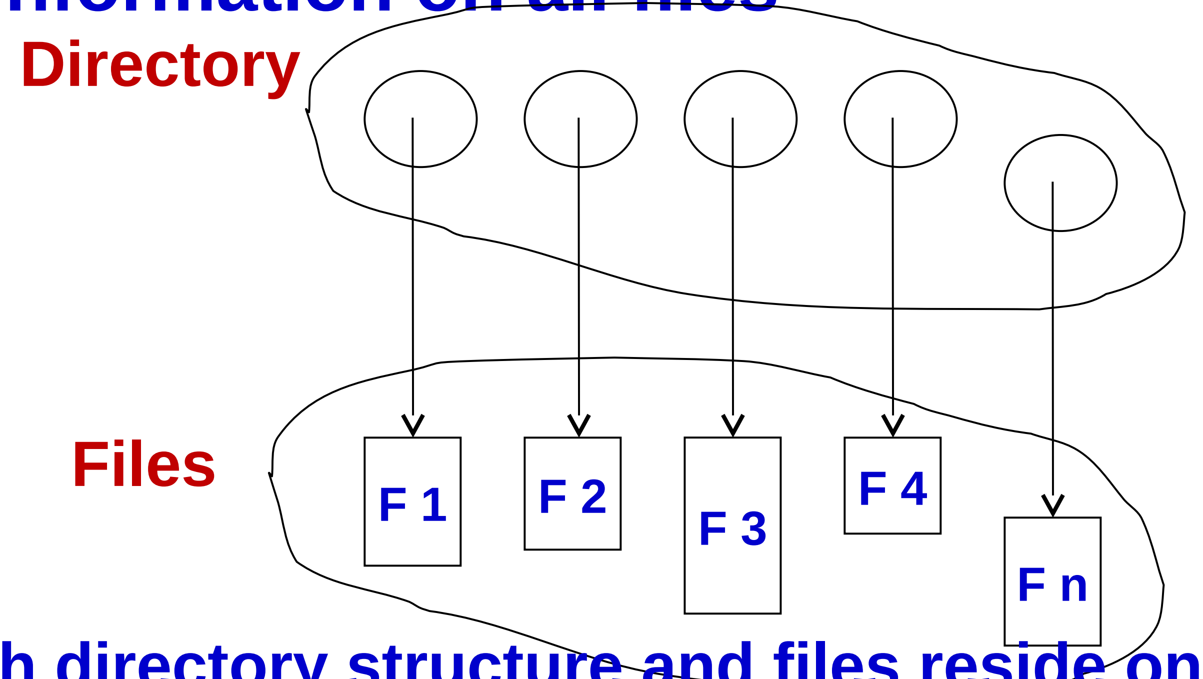
- How many bytes does this program read?

From Storage to File Systems



Filesystem

- **Filesystem**
  - Collection of Files + Directory Structure
- **Directory Structure**
  - A collection of nodes containing information on all files



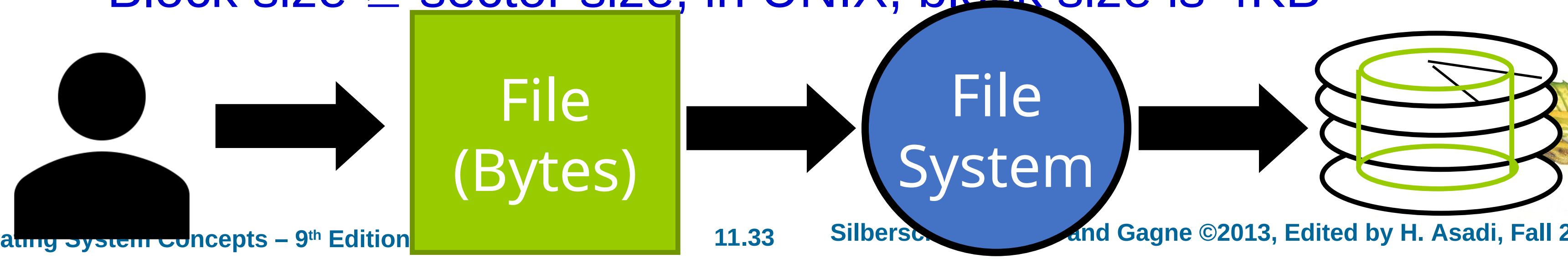
Both directory structure and files reside on disk

Building a File System

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- Classic OS situation: Take limited hardware interface (array of blocks) and provide a more convenient/useful interface with:
  - Naming: Find file by name, not block numbers
  - Organize file names with directories
  - Organization: Map files to blocks
  - Protection: Enforce access restrictions
  - Reliability: Keep files intact despite crashes, hardware failures, etc

User vs. System View of a File

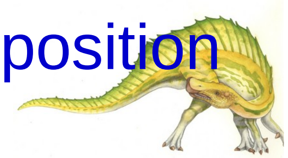
- User's view:
  - Durable Data Structures
- System's view (system call interface):
  - Collection of Bytes (UNIX)
  - Doesn't matter to system what kind of data structures you want to store on disk!
- System's view (inside OS):
  - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
  - Block size ≥ sector size; in UNIX, block size is 4KB



Disk Management

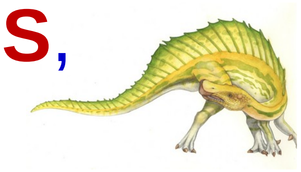
- Basic entities on a disk:
  - File: user-visible group of blocks arranged sequentially in logical space
  - Directory: user-visible index mapping names to files

- Disk is accessed as linear array of sectors
- How to identify a sector?
  - Physical position
    - Sectors is a vector [cylinder, surface, sector]
    - Not used anymore
    - OS/BIOS must deal with bad sectors
  - Logical Block Addressing (LBA)
    - Every sector has integer address
    - Controller translates from address ⇒ physical position
    - Shields OS from structure of disk



Disk Structure

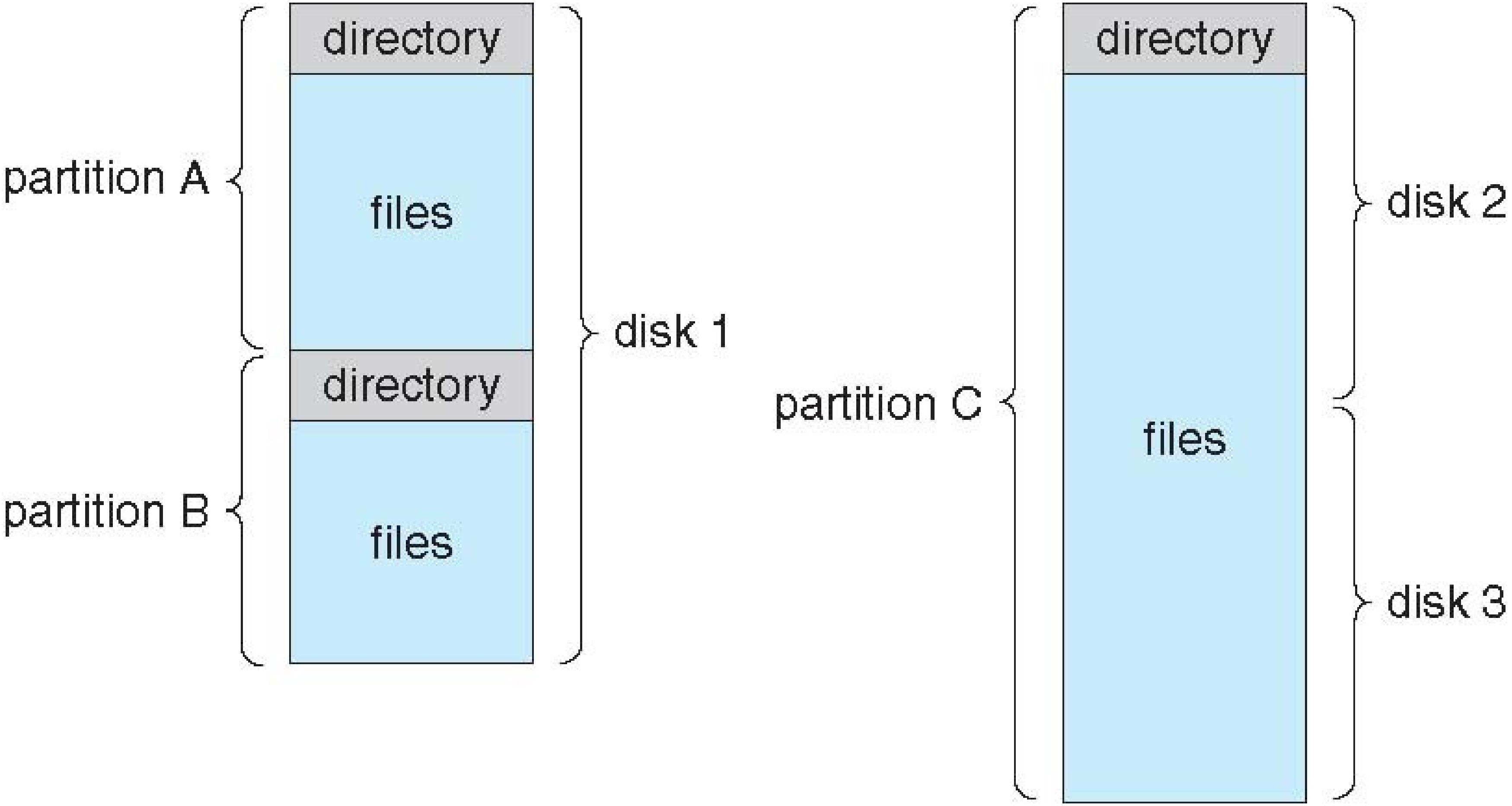
- Disk can be Subdivided into **Partitions**
- Disks or Partitions can be **RAID** protected against failure
- Disk or Partition can be Used **Raw** – without a file system, or **formatted** with a file system
- Partitions also Known as Minidisks, Slices
- Entity Containing File System known as a **Volume**
- Each Volume Containing File System also tracks that file system's info in **Device Directory** or **Volume Table of Contents**
- **General-Purpose FS** & **special-Purpose FS**, frequently all within same OS or computer



What Does File System Need?

