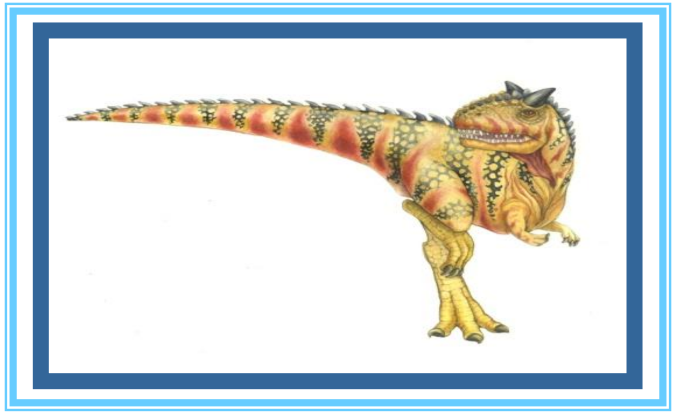
Chapter 11:
File-System Interface



Silberschatz, Galvin and Gagne ©2013

Chapter 11: File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File Sharing
- Protection

Objectives

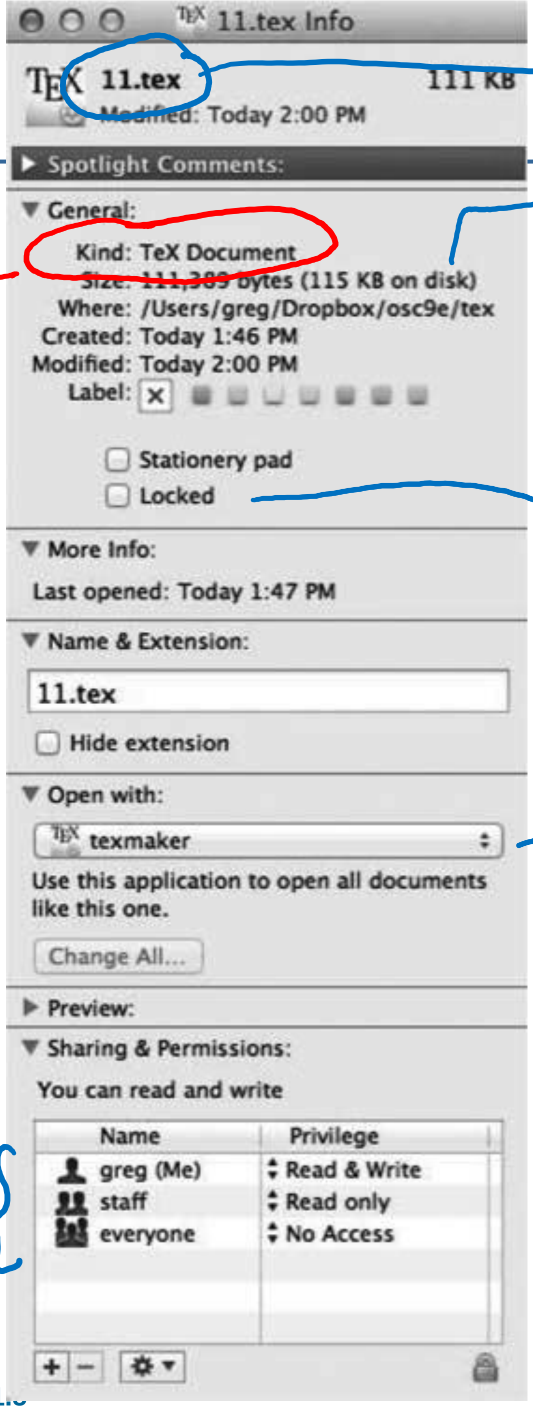
- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

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 info Window on Mac OS X

File attributes



11.tex Info

TEX 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.

Change All...

▶ Preview:

▼ Sharing & Permissions:

You can read and write

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

Type

name & extension

size

creation & Modification date

status

Application

Usernames

Priviledge / Access rights

Permissions

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
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

File Operations

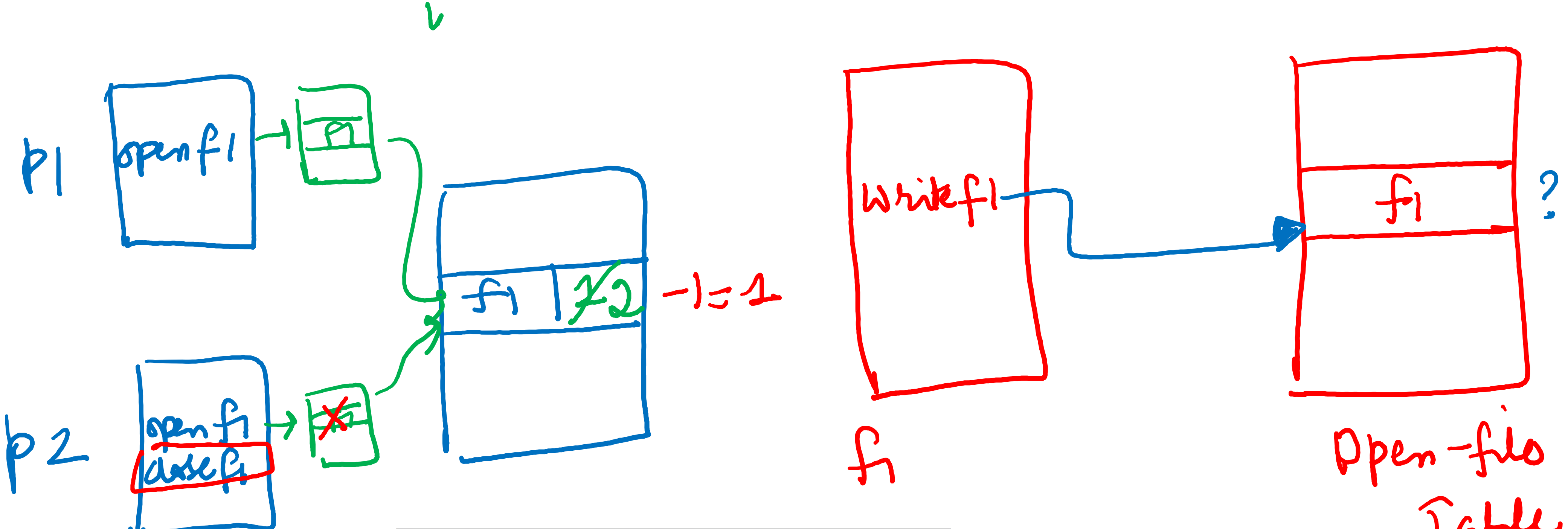
{ Search
Rename

- File is an **abstract data type**
- **Create**
- ✓ **Write –** at **write pointer** location
- ✓ **Read –** at **read pointer** location
- ✓ **Reposition within file - seek** (Random Access) → Read → Write
- ✓ **Delete**
- ✓ **Truncate** (keep file, delete contents)
- ***Open(F_i)*** – search the directory structure on disk for entry $\boldsymbol{F_i}$, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry $\boldsymbol{F_i}$ in memory to directory structure on disk

Open Files

- Several pieces of data are 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 the file: cache of data access information
  - Access rights: per-process access mode information

Open Files



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

Open File Locking

Mandatory
↓
deadlock
starvation

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

OS

(Windows)

(Linux)

file

P1, P2

File Structure

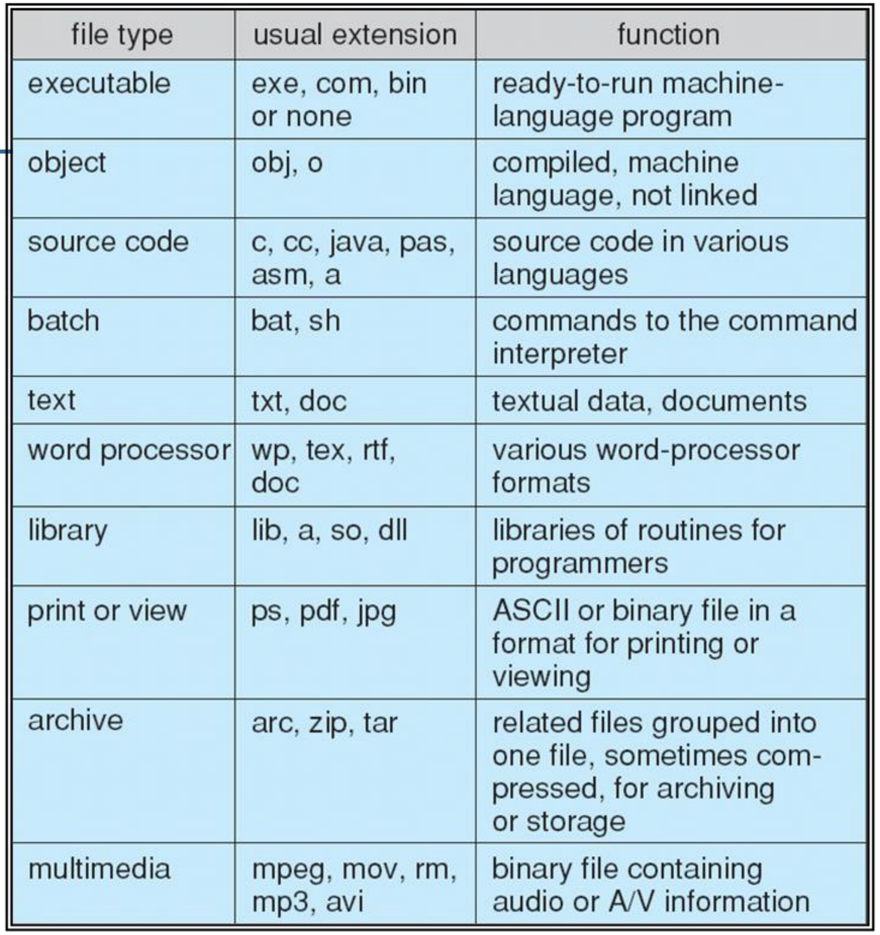
Cannot Support all file types'