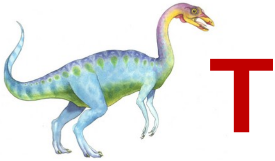
- Track free disk blocks
  - Need to know where to put newly written data
- Track which blocks contain data for which files
  - Need to know where to read a file from
- Track files in a directory
  - Find list of file's blocks given its name
- Where do we maintain all of this?
  - Somewhere on disk

A Typical FS Organization



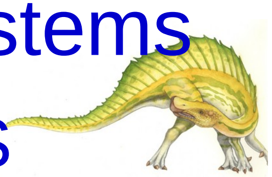
Types of File Systems

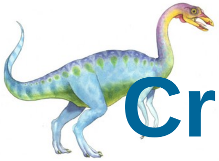
- We mostly Focus on General-Purpose FS
- Systems may have several types of FS
  - Some general- and some special- purpose
- Consider Solaris has:
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems



Types of File Systems (Cont.)

- Disk File Systems
  - Ext, ext2/3, FAT, HFS, NTFS, ZFS
- File Systems with Built-in Fault Tolerance
  - BTRFS
- File Systems Optimized for SSDs/Flash
  - JFFS, TrueFFS
- Distributed File Systems
- Distributed Parallel File Systems
- Distributed Parallel Fault-Tolerant File Systems
  - Ceph, Google File System (GFS), Hadoop FS





Critical Factors in File System Design

- (Hard) Disks Performance !!!
  - Maximize sequential access, minimize seeks
- Open before Read/Write
  - Can perform protection checks and look up where the actual file resource are, in advance
- Size is determined as they are used !!!
  - Can write (or read zeros) to expand the file
  - Start small and grow, need to make room
- Organized into directories
  - What data structure (on disk) for that?
- Need to carefully allocate / free blocks

Operations Performed on Directory

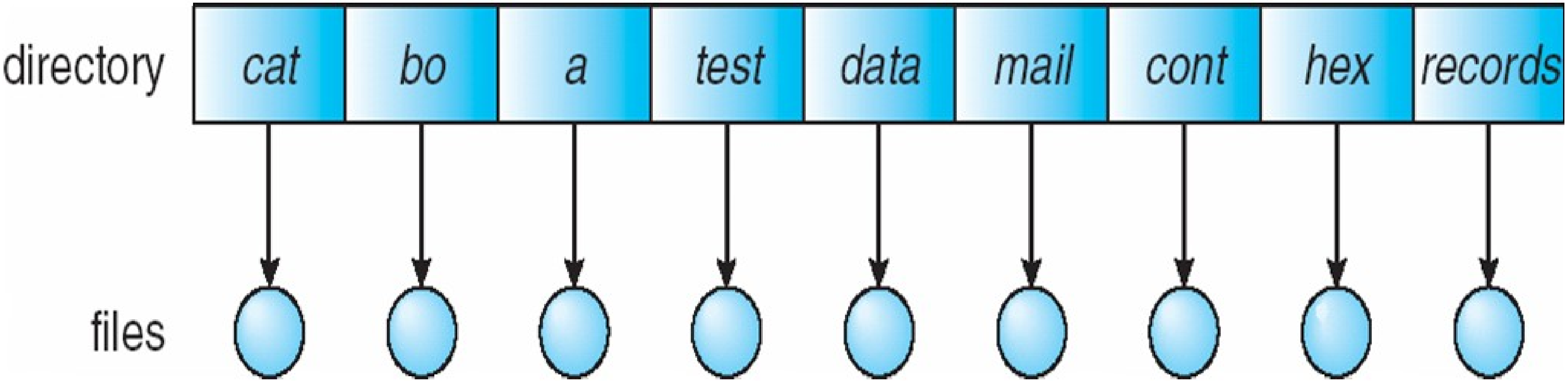
- Directory
  - Symbol table that translates file names into their directory entries
- Operations on Directory
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse file system

Directory Organization

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

Single-Level Directory

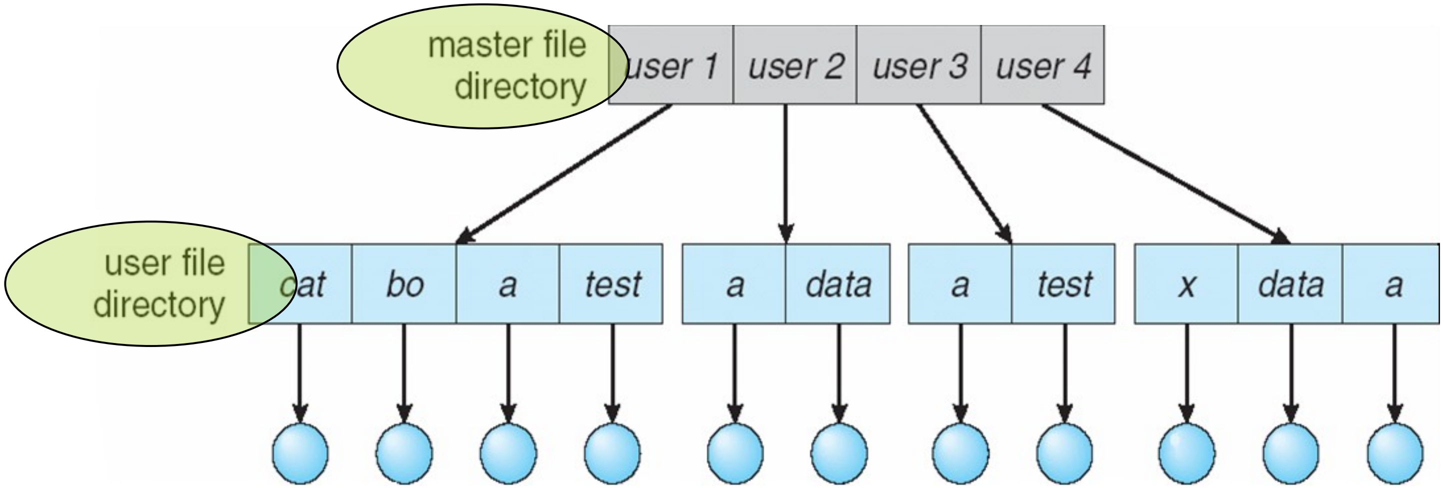
- A Single Directory for All Users



- Naming Problem
- Grouping Problem

Two-Level Directory

- Separate Directory for each User

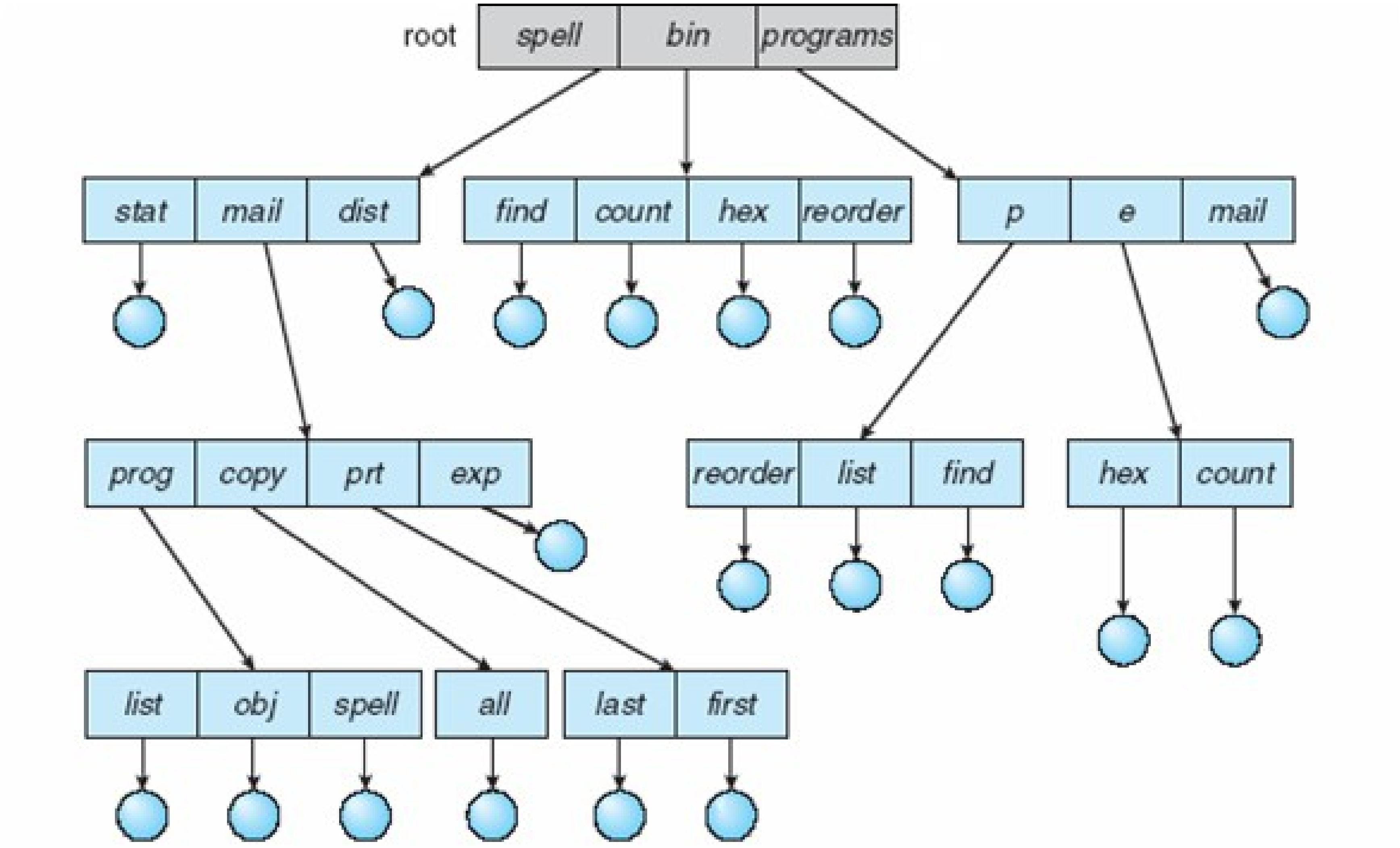


- Path name
- Can have same file name for different user
- Efficient searching

Two-Level Directory

- Pros
  - Solves name-collision problem
  - Creates protections between users
- Cons
  - Sharing a file between two users not allowed
  - No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont)