OS

Code

- 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:
  - Operating system
  - Program ✓

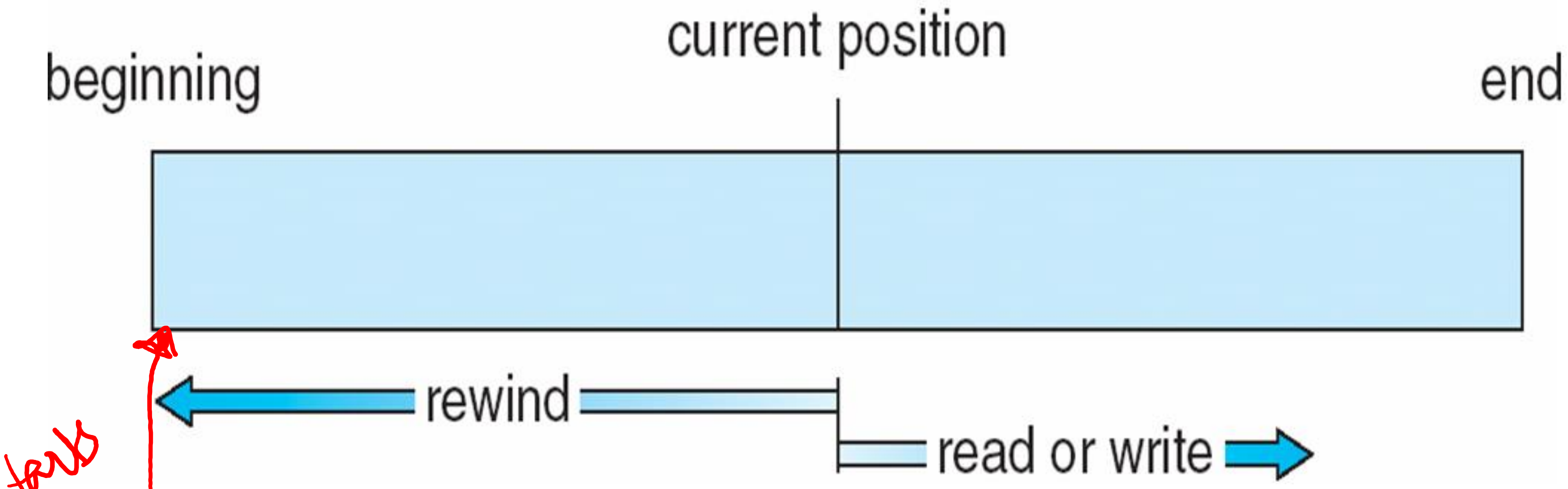
Application ✓



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

Sequential-access File

Limitations ?



Starts here always

Access Methods

- **Sequential Access**

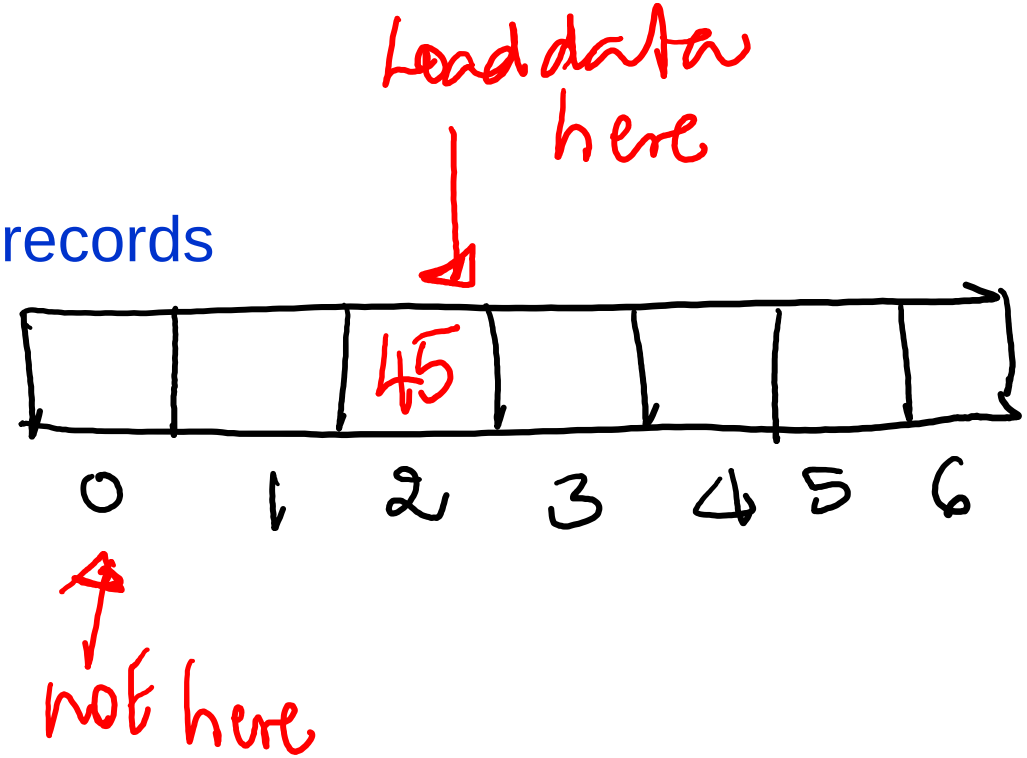
```
read next
write next
reset
```

no read after last write
(rewrite)

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

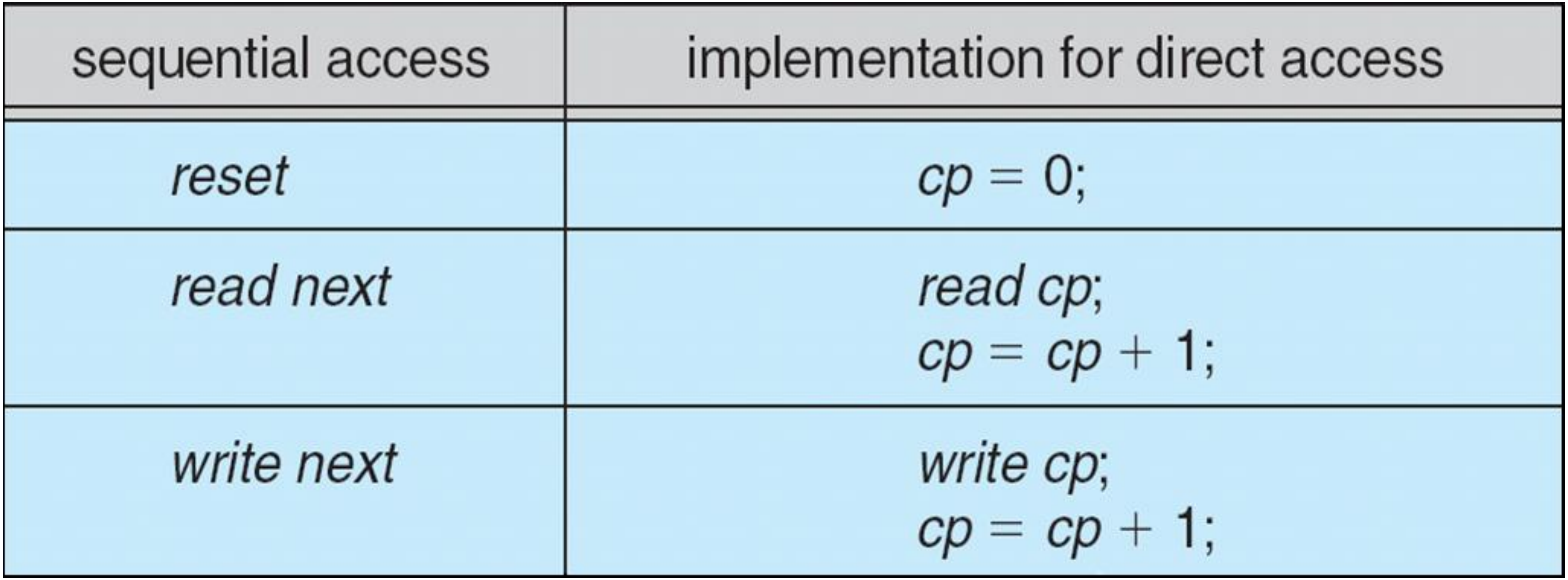
```
read n
write n
position to n
      read next
      write next
rewrite n
```

n = relative block number



- Relative block numbers allow OS to decide where file should be placed

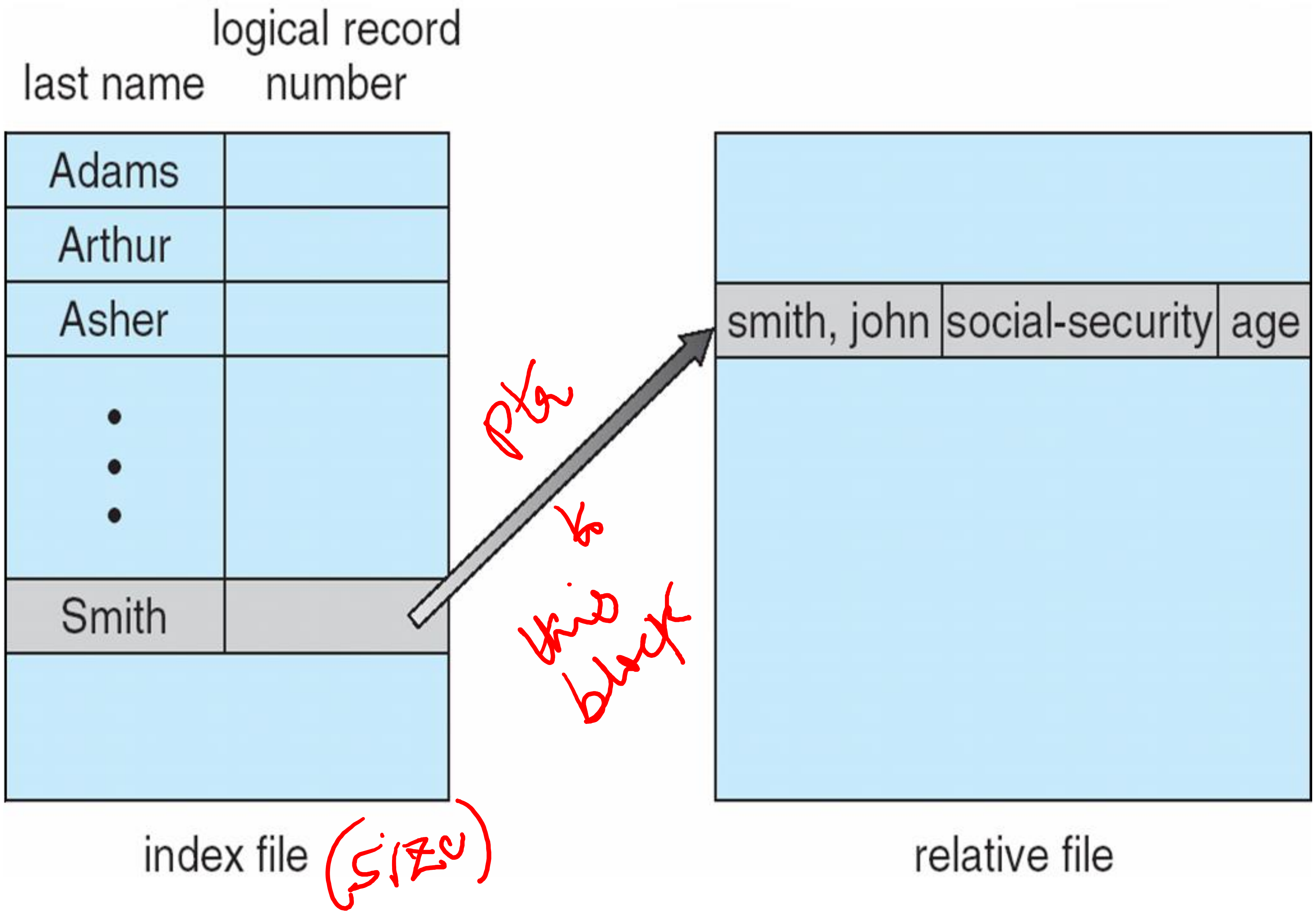
Simulation of Sequential Access on Direct-access File



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

Example of Index and Relative Files

Another way to access files



last name	logical record number
Adams	
Arthur	
Asher	
• • •	
Smith	

index file (size)