- **Absolute** or **Relative** Path Name
- Creating a new file is done in current directory
- Delete a file

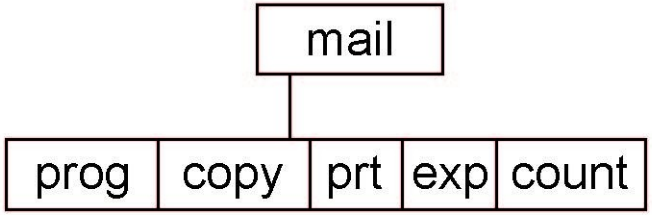
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

Example: if in current directory **`/mail`**

```
mkdir count
```

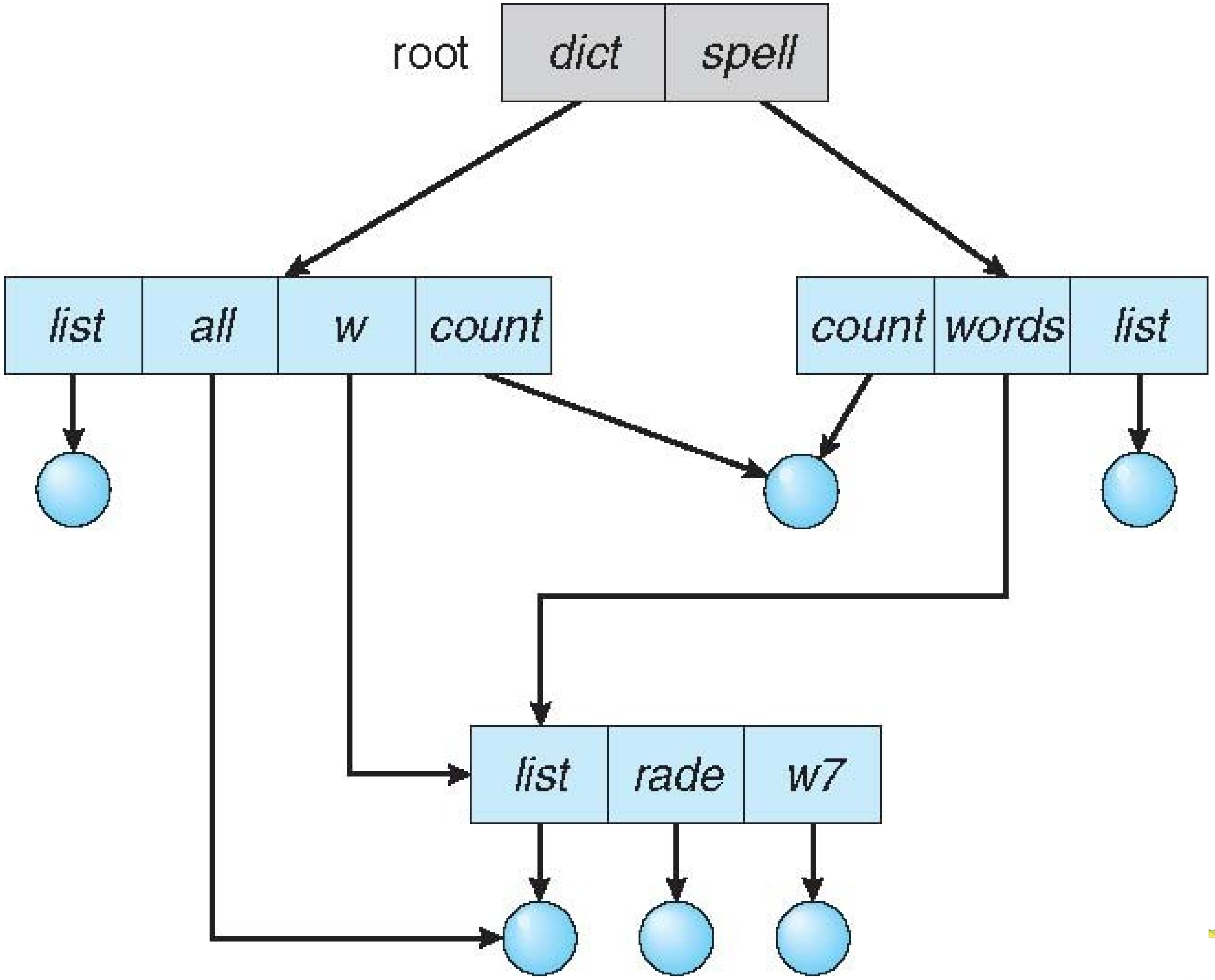

mail

prog	copy	prt	exp	count

- Deleting "mail"⇒Deleting entire subtree rooted by "mail"

Acyclic-Graph Directories

- Shared Subdirectories and Files



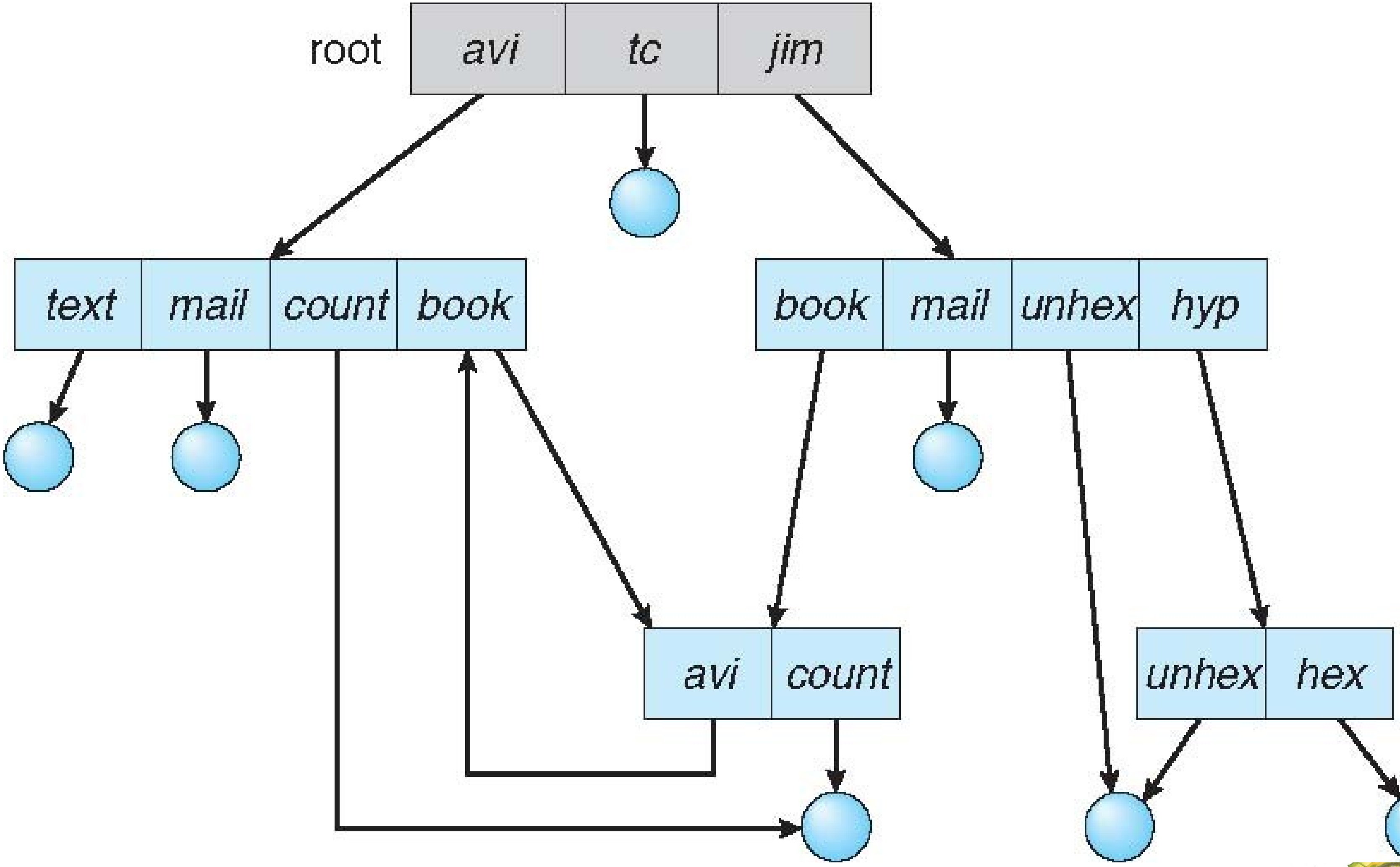
Acyclic-Graph Directories (cont.)