smith, john	social-security	age

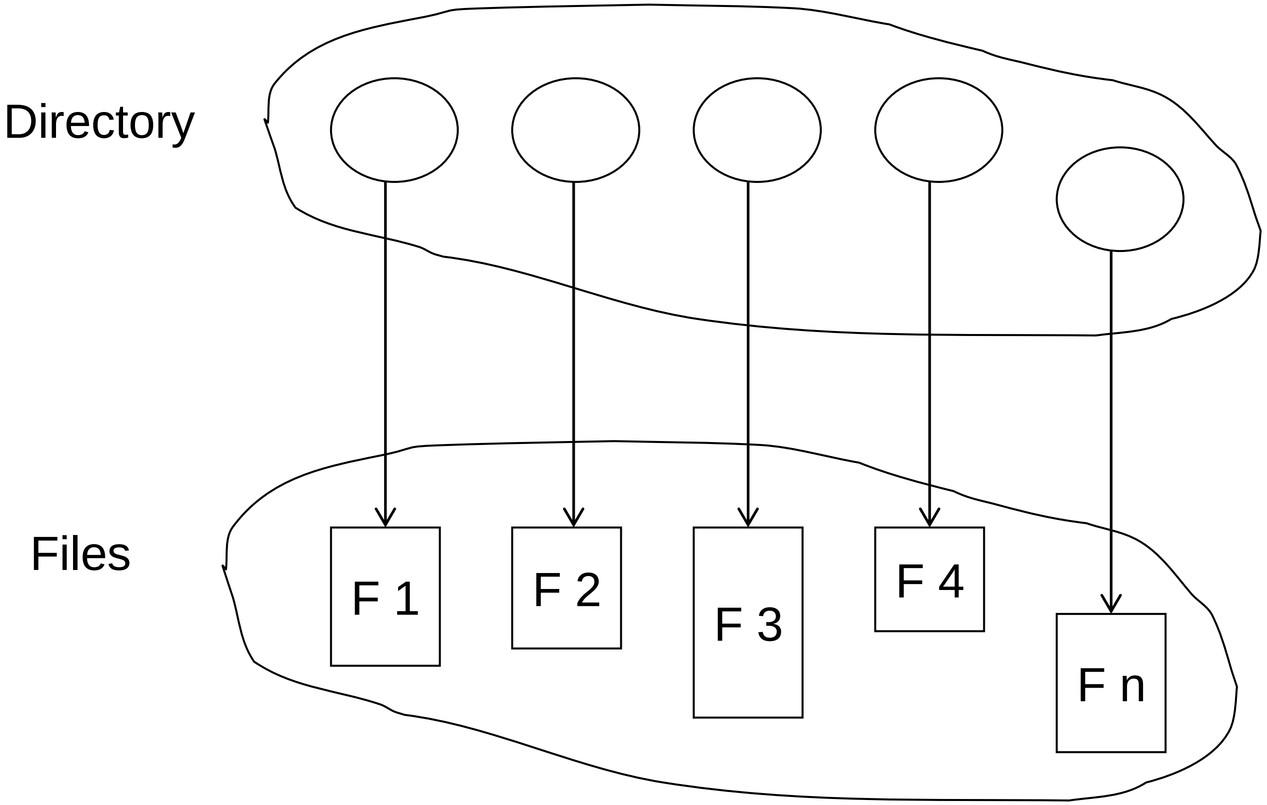
relative file

Additional

ptr to this block

Directory Structure

- A collection of nodes containing information about all files



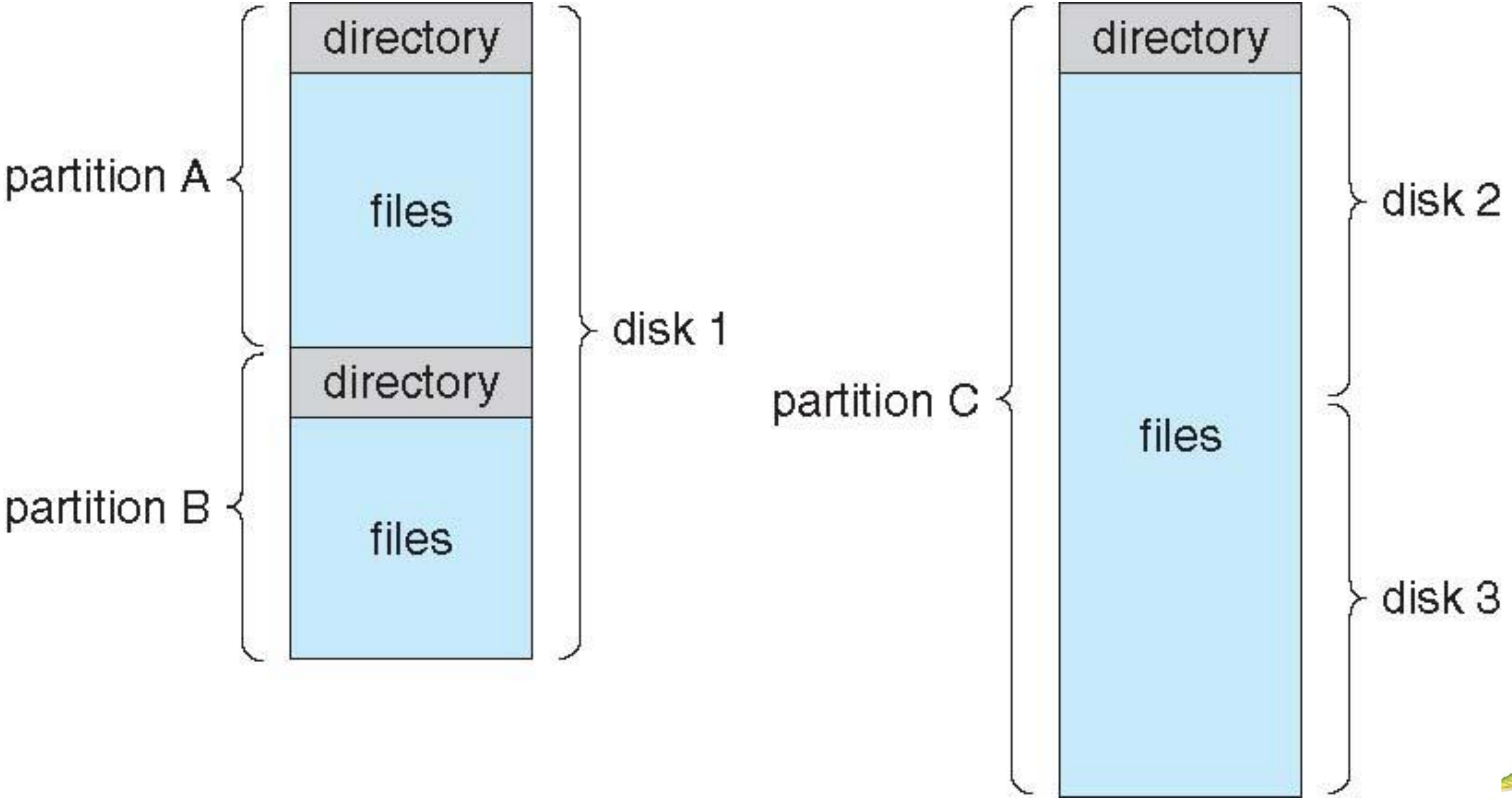
Both the directory structure and the files reside on disk

Disk Structure

Redundant Array of Independent Disks

- 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**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization



Operations Performed on $\boxed{\text{Directory}}$ Backup!

- Search for a file
- Create a file ① Position of files in directory changed | Reordering
- Delete a file ③ Gap in directory structure | defragmentation
- List a directory
- Rename a file ②
- Traverse the file system

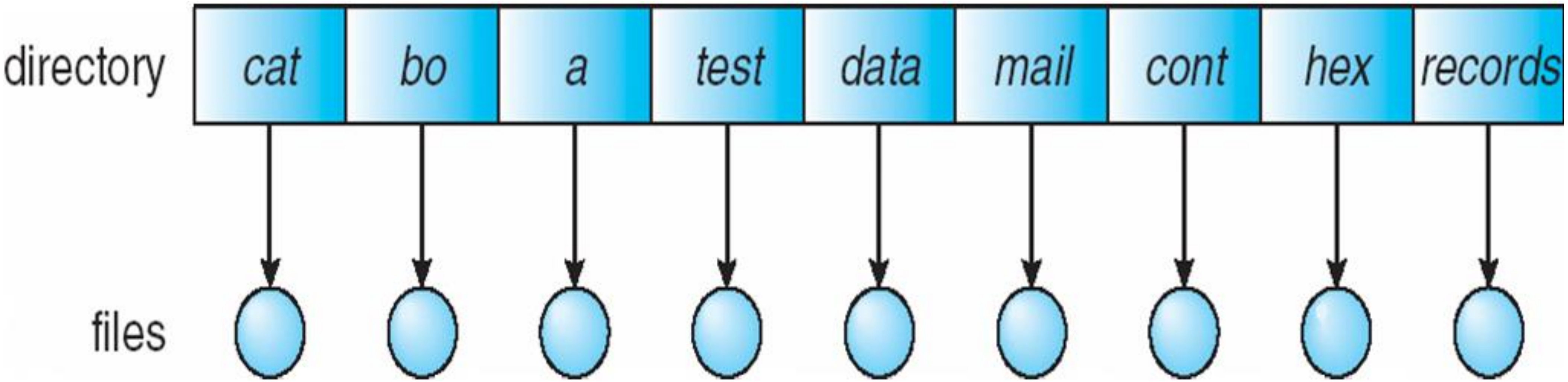
④ Search all files in all directories

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



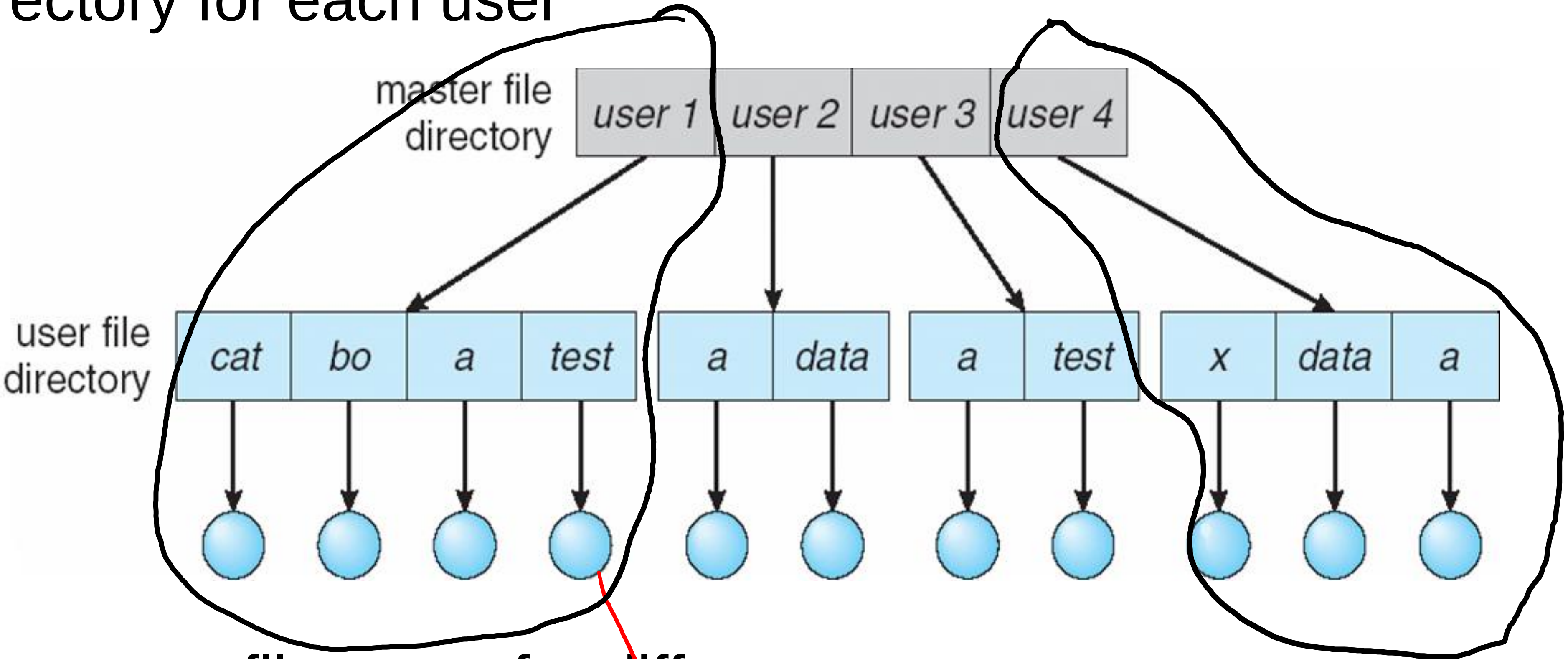
- Naming problem
- Grouping problem

Two-Level Directory

Drawbacks:

① User isolation

- Separate directory for each user

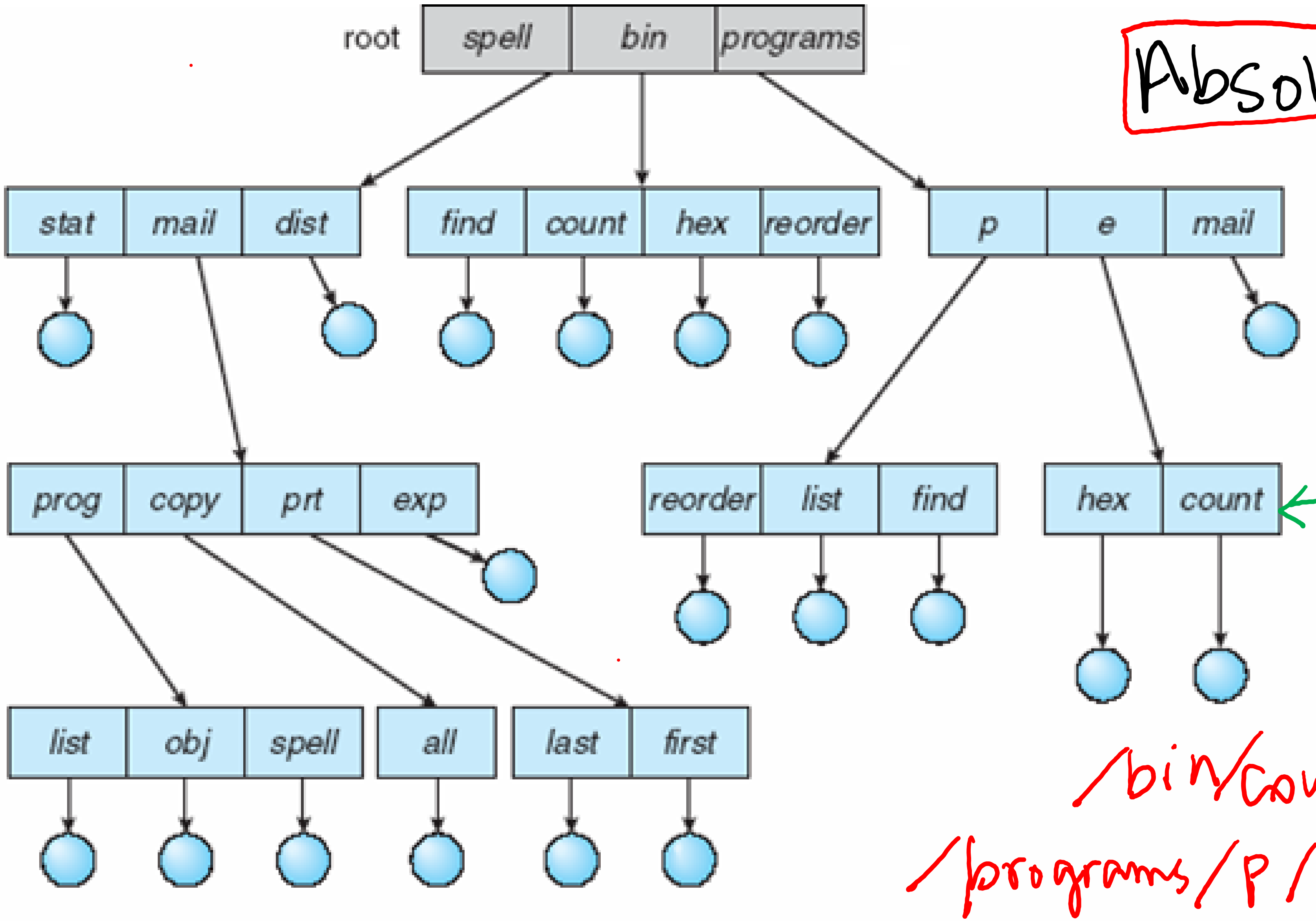


ls (executable)

- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories

n-level tree



root
spell bin programs
stat mail dist
find count hex reorder
p e mail
prog copy prt exp
reorder list find
hex count
list obj spell
all
last first

Absolute vs Relative path

../../..

/bin/count

/programs/p/list

Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - `cd /spell/mail/prog`
  - `type list`

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

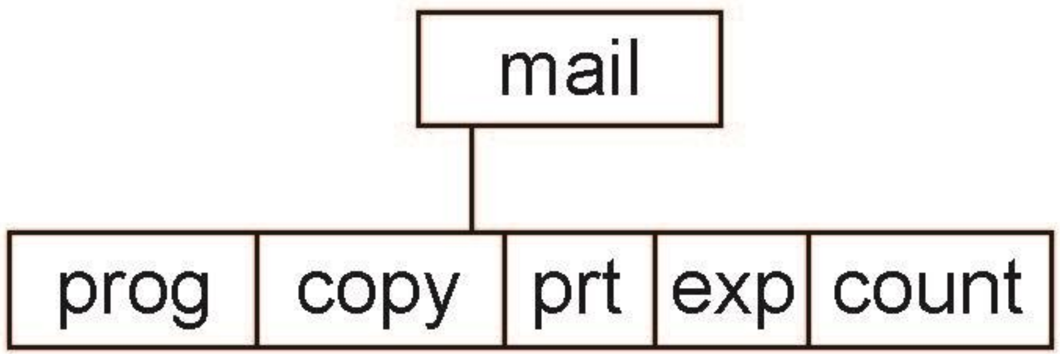
```
rm <file-name>
```

- 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 the entire subtree rooted by "mail"

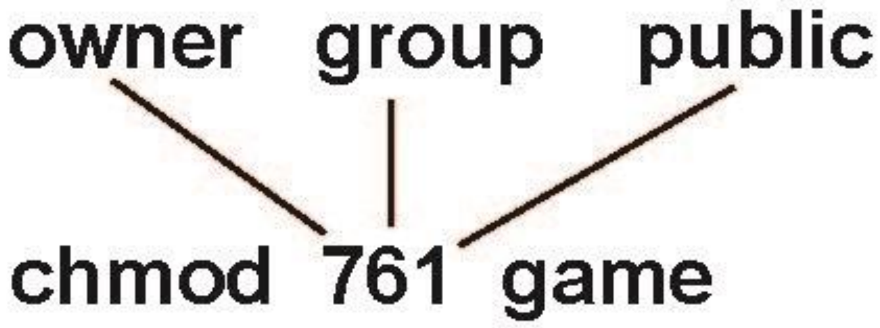
Protection

- 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

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



owner group public

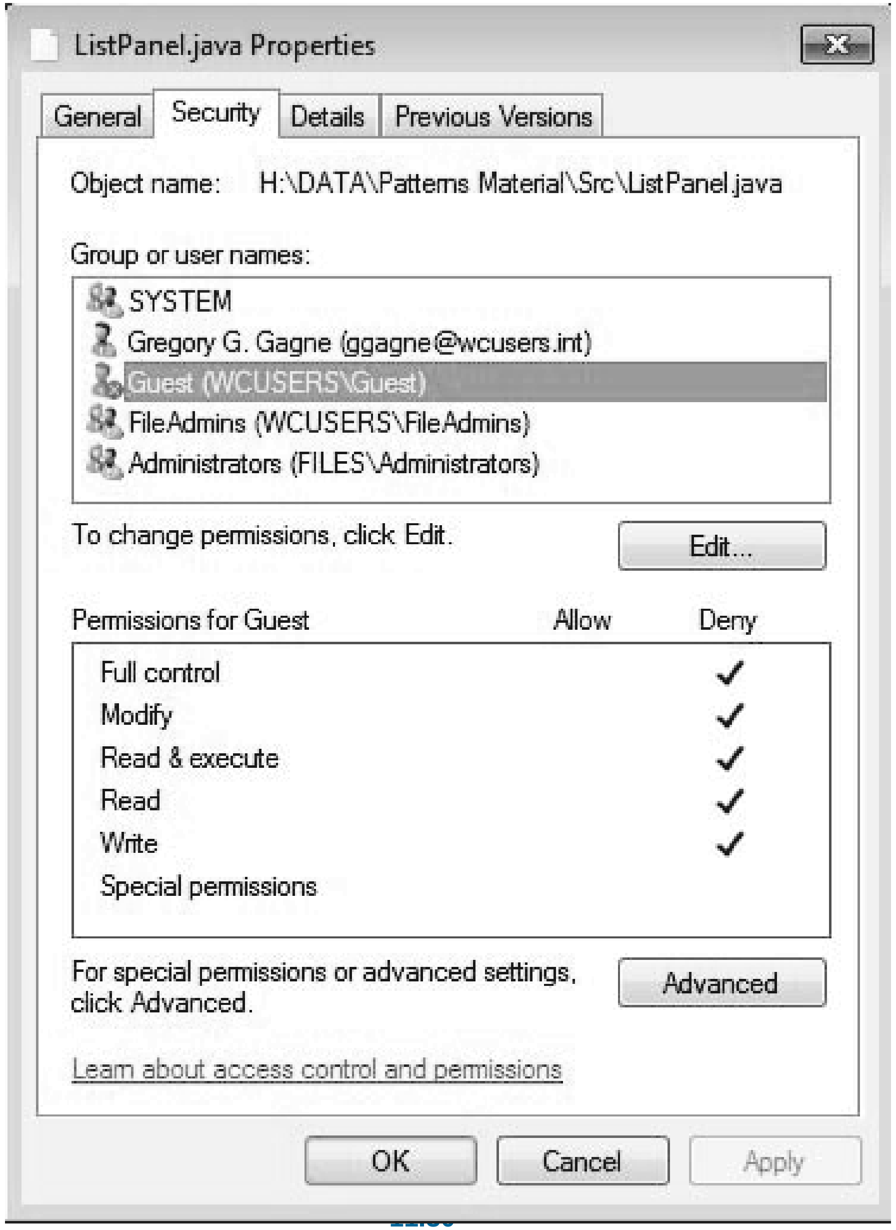
chmod 761 game

- 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)