- Two different names (aliasing)
- If ***dict*** deletes ***list*** ⇒ 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
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

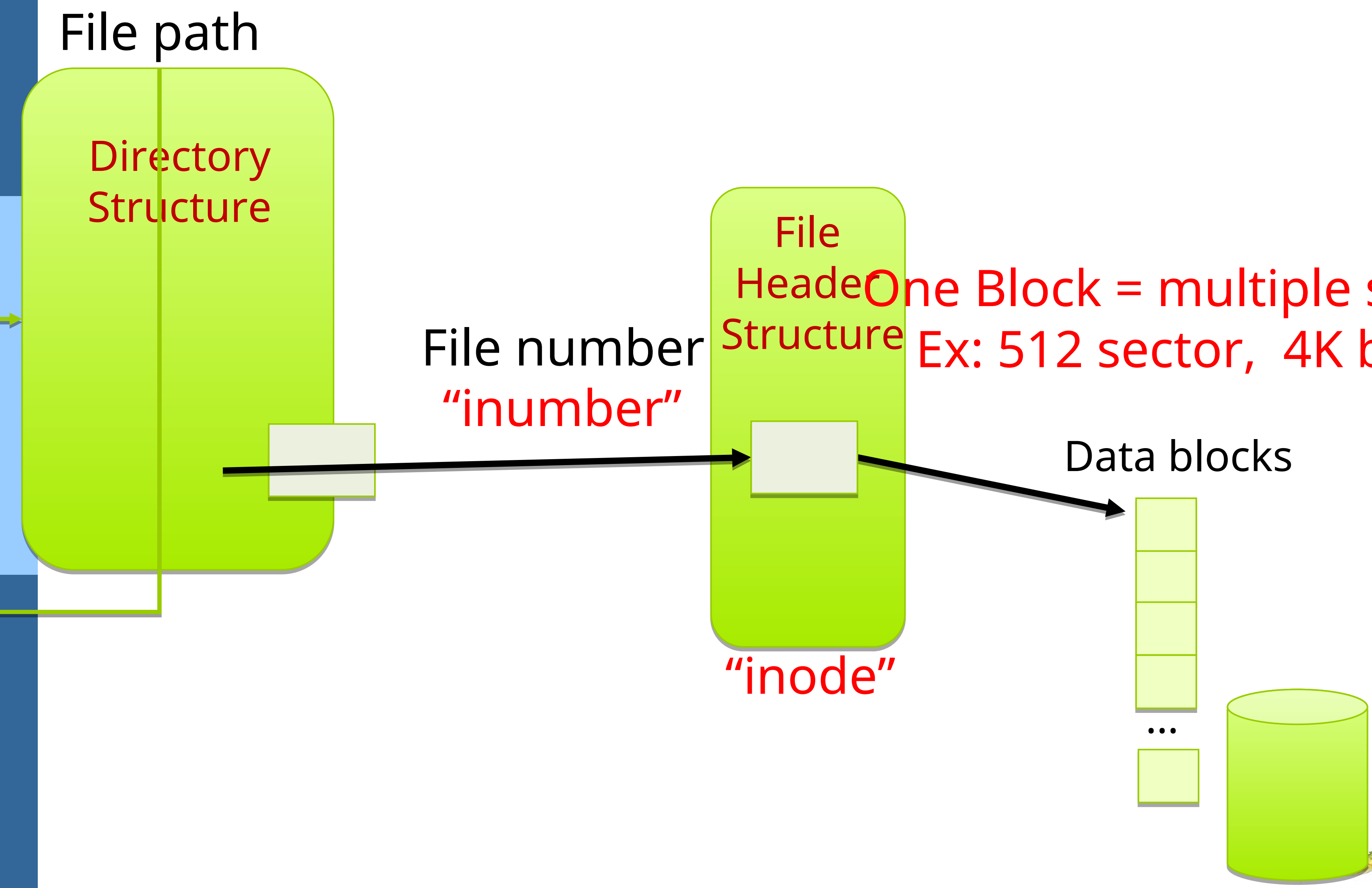
General Graph Directory



General Graph Directory (Cont.)

- 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

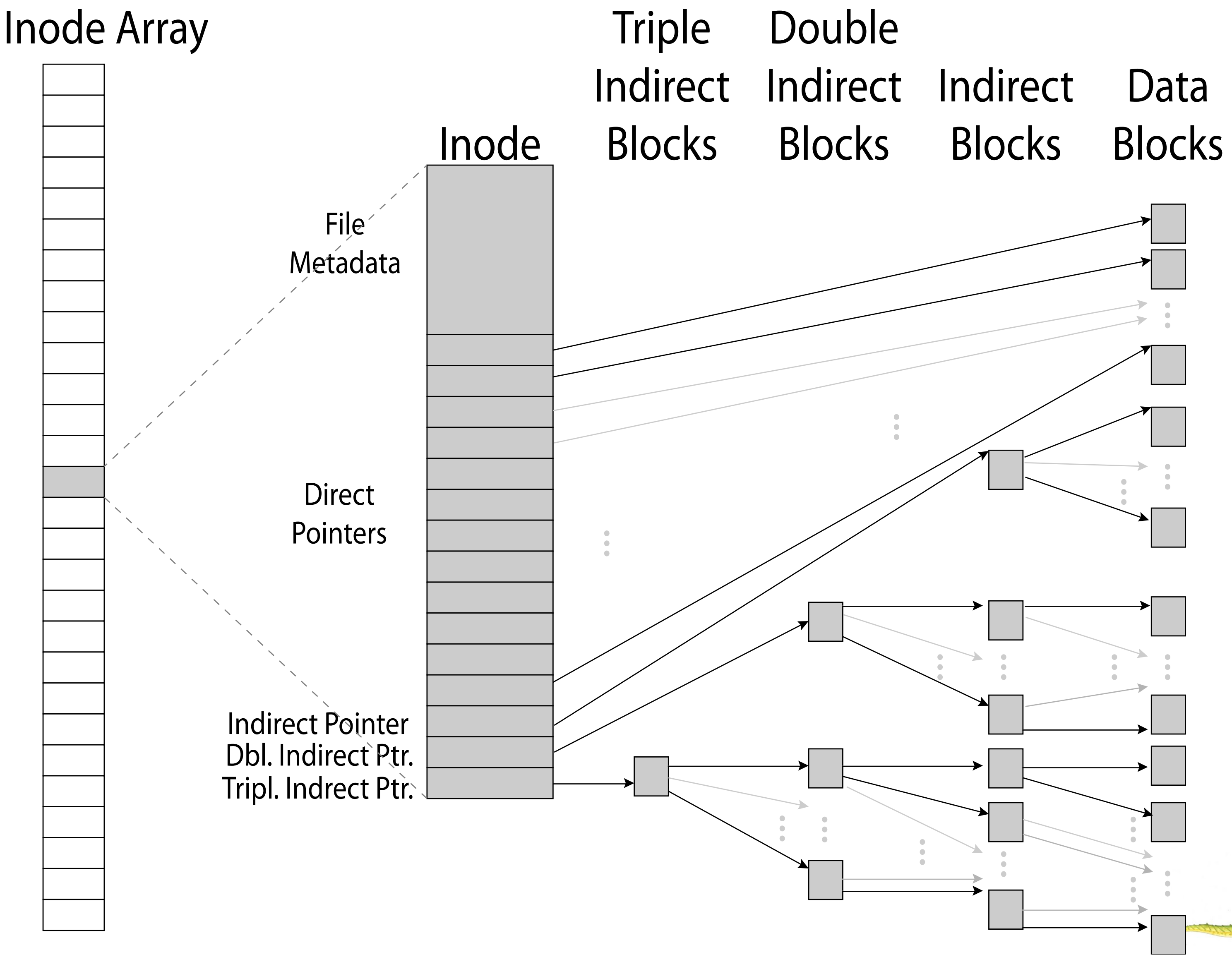
Components of a File System



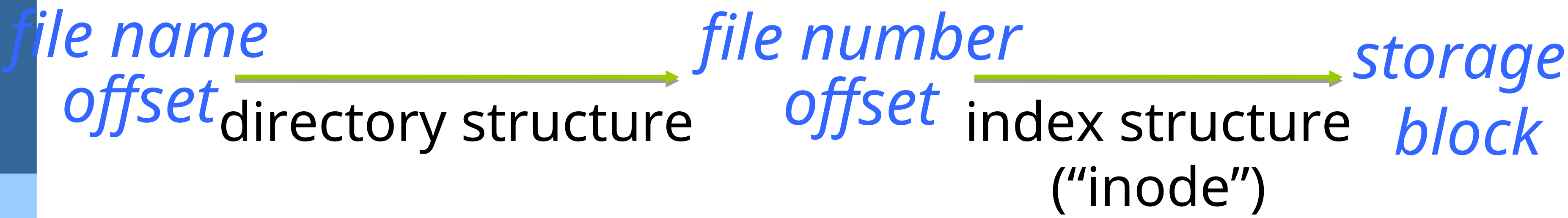
One Block = multiple sector

Ex: 512 sector, 4K block

Example of BSD/Linux-like Inode structure



Components of a File System



file name
offset

directory structure

file number
offset

index structure
(“inode”)