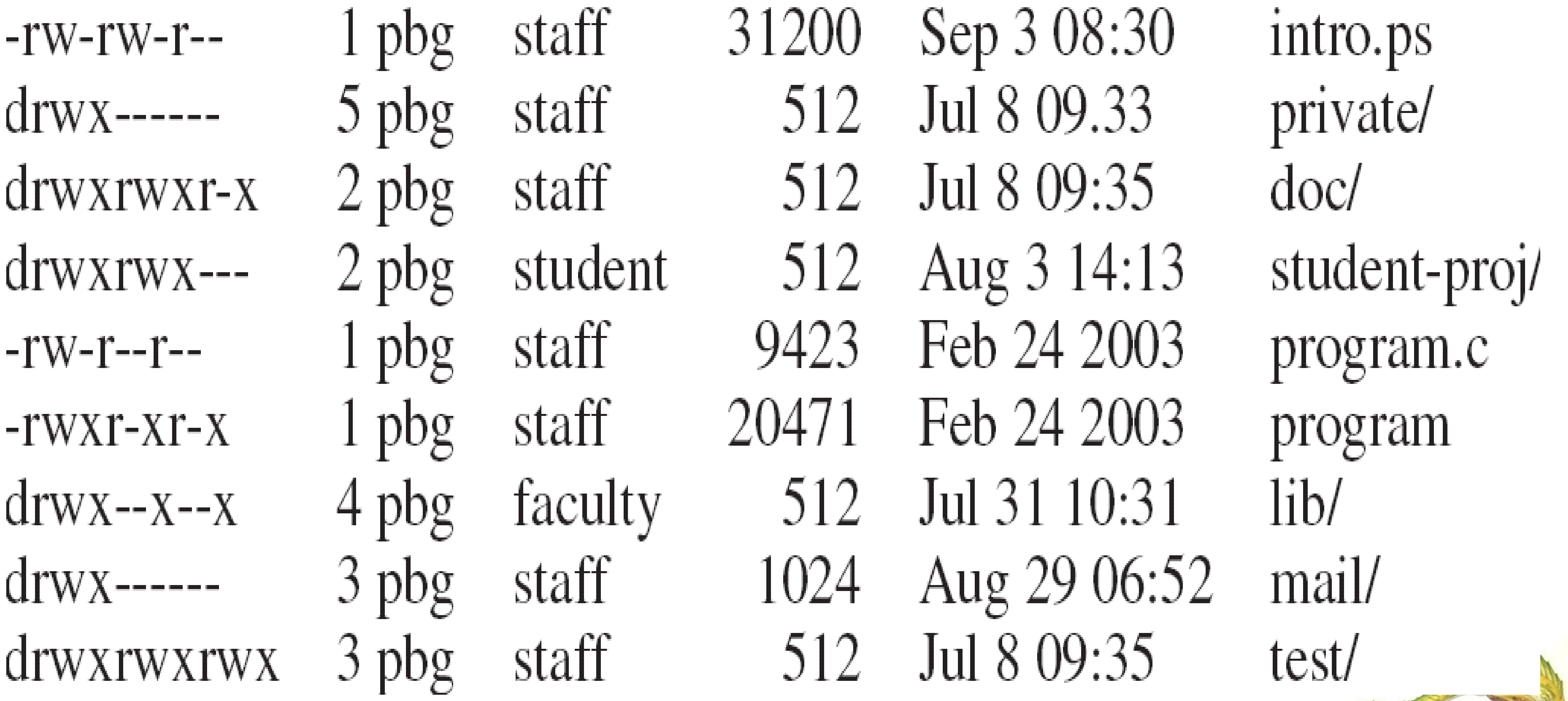
To change permissions, click Edit.

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

For special permissions or advanced settings, click Advanced.

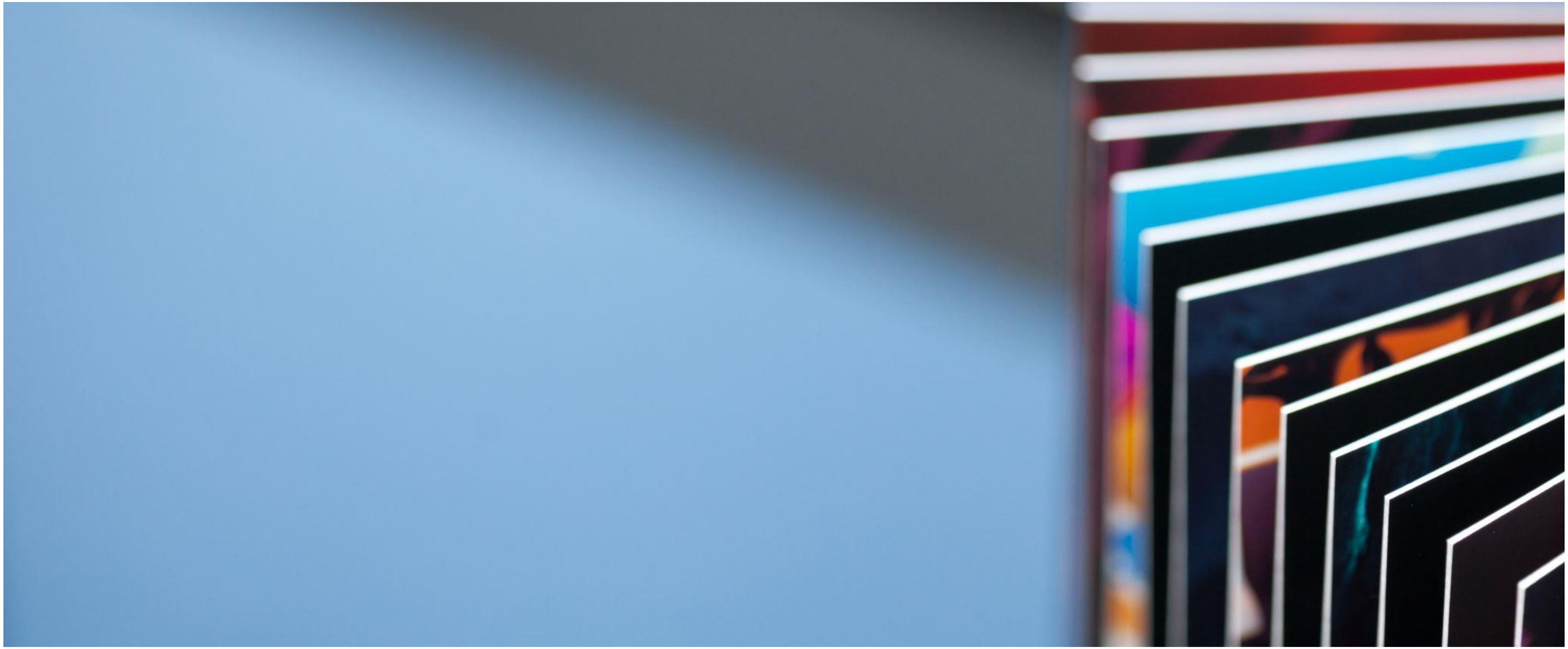
Learn about access control and permissions