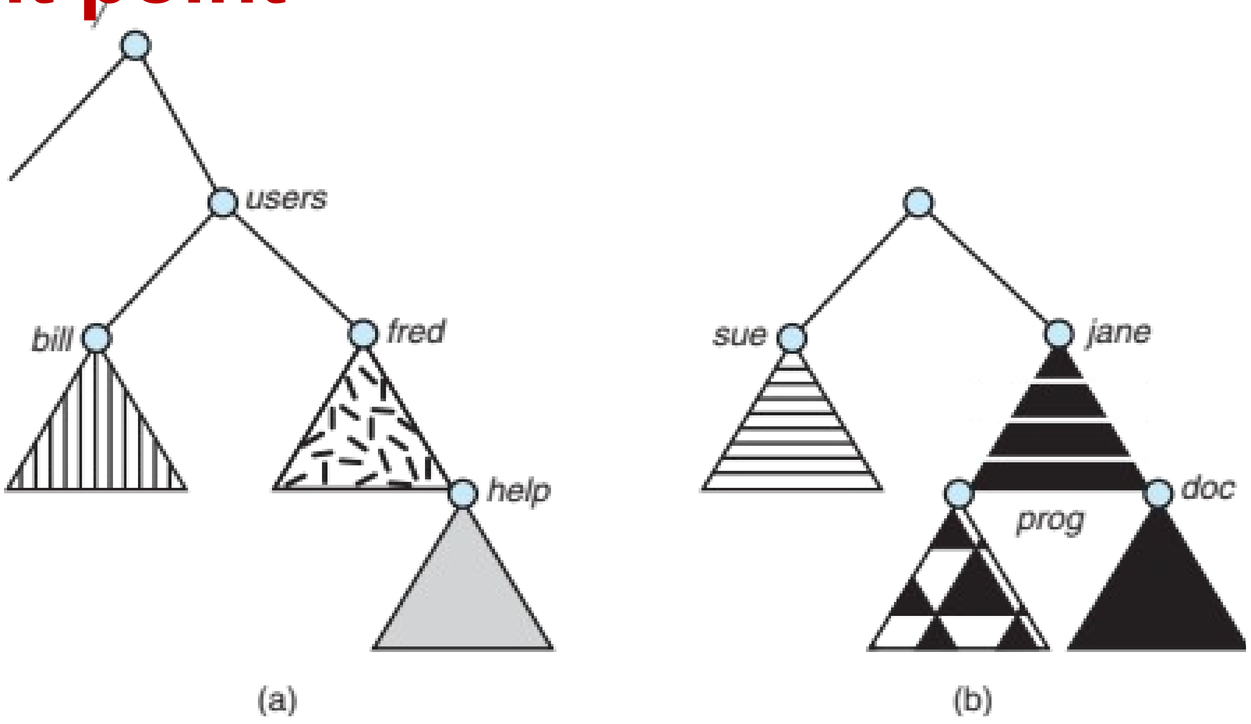
storage
block

- Open performs *Name Resolution*
  - Translates path name into a “file number”
- Read and Write operate on the file number
  - Use file number as an “index” to locate the blocks

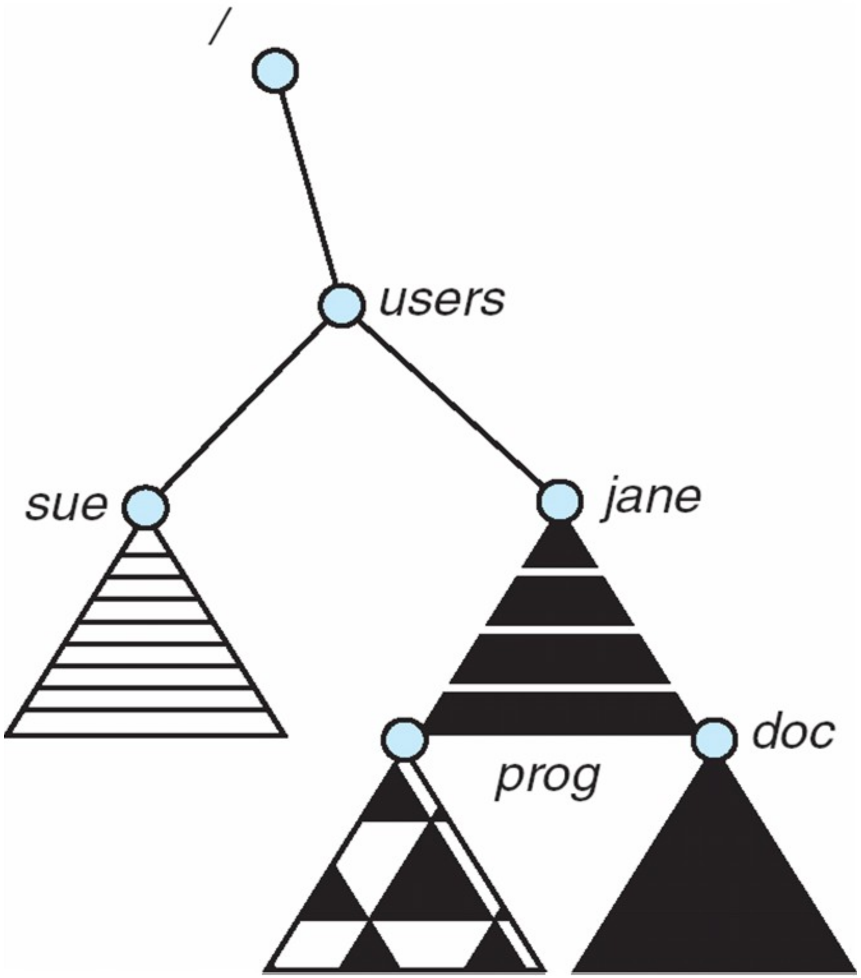
- **Four Components:**
  - **directory, index structure, storage blocks, free space map**

File System Mounting

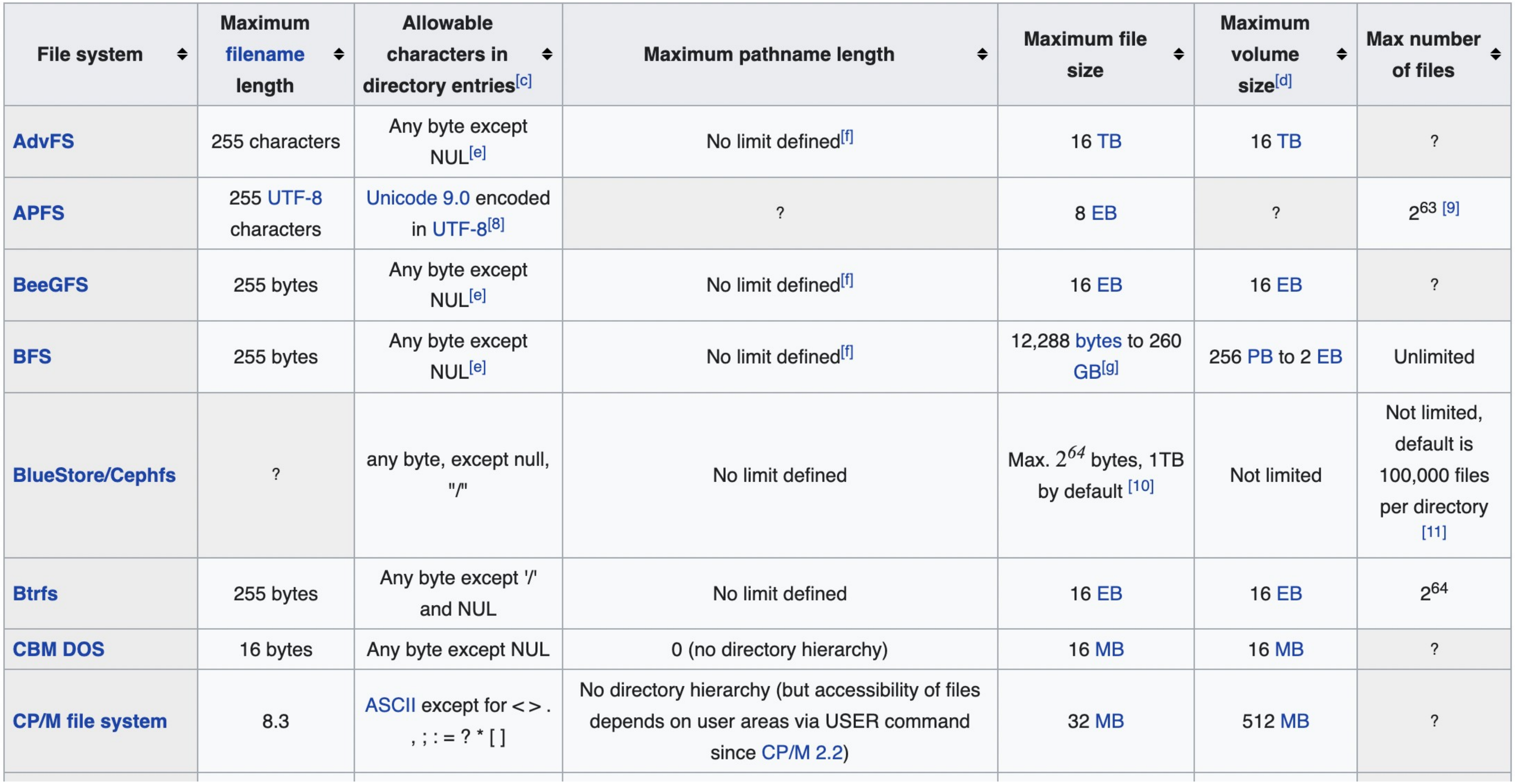
- A File System must be **mounted** before it can be accessed
- A Unmounted File System mounted at a **mount point**



Mount Point



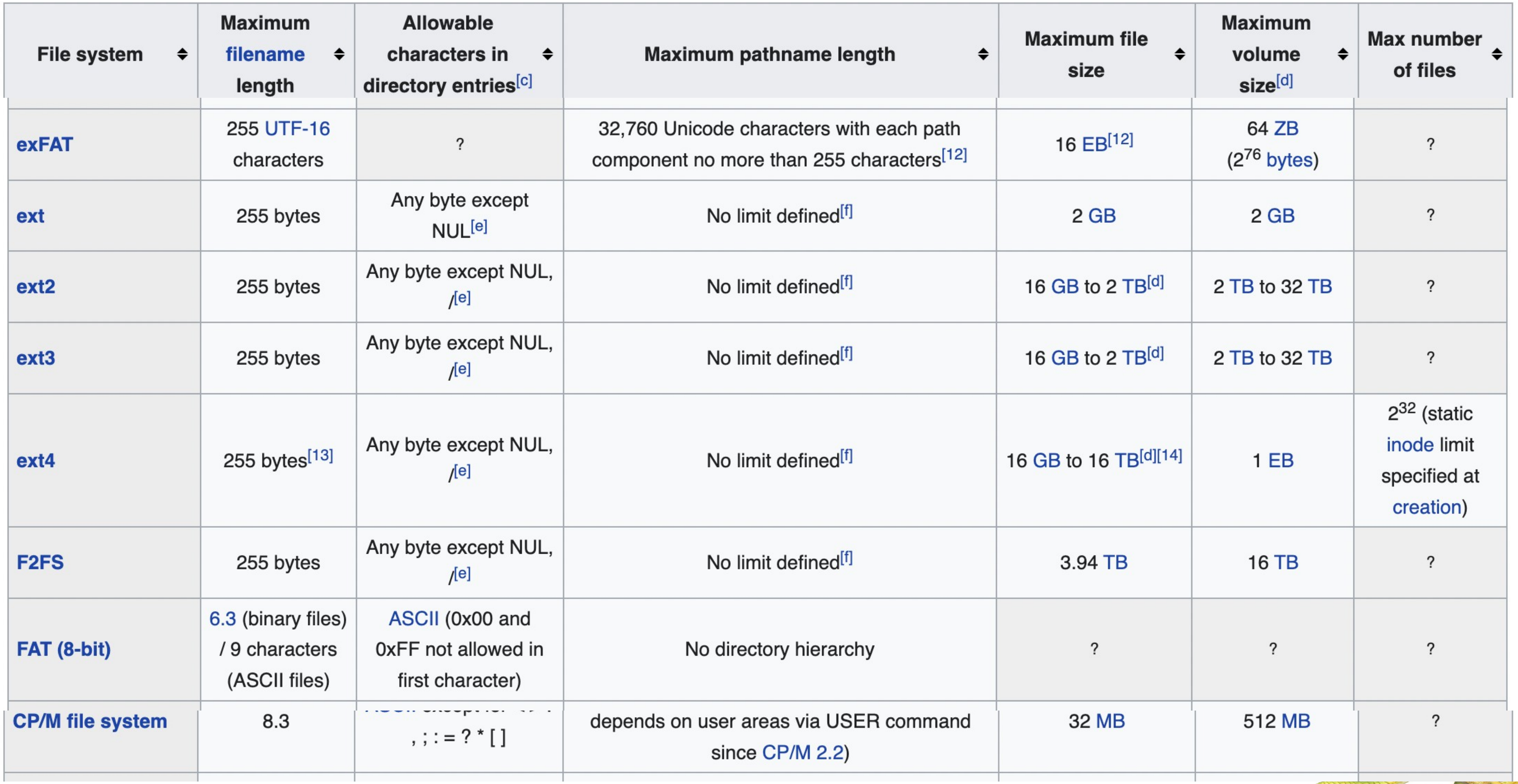
Comparison of Filesystems



File system	Maximum filename length	Allowable characters in directory entries[c]	Maximum pathname length	Maximum file size	Maximum volume size[d]	Max number of files
AdvFS	255 characters	Any byte except NUL[e]	No limit defined[f]	16 TB	16 TB	?
APFS	255 UTF-8 characters	Unicode 9.0 encoded in UTF-8[8]	?	8 EB	?	2^{63} [9]
BeeGFS	255 bytes	Any byte except NUL[e]	No limit defined[f]	16 EB	16 EB	?
BFS	255 bytes	Any byte except NUL[e]	No limit defined[f]	12,288 bytes to 260 GB[g]	256 PB to 2 EB	Unlimited
BlueStore/Cephfs	?	any byte, except null, "/"	No limit defined	Max. 2^{64} bytes, 1TB by default [10]	Not limited	Not limited, default is 100,000 files per directory [11]
Btrfs	255 bytes	Any byte except '/' and NUL	No limit defined	16 EB	16 EB	2^{64}
CBM DOS	16 bytes	Any byte except NUL	0 (no directory hierarchy)	16 MB	16 MB	?
CP/M file system	8.3	ASCII except for < > . , ; : = ? * [ ]	No directory hierarchy (but accessibility of files depends on user areas via USER command since CP/M 2.2)	32 MB	512 MB	?

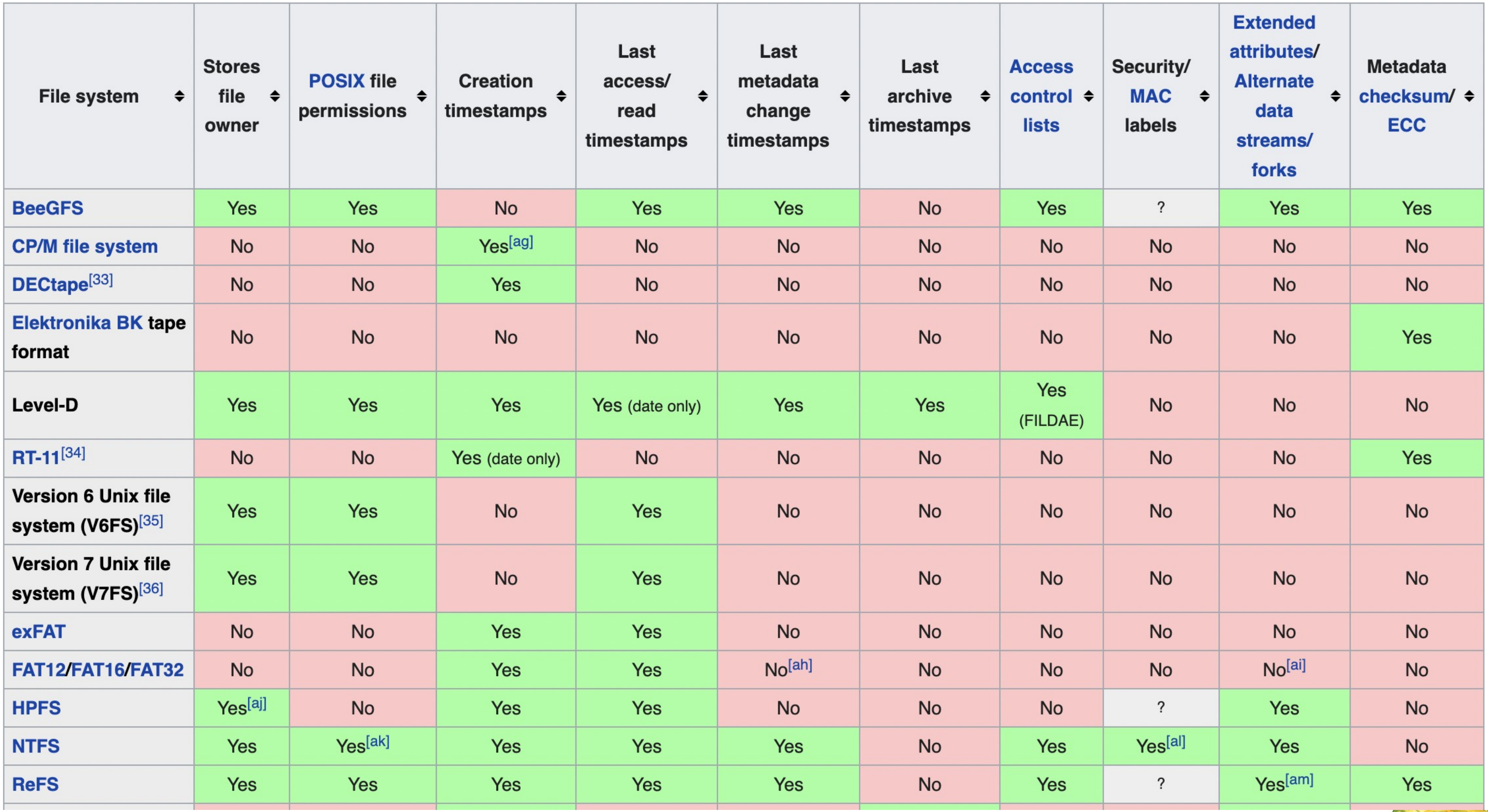
https://en.wikipedia.org/wiki/Comparison_of_file_systems

Comparison of Filesystems (cont.)