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/
```

ADDITIONAL SLIDES

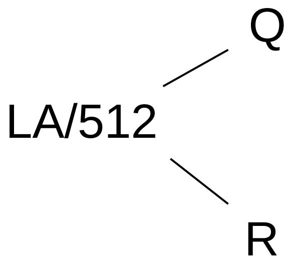


Allocation Methods - Contiguous

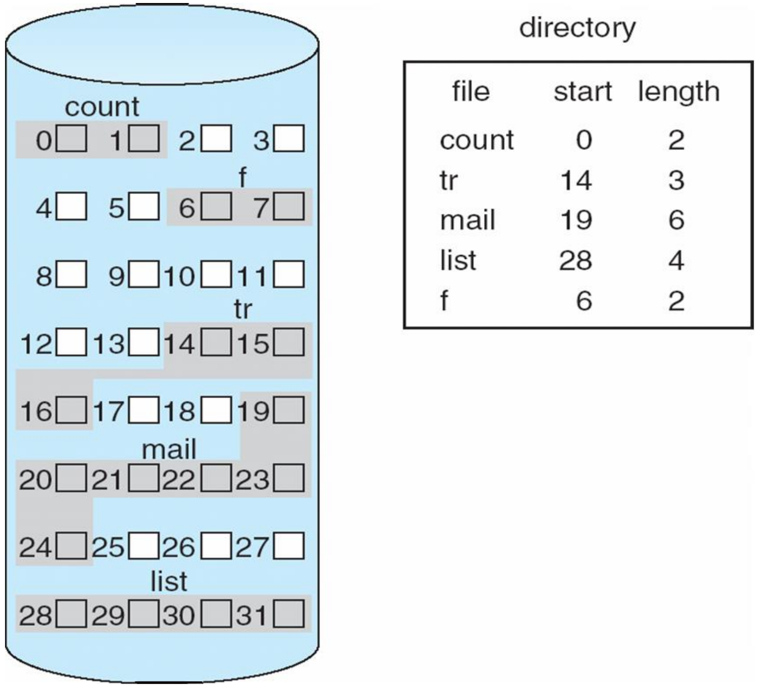
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding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

Contiguous Allocation

- Mapping from logical to physical



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



directory

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

Allocation Methods - Linked

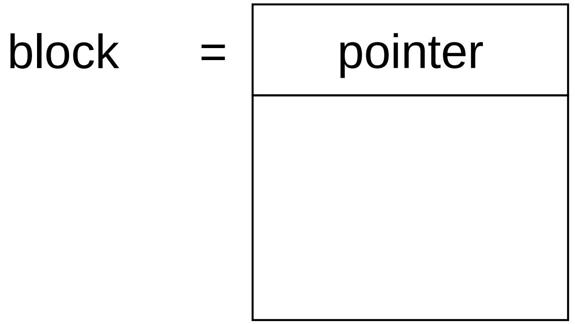
- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering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

Allocation Methods – Linked (Cont.)

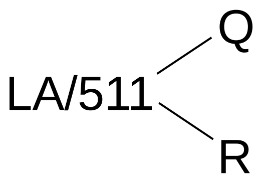
- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

Linked Allocation

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



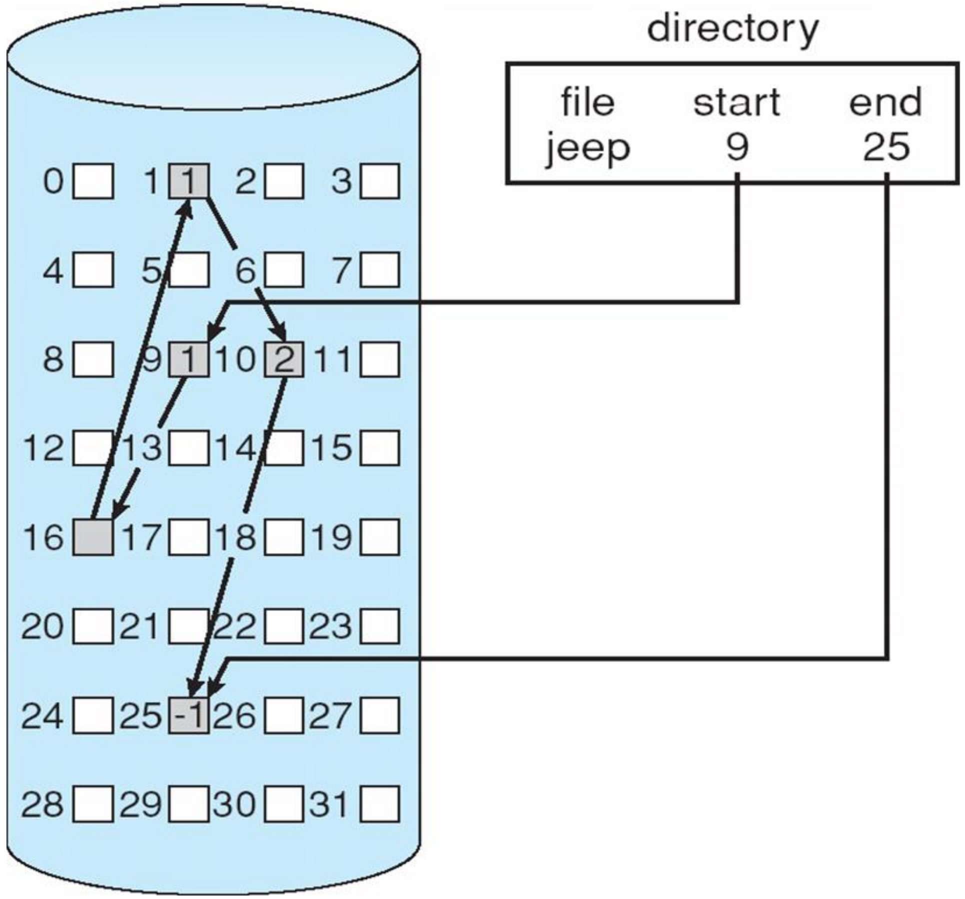
- Mapping



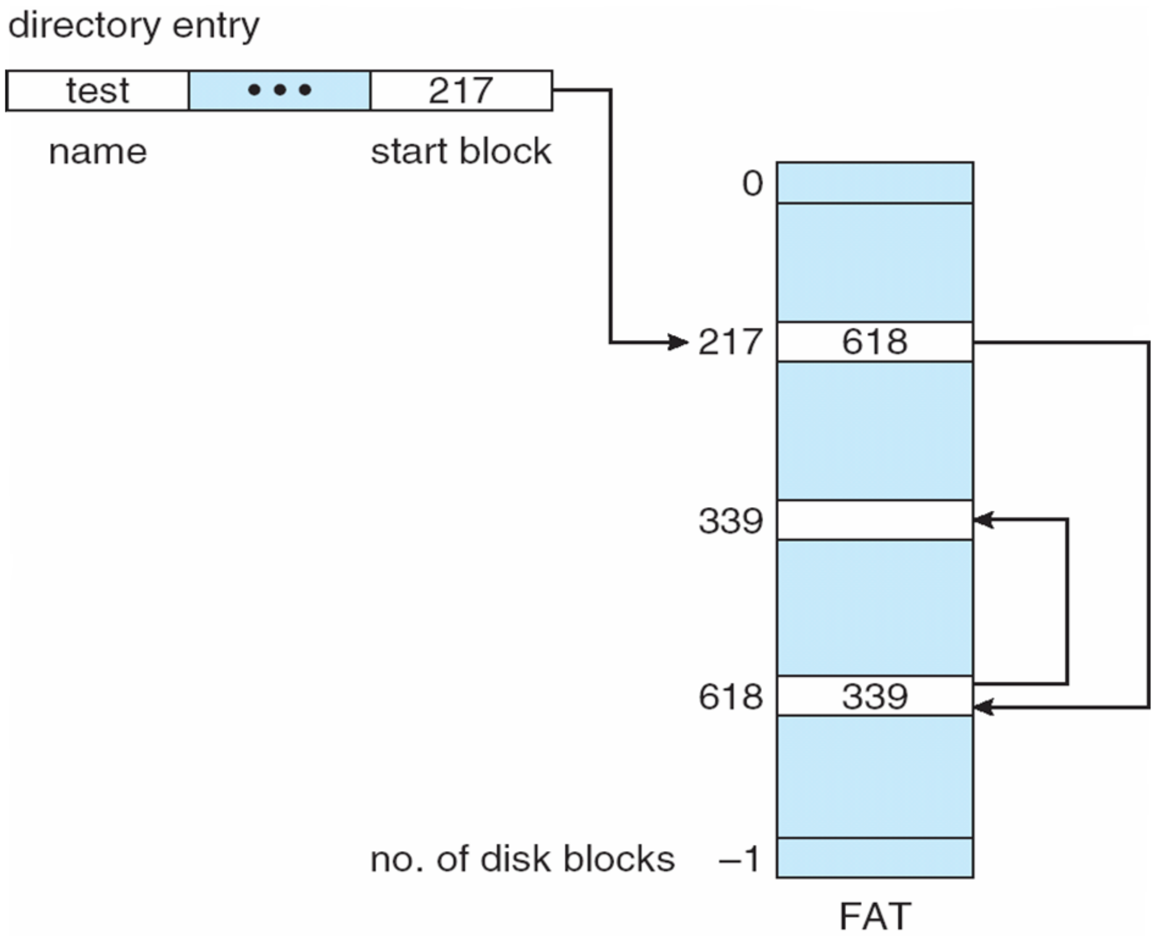
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

Linked Allocation



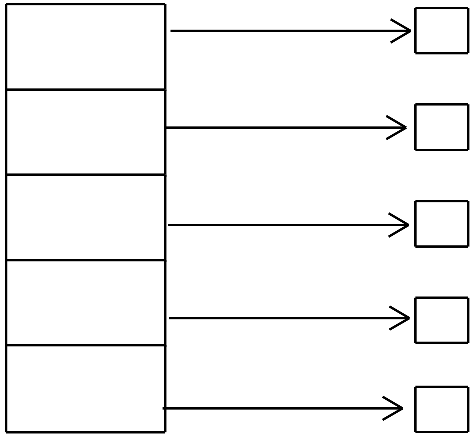