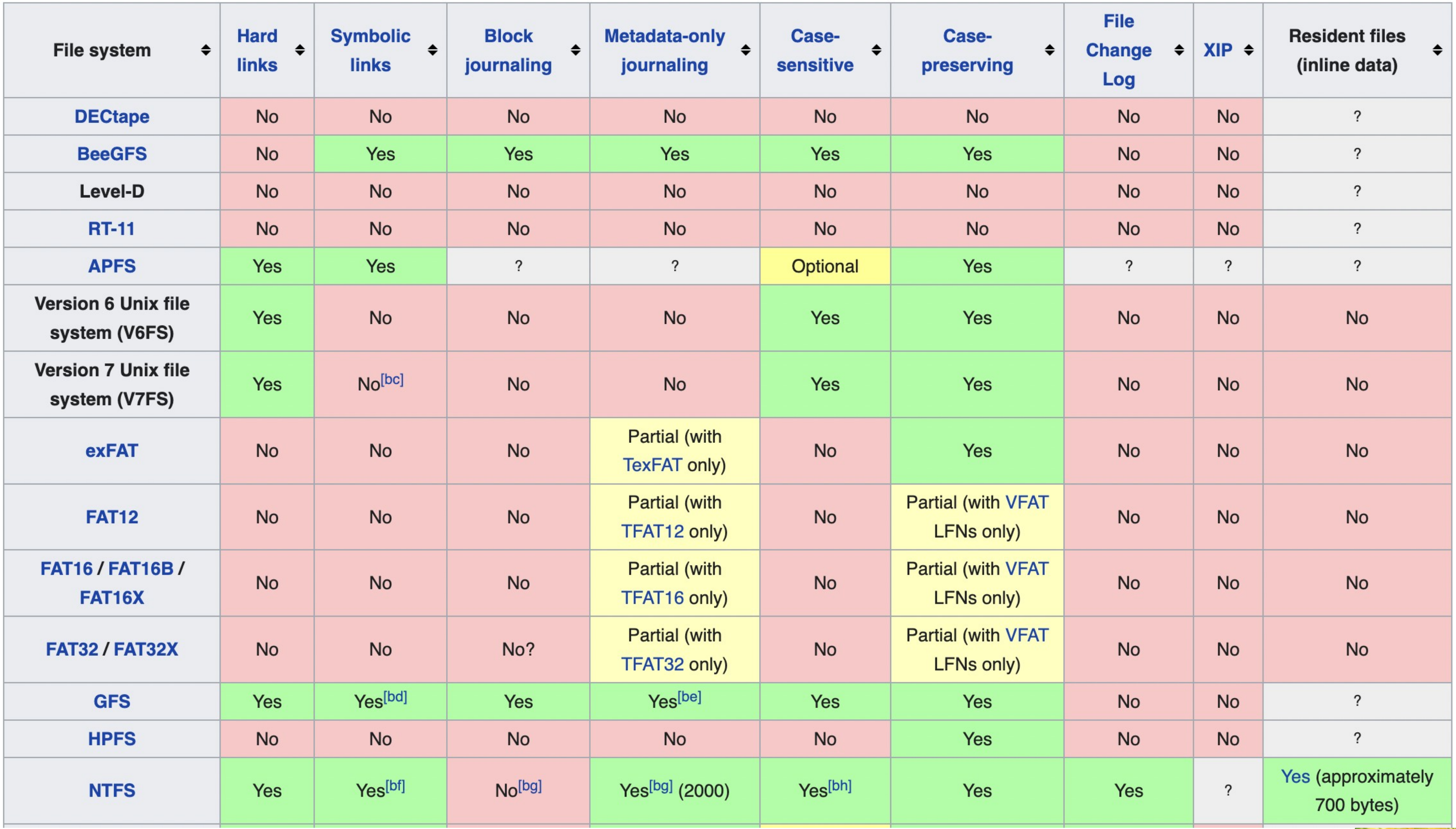
File system	Maximum filename length	Allowable characters in directory entries[c]	Maximum pathname length	Maximum file size	Maximum volume size[d]	Max number of files
exFAT	255 UTF-16 characters	?	32,760 Unicode characters with each path component no more than 255 characters[12]	16 EB[12]	64 ZB (2^{76} bytes)	?
ext	255 bytes	Any byte except NUL[e]	No limit defined[f]	2 GB	2 GB	?
ext2	255 bytes	Any byte except NUL, /[e]	No limit defined[f]	16 GB to 2 TB[d]	2 TB to 32 TB	?
ext3	255 bytes	Any byte except NUL, /[e]	No limit defined[f]	16 GB to 2 TB[d]	2 TB to 32 TB	?
ext4	255 bytes[13]	Any byte except NUL, /[e]	No limit defined[f]	16 GB to 16 TB[d][14]	1 EB	2^{32} (static inode limit specified at creation)
F2FS	255 bytes	Any byte except NUL, /[e]	No limit defined[f]	3.94 TB	16 TB	?
FAT (8-bit)	6.3 (binary files) / 9 characters (ASCII files)	ASCII (0x00 and 0xFF not allowed in first character)	No directory hierarchy	?	?	?
CP/M file system	8.3	ASCII except for < > . , ; : = ? * [ ]	depends on user areas via USER command since CP/M 2.2)	32 MB	512 MB	?

Comparison of Filesystems: Metadata



File system	Stores file owner	POSIX file permissions	Creation timestamps	Last access/ read timestamps	Last metadata change timestamps	Last archive timestamps	Access control lists	Security/ MAC labels	Extended attributes/ Alternate data streams/ forks	Metadata checksum/ ECC
BeeGFS	Yes	Yes	No	Yes	Yes	No	Yes	?	Yes	Yes
CP/M file system	No	No	Yes[ag]	No	No	No	No	No	No	No
DECtape[33]	No	No	Yes	No	No	No	No	No	No	No
Elektronika BK tape format	No	No	No	No	No	No	No	No	No	Yes
Level-D	Yes	Yes	Yes	Yes (date only)	Yes	Yes	Yes (FILDAE)	No	No	No
RT-11[34]	No	No	Yes (date only)	No	No	No	No	No	No	Yes
Version 6 Unix file system (V6FS)[35]	Yes	Yes	No	Yes	No	No	No	No	No	No
Version 7 Unix file system (V7FS)[36]	Yes	Yes	No	Yes	No	No	No	No	No	No
exFAT	No	No	Yes	Yes	No	No	No	No	No	No
FAT12/FAT16/FAT32	No	No	Yes	Yes	No[ah]	No	No	No	No[ai]	No
HPFS	Yes[aj]	No	Yes	Yes	No	No	No	?	Yes	No
NTFS	Yes	Yes[ak]	Yes	Yes	Yes	No	Yes	Yes[al]	Yes	No
ReFS	Yes	Yes	Yes	Yes	Yes	No	Yes	?	Yes[am]	Yes

Comparison of Filesystems: File Capabilities



File system	Hard links	Symbolic links	Block journaling	Metadata-only journaling	Case-sensitive	Case-preserving	File Change Log	XIP	Resident files (inline data)
DECtape	No	No	No	No	No	No	No	No	?
BeeGFS	No	Yes	Yes	Yes	Yes	Yes	No	No	?
Level-D	No	No	No	No	No	No	No	No	?
RT-11	No	No	No	No	No	No	No	No	?
APFS	Yes	Yes	?	?	Optional	Yes	?	?	?
Version 6 Unix file system (V6FS)	Yes	No	No	No	Yes	Yes	No	No	No
Version 7 Unix file system (V7FS)	Yes	No[bc]	No	No	Yes	Yes	No	No	No
exFAT	No	No	No	Partial (with TexFAT only)	No	Yes	No	No	No
FAT12	No	No	No	Partial (with TFAT12 only)	No	Partial (with VFAT LFNs only)	No	No	No
FAT16 / FAT16B / FAT16X	No	No	No	Partial (with TFAT16 only)	No	Partial (with VFAT LFNs only)	No	No	No
FAT32 / FAT32X	No	No	No?	Partial (with TFAT32 only)	No	Partial (with VFAT LFNs only)	No	No	No
GFS	Yes	Yes[bd]	Yes	Yes[be]	Yes	Yes	No	No	?
HPFS	No	No	No	No	No	Yes	No	No	?
NTFS	Yes	Yes[bf]	No[bg]	Yes[bg] (2000)	Yes[bh]	Yes	Yes	?	Yes (approximately 700 bytes)

Reading Assignment

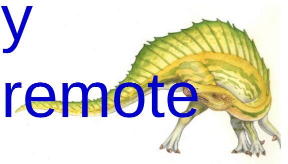
- File Sharing
- Protection

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

File Sharing – Remote File Systems

- Uses networking to allow FS access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing



File Sharing – Failure Modes

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 5 process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes only visible to sessions starting after the file is closed

Protection

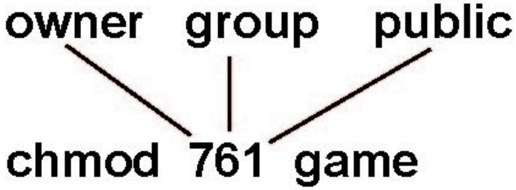
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

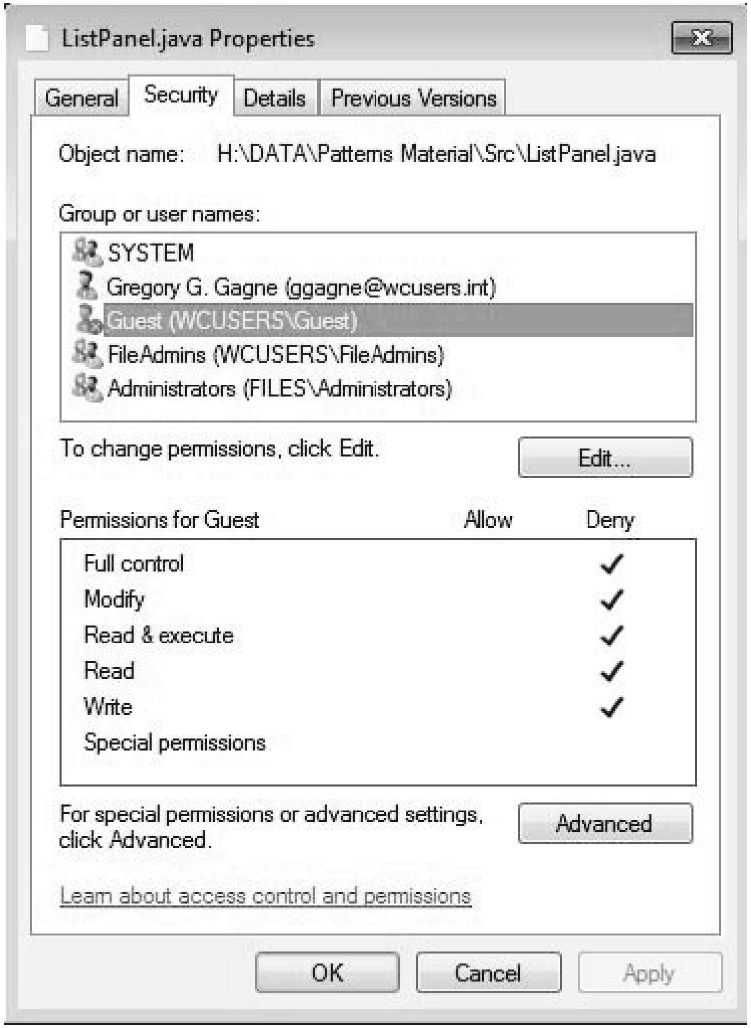
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