File-Allocation Table



Allocation Methods - Indexed

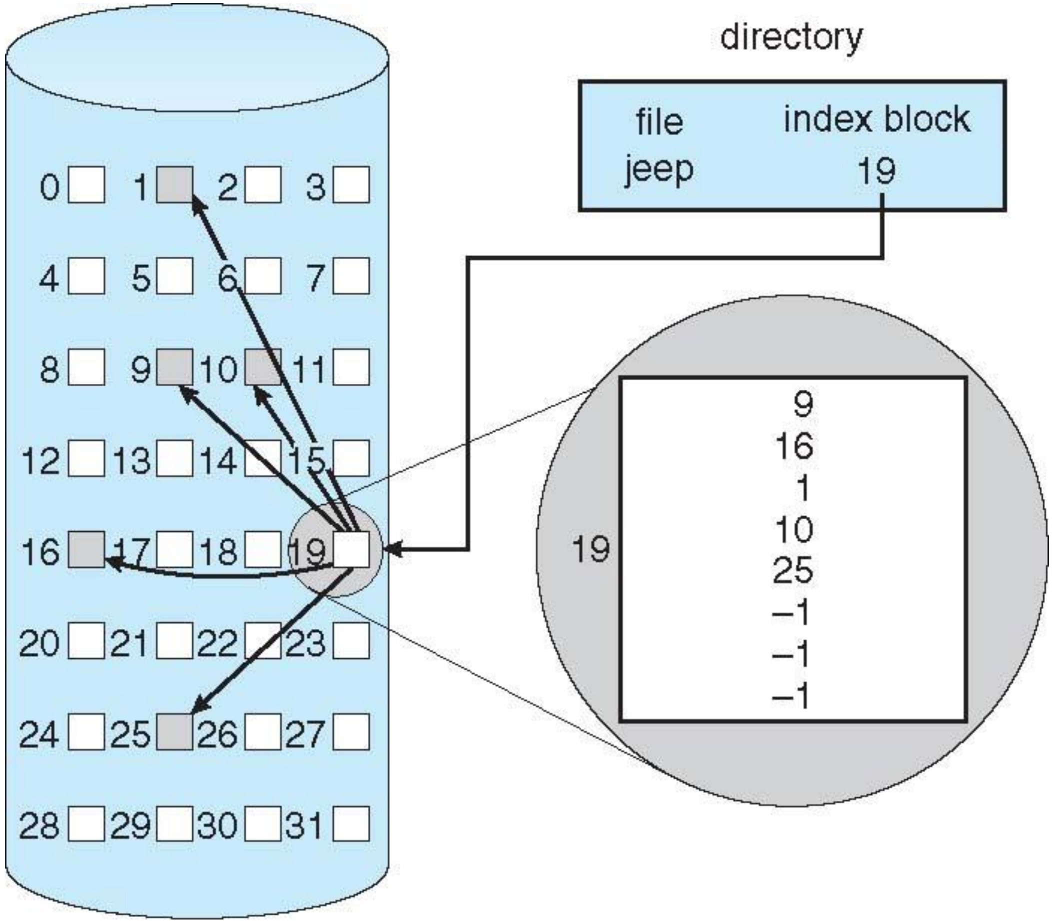
- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks

- Logical view



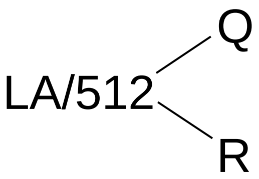
index table

Example of Indexed Allocation



Indexed Allocation (Cont.)

- 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

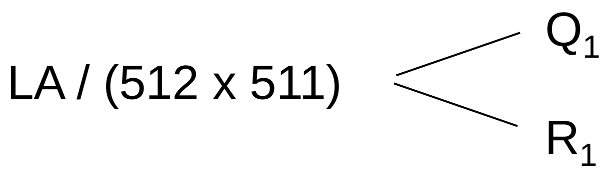


Q = displacement into index table
R = displacement into block

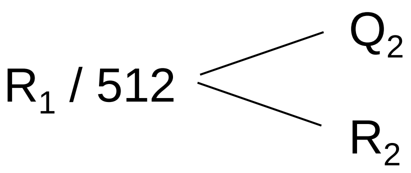
Indexed Allocation – Mapping (Cont.)

- 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)



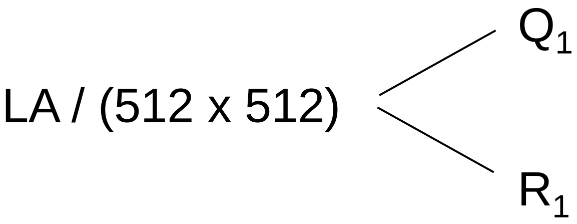
Q_1 = block of index table
R_1 is used as follows:



Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)

- 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)

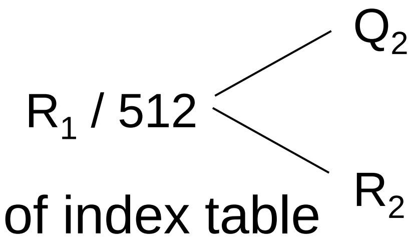


LA / (512 x 512)

Q_1

R_1

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



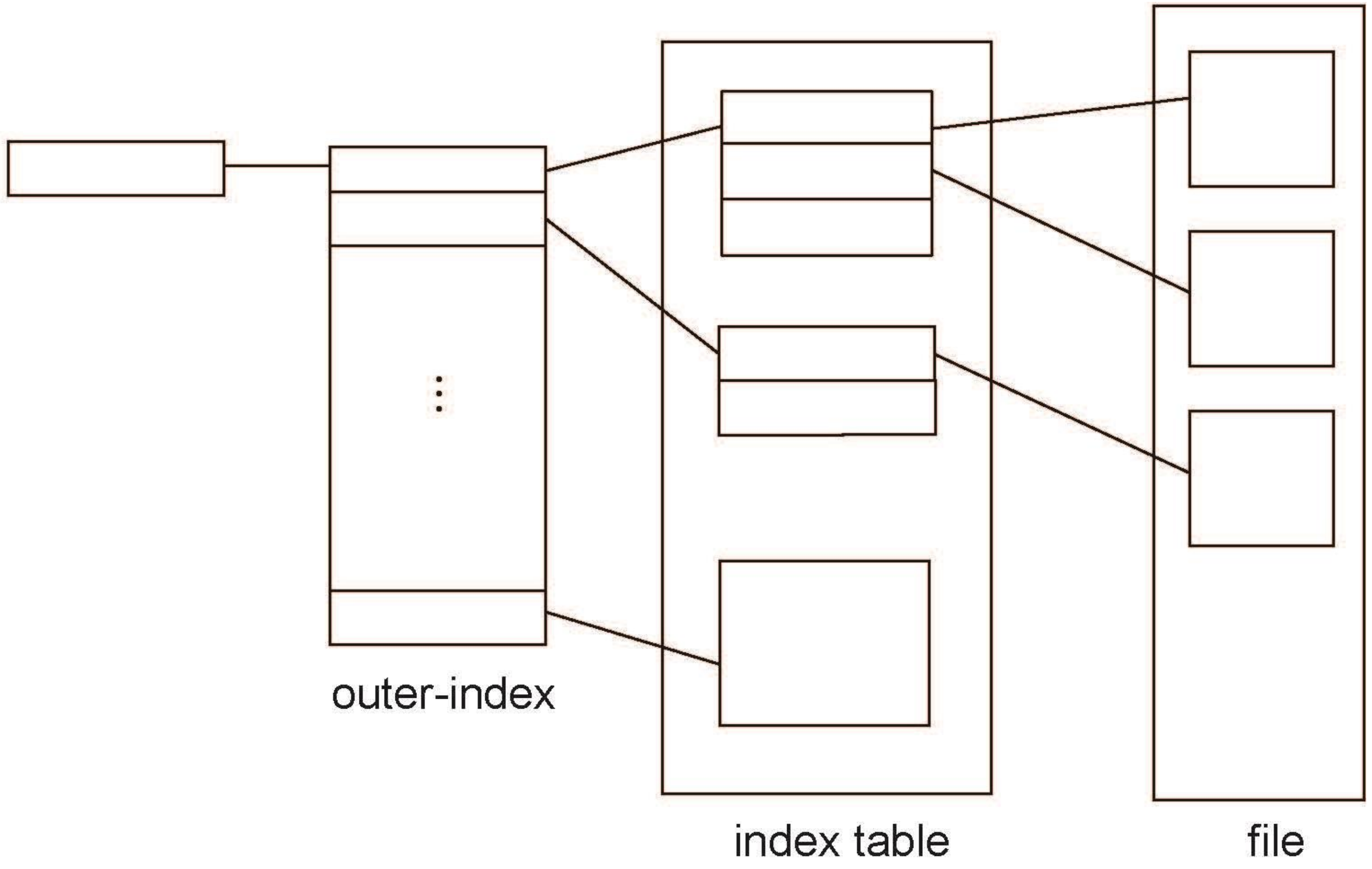
R_1 / 512

Q_2

R_2

Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)



End of Chapter 11