- Attach a group to a file

```
chgrp     G     game
```

Windows 7 Access-Control List Management



ListPanel.java Properties

Object name: H:\DATA\Patterns Material\Src\ListPanel.java

Group or user names:

- SYSTEM
- Gregory G. Gagne (ggagne@wcusers.int)
- Guest (WCUSERS\Guest)
- FileAdmins (WCUSERS\FileAdmins)
- Administrators (FILES\Administrators)

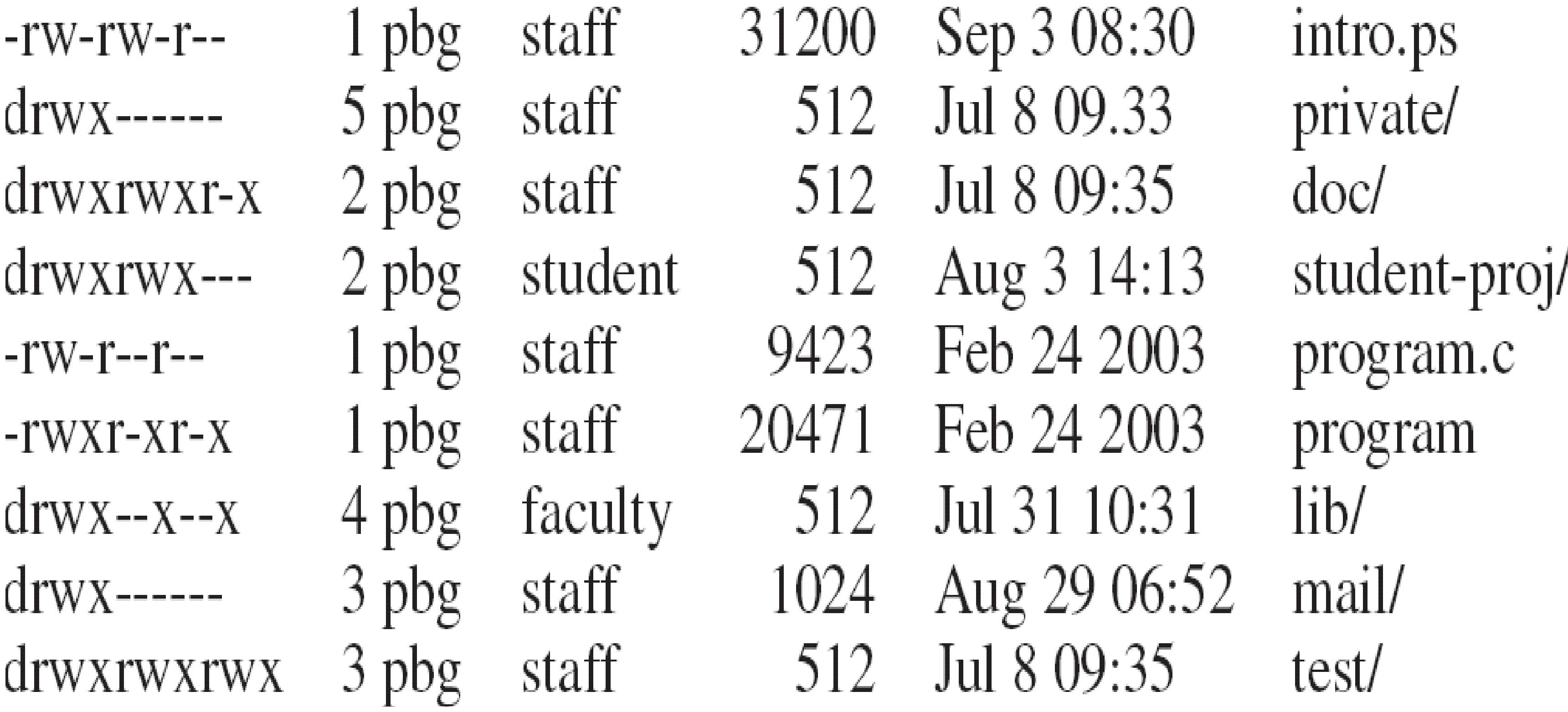
To change permissions, click Edit.

Permissions for Guest	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
Read		✓
Write		✓
Special permissions		

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

A Sample UNIX Directory Listing

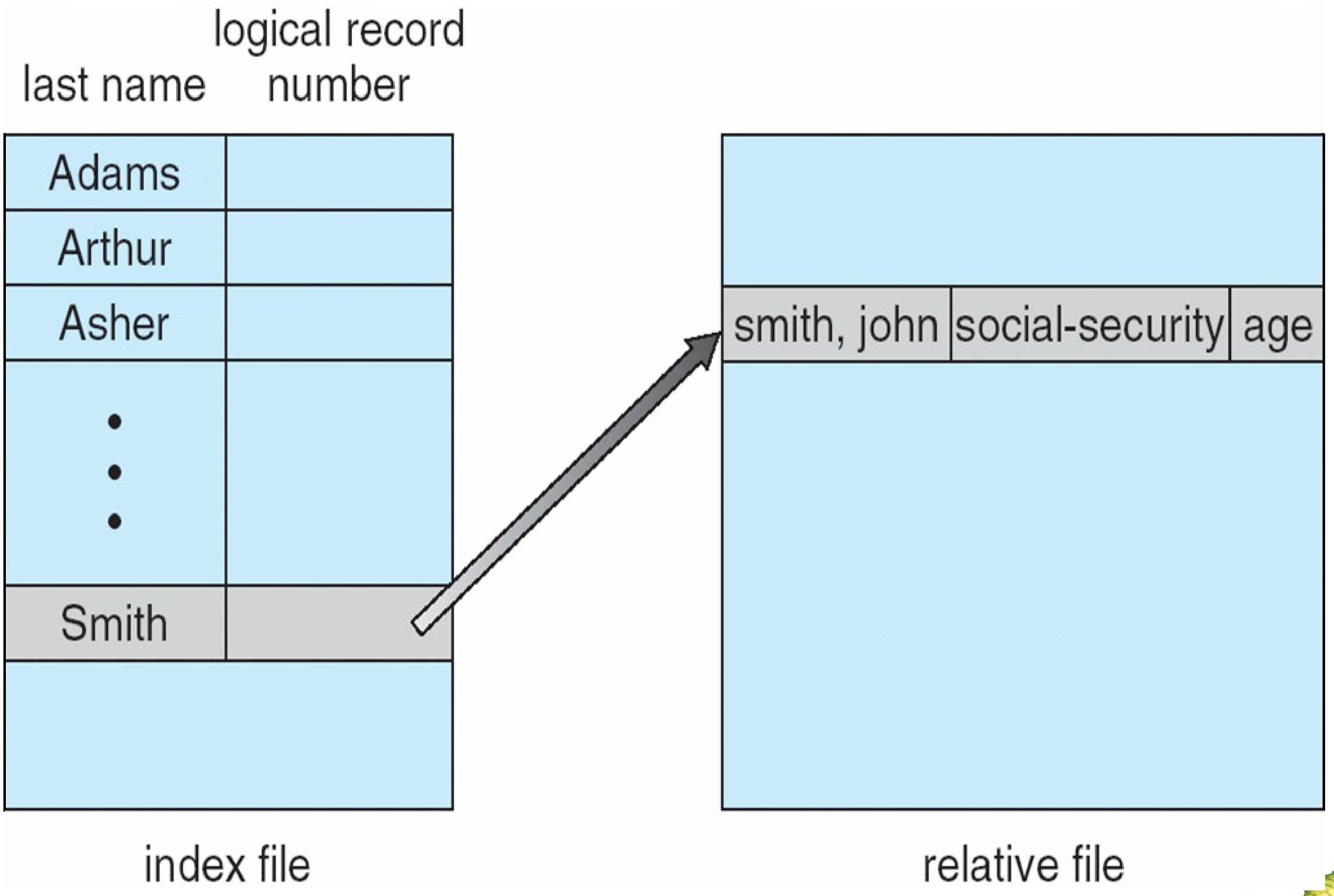


```
-rw-rw-r--    1 pbg  staff      31200  Sep 3 08:30    intro.ps
drwx------    5 pbg  staff        512  Jul 8 09.33    private/
drwxrwxr-x    2 pbg  staff        512  Jul 8 09:35    doc/
drwxrwx---    2 pbg  student      512  Aug 3 14:13    student-proj/
-rw-r--r--    1 pbg  staff       9423  Feb 24 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/
```

Other Access Methods

- Can be Built on top of Base Methods
- General Involve Creation of an index for File
- Keep index in memory for fast determination of location of data to be operated on
- If too large, index (in memory) of 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 OS
- VMS operating system provides index and relative files as another example (see next slide)

Example of Index and Relative Files



End of Lecture 10